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

Estructuras de Datos y Análisis de Algoritmos
CI-1221
Grupo 001

I Tarea programada

Profesora:
Sandra Kikut

Elaborado por:
Andreína Alvarado González | B40259
Otto Mena Kikut | B03843

25 de Septiembre del 2015

Indice

1 Introducción

Una de las partes fundamentales, a la hora de aprender sobre estructuras de datos y análisis de algoritmos, es entender bien como están constituidas las distintas partes que se deben tener en cuenta a la hora de hablar sobre modelos matemáticos, estructuras de datos, algoritmos, ... Esta tarea, consta de cuatro etapas cuyo fin es precisamente, tratar de comprender todas estas partes y la manera correcta de concebirlas.

Una primer etapa, se enfoca en la especificación de los modelos matemáticos. Es decir, su definición formal y el establecimiento de operadores básicos que permitan la implementación de cualquier algoritmo que se desee a partir de estos operadores básicos. Así mismo, a estos operadores se le definiran sus clásulas de efecto, requiere y modifica.

En una segunda etapa, se pretende implementar mediante estructuras de datos específicas, estos modelos con sus operadores básicos específicos a nivel computacional.

La tercer etapa, se basa en la implentación, de igual manera, a nivel computacional de algoritmos para árboles n-arios, tomando en cuenta, que estos algoritmos deben funcionar independientemente de qué estructura de datos se esté usando.

Finalmente, durante las lecciones, se han establecido ciertos órdenes de duración para algunos algoritmos y operadores básicos. En la cuarta etapa, se pretende realizar una serie de análisis teórico y de tiempo real de ejecución, con el fin de conocer, establecer y esperimentar respecto a la realidad entre la teoría y la práctica de la duración de estos algoritmos.

2 Objetivos

• Definir, especificar, implementar y usar los modelos lógicos Cola, Pila, Árbol n-ario tal que sí 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.

3 Enunciado

Para la primer etapa:

Especificar de manera lógica, formal y completa los operadores básicos de la Cola, Pila y Árbol n-ario. Para cada operador debe incluir: nombre, parámetros con sus tipos y las cláusulas Efecto (claro, completo y conciso), Requiere y Modifica.

Para la segunda etapa:

- Implementar el modelo Cola utilizando la estructura de datos: arreglo circular.
- Implementar el modelo Pila utilizando la estructura de datos: lista simplemente enlazada.
- Implementar el modelo Árbol n-ario tal que sí importa el orden entre los hijos de un nodo

utilizando las estructuras de datos: arreglo con señalador al padre; lista de hijos por lista implemente enlazada (lista principal) y lista simplemente enlazada (sublistas); hijo más izquierdo-hermano derecho por punteros; e hijo más izquierdo-hermano derecho por punteros, con puntero al padre y al hermano izquierdo.

Para la tercera etapa:

Especificar distintos algoritmos para el modelo árbol tal que sí importa el orden entre los hijos de un nodo, utilizando sus operadores básicos. Para cada algoritmo se debe especificar nombre, parámetros con sus tipos y las cláusulas efecto (claro, completo y conciso), requiere y modifica.

Además, hacer un programa de prueba de los algoritmos implementados. Este programa, deberá permitir usar los operadores básicos del árbol.

Para la cuarta etapa:

Hacer un análisis empírico (tiempo y espacio real) de la complejidad computacional de las estructuras de datos, operadores básicos y algoritmos implementados en esta tarea. Para ello, se deberá hacer cálculos de tiempo real de ejecución de los diferentes operadores y algoritmos, para diferentes tamaños de n (n muy grandes) y para diferentes tipos de árboles (diferentes alturas y anchuras). Dichos cálculos deberán ser mostrados en tablas y gráficos.

Además, se deberá comparar los cálculos reales con los teóricos e incluir una sección de conclusiones sobre la eficiencia de cada estructura de datos.

4 Desarrollo

4.1 Modelos

4.1.1 Modelo cola

4.1.1.1 Definición del modelo cola

Según el libro *Estructura de datos y algoritmos* de Alfred V. Aho, Jeffrey D. Ullman y John E. Hopcroft: una *cola* es un tipo especial de lista en el cual los elementos se insertan en un extremo *el posterior* y se suprimen en el otro *el anterior o frente*. Las colas a menudo se les conoce también como “FIFO” o lista “primero en entrar, primero en salir”.

4.1.1.2 Definición de operadores básicos

Crear(cola C):

Cláusulas de Crear(cola C)

Efecto	Inicializa la cola C como vacía.
Requiere	No aplica.
Modifica	Cola C.

Destruir(cola C):

Cláusulas de Destruir(cola C)	
Efecto	Destruye la cola C, dejándola inutilizable.
Requiere	Cola C inicializada.
Modifica	Cola C.

Vaciar(cola C):

Cláusulas de Vaciar(cola C)	
Efecto	Vacía cola C, dejándola con 0 elementos.
Requiere	Cola C inicializada.
Modifica	Cola C.

Vacía(cola C), devuelve algo de tipo booleano:

Cláusulas de Vacía(cola C)	
Efecto	Devuelve verdadero si cola C vacía y falso si no.
Requiere	Cola C inicializada.
Modifica	No aplica.

Agregar(elemento e, cola C):

Cláusulas de Agregar(elemento e, cola C)	
Efecto	Agrega un nuevo elemento en la cola C, de manera que este elemento quede en la parte de atrás de la cola.
Requiere	Cola C inicializada.
Modifica	Cola C.

Sacar(cola C):

Cláusulas de Sacar(cola C)	
Efecto	Borra el primer elemento en cola C.
Requiere	Cola C inicializada y con al menos un elemento.
Modifica	Cola C.

Frente(cola C), devuelve algo de tipo elemento:

Cláusulas de Frente(cola C)	
Efecto	Devuelve el primer elemento en cola C.

Requiere	Cola C inicializada y con al menos un elemento.
Modifica	No aplica.

