INFORMATIK-BIBER SCHWEIZ
CASTOR INFORMATIQUE SUISSE
CASTORO INFORMATICO SVIZZERA

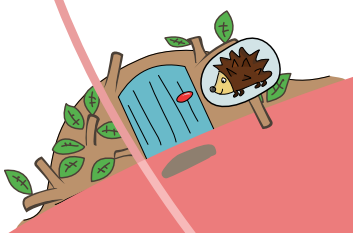
Quesiti e soluzioni 2025

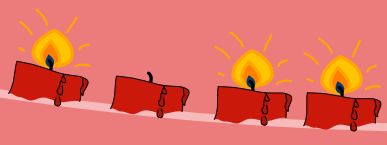
Tutte le Categorie

https://www.castoro-informatico.ch/

A cura di:

Susanne Thut, Nora A. Escherle, Masiar Babazadeh,
Christian Giang, Jean-Philippe Pellet





01010011010101100100100101000001001011010101001100101010010100100101000010100110010101010011010100011

SSII

www.svia-ssie-ssii.ch
schweizerischerverein für informatikinderausbildung//société suisse pour l'informatique dans l'enseignement//società svizzera per l'informatica nell'insegnamento

HASLERSTIFTUNG ETH zürich | ABZ AUSBILDUNGS- UND BERATUNGSZENTRUM FÜR INFORMATIKUNTERRICHT Scuola universitaria professionale della Svizzera italiana SUPSI hep/ PH LUZERN PÄDAGOGISCHE HOCHSCHULE

Hanno collaborato al Castoro Informatico 2025

Masiar Babazadeh, Jean-Philippe Pellet, Andrea Maria Schmid, Giovanni Serafini, Susanne Thut

Capo progetto: Nora A. Escherle

Un particolare ringraziamento per il lavoro sui quesiti del concorso Svizzero va a:
Patricia Heckendorn, Gymnasium Kirschgarten
Juraj Hromkovič, Regula Lacher: ETH Zürich, Ausbildungs- und Beratungszentrum für Informatikunterricht
Jens Hartmann, Stephan Koch, Dirk Schmerenbeck und Jacqueline Staub: Universität Tier, Germania

La scelta dei quesiti è stata svolta in collaborazione con gli organizzatori dei concorsi in Germania, Austria e Ungheria. Ringraziamo specialmente:
Philip Whittington, Silvan Horvath: ETH Zürich, Ausbildungs- und Beratungszentrum für Informatikunterricht
Wolfgang Pohl, Karsten Schulz, Franziska Kaltenberger, Margaretha Schlüter, Kirsten Schlüter, Michael Weigend: Bundesweite Informatikwettbewerbe (BWINF), Germania
Wilfried Baumann: Österreichische Computer Gesellschaft, Austria
Gerald Futschek, Lukas Lehner: Technische Universität Wien, Austria
Zsuzsa Pluhár, Bence Gaal: ELTE Informatikai Kar, Ungheria

La versione online del concorso è stata creata su cuttle.org. Ringraziamo per la buona collaborazione:
Eljakim Schrijvers, Justina Oostendorp, Alieke Stijf, Kyra Willekes: cuttle.org, Olanda
Andrew Csizmadia: Raspberry Pi Foundation, Regno Unito

Per il supporto durante le settimane del concorso ringraziamo:
Gabriel Thullen: Collège des Colombières, Versoix
Eveline Moor: Società svizzera per l'informatica nell'insegnamento

I compiti di programmazione sono stati creati e sviluppati appositamente per la piattaforma online. Desideriamo ringraziare le seguenti persone:
Jacqueline Staub: Universität Tier, Germania
Dirk Schmerenbeck: Universität Trier, Germania
Dave Oostendorp: cuttle.org, Olanda

Ringraziamo l'ETH per l'organizzazione e lo svolgimento della finale del Castoro:
Dennis Komm, Hans-Joachim Böckenhauer, Angélica Herrera Loyo, Andre Macejko, Moritz Stocker, Philip Whittington, Silvan Horvath: ETH Zürich, Ausbildungs- und Beratungszentrum für Informatikunterricht
Per la correzione dei compiti finali:
Clemens Bachmann, Morel Blaise, Tobias Boschung, Davud Evren, Jay Forrer, Sven Grübel, Urs Hauser, Fabian Heller, Jolanda Hofer, Alessandra Iacopino, Saskia Koller, Richard Královič, Jan Mantsch, Adeline Pittet, Alexander Skodinis, Emanuel Skodinis, Jasmin Sudar, Valerie Verdan, Chris Wernke

Per la traduzione dei compiti finali in francese:
Jean-Philippe Pellet: Haute école pédagogique du canton de Vaud

Christoph Frei: Chragokyberneticks (Logo Informatik-Biber Schweiz)
Andrea Leu, Maggie Winter: Senarclens Leu + Partner AG

Un ringraziamento speciale va ai nostri grandi sponsor Juraj Hromkovič, Dennis Komm, Gabriel Parriaux e la Fondazione Hasler. Senza di loro, questo concorso non esisterebbe.

L'edizione dei quesiti in lingua tedesca è stata utilizzata anche in Germania e in Austria.

La traduzione francese è stata curata da Elsa Pellet mentre quella italiana da Christian Giang.

INFORMATIK-BIBER SCHWEIZ
CASTOR INFORMATIQUE SUISSE
CASTORO INFORMATICO SVIZZERA

Il Castoro Informatico 2025 è stato organizzato dalla Società Svizzera per l'Informatica nell'Insegnamento (SSII) e sostenuto in modo significativo e generoso dalla Fondazione Hasler. Altri partner e sponsor che hanno sostenuto finanziariamente il concorso sono Abraxas Informatik AG, l'Ufficio per la scuola materna, elementare e la consulenza (AKVB) del Cantone di Berna, l'Ufficio per l'economia AWI del Cantone di Zurigo, CYON AG e UBS.

Questo quaderno è stato creato il 10 dicembre 2025 con il sistema per la preparazione di testi LaTeX. Ringraziamo Christian Datzko per lo sviluppo del sistema di generazione dei testi che ha permesso di generare le 36 versioni di questa brochure (divise per lingua e livello scolastico). Il sistema è stato riprogrammato basandosi sul sistema precedente, sviluppato nel 2014 assieme a Ivo Blöchliger. Ringraziamo Jean-Philippe Pellet per lo sviluppo del sistema `bebras`, utilizzato dal 2020 per la conversione dei documenti sorgente dai formati Markdown e YAML.

Nota: Tutti i link sono stati verificati l'01.12.2025.

Premessa

Il concorso del «Castoro Informatico», presente già da diversi anni in molti paesi europei, ha l'obiettivo di destare l'interesse per l'informatica nei bambini e nei ragazzi. In Svizzera il concorso è organizzato in tedesco, francese e italiano dalla Società Svizzera per l'Informatica nell'Insegnamento (SSII), con il sostegno della fondazione Hasler.

Il Castoro Informatico è il partner svizzero del Concorso «Bebras International Contest on Informatics and Computer Fluency» (`https://www.bebras.org/`), situato in Lituania.

Il concorso si è tenuto per la prima volta in Svizzera nel 2010. Nel 2012 l'offerta è stata ampliata con la categoria del «Piccolo Castoro» (3º e 4º anno scolastico).

Il Castoro Informatico incoraggia gli alunni ad approfondire la conoscenza dell'informatica: esso vuole destare interesse per la materia e contribuire a eliminare le paure che sorgono nei suoi confronti. Il concorso non richiede alcuna conoscenza informatica pregressa, se non la capacità di «navigare» in internet poiché viene svolto online. Per rispondere alle domande sono necessari sia un pensiero logico e strutturato che la fantasia. I quesiti sono pensati in modo da incoraggiare l'utilizzo dell'informatica anche al di fuori del concorso.

Nel 2025 il Castoro Informatico della Svizzera è stato proposto a cinque differenti categorie d'età, suddivise in base all'anno scolastico:

- 3º e 4º anno scolastico
- 5º e 6º anno scolastico
- 7º e 8º anno scolastico
- 9º e 10º anno scolastico
- 11º al 13º anno scolastico

Ogni categoria aveva quesiti classificati in tre livelli di difficoltà: facile, medio e difficile. Alla categoria del 3º e 4º anno scolastico sono stati assegnati 9 quesiti da risolvere, di cui 3 facili, 3 medi e 3 difficili. Alla categoria del 5º e 6º anno scolastico sono stati assegnati 12 quesiti, suddivisi in 4 facili, 4 medi e 4 difficili. Ogni altra categoria ha ricevuto invece 15 quesiti da risolvere, di cui 5 facili, 5 medi e 5 difficili.

Per ogni risposta corretta sono stati assegnati dei punti, mentre per ogni risposta sbagliata sono stati detratti. In caso di mancata risposta il punteggio è rimasto inalterato. Il numero di punti assegnati o detratti dipende dal grado di difficoltà del quesito:

	Facile	**Medio**	**Difficile**
Risposta corretta	6 punti	9 punti	12 punti
Risposta sbagliata	−2 punti	−3 punti	−4 punti

Il sistema internazionale utilizzato per l'assegnazione dei punti limita l'eventualità che il partecipante possa ottenere buoni risultati scegliendo le risposte in modo casuale.

Ogni partecipante inizia con un punteggio pari a 45 punti (risp., 3º e 4º anno scolastico: 27 punti, 5º e 6º anno scolastico: 36 punti).

Il punteggio massimo totalizzabile era dunque pari a 180 punti (risp., 3º e 4º anno scolastico: 108 punti, 5º e 6º anno scolastico: 144 punti), mentre quello minimo era di 0 punti.

In molti quesiti le risposte possibili sono state distribuite sullo schermo con una sequenza casuale. Lo stesso quesito è stato proposto in più categorie d'età. Questi quesiti presentavano livelli di difficoltà diversi nei vari gruppi di età.

Alcuni quesiti sono indicati come «bonus» per determinate categorie di età: non contano nel totale dei punti, ma vengono utilizzati come spareggio per punteggi identici in caso di qualificazione agli eventuali turni successivi.

Per ulteriori informazioni:

Società Svizzera per l'Informatica nell'Insegnamento
SVIA-SSIE-SSII
Castoro Informatico
Masiar Babazadeh

`https://www.castoro-informatico.ch/kontaktieren/`
`https://www.castoro-informatico.ch/`

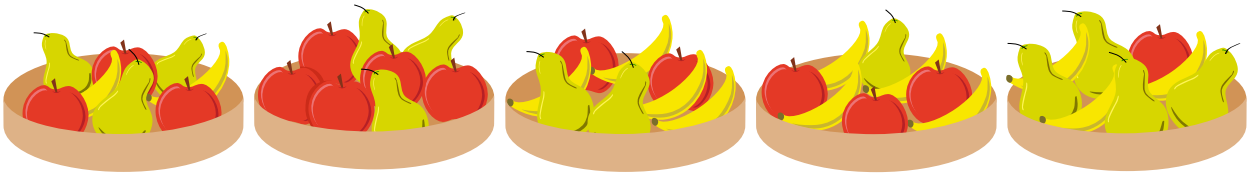
Indice

1. Cesti di frutta

A Bella piace la frutta. La sua preferita sono le mele  , poi le banane  e meno di tutte le pere  .

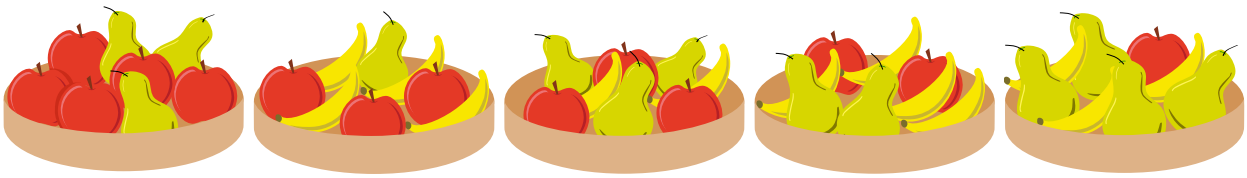
Bella ha cinque cesti di frutta. Vuole ordinare questi cesti secondo le sue preferenze. Più mele ci sono in un cesto, più a sinistra dovrebbe trovarsi. Se due cesti hanno lo stesso numero di mele, il cesto con più banane deve essere più a sinistra.

Ordina i cesti per Bella.

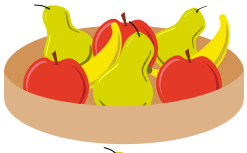
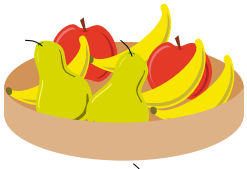
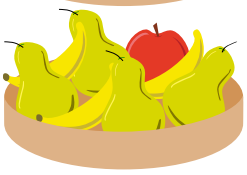


Soluzione

La risposta corretta è la seguente:



Contiamo il numero di mele, banane e pere in ogni cesto:

Cestino	Immagine	Numero di mele	Numero di banane	Numero di pere
A		**3**	**2**	3
B		**5**	0	3
C		**2**	4	2
D		**3**	**3**	1
E		**1**	3	4

Il cesto B contiene il maggior numero di mele, 5, e deve trovarsi all'estrema sinistra. Seguono i cesti A e D, ciascuno con 3 mele. Poiché il cesto D ha più banane, è il prossimo in fila, seguito dal cesto A. Questo perché Bella preferisce le banane alle pere. Dei due cesti rimanenti, il cesto C con 2 mele precede il cesto E con una sola mela. L'ordine finale è quindi: B, D, A, C, E.

Questa è l'informatica!

In questo compito, Bella vuole *ordinare* i cesti di frutta secondo le sue preferenze, cioè metterli in un ordine specifico. Siccome le mele sono il suo frutto preferito, ha senso che il cesto con il maggior numero di mele sia il più in alto nella sua classifica del cesto di frutta preferito.

È possibile ordinare un insieme di elementi solo se si può sempre decidere, per ogni coppia di elementi, quale dei due venga prima. Questo richiede una caratteristica comune a tutti gli elementi, in base

alla quale possano essere differenziati. Inoltre, deve esistere un ordine chiaro per i valori che questa caratteristica può assumere. Ad esempio, ogni libro ha almeno un autore e possiamo usare il nome del primo autore come caratteristica di ordinamento. I nomi seguono un ordine alfabetico chiaro: «Meier» viene prima di «Meyer» e «Schach» prima di «Schacht». In questo modo, possiamo ordinare l'intera collezione di libri in base all'autore.

L'ordinamento è un'attività fondamentale per l'informatica. I computer eseguono costantemente operazioni di ordinamento, soprattutto perché i dati ordinati permettono di effettuare ricerche in modo molto più rapido ed efficiente. L'informatica dispone di numerosi metodi, chiamati tecnicamente algoritmi, che possono essere utilizzati per ordinare i dati. Anche per gli specialisti, tuttavia, non è sempre semplice decidere quale di questi algoritmi sia il più adatto per uno specifico compito di ordinamento.

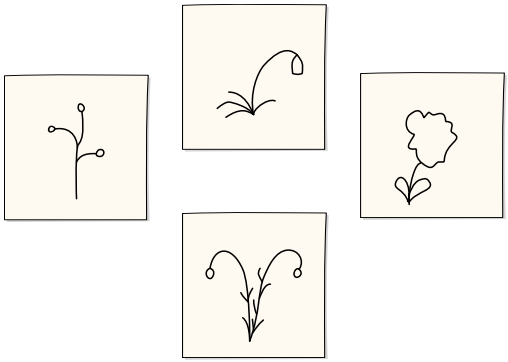
Parole chiave e siti web

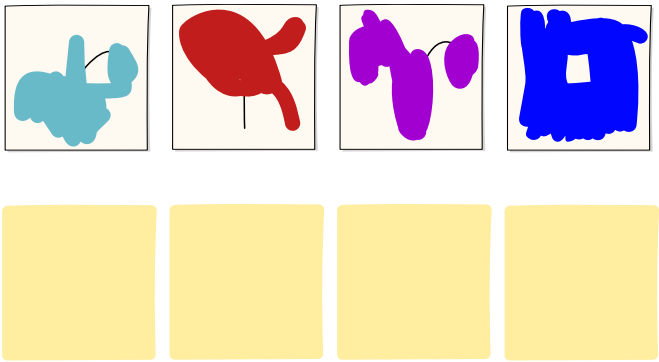
- Diversi esercizi: `https://pikas-mi.dzlm.de/inhalte/formen-erkunden/unterricht/aufgabenstellung-kompakt-„formen-sortieren“`
- Algoritmo di ordinamento: `https://it.wikipedia.org/wiki/Algoritmo_di_ordinamento`

2. Colori a dita

Lars ha disegnato dei fiori. La sorellina Carlotta trova i disegni e li dipinge con i colori a dita.

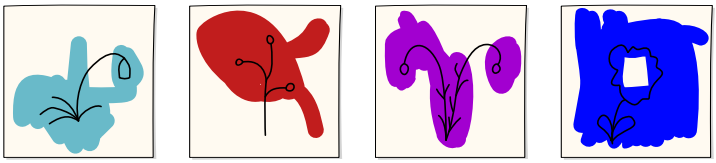
Che aspetto avevano i disegni prima di essere dipinti?





Soluzione

Questa è la risposta: qui si può vedere cosa si nasconde dietro i colori nei disegni dipinti (figure da A a D).



A B C D

Per trovare la risposta giusta dobbiamo confrontare:

1. le linee nere di ciascun disegno dipinto con le linee di tutti i disegni non dipinti e
2. le aree bianche di ciascun disegno dipinto con le aree bianche di tutti i disegni non dipinti.

Nel farlo, ci rendiamo conto che

- C'è solo un disegno non dipinto con un'area bianca nell'angolo in alto a sinistra, come nell'immagine A.
- Esiste solo un disegno non dipinto con uno stelo nella metà inferiore, come nell'immagine B.
- C'è solo un disegno non dipinto con un'area bianca al centro, come nell'immagine D.
- Questo lascia esattamente un disegno non dipinto per l'immagine C.

Questa è l'informatica!

La piccola Carlotta ha dipinto i disegni del fratello con i colori a dita. Pur facendo del suo meglio, in seguito risulta difficile riconoscere l'aspetto dei disegni non dipinti. Quando vengono restaurate le opere d'arte nei musei, gli esperti cercano di riportare le opere, sovradipinte o danneggiate, al loro stato originale. Nel farlo, spesso devono trovare riferimenti adeguati da altre fonti per ricreare accuratamente l'opera originale. Questo processo è simile a quello che avete appena fatto in questo compito.

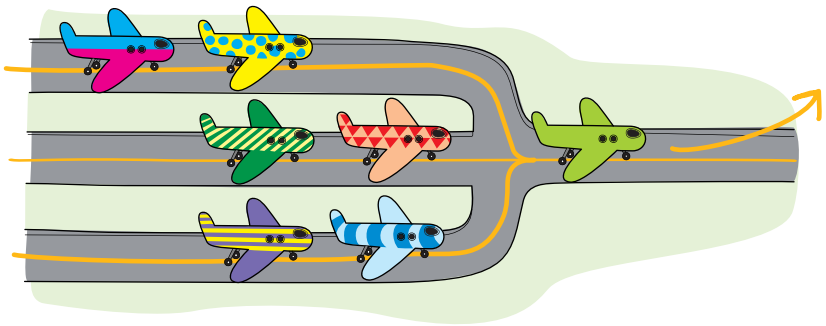
In informatica, le tecniche di restauro sono utilizzate in sistemi che generano immagini completamente nuove. Questi sistemi avanzati sono chiamati *modelli di diffusione*. Un modello di diffusione viene addestrato su numerose immagini per imparare a ripristinare le immagini digitali danneggiate. Grazie a questa conoscenza, il sistema è in grado di creare nuove immagini da immagini composte da molti piccoli punti sparsi. C'è tuttavia una differenza sostanziale nel vostro approccio a questo compito. Voi avete risolto il compito con il pensiero logico, mentre il modello di diffusione non lavora con logica: al contrario, acquisisce le sue capacità di ripristino analizzando molti esempi tramite l'apprendimento automatico.

Parole chiave e siti web

- *Modello di diffusione*: `https://www.ibm.com/it-it/think/topics/diffusion-models`
- *Apprendimento automatico*: `https://it.wikipedia.org/wiki/Apprendimento_automatico`

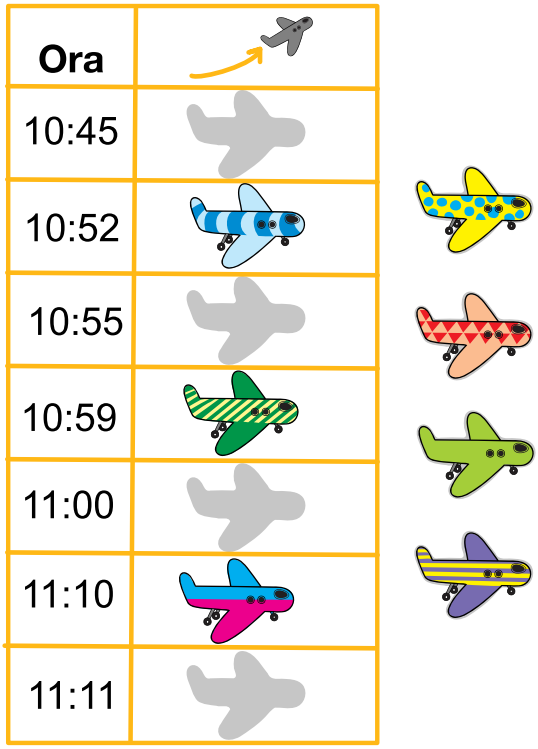
3. Aerei

Sette aerei devono decollare questa mattina. Decollano tutti sulla stessa pista a destra. Gli aerei viaggiano solo in avanti seguendo le linee, non possono tornare indietro e non possono sorpassarsi a vicenda.



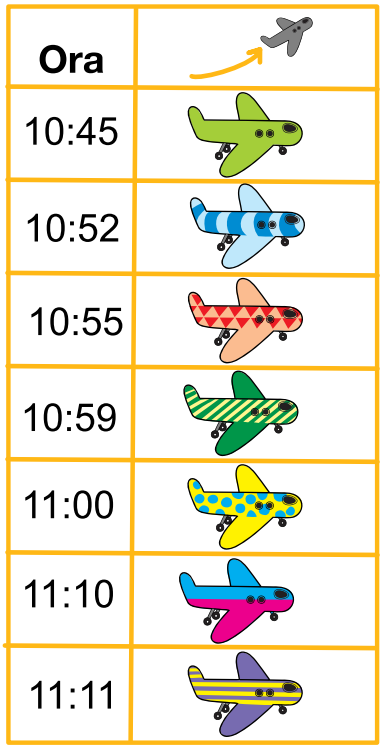
L'orario indica l'ordine di decollo dei sette aerei. Purtroppo alcuni aerei non sono ancora segnati. C'è solo un ordine possibile di partenza.

Aggiungi gli aerei mancanti all'orario.



Soluzione

Questa è la soluzione:



1. Prima (alle 10:45) è l'aereo  a decollare: è già sulla pista!
2. Il secondo (alle 10:52) è l'aereo  .
3. Il terzo aereo a decollare (alle 10:55) è  : si trova davanti all'aereo  . Affinché quest'ultimo aereo decolli successivamente, come indicato nell'orario, l'aereo  deve essere già decollato (altrimenti gli sarebbe davanti!).
4. Il quarto aereo a decollare (alle 10:59) è  .
5. Il quinto aereo a decollare (alle 11:00) è  : si trova davanti all'aereo  . Affinché quest'ultimo aereo decolli successivamente, come indicato nell'orario, l'aereo  deve essere già decollato, come per il punto 3.
6. Il sesto (alle 11:10) a decollare è  .
7. Il settimo e ultimo aereo a decollare (alle 11:11) è  .

Questa è l'informatica!

La sequenza di decollo dell'aereo dipende anche dalla sequenza di attesa. Alcuni aerei possono decollare solo quando altri in attesa davanti a loro sono già decollati. In un aeroporto vero, per pianificare la sequenza di decollo è necessario tenere conto di molte altre condizioni, come ritardi, problemi tecnici o cambiamenti meteorologici. Per la pianificazione si utilizzano solitamente programmi informatici, che possono reagire rapidamente ai cambiamenti delle condizioni e produrre comunque buoni piani.

La creazione di programmi, come il programma di decollo di questo compito, rientra nell'area dello *scheduling* in informatica. La programmazione può diventare complicata, ad esempio, se è disponibile una sola risorsa (come la pista di decollo), se alcuni eventi sono interdipendenti o se è necessario rispettare una sequenza per evitare problemi.

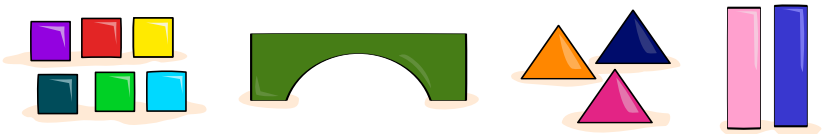
In informatica, la programmazione è considerata un problema particolarmente difficile. La sfida consiste nel calcolare un piano ottimale per ogni situazione immaginabile. Se si devono prendere in considerazione molti parametri e condizioni, anche un computer molto veloce può impiegare una quantità enorme di tempo. In questi casi, in informatica si dice che la programmazione è *NP-difficile*. Ciò significa che, sebbene sia facile verificare se un determinato piano è corretto, è estremamente difficile trovare il piano migliore. È come un puzzle particolarmente difficile: verificare la soluzione è facile, ma trovarla può richiedere molto tempo.

Parole chiave e siti web

- Scheduling: `https://de.wikipedia.org/wiki/Scheduling`
- NP-difficile: `https://it.wikipedia.org/wiki/NP-difficile`
- NP-completo: `https://it.wikipedia.org/wiki/NP-completo`

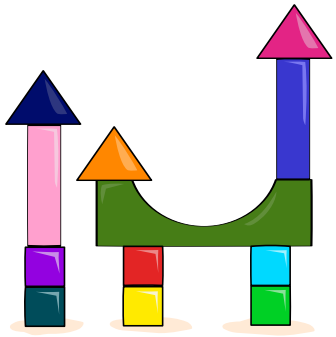
4. I mattoncini

Ali ha questi mattoncini:



6 cubi 1 ponte 3 piramidi 2 cuboidi

Zaha, la sorella di Ali, gli dà gradualmente istruzioni su cosa fare con i mattoncini. Ali esegue subito ogni istruzione che riceve. Alla fine, viene creata questa costruzione:



In che ordine Zaha ha dato le istruzioni?

Die einzelnen Anweisungen müssen in beliebiger Reihenfolge angeordnet werden können. Es gibt diese fünf Anweisungen:

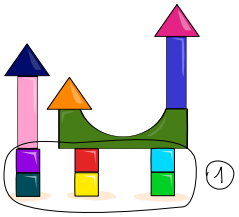
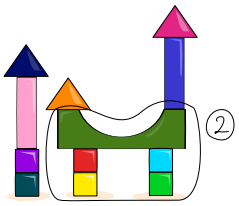
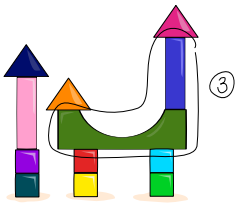
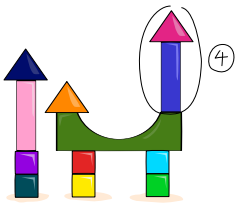
- Posizionare entrambi i cuboidi sulla strutura.
- Posizionare le torri appena costruite in fila.
- Posizionare le piramidi sulla strutura.
- Costruire tre torri con 2 cubi ciascuna.
- Posizionare il ponte sulla struttura.

Soluzione

Zaha ha dato le istruzioni in questo ordine:

1. Costruire tre torri con 2 cubi ciascuna.
2. Posizionare le torri appena costruite in fila.
3. Posizionare il ponte sulla struttura.
4. Posizionare entrambi i cuboidi sulla strutura.
5. Posizionare le piramidi sulla strutura.

Spieghiamo ora perché è proprio questa la sequenza.

Immagine	Passo
	Siccome le tre torri di cubi sono gli unici pezzi che poggiano direttamente sul terreno, devono essere costruite per prime (prima istruzione) e poi disposte in fila (seconda istruzione).
	Il ponte deve essere posizionato sopra la struttura precedente (terza istruzione). Deve essere posizionato sopra le torri di cubi, ma sotto i cuboidi e le piramidi.
	Poi si devono posizionare i cuboidi (quarta istruzione). I cuboidi si trovano infatti direttamente sul ponte, ma sotto le piramidi.
	Infine, si posizionano le piramidi (quinta istruzione). Le piramidi si trovano sui cuboidi e non c'è nessun altro blocco da aggiungere sopra le piramidi.

Siccome la maggior parte delle istruzioni indica che i blocchi devono essere posizionati sopra la costruzione (precedente), l'ordine delle istruzioni è direttamente correlato a quali mattoncini sono sopra o sotto quali altri mattoncini.

Non è possibile costruire la struttura con una sequenza di istruzioni diversa.

Questa è l'informatica!

Se un certo numero di mattoncini viene collocato singolarmente uno accanto all'altro sul pavimento, non è possibile dire a posteriori in quale ordine i mattoncini sono stati collocati sul pavimento. Le azioni, cioè posare un blocco sul pavimento, sono indipendenti l'una dall'altra. Quando i blocchi vengono messi uno sopra l'altro, vengono posizionati solo in sequenza, con il mattoncino più basso usato per primo e quello più alto per ultimo. Il posizionamento del mattoncino più basso è un prerequisito per posizionare il mattoncino successivo sopra di esso.

I programmi informatici sono costituiti da una sequenza di singole istruzioni, come quelle di Zaha in questo compito. Le singole istruzioni possono essere semplici e avere effetti altrettanto semplici, ma possono anche essere molto complicate. Quando programmano, è importante che gli informatici riflettano molto attentamente sugli effetti delle singole istruzioni e in quale ordine esse vengono eseguite. Se riflettete bene prima di programmare non avrete difficoltà a definire istruzioni nell'ordine giusto per ottenere il risultato auspicato, proprio come avete fatto in questo compito.

Parole chiave e siti web

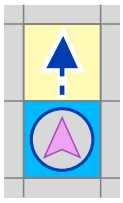
- *Istruzione*: `https://it.wikipedia.org/wiki/Istruzione_(informatica)`

5. Lefty I

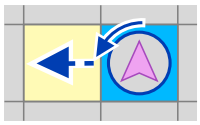
Il robot *Lefty* si muove su una griglia composta da caselle quadrate. Tra le caselle possono esserci dei muri rossi. Lefty deve raggiungere l'obiettivo verde.

Lefty può muoversi solo in due modi:

Avanzare di una casella

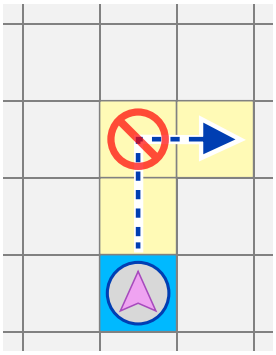


Girare a sinistra e avanzare immediatamente di una casella

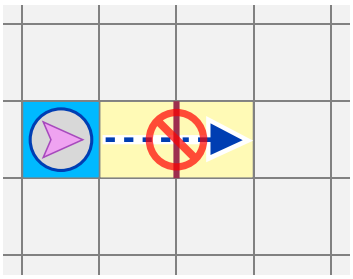


Quindi ci sono azioni che Lefty non può fare:

. . . **non** può girare a destra e. . .

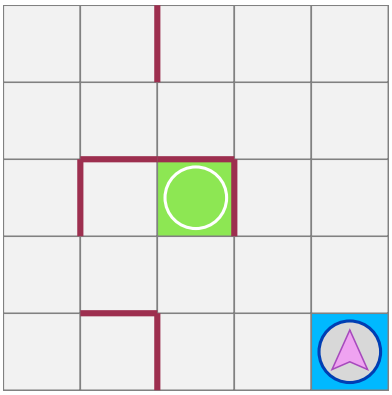


non può passare attraverso i muri.



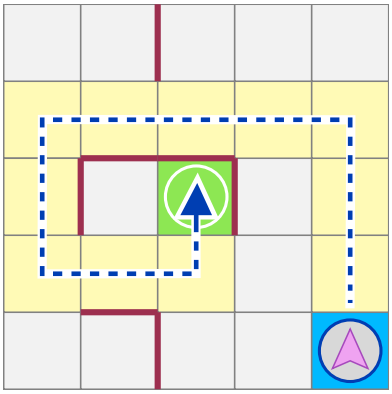
Quali caselle deve attraversare Lefty per raggiungere la destinazione?

Selezionare ***il minor numero possibile di caselle****.*



Soluzione

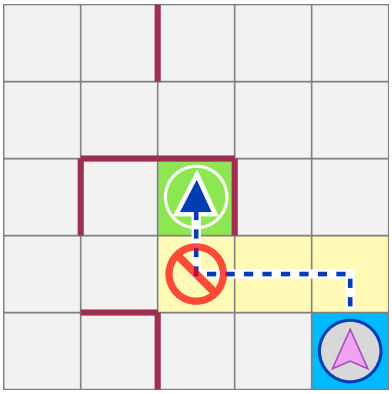
Questa è la risposta:



Se Lefty attraversa queste caselle sarà in grado di raggiunge la destinazione, muovendosi solo nei due modi in cui è in grado di muoversi.

Non c'è un altro modo per raggiungere l'obiettivo con un numero inferiore o uguale di movimenti per Lefty.

Lefty non può prendere la strada diretta perché dovrebbe girare a destra a un certo punto, cosa che non è in grado di fare.



Questa è l'informatica!

Povero Lefty! La sua funzionalità è infatti fortemente limitata. Se solo potesse fare altri movimenti! Se potesse girare a destra e magari anche arrampicarsi sui muri, raggiungere il suo obiettivo sarebbe molto più facile. Lefty si sentirebbe molto più sicuro se avesse una serie di comandi più complessi. Le azioni che un robot può fare purtroppo sono limitate da comandi definiti nel programa (software) del robot.

Ma è davvero necessario? Lefty potrebbe, ad esempio, girare a destra girando a sinistra per tre volte di seguito. Basta abolire la regola secondo cui Lefty deve muoversi in avanti subito dopo aver girato a sinistra. In quel caso potrebbe girare e muoversi in tutte le direzioni. E invece di scavalcare un muro,

potrebbe girarci intorno se c'è abbastanza spazio. Ehi Lefty, puoi vivere per sempre felice e contento con il tuo «set di comandi ridotto», a patto che chi ti programma usi i comandi con saggezza.

In informatica, questi due approcci alla progettazione del set di istruzioni di un *processore* sono i più diffusi: alcuni processori sono CISC (Complex Instruction Set Computer), altri sono RISC (Reduced Instruction Set Computer), come quello usato da Lefty in questo compito. Un CISC di solito ha molte istruzioni diverse che possono essere molto potenti (come per esempio scavalcare un muro), ma sono usate meno frequentemente. Un RISC, invece, ha solo comandi veramente necessari con effetti piuttosto semplici, che vengono usati frequentemente.

Entrambi i tipi di architettura presentano vantaggi e svantaggi. I processori di marche famose sono di tipo CISC o RISC, ma recentemente i processori RISC sono diventati un po' più popolari.

Parole chiave e siti web

- Processore: `https://it.wikipedia.org/wiki/Processore`
- Instruction set: `https://it.wikipedia.org/wiki/Instruction_set`
- CISC: `https://it.wikipedia.org/wiki/Complex_instruction_set_computer`
- RISC: `https://it.wikipedia.org/wiki/Reduced_instruction_set_computer`

6. Tornare a casa

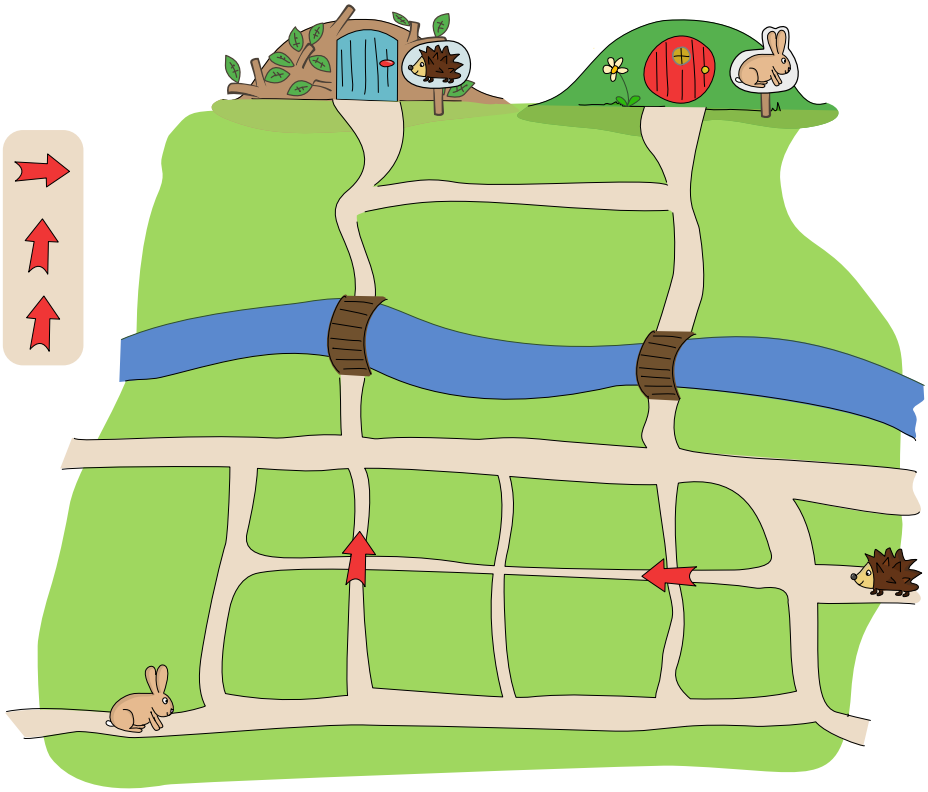
Un coniglio  e un riccio  vogliono tornare entrambi a casa propria:

 e  .

