INFORMATIK-BIBER SCHWEIZ
CASTOR INFORMATIQUE SUISSE
CASTORO INFORMATICO SVIZZERA

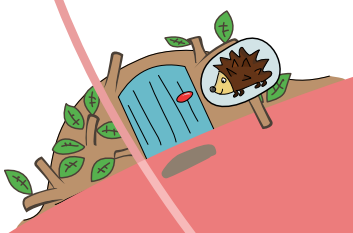
Quesiti e soluzioni 2025

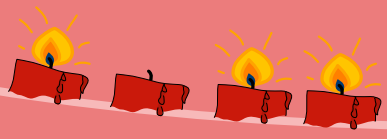
5º e 6º anno scolastico

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

A cura di:

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





0101001101010110010010010100000100101101010100110101001101001010010100010100101010100110101001100100101001001010010000101

SSII

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

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

Hanno collaborato al Castoro Informatico 2025

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

Capo progetto: Nora A. Escherle

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

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

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

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

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

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

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

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

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

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

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

INFORMATIK-BIBER SCHWEIZ
CASTOR INFORMATIQUE SUISSE
CASTORO INFORMATICO SVIZZERA

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

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

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

Premessa

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Per ulteriori informazioni:

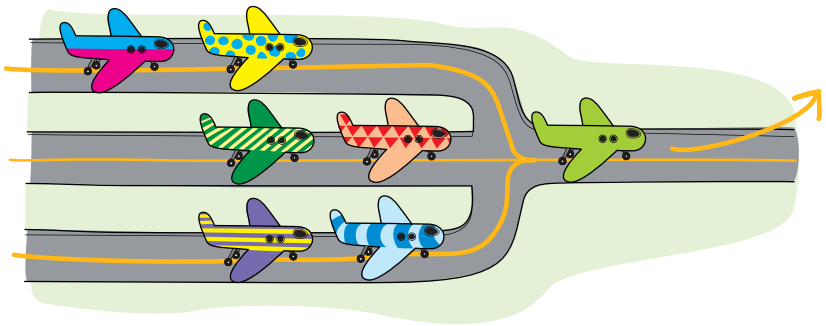
Società Svizzera per l'Informatica nell'Insegnamento
SVIA-SSIE-SSII
Castoro Informatico
Masiar Babazadeh

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

Indice

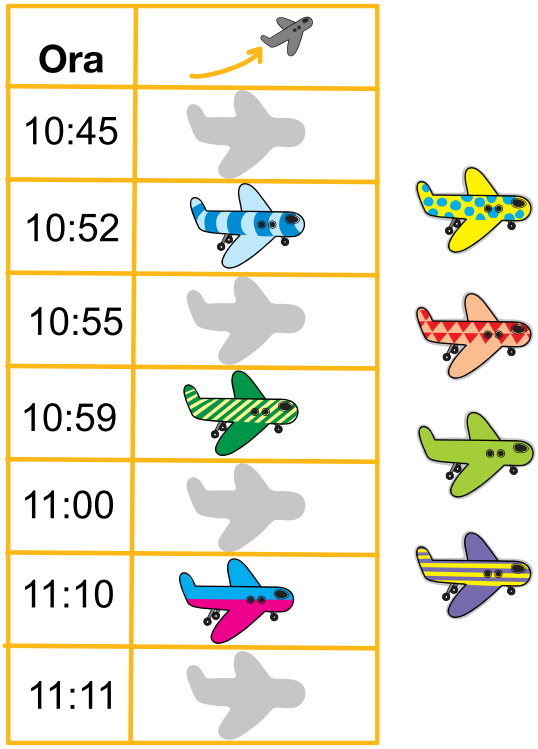
1. Aerei

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



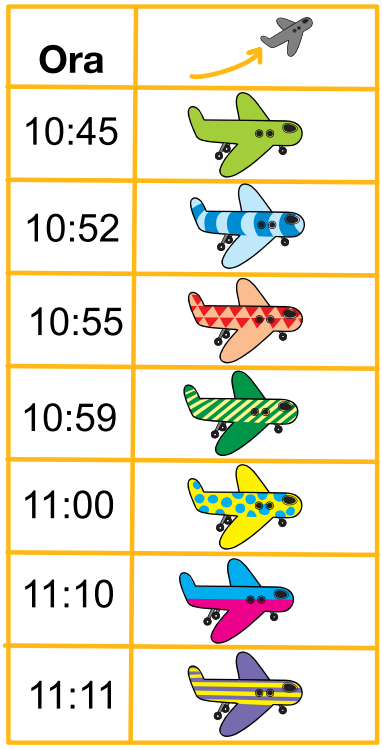
L'orario indica l'ordine di decollo dei sette aerei. Purtroppo alcuni aerei non sono ancora segnati. C'è solo un ordine possibile di partenza.

Aggiungi gli aerei mancanti all'orario.



Soluzione

Questa è la soluzione:



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

Questa è l'informatica!

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

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

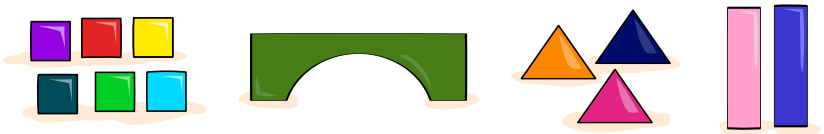
In informatica, la programmazione è considerata un problema particolarmente difficile. La sfida consiste nel calcolare un piano ottimale per ogni situazione immaginabile. Se si devono prendere in considerazione molti parametri e condizioni, anche un computer molto veloce può impiegare una quantità enorme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

Parole chiave e siti web

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

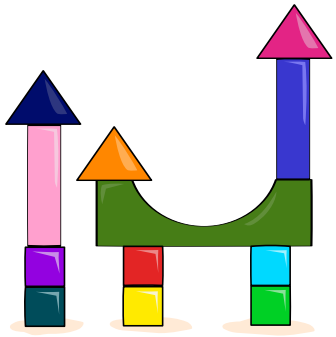
2. I mattoncini

Ali ha questi mattoncini:



6 cubi | 1 ponte | 3 piramidi | 2 cuboidi

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



In che ordine Zaha ha dato le istruzioni?

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

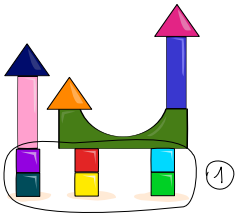
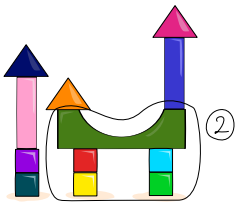
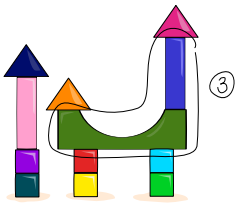
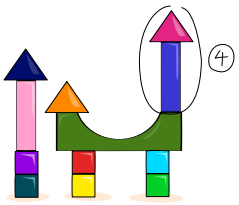
- Posizionare entrambi i cuboidi sulla strutura.
- Posizionare le torri appena costruite in fila.
- Posizionare le piramidi sulla strutura.
- Costruire tre torri con 2 cubi ciascuna.
- Posizionare il ponte sulla struttura.

Soluzione

Zaha ha dato le istruzioni in questo ordine:

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

Spieghiamo ora perché è proprio questa la sequenza.

