Examen de Programación 3
30 de julio 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 (30 puntos)

La Agencia Aeroespacial Uruguaya se encuentra abocada a la misión de enviar una flota de robots exploradores a Saturno. Uno de los problemas relevantes durante la misión es el siguiente: dada una zona a explorar, z, asignar esta tarea de exploración a un robot que esté en condiciones de alcanzar la zona z recorriendo la menor distancia posible. Para el desarrollo de la misión se cuenta con la siguiente información:

- Información geográfica aproximada: una grilla regular, G, de $M \times M$ celdas, que representa el mapa de navegación donde hay celdas *transitables* y celdas *intransitables* (por la presencia de obstáculos).
- Información de localización: la localización de la zona de interés, z, y la localización de cada uno de los robots $r \in R$. Tanto la localización de z como la de los robots se representa mediante una única celda en la grilla, que en todos los casos es transitable. No hay más de un robot en una misma celda, y ningún robot está en la celda de z.

	1	2	3	4
1				r_1
2	z			
3				
4	r_2			

Figura 1: Ejemplo con $M = 4$ y $R = \{r_1, r_2\}$. Las celdas sombreadas son intransitables. En este caso la exploración se debe asignar a r_1, que se encuentra a distancia 4 de z, mientras que r_2 está a distancia 6.

Los robots se pueden desplazar a través de una secuencia de movimientos, cada uno de los cuales se realiza en solo una de cuatro direcciones posibles, {Este, Oeste, Norte, Sur}, y siempre por celdas marcadas como transitables en la grilla. La figura 1 ilustra un escenario de ejemplo.

1. Escriba un algoritmo que determina un robot que puede alcanzar la zona z recorriendo la menor distancia posible; si no hay ninguno, se informa esta situación, si hay más de uno a distancia mínima, se elige cualquiera de ellos arbitrariamente. No es necesario determinar el camino a recorrer por el robot. Su algoritmo debe admitir una implementación cuyo tiempo de ejecución es $O(M^2)$. Reescriba cualquier algoritmo que utilice de los estudiados en el curso.
2. Demuestre que su algoritmo admite una implementación cuyo tiempo de ejecución es $O(M^2)$. Repita cualquier argumento que utilice de los estudiados en el curso.

Sugerencia: Modele el problema en términos de grafos.

Solución:

(a) Modelamos la información geográfica del problema mediante un grafo, con un vértice por cada celda transitable, y una arista por cada par de celdas transitables adyacentes entre sí (de acuerdo a las direcciones de movimiento válidas). El grafo está implícitamente representado por la grilla G. En esta representación, cada vértice está dado por un par (x, y), $1 \leq x \leq M$, $1 \leq y \leq M$, que representa una localización transitable en la grilla. El conjunto de vértices adyacentes a un par (x, y) tiene a lo sumo 4 vértices, y está compuesto por el subconjunto transitable de las localizaciones en el conjunto

$$S = \{(x', y') \in \{(x+1, y), (x-1, y), (x, y+1), (x, y-1)\} : 1 \leq x' \leq M, 1 \leq y' \leq M\}.$$

Podría construirse explícitamente el grafo usando una representación de lista de adyacencia.

El algoritmo de la figura 2 presenta una solución basada en el algoritmo BFS. el cual se ejecuta a partir del vértice asociado a la zona z. El algoritmo hace uso de la representación implícita del grafo definida anteriormente. En particular, el arreglo que marca los vértices visitados, visitado, es bidimensional y se indexa con las coordenadas x, y de localizaciones en la grilla. De forma similar, los vértices que se agregan o retiran de cola son pares (x, y) que representan localizaciones. Además de estas estructuras de datos usuales para una implementación eficiente de BFS, hacemos uso de una matriz, robots, indexada por coordenadas de localización, que contiene ceros en los lugares donde no hay robots, y en caso contrario robots$[x, y]$ contiene el índice i que identifica al robot r_i ubicado en la posición (x, y), con $1 \leq i \leq |R|$.

```
1  Algorithm Asignar exploración
2      Hacer visitado[x, y] = false para todo x, y, 1 ≤ x ≤ M, 1 ≤ y ≤ M
3      Hacer robots[x, y] = 0 para todo x, y, 1 ≤ x ≤ M, 1 ≤ y ≤ M
4      foreach r_i ∈ R do
5          Hacer robots[x_i, y_i] = i para la localización (x_i, y_i) de r_i
6      end
7      Sea (x_z, y_z) la localización de la zona z
8      Hacer visitado [x_z, y_z] = true
9      Inicializar cola conteniendo solo el par (x_z, y_z)
10     while cola no está vacía do
11         Retirar un par (x, y) de cola
12         if robots[x, y] > 0 then return robots[x, y]
13         Sea S = {(x', y') ∈ {(x + 1, y), (x − 1, y), (x, y + 1), (x, y − 1)} : 1 ≤ x' ≤ M, 1 ≤ y' ≤ M}
14         foreach (x', y') ∈ S tal que (x', y') es transitable do
15             if not visitado[x', y'] then
16                 visitado[x', y'] = true
17                 Agregar (x', y') a cola
18             end
19         end
20     end
21     Reportar que la zona z es inalcanzable.
22 end
```

