

TRÁFICO EN URBES: UNA SOLUCIÓN UTILIZANDO ALGORITMOS Y LA UTOPIA DEL COMPARTIR

Jhonatan Sebastián Acevedo Castrillón
Universidad Eafit
Colombia
jsacevedoc@eafit.edu.co

Manuel Alejandro Gutiérrez Mejía
Universidad Eafit
Colombia
magutierrm@eafit.edu.co

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

Palabras clave

Búsqueda de ruta, Grafos, Carpooling, Algoritmos, Tráfico, Metropolis, Estructura de datos.

Palabras clave de la clasificación de la ACM

Theory of computation → Analysis of algorithms and problem complexity → General → Shortest path

1. INTRODUCCIÓN

En las grandes ciudades una de las problemáticas más grandes que se tratan de solucionar es el tráfico, en algunas solo se vive en las horas punta u horas pico pero en la mayoría es una situación de todo el día. Esta problemática está directamente relacionada con la calidad de vida de las personas generando así diversos cambios que también afectarán el entorno ambiental y social que se percibe en la ciudad.

Esta situación se ha tratado de solucionar con diferentes formas, cada una con matices que son diferentes respecto a cada ciudad pero una solución con la cual se puede manejar la problemática de una manera correcta es el uso de carros que compartan una ruta o un lugar, es muy normal ver en horas de alta afluencia carros con solo un pasajero, por lo que es muy factible que se pueda dar la situación en la que una persona pueda recoger otro compañero o conocido sin desviar mucho su camino.

2. PROBLEMA

El problema consiste en diseñar un algoritmo el cual nos permita dar solución al problema del tráfico en grandes ciudades, para ello recurrimos a que una persona pueda llegar a su destino en el auto con uno o hasta cuatro acompañantes los cuales comparten una ruta parecida o un mismo destino, mejorando así el tráfico en las grandes ciudades en horas pico u horas punta y por consecuencia la calidad de vida de las personas.

3. TRABAJOS RELACIONADOS

3.1 A* based Pathfinding in Modern Computer Games
A* (“A Star”) es una modificación del algoritmo Dijkstra. Este algoritmo fue publicado por Peter Hart, Nils Nilsson y Bertram Raphael del Stanford Research Institute en 1968, este algoritmo busca para una meta el camino más cercano

desde varios lugares, A* prioriza los caminos los cuales están más cercanos al destino siendo así más eficiente que Dijkstra. A* es probablemente el algoritmo más popular para buscar caminos más cortos en los videojuegos.

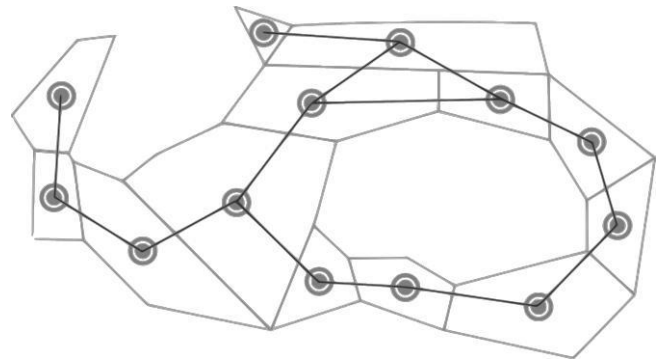


Figura 1: Camino con obstáculos en un videojuego.

3.2 Breadth First Search example (BFS) - How GPS navigation works

Breadth First Search es un algoritmo de búsqueda en un grafo, fue inventado por Konrad Zuse en 1945, BFS explora en términos iguales todas las direcciones posibles. Este algoritmo es el que tienen integrados los GPS normalmente para mostrarte el camino más corto y en algunas mejoras el que menos tráfico tiene.

