Fomentando el Car-Pooling

Juan Camilo Jiménez Rojas
Universidad
Colombia
jcjimenezr@eafit.edu.co

Santiago Espinosa Valderrama
Universidad
Colombia
sespinosav@eafit.edu.co

Mauricio Toro
Universidad Eafit
Colombia
mtorobe@eafit.edu.co

RESUMEN

Tras largas investigaciones en las bases de datos de la universidad sobre los diferentes algoritmos para poder encontrar la ruta más corta entre dos puntos hemos se ha llegado a la conclusión que el algoritmo A* es el más eficiente para este proyecto ya que diferentes empresas como Google o Waze lo implementan en sus aplicaciones.

Los problemas más relevantes relacionados a este tienen que ver generalmente con encontrar la manera mas efectiva de recorrer un grafo para un fin determinado, ya sea encontrar un circuito Hamiltoniano, Euleriano, entre otros, En un tiempo relativamente corto.

La solución que planteamos fue basada en el vecino más próximo haciendo algunas modificaciones, los resultados son bastante buenos dado que se encontró una solución optima en un tiempo relativamente bueno.

1. INTRODUCCIÓN

El servicio de Car-Pooling es un movimiento social que ayuda tanto al medio ambiente como a los estudiantes ya que tiene la ventaja que los usuarios se dirigen al mismo destino eso es un gran beneficio ya que muchas veces movilizarse en ciertas zonas el servicio público no suele ser eficiente, pero también puede llegar a ser perjudicial para el conductor ya que se puede tomar más tiempo de lo habitual y se desea promover ese movimiento.

2. PROBLEMA

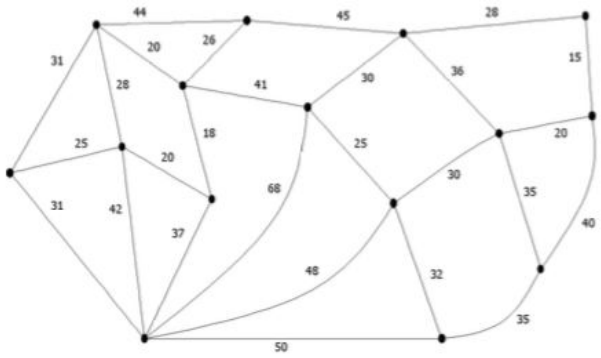
El problema que podemos encontrar es poder hacer un Car-Pooling eficiente ya que el suele usar la mejor ruta para llegar en el menor tiempo posible a su destino y en el momento de comenzar a compartir el vehículo tendrá que tomar rutas alternativas y decidir qué cantidad de personas recoger pero sin superar un 20% del tiempo del habitual del dueño del conductor y la cantidad de puestos que vehículo, con la finalidad de llegar a la universidad sin mayor retraso.

Se plantea ver el mapa de medellín como un grafo, cada esquina se volvería un vértice y cada calle un arco con una distancia y un tiempo que será evaluada al recorrer este grafo buscando el costo mínimo.

3. TRABAJOS RELACIONADOS

3.1A Muddy City

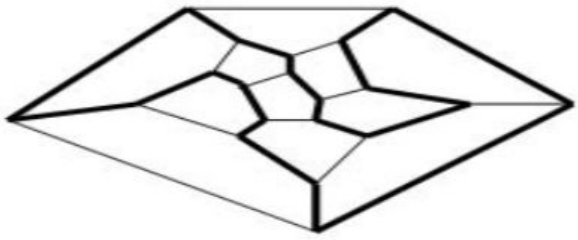
Se tiene un grafo en el cual no se tiene calles y se deben pavimentar suficientes calles para que sea posible viajar de cualquier casa a otra casa, se tiene la posibilidad de viajar a través de casas. La pavimentación debe hacerse a un costo mínimo. Esto se va a hacer con losas como pavimento.



3.2 Recorrido en ciudad

A un dodecaedro, cuerpo sólido regular con doce caras pentagonales, se la ha quitado una cara y se lo ha äplastado en el plano como muestra la figura

Imaginemos a los vértices de esta figura como ciudades y a las aristas como tramos de caminos entre dos ciudades. Se pregunta si hay un camino formado de tramos que partiendo de una ciudad visite todas las ciudades una sola vez volviendo a la ciudad de partida (**ciclo hamiltoniano**)



3.3 Coloreado de mapas

La figura 4 muestra un mapa con 4 districtos A, B, C y D. Se trata de pintar cada districto con un color de forma que dos regiones con un borde comun (que no sea un punto) tengan distintos colores y queremos hacer esto usando un minimo numero de colores. La figura 5 muestra un grafo homeomorfo al mapa, en el sentido que los vertices del grafo se corresponden con las regiones del mapa y dos vertices estan conectados por una rama cuando las regiones correspondientes tienen un borde comun. El problema se traduce en el grafo a minimizar el numero de colores al asignar un color a cada vertice de forma que cualquier rama tenga extremos de distinto color.

