Sergio Martinez A00045552

Método de la ingeniería-Proyecto Final Empresa Móvil Entregas

Una empresa móvil que realiza visitas para mostrar sus productos y lleva productos a domicilio por varios puntos de la ciudad. Esta empresa al analizar los costos que se lleva realizando las rutas de visita y entrega se dio que el valor es alto, entonces está buscando una solución para reducir el costo sin dejar de realizar las rutas completa, y también que si se añade un nuevo domicilio no genere un gasto muy grande. Entonces requieren una aplicación que les indiquen la ruta mas corta y con menos costo visitando todas las personas que solicitaron servicio de visita o domicilio.

1.Identificaicon del problema

a. ruta más corta

b. si se añade una nueva vistita, la ruta actualizada más corta

c. la ruta mas corta y con menos costo

d. el costo de todas las rutas posibles

Definición del problema: Se requiere una aplicación que indique cual es la ruta mas corta y que tenga menos valor(gasolina) en la ruta de visitas a realizar y de domicilios que ya estén y que salgan mediante se va realizando la ruta.

Requerimientos funcionales.

Nombre	**R1-Generar la ruta más corta**
Resumen	Genera un árbol con la ruta mas corta de todos los vértices
Entrada	
Grafo de rutas	
Salida	
Árbol con la ruta más corta	

Nombre	**R2- Añadir un nuevo lugar**
Resumen	Añade un vértice como lugar a visitar
Entrada	
Lugar Por Insertar	
Salida	

Si se añadió o no el lugar

Nombre	**R3-El costo de todas las rutas posible**
Resumen	Da una lista con todas las rutas posibles con su respectivo costo
Entrada	
Grafo	
Salida	
Lista con los costos de las rutas	

Nombre	**R4-**
Resumen	
Entrada	
Ninguna	
Salida	

2. Recopilación de la información.

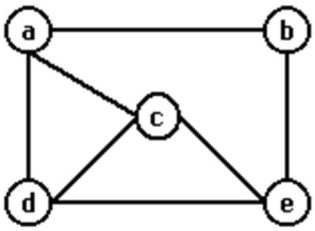
Desde un punto de vista intuitivo un grafo es un conjunto de nodos unidos por un conjunto de arcos. Un ejemplo de grafo que podemos encontrar en la vida real es el de un plano de trenes. El plano de trenes está compuesto por varias estaciones (nodos) y los recorridos entre las estaciones (arcos) constituyen las líneas del trazado.

Veremos a continuación una definición más formal de grafos. Un **grafo** G=(V,E) consiste en un conjunto V de **nodos** (vértices) y un conjunto E de **aristas** (arcos). Cada arista es un par (v,w), siendo v y w un par de nodos pertenecientes al conjunto V de nodos. Podemos distinguir entre grafos dirigidos y no dirigidos. En un **grafo dirigido** los pares (v,w) están ordenados, traduciéndose la arista en una flecha que va desde el nodo **v** al nodo **w**.

En el caso de un grafo no dirigido, los nodos están unidos mediante líneas sin indicación de dirección.

V = { a, b, c, d, e}

E = { (a, b), (a, c), (a, d), (b, e),
(c, d), (c, e), (d, e) }

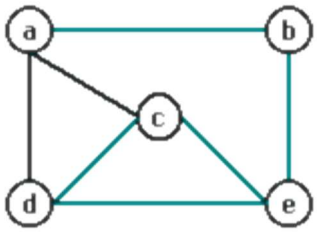


Por último, se puede definir una función que asocie a cada arco un coste: **coste(arco)**

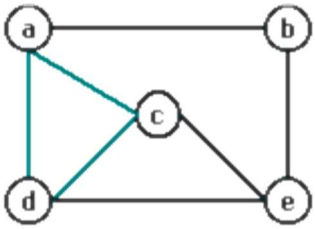
Estudiaremos a continuación algo de terminología común en los grafos.

Hablaremos de dos vértices **adyacentes** cuando estén unidos por un arco. El número de vértices adyacentes de un nodo constituye el **grado** de este. En el ejemplo los vértices adyacentes al nodo 3 son el 1, 4 y 5, siendo éste por tanto un nodo de grado tres por tener tres vértices adyacentes.

Un **camino** entre dos vértices es una secuencia de vértices tal que dos vértices consecutivos son adyacentes. En el siguiente ejemplo el camino entre el vértice **a** y el vértice **e** será la secuencia de vértices **abecde**.



