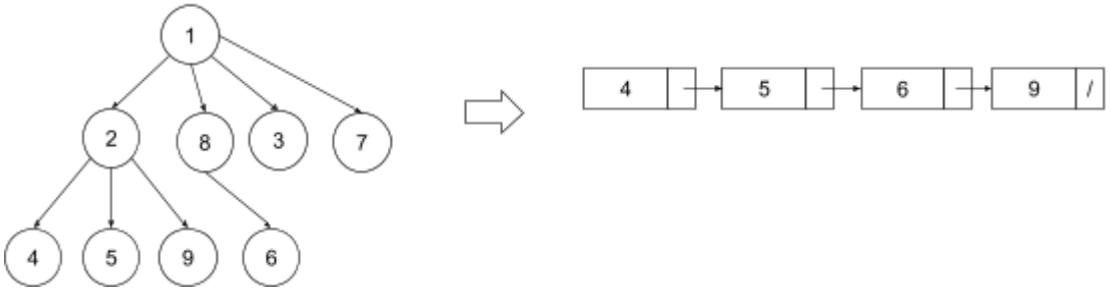
Ejercicio 1

Crear una lista ordenada con los valores de los nodos hijos de cada nodo par de un árbol n-ario.

Ejemplo:



La función debe seguir el siguiente protocolo:

```
list *hijosDePares (ntn *root);
```

Teniendo en cuenta las siguientes estructuras:

```
typedef struct ntNode ntn;
typedef struct ntList ntlist;

typedef struct ntList {
    ntn *node;
    ntlist *next;
} ntlist;

typedef struct ntNode {
    int value;
    ntlist *sons;
} ntn;
```

```
typedef struct List list;
typedef struct List {
    int value;
    list *next;
} list;
```

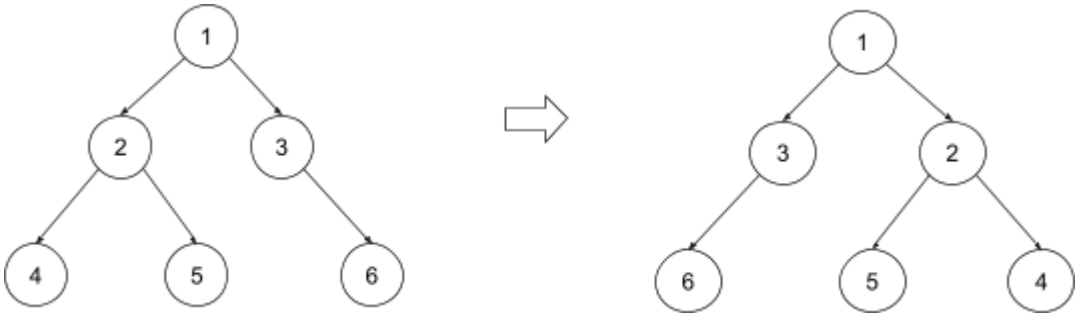
Consideraciones:

- En caso que la función llame a otras funciones debe incluir todas las funciones que utilice.
- Si utiliza una estructura auxiliar debe estar definida.

Ejercicio 2

Armar una función que cree y devuelva un árbol binario que sea el “espejo” del árbol pasado por parámetro.

Por ejemplo:



La función debe seguir el siguiente protocolo:

```
btn *mirror (btn *root);
```

Teniendo en cuenta la siguiente estructura:

```
typedef struct btNode btn;
typedef struct btNode {
    int value;
    btn *left;
    btn *right;
} btn;
```

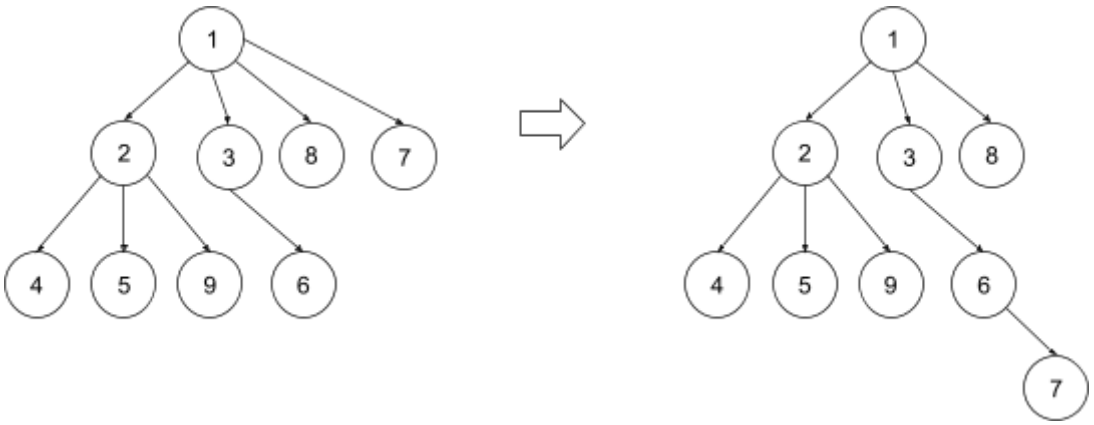
Consideraciones:

