Proyecto de Diseño y Análisis de Algoritmos

Marcos Antonio Pérez Lorenzo

September 22, 2024

1 Problemas

1.1 A story of One Country

1.1.1 Descripción

Petya decidió visitar Byteland durante el verano. Resultó que el país tiene una historia ciertamente inusual. Inicialmente existían n reinos, cada uno con su territorio demarcado por un rectángulo en el mapa, con paredes paralelas a los ejes y esquinas localizadas en coordenadas enteras. Ningún territorio se superponía con algún vecino, aunque si podían compartir fronteras (es decir, los rectángulos que los delimitan tiene un lado común). A medida que el tiempo pasó, pares de reinos se fusionaron, aunque solo si el país resultante era un rectángulo también, hasta formar el actual país de Byteland.

Cada reino construyó un castillo rectangular dentro de su territorio, con las mismas reglas de emplazamiento que el territorio mismo. Milagrosamente, después de la unificación de Byteland, todos los castillos permanecieron intactos.

Petya se pregunta si la historia del país, tan peculiar, es de hecho cierta. Quiere determinar si, dada las posiciones actuales de los castillos, es posible que Byteland surgiera como la historia cuenta.

1.1.2 Datos

Enteros $0 \leq x_0 \leq x_1 \leq 10^9$ y $0 \leq y_0 \leq y_1 \leq 10^9$ por cada uno de los $1 \leq n \leq 100000$ castillos, donde (x_0, y_0) y (x_1, y_1) son los vertices inferior izquierdo y superior derecho del n-ésimo castillo.

1.1.3 Solución

Como el problema no requiere reconstruir las fronteras, lo cual no sería posible con la información actual, sino comprobar si algún arreglo fronterizo es posible. Asi que basta con hallar un conjunto de fronteras que divide el mapa en rectángulos que contienen exactamente un castillo. Para este fin podemos replantear el problema como: comprobar si es posible una partición del plano tal que cada fragmento contiene exactamente una rectángulo.

Esta problema se puede resolver intentando cortes rectos en el plano. Formalmente, un area del plano esta correctamente particionada si contiene solo un rectángulo, o existe un corte recto del plano que lo divide en dos areas correctamente particionadas. De esta definición de deduce que la solución es: dado un conjunto de rectángulos, encontrar un corte (sea horizontal o vertical) tal que no corte ningún rectángulo, dividir el conjunto en dos subconjuntos y resolver el problema para ambos.

Formalmente:

Proof.

$$R^{(a)} = \left\{ r \in R | \forall i, j \in \mathcal{N} : i \geq j \Rightarrow (R_i^{(a)})_0 \geq (R_j^{(a)})_0 \right\} \tag{1}$$

donde:

- R el conjunto de los rectángulos, y R_i el i-ésimo rectángulo
- $r \in R$, r_i^x es la coordenada x_i del rectángulo r en el eje x

existe un corte si

$$C^{(a)}(k) = \max_{i \leq k} \left\{ (R_i^{(a)})_1 \right\} \leq (R_k^{(a)})_0 \tag{2}$$

una partición de R en k es:

$$I^{(a)}(R, k) = \left\{ r \in R | (R_i^{(a)})_1 \leq (R_k^{(a)})_0 \right\} \tag{3}$$

$$D^{(a)}(R, k) = R - I^{(a)}(R, k) = \left\{ r \in R | (R_i^{(a)})_0 \geq (R_k^{(a)})_0 \right\} \tag{4}$$

Finalmente, R es correctamente particionable si:

$$P(R) = |R| = 1 \vee \exists k : C^{(a)}(k) \wedge P(I^{(a)}(k)) \wedge P(D^{(a)}(k)) \tag{5}$$

□

Usando el predicado (5) podemos comprobar que la complejidad algorítmica de calcular su veracidad es:

Proof.

$$\mathcal{T}(n) = \mathcal{T}(k) + \mathcal{T}(n - k) + \mathcal{O}(n * \log n) \tag{6}$$

donde el término $n * \log n$ es:

$$\mathcal{O}(n) = \mathcal{O}(n * \log n) = n * \log n + n + n \tag{7}$$

donde:

