TIME COMPLEXITY AND ASYMPTOTIC NOTATION

$O \rightarrow f(n) \rightarrow$ upper bound

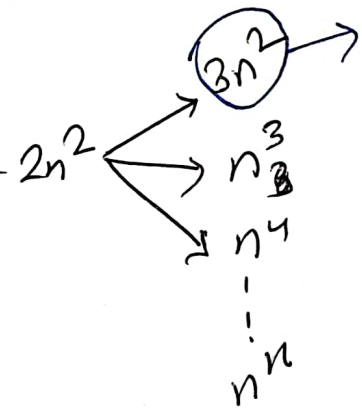
$\Omega \rightarrow f(n) \rightarrow$ lower bound

$\theta \rightarrow f(n) \rightarrow$ upper + lower bound

Big $O \rightarrow$ If $f(n)$ and $g(n)$ are two functions, then $f(n) = O(g(n))$, If there exists constants c and n_0 such that $f(n) \leq c \cdot g(n)$ for all $n \geq n_0$.

Example —

$f(n) = 2n^2$



$f(n)$ का least upper bound, $g(n)$ का क्या होगा Big O.

* If $f(n) = 4n+3$. Is $f(n) = O(g(n))$?

$\Rightarrow f(n) \leq c \cdot g(n)$

$\Rightarrow 4n+3 \leq 5n$

c ← 5 , n → $g(n)$

$f(n) = O(g(n)) = O(n)$

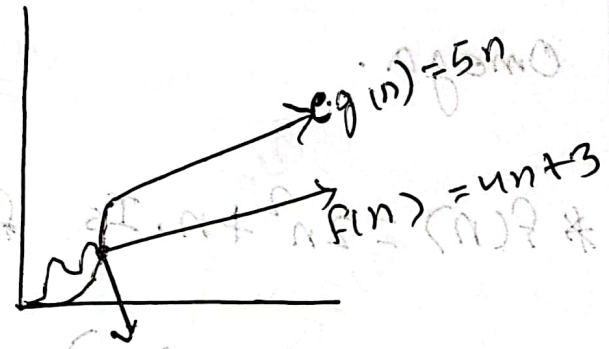
Now,

$4n+3 \leq 5n$

$\Rightarrow 3 \leq 5n - 4n$

$\Rightarrow 3 \leq n$

3 → n_0



[n_0 point এর পর থেকে $c \cdot g(n) > f(n)$]

* $f(n) = 2n^2 + n$. Find $O(g(n))$.

$\Rightarrow f(n) \leq c \cdot g(n)$

$\Rightarrow 2n^2 + n \leq 3n^2$

c ← 3 , n^2 → $g(n)$.

$f(n) = O(n^2)$

Omega (Ω) — If $f(n)$ and $g(n)$ are two functions then $f(n) = \Omega(g(n))$, if there exists constants c and n_0 such that $f(n) \geq c \cdot g(n)$ for all $n \geq n_0$.

Example -

$f(n) = 2n^2$ → greatest lower bound $g(n)$

$f(n)$ এর greatest lower bound, $g(n)$ কে বলে Omega.

* $f(n) = 2n^2 + n$. Is $f(n) \geq \Omega(g(n))$?

$\Rightarrow f(n) \geq c \cdot g(n)$

$\Rightarrow 2n^2 + n \geq 2n^2$

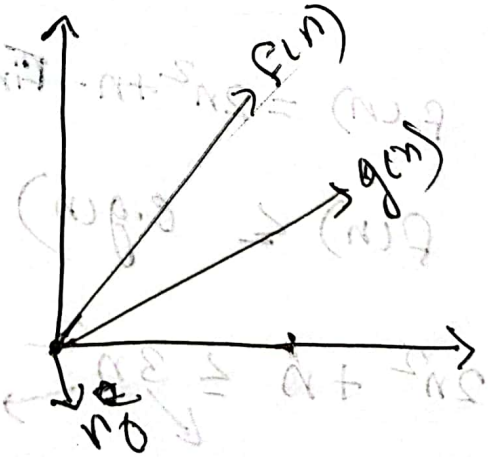
c $g(n)$

$f(n) = \Omega(n^2)$

Now,

$2n^2 + n \geq 2n^2$

$\Rightarrow n \geq 0 \rightarrow n_0$



Theta (θ) $\Rightarrow$ If $f(n)$ and $g(n)$ are two functions then $f(n) = \theta(g(n))$, if there exists constants c and n_0 such that $c_1 g(n) \leq f(n) \leq c_2 g(n)$ for all $n \geq n_0$.

* $f(n) = 2n^2 + n$, $f(n) = \theta(g(n))$ = ?

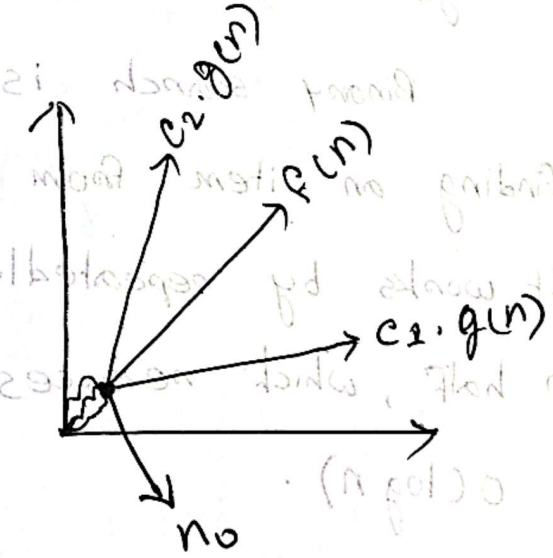
$c_1 \cdot g(n) \leq f(n) \leq c_2 g(n)$

$2n^2 \leq 2n^2 + n \leq 3n^2$

c_1, $2n^2$ → $g(n)$; c_2, $3n^2$ → $g(n)$

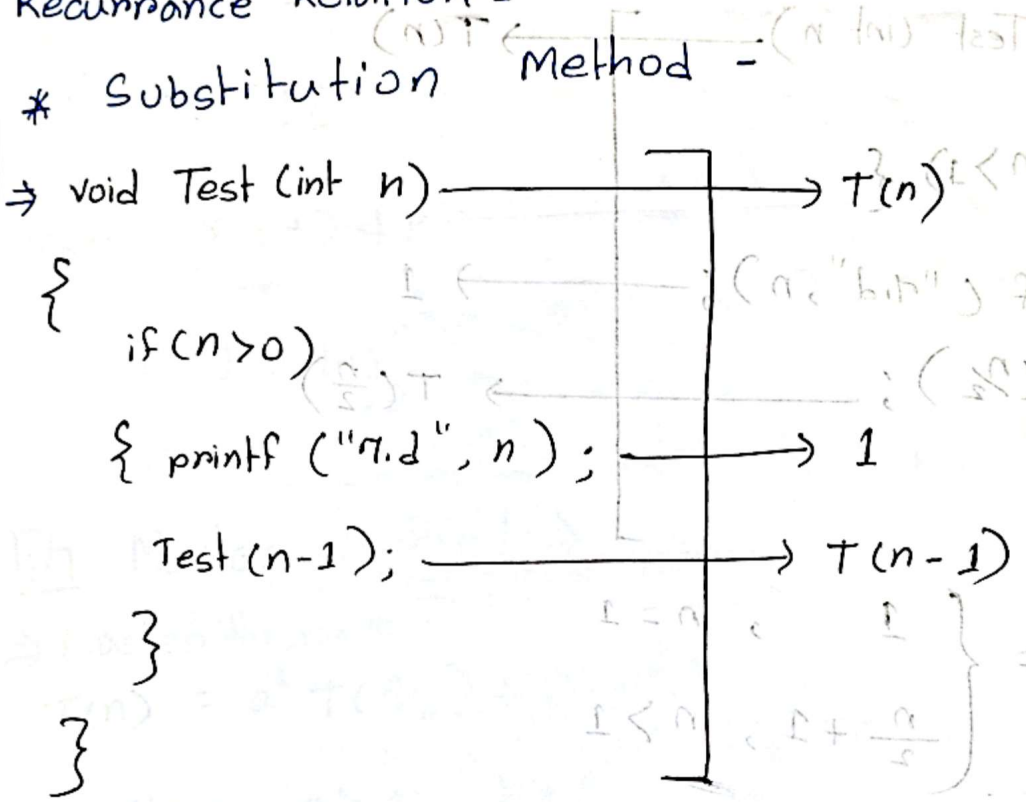
$\therefore f(n) = \theta(g(n))$

$= \theta(n^2)$



Recurrance Relation -

* Substitution Method -



$$T(n) = \begin{cases} 1 & , n = 0 \\ T(n-1) + 1 & , n > 0 \end{cases}$$

$$T(n) = T(n-1) + 1$$

$$= [T(n-2) + 1] + 1$$

$$= [[T(n-3) + 1] + 1] + 1$$

$$= T(n-3) + 3$$

$$= T(n-k) + k$$

If $k = n$

$$= T(n-n) + n$$

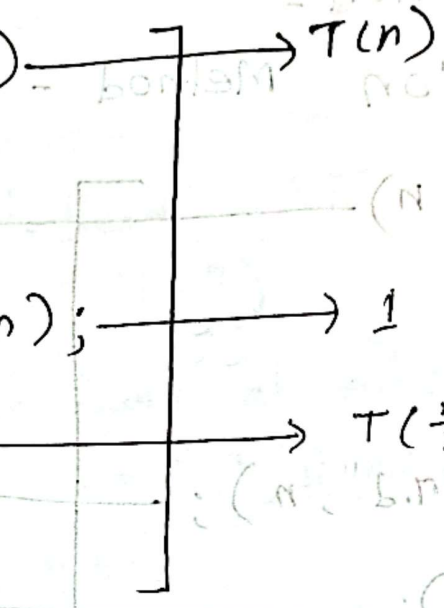
$$= T(0) + n$$

$$= 1 + n$$

∴ Time complexity = $O(n)$

⇒ void Test (int n)

```
{ if(n>1){
  printf("%d",n);
  Test(n/2);
}
```



$$T(n) = \begin{cases} 1 & , \ n = 1 \\ \frac{n}{2} + 1 & , \ n > 1 \end{cases}$$

$$T(n) = T\left(\frac{n}{2}\right) + 1$$

$$= \left[T\left(\frac{n}{4}\right) + 1\right] + 1$$

$$= \left[\left[T\left(\frac{n}{8}\right) + 1\right] + 1\right] + 1$$

$$= T\left(\frac{n}{2^3}\right) + 3$$

$$= T\left(\frac{n}{2^k}\right) + k \quad \text{when}$$

When,

$$\frac{n}{2^k} = 1$$

$$\Rightarrow 2^k = 1\,n$$

$$\Rightarrow \log_2^{2^k} = \log_2^n$$

$$\Rightarrow k \log_2^2 = \log_2^n$$

$\therefore k = \log_2^n$

$T(n) = T(1) + k$

$= 1 + \log_2^n$

Master Method ⇒

⇒ Master theorem –

$T(n) = a * T(n/b) + O(n^d)$

Here $a \geq 1$, $b > 1$

Here,

$T(n)$ = Time complexity of the algorithm on an input of size n.

a = number of recursive subproblems in each division steps.

$\frac{n}{b}$ = size of each subproblem.

$O(n^d)$ = The time complexity of combining the results of all subproblems and any additional work done outside the recursive calls.

⇒ Time complexity (TC) of the algorithm can be determined based on three cases —

(1) $O(n^d)$ if $d > \log_b^a$

(2) $O(n^d \log_b^n)$ if $d = \log_b^a$

(3) $O(n \log_b^a)$ if $d < \log_b^a$.

For binary search algorithm —

$$T(n) = 1 * T(n/2) + O(1)$$

Here, $a = 1$ [যেহেতু একটা array কে 2 ভাগে ভাগ করে নিয়ে যাচ্ছি বলে]

$\frac{n}{b} = \frac{n}{2}$; $b = 2$.

