Solución Examen de Programación 3 y III (17/12/2011)

Instituto de Computación, Facultad de Ingeniería, UdelaR

Parte Teórica Obligatoria

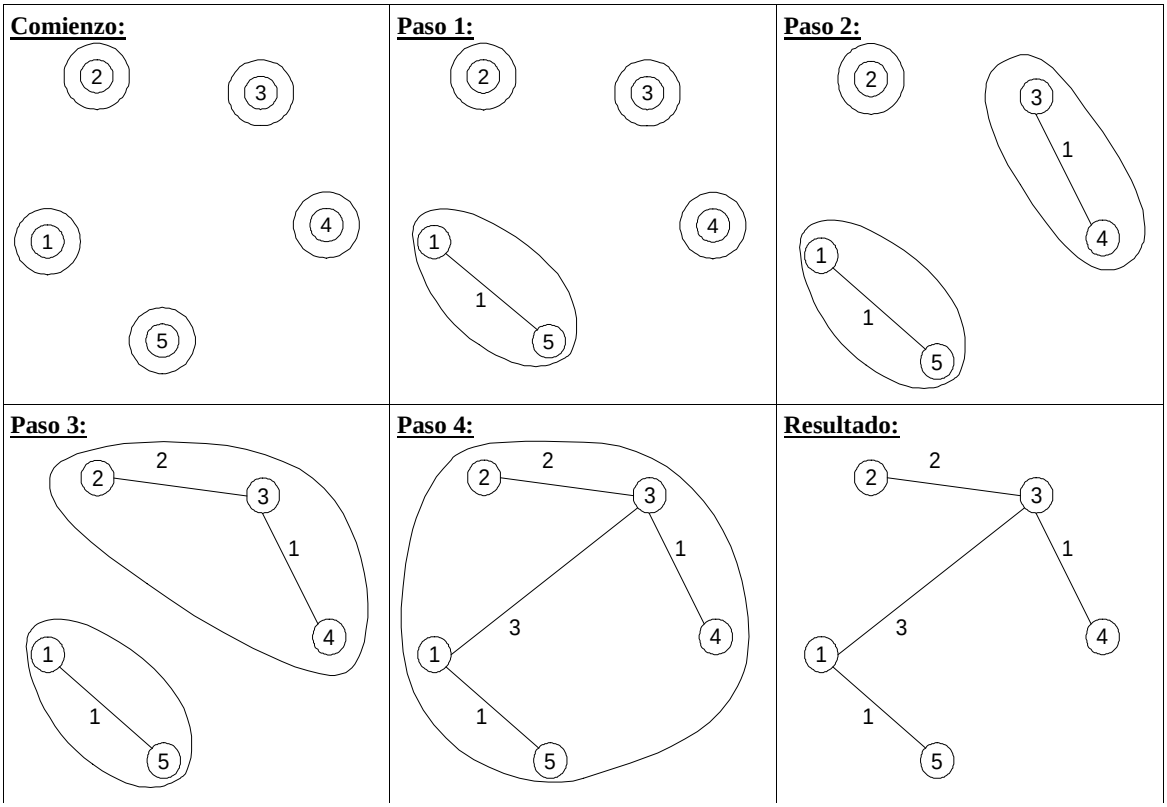
Ejercicio 1 (10 puntos)

1. Falso
2. Verdadero
3. Falso
4. Verdadero
5. Falso

Ejercicio 2 (10 puntos)

