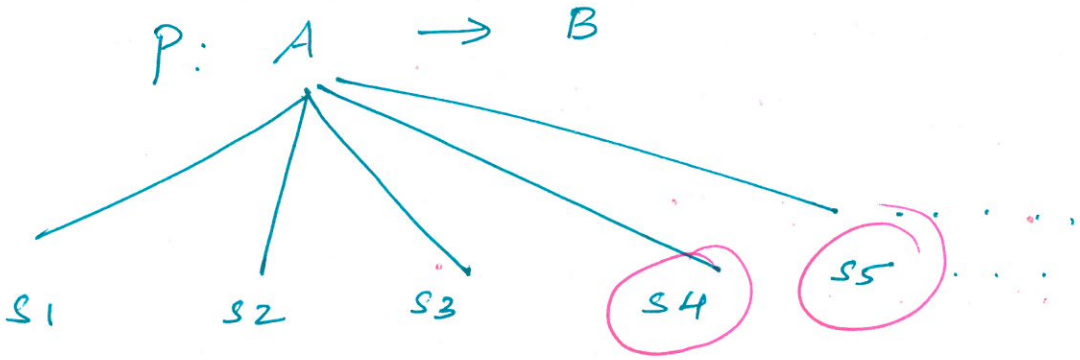
UNIT - III - Greedy and Dynamic Programming.

Syllabus

Introduction - Examples of problems that can be solved using greedy and dynamic approach - Huffman coding using greedy approach, Comparison of brute force and Huffman method of encoding - Knapsack problem using greedy approach, Complexity derivation of knapsack using greedy - Job greedy. Kruskal's algorithm - Minimum spanning tree - Tree Traversals - Minimum spanning tree - Prims algorithm - Introduction to Dynamic programming - 0/1 Knapsack problem - Complexity calculation of knapsack problem - Matrix chain multiplication using dynamic programming - complexity - Longest Common subsequence using dynamic programming - Explanation of LCS with an example - Optimal Binary search tree using dynamic programming

Greedy Method

It is one of the strategy for solving optimization problem. It is a problem which demands or which requires either minimum result or maximum result.



→ A needs to go to B place with constraint that it should go within 12 hrs. That, which satisfy the constraint is called feasible solution.

→ If I want to go to this place with minimum cost then it is called as optimal solution

→ For any problem, there can be only one optimal solution.

→ If a problem requires either minimum or maximum result is called as optimization problem.

→ Greedy Method is used to solve optimization problems.

The strategies used for solving optimization problems are

* Greedy Method
* Dynamic programming
* Branch & Bound

Greedy Method

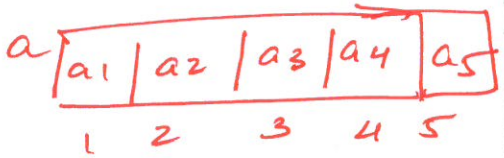
→ A problem which should be solved in stages

→ For each stage we will consider one input from a given problem and if that input is feasible then we will include it in the solution. By including all the feasible solutions we will get an optimal solution

General Greedy Method

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

n = 5



Example : → Buying Best Car by choosing based on individuals requirements

→ Hire some people to recruit for an organization.

HUFFMAN CODING

→ Huffman coding is a compression technique.

→ It is used for reducing the size of data or message

Examples: 1) files to compressed files

2) n/w → Compressed to reduce ~~data~~ cost of transmission

3)

To be learnt

↳ How a normal message is sent & what will be the cost?

↳ What is fixed size encoding?

↳ What is variable size encoding? (Huffman coding).

Problem

Message – B C C A B B D D A E C C B B A E D D C C

Length = 20

↳ ascii code is used ~~to~~ for sending english alphabets.

ASCII → 8 bits

A	65	01000001 (binary form)
B	66	01 0000 10
C	67	⋮
D	68	⋮
E	69	⋮

length = 20 × 8 = 160 bits.

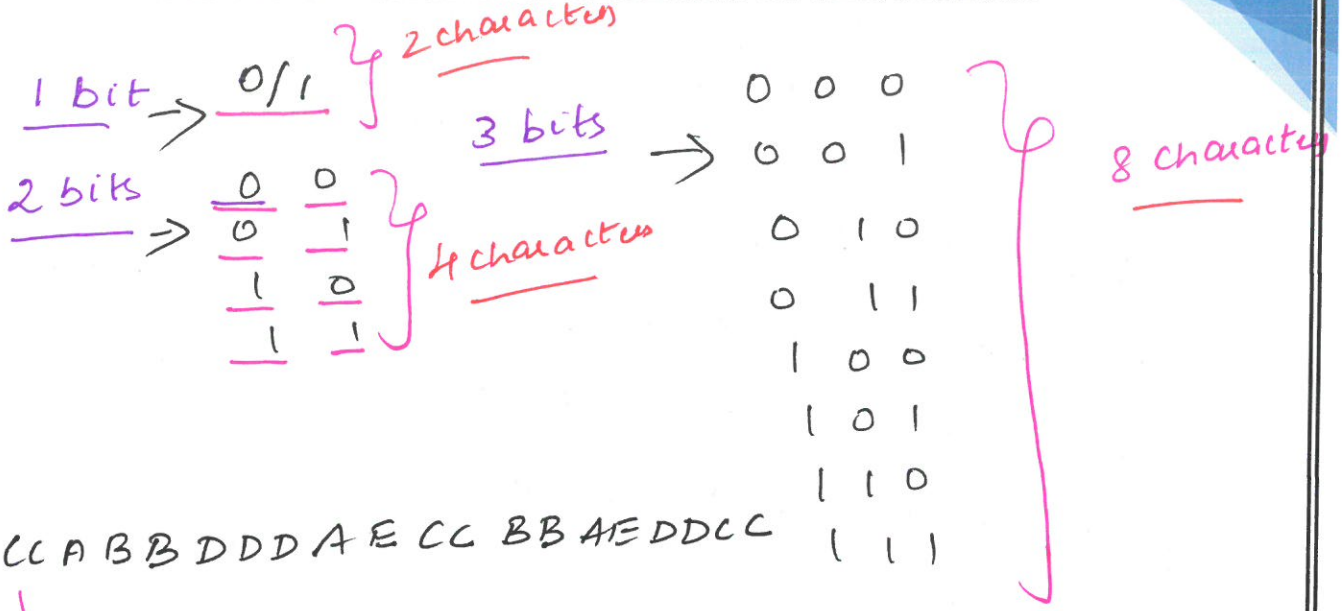
Size of the message = 160 bits.

fixed size encoding

Note:
8 bits are needed for 128 characters

Character	count / frequency		code
A	3	3/20	000
B	5	5/20	001
C	6	6/20	010
D	4	4/20	011
E	2	2/20	100
	20		

own three bit codes.



B CC A B B D D D A E CC BB AE DDCC

↓ \

001 010 .....

↳ 20 × 3 = 60 bits

How the receiver will decode? How can I remember the codes for each alphabets! So I should also have the table to decode the message.

5 × 8 bits — characters (ASCII).

5 × 3 bits → codes

Msg — 60 bits (40 + 15)

Table — 55 bits

115 bits

from 160 to 115 bits reduced using fixed size.

Huffman Coding.

→ Huffman says that we dont have to take fixed size codes for the alphabets. Some characters or alphabets may be appearing few times lesser & few more number of times.

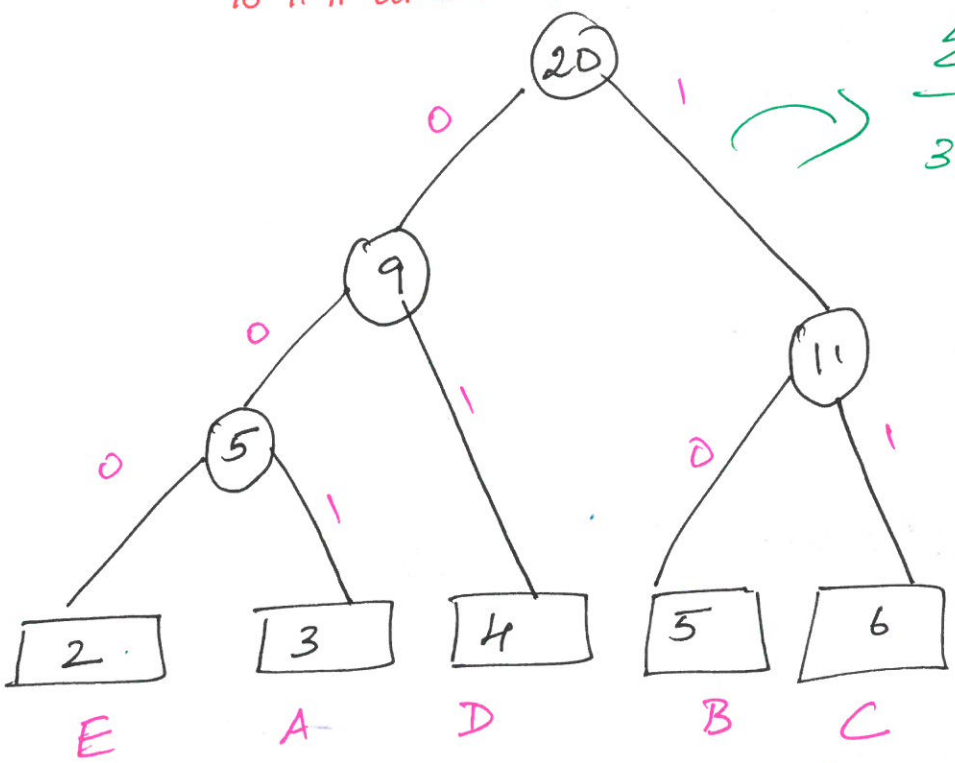
→ So if we use small size code for more appearing characters then the size of the entire message will be definitely reduced.

→ Huffman code follows optimal merge pattern.

→ Own code should arrange in to be generated, we count. sort the count of the characters. increasing order of their

Message - B C C A B B D D A E C C B B A E D D C C

10 11 11 001 10 10 01 01 001 .....



$\Sigma d_i * f_i$

$3\times2 + 3\times3 + 2\times4 + 2\times5 + 2\times6$

$= 45$ //.

Char	Count	code	
A	3	001	$3\times3 = 9$
B	5	10	$5\times2 = 10$
C	6	11	$6\times2 = 12$
D	4	01	$4\times2 = 8$
E	2	001	$2\times3 = 6$
	20		45

Message for encoding = 45 bits.

Chart / Table / Tree for decoding should also be considered.

