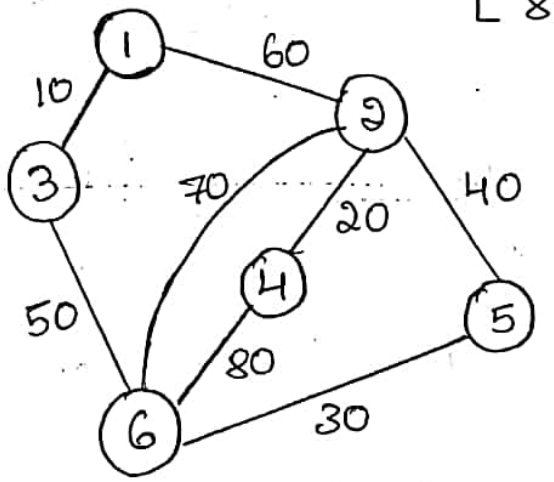
Fourth Semester B.E Degree Examination, Dec. 2018/Jan. 2019

Design and Analysis of Algorithms

5)

a) Explain the concept of greedy technique for Prime's algorithm obtain a minimum cost Spanning tree for the graph shown in fig. [8 marks]



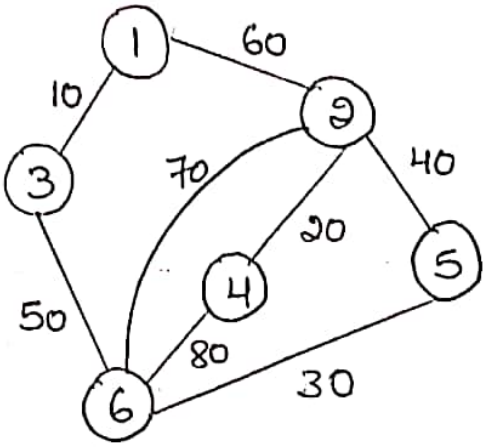
⇒ **Greedy technique for Prim's algorithm**

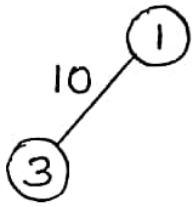
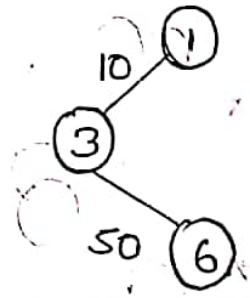
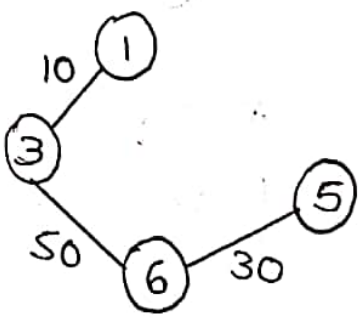
* Prim's Algorithm is a famous greedy algorithm
* It is used for finding the minimum spanning Tree (MST) of a given graph.
* this means it finds a subset of the edges that forms a tree that includes every vertex, where the total weight of all the edges

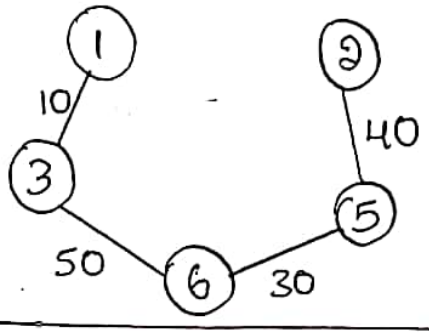
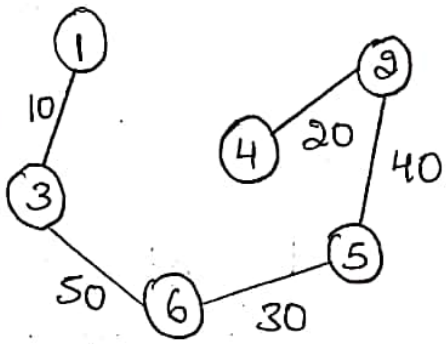
in the tree is minimized.

* To apply the prim's algorithm, the given graph must be weighted, connected and undirected.

Consider the given graph



Tree vertices	Remaining vertices	Illustration
3(-,-)	1(3,10), 6(3,50)	
1(3,10)	2(1,60), 6(3,50)	
6(3,50)	2(1,60) 4(6,80) 5(6,30)	

Tree vertices	Remaining vertices	Illustration
5(6, 30)	2(5, 40), 2(1, 60) 4(6, 80)	
2(5, 40)	4(2, 20)	

Cost of minimum Spanning Tree

= Sum of all edge weights

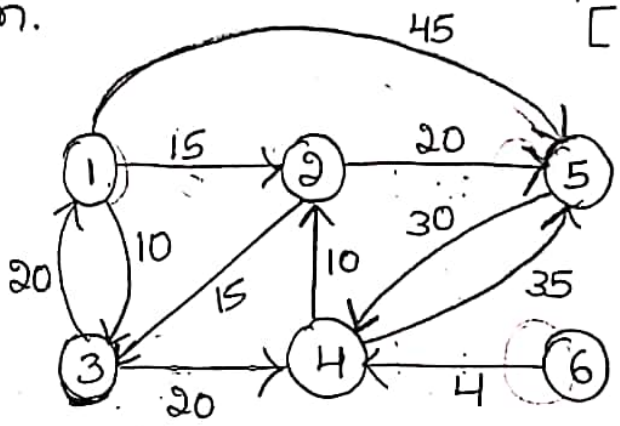
= 10 + 50 + 30 + 40 + 20

= **150 units**

∴ Cost of minimum spanning tree = 150 units

5)

b) Solve the below instance of the single source shortest path problem with vertex 6 as source. with the help of suitable algorithm. [8 marks]



⇒ Dijkstra's Algorithm → Single-source shortest-paths problem.

