FEBRERO 2019 – SEMANA 2 (MODELO A)

Grado en Ingeniería Informática y Grado en Ingeniería en Tecnologías de la Información

Normas de valoración del examen:

- La nota del examen representa el 80% de la valoración final de la asignatura (el 20% restante corresponde a las prácticas).
- Cada cuestión contestada correctamente vale 1 punto.
- Cada cuestión contestada incorrectamente baja la nota en 0.3 puntos.
- Debe obtenerse un mínimo de 3 puntos en las cuestiones para que el problema sea valorado (con 3 cuestiones correctas y alguna incorrecta el examen está suspenso).
- La nota total del examen debe ser al menos de 4.5 para aprobar.
- **Las cuestiones se responden en una hoja de lectura óptica**.

Examen tipo A:

Cuestiones:

1. Con respecto a la resolución de colisiones en las funciones Hash se puede afirmar que:

   (a) El recorrido cuadrático es aplicable siempre que se trate de hashing abierto.
   (b) Si el factor de carga es 1 se puede resolver mediante hashing cerrado.
   (c) El hashing abierto contempla un método de resolución conocido como "recorrido mediante doble hashing".
   (d) Ninguna de las anteriores es correcta.

a) **Falso**. El recorrido cuadrático se da en el caso de recorrido lineal, donde la probabilidad de nuevas colisiones es bastante alta para determinados patrones de claves. Hay otro método basado en una expresión cuadrática g(k) que permite mayor dispersión de las colisiones por la tabla, al mismo tiempo que proporciona un recorrido completo por la misma.

b) **Falso**. La elección del método Hashing Abierto y Hashing Cerrado dependen de varios factores, entre ellos del factor de carga.

El factor de carga es un valor entre 0 (tabla hash vacía) o 1 (tabla hash llena) que mide la proporción de la tabla Hash ya ocupada. Su fórmula es la siguiente, donde M es el tamaño de la tabla, y n el número de índices ocupados:

$$\delta = \frac{n}{M}$$

Normalmente es necesario extender la tabla con entradas cuando se ha superado el factor 0'5. Si no es así, lo que se realiza en caso de colisión es un recorrido de entradas vacías hasta encontrar una.

Si el factor de carga es 1 se puede resolver doblando el tamaño de la tabla mediante Hashing Abierto.

c) Falso. El Hashing Cerrado resuelve las colisiones mediante algunos de los siguientes métodos:

- Recorrido Lineal
- Recorrido Cuadrático

- Recorrido mediante Doble Hashing

Respuesta D)

2. Durante la ejecución de un algoritmo de minimización por Ramificación y Poda, la estimación optimista de la cima del montículo es mayor o igual que la cota superior hasta el momento. Esto implica:

   (a) Definitivamente el algoritmo ha terminado.
   (b) Se actualiza la cota superior con la cima del montículo y se sigue con la exploración porque no hemos terminado.
   (c) Se descarta la cima del montículo y se sigue con el siguiente elemento del montículo porque no hemos terminado.
   (d) Ninguna de las anteriores es correcta.

Respuesta A)

3. Sea el problema de la mochila en su versión de objetos no fraccionables solucionado mediante programación dinámica. Suponga que se dispone de 4 objetos con volúmenes {1,2,5,6} y que aportan unos beneficios de {1,5,15,20}, respectivamente. Suponga también que dispone de una mochila con una capacidad máxima de 12. Indique cuál sería el contenido de la tabla de resultados en la fila correspondiente al objeto de volumen 6, si dichos objetos se consideran en orden creciente de volúmenes.

   (a) 0 1 5 6 6 15 16 20 21 25 26 35 36
   (b) 0 1 5 6 6 15 20 21 25 25 26 35 36
   (c) 0 1 5 6 6 15 21 25 25 26 26 35 36
   (d) Ninguna de las anteriores.

