Parcial de Programación 3
30 de noviembre 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 (18 puntos)

El equipo de Fórmula 1 *GreenCow* debe planificar la estrategia de carrera. Una carrera consiste en n vueltas a un circuito. Como los neumáticos se degradan, el tiempo por vuelta con un mismo conjunto de neumáticos empeora a medida que pasa el tiempo. A su vez, a lo largo de la carrera el rendimiento del automóvil cambia a medida de que consume combustible y es más liviano, lo que impacta en el tiempo por vuelta. Al final de cualquier vuelta, se puede parar y hacer un cambio de neumáticos. Debe decidirse cuándo hacerlo para minimizar el tiempo total de carrera.

Mediante simulaciones, se estimó un conjunto de valores $s_{i,j}$ ($1 \leq j \leq i \leq n$) que indican el tiempo estimado de vuelta del automóvil en la vuelta i-ésima de la carrera, con neumáticos que tienen j vueltas de uso.

Si se para el automóvil para cambiar neumáticos al final de una vuelta, se estima que se perderá un tiempo P. No es obligatorio parar durante la carrera ni hay un límite de paradas (además del límite obvio, dado que se puede parar a lo sumo una vez por vuelta).

A modo de ejemplo, en la tabla de la Figura se esquematizan tres estrategias para una carrera de 10 vueltas: sin parar, parando en la vuelta 3, o parando en las vueltas 2,4,6, y 8. El tiempo total para cada estrategia es la suma de los valores de cada fila.

paradas \vuelta	1	2	3	4	5	6	7	8	9	10
ninguna	$s_{1,1}$	$s_{2,2}$	$s_{3,3}$	$s_{4,4}$	$s_{5,5}$	$s_{6,6}$	$s_{7,7}$	$s_{8,8}$	$s_{9,9}$	$s_{10,10}$
3	$s_{1,1}$	$s_{2,2}$	$s_{3,3} + P$	$s_{4,1}$	$s_{5,2}$	$s_{6,3}$	$s_{7,4}$	$s_{8,5}$	$s_{9,6}$	$s_{10,7}$
2,4,6,8	$s_{1,1}$	$s_{2,2} + P$	$s_{3,1}$	$s_{4,2} + P$	$s_{5,1}$	$s_{6,2} + P$	$s_{7,1}$	$s_{8,2} + P$	$s_{9,1}$	$s_{10,2}$

Figura 1: Tiempos por vuelta para diferentes estrategias.

(a) Especifique una relación de recurrencia que permita calcular el tiempo mínimo para una carrera de n vueltas.

Sugerencia: Puede utilizar una función $OPT(i, j)$ que represente el mínimo tiempo necesario para recorrer desde la vuelta i hasta la vuelta n, donde j es la cantidad de vueltas recorridas desde el último cambio de neumáticos.

(b) Dé un **algoritmo eficiente iterativo** usando la técnica de programación dinámica para calcular la recurrencia de la parte anterior, que retorne el valor del tiempo óptimo de carrera.

(c) Dé un algoritmo que devuelva una estrategia de carrera. Esto es, una lista de enteros representando las vueltas en las que se debe parar para que el tiempo de carrera estimado sea óptimo. Puede asumir que tiene disponibles las estructuras que haya construido en las partes anteriores.

Solución:

Este ejercicio es similar a un ejercicio de práctico (Kleinberg & Tardos, Ex. 6.9). La sugerencia lleva a resolverlo con la estrategia que se siguió en el práctico. Hay otras formas de hacerlo. Presentamos una solución alternativa.

(a)

- **Solución siguiendo la sugerencia de la letra:**

  Para la vuelta n-ésima, el tiempo óptimo viene dado por:

  $$OPT(n, j) = s_{n,j} \ (1 \leq j \leq n)$$

  En cualquier otra vuelta i-ésima de la carrera, podemos considerar dos casos: parando al final de la vuelta, o no.

  Si paramos al final de la vuelta i, el tiempo a considerar será el costo de la vuelta i-ésima (el costo $s_{i,j}$, agregado al costo P de parar), sumado al tiempo óptimo desde la siguiente vuelta en adelante:

  $$OPT(i, j) = s_{i,j} + P + OPT(i + 1, 1)$$

  Si no se para en la vuelta i, calculamos de forma similar:

  $$OPT(i, j) = s_{i,j} + OPT(i + 1, j + 1)$$

  Finalmente, el óptimo viene dado por:

  $$OPT(i, j) = s_{i,j} + \text{mín}\{P + OPT(i + 1, 1), OPT(i + 1, j + 1)\} \ (1 \leq j \leq i < n)$$

  El tiempo óptimo de carrera es $OPT(1, 1)$.

