Universidad de Costa Rica
Facultad de Ingeniería
Escuela de Computación e Informática

Estructuras de Datos y Análisis de Algoritmos
CI-1221
Grupo 04

III Etapa I TP

Tarea Programada 1:
Análisis de algoritmos y estructuras de datos.

Profesora:
Sandra Kikut

Elaborado por:
Carrion Claeys Archibald Emmanuel C01736
Arce Castillo Fernando C10572
Saltachín Solano Javier C17632

Día 19 de mes 10 del año 2022

Tabla de Contenidos Mínimos

1. Introducción

En el siguiente trabajo práctico describiremos e implementaremos distintos modelos lógicos como lo son árbol n-ario, cola y lista indexada. Con el propósito de implementar dichos modelos lógicos se utilizan distintas estructuras de datos aprendidas en clase, como lo son lista simplemente enlazada, arreglo circular, lista de hijos, estructura hijo-más-izquierdo hermano-derecho, etc.

También se incluyen cláusulas necesarias para el funcionamiento de este programa, como lo son el manual de usuario, cláusulas requiere y modifica de cada método e instrucciones para su compilación.

2. Objetivos

Objetivo general:

Comprender el funcionamiento e implementación de los modelos lógicos cola, lista y árbol n-ario.

Objetivos específicos:

- Implementar el modelo lógico cola, mediante la estructura de datos arreglo circular.
- Implementar el modelo lógico lista indexada, mediante la estructura de datos lista simplemente enlazada.
- Implementar el modelo lógico árbol n-ario, mediante las estructuras de datos arreglo con señalador al padre, lista de hijos e Hijo Más Izquierdo-Hermano Derecho.

3. Enunciado (Descripción del Problema)

1. Especificar los siguientes algoritmos para el modelo Árbol tal que No importa el orden entre los hijos de un nodo.

Para cada algoritmo debe incluir: nombre, parámetros con sus tipos y las cláusulas Efecto (claro, completo y conciso), Requiere y Modifica:

- Averiguar cuál es el hermano izquierdo de un nodo n
- Averiguar si el árbol tiene etiquetas repetidas

- Averiguar la profundidad de un nodo n (distancia que hay desde el nodo n hasta la raíz)
- Averiguar cuántos niveles tiene el árbol haciendo un recorrido en pre-orden
- Averiguar cuántos niveles tiene el árbol haciendo un recorrido por niveles
  Listar las etiquetas del i-ésimo nivel
- Listar las etiquetas del árbol en pre-orden
- Listar las etiquetas del árbol por niveles
- Buscar una etiqueta e (devuelve el primer nodo con etiqueta e encontrado o devuelve NodoNulo en caso de que la etiqueta e no esté en el árbol)
- Eliminar todo el subárbol que se genera a partir de un nodo n. Debe hacer un recorrido por niveles.

■ Se sugiere que en lugar de usar un modelo Cola, use un modelo Lista Indexada. Recuerde que el operador básico de borrado requiere que el nodo sea una hoja.

- Construir un árbol completo de i niveles y k hijos por nodo a partir de la información que contiene una Lista Indexada L, tal que L contiene las (ki – 1) / (k-1) etiquetas del árbol.

■ Las etiquetas están acomodadas en L de acuerdo al orden de un recorrido por niveles del Árbol.

2. Implementar los algoritmos del punto anterior para el modelo Árbol n-ario tal que NO importa el orden entre los hijos de un nodo. Para realizar esta implementación debe usar los operadores básicos del modelo árbol y por lo tanto la implementación de cada algoritmo deberá ser independiente de la representación o estructura de datos usada para implementar el árbol.

Algunos algoritmos requieren el uso de una Lista Indexada o una Cola, en cuyo caso se deberán usar los operadores básicos de dichos modelos.

3. Hacer un programa de prueba que permita al usuario utilizar los diferentes operadores básicos y algoritmos del modelo Árbol n-ario tal que NO importa el orden entre los hijos de un nodo. Obviamente deben incluirse algunos algoritmos sencillos que permitan generar una interfaz adecuada para que dicha prueba se pueda realizar. Considere, por ejemplo, que el usuario del árbol trabajará con etiquetas y no con nodos y por lo tanto usted deberá “traducir” etiquetas a nodos y viceversa.

Debe tomar en cuenta que, mediante la cláusula “include”, el programa de prueba deberá “escoger” la estructura de datos que se usará para representar al árbol, lo cual generará 5 códigos ejecutables diferentes (uno por cada estructura de datos usada para implementar al árbol).

