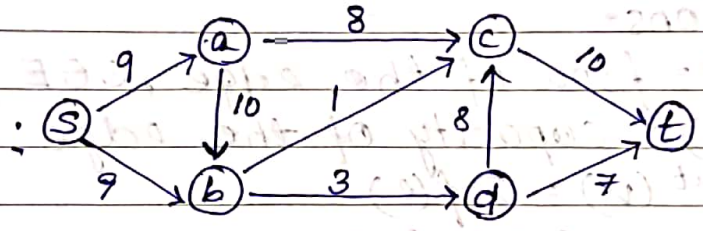
(Ch1)

Network Flow

17/03/23

* Maximum-Flow Problem-
A flow network is a directed graph $G(V, E)$ with the following features:
① Edge Capacity - For each edge $e=(u,v) \in E$, edge capacity denoted by $C(e)$
② There is a single source node $s \in V$
③ There is a single sink node $t \in V$
Other than s,t all are called internal node



Assumptions:
① No edge enters the source s
② No edge leaves the sink t
③ All capacity are +ve Integers

* Defining Flow- An s→t flow / s-t flow is a function f that maps each edge $e \in E$ to a non-negative real number $f: E \to \mathbb{R}^+$, then the value $f(e)$ represents the flow carried by edge $e \in E$.

* Properties- A flow f must satisfy the following properties:
① Capacity Condition
→ for each edge $e \in E$
$0 \leq f(e) \leq C(e)$

② Conservation Condition

for each internal node $v \in V$, other than source s and sink t, we have, $f^{in}(v) = f^{out}(v)$, i.e., inflow = outflow

* Notations-

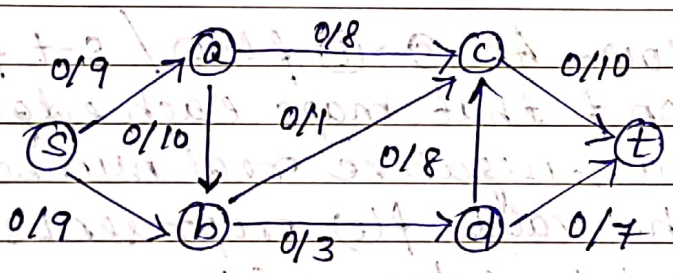
① $f(e)$ = flow of the edge, $e \in E$

② $c(e)$ = capacity of the edge:, $e \in E$

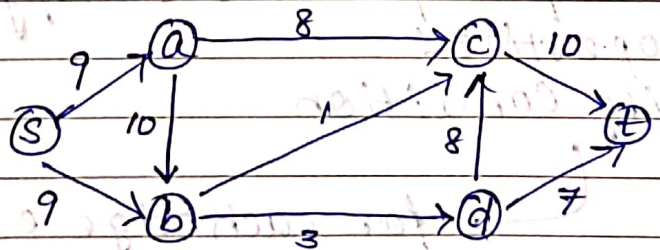
③ $f^{out}(v) = \sum_{e \text{ out of } v} f(e)$

④ $f^{in}(v) = \sum_{e \text{ into } v} f(e)$

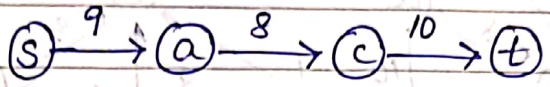
Original Graph -



Residual Graph -

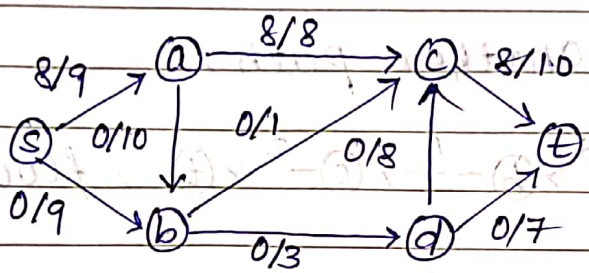


We have to choose a path from s→t, from the *residual graph*.

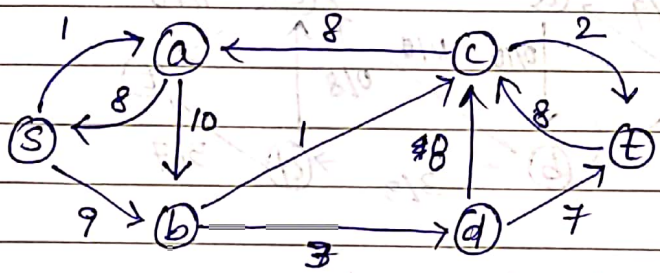


, flow = 8

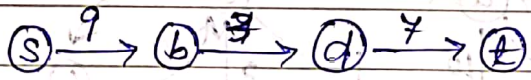
Updating Original Graph –



New residual graph –

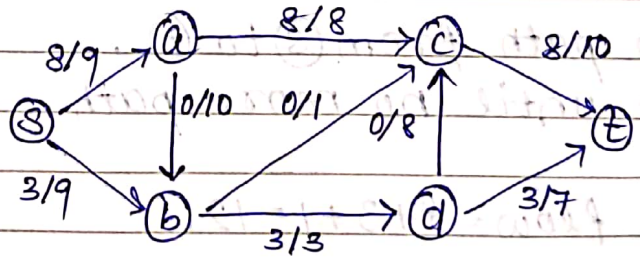


Selecting another path,

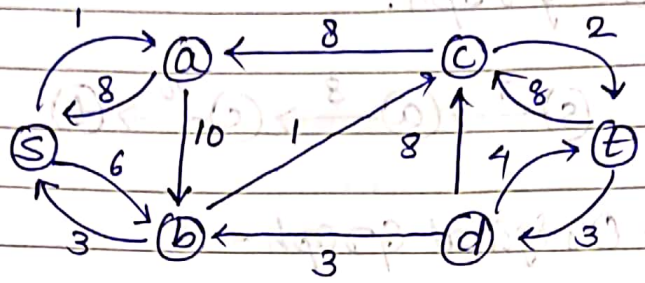


, flow = 3

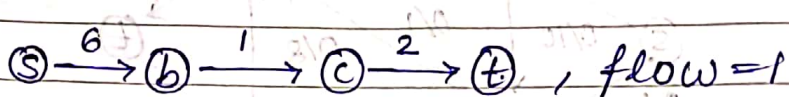
Updating Original Graph –



New residual graph,

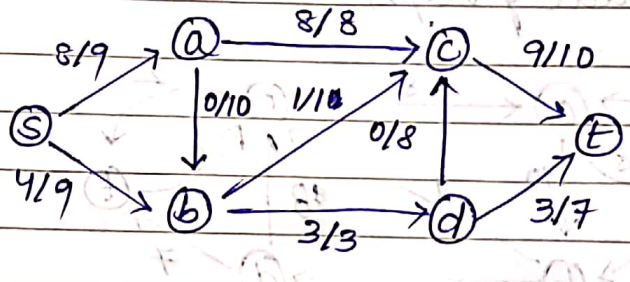


Selecting another path,

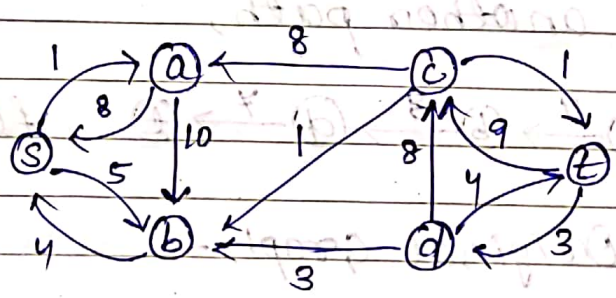


, flow = 1

Updating original graph,



New residual graph,



No more path from s to t.
Solve until no more path.

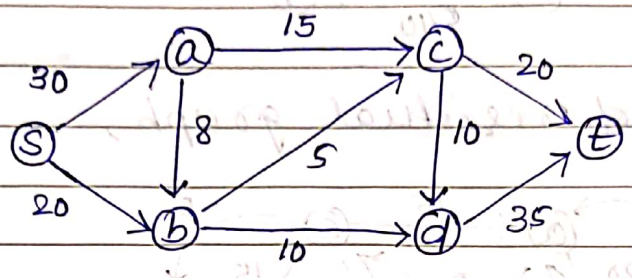
Total flow = 8 + 3 + 1 = 12.

Value of a flow, $v(f) = f^{out}(s) = 8+4$

$\Rightarrow v(f) = 12$

20/03/23

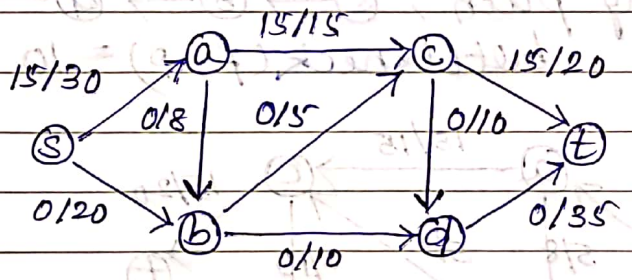
Q) Given residual graph, find the maxflow



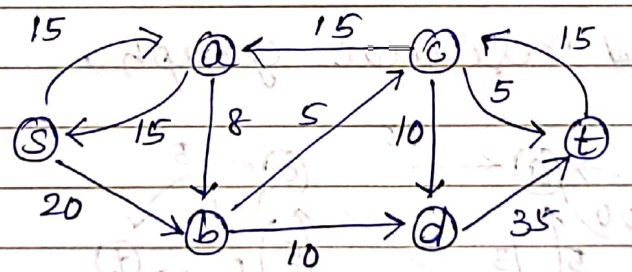
Selecting path, $s \xrightarrow{30} a \xrightarrow{15} c \xrightarrow{20} t$

flow, bottleneck(f, p) = 15

Original graph becomes,



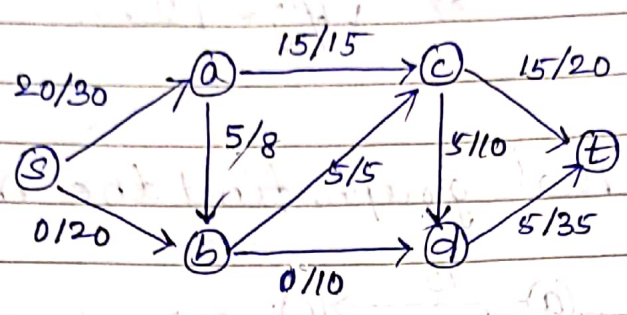
Updated Residual Graph,



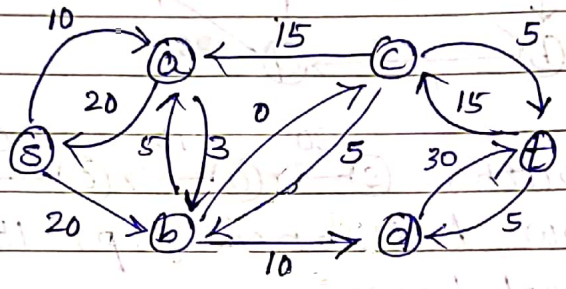
Selecting path, $s \xrightarrow{15} a \xrightarrow{8} b \xrightarrow{5} c \xrightarrow{10} t \xrightarrow{35} t$

bottleneck(f, P) = 5

Original Graph becomes,

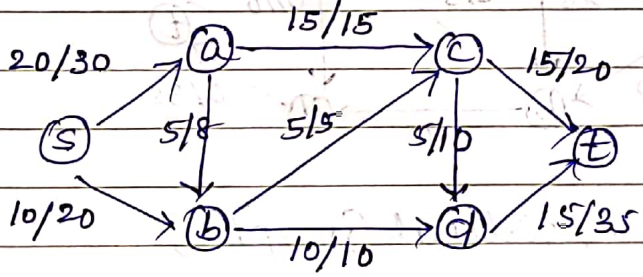


Updated residual graph,

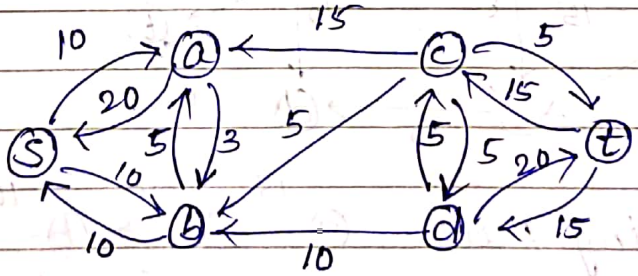


Selecting path, $s \xrightarrow{20} b \xrightarrow{10} d \xrightarrow{30} t$

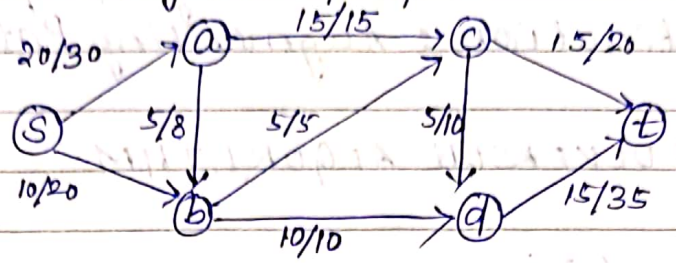
bottleneck(f, p) = 10



Updated residual graph,



Final Original Graph –



Maxflow = $f^{out}(s) = 20+10 = 30$.