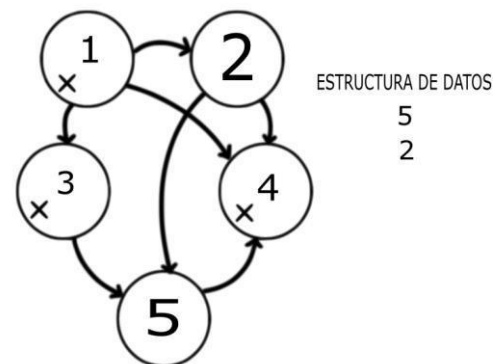


Figura 2: Grafo dirigido con implementación en BFS

3.3 El problema de la gira del caballo

Se trata de un problema clásico cuyo objetivo trata de encontrar una secuencia de movimientos que permitan a una ficha “caballo” visitar cada posición del reconocido tablero ajedrez exactamente una vez, una de esas secuencias se llama “gira”. Claramente es un problema que requiere un pensamiento profundo así como potencia computacional.

La solución más práctica de este problema conocida hasta el momento consta de representarlo como un grafo

movimientos legales de la ficha “caballo” en ajedrez y usar un algoritmo de recorrido de grafos para encontrar una ruta de longitud (filas x columnas -1) donde cada vértice del grafo se visite exactamente una vez.

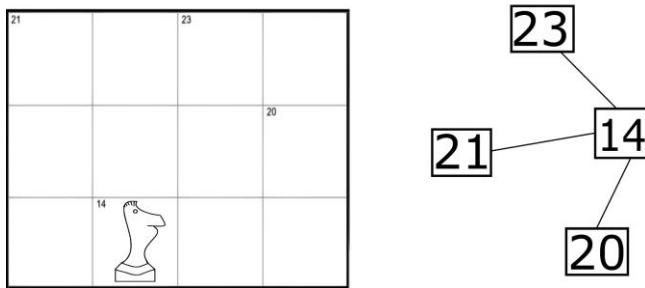


Figura 3: Tablero de ajedrez con la figura del caballo.

3.4 Problema del vendedor ambulante

Este problema se genera a partir de la situación en la que dado un conjunto de ciudades representadas como nodos y las distancias entre cada par de ellas, se quiere buscar ¿cuál es la ruta más corta posible en la que el vendedor visita cada ciudad exactamente una vez y al finalizar la el recorrido regresa a la ciudad de origen?. Se trata de un conflicto en el que se busca ahorrar costes y por lo tanto una mayor eficiencia en el sistema.

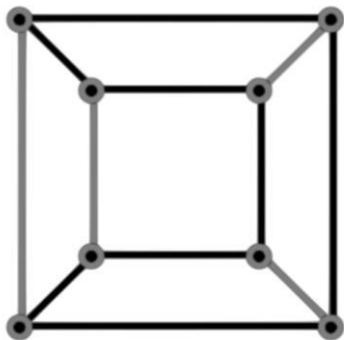
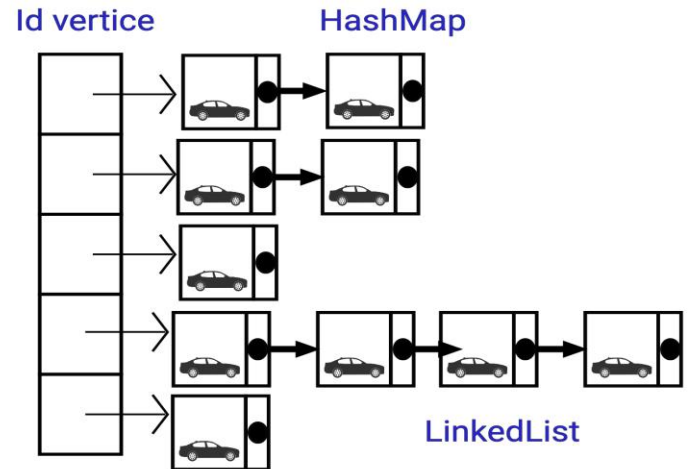


Figura 4: Grafo que representa el problema del vendedor ambulante

4. Representar los vértices cercanos gracias al HashMap.

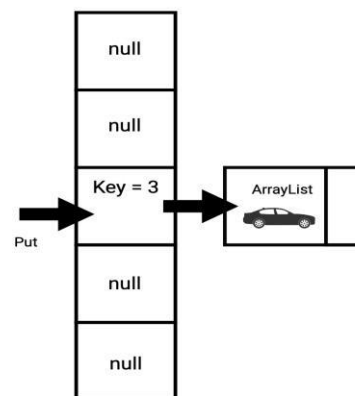
A continuación, explicamos la estructura de datos y el algoritmo.

