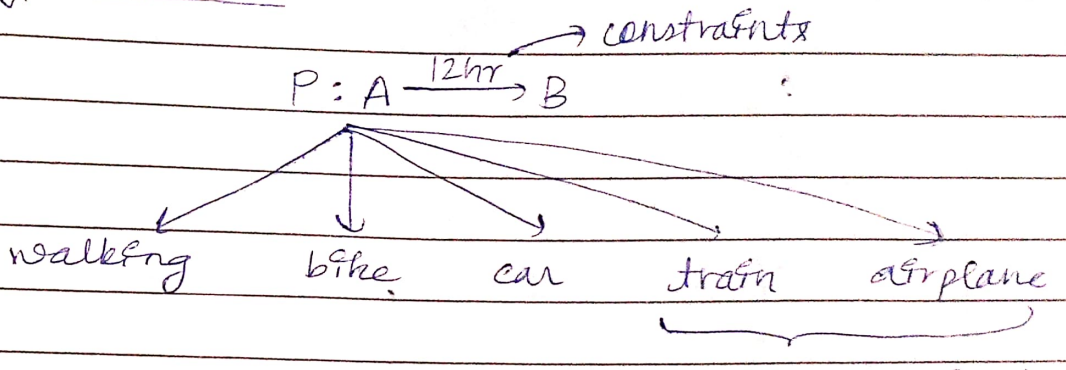
Greedy Method:



The problem which requires either minimum result or maximum result is called optimization problem.

Solutions which satisfy the constraints are called feasible soln. & best of them is optimal soln.

Strategies used for solving optimization problems:

1) Greedy method
2) Dynamic programming
3) Branch & Bound

Greedy method: 1) It says that a problem must be solved in stages
2) Consider one input from a given problem & if that input is feasible then we include it in the soln.
3) And then choose the best one of the feasible soln., that would be our optimal soln.

```
Algorithm Greedy (a, n){
    for i=1 to n do
    {
        x = Select(a);
        if Feasible(x) then
        {
            solution = solution + x;
        }
    }
}
```

Knapsack Problems

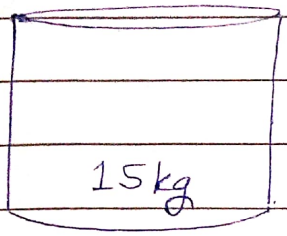
n = 7

Objects (O)	1	2	3	4	5	6	7
Profits (P)	10	5	15	7	6	18	3
Weight (W)	2	3	5	7	1	4	1
P/W	5	1.3	3	1	6	4.5	3

x (1 , 2/3 , 1 , 0 , 1 , 1 , 1)

x_1 x_2 x_3 x_4 x_5 x_6 x_7

$0 \leq x \leq 1$



15 - 1 = 14	3 - 1 = 2
14 - 2 = 12	2 - 2 = 0
12 - 4 = 8	
8 - 5 = 3	

Constraint:

$\sum x_i w_i \leq m$

Objective:

$\max \sum x_i p_i$

$$\sum_{i=1}^{7} x_i \times w_i = 1 \times 2 + \frac{2}{3} \times 3 + 1 \times 5 + 0 \times 7 + 1 \times 1 + 1 \times 4 + 1 \times 1$$

$$= 2 + 2 + 5 + 1 + 4 + 1$$

$$= 15$$

$$\sum_{i=1}^{7} x_i \times P_i = 1 \times 10 + \frac{2}{3} \times 5 + 1 \times 15 + 1 \times 6 + 1 \times 18 + 1 \times 3$$

$$= 54.6$$

Time complexity = $O(N * \log N)$

Space complexity = $O(N)$

Job Sequencing with Deadlines

$n = 5$

Jobs	J_1	J_2	J_3	J_4	J_5
profits	20	15	10	5	1
deadlines	2	2	1	3	3
			×		

0 J_2 1 J_1 2 J_4 3

Time complexity = $O(N^2)$

Space Complexity: $O(N)$

First we see the job with highest profit which is J_1 & put it just before its deadline, then we come at J_2 & as 1→2 slot is filled we put it to the left of J_1

Now we come at J_3 but no slot is left for it so cancel it

& put J_4 in the next slot

$20 + 15 + 5 = 40$

∴ Ans. $\{J_2, J_1, J_4\}$

∴ Seq. = $J_2 \rightarrow J_1 \rightarrow J_4$ or $J_1 \rightarrow J_2 \rightarrow J_4$

