$T(n) = \frac{16}{13} n^2 + n^{\log_4 3}$

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

→ Module-3

(Greedy Method) Dynamic programming method.

→ Greedy Method :-

problem P → no. of inputs

Select some inputs → solution 1 } possible solution or
from set of i/p   " → solution } feasible solution.

· Identify Best solution/ from feasible solution
optimal

To find it. use a (objective function) ↪ max. profit or min. cost.

Example:-

Customer went to shop
purchased for 48/-
he have 100/- given to shop keeper
He have to return ~~the~~ 52/- to customer → Solution.

① To get the shop keeper uses
I/P:- 2000, 500, 200, 100, 50, 20, 10, 5, 2, 1 (currency notes)

② Solution 1 → <50, 2> } feasible solution.
" 2 → <50, 1, 1>
" 3 → <20, 20, 10, 2>

③ Which is best amoung feasible solution. by objective function (using min. no. of notes)
<50, 2> → optimal.

- Applications :-
1.) Fractional knapsack problem
2.) Minimum cost spanning tree problem.

3. Single source shortest path problem
4. Job sequencing with Dead lines problem
5. String Matching problem.

- Fractional Knapsack problem:-

0. Greedy !-

Bag with capacity given by m

No. of items = n

Items into bag → In any order (at a time only one item should be placed)

Item — weight = w_i
Item — profit = P_i

Item into bag
↳ Capacity of bag ~~increases~~ decreased by weight of items

ex: $m = 30 - 15 = 15$
$w_1 = 15$

Firstly verify is it possible to place full item ~~can~~ be into the bag so that profit is full else some part; part of profit

$x_i = 0 | 1 | 1/2 | 1/3$
↓
how much part of item is placed in bag.

① Sum of weights of items placed into bag shouldn't exceed bag capacity.

② placing items in bag should get max profit (objective function)

Ex:- Find the optimal solution for the following knapsack problem

capacity of bag => $m = 20$

6 ways to place items.

No. of items $n = 3$

weights of items $(w_1, w_2, w_3) = (\overset{1}{18}, \overset{2}{15}, \overset{3}{10})$

profits of items $(P_1, P_2, P_3) = (25, 20, 15)$

Solution → weights with less weight considered 1st
(or)
profit with high profit considered 1st.

1.) per unit profit of items: $25/18, 20/15, 15/10$
$= \underset{1}{1.38}, \underset{2}{1.33}, \underset{3}{1.5}$

2.) Identify order for placing items: 3, 1, 2 (as per (per unit profit))

$W \Rightarrow w_3 * x_3 = 10 * 1 = 10$

$P \Rightarrow P_2 * x_3 = 15 * 1 = 15$

$m = 20 - 10 = 10 \rightarrow$ remaining bag capacity.

$m = 10$

$x_1 = 18 \rightarrow 10/18 \; (\overset{w\Rightarrow}{w_1 * x_1}) \quad 18 * \frac{10}{18} = 10$

$10 < 18 \rightarrow$ Not possible $\qquad \overset{P\Rightarrow}{(P_1 * x_3)} = 25 * \frac{10}{18}$

∴ Out 18 can place 10 units.

Profit = 13.88

$m = 10 - 10$
$= 0$

$x_2 \rightarrow$ can't be place ∴ zero "0" $\quad w = 15 * 0 = 0$
$w = 0;\ p = 0 \qquad P = 20 * 0 = 0$

Total Profits $= 15 + 13.88$
$= 28.88$

Optimal solution $= \langle x_1, x_2, x_3 \rangle = \langle 10/18, 0, 1 \rangle \Rightarrow \underset{\text{max.profit}}{\underset{\downarrow}{28.88}}$

Ex: Capacity of bag $m = 10$

no. of items $n = 7$

weights of items $= (w_1\ w_2\ w_3\ w_4\ w_5\ w_6\ w_7)$
$= (1, 4, 3, 2, 3, 6, 7)$

profits of items $= (P_1\ P_2\ P_3\ P_4\ P_5\ P_6\ P_7)$
$(7\ 18\ 6\ 7\ 5\ 3\ 4)$

① per unit profit of item $= \frac{7}{1}, \frac{18}{4}, \frac{6}{3}, \frac{7}{2}, \frac{5}{3}, \frac{3}{6}, \frac{4}{7}$

$= (7, 4.5, 2, 3.5, 1.66, 0.5, 0.57)$

② Identify order for placing $= (1, 2, 4, 3, 5, 7, 6)$

$m = 10$

$x_1 = 1 \Rightarrow w = w * x_1 = 1 * 1 = 1$

$P = P_1 * x_1 = 7 * 1 = 7$

$m = 10 - 1 = 9$

$m = 9$

$x_2 = 1$

$w = 4 * 1 = 4$

$p = 18 * 1 = 18$

$m = 9 - 4 = 5$

$m = 5$

$x_1 = 1$

$w = 2 * 1 = 2$

$p = 7 * 1 = 7$

$m = 5 - 2 = 3$

$m = 3 \quad x_1 = 1 \quad w = 3 * 1 = 3$

$p = 6 * 1 = 6$

$m = 3 - 3 = 0$

$x_5 = 0, w = 0, p = 0$

$x_7 = 0, w = 0, p = 0$

$x_6 = 0, w = 0, p = 0$

$\therefore$ Total profit $= 7 + 18 + 7 + 6$

$\Rightarrow 38.$

$\langle x_1, x_2, x_3, x_4, x_5, x_6, x_7 \rangle$

$\Rightarrow \langle 7, 18, 6, 7, 0, 0, 0 \rangle$ = profit

$= 38.$

$\rightarrow \langle 1, 1, 1, 1, 0, 0, 0 \rangle$

→ Algorithm of Knapsack problem :-

Ex: Find the optimal solution for following knapsack problem.

$m = 12 \quad n = 4$

$(w_1, w_2, w_3, w_4) = (4, 7, 5, 7)$

$(P_1, P_2, P_3, P_4) = (8, 7, 7, 6)$

1. per unit profit of item $= \frac{8}{4}, \frac{7}{7}, \frac{7}{5}, \frac{6}{7}$

$= (2, 1, 1.4, 0.8)$

2. Identify order for placing $= (1, 3, 2, 4)$

- $m = 12$

$\boxed{x_1 = 1}$

$w = x_1 \times w_1$

$= 1 \times 4 = 4$

$P = P_1 * x_1$

$= 8 * 1 = 8$

$\boxed{P = 8}$

$m = 12 - 4 = 8$

- $m = 8$

$\boxed{x_3 = 1}$

$w = 5 \times 1 = 5$

$P = P_3 * x_3$

$= 7 \times 1 = 7$

$\boxed{P = 7}$

$m = 8 - 5 = 3$

- $m = 3$

$\boxed{x_2 = 3/7}$

$w = 7 \times \frac{3}{7} = 3$

$w = 3$

$P = P_2 \times x_2$

$= 7 \times \frac{3}{7} = 3 \quad \boxed{P = 3}$

- $m = 3 - 3 = 0$

$m=0$

$x_4 = 0$, $P=0$

Totalprofit = $7+8+3 = 18$

optimal solution = $\langle x_1\ x_3\ x_2\ x_4 \rangle \Rightarrow \langle x_1, x_2, x_3, x_4 \rangle$

$= \langle 1, 1, 3/7, 0 \rangle \quad \langle 1, 3/7, 1, 0 \rangle$

→ Algorithm:-

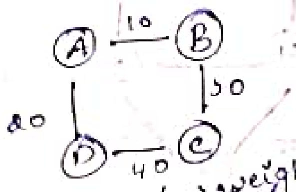
```
Algorithm Knapsack(m, n, w, P, x, profit){
//'m' is capacity of bag
//'n' is no. of items
//'w' is an array containing weights of items.
     size of 'w' is 'n'.
//'p' is an array containing profits of items size of 'p' is 'n'.
//'x' is an array of size 'n' used to store optimal solution.
//'profit' is used to store maximum profit.

    profit = 0;
    for (int i=1; i<=n; i++){
        pp[i] = p[i] / w[i];
          x[i] = 0;
    }
    for (i=1; i<=n-1; i++){
        for (j=i+1; j<=n; j++){
            if (pp[i] < pp[j]){
                swap(pp[i], pp[j]);
                Swap(w[i], w[j]);
                Swap(p[i], p[j]);
            }
        }
    }
    for(i=1; i<=n; i++){
        if (w[i] > m) {
            k=i;
            break;
        }
```

```
else{
      x[i] = 1;
      m = m- w[i] * x[i];
      profit = profit + p[i] * x[i];
   }
}
  x[i] = m/w[i]; (or) x[K] = m/w[K];
  profit = profit + p[i] * x[i];
}* for (i=1; i<=n; i++)
         write x[i];
      write profit;
}*
```

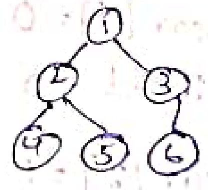
→ **Minimum Cost Spanning Tree (MCST)** :-

Graph :- forms cycles
- undirected
- directed

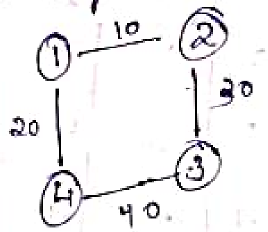


↳ weighted undirected graph.

Trees:- doesn't form cycles

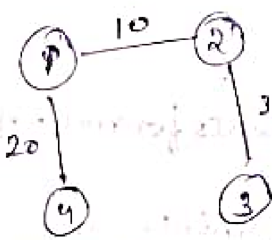


For minimum spanning remove cycles.



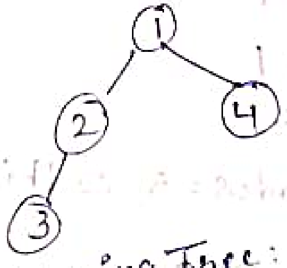
⇒

①



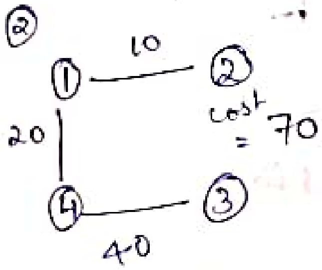
cost = 60

=)

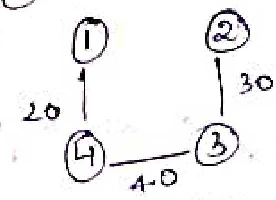


↳ spanning Tree: obtained by converting graph to tree with removing edges.

