Examen de Programación 3 y III (14/07/2012)

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. Puede usarse lo dictado en el curso sin demostrarlo, indicando claramente lo que se está usando.
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 un **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. 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)

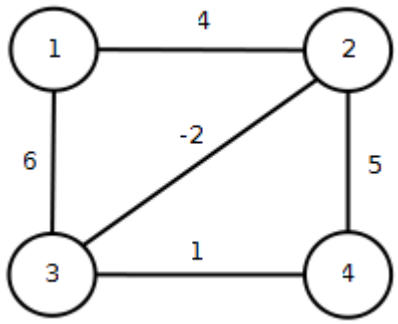
Dado el siguiente algoritmo que dado un arreglo *A*, de tamaño *n*, retorna *true* si **más** de la mitad de los elementos del arreglo son menores que el parámetro *x* y *false* en caso contrario y considerando como operación básica ***solo*** la comparación (*A[i] < x*) con costo ***1***:

```
int masDeLaMitadMenores(int * A, int n, int x){
   int i = 0;
   int cant = 0;
   while ((i<n) && (cant <= n/2)){
      if (A[i]<x){
         cant++;
      }
      i++;
   }
   return (cant>n/2)
}
```

1. Indique cuál es el mejor y peor caso.
2. Calcule el costo en el peor caso $(T_w(n))$ y el costo en el mejor caso $(T_b(n))$.

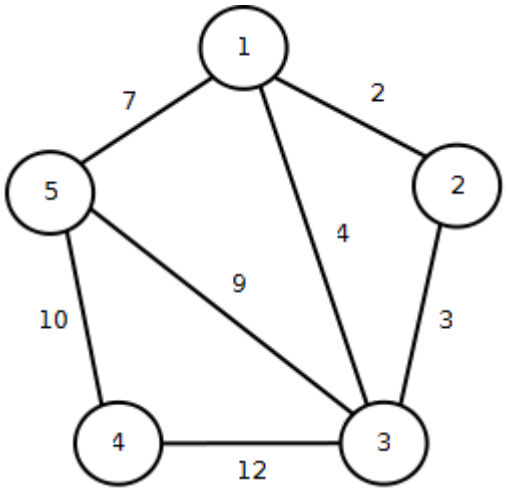
Ejercicio 2 (10 puntos)

a) ¿Qué problema resuelve el algoritmo de *Floyd* y qué técnica vista en el curso utiliza?
b) Muestre paso a paso y en forma detallada la aplicación del algoritmo de *Floyd* al siguiente grafo.



Ejercicio 3 (10 puntos)

a) ¿Qué problema resuelven el algoritmo de *Prim* y el algoritmo de *Kruskal*?
b) Muestre paso a paso y en forma detallada la aplicación del algoritmo de *Kruskal* al siguiente grafo.



Ejercicio 4 (10 puntos)

a) Escriba el pseudocódigo del algoritmo *Insertion Sort* visto en el curso.
b) Escriba el pseudocódigo del algoritmo *Selection Sort* visto en el curso
c) Se desea ordenar en forma ascendente un arreglo de enteros con los valores **2, 3, 5, 1, 4**. Considerando los algoritmos de las partes a) y b) (*Insertion Sort* y *Selection Sort*), determine cual algoritmo emplea menos ***esfuerzo*** en ordenarlo, considerando que la comparación entre elementos del arreglo implica ***una unidad de esfuerzo*** y el intercambio de elementos en el arreglo implica ***dos unidades de esfuerzo*** (solo si son elementos diferentes). Justifique su respuesta mostrando paso a paso la ejecución de los algoritmos, especificando en cada paso qué elementos se comparan, cuáles intercambios se realizan y cómo queda el arreglo al final de cada paso.

Problemas

Problema 1 (30 puntos)

Un vendedor está encargado de vender una colección de ***n*** objetos, para lo cual contacta a ***m*** posibles compradores y obtiene las ofertas que ellos hacen, manteniéndolas en la tabla *OFERTAS [i][j]*, $1 \leq i \leq m$, $1 \leq j \leq n$; que indica la cantidad que el comprador ***i*** pagaría por el objeto ***j***. Cada comprador tiene un límite, *LIMITE [i]*, $1 \leq i \leq m$; que indica que el total de sus compras no puede superar ese límite. Aunque el límite es mayor o igual a cada una de sus ofertas, es posible que sea inferior a la suma de todas ellas.
El actual dueño de los objetos establece para cada objeto un precio base, *BASE [j]*; $1 \leq j \leq n$; ningún objeto puede venderse a un precio inferior a su precio base. Puede haber algún comprador cuya oferta por algún objeto sea inferior al precio base, pero se puede asumir que para cada objeto hay al menos un comprador que está dispuesto a pagar su precio base o más.
El vendedor quiere saber como puede obtener la recaudación máxima cumpliendo con la condición de que todos los objetos sean vendidos.

