U00834259
Sindhuja Yerramalla

HomeWork 3

Problem 1: Problem 5.9 (b) (d) (g) from DPV

b) The statement is True.

we cannot have any cycles in MST because MST is tree.

If the edge e is unique and heaviest weight in the cycle. then we need to remove that edge to form a tree. If we remove any other edge it would still be tree but not MST as a tree with less weight is possible.

d) The statement is true.

If the lightest weight of the graph is unique then it is definitely a part of a MST. In both Kruskals and Prims algorithm. We sort the edges based on their weight and then start adding those edges to the tree. So the edge with the lightest weight is always present in the MST.

g) The statement is False

In calculating the MST using prims is similar to the algorithm that is used in Dijkstra's but the only difference is the ordering of the priority queue in both the cases. In shortest path we consider the shortest distance to a vertex but in prims we consider the shortest edge from the vertex irrespective of the distance from the source.

Problem 2 : Problem 5.21 from D.P.V.

a) A tree is a graph with no cycles. A MST is a tree with minimum total weight of edge. So if we remove a highest weight edge from a cycle. we are removing cycles. Then we are left with the tree that connects all the vertices and the total weight of the tree is minimum.

So, the result gives the required tree with the minimum weights also called as MST.

b) let T be the MST of Graph G. where G has n edges. let the edges be ordered by weight as e1, e2, e3 etc. The edge e is not present in the graph T because T has no cycles and the deleted edge is the heaviest in the cycle, therefore we remove any edge e present in the cycle from the graph G. In order to ensure that any edge with the highest weight in the cycle of G is removed from T after processing all the edges, we only delete edges from cycle in G.

Scanned with CamScanner

Since a smaller weighted tree may exist, the MST of G is not the case if e is present in cycle and T. As This indicates that the subgraph that results from deleting the edge from G with the highest weight returns the MST of graph, which is the tree with the lowest weight.

c) To determine whether a cycle is present in a graph, we use DFS. To determine whether a cycle contains an edge, we take one of the edge's vertices and use. DFS. If a visited vertex is found again, during exploration, we can claim that a cycle has begun.

A DFS has a time complexity of $O(|V|+|E|)$.

→ If a vertex of the edge $e(u,v)$ is considered the vertex u is visited once again at the end of the exploration. The exploration skips over the vertex u if it is not in a cycle. This proves the algorithm's accuracy.

d). Time Complexity:

For sorting it takes $O(|E|\log|E|)$ by merge sort.

For iterating through the edges it takes $O(|E|)$ time. and the DFS for finding cycles $O(|V|+|E|)$ which is $O(|E|)$ as $|V| <= 2|E|$. and removing edges take $O(1)$

∴ Total time complexity is $O(|E|\log|E| + |E|^2)$ which is $O(|E|^2)$.

Scanned with CamScanner

Problem 3 :- 5.31 from DPV.

The algorithm used by the server is optima, since there is no priority for any customers and all the orders and the waiting times are given at once.

Because of the server's greedy algorithm, clients with shorter wait times are served first, reducing the first customer's waiting time.

Consider an example:

Customer1 1

Customer2 2

Customer3 3.

If served in increasing order

c1 waits for 1, c2 waits for 3, c3 waits for 6. So total wait time is 10

If served in decreasing order:

c3 waits for 3, c2 waits for 5, c1 waits for 6. Total wait time is 14

So server is following the best possible order

Algorithm

Sort customer based on wait time

T=0

For I in sorted Customers:

    T = T + customer[i]. wait time.

return T

The time complexity is o(nlogn) for sorting & o(n) for iterating. so total time complexity is o(nlogn) ⁄⁄.

References :- Referred to many websites, Textbook, lecturenotes, chegg and worked with few classmates.