CARPOOLING NOS SOLUCIONA EL TRÁFICO

Daniel Felipe Gómez Martínez
Universidad Eafit
Colombia
dfgomezm@eafit.edu.co

Daniel García García
Universidad Eafit
Colombia
dgarciag@eafit.edu.co

Cesar Andrés García Posada
Universidad Eafit
Colombia
cagarciap@eafit.edu.co

RESUMEN

Para nadie secreto que el tráfico aumenta cada día en nuestra ciudad, lo que genera diversos problemas, como congestión en las vías, deterioro de la calidad del aire, ineficiencia en la movilidad, perdida del tiempo, entre otros problemas. Una estrategia que nace con el fin de ayudar a solucionar esta problemática es conocida como Vehículos Compartidos. En el presente escrito se buscara dar informe sobre el algoritmo realizado para llevar a cabo dicha estrategia. Bien sabemos que una cantidad de puntos en la ciudad se puede ver claramente como un grafo, entonces el problema es cómo trabajar con dicho grafo. Para comenzar será indispensable tener de una manera ordenada cada vértice del grafo con sus respectivos vértices adyacentes, el paso siguiente será determinar cómo transitar entre dichos vértices; este flujo de vértice a vértice se realizara con una serie de condiciones y con una estrategia fundamental; la cual nos dice que visitemos primero el vértice más lejano al punto de destino y de ahí empecemos a ir a los vértices más cercanos. Así nos aseguramos que las personas que se encuentran cerca al punto de destino pueden ser recogidas por otras personas y a su vez intentamos reducir el costo del viaje a las personas que se encuentran más retiradas.

PALABRAS CLAVES

Grafo completo → Grafo dirigido → Camino más corto → Estructuras de datos → Complejidad → Eficiencia → Ordenamiento → Recorrido de grafos → Tráfico → Reducción de costos.

1. INTRODUCCIÓN

En la ciudad de Medellín, por cada 1000 habitantes hay aproximadamente 345 vehículos y más del 60% de los vehículos transitan en la ciudad todos los días. Los niveles de contaminación aumentan significativamente en el Valle de Aburrá y en el país, por lo que se llevan a cabo campañas como "El Pico y Placa ambiental" entre otras. Además, los numerosos embotellamientos debido a la cantidad de vehículos generan retrasos y pérdida de tiempo. Es esencial disminuir en la mayor proporción posible la cantidad de vehículos que transitan en las calles, y para esto, una estrategia de solución puede ser "Vehículos Compartidos" o "Carpooling". Estrategia que viene tomando fuerza en la Universidad Eafit, donde personas que poseen vehículos particulares, puedan recoger a otras personas durante su trayecto de vieja hacia la universidad, sin que esto afecte en gran medida su tiempo estimado de viaje.

2. PROBLEMA

Para ayudar al medio ambiente y la calidad de vida de los habitantes de la ciudad de Medellín, el problema es realizar un algoritmo de manera eficiente para reducir el tráfico de la ciudad, esto a través de la estrategia de los Vehículos Compartidos. El algoritmo debe determinar la cantidad mínima de vehículos para que todas las personas puedan llegar a su destino, se asumen diferentes aspectos:

1) Inicialmente cada persona llega a la universidad en un vehículo, es decir, cada persona usa un vehículo.
2) Todas las personas van al mismo lugar de llegada (Universidad Eafit).
3) En cada vehículo pueden ir máximo 5 personas.
4) El conductor del coche debe tomar la ruta más corta que sea eficiente a su trayecto de viaje.
5) Cada persona que se moviliza en un coche, tiene un máximo de tiempo que se puede demorar, este tiempo se calcula mediante la siguiente formula: ***[A]***

$$C = I + (I * p)$$

- Donde C corresponde al costo total de tiempo que una persona se puede gastar.
- Donde I corresponde al costo inicial que una persona tiene desde su lugar de salida, hasta la universidad.
- Donde p corresponde a la contante de tiempo estipulada por el usuario.
-

Es decir, cada persona se puede demorar el tiempo inicial de viaje desde su lugar de salida hasta la universidad más un tiempo extra determinado por la formula anteriormente expuesta.

3. TRABAJOS RELACIONADOS

3.1 Problema de viaje canadiense

El problema del viaje canadiense, fue definido por *Papadimitriou y Yannakak* en 1991 [2]. Este problema es encontrar la ruta más corta entre un punto de origen y un

punto a destino. Estos puntos están en un grafo G = (V, E) (G es un grafo, V son nodos del gráfico y E son bordes) pero en este problema, el viajero no conoce la información completa sobre el recorrido (que se simula en un grafo), es decir, los caminos pueden sufrir cambios. [3] En particular, algunos de los bordes del gráfico pueden quedar bloqueados e intransitables. El viajero tiene dos opciones para resolver el problema:

1) Avanza en su viaje y encuentre un camino bloqueado y determine un nuevo camino. El costo de la nueva ruta se agrega al último cálculo de costo.

2) Pagar un recargo sobre el costo y determine previamente los bordes bloqueados, pero los bordes bloqueados pueden convertirse en un borde pasable.

