
UNIVERSIDAD DE
COSTA RICA

# UNIVERSIDAD DE COSTA RICA

## FACULTAD DE INGENIERÍA

## ESCUELA DE CIENCIAS DE LA COMPUTACION E INFORMÁTICA

## Estructuras de Datos y Análisis de Algoritmos

## CI0116

**Grupo 003**

**Profesora: Sandra Kikut**

## I Tarea Programada

**Elaborado por:**

**Esteban Quesada Quesada - B96157**

**Steven Nuñez Murillo - B95614**

**Sebastián González Varela - B93457**

**31 de octubre del 2020**

1. Introducción
2. Objetivos
3. Enunciado (Descripción del Problema)
4. Desarrollo
    4.1. Modelos
        4.1.1. Modelo 1
            4.1.1.1. Definición Pila
            4.1.1.2. Definición y especificación de operadores básicos del Modelo 1
        4.1.2. Modelo 2.
            4.1.1.1. Definición Cola
            4.1.1.2. Definición y especificación de operadores básicos del Modelo 2
        4.1.3. Modelo 3
            4.1.1.1. Definición Lista
            4.1.1.2. Definición y especificación de operadores básicos del Modelo 3
        4.1.4. Modelo 4
            4.1.1.1. Definición Árbol
            4.1.1.2. Definición y especificación de operadores básicos del Modelo 4

    4.2. Estructura de Datos
        4.2.1. Estructura de Datos 1
            4.2.1.1. Diagrama y descripción E.D. 1
            4.2.1.2. Definición en C++ de la Estructura de Datos
        4.2.2. Estructura de Datos 1
            4.2.1.1. Diagrama y descripción E.D. 1
            4.2.1.2. Definición en C++ de la Estructura de Datos
        4.2.3. Estructura de Datos 1
            4.2.1.1. Diagrama y descripción E.D. 1
            4.2.1.2. Definición en C++ de la Estructura de Datos

    4.3. Algoritmos
        4.3.1. Algoritmo 1
            4.3.1.1. Definición y especificación del algoritmo
            4.3.1.2. Descripción, detalles, pseudolenguaje (si aplica)
    4.3.1. Algoritmo 1
        4.3.1.1. Definición y especificación del algoritmo
        4.3.1.2. Descripción, detalles, pseudolenguaje (si aplica)

5. Manual del Usuario
5.1. Requerimientos de Hardware
5.2. Requerimientos de Software

5.3. Arquitectura del programa
5.4. Compilación
5.5. Especificación de las funciones del programa
6. Datos de Prueba
6.1. Formato de los datos de prueba
6.2. Salida esperada
6.3. Salida obtenida (Análisis en caso de fallo)

7. Análisis de Algoritmos
7.1. Listado y justificación de modelos, operadores y estructuras de datos a analizar.
7.2. Casos de estudio, tipos de entrada, tamaños de entrada, diseño de experimentos.
7.3. Datos encontrados, tiempos y espacio presentados en tablas y gráficos.
7.4. Análisis de los datos.
7.5. Comparación de datos reales con los teóricos.
7.6. Conclusiones con respecto al análisis realizado.
8. Listado de Archivos (Estructura de las Carpetas)
9. Referencias o Bibliografía

**Introducción:**
En el presente trabajo se definen, especifican e implementan los modelos lógicos pila,cola y Árbol n-ario tal que no importa el orden entre los hijos de un nodo, en cada modelo se analizan sus operadores. Asimismo, se presenta la implementación de una cola circular, una pila que utiliza la estructura de datos de lista simplemente enlazada, y cuatro diferentes estructuras del modelo árbol n-ario, las cuales son arreglo con señalador al padre, lista de hijos por lista simplemente enlazada(lista principal), hijo más izquierdo-hermano derecho e hijo más izquierdo-hermano derecho tal que el último hijo de un nodo apunta a su padre.
Por otra parte, se desarrollan e implementan una serie de algoritmos capaces de funcionar con cada uno de los árboles que se implementaron. Existe un menú que es capaz de utilizar cada operador básico de los árboles implementados, así como cada uno de los algoritmos que se desarrollaron para los mismos. Posteriormente se encuentra un análisis empírico, el cual toma las 4 estructuras de árboles para realizar un estudio con 2 operadores básicos y 6 algoritmos. De esta forma se compara el funcionamiento de cada operador y algoritmo según la estructura que se utilice.

**Objetivos:**
-Definir, especificar, implementar y usar los modelos lógicos Pila, Cola y Árbol n-ario tal que NO importa el orden entre los hijos de un nodo.
-Realizar un análisis teórico y un análisis real del tiempo de ejecución de las diferentes estructuras de datos y algoritmos utilizados.

**Enunciado:**
Definir, especificar e implementar modelos lógicos. Asimismo realizar una serie de análisis teóricos y un análisis del tiempo real del tiempo de ejecución de diferentes estructuras de datos, operadores y algoritmos utilizados.

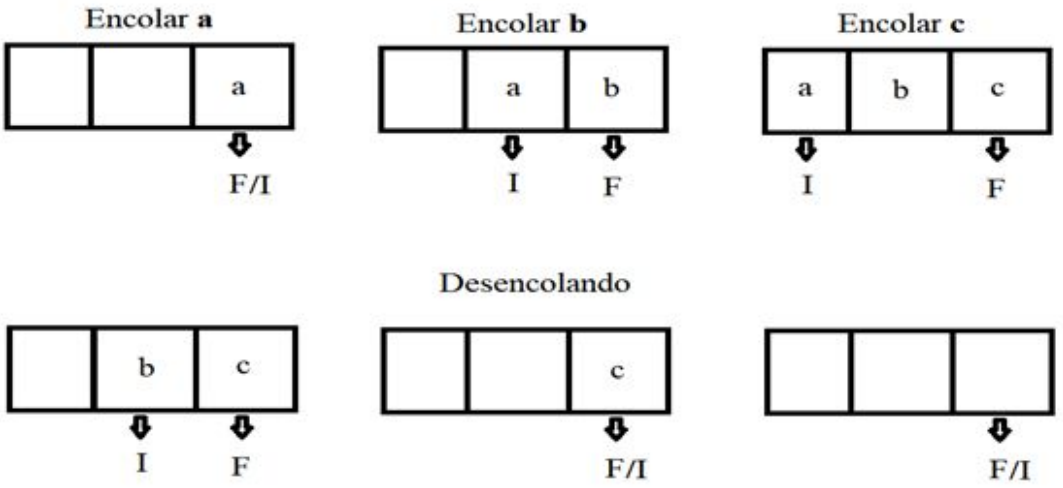
**Modelos:**
**Modelo 1 Pila**
**Definición:**Es una variante especial de una lista en la que todas las inserciones y borrados de elementos se dan por un extremo de la lista al cual se le conoce como tope,a este modelo también se le conoce como una lista LIFO(primero en entrar,primero en salir), la pila es una sucesión de elementos con relaciones de precedencia.

