Examen de Programación 3

13 de julio de 2018

En recuadros con este formato aparecerán aclaraciones que cumplen una función explicativa pero que no eran requeridos como parte de la solución.

Parte obligatoria

Esta parte es eliminatoria. Para la aprobación del examen debe obtenerse un mínimo del **50 % de esta parte (20 puntos)**. En caso de no llegar a dicho mínimo, **NO** se corregirán los problemas.

Pregunta 1 (13 puntos)

Sea un grafo $G = (V, E)$.

(a) Defina ordenamiento topológico.

(b) ¿Qué condiciones debe cumplir G para asegurar que existe un ordenamiento topológico?

(c) Describa un algoritmo que obtenga un ordenamiento topológico de G en caso de que exista.

Solución:

(a) Si G es un grafo dirigido, un ordenamiento topológico de G es un ordenamiento de todos sus vértices de tal forma que si $(u, w) \in E$ entonces u aparece antes que w en el ordenmiento. Es decir, si se considera $n = |V|$, $v_1, v_2, \ldots, v_n$ es un ordenamiento topológico si para cada $(v_i, v_j) \in E$ se cumple $i < j$.

(b) Para que exista un ordenamiento topológico, G debe ser un grafo dirigido y acíclico.

(c) Se va a construir un ordenamiento topológico para el grafo dirigido y acíclico G

Se repite lo siguiente mientras el G no sea vacío:

I Se selecciona $v \in V$ tal que no tiene aristas entrantes.

II Se coloca v al final del ordenamiento topológico.

III Se considera $V' = V - \{v\}$ y $E' = E - \{(v, u) \in E | u \in V\}$ se toma $G = (V', E')$.

Observar que como G es dirigido y acíclico, hay al menos un nodo $v \in V$ que no tiene aristas entrantes, lo que justifica el paso I. Observar que en el paso III, al sacar un vértice de G y todas sus aristas salientes, el grafo sigue siendo dirigido y acíclico.

Pregunta 2 (13 puntos)

Suponga que X, Y, Z son problemas que cumplen $X \in \mathcal{P}$, $Y \in \mathcal{NP}$ y Z es $\mathcal{NP}$-*completo*. Para cada una de las siguientes sentencias responda si es siempre verdadera o siempre falsa o no se puede afirmar ninguna de las dos anteriores. Si se da el tercer caso (no se pueda afirmar nada) ¿qué se debe cumplir para que sea verdadera? Justifique cada respuesta.

1. $Y \leq_P Z$
2. $X \leq_P Z$
3. $Z \leq_P Y$
4. $Z \leq_P X$

Solución:

1. $Y \leq_P Z$

   Siempre verdadera. Esto se concluye de la definición de la clase de problemas $\mathcal{NP}$-*completo*.

2. $X \leq_P Z$

   Siempre verdadera. Esto es consecuencia del punto anterior y de que $\mathcal{P} \subseteq \mathcal{NP}$.

3. $Z \leq_P Y$

   No se puede en general afirmar nada. Si fuera cierto se podria deducir que Y es $\mathcal{NP}$-*completo* porque pertenece a $\mathcal{NP}$ y es al menos tan difícil como cualquier problema *NP-completo*.

4. $Z \leq_P X$

   No se puede en general afirmar nada. Si fuera cierto para algún par de problemas X, Z entonces se cumpliría que $\mathcal{P} = \mathcal{NP}$ porque cualquier problema *NP-completo* (y como consecuencia todo problema $\mathcal{NP}$) podría resolverse en tiempo polinómico.

Pregunta 3 (14 puntos)

Sea $G = (V, E)$ un grafo conexo no dirigido con aristas ponderadas cuyos costos son todos distintos.

(a) Enuncie la propiedad `Cut` (`Cortadura`).

(b) ¿Puede haber dos árboles de cubrimiento de costo mínimo diferentes? Demuestre o dé un contraejemplo.

(c) ¿El resultado de una ejecución del algoritmo de Prim puede ser distinto del resultado de una ejecución del algoritmo de Kruskal? Justifique su respuesta.

Solución:

(a) Sean S un subconjunto de vértices tal que $\emptyset \subset S \subset V$ y $e = (v, w)$ la arista de menor costo tal que $v \in S$ y $w \in V - S$. Entonces todos los árboles de cubrimiento de costo mínimo de G contienen a e.

(b) No, si los costos de las aristas son todos distintos hay un único árbol de cubrimiento de costo mínimo.

Sea $T = (V, E')$ un `ACCM` de G y $e \in E'$ una arista genérica de T.
Se considera $T' = (V, E' - \{e\})$. Como T es un árbol, T' tiene dos componentes conexas. Sean S y $V - S$ los conjuntos de vértices de estas dos componentes. Estos conjuntos constituyen una cortadura de G. Como e es la única arista de la cortadura $(S, V - S)$ que pertenece a T debe ser la de menor costo; si no lo fuera T no sería de costo mínimo.
Aplicando la propiedad de la parte (a) la arista de menor costo de esa cortadura debe pertenecer a todo `ACCM`, por lo tanto e debe pertenecer a todo `ACCM`.
Al ser e es una arista genérica de T, el mismo razonamiento se puede aplicar a todas las aristas de T, concluyendo que cada arista de T pertenece a todo `ACCM` y por lo tanto Tes único.

(c) No, porque ambos algortimos resuelven el problema de encontrar el árbol de cubrimiento de costo mínimo y por la parte (b) existe un único árbol de cubrimiento de costo mínimo.

