SECTION A

(Answer **ANY THREE** questions from this section in Script A)

1. a) What is an algorithm? What are the basic criteria that an algorithm must satisfy? (1
   b) Describe the steps in applying dynamic programming strategy when developing an algorithm. (1
   c) Draw the recursion tree for $T(n) = 4T(n/2) + cn$, where c is a constant. Provide a good asymptotic upper bound on its solution and verify your bound by substitution method. (1

2. a) Write down the best Big-oh (O) characterization for each of the following running time estimates of different algorithms. (1
      i) $1000n^2 + 16n + 2^n$
      ii) $\log(n) + 10000$
      iii) $50n + n\log(n^2) + 1000\log(n)$
      iv) $2^{20} + 3^7$
   b) Write pseudo code for a backtracking algorithm to solve m coloring problem. (10
   c) What are the differences between performance analysis and performance measurement of an algorithm? (07
   d) What are the differences between branch-and-bound and backtracking paradigm? (06

3. a) Consider a 0|1 knapsack problem where knapsack *capacity* $(m) = 10$, *weight* $= \{10, 3, 5\}$ and *profit* $= \{40, 20, 30\}$. Now explain the application of dynamic programming and greedy algorithm to find an optimal solution. (10
   b) Consider a sum of subset problem with six items $(n = 6)$ and $sum(m) = 30$. The items are $w[1:6] = \{5, 10, 12, 13, 15, 18\}$. Now draw a state space tree for the above problem using backtracking approach. (10)
   c) Explain how the solution space is reduced from N^N to $N!$ in N-Queen problem. (09)
   d) What do you mean by approximation algorithm? Why do we need approximation algorithm? (06)

4. a) Write the features of a divide and conquer paradigm. How will you compute time complexity of a divide and conquer paradigm? (10)
   b) What do you mean by greedy choice property? What are the characteristics of the optimization problem, in the context of greedy algorithm? Write down an algorithm for solving fractional knapsack problem using greedy method. (10)
   c) Consider the following table where p is the probability and k is the key value. Construct Optimal Binary Search Tree (OBST). (15)

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

SECTION B

(Answer **ANY THREE** questions from this section in Script B)

5. a) What is the pre-condition? Write down the pre-condition of Bellman-Ford algorithm and explain. (07)
   b) In scheduling independent tasks problem, let the number of processor, $m = 3$, number of tasks $n = 7$, where $(t_1, t_2, t_3, t_4, t_5, t_6, t_7) = (5, 2, 2, 4, 3, 5, 3)$. Schedule the tasks by Longest Processing Time (LPT) rule and then find the time difference between LPT scheduling and optimal scheduling. (13)

c) What is topological sort? The directed graph given in Fig. 5(c) shows the prerequisite relation among courses of different semesters. Give a linear ordering of nodes of the graph showing visiting time and finishing time for each node. Draw your own opinion about whether it is possible to give a linear ordering of nodes if a new directed edge is inserted in the graph from the node CSE 4205 to CSE 2173. (15)

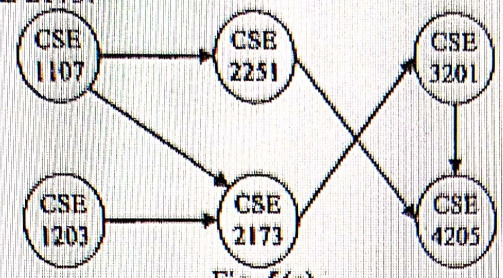


Fig. 5(c)

6. a) What is spanning tree? Consider the following undirected graph where the value of each edge represents the length of that edge in Fig. 6(a). Apply Prim's algorithm. (13)
   i) What is the length of the shortest path between A and D?
   ii) What is the length of the edges in a minimum spanning tree?

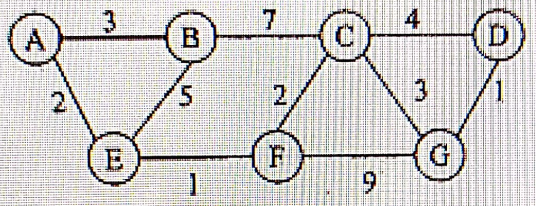


Fig. 6(a)

b) What is Lower Bound? Derive a Lower Bound for three elements insertion sort using comparison tree. (15)

c) Define Least Cost (LC) search. How LC search can be converted into BFS and DFS? (07)

7. a) Define *NP*-hard and *NP*-Complete problems using example(s). (07)
   b) Justify the statement – "All *P* problems are always *NP*". (10)
   c) Explain bin packing problem. Provide an example. For a bin size, $B = 10$, pack the bins with the following weighted items. (18)

Items	7	2	5	1	9	4	3	6

Follow:
   i) First Fit
   ii) Best Fit
   iii) Next Fit

8. a) What do you understand by max-flow network? (05)
   b) What are the differences between Ford-Fulkerson and Edmonds-Karp algorithm? Apply Ford-Fulkerson algorithm for the network in Fig. 8(b), (20)

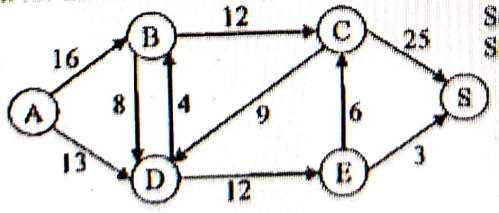


Fig. 8(b)

Find (i) Max flow (ii) Min cut.

c) Find the shortest path from the following graph in Fig. 8(c) by using Bellman-Ford algorithm. (10)

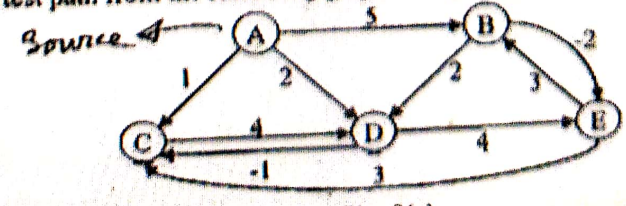


Fig. 8(c)

2017 algo

(1) An algorithm is a finite set of instructions that, if followed, accomplishes a particular task. In addition, all algorithm must satisfy the following criteria.