Entrambi seguono il sentiero che hanno davanti. Cambiano direzione solo quando arrivano a un incrocio con una freccia, seguendo la direzione della freccia.

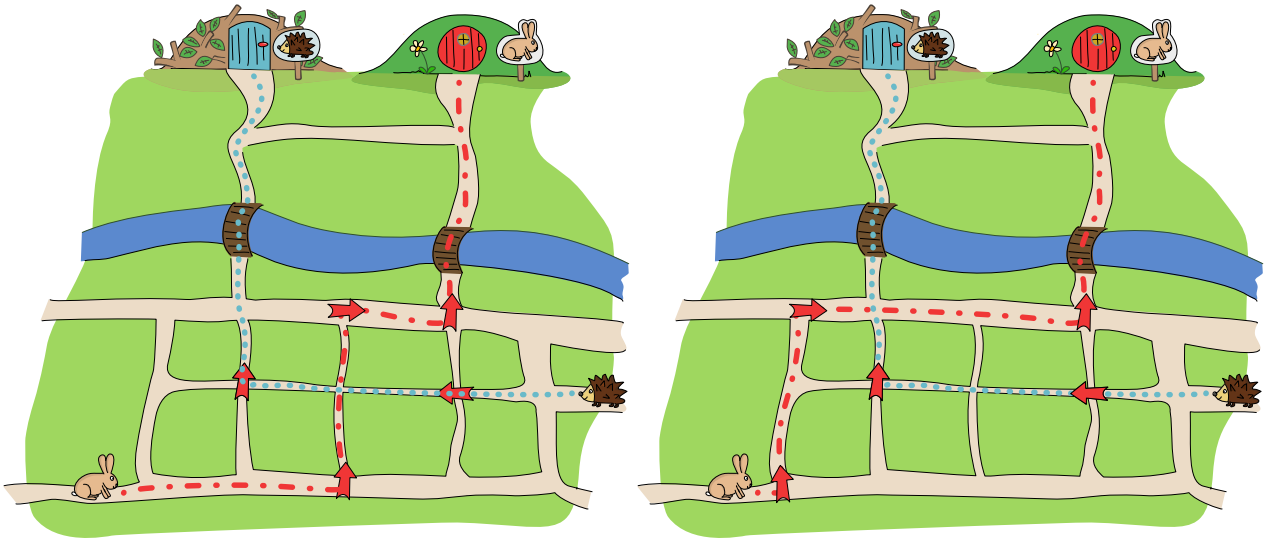
In alcuni incroci sono già presenti delle frecce. Il riccio trova la strada di casa seguendo le frecce, ma il coniglio no. Fortunatamente sono rimaste tre frecce.

Posiziona le tre frecce rimanenti sugli incroci in modo che il coniglio ***e*** *il riccio* ***entrambi*** *trovino la strada di casa.*



Soluzione

Ci sono due risposte corrette:



Per tornare a casa, il coniglio deve a un certo punto «salire» (in direzione delle case). Può farlo al primo o al terzo incrocio che incontra.

È importante che il coniglio e il riccio non arrivino mai alla stessa freccia sul loro percorso, perché altrimenti da lì prenderebbero entrambi la stessa strada e quindi arriverebbero entrambi alla stessa casa. Questo accadrebbe, ad esempio, se il coniglio salisse al secondo o al quarto incrocio e incontrasse delle frecce che indicano al riccio la strada di casa. Il coniglio non può salire nemmeno al quinto incrocio; dovrebbe poi girare a sinistra, ma non c'è nessuna freccia che permetta di girare a sinistra.

Il coniglio può quindi salire solo alla prima o alla terza intersezione. In seguito, le altre frecce devono per forza essere posizionate come mostrato. Questo è l'unico modo per il coniglio di tornare a casa senza disturbare il percorso del riccio.

Questa è l'informatica!

Nel loro cammino, il coniglio e il riccio seguono questa regola all'incrocio: «Se all'incrocio c'è una freccia, seguite la direzione della freccia. Altrimenti, proseguite dritto».

In informatica, una regola di questo tipo si chiama *algoritmo*. L'algoritmo rimane sempre lo stesso: indica se e come cambiare direzione quando si incontra un incrocio. I cambi di direzione sono determinati dalle frecce e sono dati che l'algoritmo *elabora*. In informatica si parla anche di *ingressi* per l'algoritmo: nel del percorso del coniglio e del riccio di questo compito, l'input, cioè la posizione e il tipo di frecce, determina se l'*output* dell'algoritmo è corretto, cioè se i percorsi intrapresi dagli animali li portano entrambi a casa.

Una regola simile potrebbe valere per il lavaggio di un capo di abbigliamento: «Selezionare la temperatura di lavaggio indicata sull'etichetta di lavaggio». Solo se l'input, cioè la temperatura

indicata sull'etichetta di lavaggio, è corretto, anche l'output è corretto, cioè un capo d'abbigliamento pulito esattamente della stessa taglia che aveva prima del lavaggio.

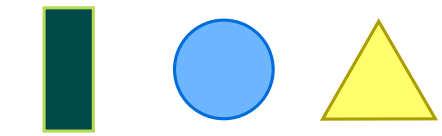
Questi esempi dimostrano che è fondamentale che la qualità dell'output di un algoritmo dipenda dalla qualità dell'input. È il caso, ad esempio, di molti sistemi di intelligenza artificiale. I sistemi di IA lavorano con modelli della realtà e questi modelli sono formati da dati. Se questi dati non sono ben selezionati, anche il modello non funzionerà bene. Un modello di IA sulle malattie, ad esempio, non funzionerà bene per le donne se i dati in ingresso provengono solo da uomini.

Parole chiave e siti web

- Pathfinding: `https://en.wikipedia.org/wiki/Pathfinding`

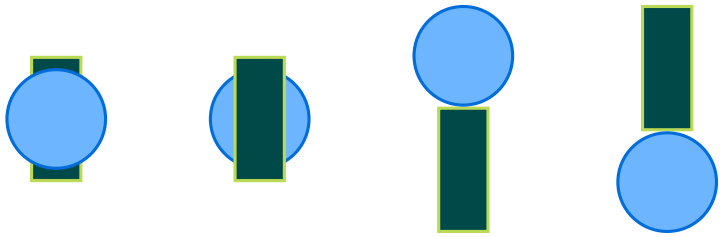
7. Hivobu

Nella terra di Hivobu, ci sono alcune forme alle quali sono assegnati dei nomi.



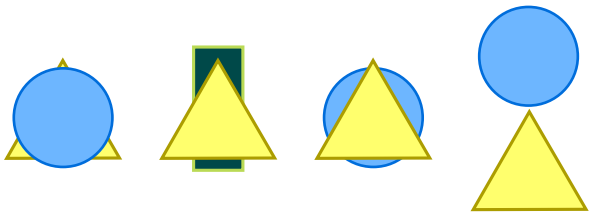
RAH OH TEH

Se si posizionano due forme una sovrapposta all'altra o una sopra o sotto l'altra in Hivobu, la figura finale ha un nuovo nome.



OH RAH CO RAH OH CO OH RAH DU RAH OH DU

Che cosa significa TEH OH CO in Hivobu?



A) B) C) D)

Soluzione

La risposta C è corretta.

Sappiamo come si chiamano le singole forme:

- Il rettangolo  si chiama: RAH;
- il cerchio  significa: OH; e
- Il triangolo  significa: TEH.

Sappiamo anche come si indica quando due forme sono poste una sovrapposta all'altra o una sopra/sotto l'altra: prima viene il nome della prima forma, poi il nome della seconda forma, e poi si aggiunge una parte finale:

- se le forme si sovrappongono: CO;
- se le forme sono messe una sopra l'altra: DU.

Nella risposta C, abbiamo un triangolo (TEH) e un cerchio (OH), con il triangolo sovrapposto al cerchio (CO), dunque abbiamo il nome TEH OH CO

Ma quali sono le altre risposte di Hivobu?

- La risposta A significa: OH TEH CO - un cerchio (OH) e un triangolo (TEH) che si trova dietro (CO) il cerchio.
- La risposta B significa: TEH RAH CO- un triangolo (TEH) e un rettangolo (RAH) che si trova dietro (CO) il triangolo.
- La risposta D significa: OH TEH DU - un cerchio (OH) e un triangolo (TEH) che si trova sotto (DU) il cerchio.

Questa è l'informatica!

I computer sono intelligenti? I computer possono calcolare cose complicate molto velocemente, trovare informazioni su Internet e molto altro. Ma non per questo possiamo considerarli intelligenti, perché per farglielo fare bisogna dirgli esattamente come farlo. Anche i sistemi di intelligenza artificiale con cui sembriamo in grado di parlare normalmente devono essere minuziosamente istruiti su come farlo.

In pratica, i computer capiscono solo istruzioni precise che devono essere formulate in linguaggi con strutture chiare. In informatica, tali linguaggi sono chiamati anche *linguaggi formali*, e i computer comprendono bene solo quei linguaggi formali che hanno una struttura piuttosto semplice.

È come in questo compito: ciò che abbiamo imparato sulla lingua di Hivobu quando si dà un nome ad una figura ha una struttura molto semplice. Prima si nominano due forme, poi un comando dice cosa succede alle due forme. Se si confonde questa *sintassi*, cioè la struttura del linguaggio, ad esempio se si enuncia il comando al centro invece che alla fine, nessuno in Hivobu sa cosa si intende. In informatica, questo tipo di sintassi (dire prima le cose, poi il comando) è noto come *notazione postfissa*. Anche un computer che si aspetta un'istruzione in notazione postfissa si confonderebbe in caso di errore.

Parole chiave e siti web

- Notazione postfissa: `https://it.wikipedia.org/wiki/Notazione_polacca_inversa`

8. Legno per la diga

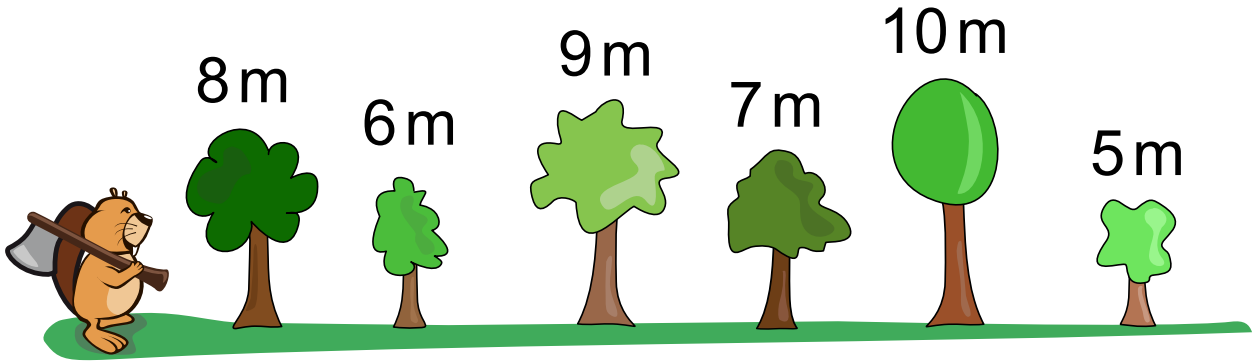
Alcuni castori devono abbattere degli alberi per costruire la loro prossima diga.

La scelta è ricaduta su sei possibili alberi. I castori sanno quanti metri di legno ha ogni albero e vogliono avere il maggior numero di metri di legno possibile. Sono liberi di scegliere il primo albero da abbattere, ma quando devono tagliare l'albero successivo, sono obbligati a seguire due regole:

- Regola 1: l'albero successivo deve essere più a destra di quello precedente, ma non per forza adiacente.
- Regola 2: l'albero successivo deve essere più piccolo, cioè avere meno metri di legno del precedente.

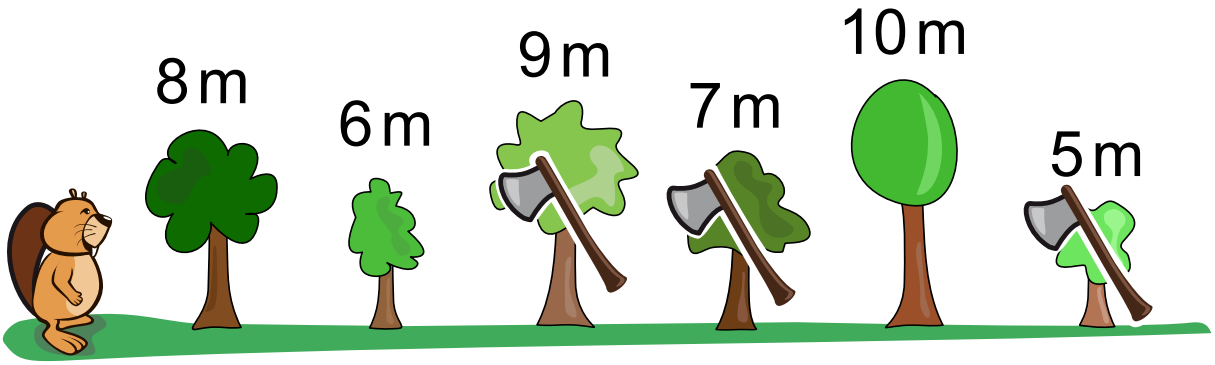
Ad esempio, se tagliano l'albero di 6 metri, possono tagliare solo quello di 5 metri. Alla fine si ritroverebbero con un totale di 11 metri di legno.

Quali alberi possono essere abbattuti dai castori secondo le regole, in modo da ottenere il maggior numero possibile di metri di legno?



Soluzione

La risposta corretta è la seguente:



Il compito è trovare una sottosequenza di alberi che diminuiscono in metri di legno da sinistra a destra in modo che la somma dei metri di legno ottenuti in questa sottosequenza sia massima. Poiché stiamo cercando la massima quantità totale, dobbiamo considerare solo le sottosequenze che non possono essere estese, perché l'aggiunta di qualsiasi albero alla sottosequenza violerebbe le regole. Chiamiamo tali sottosequenze *complete*.

Ad esempio, se iniziamo con l'albero di 8 metri (chiamaiamolo «8»), ci sono due sottosequenze complete: **8, 6, 5** e **8, 7, 5**. Le sottosequenze **8, 7** (estendibile con l'albero 5) e **8, 5** (estendibile con l'albero 6 o l'albero 7) non sono invece complete perché estendibili.

La tabella mostra tutte le sottosequenze complete. Mostra anche quanti metri di legno hanno a disposizione i castori alla fine quando abbattono gli alberi per ogni sottosequenza completa.

sottosequenza completa	metri di legno
8, 6, 5	19
8, 7, 5	20
6, 5	11
9, 7, 5	21
10, 5	15

I castori possono quindi abbattere gli alberi 9, 7 e 5 secondo le loro regole, in modo da ottenere la massima quantità di legno possibile, ossia 21 metri.

Questa è l'informatica!

I castori cercano di trovare un numero di alberi in modo da ottenere il maggior numero possibile di metri di legno considerando le loro regole. In questo compito, 21 metri di legno è il risultato *ottimale*, cioè il migliore che possono ottenere. Rispondendo a questo compito, avete quindi risolto un *problema di ottimizzazione*.

In generale, un problema di ottimizzazione consiste nel trovare il miglior valore possibile. Può trattarsi di un valore massimo, come nel caso di questo compito, o di un valore minimo, ad esempio se si sta

cercando il modo più veloce (minor tempo) per andare a scuola. È nella natura della maggior parte delle persone cercare l'opzione più veloce, più breve, più economica o più divertente per un progetto: questa è ottimizzazione. Si parla di ottimizzazione anche quando un'azienda cerca di pagare ai propri dipendenti il minor stipendio possibile, o quando si affitta un appartamento nel tentativo di ottenere il più alto reddito locativo possibile.

Molti problemi di ottimizzazione sono così complessi che vengono risolti con l'aiuto di sistemi informatici, cioè programmi per computer o «app». Se un'azienda di consegna pacchi sta cercando un modo per pianificare i percorsi dei suoi veicoli di consegna in modo da ridurre al minimo il consumo di energia, o un fornitore di energia vuole utilizzare i suoi impianti per produrre energia rinnovabile nel modo più economico possibile, è necessario tenere conto di così tanti dati e condizioni o il sistema di controllo deve essere così flessibile che spesso si ottengono risultati molto migliori con l'aiuto dei computer. È quindi positivo in informatica sia possibile utilizzare molti metodi per risolvere i problemi di ottimizzazione e possa aiutarci a definire quali metodi di ottimizzazione possono essere utilizzati per quali tipi di problemi.

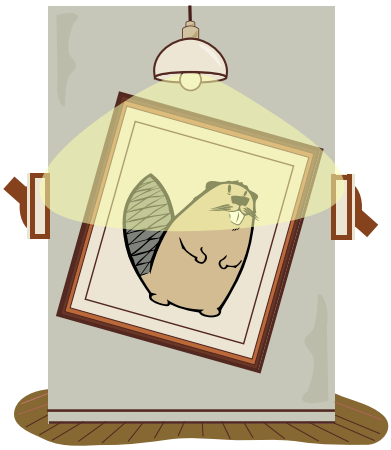
Per coloro che sono particolarmente interessati, il problema di questo compito è noto in informatica anche come «Somma massima decrescente successiva». Su `questo sito web` è possibile vedere come approcci semplici (ma non ottimizzati) per arrivare alla soluzione di un problema di ottimizzazione possano gradualmente portare ad approcci sempre migliori e come questi possano essere implementati sottoforma di programmi.

Parole chiave e siti web

- Problema di ottimizzazione: `https://u-helmich.de/inf/TSP/TSPIndex.html`
- Metodo forza bruta: `https://it.wikipedia.org/wiki/Metodo_forza_bruta`

9. Lampada pazzesca

Viktoria Volt ha costruito una lampada pazzesca. La lampada ha due interruttori: uno a sinistra e uno a destra. Ogni interruttore può essere **acceso** (,) o **spento** (,).



La lampada ha anche un terzo interruttore segreto: il quadro! A seconda di come è appeso il quadro ( ,  o ), gli interruttori funzionano in modo diverso.

Questa tabella mostra come gli interruttori accendono o spengono la luce:

Immagine	Interruttore	Luce
	esattamente uno è **acceso**: o	**accesa**
	altrimenti	**spenta**
	entrambi sono **spenti**:	**spenta**
	altrimenti	**accesa**
	entrambi sono **accesi**	**accesa**
	altrimenti	**spenta**

L'interruttore sinistro è spento , quello destro è acceso .

Come deve essere appeso il quadro in modo che la luce sia spenta?

Soluzione

La risposta è: se il quadro è appeso così  (cioè inclinato a sinistra), la luce è spenta.

Osserviamo da vicino i tre modi in cui il quadro può essere appeso:

Poiché esattamente un interruttore è **acceso** (ovvero quello di destra), la luce è **accesa**. Questa risposta è sbagliata.

Poiché entrambi gli interruttori della luce non sono **spenti**, la luce è **accesa**. Anche questa risposta è sbagliata.

Poiché entrambi gli interruttori della luce non sono accesi, la luce è spenta. Questa risposta è corretta.

Questa è l'informatica!

Gli interruttori della lampada di Viktoria possono avere due valori ciascuno: **acceso** o **spento**. Anche la luce ha solo questi due valori. In ogni posizione dell'immagine, il valore della luce può essere calcolato in base ai valori degli interruttori, come mostrato nella tabella.

Anche la più piccola unità di memoria dei computer, il *bit*, può accettare solo due valori. In informatica, questi valori sono spesso chiamati **on** e **off**, ma talvolta anche **vero** e **falso** o addirittura **1** e **0**. Così come è possibile combinare i numeri con gli *operatori* (+, -, ×, :) e calcolare nuovi valori a partire da essi, è possibile combinare i bit con gli *operatori logici*. Questi operatori sono integrati nei computer come *porte logiche*. Le porte possono essere utilizzate per combinare e calcolare i bit in tutti i modi possibili. I computer possono quindi elaborare e modificare i dati in quasi tutti i modi.

L'immagine in questo compito decide quale operazione logica attuare con i due interruttori della luce. Se l'immagine è appesa diritta, gli interruttori funzionano come uno XOR («either or»): se l'interruttore destro o quello sinistro sono accesi, la luce è accesa. Se il quadro è inclinato a destra, funzionano come un OR («or»): se l'interruttore sinistro o quello destro (cioè entrambi o uno dei due) sono accesi, la luce è accesa. Se l'immagine è inclinata a sinistra, gli interruttori funzionano come un AND («and»): se gli interruttori destro e sinistro sono accesi (entrambi contemporaneamente), la luce è accesa.

Parole chiave e siti web

- Logica: `https://it.wikipedia.org/wiki/Logica`
- Porte logiche: `https://it.wikipedia.org/wiki/Porta_logica`

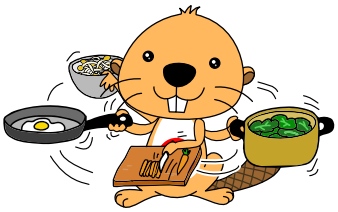
- Algebra di Bool: `https://it.wikipedia.org/wiki/Algebra_di_Boole`

10. Bibimbap

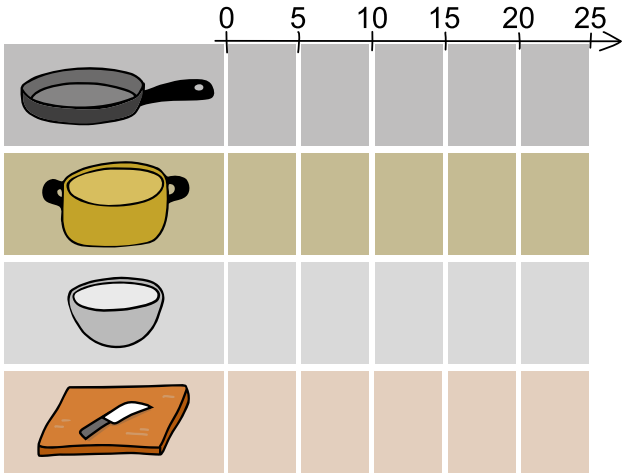
Un cuoco vuole preparare un piatto tradizionale coreano, il bibimbap (비빔밥). Utilizza quattro utensili: una pentola, una padella, un tagliere e una ciotola. I quattro ingredienti per il bibimbap vanno preparati in questo modo:

	Spinaci:	prima cuocere (per 10 minuti), poi tagliare (5 minuti)
	Germogli di soia:	prima innaffiare (5 minuti), poi cuocere (10 minuti)
	Carote:	prima tagliare (5 minuti), poi friggere (10 minuti)
	Uovo:	friggere (5 minuti)

Lo chef può lavorare con diversi strumenti contemporaneamente, ma può utilizzare solo uno strumento per un ingrediente alla volta. Ad esempio, lo chef può bollire gli spinaci nella pentola e friggere un uovo nella padella, ma non può friggere un uovo e delle carote nella padella allo stesso tempo.

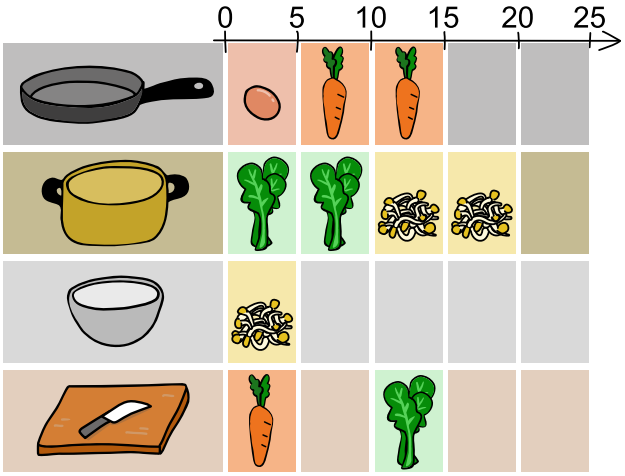


Definisci un piano che consenta allo chef di preparare gli ingredienti per il bibimbap nel minor tempo possibile.



Soluzione

La risposta corretta è la seguente:



Lo chef può utilizzare più strumenti contemporaneamente, ma uno strumento può essere utilizzato solo per un ingrediente alla volta. È quindi necessario considerare per ogni strumento in quali momenti si possono lavorare gli ingredienti e il piano complessivo più breve per la preparazione del tutto.

La preparazione richiederà almeno 20 minuti, poiché sia gli spinaci che i germogli devono essere cotti nella pentola per 10 minuti ciascuno. Se gli spinaci vengono cotti per primi, i germogli possono essere messi a bagno nello stesso momento. Gli spinaci possono essere tritati mentre i germogli cuociono. È preferibile friggere prima l'uovo nella padella, in quanto il cuoco può tagliare le carote durante questo tempo e friggerle dopo l'uovo, come mostrato nell'immagine qui sopra.

Il piano sopra mostra come tutti gli ingredienti possano essere preparati non solo in un tempo minimo di 20 minuti, ma anche in un tempo massimo di 20 minuti. Tutti i piani che non richiedono più di 20 minuti in totale e in cui (a) le fasi di lavorazione hanno la durata corretta e (b) le fasi di lavorazione dei singoli ingredienti avvengono una dopo l'altra e nell'ordine corretto sono risposte corrette.

Questa è l'informatica!

Questo compito riguarda la pianificazione di una sequenza di attività. In informatica, questa attività è nota come *scheduling*. Come per molti problemi di pianificazione, le risorse disponibili sono limitate: la pentola, la padella, il tagliere e la ciotola sono utensili singoli e utilizzabili uno per volta. Inoltre, alcune attività possono svolgersi contemporaneamente (il cuoco è davvero bravo!), mentre altre devono svolgersi in sequenza, perché utilizzano lo stesso ingrediente o lo stesso utensile.

Questa pianificazione è un problema molto antico a cui si pensava anche prima che esistessero i computer. Le tecniche utilizzate oggi negli strumenti di pianificazione dei progetti risalgono all'epoca dei primi computer, ovvero agli anni cinquanta. Tra queste c'è il «metodo del percorso critico» (in inglese: Critical Path Method, CPM), che corrisponde all'incirca alla procedura che abbiamo seguito

nella spiegazione della risposta precedente: determinare una sequenza di attività interdipendenti che richieda il maggior tempo possibile e che quindi debba essere svolta senza interruzioni tra le attività.

Poco dopo la pubblicazione del CPM, John Fondahl, professore dell'Università di Stanford, rimase insoddisfatto delle implementazioni informatiche del metodo. Ha presentato quindi i propri approcci per implementare il CPM senza computer. È interessante notare che proprio questi approcci sono ancora oggi utilizzati nella maggior parte delle soluzioni software per la pianificazione dei progetti, come aveva previsto anche John Fondahl stesso. Gli algoritmi esistono da oltre mille anni e ancora oggi vale la pena di pensare ad essi senza per forza considerare immediatamente come possano essere eseguiti dai computer. Alla fine, quanto migliore è l'algoritmo, tanto più è adatto all'implementazione su computer!

Parole chiave e siti web

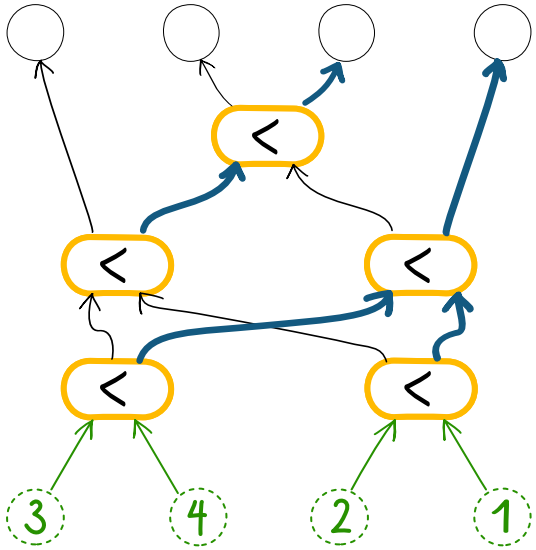
- Scheduling: `https://www.swisseduc.ch/informatik/theoretische_informatik/scheduling/algorithmus1.html`
- Metodo del percorso critico: `https://it.wikipedia.org/wiki/Metodo_del_percorso_critico`
- Relazione tecnica di Stanford John Fondahl (vedi prefazione): `https://catalog.hathitrust.org/Record/005766951`

11. La macchina dei numeri

I castori hanno una macchina particolare.

Nei campi di immissione ◯ si inseriscono quattro numeri, ad esempio 3, 4, 2 e 1.

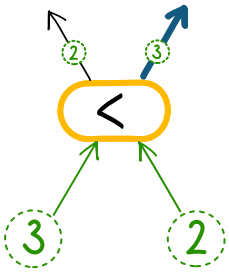
I numeri si muovono verso l'alto nella macchina lungo frecce e snodi < fino ai campi di uscita ◯.



Ciascuno dei cinque snodi confronta i due numeri in entrata e manda...

- ... il numero più piccolo a sinistra e
- ... il numero più grande a destra.

Per esempio:



A cosa serve la macchina?

A) Ordina i numeri in ordine decrescente. Risultato: 4, 3, 2, 1.

B) Ordina i numeri in ordine crescente. Risultato: 1, 2, 3, 4

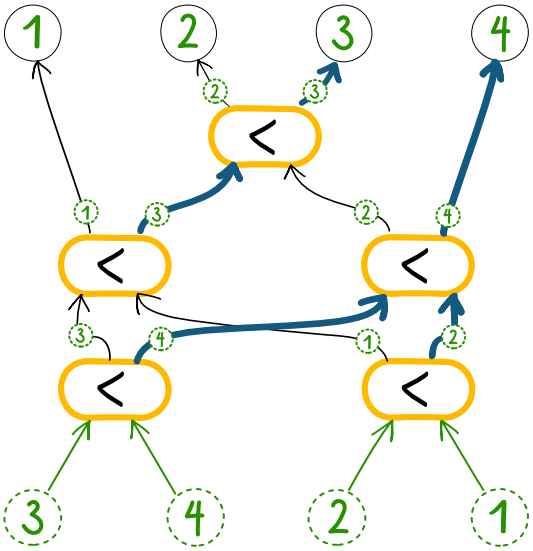
C) Visualizza i numeri nello stesso ordine. Risultato: 3, 4, 2, 1

D) Visualizza i numeri in ordine inverso. Risultato: 1, 2, 4, 3

Soluzione

La risposta B è quella corretta. La macchina ordina i numeri in ordine crescente. Il risultato con in numeri dati nell'esempio è 1, 2, 3, 4

Provando la macchina, è possibile determinare la risposta corretta ed escludere tutte le altre risposte.



Nel primo livello di esecuzione, la macchina confronta due numeri. Nel secondo livello confronta i due numeri più grandi e i due numeri più piccoli (risultanti dal confronto avvenuto nel primo livello) per determinare il valore più grande (massimo) e il valore più piccolo (minimo) dei quattro numeri. L'ordinamento viene quindi completato confrontando i due numeri rimanenti.

Questa è l'informatica!

In informatica, la macchina dei numeri che deriva da questo compito è nota come *rete di ordinamento*. Una rete di ordinamento è costituita da una serie di componenti identici e molto semplici, i *comparatori*. Ogni comparatore riceve due valori numerici su due linee di ingresso e li confronta. In seguito inoltra i valori a due linee di uscita: il valore più piccolo sulla linea di sinistra e il valore più grande sulla linea di destra. Le reti di ordinamento vengono anche visualizzate in orizzontale, con gli ingressi a sinistra e le uscite a destra e i comparatori come ponti. In questo caso, il numero più piccolo viene solitamente indirizzato verso l'alto e il numero più grande verso il basso.

Qualsiasi sequenza di numeri può essere ordinata combinando un numero sufficiente di comparatori. La macchina dei numeri in questo compito mostra come quattro numeri possano essere ordinati con cinque comparatori. Sono necessari almeno nove comparatori per cinque numeri e almeno dodici per sei numeri.

Le reti di ordinamento hanno un'importanza pratica nell'informatica perché i comparatori sono componenti elettronici molto semplici e quindi poco costosi e le reti di ordinamento possono quindi essere facilmente realizzate sull'hardware (e quindi, a livello fisico). Alcuni dei comparatori possono funzionare simultaneamente, accelerando così il processo di ordinamento. Infatti in generale il numero

di passi non paralleli in una rete di ordinamento è inferiore al numero di elementi da ordinare. Uno svantaggio delle reti di ordinamento è che sono progettate solo per ingressi di lunghezza fissa.

Parole chiave e siti web

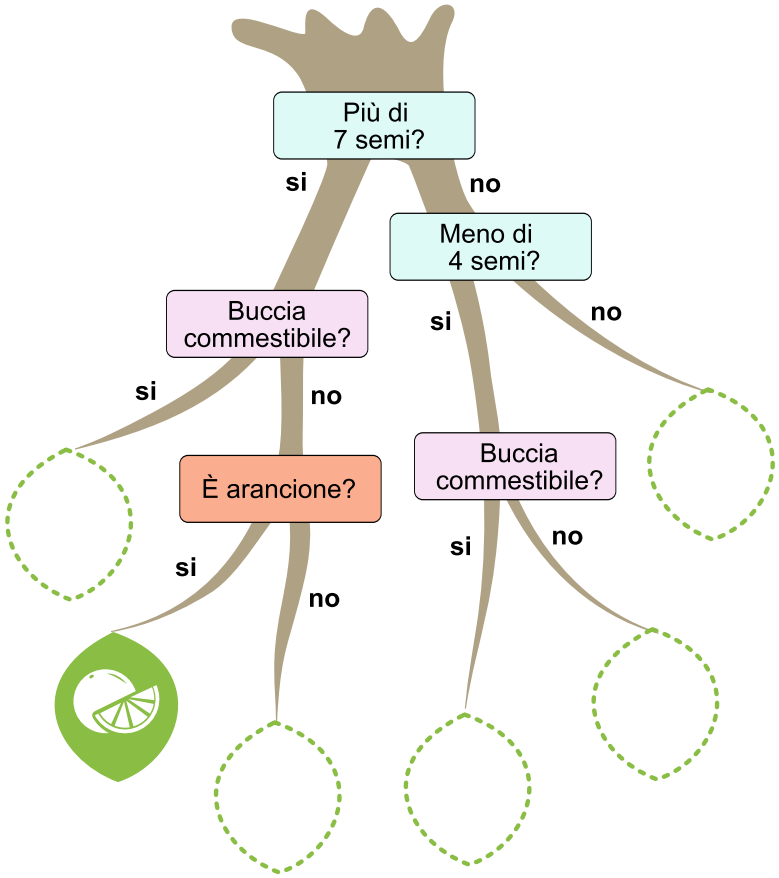
- *Rete di ordinamento*: `https://www.coderdojotrento.it/csun2/`
- *Comparatore*: `https://it.wikipedia.org/wiki/Comparatore_(elettronica)`
- *Calcolo parallelo*: `https://it.wikipedia.org/wiki/Calcolo_parallelo`

12. Albero di decisione

In una classe scolastica si analizzano i frutti. Per ogni frutto vengono analizzate tre proprietà e dai valori si determina il tipo di frutto. Le proprietà sono: colore esterno, numero di semi e commestibilità della buccia. La classe ha annotato i valori e i tipi di frutta risultanti per dieci frutti in una tabella:

Colore	**numero di semi**	**buccia commestibile?**	**tipo del frutto**
verde	391	no	cocomero
giallo	5	sì	mela
arancione	9	no	arancia
giallo	0	no	banana
rosso	5	sì	mela
verde	0	sì	uva
rot	206	sì	fragola
verde	6	sì	mela
arancione	10	no	arancione
marcio	173	sì	fragola

Per creare questa tabella, la classe ha usato un albero di decisione. Un albero di decisione ha l'aspetto di un albero capovolto: la radice è in alto e le foglie sono in basso. La radice e i rami dell'albero sono etichettati con domande a cui si può rispondere con un sì o un no. Partendo dall'alto, si risponde alla prima domanda e ci si sposta sul ramo a seconda della risposta (sì o no). Si continua rispondendo alla domanda successiva, e così via, raggiungendo una foglia. Su questa foglia sarà indicato il tipo di frutto.