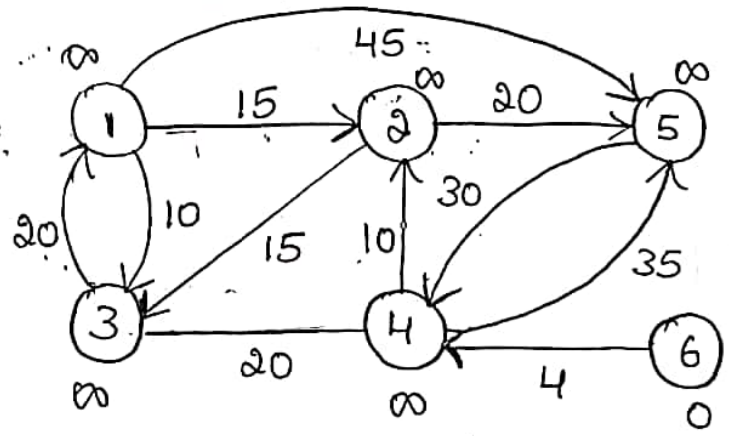
For a given vertex called the source in a weighted connected graph, it find shortest paths to all its other vertices.

Step 01:- The following two sets are considered

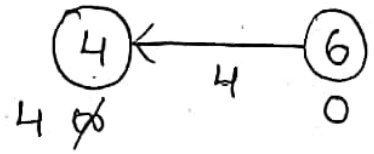
- Unvisited set = {1, 2, 3, 4, 5, 6}
- Visited set = { }

Step 02:- The two variables Π and d are created for each vertex & initialized as

- $\pi[1] = \pi[2] = \pi[3] = \pi[4] = \pi[5] = \pi[6]$
  $= NIL$
- $d[6] = 0$
- $d[1] = d[2] = d[3] = d[4] = d[5] = \infty$



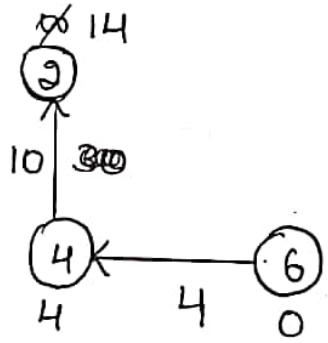
Step 3:- Vertex 6 is choosen, because it is source vertex.