(1) Input: Zero on more quantities are externally supplied.

(2) Output: At least one quantity is produced.

(3) Definiteness: Each instruction is clear and unambiguous.

(4) Finiteness: If we trace out the ~~intru~~ instruction of an algorithm, then for all cases, the algorithm terminates after a finite number of steps.

(5) Effectiveness: Every instruction must be very basic so that it can be carried out, in principle, by a person using only pencil and paper. It is not enough that each operation be definite as in criterion 3; it also must be feasible.

1 (b)

When developing a dynamic-programming algorithm, we follow a sequence of four steps:

① Characterize the structure of an optimal solution

② Recursively define the value of an optimal solution

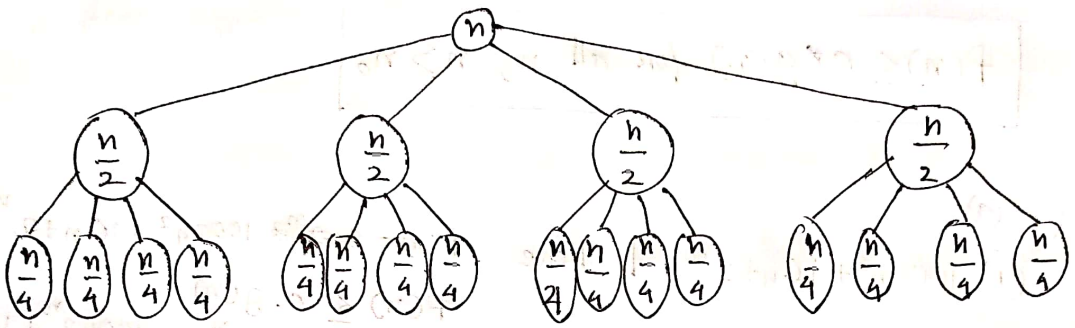
③ Compute the value of an optimal solution, typically in a bottom-up fashion.

④ Construct an optimal solution from computed information.

[· · → P- 359]

1(c) $T(n) = 4T(n/2) + cn$, where c is constant.

Here is an example for $n = 4$



We can see by an easy substitution that the answer is $\Theta(n^2)$. Suppose that,

$$T(n) \leq c' n^2 \quad \text{then,}$$

$$T(n) = 4T(n/2) + cn \leq c' n^2 + cn$$

which is $\leq c' n^2$ whenever we have that $c' + \frac{c}{n} \leq 1$, which, for enough n is true so long as $c' < 1$.

We can do a similar thing to show that it is also bounded below by n^2

Defⁿ: [Big 'oh'] The function $f(n) = O(g(n))$ iff there exist positive constants c and n_0 such that, ~~$f(n) \leq c * g$~~

$$f(n) \leq c * g(n) \text{ for all } n,\ n \geq n_0$$

(2) (a)

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

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

$1000n^2 + 16n + 2^n \leq 1000.2^n + 16.2^n + 2^n$

$1000n^2 + 16n + 2^n \leq 1016.2^n$

$\therefore f(n) = O(2^n)$

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

(ii) $\log(n) + 10000$ | Here, $f(n) = \log(n) + 1000$

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

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

$\log(n) + 1000 \leq \log(n) + 1000\log(n)$

$\log(n) + 1000 \leq 1001\log(n)$

$\therefore f(n) = O(\log n)$

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

$\Rightarrow 2n\log(n) + 50n + 1000\log(n)$

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

Here,

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

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

$2n\log(n) + 50n\log n + 1000n\log n$

$2n.\log(n) + 50n + 1000\log(n) \leq 1057n\log n$

$f(n) \leq O(n\log n)$

(iv) $2^{20} + 3^7$

$\therefore O(1)$

(2)
(b)

```
Algorithm mColoring(k)
// This algorithm was formed using the recursive backtracking
// schema. The graph is represented by its boolean adjacency
// matrix G[1:n, 1:n]. All assignments of 1,2,....,m to
// the vertices of the graph such that adjacent vertices are
// assigned distinct integers are printed. k is the index
// of ~~vert~~ the next vertex to color.
{
  repeat
  {// Generate all legal assignments for x[k]
    NextValue(k); // assign to x[k] a legal color
    if (x[k]=0) then return; // no new color possible
    if (k=n) then                    // At most m colors have been
        write (x[1:n])               // used to color the n vertices
    else mColoring(k+1);
  } untill (false)
}
```

2 (c)

- From a performance point of view, we define two criteria
  - space complexity: the amount of memory needed by a program to run to completion.
  - time complexity: the amount of computer time needed by a program to run to completion.
- Two phases in performance evaluation
  - performance analysis: a priori estimates
  - performance measurement: a posteriori testing

2(d)

[3] (লাল book 308)

3(a) 0/1 knapsack

$S^i = (P_i, W_i)$

capacity (m) = 10

weight = {10, 3, 5}

profit = {40, 20, 30}

$S^0 = \{(0,0)\}$; $S^0_1 = \{(40, 10)\}$

$S^1 = \{(0,0), (40,10)\}$ $S^2_1 = \{(20, 3), (60, 13)\}$

$S^2 = \{(0,0), (40,10), (20,3), (60,13)\}$

$S^2_1 = \{(30,5), (70,15), (50,8), (90,18)\}$

The ~~is~~ optimal solution is (50, 8) ≡ (Profit, Weight)

