Examen de Programación 3
14 de febrero 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 T un árbol binario con raíz, es decir, T es o bien una hoja o está compuesto por un nodo raíz y subárboles izquierdo y derecho, T_I y T_D respectivamente, que son en sí mismos árboles binarios con raíz. Además, cada nodo v (ya sea hoja o interno) tiene asociado un peso positivo, $w(v)$. El peso de un conjunto de nodos es la suma de los pesos de los nodos del conjunto.

Un conjunto de nodos de T es *independiente* si ninguno de sus nodos es padre de otro. Definimos las siguiente funciones para árboles binarios con raíz:

1. $q(T)$ es el peso máximo que puede tener un conjunto de nodos independientes de T que no incluya la raíz.
2. $p(T)$ es el peso máximo que puede tener un conjunto de nodos independientes de T (puede incluir la raíz).

(a) Diseñe mediante la técnica *Divide y Vencerás* un algoritmo que, dado un árbol T, calcule $q(T)$ y $p(T)$.

(b) Demuestre que el algoritmo admite una implementación con tiempo de ejecución $O(n)$, donde n es la cantidad de nodos de T. Puede asumir que T es perfecto (todas las hojas están a la misma profundidad y todos los niveles están completos).

Solución:

(a) Sea T un árbol con raíz v. Si T es una hoja, de manera trivial tenemos $p(T) = w(v)$ y $q(T) = 0$.

En el caso general se resuelve el problema en cada subárbol, T_I y T_D.

La unión de un conjunto independiente de nodos de T_I y un conjunto independiente de nodos de T_D es un conjunto independiente de nodos de T porque ningún nodo de T_I es padre de uno de T_D y viceversa. Por lo tanto

1. Si no se incluye la raíz se pueden unir los conjuntos independientes de mayor peso de cada subárbol, de donde $q(T) = p(T_I) + p(T_D)$.
2. Para el cálculo de $p(T)$ la opción es incluir o no la raíz. Si no se la incluye el resultado es $q(T)$. Si se la incluye no se puede incluir sus hijos, o sea, las raíces de T_I, y T_D. Los pesos de los conjuntos independientes de mayor peso de T_I y T_D que no incluyen sus raíces son $q(T_I)$ y $q(T_D)$ respectivamente. Entonces $p(T) = \text{máx}\left\{w(v) + q(T_I) + q(T_D), q(T)\right\}$.

```
1 Algorithm MaxInd (T)
2     if T es hoja then
3         return (w(v), 0)
4     else
5         (p_I, q_I) ← MaxInd(T_I)
6         (p_D, q_D) ← MaxInd(T_D)
7         q_T ← p_I + p_D
8         return (máx{w(v) + q_I + q_D, q_T}, q_T)
9 end
```

Figura 1: Algoritmo para calcular p y q.

(b) Sea $f(n)$ el tiempo de ejecución máximo entre todos los árboles perfectos con n nodos. Como en un árbol perfecto, T, se cumple que T_I y T_D son árboles perfectos que tienen el mismo tamaño, $\frac{n-1}{2}$, cada una de las llamadas recursivas de los pasos 5 y 6 requiere como máximo tiempo $f\left(\frac{n-1}{2}\right)$. Por lo tanto, f satisface la siguiente relación de recurrencia

$$f(1) \leq c \tag{1}$$

$$f(n) \leq 2f\left(\frac{n-1}{2}\right) + c\,, \quad n > 1\,, \tag{2}$$

donde c es una constante positiva que acota superiormente tanto el tiempo de ejecución del paso base como el tiempo de ejecución de las operaciones adicionales a las llamadas recursivas en el paso inductivo.

Demostramos por inducción en n que se cumple $f(n) \leq cn$ para todo n, $n \geq 1$, lo cual prueba que $f(n)$ es $O(n)$. Para $n = 1$ la tesis es consecuencia inmediata de (1). Para $n > 1$, aplicando la hipótesis de inducción a (2) obtenemos

$$f(n) \leq 2c\frac{n-1}{2} + c = cn\,. \tag{3}$$

La tesis de inducción, $f(n) \leq cn$, puede deducirse buscando demostrar por inducción en n que se cumple $f(n) \leq kn$ para cierta constante k positiva a determinar.

Para $n = 1$, se requiere $f(1) \leq k$ y sabemos por (1) que esto se cumple para $k \geq c$.

Asumiendo la hipótesis inductiva $f(n-1) \leq k\,(n-1)$, de (2) obtenemos

$$\begin{aligned} f(n) &\leq 2k(n-1)/2 + c \\ &= kn - k + c. \end{aligned}$$

Se requiere $f(n) \leq kn$, lo cual se cumple si $kn - k + c \leq kn$, para lo cual es suficiente que se cumpla $k \geq c$.

