Examen de Programación 3 y III (12/07/2010)

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.
- 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)

Dado un árbol AVL de *n* nodos, representado mediante el estructurado *NodeAVL*, escribir un algoritmo que calcule la altura del árbol cuyo costo $T_w(n) \in \theta(\log n)$. **Justifique por qué el algoritmo propuesto tiene ese orden.**

El encabezado de la función a implementar debe ser: `int altura (NodeAVL* arbol);`

```
struct NodeAVL
{
          int clave;
          int fb;
          struct NodeAVL * izq;
          struct NodeAVL * der;
};
```

Ejercicio 2 (10 puntos)

Dadas las siguientes funciones arbitrarias $f: N \to R^*, g: N \to R^*$ y $h: N \to R^*$, partiendo de las definiciones de límite θ, Ω y O:

2.1 Pruebe que si el $\lim_{n \to +\infty} \frac{f(n)}{g(n)} = 0$ entonces $f(n) \in O(g(n))$ y $g(n) \notin O(f(n))$

2.2 Indique si es verdadera o falsa la siguiente afirmación: Si $h(n) \in \Theta(n^2)$ entonces $2^{h(n)} \in \Theta(2^{n^2})$. En caso de ser verdadero o falso, demuestre su afirmación.

Ejercicio 3 (10 puntos)

3.1 Proporcione el pseudocódigo del *Selection Sort*.

3.2 Dibuje el árbol de decisión del Selection Sort de la parte anterior para una entrada de tamaño 3.

Ejercicio 4 (10 puntos)

4.1 Defina punto de articulación para un grafo no dirigido.

4.2 Dado un grafo *G* y un vértice *v*, implementar en C* el algoritmo:

```
bool esPuntoArticulacion(Grafo G, int v)
```

que determina si ***v*** es punto de articulación de *G* **sin recorrer el grafo G más de una vez.**

- **Nota:** asuma que tiene definido el TAD Lista y TAD Grafo. La especificación de los mismos se encuentra definida al final del Problema 1.

Problemas

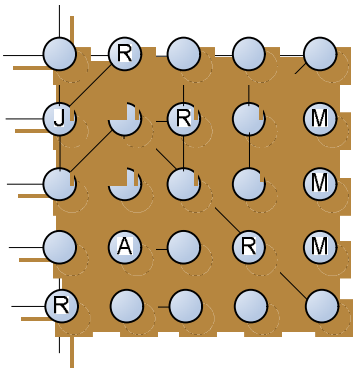
Problema 1 (30 puntos)

Para el mundial de fútbol robótico le han solicitado a usted que implemente una de las estrategias de ataque (*ataqueProfundo*) a ser invocada cuando los robots se encuentran con la pelota y cerca del arco rival. Dicha técnica constará de la búsqueda de un camino hacia el área rival de forma de concretar.

Para ello, cada vez que es invocada, se obtiene un mapa de la cancha modelado como un grafo (*G*) donde cada nodo representa una posición y se clasifica entre los siguientes tipos (*TipoNodo*):

LIBRE: indica que la posición de la cancha esta libre.
AMIGO: indica la presencia de un compañero de equipo en esa posición.
JUGADOR: indica la posición del robot
RIVAL: indica la presencia de un rival en esa posición.
META: indica una posición del área rival.

La contigüidad de las posiciones define la relación de adyacencia entre los nodos de G, análogamente a una matriz, como se muestra en el siguiente ejemplo:



Cada posición (Nodo) es identificada a través de un identificador (*int id*).

Bajo esta modalidad el robot solo buscará avanzar a través de nodos **LIBRES** y **solamente en direcciones hacia el arco rival (adelante)**, por lo que las posiciones adyacentes posibles de avance para (*J*) en el ejemplo anterior serían solo 2.

Además, la estrategia **deberá** considerar la cantidad de veces que el robot es *marcado* por algún RIVAL. Se considera como *marcado*, cada vez que el robot tenga al menos un rival en una posición **adyacente**. La estrategia deberá descartar caminos en los que el robot sea marcado ***K*** veces **SIN** haber llegado al área (META) dado el riesgo que implica. Considerando el ejemplo anterior, con K=4 no es posible determinar un camino, mientras que con K=5 sí lo es.

Se pide:

1) **Implemente** la estrategia `Lista ataqueProfundo(int posicion, Grafo cancha)`, que recibe la *posición* del robot y el *Grafo* que mapea el estado de la cancha; y retorna la Lista con las posiciones (id) del camino obtenido en caso que sea posible, ó en caso contrario, la lista vacía, lo cual

representará que no es posible llegar al área.

