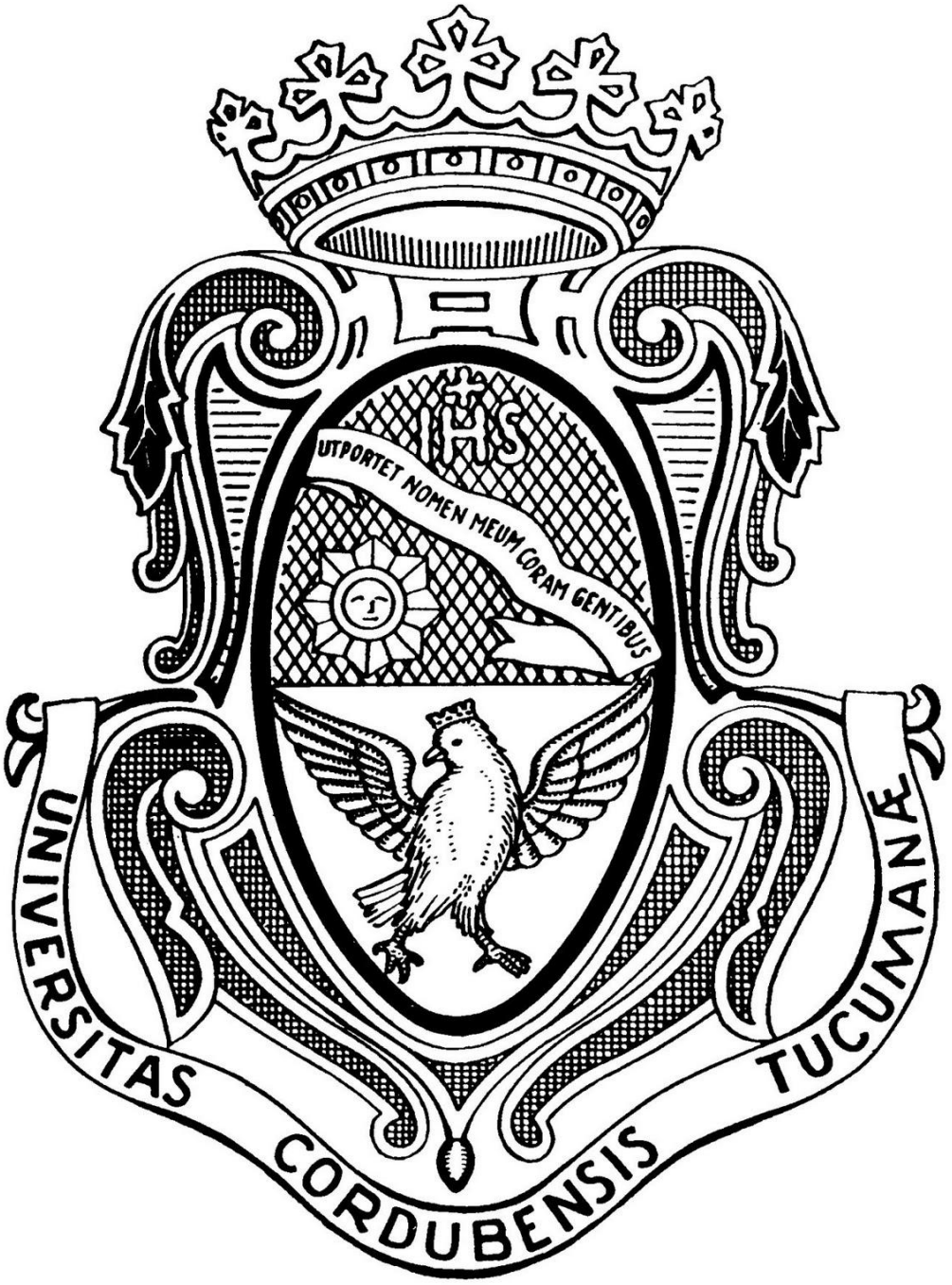
Algoritmos y Estructuras de Datos
Parcial 2



Alumnos:

Natale, Alfredo	95558906
Pasolli, Nestor Geremias	42853888
Moyano, Ghelsy Fernanda	42522757

Índice

Problema

Los archivos con formato JSON (JavaScript Object Notation - Notación de Objetos de JavaScript) tienen un formato ligero y son utilizados para el intercambio de datos. Su definición puede consultar se https://www.json.org/json-es.html y en https://www.mclibre.org/consultar/informatica/lecciones/formato-json.html.

