**Introduction**

In traveling, there are so many possible routes to take to go to a certain destination. However, there are chances that most routes taken are not optimal since some are of higher costs. If such happens, resources and time are rather wasted than maximized. The optimized solution is the route with the least total distance from one source to a certain destination, regardless of the traffic condition.

Since most of the transactions with the government centers are needed to be done personally, the group developed a mapping system which is accessible in all government centers found in Quezon City. The target users are people who want to process their credentials or documents that are related to a certain government center.

**Background of the Project**

Since most of the transactions performed in the government centers are personal, there is a need to go to the certain place. And sometimes, there is a chain of transactions to be done sequentially from one place to another.For example, someone has to go to DENR to get a permit for a certain project then he will go to PhilCOA for his coconut plantation registration. What if he wastes his resources traversing through the Office of the Ombudsman, DOJ, PDEA, etc., without any need to go to such centers? That’s why the group made a project that applies Dijkstra’s algorithm to provide an optimal path to go from one place to another to prevent excessive waste of resources and time.

**Project Objective**

This Project aims to:

1. Help the users identify the shortest path to take when traveling from a certain location to any government center along the Quezon City.

2. Test the effectiveness of Dijkstra’s Algorithm in finding the single source shortest path, optimally, even in real road networks.

3. Produce a result with the use of a program that generates a solution based on the problem and the user’s input.

4. Determine the least total distance between the starting node and the end node inputted by the user.

**Scope and Limitation of the Project**

This project is limited only in the government centers in Quezon City and was intended to guide government workers, as well as people who need to transact with a certain government center.

This project does not cover the private sectors and the public utility jeepney drivers within the area.

**Review of Related Literature**

**Three Fastest Shortest Path Algorithms on Real Road Networks: Data Structures and Procedures**

By: F. Benjamin Zhan

It is well known that computing shortest paths over a network is an important task in many network and transportation related analyses. Choosing an adequate algorithm from the numerous algorithms reported in the literatur3e is a critical step in many applications involving real road networks. In a recent study, a set of three shortest path algorithms that run fastest on real road ne3tworks has been identified. These three algorithms are: 1) the graph growth algorithm implemented with two queues, 2) the Dijkstra algorithm implemented with approximate buckets, and 3) the Dijkstra algorithm implemented with double buckets. As a sequel to that study, this paper reviews and summarizes these three algorithms, and demonstrates the data structures and procedures related to the algorithms. This paper should be particularly useful to researchers and practitioners in transportation, GIS, operations research and management sciences.

**Dijkstra’s Algorithm On-Line: An Empirical Case Study from Public Railroad Transport**

Traffic information systems are among the most prominent real world applications of Dijkstra’s algorithnm for shortest paths. We consider the scenario of a central information server in the realm of public railroad transport on wide area networks. Such a system has to process a large number of online queries for optimal travel connections in real time. In practice, this problem is usually solved by heuristic variations of Dijktra’s algorithm, which do not guarantee an optimal result. We report results from a pilot study, in which we focused on the travel time as the only optimization criterion. In this study, various speed up techniques for Dijkstra’s algorithm was analyzed empirically. This analysis was based on the timetable data of all German trains and on snapshot of half a million customer queries

**Algorithm Design**

The design of the algorithm is implementing Dijkstra’s algorithm. We used an array data structure, where each element stores the coordinate of the place, a linked list which serves as an adjacency list referencing a set of nodes, a double precision variable that carries the distance from the source, an integer which references the node incident to such node, and a Boolean variable to determine if that certain node is already traversed or not. Since the user’s input is based on a choice selection, we get the selected index of the choice and set the referenced node array’s total distance by 0. Then it perform repetitive exhaustive search for the unvisited node with the smallest total distance from the source. Then,it will check the adjacency list of that node and set the distance of the adjacent node from the referenced node.For computing the distance, we used the distance formula plus the distance from the source to the referenced node, if the distance calculated than the total distance is carrying, then it will set the total distance of the adjacent node to the calculated distance and the origin of the adjacent node will be referencing to the reference node. The steps will repeat until the smallest, unvisited node is the ending node.