2) Para los casos en que no se determina un camino, el sistema de control del robot buscará hacer un pase, por lo que invoca a la estrategia *int paseInmediato(int posición, Grafo cancha)*, que retorna la posición del AMIGO mas cercano, donde ante igualdad de distancia (entre nodos) se deberá tener **preferencia** por las posiciones que se encuentran **adelante** (ignora a los rivales). **Implemente** dicha estrategia.

Asuma los siguientes Tipos y Funciones dados:

- TAD Lista y TAD Grafo
- `TipoNodo getTipoNodo(Grafo g, int id); // Dada una posición retorna su TipoNodo.`
- `Lista posAdelante(Grafo g, int posición); // Retorna los` ***`id`*** `de los auxnodos adelante del indicado por` ***`posición`***`.`

TAD Grafo

```
Grafo construirGrafo (int cantNodos);
// Construye un grafo con 'cantNodos' nodos.
void agregarAristaGrafo (Grafo& grafo, int nodoOrigen, int nodoDestino, int costo);
// Precondicion: 0 <= nodoOrigen, nodoDestino < cantNodosGrafo (grafo)
//  Se agrega una arista a 'grafo'.
int costoAristaGrafo (Grafo & grafo, int nodoOrigen, int nodoDestino);
// Precondicion: existe la arista [nodoOrigen, nodoDestino] en el grafo.
//  Devuelve el costo de la arista [nodoOrigen, nodoDestino].
bool existeAristaGrafo (Grafo grafo, int nodoOrigen, int nodoDestino);
// Precondicion: 0 <= nodoOrigen, nodoDestino < cantNodosGrafo (grafo)
// Retorna true sii existe una arista dirigida desde 'nodoOrigen' hacia
// 'nodoDestino' en el grafo.
int cantNodosGrafo (Grafo grafo);
// Retorna la cantidad de nodos del grafo.
Lista adyacentesNodoGrafo (Grafo grafo, int nodo);
// Precondicion: 0 <= nodo < cantNodosGrafo (grafo)
// Retorna la lista id de nodos adyacentes a 'nodo' del grafo.
void destruirGrafo (Grafo & grafo);
// Libera la memoria asociada a la estructura.
```

TAD Lista de Enteros

```
void listaInsertar (Lista l, int n);
// Agrega el entero n al principio l.
void listaInsertarUltimo (Lista l, int n);
// Agrega el entero n al final de l.
bool listaEstaVacia (Lista l);
// Retorna true sii l está vacía.
bool listaPerteneceElem (Lista l, int n);
// Retorna true sii el elemento n pertenece a l.
void listaSacar (Lista l);
// Saca el primer elemento de l.
// Precondicion: !ListaEstaVacia (l).
void listaSacarUltimo (Lista l);
// Saca el ultimo elemento de l.
// Precondicion: !ListaEstaVacia (l).
int listaPrimero (Lista l);
// Retorna el primer elemento de l.
// Precondicion: !ListaEstaVacia (l).
int listaUltimo(Lista l);
// Retorna el ultimo elemento de l.
// Precondicion: !ListaEstaVacia (l).
void listaDestruir (Lista l);
// Libera la memoria de l.
```

Problema 2 (30 puntos)

Una casa de música ofrece como servicio un sitio web en el cual permite la descarga de ringtones. La casa de música ofrece *n* ringtones diferentes, donde de cada uno de estos se conoce: su precio, su ranking y su tamaño de descarga en bytes.

Un adolescente quiere descargar un conjunto de ringtones tal que, el tamaño de descarga total sea el más bajo posible, el precio total de dicha elección no supere su presupuesto P, y a su vez, el total del valor del ranking de los ringtones seleccionados, sea como mínimo R. Téngase en cuenta que no se pueden repetir ringtones.

Asuma que se tienen definidas las siguientes funciones:

- *p(k)* indica el precio del *k*-ésimo ringtone con $1 \leq k \leq n$.
- *r(k)* indica el ranking del *k*-ésimo ringtone con $1 \leq k \leq n$.
- *d(k)* indica el tamaño de descarga en bytes del *k*-ésimo ringtone con $1 \leq k \leq n$.

Se pide:

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.

Sugerencia: tenga en cuenta para definir el/los caso/s base el hecho que no hayan mas ringtones disponibles, discriminando según el ranking sea mayor ó menor igual a cero.