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

Instituto de Computación, Facultad de Ingeniería, UdelaR

1. Este examen dura 4 horas y contiene **4** carillas. El total de puntos es 100. Para su aprobación necesita 60 puntos.
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.

Parte Teórica Obligatoria

Esta parte es eliminatoria, vale **40** puntos y usted debe obtener como mínimo el **50% de esta parte (20 puntos)**. En caso de no llegar a dicho mínimo, **NO** se corregirán los problemas. Usted podrá encontrar planteos prácticos. Los mismos deben ser resueltos justificando detalladamente la correspondencia con la base teórica que utilice en su respuesta.

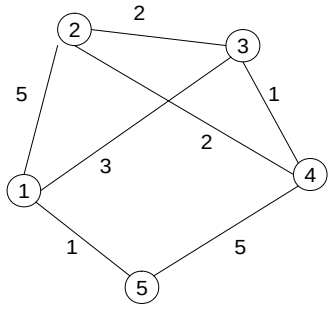
Ejercicio 1 (10 puntos)

Contestar *Verdadero* (*V*) o *Falso* (*F*) a los siguientes enunciados (no se requiere justificación). Cada respuesta correcta suma dos puntos y cada respuesta incorrecta resta dos puntos. El puntaje de este ejercicio no será inferior a cero puntos.

1. En todo grafo $G = (V,E)$ conexo y con costos positivos asociados a sus aristas, el camino de costo mínimo entre dos vértices cualesquiera de *G* está completamente contenido en alguno de los árboles de cubrimiento de costo mínimo de *G*.
2. Cuando es necesario rebalancear un árbol AVL mediante una rotación *LR*, se obtiene el mismo resultado aplicando primero una rotación *RR* (al nodo adecuado) y luego una rotación *LL* (al nodo adecuado).
3. El problema de la mochila visto en el curso cuando no se permite fraccionar los objetos, es resoluble aplicando *Greedy*.
4. En la formalización de *Backtracking*, las restricciones implícitas son reglas que indican como se relacionan entre sí las componentes de una tupla del espacio de soluciones, para que la misma sea solución.
5. Dado un grafo no dirigido $G = (V, E)$, un conjunto *C* contenido en *V* es un `clique` si y sólo si el subgrafo inducido por *C* en *G* es completo. Todos los vértices de un `clique` aparecen necesariamente consecutivos en un camino de un árbol del bosque generado por la recorrida en profundidad (*DFS*) de *G*.

Ejercicio 2 (10 puntos)

Sea G el siguiente grafo no dirigido:



1. Muestre paso a paso y de forma detallada como aplica el algoritmo de Kruskal.
2. Muestre paso a paso y de forma detallada como aplica el algoritmo de Prim a partir del vértice 1.

Ejercicio 3 (10 puntos)

1. Considerando como operación básica la asignación en la variable sum. Calcule el costo del siguiente algoritmo:

```
int sum = 0;
for (int i=0; i < n; i++) {
     for (int j=0; j < i; j++) {
          sum = sum + 1;
     }
}
int copiaSum = sum;
for (int i=0; i < copiaSum; i++) {
     sum = sum + 1;
}
```

2. Partiendo de las definiciones de: límite, θ, Ω y O:
   - Pruebe que $n^2 + 2$ pertenece a $O(n^3 + 1)$.
   - Pruebe que Θ define una relación de equivalencia R. Indicando cual es la relación y realizando la prueba.

Ejercicio 4 (10 puntos)

Considere el siguiente algoritmo de ordenación por comparaciones de elementos:

```
void Ordenar( arreglo &A, int p, int q){
  int medio;
  if (p<q) {
    medio = (p+q) / 2;
    Ordenar(A, p, medio);
    Ordenar(A, medio+1, q);
    Combinar(A, p, medio, medio+1, q);
  }
}
```

1. Para un arreglo de tamaño **n** asumiendo que la operación Combinar tiene costo `n-1` y que $n = 2^k$: Calcule el costo $T_W(n)$ e indique el ORDEN en este caso. ¿Cual es la cota inferior OMEGA para el problema de ordenación en este caso? Indique el ORDEN EXACTO del algoritmo dado en este caso.

