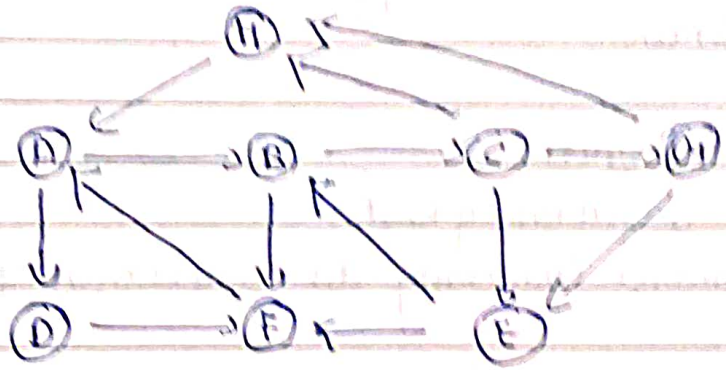
16/02/2024

ALGORITHM - DESIGN :-



Q1 write adjancency list of the above graph.

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

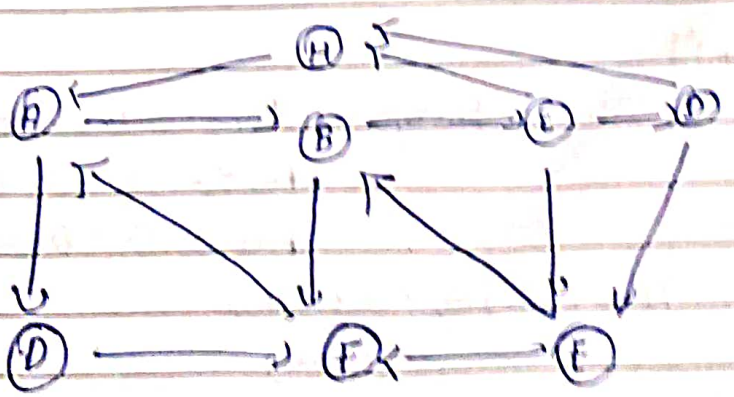
Q2 How to traverse or search the graph?

In two ways a) DFS → By stack
b) BFS → By queue.

DFS :-

DFS (G, v) where G → graph
v → Vertices (starting)

stack → of = { } → empty.



Loop while Stack mat empty:-

1) POP from Stack.
2) Print it.
3) Push all the Vertex from A's adjacency list

"An element can only be traversed for one time only"

Pseudo Code:-

```
DFS (G, V)
Stack S = {}
For each Vertex 'v' set visited [v] = False
    Push S, v;
while S is mat empty do;
        u = POP S;
    if (! visited [u]) then.
then (u) = true;
    Print u;
For each visited neighbour w of u:
        Push S, w;
        end if;
        end while;
        end DFS;
```

Bytes = 8bit
1 Kilobyte = 1024B
megabyte = 1024 KB
Gigabyte -> 1024 MB.
Tera byte -> 1024 GB
CPU -> 3.02 GHz -> 3.02×10^9 /Sec

Complexity.
-> Time
-> Space.

Time Complexity [T(n)]

Bubble Sort:-

$T(n) = cn^2$

Always
Pass = 0 -> O(n)
Print n = n (n)
Print 0 = 0(n).

$T(n) = 2n^2 + 3n + 5$
$= O(n^2)$

Recursive ways!-

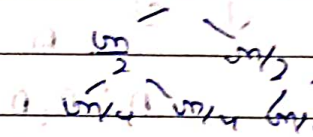
$T(n) = 2T(\frac{n}{2}) + n$ $\quad T(n/2) = 2T(\frac{n}{2}) + \frac{n}{2}$.

$T(n) = 4T(\frac{n}{4}) + n + n = 4T(\frac{n}{4}) + 2n$.

$T(n) = 2^k T(\frac{n}{2^k}) + kn$ $\quad \frac{n}{2^k} = 1$ $\quad n = 2^k$

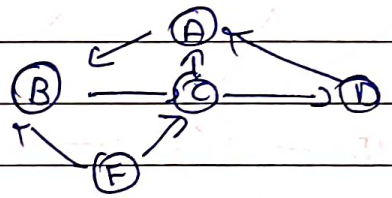
$n = k \log_2$.

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



Graph:-

$G = (V, E)$



Adjacency matrix:-

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

Adjancy list:

A : B

B : C

C : A, D

D : A

E : B, C

Indegree of vertex:- Edge towards vertex

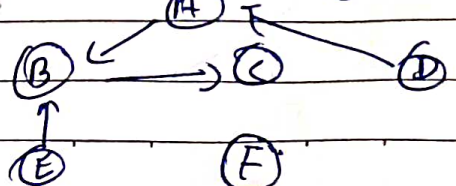
outdegree of vertex:- Edges going from vertex

- A graph that has one or more cycle is a cyclic graph.

Cycle :- It is path that start and end at same vertex and includes atleast one or other vertex.

- A tree is a graph with out any cycle.

A acyclic connected graph can be a tree.

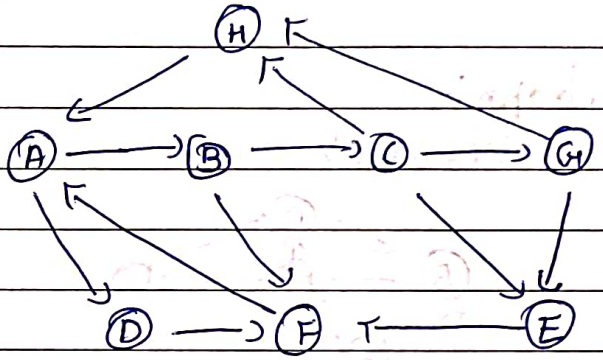


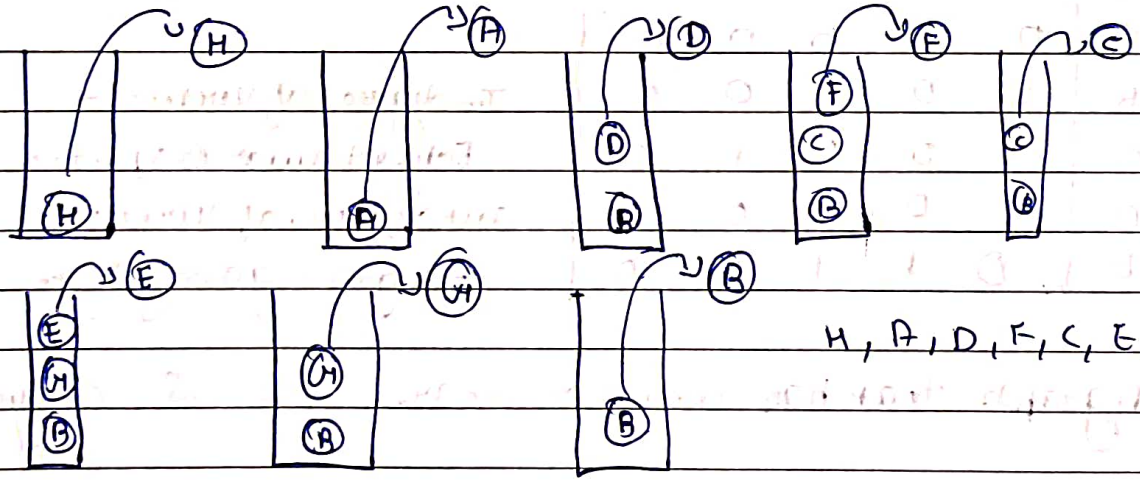
Complete graph :-
Start from one - edge and reach to every edge.

-> Directed Complete graph = $n(n-1)$
-> undirected Complete graph = $\frac{n(n-1)}{2}$.

Graphs :- BFS / DFS.
Adjacency list

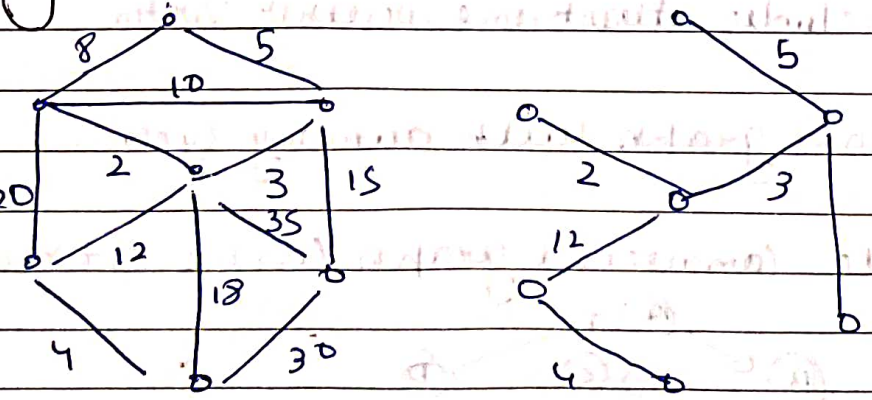
H: A
A: B, D.
B: C, F, G
C: H, F, E
G: F, H
D: F
F: A
E: F





H, A, D, F, C, E, G, B.

Spanning Tree.



BFS.

Code :-

```
import java.util.*;
class graph {
    Private int vm_v;
    Private list<Integer>[] vm_adj;
    Graph(int v) {
        vm_v = v;
        vm_adj = new LinkedList[v];
        For (int i=0; i<v; ++i)
            vm_adj[i] = new;
        LinkedList<Integer>();
    }
    Public void addEdge(int v, int u){
        vm_adj[v].add(u);
    }
    Public void BFS(int s){
        LinkedList<Integer> Q = new LinkedList();
        boolean[] visited = new boolean[vm_v];
        Q.add(s);
        visited[s] = true;
        while(!Q.isEmpty()){
            int v = Q.pop();
            System.out.println(v+" ");
            For(int u: vm_adj[v]){
                if(!visited[v]){
                    Q.add(v);
                    visited[v] = true;
                }
            }
        }
    }
    Public void main(String[]args){
        Graph G = new Graph(5);
        G.addEdge(0,1);
        G.BFS(0);
    }
}
```

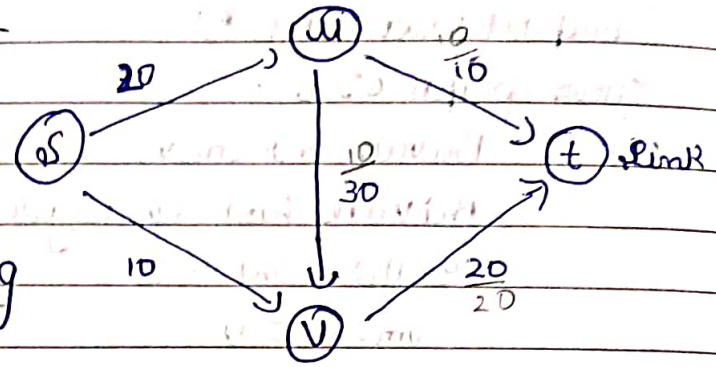
26/02/2024

CH-1

NETWORK FLOW

Network Flow :-

The maximum flow Problem :-



A flow network is a directed graph $G = (E, V)$ with following features

1) Associated with each edge e is a capacity, which is a non-negative integer it is denoted by capacity of edge (e.

2) There is a single source node (s)

3) There is a single sink node (t)

4) other nodes except s and t are known as internal nodes.

Assumptions :-

1) NO edges enter the source s and no edges leaves the sink t

2) There is atleast one edge incident to each node.

3) All capacities are non-negative integers.

Flow definition :-

we say that an s-t flow is a function f that maps each edge to a non-negative real number.

