Examen de Programación 3 y III (26/02/2009)

Instituto de Computación, Facultad de Ingeniería
SOLUCION

Ejercicio 1. (36 puntos)

a) Considere el siguiente algoritmo:

1. Mejor caso: se da cuando la invocación f2(m, A) retorna falso, f1 solo se ejecuta una vez.
   Peor caso: se da cuando la invocación de f2(m, A) retorna verdadero para todo $0 < i < m$.

2. Mejor caso: se da cuando la invocación f2(m, A) retorna falso (idem anterior).
   Peor caso: se da cuando la invocación de f2(m, A) retorna verdadero para todo $1 < i < m$ ya en el caso m=1 no importa si f2 da verdadero o falso.

3.

T(0) = 0
T(1) = 0
T(2) = 1 + T(1) = 1
T(3) = 1 + T(2) + T(2) = 3
T(4) = 1 + T(3) + T(3) + T(3) = 1+3+3+3 = 10
…
T(m) = (m-1)+(m-1)*T(m-1)
T(m-1) = (m-2)+(m-2)* T(m-2)

T(m) = (m-1)+(m-1)*((m-2)+(m-2)* T(m-2))

$$\Rightarrow T(m) = (m-1) + (m-1)(m-2) + (m-1)(m-2) * T(m-2)$$

$$T(m-2) = (m-3) + (m-3) * T(m-3)$$
$$\Rightarrow T(m) = (m-1) + (m-1)(m-2) + (m-1)(m-2)((m-3) + (m-3) * T(m-3))$$
$$= (m-1) + (m-1)(m-2) + (m-1)(m-2)(m-3) + (m-1)(m-2)(m-3) * T(m-3)$$

…

$$T(m) = (m-1) + (m-1)(m-2) + (m-1)(m-2)(m-3) + ... + (m-1)(m-2)(m-3)...4 * 3$$

$$T(m) = \sum_{i=2}^{m-2} (m-1)!/i!$$

4.

El `for` no tiene ningún efecto ya que el código dentro de su bloque no depende de `i`. Por lo que es equivalente al siguiente código.

```
bool f1(int m, double* A){

    bool ret = true;
    if (f2(m,A)&&(m>1)){
        ret = ret && f1(m-1, A);
    }
    return ret;
}
```

Inspeccionando el código más finamente se puede ver que devuelve siempre true, por lo que un código que devuelva siempre `true` y que cumpla $t1(m) \in \Theta(m)$ es suficiente. Por ejemplo:

```
bool f1(int m, double* A){
        for(int i=1; i<m; i++)
            f2(m,A);
    return true;
}
```

b)

b1)

1. $f \in \Theta(g) \Leftrightarrow g \in \Theta(f)$

Demostración:

($\Rightarrow$) H) $f(n) \in \Theta(g(n))$

T) $g(n) \in \Theta(f(n)) \underset{def}{\Leftrightarrow} g(n) \in O(f(n)) \wedge g(n) \in \Omega(f(n))$

Por hipótesis y definición de Θ:

$$\left.\begin{array}{l} f(n) \in O(g(n)) \Rightarrow \exists c, n_0 / f(n) \le cg(n) \\ f(n) \in \Omega(g(n)) \Rightarrow \exists c_1, n_1 / f(n) \ge c_1 g(n) \end{array}\right\} c_1 g(n) \le f(n) \le cg(n)$$

$$\underline{c_1 g(n) \le f(n)} \le cg(n) \Leftrightarrow c_1 g(n) \le f(n) \Leftrightarrow g(n) \le \frac{1}{c_1} f(n)$$

Tomo $c_2 = \left\lceil \frac{1}{c_1} \right\rceil, n_2 \ge n_1 \Rightarrow g(n) \in O(f(n))$ ✓

Por otro lado:

$$c_1 g(n) \le \underline{f(n) \le cg(n)} \Leftrightarrow f(n) \le cg(n) \Leftrightarrow \frac{1}{c} f(n) \le g(n)$$

Tomando $c_3 = \left\lceil \frac{1}{c} \right\rceil, n_3 \ge n_0 \Rightarrow g(n) \in \Omega(f(n))$ ✓

