Name :- Vandit Jyotindra Gajjar
Student ID :- a1779153

Quiz - 3

Q 1.1 Shortest path (s - t)

We have a weighted directed graph
$G = (V, E, w)$ in which
V is number of vertices
E is number of Edges
w is number of weights
X be a shortest path $s - t$ for $s, t \in V$.

Here to prove the given path is shortest path or not.

→ We will consider one of example to demonstrate the proof.
Consider a Graph X with 3 nodes (m, n, o) with directed edges. shown in figure 1.

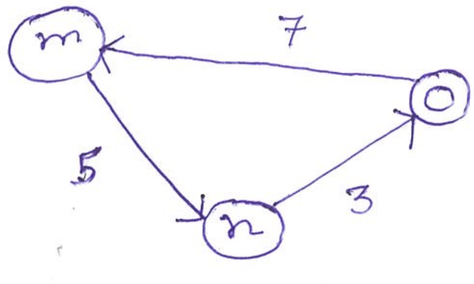

Figure 1

Here the shortest path from m to o

is $= m > n > o$
$= (m \to n) + (n \to o)$
$= 5 + 3$
$= 8$

So, the shortest path from m to o is 8.

→ Now, as per the condition, we need to double the edges & weight of the graph. shown in figure 2.

P.T.O.

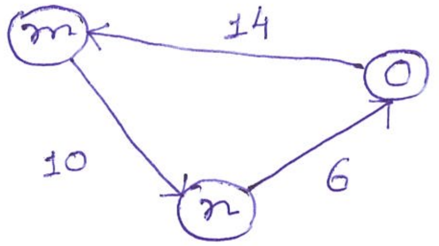

Figure 2

Here, the shortest path from m to o is

= m > n > o

= (m → n) + (n → o)

= 10 + 6 = 16

So, the shortest path from m to o is 16.

→ Hence, we can prove here that by doubling weights of the graph the shortest path will remain shortest path. In general, considering any linear transformation applied to the graph, the shortest paths will continue to be shortest path, also the path will have same ordering (m > n > o).

P.T.O

Q 1.2

Dijkstra's algorithm for the graph The graph shown in figure 3.

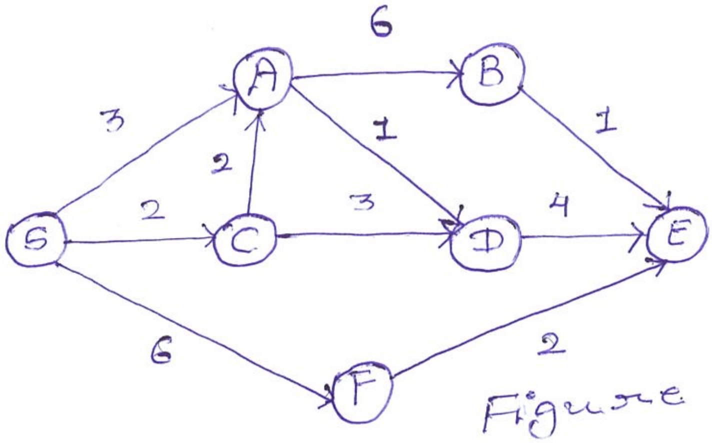

Figure 3

In Dijkstra's algorithm picks the unvisited vertex with the lowest distance, calculates the distance through it to each unvisited neighbor, and updates the neighbor's distance if smaller.

→ So by following the algorithm key facts we will visit the vertices in the following order

$S \rightarrow C \rightarrow A \rightarrow D \rightarrow F \rightarrow E \rightarrow B$

→ Here the algorithm will relax the edge from $D \rightarrow E$ before the edge from $F \rightarrow E$. As D is closer to S than F is.

The result will be in Figure 4.

