Dynamic Programming

→ Used largely to solve -optimisation problems in industries

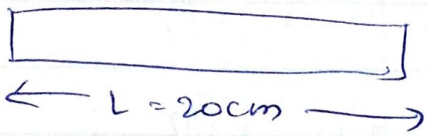
Ex:- Viterbi Algorithm
Function -tree algorithm
Bellman-ford algo
Dijkstra's Algo

Steps:-(Major)
1) You have to make a sequence of steps
2) Either maximise or minimise

→ Mostly used when we make a "sequence of decisions"

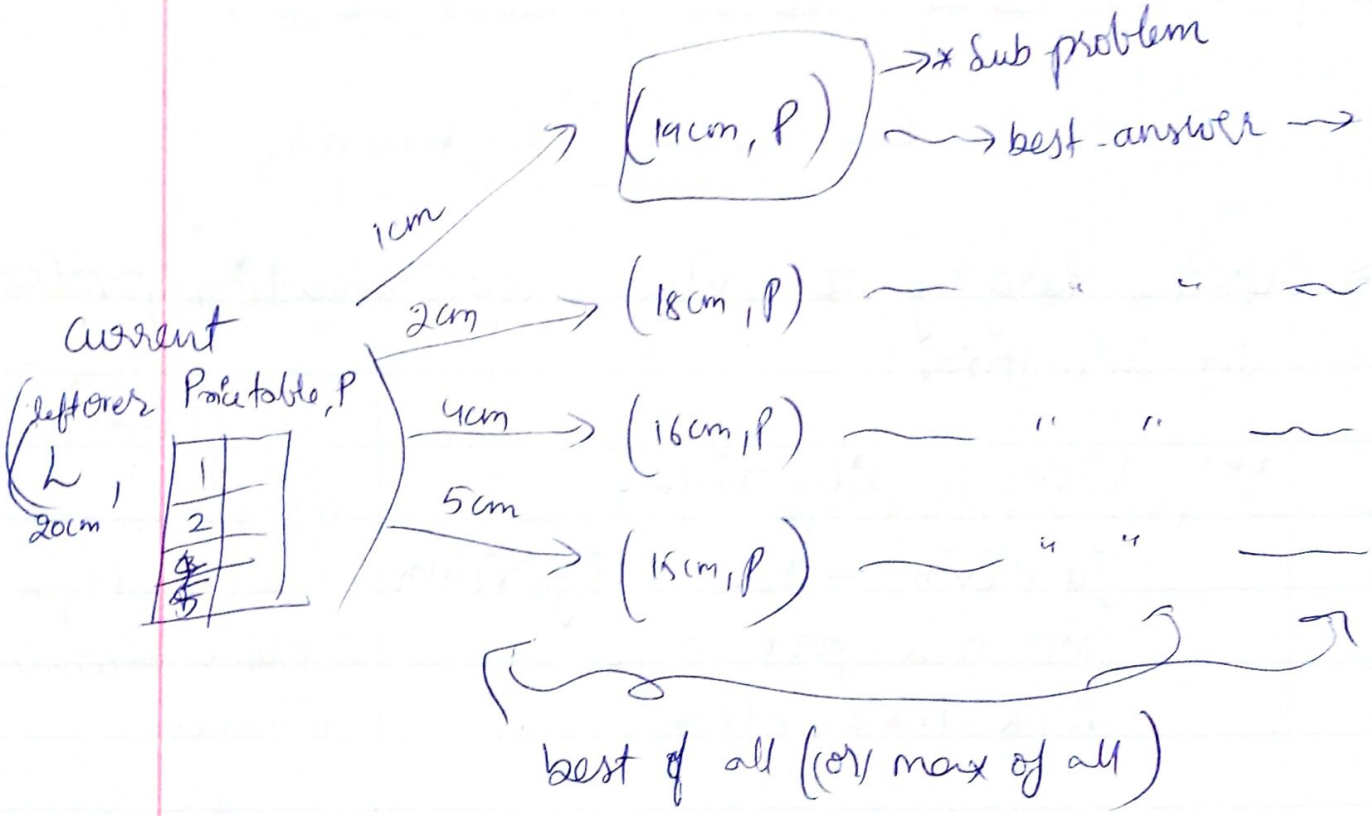
Rod-Cutting Problem:-

cm	₹/unit
1 cm	20
2 cm	28
4	36
5	45



Optimal Sub-structure:-

① First stage your decision (the first step) and consider left-over part-next



Revenue :-

for 1 cm path : 20¢ + BestRevenue (19, P)

for 2 cm path : 28¢ + BestRevenue (18, P)

for 4 cm path : 36¢ + BestRevenue (16, P)

for 5 cm path : 45¢ + BestRevenue (15, P)

) max is the best for 20 cm.

STEPS AFTER FINDING OPTIMAL SUB-STRUCTURE

1) Write a recurrence

2) Memoize the recurrence

3)

Rod-Cutting Pblm:-

1. Write a Recurrence:

base cases

$$max\_Rev(L,P) = max \begin{cases} 20 + max\_Rev(L-1,P) \\ 28 + max\_Rev(L-2,P) \\ 36 + max\_Rev(L-4,P) \\ 45 + max\_Rev(L-5,P) \end{cases}$$

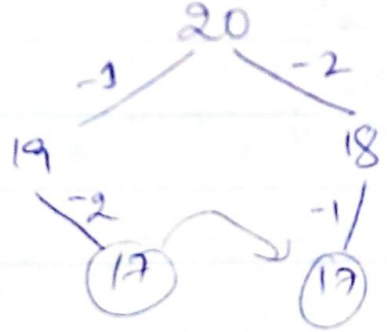
$maxRev(0,P) = 0$

$maxRev(L,P) = -\infty$ if $L < 0$

2) Memoizing:

① Top-Down- using the recursion. As you solve a problem, fill it in the hash table.