$d[6] + 4 = 0 + 4 = 4 < \infty$

$\therefore d[4] = 4$ and $\pi[4] = 6$

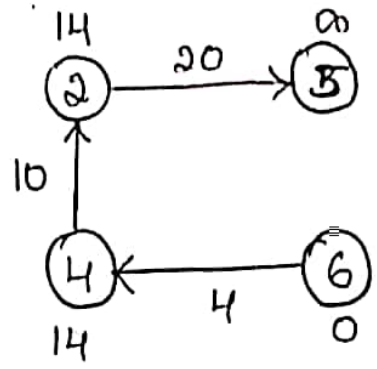
Step 4:-



$d[4] + 10 = 4 + 10 = 14 < \infty$

$d[2] = 14$ and $\pi[2] = 3$

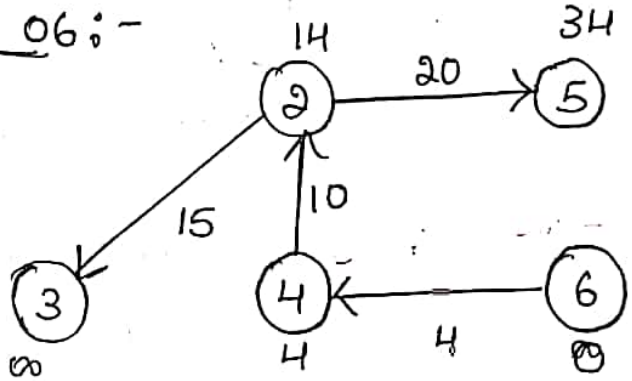
Step 05:-



$d[2] + 20 = 14 + 20 = 34 < \infty$

$d[5] = 34$ and $\Pi[5] = 2$

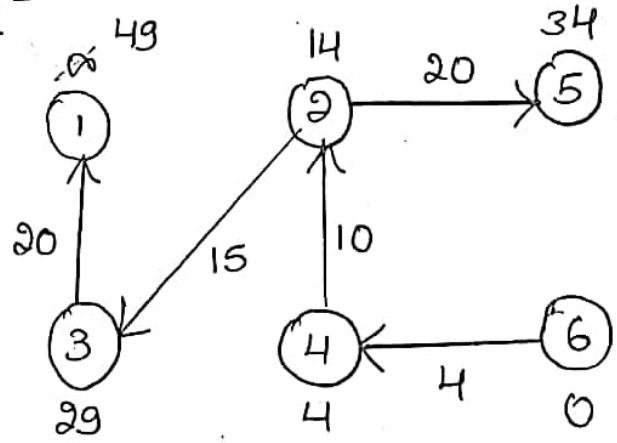
Step 06:-



$d[2] + 15 = 14 + 15 = 29 < \infty$

$\therefore d[3] = 29$ and $\Pi[3] = 2$

Step 07:-



$d[3] + 20 = 49 < \infty$

$\therefore d[1] = 49$ and $\Pi[1] = 3$

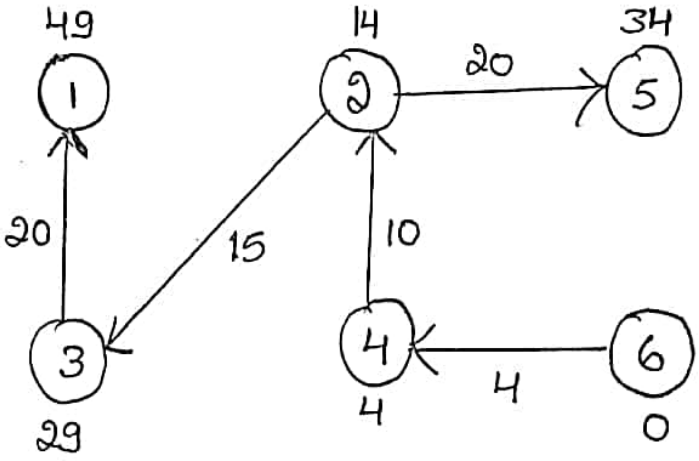
By considering all the steps now the updated sets are as follow

- unvisited set = { }
- visited set = {1, 2, 3, 4, 5, 6}

Now, all vertices of the graph are processed.

our final shortest path tree is as shown below.

from 6 to 4 : length 0 + 4 = 4
from 6 to 2 : length 4 + 10 = 14
from 6 to 5 : length 14 + 20 = 34
from 6 to 3 : length 14 + 15 = 29
from 6 to 1 : length 29 + 20 = 49



It represents the shortest path from Source vertex '6' to all other remaining vertices.

The order in which all the vertices are processed is: 6, 4, 2, 5, 3, 1.

06

a) What are Huffman tree? explain. Construct a Huffman code for the following data: [8 marks]

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

Encode DAD_CBE using Huffman encoding.

⇒ Huffman coding tree or Huffman tree is a full binary tree in which each leaf of the tree corresponds to a letter in the given alphabet.

Huffman coding :-

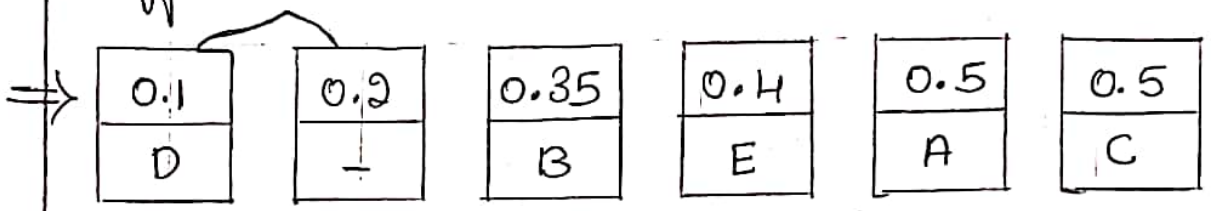
* Huffman coding is a famous greedy Algorithm
* It is used for the lossless compression of data.
* It uses variable length encoding.
* It assigns variable length code to all the characters.
* The code length of a character depends on how frequently it occurs in the given text.
* The character which occurs most frequently gets the smallest code.

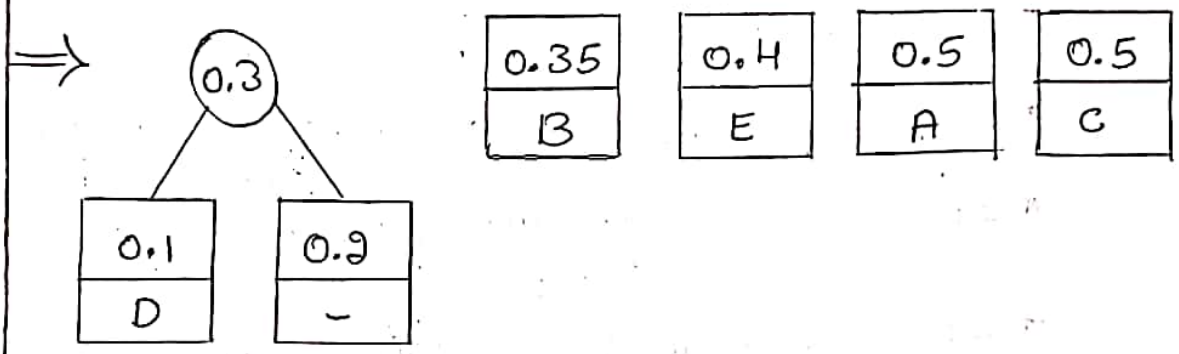
* The character which occurs least frequently gets the largest code.
* It is also known as Huffman Encoding.

