Knapsack Problem

$n = 7$
$m = 15$

Object : O	1	2	3	4	5	6	7
Profits : P	10	5	15	7	6	18	3
Weights : W	2	3	5	7	1	4	1
P/W	5	1.3	3	1	6	4.5	3

constraints

$\sum x_i w_i \leq M$

objective

max. $\sum x_i p_i$

→ Profit → Maximum optimization problem

→ constraints → total object included in the bag
The weight less or equal to 15 kg

→ we can solve it in various object

→ we can take it ~~all~~ one object, and take it, it's not ~~optimal~~ maximal

→ we can take all, all can not fitted in to the bag.
It is not feable, so it is not optimal

→ We need maximum result.

For this problem, we have many solutions, but which solution given the maximum result, it is an optimal soln of the problem.

→ include the object in to the Bag.

Object x ($\overset{0-1}{x_1}$ $\overset{2/3}{x_2}$ x_3 x_4 x_5 x_6 x_7)

x can be $0 \leq x \leq 1$,

	x_1	x_2	x_3	x_4	x_5	x_6	x_7
	2	6	4	1	1	3	5
	1	2/3	1	0	1	1	1

We can take the value of fractions.

→ $15 - 1 = 14$ kg ①
$14 - 2 = 12$ " ②
$12 - 4 = 8$ " 3
$8 - 5 = 3$ " 4
$3 - 1 = 2$ 5
$2 - 2 = 0$ 6

→ Knapsack soln are devisible

* → Take first the highest ~~case~~. profit

→ But small object will be given more profit

→ Finds the highest profit by weights.

Calculate the ~~vatn~~ ~~profit~~ weight

$\sum x_i w_i = 1*2 + 2/3*3 + 1*5 + 0*7 + 1*1 + 1*4 + 1*1$

$= 15$ (weight)

<u>Profit.</u>

$= 1 \times 10 + 2/3 * 5 + 1*15 + 1*6 + 1*18 + 1*3$

$\sum x_i p_i = 54.6$

27.29

NP, P, NP-Hard, NP-Complete

Algorithm

Polynomial time | exponential time

p class
1. Linear Search - $O(n)$
2. Binary Search - $O(\log n)$
3. Insertion Sort - $O(n^2)$
4. Merge Sort → $O(n \log n)$

1. 0/1 Knapsack - 2^n
2. Sum of Subset - 2^n
3. Graph cloning - 2^n
4. Travelling sale man - 2^n

if. $n = 7$, consider the Insertion Sort

$n^2 = 49$.

if $n = 2^7 - 128$

* we can see, Polynomial time is faster than exponential.

P-class - A problem can be solved in polynomial time.

NP-class :- (Non-deterministic polynomial time) - A problem which can not be solved in polynomial time but can be verified by polynomial time.

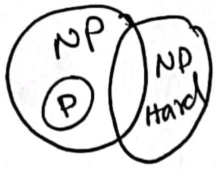
```
NSearch (A, n, key) {
          ↳ no of arrays
              ↳ no of elemen
                  ↳ key search → 3
non-deterministic// j = choice();
    if (A[i]==key) {
        write(i);
        sucess();
    }
    write (0);
    failure();
}
```

A	5	6	7	8
	1	2	3	1

key = 7.

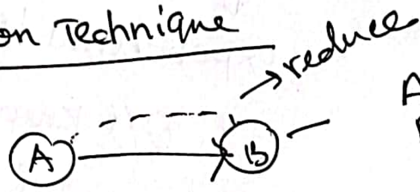
~~NP-Hard~~

~~NP~~



(*) → If we reduce the time by polynomial time then we ~~can~~ call it's NP Hard (*) can defi..

Reduction Technique



A, B } belongs to NP.

