INFORMATIK-BIBER SCHWEIZ
CASTOR INFORMATIQUE SUISSE
CASTORO INFORMATICO SVIZZERA

Quesiti e soluzioni 2023

9º e 10º anno scolastico

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

A cura di:

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

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 hep/ Scuola universitaria professionale della Svizzera italiana SUPSI

Hanno collaborato al Castoro Informatico 2023

Masiar Babazadeh, Susanne Datzko-Thut, Jean-Philippe Pellet, Giovanni Serafini, Bernadette Spieler

Capo progetto: Nora A. Escherle

Un particolare ringraziamento per il lavoro sui quesiti del concorso Svizzero va a:
Juraj Hromkovič, Angélica Herrera Loyo, Regula Lacher und Manuel Wettstein: ETH Zürich, Ausbildunges- und Beratungszentrum für Informatikunterricht
Tobias Berner: Pädagogische Hochschule Zürich
Christian Datzko: Wirtschaftgymnasium und Wirtschaftsmittelschule, Basel
Fabian Frei: CISPA - Helmholtz-Zentrum für Informationssicherheit
Sebastian Knüsli: Gymnasium Kirschgarten, Basel

La scelta dei quesiti è stata svolta in collaborazione con gli organizzatori dei concorsi in Germania, Austria, Ungheria, Slovacchia e Lituania. Ringraziamo specialmente:
Valentina Dagienė, Vaidotas Kinčius: Bebras.org, Lituania
Wolfgang Pohl, Jakob Schilke: Bundesweite Informatikwettbewerbe (BWINF), Germania
Hannes Endreß: Materna Information & Communications SE, Germania
Ulrich Kiesmüller: Simon-Marius-Gymnasium Gunzenhausen, Germania
Kirsten Schlüter: Bayerisches Staatsminiserium für Unterricht und Kultus, Germania
Margareta Schlüter: Universität Tübingen, Germania
Jacqueline Staub: Universität Trier, Germania
Michael Weigend: WWU Münster, Germania
Wilfried Baumann, Liam Baumann, Josefine Hiebler: Österreichische Computer Gesellschaft, Austria
Gerald Futschek: Technische Universität Wien, Austria
Zsuzsa Pluhár: ELTE Informatikai Kar, Ungheria

La versione online del concorso è stata creata su cuttle.org. Ringraziamo per la buona collaborazione:
Eljakim Schrijvers, Justina Dauksaite, Arjan Huijsers, Dave Oostendorp, Alieke Stijf, Kyra Willekes: cuttle.org, Olanda
Chris Roffey: UK Bebras Administrator, Regno Unito

Per il supporto durante le settimane del concorso ringraziamo:
Hanspeter Erni: Direttore scuola media di Rickenbach
Gabriel Thullen: Collège des Colombières, Versoix

Ringraziamo l'ETH per l'organizzazione e lo svolgimento della finale del Castoro:
Dennis Komm, Hans-Joachim Bückenhauer, Jan Lichensteiger, Moritz Stocker: ETH di Zurigo, Ausbildunges- und Beratungszentrum für Informatikunterricht
Per la correzione dei compiti finali:
Fiona Binder, Joel Birrer, Marlene Bötschi, Danny Camenisch, Gianluca Danieletto, Alexander Frey, Sven Grübel, Laure Guerrini, Charlotte Knierim, Richard Královič, Yanik Künzi, Kenli Lao, Sandro Marchon, Zoé Meier, Dario Näpfer, Kai Zürcher

Per la traduzione dei compiti finali in francese:
Jan Schönbächler: Lycée-Collège de l'Abbaye de St-Maurice

Christoph Frei: Chragokyberneticks (Logo Informatik-Biber Schweiz)
Andrea Leu, Maggie Winter, Lena Frölich: 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 2023 è stato organizzato dalla Società Svizzera per l'Informatica nell'Insegnamento (SSII) con il sostegno determinante della fondazione Hasler. Gli sponsor del concorso sono l'Ufficio per l'economia e il lavoro del Cantone di Zurigo e UBS.

Questo quaderno è stato creato il 27 ottobre 2024 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'01.12.2023.

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 2023 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 («Piccolo Castoro»)
- 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., Piccolo Castoro: 27 punti, 5º e 6º anno scolastico: 36 punti).

Il punteggio massimo totalizzabile era dunque pari a 180 punti (risp., Piccolo castoro: 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:

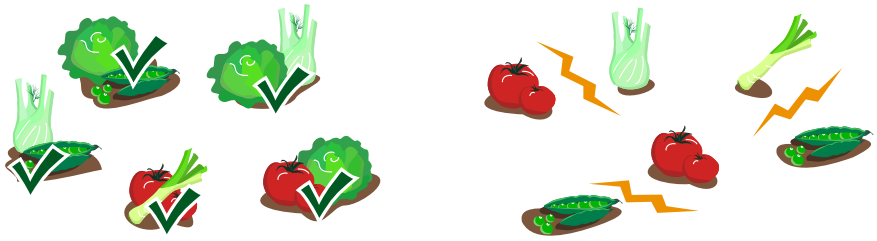
SVIA-SSIE-SSII Società Svizzera per l'Informatica nell'Insegnamento
Castoro Informatico
Masiar Babazadeh

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

Indice

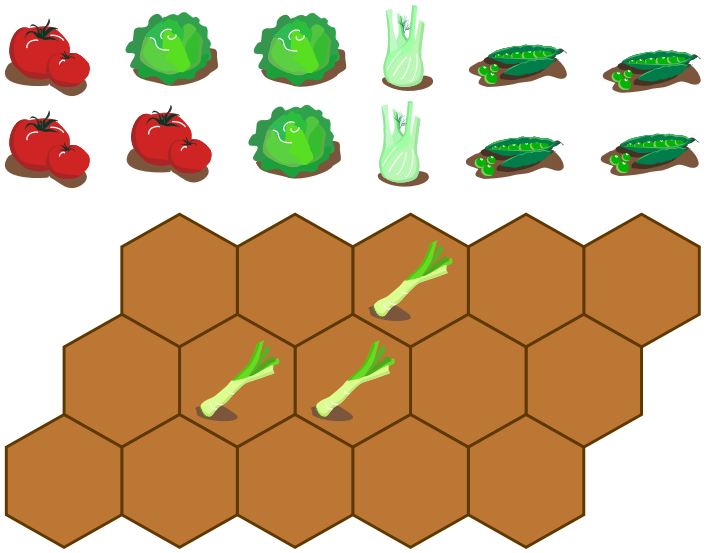
1. Orto di Lisa

Lisa crea un orto. Vuole piantare cinque ortaggi diversi. Alcuni ortaggi vanno d'accordo tra loro ✔, altri no ↯:



Lisa ha diviso l'orto in aree esagonali. Vuole piantare esattamente un ortaggio in ogni area.

Lisa ha già piantato porri  in tre aree.

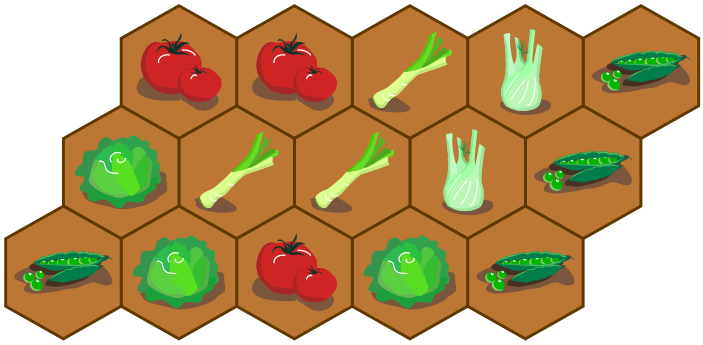


Quando si pianta, Lisa osserva la seguente regola: gli ortaggi che non vanno d'accordo non devono essere piantati in zone che si toccano.

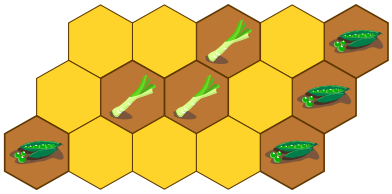
Pianta tutte le aree ancora libere seguendo la regola di Lisa!

Soluzione

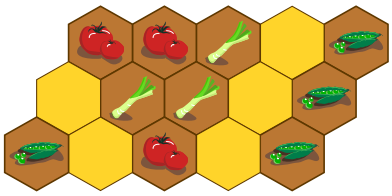
La risposta gista:



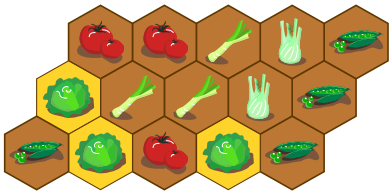
Poiché i piselli non vanno d'accordo con i porri, Lisa non pianta i piselli nelle aree chiare. Solo le aree rimanenti rimangono per i piselli.



Poiché i pomodori non vanno d'accordo con i piselli, Lisa non pianta i pomodori nelle aree chiare. Può piantare i pomodori nelle altre zone; i pomodori vanno d'accordo con i porri.



Poiché i pomodori non vanno d'accordo con i finocchi, Lisa non li pianta nelle aree chiare. Può piantare il finocchio nelle due aree tra i porri e i piselli. Può piantare la lattuga nelle aree chiare: Lisa non è a conoscenza di alcuna discordanza tra gli ortaggi vicini e la lattuga.



Questa è l'informatica!

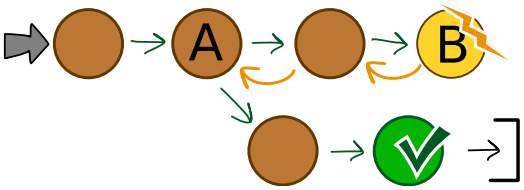
Se si vuole piantare ortaggi in modo che il raccolto sia il più abbondante possibile, si deve osservare molte *condizioni*: Ad esempio, le singole varietà hanno esigenze diverse in termini di spazio, nutrienti e luce. In questo compito consideriamo solo un tipo di condizione: la compatibilità tra le varietà di ortaggi.

Per trovare un piano per l'orto di Lisa che rispetti tutte le condizioni di compatibilità, si potrebbe procedere in questo modo: si provano sistematicamente tutte le combinazioni per disporre gli ortaggi sull'orto. Solo quando l'orto è pieno, si verifica se questa combinazione soddisfa tutte le condizioni ed è una soluzione al problema di Lisa. In informatica, tale prova di tutte le combinazioni è nota come metodo *forza bruta*. Per problemi con molte combinazioni e poche soluzioni, procedere secondo questo metodo può richiedere molto tempo.

Pertanto, di solito è meglio procedere per gradi e considerare tutte le condizioni a ogni passo. In questo modo possiamo trovare la soluzione al problema di Lisa, una combinazione o una piantumazione «sbagliata» dell'orto infatti non può verificarsi.

Fortunatamente, la soluzione si può trovare in modo diretto: ci sono sempre aree in cui possiamo piantare alcuni degli ortaggi rimasti. Questo di solito non funziona sempre.

Se si cerca di assemblare la soluzione passo dopo passo, ci possono essere diverse possibilità di soddisfare tutte le condizioni in un unico passo A.



A seconda della scelta, in una fase successiva B potrebbero non esserci più opzioni. Quindi si fanno gli ultimi passi indietro fino ad arrivare al passo A con diverse possibilità. A questo punto si sceglie un'altra possibilità e si cerca di trovare una soluzione.

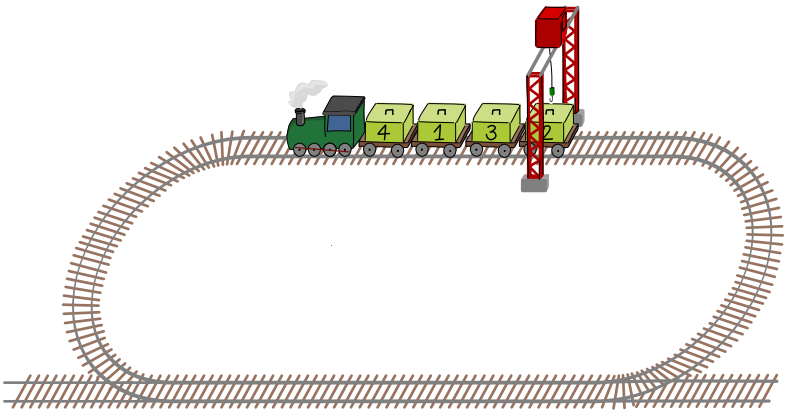
In informatica, questo ritorno sui propri passi è noto come *backtracking*.

Parole chiave e siti web

- Metodo forza bruta: `https://it.wikipedia.org/wiki/Metodo_forza_bruta`
- Backtracking: `https://it.wikipedia.org/wiki/Backtracking`

2. Scarico del treno

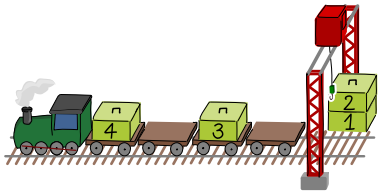
Un treno traina vagoni con casse numerate. La gru si trova in una posizione fissa e scarica le casse. Per scaricare una cassa, questa deve essere posizionata direttamente sotto la gru.



La gru deve scaricare le casse, partendo da 1, in ordine crescente. Il treno può andare solo in avanti. Quando è passato sotto la gru, deve fare un giro per poter scaricare altre casse.

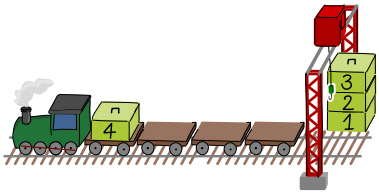
In questo modo la gru scarica le casse 1, 2, 3 e 4 nell'ordine corretto:

Turno 1:



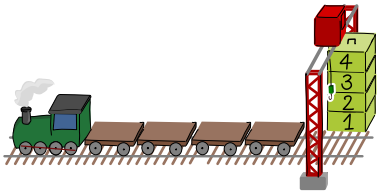
Salta la cassa 4, scarica la cassa 1, salta la cassa 3 e scarica la cassa 2 2.

Turno 2:



Salta la cassa 4 e scarica la cassa 3.

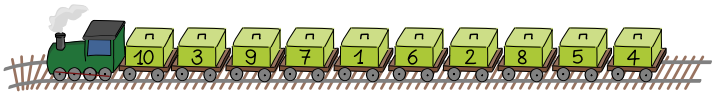
Turno 3:



Scarica la cassa 4.

Quindi il treno deve percorrere tre giri affinché tutte le casse siano scaricate nell'ordine corretto.

Quanti turni sono necessari per scaricare il seguente treno?



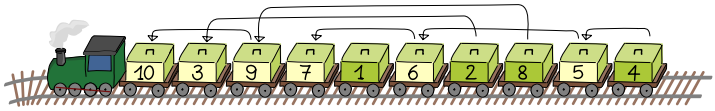
A) 1 turno
B) 2 turni
C) 3 turni
D) 4 turni
E) 5 turni
F) 6 turni
G) 7 turni
H) 8 turni
I) 9 turni
J) 10 turni

Soluzione

La risposta corretta è 7 turni.

L'ordine prescritto per lo scarico è 1, 2, 3, 4, 5, 6, 7, 8, 9, 10. Al primo turno, la gru scarica le casse 1 e 2 insieme. Nel secondo turno, la gru scarica insieme 3 e 4, poi 5, poi 6, poi 7 e 8 insieme, poi 9 e infine 10. Questo corrisponde a 7 turni.

In alternativa, si può sfruttare il fatto che ogni volta che viene richiesto il numero di casella successivo a sinistra di uno dei numeri di casella della sequenza, è necessario un ulteriore giro di scarico.



Ad esempio, poiché il 3 si trova a sinistra del 2, viene saltato per scaricare il 2, quindi è necessario un giro supplementare per portare il 3 sotto la gru. Nella mossa data ci sono sei coppie di questo tipo (2,3), (4,5), (5,6), (6,7), (8,9) e (9,10), quindi sono necessari altri 6 turni, per un totale di 7 turni.

Questa è l'informatica!

Se per un numero qualsiasi della sequenza 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 la cassa con il numero successivo più grande si trova più a sinistra sul treno, si parla di *inversione*. Ogni inversione di questo tipo richiede un giro in più. Se contiamo il numero di inversioni, otteniamo la risposta.

Il conteggio delle inversioni rispetto a una sequenza desiderata ha molte applicazioni. In alcuni algoritmi di ordinamento, come il *bubble sort*, il numero di inversioni ci dice quante permutazioni sono necessarie per ordinare una particolare sequenza. Quando due clienti classificano lo stesso insieme di articoli, il numero di inversioni nelle loro classifiche ci dice quanto le loro preferenze siano simili. Questa funzione viene utilizzata dai negozi online per identificare i clienti «simili» e consigliare loro i prodotti.

