Karnatak Law Society's

VishwanathRao Deshpande Institute of Technology, Haliyal

Department of Computer Science & Engg

Sub: Design and Analysis of Algorithms [18CS42]

Model Question Paper 1 2019

Staff Name: Prof. Yasmeen Shaikh

MODULE - 1

1a Define algorithm. Discuss the criteria of an algorithm with an example. [6M]

→ An algorithm is a finite set of instructions that, if followed, accomplishes a particular task. All algorithms must satisfy the following criteria

i) Inpnt - zero or more quantities are externally supplied

ii) Outpnt - Atleast one quantity is produced.

iii) definiteness - Each instruction is clear and unambiguous.

iv) Finiteness - If we trace out the instructions of an algorithm, then for all cases the algorithm terminates after a finite number of steps.

v) Effectiveness - Every instruction must be very basic so that it can be carried using pencil & paper, It is not enough that operation is definite but it also must be feasible.

1b What are the various basic asymptotic efficiency classes? Explain Big O, Big Omega & Big theta asymptotic notations [8M]

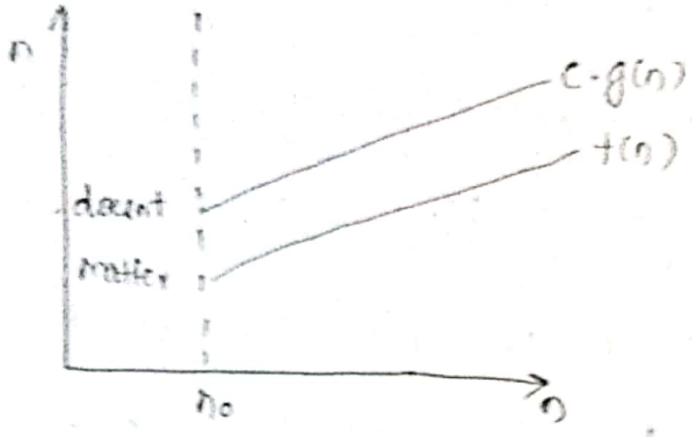
→ The various basic asymptotic efficiency classes are 1, $\log n$, n, $n\log n$, n^2, n^3, 2^n & $n!$. which are constant, logarithmic, linear, n log n, quadratic cubic, exponential & factorial respectively.

Big oh Notation

A function $t(n)$ is said to be in $O(g(n))$ denoted by $t(n) \in O(g(n))$, if $t(n)$ is bounded above by some constant multiple of $g(n)$ for all large n i.e. if there exist some positive constant c & some non negative integer n_0 such that

$$t(n) \leq c \cdot g(n) \text{ for all } n \geq n_0$$

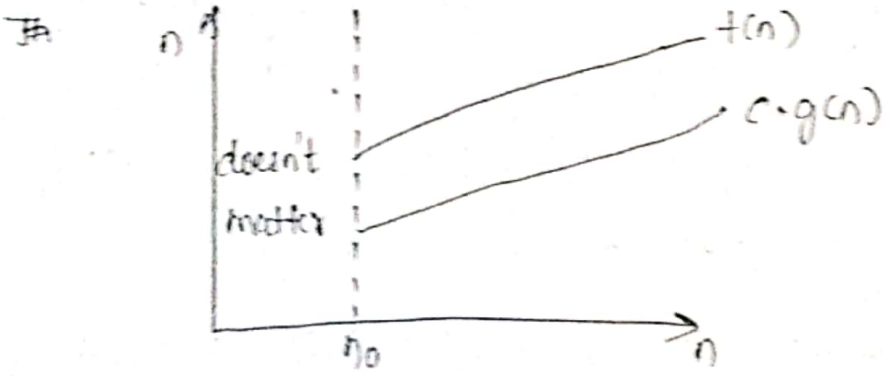
Graph given below represents Big oh notation



ii) Big Omega Notation (Ω)

A function $t(n)$ is said to be $\Omega(g(n))$ denoted by $t(n) \in \Omega(g(n))$, if $t(n)$ is bounded below by some positive constant multiple of $g(n)$ for all large n, i.e. if there exist some positive constant c and some non negative integer n_0 such that

$$t(n) \geq c \cdot g(n) \quad \text{for all } n \geq n_0$$



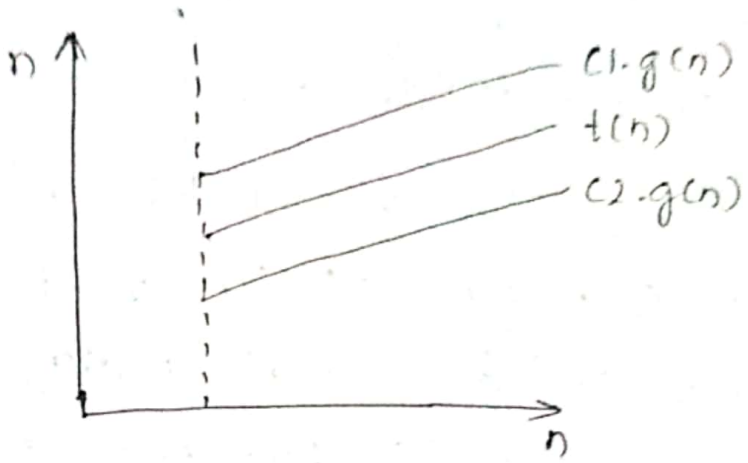
iii) Big Theta Notation

A function $t(n)$ is said to be in $\Theta(g(n))$ denoted by $t(n) \in \Theta(g(n))$, if $t(n)$ is bounded above & below by some positive constant multiple of $g(n)$ for all large n. i.e

if there exist positive constant c_1 & c_2 and nonnegative integers n_0 such that

$$c_2 g(n) \leq t(n) \leq c_1 \cdot g(n) \text{ for all } n \geq n_0$$

the above expression can be shown as



1c. Discuss about Important problem types and fundamental data structures. [6M]

→ Important problem types

1. sorting – These problems arrange items in ascending or descending order. Two important properties of these algorithm is that they satisfy stability & are in place

2. searching – deals with searching a given value in a given set of elements.

3. String processing – performs operations on strings

4. Graph problems – Graphs are used for modeling a variety of real life applications including transportation, communication

network, project scheduling & games.

5. Combinatorial problems - ask to find a combinatorial object such as permutation, combination or subset - that satisfy certain constraints & has some desired property.

6. Geometric problems - deal with geometric objects such as points, lines & polygons

Fundamental data structures

1. Array is a sequence of n items of the same data type that are stored contiguously in computer memory & made accessible by specifying value of array's index

2. String - sequence of characters.

3. Linked list - is sequence of zero or more elements called nodes each containing two kinds of information: data field & link field. Types of lists are singly linked list, DLL, CLL.

4. Stack is a LIFO data structure where elements are inserted and deleted from end called top.

5. Queue is a FIFO data structure where elements are inserted from rear end and

deleted from front end.

6) Graph - $G = (V, E)$ is defined by a pair of vertices & edges. A graph can be directed or undirected.

2a. Outline an algorithm to find maximum of n elements and obtain its time complexity. [7M]

→ Algorithm

```
MaxElement (A[0...n-1])
// Determine the value of the largest element
   in a given array */
// Input : array A[0...n-1]
// Output : value of largest element
    maxval ← A[0]
    for i ← 1 to n-1 do
        if A[i] > maxval
            maxval ← A[i]
    return maxval.
```

Time efficiency can be calculated as follows:

Step 1 : In this problem, size of input is n

Step 2 : Basic operation is

if A[i] > maxval.

Steps: Each time the control enters the loop the basic operation is executed once

for i ← 1 to n-1 do

if A[i] > maxval

$$f(n) = \sum_{i=1}^{n-1} 1$$

$$= n-1-1+1$$

$$f(n) = n-1$$

Step 4: The total number of times the basic operation is executed remains same in best case & worst case.

So $f(n)$ is expressed using Θ notation

$$\frac{1}{3} * n \leq n-1 \leq 1 * n$$

$c_1 = 1/3$, $c_2 = 1$, $g(n) = n$ for $n \geq 1$

$\therefore f(n) \in \Theta(g(n))$.

2b Discuss an algorithm to search an element in an array using sequential search. Discuss the best case, worst case & average case efficiency of this algorithm. [7M]

→ Algorithm

SeqSearch (A[0...n-1], key)

// search element in A[0...n-1]

// Input: A & key

// Output : returns the index of the element on successful search & return -1 on unsuccessful search

```
i ← 0
A[n] ← key
while (A[i] ≠ key) do
    i ← i+1
if (i < n)
    return i
else
    return -1
```

Analysis

1. The input size is n
2. The basic operation is A[i] ≠ key
3. The running time of the algorithm is not only dependent upon input size but it is dependent on worst case, best case & average case time complexities

If key element to be searched is present at nth location then clearly the algorithm runs for longest time.

$C_{worst} = n$.

If key element is present at first position. Then algorithms runs for very short time

$\therefore C_{best} = 1$

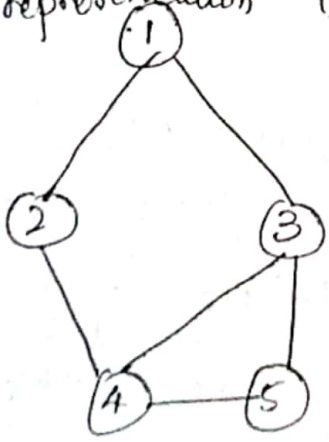
If key element to be searched is present at middle position in the array, the number of comparisons would be $\leq n/2$

$\therefore C_{Avg} = n/2$.

2c. Discuss adjacency list and adjacency matrix representation of a graph. [6M]

→ Consider a graph G of n vertices and the Matrix M. If there is an edge present between vertex v_i & v_j then $M[i][j] = 1$ else $M[i][j] = 0$

This representation is adjacency matrix.



⇒

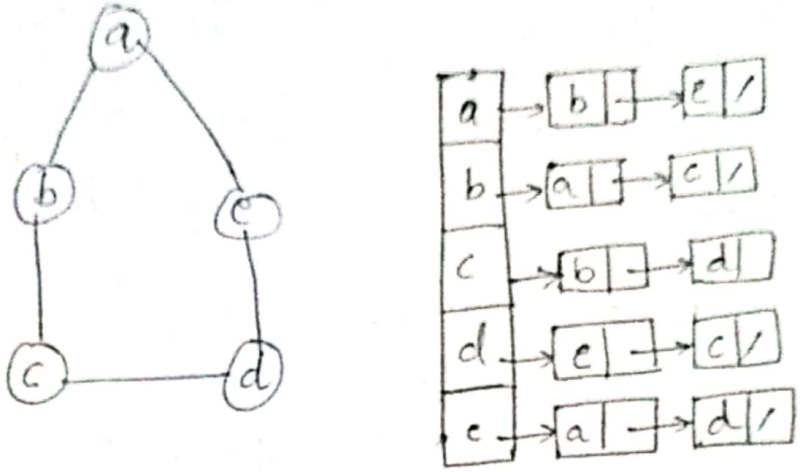
	1	2	3	4	5
1	0	1	1	0	0
2	1	0	0	1	0
3	1	0	0	1	1
4	0	1	1	0	1
5	0	0	1	1	0

Adjacency linked list.

Let G = V, E be a graph.

An array of lists is used. The size of the array

is equal to the number of vertices. A[i] represent the ith vertex. The edges adjacent to i are represented using link.



MODULE - 2

3a. Explain the concept of divide and conquer. Write the recursive algorithm to perform binary search on list of elements. [7M]

→ In divide and conquer method, a given problem is

i) divided into smaller subproblems

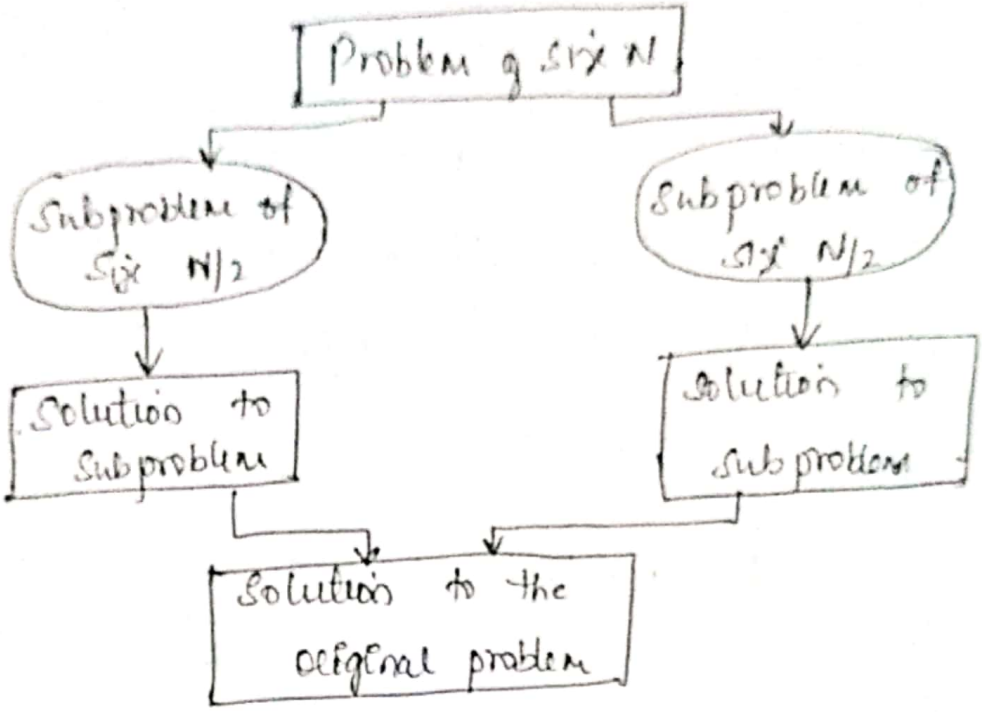
ii) These subproblems are solved independently.

