2K17

1@

An algorithm is a finite set of instruction that is followed to accomplish a particular task.

Basic criteria:-

Input:- Zero or more quantities are externally supplied.

Output:- At least one quantity must be produced.

Definiteness:- Each instruction is clear and unambigious

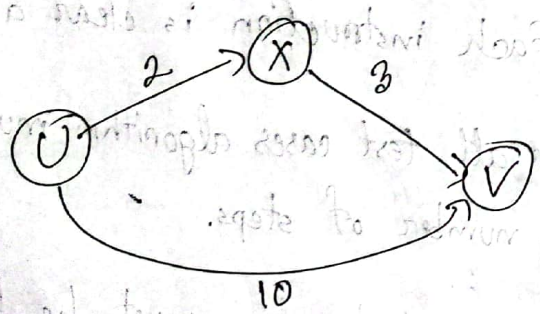
Finiteness:- For all test cases algorithm must terminate within finite number of steps.

Effectiveness:- Every instruction must be basic so that it is carried out in principle.

1(b)

1. Overlapping subproblem:- Break the problem into small subproblem and store the result of those subproblems which we will need later.

2. Optimal substructure:- A given problem has optimal substructure property if the optimal solution of the problem can be obtained from it's subproblem.



Here the problem is to go from U to V. But Breaking it to U→X then X→V will give optimal solution. So we solve those subproblems to get the result.

steps:-

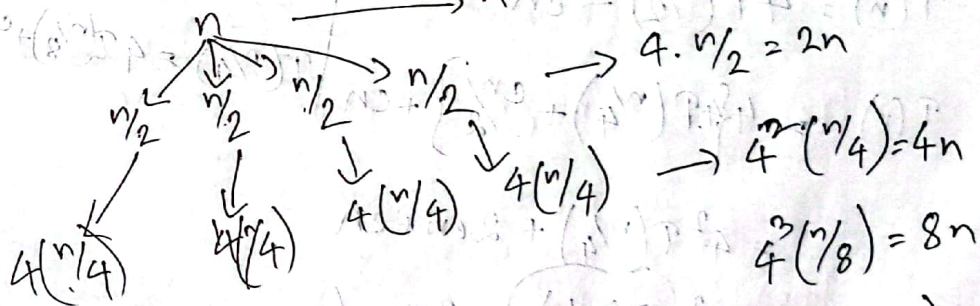
i) Characterize the structure of optimal sol^n.

ii) Recursively define the value of an u 4

iii) Compute the value a u u

iv) Construct optimal sol^n. from

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



we know,

$\frac{n}{2^k} = 1$

$\Rightarrow n = 2^k$

$\Rightarrow k = \log n$

$\therefore T(n) = 4^k T(1) + (n + 2n + 4n + \ldots 2^{n-1} \cdot n)$

$= 4^{\log n} T(1) + n(1 + 2 + 4 + \ldots 2^{n-1})$

now,

$4^{\log n} = x$

$\Rightarrow 2^{2\log n} = x$

$\Rightarrow \log x = 2\log n$

$\Rightarrow x = n^2$

$\downarrow$

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

$\Rightarrow n^2 + n \cdot \frac{n-1}{1}$

$\approx n^2$

with substitution.

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

$T(n/2) = 4\{T(n/4)\} + \frac{cn^2}{2}$

$T(n/4) = 4T(n/8) + \frac{cn^2}{4}$

$T(n) = 4\{4 \cdot T(n/4) + cn^2/2\} + cn$

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

$= 4^2 T(n/4) + 3cn$ ✓

$= 4^2 \{4 \cdot T(n/8) + cn/4\} + 3cn$

$= 4^3 \cdot T(n/8) + 4cn + 3cn$

$= 4^3 \cdot T(n/8) + 7cn$

$\vdots$

$4^k T(n/2^k) + (2^k - 1)cn$

let,

$n/2^k = 1$

$\Rightarrow n = 2^k$

$\Rightarrow \log_2 n = k$

$2^{\log_2 n} =$

so,

$T(n) = 4^{\log_2 n} T(1) + (2^{\log n} - 1)cn$

$= n^2 + (n-1)cn$

$= n^2 + cn^2 - cn$

$\approx O(n^2)$ (Proved)

12

2(a)

i) $1000n^2 + 16n + 2^n = f(n)$

The biggest term $= 2^n$.

