Soluciones a los problemas

OIE 2020

18 de abril 2020

Día 2: Baq y las distancias entre ciudades

Enunciado

Construir un grafo completo con pesos en las aristas que sean una permutación de $\{1, \ldots, \binom{n}{2}\}$ tal que la distancia mínima entre dos vértices sea igual al peso de la arista que los une.

Autor: Félix Moreno Peñarrubia

- Número de soluciones con 100 puntos: 13
- Primera solución con 100 puntos: 0:14, Vicent Baeza Esteve

Día 2: Baq y las distancias entre ciudades

Solución

Entre el vértice i y el vértice j (indizados desde 1), con $i > j$ poner $\frac{(i-2)(i-1)+2j}{2}$ como peso funciona. Comprobamos que funciona para todo triángulo con $i > j > k$:

$$(j-2)(j-1)+2k+(i-2)(i-1)+2k \geq (i-2)(i-1)+(j^2-3j+2)+4k \geq$$
$$\geq (i-2)(i-1)+j^2-3j+6 \geq (i-2)(i-1)+2j$$

(En la última desigualdad se usa $j \geq 2$)

Motivación

Esta solución se puede encontrar tratando de buscar un patrón "inductivo" o "recursivo", ya la solución $n+1$ se construye a partir de la n de forma natural. Además, reetiquetando los vértices, la solución anterior es equivalente a escribir los números $\binom{n}{2}, \ldots, 1$ en orden decreciente según el formato que pedía el problema.

Día 2: Transporte Carísimo

Enunciado

Encontrar la distancia mínima desde un nodo a todos los otros en un grafo con pesos, donde la distancia recorrida en un camino $v_1, \ldots, v_n$ se define como $\sum_{i=1}^{n-1} \sum_{j=1}^{i} w(v_j v_{j+1})$

Autor: Félix Moreno Peñarrubia

- Número de soluciones con 100 puntos: 4
- Número de soluciones con 62 puntos: 1
- Número de soluciones con 25 puntos: 6
- Primera solución con 100 puntos: 0:51, Vicent Baeza Esteve

Día 2: Transporte Carísimo

Solución cúbica

La definición de la distancia se puede pensar como lo siguiente: si queremos encontrar el camino óptimo de longitud L, el peso de la i-ésima arista que visitemos lo ponderamos multiplicando por $(L - i)$. Hay un par de formas de solucionar el problema para una L fija:

- Dijkstra: hacemos L copias de los nodos, cada una para representar que ese nodo es el i-ésimo que visitamos. Entonces podemos calcular los pesos resultantes y aplicar Dijkstra tal cual, con complejidad $\mathcal{O}(L(n+m)\log Lm)$
- Bellman-Ford: La idea del algoritmo de Bellman-Ford es muy apropiada para este problema, ya que la i-ésima iteración que haces con el algoritmo corresponde con el i-ésimo paso que darías en un camino óptimo. Por tanto, puedes simplemente ir aplicando las ponderaciones en cada iteración del algoritmo. Complejidad $\mathcal{O}(Lm)$

Esto da una solución cúbica, ya que hay que probar para todos los valores de L de 1 a n.

Día 2: Transporte Carísimo

Solución cuadrática

La observación que hay que hacer para bajar a complejidad cuadrática es que no hace falta probar todas las L, sino solo $L = n$, ya que los caminos de longitud inferior los puedes pensar como caminos en los que en los primeros pasos no te mueves del primer vértice.

Día 2: Hermanos

Enunciado

Dadas unas piezas de dominó decid si al tirar la primera va a caer la última, y, en caso que se cumpla esta propiedad, si si eliminamos alguna de las piezas se va a seguir cumpliendo.

Autor: Cesc Folch Aldehuelo

- Número de soluciones con 100 puntos: 5
- Número de soluciones con 60 puntos: 4
- Número de soluciones con 35 puntos: 1
- Número de soluciones con 25 puntos: 1
- Primera solución con 100 puntos: 0:28, Oscar Balcells Obeso

Día 2: Hermanos

Solución $\mathcal{O}(n)$

- Recorríamos las piezas en orden y nos guardábamos dos enteros D_0 y D_1 que eran los dos máximos valores de $p_i + h_i$ que habíamos visto hasta el momento, con $D_0 \geq D_1$.
- Inicialmente $D_0 = D_1 = p_0 + h_0$, porque no podemos quitar la primera pieza.
- Si en algún punto $p_i \geq D_0$, la respuesta era "Pep".
- Si esto no pasaba y en algún punto $p_i \geq D_1$, la resupesta era "Ivet"
- En caso contrario la respuesta era "Cesc".