- **Solución alternativa:** Construiremos una función OPT_{aux} con semántica distinta a la sugerencia. $OPT_{aux}(i, j)$ denota el tiempo óptimo de carrera entre la vuelta 1 y la vuelta i-ésima, habiendo cambiado de neumáticos hace j vueltas (al completar la vuelta i-ésima). Consideramos la vuelta i-ésima, en los casos donde no se paró en la vuelta inmediátamente anterior, tenemos:

  $$OPT_{aux}(i, j) = OPT_{aux}(i - 1, j - 1) + s_{i,j} \ (1 < i \leq n, 1 < j \leq i)$$

  Cuando sí se paró en la vuelta inmediátamente anterior, tenemos:

  $$OPT_{aux}(i, 1) = P + s_{i,1} + \min_{1 \leq k \leq i-1} \{OPT_{aux}(i - 1, k)\}$$

  El caso base de la recurrencia viene dado por:

  $$OPT_{aux}(1, 1) = s_{1,1}$$

  Definimos la función $OPT(i)$ denotando el tiempo mínimo requerido para completar las primeras i vueltas. Se considera la mejor estrategia posible al margen de la duración del uso del último juego de neumáticos:

  $$OPT(i) = \min_{1 \leq k \leq i} \{OPT_{aux}(i, k)\} \ (1 < i \leq n)$$

  El tiempo óptimo de carrera es $OPT(n)$.

(b)

- **Se presenta la solución siguiendo la sugerencia de la letra:**

```
compute-opt(s[][], P, OPT[][]){
  for j from 1 to n {
    OPT[n][j] = s[n][j]
  }
  for i from n-1 downto 1 {
    for j from i downto 1 {
     OPT[i][j] = s[i][j] + min(P + OPT[i+1][1], OPT[i+1][j+1]);
    }
  }
  return OPT[1][1];
}
```

- **Solución alternativa:**

```
compute-opt(s[][], P, OPT_aux[][], OPT[]){
  OPT_aux[1][1] = s[1][1]
  for i from 2 to n {
       OPT_aux[i][1] = P + s[i][1] + mín_{1≤k≤i−1}{OPT_aux[i − 1,k]};

       for j from 2 to i {
         OPT_aux[i][j] = OPT_aux[i-1][j-1] + s[i][j];
       }
  }
  for i from 1 to n {
    OPT[i] = mín_{1≤k≤i}{OPT_aux[i][k]};
  }
  return OPT[n];
}
```

(c)

- **Se presenta la solución siguiendo la sugerencia de la letra:**

```
return-policy(OPT[][]){
  pol = empty();
  j = 1;
  for i from 1 to n {
    if OPT[i][j] == s[i][j] + P + OPT[i+1][1]
    then { pol.add(i); j=1;}
    else j++;
  }
  return pol;
}
```

- **Solución alternativa:**

```
return-policy(OPT_aux[][], OPT[]){
  pol = empty();
  min = OPT[n];
  for i from n to 1 {
    k=1;
    while min!=OPT_aux[i][k]
```

```
7           k++;
8       if k==1 then pol.add(i);
9       min = OPT_aux(i,k);
10    }
11    return pol;
12  }
```

Ejercicio 2 (14 puntos)

(a) Defina:

- red de flujo,
- flujo s-t,
- valor de flujo,
- corte s-t,
- capacidad de corte.

(b) Dada una red de flujo s-t, sean f un flujo s-t y (A, B) un corte s-t genéricos. Demuestre que el valor de f es menor o igual a la capacidad de (A, B). Mencione qué definiciones o propiedades usa en la demostración.

Puede asumir como demostrado que el valor del flujo es igual a la diferencia entre el flujo saliente de A, $f^{out}(A)$, y el flujo entrante de A, $f^{in}(A)$.

Solución:

Ver teórico

(a) 7.1

Una *red de flujo* es un grafo dirigido con un único nodo fuente, s, un único nodo sumidero, t y con aristas con una capacidad asociada que es un entero (se asumee en el libro; en general puede ser un real) no negativo. Se asume que todos los nodos tienen aristas incidentes. La capacidad de la ariste e se denota con c_e.

