DESIGN AND ANALYSIS OF ALGORITHMS (DAA)

ASSIGNMENT-3

Submitted by: ASTHA CHOUDHARY   Roll no- 30
Section: DS1

Q1 what do you mean by Minimum spanning Tree? What are the application of MST.

Ans: A tree which spans all the vertices of a graph having minimum total weight of edges is known as a minimum spanning tree. It is a connected graph which has no cycles.

Applications of MST :-

→ Games Development → in generating procedural maps creating paths between points of interests in gaming world.

→ Image Processings → used in image segmentation and edge detection tasks.

→ Network designing → helps in designing of network with least possible costs of designing purposes.

→ Civil Engineering and Transportation → To minimize the cost of purpose while ensuring connectivity between different location as per purpose.

→ Clusterings → In data analysis used for clustering data points.

Eg →

Ans ②

① Prim's Algorithm -

Time Complexity:

* Depends on Data structure used to represent Graph
* And priority queue used to select the next minimum edge.

When using adjacency list and Binary heap (such as priority queue)

$$O(\underbrace{(V+E)}_{\text{Vertices}} \log V)$$

Edges

If fibonacci heap is used:

$$O(E + V \log V)$$

Space Complexity:

$O(V+E)$

Due to storage for the adjacency list and the data structures used (priority queue, visited set)

② Kruskal's Algorithm -

Time Complexity:

* Starts by sorting the edges, $O(E \log E)$
* Also uses union-find data structure to check cycles, which takes nearly constant time for each operation.

$$O(E \log E)$$

Space Complexity:

$O(V+E)$

Due to edge list and the union-find data structure.

(3) Dijkstra's Algorithm -

Time Complexity:

* Depends on the data structure used to represent the graph and the priority queue used.
* Using adjacency list and binary heap:

$$O((V+E)\log V)$$

* If fibonacci heap is used,

$$O(E + V\log V)$$

Space Complexity

$O(V+E)$

Due to adjacency list, priority queue and storage for distances and predecessors.

(4) Bellman-Ford Algorithm:

$O(VE)$

no. of vertices ← V, E → no. of edges.

As the algorithm iterates over all the edges upto V-1 times.

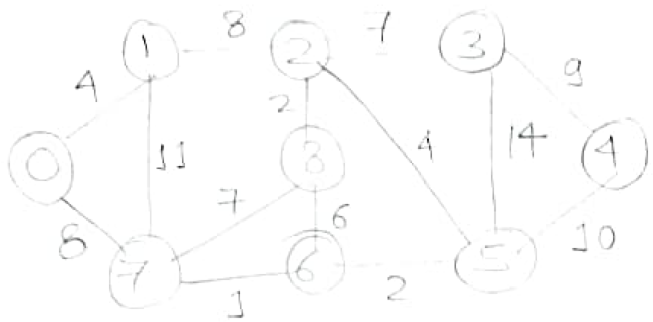
Space Complexity

$O(V+E)$

Due to storage of distances and predecessors of each vertex, as well as the edge list.

Ans - 3

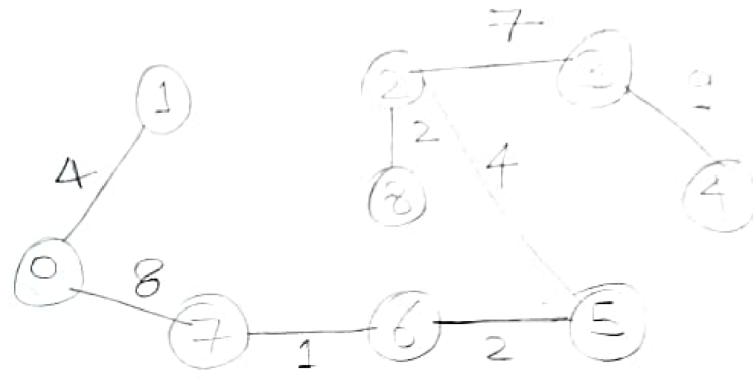
Kruskal Algorithm -



u	v	w
7	6	1
2	8	2
6	5	2
0	1	4
2	5	4
6	8	6
7	8	7
2	3	7
0	7	8
1	2	8
3	4	9
5	4	10
1	7	11
3	5	14

Increasing Weight Order

↓ MST



MST Weight : 37

Ans-4

1. Increasing every edge by 10 units:

* It is an additive transformation.
* Since the absolute difference b/w^n the weights of paths remains the same, the relative ordering of path does not change.