Ex:



→ hook it up, when you are solving a simpler problem.

③ Bottom-Up approach:-

→ Solve the problems of smaller size and fill it in the hash table (memoize).

→ Later run the for loop & solve the problem.

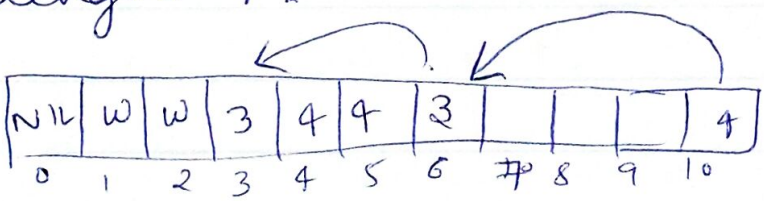
$$T[3] = \max \begin{cases} 0 \\ T[0]+1.8 = 1.8 \\ T[-1]+2 = -\infty \end{cases}$$

$$T[4] = \max \begin{cases} 0 \\ T[0]+2 = 2 \\ T[1]+1.8 = 1.8 \end{cases}$$

⋮ ⋮ ⋮ ⋮

→ To recover a solution, we write an another table, which is going to annotate the decision that is giving us the best solution

→ ① After filling up the decision table, look out for decision at $s[10]$, it gives to cut the rod of length 4.

② 

② Next, if you cut the rod of length "4", then you'll be left with 'rod of length '6'.

③ At '6', the decision is to cut the rod of length "3". If you cut the rod of length "3", then you'll be left with rod of length '3'.

4. Recover the solution:

Ex:

	cm	$
l_1	3	1.8
l_2	4	2

$L = 10$

$T[0] = 0$

$l_1, \ldots, l_n$ } natural numbers

L: Natural numbers

for $i = 1$ to L

$$T[i] = \begin{cases} \max \begin{cases} T[i-l_1] + p_1 \\ T[i-l_n] + p_n \end{cases} \\ 0 \end{cases}$$

$L = 10$

memo-table T = $-\infty$ | 0 | 0 | 0 | 1.8 | 2 | 2 | 3 | | | | 4 |

(optimal cost to go)

<0 0 1 2 3 4 5 6 7 8 9 10

decision S = = | | waste | waste | 3 | 4 | 4 | 3+1 | | | | 5+1 |

$$T[1] = \max \begin{cases} 0 \\ T[-2] + 1.8 = -\infty \\ T[-3] + 2 = -\infty \end{cases}$$

~~R C (1, P)~~

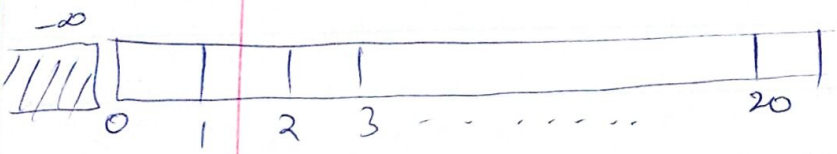
① Rod-cutting(1, P)

② Rod-cutting(2, P)

③ Rod-cutting(4, P)

④ Rod-cutting (5, P)

→ memoize table:



for $L = 1$ to L

$$T(L) = \max \begin{cases} 20 + T(L-1) \\ 28 + T(L-2) \\ 36 + T(L-4) \\ 45 + T(L-5) \end{cases}$$

4. At '3', the decision is to cut the rod of length '3'.

5. So, finally the decision is [4 → 3 → 3]

omsairam omsrivenkatesayanamaha ommahaganadhipatayanamaha omnamaha shivayaha.

Greedy Algorithms

→ locally optimal choice
→ myophic (blind) choice

Coin Changing Problem :-

1, 5, 10, 25, 50 → cent coin's

92-cents = ? # smallest no. of coins possible

= 1×50 + 1×25 + 1×10 + 1×5 + 2×1 ⇒ 92 ¢

Ex 2 : If you have denominations like

1, 14, 25

28¢ = ?