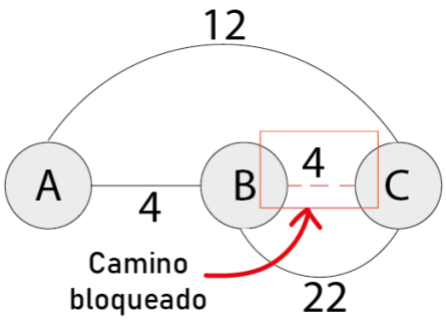

Imagen I. *Ejemplo Problema del viaje canadiense.*

En la figura anterior, el viajero quiere ir del nodo A al nodo C, cuando llega al nodo B, se da cuenta de que el borde entre B y C está bloqueado. El viajero debe calcular la nueva ruta más corta, que sería regresar al nodo A e ir directamente al nodo C, pero el costo de esta nueva ruta se agregará al costo previamente calculado. La otra opción que tiene el viajero es pagar un costo adicional en el nodo A, donde se determinan las rutas bloqueadas.

3.2 Breaking the Breakdown

Si quisiéramos definir el nombre del problema en español, podríamos darle diferentes nombres: "Acabando con el tráfico", "Acabando el embotellamiento" o algo más coloquial "Dejando el trancón".

El Breakdown se conoce como el momento en el que en una determinada área salen menos vehículos de los que entran, en consecuencia, se aumenta significativamente el tráfico en la vía de la ciudad. Un algoritmo creado en la *Universidad Tecnológica de Nanyang de Singapur* ha logrado reducir la congestión de tráfico, es desarrollando un algoritmo que evita las colisiones mediante la redirección de vehículos que hacen uso de rutas alternativas, haciendo así cambios en la red vial. Esto es posible gracias a lo que conocemos como Inteligencia Artificial (IA).

El algoritmo consiste en asignarle a un vehículo una red de tráfico, esta red de tráfico es determinada por medio de millones de cálculo, con el fin de evitar que muchos vehículos colisionen o entren en una misma red de tráfico, debido a que es imposible determinar una red vial para cada vehículo, lo que se busca es el mínimo número de vehículos que entran a una red vial. [4]

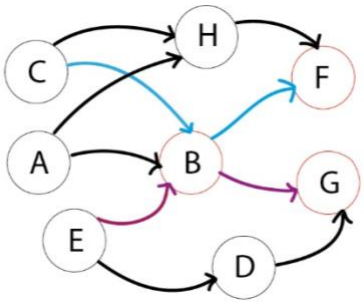

Imagen II. *Simulación del problema de Breakdown, en donde al nodo B llegan más vehículos en comparación a los que salen.*

Nota: En la imagen anterior podemos ver que el trayecto que se lleva desde el nodo E hasta el nodo G está representado por el color purpura, el trayecto desde el nodo C al nodo F se encuentra de color azul.

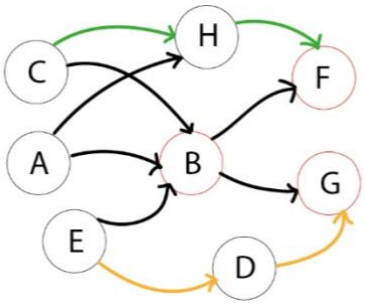

Imagen III. *Posible solución haciendo un redireccionamiento de ciertos vehículos*

Nota: En la imagen se puede ver que el camino Azul, y purpura (*Ver Imagen II*). Puede ser reemplazado por el camino de color verte y amarillo respectivamente (*Ver Imagen III*).

3.3 Reducción de Taxis

Investigadores italianos de *CNR (Consiglio Nazionale Delle Ricerche)*, junto con expertos de *MIT (Massachusetts Institute of Technology)* y la *Universidad de Cornell* en *New York - Estados Unidos* desarrollaron un algoritmo que puede reducir la cantidad de vehículos, esto es debido al tráfico en

la ciudad de Nueva York causado en gran parte por los taxis. El programa garantiza los mismos niveles de reducción del servicio de taxi, además de ofrecer turnos de trabajo más cortos para los taxistas. El método se basa en el modelo "Red de intercambio de vehículos" y lo que hace es encontrar la secuencia de viajes que puede asistir a un solo vehículo, analizando los tiempos y las coordenadas del punto de recogida y descenso del pasajero. Se organizaron taxis para que cada uno hiciera la ruta más corta sin un pasajero, reduciendo así la cantidad de taxis que ofrecen el servicio en una ciudad. [5]

3.4 Pathfinding

Pathfinding es una base importante para los sistemas de navegación, ya que se utiliza en áreas de robótica y, especialmente, en videojuegos. El objetivo principal es encontrar la mejor manera posible de llegar desde un punto inicial hasta un punto final en representaciones del entorno llamadas mapas de malla. En el caso de los videojuegos (especialmente en los juegos de estrategia), también se debe tener en cuenta que la ruta debe tener el Mínimo de posibles obstáculos que retrasen su llegada a su destino. Para desarrollar el algoritmo se realiza una modificación al algoritmo de Dijkstra para encontrar la ruta más corta de un punto a otro, tomado en cuenta que la modificación al algoritmo de Dijkstra está orientado a la geometría computacional se implementa mediante la estructura del videojuego donde se utilizan técnicas de geometría clásica, topología, teoría de grafos, teoría de conjuntos y álgebra lineal. [6]

