INFORMATIK-BIBER SCHWEIZ
CASTOR INFORMATIQUE SUISSE
CASTORO INFORMATICO SVIZZERA

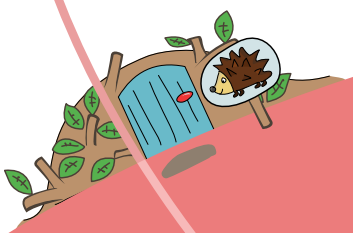
Quesiti e soluzioni 2025

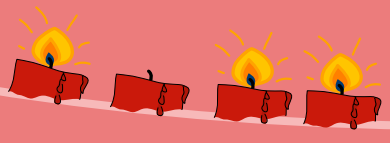
7º e 8º anno scolastico

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

A cura di:

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





01010011010101100100100
10000010010110101010011
01010011010010010100101
00101101010100110101001
01001001010010010100001

SSII

www.svia-ssie-ssii.ch
schweizerischerverein für informatikin
derausbildung//société suisse pour l'infor
matiquedans l'enseignement//società sviz
zera 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

Hanno collaborato al Castoro Informatico 2025 i

Premessa iii

Indice v

1. La macchina dei numeri 1

2. Albero di decisione 5

3. Stella di luci 11

4. Istruzioni di montaggio 15

5. Vasi di fiori 19

6. Dalla foglia al legno 23

7. Arcipelago dei castori 27

8. Lefty II 31

9. Giornata nebbiosa 35

10. Albero genealogico 39

11. Servizio di corriere 41

12. L'aquilone perduto 45

13. Corona dell'Avvento 49

14. Mappa di luminosità 53

15. Scopriamo Seul! 57

16. Laghi di montagna 61

17. Parcheggi 65

18. Ancora più legno 71

A. Autori dei quesiti 74

B. Partner accademici 75

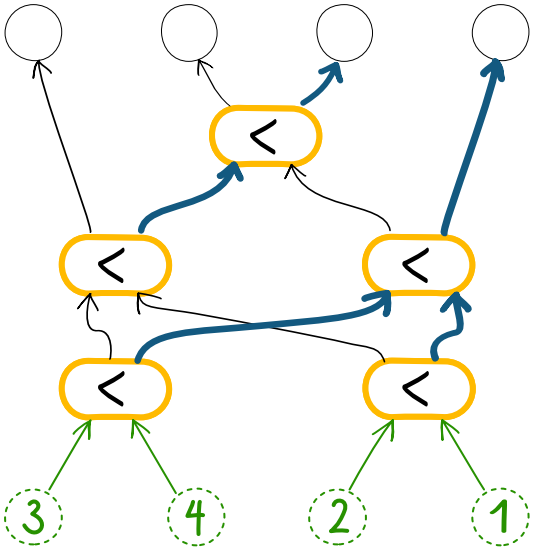
C. Sponsoring 76

1. 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.

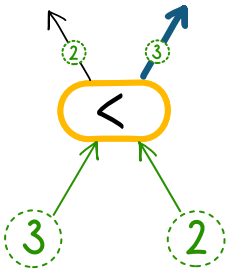
I numeri si muovono verso l'alto nella macchina lungo frecce e snodi < fino ai campi di uscita ◯.



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

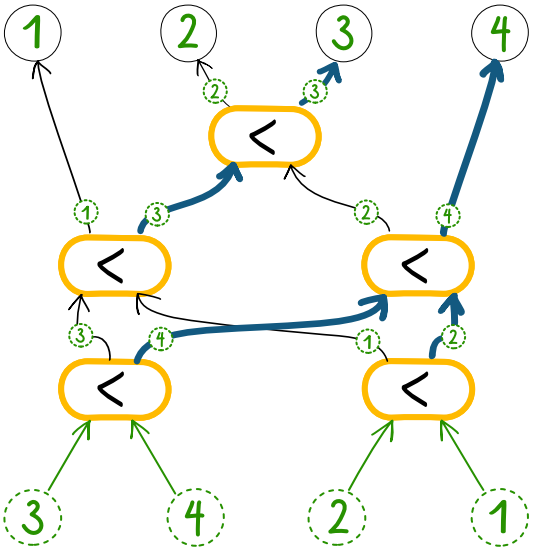
C) Visualizza i numeri nello stesso ordine. Risultato: 3, 4, 2, 1

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

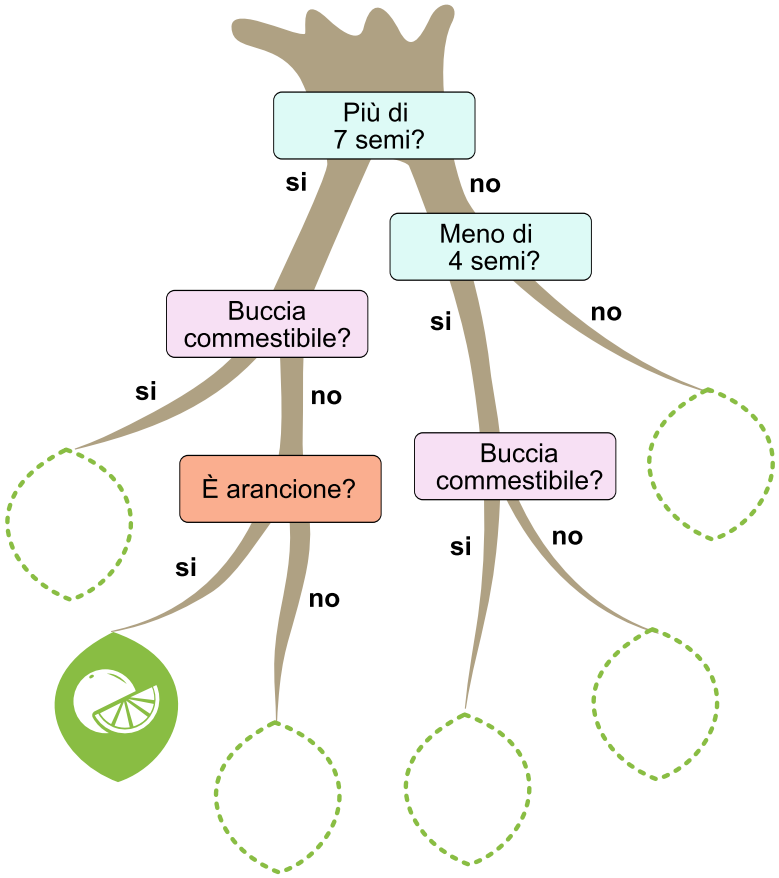
- *Rete di ordinamento*: `https://www.coderdojotrento.it/csun2/`
- *Comparatore*: `https://it.wikipedia.org/wiki/Comparatore_(elettronica)`
- *Calcolo parallelo*: `https://it.wikipedia.org/wiki/Calcolo_parallelo`

2. 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.

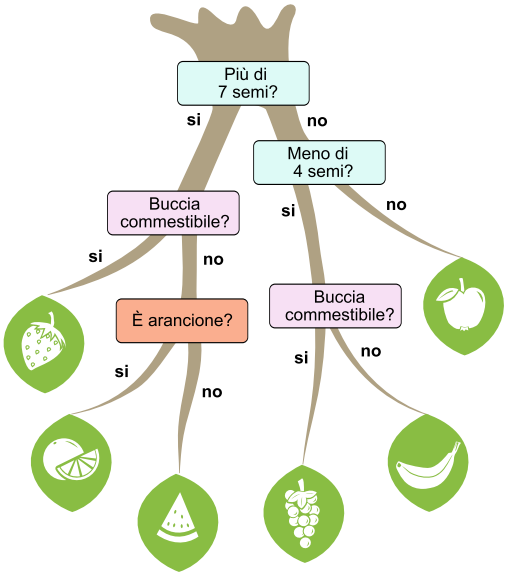


Purtroppo l'albero delle decisioni usato dalla classe si è rotto dopo il decimo frutto e quasi tutte le foglie sono cadute.

Dov'erano le foglie?