②

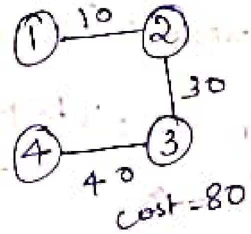


③



cost = 90

④



cost = 80

as minimum cost is 60 it is minimum spanning Tree.

Given weighted graph

no. of nodes = n } for spanning Tree.
no. of edges = n-1 }

- Adjacency Matrix :- (n × n size)
(or) cost matrix (or) weight matrix

	1	2	3	4
1	∞	10	∞	20
2				
3				
4				

matrix form :-

n-1

	1	2
1	1	2
2	1	4
3	2	3

1.) Prim's algorithm

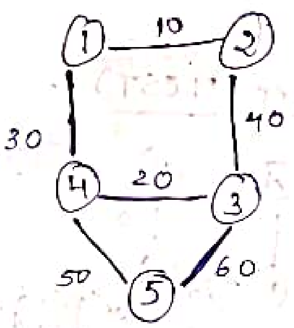
2.) Kruskal's algorithm

→ Prism's algorithm :-

1.) Select a node 1

near [1] = 0

min cost = 0



Cost :-

	1	2	3	4	5
1	∞	10	∞	30	∞
2	10	∞	40	∞	∞
3	∞	40	∞	20	60
4	30	∞	20	∞	50
5	∞	∞	60	50	∞

2.) near [2] = 1

near [3] = 1

near [4] = 1

near [5] = 1

3.) as no. of nodes = 5 so this repeats for n-1 = 4 iterations

MCST

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

Iteration 1:-

a) cost [2, near[2]] = 10 → minimum cost should be identify

cost [3, near[3]] = ∞

cost [4, near[4]] = 30

cost [5, near[5]] = ∞

b) Add edge (1, 2) to MCST

near [2] = 0 → To indicate it is already added to MCST

min cost = 0 + cost [1, 2] = 0 + 10 = 10,

c) near[3] = 2 → B/w 1, 2 consider minimum value.

near[4] = 1 update near values which are not used.

near[5] = 1

↳ If (∞, ∞) both are same as they are not min then previous value.

Iteration 2 :-

a) cost [3, $\overset{2}{\text{near}}$[3]] = 40
cost [4, near [4]] = 30 → minimum.
cost [5, near [5]] = ∞

b) add edge (1, 4) to MCST
near [4] = 0
min cost = 10+30 = 40

c.) near [3] = 4
near [5] = 4

Iteration 3:-

a) cost [3, $\overset{4}{\text{near}}$ [3]] = 20 → minimum
cost [5, near [5]] 50

b) add edge (3, 4) to MCST
near [3] = 0
min cost = 10 + 30 + 20 = 60

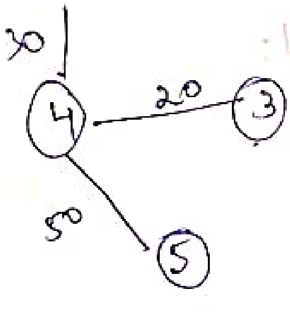
c) near [5] = 4

Iteration 4:-

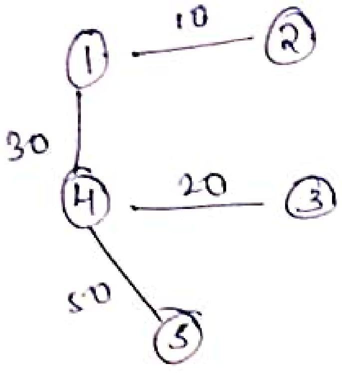
a) cost [5, $\overset{4}{\text{near}}$ [5]] = 50 → minimum

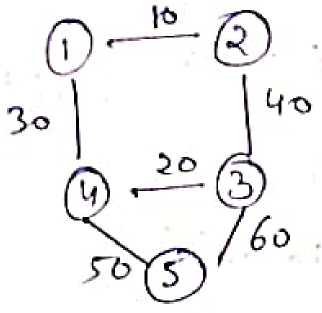
b) add edge (5, 4) to MCST
near [5] = 0
min cost = 10+30+20+50 = 110

c.) —



MCST :- 110





```
Algorithm prim's(n, cost, t){
//'n' is no. of nodes in the graph
//'cost' is adjacency matrix of the graph. Size of 'cost' is 'nxn'.
//'t' is matrix used to store edges of MCST. Size of 't' is 'n-1x2'
        near[1]=0;
        mincost = 0;

        for(i=1; i<=n; i++){
                if(near[] ≠ 0){
                        near[i]=i;
                }
        }

        for(x=1; x<=n-1; x++){
                min=∞;
            for(i=1; i<=n; i++){
                    if(near[i]≠0){
                            if(cost[i,near[i]]<min){
                                    min = cost[i, near[i]];
                            }
            }       }                r=i;
                                     c= near[i];
                            }
            }       }
```

```
b)      t [r,1]= r;
        t [r,2] = c;
        min cost = mincost + cost [r, c];
        near[r]=0;

c)   for(i=1; i<=n; i++){
          if (near[i]≠0){
               if(cost[i, new[i]] > cost [i, r]){
                        near[i]= r;
               }
          }
     }
  }*
]
```

→ Kruskal's algorithm :-

1.) Construct Minheap with weights of edges in given graph.

2.) Construct Disjoint sets with nodes of given graph.

3.) Repeat until Minheap ~~haver~~ becomes empty.

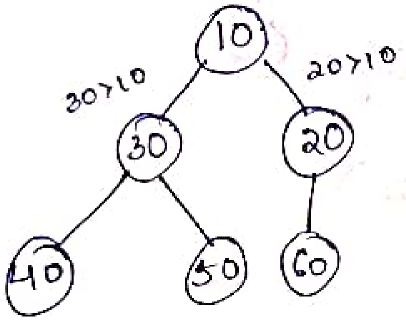
4.) a) Delete root node from Minheap & reconstruct the Minheap.

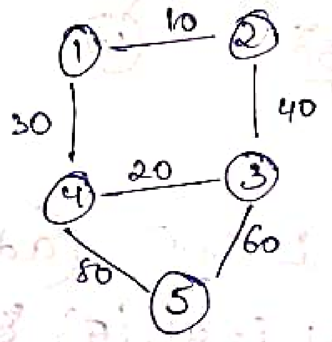
b.) Identify the corresponding edge.

c.) If the nodes of edge are in different disjoint sets, then include the edge in MCST & continue the corresponding disjoint sets otherwise, discard the edge.

[root < child]

1.) 10, 30, 20, 40, 50, 60





2) {1}, {2}, {3}, {4}, {5}   No. of disjoint sets are no of nodes present in graph.

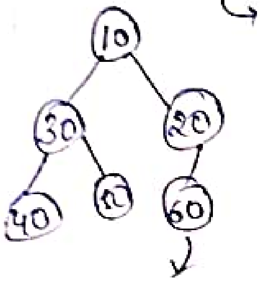
	1	2
1	1	2
2	3	4
3	1	4
4	4	5

3) Iteration1:-

a) Delete 10

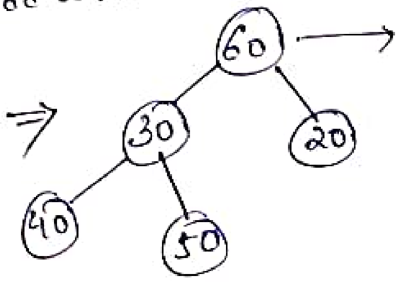
b.) edge is (1,2)

c) {1,2}, {3}, {4}, {5}
↳ add into same set.



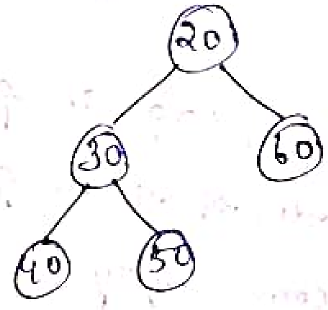
take Last node as root.

⇒



→ Not Satisfing the condition of minheap so take the min child & make it as root

⇓

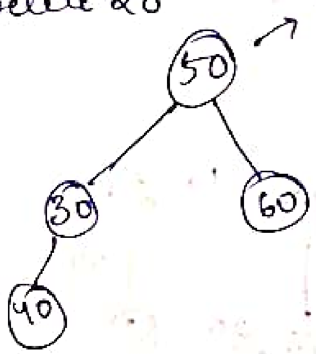


b) edge is (1,2)

c.) {1,2}, {3}, {4}, {5}

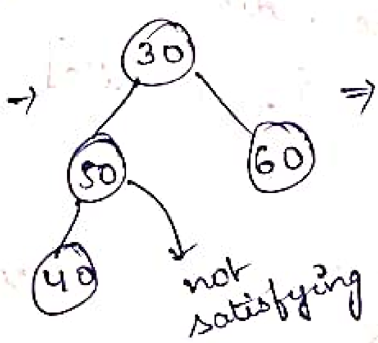
Iteration 2:-

a) Delete 20

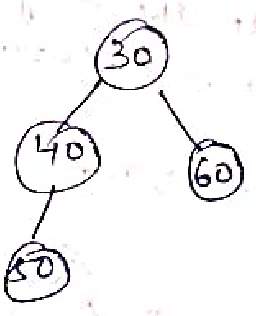


→ not satisfying

→



⇒

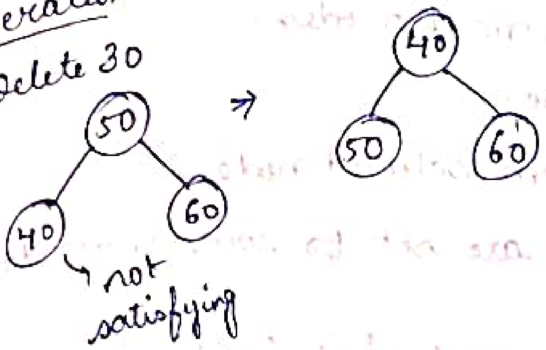


b.) edge is (3,4) (add 3,4 to MCST)

c.) {1,2}, {3,4}, {5}

Iterations:-

a.) Delete 30

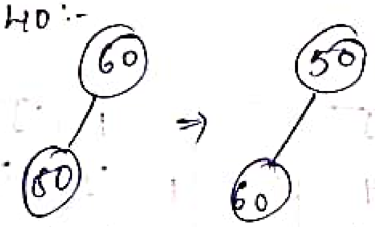


b.) edge is (1, 4)

c.) {1, 2, 3, 4} {5}.

