WAZE GUATEMALA

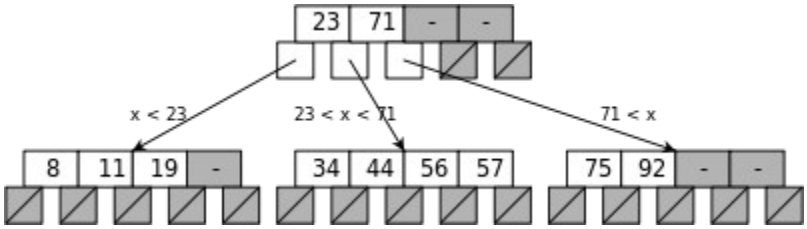
MANUAL TECNICO

Celia Esmeralda Vargas Lopez.

ARBOL B

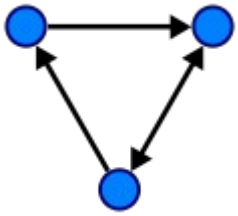
B-árboles son estructuras de datos de árbol que se encuentran comúnmente en las implementaciones de bases de datos y sistemas de archivos. Al igual que los árboles binarios de búsqueda, son árboles balanceados de búsqueda, pero cada nodo puede poseer más de dos hijos.

Los árboles B mantienen los datos ordenados y las inserciones y eliminaciones se realizan en tiempo logarítmico amortizado.



GRAFOS DIRIGIDOS

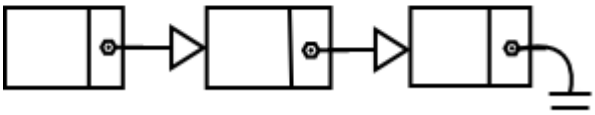
es un tipo de grafo en el cual las aristas tienen un sentido definido, a diferencia del grafo no dirigido, en el cual las aristas son relaciones simétricas y no apuntan en ningún sentido.



PILAS Y LISTAS

La lista enlazada es un TDA que nos permite almacenar datos de una forma organizada, al igual que los vectores pero, a diferencia de estos, esta estructura es dinámica, por lo que no tenemos que saber "a prioridad" los elementos que puede contener.

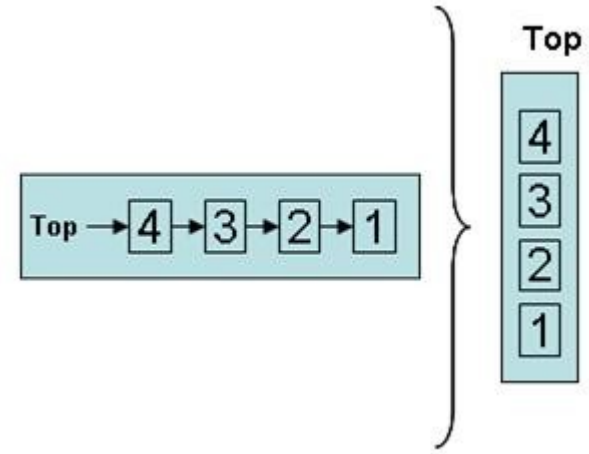
Estructura de un nodo



Lista enlazada

Una pila, es una estructura de datos en la que el último elemento en entrar es el primero en salir, opr lo que también se denominan estructuras LIFO (Last In, First Out).

En esta estructura sólo se tiene acceso a la cabeza o cima de la pila.



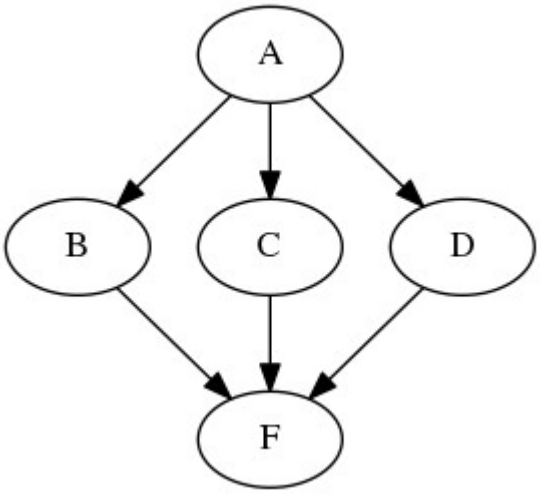
HERRAMIENTAS EXTRAS

Graphviz

Graphviz consiste en un lenguaje de descripción de gráficos llamado DOT, un conjunto de herramientas y librerías que pueden generar o procesar archivos DOT:4