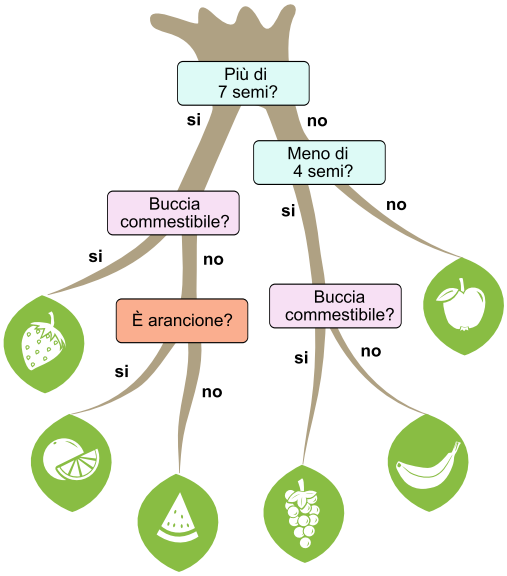


Purtroppo l'albero delle decisioni usato dalla classe si è rotto dopo il decimo frutto e quasi tutte le foglie sono cadute.

Dov'erano le foglie?

Soluzione

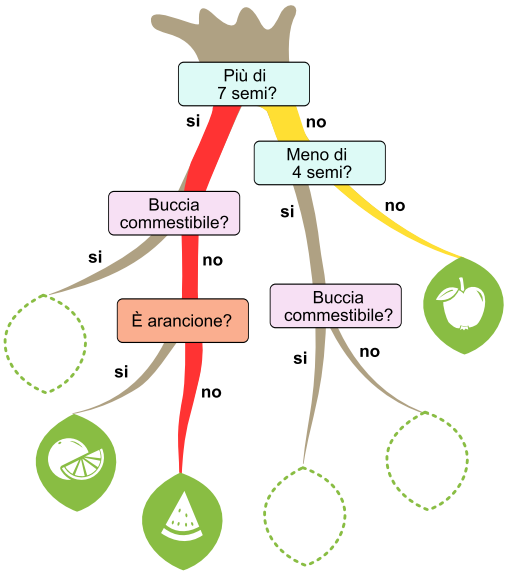
Questa è come si presenta l'albero di decisione una volta aggiustato:



I frutti sulle foglie sono (da sinistra a destra): Fragola, arancia, cocomero, uva ![uva-in], banana, mela.

Come si fa a trovare i posti giusti per le foglie?

Le varietà di frutta nella tabella sono state determinate utilizzando l'albero di decisione. Scorriamo quindi le righe della tabella e vediamo a quale foglia conduce il percorso dell'albero di decisione in base alle domande. La foglia con la varietà di frutta specificata nell'ultima colonna della tabella deve trovarsi a questo punto. L'immagine mostra i percorsi attraverso l'albero di decisione per le prime due righe della tabella:



Riga 1 (percorso rosso): Il frutto ha 391 semi (Più di 7 semi? - Sì), la buccia non è commestibile (Buccia commestibile? - No) e il frutto è verde (È arancione? - No). Questo percorso conduce alla terza posizione della foglia da sinistra. Qui deve andare la foglia con il tipo di frutto inserito in questa riga della tabella: il cocomero.

Riga 2 (percorso giallo): Il frutto ha 5 semi (Più di 7 semi? - No. Meno di 4 semi? - No). Il percorso conduce alla posizione della foglia all'estrema destra, quindi la foglia con la mela deve andare lì.

In modo simile

- La riga 3 conduce alla seconda posizione della foglia da sinistra, dove si trova già l'arancia;
- La riga 4 conduce alla seconda posizione della foglia da destra, dove deve trovarsi la banana;
- La riga 5 conduce alla mela;
- La riga 6 conduce fino alla terza posizione della foglia da destra, dove deve trovarsi l'uva;
- Infine, la riga 7 conduce fino alla posizione della foglia all'estrema sinistra, dove deve trovarsi la fragola.

Questa è l'informatica!

Un *albero di decisione* è un modo semplice ma intelligente di categorizzare (o in termini tecnici: *classificare*) le cose in gruppi o categorie e di prendere decisioni in base a questa categorizzazione. Gli alberi decisionali sono utilizzati in molti sistemi informatici: ad esempio nei programmi di diagnostica medica, negli assistenti online che aiutano a trovare i prodotti e anche nei videogiochi, quando il software deve decidere come deve reagire un personaggio.

L'albero di decisione in questo compito è stato creato dall'insegnante di biologia. Ha impostato l'albero e ha considerato quali domande sono importanti, in quale ordine devono essere poste e come l'albero dovrebbe ramificarsi in modo che i frutti possano essere classificati correttamente.

In un'area dell'informatica che è diventata sempre più importante negli ultimi anni, ovvero l'intelligenza artificiale (IA), di solito si tratta anche di classificare i dati osservati e quindi di prendere decisioni. Gli strumenti di classificazione necessari - come ad esempio un albero di decisione - devono essere determinati automaticamente dai dati osservati. I metodi informatici per la creazione automatica di strutture di classificazione e decisione a partire dai dati sono anche detti metodi di *machine learning* o di *data science*. Gli alberi decisionali possono anche essere «appresi», cioè costruiti automaticamente, utilizzando algoritmi come CART o C4.5. Questi metodi scoprono da soli, sulla base di numerosi esempi, quali domande devono essere poste per prendere buone decisioni. Gli alberi decisionali generati automaticamente vengono utilizzati, ad esempio, per riconoscere le e-mail di spam, fare diagnosi mediche o identificare specie vegetali.

La maggior parte dei sistemi di IA conosciuti non utilizza alberi decisionali. Molti sistemi utilizzano invece grandi *modelli* di *reti neurali*. Queste strutture non contengono domande decisionali comprensibili, ma piuttosto modelli statistici complessi che di solito sono difficili da comprendere per gli esseri umani.

Parole chiave e siti web

- Albero di decisione, `https://it.wikipedia.org/wiki/Albero_di_decisione`
- Classificazione, `https://it.wikipedia.org/wiki/Schema_di_classificazione`
- Intelligenza artificiale, AI Unplugged `https://www.aiunplugged.org`

13. Stella di luci

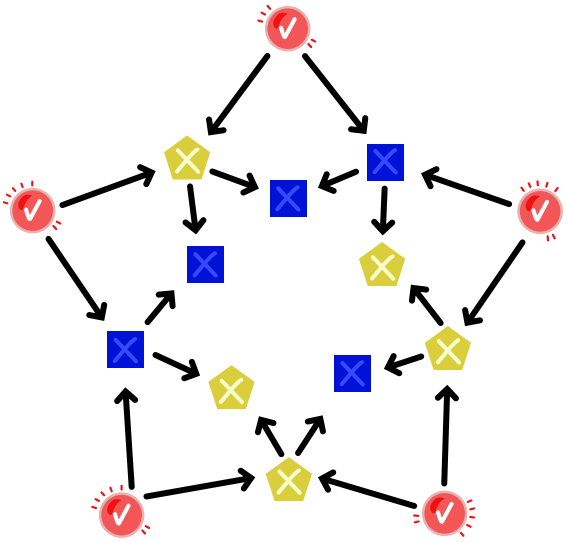
Sophie ha tre tipi di luci nella sua scatola elettronica: rossa rotonda, blu quadrata e gialla pentagonale. Sophie ha deciso di collegare alcune luci per formare una «stella di luci», nella quale le luci rosse sono alle estremità della stella, mentre le luci blu e gialle sono all'interno della stella. Queste ultime ricevono la corrente tramite le frecce. Sophie ha strutturato la stella in modo che ogni luce blu o gialla abbia esattamente due «luci di controllo».

Ecco come funzionano le luci:

- Sophie può accendere e spegnere solo le luci rosse.
- Una luce blu è accesa quando entrambe le luci di controllo sono accese; altrimenti è spenta.
- La luce gialla è accesa quando è accesa esattamente una delle due luci di controllo, altrimenti è spenta.

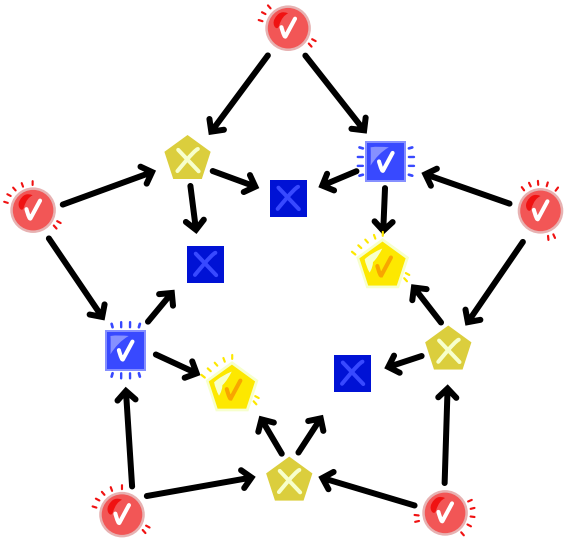
Sophie accende tutte le luci rosse.

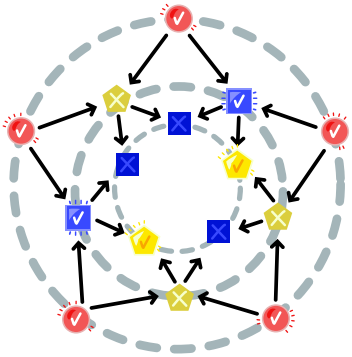
Quali altre luci sono accese?



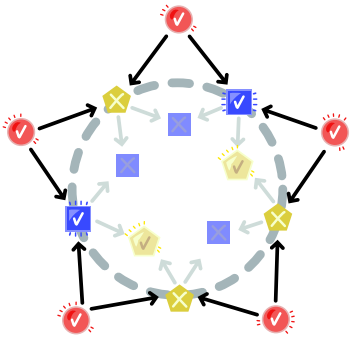
Soluzione

Questa è la risposta:

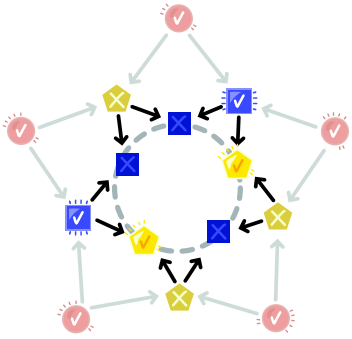




Le luci formano tre anelli: un anello esterno, un anello centrale e un anello interno. L'anello esterno ha solo luci rosse, mentre gli altri due anelli hanno luci sia blu che gialle.



Sophie accende tutte le luci rosse esterne, dunque tutte le luci di controllo delle luci centrali sono accese. Ciò significa che solo e unicamente le luci blu nell'anello centrale risulteranno accese (perché entrambe le luci di controllo sono accese) mentre le luci gialle saranno spente (perché entrambe le luci di controllo sono accese: ricordiamoci che le luci gialle sono accese solo quando esattamente *una* luce di controllo è accesa e l'altra è spenta).



Nell'anello interno è il contrario: ogni luce dell'anello interno ha una luce di controllo gialla e blu proveniente dall'anello centrale. Sappiamo già che una di queste due luci è sempre accesa. Ecco perché le luci gialle dell'anello interno saranno accese, mentre quelle blu saranno spente.

Questa è l'informatica!

Le luci della stella di Sophie possono essere solo accese o spente. Si può quindi dire che ogni luce ha due possibili stati o «valori». Questo sistema binario è diverso da altri sistemi: un semaforo, ad esempio, ha tre valori («rosso», «giallo» e «verde»), mentre un orologio digitale o la cassa di un supermercato ne hanno moltissimi. Per convenzione, in informatica, i due stati «acceso» e «spento» sono rappresentati dai numeri **1** e **0**.

Come esistono operatori matematici, ad esempio l'addizione e la sottrazione, per calcolare con i numeri, esistono anche operatori per calcolare con questi due valori (1 e 0). Questi operatori logici calcolano un risultato a partire da due valori di ingresso. In questo compito ne vengono utilizzati due:

- L'operatore AND (dall'inglese «e») restituisce 1 se, e solo se, entrambi i valori di ingresso sono 1. Se anche solo uno dei due ingressi è 0, il risultato è 0.
- L'operatore XOR (dall'inglese «o esclusivo») restituisce 1 se i valori di ingresso sono diversi tra loro (cioè, uno è 1 e l'altro è 0). Se i valori di ingresso sono uguali (entrambi 1 o entrambi 0), il risultato è 0.

Il termine «esclusivo» dello XOR serve a distinguerlo dall'operatore OR (dall'inglese «o»). L'operatore OR è «inclusivo»: restituisce 1 se almeno uno dei due ingressi è 1 (e anche se lo sono entrambi). L'operatore XOR, invece, «esclude» il caso in cui entrambi gli ingressi siano 1.

Questi e altri operatori simili sono noti in informatica come operatori logici. Essi possono essere facilmente costruiti come circuiti elettronici. Questi circuiti, a loro volta, sono gli elementi di base dei processori dei computer. L'intero sistema funziona perché l'unità più piccola di informazione di un computer, il bit, ha anch'essa solo due valori (1 o 0). Assemblando abilmente questi semplici circuiti, i computer possono eseguire calcoli molto complicati e controllare l'esecuzione di qualsiasi programma.

Parole chiave e siti web

- Logica: `https://it.wikipedia.org/wiki/Logica`
- Algebra di Boole: `https://it.wikipedia.org/wiki/Algebra_di_Boole`
- AND: `https://it.wikipedia.org/wiki/Congiunzione_logica`
- XOR: `https://it.wikipedia.org/wiki/Disgiunzione_esclusiva`
- OR: `https://it.wikipedia.org/wiki/Disgiunzione_logica`
- Flip-flop: `https://it.wikipedia.org/wiki/Flip-flop`

14. Istruzioni di montaggio

Hai questi mattoncini:

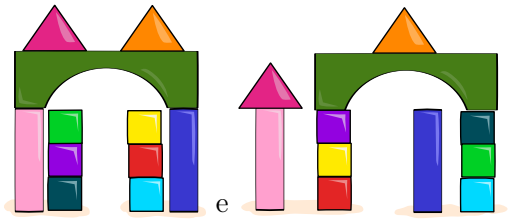
6 cubi | 2 cuboidi | 1 ponte | 2 piramidi

Il tuo amico ti dà queste istruzioni per costruire delle strutture con i mattoncini:

1. Prendere tre cubi.
2. Impilare i cubi l'uno sull'altro per costruire una torre.
3. Costruire un'altra torre con i tre cubi rimanenti.
4. Posizionare i cuboidi accanto alle torri.
5. Posizionare il ponte sulla struttura.
6. Prendere le due piramidi e posizionarle sul proprio edificio.

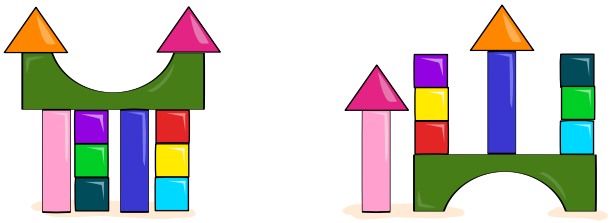
Quando si costruisce, è necessario seguire l'ordine delle sei istruzioni. È comunque possibile costruire molte strutture diverse con le istruzioni di costruzione.

Due esempi:

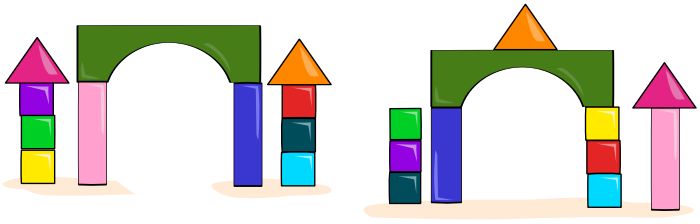


e

Ecco altre quattro costruzioni. ***NON*** *è possibile costruirne uno con le istruzioni date. Quale?*



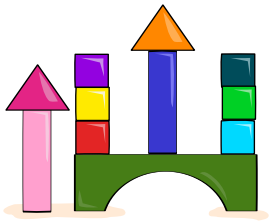
A) B)



C) D)

Soluzione

La risposta B è quella corretta.



Sappiamo già che è possibile costruire strutture diverse con le istruzioni date. Le istruzioni non sono quindi *univoche*. Una cosa è però chiara: la sequenza delle singole istruzioni deve essere seguita nel dato ordine. Vediamo più da vicino cosa può accadere se si seguono passo dopo passo le istruzioni per la costruzione. Definiamo in una tabella l'aspetto della costruzione dopo aver eseguito le singole istruzioni e verifichiamo quale costruzione corrisponde alla descrizione.

Dopo la fase	Descrizione	A	B	C	D
1 e 2	Una torre di tre cubi.	✓	✗	✓	✓
3	Ora le torri di tre cubi sono due.	✓	✗	✓	✓
4	Due torri e due cuboidi.	✓	✗	✓	✓
5	Due torri e due cuboidi con un ponte sopra. Il ponte si trova sulle parti della struttura costruite in precedenza, siano esse torri o cuboidi.	✓	✗	✓	✓
6	Come sopra, con le piramidi che si trovano sulle parti della struttura costruite in precedenza (torri, cuboidi o ponti).	✓	✗	✓	✓

La costruzione della risposta B non corrisponde alle istruzioni di costruzione. In particolare, l'istruzione 5 non è stata implementata correttamente, infatti l'istruzione 5 specifica che il ponte deve essere posizionato *sulla* struttura esistente. Nella struttura della risposta B, tuttavia, il ponte non si trova *sopra* almeno un'altra parte della struttura, ma solo *sotto* alcune parti, ossia le due torri e un cuboide. La struttura della risposta B può essere costruita solo se il ponte viene costruito sotto alle due torri e a un cuboide.

La tabella mostra che le costruzioni per le risposte A, C e D possono essere costruiti utilizzando le istruzioni date.

Questa è l'informatica!

Questo compito mostra l'importanza di istruzioni chiare e non ambigue. Ne sono un esempio le istruzioni di costruzione o le ricette di cucina. Istruzioni formulate in modo ambiguo possono portare a risultati diversi e imprevedibili. Per esempio, le istruzioni per la costruzione di questo compito non

specificano chiaramente dove devono essere posizionati o posati esattamente i singoli cuboidi o le torri costruite con i cubi.

I computer hanno bisogno di istruzioni chiare e prive di ambiguità se vogliono lavorare come gli esseri umani desiderano. L'informatica chiama tali istruzioni chiare *algoritmi*: istruzioni passo-passo che servono per completare un compito o risolvere un problema. Anche le singole istruzioni di un algoritmo devono essere inequivocabili, perché a differenza degli esseri umani i computer non possono interpretare o «indovinare». Hanno bisogno di istruzioni inequivocabili per poter svolgere i compiti in modo prevedibile. Istruzioni ambigue nei programmi per computer possono far sì che i programmi non vengano eseguiti come previsto, che il software non funzioni bene o che produca risultati inaspettati.

Prima di scrivere un programma, lo sviluppatore deve considerare quali *vincoli* devono essere definiti per ottenere un risultato prevedibile. In questo compito tali vincoli possono includere l'esatta disposizione verticale e orizzontale dei blocchi, il posizionamento delle parti della struttura e il modo in cui i blocchi devono essere collegati a quelli già presenti.

Parole chiave e siti web

- *Algoritmo*: `https://it.wikipedia.org/wiki/Algoritmo`
- *Programmazione a vincoli*: `https://it.wikipedia.org/wiki/Programmazione_a_vincoli`

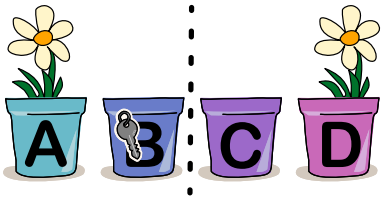
15. Vasi di fiori

Il castoro Florian decora l'ingresso della sua tana con vasi di fiori. In alcuni vasi è piantato **esattamente un fiore**, altri invece sono **vuoti**.

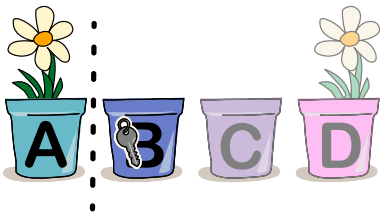
Florian vuole nascondere la chiave di riserva di casa sua in un vaso. Per sapere come trovarla, ci spiega il suo metodo:

«Per prima cosa, osservate tutti i vasi e contate quanti fiori sono piantati in totale nei vasi. Se il numero di fiori è pari, la chiave si trova nella metà sinistra dei vasi, altrimenti si trova nella metà destra. Ora guardate solo la metà in cui si trova la chiave e ripetete il procedimento finché non rimane un solo vaso. È lì che è nascosta la chiave».

Florian mostra un esempio di come trovare la chiave in 4 vasi A, B, C, D.



Considera i vasi A, B, C e D. Ci sono in totale 2 fiori, quindi un numero **pari**. Ciò significa che la chiave si trova nella metà **sinistra**, ovvero nel vaso A o nel vaso B.



Considera i vasi A e B. C'è un totale di 1 fiore, quindi un numero **dispari**. Ciò significa che la chiave si trova nella metà **destra**, ovvero nel vaso B.

Florian ha otto vasi di fiori e nasconde la chiave nel vaso C. In quali vasi deve piantare dei fiori in modo che la chiave possa essere trovata con il suo metodo?

Ci sono diverse risposte corrette. Anche lo 0 è un numero pari.

Soluzione

Una risposta corretta è:

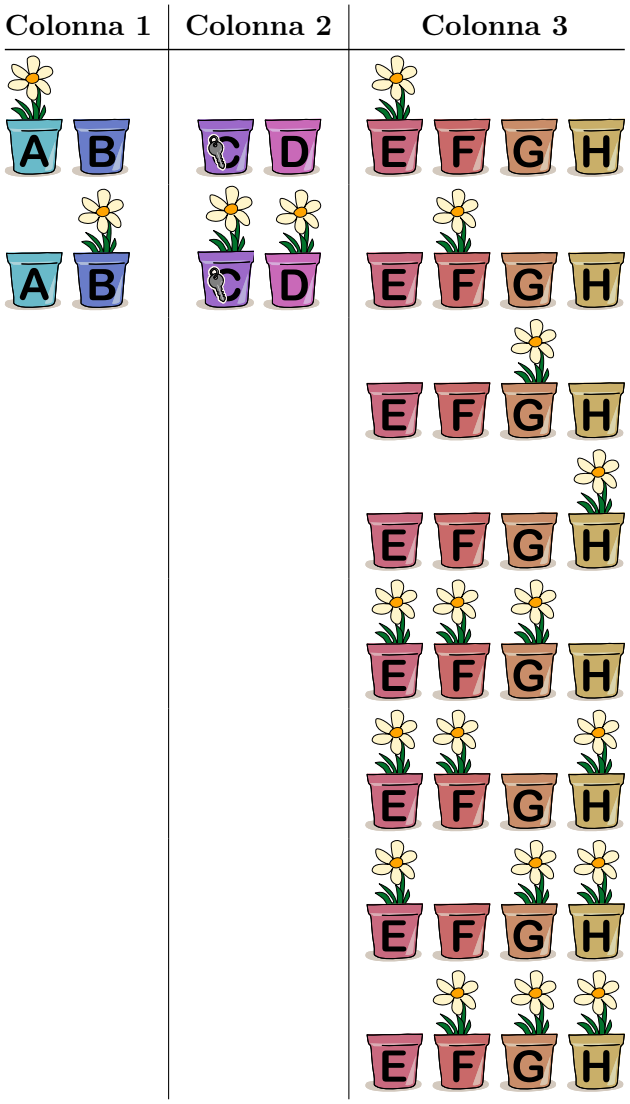
Un'altra risposta corretta è:

In totale, ci sono ben 32 risposte corrette ed è possibile determinarle in questo modo:

È più facile costruire la risposta «dal basso verso l'alto», cioè iniziare con piccole parti della risposta e lavorare da lì fino alla risposta completa.

- Se la chiave è nel vaso C, i vasi C e D vengono considerati per ultimi nella ricerca della chiave. Per decidere a favore della metà sinistra C, deve esserci un numero pari di fiori in C e D. Quindi o c'è un fiore in entrambi i vasi o in nessuno dei due.
- Nella fase precedente della ricerca si sono considerati i vasi da A a D. Poiché deve essere scelta la metà destra (C e D), nei vasi da A a D è necessario avere un numero dispari di fiori. Come descritto in precedenza, le metà C e D hanno comunque un numero pari di fiori. Pertanto, solo uno dei due vasi A o B deve contenere un fiore.
- Nella prima fase di ricerca della chiave, si considerano tutti i vasi. Poiché deve essere scelta la metà sinistra (da A a D), è necessario un numero pari di fiori in totale. Siccome nella metà sinistra già considerata c'è un numero dispari di fiori, occorre un numero dispari di fiori nella metà destra (da E a H). Potete quindi scegliere tra 1 o 3 fiori per i vasi E-H e distribuirli a piacimento tra di essi.

La seguente tabella riassume tutte le risposte corrette: Selezionando un'opzione da ogni colonna, è possibile compilare una risposta corretta. In questo modo si ottengono tutte le $2 \times 2 \times 8 = 32$ risposte corrette.



Questa è l'informatica!

Florian *codifica* la posizione della chiave utilizzando una sequenza di fiori nei suoi vasi. I suoi amici possono decodificare questa rappresentazione perché conoscono il metodo di codifica di Florian. Per consentire ai computer di elaborare i dati, questi non vengono memorizzati in una forma naturale e facilmente comprensibile per gli esseri umani, ma in una forma codificata che può essere letta dai computer. L'informatica conosce molti metodi di codifica. Alcune codifiche hanno lo scopo di risparmiare spazio di archiviazione; si parla di *compressione*. Se la codifica richiede la conoscenza di una cosiddetta «chiave» oltre alla conoscenza del metodo della stessa, si parla anche di *crittografia*. In entrambi i casi, è importante che la decodifica ripristini esattamente i dati originali, cioè che sia unica. Ogni combinazione di vasi con e senza fiori definisce chiaramente in quale vaso è nascosta la chiave, quindi il «codice dei fiori» in questo compito soddisfa questo requisito.

Il metodo di Florian per trovare la chiave nei vasi di fiori funziona in modo simile alla *ricerca binaria*, in quanto lo spazio di ricerca viene dimezzato a ogni passo. In questo modo si raggiunge l'obiettivo

abbastanza rapidamente. Con il doppio dei vasi di fiori nei quali la chiave potrebbe essere nascosta, sarebbe necessario solo un passo in più.

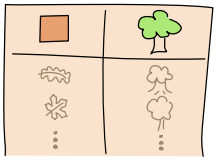
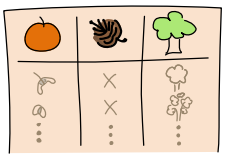
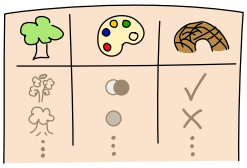
Il modo migliore per trovare la risposta corretta a questo compito è il cosiddetto *approccio bottom-up*. Le parti più piccole della risposta vengono considerate per prime, poiché le parti più grandi dipendono da esse. Invece di provare tutte le possibili combinazioni, che in questo caso sarebbero $2^8 = 256$, si può arrivare più rapidamente alla risposta corretta lavorando abilmente.

Parole chiave e siti web

- Codice: `https://it.wikipedia.org/wiki/Codice_(teoria_dell'informazione)`
- Ricerca binaria (o dicotomica): `https://it.wikipedia.org/wiki/Ricerca_dicotomica`

16. Dalla foglia al legno

A Emil e ai suoi amici piace fare escursioni. Durante le loro escursioni, raccolgono informazioni sugli alberi che vedono e le raccolgono in lunghe tabelle.

Tabella	Descrizione
	**Severin** raccoglie informazioni sulle forme delle foglie e sulle specie di alberi corrispondenti.
	**Quirina** raccoglie informazioni sui frutti degli alberi, se provengono da conifere e sulle specie di alberi corrispondenti.
	**Ladina** raccoglie informazioni sulle specie di alberi, sul colore del loro legno e sulla loro idoneità alla costruzione di dighe per castori.

Emil ha trovato una foglia nella foresta e ne riconosce la forma. Ora vuole scoprire se la specie di albero in questione fornisce legno idoneo per costruire dighe.

A chi dei suoi amici Emil deve chiedere, e in quale ordine, per scoprirlo?

A) Solo Ladina.

B) Prima Severin, poi Quirina.

C) Prima Severin, poi Ladina.

D) Prima Quirina, poi Severin, poi Ladina.

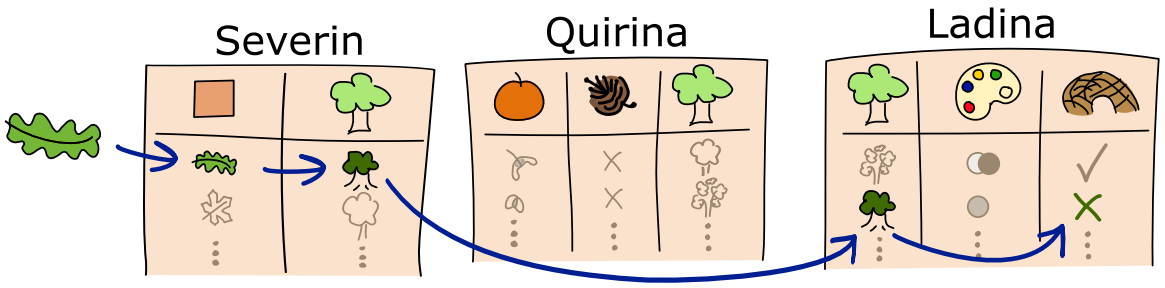
Soluzione

La risposta C è corretta: prima Severin, poi Ladina.

L'informazione se una specie di albero fornisce legno idoneo per le dighe può essere trovata solo nella tabella di Ladina. Tuttavia, se Emil ha solo informazioni sulla foglia, non può selezionare una riga dalla tabella di Ladina. Ha bisogno di informazioni sulla specie di albero ![specie] o sul colore del legno ![colore]. Non è quindi sufficiente chiedere a Ladina e di conseguenza la risposta A è sbagliata.

La tabella di Quirina non contiene informazioni sulle foglie o sul legno per dighe. La sua tabella non è utile a Emil, quindi le risposte B e D sono sbagliate.

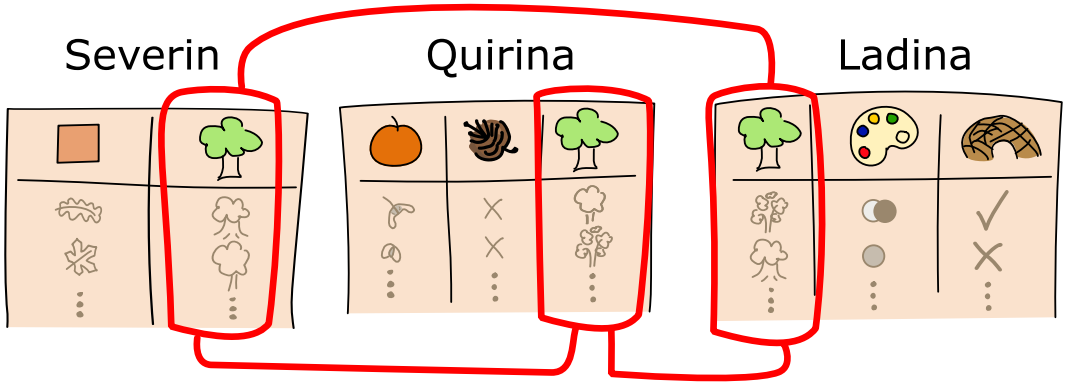
La tabella di Severin contiene informazioni sulle foglie. Siccome Emil conosce la forma della foglia, può prima selezionare la riga appropriata nella tabella di Severin e ottenere le informazioni mancanti sulla specie di albero. Può quindi utilizzare questa informazione per selezionare la riga appropriata nella tabella di Ladina e ottenere le informazioni che sta cercando sul legno. Per esempio, se Emil scopre dalla forma della foglia e dalla tabella di Severin che si tratta di una foglia di quercia, può selezionare la riga appropriata nella tabella di Ladina e scoprire se le querce forniscono legno idoneo per costruire dighe.



Questa è l'informatica!

Questo compito illustra i concetti di base delle *basi di dati (o database) relazionali*. I database sono utilizzati con grande frequenza nei sistemi informatici per gestire piccole e grandi quantità di dati. I database relazionali sono costituiti da tabelle con dati, proprio come le tabelle create dagli amici di Emil. In una tabella ogni colonna è chiamata *attributo* e in ogni riga è presente un record di dati. Le tabelle possono avere *relazioni* con altre tabelle tramite un attributo comune - qui la «specie di albero» - che nel gergo informatico è chiamata *chiave* e stabilisce relazioni tra tabelle diverse.

La richiesta di Emil di informazioni su del buon legno per una diga per castori, basata sulla forma delle foglie, sarebbe chiamata *query* in un sistema con database. Questa query richiede l'unione di diverse tabelle per ottenere le informazioni desiderate. L'operazione *join* combina temporaneamente le righe di diverse tabelle in una tabella comune più grande, utilizzando una chiave comune. In questo modo, i dati distribuiti in diverse tabelle (in questo caso le specie di alberi, la forma delle foglie e il legno di castoro) possono essere uniti per rispondere alla query. I dati delle tabelle degli amici di Emil in questo compito possono essere collegati tra loro tramite la specie di albero:



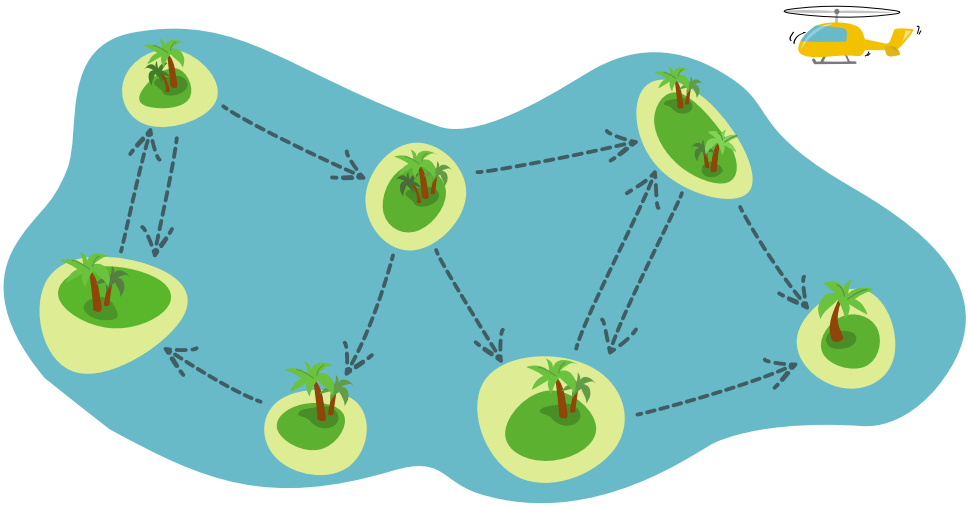
I database relazionali sono così importanti che esiste un linguaggio separato in informatica per interrogare ed eseguire altre operazioni sui database, chiamato *SQL (Structured Query Language)*.

Parole chiave e siti web

- Base di dati: `https://it.wikipedia.org/wiki/Base_di_dati`
- Chiave: `https://it.wikipedia.org/wiki/Chiave_(basi_di_dati)`
- SQL: `https://it.wikipedia.org/wiki/Structured_Query_Language`
- Ennupla: `https://it.wikipedia.org/wiki/Ennupla`

17. Arcipelago dei castori

Ci sono sette isole al largo della costa della Bebrasia, collegate da traghetti che viaggiano da isola a isola. I traghetti si muovono seguendo la direzione delle frecce come mostrato dalla mappa.



Un team di ricerca vuole esplorare la fauna selvatica di tutte e sette le isole. Ecco come si sono organizzati:

1. Il team di ricerca vola con un elicottero  su un'isola,
2. utilizza i traghetti per visitare altre isole, e
3. infine, ritorna sull'isola dove è atterrato per il volo di ritorno con l'elicottero.

Il team si rende conto che un solo viaggio non è sufficiente per visitare tutte le isole.

Qual è il numero minimo di viaggi che il team deve fare?