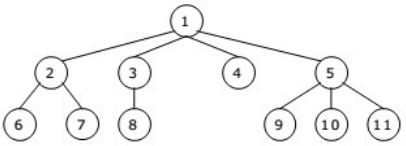
Ud deberá realizar el procesamiento de un archivo, preferentemente el archivo de descarga desde https://api.nobelprize.org/v1/laureate.json que corresponde a quienes han sido laureados con los premios Nobel.

La estructura de un archivo tiene distintos niveles de profundidad con múltiples objetos en cada nivel, por lo que este formato puede verse como un árbol m-ario. En concreto, la información con formato de “arreglo” puede verse como una serie de m-nodos hijos del objeto donde dicho arreglo está contenido.

La lectura del archivo debe cargarse en memoria sobre una estructura de un árbol m-ario. Este tipo de árbol debe implementarse sobre un árbol binario. La forma de implementar un árbol m-ario sobre un árbol binario es la siguiente:

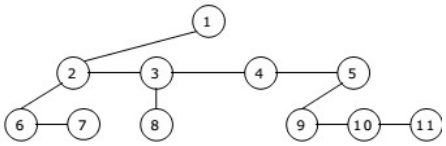
La raíz del árbol m-ario es la misma que la del binario. El hijo más hacia la izquierda del árbol mario es el izq del binario. Los restantes hijos del m-ario (hermanos del izquierdo), se implementan como una cadena de hijos derechos del primer hijo.

Puede verse gráficamente:

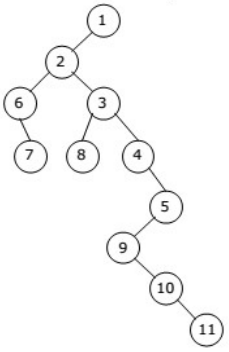


Solution:

Stage 1 tree by using the above mentioned procedure is as follows:



Stage 2 tree by using the above mentioned procedure is as follows:



Puede verse que los 4 hijos de 1, se distribuyen de la siguiente forma: el nodo 2 es hijo izquierdo del nodo 1, los restantes 3 nodos forman una cadena de hijos derechos sobre el nodo 1. De igual forma, el nodo 9 es hijo izq del 5 y los restantes hijos de 5, forman una cadena como derechos sobre 9.

Deberá implementar de esta forma un árbol m-ario sobre un árbol binario. Sobre esta implementación la inserción de hijos (hermanos) deberá ser de forma tal que permita reordenarlos una vez cargada toda la información y alternativamente cargarlos en forma ordenada, es decir, cada nuevo nodo que sea hermano de otro (insertado sobre la cadena de la rama derecha), debe quedar insertado en forma ordenada creciente respecto de algún valor que tenga dicho objeto (nombre, país, fecha nacimiento, etc).

Deberá implementar un recorrido pre-orden en el m-ario para poder listar el resultado de la carga y ordenamiento, de forma tal que si el ordenamiento fue realizado por fecha de nacimiento, ese sea el orden en que figuren los datos. Una estructura de pila puede ser de ayuda para esta funcionalidad.

Desarrollo

Metodo main

Primeramente el método principal contiene un menú para el usuario. Imprime las distintas opciones para que el usuario decida qué operación realizar con el archivo JSON.

El usuario puede elegir en un primer momento cargar los datos en el árbol, ya sea en el orden por defecto (como se encuentran en el JSON) o de manera ordenada. Si el usuario elige hacerlo de forma ordenada, el programa le solicitará el criterio de ordenamiento. Luego, el usuario podrá elegir mostrar los resultados cargados (o cargados de forma ordenada), reordenarlos por algún criterio, o finalizar la ejecución.

