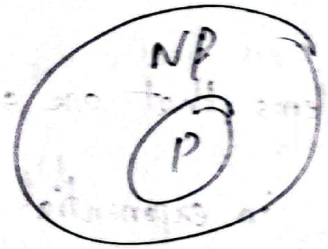
NP problems are solved by non-deterministic algorithms, which means we don't know the statement inside. But we can discover them later on and reduce their time complexity to solve in polynomial time. which means the problem is now reduced to P class. So, all NP problems can be a part of P class.



2K18

1. (a) see 2K12

space → (3+3)×2

2. (b)

```
int ** matrix_sum(int m[][3], int m2[][3]){
    int i, j, **m3;          space → 2+2*
                             time → 1*m
    m3 = malloc(sizeof(int*)*3);   space → 2*3
                                   time → 1*m
    for(i=0; i<3; i++)  ⟶ time = (3+1)m
                          space
    {
        for(j=0; j<3; j++)  ⟶ time = 3(3+1)m²
        {
            m3[i][j] = m[i][j] + m2[i][j];
                                   time → 3*3m²
        }
    }
    return m3   time = 3
};
```

total :- 22 byte space

time :- m + m + 4m + 12 + 9m² + 3m²

⇒ 9m + 21m²

⇒ ≈ m².

1 (c) (i) $\log(n!) = f(n)$

$\Rightarrow \log(n \cdot n-1 \cdot n-2 \cdots 3.2.1) \geq f(n)$

$\Rightarrow \log(n \cdot n \cdot n \cdots 2n \, 1) \geq f(n)$

$\Rightarrow (\log n + \log n + \log n \cdots \log n) \geq f(n)$

$\Rightarrow 1.n \log n \geq f(n)$

$c.g(n) \geq f(n)$

$\therefore O(n \log n)$ Ans

(ii) $6.2^n + n^2 = f(n)$

$\Rightarrow 6.2^n + 2^n \geq f(n)$

$\Rightarrow 7.2^n \geq f(n)$

$c.g(n) \geq f(n)$

$\therefore O(2^n)$

(iii) $f(n) = 1000n^2 + 100n - 6$

$f(n) \leq 1000n^2 + 100n^2 - 6n^2$

$f(n) \leq 1094 n^2$

$f(n) \leq c.g(n)$

$O(n^2)$

(iv) $f(n) = \dfrac{6n^3}{\log n + 1}$

$\Rightarrow \dfrac{6n^3}{\log n + \log n} \geq f(n)$

Here nominator needs to be big so n^3.

denominator needs to be small. so lowest $\log n$

$\Rightarrow \dfrac{3 \, 6n^3}{2 \log n} \geq f(n)$

$\Rightarrow 3 \dfrac{n^3}{\log n} \geq f(n)$

$\Rightarrow c.g(n) \geq f(n)$

$\therefore O(n^3/\log n)$ Ans

2(a) $T(n) = \begin{cases} a & , n = 1 \\ 2T(n/2) + cn & n > 1 \end{cases}$

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

$T(n/2) = 2T(n/4) + cn/2$

$T(n/4) = 2T(n/8) + cn/4$

$= 2(2T(n/4) + cn/2) + cn$

$= 2^2 T(n/4) + 2cn$

$= 2^2 (2T(n/8) + cn/4) + 2cn$

$= 2^3 T(n/8) + 3cn$

$\vdots$

$2^k T(n/2^k) + \frac{k(k-1)}{2} kcn$

$n/2^k = a1$

$\Rightarrow n = 2^k \cdot a1$

$\Rightarrow \log n = k$

$T(n) = 2^{\log n} \cdot a + \frac{\log n \cdot cn}{a}$

$= na + \frac{cn \log n}{a}$

$= na + cn \log n$

$O(n \log n)$ Ans

2(b)

Branch and Bound	Backtrack
i) It is used to solve optimization problem	i) It is used to find all possible solutions available
ii) It may traverse the tree in any manner DFS/BFS	ii) It traverses the state space tree by DFS
iii) When it realizes that it has a better solution it abandons the pre solution	iii) When it realizes that it has made a bad choice, it goes back and undoes the last choice.
iv) It completely searches the sst for optimal solution.	iv) It searches the sst until it has found a solution.
v) It involves a bounding function	v) It involves feasibility function.