Interation 4:-

a.) Delete 40:-



b.) edge is (2,5) ↳ same disjoint set then discard ↳ no need to add in MCST.

c.)

Iteration 5:-

a.) Delete 50:

b.) edge is (4,5)

c.) {1, 2, 3, 4, 5}

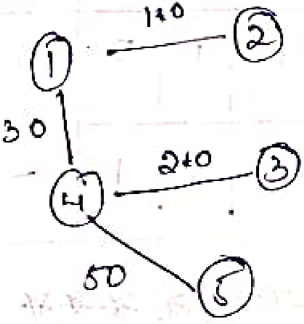
Iteration 6:-

a.) Delete 60:- (as empty iterations ends)

b.) edge is (3, 5)

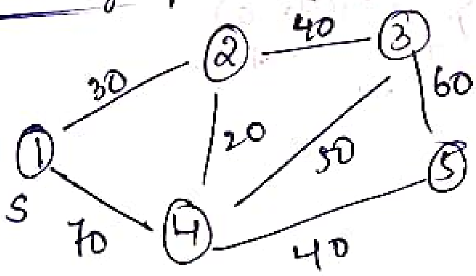
c.) Discard (3, 5)

Final MCST:-



→ Single Source Shortest path:-

weighted graph:-



Inputs:- 1) Weighted graph

2) Source node (any node in graph)

Cost:-

	1	2	3	4	5
1	∞	30	∞	70	∞
2	30	∞	40	20	∞
3	∞	40	∞	50	60
4	70	20	50	∞	40
5	∞	∞	60	40	∞

Dijkstra's Algorithm:-

- We get (n+1) iterations, where 'n' is no. of nodes.
- Node selected is initially source node.
- dist[i] ⇒ distance (cost) of each node from selected node.
- Pr[i] ⇒ All nodes except source node are set to source node, Pr of source node is set to zero.
- initially all set[] ⇒ 0 ⇒ no node is included in the set.
- dist[i] ⇒ Current shortest distance b/w source node & destination node

Pr[i] ⇒ the via node through which we get that shortest distance

Dijkstra's algorithm:

n=5, n+1=6 rows.

Iterations	Set[i]					Node Selected (u)	dist[i]					Pr[i]				
	1	2	3	4	5		1	2	3	4	5	1	2	3	4	5
Initial	0	0	0	0	0	1	∞	30	∞	70	∞	0	1	1	1	1
1	1	0	0	0	0	2	∞	30	70	50	∞	0	1	2	2	1
2	1	1	0	0	0	4	∞	30	70	50	90	0	1	2	2	4
3	1	1	0	1	0	3	∞	30	70	50	90	0	1	2	2	4
4	1	1	1	1	0	5	∞	30	70	50	90	0	1	2	2	4
5	1	1	1	1	1	-	∞	30	70	50	90	0	1	2	2	4

shortest paths

To node 2	To node 3	To node 4	To node 5
pr[2] = 1	pr[3]=2	pr[4]=2	pr[5]=4
	pr[2] = 1	pr[2] = 1	pr[4] = 2
			pr[2] = 1
$1 \xrightarrow[30]{} 2$	1 → 2 → 3	4 ← 2 ← 1	5 ← 4 ← 2 ← 1
2 ← 1	3 ← 2 ← 1		

→ Algorithm:-

```
Algorithm Dijkstras(n, cost, s)*{
//'n' is no. of nodes in the given graph.
//'cost' is adjacency matrix of given graph.
//'s' is source node in the given graph.
    for(i=1; i<=n; i++){
        set[i] = 0;
        dist[i]= cost[s,i];
        Pr[i] = s;
    }
    u=s;
    pr[s] =0;
    for(x=1; x<=n; x++){
        set[u]=1;
        min = ∞;
      for(i=1; i<=n; i++){
          if(set[i] = =0){
              if(dist[i] < min){
                  min = dist[i];
                  u = i;
              }
          }
      }

      for(i=1; i<=n; i++){
          if(set[i]==0){
              if(dist[i] > dist[u]+ cost[u, i]){
                  dist[i] = dist[u] + cost[u,i];
                  Pr[i] = u;
    }   }   }
```

```
for(i=1;i<=n;i++){
    if(s ≠ i){
        K= i;
        write K;

        while(K ≠ s){
            K= pr[K];
            write("<—",K);
        }
    }
 }
}
```

→ **Job Sequencing With Deadlines problem:-**

→ finding max profit

n- Jobs
↳ n no. of Jobs are executed by 1 machine

1- machine

Execution time of each Job is 1 unit of time = 1sec.

- If 1 is deadline

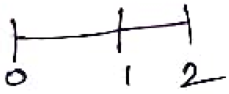

should be completed in this slot

Job
- deadline - ⓓⁱ
- profit - P_i

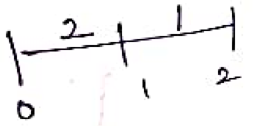
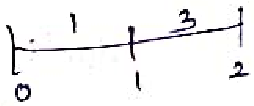
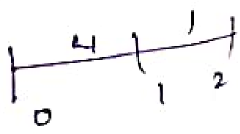
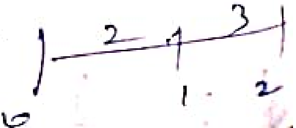
if job is completed before deadline then the corresponding profit is achieved.

Ex: n = 4 $(d_1, d_2, d_3, d_4) = (2, 1, 2, 1)$

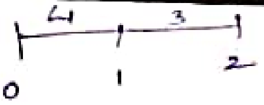
$(P_1, P_2, P_3, P_4) = (100, 10, 15, 27)$



as max is '2'

Feasible Solutions	Job sequence	Profit	
{1, 2}	2 , 1	100+10=110	
{1, 3}	1 , 3 (or) 3 , 1	15+100=115	
{1, 4}	4 , 1	100+27=127	
{2, 3}	2 , 3	10+15 =25	

{3,4}   4, 3   15+27 = 42



{2,4} → Not feasible

Finding max profit

127 → {1,4} → optimal solution.

ex! $n = 5$

$(P_1, P_2, P_3, P_4, P_5) = (15, 1, 20, 10, 5)$

$(d_1, d_2, d_3, d_4, d_5) = (2, 3, 2, 1, 3)$

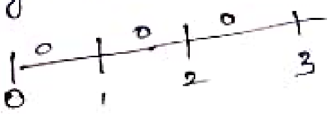
1.) Sort the jobs in decreasing order of profit

$(P_1, P_2, P_3, P_4, P_5) = (20, 15, 10, 5, 1)$

$(d_1, d_2, d_3, d_4, d_5) = (2, 2, 1, 3, 3)$

2.) Identify maximum deadline - max d - 3

3.) Initialize the slots as empty.

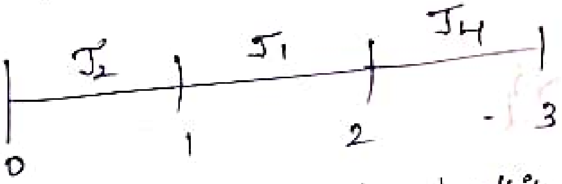


slot[1] = 0
slot[2] = 0
slot[3] = 0

4) Repeat the following for each Job.

a.) Identify the slot for the Job by verifying the slots from the slot indicated by the deadline of the Job until the first slot.

b.) If a slot is identified for the Job, then update the profit



Profit: 20
Profit = 20 + 15 = 35
Profit = 35 + 5 = 40

J3 is 1 can't be placed skip it

∴ max profit = 40

J5 is 3 can't be placed skip it

optimal solution - {1,2,4} - 40

→ Algorithm:-

```
Algorithm JSWD(n, P, d){
//'n' is no. of Jobs
//'p' is profit 'P[1...n]' is an array containing
                           profits of items.
//'d[1..n]' is an array containing deadlines of Jobs.
        maxd = 0; profit = 0;
        for(i=1; i<=n-1; i++){
            for(j=i+1; j<=n; j++){
                if(P[i] < P[j]){
                    swap(P[i], P[j]);
                    Swap(d[i], d[j]);
                }
            }
        }
        for(i=1; i<=n; i++){
            if(d[i] ≥ maxd){
                maxd = d[i];
            }
        }
        for(i=1; i<=maxd; i++){
            slot[i] = 0;
        }
        for(i=1; i<=n; i++){      // i → Job
            for(j=d[i]; j>=1; j--){      // j → slot
                if(slot[j] == 0){
                    slot[j] = i;
                    profit = profit + P[i];
                    break;
                }
            }
        }
        for(i=1; i<=maxd; i++){
            write slot[i];
        write profit;
}
```

→ Dynamic Programming Method :-

Problem (P) — no. of inputs
set of inputs
↓
subset of inputs → solution 1
→ solution 2 } different solution i.e feasible / possible solutions.
objective function | find
↓
best solution / optimal solution.

Greedy	Dynamic Programming
1.) No guarantee of optimal solution.	1.) Will Guarantee optimal solution of the problem.
2.) No need to divide the problem	2.) Divide the problem into no. of sub problems & find optimal solutions of sub problems & store optimal solution of sub prob. combine ~~consider~~ optimal solutions of sub problems to get optimal solution of entire problem.
(3,4,5 based on 2.)	
3.) Less complex	3.) More complex.
4.) More Efficient	4.) Less efficient
5.) Less time	5.) More time

→ Applications of Dynamic programming method :-

1.) All pairs shortest path problem
2.) 0/1 Knapsack
3.) Matrix chain multiplication
4.) Travelling salesperson
5.) optimal Binary Search Tree.
6.) Longest common subsequence

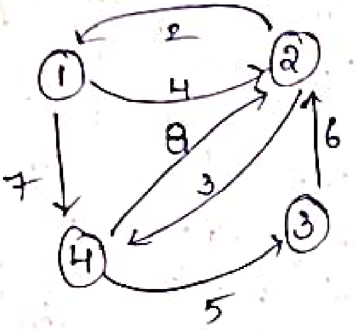
- All pairs shortest paths :-

Input :- Weighted Graph

→ should be shortest path b/w all pairs
(1,2), (1,3), (2,1), (2,3), (3,1) (3,2)

→ By applying floyd's algorithm we find shortest path.