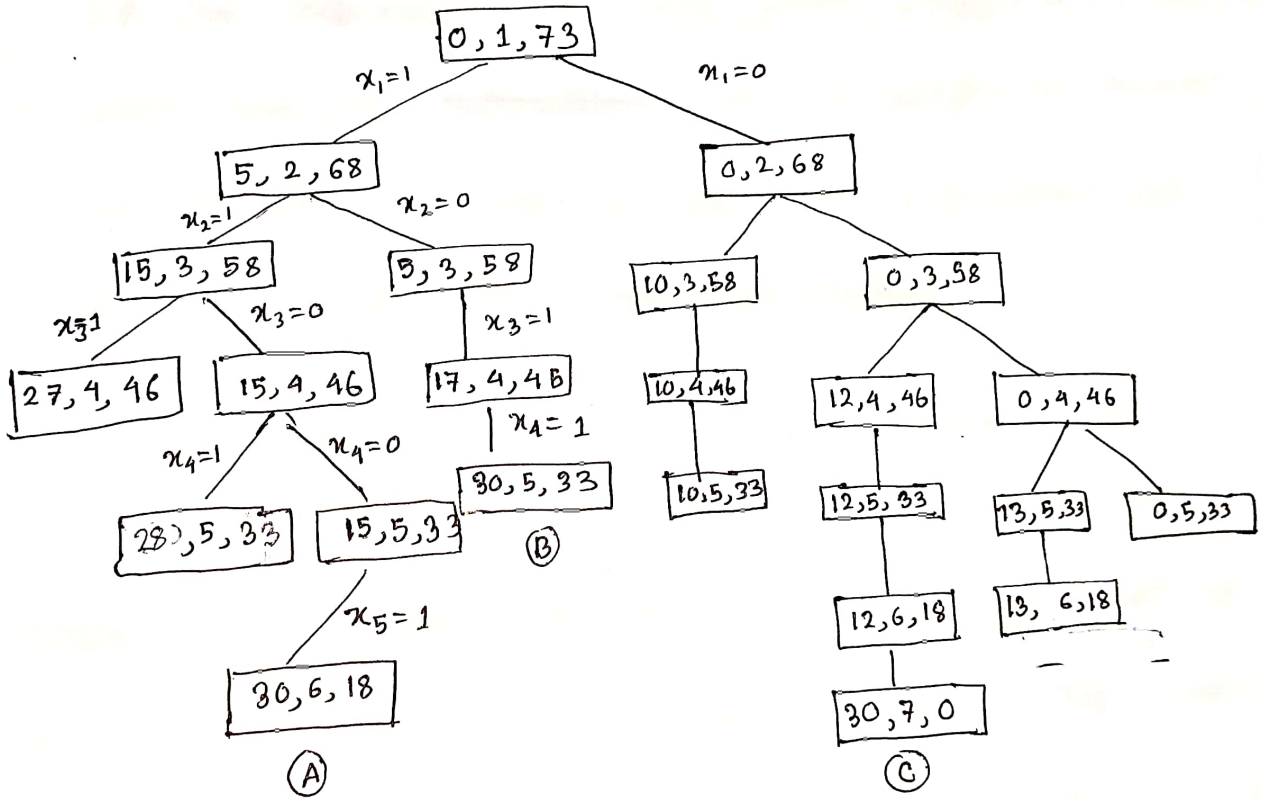
3 (b)

Sum of subset problem with six items ($n=6$) and sum(m)=30.

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

Here, initially $S=0$, $K=1$ and $r = 73$

The state space tree is given below



[For better view see sahni's book - page 358 Example - 7.6 and figure - 7.10 (page 360)]

3 (c)

④

ⓐ The features of a divide and conquer paradigm. (Coremen P-66)

(i) Divide the problem into a number of subproblems that are smaller instances of the same problem.

(ii) Conquer the subproblems by solving them recursively. If the subproblem sizes are small enough *, however, just solve the subproblems in straightforward manner.

(iii) Combine the solutions to ~~be~~ the subproblems into the solution for the original * problem.

(Red book, P-128)

$$T(n) = \begin{cases} g(n) & \rightarrow n \text{ is small} \\ T(n_1) + T(n_2) + \ldots + T(n_k) + f(n) & \rightarrow \text{otherwise} \end{cases}$$

where $T(n)$ is the time for DAndC on any input of size n and $g(n)$ is the time to compute the answer directly ~~from~~ for small input. The function $f(n)$ is the time for dividing P and combining the solutions to subproblems.

The complexity of many divide-and-conquer algorithms is given by recurences of the form,

$$T(n) = \begin{cases} T(1) & n = 1 \\ aT(n/b) + f(n) & n > 1 \end{cases}$$

Where, a and b are known constants. We assume that $T(1)$ is known and n is power of b ($n = b^k$)

(Coneman R499)

④ (b) Greedy-choice property: we can assemble a globally optimal solution by making locally optimal (greedy) choices. In other words, when we are considering which choice to make, we make the choice that looks best in the current problem, without considering results from subproblems.