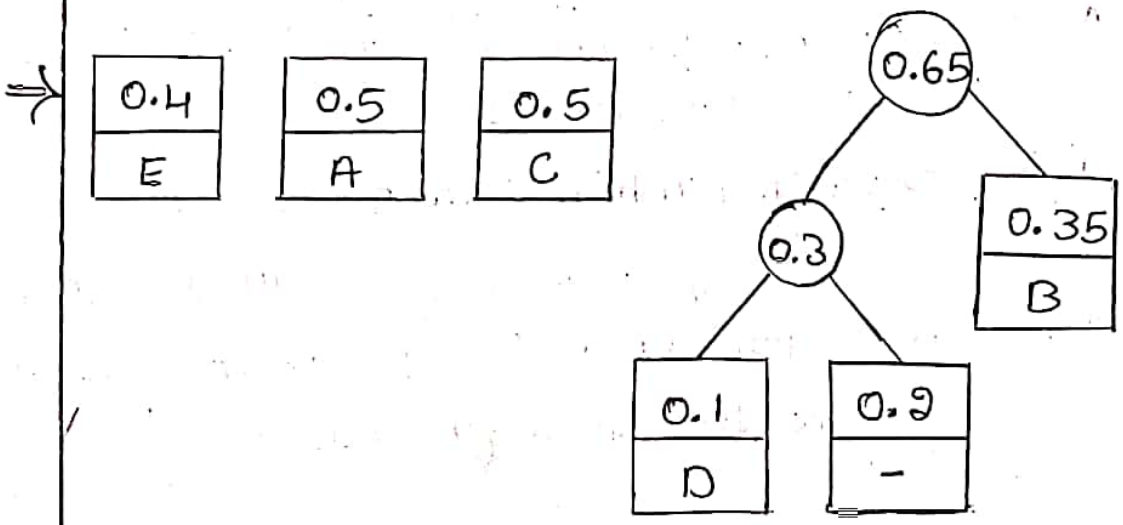
Consider the given data.

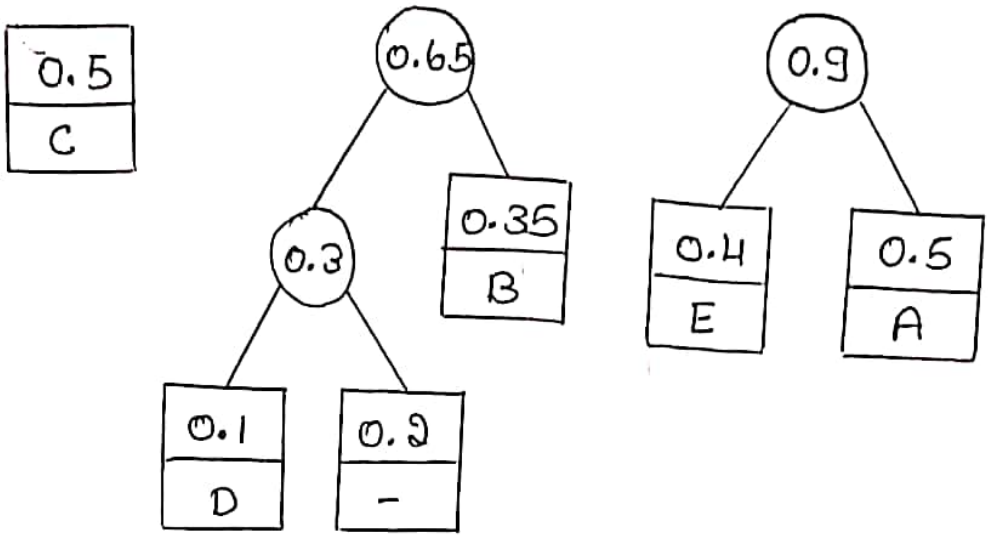
character	A	B	C	D	E	-
probability	0.5	0.35	0.5	0.1	0.4	0.2

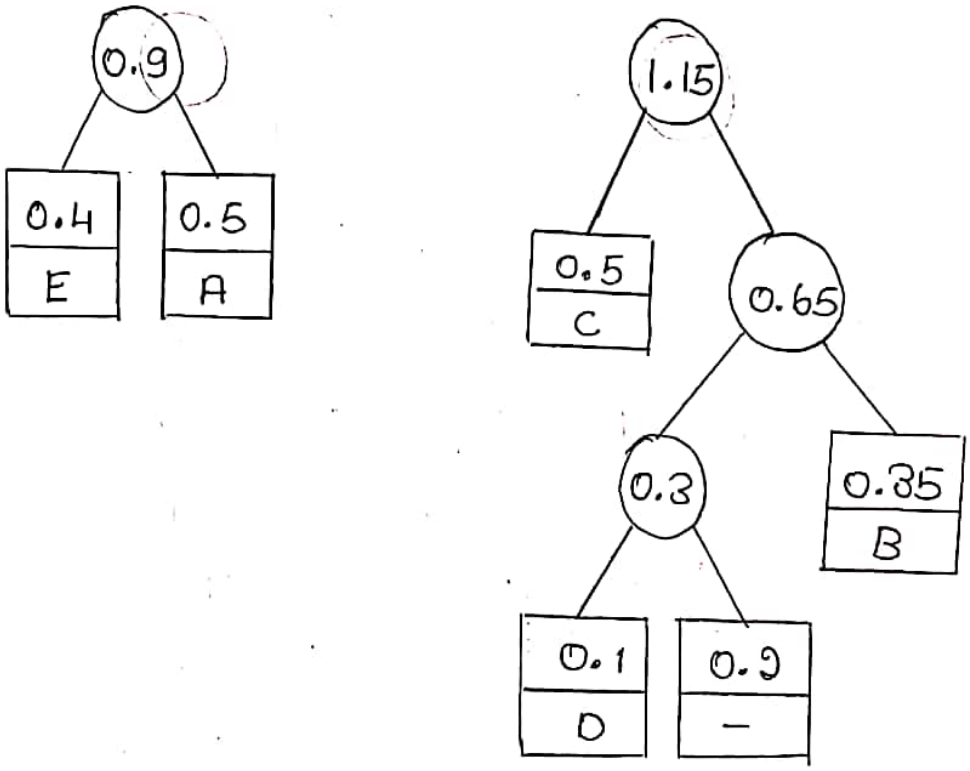
Huffman tree construction.

