Abdul Saboor

20L-1113

BDS-5A1

Assignment # 6

**Question # 1:**

* Firstly, initialize distance array to all vertices with minimum and store starting course with 0’s.
* Then apply topological sort along this to get courses in topological order.
* Now, use topological order to find distance between start and end course in a loop by incrementing the distance after one course is processed.

**Question # 2:**

The idea is that we start from the given start node but rather than saving a single vertex as parent, we store a list of parents if more than one node gives the same distance. Number of unique shortest paths from **S** to **S** is 1. For all other vertices, number of distinct shortest paths are the sum of distinct shortest paths of their parents.

The pseudocode is as following:

*BFS\_Algorithm(Graph,S){*

*distinct\_paths[1…v]*

*color[1…v]*

*p[1…v] // here each element is a list*

*for all v in V: // V = vertices*

*distinct\_paths[v] = 0*

*color[v] = white*

*dist[v] = ∞*

*p[v] = NULL*

*color[s] = grey*

*dist[s] = 0*

*distinct\_path[v] = 1*

*Queue Q // FIFO queue*

*Q.insert(s)*

*While (Q is not empty):*

*u=Q.pop()*

*for each v in u:*

*if (color[v] == white):*

*color[v] = grey*

*dist[v] = dist[u] + 1*

*p[v].insert(u)*

*distinct\_path[v] += distinct\_path[u]*

*Q.insert(v)*

*else:*

*if (dist[v] == dist[u+1]):*

*p[v].insert(u)*

*distinct\_path[v] += distinct\_path[u]*

*color[u] = black*

**Question # 3 (A):**

It is true, usually Dijkstra’s algorithm takes O((E+V)\*log(V)) time.

SO, for a Korchoff graph,

|E| = #(edges in the tree) + #(edges going from leaves to root)

#(edges in the tree part) = |V|- 1

#(edges going from leave to root) = |V|

Therefore, |E|+ 2|V|

Hence a call to Dijkstra’s takes only O(V\*log(V))

**(B):**

* Find the path from **s** to all its descendants in the tree using a DFS algorithm starting from **s** and going to root while ignoring the (x,r) type edges
* Find shortest path from **s** to **r** in linear time. This can be done in same DFS as above by specially treating (x,r) type edge. (Updating a min\_so\_Far variable for **r** with each (x,r) type edge)
* Find paths from **r** to every other node using a DFS (ignoring **s** and its descendants, i.e., ignoring any (x,s) & (x,r) type edge in the traversal.

**Question # 4:**

A source vertex will be hotels

* Minimum risk tree will be generated. It will contain the minimum risk to reach other houses.
* A priority queue will be used which is filed with vertices and a priority. Priority represents risks of getting to that index.
* Initialize priority of source with 0 and all other with infinity
* With each iteration, lowest priority vertex will be processed
* If any of the adjacent vertices have been visited then they will be skipped. Otherwise, we will compile the priority of adjacent vertex with sum of the edges weight and priority of the current vertex.
* After the initialization, if the source v has a cost of O, the two has cost x. x+O < infinity (cost of getting current V + edge (v,u) < cost of getting to u)
* If cost of getting to current vertex + edge cost of getting to an adjacent vertex < priority of adjacent vertex in our queue, then priority of adjacent vertex in queue will be updated to calculated risk.
* Thus, the order in priority queue can change the updated adjacent vertex. It can move up or down in priority thus affecting where it is