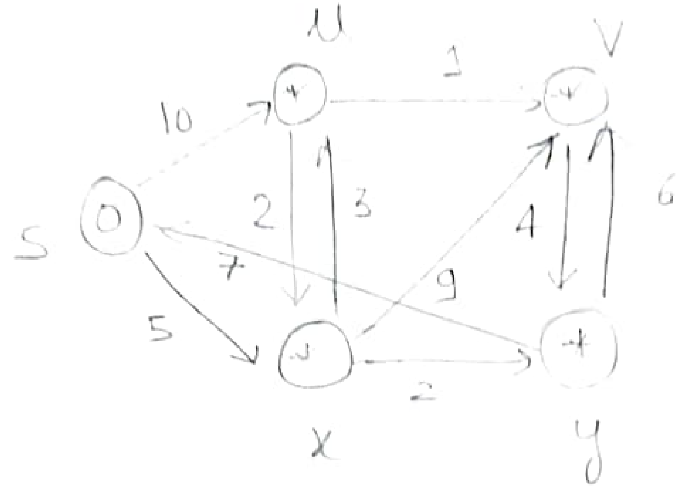
Thus, the shortest path from source vertex 's' to destination vertex 't' remains the same after the modification.

2. Multiplying every edge weight by 10 units:

* It is a multiplicative transformation.
* No change, i.e shortest path from 's' to 't' remains same after modification.

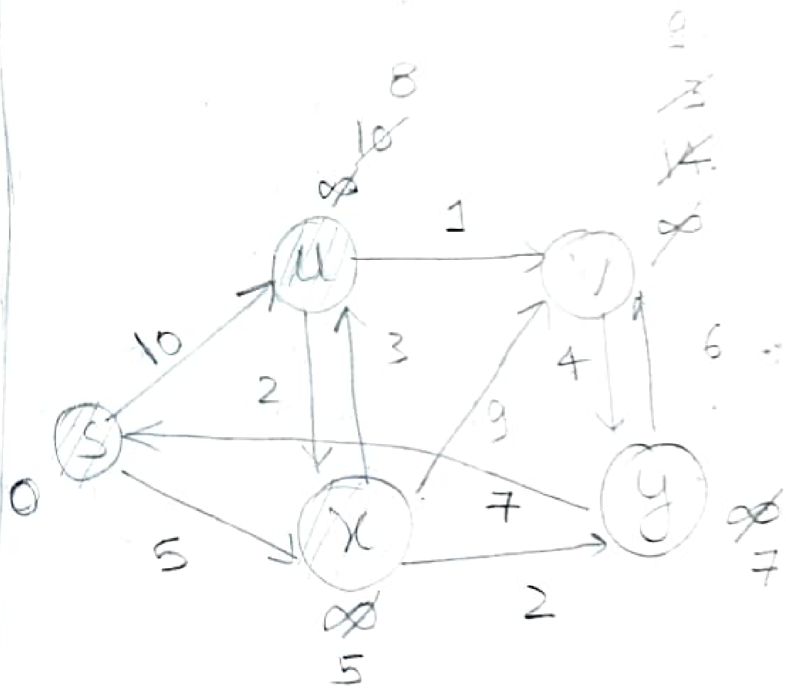
→ As the changes are uniform across all edges, there will be no change in shortest path from 's' to 't'. Relative order of path weights remains the same, preserving the shortest path between two vertices.

Ans - 5



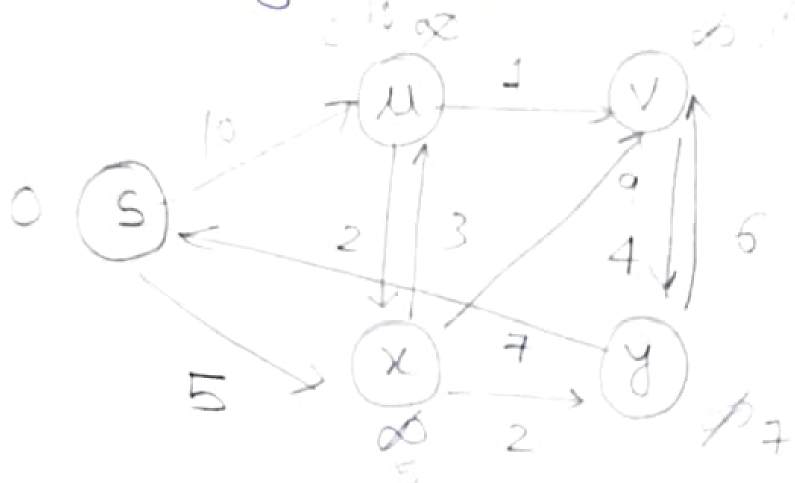
Dijkstra's Algorithm -

Selected Vertex	S (Source)	u	v	x	y
S	$\boxed{0}$	~~∞~~	~~∞~~	~~∞~~	~~∞~~
x		~~10~~	~~∞~~	$\boxed{5}$	~~∞~~
y		8	~~14~~		$\boxed{7}$
u		$\boxed{8}$	~~13~~		
v			$\boxed{9}$		