Notas:

a) Cada oferta se acepta completamente o no se acepta. Es decir, si *OFERTAS [i][j]* > *BASE [j]*, entonces no puede ocurrir que el precio de venta ***v*** cumpla: *OFERTAS [i][j]* > ***v*** ≥ *BASE [j]*.
b) Considere definido *MAYOR [j]*, $1 \leq j \leq n$; que contiene, para cada objeto *j*, la mayor de las ofertas que algún comprador hizo por el objeto.

Se pide:

a) Para cada una de las siguientes situaciones explique si se puede asegurar que el problema tiene solución. Si la respuesta es negativa puede mostrarlo con un ejemplo.
   1) Las condiciones planteadas en el problema.
   2) Asumiendo que el límite de cada comprador es mayor o igual a la suma de todas sus ofertas.

b) ¿Es posible resolver el problema mediante un algoritmo ávido? En caso afirmativo escriba el pseudocódigo. Si no es posible muestre un contraejemplo.

c) Formalizar el problema en términos de ***Backtracking*** indicando forma de la tupla, restricciones explícitas e implícitas y, si corresponde, función objetivo y predicados de poda.

d) Suponga que en la construcción de una tupla ***t*** se han asignado los primeros ***h*** objetos, con $1 \leq h < n$, y que ***s*** es la mejor solución obtenida hasta el momento. Formalice los siguientes enunciados y explique si son predicados de poda o no:
   1) El total de las ventas ya asignadas en la tupla ***t*** es menor que la suma de las ventas de los primeros ***h*** objetos en la tupla ***s***.
   2) Existe un comprador, identificado por ***k***, $1 \leq k \leq m$, para el cual se cumple que la suma de las ventas ya adjudicadas a él más el precio base del objeto $h + 1$ supera el límite del comprador ***k***.

Problema 2 (30 puntos)

Se desea simular el contagio de una población de individuos provocada por una mutación del virus de la varicela, que al igual que el virus de la varicela común, contagia por contacto directo. El virus en un individuo contagiado (huésped) puede tener el siguiente comportamiento:

- **latente**: el huésped es portador de la enfermedad y puede contagiar a otros, pero no padece la enfermedad
- **activo**: el huésped es portador de la enfermedad y puede contagiar a otros y padece la enfermedad.

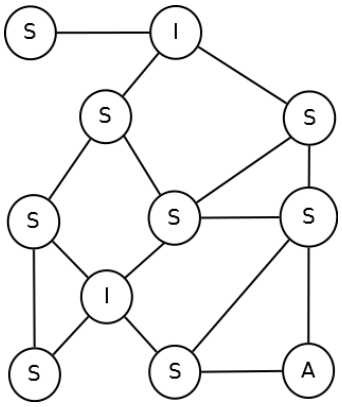
Independientemente del comportamiento del virus en un huésped, ese huésped será un nuevo foco de contagio para otros individuos. Cuando un huésped entra en contacto con un individuo sano, esta mutación contagia al individuo sano de manera selectiva, basándose en como está presente en el huésped y alternando entre los posibles comportamientos. Por ejemplo si el virus se encontraba activo en A y este contagia a B, B lo tendrá en estado latente. Si B contagia a C, C lo tendrá en estado activo y así sucesivamente.

Un individuo solo es contagiado por el virus una única vez, aun habiendo estado en contacto directo con distintos huéspedes.
Dentro de la población pueden existir individuos **inmunes**, o sea individuos que no son contagiados por el virus, y por lo tanto no contagian a otros individuos ni padecen la enfermedad.
El huésped contagia a todos los individuos sanos, no inmunes, con los que está en contacto directo simultáneamente.

El comportamiento de esta mutación dificulta la identificación de los focos de contagio dentro de una población, lo que evita su contención a través del aislamiento de los focos activos; por lo que para la etapa inicial del simulador, solamente se busca reproducir el comportamiento del virus a partir de un único foco activo en una población dada.

Se pide:

a) A partir de la siguiente población, donde se muestra el estado de cada individuo (**S**ano, **I**nmune, **L**atente o **A**ctivo) y los contactos entre los individuos de la misma, simule el comportamiento del virus si se considera que el foco activo es el individuo marcado con **A**.



b) Implemente en C* la operación *void simMutacionVaricela(Grafo población, int origen)* que actualiza los valores del campo estado de un grafo de nodos Individuo (*población*) representando la simulación del impacto que tiene un foco activo en *origen* (el estado del nodo origen es ACTIVO)

Asuma la existencia de:

- TAD Lista y Cola de enteros.
- ```
  struct Individuo{
        int id;
        int estado;
        ListaEnteros* adyacentes;
  }
  ```
- constantes *int* LATENTE, ACTIVO, SANO, INMUNE definidas.
- *bool esInmune(Grafo poblacion, int id),* indica si el individuo *id* de *población* es inmune.
- *void actualizarEstado(Grafo población, int id, int estado)*
- TAD Grafo de *Individuos*, análogo a Grafo de Enteros con *get* y *set* para cada uno de miembros de *Individuo*.
  ```