Replace all the n terms with 2^n,

$f(n) \leq 1000 \cdot 2^n + 16 \cdot 2^n + 2^n$

$\Rightarrow f(n) \leq 1017 \cdot 2^n$

$\Rightarrow f(n) \leq c \cdot g(n)$

$\therefore g(n) = 2^n$

$\therefore O(2^n)$ (Ans)

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

ii) $\log(n) + 10000 = f(n)$

$f(n) \leq$ ~~$10000 + \log n$~~

$10000 \times \log n + 10000$

$f(n) \rightarrow (10000) \log n$

$c \qquad g(n)$

~~$O(n)$~~ $O(\log n)$

iii) $50n + n\log n^2 + 1000\log(n) = f(n)$

$\Rightarrow f(n) \leq 50n\log n^2 + n\log n^2 + 1000\log n^2 . n$

$\Rightarrow f(n) \leq 100\, n\log n + 2n\log n + 2000 n\log n$

$\Rightarrow f(n) \leq 2102 n\log n$

$\Rightarrow f(n) \leq c\, g(n)$

$g(n) = n\log n$

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

iv) $2^{20} + 3^2 = f(n)$

$\Rightarrow f(n) \leq 2^{20} \cdot n^0 + 3^2 \cdot n^0$

$\Rightarrow f(n) \leq n^0 (2^{20} + 3^2) n^0$

$\Rightarrow f(n) \leq c \cdot g(n)$

$\therefore g(n) = n^0 = 1$

$\therefore O(1)$ Ans

2 (c)

performance analysis is theoretically calculating the time complexity and space complexity of an algorithm.

Performance measurement is measuring the execution time of a program for some definite inputs.

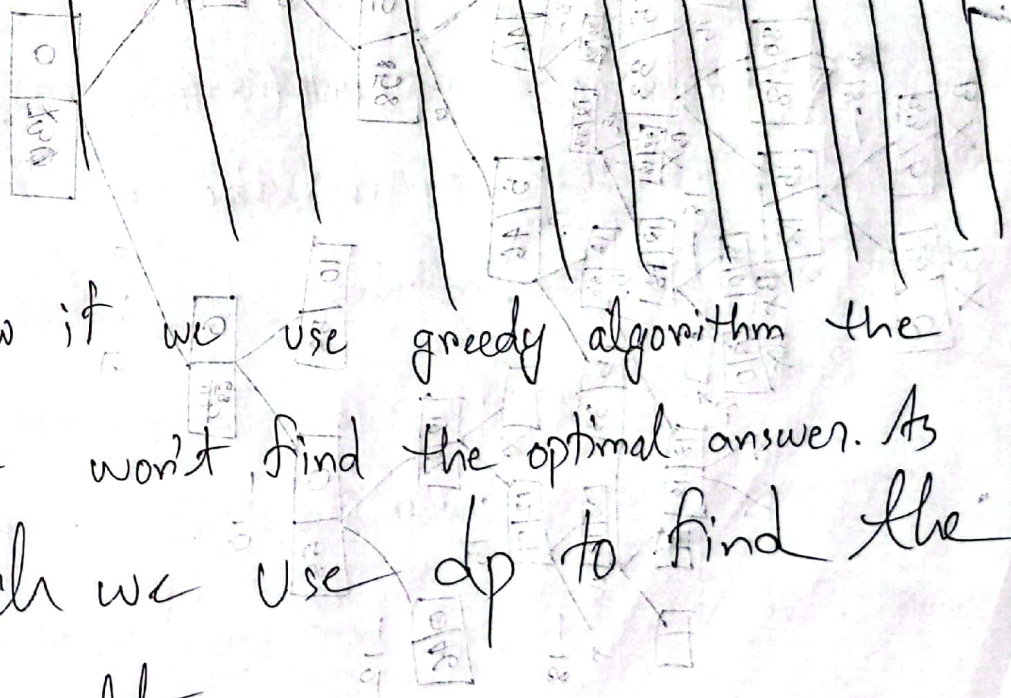
2(d)

Branch	Backtracking
1) It is used to find the optimal combination of solution	1) Used to find all possible solution.
11) Finds the solution by dividing the problem in least new restricted subproblem.	11) Finds the solution by dividing the problem with all possible subproblem and solving them.
111) Uses a general level by level solutions like BFS.	111) Uses a depth first analysis to find the solution like DFS.