Weighted Graph :-



n = 4

$A^0 \cdots A^n \Rightarrow$ matrix

$A^0 \cdots A^4$.

$A^k[i,j]$

$k \in 0 \cdots n$

~~A' ⇒ based on A°~~

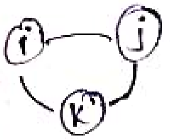
$A^{k}[i,j] = \min \{ A^{k-1}[i,j], A^{k-1}[i,k] + A^{k-1}[k,j] \}$

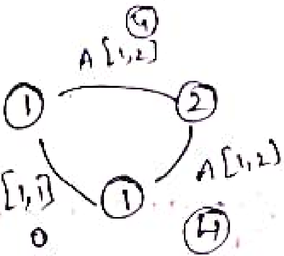
via ↓ source ↓ destination.

A° ⇒ adjacency ~~graph~~ matrix of given graph

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

→ diagonal zero





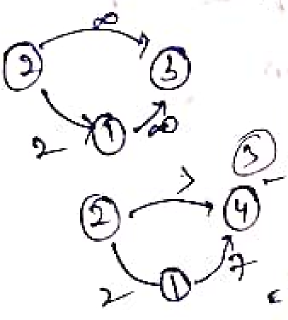
same distance go with previous distance.

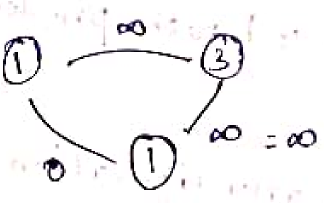
A' → via

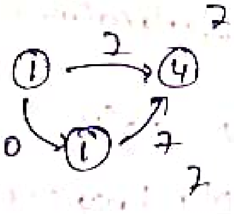
	1	2	3	4
1	0	4	∞	7
2	2	0	∞	>
3	∞	6	0	∞
4	∞	8	5	0

← destination

↓ source





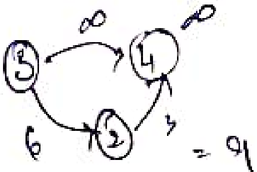


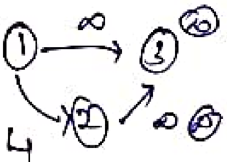
A²

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

The via node value will same weight ∴ directly can write values from previous matrix

$A^{②} \rightarrow$ ∴ 2nd row, 2nd column } Remains same.





A^3

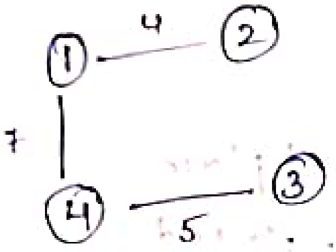
	1	2	3	4
1	0	4	∞	7
2	2	0	∞	3
3	8	6	0	9
4	10	8	5	0

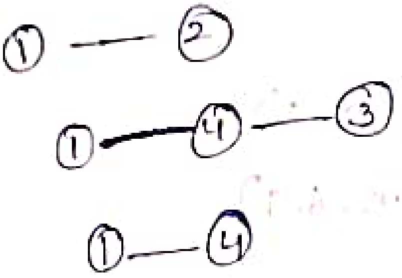
A^4

	1	2	3	4
1	0	4	12	7
2	2	0	8	3
3	8	6	0	9
4	10	8	5	0

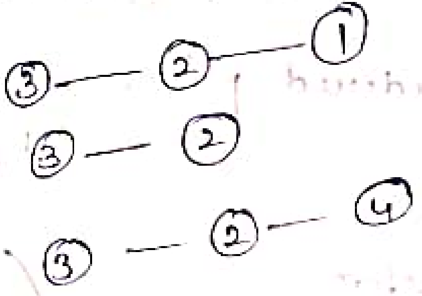
Floyd's algorithm:-

from node ①

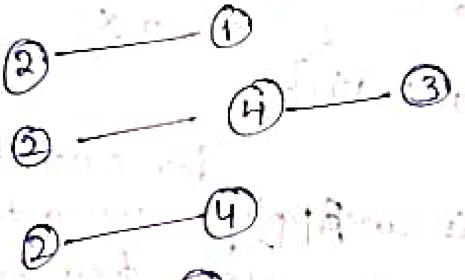




from node ③

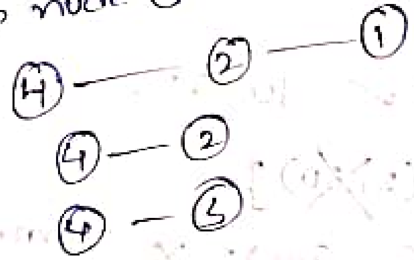


from node ②



12 shortest paths.

from node ④



→ 0/1 Knapsack problem:-

bag → capacity - m

items → n   items into the bag any order one at a time.

items — weight - w

items — profit - P

$x_1 = 1/0/½$

$0 \le x \le 1$ → In fractional knapsack

Here in 0/1 knapsack

we can't place some part of item (Partial)

only fully or notplaced

→ ordered sets → ordered pairs. → (P, ω).

(Contains one(or) more ordered pairs)

↳ Identify the optimal solution.

Ex: Find optimal solution for the 0/1 knapsack following by

$n = 4$ $m = 25$

$(P_1, P_2, P_3, P_4) = (2, 5, 8, 1)$

$(w_1, w_2, w_3, w_4) = (10, 15, 6, 9)$

↓

Sum= 40 indicates that can't place all 4 items: only some are placed

① construct ordered pairs

$S^0 = \{(\overset{P}{0}, \overset{w}{0})\}$

↓

position of bag before inserting items into bag.

$S_1^0 = \{(\overset{\nearrow S^0 + (P_1, w_1)}{2}, 10)\}$

↓

1 item placing in default order

add previous 2 sets

$S^1 = S^0 \cup S_1^0$

$\{(0, 0), (2, 10)\}$

↓

On this we Apply Dominance rules for ordered set obtained from before "2"

② $0 < 2$ ✓ $0 > 10$ ✗

① $0 < 25$ ✓ $10 < 25$ ✓

{If $\{(0,0), (2, \cancel{30})\}$

$0 < 25$ ✓ $30 < 25$ ✗ remove that pair}

Dominance rules

1.) (P, ω) $\omega > m$ then remove that pair

2.) $(P_i, \overset{\times}{w_i})$ · (P_j, w_j) both satisfies 2 rules then remove first pair

$P_i < P_j$ $w_i > w_j$ (less weight more profit).

$S_1^1 = \{(\overset{S^1 + (P_2, w_2)}{5, 15}), (\overset{S^1 + (P_2, w_2)}{7, 25})\}$

$S^2 = S^1 \cup S_1^1$

$= \{(0,0), (2,10), (5,15), (7,25)\}$ → apply dominance rule

① $0 < 25$ $10 < 25$ $15 < 25$ $25 \leq 25$ ✓   ② $0 \not< 2$ $\underset{\times}{0 > 10}$, $2 < 5$ $\underset{\times}{10 > 15}$, $5 < 7$, $\underset{\times}{15 > 25}$

$S_1^2 = \{(8,6), (10,16), (13,21), (15,31)\}$

$S^3 = S^2 \cup S_1^2$

$= \{(0,0), (2,10), (5,15), (7,25), (8,6), (10,16), (13,21), \cancel{(15,31)}\}$

$\downarrow$ w > m
31 > 25
so remove

$S^3 = \{(0,0), (2,10), (5,15), \cancel{(7,25)}, (8,6), (10,16), (13,21)\}$

7<8 ✓ 25>6 ✓ remove (7,25)

5<8 ✓ 15>6 ✓

$S^3 = \{(0,0), (2,10), \cancel{(5,15)}, (8,6), (10,16), (13,21)\}$

Check again 2nd dominance rule until it is not applicable

$S^3 = \{(0,0), \cancel{(2,10)}, (8,6), (10,16), (13,21)\}$

2<8 10>6

$S^3 = \{(0,0), (8,6), (10,16), (13,21)\}$

$S_1^3 = \{(1,9), (9,15), (11,25), (14,30)\}$

$S^4 = S^3 \cup S_1^3$

$S^4 = \{(0,0), (8,6), (10,16), (13,21), (1,9), (9,15), (11,25), \cancel{(14,30)}\}$

$\downarrow$ 30<25 X

13<1 X 21>9 ✓ 1<9

$S^4 = \{(0,0), (8,6), (10,16), \cancel{(13,21)}, (1,9), (9,15), (11,25)\}$

16>9 ✓

$[S^4 = \{(0,0), (8,6), (10,16), (1,9), (9,15), (11,25)\}$

$S^4 = \{(0,0), (8,6), \cancel{(10,16)}, (1,9), (9,15), (11,25)\}]$ X ✓

$S^4 = \{(0,0), (8,6), (1,9), (9,15), (11,25)\}$

$S^4 = \{(0,0), (8,6), (10,16), (13,21), (1,9), (9,15), (11,25)\}$

$S^0 = \{(0,0)\}$ S_1^0

$S_1^1 = \{(0,0), (2,10)\}$ S_1^1

$S_2^2 = \{(0,0), (2,10), (5,15), (7,25)\}$ S_2^2

$= \{(0,0), (8,6), (10,16), (13,21)\}$ S_1^3

$S_3^3 = \{(0,0), (8,6), (10,16), (13,21),$

$S^4 = \{(0,0), (8,6), (10,16), (13,21), (1,9), (9,15, (11,25)\}$

② Selecting form last S^n highest profit. i.e from S^4

$S^4 = \{(0,0), (8,6), (10,16), \boxed{(13,21)}, (1,9), (4,15), (11,25)\}$

↓

If right side then.
i.e item is added to bag

In S^3 left side ordered set
i.e item is not added into the Bag

status of last item

$\boxed{x_4 = 0}$ check in S^3, S_1^3

S^0	S_1^0	
S^1	S_1^1	X
S^2	S_1^2	+
S^3	S_1^3	+
S^4	8	
↓ before adding sets	↓ after adding sets	

(13, 21) check in S^2, S_1^2

$\boxed{x_3 = 1}$ when '1' then remove the profit & wt from that pair

(13, 21) − (8, 6) = (5, 15)

(5, 15) check in S^1, S_1^1

$\boxed{x_2 = 1}$

(5, 15) − (5, 15) = (0, 0) check in S^0, S_1^0

(0, 0)

$\boxed{x_1 = 0}$

optimal Solution $\langle x_1\ x_2\ x_3\ x_4 \rangle$
$\langle 0, 1, 1, 0 \rangle$ out of 4 loaded
2 items
2nd, 3rd

$w = 15 + 6 = 21$
$P = 13$

(13, 21)

→ Matrix chain Multiplication :-

$A_{2\times2}$ $B_{2\times2}$

$A = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$ $B = \begin{bmatrix} 2 & 2 \\ 2 & 2 \end{bmatrix}$

$AB = \begin{bmatrix} 1\times2 + 1\times2 & 1\times2 + 1\times2 \\ 1\times2 + 1\times2 & 1\times2 + 1\times2 \end{bmatrix}$

$8 = 2\times2\times2$

No. of multiplication required to multiply two matrices = No. of rows in 1st matrix × no. of columns in 1st matrix × no. of colums in 2nd matrix

$A_{3\times4}$ $B_{4\times2}$

$3\times4\times2$

- Multiply more than 2 matrices.

