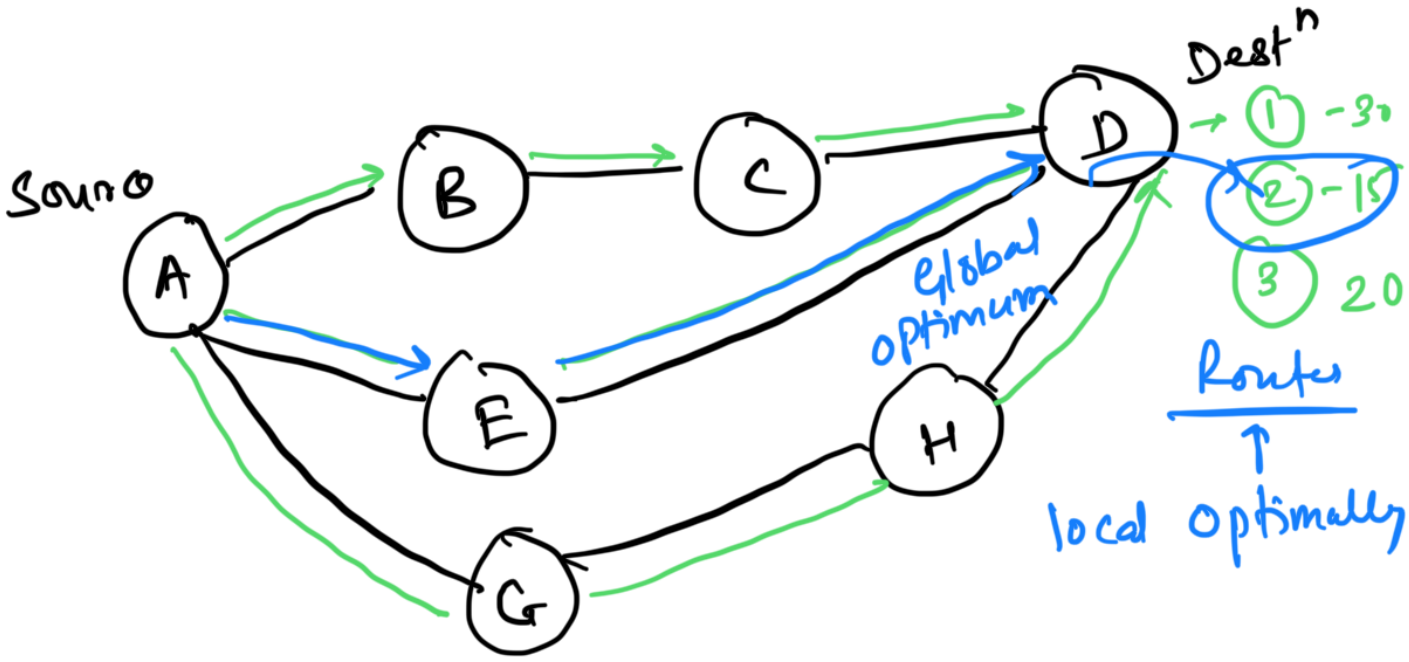
Graph
- Shortest Path Algorithm
- Minimum cost spanning Trees.

Algorithm strategy –

1. Greedy Technique
2. Dynamic Programming



Greedy Algorithm

- A class of algorithm that makes locally optimal choices at each step, hoping to get the final optimal solution.

- Works when the problem exhibits –

1. **Greedy Choice Property** –
   – Making the best local choice leads to the global optimal solution.

2. **Optimal substructure** –
   – The optimal solution can be constructed from the optimal solutions of its subproblems.

Steps for Creating a Greedy Algorithm –

1. Define the problem -

- clearly state the objectives of the problem to be optimized.

2. Identify the Greedy Choice -

- Find the locally optimal choice at each step.

3. Make the Greedy Choice -

- Select the best option based on the current state

4. Repeat the process -

- Continue making greedy choices until a Solution is reached.