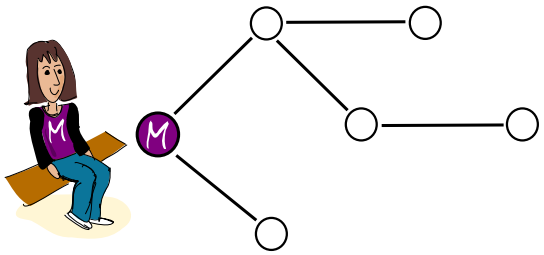
Parole chiave e siti web

- Algoritmo di ordinamento: `https://it.wikipedia.org/wiki/Algoritmo_di_ordinamento`
- Bubble sort: `https://it.wikipedia.org/wiki/Bubble_sort`

3. Il villaggio di Martina

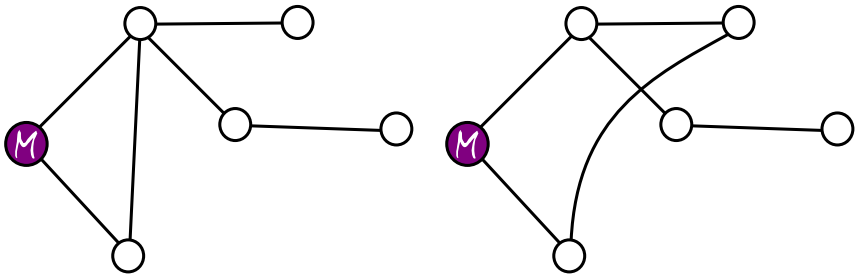
Nel villaggio di Martina ci sono sei case. Ci sono anche sentieri che possono essere utilizzati per camminare da una casa all'altra. Martina ha bisogno della stessa quantità di tempo per tutti questi percorsi.

Martina ha disegnato una mappa speciale del villaggio. In essa ha disegnato i percorsi esatti che può utilizzare per raggiungere le altre case.



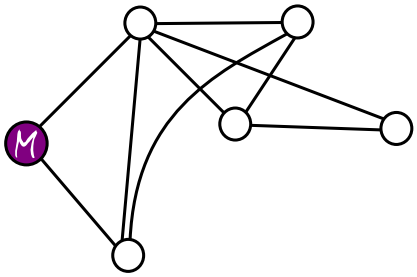
C'è anche una vera e propria mappa del villaggio, con tutti i sentieri.

Quale di questi disegni non può essere la mappa corretta?

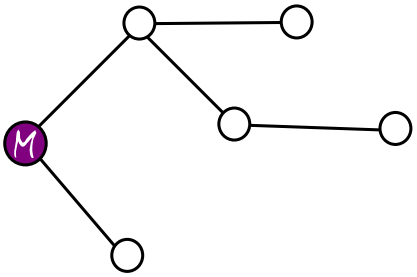


A)

B)

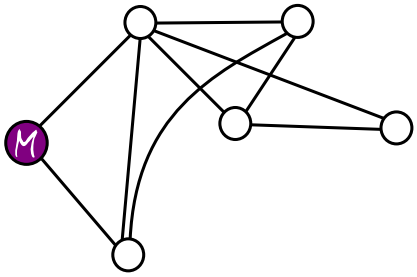


C)



D)

Soluzione

La risposta C è corretta: 

La mappa di Martina mostra che il modo più veloce per raggiungere la casa all'estrema destra è attraverso tre percorsi. Se C fosse la mappa giusta del villaggio, Martina potrebbe raggiungere questa casa più velocemente, cioè attraverso due sentieri. Quindi C non può essere la mappa giusta del villaggio.

Con le mappe A, B e D non c'è modo di raggiungere una delle altre case più velocemente che attraverso i percorsi della mappa speciale di Martina. Quindi queste mappe potrebbero essere vere e proprie mappe del villaggio.

Questa è l'informatica!

Martina è un'esperta di informatica. Ha disegnato la sua mappa come un *grafo*. I grafi sono costituiti da *nodi* (qui le case) che possono essere collegati da *bordi* (qui i percorsi). Sono adatti a modellare la realtà in molte aree dell'informatica e anche in questo compito.

Martina sa che esiste un'intera gamma di algoritmi per i grafi, ad esempio la cosiddetta ricerca in ampiezza, per risolvere compiti come «Qual è il modo più veloce per raggiungere un'altra casa?». Forse ha creato la sua particolare mappa del villaggio utilizzando una ricerca in ampiezza su un grafo più grande, la vera mappa del villaggio con tutti i percorsi.

Nella teoria dei grafi, che si occupa di grafi e algoritmi di grafi, la mappa di Martina corrisponde a un sottografo della mappa complessiva del villaggio. Il sottografo di Martina ha due caratteristiche particolari:

- Tutti i nodi sono collegati direttamente (tramite un bordo) o indirettamente (tramite più bordi).
- Non importa quali due nodi si scelgano a caso, esiste sempre un solo percorso tra i due.

Un grafo con queste caratteristiche è chiamato *albero* in informatica. La casa di Martina rappresenta la *radice* dell'albero. Dalla radice, Martina può raggiungere tutti gli altri nodi (le altre case del villaggio) lungo un unico percorso. Il sottografo di Martina è quindi un albero; inoltre, contiene tutti i nodi dell'intero grafo (l'intera mappa del villaggio) - ma forse non tutti i bordi. Un sottografo con queste proprietà è chiamato *albero ricoprente* dell'intero grafo.

In informatica ci sono molte applicazioni per gli algoritmi a grafo, soprattutto nel contesto delle reti (reti di traffico, reti di telecomunicazione, ...), ad esempio nel calcolo dei percorsi nei sistemi di

navigazione. Gli alberi ricoprenti possono essere utilizzati per la costruzione di reti a basso costo e possono essere utili per risolvere problemi particolarmente difficili.

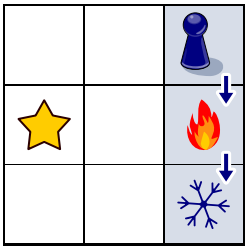
Parole chiave e siti web

- Grapho: `https://it.wikipedia.org/wiki/Grafo`
- Albero: `https://it.wikipedia.org/wiki/Albero_(grafo)`
- Ricerca in ampiezza: `https://it.wikipedia.org/wiki/Ricerca_in_ampiezza`
- Albero ricoprente: `https://it.wikipedia.org/wiki/Albero_ricoprente`

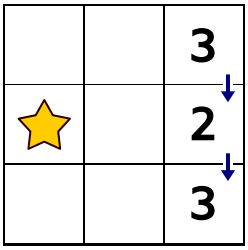
4. Più caldo, più freddo

Nina e Daniel giocano alla caccia al tesoro. Su una tavola con quadrati, Nina seleziona un quadrato e lo tiene a mente. Il tesoro è nascosto lì.

Daniel sceglie un campo di partenza. Da lì, sposta il suo pezzo da gioco di uno spazio alla volta: a sinistra, a destra, in alto o in basso.



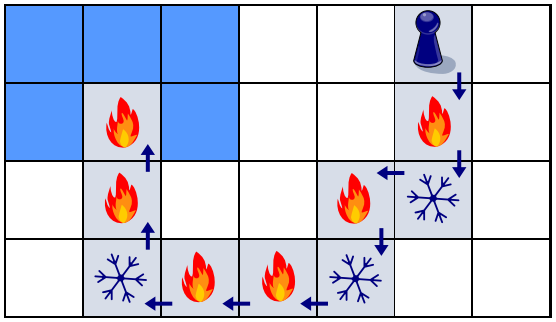
Al primo tentativo, prendono un piccolo tabellone di gioco. Nina nasconde il tesoro nella casella con la stella. Daniel inizia in alto a destra e fa due passi lungo le frecce. Dopo ogni passo, Nina dice se Daniel è più vicino al tesoro o più lontano dal tesoro rispetto a prima del passo.



L'immagine a destra mostra le distanze di Daniel dal tesoro. La distanza dal tesoro è il minor numero di passi che Daniel potrebbe attualmente compiere per raggiungere il tesoro.

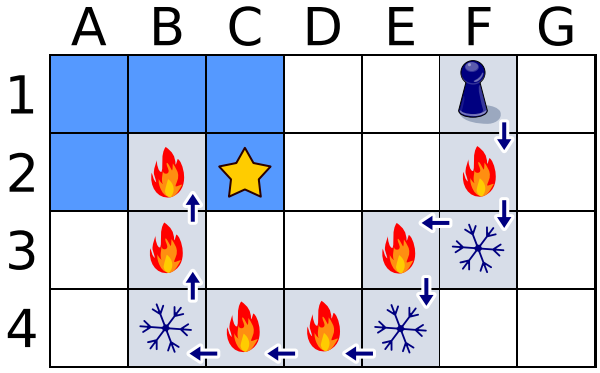
Adesso prendono una tavola più grande. Nina nasconde il tesoro in uno dei campi contrassegnati in blu. L'immagine mostra nuovamente i passi di Daniel e ciò che Nina dice dopo ogni passo.

Dove è nascosto il tesoro?



Soluzione

La risposta corretta:



Seguiamo il percorso di Daniel e il riscontro di Nina. Daniel inizia nella riga 1 del tabellone. Dopo il primo passo si trova nella riga 2 e più vicino al tesoro rispetto alla riga 1. Dopo il passo successivo si trova nella riga 3 e di nuovo più lontano dal tesoro. Dato che è rimasto nella stessa colonna, il tesoro deve trovarsi su una casella della riga 2, infatti non importa in quale colonna sia nascosto il tesoro: la via più breve per raggiungere il tesoro da un'altra colonna è quella di trovarsi nella stessa riga.

Ma in quale colonna è nascosto il tesoro? Continuando il suo cammino, Daniel si avvicina inizialmente al tesoro della riga 4 facendo qualche passo verso sinistra; in particolare, è più vicino al tesoro della colonna 3 che a quello della colonna 4. Ma dopo l'ultimo passo della riga, Daniel si trova più lontano dal tesoro della colonna 2 che da quello della colonna 3. Quindi il tesoro deve trovarsi in un quadrato della colonna 3, siccome quanto detto sopra per le colonne vale anche per le righe: la via più breve per raggiungere il tesoro da un'altra riga è quella di trovarsi nella stessa colonna.

Questa è l'informatica!

Daniel cammina (con il suo pezzo di gioco) attraverso il tabellone. Da ogni casella su cui si trova attualmente, Nina misura la distanza dalla casella con il tesoro e la utilizza per il suo feedback. Di solito, la distanza tra due punti viene misurata come la lunghezza del collegamento rettilineo tra i punti (distanza euclidea). Ma i due campi non sono, in senso stretto, dei punti. Pertanto, Nina misura la distanza tra due campi nel numero di passi che Daniel dovrebbe fare per il percorso più breve da un campo all'altro. Questa *misura* può essere generalmente applicata alle griglie ed è nota in informatica come *distanza di Manhattan*, derivata dalla pianta a griglia del quartiere newyorkese di Manhattan.

Gli informatici scelgono il modo di calcolare la distanza tra due oggetti a seconda del quesito che vogliono risolvere. Ad esempio, se si vuole misurare la distanza tra due parole della stessa lunghezza in un linguaggio naturale, si può contare il numero di punti in cui le parole differiscono; si tratta quindi della *distanza di Hamming*. Se le parole sono di lunghezza diversa, si può usare la *distanza di Levenshtein*. Le distanze giocano spesso un ruolo nell'informatica quando si tratta di trovare soluzioni ottimali a un problema. Non importa se la soluzione a un problema deve essere la più veloce, la

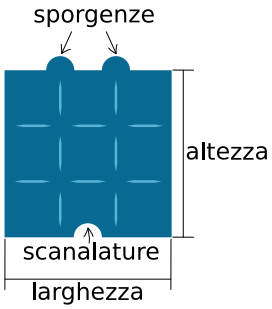
più breve o la più economica: spesso non è necessario cambiare l'algoritmo, ma solo la misura della distanza: durata, lunghezza o costo.

Parole chiave e siti web

- Distanza di Manhattan: `https://it.wikipedia.org/wiki/Geometria_del_taxi`
- Distanza di Hamming: `https://it.wikipedia.org/wiki/Distanza_di_Hamming`
- Distanza di Levenshtein: `https://it.wikipedia.org/wiki/Distanza_di_Levenshtein`

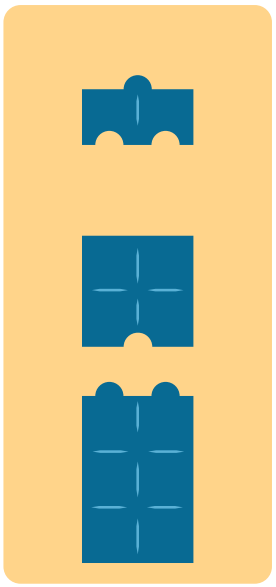
5. Mattoni di castoro

I mattoni di castoro Otto si differenziano per quattro caratteristiche:



1. larghezza: stretta, media, larga
2. altezza: piccola, media, grande
3. numero di sporgenze sulla parte superiore: zero, uno, due
4. numero di scanalature nella parte inferiore: zero, una, due.

Otto divide i mattoni in gruppi di tre. Lo fa in modo che per ogni gruppo valga quanto segue: i tre mattoni hanno per ciascuna delle quattro proprietà ...



- ... o tutte con lo stesso valore ...
- ... o tutte con tre valori diversi.

A destra, uno dei gruppi di Otto.

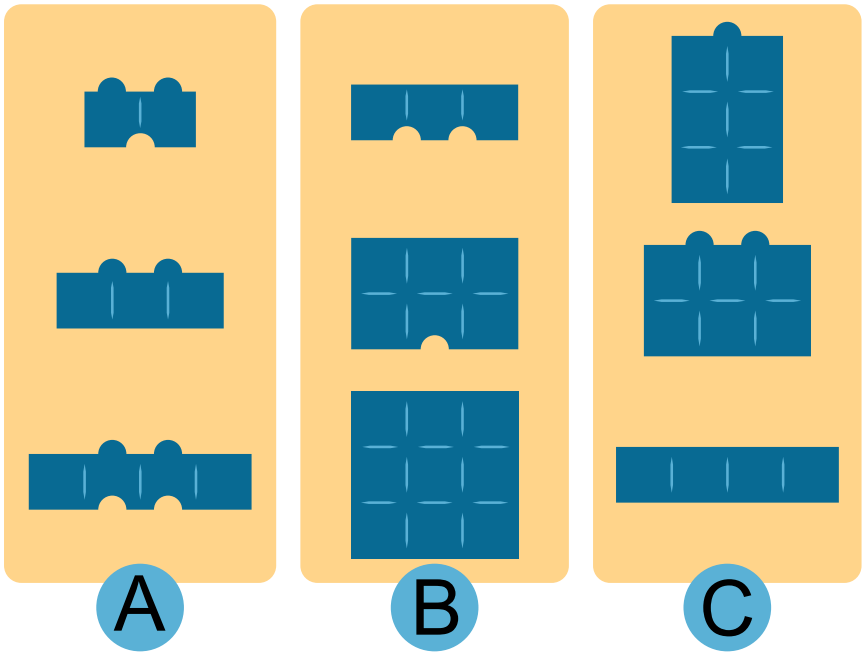
Perché questi tre mattoni hanno tutti

- la stessa larghezza,
- diverse altezze,
- numero diverso di sporgenze e
- numero diverso di scanalature.

Dividi questi mattoni in gruppi di tre, come farebbe Otto.

Soluzione

La risposta giusta:



I mattoni sono divisi in gruppi secondo le regole di Otto. La tabella mostra i tre gruppi, per i quali i valori delle proprietà sono tutti diversi o tutti uguali.

Proprietà	**Gruppo A**	**Gruppo B**	**Gruppo C**
Larghezza	diversa	uguale	diversa
Altezza	uguale	diversa	diversa
Sporgenze	uguali	uguali	diversi
Scanalature	diverse	diverse	uguali

Ma è questo l'unico modo per dividere i mattoni come farebbe Otto?

Si può considerare che se i valori di una proprietà devono essere diversi in tutti i gruppi, i valori diversi devono verificarsi in tutti i mattoni tante volte quanti sono i gruppi. In caso contrario, deve esistere almeno un gruppo in cui i valori di questa proprietà sono tutti uguali.

Un'analisi più attenta di tutti i mattoni mostra che i valori di larghezza di 2 e 4 unità si verificano solo due volte. Deve quindi esistere un gruppo in cui tutti i mattoni hanno una larghezza di tre unità.