$A_{3\times2}$ $B_{2\times4}$

AB

- A B C D E

⑭ ways

$A_{3\times2}$ $B_{2\times4}$ $C_{4\times3}$

1.) $\underset{3\times4}{(AB)}\ \underset{4\times3}{C}$ ⇒ 3×3

2.) $A\cdot(BC)$ ⇒ 3×3

$A_{3\times2}$ $B_{2\times4}$ $C_{4\times3}$ $D_{3\times4}$

1.) ((AB)(CD))

2.) (A (BC)) D

3.) A((BC) D)

4.) A(B(C D))

5.) ((AB)C) D

$$\boxed{P(n) = \sum_{K=1}^{n-1} P(K)\,P(n-K)}$$

$P(1) = 1$

P(n) ↓ no. of ways to multiple 'n' no. of matrices

- $\underline{A_{3\times2}\ B_{2\times4}}_{\,3\times4}\ \overline{B_{2\times4}\ C_{4\times3}}^{\,2\times3}$ → I/P

1.) $(AB)C = 3\times2\times4 + 3\times4\times3 = 24 + 36 = 60$

2.) $A(BC) = 2\times4\times3 + 3\times2\times3 = 24 + 18 = \boxed{42}$

} → feasible solution multiplication, operations required

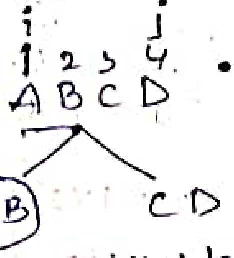
42 ↓ optimal solution.

2nd is best as less operations are required

$M[i, j]$
↓
no. of multiply operations required to multiply i with j

$\overset{1}{A}_{3\times2}$ $\overset{2}{B}_{2\times4}$ $\overset{3}{C}_{4\times3}$ $\overset{4}{D}_{3\times5}$

size → $d_0 \times d_1$ $d_1 \times d_2$ $d_2 \times d_3$ $d_3 \times d_4$



• divide into 2 parts with min one matrix

i to K

K+1 to j

$$M[i, j] = \min\{M[i, K] + M[K+1, j] + d_{i-1} d_K d_j\}$$

1 2 $d_{i-1} \times d_K$

3 4 $d_{K} \times d_j$

$d_0 \times d_2$

$d_2 \times d_4$.

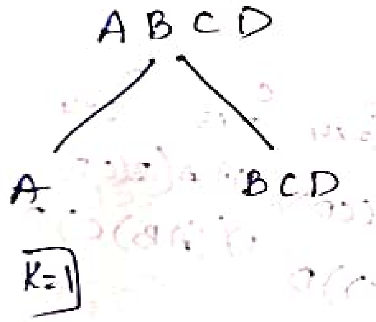
$\boxed{K = 2}$

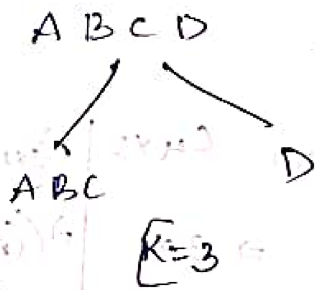
$d_0 \times d_2 \times d_4$

$i=1, j=4$

$$M[i, j] = \min_{K = i \text{ to } j-1}\{M[i, K] + M[K+1, j] + d_{i-1} d_K d_j\}$$

$K = i$ to $j-1$





$$M[i, j] = \min\{M[i, K] + M[K+1, j] + d_{i-1} d_K d_j\}$$

$K = i$ to $j-1$ $i=1$ $j=4$

Ex.) Find optimal order for multiply following matrix.

$\overset{1}{A}_{5\times6}$ $\overset{2}{B}_{6\times4}$ $\overset{3}{C}_{4\times2}$ $\overset{4}{D}_{2\times3}$

$d_0 \times d_1$ $d_1 \times d_2$ $d_2 \times d_3$ $d_3 \times d_4$

M Table, $n \times n$ = no. of matrices to be multiplied

	1	2	3	4
1	0	120	108	138
2	X	0	48	84
3	X	X	0	24
4	X	X	X	0

① 1-2, 2-3, 3-4
② 1-3, 2-4
③ 1-4

K Table

	1	2	3	4
1	0	1	1	3
2	X	0	2	3
3	X	X	0	3
4	X	X	X	0

upper triangle values are calculated.

138 => Min no. of multiplication operations.

$M[\underset{i}{1}, \underset{j}{2}] = \min\{M[1,1] + M[2,2] + 5 \times 6 \times 4\} = \min\{0+0+120\}$
$= 120$

K = 1

$M[2,3] = \min\{M[2,2] + M[3,3] + 6 \times 4 \times 2\} = \min\{0+0+48\}$
$= 48$

K = 2

$M[3,4] = \min\{M[3,3] + M[4,4] + 4 \times 2 \times 3\} = \min\{0+0+24\}$
$= 24$

K = 3

$M[1,3] = \min\{M[1,1] + M[2,3] + 5 \times 6 \times 2,$
$M[1,2] + M[3,3] + 5 \times 4 \times 2\}$

K = 1,2 (1 to 2)

$= \min\{0 + 48 + 60, 120 + 0 + 40\}$

$= \min\{108, 160\} = 108$

$M[2,4] = \min\{M[2,2] + M[3,4] + 6 \times 4 \times 3, M[2,3] + M[4,4] + 6 \times 2 \times 3\}$

K = 2,3

$= \min\{0 + 24 + 72, 48 + 0 + 36\}$

$= \min\{96, 84\} = 84$

$M[1,4] = \min\{M[1,1] + M[2,4] + 5 \times 6 \times 3, M[1,2] + M[3,4] + 5 \times 4 \times 3,$
$M[1,3] + M[4,4] + 5 \times 2 \times 3\}$

K = 1, 2, 3

$= \min\{0 + 84 + 90, 120 + 24 + 60, 108 + 0 + 30\}$

$= \min\{174, 204, 138\} = 138$

optimal order

(A)(B C)(D) K [1, 4] = 3
↓ 1st part have 3 matrices

K[1,3]

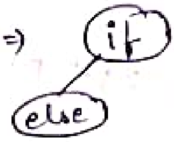
(A(B C))(D)

→ **Optimal Binary Search Tree (OBST):-**

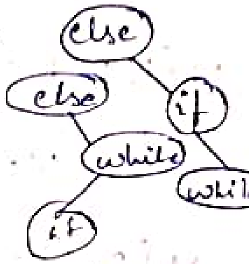
{else} one value.
└→ one node in BST

{else, if}

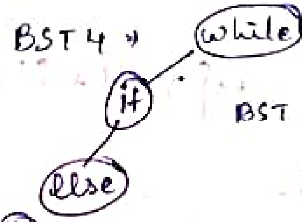
BST1 ⇒

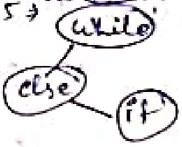

BST2 ⇒


5 possibilities
[else, if, while]

BST1 ⇒
BST2 ⇒


BST3 ⇒


BST4 ⇒


BST 5 ⇒


→ No. of BST?

n- Keywords

$$\boxed{BSTs \Rightarrow \frac{2n_{C_n}}{n+1}}$$

for $n=1 \Rightarrow \frac{2c_1}{1+1} = \frac{\cancel{2}}{\cancel{2}} = ①$

$n=2 \Rightarrow \frac{4C_2}{3} = \frac{\frac{4\times 3}{2\times 1}}{3} = ②$

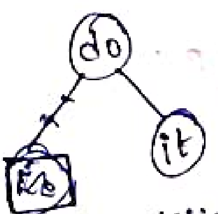
$\frac{6C_3}{4} = \frac{\cancel{6}\times 5\times 4}{\cancel{4}\times 3\times 2\times 1}$ ⑤

$n=4 \Rightarrow$ ⑭

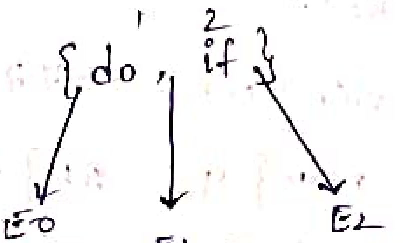
→ **Cost of BST:-** can search any no. of times.

Cost of BST = cost of $\underbrace{\text{Succesful search}}$ + cost of unsuccesful Search.
↓
when we search for a Keyword present in BST

{do, it} (1, 2)

BST 1


BST 2

Key words before "do"	b/w do & if	after if
break	else	while
continue	for	switch

frequencies:-

p(1) = 2 No. of times that keyword is searched

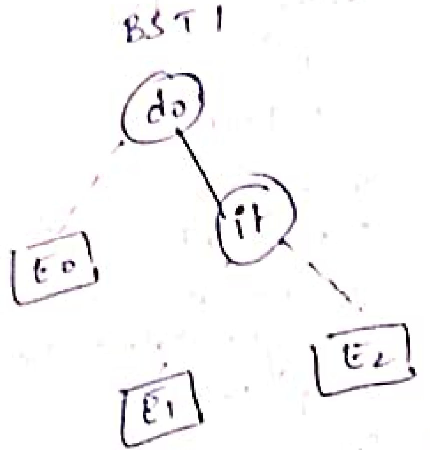
p(2) = 1

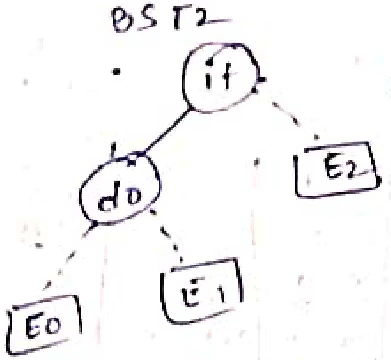
internal keywords

q(0) = 2 (for E0)

q(1) = 3

q(2) = 1 external keywords.





