INFORMATIK-BIBER SCHWEIZ
CASTOR INFORMATIQUE SUISSE
CASTORO INFORMATICO SVIZZERA

Quesiti e soluzioni 2020

Tutte le Categorie

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

A cura di:

Lucio Negrini, Christian Giang, Susanne Datzko, Fabian Frei,
Juraj Hromkovič, Regula Lacher, Jean-Philippe Pellet

01010011010101100100100
10000001001011010101001 1
01010011010010010100010 1
00101101010100110101001 1
01001001010010010010000 1

SSII

www.svia-ssie-ssii.ch
schweizerischerverein für informatikind
erausbildung//société suisse pour l'infor
matiquedans l'enseignement//società sviz
zera per l'informatica nell'insegnamento

Hanno collaborato al Castoro Informatico 2020

Susanne Datzko, Fabian Frei, Martin Guggisberg, Lucio Negrini, Gabriel Parriaux, Jean-Philippe Pellet

Capo progetto: Nora A. Escherle

Un particolare ringraziamento per il lavoro sui quesiti del concorso Svizzero va a:
Juraj Hromkovič, Michael Barot, Christian Datzko, Jens Gallenbacher, Dennis Komm, Regula Lacher, Peter Rossmanith: ETH Zürich, Ausbildunges- und Beratungszentrum für Informatikunterricht

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: Bebras.org
Wolfgang Pohl, Hannes Endreß, Ulrich Kiesmüller, Kirsten Schlüter, Michael Weigend: Bundesweite Informatikwettbewerbe (BWINF), Germania
Wilfried Baumann, Anoki Eischer: Österreichische Computer Gesellschaft
Gerald Futschek, Florentina Voboril: Technische Universität Wien
Zsuzsa Pluhár: ELTE Informatikai Kar, Ungheria
Michal Winzcer: Comenius University, Slovacchia

La versione online del concorso è stata creata su cuttle.org. Ringraziamo per la buona collaborazione:
Eljakim Schrijvers, Justina Dauksaite, Arne Heijenga, Dave Oostendorp, Andrea Schrijvers, Alieke Stijf, Kyra Willekes: cuttle.org, Olanda
Chris Roffey: University of Oxford, Regno Unito

Per il supporto durante le settimane del concorso ringraziamo:
Hanspeter Erni: Direttore scuola media di Rickenbach
Gabriel Thullen: Collège des Colombières
Beat Trachsler: Scuola cantonale di Kreuzlingen

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

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 2020 è stato organizzato dalla Società Svizzera per l'Informatica nell'Insegnamento SSII con il sostegno della fondazione Hasler.

HASLERSTIFTUNG

Questo quaderno è stato creato il 9 settembre 2021 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'01.12.2020.

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 nell'ambito del programma di promozione «FIT in IT».

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 2020 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

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 ha iniziato 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.

Per ulteriori informazioni:

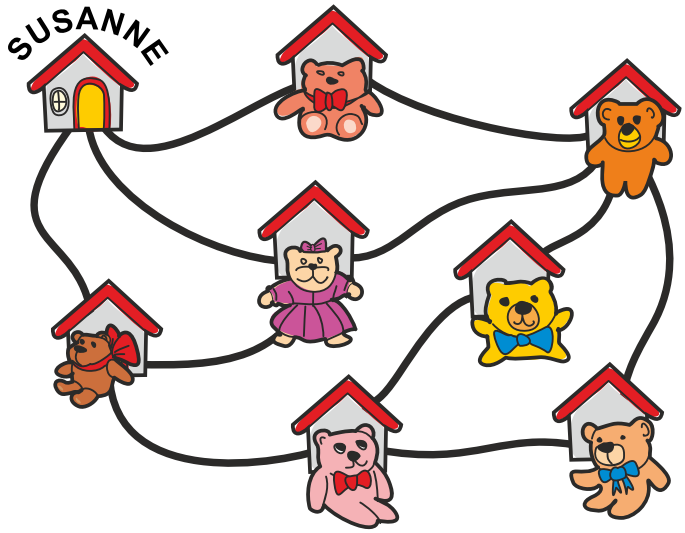
SVIA-SSIE-SSII Società Svizzera per l'Informatica nell'Insegnamento
Castoro Informatico
Lucio Negrini

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

Indice

1. Caccia agli orsacchiotti

Nel quartiere di Susanne si trovano i seguenti orsacchiotti di peluche davanti alle case.



Da casa sua, Susanne ha fatto una passeggiata passando esattamente davanti ad altre quattro case. Non è mai passata due volte su un percorso che collega una casa ad un'altra. In una casa le è sfuggito l'orsacchiotto. Gli altri tre orsacchiotti che ha visto erano:

Quale orsacchiotto è sfuggito a Susanne?

A) 

B) 

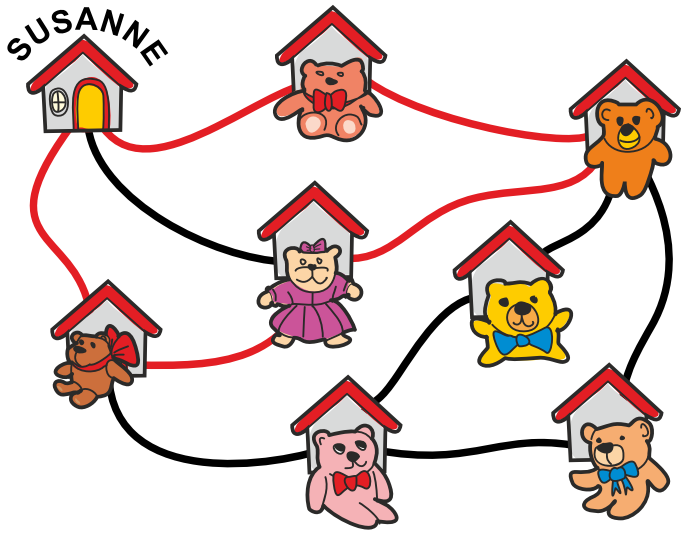
C) 

D) 

Soluzione

La risposta corretta è C).

Durante il suo giro Susanne deve essere passata davanti alle case con i tre orsacchiotti , e Questi tre orsacchiotti sono direttamente collegati da un sentiero. Al primo orsacchiotto Susanne ci arriva direttamente da casa sua. Alla fine di questo percorso si trova il terzo orsacchiotto . Da lì c'è solo una strada per tornare a casa sua, che è la strada per l'orsacchiotto . Altri modi possibili passano almeno da altri due orsacchiotti. Ma Susanne è passata davanti a solo quattro case. La seguente mappa mostra la strada fatta da Susanne:



Susanne può fare il giro in entrambe le direzioni possibili.

Questa è l'informatica!

Cloze in lezioni di lingua, problemi di matematica con campi vuoti o una caccia all'orsacchiotto con un'immagine mancante: sono tutti compiti in cui si cercano informazioni mancanti. Tuttavia, i compiti sono impostati (o *strutturati*) in modo tale che queste informazioni mancanti possano essere trovate con il *pensiero logico* o il *ragionamento*.

Nell'informatica, queste situazioni si trovano spesso. Possono per esempio presentarsi durante il trasferimento dei dati o durante il salvataggio dei dati. Pertanto, le procedure vengono utilizzate per *rilevare gli errori* o anche per *correggerli*. Se l'errore non è troppo grande, questo si ottiene salvando deliberatamente più informazioni di quelle effettivamente necessarie. In questo compito si tratta della mappa e del fatto che Susanne ha superato esattamente quattro orsacchiotti. Così si possono trovare le informazioni mancanti, cioè quale orsacchiotto le è sfuggito.

A proposito, in alcuni paesi del mondo nel 2020 sono state offerte delle «cacce all'orsacchiotto» (in inglese «teddy bear hunts»), dove gli orsacchiotti erano nascosti in diverse case, e potevano essere visti dall'esterno. Questo ha permesso ai bambini di divertirsi a nascondere e cercare insieme gli

orsacchiotti, nonostante le regole della distanza dovute al Coronavirus. L'idea di rievocare una caccia all'orso è nata dal libro illustrato «We're Going on a Bear Hunt» di Michael Rosen (1989).

Parole chiave e siti web

- Rilevazione e correzione d'errore:
  `https://it.wikipedia.org/wiki/Rilevazione_e_correzione_d'errore`
- `https://www.insider.com/coronavirus-pandemic-sparked-worldwide-bear-hunt-to-entertain-kids-2020-4`
- `https://www.youtube.com/watch?v=0gyI6ykDwds`
- `https://www.tio.ch/rubriche/ti-mamme/1431131/bambini-caccia-mondo-idea-ti-mamme`

2. Spettacolo teatrale

Uno spettacolo teatrale presenta una saggia principessa, un nobile cavaliere, un re bello e un drago malvagio. All'inizio il palco è vuoto. Durante lo spettacolo queste quattro figure entrano ed escono dal palco nel seguente ordine:

Primo atto		**Atto secondo**
Il re entra in scena		Il drago entra in scena
La principessa entra in scena		Il cavaliere entra in scena
Il re esce di scena	PAUSA	Il drago esce di scena
Il drago entra in scena		La principessa entra in scena
La principessa esce di scena		Il cavaliere esce di scena
Il drago esce di scena		La principessa esce di scena
Pausa		**Fine**

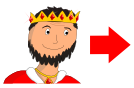
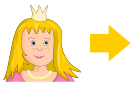
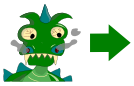
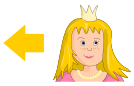
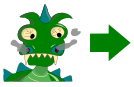
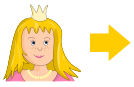
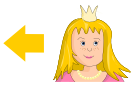
Cosa non succederà?

A) La principessa e il cavaliere sono sul palco insieme.
B) Il re e il drago sono sul palco insieme.
C) Il cavaliere entra in scena dopo la pausa.
D) Il cavaliere e il drago sono sul palco insieme.

Soluzione

La risposta corretta è B) «Il re e il drago sono sul palco insieme», perché questa affermazione non è mai vera durante tutto lo spettacolo.

Si può analizzare passo per passo:

Azione	Re sul palco?	Principessa sul palco?	Drago sul palco?	Cavaliere sul palco?	Risposte corrispondenti
Atto primo					
	Sì	No	No	No	
	Sì	Sì	No	No	
	No	Sì	No	No	
	No	Sì	Sì	No	
	No	No	Sì	No	
	No	No	No	No	
Pausa					
Atto secondo					
	No	No	Sì	No	
	No	No	Sì	Sì	C), D)
	No	No	No	Sì	
	No	Sì	No	Sì	A)
	No	Sì	No	No	
	No	No	No	No	
Fine					

Per ogni risposta è possibile verificare se l'affermazione in essa contenuta è vera o meno, esaminando le righe delle tabelle.

Nella risposta A), si cerca una riga in cui sia la principessa che il cavaliere sono sul palco insieme. Questo è il caso della quarta riga del secondo atto, perché la principessa entra nel palco dove il cavaliere si trova dalla seconda riga e rimane fino alla quinta riga. L'affermazione della risposta A) è quindi corretta almeno in un certo momento.

Nella risposta D), si cerca una riga in cui il cavaliere e il drago sono sul palco insieme. Questo è il caso della seconda riga del secondo atto, perché il cavaliere entra sul palco nella seconda riga, mentre il drago è già entrato sul palco nella prima riga e rimane fino alla terza riga. L'affermazione della risposta D) è quindi corretta almeno in un certo momento.

Nella risposta C), l'affermazione è di natura diversa. Se questo è vero, il cavaliere non deve essere entrato in scena durante tutto il primo atto. Qui bisogna guardare la colonna del cavaliere per il primo atto. C'è scritto «No» ovunque, quindi il cavaliere in realtà non è entrato in scena durante tutto il primo atto. Ma poi entra nella seconda riga del secondo atto, quindi anche l'affermazione della risposta C) è vera.

Se l'affermazione della risposta B) fosse vera, il re e il drago dovrebbero stare insieme sul palco in qualche riga. Ma in nessuna delle dodici righe c'è un «Sì» in entrambe le colonne. Il re esce di scena già nella terza riga del primo atto e non entra più in scena fino alla fine. Il drago, invece, non entra in scena fino alla quarta riga del primo atto. Forse i due si incontrano dietro il palco, ma sul palco non sono mai insieme. Pertanto l'affermazione della risposta B) non è corretta. Quindi B) è la risposta corretta.

Questa è l'informatica!

Anche se si può immaginare vividamente una storia basata sul corso dello spettacolo, solo una caratteristica è importante per ogni personaggio in questo compito: è o non è sul palco in un certo momento? Questa limitazione della visione a certe caratteristiche è chiamata *astrazione*.

Nell' informatica, tali astrazioni possono essere formulate molto bene. Per ognuna delle quattro figure si definisce una cosiddetta *variabile*, che risponde alla domanda se la figura è attualmente in scena. Le quattro variabili sono: «Re sul palco?», «Principessa sul palco?», «Drago sul palco?» e «Cavaliere sul palco?». Durante lo spettacolo, le risposte a queste domande cambiano continuamente; per ogni domanda la risposta è a volte «sì» e a volte «no». Nell' informatica, chiamiamo la risposta attuale ad una domanda il *valore* attuale della variabile associata. Il valore di una variabile può quindi cambiare continuamente (in matematica è diverso, le variabili non cambiano i loro valori nel tempo). La tabella nella spiegazione della risposta mostra le quattro variabili e i valori corrispondenti in qualsiasi momento.

Esiste anche un altro modo di considerare lo spettacolo. In ogni momento guardiamo quali personaggi sono sul palco in quel momento (quindi guardiamo i valori attuali delle quattro variabili.) Ogni possibile combinazione di figure la chiamiamo uno *stato* del palco. Così, quando una figura entra o esce dal palco, lo stato del palco cambia. Possiamo chiamarla quindi una *transizione* del palco da uno stato all'altro. Se si prende un pezzo di carta e si disegna un cerchio separato per ogni possibile stato (cioè ogni combinazione di figure), si può considerare il tutto come un'astrazione del palco.

Inoltre, è possibile disegnare le possibili transizioni come frecce che portano da uno stato all'altro. Se facciamo anche questo, abbiamo quello che in informatica chiamiamo *diagramma di stato* del palco.

All'inizio dello spettacolo il palco è vuoto. Per questo motivo chiamiamo lo stato corrispondente *stato iniziale*. Ora possiamo disegnare il percorso del gioco come un percorso nel diagramma di stato. Il percorso inizia nello stato iniziale e poi segue le frecce che corrispondono all'azione.

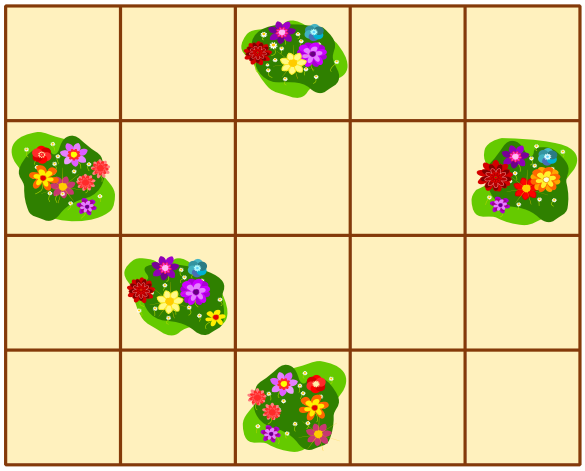
I diagrammi di stato sono molto importanti nell'informatica. Con quasi tutti i sistemi complicati, ad un certo punto bisogna pensare al diagramma di stato. Ma per gli esseri umani è spesso molto noioso lavorare con questi stati e transizioni astratti. I computer, invece, sono estremamente bravi per questo. Pertanto, vale la pena se le persone possono rappresentare i loro problemi con i diagrammi di stato in modo tale che i computer possano poi risolverli.

Parole chiave e siti web

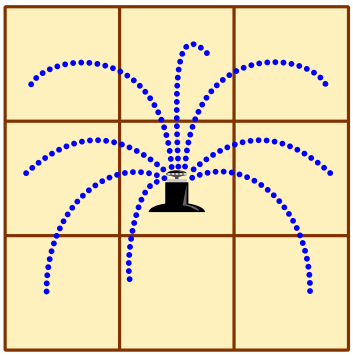
- Variabili: `https://it.wikipedia.org/wiki/Variabile_(informatica)`
- Diagramma di stato: `https://it.wikipedia.org/wiki/Diagramma_di_stato_(informatica)`

3. Annaffiare i fiori

Il giardino di Daniel è costituito da campi quadrati. In alcuni di questi campi ha piantato fiori:



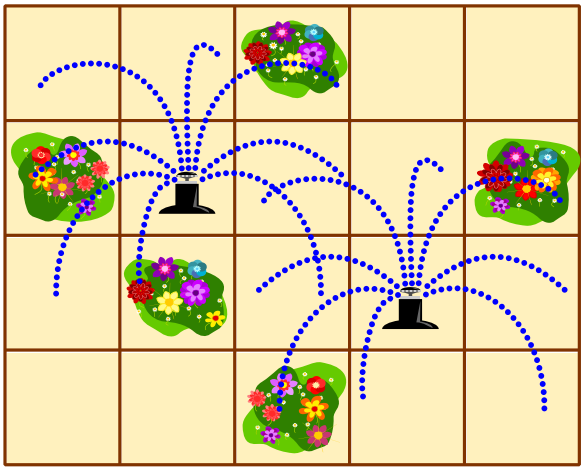
In estate vuole annaffiare i fiori con l'irrigatore a prato. Non può mettere un irrigatore nei campi con i fiori. Un irrigatore annaffia tutti i fiori negli 8 campi intorno a lui.:



Posiziona il minor numero possibile di irrigatori per annaffiare tutti i campi di fiori. Inseriscili nei campi del giardino di Daniel

Soluzione

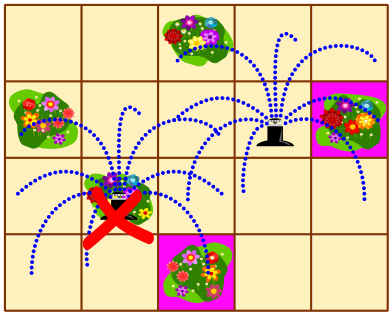
La seguente soluzione richiede due irrigatori per annaffiare tutti i campi di fiori:



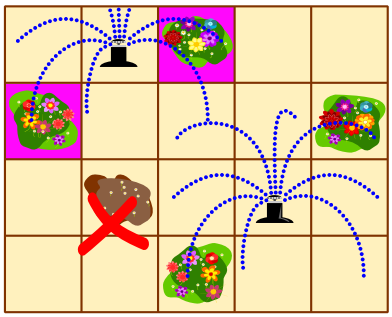
Tra il campo di fiori all'estrema sinistra e il campo di fiori all'estrema destra ci sono tre campi. Un solo irrigatore non può annaffiare campi così distanti tra loro.

Non c'è nemmeno un'altra soluzione con due soli irrigatori.

Per annaffiare il campo di fiori all'estrema destra e quello in basso al centro allo stesso tempo, un irrigatore deve essere posizionato esattamente dove si trova nella soluzione. Se si dovesse alzarne uno più in alto per annaffiare il campo di fiori in alto al centro, non si potrebbe più annaffiare il campo di fiori in basso al centro e non si potrebbero annaffiare i rimanenti tre campi di fiori con un irrigatore, perché nessun irrigatore può essere posizionato su un campo di fiori.



Per annaffiare il campo di fiori all'estrema sinistra e quello in alto al centro, si dovrebbe posizionare un irrigatore come mostrato nella soluzione o un campo sopra di esso. Ma se questo irrigatore deve annaffiare anche il campo di fiori nella seconda colonna da sinistra e quello nella terza fila dall'alto, non può essere posizionato in alto.



Questa è l'informatica!

Questo compito è un tipico problema di ottimizzazione: mentre è chiaro che tutti i campi di fiori devono essere annaffiati, il numero di irrigatori da prato necessari è variabile e deve essere il più piccolo possibile. Problemi di ottimizzazione simili sorgono quando, ad esempio, si vogliono rendere sicuri i villaggi con le caserme dei pompieri o fornire ai piazzali la ricezione dei cellulari.

In informatica si parla anche di *problemi di copertura*. Questi appartengono a una classe di problemi molto difficili in informatica. Il corretto posizionamento di un numero minimo di irrigatori era ancora abbastanza semplice. Ma la difficoltà aumenta talmente tanto con il numero di campi di fiori che presto non è possibile trovare una soluzione ottimale in un tempo ragionevole, anche con l'assistenza informatica.

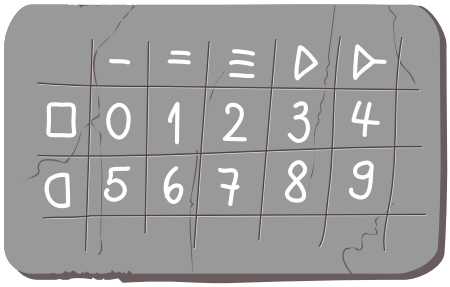
Una possibilità in questi casi è quindi quella di accontentarsi di soluzioni che possono non essere ottimali, ma che sono comunque buone. Non fa molta differenza se si posizionano 101 invece di appena 100 caserme dei pompieri o 1000 antenne di trasmettitori per cellulari invece di appena 990, ma il problema è spesso molto più facile da risolvere.

Parole chiave e siti web

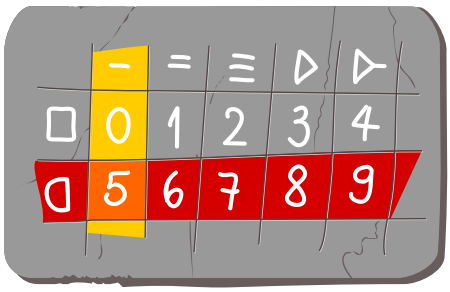
- Problema di ottimizzazione:
  `https://it.wikipedia.org/wiki/Problema_di_ottimizzazione`
- Problema di copertura:
  `https://it.wikipedia.org/wiki/Problema_di_copertura_dei_vertici`

4. Anno di costruzione del castello

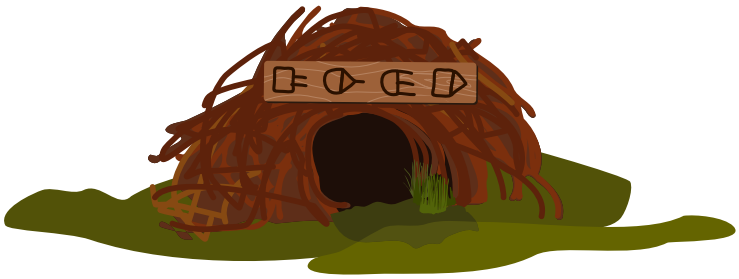
Sul cartello sopra l'ingresso di ogni castello dei castori è indicato l'anno di costruzione. I castori usano i loro caratteri per i numeri. La seguente tabella mostra come le cifre possono essere utilizzate per comporre i caratteri dei castori:



Ad esempio, i castori utilizzano la cifra «5» per formare il nuovo carattere  , che è assemblato nel seguente modo:



Questo è il castello di Cleveria:

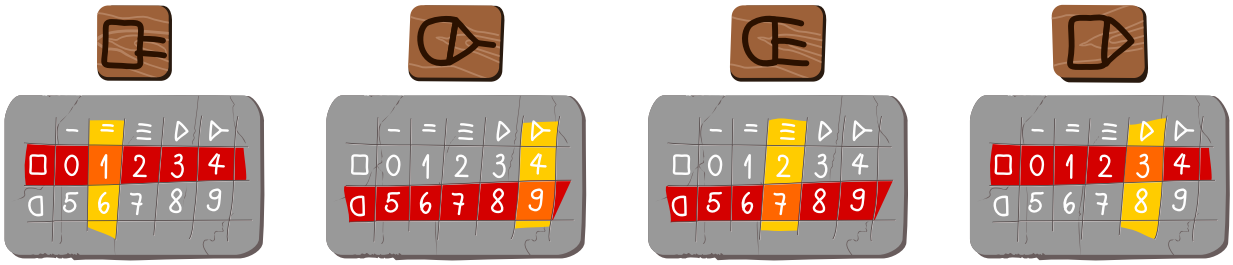


In quale anno è stato costruito il castello di Cleveria?

A) 0978
B) 1574
C) 1923
D) 1973
E) 1993
F) 2973
G) 6378

Soluzione

È possibile scoprire l'anno di costruzione del castello scegliendo la riga e la colonna appropriata per ogni simbolo. All'intersezione tra la colonna e la riga troverete il numero che cercate.



Le quattro cifre nell'ordine corretto danno il numero 1973.

Questa è l'informatica!

Mantenere la segretezza delle informazioni e proteggere i dati è un compito che risale a 4000 anni fa. A questo scopo sono stati sviluppati e utilizzati innumerevoli linguaggi segreti. Oggi la sicurezza dei dati è uno dei temi centrali dell'informatica. Uno dei metodi per proteggere i dati da letture non autorizzate è la *crittografia*. La cifratura trasforma un testo in chiaro in un testo cifrato. Ricostruire il testo in chiaro dal *testo cifrato* si chiama *decifrare*. La scienza del testo cifrato si chiama *crittologia*.

Le culture antiche utilizzavano per lo più scritture segrete, che venivano create codificando le lettere con altre lettere o con caratteri completamente nuovi. Il cifrario seguente è stato sviluppato specialmente per la competizione del castoro informatico, ma si basa su un concetto dell'antica Palestina. La regola di sicurezza dell'epoca era che si usavano solo codici segreti che si potevano imparare facilmente a memoria. Mantenere una descrizione scritta del codice segreto era considerato un rischio troppo grande. Una tabella, come si usa qui, è facile da imparare a memoria. Il famoso codice segreto dei massoni si basa su questo principio.

Invece di limitarsi a mettere insieme nuovi caratteri per le cifre, si possono anche inventare nuovi codici segreti per i testi. Per fare questo, si scrivono tutte le lettere in una tabella e si inventano nuovi simboli per le colonne e le righe, ottenendo così nuovi caratteri per tutte le lettere.

Parole chiave e siti web

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

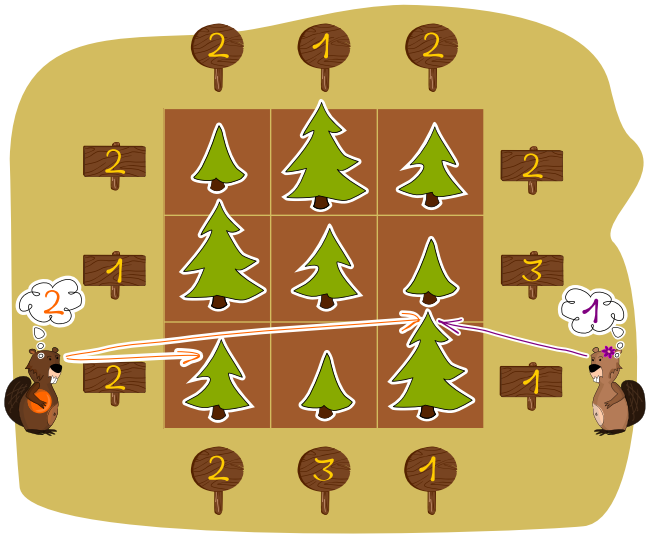
5. 3×3 sudoku con gli alberi

I castori piantano abeti in fila. Gli abeti hanno tre diverse altezze (1 , 2 e 3) e in ogni fila c'è esattamente un abete di ogni altezza.

Quando i castori guardano una fila di abeti da un'estremità, **non** possono vedere gli abeti più bassi nascosti dietro gli abeti più alti.

Alla fine di ogni fila di abeti c'è un cartello che indica quanti abeti un castoro può vedere da quel punto.

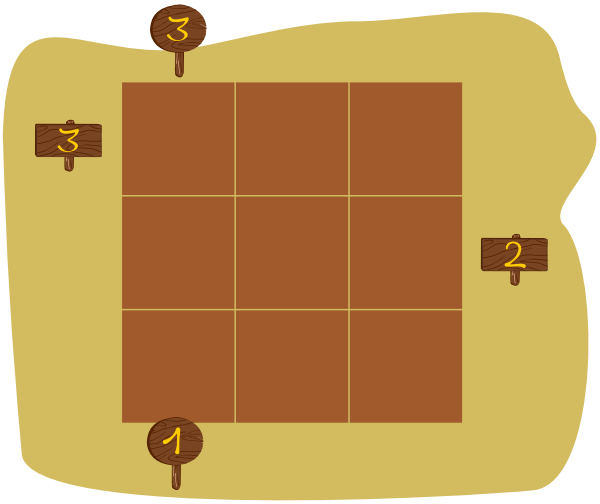
Ora i castori piantano nove abeti in un campo 3×3, come nell'esempio a destra.

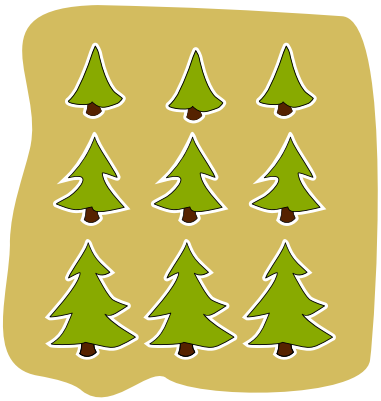


Si applicano le seguenti regole:

- in ogni riga (fila orizzontale) c'è esattamente un abete di ogni altezza;
- in ogni colonna (fila verticale) c'è esattamente un abete di ogni altezza;
- i cartelli con il numero di abeti visibili sono posizionati intorno al campo 3×3.

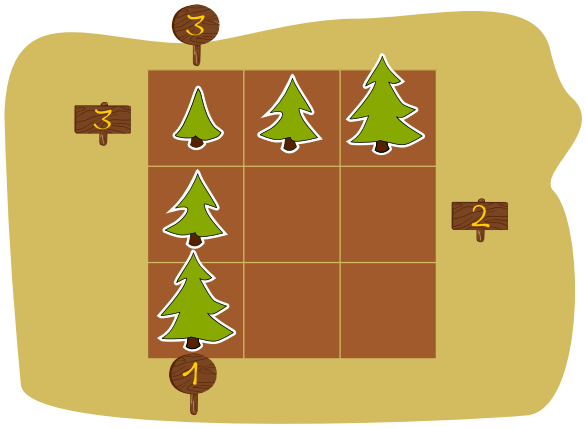
Scrivi in ogni campo l'altezza dell'albero corrispondente.