A) 2 viaggi
B) 3 viaggi
C) 4 viaggi
D) 5 viaggi
E) 6 viaggi
F) 7 viaggi

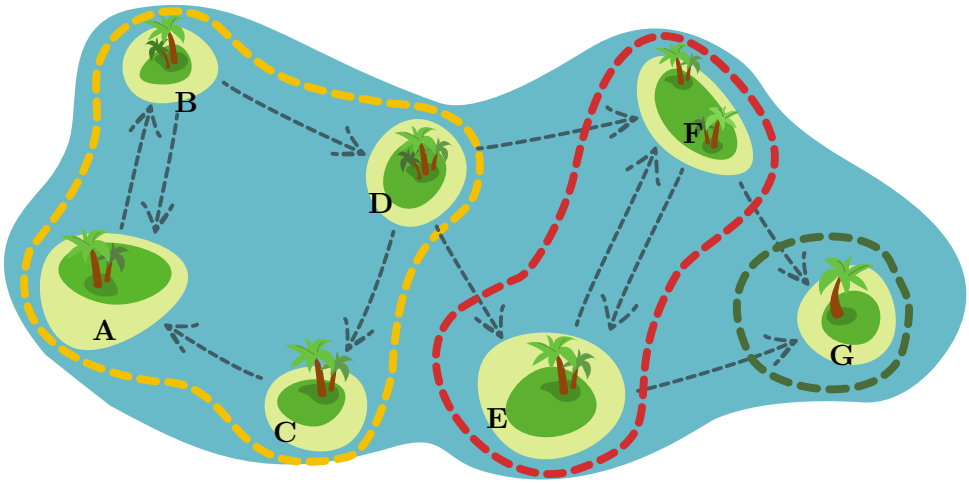
Soluzione

La risposta corretta è 3 viaggi.

Per minimizzare il numero di viaggi, il team deve massimizzare le isole visitate in ogni singola escursione. Ciascuna escursione ha però un vincolo fondamentale: deve iniziare e terminare sulla stessa isola, quella di atterraggio dell'elicottero. Di conseguenza, una volta scelta un'isola «base», il team può visitare via traghetto soltanto quelle isole dalle quali è anche possibile fare ritorno all'isola «base».

Questo vincolo porta a suddividere l'arcipelago in «gruppi di raggiungibilità reciproca». Un gruppo è definito come un insieme di isole in cui, da qualsiasi isola è possibile raggiungere via traghetto qualsiasi altra isola dello stesso gruppo. Scegliendo una qualsiasi isola di un gruppo come punto di atterraggio, il team può esplorare l'intero gruppo in un unico viaggio. Non è possibile, però, includere isole esterne, poiché una volta lasciato il gruppo non sarebbe più garantito il ritorno all'elicottero. Il numero minimo di escursioni coincide quindi con il numero di questi gruppi.

Per identificare i gruppi, si procede così: si seleziona un'isola non ancora assegnata e si includono nel suo gruppo tutte le isole che sono sia raggiungibili da essa, sia in grado di raggiungerla a loro volta. Si ripete il processo per le isole rimanenti. Come mostra la figura, le sette isole formano tre gruppi distinti: {A, B, C, D}, {E, F} e {G}. Per visitarle tutte, il team dovrà quindi effettuare tre viaggi.



Ma perché un solo viaggio non è sufficiente? Si possono visitare tutte le altre isole da alcune di esse (per esempio partendo dall'isola C)! Purtroppo in questo caso si lascia il gruppo della prima isola e quindi non si può tornare all'elicottero.

Questa è l'informatica!

Le isole sono parzialmente collegate da traghetti. Questo insieme di traghetti e isole forma un modello matematico chiamato *grafo*. Un grafo è una struttura che descrive le relazioni (i collegamenti) tra un insieme di oggetti (le isole). In termini tecnici, le isole sono detti i *nodi* (o *vertici*) del grafo, mentre le connessioni dei traghetti sono gli *archi diretti* (o *spigoli orientati*) del grafo. Sono detti «diretti»

(o «orientati») poiché la rotta di un traghetto può essere a senso unico (ad esempio, dall'isola C alla A, ma non necessariamente dalla A alla C).

Il concetto di gruppo precedentemente definito può essere applicato anche ai grafi e corrisponde esattamente a ciò che in informatica viene chiamato *componente fortemente connessa*. Una componente fortemente connessa è un insieme di nodi in un grafo diretto tale che, per ogni coppia di nodi (A e B) all'interno di quell'insieme, esiste un percorso da A a B e un percorso da B ad A. I grafi e le loro componenti fortemente connesse sono fondamentali in molte applicazioni:

- Nel World Wide Web, sono gruppi di siti web direttamente o indirettamente collegati tra loro.
- Nelle reti sociali, sono «bolle» (o community) di utenti che si seguono tutti, direttamente o indirettamente, in una rete.
- Nelle reti di trasporto, sono regioni in cui è possibile viaggiare tra tutte le fermate.

L'informatica fornisce algoritmi efficienti per identificare queste componenti in qualsiasi grafo. Il più celebre è l'algoritmo di Tarjan, sviluppato da Robert Tarjan. Tarjan è un eminente informatico americano che ha ideato numerosi algoritmi fondamentali, vincendo il «Turing Award» (il premio più prestigioso del settore) a soli 38 anni.

Parole chiave e siti web

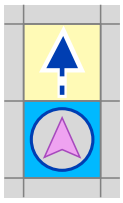
- Grafo diretto: `https://it.wikipedia.org/wiki/Digrafo_(matematica)`
- Grafo connesso: `https://it.wikipedia.org/wiki/Grafo_connesso`
- Componente fortemente connessa: `https://it.wikipedia.org/wiki/Componente_fortemente_connessa`

18. Lefty II

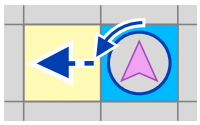
Il robot *Lefty* si muove su una griglia composta da caselle quadrate. Tra le caselle possono esserci dei muri rossi. Lefty deve raggiungere l'obiettivo verde.

Lefty può muoversi solo in due modi:

Avanzare di una casella

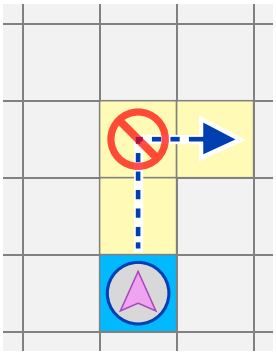


Girare a sinistra e avanzare immediatamente di una casella

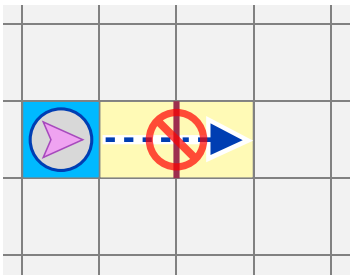


Quindi ci sono azioni che Lefty non può fare:

. . . **non** può girare a destra e. . .

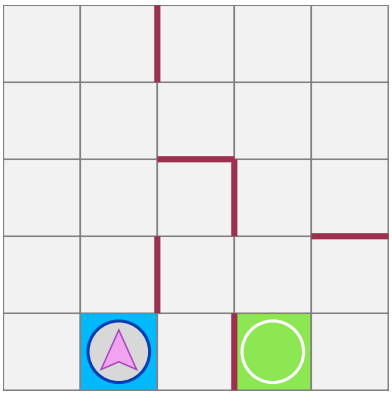


non può passare attraverso i muri.



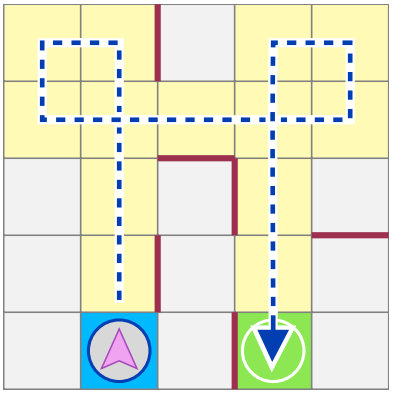
Quali caselle deve attraversare Lefty per raggiungere la destinazione?

Seleziona ***il minor numero possibile di caselle****.*



Soluzione

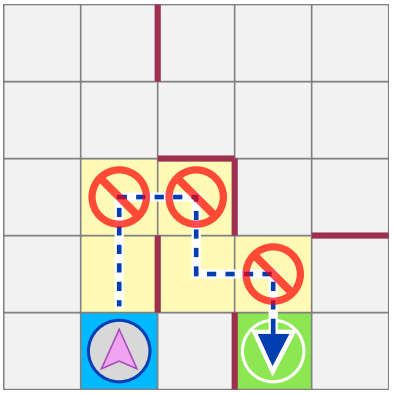
Questa è la risposta:



Se Lefty attraversa queste caselle sarà in grado di raggiunge la destinazione, muovendosi solo nei due modi in cui è in grado di muoversi.

Non c'è un altro modo per raggiungere l'obiettivo con un numero inferiore o uguale di movimenti per Lefty.

Lefty non può prendere la strada diretta perché dovrebbe girare a destra a un certo punto, cosa che non è in grado di fare.

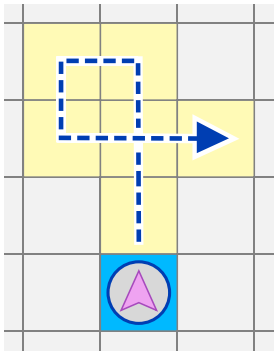


Questa è l'informatica!

Povero Lefty! La sua funzionalità è infatti fortemente limitata. Se solo potesse fare altri movimenti! Se potesse girare a destra e magari anche arrampicarsi sui muri, raggiungere il suo obiettivo sarebbe molto più facile. Lefty si sentirebbe molto più sicuro se avesse una serie di comandi più complessi. Le azioni che un robot può fare purtroppo sono limitate da comandi definiti nel programa (software) del robot.

Ma è davvero necessario? Lefty potrebbe, ad esempio, girare a destra girando a sinistra per tre volte di seguito. Basta abolire la regola secondo cui Lefty deve muoversi in avanti subito dopo aver girato a sinistra. In quel caso potrebbe girare e muoversi in tutte le direzioni. E invece di scavalcare un muro, potrebbe girarci intorno se c'è abbastanza spazio. In altre parole, un *set di istruzioni ridotto*

può essere sufficiente per un robot. Per implementare comportamenti più complessi che si verificano meno frequentemente, si possono progettare delle *subroutine* che combinano diversi comandi semplici in uno più complesso. Ad esempio, una subroutine potrebbe descrivere (ed essere usata due volte nella risposta precedente) come Lefty può riuscire a cambiare direzione verso destra in determinate condizioni:



In informatica, questi due approcci alla progettazione del set di istruzioni di un *processore* sono i più diffusi: alcuni processori sono detti CISC (Complex Instruction Set Computer), altri sono detti RISC (Reduced Instruction Set Computer), come quello usato da Lefty in questo compito. Un CISC di solito ha molte istruzioni diverse che possono essere molto potenti (come scavalcare un muro), ma sono usate meno frequentemente. Un RISC, invece, ha solo comandi veramente necessari con effetti piuttosto semplici, che vengono usati frequentemente.

Entrambi i tipi di architettura presentano vantaggi e svantaggi. I processori di marche famose sono di tipo CISC o RISC, ma recentemente i processori RISC sono diventati un po' più popolari.

Parole chiave e siti web

- Processore: `https://it.wikipedia.org/wiki/Processore`
- Instruction set: `https://it.wikipedia.org/wiki/Instruction_set`
- CISC: `https://it.wikipedia.org/wiki/Complex_instruction_set_computer`
- RISC: `https://it.wikipedia.org/wiki/Reduced_instruction_set_computer`

19. Labirinto

In uno dei giochi per computer di Momo, un robot deve attraversare una serie di caselle per raggiungere una destinazione. Su alcune caselle sono presenti degli ostacoli che non possono essere superati, ma solo aggirati cambiando direzione. In ogni livello del gioco, gli ostacoli e la destinazione possono trovarsi posizioni. Quando il robot raggiunge la destinazione, il livello è completato.

Momo può controllare il robot con una serie di comandi, definendo un programma. Per i suoi programmi può utilizzare questi quattro comandi:

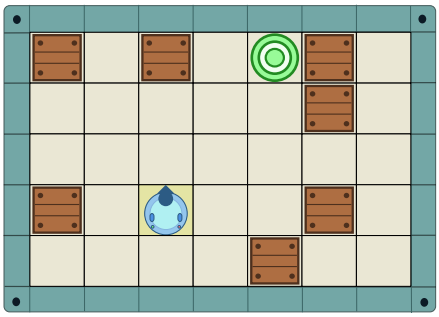
Avanza di una casella.

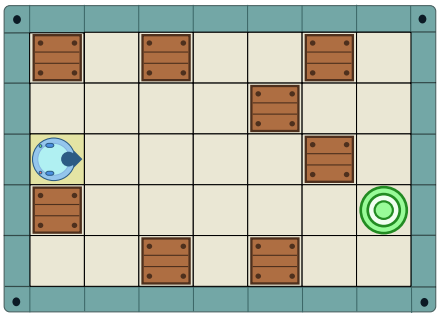
Avanza finché non puoi più proseguire.

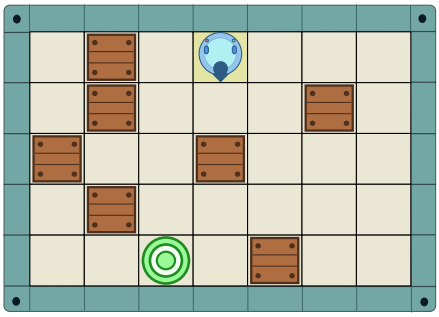
Gira di 90 gradi in senso orario.

Gira di 90 gradi in senso antiorario.

Momo conosce i tre livelli successivi e vuole scrivere un programma con il minor numero possibile di comandi che possa completare tutti e tre i livelli.







Crea questo programma per Momo!

Avanza di una casella.

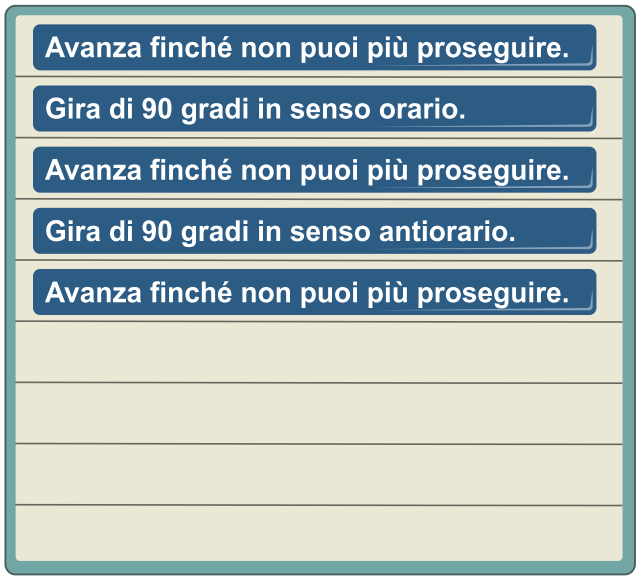
Avanza finché non puoi più proseguire.

Gira di 90 gradi in senso orario.

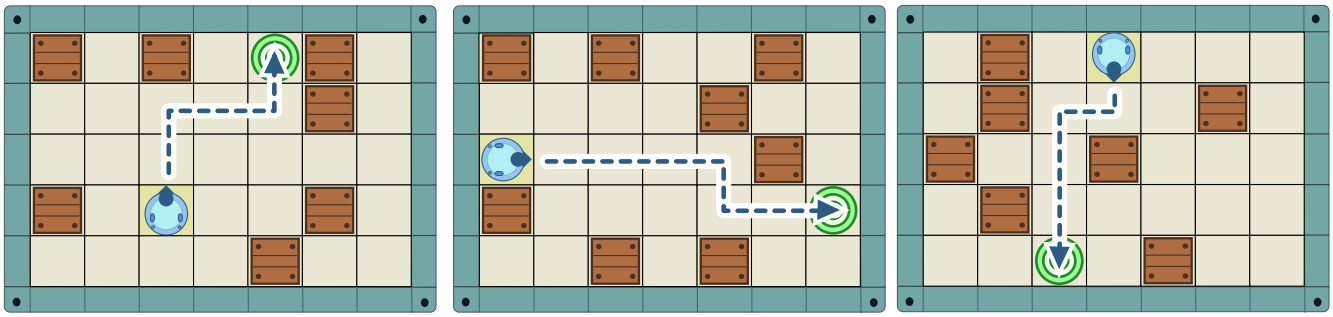
Gira di 90 gradi in senso antiorario.

Soluzione

La risposta corretta è la seguente. I singoli livelli possono essere completati con tanti programmi diversi, ma solo un programma unico è in grado di completare tutti e tre i livelli:



Le frecce mostrano come il programma controlla il robot nei tre livelli:



Nel livello 1 (a sinistra), la destinazione si trova in alto a destra sopra il robot. Pertanto, il comando «Avanzare finché non è più possibile proseguire» deve essere usato almeno due volte: una volta per spostarsi fino all'ostacolo e una volta per spostarsi verso la destinazione. Il robot guarda verso l'alto all'inizio e deve guardare nuovamente verso l'alto anche alla fine. Deve quindi girare prima in senso orario per andare a destra e poi in senso antiorario per guardare di nuovo verso l'alto. Un programma che completa il livello 1 ha quindi almeno cinque comandi.

Nel livello 2 (al centro), il robot deve evitare un ostacolo sulla strada e ha bisogno ancora una volta del comando «Avanzare finché non è più possibile proseguire». Un programma che completa il livello 2 ha quindi anche in questo caso almeno cinque comandi.

Il programma con cinque comandi gestisce anche il livello 3 (a destra). Non esiste un programma più breve per tutti e tre i livelli.

Questa è l'informatica!

I quattro comandi che il robot interpreta in questo compito formano un piccolo linguaggio di programmazione. Il robot può essere controllato mettendo in sequenza i comandi. In informatica, tali sequenze sono chiamate *sequenze di istruzioni* o *sequenze*. La sequenza è la struttura di controllo più semplice e basilare della programmazione strutturata e forma un *algoritmo*.

Il comando «Avanzare di una casella» non è strettamente necessario. Il programma finale infatti contiene più volte il comando «Avanzare finché non è più possibile proseguire». Con questo comando, il passo in avanti viene ripetuto finché il robot non incontra un ostacolo o il bordo del campo. Nella programmazione strutturata, le *ripetizioni* sono un'altra struttura di controllo di base. Possono esistere ripetizioni con un numero fisso di esecuzioni (per esempio, «Avenzare di cinque caselle»), ma forse le più importanti sono le ripetizioni con condizioni di annullamento, come in «Avanzare finché non è più possibile proseguire». Siccome con questo comando il robot può percorrere distanze rettilinee di diversa lunghezza, il programma può completare tutti e tre i livelli.

Esistono livelli che il programma che abbiamo definito non è in grado di gestire, ad esempio se il robot deve cambiare direzione più volte per raggiungere la destinazione. Gli informatici cercano di progettare gli *algoritmi* per i loro programmi in modo che non siano adattati a casi specifici, ma che funzionino *generalmente*, cioè per tutti i casi possibili. Il lavoro degli informatici è anche quello di capire se esistono già algoritmi risolutivi per problemi simili. I livelli del gioco di Momo sono un po' come dei labirinti, ed esistono algoritmi di soluzione generali per i labirinti. Sebbene in alcuni casi determinino percorsi verso l'obiettivo più complicati del necessario, sono generici e quindi in tutti i casi trovano una soluzione.

Parole chiave e siti web

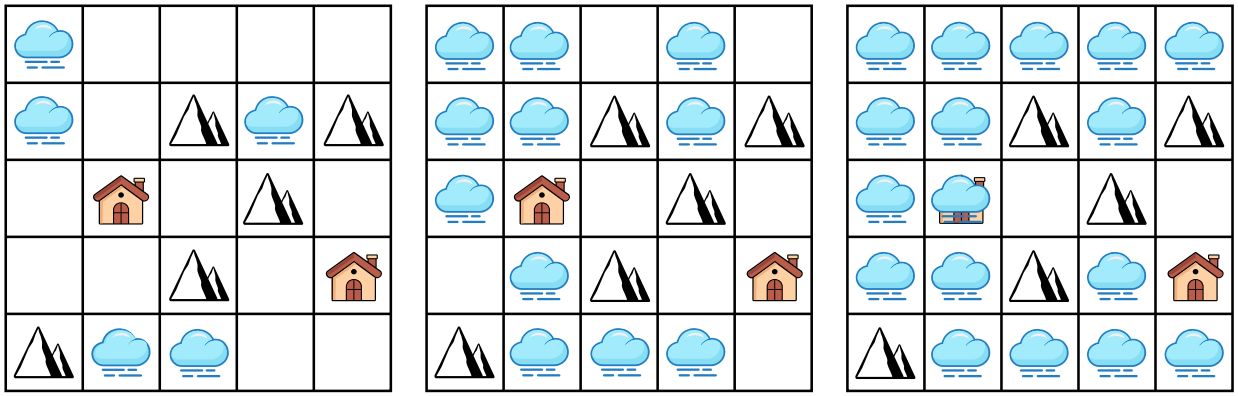
- Programmazione strutturata:
  `https://de.wikipedia.org/wiki/Strukturierte_Programmierung`
- Algoritmi per la risoluzione di labirinti:
  `https://it.wikipedia.org/wiki/Algoritmi_per_la_risoluzione_di_labirinti`

20. Giornata nebbiosa

Oggi c'è nebbia nella terra dei monti e si diffonde ad ogni ora che passa.

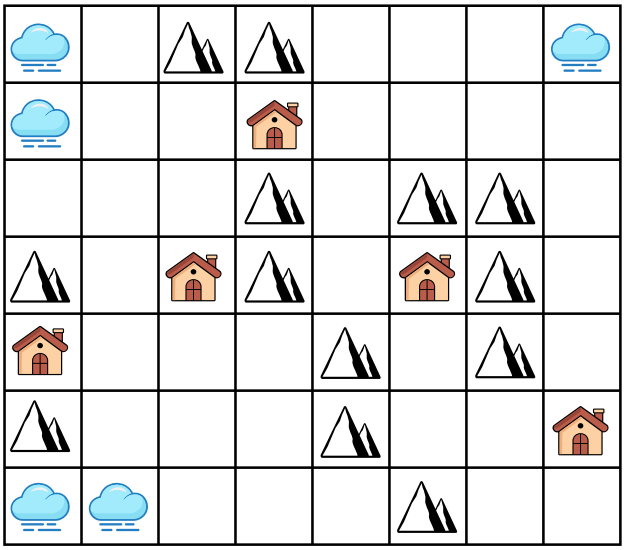
All'alba, la nebbia copre solo alcune regioni. Dopo un'ora, la nebbia si diffonde da ogni regione coperta a tutte le regioni vicine, a destra, a sinistra, in alto o in basso. Anche le case vengono coperte dalla nebbia. Solo le regioni di montagna non possono essere coperte dalla nebbia.

Ecco un esempio:



Alba | Dopo 1 ora | Dopo 2 ore

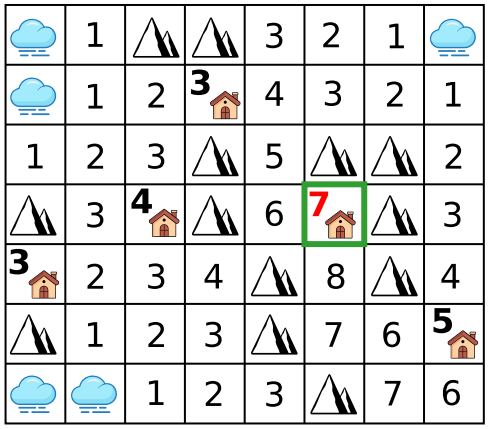
*Quale casa del paese è l'**ultima** ad essere coperta dalla nebbia?*



Soluzione

La nebbia si diffonde a macchia d'olio in tutto il territorio: prima nelle regioni limitrofe delle regioni nebbiose, poi nei vicini dei vicini (che non sono ancora nebbiosi), e così via. Tutto ciò che si deve fare è osservare finché tutte le case, tranne una, non sono scomparse sotto la nebbia. A quel punto si trova la risposta giusta.

Nell'immagine sottostante, la regione della casa che è stata l'ultima a essere coperta dalla nebbia è contrassegnata in verde. Inoltre, per ogni regione è indicato il numero di ore necessarie per essere coperta dalla nebbia. Si può notare che la casa contrassegnata ha il numero più alto di tutte le case, e c'è solo una casa con questo numero, quindi la risposta è chiara, quella è l'ultima casa ad essere coperta dalla nebbia dopo 7 ore dall'alba.



Si potrebbe anche vedere il problema da un altro punto di vista: l'ultima casa a essere coperta dalla nebbia è quella più lontana da una regione nebbiosa all'alba. È quindi possibile misurare questa distanza per tutte le case per determinare così la risposta corretta. Attenzione però, la distanza tra una casa e la nebbia viene misurata lungo la diffusione della nebbia, sulle regioni vicine non diagonali e passando attorno alle regioni montuose. Questa procedura potrebbe richiedere molto più tempo di quella descritta sopra, perché si dovrebbero calcolare $n \times m$ distanze per n case e m regioni di nebbia originali.

È interessante notare che un occhio umano veloce può essere ingannato in questo caso: A prima vista, si potrebbe pensare che la casa in basso a destra sia la più lontana da tutte le regioni di nebbia originali. Tuttavia, poiché non ci sono montagne che ostacolano la nebbia sulla strada verso questa casa, essa viene raggiunta più rapidamente rispetto alla casa della risposta corretta.

Questa è l'informatica!

La nebbia in questo compito si espande gradualmente sulle regioni del paese che possono essere raggiunte da almeno una delle regioni di nebbia originali lungo il percorso di propagazione definito. Un'area composta da tutte le regioni che possono essere raggiunte da una singola regione di nebbia può anche essere chiamata un'area *connessa*. Sulla mappa di questo compito tutte le regioni formano

un'unica area connessa. Una singola montagna aggiuntiva nella seconda fila dall'alto, quarto campo da destra, garantirebbe che le regioni siano divise in due aree di nebbia connessa.

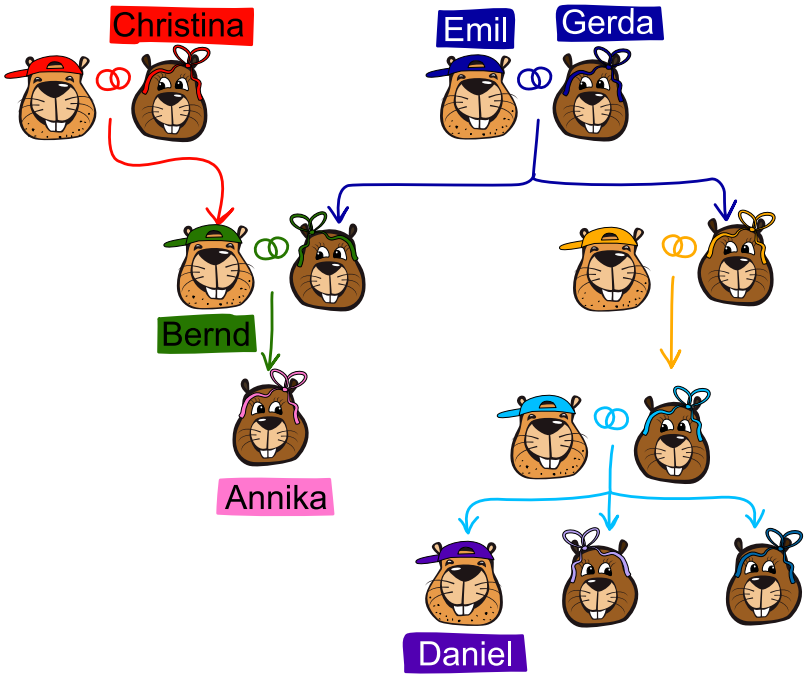
Le aree conesse sono interessanti anche per l'informatica, e in ambiti diversi. Un'area monocromatica in un'immagine (informatica) è un'area connessa di pixel dello stesso colore; può essere determinata utilizzando un algoritmo di riempimento, che funziona in modo simile alla propagazione della nebbia in questo compito. Un gruppo di adolescenti nel quale tutti sono amici di almeno un altro adolescente del gruppo è anch'esso un'area connessa. In modo molto simile, le «bolle» di una rete sociale (o social network) possono essere considerate aree connesse. L'informatica conosce anche metodi per determinare le aree connesse di tali reti, tramite algoritmi come la ricerca breadth-first o la ricerca depth-first. I metodi per determinare le aree connesse possono essere utilizzati, ad esempio, per ricolorare le aree nelle immagini o per determinare i raggruppamenti nelle reti sociali.

Parole chiave e siti web

- Flooding: `https://it.wikipedia.org/wiki/Flooding`
- Flooding (versione Wikipedia inglese con maggiori dettagli): `https://en.wikipedia.org/wiki/Flooding_algorithm`
- Algoritmo flood fill: `https://it.wikipedia.org/wiki/Algoritmo_flood_fill`

21. Albero genealogico

I castori Annika e Daniel possiedono un albero genealogico della loro famiglia. Sull'albero genealogico, i castori maschi indossano un berretto e le femmine un fiocco.



Annika utilizza una notazione abbreviata per descrivere i rapporti tra genitori e figli:

- `Padre(X)` sta per «Padre del castoro X».
- `Madre(X)` sta per «Madre del Castoro X».

Per esempio, il padre di Annika è Bernd e la madre di Bernd è Christina. Annika descrive questo rapporto con l'aiuto di due equazioni:

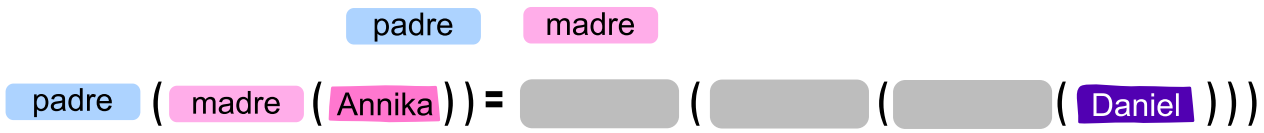
- `Padre(Annika) = Bernd`
- `Madre(Bernd) = Christina`

Annika può anche descrivere il suo rapporto con Christina con una sola equazione:

- `Madre(Padre(Annika)) = Christina` sta per «La madre del padre di Annika è Christina».

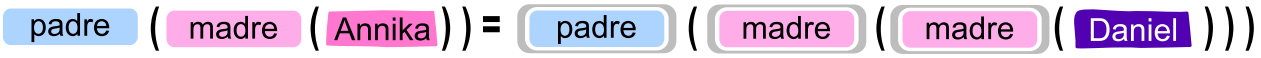
Ora vorrebbe avere un'equazione per la sua relazione con Daniel.

Completa la seguente equazione in modo che descriva la relazione tra Annika e Daniel.



Soluzione

Questa è la risposta:



Per prima cosa osserviamo il lato sinistro dell'equazione e scopriamo che Emil è il padre della madre di Annika, quindi: `Padre(Madre(Annika)) = Emil`. Per riempire gli spazi vuoti sul lato destro, dobbiamo scoprire come Daniel è imparentato con Emil. Per farlo, guardiamo l'albero genealogico e procediamo passo dopo passo da Daniel verso Emil:

1. La madre di Daniel, cioè `Madre(Daniel)`, è imparentata con Emil; il padre di Daniel no.
2. La madre della madre di Daniel, cioè la nonna di Daniel o `Madre(Madre(Daniel))`, è imparentata con Emil, perché ...
3. ... Emil è il padre della nonna: `Emil = Padre(Madre(Madre(Daniel)))`.

Questa è l'informatica!

Annika usa la sua notazione abbreviata per le relazioni tra padre e madre nell'albero genealogico, cioè `padre()` e `madre()` come *funzioni* matematiche che hanno un *valore* (un castoro genitore) per un *argomento* (un castoro come Annika o Daniel). Anche in informatica esistono funzioni, come in matematica. Una funzione implementa una procedura e può essere richiamata con uno o più argomenti (in informatica spesso chiamati anche *parametri*) e restituisce un valore come risultato dopo l'esecuzione del codice.

Questo compito mostra come le chiamate di funzione possano essere combinate (o «annidate») per ottenere calcoli più complessi. In una chiamata di funzione *nidificata*, i risultati delle chiamate di funzioni interne servono come input per le chiamate esterne. Una chiamata di funzione annidata viene valutata dall'interno verso l'esterno. Nel nostro compito, quando si valuta `Padre(Madre(Madre(Daniel)))`, si determina prima la madre di Daniel, poi la madre di sua madre e infine il padre della madre di sua madre. La *composizione di funzioni* o *annidamento* è disponibile nella maggior parte dei linguaggi di programmazione, ma è particolarmente importante per i cosiddetti *linguaggi di programmazione funzionali*.

Parole chiave e siti web

- Funzione: `https://it.wikipedia.org/wiki/Funzione_(informatica)`
- Annidamento: `https://it.wikipedia.org/wiki/Annidamento_(informatica)`
- Funzioni annidate: `https://en.wikipedia.org/wiki/Function_composition_(computer_science)`
- Programmazione funzionale: `https://it.wikipedia.org/wiki/Programmazione_funzionale`

22. Servizio di corriere

Un'immagine segreta composta da pixel neri ■ e bianchi □ deve essere trasmessa in modo sicuro. A tal fine, il corriere scompone l'immagine in due immagini composte da pixel scuri ▨ e chiari ▨ su fogli trasparenti. L'immagine segreta diventa riconoscibile solo quando i due fogli trasparenti vengono sovrapposti.

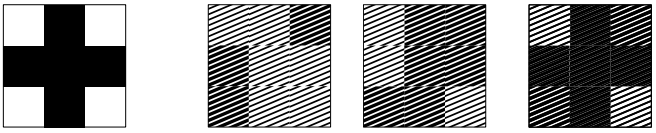


Immagine segreta | Foglio 1 | Foglio 2 | Fogli 1 e 2 sovrapposti

Le immagini per i due fogli vengono create come segue: per prima cosa, per il foglio 1 viene creato un modello casuale di pixel scuri ▨ e chiari ▨. I pixel dell'immagine per il foglio 2 vengono quindi definiti secondo la seguente regola, in base ai pixel che si trovano nella stessa posizione nell'immagine segreta e nel foglio 1:

- Se il pixel dell'immagine segreta è nero ■, allora i pixel dei fogli 1 e 2 devono essere diversi (uno scuro ▨, l'altro chiaro ▨).
- Se il pixel dell'immagine segreta è bianco □, allora i pixel dei fogli 1 e 2 devono essere uguali (entrambi ▨ o entrambi ▨).

Il foglio 1 è già stato creato per la seguente immagine segreta. Bisogna ora creare il foglio 2.

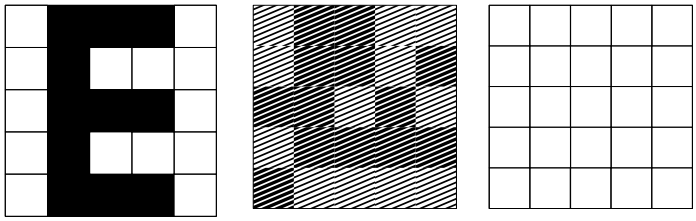
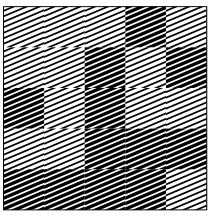


Immagine segreta | *Foglio 1* | *Foglio 2*

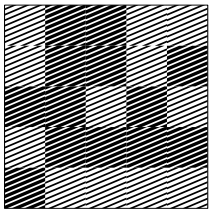
Soluzione

Questa è la soluzione.

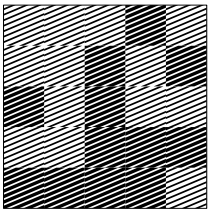


In questa immagine per il foglio 2, ogni pixel è stato impostato secondo la regola descritta sopra - in base all'immagine segreta e al foglio 1. L'immagine differisce da quella del foglio 1 solo nei punti esatti in cui l'immagine segreta presenta pixel neri.

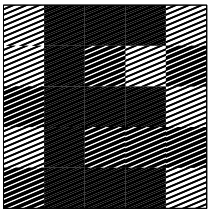
Qui si può vedere come l'immagine segreta possa essere riconosciuta sovrapponendo i fogli 1 e 2:



Foglio 1



Foglio 2



Fogli 1 e 2 sovrapposti

Questa è l'informatica!

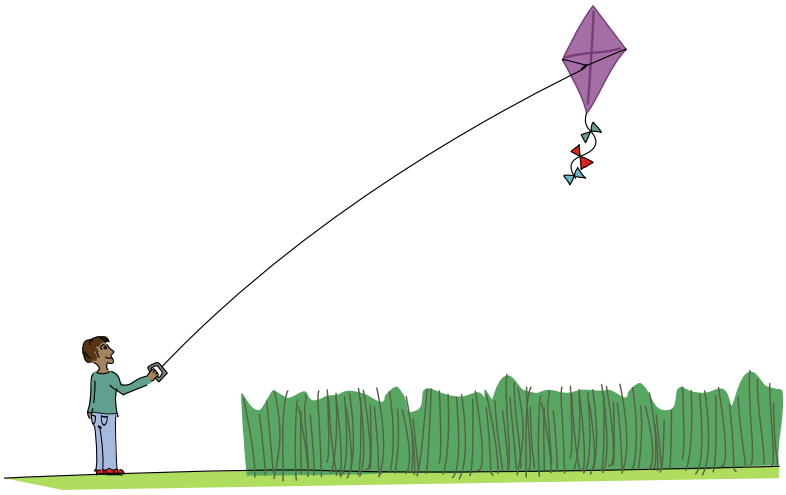
Il servizio di corriere utilizza un *processo crittografico* che si basa sulla percezione visiva ed è quindi chiamato *crittografia visuale*. Tale tecnica è stata sviluppata nel 1994 dagli scienziati israeliani Moni Naor e Adi Shamir. Il metodo risulta essere molto sicuro per quanto riguarda i singoli fogli. Essendo basato su una matrice casuale di pixel, non è possibile ottenere informazioni da ogni singolo foglio, nemmeno con l'aiuto del computer. La decrittazione è possibile - e anche piuttosto semplice - solo se sono presenti entrambi i fogli.

