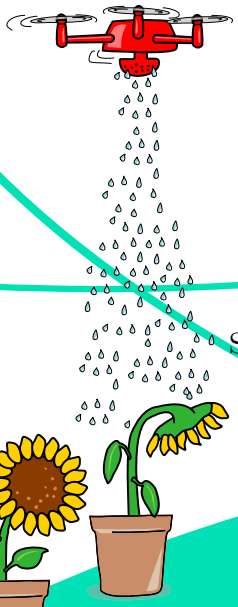


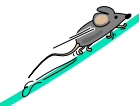


INFORMATIK-BIBER SCHWEIZ
CASTOR INFORMATIQUE SUISSE
CASTORO INFORMATICO SVIZZERA

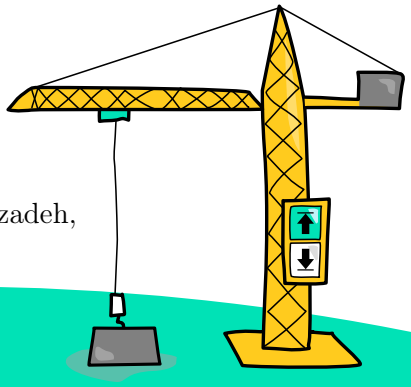
Quesiti e soluzioni 2026

3º e 4º anno scolastico



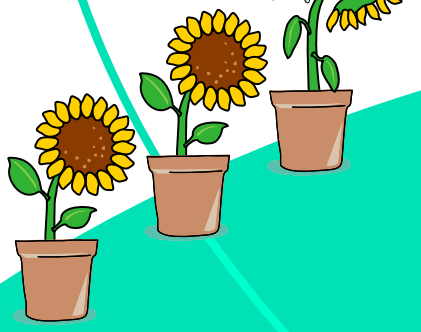


https://www.castoro-informatico.ch/

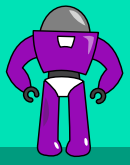


A cura di:

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









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

Hanno collaborato al Castoro Informatico 2026

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

Capi progetto: Nora A. Escherle, Janine Blattner

Un particolare ringraziamento per il lavoro sui quesiti del concorso Svizzero va a:
Juraj Hromkovič, Regula Lacher: ETH Zürich, Ausbildungs- und Beratungszentrum für Informatikunterricht
Christian Datzko, Gymnasium Liestal
Nadine Fröhlich, Università Basel
Urs Meier, Pädagogische Hochschule Luzern
Bernadette Spieler, Pädagogische Hochschule Zürich
Nils Böhnke, Mohadesa Ekhlassi, Dirk Schmerenbeck, Andreas Schneider, Jacqueline Staub und Alexander Wiß: Universität Trier, 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, Margaretha Schlüter, Kirsten Schlüter, Michael Weigend: Bundesweite Informatikwettbewerbe (BWINF), Germania
Wilfried Baumann, Liam Baumann: Österreichische Computer Gesellschaft, Austria
Gerald Futschek, Lukas Lehner: Technische Universität Wien, Austria
Zsuzsa Pluhár, Bence Gaal: ELTE Informatikai Kar, Ungheria

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

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

Per il supporto durante le settimane del concorso ringraziamo:
Valérie Verdan: ETH Zürich, Ausbildungs- und Beratungszentrum für Informatikunterricht

Ringraziamo l'ETH per l'organizzazione e lo svolgimento della finale del Castoro:
Dennis Komm, Hans-Joachim Böckenhauer, Angélica Herrera Loyo, Andre Macejko, Adeline Pittet, Valérie Verdan: ETH Zürich, Ausbildungs- und Beratungszentrum für Informatikunterricht
Per la correzione dei compiti finali:
Niu Daniel, Jay Forrer, Alessandra Frank, Oliver Graf, Kassak Jakuk, Simon Krause, Adeline Richter, Colin Rüegg, Benjamin Schlaich, Alexander Skodinis, Emanuel Skodinis, Moritz Stocker, Philip Whittington, Ursus Wigger, Guiheng Zheng, Mateo Zuluaga

© Castoro Informatico 2026, SSII

Per la traduzione dei compiti finali in francese:
Jean-Philippe Pellet: Haute école pédagogique du canton de Vaud
Responsabili dei workshop per i finalisti:
Nora A. Escherle, Susanne Falkenberg-Thut, Jean-Philippe Pellet, Gloria Pittet, Clemens Pohle, Hannah Rauterberg, Peter Rossmanith, Dirk Schmerenbeck, Benjamin Schmid, Giovanni Serafini

Christoph Frei: Chragokyberneticks (Logo Informatik-Biber Schweiz)
Andrea Leu, Sarah Beyeler, Lea Hasler, 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, Austria e Lussemburgo. Anche la versione in lingua francese era in uso in Lussemburgo.

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

© Castoro Informatico 2026, SSII

INFORMATIK-BIBER SCHWEIZ
CASTOR INFORMATIQUE SUISSE
CASTORO INFORMATICO SVIZZERA

Il Castoro Informatico 2026 è stato organizzato dalla Società Svizzera per l'Informatica nell'Insegnamento (SSII) e sostenuto in modo significativo e generoso dalla Fondazione Hasler. Altri partner e sponsor che hanno sostenuto finanziariamente il concorso sono l'Ufficio per la scuola materna, elementare e la consulenza (AKVB) del Cantone di Berna, l'Ufficio per l'economia AWI del Cantone di Zurigo, l'Assicurazione immobili del Cantone di Basilea Campagna, CYON AG, la Fondazione Informatica Svizzera e UBS.

I seguenti partner accademici ci sostengono nella creazione degli esercizi: il Politecnico federale di Zurigo (ETH Zürich), l'Ausbildungs- und Beratungszentrum für Informatikunterricht, l'Alta scuola pedagogica del Canton Vaud, la Scuola universitaria professionale della Svizzera italiana, l'Alta scuola pedagogica di Lucerna e l'Università di Treviri.

Questo quaderno è stato creato il 23 novembre 2026 con il sistema per la preparazione di testi LaTeX. Ringraziamo Christian Datzko per lo sviluppo del sistema di generazione dei testi che ha permesso di generare le 36 versioni di questa brochure (divise per lingua e livello scolastico). Il sistema è stato riprogrammato basandosi sul sistema precedente, sviluppato nel 2014 assieme a Ivo Blöchliger. Ringraziamo Jean-Philippe Pellet per lo sviluppo del sistema `bebras`, utilizzato dal 2020 per la conversione dei documenti sorgente dai formati Markdown e YAML.

Nota: Tutti i link sono stati verificati il 20.11.2026.

I quesiti sono distribuiti con Licenza Creative Commons Attribuzione – Non commerciale – Condividi allo stesso modo 4.0 Internazionale. Gli autori sono elencati a pagina 55.

Premessa

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

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

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

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

Nel 2026 il Castoro Informatico della Svizzera è stato proposto a cinque differenti categorie d'età, suddivise in base all'anno scolastico:

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

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

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

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

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

© Castoro Informatico 2026, SSII

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

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

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

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

Per ulteriori informazioni:

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

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

© Castoro Informatico 2026, SSII

Indice

Hanno collaborato al Castoro Informatico 2026 i

Premessa iv

Indice vi

1. **La formica timida** 2026-MX-03 1
2. **La mela di Dani** 2026-US-02 5
3. **Giocattoli a batterie** 2026-BR-02 7
4. **Gli adesivi di Amelia** 2026-EC-01 11
5. **Video consigliati** 2026-HU-01a 15
6. **La gru** 2026-CZ-05 19
7. **Taxi in motoslitta** 2026-AT-01 23
8. **Le piastrelle di Ana** 2026-CH-06 27
9. **L'app dei pesci** 2026-DE-08 31
10. **Testa o croce** 2026-IN-05 35
11. **Il giardino di Gerald** 2026-LT-01a 37
12. **Tacos gratis!** 2026-MX-02b 41
13. **Nascondino** 2026-US-03b 45
14. **Un aiuto dai vicini** 2026-DE-90 49

A. **Autori dei quesiti** 55

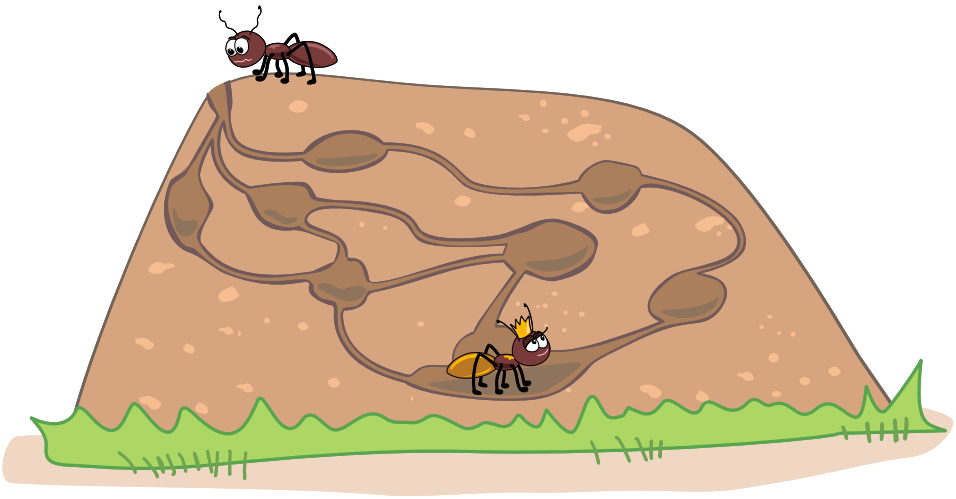
B. **Partner accademici** 56

C. **Sponsoring** 57

© Castoro Informatico 2026, SSII

1. La formica timida

Un formicaio è composto da diverse stanze collegate tra loro da gallerie, in modo tale che ciascuna stanza abbia almeno due entrate.

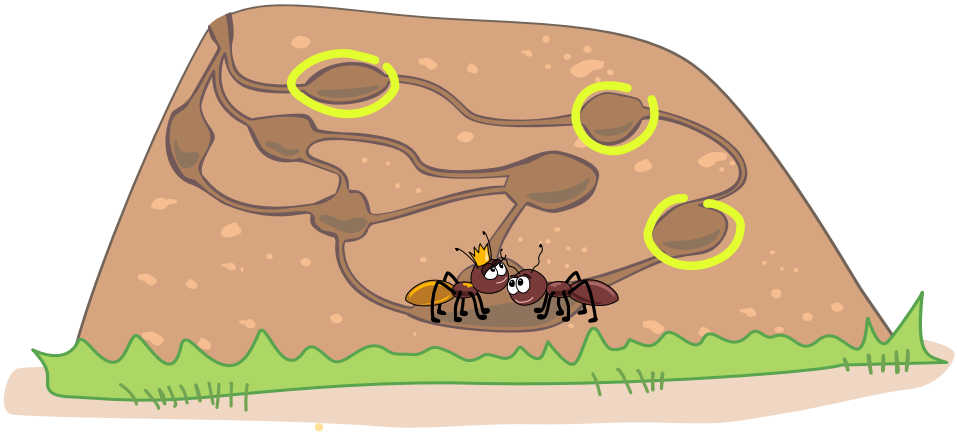


Ani, la formica timida , vuole raggiungere la regina , ma può passare esclusivamente attraverso stanze dotate esattamente di due entrate.

Per quali stanze passa Ani per andare dalla regina?

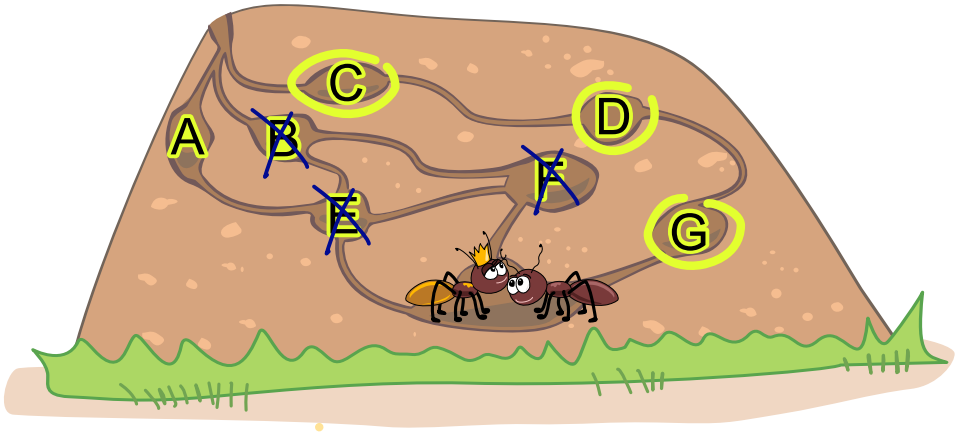
Soluzione

La risposta corretta è la seguente:



Per prima cosa, contiamo le entrate di ciascuna stanza. Ani eviterà le seguenti stanze, poiché presentano più di due entrate:

- La stanza **B** ha 3 entrate
- La stanza **E** ha 4 entrate
- La stanza **F** ha 3 entrate



La stanza **A** ha due entrate, quindi Ani potrebbe in teoria transitarvi; tuttavia, da lì non avrebbe modo di proseguire, poiché l'unica stanza successiva, la **E**, ha quattro entrate.