2. Indique el ORDEN EXACTO en caso medio de este algoritmo. Justifique su respuesta.

3. ¿Este algoritmo es óptimo en el caso medio para el problema de ordenación? ¿qué puede decir del peor caso? Justifique su respuesta.

Problemas

Problema 1 (30 puntos)

Se requiere construir un laberinto a partir de una cuadrícula de celdas (similar en formación a un tablero de ajedrez). Cada celda se separa de sus celdas contiguas o del exterior mediante cuatro paredes.

El mecanismo básico de construcción del laberinto consiste en remover la pared que separa dos celdas contiguas de forma de habilitar el acceso de una celda a la otra. El proceso general de construcción consiste en que dada una celda inicial para el problema (**`celda_inicial`**), que se establece como celda de partida y accedida en ese momento, se selecciona una celda contigua no accedida (*) y se remueve la pared que las separa; luego para ésta celda contigua, que se toma como nueva celda de partida, se repite el proceso de forma tal de remover la menor cantidad de paredes y acceder a todas las celdas de la cuadrícula. En el caso de llegarse a una celda cuyas celdas contiguas ya han sido accedidas y aún queden celdas sin acceder, se continúa el proceso en la celda accedida más recientemente que aún tenga celdas contiguas sin acceder. Además, se quiere determinar para cada celda que al final del proceso termine con una única pared removida, la cantidad de paredes que se removieron para poder acceder a cada una de ellas desde **`celda_inicial.`**

Para lo que se solicita:

1. Describir como se podrían modelar el laberinto y el proceso de su construcción; y explicar porqué el modelo cumple los requisitos.

2. Escriba un algoritmo en C* que a partir de una cuadrícula y la celda **`celda_inicial,`** implemente lo descrito en el apartado **1**. Asuma que la cuadrícula contiene *n* celdas que se identifican con números de 0 a *n*–1. En el caso de utilizar TAD auxiliares, solo declare sus estructuras y operaciones.

(*) En el momento de seleccionar la celda contigua no accedida no se asume un orden entre ellas.

Problema 2 (30 puntos)

a) **(20 puntos)** Dada una secuencia de *N* números positivos, se puede intercalar entre ellos *N-1* pares de paréntesis. Existen diferentes formas de intercalar los paréntesis en la secuencia.

De todas las formas posibles de intercalar los *N-1* pares de paréntesis, se considera como óptima, aquella para la cual, la sumatoria de sumas intermedias sea mínima.

Ejemplo: dada la siguiente secuencia de 4 elementos: 4, 1, 2, 3; una forma de intercalar los paréntesis es ((4+1) + (2+3)), donde las sumas intermedias son: (4 + 1) = *5*, (2 + 3) = *5* y (5 + 5) = *10* y la sumatoria de estas es: *5* + *5* + *10* = **20**.

Otra forma de intercalar los paréntesis en la secuencia anterior es (4 + ((1 + 2) + 3)), donde las sumas intermedias son: (1 + 2) = *3*, (3 + 3) = *6* y (4 + 6) = *10* y la sumatoria de estas es: *3* + *6* + *10* = **19**.

En este ejemplo, la sumatoria de sumas intermedias mínima se da en el segundo caso.

Resolver utilizando **Programación Dinámica** el problema de determinar el valor de la sumatoria óptima de sumas intermedias para una secuencia de largo *N*. Llame a la fórmula recursiva ***f***. Explicar qué representa el/los pasos base y cada paso recursivo de la solución, así como también qué representa la función *f* indicando sus índices, y cómo debe invocarse la misma para resolver el problema.

Suponga que tiene definido el vector $v[1..N]$ donde $v[k]$ indica el valor del *k-ésimo* elemento de la secuencia.

Nota: **NO** se pide determinar cuál es la distribución de los paréntesis sino solo el valor de la sumatoria de sumas intermedias.

b) (10 puntos)

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

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:

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

b.1) **(8 puntos)** Analice las distintas alternativas en un paso genérico del algoritmo e indique la cantidad de operaciones básicas en cada alternativa. Indique en cuál/es de las situaciones anteriores se da el peor caso.

b.2) **(2 puntos)** Plantee la recurrencia que calcula la cantidad de operaciones básicas que se realizan en el peor caso definido en el punto anterior. **NO** tiene que calcular la recurrencia.