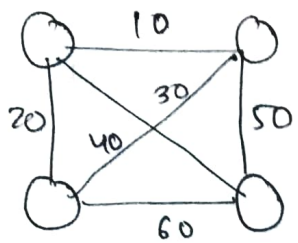
Minimum spanning Trees and examples on prims Algorithm

If graph is weight then only spanning tree is not sufficient to minimize the effort. We have to find spanning tree with min weight.

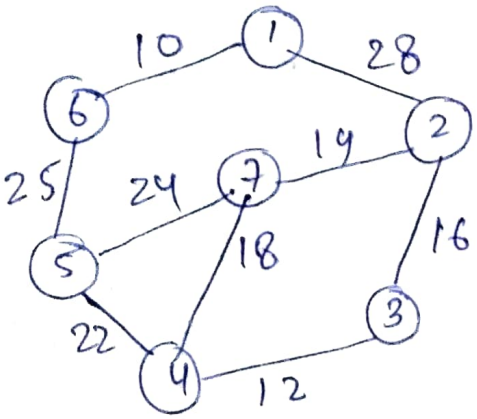
Ex

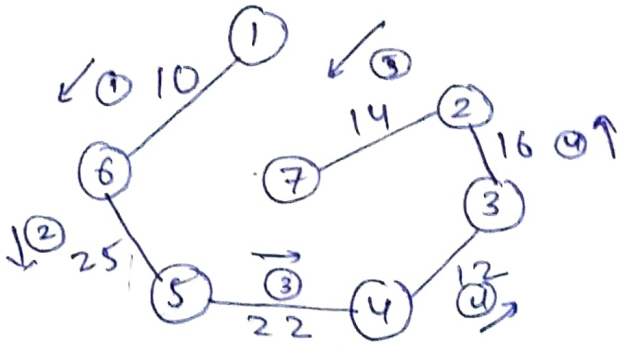


It may gives $O(n^{n-2})$ possible trees from which we can choose the efficient one. but it will go with the Time of exponential order. So to reduce the time we have two Greedy methods.

① Prim's Algorithms (Min. cost spanning Tree)

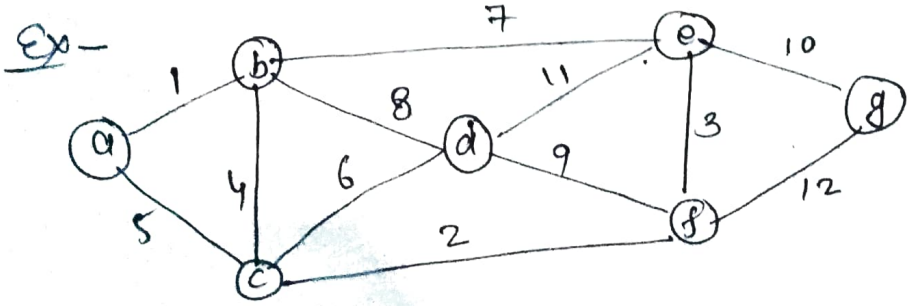
No. of edges = (n-1) in spanning Tree

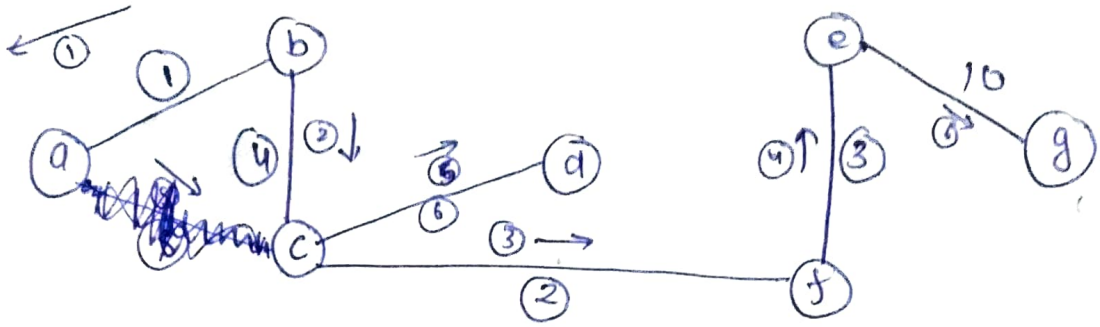




min. से start करो और जो भी Nodes Joined होते हैं उनके Neighbours में से जो min. cost में Acheive हो जाएँ उन्हें add करते जाओ।

Ex-

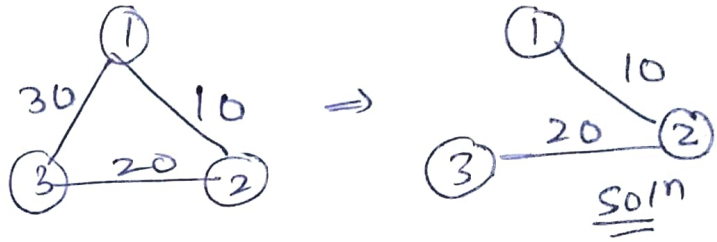




After making spanning tree find the min. weight.

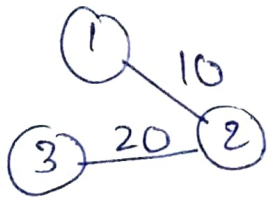
If question is given in the form of weight matrix then convert it into graph and then apply prim's mechanism.

$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{ccc} 1 & 2 & 3 \end{array} \\ \begin{bmatrix} 0 & 10 & 30 \\ 10 & 0 & 20 \\ 30 & 20 & 0 \end{bmatrix} \end{array}$$



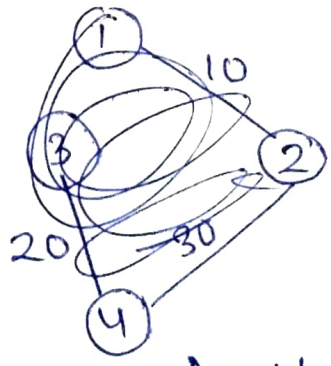
IInd method → Without constructing a graph.

