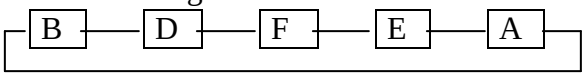
CC30A – EXAMEN – miércoles 23 de noviembre de 2005

1. EL TDA Circulo representa una estructura en que cada dato es único y está conectado con el anterior y con el siguiente en la forma indicada en el siguiente ejemplo:



Al respecto, la siguiente tabla define los métodos disponibles en la clase Circulo:

ejemplo	**significado**
new Circulo()	inicializa un Circulo vacío (sin datos)
boolean contiene(Object x)	true si x está en el Circulo (false si no)
void agregar(Object x)throws YaExiste	agrega x al final del Circulo
void borrar(Object x)throws NoExiste	Borra x del circulo
Object siguiente(Object x)throws NoExiste	Entrega dato siguiente a x. Ej: el siguiente de "A" es "B",y el sgte de "Z" no existe
Object anterior(Object x)throws NoExiste	Entrega dato anterior a x. Ej: el anterior a "B" es "A"
int largo()	Nº de datos en el Circulo. Ej: 5
Object primero()throws NoExiste	Entrega primer dato del Circulo (en el ejemplo "B")

a) Use la clase Circulo para escribir el método de encabezamiento

static public int distancia(Circulo X,Object Y,Object Z) throws NoExiste

que entregue la distancia (en nº de datos) que hay en el Circulo X para ir desde Y hasta Z. Por ejemplo, distancia(C,"D","E") es 2 y distancia(C,"E","B") también es 2 (puesto que es más corto pasar por "A").

Nota. Implemente la búsqueda en 2 threads, que recorran concurrentemente en ambos sentidos.

```
//calcular distancia  hacia atrás en 2º thread:0.2
T t=new T(X,Y,Z);  //0.1
t.start();         //0.1
//calcular distancia hacia delante:0.3
int d=0;
while(!Y.equals(Z)){//0.1
  Y=X.siguiente(Y);//0.1
  ++d;             //0.1
}
//determinar y entregar menor distancia: 0.5
t.join();          //0.1
return Math.min(d,t.resultado());//0.4
//Thread: 1 punto
//declaraciones: 0.2
class T extends Thread{
private Circulo X;
private Object Y,Z;
private int d;
//constructor: 0.2
public T(Circulo X,Object Y,Object Z){//0.1
  this.X=X, this.Y=Y, this.Z=Z;//0.1
}
//método run: 0.4
public void run(){
  d=0;               //0.1
  while(!Y.equals(Z)){//0.1
    Y=X.anterior(Y);  //0.1
    ++d;             //0.1
  }
}
//método resultado: 0.2
public int resultado(){ return d;}
```

b) Escriba el método **anterior** (de la clase Circulo) suponiendo que la representación es un arreglo.
Nota. escriba las declaraciones correspondientes.

```
//representación: 0.5 ptos
int N=100, n=0;         //0.2
Object[]a=new Object[N];//0.3
//método: 1.5 ptos
public Object anterior(Object x)throws NoExiste{
  int i=indice(x,a,n);           //0.5
  if(i<0) throw new NoExiste();  //0.5
  if(i==0) return a[n-1];        //0.3
  return a[i-1];                 //0.2
}
```

c)Escriba el método **borrar** suponiendo que la representación es una lista de doble enlace circular, es decir, que el último nodo apunta al primero, y el primero al último.
Nota. escriba las declaraciones correspondientes.

```
//representación: 0.5 ptos
class Nodo{             //0.1
public Object valor;    //0.1
public Nodo ant, sgte;  //0.2
}
private Nodo primero;   //0.1

//método: 1.5 ptos
public void borrar(Object x)throws NoExiste{
  Nodo r=referencia(x);//0.1
  if(r==null)           //0.1
    throw new NoExiste();//0.2
  if(r.sgte==r)         //0.1
    primero=null;       //0.1
  else{
    r.sgte.ant=r.ant;   //0.1
    r.ant.sgte=r.sgte;  //0.1
    if(r==primero) primero=primero.sgte;//0.2
  }
}
//buscar x en lista enlazada circular: 0.5
private Nodo referencia(Object x){
  for(Nodo r=primero; r!=null; r=r.sgte){//0.2
    if(r.valor.equals(x))return r;       //0.1
    if(r.sgte==primero) break;          //0.1
  }
  return null;//0.1
}
```