iii) Combining all the solutions of subproblems into a solution of the whole problem.

divide and conquer consists of two parts

i) divide - divide the problem into a number of subproblems. These subproblems are solved recursively.

iii) Conquer - The solution to the original problem is then formed from the solutions to the subproblems



Algorithm

```
BinarySearch (A, key, low, high)
  // purpose : search for an item in the list
  // Input : A, key,
  // output : If successful, position is returned
              else -1 is returned for unsuccessful
              search.
  m ← (low + high)/2
  if (key = A[m])
      return m
  else if (key < A[M])
```

BinarySearch (A, key, low, m-1)

else

BinarySearch (A, key, m+1, high)

end

3b. Develop a recursive algorithm to find the maximum and minimum element from the list. Illustrate with an example. [Jan]

→ Algorithm

void MaxMin(int i, int j, Type &max, Type &min)

// a[1:n] is a global array

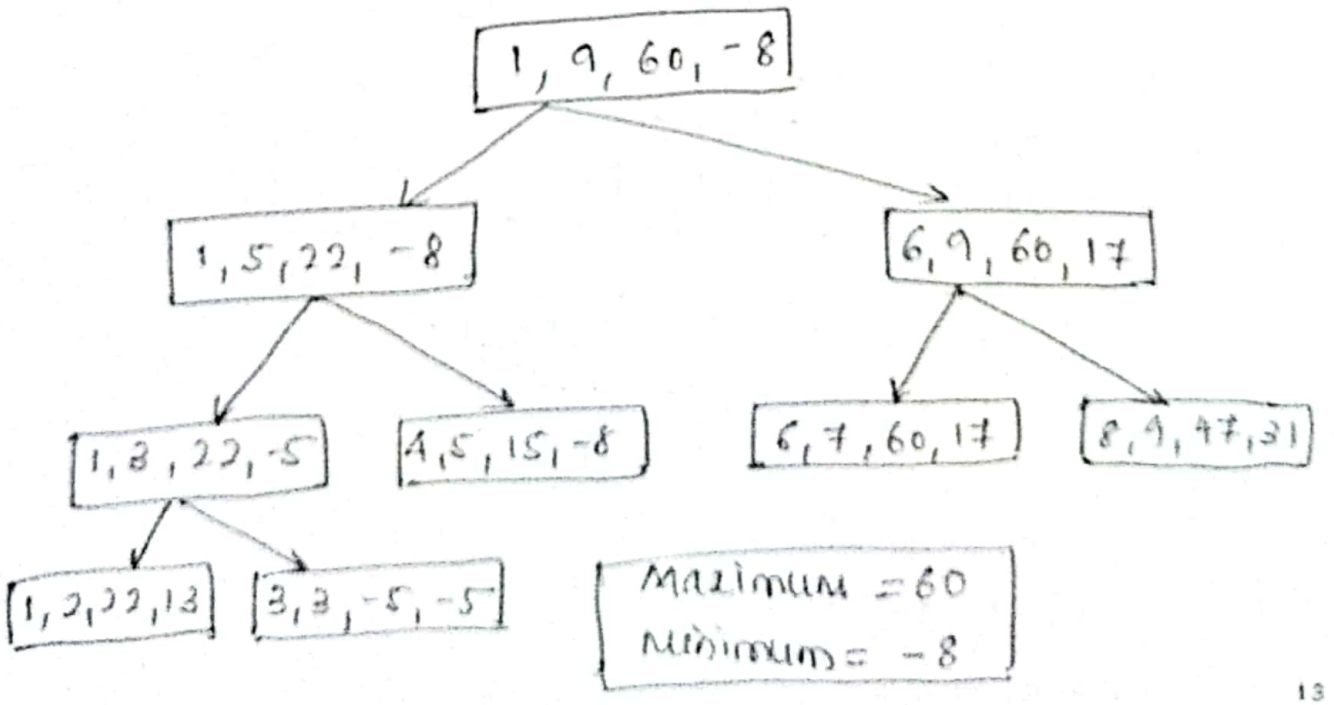
// parameters i & j are integers 1 <= i <= j <= n

// The effect is to set max and min to the largest & smallest value in a[i:j] respectively

```
if (i == j)
    max = min = a[i]
else if (i == j-1)
    {
     if (a[i] < a[j])
        {
         max = a[j];
         min = a[i];
        }
```

```
        else
          {
            max = a[i];
            min = a[j];
          }
      }
    else
      {
        int mid = (i+j)/2;
        MaxMin (i, mid, max, min);
        MaxMin (mid+1, j, max1, min1);
        if (max < max1)
            max = max1;
        if (min > min1)
            min = min1;
      }
}
```

Consider elements 22, 13, -5, -8, 15, 60, 17, 31, 47



3C Apply Quicksort on the following set of elements. [6M]

$60_0, 70_1, 75_2, 80_3, 85_4, 60_5, 55_6, 50_7, 45_8$

→ Pivot = 60 i = 0 j = 9

① while 0 <= 9 T

do i = 1 60 >= 70 F

do j = 8 60 < 45 F

if i < j T exchange 70 & 45

$60_0, 45_1, 75_2, 80_3, 85_4, 60_5, 55_6, 50_7, 70_8$

② while (1 <= 8)

do i = 2 60 >= 75 T

do j = 7 60 < 50 F

if i < j T exchange 75 & 50

$60_0, 45_1, 50_2, 80_3, 85_4, 60_5, 55_6, 75_7, 70_8$

③ while (2 <= 7)

do i = 3 60 >= 80 F

do j = 6 60 < 55 F

if i < j T exchange 80 & 55

$60_0, 45_1, 50_2, 55_3, 85_4, 60_5, 80_6, 75_7, 70_8$

④ while 3 <= 6

do i = 4 60 >= 85 F

do j = 5 60 < 60 F

if i < j T exchange 85 & 60

$60_0, 45_1, 50_2, 55_3, 60_4, 85_5, 80_6, 75_7, 70_8$.

⑤ while 4 <= 5

do i = 5 60 >= 85 F

do j = 4 60 < 60 F

if i < j F

end while

Exchange a[4] & a[0], 1st partition obtained is

60 45, 50, 55, 60, 85, 80, 75, 70 .

Consider 1st partition elements

60, 45, 50, 55
0 1 2 3

pivot=60 i=0 j=4

while 0<=4

do i=1 60>=45 T

do i=2 60>=50 T

do i=3 60>=55 T

do i=4 Not possible

do j=3 60<55 F

if i<j F

exchange a[0] & a[3]

55, 45, 50, [60]
0 1 2

pivot=55 low=0 high=3

while 0<=3

do i=1 55>=45 T

do i=2 55>=50 T

do i=3 NP

do j=2 55<50 F

if i<j F

exchange a[0] & a[2]

50, 45, [55]

0 1
50, 45

pivot=50 i=0 low=2

whik (0<=2)

do i=1 50>=45 T

do i=2 NP

do j=1 50<45 F

if i<j F

exchange a[0] & a[1]

we get 45, [50]

Now we apply partition on

50 & 45

Consider 2nd partition elements

85, 80, 75, 70
0 1 2 3

Pivot = 85 low = 0 High = 3

① while 0 <= 4

do i = 1 85 >= 80 T

do i = 2 85 >= 75 T

do i = 3 85 >= 70 T

do i = 4 NP

do j = 3 85 < 70 F

if i < j F

exchange a[0] & a[3]

70, 80, 75, [85]

② Pivot = 70, low = 0, high = 3

while 0 <= 3

do i = 1, 70 >= 80 F

do j = 2 70 < 75 T

do j = 1 70 < 80 T

do j = 0 70 < 70 F

if i < j F

exchange a[0] & a[0]

we get [70], 80, 75

Consider 80, 75
0 1

Pivot = 80, low = 0, high = 2

while 0 <= 2

do i = 1. 80 >= 75 T

do i = 2 NP

do j = 1 80 < 75 F

if i < j F

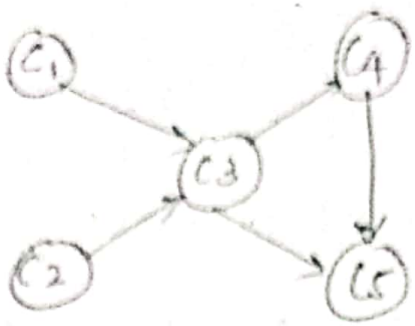
exchange a[0] & a[1]

75, [80]

Finally arranging all partitions we get sorted elements. They are

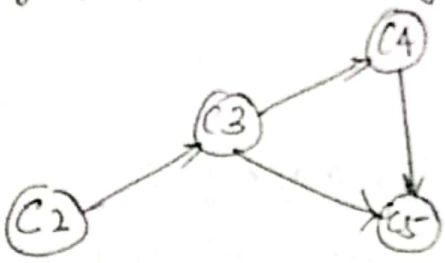
45, 50, 55, 60, 60, 70, 75, 80, 85

4a Apply Source removal method to obtain Topological Sort for the given graph.

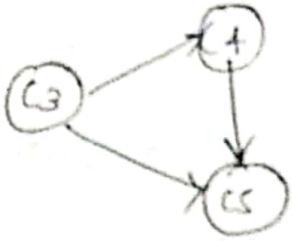


→ Solution:

Step 1 c_1 is independent so delete c_1 from above graph, the resulting graph is



Step 2: c_3 is independent so delete c_2 from above graph, the resulting graph is



Step 3: c_3 is independent so delete c_3, from above graph, the resulting graph is

Step1 : C4 is independent, so delete C4 from above graph

(C5)

Step5 : C5 is the only node left delete it. We are left with empty graph

Topological sort is C1 → C2 → C3 → C4 → C5

4b Write an algorithm to sort N numbers by applying Merge sort. [7M]

→ Algorithm

```
Mergesort (A, low, high)
// purpose : sort the elements of array between
             lower bound & upper bound
// Input : A is unsorted vector with low & high
           as lower bound & upper bound
// output : A is a sorted vector
  if (low < high)
     mid ← (low + high)/2
     mergesort (a, low, mid)
     mergesort (a, mid+1, high)
     simplemerge (a, low, mid, high)
  end if
```

Algorithm

```
Simple merge ( a , low , mid, high)
// purpose : merge two sorted arrays
// Input : A is sorted from index position low
           to mid
        A is sorted from mid+1 to high
// output : A is sorted from low to high
  i ← low , j ← mid +1 , k ← low
  while (i <= mid & j <= high)
    if (A[i] < A[j]) then
        C[K] ← A[i]
        i ← i+1
        k ← k+1
    else
        C[k] ← A[j]
        j ← j+1
        k ← k+1
    end if
  end while
  while (i <= mid)
        C[K] ← A[i]
        k ← k+1,
        i ← i+1
  end while
```

```
while (j <= high)
    c[k] ← a[j]
    k ← k+1
    j ← j+1
end while
for i = low to high
    a[i] ← c[i]
end for.
```

4c Apply Strassens matrix multiplication method to multiply the given two matrices. Discuss how this method is better than general matrix multiplicaton method.

$$\begin{bmatrix} 4 & 3 \\ 2 & 1 \end{bmatrix} \times \begin{bmatrix} 2 & 5 \\ 1 & 6 \end{bmatrix} \qquad [7M]$$

→ Strassens matrix multiplicaton can be accomplished by using formula given below

$$\begin{bmatrix} C_1 & C_2 \\ C_3 & C_4 \end{bmatrix} = \begin{bmatrix} A1 & A2 \\ A3 & A4 \end{bmatrix} * \begin{bmatrix} B1 & B2 \\ B3 & B4 \end{bmatrix}$$

$$= \begin{bmatrix} m_1 + m_4 - m_5 + m_7 & m_3 + m_5 \\ m_2 + m_4 & m_1 + m_3 - m_2 + m_6 \end{bmatrix}$$

where

$m_1 = (A1 + A4) * (B1 + B4)$

$m_2 = (A3 + A4) * B1$

$m3 = A1 * (B2 - B4)$

$m4 = A4 * (B3 - B1)$

$m5 = (A1 + A2) * B4$

$m6 = (A3 - A1) * (B1 + B2)$

$m7 = (A2 - A4) * (B3 + B4)$

Now substituting value of $A1=4$, $A2=3$, $A3=2$, $A4=1$, $B1=2$, $B2=5$, $B3=1$, $B4=6$ we get

$m1 = (A1 + A4) * (B1 + B4) = (4+1) * (2+6) = 40$

$m2 = (A3 + A4) * B1 = (2+1) * 2 = 6$

$m3 = A1 * (B2 - B4) = 4 * (5-6) = -4$

$m4 = A4 * (B3 - B1) = 1 * (1-2) = -1$

$m5 = (A1 + A2) * B4 = (4+3) * 6 = 42$

$m6 = (A3 - A1) * (B1 + B2) = (2-4) * (2+5) = -14$

$m7 = (A2 - A4) * (B3 + B4) = (3-1) * (1+6) = 14$

Now

$$\begin{bmatrix} C1 & C2 \\ C3 & C4 \end{bmatrix} = \begin{bmatrix} m1 + m4 - m5 + m7 & m3 + m5 \\ m2 + m4 & m1 + m3 - m2 + m6 \end{bmatrix}$$

$$= \begin{bmatrix} 40 + (-1) - 42 + 14 & -4 + 42 \\ 6 + (-1) & 40 + (-4) - 6 + (-14) \end{bmatrix}$$

$$\begin{bmatrix} C1 & C2 \\ C3 & C4 \end{bmatrix} = \begin{bmatrix} 11 & 38 \\ 5 & 16 \end{bmatrix}$$

We know that the time complexity of an algorithm to multiply 2 matrices using straightforward method is $O(n^3)$.