Cuando este camino no tiene vértices repetidos se dice que es **simple**. Salvo en el caso de que el primer y último vértice del camino sean el mismo, en cuyo caso hablaremos de un **ciclo**



La siguiente clasificación, aunque no es completa, presenta las principales características que nos podemos encontrar en los grafos:

- **Grafo conexo**: Cuando entre cada dos nodos del grafo hay un camino.
- **Bosque**: Es un grafo sin ciclos.
- **Árbol libre**: es un bosque conexo.

La representación más extendida de los grafos es mediante lo que se llaman **Matrices de adyacencia**. Si el número de vértices del grafo es **N**, la matriz de adyacencia es una matriz tal que:

$A_{N \times N} = (a_{ij}) \quad i, j \in \{1..N\} \quad a_{ij} = \{n^o \text{ de arcos que hay entre } i \text{ y } j\}$

Mediante esta representación, la matriz de adyacencia del ejemplo sería

	a	b	c	d	e
a	0	1	1	1	0
b	1	0	0	0	1
c	1	0	0	1	1
d	1	0	1	0	1
e	0	1	1	1	0

Fuente: https://www.ciberaula.com/cursos/java/grafos_java.php

Búsqueda en anchura (en inglés *BFS - Breadth First Search*) es un algoritmo de búsqueda no informada utilizado para recorrer o buscar elementos en un grafo (usado frecuentemente sobre árboles). 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. A continuación para cada uno de los vecinos se exploran sus respectivos vecinos adyacentes, y así hasta que se recorra todo el árbol.

Fuente: https://es.wikipedia.org/wiki/B%C3%BAsqueda_en_anchura

Una **Búsqueda en profundidad** (en inglés DFS o *Depth First Search*) 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. Cuando ya no quedan más nodos que visitar en dicho camino, regresa (Backtracking), de modo que repite el mismo proceso con cada uno de los hermanos del nodo ya procesado.

Fuente: https://es.wikipedia.org/wiki/B%C3%BAsqueda_en_profundidad

El algoritmo de Dijkstra, también llamado algoritmo de caminos mínimos, 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

Fuente: https://es.wikipedia.org/wiki/Algoritmo_de_Dijkstra

el algoritmo de Floyd-Warshall, descrito en 1959 por Bernard Roy, 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. El algoritmo de Floyd-Warshall es un ejemplo de programación dinámica.

Fuente: https://es.wikipedia.org/wiki/Algoritmo_de_Floyd-Warshall

El algoritmo de 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.

En otras palabras, el algoritmo encuentra un subconjunto de aristas que forman un árbol con todos los vértices, donde el peso total de todas las aristas en el árbol es el mínimo posible. Si el grafo no es conexo, entonces el algoritmo encontrará el árbol recubridor mínimo para uno de los componentes conexos que forman dicho grafo no conexo.

Fuente: https://es.wikipedia.org/wiki/Algoritmo_de_Prim#C%C3%B3digo_en_JAVA

El algoritmo de Kruskal es un algoritmo de la teoría de grafos para encontrar un árbol recubridor mínimo en un grafo conexo y ponderado. Es decir, busca un subconjunto de aristas que, formando un árbol, incluyen todos los vértices y donde el valor de la suma de todas las aristas del árbol es el mínimo. Si el grafo no es conexo, entonces busca un bosque expandido mínimo (un árbol expandido mínimo para cada componente conexa). Este algoritmo toma su nombre de Joseph Kruskal, quien lo publicó por primera vez en 1956.12. Otros algoritmos que sirven para hallar el árbol de expansión mínima o árbol recubridor mínimo es el algoritmo de Prim, el algoritmo del borrador inverso y el algoritmo de Boruvka.

Fuente: https://es.wikipedia.org/wiki/Algoritmo_de_Kruskal

3.Busqueda de Soluciones Creativas

Alternativa 1:

- Usar el algoritmo Dijkstra para encontrar el camino mas coroto en el grafo de lugares a realizar la entrega.

Alternativa 2:

- Usar el algoritmo Prim para encontrar el árbol de recubrimiento mínimo para así saber la ruta más corta y con menos gasto en el grafo

Alternativa 3:

- Usar el algoritmo de Kruskal para encontrar el encontrar los diferentes caminos en el grafo no conexo

Alternativa 4:

- Usar el algoritmo Prim para encontrar el árbol de recubrimiento mínimo para así saber la ruta más corta y con menos gasto en el grafo
- Usar el algoritmo de Kruskal para encontrar el encontrar los diferentes caminos en el grafo no conexo