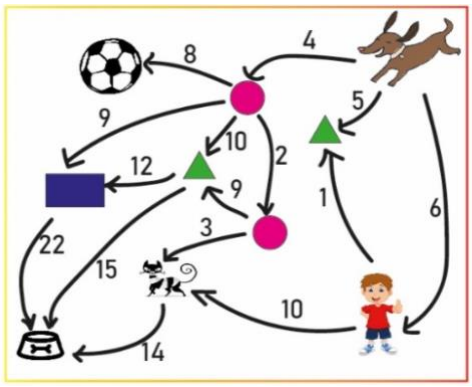

Imagen IV. *Camino más corto de un punto a otro usando la modificación al algoritmo de Dijkstra.*

3.5 Carpooling

Rectángulo de rutas posibles

Este algoritmo de Carpooling desarrollado para una tesis universitaria, se basa en el cálculo de rutas en google maps desde un punto de origen hacia un punto destino, luego se hará una filtración donde se determina si un punto P está dentro de un polígono convexo. Dado un segmento [A, B] y un punto P, el producto vectorial es utilizado para saber si el punto está a la derecha o a la izquierda del segmento. Luego se calculan los puntos basados en latitud y longitud que conforman el rectángulo que contiene a los puntos de origen y destino de las rutas con auto, y se comprueba si el origen y el destino están dentro de este rectángulo o no. En caso de que esté, el algoritmo devolverá esta ruta posible. [7]

Branch and Bround

Este algoritmo se basa en la creación de un árbol binario de búsqueda, a partir de esto se tiene que si la menor rama para algún árbol nodo (conjunto de candidatos) A, es mayor que la rama del nodo padre hacia otro nodo B, entonces A se descarta de la búsqueda, conocido como poda. Se implementa manteniendo una variable global m que guarda el mínimo nodo padre visto en las subregiones ya evaluadas. Cualquier nodo cuyo nodo hijo es mayor que m se descarta. La recursión se aplica cuando un conjunto candidato S queda con un sólo elemento, o también cuando el nodo padre para este conjunto S coincide con un nodo hijo. De cualquier forma, cualquier elemento de S va a ser el mínimo de una función en S.

Si el problema en el nodo actual es mayor que el mejor punto entero calculado, el algoritmo borra ese nodo del árbol, y por consiguiente las ramas que lleva debajo. Si el algoritmo encuentra una solución con un valor menor que el mejor punto hasta el momento, se renueva el mejor valor en ese punto y pasa a analizar el siguiente nodo. Si el problema es conveniente pero no entero, y el valor es menor que el mejor punto hasta el momento, sucede la ramificación. [8]

4. SOLUCIÓN DEL PROYECTO

4.1 Estructura de datos

Para la realización de la primera solución, se usaron las siguientes estructuras de datos:

- ***Grafo:*** grafo de los puntos en la ciudad de Medellín.

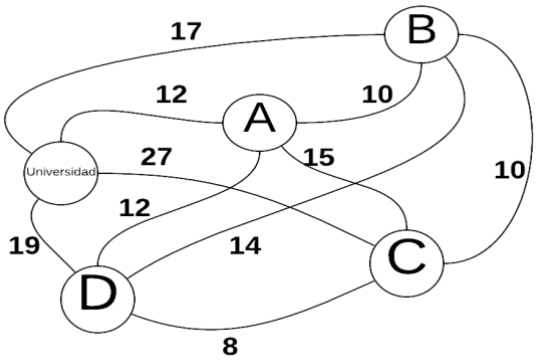

Imagen V*. Ejemplo Grafo completo no dirigido. Simulación de un grafo de la ciudad con 5 vértices*

- ***Matriz:*** Para el control y maneo del grafo, utilizamos la implementación del grafo por medio de la matriz, donde se almacena el valor del peso correspondiente a cada nodo.

	Universidad	***A***	***B***	***C***	***D***
Universidad	0	7	17	27	19
A	7	0	10	15	12
B	17	10	0	10	14
C	27	15	10	0	8
D	19	12	14	8	0

Imagen VI. *Grafo de Imagen V en matriz para tener acceso a los costos de una manera más rápida y eficiente*

- ***LinkedList:*** Estructura necesaria para tener de una manera organizada los nodos adyacentes a un nodo especifico

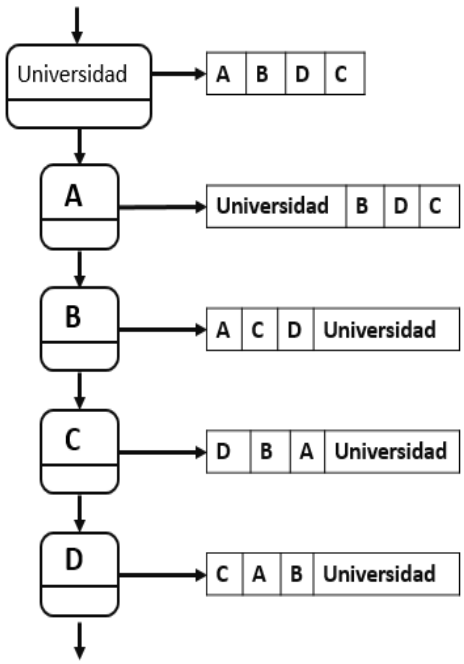

Imagen VII. *LikedList de arreglos, e la cual se ordenan los vértices adyacentes a cada vértice de menor a mayor costo.*

