Algoritmos y Estructuras de Datos III - Clase pre parcial
05-MAY-2023
Julián Braier

1) (2022 2C 1R) El mejor amigo de Tuki, Pipo, también quiere aprender a contar cadenas. Como Tuki es un maestro de esta técnica, ahora le da ejercicios a su amigo para que aprenda y practique. Puntualmente, le propuso el siguiente desafío: contar cuántas cadenas de números naturales $a_1 \dots a_n$ existen tales que $0 \leq a_i \leq c$ para todo i, y exactamente k números de la cadena son primos. Como Pipo no creyó que este problema fuera lo suficientemente interesante, Tuki agregó la condición de que la cadena no puede tener m números consecutivos que no sean primos. Para facilitar su trabajo, le alcanzó una función esPrimo que permite calcular, dado un número x, si x es primo en $O(1)$.

   a) Definir en forma recursiva la función $f_{c,m} : \mathbb{N}^3 \to \mathbb{N}$ donde $f_{c,m}(i, k, t)$ calcula cuántas cadenas $a_1 \dots a_i$ existen tales que $0 \leq a_j \leq c$ para todo j, la cantidad de números primos entre los a_j es exactamente k, nunca ocurre que m números consecutivos no son primos, y, si $t \leq i$, **entonces hay un a_j con $1 \leq j \leq t$ tal que a_j es primo**. Señalar qué llamado debe hacer Pipo para resolver el problema.

      Ayuda: Utilice el tercer parámetro para asegurar que no haya m números consecutivos que no sean primos.

   b) Demostrar que $f_{c,m}$ tiene la propiedad de superposición de subproblemas.

   c) Definir un algoritmo *top-down* para calcular $f_{c,m}(i, k, t)$ indicando claramente las estructuras de datos utilizadas y la complejidad resultante.

   d) Escribir el (pseudo-)código del algoritmo top-down resultante.

   Complejidad: la complejidad temporal del algoritmo resultante para calcular $f_{c,m}(i, k, t)$ debe ser $O(c + i \min\{i, k\} \min\{i, m\})$.

2) (2022 1C 1R) La nueva reglamentación de una ciudad establece que tiene que haber una estación de policía a no más de 5 cuadras de cada esquina de la ciudad. Tenemos un grafo G cuyos n vértices representan las esquinas, donde sabemos cuáles esquinas están conectadas por cuadras. Queremos determinar todas las esquinas que no satisfacen la reglamentación, sabiendo que hay k estaciones de policía ubicadas en las esquinas $\{p_1, \dots, p_k\}$. Diseñar un algoritmo que resuelva el problema en tiempo lineal. Explicar claramente su implementación y por qué es correcto.

3) (2022 2C 1P) Decimos que un grafo pesado G es un *árbol enredado* si existe un ciclo C de 3 vértices tal que $G - E(C)$ (i.e., el grafo que resulta de sacarle a G las aristas de C) es árbol generador mínimo de G. Decimos que el ciclo C es un *nudo* de G.

   a) Mostrar un árbol enredado G para el cual alguna ejecución del algoritmo de Kruskal encuentra un AGM T' tal que las aristas en $G - E(T')$ no forman un nudo de G.

   b) Sea X un árbol generador cualquiera de un árbol enredado G que tiene un nudo C. Demostrar que al menos una de las aristas de $G - E(X)$ pertenece a C, cualquiera sea el nudo C.

   c) Dar un algoritmo para encontrar un nudo de G y su correspondiente AGM $T = G - E(C)$. **Sugerencia:** usar el item anterior para determinar aristas candidatas de C. ¿Cuántos candidatas puede haber?

   Complejidad: El mejor algoritmo que conocemos para encontrar un nudo tiene complejidad temporal $O(n)$. El algoritmo propuesto debe tener complejidad temporal $O(n^2)$.[1]

4) (2022 2C 1P) Un *modelo de intervalos* es una secuencia $\mathcal{I} = [s_1, t_1], \dots, [s_n, t_n]$ de intervalos cerrados tales que $0 \leq s_1 \leq s_2 \leq \dots \leq s_n$. El *grafo de intervalos* de $\mathcal{I}$ es el grafo $G(\mathcal{I})$ con n vertices $v_1, \dots, v_n$ tal que v_i y v_j son adyacentes si y solo si $t_i \geq s_j$ para todo $1 \leq i < j \leq n$.

