* Dynamic Programming:

Exploding problem → Polynomial Time (n^c)

Catlan No = $\frac{1}{n}\binom{2n-2}{n-1}$

Matrix Chain Multiplication Problem.

Memory element is used to keep records

It is parenthization problem

Given: $A_1, A_2 \ldots A_n$ (Input Matrices)

Parenthsize these matrices such that total no of multiplications will be minimum for all possible parenthesization.

$A_i \longrightarrow P_{i-1} \times P_i$

A_1 A_2 . . A_n

$P_0 P_1$ $P_1 P_2$ $P_{n-1} P_n$

Function M(i, j)

```
if i = j then return M = 0;
else min = +∞ (very high value)
    for k = i to j - 1 do
        mult = M(i,k) + M(k+1,j) + P[i-1] * P[k] * P[j]
        if mult < Min then
            min = mult ;
        end if
    end for
end if
```

This algorithm is able to find : minimum no of multiplications

Complexity:

$$T(n) = \sum_{k=1}^{n-1} (T(k) + T(n-k)) + O(n)$$

$T(1) = \text{const} = 1.$

$$\therefore \quad T(n) = 2 \sum_{k=1}^{n-1} T(k) + O(n)$$

$T(n) \geq U(n)$

let $U(n) = 2(U(n-1)) + 1.$

$U(n) \approx 2^n - 1$

$U(n) \approx 2^n$

$\therefore \quad T(n) \geq 2^n$

Thus complexity of recursive approach is exponential

```
Procedure Matchain (P)
    for i = 1 to n do M[i, i] = 0; end for;
    for l = 2 to n do
        for i = 1 to n - l + 1 do
            j = i + l - 1 ; M[i, j] = + ∞;
            for k = i to j - 1 do
                mult = M[i, k] + M[k + 1, j] +
                P[i - 1] * P[k] * P[j];
                if mult < M[i, j] then
                    M[i, j] = mult ; S[i, j] = k;
                end if
            end for.
        end for
    end for.
```

3 Nested Loops $\rightarrow O(n^3)$

Trackpoint

M & S

$^6C_3 = \frac{1}{4}\left(\frac{6!}{3!3!}\right)$

$\frac{6 \times 5 \times 4}{3 \times 2}$

$P = |4|2|5|1|3|$

$A_1 \quad A_2 \quad A_3 \quad A_4$

$P_0 \times P_1 \quad P_1 \times P_2 \quad P_2 \times P_3 \quad P_3 \times P_4$

1 $((A_1 A_2)(A_3 A_4)) \rightarrow$ $4 \times 2 \times 5$ $5 \times 1 \times 3$ $4 \times 5 \times 3$

$4 \times 2 \times 5 = 40$ $5 \times 1 \times 3 = 15$ $= 55$ 115

2 $(A_1((A_2 A_3) A_4)) \rightarrow$ $2 \times 5 \times 1$ $2 \times 1 \times 3 = 6$ $= 16$ 40

3 $(((A_1 A_2) A_3) A_4) \rightarrow 72$

4 $(A_1 (A_2 (A_3 A_4))) \rightarrow 69$

5 $((A_1 (A_2 A_3)) A_4) \rightarrow 30$ $4 \times 2 \times 3$

2×3

$15 + 30 + 24$

$2 \times 5 \quad 5 \times 1, \quad 4 \times 1 \quad 1 \times 3$

$2 \times 5 \times 1 \quad 4 \times 1 \times 3 = 12$

M:

	1	2	3	4	
1	0	∞↓ 40	18	(40)	→ 30
2		0	∞↓ 10	45	→ 16
3			0	∞↓ 15	
4				0	

S:

	1	2	3	4
1		1	1	3
2			2	3
3				3
4				

← Track Point.

$[A_1 (A_2 A_3)] A_4$

```
print_Parenth(S,i,j)
    if i=j print "A_i"
    else
        print "("
        print-parenth(S, i, S[i,j])
        print parenth [S, S[i,j]+1, j)
        print ")"
    end if
```

Space Complexity $O(n^2)$

$A_1 \quad A_2 \quad A_3 \quad A_4$

$i = 1 \qquad j = 4.$

$(\underbrace{\quad}_{[1,3]} \quad \underbrace{A_4)}_{[4,4]})$

$((\underbrace{A_1 \quad}_{[1,1]} \underbrace{\quad)}_{[2,3]} \; A_4)$

$((A_1 (\underbrace{A_2}_{[2,2]} \; \underbrace{A_3) A_4}_{[3\,3}),$

* Rod Cutting Problem (without Dynamic Approach : $O(2^n)$, with Dynamic Approach : $O(n^2)$)

Rod of length n and the table of prices P_i for $i = 1, 2, 3 \ldots n$

Where P_i is the price of a rod of length 'i'.

length 'i'	1	2	3	4	5	6	7	8	9	10
Price 'P_i'	1	5	8	9	10	17	17	20	24	30

The total possibilities of cuts are 2^{n-1}. be

A rod of length n can be cut in 2^{n-1} different ways Since we can choose cutting or not cutting at all distances ; $[1 \le i \le n-1]$ from left end

Dynamic programming is Comming under bottom up approach

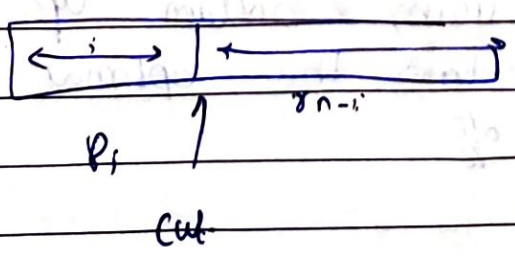
In case of recursive approach.

We can calculate the maximum values r_n ~~Variance r_n~~ Input terms of optimal values for Shorted rods

$r_n = \max (P_n, r_1 + r_{n-1}, r_2 + r_{n-2}, \ldots r_{n-1} + r_1)$

Set $r_0 = 0$ and $r_n = \max (P_i + r_{n-i})$

$1 \le i \le n$.