**Definición y especificación de operadores básicos del modelo Pila:**
• Crear
Parámetros:pila P
Efecto:inicializa P cómo pila vacía
Requiere:pila P no inicializada o P destruida
Modifica:pila P
• Destruir
Parámetros:pila P
Efecto:Destruye la pila P y la deja inutilizable
Requiere:pila P inicializada

Modifica:pila P
• Vaciar
Parámetros:pila P
Efecto:Hace que la pila P quede vacía
Requiere:pila P inicializada
Modifica:pila P
• ¿Vacía?
Parámetros:pila P
Efecto:Devuelve verdadero/ falso si la pila está vacía
Requiere: pila P inicializada
Modifica:nada
• Poner
Parámetros: pila P,elemento e
Efecto:Agrega un elemento en el tope de la pila
Requiere:pila P inicializada y elemento válido
Modifica:pila P
• Quitar
Parámetros:pila P
Efecto:Borra el elemento en el tope de la pila
Requiere:pila P inicializada y no vacía
Modifica:P
• Tope?
Parámetros:pila P
Efecto:devuelve el elemento en el tope de la pila
Requiere:P inicializada y P no vacía
Modifica:nada

**Modelo 2**

**Cola:** Este es un modelo que se encuentra basado en el modelo Lista y en el cual los elementos son insertados por solo uno de sus extremos mientras que los mismos son accesados y eliminados por el extremo contrario a este, por lo general y gracias a esa característica mencionada anteriormente es que es conocido como listas primero en entrar, primero en salir. Además de esto en las colas los agregados suelen realizarse por el final y los accesos y borrados por el inicio como se muestra en la siguiente imagen:




**2) Operadores Básicos**

**Crear(C)**

**Parámetros:** Cola denominada C.

**Efecto:** Inicializa C como una cola vacía.

**Requiere:** C no inicializada o C destruida.

**Modifica:** C.

**Destruir(C)**

**Parámetros:** Cola denominada C.

**Efecto:** Destruye C y lo deja inutilizable .

**Requiere:** C inicializada.

**Modifica:** C.

**Vaciar(C)**

**Parámetros:** Cola denominada C.

**Efecto:** Hace que C quede vacía .

**Requiere:** C inicializada.

**Modifica:** C.

**Vacía(C)**

**Parámetros:** Cola denominada C.

**Efecto:** Devuelve verdadero si C está vacía y falso si no.

**Requiere:** C inicializada.

**Modifica:** N/A.

**Agregar(e, C)**

**Parámetros:** Elemento denominado e y Cola denominada C.

**Efecto:** Agrega un elemento por la parte final de la Cola.

**Requiere:** C inicializada.

**Modifica:** C.

**Sacar(C)**

**Parámetros:** Cola denominada C.

**Efecto:** Saca y retorna el elemento que se encuentra en la parte frontal de la Cola.

**Requiere:** C inicializada y no vacía.

**Modifica:** C.

**Modelo 3**

**Lista:** Se define como una sucesión de elementos que tienen entre sí una relación de procedencia. Pueden ser indexadas, es decir tienen un índice, posteriormente se encuentra la posicional, donde la posición es una noción abstracta y por último la de tipo de elementos .Listas constituyen una estructura flexible en particular, porque pueden crecer y acortarse según se requiera, los elementos son accesibles y se pueden insertar y suprimir en cualquier posición de la lista. Las listas también pueden concatenarse entre sí o dividirse en sublistas, se presentan de manera rutinaria en aplicaciones como recuperación de información, traducción de lenguajes de programación y simulación. Matemáticamente, una lista es una secuencia de cero o más elementos de un tipo determinado(por lo general se denominará de tipo elemento).

Iniciar(L):

Parámetros: Lista L

Efecto:inicializa una lista L como lista vacía

Requiere: lista L no inicializada o destruida

Modifica:Lista L

Destruir(L):

Parámetros: Lista L

Efecto: deja lista L  inutilizable

Requiere:lista L inicializada

Modifica:lista L

Vaciar(L):

Parámetros: Lista L

Efecto:vacía lista L. No es necesario

Requiere:lista L inicializada

Modifica: lista L

Vacía(L)

Efecto:retorna verdadero si la lista L está vacía o falso si no, requiere que la lista esté inicializada.

Parámetros: Lista L

Requiere:lista L inicializada

Modifica:nada

Insertar(e,p,L)

Efecto: agrega un elemento e en la posición p de la lista L

Parámetros:elemento e,lista L

Requiere:lista L inicializada y elemento e y posición p válidos

Modifica: lista L

Borrar(e,L)

Parámetros: elemento e , lista L

Requiere:lista L inicializada y elemento e válido

Modifica: lista L

Recuperar(p,L)

Parámetros:posición p de la lista,lista L

Requiere:nodo n válido y lista L inicializada

Modifica:nada

Primera(L)

Efecto: devuelve la posición p de el primer elemento de la lista L

Parámetros:lista L

Requiere: L inicializada y primer elemento válido

Modifica: nada

AgregarAlFinal(e, L)

Efecto:agrega un elemento e al final de la lista L

Parámetros:elemento e,lista  L

Requiere:lista L inicializada y elemento e válido

Modifica: lista L

Primera(L)

Efecto: Devuelve la primera posición de la lista L

Parámetros:lista L

Requiere: lista L inicializada

Modifica:nada

Última(L)

Efecto:Devuelve la última posición de la lista L

Parámetros:lista L

Requiere:lista L inicializada

Modifica:nada

Siguiente(p,L)

Efecto:Devuelve la siguiente posición de la posición p en la lista L

Parámetros:posición p,lista L

Requiere:lista L inicializada y posición p valida

Modifica:nada

Anterior(p,L)

Efecto:Devuelve la anterior posición de la posición p en la lista L

Parámetros:posición p,lista L

Requiere:lista L inicializada y posición p válida

Modifica:nada

Modificar:

Parámetros:p, e, L

Efecto: cambia el elemento e  de la p por el elemento e de parámetro

Requiere: lista L inicializada, posición p y elemento e válidos

Modifica: elemento e de la lista L,

Intercambiar(p1,p2,L)

Parámetros:lista L, posiciones p1 y p2

Efecto:intercambia las posiciones p1,p2 de la lista L

Requiere:lista L inicializada y posiciones p1,p2 validas

Modifica:lista L

NumElem(L)

Parámetros:lista L

Efecto: devuelve la cantidad de elementos de la lista L

Requiere:lista L inicializada

Modifica:nada

**Modelo 4**

**Árbol n-ario**

Es una estructura de datos donde cada nodo posee un número indeterminado de hijos. es una estructura recursiva, y corresponde a la generalización de un árbol binario de cuyos nodos pueden desprenderse múltiples árboles binarios. las reglas que aplican a los árboles binarios pueden ser fácilmente transpoladas a los árboles n-arios así como los consejos base.

En un árbol n-ario, de grado n, cada nodo interno puede tener como máximo n nodos descendientes, y puede almacenar como máximo n-1 claves ordenadas.

Un árbol n-ario puede tomarse como un árbol de n elementos asociados a cada uno de sus componentes. se pueden encontrar 3 tipos de recorridos para este tipo de árbol:

*in order

*pre order

*post order

**Posee los mismos conceptos que un árbol binario:**

*Nodo: elementos del árbol.

*Raíz: nodo inicial del árbol.

*Hoja: nodo sin hijos.

*Camino: nodos entre dos elementos incluyéndolos.

*Rama: camino entre la raíz y una hoja.

*Altura: número de nodos en la rama más larga.

*Peso: número de nodos en el árbol.

**Orden de un elemento:**

Número de subárboles asociados.

Una hoja es un elemento de orden 0.

**Orden de un árbol n-ario:**

Máximo orden de sus elementos.