ASCII 5×8 = 40 bits

Codes = 12 bits

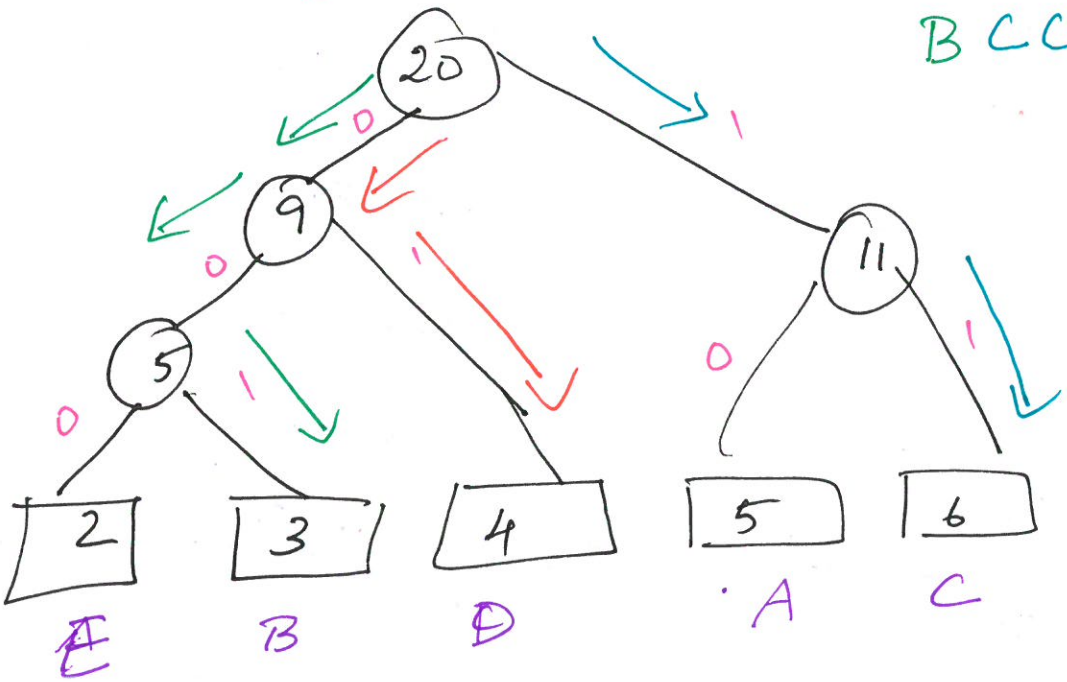
Table 52 bits

Total 45 + 52 = 97 bits

Decoding using Huffman.

Message : B C C D A C C B D A B C C D E A A E D A

001 11 11 01 00 11 11 . . . . .



B C C D . . . .

Huffman Coding Algorithm

Create a priority queue Q consisting of each unique character

Sort them in ascending order of their frequencies

for all the unique characters:

Create a newNode

extract minimum value from Q and assign it to leftchild of newnode

extract minimum value from Q and assign it to rightchild of newnode

Calculate the sum of these two minimum values and assign it to the value of newnode

Insert this newnode into the tree

return rootNode.

Time Complexity.

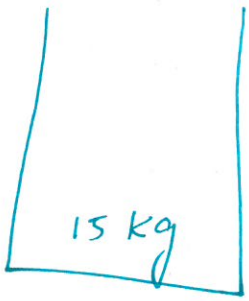
→ Moving an element from root to leaf requires O(log n) and this is done for n/2 elements

→ Sorting and comparisons total time is O(n log n) worst case.

Knapsack Problem

- Fractional Knapsack (Greedy)
- 0/1 Knapsack (Dynamic programming)

Fractional Knapsack Problem



Problem statement:

How you will select the items so that you will get maximum profit

Maximization problem

Max Profit

objects:	1	2	3	4	5	6	7
Profit (P):	5	10	15	7	8	9	4
weight (w):	1	3	5	4	1	3	2

Objects	Profit (P)	Weight (w)	Remaining weight
3	15	5	15-5=10
2	10	3	10-3=7
6	9	3	7-3=4
5	8	1	4-1=(3)
4	$7 \times \frac{3}{4} = 5.25$	3	3-3=0

Total Profit = 47.25

Minimum Weight

Objects	Profit (P)	Weight (w)	Remaining weight
1	5	1	15 - 4 = 14
5	8	1	14 - 1 = 13
7	4	2	13 - 2 = 11
2	10	3	11 - 3 = 8
6	9	3	8 - 3 = 5
4	7	4	5 - 4 = 1
3	$15 \times \frac{1}{5} = 3$	1	1 - 1 = 0

Total Profit = 46

Maximum P/W Ratio

Objects:	1	2	3	4	5	6	7
Profit (P):	5	10	15	7	8	9	4
Weight (w):	1	3	5	4	1	3	2
(P/w) :	5	3.3	3	1.75	8	3	2

Objects	Profit (P)	Weight (w)	Remaining weight
5	8	1	15-1 = 14
1	5	1	14-1 = 13
2	10	3	13-3 = 10
3	15	5	10-5 = 5
6	9	3	5-3 = 2
7	4	2	2-2 = 0

Total Profit = 51

Maximum profit is attained by Profit/Ratio. This is the fractional Knapsack which is solved by Greedy method.

Algorithm

Algorithm: Greedy-Fractional-knapsack (w[1...n], p[1...n], W)

```
for i = 1 to n
do x[i] = 0
weight = 0
for i = 1 to n
    if weight + w[i] ≤ W then
    x[i] = 1
    weight = weight + w[i]
    else
      x[i] = (W - weight) / w[i]
      weight = W
      break
return x
```

Analysis

If the provided items are already sorted in descending order of $\frac{Pi}{wi}$, then the whileloop takes time O(n). Therefore the total time including the sort is O(n log n).

Example

Weight (Wi)	7	3	4	5
Profit (Pi)	42	12	40	25

Bag capacity (m) = 10 {constraint - $\sum x_i w_i \leq m$}

Pi/Wi	6	4	10	5
x_i $(0 \leq x \leq 1)$	6/7	0	1	0

10 − 4 = 6

6 − 6 = 0

$\sum x_i w_i = (6/7)*7 + 0*3 + 1*4 + 0*5 = 10$

$\sum x_i P_i = (6/7)*42 + 0*12 + 1*40 + 0*25 = 76$

Knapsack Using Brute force Technique (Exhaustive search)

Capacity (W) = 10

S.No	1	2	3	4
Weight (wi)	7	3	4	5
Value (vi)	$42	$12	$40	$25

Subset	Total Weight	Total value
ϕ	0	$0
{1}	7	$42
{2}	3	$12
{3}	4	$40
{4}	5	$25
{1,2}	10	$54
{1,3}	11	Not feasible
{1,4}	12	Not feasible
{2,3}	7	$52
{2,4}	8	$37
{3,4}	9	$65
{1,2,3}	14	Not feasible
{1,2,4}	15	Not feasible
{1,3,4}	16	Not feasible
{2,3,4}	12	Not feasible
{1,2,3,4}	19	Not feasible

Total Items (n) = 4

Time Complexity $O(n) = 2^4 = 16$.

$O(n) = 2^n$ //.

0/1 Knapsack Problem using Dynamic Programming.

$m = 8 \quad P = \{1, 2, 5, 6\}$

$n = 4 \quad w = \{2, 3, 4, 5\}$

V

Pi	wi
1	2
2	3
5	4
6	5

	0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0	0
1	0	0	1	1	1	1	1	1	1
2	0	0	1	2	2	3	3	3	3
3	0	0	1	2	5	5	6	7	7
4	0	0	1	2	5	6	6	7	8

$V[i,w] = \max\{V[i-1, w], V[i-1, w-w[i]] + P[i]\}$

$x_1 \quad x_2 \quad x_3 \quad x_4$

$\{0 \quad 1 \quad 0 \quad 1\}$

$8 - 6 = 2$ PROFIT

$2 - 2 = 0$

$$V[4,1] = \max\{V[3,1],\ V[3,1-5] + 6\}$$

$3 - 4$ → undefined

Note

i - row

w - column

V[i-1] - prev row value

w[i] - wt of an object

P[i] - profit of an object

Algorithm

Algorithm DP_Binary_Knapsack (V, N, M)

// Description: Solve binary knapsack problem using dynamic programming

// Input : Set of items x, set of weight N, Profit of items V and Knapsack capacity M

// Output : Array V, which holds the solution of problem

```
for i ← 1 to n do
    V[i, 0] ← 0
end
for i ← 1 to M do
    V[0, i] ← 0
end
for i ← 1 to n do
    for j ← 0 to M do
        if w[i] ≤ j then
            V[i, j] ← max{V[i-1, j], v[i] + V[i-1, j - w[i]]}
        else
            V[i, j] ← V[i-1, j]  // w[i] > j
        end
    end
end
```

```
Algorithm Trace_Knapsack (w, v, M)
// w is array of weight of n items
// v is array of value of n items
// M is the knapsack capacity

SW ← { }
SP ← { }
i ← n
J ← M
while (j>0) do
   if (V[i, j] == V[[i-1, j] then
    i ← i-1
   else
    V[i, j] ← V[i, j] - vi
    j ← j - w[i]
    SW ← SW + W[i]
    SP ← SP + V[i]
 end
end.
```

Running time complexity using DP can be solved by n (number of items) & M (Capacity of knapsack)

O(nM) is the time complexity

Example

$$V[i,j] = \begin{cases} 0 & \text{if } i=0 \text{ or } j=0 \\ V[i-1,j] & \text{if } j < w_i \\ \max\{V[i-1,j],\ v_i + V[i-1,j-w_i]\} & \text{if } j \geq w \end{cases}$$

j →

	Item detail		0	1	2	3	4	5
i = 0			0	0	0	0	0	0
i = 1	$w_1=2$	$v_1=3$	0	0				
i = 2	$w_2=3$	$v_2=4$	0					
i = 3	$w_3=4$	$v_3=5$	0					
i = 4	$w_4=5$	$v_4=6$	0					

Filling first column $j=1$

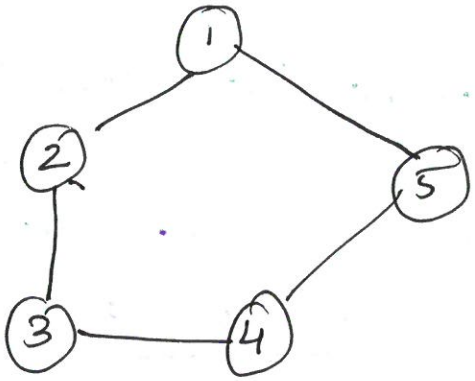
$V[1,1] \Rightarrow i=1, j=1, w_i = w_1 = 2$

As, $j < w_i$, $V[i,j] = V[i-1,j]$

$V[1,1] = V[0,1] = 0$

Minimum Spanning Tree

What is a spanning Tree?



$G = (V, E)$

G - Graph

V - Vertices

E - Edges

Spanning Tree of G?

$G' = (V', E')$

Conditions for spanning tree

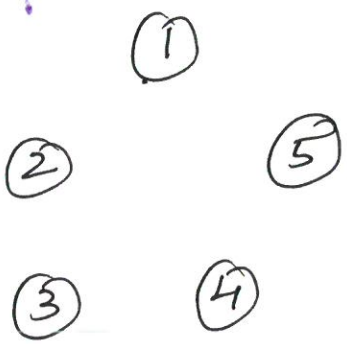
$V' = V$ → Same No: of vertices

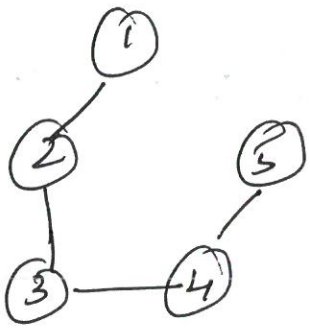
$E' \in E$

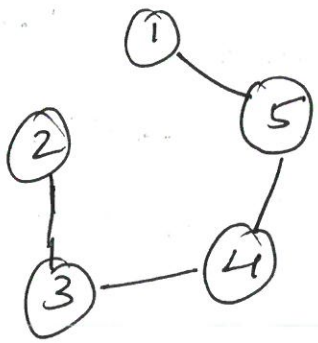
$E' = |V| - 1$ → No: of edges would be no: of vertices - 1

→ A graph can have more than one spanning tree.