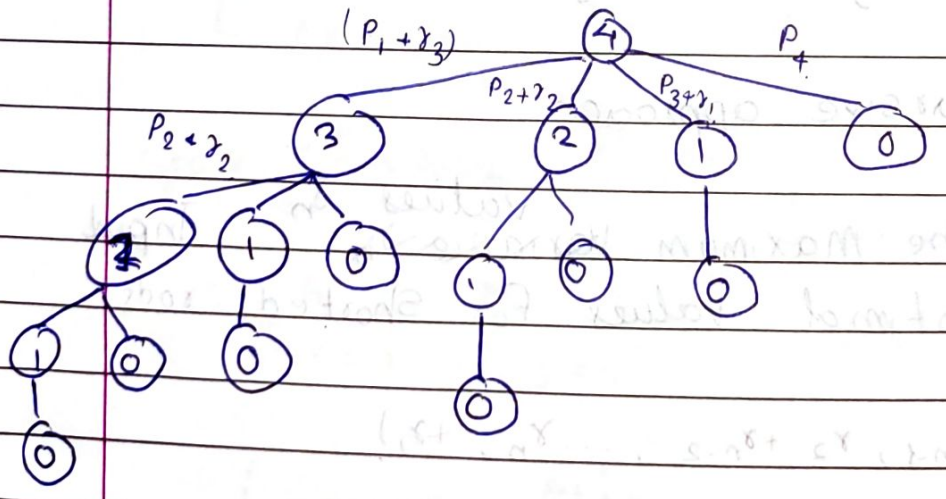


```
Cut_Rod(P, n)
if n=0 then
        return 0;
end
q = -∞
For i = 1 to n do
    q = max(q, P[i] + (ut_Rod, (P, n-i)));
end
return q;
```

T(n) = Total calls made to cut rod fn with rod length 'n'

$$T(n) = \begin{cases} 1 + \sum_{0 \le i \le n-1} T(i) & \text{if } n = 0 \\ 1 & \end{cases}$$

Complexity $O(2^n)$



* Dynamic Programming using bottom up approach
//Array S[0, 1.... n] stores the optimal size of the first piece to cut off.

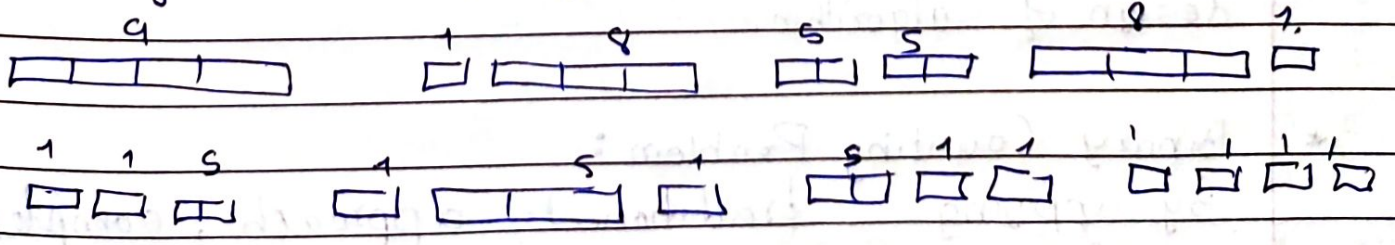
```
r[0] = 0;
for j = 1 to n do
    q = -∞; //eg. 99999
    for i = 1 to j do
        // Solve the problem of size j
        if q < p[i] + r[j-i] then
            q = p[i] + r[j-i];
            s[j] = i; // stores the size of 1st piece
        end
    end.
    r[j] = q;
end
```

```
While n > 0 do
    // print sizes of pieces
    print s[n];
    n = n - s[n];
end
```

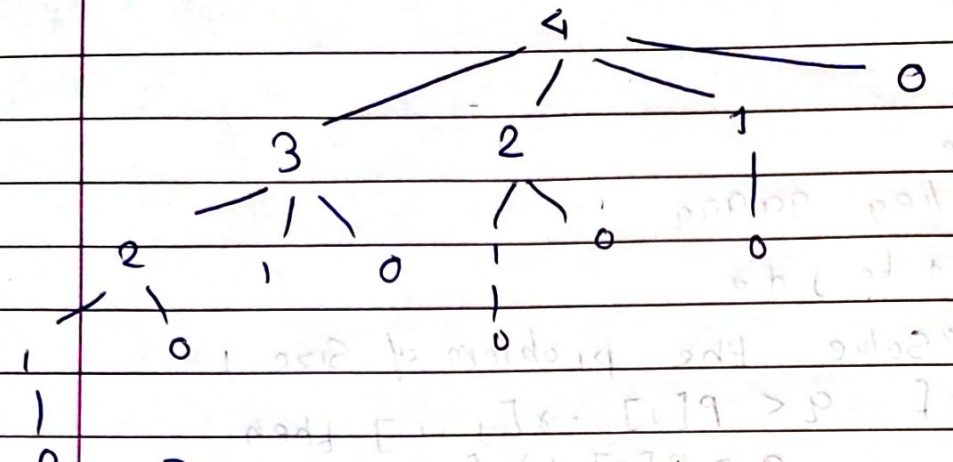
* Rod Cutting: radford University

length i	1	2	3	4	5	6	7	8	9	10
Price p_i	1	5	8	9	10	17	17	20	24	30

(Using bottom up approach)



The 8 possible ways of cutting up a rod of length 4.



The recursion tree showing recursive calls resulting from a call CUT-ROD(p,n) for $n=4$.
Each node label gives the size n of the corresponding subproblem, so that an edge from a parent with label s to a child with label t corresponds to cutting off an initial piece

length i	0	1	2	3	4	5	6	7	8	9	10
Price P:	0	1	5	8	9	10	17	17	20	24	30
r[i]	0	1	5	8	10	13	17	18	22	25	30
s[i]	0	1	2	3	2	2	6	1	2	3	10

* Amortized Analysis:

Amortize - Gradually write off the initial cost over a period.

- It is an analysis technique which can influence the design of algorithm.

* Binary Counting Problem:
By applying traditional approach, complexity is $O(n \log n)$

- If trailing zero is obtained from ~~left~~ right to left, flip it to 1 and stop
- If trailing One is obtained flip it and next flip the first one that is obtained moving from right to left.
- If two consecutive trailing zero is obtained flip both to one and then flip the next zero to one.

* Procedure Increment.

```
i = 0
while A[i] = P do
        A[i] = 0 ; i = i + 1;
end while
A[i] = 1;
```

* Acounting Method

This analysis is based on amortized cost. if the amortized cost exceeds the actual cost, the excess remains with data structure as a credit

Assume

0 → 1 2 Rs.

1 → 0 0 Rs

A[i].	A[3]	A[2]	A[1]	A[0]
	0	0	0	0
	0	0	0	1
	0	0	1	0
	0	0	1	1
	0	1	0	0
	0	1	0	1
	0	1	1	0
	0	1	1	1

2 Rs

At each step we are utilizing 2 rs. and this goes till n So.

time Complexity = $O(2n) = O(n)$

If the amortized cost is small enough so that the actual cost cannot be covered then, it is paid by the credit
Each increament has amortized cost of 2Re
So that there are almost 2n bit changes

* Potential Method :
It is similar to accounting method the only difference is that no explicit credit is saved instead the credit is expressed as potential of the data structure involved with the algorithm.

c_i = Actual Cost of i^{th} Operation.
D_i = Data Structure after i^{th} operation

$\phi(D_i)$ = Potential of D_i

$a_i = c_i + \phi(D_i) - \phi(D_i - 1)$
a_i = amortized cost of i^{th} operation

$$\sum_{i=1}^{n} a_i = \sum_{i=1}^{n} (c_i + \phi(D_i) - \phi(D_i - 1))$$

$$= \sum_{i=1}^{n} c_i + \phi(D_n) - \phi(D_0)$$

If we choose the potential function ϕ such that
$\phi(D_0) = 0$ and $\phi(D_n) \geq 0$

$$\sum_{i=1}^{n} c_i \leq \sum_{i=1}^{n} a_i$$

⇒ amortized cost is upper bound for actual cost

$\phi(D_i) = b_i$

b_i indicates how many total 1's are present in bit stream

$\phi(D_i) - \phi(D_{i-1})$ ← No of 1's in prior stream.