4.2 Operaciones de la estructura de datos

Para la solución del problema, requerimos operaciones en las estructuras de datos seleccionadas, por ende, a la hora de trabajar con los nodos del grafo, tenemos operaciones como:

- **Consulta LinkedList:** para recuperar un determinado dato, en una LinkedList tenemos la operación *.get(x)*

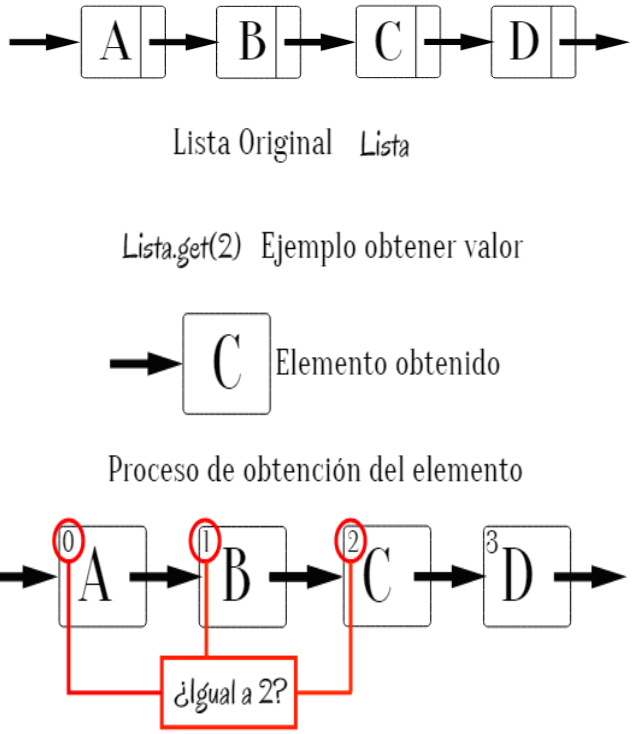

Imagen VIII. *Operación .get() LinkedList*

Con esto podemos encontrar que la complejidad en el peor de los casos para obtener un determinado valor es de O(n) debido a que para buscar un elemento se realiza un recorrido de toda la lista comparando posición con el índice de búsqueda

- **Agregado LinkedList:** Para agregar un valor a una linkedList tenemos el método *.add(value)*

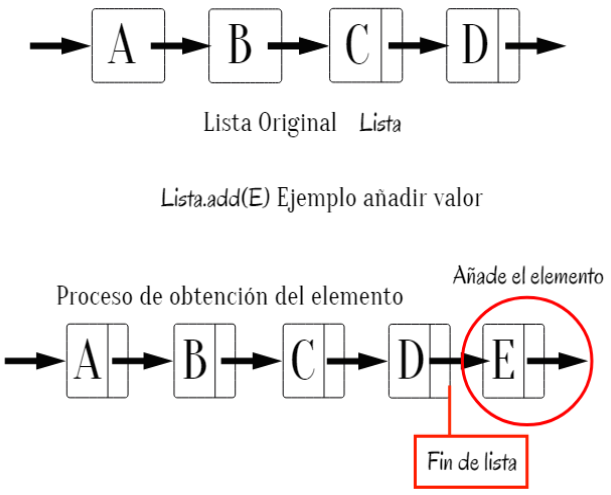

Imagen IX. *Operación .add(x) LinkedList*

Con esto podemos comprobar que el método de adicionar un elemento dentro de una LinkedList es de complejidad en el peor de los casos de O(1). Debido a que para agregar un elemento se agrega al final de la lista.

- **Eliminar elemento LinkedList:** Para eliminar un elemento de una LinkedList tenemos la operación *.remove(index)*

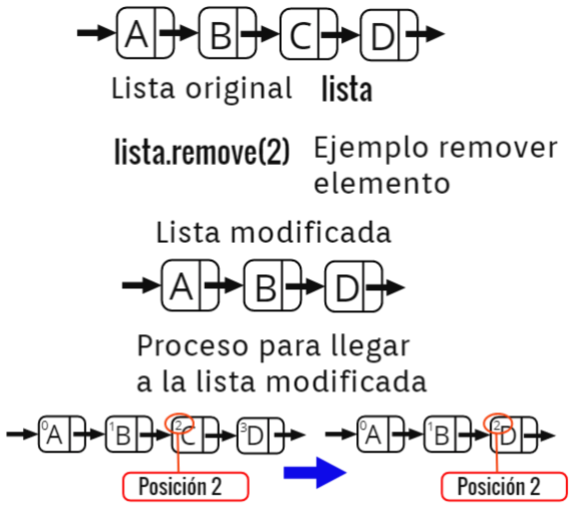

Imagen X. *Eliminar un elemento de una LinkedList por medio de la operación .remove(index)*

En el ejemplo anterior pudimos eliminar un elemento en una determinada posición, cabe resaltar que si no se ingresa la posición que se desea eliminar, la LinkedList eliminara por defecto la primera posición. Por este hecho la complejidad en el peor de los casos es de O(1)

- **Búsqueda ArrayList:** para recuperar un determinado dato en un ArrayList tenemos la operación *.get(x)*