4. Desarrollo

Modelo Cola: El modelo lógico Cola consiste en un grupo de datos, tal que ellos se encuentran ordenados linealmente (sin ramificaciones), de manera análoga a una fila de espera. En él solo se pueden insertar datos por un extremo, y extraer datos por el otro extremo.

Esta inserción-extracción corresponde al orden FIFO (First in First Out), es decir, que el Elemento que entró primero respecto a los demás es el primero en salir del modelo.

Definición y especificación de operadores básicos del modelo Cola.

Crear

Nombre	Crear
Parámetros	Una Cola
Efecto	Inicializa una estructura de datos para representar al modelo lógico Cola
Requiere	Una Cola que no esté inicializada
Modifica	Crea una estructura de datos

Destruir

Nombre	Destruir
Parámetros	Una Cola inicializada aún no destruida
Efecto	Borra la estructura Cola y los datos que contiene
Requiere	Una Cola inicializada

Modifica	Elimina la estructura de datos, libera el espacio utilizado por la estructura

Encolar

Nombre	Encolar
Parámetros	Una Cola, un elemento
Efecto	Agrega al inicio de la Cola el elemento dado por parámetro
Requiere	Una Cola inicializada, un Elemento válido
Modifica	La cantidad total de datos en la Cola, agrega un elemento a la Cola

Desencolar?

Nombre	Desencolar
Parámetros	Una Cola
Efecto	Devuelve un elemento almacenado en la Cola (por definición, el primero que haya entrado)
Requiere	Una Cola inicializada
Modifica	El tamaño de la Cola disminuye en una unidad y se extrae un elemento.

NumElem

Nombre	NumElem
Parámetros	Una Cola
Efecto	Devuelve un valor entero igual a la cantidad de elementos en la Cola

Requiere	Una Cola inicializada
Modifica	No modifica la estructura de datos.

Modelo Lista Indexada: Es un modelo lógico que extiende a manera de especialización a la Lista Posicionada. Consiste en un contenedor de agrupación concreta, finita, ordenada y discreta de elementos (agnósticos al tipo de dato agrupado), donde se dice que cada elemento tiene una posición ordinal en la agrupación.

Las posiciones de un elemento en una Lista Indexada tienen un índice numérico discreto concreto conocido. Para esta Lista se puede hablar de la i-ésima posición de elementos, y es consenso que el índice de aquella posición esté contenido en un rango acotado por el tamaño de la Lista.

Definición y especificación de operadores básicos del modelo Lista Indexada.

Crear

Nombre	Crear
Parámetros	Una Lista Indexada
Efecto	Inicializa una estructura de datos subyacente para representar al modelo.
Requiere	Que la Lista Indexada alimentada como parámetro esté sin inicializar
Modifica	La Lista Indexada tiene su estructura de datos subyacente inicializada.

Destruir

Nombre	Destruir
Parámetros	Una Lista Indexada inicializada

Efecto	Destruye a la estructura de datos subyacente de la Lista Indexada, liberando su espacio.
Requiere	Que la Lista Indexada alimentada como parámetro esté inicializada.
Modifica	La Lista Indexada ya no tiene a su estructura de datos subyacente inicializada, sino destruida

Insertar

Nombre	Insertar
Parámetros	● Una Lista Indexada ● Un Elemento ● Una Posición (número entero como índice de inserción)
Efecto	Inserta en la estructura de datos subyacente al elemento tal que aquel elemento toma aquella posición
Requiere	Que: ● La Lista Indexada alimentada como parámetro esté inicializada. ● El Elemento alimentado como parámetro esté inicializado ● La Posición de inserción alimentada se refiera a un índice válido para la inserción: entre el principio (índice 0) y la posición hipotética después de la última (índice igual al tamaño de la lista)
Modifica	● La Lista Indexada crece en tamaño, en 1 unidad su cantidad de elementos. ● Si existe previamente un elemento en la posición de inserción, entonces aquel elemento preexistente y todos aquellos en posiciones siguientes (si existen) incrementan en 1 unidad su índice de posición. ● Si no existe previamente un elemento en la posición de inserción, entonces el elemento adopta esa posición y no modifica la posición de ningún otro.