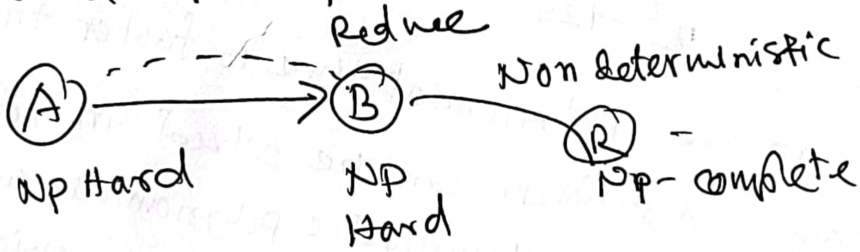
→ A reduce to B.

→ A reduce to B, if the problem of A algorithm solved by the polynomial time.

→ Reduce time always in a polynomial time.

(*)

(*) The problem of NP is ~~not~~ reduced by polonial then we it's called NP-Hard



→ If A is reduced by Np Hard then we can say B is also NP Hard. if A is solved by polynomial time, B is also solved by polynomial.

→

We can define two classes -

P → P is a set of those deterministic algorithms which are taking polynomial time

~~NP~~

ex- ....

NP - These algorithms are nondeteministic but they take the polynomial time for exponential time.

CNF - Satifiability — 2^n.-

$x_i = \{x_1, x_2, x_3\}$

$$CNF = \underbrace{(x_1 \vee \bar{x_2} \vee x_3)}_{c_1} \wedge \underbrace{(\bar{x_1}, \vee x_2 \vee \bar{x_3})}_{c_2}$$

↳ Propositional claus

= disjunction

disjunction

Having with conjunction

→ satifiability problem is to find out for what values of x_i, and above formula is true.

→ what is the possible value of x_i

→ place then the value of x_i and check for which of these value is true.

→ we should find out all possible values for which it is true

→ How much time it take for solving?

→ How many value try?

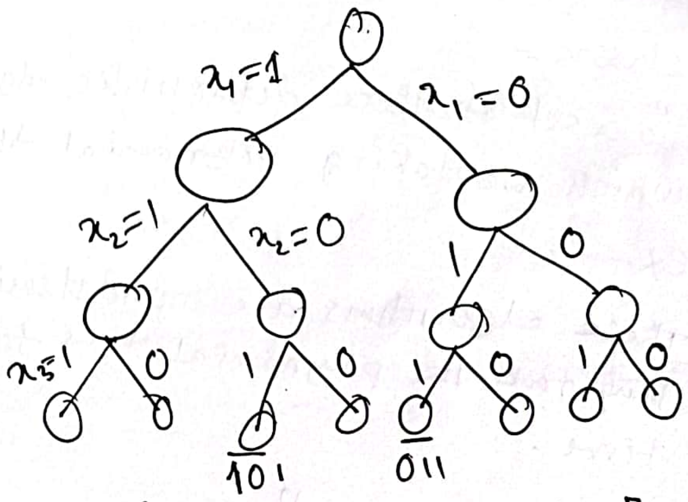
$$\boxed{2^3 \Rightarrow 2^n}$$

exponential time taking problem?

x_1	x_2	x_3
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	1	0
1	1	1

8-possibilities

Similar to exponential time.



→ The path from root to the leaf of this tree gives a soluntion.

→ if ~~sta~~ satistied satifiability solved in polynomial time ~~101~~ all can be solved in polynomial time.

→ or any of them is solved then all of them can be solved in polynomial time.

0/1 Knapsack

$P = \{10, 8, 12\}$ $\quad n=3$

$W = \{5, 4, 3\}$ $\quad m=8$

$x_i = \{011, 011, 011\}$.

x_1	x_2	x_3
0	0	0
0	0	0

⋮

$2^3 = 2^n$.

Longest Common Subsequence
Aynamic Programming

→ Graph
→ Depth-First-Search
→ Breadth-First-Search
→ Topological Sort
→ Strongly Connected Component

→ Greedy Graph Algorithm
→ Minimum Spanning Tree
→ Prim-Jarnik Algorithm
→ Kruskal's Algorithm.

* Dynamic programming *
Longest Common Sub-sequence (LCS)

1. LCS using Recursion
2. LCS using Memorization
3. LCS using Dynamic Programming

There are two strings as

string1 → a b c d e f g h i j

string2 → c d g i

→ There are different characters are matching with string1.
→ This is a sequence cdgi, that is there is string1. dgi
→ They are in same order, & same sequence
→ so, cdgi is the longest common subsequence.
→ what is longest?