**Algorithm Simulation**

**NO**

**x = INDEX OF THE SELECTED SOURCE?**

**x < 42**

**NO**

**YES**

**START DIJKSTRA’S ALGORITHM**

**x = 0**

**x++**

**graph[x].total = 0**

**A**

**BREAK**

**YES**

**BEGIN**

**INITIALIZE THE GRAPH**

**INPUT SOURCE AND DESTINATION BY CHOOSING IN THE CHOICEBOX**

**USE DIJKSTRA’S ALGORITHM**

**OUTPUT THE SHORTEST PATH WITH ITS TOTAL DISTANCE**

**END**

**SOURCE = DESTINATION?**

**NO**

**YES**

**SHOW DIALOG BOX “ALREADY AT THE DESTINATION”**

**G**

**STOP DIJKSTRA’S ALGORITHM**

**F**

**while(true)?**

**YES**

**NO**

**A**

**G**

**BREAK**

**s = x**

**BREAK**

**s++**

**NO**

**x < 42**

**YES**

**x++**

**NO**

**!graph[x].visited && graph[x].total < graph[s].total**

**YES**

**x = s**

**IS graph[s] VISITED?**

**NO**

**YES**

**s = 0**

**A**

**C**

**util = util.next**

**D**

**graph[s].visited = true**

**F**

**E**

**NO**

**s = DESTINATION?**

**YES**

**ListNode util = graph[s].Arcset.head()**

**E**

**G**

**D**

**B**

**C**

**tempIndex = s**

**tempIndex = s**

**tempDist < graph[util.data].total?**

**NO**

**YES**

**B**

**NO**

**CONTINUE**

**tempDist = graph[s].total + Math.abs(Math.sqrt((Math.pow((graph[s].x - graph[util.data].x),2)) + (Math.pow((graph[s].y - graph[util.data].y),2))))**

**YES**

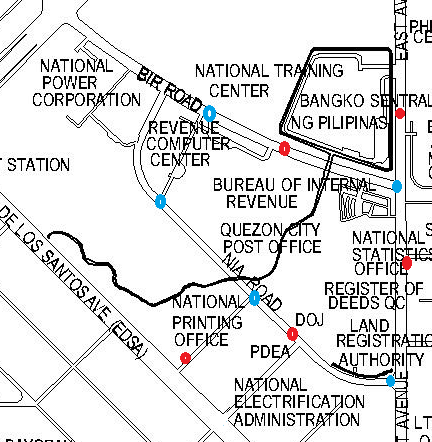
**IS graph[util.data] VISITED?**

**util != null?**

**YES**

**NO**

In a Node array of 42 elements, we applied Dijkstra’s Algorithm for the graph problem. Let us observe on how the algorithm works in this graph fragment.



First, the algorithm initializes the Array of nodes and set up their adjacency lists and coordinates. Then we set all of their total distances to infinity and their origin to the value of their index in the array and their Boolean variable visited is set to false.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Node Name | Index | Total Distance | Coordinates | Origin | Visited | Adjacency List |
| BSP | 0 | MAXVALUE | (399,133) | 0 | False | 1 |
| DN(BSP,NSO) | 1 | MAXVALUE | (391,187) | 1 | False | 0,2,3 |
| NSO | 2 | MAXVALUE | (406,262) | 2 | False | 1,5 |
| BIR | 3 | MAXVALUE | (285,152) | 3 | False | 1,4 |
| DN(BIR) | 4 | MAXVALUE | (209,109) | 4 | False | 3,6 |
| DN(DOJ,NSO) | 5 | MAXVALUE | (390,381) | 5 | False | 2,7 |
| DN(NIA,BIR) | 6 | MAXVALUE | (163,204) | 6 | False | 4,8 |
| DOJ | 7 | MAXVALUE | (295,332) | 7 | False | 5,8 |
| DN(DOJ,NPO) | 8 | MAXVALUE | (256,300) | 8 | False | 6,7,9 |
| NPO | 9 | MAXVALUE | (185,387) | 9 | False | 8 |

Note: DN=dummy Node. MAXVALUE= Infinite

Then the user makes an input. Let us assume that the User Input is from BSP to NPO. First we set the total Distance of the User’s input of Starting node to zero. Then perform Exhaustive search for the Node with the smallest total distance. Then we assign values via the total distance from the starting node, plus the Distance formula from the referenced node to the adjacent nodes.

0th Pass:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Node Name | Index | Total Distance | Coordinates | Origin | Visited | Adjacency List |
| BSP | 0 | 0 | (399,133) | 0 | False | 1 |
| DN(BSP,NSO) | 1 | MAXVALUE | (391,187) | 1 | False | 0,2,3 |
| NSO | 2 | MAXVALUE | (406,262) | 2 | False | 1,5 |
| BIR | 3 | MAXVALUE | (285,152) | 3 | False | 1,4 |
| DN(BIR) | 4 | MAXVALUE | (209,109) | 4 | False | 3,6 |
| DN(DOJ,NSO) | 5 | MAXVALUE | (390,381) | 5 | False | 2,7 |
| DN(NIA,BIR) | 6 | MAXVALUE | (163,204) | 6 | False | 4,8 |
| DOJ | 7 | MAXVALUE | (295,332) | 7 | False | 5,8 |
| DN(DOJ,NPO) | 8 | MAXVALUE | (256,300) | 8 | False | 6,7,9 |
| NPO | 9 | MAXVALUE | (185,387) | 9 | False | 8 |