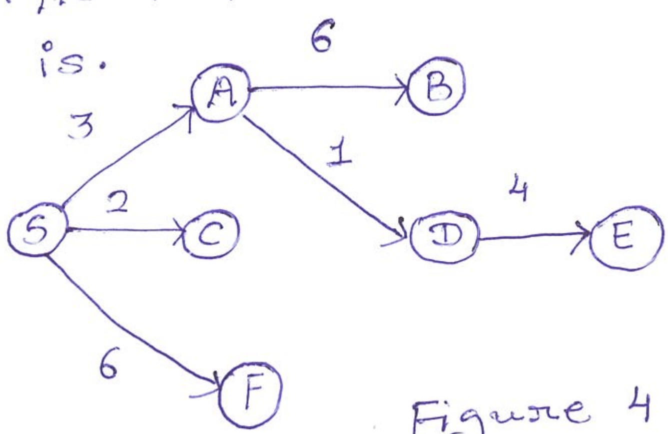

Figure 4

P.T.O

Q 1.3

The time-complexity of Floyd-Warshalls algorithm is $O(n^3)$, whereas best-time complexity for Dijkstra's algorithm is $O(m + n\log n)$ for shortest path.

As per the question, if we are using fibonacci heap for Dijkstra's algorithm for storing the distances (i.e. all nodes vertices distance/costs) the time complexity will be $O(M + N\log N)$ Here m will be N^2 as fibonacci heap is used.

so by comparison, we can see that dijkstra's algorithm with fibonacci heap is more efficient than Floyd-warshalls.

$$O(n^3) > O(n^2 + n\log n) \quad \text{---- (1)}$$

However, for arbitary graphs such as considering combination of positive and negative edge costs the dijkastra's algorithm won't work.

for example a graph is given in figure 5.

P.T.O

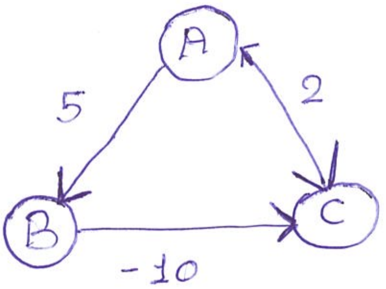

Figure 5

In Dijkstra's algorithm a vertex is marked as closed - the algorithm found the shortest path to it, thus will never develop the node again - it assumes the path developed to this path is the shortest.

But for negative weights this won't true.

V = (A, B, C)

E = (A, C, 2), (A, B, 5), (B, C, -10)

So here the dijkstra's algorithm will develop A → C.

But the actual path should be A → B → C.

Conclusion :- So, the dijkstra's algorithm will works better if weights / costs are positive otherwise Floyd warshalls works better for positive - negative weights.

P.T.O

Q. 1.4 Solve the graph using Floyd-warshall's algorithm

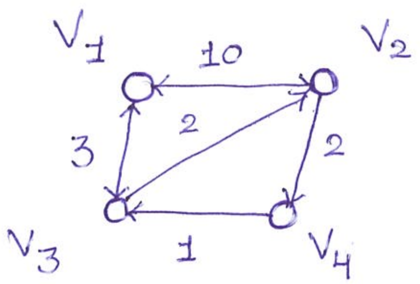

Figure 6

If d_mat = 0

Destination

$$d\_mat^0 = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 10 & 3 & \infty \\ 2 & 10 & 0 & \infty & 2 \\ 3 & 3 & 2 & 0 & \infty \\ 4 & \infty & \infty & 1 & 0 \end{array}$$

Source

Node V_1 Destination

$$d\_mat^1 = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 10 & 3 & \infty \\ 2 & 10 & 0 & 13 & 2 \\ 3 & 3 & 2 & 0 & \infty \\ 4 & \infty & \infty & 1 & 0 \end{array}$$

Source

Node V_2 Destination

$$d\_mat^2 = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 10 & 3 & 12 \\ 2 & 10 & 0 & 13 & 2 \\ 3 & 3 & 2 & 0 & 4 \\ 4 & \infty & \infty & 1 & 0 \end{array}$$

Source

Node V_3 Destination

$$d\_mat^3 = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 5 & 3 & 7 \\ 2 & 10 & 0 & 13 & 2 \\ 3 & 3 & 2 & 0 & 4 \\ 4 & 4 & 3 & 1 & 0 \end{array}$$

Source

Node V_4 Destination

$$d\_mat^4 = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 5 & 3 & 7 \\ 2 & 6 & 0 & 3 & 2 \\ 3 & 3 & 2 & 0 & 4 \\ 4 & 4 & 3 & 1 & 0 \end{array}$$

Source

P.T.O

Q 2.1 Applying kruskal's algorithm for the graph.