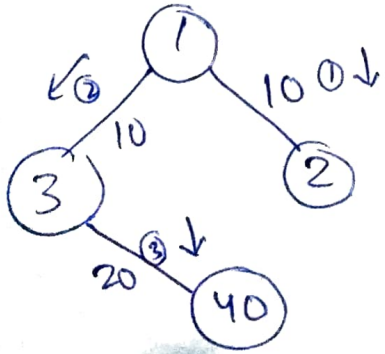
① start from 1 add the edge with min. cost



②

$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 10 & 10 & 50 \\ 10 & 0 & 40 & 30 \\ 10 & 40 & 0 & 20 \\ 50 & 30 & 20 & 0 \end{bmatrix} \end{array}$$





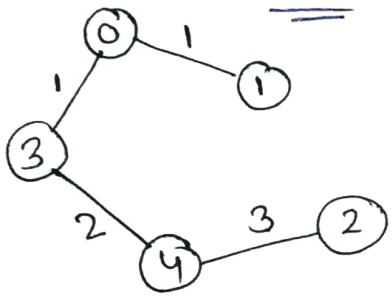
Ans

Don't add the node which is already added.

	0	1	2	3	4
0	0	1	8	1	4
1	1	0	12	4	9
2	8	12	0	7	3
3	1.	4	7	0	2
4	.4	9	3	2	0

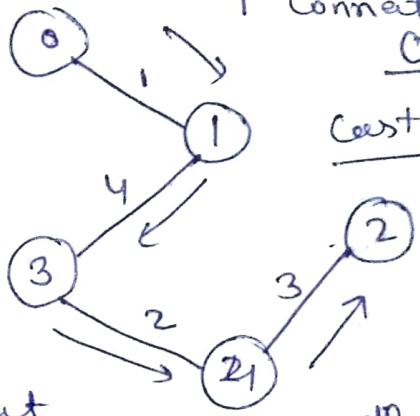
Gate-2010

Soln



min. → 7

Condition → Make a spanning tree such as '0' is a leaf | only one edge can connect to 0.

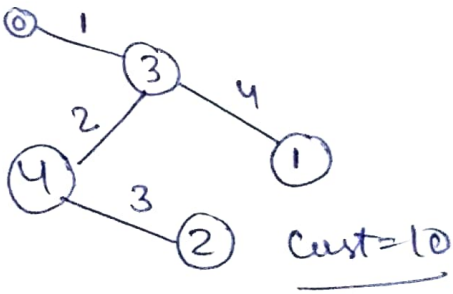


Cost=10

But with given condn.

You can choose
0 → 1
0 → 3

Two ways possibe



Cost=10

" Here both gives cost 10 but its not always necessary"

Prims Algorithm Implementation without min. heap

```
Prims ( E, cost, n, t)
// E is the set of edges. Cost in (nxn) adjacency matrix
// MST is computed and stored in array t[1:n-1, 1:2]
{
```

```
1.) let (k,l) be an edge of min cost in E;
2.) mincost = Cost[k,l];
3.) t[1,1] = k; t[1,2] = l;
4.) for (i=1 to n)
        5. if (Cost[i,l] < Cost[i,k]) then near[i] = l;
        6. else near[i] = k;
7.) near[k] = near[l] = 0;
8.) for (i=2 to n-1){
    9.) let j be an index such that near[j] ≠ 0
    10.) Cost[j, near[j]] is minimum;
    11.) t[i,1] = j, t[i,2] = near[j];
12.) mincost = mincost + cost[j, near[j]];
13) near[j] = 0;
14.) for(k=1 to n)do
15.) if ((near[k] ≠ 0 and (Cost[k, near[k]] > Cost[k, j]))
        then near[k] = j;
}
}
```

$T(n) = O(V^2)$ | $O(n^2)$

Just to read and understand such that you become habitual of reading long Algorithms. Not necessary for gate.

Prims using min heap :-

Available in Corenaun Book

$T(n) = O(E \log V)$ // normal min heap

$T(n) = O(V \log V + E)$ // Fibonnaci heap

MST_PRIMS (G, cost, r) // using min heap

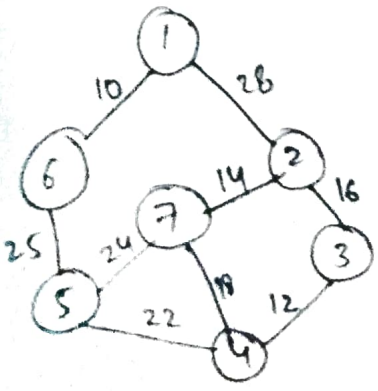
{ Heap() ← | ↓ matrix | └ root → 1st active key }

1. for each vertex $u \in G$ vertices
   {
   2. $u.key = \infty$
   3. $u.\pi = NIL$
   }
4. $r.key = 0$
5. $Q = G.vertices$ (// Q is min heap as Build_Heap procedure)
6. while ($Q \neq \emptyset$)
   {
   7. $u = Extract\_MIN(Q)$ → $O(\log v)$
   8. for each vertex 'v' adjacent to 'u'
      {
      9. if $v \in Q$ and $cost(u,v) < v.key$
         10. $v.parent = u$
         11. $v.key = w(u,v)$ → $O(\log v)$ (decrease key // key)
      }

Keys → least cost from already added values of nodes

π → Parent

$O(v \log v)$ Time for extract min & $O(v^2 \log v)$ time for decrease key



	1	2	3	4	5	6	7
key	∞	∞	∞	∞	∞	∞	∞
π	N	N	N	N	N	N	N

take $r = 1$

then, $1.key = 0$ and create a node 1

then,

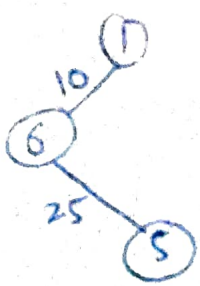
$2.key = 28$ and $6.key = 10$

$2.\pi = 1$ ⟶ $6.\pi = 1$

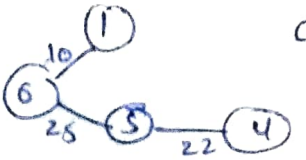
then take $6.\pi$ out bcoz it has min value of key.

then $5.key = 25$ and extract min = (5)

