Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ciencias y Sistemas
Estructuras de Datos

MANUAL TÉCNICO

Henry Ronely Mendoza Aguilar
Carné: 202004810
PROYECTO FASE 3

Objetivos y Alcances del Sistema

Objetivo Principal

- Brindar la solución óptima para la generación de rutas y manejo deinformación de usuarios

Objetivos Específicos

- Diseñar una plataforma agradable e intuitiva por medio del lenguaje de programación Java.
- Aprovechar de mejor maneral los recursos brindados por el entorno de trabajo NetBeans
- Programar un sistema funcional para cualquier usuario y hardware en el que se utilice.

Requisitos del Hardware

- 512 MB de RAM
- 20MB de disco duro para almacenamiento del software (ya que puede aumentar con la generación de imágenes).
- Procesador Pentium 2 a 266Mhz

Requisitos del Software

- Java
- Graphviz
- Windows 10 (64-bit), Linux o Mac OS

Librerías Empleadas

Graphviz:

Graphviz (abreviatura de Graph Visualization Software) es un paquete de herramientas de código abierto iniciado por AT&T Labs Research para dibujar gráficos especificados en scripts de lenguaje DOT que tienen la extensión de nombre de archivo "gv". También proporciona bibliotecas para que las aplicaciones de software utilicen las herramientas. Graphviz es un software gratuito con licenciade Eclipse Public License.

Formato JSON:

JSON (JavaScript Object Notation) es un formato de archivo estándar abierto y un formato de intercambio de datos que utiliza texto legible por humanos para almacenar y transmitir objetos de datos que consisten en pares atributo-valor y arreglos (u otros valores serializables). Es un formato de datos común con diversosusos en el intercambio electrónico de datos, incluido el de aplicaciones web con servidores.

JSON es un formato de datos independiente del idioma. Se derivó de JavaScript, pero muchos lenguajes de programación modernos incluyen código para generar y analizar datos en formato JSON. Los nombres de archivo JSON usan la extensión.json.

Bcrypt:

es una función de hashing de contraseñas diseñada por Niels Provos y David Mazières, basada en el cifrado Blowfish y presentada en USENIX en 1999. Además de incorporar una sal para proteger contra los ataques de la tabla arcoíris, bcrypt es una función adaptativa: con el tiempo, el recuento de iteraciones se puede aumentar para que sea más lento, por lo que sigue siendo resistente a los ataques de búsqueda de fuerza bruta, incluso con una potencia de cálculo cada vez mayor.

La función bcrypt es el algoritmo hash de contraseña predeterminado para OpenBSD y fue el predeterminado para algunas distribuciones de Linux como SUSE Linux.

Hay implementaciones de bcrypt en C, C++, C# , Embarcadero Delphi , Elixir , Go, Java, JavaScript , [Perl , PHP , Python , [9] Ruby y otros idiomas

Estructuras de Datos

Tabla Hash

Una tabla hash, matriz asociativa, hashing, mapa hash, tabla de dispersión o tabla fragmentada es una estructura de datos que asocia *llaves* o *claves* con *valores*. La operación principal que soporta de manera eficiente es la *búsqueda*: permite el acceso a los elementos (teléfono y dirección, por ejemplo) almacenados a partir de una clave generada (usando el nombre o número de cuenta, por ejemplo). Funciona transformando la clave con una función hash en un *hash*, un número que identifica la posición (*casilla* o *cubeta*) donde la tabla hash localiza el valor deseado.

Las tablas hash se suelen implementar sobre *vectores* de una dimensión, aunque se pueden hacer implementaciones multi-dimensionales basadas en varias claves. Como en el caso de los arrays, las tablas hash proveen tiempo constante de búsqueda promedio O(1),[1] sin importar el número de elementos en la tabla. Sin embargo, en casos particularmente malos el tiempo de búsqueda puede llegar a O(*n*), es decir, en función del número de elementos.

Comparada con otras estructuras de arrays asociadas, las tablas hash son más útiles cuando se almacenan grandes cantidades de información.

