IMPRIMIR RAMAS DEL ARBOL

```
public ListaCD<ListaCD<T>> imprimirRamas(){
        ListaCD<ListaCD<T>> l=new ListaCD<ListaCD<T>>();
        if(this.esVacio()||this.esHoja(this.raiz))
                return l;
        Cola <T> c=this.buscarHojas();
        while(!c.esVacio()){
                T x= c.deColar();
                 l.addFin(this.getCamino(x));
        }
        return l;
}

/*************************************************************/
public Cola<T>  buscarHojas(){
        Cola<T> l=new Cola<T>();
        buscarHojas(this.raiz, l);
        return (l);
}

/*************************************************************/
private void buscarHojas(NodoB<T> r, Cola<T> l){
        if (r!=null){
                if(this.esHoja(r))
                        l.enColar(r.getInfo());
                buscarHojas(r.getIzq(), l);
                buscarHojas(r.getDer(), l);
        }
 }

/*************************************************************/
public ListaCD<T> getCamino(T info){
        ListaCD<T> l= new ListaCD<T>();
        getCamino(this.raiz,info,l);
        return(l);
}
```

```
/*************************************************************/
private void getCamino(NodoB<T>r,T info,ListaCD<T>l){
        if(r==null)
                return;
        if(r.getInfo().equals(info)){
                l.addFin(info);
                return;
        }
        l.addFin(r.getInfo());
        if(this.esta(r.getIzq(), info))
                 getCamino(r.getIzq(),info,l);
        else
                getCamino(r.getDer(),info,l);
     }
}
```

EJERCICIO QUE PERMITE HALLAR LA RUTA MINIMA ENTRE DOS NODOS

```
public ListaCD RutaMinima(T  info1,T info2){
            ListaCD<T> l1=new  ListaCD<T>();
            ListaCD<T> l2=new  ListaCD<T>();

            if(this.raiz==null)
                    return l1;

            if(this.raiz.getInfo().equals(info1)){
                    ListaCD<T> l=new  ListaCD<T>();
                    this.hallarCamino(this.raiz, info2, l);
                    return  l;
             }
             if(this.raiz.getInfo().equals(info2)){
                     ListaCD<T> l=new  ListaCD<T>();
                     this.hallarCamino(this.raiz, info1, l);
                    return  l;
             }
             this.hallarCamino(this.raiz, info1, l1);
             this.hallarCamino(this.raiz, info2,l2);

             ListaCD<T> Ruta=this.DefinirRuta(l1,l2);
             return Ruta;
```

```
}

/*************************************************************/
private ListaCD<T> DefinirRuta(ListaCD<T> l1,ListaCD<T> l2){
            ListaCD<T> ruta=new ListaCD<T>();
            T  Repetido=null;
            for(T x:l1){
                  ruta.addFin(x);
                   if(this.seRepite(x,l2)){
                              Repetido=x;
                              break;
                  }
            }
             this.LlenarResto(ruta,l2,Repetido);
            return ruta;
}

/*************************************************************/
private void LlenarResto(ListaCD<T> ruta,ListaCD<T> l,T repetido){
           int i=l.indexOf(repetido);
           while(--i>=0){
                  ruta.addFin(l.get(i));

           }
}

/*************************************************************/
private boolean seRepite(T repetido,ListaCD<T> l2){
          for(T y:l2){
                  if(y.equals(repetido))
                              return true;
           }
          return false;
}

/*************************************************************/
private boolean hallarCamino(NodoB<T> r,T info,ListaCD<T> l){

            if(this.esHoja(r)&&!r.getInfo().equals(info))
                        return  false;
            if(r==null)
                        return false;
             if(r.getInfo().equals(info)){
```

```
                l.addFin(r.getInfo());
                return  true;
            }
            if(this.hallarCamino(r.getIzq(), info, l)|| this.hallarCamino(r.getDer(), info, l)){
               l.addFin(r.getInfo());
               return true;
            }
            return  false;
}
```

RETORNAR EL MAYOR ELEMENTO DEL NIVEL

```
public T mayorNivel(int n) {
    if(n<0)
        return null;
    ListaCD<T> l= new ListaCD<T>();
    this.nodosNivel(this.raiz,l,n,0);

    if(l.esVacio())
        return null;
    return(this.mayorNodo(l));
  }

/*************************************************************/
private void nodosNivel(NodoB<T> r,ListaCD<T> l,int nb,int na) {
    if(r==null)
        return;

    if(na==nb) {
        l.addFin(r.getInfo());
        return;
    }
    nodosNivel(r.getIzq(), l, nb, na+1);
    nodosNivel(r.getDer(), l, nb, na+1);
  }
```

```
/*************************************************************/
private T mayorNodo(ListaCD<T> l){
      Iterator<T> it=l.iterator();
      T may= l.get(0);
      T num=null;
      while(it.hasNext()){
           num=it.next();
           int compara=((Comparable)may).compareTo(num);

           if(compara<0){
                    may=num;
           }
       }
       return may;
}
```