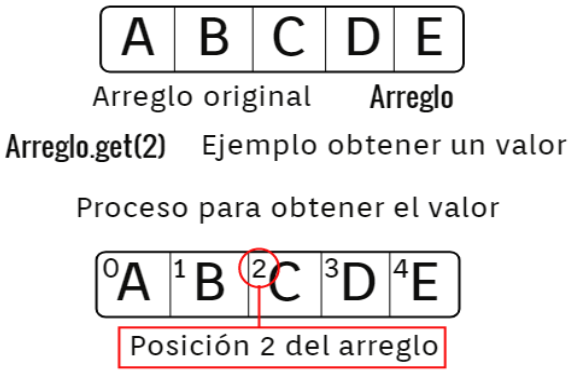

Imagen XI. *Búsqueda de un elemento en ArrayList por medio de la función .get(index)*

Debido al proceso "mágico" que se realiza en el ArrayList podemos concluir que la complejidad para buscar un elemento en el peor de los casos es de O(1).

- **Agregar elemento ArrayList:** Para poder agregar un elemento en un ArrayList tenemos la operación *.add(value)*

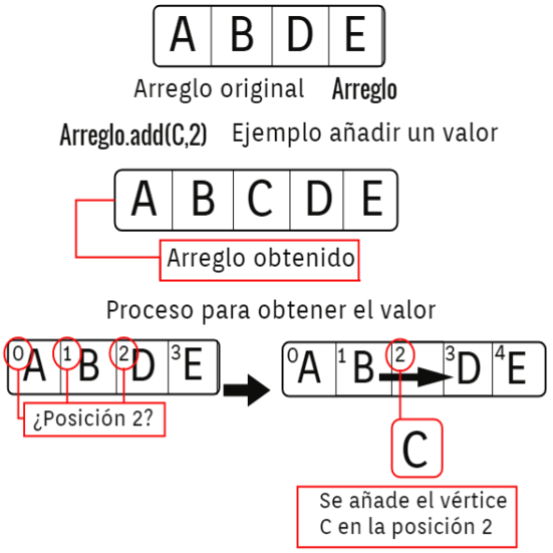

Imagen XII. *Añadir un elemento a un ArrayList, con la operación .add(value, index)*

En el ejemplo anterior podemos agregar un elemento en cualquier posición dela arreglo. El ArrayList interiormente realiza una operación en la que empuja (en caso de que existan) los valores que hay después del index deseado. Por este motivo esta operación tiene en el peor de los casos una complejidad de O(n) debido a que en el peor de los casos

debe de recorrer todo el arreglo. Cabe destacar que no siempre es necesario colocar un index (posición en donde se agrega el elemento) si usamos la operación de la manera *.add(value)* este elemento se agregara en la última posición del ArrayList

4.3 Criterios de diseño de la estructura de datos

La estructura de datos utilizada es una LinkedList de listas. (Inicialmente se trabaja con una LinkedList con arreglos), con el fin de ordenar de menor a mayor los nodos adyacentes a cada nodo. Esto con el objetivo de realizar descartes a medida que se va calculando un camino especifico. Además, tomando en cuenta que la complejidad de los métodos de la LinkedList son bastantes eficientes.

4.4 Análisis de Complejidad [9]

A continuación, se muestra una tabla con la complejidad de algunos métodos de la LinkedList

MÉTODO	COMPLEJIDAD
get()	O(n)
add()	O(1)
remove()	O(1)
contains()	O(n)
size()	O(n)

Tabla 1. *Complejidad métodos LinkedList*

A su vez mostramos una tabla de complejidad de algunos métodos del ArrayList con el fin de realizar comparaciones entre ArrayList y LinkedList:

MÉTODO	COMPLEJIDAD
get()	O(1)
add()	O(n)
remove()	O(n)
contains()	O(n)
size()	O(n)

Tabla 2. *Complejidad métodos ArrayList*

4.5 Algoritmo

Tomando en cuenta que se quiere determinar la cantidad mínima de vehículos que se requieran para que todas las personas puedan llegar a la universidad, usando la estrategia de vehículos compartidos, tenemos que es necesario disminuir el costo de traslado cada vértice más lejano de la universidad, o bien sea aprovechar el recorrido de este recogiendo otras personas.

Lo dicho anteriormente se realizará por medio de la siguiente proceso o algoritmo:

1) Se crea un arreglo con el costo adicional que desde cada vértice se puede gastar, esto por medio de la formula explicada en el numeral 2, este arreglo es necesario para realizar comparaciones más adelante a la hora de formar y/o completar un vehículo.
2) Debido a que es un grafo completo, se realiza un ordenamiento de todos los vértices adyacentes a cada vértice.
    a. Debido a que todos deben de llegar al mismo punto, a medida que completemos un vehículo, bien sea con los cinco pasajeros permitidos o no, se tomara el vértice más alejado al punto de llegada, en este caso la Universidad.
    b. Cabe destacar que todo vértice visitado, no se puede visitar dos veces, es decir, una persona no puede ir en dos vehículos a la vez. Para esto se tiene un arreglo de tipo ***[B]*** Boolean – El cual, para explicación del algoritmo se llamara Vértices visitados – con el fin de determinar en cada iteración que vértice ya fue visitado para evitar visitarlo nuevamente.
3) Después de determinado el vértice más alejado del lugar de llegada, se realizan los siguientes procesos:
    a. Se escoge el vértice de menor costo, este vértice es un vértice adyacente al vértice más alejado de la universidad. La ventaja de tener los vértices adyacentes ordenados permite que si en una evaluación un vértice no pasa la prueba los siguientes no pasaran la prueba.
    b. Se calcula el costo hasta el momento.
    c. Se determina si este costo es menor o igual al costo analizado en el puto 1. Es decir, se determina si es viable la realización del recorrido calculado.

