Examen de Programación 3
12 de diciembre 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 (35 puntos)

El objetivo de este ejercicio es construir un algoritmo que tiene como entrada un grafo $G = (V, E)$, no necesariamente conexo, y determina qué aristas agregar a G de tal forma que el resultado sea un grafo conexo. La salida del algoritmo es un conjunto A (donde cada elemento de A es un conjunto de dos vértices), tal que el grafo $G' = (V, E \cup A)$ es conexo.

El algoritmo debe ser eficiente en los siguientes sentidos:

1. Se agrega la menor cantidad posible de aristas.
2. Admite una implementación cuyo tiempo de ejecución es $O(n + m)$, donde $n = |V|$ y $m = |E|$.

(a) Dé un algoritmo que resuelva el problema. Reescriba cualquier algoritmo estudiado en el curso que utilice en su solución.

(b) Demuestre que su algoritmo admite una implementación cuyo tiempo de ejecución es $O(n + m)$. Reescriba cualquier argumento estudiado durante el curso que utilice en su demostración.

Solución:

(a) La figura 1 presenta un algoritmo que resuelve el problema.

```
1  Algorithm Conectar
2      Hacer visitado [u] = false para todo u ∈ V
3      Hacer A = {}
4      Invocar DFS(1)
5      for u = 2 to n do
6          if not visitado [u] then
7              Agregar {u − 1, u} a A
8              Invocar DFS(u)
9          end
10     end
11     Devolver A
12 end
13 Procedure DFS(u)
14     visitado [u] = true
15     foreach v adyacente a u do
16         if not visitado [v] then
17             Invocar DFS(v)
18         end
19     end
20 end
```

Figura 1: Algoritmo basado en DFS para unir las componentes conexas de un grafo agregando la menor cantidad posible de aristas.

(b) Representando A mediante una lista, los pasos 3 y 7 requieren tiempo $O(1)$. Por lo tanto, como los pasos 2 y 5 involucran $O(n)$ iteraciones, el tiempo total de ejecución del algoritmo, excluyendo las llamadas a DFS, es $O(n)$, que a su vez es claramente $O(n + m)$. A continuación veremos que el tiempo

total insumido entre todas las llamadas a DFS es $O(n+m)$, lo cual demuestra que el tiempo total de ejecución del algoritmo completo es $O(n+m)$.

Todas las invocaciones a DFS (en los pasos 4, 8, 17) se realizan para vértices que aún no han sido marcados como visitados en el arreglo visitado. Esto es consecuencia de la asignación inicial en el paso 2 y las condiciones en los pasos 6 y 16. Como los vértices para los cuales se invoca DFS son marcados como visitados al inicio de la ejecución de DFS en el paso 14, nunca se repite una invocación a DFS para un mismo vértice.

Como no se repite la visita de ningún vértice, la cantidad total de veces que se ejecuta el ciclo del paso 15 a lo largo de todas las ejecuciones de DFS no es mayor que $\sum_{u \in V} n_u$, donde n_u denota el grado del vértice u (de hecho esa es exactamente la cantidad de iteraciones porque todo vértice se visita exactamente una vez). Como se cumple que $\sum_{u \in V} n_u = 2m$, porque cada arista contribuye con un unidad al grado de cada uno de los dos vértices que une, tenemos que la cantidad total de iteraciones que se realizan del ciclo del paso 15 es $O(m)$.

Usando una representación de lista de adyacencia para G, iterar sobre los vértices adyacentes a un vértice u requiere tiempo $O(1)$ por iteración. Como los pasos 14 y 16 requieren tiempo $O(1)$ y la cantidad de invocaciones a DFS es $O(n)$, concluimos que el tiempo total requerido para la ejecución de todas las invocaciones a DFS es $O(n+m)$.

Ejercicio 2 (35 puntos)

Usted trabaja en un equipo de automovilismo y su objetivo es optimizar la planificación de una carrera.

La carrera consiste en completar n vueltas de un circuito. Existen dos tipos de neumáticos disponibles: tipo A y tipo B. El rendimiento del automóvil con cada tipo depende de las condiciones meteorológicas (se ignora cualquier otro factor). En función del pronóstico meteorológico se tiene una colección de valores $t(i, X)$, para $i \in \{1, \ldots, n\}$, $X \in \{A, B\}$, que representan el tiempo estimado para la vuelta i-ésima si esta vuelta es realizada con neumáticos de tipo X.

Al inicio de cada vuelta el automóvil puede hacer una parada opcional y cambiar de neumáticos, lo que insume un tiempo adicional estimado P. El tipo de neumático usado al comienzo de la carrera puede seleccionarse previamente.

Se desea determinar qué tipo de neumático usar en cada vuelta i, $1 \leq i \leq n$, de forma de minimizar el tiempo requerido para completar la carrera, incluyendo el tiempo insumido en todas las paradas que eventualmente se decida hacer.