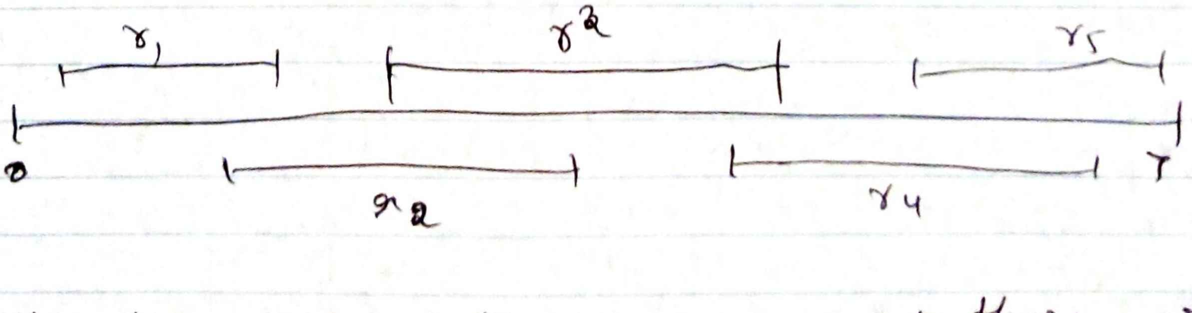
Greedy algorithm will choose,

25¢ + 3×1¢ = 4 coins,

but we can do it 2 coins

greedy algo fails,

Room Scheduling Algorithm



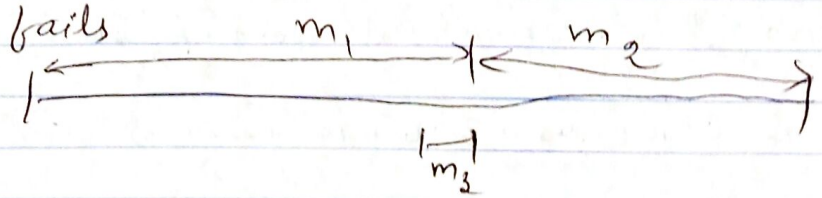
We have a meeting room, and there are requests to the meeting room, but the ti slots overlap.

But we can accomodate only one request at a time.

→ We should make sure that, maximum # of meetings take place.

Choices:

① Shortest meeting first:-



→ In the above case, you can schedule only one meeting. But we could have done '2' meetings.

2) S.M.F is sub-optimal.

② Earliest Ending meeting first:-

Greedy Algorithm - Continuation:

1) Huffman Code:

* To compress the information and use less memory, we encode characters.

* There are different types of encoding patterns (or) standards

Ex:- ASCII.

x In ASCII (American Standard for Information Interchange), every character is given a hexadecimal - number.

a - Z, A - Z

Ex:- a - 0x40 ;
b - 0x41 } 8 bits

So, if we have 10,000 characters, we use 80000 bits.

* To compress the data, We can use "small length" codes for most repeating chars and "large length" codes for less frequent chars.

So that we can use less memory

→ For example, a, e → repeats more in english characters, so, assign bits of small length to a, e, and for less repeating characters assign bigger length bits.

e := 1
a := 01
z := 00000000 11111111

But, Decoding = ?

→ Coding & Decoding made easy in Prefix Codes:

Prefix Codes:-

Ex:- a, f

Code(a) cannot be a proper prefix of codeward (f)

↓ (01) any character — any character

Proper - Prefix = ?

b_1 : 0001
b_2 : 000

} : "b_2" is proper prefix of "b_1"

→ ASCII codes are prefix codes

Ex:-

x : 00001

y : 0000

→ We can arrange code-words in the form of a tree in "Pre-fix" Codes

(a, b, c, d, e)

Ex:-

a : 01

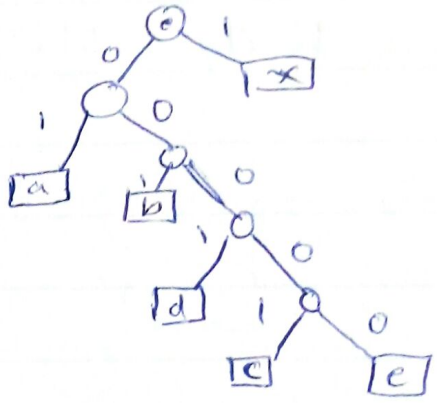
b : 001

c : 00001

d : 0001

e : 00000

decision tree



Ex:-

00001 0 1 001 1 0 1 1 0 1 0 1 0 1 0 1

c a a x x a x a x a x a

→ ASCII is a 8 bit prefix code

↓

can be stored in "Hash Table"

Optimal Pre-fix Code :-

If we have document containing, 10^6 chars containing

a, b, c, d, e

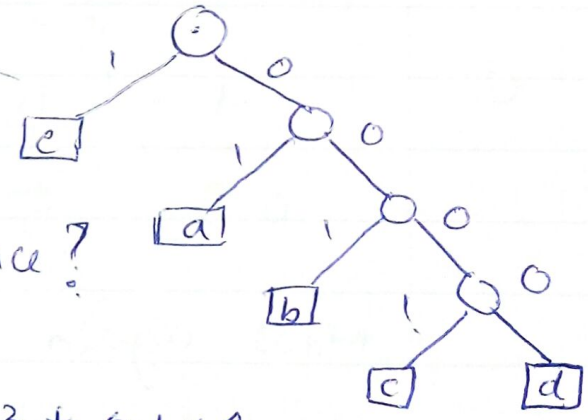
0.3, 0.2, 0.1, 0.05, 0.35 → fractions of occurence

a : 000
b : 001
c : 010
d : 011
e : 100

} If we use '3' bits, we need 3×10^6 bits of space.

→ considering above occurences,

using this code



10^6 bits document size requires how much space?

$= 0.35 * + 0.3 \times 2 + 0.2 \times 3 + 0.1 \times 4$

$+ 0.05 \times 4$

$= 2.15$ bits per character.

