Examen de Programación 3 y III (19/07/2014)

Instituto de Computación, Facultad de Ingeniería, UdelaR

1. Este examen dura 4 horas y contiene **4** carillas. El total de puntos es 100 y se requieren 60 para su aprobación.
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*, el uso de *cout* y *cin y el tipo bool*.
3. NO se puede utilizar ningún tipo de material de consulta.
4. No se contestarán 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 solo lado y escribir con lápiz, iniciando cada ejercicio en hoja nueva.
- Poner en la primera hoja la cantidad de hojas entregadas, y entregar el **índice** indicando en qué hoja se respondió cada problema.

Parte Obligatoria

Esta parte es eliminatoria, para la aprobación del examen debe obtenerse un mínimo del **50% de esta parte (20 puntos)**. En caso de no llegar a dicho mínimo, **NO** se corregirán los problemas.

Ejercicio 1 (10 puntos)

a) Enuncie las definiciones de $\Omega(f)$ y $\Theta(f)$.
b) Demuestre que: si $f \in \Theta(g) \Rightarrow g \in \Theta(f)$.

Ejercicio 2 (10 puntos)

a) Implemente el algoritmo **merge sort** respetando el siguiente encabezado:

void merge_sort(int* v, int n);

Puede asumir implementada la siguiente función:

void merge(int* v, int a_start, int a_end, int b_start, int b_end);

Realiza el merge de las porciones de v delimitada por los índices asumiendo que las porciones se encuentran ordenadas.

Nota: No debe utilizar vectores auxiliares (solamente el vector de entrada v)

b) Explique brevemente qué estrategia utiliza dentro de las vistas en el curso y como aplica en este caso.

Ejercicio 3 (10 puntos)

Una empresa de importación que consolida compras de clientes en el exterior desea optimizar el uso de sus contenedores. Para definir cómo cargar un contenedor de capacidad C se debe maximizar la cantidad de compras que lleva buscando satisfacer la mayor cantidad de clientes. A su vez se considera una restricción impuesta por la aseguradora que establece que el valor total de las compras que transporta un contenedor puede ser a lo sumo M pesos ($).

Cuenta con un vector compras[] de largo N, donde cada elemento compras[i] representa una compra (indivisible) y contiene los siguientes campos:

- tamaño: espacio que ocupa la compra (en unidades de capacidad) (>0).
- valor: valor de la compra en $ (>0)

a) Defina la función recursiva *maxCompras* que retorna la cantidad máxima de compras incluidas en un contendor respetando las restricciones. Indique brevemente el significado de los parámetros y pasos definidos.

Ejercicio 4 (10 puntos)

Para la realidad planteada en el ejercicio 3:

a) Defina en seudocódigo explicitando el criterio de decisión un algoritmo que utiliza **greedy** para la selección de compras maximizando la cantidad total.
b) Justifique si esta estrategia es válida (obtiene solución óptima) para resolver el problema planteado particularmente.

Problemas

Problema 1 (30 puntos)

Un grupo de amigos decidió ahorrar durante cuatro años para poder presenciar el "Segundo Maracanazo". Su planificación inicial era ir a ver sólo los partidos de la celeste hasta la final. Como su itinerario se vio frustrado luego del partido de octavos de final (Uru-Col), decidieron reorganizar su recorrida mundialista con el dinero que les quedaba.

El grupo decidió calificar cada partido aún no disputado según sus preferencias. Calificaron los 6 partidos de octavos que restaban, los 4 de cuartos de final, las 2 semi-finales, el partido por el tercer puesto y la final. Cuanta más calificación se otorgaba al partido, mayor era la preferencia.

Las entradas a los partidos variaban en costo. Cada partido se jugaba en una de las sedes del mundial. Para poder asistir a un determinado partido el grupo debía viajar a la sede donde se disputaba. Para ello contaban con dos medios de transporte: avión y ómnibus. Los medios de transporte variaban en precio y duración. La estadía se asumió gratuita en cualquiera de las sedes ya que otros amigos los alojaban.