$= b_i - b_{i-1}$

$= (b_{i-1} - t_{i-1} + 1) - b_{i-1}$

$= 1 - t_{i-1}$

$a_i = 1 - t_{i-1}$

t = trailing ones

$c_i = t_{i-1} + 1$

$a_i = c_i + \phi(D_i) - \phi(D_{i-1})$

$= t_{i-1} + 1 + b_i - b_{i-1}$

$= t_{i-1} + 1 + 1 - t_{i-1}$

$= 2$

Therefore: for n number of operation.

$\sum_{i=1}^{n} a_i = 2n$

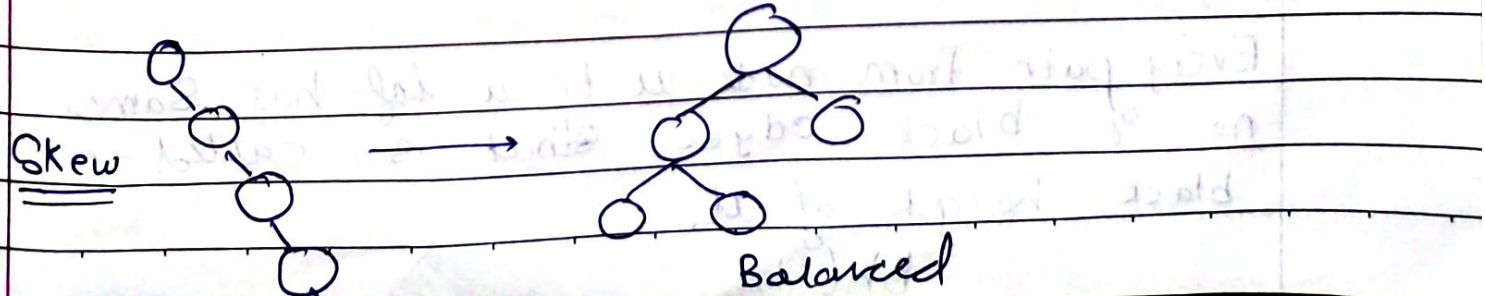
⇒ 2n is upper bound for actual cost

Binary Search Tree.

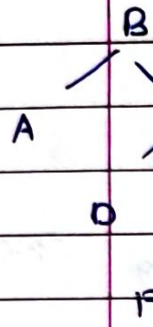
Red Black Tree (Self Binary Tree)

It is also an advanced binary Search tree.

Also called RB Tree.



- Rotation Operation.
  - Single
  - Double Rotation.



Left	Color	Key	Pointer to Parent	Right

edge. We give color to edges.

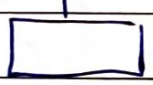
Red. If edge is black then node is black.

black edge

black.

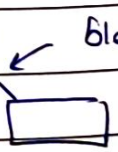
black.

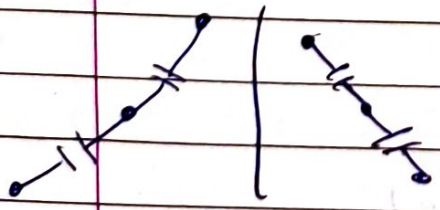
- Leaf Nodes will be empty pointer



empty leaf Node

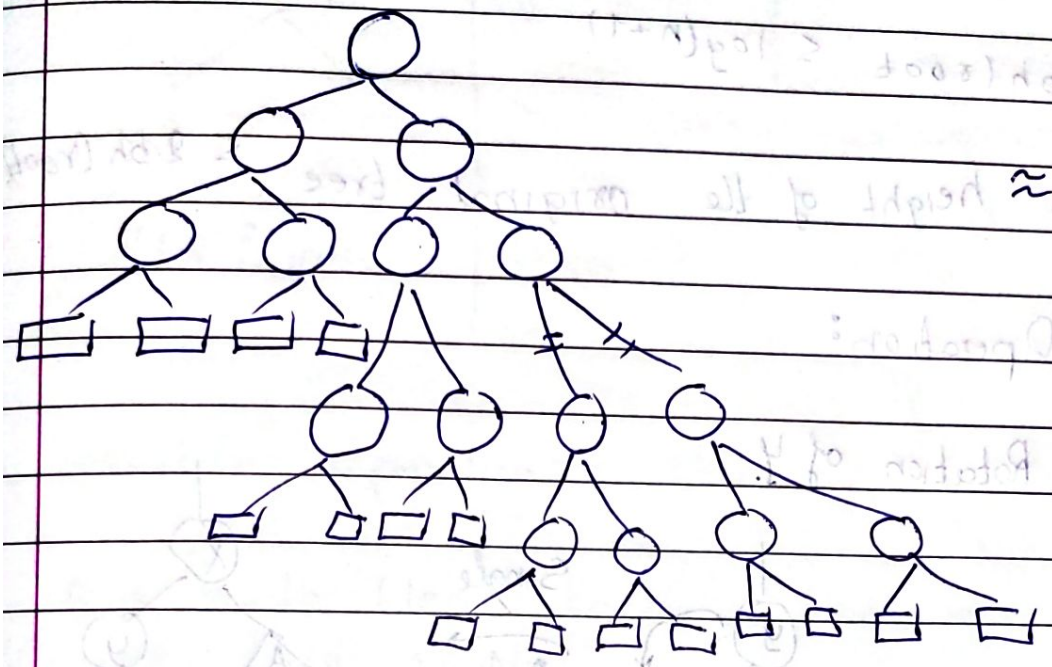
* Properties of R.B tree.
  - each edge has a color either red or black.
  - every edge to a leaf is black.
  - No path has two consecutive red edges.





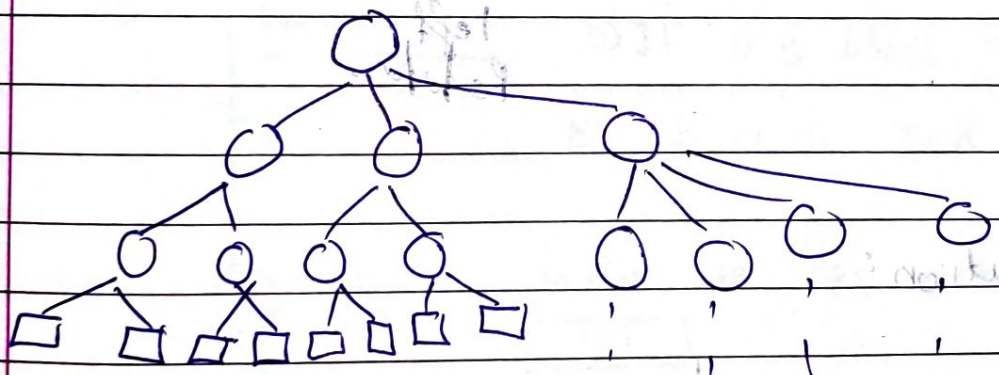
Consecutive red edges.