figura 4

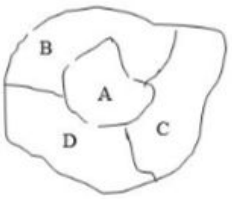
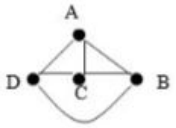


figura 5



3.4 El problema del caballo en el juego de ajedrez

Consideremos un tablero de ajedrez. y un caballo. Se pregunta si es posible que el caballo parta de un casillero y visite todos los otros 63 casilleros una solo vez volviendo al punto inicial. (ciclo hamiltoniano)

4. Algoritmo A*(Modificado)

A* está catalogado con unos de los mejores algoritmos de búsquedas en grafos o árboles binarios, debido a la forma eficiente de cómo maneja sus costes.

Su función de evaluación usar valores de los nodos de la red con un valor heurístico y los arcos con el valor de para dirigirse a los nodos conexos.

El algoritmo calculará la distancia total la cual será la mínima, después desde su posición inicial comenzará a recoger 4 personas más, en cada nodo que recorre recogiendo una persona se calculará la menor distancia siendo así, eligiendo a cual personas recoger para tener el recorrido más corto.

4.1 Estructura de datos

Gráfica 1: Grafo con diferentes distancias, cada nodo representa un lugar.

4.2 Operaciones de la estructura de datos

Heuristica

- Nodo 0: 30
- Nodo 1: 20
- Nodo 2: 20
- Nodo 3: 50
- Nodo 4: 0

Gráfica 2: Árbol del grafo con las fórmulas realizadas para llegar a los diferentes nodos.

En el gráfico anterior podemos ver que cada no tiene un valor heurístico asignado y la sima entre la distancia de el nodo anterior más el valor heurístico podemos calcular la ruta más corta entre el Node 0 y el Node 4 que es el destino.

4.3 Criterios de diseño de la estructura de datos

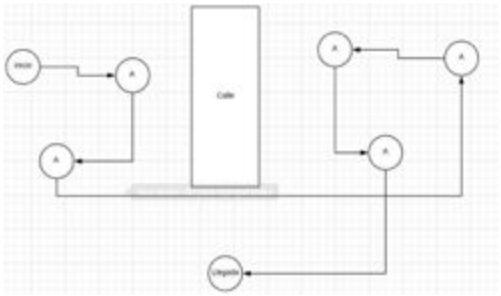
- Es una combinación entre búsquedas del tipo primero en anchura con primero en profundidad: mientras que h(n) tiende a primero en profundidad, g(n) tiende a primero en anchura. De este modo, se cambia de camino de búsqueda cada vez que existen nodos más prometedores.

- Su velocidad puede llegar ser mucho mayor que su algoritmo padre que es el Dijkstra y esa es la prioridad en nuestro proyecto
- La facilidad de implementación y modificación..
- Tiene algunas buenas referencias acerca elementos de ruda de programación en la industria como Google y Waze implementa este algoritmo para encontrar las rutas más cortas entre dos puntos.

4.4 Análisis de complejidad

Método	Complejidad
Búsqueda Fonética	O(1)
Imprimir búsqueda fonética	O(m)
Insertar palabra busqueda fonética	O(1)
Búsqueda autocompletado	O(s + t)
Insertar palabra en TrieHash	O(s)
Añadir búsqueda	O(s)

4.5 Gráfico de solución.



4.6 Cálculo de la complejidad del algoritmo

Sub problema	Complejidad
Crear el grafo	O(V)
Obtener sucesores	O(E)
Encontrar nodos	O(E)
Búsqueda	O(V log V)
Complejidad Total	O(V log V)

4.7 Criterios de diseño del algoritmo.

El problema requiere del análisis más efectivo para elegir el camino más corto para recoger a los usuarios.

Tras investigar las diferentes opciones de búsqueda en grafos y reconocer las necesidades del problema, se llega a una conclusión:

Identificando que la conglomeración de los clientes es frecuente en diferentes situaciones, lo que facilita el hecho de que el algoritmo A* (Modificado) sea una solución viable, debido a que en grafos donde los nodos tienen una cantidad considerable de vértices adyacentes, este algoritmo trabaja bastante bien, para el problema que estamos solucionando, solo es adaptarlo para nuestro problema

4.8 Tiempos de Ejecución

	Conjunto de Datos 1	***Conjunto de Datos 2***	***...Conjunto de Datos n***
Mejor caso	10 sg	20 sg	5 sg
Caso promedio	12 sg	10 sg	35 sg
Peor caso	15 sg	21 sg	35 sg