Soluzione

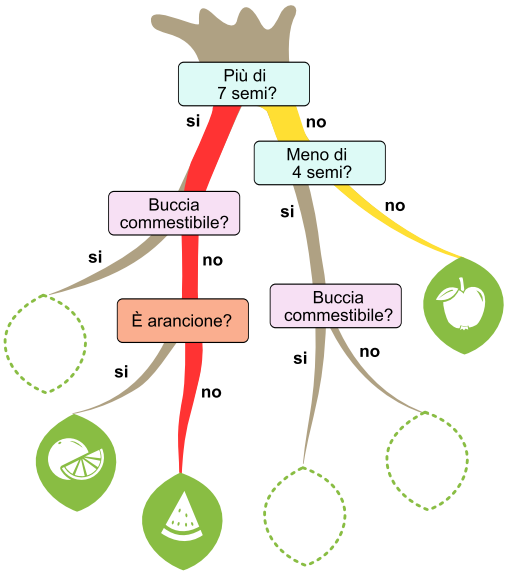
Questa è come si presenta l'albero di decisione una volta aggiustato:



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`

3. Stella di luci

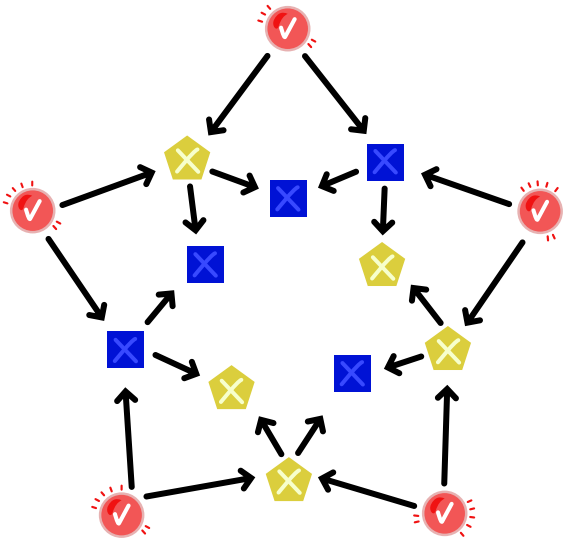
Sophie ha tre tipi di luci nella sua scatola elettronica: rossa rotonda, blu quadrata e gialla pentagonale. Sophie ha deciso di collegare alcune luci per formare una «stella di luci», nella quale le luci rosse sono alle estremità della stella, mentre le luci blu e gialle sono all'interno della stella. 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.

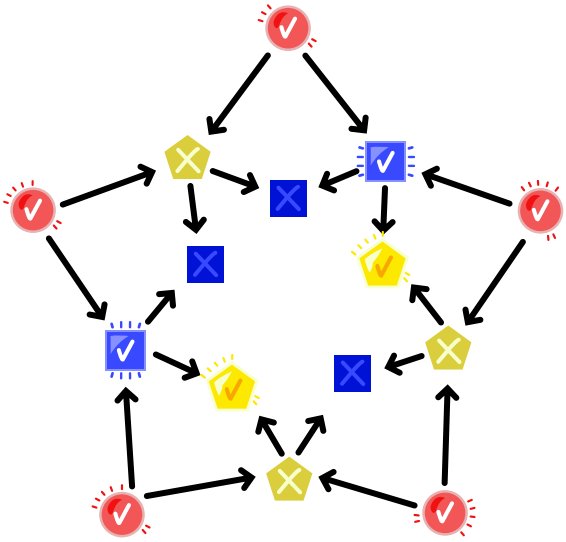
Sophie accende tutte le luci rosse.

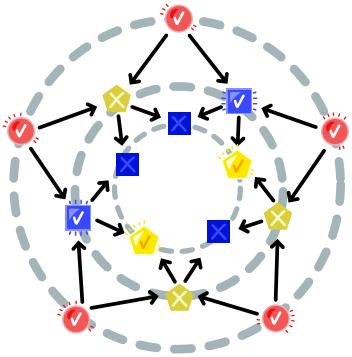
Quali altre luci sono accese?



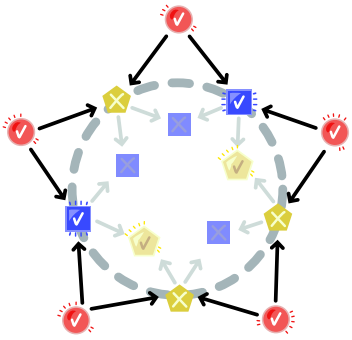
Soluzione

Questa è la risposta:

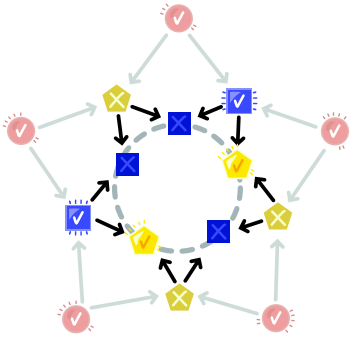




Le luci formano tre anelli: un anello esterno, un anello centrale e un anello interno. L'anello esterno ha solo luci rosse, mentre gli altri due anelli hanno luci sia blu che gialle.



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`

4. Istruzioni di montaggio

Hai questi mattoncini:

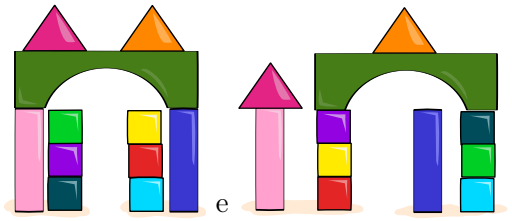
6 cubi | 2 cuboidi | 1 ponte | 2 piramidi

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.

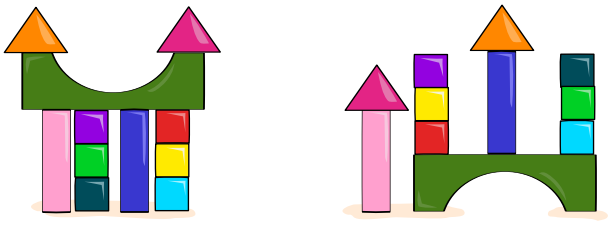
Quando si costruisce, è necessario seguire l'ordine delle sei istruzioni. È comunque possibile costruire molte strutture diverse con le istruzioni di costruzione.

Due esempi:

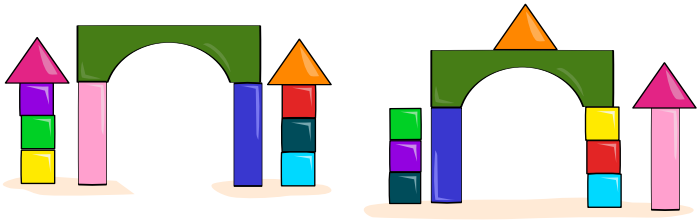


e

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



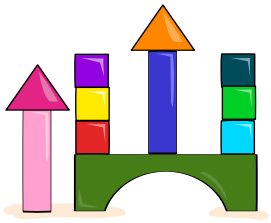
A) B)



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`

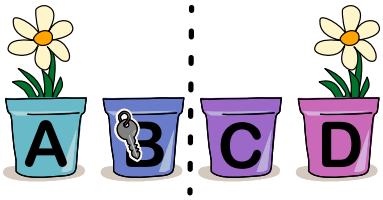
5. Vasi di fiori

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**.