Every path from node u to a leaf has same no of black edges that is called as black height of u.



≈ Approximately Balanced.

*Claim:
A red-black tree with N internal node has height at most $2 * \lg(n+1)$ or $O(\log n)$



No of node.

no of nodes = n	then leaves = n+1
1	2
n	n+1 ← leaves

In the contraction process, the tree nodes will have degree 2, 3, 4.
The contracted tree has leaves. greater than or equal to $2^{bh(root)}$ leaves $> 2^{bh(root)}$

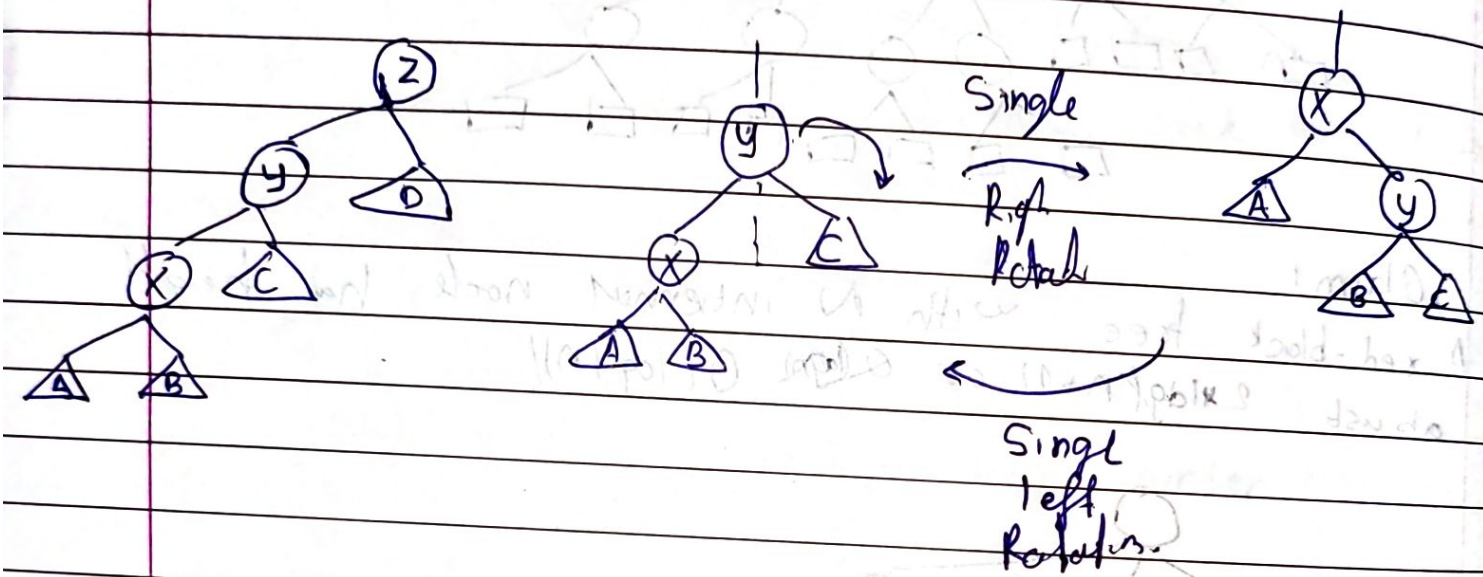
But $2^{bh(root)} < n+1$.

$bh(root) \leq \log(n+1)$

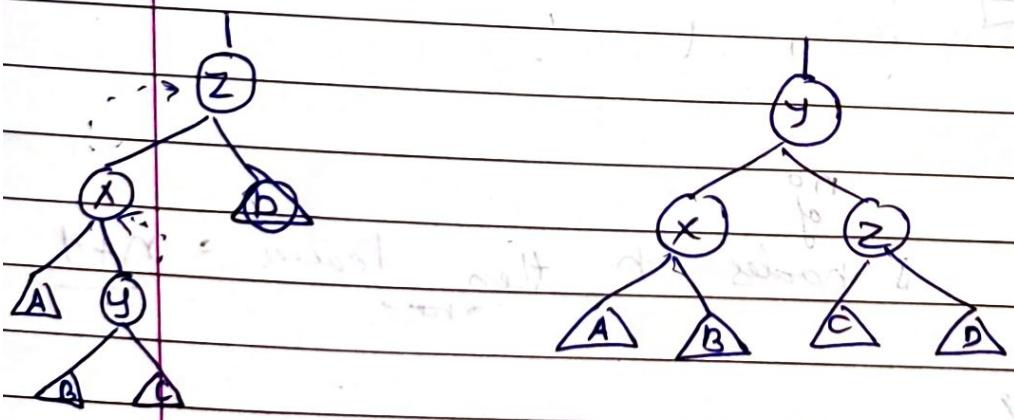
∴ Maximum height of the original tree $\leq 2bh(root)$

* Insertion Operation:

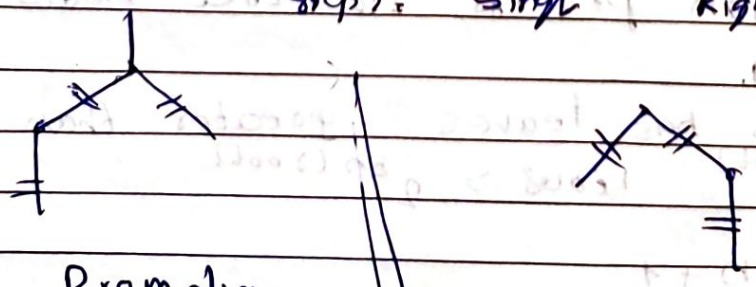
Single left Rotation of y.