**Pregunta 6**

| Nombre | Parámetros | Efecto | Requiere | Modifica |
|---|---|---|---|---|
| Crear/Iniciar | A (tipo árbol) | Inicializa A como árbol vacío | A no inicializado o destruido. | Modifica A |
| Destruir | A (tipo árbol) | Destruye el árbol A y lo deja inutilizable | A inicializado. | Modifica A |
| Vaciar | A (tipo árbol) | Hace que A quede vacío | A inicializado | Modifica A, eliminando todo el contenido de A |
| Vacío<br>Devuelve un booleano | A (tipo árbol) | Devuelve verdadero si A está vacío y falso si A no está vacío | A inicializado | No modifica nada |
| PonerRaíz | e(tipo elemento)<br>A(tipo árbol) | Crear la raíz del árbol A. Dicha raíz tendrá etiqueta e | A inicializado y vacío | Modifica A, añadiendo una raíz al árbol vacío |
| AgregarHijo<br>Devuelve un nodo | n(tipo nodo)<br>e(tipo elemento)<br>A(tipo árbol) | Agrega un hijo con etiqueta e al nodo n. Además, devuelve el nodo creado | A inicializado y n válido en A | Modifica A, agregando un nuevo hijo |
| AgregarÚltimoHijo<br>Devuelve un nodo | n(tipo nodo)<br>e(tipo elemento)<br>A(tipo árbol) | Agrega un hijo con etiqueta e al nodo n en la última posición(la más a la derecha). Además devuelve el nodo creado | A inicializado y n válido en A | Modifica A, agregando un nuevo hijo |
| BorrarHoja | n(tipo nodo)<br>A(tipo árbol) | Borra el nodo n del árbol A | A inicializado, n válido en A y n sin hijos | Modifica A, quitándole una hoja |
| ModificaEtiqueta | n(tipo nodo)<br>e(tipo elemento)<br>A(tipo árbol) | Hace que el nodo n tenga etiqueta e, es decir, cambia la etiqueta antigua por la nueva indicada por e | A inicializado y n válido en A | Modifica A, específicamente el nodo n indicado |
| Raíz<br>Devuelve un nodo | A(tipo árbol) | Devuelve el nodo raíz de A. Si A está vacío | A inicializado | No modifica nada |

| | | | | |
|---|---|---|---|---|
| | | devuelve nodo nulo | | |
| Padre<br>Devuelve un nodo | n(tipo nodo)<br>A(tipo árbol) | Devuelve el padre de n. Si n es la raíz devuelve nodo nulo. | A inicializado y n válido en A | No modifica nada |
| HijoMásIzquierdo<br>Devuelve nodo | n(tipo nodo)<br>A(tipo árbol) | Devuelve el hijo más izquierdo(primer hijo) de n. Si n no tiene hijos, devuelve Nodo nulo. | A inicializado y n válido en A | No modifica nada |
| HermanoDerecho | n(tipo nodo)<br>A(tipo árbol) | Devuelve el hermano derecho de n. Si n no tiene hermano derecho, devuelve Nodo nulo. | A inicializado y n válido en A | No modifica nada |
| Etiqueta<br>Devuelve una etiqueta | n(tipo nodo)<br>A(tipo árbol) | Devuelve la etiqueta de n | A inicializado y n válido en A | No modifica nada |

**Estructura de Datos:**

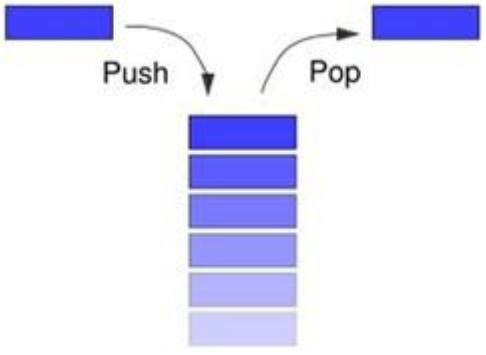
**Estructura de Datos Pila:**

Una pila, es una estructura de datos en la que el último elemento en entrar es el primero en salir, por lo que también se denominan estructuras LIFO, es decir, Last In First Out o también estructuras lineales con una política UEPS, es decir último en entrar, primero en salir.

En esta estructura sólo se tiene acceso a la cabeza o cima de la pila, también solo se pueden insertar elementos en la pila cuando esta tiene espacio y solo se pueden extraer elementos de la pila cuando tenga valores.

| Operaciones asociadas con la pila |
|---|
| Crear la pila |
| Ver si la pila está vacía |
| Insertar elementos en la pila |
| Eliminar un elemento de la pila |
| Vaciar la pila |

Representación gráfica de la operación de una pila




Las operaciones básicas en una pila son push y pop

- Push me permite insertar un elemento a la pila
- Pop extrae un elemento de la pila

La forma de implementar una pila es a través de:

· Por medio de un arreglo unidimensional

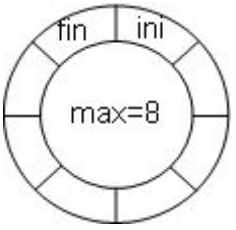
· Con una lista de elementos.

**Estructuras de datos Colas Circular:**

Una cola circular es una estructura de datos lineal que hace un uso más eficiente de la memoria disponible para su almacenamiento, sin la necesidad de requerir más espacio, utilizando el que esté desocupado. La cola se controla en forma circular, es decir, el elemento anterior al primero es el último.

Para crear una cola circular se debe tener el control de tres puntos dentro de la cola que son:

- La posición del primer elemento en la cola (inicio),
- La posición del último elemento en la cola (final) y
- El tamaño de la cola (máximo)

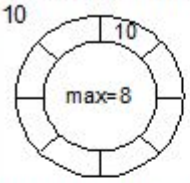
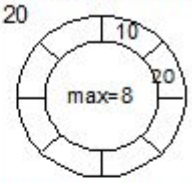
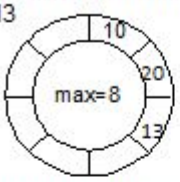
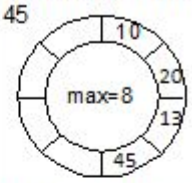
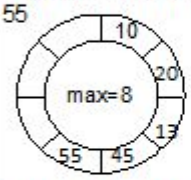
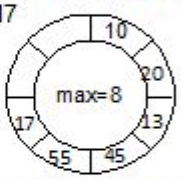
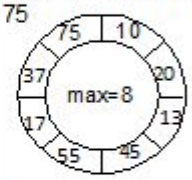



Tomando en cuenta que en una cola simple se controlan las operaciones cola vacía y cola llena, en una cola circular también se deben controlar estos dos aspectos.

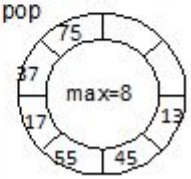
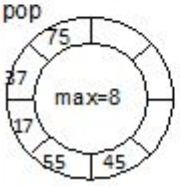
- La cola circular está vacía cuando el inicio de la cola se encuentra fuera del arreglo.
- La cola circular está llena cuando el inicio se localiza en la primera posición y el fin se encuentra una posición entes del tamaño del arreglo o cuando la posición del inicio es igual al fin más uno.

Funcionamiento:

Funcionamiento:

| | | | |
|---|---|---|---|
| Insertar el número 10   | Insertar el número 20   | Insertar el número 13   | Insertar el número 45   |
| Insertar el número 55   | Insertar el número 17   | Insertar el número 37   | Insertar el número 75   |

Si se intenta insertar otro elemento entonces marcaria cola llena.

| | | | |
|---|---|---|---|
| Sacar un valor con pop   | Sacar un valor con pop   | Sacar un valor con pop   | Insertar el número 30   |

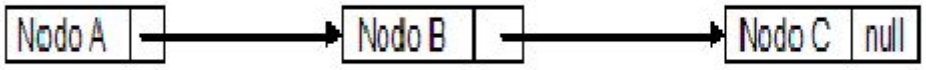
**Estructura de datos Lista Simplemente Enlazada**

Una lista enlazada simple es una colección de nodos que tienen una sola dirección y que en conjunto forman una estructura de datos lineal. Cada nodo es un objeto compuesto que guarda una referencia a un elemento (dato) y una referencia a otro nodo (dirección).