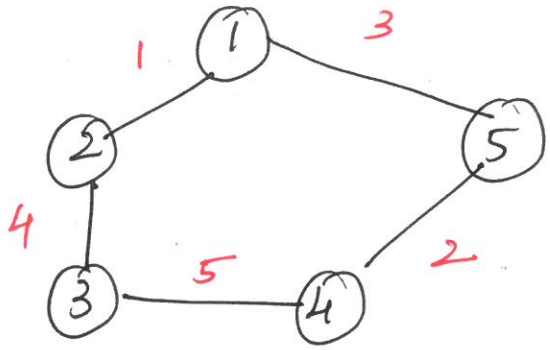
Example





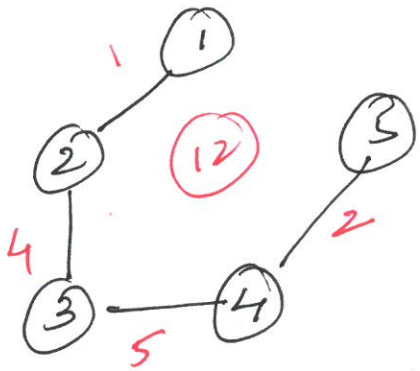


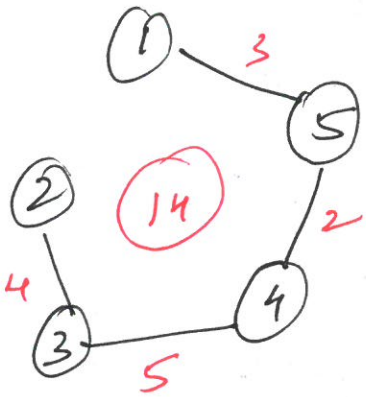
What is Minimum spanning Tree?

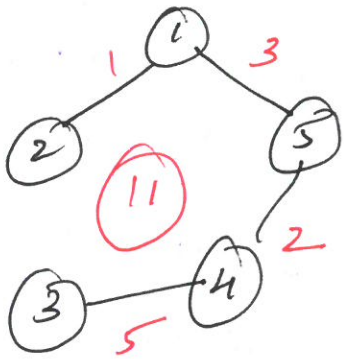


Suppose the graph contains weights, the minimum cost of the total weights is the minimum spanning tree

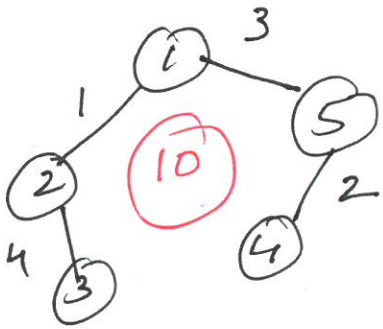
Eg







Consider the edge which is having less cost



→ minimum //.

Properties of Spanning Tree

→ Removing one edge from spanning tree will make it disconnected

→ Adding one edge to the ST will create a loop

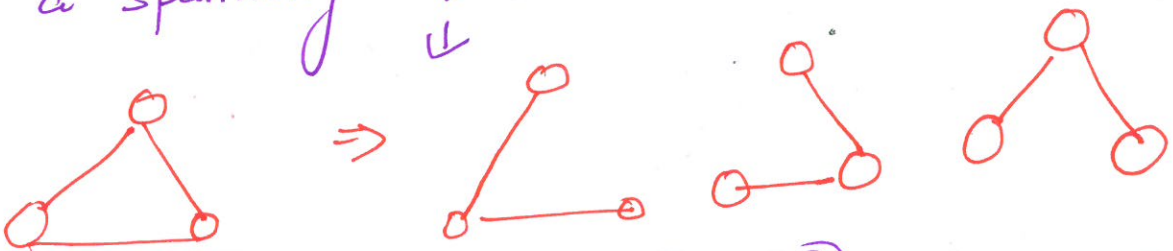
→ If each edge has distinct weight then there will be only one & unique MST

→ A complete undirected graph can have n^{n-2} no: of spanning tree

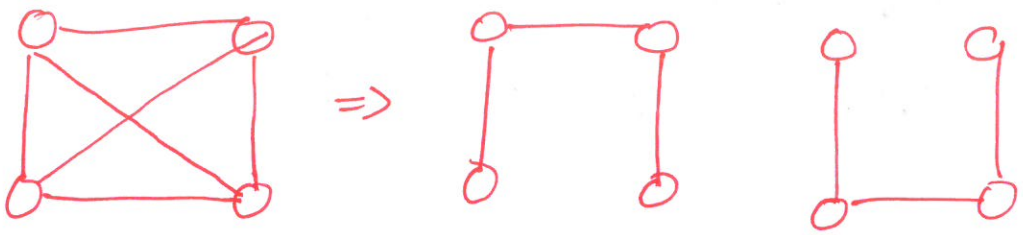
→ Every connected undirected graph has atleast one spanning tree

→ Disconnected graph does not have any Spanning tree

→ From a complete graph by removing max $(e - n + 1)$ edges we can construct a spanning Tree



$3 - 3 + 1 = 1$



$n^{n-2} = 16$

$6 - 4 + 1$

$= 2 + 1 = 3$ //

Prim's Algorithm

Steps

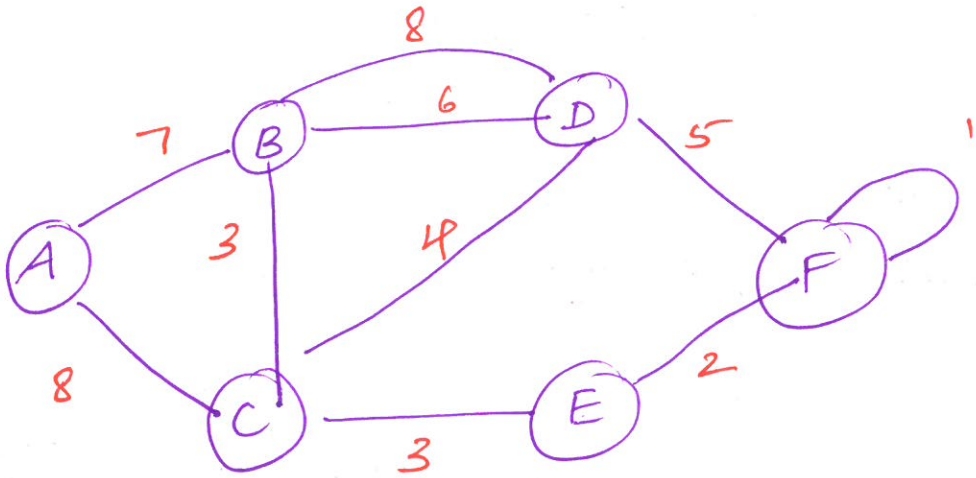
⇒ 1. First check if Graph contains a loop edge or parallel edge

⇒ 2. If it contains loop edge, remove them

⇒ 3. If it contains parallel edge, remove the parallel edge that is having more cost.

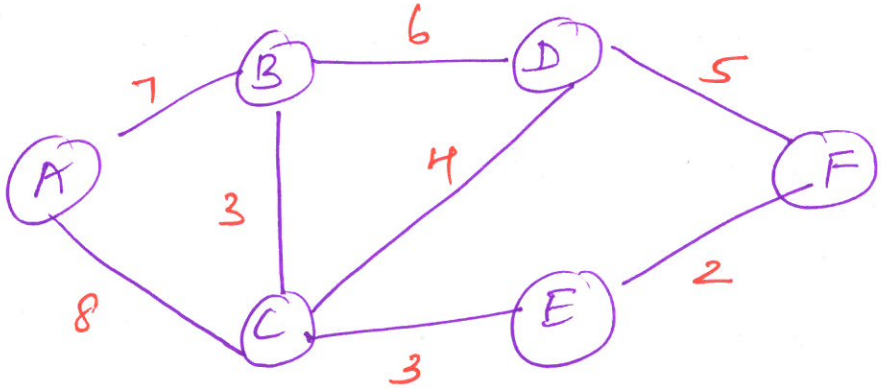
⇒ 4. Now select any node and from that find the edge having minimum const and in all steps we have to check the previous steps edges that are not selected for minimum cost. Follow the steps

till vertices in minimum spanning tree and original graph is same.



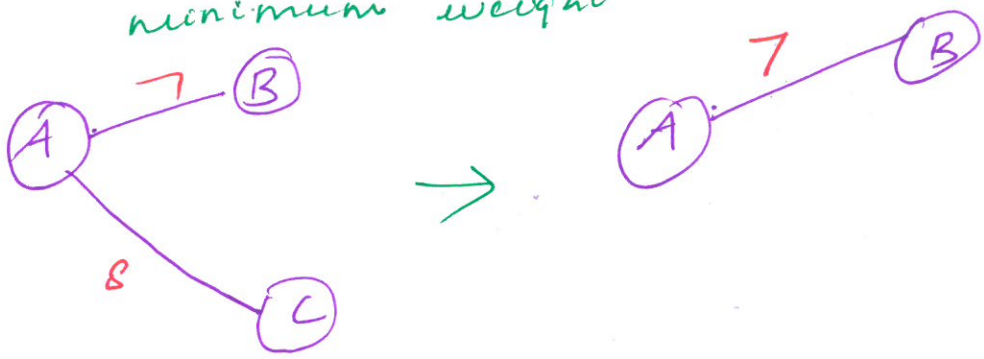
Remove all loops and parallel edges

8 & 6 take minimum edge



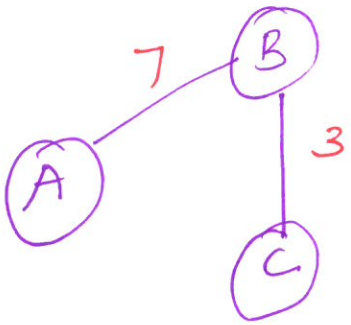
Choose any arbitrary node as the roote node (vertex)

Check out all the outgoing edges from this root node. from that choose the minimum weight



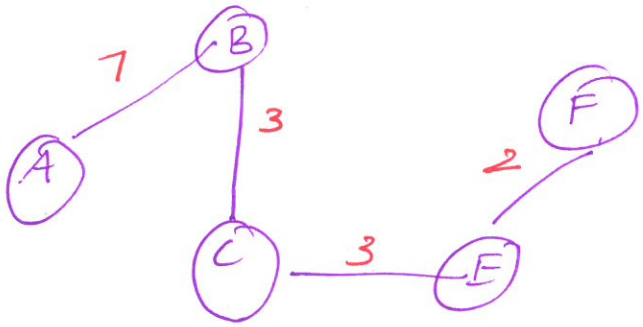
Now check all the outgoing edges from A as well as B