(⇐) H) $g(n) \in \Theta(f(n))$

T) $f(n) \in \Theta(g(n))$

Por hipótesis: $g(n) \in \Theta(f(n)) \underset{def}{\Leftrightarrow} \begin{cases} g(n) \in O(f(n)) \underset{def}{\Leftrightarrow} \exists c, n_0 / g(n) \le c \cdot f(n), n \ge n_0 \\ g(n) \in \Omega(f(n)) \underset{def}{\Leftrightarrow} \exists c_1, n_1 / g(n) \ge c_1 \cdot f(n), n \ge n_1 \end{cases} \Rightarrow$

$\underline{c_1 f(n) \le g(n)} \le cf(n) \Rightarrow c_1 f(n) \le g(n) \Leftrightarrow f(n) \le \left\lceil \frac{1}{c_1} \right\rceil g(n)$

Tomando $c_2 = \left\lceil \frac{1}{c_1} \right\rceil$ y $n_2 \ge n_1 \Rightarrow f(n) \in \Omega(g(n))$ ✓

También se tiene que:

$c_1 f(n) \le \underline{g(n) \le cf(n)} \Rightarrow g(n) \le cf(n) \Leftrightarrow \left\lceil \frac{1}{c} \right\rceil g(n) \le f(n)$

Tomando $c_3 = \left\lceil \frac{1}{c} \right\rceil$ y $n_3 \ge n_2 \Rightarrow f(n) \in O(g(n))$ ✓

2. $(n+a)^b \in \Theta(n^b)$

$n^5 2^n \in \Omega(n^{10}) \underset{def.\Omega}{\Leftrightarrow} \exists c \in R^+, \exists n_0 \in N, \forall n \in N, n > n_0, n^5 2^n \ge cn^{10}$

Luego $2^n \ge cn^5$ sea $c = 1 \Rightarrow n \ge 5\log_2(n)$.

Finalmente, se debe encontrar n_0 tal que se cumpla $n \ge 5\log_2(n) \nabla n \ge n_0$.
Una forma de determinar dicho n_0 es derivar la ecuación $n \ge 5\log_2(n)$ y encontrar los puntos de corte.
Otra manera es utilizar $n_0 = 2^{10}$, el cual es facil comprobar que cumple la desigualdad previamente mencionada.

Demostración:

Se conoce que si $h_1(n) \le f(n) \le h_2(n)$ y $h_1(n) \in \Omega(g(n))$ y $h_2(n) \in O(g(n))$ luego $f(n) \in \Theta(g(n))$

Sea $h_1(n) = n^b$ y $h_2(n) = (n+a)^b$

Por definición de Ω se sabe que $n^b \in \Omega(n^b)$.

Además
$(n+a)^b = n^b + An^{b-1} + Bn^{b-2} + Cn^{b-3} + ... + an^0 = n^b + An^{b-1} + Bn^{b-2} + Cn^{b-3} + ... + a$,
que es un polinomio de grado b en n. Con lo cual $(n+a)^b \in O(n^b)$
De lo anterior $(n+a)^b \in \Theta(n^b)$

3. $$\left.\begin{array}{l} h_1(n) \le f(n) \le h_2(n) \forall n \ge n_0 \\ h_1(n) \in \Omega(g(n)) \\ h_2(n) \in \mathrm{O}(g(n)) \end{array}\right\} \Rightarrow f(n) \in \Theta(g(n))$$

Demostración:

$$f(n) \in \Theta(g(n)) \Leftrightarrow \begin{cases} f(n) \in \Omega(g(n)) \\ f(n) \in \mathrm{O}(g(n)) \end{cases}$$

$$\left.\begin{array}{l} \underline{h_1(n) \le f(n)} \le h_2(n), \forall n \ge n_0 \\ h_1(n) \in \Omega(g(n)) \underset{def.\Omega}{\Rightarrow} \exists c \in R^+, \exists n_0 \in N, \forall n, n > n_0, cg(n) \le h_1(n) \end{array}\right\} cg(n) \le h_1(n) \le f(n)$$
$$\Rightarrow cg(n) \le f(n), \forall n > n_0 \underset{def.\Omega}{\Rightarrow} f(n) \in \Omega(g(n)) \quad \checkmark$$

$$\left.\begin{array}{l} h_1(n) \le \underline{f(n) \le h_2(n)}, \forall n \ge n_0 \\ h_2(n) \in \mathrm{O}(g(n)) \underset{def.\mathrm{O}}{\Rightarrow} \exists c' \in R^+, \exists n_0' \in N, \forall n, n > n_0', h_2(n) \le c' g(n) \end{array}\right\} f(n) \le h_2(n) \le c' g(n)$$
$$\Rightarrow f(n) \le c' g(n), \forall n > n_0' \underset{def.\mathrm{O}}{\Rightarrow} f(n) \in \mathrm{O}(g(n)) \checkmark$$