(ii)

Brute force	Backtracking
i) we check all possible combinations	i) we ~~only~~ don't consider all combinations
ii) Finds best and feasible solution by performing exhaustive search.	ii) Checks at each step if it is correct or not, if it is correct than goes further if not then changes path
iii) can be very ~~not~~ slow	iii) can be faster than backtracking
iv) No constraints are applied.	iv) Checks by the applied constraints.

2(c)

Constraint means something that imposes a limit or restriction that prevents something from occuring.

In backtracking constraints help from checking all possible solutions. Through the help of constraints the backtracking algorithm choices the solution, if it has made a bad choice it goes back and choices a new path. This decision to make whether it has choosen a bad way is done by constraints. The constraints in N-queen problem.

(explicit)

i) No two queen can be placed in the same row (imp)

ii) No two " " " " " same column (imp)

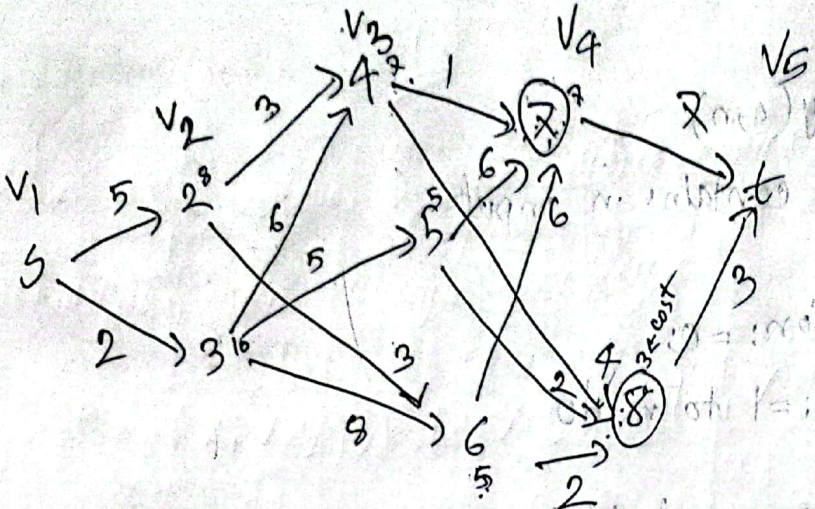
iii) No " " " " " " same diagonal

3(b)

```
Greedy(a,n)
// a[1:n] contains n inputs.
{
    solution := 0;
    for i := 1 to n do
    {
        x := select(a);
        if feasible(solution, x) then
            solution := Union(solution, x);
    }
    return solution;
}
```

3(c)

A problem is said to satisfy the principle of optimality if the subsolutions of an optimal solution of the problem are themselves optimal solution for their subproblems.



Forward approach:

$$c[i,j] = \min_{l} \{ \text{cost}(j,l) + \text{cost}(i+1,l) \}$$

(stage, vertex; next stage, connected vertex)

vertex	s	2	3	4	5	6	7	8	t
Cost	12	8	10	7	5	5	7	3	0
dest	3	6	5	8	8	8	t	t	

$c(4,7) = (7+0)$

$c(4,8) = (3+0)$

$c(3,4) = (8,7) = 7$

$c(3,5) = (13,5) = 5$

$c(3,6) = (13,5) = 5$

path = $s \to 3 \to 5 \to 8 \to t$

4(a)

Approximation algorithm are effecient algos that find approximate solutions to optimization problems (NP-hard). (see 2K12 for details)

Why need approximate?

→ An appro algo gurantees to run in polynomial time though it doesn't guaranteee the most effective solve.

→ It guarentees to seek out high accuracy and top quality solution.

→ Used for problems where exact polynomial time algo are known but are too expensive due to input size.

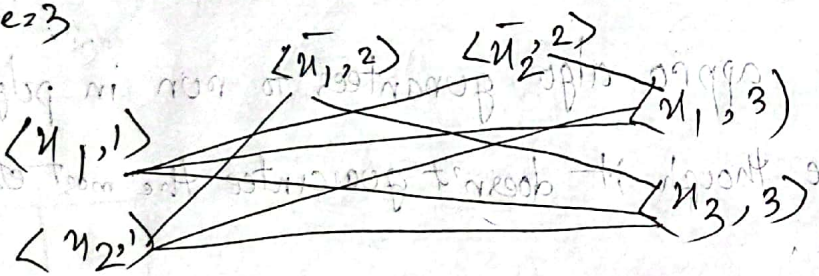
4(b)

let us declare a conjective normal form formulae for clique problem,

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

c_1 c_2 c_3 clauses.

prepare a graph which is having a clique size=3



$$G = \{\langle a, i\rangle \mid a \in c_i\}$$

$$E = \{\langle\langle a,i\rangle, \langle b,j\rangle\rangle \mid i \neq j,\ b \neq \bar{a}\}$$

$k = 3$ for all nodes.