Soluzione

Sul campo, due cartelli indicano che da quelle posizioni si possono vedere tre abeti. Tutti e tre gli abeti in fila possono essere visti solo se gli abeti sono disposti in modo tale che la loro altezza aumenti, cioè da questa posizione. Questo determina la colonna a sinistra e la riga superiore:

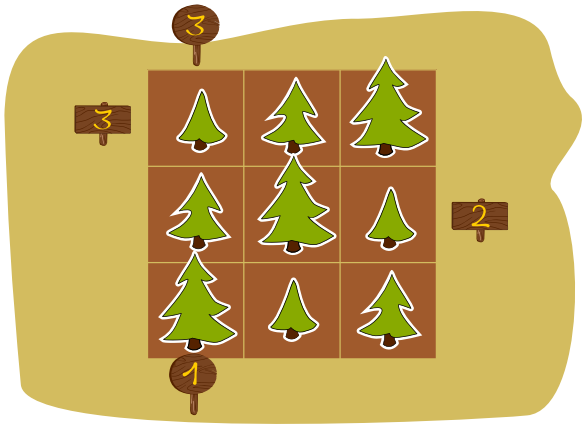


Il cartello a destra con il 2 richiede che da lì siano visibili due abeti, quindi deve esserci un abete di altezza 3 proprio al centro e questa fila centrale è quindi 2 (), 3 (), 1 ().

Gli altri campi sono compilati seguendo la regola del «Sudoku» secondo la quale deve esserci esattamente un abete per ogni altezza in ogni fila.

Al centro della fila inferiore deve esserci un abete di altezza 1 () perché le altre due altezze nella colonna centrale sono già assegnate. Infine, un abete di altezza 2 () deve essere posizionato in basso a destra per completare la fila.

La soluzione completa si presenta così:



Questa è l'informatica!

Questo compito si concentra su tre competenze di base degli informatici.

La prima è quella di trovare una soluzione che rispetti determinati vincoli o di correggere una soluzione proposta, se necessario.

In secondo luogo, si tratta della capacità di ricostruire gli oggetti a partire da informazioni parziali sulla loro rappresentazione. Questo è legato alla generazione di oggetti (rappresentazioni di oggetti) a partire dalle limitate informazioni disponibili, se si conosce la regolarità degli oggetti. Tali procedure possono essere utilizzate anche per la compressione di dati.

In terzo luogo, tali campi ad albero con cartelli possono essere utilizzati per generare codici autoverificanti. Gli errori che si verificano durante l'immissione o il trasporto delle informazioni possono essere rilevati o persino corretti automaticamente.

Parole chiave e siti web

- Sudoku
- Rilevazione e correzione d'errore:
  `https://it.wikipedia.org/wiki/Rilevazione_e_correzione_d'errore`

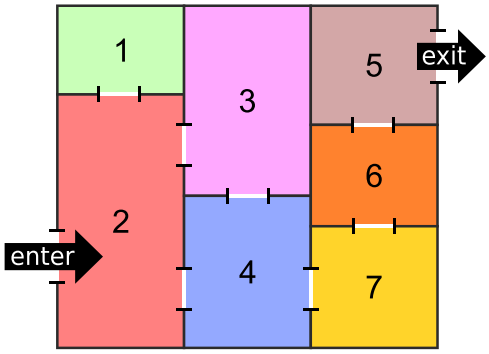
6. Visita al museo

Per un nuovo museo vengono proposte quattro planimetrie per le stanze. Ogni piano contiene le sette stanze da 1 a 7, e le stanze dovrebbero essere progettate in modo tale che i visitatori possano visitare tutte le stanze senza entrare due volte in una stanza.

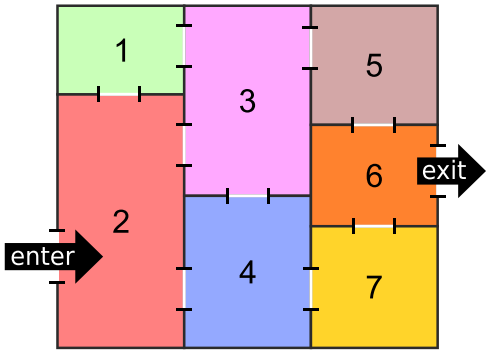
I visitatori iniziano la visita da «enter» e escono dal museo da «exit».

Quale piantina permette ai visitatori di entrare e uscire da ogni stanza esattamente una volta?

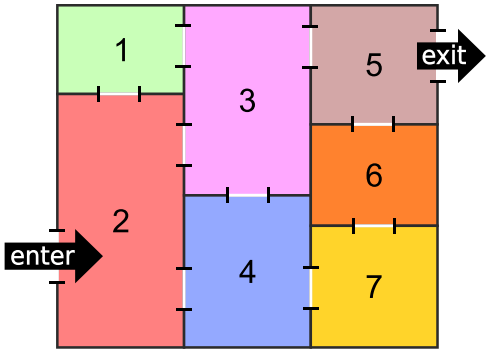
A)



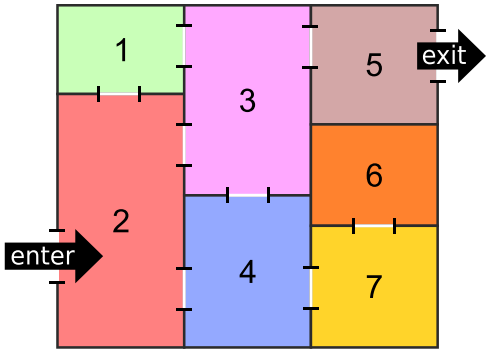
B)



C)

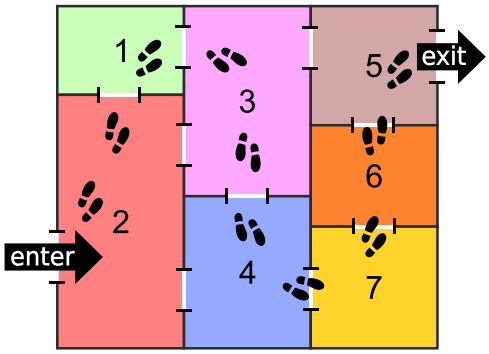


D)



Soluzione

Solo la piantina C permette ai visitatori di entrare e uscire da ogni stanza esattamente una volta. L'ordine delle stanze è: 2, 1, 3, 4, 7, 6, 5.



In generale, un tour di questo tipo è sempre impossibile se una delle stanze ha un solo ingresso. La spiegazione è la seguente: Se un visitatore entra in questa stanza, quando esce deve tornare nella stanza da cui è venuto, infrangendo la regola di entrare in ogni stanza una sola volta.

Nella piantina A, la camera 1 ha un solo ingresso.

Nella piantina D, la camera 6 ha un solo ingresso.

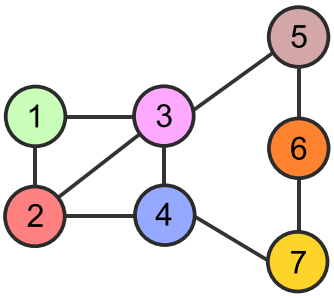
Nella piantina B, l'ultima stanza 6 è raggiungibile dalla stanza 5 o dalla stanza 7. Se il visitatore arriva dalla stanza 5, può entrare nella stanza 7, ma dopo può raggiungere l'uscita solo attraverso la stanza 6 (o viceversa), che ancora una volta infrange le regole.

Questa è l'informatica!

La maggior parte dei bambini o degli adolescenti risolve il problema per tentativi senza ulteriori rappresentazioni astratte. In questo modo utilizzano in una certa misura la strategia generale del *backtracking*. Per lo meno si rendono conto che si può imparare dai tentativi falliti e in questo caso si può tornare indietro per provare un'altra opzione. Allo stesso tempo, si confrontano con l'importante concetto di *non determinismo*, perché spesso hanno diverse opzioni tra cui scegliere.

Il compito è un esempio di un problema ben noto nell'informatica, la ricerca di un *cammino hamiltoniano*. In una rappresentazione astratta sotto forma di grafo, ogni stanza corrisponde ad un *nodo* e ogni porta tra due stanze corrisponde ad un *arco* tra i due nodi corrispondenti.

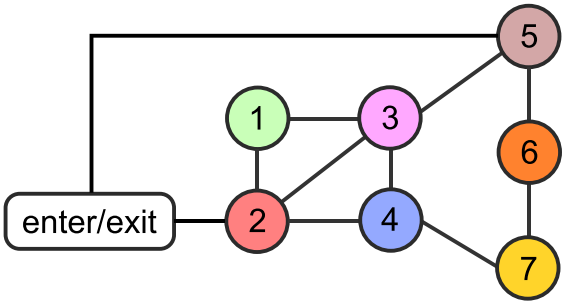
In termini astratti, il compito si presenta così:



Il compito ora è quello di trovare un percorso in questo grafo con le seguenti proprietà:

1. Il percorso inizia in 2 («ingresso»)
2. Il percorso termina in 5 («uscita»).
3. Ogni nodo viene visitato esattamente una volta.

Se si combina lo spazio esterno in un nodo, allora il tutto corrisponde alla ricerca di un *ciclo hamiltoniano* (un giro), dove ogni nodo viene anche attraversato esattamente una volta e si finisce di nuovo al nodo di partenza.



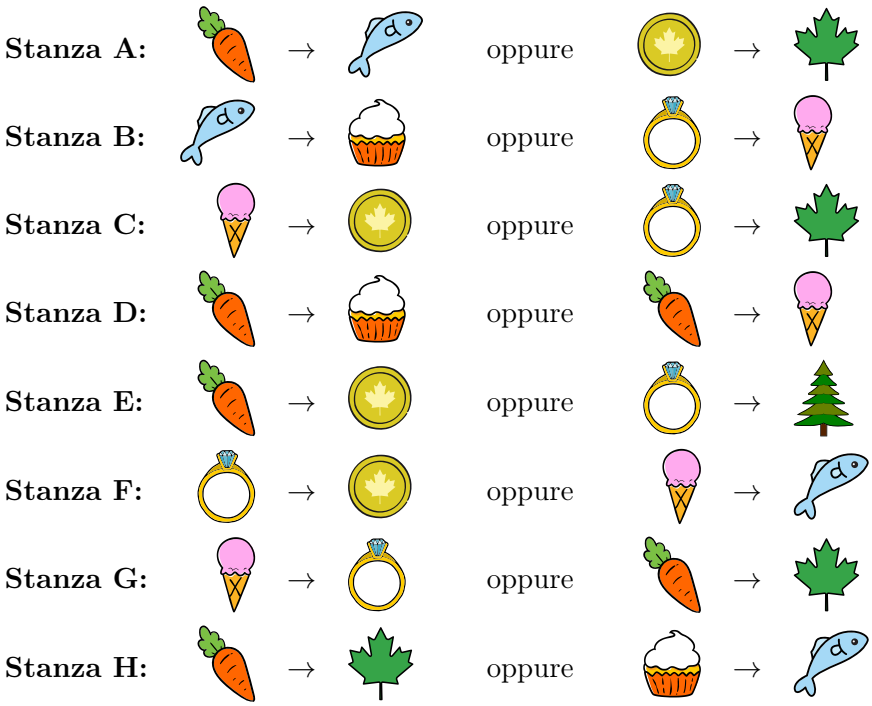
Parole chiave e siti web

- Teoria dei grafi: `https://it.wikipedia.org/wiki/Teoria_dei_grafi`
- Cammino hamiltoniano: `https://it.wikipedia.org/wiki/Cammino_hamiltoniano`

7. Castoro al castello

Un castoro intelligente ha bisogno di un abete per costruire una diga nel fiume. Purtroppo ha solo una carota. Oggi è giorno di mercato nel castello e il castoro vuole scambiare la sua carota con un abete.

Ogni stanza del castello offre due offerte di scambio. La tabella mostra queste offerte:



Per esempio, nella stanza B, il castoro può ottenere un cono per un anello, ma non viceversa.

In quale ordine il castoro intelligente deve attraversare le stanze per possedere finalmente l'abete desiderato?

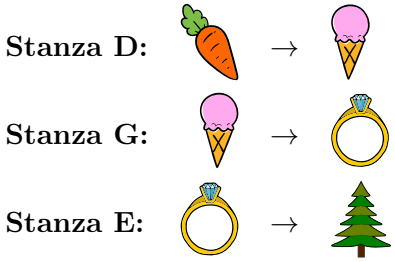
A) DGE: Prima la stanza D, poi la stanza G e infine la stanza E.
B) GGE: Prima la stanza G, poi di nuovo la stanza G e infine la stanza E.
C) AGE: Prima la stanza A, poi la stanza G e infine la stanza E.
D) DBC: Prima la stanza D, poi la stanza B e infine la stanza C.

Soluzione

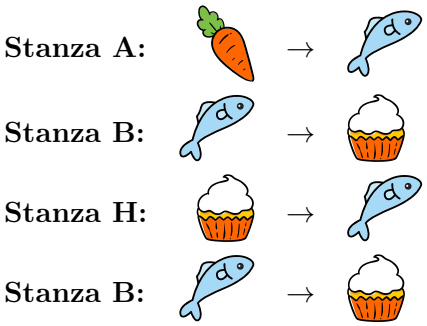
La risposta corretta è A) DGE: prima la stanza D, poi la stanza G e infine la stanza E.

Nella stanza D il castoro scambia la sua carota con un cono. Poi va nella stanza G, dove scambia il cono con un anello. Alla fine il castoro va nella stanza E per scambiare l'anello con un abete.

Questa sequenza complessiva si presenta così:



Per trovare un ordine adeguato delle stanze, due diverse strategie sono utili. La prima strategia cerca di considerare tutte le possibilità di scambio. Si inizia con il primo scambio, dove si può scambiare la carota in cinque stanze (A, D, E, G e H) con 6 oggetti diversi. In seguito, tutte le possibilità di scambio per questi 6 oggetti vengono nuovamente considerate. Questo è complesso e si può anche girare in cerchio, come nell'esempio seguente, dove il castoro può visitare le stanze B e H tutte le volte che vuole:

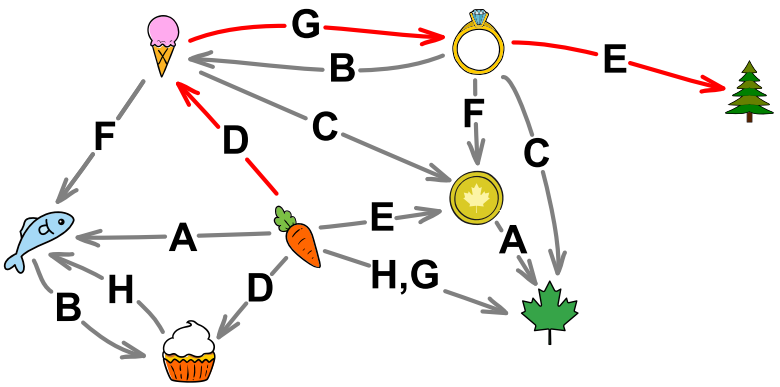


Pertanto questa prima strategia è molto complessa e solo con un po' di fortuna si può avere successo in breve tempo.

La seconda strategia porta rapidamente all'obiettivo in questo compito concreto. Si basa sull'avvio della ricerca dall'obiettivo desiderato, l'abete. Solo nella stanza E il castoro può ottenere l'abete desiderato e l'abete può essere ottenuto solo in cambio di un anello. Il prossimo oggetto desiderato è quindi un anello! L'anello può essere ottenuto solo in una stanza, la stanza G, in cambio di un cono. È possibile ottenere il cono sia nella stanza B scambiandolo con un anello, che nella stanza D scambiandolo con una carota. Quindi il castoro intelligente deve iniziare il suo scambio nella stanza D.

Per una migliore visione d'insieme, la tabella degli scambi possibili può essere visualizzata sotto forma di grafo orientato con archi. Ogni nodo del grafo rappresenta un oggetto di scambio e ogni

arco in uscita rappresenta una possibilità di scambio. Inoltre, l'arco indica lo spazio in cui esiste questa possibilità di scambio.



Questa rappresentazione visiva degli oggetti di scambio, delle possibilità di scambio e delle stanze permette di scoprire facilmente come arrivare dalla carota all'abete, ovvero su un percorso nel grafo orientato: Prima la stanza D, poi la stanza G e infine la stanza E.

Questa è l'informatica!

I *processi di calcolo* possono essere considerati a livello generale come *conseguenze di trasformazioni* (qui si tratta di processi di scambio) o, in modo equivalente, come conseguenze di *stati* di un sistema. Lo stato iniziale del sistema nel nostro caso è la carota e la trasformazione (*la transizione*) da carota a cono cambia questo stato in cono.

Una transizione porta così da uno stato all'altro. Una sequenza di transizioni è anche chiamata *calcolo*.

Questo compito si occupa quindi anche di calcoli a livello molto generale. In questo caso, il sistema *non è deterministico*; ciò significa che ci sono a volte diverse possibili fasi di calcolo, cioè, come nel compito, diversi possibili scambi. Il non determinismo è un altro importante concetto di modellazione nell'informatica. Le possibili fasi di calcolo sono descritte da *regole di trasformazione* (la tabella con possibilità di scambio). Determinare se il castoro può scambiare una carota con un abete, cioè se un certo stato obiettivo del sistema può essere raggiunto da un certo stato di partenza, è il famoso STCON (*st-connettività*) con numerose applicazioni.

Il compito di cui sopra mostra che a volte è una buona idea cercare lo stato di partenza dallo stato di destinazione invece che il contrario. Questa strategia è chiamata anche *ricerca inversa*.

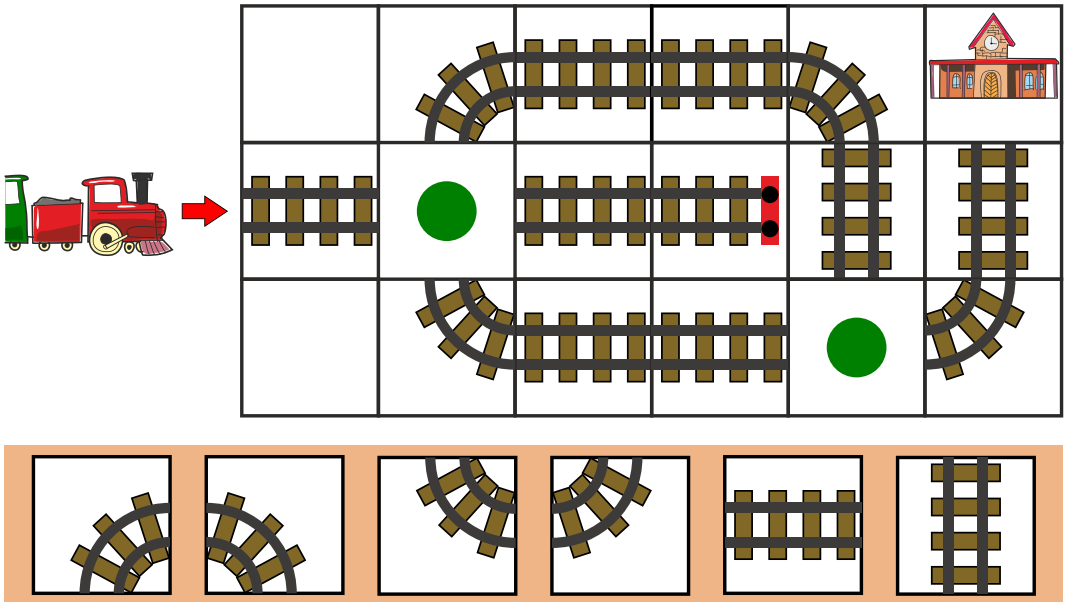
Quando si confrontano le diverse strategie di soluzione, si può notare che il grafo orientato è un modo illustrativo di rappresentare un cosiddetto *spazio di stato* di un sistema con tutte le possibili transizioni tra gli stati. In questo modello di base si potrebbero affrontare i ben noti *algoritmi di ricerca* di base nei grafi, cioè la *ricerca in ampiezza* e la *ricerca in profondità*.

Parole chiave e siti web

- Grafi: `https://it.wikipedia.org/wiki/Grafo`
- Ricerca in profondità: `https://it.wikipedia.org/wiki/Ricerca_in_profondità`
- Ricerca in ampienzza: `https://it.wikipedia.org/wiki/Ricerca_in_ampiezza`

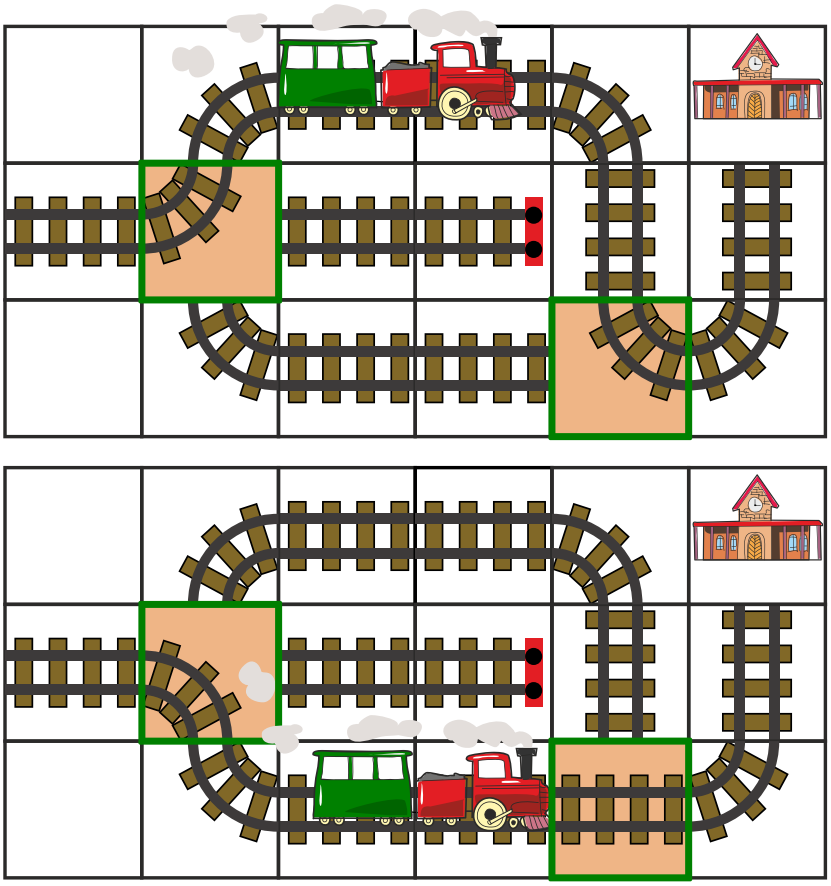
8. Prossima fermata, stazione!

Scegli i binari corretti da mettere nei campi con il punto verde affinché il treno possa raggiungere la stazione.



Soluzione

Per questo problema ci sono le seguenti 2 soluzioni:



Con altre combinazioni il treno deraglia o si scontra con il paraurti.

Questa è l'informatica!

Proprio come un treno viaggia ostinatamente sui binari, un computer esegue ostinatamente le istruzioni di un programma. Non è in grado di riconoscere quando il programma contiene un errore e può bloccarsi, così come un treno può deragliare se i binari sono costruiti in modo errato. Quindi, quando si scrive un programma, bisogna stare molto più attenti che, per esempio, spiegare la strada per la stazione a una persona.

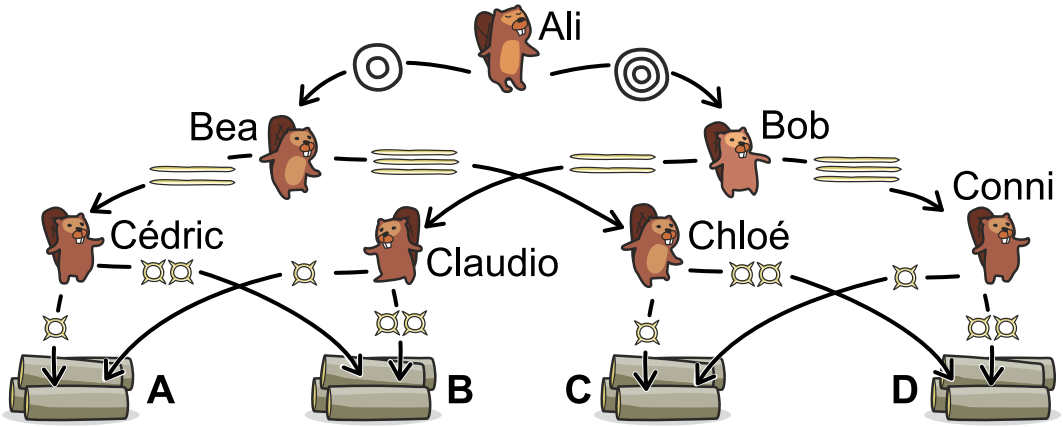
Il compito di cui sopra consiste nell'inserire i comandi mancanti nei punti giusti di un programma in modo da raggiungere l'obiettivo.

Parole chiave e siti web

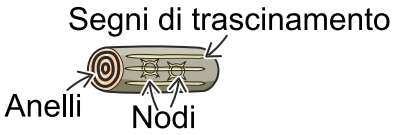
- Programma
- Instruzione: `https://it.wikipedia.org/wiki/Istruzione_(informatica)`
- `https://it.wikipedia.org/wiki/Algoritmo`

9. Tronchi e pile

Nel villaggio dei castori i tronchi sono divisi in quattro gruppi (A, B, C, D) secondo tre caratteristiche (numero di anelli del tronco, segni di trascinamento sulla corteccia e numero di nodi). Il seguente diagramma di decisione mostra come si dividono fra i gruppi.

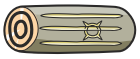


Per esempio, questo tronco viene inserito nella pila D in seguito alle seguenti decisioni:



- Ali vede tre anelli e dà il tronco a Bob;
- Bob vede tre segni di trascinamento e dà il tronco a Conni;
- Conni vede due nodi e mette il tronco sulla pila D.

Su quale pila è collocato questo tronco?



A) Pila A
B) Pila B
C) Pila C
D) Pila D

Soluzione

La risposta corretta è la pila C. Questo perché Ali vede due anelli e dà il tronco a Bea. Bea vede tre segni di trascinamento e dà il tronco a Chloé. Chloé vede un nodo e mette il tronco sulla pila C.

Se lo si desidera, è possibile determinare per ogni pila quali tronchi appartengono alla rispettiva pila. Ci sono due tipi di tronchi su ogni pila.

Sulla pila **A**:

- Tronchi con 2 anelli, 2 segni di trascinamento e 1 nodo.
- Tronchi con 3 anelli, 2 segni di trascinamento e 1 nodo.

Sulla pila **B**:

- Tronchi con 2 anelli, 2 segni di trascinamento e 2 nodi.
- Tronchi con 3 anelli, 2 segni di trascinamento e 2 nodi.

Sulla pila **C**:

- Tronchi con 2 anelli, 3 segni di trascinamento e 1 nodo.
- Tronchi con 3 anelli, 3 segni di trascinamento e 1 nodo.

Sulla pila **D**:

- Tronchi con 2 anelli, 3 segni di trascinamento e 2 nodi.
- Tronchi con 3 anelli, 3 segni di trascinamento e 2 nodi.

Questa è l'informatica!

Questo compito tocca diversi concetti dell'informatica.

In primo luogo, viene affrontato il concetto di diagrammi decisionali, che hanno applicazioni molto versatili nel campo dell'informatica. Qui vengono utilizzati per classificare gli oggetti in categorie selezionate (molto spesso si tratta di alberi di decisione, un tipo speciale di diagrammi decisionali. Il diagramma decisionale del compito non è un albero decisionale in questo caso, perché al livello più basso due gruppi sono posizionati sulla stessa pila).

Qui si può anche pensare al diagramma decisionale come a una rappresentazione astratta dei valori di una funzione di diverse variabili. A livello terminologico, in informatica si parla di «branching programs» (inglese per «programmi di ramificazione»).

Si riferisce anche al concetto di attributi (caratteristiche o proprietà) degli oggetti. Qui gli oggetti hanno tre attributi (anelli del tronco, segni di trascinamento, nodi), con ogni attributo che ha due possibili valori (due o tre anelli annuali o segni di trascinamento e uno o due nodi).

Ci sono molte possibili applicazioni per tali diagrammi decisionali. Uno di questi è la classificazione dei pacchetti di dati quando vengono inviati attraverso una rete (con router o switch).

Parole chiave e siti web

- Albero di decisione: `https://it.wikipedia.org/wiki/Albero_di_decisione`

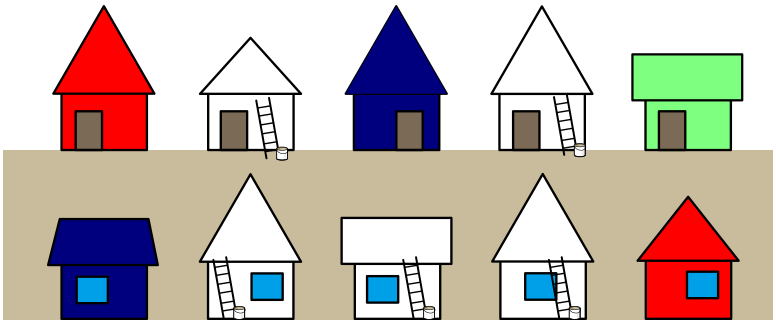
10. Case colorate

Gli abitanti di una strada vogliono dipingere con dei colori le loro case bianche. Ogni casa dovrebbe avere uno dei tre colori: verde chiaro, rosso o blu scuro. Le seguenti regole si applicano per evitare di sembrare noioso:

- Due case che si trovano direttamente l'una accanto all'altra non devono avere lo stesso colore.
- Due case che si trovano direttamente l'una di fronte all'altra non devono avere lo stesso colore.

Alcuni residenti hanno già dipinto le loro case a colori. I restanti residenti devono ora dipingere le loro case in modo che le regole non vengano violate.

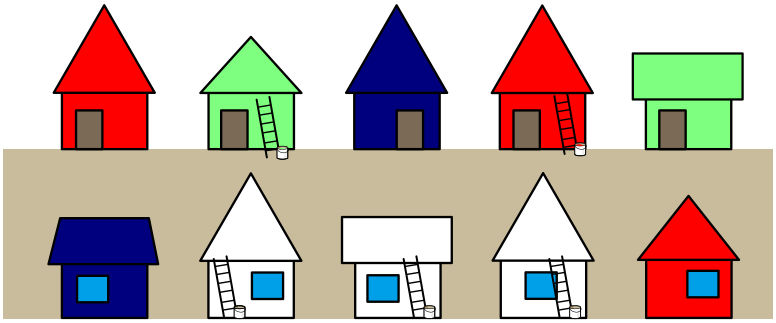
Trova i colori corrispondenti per i residenti.



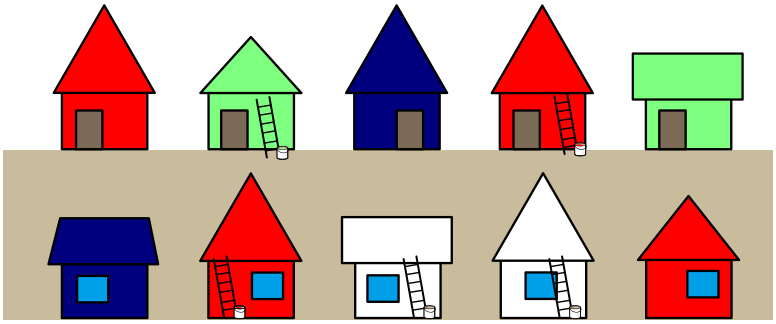
Soluzione

Il modo più semplice per scoprire i colori delle case è scoprirli passo dopo passo.

