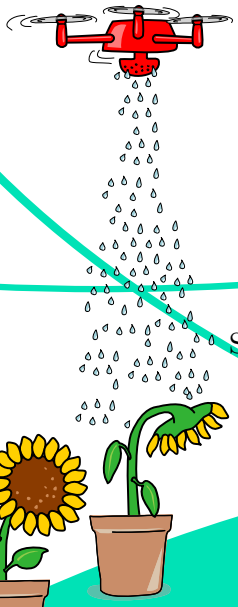


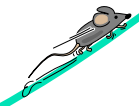


INFORMATIK-BIBER SCHWEIZ
CASTOR INFORMATIQUE SUISSE
CASTORO INFORMATICO SVIZZERA

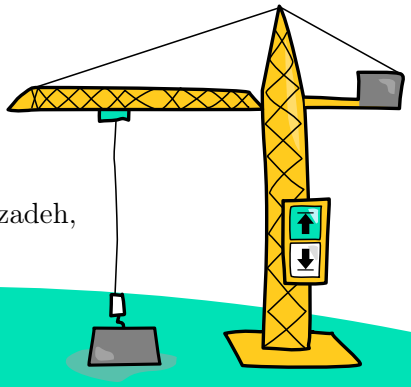
Quesiti e soluzioni 2026

5º e 6º anno scolastico



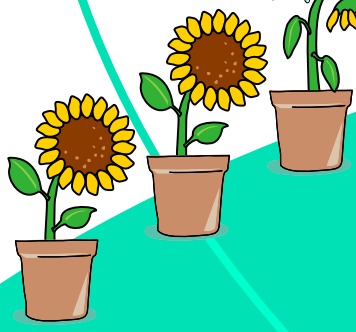


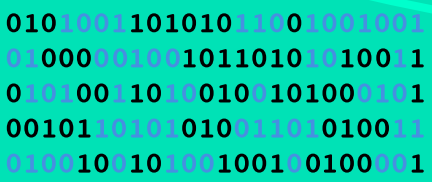
https://www.castoro-informatico.ch/



A cura di:

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

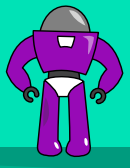




SSII

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







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

Hanno collaborato al Castoro Informatico 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 69.

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. **Giocattoli a batterie** 2026-BR-02 1
2. **Gli adesivi di Amelia** 2026-EC-01 5
3. **Video consigliati** 2026-HU-01a 9
4. **Pile di laptop** 2026-CH-03a 13
5. **Passeggiata nel parco** 2026-DE-01 17
6. **Il robot di sorveglianza** 2026-DE-07 21
7. **La fabbrica di sciroppo** 2026-KR-04 25
8. **Compiti a casa** 2026-SI-01a 29
9. **L'app dei pesci** 2026-DE-08 33
10. **Testa o croce** 2026-IN-05 37
11. **Colora il quadrato** 2026-DE-09 39
12. **La fabbrica di auto** 2026-JP-03 43
13. **Il folletto di fuoco** 2026-CZ-02 47
14. **Il metodo di Muddy** 2026-DE-10 51
15. **Tacos gratis!** 2026-MX-02b 55
16. **Nascondino** 2026-US-03b 59
17. **Un aiuto dai vicini** 2026-DE-90 63

A. **Autori dei quesiti** 69

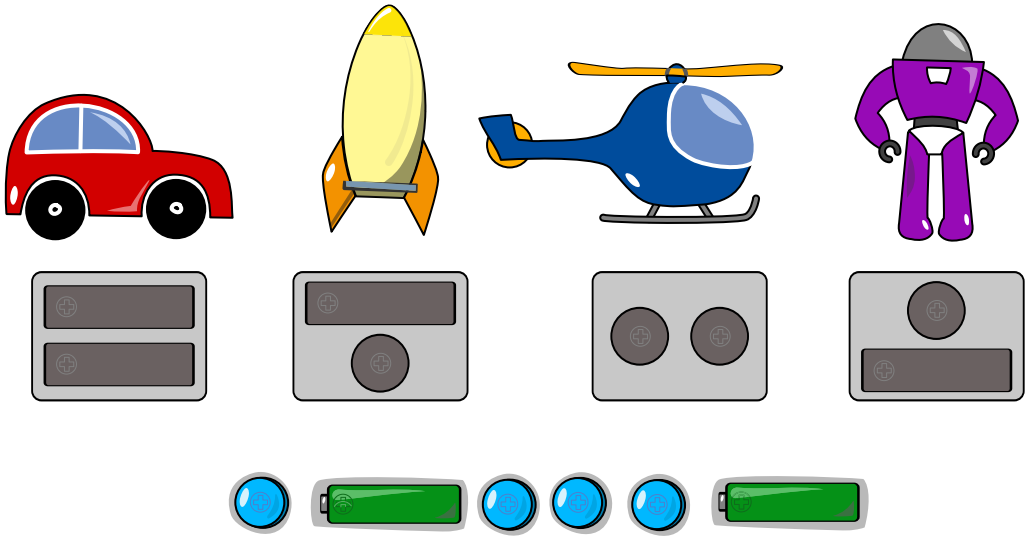
B. **Partner accademici** 71

C. **Sponsoring** 72

© Castoro Informatico 2026, SSII

1. Giocattoli a batterie

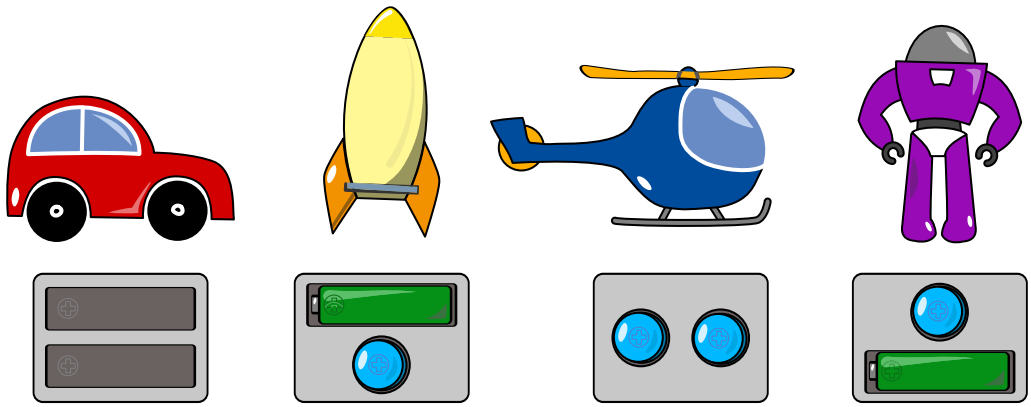
I 4 giocattoli di Bona hanno bisogno di batterie.
I giocattoli funzionano solo quando tutti gli scomparti batterie sono pieni.
Bona vuole che funzionino più giocattoli possibile.



Come bisogna mettere le batterie negli scomparti?

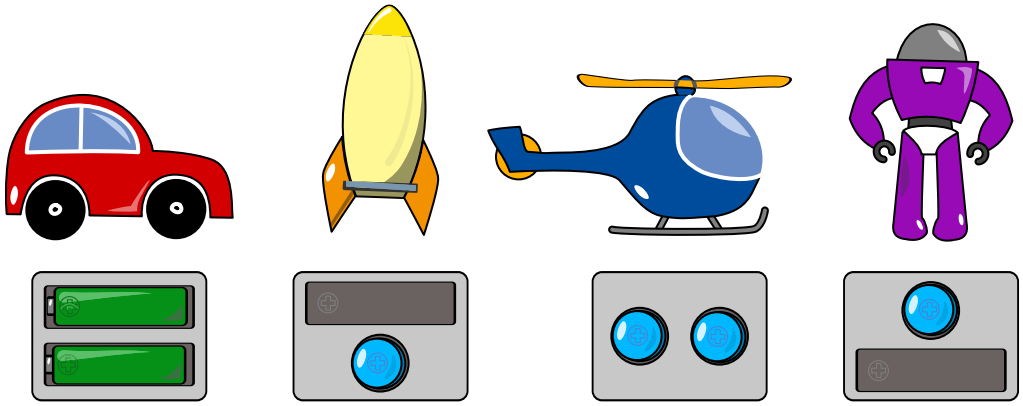
Soluzione

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

2. Gli adesivi di Amelia

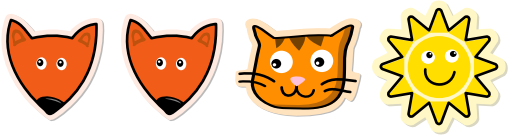
Ad Amelia piace mettere gli adesivi in fila seguendo sempre queste regole:

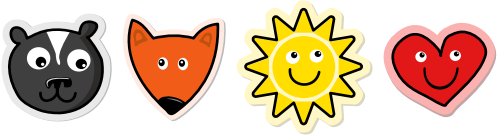
All'inizio posiziona due o tre adesivi con animali. Dopodiché aggiunge altri adesivi in base a queste condizioni:

- 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) 

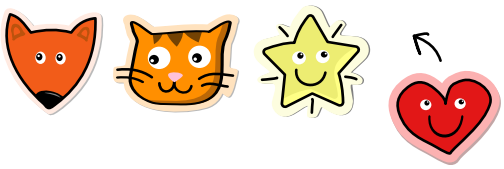
D) 

Soluzione

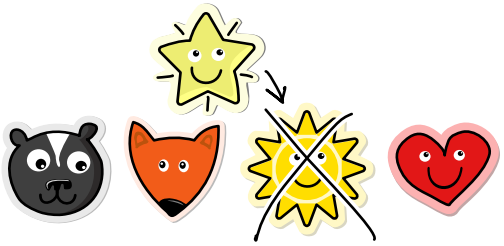
La risposta corretta è B.

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

3. Video consigliati

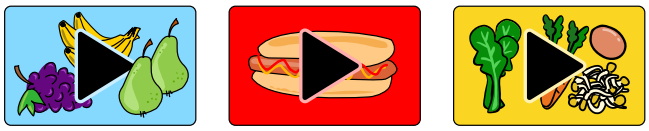
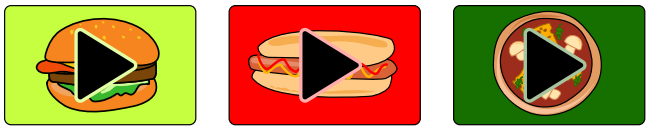
A Zara piace guardare i video su ZickZock.
ZickZock le propone sempre nuovi video. Funziona così:

- 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.

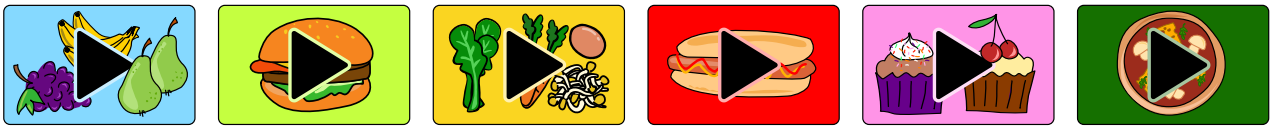
Zara ha visto questi video:



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:



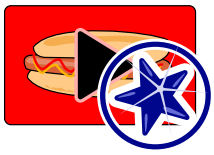
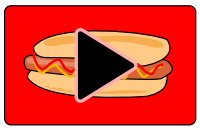
Cerca nella tabella le persone che hanno visto almeno due video uguali a quelli visti da Zara.



