Parcial de Programación 3
27 de septiembre de 2018

Ejercicio 1 (15 puntos)

Dado un grafo no-dirigido $G = (V, E)$ con n vértices y m aristas, se busca determinar si es posible colorear cada uno de sus vértices con uno de los colores rojo o azul de tal forma que dos vértices adyacentes no tengan el mismo color.

(a) Dé un algoritmo basado en **DFS** que determine una coloración de los vértices en dicha forma o que indique que no es posible hacerlo. El tiempo de ejecución del algoritmo debe ser $O(m+n)$. Reescriba cualquier algoritmo que utilice de los estudiados en el curso.

(b) Mostrar que el tiempo de ejecución del algoritmo es $O(m+n)$. Reescriba los argumentos que se hayan visto en el curso.

Solución:

(a) El problema se puede resolver coloreando los vértices mientras se realiza una recorrida DFS del grafo. Para cada componente conexa, luego de colorear un primer vértice, cada vértice adyacente no visitado se colorea con un color diferente al del vértice por el cual se llega. Se determina que no se puede colorear el grafo en el caso que se llega a un vértice adyacente visitado desde un vértice de igual color. El algoritmo de la Figura 1, basado en DFS, resuelve el problema.

```
1  Algorithm BicolorearGrafo
       /* La marca de no visitado se establece con color blanco.          */
2      Hacer color [v] := blanco para todo v ∈ V
       /* Se ejecuta DFS-Color por cada componente conexa.                */
3      foreach v ∈ V do
4          if color [v] = blanco then
5              DFS-Color(v,azul)
6          end
7      end
8  end
9  Procedure DFS-Color(v,c)
10     color [v] := c
11     foreach w : (v,w) ∈ E do
12         if color [w] = blanco then
13             if color [v] = rojo then
14                 c := azul
15             else
16                 c := rojo
17             end
18             DFS-Color(w,c)
19         else
20             if color [w] = color [v] then
21                 Reportar imposibilidad de coloración
                   /* La arista (v,w) integra un ciclo de largo impar.    */
22                 TERMINAR
23             end
24         end
25     end
26 end
```

Figura 1: Algoritmo basado en DFS para bicolorear grafo.

(b) El algoritmo es implementado recursivamente. En el siguiente análisis asumimos una representación de lista de adyacencia para el grafo, lo cual permite implementar el bucle del paso 11 iterando sobre la lista de adyacentes de v. El procedimiento `DFS-Color` es invocado para cada uno de los n vértices solo una vez al pasar el vértice de color blanco a otro color. Para cada vértice se exploran sus vértices adjacentes, por lo que cada arista se indaga dos veces. O sea, si la cantidad de vértices adyacentes al vértice v es n_v, el bucle del paso 11 se ejecuta a lo sumo $\sum_{v \in V} n_v = 2m$ veces. Por lo tanto $O(m + n)$ es una cota superior para el tiempo de ejecución del algoritmo.

Ejercicio 2 (17 puntos)

Un tambo uruguayo almacena la leche que genera en un tanque para su pasteurización a partir de variaciones de temperatura. El calentamiento y enfriamiento de la leche se deben realizar lentamente y de forma controlada, ya que de otra manera el producto se estropea. Específicamente, se sabe que el producto corre riesgo de estropearse si durante el proceso de pasteurización la leche alcanza una temperatura d grados superior a la que tuvo en algún momento anterior. El tambo cuenta con un sistema que registra y envía n mediciones de temperatura a un servidor, que se almacenan en un arreglo $T = [t_1, ..., t_n]$ en el orden temporal en que fueron medidas. El número n es una potencia de 2 mayor o igual 2. La empresa le pide a usted que desarrolle un algoritmo que tome como entrada el arreglo T y determine si hay riesgo de que la leche esté estropeada.

(a) Presente un algoritmo que sigue una estrategia de tipo divide y vencerás para resolver el problema en tiempo $O(n\log(n))$. Justifique que el tiempo de ejecución es $O(n\log(n))$ escribiendo una relación de recurrencia y explicando el origen de cada término (no es necesario resolverla).

(b) Sea $T(n)$ una función definida sobre los naturales que son potencia de 2 y mayores o iguales a 2 que satisface

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

donde c es una constante positiva. Muestre que $T(n)$ es $O(n)$.
Sugerencia: Muestre por inducción que $T(n) \leq cn - c$.

(c) Explique qué modificaciones hacer a su algoritmo de la parte a para que el tiempo de ejecución sea $O(n)$ (si no se necesita ninguna modificación, justifique).

Solución:

(a) El algoritmo de la figura 2 resuelve el problema. Para $n = 2$, la ejecución del paso base demanda un tiempo que es $O(1)$. Para $n > 2$ se realizan dos llamadas recursivas sobre problemas de tamaño $n/2$; excluyendo estas llamadas, el tiempo de ejecución del resto de los pasos es $O(n)$ (de hecho es $O(1)$ para todos salvo el paso 8). Por lo tanto, el tiempo de ejecución total del algoritmo, $T_1(n)$, satisface la recurrencia

$$T_1(2) \leq k, \tag{3}$$
$$T_1(n) \leq 2T_1(n/2) + kn, \quad n > 2, \tag{4}$$

donde k es una constante positiva. Sabemos del material teórico estudiado en el curso que toda función que satisface esta recurrencia es $O(n\log n)$.

```
1  Algorithm Estropeada ( T )
       /* Temperaturas en T = [t_1, ..., t_n]                          */
2      if n = 2 then
3          Devolver true si t_2 − t_1 > d y false en otro caso
4      else
5          if Estropeada ( [t_1, ..., t_{n/2}] ) or Estropeada ( [t_{n/2+1}, ..., t_n] ) then
6              devolver true
7          else
8              Sea m = mín{t_1, ..., t_{n/2}} y M = máx{t_{n/2+1}, ..., t_n}
9              Devolver true si M − m > d y false en otro caso
10         end
11     end
12 end
```

Figura 2: Algoritmo para detectar si la leche puede estar estropeada.