Ejercicio 2 (30 puntos)

Considere un conjunto de variables booleanas $\{x_1, \ldots, x_n\}$.

Una conjunción sobre estas variables es una fórmula de la forma $t_1 \wedge \ldots \wedge t_k$ donde $t_i \in \{x_1, \ldots, x_n, \overline{x_1}, \ldots, \overline{x_n}\}$. Notar que una conjunción es análoga a una cláusula como las consideradas en el problema de satisfacibilidad booleana (SAT), usando operadores de conjunción en lugar de disyunción.

Considere un conjunto de conjunciones $C_1, \ldots, C_m$.

El problema $FALSABLE$ consiste en decidir si existe una asignación de valores de verdad para las variables de forma tal que la fórmula

$$F = \bigvee_{i=1}^{m} C_i$$

es falsa.

Demostrar que $FALSABLE$ es $NP-$Completo.

Solución:

Para probar que $FALSABLE \in NP$ construimos un certificador análogo al usado en SAT. Dada una asignación y un conjunto de conjunciones, hay que chequear que en cada conjunción haya un literal negativo. Esto se realiza trivialmente en tiempo lineal, por tanto polinómico.

Para probar la completitud se puede probar que $SAT \leq_P FALSABLE$. Para eso, dada una instancia de SAT, la transformamos considerando de la siguiente manera: Intercambiamos en la fórmula todos los conectivos (incluido el signo de los literales), obteniendo una nueva fórmula que puede usarse como entrada de $FALSABLE$. La fórmula resultante es falseable sii hay un literal negativo en cada conjunción. Usando la misma valuación eso se traduce a un literal positivo en cada cláusula en la fórmula original. Por lo tanto, con la respuesta de una única llamada a $FALSABLE$ podemos resolver SAT. La transformación es trivialmente polinomial.

Como $FALSABLE \in NP$ y $SAT \leq_P FALSABLE$; y sabemos que $SAT \in NP$, entonces $FALSABLE$ es NP-completo, como queríamos.

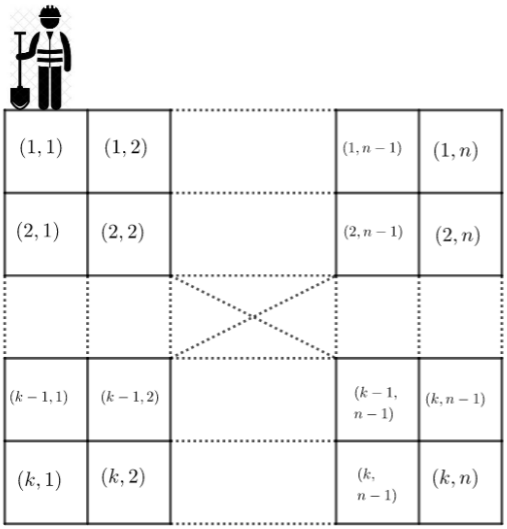

Figura 2: Coordenadas del mapa del subsuelo ($n \times k$ metros).

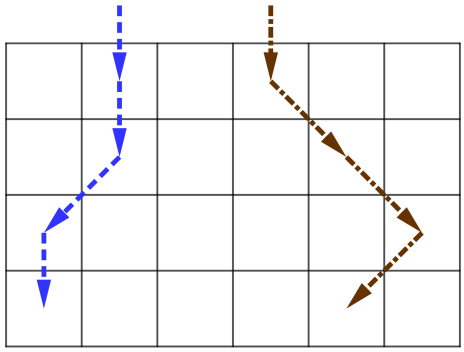

Figura 3: Ejemplos de planes de la perforación para $n = 6$ y $k = 4$.
$[V, V, I, V]$, con posición de inicio 2, con ganancia $v_{1,2} + v_{2,2} + v_{3,1} + v_{4,1}$.
$[V, D, D, I]$, con posición de inicio 4, y ganancia $v_{1,4} + v_{2,5} + v_{3,6} + v_{4,5}$.

Ejercicio 3 (35 puntos)

Un minero quiere programar una maquinaria para realizar una única perforación en una veta de mineral, y optimizar el valor del material extraído.

A nivel de la superficie, la zona útil es una franja delgada de n metros de longitud, cuyo ancho es despreciable a los efectos de este problema. La excavación se va a realizar hasta k metros de profundidad.

El minero tiene un mapa aproximado del subsuelo, que divide el área de excavación en sectores cuadrados de un metro de lado, asignándole a cada sector una coordenada (i, j), con $1 \leq i \leq k$, $1 \leq j \leq n$. Para cada sector (i, j) se conoce el valor aproximado $v_{i,j}$ que se obtiene al extraer el mineral de ese sector. Por comodidad definimos además $v_{i,0} = v_{i,n+1} = -\infty$ para todo i, $1 \leq i \leq k$. La figura 1 ilustra el esquema de coordenadas.