La referencia que guarda un nodo a otro nodo se puede considerar un enlace o un puntero hacia el segundo nodo y el salto que los relaciona recibe el nombre de salto de enlace o salto de puntero. El primer nodo de una lista recibe el nombre de cabeza, cabecera o primero y el último es llamado final, cola o último (es el único nodo con la referencia a otro objeto como nula).

Un nodo de una lista enlazada simple puede determinar quien se encuentra después de él pero no puede determinar quien se encuentra antes, ya que solo cuenta con la dirección del nodo siguiente pero no del anterior.

Cabeza final

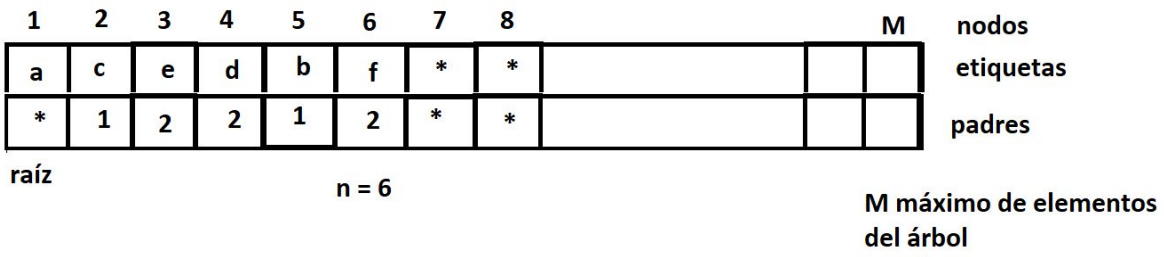



Los elementos cabeza y final de la figura son referencia creadas a partir de la clase que se crean los nodos.

Estructura de Datos:Arreglo con señalador al Padre

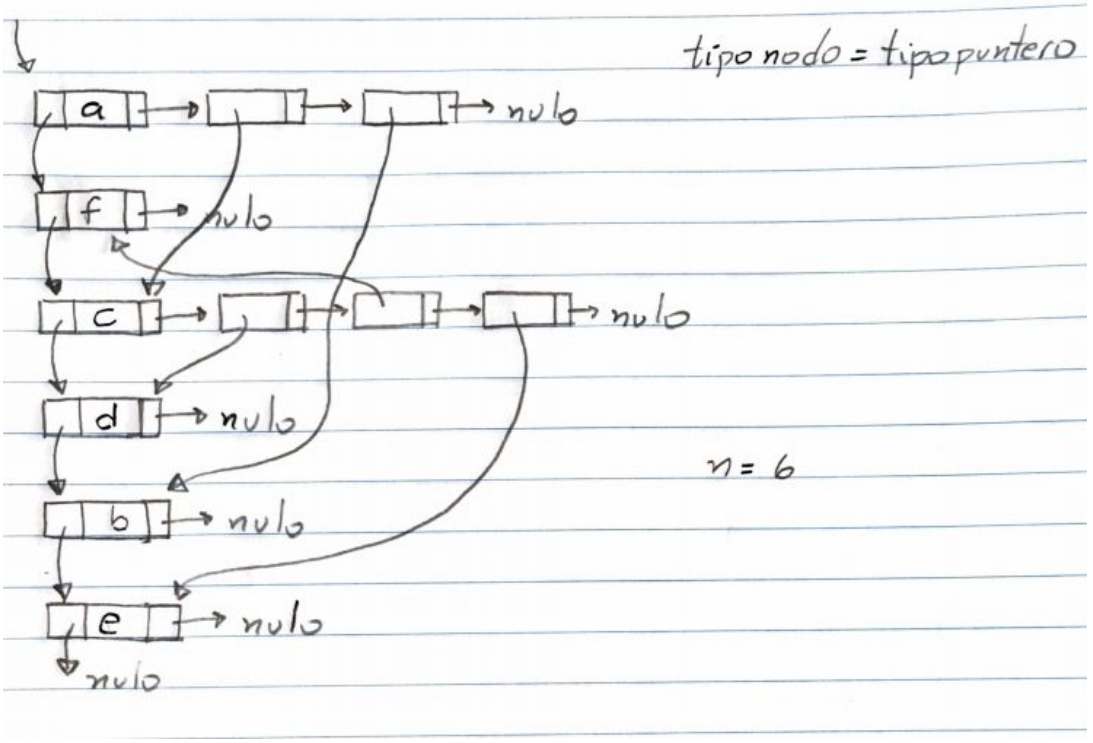
Un arreglo con señalador al padre es un conjunto de datos almacenados en memoria estática de manera continua donde los elemento se diferencian por el un índice,y se tiene un índice que indica en qué espacio del arreglo se encuentra el padre ,La raíz(nodo 1) siempre va en la primera posición. Los hijos siempre van después de su padre. Los nodos están en cualquier orden en que el usuario ha creado el árbol lógico, esto debido a que no importa el orden entre los hijos y a que el árbol se puede crear en cualquier orden, pero siempre los padres antes que los hijos. La estructura cuenta con un arreglo y un campo entero n.

Si se agrega un nuevo nodo siempre se hace en el campo n+1, en tiempo constante O(1), si se va a eliminar un nodo hoja hay que hacer un corrimiento y una reestructuración o actualización de los padres. Esta estructura de datos es iterativa mientras que el modelo es recursivo.



Estructura de Datos:Lista de Hijos

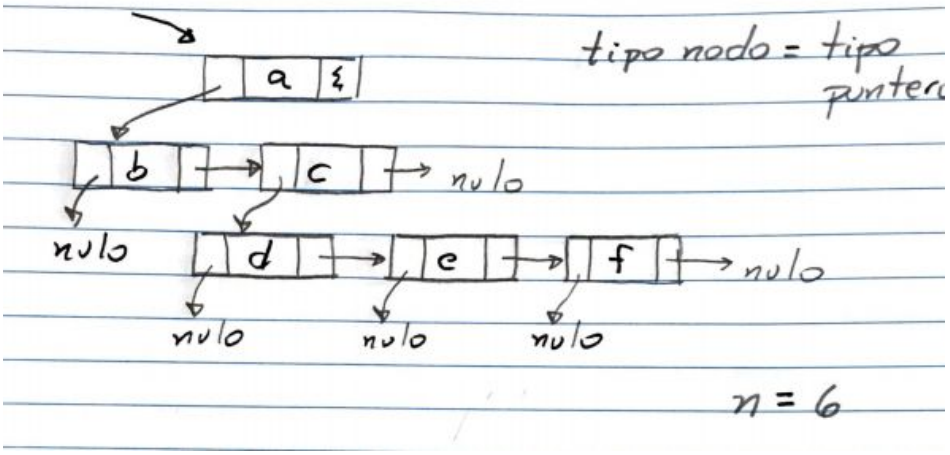
Una lista de hijos puede ser una lista simplemente enlazada,arreglo, lista doblemente enlazada, donde se hace una división entre la lista principal de elementos, y cada nodo de la lista tiene una sublista con punteros a los sus hijos que se encuentran en la lista principal,el nodo raíz tiene que estar al inicio de la lista principal y el resto de nodos pueden estar en cualquier orden,en las sublistas de hijos los hijos pueden estar en cualquier orden, se puede o no usar un contador para esta estructura, el orden de los nodos y de los hijos depende del orden en el que se creó el árbol , para esta estructura el modelo es recursivo y la estructura de datos



Estructura de Datos:Hijo Mas Izquierdo-Hermano Derecho

Esta estructura de datos se basa en espacios de memoria creados como nodos con punteros , cada nodo tiene un puntero hacia su primer hijo(hijo más izquierdo) y un puntero a su hermano derecho