$f : E \to R^+$

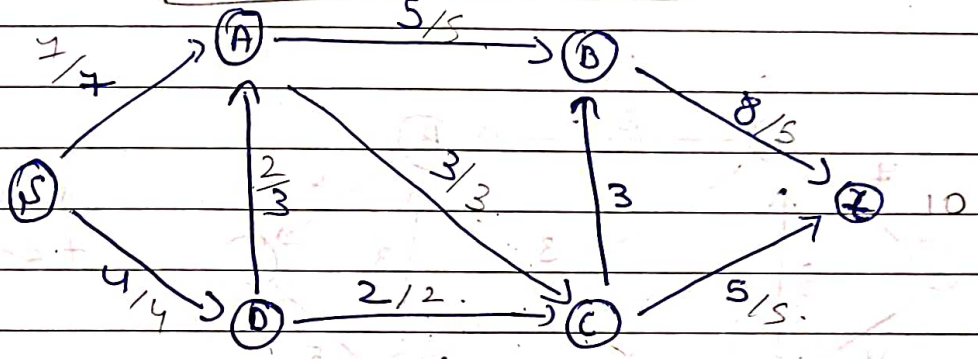
The value $f(e)$ intuitively represents the amount of flow carried by the edge e.

A flow f must satisfy the following:-

1) (Capacity Condition) for each edge $e \in E$ we have $0 \le f(e) \le c_e$

2) (Conservation Condition) For each node $v \in V$ other than s and t we have

$$\sum_{e \text{ into } v} f(e) = \sum_{e \text{ out of } v} f(e)$$



The residual graph

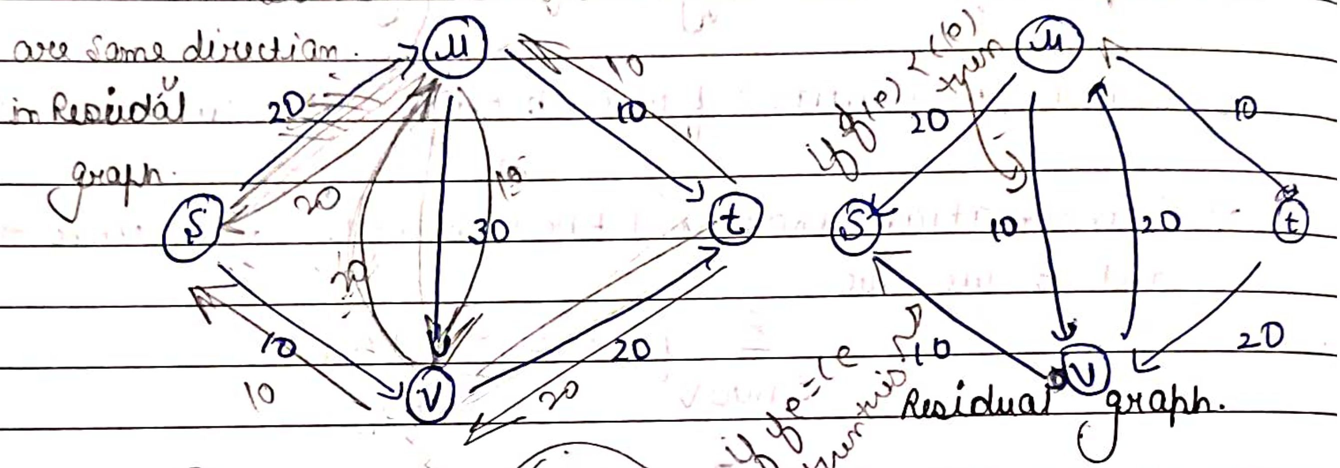
To provide a systematic way to search for forward backward operations we define the residual graph.

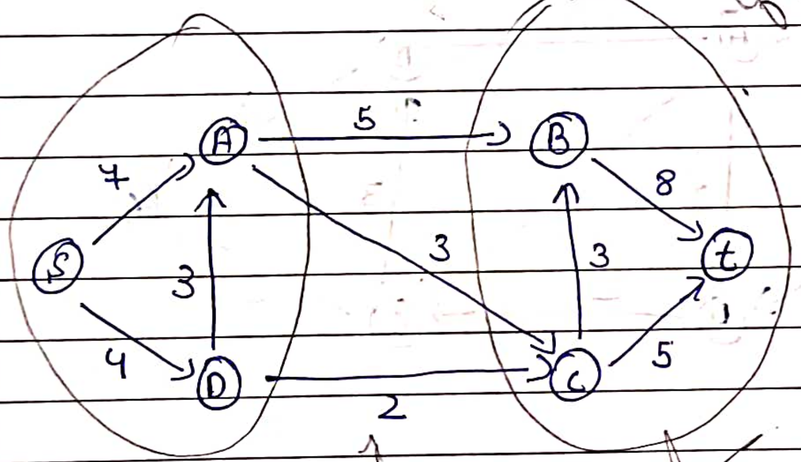
Given a flow network G and a flow f on G, we define the residual graph G_f of G with respect to f as follows.

1) The nodes of G_f will be as that of G
2) For each edge $e = (u, v)$ on which $f(e) < c_e$, there are $(c_e - f(e))$ "leftover" units of capacity. So we include the edge e in G_f with a capacity $c_e - f(e)$
3) For each edge $e = (u, v)$ of G on which $f(e) > 0$, there are $f(e)$ units of flow that we can undo if we want to do by pushing it backwards so, we include $e' = (v, u)$ in G_f with a capacity of $f(e)$

27/02/2014

The used lines are in reverse direction and unused in turn are same direction in Residual graph.





S → 11
t ← 13

Bottom Neck

Min-cut :- By removing the edge, ~~Vertex the Vertex~~ graphs splits in two parts.

Bottleneck:-

Let P be a simple s-t Path in G_f we define bottleneck (P, f) to be the minimum residual capacity of any edge on P with respect to f.

The result of argument (f, P) is a new graph f' in G.

Lemma 7.1 :-

f' is also a flow in G.

Proof :-

Capacity condition. → New flow / Residual graph flow

$$0 \le f(e) \le f'(e) = f(e) + \text{bottleneck}(P, f) \le$$

$$f(e) + (C_e - f(e)) = C_e$$

f(e) → flow of edge

C_e → Capacity of edge

$\forall e \; f(e) \geq f'(e)$

$= f(e) - \text{bottleneck}(P, f) \geq f(e)$

For Conservation :-

At each node :

Change in flow entering = Change in amount of flow exiting.

For Q - Edmonds Algorithm :- → it helps us to find the maximum flow.

Max - flow.

Initially $f(e) = 0$ for all e in G, while there is an s-t Path in residual graph G_f.

Let P be a Simple s-t Path in G_f

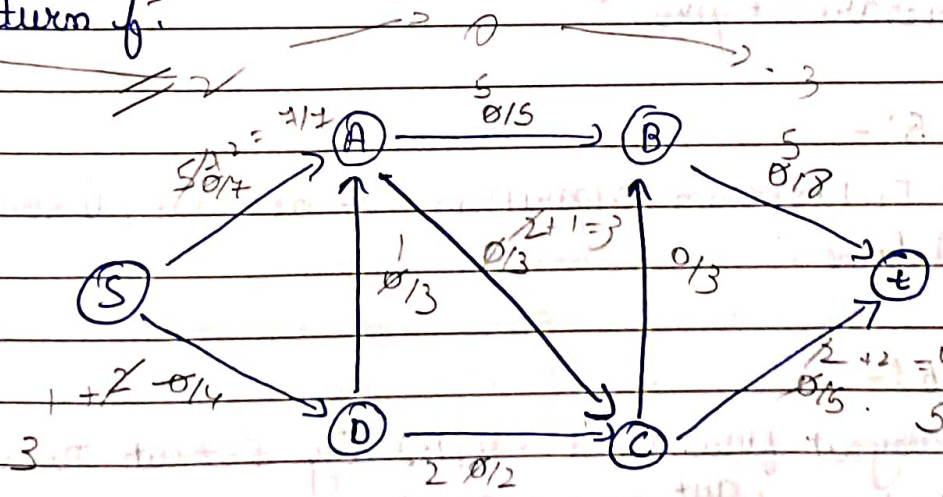
$f' = \text{augument}(f, P)$

update f to be f'

update the residual graph G_f to be $G_{f'}$

endwhile.

return f.



All possible Path

S A B T ✓

S D C T ✓

S A C T ✓

S D A C T ✓

S D A C B T ×

S A C B T ×

S D C B T ×

S D A B T ×

Max - flow = 10

Lemma 7.2:-
At every intermidiate stage of the algorithm, the flow values ($f(e)$) and the residual capacity is c_f are integers.

Proof:- → bottleneck after graph
- It is True before loop.
- Let it be true after j iterations.
- Since all residual capacities are integers in G_f the value bottleneck ($P.f$) for the path found in iteration j will be integers.
- Thus, the flow f will have integer value.

Lemma 7.4 :-
The Robin-Karp algorithm terminates at most C iterations of the while loop.
↙ Ford-Fulkerson

Proof :- Start the loop with 0
C is the bottleneck capacity
All capacities are integers.

Proof :- We Noted above that the algo terminates in at most C iteration.

Lemma 7.5:-
The Ford-Fulkerson algorithm can be implimented in $O(m C)$ time.

Lemma 7.6 :-
Let f be any s-t flow and (A, B) any s-t cut Then $V(f) = f^{out}(A) - f^{in}(A)$

Proof:-
$$V(f) = f^{out}(s)$$
$$f^{in}(s) = 0.$$
$$V(f) = f^{out}(s) - f^{in}(s)$$
$$f^{out}(v) - f^{in}(v) = 0$$

$$V(f) = \sum_{v \in A} (f^{out}(v) - f^{in}(v))$$

$$= \sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in to } A} f(e) = f^{out}(A) - f^{in}(A)$$

Lemma 7.7:-

Let f be any s-t flow and (A, B) be any s-t cut Then

$$V(f) = f^{in}(B) - f^{out}(B)$$

Proof:-

If we Set $A = V - t$ and $B = t$ than we have

$$V(f) = f^{in}(B) - f^{out}(B)$$

$$= f^{in}(t) - f^{out}(t).$$

Lemma 7.8:-

Let f be any s-t flow and (A, B) by and cut Then

$$V(f) \leq c(A, B)$$

Proof:-

$$V(f) = f^{out}(A) - f^{in}(A) \leq f^{out}(A)$$

$$= \sum_{e \text{ out of } A} f(e) \leq \sum_{e \text{ out of } A} c_e$$

$$= c(A, B)$$

Lemma 7.9:-

If f is an s-t flow Such that there is no s-t Path in the residual graph G_f then there is an s-t cut (A^*, B^*) in G for which $V(f) = c(A^*, B^*)$

Proof:-

Let A^* denotes a Set of nodes in V in G for which there is an s-V Path in G_f.

$$B^* = V - A^*$$

$$V(f) = f^{out}(A^*) - f^{in}(A^*)$$

$$= \sum_{e \text{ out of } A^*} f(e) - \sum_{e \text{ in to } A^*} f(e)$$

$= \sum_{e \text{ out of } A} c(e) - 0 = c(A^*, B^*)$

Lemma 7.10 :-
The flow f returned by ford-function algorithm is a maximum flow.

Proof :-
Ford Fulkerson algorithm terminates when there is no s-t Path, This implies it returns the maximum flow.

Lemma 7.11
Given a flow f of maximum value we can compute an s-t cut of minimum capacity in O(m) time.

