Path Finding Algorithm Implementation for Delivery Service in Medellin

Ricardo Rafael Azopardo Cadenas
Universidad Eafit
Colombia
rrazopardc@eafit.edu.co

Jhon Jairo Chavarria
Universidad Eafit
Colombia
jjchavarrg@eafit.edu.co

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

ABSTRACT

The objective of this project was to design a path finding algorithm with minimum cost for a delivery service in Medellin, for this purpose of understanding algorithm design criteria and data structure selection. The selection criteria of this algorithm must guarantee the following, the solution must be of or near to minimum cost in the path found and the algorithm must have high performance for the size of graph being worked. The solution we came up with was the A* algorithm, which had pleasing results in time complexity, execution time and memory consumption and we can conclude the A* algorithm is an excellent choice for this type of optimization problems.

Author Keywords

Data Structures, TSP, programming, Algorithms, optimization problems, shortest path, minimum cost.

ACM Classification Keywords

- CCS → Theory of computation → Design and analysis of algorithms
- CCS → Mathematics of Computing→ Discrete Mathematics→ Data structures → Data access methods.

INTRODUCTION

In computer science and algorithmic design, optimization problems will always be a topic of interest, which means given a problem find the best solution from all feasible solutions. Finding efficient and optimal solutions, and specially of minimum cost are of great importance in many areas, such as taking important and accurate decisions in logistic and planning which contributes in interesting and important topics such a sustainable development and pathfinding.

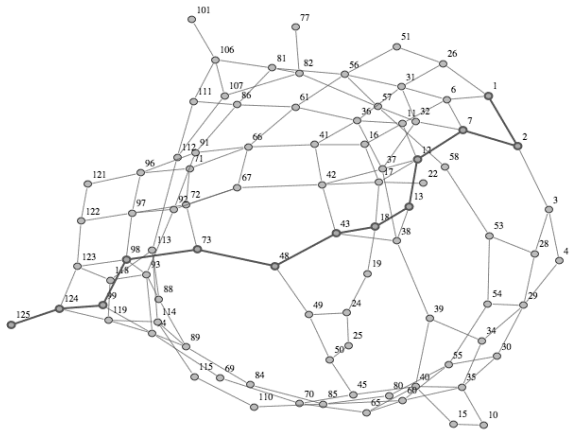

Figure 1. Given a set of points, find the shortest path between two points. This is a popular optimization problem, in this report we will treat a similar more complex issue.

PROBLEM

In this report we treat a problem similar to the traveling salesman problem, which simulates a home delivery service, given a city bidimensional map and certain points in the city determine how to visit each point at least once and return to the starting point finding the shortest path and with minimum cost.

RELATED WORK

Greedy Algorithms: Closest Neighbor

Greedy algorithms achieves an optimal solution for a given problem, based on the principle: in each point make the most efficient decision, this will approximate the best global result. For the problem of the TSP the application of the greedy algorithm may throw undesirable results, the algorithm always makes the decision that seems to be the best at that time, and only has an opportunity to choose an optimal solution and never retreats or reverses its decisions. In this specific problem choosing the best path at a time will make for an optimal solution, but not the correct one, greedy algorithms for TSP like problems must be complemented with graph heuristic that serves as a guide for the greedy method.

Minimum Spanning Tree

A minimum spanning tree is a representation of a weighted, connected and undirected graph, it constructs a tree representation of the original graph with proportional weights calculated using the graph original weights. This data structure is efficient for TSP like optimization problems and can find optimal solutions in polynomial times, in this case, the map forms a directed graph, and so using a minimum spanning tree structure may require the reconstruction of the graph in an undirected form.

Lin–Kernighan Heuristic

The Lin-Kernighan heuristic, is a combinatorial optimization methods which involves swapping possible paths to form the best possible complete path between two points. Heuristic methods such as the Lin-Kernighan heuristic will be required to find optimal solutions and complement greedy methods for correctness.

A* Algorithm

The A* algorithm is an heuristic improved extension of the Dijkstra Algorithm for finding the shortest paths between nodes, briefly the algorithm calculates the distance between the starting point to all other nodes and heuristically calculates the distance between the end point to all other nodes, using both distances the algorithms constructs a path taking in considerations both starting point and end point distances. Taking both distances in consideration a closest neighbor algorithm will be correctly guided and will efficiently take the best route to the end node, without the need to visit all nodes and finding a solution equal or close to the best solution.

DATA STRUCTURES

The data structure implemented in this project consists of multiple HashMaps and ArrayLists. The composition of our Data Structure goes as follows: The data structure is composed of a HashMaps with semesters as keys and points to ArrayLists of courses as values, in these lists there are Course objects with a HashMap of the students in the course, this HashMap has students as keys and their final grades in the course as values.

Additionally, every Course object also have an ArrayList of the students in the course.