* A path in the residual graph from source to destination : Augmenting Path

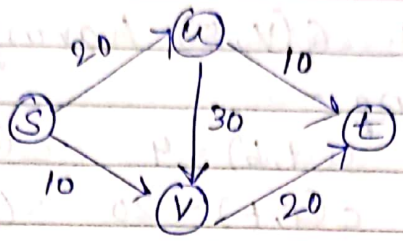
FORD-FULKERSON ALGORITHM-

```
maxFlow(){
    initially f(e)=0 for all e in G,
    while there is an augmenting path
    s-t in the residual graph Gf
            f'= augment(f,P),
            update f to be f',
            update the residual graph Gf
            to be Gf',
    end while
    return(f),
}
```

path
flow

```
augment(f,P){
    let b= bottleneck(P,f),
    for each edge(u,v)∈P
        if e=(u,v) is a forward edge
        then
            f'(e)=f_prev(e)+b,
        else
            (u,v) is a backward edge,
            f'(e)=f_prev(e)-b,
        end if
    end for
    return(f),
}
```

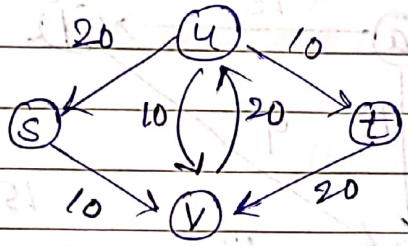
Q)



Selecting path $s \xrightarrow{20} u \xrightarrow{30} v \xrightarrow{20} t$

bottleneck(f, P) = 20

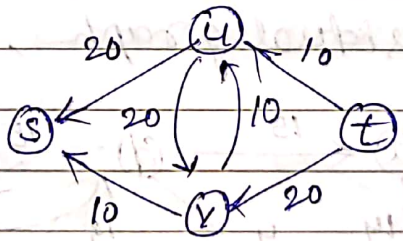
Residual graph:



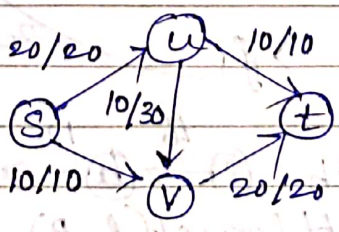
Selecting path $s \xrightarrow{10} v \xrightarrow{20} u \xrightarrow{10} t$

bottleneck(f, P) = 10

Residual graph:



Final Original Graph,



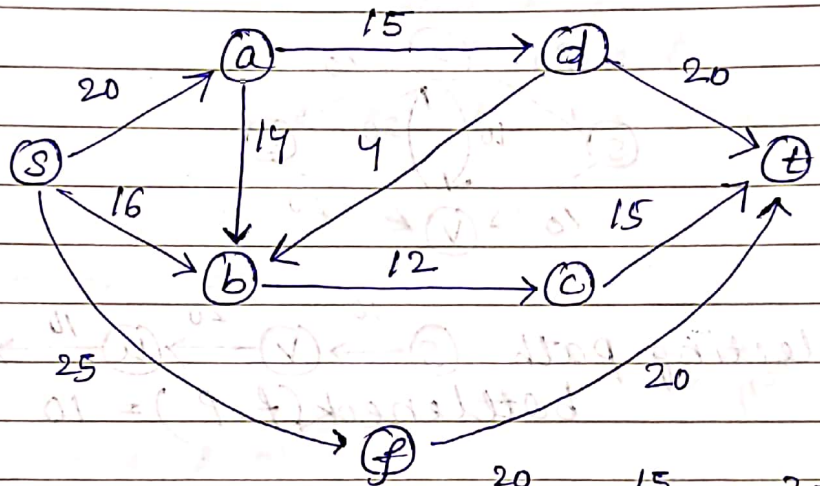
Maxflow;

$f^{out}(s) = 10 + 20 = 30$

Q) Let a graph G(V, E) having all the capacities as following -

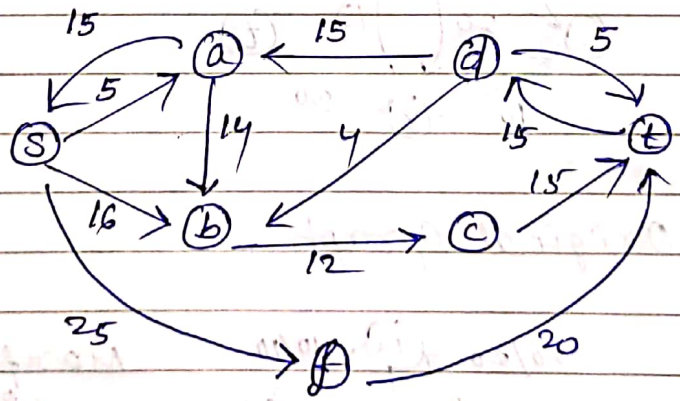
c(a,d) = 15 c(d,b) = 4 c(b,c) = 12 c(a,b) = 14
c(s,b) = 16 c(d,t) = 20 c(s,a) = 20 c(c,t) = 15
c(s,f) = 25 c(f,t) = 20

c(u,v) denotes the capacity of the edge (u,v) calculate maxflow and draw all the residual graphs.



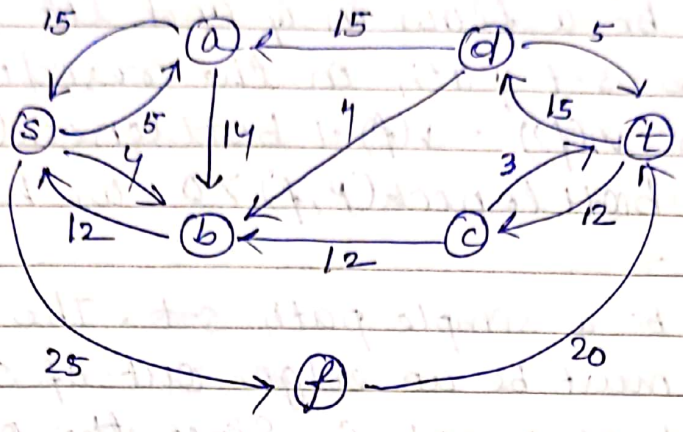
Selecting path, s → a → d → t (20, 15, 20)
bottleneck(f,P) = 15

Updated residual graph,



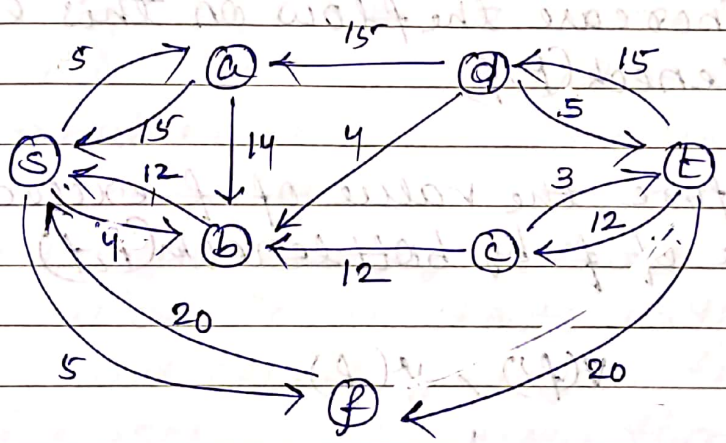
Selecting path s → b → c → t (16, 12, 15)
bottleneck(f,P) = 12

Updated residual graph,

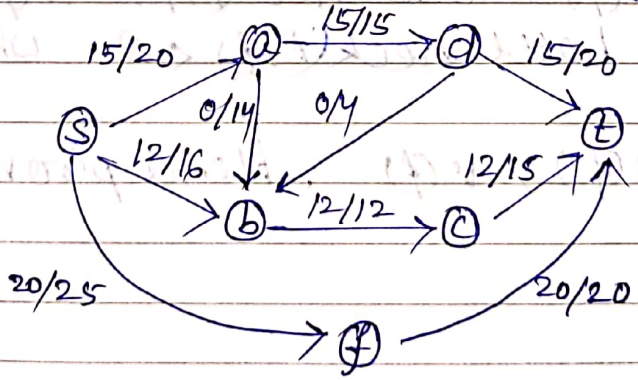


Selecting path s → f → t 25 20

bottleneck(f, P) = 20



Original Graph becomes,



Maxflow, $f^{out}(s) = 15 + 12 + 20 = 47$

21/03/23

Corollary 1

- Let f be a flow in G and let P be a simple s-t path in the residual graph G_f, then $v(f') = v(f) + \text{bottleneck}(P, f)$ and since $\text{bottleneck}(P, f) > 0$, we have $v(f') > v(f)$

Proof -

Let P be a simple path s-t. The first edge e of P must be an edge out of s in the residual graph G_f. Since the original graph G has no edges entering s, the edge e must be a forward edge. So, we increase the flow on this edge by $\text{bottleneck}(P, f)$

Therefore, the value of f exceeds the value of f by $\text{bottleneck}(P, f)$

$$v(f') > v(f)$$

↓ If false then

$$v(f') < v(f)$$

$$\Rightarrow v(f) + \text{bottleneck}(P, f) < v(f)$$

$$\Rightarrow \text{bottleneck}(P, f) < 0. \text{ which is false}$$

So, $v(f') > v(f)$. Hence proved.

* Corollary 2

Observation - If all the edges out from the source s is completely saturated with the flow f and let $f^{out}(s) = \sum_{e \text{ out of } s} f(e) = c$

- Let all the capacities in the flow network G are integers. Then the Ford-Fulkerson Algorithm terminates in atmost c iterations of the while loop

Proof - We have $v(f) \leq c$ because of the capacity condition all the leaving the source s. Now $v(f)$ maintained by the Ford Fulkerson algorithm increased in each iteration.

Let it increase by atleast one in each iteration. So, it starts with the value zero and cannot go higher than c, the while loop in the algorithm can run for atmost c times.

* ~~Corollary 3~~

Observation - If all the edges out from the source s is completely saturated with the flow f and let

$$f^{out}(s) = \sum_{e \text{ out of } s} f(e) = C$$

* Corollary 3 - T.C. of Ford Fulkerson Algorithm

- Let all the capacities in the flow network G are integers. Then the Ford-Fulkerson algorithm can be implemented to run in $O(mC)$ times.

Proof -

While loop of the Ford Fulkerson algorithm terminates in atmost C times. Let the original graph G has 'm' no. of edges and 'n' no. of vertices. To find an augmenting path (s-t) in the residue graph G_f we use BFS/DFS. So for finding the path, time complexity is $O(m+n) \approx O(m)$ because $m > n$

So, therefore, total time for 'C' no. of iterations is $= O(mC)$

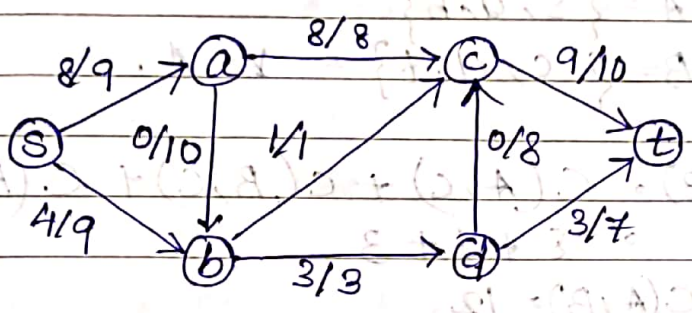
22/03/23

Page 15

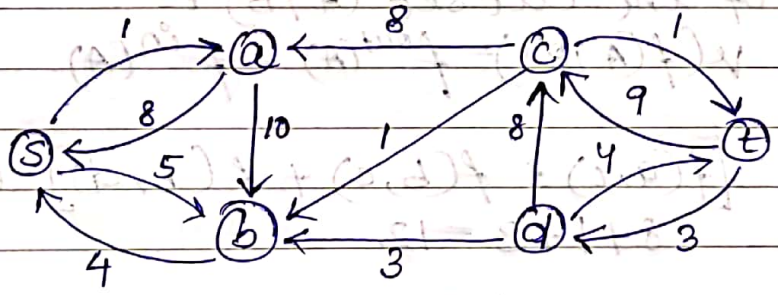
Maximum Flow and Minimum Cut.

Cut Set - Let $G(V, E)$ be graph where V is a set of vertices and E is a set of edges. An $(s-t)$ cut is a partition (A, B) of the vertex set V such that $s \in A$ and $t \in B$, such that $V = A \cup B$ and $A \cap B = \phi$

Eg. 1



$$f^{out}(s) = f(s,a) + f(s,b) = 8+4 = 12$$



Notation - The capacity of cut (a, b) is denoted by $c(A, B)$ and is defined by

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

From the graph,

$V = \{s, a, b, c, d, t\}$

$A = \{s, a, b\}$ $\quad B = \{c, d, t\}$

$V = A \cup B$, and $A \cap B = \phi$

$A = \{ v \in V$ such that, v is reachable from the source node 's' in residual graph $G_f \}$

→ from source, vertex 'a' and 'b' are directly rechable.

$A = \{ s, a, b \}$

$B = \{ c, d, t \} = V - A.$

$\therefore C(A,B) = C(A,c) + c(B,C) + C(B,D)$

$= 8 + 1 + 3$

$\Rightarrow C(A,B) = 12$

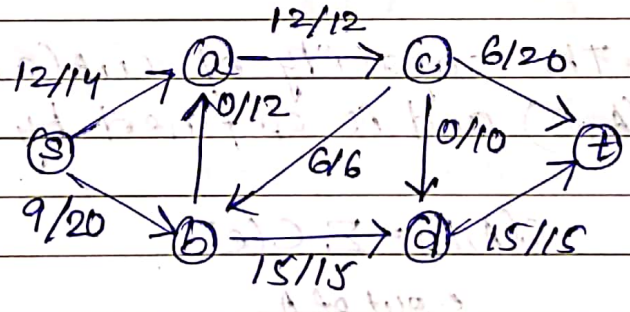
flow of the cutset (A,B) is

$v(f(A,B)) = f^{out}(A) - f^{in}(A)$

$= [f(a,c) + f(b,c) + f(b,d)] - 0$

$= 8 + 1 + 3 = 12$

Eg.2:



Calculate C(A,B) and v(f(A,B)) of the cutset (A,B) is - $A = \{ s, a, b \}$

$B = \{ c, d, t \}$

Capacity of cut set -

$$c(A,B) = c(a,c) + c(b,d)$$
$$= 12 + 15 = 27$$

$$v(f(A,B)) = 27 - 6 = 21$$

$[f(a,c) + f(b,d)]$ — A to B

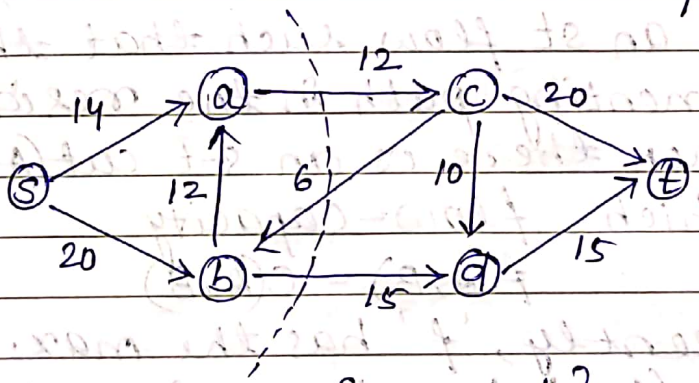
$[f(c,b)]$ — B to A

24/03/23

Notation – Let (A, B) be a cut of G, then

① Capacity of the cutset $= c(A,B) = \sum_{e \text{ out of } A} c(e)$

② Flow of the cutset : $f(A,B) = f^{out}(A) - f^{in}(A)$
$= \sum f(v) - \sum f(e)$



Suppose, $A = \{s, a, b\}$
$B = \{c, d, t\}$

A, B is a cutset
$A \cup B = V$
$A \cap B = \phi$

$$c(A,B) = c(a,c) + c(b,d)$$
$$= 12 + 15$$
$$= 27$$

$$f(A,B) = 12 + 15 - 6 = 21$$

* Corollary - Let f be any s-t flow and (A,B) be any s-t cut set. Then:

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

We know, $f(A,B) = f^{out}(A) - f^{in}(A) \leq c(A,B)$

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

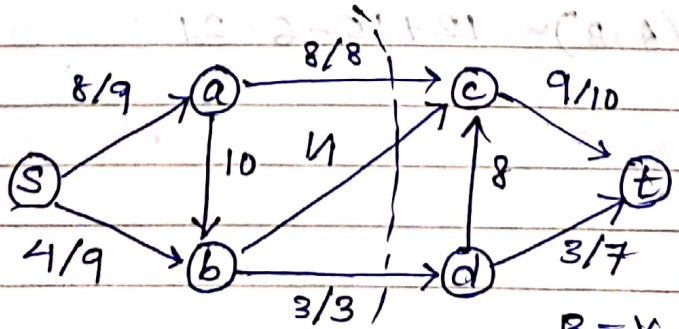
* Maxflow Minimum Cut Theorum -
If f is an st flow such that there is no augmenting path in the residual graph G_f, then there is an s-t cut (A,B) in G for which flow = capacity

$$f(A,B) = c(A,B)$$

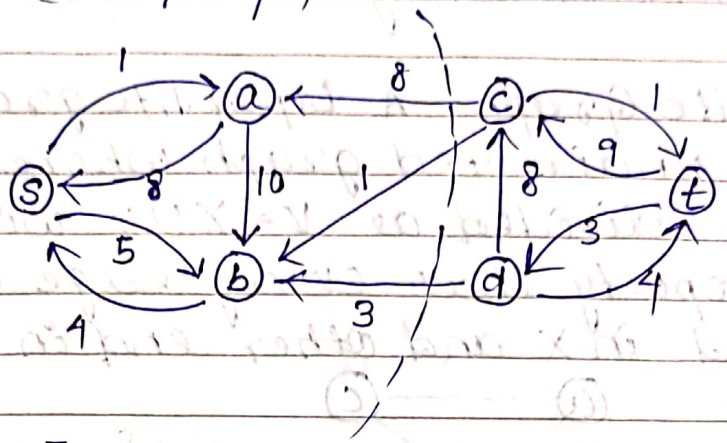
Consequently, f has the max. value in G, then (A,B) has the min. capacity

* capacity of min cut is always equal to the maxflow.
* min. of all cut is equal to the max flow

Proof: To prove, $f(A,B) = c(A,B)$
given there is no augmenting path in G_f.
Consider $A = \{v; v$ is reachable from the source 's' in residual graph$\}$



Residual Graph :

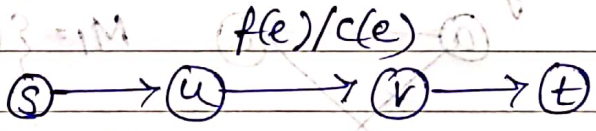


Case 1 : For forward edge $A \to B$

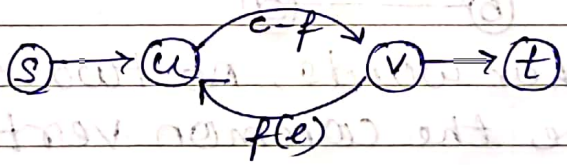
Suppose (u, v) is an edge where $u \in A, v \in B$ then, $f(e) = c(e)$

If not then, $f(e) < c(e)$ then v is reachable from 's' in G_f which contradicts the assumption.

Original :



Residual : when $f(e) < c(e)$



Case 2 : Backward edge theorum -

Let $u \in A$ and $v \in B$, (v, u) is a backward edge then $f(e) = 0$. If not, then $f(e) > 0$ then $v \in A$ [contradicts the assumption]

$$f(A, B) = \sum_{e \text{ out of } A} f(e) - \sum_{e \text{ into } A} f(e) = \sum_{e \text{ out of } A} c(e) - \sum 0$$

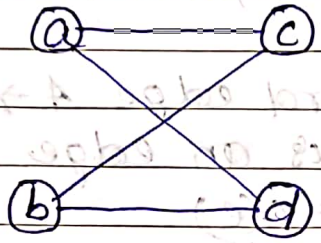
$$\Rightarrow f(A, B) = \sum_{e \text{ out of } A} c(e) = c(A, B) \quad \text{[Proved]}$$

27/03/23

* Bipartite Matching:

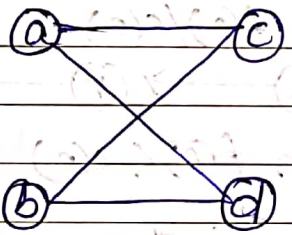
- Bipartite Graph - A bipartite graph $G(V, E)$ is an undirected graph where set V can be divided as $V = X \cup Y$ with the property that every edge $e \in E$ has one end in X and other end in Y.

eg.-



$X = \{a, b\}$
$Y = \{c, d\}$

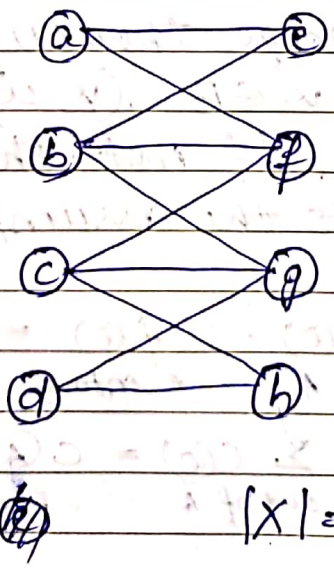
- Matching - A matching $M \subseteq E$ in G is such that each node v appears in atmost one edge in M.



$M_1 = \{(a, c), (b, d)\}$

$M_2 = \{(a, b), (c, d)\}$

In other words, no two edges of M share the common vertex.



$M_1 = \{(a, e), (b, f), (c, g), (d, h)\}$

$M_2 = \{(a, f), (b, g), (c, h)\}$

$M_3 = \{(b, e), (c, f), (d, g)\}$

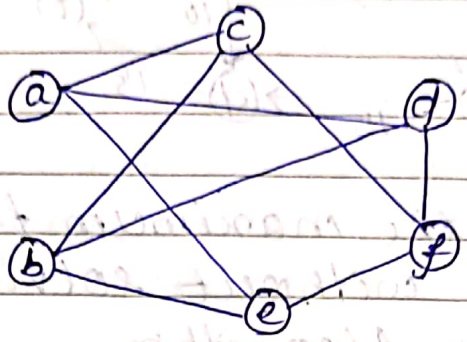
$|M_1| = 4, |M_2| = 3, |M_3| = 3$

$|X| = |Y| = |M_1| = 4$

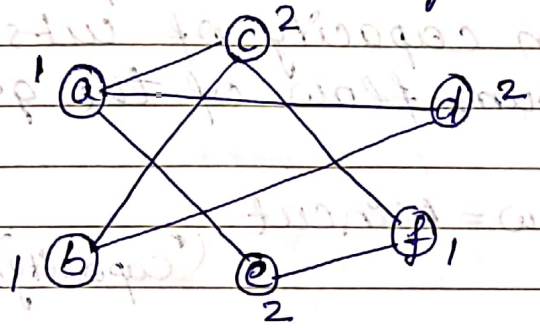
Perfect Matching

If $|X| = |Y| = |M|$ then M is called perfect matching

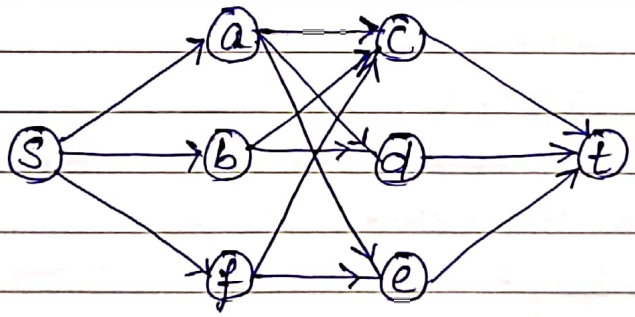
Q)



Is this graph a bipartite graph? If it is bipartite then find the matching using Ford Fulkerson Algorithm.

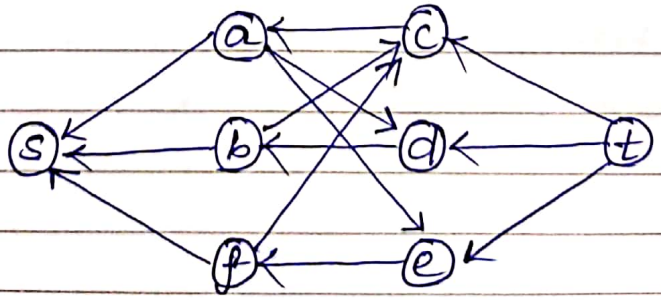


$X = \{a, b, f\}$
$Y = \{c, d, e\}$



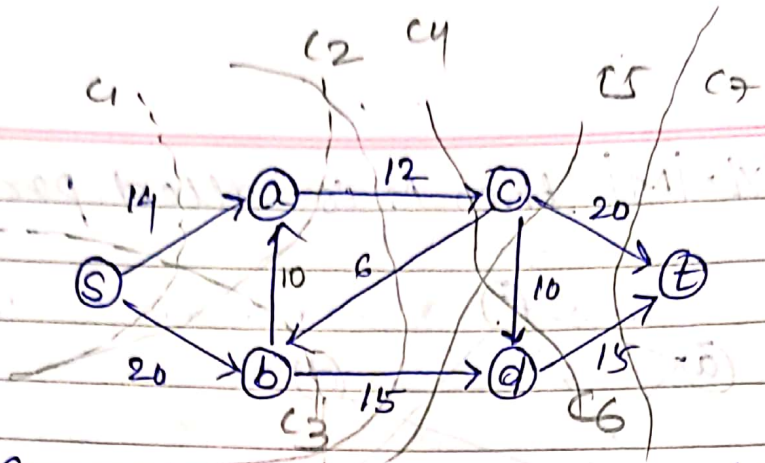
$s \to a \to c \to t$
$s \to b \to d \to t$
$s \to f \to e \to t$

Residual Graph -



$c \to a$
$d \to b$
$e \to f$
} back flow

Q)



Calculate the maximum flow of the network G without solving Ford-Fulkerson Algorithm.

Take all the possible cutset of graph G and write all their capacities The minimum capacity of cutset is the maximum flow of the graph

$$\text{maxflow} = \text{mincut} = \text{minimum}(\text{capacity})$$

$C_1(S) = 14+20 = 34$

$C_2(S,a) = 12+20 = 32$

$C_3(S,b) = 14+15+10 = 39$

$C_4(S,a,b) = 12+15 = 27$

$C_5(S,a,c,b) = 20+10+15 = 45$

$C_6(S,a,c,b,d) = 12+15 = 27$

$C_7(S,a,b,c,d) = 20+15 = 27$

mincut

NP and Computational Intertact

29/03/23

* **Problem Reduction –**
Notation: $Y \leq_p X$ Y is polynomial time reducible to X

* Simple Reduction – To solve problem Y, we reduce the instances of Y to an instances of X then solve the problem X. Then the solution of X is also the solution of Y.
Then total time complexity to solve Y is equal to: reduction time + T.C. of X

eg 1:

Problem Y	Problem X
Input – 2 integers X & Y	Input – 2 integers X & Y
Output – LCM of (X & Y)	Output – GCD of (X, Y)

Reduction:-

```
LCM(x, y) {
    take x, y
    return xy / GCD(X,Y)
}
```

LCM $\leq_p$ GCD

* Problem Y is reduced wrt X.
Formula: $LCM(x,Y) \times GCD(X,Y) = XY$

eg2: Closest Pair Problem
Find the pair of numbers within a set that have the smallest difference between them.

Input – A set S of n numbers, and threshold t.
Output – Is there a pair $s_i, s_j \in S$ such that $|s_i - s_j| \leq t$?

Reduction –
closeEnoughPair (S, t)
① sort(S)
is min $(s_{i+1} - s_i) \leq t$:
print (s_i, s_{i+1})

T.C. = $O(n \log n)$

{closestPair $\leq_p$ Sorting}

* Convex Hull
Input – A set S of n points in plane
Output – Find the smallest convex polygon containing all points of S.
T.C. = $O(n^2)$

Reduction –
① sort(S)
② for each $i \in S$, create point (i, i^2)
③ Call subroutine ConvexHull on this pointset
④ From the leftmost point in the hull join with the rightmost point

T.C. = $O(n \log n)$

{Sorting $\leq_p$ Convex Hull}

31/03/23

AD2 · Quiz 1

① Consider the following network flow $G(V,E)$, where $V = \{s, a, b, c, d, t\}$ and capacity of the edges are : $c(s,a) = 18$, $c(s,d) = 17$, $c(s,c) = 14$, $c(a,b) = 12$, $c(a,c) = 13$, $c(b,t) = 14$, $c(c,d) = 12$, $c(c,t) = 15$ $c(d,t) = 19$.

ⓐ List all the simple s-t augmented paths to find the maxflow of G.

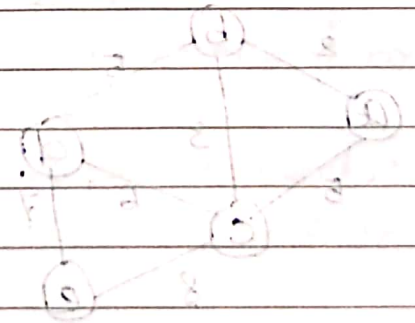
ⓑ Show each residual graph

ⓒ Find a minimum cut set

② ⓐ Give the equations/inequalities for capacity constraints and flow conservation in the flow network G.

ⓑ Construct a graph G with 2n vertices and n^2 edges such that G has exactly one unique perfect matching

ⓒ Let f be a flow and (A,B) is cutset, then prove that $f(A,B) \leq C(A,B)$ where $C(A,B)$ is the capacity of the cut.

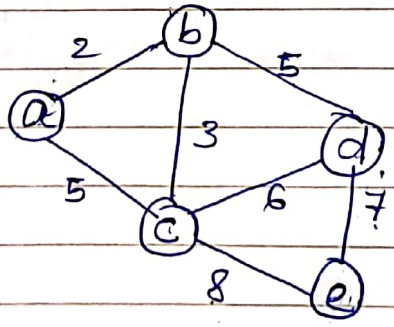


03/04/22

* Decision Problem
The simplest interesting class of problems have answer restricted to true & false. These are called decision problem.
eg:
Problem - The travelling salesman decision problem. (also complete graph = no. of tours on distinct)
Input - A weighted graph G and integer k
Output - Does there exist a TSP tour with cost ≤ k?

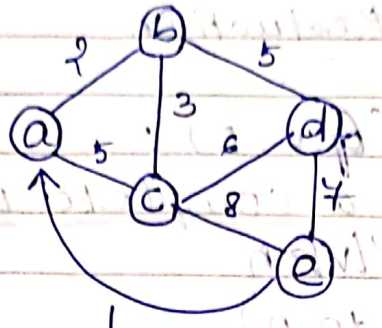
* Travelling Salesman Problem - Traverse all the vertices exactly one and come back to the source/start with minimum cost.

