Contexto problemático:

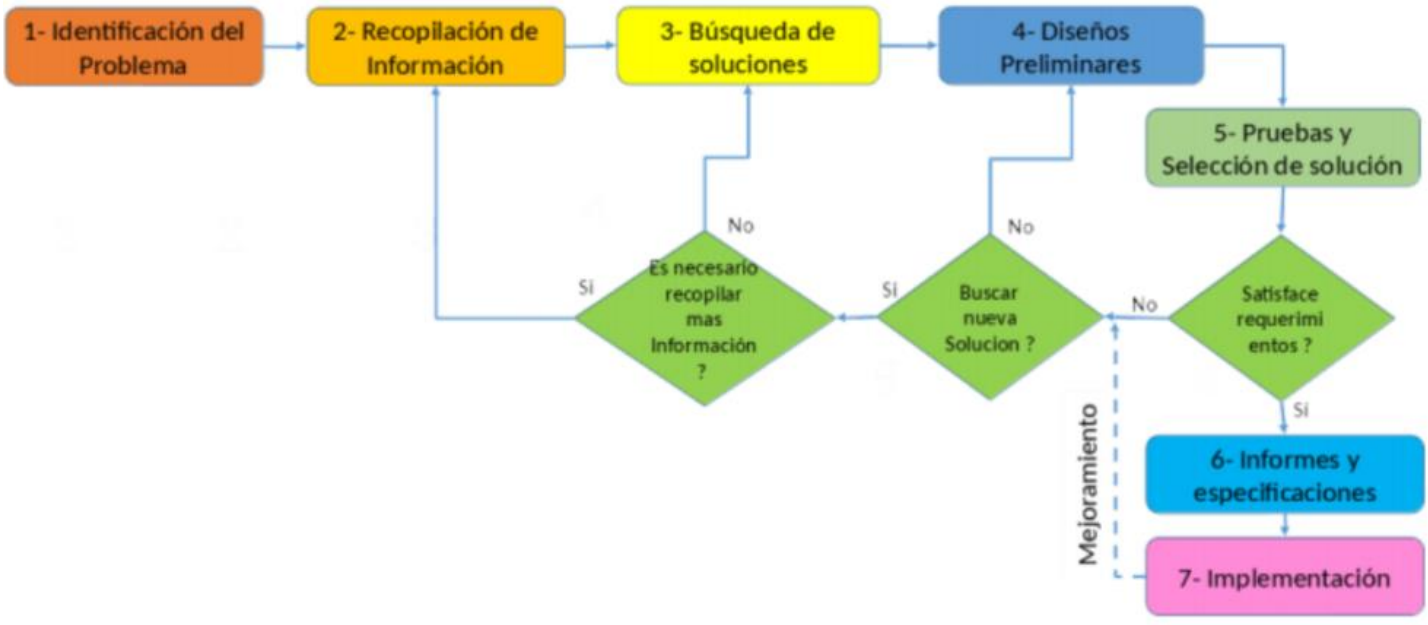
Rappi, una compañía multinacional que ha estado en auge desde 2015 como una de las mejores compañías de comercio y distribución electrónica, se ha enfrentado a una serie de problemas en este último año que han generado inconformidades en sus clientes, estas, con respecto al servicio que la empresa está prestando; una de las causas de la inconformidad de los clientes radica en el tiempo de entrega de cada uno de los productos.

Es por esto que Rappi ha tomado la decisión de contratar a un grupo de jóvenes desarrolladores de software para llevar a cabo la creación de una nueva aplicación que le permita a los encargados del domicilio cumplir con las entregas de manera eficaz; y así ir mejorando su servicio y mermar las inconformidades de los clientes.

Desarrollo de la solución:

Para resolver el caso problemático anterior se eligió el método de la ingeniería del libro “Introduction to Engineering” de Paul Wright.

Los pasos para conseguir el desarrollo de la solución se describen en el siguiente diagrama de flujo:



Paso 1. Identificación del problema:

El problema que nos presenta el cliente lo hemos acogido y hemos encontrado los siguientes requerimientos, que, al ser solucionados, se entregará la aplicación solicitada por el cliente.

