ijecs open access www.ijecs.in
International Journal Of Engineering And Computer Science ISSN:2319-7242
Volume 2 Issue 3 March 2013 Page No. 569-572

Increasing no. of nodes for Dijkstra algorithm without degrading the performance

Nikita Jaiswal , Rajesh Kumar Chakrawarti

COMPUTER SCIENCE AND ENGGINEARING

Shri vaishnav institute of technology and science Indore
njaiswl90@gmail.com, rajesh_kr_chakra@yahoo.com

Abstract:- *In most of the shortest path problems like vehicle routing problems and network routing problems, we only need an efficient path between two points—source and destination, and it is not necessary to calculate the shortest path from source to all other nodes. Dijkstra algorithm is called " Single Source Shortest Path ".This paper introduces the Dijkstra algorithm in detail, and illustrates the method of implementation of the algorithm and the disadvantages of the algorithm .this algorithm applied on Directed weighted graph to find shortest path between two nodes ,but all weights in the graph should be non negative. In this paper we also discuss about how we can improve this algorithm in terms of finding path according to cost by increasing some no. of nodes.*

I. INTRODUCTION

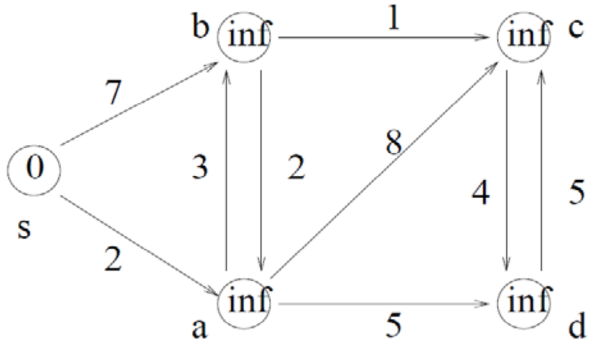
With the popularity of the computer and the development of the geographic information science, GIS has been increasingly extensive and in-depth applications for its powerful functions. As one of the most important functions, network analysis has played an important role in lots of fields, such as electric navigation, traffic tourism, urban planning and electricity, communications, and other various pipe network designs and soon. The key problem about network analysis is his shortest path analysis. The shortest path analysis not only refers to the shortest distance in general geographic sense, but also extends to other measurements, such as time, cost, and the capacity of the line.

How do we find the shortest path between two vertices on a weighted graph? To solve this kind of problem Dijkastra algorithm is introduced. Dijkstra's Algorithm was created in 1959 by Dutch computer scientist Edsger Dijkstra. While employed at the Mathematical Centre in Amsterdam, Dijkstra was asked to demonstrate the powers of ARMAC, a sophisticated computer system developed by the Mathematical Centre. Part of his presentation involved illustrating the best way to travel between two points and in doing so, the shortest path algorithm was created. It was later renamed Dijkstra's Algorithm in recognition of its creator. Dijkstra's algorithm is a graph search algorithm that solves the single-source shortest path problem for a graph with nonnegative edge path costs, producing a shortest path. Because of weighted graph we can't used BFS, The problem is that vertices on the graph are no longer visited in the same order of closest to the source node. Dijkastra algorithm uses the greedy approach to solve this problem. There are some modifications done in BFS for Dijkastra algorithm. BFS's queues are replaced by priority queue. Vertices are added to the Priority Queue by their distance away from the source. The algorithm Dijkstra is the theoretical foundation for solving the problem about the shortest path

II. HOW DIJKASTRA WORKS

To know that how Dijkstra works we just take an example:

Remark: The predecessor pointer pred[] is for determining the shortest paths.