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

4. Pile di laptop

Quattro bambini frequentano un corso di informatica. Ogni bambino usa sempre lo stesso laptop.

Dopo la lezione mettono i loro laptop in 2 contenitori. Li impilano in modo che il giorno dopo ognuno possa prendere il proprio laptop direttamente dall'alto, senza dover spostare gli altri laptop.

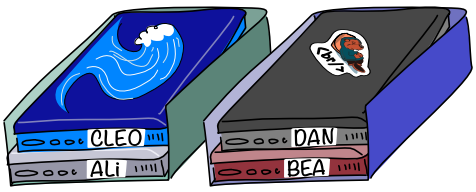
Oggi, dopo la lezione, escono dall'aula in quest'ordine:

1. Ali
2. Bea
3. Cleo
4. Dan

Guardando l'orario delle lezioni, sanno già che domani arriveranno al corso di informatica in quest'ordine:

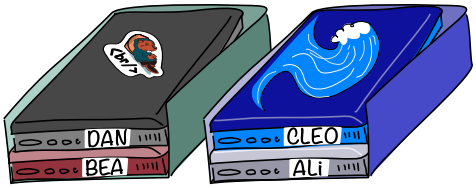
1. Bea
2. Ali
3. Dan
4. Cleo

In che modo i bambini devono disporre in pila i laptop oggi, uscendo dall'aula?

A) 

B) 

C) 

D) 

Soluzione

La risposta corretta è B.

Verifichiamo se ciascuna delle quattro opzioni di risposta è compatibile con l'ordine di oggi e con quello di domani.

Risposta B: Domani, quando vorranno riprenderli, Bea e Ali li ritireranno uno dopo l'altro dal contenitore di sinistra, mentre Dan e Cleo faranno lo stesso da quello di destra.

Quando domani i bambini vogliono prendere i loro laptop, Bea e Ali prendono i loro laptop uno dopo l'altro dal contenitore di sinistra, e Dan e Cleo prendono i loro laptop uno dopo l'altro dal contenitore di destra. La risposta B è quindi corretta.

Risposta A: Oggi i bambini possono impilare i laptop come nella risposta A: Ali mette il suo laptop nel contenitore di sinistra, Bea il suo in quella di destra, poi Cleo il suo in quello di sinistra e Dan il suo in quello di destra.

Ma domani Bea arriva per prima: per prendere il suo laptop deve prima togliere il laptop di Dan dal contenitore di destra. La risposta A è quindi sbagliata.

Risposta C: Oggi i bambini *non* possono impilare i laptop come nella risposta C: Ali mette per primo il suo laptop in uno dei contenitori, quindi il suo laptop può stare solo in fondo al contenitore; qui invece il laptop di Ali si trova in cima al contenitore di destra. La risposta C è quindi sbagliata.

Risposta D: Oggi i bambini possono impilare i laptop come nella risposta D: Ali mette il suo laptop nel contenitore di destra, Bea il suo in quello di sinistra, poi Cleo il suo in quello di destra e Dan il suo in quello di sinistra. Sono disposti come nella risposta A, ma con i contenitori invertiti.

Anche in questo caso, domani Bea non può prendere per prima il suo laptop senza togliere prima il laptop di Dan dal contenitore, che questa volta è quello di sinistra. Anche la risposta D è quindi sbagliata.

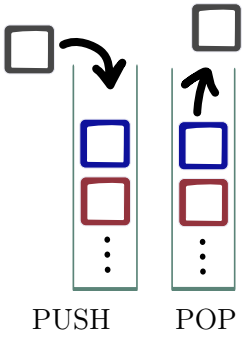
Questa è l'informatica!

Dai contenitori si può prelevare sempre e solo il laptop che si trova in cima. Il computer inserito per ultimo nel contenitore deve essere rimosso per primo. I due laptop in un contenitore formano infatti una pila.

Anche in informatica esiste una struttura dati analoga, la *pila* (spesso indicata con il termine inglese «stack»), utile per memorizzare temporaneamente dei dati durante l'esecuzione di un calcolo.

Quando utilizza una pila, un programma può eseguire due operazioni: inserire un elemento in cima (push) oppure rimuovere l'elemento che si trova attualmente in cima (pop). In questo modo uno stack realizza il principio di memorizzazione *LIFO* (abbreviazione dell'inglese: last in, first out).

 © Castoro Informatico 2026, SSII



Se i bambini avessero un solo contenitore, i laptop potrebbero formare una sola pila. Allora, secondo il principio LIFO, all'inizio della lezione di informatica i bambini dovrebbero prendere i laptop esattamente nell'ordine inverso rispetto a quello in cui li hanno messi nel contenitore alla fine della lezione precedente. Con due contenitori i bambini hanno più possibilità di impilare e riprendere i laptop. Una seconda pila aumenta quindi la flessibilità: in questo modo, grazie allo schema scelto dai bambini, domani i computer potranno essere recuperati in un ordine diverso rispetto al semplice inverso di quello odierno.

Parole chiave e siti web

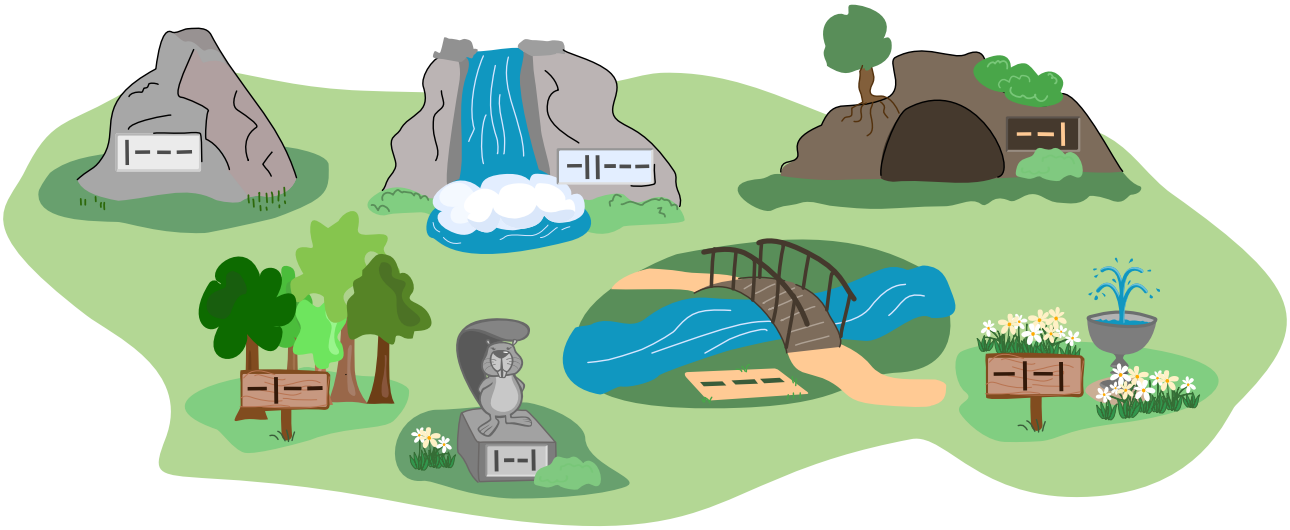
- *Pila (stack, LIFO)*: `https://it.wikipedia.org/wiki/Pila_(informatica)`



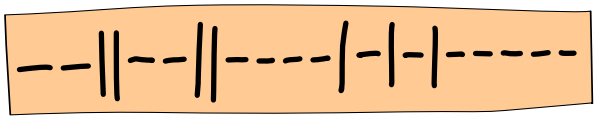
© Castoro Informatico 2026, SSII

5. Passeggiata nel parco

Nel parco dei castori ci sono diversi luoghi speciali. Ognuno di questi luoghi ha un contrassegno, fatto con i simboli - e |.



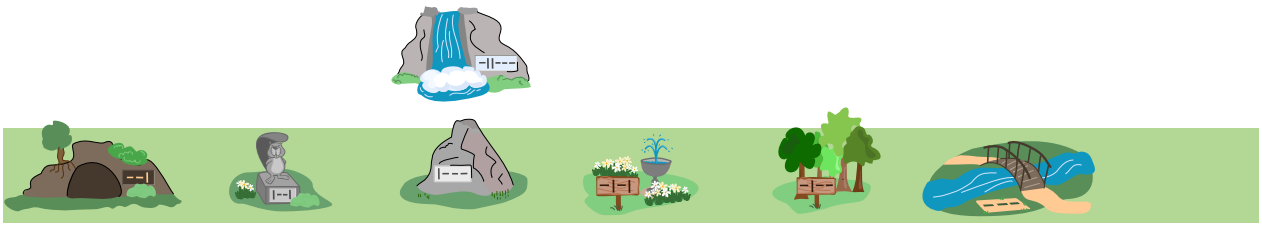
Simon fa una passeggiata nel parco dei castori. Ogni volta che passa davanti a un luogo speciale, annota il relativo contrassegno su un foglio, uno dopo l'altro. Alla fine ha scritto questi simboli:



Davanti a quali luoghi è passato Simon, e in quale ordine?

Soluzione

La risposta corretta è la seguente:



Questo è ciò che Simon ha annotato durante la sua passeggiata:

`--||--||----|-|-|-----`

I contrassegni dei luoghi non hanno tutti lo stesso numero di simboli. Per questo motivo, all'inizio non sappiamo esattamente dove termini un contrassegno e ne inizi uno nuovo nella nota di Simon.

Per trovare la risposta corretta, leggiamo la nota da sinistra a destra, cioè nell'ordine in cui Simon ha scritto i contrassegni. Verifichiamo intanto se uno o più simboli consecutivi corrispondono all'inizio di un contrassegno.

Diversi contrassegni iniziano con il primo simbolo `-`. Quindi non possiamo ancora dire quale contrassegno Simon ha annotato per primo e, di conseguenza, davanti a quale luogo è passato per primo.

Con i primi due simboli `--` iniziano esattamente due contrassegni: quello della grotta `--|` e quello del ponte `---`. Anche a questo punto, quindi, non sappiamo ancora quale contrassegno ha annotato per primo e, di conseguenza, davanti a quale luogo è passato per primo.

Con i primi tre simboli `--|` inizia solo il contrassegno della grotta. Ora sappiamo che Simon è passato per primo davanti alla grotta e ne ha annotato il contrassegno. Sappiamo anche che a questo punto inizia un nuovo contrassegno.

Il simbolo successivo nella nota di Simon è `|`. Iniziano così solo i contrassegni della statua `|--|` e della roccia `|---`, lunghi entrambi quattro simboli. I quattro simboli successivi della nota di Simon `|--|` formano esattamente il contrassegno della statua. La statua è quindi il secondo luogo davanti a cui è passato Simon.