Per trasmettere messaggi segreti, un foglio 1 casuale ma fisso potrebbe essere memorizzato come «chiave» sia dal mittente che dal destinatario. In questo modo, per ogni nuovo messaggio dovrebbe essere generato e trasmesso soltanto il foglio 2. Tuttavia, se la chiave, cioè il foglio 1, viene utilizzata più volte, la procedura non è più completamente sicura. Questo problema si presenta generalmente con la crittografia che funziona secondo il principio del *one-time pad*. La chiave deve essere lunga (almeno) quanto il messaggio segreto e deve essere generata in modo casuale. La crittografia è generata da una combinazione reversibile dei caratteri corrispondenti del messaggio e della chiave. In informatica, l'operazione XOR è generalmente utilizzata come combinazione reversibile per i messaggi costituiti da bit che i computer si scambiano tra loro. La combinazione di pixel chiari e scuri in questo compito corrisponde esattamente a questa operazione.

Parole chiave e siti web

- Crittografia visuale: `https://it.wikipedia.org/wiki/Crittografia_visuale`

23. L'aquilone perduto

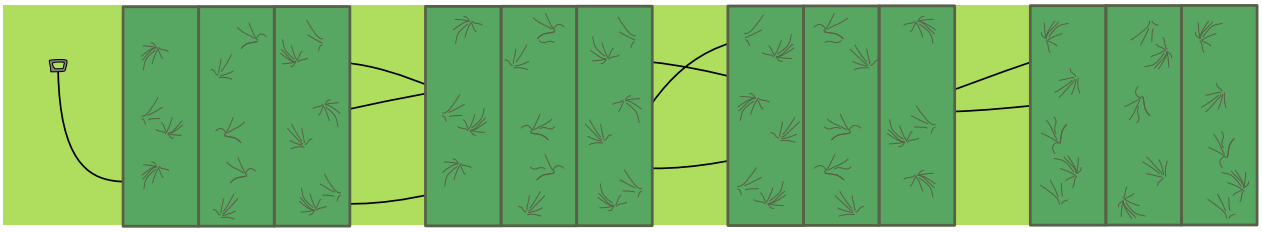


Che sfortuna! Asterios ha perso il suo aquilone nel prato. La corda dell'aquilone si è impigliata nell'erba alta e Asterios non riesce a ritrovarlo.

Il prato è diviso in 15 aree che possono essere ispezionati singolarmente.

Asterios ha già cercato in 3 aree del prato. Osservando attentamente come la corda attraversa queste aree, Asterios si rende conto che ora deve cercare solo in un'altra area per sapere con certezza dove si trova l'aquilone.

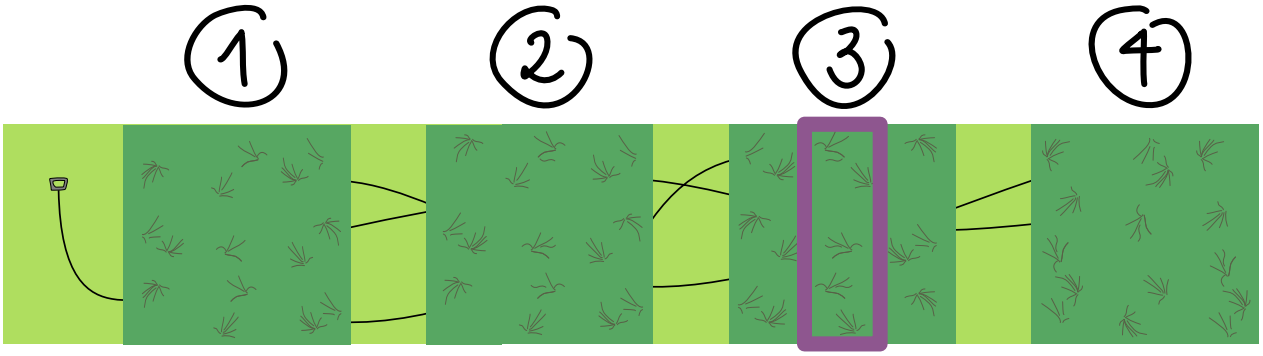
Di quale area si tratta?



Soluzione

Questa è la risposta:

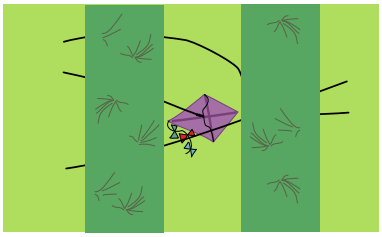
Asterios deve cercare nel campo evidenziato in viola per scoprire con certezza dove si trova l'aquilone.



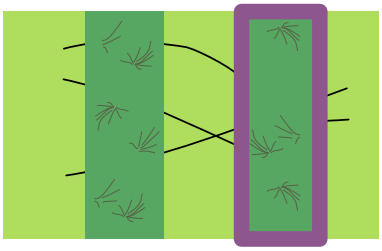
La soluzione si può trovare in due fasi. Per prima cosa pensate a quale dei quattro blocchi 1, 2, 3 e 4 deve trovarsi l'aquilone. Ricordate che la corda dell'aquilone inizia a sinistra e l'aquilone è appeso all'altra estremità della corda.

L'aquilone non può trovarsi nel blocco 4 perché la corda dell'aquilone entra ed esce dal blocco 4. L'aquilone deve trovarsi a destra del blocco 2 perché si vedono tre segmenti di corda tra il blocco 2 e il blocco 3. Due segmenti di corda devono appartenere a un anello (formato dalla corda stessa) nella parte destra del prato, mentre uno conduce all'aquilone.

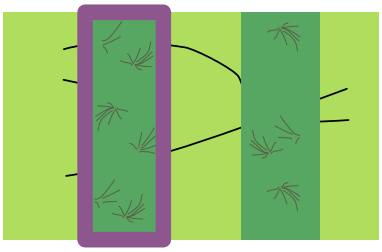
Quando Asterios cerca il campo centrale nel blocco 3, si possono distinguere tre casi:



Caso 1: l'aquilone si trova nel campo. Fantastico! L'aquilone è stato trovato e la ricerca è terminata.



Caso 2: nel campo non si vede l'aquilone, ma Asterios trova tre segmenti di corda che conducono al campo destro. Poiché da questo campo solo due segmenti di corda conducono a destra, l'aquilone deve trovarsi nel campo destro.



Caso 3: nel campo non si vede l'aquilone, ma Asterios trova due segmenti di corda che vanno dal campo di sinistra a quello di destra. Quindi l'aquilone deve trovarsi a sinistra di questo campo. Infatti, i due segmenti di corda appartengono a un anello nella parte destra del prato.

In ogni caso, sappiamo in quale casella si trova l'aquilone dopo aver cercato nella casella viola.

Questa è l'informatica!

Quando si cerca sistematicamente tra i campi, si effettua una ricerca deterministica: ogni nuova informazione - le osservazioni su un campo cercato - permette di trarre conclusioni specifiche sui campi vicini. Una proprietà *topologica* della corda dell'aquilone gioca un ruolo importante in questo caso: la corda forma anelli chiusi, e ogni area delimitata da due linee rette (come i campi in questo compito) contiene o un numero pari di segmenti o il punto di svolta di un anello.

Le proprietà topologiche descrivono le strutture create dalla disposizione e dalla connessione degli elementi, indipendentemente dalle dimensioni, dalle distanze o dagli angoli. Non si tratta quindi di quanto sia grande o lungo qualcosa, ma di come le cose sono collegate tra loro e di quanti percorsi o passaggi ci sono.

La ricerca sistematica con interpretazione topologica non è solo un appassionante gioco mentale, ma ha anche un'importanza pratica nell'informatica. Ad esempio, una rete stradale può essere vista come un sistema di punti e connessioni, come incroci e strade. Questa struttura aiuta gli algoritmi di ricerca a trovare percorsi mirati, a riconoscere le deviazioni e a mostrare come raggiungere la destinazione il più rapidamente possibile.

Anche la struttura topologica delle reti elettriche fornisce indizi cruciali per la risoluzione dei problemi: circuiti paralleli o interruzioni mostrano spesso dove potrebbe trovarsi il guasto, senza alcuna misurazione precisa, solo la struttura logica della rete.

Parole chiave e siti web

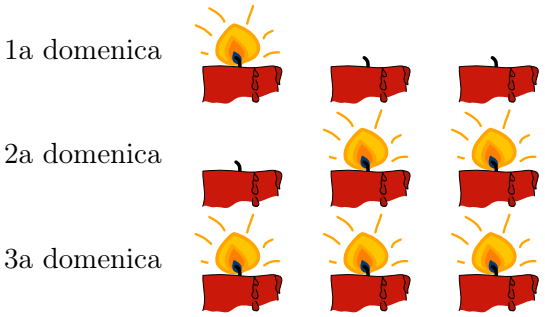
- Topologia: `https://it.wikipedia.org/wiki/Topologia`
- Algoritmo di ricerca: `https://it.wikipedia.org/wiki/Algoritmo_di_ricerca`

24. Corona dell'Avvento

La tradizione vuole che si accendano delle candele nelle quattro domeniche che precedono il Natale: 1 candela la prima domenica, 2 candele la seconda domenica, e così via.

Chris ama questa tradizione e possiede quattro candele, tutte della stessa lunghezza. A Chris piacerebbe molto che le quattro candele fossero ancora tutte della stessa lunghezza dopo l'ultima domenica, ma dovrebbe accendere ogni candela lo stesso numero di volte, cosa non possibile in quattro domeniche.

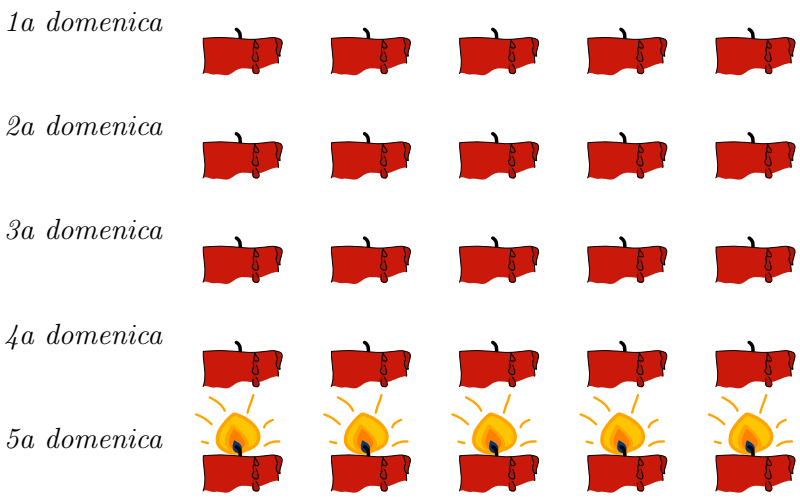
Se la tradizione prevedesse solo tre domeniche (e candele) ciò sarebbe invece possibile, perché Chris accenderebbe ogni candela esattamente due volte:



Chris crede che ciò sarebbe possibile anche con cinque domeniche (e candele).

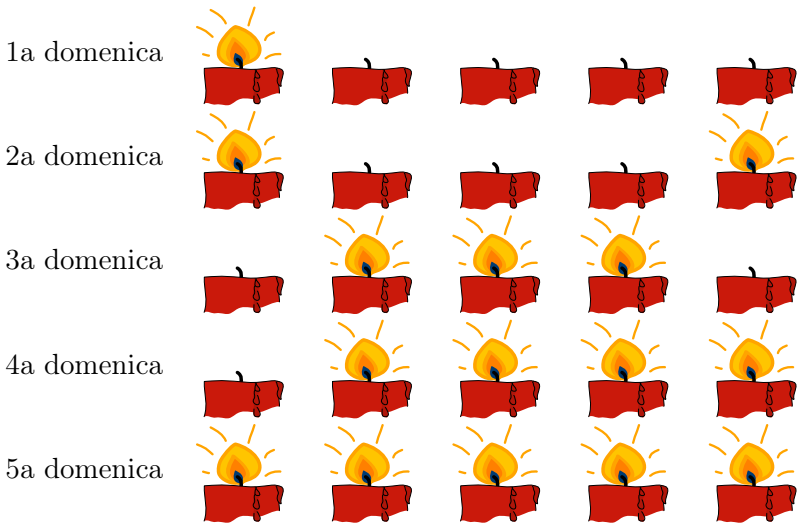
Mostra a Chris come accendere ogni candela lo stesso numero di volte.

Abbiamo già acceso le candele per la quinta domenica.



Soluzione

Chris può accendere le candele in questo modo, ognuna esattamente tre volte:



Esistono diversi modi per ottenere questo risultato. In ognuno di questi casi si può procedere come segue:

1. Raggruppare le domeniche in coppie i cui numeri sommati formano il numero totale di domeniche; per cinque domeniche, si tratta delle coppie 1 e 4, e 2 e 3.
2. per ciascuna di queste coppie, le candele vengono accese in modo tale che i due gruppi di candele accese nelle rispettive domeniche siano disgiunti - ad esempio, la candela 1 (da sinistra) la domenica 1 e le candele da 2 a 5 la domenica 4 (se ogni candela viene considerata come un bit con 1 = acceso, 0 = non acceso, allora le due sequenze di bit delle candele delle domeniche devono essere complementari tra loro per ciascuna coppia di domeniche). Ciò significa che ogni candela si accende esattamente una volta nelle due domeniche della coppia.
3. Inoltre, ogni candela viene accesa ancora una volta l'ultima domenica (domenica 5).

Così facendo, ogni candela viene accesa lo stesso numero di volte. Di conseguenza, la quinta domenica le candele saranno tutte della stessa lunghezza.

Questa è l'informatica!

A prima vista, questo problema sembra essere un problema piuttosto matematico. In realtà, esiste una prova che dimostra che il problema delle candele di Chris è risolvibile per qualsiasi numero dispari di domeniche. La strategia presentata nella spiegazione della soluzione infatti funziona per i numeri dispari di domeniche.

Tuttavia, se si vuole scoprire nello specifico come accendere le candele, è necessario descrivere un algoritmo che specifichi la sequenza di accensione - o meglio ancora: che enumeri tutte le possibili soluzioni. Questo algoritmo si basa sulla spiegazione precedente: sia n il numero (dispari) di domeniche:

1. L'ennesima domenica: accendere tutte le n candele.
2. per $i = 1$ a $(n-1)/2$:
   a) Accendere le prime i candele la domenica i e le ultime $n-i$ candele la domenica $n-i$.

Il passo 2a di questo algoritmo può essere eseguito in tutti i modi che soddisfano la condizione citata nella spiegazione al punto 2.

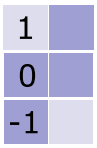
Lo sviluppo di algoritmi per risolvere i problemi è uno dei compiti più importanti degli informatici. Se un algoritmo si basa su una solida modellazione matematica, è più facile dimostrarne le proprietà desiderate che non senza tale modellazione. Per l'algoritmo di cui sopra, è possibile dimostrare che funziona per qualsiasi numero dispari di domeniche.

Parole chiave e siti web

- Algoritmi: `https://it.wikipedia.org/wiki/Algoritmo`

25. Mappa di luminosità

Le immagini digitali sono spesso costituite da pixel. Sandra vuole creare delle mappe di luminosità per queste immagini composte da pixel. Per farlo, inserisce prima una cornice di pixel bianchi attorno all'immagine. Quindi determina un valore di luminosità per ogni pixel dell'immagine secondo il seguente metodo:

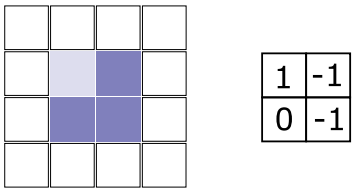


1, se il pixel è più chiaro del pixel adiacente alla sua destra.

0, se il pixel ha la stessa luminosità del pixel adiacente a destra.

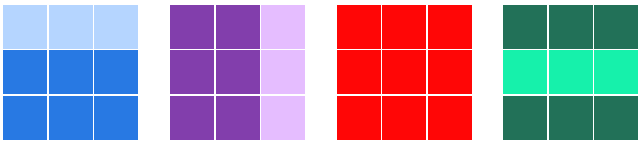
-1, se il pixel è più scuro del pixel adiacente alla sua destra.

Qui si può vedere un'immagine composta da quattro pixel (più i pixel bianchi della cornice) e la relativa mappa di luminosità.



Di seguito sono riportate quattro immagini con nove pixel ciascuna. Tre di esse hanno la stessa mappa di luminosità.

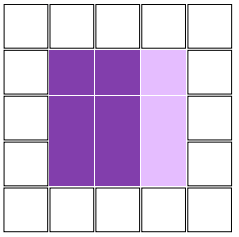
Quale delle immagini è l'unica con una mappa di luminosità ***diversa****?*



A) B) C) D)

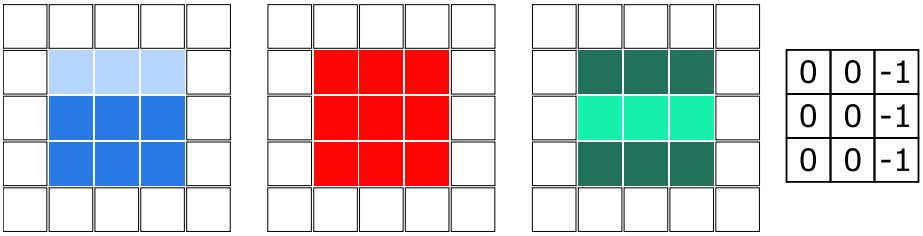
Soluzione

La risposta è l'immagine B:



0	-1	-1
0	-1	-1
0	-1	-1

Le altre tre immagini hanno tutte la stessa mappa di luminosità:



0	0	-1
0	0	-1
0	0	-1

Per trovare la risposta, si possono calcolare le mappe di luminosità di tutte e quattro le immagini e confrontarle. Oppure si può stabilire che una mappa di luminosità registra solo le differenze di luminosità in direzione orizzontale. Poiché le tre immagini nelle risposte A, C e D non presentano differenze di luminosità in direzione orizzontale, le loro mappe di luminosità devono essere uguali.

Questa è l'informatica!

L'informatica utilizza molti formati diversi per salvare immagini sui computer. Tutti questi formati utilizzano fondamentalmente due metodologie rappresentazione grafica: la *grafica vettoriale* o la grafica *grafica raster*. In quest'ultimo caso, i singoli punti raster sono chiamati anche *pixel* (abbreviazione di: picture element). Nel caso più semplice, ogni pixel può essere nero o bianco e può quindi essere rappresentato da un singolo bit con i valori 0 o 1. I pixel colorati sono descritti da diversi valori che, ad esempio, indicano la proporzione dei colori rosso, verde e blu nel colore finale del pixel.

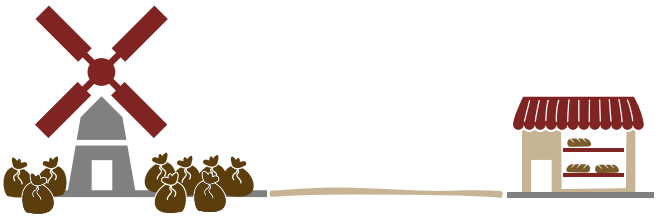
La mappa di luminosità in questo compito è simile al concetto di *convoluzione* tra l'immagine e un *filtro*. A seconda del filtro, diverse proprietà strutturali dell'immagine possono essere rese riconoscibili da tale convoluzione, come angoli, bordi o aree di luminosità uniforme. In questo modo il computer può interpretare più facilmente le informazioni contenute nell'immagine.

Le cosiddette *Reti neurali convoluzionali* (CNN), spesso componenti centrali dei sistemi di intelligenza artificiale, utilizzano il concetto di convoluzione per il riconoscimento delle immagini. Per riconoscere oggetti complessi in un'immagine, una CNN impara a sviluppare da sola i filtri adatti, che utilizza per convolvere l'immagine. Ciò consente di distinguere tra immagini di cani e gatti, ad esempio, o di rilevare tumori nelle scansioni mediche.

Parole chiave e siti web

- Visione artificiale: `https://it.wikipedia.org/wiki/Visione_artificiale`
- Pixel: `https://it.wikipedia.org/wiki/Pixel`
- Convoluzione: `https://it.wikipedia.org/wiki/Convoluzione`
- Matrice di convoluzione: `https://it.wikipedia.org/wiki/Matrice_di_convoluzione`
- Rete neurale convoluzionale:
  `https://de.wikipedia.org/wiki/Convolutional_Neural_Network`

26. Trasporto farina



Albert e Marco lavorano in una panetteria. Quando la farina in panetteria finisce, bisogna andare a recuperarla al mulino. Solo una persona alla volta può uscire per consentire all'altra di servire i clienti.

Le loro prestazioni nel recupero della farina variano:

Albert raccoglie 13 kg di farina in un'ora.

Marco raccoglie 5 kg di farina in 30 minuti.

Per entrambi vale la seguente regola: dopo tre viaggi al mulino, sono necessari 30 minuti di riposo. I clienti possono essere serviti durante questo periodo di riposo.

Albert e Marco vogliono recuperare più farina possibile in 8 ore. Possono farlo esattamente in una delle seguenti condizioni. In quale?

A) Albert deve andare per primo.
B) Marco deve andare per primo.
C) Marco deve andare per ultimo.
D) Albert non deve andare per ultimo.
E) Marco deve andare esattamente una volta.

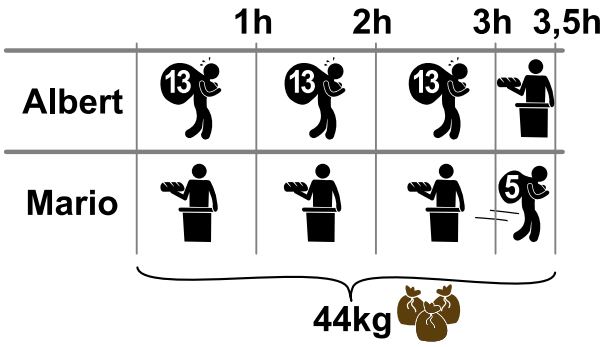
Soluzione

La risposta A è corretta.

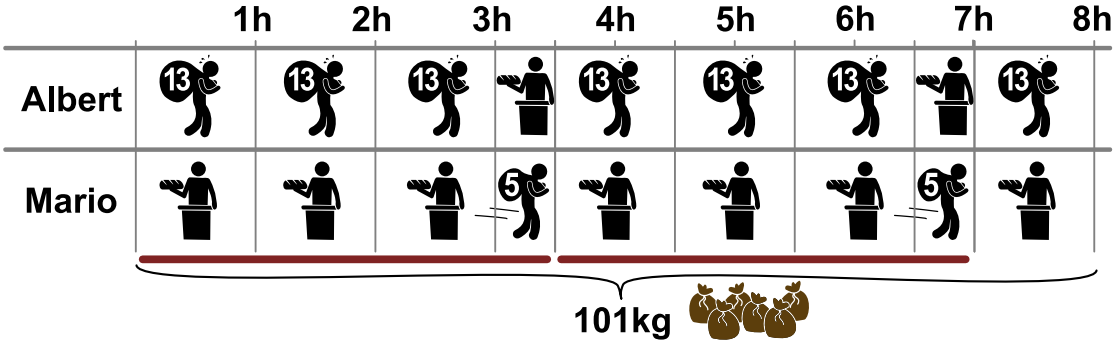
Sappiamo che:

- Albert recupera 13 kg all'ora.
- Marco recupera 5 kg in 30 minuti, cioè 10 kg all'ora.
- Qualcuno deve sempre rimanere in panetteria.
- Tutti possono andare al massimo tre volte di seguito, dopodiché devono rimanere in panetteria per 30 minuti.
- Mentre uno di loro rimane nella panetteria, l'altro può andare a recuperare altra farina.

Albert riesce a recuperare più farina all'ora di Marco. Per questo motivo dovrebbe andare il più possibile. Marco cammina solo quando Albert si riprende  dopo aver camminato tre volte! In questo modo, possono raccogliere un massimo di 44 kg di farina in 3,5 ore:



Siccome Albert nel frattempo si è ripreso, possono ripetere questo schema (breve: A, A, A, M) e ottenere 88 kg di farina in 7 ore. Poi Albert può ripartire per un ultimo viaggio. In totale, possono raccogliere un massimo di 101 kg di farina in 8 ore se Albert va per primo.



Risposta B: Anche se Marco va per primo, possono ottenere 44 kg di farina in 3,5 ore, ma questa volta seguendo lo schema (M, A, A, A). Dopo due ripetizioni e 88 kg di farina, rimane ancora un'ora. Poiché Albert deve recuperare, Marco potrebbe andare due volte durante quest'ultima ora e recuperare 10 kg di farina. In questa situazione però è possibile raccogliere meno farina di prima, cioè $44 + 44 + 10 = 98$ kg.

Risposta C e D: Se Marco va per ultimo, Albert non può andare per ultimo. Quindi le risposte C e D sono uguali. Anche in queste condizioni, possono raccogliere 88 kg di farina in 7 ore, ma nell'ultima ora possono raccogliere solo 10 kg, quindi in totale solo 44 + 44 + 10 = 98 kg.

Risposta E: Se Marco va esattamente una volta, possono raccogliere di nuovo 44 kg di farina nelle prime 3,5 ore. Nelle seconde 3,5 ore, tuttavia, Albert deve riprendersi per 30 minuti e, poiché Marco non può andare di nuovo, in questo lasso di tempo raccolgono solo 39 kg. Dopodiché, Albert può andare di nuovo, quindi in questa condizione possono raccogliere solo 44 + 39 + 13 = 96 kg di farina.

Questa è l'informatica!

Se Albert e Marco vogliono recuperare la maggior quantità di farina possibile in un certo tempo, devono risolvere un problema di pianificazione e ottimizzazione:

- Devono pianificare i viaggi e rispettare le restrizioni (periodi di riposo e limitazioni di occupazione in panetteria).
- Devono suddividersi i viaggi.
- Si vuole massimizzare la quantità totale di farina recuperata (obiettivo di ottimizzazione).

I problemi di pianificazione e ottimizzazione si presentano in molti ambiti della vita. L'obiettivo più importante è l'*efficienza*; ovvero ottenere il massimo possibile in determinate condizioni. Negli impianti di produzione si dovrebbe produrre il più possibile con le persone e le macchine disponibili, in un aeroporto si dovrebbe gestire il maggior numero possibile di voli e passeggeri ai banchi di check-in e alle porte d'imbarco disponibili, e così via. Quando i problemi di pianificazione e ottimizzazione diventano più grandi, i programmi informatici aiutano a risolverli grazie ad algoritmi conosciuti nel campo dell'informatica. Tali problemi, per i quali devono essere rispettate delle condizioni, si presentano anche all'interno dei processori dei computer, perché anch'essi devono essere efficienti. Per esempio, quando si eseguono le istruzioni, bisogna tenere conto del fatto che le unità logiche aritmetiche possono eseguire solo un'istruzione alla volta (restrizione di allocazione). I comandi che leggono i dati dalla memoria devono attendere l'arrivo dei dati e quindi avere dei «tempi morti», come Albert e Marco dopo aver camminato tre volte in questo compito.

Parole chiave e siti web

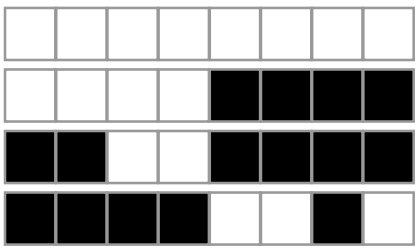
- Problema di ottimizzazione: `https://it.wikipedia.org/wiki/Problema_di_ottimizzazione`
- Scheduler: `https://it.wikipedia.org/wiki/Scheduler`

27. Nero e bianco

Sarah vuole descrivere sequenze di caselle bianche e nere con delle lettere. Per farlo, applica questo algoritmo alla sequenza di caselle:

- Se tutte le caselle della sequenza sono bianche, scrive `B`.
- Se tutte le caselle della sequenza sono nere, scrive `N`.
- Se la sequenza contiene caselle bianche e nere, scrive `x` e procede come segue:
  - Applica l'algoritmo alla metà sinistra della sequenza.
  - Applica l'algoritmo alla metà destra della sequenza.

Qui si può vedere la sequenza di lettere che l'algoritmo produce per alcune sequenze di caselle:



`B`

`xBN`

`xxNBN`

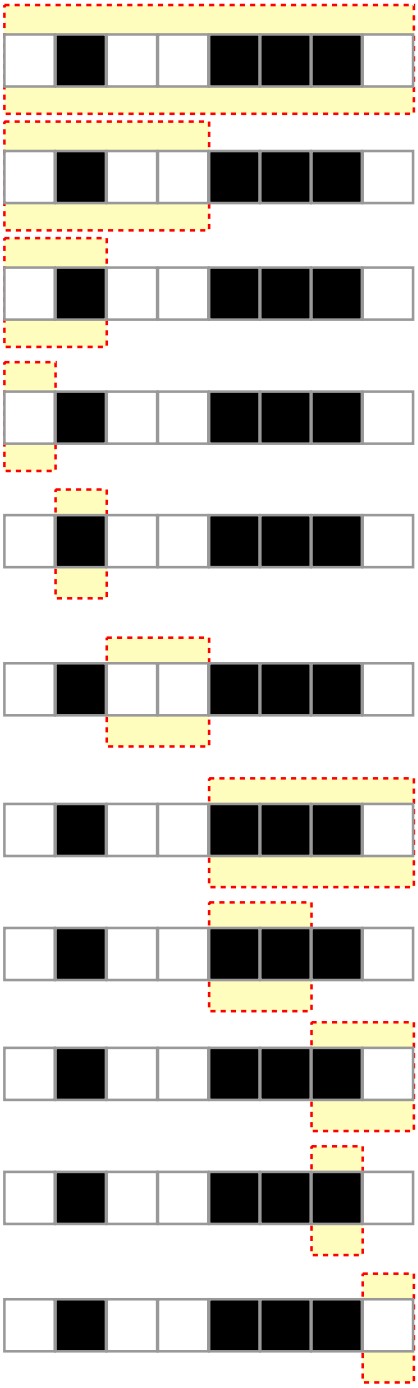
`xNxBxNB`

Come viene rappresentata questa sequenza di caselle seguendo l'algoritmo di Sarah?

Soluzione

Questa è la rispota corretta: `xxxBNBxNxNB`.

Applichiamo l'algoritmo alla sequenza di caselle e costruiamo la sequenza delle lettere passo dopo passo. Nelle immagini, la sequenza di caselle a cui si sta applicando l'algoritmo è evidenziata in giallo.



`x` - La sequenza contiene caselle bianche e nere, quindi l'algoritmo scrive `x` e si applica alla metà sinistra della sequenza.

`xx` - La sequenza contiene caselle bianche e nere, quindi l'algoritmo scrive `x` e si applica alla metà sinistra della sequenza.

`xxx` - La sequenza contiene caselle bianche e nere, quindi l'algoritmo scrive `x` e si applica alla metà sinistra della sequenza.

`xxxB` – Tutte le caselle nella sequenza sono bianche, quindi l'algoritmo scrive `B`. L'algoritmo termina per questa sequenza e si applica alla metà destra della sequenza precedente.

`xxxBN` - Tutte le caselle nella sequenza sono nere, quindi l'algoritmo scrive `N`. L'algoritmo termina per questa sequenza e si applica alla metà destra della sequenza precedente.

`xxxBNB` - Tutte le caselle nella sequenza sono bianche, quindi l'algoritmo scrive `B`. Ora abbiamo processato la metà sinistra della sequenza totale e possiamo contuniare con la metà destra della sequenza totale.

`xxxBNBx` - La sequenza contiene caselle bianche e nere, quindi l'algoritmo scrive `x` e si applica alla metà sinistra della sequenza.

`xxxBNBxN` - Tutte le caselle nella sequenza sono nere, quindi l'algoritmo scrive `N`. L'algoritmo termina per questa sequenza e si applica alla metà destra della sequenza precedente.

`xxxBNBxNx` - La sequenza contiene caselle bianche e nere, quindi l'algoritmo scrive `x` e si applica alla metà sinistra della sequenza.

`xxxBNBxNxN` - Tutte le caselle nella sequenza sono nere, quindi l'algoritmo scrive `N`. L'algoritmo termina per questa sequenza e si applica alla metà destra della sequenza precedente.

`xxxBNBxNxNB` - Tutte le caselle nella sequenza sono bianche, quindi l'algoritmo scrive `B`. Ora abbiamo processato anche la metà destra della sequenza totale e il compito è quindi finito.

Questa è l'informatica!

L'algoritmo di Sarah ha una caratteristica molto particolare: se la sequenza di caselle in ingresso è di colore misto, si applica, per così dire, alla metà sinistra e alla metà destra dell'ingresso, una dopo l'altra. In questo modo, il compito di descrivere l'ingresso come una sequenza di lettere viene suddiviso in due sotto-compiti più piccoli. Questo è utile se, tra le altre cose, i sottocompiti sono più facili da lavorare rispetto all'intero compito. La divisione dei problemi in sottoproblemi (auspicabilmente più facili) è nota in informatica come «divide et impera», in accordo con la strategia nota nell'antica Roma. Se una procedura di soluzione affronta i sottoproblemi nello stesso modo del compito complessivo, cioè si applica ai sottoproblemi e poi in particolare li suddivide in parti più piccole, la procedura segue anche il principio del *algoritmo ricorsivo*, proprio come l'algoritmo di Sarah in questo compito. L'algortimo ricorsivo è usato molto frequentemente in informatica, sia per l'ordinamento dei dati, sia per la costruzione di file system e molto altro ancora.

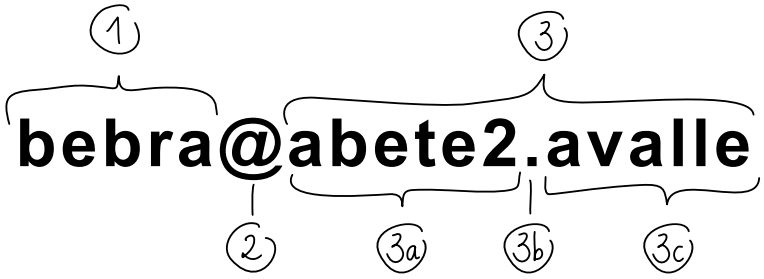
L'algoritmo di Sarah descrive o *codifica* le sequenze di caselle con lettere. Una codifica deve essere reversibile; deve quindi esistere una procedura per riconvertire le lettere nella sequenza originale di caselle. Se la descrizione delle lettere è integrata dal numero di caselle della sequenza descritta, ciò è possibile senza problemi: potrebbe anche essere necessario un algoritmo ricorsivo! Idealmente, la codifica che l'algoritmo di Sarah produce è molto più breve della sequenza di caselle inserita. Ad esempio, una sequenza di 1024 caselle bianche viene descritta con una singola `W`. L'algoritmo di Sarah quindi non esegue solo la codifica, ma anche la *compressione dei dati* (descrizione dei dati con risparmio di spazio), seguendo idee simili ai metodi per comprimere i dati delle immagini (ad esempio nel formato fotografico JPEG) o i dati video.

Parole chiave e siti web

- Algortimo ricorsivo: `https://it.wikipedia.org/wiki/Algoritmo_ricorsivo`
- Compressione dei dati: `https://it.wikipedia.org/wiki/Compressione_dei_dati`
- Albero quadramentale: `https://it.wikipedia.org/wiki/Albero_quadramentale`

28. Riconoscimento degli indirizzi e-mail

I castori informatici hanno bisogno di un sistema che riconosca se una stringa di caratteri è un indirizzo e-mail valido. Un indirizzo e-mail valido è composto da tre parti:



	Parte dell'indirizzo	Spiegazione	Valori possibili
1	**Nome utente**	una stringa arbitraria non vuota di lettere minuscole e cifre	0–9, a-z
2	**Chiocciola**		@
3	**Nome server**		
3a	**Nome del dominio**	una stringa arbitraria non vuota di lettere minuscole e cifre	0–9, a-z
3b	**Punto**		.
3c	**Nome del dominio di primo livello**	una stringa arbitraria non vuota di lettere minuscole e cifre	0–9, a-z

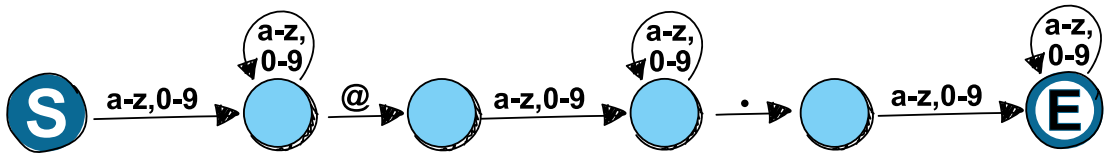
Le stringhe «non vuote» sono composte da almeno un carattere. «bebra@abete2.avalle» è un esempio di indirizzo e-mail valido di un castoro.