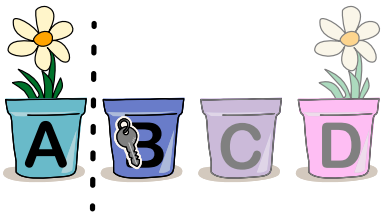
Florian vuole nascondere la chiave di riserva di casa sua in un vaso. Per sapere come trovarla, ci spiega il suo metodo:

«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:

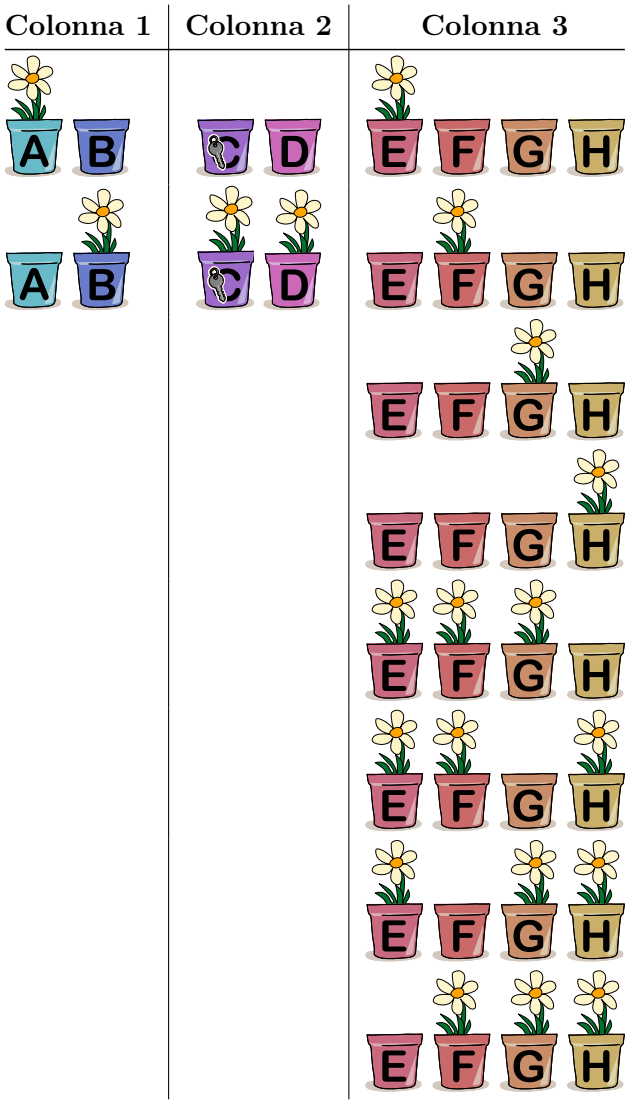
Un'altra risposta corretta è:

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.

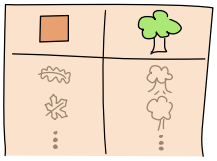
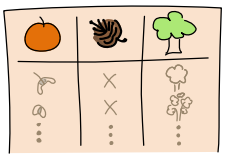
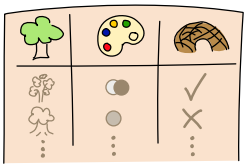
Il modo migliore per trovare la risposta corretta a questo compito è il cosiddetto *approccio bottom-up*. Le parti più piccole della risposta vengono considerate per prime, poiché le parti più grandi dipendono da esse. Invece di provare tutte le possibili combinazioni, che in questo caso sarebbero $2^8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

6. 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.

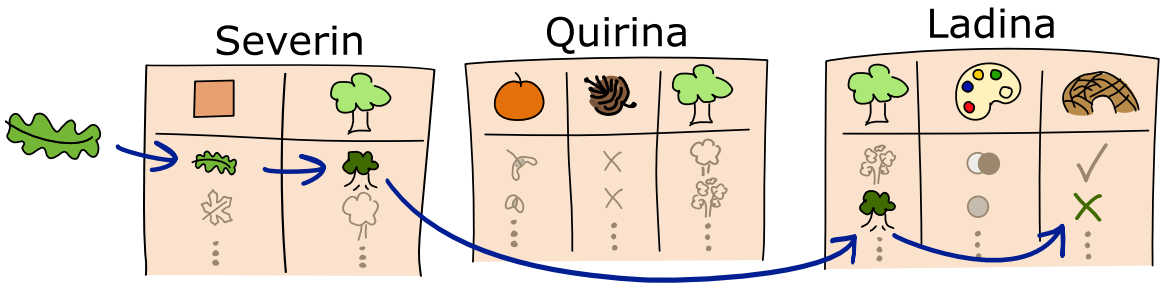
Soluzione

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.

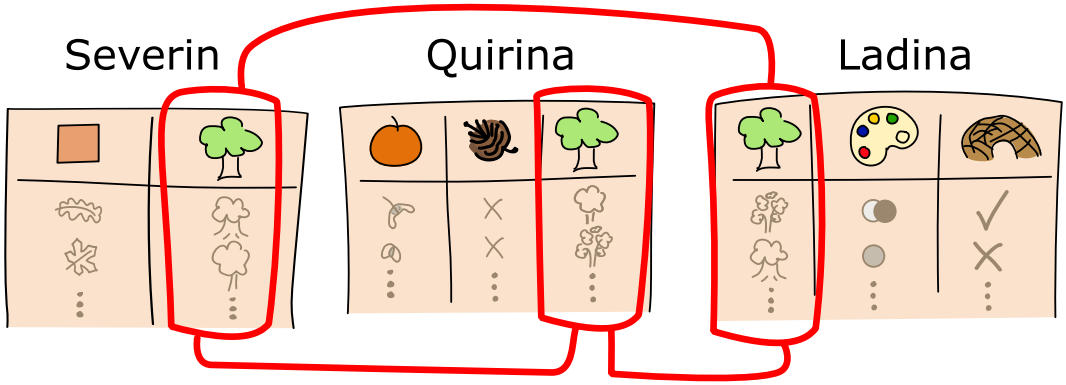
La tabella di Severin contiene informazioni sulle foglie. Siccome Emil conosce la forma della foglia , può prima selezionare la riga appropriata nella tabella di Severin e ottenere le informazioni mancanti sulla specie di albero . Può quindi utilizzare questa informazione per selezionare la riga appropriata 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 :



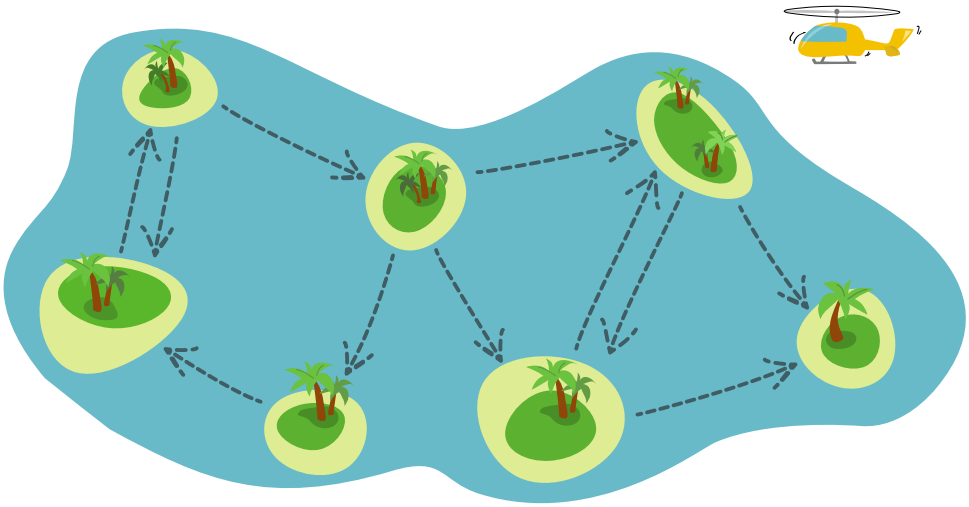
I database relazionali sono così importanti che esiste un linguaggio separato in informatica per interrogare ed eseguire altre operazioni sui database, chiamato *SQL (Structured Query Language)*.