Job consid.	Slot origin	Solution	Profit
			0
J_1	[1, 2]	J_1	20
J_2	[0, 1] [1, 2]	J_1, J_2	20+15
J_3 ×	[0, 1] [1, 2]	J_1, J_2	20+15
J_4	[0, 1] [1, 2] [2, 3]	J_1, J_2, J_4	20+15+5
J_5 ×	[0, 1] [1, 2] [2, 3]	J_1, J_2, J_4	40

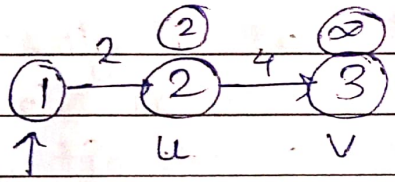
$n = 7$

Jobs	J_1	J_2	J_3	J_4	J_5	J_6	J_7
profits	35	30	25	20	15	12	5
deadlines	3	4	4	2	3	1	2

0 J_4 1 J_3 2 J_1 3 J_2 4

20 25 35 30 = 110

Dijkstra's Algo:



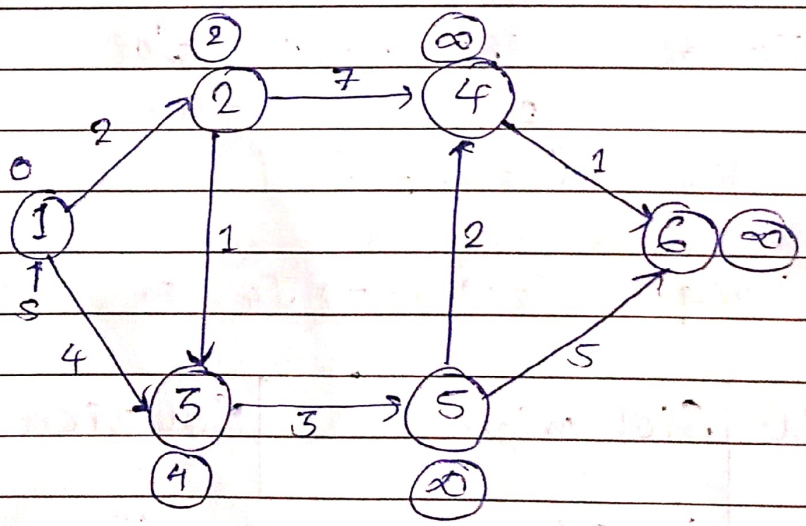
$2 + 4 < \infty$

Relaxation:

if $[d[u] + c[u,v] < d[v]$

$d[v] = d[u] + c(u,v)$

Q:



→ First assign resp. weights if we have a direct path else if not direct path assign weight as ∞

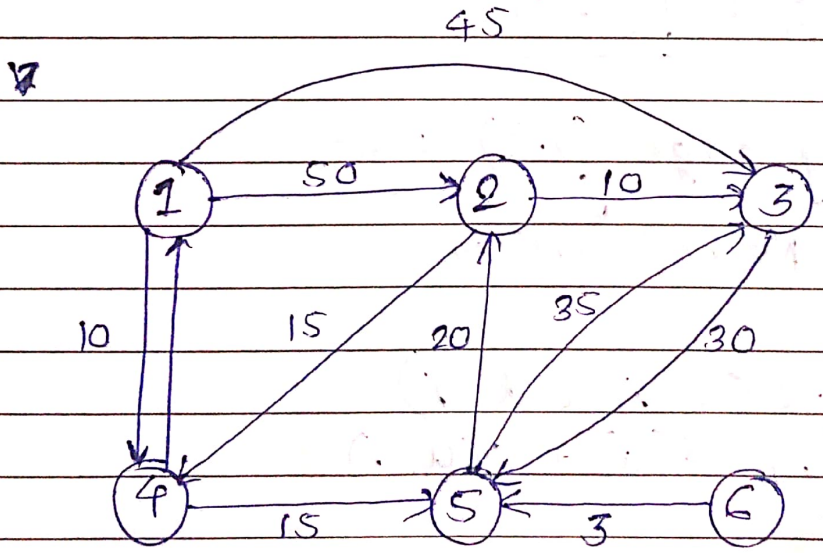
→ Repeat these steps till you solve the problem
→ Select the shortest path of ~~already~~ the ~~assign~~ direct path from given vertex
→ perform relaxation now i.e. check the connected nodes & update their distances if they are less than current distance