Le stanze **C**, **D** e **G** dispongono tutte di due entrate. Il tragitto che attraversa le stanze **C**, **D** e **G** è l'unico percorso percorribile da Ani per raggiungere la regina, poiché ogni altra via obbliga a transitare attraverso almeno una stanza con più di due entrate.

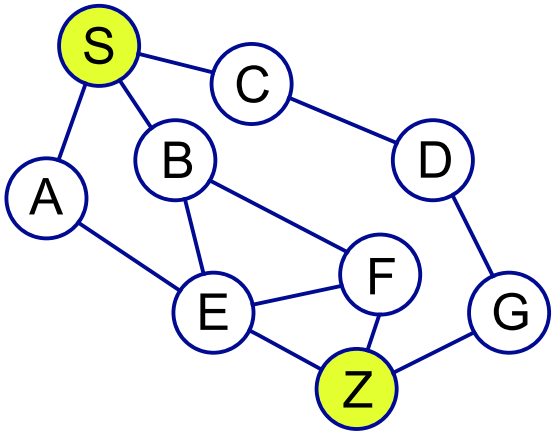
Questa è l'informatica!

Le stanze e le gallerie del formicaio costituiscono nel loro insieme una rete. Esistono innumerevoli esempi di reti nella vita di tutti i giorni: computer connessi a Internet, località collegate da una rete stradale, persone legate da rapporti di amicizia, e così via.

 © Castoro Informatico 2026, SSII

Molti software gestiscono e analizzano le reti (si pensi ai sistemi di navigazione satellitare che elaborano vaste reti stradali). Per rappresentare le reti in modo formale, l'informatica si affida ai *grafi*, strutture costituite da *archi* (che rappresentano i collegamenti) e da *nodi* (che rappresentano gli elementi connessi).

Anche le stanze e le gallerie del formicaio si possono rappresentare come un grafo. I nodi (disegnati come cerchi) sono le stanze, più l'ingresso S, e gli archi (linee) sono le gallerie:



In informatica esistono svariati algoritmi per studiare i grafi e le loro proprietà. In questo problema abbiamo analizzato il *grado di un nodo*, ovvero il numero di archi ad esso incidenti.

La formica Ani ricerca un percorso dall'ingresso S all'uscita Z transitando esclusivamente per nodi di grado 2. Lo stesso principio si applica in ambito informatico alla protezione dei dati sensibili («timidi»), che privilegiano rotte sicure attraverso nodi scarsamente interconnessi: i malintenzionati preferiscono infatti monitorare i nodi altamente connessi e trafficati (i cosiddetti *hub*) per intercettare la maggiore quantità possibile di informazioni.

Parole chiave e siti web

- Teoria dei grafi: `https://it.wikipedia.org/wiki/Teoria_dei_grafi`
- Grado di un nodo: `https://en.wikipedia.org/wiki/Degree_(graph_theory)`
- Instradamento (routing): `https://it.wikipedia.org/wiki/Instradamento`

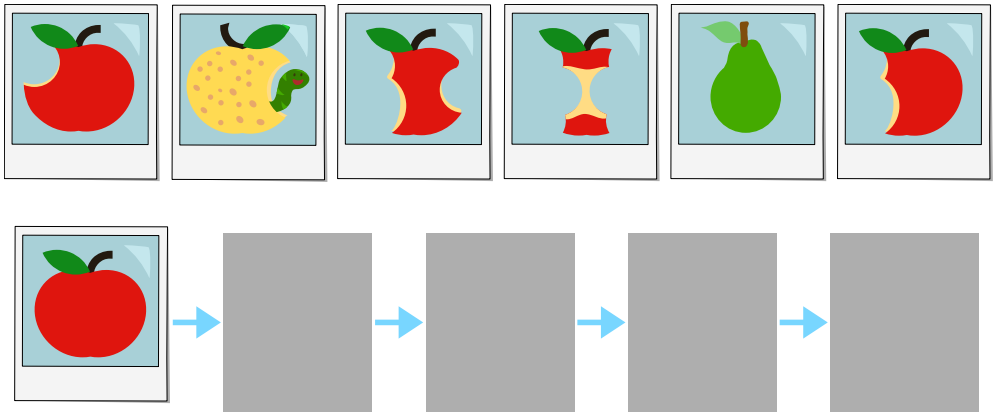


© Castoro Informatico 2026, SSII

2. La mela di Dani

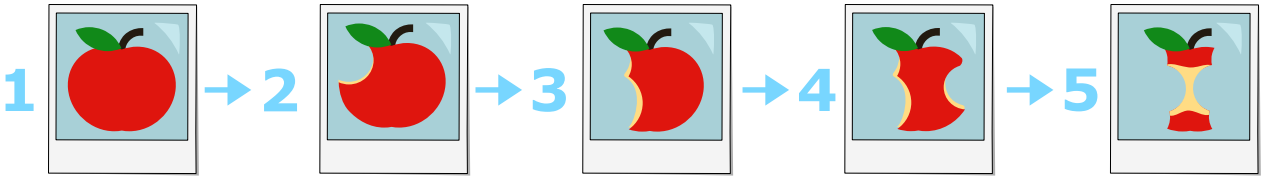
Dani ha mangiato una mela rossa.
Mentre la mangiava, ha scattato alcune foto alla mela.
Ora le foto si sono mescolate, anche con altre foto.

In quale ordine Dani ha scattato le foto della mela rossa?



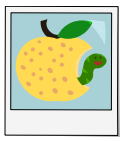
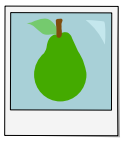
Soluzione

La risposta corretta è la seguente:



Quando si mangia una mela, ne rimane sempre di meno. Nell'ordine corretto, quindi, in ogni foto si deve vedere meno mela che nella foto precedente – proprio come qui sopra:

1. La mela rossa è ancora intera.
2. La mela rossa ha un morso.
3. La mela rossa è mangiata quasi per metà.
4. La mela rossa ha un morso anche dall'altro lato.
5. Della mela rossa è rimasto solo il torsolo.

Le foto  e  non sono foto della mela rossa.

Questa è l'informatica!

Nelle foto di Dani si vede ogni volta una quantità diversa di mela rossa. Per questo si possono confrontare due foto e stabilire con certezza che la foto in cui si vede più mela è stata scattata prima dell'altra. Succede lo stesso con due numeri diversi: si può sempre stabilire con certezza quale dei due è più piccolo. Per questo si può dire che le foto, proprio come i numeri, hanno un ordine.

Quando le cose hanno un ordine, si possono mettere in sequenza: i numeri secondo la loro grandezza e le foto della mela rossa secondo il momento in cui Dani le ha scattate. Questa operazione si chiama anche ordinamento. In questo compito abbiamo ordinato dalla mela più grande alla più piccola, in questo modo: per prima cosa abbiamo preso la foto in cui la mela era più grande. Poi, tra le foto rimaste, abbiamo preso di nuovo quella in cui la mela era più grande. E così via. Questo metodo di ordinamento è conosciuto in informatica e ha persino un nome inglese: *Selection Sort* (ordinamento per selezione). È un metodo molto pratico, per esempio quando si vogliono ordinare i libri di uno scaffale per autore, ma non è particolarmente veloce. Poiché di solito i computer devono ordinare moltissimi dati, l'informatica conosce metodi di ordinamento più veloci e più adatti ai programmi per computer.

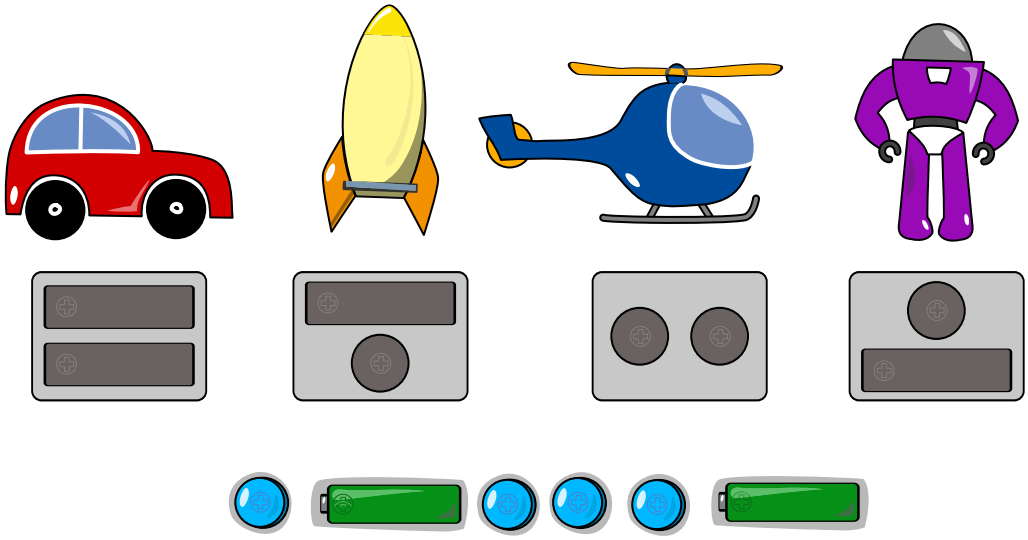
Parole chiave e siti web

- Algoritmo Selection Sort: `https://it.wikipedia.org/wiki/Selection_sort`
- Algoritmo: `https://scratch.mit.edu`, `https://it.wikipedia.org/wiki/Algoritmo`

 © Castoro Informatico 2026, SSII

3. Giocattoli a batterie

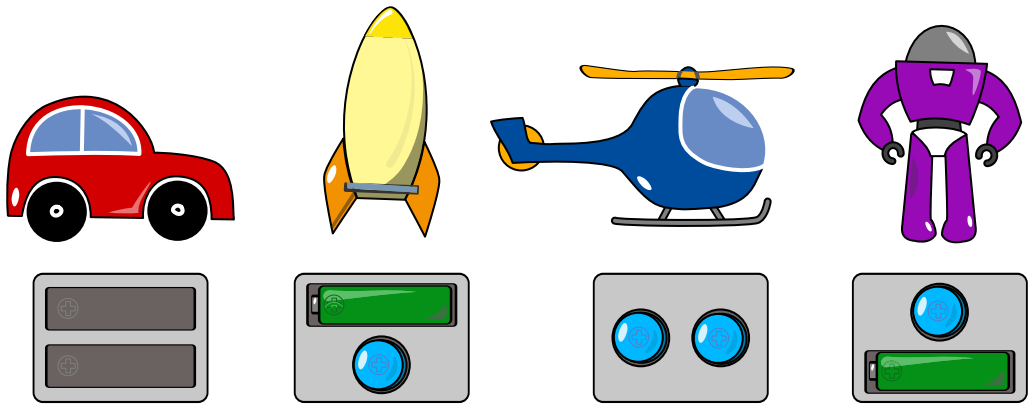
I 4 giocattoli di Bona hanno bisogno di batterie.
I giocattoli funzionano solo quando tutti gli scomparti batterie sono pieni.
Bona vuole che funzionino più giocattoli possibile.



Come bisogna mettere le batterie negli scomparti?