Characteristics of Greedy Algorithms -

1. Simple to implement
2. fast & Efficient in solving problem
3. Locally optimal choice made at each step
4. Decisions are based on current information, past choices cannot be considered

Examples:

1. Dijkstra's Algorithm - finds shortest path

2. Kruskal's Algorithm - Find the minimum spanning tree

3. Huffman Coding - Compresses data by assigning shorter code to more frequent symbols.

<u>Applications</u> -

1. Task scheduling - min. waiting time
2. Resource allocation
3. Data Compression
4. Network Designs

Dynamic Programming

- A method for solving complex problems by breaking them into simpler subproblems.

- <u>Key concepts</u> -

1. Overlapping Subproblem -
   - Repeated calculations for the same problem

2. Optimal Substructure -
   - The optimal solution to a problem can be constructed from the optimal solutions of its subproblem

<u>Steps in Dynamic Programing (DP)</u> - (working)

1. Identify the problem
2. Solve each subproblem & store the result
3. Build the solution to the main problem using stored results
4. Avoid redundant computations by reusing stored solutions

Approaches for Dynamic Programming

1. Top Down Approach (Memoization)

Subproblems eg. Recursive fibonacci

2. Bottom Up Approach (Tabulation)

smallest problem

eg. Iterative fibonacci using Table

Examples -

1. Fibonacci Sequence
2. Longest Common Subsequent (LCS)
3. Knapsack Problem
4. Shortest Paths.

Greedy Algorithm	DP
Locally optimal sol^n	solve subproblems & builds up to global solution
Not alway guaranted Optimal sol^n	Guarantee globally optimal sol^n

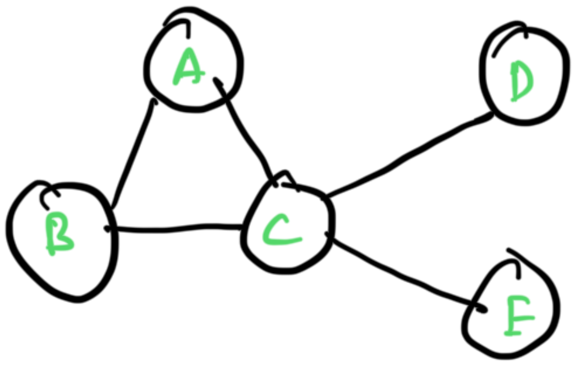
TC: linear or poly
$O(n)$ $O(n^m)$

eg Kruskal eg.

