Parcial de Programación 3

8 de diciembre 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 (10 puntos)

(a) Dé un ejemplo de una red de flujo en la cual todas las capacidades de las aristas son impares, y la red tiene un flujo máximo de valor par. Justifique su respuesta diciendo explícitamente cuál es el flujo y su valor, verificando que satisface todas las propiedades requeridas para un flujo, y argumentando que es máximo (mencione los resultados teóricos que utilice).

(b) Sea $G = (V, E)$ una red de flujo $s-t$ tal que todas las capacidades de sus aristas son pares. Demuestre que el valor de flujo máximo de la red es par.

Solución:

(a) Definimos $G = (\{s, u, v, t\}, \{(s, u), (s, v), (u, t), (v, t)\})$, con todas las capacidades en 1. Sea f el flujo $s-t$ que transporta una unidad de flujo por todas las aristas. El valor de f es la suma de los flujos salientes de s, que es igual a 2. Satisface las restricciones de capacidad, ya que el flujo de toda arista es 1, que está entre 0 y su capacidad. También satisface las ecuaciones de balance de flujo, ya que tanto u como v reciben una unidad de flujo y emiten la misma cantidad. Finalmente, el valor de f es igual a la capacidad del corte $(\{s\}, \{u, v, t\})$. Como el valor de un flujo está acotado superiormente por la capacidad de cualquier corte, esto implica que ningún flujo tiene valor mayor que el de f.

(b) El valor de flujo máximo coincide con la capacidad de un corte de mínima capacidad, que es par por ser la suma de capacidades de aristas que son todas pares.

Ejercicio 2 (20 puntos)

Queremos cortar una secuencia de números, $x = (x_1, \ldots, x_n)$, en fragmentos consecutivos, intentando que en cada fragmento los números sean cercanos entre sí, sin realizar una cantidad excesiva de cortes.

Formalmente, definimos una partición $(x_1, \ldots, x_{f_1}), (x_{s_2}, \ldots, x_{f_2}), \ldots, (x_{s_k}, \ldots, x_n)$, con $s_{h+1} = 1 + f_h$, $1 \leq h < k$, mediante cortes en ciertas posiciones, $s_2, \ldots, s_k$, elegidas con un criterio de minimización que se explica a continuación. Cada fragmento tiene un costo que es la diferencia entre el máximo y el mínimo de sus elementos. En particular, el costo de un fragmento que tiene solo un elemento es 0. La partición tiene un costo que es la suma de los costos de sus fragmentos, más un costo adicional constante, c, por cada corte realizado. O sea, si la partición consiste en k fragmentos, su costo es $\left(\sum_{h=1}^{k} C_h\right) + c\,(k-1)$, donde C_h es el costo del fragmento h. En particular, si no se hace ningún corte el costo de la partición es el costo del único fragmento que la compone, que abarca toda la secuencia original.

Ejemplo: Sea la secuencia $x = (5, 2, 3, 4, 4)$ y $c = 1$. La partición $(5, 2, 3, 4, 4)$ tiene costo $5 - 2 = 3$. La partición $(5, 2, 3), (4, 4)$ tiene costo $5 - 2 + 4 - 4 + c = 4$. La partición $(5), (2), (3), (4, 4)$ tiene costo $3c = 3$.

Debemos encontrar posiciones de corte, $(s_2, \ldots, s_k)$, que minimicen el costo de la partición.

Definimos $d_{i,j}$ como la diferencia máxima entre los elementos en $(x_i, \ldots, x_j)$, $1 \leq i \leq j \leq n$. Asumimos que $d_{i,j}$ puede obtenerse en tiempo $O(1)$ (por ejemplo a través de una matriz precalculada).

(a) Dé una relación de recurrencia para $\mathrm{OPT}(j)$, $1 \leq j \leq n$, definida como el costo mínimo de una partición para el prefijo $(x_1, \ldots, x_j)$. Explique la procedencia de cada término de la expresión.