Node	Distance from source S
s	0
x	5
y	7
u	8
v	9

Bellman Algorithm-



edges-

(s,u)	(s,x)	(x,u)	(x,y)	(x,v)	(u,v)	(u,x)	(y,v)	(y,s)	(v,y)
✓	✓	✓	✓	✓	✓	✗	✗	✗	✗

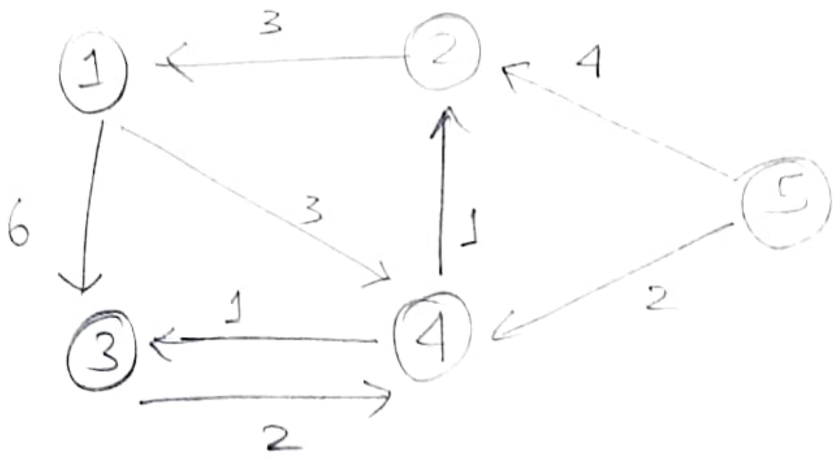
1
2] same
3

✓→ edges updated
✗→ no updation needed

	s	u	v	x	y
i = 1	$\boxed{0}$	~~∞~~ 10	∞ 14	~~∞~~ 5	~~∞~~ 7
i = 2		8	9		
i = 3		8	9	5	7

Vertex	Dist. from s
s	0
x	5
y	7
u	8
v	9

Ans - 6



Floyd Warshall Algorithm -

$$D_0 = \begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & 0 & \infty & 6 & 3 & \infty \\ 2 & 3 & 0 & \infty & \infty & \infty \\ 3 & \infty & \infty & 0 & 2 & \infty \\ 4 & \infty & 1 & 1 & 0 & \infty \\ 5 & \infty & 4 & \infty & 2 & 0 \end{array}$$

$$D_1 = \begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & 0 & \infty & 6 & 3 & \infty \\ 2 & 3 & 0 & 9 & 6 & \infty \\ 3 & \infty & \infty & 0 & 2 & \infty \\ 4 & \infty & 1 & 1 & 0 & \infty \\ 5 & \infty & 4 & \infty & 2 & 0 \end{array}$$

$$D_2 = \begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & 0 & \infty & 6 & 3 & \infty \\ 2 & 3 & 0 & 9 & 6 & \infty \\ 3 & \infty & \infty & 0 & 2 & \infty \\ 4 & 4 & 1 & 1 & 0 & \infty \\ 5 & 7 & 4 & 13 & 2 & 0 \end{array}$$

$$D_3 = \begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & 0 & \infty & 6 & 3 & \infty \\ 2 & 3 & 0 & 9 & 6 & \infty \\ 3 & \infty & \infty & 0 & 2 & \infty \\ 4 & 4 & 1 & 1 & 0 & \infty \\ 5 & 7 & 4 & 13 & 2 & 0 \end{array}$$

$$D_4 = \begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & 0 & 4 & 4 & 3 & \infty \\ 2 & 3 & 0 & 7 & 6 & \infty \\ 3 & 6 & 3 & 0 & 2 & \infty \\ 4 & 4 & 1 & 1 & 0 & \infty \\ 5 & 6 & 3 & 3 & 2 & 0 \end{array}$$

Resultant Matrix

Time complexity: $O(n^3)$

($n \rightarrow$ no. of vertices in Graph)

* As algorithm consists of three Nested loops (one for each pair of vertices and one for each intermediate vertex.)

Space complexity: $O(n^2)$

($n \rightarrow$ no. of vertices)

* Because algorithm maintains a 2-D array to store shortest paths b/w each pair of vertices