Parcial de Programación 3
30 de noviembre de 2021

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)

Un proveedor de conexión a Internet ofrece dos modalidades de cobro: tarifa por *consumo* y tarifa *plana*.

- La tarifa por consumo se puede contratar para cada día independientemente, y el costo para cada día es tK_c, donde t es el volumen de tráfico registrado en ese día y K_c es el costo por unidad de tráfico.
- La tarifa plana se contrata por bloques de 7 días consecutivos y tiene un costo fijo total K_p por esos 7 días.

Sea $T = t_1, t_2, \ldots, t_n$ una estimación del volumen de tráfico para los próximos n días. Una *planificación* para T es un conjunto P, $P \subseteq \{1, 2, \ldots n\}$, que determina los **días en los cuales se contrata tarifa plana.** Es decir, cada día en el que comienza un bloque de 7 días de tarifa plana.

Denotamos con $\bar{P}$ al conjunto de días que no están cubiertos por las contrataciones de tarifa plana indicadas por P, es decir, el conjunto de días i, $1 \leq i \leq n$, para los cuales ningún $p \in P$ cumple que $p \leq i < p + 7$. Una planificación P tiene un *costo* asociado,

$$C(P) = |P|K_p + \sum_{i \in \bar{P}} t_i K_c \,.$$

Una planificación es *óptima* si tiene costo mínimo entre todas las posibles planificaciones.

Notar que es posible contratar tarifa plana en cualquier día, aún cuando resten menos de 7 días para alcanzar el día final n.

(a) Defina una relación de recurrencia para una función que permita calcular el costo de una planificación óptima. Justifique explicando la procedencia de cada término.

(b) Escriba un algoritmo iterativo que utilice la técnica de Programación Dinámica para obtener el costo de una planificación óptima.

(c) Escriba un algoritmo que devuelva una una estructura de datos con la planificación óptima. Recuerde que una planificación solo contiende el día inicial de cada bloque de 7 días. Puede usar estructuras auxiliares obtenidas en la parte anterior sin reescribirlas.

Solución:

(a) Para $1 \leq i \leq n + 7$, definimos $OPT(i)$ como el costo de una planificación óptima para los días $i, i + 1, \ldots, n$. Cuando el conjunto de días en consideración es vacío, es decir, cuando $i > n$, el costo incurrido es nulo, lo que da origen al paso base de la recurrencia (1). En caso contrario, el primer día en consideración, es decir el día i, solo puede ser costeado o bien por la contratación de una tarifa plana, que tiene costo K_p, o bien pagando el consumo de ese día particular, que tiene un costo $t_i K_c$. Con la primera opción, todos los días hasta $\min\{i + 6, n\}$ quedan pagos, y la forma más económica de costear el resto de los días está dado por $OPT(i + 7)$, por definición de OPT. Con la segunda opción solo se paga el día i, y la forma más económica de costear el resto de los días está dado por $OPT(i + 1)$, por definición de OPT. Como estas son las únicas dos alternativas para cubrir el costo del primer día, el costo de una planificación óptima queda determinado por el mínimo costo entre las dos opciones, tal como establece (2).

$$OPT(i) = 0\,, \quad n < i \leq n + 7. \tag{1}$$

$$OPT(i) = \min\{t_i K_c + OPT(i + 1), K_p + OPT(i + 7)\}\,, \quad 1 \leq i \leq n\,, \tag{2}$$

(b) El algoritmo se presenta en la figura 1.

```
1 Algorithm CostoMínimo
2     Hacer OPT[i] = 0, para todo i, n < i ≤ n + 7
3     for i = n downto 1 do
4         OPT[i] = mín{t_i K_c + OPT[i + 1], K_p + OPT[i + 7]}
5     return OPT[1]
6 end
```

Figura 1: Algoritmo para determinar el costo de una planificación óptima

(c) El algoritmo se presenta en la figura 2.

```
1  Algorithm PlanificaciónÓptima
2      Crea un conjunto vacío P
3      Hacer i = 1
4      while i ≤ n do
5          if OPT[i] = K_p + OPT[i + 7] then
6              Agregar i a P
7              Hacer i = i + 7
8          else
9              Hacer i = i + 1
10     return P
11 end
```

Figura 2: Algoritmo para determinar una planificación óptima

Ejercicio 2 (18 puntos)

Un camino *hamiltoneano* en un grafo G es un camino que visita todos los vértices de G, pasando una y solo una vez por cada uno. Le llamamos *CH* al problema de decisión que consiste en determinar si en un grafo dado existe algún camino hamiltoneano. Por ejemplo, el grafo de la figura 3 es una instancia NO para el problema *CH*, porque no hay forma de recorrer todos los vértices sin pasar más de una vez por w. Sin embargo, si eliminamos el vértice t del grafo de la figura obtenemos una instancia SÍ, ya que por ejemplo el camino u, w, v es hamiltoneano. El problema *CH* es NP-Completo.