Esaminiamo quindi gli altri simboli passo dopo passo. In questo modo scopriamo che Simon è passato davanti a questi luoghi speciali nel seguente ordine:

1. Grotta `--|`
2. Statua `|--|`
3. Roccia `|---`
4. Fontana `-|-|`
5. Gruppo di alberi `-|--`
6. Ponte `---`

 © Castoro Informatico 2026, SSII

La nota di Simon contiene anche il contrassegno della cascata `-||---`. Tuttavia, non esiste né un singolo contrassegno né una sequenza di contrassegni consecutivi che corrisponda alla prima parte della nota, ovvero `--||-`. Simon quindi non è passato davanti alla cascata.

Questa è l'informatica!

Hai appena decodificato una sequenza di parole di codice (i contrassegni dei luoghi speciali) a partire da una codifica continua, senza separatori tra le parole di codice.

In questo esercizio hai scomposto un codice (la nota di Simon) in una sequenza di parole in codice, riuscendo così a decodificarlo e a ricavare l'ordine dei luoghi.

Questa scomposizione (e quindi la decodifica) è stata possibile soprattutto perché il codice utilizzato in questo compito è *privo di prefissi*: nessuna parola di codice è il prefisso di un'altra, ovvero nessuna parola di codice inizia con un'altra.

I *codici privi di prefissi* (noti anche come codici prefisso) rivestono un ruolo fondamentale in informatica, in particolare nella compressione dei dati. La loro proprietà fondamentale garantisce che la decodifica sia univoca: se una sequenza di simboli di codice (che spesso nei sistemi informatici sono bit) viene letta in ordine, i confini tra le parole sono netti. In qualsiasi punto della sequenza risulta chiaro dove termini una parola e dove inizi la successiva, senza necessità di separatori o di anticipare la lettura.

Un esempio noto di come si creano codici privi di prefissi è la *codifica di Huffman*, sviluppata nel 1952 da David A. Huffman. Questo metodo di compressione estremamente diffuso assegna parole di codice più brevi ai caratteri più frequenti e parole più lunghe a quelli più rari.

Le codifiche di Huffman sono costruite in modo da essere prive di prefissi. In questo modo si garantisce che i dati compressi possano essere decodificati in modo affidabile ed efficiente.

Parole chiave e siti web

- *Codici privi di prefissi (codici prefissi)*: `https://it.wikipedia.org/wiki/Codice_prefisso`
- *Compressione dei dati*: `https://it.wikipedia.org/wiki/Compressione_dei_dati`
- *Codifica di Huffman*: `https://it.wikipedia.org/wiki/Codifica_di_Huffman`

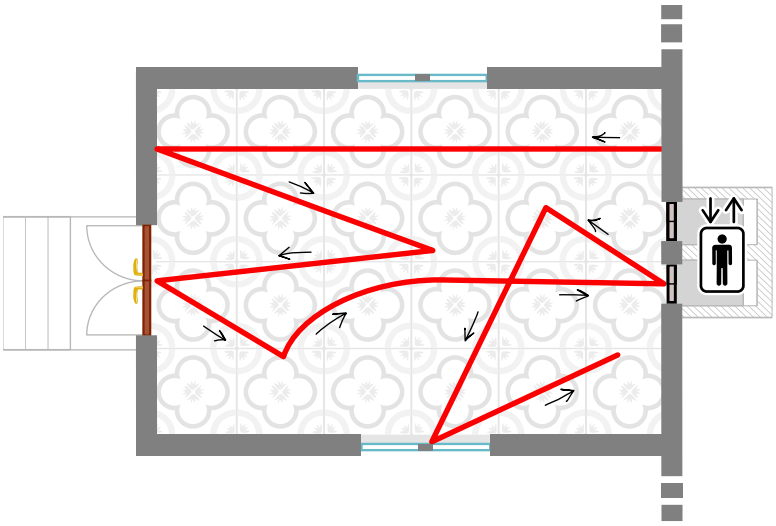


© Castoro Informatico 2026, SSII

6. Il robot di sorveglianza

Un robot di sorveglianza si muove in un atrio vuoto.
Se non succede niente di particolare, procede sempre dritto.
Ci sono però tre eccezioni:

1. Se il robot tocca un ostacolo, cambia direzione a caso.
2. Se rileva un movimento, cambia direzione e si dirige verso il punto in cui avviene il movimento.
3. Se sente un rumore, cambia direzione e si dirige verso il punto da cui proviene il rumore.



Questa è la pianta dell'atrio. Le pareti, la porta a sinistra, le finestre (in alto e in basso) e gli ascensori (a destra) sono ostacoli per il robot.
La linea rossa mostra il percorso che il robot ha fatto oggi.

Durante il giro di oggi sono successi tre eventi:



Una palla colpisce una finestra.

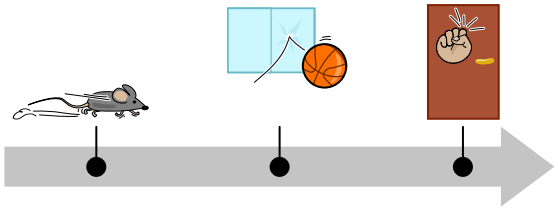


Qualcuno bussa alla porta.



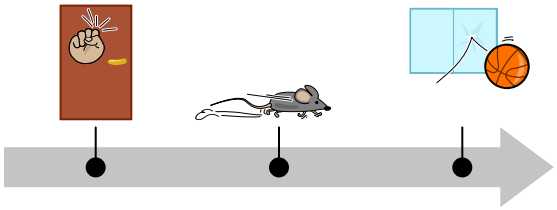
Un topo corre attraverso l'atrio.

In quale ordine sono successi gli eventi?

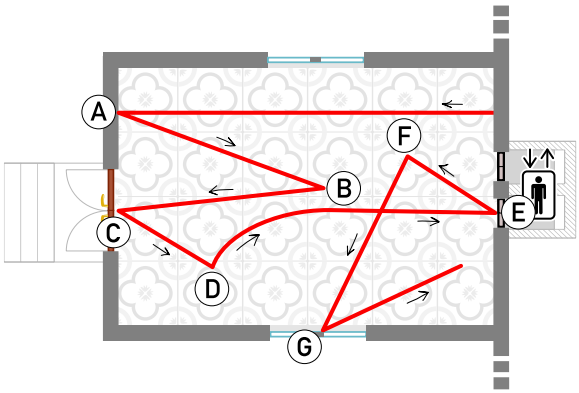


Soluzione

La risposta corretta è la seguente:



Osserviamo il percorso del robot:



Nelle posizioni A, C, E e G il robot tocca degli ostacoli e, ogni volta, cambia direzione a caso (regola 1).

Anche nelle posizioni B, D e F il robot cambia direzione, ma lì non ci sono ostacoli. Gli eventi devono quindi essere successi proprio in queste posizioni.

Nella posizione D si nota che, da lì in poi, il robot percorre una curva, cioè continua a cambiare direzione. Questo si può spiegare solo con l'evento «Un topo corre attraverso l'atrio.» Il topo corre da sinistra verso destra: per questo il robot rileva continuamente un nuovo movimento e cambia direzione ogni volta verso destra, cioè verso il movimento (regola 2). In questo modo cerca di seguire il topo, finché non riesce più a rilevarne il movimento.

Nella posizione B il robot cambia direzione verso la porta. Poiché in seguito procede dritto, in questo caso non sta seguendo un movimento. Qui il robot ha quindi sentito un rumore alla porta (regola 3): «Qualcuno bussa alla porta».

Nella posizione F il robot cambia direzione verso una finestra e poi prosegue dritto. Anche qui ha sentito un rumore (regola 3), questa volta alla finestra: «Una palla colpisce una finestra».

I tre eventi sono quindi successi nell'ordine indicato sopra.

Questa è l'informatica!

I *robot di sorveglianza* vengono impiegati, per esempio, in edifici pubblici, magazzini e complessi di uffici. Di solito questi robot si muovono in aree chiuse e prestabilite, riconoscono eventi particolari

e agiscono secondo regole programmate. Anche se i robot di sorveglianza moderni possono usare l'apprendimento automatico o sistemi di percezione avanzati, il loro comportamento di base continua a essere determinato da algoritmi che trasformano i dati dei sensori in azioni.

Nel caso dei *sistemi autonomi*, come i robot di sorveglianza o i veicoli, l'informatica distingue diversi livelli di autonomia. I sistemi autonomi più semplici, come il robot di questo compito, seguono delle regole, per esempio «Se rilevi un ostacolo, gira» oppure «Se senti un rumore, muoviti in quella direzione.» Il comportamento dei sistemi di questo livello è prevedibile e si può ricondurre direttamente alle regole programmate.

I sistemi autonomi più avanzati, invece, devono potersi muovere liberamente nel mondo. Il mondo reale, tuttavia, è complesso e possono succedere eventi difficilmente prevedibili. Un veicolo autonomo, per esempio, deve saper decidere da solo e in modo sensato come guidare in qualsiasi situazione. È però impossibile stabilire una regola specifica per ciascuna delle situazioni che un veicolo potrebbe incontrare sulla strada. Per questo i sistemi con livelli di autonomia più alti non usano regole fisse: hanno invece degli obiettivi e decidono in modo dinamico come agire per raggiungerli. Per farlo possono usare i dati dei sensori, metodi di pianificazione e l'apprendimento automatico. Questi sistemi vengono chiamati anche *agentici*, poiché pianificano e stabiliscono autonomamente le proprie azioni per raggiungere i loro obiettivi.

Parole chiave e siti web

- Robotica: `https://it.wikipedia.org/wiki/Robotica`
- Sensore: `https://it.wikipedia.org/wiki/Sensore`
- Attuatore: `https://it.wikipedia.org/wiki/Attuatore`
- Robot di sorveglianza: `https://wirtschaftslexikon.gabler.de/definition/sicherheitsroboter-122267`
- Veicoli autonomi: `https://it.wikipedia.org/wiki/Autovettura_autonoma`
- Open Roberta Lab: `https://lab.open-roberta.org`
- Kiddle - Robot Facts for Kids: `https://kids.kiddle.co/Robot`



© Castoro Informatico 2026, SSII

7. La fabbrica di sciroppo

Una fabbrica produce sciroppo. Alcuni dei lavori vengono svolti da robot.

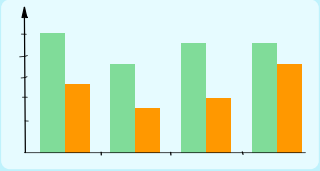
Uno schermo mostra due tabelle con le informazioni sui sei robot:

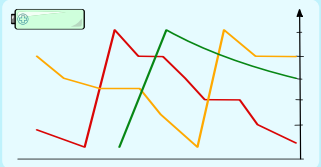
- a sinistra sono indicati il produttore e la posizione attuale di ciascun robot,
- mentre a destra compaiono la funzione e il livello della batteria.

Entrambe le tabelle includono i numeri identificativi dei robot.