Un *flujo s-t* es una función f que asigna a cada arista e un real no negativo. Debe cumplir dos condiciones:

1. (Condición de capacidad). Para cada arista el flujo no puede ser mayor que la capacidad, $0 \leq f(e) \leq c_e$.
2. (Condición de conservación). Para cualquier nodo v, distinto de s y t, el total de flujo entrante es igual al total de flujo saliente: $\sum_{e\ in\ v} f(e) = \sum_{e\ out\ v} f(e)$.

El *valor del flujo*, denotado con $v(f)$, es el total de flujo saliente de s: $v(f) = \sum_{e\ out\ s} f(e)$.

7.2

Un *corte s-t* es una partición (A, B) de los vértices del grafo tal que s pertenece a A y t pertenece a B.

La *capacidad de corte* (A, B), denotada con $c(A, B)$, es la suma de las capacidades de las aristas salientes de A: $c(A, B) = \sum_{e\ out\ A} c_e$.

(b) (7.8)

Asumimos como demostrado que el valor del flujo es igual a la diferencia entre los flujos saliente y entrante de A.

Por definición el flujo en cada arista es no negativo por lo que el total de flujo entrante en A es mayor o igual a cero. Entonces, el valor del flujo es menor o igual al flujo saliente de A (la igualdad se da si y solo si el flujo entrante es 0). El flujo saliente de A es la suma de los flujos de las aristas salientes de A. Por la condición de capacidad el flujo en cada arista no puede ser mayor que la capacidad de la arista, por lo que el flujo saliente de A es menor o igual a la suma de las capacidad salientes, que es la capacidad del corte (A, B) (la igualdad se da si y solo si todas las aristas salientes están saturadas).

$$\begin{aligned} v(f) &= f^{out}(A) - f^{in}(A) \\ &\leq f^{out}(A) \\ &= \sum_{e \ out \ A} f(e) \\ &\leq \sum_{e \ out \ A} c_e \\ &= c(A, B). \end{aligned}$$

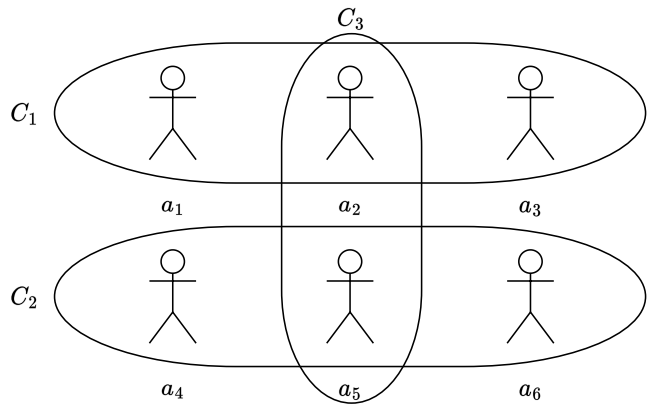
De lo menciondo, el valor del flujo es igual a la capacidad del corte si y solo si en cada arista entrante en A el flujo es 0, y en cada arista saliente de A el flujo es igual a la capacidad.

Ejercicio 3 (18 puntos)

Una empresa de software está comenzando un nuevo proyecto y necesita contratar a un grupo de personas calificadas para realizar un conjunto de tareas diferentes (liderar el proyecto, programar, testear, etc.). A partir de un estudio de los currículums de los aplicantes, la empresa define para cada tarea qué aplicantes están calificados para realizarla.

En este contexto, considere el problema de decisión *CONTRATAR* definido de la siguiente manera: Dado un conjunto de n aplicantes $A = \{a_1, \ldots, a_n\}$, un conjunto de m tareas $T = \{t_1, t_2, \ldots, t_m\}$, y un conjunto $C = \{C_1, \ldots, C_m\}$, con m subconjuntos de aplicantes $C_i \subseteq A$, donde los aplicantes de C_i están calificados para realizar la tarea t_i, ¿es posible contratar a lo sumo k aplicantes de forma que haya al menos uno calificado para cada una de las n tareas?

En la Figura 2 se presenta un ejemplo de una instancia **SI** del problema *CONTRATAR*, $(A, T, C, k = 2)$, donde el conjunto de aplicantes $\{a_2, a_5\}$ tiene al menos un aplicante calificado para realizar cada una de las tareas.