{Hamiltonian Cycle - A cycle that visits every vertex of an (undirected) unweighted graph G without repetition.}



TSP: $a \xrightarrow{2} b \xrightarrow{5} d \xrightarrow{7} e \xrightarrow{8} c \xrightarrow{5} a$, cost = 2+5+7+8+5
∴ cost = 26

* let k = 30, cost ≤ k
(26 ≤ 30)
26



$a \xrightarrow{2} b \xrightarrow{3} c \xrightarrow{6} d \xrightarrow{7} e \xrightarrow{1} a$, cost = 2+3+6+7+1 = 19

* let k = 25, cost ≤ k
(19 ≤ 25)

* Optimization Problem -

Problem - The Travelling Saleman Optimization Problem

Input - A weighted graph G

Output - which tour, $\{v_1, v_2, \dots v_n\}$

$$\text{minimize} = \sum_{i=1}^{n-1} c(v_i, v_{i+1}) + c(v_n, v_1)$$

* Hardness Reduction :-

Problem Y : Hamiltonian Cycle

Input : An unweighted graph G (Undirected)

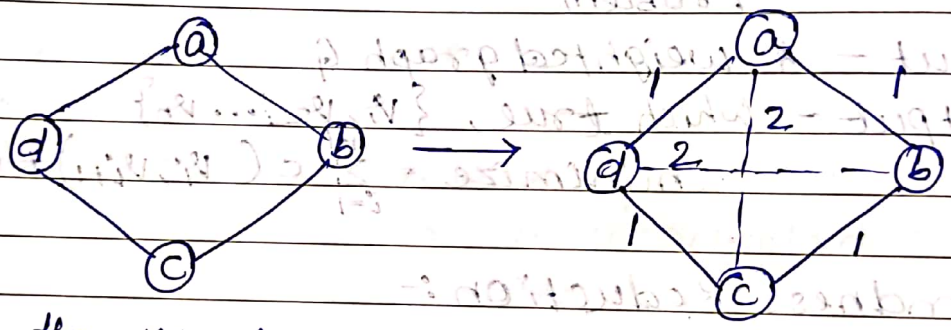
Output : Does there exist a simple tour that visits each vertex of G without repetition

Problem X = TSP

* Reducibility / Reduction

Hamiltonian (G, V, E) :

① construct a complete weighted graph
② |V| = n
③ for (i = 1 to n)
④ for (j = 1 to n)
⑤ if (i, j) ∈ E
⑥ weight (i, j) = 1
⑦ else
⑧ weight (i, j) = 2 (or anything more than 1)
⑨ return the answer to TSP decision problem (G', n)



Hamiltonian Cycle

TSP Problem (Complete and weighted)

for this graph; cost = 4
if cost == |V|, return true

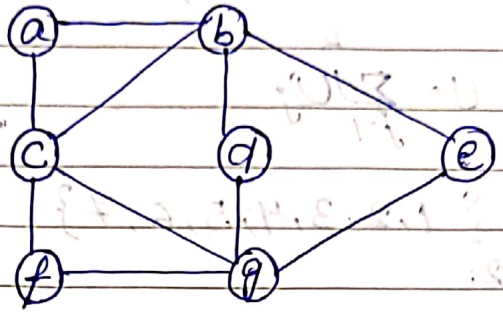
* if the edge ∈ E then weight = 1 or else 2. (ALWAYS)

T.C = O(n²)

{Hamiltonian Cycle ≤p TSP}

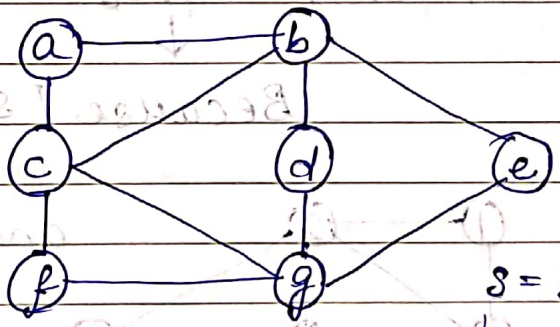
04/04/23

* Independent Set – It is a set of vertices $S \subseteq V$ of a graph $G(V,E)$ such that for any pair of vertices of S have no same edge.



$S = \{a, d, f, e\}$
$S = \{a, g\}$

* Vertex Cover Set → It is a set of vertices $S \subseteq V$ of a graph $G(V,E)$ such that for every edge $e \in E$ contains atleast one end vertex in S.



b, c, g

$S = \{c, b, g\}$
$S = \{c, a, d, e, g\}$

Reduction:

vertex cover (G, k)
$G' = G$
$k' = |V| - k$
return the answer to independent set (G', k')

{Vertex Cover $\leq_p$ Independent Set}

05/04/23

* Set cover - Given a set U of n elements, a collection $S_1, S_2, S_3, \ldots S_n$ of subsets of U and a number k. Does there exist a collection of atmost k of these set whose union is equal to U

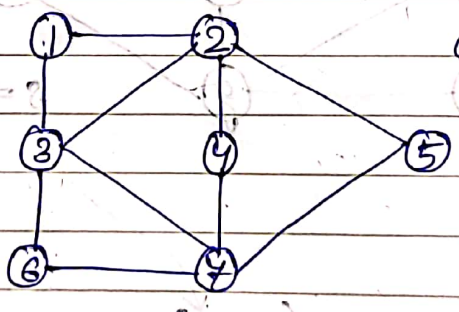
$$U = \bigcup_{j=1}^{k} S_j \qquad S_j \in \{S_1, S_2, \ldots S_n\}$$

$U = \{1, 2, 3, 4, 5, 6, 7\}$
$k = 2$

$S_1 = \{3, 7\}$ $S_4 = \{3, 4, 5, 6\}$
$S_2 = \{1\}$ $S_5 = \{2, 4\}$
$S_3 = \{5\}$ $S_6 = \{1, 2, 6, 7\}$

Set Cover = S_4, S_6

Because $[S_4 \cup S_6 = U]$

eg: 

and $k = 3$

Minimum vertex cover = {2, 3, 7}

$E = \{(1,2), (1,3), (2,3), (2,4), (2,5), (3,6), (3,7), (4,7), (5,7), (6,7)\}$

$|E| = 10$

$S_1 = \{(1,2), (1,3)\}$
$S_2 = \{(2,1), (2,3), (2,4), (2,5)\}$
$S_3 = \{(1,3), (2,3), (3,6), (3,7)\}$
$S_4 = \{(2,4), (4,7)\}$
$S_5 = \{(2,5), (5,7)\}$
$S_6 = \{(3,6), (6,7)\}$
$S_7 = \{(3,7), (4,7), (5,7), (6,7)\}$

$\underbrace{S_2 \cup S_3 \cup S_7}_{\text{same as minimum vertex cover}} = E$ (2, 3, 7 = Vertex Cover) [Proved]

Set Cover $\leq_p$ Vertex Cover (reduction) ↓

* Reduction :-

① Create setcover Instance
$k = k$, $U = E$, $S_v = \{e \in E : e \text{ incident to } v\}$

② setcover of size $\leq k$ iff vertex cover of size $\leq k$

* Satisfiability (SAT)
Let X be a set of 'n' boolean variables $x_1, x_2, \ldots x_n$ and instances of SAT is boolean formula.
ϕ composed of
1. n boolean variables $x_1, x_2, \ldots x_n$
(x_1, x_2 are called literals)
2. m boolean connections: any boolean function with one or more inputs & one output

Note: A boolean function ϕ is called satisfiable if the output of $\phi = 1$ for any truth assignment.

$\vee$ = OR
$\wedge$ = and

$\phi(x_1, x_2, x_3) = \underbrace{(x_1 \vee \bar{x}_2)}_{c_1} \wedge \underbrace{(\bar{x}_1 \vee \bar{x}_3)}_{c_2} \wedge \underbrace{(x_2 \vee \bar{x}_3)}_{c_3}$
clause

Truth Assignment are -

x_1	x_2	x_3
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

here c_1, c_2, c_3 are called clauses

conjunction = and
disjunction = or

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

$x_1 = 0, x_2 = 1, x_3 = 1, x_4 = 0$, value of ϕ

$(x_1 \vee \bar{x}_3 \vee x_4) = 0 \vee \bar{1} \vee 0 = 0$.

So, $\phi(x) = 0$

07/04/23

eg. $\phi(x_1, x_2, x_3) = (x_1 \vee \bar{x}_2) \wedge (\bar{x}_1 \vee \bar{x}_2) \wedge (x_2 \vee \bar{x}_3)$

Is ϕ satisfying for $x_1 = x_2 = x_3 = 1$

$c_1 = 1, c_2 = 0, c_3 = 1, c_1 \wedge c_2 \wedge c_3 = 0$

~~Date~~

3-SAT - A 3 SAT formula/function contain exactly 3 term/literals in each clauses of cf.

* n-SAT contains exactly 'n' literals in all the clauses.

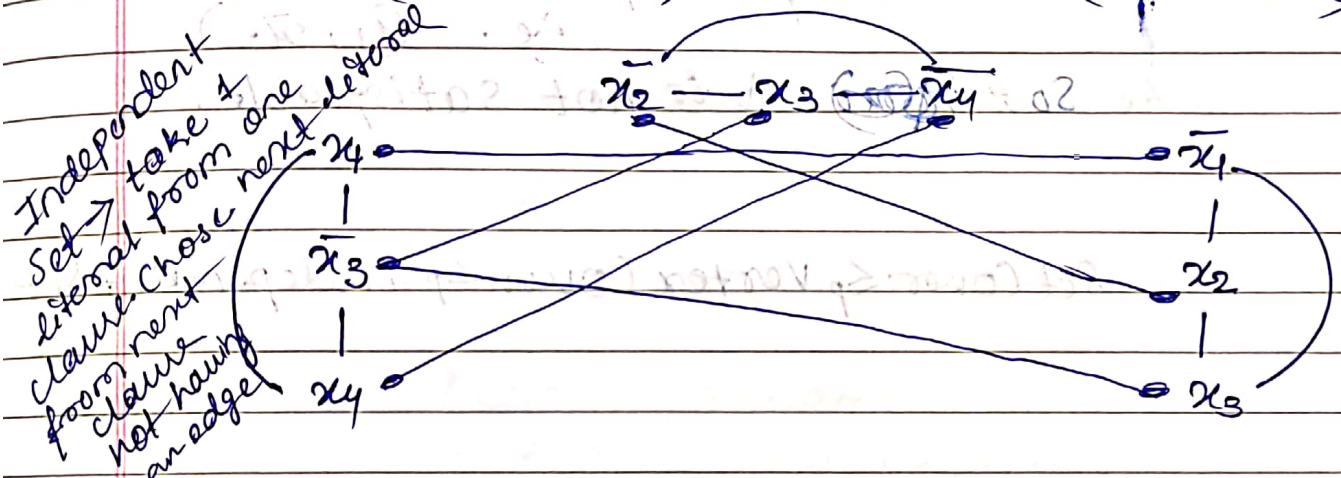
(Polynomial Time Reducible)

3 SAT $\leq_p$ Independent Set

Reduction-

1. Create a vertex for each literal
2. Connect each literal to each other : two literals in the same clause
3. Connect each literal, x_i to $\bar{x_i}$.

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



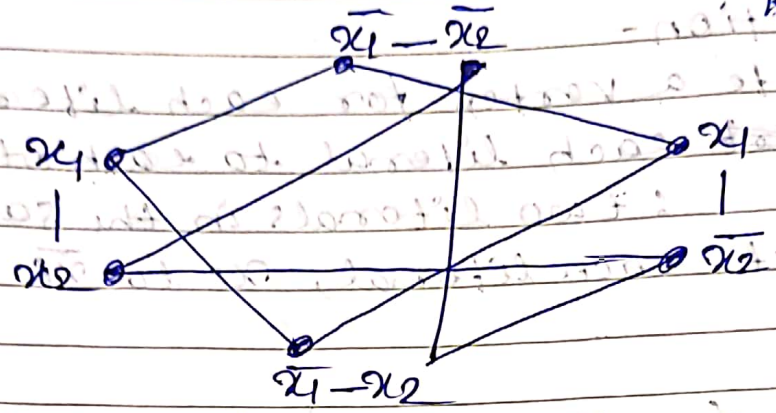
* Theorem

Let 'n' be the number of variables in ϕ and 'm' be the number of boolean clauses, then ϕ can be satisfied iff the graph has an independent set of size 'm'.

✓ nSAT is satisfiable iff there exist an independent set of size equal to the no. of clauses present in nSAT.

✓ 3 SAT - Every clause should have 3 literals.

eg $\phi(x) = (x_1 \vee x_2) \wedge (\bar{x_1} \vee \bar{x_2}) \wedge (x_1 \vee \bar{x_2}) \wedge (\bar{x_1} \vee$



$k = 4$ (no. of clause

Maxm. size of independent set = 2
i.e. $(x_1, \bar{x_2})$,

As $2 \neq k$, ϕ is not satisfiable.

Set Cover $\leq_p$ Vertex Cover $\leq_p$ Independent Se

10/04/23

Efficient Certification and definition of NP.

Computation Intractability consists 2 ingredients:

① Reducibility study
② Study class of problems.

$P = \{ X : \text{Set of all problems } X \text{ for which there exists an algo } A \text{ with a polynomial running time that solves } X \}$

$2^n, n^2, n^6$
$n \log n$

$P = \{ \text{sorting problems, GCD, Merge Problem} \}$

In other words -

$\{ X : X \text{ is solvable in polynomial time} \}$

$2^n, n^2$

$NP \{ X : \text{Decision problems with poly size certificates and polytime certifier for YES inputs.} \}$

not solvable in polynomial time but decision will be true in polynomial time

In other words, NP to be the set of all problems for which there exists an efficient certifier

- To understand Certificate and certifier
  ① 3-SAT Problem - The certificate "t" is an assignment of truth values to the variables.; the certifier evaluates the given set of clauses w.r.t this assignment.

if 'n' comes in power generally NP

3SAT always NPC. and in PSPACE

** 3SAT ↔ IS ↔ VC ↔ SC

{Polytime reducible to set cover}

NP ≤p NPC ≤p NPH

② Independent set Problem – The certificate "t" is the set of atleast "k" vertices ; the certifier "b" checks for those vertices no edges join any pair of them.

③ Set cover Problem – The certificate 't' is the "k" set from the given collection the certifier "b" checks that union of these sets is equal to the whole set U.

④ Vertex Cover ⑤ TSP Optimization

* NP-Hard = { X : every problem Y ∈ NP reduces to X i.e. Y ≤p X }

eg. TSP Optimization Problem:

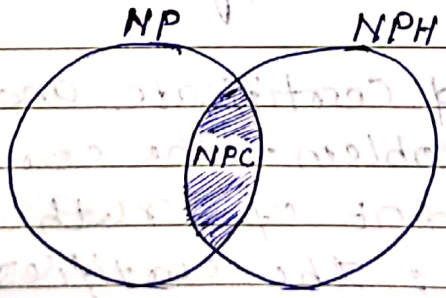
When reducing independent set can achieve of. Also with set cover, hamiltonian cycle.