Produttore	N. robot	Posizione
Robras	R-01	Magazzino
Robras	R-03	Spedizione
Robras	R-05	Lavaggio bottiglie
Bibobot	R-02	Imbottigliamento
Bibobot	R-04	Cucina dello sciroppo
Bibobot	R-06	Cucina dello sciroppo

N. robot	Funzione	Batteria
R-01	Trasporto	85 %
R-02	Chiusura	25 %
R-03	Trasporto	48 %
R-04	Miscelazione	90 %
R-05	Risciacquo	75 %
R-06	Mescolamento	55 %









Quando il livello della batteria di un robot è sotto il 30 %, il robot deve andare alla stazione di ricarica.

Un robot deve andare alla stazione di ricarica. Dove si trova?

A) Magazzino
B) Imbottigliamento
C) Spedizione
D) Cucina dello sciroppo

Soluzione

La risposta corretta è B.

Produttore	N. robot	Posizione
Robras	R-01	Magazzino
Robras	R-03	Spedizione
Robras	R-05	Lavaggio bottiglie
Bibobot	R-02	Imbottigliamento
Bibobot	R-04	Cucina dello sciroppo
Bibobot	R-06	Cucina dello sciroppo

N. robot	Funzione	Batteria
R-01	Trasporto	85 %
R-02	Chiusura	25 %
R-03	Trasporto	48 %
R-04	Miscelazione	90 %
R-05	Risciacquo	75 %
R-06	Mescolamento	55 %

A destra si vede che il robot R-02 ha la batteria solo al 25 %. Siccome è sotto il 30 %, il robot R-02 deve andare alla stazione di ricarica. Come si vede a sinistra, il robot si trova nel reparto Imbottigliamento.

	Produttore	N. robot	Posizione
a)	Robras	R-01	Magazzino
b)	Robras	R-03	Spedizione
	Robras	R-05	Lavaggio bottiglie
	Bibobot	R-02	Imbottigliamento
c)	Bibobot	R-04	Cucina dello sciroppo
c)	Bibobot	R-06	Cucina dello sciroppo

N. robot	Funzione	Batteria
R-01	Trasporto	85 %
R-02	Chiusura	25 %
R-03	Trasporto	48 %
R-04	Miscelazione	90 %
R-05	Risciacquo	75 %
R-06	Mescolamento	55 %

La risposta A non è corretta: nel Magazzino si trova soltanto il robot R-01, che non ha bisogno di ricarica poiché la sua batteria è all'85% (quindi superiore al 30%).

La risposta C non è corretta: nel reparto Spedizione c'è solo il robot R-03, la cui batteria è al 48% (quindi sopra la soglia del 30%).

La risposta D non è corretta: nella Cucina dello sciroppo operano i robot R-04 e R-06, entrambi con livelli di batteria ampiamente superiori al 30% (rispettivamente al 90% e al 55%).

Questa è l'informatica!

Le tabelle di questo compito mostrano dei dati sui robot della fabbrica di sciroppo. I record (cioè le righe) della tabella a sinistra contengono il produttore e la posizione del robot, quelli della tabella a destra contengono la funzione e il livello della batteria. Siccome entrambe le tabelle contengono anche il numero del robot, si possono collegare i dati delle due tabelle e trovare così la posizione del robot che deve andare alla stazione di ricarica.

In informatica si parla di *banca dati* (o *database*) quando le informazioni relative a determinati oggetti (come i robot di una fabbrica, i clienti di un'azienda o gli articoli di un e-commerce) vengono archiviate in più tabelle e messe in relazione tra loro tramite delle *chiavi*. All'interno di una tabella, una chiave deve essere univoca, ossia comparire in una sola riga; per questo motivo si ricorre solitamente a un

codice identificativo (come un numero cliente o un codice articolo). Nel nostro problema, il numero del robot funge proprio da chiave. Sfruttando le chiavi, le banche dati permettono di combinare più tabelle in un'unica tabella virtuale tramite l'operazione di *join*, aggregando così tutte le informazioni correlate.

Questo processo di aggregazione di dati provenienti da fonti differenti prende il nome di integrazione delle informazioni. Tale operazione è comunemente utilizzata, ad esempio, nei negozi online per verificare le scorte di magazzino di un articolo, oppure nella ricerca scientifica per unire le misurazioni provenienti da strumenti diversi. In questo esercizio hai dovuto svolgere esattamente questo compito, unendo i dati di due tabelle distinte.

Parole chiave e siti web

- Banca dati: `https://it.wikipedia.org/wiki/Base_di_dati`
- Integrazione delle informazioni: `https://it.wikipedia.org/wiki/Data_integration`



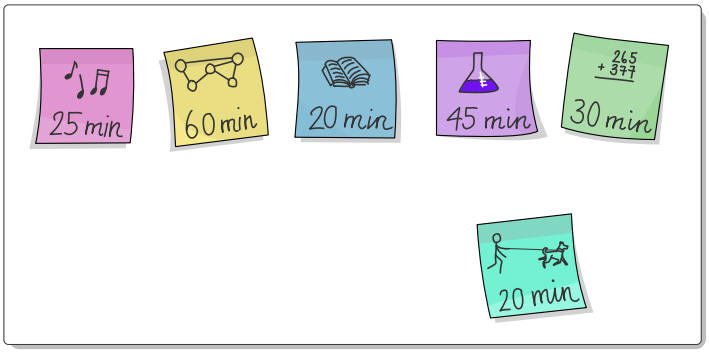
© Castoro Informatico 2026, SSII

8. Compiti a casa

Sarah svolge i compiti in modo sistematico, uno dopo l'altro. Come compito successivo sceglie sempre quello che richiede meno tempo. Lo fa all'inizio e ogni volta che ne finisce uno.

Oggi Sarah ha cinque compiti. Prima annota ogni compito e quanto tempo le serve per farlo. Poi comincia.

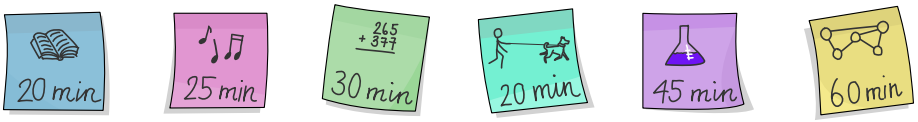
Dopo aver lavorato per 50 minuti, si ricorda che deve anche portare a spasso il cane per 20 minuti. Annota anche questo come compito:



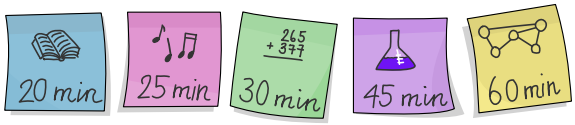
In quale ordine Sarah svolge i compiti?

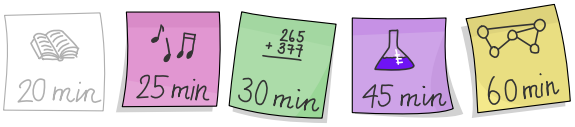
Soluzione

La risposta corretta è la seguente:



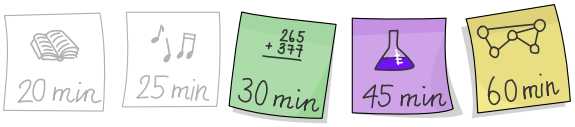
Sarah seleziona sempre il compito che richiede meno tempo. Per individuarlo facilmente, disponiamo i compiti in una lista ordinata in base alla durata, in modo che la prossima attività da svolgere si trovi sempre in prima posizione.





Per primo sceglie inglese 

, il compito più veloce di tutti.



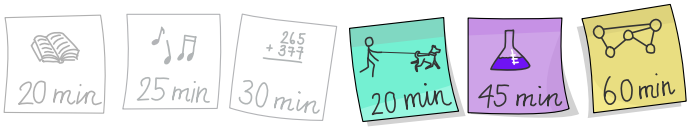
Poi sceglie il compito di musica 

. Quando lo ha finito, ha lavorato in tutto 45 minuti.

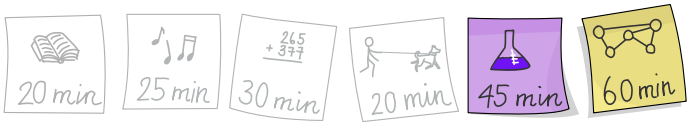


Sceglie il compito successivo: matematica 

.

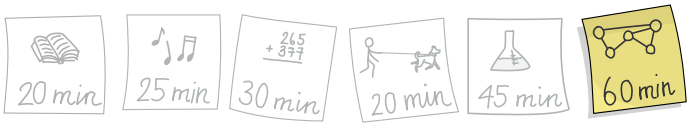


Dopo 5 minuti di calcoli concentrati, si ricorda della passeggiata di 20 minuti con il cane. Questo compito viene aggiunto alla lista rimanente e messo al primo posto, in base al tempo che richiede.



Ora Sarah porta a spasso il cane 

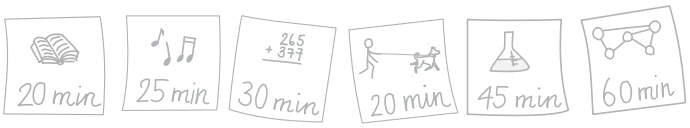
.



Poi sceglie il compito successivo della lista, ovvero chimica 

.

 © Castoro Informatico 2026, SSII



Infine Sarah sceglie il compito più lungo di tutti: informatica 

. Ora la sua lista dei compiti è finalmente vuota.

Questa è l'informatica!

Tra i compiti ancora da svolgere, Sarah sceglie sempre quello che richiede meno tempo. Questa strategia è nota in informatica con il nome di *Shortest Job Next* («prima il lavoro più breve»)...

Si tratta di una regola di pianificazione che, per esempio, i sistemi operativi dei computer usano per decidere quale compito la CPU del computer deve eseguire come prossimo. Il sistema operativo sceglie, dalla lista dei compiti non ancora conclusi, quello che probabilmente richiede il tempo di esecuzione più breve. Affinché questa scelta non faccia perdere tempo inutilmente, di solito la lista viene gestita come una *coda di priorità*, dalla quale si può scegliere il compito successivo in modo efficiente. I compiti possono aggiungersi in momenti diversi ed essere inseriti nella coda, come la passeggiata con il cane in questo compito del Castoro. In ogni caso, il nuovo compito può essere scelto solo quando il compito in corso è terminato.

Il vantaggio di Shortest Job Next è che i compiti brevi vengono svolti rapidamente, senza lunghe attese. Ai compiti più lunghi però può capitare di dover aspettare molto a lungo (in informatica si dice anche che «muoiono di fame», in inglese *starvation*), se continuano ad aggiungersi nuovi compiti brevi. Tuttavia, in questo problema del Castoro nessuno rischia di rimanere «affamato», né Sarah né il suo cane: per fortuna i compiti non sono così tanti e se ne aggiunge uno solo.