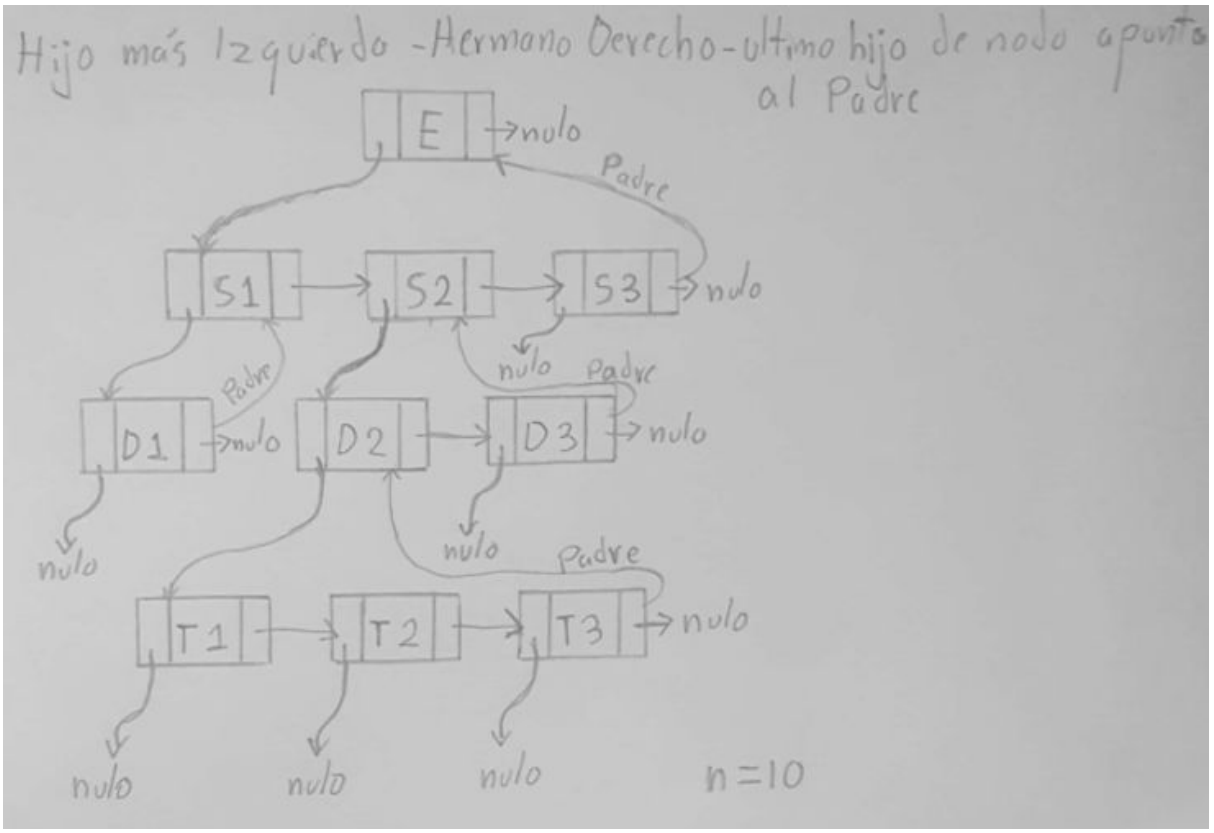
Esta estructura puede tener o no un contador, esta estructura es análoga al modelo y recursiva




Estructura de Datos:Hijo Mas Izquierdo-Hermano Derecho señalador al Padre

Esta estructura de datos se basa en espacios de memoria creados como nodos con punteros , cada nodo tiene un puntero hacia su primer hijo(hijo más izquierdo) y un puntero a su hermano derecho, si el nodo es el último hijo de un nodo(hijo msa derecho) entonces también tiene un puntero hacia su padre

Esta estructura puede tener o no un contador, esta estructura es análoga al modelo y recursiva



**Algoritmos**

| Nombre Algoritmo | Parámetros | Efecto | Requiere | Modifica |
|---|---|---|---|---|
| Hermano Izquierdo de un nodo n | n (tipo nodo)<br>A (tipo árbol) | Busca y retornar el hermano izquierdo de un nodo. | Árbol inicializado y nodo válido en el árbol. | N/A |
| Altura de un nodo | n(tipo nodo)<br>A(tipo árbol) | Retorna la altura de un nodo en un árbol. Es decir el número de descendientes de un nodo. | Árbol inicializado y nodo válido en árbol. | N/A |
| Etiquetas Repetidas | A(tipo árbol) | Retorna verdadero si existen etiquetas repetidas en el árbol y falso si no hay repetidas. | Árbol inicializado y con elementos. | N/A |
| Profundidad Nodo | n(tipo nodo)<br>A(tipo árbol) | Retorna la profundidad de un nodo, es decir la longitud de la trayectoria a su raíz . | Árbol inicializado y nodo válido. | N/A |
| Listar Etiquetas del i-ésimo nivel del árbol en Preorden | i (tipo int)<br>A(tipo árbol) | Lista todas las etiquetas de los nodos que se encuentran en el i-ésimo nivel del árbol haciendo un recorrido en preorden. | Árbol inicializado y nivel válido. | N/A |
| Listar Etiquetas del i-ésimo nivel del árbol por niveles | i (tipo int)<br>A(tipo árbol) | Lista todas las etiquetas de los nodos que se encuentran en el i-ésimo nivel del árbol | Árbol inicializado y nivel válido. | N/A |

| | | | | |
|---|---|---|---|---|
| | | haciendo un recorrido por niveles. | | |
| Buscar Etiqueta | Etiqueta (tipo elemento)<br>A (tipo árbol) | Busca una etiqueta en el árbol y retorna verdadero si se encuentra o falso si no lo está. | Árbol inicializado y con elementos. | N/A |
| Listar Árbol con recorrido en Preorden | A (tipo árbol) | Se encarga de listar el árbol haciendo un recorrido en preorden. | Árbol inicializado y con elementos. | N/A |
| Listar Árbol con recorrido por niveles | A(tipo árbol) | Se encarga de listar el árbol haciendo un recorrido en niveles. | Árbol inicializado y con elementos. | N/A |
| Copiar Árbol | Árbol 1(tipo árbol)<br>Árbol 2 (tipo árbol) | Copia el árbol 1 y lo pega en el árbol 2, creando un clon exacto del árbol 1. | Árboles inicializados y con elementos. | Modifica al árbol 2 |
| Comparar si dos árboles son iguales | A 1(tipo árbol)<br>A 2 (tipo árbol) | Compara ambos árboles para verificar si son exactamente iguales. | Árboles inicializados. | N/A |
| Recorrido en preorden simulando la recursividad con una pila | A(tipo árbol) | Se encarga de listar el árbol haciendo un recorrido en preorden, simulando la recursividad del compilador con la pila. | Árbol inicializado y con elementos. | N/A |

**Manual del Usuario**

5.1 **Requerimientos de Hardware:**
Un ordenador capaz de procesar el programa, los ordenadores actuales, todos en su mayoría son capaces de procesarlo.
5.2. **Requerimientos de Software**
C++ instalado en su ordenador
Si se trabaja en linux solo se necesita tener instalado el compilador «g++» y un editor de texto como «kate». Si no es así, desde el terminal de linux se ejecuta la instrucción:

1. apt-get install g++
2. apt.get install kate

Para crear nuestro programa ejecutaremos el editor:

1. kate nombreprograma.cc

Para usar el compilador solo se debe usar la siguiente instrucción:

g++ nombreprograma.cc -o nombreprograma.x

Una vez compilado ejecutamos el programa usando la instrucción:

./nombreprograma.x

5.3. Arquitectura del programa
En primer lugar se cuenta con una pila, la cual ha sido implementada con todos los operadores básicos, la cual se encuentra implementada en un .h y un .cpp:

1. Pila.h
2. Pila.cpp

En otras palabras esta pila es una representación del modelo visto en clase, su diseño se realiza siguiendo las cláusulas vistas en clase del modelo pila. Esta clase es necesaria para la utilización en el main. para desarrollar el algoritmo de realizar un recorrido en preorden simulando la recursividad del compilador.
No tiene ninguna interfaz, ni accede a otros archivos como .txt para su funcionamiento.