Proof :-
we construct the Residual graph G_f and Perform BFS and DFS to determine the set A^* of all ~~in~~ nodes that s can reach.
$B^* = V - A^*$

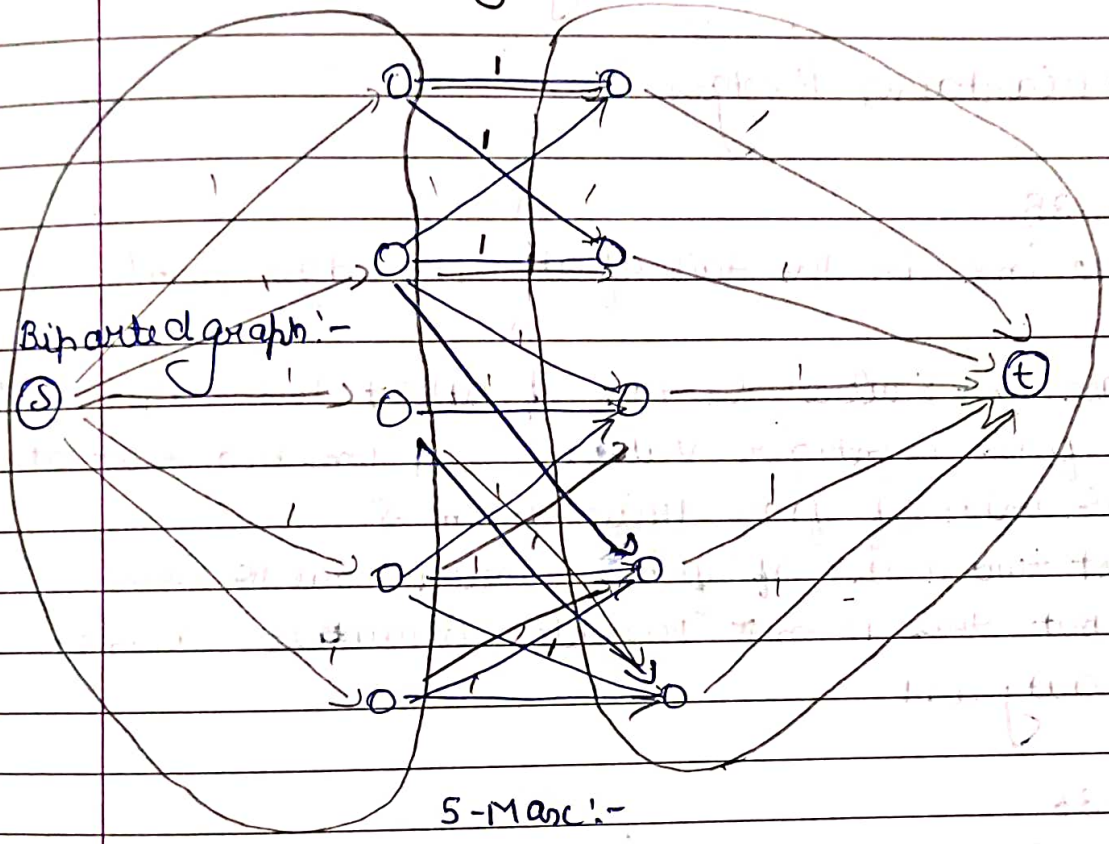
Lemma 7.12
In every flow network, there is a flow f and a cut (A, B) so that $v(f) = c(A, B)$

Proof :-
f - is maximum flow.
if f' is greater than f it must increase $c(A, B)$
if $c(A', B') < c(A, B)$ this means f became Smaller.

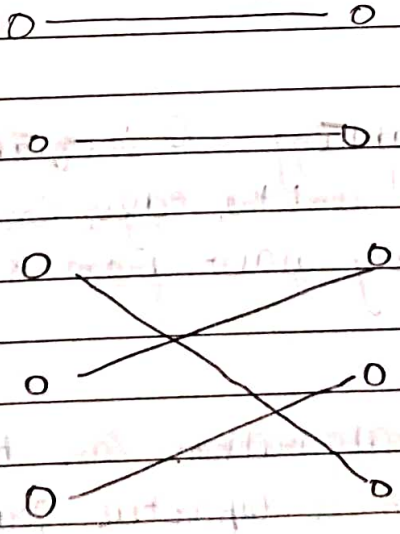
Lemma 7.13
Max flow min cut theorem :-
In every flow network, the maximum value of an s-t flow is equal to minimum capacity of s-t cut

Lemma 7.14:-
If all capacities in the flow network are integers, then there is a maximum flow f for which every flow value f(e) is an integer.

Bipartite Matching:- → Ford - Fulkerson.



5-Marc:-



Suppose there is a flow f' in G' of value k. We know there is an integer valued flow f of value k, Since here all capacities are 1 that means. so the values are either zero or 1 for each e.

Consider the set M' of edges from (x, y) and the value of each edge is 1

Lemma 7.34
M' Contains K edges.

Proof :-
Consider the Set (A, B) in A' with $A = \{s\} \cup X$
The value of flow = Total flow leaving A - total flow entering A.
$= K - 0$
Thus M' Contains K edges.

Lemma 7.35
Each node in X in the tail of atmost edges in M'

Proof :-
-> Suppose $x \in X$ where the tail of atleast two edges in M'
-> Since our flow is intiger valued, this means that at least two units of flow leave from s.
-> So, atleast two units of flow would have to come into x. But this is not Possible. Because, a single edges capacity is 1

Lemma 7.36
Each node in X is the Head of atmost one edge in M'

Lemma 7.37
The size of maximum matching in G is equal to the values of the maximum flow in G' and the edges in such a matching in G. are the edges that carry flow from X to Y in G'

Lemma 7.38
The ford - fullerson algorithm can be used to find a maximum matching in a bipartite graph in O(mn) time.

P - / NP / NP - Complete / NP - Hard

05/03/2024 CH-2

Reduction for algorithm

If we can translate the input for a Problem we want to Solve into inputs for a Problem we know how to Solve, we can compose the translation and the Solution into an algorithm for our Problem.

Example :- Bipartite
→ Network flow.

Example :- Closest Pair.
(5, 7, 3, 2, 1, 0, 6, 9, 8 ....
) 1. Sort.

Closet Pair:-
The Closet Pair Problem asks to find the Pair of number with in a Set that have the smallest difference between. Then we can make it decision Problem by asking if the value is less than some there Shold.

input : A Set S of n number, and a threshold t

output : Is there a Pair of $i, S_j \in S$ such that $|S_i - S_j| < t$?

Example :- LCM
(25, 60)
25 : 1, 5
60 : 1, 2, 3, 5

GCD / LCM :- (Euclidian Algorithm) :-

gcd (25, 60)

q	m_1	m_2	r
2	60	25	10
2	25	10	5
2	10	5	0
	5	0	

gcd (25, 60) = 5.

LCM :-

Problem :- Least Comman multipale.

input :- Two intiger x and y.

Output :- Smallest intiger m such that m is a multipal of x and y

GCD :- Greatest Commandivisar

Problem :- Two intiger x and y.

Output :- Return the largest intiger d such that d divides x and y.

Example :- Find the LCM and GCD of (24, 36) using Euclidian Algorithm.

Ans - GCD

q	m_1	m_2	r
1	36	24	12
2	24	12	0
	12	0	

gcd (24, 36) = 12

LCM :- LCM (24, 36) = 2.

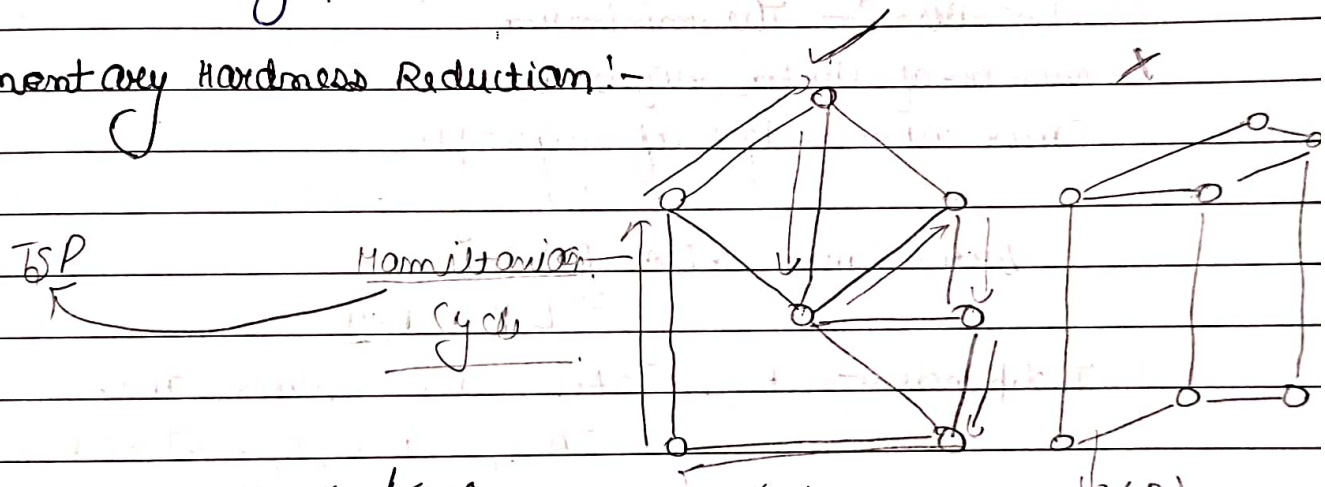
Convex Hull :-

A Polygon is Convex if the straight line segment drawn between drawen between any two Points inside the Polygon Ponust lie Completely with in the Polygon.

Problem :- Convex Hull
Input :- A Set of n Points in the Plane.
Output :- Find the Smallest Convex Polygon containing all Points of S.

We must translate each number to a Point we do so by mapping x (x, x²). If means each integer is mapped to a Point on the Parabola y = x². Since Parabola is Convex every Point must be in Convex hull.

Elementary Hardness Reduction :-



Hamiltonian Cycle :- // TSP :-
Hamiltonian ↳ (Travelling Sales Person)

- In Hamiltonian we have many options / Paths
- In Travelling Sales Person there is only one Path (where the weight should be minimum)
- Unweighted graph ∈ O
- weighted graph ∈ O

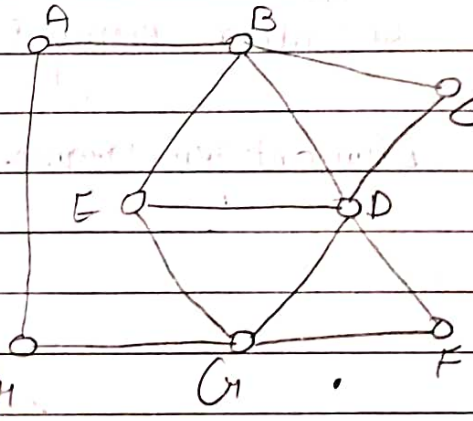
Travelling Sales Person ⟶ Hamiltonian Cycle. //
↳ If Can MAKE a Solution For Hamiltonian Cycle.

Hamiltonian Cycle :-
Input :- An unweighted Graph G. //
Output :- Does there exists a Simple tour that Visits each Vertex of G without repetation. //

Reduction from Hamiltonian Cycle to Traveling Salesperson is both efficient and truth preserving.

A fast algorithm for travelling Sales person means a fast algorithm for Hamiltonian Cycle. This implies their time complexity are related. Independent Set - For a graph $G = (V, E)$ a set of nodes $S \subseteq V$ is called independent set if no two nodes in S are connected by an edge $e \in E$

Independent Set and Vertex Cover:-



Vertex Cover:- The minimum Number of Vertex required to cover all the edges of the graph.

Vertex Cover → {A, B, D, G}
or [H, B, D, G]

Independent Set = Total Vertex - Vertex Cover
= [A, B, C, D, E, F, G, H] - [A, B, D, G]
= [H, C, E, F]

- It ask for a small set of vertices that contacts each edge in a graph.

input:- A graph $G = (V, E)$ and an integer $k \leq |V|$
output:- Is there a subset S of at most k vertices such that every $e \in E$ contains at least one vertex in S.

A simple reduction shows that the two problems are identical. The hardness of Vertex Cover implies that Independent Set also be hard.