Parole chiave e siti web

- Shortest Job Next (anche Shortest Job First): `https://it.wikipedia.org/wiki/Shortest_job_first`
- Coda di priorità: `https://it.wikipedia.org/wiki/Coda_di_priorità`



© Castoro Informatico 2026, SSII

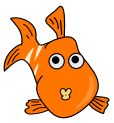
9. L'app dei pesci

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:

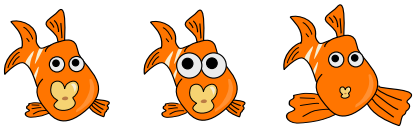


Se ora scegliamo nell'app uno smiley che presenta alcuni elementi grandi, una determinata parte del pesce si ingrandirà di conseguenza. Ad esempio:

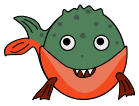
Smiley



Pesce



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



Nell'app scegli questo smiley:

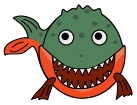


Come diventa allora l'immagine del pesce?

A) 

B) 

C) 

D) 

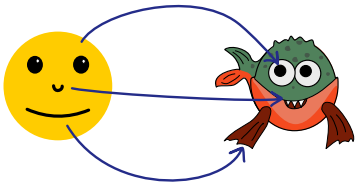
Soluzione

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`

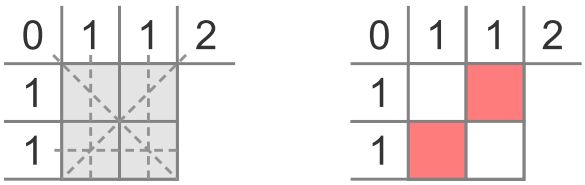
 © Castoro Informatico 2026, SSII

11. Colora il quadrato

In una griglia quadrata, le caselle devono essere colorate di rosso o di bianco. All'inizio tutte le caselle sono grigie.

I numeri sui bordi indicano quante caselle devono essere **rosse** in ogni riga, in ogni colonna e in ognuna delle due diagonali. Tutte le altre caselle devono diventare bianche.

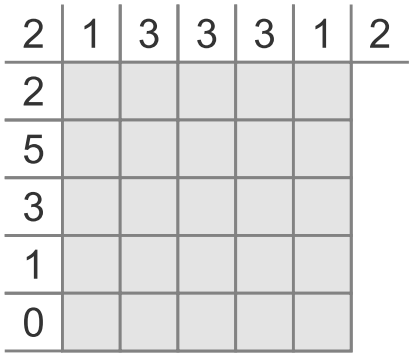
Ecco un piccolo esempio con un quadrato 2×2:



In ogni riga e in ogni colonna deve esserci esattamente una casella rossa. Sulla diagonale che va dall'angolo in alto a sinistra a quello in basso a destra non sono permesse caselle rosse, sull'altra diagonale devono esserci esattamente 2 caselle rosse.

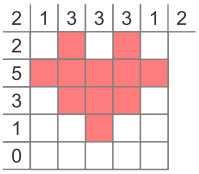
Questa colorazione è l'unica soluzione per questo quadrato.

Qual è la soluzione per questo quadrato 5×5?

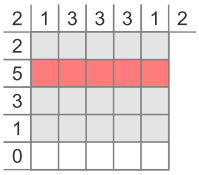


Soluzione

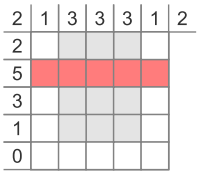
La risposta corretta è la seguente:



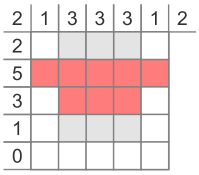
Non c'è nessun altro modo di colorare le caselle rispettando le indicazioni. Lo mostra il procedimento descritto qui sotto per trovare questa soluzione:



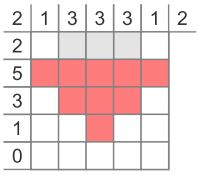
Passo 1: La riga con l'indicazione **5** deve essere colorata tutta di rosso, quella con l'indicazione **0** tutta di bianco.



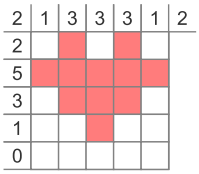
Passo 2: Le caselle ancora grigie nella prima e nella quinta colonna devono essere bianche: in queste colonne c'è infatti già una casella rossa.



Passo 3: Le altre caselle grigie della riga centrale devono essere colorate di rosso: solo così questa riga può avere 3 caselle rosse.



Passo 4: Nella quarta riga, le caselle 2 e 4 devono essere bianche a causa delle indicazioni per le diagonali; quindi la casella 3 deve essere rossa.



Passo 5: Infine, nella prima riga le caselle 2, 3 e 4 devono necessariamente essere colorate in sequenza **rosso-bianco-rosso**.

Senza le indicazioni per le diagonali ci sarebbero più soluzioni.

Questa è l'informatica!

Ogni possibile colorazione in rosso e bianco del quadrato 5×5 rappresenta una potenziale soluzione del rompicapo. Tutte insieme costituiscono il cosiddetto *spazio delle soluzioni*. Esiste un algoritmo molto semplice per trovarla: testare ciascuna candidata verificando se rispetta tutte le condizioni. Tuttavia, per un quadrato 5×5 le combinazioni possibili sono 2^{25} (pari a 33'554'432), e occorrerebbe troppo tempo per generarle e verificarle tutte. Per non parlare di un quadrato 6×6, con oltre 68 miliardi di candidate, o di un quadrato 7×7, che ne conta oltre 562'000 miliardi!

 © Castoro Informatico 2026, SSII

Per questo motivo in informatica si impiegano metodi di ricerca più efficienti. Quando una soluzione deve soddisfare determinati vincoli, questi ultimi vengono sfruttati per costruirla passo dopo passo. Spesso si ricorre al backtracking: se una scelta effettuata in corso d'opera (ad esempio l'assegnazione di un colore) conduce a un vicolo cieco in cui non è più possibile trovare una soluzione, si torna sui propri passi per provare un'altra opzione.

Il vantaggio di questo approccio è che si applica a qualsiasi problema che richieda di esplorare uno spazio delle soluzioni, indipendentemente dalle dimensioni o dal numero di soluzioni. L'efficacia è massima quando, per la maggior parte delle decisioni, i vincoli non lasciano margini di scelta (ad esempio imponendo un unico colore possibile). Nel procedimento illustrato sopra ciò si è verificato in ogni passaggio, permettendo di giungere alla soluzione senza dover mai tornare indietro.

Parole chiave e siti web

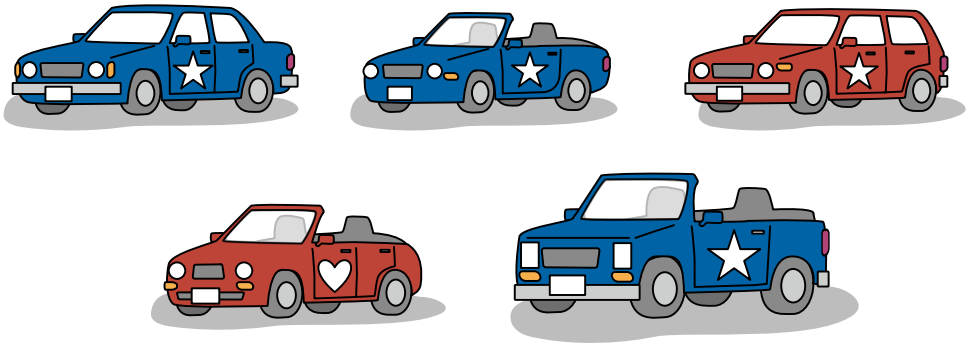
- Algoritmi: `https://it.wikipedia.org/wiki/Algoritmo`
- Ricerca nello spazio delle soluzioni: `https://en.wikipedia.org/wiki/Feasible_region`
- Backtracking: `https://it.wikipedia.org/wiki/Backtracking`



© Castoro Informatico 2026, SSII

12. La fabbrica di auto

In una fabbrica vengono costruiti questi cinque modelli di auto:



Un robot riconosce queste caratteristiche nelle auto appena costruite:

- Colore dell'auto: blu o rosso
- Logo: stella o cuore
- Porte: due (solo davanti) o quattro (davanti e dietro)
- Tetto: aperto o chiuso
- Fari: rotondi o squadrati

Per ogni nuova auto, il robot deve stabilire con certezza a quale modello appartiene.
Binia, esperta di informatica, dice: per farlo, il robot deve osservare solo tre caratteristiche.

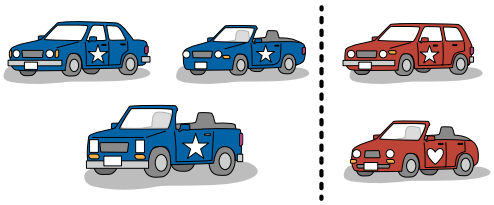
Quali sono queste tre caratteristiche? Cerchiale.

Colore dell'auto, Logo, Porte, Tetto, Fari

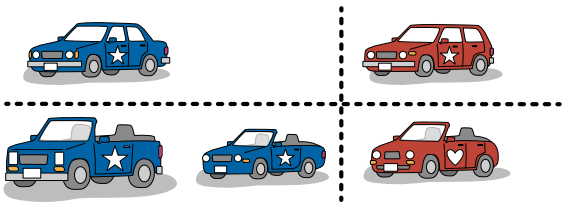
Soluzione

Il robot può stabilire con certezza il modello di ogni nuova auto se osserva il colore dell'auto, il tetto e i fari.

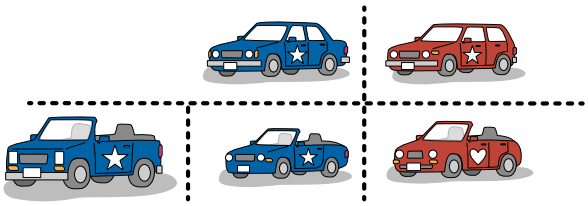
Con ogni caratteristica il robot può dividere i modelli in due gruppi. Con la giusta combinazione di tre caratteristiche può dividere i modelli in gruppi che contengono un solo modello ciascuno; così può assegnare una nuova auto a quel modello. Per trovare le tre caratteristiche giuste si può procedere così:



Si comincia con una caratteristica, per esempio il colore dell'auto, e si dividono i modelli di auto in due gruppi: auto blu e auto rosse.



Poi si cerca una caratteristica che permetta di porre un'ulteriore suddivisione. Possiamo procedere per esempio con il tetto, perché con il colore dell'auto e il tetto si ottengono quattro gruppi: due modelli blu con il tetto aperto, un modello blu con il tetto chiuso, un modello rosso con il tetto aperto, un modello rosso con il tetto chiuso.



Ora bisogna dividere solo il gruppo con due modelli. Per farlo è adatta la caratteristica dei fari, poiché uno dei due modelli blu con il tetto aperto ha i fari rotondi e l'altro squadrati.

C'è una sola risposta corretta. La tabella seguente mostra che solo la combinazione di caratteristiche colore dell'auto, tetto e fari è diversa per ogni modello. Con questa combinazione di caratteristiche si possono distinguere tutti i modelli di auto tra loro - e assegnare con certezza ogni nuova auto a un modello.