[1] Recordar que $O(n) \subset O(n^2)$, con lo cual el algoritmo propuesto puede tener complejidad $O(n)$.

a) Proponer un algoritmo goloso que, dado un modelo de intervalos $\mathcal{I}$ cuyo grafo $G(\mathcal{I})$ es conexo, encuentre un árbol v_1-geodésico de $G(\mathcal{I})$. Recordar que un árbol T es v_1-geodésico cuando la distancia entre v_1 a v_i en T es igual a la distancia entre v_1 y v_i en G para todo $1 \leq i \leq n$. **Ayuda:** recordar el trabajo práctico, observando qué propiedad cumplen los intervalos correspondientes a cualquier camino de v_1 a v_i.

b) Demostrar que el algoritmo propuesto es correcto. **Ayuda:** recordar el trabajo práctico.

1. Solución

1) a) Sea p la cantidad de primos entre 0 y c. Podemos precomputar este valor en $O(c)$ usando la función esPrimo. La cantidad de no primos será $c + 1 - p$.

Empezamos pensando los casos base: $i = 0$. Hay una única cadena vacía. Esta sólo es válida si me están pidiendo que no tenga ningún número primo ($k = 0$). Además, si $t \leq i$, tendría que tener un a_j primo, con $1 \leq j \leq t$. Con $t \leq 0$ esta condición sería falsa.

Con $i \neq 0$ tenemos más casos base. Si nos piden una cantidad negativa de números primos ($k < 0$), o que entre los primeros 0 haya al menos un primo ($t \leq 0$), ya sabemos que no hay cadena que cumpla.

Caso recursivo, tratamos de definir $f_{c,m}(i, \_, \_)$ en función de $f_{c,m}(i-1, \_, \_)$. Definimos el valor del primer número y resolvemos para el sufijo con un llamado recursivo. Tenemos dos opciones:

- Poner un número primo (hay p maneras de hacerlo). Al sufijo le tendríamos que pedir que tenga exactamente $k - 1$ primos para que la cadena total, incluyendo al primer número, tenga k. Dado que acabamos de poner un primo, basta con pedir que alguno de los siguientes m sea primo para cumplir con que no haya m no primos consecutivos.
- Poner un no primo ($c+1-p$ maneras). Seguimos necesitando k primos en el sufijo. Como queríamos que uno de los primeros t caracteres sea primo y a_1 no es primo requerimos que el sufijo tenga un primo entre los primeros $t - 1$ caracteres.

Sumamos las dos cantidades.

$$f_{c,m}(i,k,t) = \begin{cases} 1 & i = 0 \wedge k = 0 \wedge t > 0 \\ 0 & i = 0 \wedge (k \neq 0 \vee t \leq 0) \\ 0 & i \neq 0 \wedge (k < 0 \vee t \leq 0) \\ p * f_{c,m}(i-1, k-1, m) + (c+1-p) * f_{c,m}(i-1, k, t-1) & cc \end{cases}$$

El llamado que se debe hacer para resolver es $f_{c,m}(n, k, m)$.

b) Para resolver $f_{c,m}(i, \_, \_)$ hacemos en peor caso 2 llamados a $f_{c,m}(i-1, \_, \_)$. La cantidad de llamados recursivos para resolver $f_{c,m}(n, k, m)$ es $\Omega(2^n)$.

La cantidad de casos diferentes es $O(n * \text{mín}\{n,k\} * \text{mín}\{n,m\})$. El parámetro i toma $O(n)$ valores. Si k es negativo o mayor a i podemos responder 0 en $O(1)$, así que sólo toma $O(\text{mín}\{n,k\})$ valores interesantes. Si $m > n$ podríamos tomar $m = n+1$ y obtendríamos la misma respuesta y t sólo puede tomar valores entre 0 y m.

Hay superposición de subproblemas cuando $2^n >> n * \text{mín}\{n,k\} * \text{mín}\{n,m\}$.

2) $G = (V, E)$. Modelamos con $G' = (V', E')$.

- $V' = V \cup \{w\}$
- $E' = E \cup \{(p_i, w) : 1 \leq i \leq k\}$

En castellano: armamos un modelo agregando a G un vértice w y uniéndolo con una arista a cada vértice que tenga una estación de policías.

Lema: Para todo $v \in V$ vale que v tiene un camino en G' de longitud x a w sii tiene un camino en G de longitud $x - 1$ a alguna estación de policía.