4.1 Estructura de datos

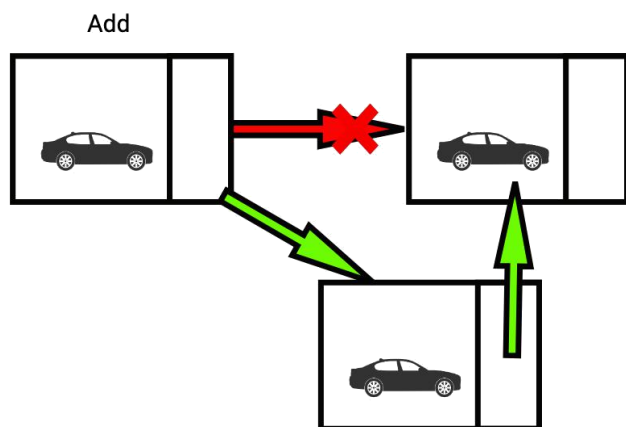


Gráfica 1: HashMap de IdVertice a una LinkedList de autos.

4.2 Operaciones de la estructura de datos



Gráfica 2: Imagen del método put en un HashMap.



Gráfica 3: Imagen del método add en una LinkedList.

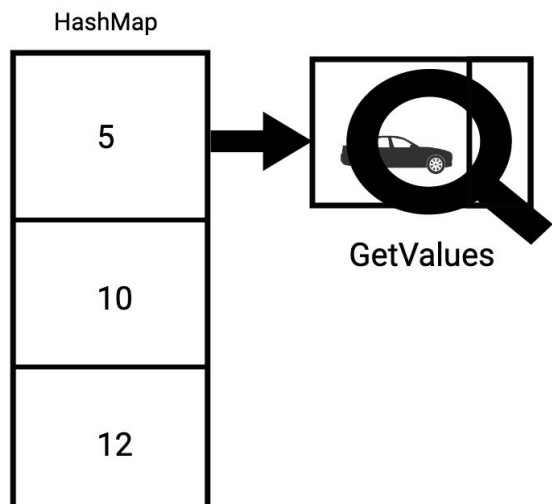
4.3 Criterios de diseño de la estructura de datos

Decidimos escoger el HashMap con valores de LinkedList ya que para el HashMap la complejidad del método acceder y añadir es $O(1)$ además que el HashMap nos permite tener una key la cual podremos usar utilizando operaciones aritméticas para hallarla, en el caso del LinkedList su acceso es $O(n)$ pero su método para insertar tiene complejidad $O(1)$ por lo tanto estas dos estructuras de datos nos ayudarán a mejorar completamente el tiempo de ejecución del algoritmo.

4.4 Análisis de Complejidad


Método	Complejidad
Leer archivo	$O(n^2)$
Separar en el HashMap	$O(n)$
Ingresar en LinkedList	$O(1)$
Escribir archivo	$O(n)$

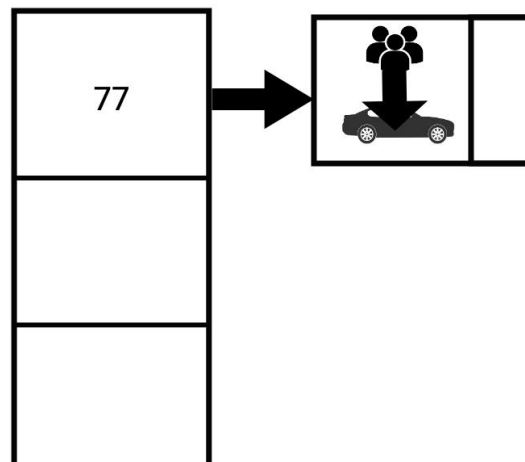
Tabla 1: Tabla para reportar la complejidad



Gráfica 2: Imagen del método getValues en un HashMap.

4.5 Algoritmo


X
Y
-75,5785878
6,203001
HashMap
 $57+20 = 77$



Gráfica 4: Como se introducen las personas en coche dependiendo de sus coordenadas.