Dei cinque mattoni di larghezza di tre unità, nessuno ha una sola sporgenza. Pertanto, non è possibile formare un gruppo con un numero diverso di sporgenze. Ma ci sono tre mattoni con zero sporgenze - e tutti hanno altezze diverse e un numero diverso di scanalature. Pertanto, il gruppo B è l'unico gruppo possibile di mattoni con larghezza di tre unità.

Negli altri due gruppi, le larghezze devono essere tutte diverse.

Nei sei mattoni rimanenti, i valori di altezza di tre unità e due unità si verifica solo una volta. Deve quindi esistere un gruppo in cui tutti i mattoni hanno un'altezza di un'unità. Il gruppo A è l'unico gruppo possibile di tre mattoni con altezza di un'unità che corrisponde alle idee di Otto. Restano i tre mattoni del gruppo C. Anch'essi formano un gruppo di tre, proprio come vorrebbe Otto.

Questa è l'informatica!

In questo compito i mattoni sono descritti usando quattro *proprietà* (o *attributi*).

Per poter dividere i mattoni in gruppi di tre come vuole Otto, è necessario conoscere i valori delle proprietà di ciascun mattone.

Per questo, è sufficiente guardare ciascun elemento costitutivo. Un programma informatico che deve creare i gruppi di tre non può «vedere» e ha bisogno di una descrizione in una *struttura di dati*.

Ad esempio, i mattoni in una *base di dati* possono essere descritti come righe di una tabella. Le colonne della tabella corrispondono alle proprietà e ogni riga (chiamata anche *dataset*) contiene i valori di un mattone nelle colonne appropriate:

Mattone n.	**Larghezza**	**Altezza**	**Sporgenze**	**Scanalature**
1	1	3	1	0
2	2	2	2	0
...	...	...	...	...

La progettazione di tabelle di basi di dati è un'attività comune per gli informatici.

È necessario essere scrupolosi e considerare quali proprietà degli oggetti sono importanti per l'elaborazione da parte di un programma informatico. Le modifiche successive non sono così facili, soprattutto se i dati di molti oggetti sono già memorizzati.

Parole chiave e siti web

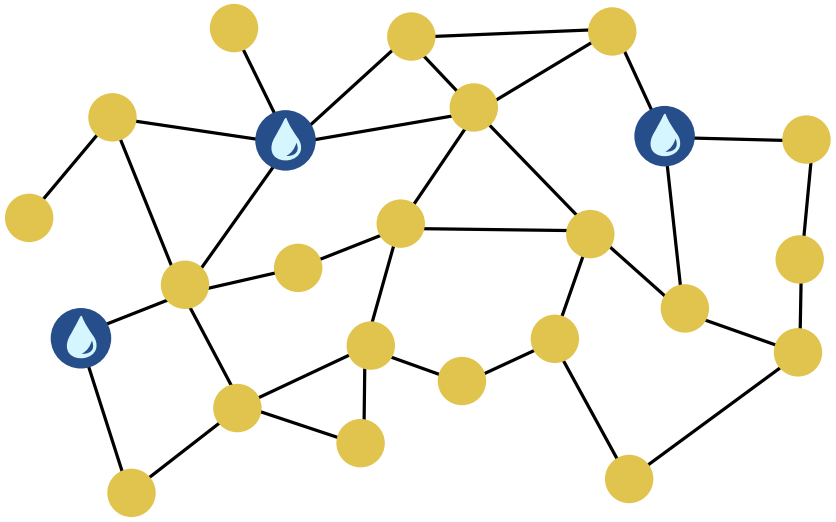
- Struttura dati: `https://it.wikipedia.org/wiki/Struttura_dati`
- Basi di dati: `https://it.wikipedia.org/wiki/Tabella_(basi_di_dati)`

6. Fontana

L'estate è calda in città. Per questo il sindaco ha fatto installare delle fontane con acqua potabile.

Le fontane devono essere posizionate in modo tale che per raggiungerle non si debbano percorrere più di due segmenti di strada da ogni angolo di strada. Solo in quel caso il sindaco sarà soddisfatto.

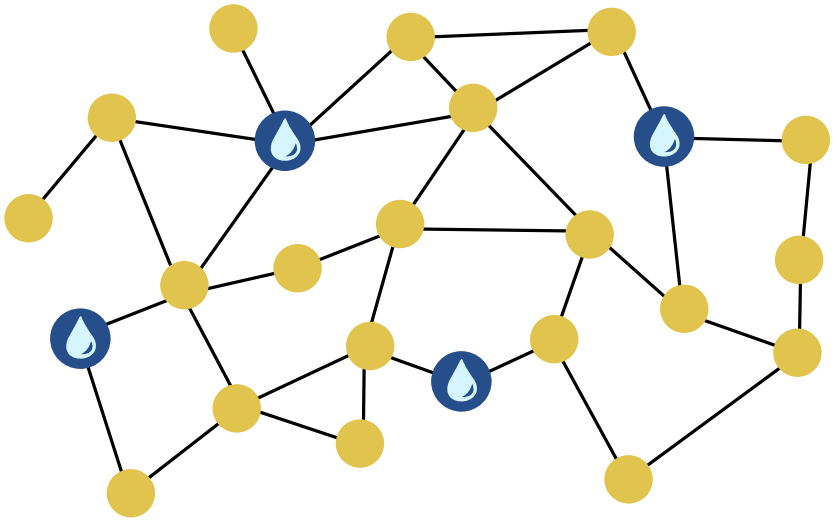
Ecco una mappa della città. Le linee sono segmenti di strada e i punti sono angoli di strada. In tre angoli ci sono già delle fontane .



Colloca un'altra fontana in modo che il sindaco sia soddisfatto.

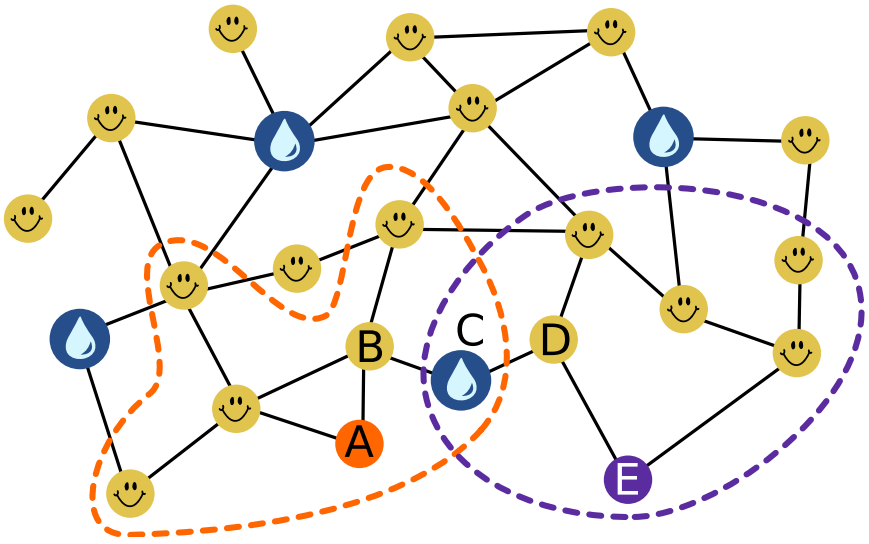
Soluzione

La risposta corretta:



Collocando un'altra fontana in basso al centro, per raggiungere una fontana si devono percorrere al massimo due segmenti di strada da ogni angolo di strada. Così facendo il sindaco sarà soddisfatto.

Come possiamo scoprire a quale angolo della strada si prevede l'installazione di un'altra fontana? Nella mappa della città segniamo tutti gli angoli delle strade con un , che non si trovino a più di due tratti stradali di distanza da una delle fontane già esistenti. Per quanto riguarda questi angoli, il sindaco può già ritenersi soddisfatto.



Per i cinque angoli di strada rimanenti, A, B, C, D ed E, posizioniamo un'altra fontana in C. In questo modo, da questi angoli alla fontana successiva si devono percorrere al massimo due segmenti di strada.

L'angolo C è l'unica posizione per una nuova fontna che lo consente. Se consideriamo per gli angoli A ed E rispettivamente tutti gli altri angoli che possono essere raggiunti attraverso due tratti di

strada (delineati con linee tratteggiate nella figura), l'angolo di strada C è l'unico che soddisfa questa condizione per A *e* E.

Questa è l'informatica!

La mappa della città può essere modellata come un *grafo*. Si tratta di uno strumento importante per l'informatica per modellare le relazioni tra gli oggetti e rispondere alle domande relative a queste relazioni. In questo caso, gli angoli delle strade possono essere intesi come oggetti e quindi come *nodi* del grafo. La relazione tra due oggetti è modellata nel grafo da *bordi*, che sono rappresentati come linee di collegamento. In questo caso, un bordo tra due angoli di strada significa che sono collegati da un segmento di strada. Questa relazione può essere chiamata vicinato. Tuttavia, i bordi possono modellare anche altre relazioni, come per esempio l'amicizia.

In questo compito, si deve trovare un sottoinsieme di nodi (per la creazione delle fontane) tale che ogni nodo al di fuori di questo sottoinsieme sia collegato tramite un percorso a un «nodo fontana» lungo al massimo due bordi. Nella terminologia informatica, questo si chiama trovare un «insieme dominante a distanza 2». In generale (per tutti i percorsi di lunghezza $k \geq 1$), la ricerca del più piccolo sottoinsieme possibile è uno dei problemi più difficili dell'informatica.

Questi «insiemi dominanti a distanza minima k» hanno assunto un ruolo sempre più importante negli ultimi tempi, soprattutto nel campo della *Social Computing* (in italiano anche *socioinformatica*): Per il trattamento automatico dei dati attraverso le reti sociali (ad esempio, per rilevare la diffusione di fake news) le relazioni di fan o follower tra gli utenti sono modellate come un grafo. Questi grafi possono essere così grandi che è possibile visualizzare solo una selezione rappresentativa di utenti (la più piccola possibile) - ad esempio, un «insieme dominante a distanza minima 3». Poiché la selezione veramente minima non può essere calcolata in modo efficiente, l'informatica sviluppa delle procedure, che calcolano le selezioni più piccole possibili, ma non garantite, in un tempo breve.

Parole chiave e siti web

- Socioinformatica: `https://it.wikipedia.org/wiki/Socioinformatica`

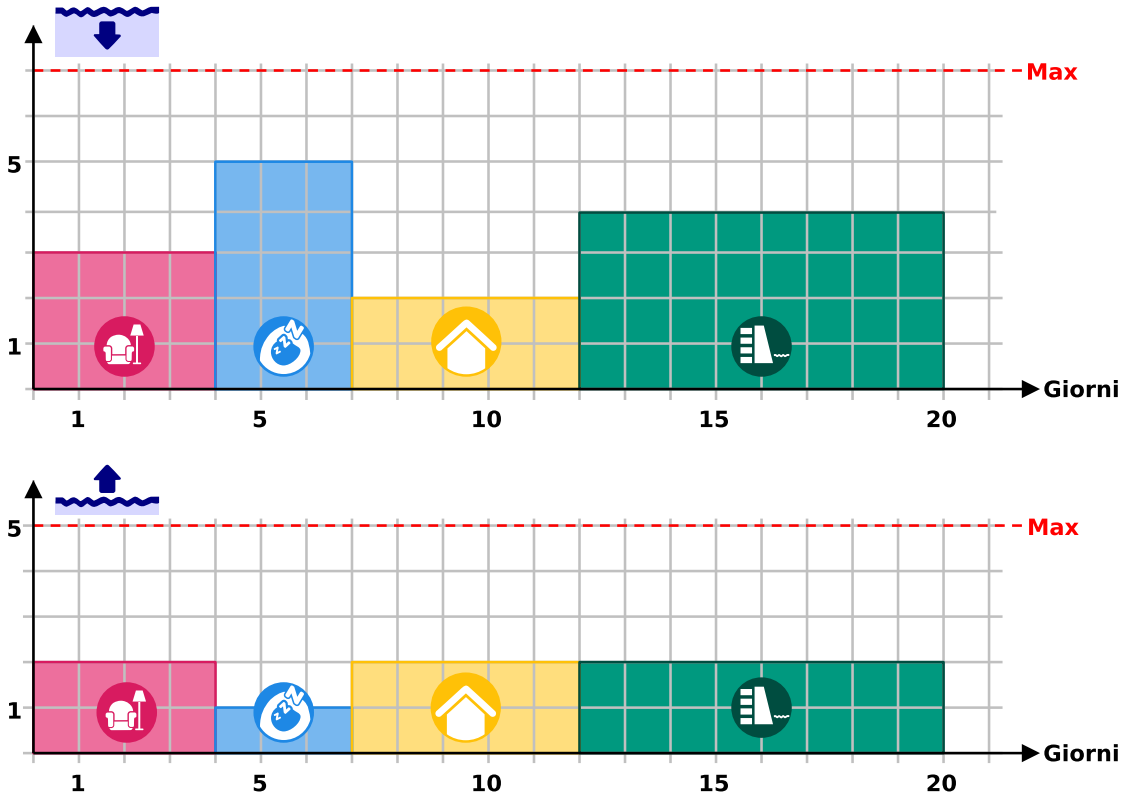
7. Castello per castori

Un castello per castori è composto da 4 parti, tutte in parte sott'acqua e in parte sopra l'acqua. Quando si costruisce una castello per castori, ogni operaio coinvolto lavora solo sotto l'acqua o solo sopra l'acqua. Per ogni parte, il lavoro viene svolto contemporaneamente sopra e sotto l'acqua. La tabella mostra, per ogni parte, quanto tempo occorre per costruire il castello per castori e quanti operai sono necessari sotto e sopra l'acqua per farlo.

Parti	**Salone**	**Grotta del sonno**	**Tetto**	**Diga**
Durata della costruzione	4 giorni	3 giorni	5 giorni	8 giorni
	3	5	2	4
	2	1	2	2

Il tetto può essere costruito solo quando la grotta del sonno è terminata! Per tutte le altre parti, l'ordine non ha importanza.

Per costruire un nuovo castello sono disponibili al massimo 7 operai subacquei e 5 operai sopra l'acqua. È anche possibile costruire diverse parti contemporaneamente. Ecco un piano di lavoro per finire il castello in 20 giorni.

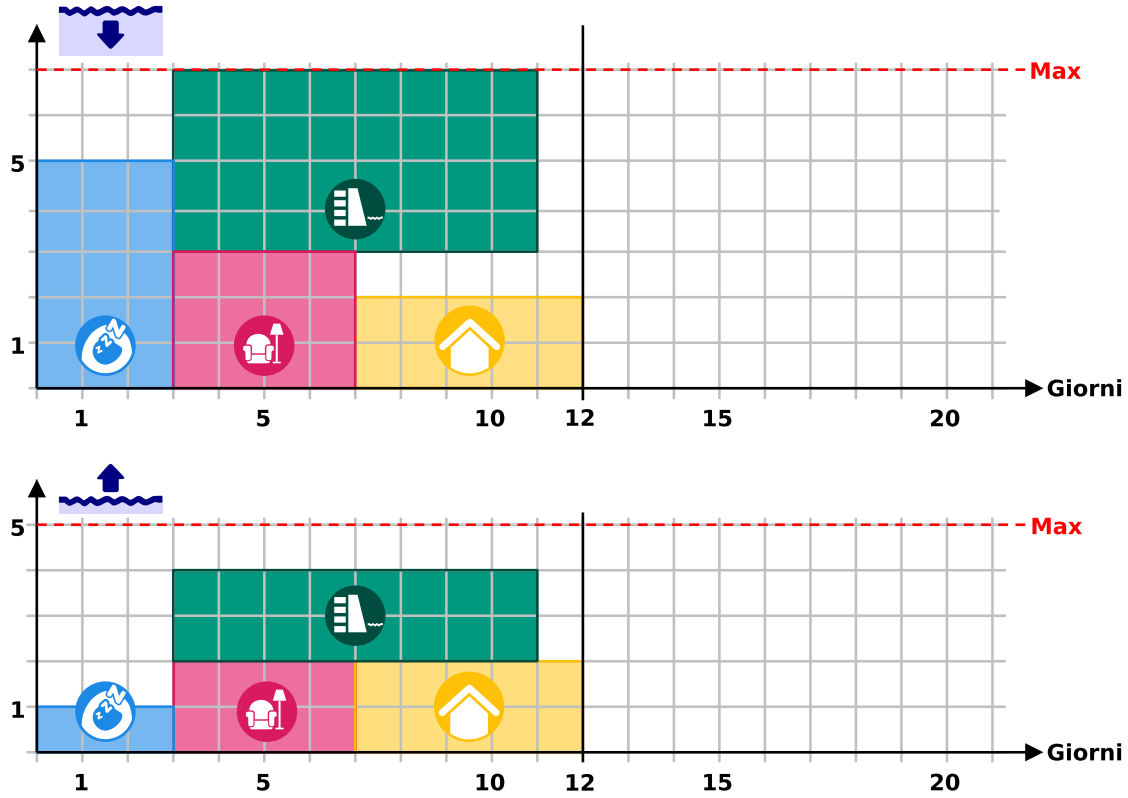


Elabora un piano per completare il castello per castori nel minor numero di giorni possibile. Quanti giorni sono necessari?