$5.\pi = 6$



then 4.key = 22
then 4.π = 5



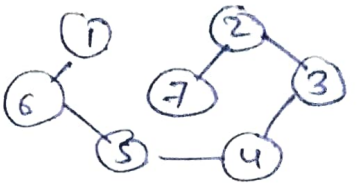
and 7.key = 24
7.π = 5

3.key = 12 3.π = 4
7.key = 18 7.π = 4

extract (3)

after extract (2)
7.key = 14
7.π = 2

extract (7)



2.key = 16
2.π = 3

In Algo. Prim's with or without heap you will get the same answer.

for decrease key in worst case there may be V adjacent nodes.
So decrease key will take $O(V^2 \log V)$

But on aggregate analysis
Decrease key = $O(E \log V)$

So,
$$T(n) = O(V \log V + E \log V + E)$$
$$= O(E \log V)$$ proved

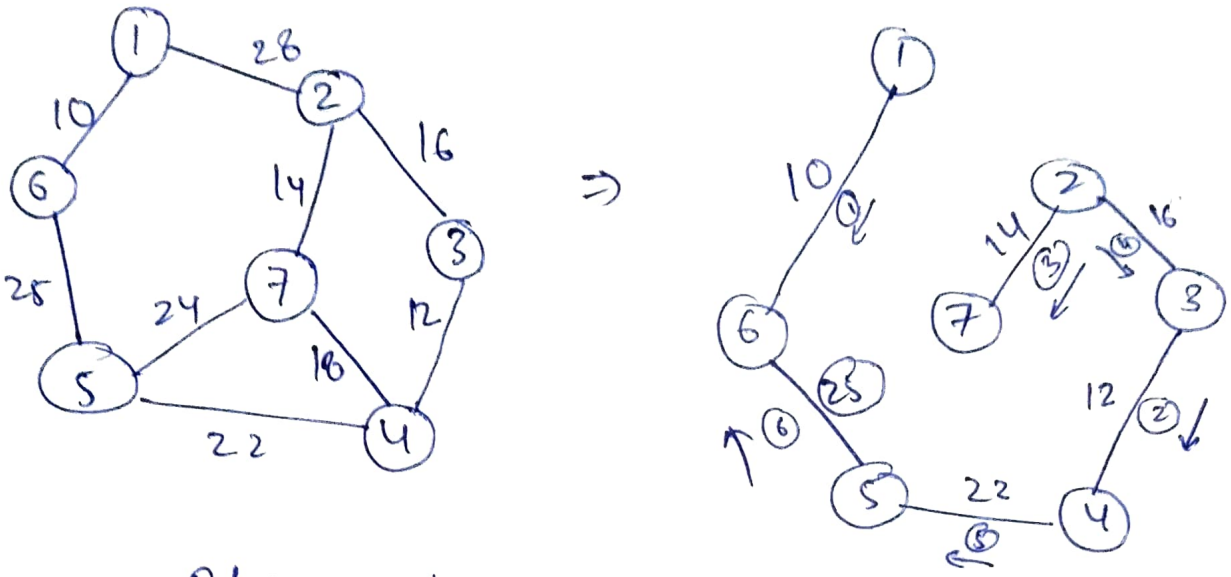
Fibonacci Heap (to read separately) → Not in Syllabus

$$T(n) = O(V \log V + E)$$

In case of dense graph $E = O(V^2)$
So, $T(n) = O(V^2)$ is better than $O(V^2 \log V)$

In case of sparse graph $E = O(V)$
$T(n) = O(V \log V)$ is better than $O(V^2)$.

Kruskal's Algorithm



Always choose the least cost edge in start. But new ~~node~~ edge can be from any way.

In prims we get one tree but in kruskal's we may get forest during execution.

Consider, cycle doesn't formed during select a minimum edge.

In prims or kruskal's if cost of different edges is different we get same tree but if cost is same on any two edge or more we may get different spanning trees.

→ In any case total cost must be same.

Q Let 'G' be an undirected graph with distinct edges weights. Let max be the edge with max. weights and min ~~with~~ be the edge with min. weight, which of the following is false?

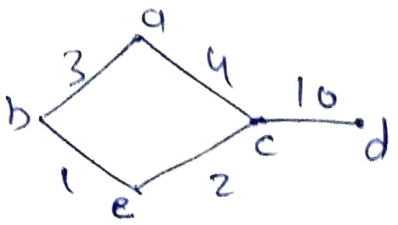
a) Every minimum spanning tree of 'G' must contain min.

b) If max e is in a minimum spanning tree, then its removal must disconnect G.

c) No min. spanning tree contains max e.

d) 'G' has a unique minimum spanning tree.

a) In min spanning tree we definitely contain the edge with min. weight.

b) 

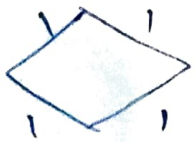
It is a worst case 10 needs to connect d Removal of 10 in a tree disconnect graph.

c) False, explaination given is point b. (Ans)

d) With all distinct ~~element~~ weight, we get a unique min. spanning tree.

If in same Question (Distint weight edge is not given then).

Multiple edge can have (mine) and (max).

a) 

→ You might have atleast one edge of mine so its true

b) True, always in situation discussion above

c) false, bcoz of true.

d) false, bcoz we may get many spanning tree.

Gate-2007 Let 'w' be the min weight.

Explaination given.

DAA Notebook 2nd

Matrix chain Multiplication (Dynamic Programming)

A(3x2) × B(2x3)

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ a_{23} & a_{33} \end{bmatrix} \quad \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \end{bmatrix}$$

A x B → (3x3) matrix

$$\begin{bmatrix} a_{11}\cdot b_{11} + a_{12}\cdot a_{21} & & \\ & & \\ & & \end{bmatrix}$$

To get a single element in (3x3) matrix we doing multiplication twice

So, total no. of times we will do the multiplication is → 3x3x2 → 18 times

So, $\begin{bmatrix} \quad \end{bmatrix}_{R \times R} \begin{bmatrix} \quad \end{bmatrix}_{R \times Q} \Rightarrow \begin{bmatrix} \quad \end{bmatrix}_{R \times Q}$

Total multiplication (scalar) = $(R \times Q \times P)$

Chain multiplication Introduction

If we want to multiply 3 matrix

$A_{2\times1}$ $B_{1\times2}$ $C_{2\times4}$

All the matrix should be compatible

$(A \times B) \times C$ or $A \times (B \times C)$ → on both case final answer will be same due to associativity

In case (1) $((A_{2\times1} B_{1\times2}) C_{2\times4})$

$A_{2\times1} \; B_{1\times2} \rightarrow 4$

$D_{2\times2} \; C_{2\times4} \rightarrow 16$

Total scalar multiplication = $4 + 16$

$= 20$

In case (2) $(A_{2\times1} (B_{1\times2} C_{2\times4}))$

$B_{1\times2} \; C_{2\times4} \rightarrow 8$

$A_{2\times1} \; D_{1\times4} \rightarrow 8$

Total scalar multiplication = $8 + 8$

$= 16$

In case (2) we save some space.