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

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

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

Questa è l'informatica!

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

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

Parole chiave e siti web

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

3. Tornare a casa

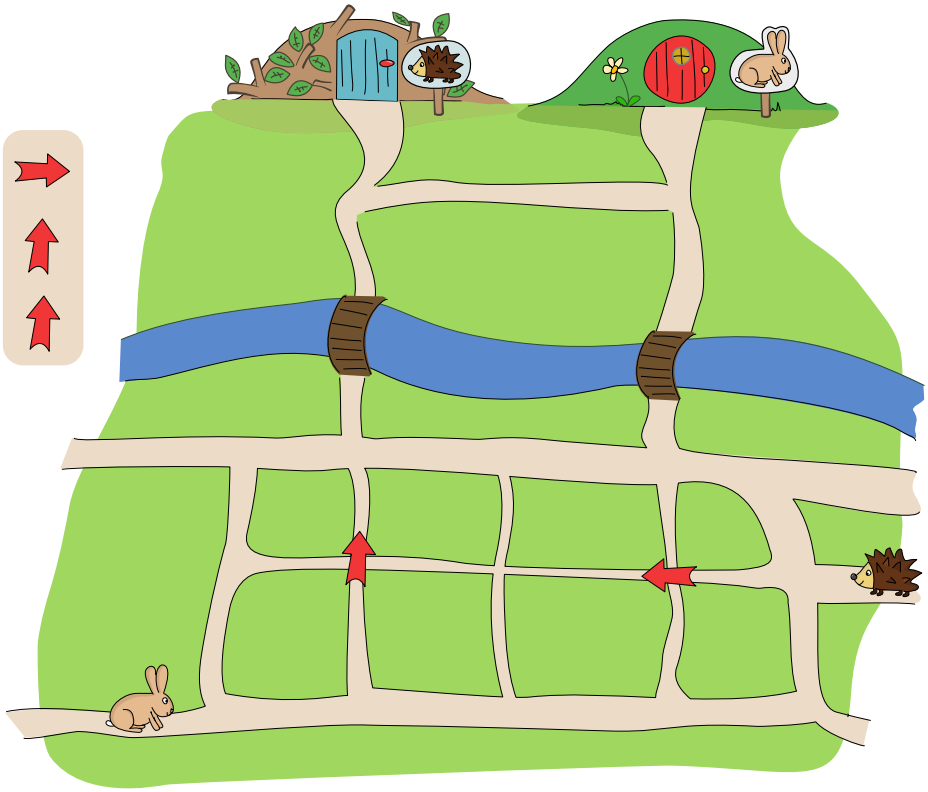
Un coniglio  e un riccio  vogliono tornare entrambi a casa propria:

 e  .

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

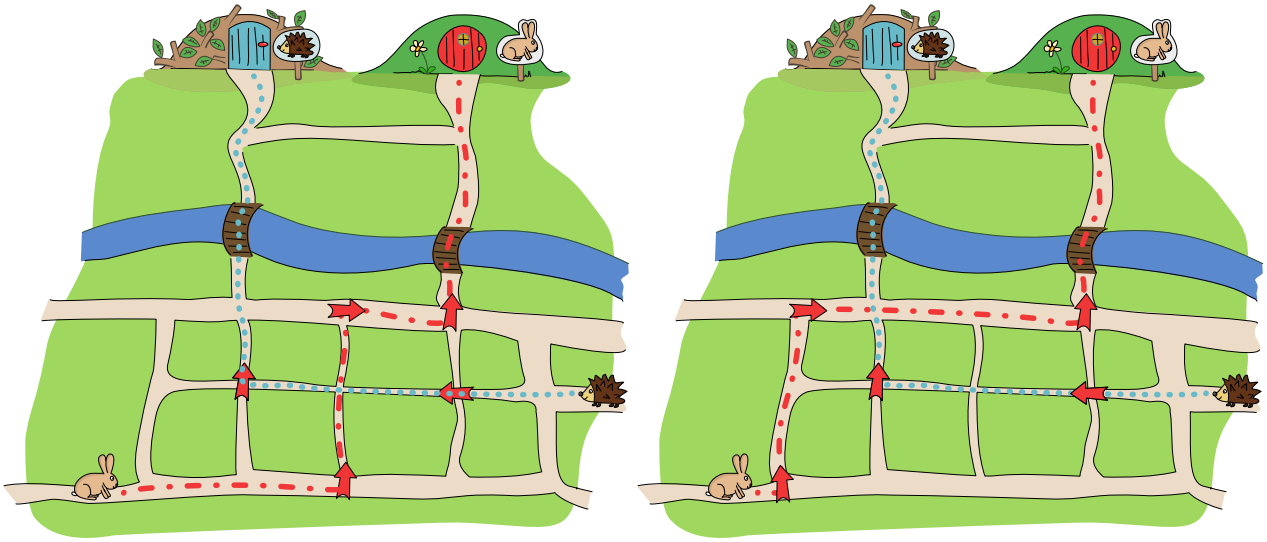
In alcuni incroci sono già presenti delle frecce. Il riccio trova la strada di casa seguendo le frecce, ma il coniglio no. Fortunatamente sono rimaste tre frecce.

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



Soluzione

Ci sono due risposte corrette:



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

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

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

Questa è l'informatica!

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

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

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

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

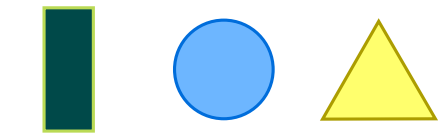
Questi esempi dimostrano che è fondamentale che la qualità dell'output di un algoritmo dipenda dalla qualità dell'input. È il caso, ad esempio, di molti sistemi di intelligenza artificiale. I sistemi di IA lavorano con modelli della realtà e questi modelli sono formati da dati. Se questi dati non sono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

Parole chiave e siti web

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

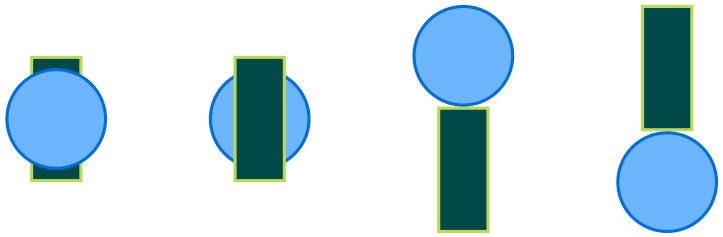
4. Hivobu

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



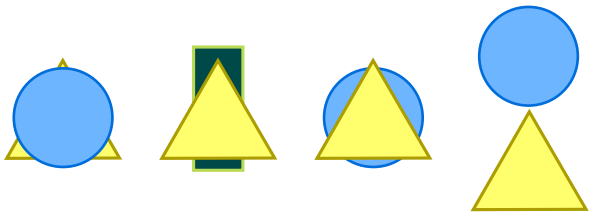
RAH OH TEH

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



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

Che cosa significa TEH OH CO in Hivobu?



A) B) C) D)

Soluzione

La risposta C è corretta.

Sappiamo come si chiamano le singole forme:

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

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

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

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

Ma quali sono le altre risposte di Hivobu?

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

Questa è l'informatica!