Demo:

- Ida: Sea $v_0, v_1, ..., v_{x-1}, v_x$ el camino de v a w ($v_0 = v$ y $v_x = w$). Por construcción de G' sabemos que v_{x-1} tiene estación de policía. Entonces $v_0, v_1, ..., v_{x-1}$ es un camino de longitud $x-1$ a alguna estación de policía.
- Vuelta: Tenemos en G el camino de long $x-1$ a la estación de policía agregando la arista que va de la estación a w tenemos el camino de longitud x a w.

Solución: Modelar con G'. Hacer en ese grafo BFS desde w para calcular distancia al resto de los vértices. Una esquina está a más de 5 cuadras de una estación de policías sii el vértice que la representa está a distancia mayor a 6 de w.

3) (solución parcialmente basada en una de Santiago Cifuentes)

a) Se puede buscar un contraejemplo. Un K_4 con todas las aristas con el mismo costo sirve. Es árbol enredado porque $2, 3, 4, 2$ es un nudo de G. Una ejecución de Kruskal podría encontrar el AGM $T' = \{(1,2), (2,3), (3,4)\}$. Las tres aristas en $G - E(T')$ no forman un ciclo.

b) Notemos que no puede ser que todas las aristas del ciclo C pertenezcan a X (X no tiene ciclos). Luego, una de estas está en $G - E(X)$.

c) **Observación:** un árbol enredado tiene que $m = n + 2$, ya que es un árbol más tres aristas. $m = O(n)$. Tenemos que encontrar el triángulo C de mayor peso tal que $G - E(C)$ sea un árbol.

   1) Computamos con DFS un AG de G. Llamémoslo T'. ($O(n+m) = O(n)$)
   2) Usando la info del ítem anterior sabemos que para todo nudo C al menos una de las tres aristas de $G - E(T')$ pertenece a C. Para cada arista $vw \in G - E(T')$ solo hay que calcular la intersección de los vecindarios $N(v) \cap N(w)$ y calcular para cada $z \in N(v) \cap N(w)$ el valor $c(vw) + c(wz) + c(zv)$. Esto se puede hacer en $O(n)$ creando un vector d_v y otro d_w y guardando $d_v[z] = c(vz)$ (si vz no es una arista se pone $-\infty$). Hay $O(n)$ triángulos por arista: $O(n)$ triángulos en total.
   3) Además debemos chequear que al quitar estas tres aristas de G nos queda un árbol. Chequeable con DFS/BFS en $O(n+m) = O(n)$. Sólo consideramos esos triángulos.
   4) De los triángulos considerados elegir el de máximo costo.

   Bonus: para ver que es $O(n)$ en total acotar la cantidad de triángulos candidatos por una constante.

4) (solución enteramente robada -con pedido de permiso- a Santi Cifuentes)

Los nodos adyacentes a v_1 (es decir, a distancia 1) se pueden encontrar fácilmente teniendo en cuenta que estos tienen que ser los intervalos $v_2 \dots v_k$ para algún k (específicamente el primer k tal que $s_{k+1} > t_1$). Luego, para encontrar los nodos a distancia 2, alcanza con elegir (golosamente) el intervalo adyacente a v_1 con mayor t_i y buscar los adyacentes a este recorriendo los nodos a partir de v_{k+1}.

El algoritmo funciona en fases: en la j-esima fase buscamos los nodos adyacentes al nodo distinguido v_{i_j} determinado por la fase anterior, definiendo las distancias de estos nodos como $d(v_1, v_{i_j}) + 1$. Aparte, nos quedamos con el nodo de mayor t, y designamos a este como el distinguido para la siguiente iteración. Tomamos $i_1 = 1$.

Probemos por inducción que al comienzo de la j-esima fase el algoritmo definió un árbol v_1-geodésico que contiene a todos los nodos a distancia menor a j. Para $j = 1$ esto es cierto.

Para demostrar el paso inductivo, supongamos que el algoritmo ya encontró todos los nodos a distancia menor que $j+1$. Para hacer esto en la última iteración usó a un nodo candidato v_{i_j}, y cortó la iteración una vez que encontró al nodo v_k con $s_k > t_{i_j}$. Notemos que todos los nodos a distancia $j+1$ de v_1 tienen que ser adyacentes a algún nodo a distancia j. En particular, seguro son adyacentes al nodo distinguido definido en la j-ésima iteración, ya que este tiene inicio anterior a cualquiera de estos nodos a distancia $j+1$ (sino tendrían distancia menor) y final posterior a todos los que están a distancia j. Por lo tanto, estos van a ser encontrados cuando se recorran los nodos restantes partiendo del último visitado (el v_k).