Cola circular, es una estructura de datos implementada en un único .cpp (Cola Circular.cpp) la cual no se comunica con ninguna otra implementación. No necesita acceder a ninguna otra clase para que esta misma funcione. Esta clase no cuenta con ninguna interfaz para su funcionamiento. Su diseño está basado en la cola circular vista en clase. Se desarrollan los operadores básicos de una cola y se lidia con el aspecto de ser circular, evitando ambigüedades de si la cola está llena o no, usando un campo entero encargado de decir cuántos elementos posee la cola en determinado momento.

Asimismo, se cuenta con una implementación de una cola no circular en dos archivos, un .h y un .cpp, los cuales se denominan Cola.h y Cola.cpp, estos se comunican con el main para probar los árboles, debido a que se utilizan para los algoritmos implementados, un ejemplo de un algoritmo que utiliza esta implementación es el RecorrdoPorNiveles. Esta colas no cuentan con ninguna interfaz. Asimismo su diseño está basado en la estructura Cola vista en clase con sus operadores más básicos y necesarios.

El programa principal de los árboles está compuesto por una clase main la cual contiene todos los algoritmos de los árboles, asimismo, posee el menú o la clase prueba que utiliza el usuario para poder utilizar las diferentes estructuras de los árboles. Cada árbol está implementado en un .h aparte, es decir:

1. arregloArbol.h
2. arbolLista.h
3. arboln-arioC.h
4. arboln-arioC.h

Asimismo, el programa cuenta con una

5.4. Compilación

Para la ejecución de debe tomar en cuenta que aunque después del compilado se muestran mensaje de precaución pero esto no son más que advertencias por la implementación, después del compilado se puede llamar al ejecutable normalmente y funciona correctamente

1. Para compilar el programa se debe utilizar el comando:
   g++ -o prueba main.cpp
2. Para ejecutar el programa solamente en el cmd se escribe el nombre del ejecutable sin ningún parámetro, es decir: prueba
3. Si se desea utilizar un determinado árbol en el programa principal, es decir en el main.cpp, solamente se debe acceder a esta clase e incluir el árbol que se desea usar y comentar los demás árboles. Una vez se cambia en el .cpp se compila de nuevo el programa, y se ejecuta para que en el menú se esté usando el árbol correcto.

5.5. Especificación de las funciones del programa

El programa cuenta con un menú principal con múltiples funciones para el

Opción 0:Salir
Ingreso Datos: Ninguno
Salida: Se sale del programa y termina la ejecución
Restricciones:Ninguna

Opción 1:Iniciar Árbol
Ingreso Datos:Ninguno

Salida:Inicializa un árbol
Restricciones:el árbol no debe estar previamente inicializado

Opción 2:Destruir Árbol
Ingreso Datos: ninguno
Salida:se destruye el árbol
Restricciones:el árbol debe estar previamente inicializado

Opción 3:Vaciar Árbol
Ingreso Datos:ninguno
Salida: el árbol queda vacío,sin datos
Restricciones:el árbol, debe estar previamente inicializado

Opción 4:Preguntar Vacío
Ingreso Datos:ninguno
Salida: muestra un verdadero o falso
Restricciones:el árbol debe estar previamente inicializado

Opción 5:Poner Raiz
Ingreso Datos: Por consola se debe ingresar la etiqueta de la raiz del arbol
Salida: ninguna
Restricciones: el árbol debe estar previamente inicializado y sin una raíz creada

Opción 6:Agregar Hijo
Ingreso Datos:Por consola se debe ingresar la etiqueta del nuevo hijo y la etiqueta del nodo al cual se le desea agregar un hijo
Salida: ninguna
Restricciones:El árbol debe estar previamente inicializado, y tanto la etiqueta del hijo como la etiqueta del nodo al que se le agrega el hijo deben ser válidas

Opción 7:Agregar Último Hijo
Ingreso Datos:Por consola se debe ingresar la etiqueta del nuevo hijo y la etiqueta del nodo al cual se le desea agregar un ultimo hijo
Salida:
Restricciones:El árbol debe estar previamente inicializado y tanto la etiqueta del hijo como la etiqueta  nodo al que se le agrega el hijo deben ser válidas

Opción 8:Borrar Nodo Hoja
Ingreso Datos: Por consola se debe ingresar la etiqueta del nodo que se desea borrar del árbol
Salida:ninguna
Restricciones:El árbol debe estar previamente inicializado

Opción 9:Modificar Etiqueta

Ingreso Datos:Por consola se debe ingresar la etiqueta del nodo al que se le desea modificar la etiqueta y se debe ingresar la nueva etiqueta
Salida:ninguna
Restricciones:El árbol debe estar previamente inicializado, y tanto la etiqueta del nodo que se quiere modificar y la nueva etiqueta deben de ser válidas

Opción 10:Hijo Más Izq
Ingreso Datos:por consola se debe ingresar la etiqueta del nodo al cual se quiere averiguar el hijo mas izquierdo
Salida: Etiqueta del hijo mas izquierdo del nodo
Restricciones:El árbol debe estar previamente inicializado y la etiqueta del nodo debe ser válida

Opción 11:Hermano Derecho
Ingreso Datos:por consola se debe ingresar la etiqueta del nodo al cual se quiere averiguar el hermano derecho
Salida:etiqueta del hermano derecho del nodo
Restricciones:El árbol debe estar previamente inicializado y la etiqueta del nodo que se quiere averiguar el hermano derecho debe ser válida

Opción 12:Raiz
Ingreso Datos:ninguno
Salida: etiqueta de la raíz
Restricciones:El árbol debe estar previamente inicializado

Opción 13:Padre
Ingreso Datos:por consola se debe ingresar la etiqueta del nodo al cual se desea averiguar el nodo padre
Salida:etiqueta del padre del nodo, si es la raíz la salida es un null
Restricciones:El árbol debe estar previamente inicializado y la etiqueta del nodo debe ser válida

Opción 14:Núm Hijos
Ingreso Datos:etiqueta del nodo al cual se desea averiguar la cantidad de hijos
Salida: cantidad de hijos que posee el nodo(int)
Restricciones:El árbol debe estar previamente inicializado y la etiqueta del nodo debe ser válida

Opción 15: EsHoja
Ingreso Datos: por consola se debe ingresar la etiqueta del nodo que se desea averiguar si es una hoja
Salida: verdadero o falso si el nodo es hoja
Restricciones:El árbol debe estar previamente inicializado y la etiqueta del nodo debe ser válida

Opción 16:Núm Nodos
Ingreso Datos:por consola se debe ingresar la etiqueta del nodo que se desea averiguar la cantidad de hijos que tiene
Salida: cantidad de hijos del nodo(int)
Restricciones:El árbol debe estar previamente inicializado y la etiqueta del nodo debe ser válida

Opción 17:Imprimir Árbol Por Niveles
Ingreso Datos:ninguno
Salida: etiquetas del árbol dado por un recorrido por niveles
Restricciones:El árbol debe estar previamente inicializado

Opción 18:Menú de Algoritmos
Ingreso Datos:ninguno
Salida:despliega un submenú
Restricciones: ninguna

Esta última opción despliega un submenú en el cual se tienen las siguientes opciones
Opción 0:Salir
Ingreso Datos:ninguno
Salida: Vuelve al menú principal
Restricciones: Ninguna