		CAPACIDADES												
	0		**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	
		0	0	0	0	0	0	0	0	0	0	0	0	0	**BENEFICIOS**
PESOS	**1**	0	1	1	1	1	1	1	1	1	1	1	1	1	**1**
	2	0	1	5	6	6	6	6	6	6	6	6	6	6	**5**
	5	0	1	5	6	6	15	16	20	21	21	21	21	21	**15**
	6	0	1	5	6	6	15	20	21	25	26	26	35	36	**20**

Para realizar el cálculo de cada celda, se realiza como:

Valor máximo entre

- Valor de la misma columna en la fila anterior
- Beneficio del Peso + X

X = Valor en la columna Y de la fila anterior
Y = Capacidad de la mochila (C) – Peso

Respuesta D)

4. En relación al esquema algorítmico de vuelta atrás. ¿Cuál de las siguientes afirmaciones es **cierta**?:

   (a) Implementa un recorrido en amplitud de forma recursiva sobre el grafo implícito del problema.
   (b) El esquema de vuelta atrás siempre encontrará una solución al problema, si es que ésta existe.
   (c) Siempre es más eficiente que el esquema voraz, de forma que, si ambos son aplicables, nos decantaremos por utilizar el esquema de vuelta atrás.
   (d) Todas las afirmaciones anteriores son falsas.

a) **Falso**. Realiza un recorrido en profundidad del grafo implícito del un problema, podando aquellas ramas para las que el algoritmo puede comprobar que no pueden alcanzar una solución al problema.
b) **Cierto**. El objetivo del algoritmo puede ser encontrar una solución o encontrar todas las posibles soluciones al problema. En el primer caso el algoritmo se detiene al encontrar la primera solución, evitando así la construcción del resto del grafo del problema, lo que puede suponer una gran ganancia en eficiencia.
c) **Falso**. El esquema voraz es siempre más eficiente que el esquema de vuelta atrás.
d) **Falso**. La respuesta **b** es la correcta.

Respuesta B)

5. Con respecto al algoritmo de Prim indique cuál de estas afirmaciones es **falsa**:

   (a) El algoritmo parte necesariamente de un nodo que hay que seleccionar y que será la raíz del árbol de recubrimiento.
   (b) El algoritmo finaliza cuando el árbol de recubrimiento contiene *n-1* aristas, siendo *n* el número de nodos.
   (c) Si para representar el grafo se utiliza una lista de adyacencia junto con un montículo para representar los candidatos pendientes el coste del algoritmo es *O(a log n)*, lo que resulta más apropiado que usar una matriz de adyacencia cuando el grafo es denso.
   (d) La función de selección en cada paso añade al árbol de recubrimiento una arista de coste mínimo tal que la estructura resultante sigue siendo un árbol.

a) **Cierto**. El algoritmo de Prim necesita partir de un nodo, el cual hay que seleccionar, siendo el nodo raíz del árbol de recubrimiento.
b) **Cierto**. Según el esquema detallado del algoritmo de Prim, el bucle se ejecuta hasta que el árbol de recubrimiento contiene n-1 aristas, donde *n* es el número de nodos.
c) **Falso**. Si el grafo se implementa mediante listas de adyacencia, y se utiliza un montículo para representar los candidatos pendientes, el coste será de O(n log n + a log n) = O(a log n). Esto resulta más apropiado que usar una matriz de adyacencia solo cuando el grafo es disperso (Un grafo es disperso cuando el número de aristas es cercano a n). Si el grafo fuese denso (número de aristas cercano a n^2), el coste sería O(n^2 log n), el cual es peor que O(n^2).
d) **Cierto**. En cada paso del algoritmo de Prim se añade al árbol una arista de coste mínimo (u, v) tal que AR U {(u, v)} sea también un árbol. Nótese que la arista (u, v) que se selecciona en cada paso es tal que o bien u o v está en AR.