$n^d = 1$, so, $d = 0$

Now, $\log_b^a = \log_2^1 = 0 = d$

$\therefore T(n) = O(\log^n_2)$ [if $d = \log_b^a$, $T(n) = O(n^d \log_b^n)$]

For Ternary search algorithm –

$T(n) = 1 * T(\frac{n}{3}) + O(1)$

Here, $a = 1$ [1 তাকে নিয়ে কাজ করে তাই]

$b = 3$, $n^d = 1$, so, $d = 0$

Now, $\log_b^a = \log_3^1 = 0 = d$

$\therefore T(n) = O(\log_3^n)$

For merge sort algorithm –

$T(n) = 2 * T(\frac{n}{2}) + O(n)$.

Here, $a = 2$ [2 তাকে নিয়েই কাজ করে]

$b = 2$.

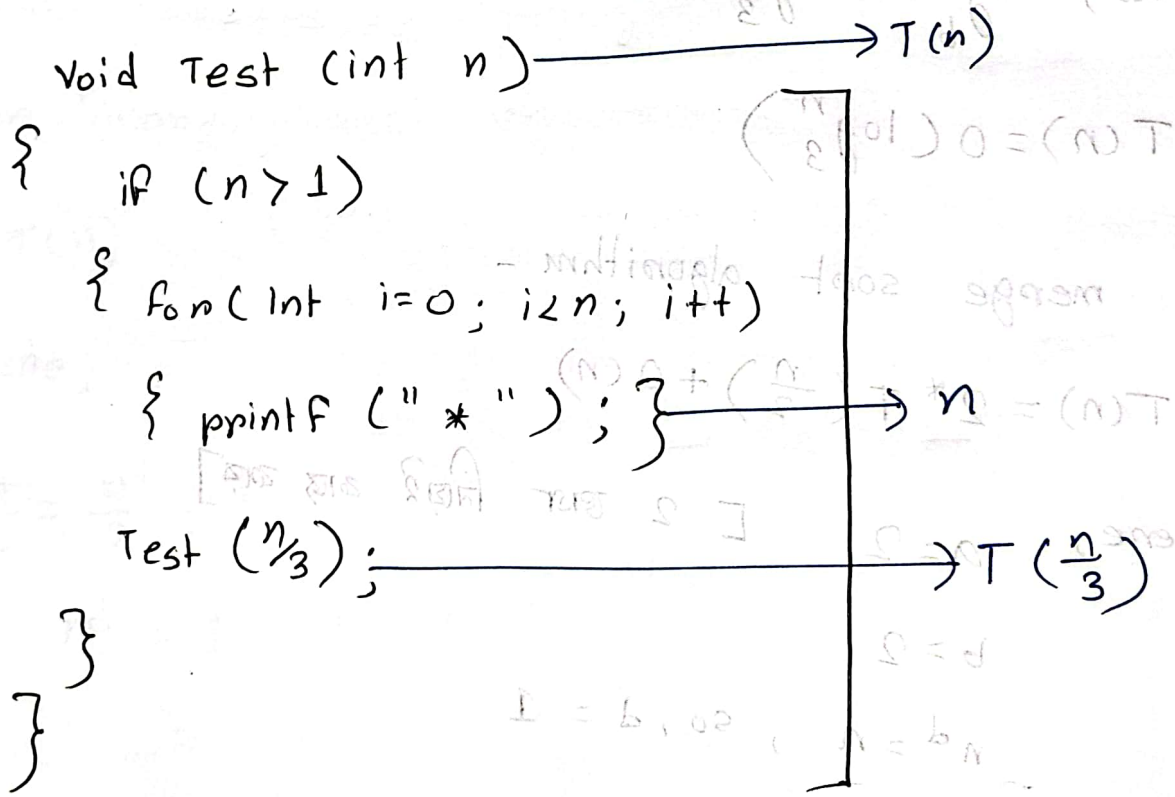
$n^d = n$, so, $d = 1$

Now, $\log_b^a = \log_2^2 = 1 = d$

$T(n) = O(n \log_2^n)$

CT QUESTION

⇒ Derive recursive relation or equation from the following function and find the time complexity using master method if possible.



Recursive relation ⇒

$$T(n) = \begin{cases} 1, & \text{when } n \leq 1 \\ T(\frac{n}{3}) + n, & \text{when } n > 1 \end{cases}$$

$$T(n) = 1 \times T\left(\frac{n}{3}\right) + n$$

$$= 1 \times T\left(\frac{n}{3}\right) + n$$

$a = 1$, $b = 3$, $n^d = n$, So, $d = 1$.

Now,

$$\log_b^a = \log_3^1 = 0 < d.$$

Here,

$$d > \log_b^a.$$

So, $T(n) = O(n^d) = O(n)$.

DYNAMIC PROGRAMMING

Divide and Conquer — Disjoint subproblems.

Dynamic Programming — Overlapping subproblems.

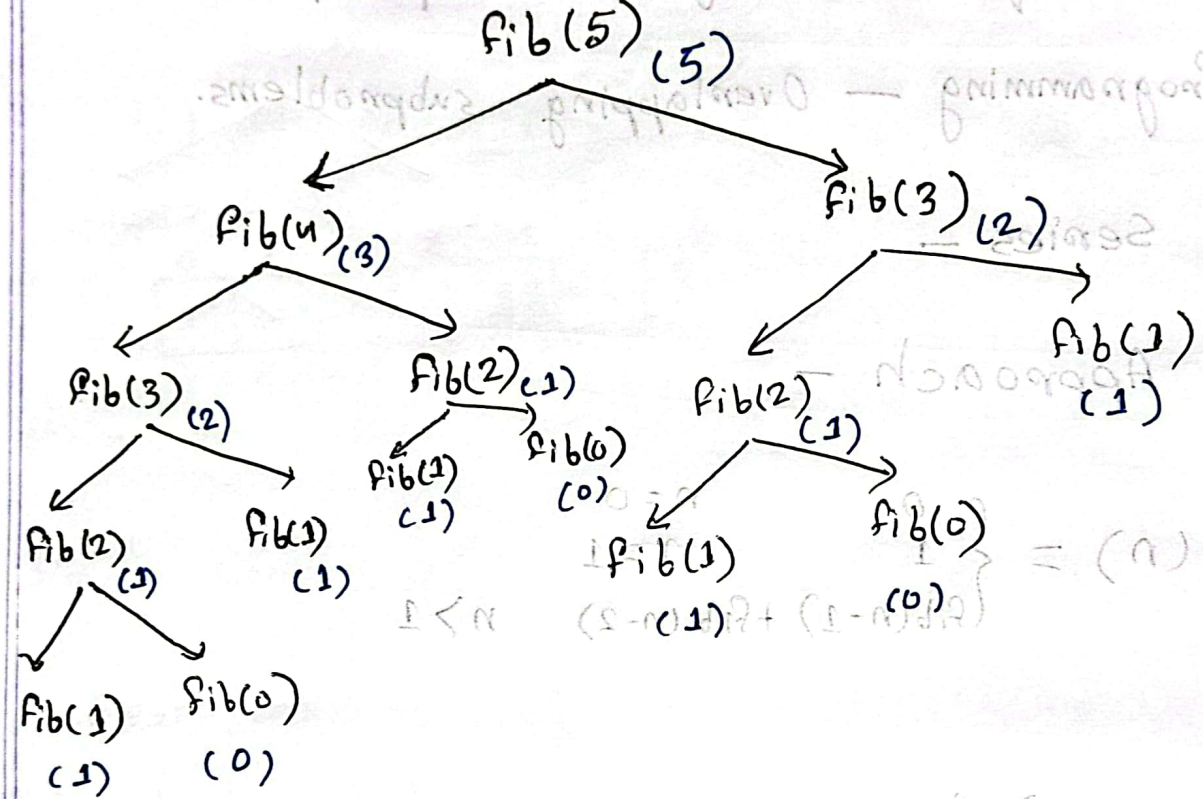
Fibonacci Series —

Recursive Approach —

$$fib(n) = \begin{cases} 0 & n = 0 \\ 1 & n = 1 \\ fib(n-1) + fib(n-2) & n > 1 \end{cases}$$

```
int fib (int n) {
    if (n <= 1)
        return n;
    return fib(n-1) + fib(n-2);
}
```

Simulation —



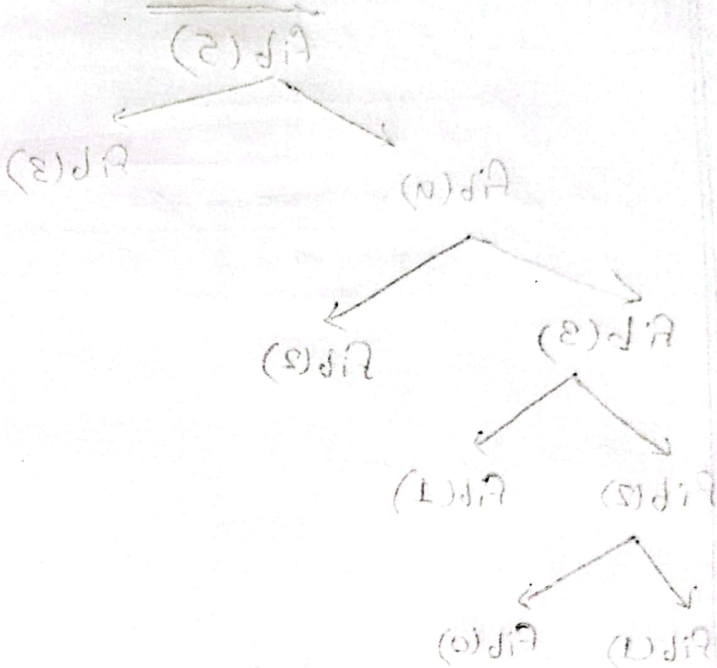
$$T(n) = T(n-1) + T(n-2) + 1$$

$$\simeq 2T(n-1) + 1$$

$a = 2$, $b = 1$; $a \geq 1$, $b \neq 1$.

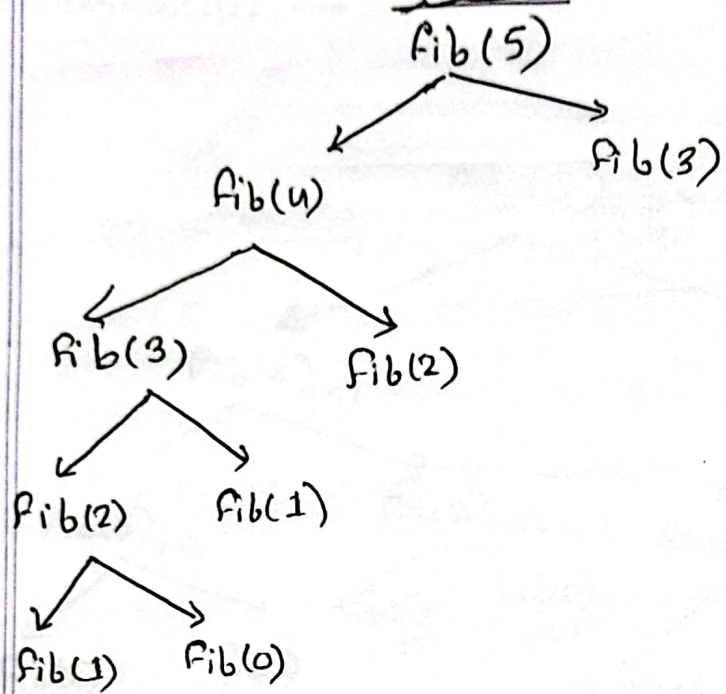
$T(n) = O(2^n)$ → Exponential complexity.

DP Approach -



```
int dp[n+1];
int fibonacci(int n)
{
    if (n == 0)
    { return 0;
    }
  else if (n == 1)
  { return 1;
  }
  else if ( dp[n] == 0)
  {
      dp[n] = fibonacci(n-1) + fibonacci(n-2);
  }
  return dp[n];
}
```

Simulation – [DP]



Factorial —

Recursive approach —

```
Factorial(n):
{
  If n==0  or  n == 1
      return 1;
  return n* Factorial(n-1)
}
```

DP Approach —

dp[n+1] # An array of size n+1 to store the results

```
Factorial(n)
{
    If n==0 or n==1
        return 1

    Else If dp[n] == 0
        dp[n] = n * Factorial(n-1)

    return dp[n].
}
```

LCS - Longest Common Subsequence

Subsequence vs substring.