Worst case T.C. : $\Theta(n^2)$ or $\Theta(|V|^2)$

91

Selected vertex	2	3	4	5	6
1	(2)	4	∞	∞	∞
1,2	[2]	(3)	9	∞	∞
1,2,3	[2]	[3]	9	(6)	∞
1,2,3,5	[2]	[3]	(8)	[6]	11
1,2,3,5,4	2	3	8	6	9

V	d[v]
2	2
3	3
4	8
5	6
6	9



Sel. vertex	2	3	4	5	6
4	50	45	(10)	∞	∞
5	50	45	(10)	(25)	∞
2	(45)	45	(10)	(25)	∞
3	(45)	(45)	(10)	(25)	∞
	(45)	(45)	(10)	(25)	∞

V	d[v]
2	45
3	45
4	10
5	25
6	∞

Drawbacks of Dijkstra's Alg.

(1) Doesn't work of negative edges
(2) at all times, sometimes works sometimes doesn't.

BELLMAN FORD

→ Relax every edge (V-1) times
↓
No. of vertices

→ if (d[u] + c(u,v) < d[v])
d[v] = d[u] + c(u,v)

Be careful,
Negative edges are ok
Negative cycles are not

Idea: For increasing lengths K, compute shortest paths with atmost K edges

Observation: Shortest path can have atmost $n-1$ edges. (n = no. of vertices)

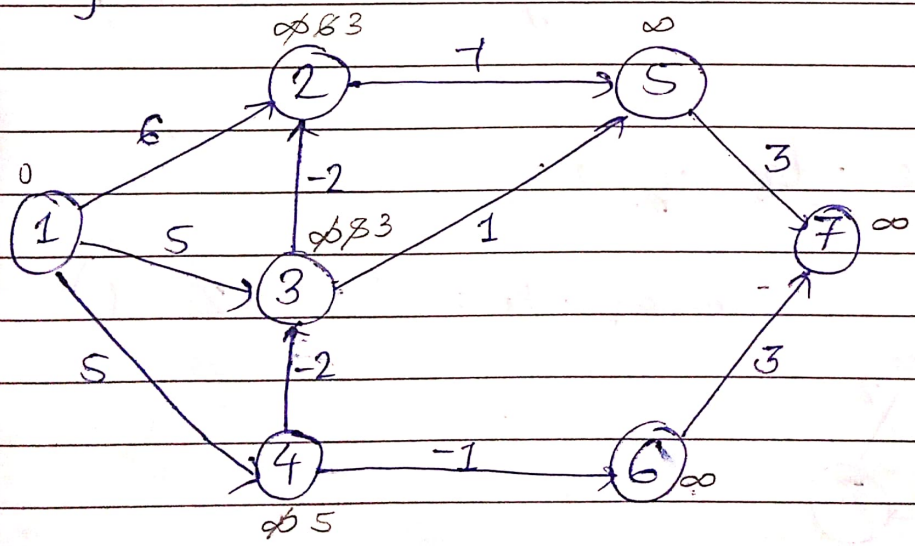
Recurrence: Path of length K from v to u breaks up as.

- Path of length K-1 from v to some neighbour w of u
- Edge from w to u

$d^k[u]$ → shortest path from v to u using atmost k edges

$$d^k[u] = \min\left(d^{k-1}[u],\ d^{k-1}[w] + cost[w,u] \text{ for each neighbour } w \text{ of } u \right)$$

E.g.



Vertices

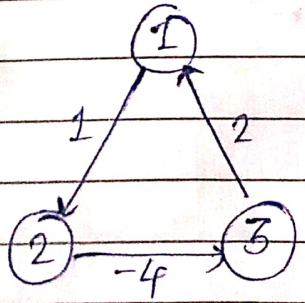
Path length k	1	2	3	4	5	6	7
0	0	∞	∞	∞	∞	∞	∞
1	0	6	5	5	∞	∞	∞
2	0	3	3	5	5	4	∞
3	0	1	3	5	2	4	7
4	0	1	3	5	0	4	5
5	0	1	3	5	0	4	3
6	0	1	3	5	0	4	3

Complexity : $O(n^3)$ → Adj. matrix
$O(nm)$ → Adj. list

Optimizations

→ Maintain a queue of those vertices whose distance reduces in iteration k.
→ Only need to update neighbours of these vertices in iteration k+1.
→ Terminate if no change observed for some k.