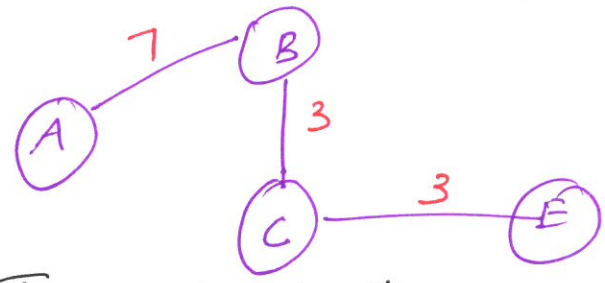
8, 3 & 6 (minimum) we have to choose



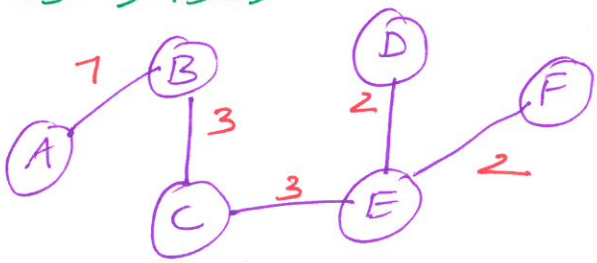
8, 6, 3 & 4

8, 6, 2, 2, 4





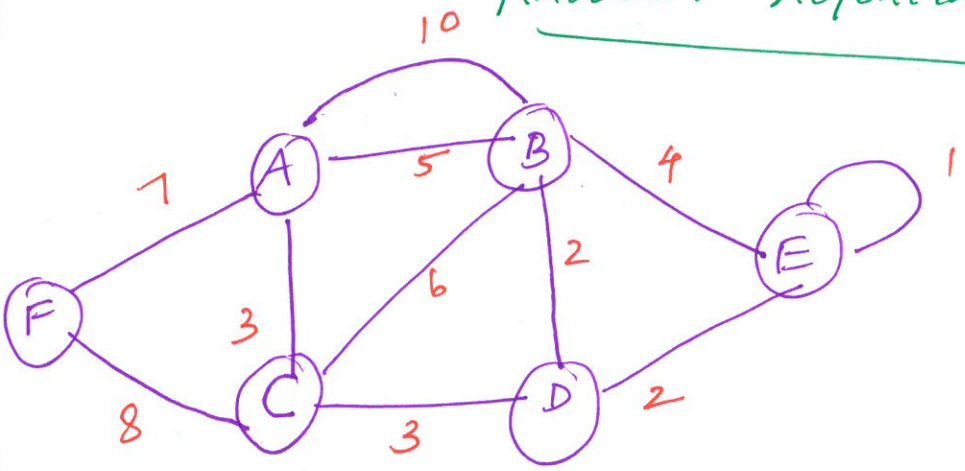
5, 2, 4, 6, 8



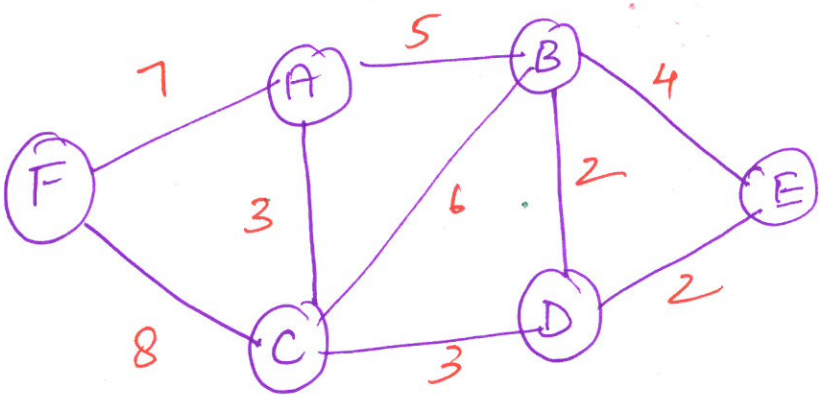
Time complexity

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

Kruskal's Algorithm – Greedy



1) Remove all loops & parallel edges.



2) Arrange all edges according to edge weight in increasing order of weight.

BD = 2 ✓

DE = 2 ✓

AC = 3 ✓

CD = 3 ✓

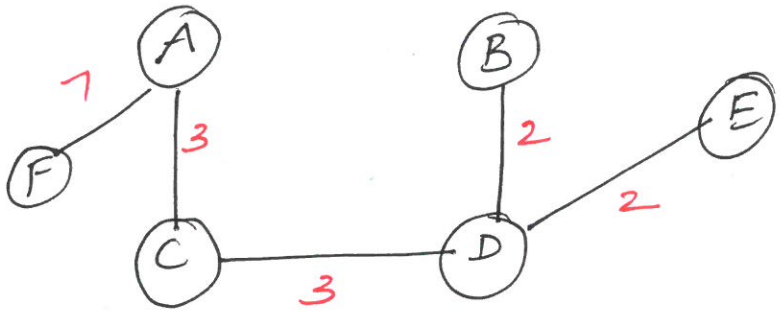
BE = 4 ✗

AB = 5 ✗

BC = 6 ✗

AF = 7 ✓

FC = 8



Property

→ MST does not contain any cycles

→ If n number of vertices in a graph then it should have same number of vertices and n-1 edges.

Time Complexity

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

Greedy Approach

Feasible: It has to satisfy the problem's constraints.

Locally optimal: It has to be the best local choice among all feasible choices available on that step.

Irrevocable: Once made, it cannot be changed on subsequent steps of the algorithm.

Tree Traversal

Inorder: Left Root Right

Preorder: Root Left Right

Postorder: Left Right Root

Inorder:

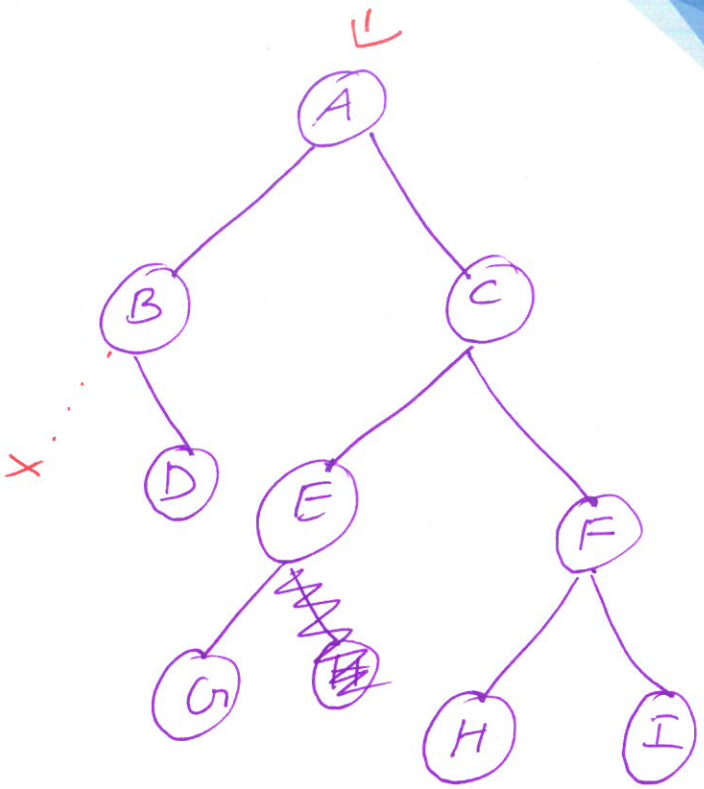
B D A G E C H F I

Preorder:

A B D C E G F H I

Postorder:

D B G E H I F C A



Inorder Traversal

Algorithm

Until all nodes are traversed -

Step 1 - Recursively traverse left subtree

Step 2 - Visit root node

Step 3 - Recursively traverse right subtree

Pre-order Traversal

Algorithm

Until all nodes are traversed

Step 1 - Visit root node

Step 2 - Recursively traverse left subtree

Step 3 - Recursively traverse right subtree

Post-order Traversal

Algorithm

Untill all nodes are traversed-

Step 1 - Recursively traverse left subtree

Step 2 - Recursively traverse right subtree

Step 3 - Visit Root Node

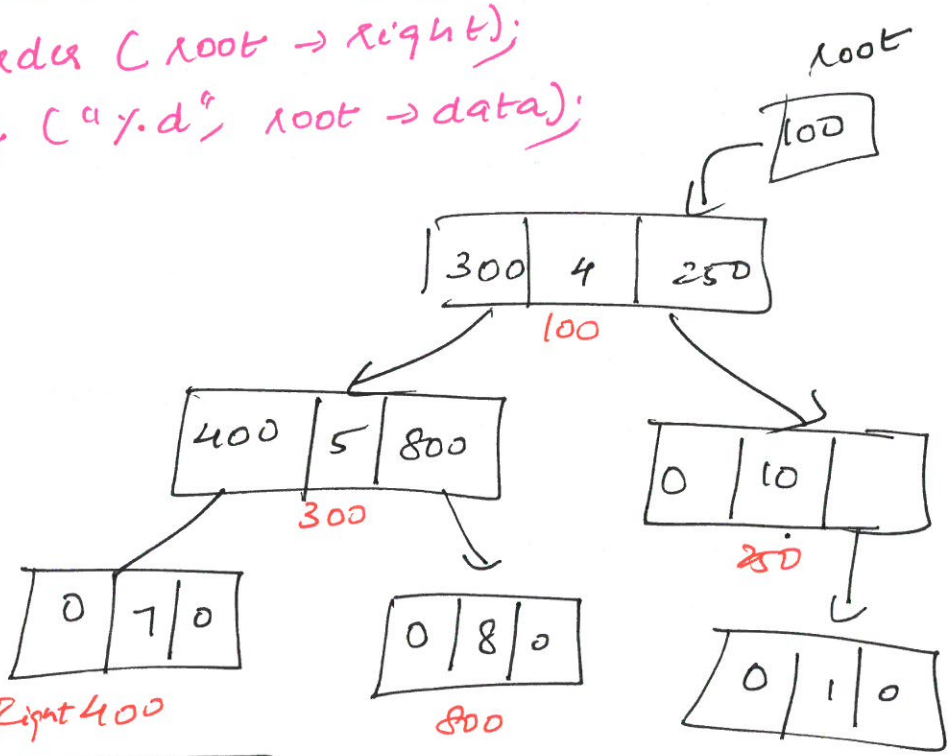
Program

```
void main()
{
  struct node * root;
  printf("Preorder is:");
  preorder(root);
```

```
Inorder (root);
Postorder (root);
void Preorder (struct node * root)
{
    if (root == 0)
    {
        return;
    }
    else
    {
        printf ("%d", root -> data);
        preorder (root -> left);
        preorder (root -> right);
    }
}
void Inorder (struct node * root)
{
    if (root == 0)
    {
        return;
    }
    else
    {
        Inorder (root -> left);
        Printf ("%d", root -> data);
        Inorder (root -> right);
```

```
    }
}

void Postorder (struct node *root)
{
   if (root ==0)
   {
     return;
   }
   else
   {
     postorder (root -> left);
     postorder (root -> right);
     printf ("%d", root -> data);
   }
}
}
```