1st Pass:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Node Name | Index | Total Distance | Coordinates | Origin | Visited | Adjacency List |
| BSP | 0 | 0 | (399,133) | 0 | True | 1 |
| DN(BSP,NSO) | 1 | 54.58 | (391,187) | 0 | False | 0,2,3 |
| NSO | 2 | MAXVALUE | (406,262) | 2 | False | 1,5 |
| BIR | 3 | MAXVALUE | (285,152) | 3 | False | 1,4 |
| DN(BIR) | 4 | MAXVALUE | (209,109) | 4 | False | 3,6 |
| DN(DOJ,NSO) | 5 | MAXVALUE | (390,381) | 5 | False | 2,7 |
| DN(NIA,BIR) | 6 | MAXVALUE | (163,204) | 6 | False | 4,8 |
| DOJ | 7 | MAXVALUE | (295,332) | 7 | False | 5,8 |
| DN(DOJ,NPO) | 8 | MAXVALUE | (256,300) | 8 | False | 6,7,9 |
| NPO | 9 | MAXVALUE | (185,387) | 9 | False | 8 |

2nd Pass:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Node Name | Index | Total Distance | Coordinates | Origin | Visited | Adjacency List |
| BSP | 0 | 0 | (399,133) | 0 | True | 1 |
| DN(BSP,NSO) | 1 | 54.58 | (391,187) | 0 | True | 0,2,3 |
| NSO | 2 | 76.49 (131.07) | (406,262) | 1 | False | 1,5 |
| BIR | 3 | 111.62 (166.20) | (285,152) | 1 | False | 1,4 |
| DN(BIR) | 4 | MAXVALUE | (209,109) | 4 | False | 3,6 |
| DN(DOJ,NSO) | 5 | MAXVALUE | (390,381) | 5 | False | 2,7 |
| DN(NIA,BIR) | 6 | MAXVALUE | (163,204) | 6 | False | 4,8 |
| DOJ | 7 | MAXVALUE | (295,332) | 7 | False | 5,8 |
| DN(DOJ,NPO) | 8 | MAXVALUE | (256,300) | 8 | False | 6,7,9 |
| NPO | 9 | MAXVALUE | (185,387) | 9 | False | 8 |

3rd Pass:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Node Name | Index | Total Distance | Coordinates | Origin | Visited | Adjacency List |
| BSP | 0 | 0 | (399,133) | 0 | True | 1 |
| DN(BSP,NSO) | 1 | 54.58 | (391,187) | 0 | True | 0,2,3 |
| NSO | 2 | 131.07 | (406,262) | 1 | True | 1,5 |
| BIR | 3 | 166.20 | (285,152) | 1 | False | 1,4 |
| DN(BIR) | 4 | MAXVALUE | (209,109) | 4 | False | 3,6 |
| DN(DOJ,NSO) | 5 | 120.07 (251.14) | (390,381) | 5 | False | 2,7 |
| DN(NIA,BIR) | 6 | MAXVALUE | (163,204) | 6 | False | 4,8 |
| DOJ | 7 | MAXVALUE | (295,332) | 7 | False | 5,8 |
| DN(DOJ,NPO) | 8 | MAXVALUE | (256,300) | 8 | False | 6,7,9 |
| NPO | 9 | MAXVALUE | (185,387) | 9 | False | 8 |

4th Pass:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Node Name | Index | Total Distance | Coordinates | Origin | Visited | Adjacency List |
| BSP | 0 | 0 | (399,133) | 0 | True | 1 |
| DN(BSP,NSO) | 1 | 54.58 | (391,187) | 0 | True | 0,2,3 |
| NSO | 2 | 131.07 | (406,262) | 1 | True | 1,5 |
| BIR | 3 | 166.20 | (285,152) | 1 | False | 1,4 |
| DN(BIR) | 4 | MAXVALUE | (209,109) | 4 | False | 3,6 |
| DN(DOJ,NSO) | 5 | 251.14 | (390,381) | 5 | True | 2,7 |
| DN(NIA,BIR) | 6 | MAXVALUE | (163,204) | 6 | False | 4,8 |
| DOJ | 7 | 106.89 (358.03) | (295,332) | 7 | False | 5,8 |
| DN(DOJ,NPO) | 8 | MAXVALUE | (256,300) | 8 | False | 6,7,9 |
| NPO | 9 | MAXVALUE | (185,387) | 9 | False | 8 |