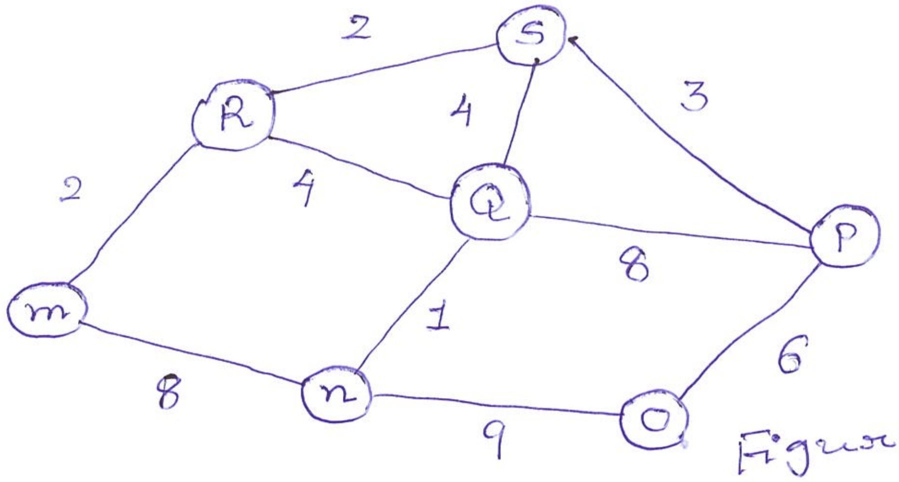

Figure 7

→ sort the edges as per kruskal's algorithm in ascending order
T = True ; F = False

edge	cost	
n - Q	1	T
m - R	2	T
R - S	2	T
S - P	3	T
S - Q	4	T
R - Q	4	F
P - O	6	T
Q - P	8	F
m - n	8	F
n - O	9	F

P.T.O

→ Final spanning tree

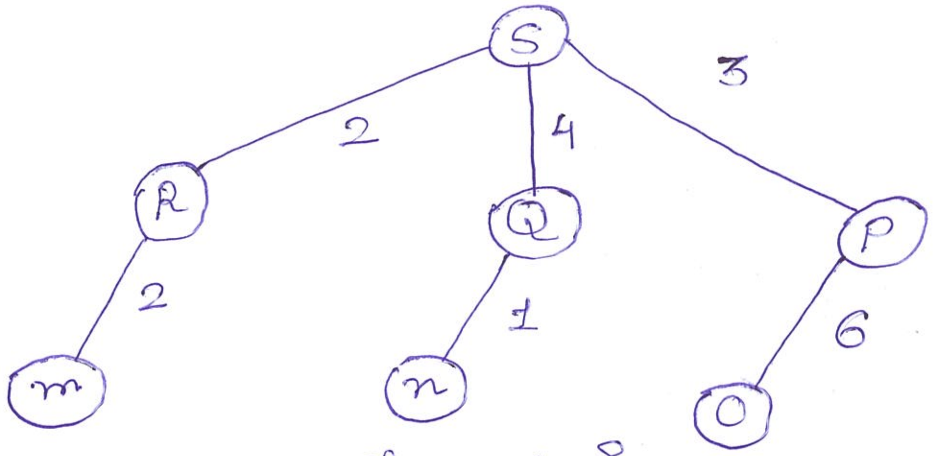

Figure 8

Q 2.2

We use Union-find data-structure in Kruskal's Algorithm in Order to discover or Create a cycle in the graph, for that graph whether we need ~~a~~ a supplemental edge or not.

→ Considering minimum spanning tree as the tree is having no cycles and thus we do not require any cycle in here.

Q 2.3 Find operation implementation in the Union-find data structure

→ In Kruskal's Algorithm as edges are added, components change, so the whole structure is dynamic.

Using the find operation, it compress the tree while finding and makes all the ancestors link directly to the root.