$A = \{a_1, a_2, a_3, a_4, a_5, a_6\}$

$T = \{t_1, t_2, t_3\}$

$C = \Big\{C_1 = \{a_1, a_2, a_3\}, C_2 = \{a_4, a_5, a_6\}, C_3 = \{a_2, a_5\}\Big\}$

Figura 2: Instancia de *CONTRATAR* que responde SI para $k = 2$.

(a) Demuestre que *CONTRATAR* es $\mathcal{NP}$-Completo. Enuncie cualquier argumento que utilice de los estudiados en el curso.

Sugerencia: Utilice *Vertex Cover* como problema de referencia.

Importante:

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

Solución:

(a) Para probar que un problema X es $\mathcal{NP}$-Completo debemos probar que $X \in NP$ y que $\forall Y \in NP, Y \leq_p X$. Como *Vertex Cover* es $\mathcal{NP}$-Completo, cumple que $\forall Y \in NP, Y \leq_p VertexCover$. En este sentido, por la propiedad (8.9) del libro, si mostramos que *Vertex Cover* $\leq_P$ *CONTRATAR* se cumple también que $\forall Y \in NP, Y \leq_p CONTRATAR$.

Primero mostramos que $CONTRATAR \in NP$. Para esto, definimos un certificado para una instancia (A, T, C, k) como un conjunto de aplicantes, definido mediante un arreglo de bits de tamaño n, donde el i-ésimo bit del arreglo es 1 si el aplicante a_i pertenece al conjunto, y 0 en otro caso.

Afirmamos que el algoritmo de la figura 3 es un certificador eficiente para *CONTRATAR*. El algoritmo recibe una representación binaria de una instancia (A, T, C, k) del problema y una tira de n bits que

representa un certificado Z.

```
1 Algorithm Certificador-CONTRATAR((A, T, C, k), Z)
2     if La cantidad de bits de Z que son 1 es mayor que k then return false
3     Sea T' un arreglo booleano de tamaño m
4     Inicializar cada posición de T' en false
5     foreach t_j en T do
6         foreach a_i en C_j do
7             if Z[i] == 1 then Hacer true la posición j de T'
8     if Todas las posiciones de T' son true then return true
9     else return false
```

Figura 3: Certificador eficiente para *CONTRATAR*.

Consideremos una instancia arbitraria (A, T, C, k) de *CONTRATAR*. Si (A, T, C, k) es una instancia SÍ, entonces existe un conjunto S de a lo sumo k aplicantes, donde existe al menos un aplicante calificado para realizar cada tarea. El certificado Z definido como un arreglo de bits donde el bit $i = 1$ si $a_i \in S$, 0 en otro caso, es de largo polinomial en el tamaño de la instancia, ya que su largo es n. Con este certificado, la condición del paso 2 del algoritmo certificador es falsa porque la cantidad de posiciones donde el bit $i = 1$ es menor o igual a k. Por otro lado, como el conjunto de aplicantes S tiene al menos un aplicante calificado para realizar cada una de las tareas, la condición del paso 7 del algoritmo se va a cumplir al menos una vez por cada posición de T'. De esta manera, la condición del paso 8 va a ser **true**, y el certificador va a retornar **true**.

Por otra parte, si existe una tira Z que hace que el certificador responda **true**, entonces la instancia es SÍ. Efectivamente, para que el certificador responda **true** debe cumplirse que Z tiene a los sumo k bits en 1, porque de lo contrario se devolvería **false** en el paso 2. Además, el subconjunto de aplicantes correspondiente a las posiciones de los bits en 1, debe tener al menos uno calificado para realizar cada una de las tareas, porque de lo contrario existiría al menos una posición del arreglo T' con valor **false** durante la ejecución del paso 8, lo que haría que el algoritmo certificador retornara **false**.

Concluimos que (A, T, C, k) es una instancia SÍ si y solo si existe un certificado Z de largo n (que es polinomial en el tamaño de la instancia) que hace que el algoritmo del la figura 3 responda **true** para las entradas (A, T, C, k), Z.

Respecto al tiempo de ejecución del algoritmo certificador, el paso 7 se ejecuta no más de $m \times n$ veces, mientras que los pasos 3, 4, y 8 se ejecutan en tiempo $O(N)$, y el resto en tiempo $O(1)$. Por lo tanto, el tiempo de ejecución del algoritmo certificador es polinomial en el tamaño de la entrada.