- No se debe perder el árbol original.
- El algoritmo debe funcionar para cualquier árbol binario.
- El árbol resultante debe ser un árbol independiente.
- En caso que la función llame a otras funciones debe incluir todas las funciones que utilice.
- Si utiliza una estructura auxiliar debe estar definida.

Ejercicio 3

Armar una función que mueva el nodo de un árbol n-ario a otro padre.

Ejemplo: mover de 7 como hijo de 6



La función debe seguir el siguiente protocolo:

```
void moveNode (ntn *root, int nodo, int nuevoPadre);
```

Teniendo en cuenta la siguiente estructura:

```
typedef struct ntNode ntn;
typedef struct ntList ntlist;

typedef struct ntList {
    ntn *node;
    ntlist *next;
} ntlist;

typedef struct ntNode {
    int value;
    ntlist *sons;
} ntn;
```

Consideraciones:

- El nodo a mover NO debe ser la raíz.
- El nodo a mover puede estar en cualquier posición de la lista de hijos.
- Si el nodo a mover tiene hijos, los debe conservar.
- No se repiten claves en el árbol.
- Si el “nuevo padre” no existe o el nodo no existe, la subrutina no debe hacer nada.
- El árbol resultante debe ser un árbol independiente.
- En caso que la función llame a otras funciones debe incluir todas las funciones que utilice.
- Si utiliza una estructura auxiliar debe estar definida.

Ejercicio 4

Desarrollar el código en C para:

Crear un nuevo digrafo ponderado a partir de otro, con una lista de claves (o valores string) de los vértices que deben ser incluidos en el nuevo grafo. Debe incluir solo las aristas que involucren a los vértices del nuevo grafo.

Entrada de la función principal:

- puntero a digrafo original (implementación a elección)
- puntero a lista con las claves o valores strings a incluir en el nuevo grafo (implementación a elección)

Salida de la función principal:

- puntero a nuevo digrafo

Consideraciones:

- El nuevo grafo debe tener las aristas de los vértices involucrados. Por ejemplo si “Madrid” y “Londres” están en la lista y existe una arista que los conecta, la misma debe existir en el nuevo digrafo.
- Los valores de las claves son “strings”, y en el digrafo resultante puede que deban adoptar otro índice.
- Puede utilizar cualquier implementación de grafos o listas.
- Debe definir o transcribir todas las estructuras y algoritmos adicionales utilizados.
- Debe incluir todas las funciones utilizadas.

Ejercicio 5

Desarrollar el código en C para:

Crear un nuevo grafo a partir de otros 2 grafos, teniendo en cuenta los nombres (o claves) de los vértices.

Entrada de la función principal:

- puntero a grafo 1 (implementación a elección)
- puntero a grafo 2 (implementación a elección)

Salida de la función principal:

- puntero a nuevo grafo

Consideraciones:

- Los grafos no son dirigidos, ni ponderados.
- Los valores de las claves son “strings”, y en los grafos pueden encontrarse en índices diferentes. Por ejemplo: el grafo 1 tiene a “Madrid” con índice 0, pero en el grafo 2 “Madrid” está en el índice 3.
- Puede utilizar cualquier implementación.
- Debe definir o transcribir todas las estructuras y algoritmos adicionales utilizados.
- Debe incluir todas las funciones utilizadas.

Ejercicio 6

Desarrollar el código en C para:

Mostrar el costo mínimo y el camino mínimo entre 2 nodos de un digrafo ponderado, que necesariamente pasen por un 3er nodo pasado por parámetro.

Entrada de la función principal:

- Puntero al digrafo (implementación a elección)
- Índice del nodo origen.
- Índice del nodo destino.
- Índice del nodo por el cual debe pasar.

Consideraciones:

- Puede utilizar cualquier implementación.
- Debe definir o transcribir todas las estructuras y algoritmos adicionales utilizados.
- Debe incluir todas las funciones utilizadas.