3(a)

Cost table

	0	1	2	3	4	5	6	7	8	9	10
(40,10)	0	0	0	0	0	0	0	0	0	0	40
(20,3)	0	6	0	20	20	20	20	20	20	20	40
(30,5)	0	0	0	20	20	30	30	30	50	50	50



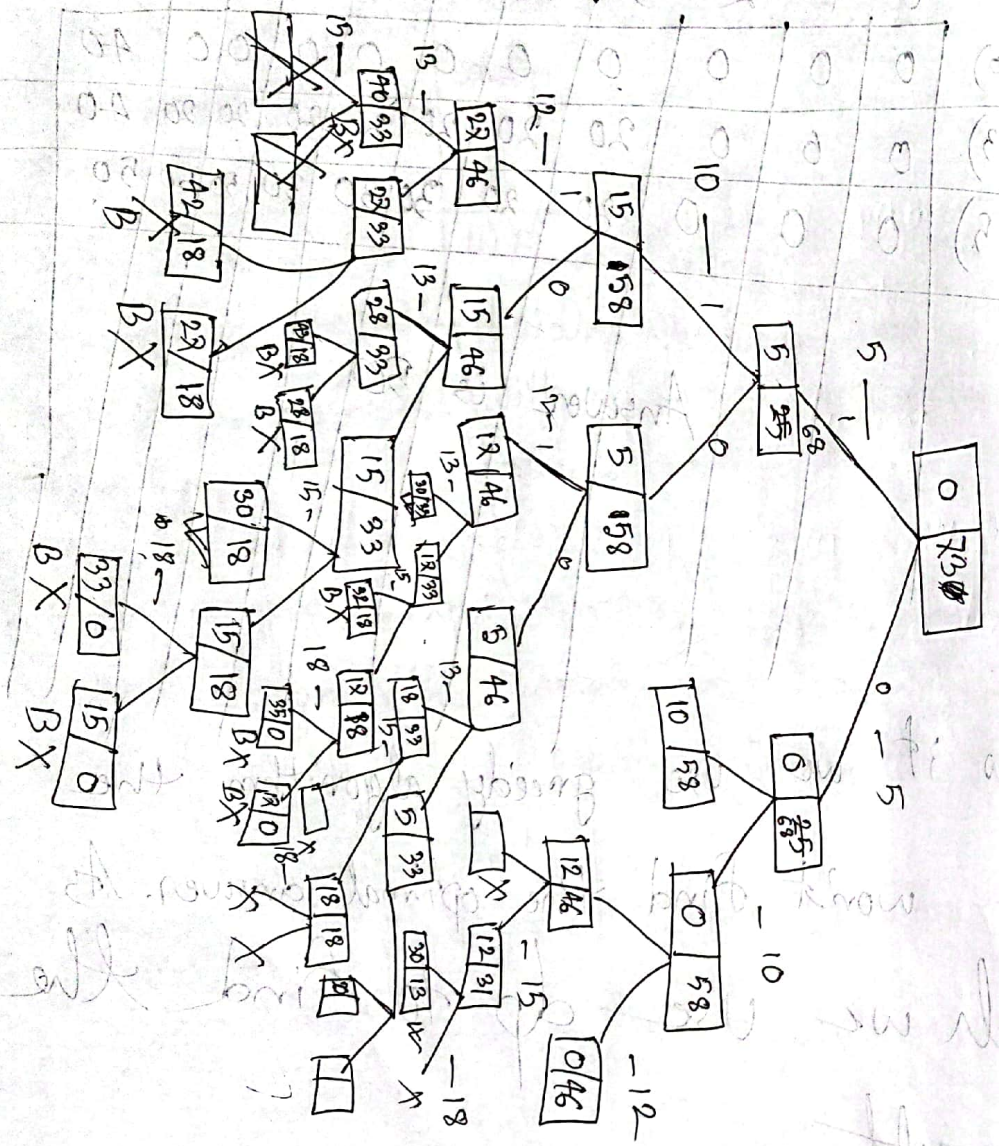
Answer is 50

Now if we use greedy algorithm the we won't find the optimal answer. As such we use dp to find the result.

