Examen de Programación 3

2 de febrero 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.

De las preguntas 1 a 3 **únicamente** se deberán responder 2.

Pregunta 1 (13 puntos)

Sea una red de flujo dada por un grafo dirigido $G = (V, E)$ con nodo fuente $s \in V$, nodo destino $t \in V$ y capacidad $c_e \geq 0$ en cada arista $e \in E$.

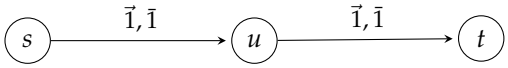
Indique si las siguientes afirmaciones son verdaderas o falsas. Justifique.

(a) Dado un flujo máximo, existe un único corte de capacidad mínima.

(b) Un corte de capacidad mínima define un único flujo máximo.

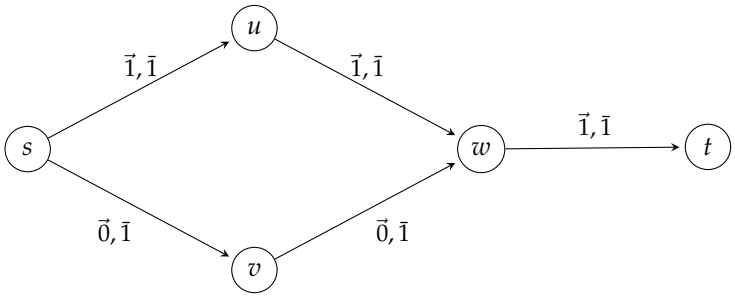
Solución:

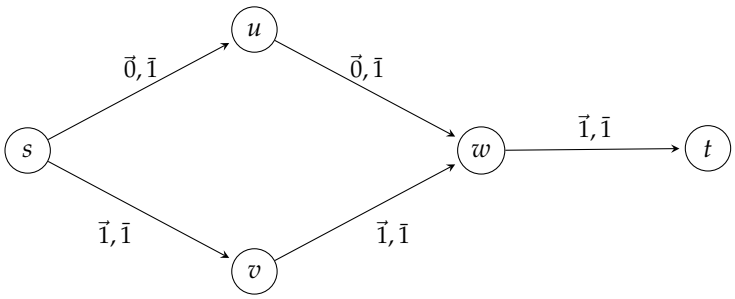
(a) Falsa. Contraejemplo. Dada la red:



Los cortes $(\{s\}, \{u, t\})$ y $(\{s, u\}, \{t\})$ son ambos de capacidad mínima.

(b) Falsa. Contraejemplo.





En ambas redes el corte de capacidad mínima es $(\{s, u, v, w\}, \{t\})$ pero los flujos son distintos.

Pregunta 2 (13 puntos)

Sea el problema que dada una cantidad de dinero y un conjunto de monedas de distintos valores, cambia a monedas el dinero utilizando la mínima cantidad de monedas.

Considere el siguiente algoritmo para resolver dicho problema:

```
mientras dinero > 0 hacer
  sea i la moneda de mayor valor que cumple valor(i) <= dinero entonces
  k := dinero div valor(i)
  dinero = dinero - k * valor(i)
```

(a) ¿Cuál de las estrategias vistas en el curso utiliza? Justifique.

(b) Si para una instancia del problema hay solución, ¿el algoritmo siempre obtiene una de ellas? En caso afirmativo, ¿siempre retorna la solución óptima? Justifique o dé un contraejemplo en cada caso.

Solución:

(a) Utiliza la técnica Greedy porque en cada paso selecciona la moneda de mayor valor que no excede el monto al que se quiere llegar.

(b) El algoritmo no obtiene siempre una solución. Se considera el siguiente contraejemplo: se tienen monedas de 3 y 5 pesos y se quieren cambiar 9 pesos. Como el algoritmo selecciona las monedas de mayor a menor, selecciona primero la de 5 y luego una de 3. De esta forma no obtiene solución, cuando la selección de las 3 monedas de 3 pesos es una solución factible.

Aún cuando se encuentra solución, la misma no siempre es óptima. Se considera el siguiente contraejemplo: se tienen monedas de 8, 5 y 1 peso y se quieren cambiar 10 pesos. Como el algoritmo selecciona las monedas de mayor a menor, selecciona primero una moneda de 8 y luego dos de 1 peso. Si bien esta solución es factible, no es óptima porque alcanza con dos monedas de 5 pesos para solucionar el problema.

Pregunta 3 (13 puntos)

(a) ¿Qué tiempos se deben considerar para plantear una relación de recurrencia en un problema divide y conquista?

(b) Escriba la relación de recurrencia para el MergeSort, indicando la procedencia de cada uno de los términos.

