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

Instituto de Computación, Facultad de Ingeniería, UdelaR

Parte Teórica Obligatoria

Ejercicio 1 (10 puntos)

```
int altura(NodeAVL* arbol)  {
      if (arbol == NULL)
            return -1;
      else {
            if ( arbol->fb == 1)
                  return 1 + altura(arbol->izq);
            else
                  return 1 + altura(arbol->der);
      }
}
```

La función altura es invocada h + 2 veces siendo h la altura del árbol. Por ser un AVL, como se vio en el curso se cumple que la altura está acotada por $\log(n+1) \leq h \leq 1.44\log(n+2)$.

Aplicando la regla del límite se verifica que $\log(n+1) \in \Theta(\log(n))$ y $1.44\log(n+2) \in \Theta(\log(n))$.

Concluyéndose que el costo de altura es de orden $\Theta(\log(n))$.

Ejercicio 2 (10 puntos)

2.1)

*1) Desarrollo de definición de $f(n) \in O(g(n))$:

$(\exists K \in R^+)(\exists n_0 \in N)(\forall n \in N)\left(n \geq n_0 \rightarrow f(n) \leq K * g(n)\right)$

*2) Desarrollo de definición de $\lim_{n\to+\infty} \frac{f(n)}{g(n)} = 0$

Sii (definición límite de sucesiones)

$(\forall \varepsilon \in R^+)(\exists n_1 \in N)(\forall n \in N)\left(n \geq n_1 \rightarrow \left| \frac{f(n)}{g(n)} - 0 \right| < \varepsilon \right)$

Partiendo de *2), operando y teniendo en cuenta que f y g son funciones positivas se llega a que

$(\forall \varepsilon \in R^+)(\exists n_1 \in N)(\forall n \in N)\left(n \geq n_1 \rightarrow f(n) < \varepsilon * g(n)\right)$

Luego, se elige un ε cualquiera. Esto verifica la definición $f(n) \in O(g(n))$ escrito en *1).

Resta probar que $g(n) \notin O(f(n))$

Lo anterior equivale a decir que no se cumple la afirmación $(\exists K \in R^+)(\exists n_0 \in N)(\forall n \in N)\left(n \geq n_0 \rightarrow g(n) \leq K * f(n)\right)$

Sii (equivalencia lógica)

No existe $K \in R^+$ tal que $(\exists n_0 \in N)(\forall n \in N)\left(n \geq n_0 \rightarrow g(n) \leq K * f(n)\right)$

Para probarlo se supone por absurdo que existe $K \in R^+$ tal que $(\exists n_0 \in N)(\forall n \in N)\big(n \geq n_0 \rightarrow g(n) \leq K * f(n)\big)$.

Por *2)

$$(\forall \varepsilon \in R^+)(\exists n_1 \in N)(\forall n \in N)\left(n \geq n_1 \rightarrow \frac{1}{\varepsilon} f(n) < g(n) \right)$$

Si tomo el $n_3 = \max(n_0, n_1)$ y $\varepsilon = \frac{1}{K}$ entonces se cumple

$(\forall n \in N)\big(n \geq n_3 \rightarrow K * f(n) < g(n)\big)$ y $(\forall n \in N)\big(n \geq n_3 \rightarrow g(n) \leq K * f(n)\big)$

Lo cual es una contradicción.

2.2)