4.1.2 Modelo pila

4.1.2.1 Definición del modelo pila

Según el libro *Estructura de datos y algoritmos* de Alfred V. Aho, Jeffrey D. Ullman y John E. Hopcroft: una pila es un tipo especial de lista en la que todas las inserciones y supresiones tienen lugar en un extremo denominado *tope*. A menudo a las pilas también se les conoce como "LIFO" o listas "último en entrar, primero en salir".

4.1.2.2 Definición de operadores básicos

Iniciar(pila P):

Cláusulas de Iniciar(pila P)	
Efecto	Inicializa la pila P como vacía.
Requiere	No aplica.
Modifica	Pila P.

Destruir(pila P):

Cláusulas de Destruir(pila P)	
Efecto	Destruye la pila P, dejándola inutilizable.
Requiere	Pila P inicializada.
Modifica	Pila P.

Vaciar(pila P):

Cláusulas de Vaciar(pila P)	
Efecto	Vacía pila P, dejándola con 0 elementos.
Requiere	Pila P inicializada.
Modifica	Pila P.

Vacía(pila P), devuelve algo de tipo booleano:

Cláusulas de Vacía(pila P)	
Efecto	Devuelve verdadero si pila P vacía y falso si no.
Requiere	Pila P inicializada.
Modifica	No aplica.

Poner(elemento e, pila P):

Cláusulas de Vacía(pila P)	
Efecto	Agrega un nuevo elemento en la pila P.
Requiere	Pila P inicializada.
Modifica	Pila P.

Quitar(pila P):

Cláusulas de Quitar(pila P)	
Efecto	Borra el último elemento que se puso en pila P.
Requiere	Pila P inicializada y con al menos un elemento.
Modifica	Pila P.

Tope(pila P), devuelve algo de tipo elemento:

Cláusulas de Tope(pila P)	
Efecto	Devuelve el último elemento que se puso en pila P.
Requiere	Pila P inicializada y con al menos un elemento.
Modifica	No aplica.

4.1.3 Modelo árbol n-ario

4.1.3.1 Definición del árbol n-ario

Según el libro *Estructura de datos y algoritmos* de Alfred V. Aho, Jeffrey D. Ullman y John E. Hopcroft: en general, un árbol impone una estructura jerárquica sobre una colección de objetos. En específico, un árbol es una colección de elementos llamados *nodos*, uno de los cuales se distingue como *raíz*, junto con una relación de "paternidad" que impone una estructura jerárquica sobre los nodos. Un nodo, como un elemento de una lista, puede ser del tipo que se desee. A menudo se representa un nodo por medio de una letra, una cadena de caracteres o un círculo con un número en su interior. Formalmente, un árbol se puede definir de manera recursiva como sigue:

(1) Un solo nodo es, por sí mismo, un árbol. Ese nodo es también la raíz de dicho árbol.
(2) Supóngase que n es un nodo y que $A_1, A_2, ..., A_k$ son árboles con raíces $n_1, n_2, ..., n_k$, respectivamente. Se puede construir un nuevo árbol haciendo que n se constituya en el padre de los nodos $n_1, n_2, ..., n_k$. En dicho árbol, n es la raíz y $A_1, A_2, ..., A_k$ son los *subárboles* de la raíz. Los nodos $n_1, n_2, ..., n_k$ reciben el nombre de *hijos* del nodo n.

Además, dos árboles A_1 y $A2$, son iguales sí y solo sí todos los hijos de cada nodo $n_{1,1}, n_{1,2}, ..., n_{1,m}$ del árbol A_1, se encuentra en el mismo orden que los hijos de los nodos $n_{2,1}, n_{2,2}, ..., n_{2,m}$ del árbol A_2. Es decir, importa el orden entre los hijos.

4.1.3.2 Definición de operadores básicos

Crear(árbol A):

Cláusulas de Crear(árbol A)	
Efecto	Inicializa el árbol A como vacío.
Requiere	No aplica.
Modifica	Árbol A.

Destruir(árbol A):

Cláusulas de Destruir(árbol A)	
Efecto	Destruye el árbol A, dejándolo inutilizable.
Requiere	Árbol A inicializado.
Modifica	Árbol A.

Vaciar(árbol A):

Cláusulas de Vaciar(árbol A)	
Efecto	Vacía árbol A, dejándolo sin etiquetas.
Requiere	Árbol A inicializado.
Modifica	Árbol A.

Vacío(árbol A), devuelve algo de tipo booleano:

Cláusulas de Vacío(árbol A)	
Efecto	Devuelve verdadero si árbol A vacío y falso si no.
Requiere	Árbol A inicializado.
Modifica	No aplica.

PonerRaíz(elemento e, árbol A):

Cláusulas de PonerRaíz(elemento e, árbol A)	
Efecto	Agrega la raíz de árbol A.
Requiere	Árbol A inicializado y con 0 elementos.
Modifica	Árbol A.

AgregarHijo(elemento e, nodo n, árbol A):

Cláusulas de AgregarHijo(etiqueta e, nodo n, árbol A)	
Efecto	Le agrega un nuevo hijo al nodo n con etiqueta e en árbol A.

Requiere Árbol A inicializado y n válido en A.
Modifica Árbol A.

BorrarHoja(nodo n, árbol A):

Cláusulas de BorrarHoja(nodo n, árbol A)
Efecto Elimina la hoja n en árbol A.
Requiere Árbol A inicializado, n válido en A y n hoja.
Modifica Nodo n en árbol A.

ModificarEtiqueta(etiqueta e, nodo n, árbol A):

Cláusulas de ModificarEtiqueta(etiqueta e, nodo n, árbol A)
Efecto Modifica la etiqueta del nodo n por e en árbol A.
Requiere Árbol A inicializado y nodo n válido en A.
Modifica Nodo n en árbol A.

Raíz(árbol A), devuelve algo de tipo nodo:

Cláusulas de Raíz(árbol A)
Efecto Devuelve el nodo raíz en árbol A.
Requiere Árbol A inicializado y con al menos 1 elemento.
Modifica No aplica.