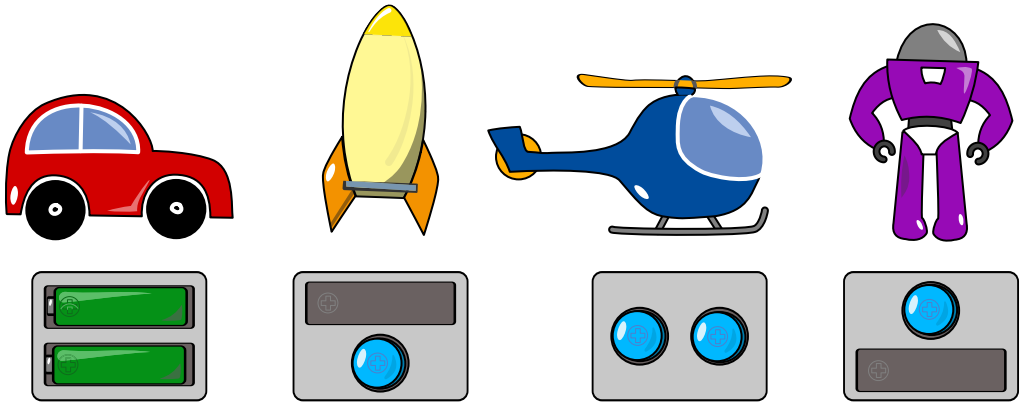
Soluzione

Se metti le pile negli scomparti in questo modo, riusciamo a far funzionare tre giocattoli:



Non è possibile far funzionare tutti e quattro i giocattoli. Per farlo ci servirebbero quattro pile lunghe, ma ce ne sono solo due.

È possibile riporre le batterie in un altro modo? Si potrebbero usare le pile lunghe per la macchinina, invece che per il razzo e il robot. In questo caso però funzionerebbero solo la macchinina e l'elicottero:



Per far funzionare tre giocattoli, bisogna quindi mettere le pile negli scomparti esattamente come sopra.

Questa è l'informatica!

Per trovare una risposta, si potrebbe provare a far funzionare i giocattoli uno dopo l'altro. Meglio un uovo oggi che una gallina domani! Questo modo di procedere «goloso», in cui si vuole avere il prima possibile un giocattolo che funziona, però non dà la soluzione migliore: se iniziamo a sinistra con la macchinina e per ogni giocattolo mettiamo nello scomparto più pile adatte possibile, alla fine funzionano solo due giocattoli (vedi sopra). Chi vuole trovare la soluzione migliore in modo sistematico e sicuro, deve provare tutti i modi possibili di mettere le pile negli scomparti.

In informatica i problemi in cui si cerca la soluzione migliore (detta anche: ottimale) si chiamano *problemi di ottimizzazione*. Quando si usano i computer per cercare soluzioni ottimali, di solito devono elaborare molti dati, con molte risposte possibili. In quel caso il metodo «prova tutto» richiederebbe

 © Castoro Informatico 2026, SSII

troppo tempo. Per questo l'informatica conosce tanti metodi per cercare soluzioni ottimali. Uno di questi si chiama «greedy», in italiano: goloso. Il metodo greedy sceglie sempre come passo successivo quello che promette il guadagno più grande per risolvere il problema. Ma il metodo «greedy» non trova sempre la soluzione ottimale. Come abbiamo visto sopra, è così anche per i giocattoli di Bona.

Parole chiave e siti web

- Algoritmo greedy: `https://it.wikipedia.org/wiki/Algoritmo_greedy`
- Problema di ottimizzazione: `https://it.wikipedia.org/wiki/Problema_di_ottimizzazione`



© Castoro Informatico 2026, SSII

4. Gli adesivi di Amelia

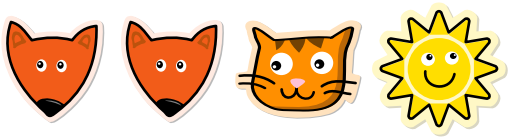
Ad Amelia piace mettere gli adesivi in fila seguendo sempre queste regole:

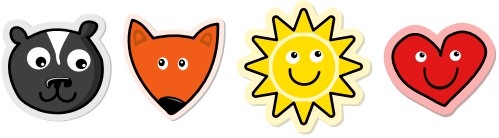
All'inizio posiziona due o tre adesivi con animali. Dopodiché aggiunge altri adesivi in base a queste condizioni:

- Se gli adesivi con animali sono due, subito dopo viene inserita una stella ,
  se sono tre, viene inserito un sole .
- Se gli adesivi con animali sono tutti diversi, alla fine viene inserito anche un cuore .

Solo una di queste sequenze di adesivi è stata creata da Amelia, quale?

A) 

B) 

C) 

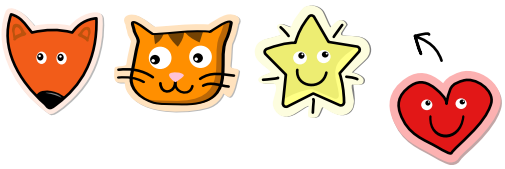
D) 

Soluzione

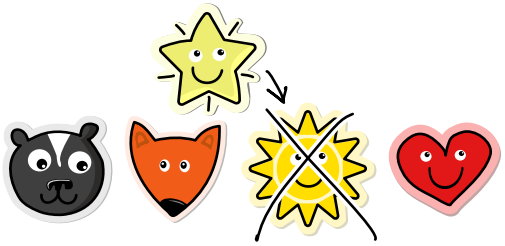
La risposta corretta è B.

La sequenza della risposta B contiene tre adesivi con animali, seguiti da un sole: corrisponde esattamente al modo in cui agisce Amelia. Inoltre, poiché i tre adesivi con animali non sono tutti diversi, non è presente alcun cuore alla fine, rispettando perfettamente la regola.

Le altre sequenze non possono essere state create da Amelia per i seguenti motivi:



La sequenza della risposta A presenta due adesivi con animali seguiti da una stella, come previsto. Tuttavia, poiché i due adesivi con animali sono diversi tra loro, Amelia avrebbe dovuto aggiungere un cuore alla fine, che qui invece manca.



La fila della risposta C ha due adesivi con animali. Amelia metterebbe in seguito una stella, ma qui c'è un sole. Il cuore però è giusto: Amelia lo metterebbe, perché i due adesivi con animali sono diversi.



La fila della risposta D ha tre adesivi con animali. Amelia metterebbe un sole, ma qui manca. Anche in questa fila il cuore è giusto: Amelia lo metterebbe, perché i tre adesivi con animali sono tutti diversi.

Questa è l'informatica!

Amelia aggiunge gli adesivi supplementari seguendo una regola fissa, il che ci permette di stabilire quali sequenze siano opera sua. I codici a barre funzionano in modo analogo alla fila di adesivi di questo problema. Li troviamo comunemente sulle confezioni dei prodotti, come nell'immagine qui sotto:



 © Castoro Informatico 2026, SSII

Il codice a barre scansionato alla cassa del negozio corrisponde al numero a 13 cifre stampato sotto di esso. In origine, tuttavia, il codice identificativo del prodotto è composto da sole 12 cifre: la 13ª cifra (quella finale) viene calcolata a partire dalle precedenti applicando una regola matematica fissa. Durante la lettura, il lettore di cassa ripete il calcolo per verificare l'assenza di errori (dovuti, ad esempio, a etichette danneggiate o sporche). Per questo motivo, l'ultima cifra prende il nome di *cifra di controllo*. Le cifre di controllo sono impiegate in moltissimi altri codici e codifiche numeriche, come nei codici ISBN dei libri o nei codici IBAN per i conti bancari. Quando i dati vengono elaborati automaticamente dai sistemi informatici, risulta fondamentale rilevare tempestivamente eventuali errori di lettura o trasmissione per evitare inconvenienti spiacevoli: ad esempio, per evitare di pagare una grande confezione di cioccolatini al prezzo di una bottiglietta d'acqua a causa di una scansione errata!

Parole chiave e siti web

- Cifra di controllo: `https://it.wikipedia.org/wiki/Numero_di_controllo`
- Codice EAN: `https://it.wikipedia.org/wiki/European_Article_Number`



© Castoro Informatico 2026, SSII

5. Video consigliati

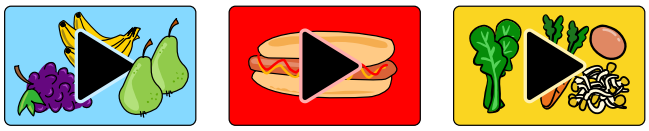
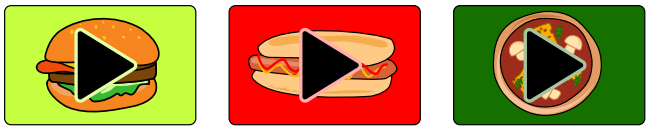
A Zara piace guardare i video su ZickZock.
ZickZock le propone sempre nuovi video. Funziona così:

- ZickZock cerca le persone che hanno visto almeno due video uguali a quelli visti da Zara.
- Dopodiché propone tutti i video che queste persone hanno visto, ma che Zara non ha ancora visto.

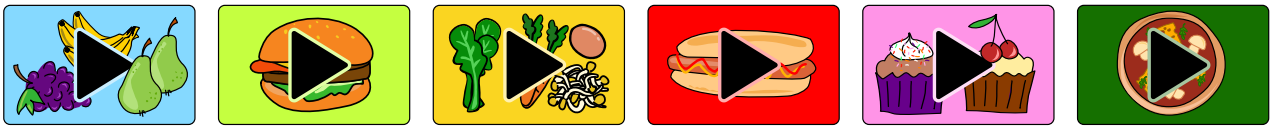
Zara ha visto questi video:



I suoi amici Anna, Bert e Chloe hanno visto questi video:

Persona	**Video**
Anna	
Bert	
Chloe	

Quali di questi video propone ZickZock a Zara?



Soluzione

La risposta corretta è la seguente:

ZickZock propone a Zara questi video:



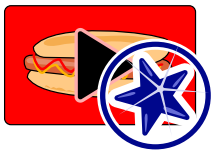
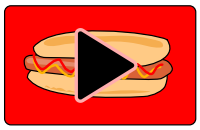
Trovi la risposta corretta se esamini la tabella con i video degli amici di Zara esattamente come fa ZickZock:

 Cerca nella tabella le persone che hanno visto almeno due video uguali a quelli visti da Zara.

 Nelle righe delle persone che hai trovato, cerca i video che Zara non ha ancora visto: sono questi i video che ZickZock consiglia.

Proviamo:

- **Anna** ha visto esattamente due video uguali a quelli di Zara, cioè i primi due. ZickZock propone quindi il terzo video, perché Zara non l'ha ancora visto.
- **Bert** ha visto esattamente due video uguali a quelli di Zara, cioè quello a sinistra e quello a destra. ZickZock propone quindi il video al centro, perché Zara non l'ha ancora visto.
- **Chloe** ha visto un solo video uguale a quelli di Zara, cioè il primo. Perciò ZickZock non propone nessun video di Chloe.

Persona	**Video visti**		
Anna			
Bert			
Chloe			

Questa è l'informatica!

ZickZock è un *sistema di raccomandazione*. Forse conosci già qualcosa di simile grazie ad app come Netflix, YouTube o Spotify, che ti propongono video, canzoni o altri contenuti che potrebbero piacerti.

 © Castoro Informatico 2026, SSII

L'app ZickZock di questo compito usa una forma semplice di un metodo che in informatica si chiama *filtraggio collaborativo*, in cui i suggerimenti nascono dal confronto con altre persone. L'algoritmo di suggerimento dell'app cerca persone a cui piacciono le stesse cose che piacciono a te, o cose simili. Se queste persone hanno visto un video o ascoltato una canzone che tu non conosci ancora, l'app può proporti proprio quel video o quella canzone.

I sistemi di raccomandazione possono però usare anche altri metodi. Nelle *raccomandazioni basate sui contenuti*, l'algoritmo dell'app tiene conto soprattutto dei contenuti che ti hanno interessato finora. Se in un'app di notizie leggi spesso articoli sullo spazio, l'app potrebbe proporti altri articoli su razzi, pianeti o nuove missioni spaziali.

Questi suggerimenti sono utili, perché ti aiutano a trovare più facilmente cose che corrispondono ai tuoi interessi. Tuttavia, possono anche contribuire a creare una *bolla di filtraggio*: l'app ti mostra soprattutto cose simili a quelle che ti piacciono già, e così rischi di perderti cose importanti o sorprendenti di altri ambiti.