Soluzione

12 giorni è la risposta corretta.

Questo è un piano per finire il castello di castoro in 12 giorni:



Il piano con il tempo di costruzione più breve può essere determinato in due fasi:

1. Innanzitutto, la grotta del sonno deve essere progettata prima al tetto. Poiché la grotta richiede 5 operai subacquei, la diga 3 e il salone 4, la grotta - con la limitazione di 7 operai subacquei - non può essere costruita contemporaneamente alla diga o al salone. Pertanto, la grotta deve essere costruita per prima e le altre tre parti in seguito.
2. La diga e il salnoe possono essere costruiti contemporaneamente dopo la grotta, oppure una delle due parti contemporaneamente al tetto. Tuttavia, non è possibile costruire tutte e tre le parti contemporaneamente, perché insieme richiedono $3 + 4 + 2 = 9$ operai subacquei, un numero superiore a quello disponibile. Il tempo di costruzione più breve si ottiene se le due parti con i tempi di costruzione più brevi (tetto e salone) vengono costruite una dopo l'altra e la diga viene costruita contemporaneamente.

Questa è l'informatica!

Pianificare il corso ottimale, il più rapido possibile, di un progetto è un compito difficile in cui è necessario tenere conto di alcune condizioni. Spesso ci sono dipendenze temporali tra le sottoattività di un progetto; ad esempio, ci possono essere sottoattività che possono essere iniziate solo dopo

che un'altra sottoattività è stata completata, come nel caso del tetto e della grotta. Inoltre, ogni sottocompito richiede determinate risorse, come manodopera, tempo e attrezzature. Quando si pianifica un progetto, è utile poterlo rappresentare bene. I diagrammi mostrati in questo compito sono un tipo di diagramma di Gantt sviluppato da Henry Gantt (1861−1919) tra il 1910 e il 1915; rappresentazioni simili erano utilizzate indipendentemente da Gantt nello stesso periodo in Germania. Mostrano l'utilizzo delle risorse (in questo caso i due tipi di manodopera) nel tempo.

Il progetto ottimale per il castello di castoro può essere pensato nella vostra testa, provando tutte le possibilità consentite. Per i progetti più grandi, questo richiederebbe troppo tempo e diventerebbe troppo confuso. I programmi informatici possono essere d'aiuto in questo caso, ed è per questo che la gestione di programmi (*scheduling*) è un argomento importante dell'informatica. Come spesso accade per i problemi difficili, sono state sviluppate procedure che, invece di un piano ottimale garantito, producono un piano con un requisito temporale leggermente più ampio, ma comunque molto buono. La programmazione viene applicata anche al controllo dei computer stessi, i cui processi competono per le risorse (potenza di calcolo, accesso alla memoria, accesso a dispositivi esterni come dispositivi di memorizzazione, stampanti o interfacce di rete).

Parole chiave e siti web

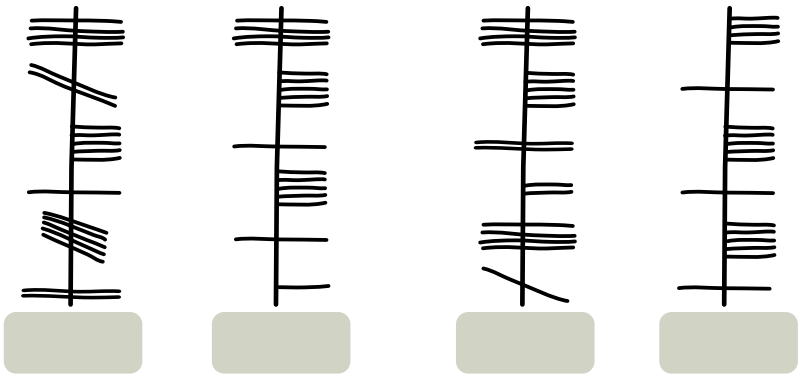
- Diagramma di Gantt: `https://it.wikipedia.org/wiki/Diagramma_di_Gantt`
- Software gestionale: `https://it.wikipedia.org/wiki/Software_gestionale`

8. Ogham

Sue conosce l'antico alfabeto irlandese Ogham. Ogni lettera è composta da uno o più tratti disposti su una lunga linea. Due lettere consecutive sono separate da uno spazio.

Sue usa l'Ogham come codice. Codifica quattro parole (i suoi tipi di frutta preferiti in tedesco): ANANAS, BANANE, MELONE e ORANGE.

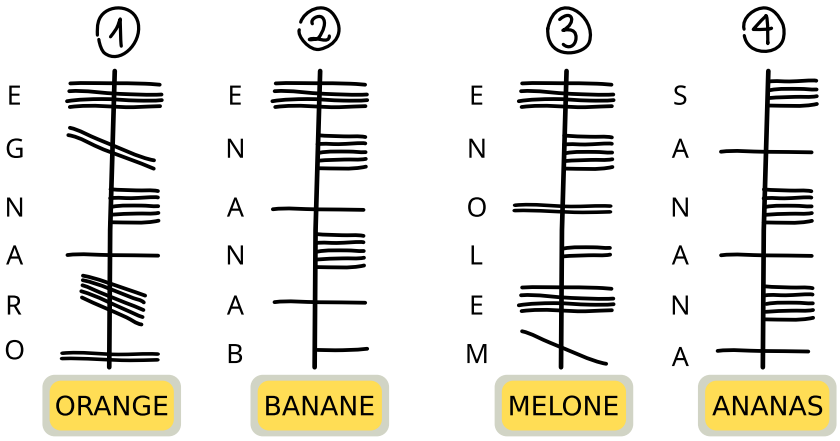
Quale parola corrisponde a quale codice Ogham?



ANANAS BANANE MELONE ORANGE

Soluzione

La risposta corretta:



Esistono vari modi per determinare l'assegnazione corretta. In ogni caso, però, bisogna scoprire in quale direzione sono scritte le lettere lungo la linea verticale. A questo proposito ci viene in aiuto la parola ANANAS, particolarmente suggestiva. In essa la lettera A ricorre tre volte, con una lettera diversa tra l'una e l'altra.

Solo nel codice Ogham 4 una lettera ricorre tre volte, e anche lì c'è una lettera in mezzo. Il codice 4 è quindi l'unico a cui si adatta la parola ANANAS. Questo dimostra che nell'Ogham le parole sono scritte dal basso verso l'alto e che la lettera A, che ricorre tre volte nell'Ogham, è scritta come una linea orizzontale che attraversa la linea verticale.

La lettera A in Ogham ricorre solo due volte nel codice 2. Anche a causa della codifica di N (cinque linee orizzontali a destra della linea) scoperta da ANANAS e della disposizione delle altre lettere, solo BANANE si adatta a questo codice. ORANGE si adatta solo al codice 1 perché la lettera A in Ogham si trova esattamente una volta. Ora rimane solo il codice 3; deve quindi essere la parola Ogham per MELONE e contiene le lettere Ogham E e N scoperte dalle altre parole nei posti appropriati.

Questa è l'informatica!

In questo compito, un testo sconosciuto deve essere decodificato o decifrato. Non si tratta di un compito molto difficile, perché il testo originale è noto. Inoltre, il testo sconosciuto è suddiviso in lettere e parole allo stesso modo del testo noto. Quando si decifra un testo segreto o un testo in una scrittura sconosciuta di cui non si conosce il testo in chiaro, spesso è utile pensare alla frequenza delle lettere e delle parole e su questa base cercare di trovarle nel testo. Alcuni alfabeti e scritture antiche sono stati decifrati in questo modo. Diventa difficile, tuttavia, quando i caratteri del testo sconosciuto non sono così facili da assegnare alle lettere e alle parole della lingua conosciuta, come nel caso dell'Ogham. In questi casi, l'unico modo per aiutarsi è confrontare il testo con testi o scritture note, come in questo compito. Per esempio, i geroglifici egiziani non sono stati decifrati per secoli finché, per caso, è stata trovata una pietra con geroglifici e due scritture conosciute, la Stele di Rosetta. Lo stesso testo è stato trovato tre volte sulla pietra. Era scritto in lingue diverse, ma conteneva sempre

gli stessi nomi. In questo modo è stato possibile decifrare elementi essenziali dei geroglifici. Tuttavia, questo non vale per tutte le scritture: I circa 650 caratteri della cultura Maya non sono ancora stati completamente decifrati, così come le scritture Lineare A e Lineare B della regione mediterranea.

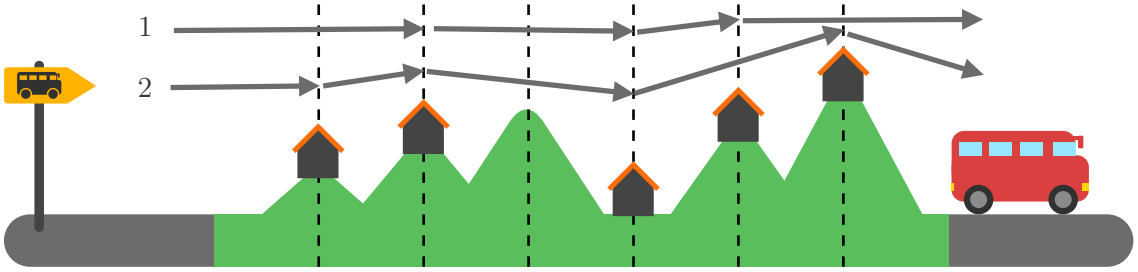
Anche in informatica i caratteri e i testi vengono decodificati, dopo essere stati precedentemente criptati per una trasmissione di dati a prova di intercettazione. Tuttavia, si utilizzano procedure completamente diverse rispetto alla codifica di parole in altre scritture. Codifiche così semplici sono troppo facili da decodificare, specialmente con l'aiuto dei computer, grazie alle considerazioni già citate sulla frequenza delle lettere e delle parole.

Parole chiave e siti web

- Crittografia: `https://it.wikipedia.org/wiki/Crittografia`
- Crittoanalisi: `https://it.wikipedia.org/wiki/Crittoanalisi`
- Alfabeto ogamico: `https://it.wikipedia.org/wiki/Alfabeto_ogamico`

9. Escursioni

A Mia piacciono le vacanze a piedi, in cui soggiorna ogni notte in un posto diverso. Per la sua prossima vacanza, Mia ha una mappa della regione. La mappa mostra il punto di partenza di Mia, la sua destinazione e tutti i luoghi in cui può soggiornare.



Mia ha diviso la regione in sezioni con linee tratteggiate. Può percorrere solo uno o due tratti alla volta in un giorno. Ha già messo sulla mappa due diverse passeggiate che può fare:

- L'escursione 1 prevede tre soggiorni
- L'escursione 2 prevede quattro soggiorni.

Mia può però fare altre escursioni.

Quante escursioni diverse può fare Mia in totale? Conta anche le escursioni 1 e 2.

A) 2 escursioni

B) 3 escursioni

C) 4 escursioni

D) 5 escursioni

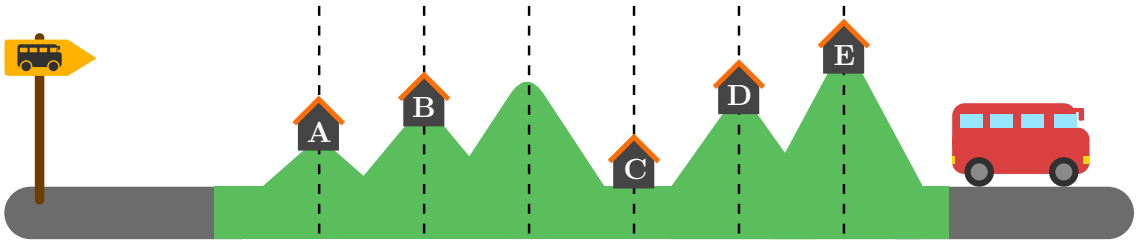
E) 6 escursioni

F) 7 escursioni

G) 8 escursioni

Soluzione

La risposta corretta è E) 6 escursioni.



Innanzitutto ci rendiamo conto che Mia deve pernottare a **B** e **C** perché la distanza tra questi due luoghi è la massima distanza (2) che può percorrere in un solo giorno. Quindi, per il tragitto da **B** a **C**, Mia ha solo un'opzione.

Ora possiamo determinare le possibilità per le altre parti del cammino: Dal punto di partenza ( ) a **B**, Mia può percorrerlo tutto d'un fiato o pernottare in **A** tra un tratto e l'altro; queste sono due possibilità (come nelle escursioni 1 e 2). Da **C** alla destinazione ( ) Mia deve percorrere tre tratti, e può pernottare dopo ogni tratto. Pertanto, può dividere l'intera camminata in tutte e tre le combinazioni di tratti 1 e 2:

- **C** → **D** → **E** → ;
- **C** → **E** → ;
- **C** → **D** → .

Quindi il numero totale di tutte le escursioni che Mia può fare è $2 \times 1 \times 3 = 6$.

Questa è l'informatica!

A volte il numero di possibilità per eseguire un determinato compito può essere molto grande. Ad esempio, esistono circa 14 milioni di modi per scegliere 6 numeri diversi da 1 a 49. E ci sono circa mezzo miliardo di modi per scrivere i numeri da 1 a 12 in diverse sequenze. Anche questo richiede al computer un po' di tempo.

È fortuito che in questo compito non ci sia un soggiorno dopo la terza sezione e che il conteggio di tutte le passeggiate che Mia può fare possa essere diviso in tre parti. Il problema del conteggio viene scomposto in tre problemi di conteggio più piccoli, per così dire. In informatica, la tecnica della *scomposizione del problema* è spesso utilizzata nella progettazione di algoritmi. Questo principio di soluzione è noto anche come *divide et impera*.

Alcuni importanti algoritmi di ordinamento, ad esempio, funzionano secondo questo principio. Anche la programmazione dinamica, un metodo per la soluzione algoritmica di problemi di ottimizzazione (descritto nel 1957 da Richard Bellman), si basa su questo principio: se si riconosce che le soluzioni ottimali di un problema sono composte dalle soluzioni ottimali di sottoproblemi, si può usare questo principio per «iniziare in piccolo», per così dire: In primo luogo, le soluzioni dei sottoproblemi più piccoli vengono calcolate direttamente e poi combinate per formare le soluzioni dei successivi

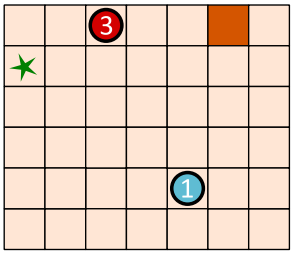
sottoproblemi più grandi. Questa operazione viene ripetuta fino a trovare la soluzione ottimale per il problema completo. Poiché le soluzioni parziali trovate spesso contribuiscono alle soluzioni di molte parti più grandi, vengono memorizzate per evitare di ripetere calcoli identici. La programmazione dinamica può anche essere molto utile per contare le possibilità, come in questo problema.

Parole chiave e siti web

- Decomposizione del problema, scomposizione
- Divide et impera: `https://it.wikipedia.org/wiki/Divide_et_impera_(informatica)`
- Programmazione dinamica: `https://it.wikipedia.org/wiki/Programmazione_dinamica`

10. Go-Bot

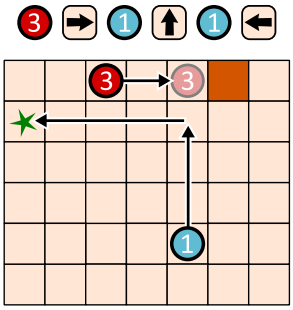
I Go-Bot sono robot molto semplici. Si muovono su una tavola con delle caselle.



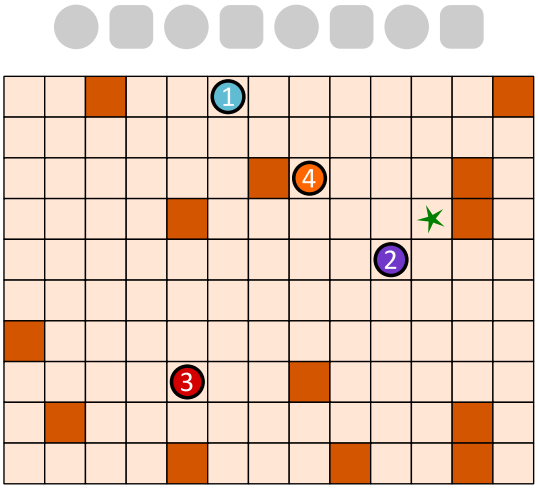
Per controllarli, bisogna prima selezionare uno dei go-bot. Quindi si invia il Go-bot in una direzione con un comando a freccia: su ⬆, giù ⬇, sinistra ⬅ e destra ➡. Il Go-Bot procede ostinatamente dritto finché non arriva direttamente davanti a un ostacolo 🟧 o a un altro robot. Rimane lì finché non riceve un nuovo commando.