"programming"

gramm - both subsequence and substring

gamng - subsequence, not substring.

* Substring এ main sequence এর সব same sequence এ element থাকবে।

Now,

S = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}

E = {2, 4, 6, 8, 10}

X_1 = {2, 4}

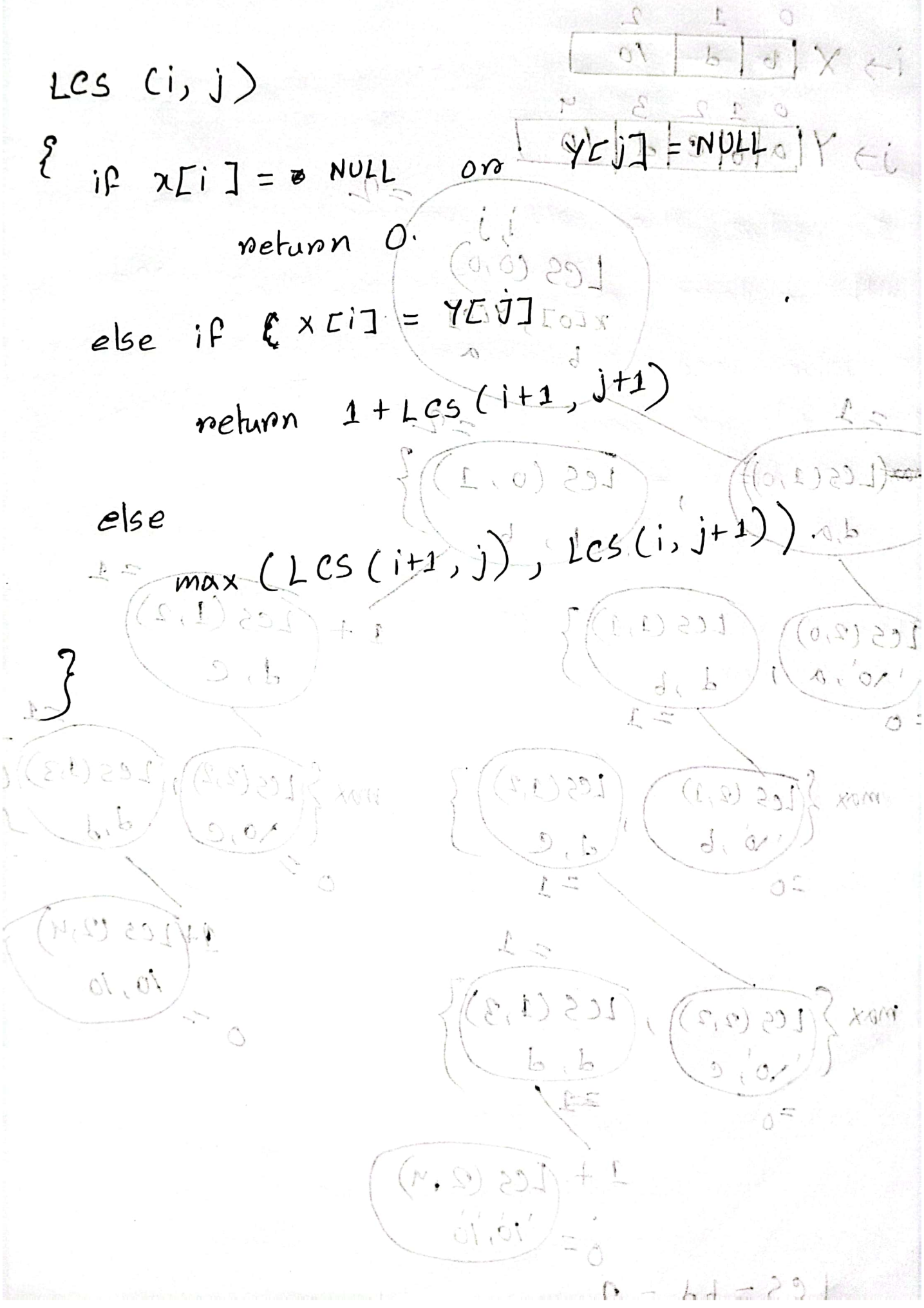
X_2 = {2, 4, 6}

X_3 = {2, 4, 6, 8, 10} → Longest.

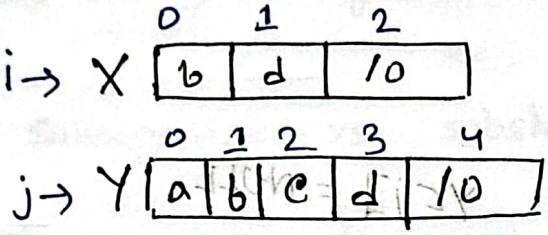
→ Common subsequence of S and E

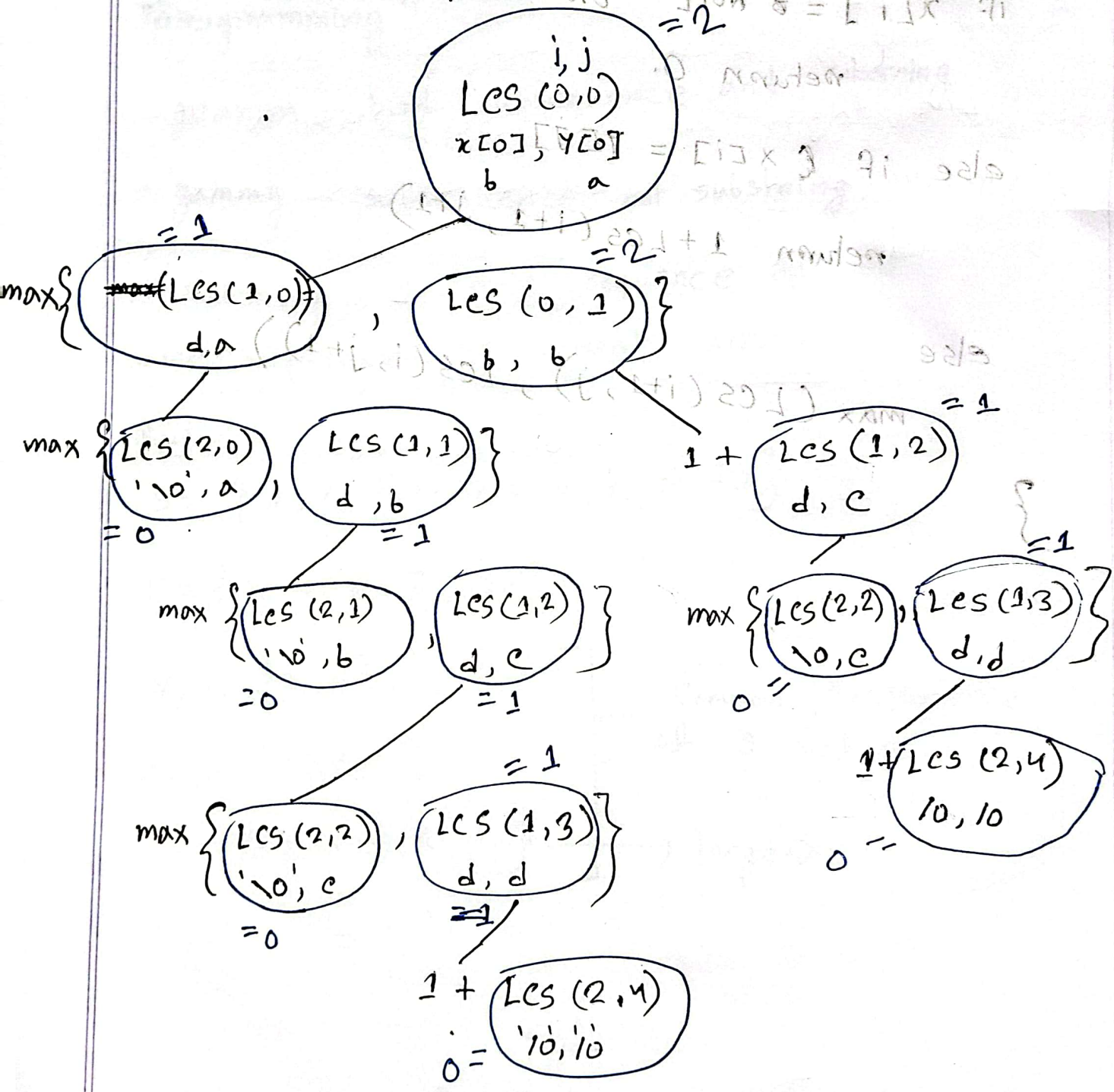
LCS - Recursive Approach -

```
LCS (i, j)
{
    if x[i] = NULL or Y[j] = NULL
        return 0.

    else if { x[i] = Y[j]
        return 1 + LCS (i+1, j+1)

    else
        max (LCS (i+1, j), LCS (i, j+1))
}
```



Simulation –





LCS = bd = 2

LCS [DP Approach] –

~~LCS (i, j)~~

To find the ~~set~~ LCS of 2 sequence, we need to follow the steps given below –

- Create a matrix dp of $m+1$ by $n+1$ size where m and n are sizes of the given sequences.
  [dp [i, j] = LCS of x_i and y_j]
- Traverse each cell row-wise or column-wise.
- If $i=0$ or $j=0$, then set $dp[i,0] = dp[0,j] = 0$
- If $x_i = y_j$, set $dp[i,j] = dp[i-1, j-1] + 1$
- If $x_i \neq y_j$, set $dp[i,j] = \max(dp[i-1, j], dp[i, j-1])$

Knapsack —

Algorithm —

To solve the 0/1 knapsack problem with n items, each with a weight and a value, and a knapsack with a maximum capacity, W, we need to follow the steps given below —

(1) Create a matrix dp of ~~size~~ n+1 by w+1. [where dp[i][w] = the maximum value of items 1 ... i with weight limit w].

(2) If i=0 or w=0, set dp[i,0] = dp[0,w] = 0.

(3) Traverse each cell row-wise or column-wise.

(4) If weight of current item i exceeds the remaining capacity w,

set dp[i, w] = dp[i_0 - 1, w].

[skip the current item and take value of previous row].

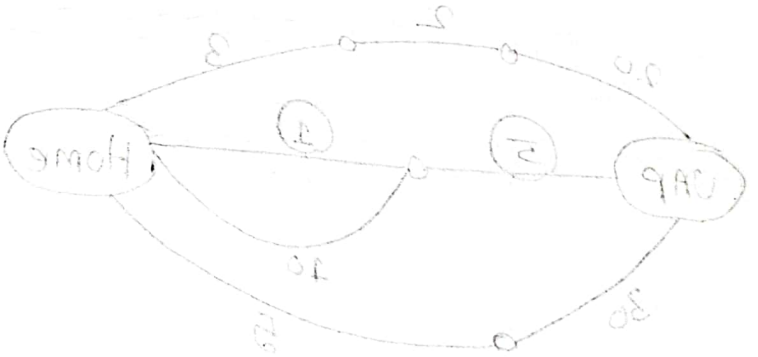
(5) If the current item can fit (weight [i-1] <= w):

~~set dp[i, w] = max(dp[i-1, w], dp[i-1, w-weight[i]]+~~

set dp[i, w] = max(dp[i-1, w], dp[i-1, w - w[i]] + v(i))

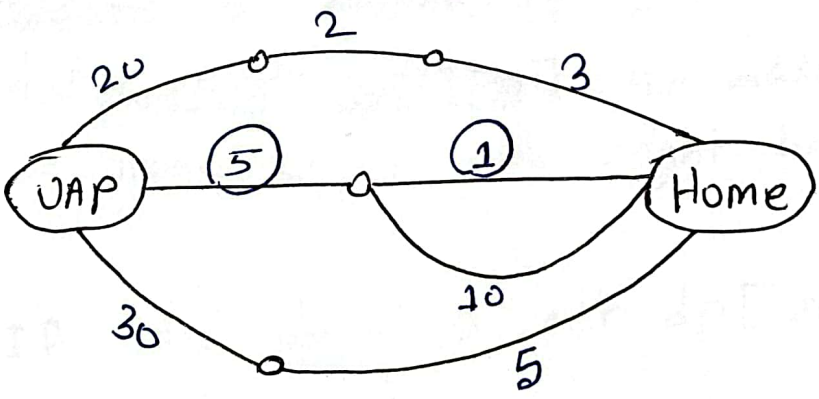
[choose the maximum between skipping the item or taking it)

(6) The value at dp[n, w] will be the maximum value ~~sho~~ that can be achieved with the given capacity.