Another example to change some sequence.

String1: a b c d e f g h i j

String2: e c d g i

Let us find the subsequence, when you find the matches, make sure it ~~to~~ follows the sequence. these matches should not intersect with others.

→ That means, e ~~appea~~ has appeared here
c should following that e. not at the backside of e.
d is not allow

Once subsequence we got it -

egi. → this is also subsequence

cdgi → also subsequence.

this is longest subsequence

Let us take another example -

String1: a b d a c e

String2: b a b c e

b a c e } two subsequence,
a b c e } which is same "

→ The problem is finding whether the set of characters in these two strings are matching or not

→ The matching character need ~~to~~ not be continious.

LCS (Recursive)

A | b | d | \0 | m
0 1 2

B | a | b | c | d | \0 | n
0 1 2 3 4

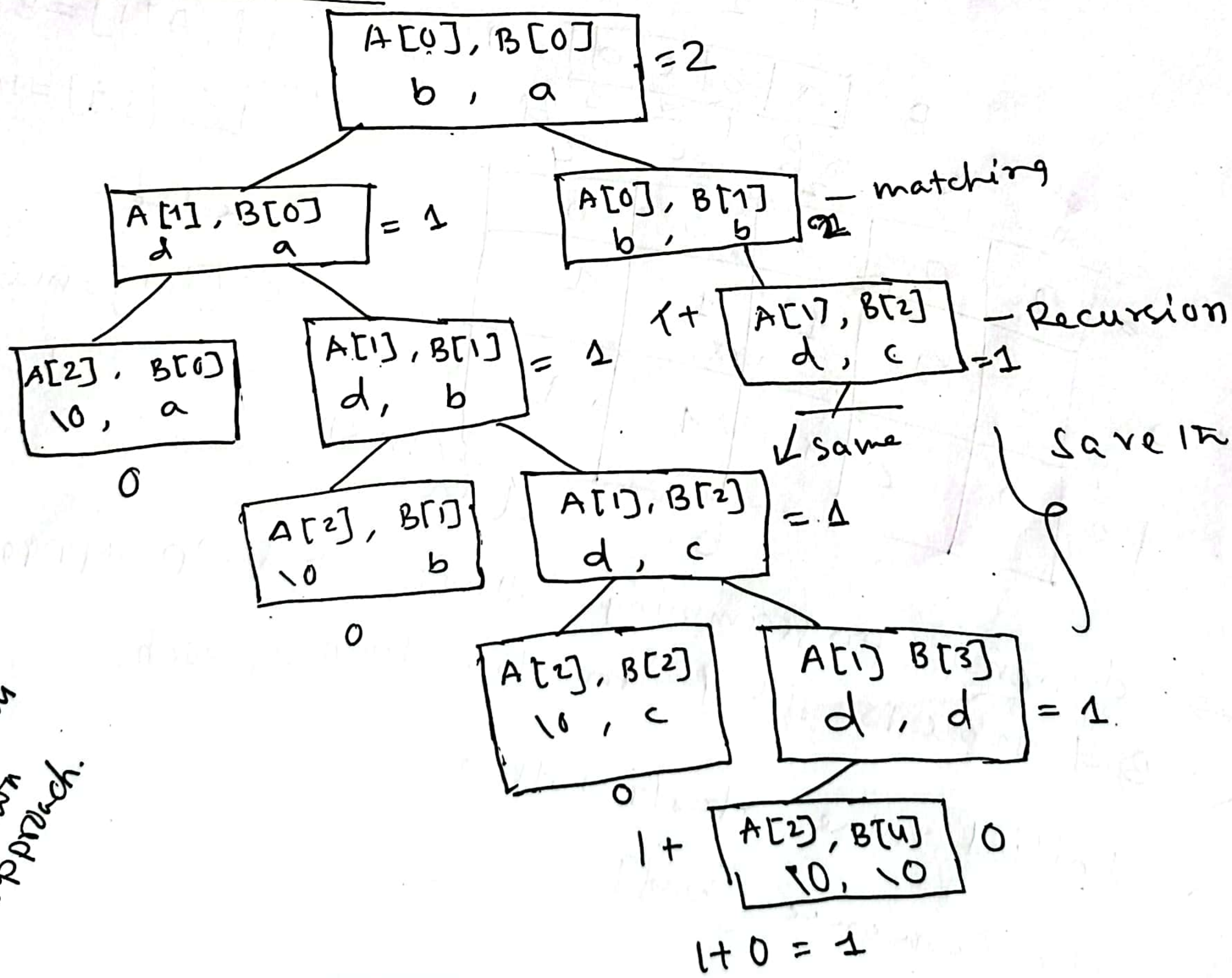
```
int LCS (i, j) {
  if (A[i] == '\0' || B[j] == '\0')
     return 0;
  else if (A[i] == B[j])
     return 1 + LCS(i+1, j+1);
  else
     return max (LCS(i+1, j),
                 LCS(i, j+1));
}
```