This HashMap based structure was selected because the complexity of most search commands in this implementation are O(1), also the HashMaps are highly convenient for this specific problem, because most of the values point to another value.

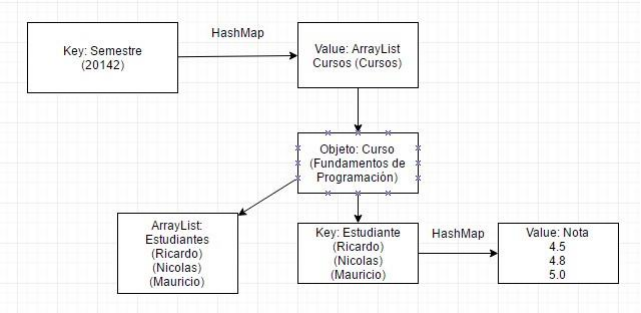

Figure 2. HashMap based implementation of the Data Structure.

ALGORITHM

The implementation need to work with two main search algorithms.

The first one will print all the students in a specified course (in a specific semester) and their final grade. The method used to implement this algorithm receives the date of a semester and the name of a course, the method will check directly in the HashMap of the specific semester and will search in the ArrayList for the specific course, afterwards the program will print every student in the ArrayList of students inside the course, and their final grade pointed in the students' HashMaps.

The second method receives the name of a student and the semester they are in, and prints all the courses the student is currently viewing and their final grade in each one. The method will check directly in the HashMap of the specific semester and search in all the courses for the student, every time the student is found the name of the course will be printed next to the final grade of the student in the course.

Both algorithms have a complexity of O(n*m) for the worst case.

Data Structure Design Criteria

The main criteria in consideration for the design of the data structure was the time complexity used for both searches and the memory consumption of the program. The HashMap based data structure was designed in a way in which all values point directly to another value while avoiding unnecessary object repetition, ex. avoid repeating courses for every student. The data structure was implemented in a way in which most searches would have a O(1) complexity and a O(n*m) in the worst case scenario.

Complexity Analysis

Complexity analysis for main methods:

Method	Complexity
agregarSemestre	O(1)
agregarCurso	O(n^2)
agregarEstudiante	O(1)
agregarNota	O(n)
obtenerEstudiante	O(n)
buscarCursosYNotasFinales	O(n*m)
buscarEstudiantesEnUnCurso	O(n*m)

Table 1: Complexity Report Table

HashMap Based Data Structure Implementation

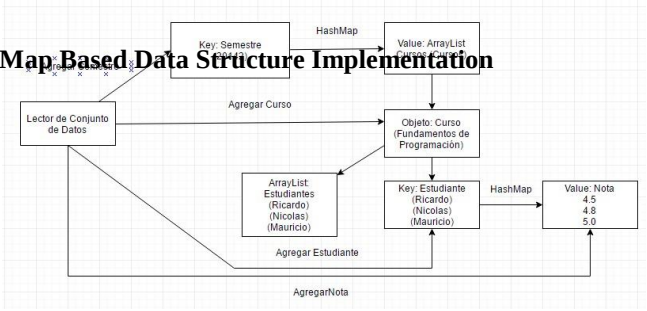

Figure 3. Final Implementation of the Data Structure.

Data Structure Operations

Students and Final Grade in a Specific Course Search:

The method for this function receives the date of a specific semester and the name of a specific course as Strings. The method will use the semester string as the as key in the first HashMap which points to the ArrayList of the courses in that Semester, afterwards the method will use the course string for searching for the specific course in the ArrayList, afterwards the method will print all the students in the specific course using the Students ArrayList inside the course object and will use their names as keys for their final grades in the Students → Grade HashMap inside the course, finally printing their final grade.

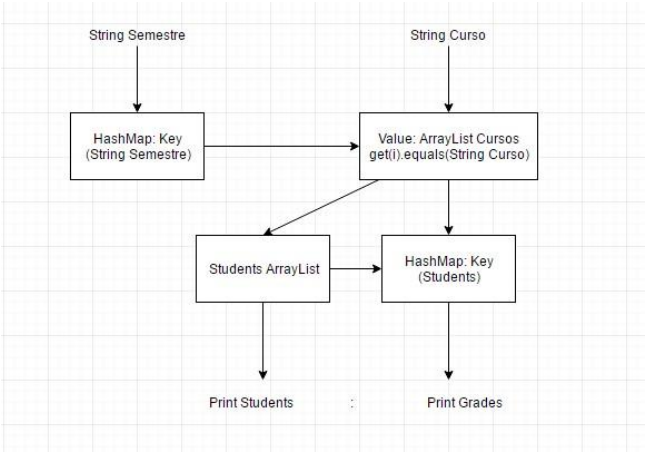

Figure 4: Students and Final Grades Search sequence figure.

