Parcial de Programación 3
19 de septiembre de 2019

En recuadros con este formato aparecerán aclaraciones que cumplen una función explicativa pero que no eran requeridos como parte de la solución.

Ejercicio 1 (10 puntos)

Considere una instancia del problema de emparejamiento estable entre un conjunto $M = \{m_1, m_2, m_3, m_4\}$ y otro $W = \{w_1, w_2, w_3, w_4\}$, con listas de preferencias dadas por las siguientes tablas. A modo de ejemplo, m_1 prefiere en primer lugar a w_1, mientras que para w_1 la mejor opción es m_4.

m_1	w_1	w_2	w_3	w_4
m_2	w_1	w_4	w_3	w_2
m_3	w_2	w_1	w_3	w_4
m_4	w_4	w_2	w_3	w_1

w_1	m_4	m_3	m_1	m_2
w_2	m_2	m_4	m_1	m_3
w_3	m_4	m_1	m_2	m_3
w_4	m_3	m_2	m_1	m_4

(a) Presente una sucesión de formaciones y separaciones de parejas que se realizan a lo largo de una ejecución del algoritmo de Gale-Shapley.

(b) Muestre que para esta instancia del problema existe una **única** solución. Enuncie explícitamente todos los resultados teóricos que utilice.

Sugerencia: Repita la parte a invirtiendo los roles de M y W.

Solución:

(a) Con M en rol de proponente una ejecución posible da como resultado las siguientes formaciones y separaciones de parejas

1. Se forman (m_1, w_1), (m_3, w_2) y (m_4, w_4).
2. Se separa (m_4, w_4) y se forma (m_2, w_4).
3. Se separa (m_3, w_2) y se forma (m_4, w_2).
4. Se separa (m_1, w_1) y se forma (m_3, w_1).
5. Se forma (m_1, w_3).

(b) Con W en rol de proponente una ejecución posible da como resultado las siguientes formaciones y separaciones de parejas

1. Se forman (w_2, m_2), (w_3, m_4) y (w_4, m_3).
2. Se separa (w_4, m_3) y se forma (w_1, m_3).
3. Se separa (w_2, m_2) y se forma (w_4, m_2).
4. Se separa (w_3, m_4) y se forma (w_2, m_4).
5. Se forma (w_3, m_1).

Cuando el rol de proponente lo lleva adelante el conjunto X, el algoritmo de Gale-Shapley forma parejas (x, y), $x \in X$, tales que:

- y es la persona de mayor preferencia para x tal que (x, y) son una pareja en algún emparejamiento estable,
- x es la persona de menor preferencia para y tal que (x, y) son una pareja en algún emparejamiento estable.

En la instancia del problema presentada en este ejercicio, el emparejamiento estable que se forma es el mismo (a menos del orden en que se listan los componentes de cada pareja),

$$S = \{(w_1, m_3), (w_2, m_4), (w_3, m_1), (w_4, m_2)\},$$

tanto cuando M propone como cuando W propone. En consecuencia, para toda persona se verifica que su mejor pareja válida coincide con su peor pareja válida y, por lo tanto, esa es su única pareja posible en todo emparejamiento estable.

Ejercicio 2 (18 puntos)

Dado un arreglo A que contiene n números enteros, $A_1, A_2, \ldots, A_n$, donde n es potencia de 2, queremos encontrar un conjunto no vacío de celdas contiguas, $A_i, \ldots, A_j$, $1 \leq i \leq j \leq n$, cuya suma sea lo más grande posible. Por ejemplo, para el arreglo $A = [-1, 3, -2, 5, 0, -1, -1, -1]$ existen dos soluciones posibles; una de ellas está dada por los índices de inicio y fin $i = 2$, $j = 4$, y la otra por $i = 2$, $j = 5$. El valor de la suma es 6 en ambos casos.

(a) Dé un algoritmo de tipo Divide y Vencerás que devuelve los índices de inicio y fin de un intervalo de suma máxima junto con el valor de la suma. Su algoritmo debe admitir una implementación cuyo tiempo de ejecución sea $O(n \log n)$.

Sugerencia: Para cada subproblema calcule, además de su solución, sumas de celdas contiguas que parten desde un extremo.

(b) Muestre que su algoritmo admite una implementación cuyo tiempo de ejecución es $O(n \log n)$. Enuncie (sin demostración) cualquier resultado teórico estudiado en el curso.

Solución:

(a) El algoritmo se presenta en la figura 1.