Borrar

Nombre	Borrar
Parámetros	● Una Lista Indexada ● Una Posición (número entero como índice de borrado)
Efecto	Elimina de la estructura de datos subyacente al elemento que posea aquella dirección.
Requiere	Que: ● La Lista Indexada alimentada como parámetro esté inicializada. ● La Lista Indexada alimentada como parámetro no esté vacía. ● La Posición de borrado alimentada se refiera a un índice válido para el borrado: entre el principio (índice 0) y el final (índice igual a una unidad menos que el tamaño de la lista)
Modifica	● La Lista Indexada disminuye en tamaño, en 1 unidad su cantidad de elementos. ● El elemento preexistente en la posición de borrado es destruido, y todos aquellos en posiciones siguientes (si existen) disminuyen en 1 unidad su índice de posición.

Recuperar?

Nombre	Recuperar
Parámetros	● Una Lista Indexada ● Una Posición (número entero como índice de recuperado)
Efecto	Obtiene los datos de un Elemento que tenga la posición especificada como parámetro de la Lista Indexada.
Requiere	Que: ● La Lista Indexada alimentada como parámetro esté inicializada. ● La Lista Indexada alimentada como parámetro no esté vacía.

	• La Posición de recuperado alimentada se refiera a un índice válido: entre el principio (índice 0) y el final (índice igual a una unidad menos que el tamaño de la lista)
Modifica	La Lista Indexada y su estructura de datos no sufren ningún cambio. Se retorna una copia del Elemento citado.

Modificar

Nombre	Modificar
Parámetros	• Una Lista Indexada • Un Elemento (para sustitución) • Una Posición (número entero como índice para modificación)
Efecto	Reemplaza en la estructura de datos subyacente al elemento preexistente en aquella posición con el nuevo elemento alimentado.
Requiere	• La Lista Indexada alimentada como parámetro esté inicializada. • El Elemento alimentado como parámetro esté inicializado • La Posición de modificación alimentada se refiera a un índice válido para la inserción: entre el principio (índice 0) y el final (índice igual a una unidad menos que el tamaño de la lista)
Modifica	Únicamente el elemento referido por la posición (basado en un índice) es modificado: sus datos son reemplazados con los datos del elemento alimentado. El resto del modelo lógico y la estructura de datos subyacentes no sufren ningún cambio.

NumElem?

Nombre	NumElem
Parámetros	• Una Lista Indexada
Efecto	Obtiene la cantidad de elementos existente en la Lista Indexada.

Requiere	• La Lista Indexada alimentada como parámetro esté inicializada.
Modifica	La Lista Indexada y su estructura de datos no sufren ningún cambio. Se retorna la cantidad de elementos existente en la Lista indexada.

Modelo Logico Arbol n-ario: El modelo lógico Árbol n-ario es un tipo de dato abstracto no lineal ya que está formado por una multitud de ramificaciones, de manera análoga a un árbol orgánico encontrado en la naturaleza.

Este modelo está conformado por Nodos y una jerarquía entre ellos. Se dice que cada Nodo, salvo la raíz (el Nodo más "arriba"), tiene solo un Nodo padre, pero cada Nodo puede tener n (cantidad arbitraria de) hijos (n-ario). Así, cada Nodo puede tener o una relación de preeminencia (padre-hijo), o de igualdad (hermanos).

Crear

Nombre	Crear
Parámetros	Un Árbol
Efecto	Crea un Árbol y le asigna un espacio
Requiere	Un Árbol que no haya sido inicializado
Modifica	Se inicializa la estructura de datos subyacente.

Destruir

Nombre	Destruir
Parámetros	Un Árbol
Efecto	Destruye la estructura de datos subyacentes del Árbol
Requiere	Un Árbol que haya sido inicializado
Modifica	La estructura del Árbol, libera el espacio usado

PonerRaíz

Nombre	PonerRaíz
Parámetros	● Un Árbol ● Un Nodo
Efecto	Asigna al Nodo que se brinda como parámetro como la raíz del Árbol.
Requiere	● Un Árbol inicializado sin una raíz. ● Un Nodo no nulo
Modifica	Se asigna el Nodo brindado como la raíz del Árbol.

AgregarHijo?

Nombre	AgregarHijo
Parámetros	● Un Árbol ● Una Etiqueta. ● Un Nodo “padre”
Efecto	Crea un Nodo que contiene la Etiqueta dada y lo agrega como hijo más izquierdo del Nodo “padre”
Requiere	● Un Árbol inicializado ● Un Nodo “padre” inicializado que hace parte del Árbol dado ● Una Etiqueta válida (que mantenga relación de tipos con los elementos del Árbol).
Modifica	Modifica la estructura del Árbol, agrega un hijo un Nodo del Árbol

AgregarHijoMásDerecho?

