Examen de Programación 3
10 de febrero de 2020

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 $A = \{a_1, \ldots, a_n\}$ un conjunto y sea $B_1, B_2, \ldots, B_m$ una colección de subconjuntos de A, es decir, $B_i \subseteq A$ para todo i, $1 \leq i \leq m$. Decimos que un conjunto $S \subseteq A$ *golpea* a $B_1, B_2, \ldots, B_m$ si S contiene al menos un elemento de cada B_i, es decir, si para todo i, $1 \leq i \leq m$, se cumple que $S \cap B_i \neq \emptyset$.

Definimos el *Problema del Conjunto Golpeador (PCG)* como sigue: Dados un conjunto finito A, una colección $B_1, B_2, \ldots, B_m$ de subconjuntos de A, y un número natural k, ¿existe un conjunto $S \subseteq A$, con $|S| \leq k$, que golpea a $B_1, B_2, \ldots, B_m$?

Por ejemplo, el conjunto $A = \{1, 4, 5, 9\}$, con la colección de subconjuntos $B_1 = \{1, 5\}, B_2 = \{1, 4\}, B_3 = \{9\}$ y el natural $k = 2$, definen una instancia SÍ de *PCG*, porque el conjunto $S = \{1, 9\}$ golpea a B_1, B_2, B_3 y tiene cardinalidad 2, que no es mayor a k.

(a) Demuestre que *Vertex Cover* $\leq_p$ *PCG*.

(b) Defina la clase de complejidad $\mathcal{NP}$.

(c) Asumiendo que $PCG \in \mathcal{NP}$ (no es necesario demostrarlo), demuestre que *PCG* es $\mathcal{NP}$-completo. Puede usar sin demostración resultados contenidos en el material teórico del curso.

Solución:

(a) Dada una instancia (G, k) de *Vertex Cover*, con $G = (V, E)$, generamos una instancia $(A, B_1, ..., B_m, k')$ de *PCG* de la siguiente manera:

- Definimos $A = V$.
- Para cada arista $\{u, v\} \in E$ agregamos el conjunto $B_i = \{u, v\}$.
- Definimos $k' = k$.

La transformación se puede implementar de modo que el tiempo de ejecución sea polinomial en el tamaño de la instancia de *Vertex Cover*. Por ejemplo, usando una representación de listas de adyacencia para G, con $V = \{1, 2, \ldots, n\}$, los conjuntos $A, B_1, B_2, \ldots, B_m$ se pueden construir en una sola pasada por el grafo G:

1. Para cada $v \in V$:
   a) Agregar v a A
   b) Para cada vértice w adyacente a v con $w > v$ (esta condición evita duplicar conjuntos), agregar un nuevo conjunto B_i a la colección, que definimos como $B_i = \{v, w\}$.

El conjunto A podría estar representado implícitamente por su tamaño, n, asumiendo que A está formado por todos los naturales entre 1 y n (como hacemos de hecho para representar el conjunto de vértices de un grafo). Representando los conjuntos $B_1, B_2, \ldots, B_m$ mediante listas enlazadas, por ejemplo, este algoritmo de transformación se ejecuta en tiempo $O(m + n)$, que es polinomial en el tamaño de la representación de G.

Resta ver que la transformacion es correcta, es decir que (G, k) es una instancia SÍ de *Vertex Cover* si y solo si el resultado de aplicar la transformación a (G, k), $(A, B_1, ..., B_m, k')$, es instancia una SÍ de *PCG*.

Supongamos que (G, k) es instancia SÍ de *Vertex Cover*. Entonces, existe una cobertura de aristas por vértices para G, $S \subseteq V$, con $|S| \leq k$. Observamos que $S \subseteq A$, porque $A = V$ por definición, y también se cumple $|S| \leq k'$, porque $k' = k$. Vamos a mostrar que además S golpea a $B_1, ..., B_m$,

lo cual demuestra que $(A, B_1, ..., B_m, k')$ es una instancia SÍ de *PCG*. En efecto, como todo conjunto $B_i = \{u, v\}$ en la instancia de *PCG* es por definición una arista de G, y S es una cobertura de aristas por vértices para G, se cumple que $u \in S$ o $v \in S$, por lo cual S golpea B_i.

Supongamos ahora que la instancia $(A, B_1, ..., B_m, k')$ obtenida mediante la transformación es una instancia SÍ de *PCG*. Entonces existe $S \subseteq A$, $|S| \leq k'$, tal que S golpea a $B_1, ..., B_m$. Observamos que S es un conjunto de vértices de G de tamaño no superior a k. Vamos a mostrar que además S es una cobertura de aristas por vértices para G, lo cual implica que (G, k) es instancia SÍ de *Vertex Cover*. Por definición de la transformación, cada arista $\{u, v\} \in E$ genera un conjunto $B_i = \{u, v\}$ que, como es golpeado por S, satisface que $u \in S$ o $v \in S$. Por lo tanto todas las aristas de G tienen un extremo en S y concluimos que S es una cobertura de aristas por vértices para G.

