CONCURSO BEBRAS · ESPAÑA

Bebras 2024

1.º y 2.º de Bachillerato · Grupo VI (16–18 años)

Cuadernillo con las 15 tareas del concurso, sus soluciones explicadas y las secciones «Esto es informática» y «Esto es pensamiento computacional».

bebras.ehu.es

Presentación

Bebras es un desafío internacional de informática y pensamiento computacional que se celebra cada curso en más de 80 países. Sus tareas son problemas breves, resolubles sin conocimientos previos de programación, que acercan al alumnado a ideas fundamentales de la informática: algoritmos, estructuras de datos, lógica, grafos o representación de la información.

Este cuadernillo recoge las 15 tareas que resolvió el alumnado de 1.º y 2.º de Bachillerato en la edición de 2024, realizada en la plataforma ViLLE (curso «2024 Spanish VI»). Cada tarea valía 10 puntos, con un máximo de 150. Para cada tarea se incluye:

- **Enunciado**, en una página que se puede fotocopiar para trabajar en clase.
- **Solución** con la respuesta correcta y su explicación.
- **Esto es informática**: el concepto informático que hay detrás de la tarea.
- **Esto es pensamiento computacional**: las destrezas que se ponen en juego al resolverla.

La dificultad indicada es la que asignó el taller internacional Bebras para la franja de 16 a 19 años; cuando una tarea no tenía asignada esa franja, se indica entre paréntesis la dificultad en la franja más próxima. Las opciones de respuesta aparecen en el orden de los materiales originales; en ViLLE el orden pudo ser distinto.

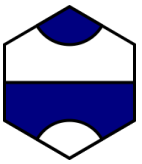
Resumen de tareas y respuestas

N.º	Tarea	Código	Dificultad	Respuesta
1	Palago	2024-HU-04	Difícil	B) Gana el blanco.
2	Préstamos de canicas	2024-AT-03	—	C) Tim.
3	Red ferroviaria	2024-PK-03	Media	A) 13.
4	Bolas rojas y azules	2024-BG-01b	Media	azul, roja, roja, azul, azul, azul, roja, roja.
5	Exploración	2024-DE-05	Media	① ningún cartel · ② a la que apunta la flecha.
6	El mapa del tesoro	2019-VN-04	Media	D.
7	Amigos	2024-LT-05	Fácil	Dave y Fred.
8	La puntuación más alta	2024-SK-04	Fácil	A B B C **C** B **B B** B A A A → 2 + 2 + 4 + 3 = 11 puntos.
9	En busca del tesoro	2024-HU-03	— (fácil en 14–16)	4.
10	Días soleados	2024-DE-04b	Fácil	① hizo sol · ② ningún castor nadó en el estanque de la cascada helada.
11	La máquina de globos	2024-DE-06a	Difícil	Por ejemplo, B E B C A C B D B (hay 4 respuestas válidas).
12	El muro de ladrillos	2024-FI-03	Media	A y C.
13	Cambia y avanza	2024-BE-01b	Media	D.
14	Cadenas de palabras	2024-CA-02	Fácil	C) BAT → CAT → COT.
15	La pulsera Bebras	2024-TW-03	— (fácil en 14–16)	D) 6 cuentas.

1 Palago

2024-HU-04 · Hungría
Dificultad: Difícil

Palago es un juego para dos personas que se juega con fichas hexagonales idénticas, coloreadas de azul y blanco como se muestra.



Una persona juega con el blanco y la otra con el azul. El objetivo es formar **formas cerradas** de tu color.

Quien empieza coloca dos fichas de modo que los colores de los bordes que se tocan coincidan. A partir de ahí, en cada turno se colocan siempre **dos fichas**, respetando estas reglas:

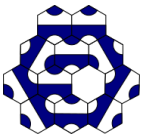
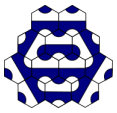
1. Al menos una de las dos fichas debe estar en contacto con las fichas ya colocadas.
2. Las dos fichas colocadas deben tocarse entre sí.
3. Todos los colores de los bordes que se tocan deben coincidir.

Gana quien consigue una forma cerrada de su propio color que contenga al menos un tramo recto. Pero pierde quien, en la misma jugada, cierra una forma de cada color. Si para ganar basta con una ficha, no hace falta colocar la segunda.

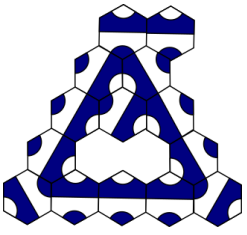


En esta situación el azul tiene una forma cerrada, pero no contiene ningún tramo recto, así que el azul no ha ganado.

Ejemplo

Situación de la partida	Siguiente jugada del blanco	
		El blanco gana: ha formado una forma blanca cerrada (completamente rodeada de azul).
		El blanco pierde: es su turno y los dos jugadores tienen al menos una forma cerrada. Gana el azul.

Esta es la situación de una partida de Palago:



Las dos personas intentan siempre ganar lo antes posible. ¿Qué ocurre en esta partida?

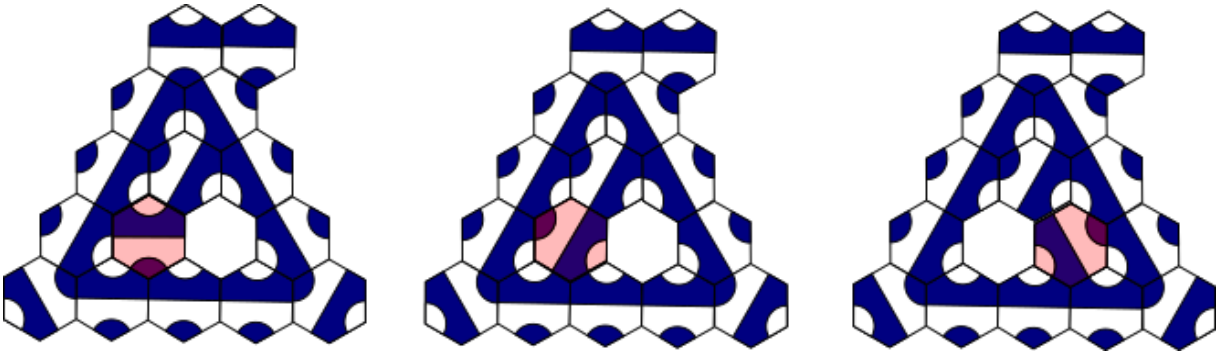
A) Gana el azul.
B) Gana el blanco.
C) No gana nadie.
D) Depende de quién coloque la siguiente ficha.

Respuesta correcta: B) Gana el blanco.

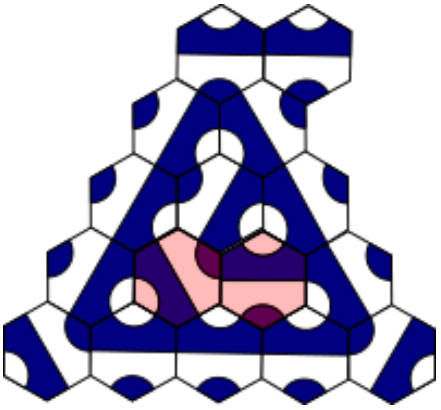
Explicación

La respuesta correcta es la **B**.

Si mueve el blanco, puede completar una forma blanca con una sola ficha sin cerrar ninguna forma azul. Las imágenes muestran varias maneras de hacerlo:



Si mueve el azul, puede rellenar el hueco y cerrar su forma usando las dos fichas. Pero al hacerlo cerrará también una forma blanca, así que pierde la partida.



Si el azul no cierra ninguna forma (y por tanto no gana), el blanco puede hacerlo en su siguiente turno, como se ha explicado arriba. Y el azul no tiene forma de cerrar una figura alrededor de todas las fichas ya colocadas en un solo turno, así que, como muy tarde en el turno siguiente, gana el blanco.

El problema se basa en una posición descrita por Dave Dyer.

Esto es informática

En un juego de estrategia por turnos como Palago hay que tener una estrategia para ganar, o al menos para no perder, en las próximas jugadas: prever los siguientes pasos y sus consecuencias, y decidir en función de ellas.

Muchos juegos de estrategia pueden representarse con un **árbol de juego**: la raíz es la posición inicial y de cada posición salen todas las jugadas posibles. Es un grafo que representa todos los estados posibles de la partida. Si se puede construir el árbol, quizá se encuentre un camino que lleve a la victoria.

En juegos como el ajedrez, el go o el Palago el árbol es enorme, así que los programas trabajan con un árbol parcial y analizan solo unas pocas jugadas hacia delante. Palago es, además, un rompecabezas creativo diseñado por Cameron Browne: muchas personas lo usan para crear «palagonianos» (habitantes de Palagonia) o animales con las fichas.

Esto es pensamiento computacional

Para resolver la tarea hay que comprender bien unas reglas nuevas, **evaluar** posiciones y razonar sobre las jugadas posibles de cada jugador (pensamiento lógico). Conviene analizar por separado los dos casos (mueve el blanco / mueve el azul), es decir, **descomponer** el problema, y anticipar las consecuencias de cada jugada, como haría un programa que explora un árbol de juego.

Palabras clave y enlaces: Árbol de juego, estrategia, teoría de juegos · es.wikipedia.org/wiki/Árbol_de_juego · en.wikipedia.org/wiki/Palago

2 Préstamos de canicas

2024-AT-03 · Austria
Dificultad: —

Tres castores amigos juegan a menudo con canicas y se las prestan unos a otros. Para saber quién debe cuántas canicas a quién, decidieron apuntar todos los préstamos en una única hoja de papel:

Ejemplo de lista compartida (no es relevante para la pregunta):

Presta	Recibe	Cantidad
Anna	Tim	5
Tim	Bernd	3
...	...	...

Con el tiempo empezaron a desconfiar. Se acusaban unos a otros de hacer trampa y pensaban que algunos préstamos nunca habían ocurrido.

Así que decidieron no depender de una sola lista a la que todos tuvieran acceso: cada castor llevaría su propia lista para detectar intentos de trampa. Al cabo de una semana comparan sus listas. ¿Ha vuelto a hacer trampa alguien?

Lista de Anna

Presta	Recibe	Cant.
Tim	Anna	2
Bernd	Tim	1
Bernd	Anna	3
Anna	Tim	2

Lista de Tim

Presta	Recibe	Cant.
Tim	Anna	2
Bernd	Tim	1
Tim	Anna	3
Anna	Tim	2

Lista de Bernd

Presta	Recibe	Cant.
Tim	Anna	2
Bernd	Tim	1
Anna	Tim	2
Bernd	Anna	3

¿Quién ha hecho trampa?

A) Anna
B) Bernd
C) Tim
D) Nadie

Respuesta correcta: C) Tim.

Explicación

La respuesta correcta es **Tim**.

Si solo nos fijamos en la columna de cantidades, (2, 1, 3, 2), parece que el tramposo es Bernd, porque es el único cuya lista tiene otra secuencia (2, 1, 2, 3). Pero no es así: la lista de Bernd solo tiene los préstamos en otro orden, y el resultado total es el mismo.

Quien hizo trampa fue Tim: cambió el préstamo (Bernd → Anna, 3) por (Tim → Anna, 3). Si nadie lo hubiera notado, habría ganado 3 canicas a costa de Bernd. Su lista es la única que no coincide con las otras dos en el conjunto de préstamos.