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

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

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

Parole chiave e siti web

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

5. Legno per la diga

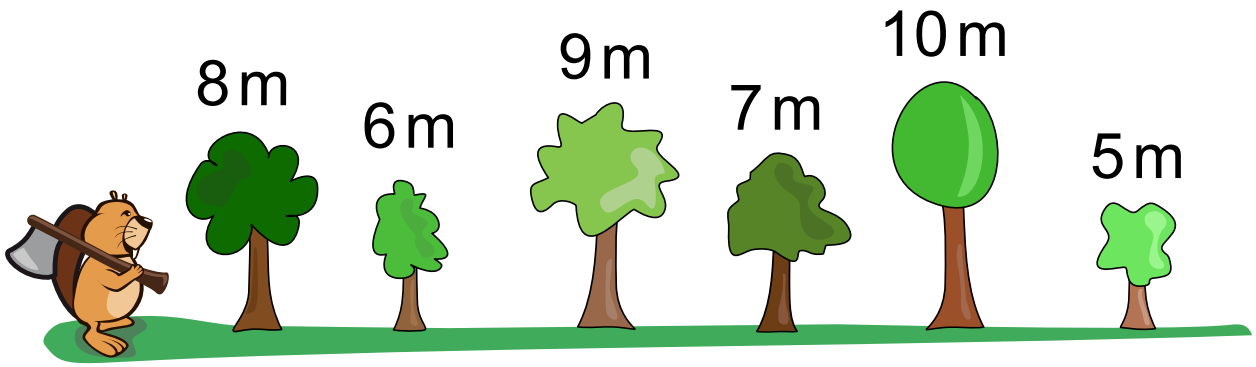
Alcuni castori devono abbattere degli alberi per costruire la loro prossima diga.

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

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

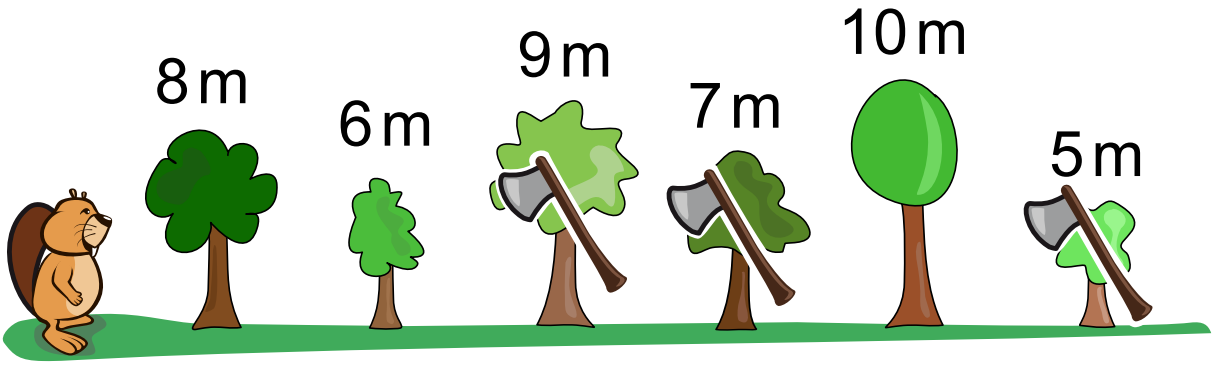
Ad esempio, se tagliano l'albero di 6 metri, possono tagliare solo quello di 5 metri. Alla fine si ritroverebbero con un totale di 11 metri di legno.

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



Soluzione

La risposta corretta è la seguente:



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

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

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

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

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

Questa è l'informatica!

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

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

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

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

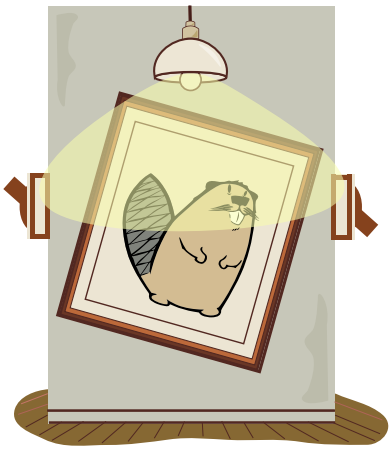
Per coloro che sono particolarmente interessati, il problema di questo compito è noto in informatica anche come «Somma massima decrescente successiva». Su `questo sito web` è possibile vedere come approcci semplici (ma non ottimizzati) per arrivare alla soluzione di un problema di ottimizzazione possano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

Parole chiave e siti web

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

6. Lampada pazzesca

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



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

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

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

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

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

Soluzione

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

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

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

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

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

Questa è l'informatica!

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

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

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

Parole chiave e siti web

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

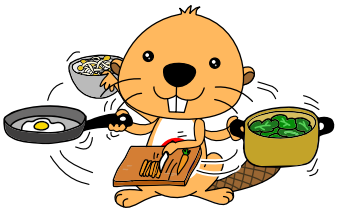
- Algebra di Bool: `https://it.wikipedia.org/wiki/Algebra_di_Boole`

7. Bibimbap

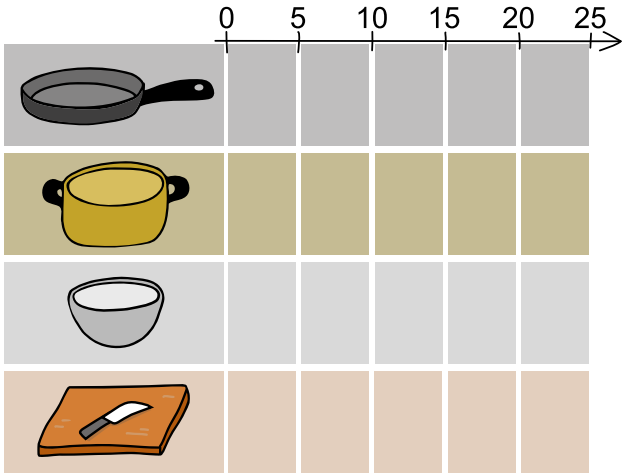
Un cuoco vuole preparare un piatto tradizionale coreano, il bibimbap (비빔밥). Utilizza quattro utensili: una pentola, una padella, un tagliere e una ciotola. I quattro ingredienti per il bibimbap vanno preparati in questo modo:

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

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

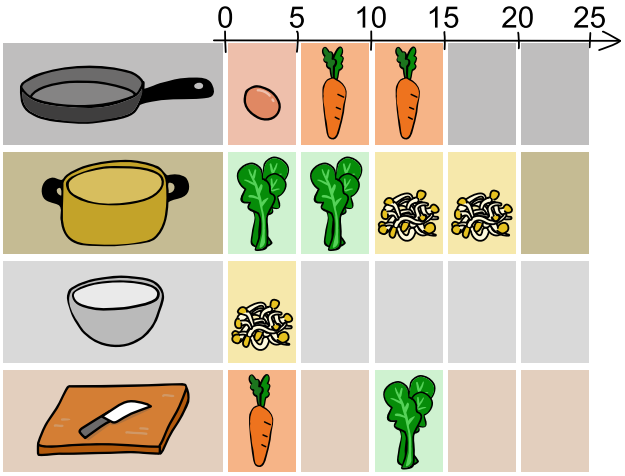


Definisci un piano che consenta allo chef di preparare gli ingredienti per il bibimbap nel minor tempo possibile.