For taking a formulae of satisfiability of k clauses then definitely a k size clique will be there.

solve → x_2 x_1 x_3
1 0 1

There each node had 3 degree with clique size-3

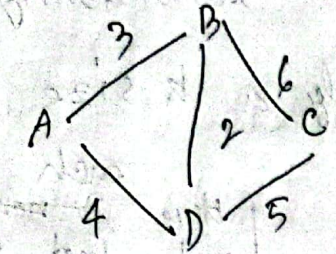
So, Clique is a np-complete

4(d)

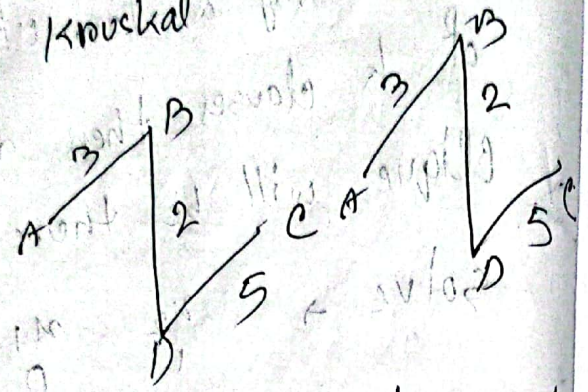
If a graph has a unique MST then both kruskal and prims will produce the same MST. However if a graph has multiple MST then both algo might produce different MST.

For example—

Kruskal



Prim

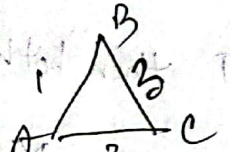


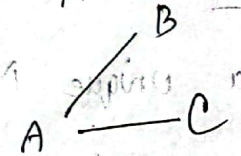
Here the graph has one MST. So the algos produ

same MST.

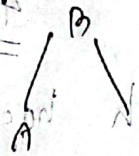
(b)

Kruskal





Prim



Here, The graph has more than one MST. So the algos might produce different or unique MST.

5 (a)

Representing a graph with adjacency lists we combines matrix with edge list. For each vertex i, store an array of the vertices adjacent to it.

memory → O(V+E)

If adjacency matrix we store the edges in a 2D matrix fashion. where row and columns are nodes. And value of A[i,j] = 1 means a edge between i and j and 0 means no.

memory → $O(V^2)$

List

Advantages

i) Fast to add new edge (O(1))

ii) Fast to delete a node (O(1))

iii) Fast to iterate over all edges

iv) memory less used.

Disadv

i) Slow to check if a edge exist or not.

ii) If edges are a lot then memory is used a lot.

Matrix

Adv

i) Fast to add a new edge O(1)

~~ii) Slow to delete a node.~~

iii) Fast to check if a edge exists

iv) Useful for dense graph or a lot of edges.

Disadv

i) Uses a lot of memory $O(n^2)$

ii) Slow to delete a node

BFS will visit the least number of nodes.

queue = A adj = B, C node = 1

~~source~~ = A

queue = A B C node = 2

queue = A B C D E u = 3

u = A B C D E F G

queue = A B C D E F G H I nodes = 4

u = A B C D E F G H I J n = 5

5 nodes visited before going to visit F.

If we apply dfs then the left subtree will be traversed first then it will go to the right subtree where it will find F.