Las tablas hash almacenan la información en posiciones pseudo-aleatorias, así que el acceso ordenado a su contenido es bastante lento. Otras estructuras como árboles binarios auto-balanceables tienen un tiempo promedio de búsqueda mayor (tiempo de búsqueda O(log *n*)), pero la información está ordenada en todo momento.

Grafo no dirigido

Un grafo no dirigido es un tipo de grafo en el cual las aristas representan relaciones simétricas y no tienen un sentido definido, a diferencia del grafo dirigido, en el cual las aristas tienen un sentido y por tanto no son necesariamente simétricas.

Matriz de Adyacencia

En teoría de grafos e informática, una matriz de adyacencia es una matriz cuadrada utilizada para representar un gráfico finito. Los elementos de la matriz indican si los pares de vértices son adyacentes o no en el gráfico.

En el caso especial de un gráfico simple finito, la matriz de adyacencia es una matriz (0,1) con ceros en su diagonal. Si el gráfico no está dirigido (es decir, todos sus bordes son bidireccionales), la matriz de adyacencia es simétrica. La relación entre un gráfico y los valores y los vectores propios de su matriz de adyacencia se estudia en la teoría de gráficos espectrales.

La matriz de adyacencia de un grafo debe distinguirse de su matriz de incidencia, una representación de matriz diferente cuyos elementos indican si los pares vértice-arista son incidentes o no, y su matriz de grado, que contiene información sobre el grado de cada vértice.

Descripción de Métodos

Para el desarrollo de la aplicación se emplearon los métodos de lectura de archivos de texto plano que posee Java las cuales son las pertenecientes al paquete IO. Pormedio de un JFileChooser se escogen los archivos para una interfaz gráfica.

```
FileReader fr = null;
try {
    String file="";
    File archivo=null;
    JFileChooser fc = new JFileChooser();
    int op = fc.showOpenDialog(this);
    if (op == JFileChooser.APPROVE_OPTION) {
        archivo = fc.getSelectedFile();
    }
    fr = new FileReader(archivo);
    BufferedReader br = new BufferedReader(fr);
    String linea;
    while ((linea = br.readLine()) != null) {
        file += "\n"+linea;
    }
```

Para la lectura de los archivos json se utilizó el JsonParser, JsonArray y JsonObjectcon las claves según el archivo que se ha cargado.

```
JsonParser parser = new JsonParser();
JsonArray capas = parser.parse(file).getAsJsonArray();

for(int i=0;i<capas.size();i++){
    JsonObject cap = capas.get(i).getAsJsonObject();
    int id_capa = cap.get("id_capa").getAsInt();
    JsonArray pixel = cap.get("pixeles").getAsJsonArray();
    Matriz pixeles = new Matriz();
    int fila,columna;
    String color;
    int x = 0, y=0;
    for (int j = 0; j < pixel.size(); j++) {
        JsonObject pxl = pixel.get(j).getAsJsonObject();
        fila = pxl.get("fila").getAsInt()+1;
        columna = pxl.get("columna").getAsInt()+1;
        color = pxl.get("color").getAsString();
```

Tabla Hash: Se implementó una tabla hash para el almacenamiento de los mensajeros del sistema.

```java
package edd.proyectol_fase3;

import java.io.FileWriter;
import java.io.PrintWriter;
import java.util.Objects;


public class TablaHash {
    private Mensajero arr [];
    private int M;
    private int count,ind_p;

    private final int primos[];

    public TablaHash(){
        this.primos = new int[]{37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97};
        this.M = primos[0];
        this.arr = new Mensajero[this.M];
        this.count = 0;
        this.ind_p = 0;
    }

    public Mensajero [] getTabla(){...3 lines }

    public int functionhash(Long Key){...5 lines }

    public int Colision(int i, int c){...11 lines }

    public void add(Mensajero elem){...15 lines }

    public void rehash(Mensajero elem){...25 lines }

    public Mensajero buscar(String key){...11 lines }

    public Mensajero getMensajero(int i){...3 lines }

    public void añadirPedido(int i,int p){...3 lines }

    public void TopMensajeros(){...66 lines }

    public void graph(){...43 lines }

    public void mostrar(){...13 lines }

}
```