Parameter i, j works starting indices of A and B. string.

if matches move both A and B string.

if doesnot match, check either move first string A or move alphabet of second string B.

Tracing Algorithm



TOP - bottom
TOP-down approach.

→ Final result is 2

Memorization is called recursion, reduce the time.

(m) → 1st string, (n) → 2nd string

	a (0)	b (1)	c (2)	d (3)	\0 (4)
0 b	2	2			
1 d	1	1	1	1	
2 \0	0	0	0	0	

→ bottom toward top

First string

$O(m \times n)$ time for solving recursive algorithm.

$\boxed{2^n \to m \times n}$

→ Colmn are representing the 2nd string

→ Rows are representing the first string

→ Memorization reduce the no of calls.

→ How many calls are made.

LCS using dynamic programming

A | b | d | m
(1, 2)

B | a | b | c | d | n
(0, 1, 2, 3)

	0	a 1	b 2	c 3	d 4
0	0	0	0	0	0
b 1	0	0	1	1	1
d 2	0	0	1	1	2

b d $O(m \times n)$

if (A[i] == B[j])

LCS[i, j] = 1 + LCS[i-1, j-1].

else

LCS[i, j] = max (LCS[i-1, j], LCS[i, j-1])

→ dynamic programming follows bottom-up approach.

→ But table is filled from top-down appoach.

→ Initial value starts with 0.

→ The sequence [b d]

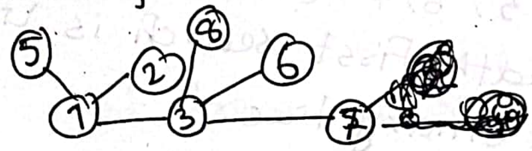
Graph

Depth-First-Search & Breadth-First-Search

→ BFS & DFS are graph travers method.

For these two search method we can use two terms

1. visiting a vertex
2. Explorating of verte.



This a ~~gr~~ tree. A tree is also a graph.

→ visiting a vertex means going on a particular vertex.

→ Exploration means we are on particular vertex then visiting all its adjacent vertices.

→ Based on two terms we will find the traversals.

First consider the BFS → 1, 3, 5, 2, ~~7, 8, 6, 4~~ 7, 8, 6 (we can take any order)

DFS → ~~1, 3, 7, 8, 6, 5~~ 1, 3, 7, 6, 8, 2, 5

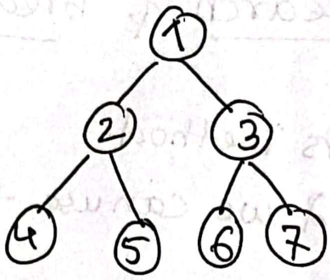
↳ there is nothing exploring in 7.

→ In BFS, we explore a vertex then go to the next vertex.

DFS, once we started exploring visited, we will suspend this vertex and start its exploration.