Nombre	AgregarHijoMásDerecho
Parámetros	● Un Árbol

	• Una Etiqueta. • Un Nodo “padre”.
Efecto	Crea un Nodo que contiene la Etiqueta dada y lo agrega como hijo más derecho del Nodo “padre”
Requiere	Una Árbol inicializado, una Etiqueta válida, un Nodo “padre” inicializado que hace parte del Árbol dado
Modifica	Modifica la estructura del Árbol, pues agrega a un Nodo nuevo como hijo del Nodo alimentado.

BorrarHoja

Nombre	BorrarHoja
Parámetros	Un Árbol, un Nodo
Efecto	Elimina una hoja de la estructura de datos
Requiere	Un Árbol inicializado, con al menos un Nodo, y que el Nodo que se brinda como parámetro sea una hoja.
Modifica	Disminuye en uno el tamaño del Árbol, elimina al Nodo que se brinda como parámetro.

Raíz?

Nombre	Raiz
Parámetros	Un Árbol
Efecto	Devuelve el Nodo correspondiente a la raíz del Árbol. Si no hay raíz devuelve un Nodo nulo.
Requiere	Un Árbol inicializado.

Modifica	No modifica la estructura de datos.

Padre?

Nombre	Padre
Parámetros	Un Árbol, un Nodo
Efecto	Devuelve un Nodo referente al padre de otro Nodo que nos brindan como parámetro, en caso de ser la raiz devuelve un Nodo nulo
Requiere	● Un Árbol inicializado, con al menos un Nodo. ● El Nodo que se pide como parámetro pertenece al Árbol
Modifica	No modifica la estructura de datos.

HijoMásIzquierdo?

Nombre	HijoMasIzquierdo
Parámetros	Un Árbol, un Nodo
Efecto	Devuelve el hijo más izquierdo de un Nodo que se pide como parámetro. Si el Nodo dado no tiene hijo, entonces devuelve un Nodo nulo
Requiere	● Un Árbol inicializado con al menos un elemento. ● Que el Nodo que se pide de parámetro pertenezca al Árbol.
Modifica	No modifica la estructura de datos.

Hermano Derecho?

Nombre	HermanoDerecho
Parámetros	Un Árbol, un Nodo
Efecto	Devuelve el hermano derecho del Nodo que se pide como parámetro. Si no existe un hermano derecho se devuelve un Nodo nulo
Requiere	● Un Árbol inicializado con al menos un Nodo ● Que el Nodo que se pide de parámetro pertenezca al Árbol
Modifica	No modifica la estructura de datos.

Etiqueta?

Nombre	Etiqueta
Parámetros	Un Árbol, un Nodo
Efecto	Devuelve la Etiqueta almacenada en un Nodo
Requiere	● Un Árbol inicializado. ● Un Nodo inicializado que existe en el Árbol dado.
Modifica	No modifica la estructura de datos.

ModificaEtiqueta

Nombre	ModificarEtiqueta
Parámetros	Un Árbol, un Nodo, una Etiqueta
Efecto	Cambia el valor de la Etiqueta de un Nodo por la Etiqueta dada
Requiere	● Un Árbol inicializado con al menos un Nodo. ● Que el Nodo tenga una Etiqueta y exista en el Árbol

Modifica	Al Nodo que se brinda como parámetro

NumNodos?

Nombre	NumNodos
Parámetros	Un Árbol, un Nodo válido
Efecto	Devuelve la cantidad de Nodos de un Árbol o subárbol. Se considera que la raíz de ese Árbol es el Nodo dado como parámetro. Si el Árbol solo tiene una raíz y ningún otro Nodo entonces devuelve 1.
Requiere	Un Árbol (o sub árbol) validado e inicializado
Modifica	No modifica la estructura de datos.

5. Manual del Usuario

5.1. Requerimientos de Hardware

Este programa fue desarrollado en una computadora con las siguientes especificaciones: CPU Intel(R) Core(TM) i3-1005G1 CPU @ 1.20GHz 1.19 GHz, con 8GB de memoria RAM.

5.2. Requerimientos de Software

Para la ejecución de este programa se requiere tener instalada un compilador de C++ (estándar 11), como lo es una versión de g++ de la 11.1.0 en adelante (la utilizada para compilar este proyecto), y un medio o ambiente para la ejecución del programa en g++.