→ We first make subsets of individual vertices. Here, we can use different data structure arrays, linked lists, trees etc. After, we extract the vertices of the edge and then check if they are in our subsets or not. If present in other subsets then by using Union operation we can combine those subsets. We continue this operation untill every edges completed in the graph. In one of case, If vertices present in same subset, than there will be a cycle and the edge isn't in the MST.

P.T.O

Q 2.4

~ A problem is said to be nondeterministic polynomial time class when the solution of the problem takes polynomial time.

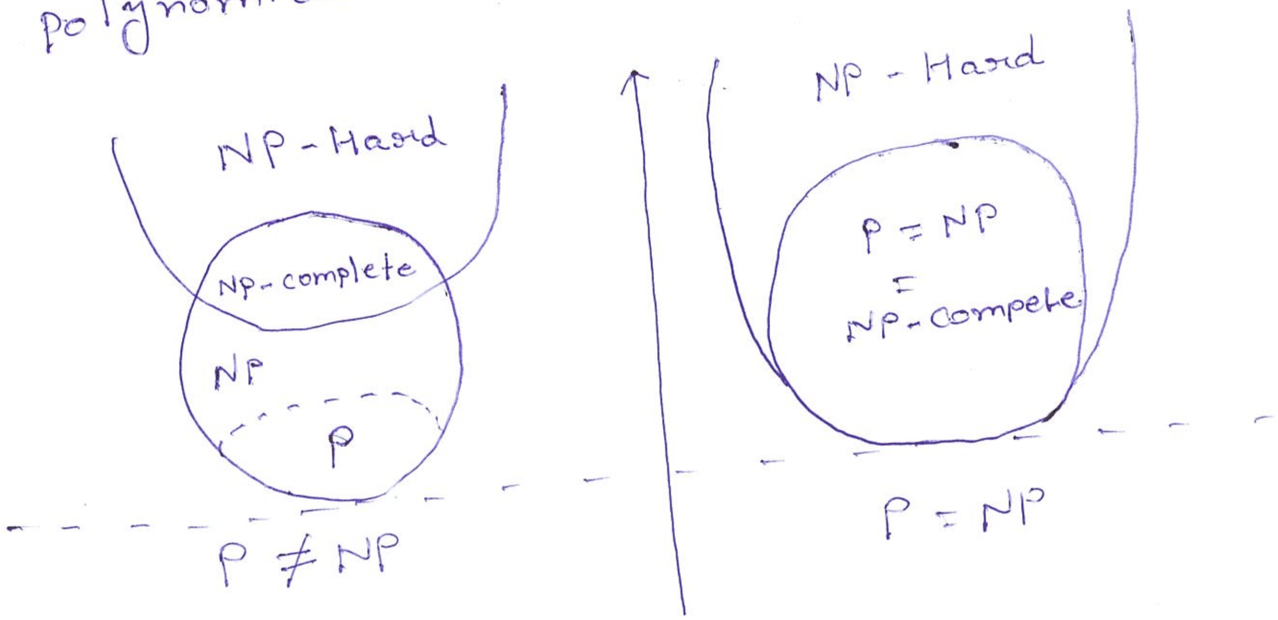

Figure 9 Euler diagram for P, NP, NP - complete, and NP - Hard

~ Example :- Travelling Salesman Problem
:- Hamiltonian path problem
:- Boolean Satisfiability Problem

P. T. O

Q 2.5

To classify a problem H is NP-Hard when for every problem L in NP, there is a polynomial-time many-one reduction from L to H.

~ somewhat equivalent definition is to require that every problem L in NP can be solved in polynomial time by an oracle machine with an Oracle for H. As from the figure 9, we can see the NP-Hard problem can't be solved in polynomial time if P ≠ NP.

~ Example :- Subset sum problem
:- Halting problem
:- Boolean formulas true quantified

P.T.O

Q 2.6

The minimum spanning tree problem is in P problem. [P-class]. So, the mst can be solved in polynomial time.

- The time complexity is O(p(n)); where p(n) is polynomial for p ~~cat~~-class problems.
- For MST it is O(m log m) where m log m is in polynomial time. for kruskal's algorithm.
- Thus from the time complexity in terms of polynomial time, we can conclude that MST is a p-type / p-class problem.

- - - - - - - - END - - - - - - - - - -

Date 10/06/2020

sign. Vandit