El problema original de tamaño n se divide en dos partes de igual tamaño, que se resuelven recursivamente en los pasos 5 y 6. Estas soluciones parciales se combinan para formar una solución general observando que todo intervalo candidato para la maximización cae en alguna de las siguientes categorías:

- Está totalmente contenido en la mitad izquierda, en cuyo caso es evaluado en el paso 5.
- Está totalmente contenido en la mitad derecha, en cuyo caso es evaluado en el paso 6.
- Comienza en una posición I en la mitad izquierda, $1 \leq I \leq n/2$, y termina en una posición J en la mitad derecha, $n/2 < J \leq n$. Para un intervalo de suma máxima en esta categoría, los índices I y J son soluciones para las maximizaciones que se realizan en los pasos 7 y 8, respectivamente, y su suma coincide con el valor S calculado en paso 9.

```
1  Algorithm SumaMaxima ( A )
2      if n = 1 then
3          Devolver (1, 1, A[1])
4      else
5          (i_1, j_1, s_1) = SumaMaxima (A[1 ... n/2])
6          (i_2, j_2, s_2) = SumaMaxima (A[n/2 + 1 ... n])
7          Tomar I ∈ arg máx { Σ_{i=I}^{n/2} A[i] }_{1≤I≤n/2}
8          Tomar J ∈ arg máx { Σ_{i=n/2+1}^{J} A[i] }_{n/2<J≤n}
9          Hacer S = Σ_{i=I}^{J} A[i]
10         if S ≥ máx{s_1, s_2} then
11             Devolver (I, J, S)
12         else
13             if s_1 ≥ s_2 then
14                 Devolver (i_1, j_1, s_1)
15             else
16                 Devolver (i_2 + n/2, j_2 + n/2, s_2)
17             end
18         end
19     end
20 end
```

Figura 1: Algoritmo para determina un intervalo de suma máxima.

(b) Sea $T(n)$ el tiempo de ejecución máximo para un arreglo de tamaño n. Como el paso 3 requiere tiempo $O(1)$, para $n = 1$ tenemos

$$T(1) \leq c\,. \tag{1}$$

Para $n > 1$, se ejecutan los pasos 4 a 18. Los tiempos de ejecución de los pasos 5 y 6 están acotados por $T(n/2)$ cada uno. Por otra parte, la ejecución de los pasos 7 y 8 requiere tiempo $O(n)$. En efecto, la maximización del paso 7 se puede implementar mediante un ciclo que itera para valores de I disminuyendo desde $n/2$ hasta 1, acumulando en cada paso el elemento $A[I]$ en una variable auxiliar, y registrando en cada paso la suma máxima alcanzada hasta ese momento. El paso 8 se puede implementar mediante un ciclo que itera para valores de J aumentando desde $n/2 + 1$ hasta n, de forma completamente análoga. El paso 9 requiere tiempo $O(1)$ usando las sumas máximas parciales calculadas en los pasos 7 y 8. Finalmente, los pasos 10 en adelante claramente requieren tiempo $O(1)$. A partir de este análisis concluimos que, para $n > 1$, se satisface

$$T(n) \leq 2T(n/2) + cn\,, \quad n > 1, \tag{2}$$

donde elegimos c suficientemente grande para que se satisfagan simultáneamente (1) y (2).

> Como para $n = 2$ se generan solo dos llamadas recursivas, donde cada una de ellas requiere tiempo acotado por una constante, y el tiempo para las operaciones adicionales que se realizan en los pasos 4 a 18 también está acotado por una constante, podríamos escribir
>
> $$T(n) \leq c\,, \quad n \leq 2 \tag{3}$$
>
> en lugar de (1), que coincide con el formato de recurrencia presentado en el libro de referencia.

Toda función T que satisface la recurrencia definida por las ecuaciones (1) y (2) es $O(n \log n)$, lo cual concluye nuestro análisis del tiempo de ejecución del algoritmo.

Es posible resolver el problema con un algoritmo de tipo Divide y Vencerás con tiempo de ejecución $O(n)$. Para ello hacemos que, además de los valores (i, j, s), nuestro algoritmo devuelva la suma total de los elementos del arreglo,

$$X = \sum_{i=1}^{n} A[i]\,, \tag{4}$$

la suma máxima de un subarreglo con el extremo inicial fijo en la primera posición de A,

$$Y = \text{máx} \left\{ \sum_{i=1}^{y} A[i] \right\}_{1 \leq y \leq n}, \tag{5}$$

y la suma máxima de un subarreglo con el extremo final fijo en la última posición de A,

$$Z = \text{máx} \left\{ \sum_{i=z}^{n} A[i] \right\}_{1 \leq z \leq n}. \tag{6}$$

Los valores de i, j, s, X, Y, Z se pueden obtener en tiempo $O(1)$ a partir de los resultados obtenidos para los subproblemas de tamaño $n/2$, lo cual implica que $T(n)$ satisface

$$T(n) \leq 2T(n/2) + c\,, \quad n > 1, \tag{7}$$

para cierta constante c. Resolviendo esta recurrencia, con el mismo paso base (1), se concluye que el tiempo total de ejecución es $O(n)$.