(b)

$w[1:6] = \{5, 10, 12, 13, 15, 18\}$

Total = 73, sum(m) = 30



solutions found = $\{5, 10, 15\} = \{1\,1\,0\,0\,1\,0\}$

$= \{12, 18\} = \{0\,0\,1\,0\,0\,1\}$

$= \{13, 12, 5\} = \{0\,0\,1\,1\,0\,0\}$ (Ans

(c)

In the N-queen problem, the statement is, we have to place the queens in such a manner that no two queens attack each other. Now for N-queen problem the board is of N×N. Applying constraint will reduce to a solution. Now we know that each queen must be in a different row. Setting the queens in different row creates an explicit constraint and the space tree is N^N. Applying

Some implicit constraint likes

1) No two queens will be on the same column.

11) No two queens will be on the same diagonal.

The first one implies that all solutions are a permutation of n tuple (As tuple number represent rows). This reduces the size from N^N to $N!$ As

(d)

Approximation algorithm are efficient algorithm to find approximate result of optimization problem specially on NP-hard problem. It is not possible to develop a generalize algorithm for NP-hard problem to get result for all inputs. So we use approximation algorithm to get a result which is near to the solution for those kinds of problem we use approximation algorithm.

4(c)

k	1	2	3	4	5
P(k)	0.25	0.20	0.05	0.20	0.30

$\Sigma P(k)$ matrix

	0	1	2	3	4	5
1	0	0.25	.45	.5	.7	1.00
2		0	0.20	.25	0.45	.25
3			0	0.05	.25	0.25 0.55
4				0	0.20	0.5
5					0	0.30
6						0

Cost + Parent matrix

	0	1	2	3	4	5
1	0	.25[1]	.65[1]	.8[1]	1.2[2]	2.05[2]
2		0	.20[2]	.30[3]	.2[2]	1.35[4]
3			0	.05[3]	.25[4]	0.85
4				0	.20[4]	0.75
5					0	.30[5]
6						0