dot

Una herramienta en línea de comandos para producir imágenes en capas de grafo dirigido en una variedad de formatos de salida (PostScript, pdf, svg, png, etc.).



PROGRAMACION

GRAFO

- private List<nodoG> nodes;

NODO

- private String ciudad;
- private listaAristas aristaNodo;

ARISTA

- private nodoG origen;
- private nodoG destino;
- private double distancia;
- private double tiempoVehiculo;
- private double tiempoPersona;
- private double gasolina;
- private double desgasteFisico;

LISTA DE ARISTAS

- private nodoArista inicio;
- private int tamanio=0;

LISTA DE CIUDADES

- private nodoCiudad inicio;
- private int tamanio=0;

CREACION DE GRAFO

```
for (int i = 0; i < listaCiudades.getTamanio(); i++) {
    nodoG nodoLista = listaCiudades.getValor(i);
    grafoCompleto.agregarNodo(nodoLista);
}
```

LISTAR DATOS DE GRAFO

```
for (int i = 0; i < grafoCompleto.getNodes().size(); i++) {
    System.out.println(grafoCompleto.getNodes().get(i).getCiudad());
    if (grafoCompleto.getNodes().get(i).getAristas() == null) {
        System.out.println("No hay aristas");
    } else {
```

```
            grafoCompleto.getNodes().get(i).getAristas().listar();
        }
    }
```

RECORRIDO DE GRAFO

/*Obtenemos nodo inicio y nodo destino buscamos en grafo el nodo inicio, obtenemos sus adyacentes y evaluamos adyacentes de cada uno. */

```
    pilaNodosRecorridos.clear();
    listaRecorridos.clear();
    if (ciudadInicio.equals(ciudadDestino)) {
        System.out.println("La ciudad origen  es la misma que el destino");
    } else {
        for (int i = 0; i < grafoCompleto.getNodes().size(); i++) {
            String ciudadEncontrada = grafoCompleto.getNodes().get(i).getCiudad();
            nodoG ciudadEncontrada2 = grafoCompleto.getNodes().get(i);


            if (ciudadEncontrada.equals(ciudadInicio)) {
                //agregamos a pila la primera ciudad
                pilaNodosRecorridos.push(ciudadInicio);
                System.out.println(ciudadEncontrada);
                recorridoAdyacentes(ciudadEncontrada2, ciudadDestino);
            } else {
            }
        }
    }
```

RECORRIDO DE ADYACENTES

```
    //obtenemos aristas de la ciudad
    listaAristas lista1 = ciudad.getAristas();
    if (lista1 == null) {//si no hay aristas quitamos de la pila
        System.out.println(" No hay aristas ");
        pilaNodosRecorridos.pop();
    } else {//Hay aristas
        //recorremos cada arista para obtener destino
```

```
for (int i = 0; i < lista1.getTamanio(); i++) {
    //agregamos a pila ciudad recorrida
    String ciudadRecorrida = lista1.getValor(i).getDestino().getCiudad();
    pilaNodosRecorridos.push(ciudadRecorrida);
    nodoG ciudadL = lista1.getValor(i).getDestino();
    //si esta ciudad es la que hemos buscado entonces se desapila,
    if (ciudadRecorrida.equals(ciudadDestino)) {
        //imprimimos la pila
        //eliminamos el dato de la pila
        //verificar porque la pila queda vacia
        llenarlista(pilaNodosRecorridos);
        pilaNodosRecorridos.pop();
    } else {//si no es la ciudad buscada
        boolean ciudadEncontrada = false;
        //buscamos en la pila si esta ciudad ya existe
        for (int j = 0; j < pilaNodosRecorridos.size(); j++) {
            String dato = (String) pilaNodosRecorridos.get(j);
            if (j == pilaNodosRecorridos.size() - 1) {
            } else {
                if (dato.equals(ciudadRecorrida)) {
                    ciudadEncontrada = true;
                    break;
                } else {
                    ciudadEncontrada = false;
                }
            }
        }
        //la ciudad ya esta en pila, no buscamos adyacentes
        if (ciudadEncontrada == true) {
            System.out.println("\t\t Existe la ciudad en pila");
            pilaNodosRecorridos.pop();
        } //no se ha encontrado la ciudad entonces buscamos sus adyacente
```

```
                else {
                    System.out.println("\t\t No existe la ciudad en pila");
                    recorridoAdyacentes(ciudadL, ciudadDestino);
                }
            }
        }
        //cuando termine el recorrido de cada nodo entonces se  desapila
        pilaNodosRecorridos.pop();
    }
}
```

