Examen de Programación 3
21 de diciembre de 2022

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)

Sea $G = (V, E)$ un grafo conexo con aristas con costo cuyos valores están en $\{1, -1\}$.

Se quiere etiquetar cada vértice con un valor en $\{1, -1\}$ de tal manera que para cada arista (u, v) las etiquetas de u y v son diferentes si y solo si el costo de la arista es -1.

(a) Diseñe un algoritmo cuyo tiempo de ejecución sea $O(|E|)$ que etiquete el grafo de esa manera, o que, si eso no es posible emita un mensaje indicándolo.

(b) Demuestre que el tiempo de ejecución del algoritmo es $O(|E|)$.

(c) Definimos el *costo-producto* de un camino como la multiplicación de los costos de sus aristas. Demuestre que si en G hay un ciclo cuyo `costo-producto` es negativo entonces no se puede realizar el etiquetado propuesto.

Observación: si $c_{(u,v)}$ es el costo de la arista (u, v), el etiquetado de los vértices debe ser tal que etiqueta$[v]$ = etiqueta$[u] \times c_{(uv)}$; como consecuencia si $c = (v_1, \ldots, v_h)$ es un camino, entonces el etiquetado debe hacer que etiqueta$[v_h]$ = etiqueta$[v_1] \times$ costo-producto(c).

Importante: aunque base su solución en algoritmos vistos en el curso, debe incluir su implementación y la demostración de su tiempo de ejecución en sus respuestas.

Solución:

(a) El costo de un camino es -1 si y solo si la cantidad de aristas con costo -1 es impar.

```
1  Algorithm HayCicloCostoMenosUno (G = (V,E))
2      etiqueta[u] ← 0 para cada u
3      Sea v cualquier vértice de V
4      Inicializar cola Q ← (v)
5      etiqueta[v] ← 1
6      while Q no es vacía do
7          u ← inicio de Q, desencolar inicio de Q
8          foreach v adyacente a u do
9              if etiqueta[v] = 0 then
10                 etiqueta[v] = etiqueta[u] × costo_(uv)
11                 encolar v en Q
12             else
13                 if etiqueta[v] ≠ etiqueta[u] × costo_(uv) then
14                     TERMINAR indicando que hay ciclo de costo -1
15     Indicar que no hay ciclo de costo -1
16 end
```

(b) La inicialización de las etiquetas de los vértices es $O(|V|)$.

Cada vértice es encolado, y por lo tanto desencolado y explorado, a lo sumo una vez porque solo se encola si su etiqueta tiene el valor inicial 0 y al encolarse se modifica y no se vuelve a cambiar. Por lo tanto la cantidad de operaciones de Q es $O(|V|)$. Como una cola se puede implementar de forma que cada operación es $O(1)$ el total del tiempo de las operaciones de Q es $O(|V|)$.

Como el grafo es conexo se cumple $|V| = O(|E|)$. Entonces el tiempo de las anteriores operaciones es$O(|E|)$.

En cada iteración del ciclo externo se procesan las aristas que iniciden en el vértice desencolado y solo ellas. Por lo tanto cada arista se procesa a lo sumo 2 veces, estableciendo $2|E|$ como cota superior. Implementando el grafo mediante listas de adyacencia acceder el acceso a cada arista es $O(1)$. Para cada arista se hacen operaciones de comparación y aritméticas en $O(1)$. Por lo tanto el tiempo de procesar G es $O(|E|)$.

Entonces hemos descompuesto el conjunto de operaciones en una cantidad $O(1)$ de subconjuntos cada uno de los cuales es $O(|E|)$, por lo que el tiempo de ejecución es $O(|E|)$.

(c) El costo de un camino es 1 o -1.