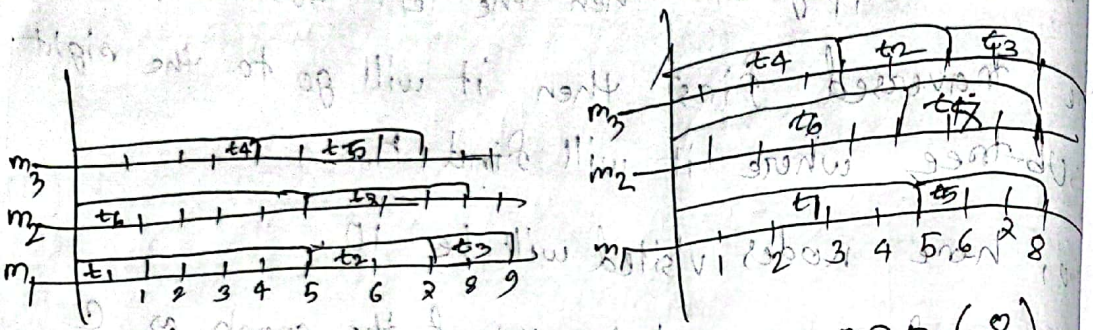
So, here nodes visited will be 10.

However if we represent the list of the graph as, C before B. Then it will go to the right subtree and visit F even faster than BFS with node taken of just 2. But if we don't know, how the input is given then BFS is our best way.

5(c)

sort in dec

$(5,5,4,3,3,2,2) = (t_1, t_6, t_4, t_5, t_7, t_2, t_3)$



LPT (9)

Opt: (8)

6(a)

parent →

0	A	A	B	E	D	E
	A	B	C	D	E	G

	A	B	C	D	E	F	G
0	~~2~~	~~3~~	~~12~~	~~∞~~	~~∞~~	~~∞~~	
A	B	C	D	E	F	G	
A	2	3	~~4~~	12	∞	∞	
	B	C	D	E	F	G	
A	2	3	4	12	~~6~~	∞	
	B	C	D	E	F	G	
A	2	3	4	12	6	~~8~~	
	B	C	D	E	F	G	
A	2	3	4	~~12~~ 9	6	8	
	B	C	D	E	F	G	

pq = ~~A~~

pq = ~~B~~ C E

pq = ~~C E D~~ D E C

pq = ~~F~~ C E

pq = ~~G~~ C E

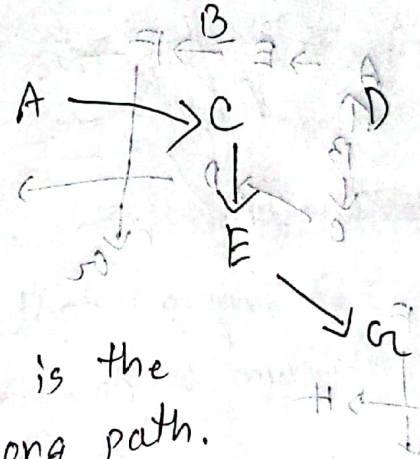
pq = ~~E~~

pq = ~~C E~~

pq = ~~E~~

	A	B	C	D	E	F	G
A		2	3	4	9	6	~~8~~ 2
		B	C	D	E	F	G
A		3	3	4	9	6	~~2~~
		B	C	D	E	F	G

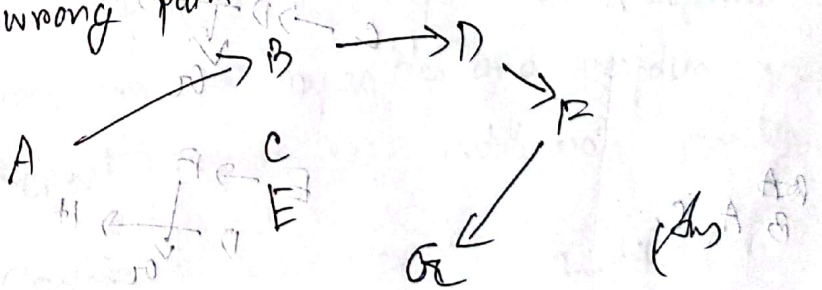
(11) wrong path was computed for G.



(Path for G)

This is the wrong path.

The correct path will be before the last iteration or before priority queue had only E. As E → G is giving wrong path.



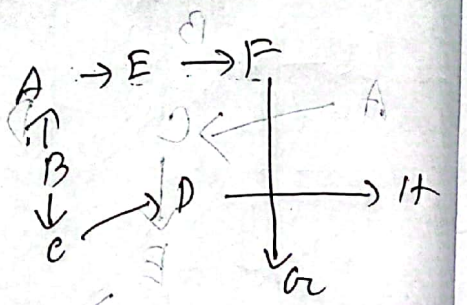
iii) We need to remove $E \to G$.
(Show again the Dijkstra).