Negative Edge cycle:



Sum of wts. = 1 + 4 + 2 = -1

Minimum Cost Spanning Tree

PAGE NO.:

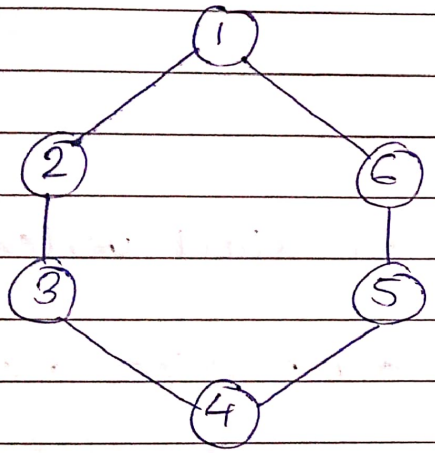
$G = (V, E)$ $V = \{1, 2, 3, 4, 5, 6\}$

$E = \{(1,2), (2,3), (3,4)\}$

Spanning tree: A sub-graph of a graph which contains all the vertices of the graph & having $V-1$ edges.
No cycles.

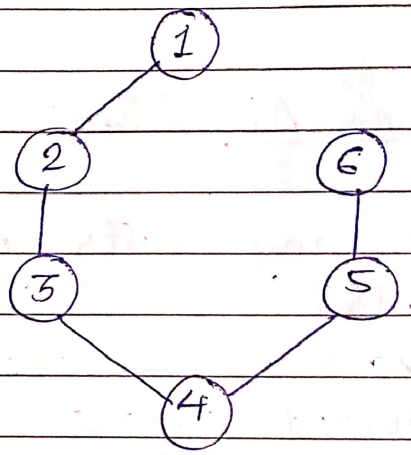
E.g.

Graph



G

Spanning tree



S

$S \subseteq G$ $S = (V', E')$

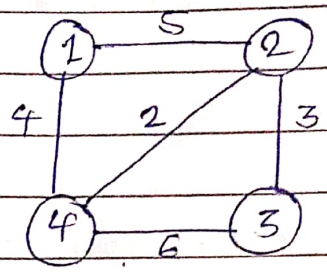
$V' = V$ $|E'| = |V'| - 1$

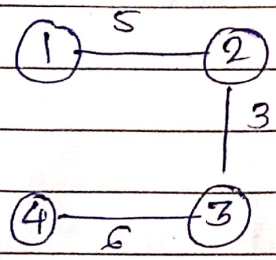
Ng. of spanning trees possible $= {}^{|E|}C_{|V|-1}$ − ng. of cycles

TC: $O(V^2)$ → for adj. matrix
$O(E * \log V)$ → adj. list + binary heap

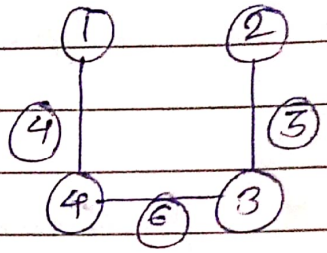
SC: $O(V)$

Graph

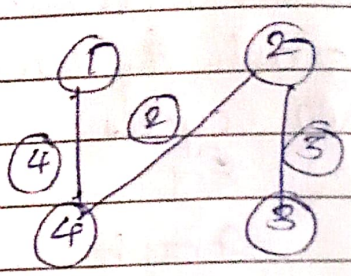




Cost = 14



cost = 13



Cost = 9

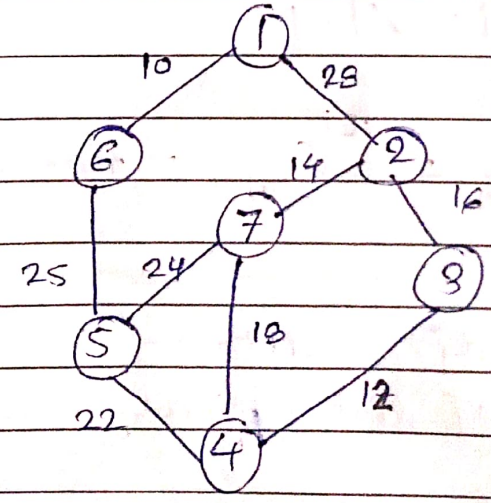
Prim's Algorithm : (Vertex)

→ Take any arbitrary vertex as start vertex of MST

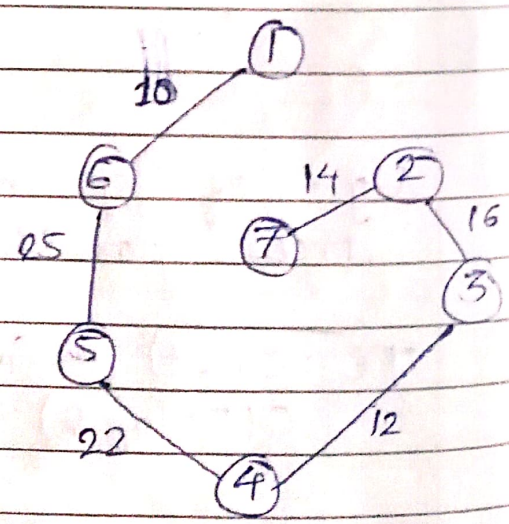
→ Follow next steps until all vertices aren't included in MST

→ Take the minimum of the connected edges & add it to MST if it doesn't form a cycle

E.g.



