BUSQUEDA DE RUTAS EFICIENTES DE REPARTO

PRESENTADO POR:

MIGUEL ANGEL SALAMANCA

JUAN CAMILO BAZURTO ARIAS

PRESENTADO A:

SEBASTIAN CAMILO MARTINEZ REYES

ESCUELA COLOMBIANA DE INGENIERÍA JULIO GARAVITO

ALGORITMOS Y ESTRUCTURAS DE DATOS

PROGRAMA DE INGENIERÍA DE SISTEMAS

BOGOTÁ D.C.

2021 – 1

1. Contexto

Suponga que usted es una persona que tiene que entregar un pedido de un punto 2, usted estando en el punto 1 debería buscar la ruta más eficiente con el fin de optimizar su trabajo. Sin embargo, sería una tarea muy tediosa ver el mapa de su ubicación y buscar la ruta más eficiente o preguntar a gente conocida si la conoce. Con la teoría de grafos y la tecnología que tenemos hoy en día, podemos resolver este problema.

2. Requisitos

2.1. Especificación

2.1.1. Entrada

Se genera una ruta con sus respectivos nodos y conexiones, y se agregaran pedidos para esta ruta, pueden ser uno o más. A estos nodos y conexiones se le agregaran los pesos entre los nodos que representaran las distancias entre los lugares donde hay que hacer entrega, esto con el fin de implementar arboles de expansión mínimo.

Ahora si el usuario lo desea tenemos la opción de recoger y llevar un paquete, esto con el fin de implementar caminos mínimos.

2.1.2. Salida

Se mostrará la ruta más eficiente por BFS, si hay más de un pedido se mostraran las etiquetas de cómo debe ser el orden de los pedidos por Topological-Sort, el cual utiliza un algoritmo muy similar a DFS.

Se mostrará la ruta más eficiente teniendo en cuenta los pesos (distancia) entre los nodos, esto gracias al algoritmo de Kruskal.

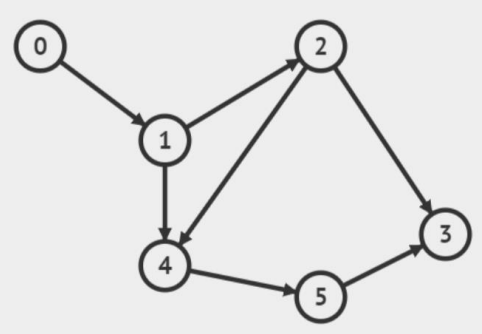
Si se selecciona la opción de recoger y enviar paquete, gracias a caminos mínimos, se le mostrara al usuario la ruta de peso mínimo para recoger el paquete.

3. Diseño

3.1. Estrategia

3.1.1. Descripción general

3.1.1.1. Grafos.

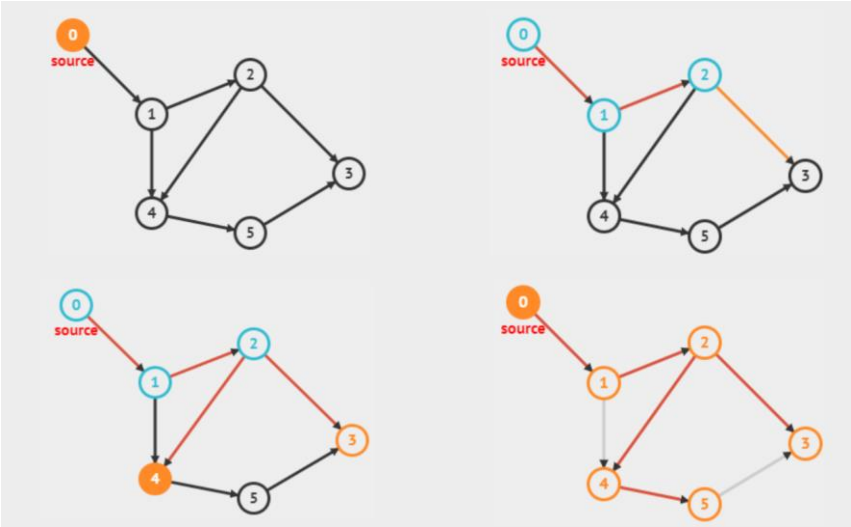


Como podemos ver en el grafo anterior, utilizaremos la teoría de grafos para representar los puntos y rutas que puede tomar un repartidor, por ejemplo, el repartidor está en el punto cero y necesita ir al punto cinco, nuestra labor es mostrarle la ruta más eficiente para llegar el punto cero al punto cinco, cuando los posibles puntos uno, dos, tres y cuatro.