Soluzione

La risposta corretta è la seguente:



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

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

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

Questa è l'informatica!

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

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

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

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

Parole chiave e siti web

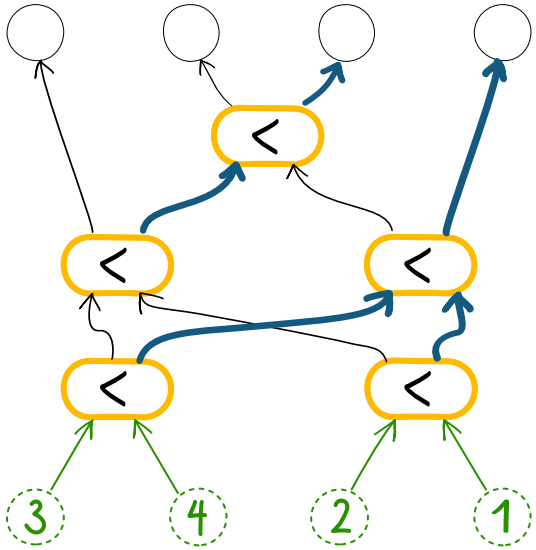
- Scheduling: `https://www.swisseduc.ch/informatik/theoretische_informatik/scheduling/algorithmus1.html`
- Metodo del percorso critico: `https://it.wikipedia.org/wiki/Metodo_del_percorso_critico`
- Relazione tecnica di Stanford John Fondahl (vedi prefazione): `https://catalog.hathitrust.org/Record/005766951`

8. 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.

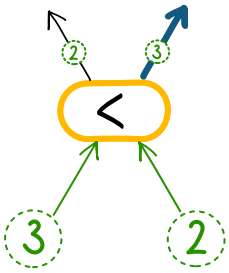
I numeri si muovono verso l'alto nella macchina lungo frecce e snodi < fino ai campi di uscita ◯.



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

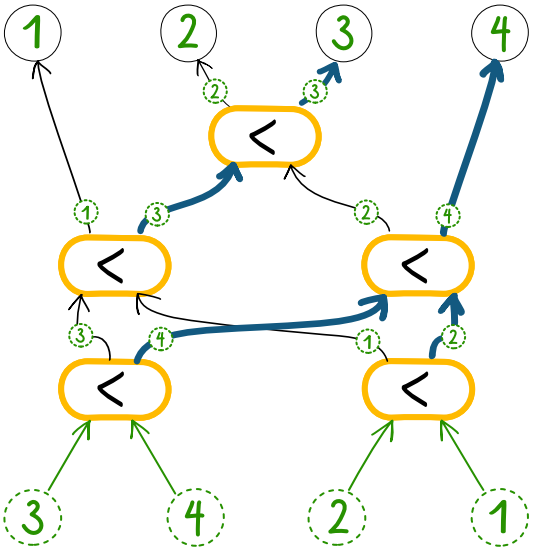
C) Visualizza i numeri nello stesso ordine. Risultato: 3, 4, 2, 1

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

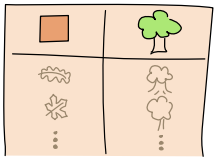
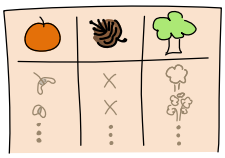
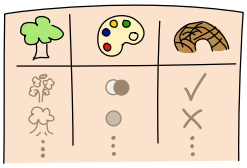
di passi non paralleli in una rete di ordinamento è inferiore al numero di elementi da ordinare. Uno svantaggio delle reti di ordinamento è che sono progettate solo per ingressi di lunghezza fissa.

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`

9. 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.

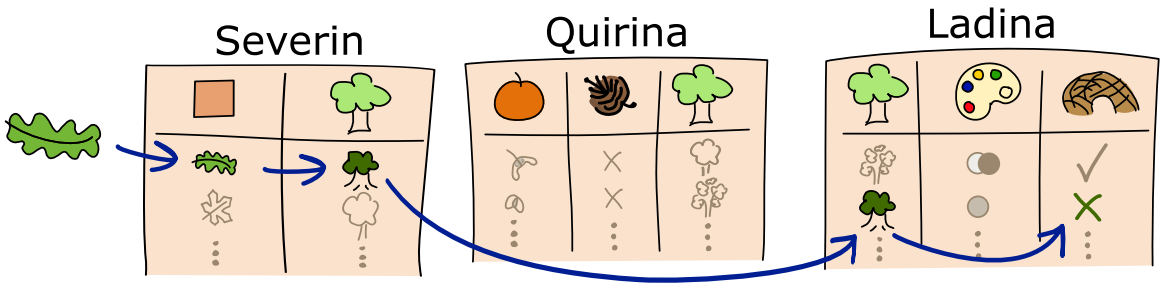
Soluzione

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.

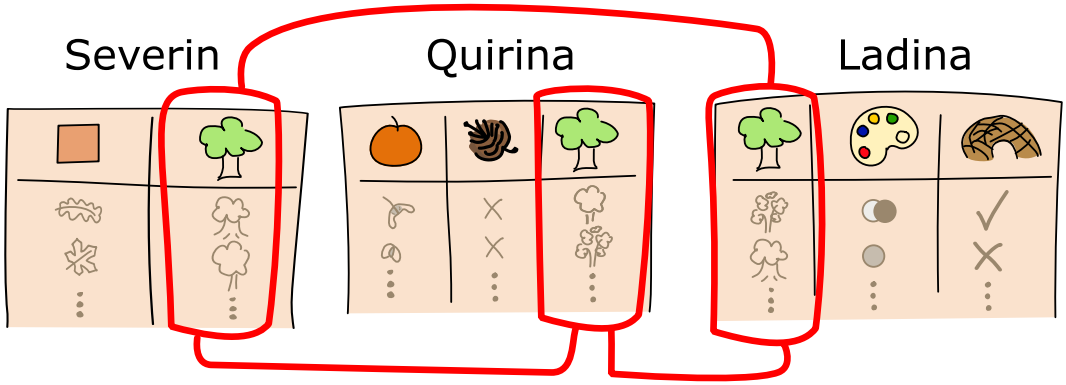
La tabella di Severin contiene informazioni sulle foglie. Siccome Emil conosce la forma della foglia, può prima selezionare la riga appropriata nella tabella di Severin e ottenere le informazioni mancanti sulla specie di albero. Può quindi utilizzare questa informazione per selezionare la riga appropriata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



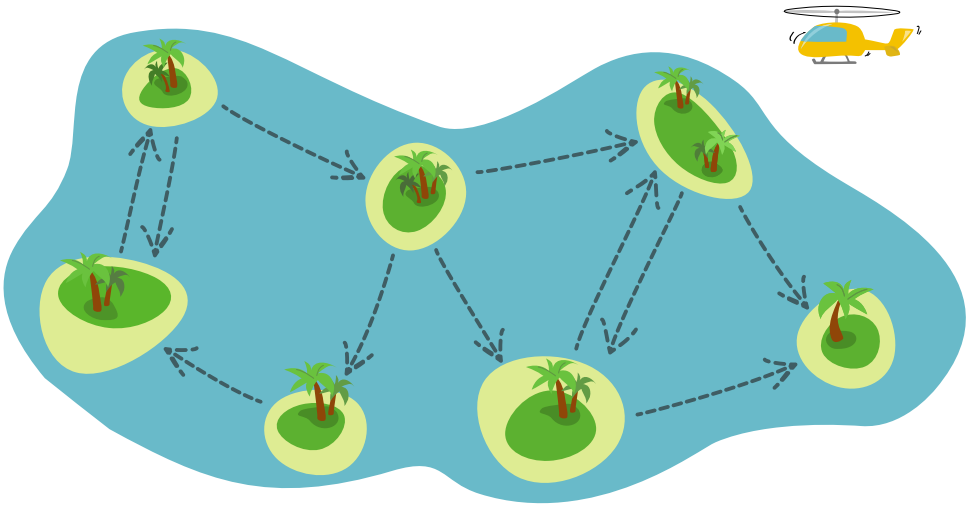
I database relazionali sono così importanti che esiste un linguaggio separato in informatica per interrogare ed eseguire altre operazioni sui database, chiamato *SQL (Structured Query Language)*.