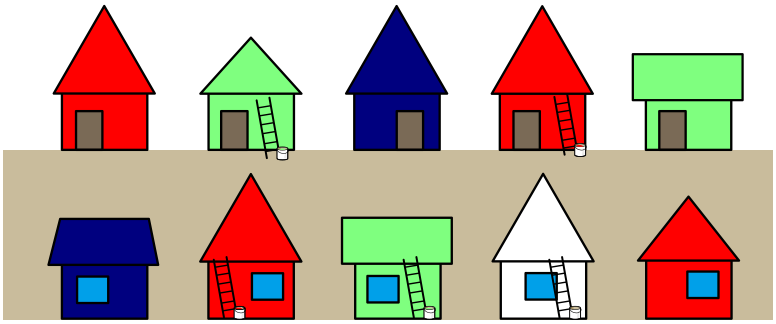
Le due case bianche sul lato superiore della strada sono circondate da due case di colore diverso a sinistra e a destra. Pertanto, possono essere dipinti in un solo colore particolare senza infrangere le regole: la casa bianca in alto a sinistra in verde chiaro e la casa bianca in alto a destra in rosso.



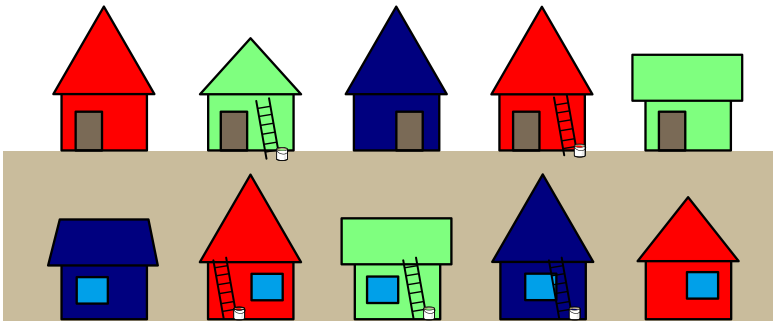
Poi si vede che la casa bianca in basso a sinistra deve essere dipinta di rosso perché la casa direttamente a sinistra è blu scuro e la casa di fronte è verde chiaro:



Quasi la stessa considerazione può essere fatta ora per la casa di mezzo sul lato inferiore della strada: Deve essere dipinta di verde chiaro, perché direttamente alla sua sinistra c'è la casa appena dipinta di rosso e di fronte c'è una casa blu scuro.



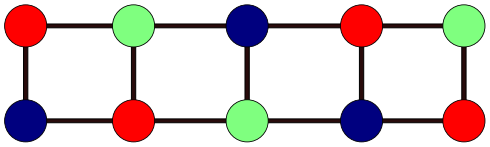
Infine, si può anche scegliere il colore per la casa bianca sul lato destro della strada: La casa direttamente a destra e la casa di fronte sono entrambe rosse, ma poiché la casa direttamente a sinistra è verde chiaro, l'unica opzione rimasta è dipingere la casa di blu scuro:



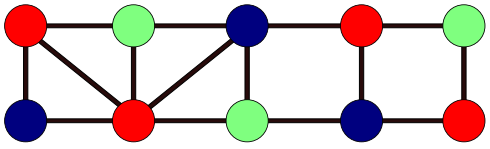
Questa è l'informatica!

In termini astratti, questo compito consiste nel trovare una soluzione che soddisfi le restrizioni date (regole). Questo è un problema molto comune nell'informatica.

Le case e i loro vicini diretti (sia a sinistra che a destra e dall'altra parte della strada) possono essere modellati bene con l'aiuto di un *grafo*, una struttura di dati ampiamente utilizzata nell'informatica. Ogni casa è un *nodo* e ogni vicinato diretto è un *arco*:



Nell' immagine, i nodi sono già colorati come le case corrispondenti. Per le case c'era la regola che le case vicine dovevano avere colori diversi. Nell' immagine la colorazione dei nodi è quindi tale che i nodi collegati direttamente da un arco non hanno mai lo stesso colore. Che ci sia una colorazione valida del grafo con tre colori non è ovvio. Se si aggiungono due archi come nell'immagine successiva, non c'è più una colorazione valida: non importa come si distribuiscono i tre colori in questo grafo, ci sono sempre due nodi direttamente collegati con lo stesso colore.



Ma con quattro colori funziona di nuovo. Forse funziona sempre con quattro colori? La risposta è di nuovo no. Ma almeno un tipo di grafi può sempre essere colorato con quattro colori: i cosiddetti *grafi planari*. Si tratta di grafi che possono essere disegnati in modo che nessun arco si incroci. (Il grafo nell'ultima immagine non è planare, cioè a causa delle connessioni dei quattro nodi all'estrema sinistra). Che i grafi planari hanno una colorazione valida con quattro colori è chiamato il *teorema dei quattro colori*.

Il teorema dei quattro colori è particolarmente interessante per la creazione di mappe. Se si immagina ogni paese come un nodo e poi si collegano i paesi vicini con un arco, si ottiene sempre un grafo planare. (A rigor di termini, per questo dobbiamo escludere l'esistenza delle cosiddette enclavi ed esclavi, cioè parti di un paese che si trovano completamente in un altro paese). Questo grafo può quindi essere colorato con quattro colori validi, e quindi anche i paesi corrispondenti sulla mappa possono essere colorati con quattro colori, in modo che i paesi vicini abbiano sempre colori diversi.

La prova che quattro colori sono sufficienti non è facile. Che cinque colori siano sufficienti era già noto 200 anni fa. I matematici Kenneth Appel e Wolfgang Haken hanno dimostrato nel 1976 che quattro colori sono sufficienti. Hanno usato un computer per controllare un gran numero di eccezioni e controesempi. Ci sono volute più di mille ore al computer per farlo. Controllare tutto a mano sarebbe stato del tutto impossibile. Molti matematici si sono poi chiesti se una tale prova sia valida, perché bisogna fidarsi del computer.

Parole chiave e siti web

- Teorema dei quattro colori: `https://it.wikipedia.org/wiki/Teorema_dei_quattro_colori`
- Grafo: `https://it.wikipedia.org/wiki/Grafo`

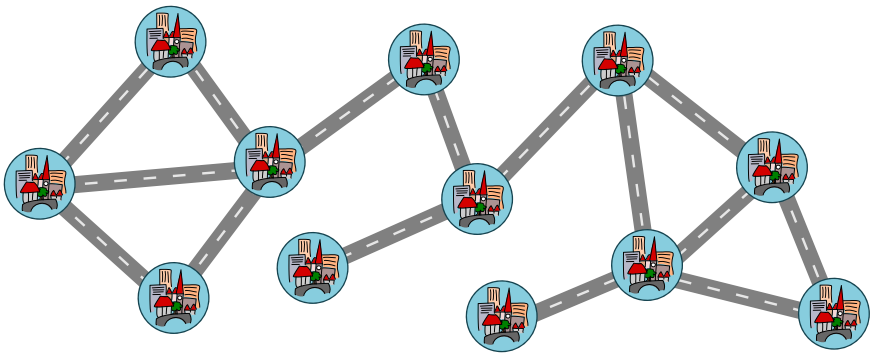
11. Considerazioni epidemiologiche

Biberland è composta da 12 città, che sono collegate da strade. Le città che sono direttamente o indirettamente collegate da strade formano una comunità commerciale. La mappa mostra quindi nella sua forma attuale un'unica comunità commerciale di 12 città.

Per contenere un'epidemia, il traffico deve essere ridotto. Il parlamento di Biberland decide di chiudere esattamente due strade per dividere le città in tre comunità commerciali separate.

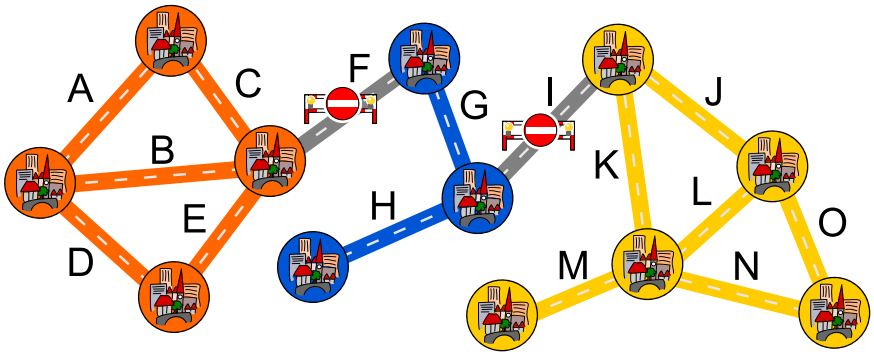
Per non isolare nessuno più del necessario, la più piccola comunità commerciale dovrebbe essere composta dal maggior numero possibile di città.

Quali due strade dovrebbero essere chiuse? Indicale.



Soluzione

La risposta corretta è: le strade F e I nella foto qui sotto devono essere bloccate. Questo crea comunità commerciali di 3, 4 e 5 città.



E' ovvio che dobbiamo guardare solo alle strade che, se sono bloccate, causeranno anche la divisione della comunità commerciale perché sono l'unico collegamento. Dopo tutto, abbiamo bisogno di due vere e proprie divisioni per raggiungere tre unità. Ad esempio, non ha senso chiudere la strada B, perché si possono comunque raggiungere tutte le città via A e C. Rimangono quindi solo le candidate F, G, H, I e M per il blocco.

Se si provano tutte le 10 possibilità di chiudere due delle cinque strade, si otterrà la risposta di cui sopra. Come essere umano, si vede subito che il blocco H o M taglierebbe solo una singola città ed è quindi fuori questione. Ciò limita ulteriormente il numero di possibilità da prendere in considerazione.

Questa è l'informatica!

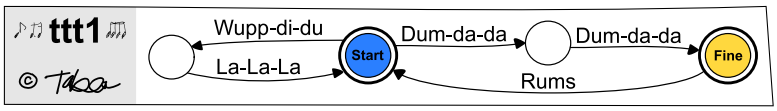
Nell'informatica, una data rete è spesso suddivisa in cosiddette *componenti connesse*. In una componente connessa, tutte le parti sono collegate tra loro attraverso percorsi diretti o indiretti, mentre non c'è alcun collegamento tra le diverse componenti connesse. Ovviamente, l'applicazione è in reti di computer dove è rilevante quali computer possono essere raggiunti da quali altri. Ma anche, ad esempio, nel riconoscimento ottico dei caratteri (OCR), è importante sapere quali punti sono «connessi».

Parole chiave e siti web

- Componente connessa:
  `https://it.wikipedia.org/wiki/Componente_connessa_(teoria_dei_grafi)`

12. Il ritmo di Tabea

Tabea ha molto successo nel creare testi di canzoni con il marchio ttt: Tabea's Tactful Texts. I testi possono essere prodotti con il seguente diagramma ttt1:



Per produrre una canzone, Tabea inizia da «Start» (Start) e segue una delle frecce in uscita. Se ci sono diverse possibilità, può scegliere quella che preferisce. Canta le sillabe corrispondenti lungo il percorso nell'ordine dato. Se raggiunge «Fine» (Fine), la canzone può finire ma può anche continuare.

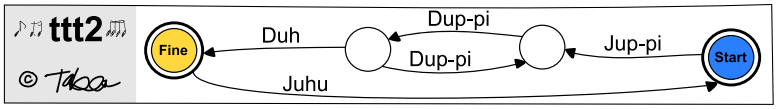
Possibili canzoni possono essere:

«Wupp-di-du La-La-La Wupp-di-du La-La-La
Dum-da-da Dum-da-da Rums Dum-da-da Dum-da-da»

Oppure

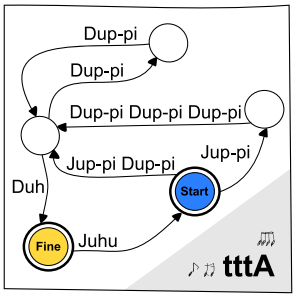
«Dum-da-da Dum-da-da Rums Wupp-di-du La-La-La
Dum-da-da Dum-da-da Rums Wupp-di-du La-La-La
Dum-da-da Dum-da-da Rums Dum-da-da Dum-da-da»

Nel novembre 2020 Tabea produce nuovi testi con il diagramma ttt2:

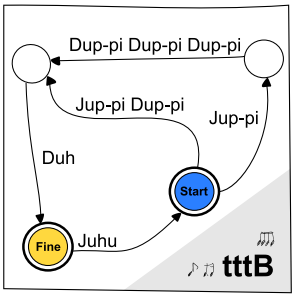


Con quali dei seguenti diagrammi si possono creare esattamente gli stessi testi come con il diagramma ttt2?

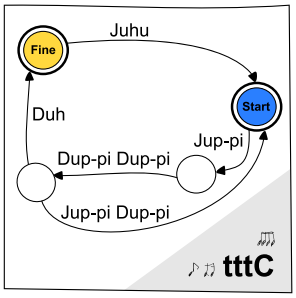
A)



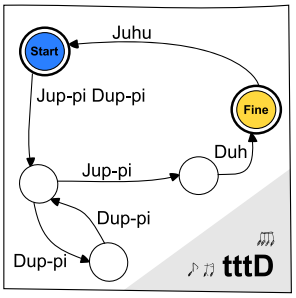
B)



C)

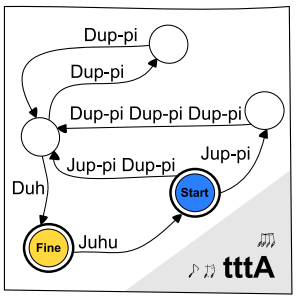


D)

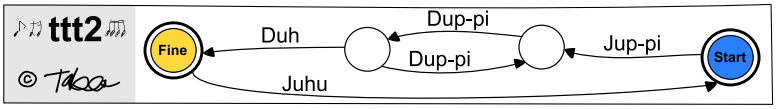


Soluzione

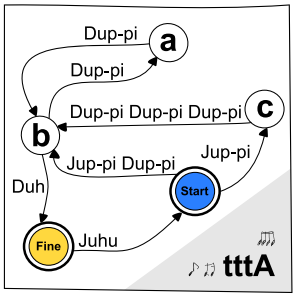
La risposta corretta è A) ossia il diagramma tttA.



Se si produce un brano con il diagramma ttt2, esso inizierà sempre con «Jup-pi» e seguirà almeno un «Dup-pi». Ora continua o direttamente con «Duh» o con un numero pari di ulteriori «Dup-pi» e poi «Duh». Ora la canzone può essere terminata o continuare con un «Juhu» e ricominciare dall'inizio.



Il diagramma tttA raggiunge esattamente lo stesso risultato: Dallo «Start», il brano può partire direttamente in b e quindi con «Jup-pi Dup-pi» o via c con «Jup-pi Dup-pi Dup-pi Dup-pi Dup-pi Dup-pi». Dopo di che, una deviazione tramite a permette di aggiungere un numero pari qualsiasi di «Dup-pi», poi si può usare «Duh» per arrivare alla fine della canzone. Proprio come in ttt2 si può ricominciare dopo «Juhu».



Sia il diagramma ttt2 che il diagramma tttA possono produrre dopo «Jup-pi» un numero dispari a volontà di «Dup-pi». Il diagramma tttB invece può produrre solo 1 o 3 «Dup-pi» di fila e il diagramma tttC solo 1 o 2. Il diagramma tttD crea invece un numero dispari di Dup-pi, ma precede il «Duh» finale con un altro «Jup-pi», che ttt2 non può creare. Quindi tttA è l'unica risposta possibile.

Questa è l'informatica!

Un compito importante dell'informatica è quello di riconoscere le strutture nei dati, ad esempio nel linguaggio, come il testo di una canzone. I diagrammi rappresentano i cosiddetti automi finiti con i quali si possono definire regole molto severe per la generazione e il riconoscimento di determinate lingue. Questo a sua volta è cruciale nello sviluppo dei linguaggi di programmazione. Nel nostro esempio,

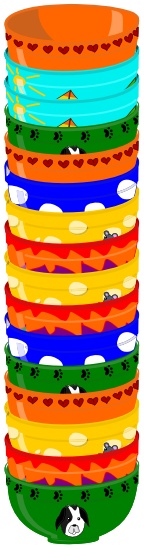
l'automa finito descrive l'insieme di canzoni che possono essere create con esso. Il riconoscimento del modello è importante anche in molti altri settori, come l'elaborazione del linguaggio naturale.

Parole chiave e siti web

- Automi finiti: `https://it.wikipedia.org/wiki/Automa_a_stati_finiti_deterministico`
- Linguaggi formali: `https://it.wikipedia.org/wiki/Linguaggio_formale`
- `https://sites.google.com/isabc.ca/computationalthinking/pattern-recognition`

13. Pila di ciotole

Tre fratelli vogliono mangiare da tre ciotole identiche a colazione. In cucina hanno un'alta pila di ciotole. Per precauzione possono prendere sempre solo una ciotola alla volta dalla cima della pila.

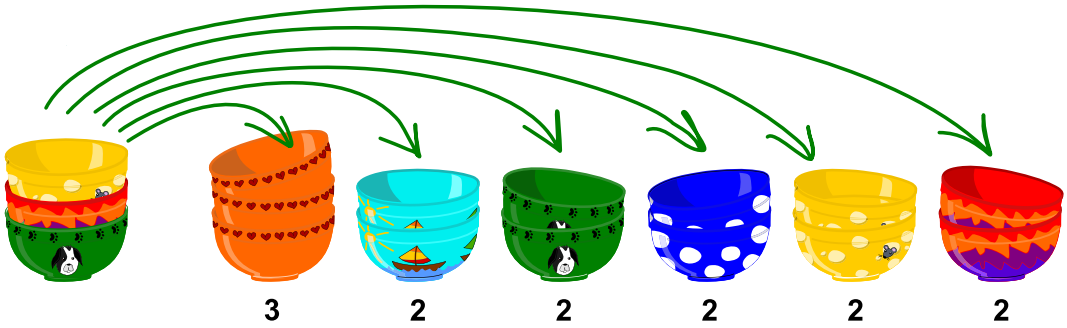


Qual è il numero minore di ciotole che si devono prendere dalla pila mostrata nella foto per averne tre di un tipo?

A) 3 ciotole
B) 4 ciotole
C) 5 ciotole
D) 6 ciotole
E) 7 ciotole
F) 8 ciotole
G) 9 ciotole
H) 10 ciotole
I) 11 ciotole
J) 12 ciotole
K) 13 ciotole
L) 14 ciotole
M) 15 ciotole
N) 16 ciotole

Soluzione

Risposta K): Almeno 13 ciotole devono essere prese dalla pila per ottenere tre ciotole dello stesso tipo.



Questa è l'informatica!

Una *pila*, in informatica spesso chiamata *stack* e a volte struttura *LIFO*, è un modo molto comune di immagazzinare le cose. Uno stack è una struttura molto semplice, ma potente, che viene spesso utilizzata nella programmazione. Ci sono regole su come mettere le cose su una pila e come rimuoverle, di solito solo dall'alto. In questo compito ci occupiamo solo di rimuovere le cose dalla pila. La regola dice che solo l'oggetto più in alto può essere preso dalla pila. Se si vuole ottenere la decima ciotola nella pila, bisogna togliere dieci ciotole una ad una. È importante avere un posto dove poter mettere le altre nove ciotole; questo vale anche per la programmazione. Se abbiamo una seconda pila e possiamo fare le pile alte quanto vogliamo, allora teoricamente potremmo già calcolare con essa tutto ciò che può essere calcolato con un computer! (Nell'informatica, questa proprietà è anche chiamata *Turing equivalenza*.) Queste semplici pile sono davvero potenti!

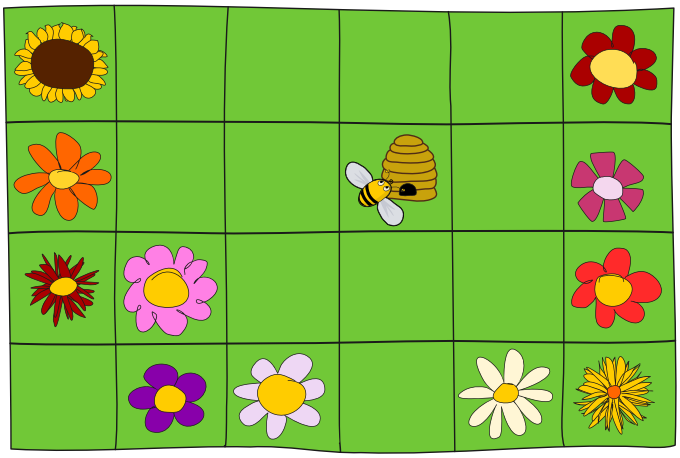
Parole chiave e siti web

- Pila: `https://it.wikipedia.org/wiki/Pila_(informatica)`
- Macchina di Turing: `https://it.wikipedia.org/wiki/Macchina_di_Turing`

14. Dall'alveare ai fiori

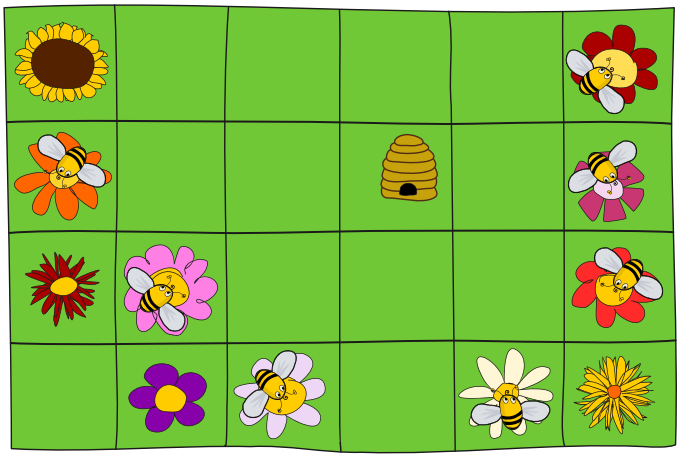
Un'ape vola in su, in giù, a sinistra o a destra. Per volare la distanza di un quadrato ci impiega 10 minuti. Vola dall'alveare, per un massimo di 30 minuti prima di tornare indietro.

Disegna un cerchio attorno ai fiori che si possono raggiungere dall'alveare in massimo 30 minuti.



Soluzione

I fiori con sopra un'ape sono raggiungibili dall'alveare in massimo 30 minuti:



L'immagine sottostante mostra per ogni campo quanti minuti necessita un'ape per raggiungerlo dall'alveare. Così in mezz'ora tutti i campi con un 10, 20 o 30 sono raggiungibili.

Riempire i numeri funziona in questo modo: Nelle caselle accanto all'alveare scriviamo 10 perché l'ape ha bisogno di 10 minuti per volare dall'alveare a lì. Poi scriviamo 20 in tutti i campi vuoti accanto a un campo con 10, perché l'ape ha bisogno di 10 minuti per volare da un campo all'altro. Continueremo a farlo. Così ne scriviamo 30 in tutti i campi vuoti accanto a un campo con 20. Poi ne scriviamo 40 in tutti gli spazi vuoti accanto a uno spazio con 30. Infine, scriviamo 50 in tutti i campi vuoti accanto a un campo con 40.

Questa è l'informatica!

Quando si risolve il compito, si calcola per ogni campo il tempo in cui un'ape può raggiungerlo dall'alveare. Per prima cosa vengono determinati i campi raggiungibili in 10 minuti. Questi vengono poi utilizzati per determinare i campi che si trovano a 20 minuti di distanza. Utilizzando i campi distanti 20 minuti, si trovano poi i campi distanti 30 minuti e così via.

Quindi utilizziamo i risultati già calcolati e memorizzati (i numeri dei campi riempiti) per calcolare ulteriori risultati (i numeri dei campi vicini, ancora vuoti). Questo principio molto generale si chiama *programmazione dinamica*. Di solito è importante l'ordine in cui vengono calcolati i risultati. Anche questo è da considerare con il volo delle api.

Nel compito un'ape vola in 10 minuti solo su, giù, a sinistra o a destra. Questo è un po' insolito, perché in realtà un'ape probabilmente volerebbe anche in diagonale sui campi. Con questa ipotesi più realistica, i campi raggiungibili in mezz'ora sarebbero limitati da un cerchio invece che da un diamante come nel compito.

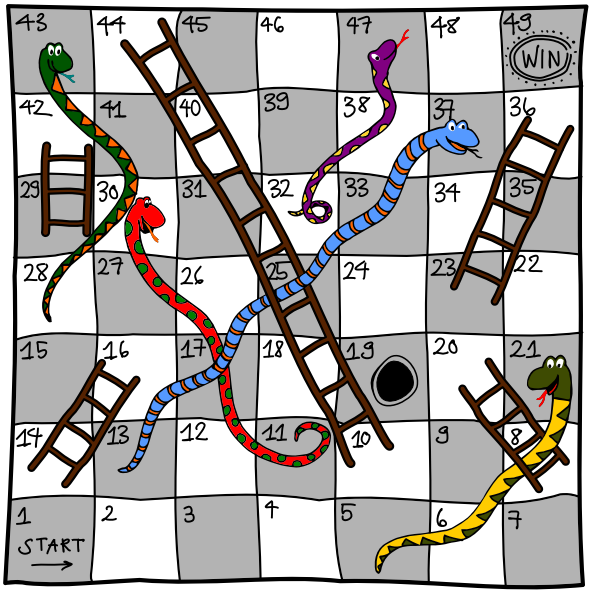
La consueta misura della distanza che porta ad un cerchio è chiamata distanza euclidea. La misura della distanza così come nel compito in cui si è autorizzati a muoversi solo orizzontalmente o verticalmente attraverso i quadrati è chiamata *distanza di Manhattan* (il nome deriva dalle reti stradali a griglia delle città moderne come Manhattan).

Parole chiave e siti web

- Programmazione dinamica: `https://it.wikipedia.org/wiki/Programmazione_dinamica`
- Distanza euclidea: `https://it.wikipedia.org/wiki/Distanza_euclidea`
- Distanza di Manhattan (o geometria del taxi): `https://it.wikipedia.org/wiki/Geometria_del_taxi`

15. Scale e serpenti

Nel gioco delle scale e serpenti tutti i giocatori partono dalla casella 1, e il primo giocatore che raggiunge la casella 49 vince. In ogni turno si tira il dado e si sposta la statuina nel campo corrispondente (tra 1 e 6).



Se si finisce in un campo con la testa di un serpente, si scivola verso il campo con la sua coda. Ma se si finisce ai piedi di una scala, si può salire fino in cima.

Esempio: stai sulla casella 26 e tiri un 3. Puoi passare a 29 e quindi avanzare immediatamente alla casella 42. Nel turno successivo tiri un 5, atterri sulla testa del serpente del campo 47 e devi tornare al campo 32.

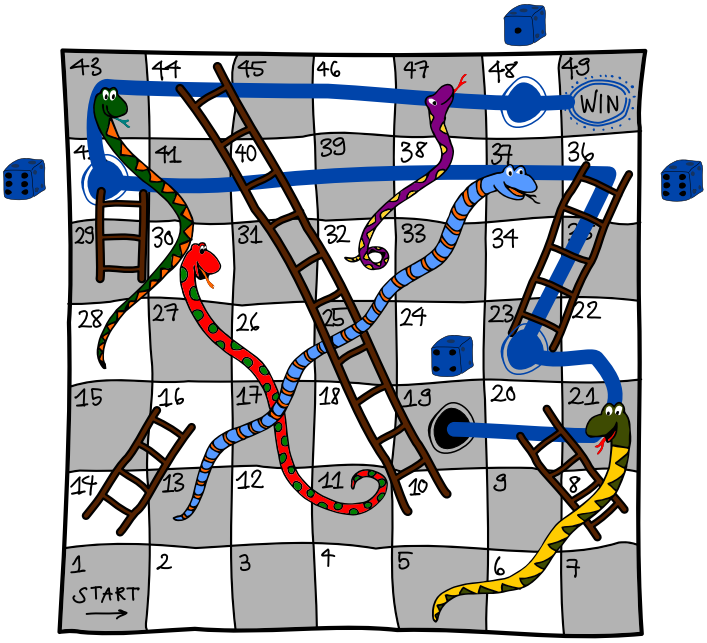
La tua statuina è sul campo 19, di quanti turni hai bisogno minimo per raggiungere il campo 49?

A) 2 turni
B) 3 turni
C) 4 turni
D) 5 turni

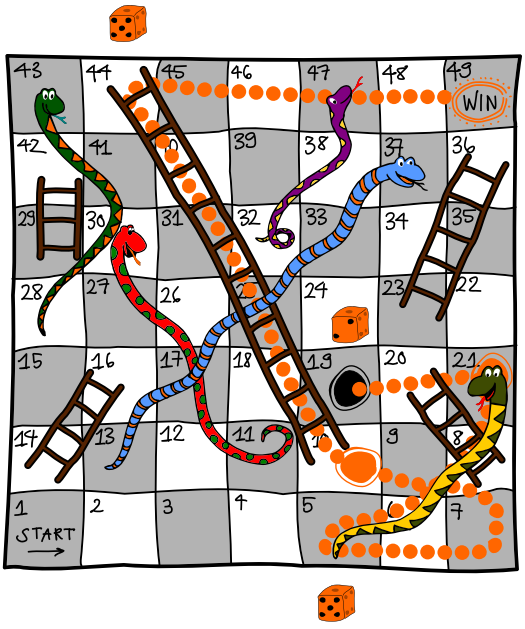
Soluzione

La risposta corretta è B) 3 turni.

Se sei avido e tieni conto solo dei tiri che ti porteranno verso la meta, hai bisogno di almeno 4 turni: Con un 4 arrivi dal 19 al 23 e con la scala al 36, da lì non ci sono più scale e hai bisogno di altri 3 tiri, per esempio 6 – 6 – 1 per raggiungere la meta.



Ma se si accetta una mossa che all'inizio sembra un peggioramento, ci si arriva in 3 turni, con i lanci 2 – 5 – 5. Dal 19 al 21 e attraverso il serpente fino alla casella 5, poi al 10 e fino al 44 e poi fino alla meta.



In 2 turni l'obiettivo non si può raggiungere. Ad un tiro dalla meta si trovano i campi 48, 46, 45, 44 e nessuno di questi campi può essere raggiunto dal 19 in un solo turno.

Questa è l'informatica!