REQUERIMIENTO FUNCIONAL # 1	
Nombre	Mostrar la ruta con la menor distancia
Resumen	Se le mostrará al usuario la ruta que tiene la menor distancia para desplazarse de un punto a otro.
Entrada	
Destino de salida del pedido y destino de llegada del usuario.	
Resultado	
Se le muestra al usuario la ruta que recorre la menor distancia para llegar a su destino.	

REQUERIMIENTO FUNCIONAL # 2	
Nombre	Mostrar el recorrido más corto de un sector.
Resumen	Se le mostrará al usuario la ruta que debe seguir para realizar el recorrido de un sector con el menor costo posible.
Entrada	
Delimitaciones del sector.	
Resultado	
Se le mostrará al usuario la ruta que tenga el menor costo para recorrer un sector determinado.	

REQUERIMIENTO FUNCIONAL # 3	
Nombre	Mostrar rutas disponibles.
Resumen	Se le mostrará al usuario las rutas disponibles para llegar a su destino.
Entrada	
Destino de salida del pedido y destino de llegada del usuario.	
Resultado	
Se le mostrará al usuario posible rutas que puede usar para llegar a un destino en específico.	

Paso 2. Recopilación de información:

Para poder dar solución al problema, primero debemos entrar en materia con el tema pedido y algunos conceptos necesarios para dar solución a él.

Recopilamos la siguiente información con el fin de aclarar todas las dudas propuestas por nuestro grupo de trabajo respecto al caso problemático:

Rappi: es una compañía multinacional colombiana de comercio electrónico, con sede principal en Bogotá, Colombia. Activa en México, Brasil, Uruguay, Argentina, Chile, Perú y Colombia, país donde fue fundada en 2015 por Felipe Villamarin, Sebastián Mejía y Simón Borrero (CEO de la compañía). Más de la mitad de sus ventas toman lugar en la Zona Metropolitana de la Ciudad de México.

Uno de las características que define Rappi es la gama ancha de los productos y los servicios disponibles para la entrega — el co-fundador Sebastián Mejía dice "queremos ser una 'tienda de todo'". La aplicación móvil permite a los consumidores pedir el mandado del supermercado, comida, y medicamentos de farmacias, pero también permite enviar dinero en efectivo a alguien, o que un corredor retire dinero de su cuenta bancaria de un cajero automático y entregárselo. Los corredores hasta pueden pasear los perros del cliente.

Su interfaz de usuario luce como estantes de supermercado y los usuarios pueden deslizar productos a su cesta. Los clientes pueden pagar en efectivo o vía tarjetas de débito y de crédito. Su plataforma incorpora Grability, una plataforma de comercio electrónico utilizada por otros detallistas como Walmart, El Corte Inglés, Grupo Exito y Cencosud. La inversión inicial en Rappi era de 2 millones de USD, y más tarde Rappi se metió a Y-Combinator lo que generó inversión adicional.

En septiembre de 2018, Rappi logró una valoración de los US$1 000 millones, convirtiéndose así en el primer "unicornio" de Colombia.

Ruta: Se trata de un camino, carretera o vía que permite transitar desde un lugar hacia otro. En el mismo sentido, una ruta es la dirección que se toma para un propósito. En las comunicaciones, la ruta de enlaces es el conjunto de puntos que permiten unir dos puntos extremos.

Ahora traeremos los conceptos vistos en clase que nos pueden ayudar a solucionar el problema:

Árbol AVL: Un árbol AVL es un árbol binario de búsqueda que cumple con la condición de que la diferencia entre las alturas de los subárboles de cada uno de sus nodos es, como mucho 1 o -1. La denominación de árbol AVL viene dada por los creadores de tal estructura (Adelson-Velskii y Landis). Recordamos que un árbol binario de búsqueda es un árbol binario en el cual cada nodo cumple con que todos los nodos de su subárbol izquierdo son menores que la raíz y todos los nodos del subárbol derecho son mayores que la raíz. Recordamos también que el tiempo de las operaciones sobre un árbol binario de búsqueda son O(log n) promedio, pero el peor caso es O(n), donde n es el número de elementos. La propiedad de equilibrio que debe cumplir un árbol para ser AVL asegura que la profundidad del árbol sea O(log(n)), por lo que las operaciones sobre estas estructuras no deberán recorrer mucho para hallar el elemento deseado. El tiempo de ejecución de las operaciones sobre estos árboles es, a lo sumo O(log(n)) en el peor caso, donde n es la cantidad de elementos del árbol. Sin embargo, y como era de esperarse, esta misma propiedad de equilibrio de los árboles AVL implica una dificultad a la hora de insertar o eliminar elementos: estas operaciones pueden no conservar dicha propiedad.