(a) Dé un algoritmo **iterativo** eficiente que determina el mínimo tiempo posible para completar la carrera. Para esto **defina y justifique** claramente una relación de recurrencia que sirva como base para un algoritmo de programación dinámica.

Sugerencia: Considere una función de la forma $OPT(i, X)$.

(b) Dé un algoritmo que construye un arreglo de tamaño n, SOL, tal que $SOL[i]$ contiene el tipo de neumático a usar en la vuelta i, $1 \leq i \leq n$, para completar la carrera en tiempo mínimo.

Solución:

(a) Para un entero i, $1 \leq i \leq n$, y un tipo de neumático X, $X \in \{A, B\}$, definimos $OPT(i, X)$ como el tiempo mínimo para completar las primeras i vueltas realizando la última vuelta con un neumático de tipo X. Como el tipo de neumático para la largada se puede elegir previamente, para $i = 1$ no es necesario evaluar la conveniencia de invertir tiempo en realizar ningún cambio de neumáticos, por lo cual tenemos

$$OPT(1, X) = t(1, X), \quad X \in \{A, B\}. \tag{1}$$

Para cada paso i, $1 < i \leq n$, tenemos dos opciones: usamos los mismos neumáticos que en la vuelta anterior o la cambiamos. En cualquiera de los dos casos, si llamamos X al tipo de neumático que decidimos usar en el paso i, la i-ésima vuelta requiere un tiempo $t(i, X)$; en el segundo caso invertimos un tiempo adicional P por la parada para el cambio de neumáticos. Observamos que una solución óptima que usa neumáticos de tipo X en la vuelta i debe minimizar el tiempo consumido en las primeras $i - 1$ vueltas, y terminar usando la neumáticos de tipo X en el primer caso y del otro tipo en el segundo caso. Denotando $\bar{X}$ el tipo de neumático opuesto a X, vemos que la función OPT satisface

$$OPT(i, X) = t(i, X) + \text{mín}\{OPT(i-1, X), OPT(i-1, \bar{X}) + P\}, \quad 1 < i \leq n, X \in \{A, B\}. \tag{2}$$

El algoritmo para calcular el tiempo total mínimo se presenta en la figura 2.

```
1 Hacer OPT[1, X] = t(1, X), para X ∈ {A, B}
2 for i = 2 to n do
3     Hacer OPT(i, A) = t(i, A) + mín{OPT(i − 1, A), OPT(i − 1, B) + P}
4     Hacer OPT(i, B) = t(i, B) + mín{OPT(i − 1, B), OPT(i − 1, A) + P}
5 end
6 Devolver mín{OPT(n, A), OPT(n, B)}
```

Figura 2: Algoritmo para calcular el tiempo total mínimo para completar la carrera. Como producto secundario calcula $OPT(i, X)$, para $1 \leq i \leq n$, $X \in \{A, B\}$.

(b) El algoritmo de la figura 3 construye una solución óptima en un arreglo de tamaño n, SOL, donde el contenido de cada posición i, $1 \leq i \leq n$, indica el tipo de neumático que se usa para la vuelta i. El algoritmo utiliza el arreglo OPT producido por el algoritmo de la parte anterior.

```
1 if OPT(n, A) < OPT(n, B) then Hacer X = A else Hacer X = B
2 for i = n downto 2 do
3     Hacer SOL[i] = X
4     if OPT(i − 1, X̄) + P < OPT(i − 1, X) then Hacer X = X̄
5 end
6 Hacer SOL[1] = X
7 Devolver SOL
```

Figura 3: Algoritmo para construir una solución óptima.

Ejercicio 3 (30 puntos)

Sea $F = (V, E)$ un bosque, es decir, un grafo tal que cada una de sus componentes conexas es un árbol. Un *conjunto independiente* es un subconjunto S de V tal que ningún vértice de S es adyacente a otro vértice de S en F. En otras palabras, para todo $u \in S$, $v \in S$, se cumple que $\{u, v\} \notin E$. Una *hoja* de F es un vértice de grado 1. En este ejercicio analizamos un algoritmo ávido, presentado en la figura 4, para encontrar un conjunto independiente de tamaño máximo para F.

```
1 Algorithm Conjunto Independiente (F)
2     Hacer F̃ = F
3     while existe una hoja en F̃ do
4         Sea v una hoja de F̃ y sea u el único vértice adyacente a v en F̃
5         Eliminar u de F̃ (y todas sus aristas incidentes)
6     end
7     Sea S el conjunto de vértices de F̃
8     Devolver S
9 end
```

Figura 4: Algoritmo para obtener un conjunto independiente de tamaño máximo en un bosque.