Respuesta C)

6. Indique cuál de las siguientes afirmaciones es **falsa** con respecto al esquema de programación dinámica:

   (a) El objetivo de este esquema es la reducción del coste del algoritmo mediante la memorización de soluciones parciales que se necesitan para llegar a la solución final.
   (b) Es igual de eficiente que el esquema divide y vencerás cuando en este último esquema se dan llamadas recursivas que se repiten en la secuencia de llamadas recursivas.
   (c) El problema de la multiplicación asociativa de matrices resuelto con programación dinámica tiene un coste temporal de $O(N^3)$, siendo *N* el número de matrices para multiplicar.
   (d) El problema de la distancia de edición entre dos cadenas resuelto con programación dinámica tiene un coste temporal de O(nm), siendo *n* la longitud de una cadena y *m* la longitud de la otra.

a) **Cierto**. El esquema de la Programación Dinámica se caracteriza por registrar resultados parciales que se producen durante la resolución de algunos problemas y que se utilizan repetidamente. De esta forma se consigue reducir el coste del cómputo, al evitar la repetición de ciertos cálculos.

b) **Falso**. Algunos casos en los que es aplicable la Programación Dinámica, estos también pueden abordarse con el esquema Divide y Vencerás. En la Programación Dinámica un problema se va dividiendo en subproblemas de tamaño menor que el original, planteándose dichas divisiones en forma de algoritmos recursivos, los cuales, para que sean eficientes, deben cumplir la condición de que no haya llamadas repetidas en la secuencia de llamadas recursivas. Cuando ocurren las repeticiones, conviene recurrir al esquema de programación dinámica, almacenando los resultados ya calculados para reutilizarlos.

c) **Cierto**. Para calcular el coste temporal, en el algoritmo hay dos bucles anidados, cada uno de ellos de orden O(N), dentro de los cuales se llama a la función *MinMultiple*, que también es de orden O(N). Por tanto, la complejidad temporal está en $O(N^3)$. Por otra parte, el coste espacial está en $O(N^2)$, ya que construimos una tabla NxN.

d) **Cierto**. El coste temporal del algoritmo viene dado por los dos bucles anidados que se utilizan para construir la tabla, y es por tanto O(nm). El coste temporal, dado por el tamaño de la tabla, es el mismo.

Respuesta B)

Problema (4 puntos). Resolver el problema de cálculo de la subsecuencia de mayor valor de un vector de enteros: Dado un vector *a[1..n]* de enteros, se pide diseñar un algoritmo eficiente – mejor que $O(n^2)$– que encuentre la subsecuencia de suma máxima dentro del vector.

$$\max_{1 \le i \le j \le n} \left\{ \sum_{k=i}^{j} a_k \right\}$$

Por ejemplo, en el vector a = [-2, 11, -4, 13, -5 , -2] la solución al problema es i=2 y j=4, con resultado $a_2 + a_3 + a_4 = 20$.

La resolución de este problema debe incluir, por este orden:

1. Elección razonada del esquema más apropiado de entre los siguientes: Voraz, Divide y Vencerás, Vuelta Atrás o Ramificación y Poda. Escriba la estructura general de dicho esquema e indique cómo se aplica al problema (0,5 puntos).

   El esquema más apropiado es Divide y Vencerás, cuyo esquema general es el siguiente:

```
fun DyV(problema)
    si trivial(problema) entonces
        dev solución-trivial
    sino hacer
        {p1, p2, ...pk} ← descomponer(problema)
        para i ∈ (1..k) hacer
            si ← DyV(pi)
        fpara
    fsi
    dev combinar(s1, s2, ...sk)
ffun
```

2. Descripción de las estructuras de datos necesarias (0,5 puntos solo si el punto 1 es correcto).

   Dividiremos el problema en tres subproblemas más pequeños, sobre cuyas soluciones construiremos la solución total.

   En este caso la subsecuencia de suma máxima puede encontrarse en uno de tres lugares. O está en la primera mitad del vector, o en la segunda, o bien contiene al punto medio del vector y se encuentra en ambas mitades. Las tres soluciones se combinan mediante el cálculo de su máximo para obtener la suma pedida.