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`

10. 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

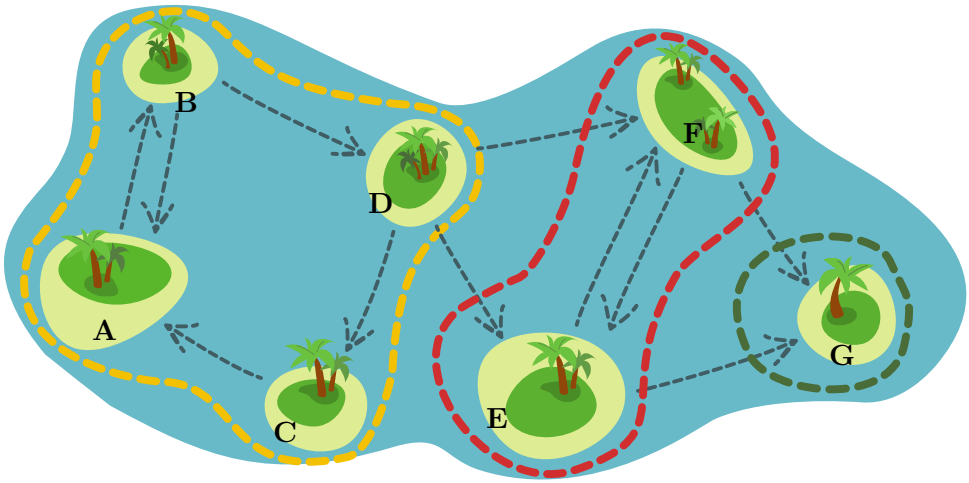
Soluzione

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

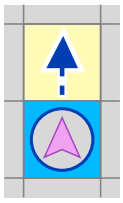
- Grafo diretto: `https://it.wikipedia.org/wiki/Digrafo_(matematica)`
- Grafo connesso: `https://it.wikipedia.org/wiki/Grafo_connesso`
- Componente fortemente connessa: `https://it.wikipedia.org/wiki/Componente_fortemente_connessa`

11. Lefty II

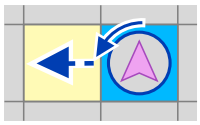
Il robot *Lefty* si muove su una griglia composta da caselle quadrate. Tra le caselle possono esserci dei muri rossi. Lefty deve raggiungere l'obiettivo verde.

Lefty può muoversi solo in due modi:

Avanzare di una casella

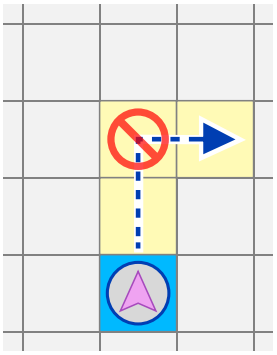


Girare a sinistra e avanzare immediatamente di una casella

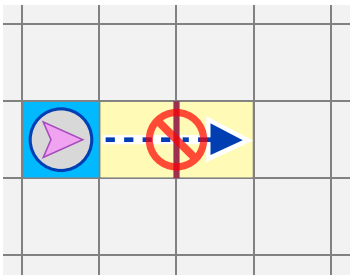


Quindi ci sono azioni che Lefty non può fare:

... **non** può girare a destra e...

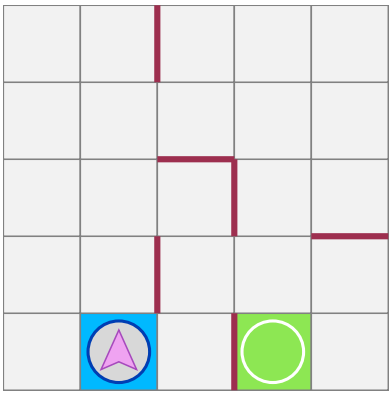


non può passare attraverso i muri.



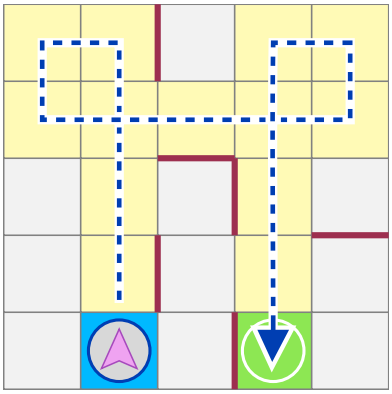
Quali caselle deve attraversare Lefty per raggiungere la destinazione?

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



Soluzione

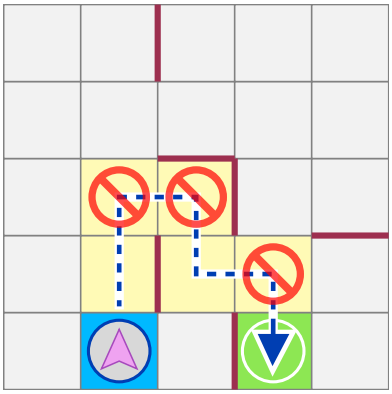
Questa è la risposta:



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.

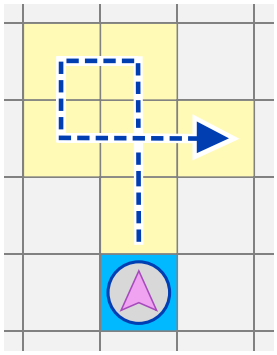


Questa è l'informatica!

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`

12. Labirinto

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

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

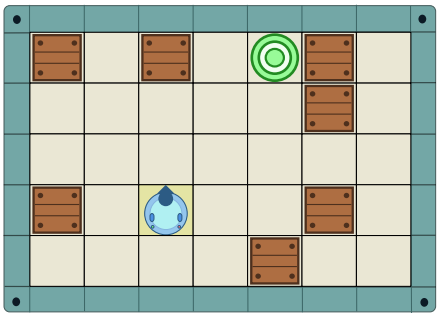
Avanza di una casella.

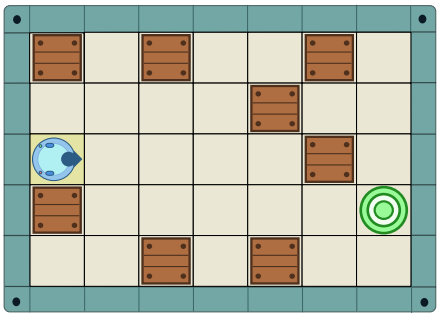
Avanza finché non puoi più proseguire.