Padre(nodo n, árbol A), devuelve algo de tipo nodo:

Cláusulas de Padre(nodo n, árbol A)
Efecto Devuelve el nodo padre del nodo n en árbol A.
Requiere Árbol A inicializado y nodo n válido en A.
Modifica No aplica.

HijoMásIzquierdo(nodo n, árbol A), devuelve algo de tipo nodo:

Cláusulas de HijoMásIzquierdo(nodo n, árbol A)
Efecto Devuelve el hijo más izquierdo del nodo n en árbol A.
Requiere Árbol A inicializado y n válido en A. Si el nodo n no tiene hijos, este método devuelve *nodoNulo*.
Modifica No aplica.

HermanoDerecho(nodo n, árbol A), devuelve algo de tipo nodo:

Cláusulas de HermanoDerecho(nodo n, árbol A)	
Efecto	Devuelve el hermano derecho del nodo n en árbol A.
Requiere	Árbol A inicializado y n válido en A. Si el nodo n no tiene hermanos derechos, este método devuelve *nodoNulo*.
Modifica	No aplica.

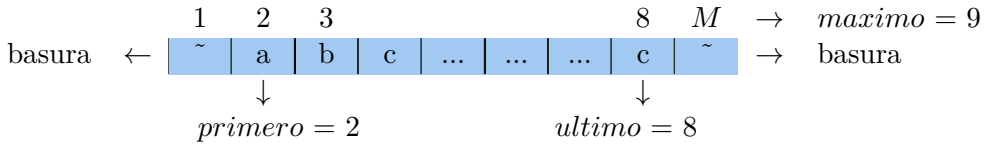
Etiqueta(nodo n, árbol A), devuelve algo de tipo etiqueta:

Cláusulas de Etiqueta(nodo n, árbol A)	
Efecto	Devuelve la etiqueta del nodo n en árbol A.
Requiere	Árbol A inicializado y n válido en A.
Modifica	No aplica.

4.2 Estructuras de datos

4.2.1 Arreglo circular

4.2.1.1 Diagrama y descripción



La estructura de datos arreglo circular, es una estructura implementada en memoria estática. Esta consiste, en un arreglo normal, que de manera lógica se le convierte en un arreglo circular, es decir, lo que lo hace circular es la manera en la que se manejan los elementos en el arreglo. Para poder darle la lógica circular a un arreglo, es necesario hacer varias cosas:

- Debe haber un apuntador al primero (*primer*).
- Debe haber un apuntador al último (*ultimo*).
- En caso de que $ultimo = maximo$, pero $maximo \; ! = n$, entonces, se deberá agregar apartir del índice 1.
- Cuando el arreglo se encuentre vacío, $ultimo = 0$ y $primero = 1$.
- Cuando el arreglo se encuentre lleno, $ultimo = 0$ y $primero = 1$. Como no se hace distinción del arreglo lleno o vacío, se pueden tomar tres caminos:
  - Definir el arreglo de tamaño $M + 1$.
  - Agregarle un tipo booleano a la estructura, que indique si está o no vacía.
  - Agregarle un tipo entero a la estructura, que indique el número de elementos.

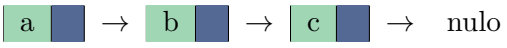
4.2.1.2 Descripción en C++

modeloCola.h:

```
1     class modeloCola{
2
3               private:
4
5          int numElem;
6          int primero;
7           int ultimo;
8     int arreglo[100];
9
10               public:
11
12     modeloCola();
13     void crear();
14     void destruir();
15     void vaciar();
16     bool vacia();
17     void agregar(int elemento);
18     void sacar();
19     int frente();
20
21 };
```

4.2.2 Lista simplemente enlazada

4.2.2.1 Diagrama y descripción



La estructura de datos lista simplemente enlazada, es una estructura en memoria dinámica implementada por punteros. Consiste en definir un primer puntero, el cual apuntará a un segundo puntero y así sucesivamente.

En una lista simplemente enlazada, los accesos se hacen a través de punteros, a diferencia del arreglo, en el cual se hacen mediante índices. Además, en la lista simplemente enlazada los elementos son contiguos, es decir, no hay posiciones vacías.

4.2.2.2 Descripción en C++

cajita.h:

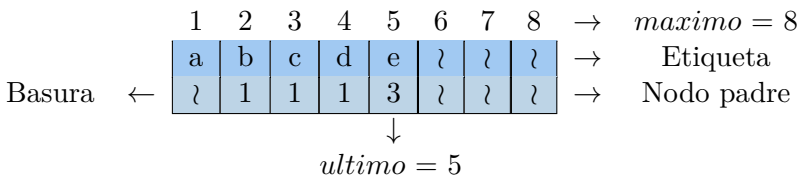
```
1  class modeloPila{
2
3  private:
4
5      cajita* siguiente;
6      int elemento;
7
8  public:
9
10     cajita();
11     cajita(int e);
12      cajita();
13     void setSiguiente;
14     cajita* getSiguiente();
15     void setElemento(int e);
16     int getElemento();
17
18 };
```

modeloPila.h:

```
1  class modeloPila{
2
3  private:
4
5      cajita* primero;
6
7  public:
8
9      modeloPila();
10     void crear();
11     void destruir();
12     void vaciar();
13     bool vacia();
14     void poner(int elemento);
15     void quitar();
16     int tope();
17
18 };
```

4.2.3 Arreglo con señalador al padre

4.2.3.1 Diagrama y descripción



Un arreglo con señalador al padre, es un arreglo normal en memoria dinámica, pero donde una posición contiene dos tipos distintos, que son la etiqueta y el nodo padre de esa etiqueta.

- Hay un señalador al último lleno.
- La raíz está en la primer posición, y en el campo del padre de la raíz, siempre hay basura (el espacio se desperdicia).
- Los hijos siempre están después del padre.