Parole chiave e siti web

- Sistema di raccomandazione: `https://it.wikipedia.org/wiki/Sistema_di_raccomandazione`
- Filtraggio collaborativo: `https://it.wikipedia.org/wiki/Collaborative_filtering`
- Bolla di filtraggio: `https://it.wikipedia.org/wiki/Bolla_di_filtraggio`



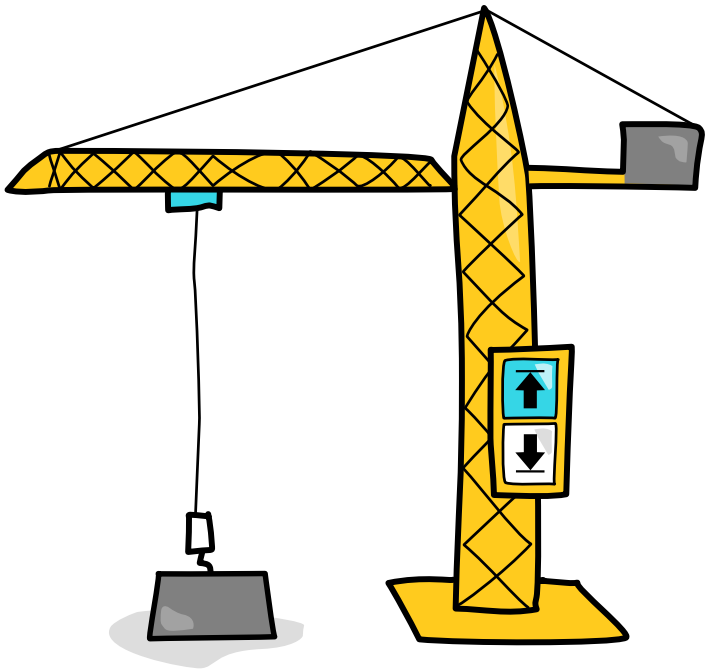
© Castoro Informatico 2026, SSII

6. La gru

Una gru ha due pulsanti: uno per «su»  e uno per «giù» .

Se premi , il carico sale fino in cima.

Se premi , il carico scende fino in fondo.



All'inizio il carico si trova in basso. Poi i pulsanti vengono premuti in questo ordine:



Quante volte il carico sale fino in cima?

A) 4 volte

B) 5 volte

C) 8 volte

D) 10 volte

Soluzione

La risposta corretta è B.

Il carico sale fino in cima solo quando si trova in basso e poi viene premuto .

Questo succede una volta proprio all'inizio.

Succede poi altre quattro volte, cioè quando il carico scende fino in fondo con  e subito dopo risale fino in cima con :



Se invece il carico è già in alto grazie a  e si preme di nuovo , non succede niente. In tutto, quindi, il carico sale fino in cima cinque volte, come nella risposta B.

Risposta A: se hai scelto «4 volte», forse non hai contato la salita all'inizio.

Risposta C: se hai scelto «8 volte», forse hai pensato che con  il carico salga sempre fino in cima.

Risposta D: se hai scelto «10 volte», forse hai contato anche le discese fino in fondo.

Questa è l'informatica!

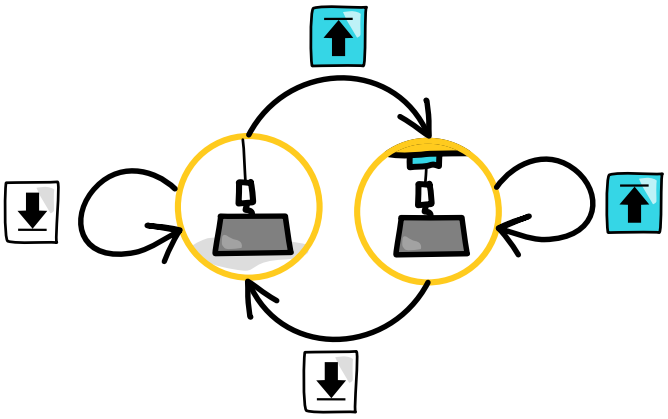
Con i due pulsanti puoi dare alla gru due comandi diversi: «Porta il carico fino in cima!» e «Porta il carico fino in fondo!». Come abbiamo già visto sopra, però, non tutti i comandi hanno un effetto. L'effetto dei comandi dipende infatti dallo *stato* in cui si trova il carico: in basso o in alto. Questa tabella mostra l'effetto dei comandi a seconda dello stato.

Comando	**Stato: carico in basso** 	**Stato: carico in alto** 
	Il carico sale.	Non succede niente.
	Non succede niente.	Il carico scende.

In informatica il funzionamento di macchine come la gru di questo compito viene spesso descritto con l'aiuto degli stati. Per farlo non si usano solo tabelle, ma spesso anche diagrammi, cioè disegni. Uno stato viene rappresentato con un cerchio, mentre delle frecce mostrano l'effetto dei comandi nei diversi stati.

 © Castoro Informatico 2026, SSII

Il diagramma degli stati della gru è questo:



Parole chiave e siti web

- *Sequenza di comandi:* `https://it.wikipedia.org/wiki/Struttura_di_controllo`
- *Automa a stati finiti:* `https://it.wikipedia.org/wiki/Automa_a_stati_finiti`
- *Automa a stati finiti (in tedesco):* `https://informatik.mygymer.ch/ef2023/02-automaten/01-automaten.html`



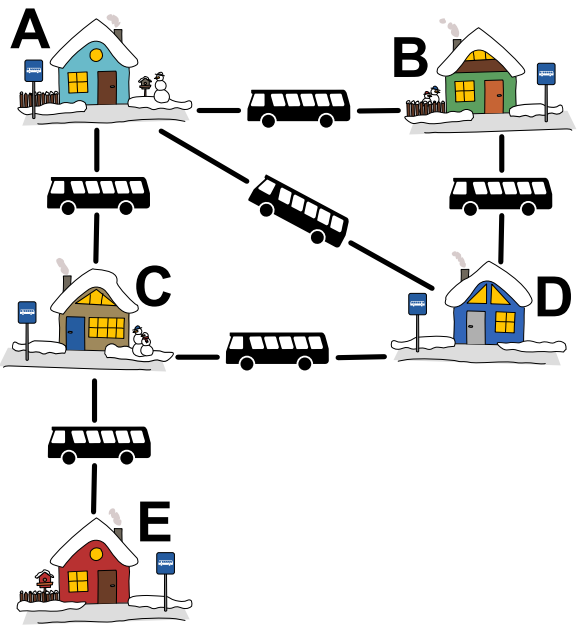
© Castoro Informatico 2026, SSII

7. Taxi in motoslitta

È inverno a Castoropoli!

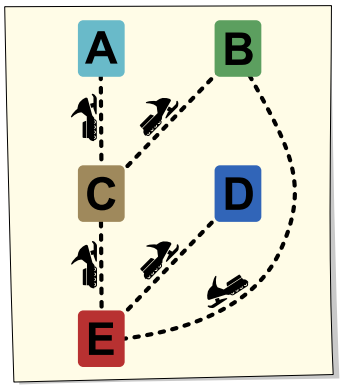
Nel villaggio ci sono cinque fermate, da A a E. La mappa a destra mostra i collegamenti diretti in autobus tra le fermate. Su ogni collegamento gli autobus viaggiano in entrambe le direzioni.

Lara ha dei taxi in motoslitta. Offre un collegamento diretto in taxi tra due fermate sempre e solo quando tra queste due fermate *non* c'è un collegamento diretto in autobus.

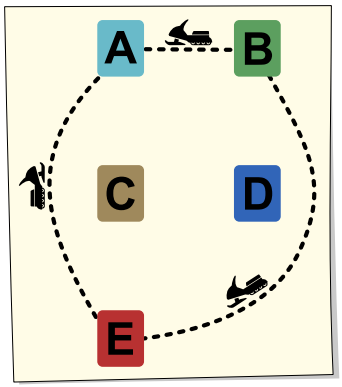


Quale mappa mostra i collegamenti diretti in taxi di Lara?

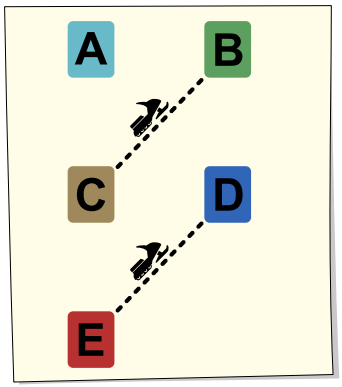
A)



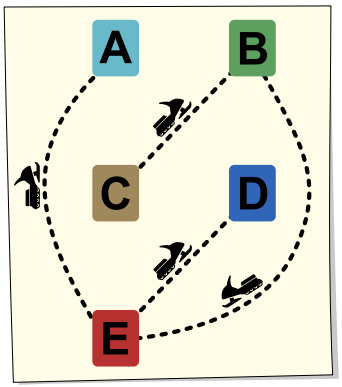
B)



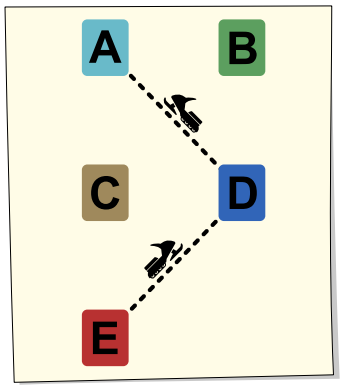
C)



D)



E)



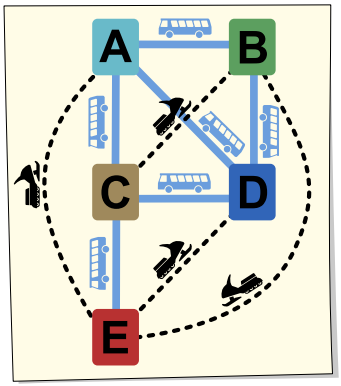
Soluzione

La risposta corretta è D.

Ecco come puoi trovare la risposta corretta: per ogni coppia di fermate tra le quali non c'è un collegamento diretto in autobus, disegni una linea tratteggiata sulla mappa degli autobus. Per farlo, scegli una dopo l'altra tutte le fermate da A a E. Per la fermata scelta verifichi, per ciascuna delle altre fermate, se tra le due c'è un collegamento diretto in autobus. Se non c'è, disegni una linea tra le due fermate.

Un esempio: dalla fermata **B** ci sono collegamenti diretti in autobus verso **A** e verso **D**, ma non verso **C** e verso **E**. Quindi disegni una nuova linea tra **B** e **C** e un'altra tra **B** ed **E**.

Alla fine si ottiene questa mappa, in cui tra due fermate qualsiasi c'è sempre un collegamento, in autobus oppure in taxi in motoslitta (le nuove linee sono tratteggiate):



Le nuove linee sono i collegamenti diretti che mancano. Poiché Lara offre un collegamento diretto in taxi proprio lì, le nuove linee (insieme alle fermate) formano esattamente la mappa dei taxi in motoslitta di Lara – come nella risposta **D**.

Questa è l'informatica!

Le cinque fermate, insieme ai collegamenti diretti in autobus, formano una rete: la rete degli autobus di Castoropoli. Le reti hanno un ruolo importante in informatica, per esempio le reti stradali e ferroviarie nei navigatori. Quando un'app collega tra loro i suoi utenti come contatti, si parla anche di «reti sociali» (in inglese *social network*).

Per rappresentare le reti, l'informatica usa una struttura chiamata *grafo*. Un grafo è composto da *nodi* e da *archi*. I nodi rappresentano le cose collegate tra loro, mentre gli archi sono coppie di nodi e rappresentano i collegamenti. Anche la rete degli autobus di Castoropoli si può rappresentare come un grafo: le fermate sono i nodi e i collegamenti diretti in autobus sono gli archi.

Nel grafo della rete degli autobus, Lara cerca proprio le coppie di nodi (cioè le coppie di fermate) che non sono archi (cioè che non hanno un collegamento diretto in autobus). In informatica si direbbe che Lara cerca il *grafo complementare* del grafo della rete degli autobus. Un grafo complementare ha

 © Castoro Informatico 2026, SSII

esattamente gli stessi nodi del grafo originale, ma inverte i collegamenti: una coppia di nodi è un arco del grafo complementare se e solo se non è un arco del grafo originale.

La mappa dei taxi in motoslitta di Lara mostra quindi il grafo complementare del grafo della rete degli autobus. È interessante notare che alcuni problemi difficili da risolvere su un grafo possono diventare più semplici sul suo grafo complementare.