Esto es informática

La idea de no guardar todas las transacciones en un único servidor central de una entidad de confianza, sino distribuirlas por una red entre iguales (*peer-to-peer*), es una de las ideas básicas de la tecnología **blockchain** (cadena de bloques) o de los **registros distribuidos**.

El método de la tarea solo funciona si la mayoría de los participantes dice la verdad. Blockchain va un paso más allá con un **algoritmo de consenso** que, por ejemplo, exigiría que quien quiera hacer trampa controle más capacidad de cálculo que todos los demás participantes juntos, algo imposible si la red es lo bastante grande. Aun así, la tarea ya muestra cómo se puede reforzar la confianza en un entorno sin confianza repartiendo el poder entre todos, en lugar de depender de una única fuente de verdad.

Esto es pensamiento computacional

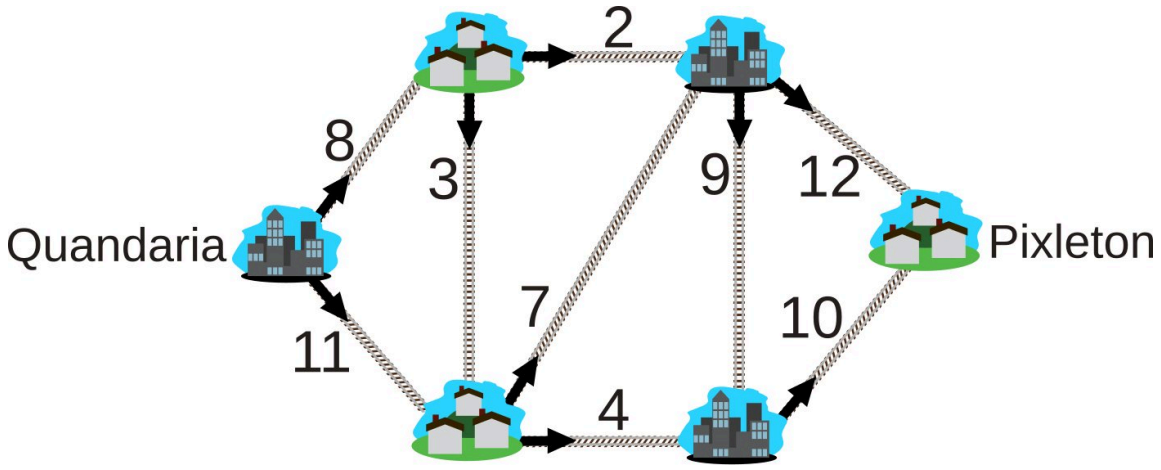
Hay que **abstraer** lo esencial de cada lista: lo que importa es el conjunto de préstamos y su efecto, no el orden en que aparecen. Después se **evalúan** las tres listas comparándolas entre sí para localizar la que discrepa de la mayoría, evitando la conclusión precipitada a la que lleva fijarse solo en un patrón superficial (la columna de cantidades).

Palabras clave y enlaces: Blockchain, registro distribuido, consenso · es.wikipedia.org/wiki/Cadena_de_bloques

3 Red ferroviaria

2024-PK-03 · Pakistán
Dificultad: Media

En el país de Bebravia, los pueblos vecinos están unidos por una red de vías de tren. Por cada vía puede circular un número limitado de trenes al día, indicado en el esquema. Trenes distintos pueden compartir una vía.



Quandaria se ha ofrecido a enviar materiales a Pixleton. En este sentido, los trenes deben seguir siempre las flechas.

¿Cuál es el número máximo de trenes que pueden salir cada día de Quandaria y llegar a Pixleton?

A) 13
B) 15
C) 19
D) 22

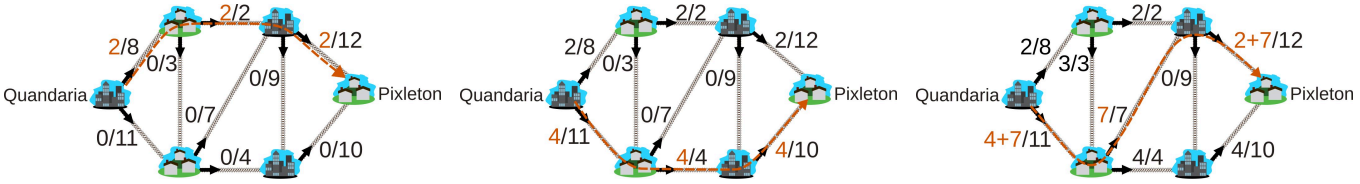
Respuesta correcta: A) 13.

Explicación

La respuesta correcta es la **A) 13**.

Lo primero que se observa es que Quandaria (Q) no puede enviar a Pixleton (P) más de 8 + 11 = 19 trenes, que es el máximo de trenes que pueden salir de Q. Así que la respuesta D) 22 es imposible.

Una forma de resolver el problema es buscar un camino lo más corto posible de Q a P (aquí, de longitud 3) y calcular cuántos trenes admite como máximo. Después se repite el proceso en el resto de la red, con la capacidad que queda en cada vía. Los esquemas muestran tres caminos encontrados así, con una capacidad conjunta de 2 + 4 + 7 = 13 trenes. Hay otras formas de llevar 13 trenes de Q a P, algunas con caminos más largos.



Para comprobar que 13 es el máximo, fíjate en las tres vías del centro de la red, con capacidades 2, 7 y 4: todos los caminos pasan por una de ellas, así que no se puede superar 2 + 7 + 4 = 13.

Esto es informática

Esta tarea es un ejemplo del **problema del flujo máximo**. Es como un juego de tuberías: hay una red de tubos con distintas capacidades y queremos saber cuánta agua puede circular desde un origen hasta un destino.

El problema se puede resolver con varios algoritmos: Ford-Fulkerson, Edmonds-Karp, Dinic y otros. Las vías centrales por las que hay que pasar forzosamente forman lo que se llama un **corte mínimo**; el teorema de flujo máximo–corte mínimo dice que ambos valores coinciden.

Cada algoritmo tiene ventajas e inconvenientes en eficiencia, facilidad de programación y tipo de redes a las que se aplica. Los algoritmos de flujo máximo se usan para optimizar el tráfico, en redes de distribución de agua o electricidad, en asignación de recursos, en redes de ordenadores y en muchos otros campos.

Esto es pensamiento computacional

Resolver la tarea requiere **pensamiento algorítmico** y **evaluación**. La solución se **descompone** en una secuencia de pasos, cada uno de los cuales consiste en encontrar un camino entre Q y P con la máxima capacidad disponible. El algoritmo termina cuando ya no queda ningún camino posible. Además hay que evaluar el resultado: el corte de 13 demuestra que no se puede hacer mejor.

Palabras clave y enlaces: Grafo, flujo máximo, corte mínimo, algoritmo de Ford-Fulkerson · es.wikipedia.org/wiki/Algoritmo_de_Ford-Fulkerson · es.wikipedia.org/wiki/Problema_de_flujo_máximo

4 Bolas rojas y azules

2024-BG-01b · Bulgaria
Dificultad: Media

Tenemos una fila de bolas rojas y azules:



Contamos el número de bolas azules de izquierda a derecha empezando por la primera bola, después empezando por la segunda, y así sucesivamente. Obtenemos esta secuencia:

```
3, 3, 2, 1, 1, 1
```

Ahora escribimos 0 si el número es par y 1 si es impar, y obtenemos:

```
1 1 0 1 1 1
```

Esta es otra secuencia de ceros y unos (par e impar): `0 1 1 1 0 1 0 0`. Construye la fila de bolas que le corresponde.

Colorea cada bola de rojo o de azul:



Respuesta correcta: azul, roja, roja, azul, azul, azul, roja, roja.

Explicación

Respuesta correcta:



Hay dos algoritmos sencillos para llegar a la solución: uno recorre la secuencia de izquierda a derecha y otro de derecha a izquierda.

De derecha a izquierda (el más sencillo). El último número dice si la última bola es azul (1) o roja (0). Hacia la izquierda, el número de bolas azules cambia exactamente cuando la secuencia cambia de 0 a 1 o de 1 a 0: si dos valores consecutivos son distintos, en esa posición hay una bola azul; si son iguales, una roja.

0	1	1	1	0	1	0	0
4 azules	3 azules	3 azules	3 azules	2 azules	1 azul	0 azules	0 azules

Así, el número de bolas azules desde cada posición es 4 3 3 3 2 1 0 0, y de ahí sale la fila.

De izquierda a derecha. Se recorren los valores de dos en dos con dos variables, `c` (valor actual) y `cc` (siguiente). Si son iguales, al pasar a la siguiente posición el número de azules no cambia, así que en la posición actual hay una bola roja; si son distintos, hay una azul. La última bola es roja si el último valor es 0 (cero azules) y azul si es 1.

```
leer c
repetir n-1 veces:
    leer cc
    si c == cc: escribir 'roja'  si no: escribir 'azul'
    c ← cc
si c == '0': escribir 'roja'  si no: escribir 'azul'
```

Esto es informática

En informática se usan muchas **codificaciones** distintas para describir una situación a un ordenador. A menudo se puede omitir información, simplificar la codificación y aun así no perder lo original. Aquí no se guardan los números exactos de bolas azules, sino solo su paridad con ceros y unos, y aun así se puede reconstruir la fila completa. Por eso, para usar una codificación hay que saber cómo se creó, es decir, qué algoritmo se aplicó.

Esta idea está en la base de los **algoritmos de compresión**, que reducen el tamaño de un archivo sin perder información; por ejemplo, el formato de imagen PNG combina varios de ellos.

La fila se puede reconstruir con un **algoritmo de una sola pasada**, que procesa los datos una sola vez, leyéndolos y descartándolos sin volver atrás. Este enfoque ahorra tiempo y memoria al cifrar, comprimir, descomprimir y, en general, al procesar grandes volúmenes de datos.

Esto es pensamiento computacional

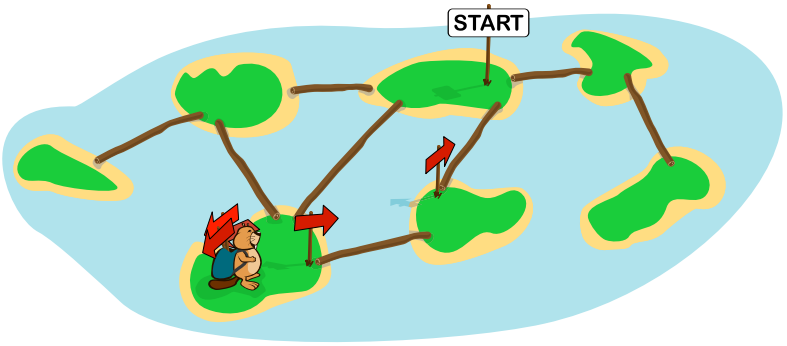
Hay que descubrir la regla que relaciona dos posiciones consecutivas (**reconocimiento de patrones**) y convertirla en un procedimiento que se repite en **bucle** (**pensamiento algorítmico**). Elegir el sentido de recorrido adecuado simplifica mucho el trabajo: es un buen ejemplo de cómo pensar en la eficiencia de un algoritmo y en cómo gestionar los datos con los mínimos recursos.

Palabras clave y enlaces: Compresión de datos, algoritmo de una pasada, paridad · es.wikipedia.org/wiki/Compresión_de_datos

5 Exploración

2024-DE-05 · Alemania
Dificultad: Media

El castor Harry está explorando una red de islas unidas por troncos, como la de la imagen. Quiere visitar todas las islas de la red. Desde cualquier isla ve todas las islas vecinas y distingue si tienen un cartel.



