Examen de Programación 3 y III (14/12/2010)

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:

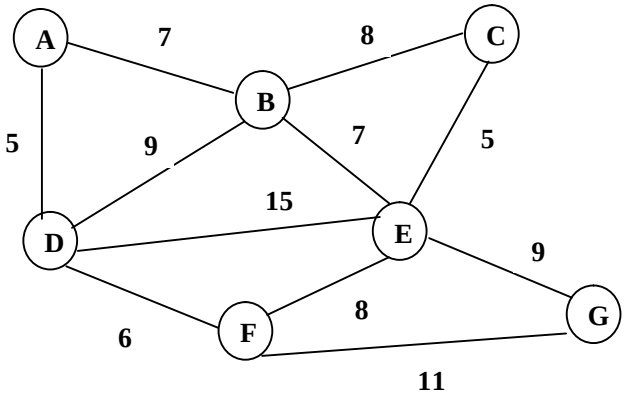
- Numerar todas las hojas e incluir en cada una el nombre y la cédula de identidad.
- Utilizar las hojas de un sólo lado y escribir con lápiz, iniciando cada ejercicio en hoja nueva.
- Completar la carátula con la cantidad de hojas entregadas, y un **índice** indicando en qué hoja se respondió cada ejercicio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

1.1 Dado el siguiente grafo:



muestre el árbol de cubrimiento de costo mínimo resultado de aplicar el algoritmo de *Prim*, partiendo del vértice **A**.

1.2 Para todo grafo $G = (V,E)$ conexo y con costos positivos asociados a sus aristas: ¿es posible encontrar un vértice x del grafo tal que la aplicación del algoritmo de *Dijkstra*, tomando x como origen, de cómo resultado un árbol de cubrimiento de costo mínimo del grafo dado? Justifique su respuesta.

Ejercicio 2 (10 puntos)

2.1 ¿El camino de costo mínimo entre 2 vértices cualesquiera de un grafo $G = (V,E)$ conexo cualquiera, genérico, y con costos asociados a sus aristas, está completamente contenido en alguno de sus árboles de cubrimiento de costo mínimo? Justifique su respuesta.

2.2 Dado un grafo no dirigido cualquiera, genérico, $G = (V,E)$, un conjunto C contenido en V es un *clique* si y sólo si para todo par de vértices de C, existe una arista que los conecta. ¿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? Justifique su respuesta.

Ejercicio 3 (10 puntos)

Dado el siguiente algoritmo que recibe como parámetro un arreglo ***a*** de tamaño ***n*** de enteros positivos, interesa calcular el costo en el mejor ($T_B(n)$) y peor ($T_W(n)$) caso, considerando simultáneamente las dos siguientes operaciones básicas:

- la comparación entre elementos de *a* con costo c_1
- la asignación en el arreglo *a* con costo c_2.

Se pide

a) Indique cuales serían los mejores y peores casos para el algoritmo dado

b) Calcule el costo del mejor caso: Plantee el costo inicialmente con c_1 y c_2, después resuelva con $c_1=2$, $c_2=4$

c) Indique en qué cambia el cálculo del costo si debe calcularlo para el peor caso. Plantee la diferencia. (no es necesario calcularlo)

Justifique su respuesta.

Sugerencia: identificar primero por separado que es lo que hace el primer bloque del algoritmo y luego el segundo bloque.

```
void algoritmo(int n, unsigned int* a)
{
    for (int i=1; i<n; i++) {
          if ( a[i-1] < a[i]  ) {

                a[i-1] = 0;
                for (int j=i; j<n; j++) {
                      a[i-1] = a[i-1] + 2*a[j];
                }
          }
    }

     for (int i = 0; i <n ; i++) {
           for (int j = i + 1; j <= n - 1; j++) {
                 if (a[i] >= a[j]) {
                      unsigned int temp = a[i];
                      a[i] = a[j];
                      a[j] = temp;
                 }
           }
     }
}
```

Ejercicio 4 (10 puntos)

Complejidad del problema de sorting en el caso medio – cota inferior en caso medio.

Considere el problema de ordenar una secuencia de ***n*** elementos por comparaciones de elementos 2 a 2 y todos los algoritmos que resuelven ***de esta manera*** el problema.

Sea el subconjunto $\boldsymbol{D'_n}$ de todas las secuencias de **n** elementos ***sin repetidos***.

Se sabe que:

1 Un algoritmo genérico *A* tiene asociado su árbol de decisión (*AD*).
2 las secuencias de D'_n son equiprobables (como entrada para el algoritmo *A*).
3 el costo del algoritmo *A* en su caso medio se denota $T_A(n)$.
4 $\boldsymbol{k_i}$ es el largo del camino de la raíz al nodo externo ***i***.
5 $\boldsymbol{p_i}$ es la probabilidad de alcanzar el nodo externo ***i***.
6 Si ***m*** es la cantidad de **nodos externos**, por el ***lema 2*** visto en el teórico la sumatoria de profundidades de los nodos externos de *AD* es mayor o igual a ***m log m.***.

Sea $F_A(n)$ una cota inferior del costo en el caso medio para un algoritmo *A* genérico que ordena una secuencia de D'_n que cumple las consideraciones anteriores.

Se podrá decir entonces que $T_A(n) \in \Omega(F_A)$ para todo algoritmo *A* entre los considerados.

Para saber cual es la complejidad del problema de sorting en su caso medio interesa encontrar la mejor cota inferior.

1. Calcule $F_A(n)$, justificando detalladamente los pasos de su razonamiento.
2. Demuestre que para el caso medio la complejidad del problema de sorting es $\Theta(n \log n)$. Sólo se pide en términos de orden exacto (Θ) y puede usar otros resultados vistos en el teórico, salvo lo que se está preguntando.