H.W :- Design probabilities of chars, that gives even worse compression

Huffman Code :- (Going Bottom Up)
↳ Optimal

a	b	c	d	e
1	1	1	1	1
0.3	0.2	0.1	0.05	0.35

① Take the two least frequent chars & make a sub-tree :

(combine two chars & make a composite character & assign frequency by adding freq. of these two)

Tree -

"cd" → composite word/char

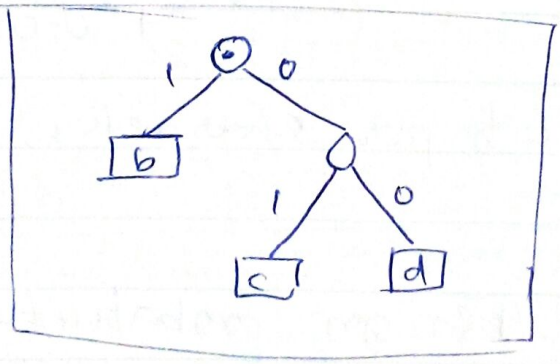
↓

0.1 + 0.05 = 0.15

②

a	b	"cd"	e
1	1	1	1
0.3	0.2	0.15	0.35

Now again you take, two least frequently occuring char



"bcd"

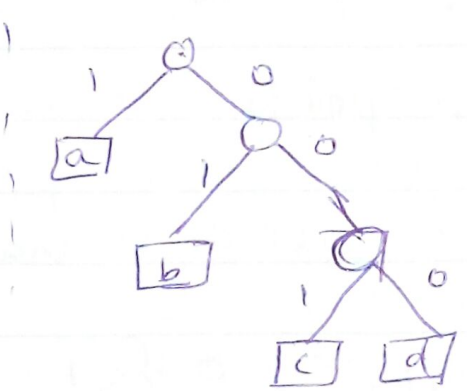
↓

0.2 + 0.15

= 0.35

(3)

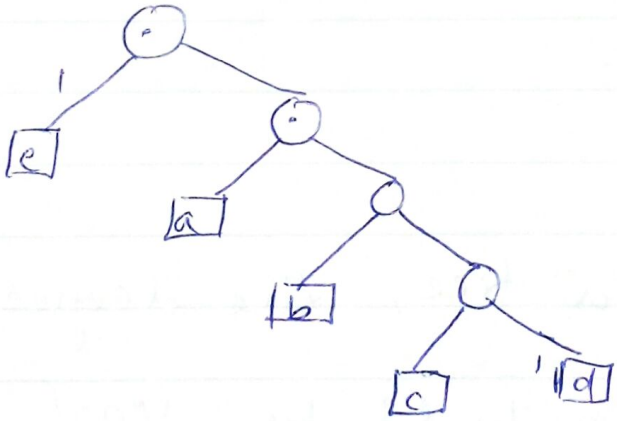
a	bcd	e
1	1	1
0.3	0.35	0.35



"abcd"
⇓
0.3 + 0.35
= 0.65

4)

abcd	e
0.65	0.35



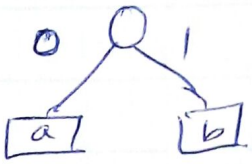
⟶

If you go from 'top to Down' (shannon's Algo). It'll be sub-optimal.

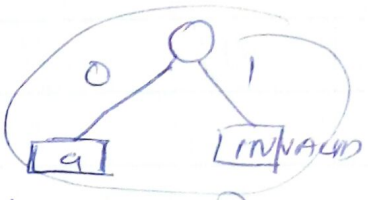
Why Huffman-Code Optimal?

Proof of Optimality:-

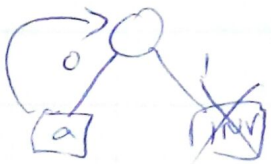
Three main points

1) Deepest node always has a sibling. It should be like 

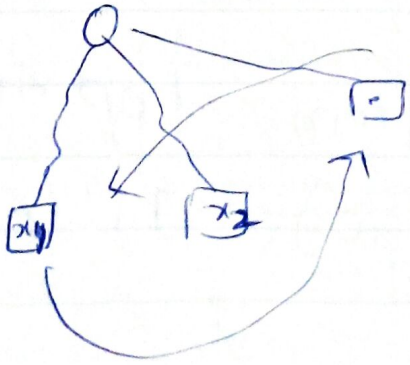
but cannot be



→ Huffman-code will not give invalid nodes

2) 

(2) In a tree, the deepest nodes, x_1 & x_2 are (or) should be least frequent characters

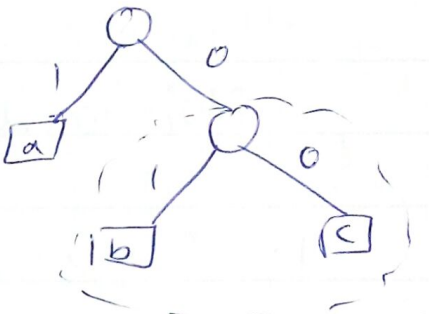


(or) If that is not the case, then exchange with the least frequent characters, node, which is up in the tree.

(3) Optimal - Substructure

Transform the input-problem, by taking '2' least frequently occurring characters & combining to 'one' character by adding probabilities (or) frequencies.,

then, the remaining tree must be optimal for the other problem



Sub-problem

Clique Decision Problem (CDP):

1. What is a clique?
2. What are Decision and Optimization problems?
3. What are NP-Hard Graph problems?
4. Procedure to prove NP-Hard
5. prove CDP is NP-Hard.