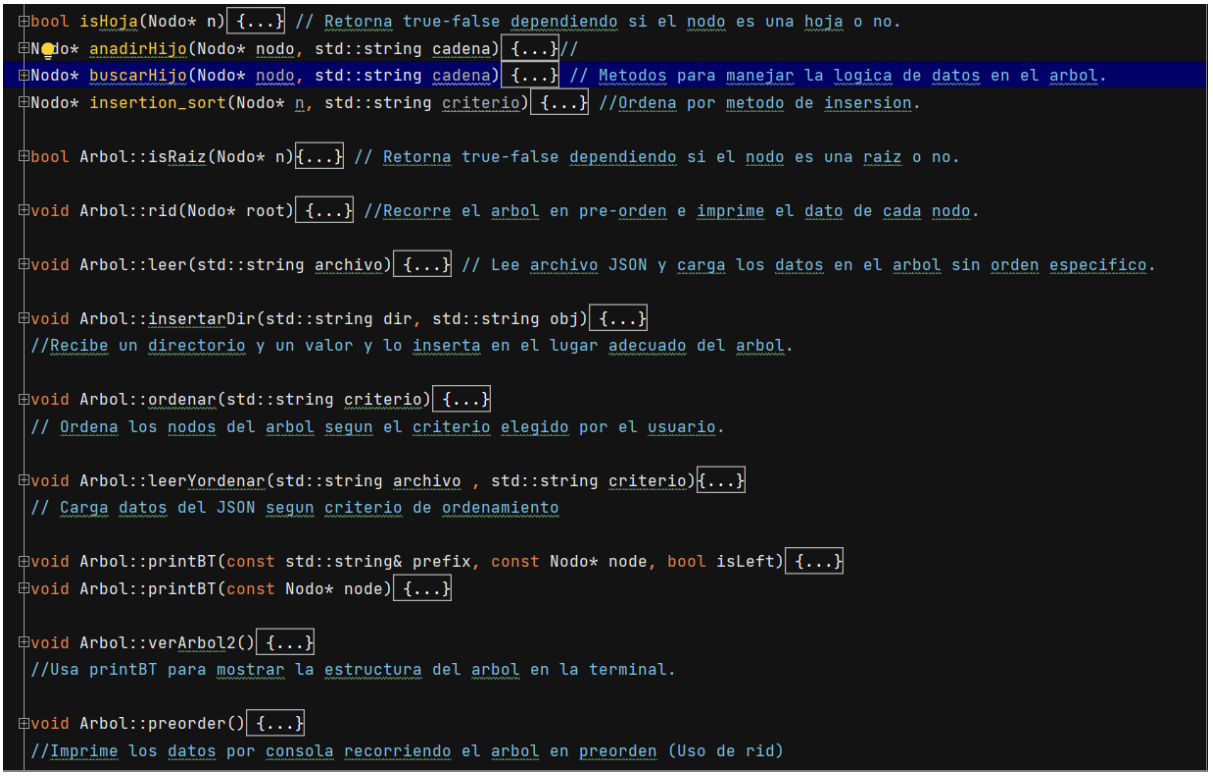
```
int main()
{
    int eleccion;
    int eleccion2;
    std::string ordenamiento;
    Arbol arbol;
    std::cout << "Elija como desea cargar los datos" << std::endl;
    std::cout << "Ingrese 1 para cargar datos" << std::endl;
    std::cout << "Ingrese 2 para cargar datos de manera ordenada" << std::endl;
    std::cin >> eleccion;
    if(eleccion == 1){
        std::cout << "Cargando datos..." << std::endl;
        arbol.leer("completo.json");
        std::cout << "Carga finalizada" << std::endl;
    }
    else{
        std::cout << "Ingrese el criterio de ordenamiento" << std::endl;
        std::cin >> ordenamiento;
        std::cout << "Cargando datos ordenadamente..." << std::endl;
        arbol.leerYordenar("completo.json",ordenamiento);
        std::cout << "Carga finalizada" << std::endl;
    }
```

```
while(true){
    std::cout << "Seleccione como continuar" << std::endl;
    if(eleccion==1){
        std::cout << "Ingrese 1 para ordenar los datos por algun criterio" << std::endl;
        eleccion++;
    }
    else{
        std::cout << "Ingrese 1 para reordenar los datos por algun criterio" << std::endl;
    }
    std::cout << "Ingrese 2 para mostrar los datos" << std::endl;
    std::cout << "Ingrese 3 para finalizar ejecucion" << std::endl;
    std::cin >> eleccion2;
    if(eleccion2 == 1){
        std::cout << "Ingrese el criterio" << std::endl;
        std::cin >> ordenamiento;
        std::cout << "Ordenando..." << std::endl;
        arbol.ordenar(ordenamiento);
        std::cout << "Ordenamiento finalizado" << std::endl;

    }
    else if (eleccion2==2){
        arbol.preorder();
    }
    else{
        break;
    }
}
```