- But, Strassen matrix multiplications uses less time to ~~now~~ multiply 2 matrices. The time complexity is $\theta(n^{2.807})$.

∴ Strassen algorithm is more efficient than the conventional matrix multiplication for sufficiently large value of n.

MODULE - III

5a Find the optimal solution to the knapsack instance n = 7, m = 15 using greedy method.

Object	1	2	3	4	5	6	7
Weight	2	3	5	7	1	4	1
Profit	10	5	15	7	6	18	3

[07M]

Soln:

Steps: compute the profit / weight ratio for each object

object	weight	profit	Ratio
1	2	10	5
2	3	5	1.66
3	5	15	3
4	7	7	1
5	1	6	6
6	4	18	4.5
7	1	3	3

Steps: Sort all items in the descending order of their profit / weight ratio.

object 5	object 1	object 6	object 3	object 7	object 2	object 4
6	5	4.5	3	3	1.66	1

Steps: start filling the knapsack by putting objects one by one.

knapsack weight	objects in knapsack	cost
17	0	0
16	object 5	6
14	object 5, object 1	16
10	objects, object 1, object 6	34
5	object 5, object 1, object 6, object 3,	49
4	object 5, object 1, object 6, object 3, object 7	52
1	object 5, object 1, object 6, object 3, object 7, object 2	57

Now the knapsack capacity to be filled is 01, but object 4 has weight of 07.

Since in fractional knapsack problem, even the fraction of any item can be taken

So, knapsack will contain following items

⟨object 5, object 1, object 6, object 3, object 7, object 2, (1/7) object 4.⟩

Total cost of knapsack = $57 + \frac{1}{7} \times 7$

$= 57 + 1$

$= 58$

5b What is Job sequencing with deadlines problem? For the given data, find the optimal job sequence & maximum profit using Greedy approach [6M]

job	J1	J2	J3	J4	J5
profit	60	100	20	40	20
deadline	2	2	3	1	1

→ Job sequencing with deadlines

- We are given n jobs. Associated with each job i, deadline $d_i > 0$ & profit $P_i > 0$. is given
- For any job i, the profit P_i is earned if and only if the job the completed by its deadline.
- only one machine is available for processing jobs.
- An optimal solution is the feasible solution with maximum profit

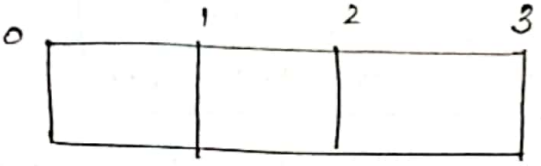
Job	J_1	J_2	J_3	J_4	J_5
Profit	60	100	20	40	20
deadline	2	2	3	1	1

Step 1: Sort all given jobs in decreasing order of their profits

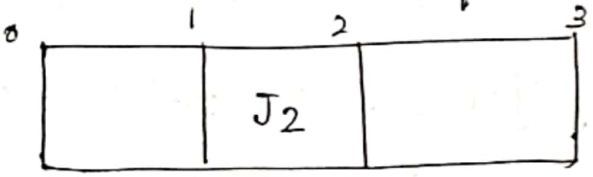
Job	J_2	J_1	J_4	J_3	J_5
profit	100	60	40	20	20
deadline	2	2	1	3	1

Step 2 : Value of maximum deadline is 3, so draw a Gantt chart with maximum time on Gantt chart = 5 units as shown below

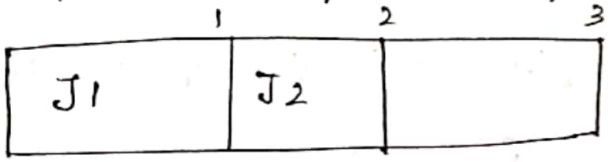
Step.



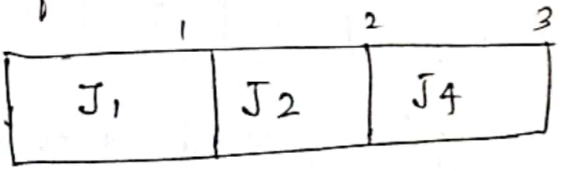
Step 3 : Now place job J2 before deadline 2.



Step 4 : Now place job J at deadline 2, but that slot is already filled. So check for any empty slot before. ∴ place Job J1 at 1



Step 4 : Now place job J4 at deadline 1, but that cell is already filled to place job J4 before 3.



Step 5 : Remaining job cannot be scheduled.

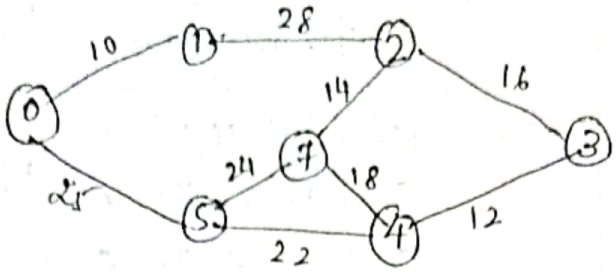
Maximum profit = Profit of J1 + profit of J2 + Profit of J4

= 100 + 60 + 40

= 200 units

Optimal schedule is J1, J2, J4

5c. Apply Prims algorithm to obtain the minimum cost spanning tree for the given weighted graph.



7M.

Solution:

Cost adjacency Matrix

	0	1	2	3	4	5	7
0	0	10	∞	∞	∞	25	∞
1	10	0	28	∞	∞	∞	∞
2	∞	28	0	16	∞	∞	14
3	∞	∞	16	0	12	∞	∞
4	∞	∞	∞	12	0	22	18
5	25	∞	∞	∞	22	0	24
7	∞	∞	14	∞	18	24	0

In the above graph least cost edge is (0, 1) with cost 10. Hence (0, 1) is selected in MST

& source node is 0

Distance matrix D is

0	1	2	3	4	5	7
0	10	∞	∞	∞	25	∞

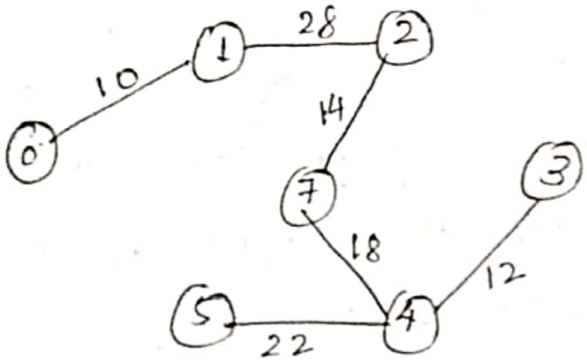
Path matrix P is

0	1	2	3	4	5	7
0	0	0	0	0	0	0

S	V = V − S	d[v] = min(d[v], cost[u,v])	P[v] = u	u, d[u]	Output u, P(u)
0	1,2,3,4,5,7	–	–	1, 10	1, 0
0,1	2,3,4,5,7	d[2] = Min(∞, 28) = 28 d[3] = min(∞, ∞) = ∞ d[4] = min(∞, ∞) = ∞ d[5] = min(25, ∞) = 25 d[7] = min(∞, ∞) = ∞	P[5] = 1 P[2] = 1	5, 25 2, 28	5, 1

0,1,2	3,4,5,7	d[3]=min(∞,16)=16 d[4]=min(∞,∞)=∞ d[5]=min(25,∞)=25 d[7]=min(∞,14)=14	P[7]=2	7, 14	7, 2
0,1,2,7	3,4,5	d[3]=min(16,∞)=16 d[4]=min(∞,18)=18 d[5]=min(25,24)=24	P[4]=7	4, 18	4, 7
0,1,2,7,4	3,5	d[3]=min(16,12)=12 d[5]=min(24,22)=22	P[3]=4 P[5]=4	3, 22	3, 4
0,1,2,7,4,3	5	d[5]=min(22,∞)=22	P[5]=3	-	-

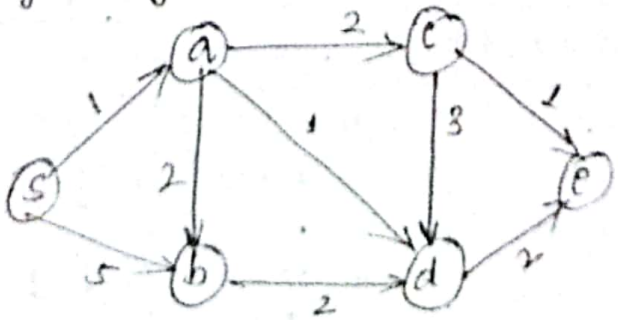
The minimum spanning tree obtained is



Cost = 10+28+14+18+22+12

= 104.

6a. Design Dikstras algorithm and apply the same to find single source shortest path for the given graph by considering 's' as the source vertex [8M]



→ Algorithm

```
Dijkstra(n, w, source, dest, d, p)
// purpose : To compute shortest distance & shortest
             path from source to destination
// Input : n, w, source, dest
// Output : d, p, s
  s[source] = 1
  for i ← 1 to n-1 do
      find u & d[u] such that d[u] is
      minimum & u ∈ V-S
      add u to S
      if (u = dest) break
      for every v ∈ V-S do
          if (d[u] + w[u,v] < d[v])
              d[v] = d[u] + w[u,v]
              p[v] = u
          end if
      end for
  end for
end of algorithm
```

For the given graph

source = S

destination = e

Cost adjacency matrix

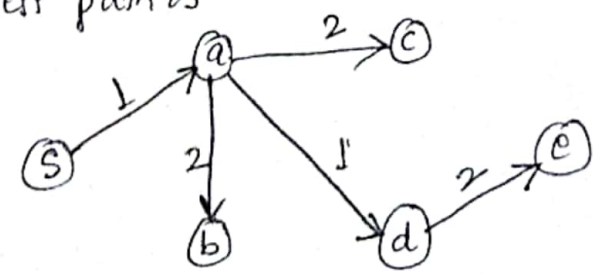
	s	a	b	c	d	e
s	0	1	5	∞	∞	∞
a	1	0	2	2	1	∞
b	∞	∞	0	∞	2	∞
c	∞	∞	∞	0	3	1
d	∞	∞	∞	∞	0	2
e	∞	∞	∞	∞	∞	0

D

0	1	5	∞	∞	∞
s	a	b	c	d	e

S	V=V−S	d[v] = min(d[v], d[u] + w[u][v])	P[v]=u	find u of d[u]
s	a, b, c, d, e	—	—	a, 1
s, a	b, c, d, e	d[b] = min(5, 1+2) = 3 d[c] = min(∞, 1+2) = 3 d[d] = min(∞, 1+1) = 2. d[e] = min(∞, 1+∞) = ∞	P[b] = a P[c] = a P[d] = a	d, 2
s, a, d	b, c, e	d[b] = min(3, 2+∞) = 3 d[c] = min(3, 2+∞) = 3 d[e] = min(∞, 2+2) = 4	P[b] = d P[c] = d P[e] = d	e, 4
s, a, d, e	b, c	d[b] = min(3, 4+∞) = 3 d[c] = min(3, 4+∞) = 3	P[b] = 3 P[c] = 3	b, 3
s, a, d, e, b	c	d[c] = min(3, 3+∞) = 3	P[c] = b	c, 3
s, a, d, e, b, c	—	—	—	—

Shortest path is

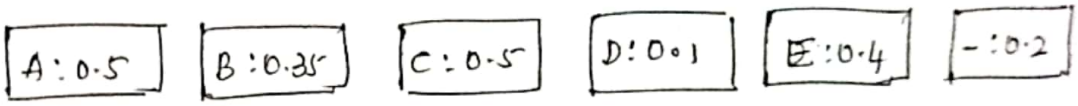


6b Construct the Huffman tree for the following data

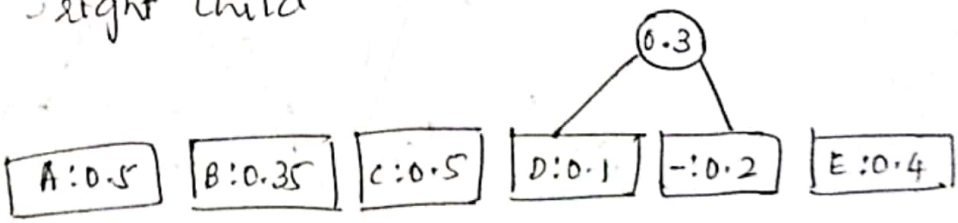
Character	A	B	C	D	E	-
Probability	0.5	0.35	0.5	0.1	0.4	0.2

Encode a) BED b) AB-CD. [5M]

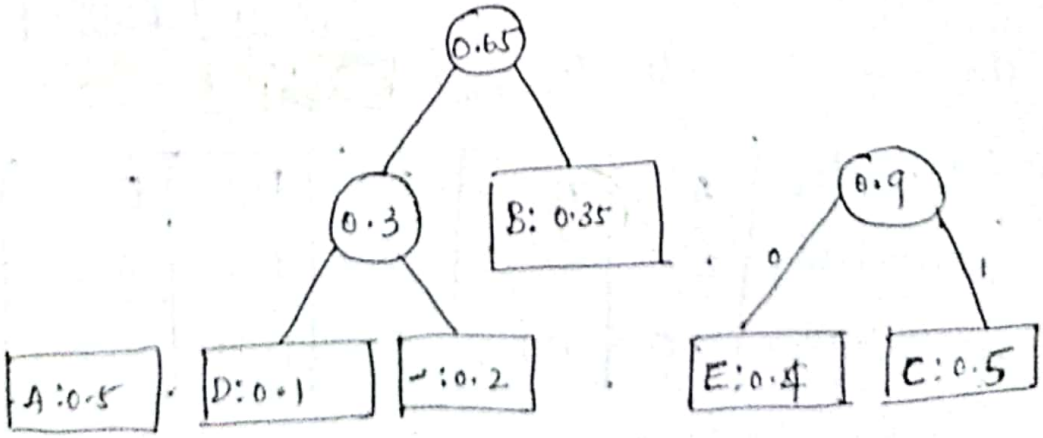
→ Step 1 : Six trees are created for characters A, B, C, D, E, -



