



INFORMATIK-BIBER SCHWEIZ CASTOR INFORMATIQUE SUISSE CASTORO INFORMATICO SVIZZERA

Quesiti e soluzioni 2024

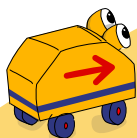
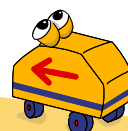
7^o e 8^o anno scolastico



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

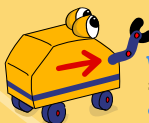
A cura di:

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



010100110101011001001001
010000010010110101010011
010100110100100101000101
0010110101010101101010011
0100100101001001001001001

SS! I



www.svia-ssie-ssii.ch
schweizerischerverein für informatik in d
erausbildung // société suisse pour l'infor
matique dans l'enseignement // società sviz
zera per l'informatica nell'insegnamento





Hanno collaborato al Castoro Informatico 2024

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

Capo progetto: Nora A. Escherle

Un particolare ringraziamento per il lavoro sui quesiti del concorso Svizzero va a:

Regula Lacher: ETH Zürich, Ausbildungen- und Beratungszentrum für Informatikunterricht

Christian Datzko: Gymnasium Liestal

Reza Jabbar, Stephan Christos Dörries: Haute école pédagogique Vaud

Ethan Major, Jan Schönbächler: Lycée-Collège de l'Abbaye

Jacqueline Staub: Universität Trier, Germania

Kirsten Schlüter: Bayerisches Staatsministerium für Unterricht und Kultus, Germania

La scelta dei quesiti è stata svolta in collaborazione con gli organizzatori dei concorsi in Germania, Austria e Ungheria. Ringraziamo specialmente:

Wolfgang Pohl, Karsten Schulz, Franziska Kaltenberger, Margaretha Schlüter, Kirsten Schlüter, Michael Weigend : Bundesweite Informatikwettbewerbe (BWINF), Germania

Wilfried Baumann, Simon Bachl, Liam Baumann, Martin Kandlhofer, Andreas Zottl : Österreichische Computer Gesellschaft, Austria

Gerald Futschek, Lukas Lehner, Florentina Voboril: Technische Universität Wien, Austria

Zsuzsa Pluhár: ELTE Informatikai Kar, Ungheria

La versione online del concorso è stata creata su cuttle.org. Ringraziamo per la buona collaborazione:

Eljakim Schrijvers, Dave Oostendorp, Justina Oostendorp, Alieke Stijf, Kyra Willekes, Jo-Ann Bolten: cuttle.org, Olanda

Andrew Csizmadia: Raspberry Pi Foundation, Regno Unito

Per il supporto durante le settimane del concorso ringraziamo:

Gabriel Thullen: Collège des Colombières, Versoix

Eveline Moor: Società svizzera per l'informatica nell'insegnamento

I compiti di programmazione sono stati creati e sviluppati appositamente per la piattaforma online.

Desideriamo ringraziare le seguenti persone:

Jacqueline Staub: Universität Trier, Germania

Dirk Schmerenbeck: Universität Trier, Germania

Dave Oostendorp: cuttle.org, Olanda

Ringraziamo l'ETH per l'organizzazione e lo svolgimento della finale del Castoro:

Dennis Komm, Hans-Joachim Bückenbauer, Dr. Angélica Herrera Loyo, Andre Macejko, Moritz Stocker, Philip Whittington: ETH di Zurigo, Ausbildungen- und Beratungszentrum für Informatikunterricht

Per la correzione dei compiti finali:

Isabel de Dios, Finlay Fehlauer, Oliver Graf, Sven Grübel, Saskia Koller, Richard Královič, Alessandra Lacopino, Jasin Pathiranage Piyumi, Oliver Pitsch, Adeline Pittet, Adeline Richter, Leo Strelén,



Fabio Sutter, Kai Zürcher

Per la traduzione dei compiti finali in francese:

Jan Schönbächler: Lycée-Collège de l'Abbaye de St-Maurice

Christoph Frei: Chragokyberneticks (Logo Informatik-Biber Schweiz)

Andrea Leu, Maggie Winter: Senarclens Leu + Partner AG

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

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

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



INFORMATIK-BIBER SCHWEIZ
CASTOR INFORMATIQUE SUISSE
CASTORO INFORMATICO SVIZZERA

Il Castoro Informatico 2024 è stato organizzato dalla Società Svizzera per l'Informatica nell'Insegnamento (SSII) con il sostegno determinante della fondazione Hasler. Gli sponsor del concorso sono l'Ufficio per l'economia del cantone di Zurigo, il cantone di Berna, la Società Svizzera di Informatica, UBS e Cyon.

Questo quaderno è stato creato il 6 dicembre 2024 con il sistema per la preparazione di testi $\text{\LaTeX}$. 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'01.12.2024.



I quesiti sono distribuiti con Licenza Creative Commons Attribuzione – Non commerciale – Condividi allo stesso modo 4.0 Internazionale. Gli autori sono elencati a pagina 78.



Premessa

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

Il Castoro Informatico è il partner svizzero del Concorso «Bebras International Contest on Informatics and Computer Fluency» (<https://www.bebbras.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» (3^o e 4^o 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 2024 il Castoro Informatico della Svizzera è stato proposto a cinque differenti categorie d'età, suddivise in base all'anno scolastico:

- 3^o e 4^o anno scolastico («Piccolo Castoro»)
- 5^o e 6^o anno scolastico
- 7^o e 8^o anno scolastico
- 9^o e 10^o anno scolastico
- 11^o al 13^o anno scolastico

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

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

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

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



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

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

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

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

Per ulteriori informazioni:

Società Svizzera per l'Informatica nell'Insegnamento

SVIA-SSIE-SSII

Castoro Informatico

Masiar Babazadeh

<https://www.castoro-informatico.ch/it/kontaktieren/>

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



Indice

Hanno collaborato al Castoro Informatico 2024	i
Premessa	iii
Indice	v
1. Pizza Party	1
2. Il robot di Roberta	5
3. Cartes Ricca	9
4. Berukone	13
5. Fianco a fianco	17
6. Superbebras	21
7. La biblioteca di Holzdorf	25
8. Pettegolezzo	29
9. Buon compleanno	33
10. Tour forestale	37
11. Braccialetto più lungo	41
12. Catene di parole	45
13. Macchina dei mattoni	49
14. Macchina dei palloncini	53
15. Pip il pirata	57
16. Crittografia delle immagini	61
17. Giorni di sole	65
18. Frowny	69
19. Foglio Bagnato	75
A. Autori dei quesiti	78
B. Partner accademici	80
C. Sponsoring	81
D. Ulteriori offerte	82



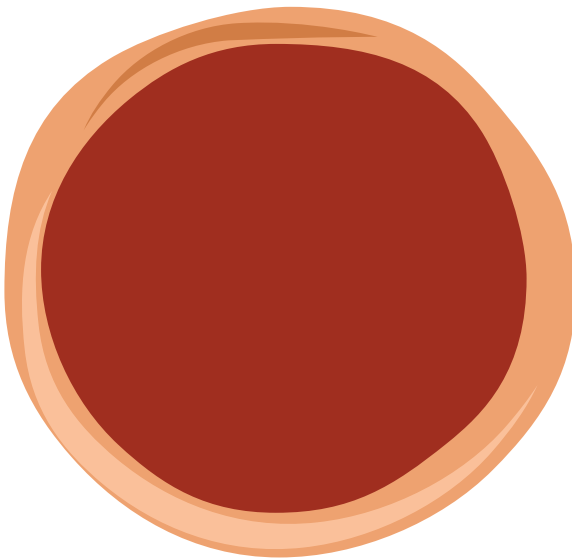
1. Pizza Party

John sta organizzando un pizza party, una festa dove si mangia la pizza. John sa quali ingredienti desiderano i suoi ospiti sulla pizza.

Alice:			
Bob:			
Cem:			
Dana:			

John vuole preparare una pizza con un totale di 3 ingredienti. Sceglie 3 ingredienti in modo da soddisfare il maggior numero possibile di desideri dei suoi ospiti.

Con quali 3 ingredienti John realizza il maggior numero di desideri?





Soluzione

Questa è la soluzione:



Per trovare la risposta giusta, si potrebbero provare tutte le combinazioni di tre componenti e contare quanti desideri soddisfa ogni combinazione. Tuttavia, esistono 20 combinazioni di questo tipo, quindi questa procedura è lunga, noiosa e soggetta a errori.

Ma non c'è nemmeno bisogno di pensare alle combinazioni. La combinazione che soddisfa il maggior numero di desideri è costituita dai tre ingredienti più desiderati singolarmente. È possibile trovare questi ingredienti organizzando la tabella del compito in modo diverso. Possiamo per esempio aggiungere una colonna separata per ogni condimento. La colonna contiene un 1 se il condimento è stato richiesto da un ospite e uno 0 se non lo è. Quindi possiamo procedere sommando i numeri di ogni colonna.

						
Alice	1	1	0	0	1	0
Bob	0	1	1	0	0	1
Cem	0	1	0	1	1	0
Dana	0	1	0	0	1	1
Somma	1	4	1	1	3	2

I funghi sono i più richiesti (4), seguiti dal formaggio (3) e dalle cipolle (2). Quando John mette questi tre ingredienti più popolari sulla pizza, esaudisce il maggior numero di desideri, per un totale di 9. Con nessun'altra combinazione di condimenti può esaudire più desideri.



Questa è l'informatica!

Il problema della selezione degli ingredienti della pizza in questo compito è un problema di *ottimizzazione*. John non sta cercando una qualsiasi combinazione di ingredienti per la sua pizza, ma la migliore - o meglio: una migliore, perché è concepibile che ci siano diverse combinazioni di ingredienti che soddisfano la maggior parte dei desideri.

Per trovare la soluzione migliore a un problema di ottimizzazione, occorre

- registrare i dati (qui: i desideri degli amici),
- calcolare nuovi dati (le frequenze dei desideri),
- tenere conto delle condizioni (deve essere infornata solo una pizza con esattamente tre ingredienti) e
- prendere una decisione (la scelta dei ingredienti).

Per prendere una decisione quando si risolve un problema di ottimizzazione, è necessario poter confrontare le possibili risposte o soluzioni tra loro. Per farlo, occorre una *misura* della bontà di una soluzione. In questo compito, il numero di desideri soddisfatti è la misura dell'ottimizzazione: più desideri vengono soddisfatti, meglio è. L'algoritmo di ottimizzazione è stato in grado di selezionare direttamente la soluzione con il valore migliore per la misura di ottimizzazione sommando i desideri per i singoli ingredienti. In questo caso, quindi, non è stato necessario confrontare le diverse soluzioni tra loro.

Gli algoritmi di ottimizzazione sono sviluppati in informatica per molti scopi, ad esempio per i processi di produzione nell'industria o per calcolare i percorsi dei veicoli di consegna dei pacchi o di altri servizi di consegna. Le misure di ottimizzazione possono essere molto diverse, come la velocità del processo produttivo o la lunghezza del percorso. Ma una cosa è sempre la stessa: bisogna pensare a come selezionare la soluzione migliore tra le tante possibili nel modo più abile possibile. Raramente è così facile come in questo compito.

Parole chiave e siti web

- Problema di ottimizzazione: https://it.wikipedia.org/wiki/Problema_diottimizzazione
- Procedura decisionale: https://it.wikipedia.org/wiki/Problema_decisionale

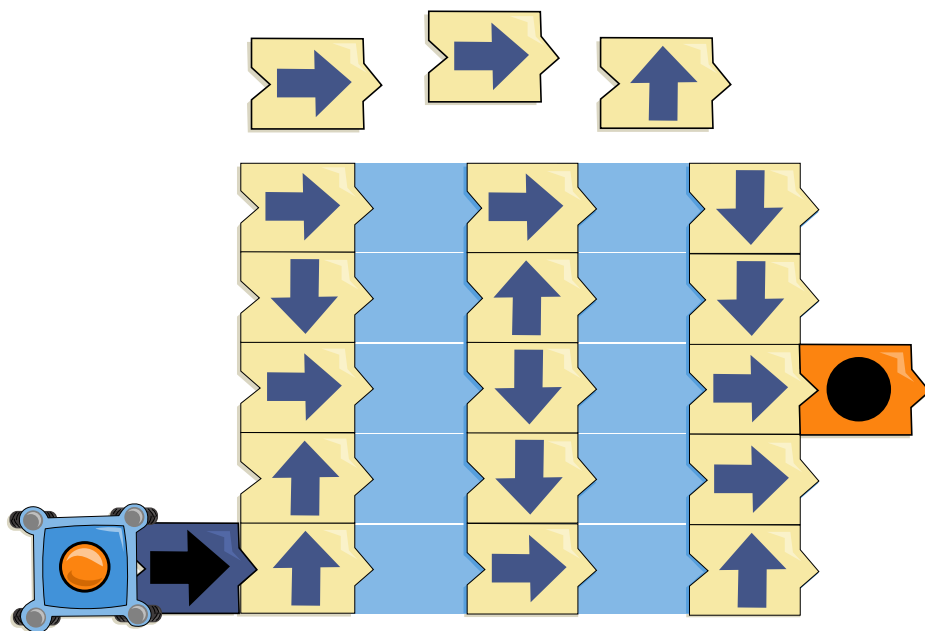




2. Il robot di Roberta

Il robot di Roberta  può muoversi solo su tessere speciali. È possibile utilizzare le carte per creare una mappa del percorso. Il robot inizia sempre sulla tessera di partenza  e deve raggiungere la tessera di destinazione . Da una tessera con una freccia, si muove nella direzione della freccia fino alla tessera successiva.

Di seguito è riportata una mappa del percorso. Il robot si sposta quindi prima a destra, poi in alto e di nuovo in alto. Poi vorrebbe girare a destra, ma non c'è nessuna carta! Il piano del percorso va dunque finito. Ci sono ancora 3 carte e 10 posti in cui possono essere posizionate.

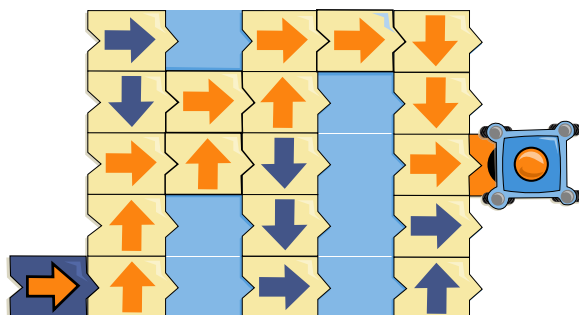


Posiziona le tre carte in modo che il robot di Roberta raggiunga la destinazione.

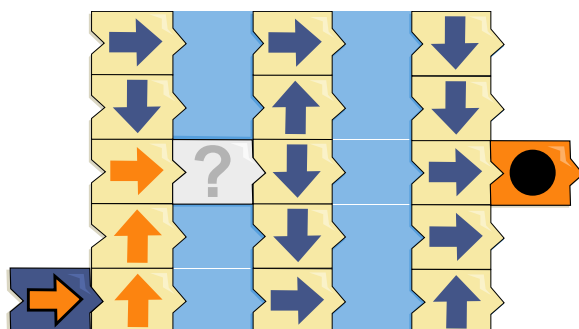


Soluzione

Questa è la soluzione:



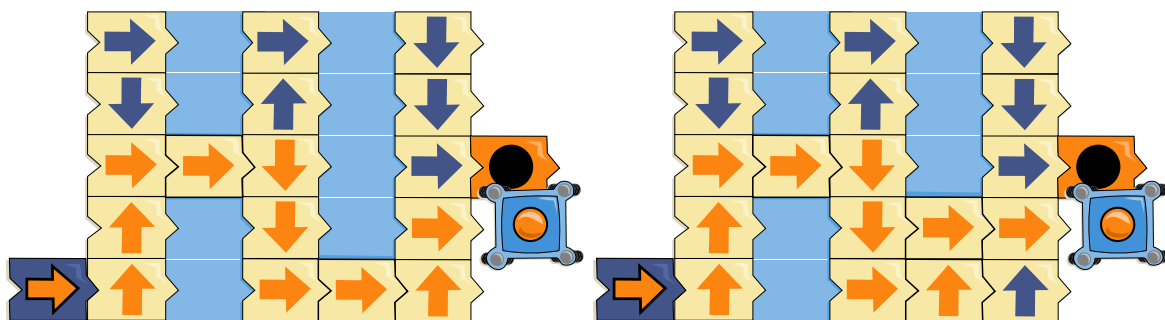
Come già indicato nel compito, il robot di Roberta si muoverà prima a destra, due volte in alto e di nuovo a destra. Una tessera deve essere posizionata nel punto contrassegnato con “?”