- $n * \log n$ para calcular (1)
- n para calcular (2) para el peor caso ($k = |R| - 1$)

- n para calcular (3) y (4)

La complejidad general del algoritmo esta dado por

$$\mathcal{T}(n) = \mathcal{O}(n^2 * \log n) \tag{8}$$

En el peor caso $k = 1$ □

Se puede mejorar este algoritmo. Primero, nótese que el cálculo de (2) en el conjunto $I^{(a)}(R, k)$ es independiente del cálculo en $D^{(a)}(R, k)$, lo que significa que nos podemos evitar la segunda llamada recursiva, por lo que la expresión (6) queda:

$$\mathcal{T}(n) = \mathcal{T}(k_i) * |\{k_i\}| + \mathcal{O}(n * \log n) \tag{9}$$

donde k_i son los posibles cortes

Segundo, se puede comprobar los posibles cortes de izquierda a derecha (usando la expresión (2)) y de derecha a izquierda, con el punto medio como limite para ambas exploraciones, de forma que se garantiza que k_i es mínimo. Para esta modificación del algoritmo:

Proof.

$$\begin{aligned}\mathcal{T}(n) &= \mathcal{T}(k_i) * |\{k_i\}| + \mathcal{O}(n * \log n) \\ &= n + \mathcal{O}(n * \log n) \\ &= \mathcal{O}(n * \log n)\end{aligned}$$

En el mejor caso ($\forall k_i = 0$:), y

$$\begin{aligned}\mathcal{T}(n) &= \mathcal{T}(k_i) * |\{k_i\}| + \mathcal{O}(n * \log n) \\ &= 2\mathcal{T}(\frac{n}{2}) + \mathcal{O}(n * \log n) \\ &= \mathcal{O}(n * \log^2 n)\end{aligned}$$

Para el peor caso ($k_0 = \frac{n}{2}$ y $k_1 = \frac{n}{2}$, $\{k_i\} = \{k_0, k_1\}$) □

1.2 Conference problem

1.2.1 Descripción

Una gran conferencia en ciencias sobrenaturales se va ha impartir en Byteland! En total n científicos de todo el mundo asistirán, cada uno de ellos envió los días en los que participan: dos enteros l_i y r_i (día de llegada y día de salida respectivamente), así como el su país de procedencia c_i, aunque algunos prefirieron no divulgar este último dato.

Todos saben que es interesante conocer académicos de otros países, así que los participantes que no puedan conocer pares de otros países se sentirán molestos. Es posible conocer nuevos amigos todos los días, incluidos los días de llegada y de salida

Los organizadores desean prepararse para lo peor, así que te encargan la tarea de calcular cuál es el mayor números de participantes molestos para un itinerario dado.

1.2.2 Datos

Enteros $1 \leq l_i \leq r_i \leq 10^6$ y $0 \leq c_i \leq 200$ para cada uno de los $1 \leq n \leq 500$ científicos que participan, donde $c_i = 0$ indica que no quiere divulgar su país de origen.

1.2.3 Solución

Cada participante tiene un periodo de tiempo para relacionarse con otros participantes, determinado por l_i y r_i, donde sabemos que se sentirá molesto si no conoce otro participante de distinto país, es decir: sea P el conjunto de participantes, $p \in P$ se sentirá molesto si $U_p = \neg\exists e \in P : [l_e, r_e] \cap [l_p, r_p] \neq \emptyset \wedge c_e \neq c_p$. De la expresión anterior se deduce lo necesario para resolver el problema: primero listar para cada $p \in P$ los participantes que coincidirán con p, es decir, algún día (o días) es común para los períodos de asistencia de ambos; teniendo los que coinciden, se cuentan aquellos que solo compartirán con otros participantes del mismo país que él. Además, para los participantes que no indican país de origen, se escoge la peor configuración, por ejemplo, si dos participantes asisten juntos, asumiremos que ambos tienen el mismo país, de forma que ambos estarán molestos.