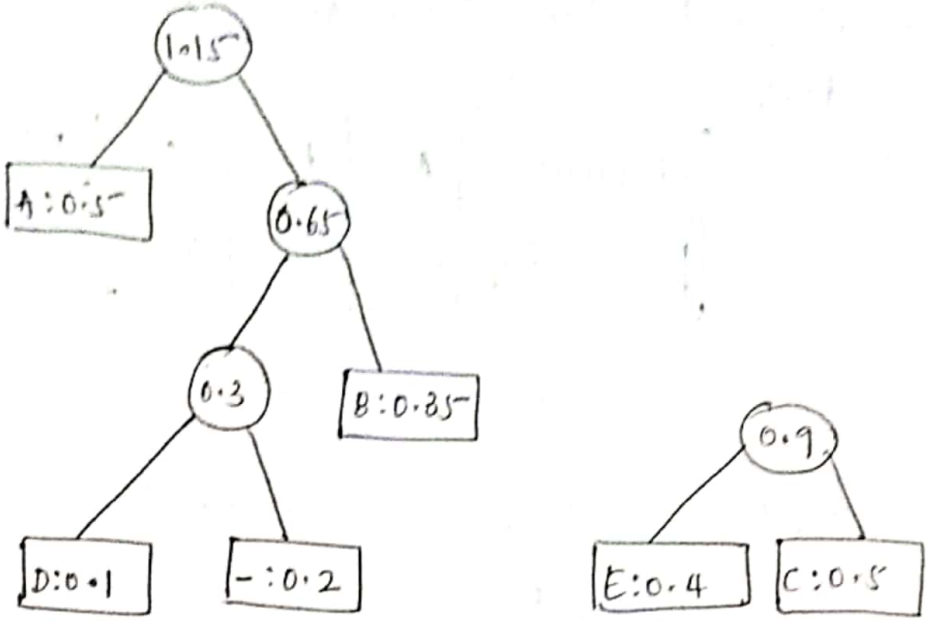
Step 2 : Identify trees with smallest frequencies & make one of them as left child & right child



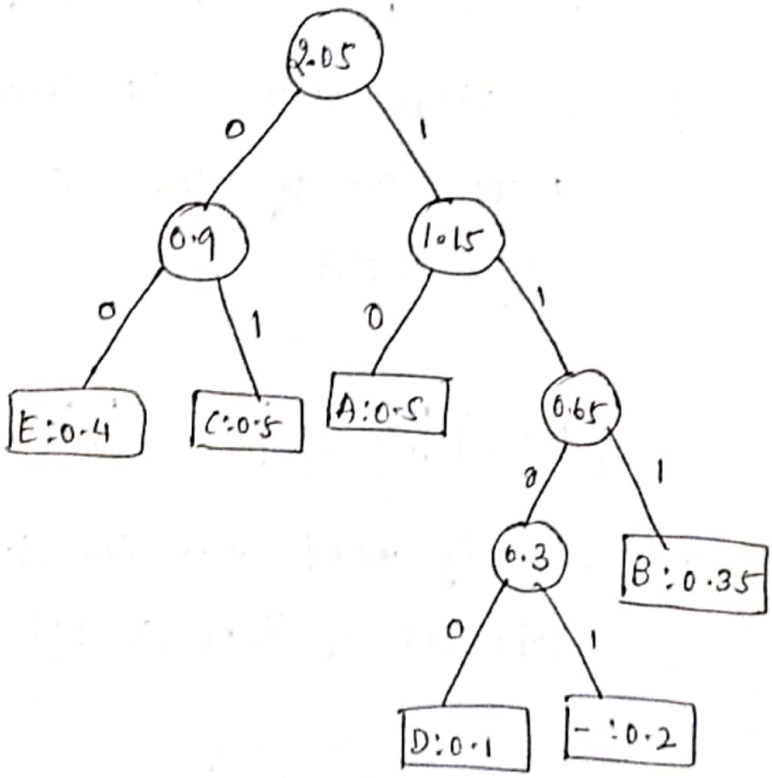
Step 3 : Identify next two smallest frequencies & make one of them as left child & right child



Step 4



Step 5 :



Now we tabulate the code words

Character	A	B	C	D	E	-
Probability	0.5	0.35	0.5	0.1	0.4	0.2
Code word	10	111	01	1100	00	1101

Encode i) BED

⇒ 111 00 1100
B E D

ii) AB_CD

⇒ 10 111 1101 01 1100
A B _ C D

6c. Define Heap. Sort the given elements using heap sort: 2, 9, 7, 6, 5, 8 [8M]

→ Heap is a complete Binary tree or almost complete Binary tree such that the key at each node is greater than or equal to the keys of its children.

There are two types of heap

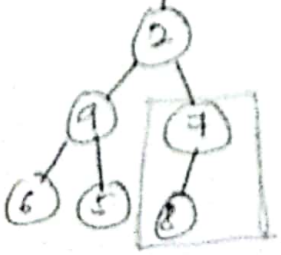
i) Ascending heap
ii) Descending heap

consider given elements

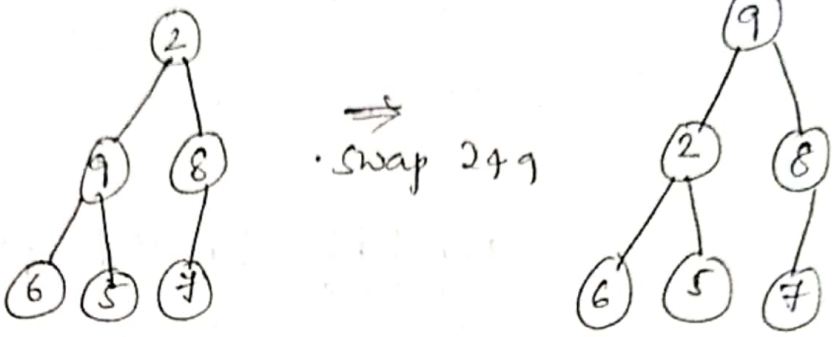
2, 9, 7, 6, 5, 8

To sort the above first create Max. heap, to sort given elements in ascending order.

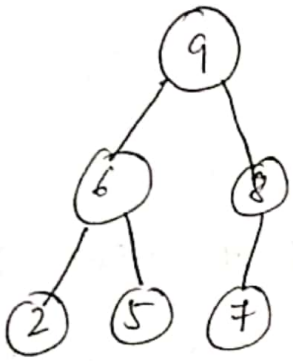
First draw Binary tree for the above elements



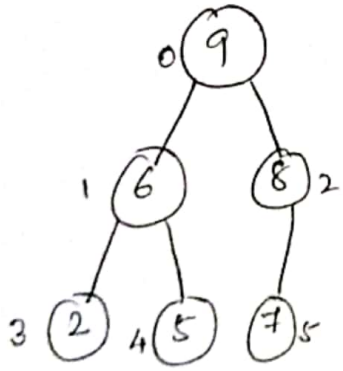
In the above, we need to swap elements 7 & 8 to satisfy max heap condition



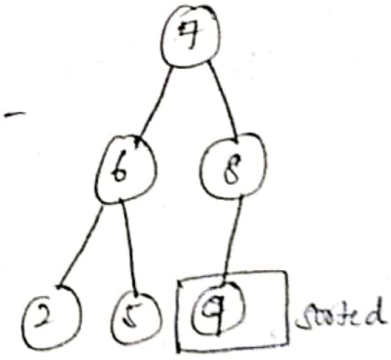
Swap 6 & 2, we get



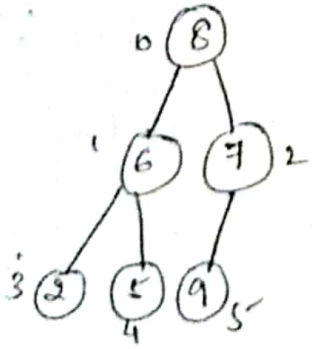
Sorting



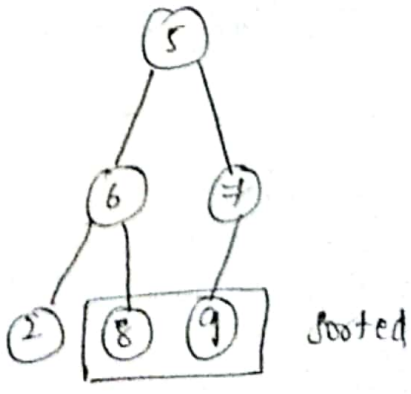
exchange a[0] & a[5]



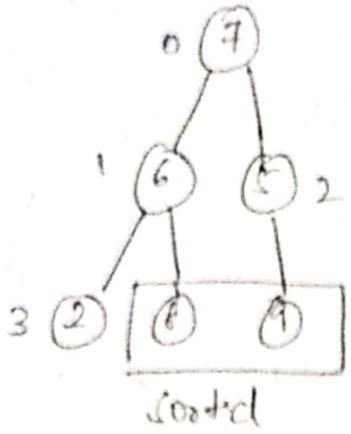
recreate heap



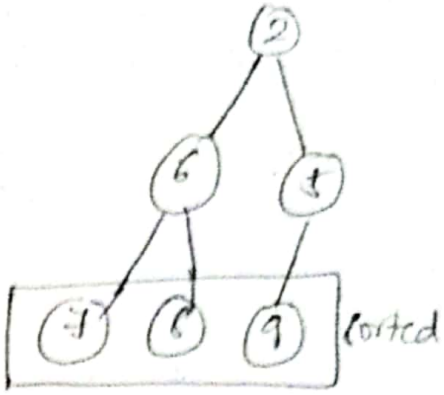
exchange a[0] & a[4]



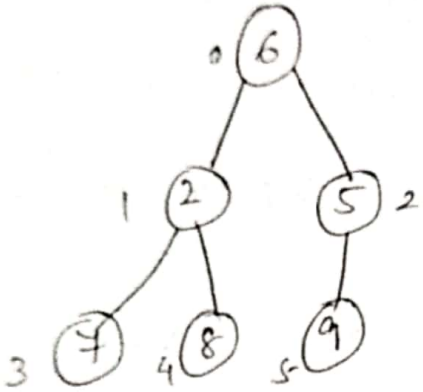
recreate heap



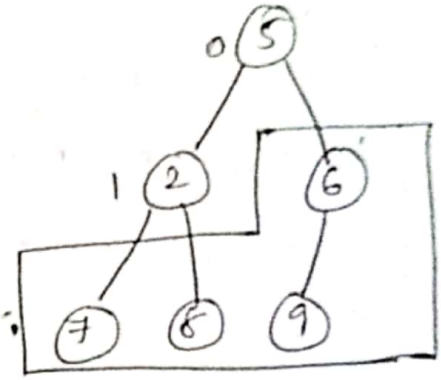
exchange
a[0] ↔ a[3]



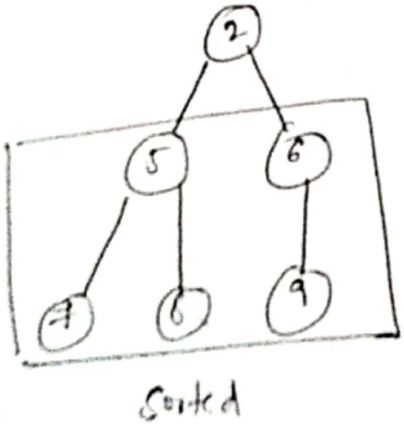
Recreate heap



exchange
a[0] = a[2]



Exchange a[0] ↔ a[0]



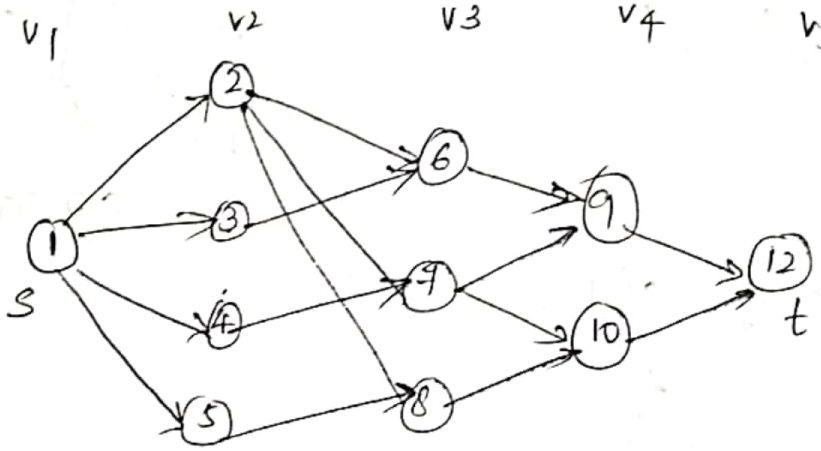
The elements in sorted array are

2, 5, 6, 7, 8, 9.

MODULE - 4

7a. Explain multistage graphs with example. Write multistage graph algorithm using forward approach [6M]

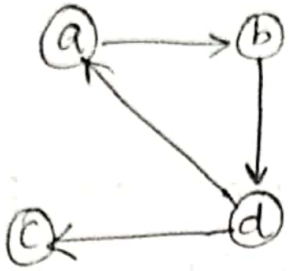
→ Multistage graph $G=(V,E)$ is a directed graph in which the vertices are partitioned into $k>2$ disjoint sets V_i, $1<i<k$. In addition, if (u,v) is an edge in E, then $u \in V_i$ & $v \in V_{i+1}$, for some i, $1<i<k$.