Parole chiave e siti web

- Grafo: `https://it.wikipedia.org/wiki/Grafo`
- Grafo complementare: `https://it.wikipedia.org/wiki/Grafo_complemento`



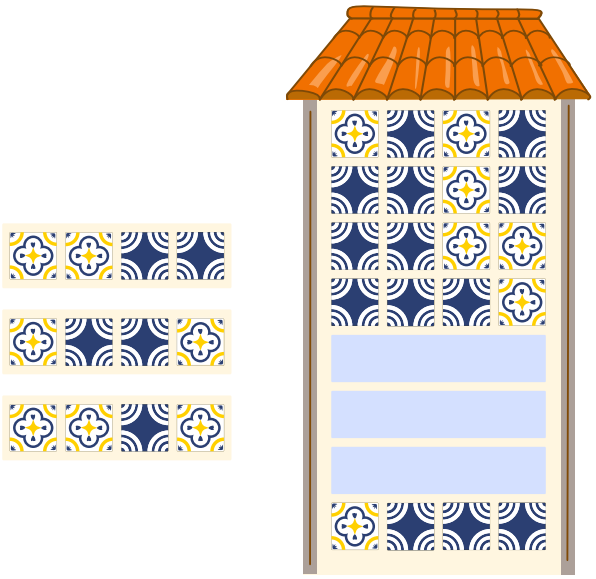
© Castoro Informatico 2026, SSII

8. Le piastrelle di Ana

Ana vive in Portogallo. Sulla facciata della sua casa sta posando delle file di quattro piastrelle ciascuna (i cosiddetti azulejos).

Ogni fila di piastrelle deve essere diversa dalla fila che si trova subito sopra in un solo punto. Solo così le piastrelle sono posate correttamente.

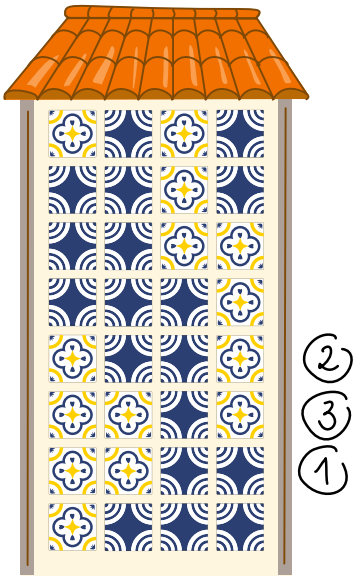
Ana non ha ancora finito: mancano tre file.



Come vanno posate correttamente sulla parete le tre file di piastrelle?

Soluzione

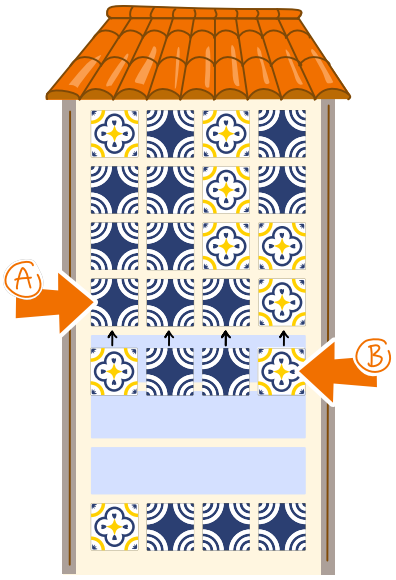
La risposta corretta è la seguente:



In questa disposizione, infatti, da una fila all'altra cambia sempre esattamente una piastrella. Le posizioni che cambiano sono evidenziate con un riquadro verde.



Ecco come trovare la risposta corretta: prendi una fila (A) che è già sulla parete e sotto la quale manca ancora una fila, e cerca l'unica fila (B) che è diversa in un solo punto. Ora manca una fila sotto questa nuova fila: cerca di nuovo l'unica fila adatta. In questo modo posi una fila dopo l'altra, finché la parete è completa.



 © Castoro Informatico 2026, SSII

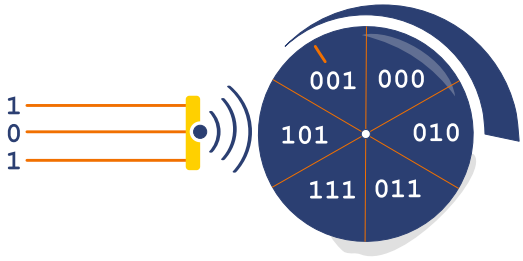
Questa è l'informatica!

Le piastrelle di questo compito hanno soltanto due motivi diversi. Per questo le file si possono immaginare anche come sequenze di zeri e di uno, cioè come file di bit: 0 per un motivo, 1 per l'altro. Per la parete di Ana vale questa regola: una fila deve differire da una fila vicina in un solo punto. Una serie di file di bit per la quale vale la stessa regola (ciascuna fila di bit differisce dalla precedente in un solo bit) si chiama *codice Gray*. La parete di Ana corrisponde quindi a questo codice Gray:

0101, 1101, 1100, 1110, 0110, 0010, 0011, 0111

Attenzione: in un codice Gray bisogna immaginare le file di bit disposte in cerchio. Per questo anche la prima fila deve differire dall'ultima in un solo bit.

I codici Gray vengono usati, tra l'altro, nei dispositivi tecnici. Un esempio: su un altoparlante c'è una manopola con cui si regola il volume, quindi la posizione della manopola determina il volume. Mentre si gira la manopola, un sensore trasmette la posizione attuale a un computer attraverso tre linee. Ciascuna linea trasmette il valore 0 oppure il valore 1, in modo tale che si possano distinguere otto posizioni consecutive della manopola. Se si usa un codice Gray, girando la manopola cambia sempre il valore di una sola linea. In questo modo si evitano bruschi cambiamenti di volume, che con un altro codice potrebbero invece verificarsi.



Parole chiave e siti web

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



© Castoro Informatico 2026, SSII

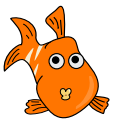
9. L'app dei pesci

L'app dei pesci permette di modificare le immagini dei pesci, utilizzando uno smiley per indicare in che modo.

Nello smiley, occhi, naso e bocca possono essere piccoli o grandi. In questo smiley è tutto piccolo:



Successivamente viene mostrata l'immagine di un pesce. L'app può ingrandire o rimpicciolire gli occhi, la bocca e le pinne. In questo pesce sono tutti piccoli:

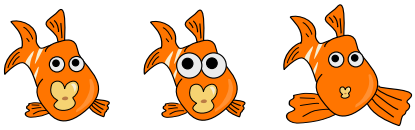


Se ora scegliamo nell'app uno smiley che presenta alcuni elementi grandi, una determinata parte del pesce si ingrandirà di conseguenza. Ad esempio:

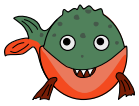
Smiley



Pesce



Ecco l'immagine di un altro pesce. Anche in questo pesce è tutto piccolo:



Nell'app scegli questo smiley:

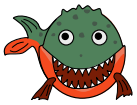


Come diventa allora l'immagine del pesce?

A) 

B) 

C) 

D) 

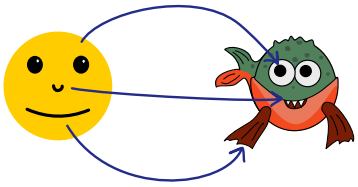
Soluzione

La risposta corretta è B.



Dagli esempi si deducono le seguenti regole:

- Se lo smiley ha gli occhi grandi, nel pesce diventano grandi gli occhi.
- Se lo smiley ha il naso grande, nel pesce diventa grande la bocca.
- Se lo smiley ha la bocca grande, nel pesce diventano grandi le pinne.



Nel problema proposto, lo smiley ha gli occhi grandi, il naso piccolo e la bocca grande. Di conseguenza, in base alle regole viste sopra, nel pesce si ingrandiranno gli occhi e le pinne, mentre la bocca rimarrà piccola.

Le altre opzioni sono errate, poiché in esse: ...

- Risposta A: ... gli occhi sono piccoli.
- Risposta C: ... la bocca è grande.
- Risposta D: ... gli occhi sono piccoli, la bocca è grande e le pinne sono piccole.

Questa è l'informatica!

L'app genera una nuova immagine combinando due immagini di input. Alcune caratteristiche dello smiley vengono trasferite a parti specifiche del pesce: la dimensione degli occhi dello smiley determina quella degli occhi del pesce, la grandezza del naso regola la bocca e la grandezza della bocca controlla le pinne. Questo esercizio illustra il concetto fondamentale di un filtro: le proprietà di un oggetto vengono trasferite in modo sistematico a quelle di un altro. L'immagine del pesce fornisce la forma di base, mentre lo smiley stabilisce quali elementi subiscono modifiche.

In informatica, i filtri rivestono un ruolo cruciale in numerose tecniche di elaborazione delle immagini. I software moderni analizzano i tratti presenti in un'immagine (come volti, occhi o altri oggetti) e utilizzano tali informazioni per modificarne o analizzarne un'altra. Un esempio celebre è lo *style transfer* (trasferimento di stile), in cui lo stile visivo di un'immagine – ad esempio la tecnica pittorica di un quadro – viene applicato a un'altra.

Mentre l'app dei pesci elabora le immagini seguendo regole fisse, nello style transfer e in molti altri metodi moderni le modalità di elaborazione non vengono definite manualmente dall'uomo, ma

apprese da modelli di apprendimento automatico (machine learning) a partire da numerosi esempi di addestramento.

Parole chiave e siti web

- Mapping (funzione di ordine superiore): `https://it.wikipedia.org/wiki/Funzione_di_ordine_superiore`
- Style transfer: `https://en.wikipedia.org/wiki/Neural_style_transfer`



© Castoro Informatico 2026, SSII

10. Testa o croce

Ogni moneta dei castori ha due facce: «testa»  e «croce» .

Alex ha davanti a sé 7 monete:



Ora decide di girare esattamente due monete per un certo numero di volte.

In quale disposizione NON si troveranno MAI le monete di Alex?

A) 

B) 

C) 

D) 

Soluzione

La risposta corretta è B.

Ogni volta che Alex capovolge esattamente due monete, su di esse compaiono facce diverse rispetto a prima. Si possono verificare tre casi:

1. Entrambe le monete ora mostrano croce, mentre prima mostravano testa,
2. entrambe ora mostrano testa, mentre prima mostravano croce, oppure
3. una ora mostra testa invece di croce e l'altra ora mostra croce invece di testa.

Il *numero di teste* (ovvero la quantità di monete che mostrano testa) varia di 2 (nei casi 1 e 2) oppure resta invariato, cambiando di 0 (nel caso 3). Di conseguenza, se il numero iniziale di teste è pari, rimarrà sempre pari; se invece è dispari, rimarrà sempre dispari.

All'inizio le monete di Alex mostrano 2 teste, un numero dunque pari. Di conseguenza, capovolgendole a due a due, Alex potrà ottenere esclusivamente configurazioni in cui il numero di teste è pari, come nelle risposte A (4 teste), C (4 teste) e D (2 teste). Le monete non potranno mai trovarsi in una disposizione con un numero dispari di teste. Soltanto nella risposta B il numero di teste è dispari, ovvero pari a 3.

Questa è l'informatica!

I numeri interi possono essere pari o dispari; questa proprietà è nota come parità. Come visto nella spiegazione precedente, la parità del numero di teste nella fila di Alex non può subire variazioni finché si girano sempre e solo due monete alla volta.

In informatica, una proprietà che rimane immutata in seguito all'applicazione di determinate operazioni prende il nome di invariante. Individuare l'invariante di questo problema (la parità del numero di teste) permette di individuare la soluzione corretta con estrema rapidità, senza dover procedere per tentativi.

Le invarianti trovano frequente impiego nell'ambito informatico, ad esempio per:

- per verificare se gli algoritmi funzionano correttamente;
- per dimostrare che certi stati non possono essere raggiunti; oppure
- per analizzare sistemi con molti stati possibili.