Cost = 2*2 + 1*1 + 2*2 +
2*3 + 1*1
= 4+1+4+6+1 = ⑯

external Keywords: E0, E1, E2
which are not present

do, if → Internal Keywords
which are present

$$\text{Cost} = \overbrace{1*2 + 2*1}^{\text{succesful search}} + \overbrace{2*2 + 2*3 + 2*1}^{\text{unsuccesful search}} = \boxed{14}$$ optimal BST

1 ↓ time to search do is 1

2 ↓ no. of times for do

2 ↓ compare with do not then left no left child ∴ one comparision

14 ↓ Best BST as len cost

By Dynamic programming method:-

→ By Dynamic programming for following

Ex: Identify optimal BST for following

n = 4

{do, if, int, while} In alphabetical order

do ↙ E0, ↓ E1, ↓ E2, ↓ E3, ↘ E4

P(1) = 3 q(0) = 2
P(2) = 3 q(1) = 3
P(3) = 3 q(2) = 1
P(4) = 1 q(3) = 1
q(4) = 1

n = 4
n+1 = 5 E

$$\overset{\text{cost of BST}}{c(i,j)} = \min_{k=i+1 \text{ to } j} \{c(i,k-1) + c(k,j)\} + w(i,j)$$

w(i,j) = P(j) + q(j) + w(i, j-1)

r(i,j) = k which gives minimum value for c(i,j)

	0	1	2	3	4
0	$w_{00}=2$ $c_{00}=0$ $r_{00}=0$	$w_{11}=3$ $c_{11}=0$ $r_{11}=0$	$w_{22}=1$ $c_{22}=0$ $r_{22}=0$	$w_{33}=1$ $c_{33}=0$ $r_{33}=0$	$w_{44}=1$ $c_{44}=0$ $r_{44}=0$
1	$w_{01}=8$ $c_{01}=8$ $r_{01}=1$	$w_{12}=7$ $c_{12}=7$ $r_{12}=2$	$w_{23}=3$ $c_{23}=3$ $r_{23}=3$	$w_{34}=3$ $c_{34}=3$ $r_{34}=4$	
2	$w_{02}=12$ $c_{02}=19$ $r_{02}=1$	$w_{13}=9$ $c_{13}=12$ $r_{13}=2$	$w_{24}=5$ $c_{24}=8$ $r_{24}=3$		
3	$w_{03}=19$ $c_{03}=25$ $r_{03}=2$	$w_{14}=11$ $c_{14}=19$ $r_{14}=2$			
4	$w_{04}=16$ $\boxed{c_{04}=32}$ $\boxed{r_{04}=2}$				

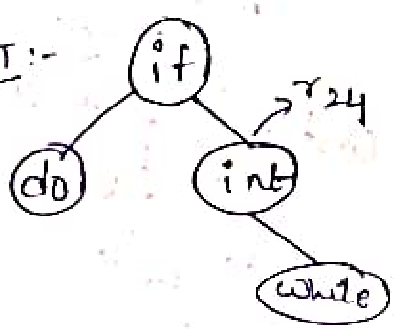
4 key words
means
0 to 4

In row '0'
take c & r as "0"
~~in row 'o'~~
weight values are
'q' values.
from row '1'
use formulas.

$$w(0,1) = P(1) + q(1) + w(0,0)$$
$$= 3+3+2 = 8$$
$$c(0,1) = \min\{c(0,0)+c(1,1)\} + w(0,1)$$
$$= \{0+0\}+8$$
$x = 1$

$r(i,j) = 1$
$r(0,1) = 1$

OBST :-



$\boxed{\text{Cost}=32}$

Ex: {do, if, int}

E0 E1 E2 E3

$P(1)=3,\ P(2)=3,\ P(3)=1$

$q(0)=2,\ q(1)=3,\ q(2)=1,\ q(3)=1$

w = weight
c = cost
r = root

	0	1	2	3
0	$w_{00}=2$ $c_{00}=0$ $r_{00}=0$	$w_{11}=3$ $c_{11}=0$ $r_{11}=0$	$w_{22}=1$ $c_{22}=0$ $r_{22}=0$	$w_{33}=1$ $c_{33}=0$ $r_{33}=0$
1	$w_{01}=8$ $c_{01}=8$ $r_{01}=1$	$w_{12}=7$ $c_{12}=7$ $r_{12}=2$	$w_{23}=3$ $c_{23}=3$ $r_{23}=3$	
2	$w_{02}=12$ $c_{02}=19$ $r_{02}=1$	$w_{13}=9$ $c_{13}=12$ $r_{13}=2$		
3	$w_{03}=14$ $c_{03}=25$ $r_{03}=2$			

→ combination of i, j ~~are~~ diff is 1

$w(0,1) = p(1) + q(1) + w(0,0)$

$= 3 + 3 + 2 = 8$

$c(0,1) = \min_{k=1} \{c(0,0) + c(1,1)\} + w(0,1)$

$= 0 + 0 + 8 = 8$

$w(i,j) = p(j) + q(j) + w(i, j-1)$

$c(i,j) = \min \{c(i, k-1) + c(k,j)\} + w(i,j)$

$k = i+1$ to j

$r(i,j) = k$

$w(1,2) = p(2) + q(2) + w(1,1)$

$= 3 + 1 + 3 = 7$

$c(1,2) = \min \{c(1,1) + c(2,2)\} + w(1,2)$

$k = 2$ to 2

$k = 2$

$= 0 + 0 + 7 = 7$

$r_{12} = 2$

$w(2,3) = p(3) + q(3) + w(2,2)$

$= 1 + 1 + 1 = 3$

$c(2,3) = \min \{c(2,2) + c(3,3)\} + w(2,3)$

$k = 3$ to 3

$= 0 + 0 + 3 = 3$

$r_{23} = 3$

$w(0,2) = p(2) + q(2) + w(0,1)$

$= 3 + 1 + 8 = 12$

$c(0,2) = \min \{c(0,0) + c(1,2)\} + w(0,2);$

$\{c(0,1) + c(2,2)\} + w(0,2)$

$k = 1$ to 2

$k = 1, 2$

$= \min\{0 + 7 + 12, 8 + 0 + 12\}$

$= \boxed{19}, 20$

$\hookrightarrow$ min

$r_{02} = 1$

$w(1,3) = p(3) + q(3) + w(1,2) = 1 + 1 + 7 = 9$

$c(1,3) = \min \{c(1,1) + c(2,3), c(1,2) + c(3,3)\} + w(1,3)$

$k = 2, 3$

$= \min\{0 + 3, 7 + 0\} + 9 = \min\{3, 7\} + 9 = 3 + 9 = 12$

$\boxed{k = 2}$

$w(0,3) = p(3) + q(3) + w(0,2) = 1 + 1 + 12 = 14$

$c(0,3) = k = 1, 2, 3 \Rightarrow 25$

$\boxed{k = 2}$

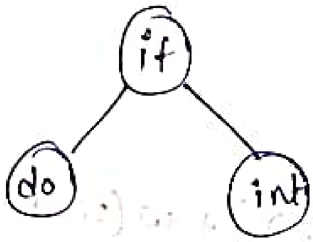
OBST :-

The cost value in the entry of last row indicates the least cost i.e c_{03} = 25

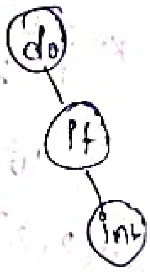
In given example $\overset{1}{\text{do}}$, $\overset{2}{\text{if}}$, $\overset{3}{\text{int}}$

r_{03} = ② second key word will be the root

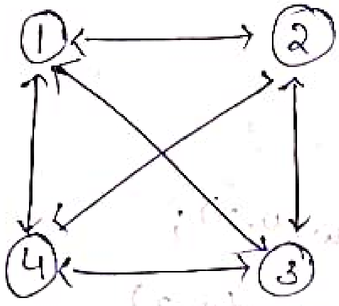
if r_{03} = 1



∴ $\boxed{\text{cost} = 25}$



→ Travelling Salesman Problem :-



	1	2	3	4
1	0	10	15	20
2	5	0	9	10
3	6	13	0.	12
4	8	8	9	0

Salesman has to start's from one node & visit every node and again to starting node.

1.) 1→2→3→4→1 cost = 10+9+12+8 = 39

Rule :-
has to visit all nodes but only once

2.) 1→2→4→3→1 cost = 10+10+9+6 = ㉟ → optimal tour less cost.
3.) 1→3→2→4→1 cost = 15+13+10+8 = 46
4.) 1→3→4→2→1 cost = 15+12+8+5 = 40
5.) 1→4→2→3→1 cost = 20+8+9+6 = 43
6.) 1→4→3→2→1 cost = 20+9+13+5 = 47

$$g(i, S) = \underset{J \in S}{\min}\{C_{ij} + g(j, S-\{j\})\}$$

cost of path
node from which starts
set of nodes

Step1:-

$|S| = 0$

$i = 2$ to 4 → Is starting from node 'i' without covering any nodes going to starting.