4.2.3.2 Descripción en C++

cajitasap.h:

```
1  class cajitasap{
2
3  private:
4
5      int etiqueta;
6      int padre;
7
8  public:
9
10     void setEtiqueta(int e);
11     int getEtiqueta();
12     void setPadre();
13     int getPadre();
14
15 };
```

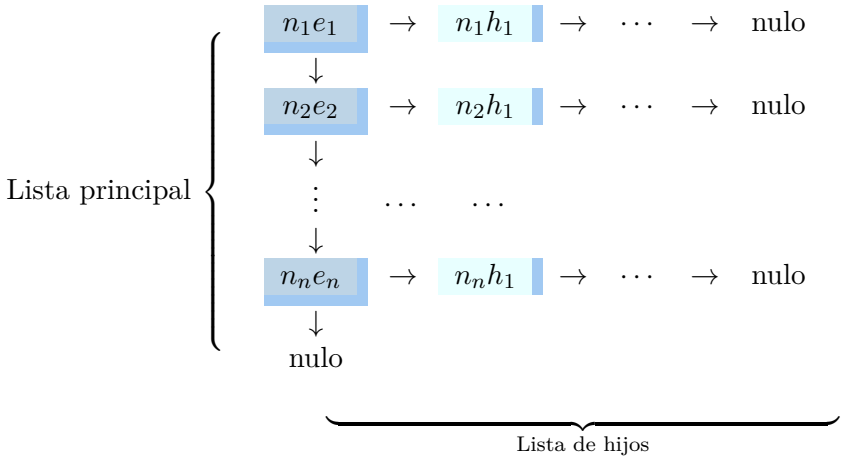
ArregloSeñAlPadre.h:

```
1  typedef int nodo;
2
3  class arbol{
4
5  private:
6
7      nodo ultimo;
8      cajitasap arreglo[100];
9      int numElem;
```

```
10
11  public:
12
13      arbol();
14      ~arbol();
15      void crear();
16      void destruir();
17      void vaciar();
18      bool vacio();
19      void ponerRaiz();
20      void arregarHijo();
21      void borrarHoja(nodo n);
22      nodo raiz();
23      nodo padre(nodo n);
24      nodo hijoMasIzq(nodo n);
25      nodo hermanoDer(nodo n);
26      nodo etiqueta(nodo n);
27      int nodo(int etiqueta);
28
29  };
```

4.2.4 Lista de hijos

4.2.4.1 Diagrama y descripción



La lista de hijos implementada por punteros, es una estructura de datos que consiste en una lista, que contiene todos los nodos y en donde cada nodo, señala a una lista donde están sus hijos.

Debido a que de la cantidad de nodos n, solo $n-1$ pueden ser hijos de alguien (la raíz no tiene padre), el tamaño máximo que puede tomar la lista de hijos, es de $n-1$, que es el caso en que la raíz es el padre de todos los $n-1$ nodos.

Además, en una lista de hijos, el primer nodo siempre es la raíz.

4.2.4.2 Descripción en C++

cajitahijos.h:

```
1  class cajitaprincipal;{
2
3  class cajitahijos{
4
5  private:
6
7      cajitaprincipal* estenodo;
8      cajitahijos* siguiente;
9
10 public:
11
12     cajitahijos(cajitaprincial* en);
13     cajitahijos();
14     ~cajitahijos();
15     void setEsteNodo(cajitaprincipal* en);
16     cajitaprincipal* getEsteNodo();
17     void setSiguiente(cajitahijos* s);
18     cajitahijos* getSiguiente();
19
20 };
```

cajitaprincipal.h:

```
1  class cajitahijos;{
2
3  class cajitaprincipal{
4
5  private:
6
7      int etiqueta;
8      cajitaprincipal* siguiente;
9      cajitahijos* primero;
10
11 public:
12
13     cajitaprincipal(int e);
14     cajitaprincipal();
15     ~cajitaprincial();
16     void setEtiqueta(int e);
17     int getEtiqueta();
18     void setSiguiente(cajitaprincipal* s);
19     cajitaprincipal* getSiguiente();
20     void setPrimero(cajitahijos* p);
```

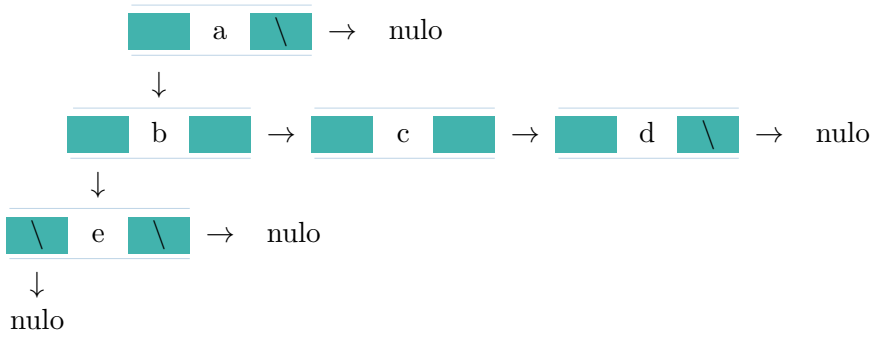
```
21      cajitahijos* getPrimero();
22
23  };
```

ListaHijos.h:

```
1   typedef cajitaprincipal* nodo;
2
3   class arbol{
4
5   private:
6
7       nodo raiz;
10
11  public:
12
13      arbol();
14      ~arbol();
15      void crear();
16      void destruir();
17      void vaciar();
18      bool vacio();
19      void ponerRaiz();
20      void arregarHijo();
21      void borrarHoja(nodo n);
22      nodo raiz();
23      nodo padre(nodo n);
24      nodo hijoMasIzq(nodo n);
25      nodo hermanoDer(nodo n);
26      nodo etiqueta(nodo n);
27      int nodo(int etiqueta);
28
29  };
```

4.2.5 Hijo más izquierdo- hermano derecho

4.2.5.1 Diagrama y descripción



La estructura de datos Hijo más izquierdo- hermano derecho, es una estructura de datos recursiva, implementada mediante punteros.

