Analysis of Single Source Shortest Path Problem

Upasana Ghosh

Roll no: UE143110
U.I.E.T, Panjab University
Email: ghoshupasana05@gmail.com

AIM

This paper presents the analysis of problem of finding the length of shortest path from the source vertex to all other vertices in directed connected weighted graph using Dijkstra's algorithm.

INTRODUCTION

The single source shortest path problem is to compute the shortest paths from a source vertex to a destination vertex in a directed weighted connected graph. A path in a graph is a walk along the edges that does not include any vertex twice, except that its first vertex might be the same as its last. To find shortest path between vertices, greedy strategy is used in Dijkstra's algorithm. This algorithm was proposed by computer scientist, Edsger Wybe Dijkstra in 1956 and was published in the year 1959.

A greedy algorithm always makes the choice that look best at the moment. It makes a locally optimal choice in the hope that this choice will lead to a global optimal solution. It does not always lead to optimal solution but for many problems optimal results can be obtained. It first make a greedy choice, the choice that looks best at that moment and then solve the resulting sub-problems without bothering to solve other sub-problems. It generates only one decision sequence unlike dynamic programming which generates multiple decision sequence. Dynamic programming is a method for solving a complex problem by breaking it down into a collection of simpler sub-problems. It finds the optimal solution to all sub-problems and then makes an informed choice as each of those sub-problems are solved just once, and their solutions are stored. Dynamic programming follows principle of optimality. According to principle of optimality, an optimal sequence of decision has the property that whatever the initial state and decision are, the remaining decision must constitute an optimal decision sequence. Hence, this approach is contrary to greedy approach.

This shortest path algorithm is widely used in network routing protocols, most notably IS-IS and Open Shortest Path First (OSPF). A variant of Dijkstra's algorithm, know as uniform-cost search is used in artificial intelligence field as an instance for best-first search. Nowadays, this algorithm is also used in Intelligent Transportation System

APPROACH

The problem of single source shortest path is solved using Dijkstra's algorithm. Let $G=(V,E)$ be a connected graph in which each edge $E(u,v)$ has an associated length $L(u,v)$. Initially, all the vertices are marked unvisited. An array *distance* is used to store the solution of single source shortest path problem. It is first initialized with direct distance from source vertex, say *sv* to other vertices. If there is no path between the

two vertices, then distance is infinity. The source vertex, *sv* is marked *visited.* The vertex with the minimum distance from source, say *minv* is selected and the *distance* array is updated if the initial distance of other vertices is larger than the distance from the source vertex to that vertex via *minv* and the vertex is unvisited else *distance* array remains the same. The vertex *minv* is marked *visited*. Next, the vertex is selected which has the smaller distance value in *distance* array and the process is repeated *n-2* number of time where *n* is the number of vertices in the graph.

EXPERIMENT

In this experiment, vertices and edges are drawn on mouse click and keyboard button click. C programming language is used in this experiment. As the vertices and edges are represented graphically, graphics.h and dos.h header files are used to implement them. C graphics using graphics.h functions can be used to draw different shapes, display text in different fonts, colors and many more. This header file can be used for animations, projects and games. Different sizes of circle, rectangle, ellipse, lines, bars and other geometrical figures can be drawn with this header file. The header file dos.h of C language contains functions for handling interrupts, producing sound, date and time functions etc. initgraph( ) function is used to initialize the graphics system. It loads a graphics driver from disk and then puts the system into graphics mode. It also resets all graphic settings (color, palette, current position, etc.) to their defaults then resets the graph result to 0. In graphics mode all the screen coordinates are mentioned in terms of pixels. Number of pixels in the screen decides resolution of the screen.

In this program, vertices are created graphically in vetex_fill() function. This function contain fillellipse(), setcolor(), outtextxy() functions whose definitions are available in graphics.h library. fillellipse() functions is used to create vertex of the graph. Setcolor() function is used to set color to this vertex. Outtextxy() function is used to print vertex number and vertex name on the graphical screen. Vertices would keeps on creating on mouse click until the user press 1 from keyboard.

Edges are created graphically in edge_fill() function. This function contain a line(), setcolor(), outtextxy()whose definitions are available in graphics.h library. Line() is used to create line in graphical screen according to the given x and y coordinates. Edge_fill() function first draws vertex graphically and then create line joining those vertices.

As the vertices and edges are drawn using mouse and keyboard click, mouse.h header file is created manually using graphic.h and dos.h header files and is added in this program. Mouse.h header file consist of several functions like initMouse(), showMouse(), hideMouse() and Isclick(). All these functions uses AX register to access mouse pointer and keyboard keys. Interrupts are called to use graphics and provide delay in taking inputs. Mouse events are tracked by using software interrupt ‘0x33’. Here ‘0x’ means that the number is in hexadecimal. The interrupt call is made using int86( ) function. This function is defined in dos.h. The union inreg and outreg hold values of registers before and after the call has been made. They are of type union REGS.