Inorder = Left Root Right 400

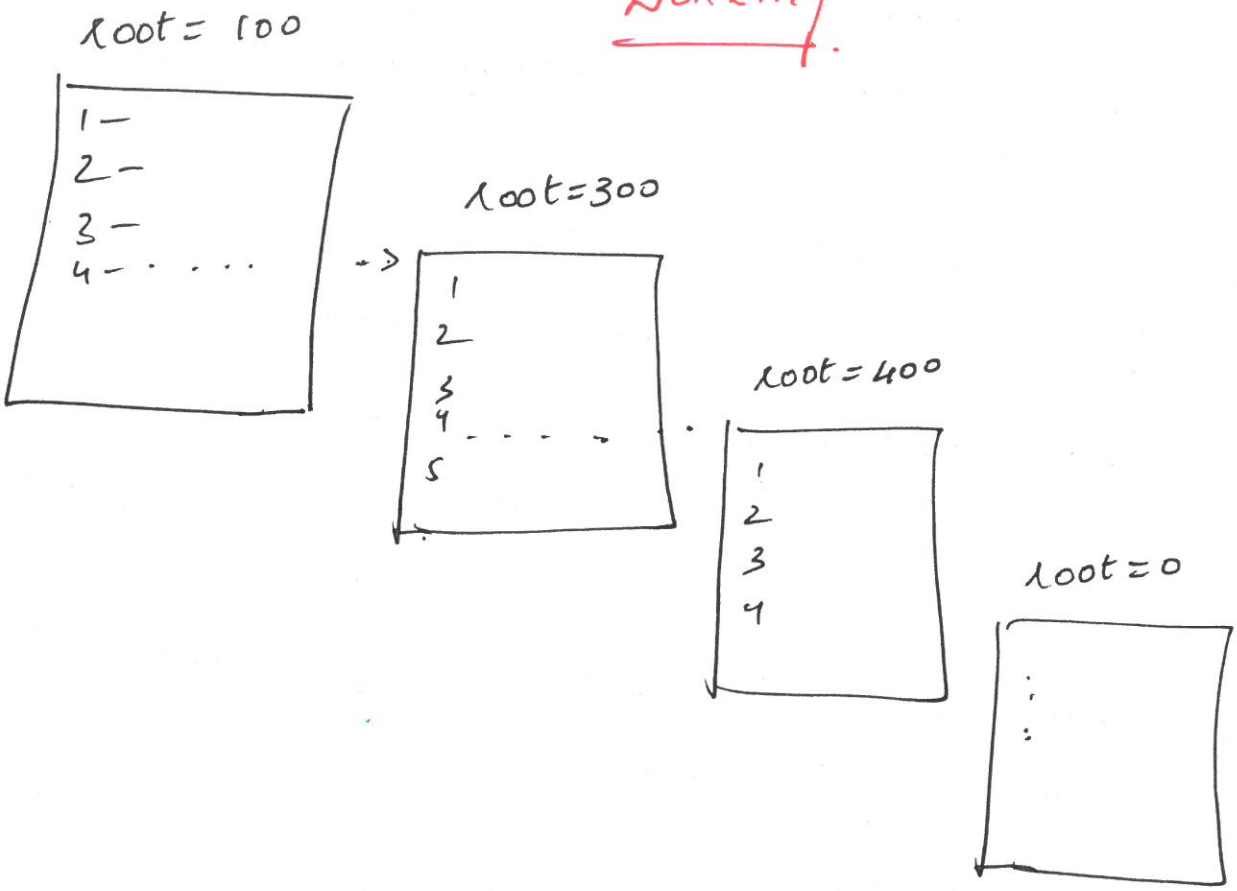
= 7 5 8 4 1 0 1

Preorder = Root Left Right

= 4 5 7 8 10 1

Postorder = Left Right Root = 7 8 5 1 10 4

Looking.



Dynamic Programming - Introduction.

Greedy & Dynamic both are used for solving optimization either maximum & minimum problem.

In Greedy we find out a predefined method for ~~solvin~~ Optimum solution but in dynamic we find out all possibilities for optimum solution.

→ This approach is different and little time consuming compared to greedy method.

→ DP are solved using recursive formulas.

→ Mostly solved using iterations

→ DP follows principle of optimality → It means taking sequence of decisions

→ In Greedy method decision is taken once but in DP every stage we take decisions.

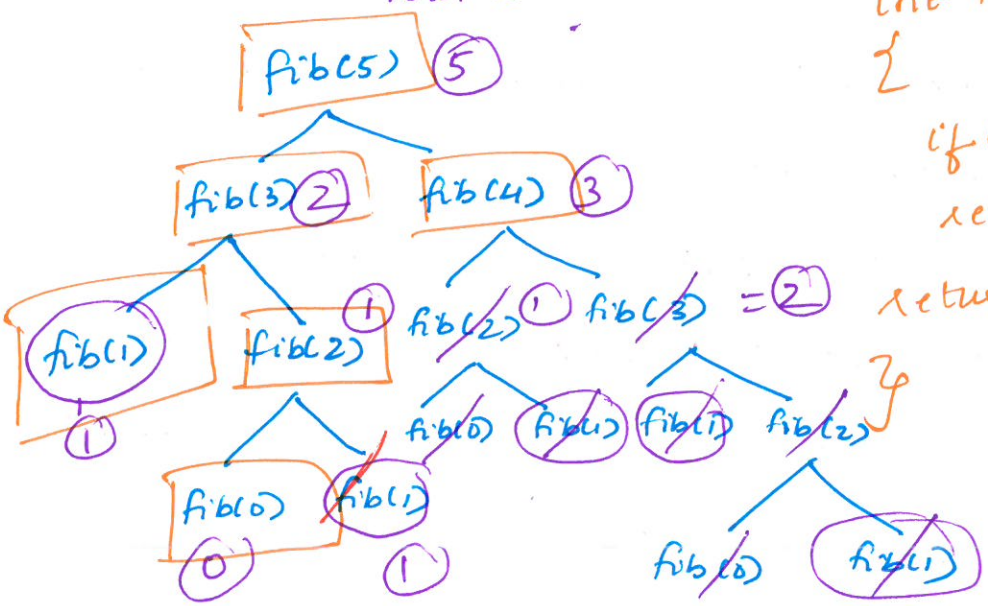
Example

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

0, 1, 1, 2, 3, 5, 8, 13.

0 1 2 3 4 5 6 7 ....

10th term?




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

n-2 cannot be taken

$T(n) = \underbrace{2T(n-1)}_{\text{recursive}} + 1 \rightarrow$ adding

$= O(2^n)$ //.

Why to call same function again & again? How to reduce the calling function?

Take a global array and initially fill it with -1. Observe the tree

Memoization

F

-1	-1	-1	-1	-1	-1
0	1	2	3	4	5

finally

fib(5) dont know

fib(3) dont know

fib(1) = 1 now mark it as 1

F

0	1	1	2	3	5
0	1	2	3	4	5

F

-1	1	-1	-1	-1	-1
0	1	2	3	4	5

fib(n) = n+1 calls

$\rightarrow \Theta(n)$

fib(2) ×

fib(0) ← 1 returns 0

6 calls made

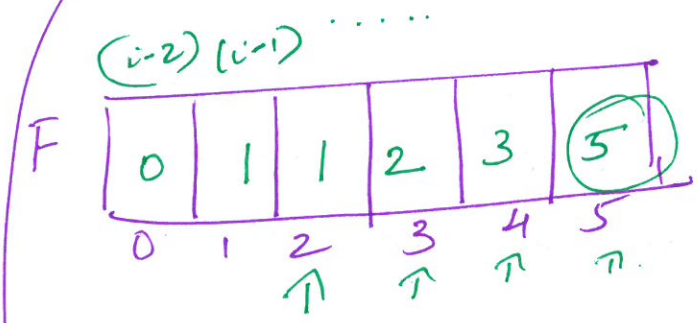
F

0	1	-1	-1	-1	-1
0	1	2	3	4	5

Memoization is Top down approach

fib(1) already called

Tabulation Method



fib(5)

-> Using recursion method

-> Bottom up approach

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

```
int fib (int n)
{
    if (n <= 1)
        return n;
    F[0] = 0; F[1] = 1;
    for (int i = 2; i <= n; i++)
    {
        F[i] = F[i-2] + F[i-1];
    }
    return F[n];
}
```

Matrix Chain Multiplication

↳ What is Matrix Multiplication

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{bmatrix} \qquad B = \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \\ b_{31} & b_{32} \end{bmatrix}$$

2×3 $\qquad$ 3×2

$$C = A \times B = \begin{bmatrix} a_{11}\times b_{11} + a_{12}\times b_{21} + a_{13}\times b_{31} & a_{11}\times b_{12} + a_{12}\times b_{22} + a_{13}\times b_{32} \\ a_{21}\times b_{12} + a_{22}\times b_{21} + a_{23}\times b_{31} & a_{21}\times b_{12} + a_{22}\times b_{22} + a_{23}\times b_{32} \end{bmatrix}$$

3 mul

2×2

How many multiplications are done?

$2 \times 3 \times 2 = 12$ multiplications for obtaining resultant matrix!

2×3 $\qquad$ 3×2

↓

$2 \times 3 \times 2$

How many dimensions are there

2×3 $\qquad$ 3×2

$d_1 \; d_2$ $\qquad$ $d_2 \; d_3$

(3 dimensions)

What is matrix chain multiplication?

$A_1 \times A_2 \times A_3$

2 3 3 4 4 2

d_0 d_1 d_1 d_2 d_2 d_3

Can I multiply altogether? It is not possible. Only two matrix is possible. Then How?

$(A_1 \times A_2) \times A_3$

2 3 3 4 4 2

$2 \times 3 \times 4 = 24$

0

Together?

2×4 4×2 → dimensions

$2 \times 4 \times 2 = 16$ mul

So

$24 + 0 + 2 \times 4 \times 2 = 40$

$A_1 \times (A_2 \times A_3)$

2 3 3 4 4 2

$3 \times 4 \times 2 = 24$

0

Together?

2×3 3×2

$2 \times 3 \times 2 = 12$ mul

So

$0 + 24 + 2 \times 3 \times 2 = 36$

Take min

→ formula using Dynamic Programming

$c[1,3] = ?$

$A_1 \times A_2 \times A_3$

2 3 3 4 4 2

d_0 d_1 d_1 d_2 d_2 d_3

$(A_1 \times A_2) \times A_3$

2 3 3 4 4 2

$c[1,2] = 24$ $c[3,3] = 0$

$c[1,2] + c[3,3] + d_0 \times d_2 \times d_3$

$24 + 0 + 2 \times 4 \times 2 = 40$

$A_1 \times (A_2 \times A_3)$

2 3 3 4 4 2

$c[1,1] = 0$ $c[2,3] = 24$

$c[1,1] + c[2,3] + d_0 \times d_1 \times d_3$

$0 + 24 + 2 \times 3 \times 2 = 36$

$c[i,k] + c[k+1,j] + d_{i-1} \times d_k \times d_j$

$c[i,k] + c[k+1,j] + d_{i-1} \times d_k \times d_j$

$$c[i,j] = \min_{i \le k < j} \{ c[i,k] + c[k+1,j] + d_{i-1} \times d_k \times d_j \}$$

How to use DP formula?

$$c[i,j] = \min_{i \le k < j} \left\{ c[i,k] + c[k+1,j] + d_{i-1} \times d_k \times d_j \right\}$$

$A_1 \times A_2 \times A_3 \times A_4$

$d_0\ d_1 \quad d_1\ d_2 \quad d_2\ d_3 \quad d_3\ d_4$

1. $A_1 (A_2 (A_3 A_4))$
2. $A_1 ((A_2 A_3) A_4)$
3. $(A_1 A_2)(A_3 A_4)$
4. $[A_1 (A_2 A_3)) A_4$
5. $((A_1 A_2) A_3) A_4$

Catalan formula

$$\frac{2nCn}{n+1}$$

$n \rightarrow$ no of matrix -1

$n = 4-1$

$$\frac{2(n-1)C_{(n-1)}}{n} \leftarrow \frac{2(n-1)C_{(n-1)}}{(n-1+1)}$$

Here 5 multiplication paranthesis are required

$$\frac{2 \times 3C_3}{4} = \frac{6C_3}{4} = \frac{6 \times 5 \times 4}{3 \times 2 \times 1 \times 4} = 5$$

If $n=5$ Matrix then 14 multiplications are required

$$c[i,j] = \min_{i \le k < j} \{ c[i,k] + c[k+1,j] + d_{i-1} \times d_k \times d_j \}$$

$A_1 \times A_2 \times A_3 \times A_4$

$d_0\ d_1 \quad d_1\ d_2 \quad d_2\ d_3 \quad d_3\ d_4$

$c[1,4] = \min_{1 \le k < 4}$

$4-1=3$ values

$k=1$: $c[1,1] + c[2,4] + d_0 \times d_1 \times d_4$

$k=2$: $c[1,2] + c[3,4] + d_0 \times d_2 \times d_4$

$k=3$: $c[1,3] + c[4,4] + d_0 \times d_3 \times d_4$

Possible Paranthesis

1) $A_1 (A_2\ A_3\ A_4)$

2) $(A_1\ A_2)\ (A_3\ A_4)$

3) $(A_1\ A_2\ A_3)\ A_4$

$c[2,4] = \min_{2 \le k < 4}$

$4-2=2$ values.

$k=2$: $c[2,2] + c[3,4] + d_1 \times d_2 \times d_4$

$k=3$: $c[2,3] + c[4,4] + d_1 \times d_3 \times d_4$

find the cost from smaller to bigger value.