4. $$\lim \frac{f(n)}{g(n)} = L \wedge 0 < L < n \Rightarrow f(n) \in \Theta(g(n))$$

Demostración:

$$f(n) \in \Theta(g(n)) \Leftrightarrow f(n) \in \mathrm{O}(g(n)) \wedge f(n) \in \Omega(g(n))$$

I. $$f(n) \in \mathrm{O}(g(n)) \Leftrightarrow \exists c \in R^+, \exists n_0 \in N, n > n_0, g(n) \le cf(n)$$

Por hipótesis, $\lim \frac{f(n)}{g(n)} = L, 0 < L < n \Rightarrow f(n) = Lg(n) \Rightarrow g(n) = \frac{1}{L} f(n) \Rightarrow g(n) \le \frac{1}{L} f(n)$,
considerando $c = \frac{1}{L}$, se cumple la definición de O, entonces: $f(n) \in \mathrm{O}(g(n))$

II. $$f(n) \in \mathrm{O}(g(n)) \Leftrightarrow \exists c' \in R^+, \exists n_0 \in N, n > n_0, g(n) \ge c' f(n)$$

Análogamente, por hipótesis:

$\lim \frac{f(n)}{g(n)} = L, 0 < L < n \Rightarrow f(n) = Lg(n) \Rightarrow g(n) = \frac{1}{L} f(n) \Rightarrow g(n) \geq \frac{1}{L} f(n)$, considerando $c' = \frac{1}{L}$, se cumple la definición de Ω, entonces: $f(n) \in \Omega(g(n))$

b2)

1. $n\log(n) \in \mathrm{O}(n^2)$

$n\log(n) \in \mathrm{O}(n^2) \underset{def.\,\mathrm{O}}{\Leftrightarrow} \exists c \in R^+, \exists n_0 \in N, \forall n \in N, n > n_0, n\log(n) \leq cn^2$

Si tomamos c = 1, sabiendo que $\log(n) \leq n, \forall n \in \mathrm{N}$, se cumple:
$n\log(n) \leq n^2 \ \forall n \in \mathrm{N} \Rightarrow n\log(n) \in \mathrm{O}(n^2)$ ✓

2. $n^5 2^n \in \Omega(n^{10})$

Ejercicio 2. (32 puntos)

a. (12 puntos) Formalización

Forma de la solución
Tupla de largo fijo N, de la forma $<x_0, x_1, \ldots, x_{N-1}>$ donde x_i indica si la electiva *i* forma parte de la solución.

Restricciones explícitas
Todos los elementos de la tupla pertenecen al conjunto {0,1}, si la electiva *i* forma parte de la solución $x_i = 1$, de lo contrario $x_i = 0$:

- $x_i \in \{0,1\} \forall i, 0 \leq i \leq n-1$.

Restricciones implícitas
El estudiante conseguir al menos 450 créditos:

- $creditosEstudiante + \sum_{i=0}^{i=N-1} x[i] \times creditosAsignatura(codElectivas[i]) \geq 450$.

Función objetivo

La función objetivo 1 es:
$f = \min(cantElectivas(t))$, donde '*cantElectivas*' es la cantidad de electivas incluidas en la tupla solución $t=<x_0, x_1, \ldots, x_{n-1}>$:

$cantElectivas(t) = \sum_{i=0}^{i=N-1} x_i$

- $f = \min_{t \in T}(cantElectivas(t))$, donde T = { $t = <x_0, x_1, ..., x_{n-1}>$ / t es factible }

Sea S el conjunto de todas las tuplas que minimizan la función objetivo 1, o sea, $S = \{t / cantElectivas(t) = f\}$

La función objetivo 2 es:
$g = \max(cantElectivasP(t))$, donde '*cantElectivasP'* es la cantidad de electivas del área de Programación incluidas en la tupla solución $t=<x_0, x_1, ..., x_{n-1}>$ y además $t \in S$:

- $g = \max_{t \in S}(cantElectivasP(t))$

Predicado de poda

Se poda una rama si la tupla en construcción $<x_0, \ldots, x_k, 0, \ldots, 0>$ contiene más electivas que la mejor solución hasta el momento:

- $cantElectivas(< x_0, x_1, ..., x_k, 0, ..., 0 >) > cantElectivas(< y_0, y_1, ..., y_{n-1} >)$
  $\forall k, 0 \le k \le n-1$, siendo $<y_0, y_1, ..., y_{n-1}>$ una tupla factible y la mejor hasta el momento.