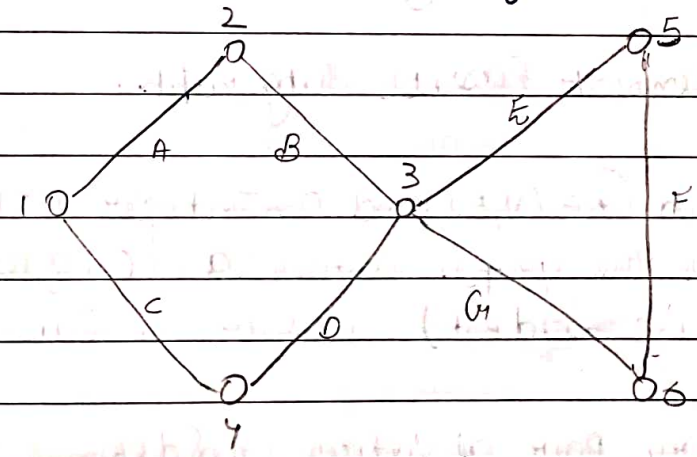
Independent Set ⟶ Vertex Cover

Hamiltonian ------> Vertex
Cycle Cover

General Movie Scheduling :- To independent Set Problem.

Proj A	Proj B	Proj C.
Jan - Mar.	Apr.	June - July.
May - Jun	Aug	

Input :-
A Set I of n Sets of Intervals on the line, intervals. R.
Output :- Can a Subset of at least K mutually non-overlaping interval which can be Selected from I.



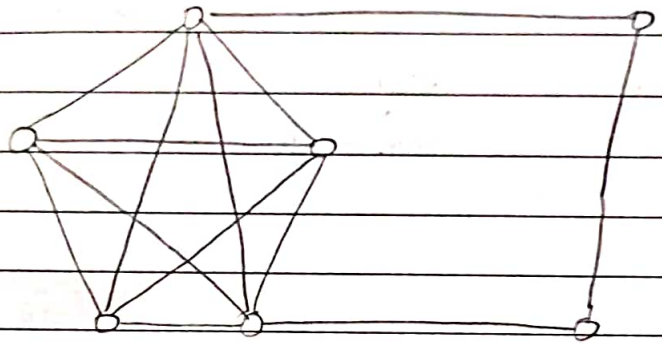
Vertices : Movies
Edges : Month

	A	B	C	D	E	F	G
1	—		—				
2	—	—					
3		—		—	—		—
4			—	—			
5					—	—	
6						—	—

NOTE :- It is equivalent to Value for independent Sets.
So, the general movie Scheduling must be hard as hard as Independent Set.

General exam Scheduling ——→ Independent Set. //

Clique :-



• A Social Clique is a group of ~~natural~~ mutual friends who all hang around together.

• A graph theoritic Clique is a Complete Sub graph. where each vertex pair has an edge between them.

• Cliques are densest Possible Subgraph.

Input :- A graph G = (V, E) and an Integer K ≤ |V|
output :- Does the graph contain a Clique of K vertices, that is (i.e → Id est) is there a Subset of S ⊂ V. where |S| ≥ K, (→ Came from Ramman in english that is. Such that every pair of vertices in S defines an ~~edd~~ edge of G

Solutions :-

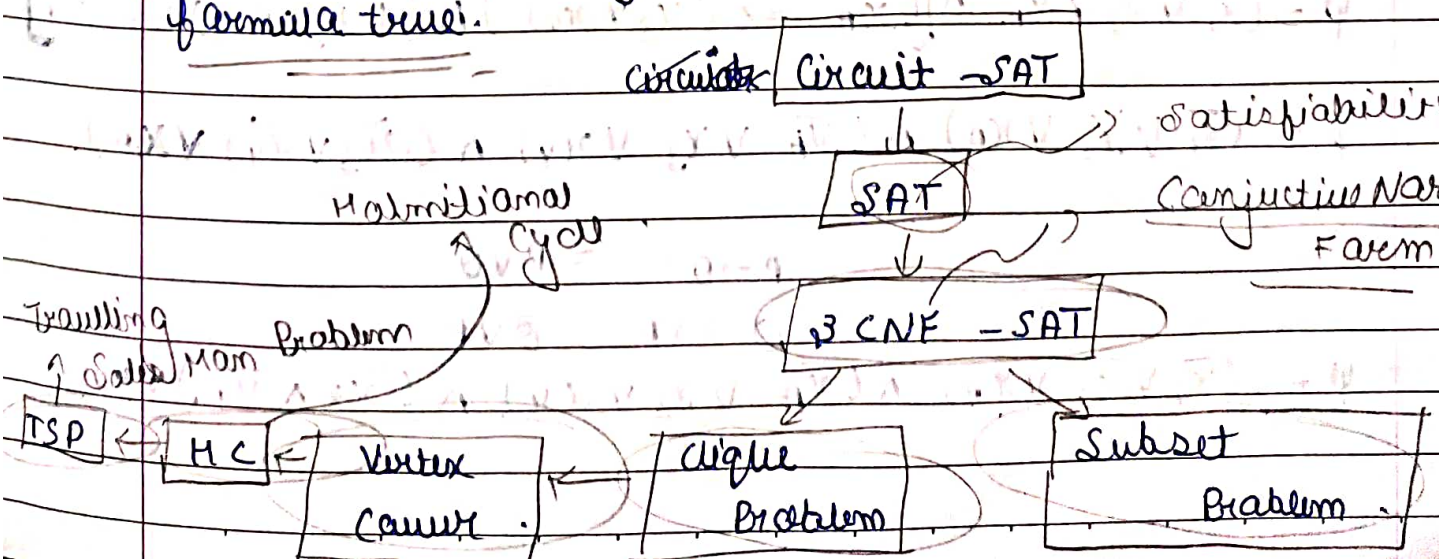
Reverse the edges and non ~~-~~ edges in Clique. It will reduce to an equivalent Independent Set.

Clique ——→ Independent Set. //

NP	NP-Hard	NP-Complete.
The Problem which is Not Solvable in Polynomial time of Period of time. but Can be verified with the answer in Polynomial Period of time.	• If a Problem which is in NP and Can be reduced to an another known Problem in Polynomial Period of time.	Independent Set ↓ Vertex Cover ↓ Set Cover ↓ NP Complete. → NP Hard. → NP Hard.
	• If X is NP Hard if and Y is unknown, is an unknown NP Problem Y and Can be reduced to X in Polynomial Period of time it will be a NP-Hard Problem	

Boolean Satisfiability Problems (BSAT Problem) :-

Boolean Satisfiability Problem is a Problem in Computer Science of determining if there exist an Interpretation of a given Boolean formula that makes the formula true.



SAT $\propto_p$ 3SAT $\propto_p$ IS | VCP $\propto_p$ SC → Set Cover

Independent Set → Vertex Cover

LAB-2

Q1 write a java program for greedy approach for activity selection Problem.

	1	2	3	4	5
w =	3	3	2	5	1
P =	10	15	10	20	8
P/w =	3.33	5	5	4	8

Sort according to P/w

	5	2	3	4	1
w =	1	3	2	5	3
P =	8	15	10	20	10
P/w =	8	5	5	4	3.33

Q2 write a java Program to implement greedy approach for fractional Knapsack Problem.

$\psi = (\bar{x}_1 \vee x_2 \vee x_3) \wedge (x_2 \vee \bar{x}_3 \vee x_4) \wedge (\bar{x}_1 \vee \bar{x}_3 \vee x_3 \vee x_4)$

$\psi = \bar{x}_1 \vee \bar{x}_3 \vee x_3 \vee x_4$ SAT to 3Sat [$P \Leftrightarrow Q$] [$P \to Q \wedge Q \to P$]

$\psi = (\bar{x}_1 \vee \bar{x}_2 \vee x_A) \wedge (x_A \Leftrightarrow (x_3 \vee x_4))$

$\psi = (\bar{x}_1 \vee \bar{x}_2 \vee x_A) \wedge [(x_A \Rightarrow (x_3 \vee x_4)) \wedge ((x_3 \vee x_4) \Rightarrow x_A)]$

$\psi = (\bar{x}_1 \vee \bar{x}_2 \vee x_A) \wedge (\bar{x}_A \vee x_3 \vee x_4) \wedge ((\bar{x}_3 \vee \bar{x}_4) \vee x_A)$

$P \to Q = \bar{P} \vee Q$

$Q \to P = \bar{Q} \vee P$

$\psi = (\bar{x}_1 \vee \bar{x}_2 \vee x_A) \wedge (\bar{x}_A \vee x_3 \vee x_4) \wedge ((\bar{x}_3 \wedge \bar{x}_4) \vee x_A)$

P → Q (P + 8) (P + Q)

M T W T F S S
Page No.:
Date:
YOUVA

$\psi = (\bar{x}_1 \vee \bar{x}_2 \vee x_A) \wedge (\bar{x}_A \vee x_3 \vee x_4) \wedge ((x_A \vee \bar{x}_3) \wedge (x_A \vee \bar{x}_4))$

$\psi = (\bar{x}_1 \vee \bar{x}_2 \vee x_A) \wedge (\bar{x}_A \vee x_3 \vee x_4) \wedge (x_A \vee \bar{x}_3 \vee x_A)$
$\wedge (x_A \vee \bar{x}_4 \vee x_A)$

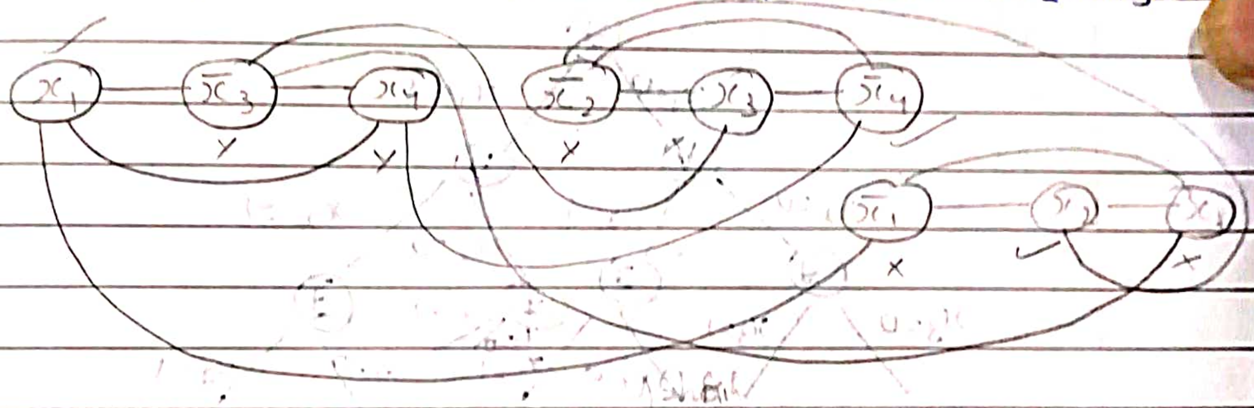
→ $P \vee P = P$

Same we use ($x_A \vee \bar{x}_3 \vee \bar{x}_3 \vee x_A$)

($x_A \vee \bar{x}_3 \vee x_A$)

3SAT ≤p IS → Independent Set

$\phi = (x_1 \vee \bar{x}_3 \vee x_4) \wedge (\bar{x}_2 \vee x_3 \vee \bar{x}_4) \wedge (\bar{x}_1 \vee x_2 \vee x_3)$



$IS = \{x_1, x_2, \bar{x}_4\}$

P ⊆ NP → P is a subset of NP → Verified in Polynomial Time
(→ that solve in) Problem Polynomial amount of time.