Solución:

(a) Hay que considerar el tiempo de dividir el problema en subproblemas más pequeños, el tiempo de resolver los subproblemas más pequeños, el tiempo de combinar sus soluciones y el tiempo de resolver el paso base.

(b) $T(n) = \begin{cases} 0, & \text{si } n = 1 \\ 2 \times T(n/2) + n, & \text{en otro caso} \end{cases}$ El término $2 \times T(n/2)$ surge del tiempo necesario para la resolución de los subproblemas más pequeños: ordenar la primera mitad del arreglo y ordenar la segunda mitad del arreglo. El término n es el tiempo necesario para dividir el problema y combinar las soluciones. En este caso dividir el problema es $O(1)$ y combinar las soluciones $O(n)$.

Pregunta 4 (14 puntos)

Considere los siguientes problemas/algoritmos vistos en el curso: Dijkstra, Programación de Intervalos (Interval Scheduling), Kruskal, Alineamiento de Secuencias (Sequence Alignment), Par de Puntos más Cercanos (Closest Pair of Points). Seleccione exactamente 3. Para cada uno explique qué es lo que resuelve y con cuál de las técnicas de diseño de algoritmos estudiadas en el curso.

Solución:

- Dijkstra: dados un grafo $G = (V, E)$ con aristas con peso no negativo, y un vértice $u \in V$, devuelve el costo del camino de menor de costo desde u a cada vértice de V. Este problema se resuelve con la técnica Greedy.
- Programación de intervalos: dada una serie de pedidos $\{1, 2, \ldots, n\}$ donde cada pedido i comienza en un tiempo $s(i)$ y termina en un tiempo $f(i)$, devuelve el conjunto más grande de pedidos compatibles (que no se solapan en el tiempo). Este problema se resuelve con la técnica Greedy.
- Kruskal: dado un grafo $G = (V, E)$ no dirigido con aristas ponderadas, devuelve un árbol de cubrimiento de costo mínimo de G. Este problema se resuelve con la técnica Greedy.
- Alineamiento de secuencias: dados dos strings, retorna el alineamiento de costo mínimo entre ambos. Este problema se resuelve con la técnica Programación Dinámica.
- Par de puntos más cercanos: dado un conjunto de puntos $P = \{p_1, \ldots, p_n\}$ en el plano, retorna un par de esos puntos, tal que la distancia euclideana entre ambos sea mínima entre todos los pares de puntos. Este problema se resuelve con la técnica Divide y Conquista.

Problemas

Problema A (30 puntos)

Se quiere organizar una reunión de n empresarios. Uno de ellos, el organizador, debe asignar en qué mesa se sentará cada participante, de forma de no asignar a una misma mesa dos empresarios en conflicto. Cada empresario se identifica con un número entre 0 y $n-1$. Se asume que si h está en conflicto con k, entonces k está en conflicto con h. Se cuenta con el arreglo $conflictos$, que en la celda i, $0 \leq i < n$, contiene la lista con los identificadores de los empresarios en conflicto con i. El organizador se propone determinar si alcanzan dos mesas para resolver el problema.

(I) Modele el problema en términos de grafos. ¿Qué significa, en términos de grafos, que alcancen dos mesas?

(II) Suponiendo que alcanzan dos mesas y al organizador le corresponde la mesa 1, ¿puede haber más de una solución? Justifique.

(III) Implemente la función *alcanzan_dos_mesas* que, si alcanzan dos mesas para ubicar a los empresarios, devuelve `true` y en el arreglo `mesa` queda el identificador de la mesa 1 o - 1 que le es asignada a cada uno; en otro caso devuelve `false` y el arreglo `mesa` es ignorado.

```
/* Precondiciones: n > 0, `conflictos' es un arreglos de n listas */
bool alcanzan_dos_mesas (int n, Lista *conflictos, int *&mesa);
```

Se pueden usar (sin necesidad de implementarlas) las operaciones de Lista o de otro TAD conocido. No se pueden usar otras funciones auxiliares.

Solución:

(I) El problema se modela con un grafo no dirigido. Cada vértice representa un empresario y entre los vértice *u* y *v* hay una arista si y sólo si hay conflicto entre los empresarios representados por esos vértices.

Alcanzan dos mesas si y sólo si el grafo es bipartito.

(II) Puede haber más de una solución si el grafo no es conexo. En cada componente se le puede asignar la mesa 1 a cualquiera de las dos partes.