I due tipi di carte disponibili sono «in alto»  o «a destra» .

Se si selezionava , il robot si spostava a destra, poi due volte in basso e di nuovo a destra. A questo punto si può selezionare di nuovo  oppure  e poi .

In entrambi i casi, il robot atterrebbe sotto la destinazione:



Quindi dobbiamo per forza utilizzare la tessera . Ora solo  può essere selezionata come seconda tessera, perché l'unica  è già stata utilizzata. Il robot si sposterebbe quindi a destra, in alto, a destra e da lì in alto e a destra. A questo punto, l'ultima tessera disponibile  deve essere posizionata in modo che il robot possa spostarsi a destra, in basso due volte e a destra una volta per arrivare alla destinazione.

Esiste quindi un'unica soluzione corretta.



Per inciso, è possibile pensare a questa soluzione anche a ritroso, in modo più rapido: È possibile raggiungere la destinazione solo dalle prime tre posizioni possibili nella colonna azzurra a destra. Tuttavia, è possibile raggiungerla solo dall'alto, quindi  deve essere in alto a destra. Tuttavia, si può arrivare a destinazione solo dalle prime due tessere al centro, quindi per arrivarci bisogna posizionare un altro  nella colonna azzurra a sinistra.

Questa è l'informatica!

Di norma, questo problema si risolve come descritto nella spiegazione della risposta: si prova una strada e se non porta alla destinazione, si torna indietro e si prova un'altra strada. In informatica si parla di *ricerca in profondità*, perché prima si prova un percorso completo e solo dopo si prova quello successivo. Poiché spesso si deve tornare indietro di uno o più passi per provare una soluzione diversa, si parla di *backtracking*. In ogni caso, però, il problema viene risolto in avanti. Tuttavia, alcuni problemi si risolvono meglio all'indietro, guardando dall'obiettivo per capire come arrivarci. In realtà, questo tipo di problema non si presta bene a essere risolto all'indietro, perché mentre in avanti si sa sempre qual è la tessera successiva, all'indietro può capitare di arrivare da più tessere (la freccia punta in avanti e all'indietro bisogna guardare tutte le tessere per vedere se puntano nella direzione della tessera). In questo caso particolare, tuttavia, è facile riconoscere quale sia l'unica soluzione corretta.

Parole chiave e siti web

- Ricerca in profondità https://it.wikipedia.org/wiki/Ricerca_in_profondità
- Backtracking <https://it.wikipedia.org/wiki/Backtracking>





3. Cartes Ricca

Barbara colleziona carte con mostri, le «Ricca». Ecco le sue carte Ricca:



Le proprietà di un mostro Ricca sono indicate su ogni carta, ad esempio il nome (👤) o se il Ricca ha i denti (🦷). Le proprietà hanno dei valori. Nella scheda 2, ad esempio, 👤 ha il valore MONI e 🦷 ha il valore ✓: il Ricca sulla scheda si chiama quindi MONI e ha i denti.

Barbara si rende conto che sulle carte ci sono solo tre tipi diversi di valori. Li chiama «tipi»:

- Testo** Varie lettere concatenate
- Numero** cioè 1, 2, 3 ecc.
- si/no** I caratteri ✓ e X, che significano «sì» e «no».

Assegna i tipi corretti alle proprietà.

si/no	si/no	Testo	Numero	Numero
👤	👁️	🦊	🦋	🦷



Soluzione

				
Testo	Numero	Numero	si/no	si/no

Questa è la soluzione:

Questa tabella contiene i valori delle cinque proprietà per tutte e sei le carte:

Carta					
1	JOSI	3	2	X	✓
2	MONI	3	2	X	✓
3	KILI	2	2	✓	✓
4	BENI	2	2	X	✓
5	LORI	2	5	X	✓
6	PHIL	2	2	X	✓

Ora possiamo vedere dalla tabella:

- Tutti i valori della proprietà  sono di tipo **Testo**.
- Tutti i valori delle proprietà  e  sono di tipo **Numero**.
- Tutti i valori delle proprietà  e  sono di tipo **si/no**.

Questa è l'informatica!

Barbara ha notato che sulle sue carte Ricca vengono utilizzati diversi tipi di valori. Distinguere questi tipi può aiutare a organizzare le carte. Poiché Barbara sa che tutti i valori della proprietà  sono numeri, può confrontare il numero di gambe sulle carte Ricca e ordinare le carte in base al numero di gambe.

Una carta Ricca è come un record di dati memorizzato in un computer. Tuttavia, l'elettronica delle memorie dei computer è in grado di memorizzare solo 1 e 0. I programmi per computer utilizzano quindi anche «tipi di valore», che in informatica vengono chiamati *tipi di dati*, per determinare come il computer deve interpretare e manipolare questi 1 e 0. I tipi di dati possono anche essere utilizzati per verificare se e come i dati memorizzati possono essere combinati tra loro.

I tipi di valore di questo compito del castoro corrispondono a importanti tipi di dati della programmazione:

Testo è spesso chiamato “stringa” nei linguaggi di programmazione: I dati di questo tipo sono sequenze di caratteri (lettere, cifre e altri caratteri).

Numero corrisponde al tipo “numero intero”: I dati di questo tipo sono numeri interi, cioè ... -2, -1, 0, 1, 2, ...

si/no corrisponde al tipo “booleano”: questo tipo di dati riconosce solo due valori diversi, spesso chiamati “vero” e “falso” (1 e 0).



Parole chiave e siti web

Tipi di dato: https://it.wikipedia.org/wiki/Tipo_di_dato

Stringa : [https://it.wikipedia.org/wiki/Stringa_\(informatica\)](https://it.wikipedia.org/wiki/Stringa_(informatica))

Numero intero: [https://it.wikipedia.org/wiki/Numero_intero_\(informatica\)](https://it.wikipedia.org/wiki/Numero_intero_(informatica))

Booleano: [https://it.wikipedia.org/wiki/Booleano_\(informatica\)](https://it.wikipedia.org/wiki/Booleano_(informatica))





4. Berukone

Giochiamo a Berukone! Un puzzle di Berukone consiste in una griglia con dei numeri all'interno di alcune celle. Due numeri identici formano una coppia. Per risolvere il puzzle, bisogna collegare i due numeri con una linea per tutte le coppie. Questa linea di collegamento deve correre orizzontalmente o verticalmente da una cella all'altra, ma può ruotare di 90 gradi su di una cella. Una linea non può passare attraverso un altro numero o un'altra linea.

Purtroppo, ci sono enigmi di Berukone che non possono essere risolti.

Questo Berukone può essere risolto:

		1
1	2	
		2

Questo Berukone *non* può essere risolto:

2	1	
		2
	1	

Esattamente uno dei seguenti enigmi di Berukone non può essere risolto. Quale?

			3
3			
	2	1	2
1			

A)

3			3
		1	
	2		2
1			

B)

3			
		1	3
	2		
1			2

C)

			3
3		1	
	2		2
1			

D)

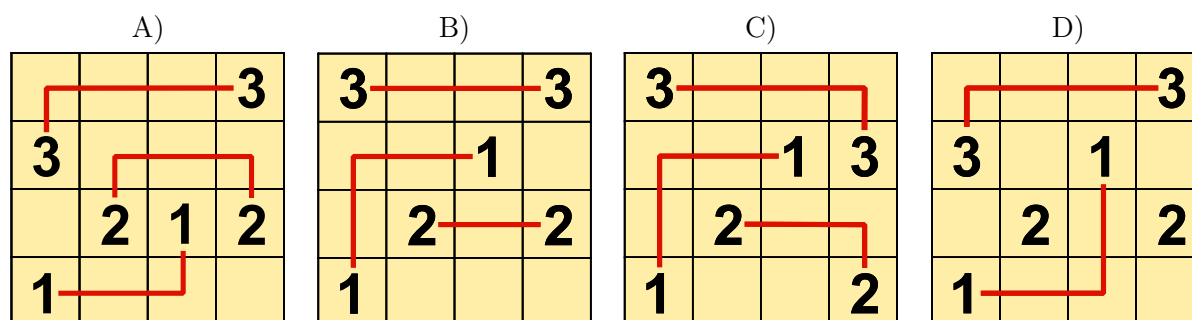


Soluzione

			3
3		1	
	2		2
1			

La risposta D è corretta:

Il Berukone D è l'unico dei quattro enigmi che non può essere risolto. Nel seguente modo invece si possono risolvere i puzzle A, B e C e cercare di risolvere il puzzle D:



Nel puzzle D non c'è modo di collegare i due numeri per tutte le coppie secondo le regole di Berukone. In particolare, può esistere una linea di collegamento conforme alle regole solo per una delle due coppie di 1 e 2. L'unica linea che può essere usata per collegare gli 1 senza che la linea passi per un altro numero è mostrata sopra. Tutte le possibili linee tra i numeri 2 che non passano per almeno un altro numero attraverserebbero la linea tra gli 1. Non c'è quindi modo di collegare i numeri 2 con una linea senza infrangere le regole.

Questa è l'informatica!

Per gli enigmi più semplici di Berukone in questo compito del castoro informatico è relativamente facile capire se possono essere risolti o meno e come possono essere risolti. Ma immaginate puzzle più grandi, con molti più numeri: in quel caso potrebbe non essere così facile trovare una soluzione. Potrebbero esserci molti modi per collegare le coppie di numeri e ci vorrebbe molto tempo per provarli tutti.

Si può però cercare di essere più intelligenti. Quando risolvete un enigma, potete procedere passo dopo passo: collegate le coppie una dopo l'altra o cercate di riempire i campi della griglia uno dopo l'altro. Non appena si può prevedere che il percorso attuale non porta più alla soluzione, significa che almeno un passo precedente era sbagliato. Allora si può annullare l'ultimo passo (e forse anche più passi), finché non si arriva a un punto in cui è possibile un passo alternativo che potrebbe ad una soluzione migliore. In altre parole, si torna indietro sul percorso tracciato con i passi precedenti per seguire una strada diversa.



In informatica, questo metodo è chiamato *backtracking*. Viene spesso utilizzato per consentire ai programmi informatici di risolvere automaticamente i problemi, siano essi reali o semplici enigmi. Gli algoritmi di backtracking funzionano come descritto sopra: cercano di costruire soluzioni passo dopo passo e annullano i passaggi se possono prevedere che il percorso attuale non porterà a una soluzione. La loro capacità di (a) prendere buone decisioni sul passo successivo e (b) prevedere se il percorso attuale può portare a una soluzione o meno è fondamentale per il loro successo. Gli algoritmi di backtracking devono spesso affrontare problemi difficili per i quali non sono note soluzioni specifiche. Nel 2010, alcuni ricercatori giapponesi hanno scoperto che la soluzione dei puzzle Arukone (una variante del Berukone, noto anche come Numberlink, in cui nessun campo può essere lasciato vuoto in una soluzione) è uno dei problemi più difficili conosciuti in informatica.

Parole chiave e siti web

- Backtracking: <https://it.wikipedia.org/wiki/Backtracking>
- Divide et impera: [https://it.wikipedia.org/wiki/Divide_et_impera_\(informatica\)](https://it.wikipedia.org/wiki/Divide_et_impera_(informatica))





5. Fianco a fianco

Ava ha cambiato scuola. Oggi incontra online i suoi amici della vecchia scuola. I suoi amici sono tutti seduti in fila, uno accanto all'altro.

Ogni bambino siede davanti alla propria telecamera. Tuttavia, in ogni immagine della telecamera sono visibili anche i bambini seduti accanto a loro.

Ecco come Ava vede i suoi amici sul suo schermo:



Come sono disposti gli amici uno accanto all'altro nella fila?





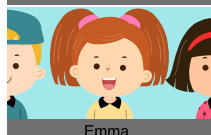
Soluzione

Questa è la soluzione:

Le immagini della telecamera rivelano chi è seduto accanto a chi.



Nell'immagine della telecamera in basso a destra dello schermo, non si vede nessun bambino a sinistra di James.



... è visibile nell'immagine in alto a sinistra dello schermo: Emma. La bambina che si riconosce a destra di James.



... si vede nell'immagine in basso a sinistra: Diana. Il bambino a destra di Emma.

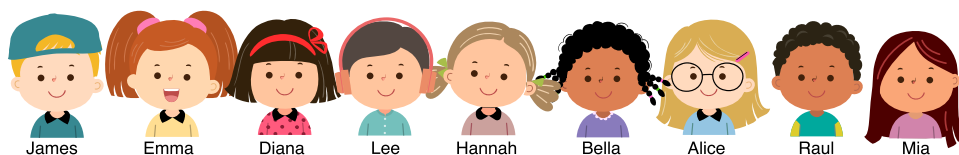
...

...



... è visibile in alto al centro dell'immagine: Mia. Non c'è nessun bambino alla destra di Mia. E il bambino a sinistra di Lee.

Nel complesso, le immagini della telecamera mostrano i nove amici seduti l'uno accanto all'altro:

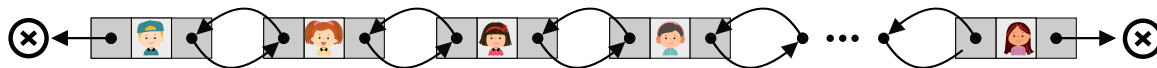


Questa è l'informatica!

Nelle immagini della telecamera, non solo è possibile vedere un bambino, ma anche riconoscere quali bambini si trovano a sinistra e a destra di esso. Gli informatici direbbero: le immagini contengono *referimenti* ai bambini accanto e le immagini sono (prevalentemente) *concatenate* in due direzioni.

Dal punto di vista dell'informatica, le immagini della telecamera in questo compito formano una *lista doppiamente concatenata*. Un elenco è generalmente una struttura di dati con nodi e riferimenti tra i nodi; i nodi contengono dati o *elementi di dati*. In una lista doppiamente concatenata, ogni nodo contiene un elemento, un riferimento al nodo precedente e un riferimento al nodo successivo. In questo compito, ogni immagine della telecamera (nodo) mostra il figlio (elemento) e i riferimenti alle immagini della telecamera e quindi anche agli elementi a destra e a sinistra del figlio.

Per trovare la risposta corretta, scorrete l'elenco dei bambini e seguite i riferimenti per mettere i bambini nell'ordine corretto. A differenza dell'elenco a collegamenti circolari (come nel compito «Il sonaglio di Oliver»), è possibile scorrere in entrambe le direzioni. Tuttavia, l'elenco ha un inizio (*capo*) e una fine. Nella nostra lista, James è a un capo e Mia alla fine dell'altro capo. I loro collegamenti a sinistra e a destra sono vuoti.



Un esempio di elenco a doppio collegamento è la cronologia del browser: È possibile saltare all'indietro o in avanti a piacimento. La pagina attualmente visualizzata (da cui si torna indietro nella cronologia) si trova in cima all'elenco.