Parole chiave e siti web

- Invariante: `https://it.wikipedia.org/wiki/Invariante_(informatica)`
- Parità: `https://it.wikipedia.org/wiki/Bit_di_parità`

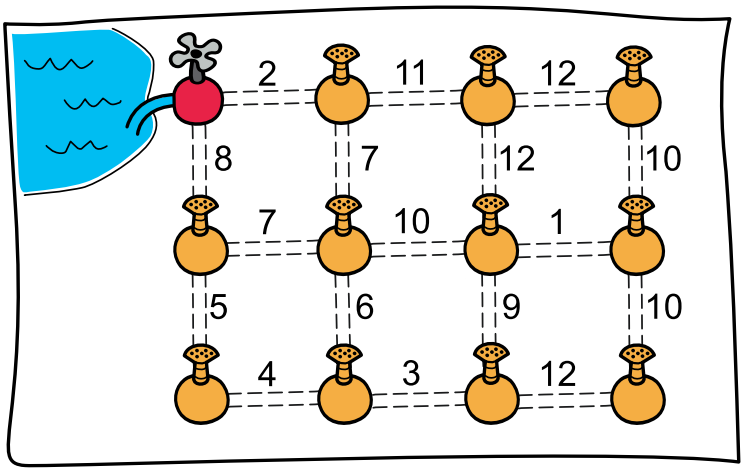
 © Castoro Informatico 2026, SSII

11. Il giardino di Gerald

Il giardiniere Gerald ha installato un nuovo impianto di irrigazione nel suo giardino fiorito.

Per farlo, Gerald ha utilizzato lo schema sottostante, in cui sono illustrati:

- la pompa 
- tutti gli irrigatori  e
- tutti i tratti ⚌⚌⚌⚌⚌ in cui è possibile posare i tubi.



Nello schizzo è indicato anche quanti metri di tubo servono in ogni tratto.

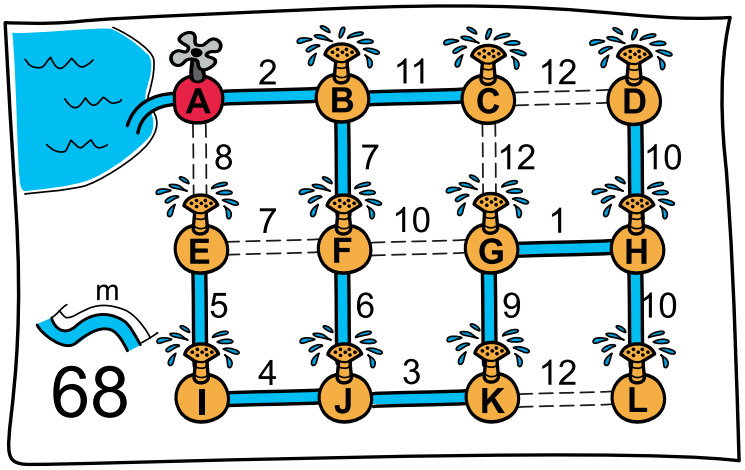
Gerald non ha posato i tubi in ogni tratto disponibile, ma solo il quantitativo necessario affinché

- tutti gli irrigatori siano raggiunti dall'acqua della pompa **e**
- impiegando il minor numero possibile di metri di tubo in totale.

In quali tratti Gerald ha posato i tubi?

Soluzione

La risposta corretta è la seguente:



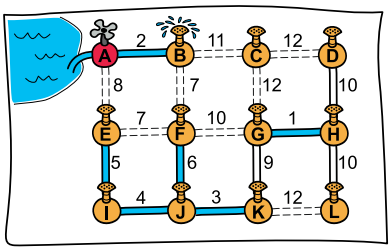
Per semplificare la spiegazione, abbiamo etichettato gli irrigatori con delle lettere e distinto i tratti in base alla lunghezza del tubo richiesto. Parleremo inoltre di *ciclo* quando, partendo da un irrigatore (o dalla pompa) e seguendo i tubi installati, è possibile tornare al punto di partenza senza percorrere alcuno tratto più di una volta.

Per trovare la risposta corretta si può usare la seguente strategia:

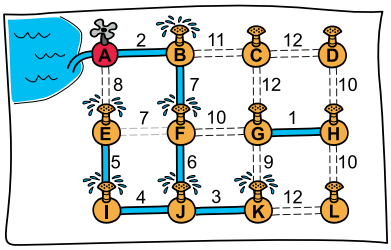
- Esamina i tratti in ordine di lunghezza, partendo dal più corto.
- Posa un tubo in un tratto se e solo se così *non* si forma un ciclo.

Perché è opportuno evitare i cicli? Un ciclo di tubi richiede più materiale del necessario: rimuovendo un tubo da un ciclo, infatti, tutti gli irrigatori continuano comunque a ricevere l'acqua. Per esempio, se nel giardino di Gerald abbiamo già posato i tubi di lunghezza 4, 5 e 6 e ora posiamo anche il tubo di 7 metri tra E e F, si forma il ciclo E, F, J, I, E, senza che nessun irrigatore in più riceva l'acqua. Il tubo tra E e F si può quindi risparmiare. Siccome la strategia comincia dai tratti (cioè dai tubi) più corti, in ogni possibile ciclo si risparmia il tubo più lungo. Così, in totale, servono meno metri di tubo possibile.

La strategia descritta sopra porta sempre a una posa dei tubi che usa meno metri di tubo possibile e collega tutti gli irrigatori alla pompa. Per il giardino di Gerald la strategia funziona così:

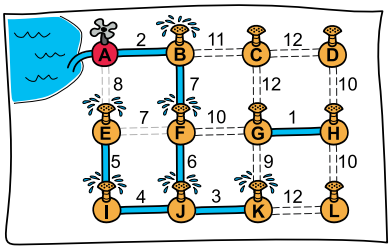


In tutti i tratti di lunghezza da 1 a 6 viene posato un tubo, perché così non si forma nessun ciclo.

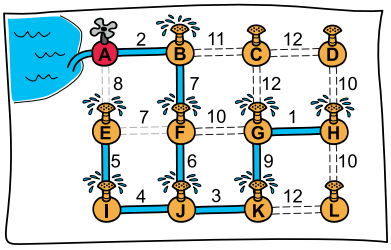


Nei tratti di lunghezza 7

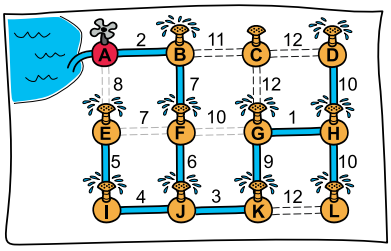
- tra B e F viene posato un tubo, ma
- tra E e F non viene posato nessun tubo, perché si formerebbe il ciclo E, F, J, I, E.



Tra A ed E (lunghezza 8) non viene posato nessun tubo, perché si formerebbe il ciclo A, B, F, J, I, E, A.

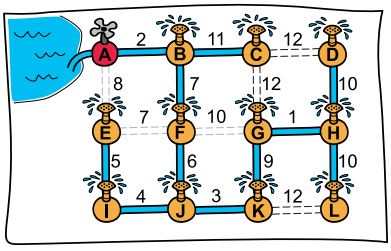


Tra G e K (lunghezza 9) viene posato un tubo.



Nei tratti di lunghezza 10

- viene posato un tubo tra D e H e uno tra H e L, ma
- tra F e G non viene posato nessun tubo, perché si formerebbe il ciclo F, G, K, J, F.



Tra B e C (lunghezza 11) viene posato un tubo.

Nei tratti di lunghezza 12 non viene posato nessun tubo, perché ogni volta si formerebbe un ciclo; sicuramente riesci a scoprire tu quale.

Questa è l'informatica!

In informatica, la pompa, gli irrigatori e i tratti di collegamento formano complessivamente un *grafo*. Un grafo è una struttura matematica costituita da oggetti detti *nodi* (o vertici) e da collegamenti tra coppie di essi, detti *archi*. Nel nostro problema, la pompa e gli irrigatori rappresentano i nodi, mentre i tratti per i tubi costituiscono gli archi. A ciascun arco è associato un valore numerico corrispondente ai metri di tubo necessari, denominato *peso* dell'arco.»

I grafi sono ampiamente utilizzati per modellare reti di ogni genere, come reti stradali, ferroviarie, elettriche o di telecomunicazioni. Poiché la posa di cavi o binari comporta dei costi elevati, si ricerca spesso un sottografo in cui tutti i nodi siano connessi tra loro utilizzando il minor numero possibile di archi e riducendo al minimo il costo complessivo (ovvero la somma dei pesi). In informatica un sottografo di questo tipo si chiama *albero di copertura minimo*. La strategia descritta sopra per trovare un albero di copertura minimo è nota come *algoritmo di Kruskal*.

Questo algoritmo appartiene alla categoria dei cosiddetti algoritmi greedy (o golosi). Durante la costruzione della soluzione, tali algoritmi compiono a ogni passo la scelta localmente ottimale (in questo caso: selezionare sempre il tratto più corto disponibile), senza valutare gli effetti a lungo termine sulle fasi successive. Nel caso dell'albero di copertura minimo, questa strategia locale garantisce di raggiungere la soluzione globalmente ottimale. Nella pratica gli algoritmi «golosi» fanno risparmiare molto denaro e molte risorse, perché trovano soluzioni in fretta e sono facili da implementare. Tuttavia trovano soluzioni ottimali solo per alcuni problemi. Prima di voler risolvere un problema in modo «goloso», bisognerebbe quindi dimostrare che fa parte di questi problemi.

Parole chiave e siti web

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

 © Castoro Informatico 2026, SSII

12. Tacos gratis!

I tacos sono un piatto messicano che si prepara in fretta. Una taquería promette: «Se per un'ordinazione di al massimo 7 tacos impieghiamo più di 15 minuti, l'ordinazione è gratis.»

La taquería ha tre cuochi, che per i diversi tipi di taco impiegano tempi diversi (in minuti):

Tipo di taco	**Ana**	**Beto**	**Carlos**
Pollo	5	3	5
Carne	3	5	5
Vegetariano	2	11	11

Quando arriva una nuova ordinazione, i tacos ordinati vengono assegnati ai cuochi uno dopo l'altro, secondo queste regole:

1. Seleziona un cuoco che finora impieghi al massimo 9 minuti. Se non vi sono cuochi che soddisfano questa condizione, procedi unicamente in base alla regola 2.
2. Scegli il cuoco che prepara il taco più velocemente. In caso di parità, scegli quello che ha bisogno di meno tempo per i tacos che ha già assegnati. Se più cuochi hanno bisogno dello stesso numero di minuti, scegli in ordine alfabetico.

Quando tutti i tacos sono stati assegnati, i cuochi iniziano a lavorare tutti insieme. Se poi un cuoco impiega più di 15 minuti, i tacos sono gratis!

Trova un'ordinazione per la quale la taquería impiega più di 15 minuti.

Utilizza il codice QR per accedere alla versione interattiva di questo esercizio.

Soluzione

Per ottenere un'ordinazione gratis, uno dei tre cuochi deve impiegare più di 15 minuti per i «suoi» tacos. Una strategia per ottenerlo potrebbe essere assegnare a un cuoco che finora lavora al massimo 9 minuti un taco per il quale impiega più di 6 minuti.

Questo è possibile solo con Beto e Carlos, con un taco vegetariano (11 minuti). Per farlo, Ana deve già impiegare più di 9 minuti per i tacos che le sono stati assegnati finora, altrimenti, secondo la regola 2, questo taco verrebbe assegnato a lei.

Impegniamo quindi Ana, ad esempio ordinando 4 volte la carne 🥩 (12 minuti). Con altre 2 porzioni di carne 🥩, Beto e Carlos riceveranno 5 minuti di lavoro ciascuno. A quel punto basterà aggiungere un ulteriore taco vegetariano 🥬: quest'ultimo verrà assegnato a Beto, che totalizzerà così 16 minuti di lavoro, facendo scattare la promozione dei 🌮 gratis!

Ci sono 11 modi diversi di ordinare all'inizio dei tacos che vengono assegnati tutti ad Ana e per i quali lei impiega più di 9 minuti (🍗 non compare, perché per questo tipo di taco Beto e Carlos sono più veloci di Ana):

- 🥩🥩🥩🥩
- 🥩🥩🥩🥬 (in 4 sequenze diverse)
- 🥩🥩🥬🥬 (in 6 sequenze diverse)

Dopodiché dobbiamo portare Beto oppure Carlos ad almeno 16 minuti con un taco vegetariano. Per farlo ci sono 4 possibilità:

- 🥩🥩🥬
- 🥬🥩🥬
- 🥬🍗🥬
- 🥩🥬🥬

Esistono dunque $11 \times 4 = 44$ combinazioni possibili per sovraccaricare la taquería e ottenere i tacos gratis. Su un totale di 2.187 ordinazioni possibili, tuttavia, la probabilità è pari a circa il 2%: non particolarmente elevata.