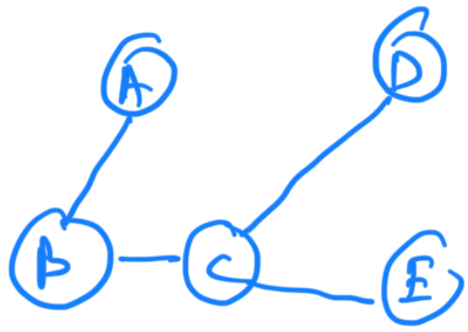
TC: overlapping
↑ TC

eg. shortest path

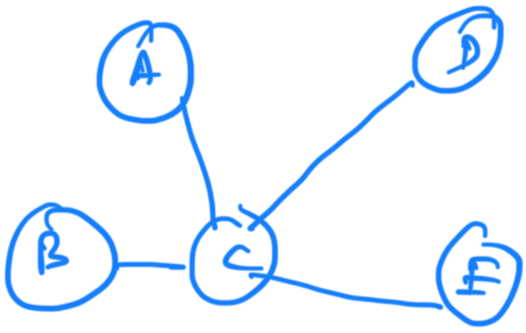
Spanning Tree: $G(V, E)$ – undirected



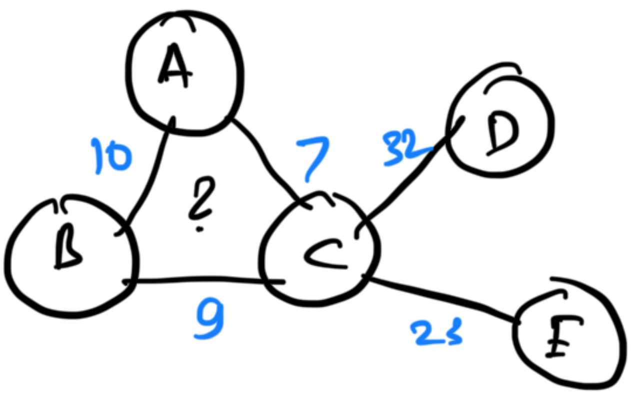
undirected graph

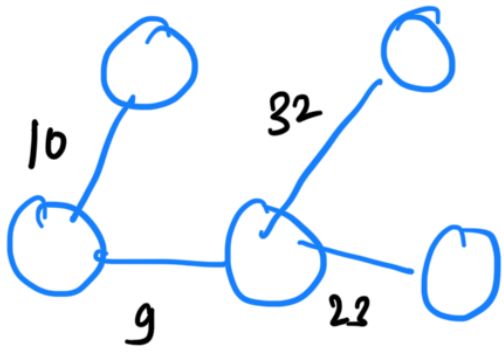


ST1

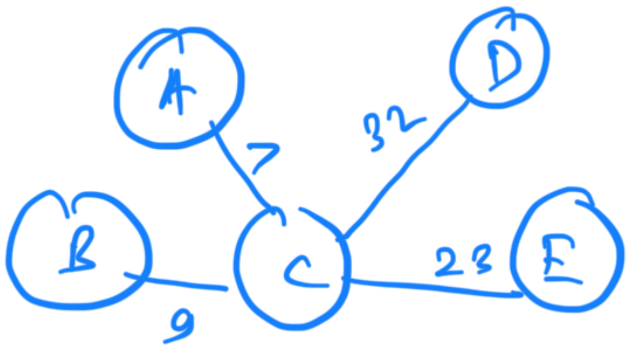


SPT2

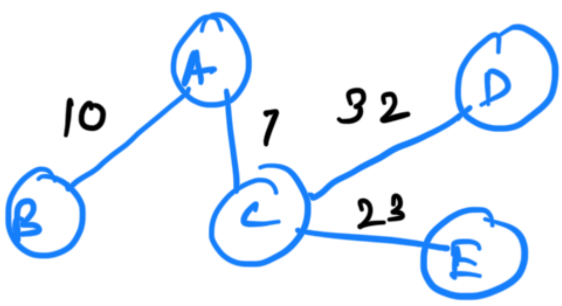




Total cost = 74



$\boxed{\text{Total cost} = 71}$



Total cost = 72

Minimum cost
Spanning Tree

Minimum Spanning Tree: (MST)

- MST is a subset of edges from a connected weighted graph that connects all the vertices without any cycles and with minimum possible total edge weight.

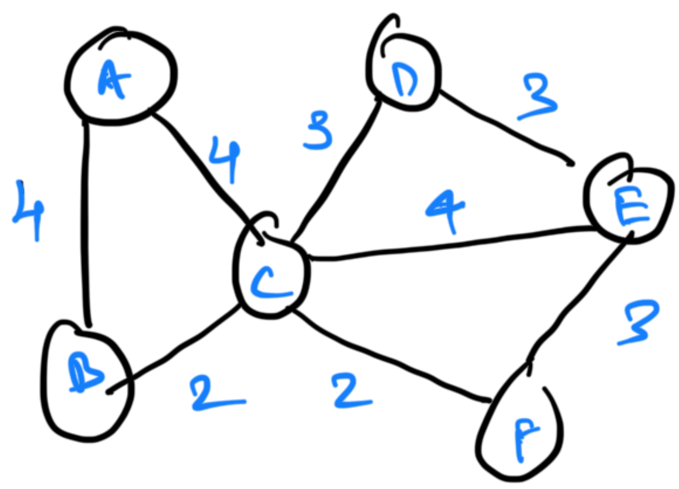
Conditions –

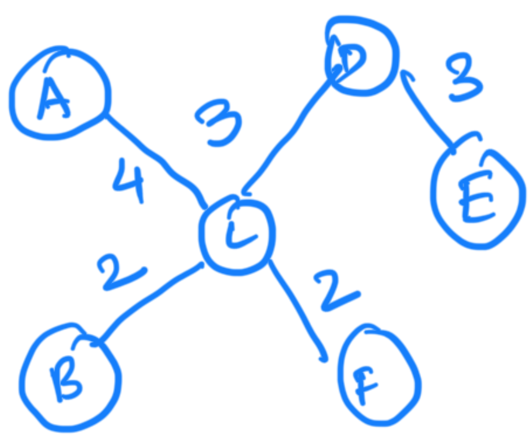
1. Graph connected
2. Graph undirected
3. Weights are unique