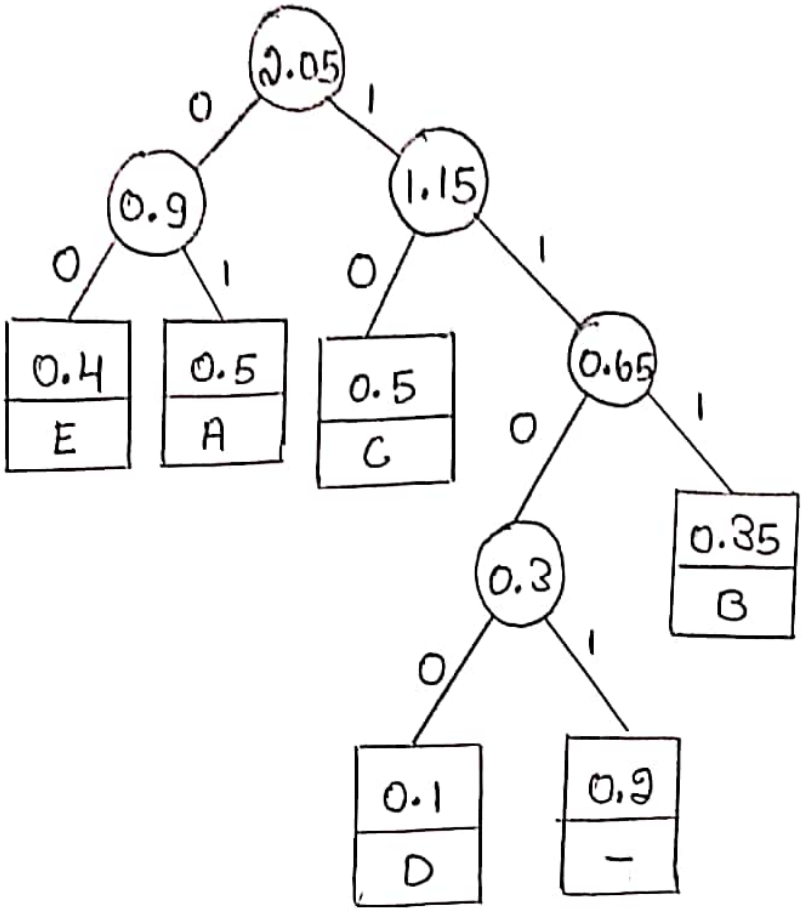












The resulting codewords are as follows

Symbol	A	B	C	D	E	–
probability	0.5	0.35	0.5	0.1	0.4	0.2
Codeword	01	111	10	1100	00	1101

hence

DAD_CBE is encoded as

1100011100110111000

06.

b) Explain transform and conquer technique sort the below list using Heap sort
3, 2, 4, 1, 6, 5 [8 marks]

⇒ Transform and conquer method works in two stages.

Stage 1:- (Transformation stage)
The problem's instance is modified more amenable to solution.

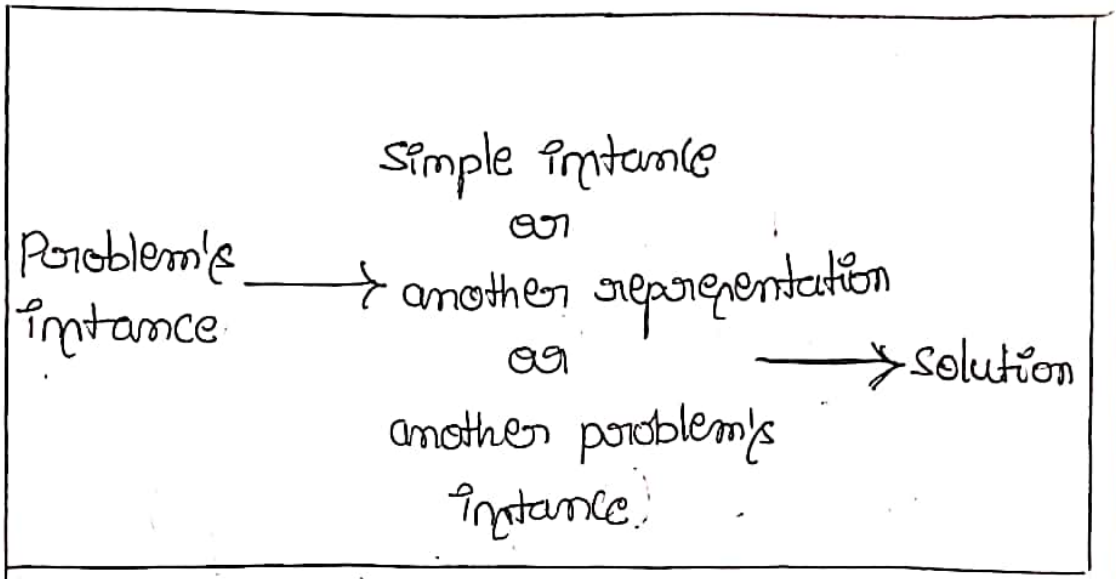
Stage 2:- (Conquering stage)
The transformed problem is solved.