La forma más simple de determinar para cada $p \in P$ cada $e \in P$ con los que coincidirán, se puede recorrer $p \in P$ comprobando si e comparte algún dia del itinerario, es decir $C_p = \{e \in P | \, [l_e, r_e] \cap [l_p, r_p] \neq \emptyset\}$; es evidente que la complejidad algorítmica de tal calculo es $\mathcal{O}(n^2)$, pero dadas las restricciones

del problema ($n \leq 500$) es aceptable; aun así es notable que existen métodos para calcular $C(p)$ en tiempo óptimo, como un SegmentTree, que permite un tiempo de construcción de $\mathcal{O}(n \log^2 n)$ para este problema específico.

Con los conjuntos $C_p \ \forall p \in P$, podemos calcular la cantidad de participantes molestos:

$$U = \sum_{p \in P} U_p \tag{10}$$

$$U_p = \begin{cases} 1 & \forall e \in C_p : c_e = c_p \\ 0 & \text{eoc} \end{cases}$$

siempre y cuando $\neg \exists p \in P : c_p = 0$. La expresión (10) da un indicio de su complejidad: se deben recorrer todos los elementos $p \in P$, y para cada cual se deben recorrer los elementos $e \in C_p$, que en el peor de los casos es $\forall p \in P : C_p = P$ (por ejemplo, si todos los participantes quieren asistir a la vez), lo que nos da $\mathcal{O}(n^2)$.

El conjunto C_p no contempla los casos donde $\exists p \in P : c_p = 0$ ya que para contemplar el peor escenario posible debemos optimizar el país del que proceden, es decir, asignarle el país que provoque mayor molestia entre sus co-participantes. Para ello se puede diseñar $(\max c_i)^k$ itinerarios diferentes, donde los $k = |\{p \in P : c_p = 0\}|$ participantes sin países se les asigna cualquiera de los posibles, luego para cada itinerario se calcula su respectivo valor U; la complejidad algorítmica de tal procedimiento es, el el peor caso ($k = n$), $\mathcal{O}((\max c_p)^n * n^2)$.

Evidentemente este es un algoritmo de fuerza bruta, muy poco eficiente, que prueba todos los posibles países asignables. Existen un número de optimizaciones: para $p \in P$ donde $c_p = 0$, se puede limitar la cantidad de países a probar a los países del conjunto $\{e \in C_p : c_e\}$, ya que asignarle un país diferente a todos los participantes $e \in P$ solo puede disminuir la cardinalidad de U; si $\forall e \in C_p : c_e = 0$ podemos asignarles a todos ellos un solo país, de forma que todos los participantes estarían molestos, o sea, $\forall e \in C_p : c_e = 0 \Rightarrow |U_p| = |C_p|$, por lo que no es necesario probar otros países. Sin embargo, estas optimizaciones solo mejoran la complejidad del caso promedio, dado que este problema es un ejemplo de optimización discreta.

1.3 Soup Time!

1.3.1 Descripción

En una región vacía, representada por un mapa cartesiano de coordenadas enteras, viven n enanos que quieren reunirse para ver su programa de televisión favorito. Cada enano se describe por su posición (x, y) en el mapa; además, cada enano puede moverse a una de las casillas $(x+1, y)$, $(x-1, y)$, $(x, y+1)$ o $(x, y-1)$ por cada unidad de tiempo. Existen además k estaciones de metro que pueden llevar al enano que las visite a cualquier otra en el mapa. Todos los enanos son muy respetuosos, por lo que no interfieren el camino de los otros, y pueden ocupar la misma casilla.

Los enanos necesitan ayuda para reunirse en el menor tiempo posible.

1.3.2 Datos

Enteros $1 \leq n \leq 10^5$ y $0 \leq k \leq 10^5$, la cantidad de enanos y estaciones de metro, respectivamente, cada uno de ellos descrito por enteros $|x_i| \leq 10^8$ y $|y_i| \leq 10^8$.

1.3.3 Solución

Sea

$$a = (a_0, a_1)$$

punto cualquiera en el plano

$$d(a, b) = |a_0 - b_0| + |b_1 - b_1| \tag{11}$$

Distancia entre los puntos a y b usando las reglas del problema
(conocida como Manhattan Distance)

Es evidente que la mejor peor solución del problema es obligar a todos los enanos a reunirse, de forma tal que el tiempo mínimo de recorrido en este caso es el tiempo que le toma al enano más alejado del punto de reunión.