$$c[i,j] = \left\{ c[i,k-1] + c[k+1,j] \right\} + \sum_{t=i}^{j} p(t)$$

$$\min_{i \le k \le j}$$

$c(1,2) = \min(.20, .25) + .45$

$= .20 + .45$

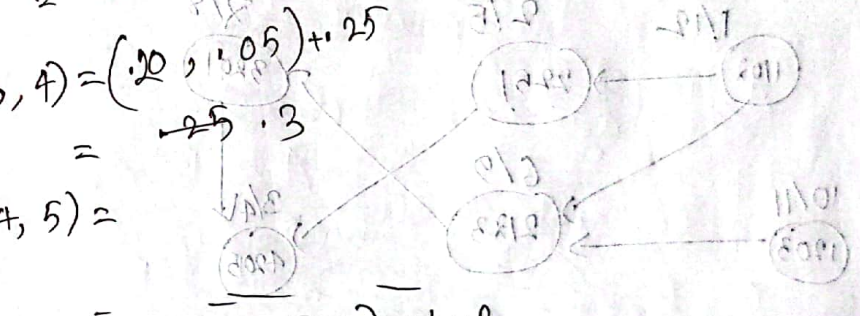
$c(2,3) = (.05, 0.20) + .25$

$= .05 + .25$

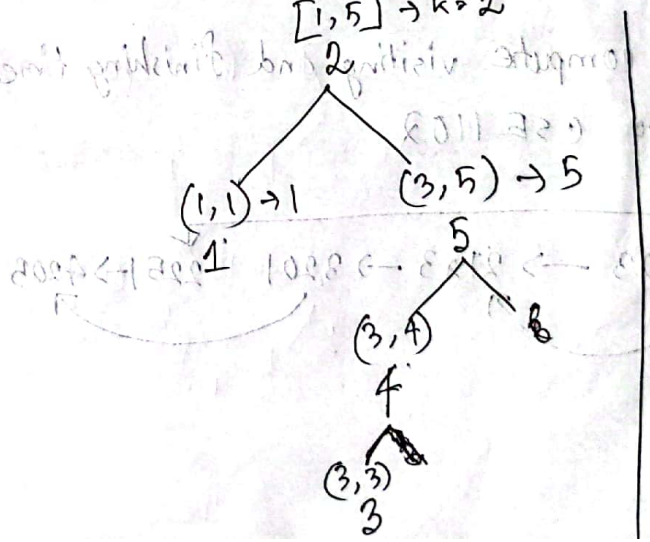
$c(3,4) = (.20, .05) + .25$

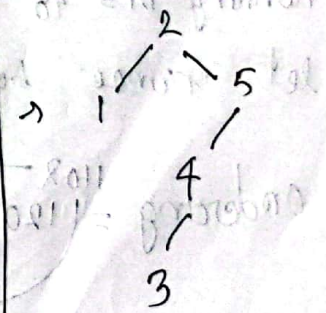
$= .25 \cdot 3$

$c(4,5) =$



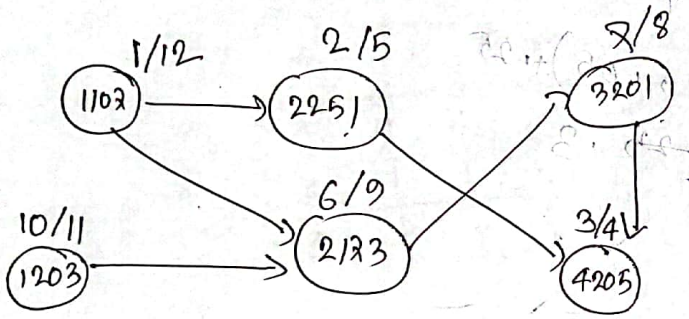
$[1,5] \to k = 2$





5 (c)

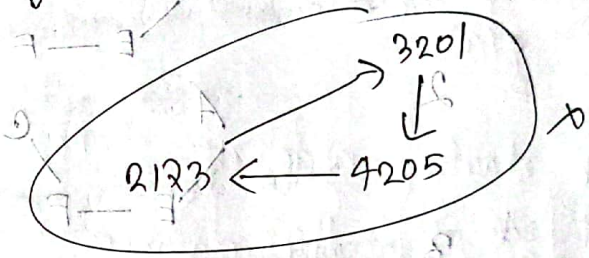
A topological sort of A DAG $G=(V,E)$ is a linear ordering of all its vertices such that if G contains an edge (u,v) then u appears before v in ordering.



Running DFS to compute visiting and finishing time let source be CSE 1102

Ordering = 1102 → 1203 → 2123 → 3201 → 2251 → 4205

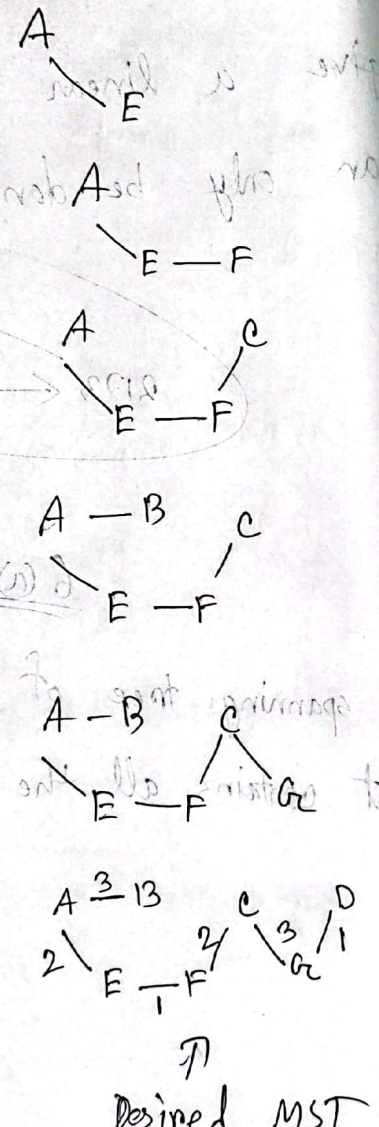
If we draw a new edge in 4205 to 2123 then we will create a cycle. Thus we can't give a linear ordering as topological sort can only be done on a DAG.



6 (a)

A spanning tree of a graph is just a subgraph that contains all the vertices and is a tree.

edges	cost	mst
A-E	2	
E-F	1	
F-C	2	
A-B	3	
C-G	3	
G-D	1	



Desired MST

i) 9

ii) 12

X(c)

Let's consider n items to be packed into bins of specific sizes.

each item has a size.

Assume the bin's capacity to be c.

Problem is to minimize the number of bins.