En la imagen, Harry empezó en una isla cualquiera y ya ha llegado a otras dos siguiendo esta estrategia:

Pon un cartel de **INICIO** en una isla cualquiera.
Cada vez que llegues a una isla, haz lo siguiente:
  Si esta isla ya tiene un cartel:
    deja el cartel como está.
  Si no:
    pon un cartel con una flecha que apunte a la isla de la que acabas de venir.
  Si hay una isla vecina que tiene __①__, entonces:
    ve a esa isla.
  Si no:
    mira el cartel de la isla en la que estás.
    Si es un cartel con flecha, entonces:
      ve a la isla __②__.
    Si no:
      ya has visitado todas las islas de la red.

Completa los huecos con las instrucciones correctas.

Hueco ①

a) ningún cartel
b) un cartel con flecha
c) el cartel de INICIO

Hueco ②

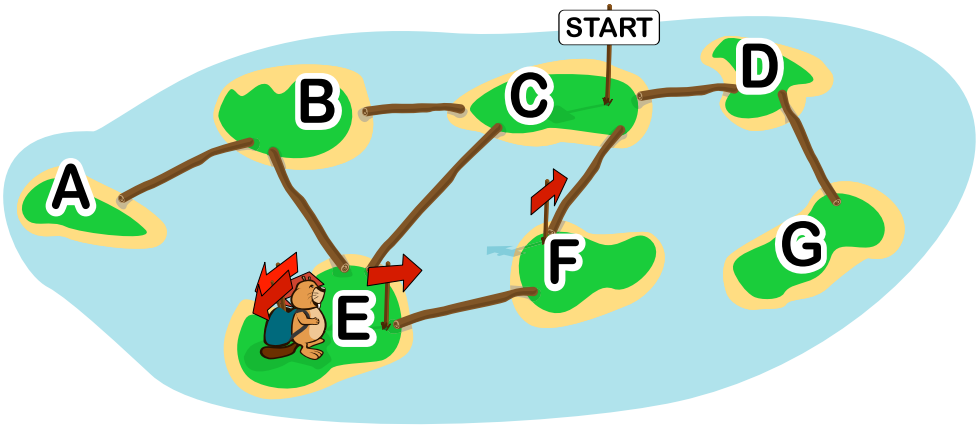
a) de la que acabas de venir
b) a la que apunta la flecha

Respuesta correcta: ① ningún cartel · ② a la que apunta la flecha.

Explicación

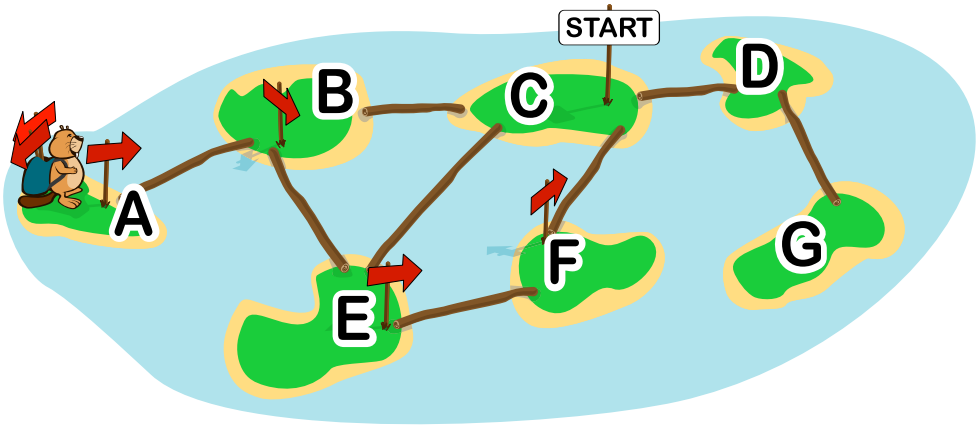
Primer hueco: «ningún cartel». Las islas en las que Harry ya ha estado tienen un cartel, y las que todavía no ha visitado no lo tienen (porque pone un cartel cada vez que llega a una isla que no lo tiene). Por tanto, debe ir a las islas sin cartel, es decir, a las que aún no ha visitado.

Si eligiera islas con flecha, daría vueltas para siempre entre islas ya visitadas. Por ejemplo, en la situación de abajo debe ir a la isla vecina sin cartel, B. Si prefiriera las islas con cartel, se movería entre C, E y F y nunca llegaría a las islas que faltan.



Si eligiera la isla con el cartel de INICIO, en la segunda vuelta del algoritmo Harry estaría otra vez en la isla inicial y el algoritmo ejecutaría por error el último «si no» («ya has visitado todas las islas»).

Segundo hueco: «a la que apunta la flecha». Así Harry retrocede hasta una isla que todavía tenga vecinas sin explorar. Si volviera a la isla de la que acaba de venir, podría quedarse yendo y viniendo para siempre entre dos islas sin vecinas nuevas. Por ejemplo, abajo está en la isla A, que no tiene vecinas sin explorar; solo puede ir a B. Desde B, si volviera a la isla de la que acaba de venir, iría a A, luego a B, luego a A... Siguiendo las flechas va a E, F y C, y desde allí a la isla D, que aún no ha visitado.



Esto es informática

La red de islas es un **grafo**: una estructura de datos formada por vértices (las islas) y aristas que los conectan (los troncos). Con grafos se modelan, por ejemplo, estaciones unidas por líneas de tren, personas unidas por amistad o posiciones de una partida de ajedrez unidas por jugadas válidas.

A menudo hay que recorrer todos los vértices de un grafo, por ejemplo para encontrar uno con ciertas características. Si el grafo es grande, hace falta una estrategia para no olvidar ninguno. La de esta tarea se llama **búsqueda en profundidad** (*depth-first search*). Sirve, por ejemplo, para resolver un laberinto. Las flechas hacen el papel de la pila que usa el algoritmo para poder retroceder (*backtracking*).

El algoritmo está escrito en **pseudocódigo**: una descripción de los pasos que mezcla lenguaje natural y convenciones de programación, pensada para que la lean personas y no un ordenador.

Esto es pensamiento computacional

Hay que leer y comprender un algoritmo abstracto e imaginar su ejecución paso a paso (**pensamiento algorítmico**). Para descartar opciones es útil **evaluar** cada posibilidad buscando un contraejemplo concreto en el que el algoritmo fallaría (un bucle infinito o una parada prematura). Es la misma forma de trabajar de quien depura un programa.

Palabras clave y enlaces: Grafo, búsqueda en profundidad, backtracking, pseudocódigo · es.wikipedia.org/wiki/Búsqueda_en_profundidad · es.wikipedia.org/wiki/Pseudocódigo

6 El mapa del tesoro

2019-VN-04 · Vietnam (versión de Suiza)
Dificultad: Media

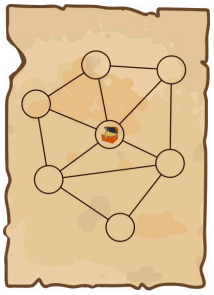
El rey de los castores gobierna siete provincias, cuyas fronteras se muestran en el mapa. En una de ellas ha escondido su tesoro:



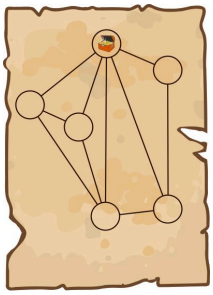
El rey encargó un mapa en el que las provincias se representan con círculos y marcó la provincia del tesoro. Dos círculos están unidos por una línea si y solo si las provincias correspondientes tienen una frontera común. Para despistar a los ladrones, el rey encargó además cuatro mapas del tesoro falsos.

¿Cuál es el mapa del tesoro correcto?

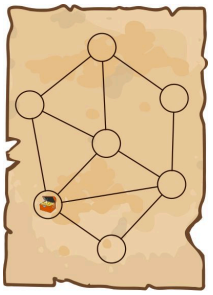
A)



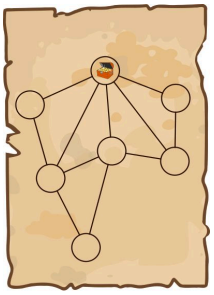
B)



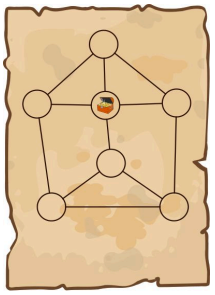
C)



D)



E)

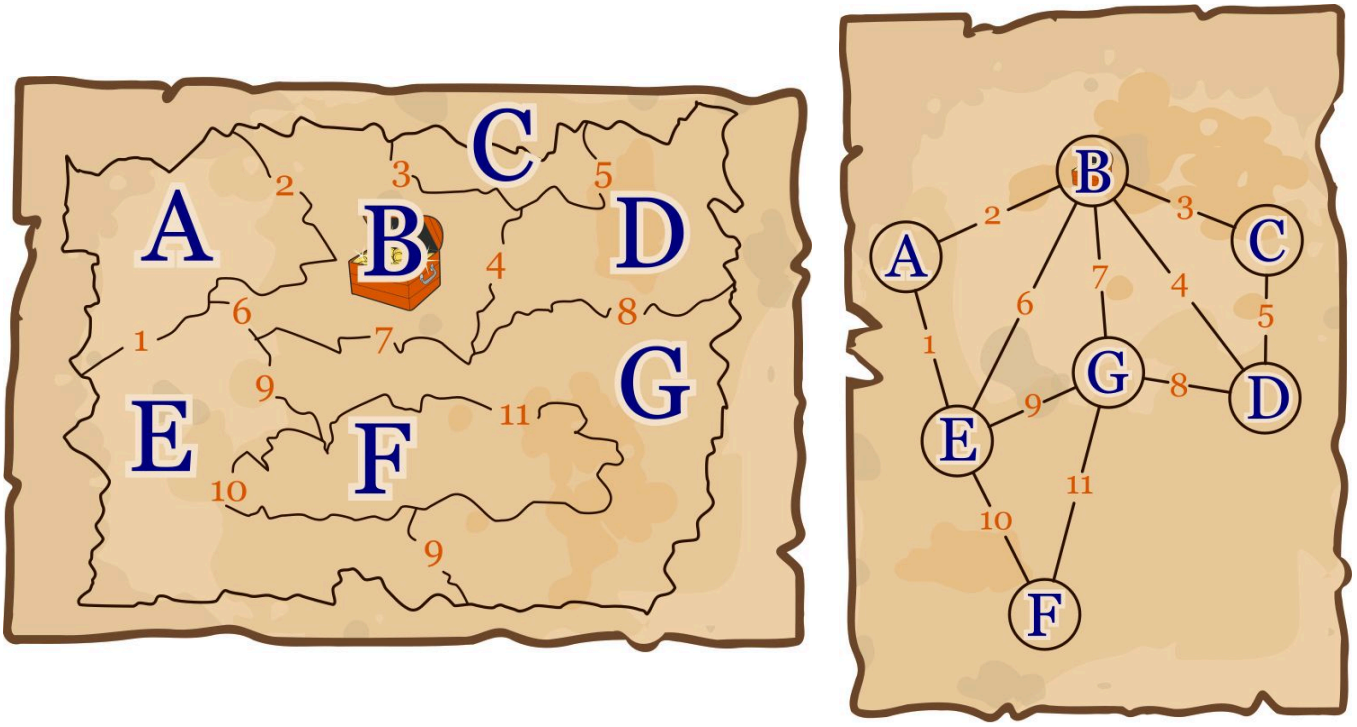


Respuesta correcta: D.