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`

7. 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

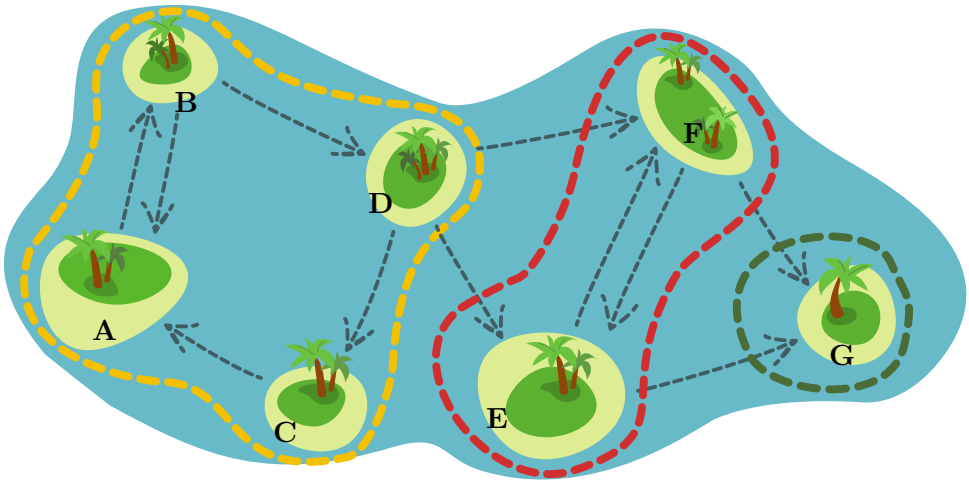
Soluzione

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

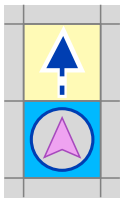
- Grafo diretto: `https://it.wikipedia.org/wiki/Digrafo_(matematica)`
- Grafo connesso: `https://it.wikipedia.org/wiki/Grafo_connesso`
- Componente fortemente connessa: `https://it.wikipedia.org/wiki/Componente_fortemente_connessa`

8. Lefty II

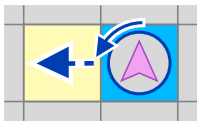
Il robot *Lefty* si muove su una griglia composta da caselle quadrate. Tra le caselle possono esserci dei muri rossi. Lefty deve raggiungere l'obiettivo verde.

Lefty può muoversi solo in due modi:

Avanzare di una casella

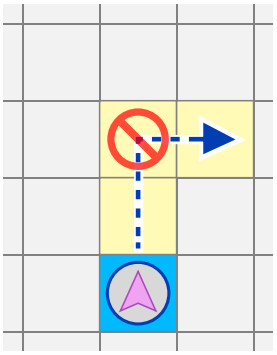


Girare a sinistra e avanzare immediatamente di una casella

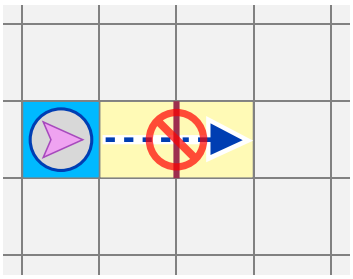


Quindi ci sono azioni che Lefty non può fare:

... **non** può girare a destra e...

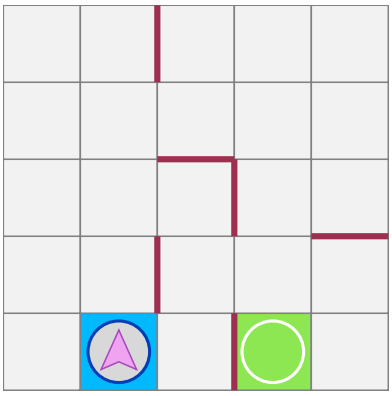


non può passare attraverso i muri.



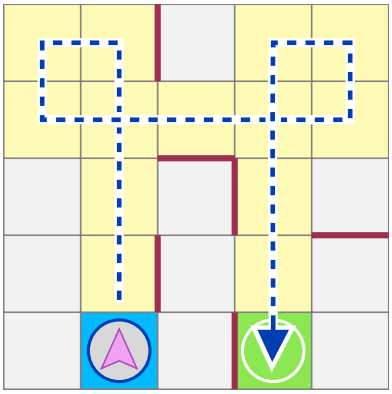
Quali caselle deve attraversare Lefty per raggiungere la destinazione?

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



Soluzione

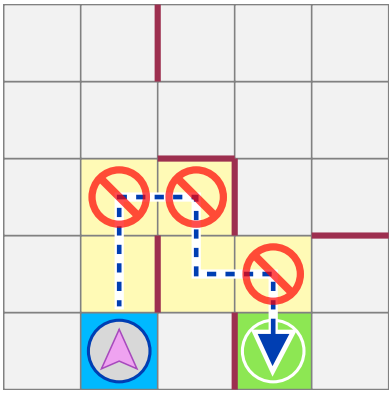
Questa è la risposta:



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.

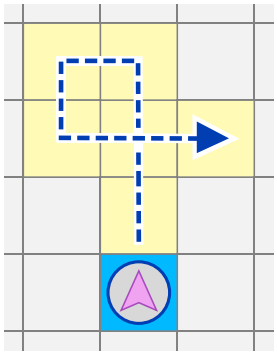


Questa è l'informatica!

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

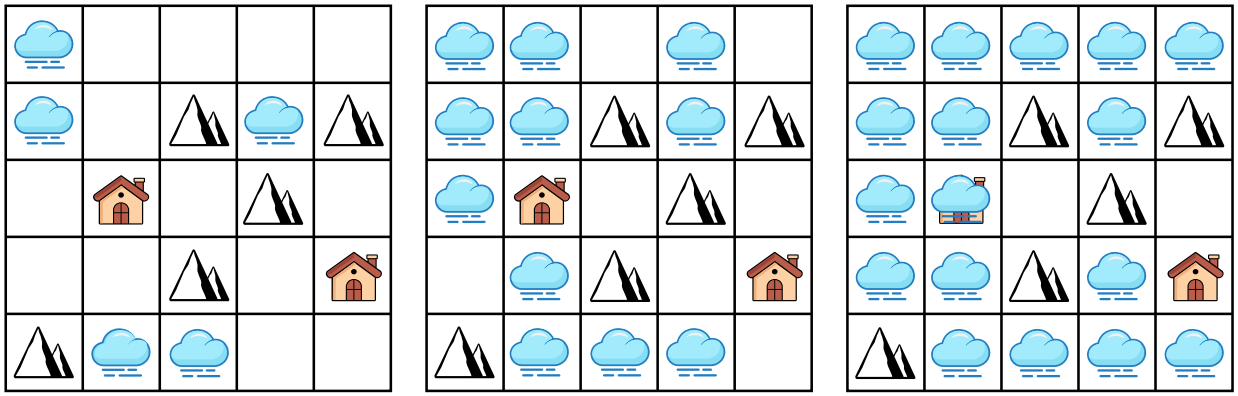
- Processore: `https://it.wikipedia.org/wiki/Processore`
- Instruction set: `https://it.wikipedia.org/wiki/Instruction_set`
- CISC: `https://it.wikipedia.org/wiki/Complex_instruction_set_computer`
- RISC: `https://it.wikipedia.org/wiki/Reduced_instruction_set_computer`

9. Giornata nebbiosa

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

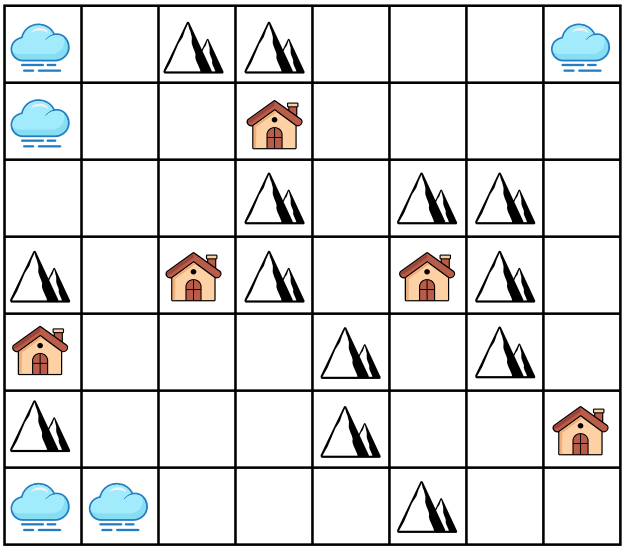
All'alba, la nebbia copre solo alcune regioni. Dopo un'ora, la nebbia si diffonde da ogni regione coperta a tutte le regioni vicine, a destra, a sinistra, in alto o in basso. Anche le case vengono coperte dalla nebbia. Solo le regioni di montagna non possono essere coperte dalla nebbia.