Ejercicio 3 (22 puntos)

Consideramos un grafo dirigido fuertemente conexo $G = (V, E)$, donde $|V| = n$, $|E| = m$, con longitudes **positivas** asociadas a sus aristas. Tenemos un "comodín" que nos da derecho a poner en cero la longitud de una y solo una arista. Queremos encontrar un camino de longitud mínima desde **un nodo dado**, s, a **otro nodo dado**, t, usando el comodín de la forma más conveniente posible.

(a) Dé un algoritmo para obtener un camino de longitud mínima desde un nodo dado a todos los demás en G **(sin usar ningún comodín)**. Aunque no es necesario demostrarlo, el algoritmo debe admitir una implementación cuyo tiempo de ejecución sea $O(m \log n)$. Reescriba cualquier algoritmo que utilice del material teórico del curso.

(b) Sea $P = v_1, \ldots v_i, v_{i+1}, \ldots, v_\ell$ un camino de longitud mínima entre todos los caminos de s a t que recorren la arista (v_i, v_{i+1}). Demuestre que $v_1, \ldots v_i$ es un camino de longitud mínima de s a v_i y que $v_{i+1}, \ldots, v_\ell$ es un camino de longitud mínima de v_{i+1} a t.

(c) Escriba un algoritmo que devuelve un camino de longitud mínima desde **un nodo dado**, s, **a otro nodo dado**, t, usando el comodín de la forma más conveniente posible. Su algoritmo debe admitir una implementación cuyo tiempo de ejecución sea $O(m \log n)$.

Sugerencia: Para todo vértice u de G, obtenga un camino de longitud mínima de s a u junto con la distancia entre ellos. Construya el grafo dirigido G' que se obtiene a partir de G invirtiendo el sentido de las aristas. Utilice G' para obtener, para todo vértice u de G, un camino de longitud mínima de u a t en G junto con la distancia entre ellos. Use esta información para evaluar todos los posibles usos del comodín.

(d) Demuestre que su algoritmo para la parte c admite una implementación cuyo tiempo de ejecución es $O(m \log n)$. Puede usar sin necesidad de demostrar resultados teóricos estudiados durante el curso.

Solución:

(a) Este problema se resuelve mediante el algoritmo de Dijkstra, que está detallado en el libro de referencia del curso.

(b) Supongamos que existe un camino de s a v_i, P'_{v_i}, de menor longitud que $v_1, \ldots v_i$. Entonces, la concatenación de P'_{v_i} con $v_{i+1}, \ldots, v_\ell$ es un camino de s a t que pasa por la arista (v_i, v_{i+1}) y tiene menor longitud que P, lo cual es absurdo. Análogamente, la existencia de camino de v_{i+1} a t de menor longitud que $v_{i+1}, \ldots, v_\ell$ lleva a exactamente la misma contradicción.

(c)
1. Construir, usando el algoritmo de Dijkstra, la longitud mínima en G de s a u, d_u, y un camino, P_u, de longitud d_u para cada uno de los vértices u de G.
2. Construir el grafo dirigido G' que se obtiene a partir de G invirtiendo el sentido de las aristas.
3. Construir, usando el algoritmo de Dijkstra, la longitud mínima en G' de t a u, d'_u, y un camino, P'_u, de longitud d'_u para cada uno de los vértices u de G'.
4. Obtener una arista (u, v) de G que minimiza la suma $d_u + d'_v$.
5. Devolver el camino que resulta de concatenar P_u con Q_v, donde Q_v es el camino en G que se obtiene invirtiendo el orden de recorrida de P'_v.

(d) El tiempo requerido para los pasos 1 y 3 es $O(m \log n)$. La construcción del grafo G' en el paso 2 no necesita hacerse explícitamente; cada vez que se necesita acceder a G' lo hacemos a través de G, intercambiando la interpretación de las listas de adyacencia de aristas entrantes y salientes. El paso 4 implica recorrer todos los vértices de G y, para cada uno, recorrer todas las aristas salientes de ese vértice, calculando, para cada una de ellas, una suma $(d_u + d'_v)$ y realizando otras operaciones que requieren tiempo $O(1)$ para recordar la arista que minimiza el valor de dicha suma. En total, el tiempo insumido en el paso 4 es $O(n + m)$. Como G es fuertemente conexo, se cumple que $n = O(m)$ y por lo tanto el tiempo total insumido en el paso 4 es $O(m)$, que es también $O(m \log n)$. Finalmente el paso 5 implica construir una lista con los vértices que encontramos en una recorrida de P_u y una recorrida de la inversión de P'_v, donde las recorridas de P_u y P'_v se realizan siguiendo la cadena de padres en el árbol construido por el algoritmo de Dijkstra. Como el camino construido no tiene más de n vértices, el tiempo requerido para este paso es $O(n)$, que, como ya hemos explicado, es $O(m \log n)$ porque G es fuertemente conexo.

Del análisis anterior observamos que el algoritmo ejecuta una cantidad constante de pasos, cada uno de los cuales insume una cantidad de tiempo que es $O(m \log n)$, de donde concluimos que el tiempo total de ejecución es $O(m \log n)$.