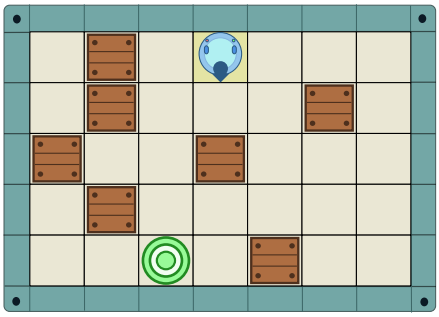
Gira di 90 gradi in senso orario.

Gira di 90 gradi in senso antiorario.

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







Crea questo programma per Momo!

Avanza di una casella.

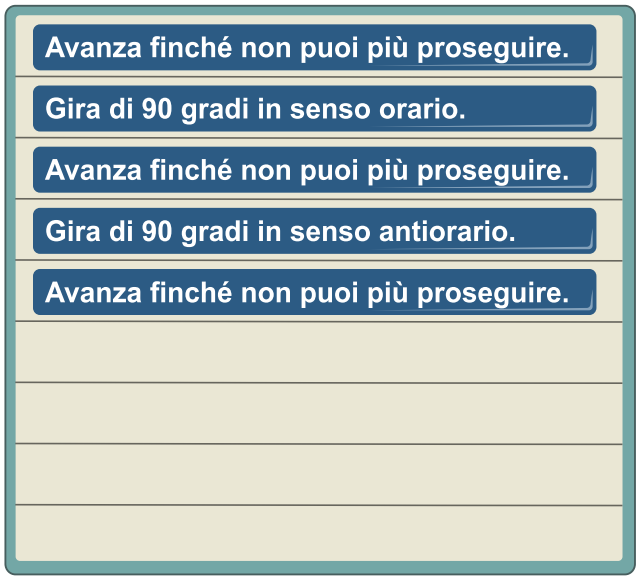
Avanza finché non puoi più proseguire.

Gira di 90 gradi in senso orario.

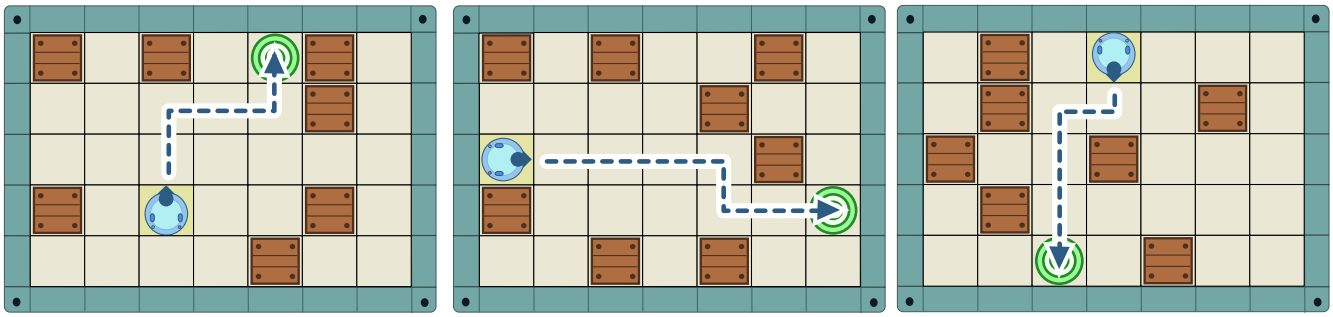
Gira di 90 gradi in senso antiorario.

Soluzione

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



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



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

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

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

Questa è l'informatica!

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

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

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

Parole chiave e siti web

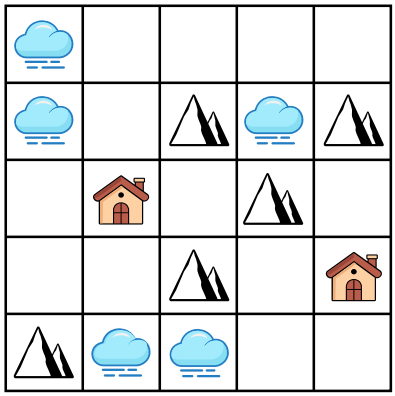
- Programmazione strutturata:
  `https://de.wikipedia.org/wiki/Strukturierte_Programmierung`
- Algoritmi per la risoluzione di labirinti:
  `https://it.wikipedia.org/wiki/Algoritmi_per_la_risoluzione_di_labirinti`

13. Giornata nebbiosa

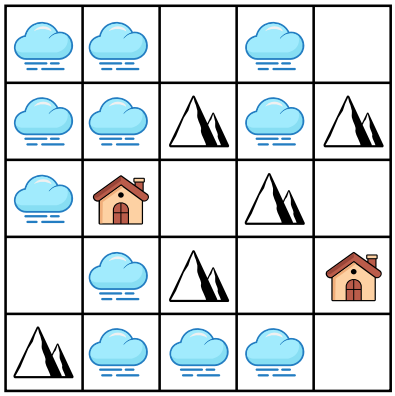
Oggi c'è nebbia nella terra dei monti e si diffonde ad ogni ora che passa.

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.

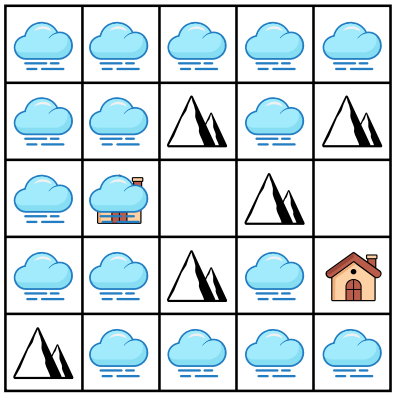
Ecco un esempio:



Alba

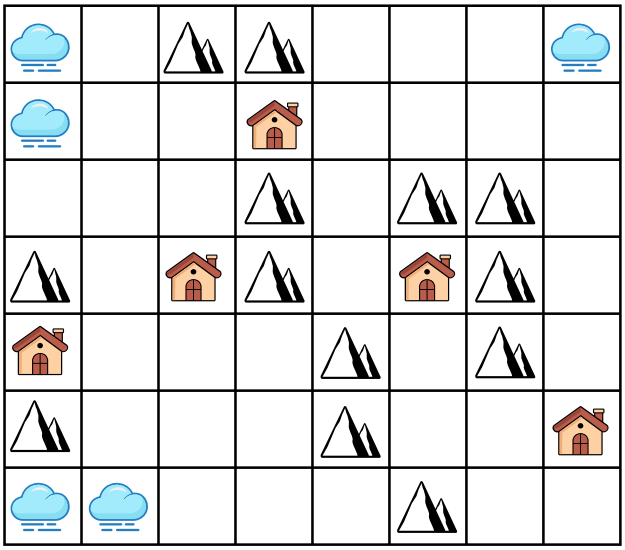


Dopo 1 ora



Dopo 2 ore

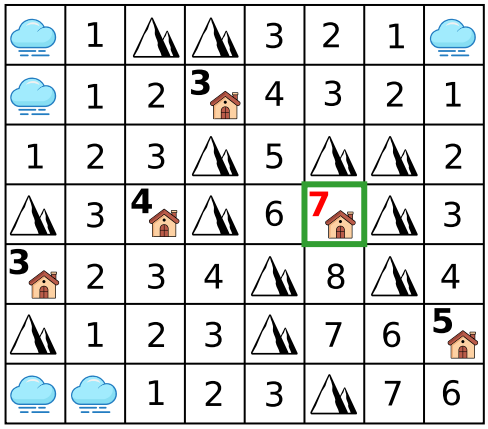
*Quale casa del paese è l'**ultima** ad essere coperta dalla nebbia?*



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.