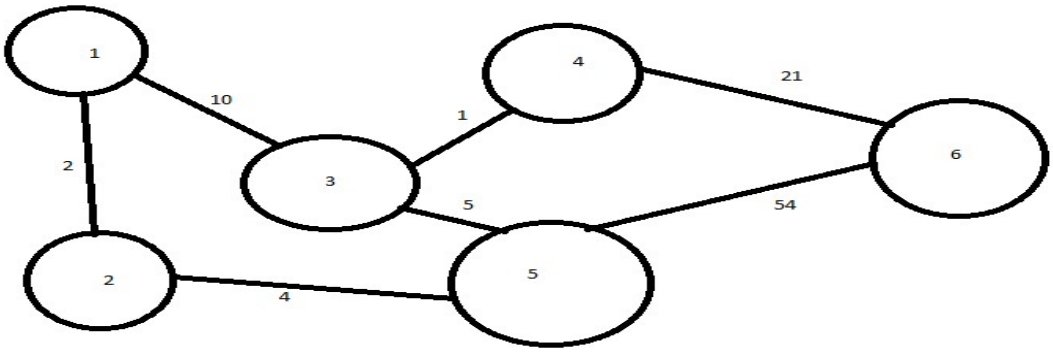
4. Transición de las ideas a los diseños preliminares.

5. Evaluación y selección de la mejor elección.

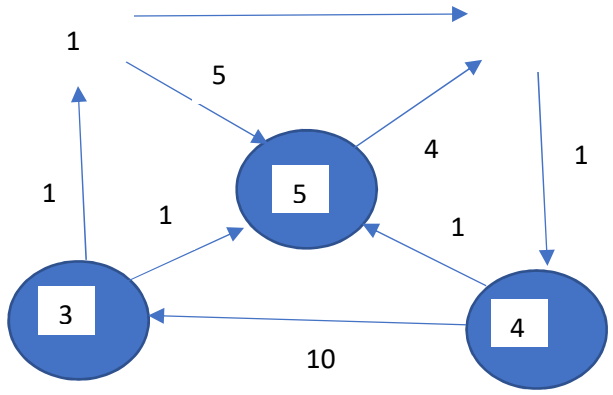
En este caso, se seleccionó la alternativa 4 como mejor solución antes las especificaciones del cliente. Además, nótese que la alternativa 4 es la única que da una solución eficaz al problema a solucionar.

6. Preparación de informes y especificaciones

TAD:

TAD GRAPH
Representación: 

Sergio Martinez A00045552



Invariante:

{E1 (Vj,Vk); 0<=j<=k}

Operations:

+CreateGraph: -----→ Graph
+addVertex:Graph x Vertex ----→ Graph
+addEgde: Graph x Vertex x Vertex
+removeVertex: Graph x Vertex
+removeEgde: Graph x Egde
+vertexAdjacent: Graph x Vertex
+areConnected: Graph x Vertex x Vertex

creatGraph()

“Crea un nuevo grafo vacio”

addVertex(Graph,Vertex)
añade un vertice al grafo

pre: Graph!=null^V1!=null

addEgde(Graph,Vertex,Vertex)
añade una arista entre los dos vertices
pre: Graph!=null^V1!=null^V2!=null

Sergio Martinez A00045552

removesVertex(Graph,Vertex) se eleimina un vertice del grafo pre: V1!=null^V2!=null

removesEgde(Graph,Egde) se elimina la arista así desconectando dos vertices

vertexAdjacent(Graph,Vertex) retorna una lista de vértices que contiene los vértices adyacentes a este vertice pre: Graph!=null^V1!=null

areConected(Graph,Vertex,Vertex) indica si dos vértices esta conectados pre: Graph!=null^V1!=null ^V2!=null

-Diseño de casos de prueba

Graph

Escenario 1: se crea un grafo simple con los siguiente vértices y arista:

(a,b),(c,d),(a,e),(b,d),(a,c),(e,d)

Prueba no.1	Objetivo: Comprobar si se crea un grafo vacio			
Clase	Método	Escenario	Valores de Entrada	Valore de Salida
Graph	createGraph()	1	0	Graph vacio

Prueba no.2	Objetivo: Comprobar si se añade un vertice al grafo			
Clase	Método	Escenario	Valores de Entrada	Valore de Salida
Graph	addVertex ()	1	f	Graph con el nuevo vertice F

Prueba no.3	Objetivo: Comprobar si se añade una arista entre dos vertices			
Clase	Método	Escenario	Valores de Entrada	Valore de Salida
Graph	addEgde ()	1	D,f	Graph con la arista entre d y f

7. implementación del diseño

Enlace al repositorio de GitHub: https://github.com/Sergiiok/ProyectFinal