 © Castoro Informatico 2026, SSII

Numero	Modello	Colore	Logo	Porte	Tetto	Fari
1		blu	stella	davanti e dietro	chiuso	rotondi
2		blu	stella	solo davanti	aperto	rotondi
3		rosso	stella	davanti e dietro	chiuso	rotondi
4		rosso	cuore	davanti e dietro	aperto	rotondi
5		blu	stella	solo davanti	aperto	squadrati

Con ogni altra combinazione di caratteristiche, almeno due modelli di auto non si possono distinguere tra loro:

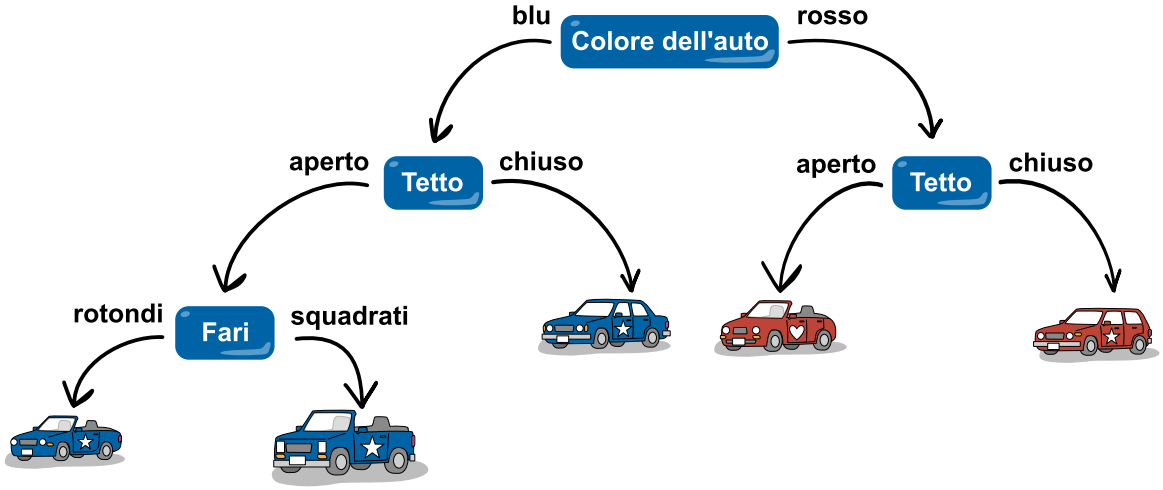
Combinazione di caratteristiche	Modelli di auto con le stesse caratteristiche
Colore, logo, porte	2 e 5
Colore, logo, tetto	2 e 5
Colore, logo, fari	1 e 2
Colore, porte, tetto	2 e 5
Colore, porte, fari	3 e 4
Logo, porte, tetto	1 e 3, e anche 2 e 5
Logo, porte, fari	1 e 3
Logo, tetto, fari	1 e 3
Porte, tetto, fari	1 e 3

Questa è l'informatica!

In questo compito un robot deve assegnare con certezza degli oggetti (le auto) a diversi gruppi (cioè i modelli di auto) in base a determinate caratteristiche (come il colore). Per un'assegnazione di questo tipo – gli esperti parlano anche di *classificazione* – il programma del robot può usare un *albero di decisione*. Un albero di decisione stabilisce quali caratteristiche il robot deve controllare, in quale ordine, e a quali assegnazioni portano i risultati dei controlli.

Scrivere un programma che fa assegnazioni in base a un albero di decisione non è troppo difficile. È più difficile costruire l'albero di decisione, cioè scegliere con astuzia le caratteristiche su cui decidere e il loro ordine. In teoria si potrebbero testare tutti i possibili alberi di decisione per individuare quello corretto, tuttavia, poiché il numero di alberi possibili è solitamente elevatissimo, questo approccio risulta estremamente dispendioso (se non impraticabile). Al contrario, è possibile procedere in modo euristico, sfruttando le osservazioni pratiche sugli oggetti da classificare. Per esempio, si sceglie «colore» come prima caratteristica, perché in base al colore si possono formare due gruppi di auto, o di modelli di auto, più o meno della stessa grandezza. Come seconda caratteristica non si sceglie il «logo», perché tutte le auto blu hanno la stella come logo, quindi controllare questa caratteristica non aiuta a fare l'assegnazione.

Ecco l'albero di decisione per il robot di questo compito:



Generare un albero di decisione a partire da esempi è un procedimento che si può persino automatizzare. Già negli anni Ottanta, nell'ambito della ricerca sui *sistemi basati sulla conoscenza* e sull'*apprendimento automatico*, gli informatici hanno studiato come ricavare alberi di decisione da casi di esempio e dalla frequenza delle caratteristiche osservate.

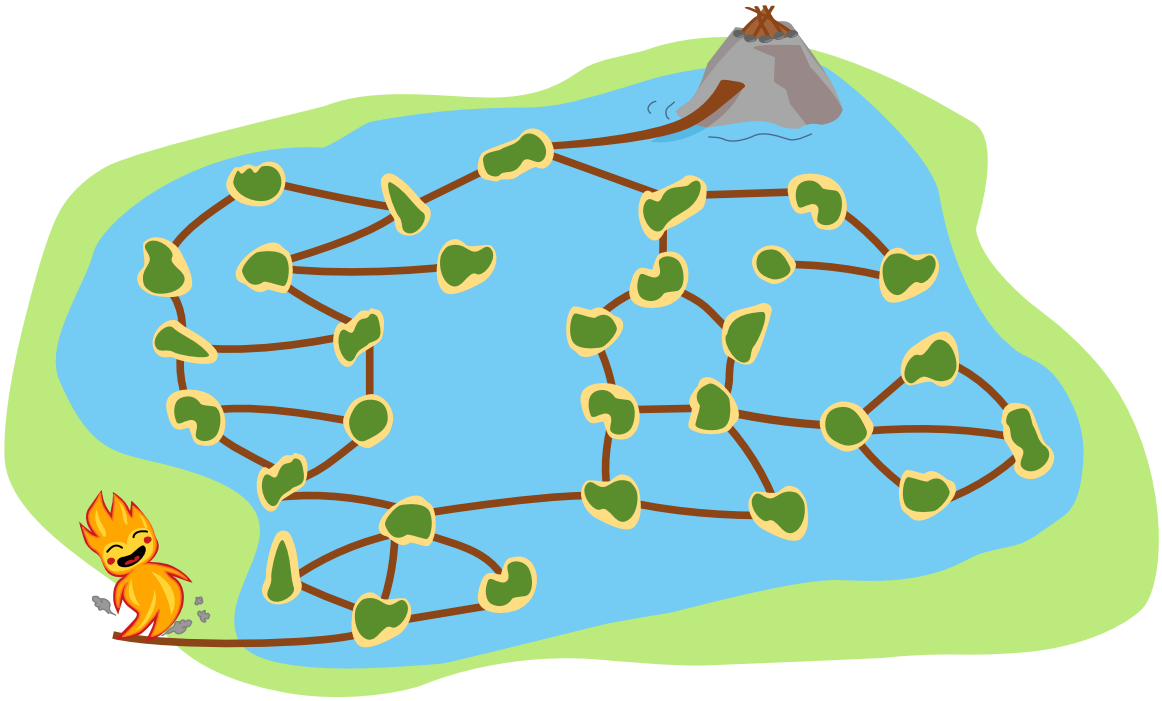
Parole chiave e siti web

- Albero di decisione: `https://it.wikipedia.org/wiki/Albero_di_decisione`
- Sistemi basati sulla conoscenza: `https://it.wikipedia.org/wiki/Sistema_basato_sulla_conoscenza`
- Apprendimento automatico: `https://it.wikipedia.org/wiki/Apprendimento_automatico`

 © Castoro Informatico 2026, SSII

13. Il folletto di fuoco

Sul lago ci sono molte isole, collegate tra loro da passerelle.



Il folletto di fuoco  vuole raggiungere il focolare  sulla roccia e, lungo il percorso, vuole visitare il maggior numero possibile di isole.

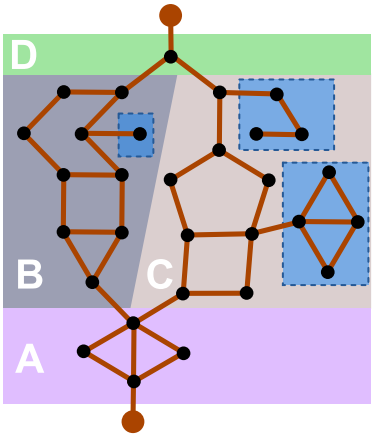
Il folletto deve camminare sulle passerelle, perché l'acqua del lago lo spegnerebbe. Le passerelle, però, sono di legno e bruciano non appena il folletto ci passa sopra: per questo non può attraversare due volte la stessa passerella. Le isole, invece, può visitarle anche più volte.

Qual è il numero massimo di isole che il folletto di fuoco può visitare?

Soluzione

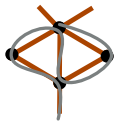
La risposta corretta è 13 isole.

Per la spiegazione semplifichiamo la rappresentazione: disegniamo dei punti per le isole e delle linee per le passerelle.

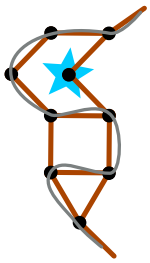


Possiamo dividere le isole del lago in quattro sezioni: le quattro isole in basso (A), l'ultima isola prima della roccia, in alto (D), e, in mezzo, una sezione a sinistra (B) e una a destra (C). Queste sezioni si possono esaminare indipendentemente l'una dall'altra, poiché tra due sezioni c'è al massimo una passerella. Il folletto non può quindi andare avanti e indietro tra le sezioni. In particolare, delle due sezioni centrali B e C può visitarne soltanto una, perché tra queste due sezioni non c'è nessuna passerella. Infine notiamo che ci sono gruppi di isole collegati alle altre isole da un'unica passerella (vedi i rettangoli blu). Se il folletto ci entrasse, non potrebbe più tornare indietro e non riuscirebbe a raggiungere il focolare. Non dobbiamo quindi tenere conto di questi gruppi.

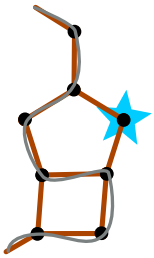
Ora esaminiamo le singole sezioni.



Nella sezione A ci sono quattro isole e abbastanza passerelle per visitarle tutte e quattro, usando ciascuna passerella al massimo una volta.



Nella sezione B ci sono nove isole (senza contare quella nel rettangolo blu). Il folletto non può visitarle tutte: almeno un'isola resta esclusa. Se il folletto visitasse proprio quest'isola (contrassegnata da una stella), dovrebbe saltarne almeno altre due. Nella sezione B il folletto può quindi visitare al massimo otto isole.



La sezione C ha otto isole (senza contare quelle nei rettangoli blu). Anche qui almeno un'isola resta esclusa. E anche qui vale lo stesso: se il folletto visitasse proprio quest'isola (contrassegnata da una stella), dovrebbe saltarne almeno altre due. Nella sezione C il folletto può quindi visitare al massimo sette isole.