Esta esstructura consiste en un serie de “cajitas” conformadas por tres campos: un puntero al hijo más izquierdo, la etiqueta y un puntero al hermano derecho.

En el caso de la raíz, como no tiene hermanos, el puntero correspondiente estará siempre en nulo.

4.2.5.2 Descripción en C++

cajitahmi-hd.h:

```
1  class cajitahmihd{
2
3  private:
4
5      int etiqueta;
6      cajitahmihd* hijoMasIzq;
7      cajitahmihd* hermanoDer;
8
9  public:
10
11     cajitahmihd() :  hijoMasIzq(0), hermanoDer(0);
12     cajitahmihd(int e) :  etiqueta(e), hijoMasIzq(0)
       , hermanoDer(0);
13     ~cajitahmihd();
14     void setEtiqueta(int e);
15     int getEtiqueta();
16     void setHijoMasIzq(cajitahmihd* h);
17     cajitahmihd* getHijoMasIzq();
18     void setHermanoDer(cajitahmihd* h);
19     cajitahmihd* getHermanoDer();
20
21 };
```

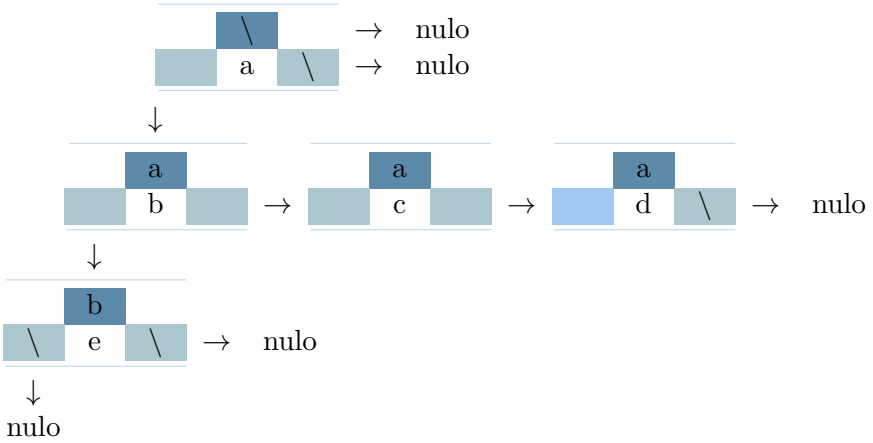
LMI-HD.h:

```
1  typedef cajitahmihd* nodo;
2
3  class arbol{
4
5  private:
6
7      nodo raiz;
8
9      destruirRec(nodo n);
10     vaciarRec(nodo n);
11     borrarHojaRec(nodo n);
12
```

```
13  public:
14
15      arbol();
16      ~arbol();
17      void crear();
18      void destruir();
19      void vaciar();
20      bool vacio();
21      void ponerRaiz();
22      void arregarHijo();
23      void borrarHoja(nodo n);
24      nodo raiz();
25      nodo padre(nodo n);
26      nodo hijoMasIzq(nodo n);
27      nodo hermanoDer(nodo n);
28      nodo etiqueta(nodo n);
29      int nodo(int etiqueta);
30
31  };
```

4.2.6 Hijo más izquierdo- hermano derecho-padre

4.2.6.1 Diagrama y descripción



La estructura de datos Hijo más izquierdo- hermano derecho, con puntero al padre, es de igual manera una estructura de datos recursiva, implementada mediante punteros.

Esta esstructura consiste en un serie de "cajitas" conformadas por cuatro campos: un puntero al hijo más izquierdo, un puntero al padre, la etiqueta y un puntero al hermano derecho.

En el caso de la raíz, como no tiene hermanos, ni tampoco padre, los punteros correspondientes estarán siempre en nulo.

4.2.6.2 Descripción en C++

cajitahmi-hd-p.h:

```
1  class cajitahmihdp{
2
3  private:
4
5      int etiqueta;
6      cajitahmihdp* hijoMasIzq;
7      cajitahmihdp* hermanoDer;
8      cajitahmihdp* padre;
9
10 public:
11
12     cajitahmihdp() :  hijoMasIzq(0), hermanoDer(0)
       , padre(0);
13     cajitahmihdp(int e, cajitahmihdp p) :  etiqueta(e)
       , hijoMasIzq(0), hermanoDer(0), padre(0);
14     ~cajitahmihdp();
15     void setEtiqueta(int e);
16     int getEtiqueta();
17     void setHijoMasIzq(cajitahmihdp* h);
18     cajitahmihdp* getHijoMasIzq();
19     void setHermanoDer(cajitahmihdp* h);
20     cajitahmihdp* getHermanoDer();
21     void setPadre(cajitahmihdp* p);
22     cajitahmihdp* getPadre();
23
24 };
```

LMI-HD-Padre.h:

```
1  typedef cajitahmihdp* nodo;
2
3  class arbol{
4
5  private:
6
7      nodo raiz;
8
9      destruirRec(nodo n);
10     vaciarRec(nodo n);
11
12 public:
13
```

```
14      arbol();
15      ~arbol();
16      void crear();
17      void destruir();
18      void vaciar();
19      bool vacio();
20      void ponerRaiz();
21      void arregarHijo();
22      void borrarHoja(nodo n);
23      nodo raiz();
24      nodo padre(nodo n);
25      nodo hijoMasIzq(nodo n);
26      nodo hermanoDer(nodo n);
27      nodo etiqueta(nodo n);
28      int nodo(int etiqueta);
29
30  };
```

4.3 Algoritmos

4.3.1 Hermano izquierdo

4.3.1.1 Definición y especifiación

El algoritmo busca el hermano izquierdo de un nodo n, en el árbol A, usando los operadores básicos del modelo árbol n-ario.

Cláusulas de HerIzq(nodo n, árbol A)

Efecto	Devuelve el hermano izquierdo del nodo n en el árbol A.
Requiere	Nodo n válido en árbol A y n en posición distinta de 1.
Modifica	No aplica.