(Can solved + verified in Polynomial time)

Suppose x is an NP complete problem then x is solvable in Polynomial time if and only if P = NP

P ⊆ PSPACE → solvable in Polynomial space.
P ⊆ NP
NP ⊆ PSPACE (For similar value)

Only 2-SAT - Belong to NP Class and all NP others Not

Belongs to the class NP, PSPACE

M T W T F S S
Page No.:
Date:
YOUVA

Quantification (Q-SAT) :- (odd Bream) decision depends on the Boolean Bream

If $\phi(x_1, x_2, \ldots, x_m)$ is a boolean CNF Formula is the following true ?

$$\exists x_1 \forall x_2 \exists x_3 \forall x_4 \cdots \exists x_{m-1} x_m$$

T F T

$$(x_1 \vee x_2) \wedge (x_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee x_3) = T$$

F F T F T F

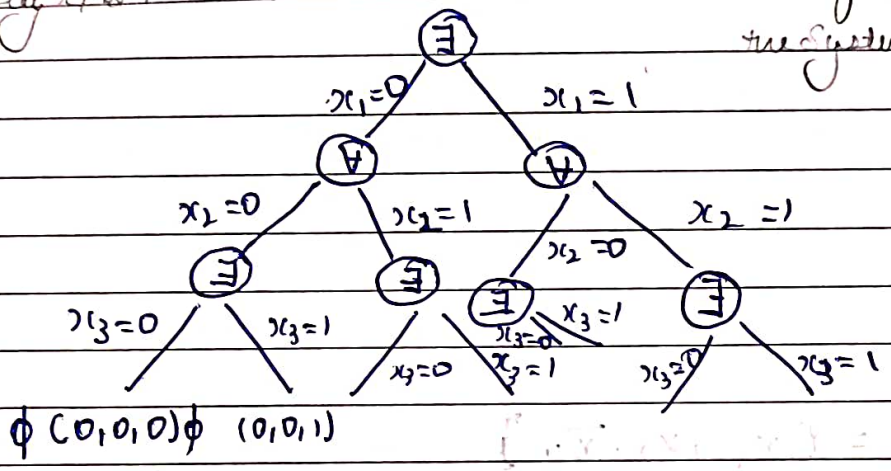
Assume False

$x_3 \to T$

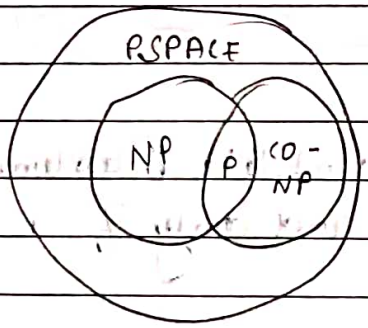
A · B

Say x_1 is T

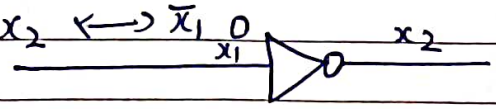
decision of first Bream evaluate the System choice.



There is an algorithm that solves 3-SAT using only a Polynomial amount of Space.



$\phi = x_2 \longleftrightarrow \bar{x}_1$



$A \longleftrightarrow B \equiv (\bar{A} + B)$

$= (1)(A + \bar{B})$

Distributive law $A(B+C) = AB + AC$

$A+(B.C) = A+B(A+B)$

Demorgan's law:

$\overline{A \cdot B} = \overline{A} + \overline{B}$

$\overline{A+B} = \overline{A} \cdot \overline{B}$

$x_2 \longleftrightarrow \overline{x_1}$

$\Rightarrow (\overline{x_2} + \overline{x_1})(x_2 + \overline{x_1})$

$= (\overline{x_1} \vee \overline{x_2}) \wedge (x_1 \vee x_2)$

$x_3 \longleftrightarrow x_1 \cdot x_2$.

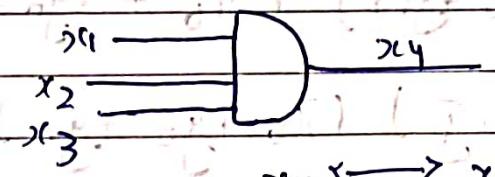
$= (\overline{x_3} + \overline{x_1 x_2})(\overline{x_3} + x_1 \cdot x_2)$.

$= (x_3 + \overline{x_1} + \overline{x_2})(\overline{x_3} + x_1 \cdot x_2)$.

$= (\overline{x_1} + \overline{x_2} + x_3)(\overline{x_3} + x_1)(\overline{x_3} + x_2)$

$\downarrow \quad \downarrow \quad \downarrow \quad \downarrow \quad \downarrow \quad \downarrow$

$\vee \quad \vee \quad \wedge \quad \vee \quad \vee$

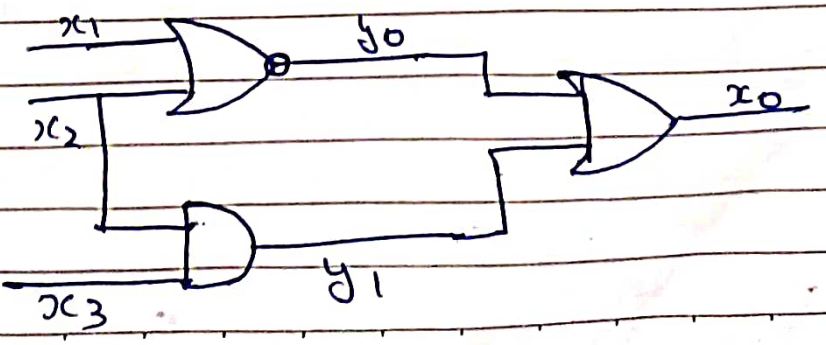


$x_4 \longleftrightarrow x_1 \cdot x_2 \cdot x_3$.

$= (x_4 + \overline{x_1 \cdot x_2 \cdot x_3})(\overline{x_4} + x_1 \cdot x_2 \cdot x_3)$.

$= (x_4 + \overline{x_1} + \overline{x_2} + \overline{x_3})(\overline{x_4} + x_1 \cdot x_2 \cdot x_3)$.

$= (x_4 + \overline{x_1} + \overline{x_2} + \overline{x_3})(\overline{x_4} + x_1)(\overline{x_4} + x_2)(\overline{x_4} + x_3)$



$y_0 \longleftrightarrow \overline{x_1 + x_2}$ $\qquad x_0 \longleftrightarrow y_0 + y_1$

$y_1 \longleftrightarrow x_2 \cdot x_3$

$y_0 \longleftrightarrow \overline{x_1 + x_2}$

$= (y_0 + (\overline{x_1 + x_2}))\ (\bar{y}_0 + \overline{x_1 + x_2})$

$= (y_0 + (x_1 + x_2))\ (\bar{y}_0 + \bar{x}_1 \cdot \bar{x}_2)$

$= (y_0 + x_1 + x_2)\ (\bar{y}_0 + \bar{x}_1)\ (\bar{y}_0 + \bar{x}_2)$

$y_1 \longleftrightarrow (x_2 \cdot x_3)$

$\Rightarrow (y_1 + (\overline{x_2 \cdot x_3}))\ (\bar{y}_1 + x_2 \cdot x_3)$

$\Rightarrow (y_1 + \bar{x}_2 + \bar{x}_3)\ (\bar{y}_1 + x_2)\ (\bar{y}_1 + x_3)$

$x_0 \longleftrightarrow y_0 + y_1$

$(x_0 + (\overline{y_0 + y_1}))\ (\bar{x}_0 + (y_0 + y_1))$

$(x_0 + \bar{y}_0 \cdot \bar{y}_1)\ (\bar{x}_0 + y_0 + y_1)$

$(x_0 + \bar{y}_0)\ (x_0 + \bar{y}_1)\ (\bar{x}_0 + y_0 + y_1)$

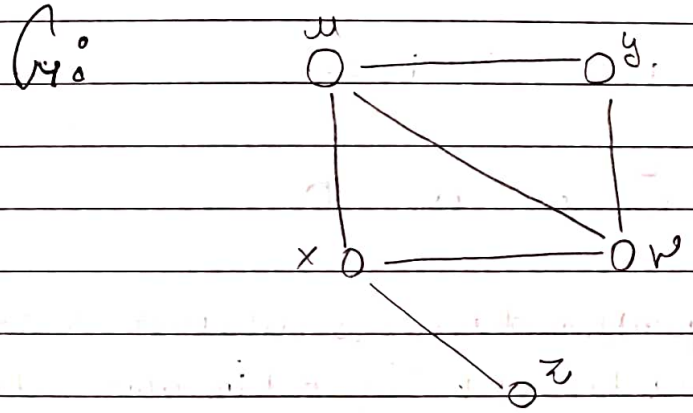
$(x_0 + [(y_0 + x_1 + x_2)\ (\bar{y}_0 + x_1)\ (\bar{y}_0 + x_2)])\ (x_0 + [(y_1 + \bar{x}_2 + \bar{x}_3)\ (\bar{y}_1 + x_2)\ (\bar{y}_1 + x_3)])$

Vertex Cover:-

- Solve Problem to optimality.
- Solve Problem in Polynomial time.
- Solve arbitary Instances of the Problem.

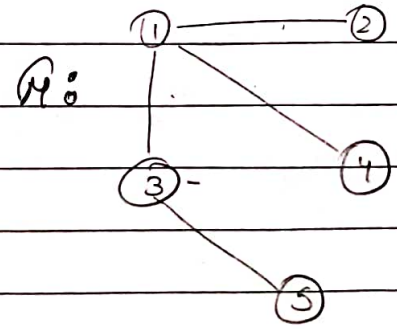
Brute Force approach:- $O(Kn^{K+1})$ ^nC_K $n=10$ $K=4$

- The claim is "let $u-v$ be an edge of graph G". G has a Vertex Cover of Size $\leq K$ if and only if atleast one of "$G-\{u\}$" and "$G-\{v\}$" have a Vertex Cover of Size $\leq K-1$



if we delete Vertex v we edge will get deleted.

- If G has a Vertex Cover of Size K then it has $\leq K(n-1)$ edges.



Vertex Cover = 2

No. of edges $\leq 2(5-1)$

$\leq 2(4) = 8$

Algorithm:-

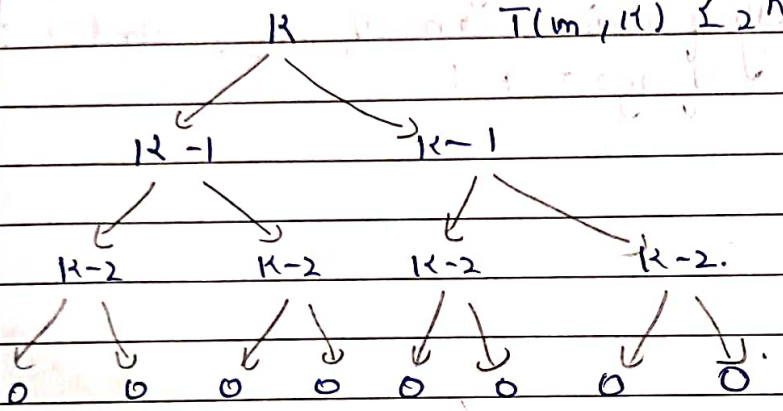
```
boolean VC (G, K)
    if (G contains no edges) return True.
    if (G contains >K*n edges) return False
    Let (u, v) be any edge in G
    a = VC(G - {u}, K-1)
```

$b = VC(G - \{V\}, K-1)$

return a or b

}

$$T(m,K) = \begin{cases} c\,m & \text{if } K=1 \\ 2T(m,K-1) + CKm & \text{if } K>1 \end{cases}$$



$T(m,K) \leq 2^K CKm$.

The following Algorithm determines whether the graph G has a vertex cover of the Provided size(K) in O(mK) time Complexity.

Approximation Algorithm. $O(2^K Km)$.

a) Load Balancing :- (Distribution of work among machine)

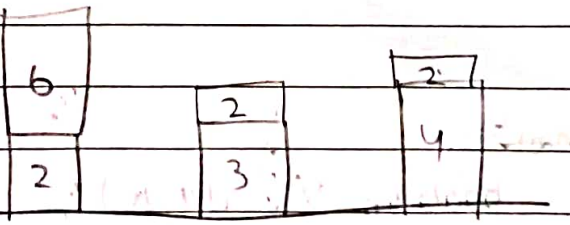
m -> No of machine

where $m' \geq m$ n -> No of work

is possible.