Rutas: Para el almacenamiento de Lugares y rutas se utilizó una lista de listas

```java
public class Lugar {
    int id;
    String departamento, nombre,sucursal;
    Lista Rutas ;
    boolean visitado;

    public Lugar(int id, String departamento, String nombre, String sucursal) {
        this.id = id;
        this.departamento = departamento;
        this.nombre = nombre;
        this.sucursal = sucursal;
        this.Rutas = new Lista();
        this.visitado = false;
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getDepartamento() {
        return departamento;
    }

    public void setDepartamento(String departamento) {
        this.departamento = departamento;
    }

    public String getNombre() {
        return nombre;
    }
```

Camino mas corto: Se implementó el algoritmo de dijsktra para encontrar la ruta optima dentro de los pedidos.

```java
package edd.proyectol_fase3;

import java.util.ArrayList;
import java.util.Arrays;

public class Grafo {

    int tamaño, enlaces;
    double[] distancia;
    String[] ruta;
    ArrayList<String> vertices;
    static int[][] arr_enlaces;
    boolean[] visitados;

    public  Grafo(int n){...12 lines }

    public  void añadirVertices(String s){
        vertices.add(s);
    }

    public  void añadirVecino(int a, int b, int peso){...5 lines }

    public String [] dijkStra(int inicio, String fin){...87 lines }

}
```

Almacenamiento de datos: Se crearon diferentes clases para el almacenamiento de la información mediante la programación orientada a objetos.

```java
public class viajes {

    Lista ruta;
    double tiempo;
    String nombre;

    public viajes(Lista ruta, double tiempo, String nombre) {
        this.ruta = ruta;
        this.tiempo = tiempo;
        this.nombre = nombre;
    }
```

```java
public class Clientes {

    Long dpi;
    int id_municipio,pedidos;
    String name, password,user,correo,telefono,direccion;

    public Clientes(Long dpi, int id_municipio, String name, Stri
        this.dpi = dpi;
        this.id_municipio = id_municipio;
        this.name = name;
        this.password = password;
        this.user = user;
        this.correo = correo;
        this.telefono = telefono;
        this.direccion = direccion;
        this.pedidos = 0;
    }
```

```java
public class Mensajero {
    Long dpi;
    String nombre,apellido,licencia,genero,telefono,direccion;
    int pedidos;

    public Mensajero(Long dpi, String nombre, String apellido, String licencia, String gener
        this.dpi = dpi;
        this.nombre = nombre;
        this.apellido = apellido;
        this.licencia = licencia;
        this.genero = genero;
        this.telefono = telefono;
        this.direccion = direccion;
        this.pedidos = 0;
    }

    public int getPedidos() {
```

Uso de Graphviz

Para el uso de graphviz se utilizó la escritura de archivos con formato DOT

```java
FileWriter reporte = null;
PrintWriter pw = null;
try{
    reporte = new FileWriter("TablaHash.dot");
    pw = new PrintWriter(reporte);

    pw.println("digraph G {");
    pw.println("node [shape=\"box\"]");
    pw.println("label = \"Tabla Hash\"");
    pw.println("a0 [label=< \n <TABLE>");
    String datos = "<TR> "
                 + "<TD>indice</TD><TD>DPI</TD><TD>DPI</TD>"
                 + "</TR>";
    for (int i = 0; i < this.M; i++) {
        if(this.arr[i] != null) {
            datos += "<TR> "
                  + "<TD>"+i+"</TD><TD>"+this.arr[i].dpi+"</TD><TD>"+this.arr[i].nombre+" "+this.arr[i].apellido+"</TD>"
                  + "</TR>";
        }
    }
    datos+="</TABLE>>];";
    pw.println(datos);
    pw.println("}");

}catch(Exception e){
```