# Analysis of All Pairs 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 shortest path between every pair of vertices in directed connected weighted graph using Floyd-Warshall algorithm.

## INTRODUCTION

The all pairs shortest path problem is to compute the shortest paths of vertices of 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 all pairs, the strategy of dynamic programming is used in Floyd-Warshall algorithm. The Floyd–Warshall algorithm was published in 1962. However, it is essentially the same as algorithms previously published by Bernard Roy in 1959 and also byStephen Warshall in 1962 for finding the transitive closure of a graph and is closely related to Kleene's algorithm (published in 1956) for converting a deterministic finite automaton into a regular expression. The modern formulation of the algorithm as three nested for-loops was first described by Peter Ingerman, also in 1962.

Dynamic programming approach is used to solve all pair shortest path problem. 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 using a memory-based data structure. Many decision sequences are generated through this way. But next time, if the same sub-problem occurs, instead of re-computing its solution, system simply looks up the previously computed solution. This saves computation time at the expense of a modest expenditure in storage space. 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. 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. Dynamic approach always leads to an optimal result.

## APPROACH

The problem of all pair shortest path is solved using Floyd-Warshall algorithm. Let $G=(V,E)$ be a connected graph in which each edge $E(u,v)$ has an associated cost $C(u,v)$. A two-dimension matrix $A$ is used to store the solution of all pair shortest path problem. It is first initialized with input graph matrix, $C$. Each vertices of the graph is selected one by one and the matrix $A$ is updated by selecting the shortest path between

the selected vertex to all the other vertex in the graph. It updates the matrix *A* only if there exist some other shortest path from one vertex say, *i* to another vertex say, *j* via some intermediate vertex say, *k*.

Three *for* loops are used to achieve this. The first loop say, *L1* begins from 1 and execute *n* number of time where *n* is the number of vertices in a graph. Second loop say, *L2* begins from 1 and execute *n* number of time and the third loop say, *L3* also begins from 1 and execute *n* number of time For every pair *(i, j)* of source and destination vertices respectively, either, there does not exist some intermediate vertex say, *k* in shortest path from *i* to *j*. In this case, the value of *dist[i][j]* is not updated. There may exist some exist some intermediate vertex say, *k* in shortest path from *i* to *j*. In this case, The value of *dist[i][j]* is updated as *dist[i][k]* + *dist[k][j]*.

## 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, two 2-dimension array is taken, that is matrix *adj* and matrix *A*. 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 AllPath() function is called which computes the resultant matrix in matrix *A* 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 AllPath(cost,A,n)

{ //cost[1:*n*,1:*n*]is the cost of adjacency matrix //of a graph with n vertex

for *i* = 1 to *n* do

for *j*=1 to *n* do

*A[i][j]* ← *adj[i][j];*

for *k*= 1 to *n* do

for *i* = 1 to *n* do

for *j*=1 to *n* do

*A[i][j]=min(A[i][j],A[i][k]+A[k][j]);*

}

## RESULT

Figure 1 to Figure 7 represents the formation of vertices and edges. Figure 8 represents the final graph along with weights of edges and Figure 9 represents solution of all pair shortest path problem.