Molti compiti possono essere risolti trovando il cammino più breve tra due punti. Qui, «breve» spesso non ha il significato intuitivo. Qui, per esempio, abbiamo cercato il percorso con il minor numero di tiri e non il percorso che attraversa meno campi. Questo è conosciuto anche dai sistemi di navigazione, che ci propongono il percorso più breve o quello che ci richiede meno tempo. Nelle aziende di logistica, gli stessi dispositivi calcolano il percorso con il pedaggio più basso.

Nell' informatica, le stesse procedure (algoritmi) possono spesso essere utilizzate per compiti molto diversi se sono modellati in modo corrispondente.

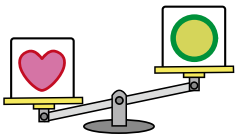
Parole chiave e siti web

- Cammini minimi: `https://it.wikipedia.org/wiki/Algoritmo_di_Dijkstra`
- Scale e serpenti: `https://it.wikipedia.org/wiki/Scale_e_serpenti`

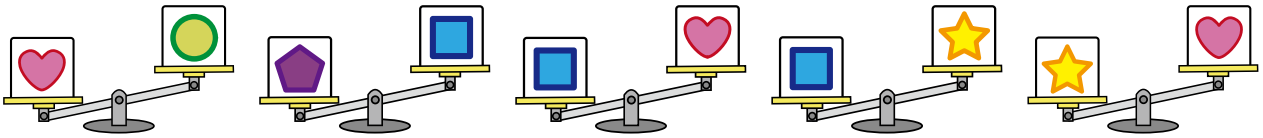
16. Comparazioni pesanti

Cinque scatole sono contrassegnate da cinque diversi simboli:  ,  ,  ,  e  .

Con l'aiuto di una bilancia si comparano due scatole alla volta. La seguente comparazione mostra, ad esempio, che  è più pesante di  :



In totale sono state effettuate cinque comparazioni:



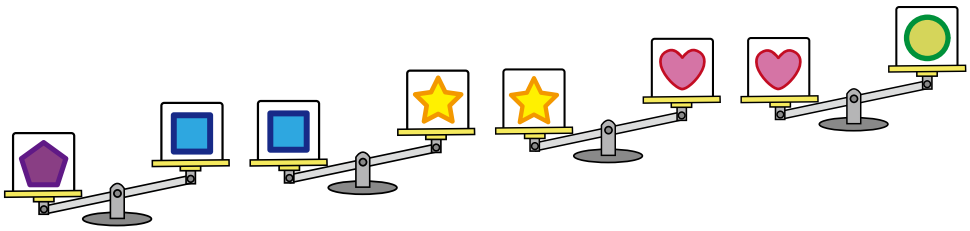
Qual è la scatola più pesante?

A) 
B) 
C) 
D) 
E) 

Soluzione

La scatola C) con il pentagono è la più pesante.

La figura seguente contiene quattro delle cinque comparazioni effettuate e tutte e cinque le scatole.



Così si vede subito: la scatola con il pentagono è più pesante di quella con il quadrato. La scatola con il quadrato è più pesante della scatola con la stella. La scatola con la stella è più pesante di quella con il cuore. E la scatola con il cuore è più pesante della scatola con il cerchio.

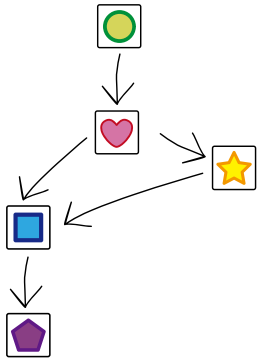
Da questo si può ora concludere che la scatola con il pentagono è più pesante di tutte le altre. Ciò è dovuto ad una speciale proprietà di comparazione dei pesi:

Se A è più pesante di B e B è più pesante di C, allora anche A è più pesante di C. Questa proprietà molto importante si chiama *relazione transitiva*.

A proposito, c'è un modo intelligente per abbreviare questo compito. Poiché si sta cercando la scatola più pesante, è sufficiente cercare la scatola che non è più leggera di qualsiasi altra scatola, e questa è solo la scatola con il pentagono.

Questa è l'informatica!

Fondamentalmente, questo compito riguarda *l'ordinamento* di qualsiasi oggetto. In informatica, per l'ordinamento vengono spesso utilizzati dei *grafi* speciali, che consistono in *nodi* (gli oggetti da ordinare) e *archi* (comparazione di due oggetti). Gli oggetti in questo compito sono le scatole e le comparazioni sono le pesate. Se si disegnano gli archi come frecce che puntano all'oggetto più pesante, il grafo per questo compito si presenta così:



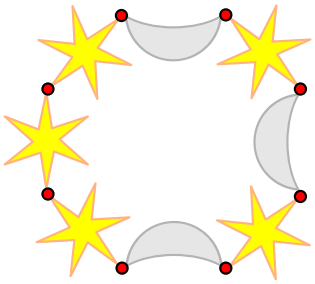
Gli oggetti devono ora essere ordinati in fila, in modo che le frecce partano sempre dagli oggetti più a sinistra verso gli oggetti più a destra. Una tale disposizione è chiamata *ordinamento topologico*. Un ordinamento topologico si ottiene molto semplicemente rimuovendo ripetutamente un nodo dal grafo verso il quale non c'è alcuna freccia puntata e ponendo i nodi rimossi in quest'ordine.

Ma attenzione: non tutti i grafi hanno un ordinamento topologico. Per esempio, non c'è un ordinamento topologico se ci sono tre frecce che puntano in un cerchio.

Parole chiave e siti web

- Relazione transitiva: `https://it.wikipedia.org/wiki/Relazione_transitiva`
- Grafo: `https://it.wikipedia.org/wiki/Grafo`
- Ordinamento topologico: `https://it.wikipedia.org/wiki/Ordinamento_topologico`

17. Braccialetto



Marie vorrebbe avere il braccialetto a destra.

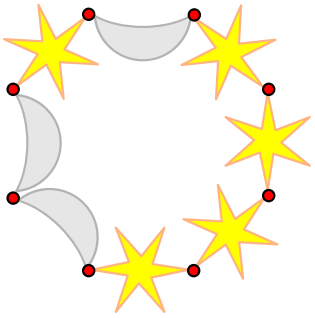
Per questo motivo dà a Jonas le seguenti istruzioni:

- Prendi una stella (★) e una luna (☾) e collegale a formare una coppia. Fallo tre volte in totale, in modo da avere tre coppie.
- Prendi queste tre coppie, girale come vuoi e collegale in una lunga catena.
- Aggiungi altre due stelle ad un'estremità della catena. Ora collega le due estremità della catena per ottenere un braccialetto.

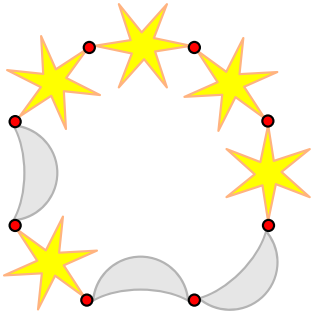
Jonas non ha una foto del braccialetto desiderato. È possibile che ottenga un braccialetto dall'aspetto completamente diverso, anche se Jonas segue esattamente le istruzioni di Marie.

*Uno dei quattro braccialetti **NON** si può ottenere se Jonas segue esattamente le istruzioni di Marie. Quale?*

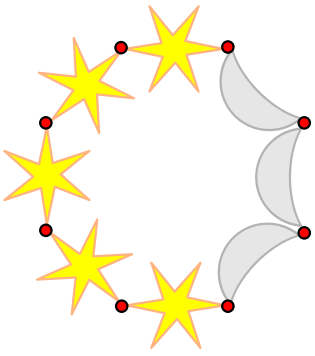
A)



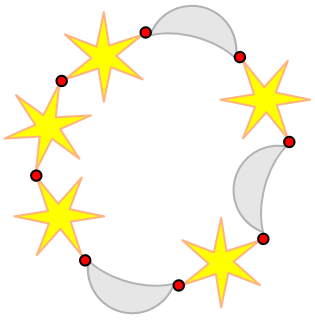
B)



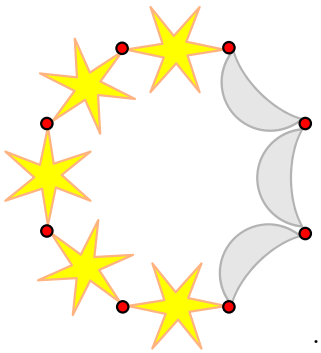
C)



D)

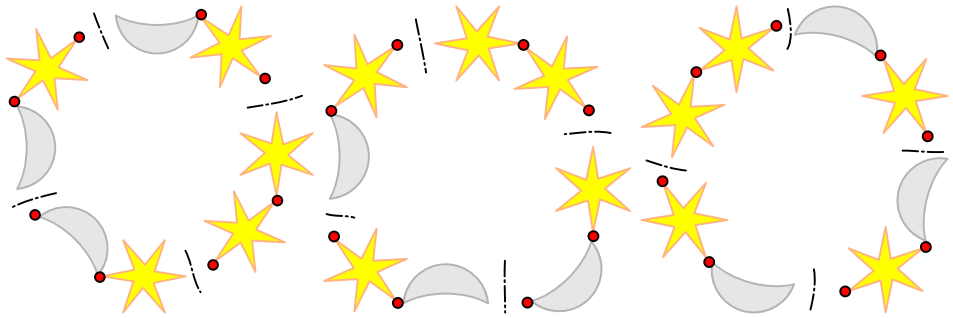


Soluzione

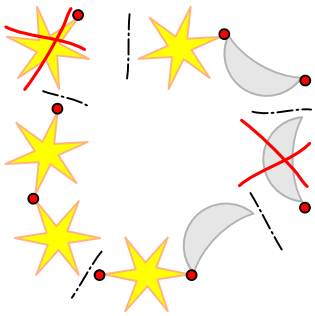
La risposta corretta è C)  .

Solo questo braccialetto non si può ottenere seguendo le istruzioni di Marie.

I braccialetti delle altre tre risposte sono corretti seguendo le istruzioni di Marie. Lo si può vedere, ad esempio, perché ognuno di questi braccialetti può essere diviso in tre coppie stella-luna e una coppia stella-stella, come mostrato nelle foto.



Una luna può essere inserita nel braccialetto solo come parte di una coppia stella-luna. Pertanto, ogni luna ha almeno una stella accanto ad essa. Quindi tre lune di fila come nel braccialetto C non sono possibili. Anche cinque o più stelle di fila sono impossibili.



Questa è l'informatica!

Quando i programmatori danno istruzioni a un computer, è importante che specifichino esattamente ciò che il computer deve fare. Altrimenti si potrebbe ottenere un risultato indesiderato. Per esempio, Marie ha dimenticato di specificare nella sua lista di istruzioni esattamente come le tre coppie stella-luna devono essere collegate tra loro. Nel braccialetto che voleva, una luna è sempre circondata da stelle. Quindi mancava qualcosa, anche se le istruzioni sembrano molto precise. Se ci fosse un

computer che controlla una macchina per fare braccialetti, le istruzioni di Marie non sarebbero abbastanza precise. Fortunatamente, i veri computer di solito si fermano e dicono: «Non so cosa intendi, perché le istruzioni non sono abbastanza chiare».

Nell'informatica ci sono molti meccanismi per descrivere le cose in modo molto preciso. Un meccanismo sono le cosiddette *grammatiche (formali)*. Una grammatica contiene *regole* che descrivono esattamente come creare determinate *parole* (una sequenza di lettere). Ad esempio, è possibile esprimere le istruzioni di Marie in una grammatica come questa:

$$\begin{aligned} &\mathrm{B} \to \mathrm{CSS} && (1)\\ &\mathrm{C} \to \mathrm{PPP} && (2)\\ &\mathrm{P} \to \mathrm{SL} && (3) \end{aligned}$$

Qui B sta per braccialetto, C per catena, P per coppia, S per stella e L per luna. Si inizia con B e poi si possono creare nuove parole applicando le tre regole di sostituzione tutte le volte che si desidera. Lo si fa fino a quando la parola è composta solo dai simboli S e L. Per esempio:

$$\begin{aligned} &\mathrm{B} \Rightarrow \mathrm{CSS} && \text{per regola (1)}\\ &\mathrm{CSS} \Rightarrow \mathrm{PPPSS} && \text{per regola (2)}\\ &\mathrm{PPPSS} \Rightarrow \mathrm{SLPPSS} \Rightarrow \mathrm{SLSLPSS} \Rightarrow \mathrm{SLSLSLSS} && \text{per regola (3)} \end{aligned}$$

Si può considerare che la grammatica di cui sopra corrisponde esattamente alle istruzioni di Marie.

L'informatica non è solo programmazione. Spesso si tratta di descrivere gli oggetti. Un insieme di regole di creazione (la grammatica o le istruzioni di Marie) può essere utilizzato per descrivere esattamente una classe di oggetti (alcune parole o i possibili braccialetti). La classe contiene esattamente quegli oggetti che possono essere creati con le regole.

Parole chiave e siti web

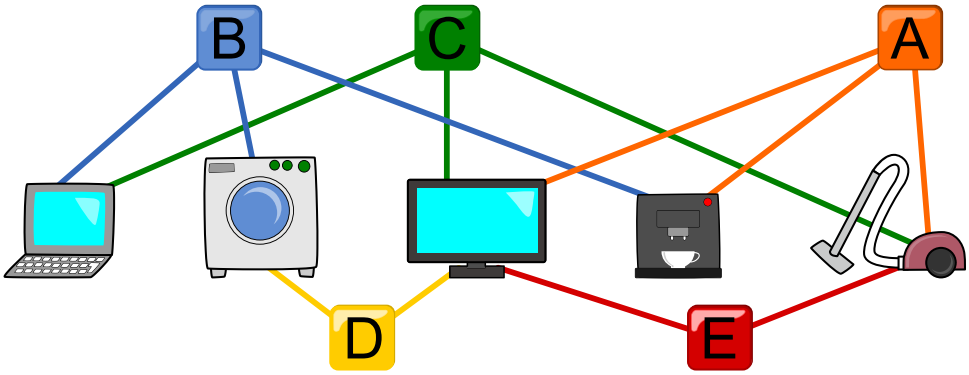
- Grammatica formale: `https://it.wikipedia.org/wiki/Grammatica_formale`

18. Elettrodomestici

Nella casa del castoro Bruno ci sono cinque elettrodomestici (computer, lavatrice, televisione, macchina per il caffè e aspirapolvere) e cinque pulsanti (A, B, C, D ed E) per accendere e spegnere. Tuttavia, il cablaggio è molto insolito. Ogni pulsante è collegato a diversi dispositivi, come mostrato nella figura sotto. Ogni volta che si preme un tasto, esso commuta tutti i dispositivi collegati: Quelli che sono spenti vengono accesi e quelli che sono accesi vengono spenti.

All'inizio tutti gli apparecchi sono spenti. Ad esempio, se si premono i pulsanti A, C ed E, l'aspirapolvere si accende perché il primo pulsante lo accende, il secondo lo spegne e il terzo lo riaccende.

Quali pulsanti deve premere Bruno affinché alla fine si accendano solo il televisore e la macchina del caffè?



Soluzione

Se si premono i pulsanti B, C, D, E (in qualsiasi ordine), si accendono solo il televisore e la macchina del caffè.

Possiamo anche scoprire sistematicamente come accendere e spegnere ogni apparecchio separatamente. Iniziamo con due semplici combinazioni:

- A + E (premendo A ed E) si comanda la macchina del caffè da sola.
- C + E (premendo C ed E) si comanda il computer da solo.

Osserviamo poi che la lavatrice può essere comandata individualmente premendo prima B e poi riportando immediatamente il computer e la macchina da caffè al punto di partenza premendo A + E e C + E. Così, tutto sommato, la lavatrice è controllata individualmente da B + A + E + C + E. Qui E appare due volte. Premere due volte lo stesso interruttore è come non averlo premuto affatto. Pertanto, la lavatrice può essere comandata anche singolarmente da B + A + C. Con questo metodo si ottiene la seguente lista di combinazioni di pulsanti per il controllo dei singoli apparecchi:

- Computer: C + E
- Macchina del caffé: A + E
- Lavatrice: A + B + C
- Televisione: A + B + C + D
- Aspirapolvere: A + B + C + D + E

Per accendere la televisione e la macchina del caffè, dobbiamo quindi premere A + B + C + D + A + E, il che semplifica a B + C + D + E, in quanto le due A si annullano a vicenda.

Questa è l'informatica!

Il sistema di dispositivi e pulsanti per l'accensione e lo spegnimento può essere modellato come un cosiddetto *automa a stati finiti*. Questo avviene come segue.

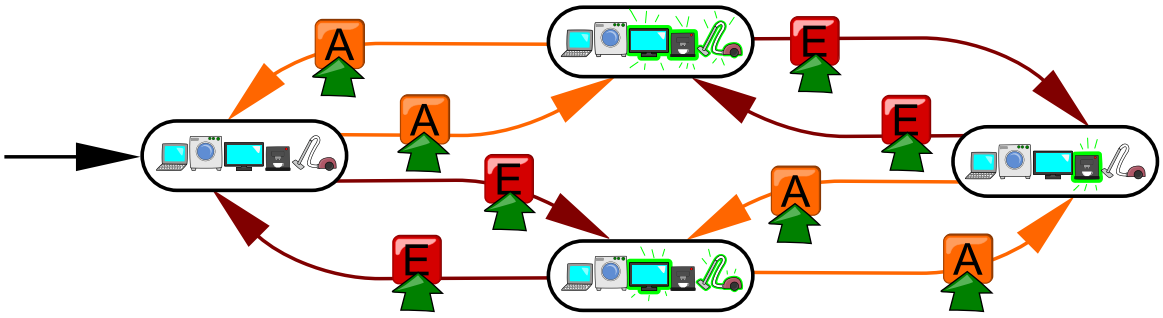
Il sistema dei cinque dispositivi ha molti *stati* diversi. Per esempio, uno stato è quando è acceso solo il televisore. Un altro stato è quando tutti gli apparecchi sono spenti (poiché tutti gli apparecchi sono spenti all'inizio, lo chiamiamo lo *stato iniziale*). E un altro stato è quello in cui sono accese solo la TV e la macchina del caffè (nel nostro esempio questo è lo *stato finale* perché è lo stato che vogliamo).

Premendo un pulsante si sposta il sistema da uno stato all'altro.

Per esempio: Se il sistema è nello stato iniziale, premendo E si passa allo stato in cui sono accesi solo il televisore e l'aspirapolvere. Un tale cambiamento di stato è anche chiamato *transizione*.

Se si disegnano gli stati del sistema come cerchi e le transizioni come frecce, si ottiene un'immagine come quella qui sotto (per ragioni di spazio, sono disegnati solo quattro stati e solo le transizioni tra di essi.) Lo stato iniziale è contrassegnato da una freccia speciale. In informatica questo si chiama

automa a stati finiti (a proposito, un automa a stati finiti è semplicemente un grafo speciale; gli stati sono i *nodi* e le transizioni sono gli *archi*). Nella foto, ora possiamo facilmente vedere in quale stato ci troviamo quando vengono premuti diversi pulsanti.



Il compito consiste nel passare dallo stato iniziale (tutti i dispositivi spenti) allo stato di destinazione (solo TV e macchina del caffè accesa). Quindi si tratta di trovare un modo per passare dallo stato iniziale allo stato di destinazione. Trovare percorsi nei grafi è un compito fondamentale dell'informatica.

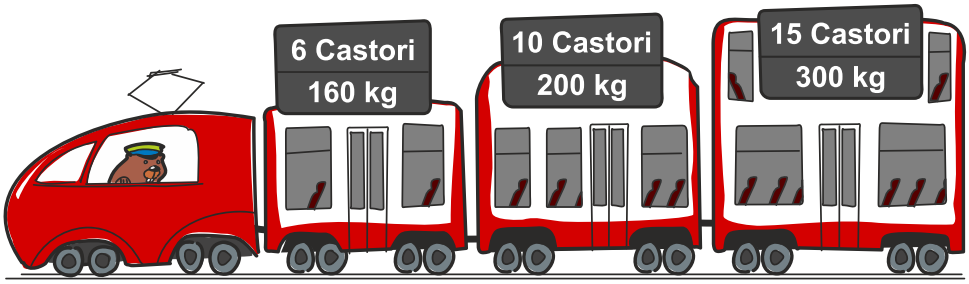
Parole chiave e siti web

- Automa a stati finiti: `https://it.wikipedia.org/wiki/Automa_a_stati_finiti`

19. Viaggio in treno

Otto famiglie di castori vorrebbero viaggiare con il treno. Le famiglie sono elencate con il numero dei loro membri e il peso del loro bagaglio nella seguente tabella:

Nome della famiglia	Numero di membri	Peso del bagaglio in kg
Ammann	3	50
Bernasconi	4	80
Camenzind	5	110
Donetta	4	80
Emery	2	40
Favre	3	70
Gerber	6	130
Huber	5	100



L'immagine mostra per ogni carrozza quanti castori e quanti bagagli possono essere trasportati al massimo. Inoltre, le famiglie devono viaggiare con i loro bagagli in una carrozza e non possono dividersi.

Qual è il numero massimo di famiglie di castori che possono viaggiare con il treno?

A) 1 famiglia di castori
B) 2 famiglie di castori
C) 3 famiglie di castori
D) 4 famiglie di castori
E) 5 famiglie di castori
F) 6 famiglie di castori
G) 7 famiglie di castori
H) 8 famiglie di castori

Soluzione

Possono viaggiare al massimo 7 famiglie di castori. Una delle possibili distribuzioni è:

	Nome della famiglia	Numero di membri	Peso del bagaglio in kg
	Gerber	6	130
	Totale:	**6**	**130**
	Ammann	3	50
	Camenzind	5	110
	Emery	2	40
	Totale:	**10**	**200**
	Bernasconi	4	80
	Donetta	4	80
	Huber	5	100
	Totale:	**13**	**260**

Le 8 famiglie di castori insieme costituiscono un totale di 32 passeggeri, mentre solo 31 posti a sedere sono disponibili sul treno. È quindi impossibile che tutte le 8 famiglie di castori possano viaggiare sul treno.

Questa è l'informatica!

L'informatica si occupa spesso di *problemi di ottimizzazione* dove le risorse limitate - come in questo caso lo spazio e la capacità di peso - devono essere utilizzate al meglio. In realtà, naturalmente, nessun passeggero dovrebbe essere lasciato indietro, ma la compagnia ferroviaria può calcolare, ad esempio, che è meglio trasportare i singoli passeggeri comodamente in taxi piuttosto che utilizzare un treno completo che poi viaggia quasi vuoto.

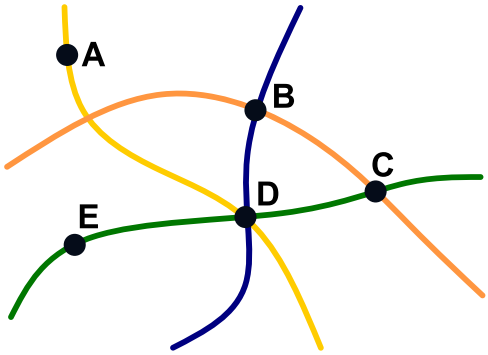
Compiti di questo tipo sono noti come *problemi dello zaino*. A volte tali problemi possono essere ridotti in modo tale da poter essere risolti con l'aiuto della *programmazione dinamica*, cioè individuando prima le possibili soluzioni parziali che possono poi essere ulteriormente sviluppate in una soluzione globale. In molti casi, tuttavia, i compiti appartengono ai cosiddetti *problemi NP-completi*, il che significa che attualmente non c'è soluzione migliore che provare in modo intelligente. La maggioranza delle persone avrà risolto questo compito proprio in questo modo.

Parole chiave e siti web

- Problema dello zaino: `https://it.wikipedia.org/wiki/Problema_dello_zaino`
- Programmazione dinamica: `https://it.wikipedia.org/wiki/Programmazione_dinamica`
- NP-completo: `https://it.wikipedia.org/wiki/NP-completo`

20. Rete ferroviaria

Questa è una mappa di 5 città e 4 linee ferroviarie. I punti neri sono le città, le linee colorate sono linee ferroviarie.

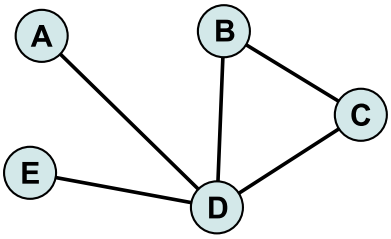


Un diagramma dovrebbe rappresentare questa mappa in modo tale che:

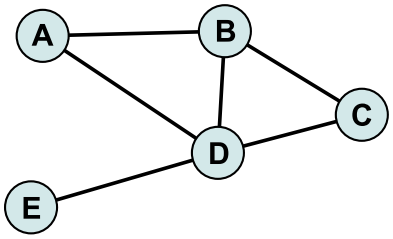
- le città sono rappresentate da cerchi, e
- due città sono collegate da una linea solo quando si trovano sulla stessa linea ferroviaria.

Quale diagramma visualizza correttamente la mappa?

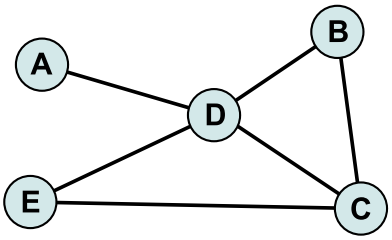
A)



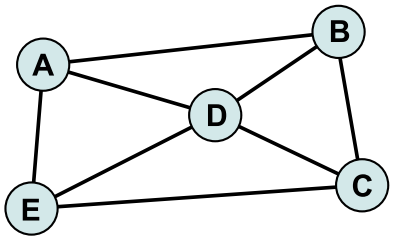
B)



C)

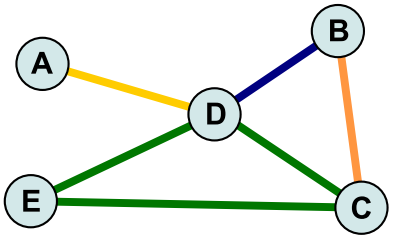


D)



Soluzione

La risposta corretta è C).



Un'analisi più approfondita della mappa dimostra che:

- Le città A e D si trovano insieme sulla linea ferroviaria gialla,
- le città B e C sono insieme sulla linea ferroviaria arancione,
- le città B e D si trovano insieme sulla linea ferroviaria blu, e
- le città C, D ed E si trovano insieme sulla linea ferroviaria verde.

Tutte le altre risposte sono sbagliate:

- Nella risposta A, manca la linea tra le città C ed E, che deve esistere a causa della linea ferroviaria verde.
- La risposta B ha lo stesso problema della risposta A e in aggiunta c'è una linea tra le città A e B, anche se non sono insieme sulla stessa linea ferroviaria.
- Nella risposta D, ci sono due linee dalla città A alla città B e dalla città A alla città E, anche se la città A non è su una linea ferroviaria comune con la città B o la città E.

I due punti seguenti meritano un'attenzione particolare:

- Anche se si può andare dalla città A alla città B se si utilizzano varie linee ferroviarie, le due città non sono sulla stessa linea ferroviaria.
- Anche se c'è una terza città sulla linea ferroviaria verde tra C ed E, C ed E sono ancora sulla stessa linea ferroviaria.

Questa è l'informatica!

Ci sono molti modi diversi di rappresentare la realtà. Ad esempio, la mappa qui sopra con le linee ferroviarie colorate è già una rappresentazione piuttosto astratta della situazione reale. Un tipo di rappresentazione molto importante è un *grafo* - un diagramma costituito da *nodi* (piccoli cerchi) e *archi* (linee tra i nodi). Questo tipo di rappresentazione viene utilizzato nella soluzione.

Molte cose diventano più facili se si sceglie un buon metodo di rappresentazione. Pertanto, è importante conoscere molte modalità di visualizzazione durante la programmazione. Spesso non è possibile affermare che una modalità di visualizzazione sia migliore dell'altra. A seconda dell'applicazione, l'uno o l'altro è più adatto. Ad esempio, il grafo della soluzione è pratico perché si può vedere direttamente che si può andare da C a E con una sola linea ferroviaria. Rispetto alla mappa, tuttavia,

si perde l'informazione che con questa linea ferroviaria si passa per la città D sulla strada dalla città C alla città E.

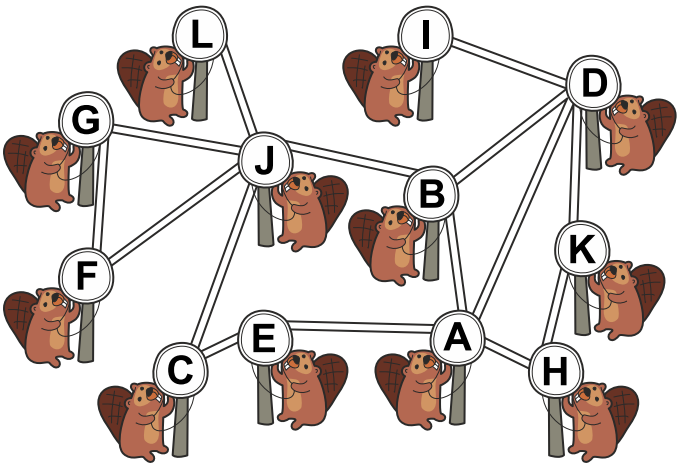
Parole chiave e siti web

- Grafo: `https://it.wikipedia.org/wiki/Grafo`
- Teoria dei grafi: `https://it.wikipedia.org/wiki/Teoria_dei_grafi`

21. Rete di comunicazione

Ai castori piace diffondere notizie tra di loro. A tale scopo utilizzano la rete di comunicazione qui sotto. Quando un castoro riceve un nuovo messaggio, lo inoltra a tutti coloro con cui è collegato da un canale di comunicazione diretta (una linea bianca). L'invio dei messaggi si effettua a turni. C'è sempre un turno tra l'invio e la ricezione.

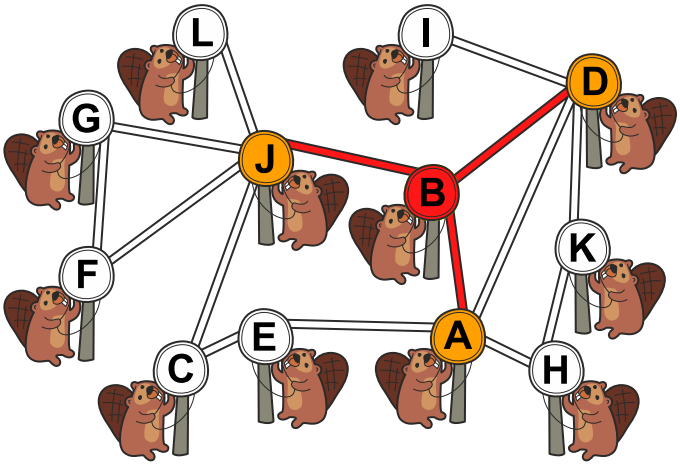
Da quale castoro un messaggio raggiunge tutti gli altri castori più velocemente, cioè nel minor numero di turni?