4.3.1.2 Descripción, detalles, pseudolenguaje

El algoritmo utiliza el operadore básicos para obtener el padre del nodo n, luego recorre los hijos del padre hasta llenar al nodo anterior al nodo n, de manera que ese nodo, será el hermano izquierdo del nodo n.

4.3.2 Etiquetas repetidas

4.3.2.1 Definición y especifiación

El algoritmo busca si el árbol A tiene etiquetas retetidas, usando los operadores básicos del modelo árbol n-ario.

Cláusulas de EtiquetasRepetidas(árbol A)

Efecto	Devuelve verdadero si el árbol A tiene etiquetas repetidas y falso si no.
Requiere	Árbol A inicializado.
Modifica	No aplica.

4.3.2.2 Descripción, detalles, pseudolenguaje

El algoritmo hace un recorrido en pre-orden por cada nodo en el árbol, de manera que determine si el nodo n_1 tiene la misma etiqueta que el nodo n_2, para $n_1 \neq n_2$.

4.3.3 Altura del nodo n

4.3.3.1 Definición y especifiación

Averigua la altura del nodo n, en el árbol A, usando los operadores básicos del modelo árbol n-ario.

Cláusulas de Altura(nodo n, árbol A)

Efecto	Devuelve la altura del nodo n en el árbol A.
Requiere	Nodo n válido en árbol A.
Modifica	No aplica.

4.3.3.2 Descripción, detalles, pseudolenguaje

El algoritmo hace un recorrido en pre-orden y con ayuda de un contador, determina la altura de un nodo.

4.3.4 Profundidad del nodo n

4.3.4.1 Definición y especifiación

Averigua la profundidad del nodo n, en el árbol A, usando los operadores básicos del modelo árbol n-ario.

Cláusulas de Profundidad(nodo n, árbol A)

Efecto	Devuelve la profundidad del nodo n en el árbol A.
Requiere	Nodo n válido en árbol A.
Modifica	No aplica.

4.3.4.2 Descripción, detalles, pseudolenguaje

El algoritmo se devuelve de padre en padre, a partir de nodo n, llevando un contador para

determinar la altura del nodo n.

4.3.5 Niveles del árbol

4.3.5.1 Definición y especifiación

Averigua los niveles del árbol A, usando los operadores básicos del modelo árbol n-ario.

Cláusulas de Niveles(nodo n, árbol A)

Efecto	Devuelve los niveles del árbol A.
Requiere	Árbol A inicializado.
Modifica	No aplica.

4.3.5.2 Descripción, detalles, pseudolenguaje

El algoritmo recorre en pre-orden el árbol, llevando un contador, de manera que cuando llegue a una raíz, compruebe si la mayor nivel hasta el momento es menor al nivel hasta esa hoja y de esta manera, modificarlo.

4.3.6 Etiquetas del i-ésimo nivel

4.3.6.1 Definición y especifiación

Lista las etiquetas del nodos en el i-ésimo nivel en el árbol A, usando los operadores básicos del modelo árbol n-ario.

Cláusulas de EtiquetasNivel(nivel i, árbol A)

Efecto	Lista los nodos del i-ésimo nivel en el árbol A.
Requiere	Nivel i válido en árbol A.
Modifica	No aplica.

4.3.6.2 Descripción, detalles, pseudolenguaje

El algoritmo llegar al i-ésimo nivel y a partir de ahí, hace un recorrido secuencial, listando las etiquetas de ese nivel.

4.3.7 Listar hijos del nodo n

4.3.7.1 Definición y especifiación

Lista las etiquetas de los hijos del nodo n, en el árbol A, usando los operadores básicos del modelo árbol n-ario.

Cláusulas de EtiquetasHijos(nodo n, árbol A)

Efecto	Listar los hijos del nodo n en el árbol A.
Requiere	Nodo n válido en árbol A y n con al menos un hijo.
Modifica	No aplica.

4.3.7.2 Descripción, detalles, pseudolenguaje

El algoritmo hace un recorrido secuencial a partir del hijo más izquierdo del nodo n, hasta listar todos sus hijos.

4.3.8 Cantidad de nodos del árbol

4.3.8.1 Definición y especifiación

Averigua la cantidad de nodos del árbol A, usando los operadores básicos del modelo árbol n-ario.

Cláusulas de CantidadNodos(árbol A)

Efecto	Devuelve la cantidad de nodos del árbol A.
Requiere	Árbol A con al menos un nodo.
Modifica	No aplica.

4.3.8.2 Descripción, detalles, pseudolenguaje

El algoritmo hace un recorrido en pre-orden y con ayuda de un contador determina la cantidad de nodos que tiene el árbol.

4.3.9 Borrar a partir del nodo n

4.3.9.1 Definición y especifiación

Borra el sub-árbol generado a partir del nodo n, en el árbol A, usando los operadores básicos del modelo árbol n-ario.

Cláusulas de BorrarSubArbol(nodo n, árbol A)

Efecto	Borra el sub-árbol que se genera a partir del nodo n en el árbol A.
Requiere	Nodo n válido en árbol A y n con al menos un hijo.
Modifica	Árbol A.

4.3.9.2 Descripción, detalles, pseudolenguaje

El algoritmo hace un recorrido en pre-orden a partir de nodo n, de manera que borra desde las primeras hojas, para lograr que en algún momento todos los nodos sean hojas y así borrarlos.

4.3.10 Borrar nodo n y trasladar

4.3.10.1 Definición y especifiación

Borra el nodo n, tal que el sub-árbol generado a partir de n se traslada al padre de n, en el árbol A, usando los operadores básicos del modelo árbol n-ario.

Cláusulas de BorrarYTrasladar(nodo n, árbol A)

Efecto	Borra el nodo n y tralada el sub-árbol generado a partir del nodo n al padre de n, en el árbol A.
Requiere	Nodo n válido en árbol A y n con al menos un hijo.
Modifica	Árbol A.