Con un'abile sequenza di comandi si deve fare in modo che il Go-Bot 1 raggiunga l'obiettivo ✶ e che si fermi esattamente lì.

In basso a sinistra c'è una tavola con due Go-Bot. Con questa sequenza di comandi, il Go-Bot 1 raggiunge l'obiettivo ✶ - vedi sotto a destra:



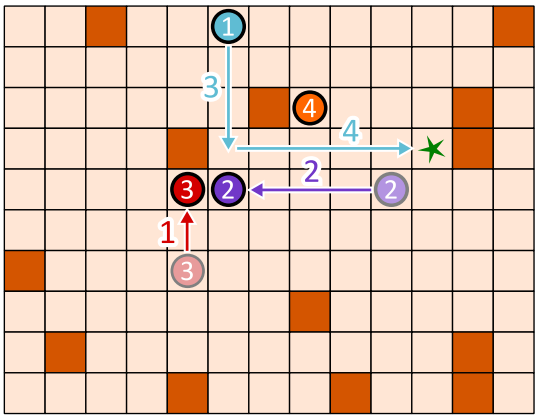
Crea una sequenza di comandi con quattro frecce che il Go-Bot 1 utilizza per raggiungere l'obiettivo ✶!



Soluzione

La risposta corretta: 3 ⬆ 2 ⬅ 1 ⬇ 1 ➡

Affinché il Go-Bot 1 raggiunga l'obiettivo attraverso una sequenza di comandi con quattro frecce, i tre Go-Bot devono cooperare. Prima 3 sale fino a fermarsi davanti a un ostacolo. Così diventa un ostacolo per 2 nel suo percorso verso sinistra. Se ora si invia 1 verso il basso, esso raggiunge 2 e da lì può andare a destra, dove si ferma prima dell'ostacolo - sulla stella.



Come trovare la giusta sequenza di comandi? Si può partire dal fondo e pensare a quale deve essere l'ultimo movimento del Go-Bot 1 verso l'obiettivo. Ci sono solo due possibilità:

a) Viene da sinistra, come nella nostra soluzione.
b) Viene dall'alto. In questo caso, il Go-Bot 4 dovrebbe essere spostato verso l'alto a destra con tre comandi per fungere da ostacolo per 1. Avremmo quindi bisogno di $3 + 2 = 5$ comandi.

Ma stiamo cercando una sequenza con quattro comandi. Quindi la possibilità a) deve essere corretta, in quanto il Go-Bot 1 arriva alla stella da sinistra. Quindi il penultimo movimento del Go-Bot 1 è dall'alto verso il basso. Affinché si fermi nel punto giusto, i robot 2 e 3 devono essere spostati prima come nell'immagine.

Questa è l'informatica!

In questo compito, diversi robot hanno lavorato insieme per raggiungere un obiettivo. Avevano compiti diversi. Il robot blu doveva raggiungere l'obiettivo e gli altri fungevano da ostacoli.

La distribuzione dei compiti è un aspetto importante della robotica. Ad esempio, in un magazzino automatizzato, diversi robot lavorano insieme per immagazzinare, prelevare e trasportare le merci. Tutte le attività sono coordinate in modo da ridurre al minimo i tempi morti inutili, da ridurre al minimo i percorsi di trasporto, da consumare poca energia e da far funzionare il magazzino nel modo più efficiente possibile.

I robot sciame sono un settore speciale della robotica. Si tratta, come i Go-Bot, di macchine semplici che lavorano insieme in un grande gruppo per risolvere un compito. In agricoltura, gli sciami di robot possono ora effettuare la semina del mais, osservare lo sviluppo delle piante e le condizioni

del terreno e, infine, raccogliere il grano. Ogni robot dello sciame è piccolo e progettato in modo semplice, ma lo sciame nel suo insieme può fare grandi cose. Questo principio si applica anche ai sistemi multi-agente: Si tratta di semplici unità software che possono lavorare insieme per risolvere problemi complessi. Il compito dell'informatica è quello di sviluppare algoritmi per il coordinamento e la cooperazione ottimali di sistemi complessivi con più attori, siano essi hardware o software.

Parole chiave e siti web

- Robotica degli sciami: `https://it.wikipedia.org/wiki/Robotica_degli_sciami`
- Robot industriali: `https://it.wikipedia.org/wiki/Robot_industriale`
- Sistema multiagente: `https://it.wikipedia.org/wiki/Sistema_multiagente`

11. Le commissioni di Emma

Emma è a casa . Deve svolgere tre compiti e tornare:

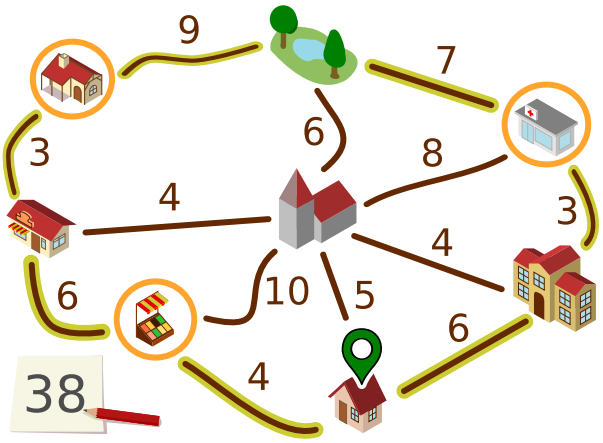
- ritirare un pacco al chiosco
- comprare frutta al mercato e
- Andare in farmacia per prendere una medicina.

Emma non sa quanto tempo impiegherà in ogni negozio. Ma il viaggio dovrebbe essere il più breve possibile.

Emma ha scritto su una mappa quanti minuti dedicherà all'attività di spostamento tra le singole località della città.

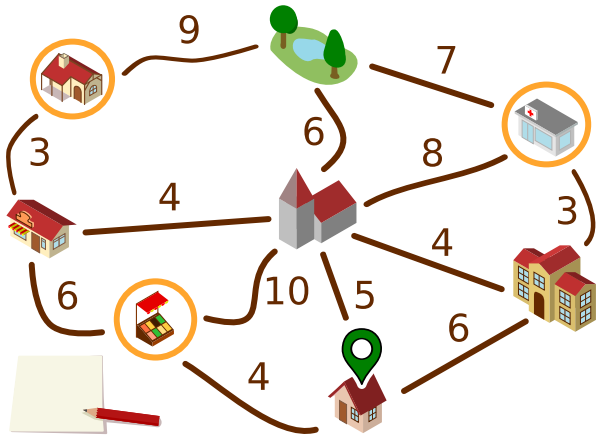
Ha anche segnato sulla planimetria il percorso che sta facendo.

Emma ha bisogno in questo caso di un totale di $6 + 3 + 7 + 9 + 3 + 6 + 4 = 38$ minuti per completare il percorso.



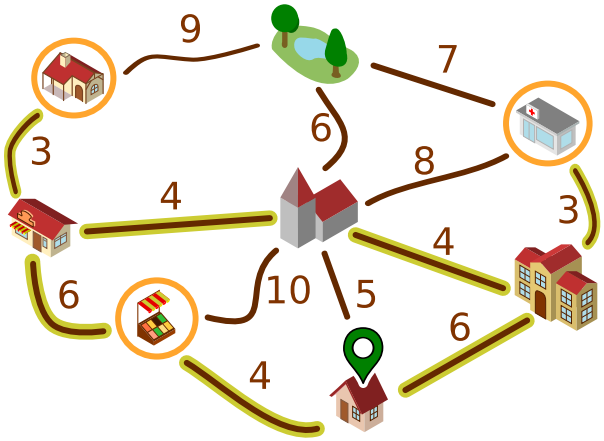
Emma si chiede se si può essere ancora più veloci. Forse è utile percorrere alcune strade più di una volta?

Determina il percorso più breve che Emma può intraprendere per completare i suoi tre compiti.

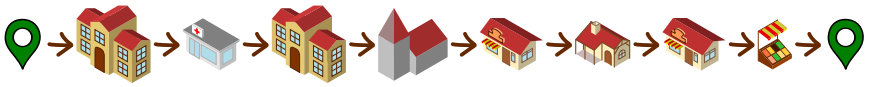


Soluzione

Questa è la soluzione:

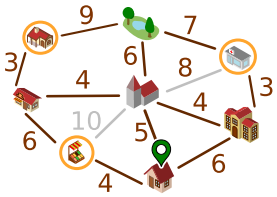


Emma può camminare lungo i percorsi selezionati (o nella direzione opposta):

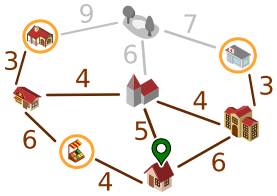


Per percorrere questa distanza ha bisogno di $6 + 3 + 3 + 4 + 4 + 3 + 3 + 6 + 4 = 36$ minuti.

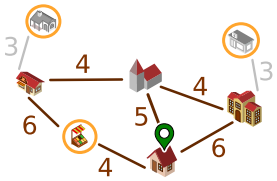
Vogliamo giustificare perché non può esistere un percorso ancora più breve. Per farlo, utilizziamo una rappresentazione semplificata del piano.



Possiamo ignorare i percorsi disegnati in grigio. Esistono percorsi più brevi tra i luoghi collegati dai percorsi, ossia attraverso altri luoghi.

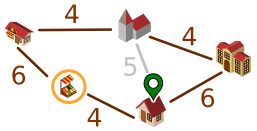


Possiamo anche ignorare il parco, Emma infatti non deve andare al parco. Inoltre, per ogni percorso che passa per il parco, esiste un'alternativa più breve.



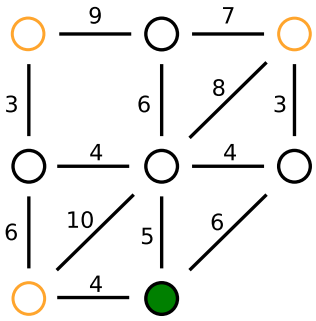
Emma deve andare in farmacia e al chiosco. Può arrivarci solo dal panificio o dalla scuola. Deve percorrere a piedi la distanza tra questi luoghi. Ci vogliono $3 + 3 = 6$, ovvero 12 minuti in totale. Ricordiamolo e combiniamo i due luoghi di cui sopra con quelli di cui sotto in uno solo.

Ora rimane solo il grafico a destra. L'inizio e la fine del percorso sono qui. Questi tre luoghi (, ,) devono essere visitati. Il percorso più breve che soddisfa questo requisito passa attraverso tutti e cinque i luoghi e lungo tutti i percorsi tranne quello grigio e richiede $4 + 6 + 4 + 6 = 24$ minuti. Con i 12 minuti di cui sopra, fanno 36 minuti. Le considerazioni precedenti dimostrano che non può esistere un percorso più breve.



Questa è l'informatica!

Per giustificare la risposta corretta è stata utilizzata una rappresentazione semplificata della mappa. Sarebbe stato possibile presentare la mappa in modo molto più astratto:



Questa rappresentazione contiene tutte le informazioni importanti per il percorso di Emma, ovvero

- Oggetti: i luoghi, con segnati i luoghi importanti per il percorso;
- e relazioni tra gli oggetti: le distanze tra i luoghi per ognuno dei quali è indicata una lunghezza.

Uno strumento importante per modellare le relazioni tra gli oggetti sono i *grafi*. I grafi sono costituiti da nodi (per gli oggetti) e da bordi (coppie di oggetti, per le relazioni). La mappa di Emma può essere modellata come un *grafo pesato*, dove alle singole relazioni vengono assegnati dei valori numerici (i *pesi*).

L'informatica è interessata a domande che possono essere poste in relazione ai grafi, e per gli algoritmi che possono essere utilizzati per rispondere alle domande. Una domanda importante per i grafi pesati è: Qual è il percorso più breve (o più veloce) tra due nodi? La «domanda del grafico» in questo compito è simile: Qual è il viaggio di andata e ritorno più breve da un nodo che visita molti altri nodi? L'informatica conosce molti algoritmi in grado di determinare in modo efficiente i percorsi più brevi nei grafi. Tali algoritmi sono implementati in software per, ad esempio, la pianificazione di itinerari.

Parole chiave e siti web

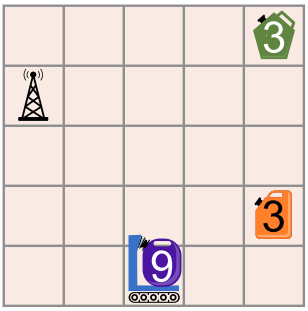
- Grafi: `https://it.wikipedia.org/wiki/Grafo`

12. Missione Zerobot

Lo Zerobot ha un serbatoio sostituibile. Zerobot si muove in una griglia: in alto, in basso, a destra e a sinistra. Ogni volta che si sposta da una casella della griglia, il livello del serbatoio diminuisce di 1 unità.

Su alcune piazze sono presenti serbatoi di ricambio, il cui numero indica il livello di riempimento. Quando Zerobot raggiunge un campo di questo tipo, cambia il suo serbatoio, indipendentemente da quanto sia pieno: prende il serbatoio di scambio, posa il serbatoio precedente sul campo e continua a guidare.

La posizione attuale di Zerobot e il livello del suo serbatoio sono mostrati nell'immagine come segue:



Allarme: i serbatoi sono difettosi e potrebbero esplodere se lasciati con del carburante!

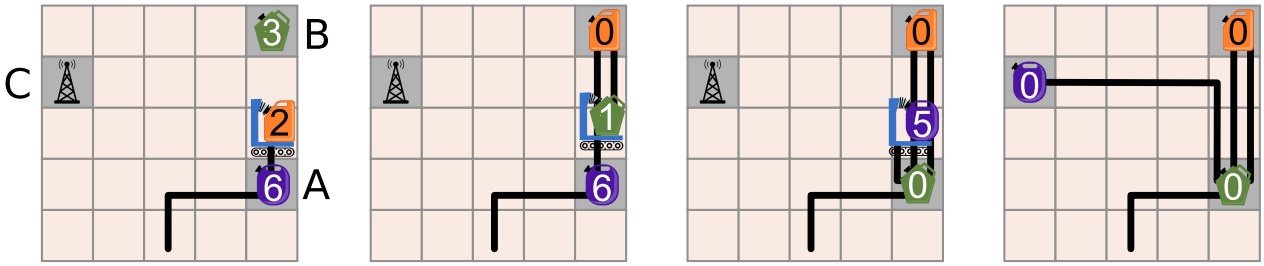
Questa è la missione di Zerobot: deve raggiungere la stazione base in modo tale che tutti i serbatoi siano vuoti alla fine (livello di riempimento 0).

Come deve muoversi Zerobot per compiere la sua missione?

Soluzione

La risposta corretta:

Zerobot può raggiungere la stazione base con 15 movimenti in modo che tutti i serbatoi abbiano un livello di riempimento pari a 0 alla fine:



Per rendere più semplice la spiegazione della risposta corretta, etichettiamo i campi con i serbatoi di scambio e la stazione base con le lettere A, B e C:

Zerobot sposta 3 campi in A e scambia ![tank_violet] (livello di riempimento 6) con  (livello di riempimento 3). Poi si sposta di 3 campi in B e scambia  (livello 0) con  (livello 3). Poi si sposta di nuovo in A e scambia  (livello 0) con ![tank_violet] (livello 6). Quindi viaggia per 6 campi fino alla stazione base C. ![tank_violet] ha un livello di riempimento pari a 0. Missione compiuta!

È questa l'unica soluzione corretta? Lo Zerobot deve compiere esattamente 15 movimenti: 15 movimenti sono il minimo necessario per consumare tutto il carburante disponibile di $9 + 3 + 3 = 15$ unità, e non c'è abbastanza carburante per altri movimenti. Per svuotare tutti i serbatoi, deve visitare entrambi i campi con serbatoi di scambio, e A anche due volte. Se lo Zerobot visitasse prima il campo B, avrebbe bisogno di 17 movimenti per raggiungere la stazione base, il che non è possibile. Pertanto, la sequenza di serbatoi mostrata è l'unica risposta corretta.

Questa è l'informatica!