Al ser viable el camino, en el arreglo de vértices visitados, se agrega el vértice analizado encontrado, con el fin de no volver pasar sobre este vértice. De otro modo seguimos con el siguiente vértice menor encontrado.

Este proceso se realiza hasta obtener un automóvil lleno o hasta visitar todos los vértices adyacentes. Posteriormente volvemos al numeral 2.a.

Este proceso se realizara hasta que todos los vértices estén visitados, es decir, que el arreglo Vértices visitados ***(Ver [B])*** cada posición este en estado True.

A continuación se realizará una simulación del algoritmo.

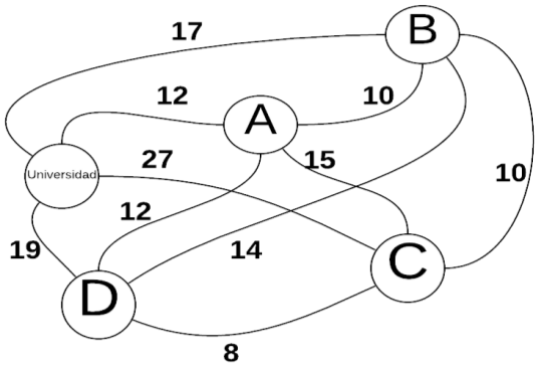

Imagen XIII. *Grafo de simulación (5 vértices contando la universidad)*

PROCESO

Para este ejemplo tomaremos como constante máxima de tiempo P = 1.3, y la cantidad de personas que pueden ir en un vehículo es máxima de 3.

1) Se ordenan de menor a mayor los vértices adyacentes a cada vértice, poniendo en primer lugar la universidad, debido a que este es el vértice al que se quiere llegar.

Universidad	0, Uni	12, A	17, B	19, D	27, C
A	0, A	10, B	12, Uni	12, D	15, C
B	0, B	10, A	10, C	14, D	17, Uni
C	0, C	8, D	10, B	15, A	27, Uni
D	0, D	8, C	15, A	14, B	19, Uni

Imagen XIV. *Ordenamiento de los vértices adyacentes a cada vértice (vértice, costo)*

2) Se crea una LinkedList de arreglos en donde almacenaremos cada carro formado

3) Elegimos el vértice más lejano desde la universidad, es decir, el que tiene mayor costo.

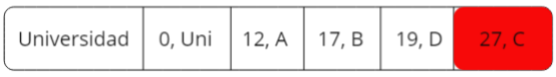

Imagen XV. *En rojo la elección del vértice no visitado más lejano de la universidad (Primera Iteración)*

A) En este caso el vértice más lejano es el vértice C queda seleccionado y al arreglo de vértices visitados, el vértice C se pone en True. (Para simular true se colocara de color rojo y para simular false se colocara verde)

Imagen XVI. *Arreglo de vértices visitados, marcando el vértice C como ya visitado*

B) Se crea un arreglo en el que iremos agregando los vértices que completan el vehículo que parte del vértice C. A esta altura del algoritmo solo es encuentra agregado el vértice C

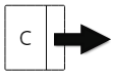

Imagen XVII. *Arreglo de vértices que conforman un vehículo, el cual parte del vértice C*

Ahora iremos hasta el vértice de menor costo del vértice seleccionado, descartando el trayecto del vértice seleccionado al mismo.

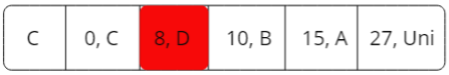

Imagen XVIII. *En rojo la elección del vértice no visitado más cercano al vértice elegido en la primera etapa del algoritmo*

A) Se calcula el costo acumulado y se compara con el $(costo * p)$

$Costo\ total\ Acumulado\ C \rightarrow D = 8$

$8 \leq [(Costo\ del\ trayecto\ C \rightarrow\ Universidad = 27) * (p = 1.3)] + costo\ del\ trayecto\ C \rightarrow Universidad$

Al ser correcto

B) Se marca como visitado el vértice D

Imagen XIX. *Arreglo de vértices visitados, marcando el vértice C y D como ya visitados*

C) Al arreglo del vehículo que parte del vértice C se agrega el vértice D

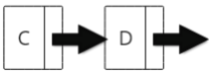

Imagen XX. *Arreglo de vértices que conforman un vehículo, el cual parte del vértice C*

El numeral dos se repite hasta que se llegue al máximo de personas en un carro o hasta que el costo supere la fórmula establecida.

