**Método de la ingeniería- Proyecto Final**

1. **Identificación del Problema**
2. La empresa requiere conocer la ruta más económica para la entrega de sus pedidos.
3. El programa debe permitir agregar o eliminar ubicaciones y calcular la ruta más corta y económica para realizar las entregas.
4. El programa debe mostrar cuales lugares ya fueron visitados (o realizado la entrega) para no volver a visitarlo o realizar la entrega.
5. El programa debe informar sobre el consumo de gasolina realizado en la ruta de entrega.
6. El programa debe tener información sobre la gasolina para poder realizar cálculos sobre los gastos de las entregas.

**Definición del problema**

Una empresa de entregas anónima de la ciudad de Cali requiere un programa economizar el consumo de combustible de sus camiones, mediante una funcionalidad que permita hallar la ruta rápida y económica entregar los pedidos. Además de eso, se requiere también que se pueda revisar desde la oficina central que pedidos se han entregado viendo en que parte de la ruta va el camión. Lo anterior con el fin de tomar datos para analizar la eficiencia de los conductores y sus ayudantes, y de esa manera tomar decisiones futuras sobre la cantidad de entregas que se deberían realizar diariamente. La empresa ha sido muy clara y especifica; quiere que el programa sea modelado por uno o varios grafos, los que se necesiten. El programa debe permitir la visualización de los camiones y sus rutas, y por consiguiente el mapa de la ciudad de Cali. Adicionalmente, se debe tener información sobre la gasolina (precio y galones usados por ruta) para calcular cuánto dinero se está invirtiendo, aproximadamente, en combustible por ruta.

Requerimientos Funcionales:

|  |  |
| --- | --- |
| Nombre | R1-Mostrar la ruta más corta para realizar las entregas |
| Resumen | Muestra la manera de entregar todas las entregas rápido y económicamente |
| Entrada | |
|  | |
| Resultado | |
| Se muestra la manera más corta y económica de realizar las entregas | |

|  |  |
| --- | --- |
| Nombre | R2-Permitir agregar más ubicaciones |
| Resumen | Agrega nuevas ubicaciones para realizar entregas |
| Entrada | |
| Nueva ubicación | |
| Resultado | |
| Se agrega la nueva ubicación | |

1. Recopilación de la información

¿Qué es un grafo?

Un **grafo** es un conjunto de objetos llamados nodos o **vértices**, que pueden estar unidos (conectados) por líneas llamada **aristas**.

¿Qué es un vértice?

un vértice o nodo es la unidad fundamental de la que están formados los grafos. Un grafo no dirigido está formado por un conjunto de vértices y un conjunto de aristas (pares no ordenados de vértices), mientras que un grafo dirigido está compuesto por un conjunto de vértices y un conjunto de arcos.

¿Qué es una arista?

una arista corresponde a una relación entre dos vértices de un grafo.

Para caracterizar un grafo G son suficientes únicamente el conjunto de todas sus aristas, comúnmente denotado con la letra E junto con el conjunto de sus vértices, denotado por V. Así, dicho grafo se puede representar como G(V,E), o bien G = (V,E).

¿Qué es BFS?

es un algoritmo de búsqueda no informada utilizado para recorrer o buscar elementos en un grafo. Intuitivamente, se comienza en la raíz (eligiendo algún nodo como elemento raíz en el caso de un grafo) y se exploran todos los vecinos de este nodo

¿Qué es DFS?

es un algoritmo de búsqueda no informada utilizado para recorrer todos los nodos de un grafo o árbol (teoría de grafos) de manera ordenada, pero no uniforme. Su funcionamiento consiste en ir expandiendo todos y cada uno de los nodos que va localizando, de forma recurrente, en un camino concreto

¿Qué es Dijkstra?

es un algoritmo para la determinación del camino más corto, dado un vértice origen, hacia el resto de los vértices en un grafo que tiene pesos en cada arista. Su nombre alude a Edsger Dijkstra, científico de la computación de los Países Bajos que lo describió por primera vez en 1959.