Greedy Algorithm —

- Makes locally optimal choice
- Once a decision made never reconsidered.
- do not always give optimal solution.



Example: 0/1 knapsack

Item	value	weight
1	$100	20
2	$120	30
3	$60	10

Capacity 50.

Objective: Max profit

constraint: capacity

Step-1: Calculate value by weight for each item

Step-2: Sort items based on the above ratio in descending order.

Step-3: Take item or not (0/1).

Solution ⇒

Item	value	weight	value/weight
1	$100	20	5
2	$120	30	4
3	$60	10	6

After sorting in descending order –

Item	Value	Weight	Value/weight
3	$60	10	6
1	$100	20	5
2	$120	30	4

If weight <= capacity, pick item.
Then find remaining capacity.

(1) Item 3 → 10 < 50

Pick item 3. capacity = 50 − 10 = 40

Profit = $60.

(2) Item 1 → 20 < 40

Pick item 1. capacity = 40 − 20 = 20

Profit = $60 + $100 = ~~$160~~ $160

(3) Item 2 → 30 > 20.

Item 2 rejected.

∴ Max Profit = Item 3 + Item 1

= $60 + $100

= $160 [Ans]

Here, we can see that, ~~maximum~~ if we took item 1 and 2, then ~~we~~ the maximum profit would be $220 within the given capacity. If we took item 2 and 3, then the maximum profit would be $180 within given capacity. $220 and $180 are greater than $160. So, we can say that greedy algorithm doesn't give optimal solution of 0/1 knapsack.

Fractional knapsack -

এখানে Example এর মাধ্যমে,

Item - 2 rejected হয়ে যা 1 অতিরিক্ত capacity থাকে অতিরিক্ত নেয়া হয়।

Item - 2 ⇒ Fraction ⇒ 20.

Max Profit ⇒

= Item 3 + Item 1 + Item 2 $\left(\frac{20}{30}\right)$

= 60 + 100 + $\cancel{120}^{4} \times \frac{20}{30}$

= $240

Example –

Capacity = 15

Item	1	2	3	4	5	6	7
Value	11	6	16	8	7	19	4
Weight	3	4	6	8	2	5	2
Value/ weight	3.67	1.5	2.67	1	3.5	3.8	2
	2	6	4	7	3	1	5

$$x \begin{pmatrix} 1 & 0 & 5/6 & 0 & 1 & 1 & 0 \\ x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & x_7 \end{pmatrix}$$

Item	Value	Weight	Remaining capacity	Total Profit
6	19	5	15-5=10	19
1	11	3	10-3=7	19+11=30
5	7	2	7-2=5	30+7=37
3	$\frac{5}{6}$ x16 = 13.3	$\frac{5}{6}$ x6 = 5	5-5=0	37+13.3 = 50.3

5<15

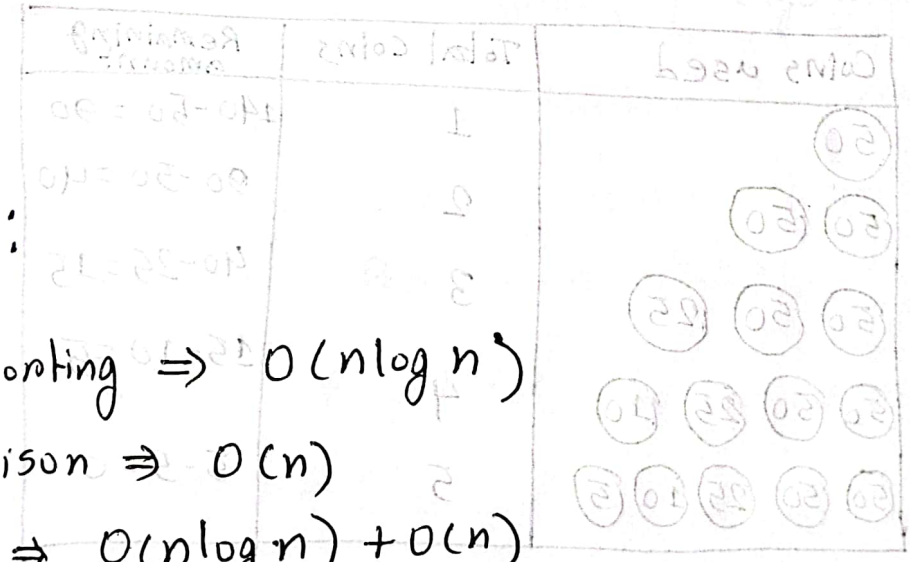
3<10

2<7

675

Algorithm :

```
for (i=0 ; i<n ; i++)
    if (arr[i].weight <= capacity)
    {
        c = c - arr[i].weight ;
        total_profit += arr[i].value ;
    }
else {
    frac = c / arr[i].weight ;
    total_profit = total_profit + frac*arr[i].value ;
}
```



Complexity :

For sorting ⇒ $O(n \log n)$

Comparison ⇒ $O(n)$

Overall ⇒ $O(n \log n) + O(n)$

since $O(n \log n)$ is prominent -

Time complexity = ~~of For~~ $O(n \log n)$.

Coin Change Problem ⇒

Objective – Minimum number of coins.

Constraints – Coins are available in infinite numbers.

Example –

$n = 140$

$C = \{5, 10, 20, 25, 50\}$

After sorting –

$C = \{50, 25, 20, 10, 5\}$

If $C[i] <= n$, pick coin

	Coins used	Total coins	Remaining amount
$50 \leq 140$	(50)	1	$140 - 50 = 90$
$50 \leq 90$	(50) (50)	2	$90 - 50 = 40$
$25 <= 40$	(50) (50) (25)	3	$40 - 25 = 15$
$10 <= 15$	(50) (50) (25) (10)	4	$15 - 10 = 5$
$5 = 5$	(50) (50) (25) (10) (5)	5	$5 - 5 = 0$

Number of coins used = 5.

*"Greedy doesn't give optimal solution of coin change problem every time" - Explain with example.

⇒ Assume, $n = 6$ and $C = \{1, 3, 4\}$.

After sorting, $C = \{4, 3, 1\}$

Coins	Total	Remaining
(4)	1	6-4=2
(4)(1)	2	2-1=1
(4)(1)(1)	3	1-1=0

Here, we can see that, if we took two coins of amount 3. That would be the minimum number of coins to make 6 with the available coins. But, using greedy algorithm, we found that 3 coins are needed for making 6 with the available coins. So, greedy doesn't always give optimal solution for coin change problem.

Algorithm :

```
int greedyCoinChange (int c[], int n, int i) {

  if (n == 0) return 0;

  if (c[i] <= n)

    return 1 + greedyCoinChange (c, n-c[i], i);

  else

    return greedyCoinchange (c, n, i+1);

}
```

Complexity ⇒

Array size = m

For sorting ⇒ O(m log m)

Total amount making ⇒ O(n).

Overall ⇒ O(m log m) + O(n)

Here, n >> m.

So, O(n) is prominent.

Time complexity = O(n).

Job sequencing with deadline ⇒

Job ID →	1	2	3	4	5
Profit →	10	5	20	1	15
Deadline →	1	3	2	3	2

0 — 1 — 2 — 3

প্রতি JobID এর জন্য একটি Timeslot।

প্রতি Timeslot এ only একটা Job।

এই example এ Timeslot 3টি - 0 to 1, 1 to 2, 2 to 3

After sorting ⇒

JobID →	3	5	1	2	4
Profit/Value →	20	15	10	5	1
Deadline →	2	2	1	3	3

JobID	Profit	Deadline	slot assigned
J_3	20	2	[1,2]
J_5	15	2	[0,1]
J_1	10	1	Rejected.
J_2	5	3	[2,3]
J_4	1	3	Rejected

Solution - (J_3, J_5, J_2)

Profit $\Rightarrow$ $20+15+5 = 40$.

Complexity $\Rightarrow$

Sorting n jobs $= O(n \log n)$

Choice $= O(n)$

Overall $= O(n) + O(n \log n)$.

Since, $O(n \log n)$ is prominent,

Time complexity $= O(n \log n)$.

GRAPH

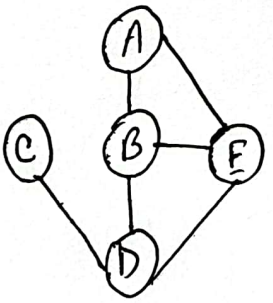
Types of Graph :

Directed ⇒ Direction আছে।

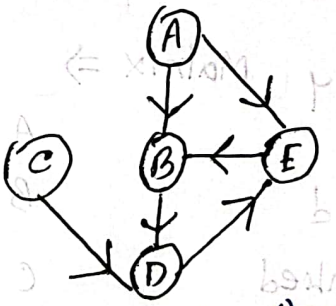
Undirected ⇒ Direction নাই।

Weighted ⇒ Weight আছে।

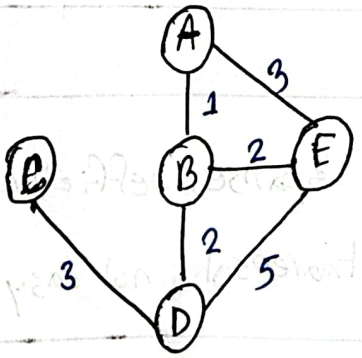
Unweighted ⇒ Weight নাই।



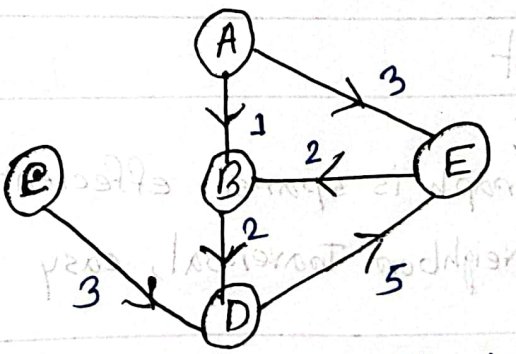
Undirected Unweighted



Directed Unweighted



Undirected Weighted



Directed weighted

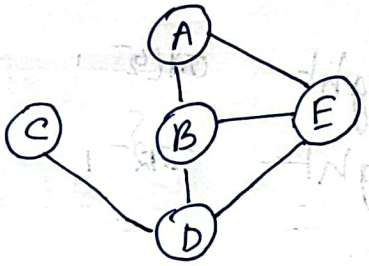
Graph Representation ⇒

Two standard Ways —

(1) Adjacency List

(2) Adjacency Matrix

Adjacency List ⇒



⇒

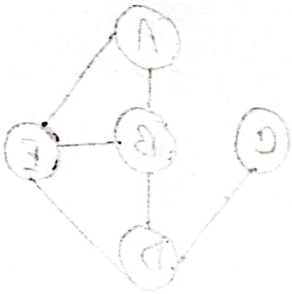
A → B, E
B → A, D, E
C → D
D → B, C, E
E → A, B, D

Adjacency Matrix ⇒

	A	B	C	D	E
A	0	1	0	0	1
B	1	0	0	1	1
C	0	0	0	1	0
D	0	1	1	0	1
E	1	1	0	1	0

1 → Linked

2 → Not linked

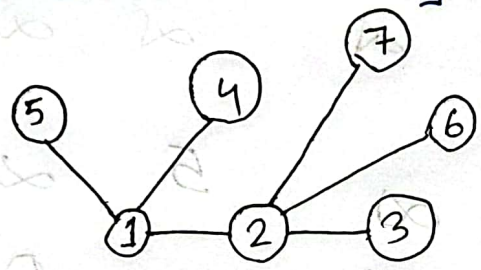


Differences:

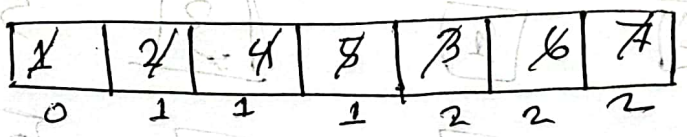
List	Matrix
Pros: (i) Graph is sparse, effecient (ii) Neighbor Traversal, easy	Cons: (i) Graph is sparse, inefficient. (ii) Neighbor traversal, not easy
Cons: (i) Graph is dense, inefficient (ii) Whether edge is present or not, not easy.	Pros: (i) Graph is dense, efficient. (ii) Whether edge is present or not, easy

Graph Traversal Algorithm:

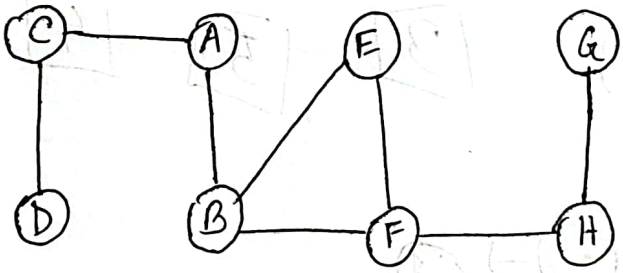
BFS [Breadth First Search]:



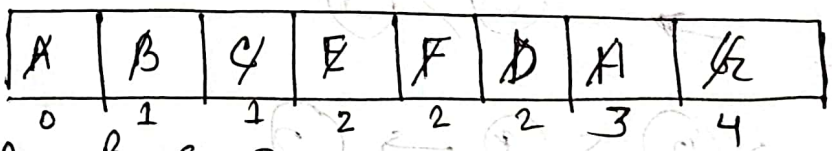
Queue :



Visited nodes: 1, 2, 4, 5, 3, 6, 7

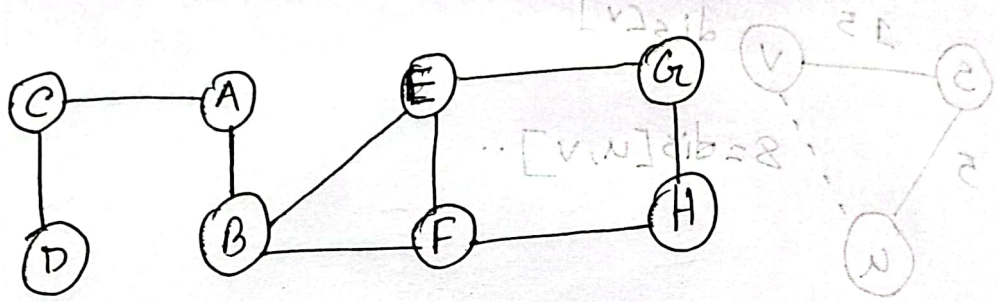


Queue :

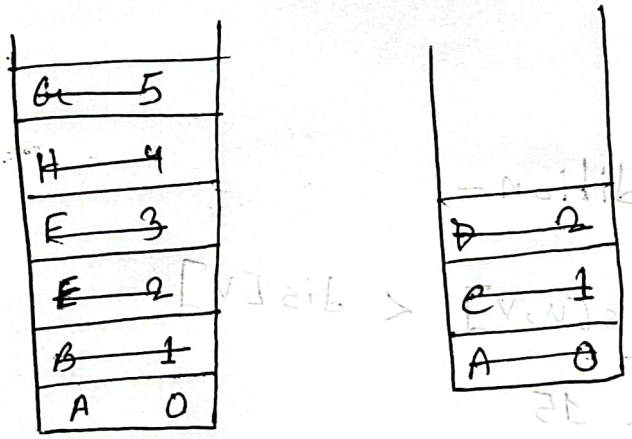


Visited: A, B, C, E, F, D, H, G

DFS (Depth first search):



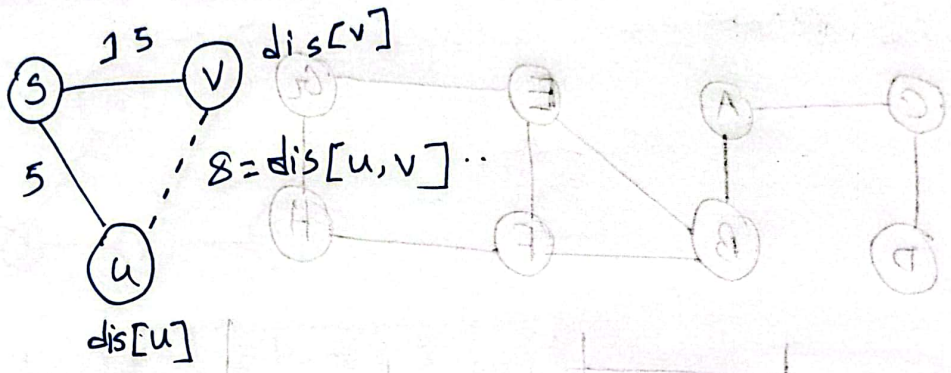
Stack :



Visiting Nodes : A , B , E, F, H, G, C, D

Complete Nodes : G , H , F, E , B, D, C , A

Dijsktra's Shortest Path Algorithm –



Relaxation condition –

$$dis[u] + dis[u,v] < dis[v]$$

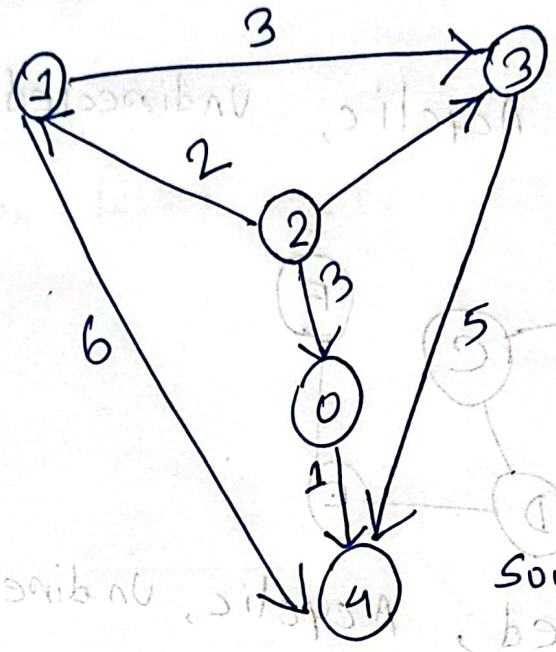
$$\Rightarrow 5 + 8 < 15$$

$$\Rightarrow 13 < 15$$

$$dis[v] = dis[u] + dis[u,v]$$

* Parent ⟶ Path
* Distance ⟶ cost

*



Source = 2

Parent

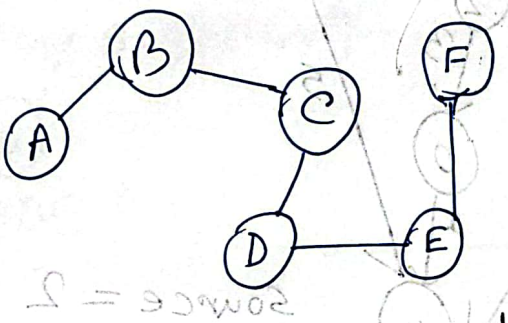
0	1	2	3	4
~~-1~~ 2	~~-1~~ 2	-1	~~-1~~ ~~2~~ 1	~~-1~~ ~~4~~ 0

Distance

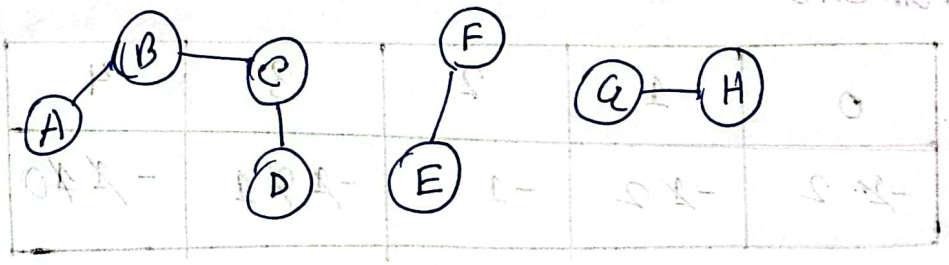
	0	1	2	3	4
2	∞	∞	[0]	∞	∞
1	Min(∞,0+3) 3	Min(∞,0+2) [2]	[0]	min(∞,0+7) 7	∞
0	[3]	[2]	[0]	Min(7, 2+3) 5	Min(∞, 2+6) 8
4	[3]	[2]	[0]	5	Min(8,3+1) [4]
3	[3]	[2]	[0]	[5]	[4]

Trees

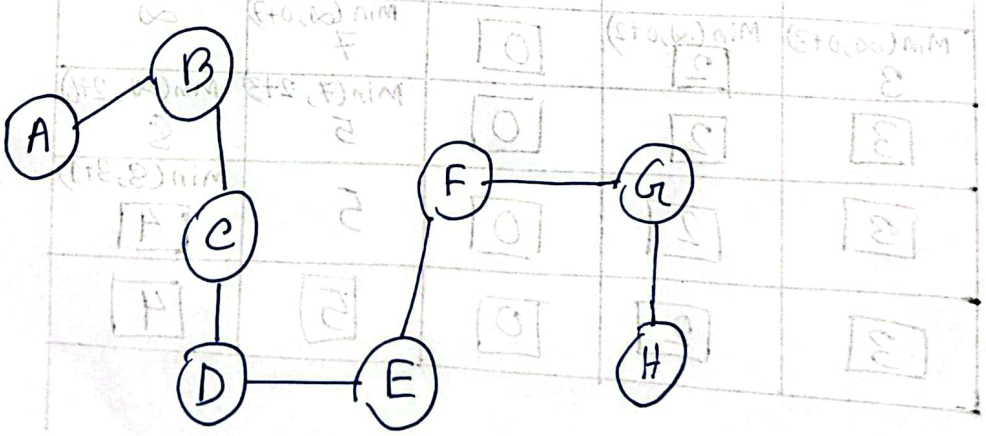
Tree : Connected, Acyclic, Undirected graph



Forest : Disconnected, Acyclic, Undirected graph



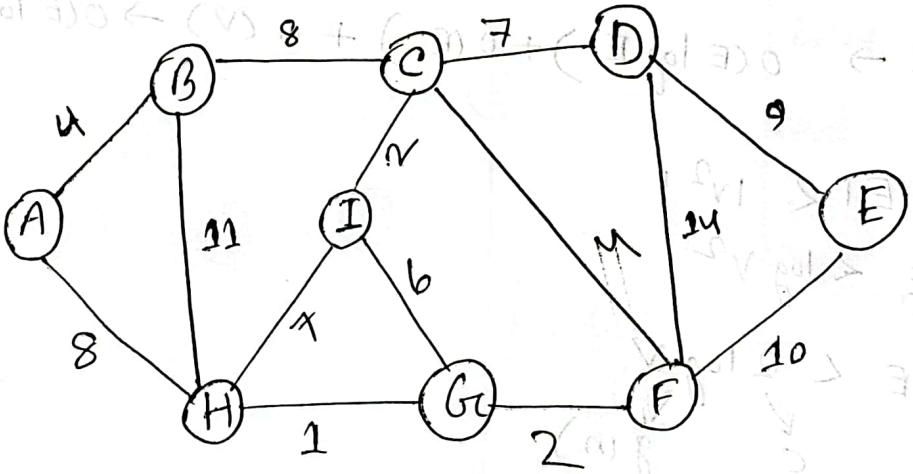
Spanning tree : A tree which includes all the vertices of a graph.