Soluzione

La risposta corretta è il castoro B. Può diffondere un messaggio a tutti gli altri castori in due turni.

Nel primo turno, il castoro B invia il messaggio ai suoi vicini, cioè i castori A, D e J collegati ad un canale di comunicazione diretta. L'immagine sottostante mostra chi ha il messaggio dopo questo primo turno.

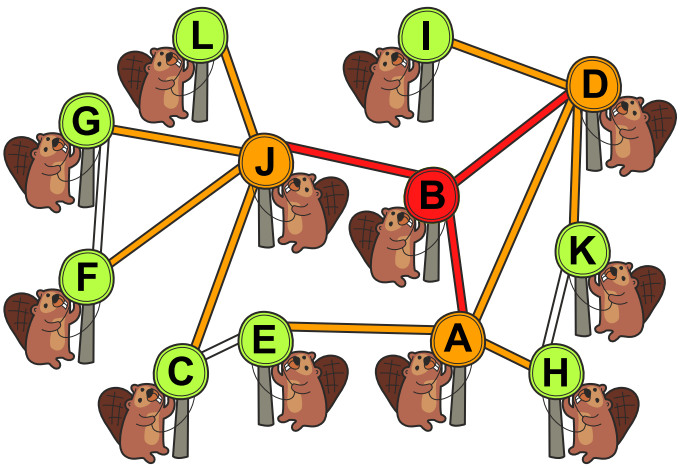


Nel secondo turno i castori A, D e J inviano il messaggio ai loro vicini: - Il castoro A invia il messaggio ai castori E e H;

- Il castoro D invia il messaggio ai castori I e K;
- Il castoro J invia il messaggio ai castori C, F, G e L.

Inoltre, il castoro B riceve il messaggio tre volte, perché anche lui è un vicino dei tre castori A, D e J. Poiché questo non è un messaggio nuovo per lui, il castoro B non lo invierà nei prossimi turni. Anche i castori A e D si invieranno di nuovo il messaggio attraverso il loro canale di comunicazione diretta, ma dopo di che non lo invieranno più perché non è più nuovo per loro.

L'immagine sottostante mostra la situazione dopo il secondo turno.



Così la notizia ha raggiunto tutti i castori in soli due turni.

Non c'è un modo più veloce, perché altrimenti un castoro dovrebbe essere collegato a tutti gli altri castori con una linea bianca per inviare il messaggio direttamente a tutti gli altri castori in un unico turno.

Il castoro B è anche l'unico dal quale un messaggio raggiunge tutti gli altri castori in soli due turni: per i castori C, E, F, G, H, J e L, il castoro I non sarebbe raggiungibile in due turni. E per i castori A, D, E, E, H, I e K, il castoro L non può essere raggiunto in due turni.

Questa è l'informatica!

La rete di comunicazione dei castori può essere descritta da un *grafo*. Ogni castoro si trova in un cosiddetto *nodo*, che in questo caso è nominato con una lettera. Le linee bianche sono chiamate *archi*, ognuna di esse collega due nodi. I messaggi si diffondono nella rete di comunicazione attraverso turni *sincronizzati*, cioè tutti i castori inviano contemporaneamente. In un turno, ogni castoro invia nuovi messaggi a tutti i suoi vicini. Quello che i castori fanno qui è quello che gli informatici chiamano *broadcasting* (inglese per «trasmettere») in una *rete di comunicazioni*. Nel compito di cui sopra, abbiamo studiato la velocità con cui una tale trasmissione può essere completata, cioè la velocità con cui un nuovo messaggio può raggiungere tutti i partecipanti.

Un compito ancora più complesso è quello di progettare le reti in modo tale che sia possibile una trasmissione veloce da tutti i nodi, ma con un numero di connessioni non troppo elevato. Il nodo del ricercato castoro B è chiamato poi il centro del grafo. In astratto, il centro è un nodo che riduce al minimo la distanza dai nodi più lontani. Non c'è quindi nessun altro nodo che avrebbe una distanza minore rispetto a tutti gli altri nodi. Nel presente compito c'è un solo centro. Tuttavia, a seconda del grafo, possono esserci diversi nodi, in modo che ognuno di essi minimizzi la distanza dai nodi più lontani da esso; quindi, un grafo può avere diversi centri.

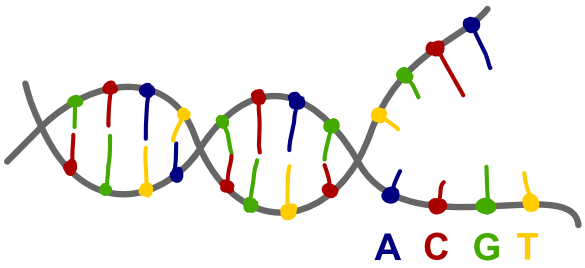
Trovare un centro non è sempre facile come in questo compito. Per prima cosa, potrebbe essere che ci vogliano diversi turni per il trasferimento tra alcuni nodi direttamente collegati. D'altra parte, i grafi possono essere semplicemente molto più grandi e complessi. Per tali grafi, si potrebbe, ad esempio, utilizzare l'algoritmo di Floyd-Warshall per trovare in modo efficiente un centro.

Parole chiave e siti web

- Grafo: `https://it.wikipedia.org/wiki/Grafo`
- Algoritmo di Floyd-Warshall: `https://it.wikipedia.org/wiki/Algoritmo_di_Floyd-Warshall`

22. Sequenza di DNA

Il nostro materiale genetico è immagazzinato in sequenze di DNA. Una sequenza di DNA è essenzialmente una sequenza di basi che si presentano nei quattro tipi A, C, G e T.



Consideriamo i seguenti tre tipi di mutazioni:

Tipo di mutazione	**Descrizione**	**Esempio**
Sostituzione	Una singola base viene sostituita da un'altra.	AT**G**GT → AT**A**GT
Cancellazione	Una singola base viene eliminata senza sostituzione.	ATG**G**T → ATGT
Inserimento	Una singola base è inserita da qualche parte.	ATGGT → A**C**TGGT

*Esattamente una delle quattro sequenze di DNA seguenti **non** può essere creata se la sequenza GTATCG subisce tre mutazioni. Qual è?*

A) GCAATG
B) ATTATCCG
C) GAATGC
D) GGTAAAC

Soluzione

La risposta corretta è D) GGTAAAC.

Il modo migliore per ottenere questa risposta è la procedura di esclusione, perché 3 mutazioni sono sufficienti per tutte le altre sequenze.

Risposta A: GTATCG ⇒ G**C**ATCG ⇒ GCA**A**CG ⇒ GCAATG
Risposta B: GTATCG ⇒ **A**TATCG ⇒ AT**T**ATCG ⇒ ATTAT**C**CG
Risposta C: GTATCG ⇒ G**A**ATCG ⇒ GAAT**G**G ⇒ GAATG**C**

Invece, sono necessarie 4 mutazioni per ottenere la sequenza dalla risposta D, per esempio le seguenti:

GTATCG ⇒ G**G**TATCG ⇒ GGT**A**ATCG ⇒ GGTAA**A**CG ⇒ GGTAAAC

Non è facile dimostrare che meno mutazioni non sono sufficienti.

Questa è l'informatica!

La rappresentazione di informazioni con *stringhe di caratteri* (sequenze di lettere) e il lavoro con esse è un compito centrale dell'informatica.

Una domanda importante è quanto le due stringhe di caratteri differiscano l'una dall'altra. Esistono diversi metodi per misurare la differenza tra due stringhe. Un metodo di misura frequentemente usato è la cosiddetta *distanza di Levenshtein*, che è definita dai tre *tipi di mutazioni* sopra descritti: la distanza di Levenshtein tra due stringhe è il numero minimo di mutazioni che può essere usato per convertire una stringa nell'altra.

Il consueto algoritmo per il calcolo della distanza Levenshtein tra due parole si basa sulla *programmazione dinamica*: qui le distanze Levenshtein tra i prefissi sempre più lunghi delle due parole vengono scritte in una tabella fino a quando alla fine i due prefissi corrispondono alle parole intere e il risultato può essere letto.

Se la correttezza dell'algoritmo è dimostrata, si può calcolare che la distanza di Levenshtein tra la sequenza di DNA originale e quella della risposta D) è esattamente 4. Questo dimostra che meno mutazioni non sono sufficienti.

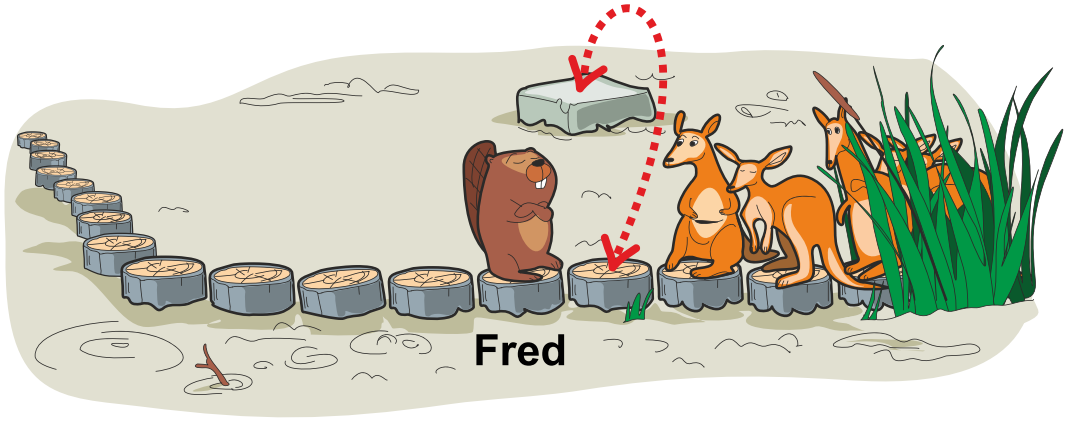
Parole chiave e siti web

- Distanza di Levenshtein: `https://it.wikipedia.org/wiki/Distanza_di_Levenshtein`

23. Il castoro testardo

Il castoro Fred incontra i canguri su un percorso di ceppi di albero. Il percorso è piuttosto stretto, così che lui e i canguri non possano passare allo stesso tempo. Ma c'è uno specifico ceppo di albero dal quale i canguri possono saltare su una pietra e da lì tornare a questo ceppo, come mostrato nella foto. Solo un animale alla volta può stare su ogni ceppo di albero e sulla pietra.

Fred vuole andare avanti. È abbastanza testardo e disposto a tornare indietro di un ceppo solo 10 volte al massimo. In avanti, invece, può andare tutte le volte che vuole.



Qual è il numero massimo di canguri che Fred può far passare?

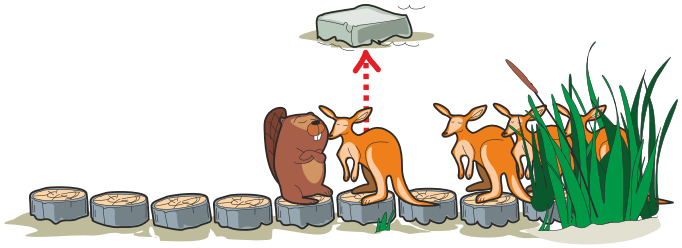
A) Più di 10 canguri.
B) Esattamente 10 canguri.
C) Esattamente 6 canguri.
D) Esattamente 4 canguri.
E) Meno di 4 canguri.
F) È impossibile dirlo con certezza.

Soluzione

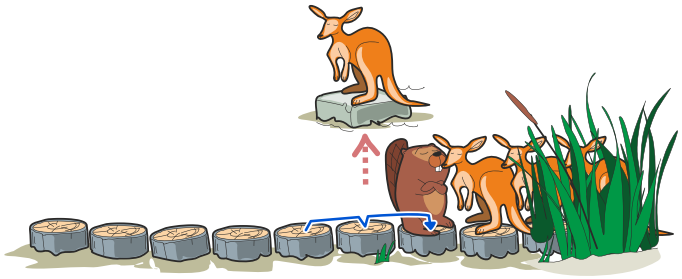
Fred può far passare un massimo di 6 canguri.

Un canguro passa Fred come segue:

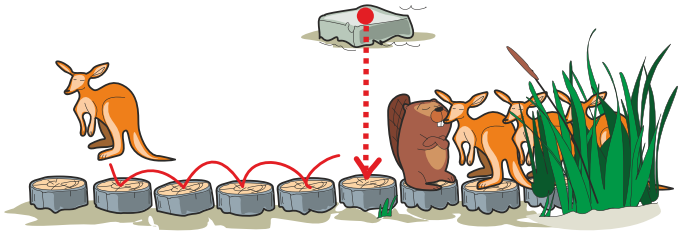
Il canguro salta sulla pietra.



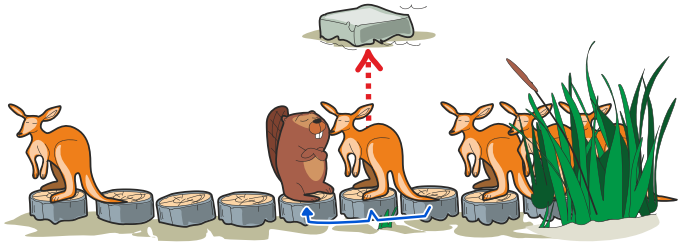
Fred cammina due ceppi d'albero in avanti.



Il canguro salta indietro e continua il suo cammino.



Se Fred ora torna indietro di due ceppi, è tornato alla posizione di partenza e può ripetere la procedura per far passare un altro canguro.



Poiché torna indietro di un massimo di 10 ceppi, può farlo cinque volte e insieme al primo canguro può far passare un massimo di 6 canguri.

Questa è l'informatica!

Nell' informatica i compiti vengono risolti, tra l'altro, con l'ausilio di algoritmi: seguendo semplici *istruzioni* e *comandi* che vengono eseguiti passo dopo passo - proprio come «Fred cammina un ceppo di albero in avanti» o «un canguro salta su una pietra».

In una *iterazione* (o «*loop*» in inglese), le sequenze di istruzioni possono essere ripetute. In questo modo, i compiti uniformi possono essere eseguiti in modo affidabile più volte. Di solito è vantaggioso creare la stessa situazione ad ogni passaggio d'iterazione - la cosiddetta *invariante*. Nel nostro caso

Fred deve tornare alla sua posizione di partenza più e più volte, in modo che la stessa procedura funzioni di nuovo per il canguro successivo.

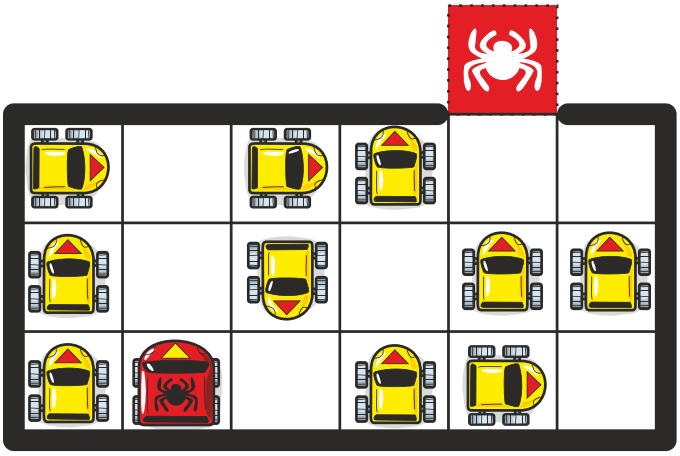
Parole chiave e siti web

- Algoritmo: `https://it.wikipedia.org/wiki/Algoritmo`
- `https://it.wikipedia.org/wiki/Programmazione_strutturata`
- Iterazione: `https://it.wikipedia.org/wiki/Struttura_di_controllo#Iterazione`

24. Auto del ragno

11 auto parcheggiano in una piazza circondata da muri con un'uscita. Ogni auto ha le seguenti possibilità di movimento:

- Un campo in avanti
- Un campo all'indietro
- Un quarto di giro a destra o a sinistra nel campo dove si trova



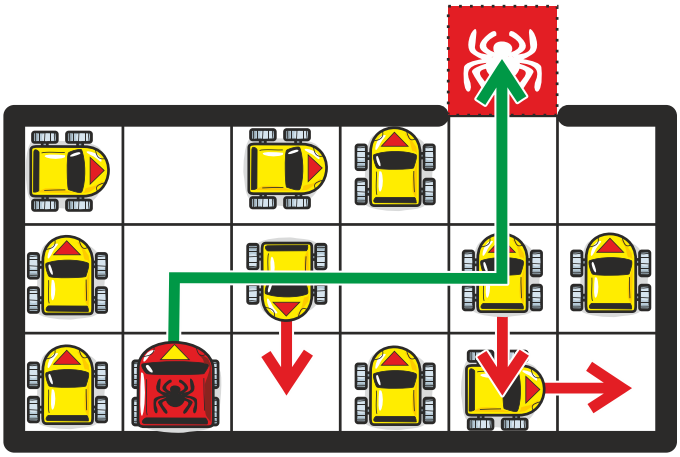
Un'auto può anche eseguire diversi spostamenti. Solo un'auto può essere presente su ogni campo alla volta.

Quanti spostamenti delle auto in totale sono necessari per portare l'auto del ragno rosso nel campo del ragno rosso?

A) 9 spostamenti
B) 11 spostamenti
C) 13 spostamenti
D) 15 spostamenti

Soluzione

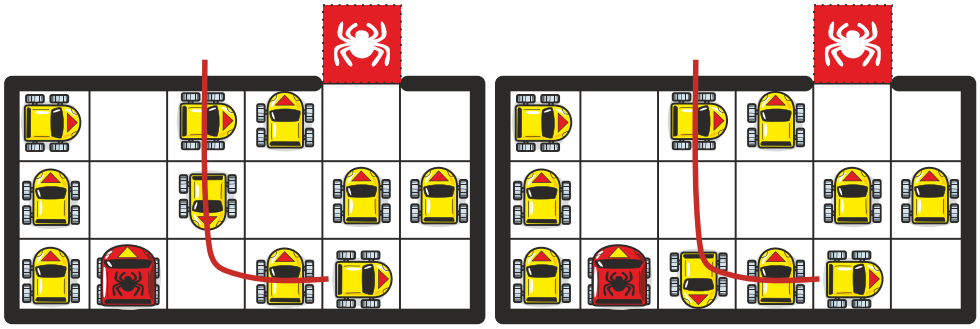
La risposta corretta è: B) 11 spostamenti. L'immagine mostra gli 11 spostamenti per portare l'auto del ragno nel campo del ragno rosso:



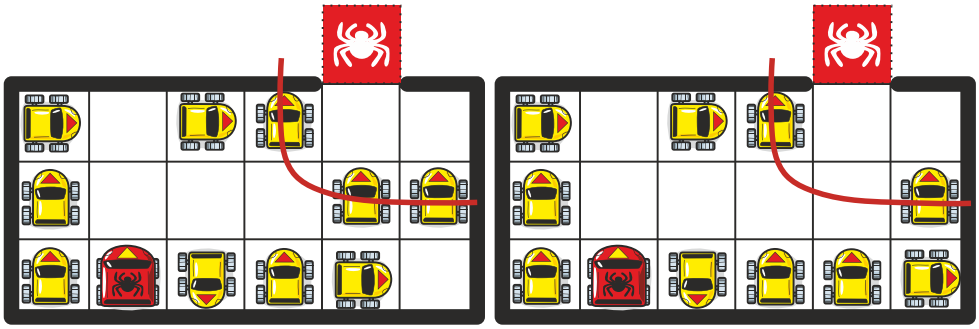
Deve ancora essere dimostrato che 11 è il numero minimo di spostamenti richiesto.

Per questo si parte dal presupposto che l'auto del ragno sia l'unica auto sul percorso. Per arrivare al campo del ragno rosso all'esterno, la macchina del ragno deve spostarsi 3 volte verso l'alto e 3 volte verso destra, deve anche girare 2 volte. Anche se questo può essere realizzato in modi diversi, sono necessari almeno $3 + 3 + 2 = 8$ spostamenti. Ma l'auto del ragno non è l'unica auto sulla piazza e servono più spostamenti per liberare la via.

Prima dobbiamo trovare una via attraverso la barricata a forma di L nella foto successiva. Questo può essere fatto in un unico spostamento come segue:



Poi dobbiamo trovare una via attraverso una seconda barricata a forma di L. Questa barricata non può essere aperta con 1 solo spostamento, ma 2 sono sufficienti come indicato di seguito.



Pertanto il numero minimo di spostamenti è di $8 + 1 + 2 = 11$ spostamenti.

Questa è l'informatica!

Spesso è molto difficile dimostrare che una soluzione trovata è ottimale. Spesso si scopre se esiste o meno una soluzione migliore solo esaminando tutte le soluzioni possibili. Questo metodo si chiama *forza bruta* (in inglese *brute force*) o *ricerca esaustiva* (in inglese *exhaustive search*), perché tutte le possibilità sono esaurite. Anche se questo metodo di solito non è praticabile a mano, per il computer è spesso una strategia facile da implementare.

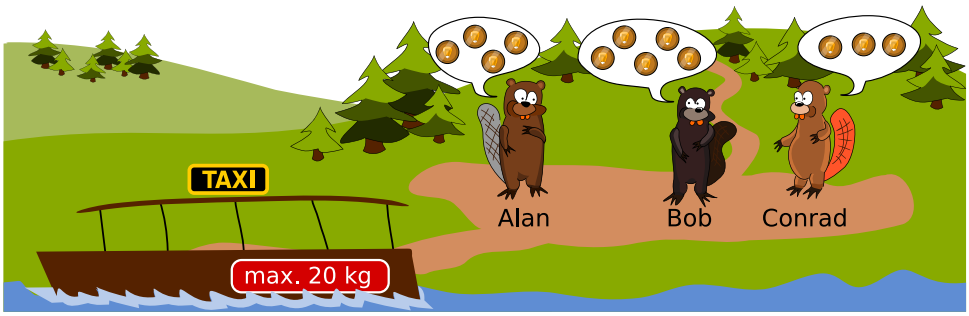
Ma a volte ci sono così tante soluzioni diverse che anche un computer è sopraffatto da ciò. In questi casi si deve cercare una strategia più adeguata. Ad esempio, vengono spesso utilizzati *algoritmi greedy* (dall'inglese per *avido*) o il principio «*branch and bound*».

Il compito è una variante del gioco *Rush Hour*. Il classico gioco per computer *Sokoban* ha anche molte somiglianze.

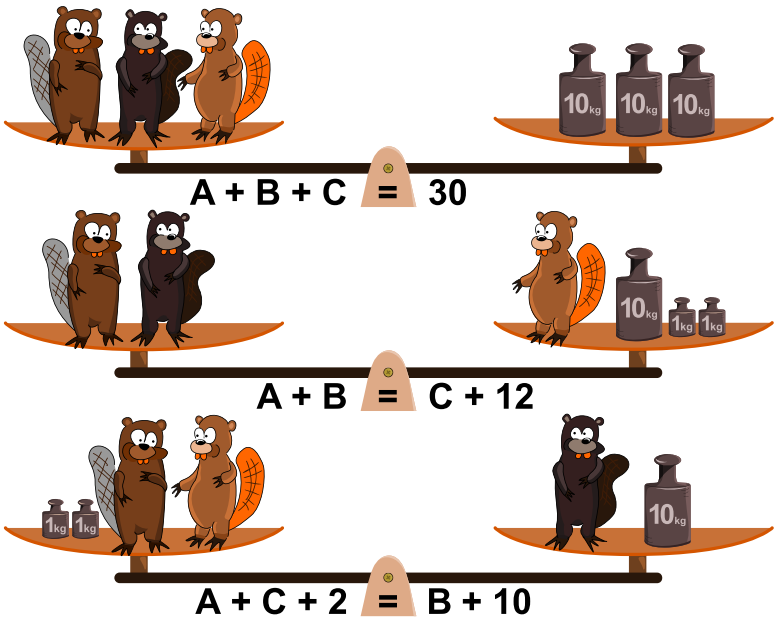
Parole chiave e siti web

- Forza bruta: `https://it.wikipedia.org/wiki/Metodo_forza_bruta`
- Branch and bound: `https://it.wikipedia.org/wiki/Branch_and_bound`
- Algoritmo greedy: `https://it.wikipedia.org/wiki/Algoritmo_greedy`
- Rush Hour

25. Taxi acquatico



I tre castori Alan, Bob e Conrad vogliono prendere un taxi acquatico. C'è solo un taxi acquatico. Alan pagherebbe 4 talleri (4×), Bob invece 5 talleri (5×) e Conrad solo 3 talleri (3×). Il taxi può trasportare un massimo di 20 kg. Pertanto il tassista effettua le seguenti pesate:



Quali castori trasporta il tassista se vuole guadagnare il più possibile?

A) Solo Bob
B) Alan e Bob
C) Bob e Conrad
D) Alan e Conrad
E) Tutti e tre: Alan, Bob e Conrad

Soluzione

La risposta corretta è: C) Bob e Conrad

Per poter elencare e poi valutare tutte le possibili soluzioni, dobbiamo prima sapere quanto pesa ogni castoro.

Sappiamo che tutti e tre insieme pesano 30 kg e quindi non tutti possono essere presi dal tassista. Se mettiamo di nuovo una copia di C(onrad) sul lato destro e sinistro della seconda bilancia, otteniamo $A + B + C = 30$ kg a sinistra e $C + C + 12$ kg a destra. Pertanto deve essere applicato $2C = 18$ kg e quindi $C = 9$ kg.

Se mettiamo di nuovo una copia di B(ob) sul lato destro e sinistro della terza bilancia, otteniamo $A + B + C + 2$ kg $= 32$ kg a sinistra e $2B + 10$ kg a destra. Quindi $2B = 22$ kg e quindi $B = 11$ kg.

Poiché $A + B + C = 30$ kg, A deve essere di 10 kg.

Così il tassista può:

- Trasportare Alan e Conrad, e guadagnare $4 + 3 = 7$ talleri.
- Trasportare Bob e Conrad, e guadagnare $5 + 3 = 8$ talleri.
- Trasportare Alan e Bob, allora lui guadagnerebbe 9 talleri, ma purtroppo i due insieme pesano 21 kg e quindi sovraccaricano il taxi d'acqua.

Pertanto la risposta corretta è C).

Ma questo non è l'unico modo per determinare il peso dei castori. Così come si sarebbe potuto sostituire $A + B$ con $C + 12$ sulla prima bilancia a sinistra nel primo passo. Si ottengono quindi $2C + 12$ kg sul lato sinistro, pari a 30 kg. Così si conclude ancora una volta che $C = 9$ kg.

Più formalmente, le tre pesate possono essere scritte come un sistema di equazioni:

I. $A + B + C = 30$ kg
II. $A + B - C = 12$ kg
III. $A - B + C = 8$ kg

Queste equazioni possono poi essere sottratte l'una dall'altra. Così la differenza I – II fornisce l'equazione:

$$2C = 18 \text{ kg} \rightarrow C = 9 \text{ kg}$$

La differenza I. – III. dà:

$$2B = 22 \text{ kg} \rightarrow B = 11 \text{ kg}$$

Da I. segue quindi $A = 10$ kg.

Questa è l'informatica!

Tutti i problemi di ottimizzazione discreti di NP possono essere rappresentati nel linguaggio delle equazioni lineari e delle disuguaglianze. Le equazioni e le disuguaglianze sono le cosiddette restrizioni, che i valori delle variabili devono soddisfare. Si ottimizza quindi il valore di una funzione delle variabili, mentre le restrizioni devono essere rispettate. Nel presente lavoro abbiamo tre variabili booleane, x_A, x_B, x_C. Si $x_A = 1$, il castoro A viene preso a bordo, altrimenti $x_A = 0$. Si ottimizza la funzione lineare $4x_A + 5x_B + 3x_C$, cercando il valore massimo. L'unica restrizione è:

$$Peso(A) \cdot x_A + Peso(B) \cdot x_B + Peso(C) \cdot x_C \leq 20.$$

Il compito può essere formulato in modo completo solo attraverso la determinazione del peso dei castori. Questo esempio di problema è un caso del *problema dello zaino*. Si cerca di mettere più valore possibile nello zaino senza superare il peso totale.

Solo 80 anni fa, tali questioni erano compito dei matematici, ma man mano che si rendevano disponibili computer sempre più potenti, sono stati sviluppati metodi di soluzione per tali problemi (ad esempio i metodi «*branch-and-bound*» o «*cutting-plane*»). Oggi questi metodi di soluzione vengono utilizzati, ad esempio, per ottimizzare la produzione, nella logistica o nelle reti di trasporto locale.

Tuttavia, la soluzione dei problemi di ottimizzazione nella pratica è ancora un compito difficile, che richiede una modellazione abile e algoritmi appositamente sviluppati, a seconda delle dimensioni e della struttura del problema. Spesso vengono combinati diversi metodi di soluzione.

Parole chiave e siti web

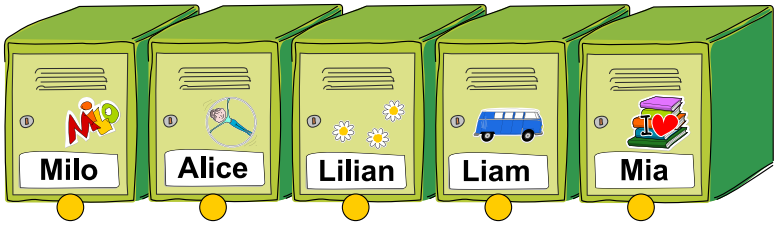
- Programmazione lineare: `https://it.wikipedia.org/wiki/Programmazione_lineare`
- Restrizione di una funzione: `https://it.wikipedia.org/wiki/Restrizione_di_una_funzione`
- Branch- and bound: `https://it.wikipedia.org/wiki/Branch_and_bound`

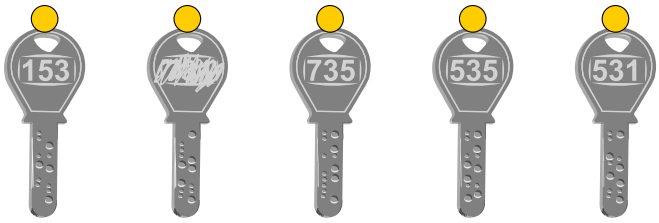
26. Armadietti

Cinque bambini hanno ciascuno un armadietto etichettato nella loro scuola. Le cinque chiavi corrispondenti hanno numeri a tre cifre. Sfortunatamente, una chiave ha un numero graffiato.

Ogni numero a tre cifre rappresenta le prime tre lettere di un nome. Una cifra sta per la stessa lettera ovunque, ad esempio 8 sta sempre per «C» o «c».