(b) Ver material teórico del curso.

(c) Tenemos que *PCG* pertenece a $\mathcal{NP}$ y además, por la primera parte, se cumple que *Vertex Cover* $\leq_p$ *PCG*. Por K&T 8.16 sabemos que *Vertex Cover* es NP-completo, de donde *PCG* también lo es por la definición de problema NP-completo y la transitividad de $\leq_p$ (K&T 8.14).

Ejercicio 2 (30 puntos)

En este ejercicio extendemos el algoritmo DFS para grafos **dirigidos**, de forma tal que **durante** la recorrida de un grafo dirigido **fuertemente conexo** G, **todas** las aristas de G son etiquetadas según tres categorías:

- **H**: aristas (u, v) donde v es hijo de u en el árbol DFS resultante.
- **A**: aristas (u, v) donde v es ancestro de u en el árbol DFS resultante.
- **O**: aristas (u, v) donde v no es hijo ni ancestro de u en el árbol DFS resultante.

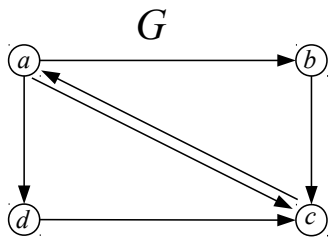


Figura 1: Grafos dirigidos de ejemplo para el ejercicio 2.

(a) Considere una recorrida DFS del grafo dirigido G de la figura 2 que comienza en el vértice a, explora a b en segundo lugar, y continúa explorando los demás vértices en orden compatible con el algoritmo DFS. Dibuje un árbol DFS para esta recorrida, identificando el orden de exploración de los vértices. Clasifique cada una de las seis aristas de G según las tres categorías definidas anteriormente.

(b) Adapte el algoritmo DFS para que etiquete todas las aristas de un grafo dirigido **fuertemente conexo** G según las tres categorías propuestas. El etiquetado debe realizarse **durante** la recorrida DFS de G (no posteriormente). Aunque no es necesario demostrarlo, el algoritmo debe admitir una implementación cuyo tiempo de ejecución es $O(m + n)$, donde m y n son la cantidad de aristas y vértices de G, respectivamente. Reescriba cualquier algoritmo que utilice de los estudiados en el curso.

Solución:

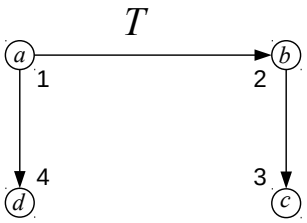


Figura 2: Árbol y orden de recorrida DFS para el ejercicio 2.

(a) La figura 3 muestra el árbol DFS pedido; el orden de recorrida se especifica mediante un número adyacente a cada vértice en el dibujo. Las aristas de G se clasifican de la siguiente manera:

- **H**: aristas $(a, b), (b, c), (a, d)$.
- **A**: arista (c, a).
- **O**: aristas $(a, c), (d, c)$.

(b) La figura 4 presenta un algoritmo que resuelve el problema.

```
1  Algorithm Clasificar
2      Hacer visitado [u] = false para todo u ∈ V
3      Hacer terminado [u] = false para todo u ∈ V
4      Invocar DFS(s), s ∈ V
5  end
6  Procedure DFS(u)
7      visitado [u] = true
8      foreach arista (u, v) saliente de u do
9          if not visitado [v] then
10             Hacer etiqueta [(u, v)] = H
11             Invocar DFS(v)
12         else if not terminado [v] then
13             Hacer etiqueta [(u, v)] = A
14         else
15             Hacer etiqueta [(u, v)] = O
16         end
17     end
18     terminado [u] = true
19 end
```

Figura 3: Modificación del algoritmo DFS para clasificar aristas.

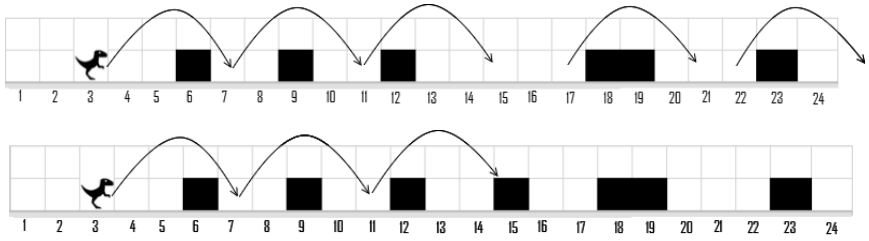
Las aristas de tipo **H** son las que se incluyen en el árbol, las cuales se detectan a medida que la recorrida DFS avanza a través de las llamadas recursivas. El resto de las aristas son aquellas por las cuales el algoritmo no avanza porque visitado$[v]$ vale true. En este caso, v es ancestro de u en el árbol si, equivalentemente, u es descendiente de v, lo cual implica que se llega a la invocación DFS(u) a través de una cadena de llamadas recursivas a partir de la invocación DFS(v). Esta situación se detecta a través de la bandera terminado, que señaliza el fin de cada invocación a DFS.