A continuación mostramos que *Vertex Cover* $\leq_P$ *CONTRATAR*, concluyendo la prueba. Consideramos una instancia arbitraria (G, k) de *Vertex Cover*, con $G = (V, E)$, $V = \{v_1, v_2, \ldots, v_n\}$, y $E = \{e_1, e_2, \ldots, e_m\}$. Construimos una instancia (A, T, C, k') de *CONTRATAR* haciendo corresponder los vértices de V con aplicantes y las aristas de E con tareas. Específicamente, definimos la siguiente transformación:

1. Para cada uno de los n vértices v_i, $1 \leq i \leq n$, se define un aplicante a_i. Definimos $A = \{a_1, \ldots, a_n\}$ como el conjunto de aplicantes.
2. Para cada una de las m aristas e_i, $1 \leq i \leq m$, se define una tarea t_i. Definimos $T = t_1, \ldots, t_m$ como el conjunto de tareas.
3. Para cada $t_i \in T$, correspondiente a la arista $e_i = (v_j, v_k)$ en G, $1 \leq i \leq m$, definimos un subconjunto de aplicantes calificados correspondiente $C_i = \{a_j, a_k\}$. Definimos $C = \{C_1, \ldots, C_m\}$.
4. Definimos $k' = k$.

Veamos que esta transformación se puede implementar de tal forma que requiere tiempo polinomial en el tamaño de la representación de la instancia de *Vertex Cover*. Si consideramos a G representado por un conjunto V de vértices y un conjunto E de aristas, el paso 1 se puede realizar en $O(n)$ recorriendo el conjunto de vértices V. Luego, tanto la construcción de las tareas t_i como la construcción

de los conjuntos C_i en los pasos 2 y 3 implican recorrer las aristas de E, lo cual requiere tiempo $O(m)$. Por lo tanto, concluimos que el tiempo total de ejecución de la transformación es polinomial en n y m, y por ende polinomial en el tamaño de la reprsentación de la instancia de *Vertex Cover*.

Vamos a probar que G tiene una Vertex Cover de a lo sumo k nodos si y solo si en la instancia construída de *CONTRATAR* es posible contratar un conjunto de a lo sumo k' aplicantes en donde haya al menos un aplicante calificado para cada una de las tareas.

Supongamos que G tiene un Vertex Cover $S = \{v_{i_1}, v_{i_2}, \ldots, v_{i_r}\}$, con $r \leq k$. Afirmamos que el conjunto de aplicantes $S' = \{a_{i_1}, a_{i_2}, \ldots, a_{i_r}\}$, cuyo tamaño r no supera k' porque $k' = k$, tiene un aplicante calificado para cada una de las tareas. En efecto, para j arbitrario, $1 \leq j \leq m$, el hecho de que S sea un Vertex Cover implica que la arista e_j es incidente a algún vértice $v_{i_q} \in S$. Por lo tanto, por la definición de C_j en el paso 3 de la transformación, se cumple que el aplicante a_{i_q} es un elemento del conjunto C_j y por lo tanto está calificado para realizar la tarea t_j. Como j es arbitrario, nuestra afirmación está probada.

Supongamos ahora que $S' = \{a_{i_1}, a_{i_2}, \ldots, a_{i_r}\}$ es un conjunto de aplicantes, $r \leq k$, donde al menos un aplicante está calificado para cada una de las tareas. Afirmamos que $S = \{v_{i_1}, v_{i_2}, \ldots, v_{i_r}\}$ es un Vertex Cover de G. Sea e_j, $1 \leq j \leq m$, una arista arbitraria de G. Como S' tiene al menos un aplicante calificado para cualquier tarea, existe un aplicante a_{i_q} que está calificado para realizar la tarea t_j. Por lo tanto, por la definición de C_j en el paso 3 de la transformación, se cumple que e_j es incidente a v_{i_q}, que es un vértice perteneciente a S. En consecuencia, como e_j es una arista arbitraria, S es un Vertex Cover de G.

Concluimos entonces que es posible resolver *Vertex Cover* a través de la transformación que acabamos de definir, que requiere tiempo polinomial, y una única invocación a un algoritmo que resuelve *CONTRATAR*. Esto demuestra que *Vertex Cover* $\leq_P$ *CONTRATAR*.