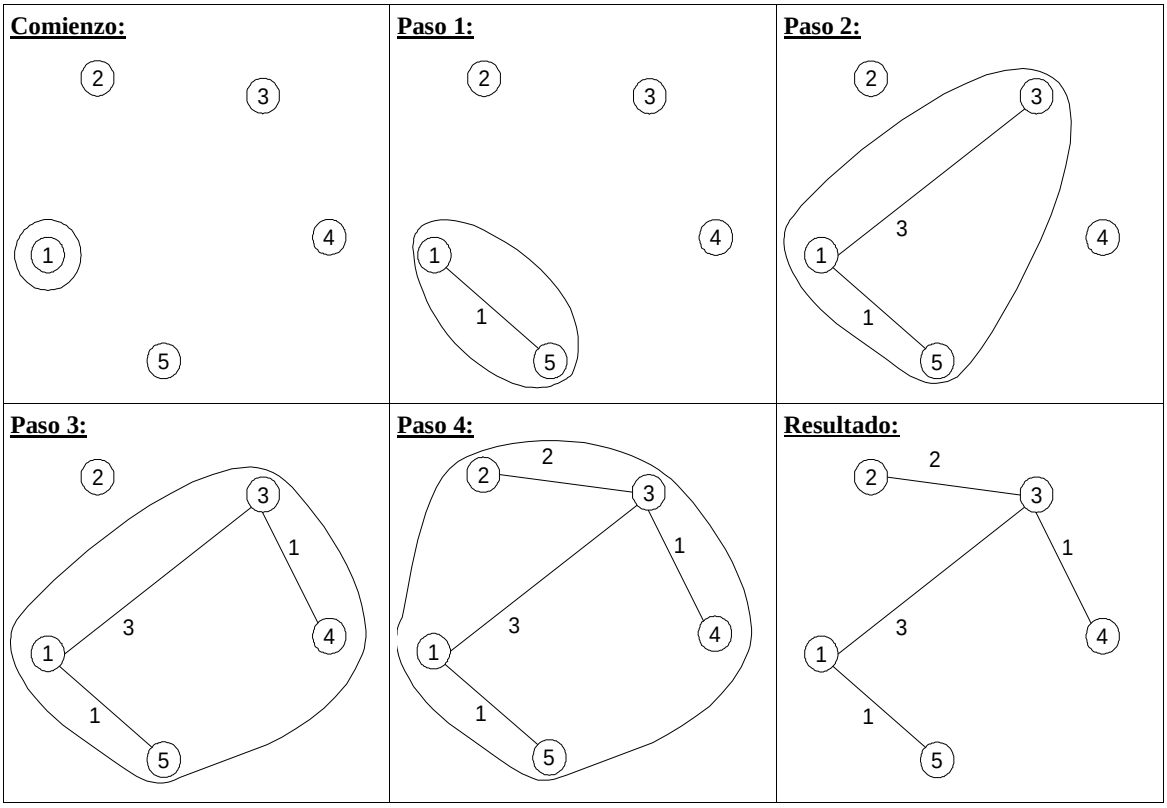
1. Kruskal

Valor	Origen	Destino	Paso
1	1	5	1
1	3	4	2
2	2	3	3
2	2	4	X
3	1	3	4
5	1	2	X
5	5	4	X



2. Prim

Valor	Origen	Destino	Paso
1	1	5	1
1	3	4	3
2	2	3	4
2	2	4	X
3	1	3	2
5	1	2	X
5	4	5	X



Ejercicio 3 (10 puntos)

1.

```
1    int sum = 0;
2    for (int i=0; i < n; i++) {
3          for (int j=0; j < i; j++) {
4                sum = sum + 1;
           }
     }
5    int copiaSum = sum;
6    for (int i=0; i < copiaSum; i++) {
7          sum = sum + 1;
     }
```

Cálculo del costo:

- 1 por la asignación de la línea **1**
- $\sum_{i=0}^{n-1}\sum_{j=0}^{i-1}1=\sum_{i=0}^{n-1} i = \frac{(n-1)n}{2}$ por la asignación en los for anidados de las líneas **3** y **4**
- $\sum_{i=0}^{copiaSum-1} 1 = \sum_{i=0}^{((n-1)n/2)-1} 1 = \frac{(n-1)n}{2}$ por la asignación en el for de la línea **6**

$$T(n) = 1+\sum_{i=0}^{n-1}\sum_{j=0}^{i-1}1+\sum_{i=0}^{(n-1)n/2}1 = 1+\sum_{i=0}^{n-1} i+\frac{(n-1)n}{2} = 1+\frac{(n-1)n}{2}+\frac{(n-1)n}{2} = 1+(n-1)n = n^2-n+1$$

2.