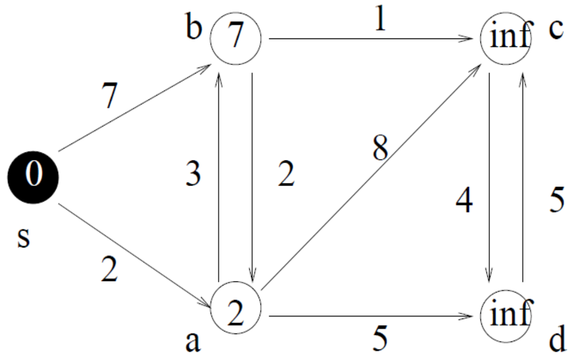


Step 1 :- Initialization

v	s	a	b	c	d
$d[v]$	0	∞	∞	∞	∞
$pred[v]$	nil	nil	nil	nil	nil
$color[v]$	W	W	W	W	W

Priority queue

v	s	a	b	c	d
$d[v]$	0	∞	∞	∞	∞

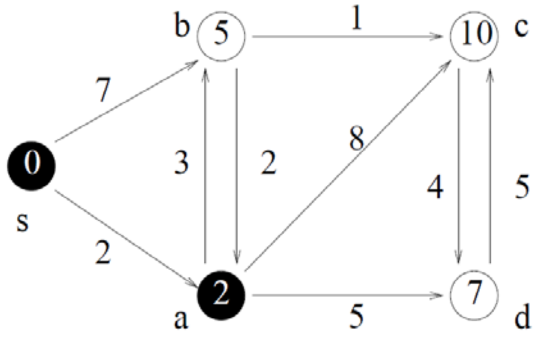


Step 2 :- As Adj(s) = {a ,b}, work on a and b and update information

v	s	a	b	c	d
$d[v]$	0	2	7	∞	∞
$pred[v]$	nil	s	s	nil	nil
$color[v]$	B	W	W	W	W

Priority queue :-

v	a	b	c	d
$d[v]$	2	7	∞	∞

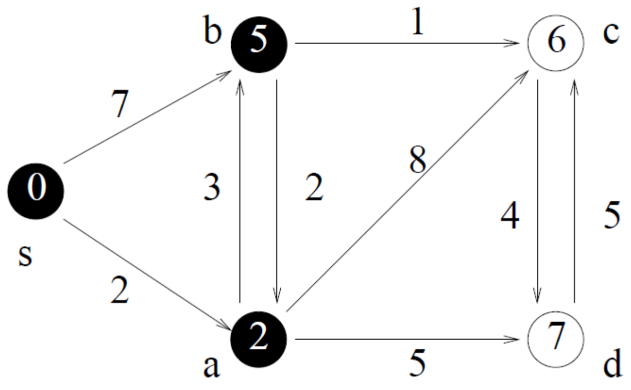


Step 3 :- After Step 1, a has the minimum key in the priority queue. As Adj[a] = {b,c,d}, work on b,c,d and update information

v	s	a	b	c	d
$d[v]$	0	2	5	10	7
$pred[v]$	nil	s	a	a	a
$color[v]$	B	B	W	W	W

Priority queue :-

v	b	c	d
$d[v]$	5	10	7

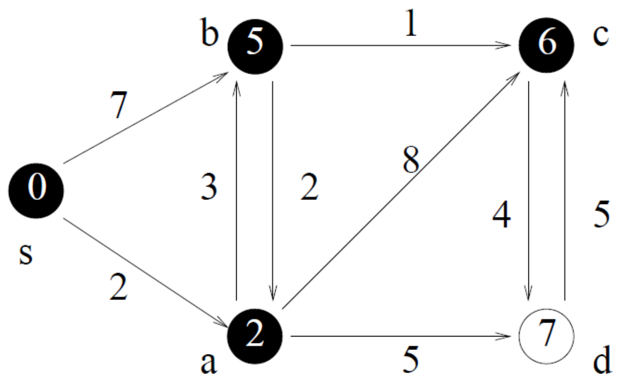


Step 4 :- After Step 3,b (has the minimum key in the priority queue. As Adj[b]={a ,c}, work on a' ,c) and update information.

v	s	a	b	c	d
$d[v]$	0	2	5	6	7
$pred[v]$	nil	s	a	b	a
$color[v]$	B	B	B	W	W

Priority queue :-

v	c	d
$d[v]$	6	7

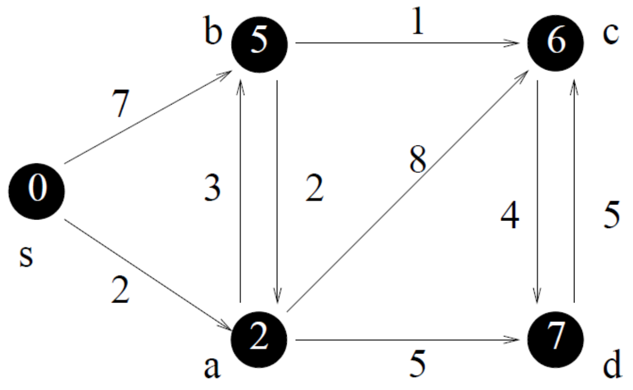


Step 5 :- After Step 4,c) has the minimum key in the priority queue. As Adj[c]= {d} , work on d and update information.