Ejercicio 3 (35 puntos)

Un grupo de estudiantes desarrolló un videojuego donde un dinosaurio debe atravesar un camino compuesto por n baldosas contiguas, donde cada una puede estar libre o tener un obstáculo. En cada paso, si el dinosaurio se encuentra en la baldosa i, podrá optar por moverse a la siguiente baldosa, $i+1$, o dar un salto hacia la baldosa $i+4$.

El jugador gana si logra que el dinosaurio atraviese el camino completo sin caer en ningún obstáculo.

Los estudiantes desean saber si para la disposición elegida de los obstáculos existe una secuencia de movimientos y saltos que permite ganar el juego. En la siguiente figura se muestran dos ejemplos, en los que se representaron únicamente los saltos. El primero de ellos muestra una secuencia ganadora, mientras que para el segundo ejemplo no existe una secuencia que permita ganarlo. Notar que, como en el primer ejemplo, una secuencia ganadora no necesariamente debe caer en la baldosa n; alcanza con saltar más allá de ella.



(a) Defina y **justifique** una relación de recurrencia que determine si, para una disposición de los obstáculos dada, es posible ganar el juego si se parte desde la primera baldosa.
Sugerencia: Considere una función booleana de la forma `M(i)` que determina si es posible ganar el juego partiendo desde la baldosa i.

(b) Dé un algoritmo para determinar, en caso de existir, una secuencia de movimientos y saltos que permita ganar el juego partiendo desde la primera baldosa. El algoritmo debe admitir una implementación cuyo tiempo de ejecución es polinomial en n (no es necesario demostrarlo).

Solución:

(a) Representamos la disposición de obstáculos en el juego mediante un arreglo `B(i)`, $1 \leq i \leq n$, que indica si la baldosa i está libre o tiene un obstáculo.

A continuación veremos cómo calcular la función booleana `M(i)`, que determina si es posible ganar el juego partiendo desde la posición i. Como en su último movimiento el dinosaurio puede potencialmente saltar hasta 4 posiciones más allá de la última baldosa (si salta desde una baldosa en posición mayor que $n-4$), extendemos la definición de M para valores de i mayores que n. Concretamente, como no hay obstáculos que sortear más allá de la baldosa n, definimos

$$\mathrm{M}(i) = \texttt{True}, \quad n < i \leq n+4. \tag{1}$$

Observar que resolviendo este problema más general, la solución de nuestro problema original está dada por el valor de `M(1)`. Para esto, consideremos la baldosa i desde la que parte el dinosaurio, siendo $1 \leq i \leq n$. Para que exista posibilidad de ganar, la baldosa i debe estar libre, de lo contrario el juego está perdido. De cumplirse lo anterior, es decir si `B[i] = Libre`, el dinosaurio puede optar ahora por avanzar a la siguiente baldosa o dar un salto. Si decide avanzar, pasará a estar en la baldosa $i+1$, por lo que ahora resta determinar si podemos ganar partiendo desde la nueva posición, lo cual esta dado por el valor de $M(i+1)$. Por el contrario, si decide dar un salto, resta ver si podemos ganar el juego partiendo desde la baldosa $i+4$, lo cual esta dado por $M(i+4)$.

Los dos escenarios anteriores constituyen todos los casos posibles y, puesto que podemos elegir cualquiera de ellos, concluimos que es posible ganar el juego si y solo si es posible hacerlo con alguna de estas dos alternativos. Por lo tanto tenemos

$$\mathrm{M}(i) = \Big(\texttt{B[i]} = \texttt{Libre}\Big) \wedge \Big(\mathrm{M}(i+1) \vee \mathrm{M}(i+4)\Big), \quad 1 \leq i \leq n\,, \tag{2}$$

que junto con (1) definen una relación de recurrencia para M.

(b) El algoritmo de la figura 5 construye una secuencia ganadora, si existe, o responde que no hay solución en caso contrario.

```
1  Hacer M[i] = true para n < i ≤ n + 4
2  for i = n downto 1 do
3      Hacer M[i] = (B[i] = libre) and (M[i + 1] or M[i + 4])
4  end
5  if not M[1] then Responder que no existe solución y terminar
6  Hacer SOL = secuencia vacía
7  Hacer i = 1
8  while i ≤ n do
9      if M[i + 4] then
10         Agregar saltar a SOL y sumar 4 a i
11     else
12         Agregar avanzar a SOL y sumar 1 a i
13     end
14 end
```

Figura 4: Algoritmo para construir una secuencia ganadora o determinar que no existe forma de ganar.