(III)
```
bool componente(Lista *conflictos, int *&mesa, int pos) {
  Cola q = crear_q();
  mesa[pos] = '1';
  bool se_puede = true;
  encolar(pos, q);
  while (se_puede && (!es_vacia_q(q))) {
    int u = frente(q);
    desencolar(q);
    Lista lst = conflictos[u];
    while (!es_vacia_l(lst)) {
      int v = primero(lst);
      lst = resto(lst);
      if (mesa[v] == mesa[u]) {
        se_puede = false;
      } else if (mesa[v] == 0) {
        mesa[v] = -mesa[u];
        encolar(v, q);
      } else {
        // hay conflicto entre u y v pero están en mesas diferentes
      }
    }
  }
  return se_puede;
}

bool alcanzan_dos_mesas(int n, Lista *conflictos, int *&mesa) {
  for (int i = 0; i < n; i++)
    mesa[i] = 0;
```

```
    int i = 0;
    bool alcanzan = true;
    while (alcanzan && (i < n)) {
      if (mesa[i] == 0)
        alcanzan = componente(conflictos, mesa, i);
      i++;
    }
    return alcanzan;
  }
```

Problema B (30 puntos)

Sea A una secuencia de elementos para los que está definido un orden, $A = (a_1, a_2, \ldots, a_n)$. Las subsecuencias crecientes de A son las secuencias $(a_{h_1}, a_{h_2}, \ldots, a_{h_p})$ que cumplen $1 \leq h_1 < h_2 < \cdots < h_p \leq n$ y $a_{h_i} < a_{h_{i+1}}$. O sea, son secuencias crecientes de elementos de A que aparecen en el mismo orden en que aparecen en A. Se quiere encontrar la subsecuencia creciente más larga de A (LIS, por su nombre en inglés *Longest Increasing Subsequence*). Para cada prefijo $A_i = (a_1, a_2, \ldots, a_i)$, $1 \leq i \leq n$, de A se define G_i como el largo de la subsecuencia creciente más larga de A_i y L_i como el largo de la subsecuencia creciente más larga de A_i que termina en a_i.

Ejemplo: Dado $A = (5, 2, 4, 7, 1, 6, 3)$, los valores de G_i son $1, 1, 2, 3, 3, 3, 3$ y los de L_i son $1, 1, 2, 3, 1, 3, 2$. La subsecuencia más larga que termina en a_7 puede ser tanto $(2, 3)$ como $(1, 3)$. Las subsecuencias $(5, 6)$ y $(1, 6)$ son subsecuencias crecientes que terminan en a_6, pero $L_6 = 3$ porque también existe la subsecuencia creciente $(2, 4, 6)$, que se obtiene al incluir a_6 al final de la subsecuencia creciente más larga que termina en a_3 (lo cual es válido ya que $a_3 < a_6$).

Se puede ver que $\{G_i\}$ cumple la relación de recurrencia $G_0 = 0$, $G_i = \max\{G_{i-1}, L_i\}$ para $1 \leq i \leq n$.

(I) Complete las relaciones de recurrencia, incluyendo L_i.

Nota: Para la notación se asume que el máximo de un conjunto vacío es 0.

(II) Dada una secuencia de bandejas rectangulares, $B = (b_i, \ldots, b_n)$ un mozo se propone construir la pila de mayor cantidad de bandejas posibles. Sólo puede apoyar una bandeja sobre otra si sus dos dimensiones, ancho y largo, son menores que las de la bandeja en la que se apoya. Se puede suponer que los anchos de todas las bandejas son diferentes y que los largos de todas las bandejas son diferentes, o sea, $b_i.w \neq b_j.w$ y $b_i.\ell \neq b_j.\ell$ cuando $i \neq j$, donde $b_h.w$ y $b_h.\ell$ son respectivamente el ancho y el largo de la bandeja b_h. Resuelva el problema de calcular el tamaño de la pila solución. Para ello, transforme este problema en el de la subsecuencia creciente más larga.

Solución:

(I)

$$L_i = 1 + \max_{\substack{1 \leq j < i \\ a_j < a_i}} \{L_j\}, \qquad 1 \leq i \leq n.$$

$$G_0 = 0,$$
$$G_i = \max\{G_{i-1}, L_i\} \qquad 1 \leq i \leq n.$$

(II) Cualquier pila factible recorrida desde el tope hasta el fondo es una secuencia de bandejas en las que tanto los anchos como los largos están ordenados de manera creciente.

Se ordena B según una de las dimensiones, por ejemplo el ancho. Se obtiene una permutación de B, $B' = (b'_1, \ldots, b'_n)$ con $b'_i.w < b'_{i+1}.w$, $1 \leq i < n$. Para que una pila sea factible es condición necesaria que sea una subsecuencia de B'.

Para que la pila también esté ordenada según el largo se considera la secuencia $B'.\ell = (b'_1.\ell, \ldots, b'_n.\ell)$. Las pilas factibles se corresponden con las subsecuencias crecientes de esta secuencia. Por lo tanto el largo de la pila solución es el largo de la subsecuencia creciente más larga de $B'.\ell$.