5.3. Arquitectura del programa

El programa cuenta con una carpeta principal llamada “src” que contiene el resto de carpetas necesarias para el funcionamiento del mismo.

Estas carpetas son:

- ➔ ”1”: Que contiene los siguientes archivos, referentes a la cola implementada por arreglo circular.

- ↳ Cola.hpp

➔ ”2”: Que contiene los siguientes archivos, referentes a la lista indexada implementada por LSE.

- ↳ Celda.h
- ↳ Lista.hpp
- ↳ ListaIndexada.hpp

➔ ”3.1”: Que contiene los siguientes archivos, referentes al árbol n-ario implementado mediante arreglo con señalador al padre.

- ↳ 3.1.hpp

➔ ”3.2”: Que contiene los siguientes archivos, referentes al árbol n-ario implementado mediante lista de hijos.

- ↳ 3.2.hpp
- ↳ ArbolLH.hpp
- ↳ NodoConcreto.hpp

➔ ”3.3”: Que contiene los siguientes archivos, referentes al árbol n-ario implementado mediante hijo más izquierdo- hermano derecho con contador.

- ↳ 3.3.hpp
- ↳ Arbol.hpp

➔ ”3.4”: Que contiene los siguientes archivos, referentes al árbol n-ario implementado mediante hijo más izquierdo- hermano derecho, con puntero al padre y al hermano izquierdo, sin contador.

- ↳ 3.4.hpp
- ↳ Arbol.hpp

➔ ”3.5”: Que contiene los siguientes archivos, referentes al árbol n-ario implementado mediante hijo más izquierdo- hermano derecho, tal que el último hijo de un nodo apunta al padre sin contador..

- ↳ 3.5
- ↳ Arbol.hpp

➔ ”4”: Que contiene los siguientes archivos referentes al main, donde se ejecutan los algoritmos de los árboles.

- ↳ Controlador.hpp
- ↳ main.cpp

- ➔ ”CDE”: Que contiene los siguientes archivos referentes a una cola auxiliar doblemente enlazada.
  - ↳ Cola.hpp
- ➔ ”LSE”: Que contiene los siguientes archivos referentes a una lista simplemente enlazada.
  - ↳ Celda.h
  - ↳ Lista.h

El programa consta de un archivo “main.cpp”, el cual está encargado de mostrar la interfaz al usuario.

Para la implementación de la estructura cola se utilizan los siguientes archivos:

- “Cola.cpp”: Está encargado de implementar la cola mediante arreglo circular.

Para la implementación de la estructura lista Indexada se utilizan los siguientes archivos:

- “Celda.hpp”: Está encargado de implementar los métodos de la clase celda, estos serán necesarios para la creación de una lista simplemente enlazada.
- “Lista.hpp”: Está encargado de implementar los métodos de la clase lista, que en conjunto con “Celda.h” crean una lista simplemente enlazada.
- “ListaIndexada.cpp”: Está encargado de implementar los operadores básicos de la clase lista indexada, que en conjunto con “Celda.h” crean una lista simplemente enlazada.

Para la implementación del árbol con arreglo con señalador al padre se usan los siguientes archivos:

- “3.1.hpp”: Clase utilizada para la estandarización de nodos entre los diferentes árboles.
- “Arbol_1.cpp”: Clase utilizada para implementar los operadores basicos del arbol.

Para la implementación del arbol lista de hijos:

- “3.2.hpp”: Clase utilizada para la estandarización de nodos entre los diferentes árboles.
- “ArbolLH.hpp”: Está encargado de implementar los operadores básicos de la clase árbol, que en conjunto con “Celda.h” y “Lista.h”, crean una estructura lista de hijos.

- “NodoConcreto.hpp”: Clase utilizada como Nodo del árbol n-ario.

Para la implementación del árbol con Hijo Mas Izquierdo-Hermano Derecho con contador:

- “3.3.hpp”: Clase utilizada para la estandarización de nodos entre los diferentes árboles.
- “Arbol.hpp”: Clase encargada de implementar los operadores básicos del modelo lógico árbol, como característica importante se menciona que esta clase posee contador.

Para la implementación del árbol con Hijo Mas Izquierdo-Hermano Derecho con puntero al padre y al hermano izquierdo sin contador:

- “3.4.hpp”: Clase utilizada para la estandarización de nodos entre los diferentes árboles.
- “Arbol.hpp”: Clase encargada de implementar los operadores básicos del modelo lógico árbol, como característica importante se menciona que esta clase no posee contador y posee un señalador al padre y al hermano izquierdo.