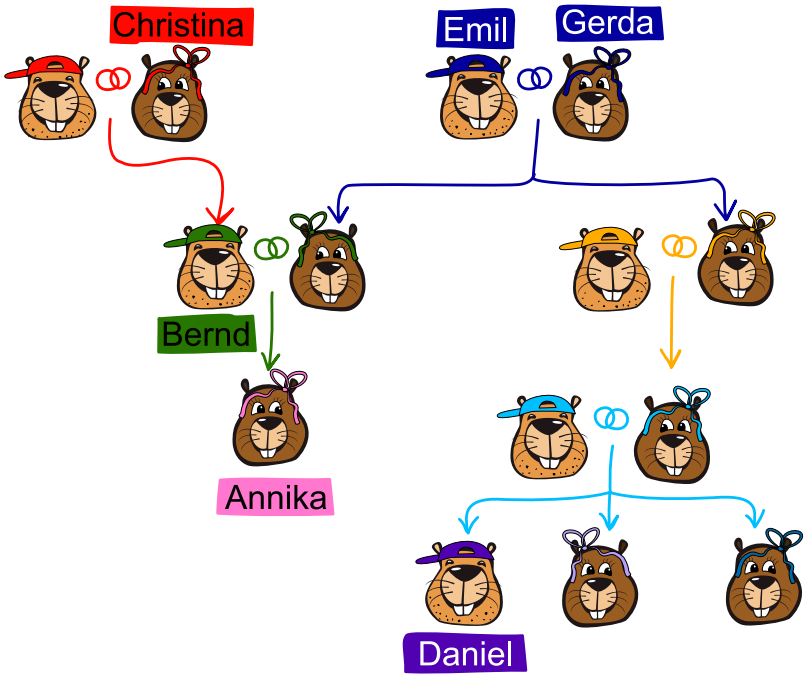
Le aree conesse sono interessanti anche per l'informatica, e in ambiti diversi. Un'area monocromatica in un'immagine (informatica) è un'area connessa di pixel dello stesso colore; può essere determinata utilizzando un algoritmo di riempimento, che funziona in modo simile alla propagazione della nebbia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

14. 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:

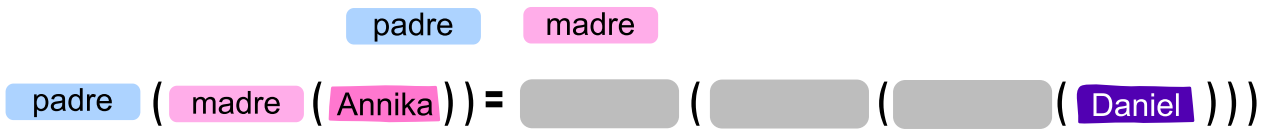
- `Padre(Annika) = Bernd`
- `Madre(Bernd) = Christina`

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».

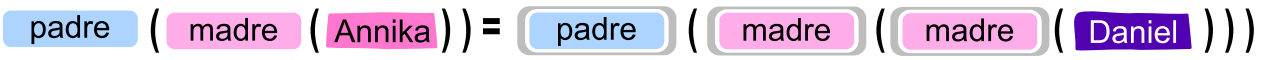
Ora vorrebbe avere un'equazione per la sua relazione con Daniel.

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`

15. 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.

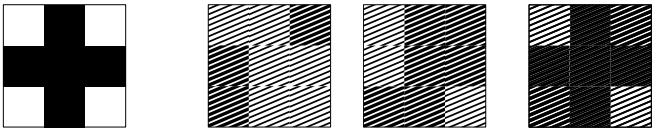


Immagine segreta | Foglio 1 | Foglio 2 | Fogli 1 e 2 sovrapposti

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.

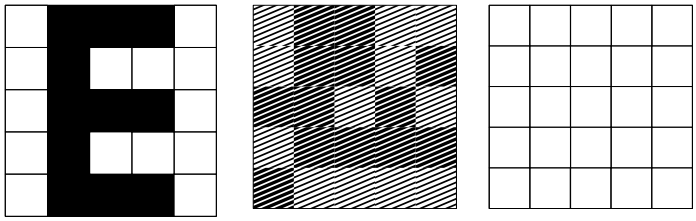
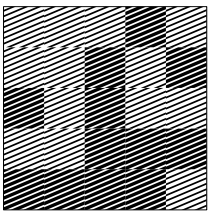


Immagine segreta | *Foglio 1* | *Foglio 2*

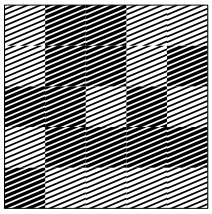
Soluzione

Questa è la soluzione.

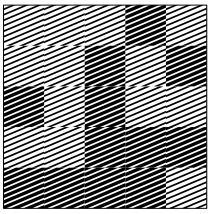


In questa immagine per il foglio 2, ogni pixel è stato impostato secondo la regola descritta sopra - in base all'immagine segreta e al foglio 1. L'immagine differisce da quella del foglio 1 solo nei punti esatti in cui l'immagine segreta presenta pixel neri.

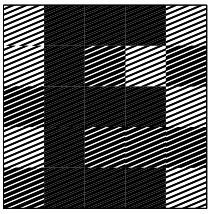
Qui si può vedere come l'immagine segreta possa essere riconosciuta sovrapponendo i fogli 1 e 2:



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

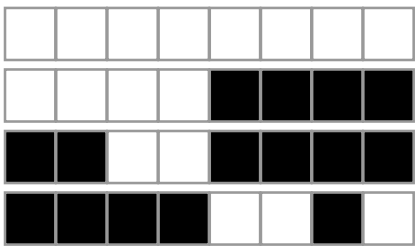
- Crittografia visuale: `https://it.wikipedia.org/wiki/Crittografia_visuale`

16. Nero e bianco

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

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

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



`B`

`xBN`

`xxNBN`

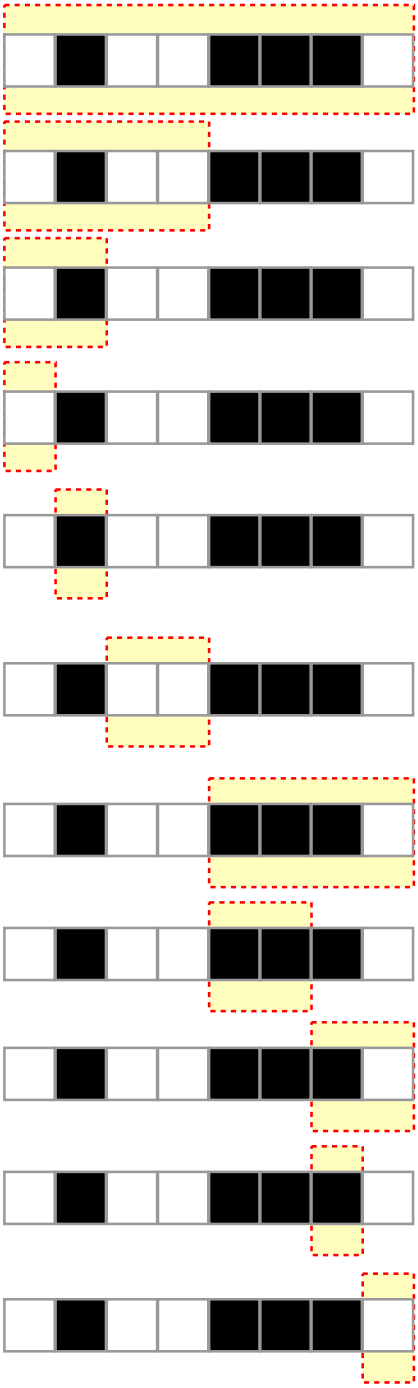
`xNxBxNB`

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

Soluzione

Questa è la rispota corretta: `xxxBNBxNxNB`.