X 2 5 1 9 4 3 6

B = 10

First Fit:-

1
2 4 6
X 5 9 3

M = 4

Best Fit:-

1
2 4 6
X 5 9 3

M = 4

iii) Next Fit:-

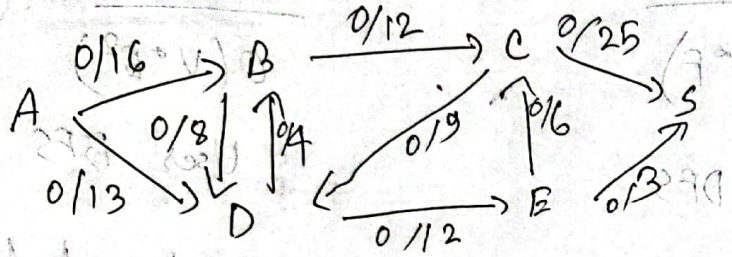
2 1 3
X 5 9 4 6

M2 5

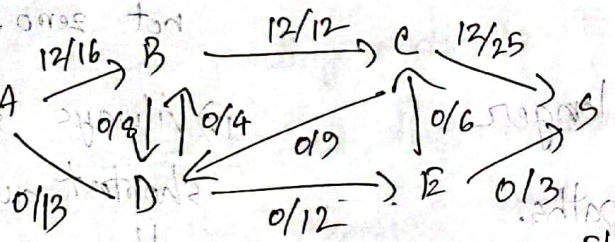
8(a)

If we define starting node as source and ending node as sink, the maximum flow that can be sent through the vertex from source to sink is called max flow.

Ford	Edmond
i) $O(E^*F)$	$O(V^*E^2)$
ii) Uses DFS	Uses BFS
iii) Randomly selects a path for augmenting	iii) Only selects a shortest path if the residual capacity not zero.
iv) can select longer augmenting paths.	iv) Always selects the shortest augmenting paths.
v) Does depend on the capacity value of any edge.	v) Doesn't depend on the capacity value of any edge.

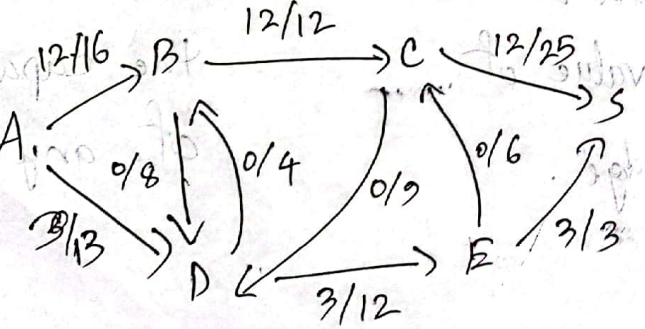


path⇒ A → B → C → S bc⇒ 12 Flow = 12+



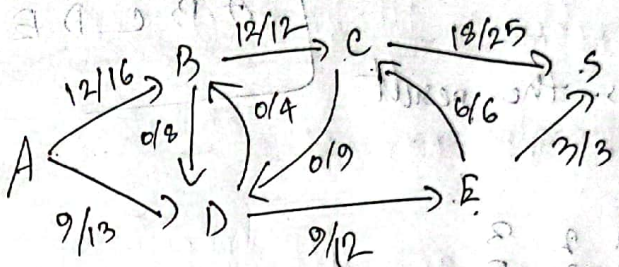
Flow = 12+3

path⇒ A → D → E → S bc⇒ 3



~~path⇒ A → B → D → E → C → S bc⇒~~

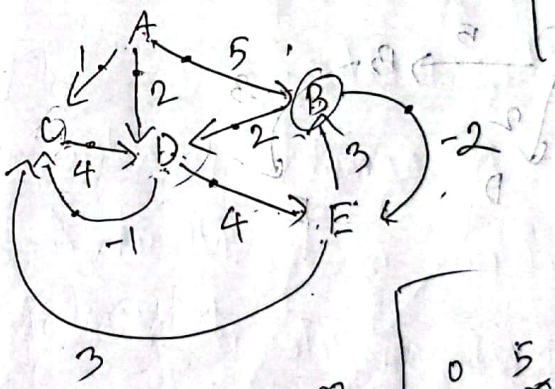
path⇒ A → D → E → C → S bc⇒ 6



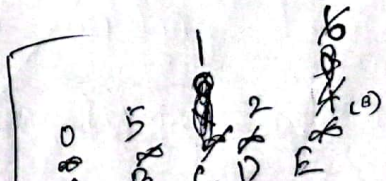
Flow = 12+3+6

max Flow = 21 (Ans)

8(c)



we need at first 4 iterations.