Fig 1: 1st vertex created

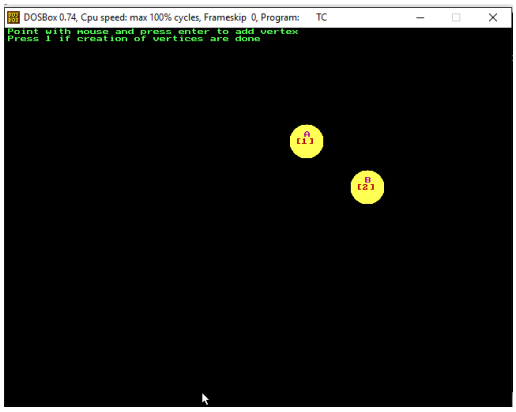


Fig 2: 2nd vertex created


Fig 3: 3rd vertex created

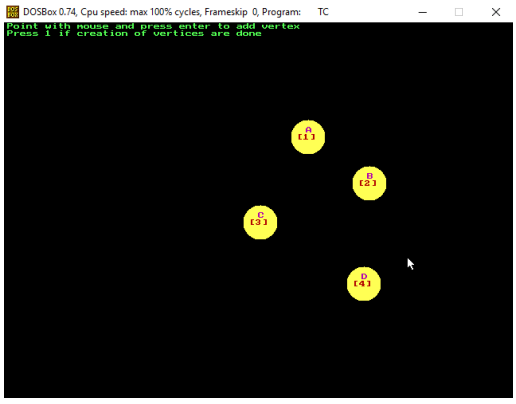


Fig 4: 4th vertex created

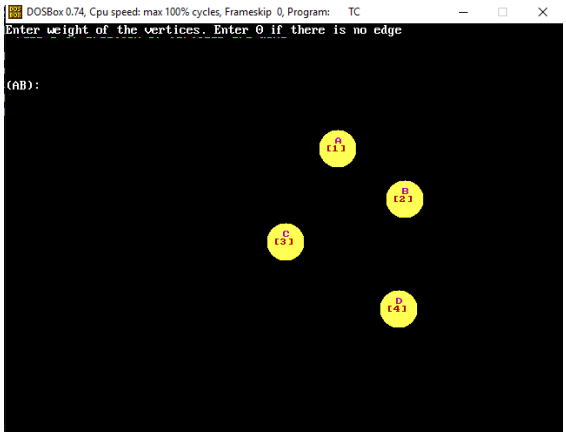


Fig 5: 1st weight is being inserted

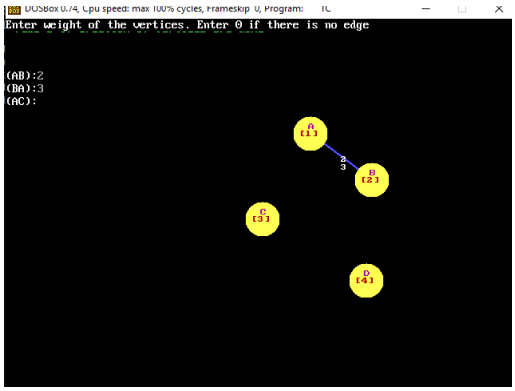


Fig 6: After insertion of first weight.

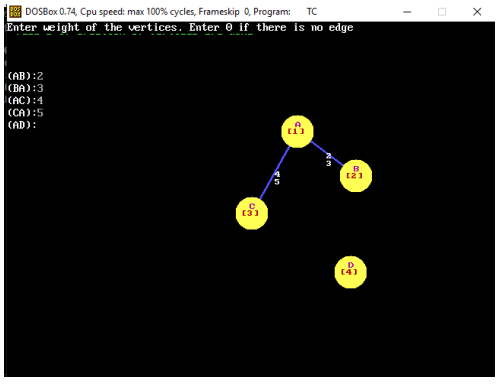


Fig 7: After insertion of second weight

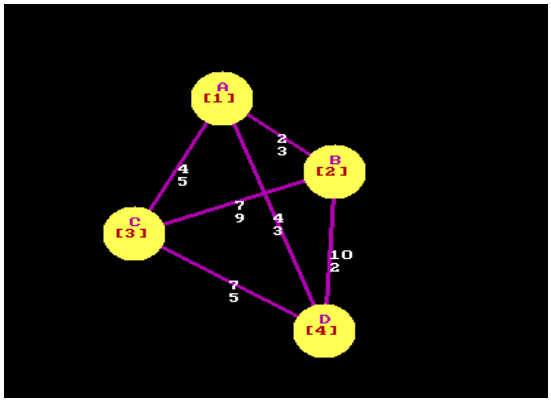


Fig 8: Graph is redrawn after user inserted weights of edges

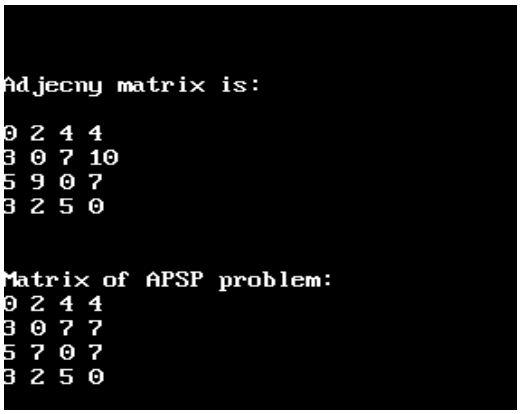


Fig 9: Solution of all pair shortest path problem

## DISCUSSION

The all pair shortest path problem is to determine a matrix $A$ such that $A(i,j)$ is the length of a shortest-path from $i$ to $j$, where $i$ and $j$ represent the vertiex of a directed connected graph under cosideration. in this paper, the solution to this problem is solved using dynamic programming . The time complexity of this program is $O(n^3)$ as it is iterated $n^3$ times which means this is highly inefficient when there are large number of vertices. But it can also be solved using greedy approach. Dijkstra's algorithm can also be used here to solve this problem. Dijkstra's algorithm detemines the length of shortest path from source vertex to all other vertex in a weighted connected graph. Hence, all pair shortest path problem can be solved using dijkstra's algorithm for each vertex $n$ number of times, where n is the number of vertex in a graph. Since, each application of this procedure require $O(n^2)$ time, the matrix $A$ can be obtained in $O(n^3)$ times. As the time complexity remains the same, there is no advantage of dynamic programming over greedy approach. But still the former require weaker restriction on edge costs than the later. The solution presented in this paper can provide appropriate solution even if the weight of edges are negative. The only restriction is graph should not consist of cycle with negaive length. If graph contain negative length cycle then the shortest path between any two vertices on this cycle has $-\infty$ as length which is incorrect.

## CONCLUSION

Floyd-Warshall algorithm provides a simple yet effective solution of all pair shortest path problem but the time complexity is still $O(n^3)$ which increases the computations when the number of vertices are large. The solution of the problem can also be obtained from greedy approach with the time complexity of $O(n^3)$. But as Floyd-Warshall algorithm requires weaker restrictions, hence it is concluded that Floyd-Warshall algorithm yields the best result with lesser number of vertices.

## 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] Mouse Programming in Turbo C in *electrosofts*. Retrieved April 3, 2016 from http://electrosofts.com/cgraphics/mouse.html

[3] Floyd Warshall Algorithm in *Wikipedia*. Retrieved April 3, 2016 from https://en.wikipedia.org/wiki/Floyd%E2%80%93Warshall_algorithm

[4] Shortest path Algorithm in *Wikipedia*. Retrieved April 3, 2016 from https://en.wikipedia.org/wiki/Shortest_path_problem