Riassumendo, il folletto di fuoco può visitare al massimo 4 isole nella sezione A, al massimo 8 isole nelle sezioni B e C (passando per la sezione B) e 1 isola nella sezione D: in totale, quindi, 13 isole.

Questa è l'informatica!

Per la spiegazione abbiamo rappresentato le isole del lago e le passerelle tra di esse come un *grafo*. Questa struttura matematica è composta da un insieme di *nodi* e da un altro insieme di coppie di nodi, chiamate *archi*. Le isole del lago si possono rappresentare in modo abbastanza diretto come un grafo, scegliendo le isole come nodi e le passerelle come archi.

Modellare un problema con un grafo è utile, poiché per molte domande e molti problemi che riguardano i grafi l'informatica conosce algoritmi in grado di risolverli. Spesso hanno un ruolo importante i *cammini* nei grafi: un cammino è una sequenza di nodi in cui tra due nodi consecutivi c'è sempre un arco.

Molti problemi sui grafi consistono nel trovare un cammino che soddisfi determinate condizioni. In questo compito, per esempio, si cerca il cammino con il maggior numero di nodi che inizia dall'«isola di partenza» del folletto, in basso a sinistra, termina sulla roccia e in cui ciascun arco può comparire al massimo una volta. In altri casi bisogna soddisfare condizioni simili, per esempio nella pianificazione del percorso di un robot che deve ispezionare delle tubature: il robot deve passare per tutti i tubi (= archi), percorrendo però in totale la distanza più breve possibile.

Parole chiave e siti web

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

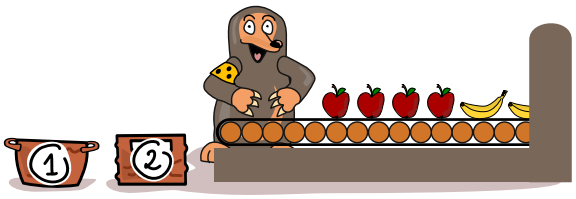


© Castoro Informatico 2026, SSII

14. Il metodo di Muddy

Su un nastro trasportatore arriva della frutta in quest'ordine: prima le mele, poi le banane e infine le pere.

La talpa Muddy vuole sapere se sul nastro trasportatore arriva lo stesso numero di frutti per ciascun tipo. Muddy sa distinguere le mele, le banane e le pere. Purtroppo non sa contare; ha però a disposizione due cesti, che sa distinguere l'uno dall'altro.



Con i frutti e i cesti può fare le seguenti azioni:



mettere un frutto in uno dei cesti



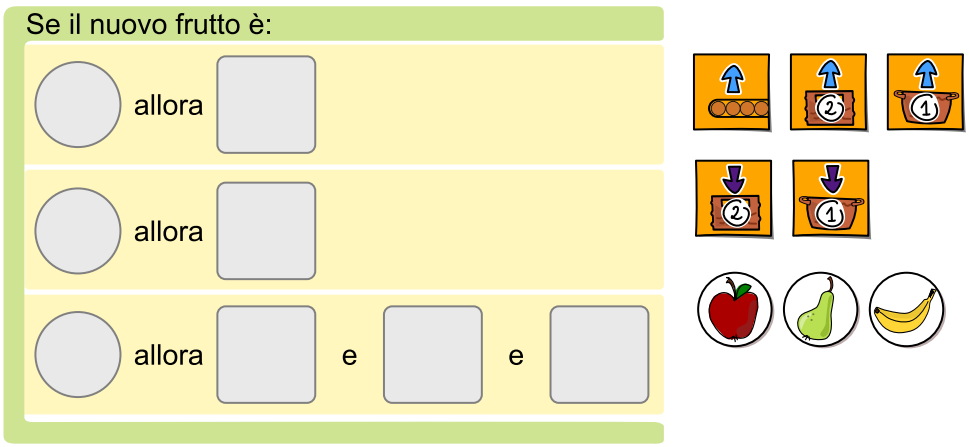
prendere un frutto qualsiasi da uno dei cesti (Muddy non può vedere quale) e metterlo da parte



prendere un frutto dal nastro e metterlo da parte

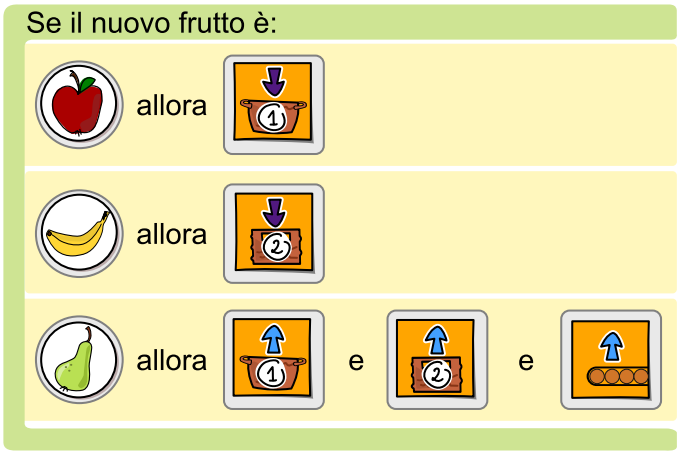
Con queste azioni Muddy ha ideato un metodo per gestire i frutti che arrivano sul nastro trasportatore. Applica questo metodo a ogni nuovo frutto che gli arriva e continua finché il nastro trasportatore non è vuoto, oppure finché non prova a prendere un frutto da un cesto vuoto. Se a quel punto sia il nastro trasportatore sia i due cesti sono vuoti, allora sul nastro è arrivato lo stesso numero di mele, banane e pere.

Come funziona il metodo di Muddy?



Soluzione

Muddy ha ideato questo metodo: mette tutte le mele nel cesto 1 e tutte le banane nel cesto 2, o viceversa. Dopodiché, per ogni pera sul nastro trasportatore prende esattamente una mela e una banana dai cesti e infine mette da parte anche la pera.



Solo se sul nastro arriva lo stesso numero di mele, banane e pere, i due cesti e il nastro trasportatore si svuotano nello stesso momento. Guardando più da vicino, ci sono diversi casi in cui Muddy smette di applicare il suo metodo:

Caso 1: almeno un cesto è vuoto, ma sul nastro c'è ancora almeno una pera. Allora, ...

- se il cesto delle mele è vuoto, sul nastro sono arrivate più pere che mele;
- se il cesto delle banane è vuoto, sul nastro sono arrivate più pere che banane.

Caso 2: il nastro trasportatore è vuoto, ma in almeno un cesto ci sono ancora dei frutti. Allora, ...

- se nel cesto ci sono ancora delle mele, sul nastro sono arrivate meno pere che mele;
- se nel cesto ci sono ancora delle banane, sul nastro sono arrivate meno pere che banane.

Caso 3: il nastro trasportatore e i due cesti sono vuoti. Allora sul nastro è arrivato lo stesso numero di frutti per ciascun tipo.

Questa è l'informatica!

Muddy usa due cesti per confrontare il numero dei frutti. Più precisamente, usa i cesti per «memorizzare» le mele e le banane finché non arrivano le pere. Poiché tiene le mele e le banane separate, per ogni pera sul nastro può prendere, senza guardare, un frutto da ciascun cesto ed essere sicuro di aver preso in tutto una mela e una banana.

In informatica esiste una struttura simile a quella di questo compito: è composta da un nastro di input (come il nastro trasportatore di Muddy) e da due *pile* (cioè memorie, come i cesti di Muddy) e si chiama *automa a due pile*. A differenza dei cesti di Muddy, però, nelle pile regna l'ordine: ciò che è stato messo per ultimo nella pila viene tolto per primo.

In generale, gli *automi* hanno un ruolo importante nell'informatica teorica. Un automa legge dei simboli di input: a seconda del simbolo letto e del suo stato attuale, esegue delle azioni e passa a uno stato successivo. L'input viene accettato se la parola è stata letta completamente ed è stato raggiunto uno stato finale. L'insieme di tutte le parole accettate costituisce il linguaggio che l'automa riconosce. Un automa definisce quindi un *linguaggio formale*, che in alternativa può essere descritto anche da una *grammatica formale*.

Nell'informatica teorica esiste infatti un legame fondamentale tra i *linguaggi formali* (insiemi di parole), le *grammatiche formali* (gli insiemi di regole che generano questi linguaggi) e gli *automi* (i modelli di macchina che riconoscono, o accettano, questi linguaggi). Determinati tipi di automi corrispondono a determinati tipi di grammatiche, e quindi di linguaggi. Per esempio, un linguaggio accettato da un automa a pila con una sola pila viene chiamato, nella *gerarchia di Chomsky* dei linguaggi formali, linguaggio libero dal contesto. I tipici linguaggi informatici, come i linguaggi di programmazione o di markup (per es. HTML), sono liberi dal contesto e possono essere elaborati rapidamente con algoritmi efficienti.

Con una sola pila, o un solo cesto, Muddy potrebbe confrontare due tipi di frutta; con due pile, tre. E così via? No: in realtà è già la seconda pila a fare la differenza decisiva, e ulteriori pile non portano alcun vantaggio. In informatica è infatti noto che un automa a due pile è già in grado di simulare una macchina di Turing. La macchina di Turing si trova in cima alla gerarchia di Chomsky e può riconoscere qualsiasi linguaggio formale che sia calcolabile. La macchina di Turing, e quindi anche l'automa a due pile, è potente quanto qualsiasi computer moderno.

Parole chiave e siti web

- linguaggio libero dal contesto (CFL – context-free language): `https://it.wikipedia.org/wiki/Linguaggio_libero_dal_contesto`
- automa a due pile: `https://en.wikipedia.org/wiki/Two-Stack_Push_Down_Automaton` (`https://it.wikipedia.org/wiki/Automa_a_pila`)



© Castoro Informatico 2026, SSII

15. 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

16. Nascondino

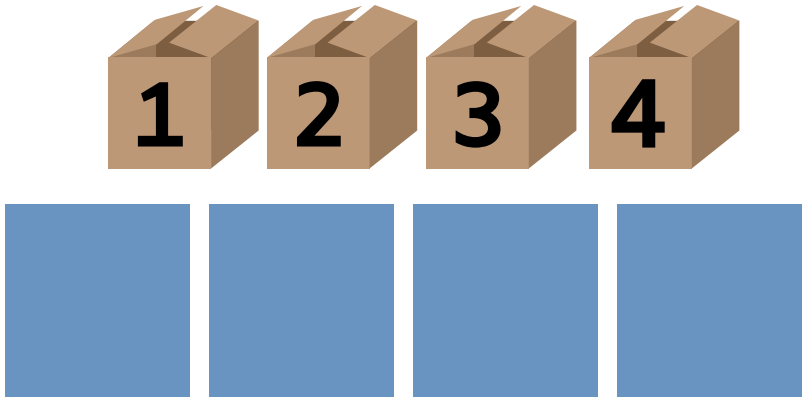
Ben si nasconde in una di quattro scatole numerate, per esempio nella scatola 2.