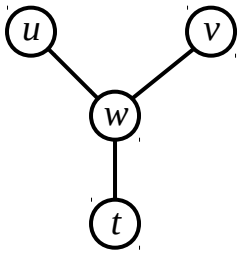

Figura 3: Grafo que no es hamitoneano. Si se elimina el vértice t, el resultado es un grafo que sí es hamiltoneano.

Un camino en un grafo es *simple* si no repite vértices. Le llamamos *CS* al siguiente problema de decisión: dado un grafo G y un natural k, ¿existe un camino simple de largo al menos k en G? (recuerde que el largo de un camino es la cantidad de aristas que contiene).

(a) Demuestre que $CH \leq_P CS$.

(b) Demuestre que *CS* es NP-Completo.

Solución:

(a) Definimos una reducción de *CH* a *CS* de la siguiente manera. Dada una instancia de *CH* determinada por un grafo G, construimos una instancia (G', k) de *CS* definiendo $G' = G$ y $k = n - 1$, donde n es la cantidad de vértices de G. Esta transformación requiere tiempo lineal en el tamaño de la instancia de *CH*, ya que copiar la representación de G requiere tiempo lineal en el tamaño de dicha representación, y asignar $k = n - 1$ requiere tiempo $O(1)$.

Veamos que G es una instancia SÍ de *CH* si y solo si (G', k) es una instancia SÍ de *CS*.

Si G es una instancia SÍ de *CH*, entonces existe un camino hamiltoneano en G, y por lo tanto también en G'. Este camino, por definición de camino hamiltoneano, no repite vértices, por lo cual es simple, y además visita n vértices, por lo cual tiene largo $n - 1 = k$. En consecuencia, (G', k) es una instancia SÍ de *CS*.

Supongamos ahora que (G', k) es una instancia SÍ de *CS*, es decir, existe en G' un camino simple de largo k. Por definición de la transformación, este es un camino simple en G de largo $n - 1$. Como no repite vértices y recorre $n - 1$ aristas, necesariamente visita n vértices distintos, lo cual implica que este camino es hamiltoneano y en consecuencia G es una instancia SÍ de *CH*.

(b) Para probar que *CS* es $\mathcal{NP}$-completo debemos probar

(I) *CS* pertenece a $\mathcal{NP}$,

(II) cualquier problema que pertenece a $\mathcal{NP}$ se puede reducir en tiempo polinomial a *CS*.

Como por letra *CH* es $\mathcal{NP}$-completo, y por la parte (a) $CH \leq_P CS$, entonces por la propiedad transitiva de la relación $\leq_P$ el segundo punto se cumple y solo resta probar que *CS* pertenece a $\mathcal{NP}$.

Para esto definimos un certificado para *CS* como una lista de vértices, ℓ. Definimos también un algoritmo certificador, $B(G, k, \ell)$, que toma como parámetros la instancia (G, k) y el certificado ℓ.

Notemos que el tamaño de la representación de una entrada es $\Omega(n + m)$, donde n es la cantidad de vértices y m es la cantidad de aristas. Por lo tanto se cumple que el largo del certificado es polinomial en el largo de la entrada.

Debemos probar que

- B tiene tiempo de ejecución polinomial en el tamaño de su entrada,
- para cada instancia (G, k), se cumple que (G, k) pertenece a CS si y solo si existe un certificado ℓ tal que $B(G, k, \ell)$ es TRUE.

El certificador consta de los siguientes pasos:

1. Si en ℓ se repite algún vértice, responder FALSE.
2. Si ℓ tiene largo menor o igual a k, responder FALSE.
3. Si dos vértices consecutivos en ℓ no son adyacentes en G, responder FALSE.
4. Si no se cumple ninguna de las condiciones anteriores, responder TRUE.

Este algoritmo requiere tiempo polinomial en el tamaño de sus entradas. En efecto, una implementación trivial del paso 1 en la que cada vértice de ℓ se busca exhaustivamente en ℓ para detectar si se repite requiere tiempo $O(|\ell|^2)$, donde $|\ell|$ denota el largo de ℓ. El paso 2 requiere tiempo lineal en $|\ell|$, y el paso 3 implica buscar $|\ell| - 1$ veces una arista en G, y cada una de estas búsquedas requiere tiempo $O(m)$. En resumen, el tiempo total que requiere este algoritmo es $O(|\ell|^2 + |\ell|m)$.