$$g(2, \phi) = \min\{C_{21}\} = \min\{5\} = 5$$

$$g(3, \phi) = \min\{C_{31}\} = \min\{6\} = 6$$

$$g(3, \phi) = \min\{C_{41}\} = \min\{8\} = 8$$

Step2:-

$|S| = 1$

$i = 2$ to 4 → node 2 to 3 → reaching nodes again to node 1

$$g(2, \{3\}) = \min\{C_{23} + g(3, \phi)\} = \min\{9+6\} = \min\{15\} = 15$$

$$g(2, \{4\}) = \min\{C_{24} + g(4, \phi)\} = \min\{10+8\} = \min\{18\} = 18$$

$$g(3, \{2\}) = \min\{C_{32} + g(2, \phi)\} = \min\{13+5\} = \min\{18\} = 18$$

$$g(3, \{4\}) = \min\{C_{34} + g(4, \phi)\} = \min\{12+8\} = \min\{20\} = 20$$

$$g(4, \{2\}) = \min\{C_{42} + g(2, \phi)\} = \min\{8+5\} = \min\{13\} = 13$$

$$g(4, \{3\}) = \min\{C_{43} + g(3, \phi)\} = \min\{9+6\} = \min\{15\} = 15$$

Step3:-

$|S| = 2$

$i = 2$ to 4

$$g(2, \{3, 4\}) = \min\{C_{23} + g(3, \{4\}), \underline{C_{24} + g(4, \{3\})}\}$$

$$= \min\{9+20, 10+15\} = \min\{29, 25\} = 25$$

$$g(3, \{2, 4\}) = \min\{C_{32} + g(2, \{4\}), \underline{C_{34} + g(4\{2\})}\} =$$

$$= \min\{13+18, 12+13\} = \min\{31, 25\} = 25$$

$$g(4, \{2, 3\}) = \underline{\min\{C_{42} + g(2, \{3\})}, C_{43} + g(3, \{2\})\}$$

$$= \min\{8+15, 9+18\} = \min\{23, 27\} = 23$$

Step4:-

$|S| = 3$

$$g(1, \{2,3,4\}) = \underbrace{\min\{c_{12} + g(2, \{3,4\})}, c_{13} + g(3, \{2,4\}), c_{14} + g(4, \{2,3\})\}$$

$$= \min\{10+25, 15+25, 20+23\}$$

$$= \min\{35, 40, 43\} = 35$$

optimal cost = 35

The optimal tour = 1 → 2 → 4 → 3 → 1

→ Longest Common Subsequence (LCS) problem:-

A subsequence of a string S is a string which is obtained by removing zero or more characters from the string S, without changing the relative order of remaining character in strings.

String S_1 = "abc"

subsequence = "ab", "a", "b", " ". (4 subsequences)

String S_2 = "abc"

subsequence = "abc", "bc", "ac", "ab", "c", "b", "a", " "

(8 subsequences)

$n \rightarrow 2^n$ subsequences

$3 \rightarrow 2^3 = 8$

LCS problem:-

Given 2 strings S_1 & S_2, we have to find the longest common subsequence b/w S_1 & S_2

S_1 = "ab"

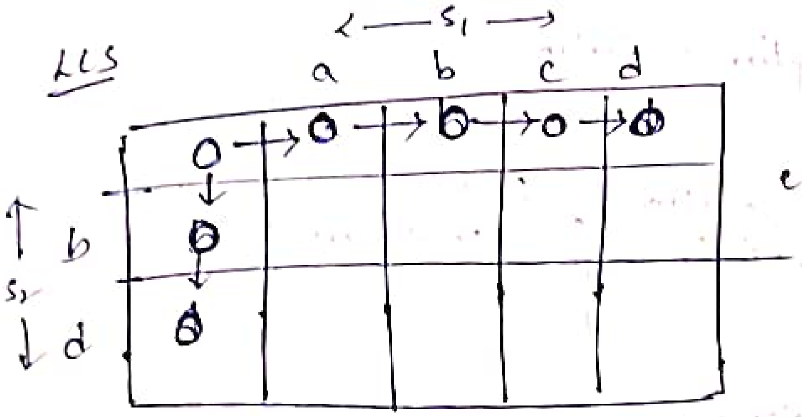
S_2 = "abc"

Common subsequences b/w S_1 & S_2 are: "ab", "a", "b", " "

longest " is "ab"

Ex: S_1 = "abcd"
S_2 = "bd"

no. of strings + 1 ⇒ are no. of rows & columns.



...if $(S_1[i] == S_2[j])$

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

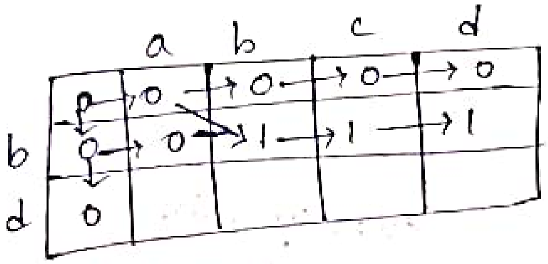
else

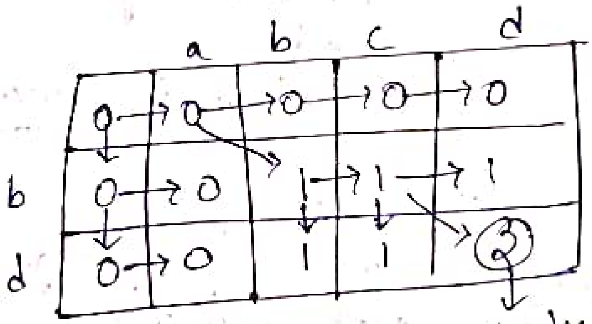
$LCS[i,j] = \max\{LCS[i-1,j], LCS[i,j-1]\}$

5 colums
3 rows

values match
takes diagonal entry.

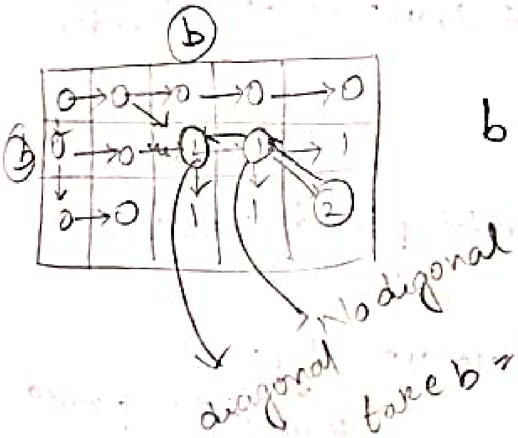
$\max\{LCS[0](0)\} = 0$





Ab same diagonal
'1' add 1 = ②

indicates length of longest common subsequence.



bd

if top ~~of~~ ~~or~~
backtrack only the character having diagonal then take that value

→ Module- 4

Iterative Improvement

P- no. of solutions — best / optimal solution

Ⓟ- basic solution

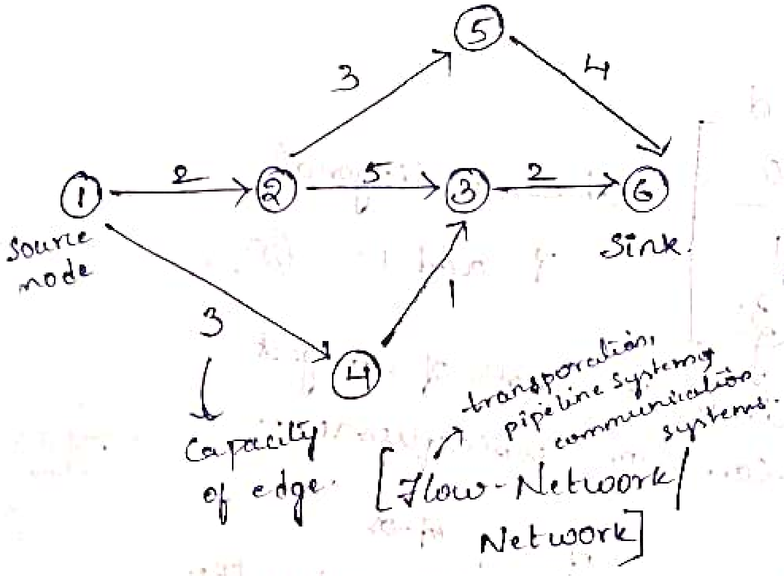
↓ Improve basic solution with (Iterations) } Iterative Improvement

optimal solution.

1) Linear programming problem
2) Maximum flow problem.
3) Maximum Matching in Bipartite graphs problem } cat-2
4) Stable Marriage problem.

→ Maximum flow problem:-

Input:- Weighted directed graph.



exactly with no incoming edge. that node is source node.

exactly having no outgoing edge that node is sink

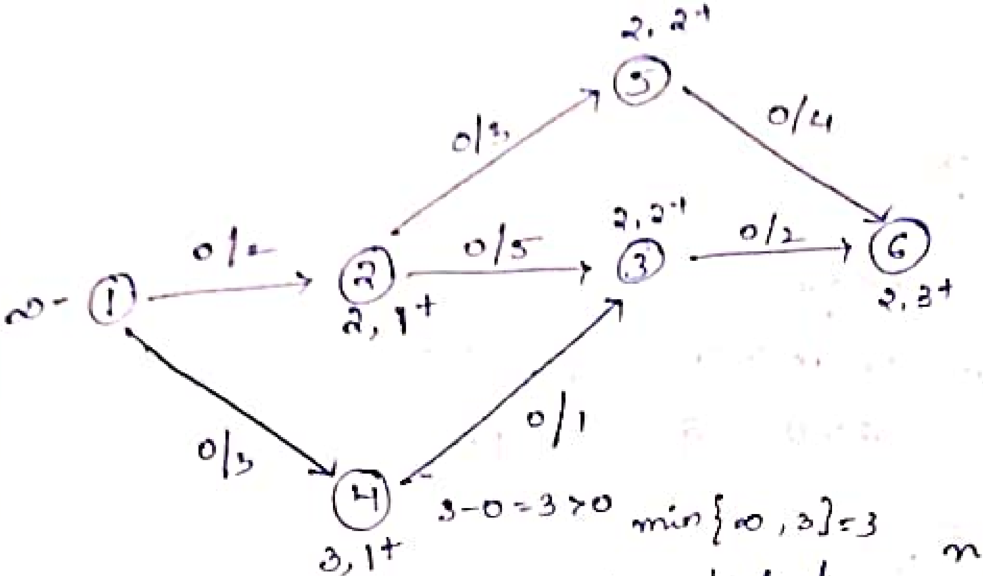
2, 3, 4, 5 → Intermediate nodes.

$1 \xrightarrow{2} 5 \xrightarrow{3} 5 \xrightarrow{4} 6$ ⇒ min value is 2

∴ max amt of material passed is 2 units.

→ Ford-Fulkerson algorithm:-

- For each ~~node~~ ~~node~~ edge capacity add flow value 0/2 ↓ 0 units are passed
- Give ∞- to the source node
- Take Q and store starting node.
- Consider the node which are connected by forward edge which is going to be deleted.
- The next node having label or not should be checked.
- Consider backward edges. • Checking sink node is labeled or not.