→ We got ~~from~~ 1 to 3 then we got 7, then we'll start explorng 7. Like this

Another example with binary tree



→ Tree is a graph. Let us perform for different search

→ As per binary tree, we can perform it level order 1

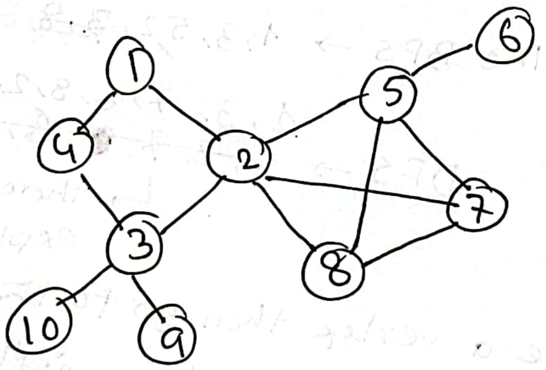
BFS : 1, 2, 3, 4, 5, 6, 7 (Level order)

↳ Breadth First search is like a level order on a binary tree.

DFS → 1, 2, 4, 5, 3, 6, 7.

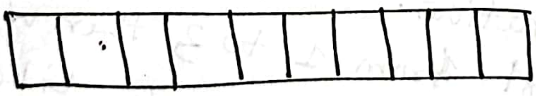
↳ BFS is like pre-order traversal of a graph.

Let us take another graph



For performing BFS we consider data structure that is Queue.

Q ⇒



→ Initial step

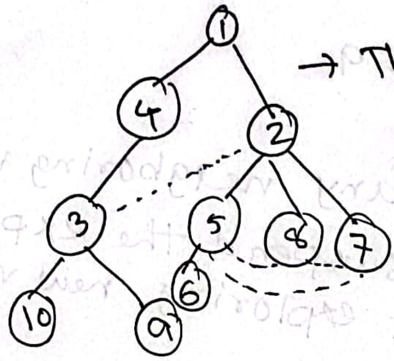
→ Repeating step

→ start exploration from any one of the vertex.

Q =

1	4	2	3	5	8	7	10	9	6	

BFS → 1, 4, 2, 3, 5, 8, 7, 10, 9, 6



→ This called BFS Spanning Tree.

→ We have started from any vertex

→ when exploring any vertex then can visit a the adjacent vertices in any order

→ If we select a vertex, must visit all its adjacent vertices

→ then go to next vertices for exploration.

Check the validity of BFS

1) 1, 2, 4, 8, 5, 7, 3, 6, 10, 9 → These are valid.
2) 5, 2, 8, 7, 6, 3, 1, 9, 10, 4

DFS

→ stack as a data structure used for DFS.

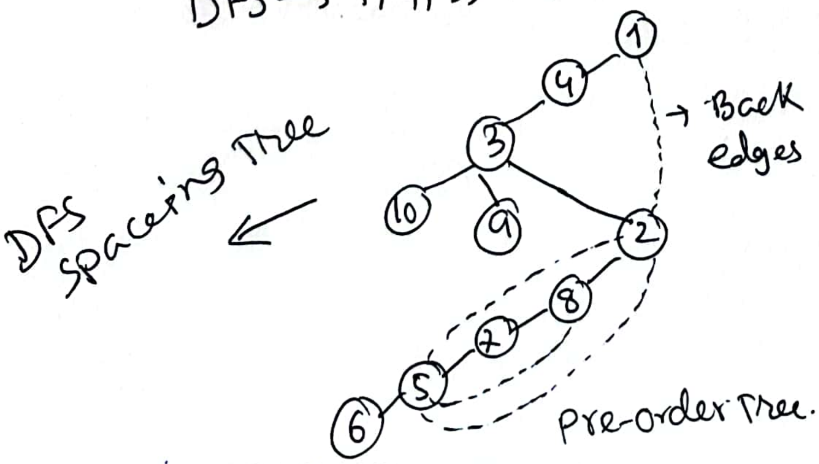
→ Start traversal from any vertex. (Initial step)

