Design And Analysis Of Algorithms
Assignment : 2

Name : Avik Samanta
Roll no. : 204101016

Submission Date : October 05, 2020

1 Ques. : 1(a)

Our **Algorithm : 1** does three things :-

1. **Traverse the Graph $G = (V, E)$ in Breadth First manner**.

- For the algorithm we are using the Queue Q, which uses FIFO concept
- Now what it is doing, first it inserts the source node s into Q [Consider it to be in level 0]. Then all the adjacent nodes of s are being inserted [which are in level 1]. Then when these nodes are processed, the adjacent nodes of them are inserted into Q [nodes of level 2], and so on. Thus our Algorithm processes all the nodes in the Graph in level by level manner.
- So, Algorithm 1 gives us the **Breath First Traversal** of the graph G

2. **cost[v] : Minimum cost** to reach node $v(\in V)$ from $s(\in V)$ [Considering the edges to be unweighted, if not then **cost[v]** indicates the **Level of the node** v in the Level Order Traversal]

- Initially the $cost[s] = 0$, means the source node can be reached with 0 cost (we are already on it), or the level of the source node is 0 (zero).
- Now look at the code snippet : $if(cost[v] = \infty)\ cost[v] = cost[u] + 1$
- $cost[v] = \infty$ means the node v is not visited yet. And u is the current node we are exploring.
- So if edge $(u, v) \in E$, and v is not explored, then we are adding 1 to the $cost[u]$ and store it in $cost[v]$. That means v is a child of u in the level order traversal, and v is one level below of u. So the cost to reach v is one more than cost to reach u.
- As Level order traversal gives us the minimum edge distance from source to any node, the $cost[v]$ keeps track of the **Minimum Cost to reach node** v (or the level of node v).

3. **nums[v] : Number of paths** from Source s to node $v(\in V)$ **with Minimum cost**

- Initially the $nums[s] = 1$, as there is always a path to reach a node from the same node, and that path is the one with no edges (zero cost). Means the source node has one single path to reach the source node (path with no edges).
- Now look at the code snippets :-
  $if(cost[v] = \infty)\ nums[v] = nums[u]$
  $if(cost[v] = cost[u] + 1)\ nums[v] = nums[u] + nums[v]$
- $cost[v] = \infty$ means the node v is not visited yet. And u is the node we are currently exploring. If there's an edge $(u, v) \in E$, and v is not explored, then v is a child of u. The paths with which we can reach u, we can also reach v through edge (u, v). So There will be same number of minimum cost paths to reach v as of u (until some new discoveries take place).

- $cost[v] = cost[u] + 1$ means v is already explored and the cost to reach u, is same as cost to reach the parent (already assigned parent) of v. So there is also an alternative path to reach v (through u) with the same minimum cost as of assigned already. In that case we are adding the $nums[u]$ (number of minimum cost paths to reach u) to the $nums[v]$ (as we recently explored there could me $nums[u]$ more paths to reach v, though u, with same minimum cost).
- So at the end the $nums[v]$ will give us the **number of paths with minimum cost** from s to v.

2 Ques. : 1(b)

Time Complexity :-
First assume, the graph $G = (V, E)$ has $|V|$ = N nodes and $|E|$ = M edges

- Initializing the $nums[v]$ list : $O(N)$ time
- Initializing the $cost[v]$ list : $O(N)$ time
- The queue Q will have exactly one entry for all the nodes [only if G is connected; if not then no entry for the nodes which are not reachable from s]. So the while loop : $O(N)$ times
- The for loop inside while loop, runs for all the adjacent nodes of u [current dequeued node], or in others words for all the edges connected to u. So, for the whole algorithm, $\sum_{\forall u \in V} adjacent(u) = 2 * M$ (M : number of edges) [multiplied by 2 : as edges are undirected, so will be counted for both the nodes connected by the edge] : $O(M)$
- Note : InsertQueue() and DeleteQueue() takes $O(1)$ time
- So, Time Complexity = $O(N) + O(N) + O(N) + O(M) = \mathbf{O(N + M)}$

3 Ques. : 2(a)

Our **Algorithm : 1** does three things :-

1. Single Source Shortest Path **using Dijkstra's Algorithm**.

- There is a well defined source s, and we find the shortest paths to reach all the nodes from the source s
- What Dijkstra's Algorithm does is that, there will be a set of nodes S for which the shortest path is already found. And each time we will select the node v (not in S) with minimum distance, we will add it to S, and then relax all the edges incident on v. relaxing means updating the cost/distance of the node (based on some selected node) if possible. Then repeat the process, until shortest path for all the nodes are found.
- Initially the S contains only s, the source node. As it has the $cost[s] = 0$, and we already have the shortest path for source s, with no edges and 0 cost

2. **cost[v] : Minimum cost / Shortest Distance** to reach node $v(\in V)$ from $s(\in V)$

- Initially the $cost[s] = 0$, means the source node can be reached with 0 cost (we are already on it)
- Now look at the code snippet : $if(cost[v] > cost[u] + l_e)\ cost[v] = cost[u] + l_e$
- $(cost[v] > cost[u] + l_e)$ means the cost of previously computed path for the node v is greater than the path we recently discovered through u.
- So we update the distance/cost of node v from source s, with the newly discovered path (which up until now is the minimum cost to reach node v).

3. **isGood[v]** : isGood[v] is False if the distance to any node adjacent vertex of v is greater than or equal to the distance of vertex v. Else the isGood[v] value will be True.

- All intermediate nodes in Dijkstra's algorithm will be False
- Only $isGood[v]$ value for the Leaf nodes can be True
- If there is only one path to reach the leaf node, then definitely will be True.
- Else one node can only be True if the last relaxing done on it, that edge belongs to the Shortest path

4 Ques. : 2(b)

Time Complexity :-
First assume, the graph $G = (V, E)$ has $|V|$ = N nodes and $|E|$ = M edges

- Initializing the $cost[v]$ list : $O(N)$ time
- Initializing the $isGood[v]$ list : $O(N)$ time
- Constructing the Priority Queue takes : $O(N)$ time
- The Priority Queue Q has N elements initially. And each time we are deleting one element from the Q. So the while loop runs for N times : $O(N)$ time
- Each time we enter the while loop, we are deleting the minimum element. $deleteMin(Q)$ takes $O(logN)$ time, when we implement Priority Queue with Min Heap. So for the whole algorithm, this $deleteMin(Q)$ will take : $O(NlogN)$ time
- The for loop inside while loop, runs for all the adjacent nodes of u [current dequeued node], or in others words for all the edges connected to u. So, for the whole algorithm, $\sum_{\forall u \in V} adjacent(u) = 2 * M$ (M : number of edges) [multiplied by 2 : as edges are undirected, so will be counted for both the nodes connected by the edge] : $O(M)$ time
- The $decreaseKey(Q, v, cost[v])$ takes (when Q is implemented using Min Heap) : $O(logN)$ time. So for the whole algorithm this will take : $O(MlogN)$ time
- So, Time Complexity = $O(N)+O(N)+O(N)+O(M)+O(MlogN)+O(NlogN)$ = **O((N + M)log N)**