Según la observación si $c = (v_1, \ldots, v_h)$ es un camino entonces el etiquetado debe hacer que etiqueta$[v_h] =$ etiqueta$[v_1] \times$ costo-producto(c). Si c es un ciclo v_1 es igual a v_h lo que implica que si costo-producto(c) no es 1 entonces no se puede hacer la asignación. Por lo tanto si hay un ciclo con coato-producto negativo, o sea -1, no se puede hacer la asignación.

Ejercicio 2 (30 puntos)

Sean x e y enteros de n bits.

(a) Diseñe un algortimo que multiplique x por y usando la estrategia *Dividir y Conquistar*. El tiempo de ejecución debe ser $O(n^{\log 3})$.

Se puede asumir que n es potencia de 2.

(b) Exprese la relación de recurrencia del tiempo de ejecución indicando la procedencia de cada término y resuélvala.

Reescriba cualquier argumento o algoritmo que use del libro de referencia del curso.

Sugerencias:

- Tenga en cuenta que si $n > 1$ entonces x e y se pueden expresar como $x = x_1 2^{n/2} + x_0$ e $y = y_1 2^{n/2} + y_0$, donde x_1, x_0, y_1 e y_0 son enteros de $n/2$ bits, y se cumple la ecuación $(x_1 + x_0)(y_1 + y_0) = x_1 y_1 + (x_1 y_0 + x_0 y_1) + x_0 y_0$.
- Considere la posibilidad de que el tiempo de ejecución cumple $T(n) \leq kn^{\log 3} - dn$ para algunos k y d.

Solución:

(a) El algoritmo de la Figura **??** resuelve el problema utilizando la primera de las sugerencias.

```
1  Algorithm MultiplicacionEnteros (x, y, n)
2      if n > 1 then
3          Sean x = x_1 2^{n/2} + x_0 e y = y_1 2^{n/2} + y_0
4          r_2 ← MultiplicacionEnteros (x_1, y_1, n/2)
5          r_0 ← MultiplicacionEnteros (x_0, y_0, n/2)
6          r_1 ← MultiplicacionEnteros ((x_1 + x_0), (y_1 + y_0), n/2) − r_2 − r_1
7          return r_2 2^n + r_1 2^{n/2} + r_0
8      else
9          return x · y
10 end
```

Figura 1: Multiplicación x por y.

(b) El tiempo de ejecución cumple la relación de recurrencia (**??**). El término $3T(n/2)$ corresponde a las tres llamadas con instancias de tamaño $n/2$. El término $c_g n$ proviene de que hay que hacer una cantidad fija de sumas y restas que tienen tiempo $O(n)$. El término c_b corresponde al caso base, que consiste en multiplicar dos bits.

$$T(n) \leq \begin{cases} c_b & \text{si } n = 1, \\ 3T(n/2) + c_g n & \text{en otro caso.} \end{cases} \tag{1}$$

Vamos a demostrar que el tiempo de ejecución es $O(n^{\log 3})$, y por lo tanto $o(n^2)$. Consideremos la conjetura $T(n) \leq kn^{\log 3} - dn$ donde k y d se deben determinar.

Vamos a probar que se cumple por inducción en n.

Por el caso base se cumple $T(1) \leq c_b$. Entonces se deben elegir k y d de modo que se cumpla

$$c_b \leq k - d. \tag{2}$$

Asumimos como hipótesis inductiva que la cojuetura se cumple para valores menores a n, en particular $n/2$. Entonces, por la relación (**??**) se cumple

$$T(n) \leq 3k(n/2)^{\log 3} - 3d(n/2) + c_g n$$
$$= kn^{\log 3} - d\,n + n(c_g - d/2)$$

que es menor o igual a $kn^{\log 3} - d\,n$ si

$$c_g - d/2 \leq 0. \tag{3}$$

Entonces para que se cumpla la conjetura hay que elegir d y k que cumplan (**??**) y (**??**):

$$d \geq 2c_g$$
$$k \geq c_b + d.$$

Notar que la conjetura $T(n) \leq kn^{\log 3}$ no hubiera servido ya que en

$$T(n) \leq 3k(n/2)^{\log 3} + c_g n$$
$$= kn^{\log 3} + c_g n$$

queda el término $c_g n$ que no permite probar el paso inductivo.