In questo compito vengono affrontati alcuni problemi fondamentali della mobilità autonoma: ogni robot mobile autonomo (come un'auto a guida autonoma) deve considerare quanta energia è disponibile sotto forma di carburante o di carica della batteria quando pianifica le sue attività. Da un lato, deve assicurarsi di raggiungere in tempo una stazione di ricarica o di rifornimento prima che la sua scorta di energia sia esaurita. D'altra parte, ci sono condizioni quadro da considerare. Nel compito, una condizione quadro è che la fornitura di energia deve essere completamente esaurita entro la fine. In realtà, si tratta soprattutto di altre condizioni quadro, come la posizione e la disponibilità delle stazioni di ricarica. Il software per il controllo dei robot mobili contiene componenti che assicurano energia sufficiente attraverso la ricarica (*gestione intelligente della carica della batteria*).

Inoltre, i programmi informatici vengono utilizzati anche per pianificare e gestire reti efficienti di stazioni di ricarica. Gli informatici stanno cercando soluzioni al problema del posizionamento delle stazioni di ricarica: le stazioni di ricarica per i robot mobili devono essere posizionate in modo tale che un robot con un certo livello minimo di carica possa raggiungere una delle stazioni di ricarica disponibili. Per la comunicazione tra le stazioni di ricarica e le auto a guida autonoma sono stati sviluppati protocolli come l'OCPP (*Open Charge Point Protocol*).

Parole chiave e siti web

- Gestione intelligente della carica della batteria: `https://www.researchgate.net/publication/364734487_Intelligent_Battery_-Recharge_Management_for_Mobile_Robots`
- Open Charge Point Protocoll (OCPP): `https://de.wikipedia.org/wiki/OCPP`

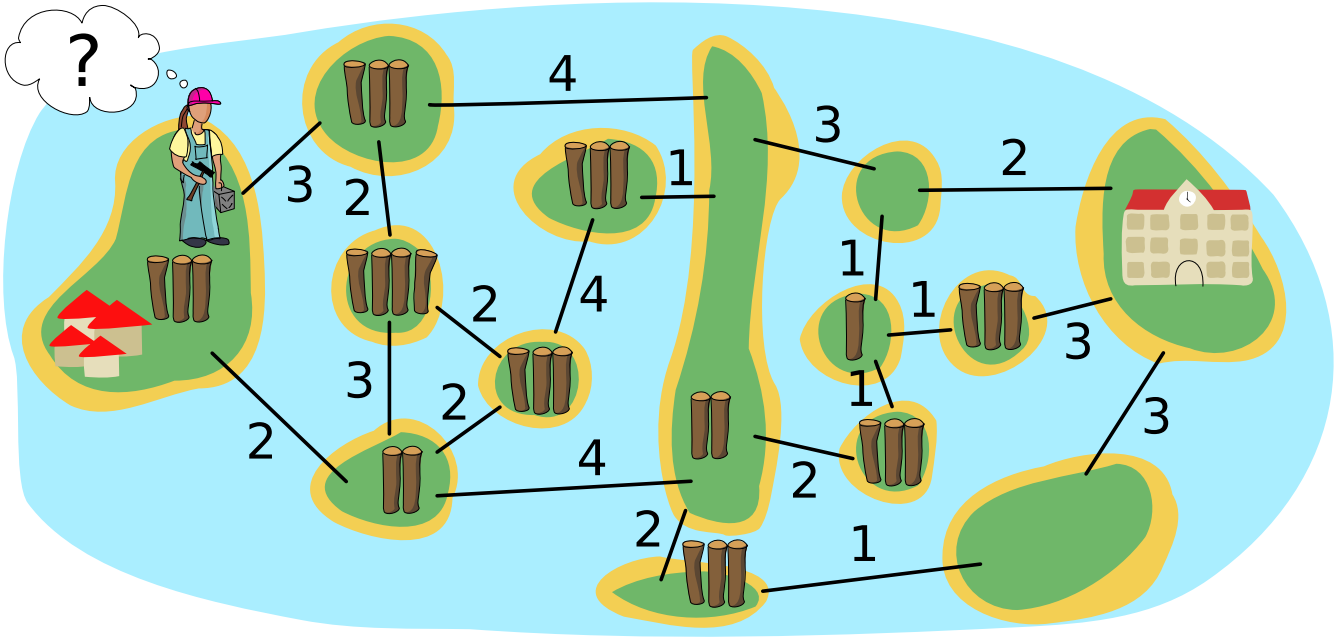
13. Costruiamo ponti!

I bambini si sono trasferiti nell'isola all'estrema sinistra. Bianca deve costruire dei ponti che permettano ai bambini di andare a scuola sull'isola all'estrema destra.

La mappa delle isole mostra quanti tronchi d'albero ci sono su ogni isola. Bianca può prendere questi tronchi d'albero per costruire ponti lungo le linee. Il numero su una linea indica quanti tronchi sono utilizzati in quel punto per un ponte. Quando c'è un ponte tra due isole, Bianca può attraversarlo e portare con sé i tronchi che ha ancora. Naturalmente, può usare ogni tronco d'albero per un solo ponte.

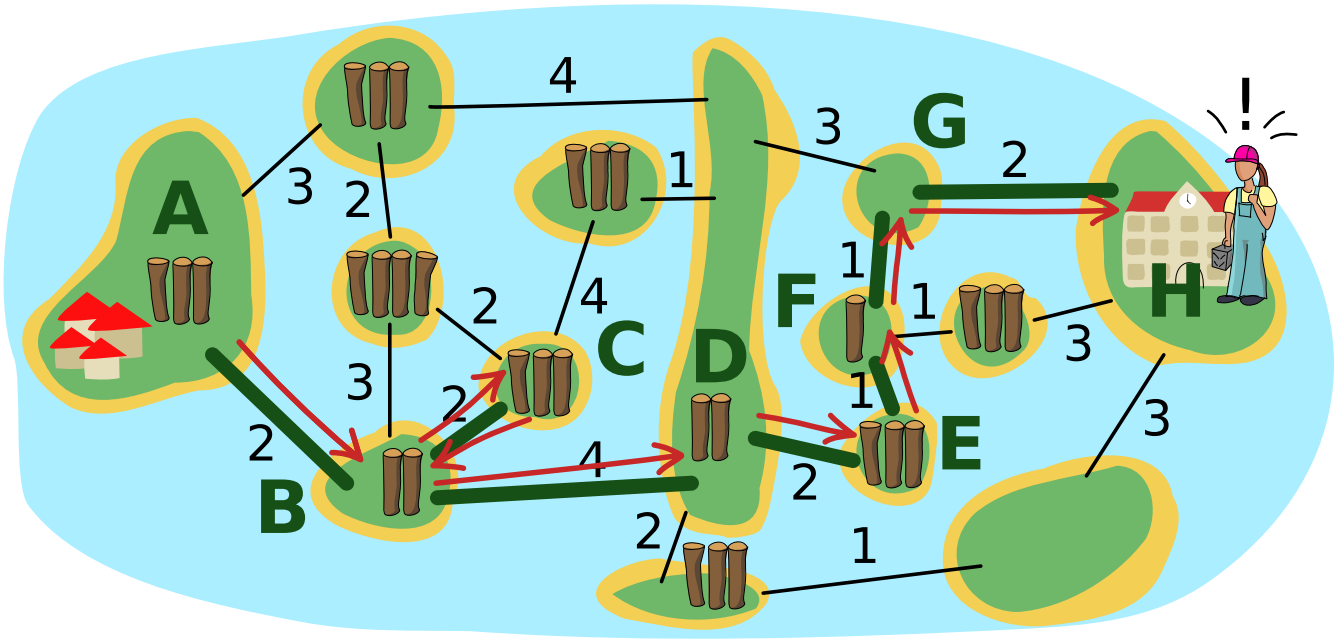
Bianca inizia sull'isola a sinistra. Il suo obiettivo è utilizzare il minor numero possibile di tronchi.

Su quali linee Bianca deve costruire i ponti per raggiungere la sua destinazione?



Soluzione

Questa è la risposta corretta:



Le linee verdi mostrano dove Bianca ha costruito i ponti. Le frecce rosse mostrano come Bianca ha attraversato i ponti:

- Sull'isola A prende i tre tronchi e ne usa due per il primo ponte. Con il tronco rimanente, attraversa il ponte e sull'isola B ha $3 - 2 + 2 = 3$ tronchi. Questo non è sufficiente per costruire un ponte verso l'isola D.
- Pertanto, costruisce un ponte con 2 tronchi fino all'isola C. Attraversa il ponte, prende i 3 tronchi dall'isola C e torna indietro. Ora ha $3 - 2 + 3 = 4$ tronchi.
- Con essi costruisce un ponte verso l'isola D, attraversa il ponte e poi ha i due tronchi dell'isola D.
- In seguito, costruisce un ponte per l'isola E e può prendere 3 tronchi. Costruisce altri ponti per le isole F e G. Così sull'isola E ha 3 tronchi, sull'isola F $3 - 1 + 1 = 3$ tronchi e sull'isola G ancora 2 tronchi.
- Questi sono esattamente sufficienti per costruire un ponte verso l'Isola H, dove si trova la scuola.

In totale, Bianca è riuscita a costruire i ponti per un percorso dall'isola A all'isola H, utilizzando 14 tronchi d'albero. Ma è possibile anche con un numero inferiore di tronchi? A tal fine, è necessario esaminare tutti i possibili percorsi. Poiché tutti conducono attraverso la lunga isola D, il problema può essere diviso in due parti: dall'isola A all'isola D e dall'isola D all'isola H:

- Per i ponti dall'isola A all'isola D, Bianca ha utilizzato 8 tronchi ed è arrivata sull'isola D senza tronchi. Il suo percorso è il seguente: $2 - [2, 2] - 4$ (dall'isola A attraverso la linea con il 2 fino all'isola B, poi tra B e C e ritorno attraverso il 2, poi attraverso il 4 fino all'isola D). Un percorso con meno tronchi sarebbe di $3 - 4$, ma può essere costruito solo con una deviazione

$(3 - [2, 2] - 4)$, quindi consuma 9 tronchi, e Bianca arriva sull'isola D con un tronco in più. Tutti gli altri percorsi dall'isola A a D consumano 9 o più tronchi.

- Per i ponti dall'isola D alla H, Bianca ha utilizzato 6 tronchi. Non può costruire il percorso diretto $3 - 2$, nemmeno con un tronco in più. Tutti gli altri percorsi dall'isola D all'isola H utilizzano 6 tronchi o più.

Non è quindi possibile costruire ponti con meno di 14 tronchi su cui i bambini possano camminare dall'isola A del villaggio all'isola H della scuola. Con i ponti che ha costruito, Bianca ha quindi raggiunto il suo obiettivo.

Questa è l'informatica!

La mappa dell'isola con i «siti di costruzione dei ponti» indicati dalle linee può essere modellata come un *grafo*: si tratta di una struttura matematica che mette in relazione gli oggetti (chiamati anche nodi) tra loro a coppie (le coppie sono chiamate anche bordi). In un grafo, le isole sono modellate come nodi e le linee come bordi. In questo caso, gli spigoli hanno dei *pesi*, cioè il numero di tronchi utilizzati per costruire un ponte lungo una linea, ma anche i nodi (il numero di tronchi su un'isola) - questo è piuttosto insolito. Per i grafi in cui solo i bordi sono ponderati, l'informatica conosce diversi algoritmi efficienti in grado di calcolare il percorso più breve (tramite bordi con somma minima di pesi) tra due nodi.

Il problema che Bianca vuole risolvere in modo ottimale in questo compito è più complicato: anche lei vuole fare il percorso più breve, ma ha un vincolo: la somma dei pesi dei nodi sul suo percorso fino a quel momento (i tronchi che potrebbe utilizzare) meno la somma dei pesi dei bordi sul suo percorso (i tronchi che ha usato per costruire il ponte) deve essere maggiore del peso del bordo che vuole percorrere dopo o dove vuole costruire un ponte. Per trovare il percorso ottimale, potrebbe essere necessario provare tutte le possibilità. La suddivisione del problema in due parti aiuta a ridurre il numero di possibilità. Inoltre, grazie al vincolo, è possibile escludere molte possibilità prima di averle provate tutte. In informatica, una procedura di questo tipo (provare ed tornare indietro per esclusione) è nota come *backtracking*.

Parole chiave e siti web

- Grafo: `https://it.wikipedia.org/wiki/Grafo`
- Backtracking: `https://it.wikipedia.org/wiki/Backtracking`

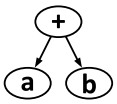
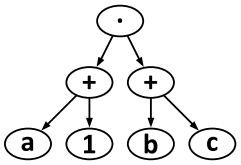
14. Notazione postfissa

Un'espressione matematica è costituita da ...

- ... un *operatore*: +, -, · o :
- ... e gli *operandi*: numeri come 1, 2, ..., lettere come a, b, ... o ancora espressioni come (1 + 2).

La struttura di un'espressione matematica può essere rappresentata come un *albero strutturale*. Questo diagramma di operatori e operandi è disegnato con un cerchio con l'operatore è collegato all'albero degli operandi da frecce. Nel caso più semplice, si tratta di cerchi con un numero o una lettera.

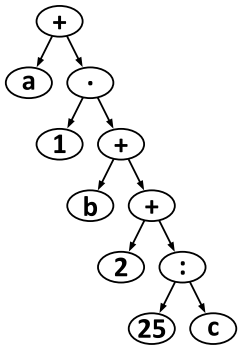
Da un albero, a sua volta, si può leggere la *notazione postfissa* di un'espressione matematica. In questa notazione, per ogni espressione, gli operandi vengono scritti per primi, seguiti dall'operatore.

Espressione matematica:	a + b	(a + 1) · (b + c)
Albero strutturale:		
Notazione postfissa:	a b +	a 1 + b c + ·

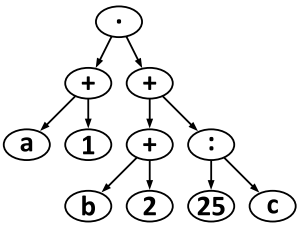
Ecco la notazione postfissa di un'altra espressione:

a 1 + b 2 + · 25 c : +

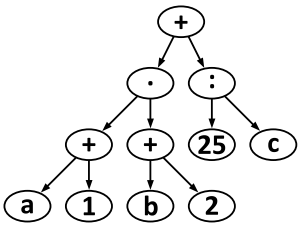
Qual è l'albero strutturale di questa espressione?



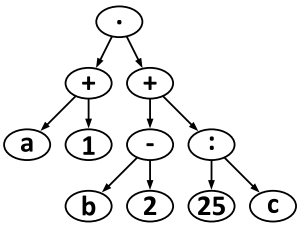
A)



B)

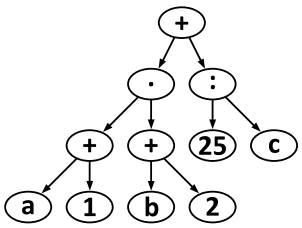


C)



D)

Soluzione



La risposta C è corretta:

Come descritto nel compito, l'operatore centrale di un'espressione matematica si trova in cima all'albero (è la sua *radice*) e alla fine nella notazione postfissa. Se si vuole trovare o creare l'albero di struttura di un'espressione in notazione postfissa, si deve cercare l'ultimo carattere della notazione postfissa in cima all'albero, in questo caso il +. Solo negli alberi delle risposte A e C si trova un + nella radice.

L'operatore + ha due operandi, uno a sinistra e uno a destra. Nella notazione postfissa, si può vedere direttamente (al penultimo carattere) che l'operando di destra dell'espressione è di nuovo un'espressione che ha l'operatore :. Nell'albero strutturale, quindi, deve esserci un : a destra sotto la radice. Questo è il caso solo dell'albero della risposta C. Quindi questa deve essere la risposta corretta.

Questo può essere dimostrato anche convertendo completamente l'albero della risposta C in notazione postfissa:

- I tre sottoalberi più «piccoli», ciascuno composto da 3 elementi, diventano a 1 +, b 2 + e 25 c :.
- I due sottoalberi di sinistra di questi tre «più piccoli» diventano l'operando di sinistra del + superiore, in modo che la trasformazione del sottoalbero di sinistra sia ora 1 + b 2 + ·. Il terzo dei sottoalberi «più piccoli» è già l'operando destro.
- Quindi la notazione postfissa della struttura ad albero della risposta C è complessivamente: a 1 + b 2 + · 25 c : +. Questa è esattamente l'espressione data nel compito.

Questa è l'informatica!