Se puede usar, sin necesidad de demostrarlo, el siguiente enunciado.

Enunciado 1: Sea S' un conjunto independiente de tamaño máximo para F tal que $S' \setminus S$ es no vacío, donde S es el conjunto devuelto al finalizar una ejecución del algoritmo. Sea u el **primero** de los vértices de $S' \setminus S$ en ser eliminado en el paso 5 durante esa ejecución, y sea v la hoja de $\tilde{F}$ adyacente a u seleccionada junto con u en el paso 4. Se cumple que $S'' = (S' \setminus \{u\}) \cup \{v\}$ es un conjunto independiente para F.

(a) Demuestre que el algoritmo termina en una cantidad finita de pasos.

(b) En las mismas condiciones que en el enunciado 1, demuestre que S'' tiene el mismo tamaño que S'.

(c) En las mismas condiciones que en enunciado 1, demuestre que $v \in S$.

(d) Demuestre la corrección del algoritmo propuesto. Para esta parte puede usar, **sin necesidad de demostrarlo**, que el conjunto S devuelto por el algoritmo es un conjunto independiente.

Sugerencia: Utilice las partes b y c en un *argumento de intercambio* para mostrar que S es de tamaño máximo.

Solución:

El enunciado 1 de la letra se puede demostrar de la siguiente manera. Sea w un vértice adyacente a v en F, $w \neq u$.

1. Como w no es adyacente a v en $\tilde{F}$ (porque v es una hoja en $\tilde{F}$ y es adyacente a $u \neq w$ en $\tilde{F}$), w debe haber sido eliminado de $\tilde{F}$ en alguna iteración anterior durante la ejecución del algoritmo. Entonces, como no se agregan vértices a $\tilde{F}$, se cumple que $w \notin S$.
2. Como u es el primero de los vértices de $S' \setminus S$ en ser eliminado de $\tilde{F}$, tenemos que $w \notin S' \setminus S$.
3. Como $w \notin S$ y además $w \notin S' \setminus S$, debemos tener $w \notin S'$ y, por lo tanto, como $w \neq v$, se cumple que $w \notin S''$.

Los puntos anteriores, sumados a que $u \notin S''$, implican que ningún vértice de S'' es adyacente a v en F. Esto implica que S'' es un conjunto independiente para F, ya que los vértices de S'' distintos de v son también vértices de S' y por lo tanto no son adyacentes entre sí.

(a) Como en cada iteración del ciclo se elimina al menos un vértice de $\tilde{F}$, en una cantidad finita de iteraciones se cumple que $\tilde{F}$ ya no tiene hojas y, en consecuencia, el algoritmo termina.

(b) Como $u \in S'$ y S' es un conjunto independiente, tenemos $v \notin S'$ porque v es adyacente a u en $\tilde{F}$ y por lo tanto también en F. Por lo tanto S'' tiene el mismo tamaño que S' y en consecuencia es también un conjunto independiente de tamaño máximo para F.

(c) La eliminación de u hace que v tenga grado cero en $\tilde{F}$ y, como nunca se agregan aristas a $\tilde{F}$, no puede ser seleccionado en el paso 4 en ninguna de las iteraciones subsiguientes. En consecuencia v nunca es eliminado de $\tilde{F}$ y por lo tanto pertenece al conjunto devuelto S.

(d) Sea S' un conjunto independiente de tamaño máximo para F, distinto de S. Notar que el hecho de que S' sea de tamaño máximo y S sea conjunto independiente implica que $S' \setminus S$ es no vacío. Vamos a transformar S' en S en una cantidad finita de pasos, manteniendo en todo momento una solución óptima para el problema. En efecto, por el enunciado 1 y la parte b, sabemos que podemos retirar un vértice u de S' y agregar otro vértice v para obtener un conjunto S'' que también es un conjunto independiente de tamaño máximo para F. Más aún, por la parte c, la cardinalidad de $S'' \setminus S$ es estrictamente más chica que la de $S' \setminus S$. Siguiendo con este procedimiento, haciendo que S'' tome el rol de S', en una cantidad finita de pasos obtenemos un conjunto independiente de tamaño máximo para F, $\tilde{S}$, tal que $\tilde{S} \setminus S$ es vacío. Como S es un conjunto independiente para F y $\tilde{S}$ es de tamaño máximo, debemos tener en realidad $\tilde{S} = S$.

Tal como se afirma en la letra, se cumple que el conjunto S devuelto por el algoritmo es un conjunto independiente. Por la condición de salida del ciclo, al momento de terminar el algoritmo todos los vértices son de grado cero. Esto implica que todos los vértices que originalmente eran adyacentes a algún vértice de S (si los hubiera) fueron eliminados en alguna de las ejecuciones del paso 5. Por lo tanto, S es un conjunto independiente para el bosque F.