Questa è l'informatica!

Le regole della taquería in questo compito hanno lo scopo di preparare rapidamente le ordinazioni. Tuttavia, per alcune ordinazioni queste regole non danno un risultato ottimale. Si può allora provare, come farebbe un «attaccante», a sfruttare questa situazione per ottenere tacos gratis. La taquería potrebbe a sua volta modificare le regole per difendersi da simili azioni, ad esempio introducendo un vincolo supplementare: se l'assegnazione di un taco fa superare i 15 minuti di lavoro a un cuoco, si sceglie un cuoco differente. Basterà questo a sventare *tutti* gli attacchi? Purtroppo no! Ti vengono in mente criteri di assegnazione totalmente sicuri?

 © Castoro Informatico 2026, SSII

In informatica, la strategia di cercare per un algoritmo un input il più difficile possibile da elaborare è nota anche come *adversarial attack* (attacco avversario). Si assume il ruolo di un attaccante e si cerca di spingere un difensore a fare qualcosa che in realtà non vorrebbe fare. Gli adversarial attack hanno un ruolo particolare nei sistemi di intelligenza artificiale (IA): con dati di addestramento manipolati di proposito, si possono provocare comportamenti sbagliati nelle reti neurali artificiali.

Nei sistemi informatici, o meglio nei programmi che li controllano, possono quindi esserci regole o algoritmi vulnerabili ad attacchi mirati. Nel caso più innocuo, questo può rallentare il funzionamento o far bloccare il programma. Sarebbe peggio se degli attaccanti sfruttassero una vulnerabilità per accedere a dati senza autorizzazione o addirittura per eseguire programmi a loro scelta. Per questo motivo gli esperti di sicurezza eseguono di proposito, loro stessi, attacchi di questo tipo, per individuare possibili errori e correggerli in tempo. Questo si chiama *adversarial testing*.

Parole chiave e siti web

- Bilanciamento del carico: `https://it.wikipedia.org/wiki/Bilanciamento_del_carico`
- Adversarial attack (attacco avversario): `https://it.wikipedia.org/wiki/Adversarial_machine_learning`
- Adversarial testing: `https://www.security-insider.de/was-ist-adversarial-testing-a-4d8f9ad661ac79548140fe547cfd5152/`
- Collaudo del software / test di sicurezza: `https://it.wikipedia.org/wiki/Collaudo_del_software`, `https://en.wikipedia.org/wiki/Security_testing`



© Castoro Informatico 2026, SSII

13. Nascondino

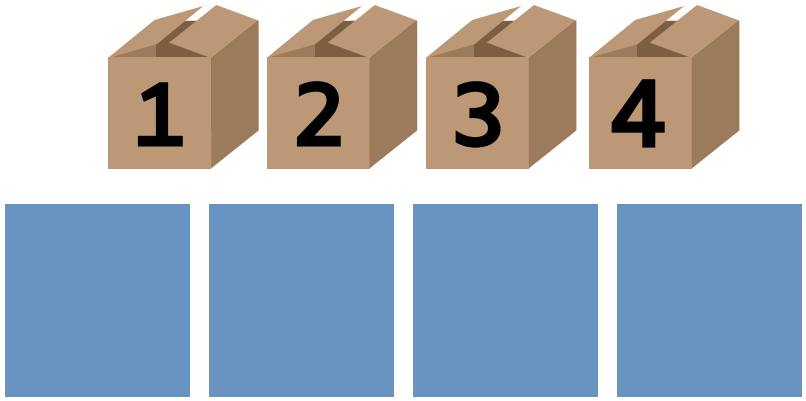
Ben si nasconde in una di quattro scatole numerate, per esempio nella scatola 2.



La sua amica Anja vuole trovarlo e a ogni turno guarda dentro una scatola. Se non lo trova, mentre Anja si sposta per controllare un'altra scatola, Ben si sposta inosservato in una scatola adiacente (a destra o a sinistra).

Anja riflette: se guarda nelle scatole nell'ordine giusto, le bastano al massimo 4 tentativi per trovare sicuramente Ben, in qualunque modo lui si sposti.

Indica un ordine di questo tipo!



Die Kisten können in die einzelnen Felder gezogen werden. Die Kisten können mehrfach verwendet werden. Man kann eine Kiste in ein anderes Feld schieben, dadurch werden die Kisten getauscht.

Soluzione

Per trovare sicuramente Ben con 4 tentativi, Anja deve guardare nelle scatole in uno di questi due ordini:

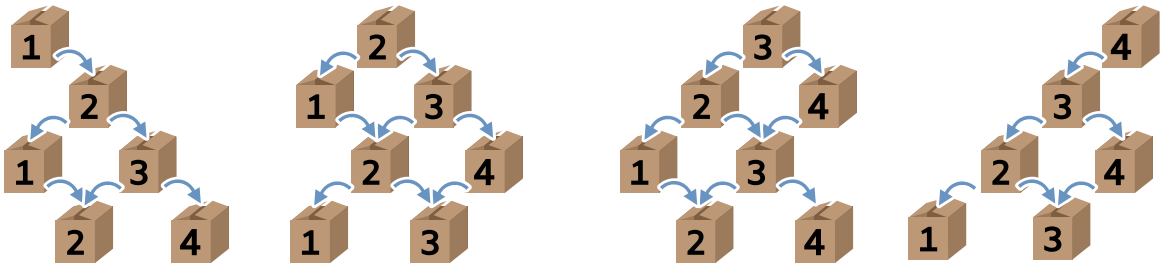


Ora mostriamo che queste risposte sono corrette. Chiamiamo *mossa* ogni spostamento di Ben in questo gioco del nascondino.

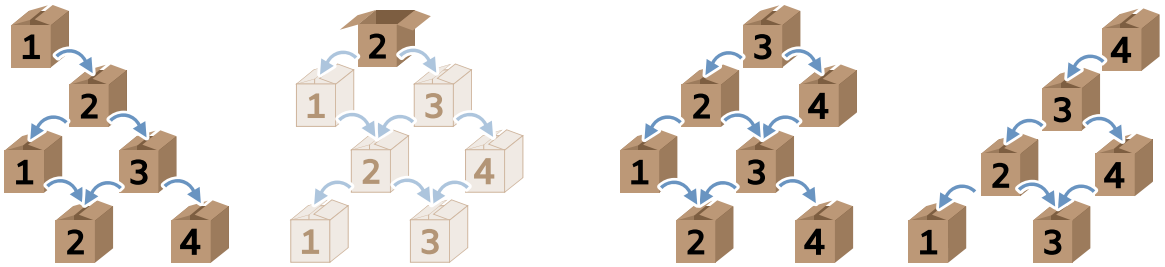
All'inizio Ben può scegliere liberamente la sua «scatola di partenza». Poi, se Ben

- si trova nella scatola 2 o 3, ha due possibilità per la mossa successiva: va a sinistra oppure a destra;
- si trova nella scatola 1 o 4, ha una sola possibilità per la mossa successiva: va a destra (dalla scatola 1) oppure a sinistra (dalla scatola 4).

Ben fa la sua mossa dopo che Anja ha guardato in una scatola senza trovarlo. Se Anja deve trovarlo con (al massimo) 4 tentativi, fino ad allora Ben può fare al massimo tre mosse. Per questo osserviamo tutte le possibili sequenze di tre mosse, partendo da ogni possibile scatola di partenza (1–4):

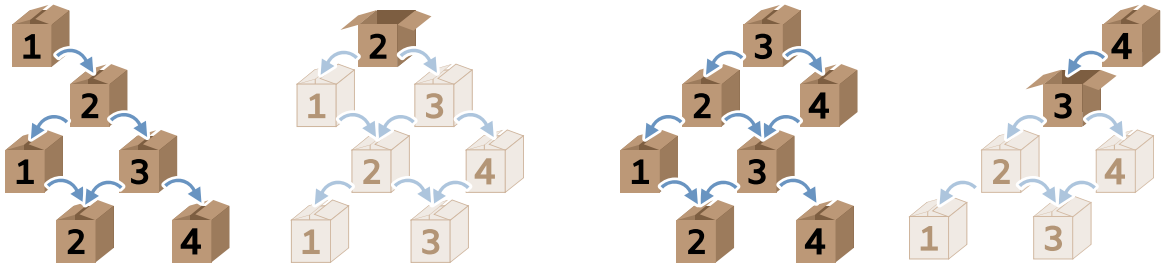


Cosa succede se Anja guarda nelle scatole nell'ordine **2-3-3-2**? Prima Anja guarda nella scatola 2. Se trova Ben lì, la scatola 2 è la sua scatola di partenza, e Ben non può più fare le mosse che partono da lì.

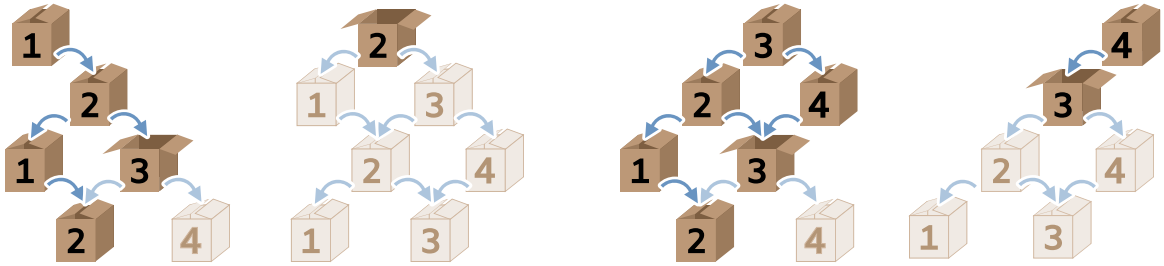


Poi Anja guarda nella scatola 3. Se trova Ben lì, questa è la sua seconda scatola. Al secondo livello delle sequenze di mosse, la scatola 3 compare ancora solo tutta a destra. Ora Ben non può più fare le mosse che partono da lì, e non dobbiamo più considerarle.

 © Castoro Informatico 2026, SSII



Come terzo tentativo, Anja guarda di nuovo nella scatola 3. Questa compare ancora in due punti al terzo livello delle sequenze di mosse. Ora Ben non può più fare le mosse che partono da lì.



Al quarto livello delle sequenze di mosse ora compare solo la scatola 2. Quindi solo la scatola 2 può essere la quarta scatola di Ben, se Anja non lo ha ancora trovato. Se ora Anja guarda nella scatola 2, Ben deve per forza essere lì.

Per la risposta **3–2–2–3** vale lo stesso ragionamento, ma a specchio: infatti le sequenze di mosse che partono dalle scatole 1 e 4 sono speculari tra loro, e lo stesso vale per quelle che partono dalle scatole 2 e 3.

Questa è l'informatica!

In informatica, le possibili sequenze di eventi che partono da uno o più punti di partenza vengono spesso rappresentate come grafi degli stati o come alberi. Si tratta di strutture dati composte da nodi e da archi, ognuno dei quali collega due nodi. Ogni nodo rappresenta un possibile stato. Ogni arco rappresenta un possibile passo da uno stato a quello successivo e per questo ha una direzione.

Queste rappresentazioni si possono usare, tra l'altro, quando ci sono due giocatori con obiettivi opposti, come in questo compito: Ben vuole nascondersi da Anja, Anja vuole trovare Ben. Con l'aiuto dei grafi delle sequenze di mosse, dal punto di vista di Anja abbiamo potuto esaminare in modo sistematico tutti i possibili comportamenti del suo «avversario» Ben e trovare così una strategia che porta Anja sicuramente al successo. Questi metodi vengono usati, tra l'altro, nella *teoria dei giochi*, nella progettazione di algoritmi e nella robotica.