(b) Diseñe un algoritmo iterativo que dada una secuencia x de largo n, devuelva el costo mínimo de una partición para x. El tiempo de ejecución debe ser $O(n^2)$ (no es necesario demostrarlo).

(c) Diseñe un algoritmo iterativo que dada una secuencia x de largo n devuelva las posiciones en donde deben empezar los fragmentos para obtener una partición de costo mínimo. El resultado debe ser un arreglo de booleanos tal que el valor en la posición i es `true` si y solo si en i empieza un fragmento.

El tiempo de ejecución debe ser $O(n^2)$ (no es necesario demostrarlo).

Puede usar estructuras de datos construidas con el algoritmo de la parte anterior.

Solución:

(a) El óptimo puede obtenerse o bien con un único fragmento o bien con más de uno.

Si es con uno solo, el resultado es la máxima diferencia entre los elementos de ese fragmento, que está dado por $d_{1,j}$. Si $j = 1$, que corresponde al paso base de nuestra recurrencia, esta es de hecho la única posibilidad.

Si es con más de un fragmento, el último elemento pertenece a un fragmento que empieza en cierta posición i menor o igual a j y mayor que 1. El resultado en este caso es la suma de la solución óptima para la subsecuencia $1 \cdots i-1$, $OPT(i-1)$, más el costo de partir para obtener el último fragmento, c, más el costo del último fragmento, $d_{i,j}$.

Tomando como convención que el mínimo de un conjunto vacío es infinito, la relación de recurrencia se puede expresar como

$$OPT(j) = \min\{d_{1,j}, c + \min_{2 \leq i \leq j}\{OPT(i-1) + d_{i,j}\}\}, \qquad 1 \leq j \leq n,$$

donde notamos que en el paso base, para $j = 1$, el mínimo es $d_{1,1}$.

Alternativamente, la misma relación de recurrencia se puede expresar separando el paso base en una ecuación específica, sin necesidad de establecer una convención para el mínimo de un conjunto vacío.

$$\begin{aligned} OPT(1) &= d_{1,1}\,, \\ OPT(j) &= \min\{d_{1,j}, c + \min_{2\le i\le j}\{OPT(i-1) + d_{i,j}\}\}\,, \qquad 2 \le j \le n\,. \end{aligned}$$

Con esta definición alternativa, el algoritmo de la figura 2, que resuelve la parte b, debe ajustarse inicializando OPT[1] como primer paso, y haciendo que la variable j tome valores desde 2 hasta n en el ciclo **for**.

(b)

Algorithm *Óptimos*
  for $j = 1$ **to** n **do**
    Hacer $OPT[j] = \min\{d_{1,j}, c + \min_{2\le i\le j}\{OPT(i-1) + d_{i,j}\}$
  end
  return *OPT[n]*
end

Figura 1: Algoritmo para calcular los óptimos para cada prefijo

(c) El algoritmo de la figura 2 obtiene las posiciones de corte en un arreglo p, utilizando como insumo el arreglo OPT calculado por el algoritmo de la figura 1.

Algorithm *Posiciones*
  Hacer $p[i] =$ **false** para todo i, $1 \le i \le n$
  Hacer $j = n$
  while $j > 0$ **do**
    if $OPT[j] = d_{1,j}$ **then**
      Hacer $i = 1$
    else
      Buscar i, $2 \le i \le j$, tal que $OPT[j] = c + OPT[i-1] + d_{i,j}$
    end
    Hacer $p[i] =$ **true**
    Hacer $j = i - 1$
  end
  return p
end

Figura 2: Algoritmo para calcular las posiciones de comienzo de cada fragmento

Ejercicio 3 (20 puntos)

Considere el problema de decisión *casiSAT* definido de la siguiente manera. La entrada está formada por un conjunto de n variables booleanas, $X = \{x_1, x_2, , \ldots , x_n\}$, y un conjunto de k cláusulas, $C_1, C_2, , \ldots , C_k$. Cada cláusula es una disyunción de términos distintos, $t_1 \vee t_2 \vee \ldots \vee t_\ell$, donde cada término es una variable o la negación de una variable, $t_i \in \{x_1, x_2, , \ldots , x_n, \bar{x}_1, , \ldots , \bar{x}_n\}$. El problema *casiSAT* consiste en determinar si existe una asignación de verdad para X tal que al menos $k-1$ cláusulas son satisfechas.