For last edge we get the wrong path.

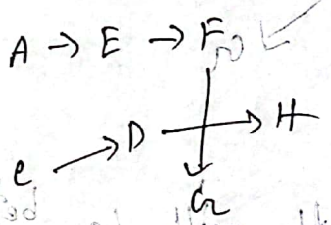
$E \to G$

(b)

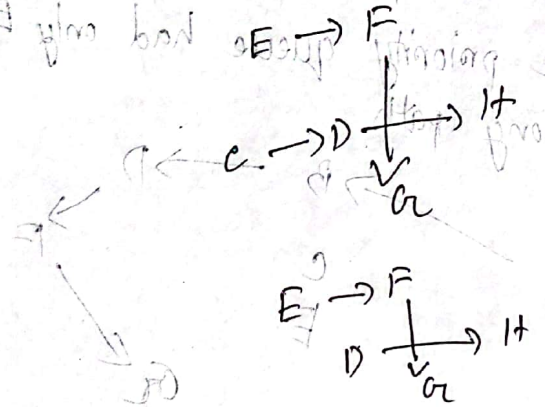
1) Find a node with in-deg = 0
ii) Remove from graph.
iii) Add to ordering.



order = B



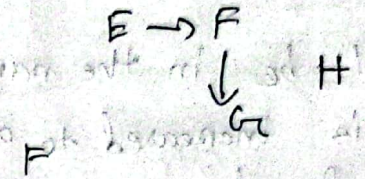
order = B A

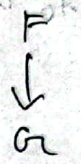


B A C

B A C D

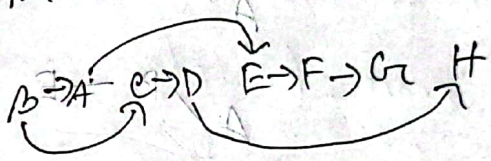
B A C D E





B A C D E F

G, H



(Ans)

Since it's lexicographical order so there won't be multiple answer. But without lexicographical order there will be multiple answers.

G (c)

i) Initialize flow of all edges to 0.

ii) Randomly select a path and augment it. Update flow, residual capacities and residual graph.

iii) Repeat process (ii) until not possible.

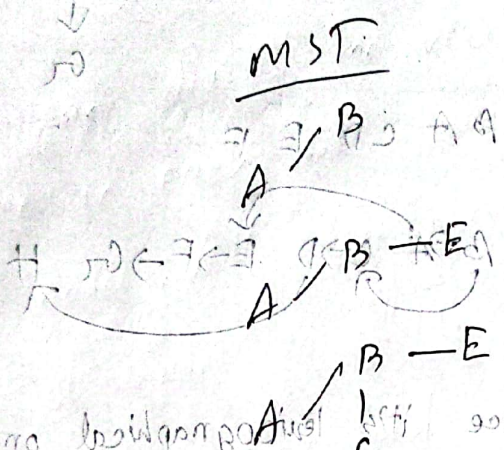
(Consider reverse paths always)

2 (a)

x will be in the range of (1 to 8) if x is increased to 9 and above another MST will be found.

edge	cost
(A, B)	1
(B, E)	3
(B, C)	5
(E, D)	5
(A, F)	6

MST



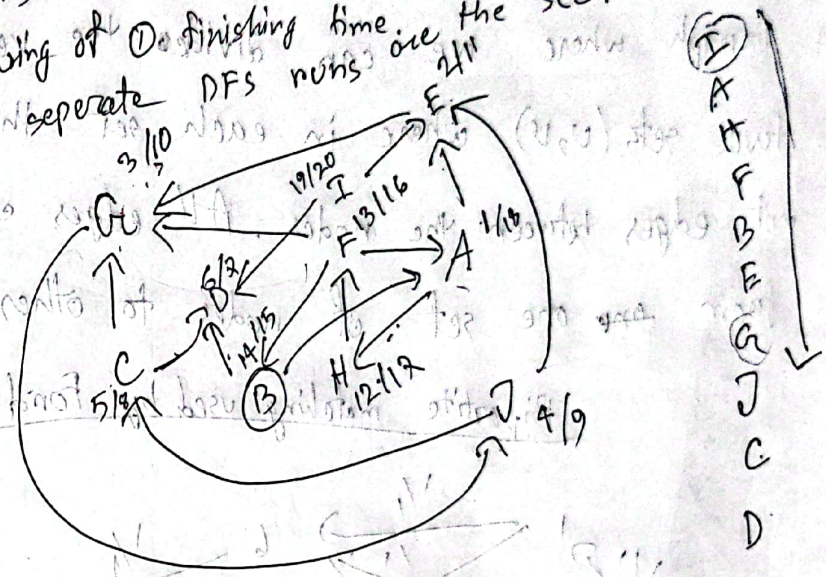
2 (b)