To initialize a mouse, the value of AX register is set as ‘0x00’. This tells the system that when the ‘0x33’ interrupt is called, it should initialize the mouse (basically get it ready for other functions). showMouse() function set the AX register value to 1 and display the mouse pointer in graphical screen. hideMouse() function set the AX register value to 2 and is used to hide the mouse pointer from the graphical screen. Isclick() function set the AX

register value to 3 and is used to get the x and y coordinate of the screen where user has clicked with mouse pointer. It also tells the mouse button (left, right or middle) the user has pressed on the graphical screen. kbhit( ) function returns nonzero integer if any key is pressed else it returns zero.

In this program, a 2-dimension array is taken, that is matrix *adj*. GetVertices() is called which stores the values of vertex coordinates and vertex name and draw vertex as soon as user clicks on the graphical screen with the mouse pointer. If user press keyboard key 1 then the program will prompt the user to input edge weight using GetEdges() function and matrix *adj* is updated. Once, all edge costs are inserted by the user, it would redraw the graph using ReDraw(). Then ShortestPath() function is called which computes the resultant array in array *distance* and displays the result on the graphical screen.

Platform used for compilation and execution:

Operating system: Window 10

RAM: 4GB

Compiler: Turbo C++ 4.0

Processor: intel core i3

ALGORITHM

Algor ShortestPath(*adj,v,n*)

{ //adj[1:*n,*1:*n]*is the length of adjacency matrix //of a graph with *n* number of vertices and *v* is //the source vertex

for *i* = 1 to *n* do

{ S[*i*] = unvisited;

distance[i] = *adj[v,i]*

}

s[v] = visited;

Distance[v] = 0;

for *num* = 2 to *n-1* do

{ choose *u* from among those vertices which is *unvisited* and *distance*[*u*] is minimum;

s[u]=visited

for each *w* adjacent to *u* with *s[w]=unvisited* do

if(*distance[w]>distance[u]+adj[u,w]*)

then *distance[w]>distance[u]+adj[u,w]*)

}

RESULT

Figure 1 to Figure 5 represents the formation of vertices and edges. Figure 6 represents the final graph along with weights of edges and Figure 7 represents solution of single source shortest path problem.