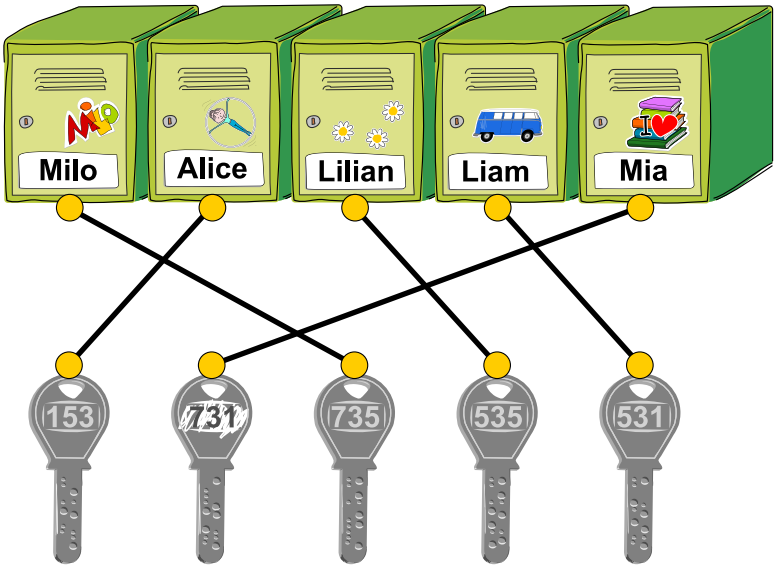
Assegna le chiavi agli armadietti corretti. Traccia delle linee tra i punti gialli.





Soluzione

Di seguito viene mostrata la soluzione corretta:



I quattro numeri conosciuti sono: 153, 735, 535, 735. Le prime tre lettere dei cinque nomi sono MIL, ALI, LIL, LIA, LIA, MIA.

Solo LIL inizia e finisce con la stessa lettera. Quindi deve includere un numero a tre cifre che inizia e finisce con la stessa cifra, e ci può essere solo un numero di questo tipo. Il numero 535 si adatta a questo modello, quindi deve appartenere a LIL. Quindi 5 sta per L e I per 3. Ora possiamo vedere che 531 deve stare per LIA, perché altrimenti non ci sono nomi che iniziano per L. Quindi 1 sta per A. Inoltre, 153 deve stare per ALI, perché altrimenti nessun nome ha una L al secondo posto. Ora solo il numero 7 e la lettera M non sono assegnati. Quindi devono stare insieme. Quindi abbiamo la seguente chiara assegnazione: 1 = A, 3 = I, 5 = L e 7 = M. Quindi 735 sta per MIL e 531 per LIA. Ora vediamo anche che la chiave con il numero graffiato appartiene a Mia e che il numero graffiato deve essere 731.

Un'altra idea per trovare l'assegnazione corretta è quella di contare la frequenza di lettere e numeri. In MIL, ALI, LIL, LIA, LIA, MIA le due lettere A e M appaiono due volte ciascuna e le lettere I e L appaiono cinque volte ciascuna. Purtroppo, questo non è ancora sufficiente per una chiara assegnazione delle lettere ai numeri. È quindi necessario fare ulteriori osservazioni, ad esempio quelle sopra descritte.

Questa è l'informatica!

Nell'informatica, i nomi e i testi sono spesso codificati con numeri.

La dichiarazione del compito indica che i numeri sulle chiavi possono essere ricavati in modo univoco dalle prime tre lettere dei rispettivi nomi. Questo funziona assegnando esattamente una cifra ad ogni lettera come codifica e utilizzando solo poche lettere. Si parla di una codifica *monoalfabetica*, poiché ogni lettera è sostituita ovunque dallo stesso carattere. D'altra parte, non è stato specificato quale

cifra è effettivamente assegnata a quale lettera. Ma la soluzione mostra come si possa trovare la corretta assegnazione con l'aiuto di alcuni indizi strutturali.

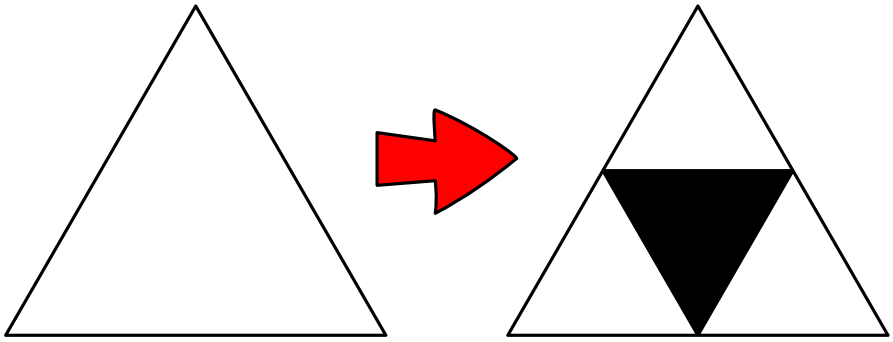
Se per la codifica non si usano solo 10 cifre, ma un simbolo per ogni lettera, allora si può utilizzare una tale sostituzione monoalfabetica come semplice codice segreto. Purtroppo il metodo di cifratura monoalfabetico non è molto sicuro, perché spesso è possibile scoprire il codice rapidamente con qualche trucco. Il compito è un esempio di questo. Fortunatamente ci sono molti sistemi di crittografia migliori. La *crittografia* è un importante sottocampo dell'informatica, in cui vengono sviluppati e analizzati vari codici segreti.

Parole chiave e siti web

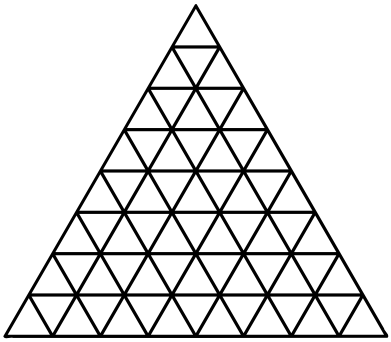
- Cifrario a sostituzione: `https://it.wikipedia.org/wiki/Cifrario_a_sostituzione`
- Crittografia: `https://it.wikipedia.org/wiki/Crittografia`

27. Triangolo di Sierpiński

Per ottenere il cosiddetto triangolo di Sierpiński, si deve prima disegnare un triangolo bianco equilatero. Poi si procede passo dopo passo. In ogni passo, ogni triangolo bianco esistente è diviso in quattro più piccoli e quello centrale è colorato di nero, come mostrato nella figura seguente:

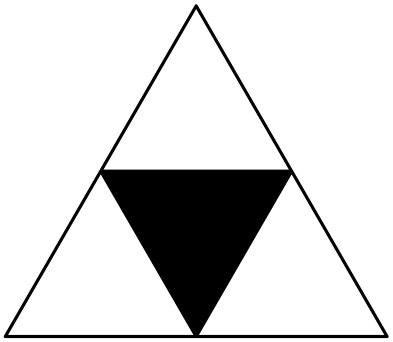


Disegna la figura che emerge dopo tre passi. Colora di nero i triangoli corretti.

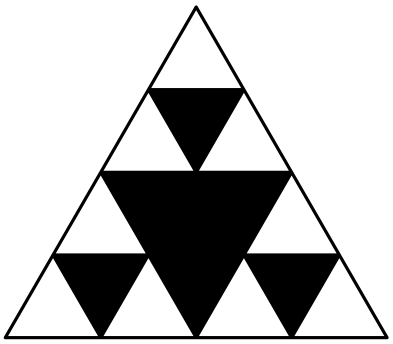


Soluzione

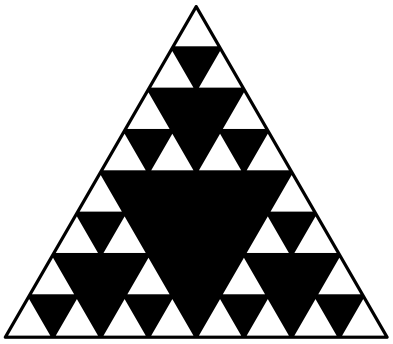
Dopo il primo passo, il triangolo centrale è nero e rimangono tre triangoli bianchi:



Nel secondo passo, questi tre triangoli parziali sono nuovamente suddivisi in quattro triangoli parziali più piccoli, di cui quello centrale è colorato di nero. Questo lascia $3 \cdot 3 = 9$ triangoli parziali bianchi più piccoli:



Nel terzo e ultimo passo, questi 9 triangoli bianchi sono divisi in quattro triangoli più piccoli e quello centrale è dipinto. La seguente figura con $9 \cdot 3 = 27$ triangoli parziali bianchi viene creata:



Questa è l'informatica!

Il triangolo di Sierpiński è un *frattale* descritto per la prima volta dal matematico polacco Wacław Franciszek Sierpiński (1882–1969) nel 1915. I frattali sono figure in cui compaiono parti sempre più piccole, simili all'intera figura. Disegnare immagini esatte dei frattali è estremamente complesso. Quando nel XX secolo apparvero i computer che potevano fare i calcoli necessari, i frattali divennero molto popolari. I frattali più noti sono la *curva di Koch* e l'insieme di *Mandelbrot*.

La costruzione del triangolo di Sierpiński è *ricorsiva* (dal latino *re-currere*: correre indietro, tornare indietro). Questo significa quanto segue: Le istruzioni per la costruzione contengono una dichiarazione che dice che bisogna rifare l'intera istruzione. Nell'esempio, questa istruzione dice: «Dividi il triangolo bianco in quattro triangoli più piccoli, colora quello centrale di nero e ripeti questa istruzione per gli altri tre triangoli.» Un passaggio attraverso le istruzioni è chiamato *passo ricorsivo*, e le istruzioni per passare attraverso le istruzioni di nuovo sono le chiamate ricorsive. (Nell'esempio, ci sono tre chiamate ricorsive per ogni passo ricorsivo.) Poiché ci sono nuove chiamate ricorsive in ogni chiamata ricorsiva, il passo ricorsivo deve essere eseguito più e più volte, il che richiede un tempo infinito. È possibile evitarlo utilizzando una *condizione di terminazione*. Nell'esempio le chiamate di ricorsione si fermano quando i triangoli diventano troppo piccoli.

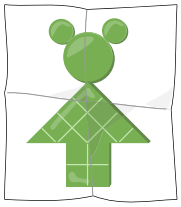
Il concetto di *algoritmo ricorsivo* è ampiamente utilizzato nell' informatica. Molti oggetti complessi — per esempio i frattali — possono essere descritti in modo compatto con la ricorsione e molti compiti complicati — per esempio il problema delle *torri di Hanoi* — possono essere risolti con algoritmi ricorsivi molto semplici.

Parole chiave e siti web

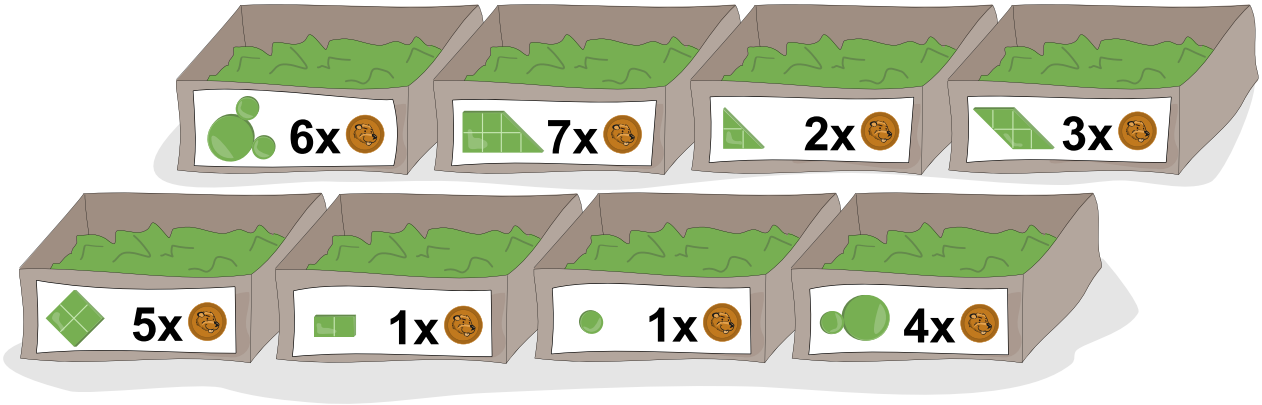
- Triangolo di Sierpiński: `https://it.wikipedia.org/wiki/Triangolo_di_Sierpiński`
- Algoritmo ricorsivo: `https://it.wikipedia.org/wiki/Algoritmo_ricorsivo`
- Frattale: `https://it.wikipedia.org/wiki/Frattale`
- Torre di Hanoi: `https://it.wikipedia.org/wiki/Torre_di_Hanoi`
- Curva di Koch: `https://it.wikipedia.org/wiki/Curva_di_Koch`
- Insieme di Mandelbrot: `https://it.wikipedia.org/wiki/Insieme_di_Mandelbrot`

28. Gioco con le tessere

Giulia vuole comprare delle tessere per realizzare questa figura:



Il negozio di giocattoli offre diverse tessere in qualsiasi quantità. I prezzi per tessera variano da 1 a 7 monete.



Le tessere possono essere girate e capovolte a piacere, ma non devono sovrapporsi.

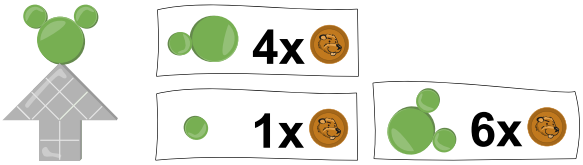
Quante monete deve spendere Giulia se sceglie l'opzione più economica?

A) 13 monete
B) 14 monete
C) 16 monete
D) 20 monete

Soluzione

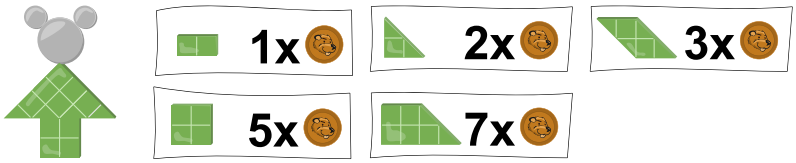
La risposta corretta è 13 monete.

Una soluzione è quella di guardare le singole parti della figura separatamente. Il modo più semplice per iniziare è quello di iniziare con la testa, che può essere posizionata solo con tessere rotonde:

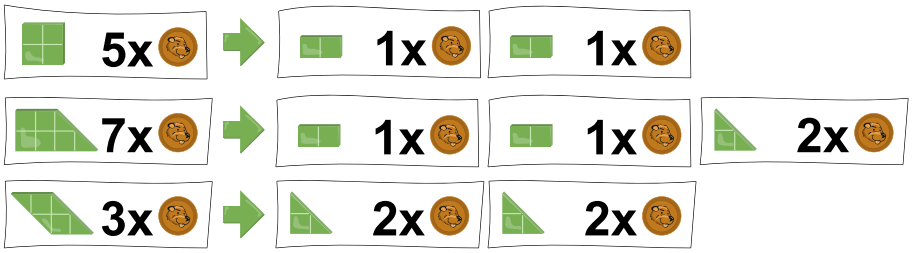


Ci sono solo due possibilità per la testa: O si usa la tessera corrispondente per 6 monete direttamente o la si mette insieme dalle altre due tessere rotonde, che insieme costano $4 + 1 = 5$ monete. La seconda opzione è più economica, quindi usiamo questa.

Il resto della figura può essere assemblato solo da tessere spigolose



Ora è possibile provare tutti i modi possibili per realizzare la figura e calcolare il prezzo per tutti. Ma questo è molto complesso. Le seguenti osservazioni sulle tessere quadrate portano più velocemente alla soluzione:



- Una tessera quadrata per 5 monete può sempre essere sostituita da due rettangoli per $1 + 1 = 2$ monete. Questo lo rende sempre più economico.
- Si potrebbe anche sostituire una tessera quadrata con due triangoli, ma sarebbe un po' più costoso con $2 + 2 = 4$ monete, quindi questa è l'opzione peggiore.

Ecco perché Giulia non compra mai un quadrato, anche se ci starebbe bene, ma sempre due rettangoli.

- Un trapezio per 7 monete può essere composto da un quadrato e da un triangolo. Se sostituiamo il quadrato con due rettangoli, $1 + 1 + 2 = 4$ monete sono sufficienti per questo trapezio.

Così Giulia non compra mai un trapezio direttamente, anche se uno ci starebbe bene, ma lo mette sempre insieme con due rettangoli e un triangolo.

- Il parallelogramma per 3 monete potrebbe essere sostituito da due triangoli più piccoli per $2 + 2 = 4$ monete. Ma questo lo rende solo più costoso, quindi non è una buona opzione.

Un parallelogramma potrebbe essere utile per Giulia, ma se questo è davvero il caso sarà rivelato solo da un approfondimento.

Versione A		**Versione B**	
	• Testa per 5 monete • Corpo di 4 rettangoli e 2 triangoli: $1+1+1+1+2+2=8$ monete		• Testa per 5 monete • Corpo di 1 parallelogrammo, 2 rettangoli e 2 triangoli: $3+1+1+2+2=9$ monete

Se Giulia non usa il parallelogramma, ha bisogno di due triangoli per posizionare i punti triangolari a sinistra e a destra della figura. Può poi spendere il resto con rettangoli, come nella versione A, che costa complessivamente $5+8=13$ monete.

Il parallelogramma si inserisce nella figura in un solo modo, come mostrato nella versione B (o la versione specchiata). Se si posiziona un parallelogramma in questo modo e il resto viene riempito di rettangoli e triangoli, la figura costa $5+9=14$ monete. Tutti gli altri posizionamenti del parallelogramma darebbero lacune che non possono essere colmate.

È quindi chiaro che 13 monete sono la soluzione più economica.

Questa è l'informatica!

Il compito di realizzare una certa figura con determinate tessere può essere estremamente complicato anche per pochissime parti. Un esempio è il gioco Tangram.

Il problema a disposizione è ancora più complicato, perché anche il prezzo totale delle tessere deve essere ottimizzato. Nell'informatica un tale problema si chiama *problema di ottimizzazione*.

Il problema è stato risolto con un importante principio dell'informatica: dividere un problema in sottoproblemi più piccoli che possono essere risolti indipendentemente l'uno dall'altro e le cui soluzioni possono poi essere messe insieme per formare una soluzione globale. In concreto, il problema è stato diviso in due sotto-problemi che possono essere risolti indipendentemente, uno per le tessere rotonde e uno per le tessere quadrate. Con le tessere spigolose, la combinazione di tessere più favorevole per un quadrato può essere riutilizzata ovunque senza doverci pensare più e più volte. Lo stesso vale per il parallelogramma.

La suddivisione in sottoproblemi indipendenti è molto importante nella programmazione. Il riutilizzo di soluzioni per problemi secondari che si verificano più di una volta può far risparmiare molto tempo. Questo si chiama principio di *modularità*. La suddivisione in sottoproblemi più piccoli è anche la base per i programmi basati sul principio «*divide et impera*» (in inglese «divide and conquer»).

Parole chiave e siti web

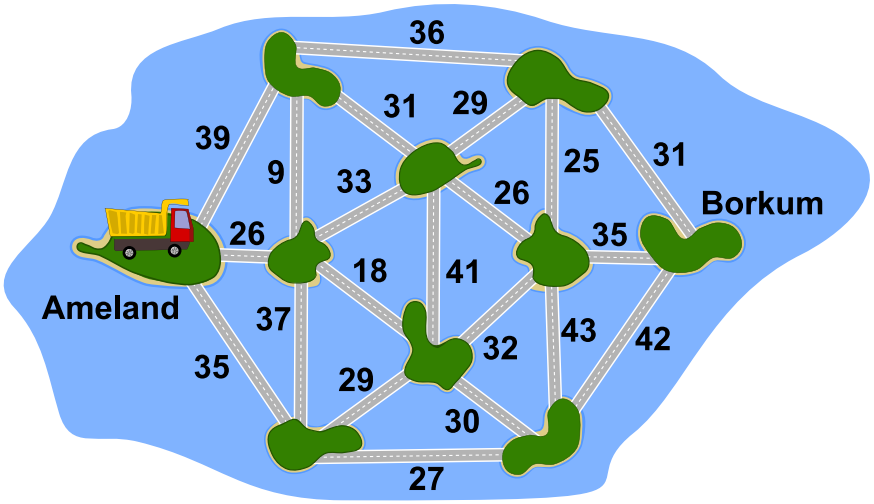
- Problema di ottimizzazione:
  `https://it.wikipedia.org/wiki/Problema_di_ottimizzazione`
- Divide et impera: `https://it.wikipedia.org/wiki/Divide_et_impera_(informatica)`
- Tangram: `https://it.wikipedia.org/wiki/Tangram`

29. L'arcipelago dei castori

L'arcipelago dei castori è composto da dieci isole collegate da ponti. Qui sotto c'è una mappa. Il numero su ogni ponte indica il peso totale massimo ammissibile in tonnellate per un camion che vuole attraversare quel ponte.

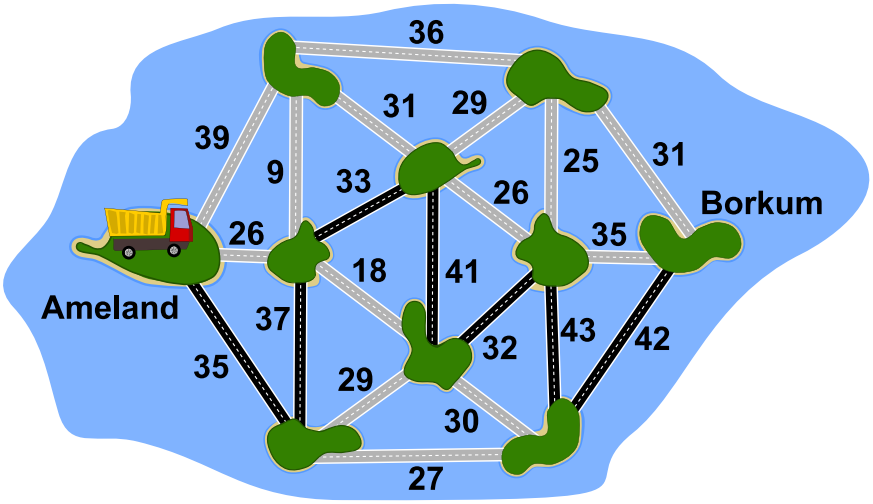
Il castoro Knuth vuole costruire una spiaggia sull'isola di Borkum. Vuole quindi trasportare quanta più sabbia possibile dall'isola di Ameland all'isola di Borkum in un solo viaggio. Non gli interessa la lunghezza del viaggio, ma non vuole passare su nessun ponte più di una volta.

Che strada deve prendere con il suo camion per arrivare a Borkum? Disegnala sulla mappa.

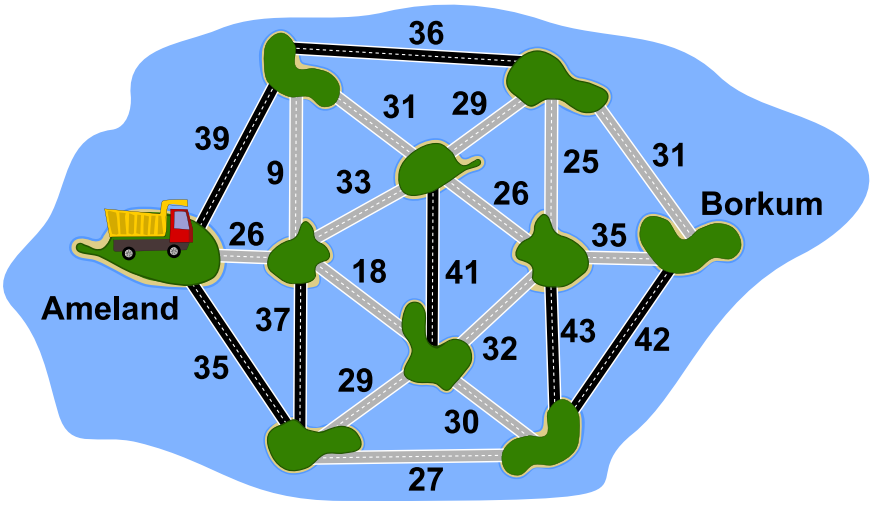


Soluzione

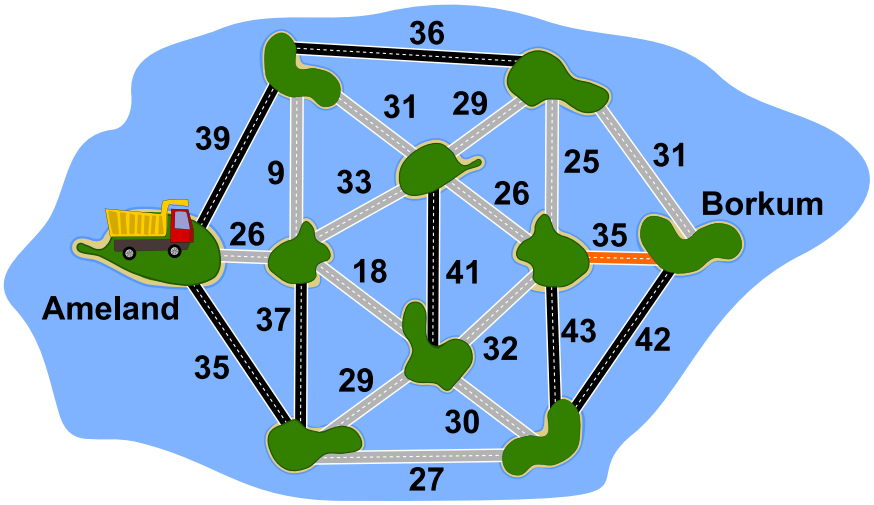
Per il viaggio, il peso totale massimo di un camion è di 32 tonnellate. Si prende il seguente percorso, ad esempio:



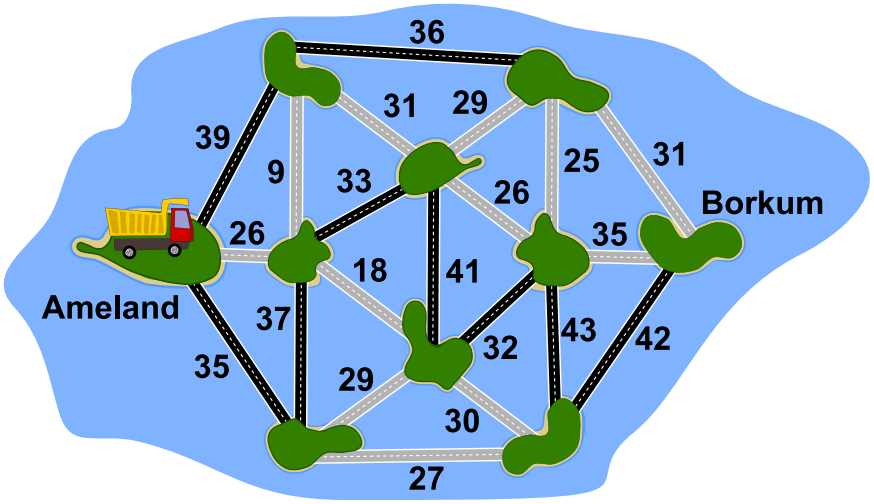
Per determinare questo, possiamo, ad esempio, per prima cosa togliere tutti i ponti dalla mappa e ordinarli in base alla loro capacità di carico. Iniziamo con quelli con la maggiore capacità di carico e li aggiungiamo alla mappa. Poi aggiungiamo quelli con la seconda maggior capacità di carico e così via. Nel seguente diagramma i ponti inseriti con le portate 43, 42, 41, 39, 39, 37, 36, 35 sono contrassegnati in nero.



Tuttavia, se inserendo un ponte dovessimo creare un cosiddetto ciclo, cioè un percorso circolare, non lo metteremmo in quanto le isole di questo ciclo sarebbero già accessibili da ponti di maggiore capacità. Nel seguente diagramma, il ponte con una capacità di carico di 35 verrebbe inserito, ma non farebbe altro che abbreviare un percorso già esistente.



Lo faremo fino a quando tutte le isole saranno collegate. Ora c'è solo una strada possibile tra ogni paio d'isole e il ponte con la capacità più piccola dà il massimo peso che stiamo cercando.



Questa è l'informatica!

Una vera e propria applicazione per la soluzione di questo compito è l'identificazione del «collo di bottiglia» (in inglese «bottleneck») nelle reti di computer, cioè la maggiore velocità di trasmissione possibile tra due computer della rete. Il compito qui riguarda il peso totale massimo di un camion in viaggio tra due isole come un collo di bottiglia. Questo è determinato dalla capacità di carico del ponte più debole. Nelle reti di computer questo sarebbe il collegamento con la larghezza di banda più bassa.

Per una soluzione, la rete può essere prima modellata, cioè semplificata, come qui presentato. Nel nostro caso, l'*algoritmo di Kruskal* crea un albero ricoprente massimo in cui il collo di bottiglia è direttamente visibile.

Parole chiave e siti web

- Albero ricoprente: `https://it.wikipedia.org/wiki/Albero_ricoprente`
- Algoritmo di Kruskal: `https://it.wikipedia.org/wiki/Algoritmo_di_Kruskal`

30. Lavagna rovinata

I castori utilizzano un codice segreto in cui ogni lettera è sostituita da un carattere completamente nuovo. Come creare i nuovi caratteri è descritto nella lavagna sottostante. Purtroppo la lavagna non è completa perché alcune parti sono state cancellate.

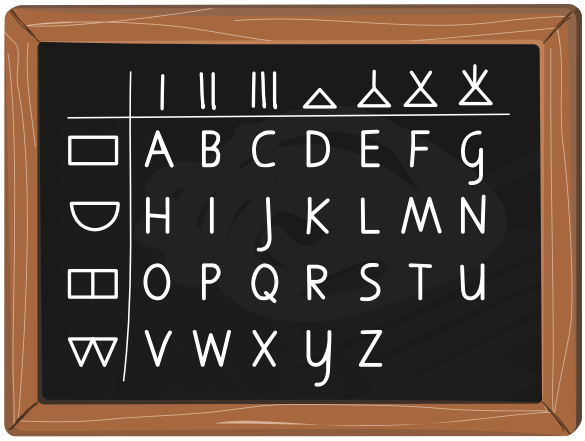
Ricostruisci il testo originale a partire dal testo cifrato attuale (decifra il testo cifrato). Quale delle 4 soluzioni è corretta?

A) INFORMATICA BELLA
B) MATEMATICA È BELLA
C) INFORMAZIONE VERA
D) INFORMAZIONI VERE

Soluzione

La risposta corretta è A), il testo decifrato è: INFORMATICA BELLA.

Ecco la lavagna di cifratura completa:



Si può facilmente ricostruire la lavagna. Le lettere dell'alfabeto latino sono disposte riga per riga da sinistra a destra. Noterete che i nuovi caratteri sono composti in modo tale che la designazione della riga corrisponde alla parte inferiore e la designazione della colonna corrisponde alla parte superiore. L'unica parte inferiore mancante nel testo cifrato è il □. Quindi questo carattere è il simbolo mancante della prima riga. I tre caratteri mancanti per le colonne possono essere determinati altrettanto rapidamente.