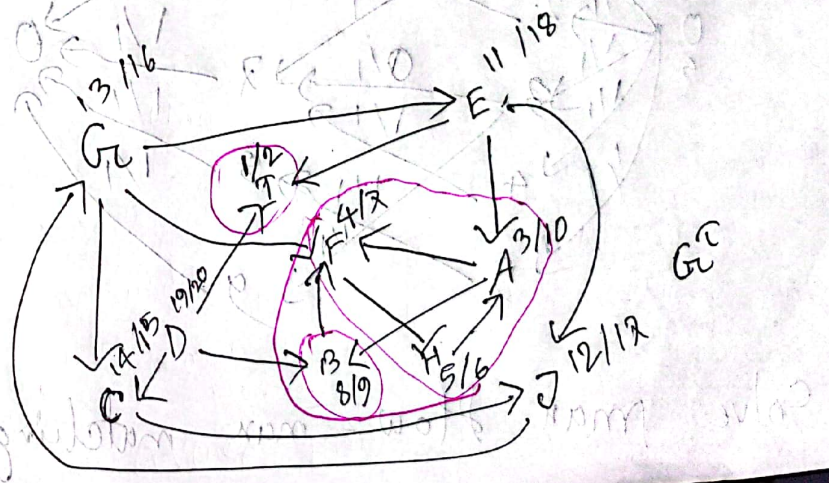
① DFS → ending time decreasing order (or put in stack to find easily)

② G^T

③ DFS run on the G^t based on the stack of ① on decreasing of ① finishing time.

④ The 2 seperate DFS runs are the scc.



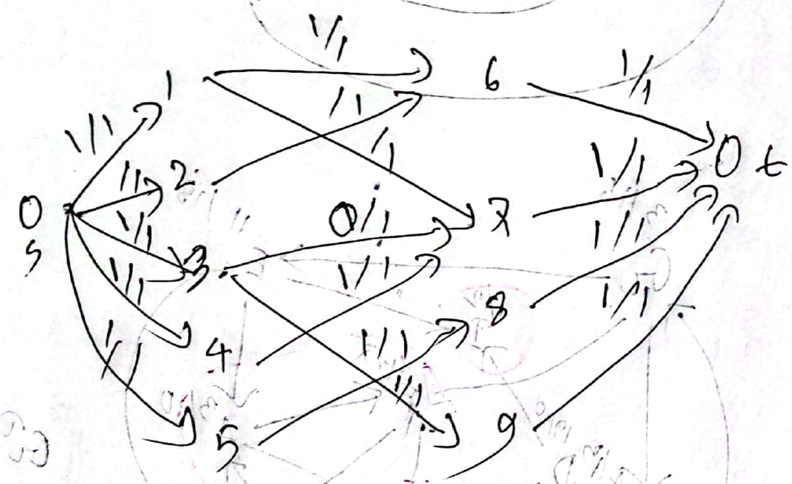


sce = {I}, {A F H B}, {E J C G}, {D}

2(c)

A Graph where we can divide the vertex into two sets (U, V) where in each set there is not edges between the nodes. All edges connect from ~~one~~ one set of nodes to other.

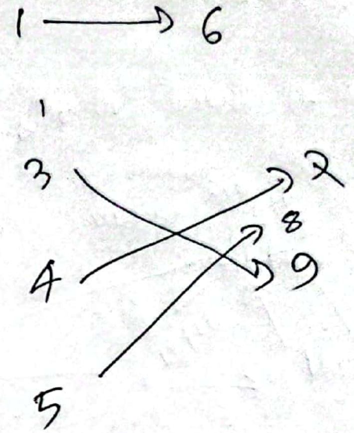
Bipartite matching used by Ford



Solve max. flow = max matching

max flow = 4,

Paths



8(b) see 2K12

8(a)