Parole chiave e siti web

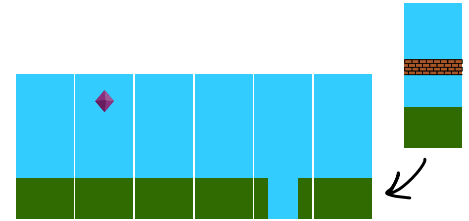
- Lista concatenata: https://it.wikipedia.org/wiki/Lista_concatenata



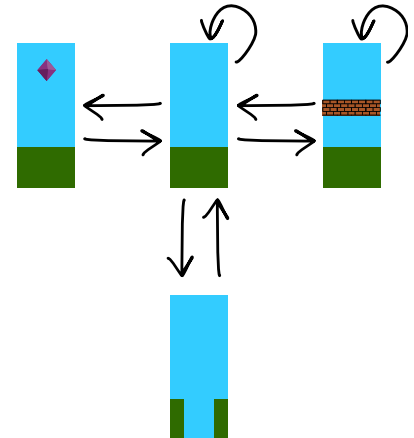


6. Superbebras

Nel gioco per computer «Superbebras», lo sfondo è costituito da una sequenza di tessere. Il computer aggiunge costantemente una nuova tessera sul lato destro della sequenza e contemporaneamente ne rimuove una sul lato sinistro. Questo sembra un movimento.



Per selezionare la tessera successiva, il computer utilizza questo diagramma. Nel diagramma, cerca la tessera precedente (sul lato destro) e le frecce che partono da essa. Quindi seleziona casualmente una tessera verso la quale punta una di queste frecce.

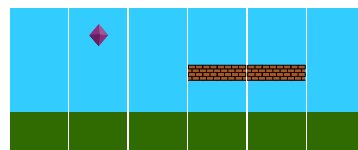


Ad esempio, dopo la tessera , il computer può selezionare nuovamente la tessera  o la tessera

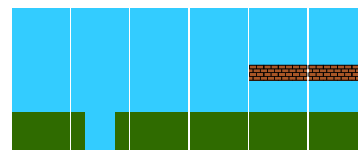


In uno sfondo Superbebras, l'ordine delle tessere deve quindi corrispondere al diagramma sopra.

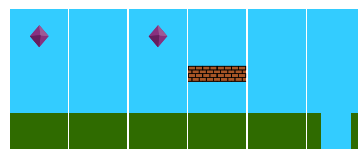
Una di queste immagini *NON* è uno sfondo di Superbebras. Quale?



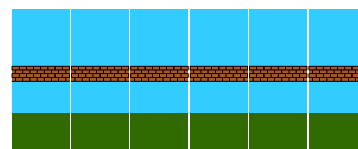
A)



B)



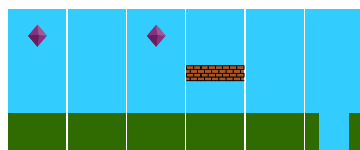
C)



D)

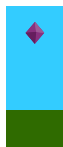


Soluzione

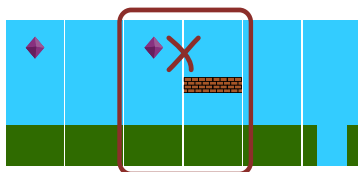


La risposta C è corretta:

L'immagine della risposta C non corrisponde al diagramma. Secondo il diagramma, una tessera con



un diamante può essere seguita solo da una tessera . Questo non è il caso dell'immagine della risposta C:



Tutte le altre immagini corrispondono al diagramma.

Esistono diversi modi per determinare se un'immagine corrisponde o meno al diagramma. Una procedura semplice consiste nello scorrere le tessere dell'immagine da sinistra a destra e verificare sempre se per due tessere adiacenti è presente una freccia nel diagramma.

Tuttavia, si può anche procedere in modo più mirato: Passare in rassegna le tessere del diagramma una dopo l'altra, iniziando da quelle da cui proviene una sola freccia. Una freccia è una condizione: $A \rightarrow B$ significa «dopo la tessera A deve venire la tessera B». Si verifica quindi se l'immagine soddisfa questa condizione.

Questa è l'informatica!

Alcuni giochi per computer, come i giochi *endless runner*, hanno uno sfondo che si muove orizzontalmente, creando l'illusione che il giocatore si muova in un mondo fantastico. A volte lo sfondo non è un'immagine costante, ma viene generato automaticamente dal computer. Si tratta della cosiddetta *generazione procedurale*. Di solito gli oggetti più piccoli vengono combinati per creare un'ampia varietà di sfondi.

Tuttavia, la combinazione di oggetti non può essere completamente casuale. Deve seguire determinate regole per evitare situazioni che il giocatore non può affrontare. Nel compito, queste regole sono rappresentate da un diagramma di tessere e frecce. Tali diagrammi, in cui gli oggetti sono collegati da frecce, sono noti in informatica come *grafi direzionati* (più precisamente: come la loro rappresentazione grafica). Gli oggetti vengono quindi indicati come nodi e le frecce come bordi diretti. I grafi diretti sono utilizzati in informatica per modellare le relazioni tra gli oggetti.

Gli *automi a stati finiti* fanno un ulteriore passo avanti: hanno degli stati, uno stato iniziale, degli stati finali e delle regole per passare da uno stato all'altro. È simile a un grafo diretto, con la



differenza che ci sono nodi speciali per stati speciali. Un automa di questo tipo può anche essere descritto graficamente, cioè attraverso il suo diagramma di transizione di stato. Gli automi finiti sono utilizzati in informatica per modellare processi semplici o per descrivere sequenze facili da costruire, come le sequenze di tessere di questo compito.

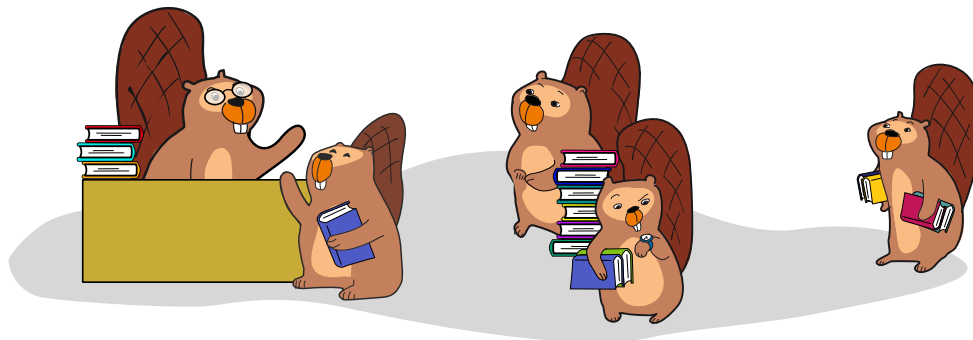
Parole chiave e siti web

- Gioco endless runner: https://it.wikipedia.org/wiki/Endless_runner
- Automa a stati finiti: https://it.wikipedia.org/wiki/Automa_a_stati_finiti





7. La biblioteca di Holzdorf

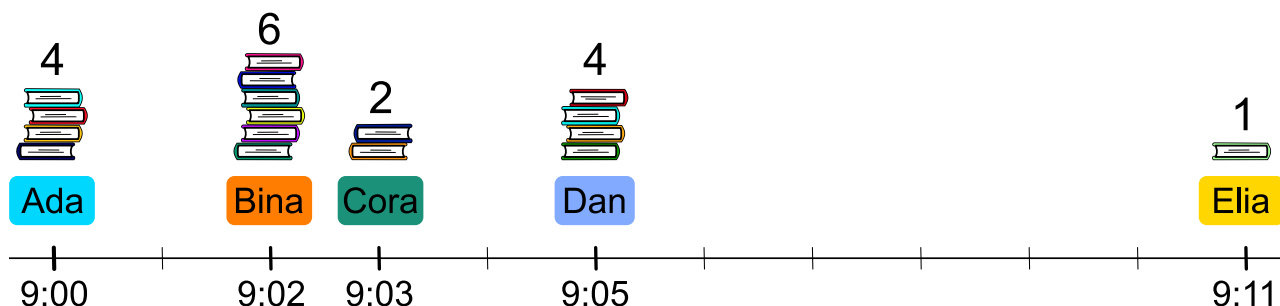


I castori di Holzdorf sono lettori accaniti. I castori devono quindi spesso aspettare in biblioteca quando vogliono restituire i libri.

Quando è il turno di un castoro, restituisce tutti i libri che ha portato con sé. Per restituire un libro ci vuole sempre un minuto esatto. Una volta che un castoro ha finito, è il turno del castoro successivo dalla zona di attesa. Si tratta sempre del castoro con il minor numero di libri.

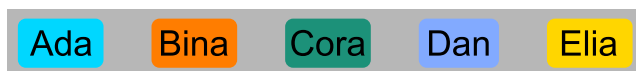
Una mattina, 5 castori arrivano uno dopo l'altro e vogliono restituire i loro libri. L'immagine mostra per ogni castoro

- quando arriva nella sala d'attesa e
- quanti libri ha portato con sé.



Ada è la prima ad arrivare e può restituire immediatamente i suoi 4 libri.

In quale ordine i castori restituiscono i libri?



← 1. 2. 3. 4. 5.

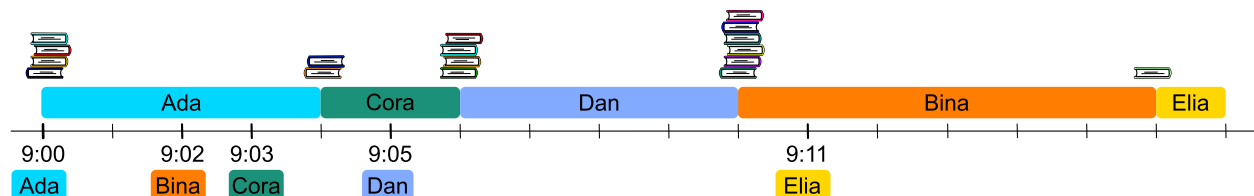


Soluzione

Questa è la soluzione:



L'illustrazione mostra come vengono restituiti i libri.



- Prima è il turno di Ada, alle 09:00. Impiega 4 minuti per restituire i suoi 4 libri. Ora sono le 09:04. Solo allora è il turno del castoro successivo.
- Nel frattempo, Bina e Cora sono arrivate e stanno aspettando. Cora ha 2 libri e Bina ne ha 6, quindi è il turno di Cora. Dopo 2 minuti, alle 09:06, ha restituito i libri.
- Nel frattempo, Dan è arrivato e sta aspettando con Bina. Dan ha 4 libri e Bina 6. Dan ha il minor numero di libri ed è il prossimo della fila. Dopo altri 4 minuti, alle 9:10, anche Dan ha finito.
- Ora è finalmente il turno di Bina. Alle 9.10 Bina è l'unica castora nella sala d'attesa, quindi ora ha il minor numero di libri.
- Nel frattempo è arrivato Elia. Quando Bina è pronta alle 9:16, è l'unico Castoro nella sala d'attesa. È poi l'ultimo della fila.

Questa è l'informatica!

Mentre un computer è in funzione, devono essere eseguite molte operazioni. Per ogni compito, viene avviato un processo che lo gestisce. È qui che entra in gioco lo *scheduler*. Si tratta di un programma che fa parte del sistema operativo di un computer. Lo scheduler controlla la sequenza dei processi. Determina quando e per quanto tempo un processo viene eseguito dall'unità centrale di elaborazione (CPU) del computer. Se la CPU è ben utilizzata, può accadere che i processi debbano aspettare. In questo caso, l'informatica riconosce diverse strategie in base alle quali gli scheduler selezionano il processo successivo da elaborare; tra le altre:

- *Priority Scheduling*: il processo con la priorità più alta viene eseguito successivamente (completamente).
- *First Come First Serve (FCFS)*: Il processo che è stato il primo nella coda è il primo a essere (completamente) eseguito. La maggior parte delle code funziona secondo questo principio nella vita quotidiana.
- *Round Robin*: i processi in attesa vengono eseguiti a turno solo per un certo tempo. Se un processo non è ancora terminato, viene interrotto e deve tornare alla fine della coda. È quindi il turno del processo successivo.



- *Shortest Job First (SJF)*: Il processo con il tempo di elaborazione previsto più breve viene eseguito per primo (completamente).

In questo compito, la biblioteca di Holzdorf utilizza la strategia *Shortest Job First* per la restituzione dei libri: Il castoro con il minor numero di libri (e quindi con il minor tempo di elaborazione) viene servito per primo.

Parole chiave e siti web

- Coda di priorità: https://it.wikipedia.org/wiki/Coda_di_priorità





8. Pettegolezzo

Quando il famoso castoro Alfred ha partecipato a una grande festa, è stato fotografato.



Quattro influencer hanno scritto di questo momento, ma non tutto quello che si legge sui social media è vero:

Solo una delle quattro affermazioni è vera. Quale?

A) Alfred aveva un cappello  e non usava un bastone .

B) Alfred indossava una cravatta  o un papillon .

C) Alfred aveva un cappello  e indossava un papillon .

D) Alfred indossava le infradito  o non indossava nemmeno le scarpe .



Soluzione

La risposta D è corretta: Alfred indossava le infradito  o non indossava nemmeno le scarpe



In questo compito si deve determinare il contenuto di verità (*vero* o *falso*) delle affermazioni su un'immagine. Ogni affermazione è composta da due sotto-affermazioni collegate da **e** o **o**.

Un'affermazione con **e** è *vera* solo se entrambe le affermazioni parziali sono *vere*.

In logica, un'affermazione con **o** è *vera* solo se almeno una delle sotto-affermazioni è *vera*.

In italiano, alcune delle seconde frasi sono abbreviate per non ripetere il soggetto o il predicato. Nell'analisi che segue, i sottodichiari sono elencati per intero:

- Affermazione A:

Alfred aveva un cappello  (*vero*) e Alfred non usava un bastone  (*falso*).

⇒ Questa affermazione è *falsa* perché non entrambe le affermazioni parziali sono *vere*.

- Affermazione B:

Alfred portava una cravatta  (*falso*) o Alfred portava un papillon  (*falso*).

⇒ Questa affermazione è *falsa* perché non almeno un'affermazione parziale è *vera*.

- Affermazione C:

Alfred indossava un cappello  (*vero*) e Alfred indossava un papillon  (*falso*).

⇒ Questa affermazione è *falsa* perché non entrambe le affermazioni parziali sono *vere*.

- Affermazione D:

Alfred indossava le infradito  (*falso*) o Alfred non indossava nemmeno le scarpe  (*vero*).

⇒ Questa affermazione è *vera* perché (almeno) un'affermazione parziale è *vera*.

Questa è l'informatica!

Non tutto quello che si legge sui cosiddetti «social media» è sempre vero. E purtroppo non sempre c'è una foto che possiamo usare per verificare la veridicità di un'affermazione.

La logica si occupa anche della verità degli enunciati: gli enunciati possono essere *veri* o *falsi*. Gli operatori logici possono essere utilizzati per collegare affermazioni (più semplici) a affermazioni più complesse. Il contenuto di verità dell'intero enunciato può quindi essere calcolato a partire dal contenuto di verità degli enunciati parziali. In questo compito del castoro informatico vengono utilizzati due operatori:



- **e**: L'intera affermazione è *vera* solo se entrambe le affermazioni parziali sono *vere*; altrimenti è *falsa*.
- **o**: L'intera affermazione è *vera* se almeno una delle affermazioni parziali è *vera*.