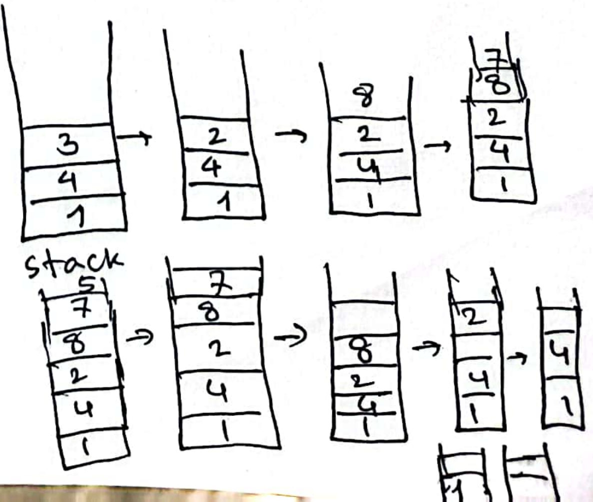
→ Repeating step

→ As new vertex is visited start exploring

→ Once visited one vertex, if one vertex is still remaining skip it.

DFS → 1, 4, 3, 10, 9, 2, 8, 7, 5, 6





Valid for DFS

1) 1, 2, 8, 7, 5, 6, 3, 9, 10, 4,

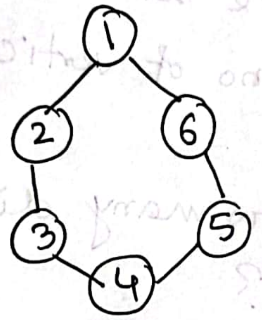
2) 3, 4, 1, 2, 5, 6, 7, 8, 10, 9

→ start from any vertex or any neigboring vertex

→ once visited a new vertex, suspend the exploration of current vertex and start exploring new vertex.

→ Time complexity for DFS and BFS is $\Theta(n)$

Minimum Cost Spanning Tree



$G = (V, E)$

$V = \{1, 2, 3, 4, 5, 6\}$ - set of vertices

$E = \{(1,2), (2,3), (3,4), \ldots\}$ → set of edges

G - Graps are represented as a set of vertices and edges

→ what is spanning tree?

↳ Spanning tree is a sub graph of graph

↳ subset of vertices and edges of subset or only one subset

↳ * Subset of edges

↳ But vertices must be as it is.

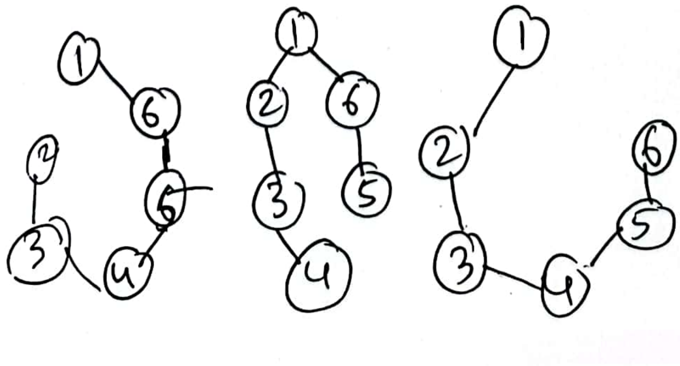
↳ Should be taken all vertices.

→ Spanning tree means, take all vertices

$|V| = n = 6$ → no. of vertices

$n-1 = 5$ → Five edges

→* Spanning tree is a subgraph of a graph having all vertices but only $(n-1)$ edges.



This is called spanning Tree

→ Tree is not a cycle, so here There is no cycle.

$S \subseteq G \longrightarrow$ S is a subgraph of Graph

$S = (V', E') \rightarrow$ S is define as set of vertices and edges.

$V' = V \rightarrow$ set of vertices are same as vertices of a graph

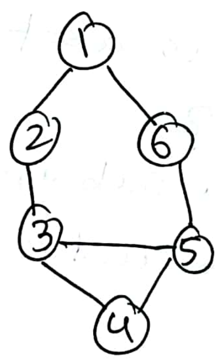
$|E| = |V| - 1 \rightarrow$ edges are no of vertices minus 1.

(*) For a given graph how many different spanning tree can be generated?

$|E| = 6$

${}^6C_5 = 6$

if any graph has more than cycle it consider the cycle.