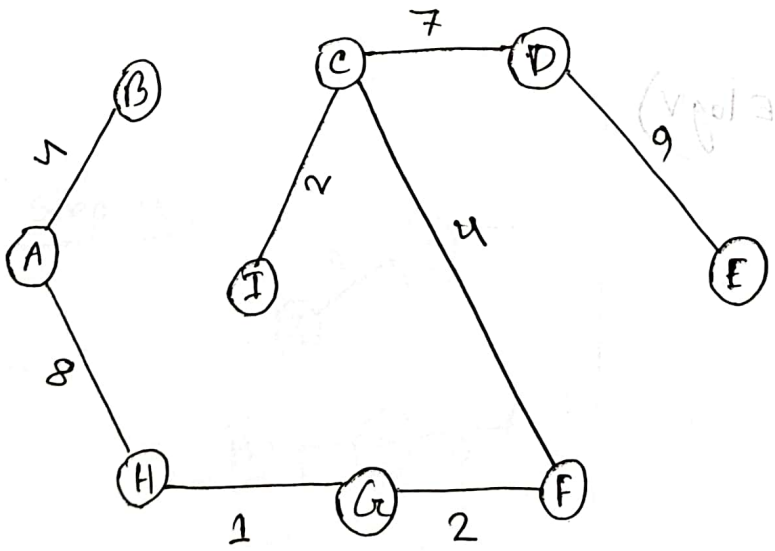


Kruskal Algorithm: Minimum weight spanning tree বের করবো।

Ascending order এ sort করবো। Then, same edge এ না থাকলে নিবো। Cycle হয়ে যায় এমন edge বাদ দিবো।

Simulation:





Edge	Weight
G-H	1 ✓
G-F	2 ✓
C-I	2 ✓
A-B	4 ✓
C-F	4 ✓
G-I	6 ✗ [Cycle]
C-D	7 ✓
H-I	7 ✗ [Cycle]
A-H	8 ✓
B-C	8 ✗ [Cycle]
D-E	9 ✓
E-F	10 ✗
B-H	11 ✗
D-F	14 ✗

Found MST with Total weight = 4+8+1+2+4+2+7+9
= 37

Time Complexity –

Sorting E edges $\rightarrow O(E \log E)$

Selecting E edges $\rightarrow O(E)$

Checking V nodes if is in the same component (Disjoint sets) $\rightarrow O(V)$

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

As, $|E| < |V^2|$

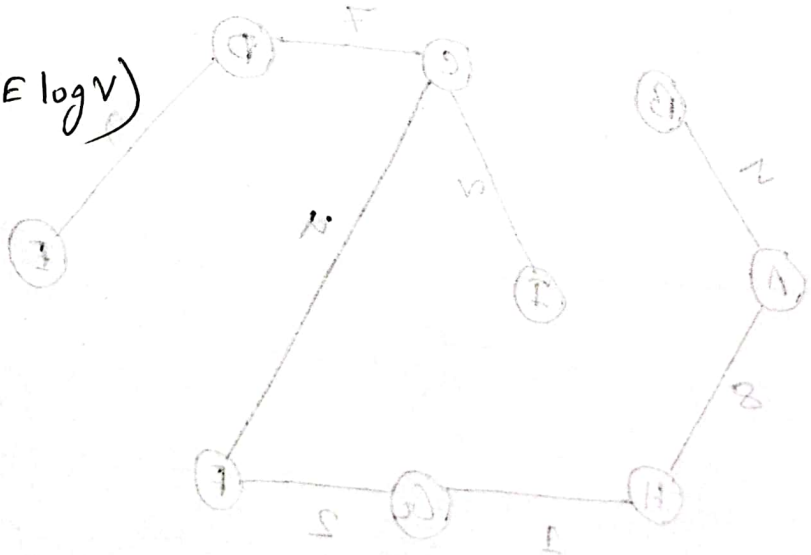
$\Rightarrow \log E < \log V^2$

$\Rightarrow \log E < 2 \log V$

$f(n) \quad c \quad g(n)$

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

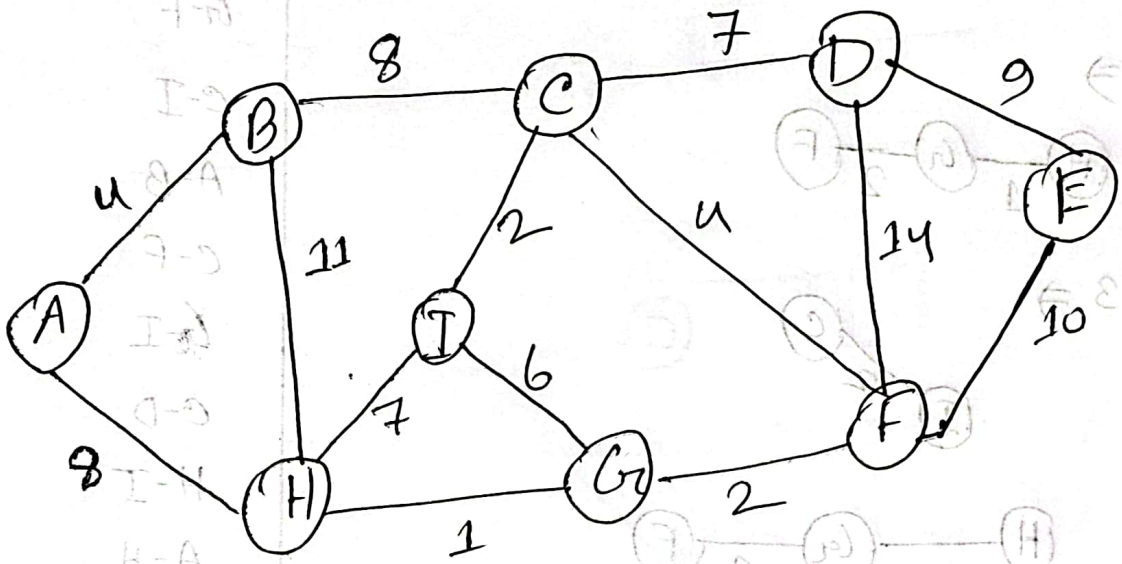
$O(E \log E) = O(E \log V)$



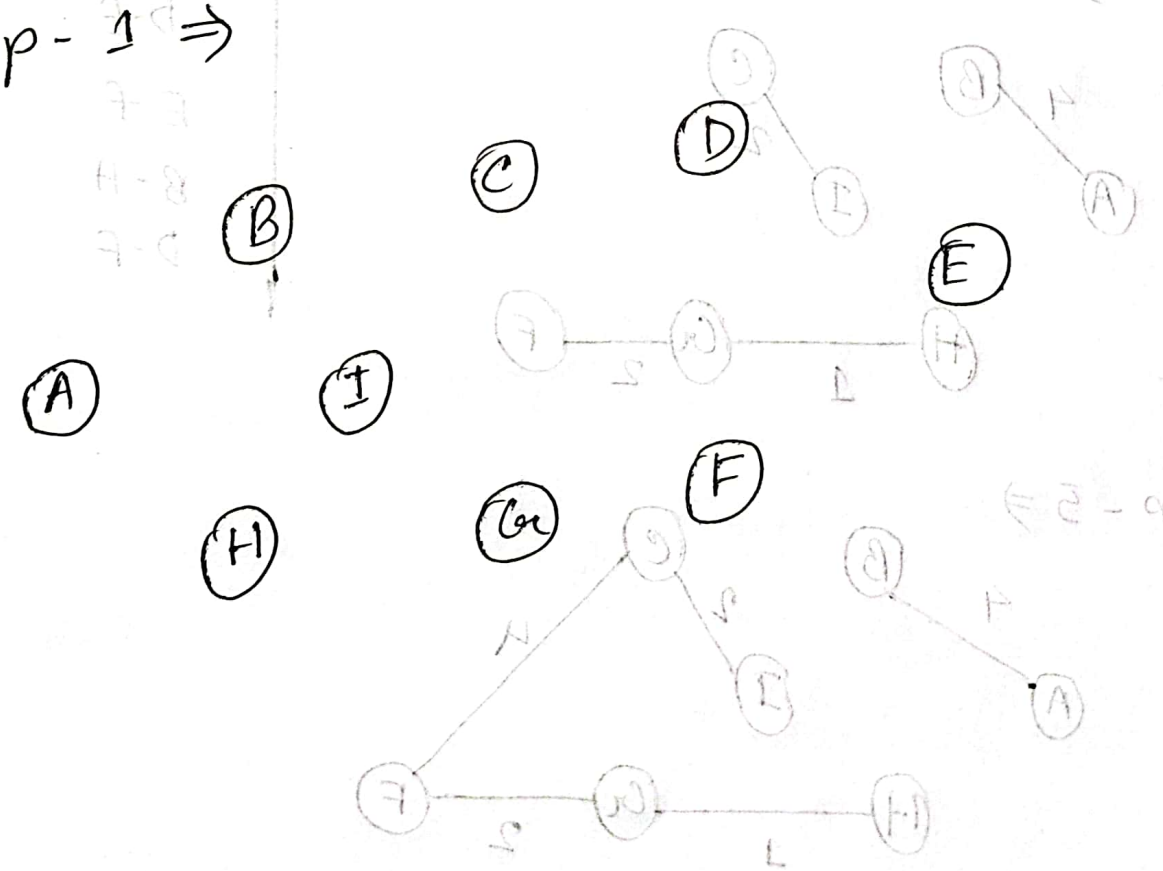
Kruskal Algorithm :

Simulation ⇒

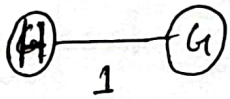
*



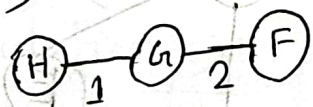
Step - 1 ⇒



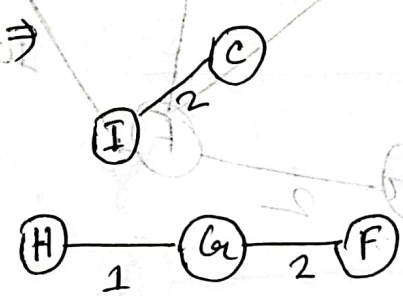
Step-1 ⇒



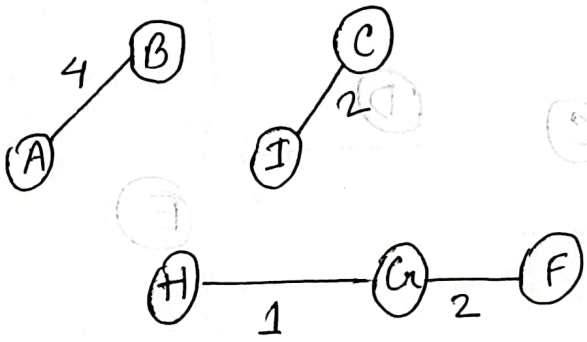
Step-2 ⇒



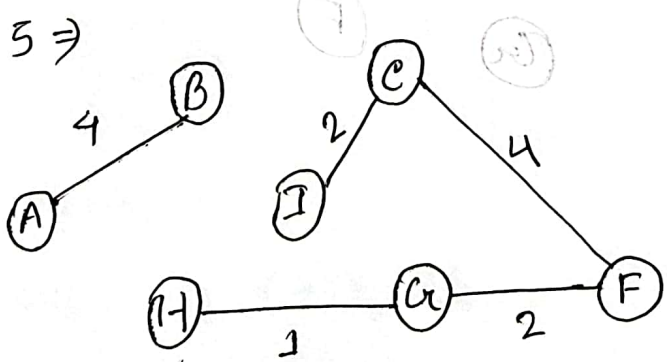
Step-3 ⇒



Step-4 ⇒

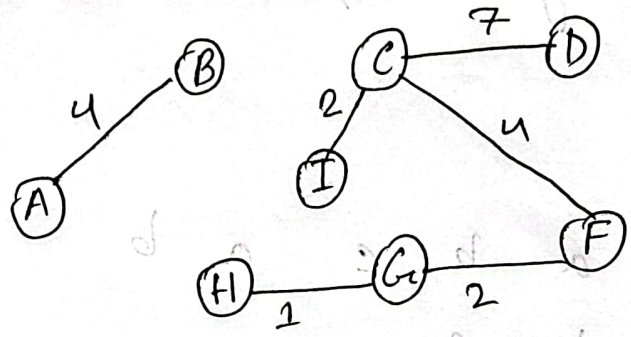


Step-5 ⇒

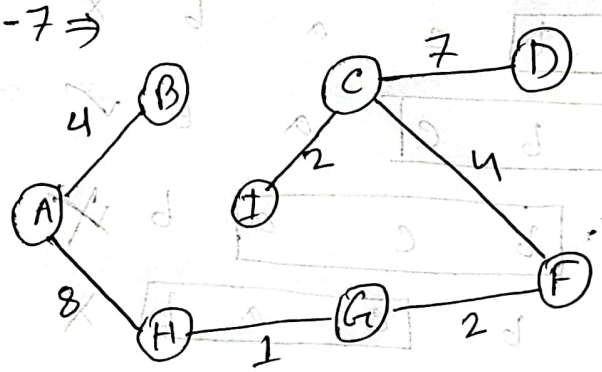


Edge	Weight
G-H	1 ✓
G-F	2 ✓
C-I	2 ✓
A-B	4 ✓
C-F	4 ✓
G-I	6 ✗ [Cycle]
C-D	7 ✓
H-I	7 ✗ [Cycle]
A-H	8 ✓
B-C	8 ✗ [Cycle]
D-E	9 ✓
E-F	10 ✗
B-H	11 ✗
D-F	14 ✗

