Page :
Date : / /

ASSIGNMENT 8

Aim: Represent a given graph using adjacency matrix / list and find the shortest path using Dijkstra's algorithm. (single source all destination)

Objective: Understand application of Dijkstra's application

Theory:

Definition of Dijkstra's Shortest Path:

(i) To find the shortest path between points, the weight or length of a path is calculated as the sum of the weights of the edges in the path.

(ii) A path is a shortest if there is no path from x to y with lower weight.

(iii) Dijkstra's algorithm finds the shortest path from x to y in order of increasing distance from x. That is, it chooses the first minimum edge, stores this value and adds the next minimum value from the next edge it selects.

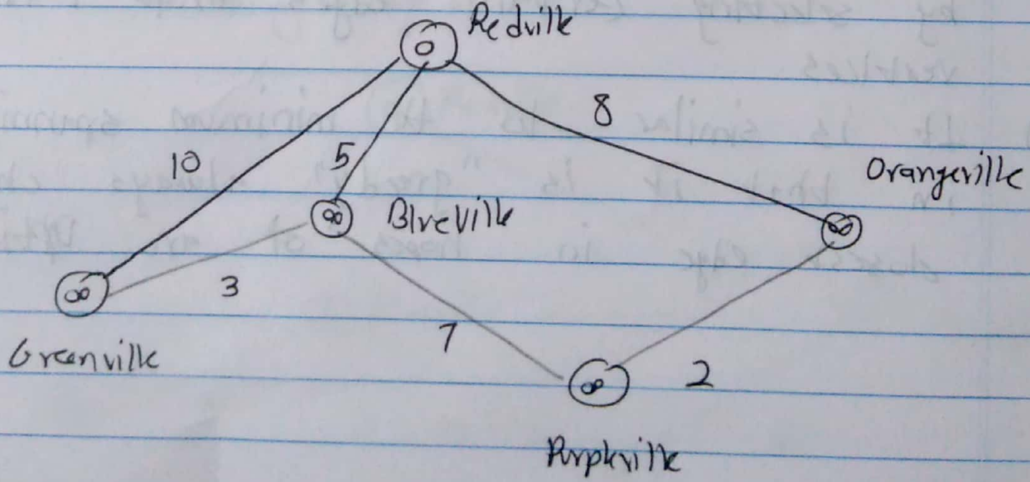
(iv) It starts out at one vertex and branches out by selecting certain edges that lead to new vertices.

(v) It is similar to the minimum spanning tree algorithm, in that it is "greedy" always choosing the closest edge in hopes of an optimal solution.

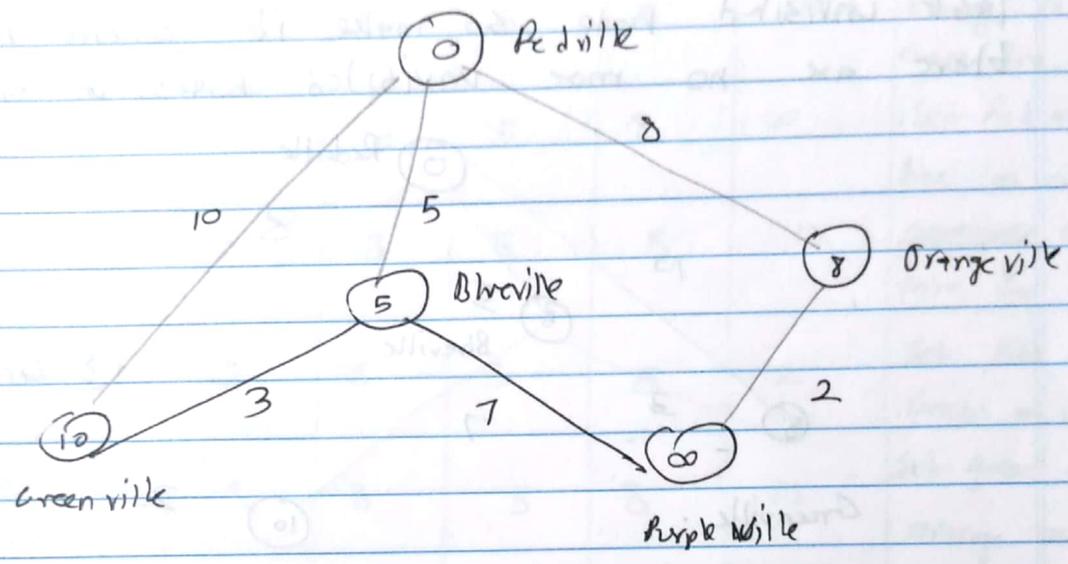
Example: Starting from Redville, find the shortest way to get to surranding towns.

- Begin with source note (city) and call this the current node. Set its value to 0. Set the value of all other nodes to infinity. Mark all nodes as unvisited.
- For each unvisited node that is adjacent to the current note do the following. If value of current note plus the value of edge is less than the value of adjacent node, change the value of the adjacent note to this value. Otherwise leave the value as is.
- Set the current node to visited. If there are still some unvisited nodes, set the unvisited note with the smallest value as the new current node, and go to step 2. If there are no unvisited nodes, then we are done.