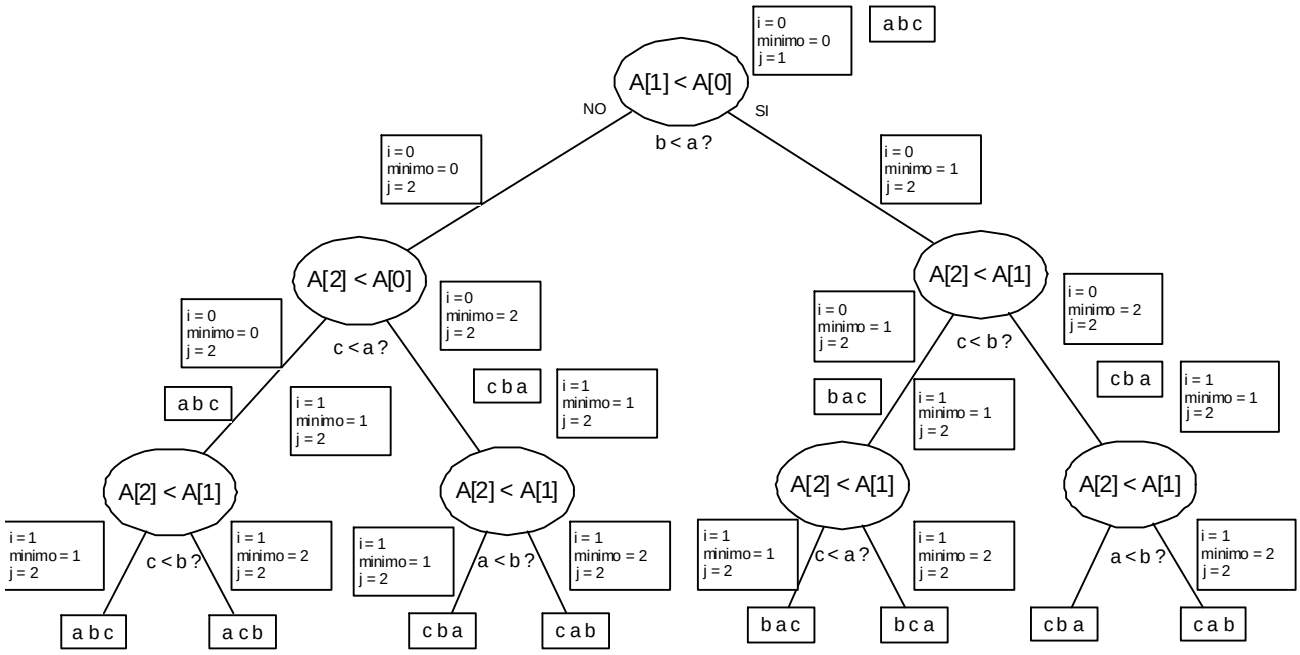
$2n^2 \in \Theta(n^2)$ y $2^{2n^2} \notin \Theta(2^{n^2})$ ya que $\lim_{n \to +\infty} \frac{2^{n^2}}{2^{2n^2}} = \lim_{n \to +\infty} 2^{-n^2} = 0$.

Ejercicio 3 (10 puntos)

3.1)

```
para i=0 hasta n-2
    minimo = i;
    para j=i+1 hasta n-1
        si lista[j] < lista[minimo] entonces
            minimo = j
        fin si
    fin para
    intercambiar(lista[i], lista[minimo])
fin para
```

3.2)



Ejercicio 4 (10 puntos)

4.1) Un vértice de un grafo no dirigido es un punto de articulación, si el subgrafo que se obtiene, borrándolo junto a todos sus aristas incidentes contiene más componentes conexas que el original.

4.2) Para verificar si un vértice *v* es PA, se obtiene un árbol de cubrimiento (DFS) con raíz *v*, si *v* tiene más de un descendiente entonces *v* es PA.

```
bool esPuntoArticulacion(Grafo G, int v) {
      int n = cantNodosGrafo(G);
      bool* marcados = new bool[n];
      for (int i = 0; i < n; i++)
            marcados[i] = false;
      return esPuntoArticulacionDFS(G, v, marcados);
}

bool esPuntoArticulacionDFS(Grafo G, int v, bool* marcados) {
      marcados[v] = true;
      Lista adyacentes = adyacentesNodoGrafo(G, v);
      int cantAdyacentes = 0;
      while (!listaEstaVacia(adyacentes)) {
            int u = listaPrimero(adyacentes);
            if (!marcados[u]) {
                  cantAdyacentes++;
                  esPuntoArticulacionDFS(G, u, marcados);
            }
            listaSacar(adyacentes);
      }
      if (cantAdyacentes > 1)
            return true;
      return false;
}
```