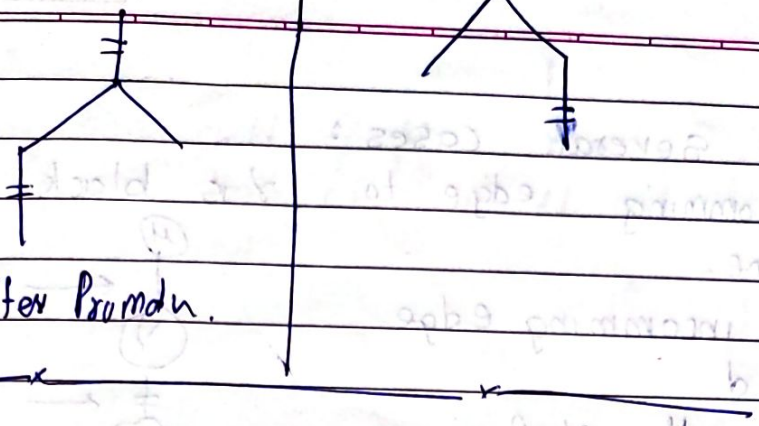
* Double Rotation:



Step: Single left Rotation of y.
Step: Single Right Rotation of y



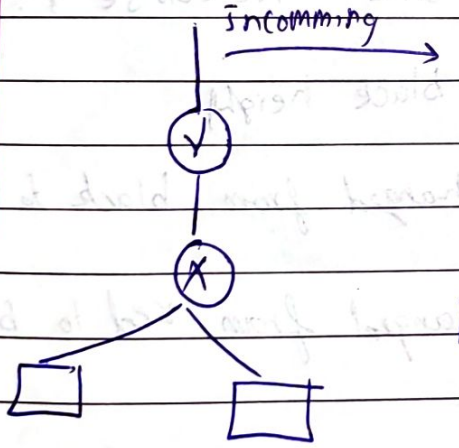
Continued



After Promotion.

* Insertion Operation:

1. At a node (leaf) both children empty or atleast one child empty.
2. Find such node where we can attach new node (x)
3. Insert new node X



a) If it is black ⇒ No problem

b) If it is red ⇒ Violation

When we are attaching any node.

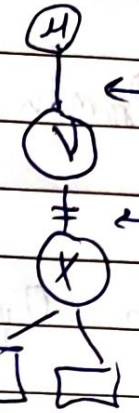
| R |

Incomming edge to root is black (While inserting (If existing tree is not empty) incomming edge is red.

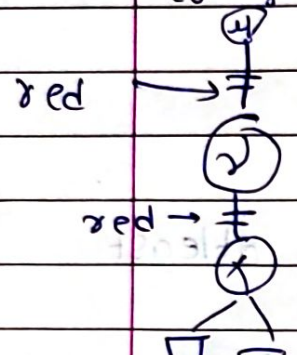
If v has a red incomming edge and a red outgoing edge then this is the only violation of red black tree property.

- There are several cases:

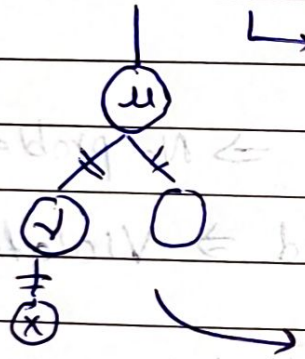
* Case 1: Incomming edge to v is black.
  Stop → Done.

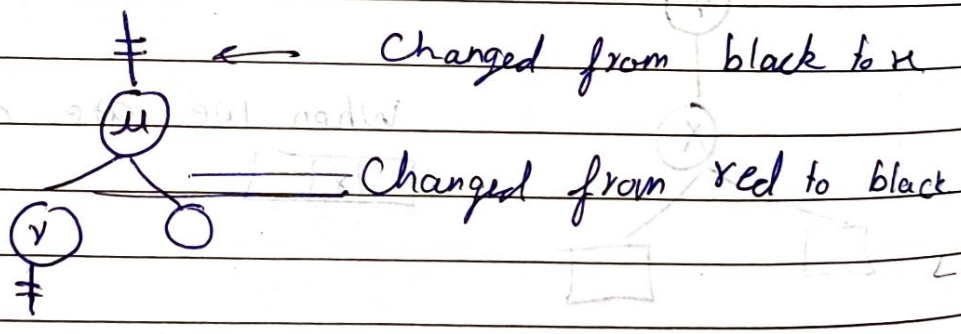


* Case 2: If incomming edge to v is red
  ↳ Set $u = v.p$ where u is parent of u.

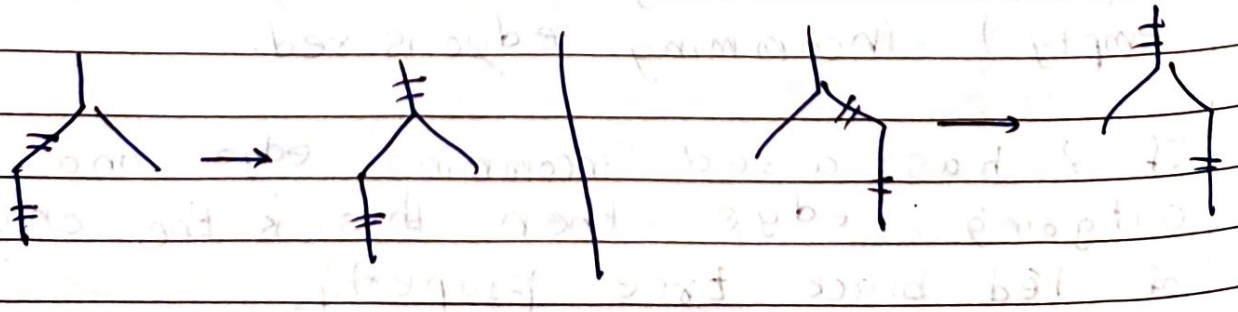


2.1: Both outgoing edges of u are red
↳ Promote u if $u.p \neq$ Nil then $v = u.p$ and Recurse v.
Increase in black height.





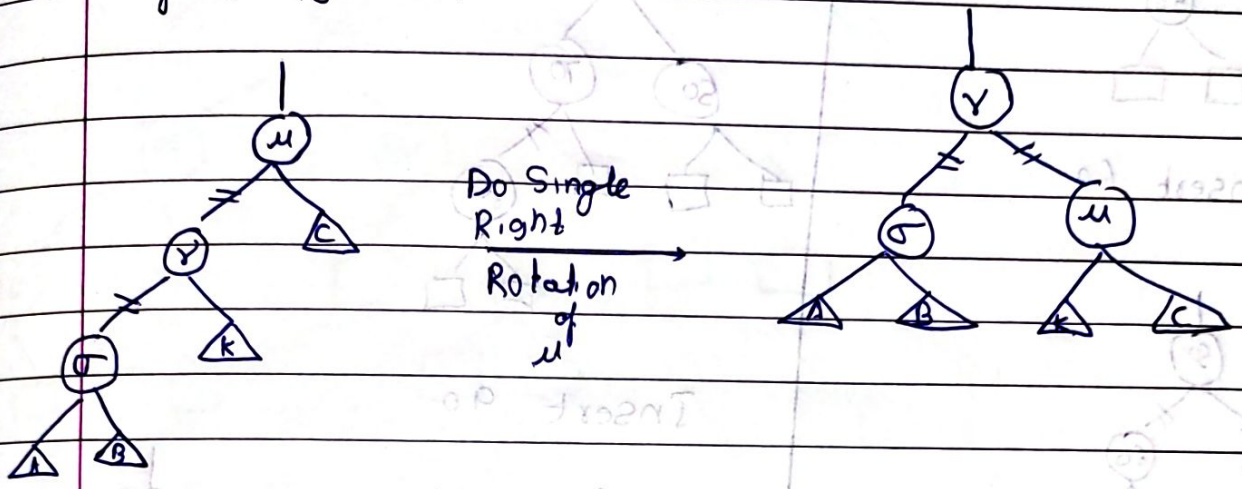
Making this change can lead to violation by $u.p$ So we set $v = u.p$ and recurse.