Ecco un esempio:



Alba | Dopo 1 ora | Dopo 2 ore

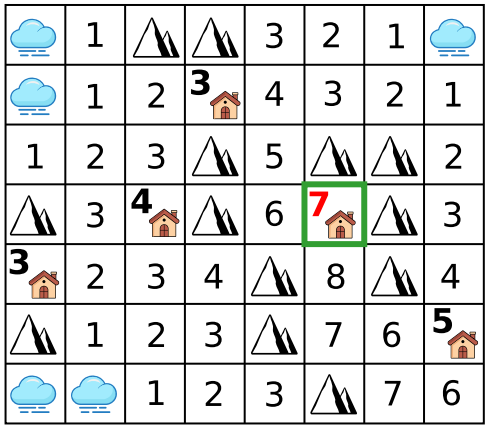
*Quale casa del paese è l'**ultima** ad essere coperta dalla nebbia?*



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.

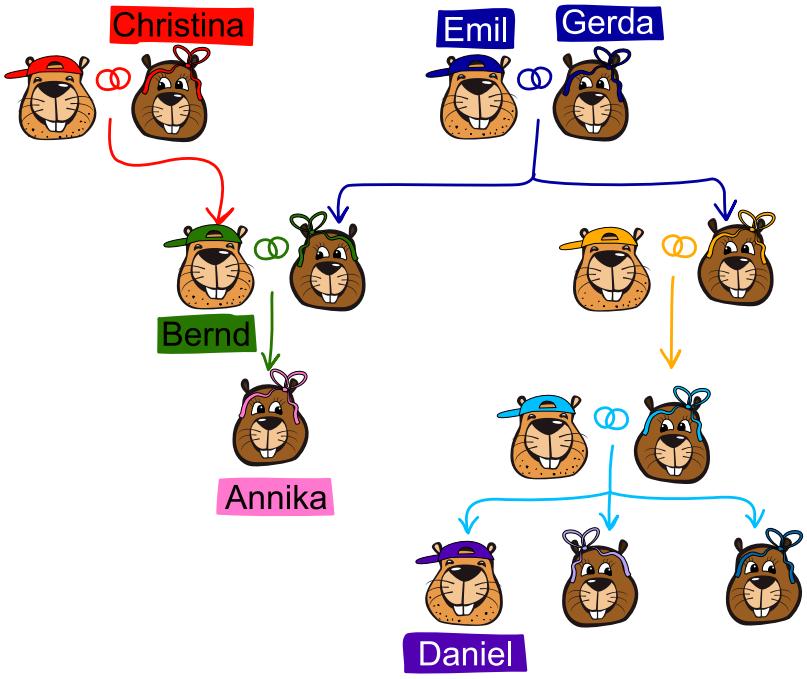
Le aree conesse sono interessanti anche per l'informatica, e in ambiti diversi. Un'area monocromatica in un'immagine (informatica) è un'area connessa di pixel dello stesso colore; può essere determinata utilizzando un algoritmo di riempimento, che funziona in modo simile alla propagazione della nebbia 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`

10. 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:

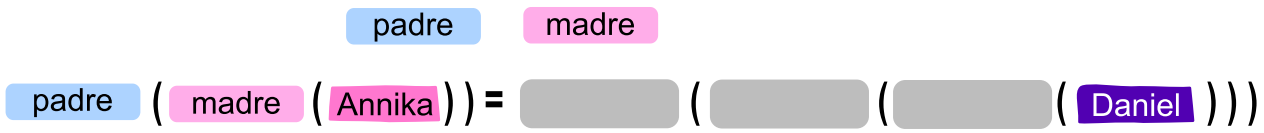
- `Padre(Annika) = Bernd`
- `Madre(Bernd) = Christina`

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».

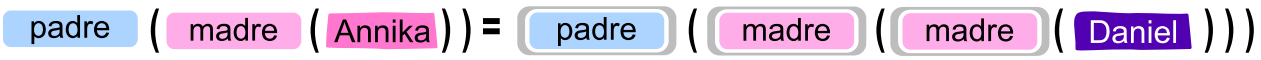
Ora vorrebbe avere un'equazione per la sua relazione con Daniel.

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`

11. 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.

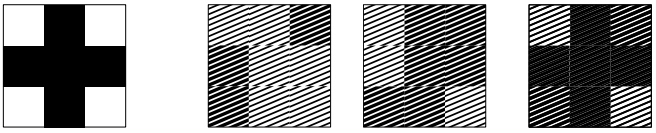


Immagine segreta | Foglio 1 | Foglio 2 | Fogli 1 e 2 sovrapposti

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.

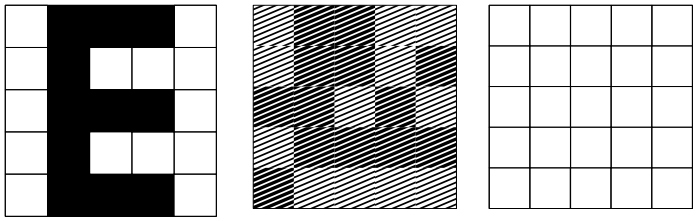
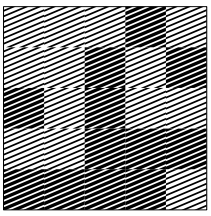


Immagine segreta | *Foglio 1* | *Foglio 2*

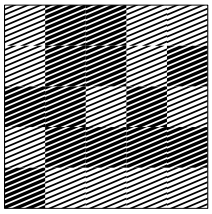
Soluzione

Questa è la soluzione.

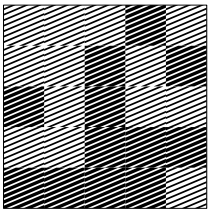


In questa immagine per il foglio 2, ogni pixel è stato impostato secondo la regola descritta sopra - in base all'immagine segreta e al foglio 1. L'immagine differisce da quella del foglio 1 solo nei punti esatti in cui l'immagine segreta presenta pixel neri.

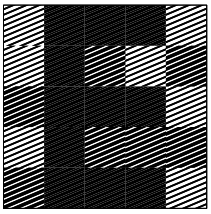
Qui si può vedere come l'immagine segreta possa essere riconosciuta sovrapponendo i fogli 1 e 2:



Foglio 1



Foglio 2



Fogli 1 e 2 sovrapposti

Questa è l'informatica!

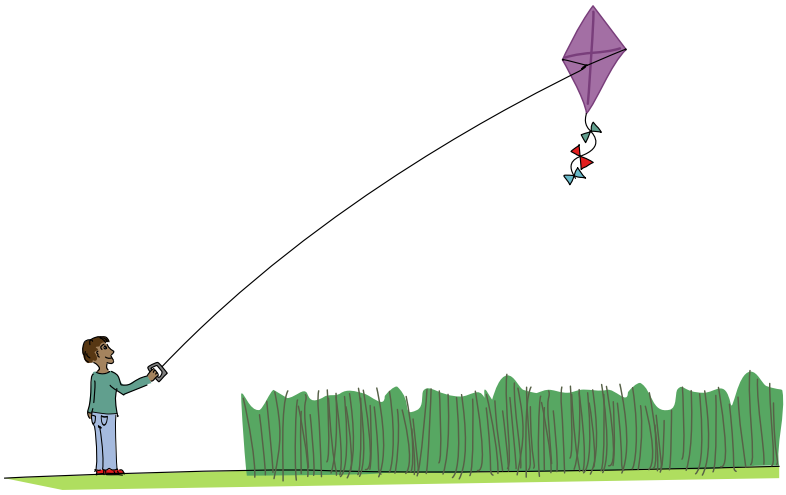
Il servizio di corriere utilizza un *processo crittografico* che si basa sulla percezione visiva ed è quindi chiamato *crittografia visuale*. Tale tecnica è stata sviluppata nel 1994 dagli scienziati israeliani Moni Naor e Adi Shamir. Il metodo risulta essere molto sicuro per quanto riguarda i singoli 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`

12. L'aquilone perduto

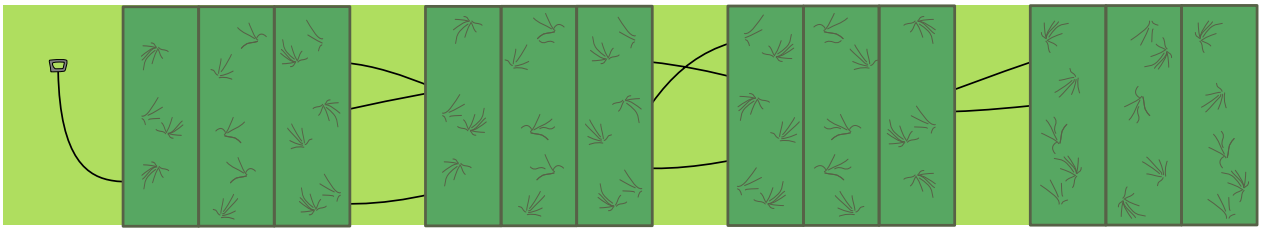


Che sfortuna! Asterios ha perso il suo aquilone nel prato. La corda dell'aquilone si è impigliata nell'erba alta e Asterios non riesce a ritrovarlo.