(b) Para $n = 2$ tenemos $cn - c = c$ y, por (1), sabemos que $T(2) \leq c$, de modo que la tesis se cumple. Para $n > 2$, por (2) se cumple

$$T(n) \leq 2T(n/2) + c$$

y, asumiendo que la tesis se cumple para potencias de 2 menores que n, tenemos

$$2T(n/2) + c \leq 2\,(cn/2 - c) + c = cn - c\,,$$

por lo cual, combinando ambas desigualdades, vemos por inducción que la tesis se cumple para todo n potencia de 2, $n \geq 2$.

(c) Para especificar el subproblema que se resuelve en cada llamada recursiva hacemos que el algoritmo reciba por parámetro los índices de inicio y fin dentro del arreglo de mediciones general. También hacemos que devuelva el mínimo y el máximo de la porción del arreglo que procesa. Esto implica modificar el paso base para devolver estos valores, lo cual se realiza en tiempo $O(1)$ y, en el caso recursivo, combinar los mínimos y máximos devueltos en las llamadas recursivas para obtener y devolver el mínimo global, m, y el máximo global, M, lo cual también se realiza en tiempo $O(1)$. De esta manera el paso 8 ahora requiere tiempo $O(1)$, por lo cual el tiempo de ejecución del algoritmo satisface la recurrencia (1)-(2) y en consecuencia es $O(n)$ por la parte anterior.

Ejercicio 3 (18 puntos)

Mauricio cuenta con un conjunto S de postulantes para formar un equipo de trabajo. Quiere descartar algunos de ellos para que el equipo tenga suficiente cohesión pero que ningún integrante tenga demasiada influencia en los otros. Decimos que un equipo E es *válido* si cumple las siguientes condiciones:

1. todo integrante de E conoce a al menos c integrantes de E y
2. todo integrante de E tiene al menos d desconocidos en E.

De cada postulante de S se conoce el conjunto de conocidos. Se asume que la relación de conocimiento es recíproca, esto es, si p_i conoce a p_j entonces p_j conoce a p_i.

Nota: Es posible que el único equipo válido sea el conjunto vacío.

(a) Dé un algoritmo ávido que devuelva un equipo válido de tamaño máximo. No es necesario implementar operaciones elementales sobre conjuntos como intersección, unión, pertenencia y cardinal.

(b) Demuestre que el algoritmo es correcto.

Sugerencia: Muestre por inducción que en ningún paso se descarta un postulante que podría pertenecer a un equipo válido.

Solución:

(a) Se propone el siguiente algoritmo. Observar que la condición del ciclo se puede verificar calculando el cardinal de la intersección entre E y el conjunto de conocidos de p. Específicamente, si C_p es el conjunto de conocidos de p, p cumple las condiciones si y solo si $|E| - d \geq |E \cap C_p| \geq c$.

```
1 Algorithm FormarEquipo
2     E = S
3     while Existe en E un integrante p que no cumple la condición 1 o la condición 2 do
4         Excluir p de E
5     end
6     Devolver E
7 end
```

Figura 3: Algoritmo para formar el equipo.

(b) El algoritmo termina porque en cada iteración se excluye un integrante y hay una cantidad finita de ellos. La condición del ciclo de la línea 3 asegura que el equipo E devuelto cumple las condiciones 1 y 2. Para comprobar que el tamaño es máximo se demuestra por inducción en la cantidad de iteraciones del ciclo, $i \in \{0 \cdots m\}$, donde m es la cantidad final de iteraciones, que al terminar la iteración i no se excluyó a ningún postulante que podría pertenecer a un equipo válido. Dicho de otra forma, nuestra solución "está por delante" de cualquier otra solución válida alternativa, A, en el sentido de que $A \subseteq E$ a lo largo de toda la ejecución del algoritmo.

Paso base Cuando $i = 0$ la proposición se cumple de manera trivial porque no se excluyó a ningún postulante.

Paso inductivo Sea $i \in \{1 \cdots m\}$ una iteración del ciclo. Por la condición de entrada al ciclo se sabe que existe un postulante p que no cumple la condición 1 o la condición 2. Por la hipótesis inductiva tenemos $A \subseteq E$. Por lo tanto, la cantidad de conocidos de p en A es menor o igual que la cantidad de conocidos de p en E. La misma relación se cumple para la cantidad de no conocidos de p en A y en E. En consecuencia, como p no cumple alguna de las condiciones en E, tampoco las cumpliría en A, de modo que $p \notin A$ y por lo tanto al quitar p de E se sigue verificando que $A \subseteq E$.

Concluimos que al terminar la ejecución se devuelve un equipo que cumple las condiciones requeridas y tiene tamaño máximo.