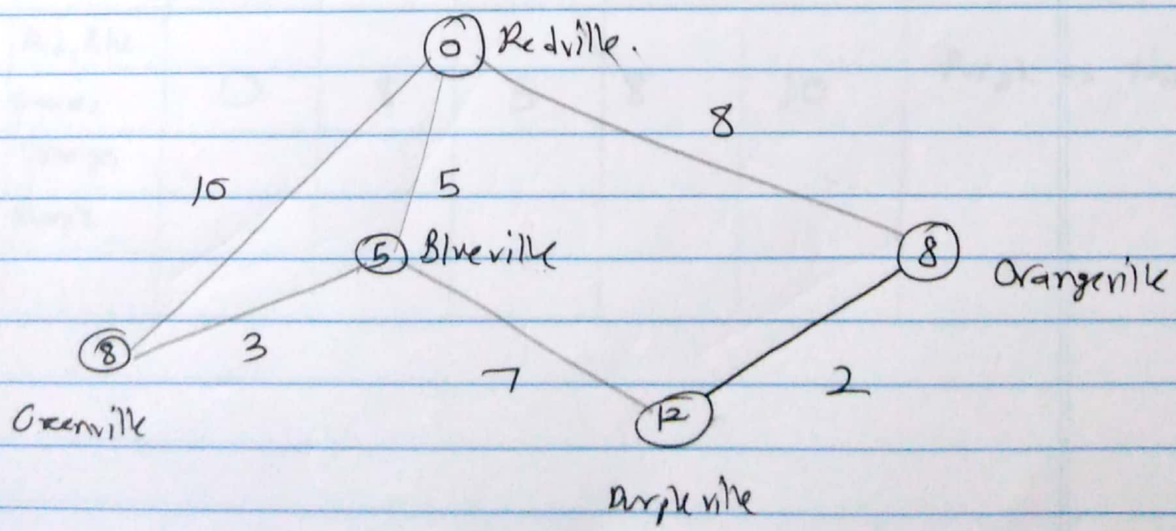
Step 1: Current note is Redville. We give it value 0. Assign all others as infity



Step 2: Next, we look at the unvisited cities our current node is adjacent to. We check if the value of connecting edge, plus value of our current node is less than the value of the adjacent node, and if so we change the value.

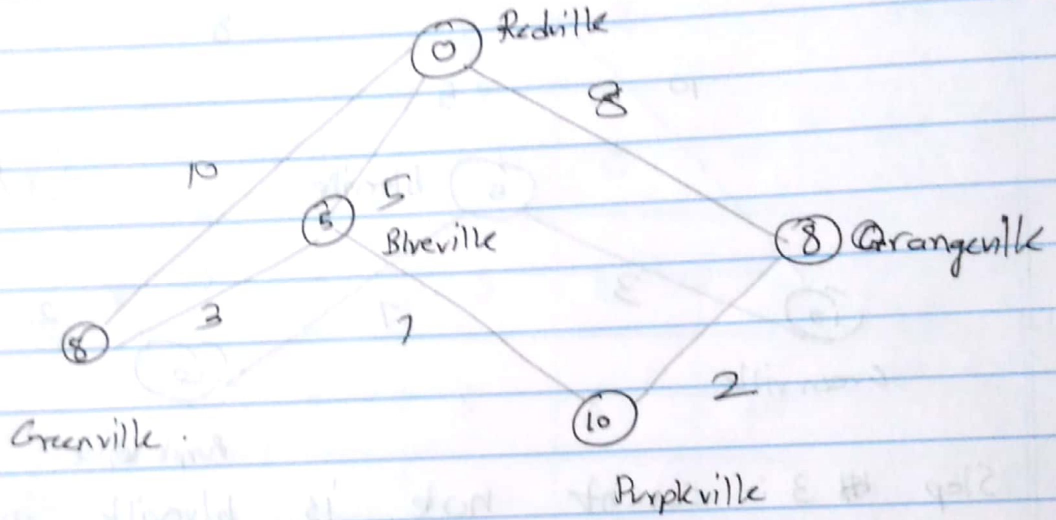


Step 3: current node is blueville and purpleville and greenville are adjacent to it. So we want to see if the value of either of those cities is less than the value of ~~the~~ Blueville plus the value of the connecting edge.



Step IV:

There is only one unvisited node adjacent to Orangeville. If we check the values, Orangeville plus connecting road is 8+2=10. Purpleville's value is 12, and so we change Purpleville's value to 10. Purpleville is our last unvisited node, so make it current node. Since there are no more unvisited nodes, we are done.



Page:
Date: / /

All above steps can be tabulated as follows:

Current	Visited	Red	Green	Blue	Orange	Purple	Description
Red	–	0	∞	∞	∞	∞	Initialize Red as current, set initial values
Red	–	0	10	5	8	∞	Change values for Green, Blue, Orange
Blue	Red	0	10	5	8	∞	Set Red as visited, Blue as current
Blue	Red	0	8	5	8	12	~~Set blue~~ Change value for green, purple
Green	Red, Blue	0	8	5	8	12	Set Blue as visited, Green as current
Orange	Red, Blue, Green	0	8	5	8	12	Set green as visited, orange as current
Orange	Red, Blue, Green	0	8	5	8	10	Change value for purple
Purple	Red, Blue, Green, Orange	0	8	5	8	10	Set orange as visited, Purple as current
–	Red, Blue, Green, Orange, Purple	0	8	5	8	10	Purple as visited

Page :
Date : / /

Algorithm : College Area represented by graph
A graph G with N nodes is maintained by its adjacency matrix cost. Dijkstra's algorithm find shortest path matrix D of graph G.

(i) Repeat step 2 for I = 1 to N
$D[i] = cost[i][i]$;

(ii) Repeat step 3 and 4 for i = 1 to N

(iii) Repeat Step 4 for j = 1 to N

(iv) if $D[j] > D[i] + D[i][j]$
then $D[j] = D[i] + D[i][j]$

(v) Stop.

Page:
Date: / /

Conclusion: Shortest path algorithm using Dijkstra's algorithm ~~was~~ for a graph was successfully implemented.