Carpooling, la mejor manera de optimizar rutas y purificar el aire

Laura Alzate Madrid
Universidad Eafit
Colombia
dalzatec1@eafit.edu.co

David Alzate Cardona
Universidad Eafit
Colombia
lalzatem@eafit.edu.co

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

RESUMEN
Medellín es una ciudad afectada por el alto tráfico, por ello la importancia del estudio sobre la reducción del tráfico. En este documento se abarcará el problema de optimización de los algoritmos para el uso del carro compartido o también conocido como carpooling. Todo esto con el fin de mejorar a la vez el estilo de vida en Medellín y el medio ambiente.

Palabras clave
Listas, backtracking, grafos tráfico, carpooling, algoritmo, costo mínimo.

1. INTRODUCCIÓN

Uno de los grandes problemas actuales de las ciudades es el alto tráfico, y Medellín es una ciudad gravemente afectada en diversos aspectos tales como la disminución de la calidad de vida y la contaminación.

En el siguiente documento se abarcará este problema y además una solución que en un futuro se espera que tenga un efecto muy positivo. En pocas palabras, se mostrara la estrategia del uso del carro compartido, también conocido como "carpooling", con el fin de reducir considerablemente el tráfico.

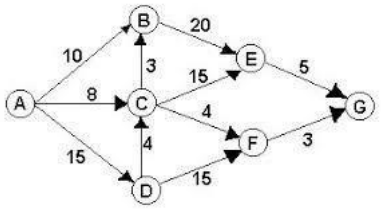
2. PROBLEMA

Se busca una manera efectiva de reducir el tráfico por medio de compartir los vehículos de personas que vayan a un mismo sitio, esto con el fin de reducir la cantidad de CO2 que se produce y la cantidad de vehículos que transitan al tiempo en las ciudades.

3. TRABAJOS RELACIONADOS

3.1 Gestión de envío

Empresas como servientrega o rappi diariamente se enfrentan al problema de recoger y entregar mercancía a diferentes zonas o localidades y estos lo deben de hacer en el menor tiempo posible, además de esto también cuentan con una carga limitada lo que hace que sea un problema más complejo.

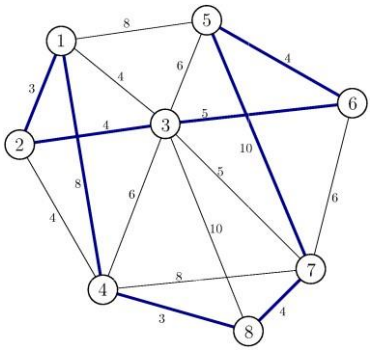


Grafica de un grafo dirigido para llegar de un punto a otro.

3.2 Problema del viajante (Travelling Salesman Problem (TSP))

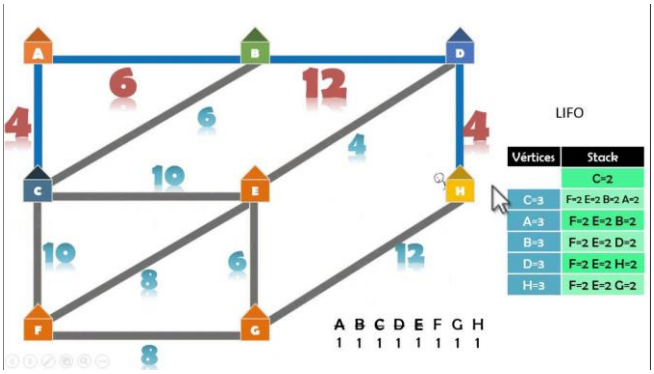
En este problema se intenta encontrar todos los recorridos posibles para visitar varias ciudades, sin repetirlas y volviendo a la ciudad origen, teniendo esto, lo siguiente es buscar la ruta más corta.

Una formulación equivalente en términos de Teoría de grafos es: dado una grafo ponderado completo (donde los vértices representan las ciudades, las aristas representan los caminos y los pesos son el costo o las distancias de estos caminos), encontrar un ciclo de Hamilton con menor peso.