To identify minimum spanning Tree

1. Kruskal's Algorithm
2. Prim's Algorithm

Greedy Algorithm





Total cost = 14

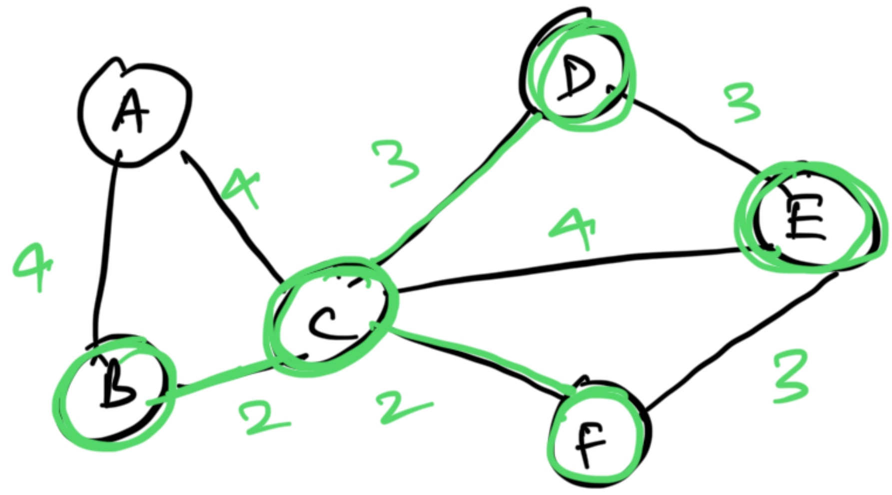
Kruskal's Algorithm

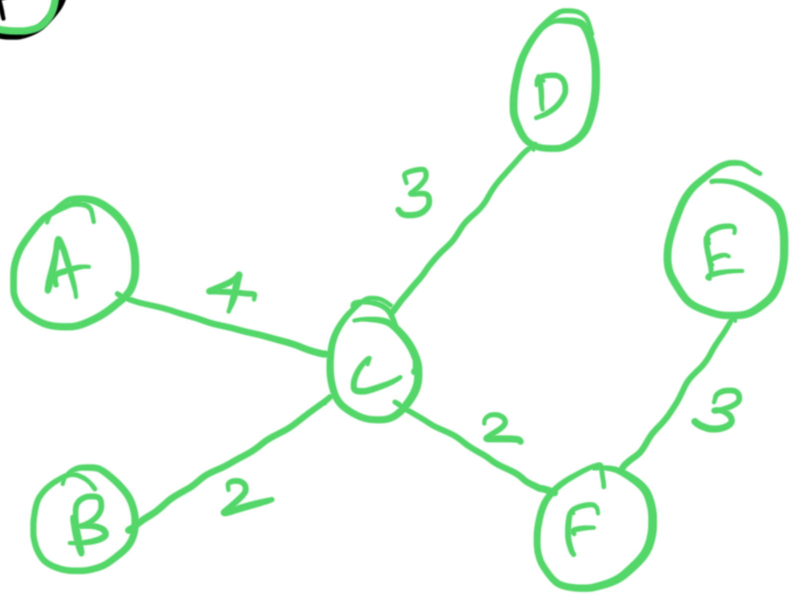
(shortest cost first)

Time Complexity $= O(E \log E)$ E = number of edges

Space Complexity $= O(E+V)$

Prim's Algorithm





Total cost = 14

Time Complexity = $O(E \log V)$

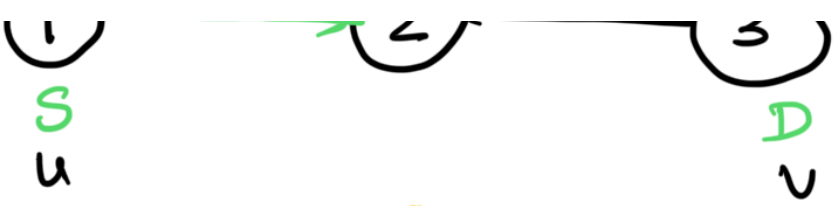
Space Complexity = $O(|V|)$

Dijkstra's Algorithm – Shortest Path in Weighted Graph

- Type - weighted graph with non-negative weights.