Initially difference with (1)

(1,2) (2,3) & (3,4)

then difference with (2) so on....

C

i \ j	1	2	3	4
1	0	24		58
2		0	16	36
3			0	40
4				0

K

i \ j	1	2	3	4
1		1	1	3
2			2	3
3				3
4				

min cost is filled in this table

Example Problem

$$c[i,j] = \min_{i \le k < j} \{ c[i,k] + c[k+1,j] + d_{i-1} \times d_k \times d_j \}$$

$A_1 \times A_2 \times A_3 \times A_4$

3 2 2 4 4 2 2 5

d_0 d_1 d_1 d_2 d_2 d_3 d_3 d_4

$c[1,1] = \min_{1 \le k < 1}$ cannot take $k < 1$ so 0

$c[1,2] = \min_{1 \le k < 2}$ (i = 1, j = 2) $k=1 \{ c[1,1] + c[2,2] + d_0 \times d_1 \times d_2 \}$

$3 \times 2 \times 4$

$0 + 0 + 3 \times 2 \times 4 = 24$

$$C[2,3] = \min_{2 \le k < 3} \{ C[2,2] + C[3,3] + d_1 \times d_2 \times d_3 \}$$

$k=2$: $C[2,2] + C[3,3] + d_1 \times d_2 \times d_3$

$0 + 0 + 2 \times 4 \times 2 = 16$

$$C[3,4] = \min_{3 \le k < 4} \{ C[3,3] + C[4,4] + d_2 \times d_3 \times d_4 \}$$

$k=3$: $C[3,3] + C[4,4] + d_2 \times d_3 \times d_4$

$0 + 0 + 4 \times 2 \times 5 = 40$

$$C[1,3] = \min_{1 \le k < 3}$$

$k=1$: $C[1,1] + C[2,3] + d_0 \times d_1 \times d_3 = 28$

$0 + 16 + 3 \times 2 \times 2$

$k=2$: $C[1,2] + C[3,3] + d_0 \times d_2 \times d_3 = 48$

$24 + 0 + 3 \times 4 \times 2$

$$C[2,4] = \min_{2 \le k < 4}$$

$k=2$: $C[2,2] + C[3,4] + d_1 \times d_2 \times d_4 = 80$

$0 + 4 + 2 \times 4 \times 5$

$k=3$: $C[2,3] + C[4,4] + d_1 \times d_3 \times d_4 = 36$

$16 + 0 + 2 \times 2 \times 5$

$$C[1,4] = \min_{1 \le k < 4}$$

$k=1$: $C[1,1] + C[2,4] + d_0 \times d_1 \times d_4$

$0 + 36 + 3 \times 2 \times 5 = 86$

$k=2$: $C[1,2] + C[3,4] + d_0 \times d_2 \times d_4$

$24 + 40 + 3 \times 4 \times 5 = 124$

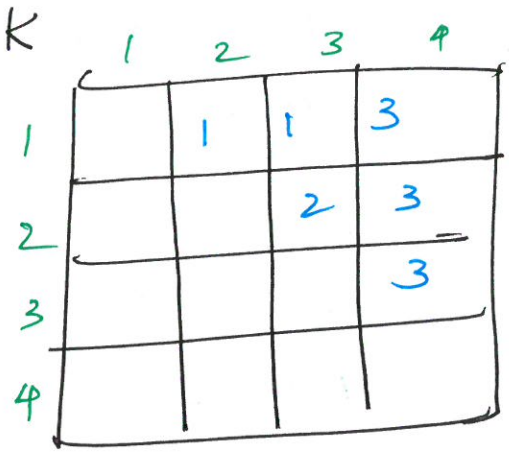
$k=3$: $C[1,3] + C[4,4] + d_0 \times d_3 \times d_4$

$28 + 0 + 3 \times 2 \times 5 = 58$

$A_1 \times A_2 \times A_3 \times A_4$

3 2 2 4 4 2 2 5

d_0 d_1 d_1 d_2 d_2 d_3 d_3 d_4



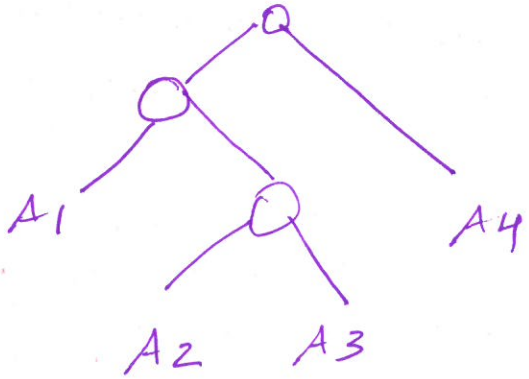
A_1 A_2 A_3 A_4

check $1 > 4$

$(A_1\ A_2\ A_3)\ (A_4)$ (3, 4)

check $1 > 3$

$((A_1)(A_2\ A_3))\ (A_4)$



$1+2+3+4 = \frac{4(5)}{2}$

$\frac{n(n+1)}{2} = n^2$ //

$n^2 \times n \Rightarrow$

$O(n^3)$ //

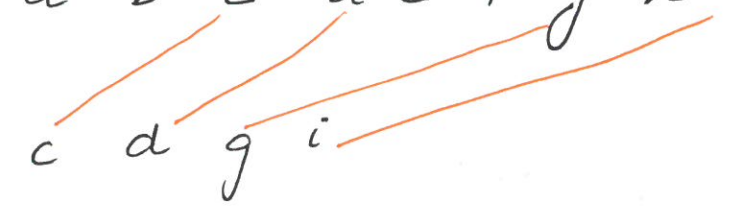
Longest Common Subsequence (LCS)

* What is LCS?
* LCS using Recursion
* LCS using Memoization
* LCS using Dynamic Programming.

Two strings

String 1 : a b c d e f g h i j

String 2 : c d g i



cdgi -> Longest sequence.

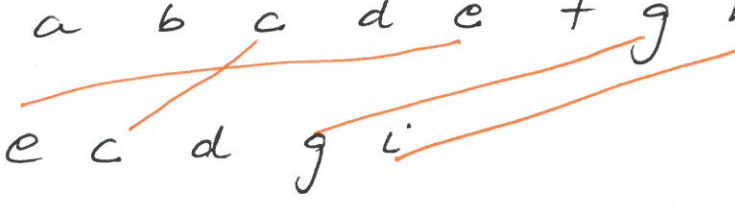
dgi

gi

Example 2

String 1 : a b c d e f g h i j

String 2 : e c d g i



* Should not intersect with each other
* c should follow e and not backside of c

e g i

c d g i

Example 3.

String 1 : a b d a c e

String 2 : b a b c e

1) b a c e

String 1 : a b d a c e

String 2 : b a b c e

2) a b c e

Recursion Method.

A

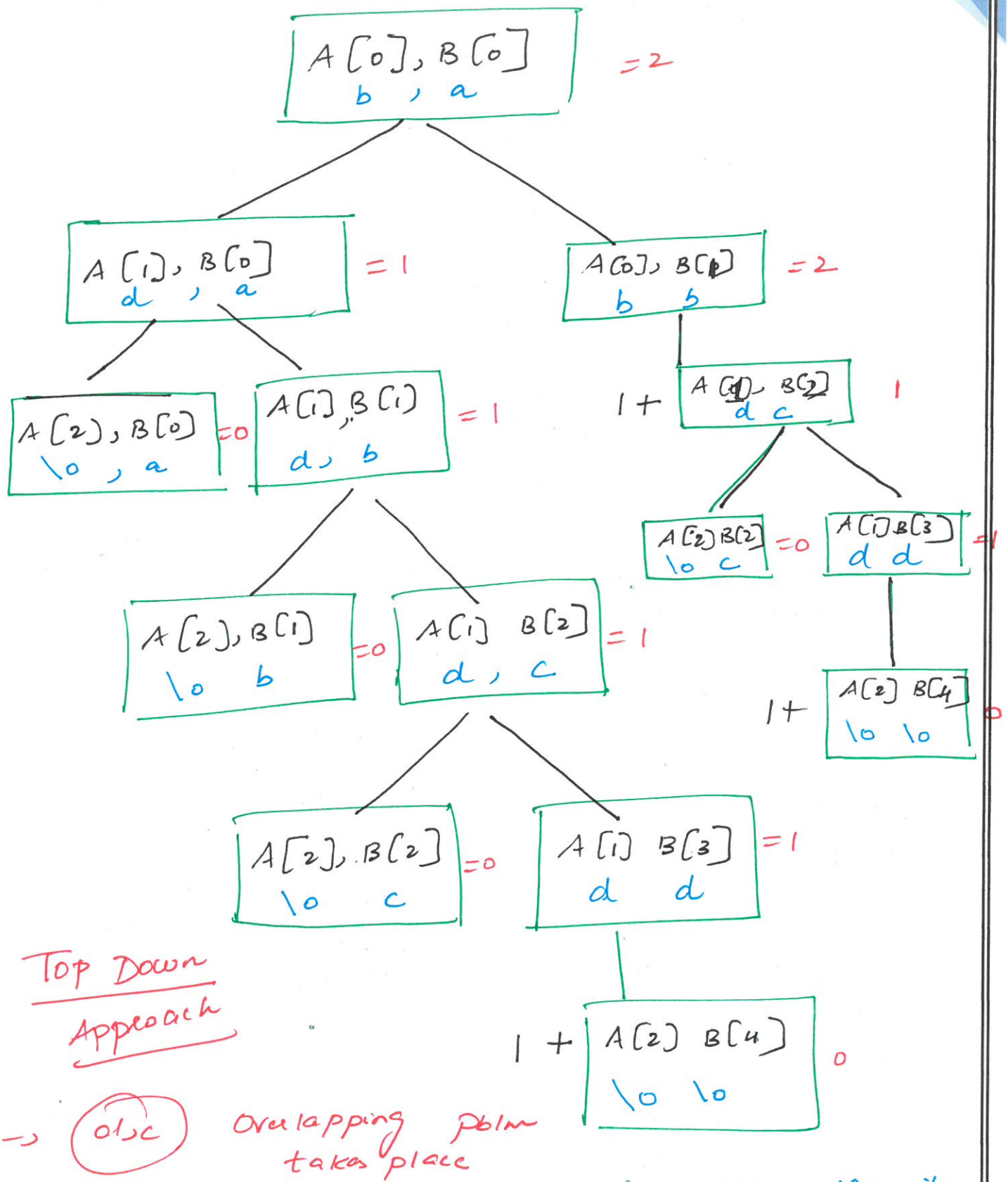
b	d	\0
0	1	2

B

a	b	c	d	\0
0	1	2	3	4

$O(2^n)$ //

```
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));
}
```



Top Down Approach

→ (0,c) Overlapping Pblm takes place

If we store the recursive part somewhere, then it is called Memorization.