Then TSP-Optimization problem is NP-Hard

* NP-Complete = { X ; X ∈ NP and X ∈ NP hard }

↓

eg. TSP O is NP
TSP O is NPH
then TSP O is NPC

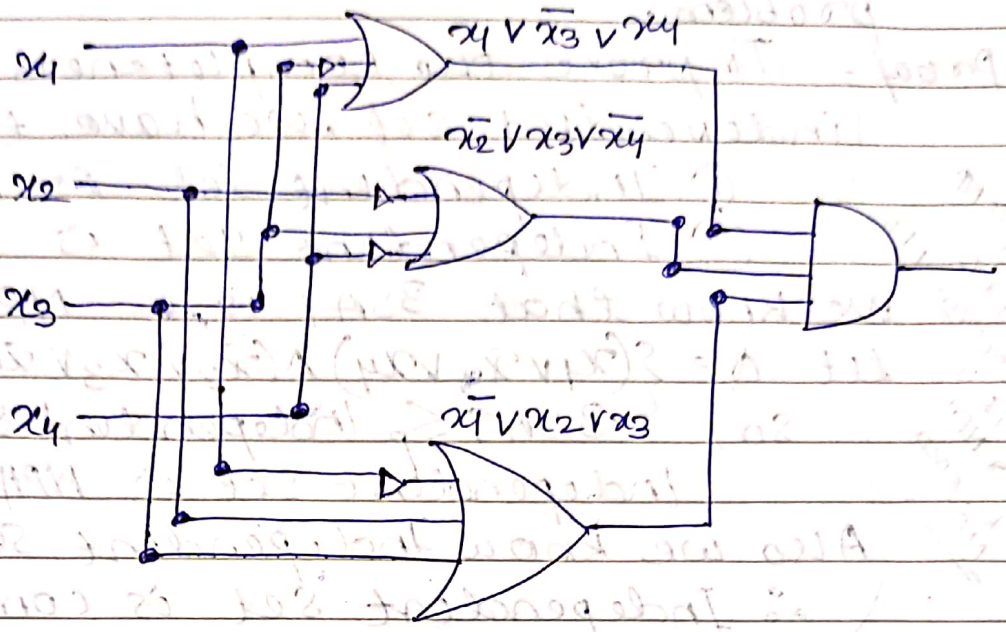


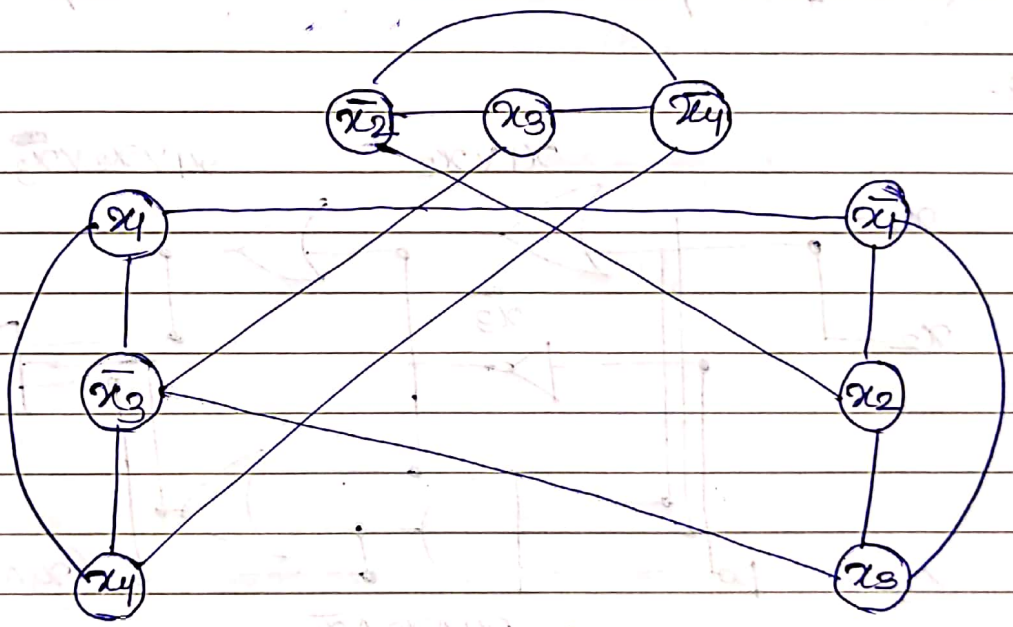
(NP) I.S — Reduces To → TSP opti Proble
(NP) S.C. — Reduces To → TSP opti Proble
(NP) H.C — Reduces To → TSP opti Proble

(NP)

then TSP O becomes NPH

Note: Circuit Satisfiability Problem is NPC





Important Note -

(1) To proove X is a NPH, just take a NPC problem and reduce to X.

(2) Now show that X is also NP problem

Then you can conclude it is NPC

Question Prove that Independent Set is NPC problem.

Proof - To prove the completeness of independent set, we have to show:

① Independent Set is NPH

② Independent Set is NP Problem

We know that 3SAT is a NPC problem

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

So, 3SAT $\leq_p$ Independent Set

Independent Set is NPH

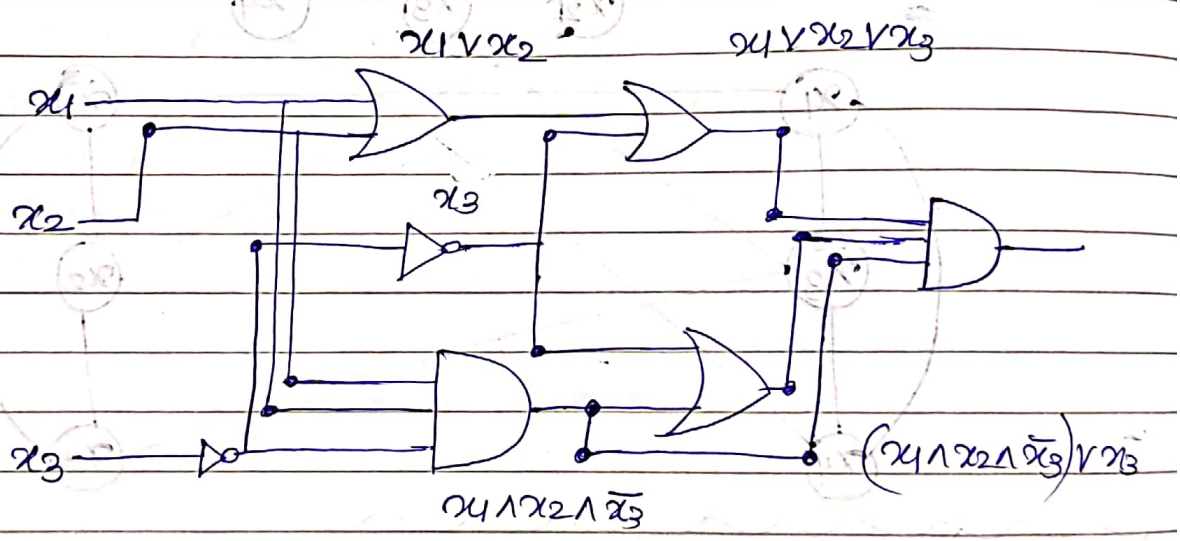
Also we know Independent Set is NP

∴ Independent Set is complete.

3SAT is NP and if 3SAT $\leq_p$ IS becomes NPH
AS IS is NP & NPH IS becomes NPC.

11/04/23

Q)



a) Reduce the above circuit satisfiablity to the formula satisfiability.

b) Check the satisfiability of the formula satisfiability.

$= \phi$

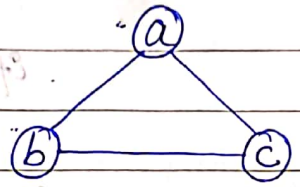
Ans) a) Output $= (x_1 \vee x_2 \vee x_3) \wedge ((x_1 \wedge x_2 \wedge \bar{x_3}) \vee x_3) \wedge (x_1 \wedge x_2 \wedge \bar{x_3})$

for $x_1 = 1, x_2 = 1, x_3 = 0$ ($\phi = 1$).

Q) Prove that vertex cover set is NPC.

Ans) Proof: i) Vertex cover set problem is NPH
ii) Vertex cover set is NP problem.

SC & VC is NP
SC ≤p VC
So, VC becomes NPH,
So, VC is NP & NPH, VC becomes NPC
As VC becomes NPC



ϕ, {a}, {b}, {c}, {a,b}, {b,c}, {c,a}, {a,b,c}

We can show that ~~3-SAT ≤p Independent Set~~ ~~Vertex Cover~~

So, vertex cover → NPC

12/04/23

Co-NP (Compliment of NP class)

It is the set of all decision problems where "NO" answers is checkable in poly. time.

In otherwords, if a problem X is in NP then its complement $\bar{X}$ is also Co-NP,

i.e., $X \in NP \Rightarrow \bar{X} \in$ Co-NP

Eg:- ① To check a prime number
② No Hamiltonian path in graph G

Q) Let S be a NPC Problem and Q, R be 2 other problems, not known to be in NP. Q is polynomial time reducible to S. And S is polynomial time reducible to R. Which one of the following statement is true:

i) R is NPC
ii) R is NPH — answer
iii) Q is NPC
iv) Q is NPH

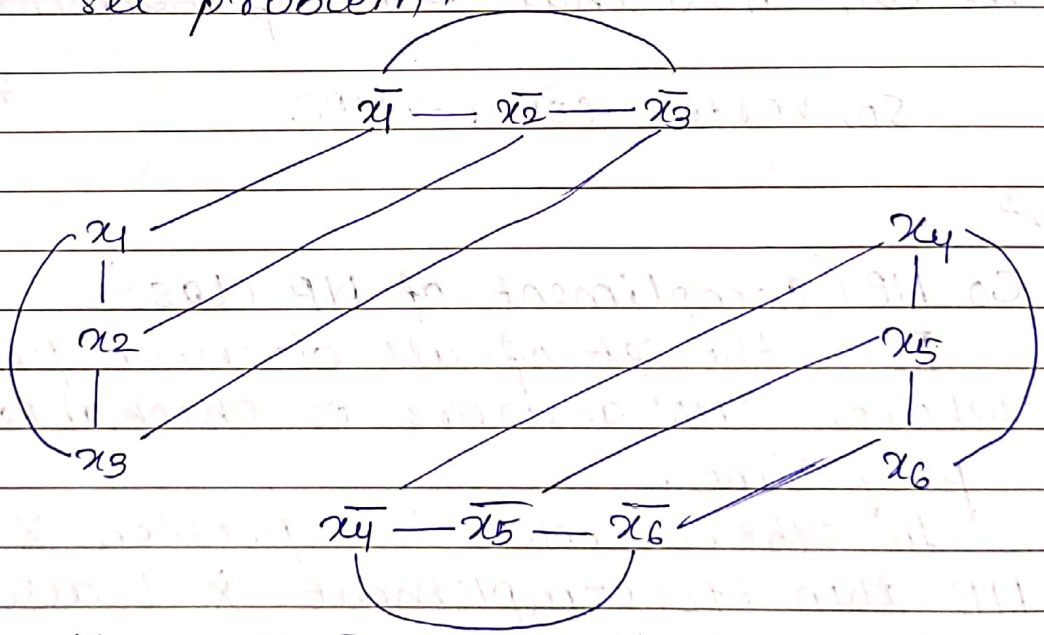
$Q \leq_p S \leq_p R$

S (NPC)

if NPC problem reducible to X then X is NPH

Q) $\phi = (x_1 \vee x_2 \vee x_3) \wedge (\bar{x_1} \vee \bar{x_2} \vee \bar{x_3}) \wedge (x_4 \vee x_5 \vee x_6) \wedge (\bar{x_4} \vee \bar{x_5} \vee \bar{x_6})$

ϕ is of 3-CNF, reduce in independent set problem.



Yes, ϕ is 3CNF.

Independent set = $\{x_1, \bar{x_2}, x_4, \bar{x_5}\}$

$|IS| = 4$

No. of clauses = 4

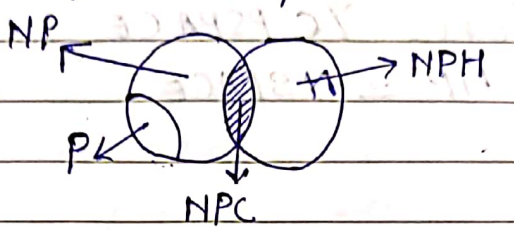
as 4 = 4, so,

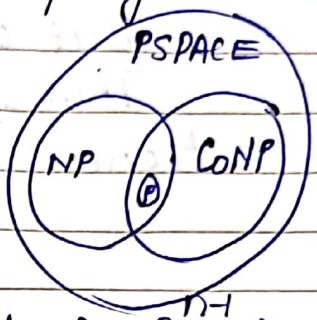
{{CNF is AND of OR}}

$3CNF \leq_p IS$

* PSPACE

The set of all problems that can be solved by an algorithm with polynomial space complexity; i.e., an algorithm that uses an amount of space is polynomial in the size of an input.

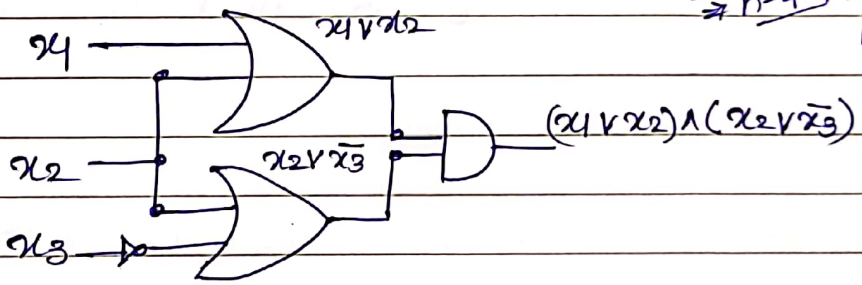




eg. ① An algorithm just count $0 \to 2^{n-1}$ in base 2 notation and the space required : n-bit counter

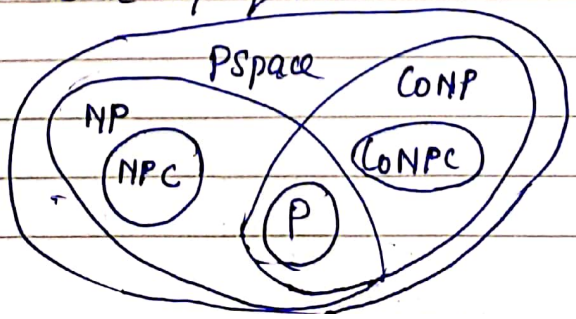
$0 \to 8$ $0 \to 2^3$ $O(4)$

$3 = n-1 \Rightarrow n = 4$ $\approx O(1)$ poly time

② 

There is an algorithm that solve 3SAT using a polynomial amount of space. We increment an n-bit counter for $0 \to 2^{n-1}$. The values in the counter corresponds to truth assignment in the following way:

→ When the counter holds a value 'q', we take it a truth assignment 'r' that sets x_i to be the value of the i^{th} bit of q.