Clase Árbol

En esta clase se encuentran todos los métodos que se encargan de la carga y ordenamiento de los datos del JSON, en la estructura, es decir el arbol m-ario implementado sobre un árbol binario, como se menciona en la consigna.



```
bool isHoja(Nodo* n) {...} // Retorna true-false dependiendo si el nodo es una hoja o no.
Nodo* anadirHijo(Nodo* nodo, std::string cadena) {...}//
Nodo* buscarHijo(Nodo* nodo, std::string cadena) {...} // Metodos para manejar la logica de datos en el arbol.
Nodo* insertion_sort(Nodo* n, std::string criterio) {...} //Ordena por metodo de insersion.

bool Arbol::isRaiz(Nodo* n){...} // Retorna true-false dependiendo si el nodo es una raiz o no.

void Arbol::rid(Nodo* root) {...} //Recorre el arbol en pre-orden e imprime el dato de cada nodo.

void Arbol::leer(std::string archivo) {...} // Lee archivo JSON y carga los datos en el arbol sin orden especifico.

void Arbol::insertarDir(std::string dir, std::string obj) {...}
 //Recibe un directorio y un valor y lo inserta en el lugar adecuado del arbol.

void Arbol::ordenar(std::string criterio) {...}
 // Ordena los nodos del arbol segun el criterio elegido por el usuario.

void Arbol::leerYordenar(std::string archivo , std::string criterio){...}
 // Carga datos del JSON segun criterio de ordenamiento

void Arbol::printBT(const std::string& prefix, const Nodo* node, bool isLeft) {...}
void Arbol::printBT(const Nodo* node) {...}

void Arbol::verArbol2() {...}
 //Usa printBT para mostrar la estructura del arbol en la terminal.

void Arbol::preorder() {...}
 //Imprime los datos por consola recorriendo el arbol en preorden (Uso de rid)
```

Conclusiones

El presente trabajo nos permitió profundizar los conocimientos sobre varios temas vistos anteriormente en clase.

En primer lugar, aprendimos a leer y cargar los datos de un archivo JSON en una estructura de tipo árbol binario, simulando así uno m-ario. Esto lo logramos utilizando la librería nlohmannjson, que nos facilitó la correcta lectura de los datos.

En segundo lugar, aprendimos a recorrer y ordenar un árbol binario (en este caso de strings), según distintos criterios aportados por el usuario que corre el programa. Para esto fue necesario aplicar nuestros conocimientos sobre algoritmos de inserción, y también profundizar en el manejo de punteros, ya que fue nuestra principal herramienta para manejar el árbol.

Restricción del programa a tener en cuenta

Una restricción importante a tener en cuenta es que el programa no valida el criterio de ordenamiento que inserta el usuario. Por esto, es imprescindible que se inserte un criterio de ordenamiento válido, para que el programa funcione correctamente.