Imagen XXI. *En rojo la selección del vértice no visitado más cercano a D*

Debido a que el vértice C en amarillo – que sería la elección en este paso – Ya está visitado, se elige el siguiente vértice no visitado, en este caso el vértice A

A) Se calcula el costo acumulado y se compara con el $(costo * p)$

$Costo\ total\ Acumulado\ para\ C \rightarrow D \rightarrow A = 20$

$Costo\ total\ Acumulado\ para\ D \rightarrow A = 12$

$20 \leq [(Costo\ del\ trayecto\ C \rightarrow Universidad = 27) * (p = 1.3)] + Costo\ del\ trayecto\ de\ C \rightarrow Universidad$

$12 \leq [(Costo\ del\ trayecto\ D \rightarrow Universidad = 19) * (p = 1.3)] + Costo\ del\ trayecto\ de\ D \rightarrow Universidad$

Al ser correcto

B) Se marca como visitado el vértice A

Imagen XXII. *Arreglo de vértices visitados, marcando el vértice C, D y A como ya visitados*

C) Al arreglo del vehículo que parte del vértice C se agrega el vértice A

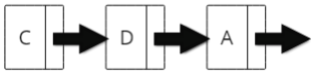

Imagen XXIII. *Arreglo de vértices que conforman un vehículo, el cual parte del vértice C*

Debido a que el carro alcanzo su máxima capacidad permitida, se calcula el costo entre el último vértice visitado y la Universidad, se evalúan las mismas condiciones y de ser válido, se tiene el primer vehículo.

A continuación se vuelve a la primera parte del numeral 3, donde elegimos el siguiente vértice no visita más alejado de la universidad

Imagen XXIV. *En rojo el vértice elegido.*

Debido a que el siguiente nodo (D) ya está visitado, por ende, el siguiente más alejado de la universidad es el vértice B.

Imagen XXV. *En amarillo vértices ya visitados*

Tomando en cuenta que el único vértice que puede ser visitado más de una vez es la Universidad, tenemos que la persona que sale del vértice B, no puede recoger a nadie, debido a que todos ya fueron recogidos, por ende el vehículo número 2 queda de la forma:

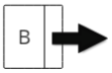

Imagen XXVI. *Segundo vehículo*

En conclusión tenemos que de 4 vehículos, pudimos reducirlos a la mitad, de esta forma:

La persona que sale del vértice C, recoge a la persona del vértice D y a la persona del vértice A:

C → D → A → Universidad

Y un segundo vehículo con la persona que sale del vértice B:

B → Universidad.

4.6 Cálculo de la complejidad del algoritmo

Sub problema	Complejidad	Significado de la variable
Ordenamiento	$O(n^2 \log n)$	Donde n significa la cantidad de Vértices del grafo
Llenar un carro	$O(5n) = 5O(n) = O(n)$	Donde n significa la cantidad de vértices del grafo
Solución Preliminar	$O(n)$	Donde n significa la cantidad de vértices del grafo
Lectura	$O(n^2)$	Donde n significa el número de vértices del grafo
Guardar Archivo	$O(m)$	Donde m significa la cantidad de vehículos
Complejidad Total	$O(n^2 \log n)$	

Tabla 3: *complejidad de cada uno de los sub problemas que componen el algoritmo. Sea N la cantidad de vértices del grafo.*

4.7 Criterios de diseño del algoritmo

Tomando en cuenta que podemos describir puntos específicos en una ciudad como como un grafo y debido a que el problema consta en disminuir la cantidad de vehículos que llegan a un determinado punto – en este caso la universidad –, nos enfocamos en cómo reducir el costo de los trayectos más largos, a través de nuevos caminos, en los cuales se pueden recoger otras personas que posean un costo del trayecto más corto.

Por este motivo decidimos trabajar con LinkedList y ArrayList, debido a que nos proporciona una manera rápida y eficiente a la hora de trabajar con grafos. Si nos fijamos en las tablas de complejidad de cada estructura podemos notar que operaciones como el get() en el ArrayList es de O(1), de igual forma la operación de adicionar (add()) en la LinkedList es de O(1).Adicionalmente, optamos por estas dos estructuras de datos con el fin de ir eliminando cada conjunto de vértices agrupados en un vehículo del ArrayList de la universidad.

4.8 Tiempos de Ejecución

A continuación se mostrara el tiempo de ejecución de cada uno de los conjuntos de datos con 11 vértices y con una constante de tiempo máximo **(*ver [A]*)** variable.

	Constante de tiempo máximo igual a 1.1	***Constante de tiempo máximo igual a 1.2***	***Constante de tiempo máximo igual a 1.3***
Mejor caso	1 ms	1ms	0ms
Caso promedio	2ms	2ms	1ms
Peor caso	2 ms	2ms	1 ms

Tabla 4: *Tiempos de ejecución del algoritmo con diferentes el conjunto de datos de 11 vértices.*

A continuación se mostrara el tiempo de ejecución de cada uno de los conjuntos de datos con 205 vértices y con una constante de tiempo máximo **(*ver [A]*)** variable.