3-0=3>0 min{∞, 3}=3

node deleted : 1

Q: 1 2 4 3 5 6 for nodes label
↑ ↑ ↑

If no label assign

cal= capacity - flow

2 - 0 = 2 > 0 should be greater than ~~flow~~ 0.

min{∞, 2}

label of node to this it is connected

2, ↳ node from which it is coming.

1⊕, forward edge 1⊖, backward edge.

node deleted : 2

for node 3 label

5-0= 5 > 0

min{2, 5} = 2

2, 2+

node 5 label

3-0 = 3 > 0

min{2, 3} = 2

2, 2+

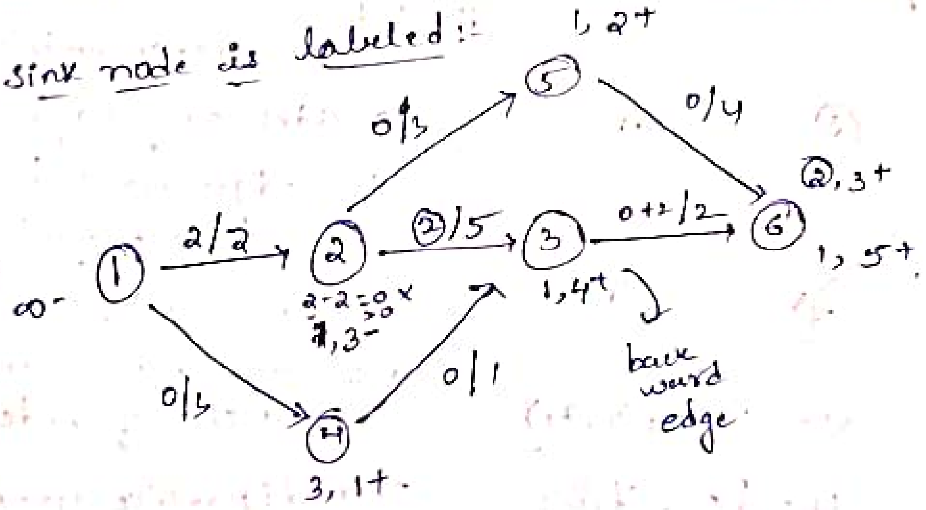
→ forward edge

Backward edge:-

node 1 is already having label.

no need to give label

If sink node is labeled::



Back ~~path~~ trace through the path it came

add the 1st label of sink node to the flow value

$Q : 1\ 4\ \underset{\uparrow}{3}\ 2\ 5\ 6$

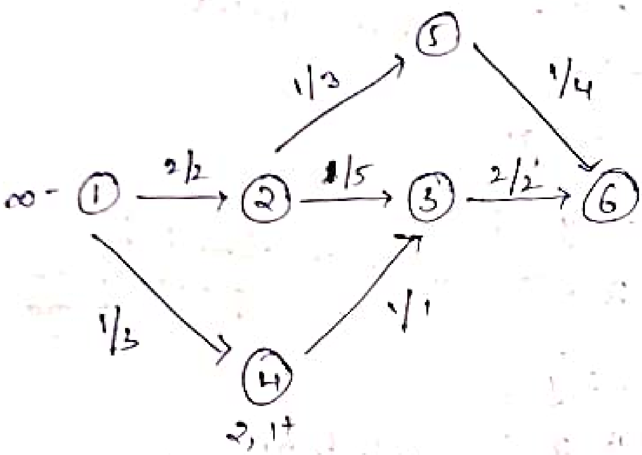
node 3 has backward edge.

2 is not having label

to label for backward edge

consider flow value > 0 $\quad$ min {1, 2}

$2 > 0 = 2$ $\quad$ 1,3⁻



for backward edge subtract the flow value.

$Q : \underset{\uparrow}{1}\ \underset{\uparrow}{4}$ The Queue is empty.

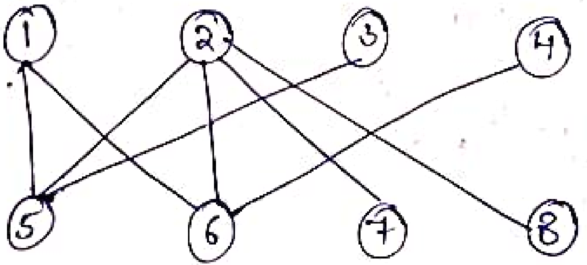
stop the process

Maximum flow = 1+2=3

consider all incoming edges to the sink node add those ~~sink~~ flow values to get maximum flow.

→ Maximum Matching in Bipartite Graph:-

1) Bipartite Graphs: It is a graph in which nodes are partitioned into 2 ~~desired~~ disjoint sets. (no common node)



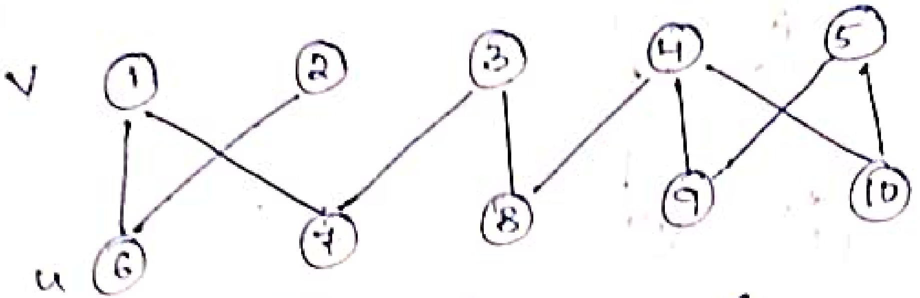
Consider any edge the edges will be connected node in 2 sets (1-5, 5-2)

No edge b/w same set of nodes

∴ Then bipartite Graph.

V = {1, 2, 3, 4}

U = {5, 6, 7, 8}



v = {1, 2, 3, 4, 5}

u = {6, 7, 8, 9, 10}

∴ Bipartite graph

1 — 6

6 — 2 ✓

No edge b/w

V & also

u ✓

Matching in a Bipartite Graph :-

M = {(1, 5), (2, 6)}

↳ ∴ Is a Matching as they not having same node

M ↓ Subset of edges in the graph such that no 2 edges will share the same ~~edge~~ node.

M1 = {(1, 5), (2, 6)} M2 = {(1, 6), (2, 5)} M3 {(1, 5), (2, 7)}

M4 {(1, 5), (2, 8)} M5 → {(1, 5), (4, 6)}

Maximum matching in a bipartite graph:-

largest no. of edges.

M1 = {(1, 5), (2, 6)}

M2 = {(1, 6), (2, 5)}

M3 = {(1, 6), (2, 7), (3, 5)}

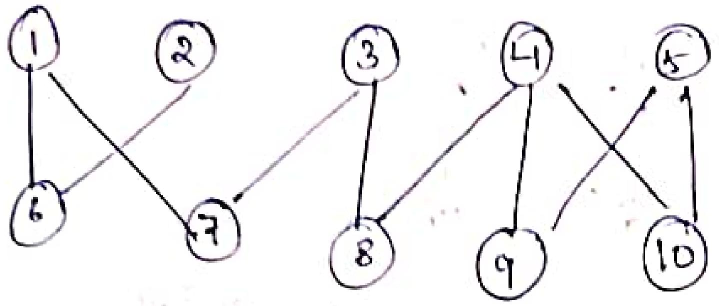
M4 = {(1, 6), (2, 8), (3, 5)}

M5 = {(1, 5), (2, 8), (4, 6)}

M6 = {(1, 5), (2, 7), (4, 6)}

⟶ maximum 3 edges.

Finding any one max matching in a bipartite graph.

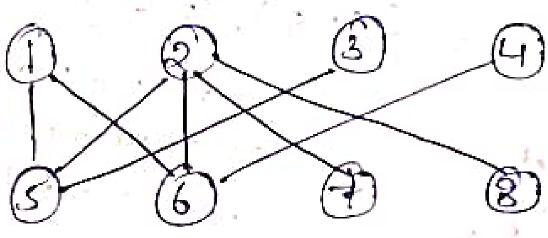


M_1 = {(1,7), (2,6), (3,8), (4,9), (5,10)} → Max 5 edges

M_2 = {(1,7), (2,6), (3,8), (4,10), (5,9)}

→ By iterative improvement method :-

Augmenting path for a matching M



M = {(1,5)}

find augmenting path of M :

Start from any free node in V   end with free node in u

V = {1, 2, 3, 4}

u = {5, 6, 7, 8}

edges are ~~not~~ alternatively $\underset{①}{\text{not in M}}$ & $\underset{②}{\text{in M}}$

$\underset{V}{2}$, 5 , 1 $\underset{u}{6}$

$\underbrace{2 \to 5}_{\text{not in M}} \underbrace{\to 1}_{\text{in M}} \underbrace{\to 6}_{\text{not in M}}$

M: {(2,5)}

Augmenting path of M : $\underset{V}{3}$, 5, 2, $\underset{u}{7}$

M = {(1, 6)}

Augmenting path of M:

$\underset{V}{2}$, 6, 1, $\underset{u}{5}$

M: {(1,6), (2,5)}

Augmenting path of M : $\underset{V}{3}$, 5, 2, $\underset{u}{7}$

Augmenting a matching with an augmenting path:-

M: {(1,5)}

Augmenting path of M : (2, 5, 1, 6)

1 ① 2 ② 3 ③ 4 → odd, even, odd.

edges in even no. & edges in odd no.

add the odd no. edge

delete the even no. edge

M: {~~(1,5)~~, (2,5), (1,6)}

M: {(2,5), (1,6)}

M= {(2, 6)}

Augmenting path of M : (1, 6, 2, 5)

M= {~~(2,6)~~, (1,6), (2,5)}

M = {(1,6), (2,5)}

start with empty matching.

M = { }

Augmenting path : 1, 5 (8 are possible)

$\underset{v}{1}$, $\underset{u}{5}$

M = {(1, 5)} odd position so add

Augmenting path : 2, 5, 1, 6

M= {~~(1,5)~~, (2, 5), (1, 6)}

M= {(2,5), (1, 6)} Augmenting path : 3, 5, 2, 7

{~~(1,5)~~, (1,6), (3,5), (2,7)}

M= {(1, 6), (3, 5), (2, 7)} Augmenting path : 4 6 2 8 → not in match ✗ Not Valid

↓

maximum matching bipartite graph.