Algorithm

```
Fgraph (G, k, n, p)
{
  cost[n] = 0.0
  for j = n-1 to 1 step -1 do
  {
    let r be a vertex such that (j,r) is an edge
    & c[j,r] + cost[r] is minimum;
    cost[j] = c[j,r] + cost[r]; d[j] = r;
  }
  p[1] = 1; p[k] = n
  for j = 2 to k-1 do p[j] = d[p[j-1]];
}
```

8b Apply Warshall algorithm to compute transitive closure for the graph below. [10M]



Solution:

The adjacency matrix for the given graph is

$$\begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \end{bmatrix}$$

Step 1: To find path from a, consider a^{th} row & a^{th} column

$(a,a)=1$ & $(a,b)=1$ $\therefore (a,b)=1$

resultant matrix is

$$\begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 \end{bmatrix}$$

Step 2: To find path from vertex b, consider b^{th} row & b^{th} column

$a,b=1$ & $b,d=1$ $\therefore a,d=1$

$d,b=1$ & $(b,d)=1$ $\therefore d,d=1$

resultant matrix is $\begin{bmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix}$

Step 3: To find path from vertex c, consider cth row & cth column

In the above matrix, there is no path from c to any other node. So no change in value.

Step 4: To find path from vertex d, consider d^{th} row & d^{th} column

$(a,d)=1 \quad (d,a)=1 \quad \therefore (a,a)=1$

$(a,d)=1 \quad (d,b)=1 \quad \therefore (a,b)=1$

$(a,d)=1 \quad (d,c)=1 \quad \therefore (a,c)=1$

$(a,d)=1 \quad (d,d)=1 \quad \therefore (a,d)=1$

$(b,d)=1 \quad (d,a)=1 \quad \therefore (b,a)=1$

$(b,d)=1 \quad (d,b)=1 \quad \therefore (b,b)=1$

$(b,d)=1 \quad (d,c)=1 \quad \therefore (b,c)=1$

$(b,d)=1 \quad (d,d)=1 \quad \therefore (b,d)=1$

The resultant matrix is

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

7(c) construct an optimal binary search tree for the following four-key set [6M]

key	A	B	C	D
Probability	0.1	0.2	0.4	0.3

→

There are 4 nodes, n = 4

We will number the nodes (A, B, C, D as 1, 2, 3, 4) for sake of simplicity.

	1	2	3	4
a[i]	1	2	3	4
	0.1	0.2	0.4	0.3

The initial tables & Root table are as given below.

	0	1	2	3	
1	0	0.1			
		0	0.2		
2			0	0.4	
3				0	0.3
4					0
5					

cost table

	1	2	3	4
1	1			
2		2		
3			3	
4				4

Root table.

c[1,0] = 0
c[2,1] = 0
c[3,2] = 0
c[4,3] = 0
c[5,4] = 0

} using formula $c[i, i-1] = 0$ & $c[n+1, n] = 0$.

$c[1,1] = 0.1$
$c[2,2] = 0.2$
$c[3,3] = 0.4$
$c[4,4] = 0.3$

} using formula $c[i,i] = p[i]$

The root table

$R[1,1] = 1$
$R[2,2] = 2$
$R[3,3] = 3$
$R[4,4] = 4$

} using formula $R[i,i] = i$

Now let us compute $c[i,j]$ diagonally using formula

$$c[i,j] = \min_{i \le k \le j} \{c[i,k-1] + c[k+1,j] + \sum_{s=i}^{j} p_s$$

① $c[1,2]$, $k = 1 \& 2$

$$c[i,j] = \min_{k=1,2} \{(c[1,0] + c[2,2] + p[1] + p[2]), \; c[1,1] + c[3,2] + p[1] + p[2]\}$$

$$= \min\{0.5, 0.4\}$$

$c[1,2] = 0.4$ for $k = 2$

$R[1,2] = 2$

$$c[2,3] = \min_{k=2,3} \{(c[2,1] + c[3,3] + p[2] + p[3]), \; c[2,2] + c[4,3] + p[2] + p[3])\}$$

$$= \min\{1.0, 0.8\}$$

$c[2,3] = 0.8$ when $k = 3$

$\therefore R[2,3] = 3$.

$c[3,4] = \min\{(c[3,2] + c[4,4] + p[3] + p[4]),$
$k=3,4 \quad (c[3,3] + c[5,4] + p[3] + p[4])\}$

$= \min\{1.0, 1.1\}$

$c[3,4] = 1.0$ when $k = 3$

$R[3,4] = 3.$

∴ The resulting table are,

	1	2	3	4	5
1	0	0.1	0.4		
2		0	0.2	0.8	
3			0	0.4	1.8
4				0	0.3
5					0

cost table

	1	2	3	4
1	1	2		
2		2	3	
3			3	3
4				4

Root table.

② $c[1,3] = \min\{(c[1,0] + c[2,3] + p[1] + p[2] + p[3]),$
$k=1,2,3 \quad (c[1,1] + c[3,3] + p[1] + p[2] + p[3]),$
$(c[1,2] + c[4,3] + p[1] + p[2] + p[3])\}$

$= \min\{1.5, 1.2, 1.1\}$

$c[1,3] = 1.1$

$R[1,3] = 3$

$c[2,4] = \min\{(c[2,1] + c[3,4] + p[2] + p[3] + p[4]),$
$k=2,3,4 \quad (c[2,2] + c[4,4] + p[2] + p[3] + p[4]),$
$(c[2,3] + c[5,4] + p[2] + p[3] + p[4])\}$

$= \min\{1.9, 1.4, 1.7\}$

$c[2,4] = 1.7$

$R[2,4] = 3.$

The resulting cost table & root table are

	0	1	2	3	4
1	0	0.1	0.4	1.1	
2		0	0.2	0.8	1.4
3			0	0.4	1.0
4				0	0.3
5					0

	1	2	3	4
1	1	2	3	
2		2	3	3
3			3	3
4				4

③ $C[1,4] = \min_{k=1,2,3,4} \{ (C[1,0]+C[2,4]+P[1]+P[2]+P[3]+P[4]),$
$(C[1,1]+C[3,4]+P[1]+P[2]+P[3]+P[4]),$
$(C[1,2]+C[4,4]+P[1]+P[2]+P[3]+P[4]),$
$(C[1,3]+C[5,4]+P[1]+P[2]+P[3]+P[4]) \}$

$= \min \{2.4, 2.1, 1.7, 2.3\}$

$C[1,4] = 1.7$

$R[1,4] = 3$

The resulting cost table & root table are

	0	1	2	3	4
1	0	0.1	0.4	1.1	1.7
2		0	0.2	0.8	1.4
3			0	0.4	1.0
4				0	0.3
5					0

Cost table

	1	2	3	4
1	1	2	3	3
2		2	3	3
3			3	3
4				4

Root table

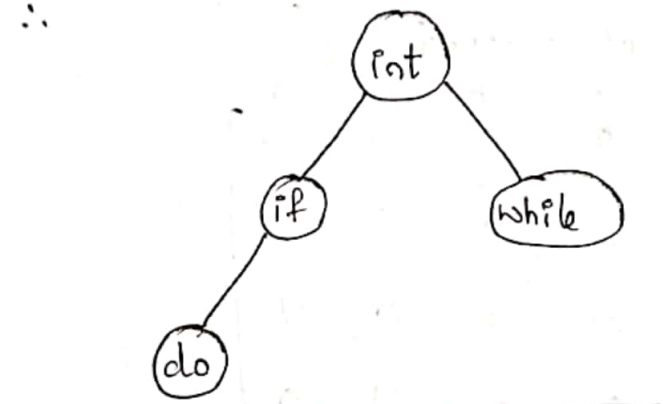
To construct tree, we have R[1][n] = R[1][4] :3

We assumed

	1	2	3	4
a[i]	do	if	int	while

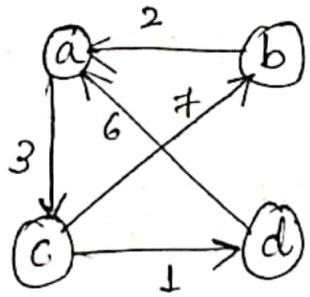
↳ int is root node.

∴



C[1, 4] = 1.7 = optimal cost

8a Apply Floyds algorithm to find all pair shortest path for the given graph. [7M]



→ Cost adjacency matrix for the above graph is

	a	b	c	d
a	0	∞	3	∞
b	2	0	∞	∞
c	∞	7	0	1
d	6	∞	∞	0

Step1 : consider shortest distance through vertex a

consider ath row & ath column elements

$(b,a) = 2 \quad (a,c) = 3 \Rightarrow (b,c) = \min\{(b,c), (b,a)+(a,c)\}$

$= 5$

$(d,a) = 6 \quad (a,c) = 3 \Rightarrow (d,c) = \min\{(d,c), (d,a)+(a,c)\}$

$= 9$

The resultant matrix is,

$$\begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{bmatrix} 0 & \infty & 3 & \infty \\ 2 & 0 & 5 & \infty \\ \infty & 7 & 0 & 1 \\ 6 & \infty & 9 & 0 \end{bmatrix} \end{array}$$

Step2 : consider shortest distance through vertex b.

consider bth row & bth column elements

$(c,b) = 7, (b,a) = 2 \Rightarrow (c,a) = \min(c,a), (c,b)+(b,a)\}$

$= 9$

$(c,b) = 7 \quad +(b,c) = 5 \Rightarrow (c,c) = \min(c,c); (c,b)+(b,c)\}$

$= 0.$

The resultant matrix is

$$\begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{bmatrix} 0 & \infty & 3 & \infty \\ 2 & 0 & 5 & \infty \\ 9 & 7 & 0 & 1 \\ 6 & \infty & 9 & 0 \end{bmatrix} \end{array}$$

Step 3 : consider shortest distance through vertex c

consider cth row & cth column elements

$(a,c) = 3 \quad (c,a) = 9 \Rightarrow (a,a) = \min\{(a,a), (a,c)+(c,a)\}$

$= 0$

$(a,c) = 3$, $(c,b) = 7 \Rightarrow (a,b) = \min\{(a,b), (a,c) + (c,b)\}$

$= 10$

$(a,c) = 3$ $(c,d) = 1 \Rightarrow (a,d) = \min\{(a,d), (a,c) + (c,d)\}$

$= 4$

$(b,c) = 5$ $(c,a) = 9 \Rightarrow (b,a) = \min\{(b,a), (b,c) + (c,a)\}$

$= 2$

$(b,c) = 5$ $(c,b) = 7 \Rightarrow (b,b) = \min\{(b,b), (b,c) + (c,b)\}$

$= 0$

$(b,c) = 5$ $(c,d) = 1 \Rightarrow (b,d) = \min\{(b,d), (b,c) + (c,d)\}$

$= 6.$

$(d,c) = 9$ $(c,a) = 9 \Rightarrow (d,a) = \min\{(d,a), (d,c) + (c,a)\}$

$= 6$

$(d,c) = 9$ $(c,b) = 7 \Rightarrow (d,b) = \min\{(d,b), (d,c) + (c,b)\}$

$= 16$

$(d,c) = 9$ $(c,d) = 1 \Rightarrow (d,d) = \min\{(d,d), (d,c) + (c,d)\}$

$= 0$

The resultant matrix is

	a	b	c	d
a	0	10	3	4
b	2	0	5	6
c	9	7	0	1
d	6	16	9	0

Step 4: consider shortest distance through vertex 4

consider 4^{th} row & 4^{th} column elements

$(a,d) = 4$ $(d,a) = 6 \Rightarrow (a,a) = \min\{(a,d), (a,d) + (d,a)\}$

$= 0$

$(a,d)=4 \quad (d,b)=16 \Rightarrow (a,b) = \min\{(a,b), (a,d)+(d,b)\}$

$= 10$

$(a,d)=4 \quad (d,c)=9 \Rightarrow (a,c) = \min\{(a,c), (a,d)+(d,c)\}$

$= 3$

$(b,d)=6 \quad (d,a)=6 \Rightarrow (b,a) = \min\{(b,a), (b,a)+(d,a)\}$

$= 2$

$(b,d)=6 \quad (d,b)=16 \Rightarrow (b,b) = \min\{(b,b), (b,d)+(d,b)\}$

$= 0$

$(b,d)=6 \quad (d,c)=9 \Rightarrow (b,c) = \min\{(b,c), (b,d)+(d,c)\}$

$= 5$

$(c,d)=1 \quad (d,a)=6 \Rightarrow (c,a) = \min\{(c,a), (c,d)+(d,a)\}$

$= 7$

$(c,d)=1 \quad (d,b)=16 \Rightarrow (c,b) = \min\{(c,b), (c,d)+(d,b)\}$

$= 7$

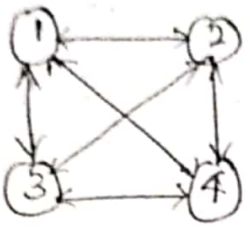
$(c,d)=1 \quad (d,c)=9 \Rightarrow (c,c) = \min\{(c,c), (c,d)+(d,c)\}$