I *valori di verità* delle intere dichiarazioni possono essere visualizzati per entrambi gli operatori in una *tabella della verità*:

Affermazione parziale A	Affermazione parziale B	A e B	A o B
<i>vero</i>	<i>vero</i>	<i>vero</i>	<i>vero</i>
<i>falso</i>	<i>vero</i>	<i>falso</i>	<i>vero</i>
<i>vero</i>	<i>falso</i>	<i>falso</i>	<i>vero</i>
<i>falso</i>	<i>falso</i>	<i>falso</i>	<i>falso</i>

L'uso della logica è fondamentale nell'informatica, compresa la programmazione. Le espressioni logiche vengono utilizzate per determinare le condizioni in cui un programma deve fare qualcosa. In alcuni programmi, ciò diventa evidente anche durante l'uso, ad esempio quando si imposta un filtro antispam in un programma di posta elettronica. Se la condizione di un filtro antispam utilizza un **e** (ad esempio: l'e-mail proviene da un mittente sconosciuto **e** l'e-mail contiene la parola «Bitcoin» nell'oggetto), allora un'e-mail finisce nella cartella spam solo se entrambe le affermazioni parziali sono vere.

Parole chiave e siti web

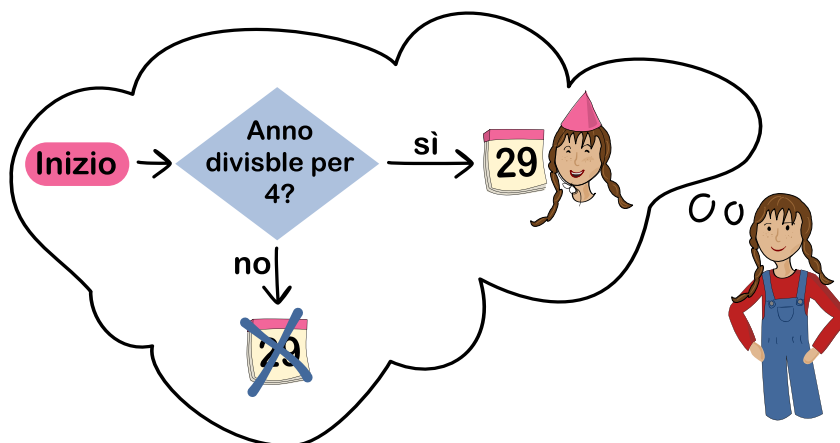
- Simboli logici: https://it.wikipedia.org/wiki/Simboli_logici
- Tabella della verità: https://it.wikipedia.org/wiki/Tabella_della_verità





9. Buon compleanno

Johanna è nata il 29 febbraio in un anno bisestile. Può festeggiare il suo compleanno il 29 febbraio solo se è un altro anno bisestile. In precedenza, ciò avveniva ogni quattro anni. Per poter determinare rapidamente se un anno è bisestile, Johanna ha creato questo «piano decisionale»:

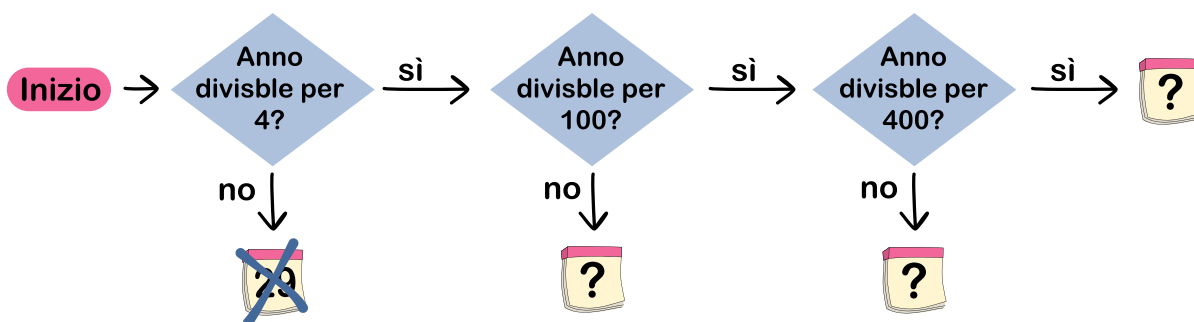


Dopo un po', Johanna scopre che è un po' più complicato determinare un anno bisestile:

- Se l'anno è divisibile per 100, non è un anno bisestile (ad esempio, 1900).
- Tuttavia, se l'anno è divisibile per 400, si tratta di un anno bisestile (ad esempio il 2000).

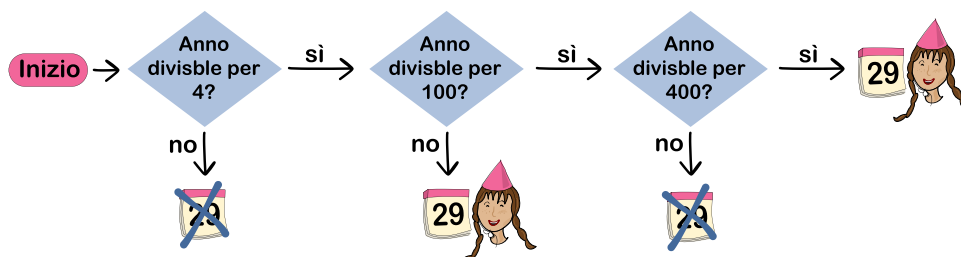
Johanna amplia il suo piano e aggiunge due domande, solo che deve ancora decidere se le risposte a queste due domande portano a un anno bisestile  oppure no .

Aiuta Johanna a scegliere la decisione giusta per ogni punto di domanda!





Soluzione



Questa è la soluzione:

Il piano decisionale di Johanna è una rappresentazione diversa delle regole per gli anni bisestili formulate nel compito. Le domande che Johanna ha aggiunto al suo piano rappresentano le nuove regole che Johanna ha imparato in seguito. È importante notare che la regola per gli anni divisibili per 400 annulla la regola per gli anni divisibili per 100. Per questo motivo Johanna ha collocato correttamente la domanda sulla «regola del 400» dopo quella sulla «regola del 100». Entrambe le domande vengono poste solo per gli anni divisibili per 4.

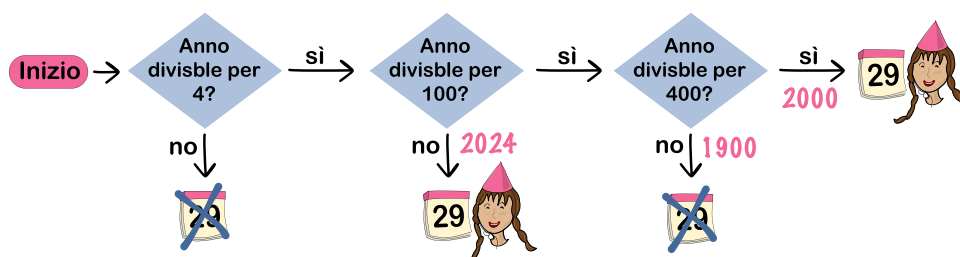
- La decisione in basso al centro è quindi presa per gli anni divisibili per 4, ma non per 100; tali anni non sono quindi nemmeno divisibili per 400. Le due nuove regole non si applicano quindi in questo caso, mentre continua a valere la regola degli anni divisibili per 4: sono anni bisestili



- La decisione in basso a destra viene presa per gli anni divisibili per 4, per 100 ma non per 400. Delle nuove regole, si applica solo la regola del 100: non sono anni bisestili
- La decisione all'estrema destra è presa per gli anni divisibili per 4, 100 e 400. In questo caso si applica la regola del 400: sono anni bisestili

Testiamo il piano finito con gli anni 1900, 2000 e 2024 già citati:

- Il 2000 è un anno bisestile. 2000 è divisibile per 4, 100 e 400. Nel piano decisionale, la decisione è quella all'estrema destra: anno bisestile. Funziona!
- Il 1900 non è un anno bisestile. Poiché il 1900 è divisibile per 4 e per 100 ma non per 400, il piano decisionale per questo anno porta alla decisione in basso a destra: non è anno bisestile. Si adatta anche a questo!
- Il 2024 è un anno bisestile. 2024 è divisibile per 4, ma non per 100 (e quindi non per 400). Il piano decisionale porta alla decisione in basso al centro per questo anno: anno bisestile. Corretto!





Questa è l'informatica!

Con il piano decisionale di questo compito, Johanna può determinare per ogni anno se l'anno è bisestile o meno. Il piano stesso ha un ambito finito (vedi sopra), le domande di decisione sono formulate chiaramente e possono essere risposte per ogni anno, Johanna può eseguire il piano per ogni anno in un numero finito di passi fino a raggiungere una decisione e deve solo ricordare il numero stesso. Il piano ha quindi tutte le proprietà di un algoritmo. Descrive un algoritmo che decide per un anno se l'anno è bisestile o meno.

Johanna ha progettato il suo piano o la descrizione del suo algoritmo di decisione sull'anno bisestile come un (semplice) diagramma di flusso del programma. I diagrammi di flusso dei programmi possono essere utilizzati per descrivere gli algoritmi senza dover ricorrere a un linguaggio di programmazione specifico. In informatica, è comune descrivere un algoritmo utilizzando un diagramma di flusso del programma o un altro formato neutro rispetto al linguaggio di programmazione, come i diagrammi di struttura o lo pseudocodice. Una descrizione di questo tipo è alla base del fatto che il processo descritto non dipende dai dettagli tecnici di uno specifico linguaggio di programmazione. L'algoritmo così descritto può poi essere tradotto dal formato neutro in un linguaggio di programmazione.

Parole chiave e siti web

- Diagramma di flusso: https://it.wikipedia.org/wiki/Diagramma_di_flusso





10. Tour forestale

La guardia forestale Flora offre visite agli alberi. Presenta alcuni degli alberi della foresta. Da tre tour precedenti sa quanto gli alberi siano stati apprezzati dai suoi ospiti.

Tour 1:  <  <  < 

Tour 2:  <  <  < 

Tour 3:  <  < 

Albero 1 < Albero 2 significa che l'albero 1 è stato meno popolare dell'albero 2.

Nei prossimi tour, Flora vuole introdurre gli alberi più popolari più tardi - il meglio arriva alla fine. Vuole sempre presentare gli alberi in un *buon ordine* che corrisponda alla popolarità degli alberi nei tre tour precedenti.

Un esempio: se Flora vuole introdurre gli alberi  e  in un buon ordine, allora deve introdurre  prima di , perché nel tour 1  era meno popolare di .

Nel suo prossimo tour, Flora vorrebbe presentare gli alberi , , ,  e .

Determina un buon ordine per questi alberi!





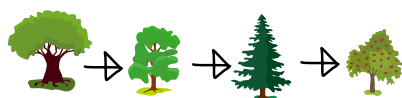
Soluzione

Questa è la soluzione:



Un buon ordine deve corrispondere alla popolarità degli alberi nei tre tour precedenti. Se riassumiamo gli «ordini di popolarità» dei tre tour in una rappresentazione, l'ordine di un nuovo tour può basarsi su questa rappresentazione.

Questo diagramma deve essere un diagramma con alberi e frecce. In esso, due alberi sono sempre collegati da una freccia quando un albero deve essere introdotto prima di un altro albero. Un diagramma di questo tipo può essere facilmente creato per ogni tour precedente. Ecco il diagramma per il tour 1:



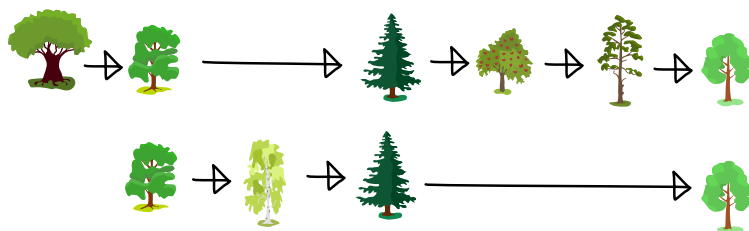
Possiamo facilmente combinare questo diagramma con quello del tour 3, perché i due tour hanno in

comune solo l'albero  :

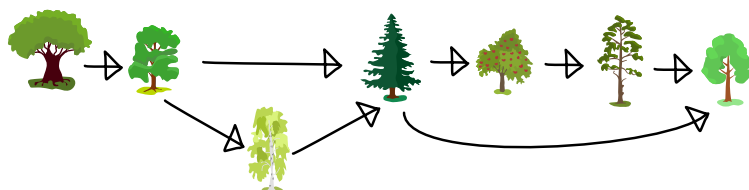


Ora dobbiamo combinare questo diagramma con quello del tour 2 (nell'immagine seguente). Tre dei

quattro alberi del tour 2 esistono già, solo  è nuovo:



Come sopra, ogni albero appare una sola volta nel diagramma combinato, ma nessuna freccia viene persa:





Utilizzando il diagramma, possiamo ora risolvere il compito molto facilmente: visitiamo gli alberi del nuovo tour nel diagramma da sinistra a destra, lungo le frecce, e otteniamo così un ordine chiaro e valido che corrisponde alla popolarità degli alberi in tutti i tour precedenti:



Questa è l'informatica!

In questo compito, un insieme di alberi deve essere ordinato in ordine crescente di popolarità. Esistono informazioni su alcuni alberi che sono stati confrontati a coppie nella loro popolarità. Per altre coppie

di alberi, come  e , non esiste un confronto diretto. Anche questi alberi possono essere confrontati, tuttavia, due proprietà sono di fondamentale importanza in questo caso:

1. il confronto di popolarità è *transitivo*. Ciò significa che se l'albero 1 < l'albero 2 e l'albero 2 < l'albero 3, allora l'albero 1 < l'albero 3.
2. i tre tour precedenti non sono contraddittori: se in uno qualsiasi dei tre tour l'albero 1 < l'albero 2, allora non esiste un altro tour con l'albero 2 < l'albero 1.

Grazie a queste proprietà, i confronti di popolarità formano un *ordine* in senso matematico. A prima vista può sembrare che questo ordine sia *parziale*, poiché non tutti gli alberi sono confrontati

a coppie, come  e . Tuttavia, gli alberi in questo compito del castoro possono essere disposti in una sequenza che corrisponde all'ordine di confronto della popolarità; chiamiamo tale sequenza un *ordinamento topologico*. L'informatica conosce un algoritmo relativamente semplice che può determinare un ordinamento topologico se non ci sono contraddizioni nei confronti a coppie noti.

A un esame più attento, si può notare che la popolarità degli alberi è un *ordine totale* grazie alla transitività. Ciò significa che l'ordinamento topologico di tutti gli alberi è unico, il che dimostra anche l'unicità della risposta corretta.

Parole chiave e siti web

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





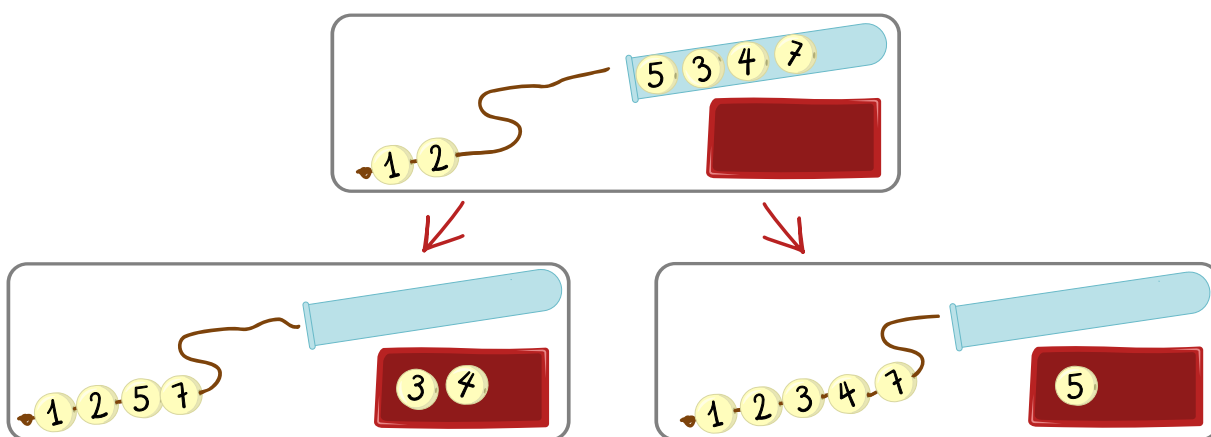
11. Braccialetto più lungo

Scott ha delle perline con dei numeri e ne fa dei braccialetti. Le perle escono da un tubo di vetro una dopo l'altra. Scott decide per ogni perla di utilizzarla per il braccialetto e infilarla nel cordoncino, oppure metterla via.