So time can be saved by the method we paranthesised the matrix.

Examples.

(1) A B C (ways to paranthesise)

(A B) C A(BC) → Both are compatible

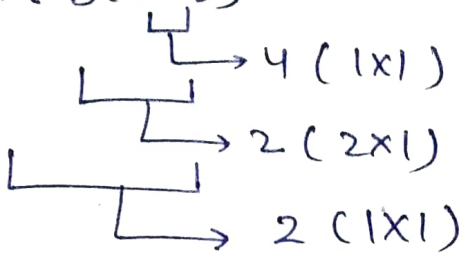
(2) A B C D

A(B(CD)) A((BC)D) (ABC)D ((AB)C)D (A(BC))D (AB)(CD)

5 ways.

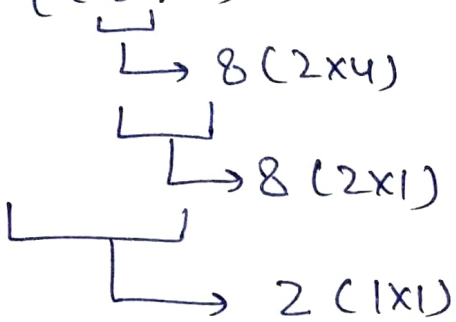
① A(B(C D))

A 1×2 B 2×1 C 1×4 D 4×1



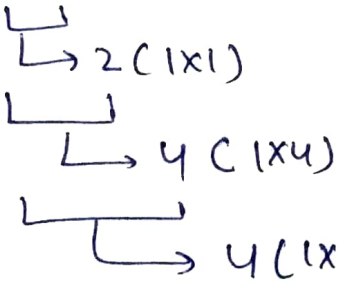
Total scalar multiplication = 8 (1×1)

② A ((BC) D)



TSM (=) 18 (1×1)

③ ((AB)C)D



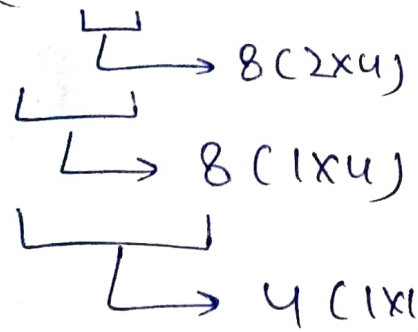
TSM = 10 (1×1)

④ (AB) (C D)

Best method in the given instance

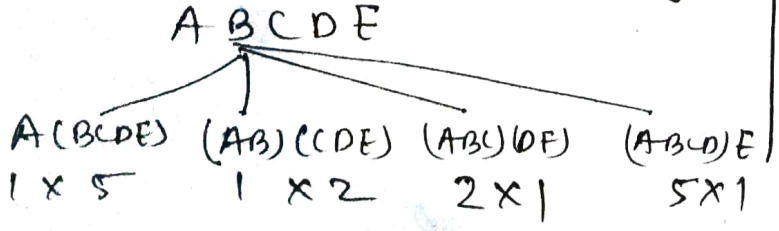
TSM = 7 (1×1)

⑤ (A(BC)) D



TSM = 20 (1×1)

for 3 matrices → 2 ways
— 4 — → 5 ways
— 5 matrices → 14 ways



In gate ask max. upto 4 matrix and order should be different to calculate TSM.

for n matrices put (n-1) in the formula

$$\frac{(2n)!}{(n+1)!\,n!}$$

for 4 matrix put n = 3
for 5 matrix put n = 4

[Same as to find the no. of BST's]

Optimal substructure and Recursive equations

↳ Devide the problem into smaller problem and then make its recursive equation to solve it.

Take n- matrices as

$A_1\ A_2\ A_3 \cdots A_n$

$p_0 \times p_1 \quad p_1 \times p_2 \quad p_2 \times p_3 \cdots p_{n-1} \times p_n$

So $[A_i (p_{i-1} \times p_i)]$

So, we have have chain of matrix

$$\underbrace{A_i\ A_{i+1}\ A_{i+2} \cdots A_j}_{\text{(optimal substructure)}} = \underbrace{(A_i\ A_{i+1} \cdots A_k)(A_{k+1} \cdots A_j)}_{\text{Parenthesize the optimal substructure (splitting)}}$$

$$\boxed{(i \le k \le j)}$$

$$\underbrace{(A_i\ A_{i+1} \cdots A_k)}_{(p_{i-1} \times p_k)} \underbrace{(A_{k+1} \cdots A_j)}_{(p_k \times p_j)} + \boxed{(p_{i-1}\ p_k\ p_j)}$$

(TSM)

Recursive Solution:

$m[i,j]$ → min. no. of matrix scalar multiplication in order to parenthesize $A[i] \rightarrow A_j$.

$[A_i A_{i+1} \cdots A_j]$

$$m[i,j] = \begin{cases} i = j \text{ then } 0; \\ \min\{m[i,k] + m[k+1,j] + p_{i-1} \times p_k \times p_j\} \quad (i \le k < j) \end{cases}$$

k can have any value in range $(i \rightarrow j-1)$.