Día 2: Castillos de Jordan

Enunciado

Determinad el mínimo número de bloques que hay que eliminar del castillo para que sea igual visto de izquierda a derecha que de abajo hacia arriba (simétrico respecto a la diagonal principal)

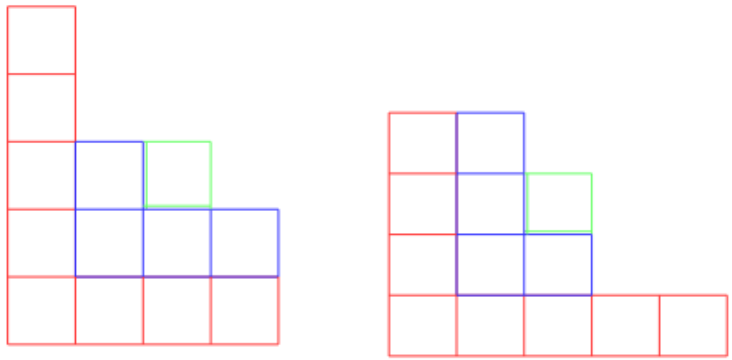
Autor: Jordi G Rodríguez Manso

- Número de soluciones con 100 puntos: 6
- Número de soluciones con 62 puntos: 2
- Número de soluciones con 31 puntos: 5
- Número de soluciones con 19 puntos: 1
- Primera solución con 100 puntos: 1:41, Pau Cantos

Día 2: Castillos de Jordan

Observación

Si hacemos la simetria del castillo respecto a la diagonal principal, el castillo bonito que buscamos es el que resulta de sobreponer el castillo original y el que resulta de hacer la simetria, y quedarse con los bloques que aparecen en los 2.



Solución

Calculamos la secuencia (b_i) que nos definen en el enunciado, que es la secuencia (a_i) del castillo simétrico. El castillo bonito que buscamos tendrá $min(a_i, b_i)$ bloques en la i-ésima columna. La respuesta será

$$\sum_{i=1}^{n} a_i - \sum_{i=1}^{n} min(a_i, b_i)$$

¿Cómo calcular (b_i)? El elemento b_i será el número de indices k tales que $a_k \geq i$. Además como los a_i són decrecientes, basta con determinar el índice k más grande tal que $a_k \geq i$.

Caso 1 ($O(\max(a_i)^n)$)

El caso pequeño se puede resolver generando todos los castillos de 8 columnas con hasta 8 bloques cada una (8^8 en total), y para cada uno comprobar si es bonito y se puede obtener quitando bloques al original. Nos quedaremos con el castillo que más bloques tenga y la respuesta será la diferencia de bloques con el original.

Caso 2 ($O(n^2)$)

Calculamos la secuencia (b_i) como hemos dicho antes. El elemento b_i será el número de indices k tales que $a_k \geq i$. Para cada i podemos iterar sobre todos los a_k.

Casos 3 y 4 ($O(n)$)

Para calcular los b_i no iteramos sobre todos los a_k cada vez. Para calcular b_{i+1}, aprovechamos que sabemos el índice k más grande tal que $a_k \geq i$ (corresponde a b_i). Empezando por ese indice, vamos reduciéndolo hasta encontrar el primero que es mayor o igual a $i + 1$.

De esta manera, no hacemos una iteración sobre los (a_i) para calcular cada b_i, sino que hacemos una iteración para todos.

La única diferencia entre los casos 3 y 4 es que en este último los valores de a_i son mucho mayores (hasta 10^9). Si calculamos toda la secuencia (b_i) completa obtendremos TLE en el último caso ya que la secuencia completa tendrá 10^9 elementos. La observación para conseguir todos los puntos es que solo nos importan los primeros n elementos.

Día 2: Juego en un grafo

Enunciado

A y B juegan a un juego en un grafo con vértices numerados. Uno de los vértices se marca como vértice ganador. Los jugadores se van turnando (empezando por A) y en cada turno el jugador selecciona una componente conexa del grafo y elimina el vértice con menor número de la componente. Gana el primer jugador que elimine el vértice ganador. Encontrar los vértices ganadores para los que gana A.

Autor: Félix Moreno Peñarrubia

- Número de soluciones con 100 puntos: 1
- Primera solución con 100 puntos: 3:42, Javier Nistal Salas

Día 2: Juego en un grafo

Planteamiento

La primera observación que podemos hacer (y que una de las subtareas da como pista) es que cada grafo tiene un árbol (bosque) asociado tal que jugar al juego en ese bosque es equivalente a jugar al juego en el grafo. Este bosque se construye asignando a cada vértice como hijos los nuevos vértices que se pueden escoger cuando se borra este vértice, es decir, los vértices de menor número de las componentes conexas en las que se divide la componente del vértice original. Notemos que el orden en el que se realicen los movimientos en el juego no afecta a cómo queda el grafo. Resolver el problema para el árbol es sencillo (contar paridad de los descendientes de cada nodo), así que una forma de plantear el problema es intentar encontrar el bosque asociado. Simulando directamente se puede hacer en $\mathcal{O}(n^2)$, pero es posible hacerlo mejor.

Día 2: Juego en un grafo

Solución $\mathcal{O}(n\alpha(n))$

Vamos a hacer la simulación del revés: empezamos por los vértices que quedan al final. Observemos que, ya que el bosque satisface que el número del padre es menor que el hijo, el vértice $n-1$ debe ser una hoja. El vértice $n-2$, si es vecino del $n-1$ será su padre y si no será otra hoja. En general, podemos ir generando el árbol de la siguiente forma:

- Iteramos con i desde $n-1$ hasta 0.
  - Para cada componente conexa de vértices ya visitados que sea vecina de i, i será padre del vértice con menor número de la componente.
  - Marcamos i como visitado.

Para hacer esto eficientemente tenemos que ir manteniendo componentes conexas que se van uniendo. Usando la estructura de datos de unión-búsqueda obtenemos una complejidad de $\mathcal{O}(n\alpha(n))$