Para la implementación del árbol con Hijo Mas Izquierdo-Hermano Derecho tal que el último hijo de un nodo apunta al padre sin contador:

- “3.5.hpp”: Clase utilizada para la estandarización de nodos entre los diferentes árboles.
- “Arbol.hpp”: Clase encargada de implementar los operadores básicos del modelo lógico árbol, como característica importante se menciona que esta clase no posee contador y el último hijo de un nodo apunta al padre.

5.4. Compilación

Para todos nuestros códigos usamos el mismo compilador: g++, versión 11.1.0.

En una terminal de cmd o Powershell en Windows, o shell de Linux, ubicada en la carpeta 4 de Tarea programada 1; ejecute el siguiente comando:

g++ *.cpp -DARBOL3_\\Numero del arbol\\ -o TEST

Aquel comando permite crear un ejecutable para el programa con el nombre TEST. El espacio \\Numero del arbol\\, debe ser llenado con el tipo de árbol que se quiera incluir (*Ver punto 5.3).

Para ejecutar este programa, se llama TEST con el siguiente comando:

.\TEST

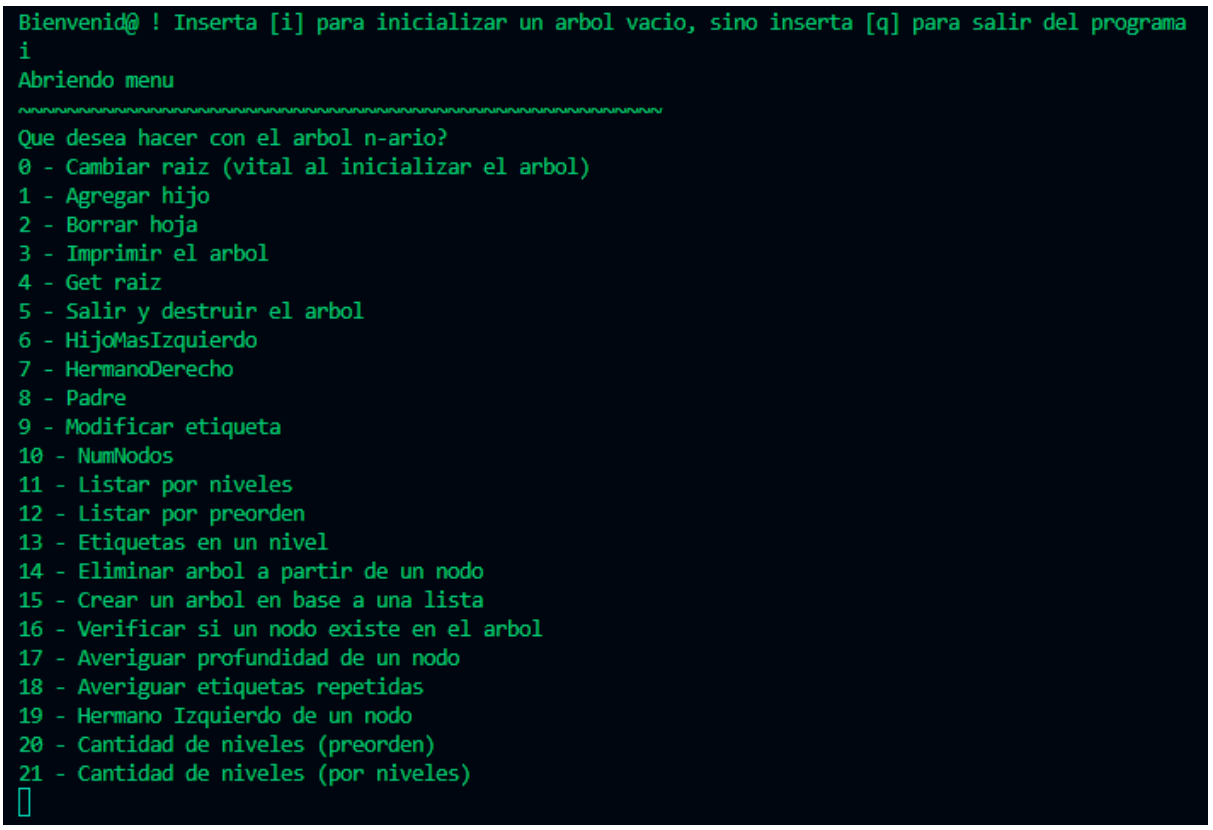
5.5. Especificación de las funciones del programa