Greedy :-

i	tj
1	2
2	3
3	4
4	6
5	2
6	2



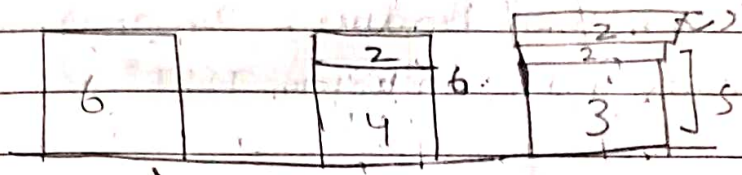
MakeSpan = 8

MakeSpan in Greedy $\leq 2m-1$

= (when sorted in increasing or

Approximation algorithm

6 4 3 2 2 2 → average work in descending order



This is odder here as before it has less work

M_1 6 M_2 6 M_3 7

Here time Span = 7

Let A(i) be the set of jobs assigned to machine M(i) then Mi needs to work for $T_i = \sum_{j \in A} t_i$ $\quad T = \max(T_i)$

Greedy Algorithm:-

i) Starts with no jobs assigned.

ii) Set $T_i = 0$ and $A(i) = \phi$ for all machines M

iii) For i = 1 to m

Let Mi be a machine that achieves the min T_k

Assign job j to machine Mi

Set $A(i) \leftarrow A(i) \cup \{j\}$

Set $T_i \leftarrow T_i + t_j$

End For.

Make Span = max time taken by the Machine.

→ Time Span of each machine

ex:- 2, 2, 2, 3, 4, 6.

M_1, M_2, M_3.

Lemma 11.1 $T^* \geq \frac{1}{m} \sum_j t_j$

T^* → optimal time span

m → no. of machines.

$\sum_j t_j = 2+2+2+3+4+6$

$= 19$

T^* = make Span with greedy = 8.

$\frac{1}{m} \sum_j t_j = \frac{19}{3} \approx 6.$

$\therefore 8 \geq 6$ (True)

Print Start

BOOK.

Lemma 11.2 The optimal makespan is atleast $T^* \geq \max_j T_j$

imp Lemma 11.3:- Greedy Balance algorithm Produces an assignment of jobs to machines with makespan $T \leq 2T^*$

(T → Regular makespan; T^* → optimal to makespan)

$i \rightarrow L[i] + T_j$

j

$L[i] - t_j \leq L[k]$

$1 \leq k \leq m$ → no. of machines

$L[i] - t_j \leq \frac{1}{m} L[k]$ → From Lemma 1 $T^* \geq \frac{1}{m} \sum_k t_k$

$L[i] - t_j \leq L^*$

$L \leq L[i]$

$\leq (L[i] - t_j) + t_j$ $\boxed{L \leq L^* + L^* = 2L^*}$

m machines.

m(m-1) jobs.

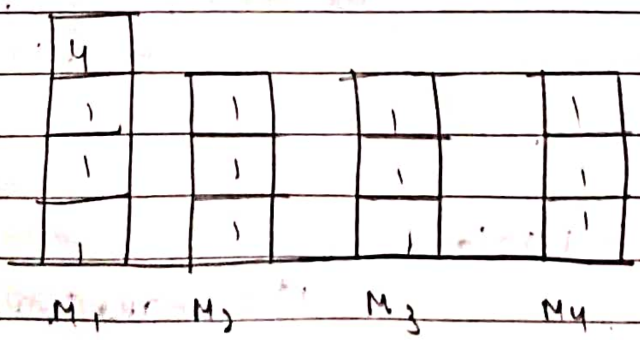
m-1 → 1

j → m.

the first m(m-1) Jobs = 1

12 Jobs → 1 m = 4.

last Jobs → m. = 4.

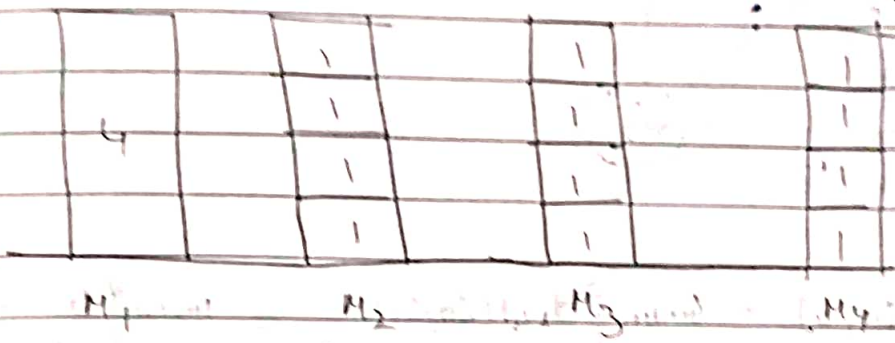


GA- makeSpan = 7

approximation:-

Short jobs in descending order:-

4, 1, 1, ... ; 1. This algorithm is known as Largest Processing Time (LPT)



Make Span = 4. (optimal solution).

~~4*~~ $T \leq 2T^*$

$7 \leq 2 \times 4$ $\quad 7 \leq 8$

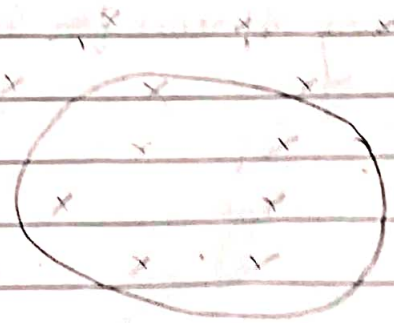
Lemma 11.4 If there are more than m jobs then $L^* \geq 2t_{m+1}$

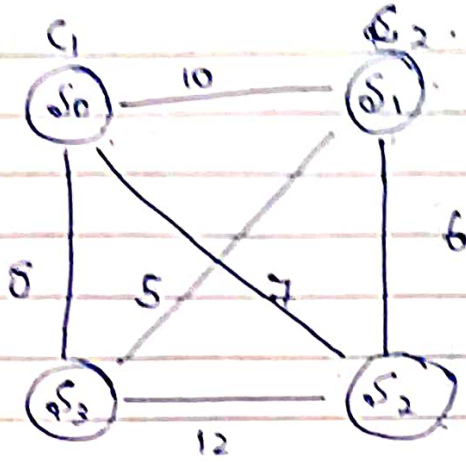
$L^* \geq 2t_{m+1}$

Lemma 11.5 LPT is $\frac{3}{2}$ Approximation algorithm.

b) Center Selection Problem:- Suppose if there are 50,000 houses in NY and government want to build a Shopping Complex which is easy to all houses (every one has to take similar path to reach by all houses)

It is a NP-Hard Problem.





$C = \{C_1, C_2\}$.

$$dist(S_0, C) = \min \begin{cases} dist(S_0, C_1) \\ dist(S_0, C_2) \end{cases} = \min \begin{cases} 0 \\ 10 \end{cases} = 0.$$

$$dist(S_1, C) = \min \begin{cases} dist(S_1, C_1) \\ dist(S_1, C_2) \end{cases} = \min \begin{cases} 10 \\ 0 \end{cases} = 0$$

$$dis(S_2, C) = \min \begin{cases} dist(S_2, C_1) \\ dist(S_2, C_2) \end{cases} = \min \begin{cases} 7 \\ 6 \end{cases} = 6$$

$$dis(S_3, C) = \min \begin{cases} dist(S_3, C_1) \\ dist(S_3, C_2) \end{cases} = \min \begin{cases} 8 \\ 5 \end{cases} = 5$$

$\max(0, 0, 6, 5)$

$= 6$ → The max Shawarma in max radius of 6 → max distance from every House.

Greedy approach for Center Selection Problem.

Improved algorithm :- we have known optimal radius

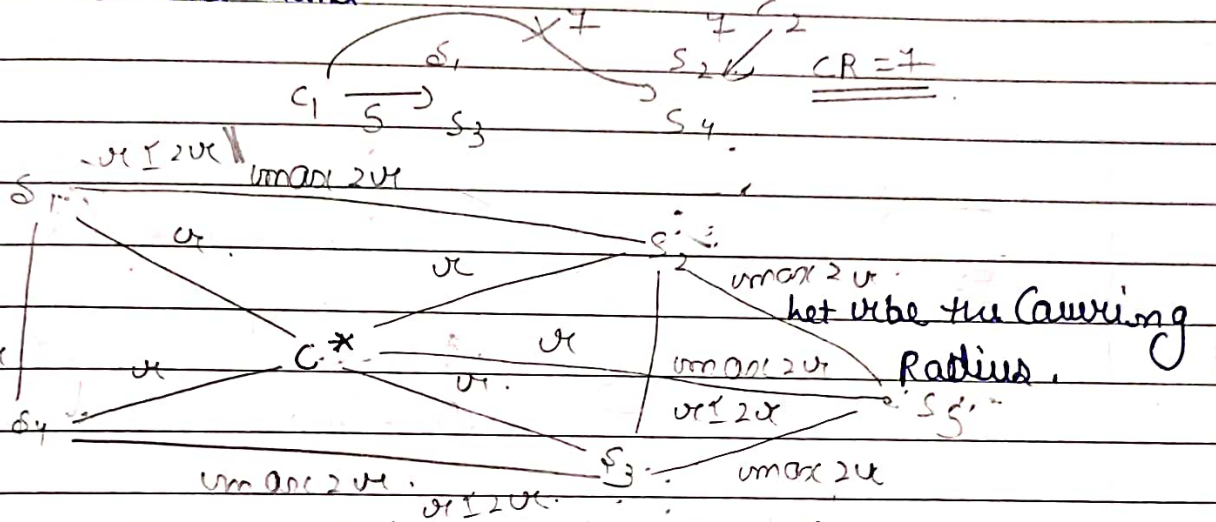
We know there is a set of K Centers C^* with radius $r(C^*)$ i.e.
Job :- Find set of K Centers whose covering radius $\leq r$

(triangle inequality).

if there are more than one center then either of the center should cover the sites for covering of radius.

ii) Greedy Approach when we known the Covering Radius :-

- Covering Radius is the minimum value of r which can cover all the sites in a plane.



∴ The atmost the maximum distance from center to any site is r.

Let r be the Covering Radius.

$S = \{S_1, S_2, S_3, S_4\}$ are max at 2r from S_1

we choose any site as a center and then see which are site at a maximum of 2r and then delete the nodes / sites from the set

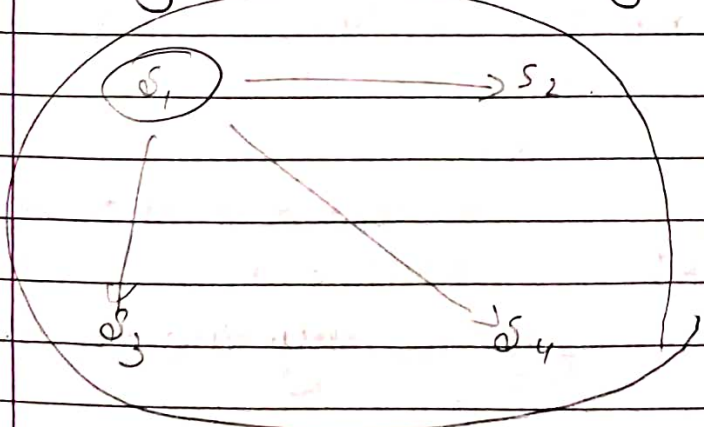
we choose S_1

$C = \{S_1, S_3\}$

$\therefore K = 2$.

iii) Greedy Approach : Covering radius r is unknown to us :-

Let $K = 2$



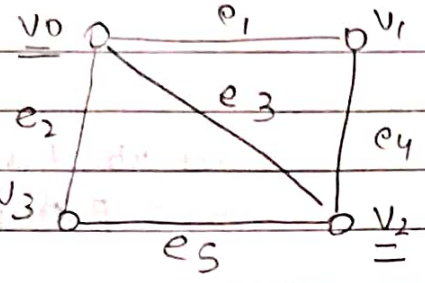
S_5 $S = \{S_1, S_2, S_3, S_4, S_5\}$

Farng :- $C = \{S_1, S_5\}$

S_4 has max dist $K = 2$ we have to find

maximum distance b/w chosen center and the other site

Set Cover Decision Problem is a NP Problem.
Set Cover optimization Problem is NP Hard.