Scott può usare una perla solo se

- il cordoncino è vuoto, oppure
- il numero della perla è maggiore di quello dell'ultima perla della stringa.

In questo esempio, **2** è l'ultima perla della stringa:



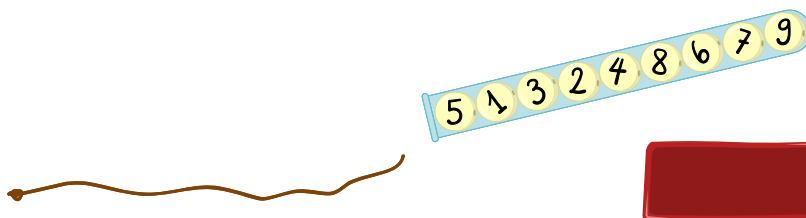
Scott può usare la prossima perla **5**, ma può anche metterla via. Quando

- usa **5**, può creare un braccialetto con 4 perline: **1 2 5 7**.
- mette via **5**, può creare un braccialetto con 5 perline: **1 2 3 4 7**.

[9]: graphics/2024-TW-03_9.svg "Perle 9 (25px inline(-0.5ex))"2

Scott riceve un nuovo tubo con perle. Vuole fare un braccialetto con il maggior numero possibile di perline.

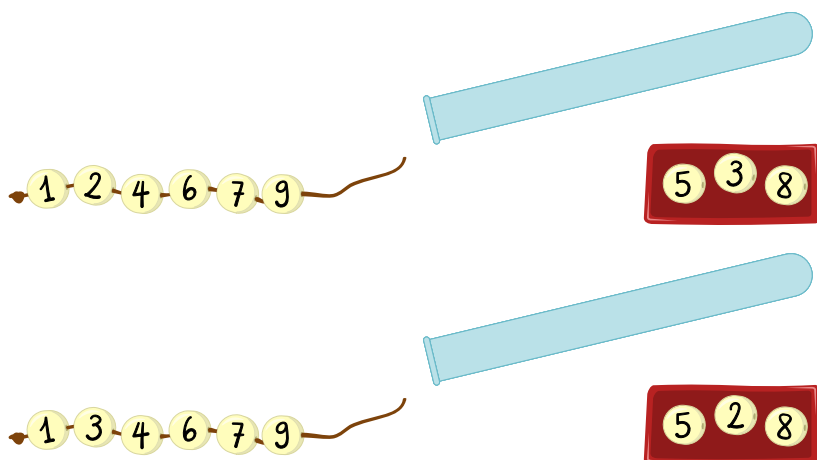
Come può riuscirci? Per ogni perla, decidi se usarla o metterla via.



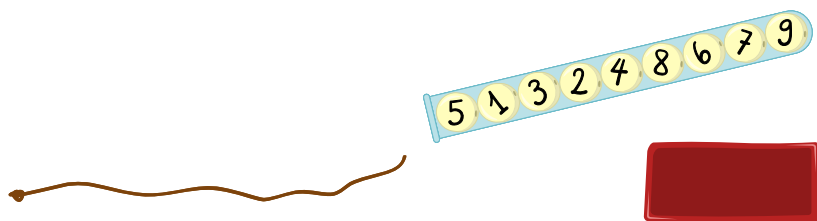


Soluzione

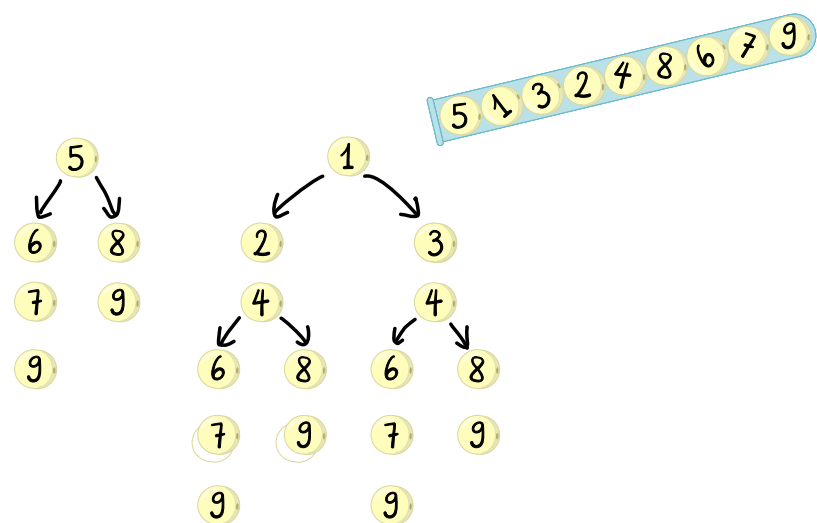
Questa è la soluzione:



Per questo tubo, potremmo elencare tutti i braccialetti che Scott può realizzare con le perline e vedere quali sono quelli con il maggior numero di perline. Tuttavia, questo richiede molto tempo. Osserviamo più da vicino i numeri sulle perline, perché i braccialetti di Scott in fin dei conti sono sequenze di numeri ordinati in ordine crescente.



Se Scott usa il primo numero (5) dal tubo, può usare solo il 6, il 7, l'8 e il 9 e creare i braccialetti 5679 e 589. Se Scott mette via il 5 e usa il numero successivo 1 come primo numero per il braccialetto, può fare più braccialetti con più perline. Questo albero di decisione mostra come Scott può utilizzare le perline e quali braccialetti può realizzare:





Se Scott scarta anche l'1, può realizzare solo braccialetti con un numero di perline inferiore a quello possibile. Se Scott usa il 2 per primo, ad esempio, può ancora creare i braccialetti 24679 e 2489. Tuttavia, questi sono contenuti nei braccialetti 124679 e 12489, che Scott può realizzare se utilizza per primo l'1.

I braccialetti con il maggior numero possibile di perline iniziano quindi con 1 e sono composti da 6 perline: 124679 o 134679.

Questa è l'informatica!

Ogni tubo di questo compito contiene una *sequenza* di perline numerate, che possiamo considerare come una sequenza di numeri. Per il modo in cui Scott usa le perline, ogni braccialetto è una *sequenza ascendente* della «sequenza di tubi»; *ascendente* significa che ogni numero della sequenza (tranne il primo) è maggiore del numero che lo precede. Per realizzare un braccialetto con il maggior numero possibile di perline, Scott deve determinare la *sequenza ascendente più lunga*.

Nel caso di lunghe sequenze di numeri, sarebbe necessario un tempo molto lungo per formare tutte le possibili sottosequenze ascendenti al fine di determinare la più lunga. Se la provetta contiene 20 perline, ad esempio, lo sforzo computazionale è già dell'ordine di un milione di passi.

Fortunatamente, gli informatici hanno sviluppato algoritmi in grado di determinare più rapidamente la sottosequenza ascendente più lunga. Si tratta di una tecnica chiamata **programmazione dinamica**. L'ordine di grandezza dello sforzo computazionale per questi algoritmi veloci per una provetta con 20 perline è inferiore a cento passi.

Parole chiave e siti web

- Albero di decisione: https://it.wikipedia.org/wiki/Albero_di_decisione
- Programmazione dinamica: https://it.wikipedia.org/wiki/Programmazione_dinamica





12. Catene di parole

La classe del signor castoro sta imparando a leggere. I bambini lavorano con catene di parole: La parola successiva in una catena differisce dalla precedente esattamente in una posizione.

Ecco un esempio: VOI – NOI – NEI – SEI



Il signor castoro scrive ora alla lavagna queste nove parole: VOI, NOI, NEI, SEI, REO, NEO, NON, NEL, REI.

La classe deve usare tutte le parole per formare 3 catene di parole di 3 parole ciascuna. Il signor castoro avverte: «A dipendenza della prima catena di parole che deciderete di formare, potreste non essere più in grado di formare le altre due catene di parole usando tutte le parole disponibili».

*Quale catena di parole la classe **non deve formare per prima**?*

- A) REO – NEO – NEL
- B) NEI – NEL – NEO
- C) NON – NOI – NEI
- D) SEI – NEI – NEL



Soluzione

La risposta C è corretta: NON – NOI – NEI

Possiamo risolvere questo compito con l'aiuto di un disegno: Per prima cosa, scriviamo tutte le parole. Poi colleghiamo con una linea due parole che differiscono esattamente in una posizione.

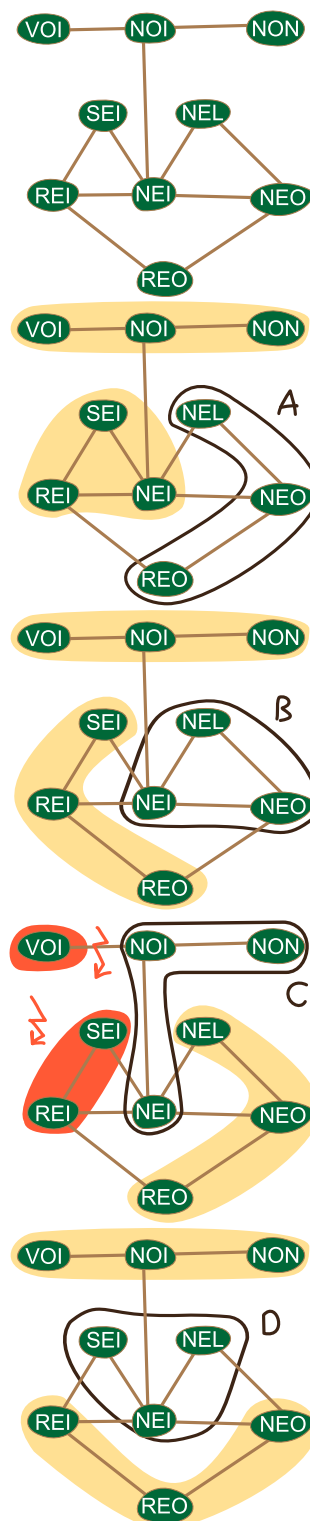
Nella risposta A si forma la catena di parole NEL – NEO – REO. Le parole rimanenti possono essere utilizzate per formare le catene di parole contrassegnate in giallo, ad esempio. Tutte le parole vengono usate esattamente una volta. Tutte le catene di parole sono composte da tre parole.

La catena di parole nella risposta B è NEL – NEO – NEI. Possiamo usare le parole rimanenti per formare le catene di parole contrassegnate in giallo. Tutte le parole sono usate esattamente una volta. Tutte le catene di parole sono composte da tre parole.

Nella risposta C si forma dapprima la catena di parole NEI – NOI – NON. Si nota che VOI è isolato. Ad esempio, possiamo formare la catena di parole contrassegnata in giallo con le parole rimanenti, ma le due parole contrassegnate in rosso e la parola VOI non possono più venire utilizzate per comporre una catena di parole valida.

La catena di parole nella risposta D è SEI – NEI – NEL. Dalle parole rimanenti possiamo formare le catene di parole contrassegnate in giallo. Tutte le parole sono usate esattamente una volta. Tutte le catene di parole sono composte da tre parole.

La classe non deve formare prima la catena di parole della risposta C.





Questa è l'informatica!

Nei disegni che abbiamo utilizzato per spiegare la soluzione, ogni parola è un *nodo* (evidenziato in verde). Un *spigolo* tra due nodi (disegnato come una linea) indica che queste due parole differiscono esattamente in una posizione. Questi due nodi sono quindi *vicini*. Nella catena di parole VOI – NOI – NEI – SEI, per esempio, VOI e NOI sono vicini, ma NOI e SEI no. In informatica, le costruzioni di nodi e spigoli sono generalmente note come *grafi* e vengono utilizzate per rappresentare le relazioni tra gli oggetti rappresentati dai nodi e gli spigoli.

La catena di parole VOI – NOI – NEI comprende tre nodi e due spigoli. Esiste un *percorso* tra VOI e NEI che collega i nodi corrispondenti attraverso il vicino comune NOI. Da ognuna delle nove parole possiamo raggiungere tutte le altre parole. Questo può essere diretto (cioè se i nodi corrispondenti sono vicini) o con l'aiuto di un percorso più lungo che si estende su diversi vicini. Per risolvere questo compito del castoro, sono necessari tre percorsi di questo tipo. Ogni nodo deve essere utilizzato esattamente una volta, per cui se si considerano solo i percorsi e nessun altro bordo, alla fine il grafo è diviso in tre *componenti*.

I grafi sono uno strumento molto utilizzato dagli informatici per modellare problemi complessi. I programmi possono risolvere molti compiti con l'aiuto dei grafi. I sistemi di navigazione, ad esempio, sono programmi che utilizzano grafi molto grandi per guidare le auto passo dopo passo dal punto di partenza alla destinazione. A ogni passo, l'auto percorre un bordo da un nodo a un vicino.

Parole chiave e siti web

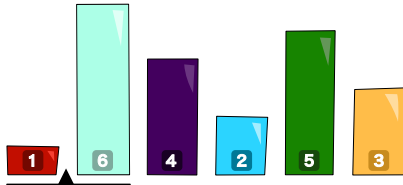
- Grafo: <https://it.wikipedia.org/wiki/Grafo>





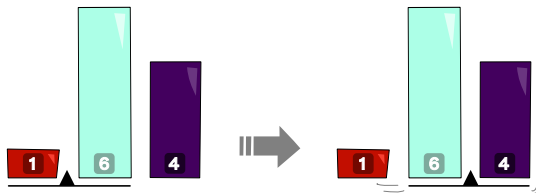
13. Macchina dei mattoni

Una macchina può spostare sei mattoni di altezza diversa. Per farlo, utilizza un marcatore (▲), che di solito è posizionato tra due mattoni. All'inizio, i mattoni e il marcatore sono posizionati in questo modo:

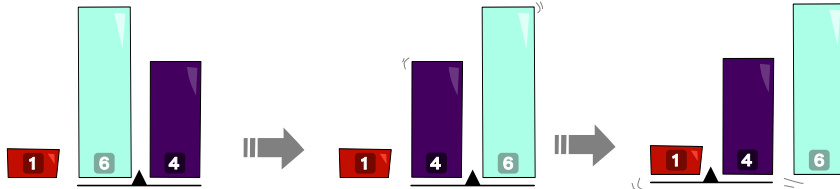


La macchina funziona quindi in conformità a queste norme:

- Se il mattone a sinistra del marcatore è **più piccolo** di quello a destra, il marcatore si sposta a destra.

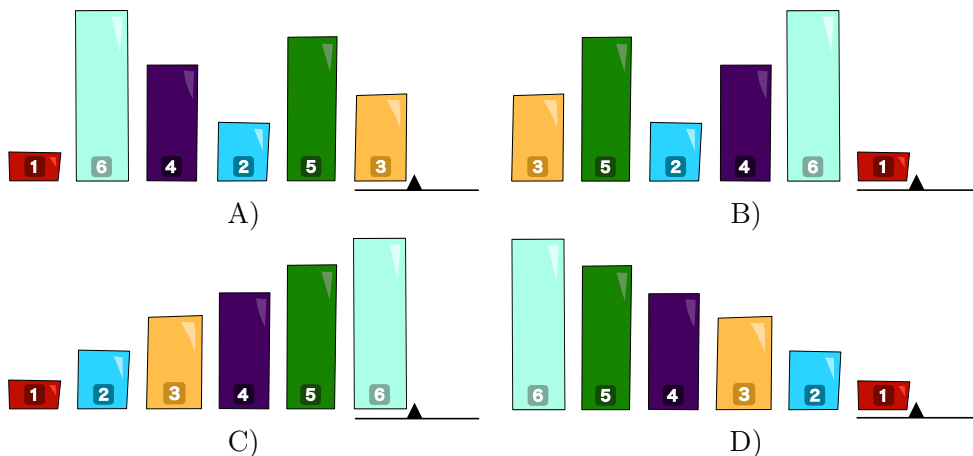


- Se il mattone a sinistra del marcatore è **più grande** di quello a destra, i mattoni vengono scambiati. Il marcatore si sposta quindi a sinistra, ma solo se si trova ancora tra due mattoni.