Characteristics of the optimization problem in context of greedy algorithm:

Greedy fractional knapsack

```
procedure GREEDY_KNAPSACK (P,W,M,X,n)
// P(1:n) and W(1:n) contain the profits and weights respectively
// ly
// objects ordered so that P(i)/W(i) ≥ P(i+1)/W(i+1).
// M is the knapsack size and X(1:n) is the solution vector

real P(1:n), W(1:n), X(1:n), M, cu;
integer i,n;
X ← 0      // initialize solution to zero
cu ← M     // cu = remaining knapsack capacity
for i ← 0 to n do
    if W(i) > cu then exit endif
    X(i) ← 1
    cu ← cu - W(i)
repeat
if i ≤ n then X(i) ← cu : W(i) endif

end GREEDY_KNAPSACK
```

④ c

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

NB: formula,

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

$$i < k \leq j$$

↓ root

i \ j	0	1	2	3	4	5
0	0	0.25	0.65^2	0.90^2	1.45^1	
1		0	0.20	0.30^2	$.75^2$	1.35^4
2			0	0.05	0.30^2	0.85^3
3				0	0.20	0.70^3
4					0	0.30
5						0

First, we will calculate the values where $j - i = 0$

When,

$i = 0$, $j = 0$, then $j - i = 0$

$i = 1$, $j = 1$, then $j - i = 0$

$i = 2$, $j = 2$, then $j - i = 0$

$i = 3$, $j = 3$; then $j - i = 0$

$i = 4$, $j = 4$; then $j - i = 0$

$i = 5$, $j = 5$; then $j - i = 0$

$$C[0,0] = C[1,1] = C[2,2] = C[3,3] = C[4,4] = C[5,5] = 0$$

Now we will calculate values where $j-i=1$

When,
$j=1, i=0$
$j=2, i=1$
$j=3, i=2$
$j=4, i=3$
$j=5, i=4$, for all $j-i=1$

The cost of $C[0,1]$ is 0.25 (it indicates key 1)
" " " $C[1,2]$ is 0.20 (" " key 2)
" " " $C[2,3]$ is 0.05 (" " key 3)
" " " $C[3,4]$ is 0.20 (" " key 4)
" " " $C[4,5]$ is 0.30 (" " key 5)

Again,

When, $j-i=2$

$j=2, i=0$;
$j=3, i=1$;
$j=4, i=2$;
$j=5, i=3$, for all $j-i=2$

for every $j-i=2$ there is two keys and two possible tree. So,

$$C[0,2] = \min \begin{Bmatrix} C[0,0]+C[1,2] \\ C[0,1]+C[2,2] \end{Bmatrix} + w(0,2) = 0.20 + 0.45 \quad [\text{2 as root}]$$
$$= 0.65$$

$$C[1,3] = \min \begin{bmatrix} C[1,1]+C[2,3] \\ C[1,2]+C[3,3] \end{bmatrix} + w(1,3) = 0.05 + 0.25 \quad [\text{2 as root}]$$
$$= 0.30$$

$$c[2,4] = \min \left\{ \begin{matrix} c[2,2] + c[3,4] \\ c[2,3] + c[4,4] \end{matrix} \right\} + w[2,4] = 0.05 + 0.25 = 0.30$$

$$c[3,5] = \min \left\{ \begin{matrix} c[3,3] + c[4,5] \\ c[3,4] + c[5,5] \end{matrix} \right\} + w[3,5] = 0.20 + 0.50 = 0.70$$

Again,

When $j - i = 3$

for every $j - i = 3$, ~~there is~~

$$c[0,3] = \min \left\{ \begin{matrix} c[0,0] + c[1,3] \\ c[0,1] + c[2,3] \\ c[0,2] + c[3,3] \end{matrix} \right\} + w[0,3] = 0.90$$

$$c[1,4] = \min \left\{ \begin{matrix} c[1,1] + c[2,4] \\ c[1,2] + c[3,4] \\ c[1,3] + c[4,4] \end{matrix} \right\} + w[1,4] = 0.75$$

$$c[2,5] = \min \left\{ \begin{matrix} c[2,2] + c[3,5] \\ c[2,3] + c[4,5] \\ c[2,4] + c[5,5] \end{matrix} \right\} + w[2,5] = 0.85$$

Again,

When, $j - i = 4$

for every $j - i = 4$,

$$c[0,4] = \min \left\{ \begin{matrix} c[0,0] + c[1,4] \\ c[0,1] + c[2,4] \\ c[0,2] + c[3,4] \\ c[0,3] + c[4,4] \end{matrix} \right\} + w[0,4] = 1.45$$

(5)(a)

(b)