Explicación

La respuesta correcta es la **D**.

En la siguiente imagen las provincias se han marcado con las letras A–G y las fronteras con los números 1–11. Entre las provincias E y G hay dos tramos de frontera, pero como solo importa si tienen frontera común, ambos llevan el número 9. Con estas marcas se comprueba fácilmente que D es un mapa correcto.



A es incorrecta: las provincias A, C y F solo tienen frontera con otras dos, así que en el mapa deberían aparecer tres círculos de los que salgan dos líneas; pero solo hay uno.

B no puede ser correcta porque solo tiene seis círculos, y hay siete provincias.

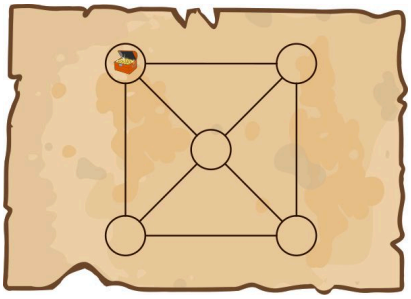
C también es incorrecta: la provincia del tesoro es vecina de cinco provincias (A, C, D, E y G), así que del círculo marcado deberían salir cinco líneas, y solo salen cuatro.

E es la que más se parece al mapa, pero no todas las fronteras están bien dibujadas: por ejemplo, a la provincia B, la del tesoro, le falta una.

Esto es informática

La tarea consiste en representar un mapa mediante un **grafo**. El grafo es una abstracción del mapa (igual que el mapa ya es una abstracción de la realidad). Como en toda abstracción, se pierde información irrelevante; aquí, la posición geográfica de las provincias: aunque B y D están a la misma altura en el mapa, en el grafo aparecen a alturas distintas. Lo relevante, qué provincias son vecinas, se conserva.

Pero una abstracción demasiado agresiva puede perder información que parecía irrelevante: el siguiente grafo es simétrico respecto a giros, así que con él no se sabría dónde está el tesoro.



Un grafo está formado por **nodos** (aquí, los círculos) y **aristas** (las líneas). En este grafo cada arista une dos nodos y no se permiten varias aristas entre los mismos dos nodos, aunque habría tenido sentido para E y G. Los grafos se usan constantemente en informática para abstraer información; muchas veces basta con reducir un problema a un problema de grafos conocido para aplicar métodos de solución que ya existen.

Esto es pensamiento computacional

La tarea trabaja la **abstracción**: pasar de un mapa con formas y posiciones a una estructura que solo recoge las relaciones de vecindad. Para descartar opciones de forma eficiente conviene buscar propiedades sencillas que se puedan contar (número de nodos, número de líneas que salen de cada círculo) en lugar de comparar los dibujos a ojo: es **evaluación** mediante invariantes.

Palabras clave y enlaces: Grafo, teoría de grafos, abstracción · es.wikipedia.org/wiki/Grafo

7 Amigos

2024-LT-05 · Lituania
Dificultad: Fácil

En una clase hay siete castores: Alex, Bess, Cora, Dave, Eric, Fred y Gigi. Algunos se conocen y otros no.



En la tabla hay una marca entre dos castores si se conocen.

	Alex	Bess	Cora	Dave	Eric	Fred	Gigi
Alex		✓	✓				
Bess	✓						✓
Cora	✓				✓		
Dave						✓	
Eric			✓				
Fred				✓			
Gigi		✓					

Cada castor recibió un mensaje distinto de alguien de fuera de la clase y lo compartió con todos los castores de la clase a los que conoce. Cada vez que un castor recibe un mensaje nuevo de alguien de la clase, lo comparte inmediatamente con todos los castores de la clase a los que conoce.

¿Qué castor o castores de la clase recibieron menos mensajes en total?

A) Alex
B) Bess
C) Cora
D) Dave
E) Eric
F) Fred
G) Gigi

Puedes seleccionar más de una respuesta.

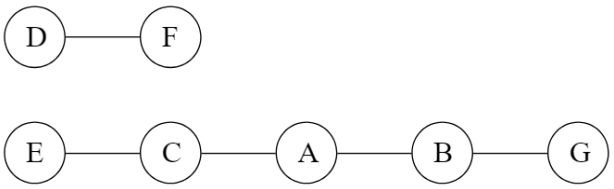
Respuesta correcta: Dave y Fred.

Explicación

La respuesta correcta es **Dave y Fred**.

Podemos dibujar un diagrama de las relaciones: cada castor es un círculo y, si dos castores se conocen, sus círculos se unen con una línea.

Dave solo conoce a Fred y Fred solo conoce a Dave, así que los unimos y no los conectamos con nadie más. Eric solo conoce a Cora; Cora conoce también a Alex; Alex conoce también a Bess y Bess conoce también a Gigi.



En el diagrama se ve que Dave y Fred solo pueden recibir 2 mensajes cada uno, mientras que Eric, Cora, Alex, Bess y Gigi reciben 5 cada uno. Por tanto, Dave y Fred son los que menos mensajes reciben.

Esto es informática

Los grafos son una herramienta de abstracción fundamental en informática. En esta tarea el grafo se presenta como una **matriz de adyacencia**: una matriz cuadrada cuyos elementos indican si cada par de vértices está conectado. El grafo es **no dirigido** (todas las aristas valen en los dos sentidos), y por eso la matriz es simétrica.

El problema está relacionado con la **conexidad** de un grafo: un mensaje llega exactamente a los castores de la misma **componente conexa**. La matriz de adyacencia es una forma habitual de representar grafos en los programas; la principal alternativa es la **lista de adyacencia**.

Esto es pensamiento computacional

Hay que encontrar una forma de **modelar** las conexiones de los datos: pasar de la tabla a un dibujo de red (cambio de representación) hace evidente la estructura. También hay que **generalizar**: darse cuenta de que el número de mensajes que recibe un castor coincide con el tamaño de su grupo conectado, sin necesidad de simular la difusión de cada mensaje.

Palabras clave y enlaces: Grafo, matriz de adyacencia, componente conexa · es.wikipedia.org/wiki/Matriz_de_adyacencia · es.wikipedia.org/wiki/Conectividad_(teoría_de_grafos)

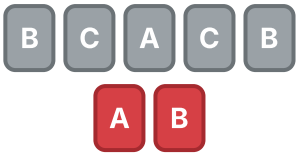
8 La puntuación más alta

2024-SK-04 · Eslovaquia
Dificultad: Fácil

En un juego se modifican secuencias de cartas con las letras A, B y C. El objetivo es conseguir la mayor puntuación posible sustituyendo algunas letras.

Por cada grupo de 2 letras iguales seguidas se obtienen 2 puntos; por cada grupo de 3 letras iguales seguidas, 3 puntos; y así sucesivamente.

Por ejemplo, esta secuencia de cinco cartas grises vale 0 puntos, porque no hay letras iguales juntas. Pero hay dos cartas rojas disponibles para mejorar la puntuación:

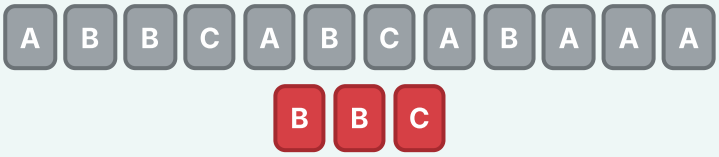


Podríamos tapar la primera C con la carta roja A y la segunda C con la carta roja B:



Ahora la puntuación es 2 + 2 = 4 puntos, por el grupo de dos A y el grupo de dos B. La primera B queda sola y no suma.

Tienes esta secuencia y tres cartas extra. Coloca las cartas rojas B, B y C sobre tres letras para conseguir la secuencia con la mayor puntuación posible.



Escribe encima de cada carta tapada la letra roja que pondrías.

Respuesta correcta: A B B C **C** B **B B** B A A A → 2 + 2 + 4 + 3 = 11 puntos.

Explicación

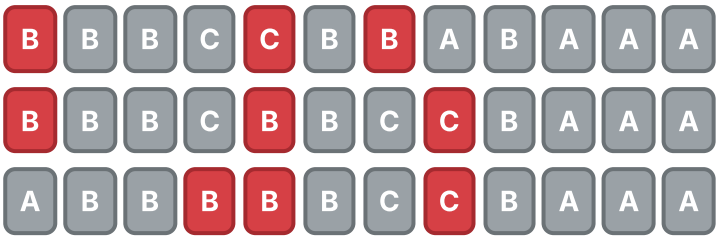
Respuesta correcta:



Esta secuencia vale 2 + 2 + 4 + 3 = **11 puntos**. No se puede hacer mejor, por estas razones:

- La A del extremo izquierdo no puede formar parte de ningún grupo, porque no hay cartas rojas con A.
- Todas las demás letras forman parte de un grupo y, por tanto, puntúan.
- El tamaño del grupo no importa: cada letra de un grupo aporta exactamente 1 punto.

Para comprobar que es la única forma de conseguir 11 puntos, se pueden repasar sistemáticamente las demás posibilidades: se colocan las tres cartas rojas de modo que cada una cree un grupo nuevo sin estropear uno existente. Salen estas secuencias:



Todas valen solo 10 puntos.

Esto es informática

Desde el punto de vista informático, es un **problema de optimización**: conseguir que algo sea lo más eficaz posible dentro de unas restricciones, es decir, encontrar la mejor de todas las soluciones posibles.

Por ejemplo, un compilador optimiza el código de un programa para que sea más rápido y ocupe menos sin cambiar su resultado. La compresión de datos también es una optimización: codificar la información con menos bits que la representación original para ahorrar espacio o transmitir menos datos.

En esta tarea colocamos las letras B, B y C para alargar un grupo de letras iguales o crear uno nuevo; calculamos la puntuación de cada secuencia y elegimos la mejor. Como en informática, buscamos la «mejor» solución dentro de las restricciones fijadas.

Esto es pensamiento computacional

Se usa la **descomposición**: dividir el problema en partes más manejables. Primero se identifican las secuencias que pueden ser relevantes, después se calcula la puntuación de cada una y, por último, se **evalúan** y se elige la mejor. La evaluación también sirve para decidir qué secuencias merece la pena considerar. Darse cuenta de que cada letra agrupada vale exactamente un punto (**generalización**) permite acotar el máximo sin probarlo todo.

Palabras clave y enlaces: Optimización, secuencias, búsqueda exhaustiva · es.wikipedia.org/wiki/Optimización_(matemática)

9 En busca del tesoro

2024-HU-03 · Hungría
Dificultad: — (fácil en 14–16)



El capitán Boris busca un tesoro escondido en una isla.

Tiene un mapa que muestra dónde está el tesoro. El mapa está dividido en 16 regiones, marcadas con las letras de la A a la P. El capitán puede introducir en un aparato especial cualquier cantidad de regiones, y el aparato le dice si el tesoro está o no en alguna de ellas (por ejemplo, si el aparato responde «sí» para A + C + D, el tesoro está en la región A, en la C o en la D).



¿Cuál es el menor número de veces que el capitán Boris tendría que usar el aparato para estar seguro de encontrar siempre el tesoro?

Respuesta: _____

Respuesta correcta: 4.