3.3 Algoritmo del vecino más próximo

El algoritmo del vecino más próximo fue, en las ciencias de la computación, uno de los primeros algoritmos utilizados para determinar una solución para el problema del viajante. Este método genera rápidamente un camino corto, pero generalmente no el ideal.



Gráfica del vecino más cercano

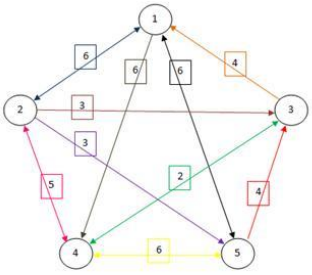
3.4 Algoritmo de búsqueda A*

El algoritmo A* es un algoritmo de búsqueda que puede ser empleado para el cálculo de caminos mínimos en una red. Se va a tratar de un algoritmo heurístico, ya que una de sus principales características es que hará uso de una función de evaluación heurística, mediante la cual etiquetará los diferentes nodos de la red y que servirá para determinar la probabilidad de dichos nodos de pertenecer al camino óptimo.

7	6	5	6	7	8	9	10	11		19	20	21	22
6	5	4	5	6	7	8	9	10		18	19	20	21
5	4	3	4	5	6	7	8	9		17	18	19	20
4	3	2	3	4	5	6	7	8		16	17	18	19
3	2	1	2	3	4	5	6	7		15	16	17	18
2	1	0	1	2	3	4	5	6		14	15	16	17
3	2	1	2	3	4	5	6	7		13	14	15	16
4	3	2	3	4	5	6	7	8		12	13	14	15
5	4	3	4	5	6	7	8	9	10	11	12	13	14
6	5	4	5	6	7	8	9	10	11	12	13	14	15

3.5 El algoritmo de Floyd

El algoritmo de Floyd es muy similar, pero trabaja con grafos ponderados. Es decir, el valor de la "flecha" que representamos en la matriz puede ser cualquier entero o infinito. Infinito marca que no existe unión entre los nodos. Esta vez, el resultado será una matriz donde estarán representadas las distancias mínimas entre nodos, seleccionando los caminos más convenientes según su ponderación ("peso"). Por ejemplo, si de "A" a "B" hay 36 (km), pero de "A" a "C" hay 2(km) y de "C" a "B" hay 10 (km), el algoritmo nos devolverá finalmente que de "A" a "B" hay 12 (km).



Mejoras en los recorridos:

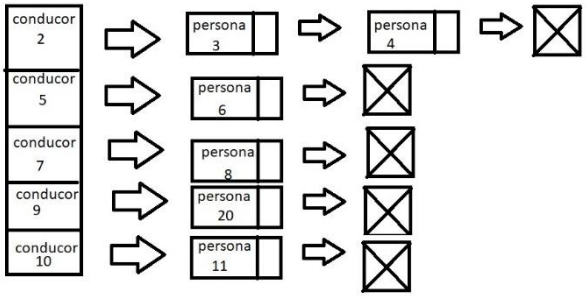
	Antes	Ahora
De 1 a 3	∞	8 por 4
De 3 a 2	∞	7 por 4
De 3 a 5	∞	8 por 4
De 4 a 1	∞	6 por 3
De 5 a 2	∞	11 por 4

De 1 a 3 se reduce a 8 km pasando por el nodo 4
De 3 a 2 se reduce a 7 km pasando por el nodo 4
De 3 a 5 se reduce a 8 km pasando por el nodo 4
De 4 a 1 se reduce a 6 km pasando por el nodo 3
De 5 a 2 se reduce a 11 km pasando por el nodo 4

4. TITULO DE UNA SOLUCIÓN DISEÑADA

Las estructuras de datos que se decidió fue LinkedList, y Backtracking. Dichas listas almacenan toda la información sobre las permutaciones posibles en nuestro algoritmo. También se tiene un método que me retorna el camino más corto de un nodo a otro, evitando que se pase del tope de 5 personas (las que caben en el carro), y tratando de no sobrepasar por mucho, el tiempo que cada persona necesitaba para ir a trabajar solo en su carro. Esto lo hace leyendo un archivo y a la vez guarda la solución o soluciones es otro aparte.

4.1 Estructura de datos



Grafica 1: Lista dentro de una lista encadenada de personas (conductores). Cada conductor contiene su número y a quienes va a llevar en su auto.