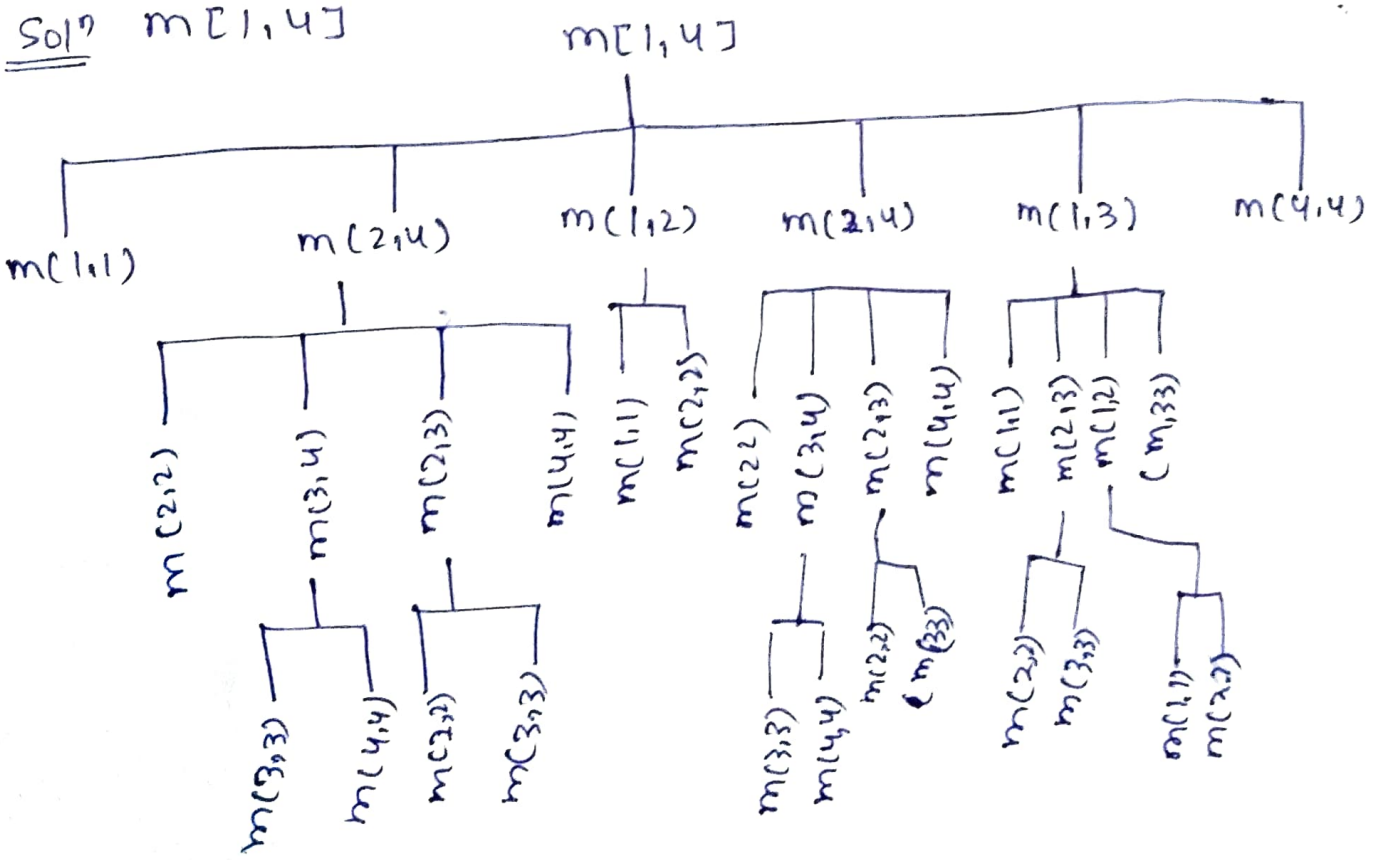
So total no. of possibilities are $(j-i)$

After finding optimal substructure and recursive eqn, we need to find overlapping subproblems.

Recursion Tree

Ex $A_1 A_2 A_3 A_4$

Soln $m[1,4]$



In worst case depth → $O(n)$

and no. of nodes → $O(2^n)$ Binary tree

It is taking exponential time and in Algorithm we generally avoid the Algorithms taking exponential time.

In this example the no. of recursive calls is $O(2^n)$.

But using dynamic programming we can solve it in very less recursive calls.

Bottom-UP Implementation :-

function calls invoked (subproblems)

(n) ← (1 1) (2 2) (3 3) (4 4) → 4 of size 1

(n-1) (1 2) (2 3) (3 4) → 3 of size 2

(n-2) (1 3) (2 4) → 2 of size 3

1 (1 4) → 1 of size 4

Total no. of subproblems are → $10 = \frac{n(n+1)}{2}$

$= O(n^2)$

In order to solve,

$A_1 A_2 A_3 \dots A_n$

total no. of subproblems to solve $O(n^2)$

Let us suppose we want to multiply 4 matrices as—

A_1	A_2	A_3	A_4
1×2	2×1	1×4	4×1
$p_0\ p_1$	$p_1\ p_2$	$p_2\ p_3$	$p_3\ p_4$

0 (11)	0 (22)	0 (33)	0 (44)
2 (12)	8 (23)	4 (34)	
10 ~~14~~ (13)	6 (24)		
7 (14)			

$(1,3) = \min \begin{cases} (1,1)+(2,3)+p_0p_1p_3 = 16 \\ (1,2)+(3,3)+p_0p_2p_3 = 10 \\ = 4 \end{cases}$

$(2,4) = \min \begin{cases} (2,2)+(3,4)+p_1p_2p_4 = 6 \\ (2,3)+(4,4)+p_1p_3p_4 = 16 \end{cases}$

$= 6$

$$(1,4) = \begin{cases} (1,1)+(2,4)+p_0p_1p_4 = 6+2 = 8 \\ (1,2)+(3,4)+p_0p_2p_4 = 6+1 = 7 \\ (1,3)+(4,4)+p_0p_3p_4 = 4+0 = 8 \end{cases}$$

Above method is purely tabular, No recursion.