Opción 1: Hermanos Izquierdo Nodo
Ingreso Datos: Por consola se debe ingresar la etiqueta del nodo que se desea conocer su hermano izquierdo
Salida: La etiqueta del nodo Hermano Izquierdo
Restricciones:Árbol previamente inicializado y etiqueta del nodo válida

Opción 2: Etiquetas Repetidas
Ingreso Datos:ninguno
Salida: verdadero o falso si el árbol tiene al menos 1 etiqueta repetida
Restricciones:árbol previamente inicializado

Opción 3: Altura Nodo
Ingreso Datos:por consola se debe ingresar la etiqueta del nodo el cual se desea saber su altura
Salida: numero int de la altura del nodo
Restricciones: árbol previamente inicializado y la etiqueta del nodo debe ser válida

Opción 4: Profundidad Nodo

Ingreso Datos:por consola se debe ingresar la etiqueta del nodo el cual se desea saber su profundidad
Salida: numero int de la profundidad del nodo
Restricciones:árbol previamente inicializado y la etiqueta del nodo debe ser válida

Opción 5: Listar Etiquetas en Preorden
Ingreso Datos:ninguno
Salida:etiquetas del árbol ordenadas según el recorrido en preorden
Restricciones: árbol previamente inicializado

Opción 6:Listar Etiquetas Por Niveles
Ingreso Datos:ninguno
Salida:etiquetas del árbol ordenadas según el recorrido por niveles
Restricciones:árbol previamente inicializado

Opción 7:Copiar Árbol
Ingreso Datos: ninguno
Salida: etiquetas de ambos árboles
Restricciones: árbol previamente inicializado

Opción 8: Árboles Iguales
Ingreso Datos:ninguno
Salida:verdadero o falso si los 2 árboles que se tienen en el programa son iguales
Restricciones:deben haber 2 árboles previamente inicializados

Opción 9:Listar Pre Orden
Ingreso Datos:ninguno
Salida: etiquetas del árbol según un recorrido en pre orden
Restricciones:árbol previamente inicializado

Opción 10:Pila PreOrden(no implementado)
Ingreso Datos:ninguno
Salida:ninguna
Restricciones:no usar

Opción 11:ListarPorNiveles
Ingreso Datos:ninguno
Salida: etiquetas del árbol según un recorrido por niveles
Restricciones:árbol previamente inicializado

Opción 12:Buscar Existencia Etiqueta
Ingreso Datos:por consola se debe ingresar la etiqueta del nodo que se quiere saber si esta en el árbol
Salida:verdadero o falso , dependiendo si existe un nodo con la etiqueta
Restricciones:árbol previamente inicializado y etiqueta del nodo válida

6.1. Formato de los datos de prueba
Por la implementación hecha con templates los árboles admiten diferentes tipos de datos, por un problema del propio menú los strings no son admitidos para agregarlos al árbol desde el menú aunque los árboles si los admite, para los diferentes casos de prueba se usaron los tipos de dato de C++ integers y chars
6.2. Salida esperada
Para el primer operador que se usó como prueba que es Agregar Hijo aunque al agregar un hijo se retorna el hijo agregado, en pantalla no existe una salida
Para el segundo operador que se usó como prueba que es Padre las salidas son las etiquetas del padre de cada nodo y nulo en caso de que el nodo probado sea la raíz la salida esperada es nulo
Para los algoritmos las salidas esperadas son para algunos algoritmos no se da una salida en pantalla, y los que sí tienen salida por pantalla son tanto salidas de verdaderos y falsos, como también salidas de etiquetas
Las salidas en general son las deseadas excepto que para la estructura lista de hijos hay un problema con el último nodo,aunque los operadores sirven bien y algoritmos que no usen este nodo

7. Análisis de Algoritmos
7.1. Listado y justificación de modelos, operadores y estructuras de datos a analizar.
Estructuras de datos:
Arreglo con señalador al padre:dado que esta estructura está implementado por un arreglo en C++ esto mejora el tiempo en la ejecución de la mayoría de algoritmos para esta estructura
Lista de hijos: tiene constantes de proporcionalidad altos que sirven para comparar y observar tiempo de ejecución altos, debido a un problema con el último nodo de esta estructura se dificulto la ejecución de algoritmos para esta estructura, aunque la ejecución de los operadores funciona correctamente.
Hijo mas Izquierdo-Hermano derecho:
Hijo mas Izquierdo-Hermano derecho con señalador al padre:dado el puntero al padre la estructura de datos se vuelve más fácil de utilizar por lo que
Operadores:
Agregar Hijo: este operador fue elegido para el análisis dado que según la teoría para las estructuras de datos implementadas el tiempo de este algoritmo es de O(1) por lo que queríamos verificar que para todos las estructuras el tiempo de los procesos son similares

Padre:este operador fue elegido dado que para la estructura del arreglo el tiempo es O(1),para la estructura hijo mas izquierdo-hermano derecho(con señalador)depende de a cual nodo se refiera, para la estructura hijo mas izquierdo-hermano derecho el tiempo es 0(n) igual que para la estructura de listas, en el peor caso de la estructura de la lista tarda O(2), por esta variabilidad se decidió usar este operador

Modelos:

Árbol-nario

7.2. Casos de estudio, tipos de entrada, tamaños de entrada, diseño de experimentos.

7.3. Datos encontrados, tiempos y espacio presentados en tablas y gráficos.

| Caso de Estudio | Tipo de Entrada | Tamaños de Entrada | Diseño de Experimentos | Estructura de Datos utilizada | Tiempo de duración | Promedio Cantidad de tiempo entre el tamaño de entrada |
|---|---|---|---|---|---|---|
| Operador Agregar Hijo | Se utilizan árboles con elementos de tipo integers | Se efectúan 10000 agregados de hijos | Se realiza una función en el main que se encarga de procesar 10000 veces la operación mencionada. | 1)arregloArbol<br>2)arbolLista<br>3)arboln-arioC<br>4)arboln-arioD | 1)999 microsegundos<br>2)997 microsegundos<br>3) 999 microsegundos<br>4) 1022 microsegundos | 1)0.0999 microsegundos/10 000<br>2) 0.0997 microsegundos/10 000<br>3) 0.0999 microsegundos/10 000<br>4) 0.1 microsegundos/10 000 |
| Operador Padre | Se utilizan árboles con elementos de tipo integers | Se efectúan 10000 búsquedas de Padre | Se realiza una función en el main que se encarga de calcular el padre para cada uno de los 10000 nodos que posee el árbol utilizado. | 1)arregloArbol<br>2)arbolLista<br>3)arboln-arioC<br>4)arboln-arioD | 1)1001 microsegundos<br>2) 960407 microsegundos<br>3) 381140 microsegundos<br>4) 308410 | 1)0.1001 microsegundos/10 000<br>2) 96.0407 microsegundos/10 000<br>3) 38.114 microsegundos/10 000 |