Plantee el problema de seleccionar los partidos a asistir en términos de **Backtracking,** definiendo formalmente y en lenguaje natural la forma de la tupla, restricciones explícitas, implícitas, función objetivo y predicados de poda que apliquen. Considere que buscaron maximizar el total de las preferencias de los partidos asistidos con el dinero que tenían.

Cuenta con la siguiente información y funciones para formalizar el problema:

- Vector de elementos Partido, *partidos[i]* con i entero en [1..15], donde i <> pUruCol identifica a un partido NO disputado .
  $pUruCol$ - Identificador del partido de Uruguay-Colombia (de donde parte el grupo de amigos).
  Ningún partido se juega en simultáneo con otro. Los identificadores (i) de los partidos NO determinan un orden cronológico.

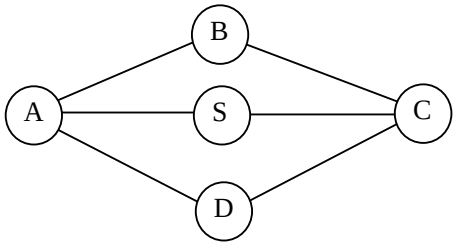
- Cada elemento Partido p contiene los siguientes campos:
  - $p.preferencia$ - Indica la preferencia del grupo para el partido p.
  - $p.entrada$ - Indica el precio de la entrada para el partido p.
  - $p.sede$ - Indica la sede del partido p.

- D - Dinero con el que cuenta el grupo de amigos después del partido (pUruCol).
- $DiferenciaPartidas[p_A][p_B]$ - Indica la cantidad de horas entre el final del partido p_A y el comienzo del partido p_B. Si el valor es negativo, p_A se disputó después que p_B.
- $DuracionAvion[s_1][s_2]$- Indica la cantidad de horas de vuelo entre las s_1 y s_2.
- $CostoAvion[s_1][s_2]$ - Indica el precio del vuelo entre las sedes s_1 y s_2.
- $DuracionOmnibus[s_1][s_2]$ - Indica la cantidad de horas del recorrido en ómnibus entre las sedes s_1 y s_2.
- $CostoOmnibus[s_1][s_2]$ - Indica el precio del ómnibus entre las sedes s_1 y s_2.

Problema 2 (30 puntos)

Sea $G = (V,E)$ un grafo conexo no dirigido.

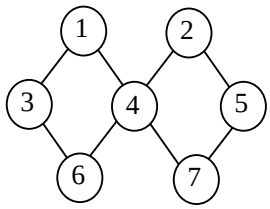
a) Sea $d_G\ (u,v)$ la distancia entre los vértices u y v en el siguiente grafo G.



Dibuje un árbol de cubrimiento de *G*, que NO pueda ser el resultado de *BFS (G,S)*, en el cual para cada vértice *v*, la longitud del camino desde *S* a *v* es igual a $d_G\ (S,v)$.

b) Sea *u* un vértice de V. Se quiere obtener para cada vértice *v* de *V* la **cantidad** de caminos de longitud mínima entre *u* y *v*. La cantidad de caminos de longitud mínima desde *u* hasta *u* se define igual a 1.

Como ejemplo, se muestra un grafo y las cantidades de caminos desde el vértice **3**:



vértice	**1**	**2**	**3**	**4**	**5**	**6**	**7**
cantidades	1	2	1	2	4	1	2

Se pide: Implemente en C* el procedimiento **caminosMinimos**:

```
/* Dado el grafo 'G', de 'n' vértices identificados de 1 a 'n', y un vértice 'u', 1 <=
u <= n, devuelve un array con la cantidad de caminos de longitud mínima desde 'u'
hasta cada vértice de 'G'.
El algoritmo debe pasar por cada arista una única vez. */

      int * caminosMinimos (Grafo G, int n, int u);
```

Asuma dados todos los TADs vistos hasta el curso.
En particular, *ListaAdyacentes (G, v)* devuelve la lista de vértices de *G* adyacentes al vértice *v*.