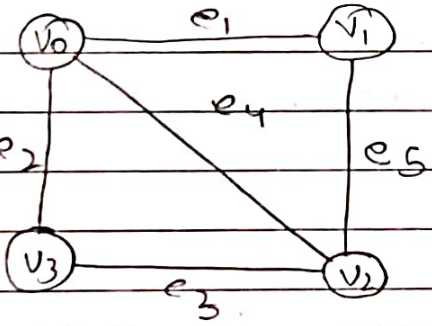
$VC = \{V_0, V_2\}$

$SC_1 = \{e_1, e_2, e_3\}$ → For V_0
$SC_2 = \{e_4, e_5\}$ → For V_2

optimized = $SC = \{\{SC_1, SC_2\}\}$ ②
set-cover

Set Cover optimisation : Approximation Algorithm :-

i) Greedy Approach :- we assign a weight to each of the Set



$SC_1 = \{e_1, e_2, e_4\} \Rightarrow w_1$
$SC_2 = \{e_3, e_5\} \Rightarrow w_2$
$SC_3 = \{e_2\} \rightarrow w_3$

In Greedy Approach :- we not only look weight but also how many elements the Set is covering.

Greedy Set Cover :- $\underline{wi}$ $\quad$ $U \rightarrow$ Universal Set all edges
$\{S_i \cap R\}$ $\quad$ $U = \{1, 2, 3, 4, 5\}$
if U is initially this R is also this
$R = \{1, 2, 3, 4, 5\}$

$c(S_1) = 5 \quad c(S_2) = 10 \quad c(S_3) = 3$

M T W T F S S | Page No.: | Date: | YOUVA

$S_0 = \{S_1, S_2, S_3\}$ $S_1 = \{4, 1, 3\}$. $S_2 = \{3, 5\}$. $S_3 = \{1, 4, 3, 2\}$.

$\alpha_1 = \frac{6w_1}{|S_1 \cap R|} = \frac{5}{3}$ $\alpha_2 = \frac{10}{2}$ $\alpha_3 = \frac{3}{4}$

$\min\left(\frac{w_i}{S_i \cap R}\right)$ as S_3 is minimum edges S_3 are removed.

$R = \{1, 2, 3, 4, 5\}$

as S is still true.

$\alpha_3 = \frac{5}{0} = \infty$ $\alpha_2 = \frac{10}{1} = 10$.

as S_2 is minimum edges common to w R and S are removed

$R = \{1, 2, 3, 4, 5\}$

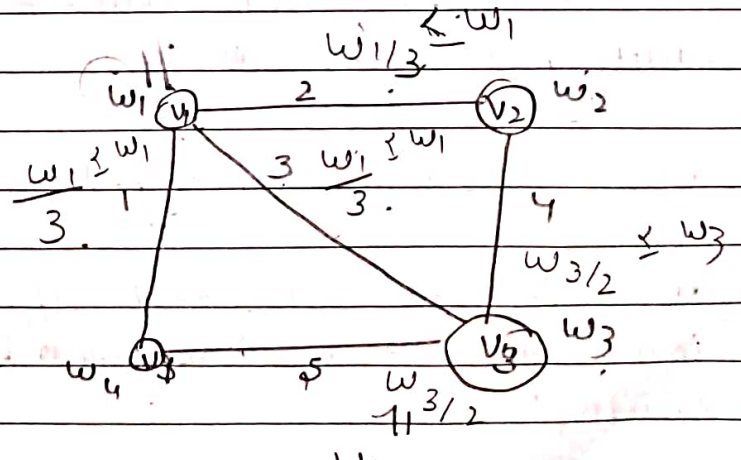
$S_{final} = \{S_2, S_3\}$

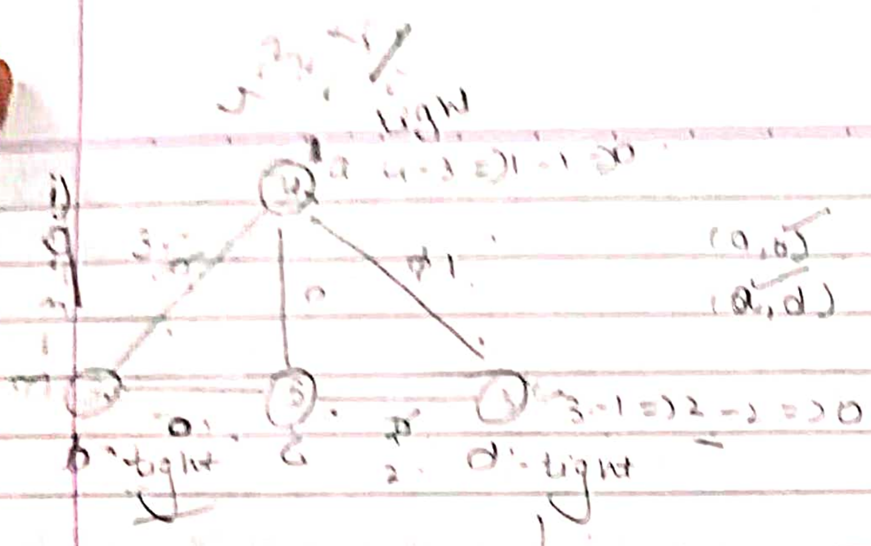
Vertex Cover:-

Pricing Method:-

Each edge must be covered by some vertex i each edge e pays a price pe ≥ 0 who to use vertex i.

Fairness:- edges incident to vertex i should pay less than equal to w_i in total for each each vertex i





By Pricing Method VC = {a, c, d}, {a, b, d}

all tight elements 4+3+3=10

with the help of edges

(a, b) —> 3 × 2 as (a, b are tight) = 6
(a, d) —> 1 × 2 as (a, d are tight) = 2
(c, d) —> 2 as only d is tight = 2

VC 6+2+2 = 10

(a, b, d) → 3 → No. of elements → Pricing Method

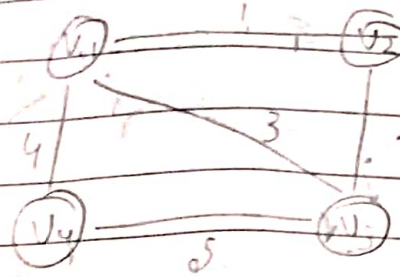
PM ≤ 2 OS → Optimal Solution (a, c) → 2 No of Elements

Local Search :-

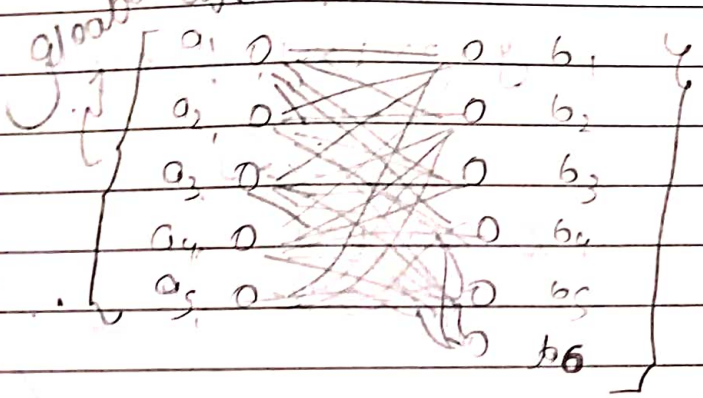
i) **Neighbour Relation :-** S ~ S' if S' can be obtained off from S by adding or deleting a single node.

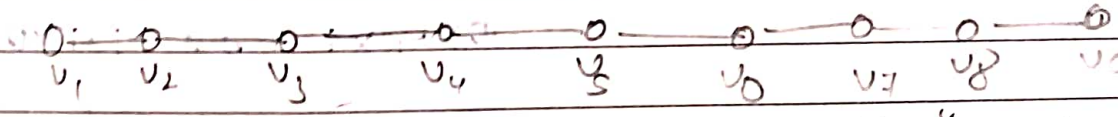
. Each vertex cover S has atmost n neighbour.

ii) Gradient descent :- Start with S = V. If there is a neighbour S' that is a vertex cover and has more lower cardinality replace S with S'



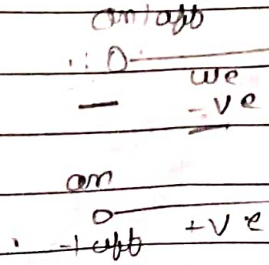
$S = \{v_1, v_2, v_3, v_4\}$.
Cardinality = 4.
Now we delete v_2
$S = \{v_1, v_3, v_4\}$. → Still all the edges are covered.
Cardinality = 3
Now we remove v_4 v_3 as edges are
$S = \{v_1, v_3\}$. = S' → Still all edges are covered
Cardinality = 2.



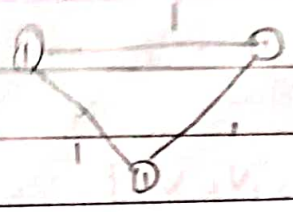


$VC_1 = \{v_2, v_3, v_5, v_6, v_8\}$ → local.
$VC_2 = \{v_2, v_4, v_6, v_8\}$, → global

Local Search Hopfield Neural Network:-
Configuration of the network is an assignment -1 or, +1 to each node v which is called the state of a particular node.
An Node in the neural network can have a state (on/off) on-data flowing through that edge.



off on data flowing → vertices are said to be in same state if they are +ve



(the two vertices have different state)

$\exists$ / we $\delta u \neq \delta v < 0$

↳ Goal

edge. (u,v) → good

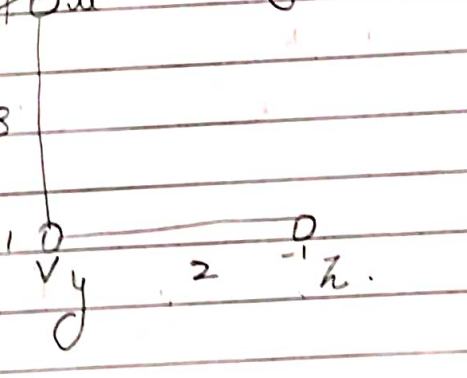
either.

we < 0, $\delta u = \delta v$

(or) we < 0, $\delta u \neq \delta v$ → For good

else bad

↝ For good we change $y + 0 - I$



x: $\exists$ we $\delta u \delta v < 0$

$3 \times 1 \times 1 < 0$

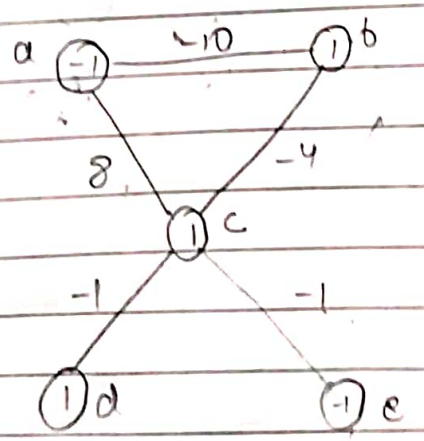
y: $2 \times 1 \times -1 + 3 \times 1 \times -1$