Teoría de Grafos: En matemáticas y en ciencias de la computación, la teoría de grafos (también llamada teoría de las gráficas) estudia las propiedades de los grafos (también llamadas gráficas). Un grafo es un conjunto, no vacío, de objetos llamados vértices (o nodos) y una selección de pares de vértices, llamados aristas (edges en

inglés) que pueden ser orientados o no. Típicamente, un grafo se representa mediante una serie de puntos (los vértices) conectados por líneas (las aristas).

Los grafos se pueden representar de diferentes maneras;

• **Lista de incidencia -** Las aristas son representadas con un vector de pares (ordenados, si el grafo es dirigido), donde cada par representa una de las aristas.

• **Lista de adyacencia -** Cada vértice tiene una lista de vértices los cuales son adyacentes a él. Esto causa redundancia en un grafo no dirigido (ya que A existe en la lista de adyacencia de B y viceversa), pero las búsquedas son más rápidas, al costo de almacenamiento extra.

• **Matriz de incidencia -** El grafo está representado por una matriz de A (aristas) por V (vértices), donde [arista, vértice] contiene la información de la arista (1 - conectado, 0 - no conectado)

• **Matriz de adyacencia -** El grafo está representado por una matriz cuadrada M de tamaño $\boldsymbol{n^2}$, donde $\boldsymbol{n}$ es el número de vértices. Si hay una arista entre un vértice x y un vértice y, entonces el elemento $\boldsymbol{m_{x,y}}$ es 1, de lo contrario, es 0.

Camino: Se denomina camino (algunos autores lo llaman cadena si se trata de un grafo no dirigido)en un grafo dirigido a una sucesión de arcos adyacentes: $C=\{(v_1,v_2),(v_2,v_3),\ldots,(v_{n-1},v_n),\ v_i \in V\}$.

Circuito: Un circuito (o ciclo para grafos no dirigidos) es un camino en el que coinciden los vértices inicial y final, un circuito se dice simple cuando todos los arcos que lo forman son distintos y se dice elemental cuando todos los vértices por los que pasa son distintos. La longitud de un circuito es el número de arcos que lo componen. Un bucle es un circuito de longitud 1.

Algoritmos de recorrido: En teoría de grafos existen dos formas de hacer recorridos dentro del grafo: **Algoritmos de Recorrido en Anchura** (BFS, por sus siglas en inglés: “Breadth-First Search”) y de **Recorrido en Profundidad** (DFS, por

sus siglas en inglés: “Depth-First Search”). Ambas técnicas constituyen métodos sistemáticos para visitar todos los vértices y arcos del grafo, exactamente una vez y en un orden específico predeterminado, por lo cual podríamos decir que estos algoritmos simplemente nos permiten hacer recorridos controlados dentro del grafo con algún propósito. Siendo la búsqueda una de las operaciones más sencillas y elementales en cualquier estructura de datos, se han estandarizado el uso de estos algoritmos para ello, por lo que se conocen como algoritmos de búsqueda. Sin embargo, es importante resaltar que pueden utilizarse para muchísimas otras operaciones con grafos que no necesariamente incluyan la búsqueda de algún elemento dentro del grafo. Ambas técnicas constituyen métodos sistemáticos para visitar todos los vértices y arcos del grafo, exactamente una vez y en un orden específico

Árbol de expansión: Dado un grafo conexo, no dirigido G. Un árbol de expansión es un árbol compuesto por todos los vértices y algunas (posiblemente todas) de las aristas de G. Al ser creado un árbol no existirán ciclos, además debe existir una ruta entre cada par de vértices.

Algoritmo de Prim: Es un algoritmo perteneciente a la teoría de los grafos para encontrar un árbol de recubrimiento 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 de recubrimiento mínimo para uno de los componentes conexos que forman dicho grafo no conexo. Cabe tener en cuenta, que el algoritmo de prim necesita un nodo (Vértice) inicial.