Se poda una rama si la tupla en construcción $<x_0, \ldots, x_k, 0, \ldots, 0>$ contiene la misma cantidad de electivas que la mejor solución hasta el momento, pero contiene menos electivas de Programación que ésta:

- $cantElectivas(< x_0, x_1, ..., x_k, 0, ..., 0 >) = cantElectivas(< y_0, y_1, ..., y_{n-1} >)$
  y
  $cantElectivasP(< x_0, x_1, ..., x_k, 0, ..., 0 >) < cantElectivasP(< y_0, y_1, ..., y_{n-1} >)$
  $\forall k, 0 \le k \le n-1$, siendo $<y_0, y_1, ..., y_{n-1}>$ una tupla factible y la mejor hasta el momento, y donde '*cantElectivasP*' representa la cantidad de electivas del área de Programación que contiene la tupla.

Se poda una rama si la cantidad de créditos del estudiante más la cantidad de créditos de la tupla en construcción $<x_0, \ldots, x_k, 0, \ldots, 0>$ más los créditos de todas las electivas que aún no han sido consideradas no supera los 450 créditos:

$$creditosEstudiante + \sum_{i=0}^{i=k} x[i] \times creditosAsignatura(codElectivas[i]) + \sum_{i=k+1}^{i=N-1} creditosAsignatura(codElectivas[i]) < 450$$

b. (20 puntos)

```
struct Tupla {
    int* electivas;          //tupla
    int cantElectivas;       //cantidad de electivas que contiene la tupla
    int cantProg;            /*
                                cantidad de electivas del area de Programacion que
                                contiene la tupla.
                             */
    int creditos;            //cantidad de creditos incluidos en la tupla
```

```
}

Tupla electivasMejorOpcion(int creditosEstudiante, int* codElectivas) {
    int creditosRestantes = 0;
    for(int i = 0; i < N; i++)
          creditosRestantes += creditosAsignatura(codElectivas[i]);
    Tupla t, sol;
    t.electivas = new int[N];
    sol.electivas = new int[N]
    for(int i = 0; i < N; i++) {
          t.electivas[i] = 0;
          t.sol[i] = 0;
    }
    t.cantElectivas = 0;
    sol.cantElectivas = MAX_INT;
    t.cantProg = 0;
    sol.cantProg = 0;
    t.creditos = 0;
    sol.creditos = 0;
    elegirElectivas(creditosEstudiante, codElectivas, t, 0, sol,
          creditosRestantes);
    //sol es la tupla solucion
    return sol;
}
```

```
void elegirElectivas(int creditosEstudiante, int* codElectivas, Tupla t,
          int actual, Tupla &sol, int creditosRestantes) {
    if (creditosEstudiante + t.creditos + creditosRestantes >= 450) {
          //Predicado de poda y restriccion implicita
          if ((t.cantElectivas < sol.cantElectivas) ||
                ((t.cantElectivas == sol.cantElectivas) &&
                (t.cantProg > sol.cantProg))) {     //Funcion objetivo
                if (actual == N) {      //Tupla completa
//                    if (creditosEstudiante + t.creditos >= 450) {
                          /*
                          Restriccion implicita (no es necesaria porque se
                          chequea junto con el predicado de poda)
                          */
                          for(int i = 0; i < N; i++)
                                sol.electivas[i] = t.electivas[i];
                          sol.cantElectivas = t.cantElectivas;
                          sol.cantProg = t.cantProg;
                          sol.creditos = t.creditos;
//                    }
                } else {
                      creditosRestantes -=
                            creditosAsignatura(codElectivas[actual]);
                      for(int i = 0; i < 2; i++) {
                            t.electivas[actual] = i;
                            t.cantElectivas += i;
                            if (codElectivas[actual] > 100)
                                  t.cantProg += i;
                            t.creditos +=
                                  i*creditosAsignatura(codElectivas[actual]);
                            elegirElectivas(creditosEstudiante, codElectivas, t,
                                  actual + 1, sol, creditosRestantes);
                            t.cantElectivas -= i;
                            if (codElectivas[actual] > 100)
                                  t.cantProg -= i;
                            t.creditos -=
                                  i*creditosAsignatura(codElectivas[actual]);
                      }
```

```
                    }
                }
            }
        }
```

Ejercicio 3. (32 puntos)

a. (15 puntos)

Sea *h* una función monótona creciente, tal que $h(x,i)$ representa el interés que se obtiene si se invierte el importe *x* en la empresa *i* con $0 \le i \le n-1$ y $x > 0$ con $x \in N$.

