Tabla Hash

Lista Doblemente Enlazada:

Para la implementación de la tabla hash se hizo el uso de una lista doblemente enlazada con los métodos siguientes

- Insertar de ultimo
- Busqueda
- Eliminacion
- Graficar nodos para poderlos meter en la tabla hash como subgrafos
- Generar PDF es para mostrar en una tabla todos los datos

```
public class DoublyLinkedList {

    class NodeCustomers{...13 lines }
    public NodeCustomers head;
    public NodeCustomers last;
    private int size;
    public DoublyLinkedList() {...5 lines }

    public int getSize(){...3 lines }


    public void InsertLast(Customers customers){...18 lines }

    public NodeCustomers Search(String _id){...10 lines }

    public void Delete(String _id){...23 lines }


     public void print() {...8 lines }

     public void generatePDF1(){...23 lines }

     public String generateNode(int cell){...17 lines }

     @Override
        public String toString() {...3 lines }
}
```

Metodos de la Tabla Hash

- Insert Node: este lo que hace es primero obtiene el valor de la función hash y luego lo inserta en la lista doble
- DeletNode: este lo que hace es lo mismo verifica el valor hash en la función y si existe elimina el elemento de la lista
- Resize: este método es el rehashing lo que hace es que verifica primero si la cantidad de elementos no ha sobrepasado la densidad del tamaño de

la tabla hash y luego lo que hacemos es copiar todos los elementos a otra tabla hash incrementado el tamaño de esta por 2

- EditNode: lo que se hace en este método es editar los elementos en la tabla hash
- GenerateReportTablaHash: genera el grafico de la tabla hash

```
public static Paragraph column1,column2,column3,column4,colu
                         data1,data2,data3,data4,data5,
private DoublyLinkedList[] tablaHash;
public static int k =1;
private float numOfItems;
private int capacity; //it is the maximum number of items th
private double load_factor;
private final double sparce = 0.75;
public HashTable() {...6 lines }
public HashTable(int capacity){...6 lines }

public int hashFuntion(String key){...9 lines }

public void InsertNode(Customers _customers){...11 lines }

public void DeleteNode(String id){...11 lines }

public void Resize(int newCapacity){...17 lines }

public void EditNode(String _dpi,String _name, String _last_

public Customers search(String _dpi){...9 lines }


public void generatePDF(){...63 lines }


public String GenerateReportTablaHash(){...25 lines }

public String SubGrafo(){...22 lines }

public int size(){
    return 0;
}
```

Doubly Linked List Circular

- Insert Begin: este método lo que realiza es insertar al inicio de la lista esto con la ayuda de los nodos start y next
- Insert last: este método lo que realiza es insertar al final de la lista con la ayuda de los nodos last y prev y start para igualarlo con last
- Delete: elimina los elementos de la lista
- EditDrivers: edita los elementos de la lista

- GeneratePDF: este método lo que realiza es una tabla en formato pdf para poder mostrar todos los valores

```
import java.io.File;
import java.io.FileOutputStream;
import java.text.SimpleDateFormat;
import java.util.Date;
import javax.swing.JOptionPane;

/**
 *
 * @author dani318200
 */
public class DoublyLinkedListCircular {
    private node_Driver start;
    private node_Driver last;
    private static final PdfPTable tabla = new PdfPTable(9);
    private static PdfPCell  titleCell;
    private static Paragraph column1,column2,column3,column4,col
                             data1,data2,data3,data4,data5,
    public int size;
    public DoublyLinkedListCircular(){...5 lines }

    public boolean isEmpty(){...3 lines }

    public int getSize(){...3 lines }

    public void insertBegin(Drivers drivers){...17 lines }

    public void insertLast(Drivers drivers){...16 lines }

    public void delete(String DPI){...25 lines }

    public void EditDrivers(String _dpi,String _newDPI,String _n

    public node_Driver Search(String DPI){...9 lines }

    public Drivers SearchForUpdate(String DPI){...6 lines }

     public void print() {...14 lines }

    public void generatePDF(){...85 lines }

    public String generateDot(){...31 lines }

    public String SubGrafo(){...31 lines }

    public void sortList(boolean asc){...28 lines }
}
```

Codigo Huffman:

NodoArbol: Esta clase lo que realiza es la simulación es de un árbol prefijo pero para es lo que realiza es un recorrido tanto por la izquierda y por la derecha teniendo en cuenta la prioridad de las palabras

```
package Huffman;
public class NodoArbol {

    private Letra Valor;
    private NodoArbol Izquierda;
    private NodoArbol Derecha;
    public static String intento = "";

    public NodoArbol(){...6 lines }

    public NodoArbol(Letra Valor){...5 lines }

    public void Mostrar(String Continuidad){...12 lines }


    public String Buscar(char Letra, String Continuidad){...22 lines }

    public void MostrarOrden(){...12 lines }

    public Letra getValor() {
        return Valor;
    }

    public void setValor(Letra Valor) {
        this.Valor = Valor;
    }

    public NodoArbol getIzquierda() {
        return Izquierda;
    }

    public void setIzquierda(NodoArbol Izquierda) {
        this.Izquierda = Izquierda;
    }

    public NodoArbol getDerecha() {
        return Derecha;
    }

    public void setDerecha(NodoArbol Derecha) {
        this.Derecha = Derecha;
    }
}
```