El algoritmo fue diseñado en 1930 por el matemático Vojtech Jarnik y luego de manera independiente por el científico computacional Robert C. Prim en 1957 y

redescubierto por Dijkstra en 1959. Por esta razón, el algoritmo es también conocido como algoritmo DJP o algoritmo de Jarnik.

Algoritmo de Kruskal: Cumple la misma función del algoritmo de Prim, pero este no necesita un nodo inicial, este ordena sus aristas y empieza a conectar una por una hasta obtener el árbol de recubrimiento mínimo.

Algoritmo de Dijkstra: También llamado algoritmo de caminos mínimos, es un algoritmo para la determinación del camino más corto dado un vértice origen al resto de vértices en un grafo con pesos en cada arista. Su nombre se refiere a Edsger Dijkstra, quien lo describió por primera vez en 1959.

Algoritmo de Floyd-Warshall: Es un algoritmo de análisis sobre grafos que permite encontrar el camino mínimo en grafos dirigidos ponderados. El algoritmo encuentra el camino entre todos los pares de vértices en una única ejecución, constituyendo un ejemplo de programación dinámica.

Fuentes:

https://es.wikipedia.org/wiki/Rappi

https://edukavital.blogspot.com/2013/01/definicion-de-ruta-compendio-de.html

http://es.tldp.org/Tutoriales/doc-programacion-arboles-avl/avl-trees.pdf

http://www.unipamplona.edu.co/unipamplona/portalIG/home_23/recursos/general/11072012/grafo3.pdf

http://decsai.ugr.es/~jfv/ed1/c++/cdrom4/paginaWeb/grafos.htm

https://www.bibliadelprogramador.com/2014/04/algoritmos-de-busqueda-en-anchura-bfs-y.html

https://jariasf.wordpress.com/2012/04/19/arbol-de-expansion-minima-algoritmo-de-kruskal/

https://www.ecured.cu/Algoritmo_de_Prim

https://www.ecured.cu/Algoritmo_de_Dijkstra

https://www.ecured.cu/Floyd-Warshall

Paso 3. Búsqueda de soluciones creativas:

En este paso, visitamos varios sitios web y lecturas propuestas por nuestro propio instructor del curso de estructuras con mucha información donde encontramos varias alternativas que de una u otra forma dan solución al problema.

La estructura vista en clase adecuada para dar solución al problema es el grafo, entonces investigaremos varios métodos y aplicaciones que se derivan de los grafos, ya que estas sub ramas de los grafos son las que nos ayudarán a resolver el problema.

- **Algoritmo de recorrido en anchura:**
  Recorrer un grafo consiste en “visitar” cada uno de los nodos a través de las aristas del mismo. Se trata de realizar recorridos de grafos de manera eficiente. Para ello, se pondrá una marca en un nodo en el momento en que es visitado, de tal manera que, inicialmente, no está marcado ningún nodo del grafo. Detallaremos el proceso para grafos no dirigidos, observando que para el caso dirigido el proceso es análogo, solo cambia el significado del concepto de adyacencia. A continuación, describimos la idea de un procedimiento recursivo para realizar un recorrido en anchura de un grafo no dirigido G. Para ello, en primer lugar, a cada nodo, v, del grafo se le asocia

un procedimiento, BFS(G, v), que se denomina recorrido en anchura de G con origen v. La idea del procedimiento BFS(G, v) es la siguiente:

- Se marca el nodo v.
- Si todos los nodos adyacentes a v están marcados, entonces TERMINAR; si no, marcar todos los nodos v1, v2, . . ., vk adyacentes a v que no estén marcados.
- Repetir el proceso con los nodos adyacentes a los nodos que se han marcado en el paso anterior.

Si G es conexo (es decir, si dos vértices cualesquiera de G siempre están conectados por un camino), entonces BFS(G, v) visita todos los nodos y aristas del grafo.

- **Algoritmo de recorrido en profundidad:**

  El algoritmo DFS posee varias aplicaciones la más importante es para problemas de conectividad, si un grafo es conexo, detectar ciclos en un grafo, número de componentes conexas, etc. Y es bastante útil en otros algoritmos como para hallar las componentes fuertemente conexas en un grafo (Algoritmo de Kosaraju, Algoritmo de Tarjan), para hallar puntos de articulación o componentes biconexas (puentes), para recorrido en un circuito o camino euleriano, topological sort, flood fill y otras aplicaciones.