Fig 1: 1st vertex created

Fig 2: 4th vertex created

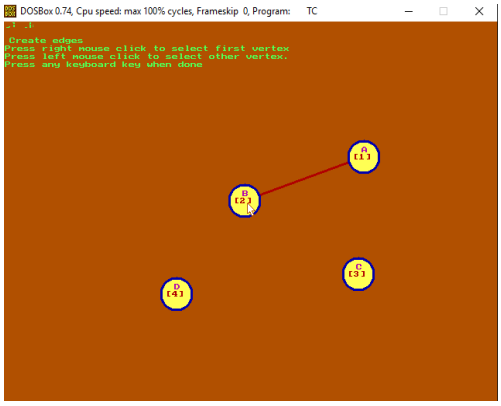

Fig 3: 1st edge created

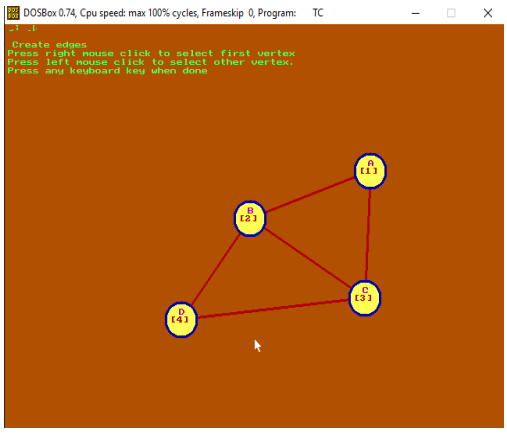

Fig 4: 5th edge created

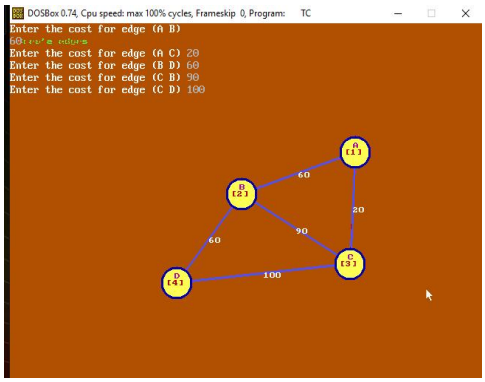

Fig 5: edge weight inserted

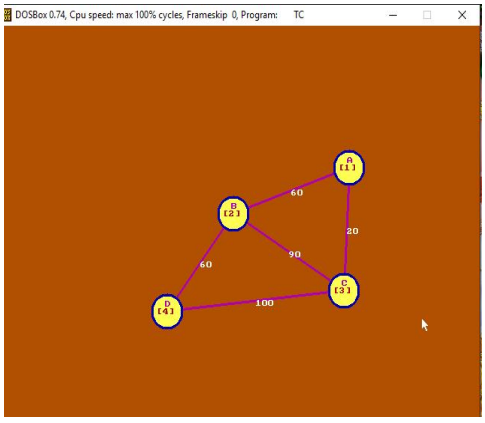

Fig 6: Resultant graph

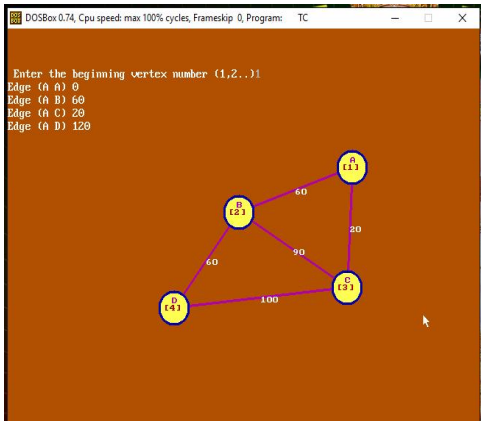

Fig 7: Shortest distance between source vertex and other verticces

DISCUSSION

The single souce shortest path problem is to determine the shortest path from the source vertex to all the remaining vertex of graph in a weighted directed digraph $G = (V,E)$. It is assumed that all the weights are positive. The weight length *adj[i,j]* is set to some large number if there is no edge between vertex *i* and vertex *j*. Time complexity of Dijkstra's algorithm is $O(n^2)$, where *n* is the number of vertices. As adjency matrix is used to store the length between vertices, $O(n^2)$ time is required just to determine the edges in a graph even if the graph is sparse. If number of vertices is very large and the graph is sparse then solving the problem using array matrix could cause a serious waste of resources and lots of useless cycles. Instead of using 2-dimension array matrix, linked list can be used. Time complexity using linked list is O(n E) where n is the number of vertices and E is the number of edges in a graph. In a sparse graph, as number of edges are lesser than number of vertices, using linked list can yield better perfomance.

Although, the most efficient algorithm used to find the shortest path between two known vertices is Dijkstra's algorithm, but it has low search efficiency due to its large computations. Some improvements on Dijkstra's algorithm are already done in terms of efficient implementation and cost matrix. Sneyers J., Schrijvers, T. and Demoen B. in "Dijkstra's Algorithm with Fibonacci Heaps" introduces an efficient implementation of Dijkstra's algorithm using Fibonacci Heaps. Brendan, H., Gordon, J. in "Generalizing Dijkstra's Algorithm and Gaussian Elimination for Solving MDPs" introduces an efficient implementation of cost matrix. Yizhen Huang, Qingming Yi and Min Shi in "An Improved Dijkstra Shortest Path Algorithm" introduces a constraint function with weighted value ω and ignores a large number of irrelevant nodes during searching the shortest path. It can find the optimal solution and greatly improve search efficiency. Seifedine Kadry, Ayman Abdallah, Chibli Joumaa in "On The Optimization of Dijkstra's Algorithm" introduces some improvement in order to reduce the number of iterations and to find easily and quickly the shortest path.

CONCLUSION

Dijkstra's algorithm provides a simple yet effective solution of single source shortest path problem but the time complexity is still $O(n^2)$ which increases the computations when the number of vertices are large. But by implementing some improved Dijkstra's algorithm, the time complexity can be reduced considerably.

REFERENCES

[1] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stin *Introduction to Algorithm*, 3rd ed. The MIT Press, Cambridge, Massachusetts, London, England

[2] Brendan, H., and Gordon, J., *Generalizing Dijkstra's Algorithm and Gaussian Elimination forSolving MDPs*. Conference: ICAPS(AIPS), 2005.

[3] Sneyers J., Schrijvers, T. and Demoen B., *Dijkstra's Algorithm with Fibonacci Heaps.* Workshop (WLP'06).

[4]Yizhen Huang, Qingming Yi and Min Shi, "*An Improved Dijkstra Shortest Path Algorithm".* Conference: ICCSEE 2013

[5] Seifedine Kadry, Ayman Abdallah and Chibli Joumaa, *On The Optimization of Dijkstra's Algorithm*, Guangdong Province Engineering Research Center project, 2012

[6] Dijkstra's Algorithm in *Wikipedia*. Retrieved April 15, 2016 from https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm