Dijkstra's

Fredrik Berzins

Fall 2023

Introduction

Dijkstra's algorithm was made by Edsger Wybe Dijkstra in the 1950s, it's a used to find the shortest path from one node to another in a weighted graph.

Implementation & Explanation

Graph

A graph is used this time to represent a travel map of Europe so the nodes are cities/towns and the edges are train tracks or ferries and the weight of an edge is the time for traverse the connection in minutes. The graph is built with 151 cities and 201 connections.

No Duplicates

No duplicates keeps track of the route up until that point this this is done to avoid lopping back to a city already on the route. But it is not a sophisticated method that will find the shortest path from the beginning it will have to search thru a lot of paths to find the shortest.

No Duplicates /w Swap Max

No Duplicates /w Swap Max keeps track of the route as No Duplicates but when it finds a route to the destination it sets the travel time to the max allowed time for a solution this is done to avoid searching for valid routes that are longer since those are not whats being searched for.

Dijkstra's

Dijkstra's algorithm is a algorithm that searches for the shortest path thru a graph. This is done by adding searched node to a array and neighboring nodes to a prioritized queue (prioritized by distance). The next node to be

searched and added to the array is the first node in the queue, it's neighboring are then added to the queue. This is done until a neighbour is matched with the destination city.

Benchmarks

The benchmarks was done as one single search for a routes, the routes tested was the following:

Start	Destination	Abbreviation
Malmö	**Göteborg**	(Mal/Göt)
Göteborg	**Stockholm**	(Göt/Sto)
Umeå	**Göteborg**	(Ume/Göt)
Göteborg	**Umeå**	(Göt/Ume)
Berlin	**Berlin**	(Ber/Ber)
Berlin	**Prag**	(Ber/Pra)
Berlin	**Gdansk**	(Ber/Gda)
Berlin	**Paris**	(Ber/Par)
Berlin	**London**	(Ber/Lon)
Berlin	**Stockholm**	(Ber/Sto)
Berlin	**Manchester**	(Ber/Man)
Berlin	**Rom**	(Ber/Rom)
Berlin	**Valencia**	(Ber/Val)
Berlin	**Bari**	(Ber/Bar)
Berlin	**Seinäjoki**	(Ber/Sei)
Berlin	**Bukarest**	(Ber/Buk)

Table 1: Benchmark routes with respective abbreviation.

Result

The results shown below in table 2 shows that the executing time compared two of the previously used search function. It's quite clear that Dijkstra's algorithm is significantly faster the the fastest previous method. this is since the previous methods have to search thru multiple valid solutions to find the shortest while Dijkstra's will always find the shortest first.

Route	No Dup	No Dup /w Swap Max	Dijkstra's
M/G	129000	266	5,4
G/St	49200	52	9,2
U/G	92000	34	8,8
G/U	132000	4480	10,2

Table 2: Time to in us to find shortest route with different methods (These times are based on a graph with 52 cities and 75 connections)

As seen below in table 3 the results show how Dijkstra's algorithm is execution time not directly related to searched elements but rather related to edges and nodes since every edge gives more possibilities for a shorter path.

Route	Execution time	Checked cities	Distance
Mal/Göt	5,7	19	153
Göt/Sto	18,2	37	211
Ume/Göt	11,9	45	705
Göt/Ume	16,1	65	705
Ber/Ber	4,4	1	0
Ber/Pra	6,3	10	259
Ber/Gda	8,3	29	454
Ber/Par	8,7	35	477
Ber/Lon	10,9	37	497
Ber/Sto	14,3	72	697
Ber/Man	17,2	80	728
Ber/Rom	21,9	110	1039
Ber/Val	23,9	113	1063
Ber/Bar	19,2	119	1246
Ber/Sei	23,9	130	1557
Ber/Buk	24,4	131	1557

Table 3: Time to find shortest path in us, Amount of checked cities in graph and shortest time for a certain route in minutes.

Conclusion

Dijkstra's algorithm is a quick and efficient algorithm but as most quick algorithms it somewhat complex to code. Compared it also uses more memory then the two other methods since you have both a array with the searched nodes and a queue with nodes to check.