G



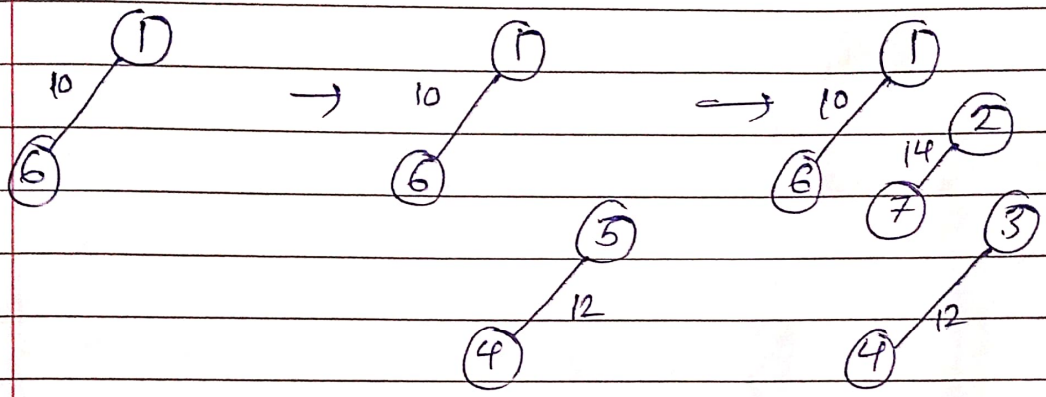
Cost = 99

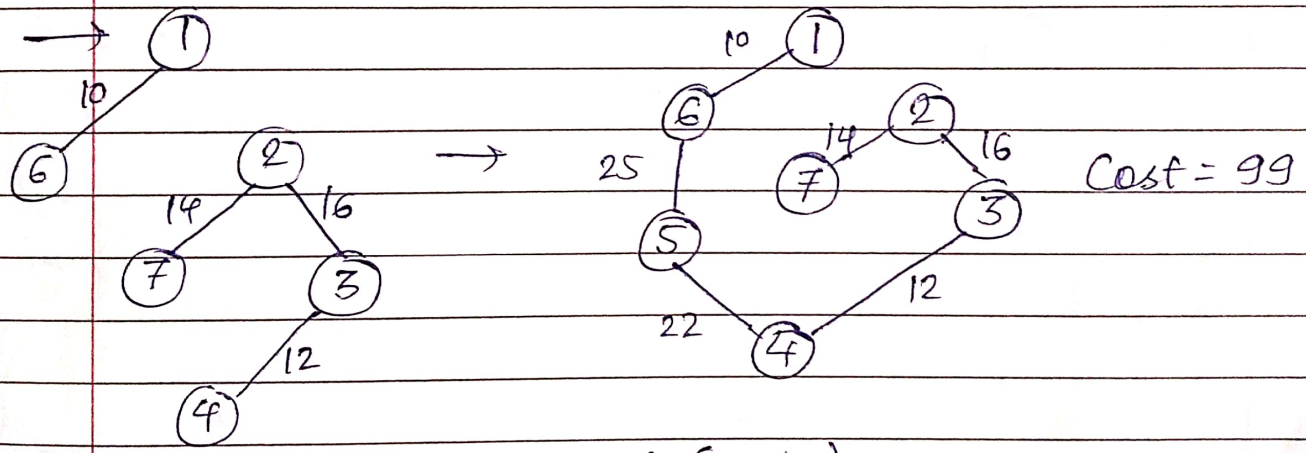
MST

Ka Kruskal's Algorithm (Edges)

→ Sort all edges in inc. order of weight
→ Repeat the step below until there are V-1 edges in the graph
  → Take the smallest edge which doesn't form a cycle

Eg. Same graph





T.C. = $\theta(|V|+|E|)$

Min-Heap:

T.C. = $\theta(n \log n)$ or $\theta(E \log V)$

S.C. = $O(V+E)$