No. of entries $= O(n^2)$

To compute $(1 \ldots n)$ we have n splits and we have compare each splits one by one

To time table to compute the splits $= O(n)$

So, total $T(n) = O(n^3)$ (3)

Space Complexity $= O(n^2)$ → space is taken by the table

Matrix chain (b)

$A_{1\times2}$ $B_{2\times1}$ $C_{1\times4}$ $D_{4\times4}$

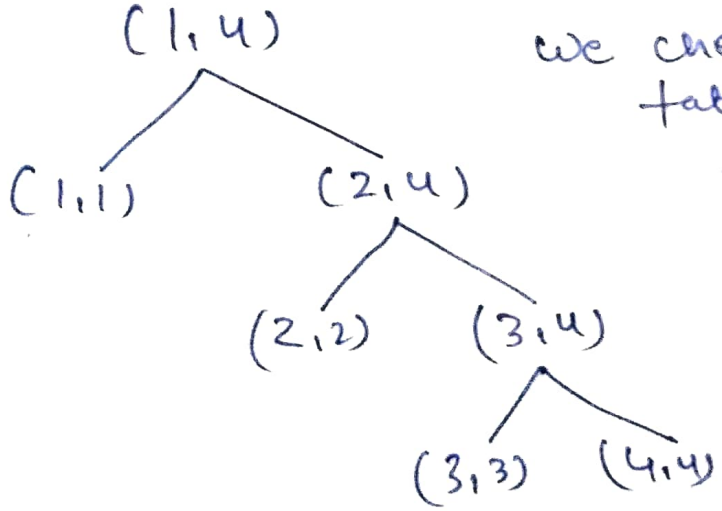
b → array of values in order of matrix (1,2,4).

m [array of elements have min. value]

S [array that save the answer].

```
Matrix-chain(p) {
1. n = p.length - 1
2. let m[1..n, 1..n] and s[1..n-1, 2..n] be new tables
3. for i = 1 to n {
4.     m[i,j] = 0 }
5. for l = 2 to n
6.     for i = 1 to n-l+1 {
7.         j = i+l-1
8.         m[i,j] = ∞ }
9.     for k = i to j-1 {
10.        q = m[i,k] + m[k+1,j] + pi-1 pk pj
11.        if q < m[i,j]
12.            m[i,j] = q
13.            s[i,j] = k
14. return m and s
```

Top-Down Approach (Memoization)



we check the value in table before the function call

No. of functions calls in both are case are same.

This method is using the recursion calls.

```
Memoized-Matrix chain(p) {
    n = p.length - 1
    Let m[1...n, 1...n] be a new table
    for i = 1 to n
        for j = 1 to n
            m[i, j] = ∞
    return Look up-chain (m, p, 1, n) }

Look up-chain (m, p, i, j) {
    if m[i, j] < ∞
    return m[i, j]
    if i == j
        m[i, j] = 0
    else for k = i to j-1
    q = Look up Chain(m, p, i, k) + loop up-chain (m, p, k+1, j)
        + p_{i-1} p_k p_j
    if q < m[i, j]
        m[i, j] = q
    return m[i, j]
}
```

Total no. of distinct function calls $\rightarrow O(n^2)$

Time taken in value evaluated of fn $\rightarrow O(n)$

Total $\boxed{T(n) = O(n^3)}$

and $\boxed{S(n) = O(n^2)}$

Example M_1 M_2 M_3 M_4

10×100 $\quad 100 \times 20$ $\quad 20 \times 5$ $\quad 5 \times 80$

11	22	33	44
0	0	0	0
12	23	34	
20000	10000	8000	
13	24		
15000	50000		
14			
19000			

$(1,3) \rightarrow (12)3$ or $1(23)$

$(1,3) = \min\{ (1,1) + (2,3) + 5000 \text{ (0, 10000)}, \; (1,2) + (3,3) + 1000 \text{ (20000, 0)} \}$

$24 = \{ (22) + (34) + 16000 = 8000 \text{ (0, 8000)}, \; (23) + (44) + 40000 = 50000 \text{ (10000, 0)} \}$

$(1,4) \rightarrow \min$

- $(1,1) + (2,4) + 80{,}000 = 130000$ (0, 50000)
- $(12) + (34) + 16{,}000 = 44000$ (20,000, 8000)
- $(1,3) + (4,4) + 4000 = 19000$ (15000, 0)

So Ans $\boxed{((1)(23))(4))}$

sub

Longest Common Subsequence

R A V I N D R A

1 2 3 4 5 6 7 8

{1, 2, 3, 4} RAVI

{1, 3, 5, 8} RVNR

{2, 7} AR

{1, 2, 5, 8} RANA

{} $\emptyset$.

Difference between Subsequence & Substrings.

↓

Substring is Contiguous selection of index but in Subsequence you could miss some index in middle but all the Indexes should be in

No. of subsequence possible = 2^m

m → no of character present in string.

Proof

$2 \times 2 \times 2 \times \ldots \times 2 \rightarrow 2^m$

$C_1\ C_2\ C_3 \ldots C_m$

Each character have two possibilities, either selected or missed out in subsequence or substring

Example → DNA comprises four types of molecules (A, G, T, C)

Take two different DNA strings as-

D_1: AGCCTCAGT

D_2: GCCT

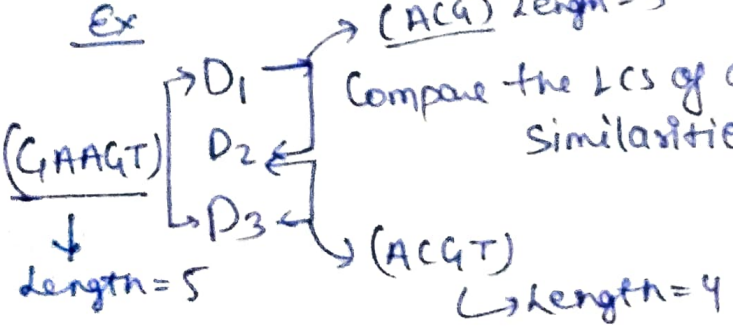
If D_1 and D_2 are equal then 100% similar

If D_2 is substring of D_1 then very much similar

and If D_1 and D_2 have no common element then we try to find how much similar both are?

→ Check Longest Common Subsequence.

Ex



So after comparing all the possible subsequences of given pair we can say D_1 and D_3 are having most similar DNA.

Longer the common subsequence, more similar they are

Ex-

$S_1 \rightarrow$ RAVINDRA $\rightarrow 2^8$ subsequence

$S_2 \rightarrow$ AJAY $\rightarrow 2^4$ subsequence.