5th Pass:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Node Name | Index | Total Distance | Coordinates | Origin | Visited | Adjacency List |
| BSP | 0 | 0 | (399,133) | 0 | True | 1 |
| DN(BSP,NSO) | 1 | 54.58 | (391,187) | 0 | True | 0,2,3 |
| NSO | 2 | 131.07 | (406,262) | 1 | True | 1,5 |
| BIR | 3 | 166.20 | (285,152) | 1 | False | 1,4 |
| DN(BIR) | 4 | MAXVALUE | (209,109) | 4 | False | 3,6 |
| DN(DOJ,NSO) | 5 | 251.14 | (390,381) | 5 | True | 2,7 |
| DN(NIA,BIR) | 6 | MAXVALUE | (163,204) | 6 | False | 4,8 |
| DOJ | 7 | 358.03 | (295,332) | 7 | True | 5,8 |
| DN(DOJ,NPO) | 8 | 50.45 (408.47) | (256,300) | 8 | False | 6,7,9 |
| NPO | 9 | MAXVALUE | (185,387) | 9 | False | 8 |

6th Pass:

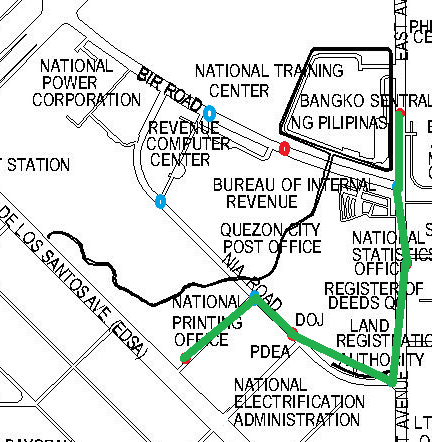
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Node Name | Index | Total Distance | Coordinates | Origin | Visited | Adjacency List |
| BSP | 0 | 0 | (399,133) | 0 | True | 1 |
| DN(BSP,NSO) | 1 | 54.58 | (391,187) | 0 | True | 0,2,3 |
| NSO | 2 | 131.07 | (406,262) | 1 | True | 1,5 |
| BIR | 3 | 166.20 | (285,152) | 1 | False | 1,4 |
| DN(BIR) | 4 | MAXVALUE | (209,109) | 4 | False | 3,6 |
| DN(DOJ,NSO) | 5 | 251.14 | (390,381) | 5 | True | 2,7 |
| DN(NIA,BIR) | 6 | 133.66 (542.13) | (163,204) | 6 | False | 4,8 |
| DOJ | 7 | 358.03 | (295,332) | 7 | True | 5,8 |
| DN(DOJ,NPO) | 8 | 408.47 | (256,300) | 8 | True | 6,7,9 |
| NPO | 9 | 112.29 (654.42) | (185,387) | 9 | False | 8 |

7th Pass:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Node Name | Index | Total Distance | Coordinates | Origin | Visited | Adjacency List |
| BSP | 0 | 0 | (399,133) | 0 | True | 1 |
| DN(BSP,NSO) | 1 | 54.58 | (391,187) | 0 | True | 0,2,3 |
| NSO | 2 | 131.07 | (406,262) | 1 | True | 1,5 |
| BIR | 3 | 166.20 | (285,152) | 1 | False | 1,4 |
| DN(BIR) | 4 | MAXVALUE | (209,109) | 4 | False | 3,6 |
| DN(DOJ,NSO) | 5 | 251.14 | (390,381) | 2 | True | 2,7 |
| DN(NIA,BIR) | 6 | 542.13 | (163,204) | 8 | False | 4,8 |
| DOJ | 7 | 358.03 | (295,332) | 5 | True | 5,8 |
| DN(DOJ,NPO) | 8 | 408.47 | (256,300) | 7 | True | 6,7,9 |
| NPO | 9 | 654.42 | (185,387) | 8 | True | 8 |

Total Distance from BSP to NPO = 654.42 units

Shortest Route from BSP to NPO:



**How the Algorithm is used in the Development of the Solution of the Problem Presented**

Since Dijkstra’s Algorithm is a greedy algorithm which finds the shortest path for the given graph by comparing the weights of the adjacent edges and vertices from one source to one destination, lt is the most suitable aid for the solution of the problem. By using this algorithm, time and space consumption is minimized. With this, the program is optimized.

**Bibliography**

<http://publish.uwo.ca/~jmalczew/gida_1/Zhan/Zhan.htm>

<http://i11www.iti.uni-karlsruhe.de/extra/publications/sww-daole-00.pdf>

<http://askville.amazon.com/Fastest-Shortest-Path-Algorithms-Real-Road-Networks-Data/AnswerViewer.do?requestId=35546416>