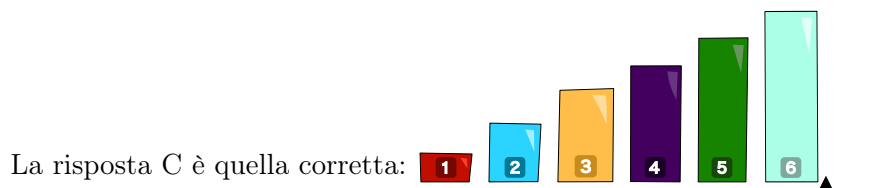
La macchina funziona fino a quando la marcatura si trova all'estrema destra accanto ai mattoni. A quel punto la macchina si ferma.

Come sono disposti i mattoni dopo l'arresto della macchina?





Soluzione



La risposta C è quella corretta:

Si può seguire il lavoro della macchina passo per passo per questo caso particolare, tuttavia è più facile riconoscere che la macchina esegue l'ordinamento dei blocchi in base alle dimensioni.

Il marcatore si sposta a destra solo se il mattone a sinistra del marcatore è più piccolo di quello a destra. Se la situazione è diversa, cioè il mattone di sinistra è più grande di quello di destra, il mattone più piccolo si sposta a sinistra e quello più grande a destra. Il mattone più piccolo, che ora si trova a sinistra del marcatore, potrebbe essere più piccolo di qualche mattone più a sinistra. Per questo motivo anche il marcatore si sposta a sinistra. Se tutti i mattoni a sinistra del marcatore sono ordinati per dimensione prima di un tale passo verso sinistra, questo è sarà ancora il caso se si esegue un ulteriore passo a sinistra.

Se il mattone più piccolo prima di un passo a sinistra è anche più piccolo dei mattoni più a sinistra, seguono altri passi a sinistra. Di conseguenza, il mattone più piccolo viene scambiato più a sinistra finché non è più grande del mattone alla sua sinistra (e quindi più grande di tutti gli altri alla sua sinistra) - o finché non si trova all'estrema sinistra se è il mattone più piccolo. Seguono passi verso destra finché non appare un altro mattone più piccolo, e poi di nuovo passi verso sinistra.

Questa operazione continua fino a quando non ci sono più mattoni «piccoli» a destra. A questo punto il marcatore fa dei passi a destra finché non si trova alla destra dei mattoni e la macchina si ferma. A questo punto tutti i mattoni si trovano alla sinistra del marcatore e vengono ordinati in ordine crescente di grandezza, come nella risposta C.

Questa è l'informatica!

I computer elaborano spesso molti dati. È importante che questi dati siano organizzati. In informatica, di solito si usa il termine «ordinare» invece che «organizzare»: ad esempio, i numeri possono essere ordinati per dimensione crescente, i dati sulle persone per data di nascita o i dati sui libri per numero ISBN. Al termine di un processo di ordinamento, i dati si trovano nell'ordine desiderato. Poiché i computer devono costantemente ordinare grandi quantità di dati, gli informatici si preoccupano, tra l'altro, di trovare i metodi di ordinamento più efficienti possibili, che richiedano poco tempo e poco spazio di archiviazione per il loro lavoro.

Il metodo di ordinamento utilizzato dalla macchina per l'ordinamento dei blocchi in questo compito si chiama *Gnome sort*. È simile al ben più noto *Bubblesort* (noto anche come «ordinamento a bolla»). Gnomesort, come Bubblesort, non è particolarmente efficiente se i dati sono ordinati in modo casuale. Tuttavia, se i dati sono già approssimativamente ordinati, entrambi i metodi sono molto veloci, forse più veloci di metodi di ordinamento come *Quicksort* o *Mergesort*, che sono generalmente molto più



efficienti. Per questo motivo, e perché Gnomesort è facile da capire e da programmare, il metodo è sicuramente interessante per alcune applicazioni.

Parole chiave e siti web

- Algoritmo di ordinamento https://it.wikipedia.org/wiki/Algoritmo_di_ordinamento
- Gnome sort https://it.wikipedia.org/wiki/Gnome_sort
- Demo di Gnomesort: <https://www.algostructure.com/sorting/gnomesort.php>
- Bubble sort (ordinamento a bolla) https://it.wikipedia.org/wiki/Bubble_sort
- Quicksort <https://it.wikipedia.org/wiki/Quicksort>
- Merge sort https://it.wikipedia.org/wiki/Merge_sort





14. Macchina dei palloncini

Una macchina dei palloncini può creare immagini gonfiando palloncini in una cornice quadrata.

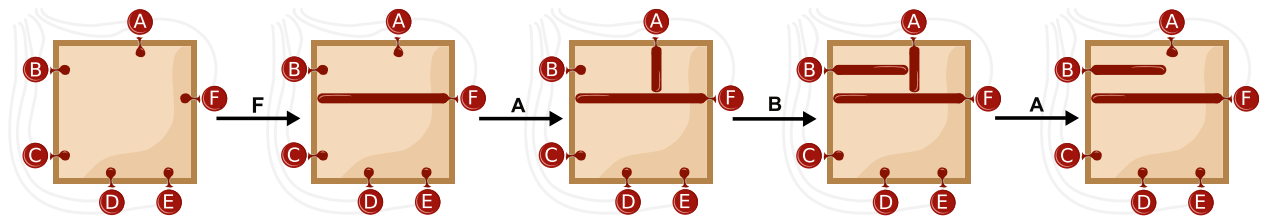
I palloncini sono etichettati con le lettere A, B, C, D, E e F.

La macchina legge una sequenza di lettere una dopo l'altra, da sinistra a destra. Prima della prima lettera, tutti i palloncini sono vuoti.

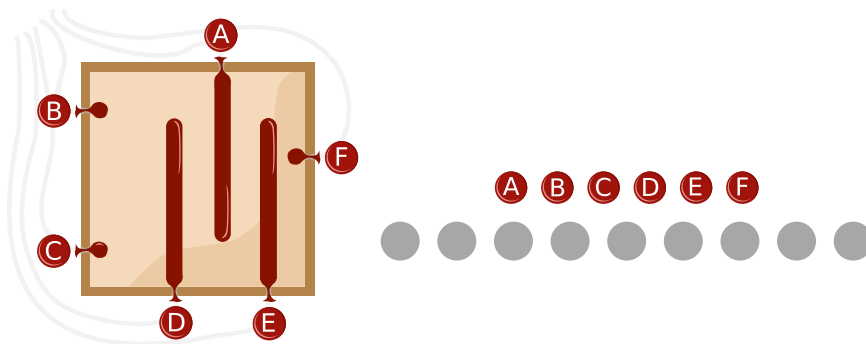
Quando la macchina ha letto una lettera, esegue le seguenti operazioni:

- Quando il palloncino con questa lettera è vuoto, viene gonfiato fino a toccare un altro palloncino o il bordo opposto della cornice.
- Quando il palloncino con questa lettera è già gonfio, viene sgonfiato.

Ad esempio, quando la macchina legge la sequenza di lettere F A B A, fa così:



La macchina legge una sequenza di 9 lettere. Alla fine ha creato questa immagine. Completa la sequenza:





Soluzione

Questa è la soluzione:

È possibile stabilire una serie di condizioni per la sequenza di lettere che la macchina legge (d'ora in poi: sequenza di lettura):

1. affinché il palloncino di E (che d'ora in poi chiameremo palloncino E) raggiunga la lunghezza indicata nel risultato, è necessario gonfiare prima B e poi E. In un secondo momento, B deve essere nuovamente sgonfiato. La sequenza di lettura deve quindi contenere B E B come una parte. (Tale parte può essere interrotta da altre lettere).
2. L'affermazione precedente per il pallone E vale anche per il pallone D. La sequenza di lettura deve quindi contenere anche B D B come una parte.
3. Affinché il palloncino A raggiunga la lunghezza indicata nel risultato, è necessario gonfiare prima C e poi A. In un secondo momento, C deve essere nuovamente sgonfiato. C A C è quindi un'altra parte della sequenza di lettura.
4. B deve essere sgonfiato prima che A venga gonfiato.
5. C deve essere sgonfiato prima che D venga gonfiato.
6. quando B viene gonfiato per il palloncino E, A deve essere sgonfiato, come all'inizio.
7. Affinché A, D ed E siano gonfiate alla fine, queste lettere possono comparire solo una volta (o tre volte, o cinque volte, ... - ma la sequenza può avere solo un totale di nove lettere). A causa delle condizioni da 4 a 6, E A D deve far parte della sequenza di lettura.

Ecco una sequenza di lettura che soddisfa tutte le condizioni: B E B C A C B D B. Mostriamo le situazioni dopo B E, dopo B E B C A e prima dell'ultima B:

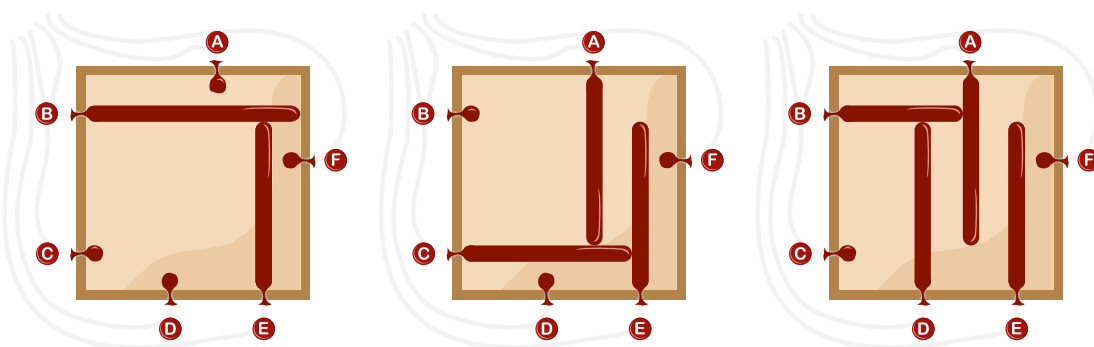
Sequenza di lettura:

B E

B E B C A

B E B C A C B D

Situazione:



La sequenza di lettura non è completamente determinata dalle condizioni. Nella sequenza qui sopra, l'ordine non è specificato dalle condizioni esattamente in due punti: B E **B C A C B** D B. Ci sono quindi in totale quattro sequenze di lettura corrette:

- B E B C A C B D B
- B E C B A C B D B
- B E B C A B C D B
- B E C B A B C D B



Questa è l'informatica!

La sequenza di lettura è come un programma che controlla la macchina dei palloncini. Ogni lettera è un'istruzione che fa sì che la macchina gonfi o sgonfi un palloncino. Le «istruzioni delle lettere» possono essere combinate per formare una sequenza di istruzioni. In questo modo si crea un programma più lungo con il quale la macchina può creare immagini più complesse. La sequenza di istruzioni è importante. Ad esempio, la sequenza di istruzioni B E crea un'immagine diversa dalla sequenza di istruzioni E B. Questo perché l'esecuzione di un'istruzione può avere un effetto sull'esecuzione delle istruzioni successive. Nella macchina dei palloncini, l'esecuzione di un'istruzione ha un effetto particolare sulla lunghezza dei palloncini che vengono gonfiati successivamente.

I programmi informatici sono costituiti anche da istruzioni che vengono combinate per formare programmi più ampi. In informatica, la forma più semplice di combinazione è chiamata *sequenza*, cioè una sequenza di istruzioni. Combinazioni più complesse sono create da speciali *strutture di controllo*, come le istruzioni di ripetizione e le istruzioni condizionali. Ogni «istruzione lettera» della macchina a palloncino in questo compito è un'istruzione condizionale. Nello stile di un linguaggio di programmazione per computer, questa istruzione può essere formulata in questo modo - assumendo che un palloncino possa avere solo due stati (vuoto e non vuoto):

Se il pallone di questa lettera è vuoto, allora:

Gonfiare il palloncino fino a ...

Altrimenti:

Sgonfiare il palloncino.

È un compito importante per gli informatici considerare attentamente gli effetti delle istruzioni durante lo sviluppo dei programmi, in modo che questi funzionino come desiderato. Questo è particolarmente difficile con le istruzioni condizionali, poiché la loro esecuzione dipende dalla valutazione della condizione specificata nell'istruzione.

Parole chiave e siti web

- Programma informatico: [https://it.wikipedia.org/wiki/Programma_\(informatica\)](https://it.wikipedia.org/wiki/Programma_(informatica))
- Struttura di controllo: https://it.wikipedia.org/wiki/Struttura_di_controllo



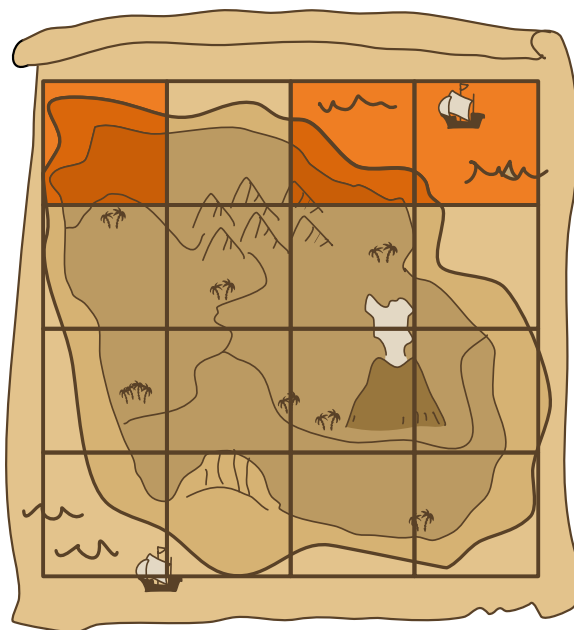


15. Pip il pirata

Il pirata Pip è alla ricerca di un tesoro su un'isola deserta. Pip ha una mappa dell'isola. La mappa è divisa in 16 quadrati.

Plapper, il pappagallo di Pip, sa quale casella contiene il tesoro. Pip può chiedere a Plapper un numero qualsiasi di quadrati. Plapper gli dice quindi se il tesoro si trova o meno in una di queste caselle.

Un esempio: Pip chiede i tre quadrati segnati nell'immagine. Se Plapper dice «sì», Pip sa che il tesoro si trova in una di queste tre caselle, ma non in quale.



Pip vuole scoprire il più rapidamente possibile quale casella contiene il tesoro.

Qual è il numero minimo di volte in cui Pip deve chiedere a Plapper?



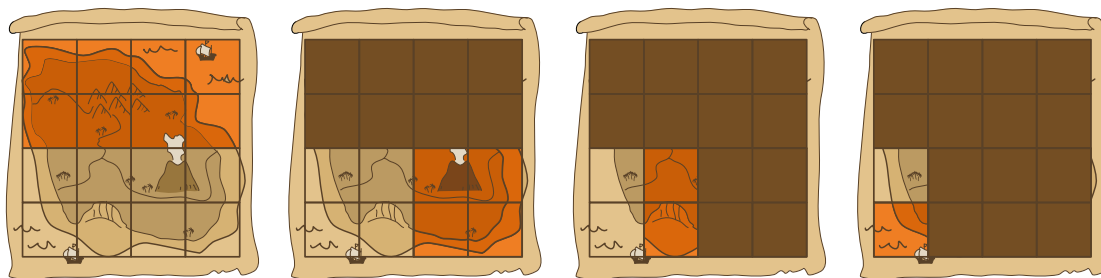
Soluzione

4 è la risposta corretta.