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.

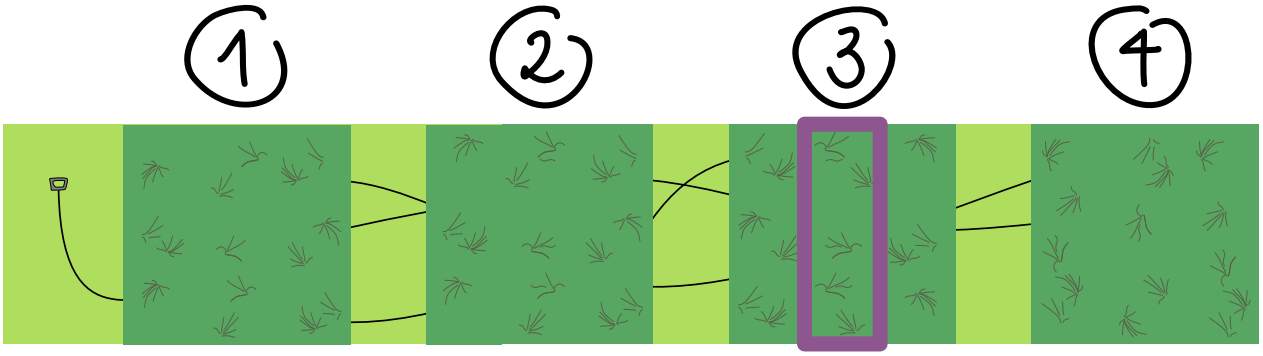
Di quale area si tratta?



Soluzione

Questa è la risposta:

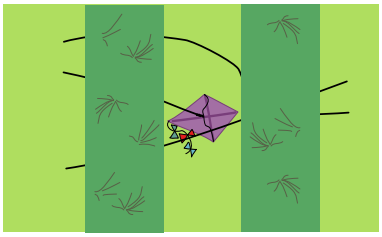
Asterios deve cercare nel campo evidenziato in viola per scoprire con certezza dove si trova l'aquilone.



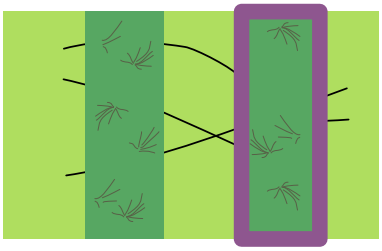
La soluzione si può trovare in due fasi. Per prima cosa pensate a quale dei quattro blocchi 1, 2, 3 e 4 deve trovarsi l'aquilone. Ricordate che la corda dell'aquilone inizia a sinistra e l'aquilone è appeso all'altra estremità della corda.

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.

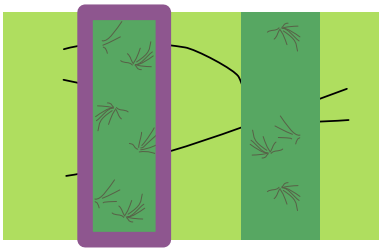
Quando Asterios cerca il campo centrale nel blocco 3, si possono distinguere tre casi:



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

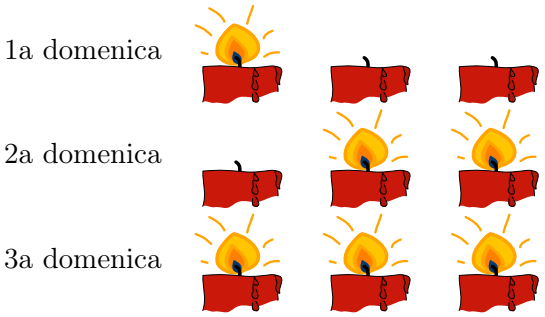
- Topologia: `https://it.wikipedia.org/wiki/Topologia`
- Algoritmo di ricerca: `https://it.wikipedia.org/wiki/Algoritmo_di_ricerca`

13. 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.

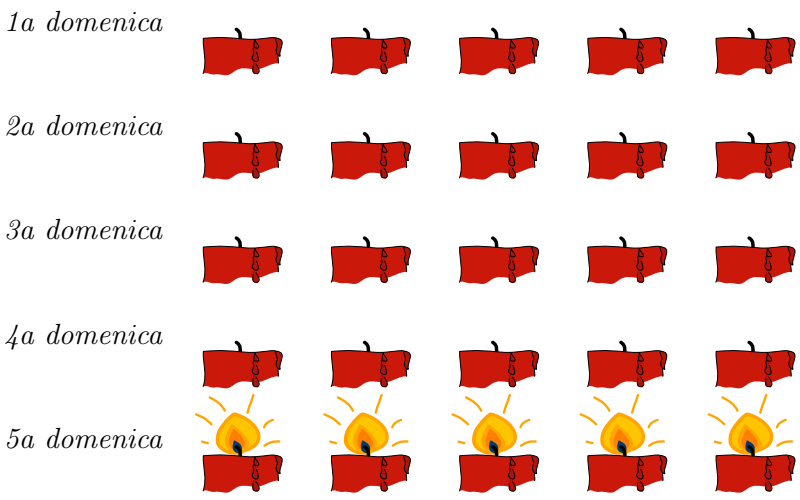
Se la tradizione prevedesse solo tre domeniche (e candele) ciò sarebbe invece possibile, perché Chris accenderebbe ogni candela esattamente due volte:



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

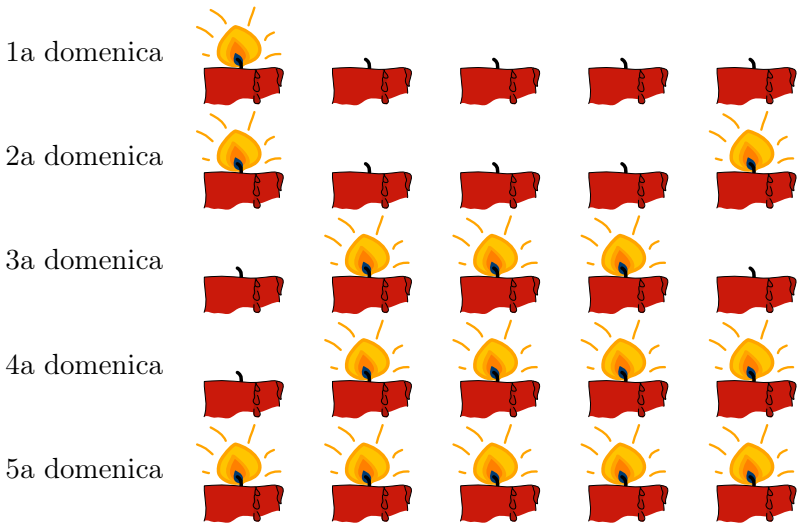
Mostra a Chris come accendere ogni candela lo stesso numero di volte.

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.

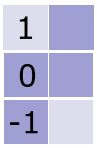
Lo sviluppo di algoritmi per risolvere i problemi è uno dei compiti più importanti degli informatici. Se un algoritmo si basa su una solida modellazione matematica, è più facile dimostrarne le proprietà desiderate che non senza tale modellazione. Per l'algoritmo di cui sopra, è possibile dimostrare che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

Parole chiave e siti web

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

14. 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:

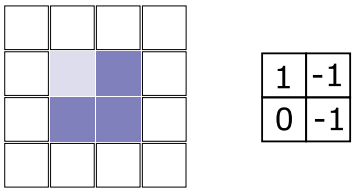


1, se il pixel è più chiaro del pixel adiacente alla sua destra.