(a) Demuestre que *casiSAT* pertenece a $\mathcal{NP}$. Repita cualquier argumento que utilice de los estudiados en el curso.

(b) Demuestre que *casiSAT* es $\mathcal{NP}$-Completo.

Solución:

Boceto de solución

(a) Definimos un certificado para *casiSAT* como una tira de valores booleanos, $v_1, v_2, \ldots$, donde cada v_i determina el valor en una asignación de verdad para una variable de X (solo es necesario asignar valores a variables que participan en alguna cláusula).

Un algoritmo certificador, B, recibe una representación de una instancia de *casiSAT*, s, y un certificado, t, y simplemente evalúa, para cada cláusula, si hay al menos un término que valga verdadero con la asignación definida por el certificado, de manera que dicha cláusula se satisface. Si al menos $k-1$ cláusulas son satisfechas, el certificador responde SÍ, y en caso contrario responde NO. Este proceso requiere recorrer a lo sumo una vez cada término de cada cláusula, y por lo tanto es polinomial en el tamaño de las entradas de B, $|s| + |t|$.

Para terminar, notamos que este algoritmo es un certificador correcto para *casiSAT*, lo cual demuestra que este problema pertenece a $\mathcal{NP}$. En efecto, si s representa una instancia SÍ de *casiSAT*, existe una asignación de verdad para las variables que participan en las cláusulas de la instancia tales que al menos $k-1$ cláusulas son satisfechas. El certificado $t = v_1, v_2, \ldots$ que asigna dichos valores a las variables hace que $B(s,t)$ responda SÍ, y se cumple que $|t|$ es polinomial en el tamaño de la instancia, ya que no supera la cantidad de términos en las cláusulas. Recíprocamente, si para una instancia, con cierta representación s, existe un certificado $t = v_1, v_2, \ldots$ que hace que $B(s,t)$ responda SÍ, entonces asignando los valores $v_1, v_2, \ldots$ a las variables de X se cumple que al menos $k-1$ cláusulas de la instancia son satisfechas, lo cual implica que se trata de una instancia SÍ.

(b) Mostraremos que $SAT \leq_P casiSAT$. Como *SAT* es $\mathcal{NP}$-Completo y *casiSAT* pertenece a $\mathcal{NP}$ por la parte anterior, esto implica que *casiSAT* es $\mathcal{NP}$-Completo. Sea S una instancia de *SAT* con un conjunto de m variables $X = \{x_1, x_2, , \ldots , x_m\}$ y un conjunto de r cláusulas $C = \{C_1, C_2, , \ldots , C_r\}$. Generamos una instancia S' de *casiSAT* con un conjunto de variables X' y un conjunto de cláusulas C' de la siguiente manera. Definimos X' de tamaño $n = m+1$ agregando a X una variable z, y definimos C' con $k = r+2$ cláusulas, obtenidas agregando a C dos cláusulas con un término cada una: z y $\bar{z}$. Esta transformación claramente requiere tiempo polinomial en el tamaño de S. Además observamos que si S es una instancia SÍ de *SAT*, entonces la misma asignación de verdad para X que satisface las cláusulas de C, junto con la asignación $z = verdadero$, determina una asignación de verdad para X' que satisface $k-1$ cláusulas de C': las de C y la cláusula z. Por otra parte, si S' es una instancia SÍ de *casiSAT*, existe una aginación de verdad para X' que satisface $k-1$ cláusulas de C'. Cualquiera sea esta asignación, alguna de las cláusulas z o $\bar{z}$ no se satisface, lo cual implica que necesariamente todas las cláusulas de C se satisfacen y por lo tanto S es una instancia SÍ de *SAT*.