v	s	a	b	c	d
$d[v]$	0	2	5	6	7
$pred[v]$	nil	s	a	b	a
$color[v]$	B	B	B	B	W

Priority queue :-

v	d
$d[v]$	7



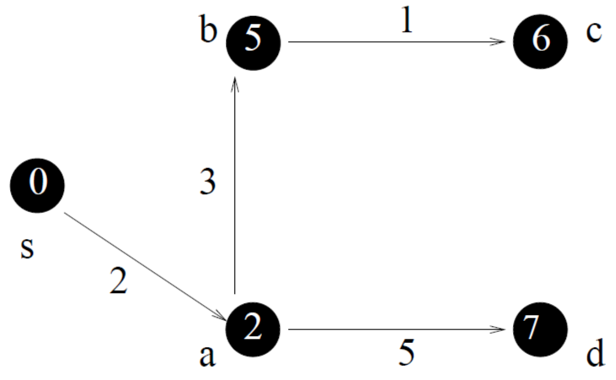
Step 6 :- After Step 5,d * has the minimum key in the priority queue. As Adj[d]={c} , work on c) and update information.

v	s	a	b	c	d
$d[v]$	0	2	5	6	7
$pred[v]$	nil	s	a	b	a
$color[v]$	B	B	B	B	B

Priority queue :-

$Q = \emptyset.$

Shortest Path Tree:



The above example shows the working of Dijkastra algorithm. In this algorithm we also create correlation matrix, adjacent matrix and distance matrix.

III. DISADVANTAGES OF DIJKSTRA ALGORITHM

The major disadvantage of the algorithm is the fact that it does a blind search there by consuming a lot of time waste of necessary resources.

Another disadvantage is that it cannot handle negative edges. This leads to acyclic graphs and most often cannot obtain the right shortest path.

The Bellman–Ford algorithm computes single-source shortest paths in a weighted digraph. It uses the same concept as that of Dijkstra's algorithm but can handle negative edges as well. It has a better running time than that of Dijkstra's algorithm.

In the algorithm and computer program, correlation matrix, adjacent matrix and distance matrix are used to compute the shortest path based on network matrix of Dijkstra. Many N*N arrays are defined to store graphical date and compute. N is referred to the number of the network nodes. When the number of the nodes is very large, it occupies a lot of CPU memory. For example, when the number of the nodes is 3000 , it needs 4*3000*3000 = 36000000 bytes—36 MB memory, and if the number is 6000, it needs 144 MB memory. If we do not improve the Dijkstra algorithm, the algorithm is very difficult to apply in network analysis for huge data.

IV. NEED FOR INCREASING NODE

As we know that Dijkstra algorithm gives shortest path between 2 nodes. if we want to apply this algorithm on map for going one city to another ,and want to find out the shortest path between those 2 city's, then Dijkstra will give the shortest path according to the distance. But some times the cost or we can say the money also affect the person. Because its not necessary that buses or trains also followed that path which is given by Dijkstra algorithm. This situation may be occurred. To solve such kind of problem we just add a priority queue and a cost wise graph in this algorithm. We gives a choice for the person that “in which type of path u want distance wise or cost wise. If person choose distance wise then Dijkstra algorithm will applied on the graph which contain distance as a weight. If person choose cost wise then Dijkstra algorithm will applied on the graph which contain cost as a weight. Because of this it’s necessary to increase a node which denotes the choice .also one addition graph is needed in which cost (cost shows the money needed to spend for travel between 2 nodes) is used as a weight.

ACKNOWLEDGMENT

The author wish to thank Mr. Rajesh Kumar Chakrawarti, Reader, CSE Department, Shri Vaishnav Institute of Technology and Science, Indore (M.P) for his support and guidance.

REFERENCES

[1] Edward P.F. Chan & Ning Zhang “Finding Shortest Paths in Large Network Systems”.page 7.

[2] Shawn J. Rutter “Dijkstra’s Algorithm Final Project”.2009,page 10.

[3] Merin Puthuparampil “Report Dijkstra’s Algorithm”.page 15

[4] Fuhao ZHANG*, Ageng QIU, Qingyuan LI “Improve On Dijkstra Shortest Path Algorithm For Huge Data”.2009. Page 4.