CS $\rightarrow$ {AA}, {A}, { }

LCS $\rightarrow$ {AA} (2) $\rightarrow$ Length.

If S_1 have leng. m then total subsequence $= 2^m$

— S_2 — n — $= 2^n$

(1) To find all the subsequence of $S_1 = 2^m = O(2^m)$

(2) find for each subsequence whether it is a subsequence of 'B' $= O(n2^m)$

(3) find the longest among common subsequences $\rightarrow O(2^m)$

Optimal Substructures and Recursive Equation

Let us take two string X and Y as

$X \rightarrow x_1 x_2 x_3 \cdots x_n$

$Y \rightarrow y_1 y_2 y_3 \cdots y_m$

$n \neq m$ or $n = m$ — Both can be possible

then, there are several cases to find subsequence

(1) If $x_n = y_m$ then reduce the size of both X and Y by 1 and start matching again

(2) If $x_n \neq y_m$ then, there may be two cases -

i) Reduce the size of x_n keeping the size of y_m same

ii) Reduce the size of y_m keeping the size of x_n same.

Above cases can be represented as —.

$$C[i,j] = \begin{cases} 0 \; ; & i=j=0 \\ 1+C[i-1,j-1] \; ; & i,j>0 \text{ and } x_i = y_j \\ \max(C[i-1,j], C[i,j-1]) \; ; & i,j>0 \;\&\; x_i \neq y_j \end{cases}$$

It is for the element matched at last position

Recursion tree and unique sub problems

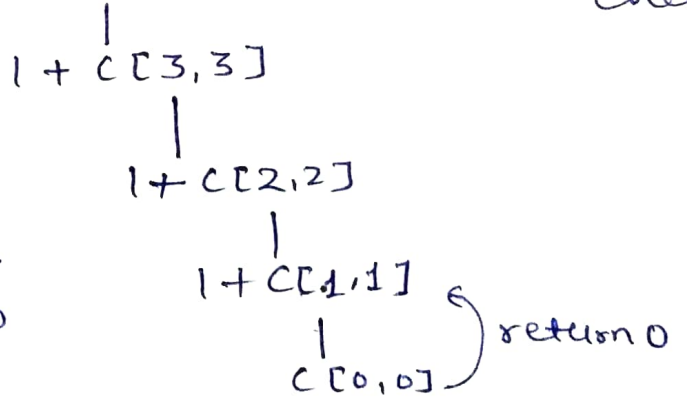
We are analysion in best and worst case both.

(1) Best Case

$X = \{A, A, A, A\}$

$Y = \{A, A, A, A\}$

In matrix multiplication we were not checking condition but here we are checking.



If there is a match we are calling 1 function and if no match we need to call two function.

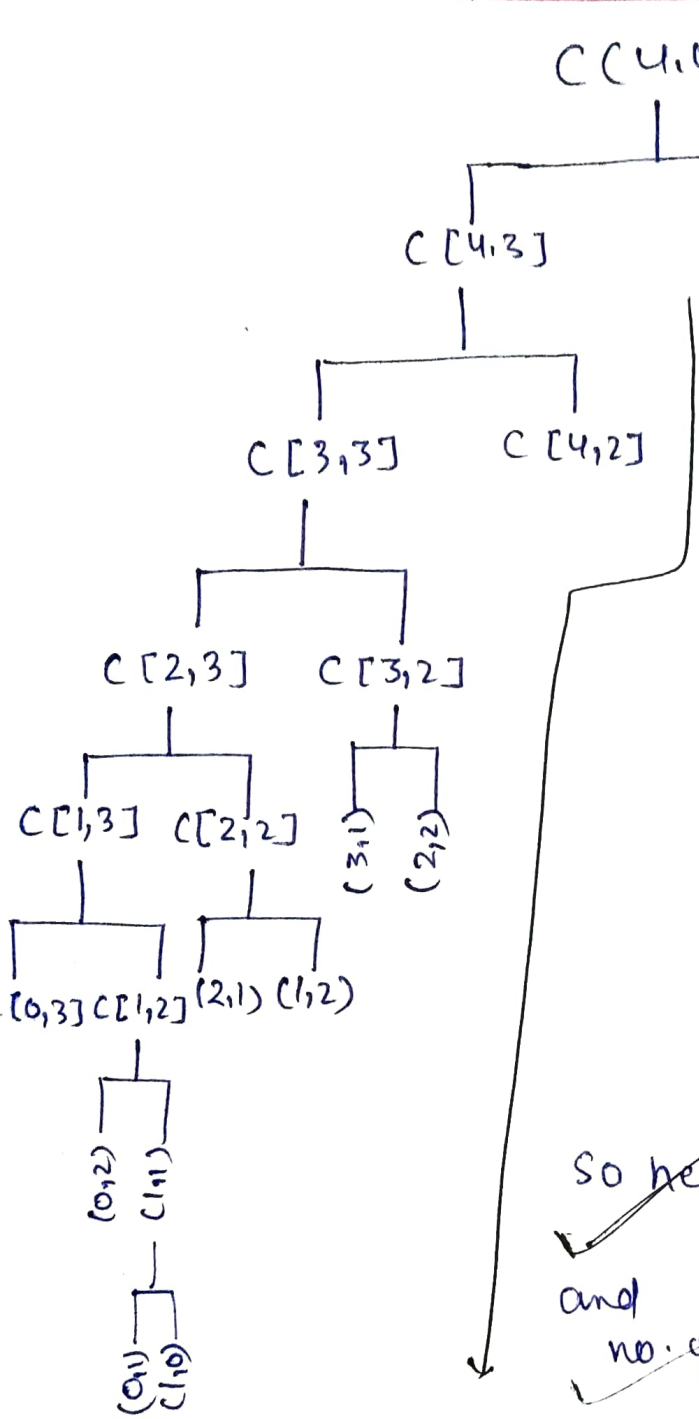
$T(n) = S(n) = O(n)$

So we are not interested

(2) Worst Case

$X = [A, A, A, A]$

$Y = \{B, B, B, B\}$



Tree is growing in a definit manner as -

C[n, m]
C[n-1, m]
C[n-1, m-1]
C[n-2, m-1]
C[n-2, m-2]
C[0, -]
C[-, 0]

So height of tree is of O(n+m).

and no. of nodes in a tree is of $O(2^{n+m})$

* No. of nodes in recursive tree in worst Case → (To find or search)

So, $T(n) = O(2^{m+n})$

But in the tree some subproblems are repeated and no. of unique subproblems = (m x n).