There are three major variation of this idea that differ by what we transform a given instance to

i) Transformation to a simpler or more convenient instance of the same problem - we call it as Instance simplification.

ii) Transformation to a different representation of the same instance - we call it as Representation change.

iii) Transformation to an instance of a different problem for which an algorithm is already available - we call it Problem reduction.



Transform and conquer strategy

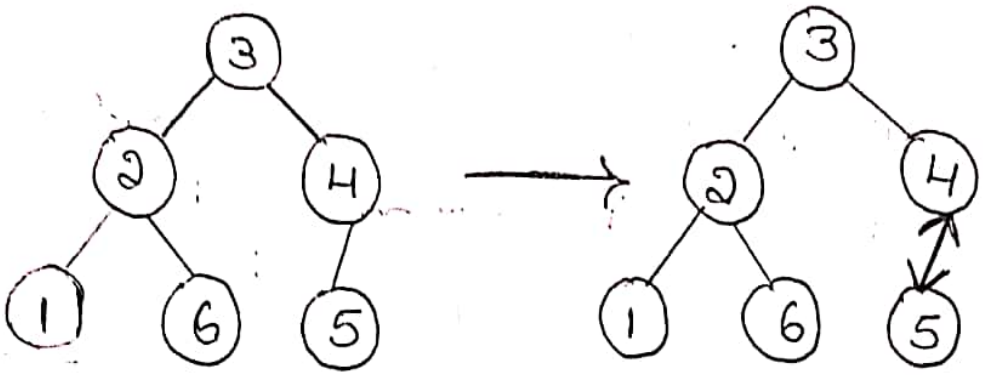
Consider the given elements

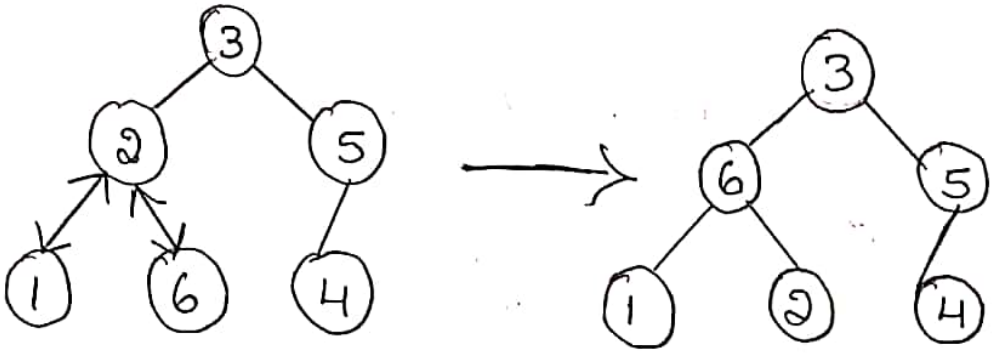
3, 2, 4, 1, 6, 5

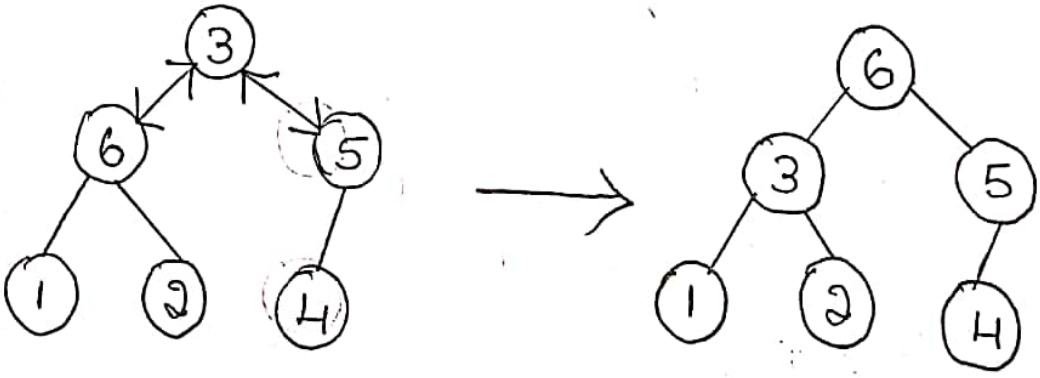
Heapsort - this is a two stage algorithm that work as follows