  DFS va formando un árbol al igual que BFS, pero lo hace a profundidad. Existen dos formas de hacer el recorrido una es usando una Pila y otra de manera recursiva:

  Usando Stack: El concepto es el mismo que BFS solo que se cambia la Cola por una Pila, el proceso es como sigue: visitar el nodo inicial y ponerlo en la pila, ahora para ver los siguientes nodos a visitar sacamos el nodo tope de la pila y vemos sus adyacentes, los que no han sido visitados los insertamos en la pila. El proceso se repite hasta que la pila se encuentre vacía (se han visitado todos los nodos).

- **Algoritmo de prim:**

  El algoritmo incrementa continuamente el tamaño de un árbol, comenzando por un vértice inicial al que se le van agregando sucesivamente vértices cuya distancia a los anteriores es mínima. Esto significa que, en cada paso, las aristas a considerar son aquellas que inciden en vértices que ya pertenecen al árbol. El árbol de recubrimiento mínimo está completamente construido cuando no quedan más vértices por agregar.

  El algoritmo podría ser informalmente descrito siguiendo los siguientes pasos:

  1. Inicializar un árbol con un único vértice, elegido arbitrariamente del grafo.
  2. Aumentar el árbol, por un lado. Llamamos lado a la unión entre dos vértices: de las posibles uniones que pueden conectar el árbol a los vértices que no están aún en el árbol, encontrar el lado de menor distancia y unirlo al árbol.
  3. Repetir el paso 2 (hasta que todos los vértices pertenezcan al árbol)

  Para hacerlo más en detalle, debe ser implementado el pseudocódigo siguiente:

  1. Asociar con cada vértice *v* del grafo un número *C[v]* (el mínimo coste de conexión a *v*) y a un lado *E[v]* (el lado que provee esa conexión de mínimo coste). Para inicializar esos valores, se establecen todos los valores de *C[v]* a +∞ (o a cualquier número más grande que el máximo tamaño de lado) y establecemos cada *E[v]* a un valor "flag"(bandera) que indica que no hay ningún lado que conecte *v* a vértices más cercanos.
  2. Inicializar un bosque vacío *F* y establecer *Q* vértices que aún no han sido incluidos en *F* (inicialmente, todos los vértices).
  3. Repetir los siguientes pasos hasta que *Q* esté vacío:
     1. Encontrar y eliminar un vértice *v* de *Q* teniendo el mínimo valor de *C[v].*

2. Añadir *v* a *F* y, si *E[v]* no tiene el valor especial de "flag", añadir también *E[v]* a *F.*
3. Hacer un bucle sobre los lados *vw* conectando *v* a otros vértices *w*. Para cada lado, si *w* todavía pertenece a *Q* y *vw* tiene tamaño más pequeño que *C[w]*, realizar los siguientes pasos:
    1. Establecer *C[w]* al coste del lado *vw.*
    2. Establecer *E[w]* apuntando al lado *vw.*
4. Devolver F.

Como se ha descrito arriba, el vértice inicial para el algoritmo será elegido arbitrariamente, porque la primera iteración del bucle principal del algoritmo tendrá un número de vértices en *Q* que tendrán todos el mismo tamaño, y el algoritmo empezará automáticamente un nuevo árbol en *F* cuando complete un árbol de expansión a partir de cada vértice conectado del grafo. El algoritmo debe ser modificado para empezar con cualquier vértice particular *s* para configurar *C[s]* para que sea un número más pequeño que los otros valores de *C* (por norma, cero), y debe ser modificado para sólo encontrar un único árbol de expansión y no un bosque entero de expansión, parando cuando encuentre otro vértice con "flag" que no tiene ningún lado asociado.

Hay diferentes variaciones del algoritmo que difieren unas de otras en cómo implementar *Q*: Como una única Lista enlazada o un vector de vértices, o como una estructura de datos organizada con una cola de prioridades, más compleja. Esta elección lidera las diferencias en complejidad de tiempo del algoritmo. En general, una cola de prioridades será más rápida encontrando el vértice *v* con el mínimo coste, pero ello conllevará actualizaciones más costosas cuando el valor de *C[w]* cambie.