Problems which are not overlapping

(3,3), (2,3), (1,3), (0,3) and so on = 9

on discarding (0,_) elements

9 problems = $\begin{bmatrix} (3,3) & (3,2) & (3,1) \\ (2,3) & (2,2) & (2,1) \\ (1,3) & (1,2) & (1,1) \end{bmatrix}$

16 Problems (1,0) (2,0) (3,0)] 9+7=16

(0,0) (0,1) (0,2) (0,3)

O(m×n) without including zero

O((m+1)×(n+1)) also O(m×n) on including zero

Bottom-up table approach

↳ T(n) = S(n) = O(m×n)

In memoized method also

$$T(n) = S(n) = O(m \times n)$$

because there also only recursive function called which are unique.

Ex - Y [A C A]

X \ Y	0	1 (A=A)	2 (C≠A)	3 (A=A)
0	0	0	0	0
1 A	0	1	1	1
2 A	0	1	1	2
3 B	0	1	1	2

→ return 2

X (A A B B)

Y (C C B D B)

X (A A B)

Y (A C A)

⇒ for match compute (1 + diagonal element)

⇒ for no match compute (max of upper and left side element)

So, longest common subsequence is of length 2.

To find subsequence

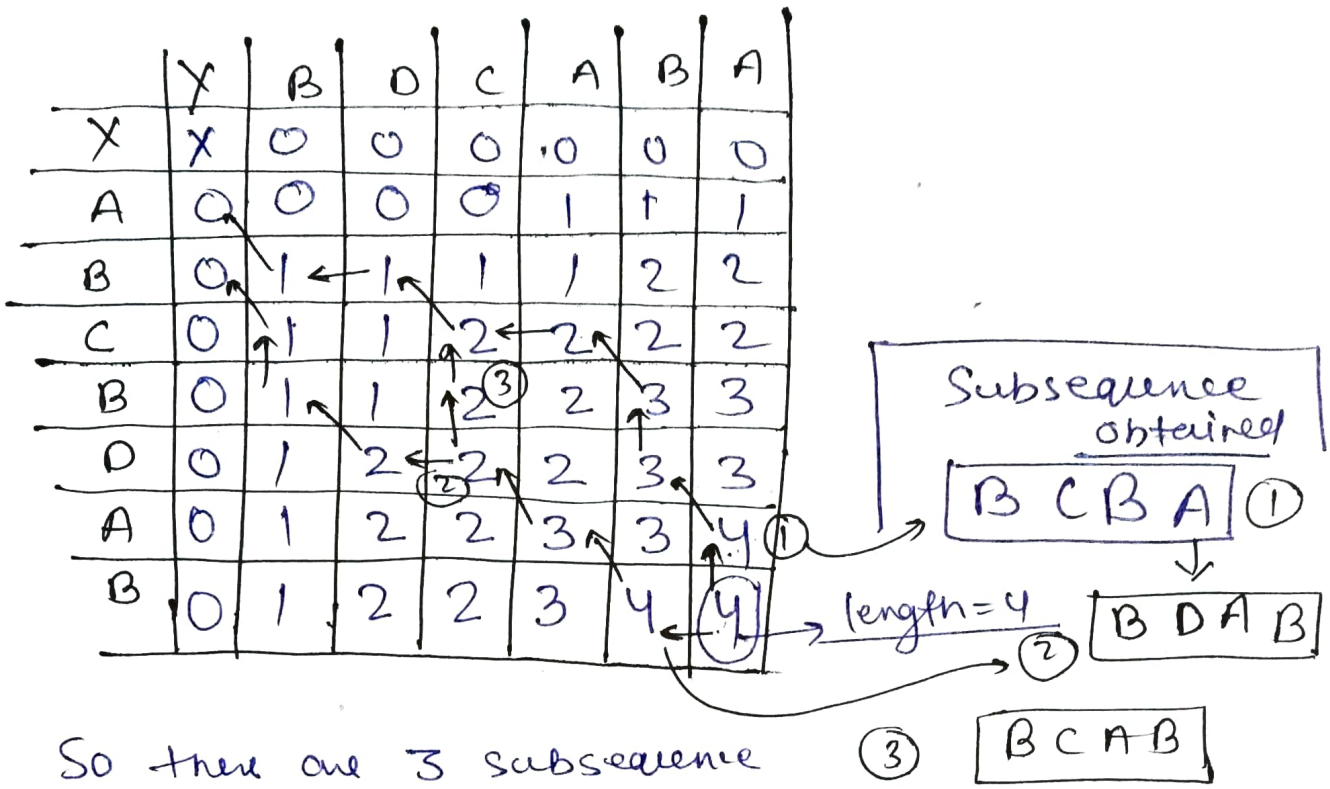
Check the element from where you arrive to last element (3,3) here it is (2,3) then check the same from (2,3) here (1,2) then (1,1) for (1,2) and in the transition elements if there is a match write it and find the sequence

अगर समझ न आए तो Video को Least 2 min में दुबारा देखो

DAA → 2 Cs → example 1

Example-2 $X = \{A\ B\ C\ B\ D\ A\ B\}$

$Y = \{B\ D\ C\ A\ B\ A\}$



So there are 3 subsequence of length 4.

Bottom-UP Dynamic Programming Approach!-

```
LCS (X, Y) {
    m = x.length
    n = y.length
    let C[0...m, 0...n] be a new table
    for i = 0 to m-1 {
        C[i, 0] = 0 }
    for (j = 0 to n-1 {
        C[0, j] = 0 }
    for (i = 1 to m) {
        for j = 1 to n {
            if xi == yj
                C[i, j] = 1 + C[i-1, j-1]
            else
                C[i, j] = max (C[i-1, j], C[i, j-1]) } }
    return C;
}
```

$T(n) = O(n \times m)$

bcoz Total $O(n \times m)$ entries are to be filled and each entry filling is taking constant $O(1)$ time.

$S(n) = O(m \times n)$