4.6 Cálculo de la complejidad del algoritmo

Sub problema	Complejidad
Leer archivo y generar hashmap	$O(n^2)$
Asignar coche a vértices y recorrer grafo	$O(n)$
Generar archivo de ejemplo	$O(n)$
Complejidad Total	$O(n^2)$

4.7 Criterios de diseño del algoritmo

Diseñamos el algoritmo teniendo en cuenta la extensión de los datos a interpretar y la eficiencia más óptima para que el problema pudiese ser solucionado en un tiempo oportuno, además, nos guiamos de la flexibilidad que nos permitió la estructura de datos HashMap consiguiendo resultados aproximados al objetivo. Pero principalmente tuvimos en cuenta ciertos patrones en el grafo resultante y sus caminos cortos lo que nos ayudó en gran medida para la relación de los vehículos con sus ocupantes.

4.8 Tiempos de Ejecución

Caso	Tiempo de ejecución (ms)
Peor	245
Promedio	201
Mejor	198

Tabla 3: Tiempos de ejecución del algoritmo con diferentes conjuntos de datos

4.9 Memoria

	Conjunto de Datos 1	Conjunto de Datos 2
Consumo de memoria	0.142 MB	0.983 MB

Tabla 4: Consumo de memoria del algoritmo con diferentes conjuntos de datos

4.10 Análisis de resultados

	HashMap	Linked List Pair
Memoria	7.5 MB	3.48 MB
Tiempo de búsqueda	145 ms	114 ms
Tiempo de asignación y recorrido	188 ms	196 ms

Tabla 5: Análisis de los resultados obtenidos con la implementación del algoritmo

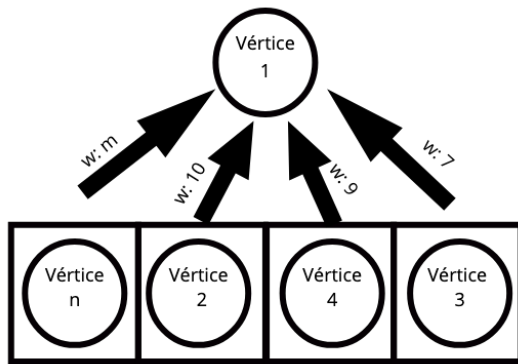
5. ALGORITMO VORAZ CON ESTRUCTURAS DE AGRUPACIÓN PARA LA IMPLEMENTACIÓN DEL CARPOOLING.

5.1 Estructura de datos

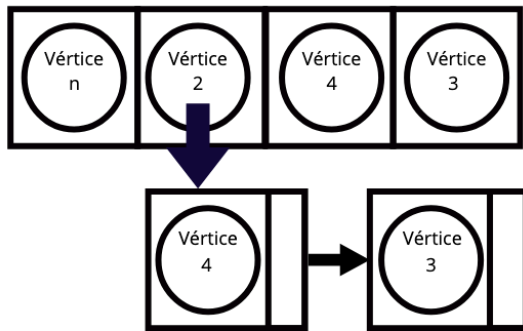
A continuación, explicamos las estructuras de datos y el algoritmo.

	Vértice 1	Vértice 2	Vértice 3	Vértice 4	Vértice n
Vértice 1	0	10	7	9	m
Vértice 2	10	0	3	3	m
Vértice 3	7	3	0	5	m
Vértice 4	9	3	5	0	m
Vértice n	m	m	m	m	0

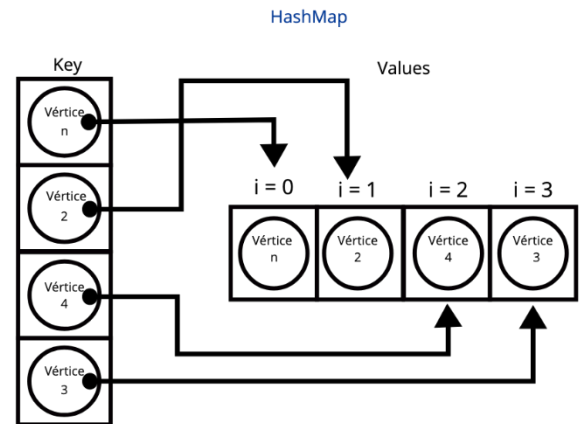
Gráfica 5: Matriz de adyacencia de todos los vértices de nuestro grafo.