Explicación

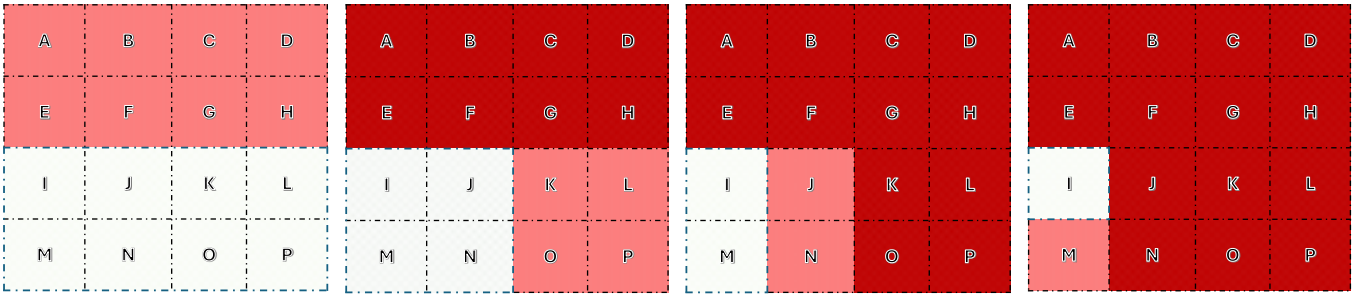
La respuesta es **4**.

Primero, el capitán debe introducir la mitad de las regiones. Puede ocurrir:

a. Que el aparato diga que el tesoro está en esas regiones: entonces divide esas regiones en dos mitades e introduce una de ellas.
b. Que el aparato diga que no está: entonces divide las regiones restantes en dos mitades e introduce una de ellas.

Repite el proceso hasta encontrar la región del tesoro. Cada consulta reduce a la mitad las posibilidades: 16 → 8 → 4 → 2 → 1, es decir, 4 consultas. Con menos no basta: con 3 respuestas sí/no solo se distinguen $2^3 = 8$ casos, y hay 16 regiones.

Por ejemplo, si introduce A, B, C, D, E, F, G, H y responde «no», puede introducir K, L, O, P (valdrían cualesquiera 4 de las restantes). Si vuelve a responder «no», introduce 2 de las que quedan (por ejemplo J y N). Si responde «no», el tesoro está en I o en M. Si introduce M y responde «no», sabe con seguridad que está en I.



Esto es informática

Para encontrar el tesoro de la forma más eficiente hay que dividir las regiones por la mitad. En informática se usa una técnica parecida para encontrar un elemento en una lista ordenada: dividir repetidamente por la mitad el espacio de búsqueda. Así se descarta la mitad de los elementos en cada paso y se encuentra el elemento en el menor número de pasos. Es la **búsqueda binaria**, que en una lista de n elementos necesita unos $\log_2 n$ pasos.

La búsqueda binaria tiene muchas aplicaciones: buscar una palabra en un diccionario, localizar un registro en una base de datos, encontrar en qué versión de un programa apareció un error (como hace `git bisect`), o decidir con el menor número de preguntas sí/no, como en el juego de «¿Quién es quién?».

Esto es pensamiento computacional

Es una tarea de **descomposición**: dividir el espacio de búsqueda en mitades para reducir el problema a otro más pequeño del mismo tipo (estrategia de «divide y vencerás»). Además hay que **evaluar** la estrategia en el peor caso: no basta con que funcione si hay suerte, debe garantizar el resultado siempre, y hay que argumentar que no existe una estrategia con menos consultas.

Palabras clave y enlaces: Búsqueda binaria, divide y vencerás, logaritmo · es.wikipedia.org/wiki/Búsqueda_binaria

10 Días soleados

2024-DE-04b · Alemania
Dificultad: Fácil



El castor Tom dice: «Los días soleados, en todos los estanques hay al menos un castor nadando».

Su hermana Kim responde: «¡Eso no es verdad! El domingo pasado es un ejemplo que demuestra que te equivocas».

Supón que Kim tiene razón. Completa la frase:
El domingo pasado _____①_____ y _____②_____.

Hueco ①

a) hizo sol
b) no hizo sol

Hueco ②

a) todos los estanques estaban llenos de castores nadando
b) ningún castor nadó en el estanque de la cascada helada
c) el castor Miguel nadó por todos los estanques
d) el castor Miguel no nadó nada

Respuesta correcta: ① hizo sol · ② ningún castor nadó en el estanque de la cascada helada.

Explicación

Primer hueco: «hizo sol». Tom hace una afirmación sobre los días soleados. Para demostrar que se equivoca hay que mostrar un día soleado que no cumpla lo que dice. Tom no dice nada sobre los días que no son soleados: su afirmación podría ser cierta pase lo que pase esos días.

Segundo hueco: «ningún castor nadó en el estanque de la cascada helada». En ese caso no todos los estanques tienen un castor nadando, porque el de la cascada helada no lo tiene. Por tanto, Tom se equivoca al decir que en todos los estanques hay al menos un castor nadando.

Las demás opciones no demuestran que Tom esté equivocado:

- «Hizo sol y todos los estanques estaban llenos de castores nadando»: en todos los estanques había castores, lo que coincide con lo que dice Tom.
- «Hizo sol y el castor Miguel nadó por todos los estanques»: en todos los estanques había al menos un castor (Miguel), de nuevo de acuerdo con Tom.
- «Hizo sol y el castor Miguel no nadó nada»: puede que otros castores nadaran en todos los estanques, y entonces la afirmación de Tom se seguiría cumpliendo.

Esto es informática

Tom y Kim discuten sobre el valor de verdad de la afirmación de Tom. Entran así en el terreno de la **lógica booleana**, en la que cada afirmación es verdadera o falsa. La lógica permite construir afirmaciones complejas con operadores como Y, O, NO y con **cuantificadores** como «para todo» y «existe al menos uno».

Negar una afirmación con «para todo» da una con «existe»: la negación de «todos los días soleados, todos los estanques tienen un castor» es «existe un día soleado y existe un estanque sin castores». La lógica ordena las discusiones y ayuda a evitar malentendidos. Además es la base de los ordenadores: los circuitos están hechos de puertas lógicas como NO e Y, y los lenguajes de programación incluyen condiciones lógicas como «SI es verano Y hace sol, ENTONCES enciende el aire acondicionado».

Esto es pensamiento computacional

Como los ordenadores se basan en la lógica, trabajar con ellos exige manejarla: traducir una situación del mundo real a una condición lógica, o encontrar la negación exacta de una condición para que un programa contemple todos los casos posibles. Esta tarea entrena justamente eso, dentro del **análisis e interpretación de datos**. También hay algo de **descomposición**, porque conviene analizar cada frase por separado.

Palabras clave y enlaces: Lógica, cuantificador, negación · es.wikipedia.org/wiki/Cuantificador · es.wikipedia.org/wiki/Lógica_de_primer_orden

11 La máquina de globos

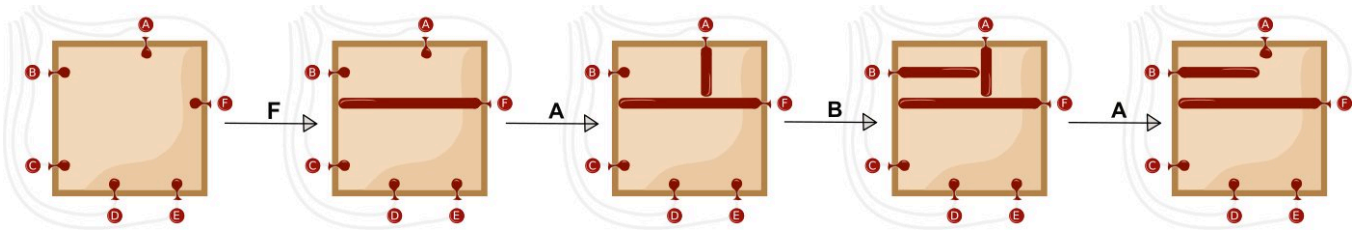
2024-DE-06a · Alemania
Dificultad: Difícil

Los castores tienen una máquina que crea imágenes inflando globos dentro de un marco cuadrado. Los globos están marcados con las letras A, B, C, D, E y F.

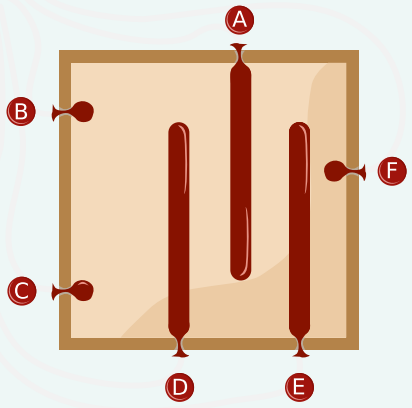
La máquina lee letras de una en una. Cuando lee una letra:

- Si el globo con esa letra está desinflado, lo infla hasta que toca otro globo o el borde opuesto del marco.
- Si no, desinfla el globo con esa letra.

Por ejemplo, si al principio todos los globos están desinflados y la máquina lee F, A, B y después A, hace lo siguiente:



Al principio todos los globos están desinflados y la máquina lee una secuencia de nueve letras. El resultado es la imagen de abajo. Escribe esa secuencia de nueve letras.



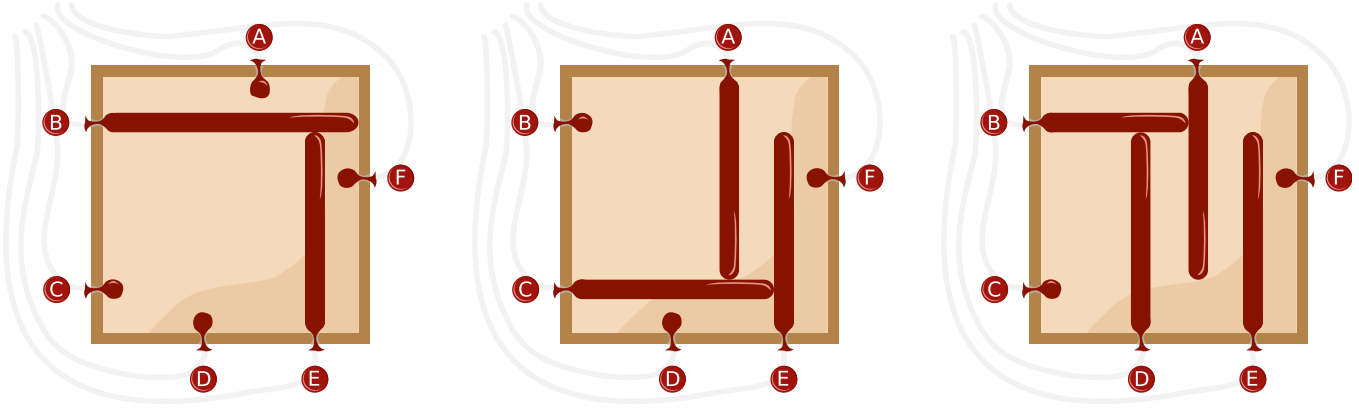
Respuesta correcta: Por ejemplo, B E B C A C B D B (hay 4 respuestas válidas).