Problemas

Problema A (30 puntos)

Se considera una ciudad cuyo mapa de calles tiene forma de matriz de N x N y donde cada esquina se puede identificar como una celda (i, j) de la misma. Un recolector debe determinar el recorrido a realizar con su vehículo desde la esquina $(1, 1)$ hasta la esquina (N, N) de forma de recolectar la mayor cantidad de objetos posibles en las esquinas de la ciudad, sujeto a la restricción de que todas las calles están flechadas en sentido creciente, tanto en la dirección horizontal como en la vertical.

Para ello se cuenta con una matriz E, donde cada $E(i, j)$ $(1 \leq i, j \leq N)$ es un entero que indica la cantidad de objetos que se pueden recolectar en la esquina (i, j) (si no hay objetos $E(i, j) = 0$).

(I) Escriba la relación de recurrencia con la que se obtiene C, la cantidad máxima de objetos que se pueden recolectar.

(II) Complete el algoritmo que está a la derecha que calcula la cantidad de objetos que se pueden recolectar y el recorrido.
Parámetros:

- E: Parámetro de entrada. Arreglo bidimensioneal con la cantidad de objetos disponibles en cada esquina. Note que no se tienen en cuenta `E[0][j]` ni `E[i][0]`, para $0 \leq i, j \leq N$.
- rec: Parámetro de salida. Arreglo de tamaño $2N$. Cada `rec[i]` $(1 \leq i \leq 2N - 1)$ indica la i-esima esquina por la que se debe pasar. Note que no se tiene en cuenta `rec[0]`.

Return: `recoleccion` retorna la cantidad total de objetos recolectados.

```
/* cantidad de calles horizontales
    y verticales de la ciudad       */
#define N 100

struct Pos {
   int fila;
   int col;
};

int recoleccion(int E[N+1][N+1], Pos rec[2*N]) {
  int C[N+1][N+1];
  for (int i = 1; i <= N; i++ )
    C[i][0] = C[0][i] = 0;
  for (int i = 1; i <= N; i++) {
    for (int j = 1; j <= N; j++) {
      // completar 1
    }
  }
  int i = 2*N - 1;
  Pos esq;
  esq.fila = esq.col = N;
  while ((esq.fila > 0) && (esq.col > 0)) {
    // completar 2
  }
  // completar 3
}
```

Solución:

(I)

$$C(i,j) = \begin{cases} 0 & \text{si } i = 0 \text{ o } j = 0, \\ \max(C(i-1,j), C(i,j-1)) + E(i,j) & \text{en otro caso.} \end{cases}$$

(II)

```
/* cantidad de calles horizontales
    y verticales de la ciudad       */
#define N 100

struct Pos {
   int fila;
   int col;
};

int recoleccion(int E[N + 1][N + 1], Pos rec[2 * N]) {
  int C[N + 1][N + 1];
  for (int i = 1; i <= N; i++)
    C[i][0] = C[0][i] = 0;
```

```
    for (int i = 1; i <= N; i++) {
      for (int j = 1; j <= N; j++) {
        C[i][j] = max(C[i - 1][j], C[i][j - 1]) + E[i][j];
      }
    }
    int i = 2 * N - 1;
    Pos esq;
    esq.fila = esq.col = N;
    while ((esq.fila > 0) && (esq.col > 0)) {
      rec[i] = esq;
      i--;
      if ((C[esq.fila - 1][esq.col] >= C[esq.fila][esq.col - 1]) &&
          (esq.fila - 1 > 0))
        esq.fila--;
      else
        esq.col--;
    }

    return C[N][N];
  }
```

Problema B (30 puntos)

El físico Archie dispone de una balanza de dos platillos y una colección M de monedas del mismo aspecto pero todas con pesos diferentes. Dadas dos monedas, usando la balanza se puede determinar cuál es la más liviana y cúal es la más pesada entre ellas.

(I) Dada una moneda m escriba un seudocódigo que obtenga el conjunto de monedas más livianas que m.

(II) Utilizando la idea de la parte (I) y aplicando la técnica DyC, explique cómo ordenar las monedas de manera creciente, indicando cuál es el caso base y cuáles son los subproblemas del caso genérico.

(III) a. Explique cómo, sin ordenar las monedas, se puede determinar cuál es la k-ésima en orden creciente de pesos, con $1 \leq k \leq |M|$.

b. Explique cuándo se da el peor caso de parte a y cuál es el orden de complejidad, tomando como operación básica una comparación con la balanza.

Solución:

(I)
```
L ← φ
Para todo i en M - {m}
  Pesar m e i
  Si i es mas liviana que m
     L ← L ∪ {i}
Return L
```

Y quedan en $P = M - (L \cup \{m\})$ las monedas más pesadas que m.

(II) La idea de la parte (I) permite, elegiendo una moneda m como pivote, separar el resto de las monedas de M en dos conjuntos, el de monedas más livianas que m, L, y el de monedas más pesadas que m, P. Se aplica la técnica DyC:

- En el caso base, el conjunto tiene 0 o 1 elementos y está ordenado de manera trivial.
- En el caso genérico, recursivamente se ordenan L y P. El resultado es L ordenado, seguido de m, seguido de P ordenado.

(III) a. Se aplica la misma idea de la parte (I). Se elige una moneda m como pivote y se obtiene el conjunto L de las monedas más livianas que m y el conjunto P de monedas más pesadas que m. Dependiendo del tamaño de L hay 3 opciones:

- $|L| \geq k$. Recursivamente se debe buscar la k-ésima moneda en L.
- $|L| = k - 1$. Este es el caso base. La k-esima moneda es m.
- $|L| < k - 1$. La k-esima moneda está en P. Recursivamente se debe buscar la $k - (|L| + 1)$-ésima moneda en L.

b. El peor caso se da cuando siempre se elige como pivote la más liviana o la más pesada y no es la buscada. El cantidad de comparaciones es $\Theta(n^2)$, siendo $n = |M|$.