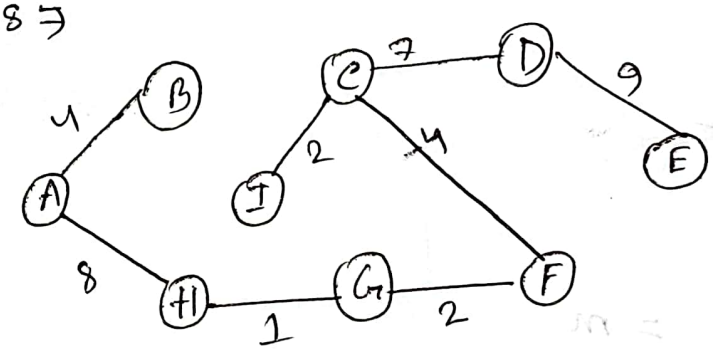
Step-6 ⇒



Step-7 ⇒



Step-8 ⇒



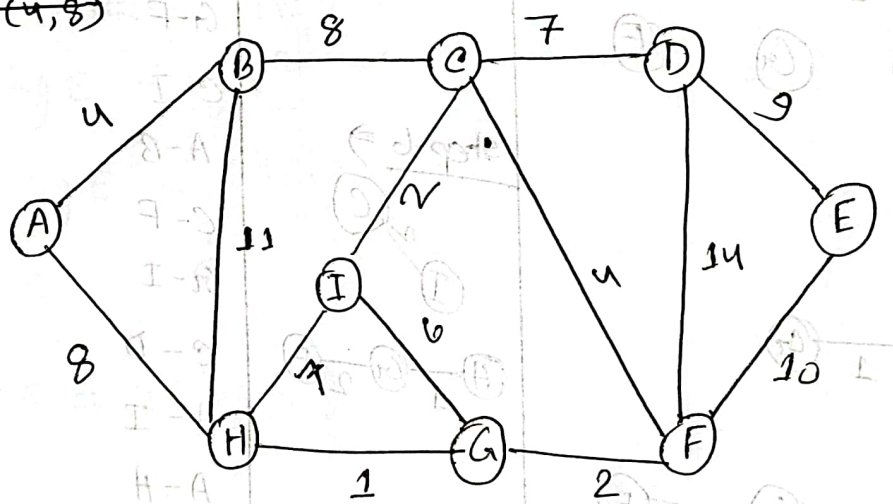
Found MST with total weight = 4+8+1+2+4+2+7+9 = 37

Prim's Algorithm —

Source node ধরবে। Then minimum weight এর ~~Edge~~ First এ source node add করে, then ওইখান থেকে যেসব যায় ওই way গুলোর minimum যেটা ওইটা নিবে।

Simulation —

~~Adding A to MST~~
~~Min (4, 8)~~



Steps ⇒ (1) Add A to MST.

(2) Min (4, 8), Add B, A-B to MST

(3) Min (8, 11, 8), Add C, B-C to MST

(4) Min (8, 11, 2, 4, 7), Add I, C-I to MST.

(5) Min (8, 11, 4, 7, 6, 7), Add, F, C-F to MST.

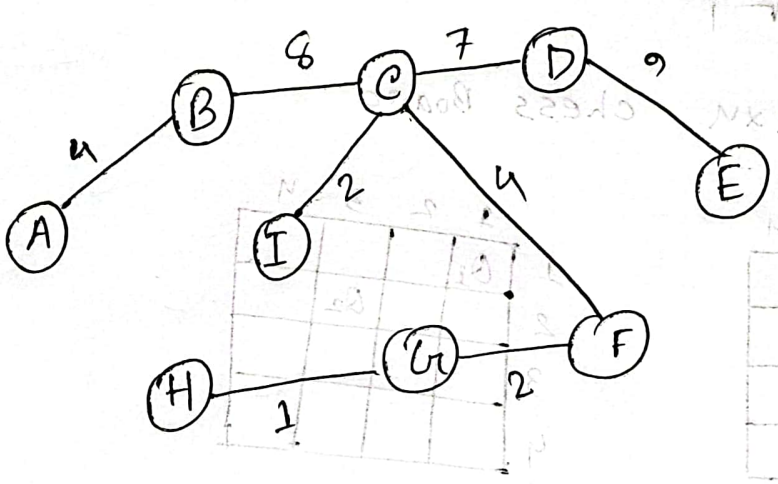
(6) Min (8, 11, 7, 6, 7, 2, 10, 14), Add, G, F-G to MST.

(7) Min (8, 11, 7, 7, 10, 14), Add H, G-H to MST.

(8) Min (7, 10, 14), Add D, C-D to MST.

(9) Min(9, 10), Add E, D-E to MST.

Found MST with total weight = 4+8+2+4+2+1+7+9
= 37



Complexity ⇒

Using Adjacency Matrix - $O(V^2)$

Using Adjacency List - $O(E \log V)$

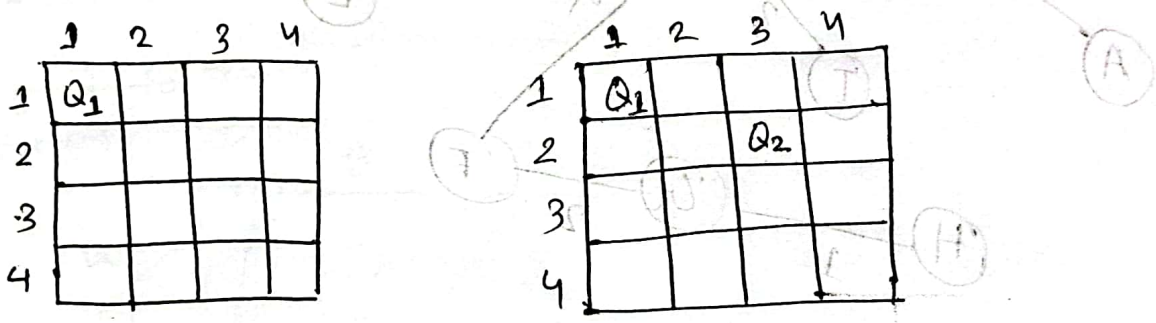
Backtracking — All Possible solutions বের করা।

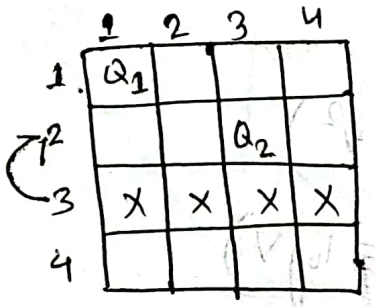
N QUEEN Problem —

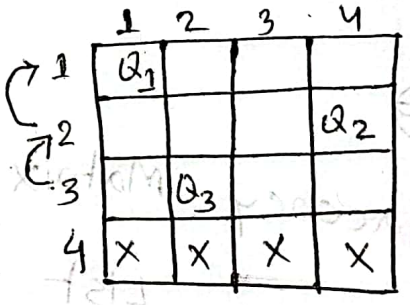
Same row, same column এবং Diagonally

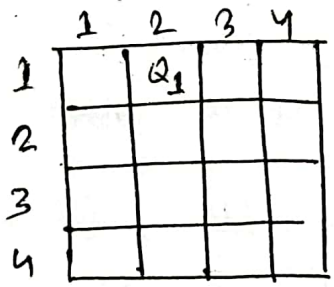
একটি করে বসানো যাবে না।

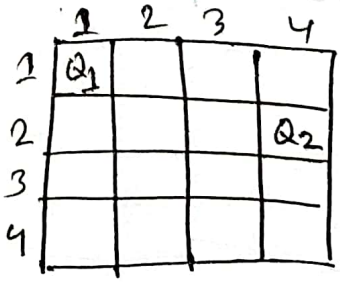
Simulation - 4×4 chess Board

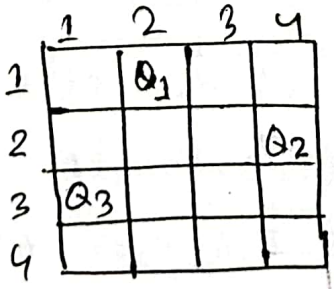


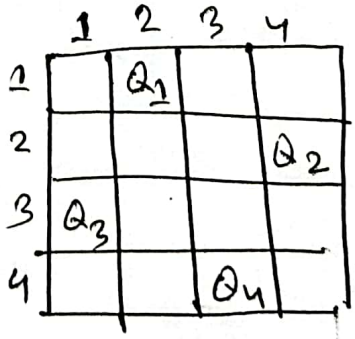




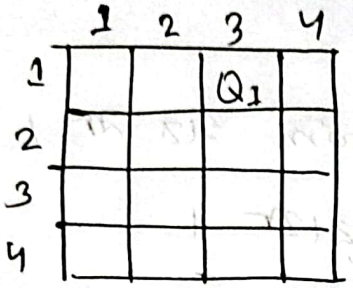


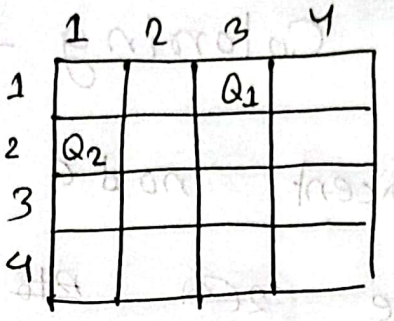


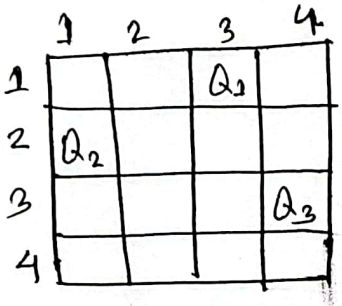


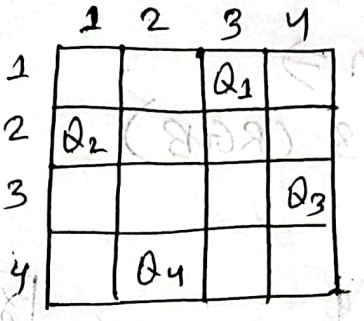


Q_1	Q_2	Q_3	Q_4
2	4	1	3









Q_1	Q_2	Q_3	Q_4
3	1	4	2

Backtracking complexity –

$T(n) = n!$

Because 1st queen ने पहले column n. 2nd queen ने दूसरे (n-1), then 3rd queen (n-2) and so on