Tiempos de ejecución con diferentes conjuntos de datos:

n	$\log_2 n$
1	0
2	1
4	2
8	3
16	4
32	5
64	6
128	7
256	8
512	9
1024	10
1,048,576	20
2,097,152	21

4.9 Memoria

	Conjunto de Datos 1	*Conjunto de Datos 2*	*...Conjunto de Datos n*
Consumo de memoria	10 MB	20 MB	5 MB

4.10 Análisis de los resultados

Tabla de valores durante la ejecución

Estructuras de autocompletado	LinkedList	Arrays	HashMap
Espacio en el Heap	60MB	175MB	384MB
Tiempo creación	1.16 - 1.34 s	0.82 - 1.1 s	2.23 - 2.6 s
Tiempo búsqueda ("a")	0.31 - 0.39 s	0.37 - 0.7 s	0.22 - 0.28 s
Tiempo búsqueda ("zyzzyvas")	0.088 ms	0.038 ms	0.06 ms
Búsqueda ("aerobacteriologically")	0.077 ms	0.041 ms	0.058 ms
Tiempo búsqueda todas las palabras	6.1 - 8.02 s	4.07 - 5.19 s	4.79 - 5.8 s

Consumo de memoria

5. VoralSAnisé

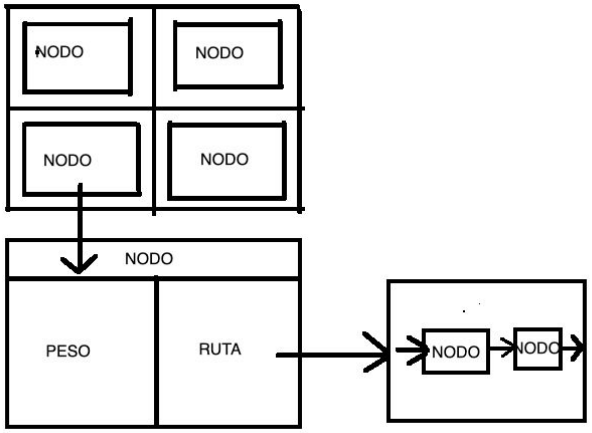
5.1 Estructura de datos

La estructura de datos que se ha implementado es voraz junto con matrices de adyacencia para poder implementar los grafos conexos, adyacencia listas de tuplas, donde las tuplas contienen el peso de la ruta y una lista con la ruta.

Organizamos los carros con respecto a la tardanza desde su partida hasta la Universidad EAFIT de mayor a menor, luego según el orden y que recojan a las personas según la adyacencias más cercana sin pasarse del límite dado e ir a la Universidad EAFIT cuando esté a punto de pasar del límite o cuando el vehículo esté lleno.

Gráfica: Imagen de ruta al vecino más cercano

5.2 Operaciones de la estructura de datos



Gráfica: Imagen de una operación de agregación de datos

Acá podemos ver una muestra de una matriz con cada nodo que es una tupla con el peso y la ruta

5.3 Criterios e diseño de la estructura de datos

Diseñamos de esta manera porque es una forma muy gráfica de representar la solución, ya que puede interpretarse como un tipo de jerarquía con las diferentes estructuras de datos que estamos utilizando tales como las tuplas y las listas.

De esta forma nos permite insertar nodos de una manera organizada y visualmente fácil de entender ya que si en un futuro deseamos mejorar nuestro código no será difícil recordar nuestra estrategia la cual consiste en dependiendo del peso que cada nodo tenga con respecto a los otros nodos el algoritmo comienza desde el nodo más lejano hasta el nodo de destino.

La eficiencia de la ruta de los nodos será óptima, porque nos aseguramos de que no quede ningún nodo sin visitar, teniendo en cuenta la restricción de la P.

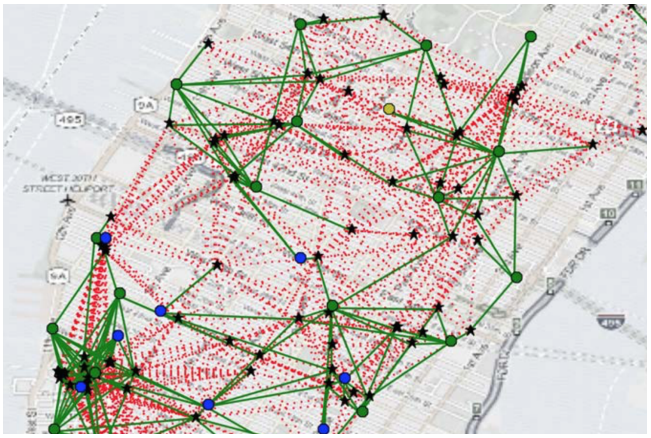
5.4 Análisis de Complejidad

MÉTODO	COMPLEJIDAD EN EL PEOR CASOS
AGREGAR NODO	O(n)
ORDENAMIENTO	O(n^2)
VERIFICAR CONDUCTOR	O(n)
ASIGNAR AL AUTO	O(n)

