Caso	nomarchs	contenidos
1	[“Arch20”, “Arch7”]	[“cambio2”,”cambio1”]
2	[“Arch27”]	[“cambio27”]

Solución ej 1:

```
a)struct NodoArchivos {
                        Archivo elem;
                        struct NodoArchivos *der, *izq;
        };

b)
//Nota los elementos se agregan al pricnipio de la lista
 TAD LCADENA
   typedef struct Nodo* LCADENA;
  //Constructoras
  LCadena Vacia();
  //Pòst: devuelve la lista vacia
  LCadena Cons(Cadena elem,LCadena l);
      //Post retorna uan nueva lista que cnsiste de agregar elem al ,
      //al pricipio,
```

//Predicado
Bool Espacia(LCadena l);//Post retorna verdadero si al lista l es vacia

//Selectoras
Cadena getHead(LCadena l);
//Pre: la lista no es vacia
//Post: retorna el primer elemento de la lista

LCadena Tail(LCadena l);
//Pre la lista no es vacia
//post retorna la lista l sin su primer elemento

//Asuma que todos los nombes de achivo que se encuentran en la lista nomarchs, estan en el arbol, ni en el arbol ni en la lista hay elementos repetidos
c)

void CambiarContenidos(Archivos as, LCadena& nomarchs, LCadena& contenidos)
{
//El orden de los if es importante ya que la lista esta ordenada de forma decreciente
   If(as!=NULL){
   If(nomarchs!=NULL && strcmp(getHead(nomarchs) , `getNombreArchivo(as->elem)`)>0){
          CambiarContenidos(as->der, nomarchs, contenidos);

   }
   //
   If(nomarchs!=NULL && strcmp(getHead(nomarchs) , `getNombreArchivo(as->elem)`)==0){
          `setContenidoArchivo(as->elem,getHead(contenidos));`
          `contenidos=tail(contenidos);`

```
        nomarchs=tail(nomarchs);
  }
  If(nomarchs!=NULL && strcmp(getHead(nomarchs) , getNombreArchivo(as->elem) )<0){

    CambiarContenidos(as->izq, nomarchs, contenidos);

  }
 }

}
```

Ejercicio 2 (13 puntos)

a) (2 pts) De una especificación mínima para el TAD Árbol binario de búsqueda de naturales (ABBNat). Dando un conjunto mínimo de operaciones constructoras, selectoras, predicado y destructoras.

Solucion:

```
typedef struct ABBNode* ABB;


ABB VacioABB();
ABB ConsTreeABB(int elem,ABB l, ABB d);
ABB getIzqABB(ABB a);
ABB getDerABB(ABB a);
int getRaizABB(ABB a);
bool EmptyABB(ABB a);
void destruir(ABB &a);
```

b) (5pts) De un representación e implemente todas las operaciones para el TAD árbol binario de búsqueda de la parte a).

Solucion:

```
typedef struct ABBNode{
   int elem;
   struct ABBNode* izq;
   struct ABBNode* der;
};

//Operqaciones Basicas del TAD

//Constructoras
ABB VacioABB(){
  return NULL;
}

ABB ConsTreeABB(int elem,ABB l, ABB d){
  ABB arbol=new ABBNode;
  arbol->elem=elem;
  arbol->izq=l;
```

```
  arbol->der=d;
  return arbol;
}

//Selectoras
ABB getIzqABB(ABB a){
    return a->izq;
}

ABB getDerABB(ABB a){
    return a->der;
}

int getRaizABB(ABB a){
    return a->elem;
}
//Predicados
bool EmptyABB(ABB a){
     return a==NULL;
}

//Destructoras
void destruir(ABB &a){
     if(a!=NULL){

       if(a->izq!=NULL){
          destruir(a->izq);
       }
       if(a->der!=NULL){
          destruir(a->der);
       }
       delete a;
       a=NULL;
     }
}
```

c) (6 pts) Sin acceder a la representación de ABBNat, implemente la función auxiliar

int SumaNivel(ABBNat a,int niveln);

que devuelve la suma de todos los naturales que se encuentran en el nivel "niveln" del árbol a.

Nota:

-No se puede utilizar ninguna función auxiliar.