I castori informatici pensano a quattro sistemi e li descrivono usando diagrammi di cerchi e frecce. Un cerchio rappresenta uno degli stati in cui il sistema può trovarsi. Una freccia descrive il passaggio allo stato successivo, il quale può essere lo stesso seguendo le frecce che partono e finiscono nello stesso stato. L'etichetta della freccia indica a quale carattere corrisponde questo cambiamento di stato.

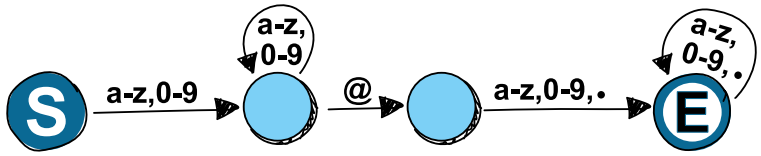
Un sistema esamina una stringa carattere per carattere, da sinistra a destra. Inizialmente si trova nello stato S. Lo stato successivo dipende dal carattere che si sta analizzando: ll sistema segue il cambiamento di stato che corrisponde al carattere che viene letto. Se il sistema si trova nello stato E dopo l'ultimo carattere, ha riconosciuto la stringa di caratteri come un indirizzo di posta elettronica.

Solo uno di questi sistemi riconosce correttamente tutti i possibili indirizzi e-mail validi. Quale?

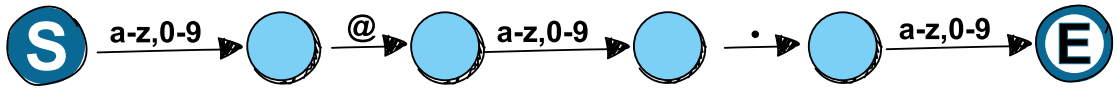
A)



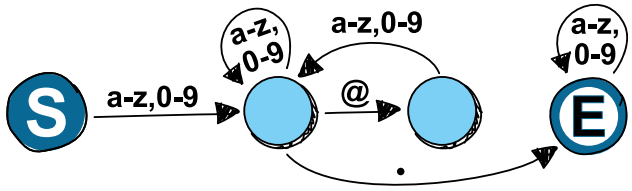
B)



C)

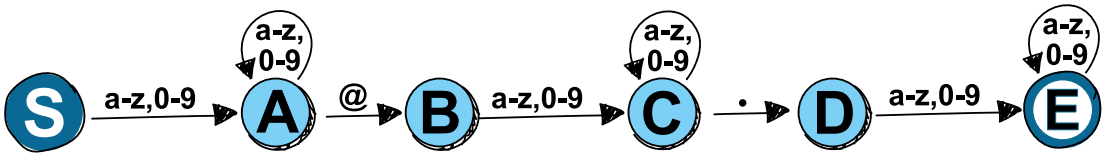


D)



Soluzione

La risposta A è quella corretta. Per analizzarla, etichettiamo tutti glisStati con delle lettere.



Il passaggio (in gergo informatico: transizione) da S ad A riconosce una lettera minuscola o un numero, in modo che il nome utente non possa essere vuoto. Il passaggio da A ad A consente di riconoscere un numero qualsiasi di altre lettere minuscole o cifre. La transizione da A a B riconosce la chiocciola. La transizione da B a C riconosce una lettera minuscola o un numero, in modo che il nome del dominio non possa essere vuoto. Il passaggio da C a C consente di riconoscere un numero qualsiasi di lettere minuscole o cifre aggiuntive. Il passaggio da C a D riconosce un punto. La transizione da D a E riconosce una lettera minuscola o un numero in modo che il nome di dominio di primo livello non possa essere vuoto. Il passaggio da E a E consente di riconoscere qualsiasi altra lettera minuscola o cifra. L'insieme corrisponde esattamente alla definizione degli indirizzi di posta elettronica del compito.

La risposta B non è corretta. Il sistema riconosce anche «castoro@internet» o «castor@.internet.com», ad esempio. Tuttavia, queste stringhe di caratteri non sono considerate indirizzi e-mail valide.

Anche la risposta C non è corretta. Il sistema è in grado di riconoscere solo alcuni indirizzi validi, ovvero solo quelli il cui nome utente, il nome di dominio e il nome di dominio di primo livello sono composti esattamente da una lettera ciascuno.

Anche la risposta D non è corretta. Questo sistema riconosce anche «castoro@abete@avalle.», ad esempio, e questa stringa di caratteri non è un indirizzo e-mail valido.

Questa è l'informatica!

I sistemi utilizzati in questo compito sono gli *automi a stati finiti*. Gli automi a stati finiti sono le cosiddette macchine riconoscitrici, alcuni degli automi più semplici. Sono costituiti da un insieme di stati, di cui uno è lo *stato iniziale* e un sottoinsieme contiene gli *stati finiti*. Se un automa finito si trova in uno stato finale dopo aver esaminato l'intera stringa, la parola esaminata è considerata riconosciuta. L'automa finito determina lo stato successivo con l'aiuto di transizioni che dipendono dai caratteri esaminati.

Per ogni automa finito esiste una *grammatica regolare* corrispondente che può generare esattamente tutte le parole (la corrispondente *lingua regolare*) che l'automa finito riconosce. Analogie simili tra grammatiche e lingue esistono anche per altre macchine riconoscitrici, sistematizzate nella *gerarchia di Chomsky* per i *linguaggi formali*.

I linguaggi riconosciuti dagli automi finiti possono essere descritti anche con espressioni regolari. L'espressione regolare per gli indirizzi di e-mail definiti in questo compito è:

```
[a-z0-9]+@[a-z0-9]+.[a-z0-9]+
```

Gli automi finiti o le espressioni regolari sono utilizzati in molti ambiti per controllare le stringhe di caratteri: se sono un indirizzo e-mail o web formato correttamente, se soddisfano determinati requisiti per una password e molto altro. Grazie alla loro struttura semplice, gli automi finiti vengono utilizzati anche in molti sistemi integrati in cui è necessario consumare pochissima energia, ad esempio perché questi sistemi funzionano solo con una batteria e sono destinati a funzionare per diversi anni.

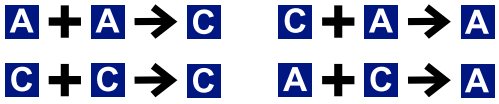
Parole chiave e siti web

- Automa a stati finiti: `https://it.wikipedia.org/wiki/Automa_a_stati_finiti`
- Grammatica regolare: `https://it.wikipedia.org/wiki/Grammatica_regolare`
- Linguaggio regolare: `https://it.wikipedia.org/wiki/Linguaggio_regolare`
- Espressione regolare: `https://it.wikipedia.org/wiki/Espressione_regolare`
- Gerarchia di Chomsky: `https://it.wikipedia.org/wiki/Gerarchia_di_Chomsky`

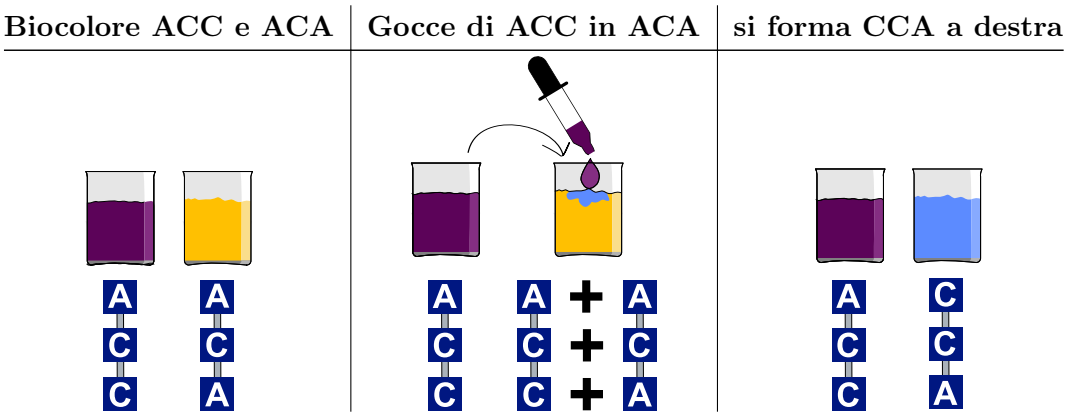
29. Biocolori

Un biocolore è una sostanza chimica speciale: le sue molecole sono ciascuna una sequenza di esattamente tre blocchi di costruzione di tipo A e C. Questa sequenza viene utilizzata anche come nome del biocolore, ad esempio: ACA.

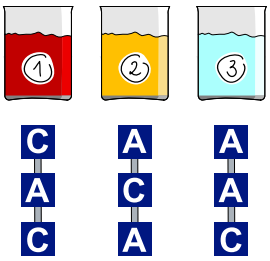
Se un biocolore viene aggiunto a un altro biocolore, viene creato un nuovo biocolore. I blocchi di costruzione del nuovo biocolore vengono creati dai blocchi di costruzione presenti nelle molecole dei due biocolori secondo queste regole:



Esempio: Se si aggiunge una goccia di ACC ad ACA, si ottiene CCA:

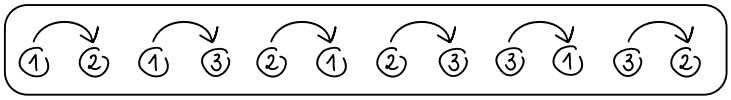


Nel laboratorio ci sono tre contenitori, 1, 2 e 3, con i biocolori CAC, ACA e AAC:



Con un'istruzione di questo tipo si può stabilire che una goccia del biocolore contenuto in un contenitore (ad esempo il contenitore 2) venga aggiunta al biocolore contenuto in un altro contenitore (ad es. contenitore 3).

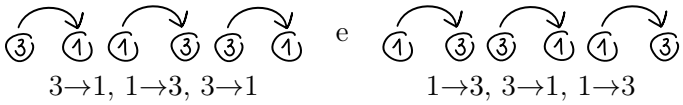
Di seguito sono riportate sei diverse istruzioni. Utilizza il minor numero possibile per scambiare i biocolori nei contenitori 1 e 3: alla fine, il contenitore 1 deve contenere AAC, il contenitore 2 deve contenere ACA e il contenitore 3 deve contenere CAC.



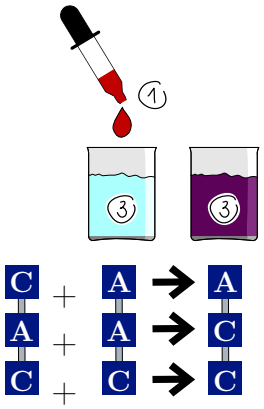
Soluzione

Questa è la risposta:

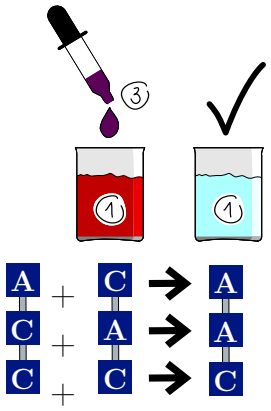
Il biocolore nei contenitori 1 e 3 può essere scambiato con tre istruzioni. Ci sono due risposte corrette:



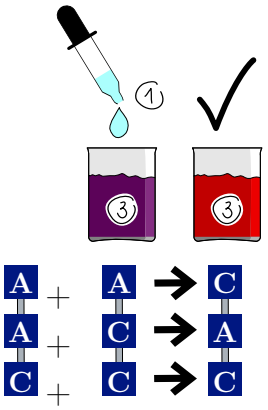
Illustriamo la prima risposta 1→3, 3→1, 1→3. All'inizio, le sostanze nei contenitori sono: 1: CAC, 2: ACA, 3: AAC.



1→3: Nel contenitore 3 si produce CAC + AAC → ACC. Le sostanze nei contenitori sono ora: 1: CAC, 2: ACA, 3: ACC.



3→1: Nel contenitore 1 si forma ACC + CAC → AAC. Le sostanze nei contenitori sono ora: 1: AAC, 2: ACA, 3: ACC.



1→3: Nel contenitore 3 si forma AAC + ACC → CAC. Le sostanze nei contenitori sono ora: 1: AAC, 2: ACA, 3: CAC. Questa è la sequenza desiderata di sostanze.

Ora dimostriamo che non è possibile scambiare i biocolori nei contenitori 1 e 3 con meno di tre istruzioni. Una sola istruzione non è ovviamente sufficiente: Con essa si può cambiare solo una sostanza, mentre noi dobbiamo cambiare due sostanze.

Se sono sufficienti due istruzioni, un'istruzione nel contenitore 1 deve produrre il biocolore AAC e l'altra istruzione nel contenitore 3 deve produrre il biocolore CAC.

Consideriamo innanzitutto il caso in cui il biocolore AAC debba essere creato prima nel contenitore 1. Le due possibili istruzioni sono:

- 2 → 1: ACA + CAC → AAA
- 3 → 1: AAC + CAC → ACC

Non è quindi possibile creare AAC nel contenitore 1 con una sola istruzione, perché la seconda istruzione non può modificare due sostanze, in questo caso non sono sufficienti due istruzioni.

Consideriamo ora il caso in cui CAC debba essere creato per primo nel contenitore 3. Le due possibili istruzioni sono:

- 1 → 3: CAC + AAC → ACC
- 2 → 3: ACA + AAC → CAA

Non è quindi possibile che il biocolore CAC venga creato nel contenitore 3 con una sola istruzione. Anche in questo caso, due istruzioni non sono sufficienti e sono quindi necessarie almeno tre istruzioni.

Questa è l'informatica!

Le regole secondo le quali il blocco di costruzione della nuova molecola viene creato a partire dai blocchi di costruzione A e C corrispondono all'operazione logica XOR (exclusive OR). Questa operazione può essere riconosciuta se A e C sono intesi come i valori di verità «vero» e «falso» rispettivamente: XOR restituisce «vero» proprio quando i suoi due operandi hanno valori di verità *diversi*.

XOR ha un significato speciale in informatica. Le proprietà di «scambio» di questa operazione possono essere utilizzate per scambiare i valori di due variabili senza la necessità di una terza variabile temporanea, come nel caso dei contenitori di questo compito.

Lo XOR è un'operazione importante anche nella crittografia. Immaginate di avere un messaggio rappresentato in codice binario. È possibile crittografarlo collegandolo a una chiave della stessa lunghezza mediante XOR. Ad esempio: 01011 (messaggio) XOR 11001 (chiave) = 10010 (testo cifrato). È possibile ripristinare il messaggio facendo lo XOR tra il testo cifrato e la chiave: 10010 (testo cifrato) XOR 11001 (chiave) = 01011 (messaggio) Senza la chiave, tuttavia, il messaggio rimane completamente nascosto, poiché ogni bit del testo cifrato può essere derivato in egual misura da uno 0 o da un 1 nel messaggio. Lo XOR è utilizzato, ad esempio, nella *crittografia corrente*, adatta ad essere utilizzata, tra l'altro, nelle comunicazioni mobili.

Parole chiave e siti web

- Operazione XOR: `https://it.wikipedia.org/wiki/Disgiunzione_esclusiva`
- Scambio tramite XOR: `https://it.wikipedia.org/wiki/Scambio_tramite_XOR`
- Cifratio a flusso attuale: `https://it.wikipedia.org/wiki/Cifrario_a_flusso`

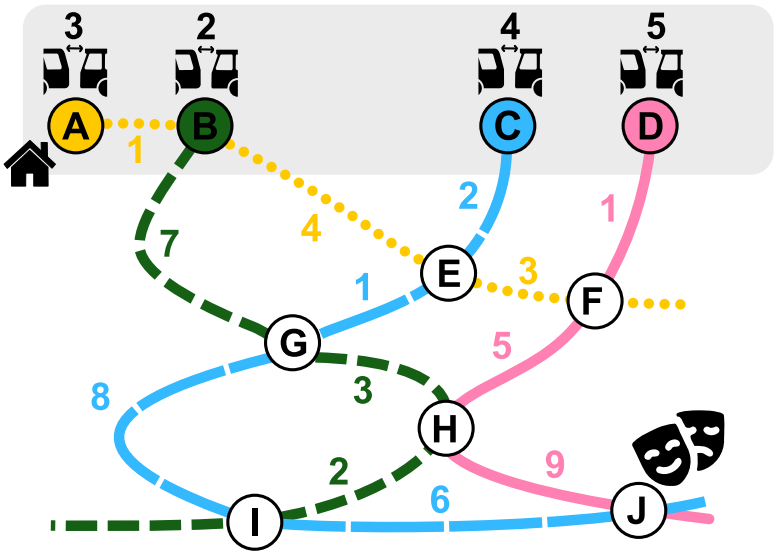
30. Trasporto pubblico

Nella città di Marcus ci sono quattro linee di autobus che partono dalle fermate A, B, C e D. La mappa qui sotto ne illustra i percorsi.

I primi autobus di ogni linea partono dalle fermate di partenza (A, B, C, D) contemporaneamente al minuto 0. Gli autobus successivi partono a intervalli di tempo specificati. Ad esempio, gli autobus partono dalla fermata A ogni 3 minuti, cioè ai minuti 0, 3, 6, 9, ...

Il numero presente sopra ogni tratto di percorso indica quanti minuti ci vogliono all'autobus per percorrerlo. Per esempio il primo autobus in partenza da A raggiunge la fermata F al minuto $0 + 1 + 4 + 3 = 8$.

Marcus abita vicino alla fermata dell'autobus A. Da lì vuole prendere il primo autobus e raggiungere il teatro. Alle fermate nelle quali le linee degli autobus si intersecano, Marcus può cambiare autobus immediatamente. Può quindi proseguire il suo viaggio su qualsiasi autobus che arrivi alla fermata alla stessa ora o più tardi di lui. Marcus sa quale percorso fare e quale autobus cambiare per arrivare al teatro il prima possibile.



Quali sono le fermate di questo percorso?

Soluzione

Il percorso più breve per raggiungere il teatro passa per le fermate A, B, E, G, H e J. Marcus può raggiungere il teatro al minuto 20, partendo da casa al minuto 0.

Come si calcola questo tempo minimo e il percorso corrispondente? Il tempo minimo necessario a Marcus per raggiungere la fermata J (e quindi il teatro) è il minimo tra:

- il tempo minimo per raggiungere la fermata H, più il tempo di attesa per l'autobus da D, più 9 minuti, e
- il tempo minimo per raggiungere la fermata I, più il tempo di attesa per l'autobus da C, più 6 minuti.

Procedendo a ritroso, possiamo calcolare le tempistiche che ci vogliono a raggiungere rispettivamente la fermata H e la fermata I, trovando la soluzione che passa per H. In alternativa, possiamo calcolare il tempo minimo per tutte le stazioni raggiungibili dalla stazione A di Marcus, tenendo a mente il tempo minimo per la fermata precedente. Procedendo in questo modo possiamo calcolare il tempo minimo per raggiungere il teatro e le fermate lungo questo percorso ottimale.

- Fermata **A**: Marcus raggiunge A al minuto **0**.
- Fermata **B**: Marcus raggiunge B al **minuto 1**, **da A**.
- Le fermate C e D sono non vengono considerate nel percorso di Marcus.
- Fermata **E**: Marcus raggiunge E solo **da B**, senza aspettare a B, e ci mette 5 minuti, raggiungendola al **minuto 5**.
- Fermata **F**: Marcus raggiunge F (senza deviazioni) solo **da E**, senza tempi di attesa a E, al **minuto 8**.
- Fermata **G**: Marcus può raggiungere G in due modi, ovvero da B o da E. (1) Marcus si trova alla fermata B dopo 1 minuto, può partire da lì al minuto 2 e raggiunge la fermata G al minuto 9. (2) Marcus raggiunge la fermata E al minuto 5, può partire da lì al minuto 6 (l'autobus che parte da C arriva a E al minuto 2, 6, 10, ...) e raggiunge la fermata G al minuto 7. Marcus può quindi raggiungere la fermata G il prima possibile al **minuto 7, da E**.
- Fermata **H**: Marcus può raggiungere H da F o G. (1) Si trova alla fermata F al minuto 8, può partire al minuto 11 (l'autobus che parte da D arriva a F al minuto 6, 11, 16, ...) e raggiunge la fermata H al minuto 16. (2) Si trova alla fermata G al minuto 7, può partire direttamente (l'autobus che parte da B arriva a G al minuto 7, 9, 11, ...) e raggiunge la fermata H al minuto 10. Marcus può quindi raggiungere H il prima possibile al **minuto 10, da G**.
- Fermare **I**: Marcus può raggiungere I da H o G. (1) Marcus si trova alla fermata H al minuto 10, può partire direttamente (l'autobus che parte da B arriva a H al minuto 10, 12, 14, ...) e raggiunge la fermata I al minuto 12. (2) Marcus si trova in G tra 7 minuti, può partire direttamente (l'autobus che parte da C arriva a G al minuto 3, 7, 11, ...) e raggiunge la fermata I al minuto 15. Marcus può quindi raggiungere I il prima possibile al **minuto 12, da H**.
- Fermata **J** / **Teatro**: Marcus può raggiungere J da H o I. (1) Dalla fermata H, Marcus si trova al minuto 10, può partire al minuto 11 (l'autobus che parte da D raggiunge H al minuto 6, 11,

16, ...) e si trova alla fermata J al minuto 20. (2) Dalla fermata I, Marcus si arriva al minuto 12, può partire al minuto 15 (l'autobus che parte da C arriva a I al minuto 11, 15, 19, ...) e raggiunge la fermata J al minuto 21. Marcus può quindi raggiungere J al più presto al **minuto 20, da H**.

Tenendo a mente da dove Marcus raggiunge una fermata nel più breve tempo possibile, possiamo seguire il suo percorso a ritroso: J ← H ← G ← E ← B ← A.

Questa è l'informatica!

L'idea chiave per determinare la risposta corretta è stata quella di ridurre il problema originale (ovvero: arrivare alla fermata del bus J il prima possibile) a due problemi più piccoli (i arrivare il prima possibile alle fermate I o H). Si può proseguire con questo metodo calcolando quando si arriverebbe a I il prima possibile tramite il calcolo di arrivo alle fermate G e H, e così via. Questa strategia di «ripetere» una funzione di calcolo (l'arrivo più veloce a una data fermata, in questo caso) a sé stessa è nota in informatica come *recursione*.

Quando abbiamo spiegato la risposta corretta, abbiamo «risolto» la ricorsione e costruito i risultati della funzione partendo dal caso più semplice passo dopo passo (in informatica si dice: *iterativamente*). Alla fine siamo stati in grado di ricondurre l'esecuzione del calcolo per J all'esecuzione del calcolo per I e H, e via dicendo, consentendoci di calcolare il risultato per J tramite i risultati dello stesso calcolo (stessa funzione) per I e H.

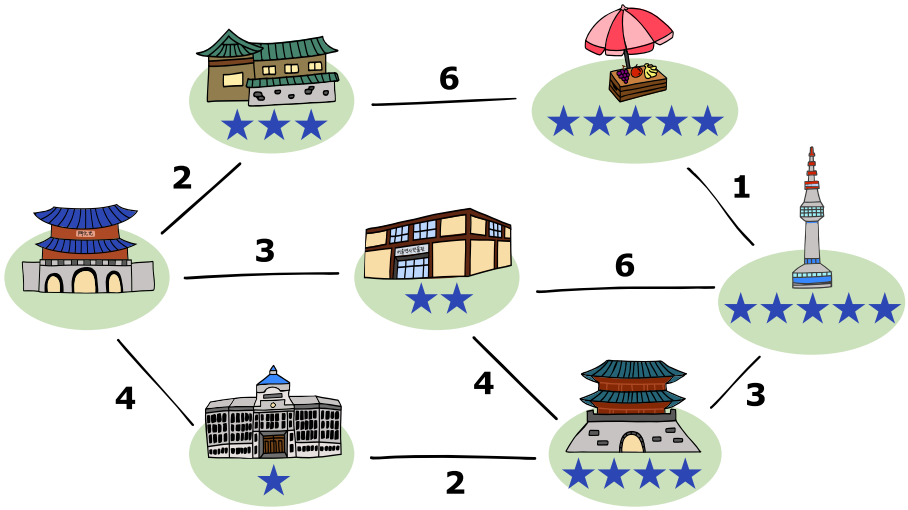
In informatica, questa strategia è chiamata *programmazione dinamica*. Può essere utilizzata per problemi di ottimizzazione, come la ricerca del primo orario di arrivo di Marcus a teatro in questo compito. La programmazione dinamica funziona proprio quando si applica il principio di ottimizzazione di Bellman, cioè quando la soluzione ottimale di un problema può essere composta dalle soluzioni ottimali dei sottoproblemi. In tal caso, spesso funziona bene perché è sufficiente calcolare ogni soluzione (parziale) una sola volta quando si costruiscono iterativamente le soluzioni (parziali).

Parole chiave e siti web

- Programmazione dinamica: `https://it.wikipedia.org/wiki/Programmazione_dinamica`
- Ricorsione: `https://it.wikipedia.org/wiki/Algoritmo_ricorsivo`
- Iterazione: `https://it.wikipedia.org/wiki/Iterazione`

31. Scopriamo Seul!

A Seul, in Corea, ci sono autobus per turisti che collegano luoghi che vale la pena vedere. L'immagine mostra i luoghi più importanti di Seul. Le stelle indicano la popolarità dei luoghi. Le linee mostrano i collegamenti in autobus. Ogni linea indica i chilometri di lunghezza del collegamento.



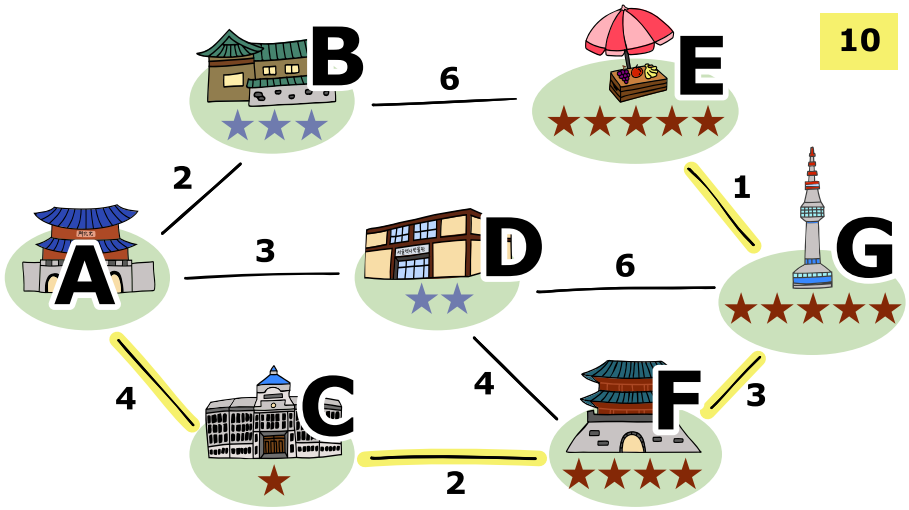
Lotte visita prima il palazzo  col tetto blu a sinistra della mappa. Da lì, vorrebbe visitare altri luoghi in autobus. Lotte ha un biglietto con cui può viaggiare per un massimo di 10 chilometri. Vuole utilizzare le coincidenze per raggiungere luoghi con il maggior numero possibile di stelle! Può visitare un luogo solo una volta e non vuole tornare al palazzo.

Quali tratte col bus deve fare Lotte con il suo biglietto per «raccogliere» il maggior numero di stelle?

Soluzione

Vogliamo trovare un percorso in autobus per Lotte che parta dal palazzo a sinistra, non sia più lungo di 10 chilometri e la porti in luoghi che abbiano il maggior numero di stelle. Pertanto, nel descrivere i percorsi, non dobbiamo considerare solo le singole località, ma anche la distanza dal palazzo e il numero di stelle «raccolte» sul percorso.

In primo luogo, etichettiamo le singole località con le lettere da A a G.



Ora è possibile descrivere la tappa di un percorso in una località con:

- le lettere dei luoghi,
- la distanza totale dalla partenza (A) a questa posizione e
- il numero totale di stelle raccolte nel percorso dall'inizio a questa posizione.

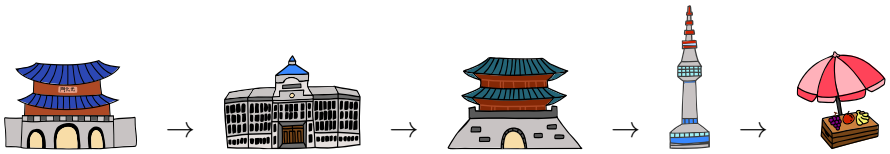
Un percorso viene definito *valido* se non supera i 10 chilometri di lunghezza. Un percorso valido è, ad esempio

- da A a B: B è a 2 km da A e ha 3 stelle, quindi questo scalo è contrassegnato da B(2,3);
- Continuare da B a E: la distanza totale percorsa aumenta da 2 km a 8 km (il percorso da B a E è di 6km), il numero totale di stelle raccolte aumenta da 3 a 8 (perché E ha 5 stelle), quindi questa tappa intermedia viene etichettata come E(8,8);
- l'ultima da E a G: vengono aggiunti 1 km di distanza e 5 stelle, in modo che questa tappa, che è anche la fine di questo percorso valido, sia etichettata come G(9,13).

Questi sono tutti i percorsi validi:

```
A → B(2,3) → E(8,8) → G(9,13)
A → D(3,2) → G(9,7) → E(10,12)
A → D(3,2) → F(7,6) → G(10,11)
A → D(3,2) → F(7,6) → C(9,7)
A → C(4,1) → F(6,5) → D(10,7)
A → C(4,1) → F(6,5) → G(9,10) → E(10,15)
```


L'ultimo percorso valido è il migliore, perché il suo punto finale E(10,15) ha il maggior numero di stelle totali. Lotte deve quindi utilizzare il suo biglietto per effettuare questi collegamenti, in modo da «raccogliere» il maggior numero possibile di stelle:



Questa è l'informatica!

Lotte ha un obiettivo in questo compito: vuole «raccogliere» il maggior numero possibile di stelle nel suo percorso. Vuole quindi *ottimizzare* un certo valore, ossia il numero totale di stelle per raggiungere il valore più alto possibile. In altre parole, Lotte ha un problema di *ottimizzazione*. Nel risolvere questo problema, è limitata dal suo biglietto, che limita la lunghezza del percorso a 10km.

L'informatica conosce molti metodi per risolvere problemi di ottimizzazione, con o senza vincoli. Tali problemi vengono quindi spesso risolti con l'aiuto di sistemi informatici. I sistemi di navigazione sono un esempio che usiamo quasi tutti i giorni e che ci aiuta a determinare i percorsi ottimali da seguire. Di solito questi sistemi consentono agli utenti di modificare il valore da ottimizzare, per esempio chiedendo se il percorso deve essere il più breve possibile o se deve consumare meno energia possibile. È anche possibile aggiungere delle limitazioni, ad esempio si possono evitare autostrade, strade a pedaggio o collegamenti con traghetti.

I metodi informatici per la ricerca di percorsi utilizzano strutture di dati che possono essere disegnate in modo molto simile alla rete di autobus di questo compito: i *grafi*. Questi consistono in un insieme di *nodi* e in un insieme di coppie di nodi, gli *spigoli*. I grafi possono essere utilizzati per modellare molto bene le relazioni tra le cose, ad esempio i collegamenti di trasporto tra i luoghi. Le distanze possono essere collegate ai bordi come *pesi*. In informatica ci sono molti algoritmi per risolvere problemi i cui dati sono gestiti sotto forma di grafi - per esempio la *ricerca in profondità*, che abbiamo usato sopra per determinare i vari percorsi per Lotte.

Parole chiave e siti web

- Problema di ottimizzazione: `https://it.wikipedia.org/wiki/Problema_di_ottimizzazione`
- Teoria dei grafi: `https://it.wikipedia.org/wiki/Teoria_dei_grafi`
- Ricerca in profondità: `https://it.wikipedia.org/wiki/Ricerca_in_profondità`

32. Laghi di montagna

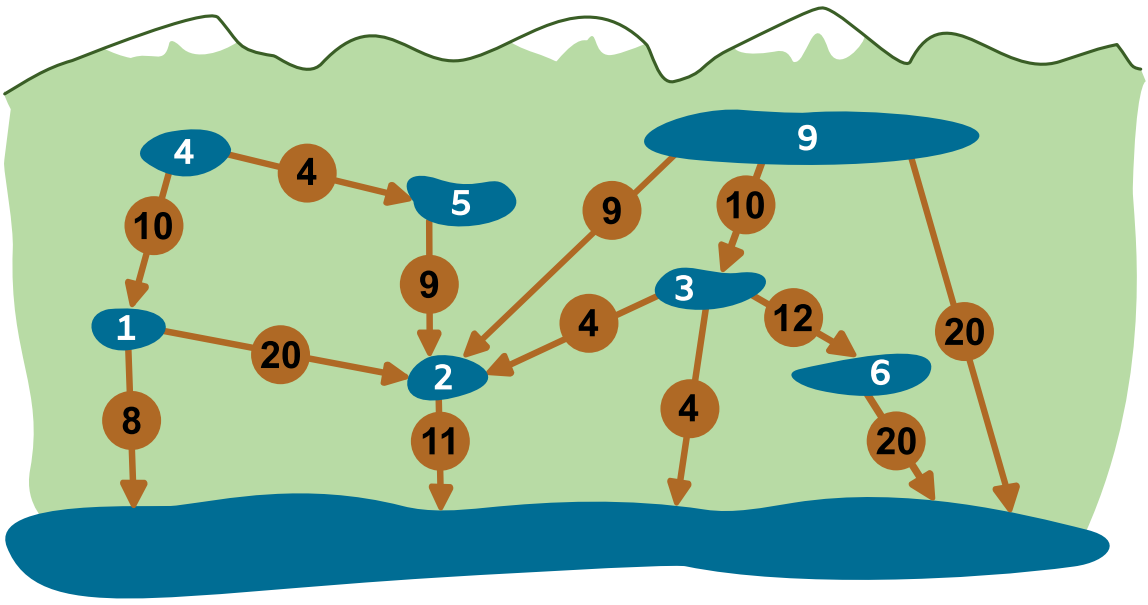
Sul massiccio montuoso sopra un bacino artificiale ci sono diversi piccoli laghi di montagna. In caso di forti piogge potrebbero straripare, il che è pericoloso. Per questo motivo, è prevista la costruzione di canali tra alcuni dei laghi. Questi canali dovrebbero essere in grado di drenare tutta l'acqua in eccesso dai laghi di montagna al bacino artificiale a valle. Allo stesso tempo, la loro costruzione dovrebbe costare il meno possibile.

Per ogni lago di montagna, un numero indica la quantità di acqua in eccesso che deve essere drenata dal lago.

In ogni punto tra due laghi in cui è possibile costruire un canale c'è una freccia che indica la direzione in cui un canale devierebbe l'acqua. Il numero sulla freccia indica la capacità del canale, cioè quanta acqua in eccesso può drenare. La capacità determina anche il costo della costruzione di un canale in quel punto.

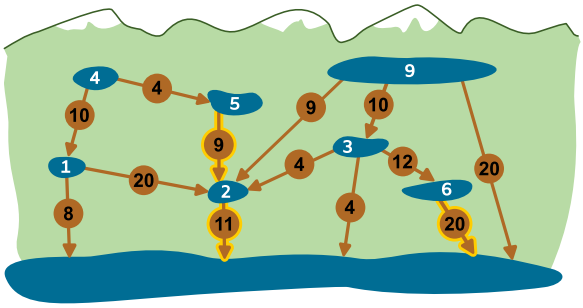
Nota: se un canale drena l'acqua di un piccolo lago di montagna in un secondo lago, l'acqua in eccesso di entrambi i laghi si raccoglie nel secondo lago.

Dove dovrebbero essere costruiti i canali per minimizzarne il costo?