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



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

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

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

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

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

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

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

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

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

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

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

Questa è l'informatica!

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

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

Parole chiave e siti web

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

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.

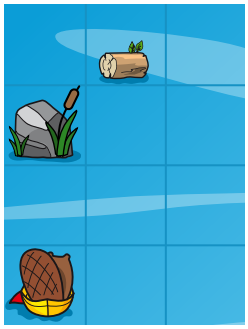
17. La secca pazza

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

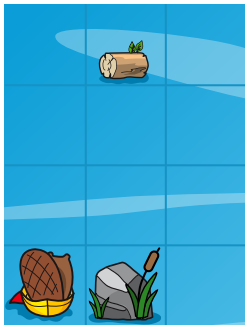
Puoi usare le seguenti istruzioni:

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

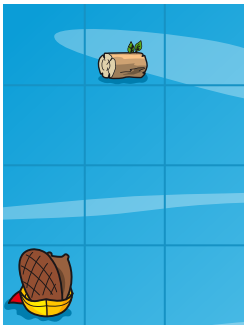
Lago 1



Lago 2



Lago 3



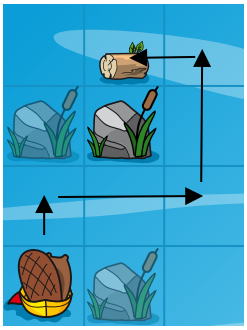
Lago 4

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

Soluzione

La soluzione corretta è la seguente:

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



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

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

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

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

Questa è l'informatica!

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

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

Parole chiave e siti web

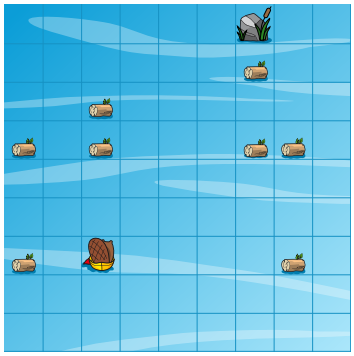
- Programmazione: `https://it.wikipedia.org/wiki/Programmazione_(informatica)`
- Sequenza: `https://it.wikipedia.org/wiki/Struttura_di_controllo`

18. Il tronco prezioso

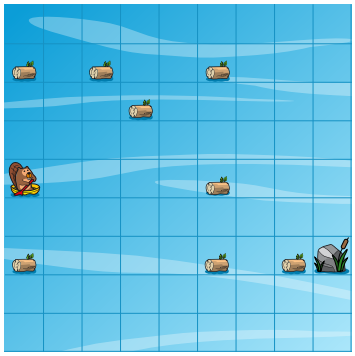
Petunia il castoro è alla ricerca di tronchi preziosi nella regione del Seeland. Il tronco più prezioso si trova sempre proprio accanto a un masso. Aiuta Petunia a scrivere le istruzioni per arrivare, in tutti e tre i laghi, alla casella con il tronco prezioso vicino al masso. Le istruzioni devono funzionare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

Puoi usare le seguenti istruzioni:

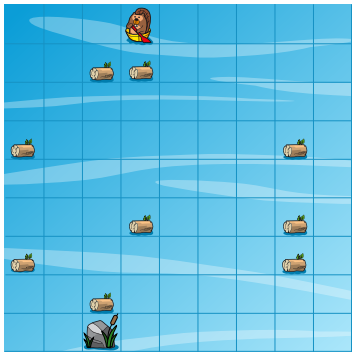
Istruzione	**Descrizione**
`move()`	Petunia si muove in avanti di una casella nella direzione in cui guarda.
`turnRight()` / `turnLeft()`	Petunia ruota sul posto di 90 gradi verso destra / sinistra.
`goToLog()`	Petunia avanza finché arriva su una casella con un tronco.



Lago 1



Lago 2



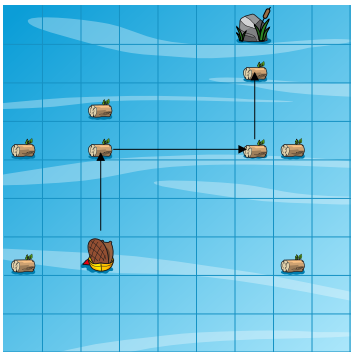
Lago 3

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

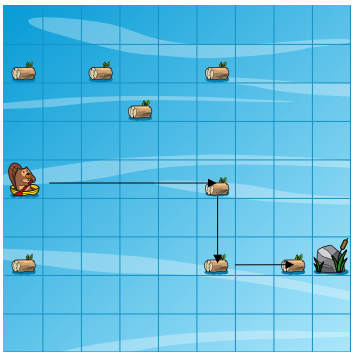
Soluzione

La soluzione corretta è la seguente:

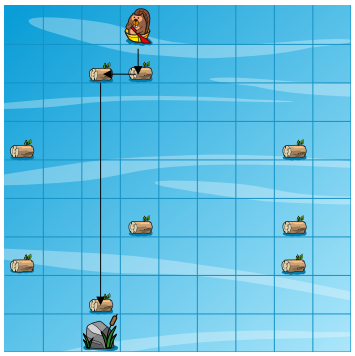
```
goToLog()
turnRight()
goToLog()
turnLeft()
goToLog()
```



Lago 1



Lago 2



Lago 3

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

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

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

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

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

Questa è l'informatica!

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

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

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

Parole chiave e siti web

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

A. Autori dei quesiti

Masiar Babazadeh

Wilfried Baumann

Gi Soong Chee

Byeonggyu Cho

Vladimir Costas

Valentina Dagienė

Christian Datzko

Nora A. Escherle

Abeer Eshra

Gerald Futschek

Christian Giang

Silvan Horvath

Alisher Ikramov

David Khachatryan

Doyong Kim

Jihye Kim

Vaidotas Kinčius

Stefan Koch

Lukas Lehner

Taina Lehtimäki

Gunwoong Lim

Mattia Monga

Anna Morpurgo

Kamohelo Motloung

Justina Oostendorp

Jean-Philippe Pellet

Emiliano Pereiro

Zsuzsa Pluhár

Wolfgang Pohl

Pedro Ribeiro

Kirsten Schlüter

Dirk Schmerenbeck

Vipul Shah

Jacqueline Staub

Nikolaos Stratis

Susanne Thut

Christine Vender

Florentina Voboril

Michael Weigend

Philip Whittington

Kyra Willekes

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/