Course and Final Grades of a Specific Student Search:

The method for this function receives the date of a specific semester and the student code of a specific student as Strings. The method will use the semester string as the as key in the first HashMap which points to the ArrayList of the courses in that Semester, afterwards for each course the method will check if the HashMap of each course contains the specific student of the student code received as a string, if the method finds the student, it will print the course containing the HashMap and the final grade of the specific student in that course.

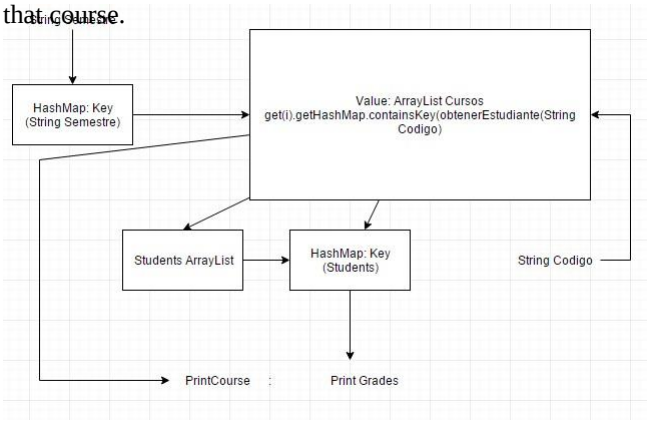

Figure 4. Final Implementation of the Data Structure.

Execution Time:

Methods	Estructura de Datos I	Estructura de Datos II	Fundamentos de Programación
Creation	17 milliseconds	17 milliseconds	14 milliseconds
Student and Grades Search	1 millisecond	3 milliseconds	2 milliseconds
Course and Grades Search	1 millisecond	0 milliseconds	0 milliseconds

Table 2. Execution times for the 3 main operations.

Memory Usage

The memory usage for the whole excel table was 2 MB and stays in a static value.

Results Analysis

The results for the data structure are highly efficient, the process that takes up most of the execution time is the scanning and creation of the data structure. The two main search algorithms individually takes minimum to no time. The memory usage has a static value of 2 MB.

Functions	Process Time
Creation	15-20 milliseconds
1 Student and Grade Search	1-3 milliseconds
20 Students and Grade Searchs	10-20 milliseconds
1 Course and Grade Search	0 milliseconds
20 Course and Grade Searchs	0-5 milliseconds

Table 3. Time Table for main operations

CONCLUSIONS

Our final project was a great experience for our study's development. We had big consideration at the time of picking the data structures, we were thinking on how they work, how we are going to join them together, and how when we have all ready, how we were going to fill up those structures with the needed data. We thought of a way in which it was going to be easy, efficient, have low memory cost and we came up with a way in which we could say that it nearly approximated to what we thought was going to be the best solution. When the coding started, everything was going according to plan. The methods were correctly written and implemented, but the most important accomplishment of our project was to manage the insertion of data. This is because we used mostly the semesters and courses, and that brought a bit of problems which were all solved after time passed. When we first thought of a solution, it was kind of impossible because we thought that the binary trees were going to be easy to implement, but it wasn't that easy. So we decided to reevaluate the situation, and then we thought to change the data structure. Clearly it was a good decision which lead us to the completion of this project. This project is just a base of what it could really be. The multiple ways we could make this even better are almost infinite, because when managing data structures, it almost lets you use a big set of data and it lets you manipulate it however you like. Finally the problem that was most difficult to solve for us was when we tried to add the data to the structures, because some data was duplicated, some was overwritten, and some of the data was not even written, but all these issues were solved progressively and now the project is done.

Future projects

This project we could catalog as very generic. The management of the data structures could be the big base of a well done program for a company, for example. This being said, we could use this project as the base of a big project on the near future. If we go to an industry and are asked for a similar project, we could use this one and upgrade it to the way which is not only used for a small data base, but for big ones also, with more functionalities, and for a big cause.

AKNOWLEGMENTS

This project was made by Nicolas Gonzalez y Ricardo Azopardo, but it wouldn´t have been possible without the support of the following people: we want to thank our senior Juan Pablo Calad Henao for the constant support and help in constructing our data structure and correcting errors in our code. We also want to thank our senior Diego Antonio Fonseca Guzman for his guidance in helping us choose the correct data structure model.

REFERENCES

[1] https://docs.oracle.com/javase/7/docs/api/java/util/Hash Map.html

[2] https://en.wikipedia.org/wiki/Sparse_matrix

[3] Ulman, J. and Codd, E. and Stonebraker, M. and Aho, A. and Astrahan, M and Chamberlin, D. The Relational Data Model. Ullman, 1988, 48.

[4]http://stackoverflow.com/questions/10299560/javahashm ap-2d-cant-get-the-right-approach-to-make-a-2dhashmap-imean-a-hash