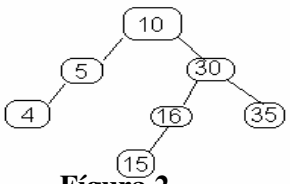


Figura 2

Ejemplos:

SumaNivel(a,1)=10
SumaNivel(a,2)=35
SumaNivel(a,4)=15

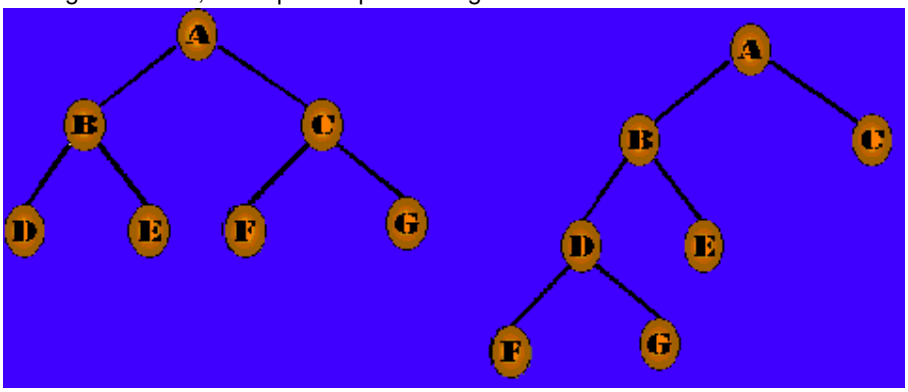
Solución:

```
//Raiz nivel 1
//Asme siempre viene un ntural >=1
   int  SumaNivel(ABBNat  a,int niveln){
   if(a!=NULL){
     if(niveln==1) return a->elem;
     else{
       return SumaNivel(a->izq,  niveln-1) + SumaNivel(a->der,niveln - 1);
     }

}return 0;//a=NULL

   }
```

Ejercicio 4 (13 puntos)

Dados dos árboles binarios a, y b , y la siguiente definición:

ÁRBOLES BINARIOS SEMEJANTES : Dos árboles binarios son semejantes si tienen el mismo número de nodos y los valores de los nodos del primer árbol son los mismos que los valores de los nodos del segundo árbol, sin importar que no tengan las mismas relaciones entre ellos.



Implemente la función bool Semejantes(Arbol a, Arbol b), accediendo a la representación del tipo Arbol(árbol binario de caracteres) que devuelve verdadero si los dos árboles son semejantes. Para resolverlo se pueden utilizar TADs y funciones auxiliares, pero debe implementar todas las funciones que utilice. Comente los pasos más importantes de su solución

```
bool Semejantes(Arbol a, Arbol b){
LChar a1,a2;
a1=GenerarListaAPartirDeArbol(a);
a2=GenerarListaAPartirDeArbol(b);
if(largoLista(a1)!=largoLista(a2)) return false;
else{
   bool distinto=false;
   int largo= largoLista(a1);
```

```
int cont=0;
int h;
char elem:
LChar auxl;
bool encontre=false;
   while(!distinto && cont<largo){
       elem=getHead(a1);
      encontre=false;
       h=0;
       auxl=a2;
       while(!encontre && h<largo){
            if(elem==getHead(auxl)){
                 encontre=true;
            }else{
              h++;
              auxl=tail(auxl);
            }
       }//while
      If(encontre){
          cont++;
          a1=tail(a1);
       }
      Else distinto=true;

    }//while
  Return distinto;//distitno vale verdadero si son semejantes
}


}
```

TAD Auxiliares utilizados:

Lista de Caracteres LChar

Usamos:

```
  Struct NodoLista* LChar;
  Struct NodoLista{
        Char elem;
        Struct NodoLista* sig;
   }
   Char getHead(LChar l){
         Return l->elem;
    }
LChar  concat(LChar a, LChar b){
    LChar aux;
    Aux=a;
    If(a==NULL)return b;
    ELSE{
         WHILE(A.>SIG!=null) Aux=Aux->SIG;
```

```
            aux->sig=b;
            return a;
    }
}

Int largoLista(LChar l){
       If(l==NULL) return 0;
       Else return 1 + largoLista(l->sig);
}

LChar tail(LChar l){
         Return l->sig;
 }
```

Se creo una función auxiliary GenerarListaAPartirDeArbol en el TAD ARBOL,

```
LChar GenerarListaAPartirDeArbol(ARBOL a){
  LChar l=Vacia();
  If(a!=NULL)
    l=  Cons(a->elem, GenerarListaAPartirDeArbol(a->izq));
    return concat(l, GenerarListaAPartirDeArbol(a->der));

}else return l;
```

Idea de la solución:
A partir de los árboles me creo una lista por cada uno de ellos con todos sus elementos.
Si el largo de las listas son distintos retorno falso, d lo contrario,
Recorro elemento por elemento de la primera lista y me fijo si se encuentra en la segunda lista, si todos los elementos de la primera lista están en la segunda retorno verdadero.(recordar que si estoy este punto las listas tienen el mismo largo), sino si encuentro algún elemento de la primera lista q no esta en la segundo retorno falso.