La *notazione postfissa*, detta anche *notazione polacca inversa*, è spesso utilizzata per formulare espressioni matematiche o di altro tipo (ad esempio nei linguaggi di programmazione) senza equivoci e in modo compatto. Se si dovesse scrivere l'espressione data dalla struttura ad albero della risposta C in notazione normale (cioè con gli operatori tra gli operandi, quindi chiamata anche *notazione prefisso*), si dovrebbero mettere le parentesi (a + 1) - (b + 2) + 25 : c, che non sono necessarie nella notazione postfissa. La notazione postfissa è stata introdotta per la prima volta da Jan Łukasiewicz (1878−1956), con l'operatore davanti agli operandi. È il modo in cui si scrive l'applicazione delle funzioni, tra le altre cose: in matematica $f(x, y)$, in programmazione `nomefunzione(argomento1,`

`argomento2, argomento3)`. Nel computer si usa, tra l'altro, quando si *parsano* le espressioni di un linguaggio di programmazione.

Nel recente passato, molte persone hanno conosciuto la notazione postfissa soprattutto con l'uso delle prime calcolatrici scientifiche: con essa si potevano inserire e calcolare espressioni matematiche complesse in modo rapido e affidabile e, soprattutto, senza parentesi. Ancora oggi, esiste una comunità che utilizza calcolatrici programmabili (come le HP 35s) con la notazione postfissa.

Parole chiave e siti web

- Notazione polacca inversa: `https://it.wikipedia.org/wiki/Notazione_polacca_inversa`
- Albero sintttico: `https://it.wikipedia.org/wiki/Albero_sintattico`
- Jan Łukasiewicz: `https://it.wikipedia.org/wiki/Jan_Łukasiewicz`
- HP 35s: `https://it.wikipedia.org/wiki/HP_35s`

15. Serratura a combinazione

Bob ha una serratura a combinazione sulla porta di casa. Per aprirla, è necessario inserire un codice numerico. Tutte le cifre del codice devono essere diverse. Attualmente, il codice è composto da cinque cifre e si legge così:

Bob si è scritto il codice, ma lo maschera un po': $n >> c$ significa che nel codice ci sono esattamente n cifre a sinistra della cifra c che sono maggiori di c. Ad esempio, Bob annota che con

$1 >> 3$

a sinistra della cifra 3 c'è esattamente una cifra (cioè 4) che è maggiore di 3. Ha scritto il codice numerico attuale in questo modo:

$0 >> 0$; $3 >> 1$; $0 >> 2$; $1 >> 3$; $0 >> 4$

Un codice di sole cinque cifre è troppo insicuro per Bob. Pertanto, pensa a un nuovo codice, dalle cifre da 0 a 7. Scrive il nuovo codice in questo modo:

$3 >> 0$; $2 >> 1$; $4 >> 2$; $4 >> 3$; $1 >> 4$; $1 >> 5$; $1 >> 6$; $0 >> 7$

Qual è il nuovo codice?

Soluzione

Ecco la risposta corretta:

Per determinare il codice, analizziamo più da vicino la notazione di Bob, una per una per le cifre da 0 a 7.

- 3 >> 0: ci sono esattamente 3 cifre a sinistra di 0 che sono maggiori di 0. La cifra 0 deve quindi trovarsi nella quarta posizione del codice.
- 2 >> 1: ci sono esattamente 2 cifre a sinistra di 1 che sono superiori a 1. La cifra 1 deve quindi trovarsi nella terza posizione del codice.
- 4 >> 2: ci sono esattamente 4 cifre a sinistra di 2 che sono maggiori di 2. Poiché le cifre più piccole 1 e 0 sono già al terzo e quarto posto, le 4 cifre più grandi devono essere al primo, secondo, quinto e sesto posto. La cifra 2 deve quindi trovarsi nella settima posizione del codice.
- 4 >> 3: ci sono esattamente 4 cifre a sinistra di 3 che sono maggiori di 3. La cifra 3 deve quindi trovarsi nell'ottava e ultima posizione del codice.
- 1 >> 4: a sinistra di 4 c'è esattamente 1 cifra maggiore di 4. La cifra 4 deve quindi trovarsi nella seconda delle cifre rimanenti; si tratta della seconda cifra del codice.
- 1 >> 5: a sinistra di 5 c'è esattamente 1 cifra maggiore di 5. La cifra 5 deve quindi trovarsi nella seconda delle cifre rimaste; si tratta della quinta cifra del codice.
- 1 >> 6: a sinistra di 6 c'è esattamente 1 cifra maggiore di 6. La cifra 6 deve quindi trovarsi nella seconda delle cifre rimaste; si tratta della sesta cifra del codice.
- 0 >> 7: non esiste una cifra superiore a 7. La cifra 7 deve trovarsi nell'ultima posizione libera, cioè nella prima posizione del codice.

Questa è l'informatica!

Bob descrive nella sua notazione come il codice si riferisce a una sequenza ordinata di cifre o numeri utilizzati.

Vediamo di nuovo il codice a cinque cifre: 0 2 4 3 1. Si crea prendendo i numeri ordinati 0 1 2 3 4 e cambiandone la posizione. Il risultato è chiamato anche *permutazione* (dei numeri da 0 a 4). In una permutazione, i numeri possono essere «stravolti» per quanto riguarda il loro ordinamento. Ad esempio, nel codice, 4 precede 3, mentre nella sequenza ordinata 3 precede 4 (perché $3 < 4$). Quindi il 3 è «sbagliato» rispetto all'ordinamento. In combinatoria, una branca della matematica, questo si chiama *inversione* o *errore*.

Il codice di Bob è quindi una permutazione, e la sua notazione indica per ogni numero quante volte è «invertito» in esso: Lo 0 è corretto, l'1 fa parte di 3 inversioni (3 >> 1: tre numeri più grandi sono davanti all'1), il 2 è corretto, il 3 è invertito una volta, il 4 è corretto. La sequenza di questi numeri

di inversione è chiamata *sequenza di inversione*. (A proposito, la somma dei numeri di inversione descrive il grado di non ordinabilità di una permutazione).

Ora abbiamo tre sequenze - il codice (o permutazione), la sequenza ordinata e la sequenza di inversione - e le riassumiamo in questa tabella:

Codice / Permutazione	0	2	4	3	1
Sequenza ordinata	0	1	2	3	4
Sequenza di inversione	0	3	0	1	0

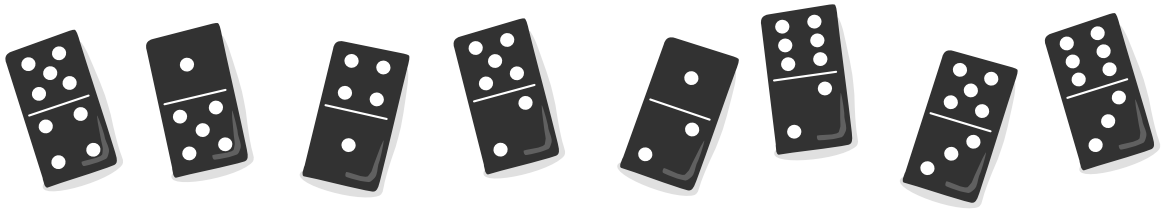
La descrizione della soluzione ha dimostrato che esiste un algoritmo efficiente che calcola la permutazione corrispondente dalla sequenza di inversione. È sufficiente ripercorrere una volta la sequenza di inversione. L'informatica si occupa spesso di problemi combinatori e conosce molti algoritmi per risolverli. Possono essere utilizzati per la soluzione automatica di rompicapo (come i Sudoku), ma anche per molti problemi «seri». La maggior parte delle volte sono molto più complicati dell'algoritmo per la soluzione di questo compito del castoro.

Parole chiave e siti web

- Permutazione: `https://it.wikipedia.org/wiki/Permutazione`

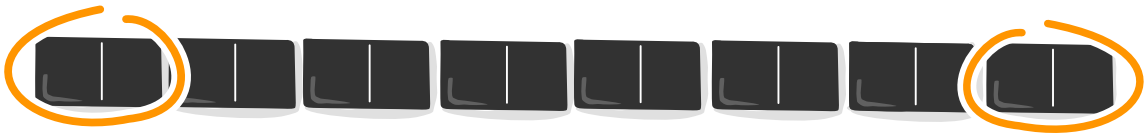
16. Domino

Ogni domino ha due caselle. Su ogni casella ci sono da 1 a 6 punti. Hai questi otto domino:



Posiziona tutti e otto i domino in fila in modo che ci sia sempre lo stesso numero di punti sulle caselle adiacenti di due domino vicini.

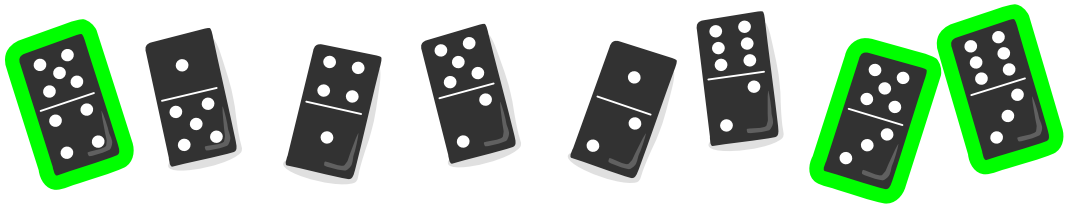
È possibile disporre varie file di questo tipo. Tuttavia, ci sono domino che non possono essere posizionati all'inizio o alla fine della fila.



Quali sono questi domino?

Soluzione

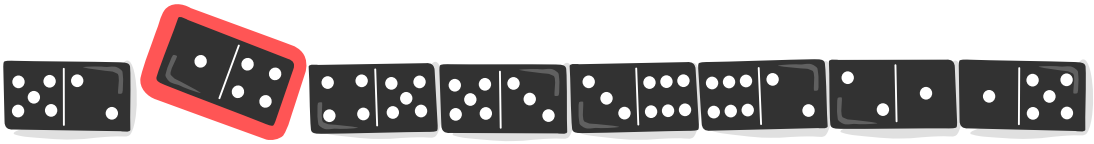
Tre delle otto pietre non possono essere posizionate all'inizio o alla fine della fila:



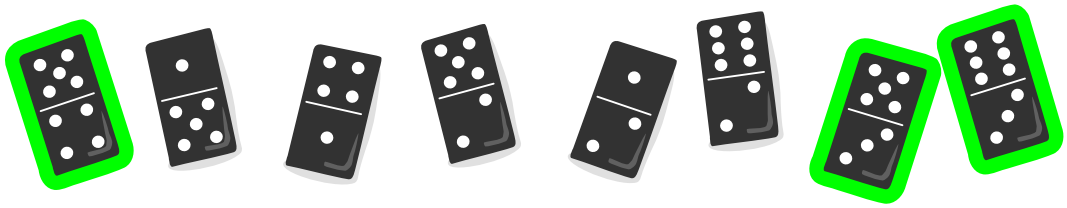
Per risolvere il compito, si esamina il numero di occhi (i punti sul domino sono chiamati anche *occhi*, come su un dado) delle 16 caselle. Registriamo la frequenza dei singoli numeri di occhi e se la frequenza è pari o dispari:

Numero di occhi	Frequenza	Pari/dispari
	3	dispari
	3	dispari
	2	pari
	2	pari
	4	pari
	2	pari

Le caselle con un numero di punti pari devono trovarsi a coppie al centro della riga o contemporaneamente all'inizio e alla fine. Tuttavia, le caselle con un numero dispari di punti non possono trovarsi tutti al centro della fila: non è possibile trovare un quadrato adiacente corrispondente per ogni quadrato con questo numero di punti; questo è possibile solo con frequenze pari. Qui si vede una riga in cui una pietra con un numero 1 (che esiste tre volte) non si adatta più al centro.

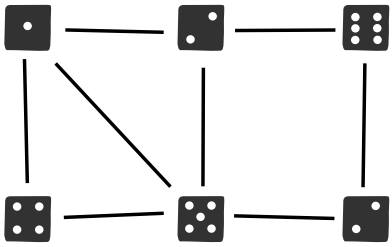


Poiché sulle otto pietre di questo compito ci sono caselle con frequenze dispari, le pietre con tale caselle devono essere posizionate all'esterno. Le pietre che hanno due caselle con frequenza pari non possono essere posizionate all'inizio o alla fine della fila. Si tratta delle seguenti pietre:

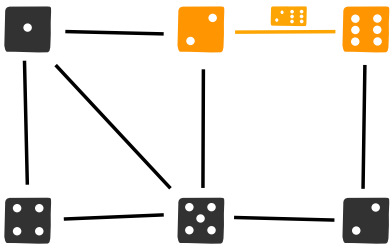


Questa è l'informatica!

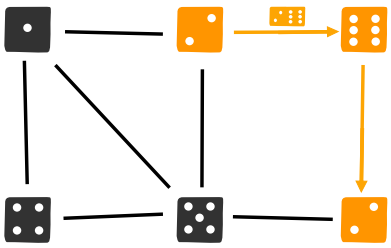
Esistono diversi modi per posizionare le otto pietre di domino di questo compito in una fila corretta. Per avere una visione d'insieme, gli informatici utilizzano i cosiddetti *grafi*:



Nel grafo qui sopra sono visibili le caselle (i cosiddetti *nodi*), che mostrano i sei numeri del domino. Le otto linee (chiamate *bordi*) rappresentano gli otto domino; ogni linea collega due caselle. Ad esempio, il domino 2−6 è rappresentato dal seguente bordo:



Per risolvere il compito, tutti e otto i domino devono essere posizionati in una fila corrispondente. Dopo aver posizionato il primo domino, è già chiaro con quale numero di punti deve iniziare il secondo domino, perché i campi adiacenti di due domino devono sempre avere lo stesso numero di punti. Nel grafo, questo si può vedere dal fatto che i domino possono essere posizionati l'uno accanto all'altro esattamente quando i loro bordi si incontrano nello stesso nodo. Per esempio, le pietre 2−6 e 6−3 possono essere accostate perché entrambe contengono il numero 6:



L'allineamento delle pietro può essere inteso come un *percorso* (una sequenza di bordi) attraverso il grafo. Questo percorso deve visitare tutti gli spigoli *esattamente una volta* per garantire che le otto pietre del domino vengano utilizzate tutte, ma anche che non vengano utilizzate più di una volta. Un percorso che visita *ogni bordo esattamente una volta* è chiamato *cammino euleriano*. Il nome risale a Leonhard Euler, matematico svizzero e inventore della teoria dei grafi. Euler è riuscito a dimostrare che in un grafo connesso, un cammino di Euler, esiste esattamente quando al massimo due nodi hanno un numero dispari di bordi che partono da quel nodo.

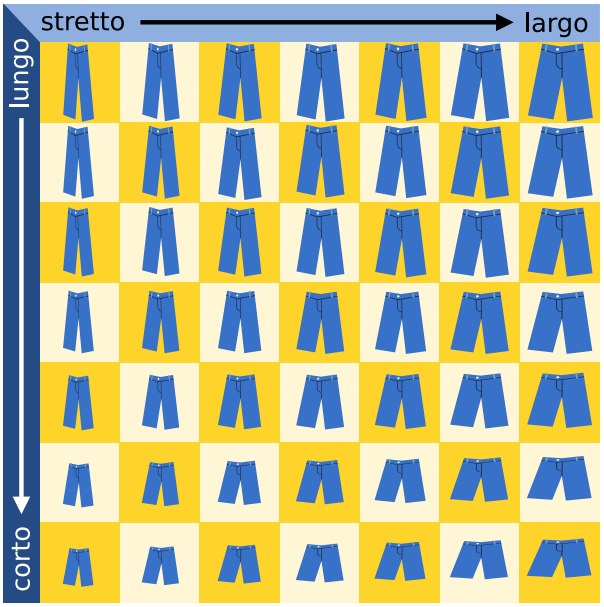
Parole chiave e siti web

- Grafo: `https://it.wikipedia.org/wiki/Teoria_dei_grafi`
- Nodi: `https://it.wikipedia.org/wiki/Vertice_(teoria_dei_grafi)`
- Cammino euleriano: `https://it.wikipedia.org/wiki/Cammino_euleriano`

17. Pantaloni adatti

Christian ha bisogno di nuovi pantaloni. Nel negozio, i suoi pantaloni preferiti sono disponibili in sette lunghezze e sette larghezze. I pantaloni di tutte le 49 taglie sono sullo scaffale, ordinati per lunghezza e larghezza.

Poiché Christian non conosce la sua taglia, deve scoprirla provandola. A ogni prova, Christian nota se i pantaloni gli vanno bene o se ha bisogno di pantaloni più corti, più lunghi, più stretti o più larghi. Affinché un paio di pantaloni sia adatto, la lunghezza e la larghezza devono essere giuste.



La commessa si lamenta: «Trovare la taglia giusta in 49 taglie può richiedere molto tempo.»

Ma Christian ha ideato un metodo per trovare la taglia giusta con il minor numero possibile di prove.