Here,

number of processor, $m = 3$

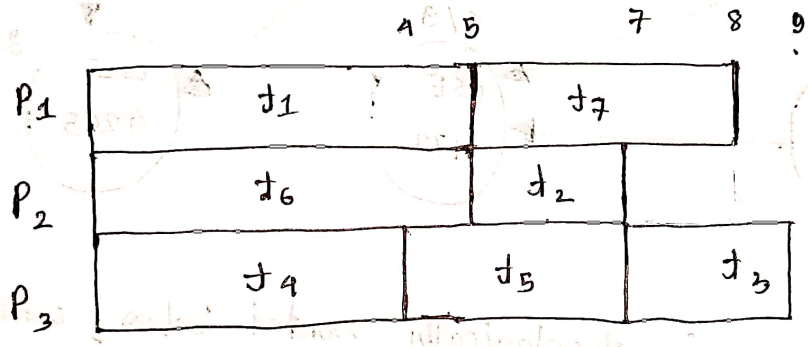
number of tasks, $n = 7$

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

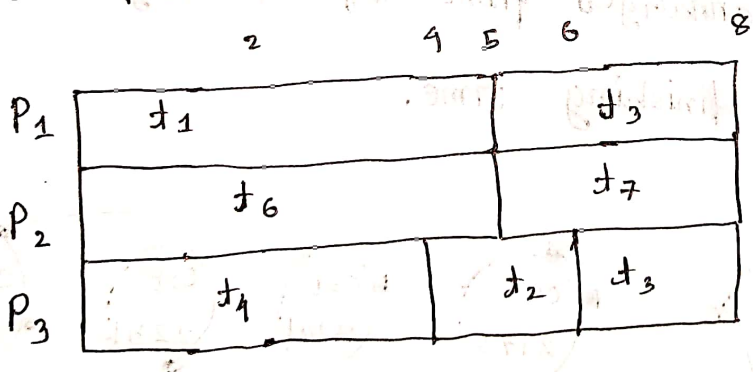
Sorting tasks in decreasing order,

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

The LPT schedule is,



The Optimal schedule is,



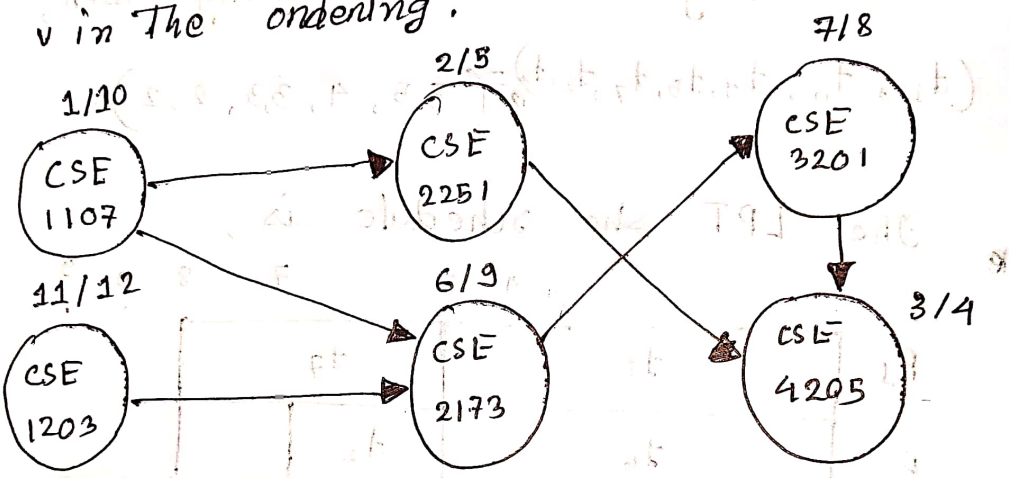
∴ Here, $F^*(I) = 8$ and $\hat{F}(I) = 9$

[optimal finish time] [LPT finish time

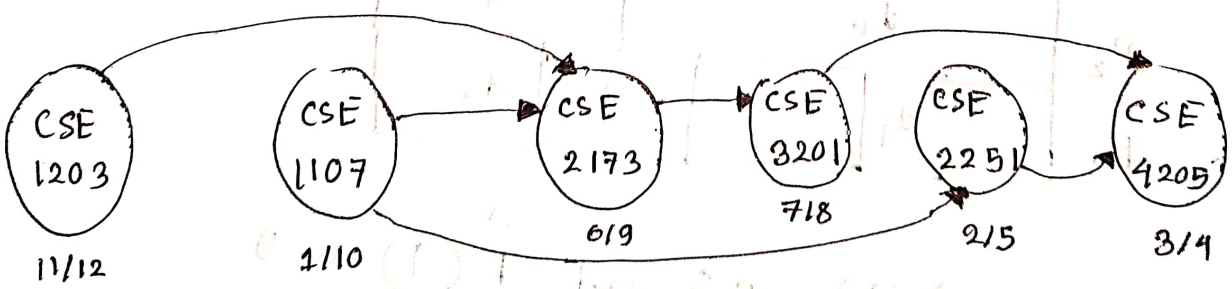
$$\frac{|F^*(I) - \hat{F}(I)|}{|F^*(I)|} = \frac{|9-8|}{8} = \frac{1}{8}$$

(c)

A topological sort of a directed graph is a linear ordering of its vertices such that every directed edge edge $u \to v$ from the vertex u to vertex v, u comes before v in the ordering.



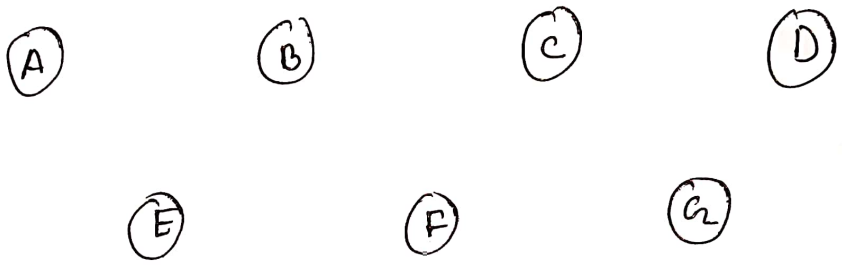
The above graph is topologically sorted below, with its CSE vertices arranged from left to right in order of decreasing finishing time.



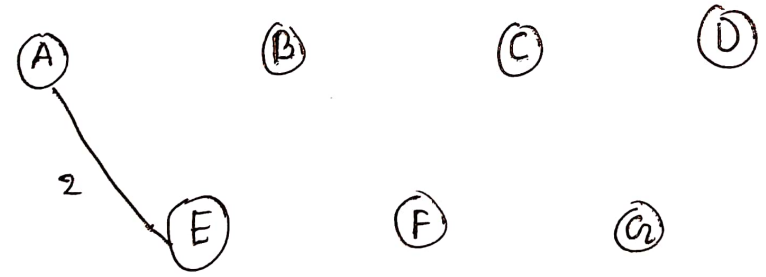
If we draw a new directed edge in the graph from the node CSE4205 to CSE 2173, there will be a cycle in graph. As we know topological sort on linear ordering is only possible in directed acyclic graph, this is not possible to do linear ordering as there is a back edge.