Q) Prove that NP ⊆ PSPACE

Proof → we need to prove that

$Y \in NP \Rightarrow Y \in PSPACE$

Since $Y \in NP$, we can reduce Y into a 3SAT Problem. We know, ie.: $Y \leq_p 3SAT$

But we know, $3SAT \in PSPACE$

Therefore, $Y \in PSPACE$

Hence, $NP \subseteq PSPACE$.

17/01/23

8 Puzzle Problem.-

Initial State

3	1	2
4	5	6
8	7	

Final State

1	2	3
4	5	6
7	8	

Planning Problem-

Conditions : Set $C = \{c_1, c_2, c_3, \ldots c_n\}$

Initial configuration : Subset $C_0 \subseteq C$ of condition initially satisfied.

Goal configuration : Subset $C^* \subseteq C$ of condition to satisfy.

Operators = $O = \{O_1, O_2, O_3, \ldots O_n\}$

Note

(i) To invoke operator O_i must satisfy certain pre-requisite conditions.

(ii) After invoking 'y', certain condition become true and some condition become false.

Planning- It is possible to apply sequence of operators to get from initial configuration to goal configuration.

Example- Can we solve 8 puzzle problem

1	2	3
4	5	6
7	8	9

condition : C_{ij} , $1 \leq i, j \leq 9$

C_{ij} means i^{th} tiles in j^{th} squarres.

Inctial State-

$C_0 = \{ C_{11}, C_{22}, C_{33}, C_{44}, C_{55}, C_{66}, C_{78}, C_{87}, C_{99} \}$

Goal State-

$C^* = \{ C_{11}, C_{22}, \ldots C_{99} \}$

1	2	3
4	5	6
7	8	9

* 8 Puzzle Invariant -
Any legal move preserve the parity of the number of pairs in inversion.

* Planning Problem is in EXPTIME (Exponential Time)

① Configuration of graph G - Include node or vertex for each of 2^n possible configuration.

② Include an edge from configuration c' to configuration c'' if one of the operators can convert from c' to c''

Planning - Is there a path from C_0 to C^* in configuration graph?

Note - Configuration graph can have 2^n nodes and shortest path can be length of $2^n - 1$ (worst case)
Planning problem is in PSPACE

Theorem - Planning problem belongs to PSPACE
① Suppose there is a path from C_1 to C_2 of length 'l'; ② Path from C_1 to midpoint and for C_2 to midpoint are each less than or equal to $(l/2)$ ③ Enumerate all possible midpoint ④ Apply recursively depth of the recursion = $\log l$.

Algorithm -

```
boolean hasPath (C1, C2, L){
    if (L ≤ 1)
        return correct answer
    for each config C'
        x = hasPath (C1, C', L/2)
        y = hasPath (C1, C', L/2)
    if (x && y)
        return true
    return false
}
```

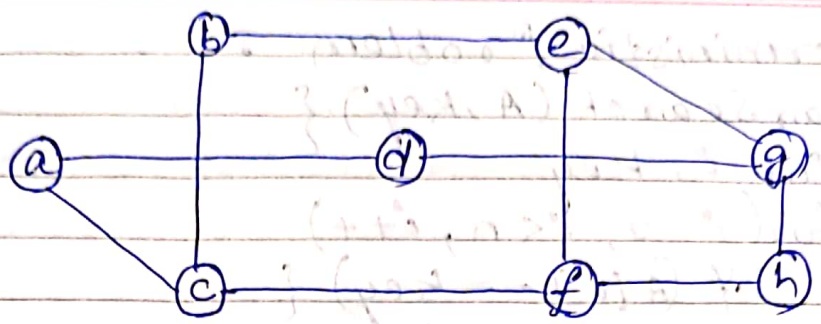
* PSPACE-PROBLEM- Competitive Facility Location Problem
The input in a graph with positive nodes weights as well as a target value B

- Consider the following game-
① 2 player alternate in selecting nodes. A node cannot be selected if a neighbour or a adjacent node has been selected previously. The game ends when no more node can be sellected
Objective - Can the 1st player prevent the 2nd player from getting profit of atleast B amount.

Note-
→ QSAT is a PSPACE complete
→ How to prove a problem X is PSPACE-Complete

Procedure ① → To show X is PSPACE
Procedure ⑪ → All PSPACE problem Y can be reduced
i.e, $Y \leq_p X$ to X

Q)



(a) find the minimum vertex cover set
(b) find the maximum independent set
(c) Define independent set and vertex cover set

Ans) a) {c, e, d, h}, {g, f, b, a}
b) {a, b, f, g}, {c, d, e, h}
c) Independent Set – is a set of vertices $S \subseteq V$ of a graph $G(V, E)$ such that for any pair of vertices of S have no same edge.

Vertex Cover Set – is a set of vertices $S \subseteq V$ of a graph $G(V, E)$ such that for every edge $e \in E$ contains atleast one end vertex in S.

19/4/23

* Deterministic Problem :

P

```
linearSearch (A, key) {
    int i, key
    for(i=0, i<n, i++)
        if (A[i] == key) {
            return found
        }else{
            return not found
        }
    }
}
```

* Non-deterministic Problem :

NP

```
linearSearch (A, key) {
    int i, key
    j = BlackBox();
    if (A[j] == key) {
        found
    }else{
        not found
    }
}
```

* Summary –

① P = { X : X is solvable in poly. time }

eg : P = { sorting, searching ... }

② NP = { X : X is solvable in non-deterministic poly. time, verification is in poly. time

eg. NP = { vertex cover, independent set ... }

(3) NPH = { X : $\forall$ all NP Y $\leq_p$ X }
(4) NPC = { X : X $\in$ NP and X $\in$ NPH }
(5) 3SAT, Circuit SAT are NPC
(6) 2SAT is P Problem.

Q) The problems 3SAT and 2SAT are :
(a) Both in P
(b) Both in NPC
✓(c) NPC and in P respectively
(d) Undecidable and NPC respectively

Q) Consider 2 desicion problem Q_1 and Q_2 such that $Q_1 \leq_p$ 3SAT and 3SAT $\leq_p Q_2$ then which one of the following is true:
✓(a) Q_1 is NP, Q_2 is NPH
(b) Q_2 is NP, Q_1 is NPH
(c) Both Q_1, Q_2 are in NP
(d) Both Q_1, Q_2 are NPH

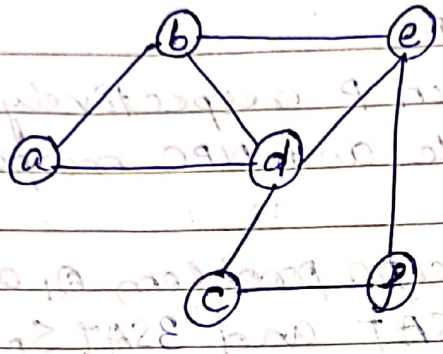
NPC → NPH → NP (reduced).

Q) Let S be an NPC problem and Q & R be 2 other problems not known to be in NP. $Q \leq_p S$ and $S \leq_p R$. Which one of the following is true:
(a) R is NPC
✓(b) R is NPH
(c) Q is NPC
(d) Q is NPH

21/04/23

* Finding the smallest vertex cover

Given a graph $G(V, E)$ and an integer 'k', To find a vertex cover of size atmost 'k' i.e., a set of vertices 's', $S \subseteq V$ of size $|s| \leq k$, such that every edge $e \in E$ has atleast one end in vertex.



$|V| = n = 6$
$|E| = 8$

Simplify all the subsets of V of size k. and see whether any of them are vertex cover.

Running Time:

(1) To get all subsets of size 'k' = ${}^{n}C_{k}$ (total no. of vertices)

(2) To check whether it is a vertex cover $= O(nk)$

$\therefore$ Total time $= {}^{n}C_{k} \times O(nk)$

$= O(nk\ {}^{n}C_{k})$

$\approx O(kn^{k+1})$

*Note- This running time is quite impractical. For eg., if $n = 1000$ and $k = 10$, then on a computer, executing a million high level instructions per second, it takes atleast 10^{24} seconds decide if G has a k node vertex cover set, which is larger

than million of years.

Do something ie, practically viable when 'k' is a small constant. A much better algorithm can be developed with a running time bound of $O(2^k \times kn)$.

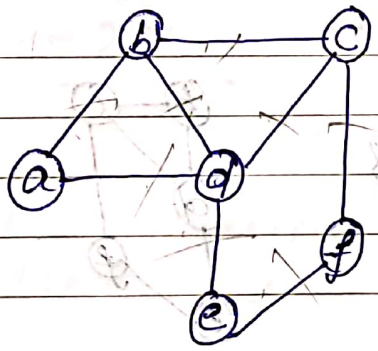
eg: $n=1000, k=10$, computer will able to execute in few seconds.

24/04/22

* Theorem 1:

If $G(V,E)$ has n nodes, the maxm. degree of any node is atmost d and there is a vertex cover of size, atmost K. Then G has atmost $(K \times d)$ edges.

eg.



$d=4$

$K=3$

$(K \times d) = 12$

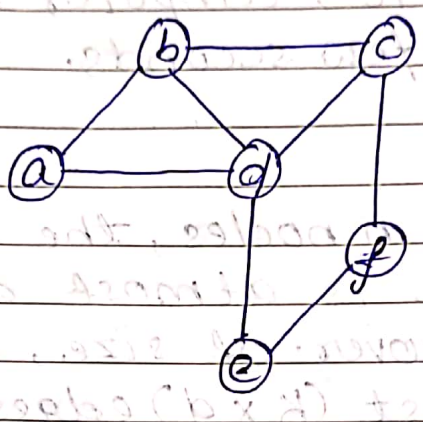
$n=6$ $k=3$ size vertex cover set

$d=4$ then, $(K \times d) = 3 \times 4 = 12$ edges (atmost)

no. of edge ≤ 12

$8 \leq 12$

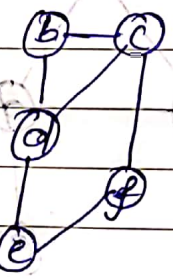
* Theorem 2:
Let $e = u, v$ be any edge of G. The graph G has a vertex cover of size atmost k, iff atleast one of the graph $G-\{u\}$ or $G-\{v\}$ has a vertex cover of size atmost $(k-1)$.



$S = \{b, d, f\}$

$G - \{a\} = \{d, e, f\}$

$G - \{b\} = \{$



* Algorithm-
To search for a k-node V.C. in G
If G contains no edges, the empty set is V.C.
If u>kd edges, it has no V.C.
else let $(u,v) \in E$ of G
recursively check if either of $G-\{u\}$ or $G-\{v\}$ has a V.C. of size $(k-1)$

In this case $T \cup \{v\}$ is a k-node V.C.
↓
vertex having highest degree

* Running Time :
$$T(n,k) = 2T(n,k-1) + O(kn)$$
$$T(n,1) \leq cn$$

Soln - Let us assume $T(n,k) \leq c\,2^k nk$
Also assume $T(n,k-1) \leq c\,2^{k-1}n(k-1)$ true for $<k$

$$T(n,k) = 2T(n,k-1) + c_1kn$$
$$= (k-1)2 \times c.2^{k-1}n + c_1kn$$
$$= c2^knk - cn + c_1kn$$
$$\leq c2^knk - (cn - c_1kn)$$
$$\leq c2^knk$$

$$O(nk2^k)$$

Revision - Ch2

* Closest Pair and Close Enough -

Given $S = \{8, 4, 10, 3, 9, 15, 18, 6\}$ and a threshold $t = 2$.

① then sort(S)

$S = \{3, 4, 6, 8, 9, 10, 15, 18\}$

then note down the difference between each element.

$S = \{3, 4, 6, 8, 9, 10, 15, 18\}$

1 2 2 1 1 5 3

Closest Pair : $(S_{i+1} - S_i) \leq t$

i.e., (3,4)(4,6)(6,8)(8,9)(9,10)

Close enough pair : the min. difference between S_i and S_{i+1}. In this case it is (1).

i.e., (3,4)(8,9),(9,10)

* Convex Hull :

Given a set $S = \{5, 2, 7, 3, 8, 4, 6, 12, 10, 11\}$

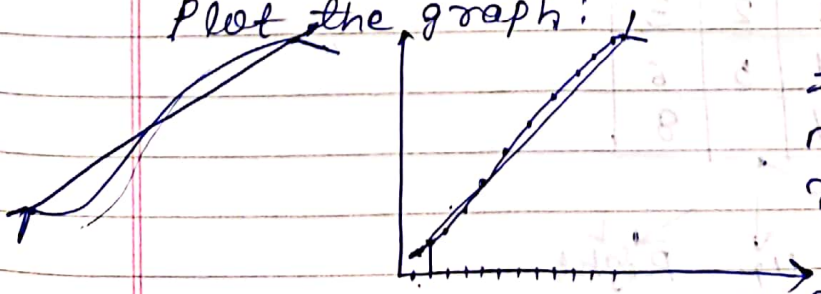
① then sort(S)

$S = \{2, 3, 4, 5, 6, 7, 8, 10, 11, 12\}$

Create points (i, i^2)

(2,4),(3,9),(4,16),(5,25),(6,36),(7,49),(8,64)
(10,100),(11,121),(12,144)

Plot the graph :



then join the left most point with the right most point to get a polygon containing all the points.