Gráfica 6: Arreglo de vértices ordenados desde el más lejano al más cercano al origen.

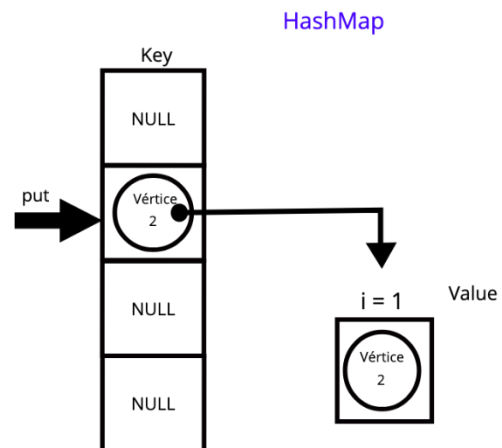


Gráfica 7: LinkedList dentro de un objeto vértice donde están todos los vecinos de menor a mayor de ese objeto vértice.

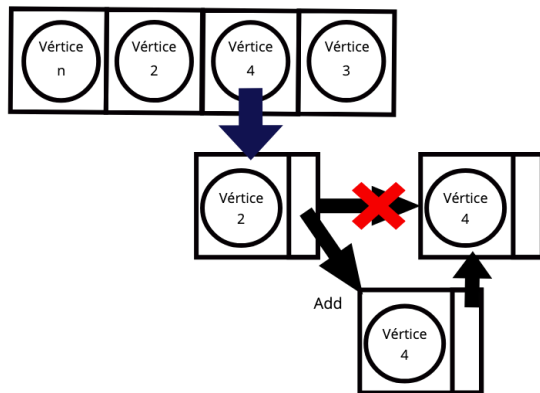


Gráfica 8: HashMap donde guardamos las posiciones de nuestro arreglo de vértices más lejanos.

5.2 Operaciones de la estructura de datos



Gráfica 9: Imagen de una operación de insertar un valor en un HashMap



Gráfica 9: Imagen de una operación de insertar un valor en una LinkedList

5.3 Criterios de diseño de la estructura de datos

Nuestra primera estructura de datos es una matriz de adyacencia, decidimos representar todos los vértices así ya que es un grafo completo y esto nos genera una matriz cuadrada, por otro lado el acceso en una matriz es $O(1)$ aunque sacrifiquemos memoria de almacenamiento reducimos la complejidad del algoritmo, nuestra segunda estructura de datos es un arreglo, optamos por esta estructura ya que como anteriormente conocemos el tamaño de este y esto facilita la inserción ya que en un Array es $O(1)$, nuestra tercer estructura de datos es una LinkedList, nos basamos en ella ya que añadir en una LinkedList es $O(1)$ y es dinámica, caso contrario de un ArrayList que aunque también su tamaño es dinámico añadir en el peor de

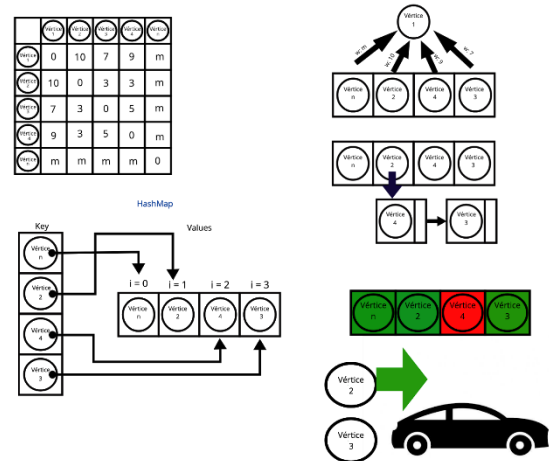
los casos es $O(n)$, la última estructura que utilizamos es un HashMap ya que es dinámico y su búsqueda y acceso es $O(1)$

5.4 Análisis de Complejidad

Método	Complejidad
Crear matriz con todos los vértices	$O(N^2)$
Guardar vértices en un arreglo de vértices	$O(N)$
Asignar arcos a los vértices	$O(N^2)$
Ordenar el arreglo de vértices	$O(N \log(N))$
Ordenar la LinkedList de vértices	$O(N \log(N))$
Guardar la posición del vértice	$O(N)$
Imprimir coches con sus vértices	$O(N^2)$

Tabla 6: Tabla para reportar la complejidad

5.5 Algoritmo



Gráfica 9: Imagen donde interactúan nuestras estructura de datos con un arreglo de visitados, finalmente se ingresan los vértices al auto.