Explicación

Hay **4 respuestas correctas** de nueve letras:

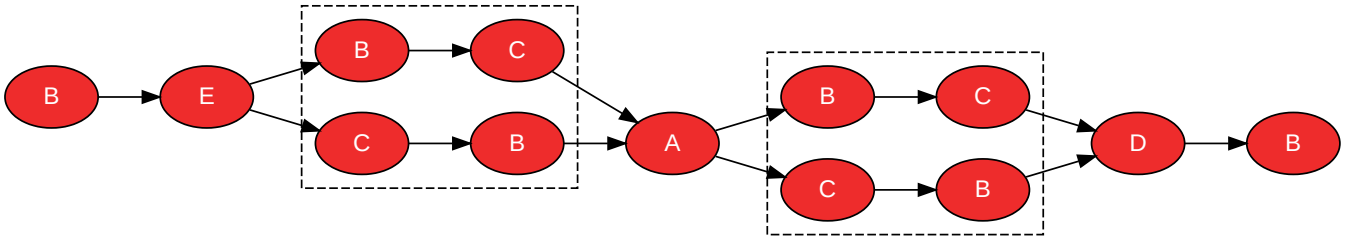
B E B C A C B D B · B E C B A C B D B · B E B C A B C D B · B E C B A B C D B

Explicamos en detalle la primera, B E B C A C B D B. Las imágenes muestran estados intermedios:



- Primera imagen: estado después de B E. Hay que inflar B antes que E para que B sirva de tope al inflar E.
- Segunda imagen: después de B E B C A. El globo C sirve de tope para A, así que debe inflarse antes que A. Además, B debe desinflarse antes de inflar A; si no, A no llegaría a la longitud necesaria.
- Tercera imagen: después de B E B C A C B D. B sirve de tope para D, y C debe desinflarse antes de inflar D.

Para encontrar todas las respuestas ayuda representar el problema con un **grafo dirigido**: una flecha de B a E indica que B debe inflarse antes que E, etc. Hay dos lugares en los que el orden de B y C puede variar (marcados con un rectángulo discontinuo).



Cada respuesta correcta es una **ordenación topológica** de este grafo; como hay 4, hay cuatro soluciones. Un truco para comprobar una respuesta: los globos que acaban inflados (A, D, E) deben aparecer un número impar de veces, y los que acaban desinflados (B, C, F), un número par (incluido 0).

Esto es informática

La secuencia de letras es un **programa** que controla una máquina. Cada letra es una instrucción que infla o desinfla un globo. Como en casi todos los programas de este tipo, el **orden de las instrucciones** es esencial: la secuencia B E produce una imagen distinta de E B.

El grafo dirigido de la explicación no tiene ciclos, por eso se llama **grafo acíclico dirigido** (DAG). Todo DAG tiene al menos una ordenación topológica y se conocen algoritmos rápidos para construirla. Los DAG tienen muchas aplicaciones en informática, por ejemplo en la planificación de tareas o en la compilación de programas (qué hay que compilar antes de qué).

Esto es pensamiento computacional

Hace falta **pensamiento algorítmico**: crear un algoritmo, un conjunto de instrucciones para la máquina, en el que el orden es esencial. El **reconocimiento de patrones** ayuda a deducir cuántas veces debe aparecer cada letra. Y es importante **evaluar** la solución: se puede ejecutar el algoritmo mentalmente, pero también usar atajos para descartar secuencias que nunca pueden ser correctas (por ejemplo, contando cuántas veces aparece cada letra). Si el recuento es correcto, todavía hay que comprobar que el resultado es el pedido.

Palabras clave y enlaces: Programación, secuencia, grafo acíclico dirigido, ordenación topológica · es.wikipedia.org/wiki/Ordenamiento_topológico · es.wikipedia.org/wiki/Grafo_acíclico_dirigido

12 El muro de ladrillos

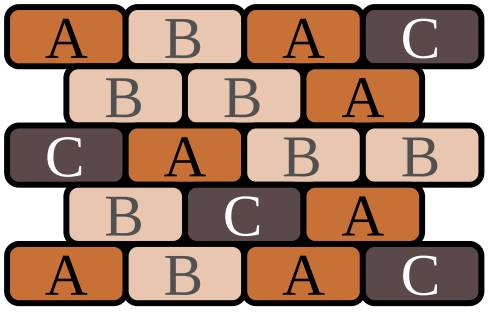
2024-FI-03 · Finlandia
Dificultad: Media

Bob construyó un muro con tres tipos de ladrillos: A, B y C.

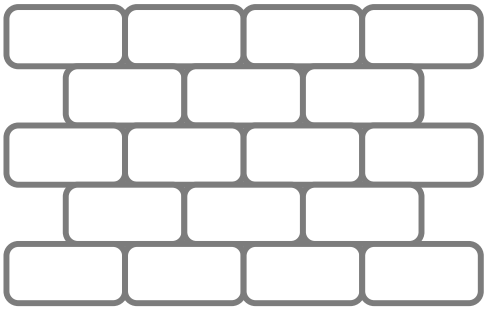
Por desgracia, lo construyó en el sitio equivocado y tiene que trasladarlo. Lo hace ladrillo a ladrillo, así:

- Quita del muro original un ladrillo que no tenga ningún ladrillo justo encima.
- Coloca ese ladrillo en el muro nuevo, en el suelo o encima de otro ladrillo, pero nunca debajo de ladrillos ya colocados.

Mientras siga estas reglas, a Bob le da igual en qué orden traslada los distintos tipos de ladrillo del muro original al nuevo.



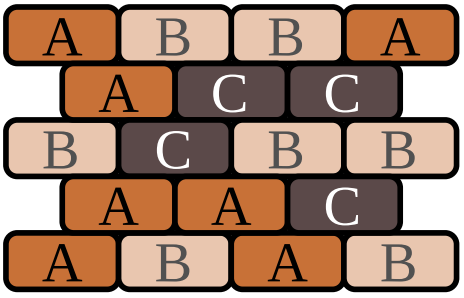
muro original



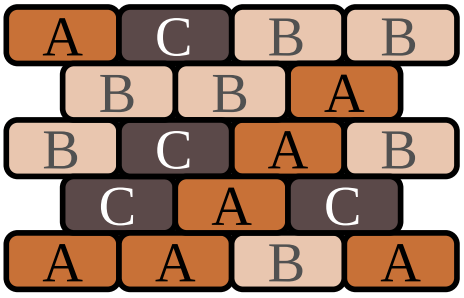
muro nuevo

De las siguientes opciones, selecciona todos los muros que Bob puede construir.

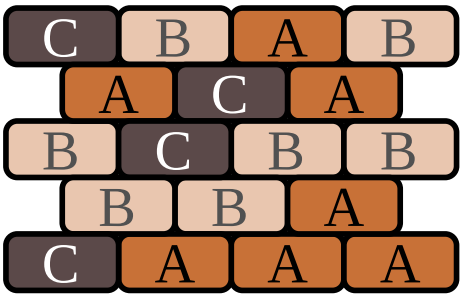
A)



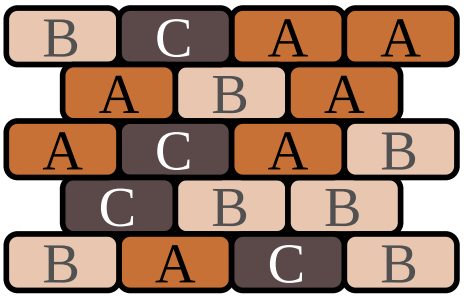
B)



C)



D)

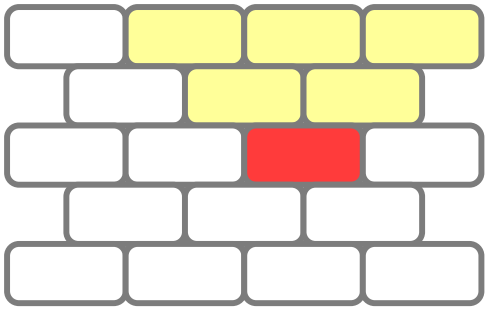


Respuesta correcta: A y C.

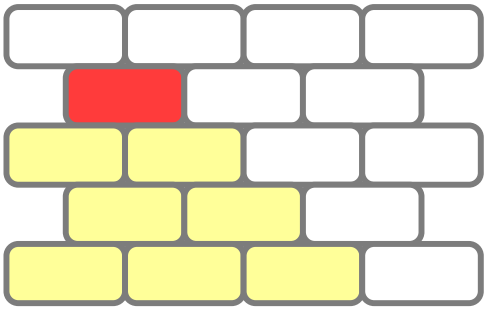
Explicación

Solo son posibles los muros **A** y **C**.

A la izquierda se ve un ladrillo del muro original marcado en rojo. Es el vértice de un triángulo invertido que llega hasta la fila de arriba: los ladrillos amarillos son exactamente los que hay que quitar antes de poder quitar el rojo. A la derecha, el ladrillo rojo del muro nuevo es el vértice de un triángulo hacia arriba (cortado por el lateral): hay que colocar todos los amarillos antes que el rojo.

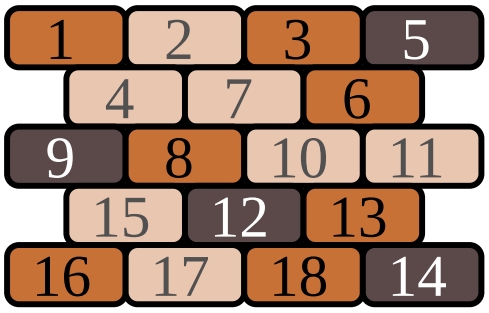


muro original

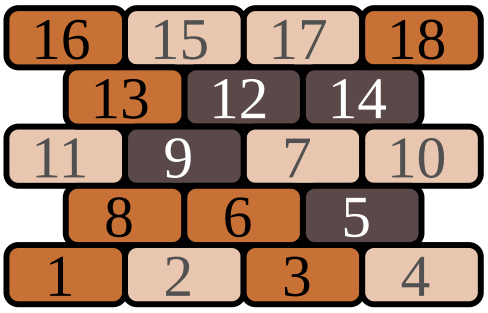


muro nuevo

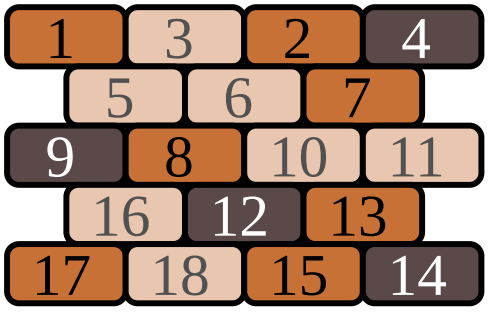
Con esta idea de los triángulos se comprueba que A y C son posibles. Abajo se muestra una construcción de cada uno (hay muchas otras); los números indican el orden en que se traslada cada ladrillo. Las reglas se cumplen porque: 1) en el muro original, todos los ladrillos del triángulo invertido de cada ladrillo tienen un número menor; 2) en el muro nuevo, todos los ladrillos del triángulo hacia arriba de cada ladrillo tienen un número menor.



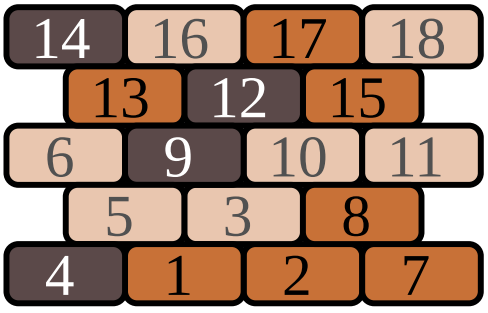
construcción de A: original



→ muro A



construcción de C: original



→ muro C

B es imposible. Pensemos en el momento en que se va a quitar del muro original el 4.º ladrillo B. En ese momento ninguno de los ladrillos de las dos filas inferiores está disponible, así que como mucho se han trasladado los 4 ladrillos A y los 2 ladrillos C de las tres filas superiores. Pero para colocar ese 4.º ladrillo B

en el muro B hacen falta antes al menos 5 ladrillos A o al menos 3 ladrillos C en el muro nuevo: contradicción.