	Constante de tiempo máximo igual a 1.1	***Constante de tiempo máximo igual a 1.2***	***Constante de tiempo máximo igual a 1.3***
Caso promedio	50ms	50 ms	48 ms
Peor caso	56 ms	54 ms	52 ms

Tabla 5: *Tiempos de ejecución del algoritmo con diferentes el conjunto de datos de 205 vértices*

4.9 Memoria

A continuación, se mostrará el consumo en memoria de cada uno de los conjuntos de datos (11 vértices y 205 vértices)

	Conjunto de 11 vértices	***Conjunto de 205 vértices***
Consumo de memoria	2 MG	9 MG

Tabla 6: *Consumo de memoria*

4.10 Análisis de los resultados

Método	***LinkedList***	***ArrayList***
LeerArchivo conjunto de 5 vértices	*-*	*1 ms*
LeerArchivo conjunto de 11 vértices	*-*	*9 ms*
LeerArchivo conjunto de 205 vértices	*-*	*149 ms*
MergeSort conjunto de 5 vértices	*1 ms*	*-*
MergeSort conjunto de 11 vértices	*1 ms*	*-*
MergeSort conjunto de 205 vértices	*72 ms*	*-*
SolucionPreliminar conjunto de 5 vértices	*1 ms*	*-*
SolucionPreliminar conjunto de 11 vértices	*1 ms*	*-*
SolucionPreliminar conjunto de 205 vértices	*18 ms*	*-*

Tabla 7: *Análisis de resultados.*

Tener en cuenta que cuando se llama a SolucionPreliminar, este método llama a CarrosCompartidos

	Conjunto de 5 vértices	***Conjunto de 11 vértices***	***Conjunto de 205 vértices***
Cantidad de vehículos que se requieren con una constante de tiempo máxima de 1.1	*-*	*4 vehículos*	*61 vehículos*
Cantidad de vehículos que se requieren con una constante de tiempo máxima de 1.2	*2 vehículos*	*4 vehículos*	*55 vehículos*
Cantidad de vehículos que se requieren con una constante de tiempo máxima de 1.3	*-*	*4 vehículos*	*51 vehículos*
Cantidad de vehículos que se requieren con una constante de tiempo máxima de 1.7	*2 vehículos*	*-*	*-*

Tabla 8: *Análisis de resultado de carros obtenidos.*

5. CONCLUSIONES

La investigación es uno de los factores más importantes del éxito, por ello es indispensable "empaparse" de datos minuciosos que permitirán un adecuado desarrollo del objetivo, no podemos olvidar el contexto y el medio en el que trabajamos, reconocer trabajos anteriores y tomarlos como guía es un muy buen comienzo.

Siempre hay mejores formas de realizar proyectos y día a día, vamos tomando habilidades de ingenieros que nos permiten ir un poco más allá de lo que creemos correcto, por este motivo, cada día que el equipo de trabajo se sentaba a desarrollar el proyecto encontraba una nueva falla, una nueva manera de hacer algo, (hablando como programadores) un simple condicional que podría solucionar muchos aspectos o un simple arreglo que nos permitiría evitar pasar por un vértice más de una vez.

Con los resultados obtenidos en tiempo de ejecución y cantidad de vehículos que se reducen podemos decir que hubo eficiencia y eficacia a la hora de analizar el problema, la posible solución con las estructuras de datos escogidas y trabajadas.

6. AGRADECIMIENTOS

Agradecemos a la Universidad Eafit por brindarnos día a día nuevas experiencias que permiten nuestro crecimiento tanto en lo personal como en lo profesional.

Agradecemos a nuestros docentes que nos han acompañado durante estos tres semestres de la carrera, enriqueciéndonos día a día con sus conocimientos y experiencias.

Finalmente, agradecemos a nuestros padres y las personas que hacen posible que estemos en una de las mejores universidades del país.

7. REFERENCES

[1] Álvarez C Víctor Andrés. 2015 *Recuperado de:* https://www.elcolombiano.com/antioquia/movilidad/en-medellin-transita-un-carro-por-cada-tres-habitantes-EB3232363

[2] Chung-Shou. Yamming Huang 2014 *Recuperado de:* https://www.sciencedirect.com/science/article/pii/S0304397514001327

[3] Bnya Zahy. Felner Ariel. Eyal Shimony Solomon. (S, f) *Recuperado de:* https://www.aaai.org/ocs/index.php/IJCAI/IJCAI-09/paper/viewFile/487/683

[4] Sabál Antonio 2017 *Recuperado de:* https://blogthinkbig.com/el-algoritmo-que-puede-acabar-con-la-congestion-del-trafico

[5] Barbieri Alberto 2018 *Recuperado de:* https://www.nobbot.com/futuro/algoritmos-reducir-trafico-nueva-york/

[6] Cuevas Carvajal Danilo Alejandro 2013 *Recuperado de: http://opac.pucv.cl/pucv_txt/txt-5000/UCE5372_01.pdf*

[7] *Quispe Montoya Ruben, Ramírez Juárez Braulio 2015 recuperado de: http://www.repositorioacademico.usmp.edu.pe/bitstream/usmp/1474/1/ramirez_jb.pdf*

[8] *Grandas Aguado Pablo 2014 recuperado de: https://e-archivo.uc3m.es/bitstream/handle/10016/27227/TFM_Pablo_Grandas_Aguado_2014.pdf?sequence=1&isAllowed=y*

[9] *Departamento Ciencias de la Comunicación 2014 Escuela de Ingenieros de Antioquia recuperado de http://www.jtech.ua.es/j2ee/publico/lja-2012-13/sesion02-apuntes.html*