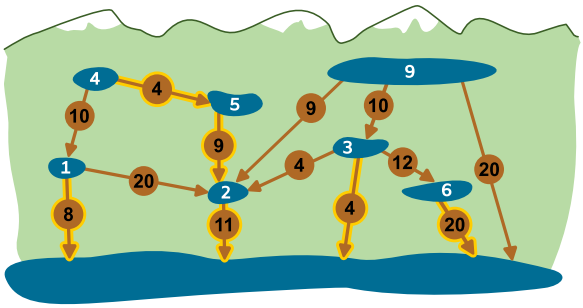


Soluzione

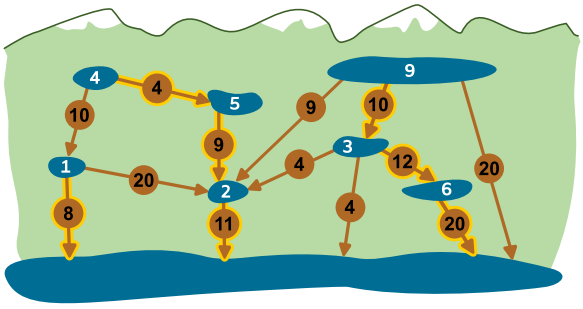
La risposta corretta è la seguente:



Per alcuni laghi esiste un solo punto in cui un canale può deviare l'acqua. Si tratta dei laghi con eccedenza d'acqua 2 in basso al centro (in breve: lago 2), 5 e 6. In questi punti è quindi necessario costruire un canale. La capacità di questi canali è sufficiente per questi tre laghi, anche considerando che dal lago 2 deve essere scaricato un eccesso totale di 7 a causa dell'afflusso dal lago 5. La costruzione di questi tre canali costa $11 + 9 + 20 = 40$.



Per i laghi 1 e 4 ci sono due punti di costruzione del canale. (a) Se il canale viene costruito dal 4 al 5, il canale dal lago 2 al lago grande deve deviare un eccesso di $4 + 5 + 2 = 11$. In questo modo la sua capacità è esaurita, quindi il lago 1 non può essere deviato nel lago 2, ma nel bacino artificiale a valle. I costi complessivi sono quindi $4 + 8 = 12$. (b) In alternativa, è possibile costruire il canale dal 4 al 1. Se poi si costruisse il canale dal lago 1 al 2, dal lago 2 dovrebbero essere deviati complessivamente $4 + 1 + 7 = 12$, cosa che non è possibile per via della capacità di 11 del canale da 2 al bacino artificiale. In questo caso, quindi, il canale dal lago 1 al bacino articifiale deve essere necessariamente costruito. Il costo totale sarebbe quindi $10 + 8 = 18$, quindi più elevato.



Restano i laghi 3 e 9. (a) Il lago 9 non può essere deviato nel lago 2 perché ciò supererebbe la capacità del canale dal lago 2 al bacino (anche se non esistesse il canale dal 4 al 5). (b) Se il lago 9 fosse deviato direttamente nel bacino, la soluzione complessiva più conveniente per i laghi 3 e 9 avrebbe un costo di $20 + 4 = 24$. (c) Se il lago 9 fosse deviato nel lago 3, si creerebbe un eccesso di 12, che potrebbe essere deviato solo nel lago 6. L'eccedenza di 18 che si crea lì può essere deviata nel bacino artificiale attraverso il canale già costruito. I costi per i laghi 3 e 9 sono quindi $10 + 12 = 22$, quindi più convenienti.

I canali selezionati possono deviare tutta l'acqua in eccesso dai laghi di montagna al bacino artificiale con un costo minimo di $40 + 12 + 22 = 74$.

Questa è l'informatica!

I laghi (laghi di montagna e bacino artificiale) e i cantieri dei canali possono essere modellati come un *grafo*. Un grafo ha *nodi* (qui rappresentati come laghi) che possono essere collegati tra loro da spigoli (qui rappresentati come i cantieri dei canali). Come in questo compito, gli spigoli possono avere una direzione e anche un *peso*, come la capacità potenziale del canale nei siti di costruzione (e il loro costo).

Modellare un problema con l'aiuto di strutture note è particolarmente utile se si possono utilizzare algoritmi di risoluzione che l'informatica già conosce. Molti problemi sono descritti come grafi e per molti di essi sono già noti efficienti algoritmi di risoluzione. Tra questi vi sono anche i problemi di flusso, come il «problema del flusso di costo minimo», che sono correlati al problema di questo compito.

Parole chiave e siti web

- Grafo: `https://it.wikipedia.org/wiki/Grafo`
- Rete di flusso: `https://it.wikipedia.org/wiki/Rete_di_flusso`
- Ricerca locale: `https://it.wikipedia.org/wiki/Ricerca_locale`

33. Parcheggi

Ad una festa in casa sono invitate 9 persone che arrivano con le loro auto. Davanti alla casa ci sono 9 posti auto disposti in 3 file che ospitano 3 auto ciascuna. Gli invitati arrivano in quest'ordine:

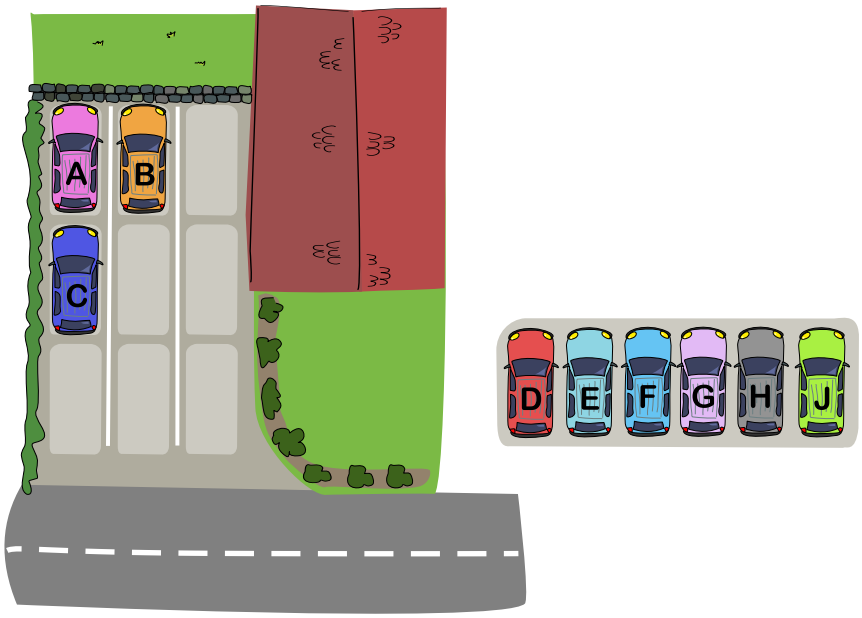
Anja, **B**eate, **C**lara, **D**avid, **E**lia, **F**rank, **G**abi, **H**arald e infine **J**ulia.

Quando gli invitati parcheggiano, ognuno sceglie una corsia di parcheggio e parcheggia il più avanti possibile.

Gli ospiti vogliono lasciare la festa in quest'ordine:

Gabi, **D**avid, **B**eate, **E**lia, **J**ulia, **C**lara, **H**arald, **A**nja e infine **F**rank.

Le auto di **A**nja, **B**eate e **C**lara sono già parcheggiate. Ora arrivano gli altri ospiti, parcheggiando uno alla volta. Vogliono parcheggiare in modo che, quando se ne vanno, nessuna auto sia bloccata da un'altra che se ne andrà più tardi.

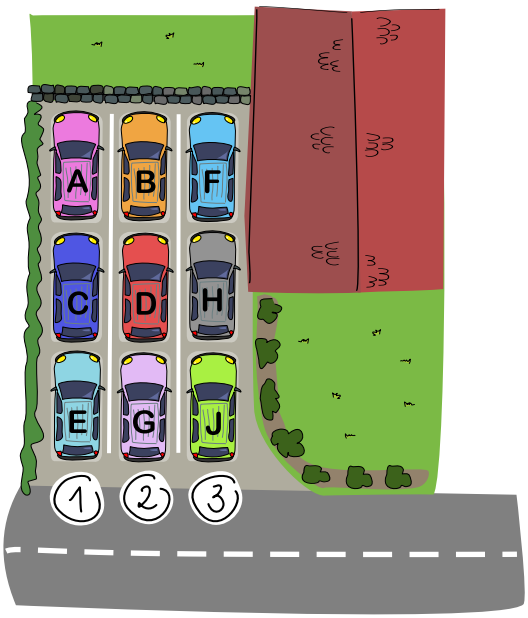


Mostra agli ospiti come parcheggiare!

Posiziona le 6 auto rimanenti nelle corsie di parcheggio. È necessario tenere conto dell'ordine di arrivo ***e*** *di partenza.*

Soluzione

Questa è la risposta:



Per prima cosa, è utile fare qualche constatazione:

a) **F**rank parte per ultimo, quindi deve parcheggiare in cima a una delle file di parcheggio.
b) **G**abi parte per primo, quindi deve parcheggiare in una delle tre corsie in fondo.
c) **B**eate è la terza persona a lasciare la festa, quindi **D**avid e **G**abi devono parcheggiare dietro di lei.

Questo ci permette di fare ulteriori considerazioni:

- A causa di a), **F**rank ha solo il posto accanto a **B**eate, nella parte anteriore della corsia di parcheggio 3.
- Dopo **A**nna, **B**eate e **C**lara, arriva **D**avid. A causa di c), **D**avid deve parcheggiare dietro **B**eate nella corsia di parcheggio 2.
- **G**abi deve parcheggiare dietro **D**avid nella corsia di parcheggio 2 a causa di c).
- Rimane solo la corsia di parcheggio 1 per **E**mil dietro **C**lara. **E**mil è la quarta persona a lasciare la festa (dopo **G**abi, **D**avid, **B**eate) e quindi deve stare in fondo a una corsia. Quando arriva, solo la corsia di parcheggio 1 permette di parcheggiare in ultima posizione, perché nella corsia di parcheggio 3 c'è solo **F**rank finora, quindi **E**mil dovrebbe parcheggiare al centro.
- Alla fine, rimane solo la corsia di parcheggio 3 per **H**arald (spazio centrale) e **J**ulia (posteriore). Fortunatamente **J**ulia vuole partire prima di **H**arald, altrimenti non ci sarebbe una risposta corretta.

Poiché esiste esattamente una posizione di parcheggio fissa per ogni singola auto, esiste una sola risposta corretta.

Questa è l'informatica!

Le auto parcheggiate nelle 3 corsie di parcheggio si comportano in modo tale che solo l'ultima auto parcheggiata può partire. È come una pila di piatti in cui è possibile rimuovere in sicurezza solo l'ultimo piatto posizionato sulla pila (quello più in alto).

L'informatica riconosce le pile, o *stack*, come struttura di dati. Questa struttura funziona come le corsie di parcheggio o le pile di piatti: l'operazione *push* aggiunge un oggetto allo stack. La funzione *top* restituisce l'ultimo oggetto aggiunto e l'operazione *pop* lo rimuove dalla pila. Nell'informatica teorica, invece, esistono modelli di calcolo che utilizzano le pile. Un automa a pila (noto anche con la sigla PDA, dall'inglese *pushdown automaton*) appartiene ai cosiddetti *linguaggi libero dal contesto*, che possono essere facilmente elaborati dai computer; ne sono un esempio i linguaggi di programmazione o i linguaggi di markup come l'HTML.

Gli stack multipli - come le corsie di parcheggio multiple in questo compito - sono utilizzati nei sistemi operativi dei computer, ad esempio per distribuire i compiti a più processori. Se si aggiunge almeno un'altra pila all'automa, è possibile utilizzarlo per modellare qualsiasi calcolo, proprio come avviene con una macchina di Turing. Il secondo stack fa la differenza!

Parole chiave e siti web

- Pila: `https://it.wikipedia.org/wiki/Pila_(informatica)`
- Multiprocessore: `https://it.wikipedia.org/wiki/Multiprocessore`
- Automa a pila: `https://it.wikipedia.org/wiki/Automa_a_pila`

34. Castoro Jones

Il Castoro Jones  si trova in una piramide piena di trappole. Alla fine di alcuni corridoi all'interno di questa piramide si trovano dei tesoro. Jones vuole raggiungere i tesori il più velocemente possibile.

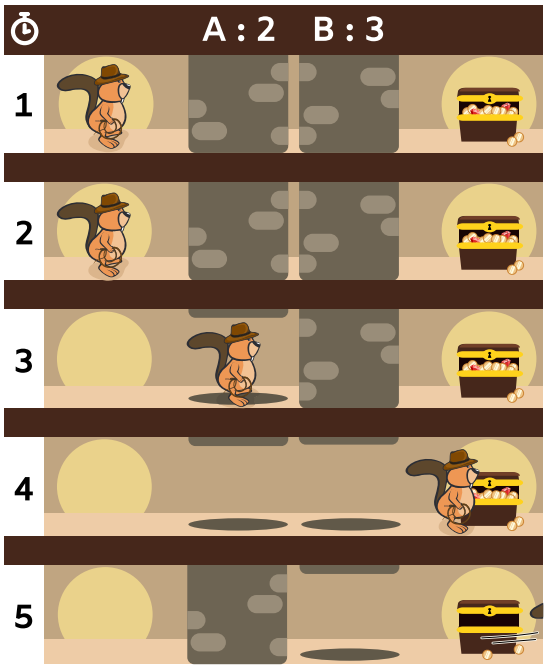
Il problema è che i tesori sono protetti da una serie di blocchi che salgono e scendono. All'inizio, tutti i blocchi sono abbassati. Non appena qualcuno entra nei corridoi, i blocchi iniziano a muoversi ritmicamente. Ogni blocco si muove su e giù con un ciclo fisso. Ad esempio, un blocco con un tempo di ciclo di 2 si alzerà dopo 2 minuti e si abbasserà di nuovo dopo altri 2 minuti.

Nella seguente immagine Jones sta cercando di superare il primo corridoio:

Questo corridoio ha due blocchi: il blocco A che ha un ciclo di 2 e il blocco B che ha un ciclo di 3. Fortunatamente, Jones ha trovato una pergamena con le istruzioni su come raggiungere il tesoro in modo sicuro e il più rapidamente possibile:

```
wait(2)
goto_block(A)
wait(1)
goto_treasure
```

Jones segue le istruzioni: aspetta 2 minuti, poi va al blocco A, aspetta 1 minuto e poi va al tesoro. Raggiunge il tesoro dopo 3 minuti:

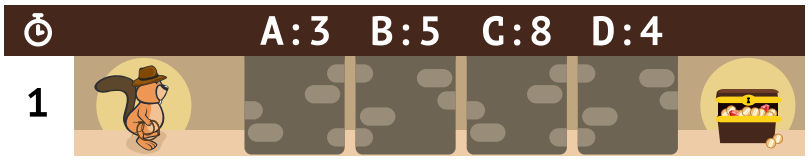


Jones si rende conto che avrebbe potuto raggiungere il tesoro con meno istruzioni e altrettanto velocemente:

```
wait(3)
goto_treasure
```

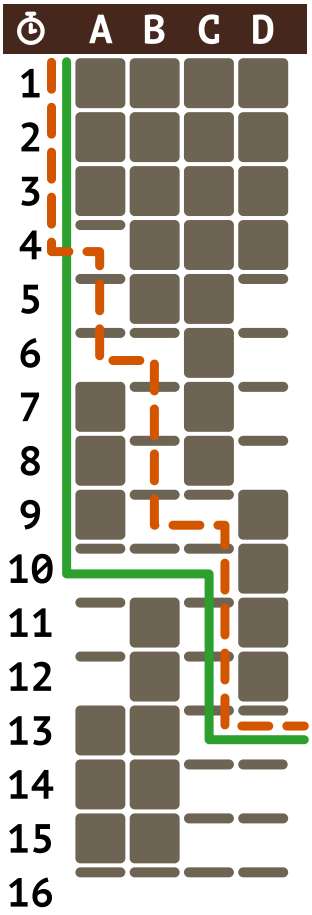
Jones raggiunge quindi il prossimo corridoio. Qui si trova confrontato con quattro blocchi, con cicli di rispettivamente 3, 5, 8 e 4 minuti.

Qual è la sequenza più breve di istruzioni che Jones può utilizzare per raggiungere il tesoro il più rapidamente possibile?



Soluzione

L'immagine mostra come Jones può muoversi nel corridoio:



Per procedere il più rapidamente possibile, può seguire le istruzioni riportate di seguito e raggiungere il tesoro dopo 12 minuti (vedi linea arancione):

```
wait(3)
goto_block(A)
wait(2)
goto_block(B)
wait(3)
goto_block(C)
wait(4)
goto_treasure
```

Tuttavia, esiste una sequenza di istruzioni più breve con la quale raggiunge il tesoro dopo comunque 12 minuti (vedi linea verde):

```
wait(9)
goto_block(C)
wait(3)
goto_treasure
```

Non c'è modo di raggiungere il tesoro con un numero ancora minore di istruzioni senza perdere tempo. L'unica possibilità per Jones sarebbe quella di superare il passaggio in una sola volta. Tuttavia, dovrebbe aspettare 15 minuti (quando tutti i blocchi sarebbero alzati nello stesso momento) e raggiungerebbe il tesoro più tardi.

Questa è l'informatica!

Beaver Jones, l'avventuriero senza fiato, vuole raggiungere il favoloso tesoro il più velocemente possibile. Vuole essere il primo! Al nostro avventuriero però piace ottimizzare e non si accontenta di una qualsiasi istruzione che lo porti avanti rapidamente. No, cerca la sequenza di istruzioni più breve che gli consenta di raggiungere il suo obiettivo. In altre parole, ottimizza con due obiettivi contemporaneamente: il minor tempo possibile e il minor numero di istruzioni possibile.

Sarebbe un po' più difficile se Jones avesse scelto obiettivi di ottimizzazione non necessariamente compatibili tra loro: da un lato, arrivare al tesoro il più velocemente possibile e, dall'altro, mantenere il suo abbigliamento da avventuriero pulito (cosa impossibile in una piramide!). In informatica e matematica, questi casi sono definiti «ottimizzazione multi-obiettivo». Di solito non esiste un unico optimum, ma diversi optima di Pareto. Un optimum di Pareto (dal nome dell'economista Vilfredo Pareto) è uno stato in cui non è possibile migliorare un valore obiettivo senza peggiorarne contemporaneamente un altro.

Come tutti i bravi informatici, anche Jones è molto attento alla qualità. In informatica, questa parola chiave non riguarda sempre la qualità o l'ottimizzazione dei risultati dei calcoli. Ad esempio, esiste anche la qualità del codice del programma, che può essere misurata in base a vari criteri, come la struttura del codice o i commenti che contiene. Tra diversi codici con la stessa funzione, quello con il minor numero di istruzioni potrebbe essere l'approccio privilegiato, come in questo compito.

Parole chiave e siti web

- Problema di ottimizzazione:
  `https://it.wikipedia.org/wiki/Problema_di_ottimizzazione`

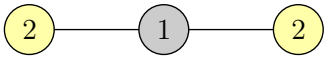
35. Pianificazione degli esami

Gli esami di maturità si avvicinano al liceo Castoro. Il liceo prevede una sessione di cinque giorni per sostenere tutti gli esami. Alcuni esami non possono essere sostenuti lo stesso giorno, si può dunque dire che hanno un «conflitto giornaliero».

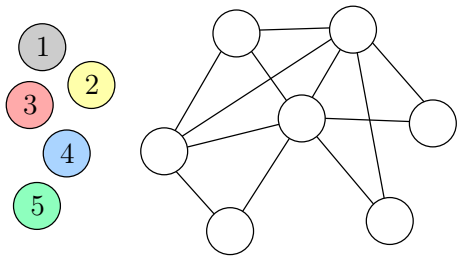
Per facilitare la pianificazione, i conflitti giornalieri sono rappresentati in un diagramma di cerchi e linee:

- Ogni esame è rappresentato con un cerchio con il giorno nel quale avviene, rappresentato da un numero.
- Una linea viene tracciata tra due cerchi quando questi due esami hanno un conflitto giornaliero, cioè non possono essere pianificati nello stesso giorno.

Ecco un esempio con tre esami: l'esame al centro ha due conflitti giornalieri, ovvero con ciascuno degli altri due esami. I numeri mostrano un modo per distribuire gli esami su i due giorni (1 e 2). L'esame al centro è pianificato nel giorno 1, mentre i due esami adiacenti nel giorno 2.



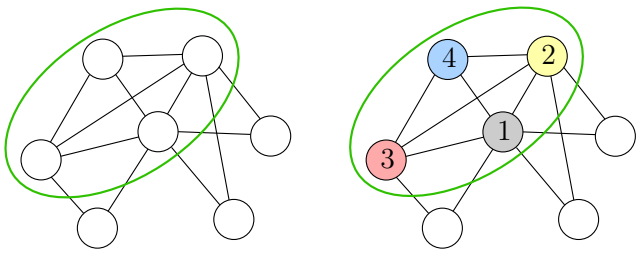
Il liceo Castoro prevede di avere sette esami nei prossimi cinque giorni. Il diagramma mostra i loro conflitti giornalieri.



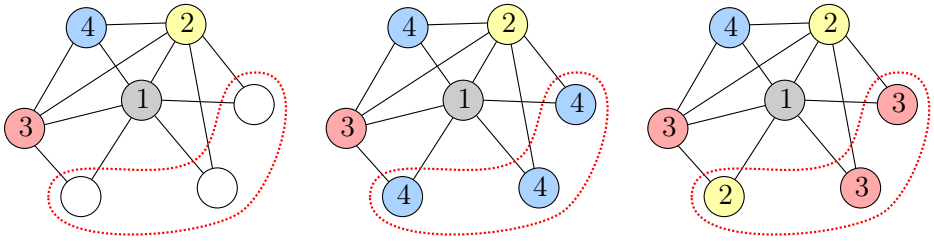
Distribuisci gli esami nel minor numero possibile di cinque giorni (da 1 a 5), tenendo conto dei conflitti giornalieri. Ci sono diverse risposte corrette.

Soluzione

Uno sguardo più attento al diagramma rivela una struttura speciale negli esami mostrati nell'angolo in alto a sinistra (immagine di sinistra). Ognuno di questi esami ha un conflitto giornaliero con tutti gli altri, per cui devono essere scritti in quattro giorni diversi. Nell'immagine a destra si può vedere un modo per distribuire questi esami su quattro giorni. Ci sono $1 \times 2 \times 3 \times 4 = 24$ altri modi di distribuire gli esami su quattro giorni, tutti corretti.



Per questa distribuzione, cerchiamo una soluzione per i tre esami rimanenti (immagine a sinistra). Nessuno di essi può avvenire il giorno 1, ma tutti possono avvenire il giorno 4 (immagine centrale). Se si vogliono tenere gli esami nei giorni 2 o 3, ciò è possibile solo come mostrato nell'immagine a destra, a causa dei conflitti giornalieri. Inoltre, questa distribuzione offre ancora sei possibilità di tenere uno o due degli esami previsti per il secondo e il terzo giorno nel quarto giorno.



Per ognuna delle 24 possibilità per i quattro esami in alto a sinistra, ci sono quindi 8 possibilità di distribuire i tre esami in basso a destra. Ci sono quindi 192 risposte corrette con i giorni {1,2,3,4} e altrettante rispettivamente con i giorni {2,3,4,5}, {1,3,4,5}, {1,2,4,5} e {1,2,3,5}. In totale, abbiamo ben $5 \times 192 = 960$ possibili risposte corrette!

// Für jede der 24 Möglichkeiten für die vier Klausuren oben links gibt es also 8 Möglichkeiten, die drei Klausuren unten rechts zu verteilen. Es gibt also 192 richtige Antworten mit den Tagen {1,2,3,4} und jeweils noch einmal so viele mit den Tagen {2,3,4,5}, {1,3,4,5}, {1,2,4,5} und {1,2,3,5}. Insgesamt sind das $5 \times 192 = 960$ richtige Antworten.//

Questa è l'informatica!

Quando si distribuiscono gli esami, è di aiuto rappresentare i conflitti giornalieri con un diagramma. Questi «problemi di distribuzione» sono di norma più grandi e devono essere risolti con l'aiuto di sistemi informatici, ma risulta sempre importante utilizzare una rappresentazione o una modellazione dei dati. Il diagramma di conflitto in questo compito è la rappresentazione di un *grafo*, una struttura particolarmente importante in informatica per modellare le relazioni, cioè i rapporti tra gli oggetti.

Un grafico è costituito da *nodi* (qui rappresentati come cerchi) e *bordi* che collegano due nodi tra loro (le linee tra i cerchi).

Il «problema della distribuzione» per contrassegnare i nodi di un grafo con il minor numero possibile di valori, in modo che due nodi collegati da un bordo abbiano valori diversi, è noto in informatica come *problema di colorazione del grafo* - i valori numerici che abbiamo usato in questo compito possono essere immaginati come colori diversi. I problemi di colorazione possono presentarsi in molti campi di applicazione, ad esempio ...

- ... nella colorazione delle carte geografiche in cui i paesi confinanti devono avere colori diversi;
- ... nella pianificazione degli esami, come in questo compito;
- ... nella pianificazione delle frequenze nelle reti radiomobili, se si vogliono evitare interferenze radio tra antenne di trasmissione vicine; oppure
- ... nela distribuzione dei treni sui binari di una stazione, poiché i treni con tempi di sosta sovrapposti devono utilizzare binari diversi.

In generale, i problemi di colorazione sono tra i problemi più difficili conosciuti in informatica, i cosiddetti problemi *NP-hard*. A seconda dell'applicazione, tuttavia, esistono problemi di colorazione speciali che sono molto più facili da risolvere. E nei casi veramente difficili si possono usare metodi che non garantiscono di trovare soluzioni ottimali, ma di solito molto buone.

Parole chiave e siti web

- Colorazione dei grafi: `https://it.wikipedia.org/wiki/Colorazione_dei_grafi`

36. Castoro di controllo

Il capo castoro schiera quattro cosiddetti *castori messaggeri*: il loro compito è formare dei messaggi tramite delle bandiere. Ogni castoro messaggero può sventolare una bandiera rossa o una gialla .

A volte capita che un castoro messaggero sventoli la bandiera sbagliata. Per riconoscerlo, il capo castoro nomina tre *castori di controllo*.

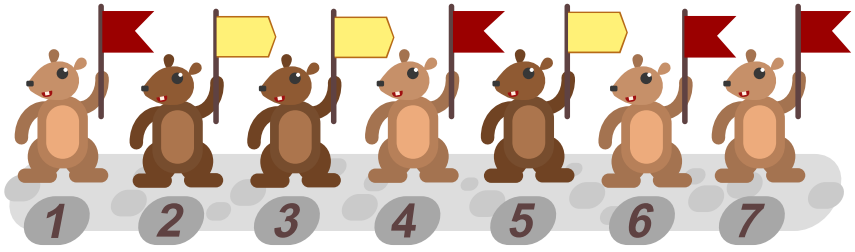
Ogni castoro di controllo tiene d'occhio tre castori messaggeri. Il castoro di controllo sventolerà una bandiera rossa qualora i tre castori messaggeri che controlla sventolino un numero dispari di bandiere rosse. Altrimenti, il castoro di controllo sventolerà una bandiera gialla. Se tutti i castori messaggeri sventolano le bandiere corrette, il castoro di controllo e i castori messaggeri che controlla sventoleranno un numero pari di biandiere rosse.

Il capo numera i castori messaggeri e i castori di controllo e li assegna nel seguente modo:

Castori messaggeri	**Castoro di controllo**
1, 2, 3	5
1, 2, 4	6
2, 3, 4	7

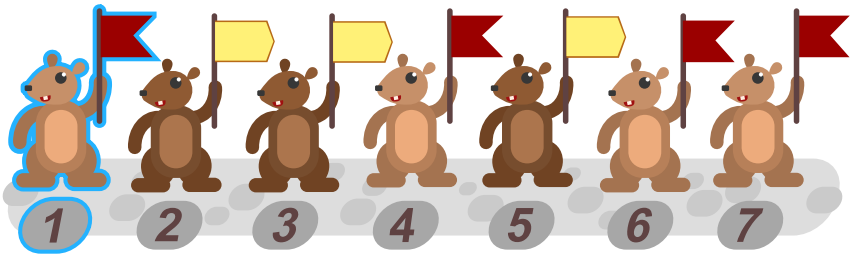
In totale sono sette le bandiere che ora vengono sventolate per formare un messaggio. Quattro messaggeri e tre di controllo. Il capo castoro vede il messaggio sottostante e si rende conto che uno dei sette castori sta sventolando la bandiera sbagliata.

Quale castoro sventola la bandiera sbagliata?

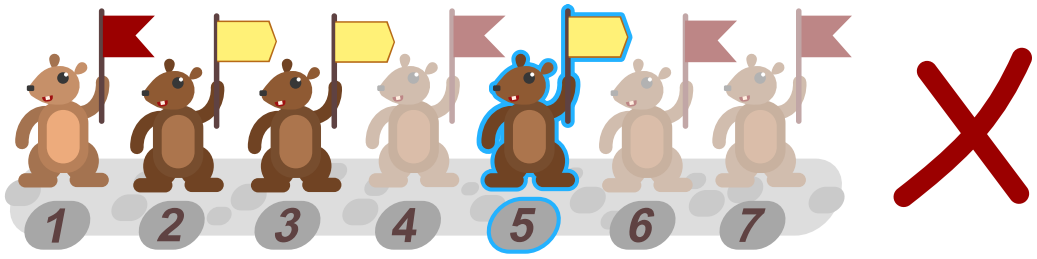


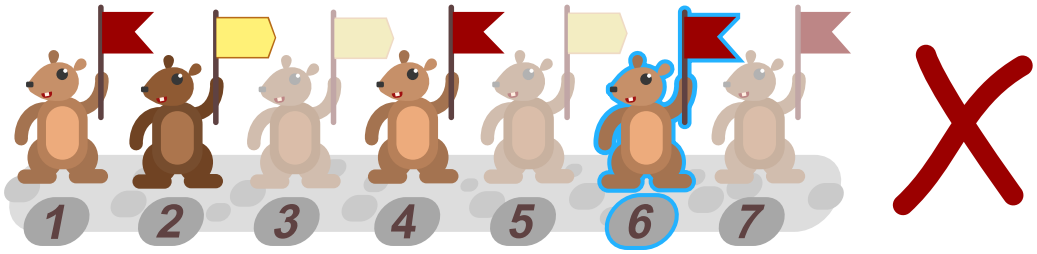
Soluzione

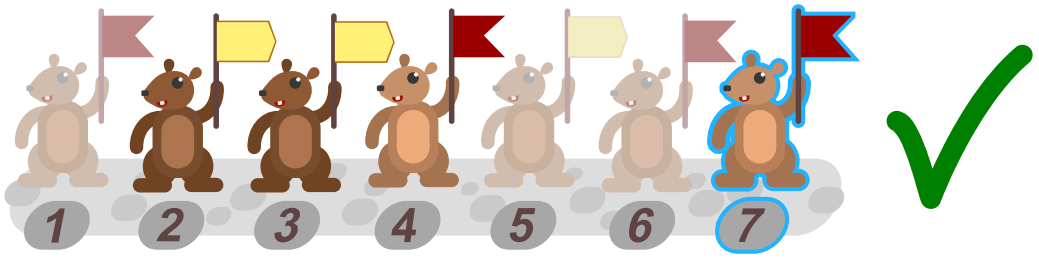
Il castoro numero 1 sventola la bandiera sbagliata.



Per semplicità, un castoro di controllo insieme ai suoi castori messaggeri vengono indicati come gruppo con il numero del rispettivo castoro di controllo. Esistono quindi tre gruppi: 5, 6 e 7. Un gruppo deve sempre sventolare un numero pari di bandiere rosse, secondo le istruzioni. Ecco i gruppi, il castoro di controllo del gruppo è quello evidenziato.







È possibile immediatamente notare che nel gruppo 5 c'è un errore: il gruppo sventola solo una bandiera rossa, quindi un numero dispari. Anche nel gruppo 6 c'è un errore: sventola tre bandiere rosse, anch'esse un numero dispari. Nel gruppo 7 è tutto a posto, perché sventola in totale due bandiere rosse, un numero pari.

I castori del gruppo 7, ovvero i castori 2, 3, 4 e 7, fanno tutto correttamente. Nel gruppo 5, quindi, potrebbe essere il castoro 1 o il castoro 5 a sventolare la bandiera sbagliata. Nel gruppo 6 potrebbe

essere invece il castoro 1 o il castoro 6 a sventolare la bandiera sbagliata. Poiché sappiamo che solo un castoro sventola la bandiera sbagliata, lo stesso castoro deve essere responsabile degli errori nei gruppi 5 e 6. Questo castoro può essere solo il numero 1.

Questa è l'informatica!

Nel mondo digitale i dati vengono codificati in modo binario, ovvero come sequenze di bit (0 e 1). Quando questi vengono trasmessi attraverso canali di comunicazione, possono verificarsi interferenze che invertono i bit: lo 0 diventa 1 o viceversa. Per garantire una corretta elaborazione delle informazioni, è necessario determinare se una trasmissione è stata interessata da un disturbo e correggere gli errori che ne sono derivati. A tal fine sono stati sviluppati delle metodologie per trovare e correggere gli errori.

Un modo semplice sarebbe quello di inviare ogni bit tre volte di seguito. Un disturbo su uno dei tre bit potrebbe essere facilmente rilevato e corretto, poiché gli altri due bit contengono ancora le informazioni corrette; la maggioranza decide quindi. Tuttavia, questo semplice procedimento comporta la necessità di trasmettere una quantità di dati tre volte superiore. Affinché le linee dati non si intasino, è quindi importante inviare il minor numero possibile di bit aggiuntivi per il rilevamento e la correzione degli errori.

Il capo castoro utilizza a questo scopo un *codice di Hamming*. In un codice di Hamming, per diversi gruppi di bit esiste un bit di controllo. Il bit di controllo viene calcolato dai bit di dati del gruppo corrispondente in modo tale che i bit di dati e il bit di controllo contengano insieme un numero pari di «1»; questa proprietà è chiamata anche «parità pari». Se in un messaggio la parità è corretta per tutti i gruppi di bit di dati e bit di controllo associati, si può presumere che il messaggio sia stato trasmesso correttamente. Se solo un bit di un messaggio codificato con il codice di Hamming è stato modificato da un disturbo, è possibile determinare correttamente il bit disturbato e il messaggio originale controllando in quali gruppi la parità non è corretta.

Con tre bit di controllo, il codice di Hamming controlla quattro bit di dati, come in questo esercizio. Con quattro bit di controllo è possibile controllare già 11 bit, la parola di codice è quindi lunga 15 bit. Con k bit di controllo, la parola di codice può essere lunga $2^k - 1$ bit. Per messaggi più lunghi sono quindi necessari solo pochi bit aggiuntivi. Il fatto che con l'aiuto del codice di Hamming sia possibile correggere solo un errore per messaggio è in molti casi sufficiente. In informatica si utilizzano anche codici che consentono di correggere più errori.

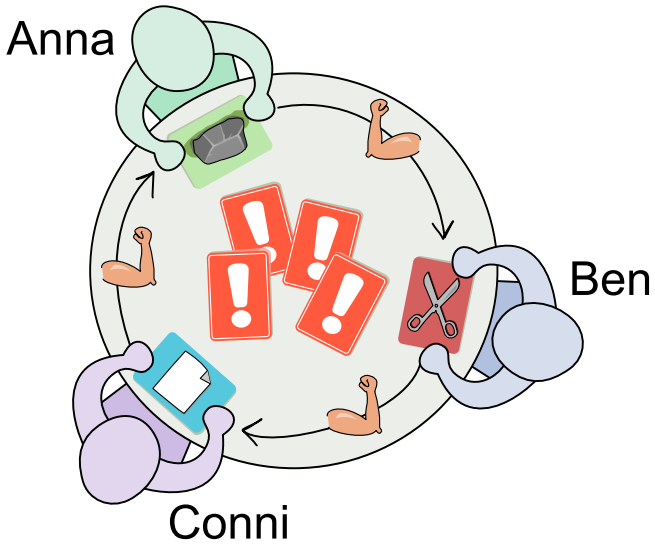
Parole chiave e siti web

- Codice di Hamming: `https://it.wikipedia.org/wiki/Codice_di_Hamming`
- Bit di parità: `https://it.wikipedia.org/wiki/Bit_di_parità`

37. Sasso, carta, forbici

Anna, Ben e Conni giocano a una partita di sasso-carta-forbici. Giocano a coppie, l'uno contro l'altro, contemporaneamente.

Tutti hanno estratto una carta da gioco: Anna ha il sasso  , Ben ha le forbici  , e Conni ha la carta  :



Si applicano le regole classiche: Sasso batte forbici, forbici batte carta, carta batte sasso. Per come sono distribuite le carte ora, tutti vincono una volta e perdono una volta. Ad esempio, Ben vince contro Conni e perde contro Anna.

Prima di analizzare il risultato di ogni coppia di giocatori, tuttavia, è necessario selezionare una delle quattro carte azione presenti al centro del tavolo e svolgere la relativa azione. Le azioni comportano lo scambio di carte tra due giocatori più volte. A ogni scambio, è possibile prendere una nuova decisione su quali due giocatori scambieranno carte tra loro, a meno che l'azione non dica il contrario.

Ben vuole vincere contro Conni a tutti i costi, indipendentemente da come viene eseguita l'azione sulla carta azione scelta.

Solo una delle quattro azioni lo garantisce. Quale?

A) Ben e Conni si scambiano le carte un numero dispari di volte.

