Examen de Programación 3 y III (14/07/2009)

Instituto de Computación, Facultad de Ingeniería

1. Este examen dura 4 horas y contiene **2** carillas. El total de puntos es 100.
2. En los enunciados llamamos C^* a la extensión de C al que se agrega el operador de pasaje por referencia &, y las sentencias *new*, *delete* y el uso de *cout* y *cin*.
3. NO se puede utilizar ningún tipo de material de consulta. Puede usarse todo lo visto en el teórico, práctico y laboratorio sin demostrarlo, indicando claramente lo que se está usando.
4. No se contestaran dudas durante la última media hora.

Se requiere:

- Numerar todas las hojas e incluir en cada una el nombre y la cédula de identidad.
- Utilizar las hojas de un sólo lado y escribir con lápiz, iniciando cada ejercicio en hoja nueva.
- Poner en la primera hoja la cantidad de hojas entregadas, y un **índice** indicando en qué hoja se respondió cada problema.

Ejercicio 1. (35 puntos)

1. Para cada afirmación indique si es verdadera o falsa, en caso de ser verdadera demuestre sin utilizar la regla del límite, y en caso de ser falsa proporcione un contraejemplo.

1.1. $$\left.\begin{array}{l} f(n) \in O(h(n)) \\ g(n) \in O(h(n)) \end{array}\right\} \Rightarrow f(n) \in \Theta(g(n))$$

1.2. $$\left.\begin{array}{l} f(n) \in O(h(n)) \\ g(n) \in \Omega(h(n)) \\ g(n) \in O(f(n)) \end{array}\right\} \Rightarrow f(n) \in \Theta(g(n))$$

1.3. $$\left.\begin{array}{l} f(n) \in O(g(n)) \\ h(n) \in \Theta(g(n)) \end{array}\right\} \Rightarrow h(n) \in \Omega(f(n))$$

2. Dado el siguiente algoritmo:

```
bool funcion (int* a, int i, int j){                                  (1)
      int corte;                                                      (2)
      if (i == j)                                                     (3)
            return calcular(a, i, i);                                 (4)
      else {                                                          (5)
            if (i < j){                                               (6)
                  corte = (i+j-1)/3;                                  (7)
                  if(funcion(a, i, corte))                            (8)
                        return calcular(a, i, corte);                 (9)
                  else                                               (10)
                         if(funcion(a, corte + 1, j))                (11)
                              return calcular(a, corte + 1, j);      (12)
                         else {                                      (13)
                              return calcular(a, i, corte);          (14)
                         }
            }
      }
}
```

La función calcular tiene el siguiente encabezado: bool *calcular(int* a, int i, int j)*
Ambas funciones reciben como parámetro de entrada un arreglo *a* y dos índices *i* y *j* que representan que en esa invocación se consideran los elementos ubicados entre los lugares *i* y *j* del arreglo *a*.
Inicialmente se invoca a *funcion* con los siguientes parámetros: *funcion(a, 1, n)*

El procedimiento *calcular* es quién realiza las operaciones básicas, y la cantidad de dichas operaciones es:

- (3j-i +1) + j^2 si $i \neq j$.
- 3 si i = j

2.1. Analice las distintas alternativas en un paso genérico del algoritmo e indique la cantidad de operaciones básicas en cada alternativa.
Indique en cuales de las situaciones anteriores se da el peor caso.

2.2. Plantee la recurrencia que calcula la cantidad de operaciones básicas que se realizan en el peor caso definido en el punto anterior.

SOLUCIÓN

1.

1.1. FALSO

$f(n) = 1$

$g(n) = n$

$h(n) = n^2$

$1 \in O(n^2)$ y $n \in O(n^2)$

pero

$1 \notin \Omega(n)$

$\Rightarrow$

$1 \notin \Theta(n)$

1.2. VERDADERO

$g(n) \in \Omega(h(n))$

$\Rightarrow$

$h(n) \in O(g(n))$ además $f(n) \in O(h(n))$

$\Rightarrow$

$f(n) \in O(g(n))$ **(1)**

$g(n) \in O(f(n))$

$\Rightarrow$

$f(n) \in \Omega(g(n))$ **(2)**

por **(1)** y **(2)** $f(n) \in O(g(n)) \cap \Omega(g(n))$

$\Rightarrow$

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

1.3. VERDADERO

$h(n) \in \Theta(g(n))$

$\Rightarrow$

$g(n) \in \Theta(h(n))$

$\Rightarrow$

$g(n) \in O(h(n))$ además $f(n) \in O(g(n))$

$\Rightarrow$

$f(n) \in O(h(n))$

$\Rightarrow$

$h(n) \in \Omega(f(n))$

2.

2.1.

Para indicar el peor caso se analizan la cantidad de operaciones que se ejecutarían para cada condición de los if del algoritmo junto con sus sentencias.

Situación 1: la condición del if de la línea (3) evalúa true, por lo tanto la cantidad de operaciones básicas en esta situación es 3, ya que se invoca al procedimiento *calcular* con i = j.