Ejercicio 3 (35 puntos)

Sean X, Y dos conjuntos y R una relación entre ellos (o sea, un subconjunto del producto cartesiano $X \times Y$). Sin pérdida de generalidad podemos suponer que $X = \{1 \dots |X|\}$, e $Y = \{1 \dots |Y|\}$.

El problema *Relación a Función*, `RF`, consiste en, dados una relación R y un entero k, determinar si existe un subconjunto de al menos k elementos de Y tal que la restricción de R definida por ese subconjunto es una función parcial de X a ese subconjunto de Y (o sea, para cada elemento de X hay **a lo sumo** un elemento del subconjunto de Y).

Por ejemplo, en la siguiente tabla se representa con 1 la pertenencia a la relación. Esa relación con $k = 2$ es una instancia `SÍ` porque $\{y2, y3\}$ cumplen que, para todo x, $R(x, y_2)$ o $R(x, y_3)$ o ambos son 0.

	x1	x2	x3	x4
y1	0	1	0	1
y2	1	1	0	0
y3	0	0	0	1
y4	1	0	0	1

Demuestre que *RF* es NP-Completo. La relación R es dada en un arrgelo bidimensional en el que las filas representan los elementos de Y y las columnas los elementos de X. **Debe utilizar *Independent Set* como problema de referencia.**

Importante: Cuando se requiera demostrar órdenes de tiempo de ejecución o de espacio para procesos o estructuras, es suficiente con enunciarlos (sin demostrarlos).

Solución:

Para demostrar que *RF* es NP-Completo se debe demostrar que:

- *RF* es $\mathcal{NP}$;
- todo problema en $\mathcal{NP}$ se puede reducir en tiempo polinomial a *RF*. Demostraremos que *Independent Set*, que sabemos que es NP-Completo, se puede reducir a *RF*, con lo cual por propiedad transitiva quedará demostrado el punto.

Demostración de que *RF* es $\mathcal{NP}$ Vamos a demostrar que existe un algoritmo certificador eficiente, B, del problema *RF*. El algoritmo tiene como entrada la instancia s y un certificado t y su tiempo de ejecución es polinomial. El tamaño de t está acotado superiormente por un polinomio en el tamaño de s, $|t| \leq p(|s|)$, donde p es un polinomio. Si el resultado de la ejecución de B es `SÍ` entonces s es una instancia `SÍ`. Si s es una instancia `SÍ` entonces existe al menos un certificado t con el cual el resultado de la ejecución de B es `SÍ`.

La instancia s es un arreglo bidimensional R de $|Y|$ filas y $|X|$ columnas de ceros y unos, y un número k. El certificado t es un string de $|Y|$ ceros y unos. Como el tamaño de la instancia s es $\Omega(|X| \times |Y|)$ el tamaño de t es polinomial en el tamaño de s.

El algoritmo verifica que la cantidad de unos de t es mayor o igual a k. En caso de serlo ejecuta un ciclo en el que en cada iteración se procesa una columna del arreglo R que corresponda a un elemento x de X. En la iteración se verifica que en la columna haya a lo sumo un 1 en las filas correspondientes a los indices de t cuyo valor es 1. Devuelve `SÍ` si y solo si lo anterior se cumple para todas las columnas.

```
1 Algorithm B ((R,k),t)
2     Si la cantidad de unos en t es menor a k termina sin certificar
3     foreach x ∈ 1..|X| do
4         Si hay más de un y ∈ 1..|Y| tal que t[y] = 1 y R[x,y] = 1 termina sin certificar
5     end
6     return SI
7 end
```

La ejecución se cumple en tiempo polinomial en $|X| \times |Y|$.

Para demostrar la corrección del algoritmo vamos a interpretar que el conjunto $\mathcal{I}$ de índices en los que el valor de t es 1 se corresponde con un subconjunto de Y que cumple con el enunciado del problema.