4.3.10.2 Descripción, detalles, pseudolenguaje

El algoritmo primero copia el sub-árbol generado a partir del nodo n en un árbol auxiliar y a partir de ahí, lo copia de nuevo en árbol original, conservando la posición de los nodos.

4.3.11 Averiguar si dos árboles son iguales

4.3.11.1 Definición y especifiación

Averigua si el árbol A es igual al árbol B, usando los operadores básicos del árbol n-ario.

Cláusulas de Iguales(árbol A, árbol B)

Efecto	Devuelve verdadero si el árbol A es igual al árbol B y falso si no.
Requiere	Árboles A y B inicializados.
Modifica	No aplica.

4.3.11.2 Descripción, detalles, pseudolenguaje

El algoritmo hace un recorrido en pre-orden de ambos árboles a la vez y en el momento en que un nodo n_a del árbol A tenga etiqueta distinta a su nodo n_b correspondiente en el árbol B, determina que estos árboles no son iguales. De igual manera, si los árboles no tienen la misma estructura o la misma cantidad de nodos, determina que estos no son iguales.

4.3.12 Listar árbol en pre-orden

4.3.12.1 Definición y especifiación

Lista en pre-orden los nodos del árbol A, usando los operadores básicos del modelo árbol n-ario.

Cláusulas de ListarPreOrden(árbol A)

Efecto	Lista en pre-orden el árbol A.
Requiere	Árbol A inicializado.
Modifica	No aplica.

4.3.12.2 Descripción, detalles, pseudolenguaje

El algoritmo realiza un recorrido en pre-orden, listando las etiquetas de los nodos del árbol A.

4.3.13 Listar árbol en post-orden

4.3.13.1 Definición y especifiación

Lista en post-orden los nodos del árbol A, usando los operadores básicos del modelo árbol n-ario.

Cláusulas de ListarPostOrden(árbol A)

Efecto	Lista en post-orden el árbol A.
Requiere	Árbol A inicializado.
Modifica	No aplica.

4.3.13.2 Descripción, detalles, pseudolenguaje

El algoritmo realiza un recorrido en post-orden, listando las etiquetas de los nodos del árbol A.

4.3.14 Listar árbol en in-orden

4.3.14.1 Definición y especifiación

Lista en in-orden los nodos del árbol A, usando los operadores básicos del modelo árbol n-ario.

Cláusulas de ListarInOrden(árbol A)

Efecto	Lista en in-orden el árbol A.
Requiere	Árbol A inicializado.
Modifica	No aplica.

4.3.14.2 Descripción, detalles, pseudolenguaje

El algoritmo realiza un recorrido en in-orden, listando las etiquetas de los nodos del árbol A.

4.3.15 Listar árbol por niveles

4.3.15.1 Definición y especifiación

Lista por niveles los nodos del árbol A, usando los operadores básicos del modelo árbol n-ario y el modelo auxiliar cola.

Cláusulas de ListarPorNiveles(árbol A)

Efecto	Lista por niveles el árbol A.
Requiere	Árbol A inicializado.
Modifica	No aplica.

4.3.15.2 Descripción, detalles, pseudolenguaje

El algoritmo realiza un recorrido por niveles, con la ayuda del modelo auxiliar pila, listando las etiquetas de los nodos del árbol A.

4.3.16 Listar árbol en pre-orden, con Pila

4.3.16.1 Definición y especifiación

Lista en pre-orden los nodos del árbol A, usando los operadores básicos del modelo árbol n-ario y usando el modelo auxiliar cola, para simular la recursividad.

Cláusulas de ListarPreOrdenPila(árbol A)

Efecto	Lista en pre-orden el árbol A, usando el modelo auxiliar pila.
Requiere	Árbol A inicializado.
Modifica	No aplica.

4.3.16.2 Descripción, detalles, pseudolenguaje

El algoritmo realiza un recorrido en pre-orden, simulando la recursividad que provee el compilador, con ayuda del modelo auxiliar pila. De esta manera, lista las etiquetas de los nodos del árbol A.

4.3.17 Buscar el nodo con etiqueta e

4.3.17.1 Definición y especifiación

Busca el nodo correspondiente a la etiqueta e, en el árbol A, usando los operadores básicos del modelo árbol n-ario.

Cláusulas de Nodo(árbol A)

Efecto	Devuelve el nodo con etiqueta e en el árbol A.
Requiere	Nodo n válido en arbol A.
Modifica	No aplica.

4.3.17.2 Descripción, detalles, pseudolenguaje

El algoritmo realiza un recorrido en pre-orden, preguntando por la etiqueta de cada nodo, hasta encontrar al nodo correspondiente a la etiqueta e.

5 Manual de usuario

5.1 Requerimientos de hardware

- Procesador Inter(R) Core(TM) i5-4200U CPU @ 1.60GHz
- Ram 6,00 GB

5.2 Requerimientos de software

- Sistema operativo Windows 10
- Arquitectura 64 bits
- Ambiente Code::Blocks

5.3 Requerimientos de software

- 64 bits

5.4 Compilación

- GNU GCC compiler

5.5 Especificación de las funciones del programa

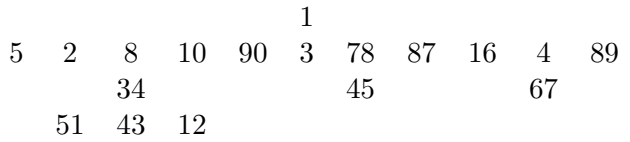
- Antes de utilizar cualquier modelo, es necesario seleccionar la opción “Crear”.
- Los modelos “Cola” y “Pila” trabajan con tipo de datos entero, al agregar un elemento en alguno de estos modelos, asegúrese de que sean enteros.
- El modelo árbol, trabaja por posiciones, al agregar una nueva etiqueta, asegúrese de que la posición que se elija sea una válida para la estructura que tenga el árbol en ese momento.
- El modelo árbol, trabaja con etiquetas de manera que no hayan etiquetas repetidas, asegúrese de no introducir etiquetas repetidas excepto en el caso en que se desee trabajar con el algoritmo que determina si hay etiquetas repetidas.
- En todas las estructuras de datos, asegúrese de hacer sus operaciones con elementos y/o etiquetas que formen parte del modelo.
- Una vez que se regrese a donde esté la opción de trabajar con un modelo, en caso de que no hay estado trabajando con un modelo anteriormente, este modelo se destruirá.