(Overlapping allowed)

8 Puzzle Problem –

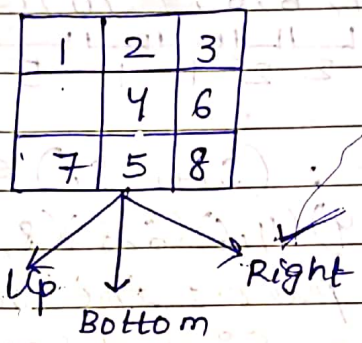
1	2	3
4	5	6
7	8	

Initial state → Goal state

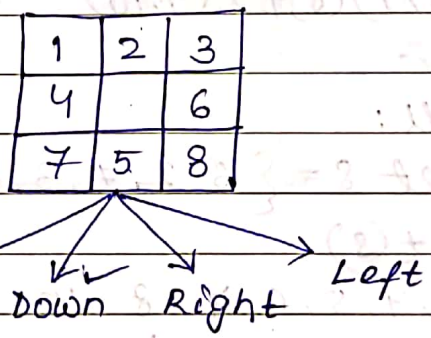
Initial = { C_{11}, C_{22}, $C_{3,3}$, C_{45}, C_{66}, C_{77}, C_{58}, C_{89} }

(element, position)

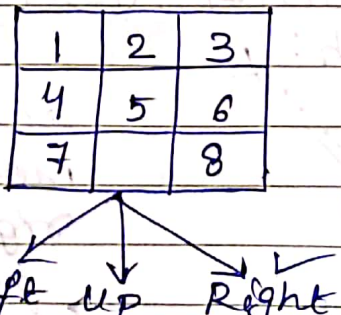
Goal = { C_{11}, C_{22}, C_{33}, C_{44}, C_{55}, C_{66}, C_{77}, C_{88} }



Optimized choice : right, cost = 1



Optimized Choice : Down , cost = 2



Optimized Choice : Right, cost = 3

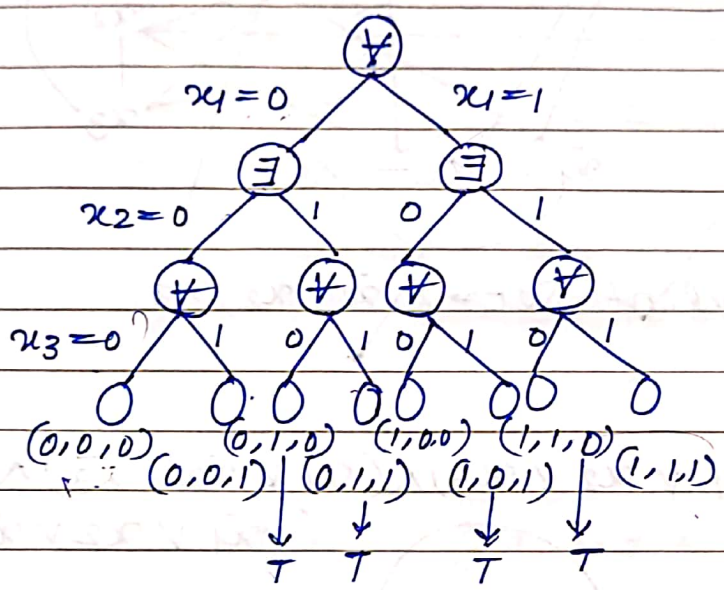
* Q-SAT:

$$\phi = (x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee \bar{x_3}) \wedge (\bar{x_1} \vee x_2 \vee x_3) \wedge (\bar{x_1} \vee \bar{x_2} \vee \bar{x_3})$$

{∃ = such that, ∀ = for all}

Check $\exists x_2 \forall x_1 \forall x_3$.

Taking x_1 condition first then x_2 and then x_3 condition.



For the values (0,1,0), (0,1,1), (1,0,1), (1,1,0) the ϕ is satisfiable.

(Ch 4)

Approximation Algorithm

08/05/23

* Load Balancing Problem
Given a set of 'm' machine, $M_1, M_2, M_3, \ldots M_m$ and a set of 'n' number of jobs; each job 'j' has a processing time t_j

Goal – Assign each job to one of the machine, so that the load placed on all machine are as balanced as possible

Let A(i) denote the set of jobs assigned to machine M_i. So machine M_i needs to work for a total time of

$$T_i = \sum_{j \in A(i)} t_j$$

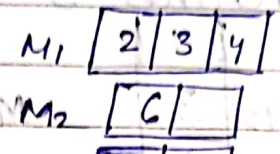
it is due to the load on machine M_i.

Minimize a quantity known as MakeSpan
It is simply the maximum load on any machine

$$T = \max_i T_i$$

Note – The scheduling problem of finding an assignment of minimum makespan is (NPH)

j	t_j
J_1	2
J_2	3
J_3	4
J_4	6
J_5	2
J_6	2



$T_1 = \sum_{j \in \{J_1, J_4, J_5\}} t_j = 2+3+4 = 9$

$T_2 = \sum t_j = 6$

$T_3 = \sum t_j = 2+2 = 4$

* Designing the algorithm –

Greedy Balance

1. Start with no jobs assigned.
2. Set $T_i = 0$ and $A(i) = \phi$ for all machine M_i
3. for $j = 1, 2, \ldots n$
4. Let M_i be a machine that achieves the minimum $\min_k T_k$
5. Assign job j to machine M_i
6. Set $A(i) = A(i) \cup \{j\}$
7. Set $T_i = T_i + T_j$
8. end for

$T.C = O(n)$ → very very less than NPH problem, so of retrieving approx. answer

eg – Sequence of jobs with t_j {2, 3, 4, 6, 2, 2}
There are 3 machines.

M_1		M_2		M_3	
J_4	6	J_5	2	J_6	2
J_1	2	J_2	3	J_3	4

$T_1 = 0+2+6 = 8$

$T_2 = 0+2+3 = 5$

$T_3 = 0+4+2 = 6$

$A(1) = \{J_1, J_4\}$

$A(2) = \{J_2, J_5\}$

$A(3) = \{J_3, J_6\}$

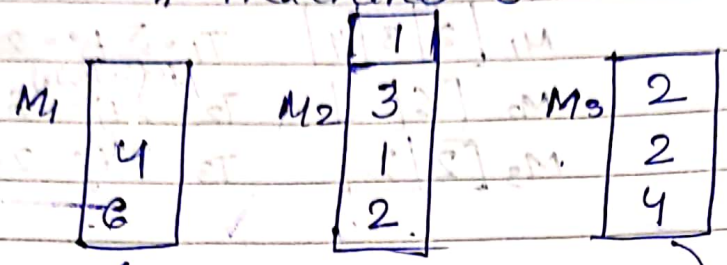
Makespan – Max. load on 3 machines ← which is of M_1

⇒ $J_1 + J_4$

⇒ $6+2 = 8$

eg- $t_j = \{6, 2, 4, 1, 3, 2, 4, 2, 1\}$

machine = 3



$T_2 = 0+2+1+3+1 = 7$

$T_1 = 0+6+4 = 10$

$T_3 = 0+4+2+2 = 8$

$A_1 = \{J_1, J_7\}$

$A_2 = \{J_2, J_4, J_5\}$ Make Span = 10

$A_3 = \{J_3, J_6, J_8\}$

09/05/23

Improved Algorithm

Sorted Balance

1. Start with no jobs assigned.
2. Set $T_i = 0$ and $A(i) = \phi$ ∀ machines M_i
3. Sort all jobs in decreasing order of processing time t_j
4. Assume that $t_1 \geq t_2 \geq \cdots \geq t_n$
5. for ($j = 1, 2 \ldots n$)
6. Let M_i be the machine that achieves the minimum $\min_k T_k$
7. assign j to machine M_i
8. Set $A(i) = A(i) \cup \{j\}$
9. Set $T_i = T_i + t_j$
10. end for

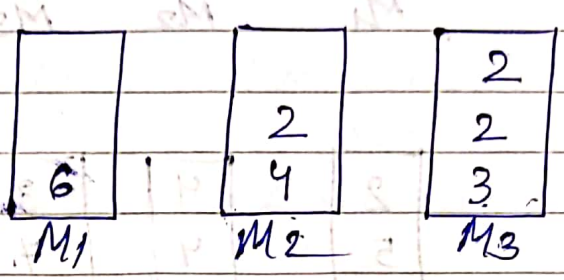
T.C = O(nlogn)

eg.

jobs	1	2	3	4	5	6
t_j	2	3	4	6	2	2

6 4 3 2 2 2

J_4 J_3 J_2 J_1 J_5 J_6 $\quad$ # machine = 3



$T_1 = 6$ $\quad$ $T_2 = 6$ $\quad$ $T_3 = 7$

$A_1 = \{J_4\}$ $\quad$ Makespan = 7

$A_2 = \{J_3, J_5\}$

$A_3 = \{J_2, J_1, J_6\}$

* Note

(1) Algorithm greedy balance produces an assignment of jobs to the machines with makespan $T \leq 2T^*$ where T^* is actual optimal solution.

(2) Algorithm sorted balance produces an assignment of jobs to machine with makespan. $T \leq \frac{3}{2} T^*$ where T^* is actual optimal solution.

approx. used in NPC, NPH

Q) Jobs: 4 6 2 7 5 4 2 3 4
machines = 3
(1) Solve using greedy approx algo.
(2) Solve using sorted balance approx. algo

Compute makespan in both

(Ans)(1)

j	t_j
J_1	4
J_2	6
J_3	2
J_4	7
J_5	5
J_6	4
J_7	2
J_8	3
J_9	4

M_1	M_2	M_3
2	4	3
5	4	7
4	6	2

$A_1 = \{J_1, J_5, J_7\}$
$A_2 = \{J_2, J_6, J_9\}$
$A_3 = \{J_3, J_4, J_8\}$

MakeSpan = 14

(2) 4 6 2 7 5 4 2 3 4
J_1 J_2 J_3 J_4 J_5 J_6 J_7 J_8 J_9

In decreasing order

7 6 5 4 4 4 3 2 2
J_4 J_2 J_5 J_1 J_6 J_9 J_8 J_3 J_7

M_1	M_2	M_3
2	2	3
4	4	4
7	6	5

$A_1 = \{J_4, J_9, J_7\}$
$A_2 = \{J_2, J_6, J_3\}$
$A_3 = \{J_5, J_1, J_8\}$

MakeSpan = 13

10/05/22

* Center Selection Problem -

- Problem Statement - Consider the following scenario; we have a set S of 'N' sites. Now we want to select 'K' centers for building big shoping malls such that people in each of these 'N' sites will shop at one of the malls.

- Formulation - We are given an integer 'K', a set S of 'N' sides and a distance function d (Eucleadian Distance). The distance function satisfies the following properties:

(i) $d(s,s) = 0 \quad \forall \; s \in S$

(ii) Symmetric $d(s,z) = d(z,s) \quad \forall \; s,z \in S$

(iii) Triangle Inequality

$$d(s,h) \leq d(s,z) + d(z,h)$$

where, $s,z,h \in S$

- Let 'C' be a set of centers. We define the distance of a city 's' to the center as

$$d(s,C) = \min_{c \in C} \{ d(s,c) \}$$

Now we say that 'C' forms an r-cover of each sites is within distance atmost 'r' from one of the centers.

So, $d(s,C) \leq r \quad \forall \; s \in S$

'r' is called covering radius of 'C' and denoted by r(C).

- Goal- Our goal is to select a set of centers C of size k for which r(C) is as small as possible.

- Algorithm:

1. S' will represent the sites that will need to be covered.
2. Initialize $S' = S$
3. $C = \phi$
4. while $(S' \neq \phi)$
5. Select any $s \in S'$ and add $s \in C$.
6. Delete all sites from S' that are at distance atmost 2r from s.
7. end while
8. if $|C| \leq K$
9. return C
10. else
11. no such center of size k exist

12/05/23

* Set Cover : A General Greedy Heuristic

Problem Statement -

Let U be a set of n elements and a list $S_1, S_2, S_3, \ldots S_m$ subsets of U. Consider each subset S_i has an associated weight $w_i > 0$. Then find a set cover C so that the total weight $w(C)$ = sum of weight of each subset

$$\Rightarrow W(C) = \sum_{S_i \in C} w_i \text{ is minimized.}$$

*Note -

(1) Set cover - is a collection of subsets, whose union is equal to U.

(2) If $w_i = 1$, then the minimum weight of a set cover is equal to the number of subsets in the set cover.

eg. - $U = \{a, b, c, d, e, f, g\}$

$S_1 = \{a, b\}$ $S_2 = \{a, c\}$ $S_3 = \{a, c, d\}$

$S_4 = \{b, d, e\}$ $S_5 = \{c, e, g\}$ $S_6 = \{e, f, g\}$

$S_7 = \{a, b, c\}$ $S_8 = \{c, d\}$

$U = S_1 \cup S_3 \cup S_6 \Rightarrow C_1 = \{S_1, S_3, S_6\}$

$U = S_2 \cup S_4 \cup S_6 \Rightarrow C_2 = \{S_2, S_4, S_6\}$

$U = S_6 \cup S_7 \cup S_8 \Rightarrow C_3 = \{S_6, S_7, S_8\}$

$w_1 = 4$ $w_2 = 5$ $w_3 = 7$ $w_4 = 6$

$w_5 = 5$ $w_6 = 3$ $w_7 = 7$ $w_8 = 3$

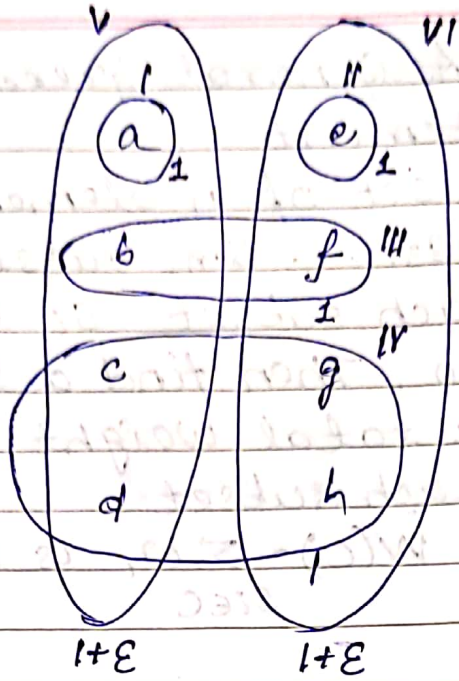
$w(C_1) = w_1 + w_3 + w_6 = 4 + 7 + 3 = 14$

$w(C_2) = w_2 + w_4 + w_6 = 5 + 6 + 3 = 14$

$w(C_3) = w_6 + w_7 + w_8 = 3 + 7 + 3 = 13$

Fall of $w_i = 1$ of size of S. U equal weight of

eg-



$C_1 = \{V, VI\}$, $w(C_1) = (1+\varepsilon) + (1+\varepsilon) = 2+2\varepsilon$

$C_2 = \{I, II, III, IV\}$, $w(C_2) = 1+1+1+1 = 4$

Notation -

Define $C_S \equiv$ average cost of elements

and $C_S = \frac{w_i}{|S_i \cap R|}$ $\forall\ s \in S_i \cap R$

*** find cost of all set cnsc with minimum weight

$S_5 = \{c, e, g\}$ $w_5 = 5$

$R = \{a, b, c, d, e, f, g\}$

So, $C_5 = \frac{5}{3} = 1.7 = C_e = C_g$

updated

$S_4 = \{a, b, c\}$ $w_4 = 7$

$R = \{a, b, d, f, h\}$

So, $C_4 = \frac{7}{2} = 3.5$

Decision - NPC
optimization - NPH
(approx)

* Greedy Set Cover -

1. Start with $R = U$ and no sets selected
2. While $(R \neq \emptyset)$
3. Select set S_i that minimize $\frac{w_i}{|S_i \cap R|}$
4. $R = R - S_i$
5. end while
6. Return the select sets.

