An approach to Carpooling, an implemented solution for traffic and contamination problems.

Alfredo José Ospino Ariza
University EAFIT
Colombia
ajospinoa@eafit.edu.co

Jonatan David Acevedo Lopez
University EAFIT
Colombia
jdacevedol@eafit.edu.co

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

ABSTRACT

What is the problem?

- Upgrading the city's movability, and solving a percentage of the city's contamination spreading.

Why is the problem important?

- Because it´s a major issue within society's deepest symptoms.

Which are the related problems?

- The increase of contaminated environments for those with breathing problems and it also affecting those who don't have any breathing issue.

Keywords:
Controlled, environment, linked, list, hash, map.

ACM CLASSIFICATION Keywords

Theory of computation → Design and analysis of algorithms → Graph algorithms analysis → Shortest paths

1. INTRODUCTION

The city of Medellin is overpopulated at least when it comes to cars, this situation has existed since 2014, since then the car population has at least doubled, so the idea with this algorithm is to optimize the way the people of the city transport and travel through it.

2. PROBLEM

The problem simplified, is that there are too many cars for the so little and so tight streets, in return causing the air to be polluted, the traffic to be terrible, and the traffic lights to be unbearable.

3. RELATED WORK

3.1 The bridges of Konigsberg

This seems to be unsolvable, but this was resolved through the theory of graphs.

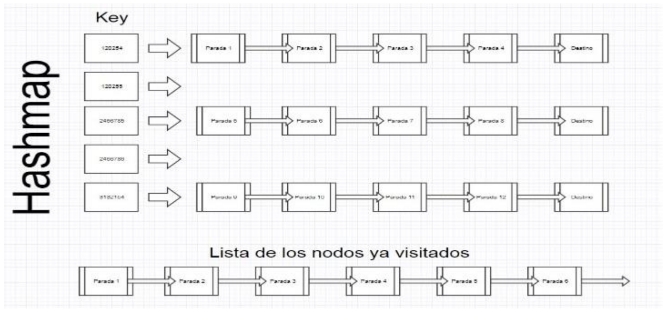
3.2 Shortest paths

This problem consists in trying to find the path between 2 nodes in a way that the sum of the paths weights results in the minimum. It's solved with Dijkstra's algorithm.

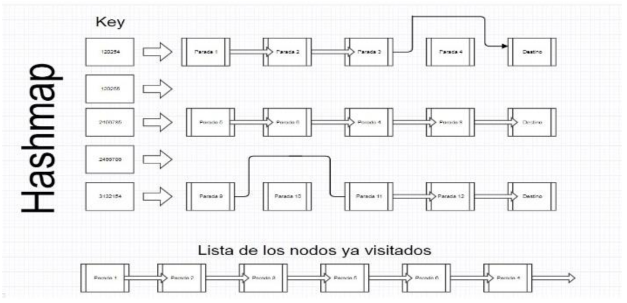
4. HashMap*

As the name implies, it uses HashMap as the data structure, it also uses linked lists, for the values, what it does is really use traverse the map, then traverse the list with a resemblance to binary search.

4.1 Data Structure



4.2 Operations of the data structure



4.3 Design criteria of the data structure

We designed this data structure because we thought that we could take the keys as each vehicle and start from linking it with all the nodes (through a list that is an object that has the coordinates, distance and current capacity of the car) to the ones it can go, this way it goes to the one that is closer to him. Once the vehicle capacity has been filled, these nodes will be saved in a list so that they are not included in the next routes (This is done since all the nodes connect with all and in this way they will only make the movement towards the nearest node optimizing the route in the best way) of the vehicle, the keys are created from nodes and those will be taken as the number of vehicles needed.

4.4 Complexity analysis

The structure's complexity is basically overall O(n log n));

- O(1) data insertion into the linked list.
- O(n) in key insertion into the HashMap

N being the number of nodes the graph has, or the spots the car needs to reach.

4.5 Algorithm

The algorithm we implemented creates "Cars" that have a max capacity of 5, that basically uses a new car after the previous one was filled, and starts with the furthest away points and finishes with closest ones.

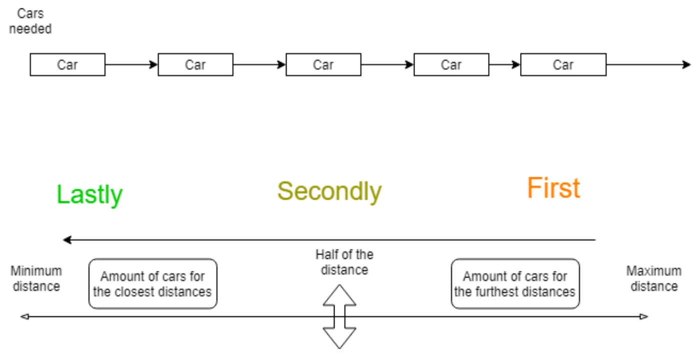


Figure 3: The addition of cars into the list of used and filled cars.

4.6 Complexity analysis of the algorithm

The variables used in this algorithm are:

- N = the number of nodes the map has.
- A = the amount of line that are read of the data set.
- C = the number of cars that are being used to "car-pool".
- L = the length of the list of the first value of the map.

The calculated complexities for the algorithm are:

Subproblems	**Complexity**
Adding nodes(vertexes) to the map	O(A)
Obtaining the furthest, "middlest", closest away node and adding the filled cars to the list	O(L)
Finding the cars needed to do the carpooling.	O(L)
Total complexity	O(L*Log n)

Table 2: Complexity of each subproblem that is part of the algorithm.

4.7 Design criteria of the algorithm

We based ourselves on something similar to a Binary Search, because this solution is very convenient for our problem, which is finding the least number of needed cars in a list, so at the end it is not very different to the previously mentioned algorithm.

4.8 Execution times

We measured the algorithms time using the Java system's own measuring tool, "*System.nanoTime()*" function, and we were given these results:

	Dataset 205	***Dataset 11***	***Dataset 4***
Best case	0.0003 s	0.00025s	0.000241 s
Average case	0.00045 s	0.000425s	0.000395 s
Worst case	0.00062 s	0.0006 s	0.00055 s

Table 3: Execution time of the algorithm for different datasets.

4.9 Memory consumption

Measure memory consumption of the algorithm for different datasets

	Dataset 205	***Dataset 11***	***Dataset 4***
Memory consumption	8.4 MB	16 MB	16 MB

Table 4: Memory consumption of the algorithm for different datasets.

4.10 Analysis of the results

	Dataset 205	**Dataset 11**	**Dataset 4**
Memory consumption	**8.4 MB**	**16 MB**	**16 MB**

Table 5: Analysis of the results obtained from the algorithm execution.

6. CONCLUSIONS

So, to resume we basically did binary search
in a list basing ourselves in the HashMap we had, and, we found the closest vertex and furthest vertex from the center which is EAFIT.

The algorithm really completes the problem in the least complicated way, it takes very little time but consumes a big amount of memory, but still, it very much fulfills the goal, that doing the least number of cars needed.

6.1 Future work

We would like to improve the algorithm’s memory consumption and also the overlapping of the algorithm’s final result, basically repeating some the already needed cars.

Also, dismiss the use of Linked Lists inside the HashMap.