ARBOL

- static int gradoArbol;
- public static nodoArbol raizArbol;

NODOS

- static int gradoTree;
- public static int cantidadCLave;
- double[] claves;
- nodoArbol[] hijos;
- public static boolean hoja;
- nodoArbol padre;

INSERCION DE NODOS

```
nodoArbol nodoAux = arbol.raizArbol;//this method finds the node to be inserted as
//it goes through this starting at root node.
if (nodoAux.cantidadCLave == 2 * gradoArbol - 1)//if is full
{
    nodoArbol nodoNUevo = new nodoArbol(gradoArbol, null);//new node
    arbol.raizArbol = nodoNUevo;
    nodoNUevo.hoja = false;
    nodoNUevo.cantidadCLave = 0;
    nodoNUevo.hijos[0] = nodoAux;
    division(nodoNUevo, 0, nodoAux);//division de taiz
```

```
            insercionNodoLleno(nodoNUevo, clave);
        } else {
            insercionNodoLleno(nodoAux, clave);
        }
```

INSERCION NODO LLENO

```
        int i = nodo.cantidadCLave;
        if (nodo.hoja) {
            while (i >= 1 && clave < nodo.claves[i - 1])
            {
                nodo.claves[i] = nodo.claves[i - 1];
                i--;
            }
            nodo.claves[i] = clave;
            nodo.cantidadCLave++;
        } else {
            int j = 0;
            while (j < nodo.cantidadCLave && clave > nodo.claves[j])//find spot to recurse
            {
                j++;
            }
            if (nodo.hijos[j].cantidadCLave == gradoArbol * 2 - 1) {
                division(nodo, j, nodo.hijos[j]);
                if (clave > nodo.claves[j]) {
                    j++;
                }
            }
            insercionNodoLleno(nodo.hijos[j], clave);//recurse
        }
```

MOSTRAR ARBOL

```
        for (int i = 0; i < nodo.cantidadCLave; i++) {
            System.out.println("**" + nodo.getValue(i) + "**");
            double numero = nodo.getValue(i);
```

```
            String dato = String.valueOf(nodo.getValue(i));
            listaArbol.add(dato);
        }
        if (!nodo.hoja)// esto se llama cuando root no es leaf;
        {
            for (int j = 0; j <= nodo.cantidadCLave; j++) // en este bucle repetimos
            {// para imprimir el árbol en
                if (nodo.getChild(j) != null) // preorden.
                {                           //llendo desde el mas izquierda
                    System.out.println("--");         //hijo a la derecha
                    imprimir(nodo.getChild(j));    //hijo
                }
            }
        }
```

CREACION DE IMAGEN A TRAVEZ DE GRAPHVIZ

```
        try {
            ProcessBuilder pbuilder;
            pbuilder = new ProcessBuilder("dot", "-Tjpg", "-o", direccionPng, direccionDot);
            pbuilder.redirectErrorStream(true);
            pbuilder.start();
        } catch (Exception e) {
            e.printStackTrace();
        }
```

SISTEMA OPERATIVO : LINUX

LENGUAJE: JAVA