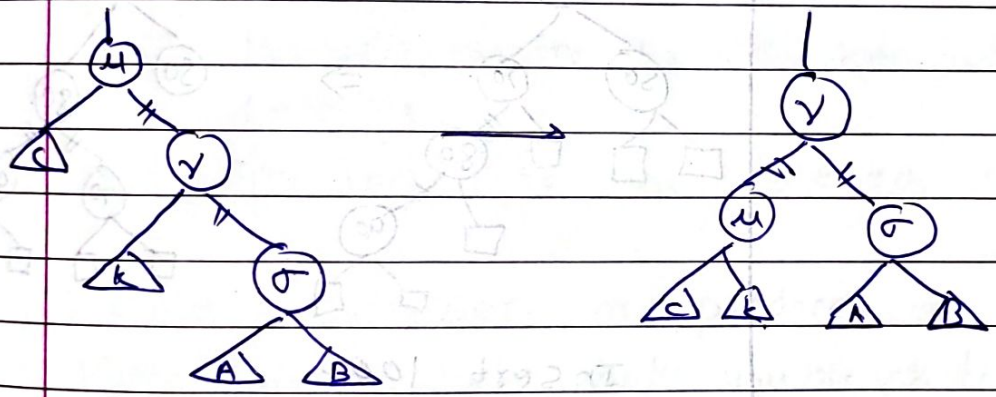


2.2:

Only one child of u is red.

v is the left child of u and left outgoing edge of v is red.



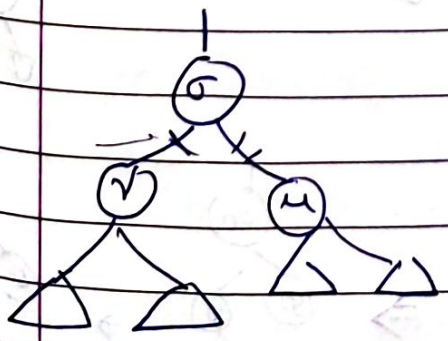


Case 2.3:

v is left child of u.

Double Rotation.

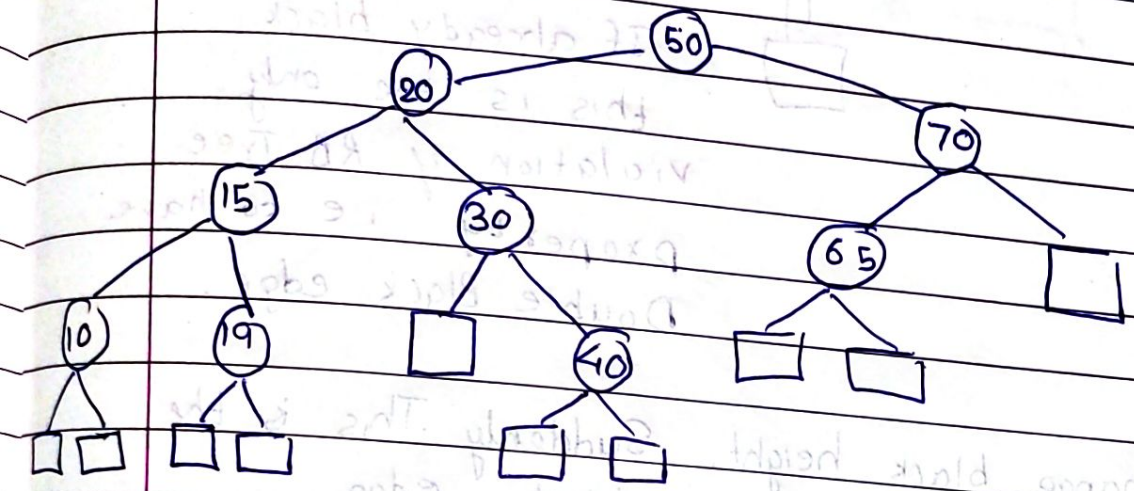
Symmetric wrt left is also applicable



* O(log n)

Completely

* Delete Operation in Red-Black Tree:



Ways to delete

1. Provide pointer to the node, which is to be deleted
2. key value to be deleted, is provided.

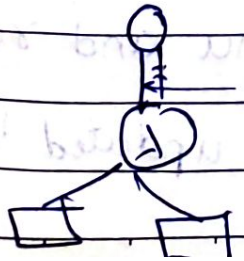
S1: Find successor. or predecessor of Node X repeatatively such that you reach to a node (leaf) ↓ where both sides are empty.

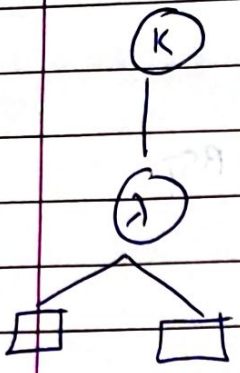
Q

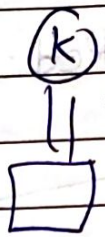
Color
- red
- black
- Double black.

Time Complexity - O(h)

↓ be the node with two leaves







double Black edge

If already black this is the only violation of RB-Tree property i.e to have Double Black edge.

We cant change black height Suddenly. This is the only violation to have double black edge

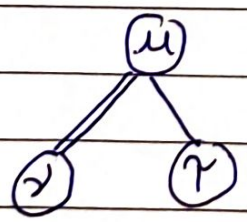
Generic Situation

Below — r ○

this node everything is fine (as per RB Tree Property)

Case 1: Incoming edge of r is double black.

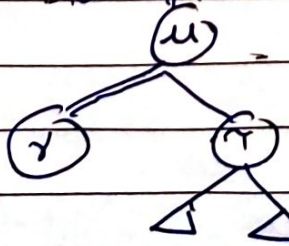
$u = r.p$ T is a sibling of r.



(Always there is a symmetric case)

Case 1.1 Tau is not a leaf.

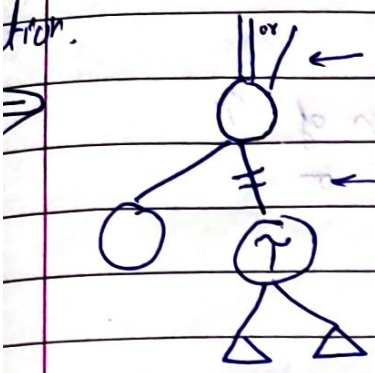
Case 1.1.1 : Both outgoing edges of Tau are black. If red → black.