$|E| = 6$

${}^7C_5 - 2 =$

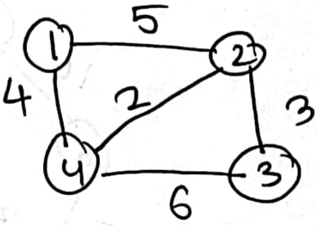
This is formula for given a graph to generate a different spanning trees are possible.

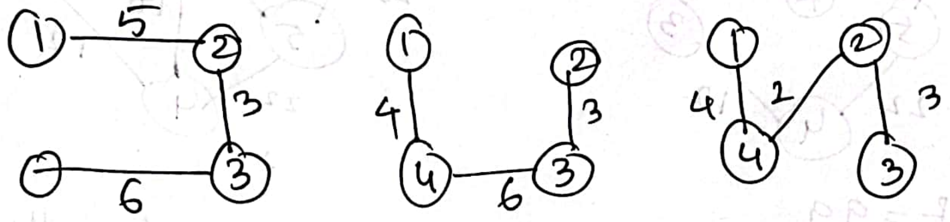
${}^{|E|}C_{|V|}$ - no. of cycle

$|V| \rightarrow$ no. of vertices

$|E| \rightarrow$ no of edges

Minimum ~~poss~~ Cost spanning Tree





Cost = 14   cost = 13   Cost = 9

→ In weighted graph, and get different spanning trees.
- The spanning trees may be varying.

→ Find the different cost of spanning Tree.

* Minimum spanning one which gives the minimum cost?
* ~~the~~ Find out the all possible spanning tree, and from that which ever is minimum.

→ But trying all possible is too Lengthy.

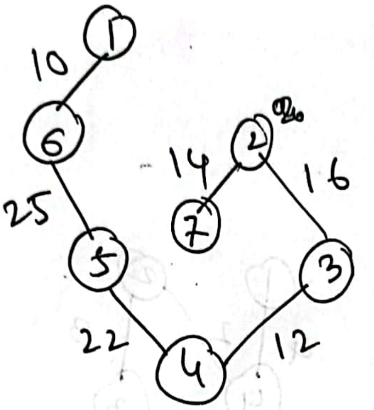
→ In that case, solve this problem using the greedy method

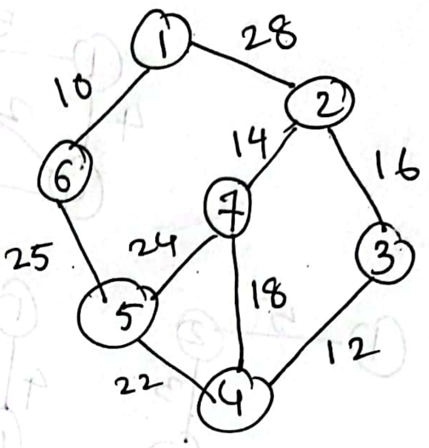
→ Greedy methods are finding the minimum cost spanning Tree

→ Using the Greedy method, don't need to find all possible spanning tree.

1. Prim's Algorithm
2. Kruskals Algorithm.

Prim's Algorithm





cost = 99

→ First, select a minimum weighted edge / smallest edge in the entire graph

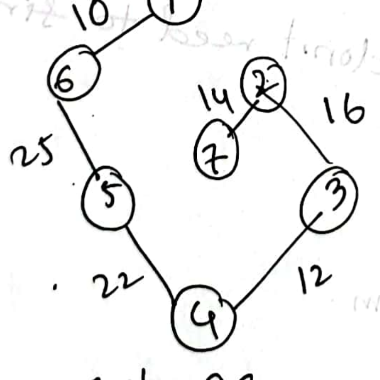
#

→ take next one, which is connected to all the reselected vertices

→ select next vertices which are connected to initial vertices.

→ Always select a minimum edge which should be connected.

Kruskal's Algorithm



→ Always select a minimum weighted edge.

→ In Kruskal's method, if it is forming a cycle don't include into the solutions.

→

cost = 99

Time $O(|V||E|)$ ⇒ no. of vertices.
⇒ no. of edges

$O(n \cdot e) = O(n^2)$