| | | | | | | |
|---|---|---|---|---|---|---|
| | | | | | microsegundos | 4) 30.85 microsegundos/10 000 |
| ListarPorNiveles | Se utilizan árboles con elementos de tipo integers | Se efectúa con 10000 nodos. | Se realiza una función en el main que se encarga de realizar un recorrido por niveles. | 1)arregloArbol<br>2)arboln-arioC<br>3)arboln-arioD | 1)60310000<br>2)59560000 microsegundos<br>3) 69810000microsegundos | 1)603.1microsegundos/10 000<br>2) 595.6 microsegundos/10 000<br>3) 698.1 microsegundos/10 000 |
| ListarPreorden | Se utilizan árboles con elementos de tipo integer | Se efectúa con 10000 nodos. | Se realiza una función en el main que se encarga de realizar un recorrido en preorden. | 1)arregloArbol<br>2)arboln-arioC<br>3)arboln-arioD | 1)39910000<br>2)59560000 microsegundos<br>3) 69810000microsegundos | 1)399.1 microsegundos/10 000<br>2) 595.6 microsegundos/10 000<br>3) 698.1 microsegundos/10 000 |
| AlturaNodo | Se utilizan árboles con elementos de tipo integer | Se efectúa con 10000 nodos. | Se realiza una función en el main que se encarga de calcular la altura de un nodo en específico, se usa la raíz como nodo. | 1)arregloArbol<br>2)arboln-arioC<br>3)arboln-arioD | 1)8976000microsegundos<br>2)19950000 microsegundos<br>3)39770000 microsegundos | 1)897.6microsegundos/10000<br>2) 199.5 microsegundos/10 000<br>3)397.7 microsegundos/10 000 |
| Copiar Árbol | Se utilizan árboles con elementos de tipo integer | Se efectúa con 10000 nodos. | Se realiza una función en el main que se encarga de | 1)arregloArbol<br>2)arboln-arioC | 1)1001 microsegundos<br>2)999 | 1)100.1 microsegundos/10000 |

| | | | | | | |
|---|---|---|---|---|---|---|
| | | | copiar un árbol denominado árbol en un arbol2 | 3)arboln-arioD | microsegundos 3)998 microsegundos | 2) 0.0999 microsegundos/10 000<br><br>3)0.0998 microsegundos/10 000 |
| Profundidad de nodo | Se utilizan árboles con elementos de tipo integer | Se efectúa con 10000 nodos. | Se realiza una función en el main que se encarga de calcular la profundidad de un nodo en específico. | 1)arregloArbol<br><br>2)arboln-arioC<br><br>3)arboln-arioD | 1)1098 microsegundos<br><br>2) 4087 microsegundos<br>3) 3990 microsegundos | 1)0.01098microsegundos/10000<br><br>2) 0.04087 microsegundos/10 000<br><br>3)0.0399 microsegundos/10 000 |
| Profundidad de nodo | Se utilizan árboles con elementos de tipo integer | Se efectúa con 10000 nodos. | Se realiza una función en el main que se encarga de calcular la profundidad de un nodo en específico. | 1)arregloArbol<br><br>2)arboln-arioC<br><br>3)arboln-arioD | 1)1098 microsegundos<br><br>2) 4087 microsegundos<br>3) 3990 microsegundos | 1)0.01098microsegundos/10000<br><br>2) 0.04087 microsegundos/10 000<br>3)0.0399 microsegundos/10 000 |

7.4. Análisis de los datos.
7.5. Comparación de datos reales con los teóricos.
7.6. Conclusiones con respecto al análisis realizado.

**Comparación, Análisis y conclusiones de datos reales con los teóricos:**
**Operador Agregar Hijo:** Como se puede observar en las tablas de datos este operador tiene el mismo tiempo de ejecución para las distintas estructuras de árboles, ya que todas cuentan con Agregar Hijo en tiempo constante. Si comparamos los datos con las tablas vistas en clase esto es verdadero, en todas las estructuras los AgregarHijo tardan tiempo constante. Por lo

tanto, como conclusión se puede observar como el operador AgregarHijo es sumamente rápido y no gasta mucho tiempo para ninguna estructura del árbol implementada.
**Operador Padre:** Como se puede observar en las tablas de datos este operador varía en su tiempo de ejecución, en la estructura de datos el tiempo de ejecución fue el de menor tiempo de ejecución lo que concuerda con la teoría ya que el tiempo para esta estructura es O(1), para la estructura lista de hijos se observa que es la que tiene el mayor tiempo de ejecución, lo cual concuerda con la teoría donde dice que el tiempo de ejecución para esta estructura de datos es O(2n), para la estructura hijomas izq-hermanoder el tiempo de ejecución está en el medio en comparación con el arreglo que es el menor y lista de hijos que es el mayor, lo cual concuerda con la teoría donde dice que el tiempo es O(n) y para la estructura hijomas izq-hermanoder con puntero al padre se puede observar que el tiempo de ejecución disminuye en comparación con la estructura hijomas izq-hermanoder sin puntero, esto se da porque el último hijo de un nodo tiene un puntero al padre y esto disminuye el tiempo de ejecución

**Algoritmos**
**ListarPorNiveles:** En este algoritmo podemos observar según las tablas de datos que para las diferentes estructuras de datos los tiempo de ejecución no varía casi en nada dado que los tiempos de este algoritmo y su funcionalidad se basan principalmente en el uso de una cola, y de los operadores básicos HermanoDerecho que para las diferentes estructuras de datos tarda segun la teoria tiempo constante para todas las estructuras, la variabilidad que se da en los tiempos es dado que se usa el operador HijoMasIzq donde si se presenta diferencias en los tiempo de ejecución y esto concuerda con la teoría, en conclusión este algoritmo por su fácil implementación, sencillez y tiempos de duración en general O(1)tiene un tiempo de ejecución bajo similar para las diferentes estructuras de datos implementadas
**Listar Pre orden:** En este algoritmo podemos observar que según las tablas de datos los tiempos de ejecución son diferentes para las distintas estructuras de datos implementadas, pero podemos observar que según los el algoritmo se ocupa el hijomasizq el cual tiene tiempo de ejecucion grandes para algunas estructuras de datos, por eso concluimos que para este algoritmo los tiempo varían mucho
**Copiar Árbol:** Para este algoritmo podemos observar que para todas las estructuras el tiempo de ejecución son bastantes similares dado que para este algoritmo los operadores se basan en un recorrido donde los tiempos son bastantes similares para las diferentes estructuras además que el operador que se usa para agregar hijo tiene tiempo constante para las diferentes estructuras de datos por lo que se concluye que para este algoritmo el tiempo de ejecución es similar para todas las estructuras de datos.
**Profundidad de Nodo:** Este algoritmo es iterativo por lo tanto los tiempos de ejecución son más bajos que otros algoritmos, en las tablas de datos podemos observar como se tarda un tiempo reducido en la mayoría de estructuras, por lo tanto no significa, mucho gasto de recursos para ninguna estructura. Para el arreglo se tarda el tiempo más bajo mientras que para los algoritmos 2 y 3 representa un tiempo de ejecución más alto.
**Altura Nodo:** Altura nodo es un algoritmo de tipo recursivo  por eso los tiempos de ejecución varían según las estructuras de datos.

8. Listado de Archivos (Estructura de las Carpetas)

3 carpetas

1. Carpeta pila
    a. Pila.h
    b. Pila.cpp
2. Carpeta ColaCircular
    a. ColaCircular.cpp
3. Carpeta Árboles
    a. main.cpp
    b. arregloArbol.h
    c. arbolLista.h
    d. arboln-arioC.h
    e. arboln-arioD.h
    f. Cola.h
    g. Cola.cpp

**9. Referencias o Bibliografía**


AHO, V., ALFRED, H., JOHN, U., & JEFFREY. (1998). ESTRUCTURAS DE DATOS Y ALGORITMOS. MEXICO, D.F.: IMPRESIONES ALDINA, S.A.

Carl Reynolds & Paul Tymann (2008). *Schaum's Outline of Principles of Computer Science*. McGraw-Hill. ISBN 978-0-07-146051-4.

Cormen, Thomas; Leiserson, Charles; Rivest, Ronald; Stein, Clifford (2009). *Introduction to algorithms*. Cambridge, Massachusetts: The MIT Press. ISBN 978-0-262-53305-8.

Hernández, R., Lázaro, J. C., Dormido, R., & Ros, S. (2001). Estructuras de datos y Algoritmos. Prentice Hall.