Initial⇒ A ∞ B ∞ C ∞ D ∞ E ∞

After 1st iter ⇒ A 0 B 5 C 1 D 2 E 3

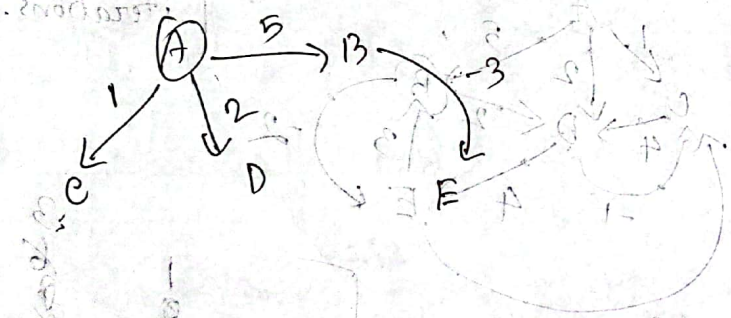
	0	5	1	2	3
After 2nd iter ⇒	A	B	C	D	E

0	5	1	2	3
A	B	C	D	E

After all iterations the result will be,

0	5	1	2	3
A	B	C	D	E

Since no change in 2nd iter all iter will result in same values. so graph will be



5 (a)

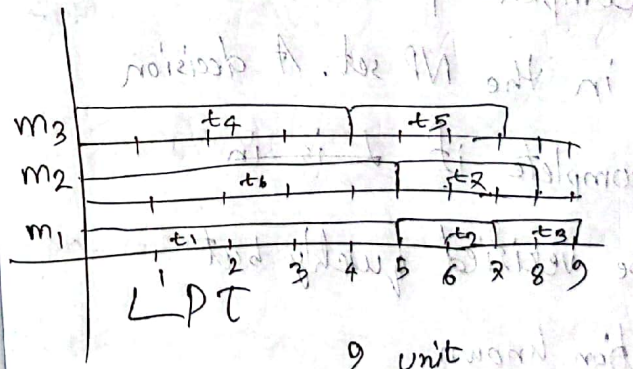
precondition are such condition which needed to be fulfilled to use an algorithm.

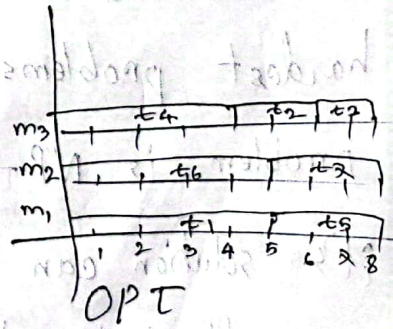
Pre-condition for Bellman-Ford:-

There should not be any negative weighted cycle. in the graph.

(b)

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





9 unit

2(a)

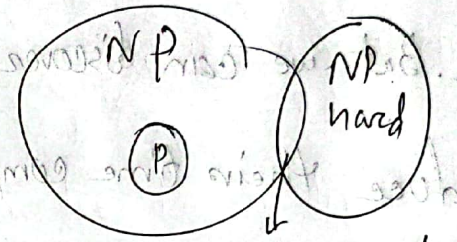
NP-hard:- A problem is NP hard if an algorithm for solving it can be translated into one for solving any NP-problem (non-deterministic polynomial time) or. NP hard therfore means "at least as hard as any NP-problem", although it might in fact be harder.

Ex:- SAT, Halting problem.

NP-complete:- NP complete problems are the hardest problems in the NP set. A decision problem is NP-complete if ~~L is in~~ its solution can be verified quickly but no efficient solution known.

also problems can be reduced to polynomial time.

Ex:- Vertex, Hamiltonian cycle.



(b)

P problems are problems that can be solved in polynomial time. example:- Multiplication, searching, sorting etc.

NP problems are problems that are solved by non-deterministic algorithm in exponential time but their solution can be verified easily by polynomial time. Ex:- TSP, knapsack 0/1.

NP problems are solved by non-deterministic algorithms, which means we don't know the statements 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.