D es imposible. Cuando se va a quitar el 3.er ladrillo B del original, ya se han quitado al menos 2 ladrillos A. El 2.º de ellos tiene que ir al extremo izquierdo de la fila central del muro D (la otra A de esa fila exigiría haber colocado ya el 3.er B). Ahora imaginemos que se acaba de colocar el 3.er ladrillo A: tiene que ser la segunda A de la tercera fila (la alternativa, el ladrillo izquierdo de la fila superior, exige haber trasladado 3 ladrillos C, lo que no puede ocurrir hasta haber quitado 4 ladrillos A). Así, el 2.º y el 3.er A van en la fila central, por lo que antes tiene que haber 4 ladrillos B en las dos filas inferiores. Pero solo se pueden quitar 4 ladrillos B del original después de haber quitado al menos 3 ladrillos A. Esta contradicción demuestra que D es imposible.

Esto es informática

La tarea trata de analizar **dependencias** (restricciones de precedencia) entre objetos, un problema clásico en informática. Las dependencias entre objetos (aquí, los ladrillos) se expresan normalmente con un **grafo de dependencias**. Un orden de procesamiento válido (aquí, el orden en que se pueden quitar y colocar los ladrillos) corresponde a una **ordenación topológica** de los objetos.

Este tipo de análisis aparece, por ejemplo, al decidir en qué orden instalar paquetes de software que dependen unos de otros, o en qué orden ejecutar las tareas de un proyecto.

Esto es pensamiento computacional

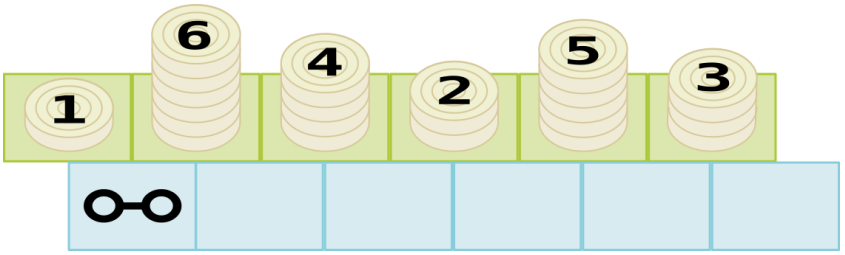
La tarea combina dos habilidades importantes: seguir reglas de forma sistemática para demostrar que en algunos casos existe solución, y construir con cuidado un **argumento lógico** para demostrar que en otros no existe; esto último suele ser más difícil. En la explicación, las dos reglas se enfrentan entre sí para llegar a una contradicción. Otra forma de abordarlo es analizar un **árbol de decisiones**, probando las consecuencias de elecciones que respetan las reglas hasta que todas las ramas resultan callejones sin salida: es más difícil convertirlo en una explicación breve, pero es el punto de partida natural, porque es justo lo que permite encontrar soluciones cuando existen.

Palabras clave y enlaces: Grafo de dependencias, ordenación topológica · es.wikipedia.org/wiki/Ordenamiento_topológico

13 Cambia y avanza

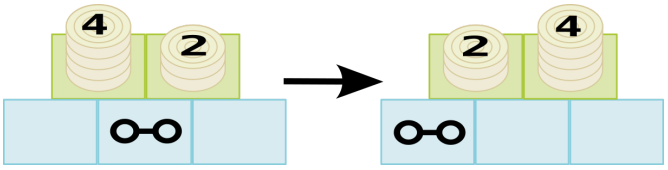
2024-BE-01b · Bélgica
Dificultad: Media

En un tablero hay dos filas de 6 casillas, como en la imagen. En las casillas de la fila de arriba hay pilas de discos, todas de distinta altura (1, 6, 4, 2, 5, 3). En la fila de abajo hay un marcador que señala dos casillas de la fila de arriba.

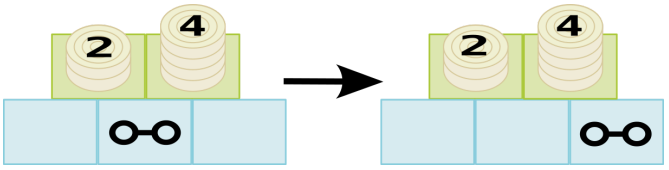


A partir de esta configuración se repiten los siguientes movimientos:

1. Si la casilla izquierda del marcador es más alta que la derecha, se intercambian las dos casillas y se mueve el marcador a la izquierda, si es posible.
2. Si la casilla izquierda del marcador es más baja que la derecha, no se intercambian y se mueve el marcador a la derecha.



movimiento 1

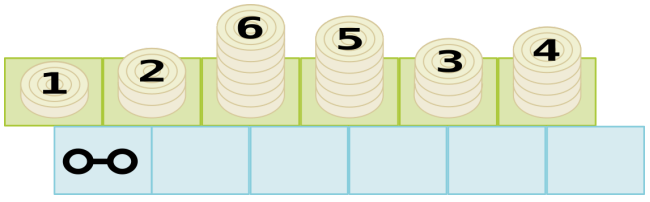


movimiento 2

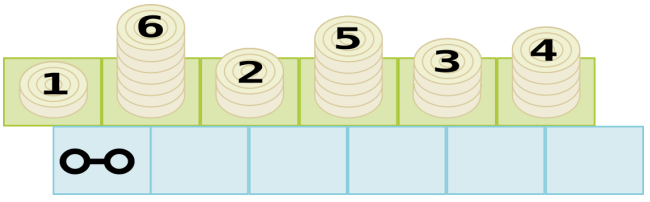
Se dejan de repetir los movimientos cuando el marcador llega a la casilla de la derecha del todo.

Estas son cuatro configuraciones iniciales distintas. ¿Con cuál llega el marcador a la casilla de la derecha del todo en el menor número de movimientos?

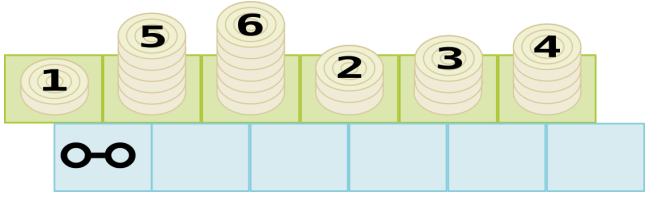
A)



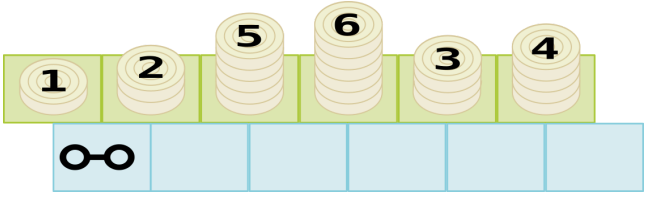
B)



C)



D)



Respuesta correcta: D.

Explicación

La respuesta correcta es la **D**.

No hace falta ejecutar todos los movimientos de las cuatro opciones y contarlos: se pueden comparar las opciones entre sí sin contarlos todos.

Para simplificar, representamos cada pila por su número de discos y la posición del marcador con un punto ●:

Opción A	1 ● 2 6 5 3 4	Opción C	1 ● 5 6 2 3 4
Opción B	1 ● 6 2 5 3 4	Opción D	1 ● 2 5 6 3 4

En la opción B, el marcador primero se mueve a la derecha (movimiento 2) y después intercambia y se mueve a la izquierda (movimiento 1):

1 ● 6 2 5 3 4 → 1 6 ● 2 5 3 4 → 1 ● 2 6 5 3 4

Tras 2 movimientos, la configuración es exactamente la de A. Por tanto, B tarda más que A.

Los primeros movimientos de A son:

1 ● 2 6 5 3 4 → 1 2 ● 6 5 3 4 → 1 2 6 ● 5 3 4 → 1 2 ● 5 6 3 4

y a esta configuración llega D tras un solo movimiento. Por tanto, A tarda más que D.

Los primeros movimientos de C son:

1 ● 5 6 2 3 4 → 1 5 ● 6 2 3 4 → 1 5 6 ● 2 3 4 → 1 5 ● 2 6 3 4 → 1 ● 2 5 6 3 4

que es la configuración inicial de D. Por tanto, D es la que necesita menos movimientos.

Esto es informática

Todas las opciones acaban en 1 2 3 4 5 6 ●: lo que hace este procedimiento es **ordenar** las pilas por altura.

En informática es muy frecuente tener que ordenar datos (de menor a mayor, de más barato a más caro...). Una razón es que buscar en una secuencia ordenada es muchísimo más rápido que en una desordenada.

El método de la tarea se llama **ordenamiento gnomo** (*gnome sort*). Es fácil de entender, de ejecutar y de programar, pero no se usa en la práctica porque es mucho más lento que algoritmos más elaborados como *quicksort* o la ordenación por mezcla (*merge sort*).

Esto es pensamiento computacional

Hay que comprender y ejecutar un **algoritmo**, pero el paso inteligente es darse cuenta de que no hace falta ejecutarlo entero para responder: basta con **reconocer** que una configuración se convierte en otra tras unos pocos pasos, de modo que el número de movimientos de una opción incluye el de otra. Comparar en lugar de contar es una forma de **abstracción** que ahorra mucho trabajo.

Palabras clave y enlaces: Ordenación, ordenamiento gnomo · es.wikipedia.org/wiki/Ordenamiento_gnomo

14 Cadenas de palabras

2024-CA-02 · Canadá
Dificultad: Fácil



El profesor Castor está enseñando a leer a su alumnado. Para ayudarles crea **cadenas de palabras**: secuencias de palabras en las que, para obtener la palabra siguiente, se cambia exactamente una letra. Por ejemplo, MUG → MUD → MAD → FAD es una cadena de palabras.

El profesor Castor tiene estas nueve palabras: BOT, SAD, BAT, CAB, COT, BAD, COB, CAT y SAT. Las agrupa en tres cadenas de tres palabras cada una, de modo que cada una de las nueve palabras aparece en una y solo una de las cadenas.

(Las palabras están en inglés, como en la tarea original; para resolverla solo importan las letras.)

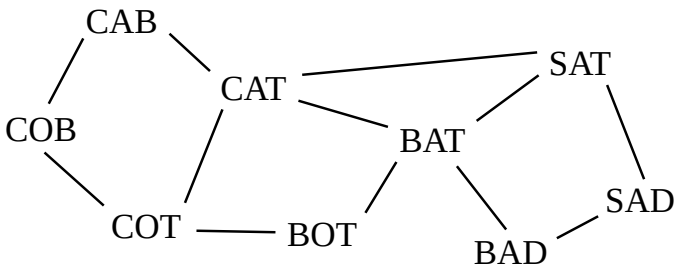
Ninguna de las siguientes secuencias incumple las reglas de las cadenas, pero una de ellas hace imposible formar tres cadenas sin repetir palabras. ¿Cuál no podría ser una de las cadenas del profesor Castor?