La máquina perfora de arriba hacia abajo, en tres posibles direcciones: diagonal izquierda, vertical y diagonal derecha. Específicamente, cuando se encuentra en un sector (i, j), puede continuar por $(i+1, j-1)$, $(i+1, j)$, o $(i+1, j+1)$ (siempre y cuando se mantenga dentro del área de excavación: $1 \leq i \leq k$, $1 \leq j \leq n$). Un *programa* es un arreglo de tamaño k de elementos del conjunto $\{I, V, D\}$ (diagonal izquierda, vertical, diagonal derecha), que indican la trayectoria que debe tomar la máquina en cada paso. El primer elemento del arreglo siempre es V (siempre se inicia perforando verticalmente).

El minero debe definir un *plan de perforación*, que consiste en un programa y una *posición de inicio* de la perforación (lo que corresponde a una coordenada entre 1 y n). La figura 2 ilustra dos ejemplos.

La ganancia de un plan de perforación es la suma de los valores de $v_{i,j}$ para los k sectores que recorre el programa partiendo desde la posición de inicio.

(a) Dada una instancia del problema, sea $OPT(i, j)$ la ganancia máxima que se puede obtener perforando a partir del sector (i, j) (incluyendo la ganancia de ese sector) llegando hasta profundidad k. Especifique una relación de recurrencia para OPT; justifique la respuesta explicando el origen de cada término.

(b) Dé un algoritmo iterativo eficiente para calcular $OPT(i, j)$ para todo i, j, $1 \leq i \leq k$, $1 \leq j \leq n$.

(c) Asumiendo conocida la matriz $OPT(i, j)$, dé un algoritmo eficiente para obtener un plan de perforación que maximiza la ganancia.

Solución:

(a) $OPT(k, j) = v_{k,j}, 1 \leq j \leq n$

$OPT(i, 0) = v_{i,0} = -\infty, 1 \leq i \leq k$

$OPT(i, n+1) = v_{i,n-1} = -\infty, 1 \leq i \leq k$

$OPT(i,j) = v_{i,j} + \text{máx}\{OPT(i+1, j-1), OPT(i+1, j), OPT(i+1, j+1)\}, 1 \leq i < k, 1 \leq j \leq n$

Notar que en la segunda cláusula, para calcular valores de la forma $OPT(i,1)$ y $OPT(i,n)$ usamos que $v_{i,0} = v_{i,n+1} = -\infty$.

También podemos no hacer uso del recurso:

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

$OPT(i,j) = v_{i,j} + \text{máx}\{OPT(i+1, j-1), OPT(i+1, j), OPT(i+1, j+1)\}, 1 \leq i < k, 1 < j < n$

$OPT(i,1) = v_{i,1} + \text{máx}\{OPT(i+1, 1), OPT(i+1, 2), \}, 1 \leq i < k$

$OPT(i,n) = v_{i,n} + \text{máx}\{OPT(i+1, n-1), OPT(i+1, n)\}, 1 \leq i < k$

(b)

```
1  procedure calcOPT(v[][],k,n){
2    create  OPT[1..k][0..n+1];
3    for j := 1 to n do
4      OPT(k,j) := v[k][j];
5    endfor;
6    for i := 1 to k do
7      OPT(i,0) := v[i][0];
8      OPT(i,n+1) := v[i][n+1];
9    endfor;
10   for i := k-1 downto 1 do
11     for j := 1 to n do
12       OPT(i,j) := v[i][j] + max {OPT(i+1,j-1), OPT(i+1,j), OPT(i+1,j↩
             +1)};
13     endfor;
14   endfor;
15   return OPT;
16 };
```

(c)

```
1    procedure calcOPT(OPT,v,k,n){
2    create PRG[1..k];
3    index := argmax_{1≤i≤n}(OPT(1,i))
4    PRG[1] := V;
5    currcol := index;
6    for i := 1 to k-1 do
7      if OPT(j,currcol) = v[j][currcol] + OPT(j+1,currcol) then
8        PRG[i+1] := V;
9      elif OPT(j,currcol) = v[j][currcol] + OPT(j+1,currcol-1) then
10       currcol  -= 1;
11       PRG[i+1] := I;
12     elif OPT(j,currcol) = v[j][currcol] + OPT(j+1,currcol+1) then
13       currcol  += 1;
14       PRG[i+1] := D;
15     endif;
16   endfor;
17   return (index,PRG);
18   }
```