Ma non è necessario ripristinare completamente la lavagna. Potete utilizzare le lettere che potete leggere direttamente dalla tabella danneggiata. In questo modo si ottiene il seguente testo con gli spazi vuoti:

I N _ O _ _ _ _ I _ _ _ _ L L _

Con questo testo con gli spazi è possibile escludere tutte le soluzioni tranne A): B) inizia con «MA», C) termina con «ERA», D) termina con «ERE».

Un'altra soluzione è riconoscere che il testo cifrato contiene due caratteri uguali alla penultima e terzultima posizione. Quindi solo A) e B) entrano in discussione. Il primo carattere può essere chiaramente identificato come «I» nella lavagna danneggiata, il che rende chiaro che la soluzione corretta è A).

Questa è l'informatica!

Mantenere la segretezza delle informazioni e proteggere i dati è un compito che risale a 4000 anni fa. A questo scopo sono stati sviluppati e utilizzati innumerevoli linguaggi segreti. Oggi la sicurezza dei dati è uno dei temi centrali dell'informatica. Uno dei metodi per proteggere i dati da letture non autorizzate è la *crittografia*. La cifratura trasforma un testo in chiaro in un *testo cifrato*. Ricostruire il testo in chiaro dal testo cifrato si chiama *decifrare*. La scienza del testo cifrato si chiama *crittologia*.

Le culture antiche utilizzavano per lo più scritture segrete, che venivano create codificando le lettere con altre lettere o con caratteri completamente nuovi. Il cifrario seguente è stato sviluppato specialmente per la competizione del castoro informatico, ma si basa su un concetto dell'antica Palestina. La regola di sicurezza dell'epoca era che si usavano solo codici segreti che si potevano imparare facilmente a memoria. Mantenere una descrizione scritta del codice segreto era considerato un rischio troppo grande. Una tabella, come si usa qui, è facile da imparare a memoria. Il famoso codice segreto dei massoni si basa su questo principio.

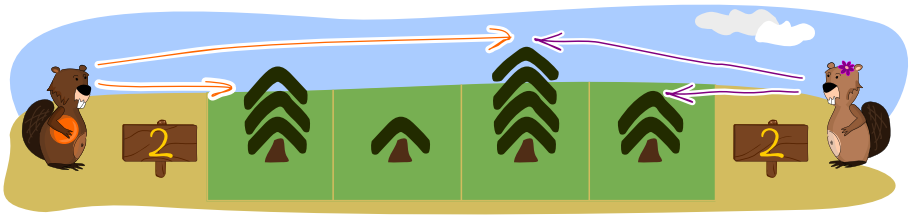
Parole chiave e siti web

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

31. 4×4 sudoku con gli alberi

I castori piantano sedici alberi (quattro alberi di altezza 4, quattro alberi di altezza 3, quattro alberi di altezza 2 e quattro alberi di altezza 1) in un campo di alberi 4 x 4, seguendo le seguenti regole:

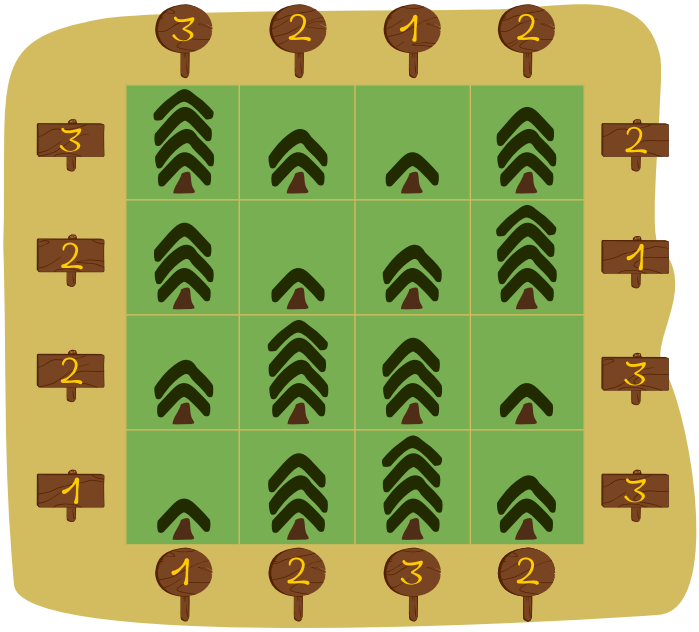
- In ogni riga (riga orizzontale) c'è esattamente un albero di ogni altezza;
- In ogni colonna (riga verticale) c'è esattamente un albero di ogni altezza.



Quando i castori guardano una fila di alberi da un lato, **non** possono vedere gli alberi più bassi nascosti dietro gli alberi più alti. Alla fine di ogni fila di alberi c'è un cartello che indica quanti alberi un castoro può vedere da quel punto. Questi cartelli con il numero di alberi visibili sono posizionati intorno al campo di alberi.

Kubko ha cercato di trasferire la descrizione del campo su un foglio di carta. Ha trasferito correttamente i numeri dei cartelli, ma si è sbagliato con quattro alberi.

Cerchia le quattro posizioni con gli alberi inseriti in modo errato e annota di lato l'altezza corretta che l'albero dovrebbe avere



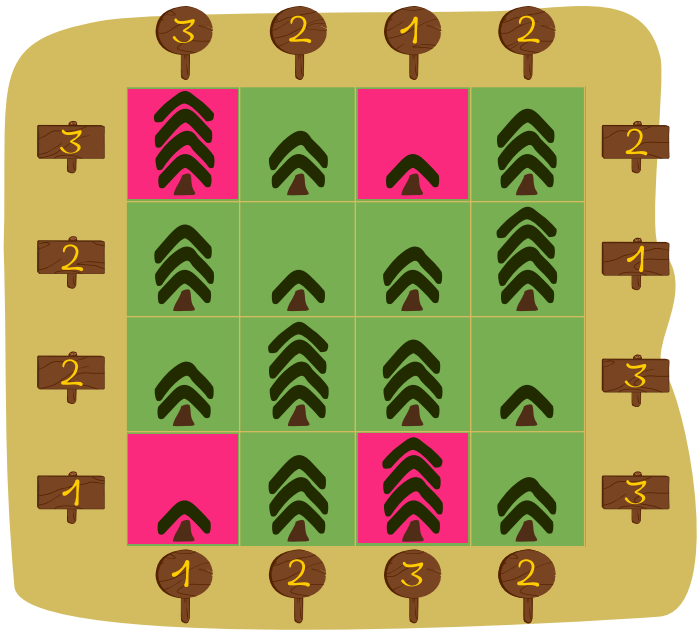
Soluzione

Prima di tutto si nota che entrambe le regole del «Sudoku» sono state seguite: C'è esattamente un albero di ogni altezza in ogni fila.

Poi si può vedere per quali righe i numeri sui pannelli sono corretti e per quali no. Si può vedere che i numeri sono corretti per le righe 2 e 3 e per le colonne 2 e 4. Per le altre righe i numeri non sono corretti, chiamiamo queste *righe problematiche*.

Ma questo non basta. Vogliamo sapere quali sono le posizioni che causano i numeri sbagliati. Per fare questo, si nota che ci sono esattamente quattro posizioni che sono contemporaneamente in una riga problematica e in una colonna problematica. Queste sono le quattro posizioni in cui le righe problematiche (cioè 1 e 4) si intersecano con le colonne problematiche (cioè 1 e 3).

Se si scambiano le coppie di alberi in questi quattro incroci problematici (contrassegnati in rosso sotto) tra le righe o le colonne, si ottiene la soluzione corretta.



Che questa sia effettivamente l'unica soluzione possibile può essere vista come segue: Esattamente quattro alberi sono sbagliati. Se un albero viene cambiato in una posizione, è necessario cambiarne almeno altri due per mantenere la regola del Sudoku soddisfatta, ovvero: un albero nella riga corrispondente e uno nella colonna. Quindi ci sono già tre cambiamenti. Le ultime due modifiche forzano di nuovo un'altra modifica nella riga o colonna corrispondente. Poiché in totale possono essere apportate solo quattro modifiche, queste due devono ora coincidere. Ciò è possibile solo se le quattro posizioni con le modifiche sono disposte in un rettangolo. Poiché è necessario apportare almeno una modifica in ogni riga problematica, la soluzione di cui sopra è l'unica possibile.

Questa è l'informatica!

Questo compito si concentra su tre competenze di base degli informatici.

La prima è quella di trovare una soluzione che rispetti i vincoli indicati, o di correggere una soluzione proposta, se necessario.

In secondo luogo, si tratta della capacità di ricostruire gli oggetti a partire da informazioni parziali attraverso la loro rappresentazione. Questo è legato alla generazione di oggetti (*rappresentazioni di oggetti*) a partire da informazioni limitate disponibili, se si conosce la regolarità dell'oggetto. Tali procedure possono essere utilizzate anche per la *compressione*.

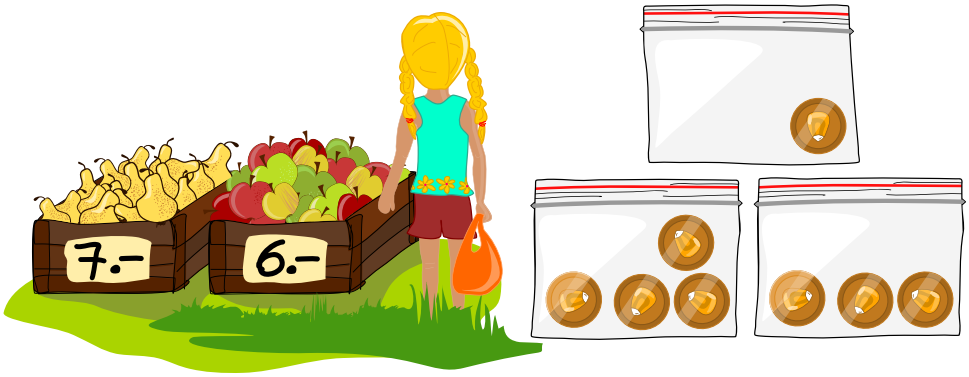
In terzo luogo, tali campi ad albero con etichette possono essere utilizzati per generare *codici autoverificanti*. Eventuali errori che si verificano durante l'inserimento dei dati o il trasporto delle informazioni possono poi essere rilevati automaticamente o addirittura corretti.

Parole chiave e siti web

- Sudoku
- Rappresentazioni di oggetti
- Compressione: `https://it.wikipedia.org/wiki/Compressione_dei_dati`
- Rilevazione e correzione d'errore: `https://it.wikipedia.org/wiki/Rilevazione_e_correzione_d'errore`

32. Sacchetto per i soldi

A Bina piace andare a nuotare. Mette i suoi soldi in sacchetti impermeabili in modo che il metallo non inizi ad arrugginire. Ieri Bina aveva con sé tre sacchetti con 1, 3 e 4 monete. Con queste monete poteva pagare una pera in modo esatto (cioè senza resto) tenendo i sacchetti chiusi, ma non una mela.



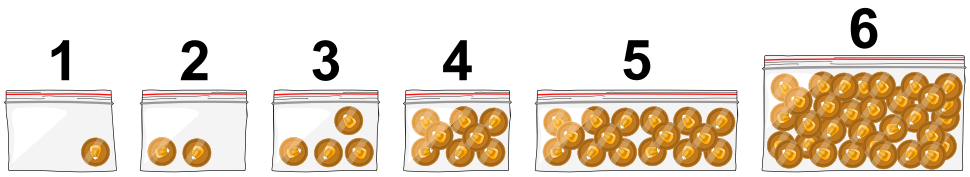
Oggi Bina ha con sé 63 monete identiche. Vuole dividerle in sacchetti diversi in modo da poter pagare qualsiasi importo compreso tra 1 e 63 monete senza dover aprire i sacchetti.

Qual è il numero più piccolo di sacchetti di cui Bina ha bisogno?

A) 4 sacchetti
B) 5 sacchetti
C) 6 sacchetti
D) 7 sacchetti
E) 8 sacchetti
F) 15 sacchetti
G) 16 sacchetti
H) 31 sacchetti
I) 32 sacchetti o di più

Soluzione

La risposta corretta è C) 6 sacchetti:



Bina può dividere le monete tra i 6 sacchetti come segue:

- Sachetto 1: 1 moneta
- Sachetto 2: 2 monete
- Sachetto 3: 4 monete
- Sachetto 4: 8 monete
- Sachetto 5: 16 monete
- Sachetto 6: 32 monete

Bina ha quindi un totale di $1 + 2 + 4 + 8 + 16 + 32 = 63$ monete nei sacchetti e può pagare qualsiasi importo totale da 1 a 63 monete in sacchetti chiusi.

Per pagare 13 monete, ad esempio, può pagare con i sacchetti 1, 3 e 4.

La tabella seguente mostra come ogni importo totale può essere pagato adeguatamente con la giusta selezione dei sacchetti. Una cella contiene un 1 se Bina usa il sacchetto corrispondente per pagare, e 0 in caso contrario.

Importo	32	16	8	4	2	1	Importo	32	16	8	4	2	1
0	0	0	0	0	0	0	**32**	1	0	0	0	0	0
1	0	0	0	0	0	1	**33**	1	0	0	0	0	1
2	0	0	0	0	1	0	**34**	1	0	0	0	1	0
3	0	0	0	0	1	1	**35**	1	0	0	0	1	1
4	0	0	0	1	0	0	**36**	1	0	0	1	0	0
5	0	0	0	1	0	1	**37**	1	0	0	1	0	1
6	0	0	0	1	1	0	**38**	1	0	0	1	1	0
7	0	0	0	1	1	1	**39**	1	0	0	1	1	1
8	0	0	1	0	0	0	**40**	1	0	1	0	0	0
9	0	0	1	0	0	1	**41**	1	0	1	0	0	1
10	0	0	1	0	1	0	**42**	1	0	1	0	1	0
11	0	0	1	0	1	1	**43**	1	0	1	0	1	1
12	0	0	1	1	0	0	**44**	1	0	1	1	0	0
13	0	0	1	1	0	1	**45**	1	0	1	1	0	1
14	0	0	1	1	1	0	**46**	1	0	1	1	1	0
15	0	0	1	1	1	1	**47**	1	0	1	1	1	1
16	0	1	0	0	0	0	**48**	1	1	0	0	0	0
17	0	1	0	0	0	1	**49**	1	1	0	0	0	1
18	0	1	0	0	1	0	**50**	1	1	0	0	1	0
19	0	1	0	0	1	1	**51**	1	1	0	0	1	1
20	0	1	0	1	0	0	**52**	1	1	0	1	0	0
21	0	1	0	1	0	1	**53**	1	1	0	1	0	1
22	0	1	0	1	1	0	**54**	1	1	0	1	1	0
23	0	1	0	1	1	1	**55**	1	1	0	1	1	1
24	0	1	1	0	0	0	**56**	1	1	1	0	0	0
25	0	1	1	0	0	1	**57**	1	1	1	0	0	1
26	0	1	1	0	1	0	**58**	1	1	1	0	1	0
27	0	1	1	0	1	1	**59**	1	1	1	0	1	1
28	0	1	1	1	0	0	**60**	1	1	1	1	0	0
29	0	1	1	1	0	1	**61**	1	1	1	1	0	1
30	0	1	1	1	1	0	**62**	1	1	1	1	1	0
31	0	1	1	1	1	1	**63**	1	1	1	1	1	1

Con meno di 6 sacchi Bina non può raggiungere il suo obiettivo. Può usare o meno ogni sacchetto quando paga, quindi ci sono esattamente due possibilità per ogni sacchetto. Con solo 5 o anche meno sacchi, avrebbe al massimo $2^5 = 2 \cdot 2 \cdot 2 \cdot 2 \cdot 2 = 32$ combinazioni possibili. Quindi potrebbe pagare al massimo 32 diversi importi totali, che non è sufficiente per tutti gli importi totali fino a 63 monete.

Questa è l'informatica!

Questo compito riguarda i *numeri binari*. I numeri binari sono studiati in matematica e informatica in diversi modi. La matematica si occupa principalmente delle loro proprietà, mentre l'informatica si occupa maggiormente delle loro applicazioni. I computer utilizzano numeri binari per rappresentare informazioni di tipo molto diverso: Documenti, immagini, voci, video e numeri, anche i programmi e le applicazioni che tutti noi utilizziamo sono codificati come numeri binari. L'unità è un *bit* (dall'inglese «*binary digit*»), che può assumere il valore 0 o 1. Quindi un bit da solo può distinguere solo due possibilità. Con due bit, tuttavia, si possono distinguere quattro possibilità: 00, 01, 10 e 11. In questo compito, Bina utilizza 6 bit (sacchetti) per rappresentare $2^6 = 64$ importi diversi.

Nei computer, i bit sono di solito raggruppati in gruppi di otto; tale gruppo di otto è chiamato *byte*. Un byte può rappresentare $2^8 = 256$ numeri diversi, da 0 a 255.

Parole chiave e siti web

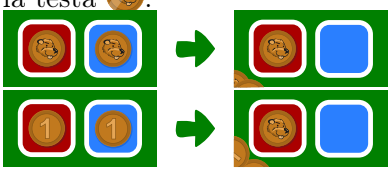
- Sistema numerico binario: `https://it.wikipedia.org/wiki/Sistema_numerico_binario`

33. Las Bebras

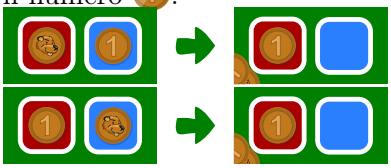
Al Casinò «Las Bebras» Gloria può giocare da John utilizzando delle monete. Gloria ha 4 monete con rappresentato sul davanti una testa, e sul retro un numero 1. Gloria lancia le prime 2 monete e ne mette una sul quadrato rosso e l'altra sul quadrato blu.

John scambia le due monete con una nuova moneta che mette sul campo rosso.

- Se le due monete sono uguali, John mette la nuova moneta sul campo rosso lasciando visibile la testa.



- Se le due monete sono diverse, John mette la nuova moneta sul campo rosso lasciando visibile il numero.



Gloria ora lancia un'altra moneta e la mette sul quadrato blu, John la sostituisce di nuovo secondo le regole di cui sopra e così via fino a quando Gloria non avrà giocato tutte e 4 le monete. La partita finisce quando John mette l'ultima moneta sul campo rosso. Se si vede il numero 1 Gloria vince!

Gloria gioca le 4 monete in ordine da sinistra a destra. In quale caso vince Gloria?

A) 

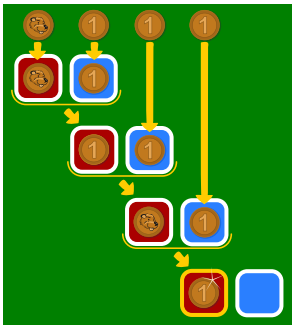
B) 

C) 

D) 

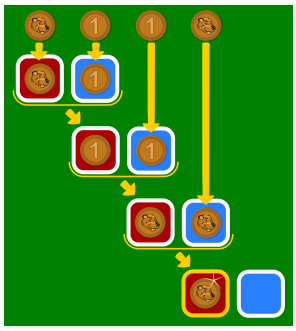
Soluzione

La risposta corretta è C). Solo per il caso C) alla fine si ottiene una moneta nel campo rosso con il numero  visibile.

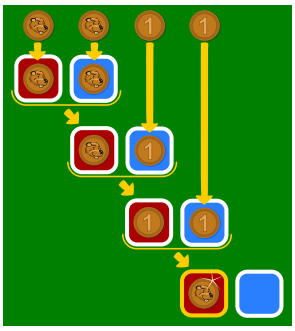


In tutti gli altri casi alla fine si ottiene una moneta nel campo rosso con visibile la testa  .

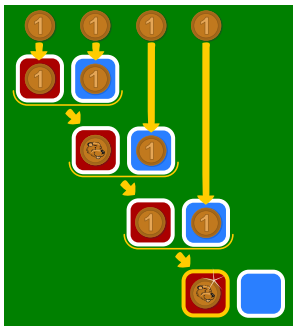
A)



B)



D)



Per ognuna delle 4 monete che Gloria gioca, ci sono 2 modi per piazzarle ( oppure ), quindi con 4 monete si può giocare un totale di $2^4 = 16$ sequenze. Se c'è un numero pari di monete con testa (o numero) in fila, alla fine del gioco la moneta viene posta con visibile la testa  sul campo rosso. Se c'è un numero dispari di monete con testa (o numero) in fila, la moneta viene messa sul campo rosso alla fine della partita con il numero visibile  . Un numero dispari di monete con testa (o numero) in fila rappresenta quindi la «sequenza vincente». Ci sono esattamente 8 sequenze con un numero dispari e 8 sequenze con un numero pari.

Questa è l'informatica!

Poiché i computer sono macchine elettroniche, l'elettricità viene utilizzata per rappresentare le informazioni che immettiamo al loro interno. Quando l'elettricità scorre, diciamo che è accesa. Se non c'è elettricità, diciamo che è spenta. Gli informatici di solito rappresentano questi due stati con i due numeri 0 e 1, che chiamiamo «rappresentazione binaria». Un'unità di informazione è chiamata «bit».

Possiamo eseguire operazioni su tali bit e combinarli, così come con le due posizioni delle monete (testa o numero).

Una di queste operazioni si chiama disgiunzione esclusiva (oppure XOR per «eXclusive OR» in inglese) ed è quella presentata in questo compito. Funziona così:

0 XOR 0 = 0
0 XOR 1 = 1
1 XOR 0 = 1
1 XOR 1 = 0

Un esempio d'uso quotidiano: su entrambi i lati di una scala ci sono due interruttori della luce che accendono o spengono la stessa luce. Se entrambi gli interruttori della luce sono alzati, la luce è accesa e se entrambi sono abbassati, la luce è anche accesa. Se uno è alzato e l'altro è abbassato, la luce è spenta.

Un gate XOR è un'implementazione elettronica del funzionamento XOR nei computer. Un gate XOR da un risultato di 1 alla sua uscita se esattamente uno dei suoi due ingressi è 1. Se entrambi gli ingressi sono uguali, il risultato in uscita è 0.

Nell' informatica, l'operazione XOR ha diverse applicazioni come per esempio:

- Ci dice se due bit sono uguali o diversi.
- Ci dice se il numero di bit di 1 è pari o dispari (lo XOR di una sequenza di bit è «vero» esattamente quando un numero dispari è «vero»).
- Nella crittografia, l'operazione XOR viene utilizzata nella crittografia simmetrica con i cosiddetti «one-time pad».

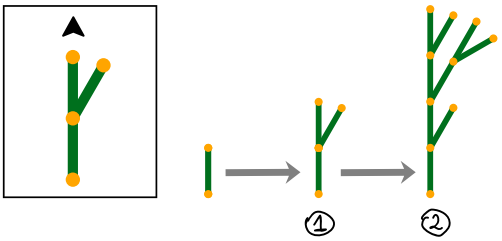
Parole chiave e siti web

- Operazione binaria: `https://it.wikipedia.org/wiki/Operazione_binaria`
- XOR: `https://it.wikipedia.org/wiki/Porta_logica#EXOR`
- Porta logica: `https://it.wikipedia.org/wiki/Porta_logica`

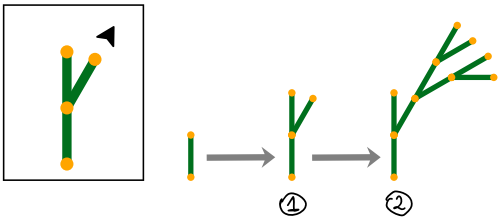
34. Alberi digitali

Un albero digitale cresce dal seguente pezzo di albero singolo:  .

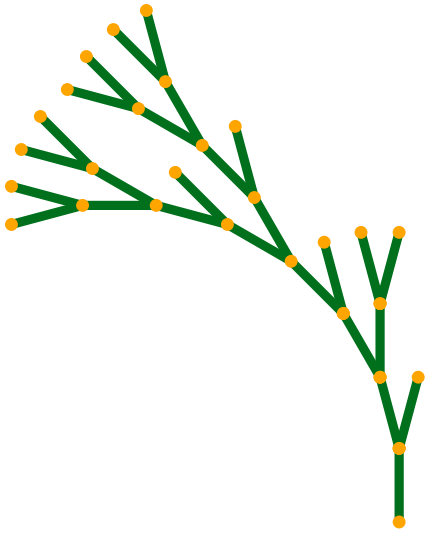
Cresce gradualmente secondo una regola di crescita predeterminata.
La regola della crescita specifica come un pezzo di albero può essere sostituito da una struttura di nuovi pezzi di albero. In ogni passo, ogni pezzo di albero viene sostituito in questo modo. La punta di una freccia indica dove e in quale direzione vengono messi insieme i pezzi dell'albero.



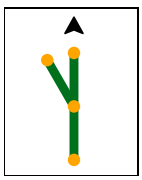
A destra ci sono due esempi di una regola di crescita e le corrispondenti prime due fasi di crescita.

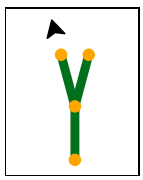


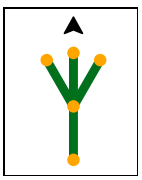
Il seguente albero digitale è cresciuto in 3 fasi di crescita:

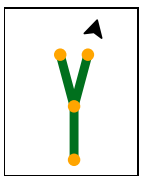


Secondo quale regola di crescita è cresciuto l'albero digitale?

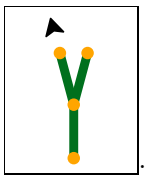
A) 

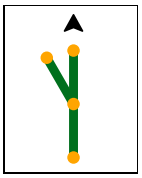
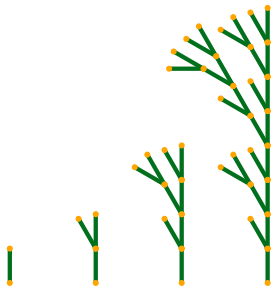
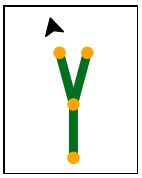
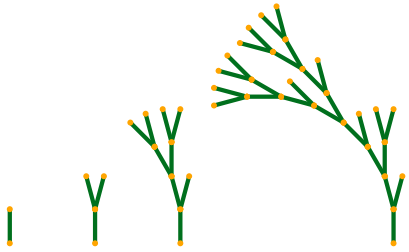
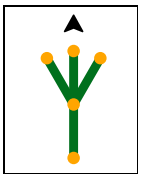
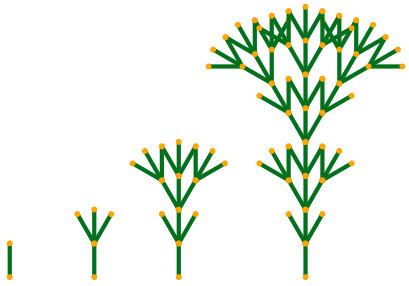
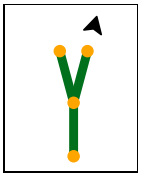
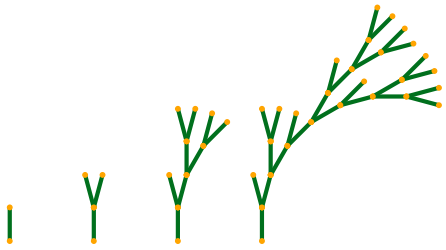
B) 

C) 

D) 

Soluzione

La risposta corretta è B)  .

Regola di crescita	3 fasi di crescita	Descrizione
		Il resto dell'albero è sempre aggiunto al ramo che punta verso l'alto, in linea retta. Forma così un tronco dritto con rami che puntano solo a sinistra.
		Il resto dell'albero viene sempre aggiunto al ramo superiore sinistro. L'albero quindi si inclina a sinistra.
		Il resto dell'albero è sempre aggiunto al centro, in linea retta. I due rami a sinistra e a destra formano una struttura uniforme e simmetrica.
		Il resto dell'albero viene sempre aggiunto al ramo superiore destro. L'albero quindi si inclina a destra.

Questa è l'informatica!

Nel compito si può vedere come l'applicazione ripetuta di una regola di crescita molto semplice possa creare figure complicate. Tali figure, che consistono di parti simili all'intera figura, sono chiamate *frattali*. I frattali sono molto spesso utilizzati dai computer, ad esempio per creare paesaggi o effetti speciali per i film.

In biologia, i cosiddetti *sistemi di Lindenmayer* (dal nome del biologo Aristid Lindenmayer) vengono utilizzati per simulare la crescita delle piante. Anche i frattali vengono creati in questo processo. Nel compito abbiamo visto quattro esempi molto semplici di un sistema di Lindenmayer.

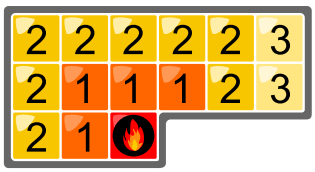
Gli alberi nel compito sono creati applicando una regola ad ogni pezzo di albero, e poi applicandola di nuovo ai pezzi di albero risultanti e così via. Tali processi sono chiamati *ricorsivi*. Il concetto degli algoritmi ricorsivi è importante nell'informatica. Con questo tipo di algoritmo è possibile descrivere molte cose complicate in modo molto semplice.

Parole chiave e siti web

- Frattale: `https://it.wikipedia.org/wiki/Frattale`
- Sistema di Lindenmayer: `https://it.wikipedia.org/wiki/Sistema_di_Lindenmayer`
  `http://paulbourke.net/fractals/lsys/`
- Algoritmo ricorsivo: `https://it.wikipedia.org/wiki/Algoritmo_ricorsivo`

35. Riscaldamento a pavimento

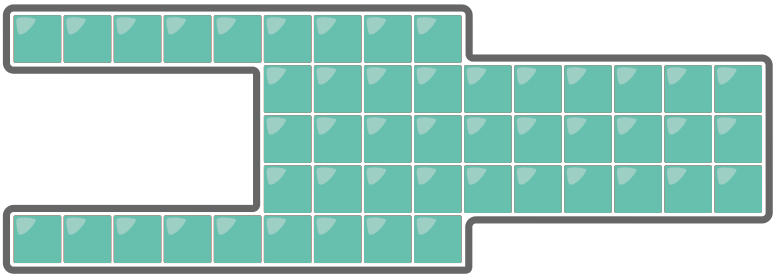
A Luis non piace vestirsi al mattino nel bagno freddo, quindi vuole che nella nuova casa venga installato il riscaldamento a pavimento. Il tecnico del riscaldamento gli consiglia l'innovativo riscaldamento a pavimento «hotspot»: un hotspot viene installato direttamente sotto una piastrella. Se l'hotspot è acceso, la piastrella è immediatamente calda.



In un minuto il calore si diffonde su tutte le piastrelle adiacenti, cioè tutte le piastrelle che toccano la piastrella già riscaldata su un bordo o un angolo. I numeri su ogni piastrella indicano dopo quanti minuti è calda.