5.6 Cálculo de la complejidad del algoritmo

Sub problema	Complejidad
Crear la matriz con los vértices adyacentes y su peso	$O(N^2)$
Crear los objetos vértices con su respectivos pesos	$O(N^2)$
Organizar vértices más lejano al más cercano	$O(N \cdot \log(N))$
Organizar vértices en la LinkedList del más cercano al más lejano	$O(N \cdot \log(N))$
Añadir puntero en el HashMap	$O(N)$
Organizar vértices en los coches	$O(N^2)$
Complejidad total	$O(N^2)$

Tabla 7: complejidad de cada uno de los sub problemas que componen el algoritmo. Donde N es el número de vértices.

5.7 Criterios de diseño del algoritmo

Decidimos utilizar un algoritmo voraz ya que aunque no da una respuesta óptima en todos los casos si se acerca mucho a la adecuada, además su complejidad no es tan alta como otro tipo de algoritmos, ya que partimos de que la complejidad de

la lectura del archivo es $O(N^2)$ y nuestro algoritmo se mantiene en esa complejidad.

5.8 Tiempos de Ejecución

	<i>Conjunto de Datos 1</i>	<i>Conjunto de Datos 2</i>	<i>Conjunto de Datos 3</i>	<i>Conjunto de Datos 4</i>
Mejor caso	155 ms	71 ms	69 ms	72 ms
Caso promedio	164.3 ms	76.3 ms	72.4 ms	73.1 ms
Peor caso	175 ms	101 ms	77 ms	75 ms

Tabla 8: Tiempos de ejecución del algoritmo con diferentes conjuntos de datos

5.9 Memoria

	<i>Conjunto de Datos 1</i>	<i>Conjunto de Datos 2</i>	<i>Conjunto de Datos 3</i>	<i>Conjunto de Datos 4</i>
Consumo de memoria	44 MB	14.4 MB	9.6 MB	9.47 MB

Tabla 9: Tiempos de ejecución del algoritmo con diferentes conjuntos de datos

6. CONCLUSIONES

Después de realizar la investigación necesaria acerca de la problemática tratada anteriormente en la introducción, decidimos implementar una solución algorítmica con base en el car-sharing. El producto fue una reducción considerable de vehículos que llegan a un mismo destino, aunque esta reducción no es máxima ya que no se tienen en cuenta ciertos casos respecto a la ubicación de las personas a agrupar por el propio funcionamiento del algoritmo, sin embargo, es una solución competente y eficiente a la problemática lo que permite aplicarla a grandes urbes citadinas, que es donde la contaminación y el exceso de flujo vehicular se manifiestan en mayor medida. De igual manera concluimos que se logró un avance notable tanto en eficiencia como en eficacia frente a la primera propuesta porque conseguimos que se tuvieran en cuenta factores hipotéticos de demora así como una disminución en el tiempo de ejecución al determinar y ordenar los datos recolectados. Finalmente, podríamos mejorar la propuesta en un futuro centrándonos en la diferencia entre vehículos y la máxima cantidad de ocupantes que esto representa, así como el estado actual

de las vías y el tráfico, además de tratar dinámicamente la ubicación de las personas a recoger.

6.1 Trabajos futuros

Próximamente nos gustaría adaptar la implementación a casos con posiciones dinámicas donde las personas puedan desplazarse cierta distancia en diferentes instantes de tiempo para que la reducción de vehículos en el destino sea aún mayor. También pensamos que sacrificar la eficiencia en una solución alterna para aumentar la efectividad podría ser útil en otras situaciones donde el tiempo de ejecución no sea un problema, por ejemplo, en destinos con una problemática vehicular menor.

AGRADECIMIENTOS

Agradecemos por el apoyo con la determinación de ciertas estructuras a los compañeros de curso José Romero y Kevin Herrera, que sirvieron para disminuir el tiempo de ejecución y el consumo de memoria de la implementación.