3.1.1.2. BFS.

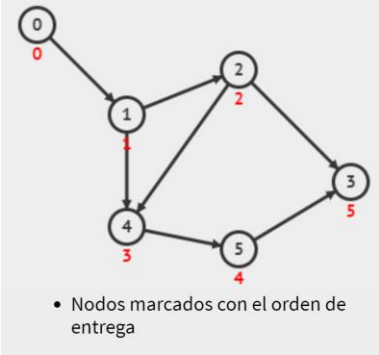
Si deseamos encontrar la ruta más corta desde el punto cero al punto 3 tendremos que saber cuáles son las distancias entre los nodos y a partir de ahí, tomar la de menor valor. Todo esto nos lo permite el algoritmo Breadth-First Search (BFS), que nos retorna todas las rutas posibles para llegar a cualquier nodo distinto al que nos encontramos.

Sabiendo esto, el usuario solo deberá brindar su ubicación y el destino al que desea llegar, el algoritmo BFS calculará todas las rutas posibles y se elegirá la más corta, mostrándosela al usuario.}



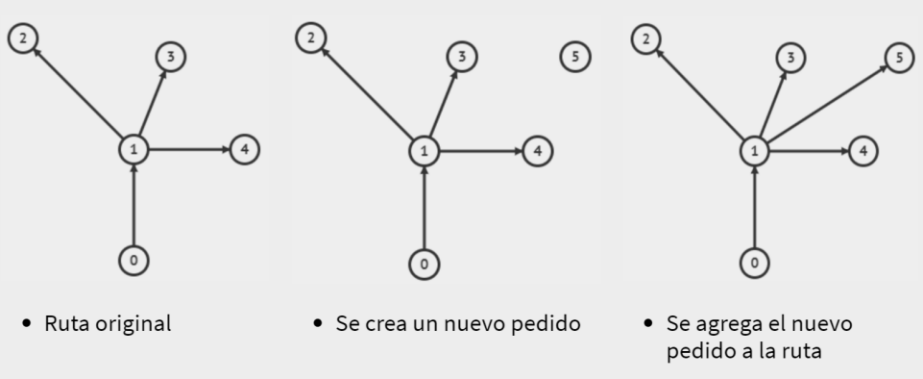
3.1.1.3. DFS.

Para ordenar las entregas de los pedidos que tiene un repartidor, hay que llevar un orden de entrega que haga la tarea más eficiente, con el algoritmo DFS y el ordenamiento Topological Sort, podemos brindarle al usuario etiquetar cada lugar que va a representar el orden en que los pedidos van a ser entregados.



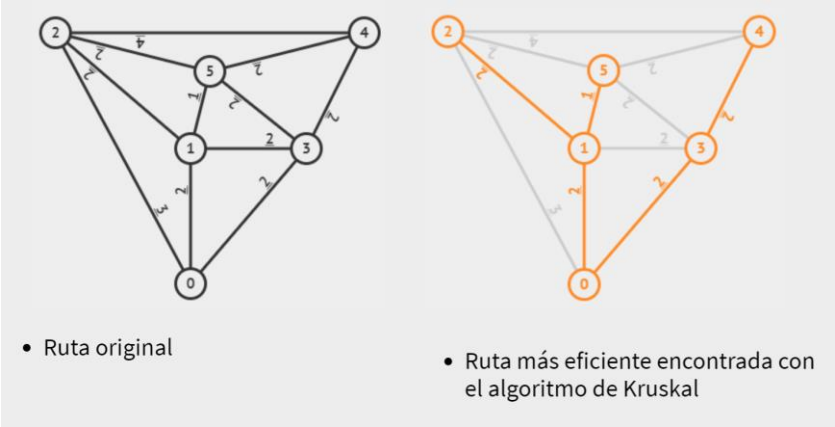
3.1.1.4. Conjuntos Disyuntos.

En el caso que a un repartidor tenga varios pedidos en tiempos de entrega distintos, la ruta va a tener que ser modificada para poder entregar estos en un tiempo optimo, con la ayuda de conjuntos disyuntos y su algoritmo unión.find vamos a poder agregar nuevos pedidos a una ruta ya creada.



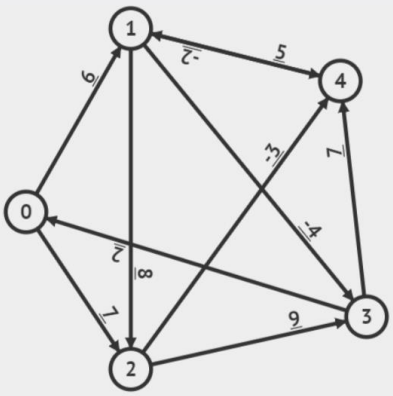
3.1.1.5. Algoritmo de Kruskal.

Para poder implementar este algoritmo tenemos que introducir un nuevo concepto, los pesos, que en este caso van a representar distancias entre lugares, el algoritmo de Kruskal nos permite encontrar el árbol de expansión mínimo (Minimum spanning tree), que nos va a servir para encontrar la ruta más eficiente en el caso de que haya varios pedidos, lo cual será en bastantes si no en todos los casos.