Di quante prove ha bisogno al massimo per trovare la taglia giusta con il metodo ideato da Christian?

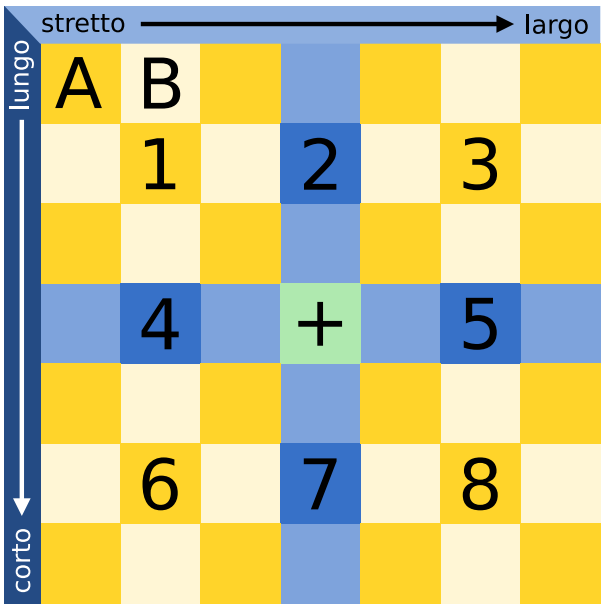
Soluzione

La risposta giusta è 2.

Christian può essere fortunato e ottenere i pantaloni della taglia giusta alla prima prova. Ma non può affidarsi alla fortuna e dunque procede secondo questo metodo:

Prima prova i pantaloni al centro (posizione + nell'immagine). Durante la prova, controlla la lunghezza e la larghezza dei pantaloni.

- Se la lunghezza e la larghezza sono giuste, ha trovato i pantaloni della taglia giusta.
- Se i pantaloni sono troppo corti e troppo larghi, i pantaloni corrispondenti si trovano nell'area 1.
- Se i pantaloni sono troppo corti ma hanno la larghezza giusta, i pantaloni corrispondenti si trovano nell'area 2.
- Se i pantaloni sono troppo corti e troppo stretti, i pantaloni abbinati si trovano nell'area 3.
- Ripete poi il metodo per le aree da 4 a 8.



Supponiamo che i pantaloni della taglia giusta siano nell'area 1. Christian sceglie i pantaloni al centro dell'area 1 per la seconda prova.

Ora ci sono di nuovo diverse possibilità:

- Se i pantaloni vanno bene, ha trovato la taglia giusta.
- Se i pantaloni sono ancora troppo corti e troppo larghi, Christian sa che i pantaloni nella posizione A sono della taglia giusta.
- Se i pantaloni sono troppo corti ma hanno la larghezza giusta, Christian sa che i pantaloni della posizione B sono della taglia giusta.
- Si può continuare così per le altre posizioni intorno alla metà dell'area 1.

Poiché in ogni area numerata il ripiano centrale ha solo un ripiano vicino in ogni direzione, non sono necessari altri controlli. Quindi Christian ha bisogno al massimo di due prove in ogni caso per trovare la taglia giusta.

Questa è l'informatica!

Il metodo utilizzato da Christian per l'adattamento è chiamato ricerca binaria in informatica. Il termine *binario* deriva dalla parola latina bis (due volte). In una ricerca binaria di un oggetto in una sequenza di oggetti ordinati, l'oggetto centrale viene confrontato con quello cercato. Se l'oggetto centrale non è già quello che si sta cercando, si sa almeno in quale metà della sequenza si trova l'oggetto che si sta cercando e si cerca nuovamente in questa metà in modo binario. In ogni fase, la sequenza viene divisa in due parti - da qui «binaria». In questo modo si arriva molto rapidamente all'oggetto cercato. Per 1.000 oggetti sono necessari circa 10 passi di ricerca, per 1.000.000 di oggetti circa 20. In generale, possiamo dire: per n oggetti sono necessari circa $\log(n)$ passi; la funzione log è il «logaritmo di due» o il logaritmo in base 2. Poiché la ricerca binaria è così veloce, viene spesso utilizzata nei programmi informatici per la ricerca di dati ordinati.

In questo compito, lo spazio di ricerca, cioè i pantaloni sullo scaffale, è ordinato in due dimensioni (lunghezza e larghezza). Pertanto, Christian può applicare immediatamente la ricerca binaria in entrambe le dimensioni. Poi l'insieme di ricerca viene diviso in un unico passaggio non in 2, ma in 8 parti, nel caso in cui Christian non abbia ottenuto direttamente la dimensione giusta.

Parole chiave e siti web

- Ricerca binaria: `https://it.wikipedia.org/wiki/Ricerca_dicotomica`
- Alrogitmo di ricerca: `https://it.wikipedia.org/wiki/Algoritmo_di_ricerca`

18. Rilevatore di conflitti

Anna e Ben vogliono costruire un «rilevatore di conflitti» che mostri se hanno un'opinione diversa.

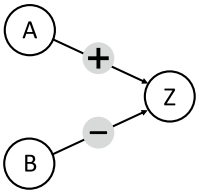
Decidono di utilizzare delle unità che possono essere in due stati, Sì e No: due unità possono essere collegate tramite un cavo che può trasmettere un segnale.

I cavi sono impostati per trasmettere un segnale positivo (+) o negativo (-) all'unità collegata alla sua destra. Quando un'unità si trova nello stato:

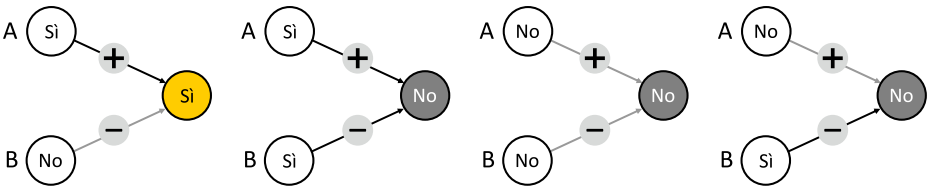
- Sì: trasmette un segnale attraverso tutti i cavi in uscita.
- No: non trasmette alcun segnale.

Un'unità collegata passa allo stato Si se riceve più segnali positivi che negativi, e allo stato No in caso contrario o se il numero di segnali positivi e negativi è lo stesso. Anna imposta lo stato dell'unità A e Ben imposta lo stato dell'unità B.

Prima Anna e Ben costruiscono questa macchina:

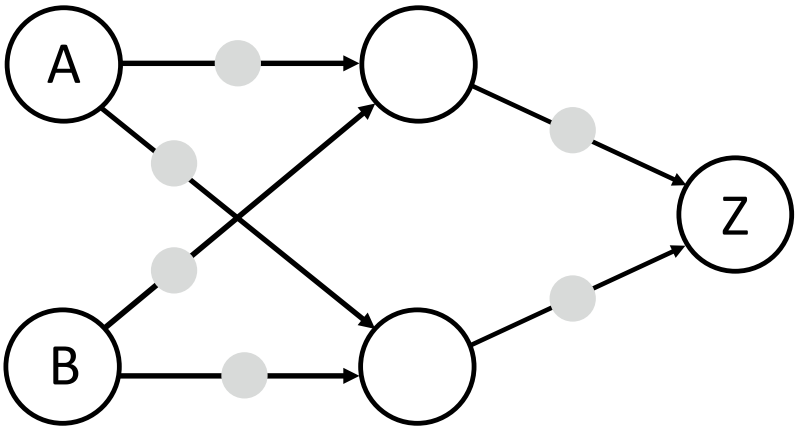


Notano che l'unità Z è Sì solo se A è sì e B è no. Questo non è ciò che vogliono: vorrebbero infatti che l'unità Z fosse Sì solo se A è sì e B e no, ma anche quando A è no e B è sì.



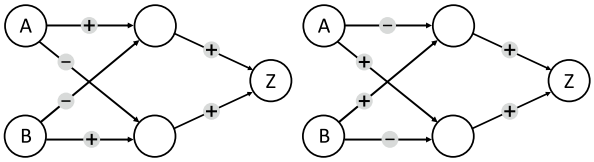
Allora Anna e Ben costruiscono una macchina più grande (in basso nell'immagine) e sono sicuri che possa essere il rilevatore di conflitti corretto: che Z sia Sì solo quando A e B sono in stati diversi (Sì e No o No e Sì). Altrimenti, Z dovrebbe essere nello stato No. Ora non resta che impostare correttamente i cavi.

Imposta per ciascun cavo la trasmissione di un segnale positivo (+) o negativo (-), in modo che il rilevatore di conflitti funzioni correttamente.

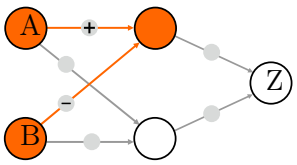


Soluzione

Queste due risposte sono corrette:

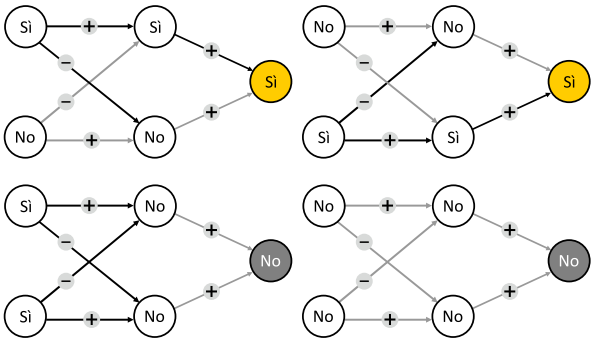


Nel rilevatore di conflitti, l'unità di uscita deve essere Sì esattamente per due ingressi diversi (A=Sì e B=No oppure A=No e B=Sì). Z può essere Sì solo se attraverso i due cavi in ingresso arrivano più segnali positivi che negativi. Almeno uno dei cavi deve quindi trasmettere un segnale positivo (+). Supponiamo che solo il cavo superiore che porta a Z sia impostato su +. Allora l'unità centrale superiore deve essere in grado di riconoscere entrambe le combinazioni di ingresso desiderate, cioè deve essere Sì in entrambi i casi. Insieme alle unità di ingresso A e B, tuttavia, questa unità forma esattamente la macchina che Anna e Ben hanno costruito all'inizio. Può essere Sì solo in uno dei casi desiderati, cioè quando uno dei suoi cavi è impostato su + e l'altro su -:



Quindi, per ciascuno dei casi di ingresso desiderati è necessaria un'unità separata al centro, una per A=Sì e B=No, l'altra per A=No e B=Sì. I cavi alla prima unità devono essere impostati su + (cavo da A) e – (B), i cavi all'altra unità su – (A) e + (B). Non è importante quale unità al centro scelga quale caso; pertanto, ci sono due possibilità per i cavi da A e B al centro. Ora, se ogni unità al centro è Sì esattamente in un caso desiderato, entrambi i cavi dal centro in Z devono essere impostati su +; solo allora Z=Sì esattamente in due casi desiderati.

Per la prima risposta corretta, l'immagine sottostante mostra la funzione del rilevatore di conflitti. Si può vedere che l'unità superiore al centro rileva il caso A=Sì e B=No, quella inferiore il caso A=No e B=Sì. La rispettiva unità trasmette un segnale positivo a Z, e Z è Sì. Per gli altri ingressi (A=Sì e B=Sì così come A=No e B=No) entrambe le unità centrali sono No, Z non riceve alcun segnale positivo ed è quindi No.



Questa è l'informatica!

Il rilevatore di conflitti elabora due valori di ingresso (Sì o No) e restituisce l'uscita Sì esattamente quando i due valori di ingresso sono diversi. Questa funzione logica si chiama «OR esclusivo» (XOR, disgiunzione esclusiva). La prima macchina descritta in questo compito da Anna e Ben (due interruttori e un'unità di uscita) è una versione semplificata di un *percettrone* descritto da Frank Rosenblatt nel 1957. L'unità di uscita riproduce una cellula nervosa (neurone) in grado di elaborare i segnali di ingresso e produrre un segnale di uscita. Con un percettrone è possibile implementare le operazioni logiche AND e OR, ma non l'OR esclusivo. Per questo è necessario un altro strato di unità di commutazione, come nella soluzione di questo compito. Solo negli anni '80 è stato riconosciuto questo aspetto (ad esempio Rumelhart, Hinton & Williams, 1986) e in seguito è stato possibile programmare reti neurali artificiali che funzionano in modo simile al cervello umano e possono, ad esempio, valutare le immagini delle telecamere e riconoscere gli oggetti. L'informatica ha sviluppato metodi per capire come reti neurali di grandi dimensioni con molti strati e unità possano eseguire i loro calcoli in modo efficiente. Tali reti costituiscono la base di molti sistemi di IA (Intelligenza Artificiale) attuali.

Parole chiave e siti web

- Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. Nature, 323(6088), 533−536: `http://www.cs.toronto.edu/~hinton/absps/naturebp.pdf`
- Percettrone: `https://it.wikipedia.org/wiki/Percettrone`
- Disgiunzione esclusiva: `https://it.wikipedia.org/wiki/Disgiunzione_esclusiva`
- Intelligenza artificiale: `https://it.wikipedia.org/wiki/Intelligenza_artificiale`

A. Autori dei quesiti

Eslam AbdElAal

Akram Ahmed

Somayah Albaradei

Laila Alharthi

Khairul Anwar

James Atlas

Leonardo Barichello

Wilfried Baumann

Tim Bell

Leonardo Cavalcante

Diego César

Zaheer Chothia

Eimear Colreavy

Raluca Constantinescu

Kris Coolsaet

Christian Datzko

Justina Dauksaite

Nora A. Escherle

Georgios Fesakis

Gerald Futschek

Emily Gates

Christian Giang

Adam Grodeck

Juan Gutiérrez

Tracy Henderson

Josefine Hiebler

Mathias Hiron

Alisher Ikramov

Thomas Ioannou

Filiz Kalelioğlu

Merel Kämper

Gohar Khachatryan

Jihye Kim

Mhairi King

Jia-Ling Koh

Sophie Koh

Víctor Koleszar

Taina Lehtimäki

Marielle Léonard

Carlos Luna

Michael Weigend

Yong Mao

Madhavan Mukund

Natalia Natalia

Tom Naughton

Zsuzsa Pluhár

Wolfgang Pohl

Estela Ramić

Omar Colon Reyes

Chris Roffey

Karima Sayeh

Kirsten Schlüter

 Margareta Schlüter

 Eljakim Schrijvers

 Giovanni Serafini

 Vipul Shah

 Rostyslav Shpakovych

 Jacqueline Staub

 Alieke Stijf

 Marianne Thut

 Monika Tomcsányiová

 Ahto Truu

 Laura Ungureanu

 Jiří Vaníček

 Michael Weigend

 Manuel Wettstein

 Kyra Willekes

B. Partner accademici

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

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

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

C. Sponsoring

http://www.haslerstiftung.ch/

Standortförderung beim Amt für Wirtschaft und Arbeit Kanton Zürich

http://www.ubs.com/

http://www.verkehrshaus.ch/
Musée des transports, Lucerne

i-factory (Musée des transports, Lucerne)

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

D. Ulteriori offerte

La Fiamma IT: `https://it-feuer.ch/it/`
In Svizzera, numerose organizzazioni si impegnano per la formazione delle giovani leve nell'ambito dell'informatica. L'iniziativa «La Fiamma IT» vuole unire queste forze e contribuire insieme a diffondere il tema nell'opinione pubblica in tutta la Svizzera. La fiamma IT presenta numerose offerte rivolte sia ai docenti che agli studenti.

CoetryLab: `https://www.coetry-lab.org/`
Il team del CoetryLab (Zürich) vuole dare ai bambini e ai giovani l'accesso alla programmazione e ai media. Il CoetryLab vuole essere il luogo di sperimentazione e progettazione extrascolastica e aprire il mondo del coding a tutti. Le loro idee possono essere realizzate in modo creativo e siti web, applicazioni, giochi e molto altro possono essere sviluppati in team o da soli.

Roteco: `https://www.roteco.ch/it/`
Il progetto Roteco consiste in una comunità di insegnanti desiderosi di preparare gli allievi per la società digitale. In questa comunità gli insegnanti trovano, sviluppano e si scambiano attività didattiche inerenti la robotica educativa e più in generale le scienze informatiche pronte da essere utilizzate in classe e vengono informati con le ultime novità e corsi in questi campi.

Diventate membri della SSII `http://svia-ssie-ssii.ch/verein/mitgliedschaft/` sostenendo in questo modo il Castoro Informatico.
Chi insegna presso una scuola dell'obbligo, media superiore, professionale o universitaria in Svizzera può diventare membro ordinario della SSII.
Scuole, associazioni o altre organizzazioni possono essere ammesse come membro collettivo.