- **Algoritmo de kruskal:**

  El algoritmo de Kruskal es un ejemplo de algoritmo voraz que funciona de la siguiente manera:

  - Se crea un bosque *B* (un conjunto de árboles), donde cada vértice del grafo es un árbol separado.
  - Se crea un conjunto *C* que contenga a todas las aristas del grafo.
  - Mientras *C* es *no vacío:*
    - Eliminar una arista de peso mínimo de *C.*
    - Si esa arista conecta dos árboles diferentes se añade al bosque, combinando los dos árboles en un solo árbol.
    - En caso contrario, se desecha la arista.

  Al acabar el algoritmo, el bosque tiene un solo componente, el cual forma un árbol de expansión mínimo del grafo.

  En un **árbol de expansión mínimo** se cumple:

  - La cantidad de aristas del árbol es la cantidad de nodos menos uno (1).

- **Algoritmo de dijkstra:**

  Primero marcamos todos los vértices como no utilizados. El algoritmo parte de un vértice origen que será ingresado, a partir de ese vértice evaluaremos sus adyacentes, como dijkstra usa una técnica greedy – La técnica greedy utiliza el principio de que para que un camino sea óptimo, todos los caminos que contiene también deben ser óptimos- entre todos los vértices adyacentes, buscamos el que esté más cerca de nuestro punto origen, lo tomamos como punto intermedio y vemos si podemos llegar más rápido a través de este vértice a los demás. Después escogemos al siguiente más cercano (con las distancias ya actualizadas) y repetimos el proceso. Esto lo hacemos hasta que el vértice no utilizado más cercano sea nuestro destino.

Al proceso de actualizar las distancias tomando como punto intermedio al nuevo vértice se le conoce como relajación (relaxation).

Dijkstra es muy similar a BFS, si recordamos BFS usaba una Cola para el recorrido para el caso de Dijkstra usaremos una Cola de Prioridad o Heap, este Heap debe tener la propiedad de Min-Heap es decir cada vez que extraiga un elemento del Heap me debe devolver el de menor valor, en nuestro caso dicho valor será el peso acumulado en los nodos.

Tanto java como C++ cuentan con una cola de prioridad ambos implementan un Binary Heap, aunque con un Fibonacci Heap la complejidad de dijkstra se reduce haciéndolo más eficiente.

- **Algoritmo de Floyd-Warshall:**

  El algoritmo de Floyd-Warshall compara todos los posibles caminos entre cada par de nodos. Esto se consigue al ir mejorando un estimado de la distancia entre dos nodos, hasta que el estimado es óptimo.
  Considerar un grafo G con nodos o vértices V, cada una numerada de 1 a N, además de una función que al ingresar i, j, k devuelve el camino más corto entre i y j pasando solo por el conjunto {1, 2, …, k}. Ahora, utilizando esta función, el objetivo es encontrar el camino más corto entre i para cada j usando solo los vértices 1 hasta k+1.
  Existen dos candidatos para cada uno de esos caminos: o el verdadero camino más corto pasando por los nodos {1, …, k}; o existe un camino que vaya desde i hasta k+1, después de k+1 hasta j que es mejor. Sabemos que el mejor camino de i a j que solo usa los nodos desde 1 hasta k está definido por la función anterior, y está claro que si existiera un camino desde i hasta k+1 hasta j, entonces el valor de este camino seria la concatenación de el

camino más corto de i hasta k+1 (usando vértices {1, …, k}) y el camino más corto desde k+1 hasta j (también usando vértices {1, …, k}).

Si v(i,j) es el valor o costo de la arista entre los nodos i y j, podemos definir la función ahora llamada camino Corto(i,j,k) en los términos de la siguiente formula recursiva: el caso base seria

camino Corto(i,j,0) = v(i,j)

y el caso recursivo seria

camino Corto(i,j,k) = min(camino Corto(i,j,k-1), camino Corto(i,k,k-1)+ camino Corto(k,j,k-1)

Esta fórmula es el corazón del algoritmo de Floyd-Warshall. El algoritmo funciona primero calculando la función para todos los pares (i,j) para k=1, después k=2, etc. Este proceso continua hasta k=n; y hemos encontrado el camino más corto para todos los pares (i, j) usando cualquier nodo intermedio.