Situación 2: la condición del if de la línea (3) evalúa false y las condiciones de los if (6) y (8) evalúan true. Se invoca el procedimiento *funcion* con tamaño de entrada $n/3$. La cantidad de operaciones básicas de la línea (9) son $n + \frac{n^2}{9}$. Por lo tanto la cantidad de operaciones básica en esta situación es $T(n) = T(n/3) + n + \frac{n^2}{9}$

Situación 3: la condición del if (3) y (8) evalúan false y las condiciones de los if (6) y (11) evalúan true. Se invoca el procedimiento *funcion* una vez con tamaño de entrada $n/3$ y otra con tamaño de entrada $2n/3$. La cantidad de operaciones básicas de la línea (12) son $n^2 + 8n/3$. Por lo tanto la cantidad de operaciones básica en esta situación es $T(n) = T(n/3) + T(2n/3) + n^2 + 8n/3$

Situación 4: la condición de los if de las líneas (3), (8) y (11) evalúan false y la condición del if (6) evalúa true. Se invoca el procedimiento *funcion* una vez con tamaño de entrada $n/3$ y otra con tamaño de entrada $2n/3$.

La cantidad de operaciones básicas de la línea (14) son $n + \frac{n^2}{9}$. Por lo tanto la cantidad de operaciones básica en esta situación es $T(n) = T(n/3) + T(2n/3) + n + \frac{n^2}{9}$

Por lo tanto el peor caso se da en la situación 3.

2.2. Plantee la recurrencia que calcula la cantidad de operaciones básicas que se realizan en el peor caso definido en el punto anterior.

$$T_w(1) = 3$$
$$T_w(n) = T(n/3) + T(2n/3) + n^2 + 8n/3$$

Ejercicio 2. (30 puntos)

Un delivery ofrece *n* menús diferentes, donde de cada menú se conoce: su precio, su contenido en proteínas y su contenido en calorías.

Dado un cliente que quiere mantener una dieta, se quiere hallar el valor calórico más bajo posible eligiendo algunos de los *n* menús, tal que el precio total de dicha elección no supere un presupuesto M, y a su vez, el valor proteico sea como mínimo W.

Téngase en cuenta que no se pueden repetir menús.

Asuma que se tienen definidas las siguientes funciones:

- p(k) indica el precio del k-ésimo menú con $1 \leq k \leq n$.
- q(k) indica el contenido en proteínas del k-ésimo menú con $1 \leq k \leq n$.
- c(k) indica el contenido en calorías del k-ésimo menú con $1 \leq k \leq n$.

Se pide:

Dar una fórmula **recursiva** para solucionar el problema. Llame a la fórmula recursiva *f*. Explicar que representa cada paso base y cada paso recursivo de la solución, como también que representa la función *f* indicando sus índices.

Sugerencia: tenga en cuenta para definir el/los caso/s base el hecho que no hayan mas menú disponibles, discriminando según el valor proteico sea mayor ó menor igual a cero.

SOLUCIÓN

Sea $f(i,m,w)$, el valor mínimo de calorías que se obtiene seleccionando entre *i menús* con $1 \leq i \leq n$, restringido al presupuesto máximo *m* y contenido proteico mínimo a cubrir *w*.

Los pasos base se dan cuando:

- No hay mas menús disponibles y no se ha cubierto el mínimo contenido proteico *w*. (Nótese que el caso representa la no existencia de solución):
  $f(0,m,w) = +\infty \quad con \quad w > 0$
- No hay mas menús disponibles y se ha cubierto el mínimo contenido proteico *w*:
  $f(0,m,w) = 0 \quad con \quad w \leq 0$

En cada paso recursivo (*i*), el presupuesto disponible podrá ser o no suficiente para cubrir el costo del menú *i*; donde caso de serlo, se tiene la opción de seleccionarlo o no. Es decir que los casos son:

- El dinero disponible no es suficiente para cubrir el costo del menú *i*, por lo que no se puede seleccionar ese menú:
  $f(i,m,w) = f(i-1,m,w) \quad si \quad p(i) > m \quad con \quad 0 < i \leq n$
- El dinero disponible es suficiente para cubrir el costo del menú *i* por lo que se tiene la opción de seleccionar o no el menú *i*, donde se optará por la opción que minimice el valor calórico:
  $f(i,m,w) = \min(f(i-1,m,w), c(i) + f(i-1,m-p(i),w-q(i))) \quad si \quad p(i) \leq m \quad con \quad 0 < i \leq n$

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

$$f(i,m,w) = \begin{cases} +\infty \quad si \quad i = 0 \quad \wedge \quad w > 0 \\ 0 \quad si \quad i = 0 \quad \wedge \quad w \leq 0 \\ f(i-1,m,w) \quad si \quad p(i) > m \quad \wedge \quad 0 < i \leq n \\ \min(f(i-1,m,w), c(i) + f(i-1,m-p(i),w-q(i))) \quad si \quad p(i) \leq m \quad \wedge \quad 0 < i \leq n \end{cases}$$

La llamada a la función para obtener el valor calórico mínimo es $f(n,M,W)$.