Pip può usare questa strategia per sapere con certezza in quale casella si trova il tesoro dopo quattro domande: Chiede esattamente la metà delle caselle in cui il tesoro può ancora trovarsi dopo le domande precedenti.

All'inizio, il tesoro può ancora trovarsi in tutte le 16 caselle. Quindi, nella domanda 1, Pip chiede 8 caselle. Se Plapper dice «sì», il tesoro può essere in queste 8 caselle; se Plapper dice «no», il tesoro può ancora essere nelle altre 8 caselle. Nella domanda 2, Pip chiede 4 caselle delle 8 rimanenti, nella domanda 3 2 caselle delle 4 rimanenti e nella domanda 4 una casella delle 2 rimanenti. A seconda della risposta alla domanda 4, Pip ora sa esattamente in quale casella si trova il tesoro.

Le immagini mostrano un possibile percorso. Pip divide i quadrati rimanenti in metà uguali. I quadrati richiesti sono segnati in rosso chiaro. Plapper dice sempre «no» e dopo quattro domande Pip sa che il tesoro si trova nella casella non segnata nell'ultima immagine.



Pip non può cavarsela con meno domande. Se non dividesse i quadrati rimanenti in due metà uguali, il tesoro potrebbe trovarsi nella parte più grande. Per trovare questa parte, Pip dovrebbe fare almeno tante domande quante ne farebbe per una metà.

Questa è l'informatica!

Il metodo utilizzato da Pip nella caccia al tesoro è chiamato *ricerca binaria* in informatica. Il termine *binario* deriva dalla parola latina bis (due volte). In una ricerca binaria di un oggetto in un insieme, l'insieme viene ripetutamente dimezzato da una serie di domande, cioè diviso in due parti - da qui «binario». Un insieme può essere dimezzato bene se gli oggetti in esso contenuti possono essere ordinati, ad esempio per dimensione; questo vale, tra l'altro, per ogni insieme di numeri. Allora esiste un oggetto intermedio nell'insieme e si può confrontare l'oggetto intermedio con l'oggetto che si sta cercando. Se l'oggetto centrale non è già l'oggetto che si sta cercando, si sa comunque quale metà dell'insieme contiene l'oggetto che si sta cercando e si cerca di nuovo in questa metà in modo binario.

In questo modo, si arriva all'oggetto cercato molto rapidamente. Per 1.000 oggetti sono necessari circa 10 passi di ricerca, per 1.000.000 di oggetti circa 20. In generale, si può dire: per n oggetti sono necessari circa $\log(n)$ passi; la funzione $\log$ è il «logaritmo di due» o il logaritmo in base 2. Poiché la ricerca binaria è così veloce, viene spesso utilizzata nei programmi informatici per la ricerca in dati ordinati.



In questo compito, lo spazio di ricerca è l'insieme dei quadrati della mappa dell'isola. È possibile organizzare i quadrati numerandoli dall'alto in basso e da sinistra a destra. Ma in questo caso, funzionerebbe anche dimezzare casualmente il numero di caselle in cui è possibile trovare il tesoro. In questo caso è solo un po' più dispendioso ricordare quale insieme di caselle è ancora possibile per la fase di ricerca successiva e deve essere nuovamente dimezzato.

Parole chiave e siti web

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





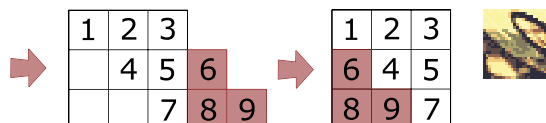
16. Crittografia delle immagini

	Esempio 3×3 Pixel	Immagine 25×25 Pixel									
Un'immagine è un rettangolo composto da righe e colonne di pixel (punti di colore)	<table border="1"> <tr><td>1</td><td>2</td><td>3</td></tr> <tr><td>4</td><td>5</td><td>6</td></tr> <tr><td>7</td><td>8</td><td>9</td></tr> </table>	1	2	3	4	5	6	7	8	9	
1	2	3									
4	5	6									
7	8	9									

Leo ha ideato un metodo per crittografare le immagini. Utilizza queste due operazioni:

Operazione O (per orizzontale)

Nella riga 1, i pixel rimangono invariati. Nella riga 2, ogni pixel si sposta di 1 a destra. Nella riga 3, ogni pixel si sposta di 2 colonne a destra.

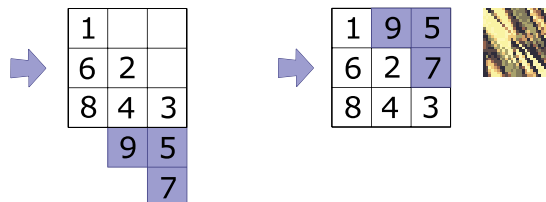


Ogni pixel della riga n si sposta a destra di $n-1$ colonne.

I pixel che si spostano oltre il bordo destro dell'immagine vengono reinseriti a sinistra nella stessa riga. L'ordine dei pixel non viene modificato nel processo.

Operazione V (per la verticale)

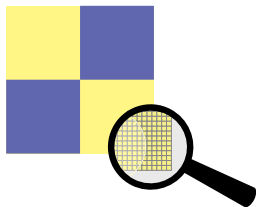
Ogni pixel della colonna n si sposta verso il basso di $n-1$ righe.



I pixel che si spostano oltre il bordo inferiore dell'immagine vengono reinseriti in cima alla stessa colonna.

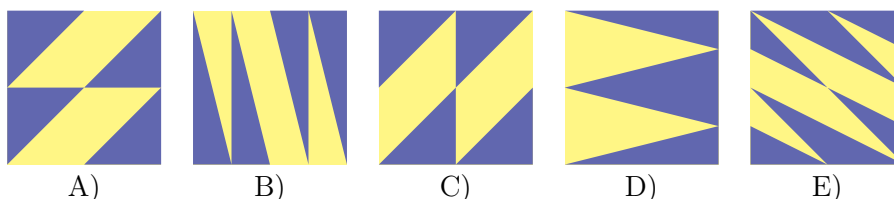


Le operazioni possono essere eseguite una dopo l'altra, in sequenza. Nell'esempio, Leo ha crittografato l'immagine (25×25 pixel) con la sequenza **OV**.



Leo cripta la seguente immagine (1000×1000 pixel) con la sequenza **VO**.

Qual è il risultato?





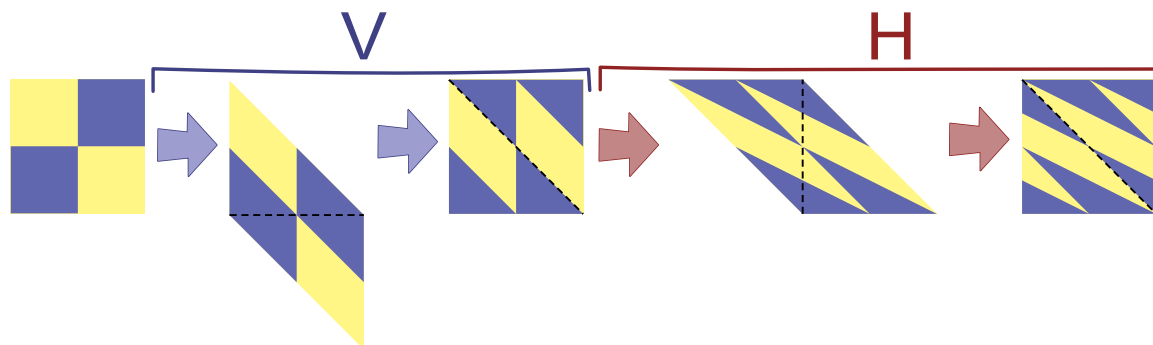
Soluzione



La risposta E è corretta:

Nell'esempio, la distorsione dell'immagine della Gioconda indica che l'operazione O provoca geometricamente un *taglio* orizzontale e l'operazione V un *taglio* verticale.

L'immagine con i quattro quadrati colorati viene prima tagliata verso il basso con l'operazione V. Il risultato è un parallelogramma con un'immagine distorta. Il risultato è un parallelogramma con un'immagine distorta. Il triangolo sporgente sul bordo inferiore viene inserito in alto (con l'operazione *spostamento parallelo*) in modo da creare nuovamente un'immagine rettangolare. Allo stesso modo, si esegue un taglio verso destra utilizzando l'operazione O. La sporgenza viene inserita a sinistra:



Il risultato della crittografia con la sequenza di operazioni **VO** corrisponde all'immagine della risposta E.

Questa è l'informatica!

Quando si trasferiscono dei dati, può essere importante crittografarli. Il mittente e il destinatario dei dati concordano il metodo di crittografia (e decrittografia) da utilizzare. Se i dati sono brevemente etichettati come D e il metodo di crittografia C (per «crittografia»), si può dire che il risultato dell'applicazione viene trasferito da C a D , cioè $C(D)$. Il destinatario deve quindi applicare la parte di decrittazione della procedura (C') e quindi ha $C'(C(D)) = D$. Ciò significa che chi osserva la trasmissione di $C(D)$ non può determinare facilmente i dati originali D .

Il metodo di crittografia delle immagini di questo compito, con le operazioni **O** e **V**, ha il vantaggio di essere facile da usare. Se il destinatario conosce la chiave utilizzata per la crittografia, in questo caso la sequenza di operazioni (ad esempio **OV**), può facilmente decifrare i dati crittografati. Tuttavia, il risultato della crittografia non può più essere riconosciuto a occhio. Se i dati vengono trasmessi in formato digitale, invece, si possono usare i computer e l'originale può essere decifrato con relativa facilità (cioè determinato senza conoscere la chiave).

Quanto più elevati sono i requisiti di sicurezza di un metodo di crittografia, tanto maggiore è lo sforzo richiesto per la crittografia e la decrittografia (ad esempio, lo sforzo computazionale e la trasmissione



di una complessa chiave segreta tra il mittente e il destinatario delle informazioni crittografate). Tuttavia, l'informatica conosce metodi moderni che richiedono uno sforzo minimo per la crittografia e la decrittografia, ma le cui proprietà speciali (per lo più matematiche) rendono praticamente impossibile decifrare i dati trasmessi.

Parole chiave e siti web

- Cifrario: <https://it.wikipedia.org/wiki/Cifrario>





17. Giorni di sole



Tom dice: «Nei giorni di sole, almeno un castoro nuota in ogni stagno».

Kim risponde: «Non è vero. La tua affermazione è stata smentita domenica scorsa».

Cosa è successo domenica scorsa? Completa gli spazi vuoti in modo da confutare l'affermazione di Tom:

Domenica scorsa il sole

e

.

brillava

i castori nuotavano in tutti gli stagni

non brillava

nessun castoro nuotava nello stagno delle cascate

il castoro Michael ha nuotato in tutti gli stagni

il castoro Michael non nuotava affatto



Soluzione

Questa è la soluzione:

Domenica scorsa il sole	brillava	e	nessun castoro nuotava nello stagno delle cascate	.
			i castori nuotavano in tutti gli stagni	
	non brillava			
			il castoro Michael ha nuotato in tutti gli stagni	
			il castoro Michael non nuotava affatto	

Tom fa un'affermazione su tutte le giornate di sole. Per confutare l'affermazione di Tom, bisogna trovare una giornata di sole per la quale la seconda parte dell'affermazione non sia vera. Poiché Tom non fa un'affermazione sui giorni non soleggiati, non si può confutarlo con un'affermazione sui giorni non soleggiati. La prima lacuna deve quindi essere riempita con «brillava».

La seconda parte dell'affermazione di Tom è a sua volta un'affermazione su tutti gli stagni. Per confutare questa affermazione, abbiamo bisogno di un'affermazione su (almeno) uno stagno in cui la seconda parte dell'affermazione - cioè che un castoro nuota nello stagno - non sia vera. Tale affermazione è «nessun castoro nuotava nello stagno delle cascate»; la seconda lacuna deve quindi essere riempita con questa affermazione.

L'affermazione di Tom non può essere confutata con le altre opzioni per colmare la seconda lacuna:

«Domenica scorsa il sole brillava e i castori nuotavano in tutti gli stagni»: Questa affermazione concorda con quella di Tom.

«Domenica scorsa il sole brillava il castoro Michael ha nuotato in tutti gli stagni»: Quindi domenica scorsa, a un certo punto, almeno un castoro ha nuotato in ogni stagno (cioè Castoro Michael), proprio come sostiene Tom per le giornate di sole. Anche questa affermazione concorda con quella di Tom.

«Domenica scorsa il sole brillava e il castoro Michael non ha nuotato affatto»: Questa affermazione non è sufficiente a confutare l'affermazione di Tom. Anche se il castoro Michael non ha nuotato, altri castori avrebbero potuto nuotare in qualsiasi stagno.

Questa è l'informatica!

In questo compito, Kim discute la veridicità dell'affermazione di Tom. Tom fa un'affermazione su tutti i giorni di sole, e per questi a sua volta un'affermazione su tutti gli stagni. L'intera affermazione può essere falsa solo se esiste almeno un giorno di sole in cui nessun castoro ha nuotato in almeno uno stagno. Pertanto, un singolo controesempio è sufficiente a confutare la sua affermazione.

La logica classica si occupa anche della verità degli enunciati: gli enunciati possono essere *veri* o *falsi*. Gli operatori logici come «e», «o» e «non» sono utilizzati per combinare affermazioni (più



semplici) in affermazioni più complesse. I *quantificatori* come «tutti» e «almeno uno» sono usati per fare affermazioni su insiemi di oggetti.

L'uso della logica è fondamentale in informatica. I sistemi informatici sono costruiti con blocchi logici come «e», «o» e «non», che utilizzano i valori 1 e 0 (tensione elettrica o assenza di tensione) invece di «vero» e «falso». Anche i linguaggi di programmazione utilizzano espressioni logiche, che vengono impiegate in affermazioni come SE è estate E il sole splende, ALLORA accendi l'aria condizionata.

Parole chiave e siti web

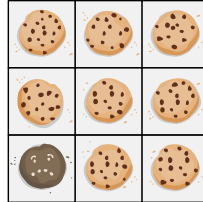
- Proposizione (logica): [https://it.wikipedia.org/wiki/Proposizione_\(logica\)](https://it.wikipedia.org/wiki/Proposizione_(logica))
- Simboli logici: https://it.wikipedia.org/wiki/Simboli_logici





18. Frowny

Ci sono 9 biscotti in una scatola. Sono tutti deliziosi, tranne il biscotto cattivo **Frowny** 🍪 in basso a sinistra.

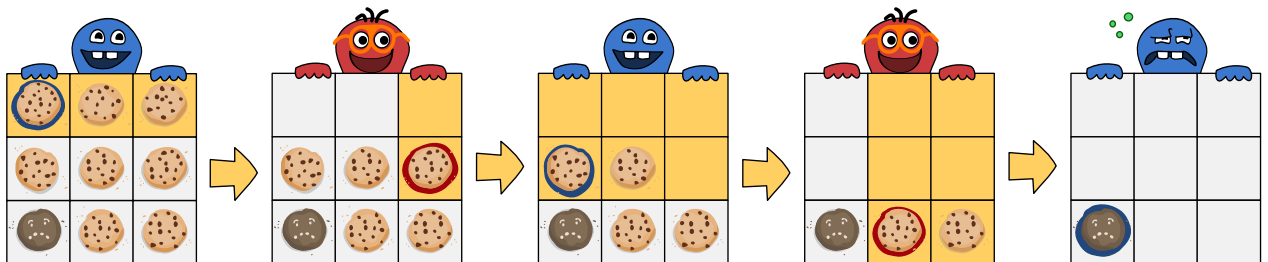


Cleo 🐙 e Dan 🍅 prendono a turno i biscotti dalla scatola finché non è vuota. Cleo ha sempre la precedenza.