$= 0$

The resultant matrix is

	a	b	c	d
a	0	10	3	4
b	2	0	5	6
c	7	7	0	1
d	6	16	9	0

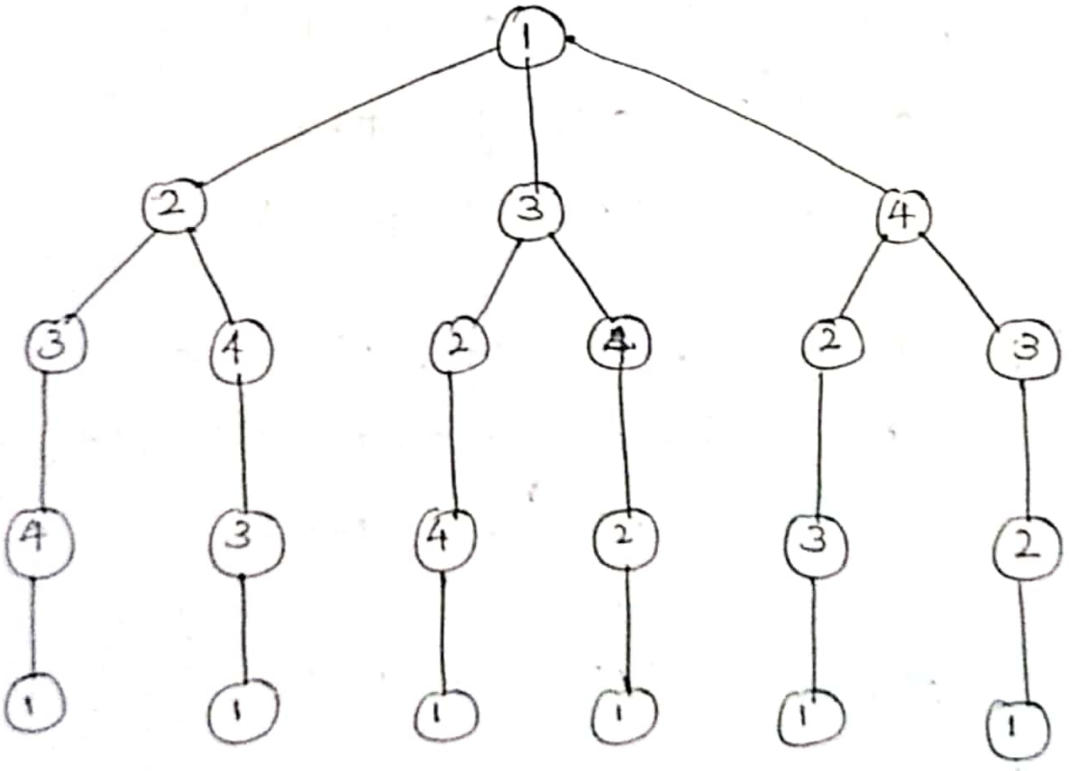
8 b Find the optimal tour for sales person using dynamic programming technique for the given graph & its corresponding edge length matrix.



$$\begin{bmatrix} 0 & 10 & 15 & 20 \\ 5 & 0 & 9 & 10 \\ 6 & 13 & 0 & 12 \\ 8 & 8 & 9 & 0 \end{bmatrix}$$

[7M]

→ We assume traveling sales person starts from city ① and visits all other cities exactly once & returns back to city ①. The possible paths can be shown using the tree given below



Step 1: Suppose we are at vertex 2 (consider leaf nodes), we have no other vertex to visit, this means we have to visit vertex 1.

$g(2, \phi) = c21 = 5$

Similarly $g(3, \phi) = c31 = 6$ (at vertex 3)

$g(4, \phi) = c41 = 8$ (at vertex 4)

Step 2: Move one level up, if we are at vertex 2, then travelling can move to vertex 3 or vertex 4

$g(2, \{3\}) = c23 + g(3, \phi) = 15$

$g(2, \{4\}) = c24 + g(4, \phi) = 18$

Similarly if we are vertex 3

$g(3, \{2\}) = c32 + g(2, \phi) = 18$

$g(3, \{4\}) = c34 + g(4, \phi) = 20$

Similarly if we are vertex 4

$g(4, \{2\}) = c42 + g(2, \phi) = 13$

$g(4, \{3\}) = c43 + g(3, \phi) = 15$

Step 3: Move one level up., if we are at vertex 2

$g(2, \{3,4\}) = \min\{c23 + g(3, \{4\}), c24 + g(4, \{3\})\}$

$= \min\{9+20, 10+15\}$

$= 25$

Similarly if we are vertex 3

$g(3, \{2,4\}) = \min\{c32 + g(2, \{4\}), c34 + g(4, \{2\})\}$

$= \min\{13+18, 12+13\}$

$= 25$

→ If we are netta 3 then

$$g(4,\{2,3\}) = \min\{c_{42} + g(2,\{3\}),\ c_{43} + g(3,\{2\})\}$$

$$= \min\{8+15,\ 9+18\}$$

$$= 23$$

Step 4 : Move one level up, we are at vertex 1

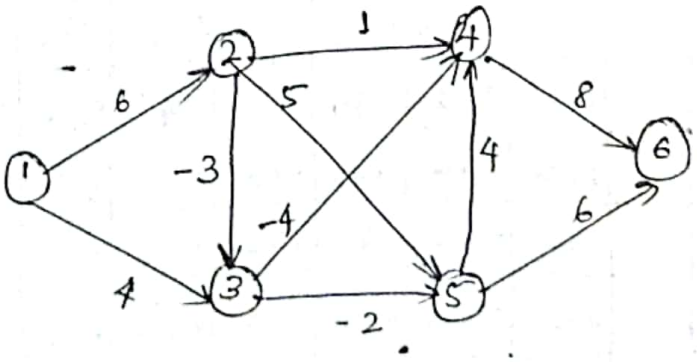
$$g(1,\{2,3,4\}) = \min\{c_{12} + g(2,\{3,4\}),\ c_{13} + g(3,\{2,4\}),\ c_{14} + g(4,\{2,3\})\}$$

$$= \min\{10+25,\ 15+25,\ 20+23\}$$

$$\boxed{g(1,\{2,3,4\}) = 35}$$

An optimal tour of the given graph has length 35 and the optimal tour is 1, 2, 4, 3, 1

8c. Find the shortest path from node 1 to every other node in the given graph using Bellman-Ford algorithm. [6M]



Solution:

The shortest path can be computed using Bellman Ford algorithm with the help of following recurrence relation

$$d[i] = cost(v, i)$$

$$d^k[u] = \min\{d^{k-1}[u], \min\{d^{k-1}[i] + cost[i, u]\}\}$$

Source = 1

cost adjacency matrix =

$$\begin{bmatrix} 0 & 6 & 5 & 5 & \infty & \infty & \infty \\ \infty & 0 & \infty & \infty & -1 & \infty & \infty \\ \infty & -2 & 0 & \infty & 1 & \infty & \infty \\ \infty & \infty & -2 & 0 & \infty & -1 & \infty \\ \infty & \infty & \infty & \infty & 0 & \infty & 3 \\ \infty & \infty & \infty & \infty & \infty & 0 & 3 \\ \infty & \infty & \infty & \infty & \infty & \infty & 0 \end{bmatrix}$$

distance D

0	6	5	5	∞	∞	∞

When k = 2.

i	d	$d^2[2]$	$d^2[3]$	$d^2[4]$	$d^2[5]$	$d^2[6]$	$d^2[7]$
		5	3	5	5	4	∞
1	0	0+6	0+5	0+5	0+∞	0+∞	0+∞
2	6	6	6+∞	6+∞	6−1	6+∞	6+∞
3	5	5+0	5+∞	5+∞	5+1	5+∞	5+∞
4	5	∞+∞	5−2	5+0	5+∞	5−1	5+∞
5	∞	∞+∞	∞+∞	∞+∞	∞+∞	∞+∞	∞+3
6	∞	∞+∞	∞+∞	∞+∞	∞+∞	∞+0	∞+3
7	∞	∞+∞	∞+∞	∞+∞	∞+∞	∞+∞	∞+0
	min =	3	3	5	5	4	6

When k = 3

	d	i	3	3	5	5	9	∞
			$d^3[1]$	$d^3[3]$	$d^3[4]$	$d^3[5]$	$d^3[6]$	$d^3[7]$
1	0	→ 1	0+6	0+5	0+5	0+∞	0+∞	0+∞
2	3	→ 2	3+0	3+∞	3+∞	3−1	3+∞	3+∞
3	3	→ 3	3−2	3+0	3+∞	3+1	3+∞	3+6
4	5	→ 4	5+∞	5−2	5+0	5+∞	5−1	5+∞
5	5	→ 5	5+∞	5+∞	5+∞	5+0	5+∞	5+3
6	4	→ 6	4+∞	4+∞	4+∞	4+∞	4+0	4+3
7	∞	→ 7	∞+∞	∞+∞	∞+∞	∞+∞	∞+∞	∞+0

min = 1 3 5 2 4 7

When k = 4

	d		$d^4[2]$	$d^4[3]$	$d^4[4]$	$d^4[5]$	$d^4[6]$	$d^4[7]$
			1	3	5	2	4	7
1	0	→ 1	0+6	0+5	0+5	0+∞	0+∞	0+∞
2	1	→ 2	1+0	1+∞	1+∞	1−1	1+∞	1+∞
3	3	→ 3	3−2	3+0	3+∞	3+1	3+∞	3+∞
4	5	→ 4	5+∞	5−2	5+0	5+∞	5−1	5+∞
5	2	→ 5	2+∞	2+∞	2+∞	2+0	2+∞	2+3
6	4	→ 6	4+∞	4+∞	4+∞	4+∞	4+0	4+3
7	7	→ 7	7+∞	7+∞	7+∞	7+∞	7+∞	7+0

min = 1 3 5 0 4 5

When k = 5

	d	i	1	3	5	0	4	5
			$d^5[2]$	$d^5[3]$	$d^5[4]$	$d^5[5]$	$d^5[6]$	$d^5[7]$
1	0	→ 1	0+6	0+5	0+5	0+∞	0+∞	0+∞
2	1	→ 2	1+0	1+∞	1+∞	1−1	1+∞	1+∞
3	3	→ 3	3−2	3+0	3+∞	3+1	3+∞	3+∞
4	5	→ 4	5+∞	5−2	5+0	5+∞	5−1	5+∞
5	0	→ 5	0+∞	0+∞	0+∞	0+0	0+∞	0+3
6	4	→ 6	4+∞	4+∞	4+∞	4+∞	4+0	4+3
7	5	→ 7	5+∞	5+∞	5+∞	5+∞	5+∞	5+0

min = 1 3 5 0 4 3

When k=6 $d^6[2]$ $d^6[3]$ $d^6[4]$ $d^6[5]$

	d		1	3	5	0	4	3
1	0	→ 1	0+6	0+5	0+5	6+∞	0+∞	0+∞
2	1	→ 2	1+0	1+∞	1+∞	1−1	1+∞	1+∞
3	3	→ 8	3−2	3+0	3+∞	3+1	3+∞	3+∞
4	5	→ 4	5+∞	5−2	5+0	5+∞	5−1	5+∞
5	0	→ 5	0+∞	0+∞	0+∞	0+0	0+∞	0+3
6	4	→ 6	4+∞	4+∞	4+∞	4+∞	4+0	4+3
7	3	→ 7	3+∞	3+∞	3+∞	3+∞	3+∞	3+0.
	min		1	3	5	0	4	3

The final distance matrix for k = 2, 3, 4, 5, 6 is shown below.

	1	2	3	4	5	6	7
Initial distance d	0	6	5	5	∞	∞	∞
k=2	0	3	3	5	5	4	∞
k=3	0	1	3	5	2	4	7
k=4	0	1	3	5	0	4	5
k=5	0	1	3	5	0	4	3
k=6	0	1	3	5	0	4	3

Shortest distance 1 to all other vertices

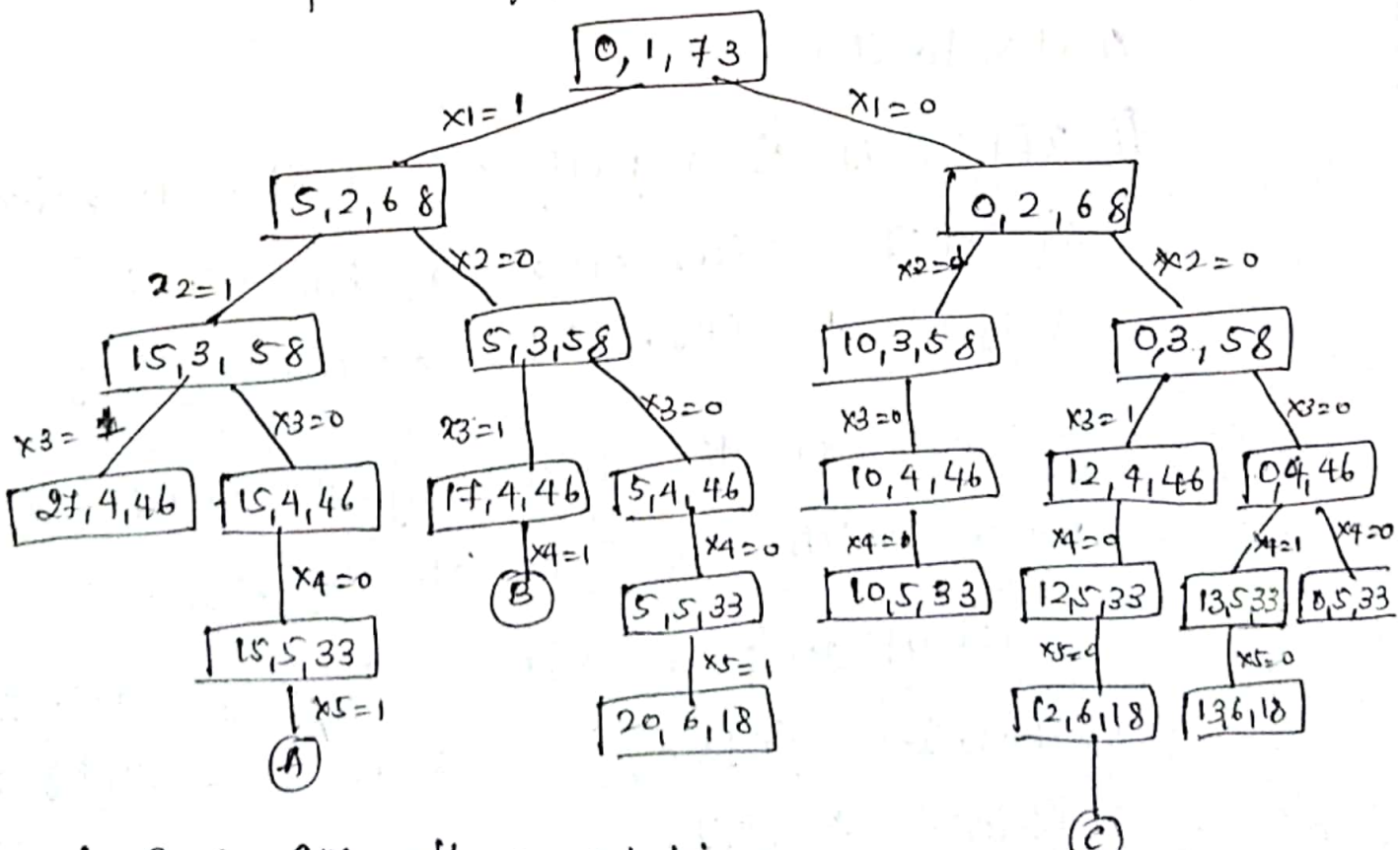
MODULE-5

Qa. What is the central Principle of backtracking? Apply backtracking to solve the below instance of some of subset problem $S = \{5, 10, 12, 13, 15, 18\}$ $d = 30$ [7M]