$= -2 - 3$

$= -5 < 0$ ✓ good

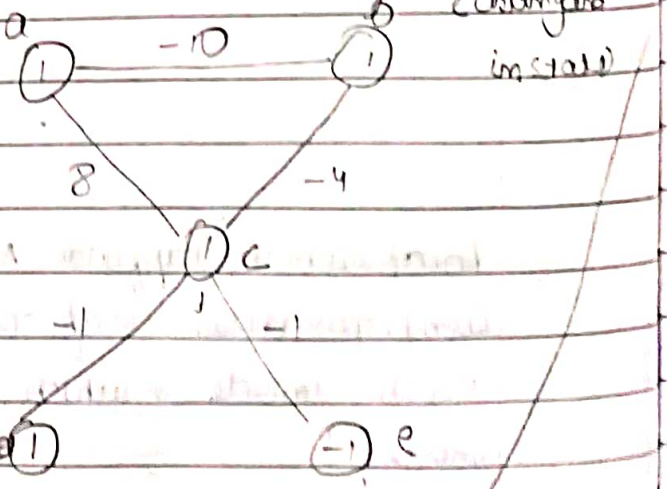
z: $2 \times -1 \times 1$

$= -2 < 0$ ✓ good

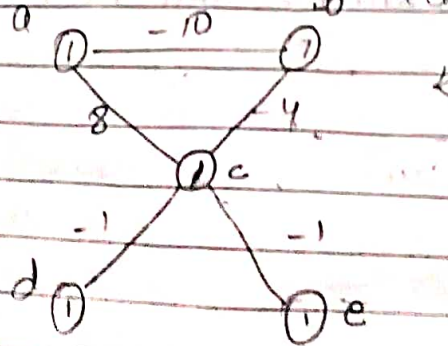


a: $10 - 1 - 2 > 0$ ✓

a: $-10 \times -1 \times 1 + 8 \times -1 \times 1 < 0$ × Bad



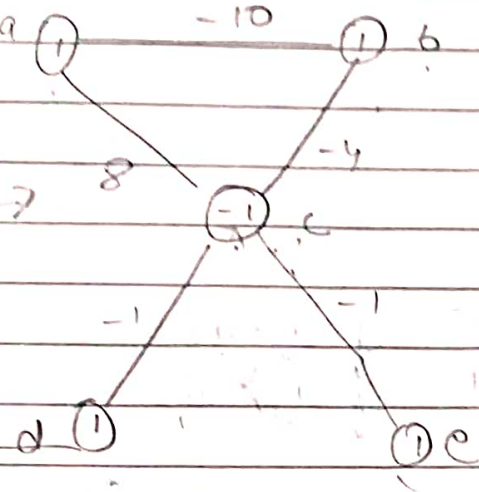
e: $-1 \times -1 \times 1 < 0$ × Bad



b: $-10 \times 1 \times 1 + -4 \times 1 \times 1$ ✓ (Good : (No change in state)

c: $8 \times 1 \times 1 + (-4) \times 1 \times 1 + (-1) \times 1 \times 1 + (-1 \times 1 \times 1)$

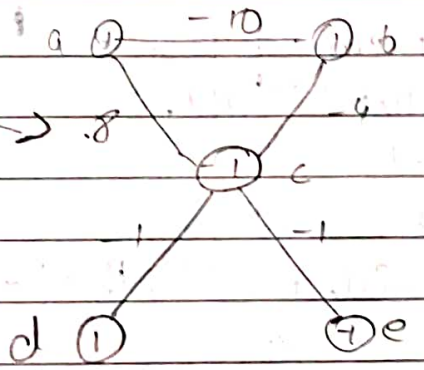
$6 = 8 - 4 - 1 - 1 = 2 > 0$ ✗ Bad : (Change in State)



as state of c will change then it will also effect the change in state of d, e, a, b as it is connected to all these Now we check for c again and then we go

Now for e:-

$1 \times -1 \times -1 > 0$ ✗ (Bad) : (Change in State)



We can check in any order a, b, c, d, e are a, c, e, a, b

The weight on the all the good edges must be greater than or equal to the weight of bad edges.

ND is Stastified in a given configuration. if the total absolute weight of all good edges incident to v is atleast as large as the weigth total absolute weight of all bad edges incident to v

Page No.: Date: YOUVA

Σ We $\delta_u \delta_v$ IO
V: e(v, w)

Randomised Algorithm:-

i) Median finding :- To find the k^{th} largest element from array

For $S = \{5, 4, 12, 6, 1, 9\}$ — 1, 4, 5, 6, 7, $\frac{n+1}{2} = 4$

1, 4, 5, 6, 7, 8 = $\frac{n}{2} = 11$

Select (S, K)

Choose Splitter $a_i \in S$: (at random) = 6 = a_i

ii) $a_i = 4$ — For each a_j in S

ii) $S^- = \{1\}$ — Put a_j in S^- if $a_j < a_i$ — i) $S^- = \{5, 4, 1\}$

ii) $S^+ = \{5\}$ — Put a_j in S^+ if $a_j > a_i$ — i) $S^+ = \{12, 9\}$

end for $3 \neq 3-1$ ✗ (we first find K)

$1 \neq 3-1$ ✗ — if $|S^-| = K-1$, — i) K = 3 (= n/2 in case of even)

a_i is the K^{th} largest element. — in odd we take ($K = \frac{n+1}{2}$)

$1 > 3$ ✗ — else if $|S^-| > K$, 3 > 3 ✓

the K^{th} largest element is in S^-

Recursively call Select (S^-, K) — i) Select ({5, 4, 1}, 3).

$1 < 3$ ✓ — else if $|S^-| < K-1$

the K^{th} element is in S^+

Recursively call Select (S^+, $K - 1 - |S^-|$).

End if. — ii) Select ({5}, 3 - 1 - 1)

= ({5}, 1)

In this case iii) Select ({5}, 1)

$a_j = 5$.

$S^- = \{\}$ $S^+ \{\}$

if $|S^-| = K-1$ => $|S^-| = 0 = K - 1 = 1 - 1 = 0$

0 = 0

∴ $a_i = 5$ is the K^{th} Largest A/O element

So the 3rd largest element is 5

(as we take K = 3 initially)

Complexity O(n)

For $S = \{5, 4, 12, 6, 1, 9\}$

i) we take

$a_j = 5$

$S^{-} = \{4, 1\}$

$S^{+} = \{12, 6, 9\}$

when we choose Median as a splitter. In this case (Median case)

$T(n) \le T(n/2) + cn$

$= O(n)$

NOTE:- when we choose Median as a splitter the time complexity $= O(n)$.

Case-I:- if we choose an splitter such that atleast ϵn elements in left ~~and $(1-\epsilon)n$ element in left~~ $\epsilon n > 0$ (and right)

if $\epsilon n = 1/4$ elements

ϵn — a_i — $(1-\epsilon)n$

$1/4$ — $3/4$ → atleast $1/4$ it can be greater then also.

$(1-\epsilon)n$ — a_i — $(\epsilon)n$

$3/4$ — $1/4$

$1/4$ → is the well centered splitter or good splitter.

and we have to atleast remove $(1-\epsilon)n$ elements in every iteration.

$T(n) \le \{cn + (1-\epsilon)n + (1-\epsilon)^2 n \; ---\}$.

$\le \frac{1}{\epsilon} cn$.

$\le O(n)$

an unordinum

Case-2 :- If we choose the minimum element as a Splitter it, is the worst case and. it is called aff-Center Splitter.

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

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

$T(n) = \frac{Cn(n+1)^2}{2}$.

Randamized quick Sort algorithm :- (using / with the help of Randomized algorithm).

```
QS(S)
  if |S| = 3
     Sort (S).
     O/P the Sorted list.        } → Base Case.
  else.
     Choose Splitter ai ∈ S uniformly at random.
  For each element aj of S.
     Put aj in S- if aj < ai
     Put aj in S+ if aj > ai
  End for.
  Recursively call QS(S-) & QS(S+).
  O/P the Sorted array of S- then ai, then
     the Sorted list S+.
  end if.
```

For Best Case $T(n) = T(n/2) + (n)$ Worst Case $O(n^2)$.

$= O(n \log n)$.

TO improve our Normalized Random quick Sort,

~~In this~~ we use Modified QS and in this we Choose Central Splitter

↳ it means there should be atleast $1/4^{th}$ element in S^- & S^+ ==

Continued.

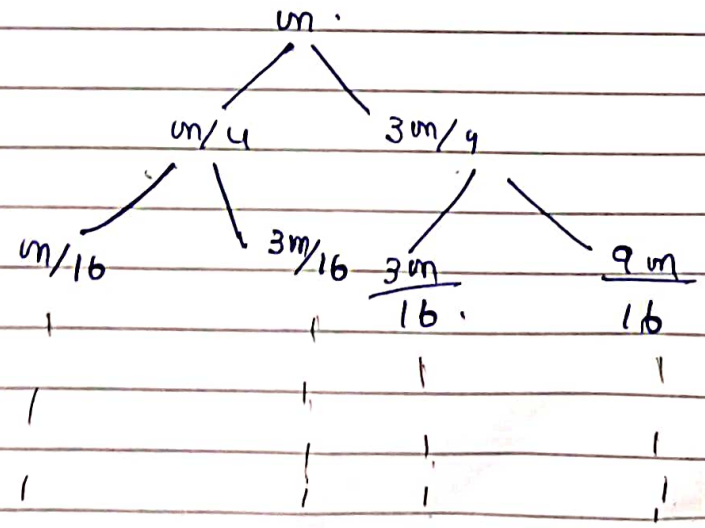
Page No.: Date: YOUVA

Modified quick Sort:-

MQS(S).
if |S| ≤ 3.
Sort S..
O/P the Sorted S.
Else
while (no Central Splitter has been found).
For each $a_j \in S$.
Put a_j in S^- if $a_j < a_i$
Put a_j in S^+ if $a_j > a_i$
end for.
if $|S^-| \geq \frac{|S|}{4}$ and $|S^+| \geq \frac{|S|}{4}$
a_i is the Central Splitter.
End if.
end while.
Recursively Call MQS(S^-) and MQS(S^+)
O/P Sorted S^-, a_i, Sorted S^+.

Central Splitter Condition

$$T(n) = T(n/4) + T(3n/4) + (n).$$



$$\frac{n}{(4/3)^K} = 1 \qquad K = \text{height of the tree.}$$

$$K = \log_{4/3} n$$

improved.

Complexity ~~is~~ of in modified quicksort $= O(n \log_{4/3} n)$.

× Complexity of normal quicksort $= O(n \log_2 n)$

If we choose median as a splitter time complexity $= O(n \log_2 n)$.

CH-1
NETWORK FLOW

- A flow graph (flow network) is a directed graph where each edge (also called an arc) has a certain capacity which can receive a certain amount of flow. The flow running through an edge must be less than or equal to the capacity.

- Each edge in the flow graph has a certain flow and capacity specified by the fraction adjacent to each edge. Initially the flow through each edge is 0 and the capacity is non-negative value.

- To find the maximum flow (and min-cut as a by product), the ford-fulkerson method repeatedly finds augmenting paths through the residual graph and augments the flow until no more augmenting paths can be found

Augmenting Paths :- An augmenting path is a path of edges in the residual graph with unused capacity greater than zero from the source s to the sink t.

Bottle Neck :-

Bottleneck = Capacity - Current flow of edge.

The bottle neck is the smallest edge on the path. We can use the bottle neck value to augment the flow along the path.

- Augmenting the flow means updating the flow values of the edges along the augmenting path.

For forward edges, this means increasing the flow by the bottleneck value.

- When augmenting the flow along the augmenting path, you also need to decrease the flow along each residual edge by the bottleneck value.

- The time complexity of the ford-fulkerson method depends on the algorithm being used to find the augmenting paths, which is left unspecified.
  or $O(fE)$.