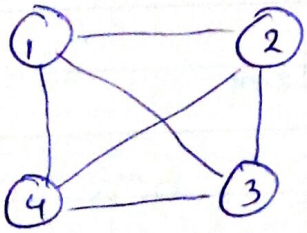
Complete Graph:-

. A graph in which every node has an edge to every other node in the graph is called "Complete Graph"

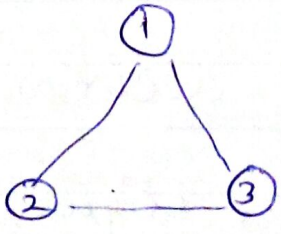
• If you consider, # of vertices = n, then

$$\#\text{ of edges} = \frac{n(n-1)}{2}$$

Examples:-



• complete graph of '4' vertices



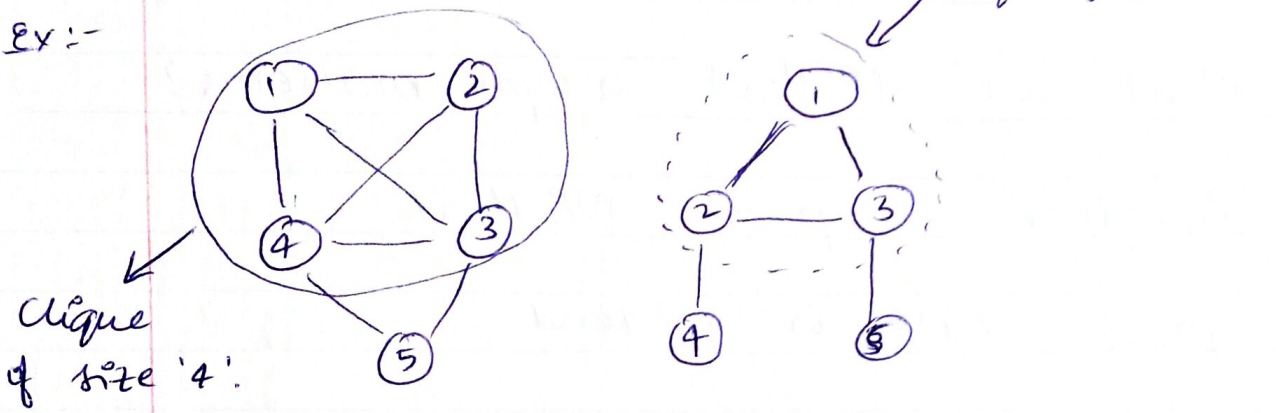
• C.G. of '3' vertices

• C.G. of '2' vertices

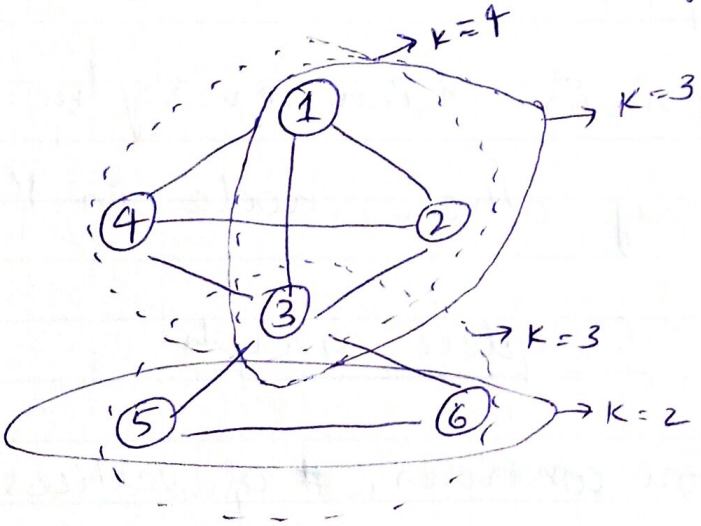
Clique :-

A subgraph of a graph, where the sub-graph is a complete graph is called clique.

Ex :-



Identify cliques in following graph:



Decision Problem :-

Problem which requires answer as "Yes" or "No" is a decision problem.

Ex: Is there a clique of size = 2 in the above problem?

Optimisation Problem:

What is the maximum size of the clique in the graph?

Reducing 3-SAT to Clique : Example:

Literals : x_1, x_2, x_3

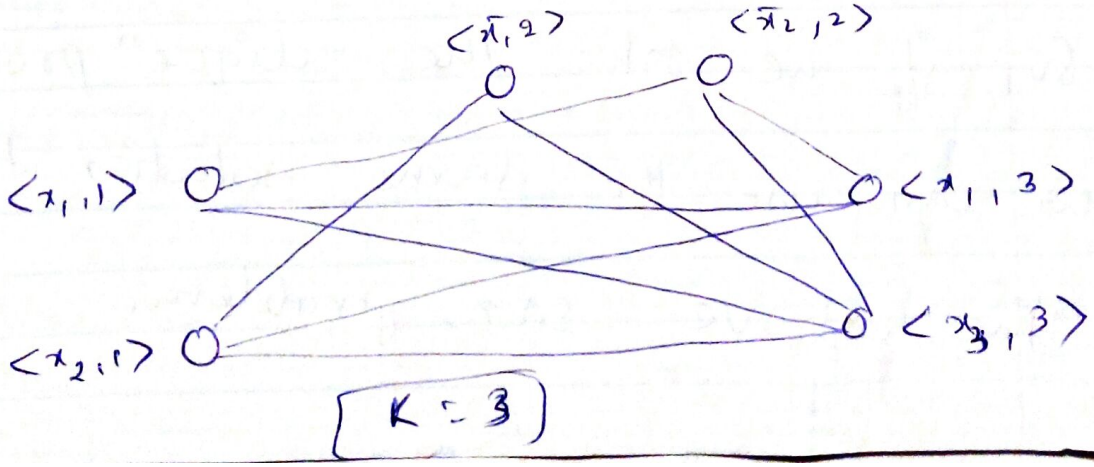
3-CNF SAT formula, $F = (x_1 \vee x_2)^{C_1} \wedge (\bar{x}_1 \vee \bar{x}_2)^{C_2} \wedge (x_2 \vee x_3)^{C_3}$