→ The principle idea is to construct solutions one component at a time & evaluate such partially constructed candidates as follows.

i) If partially constructed solution can be developed further without violating the problem constraints. It is done by taking the 1st remaining option for next component

ii) If not, alternative for remaining component need to be considered. In this case algorithm backtracks & replace last component of partial solution with its next option



A, B, C are three solutions

A = (1, 1, 0, 0, 1) B = (1, 0, 1, 1) C = (0, 0, 1, 0, 0, 1)

9c) What is a Hamiltonian circuit problem? What is the procedure to find Hamiltonian circuit of a graph? [6M]

→ Hamiltonian Circuit problem is a problem to find Hamiltonian cycles in a given graph. Hamiltonian cycle is a round trip path along n edges of G that visits every vertex once & comes to its starting position.

- In this problem we check for ~~the~~ all possible Hamiltonian cycles in a given graph.

Algorithm

Next value(k)

// x[1:k-1] is a path of k-1 distinct values if x[k]=0, then no vertex has yet been assigned to x[k]. After execution x[k] is assigned to the next highest numbered vertex which does not already appear in x[1:k-1] & is connected by an edge to x[k-1] Otherwise x[k]=0, If k=n, then x[k] is connected to x[1].

{

```
repeat
{
    x[k] = (x[k] + 1) mod (n+1);
    If (x[k] = 0) then return;
    If (G[x[k-1], x[k]] ≠ 0) then
    {
        for j = 1 to k-1 do
            if (x[j] = x[k]) then break;

        if (j = k) then
        if ((k < n) or ((k = n) & G[x[n], x[1]] ≠ 0))
        then return;
    }
} until (false);
}
```

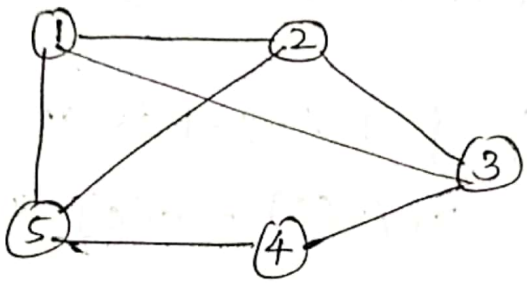
```
Algorithm Hamiltonian(k)
// This algorithm uses the recursive formulation
   of backtracking to find all the Hamiltonian
   cycles of a graph. The graph is stored as an
   adjacency matrix G[1:n, 1:n]. All cycles
   begin at node 1.
{
  repeat
    {
      nextvalue(k);
      if (x[k] = 0) then return;
          if (k = n) then write (x[1:n]);
```

else Hamiltonian (k+1);

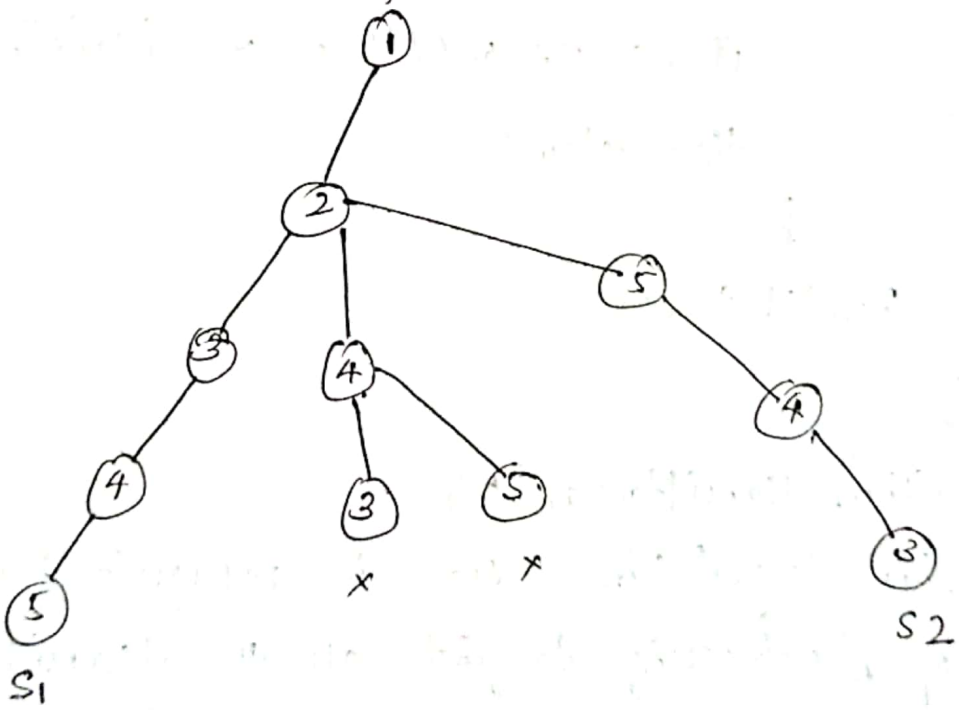
} until (false);

}

example



hamiltonian paths using above algorithm are



S1 & S2 are two paths obtained.

9b Solve the below instance of assignment problem using branch and bound algorithm.

Person	Job1	Job2	Job3	Job4
a	9	2	7	8
b	6	4	3	7
c	5	8	1	8
d	7	6	9	4

[7m]

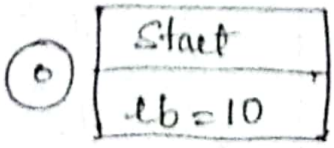
Solution

Now let us find the lower bound for the given problem. The cost of any solution, including the optimal solution can be smaller than the sum of the smallest element in each row of matrix. So take minimum of each row & add them to get the initial lower bound as shown below.

	J_1	J_2	J_3	J_4	Minimum
a	9	2	7	8	2
b	6	4	3	7	3
c	5	8	1	8	1
d	7	6	9	4	4

lower bound $lb = 10$.

The initial state space tree with lower bound is shown below



Consider persons: Assign various jobs to a 4 computer the lower bound as shown below

Let a → 1 whose cost = 9

Since a → 1, leave row a, column 1

@	J1	J2	J3	J4	Minimum
a	9	~~2~~	~~7~~	~~8~~	9
b	~~6~~	4	3	7	3
c	5	8	1	8	1
d	~~7~~	6	9	4	4

(1)

lb = 9+3+1+4

= 17

Let a = 2, whose cost = 2

Since a → 2, leave row a, Column 2.

	J1	J2	J3	J4	Minimum
a	~~9~~	2	~~7~~	~~8~~	2
b	6	~~4~~	3	7	3
c	5	~~8~~	1	8	1
d	7	~~6~~	9	4	4

(2)

lb = 2+3+1+4

= 10

Let a → 3 whose cost = 7

Since a → 3, leave row a, column 3

	J1	J2	J3	J4	Minimum
a	~~9~~	~~2~~	~~7~~	~~8~~	
b	6	4	~~3~~	~~7~~	
c	~~5~~	8	~~1~~	~~8~~	
d	~~7~~	6	~~8~~	4	

lb = 7+4+5+4

= 20

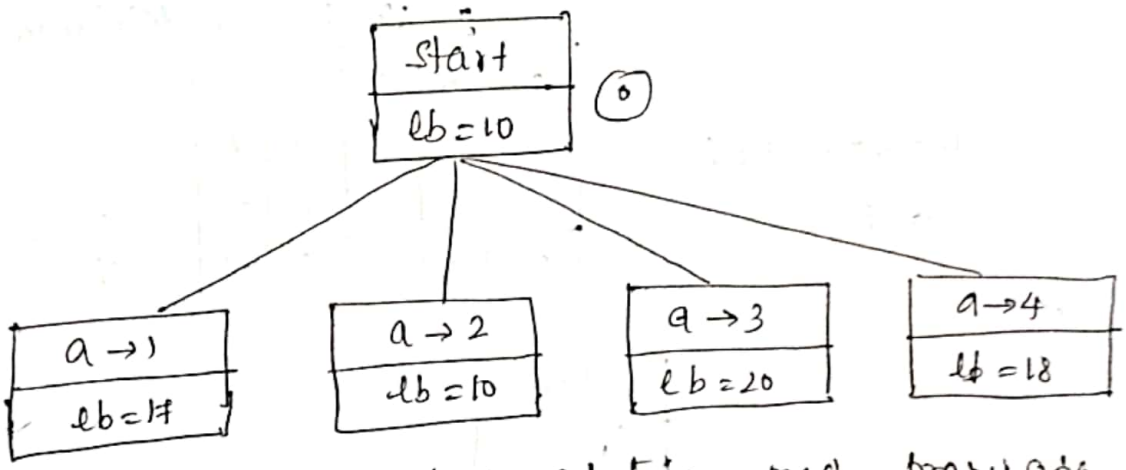
Let a → 4 where cost = 8

Since a → 4, leave row a column 4

	J1	J2	J3	J4	Minimum
a	~~9~~	~~2~~	~~7~~	~~8~~	8
b	6	4	3	~~7~~	3
c	5	8	1	8	1
d	7	6	9	~~4~~	6

lb = 8 + 3 + 1 + 6

= 18.



In the above partial solution, most promising solution is the one which has least lower bound with lb = 10., Here a is assigned job2 with cost = 2.

Consider person b: Assign various jobs to person b (leaving out job2 which is assigned to a. with cost = 2) & compute the lower bound as shown below

Let b → 1, where cost = 6

Since b → 1, leave row b, column 1.

	J1	J2	J3	J4	Minimum
a		2			2
b	6		~~3~~	~~7~~	~~6~~
c	~~8~~		1	8	1
d	~~7~~		9	4	4

lb = 2 + 6 + 1 + 4

= 13

Let b→3 where cost = 3

Since b→3, leave row b, column 3

		J1	J2	J3	J4	Minimum
a	a	·	2			2
b	b	6		3	7	3
c	c	5		∞	8	5
d	d	7		∞	4	4.

lb = 2+3+5+4

= 14

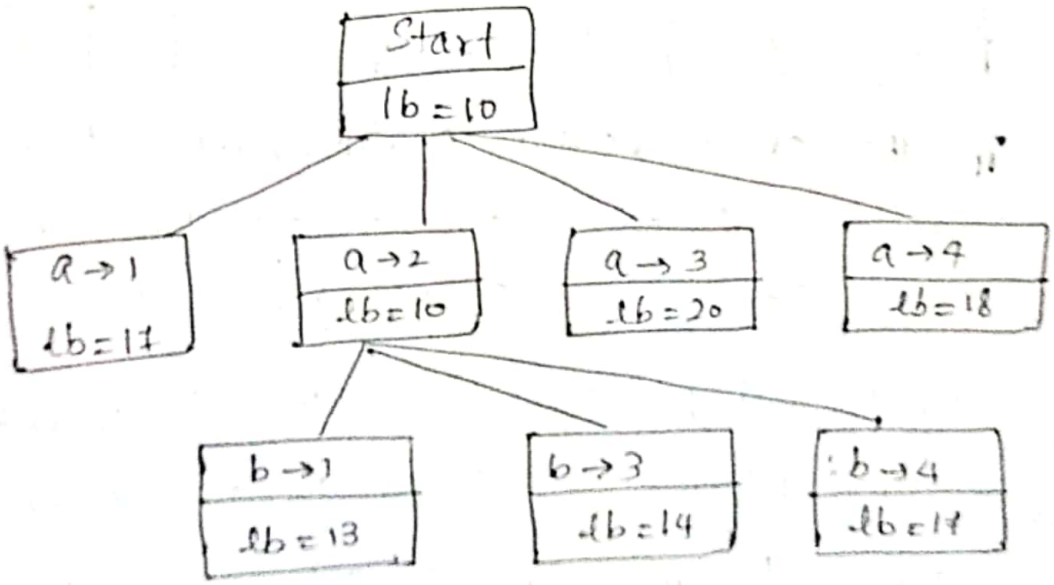
Let b→4 cost = 7

Since b→4 leave row b, column 4.