En caso de que no se sigan las indicaciones anteriores, no se asegura el correcto funcionamiento ni de la estructura, ni de las distintas operaciones que desee utilizar.

6 Datos de prueba

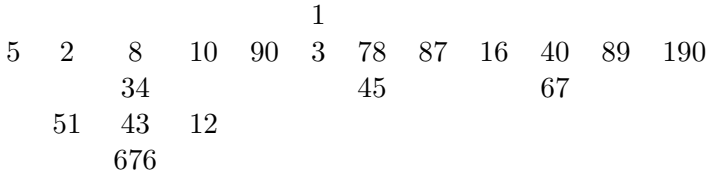
6.1 Formato de las prueba

Para las pruebas generales, se eligió un árbol 1 con la siguiente estructura:

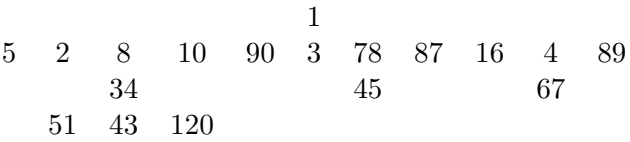


Donde se interpreta como que la raíz del árbol es 1, sus hijos son 5, 2, 8, 10, 90, 3, 78, 87, 16, 4 y 89. 8 tiene un solo hijo que es 34 y este, tiene tres hijos 51, 43 y 12. 78 tiene un único hijo que es 45 y 4 también tiene un único hijo, que es 67.

Además, para las pruebas del algoritmo que determina si dos árboles son iguales, se elijió el árbol 2, con la siguiente estructura:



, el árbol 3, con la siguiente estructura:



y el árbol 2, modificándolo de manera que quede con la misma estructura que el árbol 1.

6.2 Salida esperada

Los algoritmos arrojaron las siguientes respuestas:

Algoritmo	# de prueba	Prueba	Resultado
Hermano izquierdo	Prueba 1	Hermano izquierdo de 12?	43
	Prueba 2	Hermano izquierdo de 89?	4
Etiquetas repetidas			
Altura de nodo n	Prueba 1	Altura de 12?	0
	Prueba 2	Altura de 1?	3
	Prueba 3	Altura de 78?	1
Profundidad de nodo n	Prueba 1	Profundidad de 12?	3
	Prueba 2	Profundidad de 1?	0
	Prueba 3	Profundidad de 45?	2
Niveles del árbol	Prueba 1	$--$	4
Etiquetas i-ésimo nivel	Prueba 1	Nivel 2	5.2.8.10.90.3.78.87.16.4.89
	Prueba 2	Nivel 3	34.45.67
Etiquetas hijos del nodo n	Prueba 1	Hijos de 1	5.2.8.10.90.3.78.87.16.4.89
	Prueba 2	Hijos de 34	51.43.12
Cantidad de nodos	Prueba 1	$--$	18
Borrar sub-árbol	Prueba 1	78	Funcionó
Borrar sub-árbol y trasladar	Prueba 1	8	Funcionó
Copiar un árbol	Prueba 1	Árbol 1 en 2	Funcionó
Son iguales	Prueba 1	Árbol 1 y 2	No son iguales
	Prueba 2	Árbol 1 y 2 modificado	Son iguales
	Prueba 3	Árbol 3 y 1	No son iguales
	Prueba 4	Árbol 2 y 2	Son iguales
Listar pre-orden	Prueba 1	$--$	1.5.2.8.34.51.43.12.10.90.3 78.45.87.16.4.67.89
Listar post-orden	Prueba 1	$--$	5.2.51.43.12.34.8.10.90.3.45 78.87.16.67.4.89.1
Listar in-orden	Prueba 1	$--$	5.1.2.51.34.43.12.8.10.90.3 45.78.87.16.67.4.89
Listar por niveles	Prueba 1	$--$	1.5.2.8.10.90.3.78.87.16.4 89.34.45.67.51.43.12
Listar pre-orden pila	Prueba 1	$--$	1.5.2.8.34.51.43.12.10.90.3 78.45.87.16.4.67.89
Nodo de etiqueta	Prueba 1	Etiqueta de 90	Nodo con etiqueta 90
Nodo de etiqueta	Prueba 2	Etiqueta de 12	Nodo con etiqueta 12

6.3 Salida obtenida (análisis en caso de fallo)

Los análisis arrojaron los resultados esperados en todos los casos de prueba.

8 Listado de archivos (estructura de las carpetas)

- Código
  - Arbol n-ario
    - ArregloSeñAlPadre
      - ArrergloSeñAlPadre.cbp
      - ArrergloSeñAlPadre.h
      - ⋮
    - HMI-HD
      - HMI-HD.cbp
      - HMI-HD.h
      - ⋮
    - HMI-HD-Padre
      - HMI-HD-Padre.cbp
      - HMI-HD-Padre.h
      - ⋮
    - ListaHijos
      - ListaHijos.cbp
      - ListaHijos.h
      - ⋮
  - Cola
    - Cola.cbp
    - modeloCola.h
    - ⋮
  - ColaPrueba
    - ColaPrueba.cbp
    - ColaPrueba.h
    - ⋮
  - menú
    - menu.cbp
    - menu.h
    - ⋮
  - Pila
    - modeloPila.cbp
    - modeloPila.h
    - ⋮
  - PilaPrueba
    - PilaPrueba.cbp
    - PilaPrueba.h
    - ⋮
- Documentación
  - Documentación.pdf

9 Bibliografía

[1] Alfred V. Aho, Jeffrey D. Ullman, John E. Hopcroft. *Estructura de datos y algoritmos.* 1988.