Sea $f(i,j)$ **el interés máximo total** que se obtiene al invertir j pesos en las primeras i empresas.

Los pasos base se dan cuando:

- la cantidad a invertir es 0 para cualquier empresa, es decir, $f(i,0) = 0 \quad con \quad 0 \le i \le n-1 \quad \wedge \quad j = 0$
- se tiene una única empresa para invertir j pesos y la cantidad a invertir es mayor a cero, es decir, $f(0,j) = h(j,0) \quad con \quad i = 0 \quad \wedge \quad j > 0$

En el paso recursivo se tiene la opción de invertir o no en la empresa i, es decir:

$$f(i,j) = \max\{f(i-1, j-p) + h(p,i), f(i-1,j)\} \quad con \ 0 < i \le n-1 \quad \wedge \quad 1 \le p \le j$$

Por lo tanto, la función *f* es de la forma:

$$f(i,j) = \begin{cases} 0 \quad con \quad 0 \le i \le n-1 \quad \wedge \quad j = 0 \\ h(j,0) \quad con \ i = 0 \quad \wedge \quad j > 0 \\ \max\{f(i-1, j-p) + h(p,i), f(i-1,j)\} \quad con \ 0 < i \le n-1 \quad \wedge \quad 1 \le p \le j \end{cases}$$

La solución se tiene con $f(n-1,d)$.

b. (10 puntos)

Para la implementación de la solución se utiliza una matriz de dimensiones: *n x d+1* donde *f[i][j]* indica el interés máximo que se puede obtener al invertir *j* pesos en hasta *i* empresas.

```
//se asume la funcion h(x,i) conocida.

/*
   Dada una cantidad de empresas (n) y dinero disponible (d) se
   devuelve el interés máximo.
*/
int inversionOptima(int n, int d){
      int i,j;

      int** f = new int*[n];
      for(i=0; i<n; i++){
            f[i]= new int[d+1];
      }

      //Paso base (1)
      for (i=0; i < n; i++){
            f[i][0]= 0;
      }

      //Paso base (2)
      for (j = 1; j <= d; j++){
            f[0][j] = h(j,0);
      }

      for (i=1; i< n; i++){
            //Obtención del óptimo para i empresas con j pesos
            for (int j=1; j<=d; j++){

                  /*
                     cantidad óptima a obtener con i empresas y j pesos
                     si no se invierte dinero en la empresa i.
                  */
                  f[i][j] = f[i-1][j];

                  int tmp;
                  for (int p = 1; p <= j; p++){
                        tmp = f[i-1][j-p] + h(p,i);
                        if (f[i][j] < tmp) {
                              f[i][j] = tmp;
                        }
                  }// fin del for
            } // fin del for
      } // fin del for

      int interes = f[n-1][d];

      // Libero la memoria
      for (i=0; i<n; i++)
               delete []f[i];

      delete []f;

      return interes;
}
```

c. (7 puntos)

Como se quiere obtener **la cantidad a invertir** en cada una de las *i* empresas para obtener el interés máximo con *j* pesos, se tiene que **mantener también la decisión óptima de la inversión p para cada empresa**, es decir, se tiene que mantener que dinero se tiene que invertir para cada empresa.

Para eso se define una matriz *cantInvertida[i][j]* la cual indica el dinero que se debe invertir en la empresa *i* para obtener la inversión óptima teniendo *j* pesos disponibles. La misma es cargada cada vez que se encuentra un óptimo durante la ejecución del algoritmo b).

Para devolver la cantidad a invertir en cada empresa usando *d* pesos y las *n* empresas, se debe definir un vector de tamaño n. Donde sabiendo que en *cantInvertida[n-1][d]* se encuentra la cantidad a invertir para la empresa *n-1* con *d* pesos disponibles. Luego se procede a obtener las cantidades de las restantes empresas de la siguiente manera:

```
int r = d;
int* res = new int[n];

for (i= n-1; i>-1; i--){
      res[i] = cantInvertida[i][r];
      r = r - cantInvertida[i][r];
}
```