- Draw a graph using below rules

$$V = \{ \langle a, i \rangle \mid a \in C_i \}$$

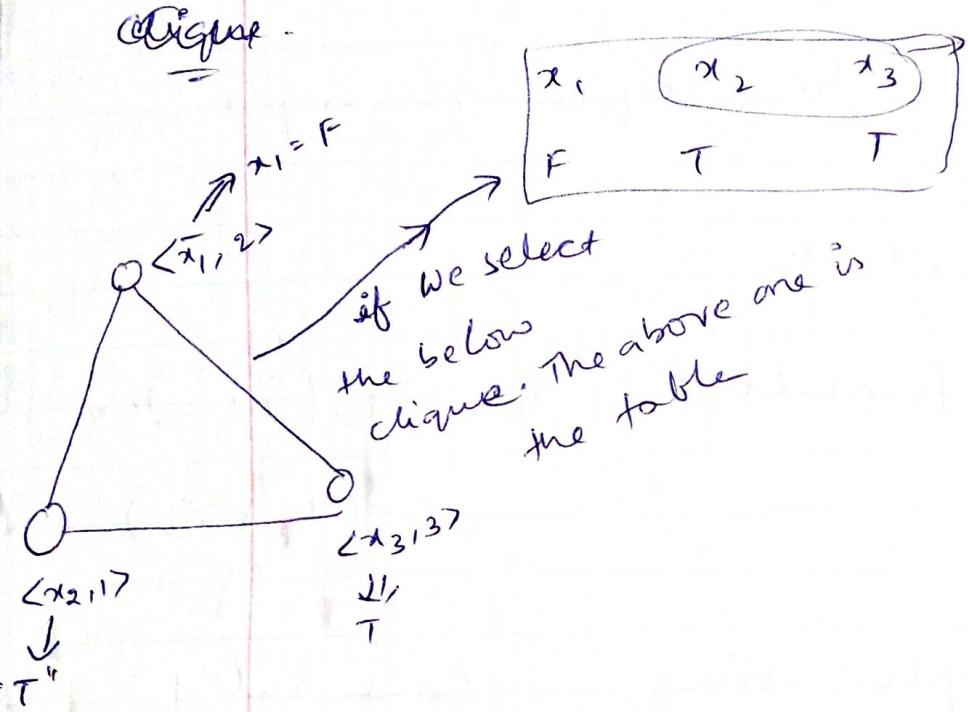
$$E = \{ \langle a, i \rangle, \langle b, j \rangle \mid i \neq j \text{ and } b \neq \bar{a} \}$$

- No connections in the same clique
- No connection b/w x_1 & $\bar{x}_1$



To find the ~~clique size~~ variable assignment values, select a clique of maximum size, and mask the corresponding ~~literal~~ variable values accordingly as ~~true, if it is present in the~~

~~clique~~.



→ If we fill the above table values in the given formula,

$$F = (x_1 \vee x_2) \wedge (\bar{x}_1 \vee \bar{x}_2) \wedge (x_1 \vee x_3)$$

$$= T$$

$C_1 = T$ ($x_1 = F$, $x_2 = T$), $C_2 = T$ ($\bar{x}_1 = T$, $\bar{x}_2 = F$), $C_3 = T$ ($x_1 = F$, $x_3 = T$)

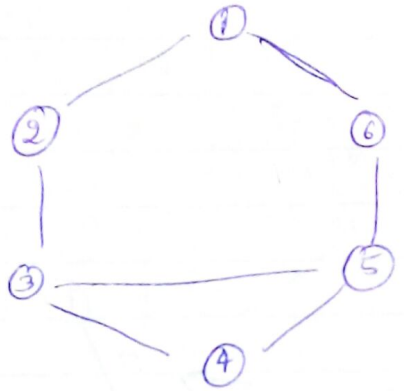
⊙ So, if we solve the "clique" problem, we can use the same solution to solve the above SAT problem.

Minimum Spanning Tree:

- No. of spanning trees for a graph

$$= \left({}^{|E|}C_{|V|-1} - \text{no. of cycles in graph} \right)$$

Ex:-



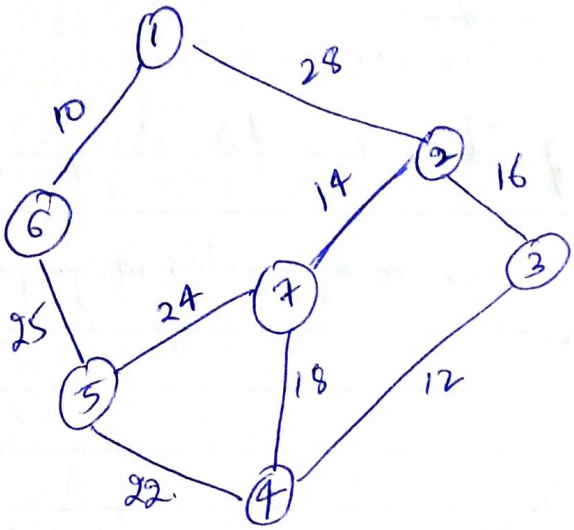
$\Rightarrow \left({}^{7}C_{5} - 2 \right)$ ← '2' cycles in graph.