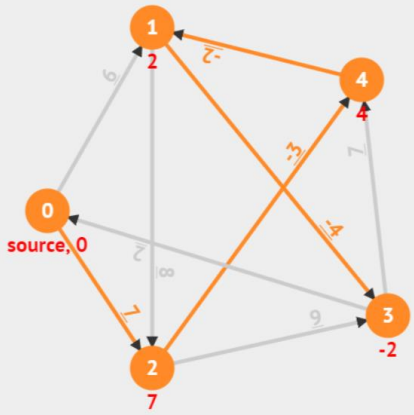


3.1.1.6. Algoritmo Bellman Ford y Dijkstra.

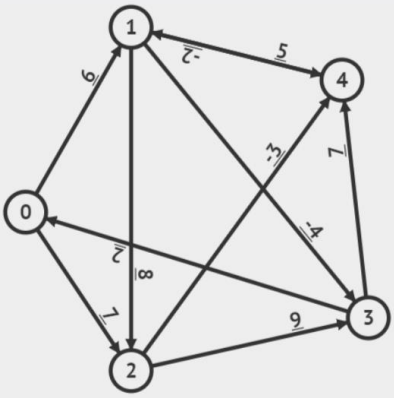
Para poder implementar Caminos Mínimos utilizaremos el algoritmo de Bellman-Ford o el algoritmo Dijkstra en los que también hablamos de los pesos en las rutas, estos algoritmos nos permiten encontrar la ruta de peso mínimo, dada esta ruta la aplicaremos para que el repartidor recoja un paquete.



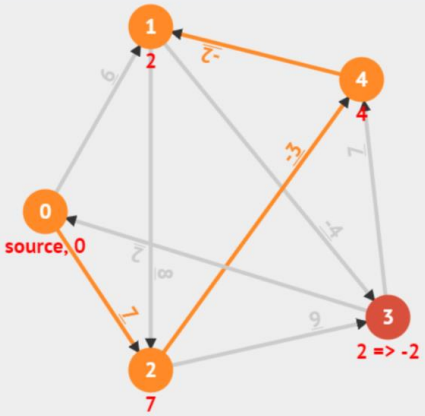
- Ruta original



- Ruta peso minimo Bellman Ford



- Ruta original



- Ruta peso minimo Dijkstra

Relación entre algoritmo Dijkstra y algoritmo de Prim:

La relación entre estos dos algoritmos se evidencia en el código, en el uso de montones y colas, también los dos algoritmos usan la misma función extract-min, la cual extrae el valor mínimo.

3.2. Casos de Prueba

Como casos de prueba se tomaron los siguientes casos:

A. BFS, DFS, conjuntos disyuntos.

- Nodos:

  V = [0, 1, 2, 3, 4, 5]

- Conexiones:

  E = [(0, 1), (0, 2), (1, 2), (2, 3), (4, 3), (4, 5)]

B. Arboles de expansión mínimo.

- Nodos:

  V = [0, 1, 2, 3, 4, 5]

- Conexiones:

  E = [(0, 1, 2), (0, 2, 3), (0, 3, 2), (1, 0, 2), (1, 2, 2), (1, 3, 2), (1, 5, 1), (2, 0, 3), (2, 1, 2), (2, 4, 4), (2, 5, 2), (3, 0, 2), (3, 1, 2), (3, 4, 2), (3, 5, 2), (4, 2, 4), (4, 3, 2), (4, 5, 2), (5, 1, 1), (5, 2, 2), (5, 3, 2), (5, 4, 2)]

C. Caminos mínimos.

- Nodos:

V = [0, 1, 2, 3, 4]

- Conexiones:

E = [(0, 1, 1), (0, 2, 10), (1, 3, 2), (2, 3, 12), (3, 4, 3)]

Los resultados de estos casos fueron los siguientes:

A. BFS, DFS, conjuntos disyuntos.

- Ruta más eficiente:

  0

  0

  1

  0->1

  2

  0->2

  3

  0->2->3

  4

  4

  5

  5

- Orden de pedidos:

  4, 5, 0, 1 ,2 ,3

B. Arboles de expansión mínimo.

- Ruta más corta:

  {(1, 5, 1), (3, 4, 2), (1, 2, 2), (0, 1, 2), (0, 3, 2)}

C. Caminos mínimos.

- Rutas de peso mínimo:

  0

0

1

0 -> 1

Distancia: 1

2

0 -> 2

Distancia: 10

3

0 -> 1 -> 3

Distancia: 3

4

0 -> 1 -> 3

Distancia: 6

4. Análisis

4.1. Temporal

4.1.1. BFS

La complejidad de este algoritmo es O(V + E)

4.1.2. Topological-Sort

La complejidad de este algoritmo es O(V + E)

4.1.3. Nuevo Pedido

La complejidad de este algoritmo es O(V)

4.1.4. Algoritmo de Kruskal.

La complejidad de este algoritmo es de O(ElogV)

4.1.5. Algoritmo Bellman Ford.

La complejidad de este algoritmo es de O(VE)

4.1.6. Algoritmo Dijkstra

La complejidad de este algoritmo es de O(V^2)