VECINOS A DISTANCIA

```
/**
 * @param T info con la info del nodo a buscar,
 * @param int x con la distancia de los vecinos,
 * @return String con los vecinos
 */

public String vecinos(T info, int x) {
          if (info == null){
                   return "El info dado es NULL";
          }
          if(this.raiz == null) {
                    return "El Arbol es Vacio";
           }
           if( this.raiz.getInfo().equals(info)){
                     return "El info es la raiz del arbol... por lo tanto no tiene vecinos...";
          }

           ListaCD<NodoB<T>> listVecinos = new ListaCD<NodoB<T>>();
           listVecinos.addFin(this.raiz);
          listVecinos = this.vecinos(listVecinos, info, x);
```

```
        if (listVecinos != null) {
                return ""+listVecinos.toString();
        } else {
                return "El info no existe en el arbol...";
        }
}



/*************************************************************/

   /**
   *@param ListaCD<NodoB<T>> listVecino con la raiz
   *@param T info del info a buscar
   *@param int x con la distancia de los vecinos
   *@return ListaCD<NodoB<T>> con los vecinos del nodo
   **/

private ListaCD<NodoB<T>> vecinos(ListaCD<NodoB<T>> listVecino, T info, int x) {
     if (listVecino.esVacio()) {
         return null;
     }
     int pos = listVecino.indexOf(new NodoB<T>(info));

     if (pos > -1) {
        ListaCD<NodoB<T>> listaNivel = new ListaCD<NodoB<T>>();
        if (pos - x > 0) {
                listaNivel.addFin(listVecino.get(pos - x));
        }
        if (pos + x < listVecino.getSize()) {
                listaNivel.addFin(listVecino.get(pos + x));
        }
        return listaNivel;
     } else {
         ListaCD<NodoB<T>> lt = new ListaCD<NodoB<T>>();
         for (NodoB<T> nt : listVecino) {
                if (nt.getIzq() != null) {
                        lt.addFin(nt.getIzq());
                }
                if (nt.getDer() != null) {
                        lt.addFin(nt.getDer());
                }
        }
```

```
            listVecino = null;
            return vecinos(lt, info, x);
        }
    }
```

EJERCICIOS RESUELTOS DE ARBOLES BINARIOS

ANCESTRO

METODO USADO PARA HALLAR EL ANCESTRO COMÚN MAS PRÓXIMO DE LOS ELEMENTOS DE LA LISTA

```
public T ancentroLista(ListaCD<T> l) {
    ListaCD<ListaCD<T>> lista = new ListaCD<ListaCD<T>>();
    for(T dato:l)
        lista.addFin(this.hallarRecorrido(dato));

     ListaCD<T> primera = lista.get(0);
     T ancentro = null;
     for (T dato2 : primera) {
         boolean valor = true;
        for (ListaCD<T> lis : lista) {
                if (lis.indexOf(dato2) == -1) {
                        valor = false;
                }
        }
        if (valor) {
                 if(dato2==primera.get(0) && dato2!=this.raiz){
                        ancentro=this.padre(dato2);
                }else{
                        ancentro = dato2;
                        return ancentro;
                }
        }
     }
     return ancentro;
  }
```

MÉTODO USADO PARAIMPRIMIR EL RECORRIDO DE UN NODO A LA RAIZ

```
public ListaCD<T> hallarRecorrido(T x) {
    ListaCD<T> l = new ListaCD<T>();

    while (((Comparable) x).compareTo(this.raiz.getInfo()) != 0) {
            l.addFin(x);
            x = this.padre(x);
    }
    l.addFin(this.raiz.getInfo());
    return l;
}
```

PRIMOS

```
public ListaCD<T> getPrimos(T dato){
    if(this.raiz==null){
        return null;
    }
    int h=0;
    h=this.nivelNodoP(dato);
    T padre=this.padre(dato);
    System.out.println(padre.toString());
    if(h==-1){
        return null;
    }
    ListaCD<T> l=new ListaCD<T>();
    this.getPrimos(raiz, padre, h, 0, l);
    return l;
  }
```

```
/*************************************************************/
private void getPrimos(NodoB<T> r, T padre, int nivel, int act, ListaCD<T> l){
        if(r==null){
                return;
         }
         if(this.esHoja(r)){
                return;
         }
        if(r.getInfo().equals(padre)){
                return;
        }
        if(r.getDer()!=null){
                if((act+1)==nivel){
                        l.addFin(r.getDer().getInfo());
                }
        }
        if(r.getIzq()!=null){
                 if((act+1)==nivel){
                        l.addFin(r.getIzq().getInfo());
                 }
         }
        act++;
        if(act==nivel){
                 return;
        }
        this.getPrimos(r.getDer(), padre, nivel, act, l);
       this.getPrimos(r.getIzq(), padre, nivel, act, l);
  }
```