15/05/23

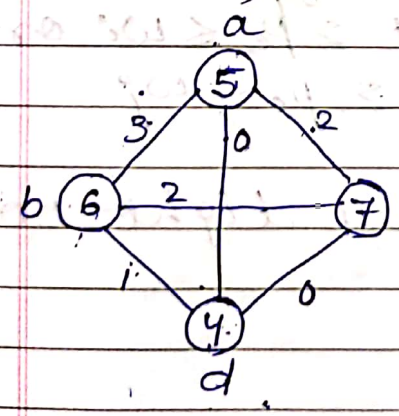
* The Pricing Method Vertex Cover

Recall -

- Vertex Cover - In a graph $G(V, E)$ is a set $S \subseteq V$ so that each edges has atleast one end in S.

Problem Statement -

consider $i \in V$ has a weight $w_i \geq 0$. The weight of a set S of vertices denoted by $w(S) = \sum_{i \in S} w_i$. Our task is to find a vertex cover S of which $w(S)$ is minimum.



$S_1 = \{a, d, b\}$ $S_2 = \{b, c, a\}$

$w(S_1) = \sum w_i = w(a) + w(d) + w(b)$
$S_1 = (a, d, b)$ $= 5 + 6 + 4 = 15$

$w(S_2) = \sum w_i = w(b) + w(c) + w(a)$
$S_2 = \{b, c, a\}$ $= 6 + 7 + 5 = 18$

Note -

→ When all weight are equal to 1, vertex cover of weight atmost 4.

→ A node i is called tight of

$$\sum_{j \leftrightarrow i} p(i,j) = w_i$$

p(i,j) → edge value to be decided.

vertex a : $w(a) = p(a,b) + p(a,d) + p(a,e)$

$5 = 3 + 2 + 0 = 5$

↓

It is a tight

vertex c : $w(c) = p(a,c) + p(b,c) + p(c,d)$

$7 \neq 2 + 2 + 0 = 4$

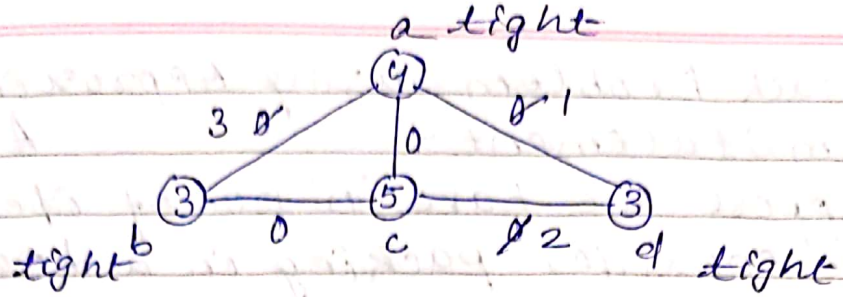
↓

It is not a tight

* VertexCoverApproximation (G, w)

1. Set initially $p_e = 0$ for all edge $e \in E$
2. while there is an edge $e = (i,j)$ such that both i and j are not tight
3. Select such an edge e(i,j)
4. Increase p_e such that $\sum p_e \leq w_i$ or $\sum p_e \leq w_j$
5. end while
6. Let S be the set of all tight nodes
7. return S.

eg



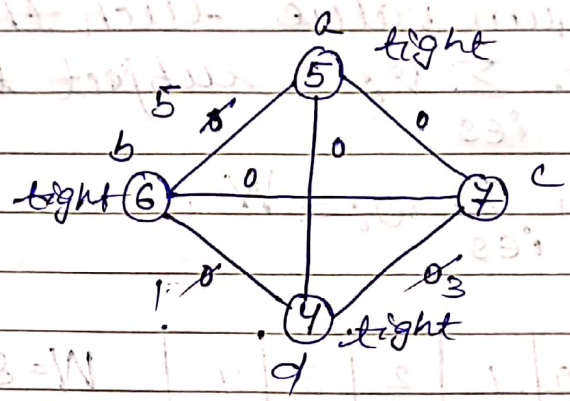
$S = \{a, b, d\}$ ← set of all tight nodes

↓

Also vertex cover

$w(S) = 4 + 3 + 3 = (10)$

eg



$S = \{a, b, d\}$ ← set of all tight nodes

↳ Also vertex cover

$w(S) = 5 + 6 + 4 = (15)$

↑

Minimum weight of vertex cover

16/05/23

Knapsack Problem using Approximation Algorithm

* Problem Statement -

Suppose we have 'n' no. of items that consider packing in a knapsack. Each item $i = 1, 2, \cdots n$ has 2 parameters,

(i) A weight w_i

(ii) And a value / price v_i

So given a knapsack capacity W, the goal is to find a subset S of items of maximum value such that, maximize $\sum_{i \in S} v_i$, subject to the condition $\sum_{i \in S} w_i \leq W$ ... (2)

eg.

item(i)	1	2	3	4
p_i	10	12	15	16
w_i	2	3	5	4

W = 8

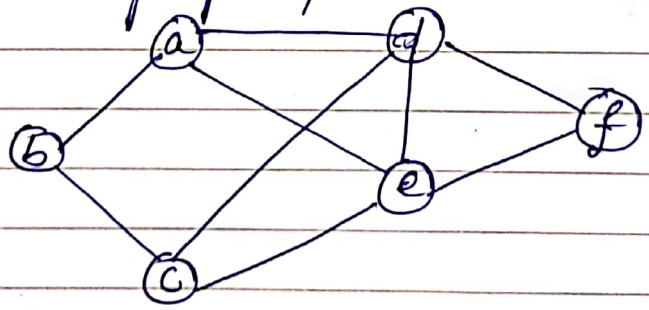
19/05/23

Class Questions -

(1) Write set cover approx. algo.

(2) Write vertex cover approx. algo.

(3) Find a near optimal V.C of the following graph -



*** you can assume weight of vertex as the degree of the vertex if not given

(i) First using V.C. approx. algo.

(ii) Second using S.C approx. algo.

30/05/23

* The Knapsack Problem

Pseudo Polynomial- The pseudo-polynomial time complexity means polynomial in the value but exponential in the size of input.

Recall Dynamic Programming

0	1	2	3	4
P_i	10	12	15	16
W_i	2	3	5	4

Knapsack Capacity = 8

* Approximation Algorithm for 0/1 knapsack

Knapsack_Approx (t):

Set $b = \left(\frac{t}{2n}\right) \max_i v_i$

take $\hat{v}_i = \left\lceil \frac{v_i}{b} \right\rceil$ $b = 1, 2, 3, \ldots n$

Return the set S of items found

O_i	1	2	3	4
v_i	10	12	15	8
w_i	3	4	2	5

* Another Dynamic Approach

We define our subproblems as follows-

The subproblems is defined by i and a target value V.

Notation $\overline{OPT}(i, V)$ is the smallest knapsack weight W, so one can obtain a solution using a subset of items $\{1, 2, \ldots i\}$ with value atleast V

We will have subproblems for all $i = 0, 1, 2, \ldots n$ including 0^{th} item and values $V = 0, 1, 2, \ldots \sum_{i=1}^{i} V_i$

A final solution is $\overline{OPT}(n, V) \leq W$

O = solution set.

* Case I:

→ if $n \notin O$ then $\overline{OPT}(n, V) = \overline{OPT}(n-1, V)$

Case II:

→ if $n \in O$ then $\overline{OPT}(n, V) = w_n + \overline{OPT}(n-1, V-V_n)$

→ if n is only $\overline{OPT}(n, V) = w_n$

* Formula

If $V > \sum_{i=1}^{n-1} V_i$ then

$\overline{OPT}(n, V) = w_n + \overline{OPT}(n-1, V-V_n)$

else

$\overline{OPT}(n, V) = \min\{\overline{OPT}(n-1, V), w_n + \overline{OPT}(n-1, \max(0, V-V_n))$

Algorithm -

knapsack(n)

Array $M_{n \times V}[0, 1, \cdots n, 0, 1, \ldots V]$

$M[i, 0] = 0 \quad \forall \quad i = 0, 1, \ldots n$

for $i = 1, 2, \ldots n$

for $v = 1, \ldots \sum_{j=1}^{i} v_j$

if $v > \sum_{j=1}^{i=1} v_j$

$M(i, v) = w_i + M[i-1, v]$

else

$M(i, v) = \min \{M[i-1, v], w_i + M[i-1, \max(0, v - v_i)]\}$

return $M[n, v] \leq W$

02/06/23

Randomized Algorithms —

Randomized Quicksort

Finding the median:

Let $S = \{a_1, a_2, \cdots a_n\}$

Defn - Median is the middle position of the sorted array.

The K^{th} largest element in S

$$K = \begin{cases} n/2 & \text{if } n = \text{even} \\ (n+1)/2 & \text{if } n = \text{odd} \end{cases}$$

Algorithm to find k^{th} largest element in S.

Select (S, k)

choose a splitter $a_i \in S$

for each element a_j of S

put a_j in S^- if $a_j < a_i$

put a_j in S^+ if $a_j > a_i$

if $|S^-| = k-1$ then

then a_i is desired element

else if $|S^-| > k$

then k^{th} largest lies in S^-

recursively call Select(S^-, k)

else

suppose $|S^-| = d < k-1$

then k^{th} largest lies in S^+

recursively call select(S^+, k-1)

Time Complexity -

(1) Best Case - Splitter always median

$$T(n) = T(n/2) + O(n)$$

(2) Average Case -

$$T(n) = T((1-\varepsilon)n) + n \quad \varepsilon > 0$$
$$= n + (1-\varepsilon)n + (1+\varepsilon)^2 n \cdots$$
$$= n[1 + (1-\varepsilon) + (1-\varepsilon) + \cdots]$$
$$= \frac{1}{\varepsilon} \times n = O(n)$$

(3) Worst Case - Splitter always minimum/maximum

$$T(n) = T(n-1) + O(n)$$
$$= O(n^2)$$

QuickSort(S)
  if |S| < 3
    sort ≤ trivially
  else
    choose a splitter $a_i \in S$ uniformly at random
    for each element 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
    recursively call QuickSort(S^-) and QuickSort(S^+)
    output the sorted set S^-, then a_i, then sorted set S^+

Expected T.C = $O(n \log n)$

* Note - An element of the set 'S' central if atleast a quarter of elements is smaller than itself or atleast a quarter of the element are longer than itself. i.e. if a central is chosen as splitter then atleast a quarter of the set will be thrown away. the set will shrink by a factor of 3/4.

```
modified QuickSort(S)
    If |S| ≤ 3
        sort S trivially
    else
        while there is no
            central splitter. choose
            a central splitter ai ∈ S
            uniformly at random.

        for each element aj ∈ S
            Put aj in S⁻ if aj < ai
            Put aj in S⁺ if aj ≥ ai
        end for
        if |S⁻| ≥ |S|/4 and |S⁺| ≥ |S|/4
            then ai is the
                central splitter
        Recursively call
        Quicksort(S⁻) & Quicksort(S⁺)
        Output the sorted set S⁻
        then ai, then sorted set S⁺
```

05/06/23.

* Hash Table - is an effective data structure for implementing dictionaries.
Three operation - ① Insert
② Search
③ Delete

- Direct Address Table -

$U = \{2, 4, 6, 1, 7, 10\}$

$h(k) = k$

max = 10
create with table with slots 0 to 10

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

we use a hash function 'h()' to compute slot from the keys.

(1) Division Method = $h(k) = k \bmod m$

m = size of the table.

$U = \{12, 13, 24, 6, 9, 18, 27, 31\}$ $m = 10$

$h(12) = 12 \% 10 = 2$ ← hash value

$h(13) = 3$

$h(24) = 4$

$h(6) = 6$

$h(9) = 9$

$h(18) = 8$

$h(27) = 7$

$h(31) = 1$

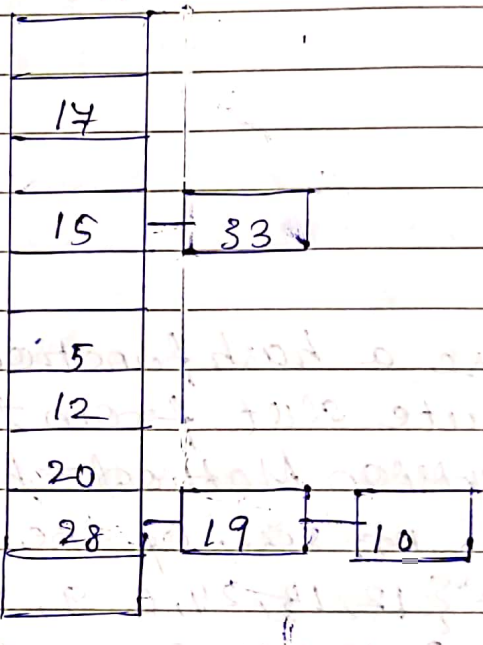
9	9
8	18
7	27
6	6
5	
4	24
3	13
2	12
1	31
0	

* Collision-
If $h(k_1) = h(k_2)$ since $k_1 \neq k_2$ keys then it is called collision

- How to avoid collision → Chaining.

Q) Demonstrate what happens when we insert the keys
$U = \{5, 28, 19, 15, 20, 33, 12, 17, 10\}$
into a hash table with collisions resolved by chaining, but the table has 9 slots and $h(k) = k \bmod 9$.

$h(5) = 5 \% 9 = 4$
$h(28) = 1$
$h(19) = 1$
$h(15) = 6$
$h(20) = 2$
$h(33) = 6$
$h(12) = 3$
$h(17) = 8$
$h(10) = 1$



* Load Factor: $\alpha = n/m$
(n = no. of keys, m = size of table)

eg. There are 25 key element & table size = 5, then $\alpha = 25/5 = 5$

Searching in HashTable : Best = $O(1)$
Avg. = $O(n)$
Worst = $O(n)$

(2) The Multiplication Method -

$h(k) = \lfloor m(kA \bmod 1) \rfloor$

08/06/23

* Universal Class of Hash Functions:-

Let $\mathbb{Z}_p = \{0, 1, 2, \ldots p-1\}$

$\mathbb{Z}_p^* = \{1, 2, 3, \ldots p-1\}$ where P is prime

Integer modulo p

We assume that the size of the universe of keys is greater than the number of slots in hash table.

$$h_{ab}(k) = ((ak+b) \bmod p) \bmod m$$
$$a \in \mathbb{Z}_p^*$$
$$b \in \mathbb{Z}_p$$

Example, $p=17$, $a=3$, $b=4$, $m=6$, $k=8$

$h_{ab}(k) = ((ak+b) \bmod p) \bmod m$

$= [illegible]$

$= ((8 \times 3+4) \% 17) \% 6$

$= ((24+4) \% 17) \% 6$

$= (28 \% 17) \% 6$

$= 11 \% 6 = 5$

$\therefore h_{ab}(8) = 5$

The family of all such hash functions is-

$H_{pm} = \{h_{ab} : a \in \mathbb{Z}_p^*, b \in \mathbb{Z}_p\}$

each hash function has maps $\mathbb{Z}_p$ to $\mathbb{Z}_m$

Note:- $|H_{pm}| = p(p-1)$ choice of $a = p-1$

choice of $b = p$

Theorem - The class of H_{pm} of hash function defined by above equation is universal.