Tabla: Tabla para reportar la complejidad

5.5 Algoritmo

El algoritmo se basará en ordenar una lista (que contendrá los nodos directos a la universidad) de menor a mayor (con el algoritmo de ordenamiento lineal que es de complejidad n ^ 2) a partir de los pesos de cada nodo, vamos a tomar el que tiene más peso como conductor, ya que es el más alejado de la universidad en relación con otros nodos. Por lo tanto, viajará desde el lugar más lejano a la universidad, llenando la lista con los nodos adyacentes hasta obtener 5 pasajeros, la lista se tomará como un automóvil y se realizará el mismo procedimiento para los siguientes nodos del último elegido.



Gráfica: Conección entre nodos, rutas verdes son las rutas a tomar y las rojas son rutas posibles que no se tomaron.

5.6 Cálculo de la complejidad del algoritmo

Sub problema	Complejidad
Crear el grafo	O(N)
Ordenar lista	$O(N^2)$
Obtener pesos	O(1)
Complejidad Total	$O(N^2)$

5.7 Criterios de diseño del algoritmo

Decidimos hacerlo de esta manera debido a la efectividad del algoritmo en relación con el grafo conexo y la forma en que se manejan los datos (las personas involucradas en el uso compartido del automóvil).

El algoritmo de ordenamiento se utiliza para la optimización de la forma de distribuir los automóviles, ya debemos saber cuáles son los más cercanos a algunos nodos y estos no agregan tiempo a su viaje a la universidad (se refiere a la p) que pueden realizarlos, aparte se eliminarán de la lista principal de nodos directos para indicar que ya se han asignado automáticamente.

5.8 Tiempos de Ejecución

	Conjunto de Datos 1	*Conjunto de Datos 2*	*...Conjunto de Datos n*
Mejor caso	2 ms	10 ms	15 ms
Caso promedio	3 ms	14 ms	40 ms
Peor caso	30 ms	64 ms	154 ms

5.9 Memoria

	Conjunto de Datos 1	*Conjunto de Datos 2*	*...Conjunto de Datos n*
Consumo de memoria	13 MB	24 MB	35 MB

5.10 Análisis de los resultados

Autocompletado en la estructura	**Lista**
Memoria	35 MB
Tiempo de creación	50 ms
Tiempo de ejecución	2ms

6. CONCLUSIONES

- Tener todo ordenado de una manera específica es más fácil tener una respuesta clara, para ellos los algoritmos de ordenamiento son muy útiles cuando se manejan problemas de este tipo.
- Trabajar con un gráfico completo es mucho más fácil de usar estructuras de datos como matrices, en este caso una lista enlazada de listas vinculadas.
- Diseñar estructuras de datos fáciles de entender se hacen muy útiles para no perder el foco del problema.

6.1 Trabajos futuros

En el futuro nos gustaría implementar este algoritmo para poder mejorar el aire de la ciudad para así aportar a nuestra madre naturaleza desde la ingeniería.

AGRADECIMIENTOS

Le agradecemos al StackOverFlow por siempre guiarnos junto con los monitores que fueron un gran apoyo para poder diseñar una efectiva estructura de datos.

REFERENCIAS

- Prim's algorithm,2019. Consulted on march 2 of 2019, Wikipedia, The free encyclopedia. Recovered of:https://en.wikipedia.org/wiki/Prim%27s_alg orithm
- Kruskal algorithm,2019. Consulted on march 2 of 2019, Wikipedia, The free encyclopedia. Recovered of:https://es.wikipedia.org/wiki/Algoritmo_de_ Kruskal
- Toy problems for real world. Consulted on march 2 of 2019. Tenderfoot: Unit 2. Recovered of: https://www.computingatschool.org.uk/data/tft/02/04Activity_Toy_Problems_Real_World.pdf
- Krustal algorithm, 2012. Consulted on march 12 of 2019. Graphs: Software for construction, edition and graph analysis. Recovered of:http://arodrigu.webs.upv.es/grafos/doku.php ?id= algoritmo_nkruskal
- Introducción a A*. (2019). Retrieved from: http://theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html
- Algoritmo de búsqueda A*. (2019). Retrieved from: https://es.wikipedia.org/wiki/Algoritmo_de_búsqueda_A*
- 3.A* Search Algorithm GeeksforGeeks. (2019). Retrieved fromhttps://www.geeksforgeeks.org/a-search-algorithm