   Los dos primeros casos pueden resolverse recursivamente. Respecto al tercero, podemos calcular la subsecuencia de suma máxima de la primera mitad que contenga al último elemento de esa primera mitad, y la subsecuencia de suma máxima de la segunda mitad que contenga al primer elemento de esa segunda mitad. Estas dos secuencias pueden concatenarse para construir la subsecuencia de suma máxima que contiene al elemento central de vector. Esto da lugar al siguiente algoritmo:

```
PROCEDURE Sumamax3(VAR a:vector;prim,ult:CARDINAL):CARDINAL;
  VAR mitad,i:CARDINAL;
      max_izq,max_der,suma,max_aux:INTEGER;
BEGIN
  (* casos base *)
  IF prim>ult THEN RETURN 0 END;
  IF prim=ult THEN RETURN Max2(0,a[prim]) END;
  mitad:=(prim+ult)DIV 2;
  (* casos 1 y 2 *)
  max_aux:=Max2(Sumamax3(a,prim,mitad),Sumamax3(a,mitad+1,ult));
  (* caso 3: parte izquierda *)
  max_izq:=0;
  suma:=0;
  FOR i:=mitad TO prim BY -1 DO
    suma:=suma+a[i];
    max_izq:=Max2(max_izq,suma)
  END;
  (* caso 3: parte derecha *)
  max_der:=0;
  suma:=0;
  FOR i:=mitad+1 TO ult DO
    suma:=suma+a[i];
    max_der:=Max2(max_der,suma)
  END;
  (* combinacion de resultados *)
  RETURN Max2(max_der+max_izq,max_aux)
END Sumamax3;
```

la función Max2 utilizada es la que calcula el máximo de dos números enteros. El procedimiento Sumamax3 es de complejidad O(n log n), puesto que su tiempo de ejecución T(n) viene dado por la ecuación en recurrencia

$$T(n) = 2T(n/2) + An$$

con la condición inicial T(1) = 7, siendo A una constante. Obsérvese además que este análisis es válido puesto que hemos añadido la palabra VAR al vector a en la definición del procedimiento. Si no, se producirían copias de a en las invocaciones recursivas, lo que incrementaría el tiempo de ejecución del procedimiento.

3. Algoritmo completo a partir del refinamiento del esquema general (2.5 puntos sólo si el punto 1 es correcto). Si se trata del esquema voraz debe hacerse la demostración de optimalidad.

   La clave para encontrar un algoritmo aún mejor (refinado), va a consistir en una modificación a la idea básica del algoritmo anterior, basada en un algoritmo del tipo "en línea" (véase el problema del elemento mayoritario, en este capítulo).

   Supongamos que ya tenemos la solución del problema para el subvector a[prim..i–1]. ¿Cómo podemos extender esa solución para encontrar la solución de a[prim..i]? De forma análoga al razonamiento que hicimos para el algoritmo anterior, la subsecuencia de suma máxima de a[prim..i] puede encontrarse en a[prim..i–1], o bien contener al elemento a[i]. Esto da lugar a la siguiente función:

```
PROCEDURE Sumamax4(VAR a:vector;prim,ult:CARDINAL):CARDINAL;
   VAR i:CARDINAL;
       suma,max_anterior,max_aux:INTEGER;
BEGIN
   max_anterior:=0;
   max_aux:=0;

   FOR i:=prim TO ult DO
      max_aux:=Max2(max_aux+a[i],0);
      max_anterior:=Max2(max_anterior,max_aux)
   END;
   RETURN max_anterior;
END Sumamax4;
```

   Este algoritmo no es intuitivo ni fácil de entender a primera vista. La clave del algoritmo se encuentra en la variable max_aux, que representa el valor de la suma de la subsecuencia de suma máxima del subvector a[prim..i] que contiene al elemento a[i], y que se calcula a partir de su valor para el subvector a[prim..i–1].

   Este valor se incrementa en a[i] mientras que esa suma sea positiva, pero vuelve a valer cero cada vez que se hace negativa, indicando que la subsecuencia de suma máxima que contiene al elemento a[i] es la subsecuencia vacía.

4. Estudio del coste del algoritmo desarrollado (0,5 puntos solo si el punto 1 es correcto).

   El algoritmo resultante es de complejidad lineal. Se puede encontrar un análisis detallado de la función en el libro:

   Autor: D. Gries.
   Título: *A Note on the Standard Strategy for Developing Loop Invariants and Loops*.
   Apartado: Science of Computer Programming, nº 2, páginas 207–214.