⑥

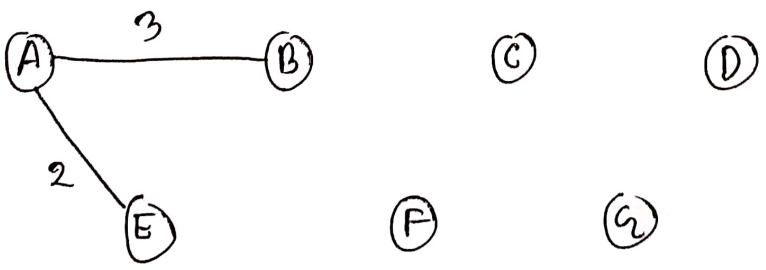
ⓐ A spanning tree is a subset of graph G, which has all the vertices covered with minimum possible number of edges. Hence, a spanning tree does not have cycles and it can not be disconnected



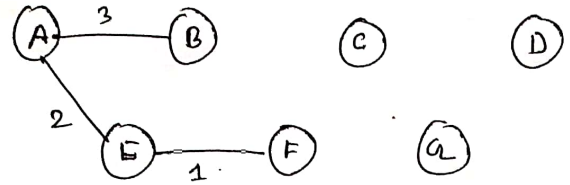
Starting from (A),



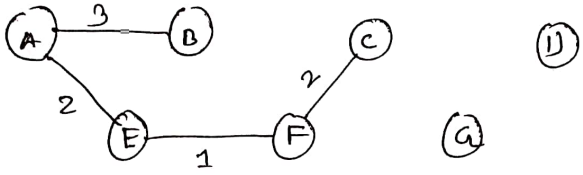
step-1



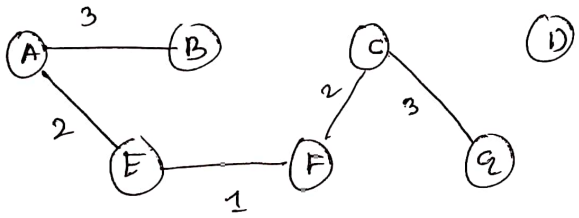
step-2



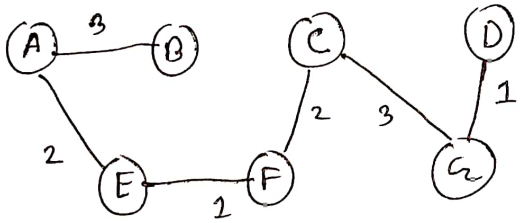
step-3



step-4



step-5



step-6

(i) length of sortest path bet^{n} A and D,

$2+1+2+3+1 = 9$

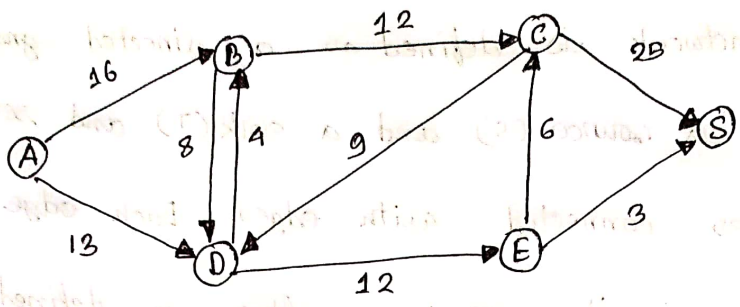
(ii) length of edges in a minimum spanning tree

$3+2+1+2+3+1 = \cancel{10}\ 12$

(8)

(a) A flow network is defined as a directed graph involving a source (S) and a sink (T) and several other nodes connected with edges. Each edge has an individual capacity. Maximum flow is defined as the maximum amount of flow that the network would allow from source to sink.

(b)

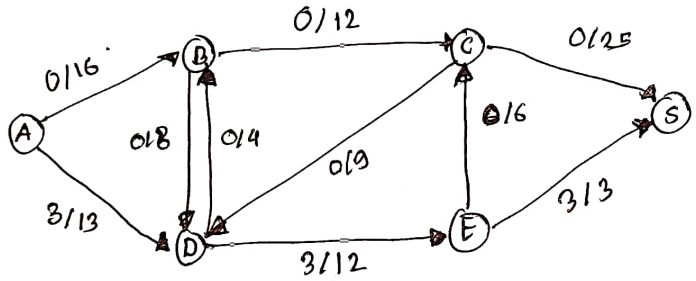


Here,

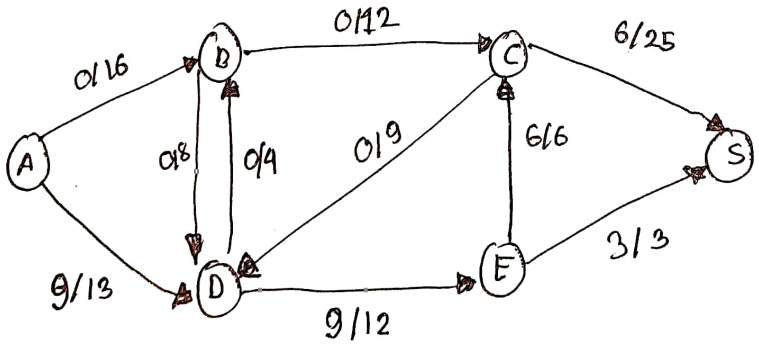
A = Source

S = Sink

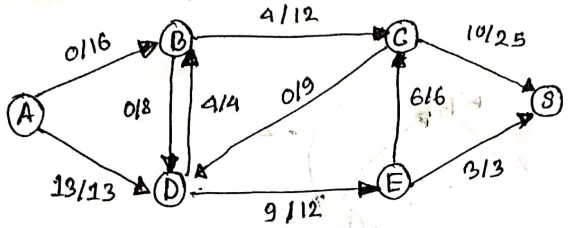
1st we will flow 3 flow in A → D → E → S path