1. $n^2+2 \in O(n^3+1)$

   Dada la definición de orden:
   O(f) = { g: N $\to R^*$ / $\exists$ c $\in$ R^+ $\exists$ n_0 $\in$ N, $\forall$ n$\in$ N, n > n_0 , g(n) $\leq$ c*f(n)}
   Se debe de encontrar un c $\in$ R^+ y un $n_0 \in$ N a partir del cual se cumpla $n^2+2 \leq c*(n^3+1)$
   Tomando $c = 2$
   $n^2+2 \leq 2*(n^3+1)$
   $n^2+2 \leq 2n^3+2$
   $2n^3-n^2 \geq 0$
   $n^2(2n-1) \geq 0$
   Las raíces son n=0 doble y n= ½, por lo tanto tomando n_0 = ½ se cumple que $\exists$ c $\in$ R^+ $\exists$ n_0 $\in$ N / $\forall$ n$\in$ N, n > n_0 , $n^2+2 \leq$ c*(n^3+1)
   Por lo tanto $n^2+2 \in O(n^3+1)$

2. θ Es una relación de equivalencia

   Para esta parte se demostrará previamente la siguiente propiedad
   $f(n) \in O(g(n)) \Leftrightarrow g(n) \in \Omega(f(n))$
   Utilizando las definiciones de O y Ω:
   i) $O(f) = \{ g : N \to R^* / \exists c \in R^+, \exists n_0 \in N, \forall n \in N, n > n_0, g(n) \leq c * f(n)\}$
   ii) $\Omega(f) = \{ g : N \to R^* / \exists c \in R^+, \exists n_0 \in N, \forall n \in N, n > n_0, g(n) \geq c * f(n)\}$
   ($\to$)
   A partir de i) se tiene que: $\forall$ n > n_1 , $\frac{1}{c_1} * f(n) \leq g(n)$
   Si se elige $n_2 = n_1$ y $c_2 = 1/c_1$:
   $g(n) \geq c_2 * f(n), \forall n > n_2 \Rightarrow g(n) \in \Omega(f(n))$

   La demostración en el otro sentido es análoga.

   Utilizaremos también las definiciones de orden exacto:
   $\theta(f) = O(f) \cap \Omega(f)$ (1)
   $\theta(f) = \{ g : N \to R^* / \exists c_1, c_2 \in R^+, \exists n_0 \in N, \forall n \in N, n > n_0, c_1 * f(n) \geq g(n) \geq c_2 * f(n)\}$ (2)

   <u>Cumple Reflexividad</u> $f \in \theta(f)$
   Utilizando la def (2)
   $\exists c_1, c_2 \in R^+, \exists n_0 \in N, \forall n \in N, n > n_0 / c_1 * f(n) \geq f(n) \geq c_2 * f(n)$
   $f(n) = f(n) \Rightarrow f(n) \leq f(n)$ y $f(n) \geq f(n) \Rightarrow f(n) \leq 1 * f(n)$ y $f(n) \geq 1 * f(n)$

$\Rightarrow c_1 = c_2 = 1$ cumple $f(n) \le c_1 * f(n)$ y $f(n) \ge c_2 * f(n)$, a partir de un $n_0 = 0$

Cumple Simetría $f \in \theta(g) \Rightarrow g \in \theta(f)$

Si $f \in \theta(g) \Rightarrow f \in O(g(n))$ y $f \in \Omega(g(n))$ por definición de orden exacto (1)

Como $f(n) \in O(g(n)) \Rightarrow g(n) \in \Omega(f(n))$ por la propiedad demostrada.

Como $f(n) \in \Omega(g(n)) \Rightarrow g(n) \in O(f(n))$ por la propiedad demostrada.

$g \in O(f(n))$ y $g \in \Omega(f(n)) \Rightarrow g \in \theta(f)$ por la definición de orden exacto (1).