Este programa permite implementar los operadores básicos del modelo lógico árbol (véase el punto 4), mediante distintas estructuras de datos (véase el punto 5.3). En adición dichos operadores básicos permite implementar los siguientes algoritmos:

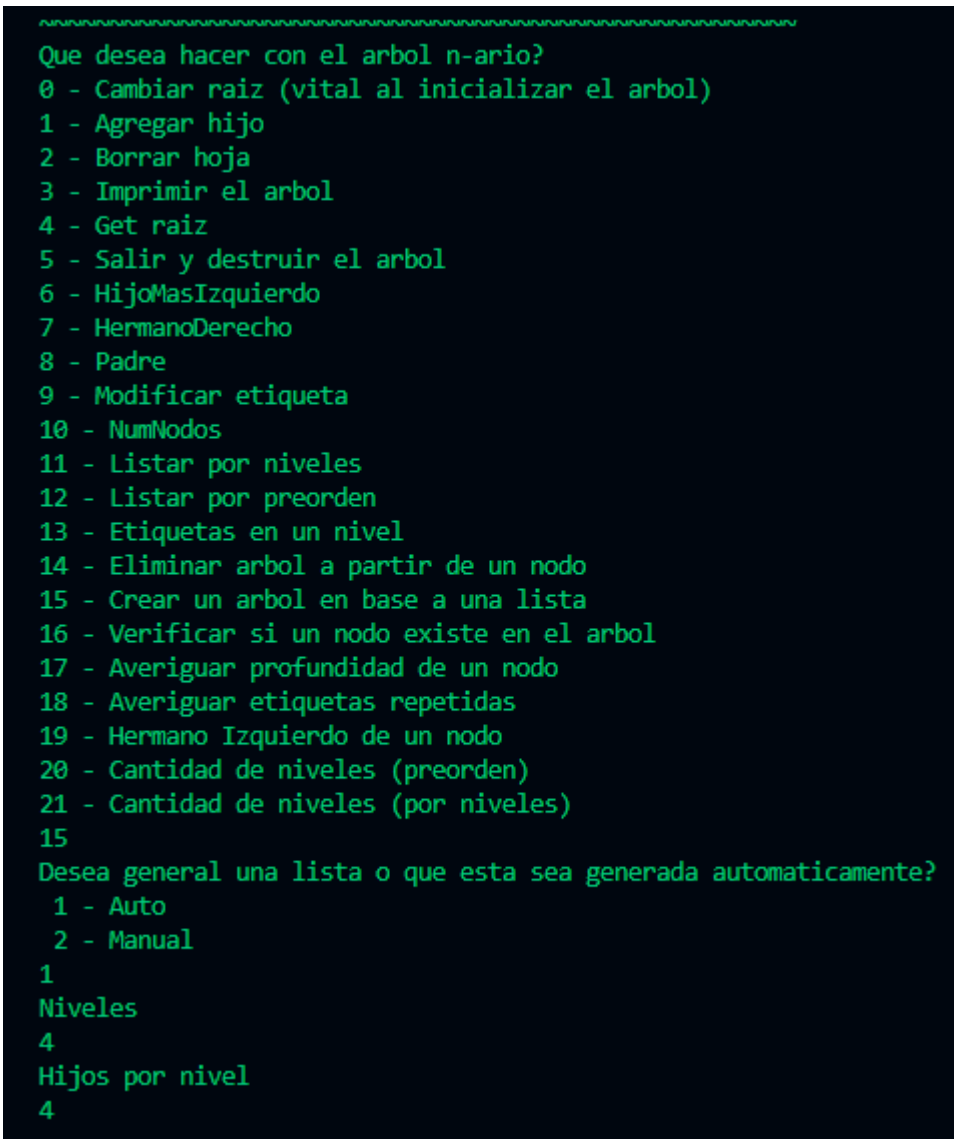
- Averiguar cuál es el hermano izquierdo de un nodo n
- Averiguar si el árbol tiene etiquetas repetidas
- Averiguar la profundidad de un nodo n (distancia que hay desde el nodo n hasta la raíz)
- Averiguar cuántos niveles tiene el árbol haciendo un recorrido en pre-orden
- Averiguar cuántos niveles tiene el árbol haciendo un recorrido por niveles
- Listar las etiquetas del i-ésimo nivel
- Listar las etiquetas del árbol en pre-orden
- Listar las etiquetas del árbol por niveles
- Buscar una etiqueta e (devuelve el primer nodo con etiqueta e encontrado o devuelve Nodo Nulo en caso de que la etiqueta e no esté en el árbol)
- Eliminar todo el subárbol que se genera a partir de un nodo n.
- Construir un árbol completo de i niveles y k hijos por nodo a partir de la información que contiene una Lista Indexada L, tal que L contiene las (ki – 1) / (k-1) etiquetas del árbol.