Parole chiave e siti web

- Albero binario: `https://it.wikipedia.org/wiki/Albero_binario`
- Teoria_dei_giochi: `https://it.wikipedia.org/wiki/Teoria_dei_giochi`

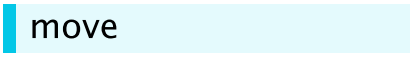
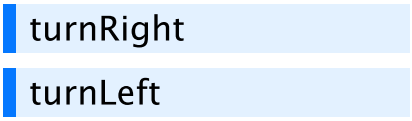
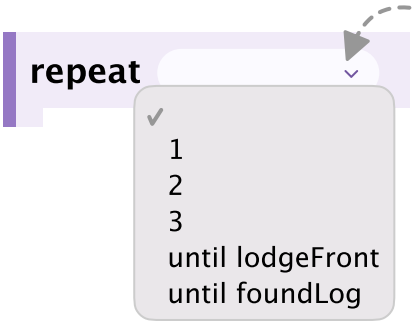


© Castoro Informatico 2026, SSII

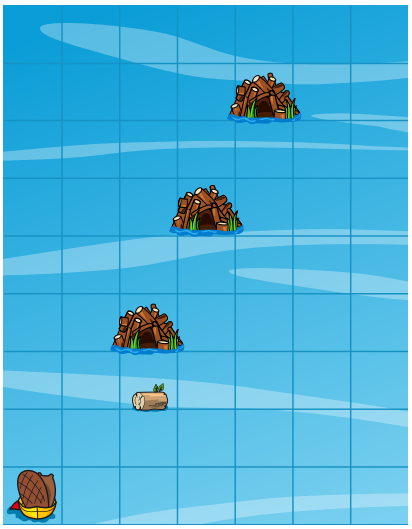
14. Un aiuto dai vicini

La castora Belinda ha urgente bisogno di tronchi. Ogni giorno un vicino diverso lascia un tronco davanti alla propria tana. Belinda vuole raccogliere il tronco ogni giorno con le stesse istruzioni di percorso. Dopo aver raccolto il tronco, non vuole più passare dai vicini successivi. Riesci ad aiutare Belinda a trovare queste istruzioni di percorso?

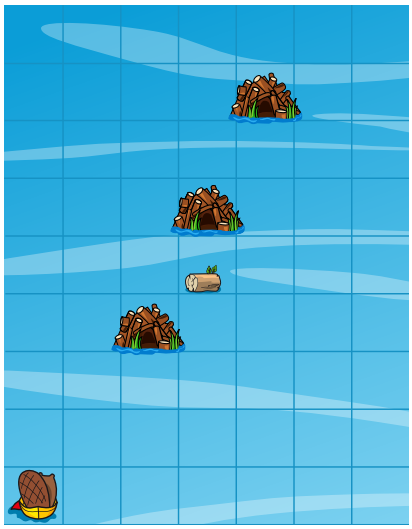
Puoi usare le seguenti istruzioni:

Istruzione	Spiegazione
	Belinda si sposta in avanti di una sola casella, nella direzione in cui guarda. Se Belinda arriva su una casella con un tronco, lo raccoglie automaticamente.
	Belinda ruota sul posto verso destra / sinistra.
	Se trascini delle istruzioni dentro il blocco, queste vengono ripetute (`repeat` = ripeti).
	Clicca sul campo chiaro e scegli quante volte ripetere le istruzioni. Con `until` puoi scegliere tra: • `lodgeFront` (finché Belinda non si trova direttamente davanti a una tana di castori), • `foundLog` (finché Belinda non ha raccolto il tronco). In tal caso, il cicolo termina dopo l'ultima istruzione contenute nel blocco `repeat`. Se non scegli niente, le istruzioni vengono ripetute *all'infinito*.

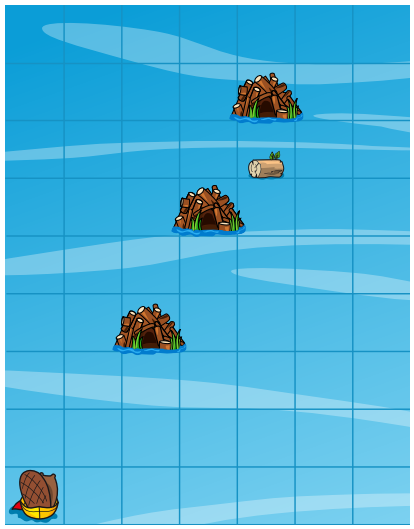
Scrivi delle istruzioni di percorso con cui Belinda raccoglie il tronco in tutti e tre i giorni. Appena ha raccolto il tronco, Belinda non deve più proseguire verso i vicini successivi.



Giorno 1



Giorno 2



Giorno 3

 © Castoro Informatico 2026, SSII

Soluzione

Una soluzione corretta è la seguente:

```
repeat until foundLog
  turnRight()
  move()
  move()
  turnLeft()
  move()
  move()
  turnLeft()
  move()
  turnRight()
```

Per arrivare al tronco in tutti i giorni, bisogna prima di tutto capire che il percorso può essere descritto con uno schema che si ripete:

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

Per arrivare il terzo giorno fino al vicino con il tronco, si scrivono le istruzioni dello schema tre volte di seguito. In questo modo Belinda raccoglie il tronco anche il primo e il secondo giorno. Le istruzioni di percorso sarebbero quindi 27 in tutto. Per accorciarle, si possono anche far ripetere le istruzioni dello schema. Per farlo, si mettono in un blocco `repeat` e si scelgono `3` ripetizioni:

```
repeat 3
  turnRight()
  move()
  move()
  turnLeft()
  move()
  move()
  turnLeft()
  move()
  turnRight()
```

Così le istruzioni di percorso non sono più 27, ma solo dieci.

In realtà abbiamo già trovato delle istruzioni di percorso per Belinda. Però la soluzione con `repeat` e un numero fisso di ripetizioni non è del tutto soddisfacente. Infatti, se il villaggio dei castori non avesse tre tane, ma per esempio cinque, dovremmo cambiare ogni volta il numero di ripetizioni nelle istruzioni di percorso.

Risolviamo il problema tenendo conto della seconda richiesta di Belinda: dopo aver raccolto un tronco, non vuole più passare dai vicini successivi. Ma è proprio quello che succederebbe se usassimo sempre `repeat` con un numero fisso di ripetizioni. Il primo giorno Belinda andrebbe comunque dal secondo e dal terzo vicino, anche se ha già raccolto il tronco dal primo vicino. Il secondo giorno succederebbe qualcosa di simile.

Per questo motivo usiamo la condizione until `foundLog` nel blocco `repeat` anziché un numero specifico:

```
repeat until foundLog
  turnRight()
  move()
  move()
  turnLeft()
  move()
  move() <-- qui Belinda raccoglie il tronco
  turnLeft()
  move()
  turnRight() <- qui raggiunge la sua nuova posizione di partenza
```

Infatti ora le istruzioni dello schema vengono ripetute solo fino a quando Belinda non ha raccolto il tronco. Prima che le istruzioni nel blocco `repeat` vengano eseguite di nuovo, la condizione `until foundLog` verifica se è già stato trovato un tronco. Se è così, le istruzioni non vengono ripetute e Belinda non prosegue verso i vicini successivi.

Attenzione: sebbene Belinda raccolga il tronco già dopo il quarto comando `move` (e quindi la condizione `foundLog` sia tecnicamente soddisfatta), i tre comandi successivi vengono comunque eseguiti tutti. Infatti, il controllo (ripetuto) tramite la condizione `foundLog` avviene sempre e solo dopo che tutti i comandi all'interno del blocco `repeat` sono stati eseguiti completamente. Perciò Belinda non si ferma subito nel punto in cui raccoglie il tronco, ma prosegue fino a raggiungere la sua successiva posizione di partenza (senza però continuare verso il vicino seguente).

```
repeat until foundLog
  turnRight()
  move()
  move()
  turnLeft()
  move()
  move() <-- qui Belinda raccoglie il tronco
  turnLeft()
  move()
  turnRight() <-- qui raggiunge la sua nuova posizione di partenza
```

Approfondimento: Abbiamo descritto il percorso con uno schema. Nel blocco `repeat` lo schema è diviso in due parti:

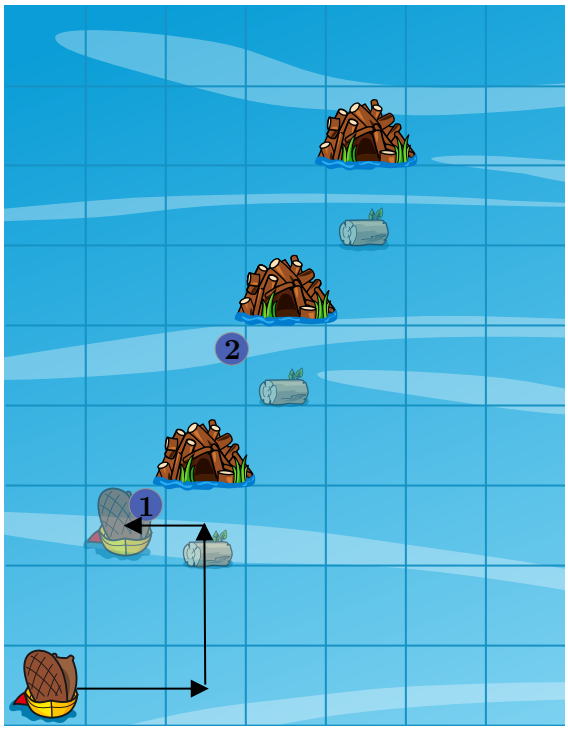
Parte 1: percorso fino al tronco

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

Parte 2: percorso fino alla nuova posizione di partenza

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

Se dividi il tuo schema in queste due parti, Belinda può seguire un percorso qualsiasi fino al tronco **e** un percorso qualsiasi fino alla posizione finale (posizione 1 per il primo tronco, posizione 2 per il secondo tronco). Belinda raccoglie comunque sempre tutti i tronchi e non prosegue mai fino al vicino successivo.



Questa è l'informatica!

Un'*invariante* (in questo caso: *invariante di ciclo*) è una proprietà che vale prima e dopo ogni ripetizione di un ciclo. Aiuta a descrivere lo stato di un sistema e a verificare che un procedimento sia

corretto. Le invarianti si usano molto spesso nei cicli, per mostrare che certe condizioni restano vere a ogni ripetizione del ciclo. Nel nostro esempio dobbiamo orientare la nostra castora in modo che la posizione di partenza (prima della ripetizione del ciclo) e la posizione finale (dopo la ripetizione del ciclo) siano identiche. La posizione di partenza/finale è definita dalla direzione in cui guarda la castora e dal punto in cui si trova rispetto alla tana di castori successiva.

Parole chiave e siti web

- Invariante di ciclo: `https://en.wikipedia.org/wiki/Loop_invariant`
- Ripetizione (ciclo): `https://it.wikipedia.org/wiki/Ciclo_(informatica)`

 © Castoro Informatico 2026, SSII

A. Autori dei quesiti

Masiar Babazadeh

Liam Baumann

Wilfried Baumann

Nils Böhnke

Emily Cavalcanti

Cesar Cepeda

Maria Cepeda

Valentina Dagienė

Christian Datzko

Heather Duan

Susanne Falkenberg-Thut

Gerald Futschek

Christian Giang

Jens Hartmann

Asterios Karagiannis

Geetanjali Khatri

Doyong Kim

Vaidotas Kinčius

Lukas Lehner

Justina Oostendorp

Özgür Özdemir

Omar Paladines

Belén Palop

Alex Parry

Jean-Philippe Pellet

Zsuzsa Pluhár

Wolfgang Pohl

Liane Raji

Lorenzo Repetto

Kirsten Schlüter

Margareta Schlüter

Dirk Schmerenbeck

Andrea Maria Schmid

Eljakim Schrijvers

Giovanni Serafini

Bronius Skūpas

Emma Sotomayor

Jacqueline Staub

Alieke Stijf

Pedro Suarez

Mario Urena

Jiří Vaníček

Michael Weigend

Kyra Willekes

B. Partner accademici

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

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

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

Pädagogische Hochschule Luzern
`https://www.phlu.ch/`

Universität Trier
`https://www.uni-trier.de/`

© Castoro Informatico 2026, SSII

C. Sponsoring

Hasler Stiftung
https://www.haslerstiftung.ch/

Amt für Kindergarten, Volksschule und Beratung, Bildungs- und Kulturdirektion, Cantone di Berna
https://www.bkd.be.ch/de/start/ueber-uns/die-organisation/amt-fuer-kindergarten-volksschule-und-beratung.html

Amt für Wirtschaft, Canton Zurigo
https://www.zh.ch/de/volkswirtschaftsdirektion/amt-fuer-wirtschaft.html

Fondazione Informatica Svizzera
https://informatics-foundation.ch

cyon
https://www.cyon.ch

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

Wealth Management IT and UBS Switzerland IT
https://www.ubs.com/

Basellandschaftliche Gebäudeversicherung
https://bgv.ch