DSA ASSIGNMENT (TASK 3)

Name : **Avinash R Changrani**

Roll_no. : **CS20B1044**

Date of Submission : **25-06-21**

Problem Title : **To find the shortest/fastest path from one place to another.**

The main Algorithm I will use is the modified Breadth First Search Algorithm to solve this problem. It will have two functions. The first function's algorithm takes input such as graph(Adjacency list), source vertex index, destination vertex index, an array to store the predecessors vertex index and an array to store the distance from the node and number of vertices(nodes). This algorithm outputs are true or false values. If the source and destination are not connected (incomplete graphs) it returns false. And if the source and destination are connected it returns true. And there is another function to print the shortest path whose algorithm takes inputs such as graph(Adjacency list), source vertex index, destination vertex index and number of vertices(nodes). This function's algorithm uses the predecessor array from the first to print out the shortest path which is the shortest path between source and destination (if a path exists). These 2 functions also use other functions such as functions to enqueue, dequeue, reversing a queue, creating a queue etc.

Algorithm for the first Function

First, We create a queue and an array to store values(0 or 1, true or false) that show whether we've visited the node or not. And then we assign all the visited values to false or 0, predecessor values to -1 and store distance to a high value(infinity).

Now, we assign the visited value of source to be true or 1 and distance of source from source is 0 so 0. And then we add source to the queue.

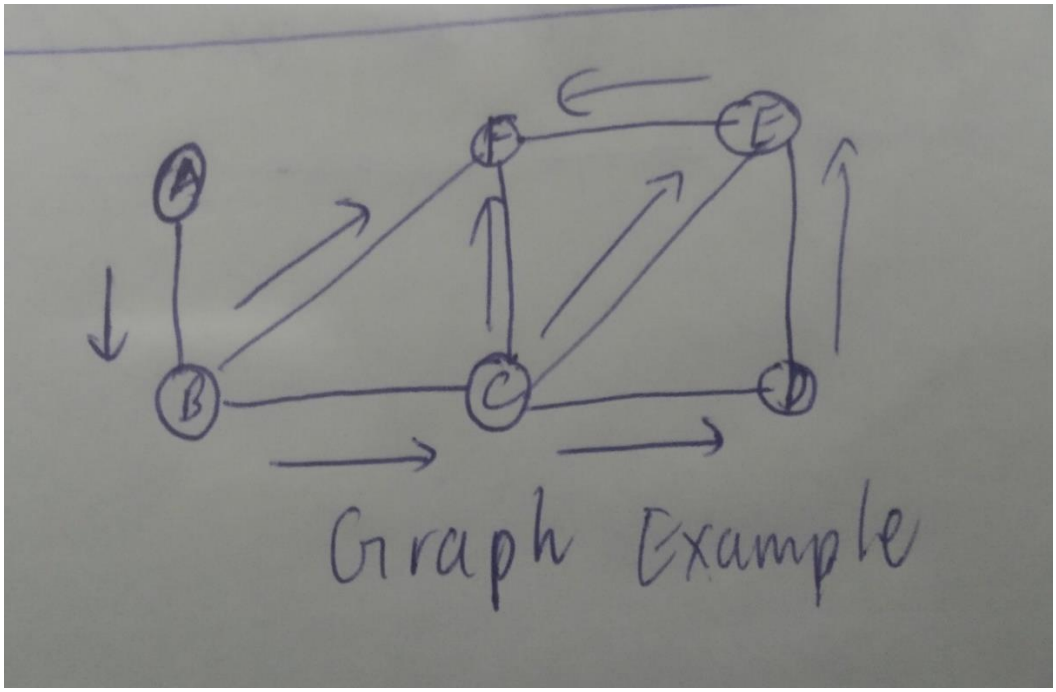
Then run a while loop until queue is empty and inside the while loop we assign the integer (say u) which stores the front end of the queue and then dequeue it.

Now using Graph(adjacency list) we check for vertices adjacent to u and if it's not visited(visited value false or 0), we assign it to true, update the distance from the node u, store the predecessor of the given node as u and add the given node to the queue (enqueue) and then a if statement to return true if the destination is the given node and outside the loop there's a return false statement which returns false if the path between source and destination doesn't exist.

Now, for the 2nd function's purpose id to print the shortest path. And it doesn't print anything if the first function returns false as there is no path(as shortest path won't exist if there isn't even a path between source and destination). Now, we create a queue which stores the path. We create a node and store destination And we enqueue the destination into the queue and then use a while loop to enqueue the predecessor of the given node and then store the predecessor of the given node as node and the loop goes on until there's no predecessor (source). And then we reverse the path (as it's form destination to source) and then print out the shortest path.

Dry Run :

The Example I took in task 2 was :



Now we take the vertices as A – 0, B – 1, C - 2, D - 3, E - 4, F - 5 and the total number of vertices are 6. Now, the source from where we want to go is A(0) and the destination to where we want to go is F(5). The adjacency list for the above graph can be given as

For A : B

For B : A,C,F

For C : B,F,D,E

For D :C,E

For E : C,D,F

For F : B,C,E

We use a distance table to represent our dry run. Initially(before the while loop) our distance table can be given as :

Vertex	Distance	Predecessor
A	0	-
B	Infinity	-
C	Infinity	-
D	Infinity	-
E	Infinity	-
F	Infinity	-

Queue : A(0)

After 1st Iteration :

Vertex	Distance	Predecessor
A	0	-
B	1	A
C	Infinity	-
D	Infinity	-
E	Infinity	-
F	Infinity	-

Queue : B(1)

After 2nd Iteration :

Vertex	Distance	Predecessor
A	0	-
B	1	A
C	2	B
D	Infinity	-
E	Infinity	-
F	2	B

Queue : C(2), F(2)

Here the loop breaks and returns true because it has found the destination node. And then 2nd function using this information stores the path from the destination to source and then prints it from source to destination.

And the 2nd Function prints out the path A-B-F

Complexity Calculation :

For 1st Function :

The time complexity of the given modified BFS algorithm can be given as O(|V| + |E|)(form) as the assigning the distance, visited and predecessor take V time. And the while loop takes |V| + |E| time (each vertex is examined and each vertex is examined once). Therefore the total complexity is of the form O(|V| + |E|). Where V is number of vertices and E is number of edges.

The Best Case :

The best case is that the source and destination are adjacent to each other which takes O(1) time.

Worst Case :

The source and destination aren't connected and the while loop keeps running for |V| + |E| time and returns false or the destination isn't found until all vertices and nearly all edges are searched.

Mathematical Formulation can be given as T(n) = T(v) + T(e)

For 2nd Function :

The Time complexity is O(V) as the shortest path may take all the vertices in the process so it runs for v time.

Best Case :

We find the destination vertex is adjacent to source So it takes O(1) time.

Worst Case :

We find that the source vertex takes all vertices on the path to reach the destination vertex. It take V time.

Space Complexity for the first and second algorithm is the same as at the worst case the queue might store all Vertices. Therefore Space complexity is O(V).