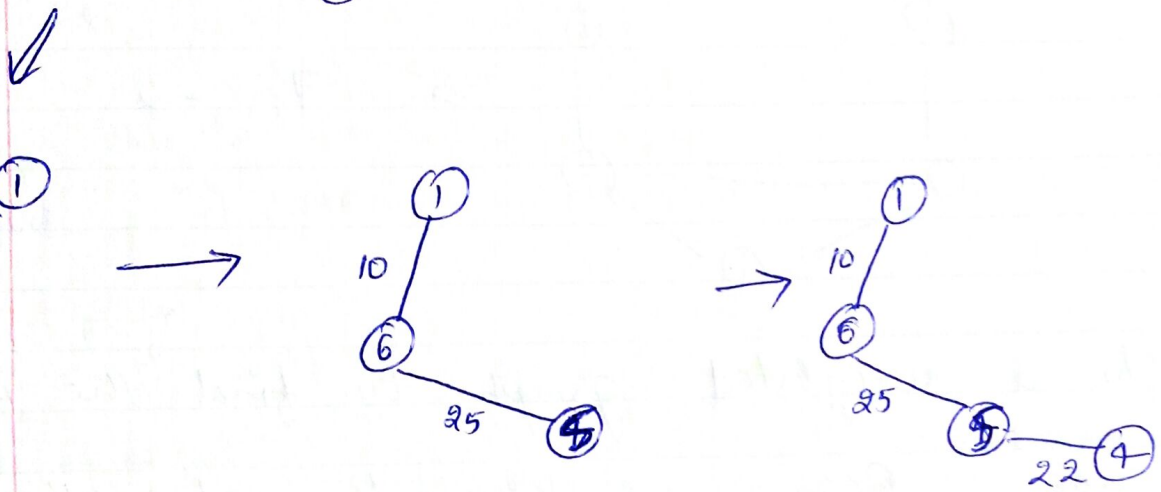
- In a weighted graph, to find the MST, we use Prim's & Kruskal's algorithm. (Greedy algorithms).

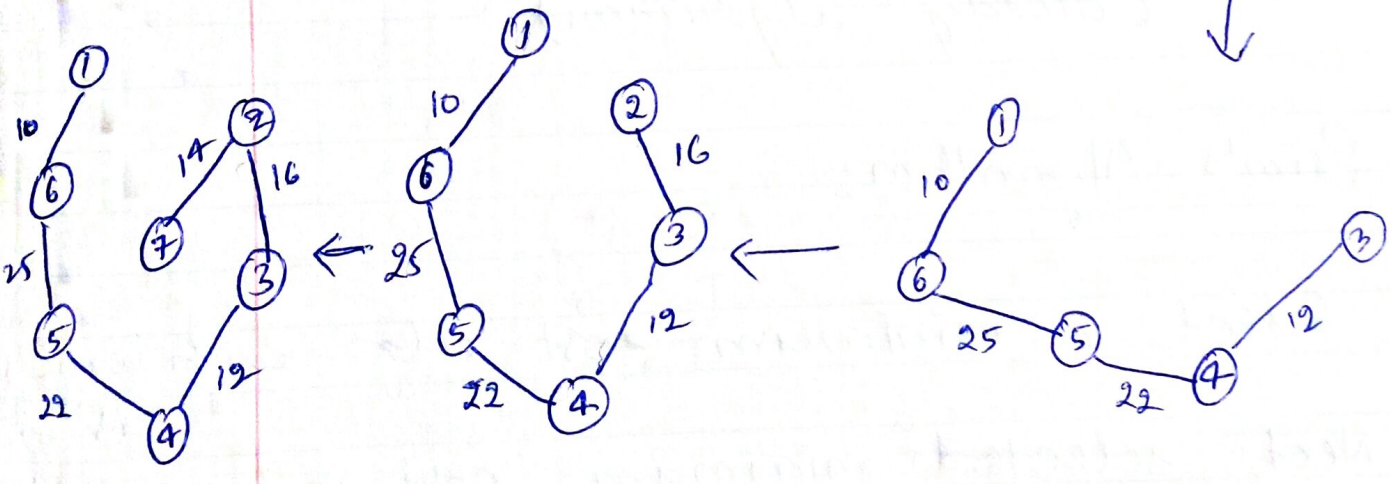
Prim's Algorithm:-

1. Select a minimum cost edge with vertices, V_1 & V_2.
2. Next, select a minimum cost edge which is connected to either 'V_1' or 'V_2'.

Example:





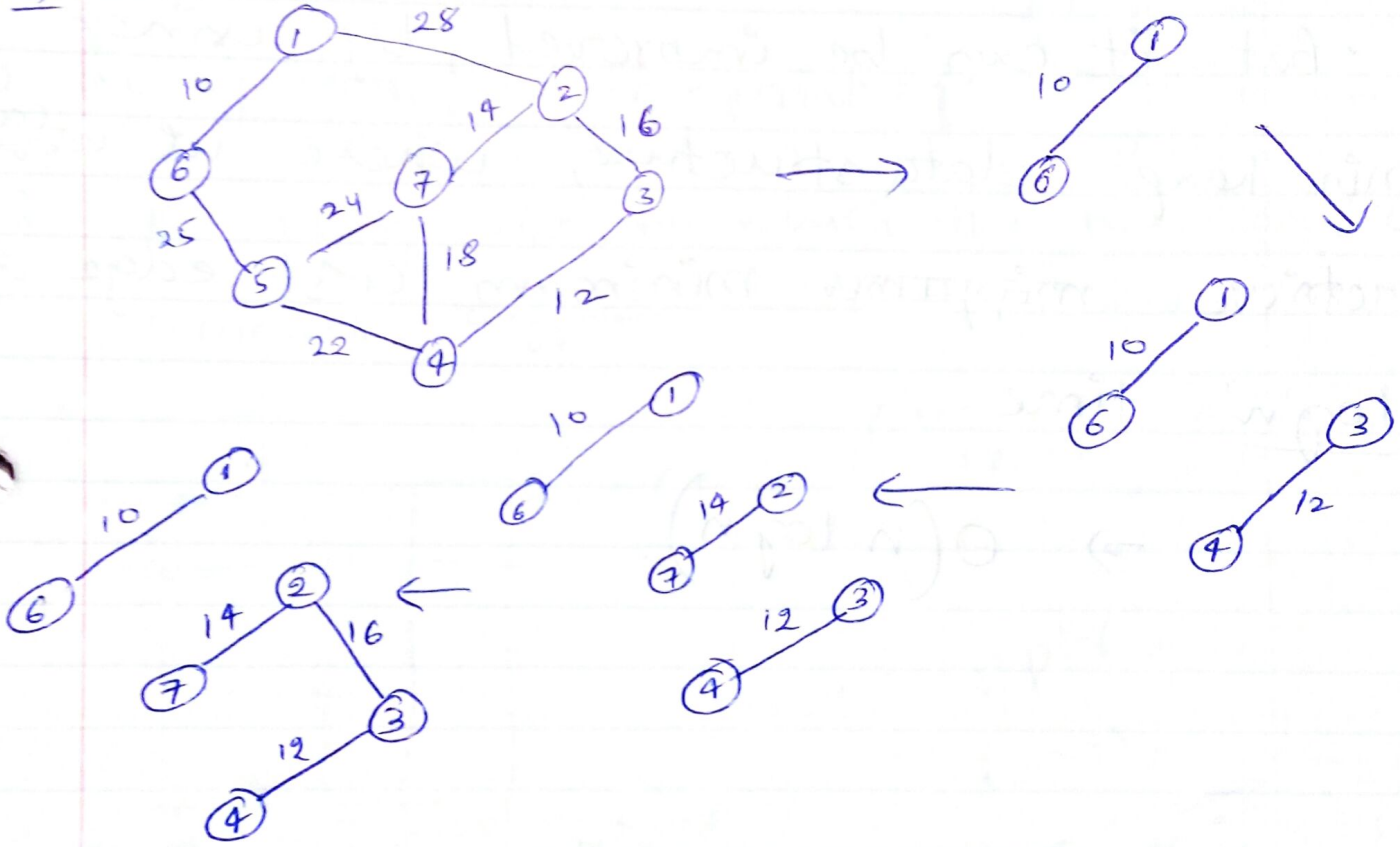


- For non-connected graphs, we cannot find MST.

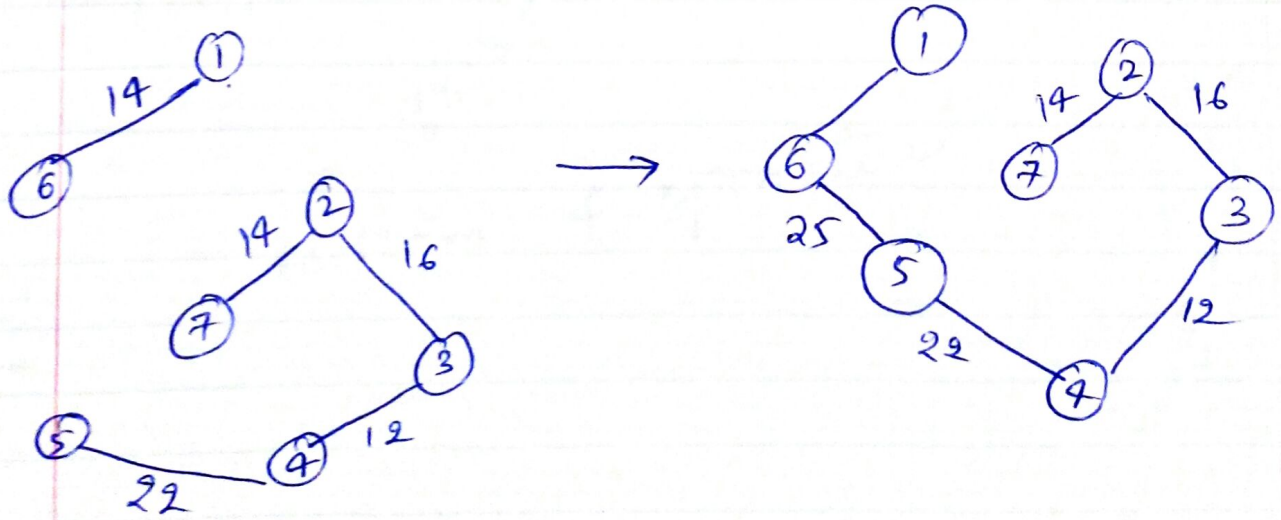
Kruskal's Algorithm:-

In this algorithm, we always select a minimum cost edge. but only if it does not form a cycle.

Ex:-



edge b/w '4' & '7' forms a cycle so, discard it



Time complexity :-

- In kruskal, we have to select $|V|-1$, edges out of $|E|$ edges. So, time taken is $O(|E|\,|V|) \Rightarrow O(n^2)$.

- But it can be improved by using "min heap" data structure, where we can retrieve minimum cost edge in "log n" time.

$\Rightarrow \Theta(n \log n)$.