Fuentes:

https://www.cs.us.es/cursos/cc-2009/material/bfs.pdf

https://jariasf.wordpress.com/2012/03/02/algoritmo-de-busqueda-depth-first-search-parte-1/

https://es.wikipedia.org/wiki/Algoritmo_de_Prim

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

https://jariasf.wordpress.com/2012/03/19/camino-mas-corto-algoritmo-de-dijkstra/

http://algoritmofloywarshall.blogspot.com/

Paso 4. Transición de las ideas a los diseños preliminares:

Revisando los algoritmos que nos ayudan a solucionar el problema los pudimos dividir así para cada función:

Para recorrer dentro del grafo:

- Algoritmo de recorrido en anchura (BFS)
- Algoritmo de recorrido en profundidad (DFS)

Para encontrar el árbol de recubrimiento mínimo:

Es decir, encontrar el grafo que pase por todos los vértices, pero con el menor peso entre sus aristas.

- Algoritmo de prim.
- Algoritmo de kruskal.

Para encontrar la ruta más corta:

- Algoritmo de dijkstra.
- Algoritmo de Floyd-Warshall.

Paso 5. Evaluación y selección de la mejor solución:

Revisando cada detalle del problema decidimos que no íbamos a utilizar los recorridos dentro del grafo, solo necesitaremos el árbol de recubrimiento mínimo y las rutas más cortas.

Así que estas son las notas más importantes para tener en cuenta en la evaluación de estos algoritmos:

Para el árbol de recubrimiento mínimo:

Las funciones de los dos algoritmos utilizados para el árbol de recubrimiento mínimo son muy parecidas, la única diferencia es que PRIM necesita un nodo inicial, en cambio Kruskal inicia listando todas las aristas en orden y empieza de mayor a menor según el orden que se maneje en el problema. Así que el algoritmo con el cual trabajaremos es con el de KRUSKAL.

Para la ruta más corta:

Las funciones de los dos algoritmos utilizados para encontrar la ruta más corta se diferencian en que DIJKSTRA funciona para encontrar la ruta más corta desde un nodo hasta los otros; y FLOYD-WARSHALL funciona para encontrar la ruta más corta desde todos los nodos hasta todos los demás nodos. Así que el algoritmo que se puede utilizar mejor es el de FLOYD-WARSHALL.

A continuación, mostraremos los dos algoritmos elegidos en pseudocódigo:

Kruskal:

```
 función Kruskal(G)
   Para cada v en V[G] hacer
     Nuevo conjunto C(v) ← {v}.
   Nuevo heap Q que contiene todas las aristas de G,
ordenando por su peso
   Defino un arbol T ← Ø
   // n es el número total de vértices
   Mientras T tenga menos de n-1 aristas y !Q.vacío()
hacer
     (u,v) ← Q.sacarMin()
     // previene ciclos en T. agrega (u,v) si u y v están
diferentes componentes en el conjunto.
     // Nótese que C(u) devuelve la componente a la que
pertenece u
     Si C(v) ≠ C(u) hacer
       Agregar arista (v,u) a  T
       Merge C(v) y C(u) en el conjunto
   Responder arbol T
```

Floyd-Warshall:

```
1 /* Suponemos que la función pesoArista devuelve el
coste del camino que va de i a j
2    (infinito si no existe).
3 También suponemos que n es el número de vértices y
pesoArista(i,i) = 0
4 */
5
6 int camino[][];
7 /* Una matriz bidimensional. En cada paso del
algoritmo, camino[i][j] es el camino mínimo
8 de i hasta j usando valores intermedios de (1..k-1).
Cada camino[i][j] es inicializado a
9
10 */
11
12 procedimiento FloydWarshall ()
13    para k: = 0 hasta n - 1
14
15          camino[i][j] = mín ( camino[i][j], camino[i]
[k]+camino[k][j])
16
17    fin para
```

Método de la Ingeniería

Aplicación para la solución de un problema

Proyecto final AED

TRABAJO PRESENTADO POR:

WBEYMERTH GALLEGO

DENNYS MOSQUERA

FANNY VARELA