- Method - Greedy Algorithm that expands the node with the smallest distance at each step.

Ex



1 — 3 ⇒ No direct path

1 — 2 — 3 ⇒ 6 units

$\cancel{\phi}$ 6 ⟶ $\boxed{\text{Relaxation}}$

↑ updated value of distance

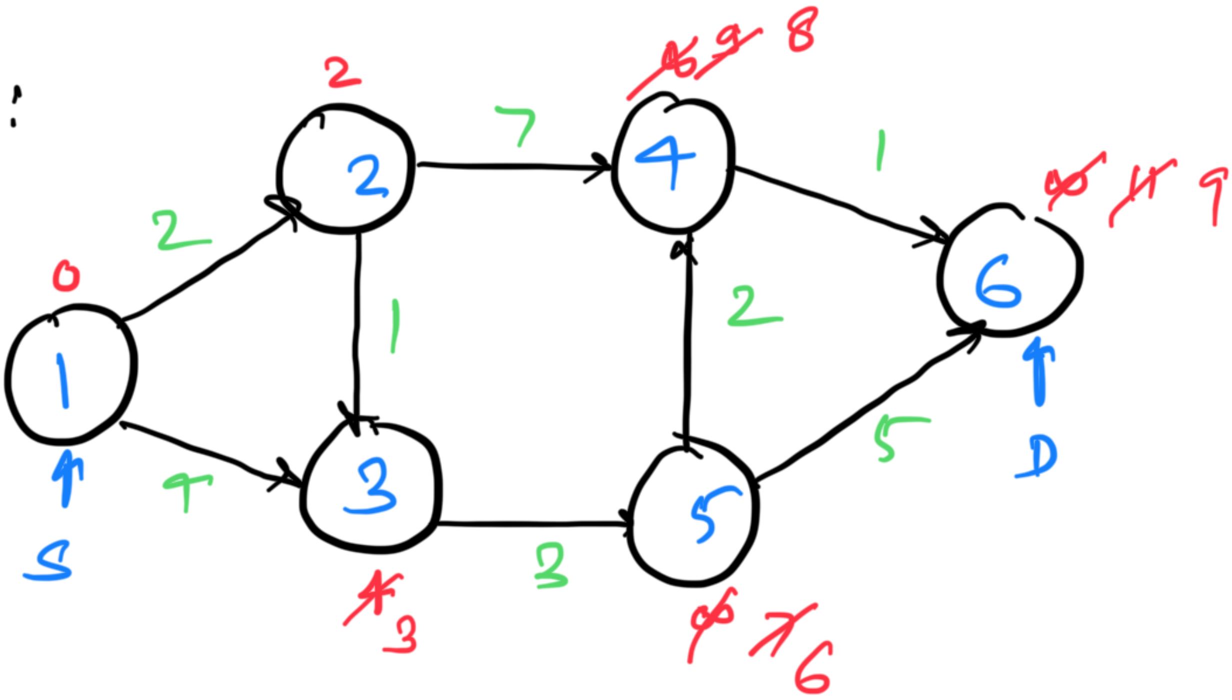
Relaxation

$$\text{if}\left(d[u] + c[u,v] < d[v]\right)$$

updation

$$d[v] = d[u] + c[u,v] \rightarrow \Big\} \text{ updation}$$

Ex:



v	d[v]
1	0
2	2
3	3
4	8
5	6
6	9

Time Complexity

$$n = |V| \times |V|$$
$$= n \times n$$

↑ vertices ↑ n vertices relaxed

$$\approx O(n^2)$$

Space → $O|V|$ ⟶ $O(n)$

↓ visited nodes

Drawback ⇒ non - negative

- Negative weighted edge ⇒ Bellman Ford

Time Complexity ⇒ $O(V+E)$

Space Complexity ⇒ $O(|V|)$