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

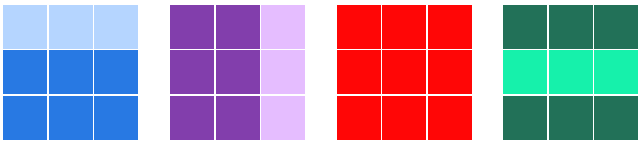
-1, se il pixel è più scuro del pixel adiacente alla sua destra.

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.

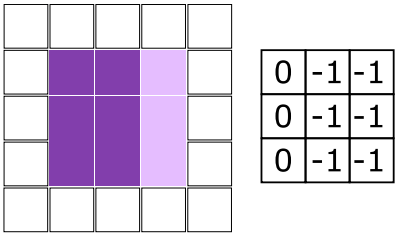
Quale delle immagini è l'unica con una mappa di luminosità ***diversa****?*



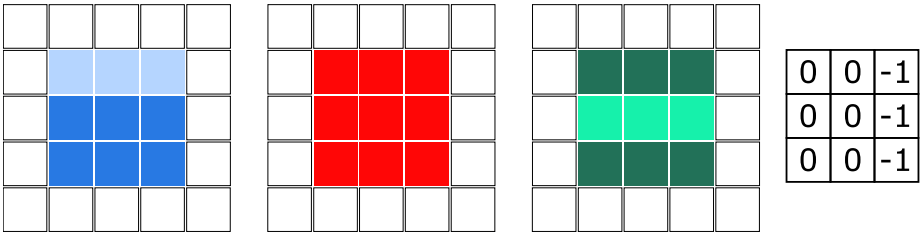
A) B) C) D)

Soluzione

La risposta è l'immagine B:



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



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.

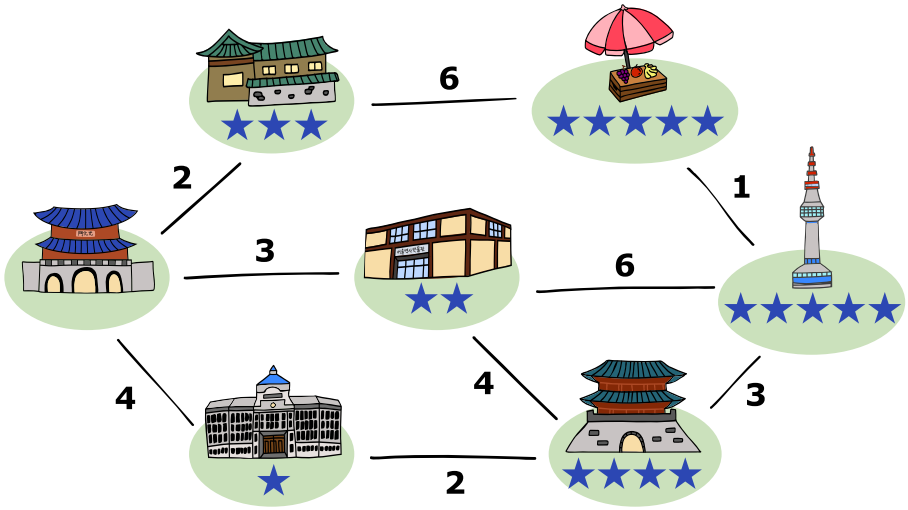
Le cosiddette *Reti neurali convoluzionali* (CNN), spesso componenti centrali dei sistemi di intelligenza artificiale, utilizzano il concetto di convoluzione per il riconoscimento delle immagini. Per riconoscere oggetti complessi in un'immagine, una CNN impara a sviluppare da sola i filtri adatti, che 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`

15. 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.



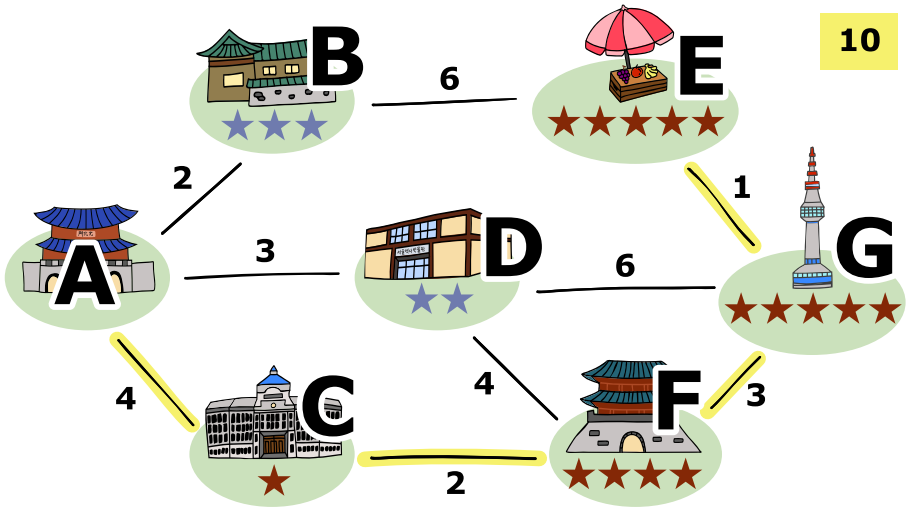
Lotte visita prima il palazzo  col tetto blu a sinistra della mappa. Da lì, vorrebbe visitare altri luoghi in autobus. Lotte ha un biglietto con cui può viaggiare per un massimo di 10 chilometri. Vuole utilizzare le coincidenze per raggiungere luoghi con il maggior numero possibile di stelle! Può visitare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

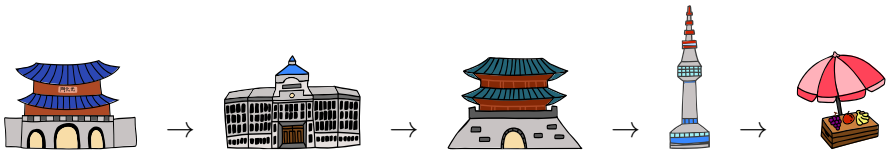
Un percorso viene definito *valido* se non supera i 10 chilometri di lunghezza. Un percorso valido è, ad esempio

- 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`

16. Laghi di montagna

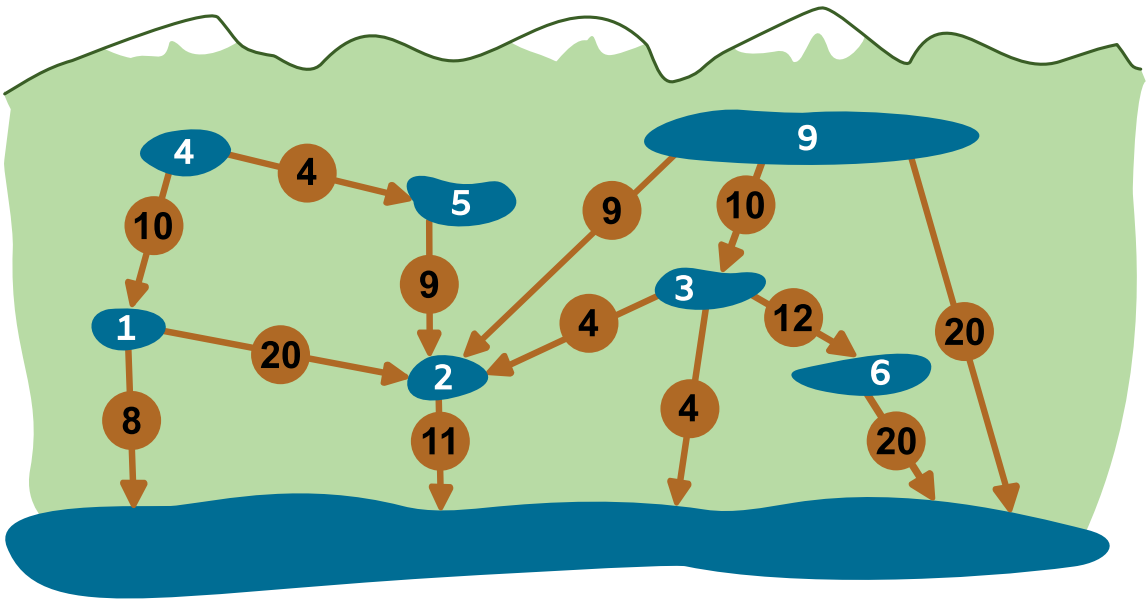
Sul massiccio montuoso sopra un bacino artificiale ci sono diversi piccoli laghi di montagna. In caso di forti piogge potrebbero straripare, il che è pericoloso. Per questo motivo, è prevista la costruzione di canali tra alcuni dei laghi. Questi canali dovrebbero essere in grado di drenare tutta l'acqua 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.