Memoization

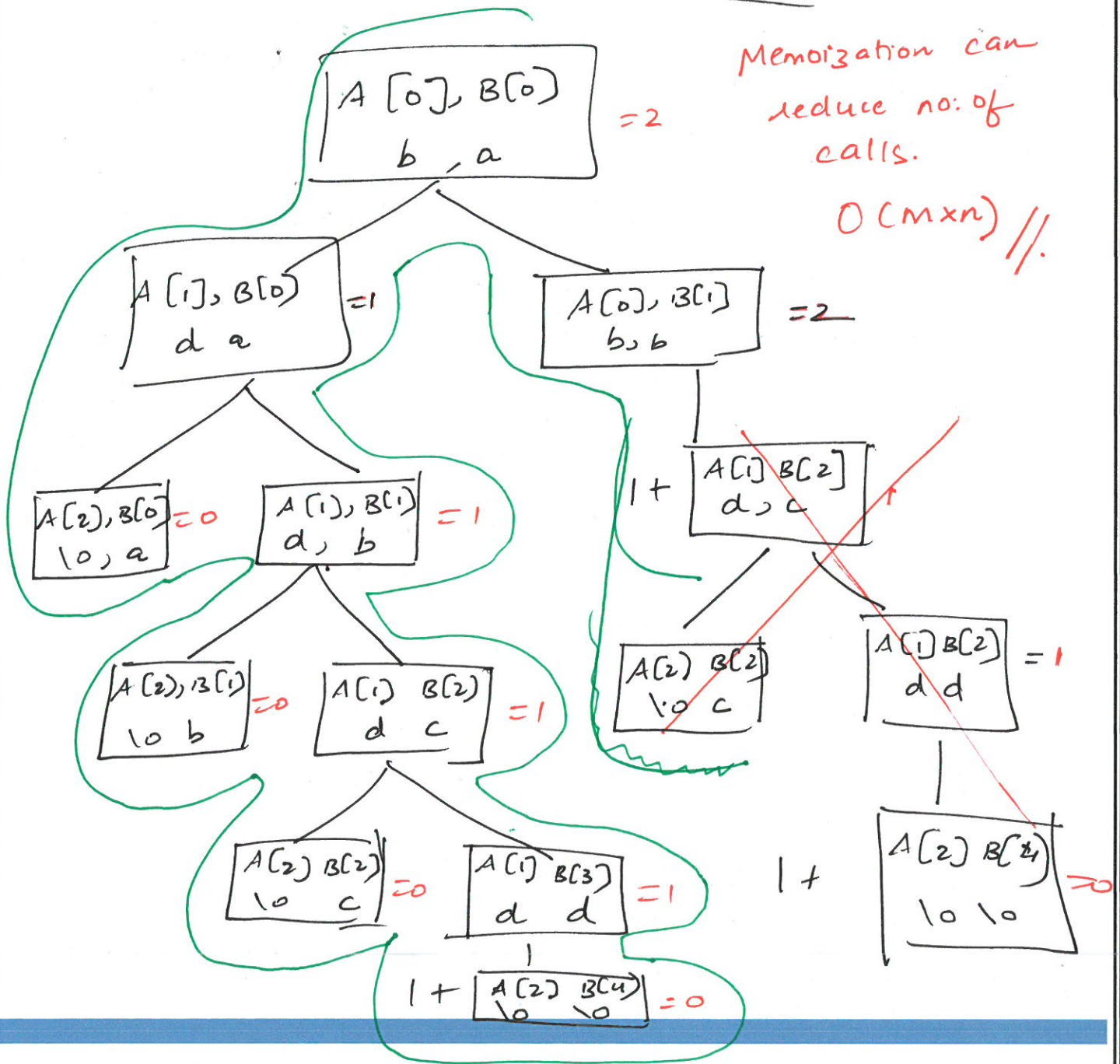
A = a b c d \0

B = b d \0

A = m

B = n

B \ A	0	1	2	3	4
b 0	2	2	·		
d 1	1	1	1	1	
\0 2	0	0	0		0



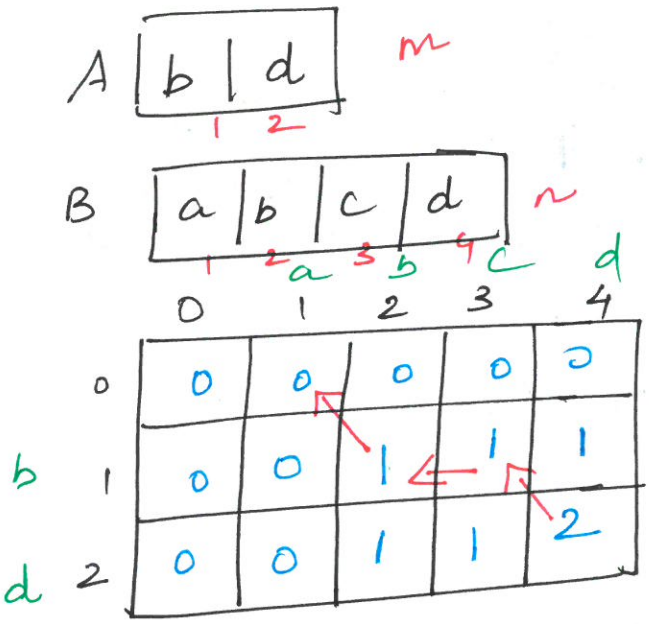
Memoization can reduce no: of calls.

O(m x n) //.

LCS Using Dynamic Programming

→ Bottom up approach

→ Table cs filled from top to bottom.



if (A[i] = B[j]) — prev diag element

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

else

LCS[i,j] = max

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

Prev row of prev column element

→ Initially start with 0 till index

(1,1) → check b & a is not same. take maximum of right diagonal

(1,2) → check b & b matching so tracing

O (m x n)

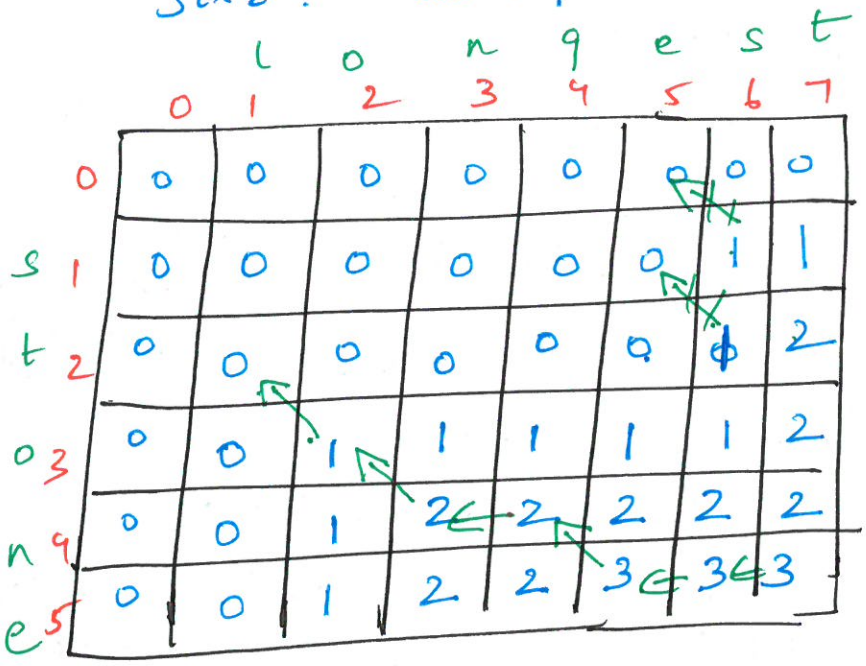
b d

Sequence.

Example 2

Str1 = s t o n e

Str2 : l o n g e s t



(m x n)

O(mn)

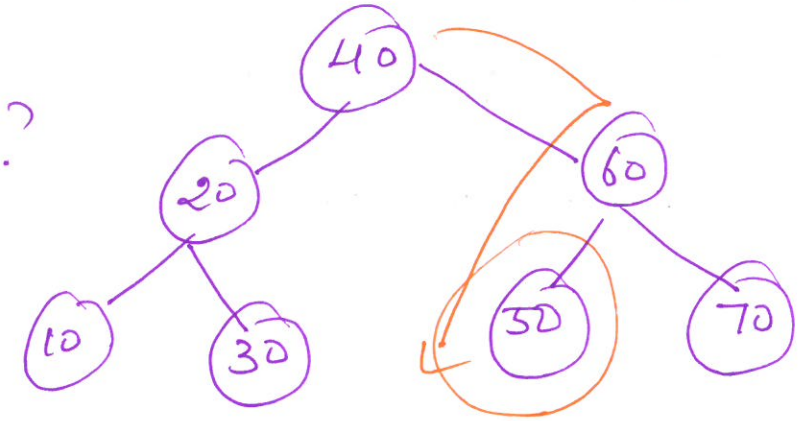
Lone 3.

Optimal Binary Search Tree

What is Binary Search Tree?

Keys -> 10, 20, 30, 40, 50, 60, 70

Key = 50?



log n

L.H.S smaller than Root.

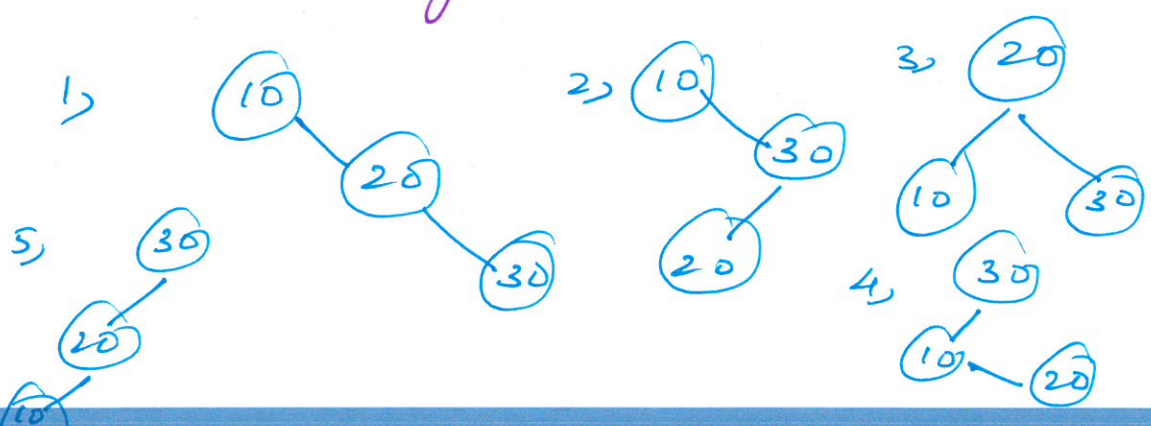
R.H.S Larger than Root

Key = 45? Key not found

Note: Search may be successful or may not be successful.

Keys -> 10, 20, 30

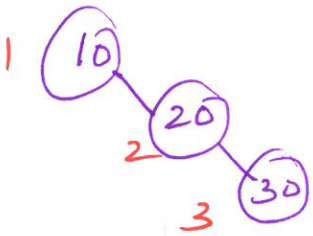
How many BST are possible?



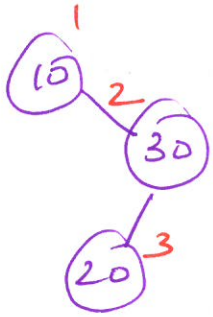
For 3 keys 5 BST are possible

So n keys $\frac{2nC_n}{n+1}$ $\frac{2\times3C_3}{3+1} = 5$ //.

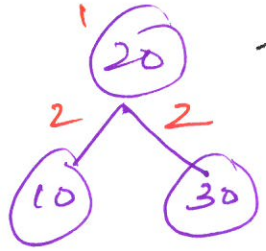
Cost for a searching is based on number of comparisons.