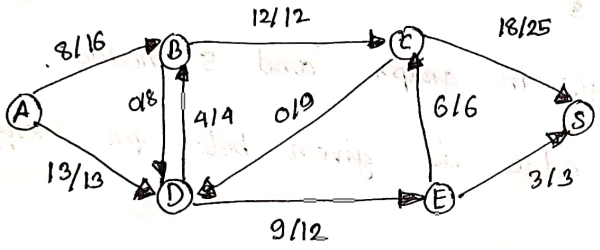
2nd we will flow 6 flow in A → D → E → C → S path



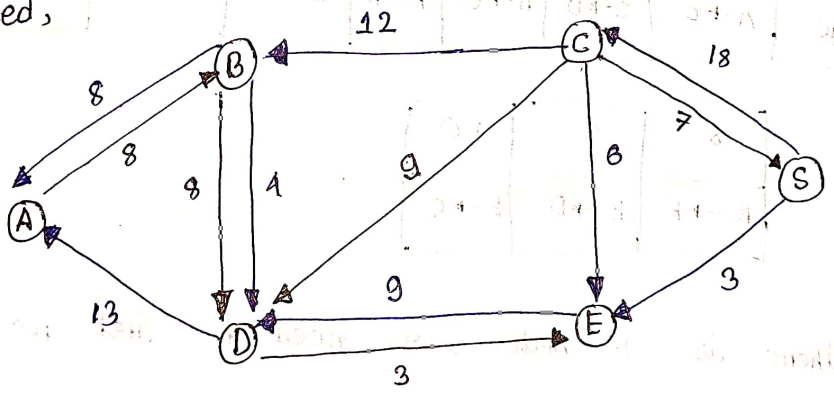
3nd we will flow 4 flow in A→D→B→C→S path.



4th we will flow 8 flow in A→B→C→S path.



Now the residual network G_f with augmenting path p shaded,

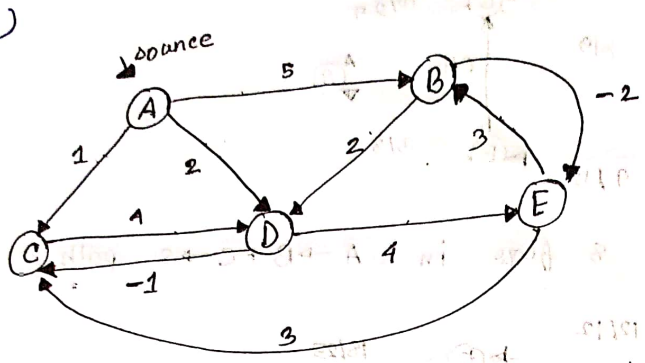


∴ There is no augmenting path left. So max flow is

18 + 3 = 21.

Bellman - Ford → Coneman - (P - 681)

8(c)



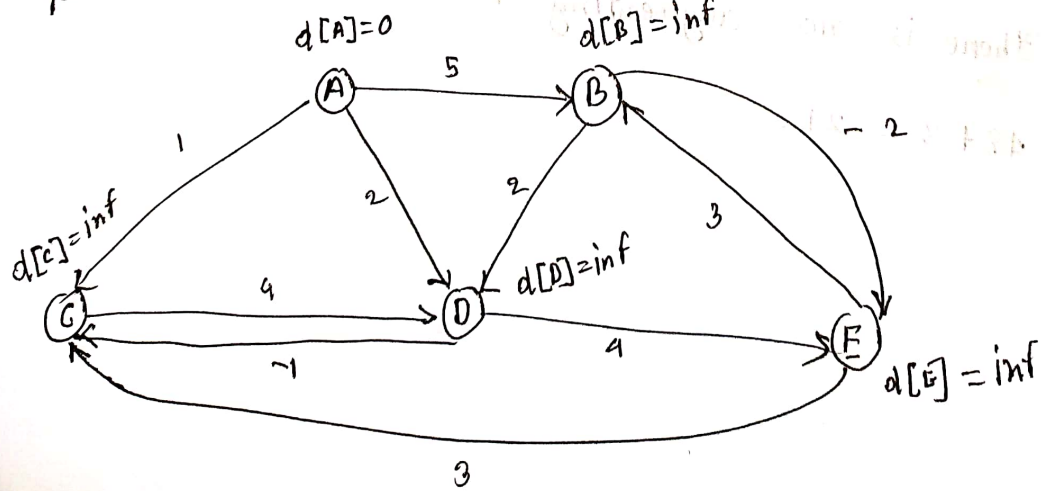
There 10 edges in graph and 5 nodes.

Let, order of edge is given below for every relaxation.

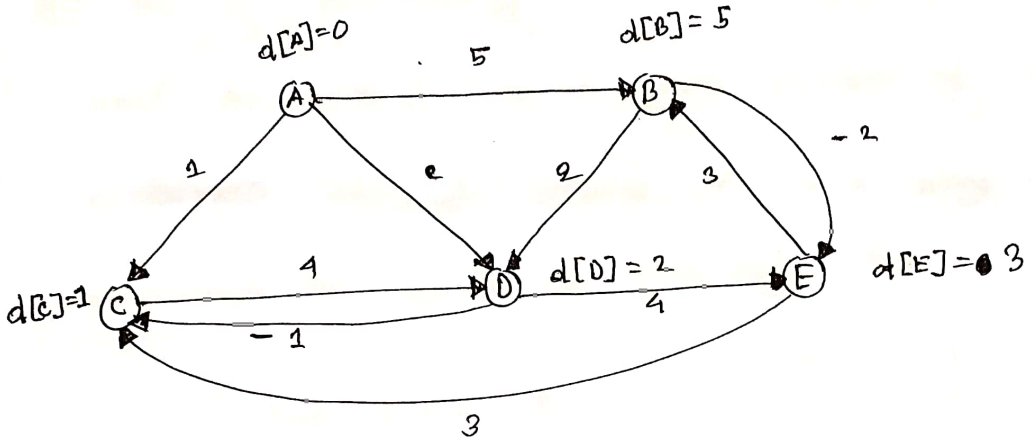
Serial	1	2	3	4	5	6	7
Edge	A → C	A → D	A → B	C → D	D → C	D → E	E → B