Cumple Transitividad $f \in \theta(g),\ g \in \theta(h) \Rightarrow f \in \theta(h)$

Si ($\exists c_1, c_2 \in R^+,\ \exists n_0 \in N,\ \forall n \in N,\ n > n_0\ /\ c_1 * g(n) \ge f(n) \ge c_2 * g(n)$

y $\exists c_3, c_4 \in R^+,\ \exists n_1 \in N,\ \forall n \in N,\ n > n_1\ /\ c_3 * h(n) \ge g(n) \ge c_4 * h(n)$)

$\Rightarrow \exists c_5, c_6 \in R^+,\ \exists n_2 \in N,\ \forall n \in N,\ n > n_2\ /\ c_5 * h(n) \ge f(n) \ge c_6 * h(n)$

Tenemos que $f(n) \ge c_2 * g(n)$ y $g(n) \ge c_4 * h(n) \Rightarrow f(n) \ge c_2 * g(n) \ge c_2 * (c_4 * h(n))$

$f(n) \le c_1 * g(n)$ y $g(n) \le c_3 * h(n) \Rightarrow f(n) \le c_1 * g(n) \le c_1 * (c_3 * h(n))$

Tomando $c_5 = c_1 * c_3$, $c_6 = c_2 * c_4$ y $n_2 = \max\{n_0, n_1\}$

Llegamos a $\exists c_5, c_6 \in R^+,\ \exists n_2 \in N\ /\ f(n) \le c_5 * h(n)$ y $f(n) \ge c_6 * h(n),\ \forall n > n_2$

Por lo tanto el orden exacto es una relación de equivalencia

Ejercicio 4 (10 puntos)

1.

El algoritmo ordenar, llama recursivamente 2 veces con la mitad del arreglo (n/2) y luego invoca a la función Combinar que tiene un costo de n-1.

Cálculo del costo en el peor caso:

$T_w(n) = T_w(\lfloor n/2 \rfloor) + T_w(\lceil n/2 \rceil) + (n-1)$

$T_w(1) = 0$

$n = 2^k$

$T_w(n) = 2T_w(n/2) + n - 1$

$2^1 T_w(n/2) = 2^2 T_w(n/2^2) + n - 2^1$

$2^2 T_w(n/2^2) = 2^3 T_w(n/2^3) + n - 2^2$

$\cdots$

$2^{k-1} T_w(n/2^{k-1}) = 2^k T_w(n/2^k) + n - 2^{k-1}$

$2^k T_w(n/2^k) = 2^k T_w(1) = 0$

Sumando y eliminando términos queda:

$$T_w(n) = \sum_{i=0}^{k-1} (n - 2^i)$$

$$T_w(n) = nk - \sum_{i=0}^{k-1} 2^i$$

$$T_w(n) = nk - 2^k + 1$$

como habíamos tomado $n=2^k$, $k = \log n$

$$T_w(n) = n\log n - n + 1$$

Por lo tanto $T_w(n) \in O(n \log n)$

La cota inferior a todo problema de sorting es $\mathrm{n \log n}$ por lo tanto $T_W(n) \in \Omega(n \log n)$

Por lo tanto tenemos que $T_W(n) \in \Omega(n \log n)$ y $T_W(n) \in \mathrm{O}(n \log n) \Rightarrow T_W(n) \in \theta(n \log n)$ por definición de orden exacto.

2.

El orden exacto en el caso medio es $\theta(\mathrm{n \log n})$

Esto es dado a que la cota inferior a todo problema de sorting es $\mathrm{n \log n}$ por lo tanto $T_A(n) \in \Omega(n \log n)$

Por otro lado el mejor caso debe de ser mejor o igual al peor caso por lo que $T_A(n) \in \mathrm{O}(n \log n)$

Lo que nos lleva a que $T_A(n) \in \theta(n \log n)$

3.

Definición de óptimo: Cualquier algoritmo $\mathrm{A_i}$ que cumpla $T_W(n) = F_W(n)$ comparaciones se dirá que es óptimo en el peor caso.

$F_W(n) = n \log n$ para un problema de sorting

En el peor caso el algoritmo Ordenar es óptimo ya que su costo es igual al mínimo costo en el peor caso para cualquier algoritmo de sorting.

$F_A(n) = n \log n$ para un problema de sorting

Análogamente pasa con el caso medio.

Problemas

Problema 1 (30 puntos)

Parte 1.
La cuadrícula y el laberinto se pueden modelar con grafos. En los que los nodos representan las celdas y las aristas representan: las paredes en la cuadrícula y los accesos (ausencia de paredes) en el laberinto, respectivamente.