B) Due giocatori si scambiano le carte un numero qualsiasi di volte, ma mai Ben con Conni.

C) Due giocatori si scambiano le carte un numero qualsiasi di volte, tra cui Ben almeno una volta con Conni.

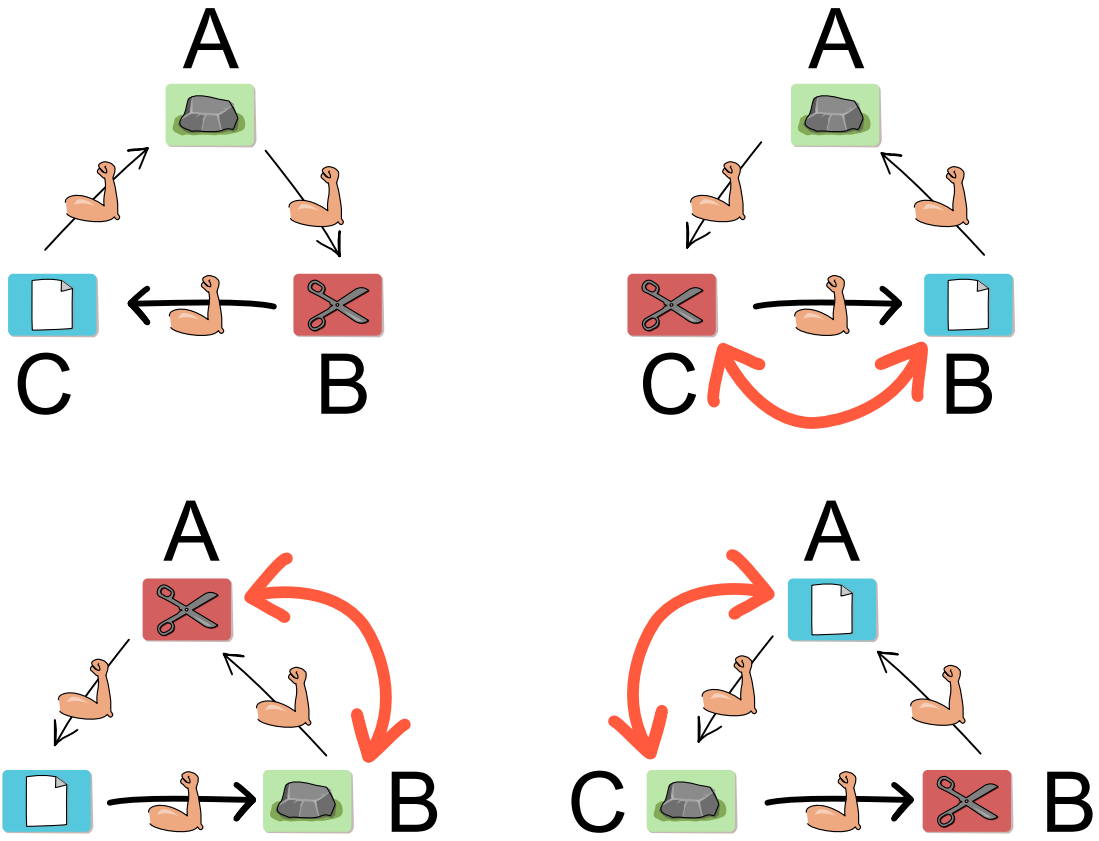
D) Due giocatori si scambiano le carte un numero pari di volte.

Soluzione

La risposta D è corretta.

In primo luogo, osserviamo che Ben vincerebbe contro Conni con la sua carta da gioco originale (forbici batte carta). Ben vuole mantenere questo vantaggio. Tre delle quattro carte azione comportano il rischio che Ben perda contro Conni. Solo la carta azione della risposta D non posta **in nessun caso** alla situazione nella quale Ben perda da Conni.

Una constatazione è utile: tutte e tre le carte da gioco sono diverse e le regole dicono che ogni carta vince contro un'altra e perde contro un'altra. Se i giocatori sono seduti in cerchio, ognuno vince contro un vicino e perde contro l'altro. Uno scambio cambia i risultati di tutti gli abbinamenti, indipendentemente da quali due giocatori si scambiano le carte:



Dopo un numero pari di scambi, viene ripristinata la situazione originale. Con un numero dispari di scambi, invece, i risultati finali vengono modificati. Poiché Ben vince contro Conni all'inizio, possiamo garantire che vinca ancora contro Conni dopo che l'azione è stata eseguita solo se si verifica un numero pari di scambi. Ciò è garantito solo con l'azione della risposta D.

Questa è l'informatica!

Nel rispondere a questo compito, è stato innanzitutto importante rendersi conto che i risultati di tutte le coppie di giocatori sono completamente invertiti da un singolo scambio. E questo accade anche se la coppia di giocatori non scambia le proprie carte. Il risultato tra Ben e Conni cambia quindi anche

se Ben e Conni non si scambiano tra loro, ma Ben scambia con Anna o Conni scambia con Anna. Un *cambiamento locale* tra una coppia ha quindi anche un *effetto globale* su tutte le coppie. Nello sviluppo di programmi informatici, questi *effetti collaterali* possono causare problemi, soprattutto se si verificano involontariamente. Questo può compromettere l'affidabilità dei programmi.

La sicurezza e l'affidabilità dei sistemi informatici svolgono un ruolo centrale nell'informatica. Ciò è particolarmente evidente nei sistemi il cui malfunzionamento può portare direttamente a gravi danni, come per esempio ai sistemi di controllo dei mezzi di trasporto, alle centrali elettriche o ai sistemi che trattano dati personali.

A tal fine, è importante che i programmi utilizzati per far funzionare il sistema siano *corretti*. Un programma è corretto se termina per ogni input e fornisce un risultato conforme alle specifiche (la correttezza può essere descritta in modo simile anche per gli algoritmi). Per i programmi con una funzione importante si cerca sempre di dimostrarne la correttezza. Tra le altre cose, le *invarianti* svolgono un ruolo importante. Si tratta di condizioni che possono essere formulate con la logica e che sono valide, ad esempio, all'inizio di ogni singola ripetizione nel caso di istruzioni ripetute. In questo compito, l'invariante «Ben vince contro Conni» vale se ogni ripetizione di scambi di carte contiene due scambi e quindi in totale viene effettuato un numero pari di scambi di carte.

Parole chiave e siti web

- Invariante: `https://it.wikipedia.org/wiki/Invariante_(informatica)`
- Teoria dei giochi: `https://it.wikipedia.org/wiki/Teoria_dei_giochi`

Compiti di programmazione

I compiti di programmazione seguenti fanno parte dei compiti bonus del concorso.

Mentre i compiti di base non hanno prerequisiti informatici, questi compiti sono più facili da risolvere se si ha qualche conoscenza di programmazione.

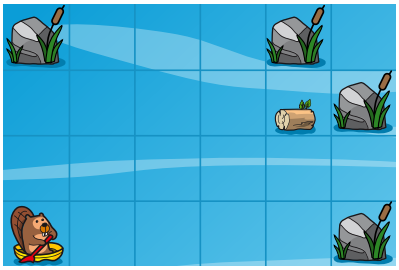
Poiché la programmazione su carta non è molto pratica, per ogni compito viene fornito un codice QR che consente di risolverlo online in modo interattivo.

38. Un solo percorso

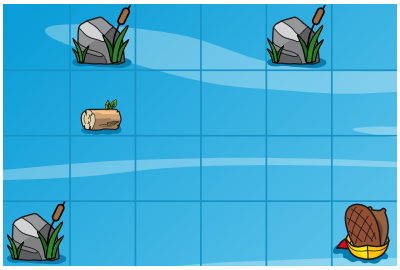
Bea la castora vuole raccogliere dei tronchi nei laghi del Seeland. Per non dover ricordare troppe cose, vuole seguire sempre lo stesso percorso. Aiuta Bea a trovare le istruzioni corrette. Clicca sui cerchi numerati sotto il lago per passare da un lago all'altro.

Puoi usare le seguenti istruzioni:

Istruzione	**Descrizione**
`turnRight()` / `turnLeft()`	Bea ruota sul posto verso destra / sinistra.
`paddle()`	Bea rema finché si trova davanti a un masso. Se si trova su una casella con un tronco, lo raccoglie.



Lago 1



Lago 2

Scrivi un programma per raccogliere il tronco in entrambi i laghi.

Soluzione

La soluzione corretta è la seguente:

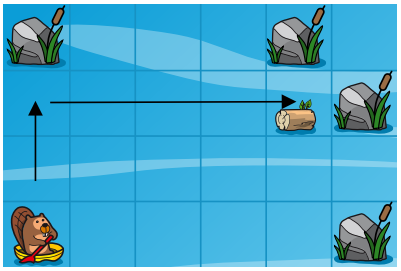
```
turnLeft()
paddle()
turnRight()
paddle()
```

Per raggiungere il tronco nel primo lago, oltre alla soluzione data, esiste anche un'altra possibilità:

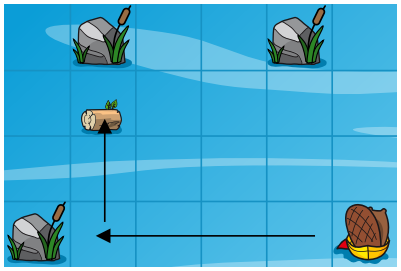
```
paddle()
turnRight()
paddle()
```

Questa soluzione sembra, a prima vista, la più semplice. Tuttavia, con essa non è possibile raccogliere il tronco nel secondo lago, perché il castoro, a causa del comando paddle(), arriva direttamente alla riva.

Una buona strategia di soluzione, quando ci sono più laghi, è quindi quella di osservarli per primi, in modo da tenere conto, nella pianificazione del percorso, della posizione dei massi e dei tronchi in entrambi i laghi.



Lago 1



Lago 2

Questa è l'informatica!

L'informatica si occupa spesso di astrazione, cioè della semplificazione di sistemi e processi complessi. Programmare permette di suddividere problemi complicati in parti più piccole e di risolverli in modo sistematico. La programmazione insegna un modo di pensare strutturato, che favorisce un approccio metodico ai problemi. Spesso cerchiamo una soluzione che possa essere utilizzata per diversi problemi simili.

In questo esempio, quindi, dobbiamo trovare una soluzione che funzioni non solo per un singolo caso, ma per più situazioni diverse.

Parole chiave e siti web

- Programmazione: `https://it.wikipedia.org/wiki/Programmazione_(informatica)`
- Sequenza: `https://it.wikipedia.org/wiki/Struttura_di_controllo`

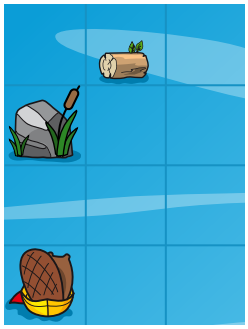
39. La secca pazza

Benno il castoro vuole raccogliere un tronco nel lago. Benno ha scoperto che una secca nel lago sposta ogni volta il masso in punti diversi. Aiuta Benno a scrivere le istruzioni per poter raccogliere il tronco, qualunque sia la posizione del masso. Clicca sui cerchi numerati sotto il lago per vedere le diverse posizioni del masso.

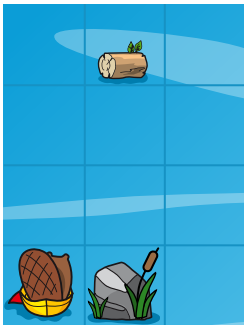
Puoi usare le seguenti istruzioni:

Istruzione	**Descrizione**
`move()`	Benno si muove in avanti di una casella nella direzione in cui guarda.
`turnRight()` / `turnLeft()`	Benno ruota sul posto di 90 gradi verso destra / sinistra.
`removeLog()`	Benno rimuove il tronco dalla casella su cui si trova.

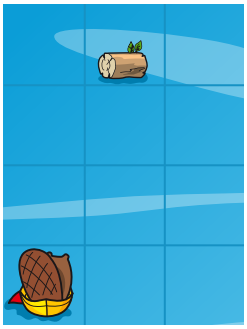
Lago 1



Lago 2



Lago 3



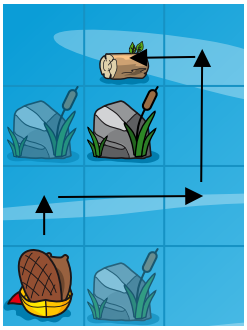
Lago 4

Scrivi un programma con il qualle Benno possa sempre raccogliere il tronco.

Soluzione

La soluzione corretta è la seguente:

```
move()
turnRight()
move()
move()
turnLeft()
move()
move()
turnLeft()
move()
removeLog()
```



Dato che la secca si sposta, non è possibile prendere il percorso diretto verso il tronco:

```
move()
move()
move()
turnRight()
move()
removeLog()
```

Questa soluzione sembra, a prima vista, la più breve, perché permette di raggiungere il tronco non solo nel lago 1, ma anche nei laghi 2 e 3. Tuttavia, nel quarto lago il tronco non può essere raccolto, perché il castoro finisce direttamente contro un masso. Naturalmente potremmo adattare il programma per permettere di raccogliere il tronco nel quarto lago. Ma così rischieremmo di ottenere una soluzione che non permette più di raccogliere il tronco in uno degli altri laghi.

Una buona strategia quando ci sono più laghi è quindi quella di osservarli sempre per primi, in modo da tenere conto, nella pianificazione del percorso, della posizione dei massi e dei tronchi in tutti i laghi.

Questa è l'informatica!

L'informatica si occupa spesso di astrazione, cioè della semplificazione di sistemi e processi complessi. Programmare permette di suddividere problemi complicati in parti più piccole e di risolverli in modo sistematico. La programmazione insegna un modo di pensare strutturato, che favorisce un approccio metodico ai problemi. Spesso cerchiamo una soluzione che possa essere utilizzata per diversi problemi simili.

In questo esempio, quindi, dobbiamo trovare una soluzione che funzioni non solo per un singolo caso, ma per più situazioni diverse.

Parole chiave e siti web

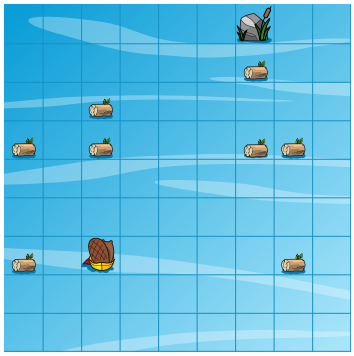
- Programmazione: `https://it.wikipedia.org/wiki/Programmazione_(informatica)`
- Sequenza: `https://it.wikipedia.org/wiki/Struttura_di_controllo`

40. Il tronco prezioso

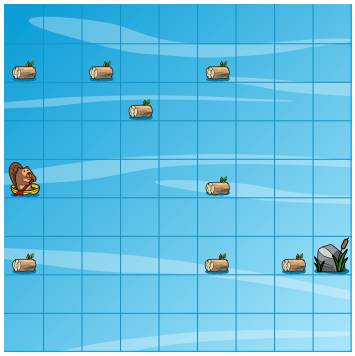
Petunia il castoro è alla ricerca di tronchi preziosi nella regione del Seeland. Il tronco più prezioso si trova sempre proprio accanto a un masso. Aiuta Petunia a scrivere le istruzioni per arrivare, in tutti e tre i laghi, alla casella con il tronco prezioso vicino al masso. Le istruzioni devono funzionare per ogni lago. Usa il minor numero possibile di istruzioni. Clicca sui cerchi numerati sotto il lago per passare da un lago all'altro.

Puoi usare le seguenti istruzioni:

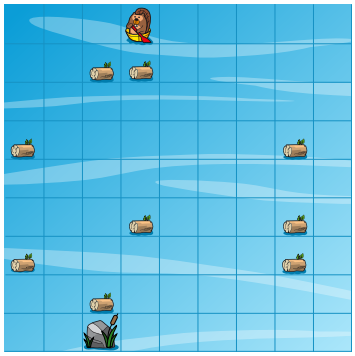
Istruzione	**Descrizione**
`move()`	Petunia si muove in avanti di una casella nella direzione in cui guarda.
`turnRight()` / `turnLeft()`	Petunia ruota sul posto di 90 gradi verso destra / sinistra.
`goToLog()`	Petunia avanza finché arriva su una casella con un tronco.



Lago 1



Lago 2



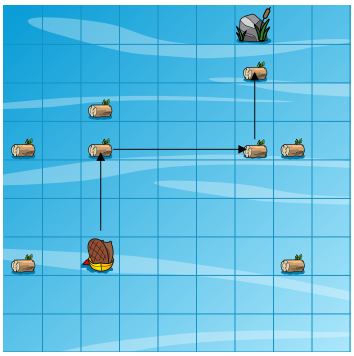
Lago 3

Scrivi un programma per arrivare al tronco prezioso vicino al masso in tutti e tre i laghi. Cerca di usare il minor numero possibile di istruzioni.

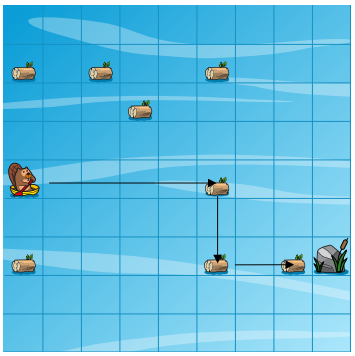
Soluzione

La soluzione corretta è la seguente:

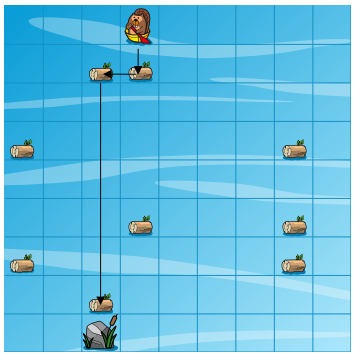
```
goToLog()
turnRight()
goToLog()
turnLeft()
goToLog()
```



Lago 1



Lago 2



Lago 3

Per utilizzare il minor numero possibile di istruzioni, è necessario usare il comando `goToLog()`. In questo modo possiamo sempre passare da un tronco all'altro, indipendentemente da quanto distano tra loro.

Se per il lago 1 usassimo invece il comando `move()`, arriveremmo comunque direttamente al tronco prezioso:

```
move()
move()
move()
turnRight()
move()
move()
move()
move()
turnLeft()
move()
move()
```

Tuttavia, in questo modo non otteniamo il programma che utilizza il minor numero di istruzioni, perché il programma risulta molto più lungo rispetto alla soluzione cercata.

Sorge anche un altro problema: con il programma più lungo, il castoro non raggiunge più il tronco prezioso nel secondo e nel terzo lago. Solo con il comando `goToLog()` riusciamo ad arrivare al tronco in tutti e tre i laghi usando la stessa sequenza di istruzioni, indipendentemente dalla distanza tra il castoro e il tronco nei diversi laghi.

Questa è l'informatica!

L'informatica si occupa spesso di astrazione, cioè della semplificazione di sistemi e processi complessi. Programmare permette di suddividere problemi complicati in parti più piccole e di risolverli in modo sistematico. La programmazione insegna un modo di pensare strutturato, che favorisce un approccio metodico ai problemi. Spesso cerchiamo una soluzione che possa essere utilizzata per diversi problemi simili.

In questo esempio dobbiamo quindi trovare una soluzione che funzioni non solo per un singolo caso, ma per più situazioni diverse. Le soluzioni programmate che funzionano solo per un caso specifico vengono chiamate Hardcode. Spesso cerchiamo di evitare che una soluzione sia hardcodata e preferiamo scrivere programmi utilizzabili per diversi problemi simili.

Invece di contare quanti passi debba fare Petunia in avanti, utilizziamo il comando `goToLog()`, che si basa su una struttura di ciclo (qui: muoviti in avanti finché non ti trovi su una casella con un tronco). Questo rende inutile il conteggio delle caselle e permette di scrivere una versione più breve ed efficace del programma.

Parole chiave e siti web

- Programmazione: `https://it.wikipedia.org/wiki/Programmazione_(informatica)`
- Sequenza: `https://it.wikipedia.org/wiki/Struttura_di_controllo`
- Iterazione: `https://it.wikipedia.org/wiki/Iterazione`

41. Ancora più legno

Linus il castoro ha bisogno di tanto legno. Purtroppo Linus non sa ancora remare molto bene. Quando comincia a remare, continua sempre fino a fermarsi davanti a un masso. Aiuta Linus a trovare un percorso per raccogliere il maggior numero possibile di tronchi.

Puoi usare le seguenti istruzioni:

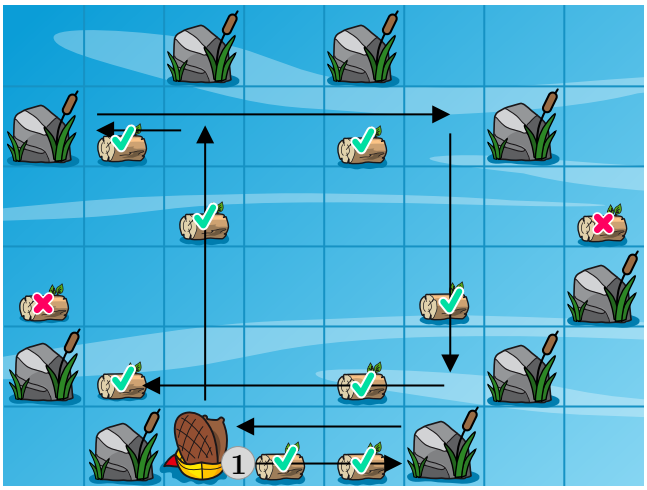
Istruzione	**Descrizione**
`turnRight()` / `turnLeft()`	Linus ruota sul posto di 90 gradi verso destra / sinistra.
`paddle()`	Linus rema in avanti finché non si trova davanti a un masso. Se rema sopra un tronco d'albero, lo raccoglie.

Scrivi un programma per raccogliere il maggior numero possibile di tronchi.

Soluzione

La soluzione corretta è la seguente:

```
turnRight()
paddle()
turnRight()
turnRight()
paddle()
turnRight()
paddle()
turnLeft()
paddle()
turnRight()
turnRight()
paddle()
turnRight()
paddle()
turnRight()
paddle()
```



Poiché possiamo eseguire un movimento in avanti solo tramite il comando `paddle()`, ogni avanzamento termina automaticamente appena prima di un masso. A causa di questa limitazione, è possibile raccogliere al massimo 8 tronchi.

Se oltre al comando `paddle()` avessimo avuto un altro comando che permettesse a Linus di fare un singolo passo, sarebbe stato possibile raccogliere più tronchi. Per farlo, però, sarebbe servito anche un ulteriore comando specifico per raccogliere i tronchi.

Questa è l'informatica!

Il problema è composto da diverse parti: nella prima parte ci occupiamo della ricerca di un percorso. In matematica come in informatica, questo tipo di situazione viene spesso descritto come un problema di grafi. A esso sono spesso collegati problemi di ottimizzazione. Infatti, in questo compito non si tratta solo di trovare il percorso corretto, ma anche di determinare il numero massimo di tronchi che possono essere raccolti lungo il percorso. Problemi di ottimizzazione ben noti sono, per esempio, il problema del commesso viaggiatore.

Nella seconda parte, dedicata alla programmazione, ci chiediamo come descrivere in modo preciso il percorso trovato in precedenza. È prima di tutto importante capire come funziona il comando `paddle()`. Dietro questo comando si nasconde un ciclo condizionato: gli stessi comandi vengono ripetuti finché una certa condizione è soddisfatta.

Il castoro Petunia controlla inizialmente se il percorso è libero. Se lo è — cioè se non c'è un masso nella casella direttamente davanti a lei — si muove avanti di una casella e ricomincia lo stesso processo di controllo. In questo modo è possibile coprire qualsiasi distanza fino a un masso.

Parole chiave e siti web

- Programmazione: `https://it.wikipedia.org/wiki/Programmazione_(informatica)`
- Sequenza: `https://it.wikipedia.org/wiki/Struttura_di_controllo`
- Iterazione: `https://it.wikipedia.org/wiki/Iterazione`
- TSP: `https://it.wikipedia.org/wiki/Problema_del_commesso_viaggiatore`

42. Intorno agli scogli

Benno il castoro vuole raccogliere dei tronchi nella regione del Seeland. Scrivi un unico programma con il quale il castoro possa raccogliere il tronco in tutti i laghi. Clicca sui cerchi numerati sotto il lago per passare da un lago all'altro.

Puoi usare le seguenti istruzioni:

Istruzione	**Descrizione**
`move()`	Benno si muove in avanti di una casella nella direzione in cui guarda.
`turnRight()` / `turnLeft()`	Benno ruota sul posto di 90 gradi verso destra / sinistra.
`removeLog()`	Benno rimuove il tronco dalla casella su cui si trova. Può raccogliere quanti tronchi vuole.

`while ...:`
`  Istruzione`
`  Istruzione`
	Benno ripete le istruzioni rientrate finché una condizione è soddisfatta. Fai attenzione ai due punti alla fine della prima riga.

Nel seguente esempio, Benno si muove in avanti di una casella finché c'è un masso alla sua destra. Quando non lo è più, continua con l'istruzione successiva non «rientrata», come in questo esempio, nel quale viene eseguita solo `turnRight()`:

```
while rockRight():
    move()
turnRight()
```

Lago 1

Lago 2

Lago 3

Scrivi le istruzioni per raccogliere il tronco in tutti i laghi. Le righe da 1 a 3 sono già scritte e non possono essere modificate.

Soluzione

La soluzione corretta è la seguente:

```
while rockRight():
    move()
turnRight()

move()
move()
move()
turnRight()
move()

while rockRight():
    move()
    move()

removeLog()
```

Una prima esecuzione del programma con le righe 1–3 già fornite mostra che Benno avanza finché alla sua destra non c'è più un masso e poi effettua una rotazione verso destra. Possiamo ora suddividere il resto del codice in due parti.

Per prima cosa dobbiamo guidare Benno intorno alla fila orizzontale superiore dei massi. Poiché questa fila è larga esattamente due massi in tutti i laghi, possiamo coprire questa distanza eseguendo tre volte il comando `move()`.

Per il tratto verso il basso, invece, non è sufficiente contare il numero di passi, perché la distanza dal tronco è diversa nei tre laghi. Abbiamo quindi nuovamente bisogno della struttura di controllo `while` per eseguire uno o più comandi in modo ripetuto.

A differenza della fila di massi sulla sinistra, nella fila di destra c'è sempre esattamente una casella vuota tra un masso e l'altro. Questo significa che, affinché la condizione `rockRight()` sia di nuovo soddisfatta, Benno deve avanzare di due caselle.

Questa è l'informatica!

L'informatica si occupa spesso di astrazione, cioè della semplificazione di sistemi e processi complessi. Programmare permette di suddividere problemi complicati in parti più piccole e di risolverli in modo sistematico. La programmazione insegna un modo di pensare strutturato, che favorisce un approccio metodico ai problemi. Spesso cerchiamo una soluzione che possa essere utilizzata per diversi problemi simili.

In questo esempio dobbiamo trovare una soluzione che funzioni non solo per un singolo caso, ma per più situazioni diverse. Le soluzioni programmate che funzionano solo per un caso specifico vengono

chiamate Hardcode. Spesso cerchiamo di evitare che una soluzione sia hardcodata e preferiamo scrivere programmi validi per diversi problemi simili.

Grazie alla struttura di controllo `while` combinata con la condizione `rockRight()`, è possibile avviare un movimento in avanti indipendentemente dal numero di massi presenti, movimento che termina solo quando la condizione non è più soddisfatta. Nel caso concreto, il movimento si interrompe quando alla destra di Benno non si trova più alcun masso. Questa struttura di controllo viene anche chiamata ciclo condizionale.

Parole chiave e siti web

- Programmazione: `https://it.wikipedia.org/wiki/Programmazione_(informatica)`
- Sequenza: `https://it.wikipedia.org/wiki/Struttura_di_controllo`
- Iterazione: `https://it.wikipedia.org/wiki/Iterazione`

43. Avanti e indietro

Benno il castoro vuole raccogliere dei tronchi in alcuni laghi della sua regione. Per farlo ha bisogno di aiuto nel definire un programma singolo con il quale possa raccogliere i tronchi in tutti i laghi. Clicca sui numeri cerchiati sotto il lago per passare da un lago all'altro.

Puoi usare le seguenti istruzioni:

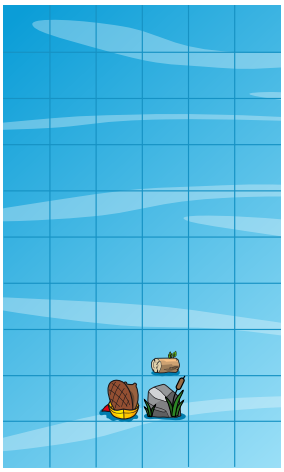
Istruzione	**Descrizione**
`move()`	Benno si muove in avanti di una casella nella direzione in cui guarda.
`turnRight()` / `turnLeft()`	Benno ruota sul posto di 90 gradi a destra / sinistra.
`removeLog()`	Benno rimuove il tronco dalla casella su cui si trova. Può raccogliere quanti tronchi vuole.

`while ...:`
`  Istruzione`
`  Istruzione`
	Benno ripete le istruzioni rientrate finché una condizione è soddisfatta. Fai attenzione ai due punti alla fine della prima riga.

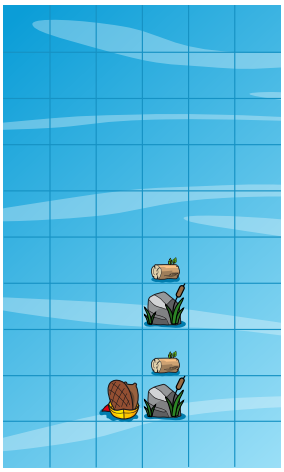
`rockRight()`	*Condizione*: Controlla se c'è una roccia alla destra di Benno.
`rockLeft()`	*Condizione*: Controlla se c'è una roccia alla sinistra di Benno.

Nel seguente esempio, Benno si muove in avanti di una casella finché c'è un masso alla sua sinistra. Quando non lo è più, continua con l'istruzione successiva non «rientrata», come in questo esempio, nel quale viene eseguita solo `turnLeft()`:

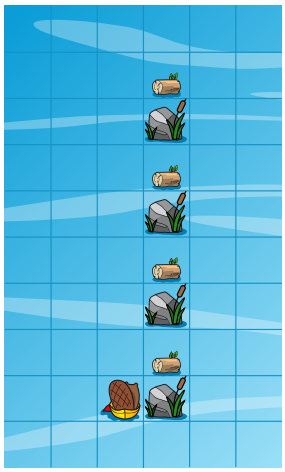
```
while rockLeft():
    move()
turnLeft()
```



Lago 1



Lago 2



Lago 3

Scrivi un programma per raccogliere tutti i tronchi in tutti i laghi.

Soluzione

La soluzione corretta è la seguente:

```
while rockRight():
    move()
    turnRight()
    move()
    removeLog()
    turnLeft()
    turnLeft()
    move()
    turnRight()
    move()
```

Poiché vogliamo scrivere un programma valido per tutti e tre i laghi, ci rendiamo subito conto che non possiamo trovare la soluzione semplicemente contando i singoli passi e movimenti. Certamente questo metodo funziona per un singolo lago, ma non funzionerà negli altri, perché il numero di massi e di tronchi cambia.

Confrontando i laghi tra loro, si riconosce però uno schema composto da un masso e da un tronco subito dietro. Questo schema si ripete in entrambi i mondi, ma un numero diverso di volte.

Riconoscere questo schema porta all'uso della struttura di controllo `while`, con la quale è possibile eseguire ripetutamente dei comandi finché una condizione è soddisfatta. Nella sua posizione iniziale, in tutti e tre i mondi c'è un masso alla destra di Benno. Se prendiamo quindi `rockRight()` come condizione del ciclo, sappiamo che questa condizione è immediatamente soddisfatta e che le istruzioni con rientro (il corpo del ciclo) verranno eseguite almeno una volta.

Poiché lo schema masso–tronco si ripete, nel corpo del ciclo dobbiamo semplicemente fare in modo che Benno termini il suo movimento di nuovo con un masso alla sua destra, in modo che la condizione `rockRight()` sia nuovamente soddisfatta e il corpo del ciclo venga eseguito un'altra volta. Non appena nella posizione finale non c'è più un masso alla destra di Benno, cioè quando la condizione non è più soddisfatta, il corpo del ciclo non viene più eseguito e l'esecuzione del programma termina.

Questa è l'informatica!

L'informatica si occupa spesso di astrazione, cioè della semplificazione di sistemi e processi complessi. Programmare permette di suddividere problemi complicati in parti più piccole e di risolverli in modo sistematico. La programmazione insegna un modo di pensare strutturato, che favorisce un approccio metodico ai problemi. Spesso cerchiamo una soluzione che possa essere utilizzata per diversi problemi simili.

In questo esempio dobbiamo trovare una soluzione che funzioni non solo per un singolo caso, ma per più situazioni diverse. Le soluzioni programmate che funzionano solo per un caso specifico vengono

chiamate Hardcode. Spesso cerchiamo di evitare che una soluzione sia hardcodata e preferiamo scrivere programmi utilizzabili per più problemi simili.

Grazie alla struttura di controllo `while` in combinazione con la condizione scelta, è possibile eseguire dei comandi in modo ripetuto finché tale condizione risulta nuovamente soddisfatta dopo l'esecuzione del corpo del ciclo (ciclo con test iniziale). Se riconosciamo inoltre lo schema sottostante (qui: masso e tronco che si alternano), possiamo generalizzare la soluzione e ottenere così un approccio che permette di risolvere anche altre istanze dello stesso problema (qui: diversi laghi) senza dover modificare il codice del programma.

Parole chiave e siti web

- Programmazione: `https://it.wikipedia.org/wiki/Programmazione_(informatica)`
- Sequenza: `https://it.wikipedia.org/wiki/Struttura_di_controllo`
- Iterazione: `https://it.wikipedia.org/wiki/Iterazione`

A. Autori dei quesiti

James Atlas
Masiar Babazadeh
Angeni Bai
Leonardo Barichello
Wilfried Baumann
Susanne Berchtold
Maksim Bolonkin
Leonardo Cavalcante
Maria Cepeda
Špela Cerar
Gi Soong Chee
Byeonggyu Cho
Anton Chukhnov
Vladimir Costas
Andrew Csizmadia
Valentina Dagienė
Darija Dasović
Christian Datzko
Justina Dauksaite
Diane Dowling
Nora A. Escherle
Abeer Eshra
Gerald Futschek
Christian Giang
Vernon Gutierrez
Silvan Horvath
Alisher Ikramov
Thomas Ioannou
Asterios Karagiannis
Blaž Kelvišar
David Khachatryan
Doyong Kim
Jihye Kim
Dong Yoon Kim
Vaidotas Kinčius
Stefan Koch
Jia-Ling Koh
Sophie Koh
Víctor Koleszar
Lukas Lehner
Taina Lehtimäki
Gunwoong Lim
Linda Mannila
Yoshiaki Matsuzawa
Hamed Mohebbi
Mattia Monga
Anna Morpurgo
Kamohelo Motloung
Justina Oostendorp
A-Yeong Park
Suchan Park
Gabriela Gomez Pasquali

Jean-Philippe Pellet

Emiliano Pereiro

Emmanuel Plan

Zsuzsa Pluhár

Wolfgang Pohl

Cesar F. Bolanos Revelo

Pedro Ribeiro

Rokas Rimkus

Kirsten Schlüter

Margareta Schlüter

Dirk Schmerenbeck

Vipul Shah

Jacqueline Staub

Alieke Stijf

Nikolaos Stratis

Supawan Tasanaprasert

Susanne Thut

Christine Vender

Florentina Voboril

Michael Weigend

Chris Wetherell

Philip Whittington

Kyra Willekes

Hsu Sint Sint Yee

B. Partner accademici

Haute école pédagogique du canton de Vaud
`http://www.hepl.ch/`

Ausbildungs- und Beratungszentrum für Informatikunterricht der ETH Zürich
`http://www.abz.inf.ethz.ch/`

La Scuola universitaria professionale della Svizzera italiana (SUPSI)
`http://www.supsi.ch/`

Pädagogische Hochschule Zürich
`https://www.phzh.ch/`

Universität Trier
`https://www.uni-trier.de/`

C. Sponsoring

Hasler Stiftung
http://www.haslerstiftung.ch/

Abraxas Informatik AG
https://www.abraxas.ch

Amt für Kindergarten, Volksschule und Beratung, Bildungs- und Kulturdirektion, Cantone di Berna
https://www.bkd.be.ch/de/start/ueber-uns/die-organisation/amt-fuer-kindergarten-volksschule-und-beratung.html

Amt für Wirtschaft, Canton Zurigo
https://www.zh.ch/de/volkswirtschaftsdirektion/amt-fuer-wirtschaft.html

Fondazione Informatica Svizzera
https://informatics-foundation.ch

cyon
https://www.cyon.ch

Senarclens Leu & Partner
http://senarclens.com/

Wealth Management IT and UBS Switzerland IT
http://www.ubs.com/