En la clase Codificador se tienen los siguientes métodos Generar y Desencriptar

- Generar: Lo que hace es recorrer el nodoArbol y con un string e ir asignándole los valores si es a la izquierda se le asigna un 0 y si es a la derecha un 1

- Desencriptar: Lo que hace es de igual manera recorrer el nodoArbol y en lugar de asignarles valores es una igualaciones y si coincide le devuelve el valor a la letra.

B Tree

```
public class BTree <T extends Comparable<T>> {
    private static final int ORDER = 6;
        private static final int MIN_KEYS = 2;
        private static final PdfPTable tabla = new PdfPTable(new float[]{7,20,20,20,20,20,20,20});
        private static PdfPCell  titleCell;
        private static Paragraph column1,column2,column3,column4,column5,column6,column7,column8,
                                 data1,data2,data3,data4,data5,data6,data7,data8;
        private static int counterPDF = 1;
        BTreeNode root;
        Vehicle vehicle;
        public class BTreeNode {...116 lines }

        public BTree() {...3 lines }

        public T search(T k) {...4 lines }
        /**
         * Este metodo fue el mas dificultoso de todos los que se realizon,
         * Para que funcionara se ralizo un metodo llamaro find, este lo que
         * realiza es que encuentra index del nodo ingresado a eliminar esto
         * es para encontrar a los nodos hermanos que se encuentran previamente
         * o de siguientes hay 3 tipos de formas de eliminar en el metodo una
         * seria para child que esta contendria el nodo a eliminar mas el balanceo a
         * realizar, estaria de la manera nextChild que este lo que realiza es ver
         * el recorrido a la derecha y el otro seria de corriendo una rama mas
         * con el nextChild.
        */


         public void generarPDF(){...58 lines }

        private void remove(BTree<T>.BTreeNode node, T k){...182 lines }


        public void Delete(T k){...3 lines }
```

```
private T search(BTreeNode node, T k) {...18 lines }

public void searchForUpdate(T k, T k2){...4 lines }

public String GenerateReportTreeB(){...12 lines }

public String SubGrafo(){...12 lines }

/*
 * B-Tree-Insert (T, k) r = root[T] if n[r] = 2t - 1 then
 * // si la raiz esta llena, tenemos que dividirlo s = allocate-node () root[T] = s
 * // el no nueva raiz cambia su estado de leaf[s] = False // tendra algunos children n[s] = 0 // por ahora c1[s] = r
 * // el child es el viejo nodo raiz B-Tree-Split-Child (s, 1, r)
 * // r es dividido B-Tree-Insert-Nonfull (s, k)
 * // s no esta lleno else B-Tree-Insert-Nonfull (r, k) endif
 */
```

```
public void insert(T k) {...15 lines }

/*
* B-Tree-Insert-Nonfull (x, k) i = n[x]
*
* if leaf[x] then
*
* // desplazar todo a la "right" hasta el punto donde debe ir la nueva key k
*
*
* while i >= 1 and k < keyi[x] do keyi+1[x] = keyi[x] i-- end while
*
* //pegue k en su lugar correcto y se sube n[k]
*
* keyi+1[x] = k n[x]++ else
*
* // buscar child donde pertenece la nueva key:
*
* while i >= 1 and k < keyi[x] do i-- end while
*
* // si k está en ci[x], entonces k <= keyi[x] (de la definición)
* // volveremos a la última key (menos i) donde encontramos esto
* // para ser verdad, luego lea en ese nodo child.
*
* i++ Disk-Read (ci[x]) if n[ci[x]] = 2t - 1 then
*
* // si el nodo child está lleno, tendremos que dividirlo
*
* B-Tree-Split-Child (x, i, ci[x])
*
* // ahora ci [x] y ci+1[x] son los nuevos childs,
* // y keyi[x] puede haber sido cambiado.
* // Veremos si k pertenece en el primero o el segundo.
* if k > keyi[x] then i++ end if
*
* // llamamos de manera recursivamente de nuevo para hacer la inserción
*
* B-Tree-Insert-Nonfull (ci[x], k) end if
*/
```

```java
private void insertNonFull(BTree<T>.BTreeNode node, T k) {...23 lines }

/*
 * B-Tree-Split-Child (x, i, y) z = allocate-node ()
 *
 * // nuevo nodo es una leaf si el antiguo nodo era
 *
 * leaf[z] = leaf[y]
 *
 * // ya que 'y' está lleno, el nuevo nodo debe tener claves t-1
 *
 * n[z] = t - 1
 *
 * // copia sobre la "mitad derecha" de 'y' en 'z'
 *
 * para j en 1..t-1 do keyj [z] = keyj + t [y] finaliza
 *
 * // copia sobre los child secundarios si 'y' no es una leaf
 *
 * si no leaf[y] entonces para j en 1..t cj [z] = cj + t [y] final para finalmente
 *
 * // habiendo "cortado" la mitad derecha de y, ahora tiene t-1 keys
 *
 * n[y] = t - 1
 *
 * //cambia todo en x desde i + 1, luego pega el nuevo child en x;
 * // 'y' tendrá la mitad de su ser anterior como ci [x] yz lo hará
 * // ser la otra mitad como ci + 1 [x]
 * //para j en n [x] +1 hasta i + 1 a cj + 1 [x] = cj [x] final para ci + 1 = z
 *
 * // las keys también deben ser cambiadas ...
 *
 * para j en n[x] hasta i a keyj + 1 [x] = keyj [x] final para ...
 * para acomodar la nueva key que estamos trayendo desde el medio de y
 * (si se está preguntando, ya que (t-1) + (t-1) = 2t-2, donde fue la otra key, está entrando en x)
 *
 *
 * keyi[x] = keyt[y] n[x]++
 *
 * // escribir todo en el disco
 *
 * Escritura en disco (y) Escritura en disco (z) Escritura en disco (x)
 */

private void splitChild(BTreeNode x, int i, BTreeNode y) {...26 lines }
```