⇒ Demote mu and recurse on

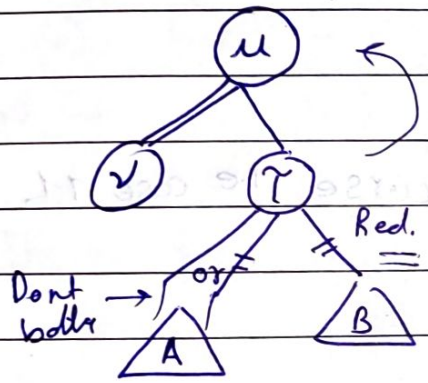
$r := u$. (r is updated to u)

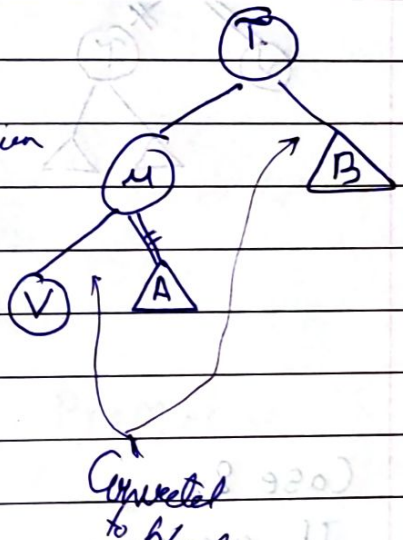
tion.



(Symmetric Case also Applicable)

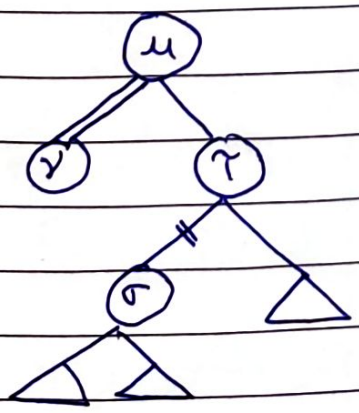
Case 1.1.2 :



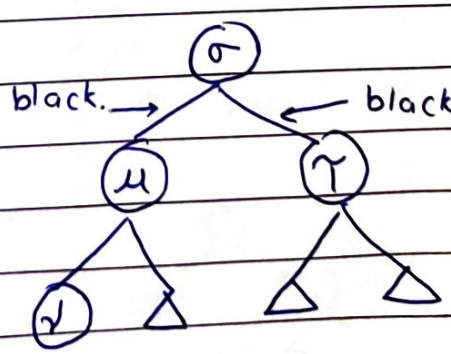


γ is the left child of μ and the right outgoing edge of τ is red.

Case 1.1.3.



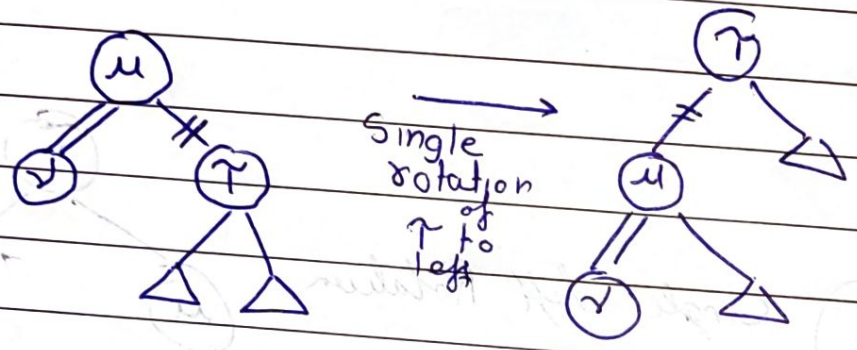
Double Rotation.



left Child of
τ and left outgoing edge of τ is red the other one is black.

We perform double rotation
1. Single right rotation of τ
2. Single left rotation of σ.

Case 1.2



And recurse i.e case 1.1 for v.

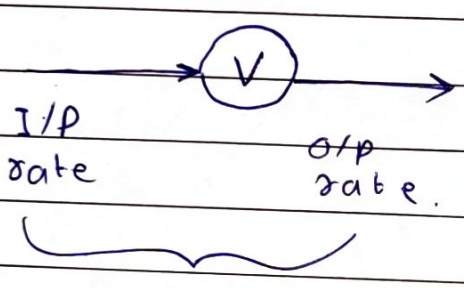
Case 2.
If incomming edge of v is black → we have done STOP

* Flow Networks.

* Directed Graph G(V,E) - V - Set of Vertices
- E - Set of Edges.

s - Source - Where material is produced
t - Sink - Where material is consumed.

Source produces material at some steady rate
Sink consumes the material at same rate



Flow Conservation Property:

* Maximum Flow Problem:
We wish to compute the greatest rate at which we can shift material from source to Sink without violating any capacity constraints

IF I = ?
(Flow Rate)

- for all (u,v) ∈ E
c(u,v) = 0.

- No self loops

we assume G is connected.

Flow : in G is Real Valued functions.
$f : V \times V \longrightarrow R$.

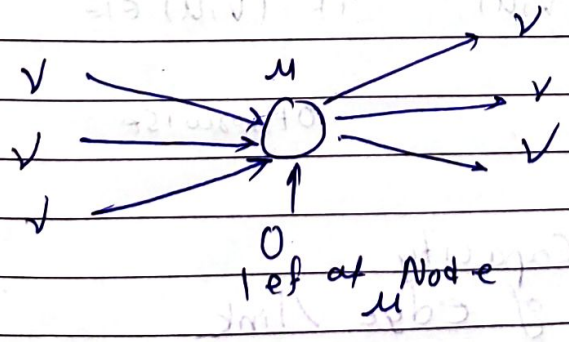
Capacity Constraints.
For $\forall u, v \in X$.
$0 \leq F(u,v) \leq C(u,v)$.

Residual Capacity.
$C(u,v) = 100$ Units
$F(u,v) = 20$ Units
Residual Capacty $= 100 - 20$ Units.
$= 80$ units

2) $V_{u,v} \in V$
$F(u,v) = -F(v,u)$

3) for all $u \in V - \{s,t\}$.