1) 

$\rightarrow \frac{1+2+3}{3} = \frac{6}{3} = 2$

2) 

$\rightarrow \frac{1+2+3}{3} = \frac{6}{3} = 2$

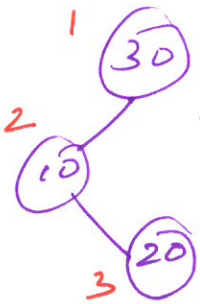
3) 

$\rightarrow \frac{1+2+2}{3} = \frac{5}{3}$

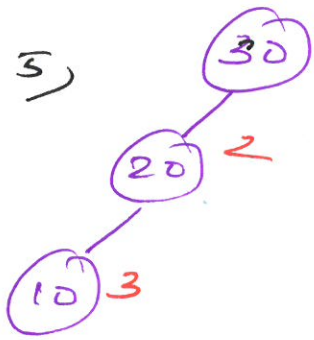
$\Rightarrow$ Balanced BST

Why it is less

$\rightarrow$ Height of tree is less

4) 

$\rightarrow \frac{1+2+3}{3} = \frac{6}{3} = 2$

5) 

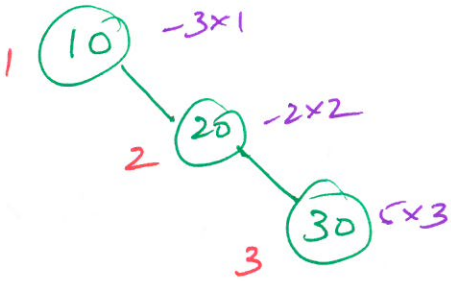
$\frac{1+2+3}{3} = \frac{6}{3} = 2$

What is Optimal Binary search Tree?

Keys → 10 , 20 , 30

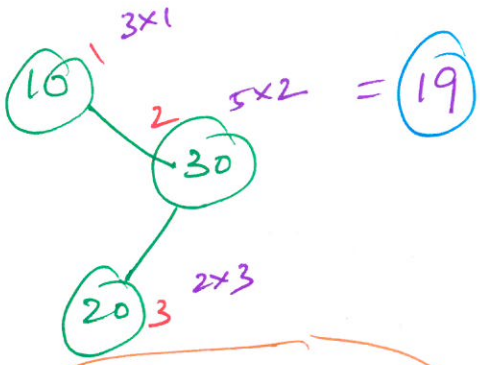
freq → 3 2 5

1)



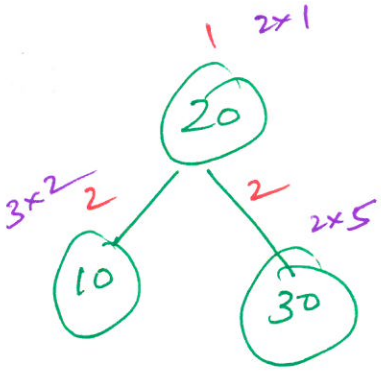
= 18

2)



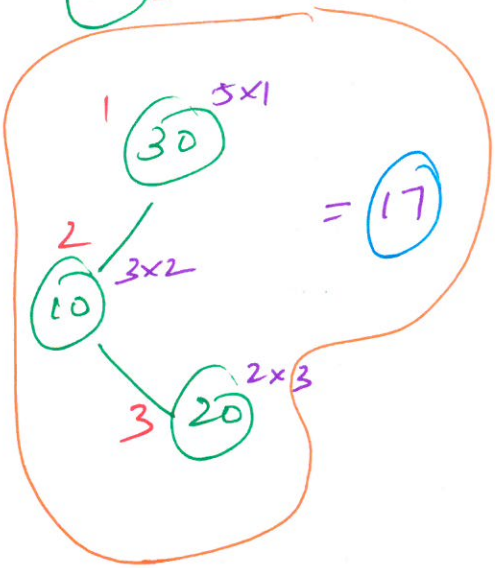
= 19

3)



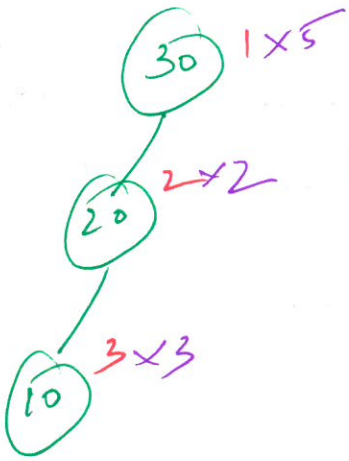
= 18

4)



= 17

5)



= 18

Based on freq tree No 4 is optimum

Optimal Binary Search Tree using DP

	1	2	3	4
Keys →	10	20	30	40
freq →	4	2	6	3

Case 1:

$l = j - i = 0$

$0 - 0 = 0$

$1 - 1 = 0$

$2 - 2 = 0$

$3 - 3 = 0$

$4 - 4 = 0$

$c[0,0]$,

$c[1,1]$, $= 0$.

$c[2,2]$

$c[3,3]$

$c[4,4]$

i \ j	0	1	2	3	4
0	0	4	8 (1)	20 (3)	26 (3)
1		0	2	10 (3)	16 (3)
2			0	6	12 (3)
3				0	3
4					0

Case 2:

$l = j - i = 1$

$1 - 0 = 1$ $(0,1)$

$2 - 1 = 1$ $(1,2)$

$3 - 2 = 1$ $(2,3)$

$4 - 3 = 1$ $(3,4)$

$\Rightarrow$

$c[0,1]=4$	$c[1,2]^2$	$c[2,3]^6$	$c[2,4]^3$
10	20	30	40
4	2	6	3

Case 2:

$l = j - i = 2$

$2-0 = (0,2)$

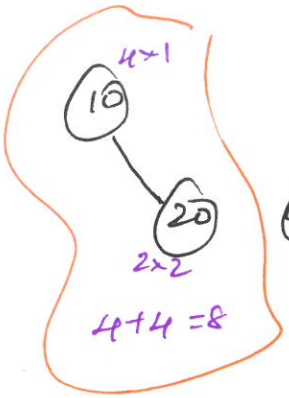
$3-1 = (1,3)$

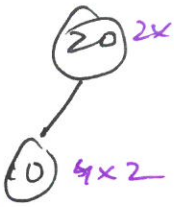
$4-2 = (2,4)$

$c[0,2]$

10 20

4 2





$c[0,1)$

$2+8=10$

$$w(0,4) = \sum_{i=1}^{4} f(i)$$

$c[0,0] + c[1,2] + w[0,2]$

$0 + 2 + 4+2$

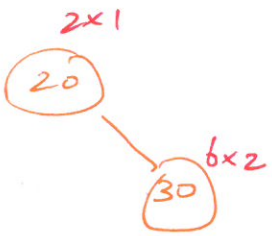
$= 8$ //

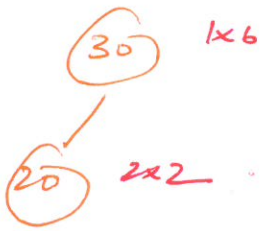
$c[0,1] = c[0,1] + c[2,2] + w[0,2]$

$4 + 0 + 6 = 10$ //.

$c[1,3]$

20	30
2	6



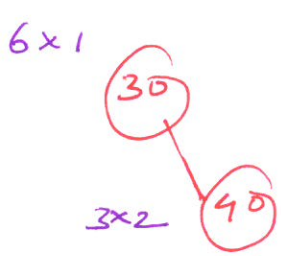


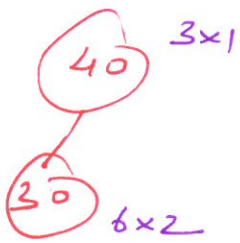
$2+12=14$

$4+6=10$

$c[2,4]$

3	4
30	40
6	3





$6+6=12$

$12+3=15$

Case 4:

$l = j - i = 3$

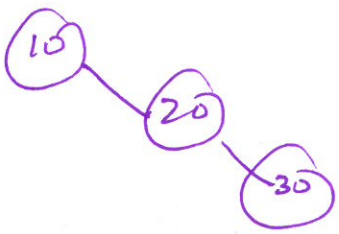
$3 - 0 = (0,3)$

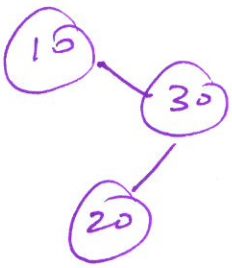
$4 - 1 = (1,4)$

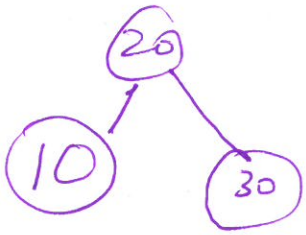
$C[0,3]$ $\quad w[0,3] = 12$

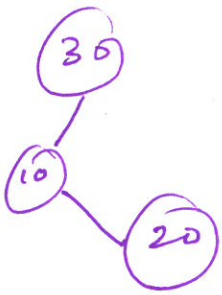
1	2	3
10	20	30
4	2	6

$w(0,4) = \sum_{i=1}^{4} f(i)$









$C[0,3] = \min \{ C[0,0] + C[1,3] + 12,\ C[0,1] + C[2,3] + 12,$

$0 + 10 + 12 \qquad 4 + 6 + 12$

$22 \qquad 22$

$C[0,2] + C[3,3] + 12 \}$

$8 + 0 + 12$

20

$c[1,4] = w[1,4] = 11$

$w(0,4) = \sum_{i=1}^{4} f(i)$

2	3	4
20	30	40
2	6	3

$$c[1,4] = \min \begin{cases} c[1,1] + c[2,4] & 0+12 \\ c[1,2] + c[3,4] & 2+3 \\ c[1,3] + c[4,4] & 10+0 \end{cases} + 11 = 16 //$$

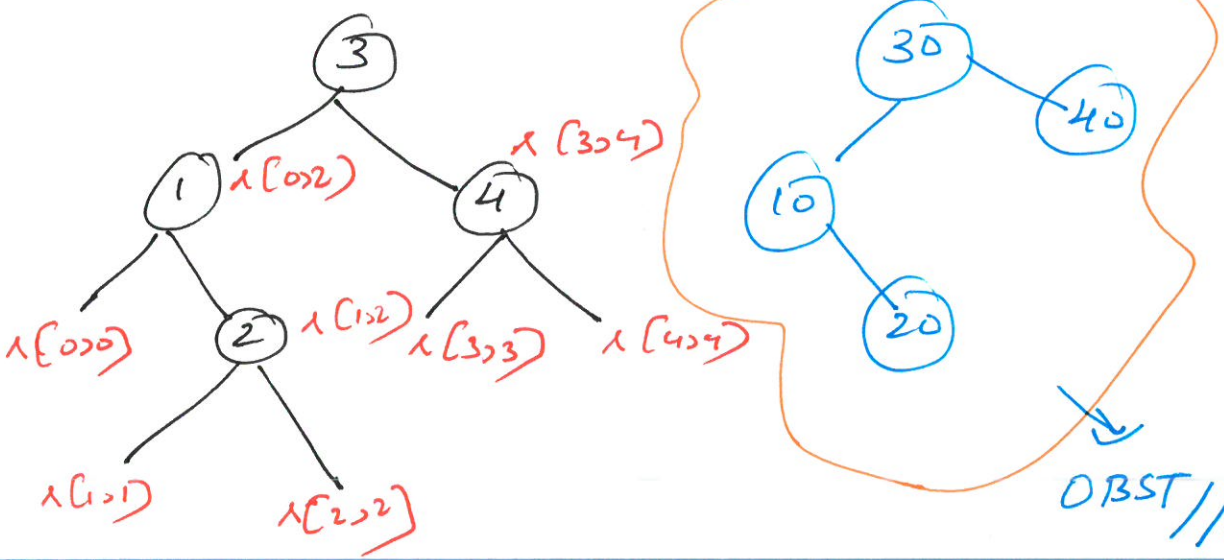
$c[0,4]$

$l = j - i = 4$

$4 - 0 = (0,4)$

$$c[0,4] = \min \begin{cases} c[0,0] + c[1,4] & 0+16 \\ c[0,1] + c[2,4] & 4+12 \\ c[0,2] + c[3,4] & 8+3 \\ c[0,3] + c[4,4] & 20+0 \end{cases} + 15 = 11+15 = 26 //$$

$r[0,4] = 3$



formula

$$c[i,j] = \min_{i \le k \le j} \{ c[i,k-1] + c[k,j] \} + w(i,j)$$

$O(n^3)$ // Time