	J1	J2	J3	J4	Minimum
a	∞	2			2
b	6		3	7	7
c	5		1	8	1
d	7		9	4	7

lb = 2+7+1+7

= 17



In the above partial solution, most promising solution is the one which has least lower bound i.e. node 5 with lb = 13. Here a is assigned job2 with cost = 2 & b is assigned job1 with cost = 6.

consider person c's Assign various jobs to person c clearing out job 2 & job 1 which is assigned to a and b with cost = 6) & compute the lower bound as shown below

Let $c \rightarrow 3$, where cost = 1

Since $c \rightarrow 3$, leave row c column 3

	J1	J2	J3	J4	Minimum
a		2			2
b	6				6
c			1	~~8~~	1
d			9	4	4

$lb = 2+6+1+4$

$lb = 13$

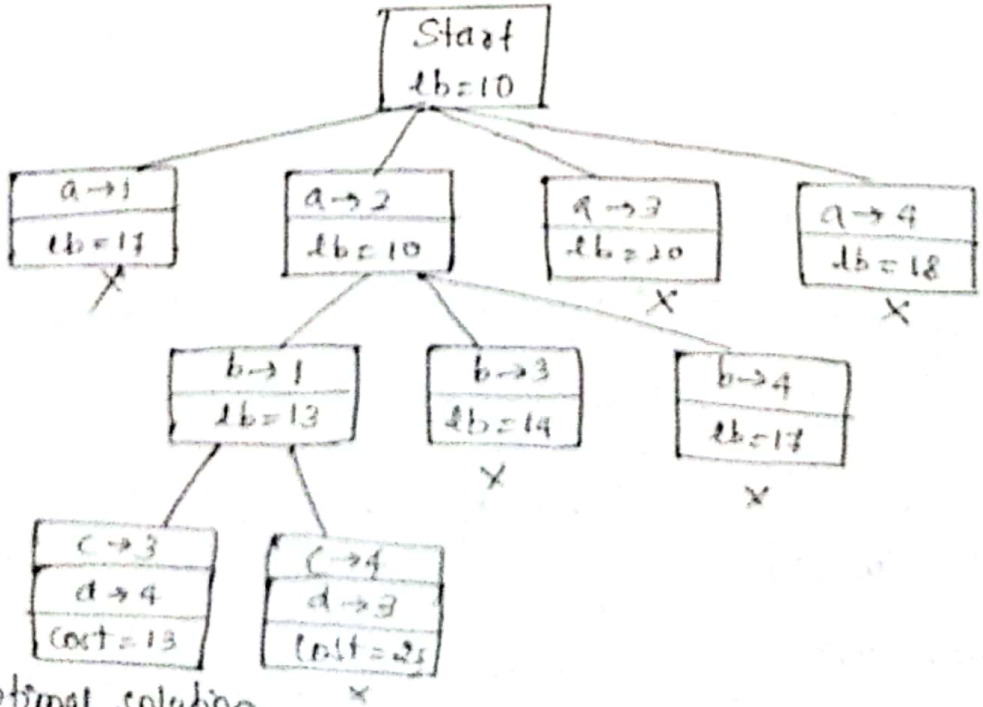
Let $c \rightarrow 4$ where cost = 8

Since $c \rightarrow 4$, leave row c column 4

	J1	J2	J3	J4	
a		2			2
b	6				6
c			X	8	8
d			9	4	9

$lb = 9+6+8+9$

$lb = 25$



Optimal solution

So, the optimal solution is

$a \to 2, b \to 1, c \to 3, d \to 4$

cost $= 2 + 6 + 1 + 4$

cost $= 13.$

10 a Illustrate N Queens Problem using backtracking to solve 4 Queens problem. [8M]

→ Solution

The n-queens are to be placed on a n×n chessboard so that no two attack i.e. no two queens are on the same row, column or diagonal.

For a 4 queens problem, we take a 4×4 chessboard & 4 queens $Q1, Q2, Q3, Q4$. We have more than one solution & we need to generate all solutions, so we use backtracking.

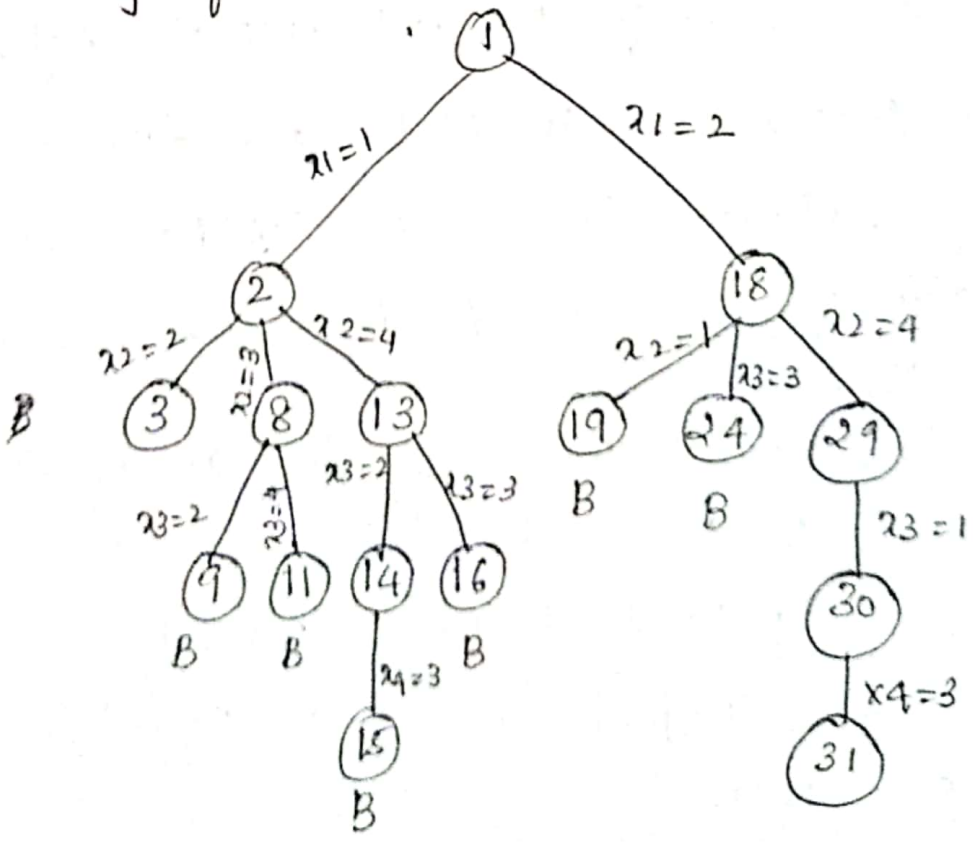
Algorithm to checks whether queen can be placed or not

```
Place(k, i)
{
  for j = 1 to k-1 do
    if x[j] = i // Two in the same column
       or abs(x[j] - i) = abs(j - k))
          // is in same diagonal
       then return false
  return true;
}
```

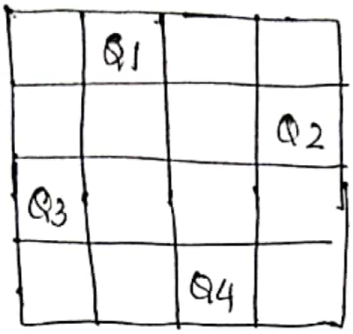
Algorithm to obtain all solutions to n queen problem

```
NQueens(k, n)
{
  for i := 1 to n do
  {
    if (Place(k, i) then
    {
      x[k] = i;
      if(k = n) then write(x[1:n]);
      else NQueens(k+1, n);
    }
  }
}
```

State space tree generated using backtracking with bounding function



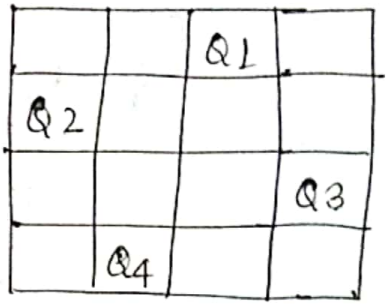
Solution 1



X	2	4	1	3

Solution 2

Take mirror image of Solution 1



X	3	1	4	2

Q10b Explain the following

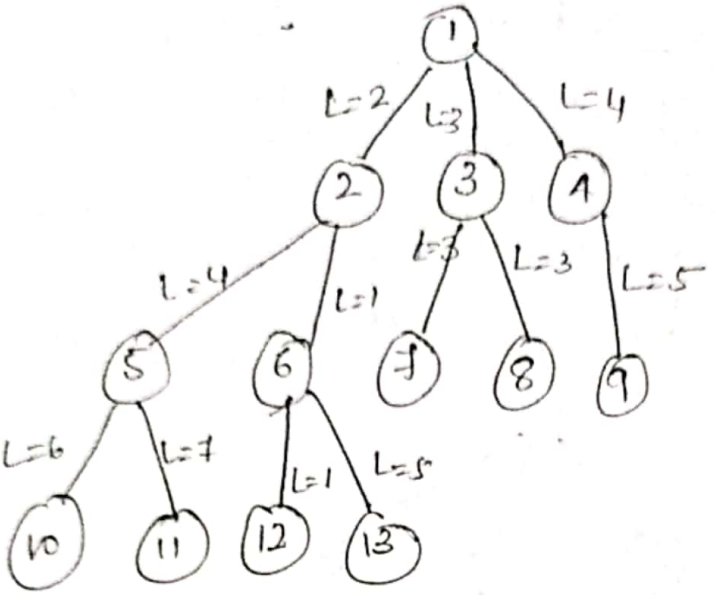
a) LC Branch & bound

b) FIFO branch & bound. [6M]

→ a) LC Branch & Bound

In both FIFO & LIFO Branch & bound the selection rules for the next-node is rigid & blind. The selection rule for next-E node does not give any preferences to a node that has a very good chance of getting the search to an answer node quickly.

In this we will use ranking function or cost function. We generate the children of E-node, among these live nodes; we select a node which has minimum cost. By using ranking function we will calculate the cost of each node.



Initially we take node 1 as E-node. Generate children of node 1, the children are 2, 3, 4.

By using ranking function we calculate the cost of 2, 3, 4 nodes, $c=2$, $c=3$, $c=4$ respectively.

Now we select a node which has minimum cost i.e. node 2.

For node 2, the children are 5, 6. Between 5 and 6, we will select 6 since its cost is minimum.

Generate children of node 6 i.e. 12 & 13. We will select node 12 since its cost $c=1$ is minimum.

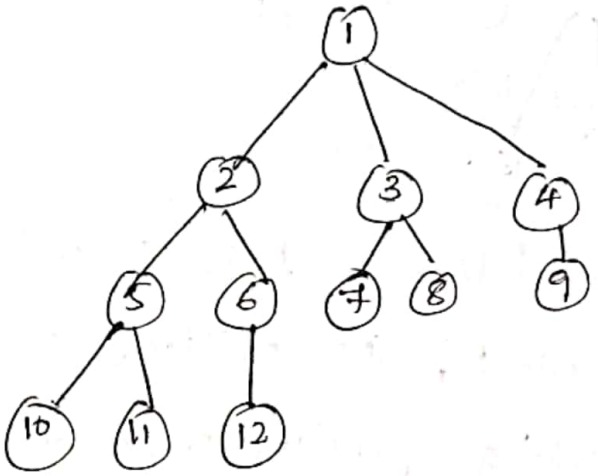
Moreover, 12 is the answer node. So, we terminate search process.

b) FIFO Branch and Bound

For this we use a data structure called Queue.

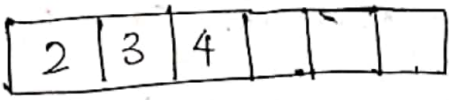
Initially Queue is empty

example

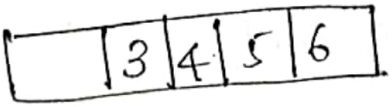


Assume the node 12 is answer node.

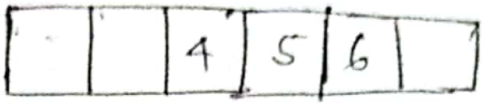
In FIFO search, first we will take E-node as a node 1. Next we generate the children of node 1. we will place all these live nodes in a queue



Now we delete an element from queue ie node 2, next generate children of node 2 & place in this queue.



Next delete an element from queue & take it as E-node, generate the children of node 3, 7, 8 are children of 3 & these live nodes are killed by bounding functions. so we will not include in the queue.



Again delete an elements from queue. Take it as E-node, generate the children of 4. Node 9 is generated & killed by bounding function

Next delete an element from queue, Generate Children of node 5 i.e. nodes 10 & 11 are generated & by bounding function, last node in queue is 6 The child of node 6 is 12 & it satisfies the conditions of the problem, which is the answer node, so search terminate

10 C. Explain the classes of NP-hard & NP-Complete problems [6M]

→ NP is the set of all decision problems solvable by non deterministic algorithms in polynomial time.

P denotes the class of all deterministic polynomial problems Hence $P \subseteq NP$.

Problems which are known to lie in P are often called as tractable. Problems which lie outside of P are often termed as intractable.

The relationship between P & NP is depicted in figure below

Let L_1 & L_2 be problems. Problem L_1 reduces to L_2 ($L_1 \propto L_2$) iff there is a way to solve L_1 by a deterministic polynomial time algorithm using a deterministic algorithm that solves L_2 in polynomial time. This definition implies that if we have a polynomial time algorithm for L_2, then we can solve L_1 in polynomial time

one can readily verify $\propto$ is a transitive relation

if $L_1 \propto L_2$ & $L_2 \propto L_3$ then $L_1 \propto L_3$

- A problem L is <u>NP-hard</u> iff satisfiability reduces to L.
- A problem L is NP-complete iff L is NP & $L \in NP$
- It is easy to see that there are NP hard problems that are not NP complete. Only a decision problem can be NP-complete.
- However an optimization problem may be NP-hard. If L_1 is a decision problem & L_2 an optimization problem then $L_1 \propto L_2$.
- One can show that knapsack problem reduces to knapsack optimization problem.
- Optimization problems cannot be NP-complete while as decision problem can. This relationship is depicted below.