Práctica 4

Objetivos

- Familiarizar al estudiante con los recorridos en arboles binarios.
- Que el estudiante sea capaz de construir un árbol binario y hacer una búsqueda sobre el mismo.

Problema

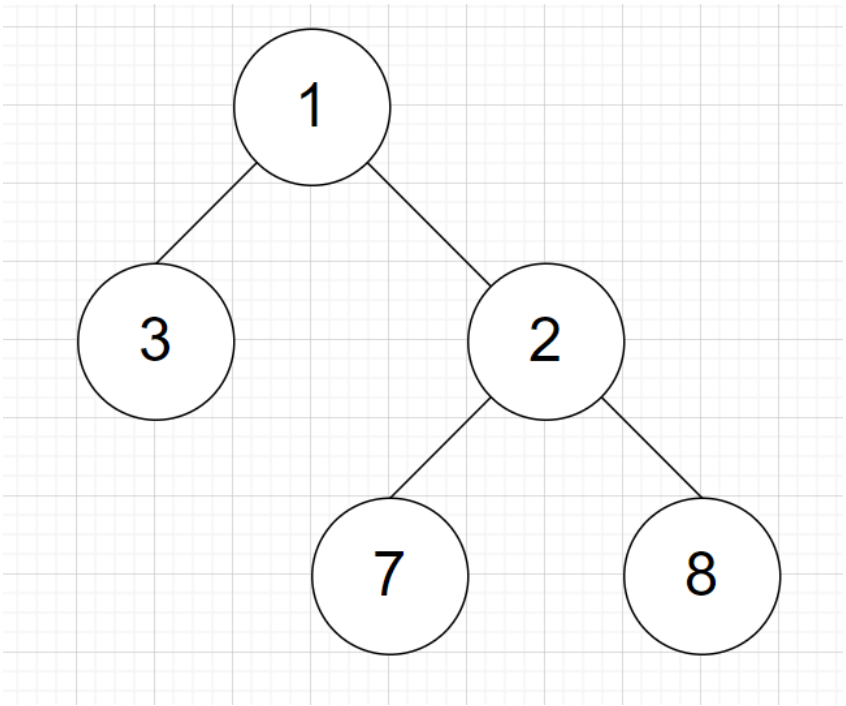
Se puede demostrar que el recorrido inorder y posorder de un árbol binario nos induce un único árbol. Es decir, si nos dan el recorrido inorder; por ejemplo,

$$[3, 1, 7, 2, 8]$$

Y el recorrido en posorden,

$$[3, 7, 8, 2, 1]$$

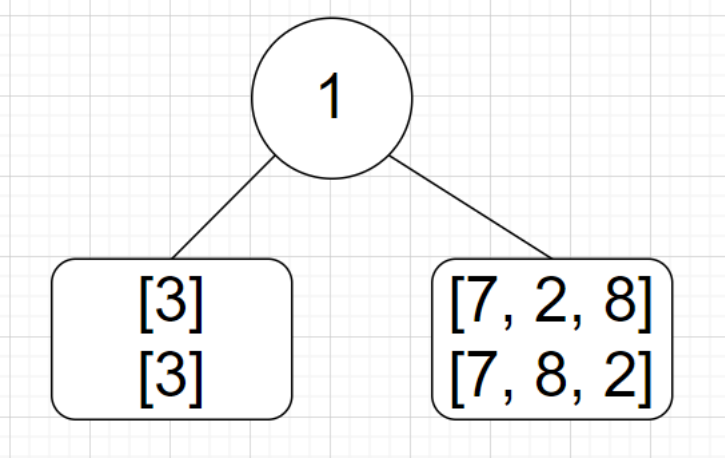
Solo existe un árbol binario que cumple tener esos recorridos y es



La razón de esto es que en el recorrido en posorden el ultimo nodo que nos queda es la raíz.

$$[3, 7, 8, 2, 1]$$

Y en el recorrido inorder podemos ver donde se encuentra dicho nodo. Entonces, todo lo que esta del lado izquierdo de ese nodo es el subárbol izquierdo. Y todo lo del lado derecho en ese lugar.



Y podemos repetir el proceso en cada hijo hasta formar el árbol completo. Solo se considera el caso baso de un arreglo vacío no forma ningún nodo.

Ejercicio

Se te darán dos arreglos del mismo tamaño y su tamaño. El primero representa el recorrido de un árbol en inorder y el segundo en posorder. Deberás imprimir el recorrido del árbol en preorder. Para esto primero deberás construir el árbol y luego hacer el recorrido del árbol en preorder.

En este practica deberás implementar una clase ArbolBinario. Con un constructor que reciba los arreglos como parámetro y debe tener un método RecorridoPreorder que cree de forma dinámica y devuelva un arreglo con el recorrido en preorder. Igual tenemos una estructura Nodo que nos representa los nodos del árbol.

```
struct Nodo {
    int id;
    Nodo *left, *right;
};

class ArbolBinario {
    public:
        ArbolBinario(int* inorder, int* posorder, int N);
        ~ArbolBinario();

        int* recorrido_preorder();
    private:
        Nodo* raiz;

};
```

Puedes añadir las funciones que creas correspondientes para complementar la clase. Sin embargo, esas funciones deberían ser privadas.

No se les olvide implementar el destructor que deberá borrar la memoria correspondiente a cada nodo. (Esto implica que cada nodo deberá ser creado con memoria dinámica).

Se te asegura que los arreglos que se te dan siempre representan un recorrido Inorder y Posorder de un árbol binario.

Ejemplo

Se te adjunta un archivo main que te facilita ejecutar el ejemplo que se mostró anteriormente.

```
int N = 5;

int recorridoInorder[] = {3, 1, 7, 2, 8};
int recorridoPosorder[] = {3, 7, 8, 2, 1};

ArbolBinario arbol(recorridoInorder, recorridoPosorder, N);
int* recorridoPreorder = arbol.recorrido_preorder();

std::cout << "Recorrido en preorder del arbol:\n";
for (int i = 0; i < N; i++)
    std::cout << recorridoPreorder[i] << " ";

delete[] recorridoPreorder;
```

Nota que dentro de la clase no es necesario eliminar la memoria del arreglo que se crea al llamar recorrido_preorder. Esa memoria se crea en la función, pero se elimina en el main.

Al finalizar el main se llama al destructor (esto lo hace el programa automáticamente) que debe eliminar la memoria usada para construir el árbol binario.

Compilación

De preferencia llamar a su librería “ArbolBinario.hpp” y las funciones que estén definidas en “ArbolBinario.cpp” para que se pueda ejecutar con el comando:

```
g++ --std=c++11 main.cpp ArbolBinario.cpp -o ArbolBinario
```

Punto extra

Se te dará un punto extra (base 10) si respondes a la siguiente pregunta.

Es cierto que un recorrido inorder y posorder de un árbol binario definen el árbol. Lo que no es cierto es que dos arreglos cualesquiera pueden definir el recorrido inorder y

posorder de un árbol. Por ejemplo, imagina que nos dicen que los siguientes arreglos son los recorridos de un árbol:

$$inorder = \{3, 7, 1, 2, 8\}$$
$$posorder = \{3, 1, 8, 7, 2\}$$

Se puede mostrar que estos nos son los recorridos de ningún árbol.

Explica en el reporte (no tienes que implementar nada) como harías para saber si los recorridos corresponden a un árbol binario sin tener que construir el árbol.