Supongamos que el certificador devuelve `SÍ`. Eso significa que t tiene al menos k unos, $|\mathcal{I}| \geq k$, y que para cada columna y de R que corresponde a un $y \in \mathcal{I}$ hay a lo sumo un x que cumple $R[x, y] = 1$. Por la defínción del arreglo R esto impica que hay al menos k elementos $y \in Y$ para cada uno de los cuales hay a lo sumo un (x, y) en la relación R. Por lo tanto s es una instancia `SÍ`.

Supongamos que s es una instancia `SÍ` de `RF`. Entonces hay un subconjunto $\mathcal{I}$ de al menos k elementos de Y tal que para cada uno de ellos hay a lo sumo un elemento de X con el cual está en R. Si se una como certificado t un string cuyo valor es 1 exactamente en los índices $\mathcal{I}$, la ejecución del algortimo llegará al ciclo porque la cantidad de unos de t no es menos a k, y por la definición del arreglo R en cada iteración del ciclo a lo sumo se encontrará un 1. Por lo tanto el certificador devuelve `SÍ`.

Demostración de que *Independent Set* se puede reducir en tiempo polinómico a *RF* Se mostrará que existe una reducción polinomial del problema NP-completo *Independent Set*, (en adelante *IS*) a *RF*. O sea, que

- una instancia de *IS* se transforma en tiempo polinomial en una instancia de *RF*,
- una instancia `SÍ` de *IS* se transforma en una instancia `SÍ` de *RF*,
- una instancia `NO` de *IS* se transforma en una instancia `NO` de *RF*.

Una instancia (G, k) de *IS* se transforma en una instancia (R, k') de *RF* de la siguiente forma:

- El conjunto Y de *RF* es igual al conjunto de vértices de G y el conjunto X de *RF* es igual al conjunto de aristas de G.
- Por cada arista (u, w) de G se incluyen $((u, w), u)$ y $((u, w), w)$ en R.
- No hay ningún otro elemento en R.
- Se toma $k' = k$.

Esta transformación se hace en tiempo polinomial.

Demostramos que una instancia (G, k) `SÍ` de *IS* se tranforma en una instancia (R, k') `SÍ` de *RF*. Si (G, k) es una instancia `SÍ` significa que existe un subconjunto U de vértices de tamaño mayor o igual a k tal que no hay aristas entre ningún par de sus vértices. Sea U' el subconjunto de Y correspondiente. Sea cualquier elemento $x = (u, w)$ de X. Por construcción, los únicos elementos de R que tienen (u, w) como primer componente son $((u, w), u)$ y $((u, w), w)$. Como (u, w) corresponde a una arista de G y entre los elementos de U no hay aristas, al menos uno de u y w no pertenece a U. Entonces para cualquier x hay a lo sumo un elemento y de U' tal que (x, y) pertenece a R. Además el tamaño de U' es mayor o igual a k' por construcción de k'. Por lo tanto (R, k') es una instancia `SÍ` de *RF*.

Demostramos que una instancia (G, k) `NO` de *IS* se tranforma en una instancia (R, k') `NO` de *RF*.
Supongamos que (G, k) es una instancia `NO` de *IS*. Sea U' un subconjunto de Y de k' elementos. El conjunto U de G que le corresponde tiene, por construcción de k', k elementos. Como (G, k) es una instancia `NO` hay una arista (u, w) tal que u y w pertenecen a U. Por lo tanto, por construcción, (u, v) es un elemento de X, u y v son elementos de U' y $((u, w), u)$ y $((u, w), w)$ pertenecen a R. Entonces para cualquier subconjunto U' de Y de k' elementos existe un $x = (u, w)$ en X y elementos u y w en U' tal que (x, u) y (x, w) pertenecen a R. Por lo tanto (R, k') es una instancia `NO` de *RF*.