Grafo

```java
private Nodo Cabeza;

public Lista(){
    Cabeza = null;
}

public void Insertar(Nodo Nuevo){...11 lines }

public void ResetearTiempo(){...7 lines }

public Nodo Buscar(String Nombre){...7 lines }

public boolean Eliminar(String Eliminar){...15 lines }

public Lista Clonar(){...9 lines }

public int TiempoTotal(){...9 lines }

public void Mostrar(){...9 lines }

public String GenerarDot(){...14 lines }

public String SubGrafo(String Prefijo) {...14 lines }

public int CantidadNodos(){...9 lines }

public Nodo getCabeza() {
    return Cabeza;
}
```

```
private Lista Nodos;

public Grafo() {
    Nodos = new Lista();
}

public void InsertarNodo(Nodo Nuevo){
    Nodos.Insertar(Nuevo);
}

public boolean InsertarCamino(String First, String Second, int Tiempo){...17 lines }

public void Mostrar(){...12 lines }

public Lista CaminoMinimo(String Init, String End){...9 lines }

public Lista ListaCamino(Lista Recorrido, Nodo Actual, Nodo Final){...23 lines }

public String GenerarDot(){...20 lines }

public String SubGrafo(){...20 lines }

public Lista getNodos() {
    return Nodos;
}

public void setNodos(Lista Nodos) {
    this.Nodos = Nodos;
}
```

Blockchain

- **Insertar:** Inserta un viaje nuevo en el blockchain
- **InsertDriver:** este método sirve para verificar el reporte de conductores
- **InsertVehicles:** este método sirve para verificar el reporte de vehículos.
- **InsertCustomers;** este método sirve para verificar el reporte de clientes
- **Eliminar:** este método sirve para elementos del blockchain.

```
    public void Insertar(Viaje Nuevo){...13 lines }

    public void InsertDriver(Viaje Nuevo){...17 lines }

    public void InsertVehicle(Viaje Nuevo){...17 lines }

    public void InsertCustomers(Viaje Nuevo){...17 lines }

    public void Eliminar(String Clave){...14 lines }

    public Viaje Buscar(String Clave){...15 lines }

    public Viaje BuscarConductor(String dpi){...9 lines }

    public Viaje BuscarVehiculo(String Placa){...9 lines }

    public Viaje BuscarCliente(String dpi){...9 lines }

    public String Encriptar(String Clave) throws NoSuchAlgorithmException{...10 lines }

    public BlockChain CopyVehiclesOtherList(){...21 lines }

    public BlockChain CopyDriversOtherList(){...22 lines }

    public BlockChain CopyCustomersOtherList(){...21 lines }
    public void GenerateGraficaReportDrivers(){...39 lines }
    public void GenerateReportTopDrivers(boolean decOrcom){...43 lines }

    public void GenerarGraficaReportVehicles(){...39 lines }

    public void GenerateReportTopVehicles(boolean decOrcom){...45 lines }

    public void GenerarGraficaReportCustomers(){...40 lines }

    public void GenerateReportTopCustomers(boolean decOrcom){...45 lines }
    public void GenerarGraficaReportViajes(){...30 lines }
    public void GenerateReportTopViajes(boolean decOrcom){...26 lines }

    public void OrderListDriver(){...26 lines }

    public void OrderListVehicle(){...25 lines }

    public void OrderListCustomers(){...25 lines }
```