2.Al TDA Diccionario se le ha agregado el método **Queue enOrden("A" o "D")**
que entrega una cola con las palabras (sin los significados) en orden ascendente o descendente.
Implemente el método en O(nlogn) suponiendo que el diccionario está representado con

a) un árbol-AVL (ABB balanceado)

```
class Nodo{
public Comparable palabra; public Object significado;
public Nodo izq, der;
}
//método enOrden: 1 punto
public Queue enOrden(String x){
  Queue q=new Queue();  //0.3
  if(x.equals("A"))     //0.1
     izqRaizDer(raiz,q);//0.2
  else
    derRaizIzq(raiz,q); //0.2
  return q;             //0.2
}
//orden ascendente: 1 pto
private void izqRaizDer(Nodo r,Queue q){//0.1
  if(r==null) return;   //0.2
  izqDerRaiz(r.izq,q);  //0.2
  q.enque(r.palabra);   //0.3
  izqDerRaiz(r.der,q);  //0.2
}
//orden descendente: 1 pto
private void derRaizIzq(Nodo r,Queue q){//0.1
  if(r==null) return;   //0.2
  derRaizIzq(r.der,q);  //0.2
  q.enque(r.palabra);   //0.3
  derRaizIzq(r.izq,q);  //0.2
}
```

b) una tabla de hashing con rehashing lineal

```
//declaraciones
static protected final int N=100;
protected Elemento[]tabla;
class Elemento{public Object palabra, significado;public boolean borrado;}
public Queue enOrden(String x){
  //crear arreglo con palabras activas: 1 punto
  Comparable[]a=new Comparable[N];        //0.2
  int n=0;                                //0.1
  for(int i=0;i<N;++i)                    //0.2
    if(tabla[i]!=null && !tabla[i].borrado) //0.3
      a[n++]=tabla[i].palabra;            //0.2
  //ordenar con algoritmo O(nlogn): 0.5
  quicksort(a,0,n-1);
  //crear y entregar Queue con resultado: 1.5
  Queue q=new Queue();  //0.3
  for(int i=0; i<n; ++i)//0.3
    if(x.equals("A"))   //0.1
      q.enque(a[i]);    //0.3
    else
      q.enque(a[n-1-i]); //0.3
  return q;             //0.2
}
```

3.Al TDA Grafo (no dirigido) se le ha agregado el método **int numeroArcos().**
Implemente el método suponiendo que el grafo está representado con

a) un arreglo de vértices y una matriz de adyacencia

```
class Grafo{
protected Vertice[] v;
protected boolean[][] a;
protected static final int N=100;
protected int n;

int na=0;                    //0.3
for(int i=0; i<n; ++i)       //0.5
  for(int j=i+1; j<n; ++j)   //1.0
    if(a[i][j])              //0.5
      ++na;                  //0.4
return na;                   //0.3
```

b) lista enlazada de vértices (y en cada vértice una lista enlazada de sus vértices adyacentes)

```
class NodoVertice{
public Vertice vertice; public NodoVertice sgte; public NodoArco lista;
}
class NodoArco{
public Vertice vertice; public NodoArco sgte;
}
class Grafo{
protected NodoVertice primero;
public Grafo(){primero=null;}

//crear un vector con todos los arcos no repetidos: 2.5 puntos
Vector v=new Vector();                          //0.2
for(NodoVertice r=primero; r!=null; r=r.sgte)   //0.5
  for(NodoArco ra=r.lista; ra!=null; ra=ra.sgte){//0.5
    Arco a=new Arco(r.vertice,ra.vertice);      //0.5
    if(v.indexOf(a) < 0) v.addElement(a);       //0.5
  }
return v.size();                                //0.3
//clase Arco: 0.5 ptos
class Arco{                          //=.1
public Vertice v, w;                 //0.1
public boolean equals(Arco x){       //0.1
  return v.equals(x.v) && w.equals(x.w) //0.1
      || v.equals(x.w) && w.equals(x.v);//0.1
}
}
```