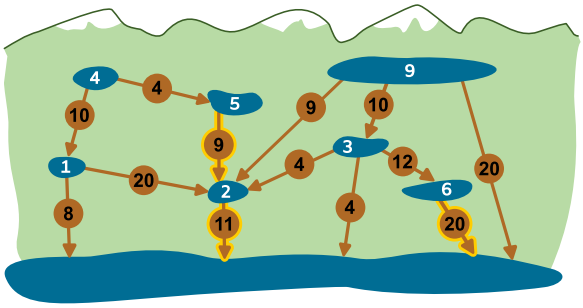
Nota: se un canale drena l'acqua di un piccolo lago di montagna in un secondo lago, l'acqua in eccesso di entrambi i laghi si raccoglie nel secondo lago.

Dove dovrebbero essere costruiti i canali per minimizzarne il costo?

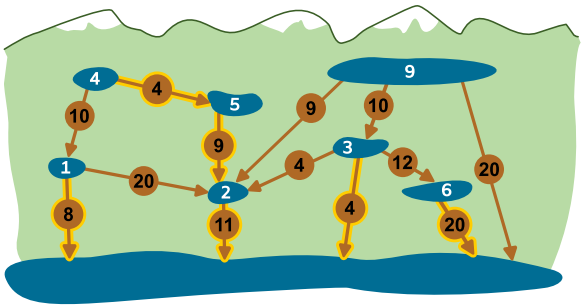


Soluzione

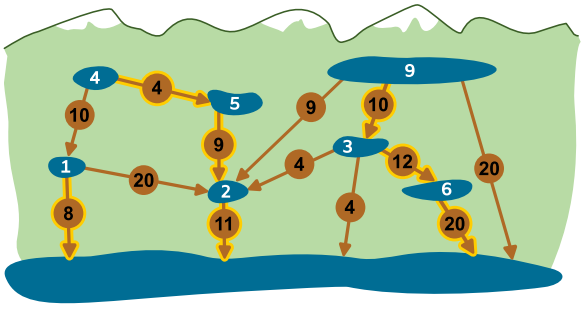
La risposta corretta è la seguente:



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`

17. 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:

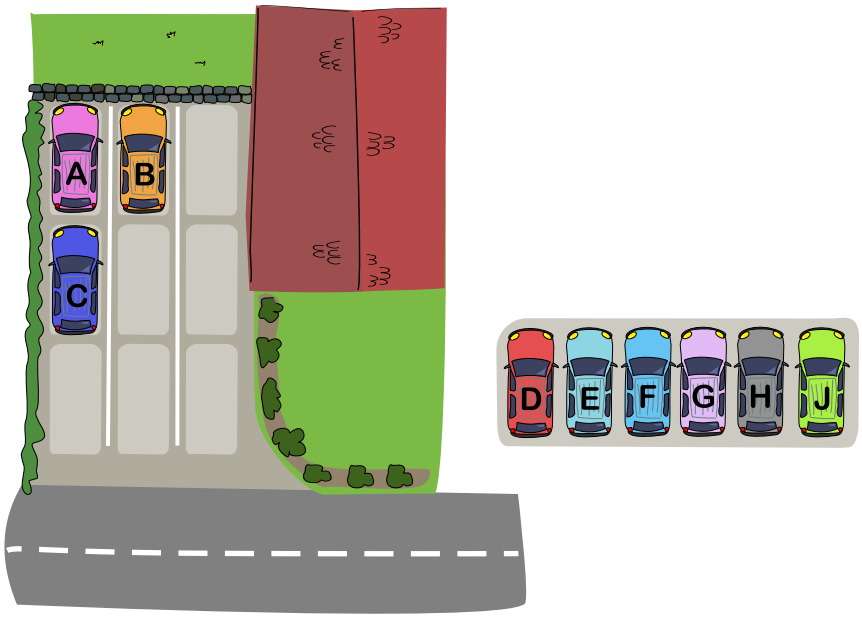
Anja, **B**eate, **C**lara, **D**avid, **E**lia, **F**rank, **G**abi, **H**arald e infine **J**ulia.

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.

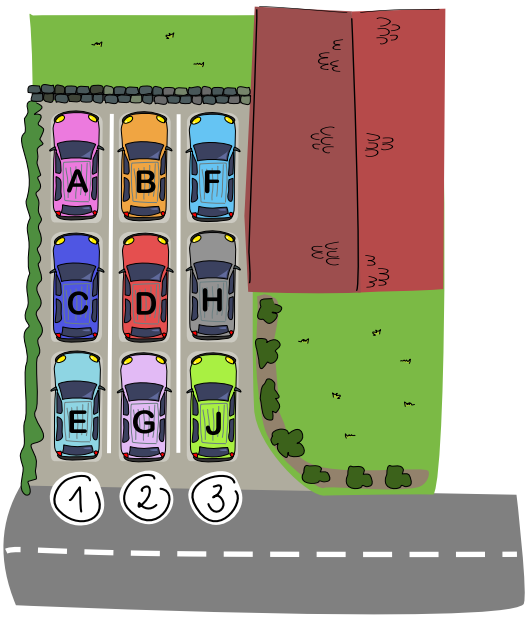


Mostra agli ospiti come parcheggiare!

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`

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.

18. 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:

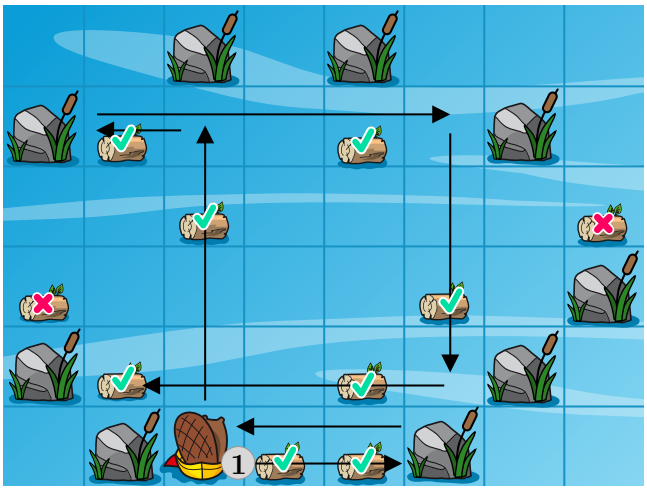
Istruzione	**Descrizione**
`turnRight()` / `turnLeft()`	Linus ruota sul posto di 90 gradi verso destra / sinistra.
`paddle()`	Linus rema in avanti finché non si trova davanti a un masso. Se rema sopra un tronco d'albero, lo raccoglie.

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`

A. Autori dei quesiti

James Atlas

Masiar Babazadeh

Wilfried Baumann

Leonardo Cavalcante

Špela Cerar

Andrew Csizmadia

Christian Datzko

Diane Dowling

Nora A. Escherle

Gerald Futschek

Christian Giang

Silvan Horvath

Alisher Ikramov

Thomas Ioannou

Asterios Karagiannis

Blaž Kelvišar

David Khachatryan

Doyong Kim

Jihye Kim

Dong Yoon Kim

Vaidotas Kinčius

Stefan Koch

Lukas Lehner

Gunwoong Lim

Linda Mannila

Anna Morpurgo

A-Yeong Park

Suchan Park

Gabriela Gomez Pasquali

Jean-Philippe Pellet

Zsuzsa Pluhár

Wolfgang Pohl

Pedro Ribeiro

Kirsten Schlüter

Margareta Schlüter

Dirk Schmerenbeck

Jacqueline Staub

Susanne Thut

Christine Vender

Florentina Voboril

Michael Weigend

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/