Luis vuole far installare 4 hotspot nel suo nuovo bagno in modo che tutte le piastrelle si riscaldino il più velocemente possibile quando vengono accese.

Sotto quali 4 piastrelle il tecnico del riscaldamento deve installare i 4 hotspots ?

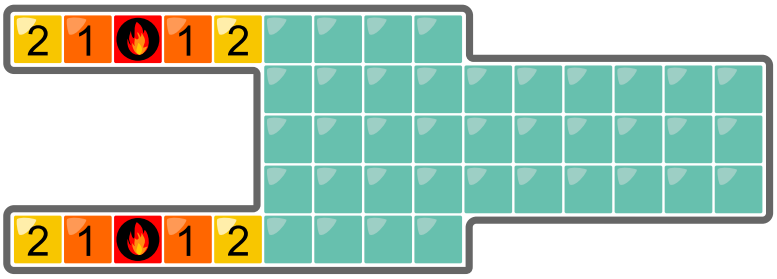


Soluzione

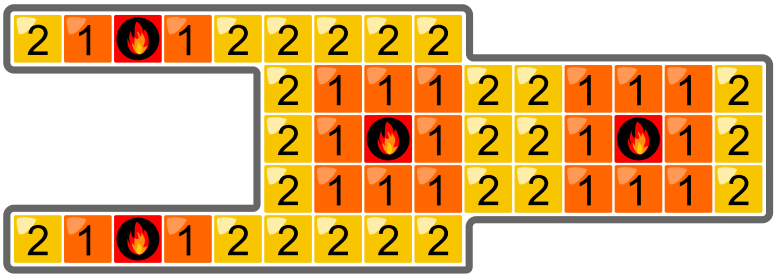
Se i 4 hotspot sono installati come mostrato nell'immagine sottostante, tutte le piastrelle del bagno si riscalderanno entro 2 minuti dall'accensione.

Questo è ottimale, perché è impossibile riscaldare tutte le piastrelle in 1 minuto con 4 punti caldi. Ogni hotspot può riscaldare infatti un massimo di 9 piastrelle nel primo minuto, cioè la propria piastrella e fino a 8 piastrelle intorno ad essa. Quindi 4 punti caldi insieme riscaldano un massimo di $4 \cdot 9 = 36$ piastrelle nel primo minuto. Il bagno ha 48 piastrelle in totale. Quindi 1 minuto non è sufficiente. Ma con 2 minuti potrebbe funzionare, visto che teoricamente fino a $4 \cdot 25 = 100$ piastrelle potrebbero essere riscaldate.

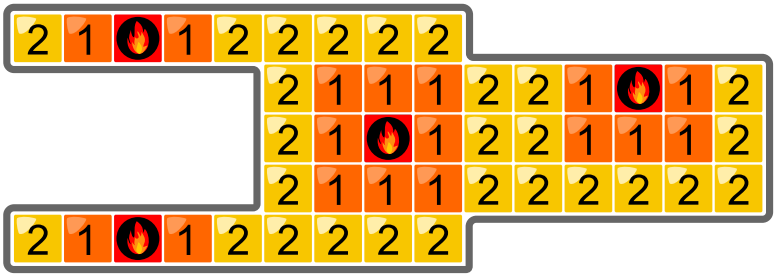
È una buona idea iniziare con i due corridoi a sinistra quando si distribuiscono gli hotspot. Con un punto caldo al centro di ogni corridoio, tutte le piastrelle del corridoio vengono riscaldate in 2 minuti:

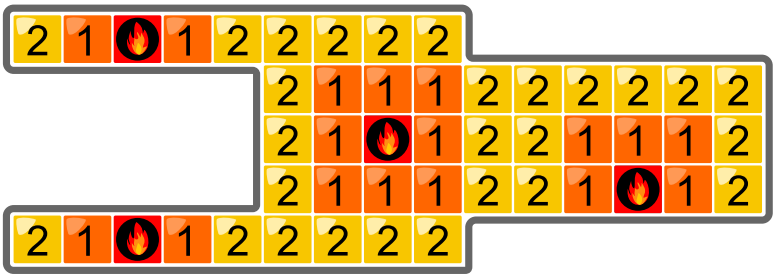


Possiamo poi collocare gli altri due hotspot in questo modo:



Sono possibili anche i seguenti due collocamenti:





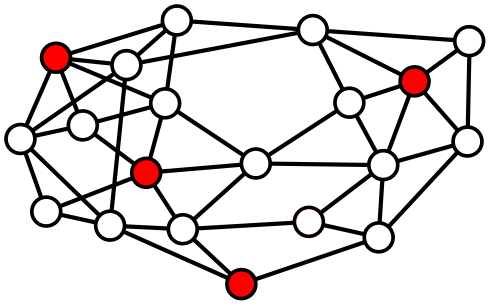
Se il bagno avesse una forma diversa, 2 hotspot potrebbero essere sufficienti per riscaldare l'intero bagno in 2 minuti con la stessa area.

Questa è l'informatica!

Il problema risolto in questo compito è legato ad un ben noto problema di ottimizzazione: Qui cerchiamo un piccolo gruppo di *nodi* in un *grafo* chiamato *insieme dominante*.

Un insieme dominante è definito come segue: Ogni nodo del grafo deve essere contenuto nel insieme dominante o avere un vicino che è contenuto nel insieme dominante. Le piastrelle del bagno possono essere interpretate come nodi. I nodi sono collegati con archi quando la piastrella vicina viene riscaldata dopo un minuto. Un set dominante del grafo risultante indica quindi i luoghi in cui è possibile posizionare gli hotspot per riscaldare il bagno in 2 minuti.

In generale è molto difficile trovare un insieme dominante minimo. Per i grafi speciali esistono algoritmi efficienti. Il disegno seguente mostra un esempio. Come si può vedere, ogni nodo bianco è vicino ad almeno un nodo rosso. Quindi i nodi rossi sono un insieme dominante.



Un'applicazione tipica è il posizionamento di hotspot WiFi in un grande edificio. I nodi del grafo sono le singole stanze. Due di esse sono adiacenti nel grafo se entrambe le stanze si trovano nel raggio d'azione di un hotspot. Le stanze che formano un insieme dominante minimo sono luoghi adatti per gli hotspot.

Parole chiave e siti web

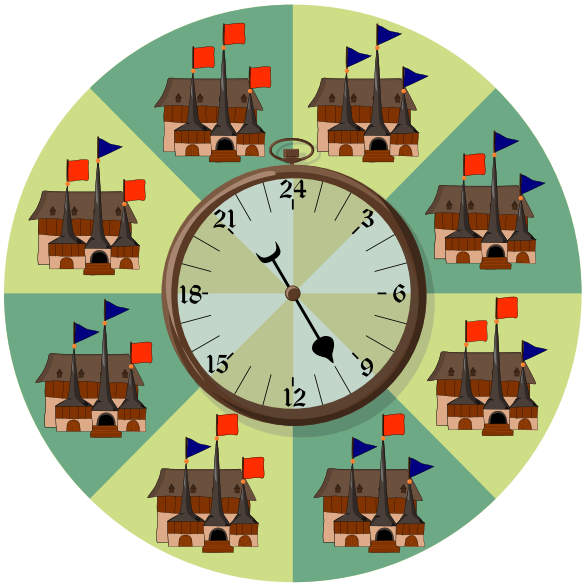
- Insieme dominante

36. Castori rilassati

In un villaggio idilliaco, i castori sono molto rilassati nel gestire il loro tempo. Suddividono la giornata in soli 8 periodi di 3 ore ciascuno. Il periodo di tempo attuale è indicato dal municipio con tre bandiere, come mostrato nell'immagine sottostante. Si utilizzano due diversi tipi di bandiere, un quadrato rosso e un triangolo blu.

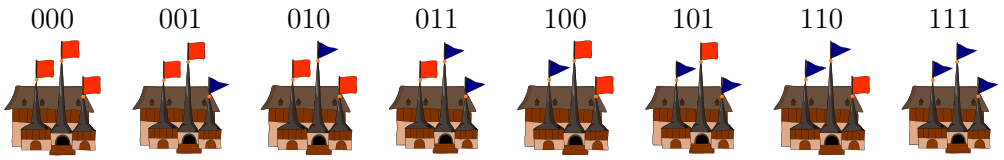
La sistemazione attuale richiede un solo cambio di bandiera a quasi tutte le transizioni. Solo a mezzanotte devono essere cambiate tre bandiere contemporaneamente. I castori vogliono introdurre una sistemazione più conveniente, dove bisogna cambiare una sola bandiera alla volta.

Trova una sistemazione più conveniente. Disegna gli schemi delle tre bandiere vicino ad ogni orario.



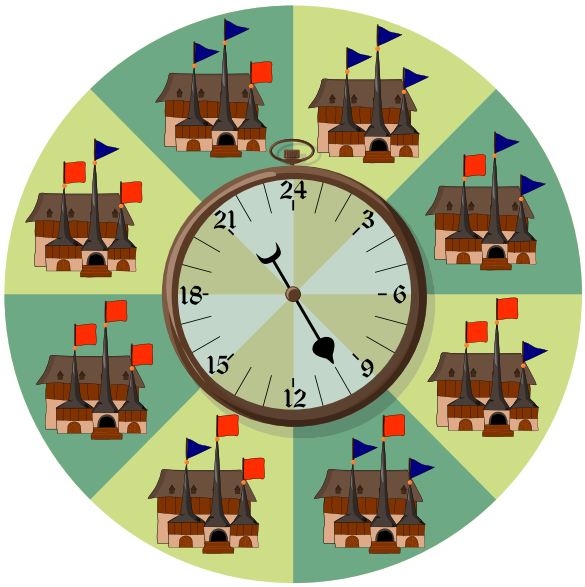
Soluzione

Gli 8 schemi possono essere rappresentati anche da numeri binari a tre cifre: 0 sta per un quadrato rosso e 1 per un triangolo blu.



Quindi gli 8 schemi sono 000, 001, 010, 011, 011, 100, 101, 101, 110, 111, ecc. Ora dobbiamo organizzare questi numeri in modo tale che tutti i numeri adiacenti differiscano solo in un posto, così come il primo e l'ultimo numero.

Questo può essere ottenuto per tentativi ed errori. Una possibile soluzione è 111, 011, 001, 001, 101, 101, 100, 000, 010, 110. Ecco l'orologio corrispondente:



Si trova sistematicamente una soluzione con il seguente metodo:

Per prima cosa, guardiamo solo i numeri che iniziano con due zeri, cioè 000 e 001. Qui ci sono due possibili configurazioni, che soddisfano entrambe la condizione sopra descritta. Scegliamo 000, 001.

Ora scriviamo di nuovo questi due numeri in ordine inverso (001, 000), ma cambiamo la seconda cifra da 0 a 1 (011, 010). Si ottiene così la sequenza dei numeri 000, 001, 011, 010, che soddisfa di nuovo la condizione.

Scriviamo questa nuova sequenza di numeri di nuovo all'indietro, ma cambiamo la prima cifra da 0 a 1 ovunque, così otteniamo 000, 001, 011, 011, 010, 110, 111, 111, 101, 100, che soddisfa di nuovo la condizione. Quindi abbiamo la soluzione desiderata.

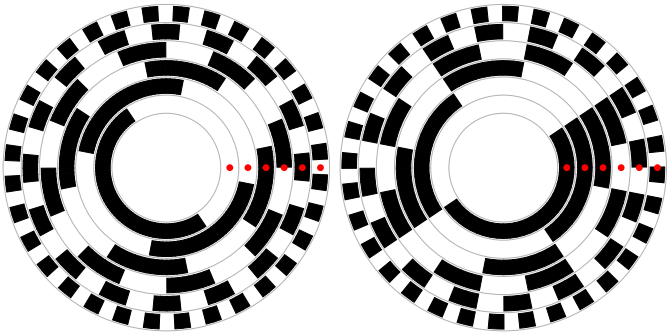
Questo metodo (invertendo la sequenza di numeri esistente e cambiando la cifra successiva più alta da 0 a 1) può essere continuato per tutto il tempo che si desidera per ottenere tali composizioni per

quante bandiere si desidera, invece che solo tre. Si può considerare il motivo per cui questo metodo funziona sempre e perché vengono utilizzati tutti i possibili modelli.

Questa è l'informatica!

Una tale composizione di numeri binari si chiama *codice di Gray* e ha molte applicazioni. Ad esempio, il fatto che solo un bit cambi tra numeri adiacenti può aiutare a risparmiare energia. In ogni caso, cambiare più bit richiede più energia, e con la normale enumerazione ascendente dei numeri binari, molti bit cambiano molto spesso allo stesso tempo.

Una famosa applicazione del codice di Gray in ingegneria è la misura degli angoli di un giradischi. Disegniamo il codice di Gray sul disco come mostrato nella figura in basso a sinistra, bianco per 0 e nero per 1. I punti rossi sono sensori posizionati su una linea e possono distinguere tra bianco e nero. I sensori possono così leggere un numero binario (una parola in codice) che codifica l'angolo di rotazione attuale del disco.



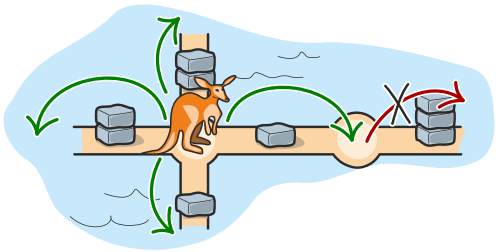
Nella figura a sinistra vediamo che il quarto sensore si trova esattamente al limite tra il bianco e il nero. Quindi il sensore legge o 001010 o 001110, entrambe le opzioni sono accettabili, in quanto l'angolo reale è esattamente al centro. Se non abbiamo un codice di Gray, il tutto sembra molto peggio. Guardiamo il normale codice binario nella figura a destra. Qui le parole di codice 111010 e 111001 si susseguono. Se i sensori sono esattamente nel mezzo, gli ultimi due sensori non possono decidere tra il bianco e il nero, quindi si potrebbe leggere il numero 111011, che è già a una certa distanza. Nel caso peggiore, i sensori si troverebbero al confine tra la parola di codice completamente bianca 000000 e la parola di codice completamente nera 111111, nel qual caso ogni sensore può passare arbitrariamente da 0 a 1, il che rende la misura dell'angolo completamente inutilizzabile.

Parole chiave e siti web

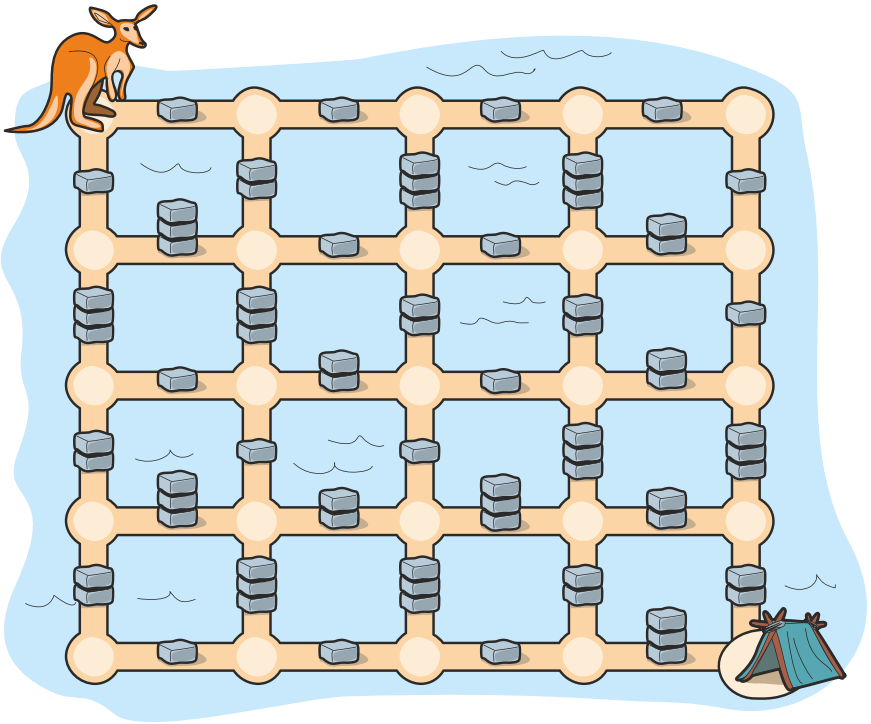
- Codice di Gray: `https://it.wikipedia.org/wiki/Codice_Gray`

37. Canguro salterino

Un canguro vuole tornare a casa . Può solo saltare seguendo il percorso e raggiungere l'incrocio successivo con un unico grande salto. Ad un incrocio può saltare a destra, a sinistra, in alto o in basso. Non può saltare sopra una pila di 3 pietre.



Il canguro vuole tornare a casa nel modo più corto.

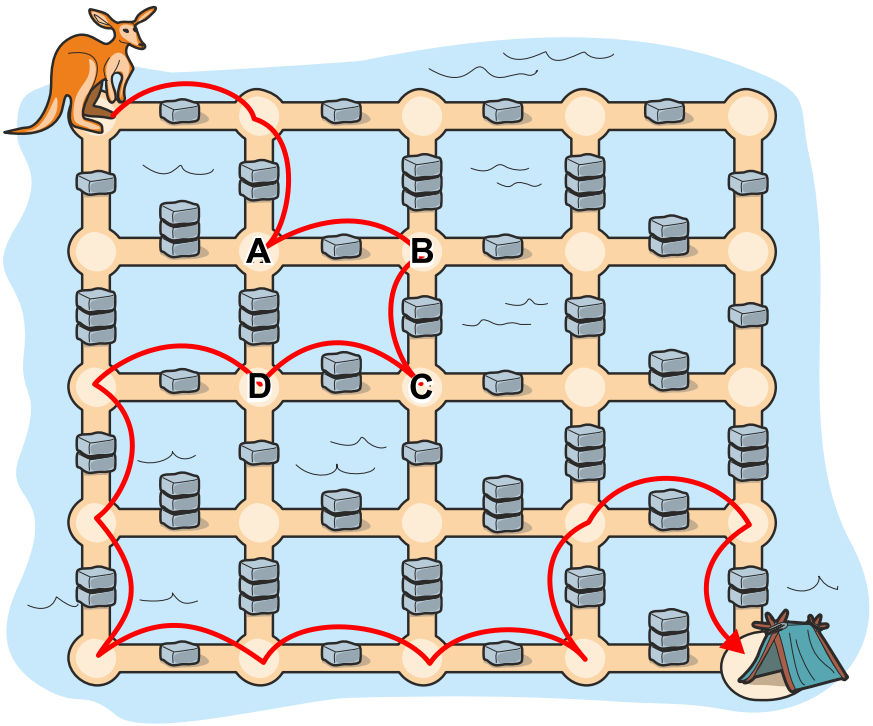


Quanti salti deve fare il canguro per tornare a casa con il percorso più corto?

A) 10 salti
B) 11 salti
C) 12 salti
D) 13 salti
E) 14 salti
F) 15 salti
G) 16 salti
H) 17 salti
I) 18 salti
J) 19 salti
K) 20 salti

Soluzione

La risposta corretta è E) 14 salti:



Il modo più semplice per iniziare la ricerca è quello di partire dalla destinazione. Si vedrà presto che c'è solo un percorso possibile di 9 salti dalla destinazione fino al punto D. Ora si deve solo trovare la via più breve dall'inizio al punto D. Con due passi il canguro raggiunge il punto A, un punto vicino al punto D. Ma non può saltare direttamente da A a D, perché c'è una pila di 3 pietre in mezzo. La deviazione più corta da A a D è via B e C, che richiede 3 salti. In totale il canguro ha bisogno di $2 + 3 + 9 = 14$ salti e tutti gli altri percorsi sono più lunghi.

Questa è l'informatica!

Per trovare un percorso, si può procedere come segue: Si cammina passo dopo passo seguendo il percorso che si vuole. Una volta raggiunto un vicolo cieco, dove tutte le direzioni sono bloccate o che portano a un punto già visitato del sentiero, si torna indietro fino a quando si trova una scelta alternativa di direzione, e poi si continua a provare.

Questo approccio è conosciuto nell' informatica come «*backtracking*» (inglese per tornare indietro). Viene utilizzato nell' informatica in vari algoritmi. Può essere utilizzato per trovare soluzioni di puzzle, sudoku o altri problemi di ottimizzazione combinatoria.

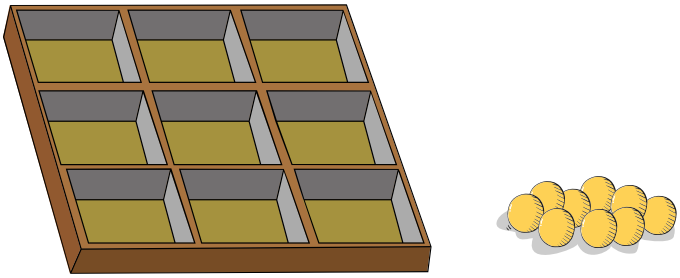
Il compito dimostra che a volte è più efficiente cercare una soluzione da dietro. Questa si chiama *ricerca all'indietro*. In questo caso, è necessario un minore backtracking perché alla fine non ci sono più opzioni. In generale non è possibile dire se una *ricerca in avanti* o all'indietro sia migliore, dipende dal problema concreto.

Parole chiave e siti web

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

38. Scomparti e biglie

Hira ha una scatola divisa in 9 scomparti e un numero illimitato di biglie:



Hira mette le biglie negli scomparti della scatola. Rispetta le seguenti regole:

- In ogni scomparto mette al massimo una biglia.
- In ogni riga e in ogni colonna il numero di biglie alla fine è pari.

Quanti schemi diversi può creare Hira con la scatola e le biglie?

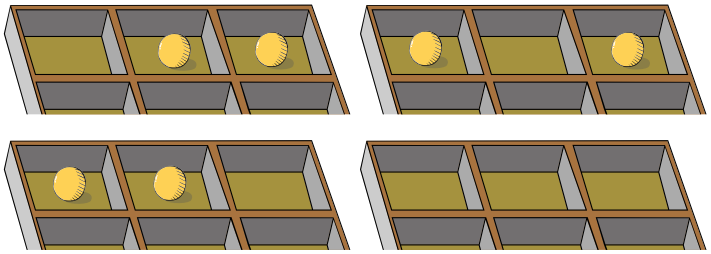
(La scatola non può essere ruotata. Lo schema con una sola biglia nell'angolo superiore sinistro è diverso da quello con una sola biglia nell'angolo superiore destro.)

A) 12
B) 16
C) 64
D) 512

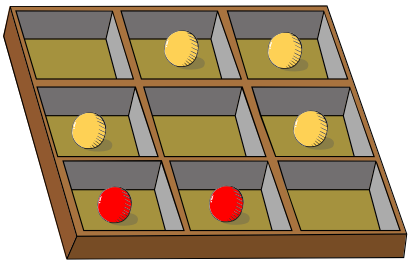
Soluzione

La risposta corretta è: B) 16.

In quanti modi Hira può riempire la prima riga? Ci deve essere un numero pari di biglie nella prima riga, cioè 0 o 2, quindi ci sono 4 modi per riempire la prima riga:



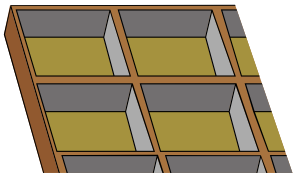
Allo stesso modo Hira ha 4 possibilità per riempire la seconda riga. Ma dopo non può scegliere nulla, perché anche le tre colonne devono contenere un numero pari di biglie. Se c'è un numero dispari di biglie nelle due righe superiori (cioè esattamente una biglia), Hira deve collocare una biglia nella terza riga di questa colonna, come avviene nelle prime due colonne dell'esempio seguente (biglie rosse):



Se le prime due righe di una colonna contengono un numero pari di biglie (cioè 0 o 2), non deve mettere una biglia nella terza riga di questa colonna, come avviene nella terza colonna dell'esempio sopra.

Poiché la scelta della prima riga è completamente indipendente dalla scelta della seconda, Hira ha 4 scelte per la prima riga e per ognuna di queste scelte ha di nuovo 4 scelte per la seconda riga. Pertanto ha un totale di $4 \cdot 4 = 16$ possibilità.

Una seconda opzione per il conteggio delle possibilità è la seguente: Si inizia guardando una parte 2×2 della scatola.

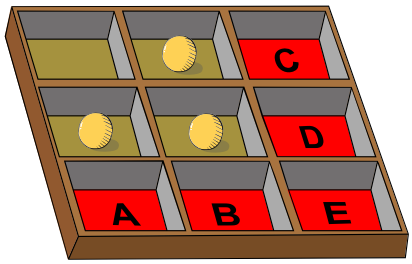


In questa parte ci sono 4 scomparti e ognuno può contenere una biglia o no. Quindi ci sono $2^4 = 16$ modi diversi per riempire questa parte.

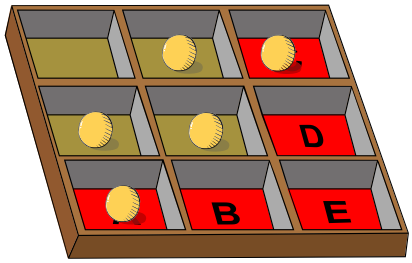
Un'osservazione importante è la seguente: Dopo che le biglie sono state collocate in questa parte della scatola, Hira non ha altra scelta per riempire gli scompartimenti rimanenti. Per ogni scomparto

sul bordo destro o nella riga inferiore, Hira deve posizionare una biglia o lasciarla fuori in modo che il numero sia pari.

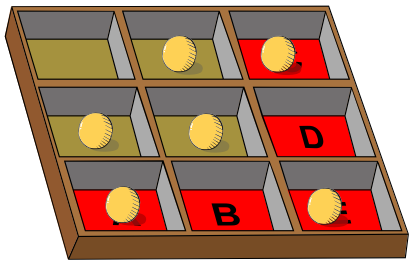
Ad esempio, Hira potrebbe riempire la parte 2×2 considerata come segue:



Poiché la prima colonna contiene una sola biglia, Hira deve mettere una biglia nello scomparto A in modo che il numero di biglie nella prima colonna sia pari. Nella seconda colonna c'è già un numero pari di biglie e quindi Hira non deve mettere una biglia nello scomparto B. Con argomenti simili si può vedere che lo scomparto D deve rimanere vuoto e Hira deve mettere una biglia in C.



Il numero di biglie in $A + B$ è quindi pari esattamente quando il numero di biglie nella parte 2×2 è pari. Esattamente lo stesso vale per la somma $C + D$. Se queste due somme sono pari, lo scomparto E può e deve rimanere vuoto; se entrambe sono dispari, Hira può e deve mettere una biglia nello scomparto E.



Questo dimostra che Hira può mettere le biglie negli scomparti della scatola in 16 modi diversi.

Questa è l'informatica!

Un importante compito dell'informatica è quello di trasferire i dati in modo sicuro. Un modo per garantire la sicurezza della trasmissione dei dati contro gli errori di trasmissione è quello di *verificare la parità*.

Alla fine del messaggio viene calcolato un bit di parità in base al messaggio da trasmettere e viene aggiunto al messaggio. Quando il messaggio viene ricevuto, il bit di parità può essere ricalcolato. Se non corrisponde al bit di parità trasmesso, è noto che c'è stato un errore di trasmissione.

In questo compito, gli scomparti dell'ultima riga e dell'ultima colonna servono come bit di parità. Se i numeri delle biglie negli scomparti sono stati trasmessi come messaggio, il destinatario può calcolare i totali delle righe e i totali delle colonne. Se questi non sono pari, il destinatario può dire a Hira che c'è stato un errore di trasmissione.

Un'altra competenza dell'informatica è la capacità di enumerare tutte le soluzioni con specifiche proprietà e quindi di determinarne il numero.

Parole chiave e siti web

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

A. Autori dei quesiti

Serge Adam
Faisal Al-Sudani
Tony René Andersen
Michael Barot
Wilfried Baumann
Carlo Bellettini
Linda Björk Bergsveinsdóttir
Daniela Bezáková
Maksim Bolonkin
Andrey Brodnik
Lucia Budinská
Špela Cerar
Sarah Chan
Marios O. Choudary
Kris Coolsaet
Valentina Dagienė
Tolmantas Dagys
Christian Datzko
Susanne Datzko
Amirmohammad Djazbi
Hanspeter Erni
Nora A. Escherle
Lidia Feklistova
Fabian Frei
Gerald Futschek
Jens Gallenbacher
Christian Giang
Tom Grubb
Yasemin Gulbahar
Husnul Hakim
Mathias Hiron
Juraj Hromkovič
Alisher Ikramov
Thomas Ioannou
Tiberiu Iorgulescu
Takeharu Ishizuka
Mile Jovanov
Ungyeol Jung
Vaidotas Kinčius
Sophie Koh
Dennis Komm
Ritambhra Korpal
Chia-Yi Ku
Regula Lacher
Taina Lehtimäki
Marielle Léonard
Judith Lin
Lynn Liu
Matija Lokar
Vu Van Luan
Hiroki Manabe
Pedro Marcelino

Hamed Mohebbi
Kwangsik Moon
Anna Morpurgo
Xavier Muñoz
Hiroyuki Nagataki
Vania Natali
Rana R. Natawigena
Tom Naughton
Ágnes Erdősné Németh
Andrei Nicolicioiu
Dejan Ozebek
Gabriel Parriaux
Jean-Philippe Pellet
Melinda Phelps
Margot Phillipps
Hannah Piper
Wolfgang Pohl
Prathyush Ponnekanti
Raymond Chandra Putra
Susannah Quidilla
Pedro Ribeiro
Chris Roffey
Peter Rossmanith
Eljakim Schrijvers
Vipul Shah
Fei Shang
Wenpan Sheng
Maiko Shimabuku
Timur Sitdikov
Emil Stankov
Preethi Sudharsha
Maciej M. Sysło
Congyu Tian
Peter Tomcsányi
Monika Tomcsányiová
Meng-ting Tsai
Jiří Vaníček
Troy Vasiga
Fan Wang
Michael Weigend
Jonas Winckler
Michal Winczer
Yang Xing
Khairul Anwar Mohamad Zaki
Binru Zhi

B. Sponsoring: concorso 2020

http://www.haslerstiftung.ch/

http://www.baerli-biber.ch/

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

Standortförderung beim Amt für Wirtschaft und Arbeit Kanton Zürich

i-factory (Musée des transports, Lucerne)

http://www.ubs.com/

http://www.oxocard.ch/
OXOcard
OXON

https://educatec.ch/
educaTEC

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

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.phlu.ch/`
Pädagogische Hochschule Luzern

`https://www.fhnw.ch/de/die-fhnw/hochschulen/ph`
Pädagogische Hochschule FHNW

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

`https://www.zhdk.ch/`
Zürcher Hochschule der Künste

C. Ulteriori offerte

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.