Resta probar que (G, k) es una instancia SÍ de CS si y solo si existe ℓ de largo polinomial en el tamaño de la representación de (G, k) tal que $B(G, k, \ell)$ responde TRUE. Si (G, k) es una instancia SÍ de CS, entonces existe un camino simple de largo al menos k en G. Sea ℓ la lista de vértices que recorre tal camino en el orden en que son recorridos. Notar que ℓ tiene largo no mayor a n, que es polinomial en el tamaño de G. Como el camino es simple y de largo al menos k, la condiciones de los pasos 1 y 2 no se cumplen. Como además ℓ contiene los vértices de un camino en G, todos los vértices consecutivos son adyacentes y por lo tanto la condición del paso 3 tampoco se cumple. En consecuencia, el algoritmo devuelve TRUE en el paso 4. Supongamos ahora que existe ℓ tal que $B(G, k, \ell)$ responde TRUE. Por la condición del paso 3, la secuencia de vértices de ℓ determinan un camino en G que, por la condición del paso 2, tiene largo al menos k (porque ℓ tiene más de k vértices). Más aún, este camino es simple, ya que por la condición del paso 1 no hay vértices repetidos en ℓ, por lo cual concluimos que (G, k) es una instancia SÍ de CS. Notar que la condición del paso 1 implica que ℓ no tiene largo mayor n.

Ejercicio 3 (14 puntos)

Se quiere ordenar según el remitente, que se identifica con una cadena de caracteres, una lista C de n correos electrónicos. Se sabe que los posibles remitentes son a lo sumo 10. La comparación entre remitentes requiere tiempo $O(1)$.

(a) Demuestre que el algoritmo de la Figura 4 para resolver el problema admite una implementación cuyo tiempo de ejecución es $O(n)$.

```
1 Algorithm Ordenar correos (C)
2     Crear L inicialmente vacía
      /* L es una lista de listas, una para cada valor de remitente. Se
         mantiene ordenada según los remitentes.                          */
3     foreach correo c en C do
4         Buscar en L una lista L_r que corresponde al remitente r de c
5         if no se encontró L_r then
6             Crear L_r e insertarla en L manteniendo el orden por remitente
7         Insertar c en L_r
8     Devolver la concatenación de las listas de L en el orden en que se encuentran
9 end
```

Figura 4: Algoritmo para ordenar correos.

(b) Considere el siguiente análisis que pretende mostrar (incorrectamente) que el tiempo de ejecución de un algoritmo basado en comparaciones que resuelve este problema de ordenamiento es $\Omega(n \log n)$:

1. Sea M_n la cantidad máxima de comparaciones (con resultado verdadero/falso) que realiza un algoritmo de ordenamiento para una entrada de largo n. Sea $x_1, x_2, \ldots, x_{M_n}$ la secuencia binaria de resultados de esas comparaciones para una cierta entrada C, completada con ceros a la derecha si para esa entrada en particular se realizan menos de M_n comparaciones.
2. Dado el algoritmo de ordenamiento, la secuencia $x_1, x_2, \ldots, x_{M_n}$ determina una permutación de los elementos de C que hace que dichos elementos queden ordenados.
3. El algoritmo de ordenamiento debe distinguir entre $n!$ posibles permutaciones para generar una salida correcta.
4. Por lo tanto se cumple $2^{M_n} \geq n!$, de donde surge que $M_n \geq \log n! = \Omega(n \log n)$.

Especifique qué es incorrecto en este análisis y explique por qué. No es necesario corregir el análisis, solo explicar lo que sea incorrecto.

Solución:

(a) Las listas se representan con una estructura que permita recorrerlas pasando de un elemento al siguiente en $O(1)$, por ejemplo mediante nodos enlazados.

La creación de L es $O(1)$.

El ciclo de la línea 3 consiste en n iteraciones y veamos que cada una es $O(1)$. Se obtiene r en $O(1)$ pasando al siguiente elemento de C. Se busca L_r comparando r con el primer elemento de cada una de las listas de L. La búsqueda termina al encontrar L_r, o llegar al final de L o a una lista cuyo primer elemento sea mayor que r. Cada comparación es $O(1)$ y la cantidad de comparaciones es no mayor a 10, o sea $O(1)$, ya que hay a lo sumo una lista por cada remitente. La posible creación de L_r es $O(1)$. La inserción de c en L_r es $O(1)$ insertando al inicio de la lista. También se puede insertar al final si la lista es circular o si tiene una cabecera con enlaces a cada extremo. Esto tiene la ventaja de que la lista resultado mantendría la estabilidad (los correos de igual remitente quedan en el mismo orden que estaban en C). Por lo tanto, cada iteración es $O(1)$ y el ciclo es $O(n)$.

La construcción de la lista resultado en la línea 8 es $O(1)$, porque con cada una de las hasta 10 listas de L se enlaza su último elemento con el primero de la siguiente.

Entonces, tenemos una cantidad fija de pasos todos acotados superiormente por $O(n)$, por lo que por KyT(2.5), el algoritmo es $O(n)$.

(b) El error está en el tercer punto. Solo hay 10^n entradas y por lo tanto esa es una cota superior a la cantidad de permutaciones que se necesita reconocer. Pero además, la misma permutación puede ordenar varias listas de entrada. Por ejemplo para $n = 4$ la permutación $P = (2, 4, 3, 1)$ ordena correctamente, entre otras, las listas $(‘d’, ‘a’, ‘c’, ‘b’)$, $(‘e’, ‘a’, ‘c’, ‘b’)$ y $(‘d’, ‘b’, ‘c’, ‘b’)$.