4.2 Operaciones de la estructura de datos

4.3 Criterios de diseño de la estructura de datos

Decidimos usar estas estructuras de datos, ya que nos pareció que podíamos hacer un buen uso óptimo de las listas (LinkedList) y Backtracking lo usamos porque en un principio es fácil de implementar pero no muy rápido, sin embargo se adapta muy bien a la estructura del problema.

4.4 Análisis de Complejidad

Método	Complejidad
Insertar lista	O(1)

4.5 Algoritmo

4.6 Calculo de la complejidad del algoritmo

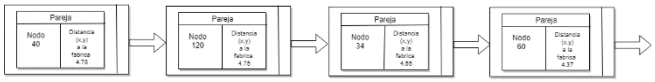
Sub problema	Complejidad
Leer archivo	O(n*n)
Escribir archivo	O(n)
Asignar un vehículo	O(n)
Camino más corto	O(n*n)
Complejidad Total	O(n*n)

Tercera Entrega

5.1 Diseño de la estructura de datos:

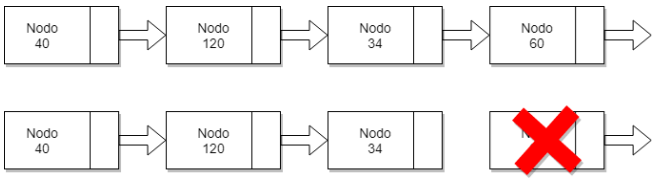
Utilizamos arreglos dinámicos (ArrayList) de parejas para ordenar el nodo más lejano al más cercano de la empresa y utilizamos un objeto llamado grado grafo para almacenar las conexiones que hay entre cada punto

Imagen de la lista de parejas ordenada desde el nodo más lejano de la fábrica según sus coordenadas.



5.2 Diseño de las operaciones de la estructura de datos:

Imagen de una operación de borrado de una lista encadenada



5.3 Criterios de diseño de la estructura de datos

Utilizamos el ArrayList de pares ya que para poder obtener un elemento del ArrayList es Orden O(1) y el obtener cualquiera de los dos datos de la pareja también es O(1).

5.4 Análisis de Complejidad

Método	Complejidad
Crear grafo	O(V+E)
Crear ArrayList de pares	O(N)
Ordenar ArrayList de pares	O(NlogN)
Recorrer arrayList	O(N)
Buscar el vecino más optimo	O(E)
recoger al vecino	O(1)
Eliminar vecino que se recoge	O(k)
Agregar al conjunto total de autos	O(1)

5.5 Algoritmo

Para solucionar la ruta más óptima lo que hicimos recorrer desde el nodo más lejano a la empresa con el fin de que si está lejos es más probable que pueda recoger a más gente; para esto implementamos un arreglo dinámico de pares, el cual se compone del nodo y de la distancia que hay de ese nodo a la empresa (para calcular esta distancia utilizamos el teorema de Pitágoras), luego de hallar la distancia del nodo este se agrega a la lista, luego de tener a todos los nodos con su distancia los ordenamos del más lejano de la fábrica al más cercano. Luego de esto creamos el objeto grafo con sus nodos y sus arcos (para esto usamos matrices).

Luego empezamos a recorrer este arreglo 1 por 1, lo que hace esto es coger el primer valor de la pareja del arreglo y busca los nodos a los que está conectado, para saber si recoge o no a una persona lo hace de la siguiente manera; este busca el nodo más cercano pero además también pregunta si la distancia del nodo al que va a ir sea menor que la distancia del nodo actual, es decir si el nodo al que vamos a visitar esta más lejos de la empresa, implica que del nodo actual se tendría que ir en una dirección contraria a la empresa, lo cual no es óptimo.