Figura 2: Algoritmo basado en BFS para determinar robot más cercano.

(b) Claramente los pasos 2 y 3 requieren tiempo $O(M^2)$. Como no hay más de un robot en una misma celda, tenemos $|R| \leq M^2$, por lo cual el tiempo requerido por el paso 4 también es $O(M^2)$. Adicionalmente los pasos 7–9 requieren tiempo $O(1)$, por lo cual concluimos que todo el proceso de inicialización previo al paso 10 requiere tiempo $O(M^2)$.

Observamos que la cantidad de iteraciones en el ciclo del paso 10 no es superior a M^2, ya que en cada iteración el paso11 retira un elemento de cola y ninguno de los M^2 pares posibles es agregado más de una vez. En efecto, en el paso 17 solo se agregan a cola pares que no están marcados como visitados al evaluar la condición del paso 15, y la ejecución de los pasos 16 y 8 garantiza que ningún par es agregado más de una vez.

Como todas las operaciones que se realizan dentro del ciclo del paso 10 requieren tiempo $O(1)$, el tiempo total de ejecución es $O(M^2)$.

Ejercicio 2 (30 puntos)

Consideramos una red de flujo *s*-*t* definida sobre un gafo dirigido $G = (V, E)$ con n nodos y m aristas, capacidades enteras, y tal que todo nodo está conectado a alguna arista. Sean f un flujo *s*-*t* y (A, B) un corte *s*-*t* cualquiera.

(a) Para $e \in E$, ¿qué cota superior e inferior debe satisfacer $f(e)$ según la definición de flujo? (*condición de capacidad*).

(b) Muestre que el valor de f, $v(f)$, está acotado superiormente por la capacidad del corte (A, B), $c(A, B)$. Justifique cada paso detalladamente.

Sugerencia: Tenga en cuenta que se cumple $v(f) = f^{out}(A) - f^{in}(A)$ y utilice la parte anterior.

(c) Supongamos que el valor máximo de flujo está acotado superiormente por una constante C. Muestre que el algoritmo de Ford-Fulkerson, presentado en la figura 3, admite una implementación cuyo tiempo de ejecución es $O(mC)$. Puede usar, sin demostración, resultados estudiados en el curso sobre la función `aumentar` (incluyendo complejidad), y sobre otros algoritmos estudiados.

```
1 Algorithm Flujo Máximo
2     Hacer f(e) = 0 para toda arista e de G
3     while existe camino simple, P, de s a t en el grafo residual G_f do
4         f' = aumentar(f, P)
5         Actualizar f a f' y G_f a G_f'
6     end
7 end
```

Figura 3: Algoritmo de Ford-Fulkerson .

Solución:

Ver material bibliográfico del curso. Para la parte (c) en particular notar que la cantidad de iteraciones no es superior a C, porque $v(f)$ se incrementa al menos en una unidad con cada iteración.

Ejercicio 3 (40 puntos)

Un herrero dispone de n tipos de varillas de longitudes $v_1 < v_2 < \ldots < v_n$ y quiere construir una de longitud L haciendo la menor cantidad de soldaduras posibles. Todos los valores son enteros y tenemos $v_1 = 1$.