Esta solución puede mejorarse usando las estaciones de metro. Para $k \geq 2$, si existe un par de estaciones e_0 y e_1 tal que para el par de puntos p_0 y p_1 (usados anteriormente, son el par de puntas más alejados entre sí) se cumple que $d(e_0, p_0) < d(p_0, p_1)$ y $d(e_1, p_1) < d(p_0, p_1)$, podemos reducir el tiempo que tome a los enanos reunirse tomando la ruta $p_0 \to e_0 \to e_1 \to p_1$.

Formalizando: sea D el conjunto de los puntos que describen los enanos, y E el conjunto equivalente para las estaciones, y sea la función que describe la zona del mapa alcanzable desde un punto p en un tiempo t

$$R(p,t) = \left\{x \in \mathcal{Z}^2 | d(x,p) \leq t\right\} \tag{12}$$

Dos enanos d_0 y d_1 se pueden reunir en t si:

$$R(d_0,t) \cap R(d_1,t) \neq \emptyset$$

De lo que se deduce que para un tiempo t, los $\{d_i\}$ enanos pueden reunirse si:

$$\forall i \leq n-1 : R(d_i) \cap R(d_{i+1}) \neq \emptyset \tag{13}$$

Además si existe una estación de metro al alcance de algún enano d_i, es decir, $\exists e_i \in E : e_i \in R(d_i,t)$, y no se cumple (13), entonces el camino del enano d_i puede tomar más de un camino, es decir, el conjunto de puntos alcanzables por d_i en un tiempo $t' > t$ es $R(d_i,t') \cup R(e_i,t')$. Generalizando:

$$Rc(d,t) = R(d,t) \cup (\cup_{e_i \in E : e_i \in R(d_i,t)} R(e_i,t)) \tag{14}$$

Es válido notar que si un enano d_i puede acceder a dos estaciones e_0 y e_1, y $d(d_i,e_0) \leq d(d_i,e_1)$ no es necesario plantearse el camino $d_i \to e_1 \to ....$

Finalmente las expresiones (14) y (13) se pueden combinar.

$$\forall i \leq n-1 : Rc(d_i) \cap Rc(d_{i+1}) \neq \emptyset \tag{15}$$

para crear el problema de optimización:

$$\begin{aligned} \min_{t} \quad & t \\ \text{s.t.} \quad & \forall i \leq n-1 : Rc(d_i) \cap Rc(d_{i+1}) \neq \emptyset \end{aligned}$$

Para obtener una solución óptima a este problema se debe primero notar que si (15) se cumple para t, también se cumple para $t+1$. Si se transforma el valor de (15) como entero (0 si no se cumple, 1 si lo hace), se obtiene un arreglo binario ordenado $\{r_i\}$, donde r_i indica si los enanos pueden encontrarse en $i+1$ unidades de tiempo. Sobre este arreglo podemos realizar una búsqueda binaria para encontrar en menor i tal que $r_i = 1$.

El costo algorítmico de calcular (14) es $\mathcal{O}(1)$, y (15) es $\mathcal{O}(n+k)$ por cada t en el peor de los casos; es notable que para calcular (14) podemos usar la primera estación alcanzable para cada enano. El costo algorítmico de la

búsqueda es $\mathcal{O}(\log T)$, donde T es el tiempo de la peor solución (que todos los enanos lleguen a una estación); además no es necesario calcular (15) para todo el espacio de búsqueda, de forma que la complejidad algorítmica final es $\mathcal{O}((n+k) * \log T)$.

Para calcular la estación más cercana a cada enano se puede usar un K-D Tree, que tiene un tiempo de construcción de $\mathcal{O}(n * \log n)$, y un tiempo de búsqueda combinado de $\mathcal{O}(n * \log n)$, dando un tiempo de preparación del algoritmo de $\mathcal{O}((n+k) * \log (n+k))$.