Stage 1 (Heap construction)

Construct a heap for a given array.







step 1 :- 3 2 [4] 1 6 5

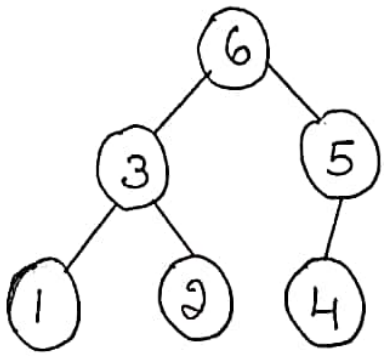
step 2 :- 3 [2] 5 1 6 4

step 3 :- [3] 6 5 1 2 4

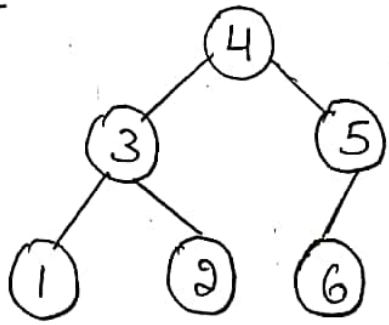
step 4 :- 6 3 5 1 2 4

Stage 2 (maximum deletion)

Apply the root-deletion operation n-1 times to the remaining heap

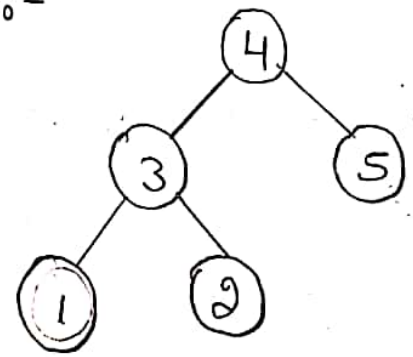


Step 1 :-



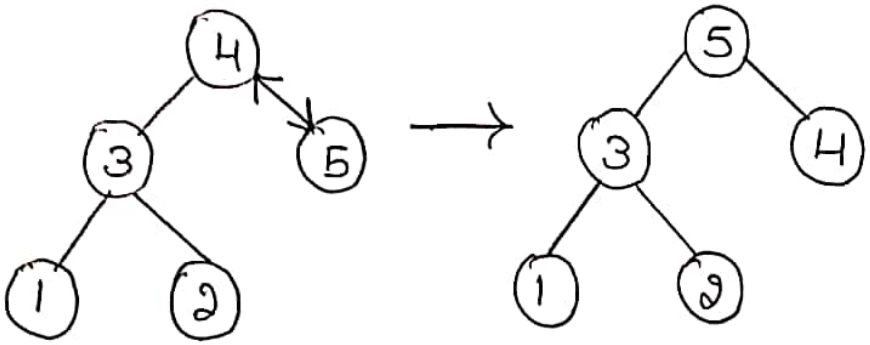
Replace the root node with the last leaf node.

Step 2 :-



Delete or remove the last leaf node i.e node 6 is deleted

Step 3: Compare the nodes & count the given elements.



therefore we have deleted root'n key from a heap. the key to be deleted is swapped with the last key.

After which the smaller tree is "heapified" by exchanging the new key in its root with the large key in its children until the parental dominance requirement is satisfied!

Fourth Semester B.E degree

Examination June/July 2018

Design and Analysis of Algorithms

05

a) Apply greedy method to obtain an optimal solution to the knapsack problem given $m = 60$, $(w_1, w_2, w_3, w_4, w_5) = (5, 10, 20, 30, 40)$, $(P_1, P_2, P_3, P_4, P_5) = (30, 20, 100, 90, 160)$ find the total profit earned. [04 marks]

⇒ given that

Knapsack capacity $m = 60$

Number of objects $n = 5$

Profit = $(P_1, P_2, P_3, P_4, P_5) = (30, 20, 100, 90, 160)$

weight = $(w_1, w_2, w_3, w_4, w_5) = (5, 10, 20, 30, 40)$

objects	1	2	3	4	5
Profit	30	20	100	90	160
weight	5	10	20	30	40
Profit/weight (P_i/w_i)	$\frac{30}{5}=6$	$\frac{20}{10}=2$	$\frac{100}{20}=5$	$\frac{90}{30}=3$	$\frac{160}{40}=4$
x	1	0	1	0	35/40 = 0.875

where x is a fraction of object taken into knapsack.

★ Greedy add objects to knapsack while the weight does not exceed capacity. objects 1 and 3 are included in the knapsack and this adds up to the weight 25.

★ The next object we consider is object 5 which has the weight of 40; but if we include object 5 entirely, the sum of weights will be $5+20+40 = 65 > 60$ it exceeds the capacity of the knapsack which is only 60, we have only 35 units left to the knapsack.

* The fraction 35/40 of object 5 is included in the knapsack to fill the capacity.