8	9	10
B → E	B → D	E → C

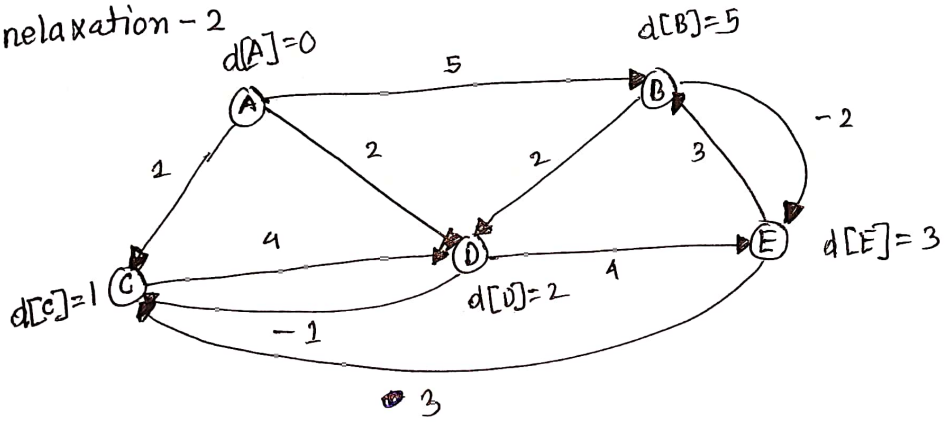
As there is 5 nodes, we need to ~~first~~ relax edges for 4 times.



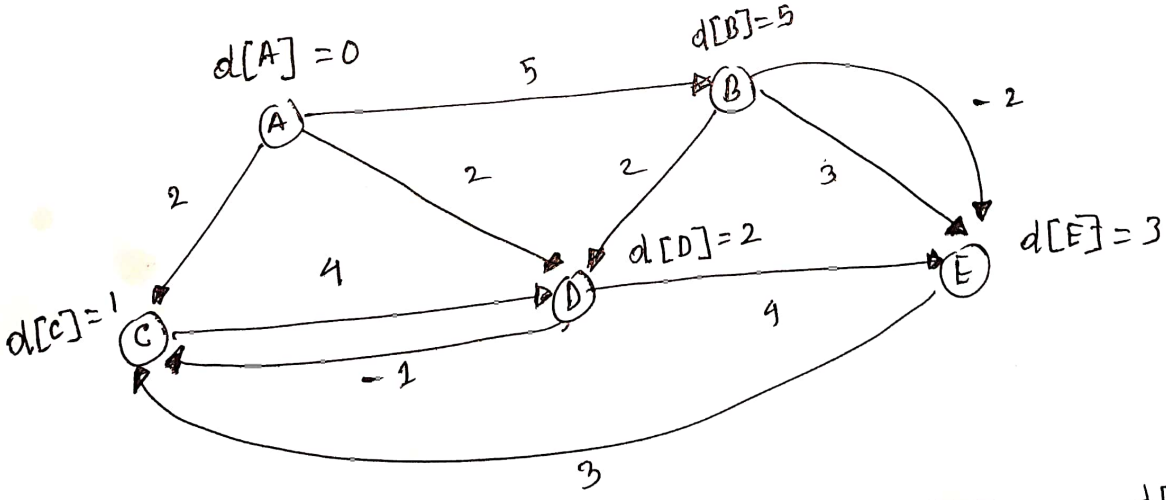
relaxation – 1



relaxation – 2



relaxation – 3 & 4 (as nothing updated from 1 to 2]



Sorthest path, d[A]=0, d[B]=5, d[C]=1, d[D]=2, d[E]=3

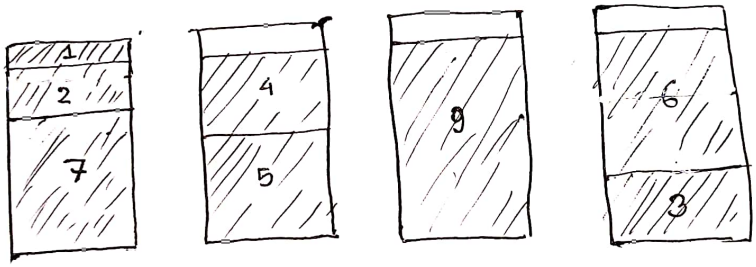
(7)

(c) In the Bin Packing problem, objects of different volumes must be packed into a finite number of bins or containers each of volume V in a way that minimizes the number of bins used.

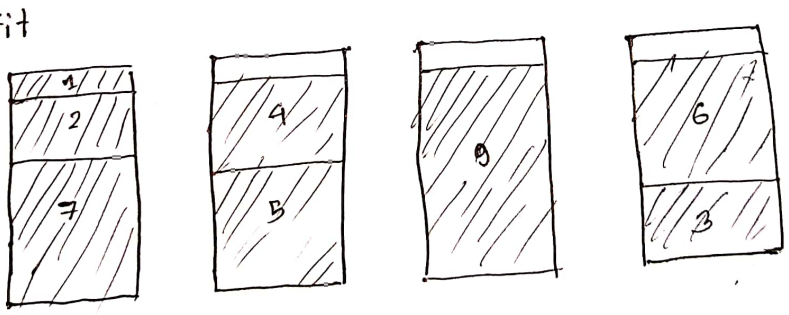
items	7	2	5	1	9	4	3	6

Here, bin size, B=10

(i) First Fit



(ii) Best Fit



(iii) Next Fit