Luego de tener el carro lleno verificamos tiempo, si recogiendo a personas se demora más que yendo solo (multiplicado por la constante de tiempo que nos dan), entonces va a ir eliminando a uno por uno hasta que el tiempo sea aceptable

4.6 Calculo de la complejidad del algoritmo

Sub problema	Complejidad
Crear el Arreglo de pares	O(N)
Ordenar el Arreglo de Pares	O(N*log N)
Crear el grafo de los datos	O(V+E)
Buscar el nodo más cercano	O(K*E)
Insertar elemento en el “auto”	O(1)

Buscar Y eliminar elemento del “auto”	O(1*K)
Añadir combinación al conjunto total	O(1)
Complejidad Total Simplificada Complejidad calculada para dato con 205 nodos	O(V+E+N(E+K+logN)) N=número de aristas E=número de vértices K=número de personas en el auto

5.7 Criterios de diseño del algoritmo

Este algoritmo lo diseñamos porque nos permite tener una solución bastante rápida y bastante asertiva ya que al ordenar los nodos de forma que el más lejano es el que recoge a la gente es más probable que este algoritmo voraz escoja el camino más acertado. Se usó algoritmo voraz con el fin de bajar los tiempos, pues antes teníamos un algoritmo hecho con backtracking por la precisión de la respuesta, pero se demoraba demasiado. Con voraz logramos rebajar el tiempo y la memoria que consume este algoritmo.

5.8 Tiempos de Ejecución

	Conjunto de 205	***Conjunto de 11***	***...Conjunto de 5***
Mejor caso	25 ms	0 ms	0 ms
Caso promedio	33.8 ms	0.5 ms	0.2 ms
Peor caso	40 ms	2 ms	1 ms

5.9 Memoria

	Conjunto de 205	***Conjunto de 11***	***...Conjunto de 5***
Consumo de memoria	7.170824 MB	3.977744 MB	4.340664 MB

5.10 Análisis de los resultados

Número de puntos	**Valor de P(multiplicador de tiempo)**	**Numero de autos**
5	1.2	2
11	1.3	5
11	1.5	4
205	1.1	81
205	1.2	65
205	1.3	55
205	1.5	46
205	1.7	45

6. CONCLUSIONES

Lo mas importante a fin de cuentas era bajar el numero de autos gracias a algo conocido como “carpooling”, al poder pasar de tener 204 autos para llegar a una empresa poder reducirlo a 81(en caso de P=1.1) es bastante significativo ya que reducimos el trafico casi un 61%

Al tener un P de 1.3 y U=204 y poder hacer que se baje hasta 55 autos hace que el medio ambiente tenga un respire muy grande ademas de reducer el trafico ya que se reduce un 75% el flujo de vehiculos en Medellin

Aplicar algoritmos voraces en vez de backtracking hace que el tiempo baje demasiado, ya que solo con el archive de 11 nodos se nos demoraba 4.5 segundos mientras que ahora el de 205 con algoritmos voraces se demora en promedio 33.8 milisegundos, al inicio tuvimos problemas ya que no encontrábamos una forma óptima para cambiar el backtracking.

6.1 Trabajos futuros

Para un futuro nos gustaría crear e implementar nuestro propio algoritmo de búsqueda de costo mínimo de manera de que sea rápido y muy optimo, que este algoritmo sea capaz de mostrar los resultados exactos. En un futuro pensamos en ir más allá de lo que logramos hacer para este proyecto, queremos implementar las estructuras de datos más adecuadas para este tipo de ejercicios, y no quedarnos tanto en reducir el tiempo, sino también pensar en reducir la memoria, intentar hacer ambos a la vez sin necesidad de empeorar el algoritmo. Buscaremos hacer un algoritmo lo más entendible para que cualquier programador que esté interesado en este proyecto pueda tener una muy buena guía para comenzar a trabajar con muy buenas bases. Nos gustaría

ensayar más estructuras de datos, e investigar sobre algoritmos heurísticos aptos para este problema de costo mínimo.

REFERENCIAS

2. Fischer, G. and Nakakoji, K. Amplifying designers' creativity with domainoriented design environments. in Dartnall, T. ed. Artificial Intelligence and Creativity: An Interdisciplinary Approach, Kluwer Academic Publishers, Dordrecht, 1994, 343-364.

Miquel Àngel Estrada, El algoritmo que abre camino a los mensajeros
https://www.agenciasinc.es/Noticias/Elalgoritmo-que-abre-camino-a-los-mensajeros

Algoritmo de búsqueda A*
https://es.wikipedia.org/wiki/Algoritmo_de_b%C3%BAsqueda_A*