(a) Para esta parte considere la instancia particular en que $n = 3$ y las longitudes de los tipos de varillas son 1, 4 y 6. Para construir varillas de longitudes 4, 8, 14 y 20, indique cuál es la cantidad mínima de varillas necesarias en cada caso; justifique su respuesta detallando cuántas varillas de cada tipo se usan en cada solución.

(b) Dé una relación de recurrencia para calcular la **cantidad mínima de varillas** necesarias, que permita resolver el problema general mediante la técnica de programación dinámica.

Sugerencia: Defina $C(i, \ell)$, cantidad mínima de varillas para construir una de longitud ℓ usando solo las de longitudes $v_1, v_2, \ldots, v_i$. ¿Cuánto valen $C(i, 0)$ y $C(1, \ell)$? Notar que $\ell \geq 0$.

(c) Dé un algoritmo **iterativo** para calcular la relación de recurrencia de la parte (b).

Solución:

(a) $L = 4$: 1 varilla de longitud 4

$L = 8$: 2 varillas de longitud 4

$L = 14$: 3 varillas, 1 de longitud 6 y 2 de longitud 4

$L = 20$: 4 varillas, 2 de longitud 6 y 2 de longitud 4

(b)

Denotemos con $S = (s_1, \ldots, s_i)$ una solución del problema para la instancia $I_{i,\ell} = ((v_1, \ldots, v_i), \ell)$, y con $|S| = \sum_{j=1}^{i} s_j$ la cantidad de varillas de la solución. Esta solución debe ser factible, o sea $\sum_{j=1}^{i} s_j v_j = \ell$, y óptima, esto es, $|S|$ es mínimo en el espacio de soluciones factibles.

Definimos $C(i, \ell)$ como en la sugerencia. Para $\ell = 0$, es claro que $C(i, \ell) = 0$ para todo i, $1 \leq i \leq n$, porque no es necesario usar ninguna varilla para construir una de largo 0. Por otra parte, para $i = 1$ solo es posible usar varillas de largo 1, por lo cual tenemos $C(i, \ell) = \ell$ para todo ℓ, $1 \leq \ell \leq L$. Estas dos observaciones determinan los casos base definidos en (1) y (2).

Para $i > 1$ y $\ell > 0$, vamos a analizar cómo descomponer una solución S para una instancia $I_{i,\ell}$ del problema, en términos de soluciones para subproblemas de $I_{i,\ell}$. Distinguimos dos casos dependiendo de la inclusión o no de varillas de tipo i (el más largo) en S. Entonces diferenciamos entre $s_i = 0$ (no se incluye ninguna) y $s_i = 1 + t_i,\ t_i \geq 0$ (se incluye al menos una).

$s_i = 0$

Notar que $\ell > 0$ implica que debe cumplirse $i > 1$.

En este caso veamos que $(s_1, \ldots, s_{i-1})$ es solución para la instancia $I_{i-1,\ell} = ((v_1, \ldots, v_{i-1}), \ell)$. Es factible ya que, como $s_i = 0$, se cumple $\sum_{j=1}^{i-1} s_j v_j = \sum_{j=1}^{i} s_j v_j = \ell$. Si no fuera óptima significaría que hay otra solución $(s'_1, \ldots, s'_{i-1})$ también factible y con menos elementos: $\sum_{j=1}^{i-1} s'_j < \sum_{j=1}^{i-1} s_j$. Pero si fuera así $S' = (s'_1, \ldots, s'_{i-1}, 0)$ sería factible para la instancia $I_{i,\ell}$ y se cumpliría $|S'| = \sum_{j=1}^{i-1} s'_j < \sum_{j=1}^{i-1} s_j = |S|$ contradiciendo la hipótesis de que S es óptima.

$s_i = 1 + t_i$

Notemos que, como se incluye una varilla de tipo i, se cumple $\ell \geq v_i$.

Veamos que $(s_1, \ldots, s_{i-1}, t_i)$ es solución para la instancia $I_{i,\ell-v_i} = ((v_1, \ldots, v_i), \ell - v_i)$. Es factible ya que, como $s_i = 1 + t_i$, se cumple $\sum_{j=1}^{i-1} s_j v_j + t_i \cdot v_i = \sum_{j=1}^{i} s_j v_j - v_i = \ell - v_i$. Si no fuera óptima, existiría otra solución, $(s'_1, \ldots, s'_i)$, también factible y con menos elementos: $\sum_{j=1}^{i} s'_j < \sum_{j=1}^{i-1} s_j + t_i$. Pero si fuera así $S' = (s'_1, \ldots, s'_{i-1}, s'_i + 1)$ sería factible para la instancia $I_{i,\ell}$ y se cumpliría $|S'| = \sum_{j=1}^{i} s'_j + 1 < \sum_{j=1}^{i-1} s_j + t_i + 1 = |S|$ contradiciendo la hipótesis de que S es óptima.