$\sum_{v \in V} f(u,v) = 0$ (flow Conservation



The value of flow is $|f| = \sum_{v \in V} F(s,v)$

Max Flow Problem is to find max $|f|$
$|f| = \sum_{v \in V} f(s,v) - \sum_{v \in V} F(v,s)$

Date:

or.

$$\sum_{v \in V} F(s,v) = \sum_{v \in V} F(v,s)$$

Max flow min cut theorem:

Some of the properties of float.

$$f(X,y) = \sum_{x \in X} \sum_{y \in Y} f(x,y)$$

for two sets $X, Y \subseteq V$

1. $f(x,x) = 0$
   $x \subseteq X$
2. $f(x,y) = -f(y,x)$
3. $F(X \cup Y, Z) = F(x,z) + F(y,z)$
4. $F(z, X \cup Y) = F(z,x) + F(z,y)$

* Ford Fulkerson Method:

uses the concept of residual capacity.

$$C_f(u,v) = \begin{cases} C(u,v) - F(u,v) & \text{if } (u,v) \in E \quad \text{—①} \\ f(v,u) & \text{if } (v,u) \in E \\ 0 & \text{otherwise} \end{cases}$$

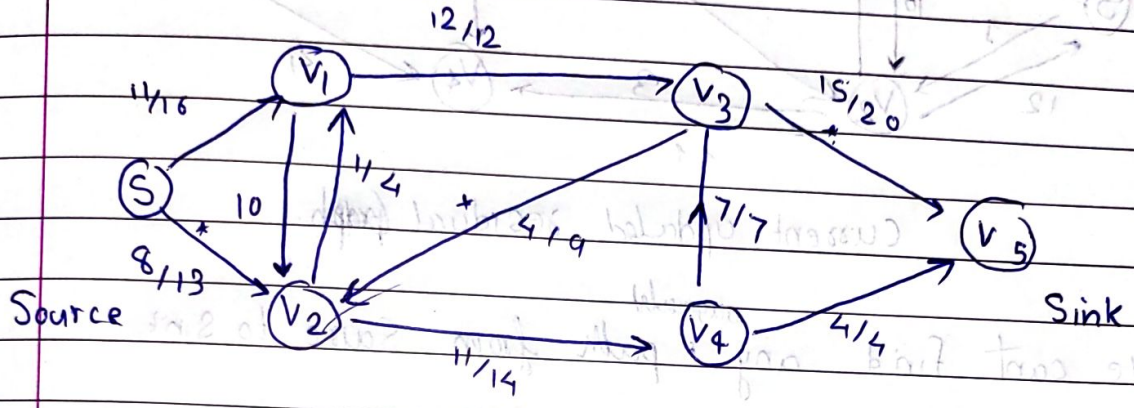
$C_f \rightarrow$ Residual Capacity.

$C \rightarrow$ Capacity of edge / link

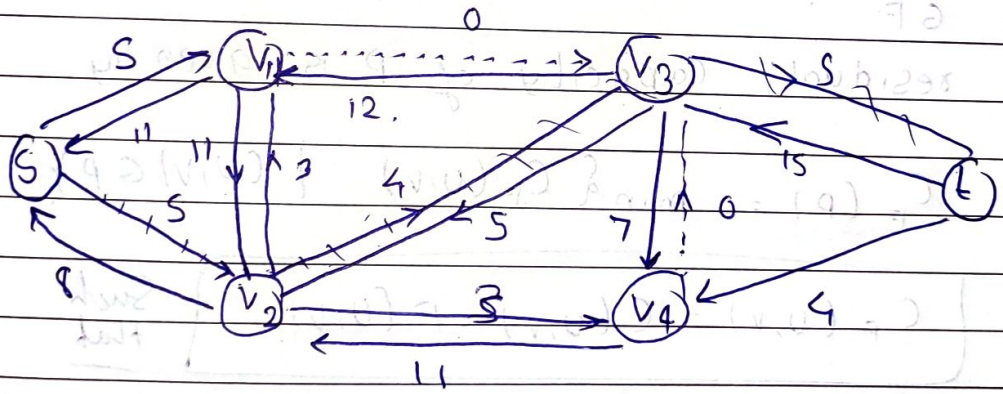
For the given N/w $G = (V, E)$ and a flow f, the residual N/w is $G_f = (V, E_f)$

Where

$$E_f = \{ (u,v) \in V \times V \mid c_f(u,v) > 0 \}$$

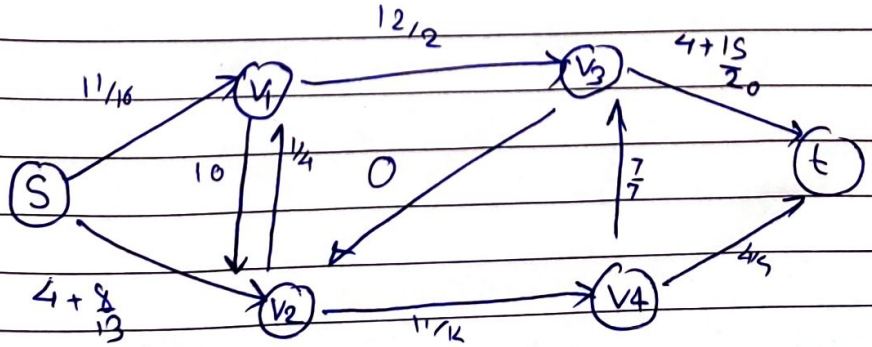


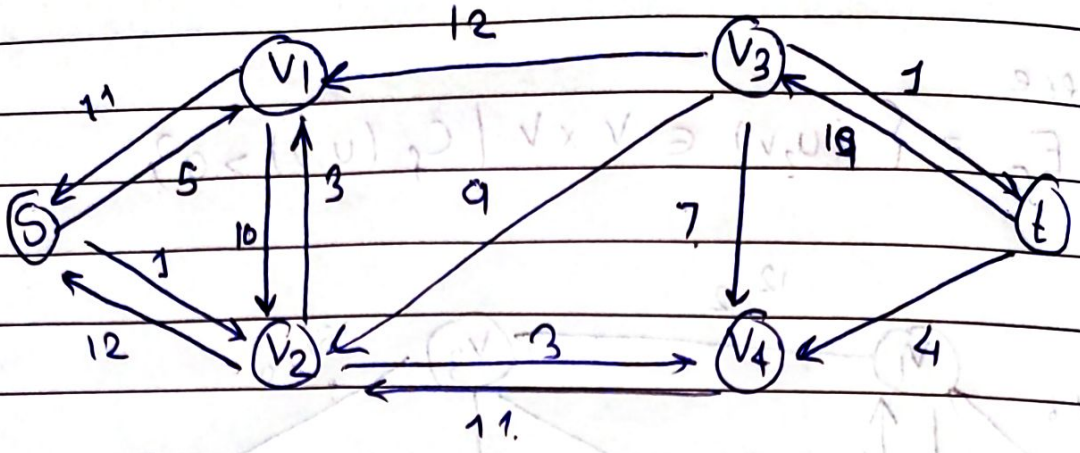
G = {



Lets Assume SV_2V_3t path
This is the augmented path.

min (5, 4, 5) = 4.





Current updated residual ~~graph~~ augmented

We can't find any path from Sour to Sink

Augmeting Path:

A path p from 's' to 't' in the residual N/w GF

The residual capacity of p is given by

$$C_f(p) = \min\{ C_f(u,v) \mid (u,v) \in p \}$$

$$C_f(u,v) = C(u,v) - f(u,v)$$ Such that