È possibile estrarre i biscotti dalla scatola solo in questo modo:

- Selezionare prima un biscotto a scelta.
- Poi bisogna prendere questo biscotto e anche tutti i biscotti dell'area rettangolare **sopra e a destra** di esso.

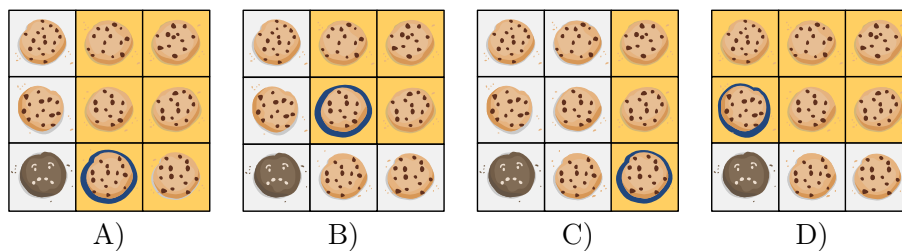
Ecco un esempio. Il biscotto selezionato è contrassegnato in blu o in rosso. Purtroppo, Cleo deve prendere il Frowny alla fine per svuotare la scatola.



L'obiettivo di Cleo è che Dan prenda Frowny, indipendentemente dai biscotti che Dan prende nel mezzo. All'inizio pensa a quali biscotti prendere per essere sicura di raggiungere il suo obiettivo. Anche se Cleo prende altri biscotti nel mezzo, persegue il suo obiettivo.

Ecco quattro opzioni di biscotti con cui Cleo può iniziare. Con una di queste può raggiungere tranquillamente il suo obiettivo.

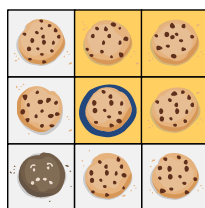
Quale?





Soluzione

La risposta corretta è B:



Se Cleo prende i biscotti in questo modo all'inizio, Dan dovrà sicuramente prendere Frowny e Cleo raggiungerà sicuramente il suo obiettivo. Con le altre tre opzioni, Dan può trovare un modo per evitarlo.

Per spiegarlo, consideriamo la «presa di biscotti» come un gioco:

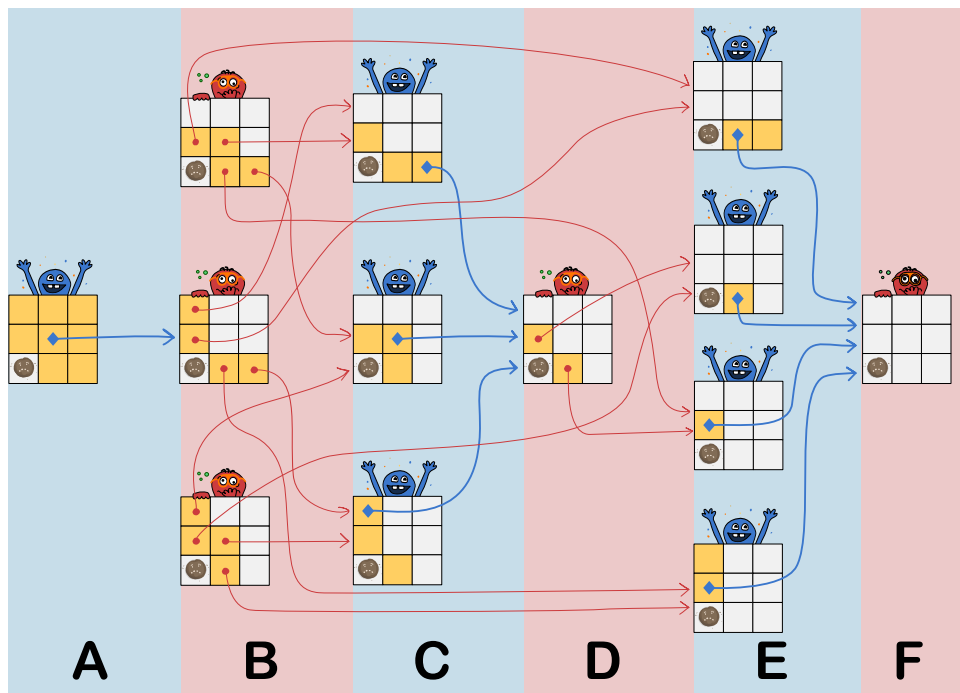
- Quando Cleo o Dan prendono i biscotti, fanno una mossa. Ogni mossa cambia la situazione del gioco (cioè quali biscotti sono al posto giusto).
- Se arriva il proprio turno e si può solo perdere, si è in una *situazione perdente*. In ogni caso, questa è la situazione in cui solo Frowny è rimasto nella scatola. Ma è anche qualsiasi altra situazione in cui si mette l'avversario con tutte le mosse possibili in una ...
- ... *situazione vincente*, in cui l'avversario può creare nuovamente una *situazione perdente* con una sola mossa.

Cleo ha quindi bisogno di una mossa nella situazione iniziale con la quale può mettere Dan in una situazione perdente.

L'immagine mostra alcune situazioni e le suddivide in aree:

- L'area F mostra la situazione perdente, in cui solo Frowny è rimasto nella scatola.
- L'area E mostra quattro situazioni vincenti, da ognuna delle quali una mossa porta alla situazione perdente di cui alla lettera F.
- L'area D mostra una situazione perdente, perché entrambe le mosse possibili conducono a una situazione vincente in E.
- L'area C mostra tre situazioni vincenti: Per ogni situazione c'è una mossa che conduce alla situazione perdente in D.
- Anche l'area B contiene situazioni perdenti: in ogni situazione, tutte le mosse possibili conducono alle situazioni vincenti in C ed E.

Si può considerare che da tutte le altre possibili situazioni non mostrate nella figura, una mossa porta a una situazione perdente. L'immagine mostra solo la situazione di partenza del gioco nell'area A, da cui esattamente la mossa della risposta B (e nessun'altra) porta a una situazione perdente. Cleo può quindi raggiungere il suo obiettivo in modo sicuro solo con questa mossa.



Questa è l'informatica!

Questo gioco può essere visto come una variante del noto gioco di strategia *Nim*. Il gioco utilizza i concetti della *teoria dei giochi*. Un *albero di gioco* che descriva tutte le mosse possibili sarebbe troppo grande per essere rappresentato. Dopo alcune situazioni di gioco i percorsi si diramano, dopo altre no. Catene più lunghe di situazioni di gioco senza diramazioni possono essere rappresentate in modo più compatto (come la freccia nell'immagine precedente). Ogni volta che ricominciamo a giocare e raggiungiamo una situazione di gioco pre-segnalata attraverso altri percorsi, possiamo saltare direttamente avanti (cioè attraverso la freccia) senza dover fare ulteriori calcoli.

L'idea di anticipare le situazioni di gioco e di riutilizzarle abilmente è una strategia generale di risoluzione dei problemi spesso utilizzata in informatica e chiamata *programmazione dinamica*.

Con l'aiuto di un computer, possiamo scrivere un programma che implementa questa strategia e risolve il gioco per un campo di gioco di qualsiasi dimensione. A ogni passo, il programma decide quale mossa deve fare un giocatore (Cleo, per esempio) per creare una situazione vincente.

Parole chiave e siti web

- Nim: <https://it.wikipedia.org/wiki/Nim>
- Teoria dei giochi: https://it.wikipedia.org/wiki/Teoria_dei_giochi
- Programmazione dinamica: https://it.wikipedia.org/wiki/Programmazione_dinamica





Compiti di programmazione

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

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

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



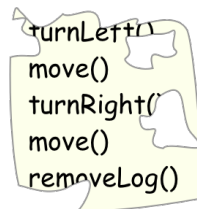


19. Foglio Bagnato

Benno ha bisogno di un tronco per la sua diga e ne vede uno dall'altra parte del lago. Ha già scritto come procedere per raccoglierlo. Purtroppo, il foglio con le istruzioni gli è caduto nel lago. Adesso è in grado di leggere solo l'ultima parte delle istruzioni.

Istruzione	Effetto
<code>move()</code>	Avanza di una casella
<code>turnRight()</code>	Gira di 90 gradi a destra
<code>turnLeft()</code>	Gira di 90 gradi a sinistra
<code>paddle()</code>	Rema fino alla roccia
<code>removeLog()</code>	Raccogli un tronco

Sul suo foglio le ultime cinque istruzioni sono:



Riscrivi le istruzioni mancanti nella parte iniziale del foglio e raccogli il tronco.

Se volete risolvere questo compito online, potete utilizzare il codice QR.





Soluzione

La soluzione è la seguente:

```
paddle()
turnRight()
paddle()
turnLeft()
paddle()
# Queste istruzioni erano ancora sul foglio
turnLeft()
move()
turnRight()
move()
removeLog()
```

Per capire quali istruzioni si trovavano all'inizio del foglio, bisogna eseguire quelle presenti al contrario e in ordine inverso.

- Il tronco viene rimesso a posto invece di `removeLog()`. Benno torna indietro di una casella invece di `move()`. Benno gira a sinistra invece di `turnRight()`. Benno torna indietro di una casella invece di `move()`. Benno gira a destra invece di `turnLeft()`.

Quando questa posizione di partenza è trovata, mancano solo le istruzioni per portare Benno dalla posizione iniziale a quella finale:

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

Invece di `paddle()`, si potrebbe anche usare `move()` per coprire il numero corretto di caselle. Tuttavia, `paddle()` è una scelta migliore qui, poiché con questo comando, Benno può avanzare fino alla pietra con una sola istruzione, indipendentemente dal numero di caselle da superare.

Questa è l'informatica!

Poiché Benno può leggere solo le ultime istruzioni del suo percorso, deve mentalmente ripercorrere i passi all'indietro per ricostruire l'inizio. Questo processo è simile al backtracking, in cui si ripercorre una sequenza di passi per tornare al punto di partenza.

Ogni azione cambia la posizione e la direzione di Benno, modificando lo stato del programma passo dopo passo. Questi cambiamenti di stato sono tipici delle macchine a stati, in cui il programma passa da uno stato all'altro tramite azioni specifiche.



Per ricostruire i passi iniziali, serve una sequenza logica che tenga conto della posizione finale di Benno e delle ultime istruzioni. Questo metodo risolutivo usa lo stato finale come punto di partenza per ricostruire l'intero percorso.

Parole chiave e siti web

- Programmazione: [https://it.wikipedia.org/wiki/Programmazione_\(informatica\)](https://it.wikipedia.org/wiki/Programmazione_(informatica))
- Backtracking: <https://it.wikipedia.org/wiki/Backtracking>



A. Autori dei quesiti

 Esraa Almajhad	 Patrik Klofáč
 Masiar Babazadeh	 Jia-ling Koh
 Leonardo Barichello	 Kitti Krenák
 Liam Baumann	 Greg Lee
 Wilfried Baumann	 Taina Lehtimäki
 Javier Bilbao	 Judith Lin
 Eugenio Bravo	 Yoshiaki Matsuzawa
 Lucia Budinská	 Anja Mejač
 Marta J. Burzanska	 Tom Naughton
 Leonard Busuttil	 Alex Parry
 Che-Chun Chen	 Praphan Pavarangkoon
 Sébastien Combéfis	 Jean-Philippe Pellet
 Kris Coolsaet	 Margot Phillipps
 Christian Datzko	 Zsuzsa Pluhár
  Susanne Datzko-Thut	 Wolfgang Pohl
 Nora A. Escherle	 J.P. Pretti
 Gerald Futschek	 Pedro Ribeiro
 Bence Gaál	 Kirsten Schlüter
 Christian Giang	 Margareta Schlüter
 Thomas Ioannou	 Dirk Schmerenbeck
 Filiz Kalelioğlu	  Eljakim Schrijvers
 Gohar Khachatryan	 Karsten Schulz
 Doyong Kim	 Yannic Schulz
 Injoo Kim	 Giovanni Serafini
 Dong Yoon Kim	 Emil Stankov
 Vaidotas Kinčius	  Jacqueline Staub



 Seiichi Tani

 Ezra Templonuevo

 Christine Vender

 Michael Weigend

 Chris Wetherell

 Kyra Willekes

 Jiangang Yao

 Hongjin Yeh

 Yi Shan Yeh

 Khairul Anwar Mohamad Zaki



B. Partner accademici



<http://www.abz.inf.ethz.ch/>

Ausbildungs- und Beratungszentrum für Informatikunterricht
der ETH Zürich.



<http://www.hepl.ch/>

Haute école pédagogique du canton de Vaud

Scuola universitaria professionale
della Svizzera italiana



<http://www.supsi.ch/>

La Scuola universitaria professionale della Svizzera italiana
(SUPSI)



<https://www.uni-trier.de/>

Universität Trier



C. Sponsoring

HASLERSTIFTUNG

<http://www.haslerstiftung.ch/>



Standortförderung beim Amt für Wirtschaft des Kantons Zürich

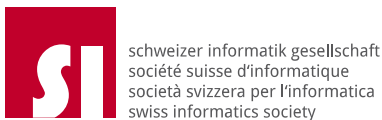


<http://www.ubs.com/>

Wealth Management IT and UBS Switzerland IT



i-factory (Musée des transports, Lucerne)



<https://swissinformatics.org/>

Schweizer Informatik Gesellschaft



<https://www.be.ch/>

Cantone di Berna



<https://www.post.ch/it>

La Posta



<https://www.cyon.ch>

cyon



<http://senarclens.com/>

Senarclens Leu & Partner



D. Ulteriori offerte



CoetryLab: <https://www.coetry-lab.org/>

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



Coding club for girls:

<https://www.epfl.ch/education/education-and-science-outreach/it/promozione-dell-educazione-e-della-scienza/jeune-public/il-coding-club-for-girls/>

Un club gratuito di programmazione per ragazze dagli 11 ai 16 anni. Le ragazze dagli 11 ai 16 anni interessate alla programmazione e all'informatica possono partecipare ai laboratori del Coding Club for Girls. I laboratori possono essere seguiti indipendentemente gli uni dagli altri. Si svolgono in due città diverse: Lugano e Ascona.

I laboratori durano 2h30 e sono supervisionati da studenti e studentesse dell'EPFL. Di tanto in tanto vengono invitati relatori specializzati nel campo delle tecnologie dell'informazione e della comunicazione per presentare i propri percorsi.



Girls code too:

<https://girlscodetoo.ch/>

We support and encourage girls to discover coding and explore a career in tech. We believe that girls are curious, capable, and creative. We work to encourage sparks of talent and curiosity by introducing girls to software tools, different coding languages and processes that are used to build apps, games, websites, and robots.

At GirlsCodeToo our goal is to close the gender gap in the tech industry and engineering universities while boosting the enrollment of girls in STEM-related education programmes and technical apprenticeships.

We provide guidance and support to girls in the following formats:

- Introductory coding workshops
- In-depth workshops in specific coding languages or technologies
- Workshops to boost confidence and interpersonal skills
- Summer camps
- Inspiring #TechTuesday talks by industry professionals



Roteco: <https://www.roteco.ch/it/>

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

010100110101011001001001
010000010010110101010011
010100110100100101000101
001011010101001101010011
010010010100100100100001

SSII

www.svia-ssie-ssii.ch
schweizerischer verein für informatik und
erausbildung // société suisse pour l'infor-
matique dans l'enseignement // società sviz-
zera per l'informatica nell'insegnamento

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.