¿Qué es Floyd-Warshall?

es un algoritmo de análisis sobre grafos para encontrar el camino mínimo en grafos dirigidos ponderados. El algoritmo encuentra el camino entre todos los pares de vértices en una única ejecución.

¿Qué es Prim?

es un algoritmo perteneciente a la teoría de los grafos para encontrar un árbol recubridor mínimo en un grafo conexo, no dirigido y cuyas aristas están etiquetadas.

¿Qué es Kruskal?

es un algoritmo de árbol de expansión mínima que encuentra un borde del menor peso posible que conecta dos árboles del bosque.

1. Búsqueda soluciones creativas
2. Transición de las ideas a los diseños preliminares.
3. Evaluación y selección de la mejor solución
4. Preparación de informes y especificaciones

|  |
| --- |
| TAD Graph |
| Representation: |
| Invariat:  No direccionado |
| Operations:  Grafo Graph  AddVert Vertex  AddArist Vertex, Vertex, Integer  getVert Vertex getVerts  DeleVert Vertex  DeleArist Arist  BFS Vertex  DFS Vertex |
|  |

|  |
| --- |
| AddVert(Vertex) |
| Adds a Vertex to the graph. Pre: void Post: V = V + 1 |

|  |
| --- |
| AddArit(Vertex,Vertex,Integer) |
| Adds a weighted edge from one vertex to another. Pre: void Post: Graph.weight = Graph.weight + e.weight |

|  |
| --- |
| getVert (Vertex) |
| Gives the information of a Vertex. Pre: Vertex  Graph ᴧ Vertex ≠ nil Post: Vertex |

|  |
| --- |
| getVerts () |
| Gives the list of the vertexes of the graph. Pre: Vertex(i)  Graph ᴧ Vertex(i) ≠ nil Post: Vertex {V1,V2...Vn} |

|  |
| --- |
| DeleVert(Vertex) |
| Deletes a Vertex of the graph. Pre:  Vertex  Graph ᴧ  Vertex ≠ nil Post: V  = V - 1 |

|  |
| --- |
| DeleArist (Edge) |
| Deletes an edge of the graph. Pre: Edge  Graph ᴧ Edge ≠ nil  Post: Graph.weight |

= Graph.weight - e.weight

|  |
| --- |
| BFS(Vertex) |
| Explores all the neighbor nodes of a vertex selected as root node. Pre: Vertex  Graph ᴧ Vertex ≠ nil Post:  void |

|  |
| --- |
| DFS(Vertex) |
| Explores the possible routes as far as possible of a vertex selected as root node. Pre: Vertex  Graph ᴧ Vertex ≠ nil Post:  void |

-Diseño de pruebas unitarias

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| P.1 | Objetivo de la prueba= comprobar si se agrega un vértice | | | |
| Clase | Método | Escenario | Valores de entrada | Resultado |
| Graph | AddVert(Vertex) | Escenario 1 |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| P.2 | Objetivo de la prueba= comprobar si se agrega un arista | | |  |
| Clase | Método | Escenario | Valores de entrada | Resultado |
| Graph | AddArit(Vertex,Vertex,Integer) | Escenario 1 |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| P.3 | Objetivo de la prueba= comprobar si retorna el vértice | | |  |
| Clase | Método | Escenario | Valores de entrada | Resultado |
| Graph | getVert (Vertex) | Escenario 1 |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| P.4 | Objetivo de la prueba= comprobar si retorna los vértices | | | |
| Clase | Método | Escenario | Valores de entrada | Resultado |
| Graph | getVerts () | Escenario 1 |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| P.5 | Objetivo de la prueba= comprobar si elimina un vértice | | |  |
| Clase | Método | Escenario | Valores de entrada | Resultado |
| Graph | DeleVert(Vertex) | Escenario 1 |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| P.6 | Objetivo de la prueba= comprobar si elimina una arista | | |  |
| Clase | Método | Escenario | Valores de entrada | Resultado |
| Graph | DeleArist (Edge) | Escenario 1 |  |  |

1. Implementación del diseño

Enlace al repositorio de GitHub: <https://github.com/Sergiiok/FinalProjectAED>