Ejercicio 3. (35 puntos)

Un chofer de una camioneta es contratado por una escuela para que pase a buscar a un conjunto de niños y los entregue a la misma. El recorrido que hace todos los días el chofer consta de ir desde su casa a la escuela pasando por la casa de todos los escolares que viajan en la camioneta.
Además, el recorrido debe hacerse en menos de 1 hora ya que sino los escolares llegarán tarde a la escuela.

Dado que el chofer desea entregar a los niños en hora a la escuela gastando la menor nafta posible, se desea minimizar la distancia recorrida logrando entregar a los niños en hora.

Se pide:
a) Formalizar el problema en términos de Backtracking.
Indicar: forma de la solución, restricciones explícitas e implícitas, predicados de poda y función objetivo.

NOTA: No se corregirá la parte b) si no se realizó **satisfactoriamente** la parte a)
b) Implementar una función en C* que resuelva el problema utilizando **Backtracking**.

Se dispone de las siguientes estructuras:

- tiempo[0..n+1][0..n+1], donde tiempo[i][j] indica el tiempo (en minutos) requerido para ir desde la casa del escolar i hasta la casa del escolar j.
- distancia[0..n+1][0..n+1], donde distancia[i][j] indica la distancia entre la casa del escolar i y la del escolar j.
- En ambos casos, 0 representa la casa del conductor y n+1 la escuela.

SOLUCIÓN

1.

Forma de la solución
Tupla de largo fijo N+2 de la forma $<x_0, x_1, \ldots, x_{N+1}>$ donde x_i indica el escolar que debe ser recogido en i-ésimo lugar.

Restricciones explícitas
Todos los elementos de la tupla pertenecen al conjunto {0, 1, …, N+1}

$$x_i \in \{0,1,...,N+1\} \quad \forall i, 0 \le i \le N+1$$

El inicio del recorrido es la casa del conductor y el final es la escuela

$$x_0 = 0 \quad y \quad x_{N+1} = N+1$$

Restricciones implícitas
No hay elementos repetidos en la tupla

$$x_i \neq x_j \quad \forall i, j \;\; 0 \le i, j \le N+1, i \neq j$$

El tiempo del recorrido debe ser menor a 60 minutos

$$\sum_{i=0}^{N} tiempo[x_i][x_i+1] < 60$$

Función objetivo
Se debe minimizar la distancia recorrida por la camioneta.

$$\min\left(\sum_{i=0}^{N} distancia[x_i][x_i+1]\right)$$

Predicados de poda
Sea $<x_0, x_1, \ldots, x_k, 0, \ldots, 0>$ la tupla en construcción y $<y_0, y_1, \ldots, y_{N+1}>$ la mejor solución hasta el momento.
Se poda una rama si la distancia recorrida de la tupla en construcción supera la distancia recorrida de la mejor solución hasta el momento

$$\sum_{i=0}^{k} distancia[x_i][x_i+1] \ge \sum_{i=0}^{N+1} distancia[y_i][y_i+1]$$

2.

```
struct Tupla {
   int* recorrido;      //tupla
   int distancia;   //distancia recorrida
}


INVOCACION:
   Tupla t, sol;
   int* escolares = new int[N+1]
   t.electivas = new int[N+2];
   sol.electivas = new int[N+2];
   t.distancia = 0;
   sol.distancia = MAX_INT;
   escolares[0] = 1;
   for(int i = 1; i < N+1; i++)
      escolares[i] = 0;
   recorridoCamioneta(tiempo, distancia, escolares, 0, t, sol, 1);
   //sol es la tupla solucion



void recorridoCamioneta(int** tiempo, int** distancia, int* escolares, int tiempoAcumulado,
Tupla t, Tupla &sol, int actual) {
   if (tiempoAcumulado < 60) {   //Restriccion implicita
      if (t.distancia < sol.distancia) {   //Funcion objetivo
         if (actual == N+1) {   //Tupla completa
            t.recorrido[actual] = N+1:   //Restriccion explicita
            t.distancia += distancia[t.recorrido[actual-1]][t.recorrido[actual]];
            tiempoAcumulado += tiempo[t.recorrido[actual-1]][t.recorrido[actual]];
            if ((tiempoAcumulado < 60) && (t.distancia < sol.distancia)) {
            //Restriccion implicita - funcion objetivo
               for(int i = 0; i < N; i++)
                  sol.recorrido[i] = t.recorrido[i];
               sol.distancia = t.distancia;
            }
         } else {
            for(int i = 1; i < N+1; i++) {
               if (escolares[i] == 0) {
                  escolares[i] = 1;
                  t.recorrido[actual] = i:
                  t.distancia += distancia[t.recorrido[actual-1]][t.recorrido[actual]];
                  tiempoAcumulado += tiempo[t.recorrido[actual-1]][t.recorrido[actual]];
                  recorridoCamioneta(tiempo, distancia, escolares, tiempoAcumulado, t, sol,
actual + 1);
                  escolares[i] = 0;
                  t.distancia -= distancia[t.recorrido[actual-1]][t.recorrido[actual]];
                  tiempoAcumulado -= tiempo[t.recorrido[actual-1]][t.recorrido[actual]];
               }
            }
         }
      }
   }
}
```