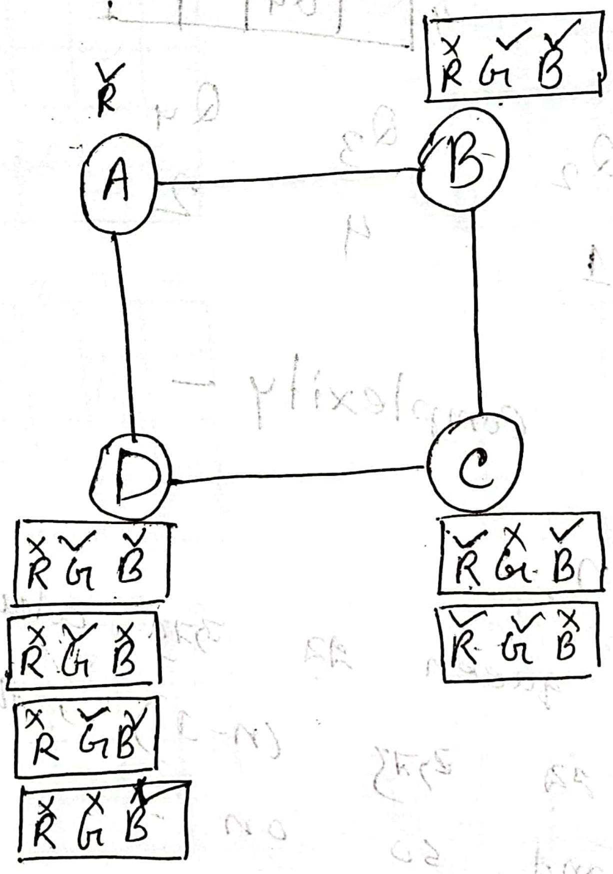
Graph Coloring —

Adjacent node same color হবে না।

Complete হলে ~~Bto~~. Box করবো।

Simulation ⇒

K = 3 (RGB)



Combinations —

(1) RGRG
(2) RGRB
(3) RGBG
(4) RBRG
(5) RBRB
(6) RBGB

String Matching Algorithm ⇒

Naive / Brute Force ⇒

* Text T ⇒ x a b c a b

Pattern P ⇒ a b c

step - 1 : [x a b] c a b X

step - 2 : x [a b c] a b ✓

step - 3 : x a [b c a] b X

step - 4 : x a b [c a b] X

Output : 1

Complexity ⇒

Text size = m

Pattern size = n

shift 0 to (n - m)

Total shift n - m + 1

Complexity = O(n - m + 1) × m

~~Rabin Carp Algorithm ⇒~~

Rabin Karp Algorithm ⇒

Keywords ⇒ Hash function, Hash code, Rolling Hash, Match hit, Spurious hit.

* Text T : x a b c a b

Pattern P : a b c

⇒ Let, ⇒ Hash Function

a = 1, b = 2, c = 3 . . . , Y = 25, Z = 26

Text T : x a b c a b

Pattern P: [a b c]

1 + 2 + 3 = 6 ⇒ Hash code

Text T: [x a b] c a b X

24 + 1 + 2 = 27

x [a b c] a b ⇒ Match Hit

1 + 2 + 3 = 6

[Hash code and Pattern same]

x a [b c a] b

2 + 3 + 1 = 6

x a b [c a b]

3 + 1 + 2 = 6

⇒ Spurious Hit

[Hash code same, pattern different]

Output : 1

Rabin Karp Algorithm এর drawback হচ্ছে spurious hit বেশি হলে performance কমে।
spurious hit কমানোর way হল Hash function strong করা।

Complexity:

Initial Hash Calculation /

Processing time $\Rightarrow O(m)$

Shifting - $O(n-m+1)$

Worst - $O(n-m+1) \times m$

Avg/Best - $O(n-m+1) \Rightarrow O(n)$

Total complexity -

Worst complexity = $O(n-m+1) \times m + O(m)$

Avg/Best = $O(n) + O(m)$

String match simulation [Rabin Karp] $\Rightarrow$

Way - 2 $\Rightarrow$

Text: T: x a b c a b

Pattern P: a b c

1 + 2 + 3 = 6

x a b c a b x

24 + 1 + 2 = 27

x [a b c] a b ✓

27 − 24 + 3 = 6

x a [b c a] b ✗

6 − 1 + 1 = 6

x a b [c a b] ✗

6 − 2 + 2 = 6

Output : 1

Approximation Algorithm :

An Approximation Algorithm is a way of dealing with NP completeness for optimization problem to determine such solutions which one close to optimal solutions in polynomial time.

C → Cost of solutions

C* → Cost of optimal solutions.

P(n) → Approximation Ratio

Maximization ⇒ $\frac{C^*}{C} \leq P(n)$

Minimization ⇒ $\frac{C}{C^*} \leq P(n)$

} $P(n) \neq 1$

Merge Sort ⇒ $O(n \log n)$

Bubble Sort ⇒ $O(n^2)$

Binary search ⇒ $O(\log n)$

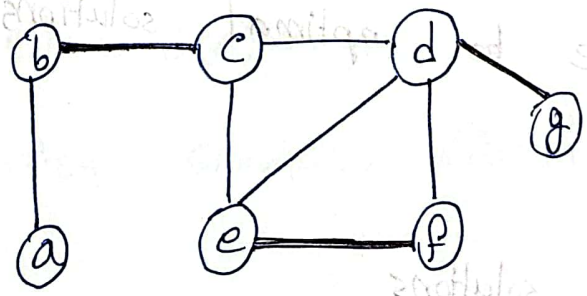
} Solving Process & time complexity known. Deterministic + Polynomial time complexity

NP Hard ⇒ Exceptional Problem ⇒ solving Process fully জানা বা এমন Problem.

↳ Non deterministic Polynomial

⇒ Exponential time complexity ⇒ 2^n [Graph coloring এমন NP Hard problem solve করার ক্ষেত্রে Polynomial time complexity বানানোর try করা হয়।

VERTEX COVER PROBLEM ⇒ [যে edge টির উপর bold করবে]



V : φ

E : {(b,c), (a,b), (c,d), (e,f), (d,e), (c,e), (d,g), (d,f)}

V: $\phi \cup \{b, c\}$

: $\{b, c\} \cup \{e, f\}$

: $\{b, c, e, f\} \cup \{d, g\}$

: $\{b, c, e, f, d, g\}$ = $\boxed{6}$ ⇒ C

☐ Vertex cover problem solve আমাদের বইতে ans করতে হবে। কারণ part approximation algorithm বুঝানোর জন্য,

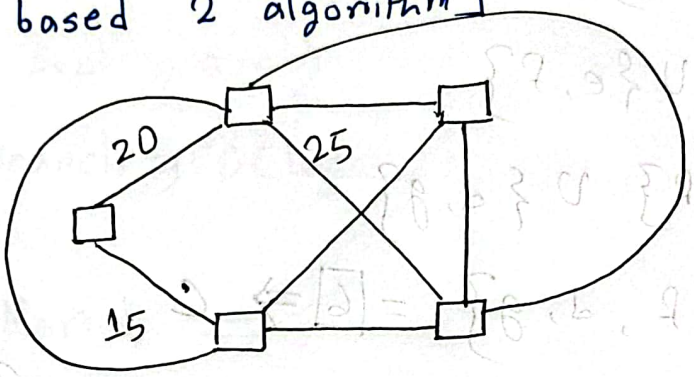
V: ϕ

V: $\phi \cup \{b, c\}$

: $\{b, c\} \cup \{d, f\}$

: $\{b, c, d, f\}$ = $\boxed{14}$ ⇒ C^*

TRAVELLING SALESMAN PROBLEM
[MST based 2 algorithm]



এটা salesman এটা city থেকে visit করে আবার আবার শুরুর city তে back করতে minimum cost এ.

Steps ⇒

(1) Combine triangle inequality [এই step question এ করার আগে (1)]

যেমন Example - এ (a, c) কে (a, b, c)

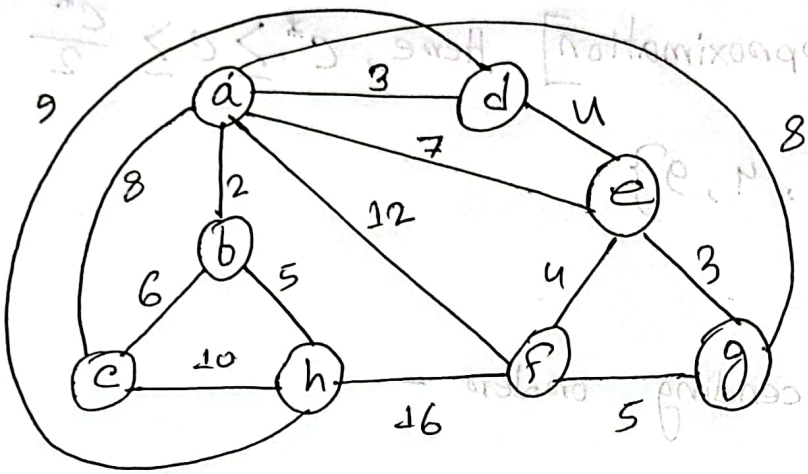
$a \rightarrow b \rightarrow c$

$dis(a,c) \leq dis(a,b) + dis(b,c)$

(2) Apply Kruskal Algorithm

(3) Apply DFS

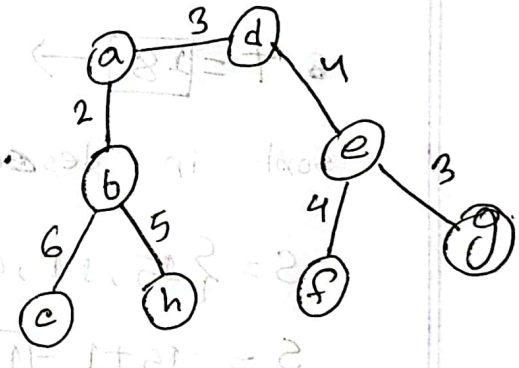
(4) Add starting node at the end of DFS Traversal.



Applying Kruskal Algorithm —

ad	3
ac	7
af	12
ab	2
gc	8
ag	8

ab 2 ✓
~~ag~~ad 3 ✓
eg 3 ✓
de 4 ✓
ef 4 ✓
bh 5 ✓
fg 5 ✗
bc 6 ✓
ae 7 ✗
ac 8 ✗
ag 8 ✗
dh 9 ✗
ch 10 ✗
af 12 ✗
hf 16 ✗



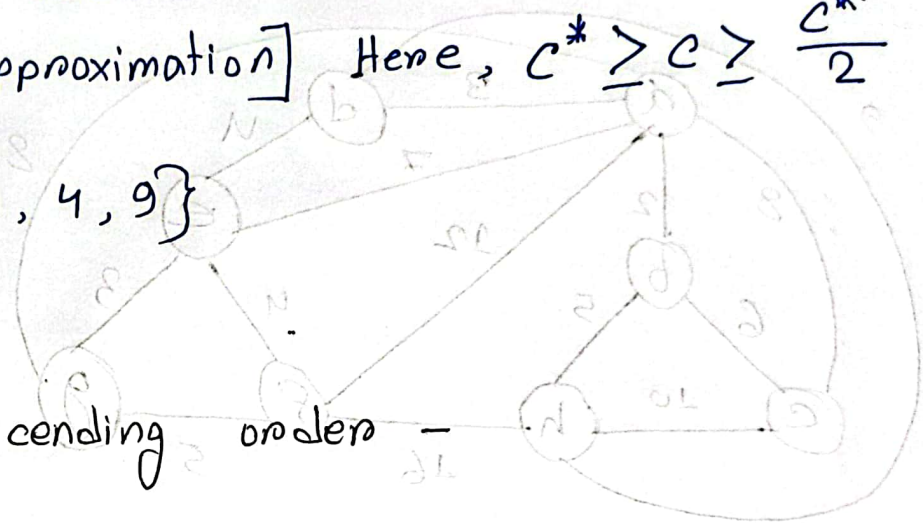
Applying DFS —

Visited — a, b, c, h, d, e, f, g, a

TSP approximate solution

Subset Sum Problem :

[$\frac{1}{2}$ Approximation] Here, $c^* \geq c \geq \frac{c^*}{2}$



* $S = \{2, 5, 1, 4, 9\}$

  $T = \{10\}$.

  Sort in descending order -

  $S = \{9, 5, 4, 2, 1\}$

  Sum $= 9 + 1 = 10$

* $S = \{5, 1, 11, 15\}$

  $t = \boxed{18} \longrightarrow c^*$

  Sort in descending order -

  $S = \{15, 11, 5, 1\}$

  $S = 15 + 1 = \boxed{16} \longrightarrow c$

$$c^* \geq c \geq \frac{c^*}{2}$$

$$\Rightarrow 18 \geq 16 \geq 9$$