6. Datos de Prueba

6.1. Formato de los datos de prueba



El formato de entrada consiste en un menú, se digita la opción que quiere ejecutar, y sigue las instrucciones, por ejemplo:



Se digita como números a la opción “1”, “4” y “4”

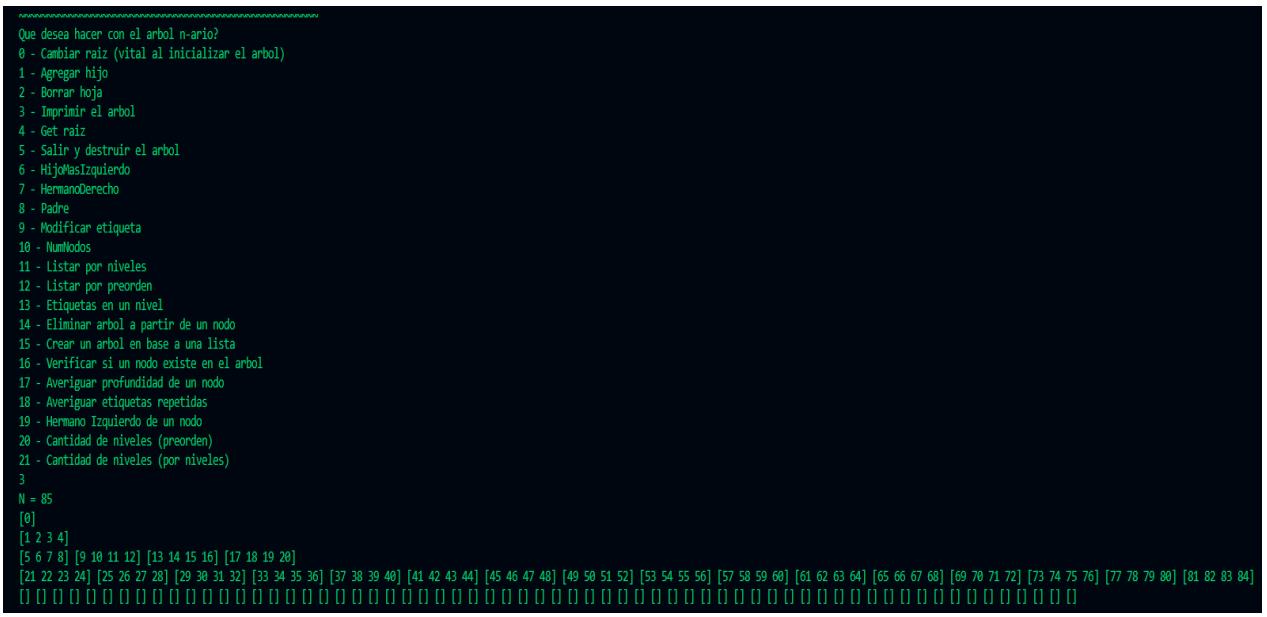
6.2. Salida esperada

Al ejecutar el código, tenemos 3 tipos de salidas esperadas:

- Menú: Da una multitud de opciones, y se escoge una de esas
- Escogencia simple: Igual que un Menú, pero tiene pocas opciones (véase imagen anterior).
- Salida normal: No pregunta entrada al usuario, puede ser solo un mensaje, o hasta imprimir el árbol

6.3. Salida obtenida (Análisis en caso de fallo)

Los varios tipos de salidos son:



Menú, recibe entrada del usuario, y devuelve el árbol impreso

8. Listado de Archivos (Estructura de las Carpetas)

code

1

Cola. hpp

2

Celda.hpp

Lista.hpp

ListaIndexada.hpp

3.1

3.1.hpp

3.2

3.2.hpp

ArbolLH.hpp

NodoConcreto.hpp

3.3

3.3.hpp

Arbol.hpp

3.4

3.4.hpp

Arbol.hpp

3.5

3.5.hpp

Arbol.hpp

4

Controlador.hpp

main.cpp

CDE

Cola.hpp

LSE

Celda.h

Lista.h