A) SAD → BAD → BAT
B) COT → COB → CAB
C) BAT → CAT → COT
D) CAT → CAB → COB

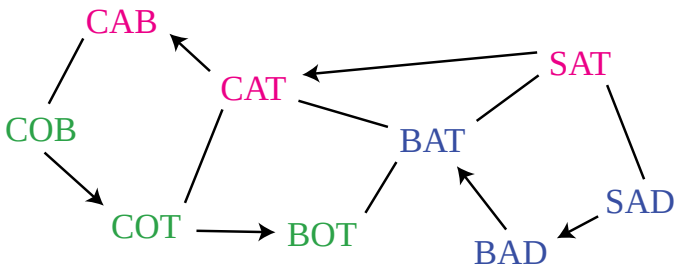
Respuesta correcta: C) BAT → CAT → COT.

Explicación

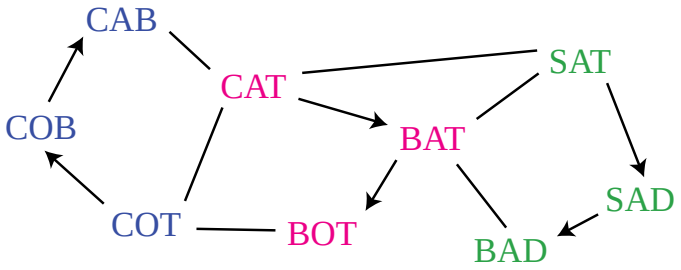
La respuesta es la **C) BAT → CAT → COT**. Para resolverlo ayuda dibujar un diagrama en el que dos palabras se unen con una línea si pueden ir seguidas en una cadena.



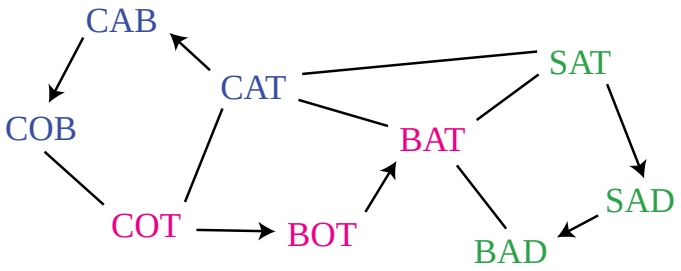
A es posible: las otras dos cadenas podrían ser SAT → CAT → CAB y COB → COT → BOT.



B es posible: las otras dos podrían ser CAT → BAT → BOT y SAT → SAD → BAD.



D es posible: las otras dos podrían ser COT → BOT → BAT y SAT → SAD → BAD.



C no es posible: en el diagrama, BOT solo está unida a BAT y a COT, así que tiene que estar en una cadena con al menos una de ellas. Como BAT y COT están las dos en la cadena de la opción C y BOT no, no queda ninguna cadena posible para BOT.

Esto es informática

En informática, este tipo de diagrama se llama **grafo** y sirve para mostrar conexiones. Está formado por un conjunto de vértices y un conjunto de aristas, cada una de las cuales une dos vértices. Aquí los vértices

son las palabras y hay una arista entre dos de ellas si se diferencian en exactamente una letra.

Una cadena de palabras es un **camino** por aristas conectadas. Como hay un camino entre dos vértices cualesquiera, el grafo es **conexo**. Agrupar las nueve palabras en tres cadenas de tres equivale a quitar aristas hasta obtener un grafo no conexo con tres **componentes** de tres vértices cada una (una partición en caminos). La pregunta consiste en comprobar si un conjunto dado de tres palabras puede ser una de esas componentes.

Esto es pensamiento computacional

Hay que **abstraer** el problema a un grafo para verlo de un vistazo y **evaluar** cada opción: comprobar si, tras usar esa cadena, las seis palabras restantes se pueden repartir en dos cadenas válidas. La clave para descartar rápido es fijarse en las palabras con pocas conexiones (como BOT), que son las que más restringen las soluciones.

Palabras clave y enlaces: Grafo, camino, grafo conexo, componente conexa · es.wikipedia.org/wiki/Grafo_conexo

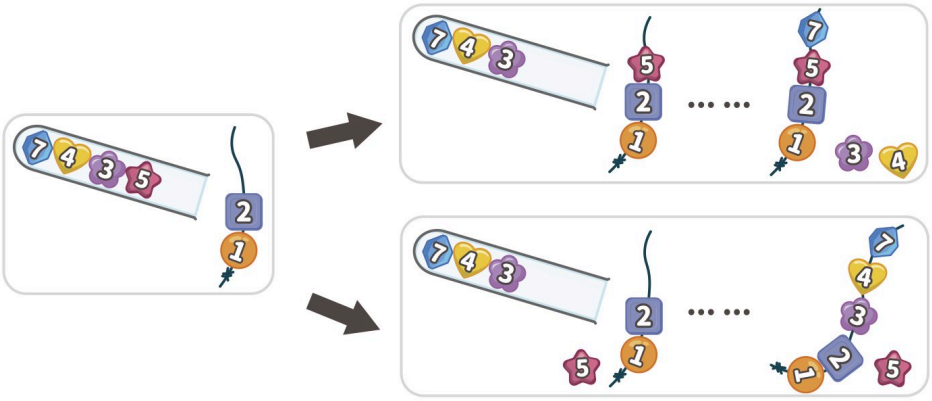
15 La pulsera Bebras

2024-TW-03 · Taiwán
Dificultad: — (fácil en 14–16)

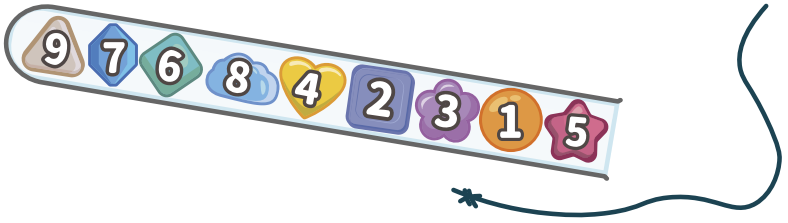
Castor está haciendo una pulsera Bebras. Saca de un tubo cuentas numeradas, de una en una. Con cada cuenta decide si la ensarta en el hilo o la aparta y no la usa. Pero solo puede ensartar una cuenta si:

- el hilo está vacío, o
- el número de la cuenta es mayor que el de la última cuenta ensartada.

En este ejemplo, la última cuenta del hilo es la n.º 2. Así que Castor puede ensartar la cuenta n.º 5 o apartarla.



Castor va a hacer una pulsera nueva con las cuentas de este tubo (la primera que sale es la 5):



¿Cuál es el mayor número de cuentas que puede ensartar?

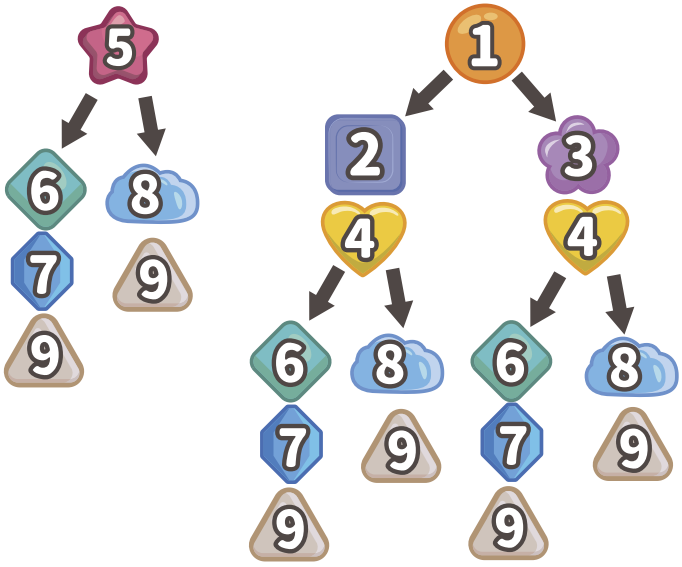
A) 3 cuentas
B) 4 cuentas
C) 5 cuentas
D) 6 cuentas
E) 7 cuentas

Respuesta correcta: D) 6 cuentas.

Explicación

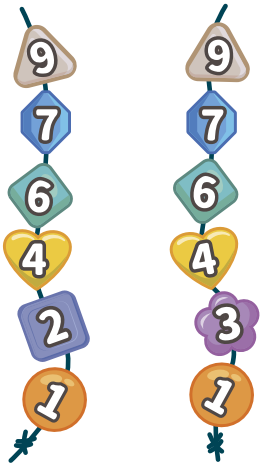
La respuesta correcta es la **D) 6 cuentas**.

Podemos construir todas las pulseras posibles con este tubo y ver cuáles tienen más cuentas. La primera cuenta que sale es la 5. Si se ensarta, después solo se pueden ensartar la 6, la 7, la 8 y la 9, y las secuencias más largas posibles son 5 6 7 9 y 5 8 9. Si se aparta la 5 y se ensarta primero la siguiente (la 1), hay más secuencias y más largas:



Si se empieza por cualquier otra cuenta, la secuencia más larga posible ya está incluida en una de las anteriores, que empiezan por la 1 o por la 5. Por ejemplo, empezar por la 2 lleva a secuencias como 2 4 6 7 9 o 2 4 8 9, incluidas en 1 2 4 6 7 9 y 1 2 4 8 9.

Por tanto, las pulseras con más cuentas empiezan por la 1 y tienen 6 cuentas: **1 2 4 6 7 9** y **1 3 4 6 7 9**.



Esto es informática

En informática, una **secuencia** es una serie ordenada de datos, como una cadena de caracteres o una secuencia de números. Cada miembro es un elemento, y una **subsecuencia** se forma eligiendo elementos de la secuencia original sin cambiar su orden.

Tanto las cuentas del tubo (en el orden en que salen) como las pulseras son secuencias de números si ignoramos la forma y el color. La tarea pide encontrar la **subsecuencia creciente más larga** (*longest increasing subsequence*) de la secuencia del tubo.

Con secuencias mucho más largas, enumerar todas las subsecuencias crecientes llevaría demasiado tiempo. Por suerte, existen algoritmos eficientes que la encuentran en un tiempo aproximadamente proporcional a n·log n: para un millón de elementos, del orden de veinte millones de pasos, nada para un ordenador moderno. Muchos problemas de subsecuencias se resuelven con la técnica de **programación dinámica**.

Esto es pensamiento computacional

Enumerar todas las soluciones posibles llevaría mucho tiempo, y muchas ni siquiera cumplirían los requisitos. Por eso la clave es encontrar una forma eficiente de identificar las soluciones posibles y descartar las imposibles: **pensamiento algorítmico**. Además, al decidir si se ensarta una cuenta hay que valorar cómo limita las cuentas que se podrán usar después; por ejemplo, ensartar la 5 impide usar de la 1 a la 4. Esa **simulación** de las consecuencias de cada decisión es otro aspecto del pensamiento computacional.

Palabras clave y enlaces: Subsecuencia creciente más larga, programación dinámica · en.wikipedia.org/wiki/Longest_increasing_subsequence · es.wikipedia.org/wiki/Programación_dinámica

Créditos y licencia

Las tareas proceden del taller internacional Bebras 2024 (Serbia), salvo «El mapa del tesoro» (2019-VN-04, taller de 2019). Cada tarea indica su código y el país que la propuso. Las imágenes son de los autores y autoras de las tareas originales.

Traducción y adaptación al castellano: Bebras España (bebras.ehu.es).

Copyright © Bebras – International Challenge on Informatics and Computational Thinking. Esta obra está bajo una licencia Creative Commons Reconocimiento-CompartirIgual 4.0 Internacional (CC BY-SA 4.0): creativecommons.org/licenses/by-sa/4.0.