Para construir el laberinto, se realiza un recorrido DFS sobre la cuadrícula a partir del nodo correspondiente a la celda inicial para determinar que paredes se remueven.
Dicho recorrido establece un árbol de cubrimiento (que por construcción representa el laberinto) cuyos nodos se corresponden con las celdas y las aristas del árbol se corresponden con las paredes removidas. Al ser un árbol la cantidad de paredes removidas, es decir sus aristas, es la menor posible.

Para determinar la cantidad de paredes removidas desde la celda inicial hasta cada una de las celdas, se establece un contador general que aumenta o disminuye cada vez que se preprocesa o posprocesa un nodo, respectivamente. El valor correspondiente a cada celda es el correspondiente al contador en el momento de su preprocesamiento.

Parte 2.

```
Grafo* construirLaberinto( Grafo* g, int nodo_inicial, int n, int
&cantpared)
{
     int* visitados = new int[n];
     int pared = 0;  // Cant pared removidas hasta nodo corriente
     for (int i=0; i<n; i++){
          visitados[i]=0;
          cantpared[i]=0;
     }

      Grafo* lab = crearGrafoVacio(n);
      dfs (lab, g, nodo_inicial, visitados, cantpared);

      delete[] visitados;
      return lab;
}

void dfs ( Grafo* &lab, Grafo* g, int nodoAct, int *pared, int*
visitados, int *cantpared)
{
      ListaEnteros* vecinos = obtenerVecinosGrafo( g, nodoAct);
      visitados[nodoAct] = 1;
      cantpared[nodoAct] = pared;
      pared++;
      while (!esVaciaListaEnteros(vecinos)){
              int v = ObtenerPrimeroListaEnteros(vecinos);
              if (!visitados[v]){
                       hacerVecinosGrafo(lab, nodoAct, v);
                       dfs(lab, g, v, visitados);
              }
              vecinos = restoListaEnteros(vecinos);
      }
      pared--;
}
```

Problema 2 (30 puntos)

a)

$f(i, j)$ representa el valor de la sumatoria óptima de sumas intermedias para la secuencia de elementos de *i* a *j*.

Pasos base: secuencia de un elemento:

$$f(i,i) = 0$$

Paso recursivo:

$$f(i, j) = \sum_{k=i}^{k=j} v[k] + \min\{ f(i,k) + f(k+1, j)\} \text{ con k = i..j-1 si j > i}$$

La llamada a la función para resolver el problema es $f(1, N)$.

b)

b.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 en 1 paso genérico del algoritmo.

Situación 1: la condición de la línea (2) evalúa true, por lo tanto la cantidad de operaciones básicas en esta situación es *1*, ya que se invoca el procedimiento con *i = j*.

Situación 2: la condición de la línea (2) evalúa false y la de las líneas (5) y (7) evalúan true, entonces se invoca a *funcion* con tamaño de entrada *n/2* ya que *funcion* se invoca con i = 1 y j = n. La cantidad de operaciones básicas de la línea (8) son $(n/2)^2 + n/2$. Por lo tanto la cantidad de operaciones básicas en esta situación es $T(n) = T(n/2) + (n/2)^2 + n/2$.

Situación 3: la condición de las líneas (2) y (7) evalúan false y las de la líneas (5) y (10) evalúan true. Se invoca a *funcion* dos veces, ambas con tamaño de entrada *n/2*. La cantidad de operaciones básicas de la línea (11) es igual a la de la situación 2, $(n/2)^2 + n/2$. Por lo tanto la cantidad de operaciones básicas en esta situación se $T(n) = 2T(n/2) + (n/2)^2 + n/2$.

Situación 4: la condición de las líneas (2), (7) y (10) evalúan false y la de la línea (5) evalúa true. Se invoca a *funcion* dos veces, ambas con tamaño de entrada *n/2*. La cantidad de operaciones básicas de la línea (13) son $(n)^2 + n/2$. Por lo tanto la cantidad de operaciones básicas en esta situación se $T(n) = 2T(n/2) + (n)^2 + n/2$.

Por lo tanto, la peor situación se da en la situación 4.

b.2)

$T(1) = 1$

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