Para $\ell < v_i$, necesariamente debemos tener $s_i = 0$ para una solución factible. Por lo tanto, por el primer punto de la discusión de casos precedente, la solución óptima en este caso es necesariamente de la forma $(s_1, \ldots, s_{i-1}, 0)$, donde $(s_1, \ldots, s_{i-1})$ es solución para la instancia $I_{i-1,\ell}$. Esto da lugar a la ecuación (3).

Para $i > 1$ y $\ell \geq v_i$, tanto $I_{i-1,\ell}$ como $I_{i,\ell-v_i}$ son instancias válidas del problema. Una solución $(s_1, \ldots, s_{i-1})$ para la primera determina una solución factible con $s_i = 0$, $(s_1, \ldots, s_{i-1}, 0)$, para el problema original, $I_{i,\ell}$, y una solución $(s_1, \ldots, s_{i-1}, t_i)$ para la segunda, determina una solución factible con $s_i > 0$, $(s_1, \ldots, s_{i-1}, t_i + 1)$, para $I_{i,\ell}$. En virtud de la discusión anterior, una de estas dos soluciones factibles, la de menor cantidad de varillas, debe ser una solución óptima para $I_{i,\ell}$. Esto determina la ecuación (4), que completa la definición de la recurrencia (1)–(4) para $C(i, \ell)$.

$$C(i,0) = 0, \quad 1 \leq i \leq n. \tag{1}$$
$$C(1,\ell) = \ell, \quad 1 \leq \ell \leq L. \tag{2}$$
$$C(i,\ell) = C(i-1,\ell), \quad 1 < i \leq n,\ 0 < \ell < v_i \tag{3}$$
$$C(i,\ell) = \min\{C(i-1,\ell), 1 + C(i,\ell - v_i)\}, \quad 1 < i \leq n,\ v_i \leq \ell \leq L \tag{4}$$

(c) El algoritmo se presenta en la figura 4. Aunque no se pide en la letra del problena, en la figura se ilustra también cómo construir una solución, esto es, un algoritmo que construye un arreglo S,

donde $S[i]$ indica la cantidad de varillas de tipo i que se usan en una solución.

```
1  Algorithm Varillas(V, n, L))
       Input: V:Arreglo con las longitudes de cada tipo
       Input: n: cantidad de tipos
       Input: L: longitud buscada
2      C[i, 0] = 0, 1 ≤ i ≤ n
3      C[1, ℓ] = ℓ, 1 ≤ ℓ ≤ L
4      for ℓ = 1 to L do
5          for i = 2 to n do
6              if ℓ < V[i] then
7                  C[i, ℓ] = C[i − 1, ℓ]
8              else
9                  C[i, ℓ] = mín(C[i − 1, ℓ], 1 + C[i, ℓ − V[i]])
       // Solución de parte (c)
       Output: Cantidad mínima de varillas necesarias
10     return C[n, ℓ]
       // Construcción de solución
       Output: S: Arreglo con las cantidades de cada tipo
11     S[i] = 0, 1 ≤ i ≤ n
12     i = n, ℓ = L
13     while ℓ > 0 do
14         if i = 1 then
15             S[1] = S[1] + ℓ /* Podría asignarse S[1] = ℓ, porque S[1] = 0 en este punto.  */
16             ℓ = 0
17         else if ℓ < V[i] or C[i − 1, ℓ] < 1 + C[i, ℓ − V[i]] then
18             i = i − 1
19         else
20             S[i] = S[i] + 1
21             ℓ = ℓ − V[i]
22     return S
23 end
```

Figura 4: Algoritmo para determinar la cantidad mínima de varillas necesarias (hasta paso 10) y para determinar la cantidad necesaria de varillas de cada tipo en una solución óptima (desde paso 11)