Se puede utilizar: $\sum_{i=1}^{n} \log i \geq n \log n - 1.5n$ y para simplificar: $\sum_{i=1}^{n} \log i \in \Theta(n \log n)$

Problemas

Problema 1 (30 puntos)

Un comerciante romano ha conseguido un lote de *n* animales (identificados por un valor en el rango *0..n-1*) para ser embarcados en uno de sus barcos a los efectos de ser vendidos en otra ciudad. De cada animal dispone de un único ejemplar.

El comerciante conoce el peso, en kilos, de cada animal y sabe que el barco puede transportar un peso máximo de G kilos sin hundirse.

El comerciante piensa vender cada animal a un determinado precio (denarios), y desea saber cual es la mayor cantidad de denarios que puede obtener por la venta de los animales que podría transportar en su barco sin que éste se hunda.

Asuma que se tienen definidos los siguientes vectores:

- *d[i]* indica el precio en denarios del *i*-ésimo animal con $0 \leq i \leq n-1$.
- *k[i]* indica el peso en kilos del i-ésimo animal con $0 \leq i \leq n-1$.

Se pide:

a) **(3 puntos)** Indique a través de un contraejemplo, porqué si aplica una estrategia ávida (Greedy) tomando como decisión óptima local seleccionar el animal que tiene mayor ganancia (valor en denarios) no llega a la solución óptima global.

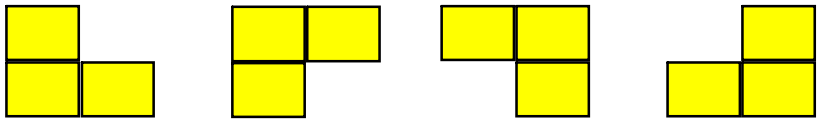
b) **(7 puntos)** ¿Es posible resolver el problema (obtener la solución óptima) aplicando una estrategia ávida (Greedy) en la cual la decisión óptima local involucre una relación entre el peso y el precio? Justifique.

c) **(15 puntos)** 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.

d) **(5 puntos)** Describa qué información adicional tendría que tener para hallar la mayor cantidad de denarios a obtener restringido a un peso G y también cual es el conjunto de animales a transportar. **No es necesario reformular la recurrencia**. Si tuviera que implementar la solución de esta parte, ¿qué estructura/s de datos adicional/es utilizaría y como las actualizaría?

Problema 2 (30 puntos)

Se ha construido una plaza en honor al Ing. William W., famoso por motivar a los estudiantes de la Fing con sus videos en la web. Se desea cubrir el área de dicha plaza con unas baldosas muy novedosas traídas de Escocia; las cuales tienen la particularidad de tener forma de "L" como muestra la siguiente figura, donde los lados (más grandes) miden 2 metros.



La plaza tiene la forma de un cuadrado cuyos lados son potencia de 2 (metros), por lo que cada metro cuadrado se ha modelado como una posición de la matriz (*Plaza[i][j]*) de dimensiones 2^n x 2^n.

En la plaza ya se ha ubicado un monumento al homenajeado que ocupa un metro cuadrado y está ubicado en la posición *Plaza[W][M] con W y M constantes* dadas.

Dado que se han comprando **solamente** las baldosas necesarias para cubrir **exactamente** el área de plaza (*sin considerar el metro cuadrado del monumento*), el director de obra le solicitó a usted, que diseñe un algoritmo que resuelva como han de ubicarse las baldosas para que los albañiles puedan colocar las baldosas. Bajo ningún concepto las baldosas podrán superponerse, ni dividirse ni tampoco ubicarse parcialmente fuera del área de la plaza.

Se pide:

1) Cuál es la **cantidad de baldosas** compradas para cubrir el área de la plaza si ésta es de NxN metros? (*no se considera el espacio destinado al monumento*)
2) Indique los pasos principales de la función ***cubrirPlaza(Posicion inicial, int dim, Posicion cubierta)*** *que* **utilizando Divide & Conquer** permite obtener el diagrama de las baldosas ubicadas de manera tal que cubran la plaza.
3) **Implemente en C*** dicha función. ***(Solo** se considerará esta parte **si 2)** se realiza **satisfactoriamente**).*

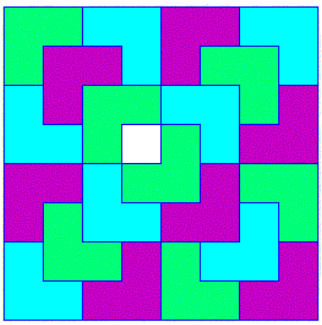
Importante: para registrar las ubicaciones **debe** utilizar las siguientes funciones y tipo:

1) Tipo Posición:

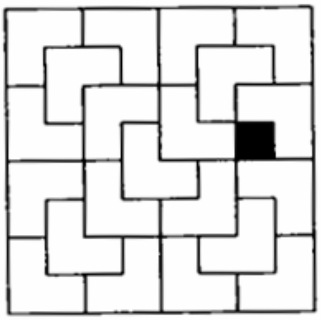
```
struct Posicion{
      int fila
      int col
}
```

2) *Posición Posición(i,j)*, crea una posición de coordenadas (*i, j*).
3) *void ubicarBaldosa(Posición a, Posición b, Posición c)*, que dadas 3 posiciones registra la ubicación de una baldosa de modo de cubrir tales posiciones.
4) *boolean estaCubierta(i,j),* indica si la posición *Plaza[i][j]* ya ha sido cubierta o no.

Ejemplos de diagramas de plazas de dimensión 8x8:



Monumento en Plaza[3][3]



Monumento en *Plaza[3][6]*