Problemas

Problema 1 (30 puntos)

```
Lista ataqueProfundo(int posicion, Grafo cancha){
      //init
      Lista camino=listaCrear();
      int marcas=0;
      return ataqueProfundoR(posicion, cancha, marcas, camino);

}
```

```
Lista ataqueProfundoR(int posicion, Grafo cancha, int marcas, camino){
      if (getTipoNodo(pos, cancha)==META) //llego al area
            return camino;
      else {      //LIBRE
            if (esMarcado(pos, cancha)) {
                  if (marcas+1==K)  //camino riesgoso -> descarta
                        return listaCrear();
                  else marcas++;
            }
            Lista aux=posAdelante(pos, cancha); //busca seguir avanzando
            while (!listaEstaVacia(aux)){
                  int nuevaPos=listaPrimero(aux);
                  if (getTipoNodo(nuevaPos, cancha)!=AMIGO &&
getTipoNodo(nuevaPos, cancha)!=RIVAL){
                        listaInsertarUltimo(camino,nuevaPos);
                        if (!listaEstaVacia(ataqueProfundo(nuevaPos, cancha,
marcas))){
                              return camino;
```

```
                        }
                        listaSacarUltimo(camino);
                  }
                  listaSacar(aux);
            }
            listaDestruir(aux);
            return listaCrear();
      }
}
```

```
// funcion auxiliar que indica si hay un rival adyacente a la posicion 'pos'
bool esMarcado(int pos, Grafo cancha){
      Lista aux2=adyacentesNodoGrafo(cancha, pos){
      while (!listaEstaVacia(aux2)){
            if (getTipoNodo(listaPrimero(aux2),cancha)==RIVAL) {  //hay rival
                  return true;
            }
            else {
                  listaSacar(aux2);
            }
      }
      delete aux;
      return false;
}
```

```
int paseInmediato(int posicion, Grafo cancha){
      //init
      Lista aux=crearLista();
      Lista marcados=crearLista();
      listaInsertar(aux, posicion);
      iistaInsertar(marcados, posicion);
      while (!EstaVaciaLista(aux)){
            pos=listaPrimero(aux);
            if (getTipoNodo(cancha, pos)==AMIGO) { //encuentra AMIGO
                  listaDestruir(aux);listaDestruir(marcados);
                  return pos;
            }
            ListaSacar(aux);
            Lista aux2=posAdelante(pos,cancha);
            while (!EstaVaciaLista(ade)) {      // inserta primero posiciones de
adelante para priorizar segun letra
                  int i= listaPrimero(ade);
                  listaSacar(ade);
                  if (!listaPerteneceElem(marcados,i)) { // posicion adyacente
no visitada
                        listaInsertarUltimo(aux, i);
                        listaInsertarUltimo(marcados, i);
                  }
            }
            listaDestruir(aux2);
            aux2=adyacentesNodoGrafo(cancha,pos); //inserta resto de las
posiciones
            while (!EstaVaciaLista(ady)) {
                  int i= listaPrimero(ady);
                  listaSacar(ady);
                  if (!listaPerteneceElem(marcados,i) &&
!listaPerteneceElem(aux)) { // posicion adyacente no visitada y no agregada a
la cola de procesamiento
                        listaInsertarUltimo(aux, i);
                        listaInsertarUltimo(marcados, i);
                  }
            }
      }
      return -1;
}
```

Problema 2 (30 puntos)

Sea $f(i,m,w)$, el valor mínimo en bytes que se obtiene seleccionando entre *i ringtones* con $1 \le i \le n$, restringido al presupuesto máximo *m* y valor en ranking total como mínimo *w*.

Los pasos base se dan cuando:

- No hay mas ringtones disponibles y no se ha cubierto el mínimo valor en ranking total *w. (*Nótese que el caso representa la no existencia de solución):
  $f(0,m,w) = +\infty \quad con \quad w > 0$
- No hay mas ringtones disponibles y se ha cubierto el mínimo valor en ranking total *w*:
  $f(0,m,w) = 0 \quad con \quad w \le 0$

En cada paso recursivo (*i*), el presupuesto disponible podrá ser o no suficiente para cubrir el costo del ringtone *i*; donde en caso de serlo, se tiene la opción de seleccionarlo o no. Es decir que los casos son:

- El dinero disponible no es suficiente para cubrir el costo del ringtone *i*, por lo que no se puede seleccionar ese ringtone:
  $f(i,m,w) = f(i-1,m,w) \quad si \quad p(i) > m \quad con \quad 0 < i \le n$
- El dinero disponible es suficiente para cubrir el costo del ringtone *i* por lo que se tiene la opción de seleccionar o no el ringtone *i,* donde se optará por la opción que minimice el valor en bytes*:*
  $f(i,m,w) = \min(f(i-1,m,w), d(i) + f(i-1,m-p(i),w-r(i))) \quad si \quad p(i) \le m \quad con \quad 0 < i \le n$

Por lo tanto, la función f es de la forma:

$$f(i,m,w) = \begin{cases} +\infty \quad si \quad i = 0 \quad \wedge \quad w > 0 \\ 0 \quad si \quad i = 0 \quad \wedge \quad w \le 0 \\ f(i-1,m,w) \quad si \quad p(i) > m \quad \wedge \quad 0 < i \le n \\ \min(f(i-1,m,w), d(i) + f(i-1,m-p(i),w-r(i))) \quad si \quad p(i) \le m \quad \wedge \quad 0 < i \le n \end{cases}$$

La llamada a la función para obtener el valor calórico mínimo es $f(n,P,R)$.