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:

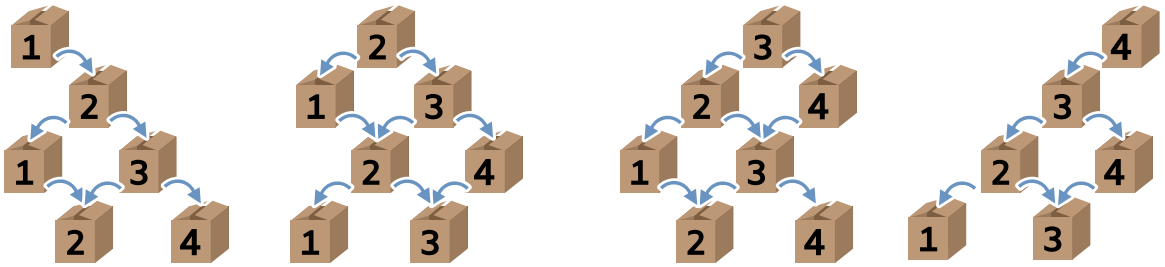


Ora mostriamo che queste risposte sono corrette. Chiamiamo *mossa* ogni spostamento di Ben in questo gioco del nascondino.

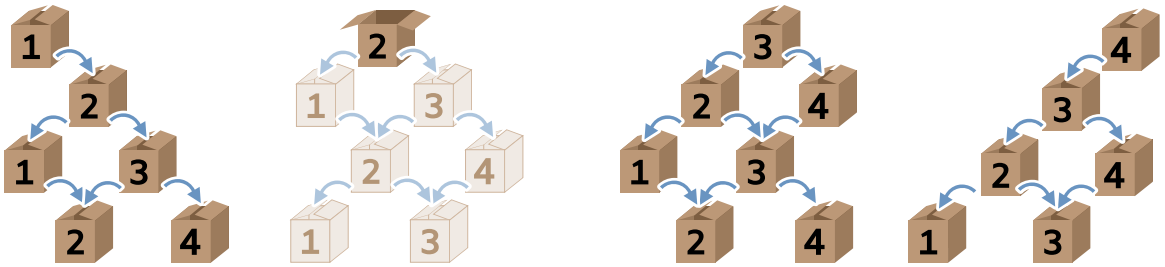
All'inizio Ben può scegliere liberamente la sua «scatola di partenza». Poi, se Ben

- 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):

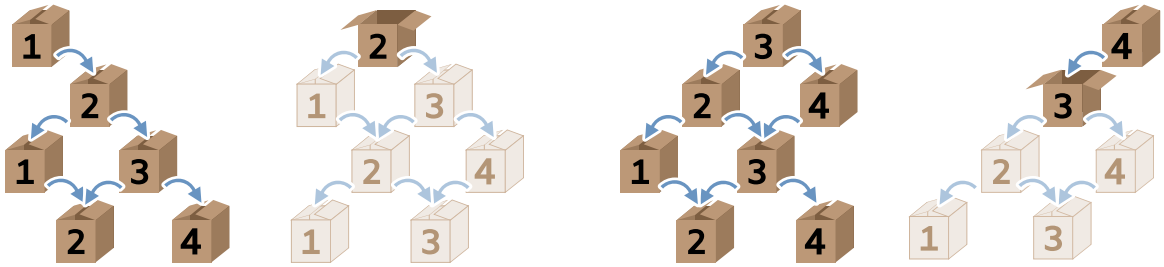


Cosa succede se Anja guarda nelle scatole nell'ordine **2-3-3-2**? Prima Anja guarda nella scatola 2. Se trova Ben lì, la scatola 2 è la sua scatola di partenza, e Ben non può più fare le mosse che partono da lì.

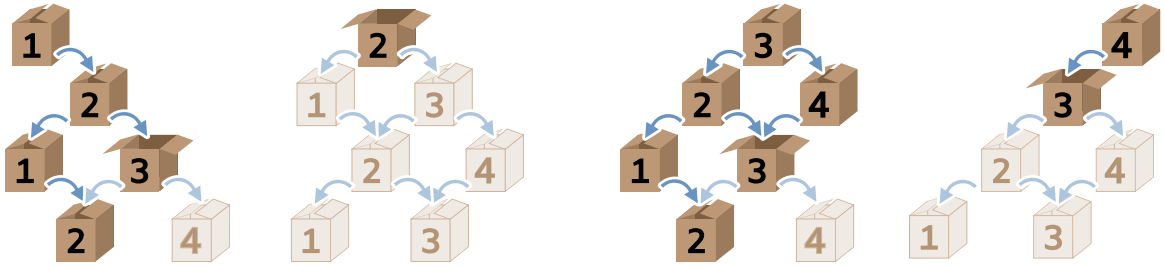


Poi Anja guarda nella scatola 3. Se trova Ben lì, questa è la sua seconda scatola. Al secondo livello delle sequenze di mosse, la scatola 3 compare ancora solo tutta a destra. Ora Ben non può più fare le mosse che partono da lì, e non dobbiamo più considerarle.

 © 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`

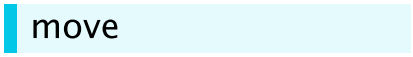
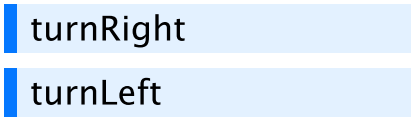
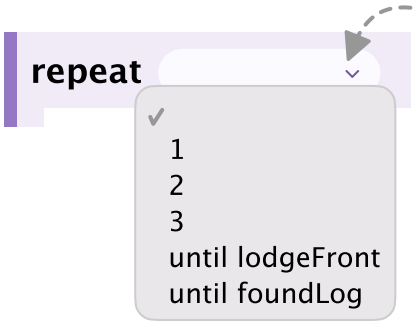


© Castoro Informatico 2026, SSII

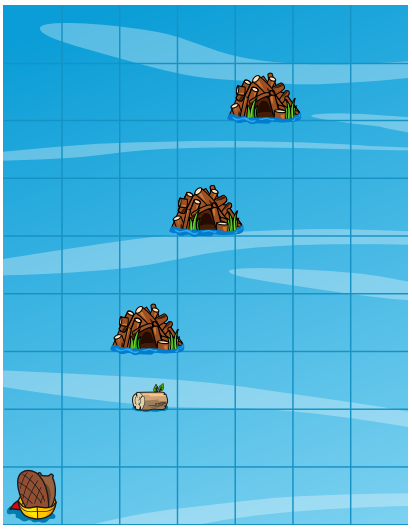
17. Un aiuto dai vicini

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?

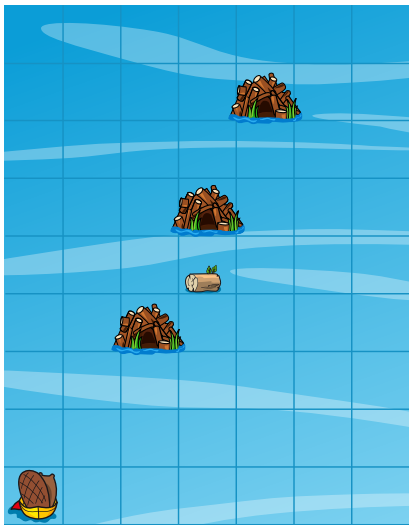
Puoi usare le seguenti istruzioni:

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*.

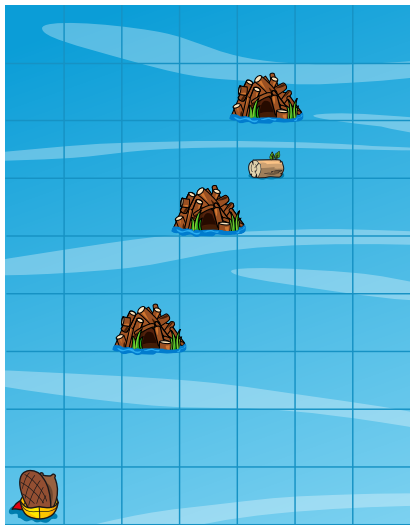
Scrivi delle istruzioni di percorso con cui Belinda raccoglie il tronco in tutti e tre i giorni. Appena ha raccolto il tronco, Belinda non deve più proseguire verso i vicini successivi.



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:

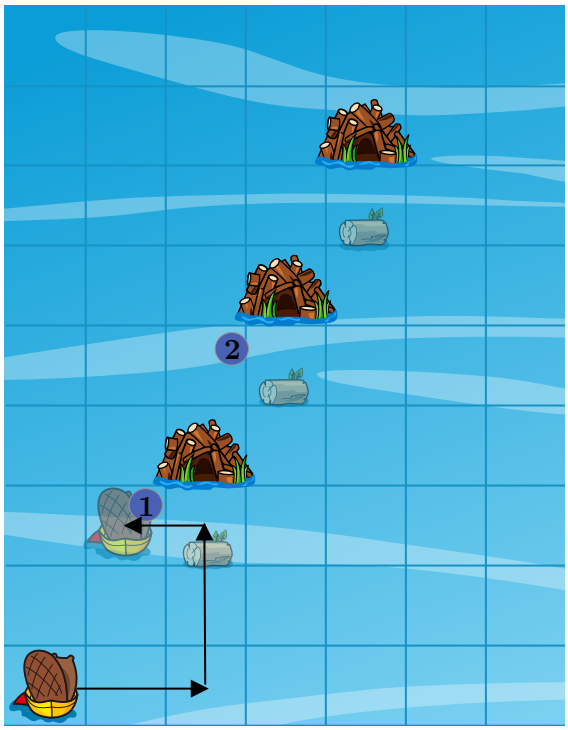
Parte 1: percorso fino al tronco

```
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)`

A. Autori dei quesiti

Maxat Aubakirov
Masiar Babazadeh
Liam Baumann
Wilfried Baumann
Leonardo Cavalcante
Emily Cavalcanti
Cesar Cepeda
Maria Cepeda
Špela Cerar
Christian Datzko
Heather Duan
Nora A. Escherle
Susanne Falkenberg-Thut
Nadine Fröhlich
Gerald Futschek
Daniel Aagrén Geert
Christian Giang
Arnheiður Guðmundsdóttir
Jens Hartmann
Yong-ju Jeon
Heidi Kaarto
Asterios Karagiannis
Geetanjali Khatri
Doyong Kim
Vaidotas Kinčius
Patrik Klofáč
Lukas Lehner
Yoshiaki Matsuzawa
Anna Morpurgo
Madhavan Mukund
Justina Oostendorp
Harriet Page
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
Maiko Shimabuku
Itsuki Shimada
Bronius Skūpas
Emma Sotomayor
Jacqueline Staub
Alieke Stijf

Pedro Suarez

Maciej M. Syslo

Yahya Tabesh

Ezra Templonuevo

Jiří Vaníček

Michael Weigend

Kyra Willekes

Yichan Yuan

© Castoro Informatico 2026, SSII

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/

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

© Castoro Informatico 2026, SSII