The total profit obtained =

$= 1 \times 30 + 1 \times 100 + 0.875 \times 160$

$= 30 + 100 + 140$

$= 270$ units

∴ the total profit earned = 270 units

05.

b) Explain Huffman algorithm with an example show the construction of Huffman tree and generate the Huffman code using this tree. [06 marks]

=> Huffman algorithm

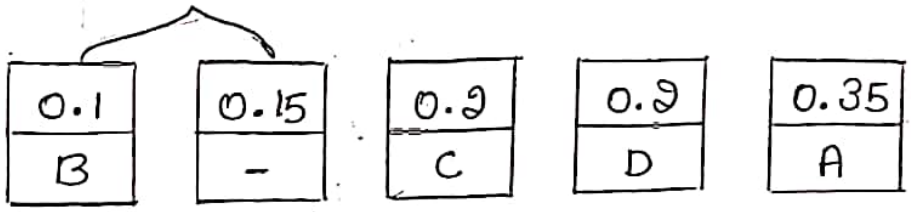
Step 01 - Initialize n one-node trees & and label them with the symbol of the alphabet given. Record the frequency of each symbol in its tree's root to indicate the tree's weight. (more generally, the weight of a tree will be equal to the sum of the frequencies in the tree's leaves).

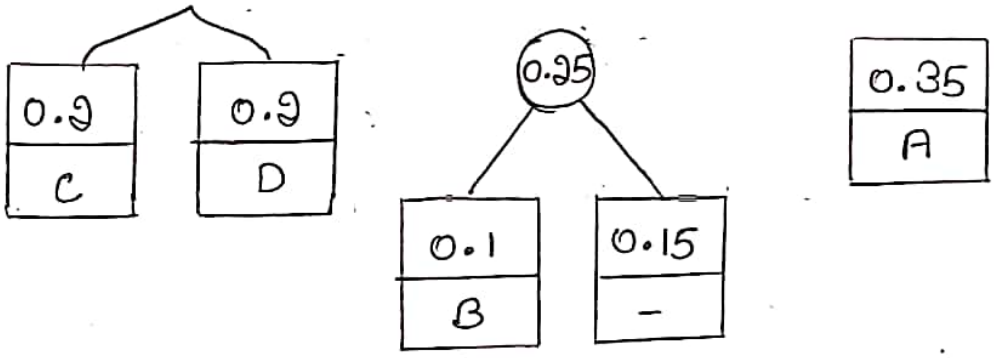
Step 02 - Repeat the following operation until a single tree is obtained. Find two trees with the smallest weight (ties can be broken arbitrarily) Make them the left and right subtree of a new tree & record the sum of their weights in the root of the new tree as its weight.

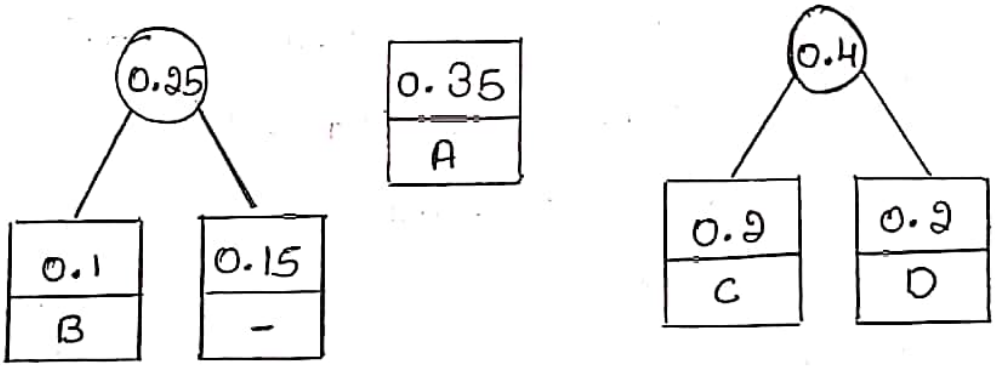
Example

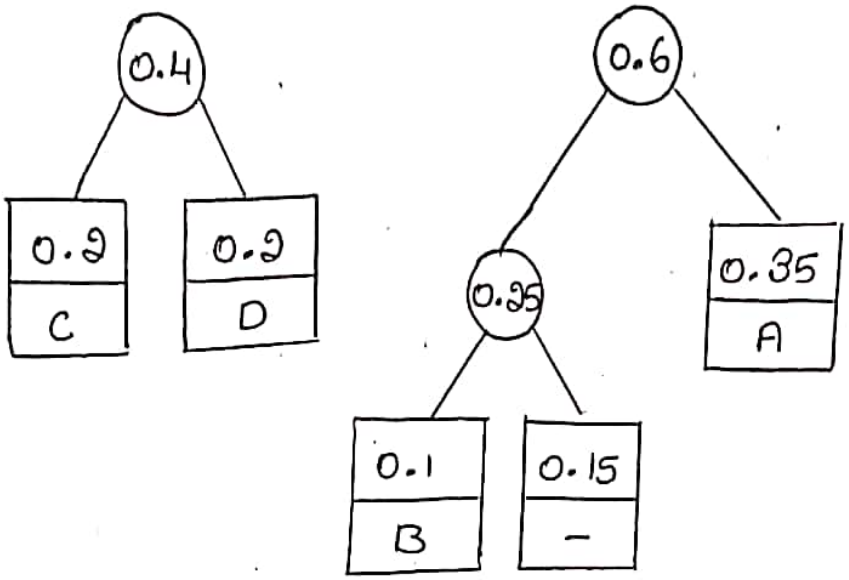
Consider the five symbol alphabet {A, B, C D, –} with the following occurrence frequencies in a text made up of these symbols:

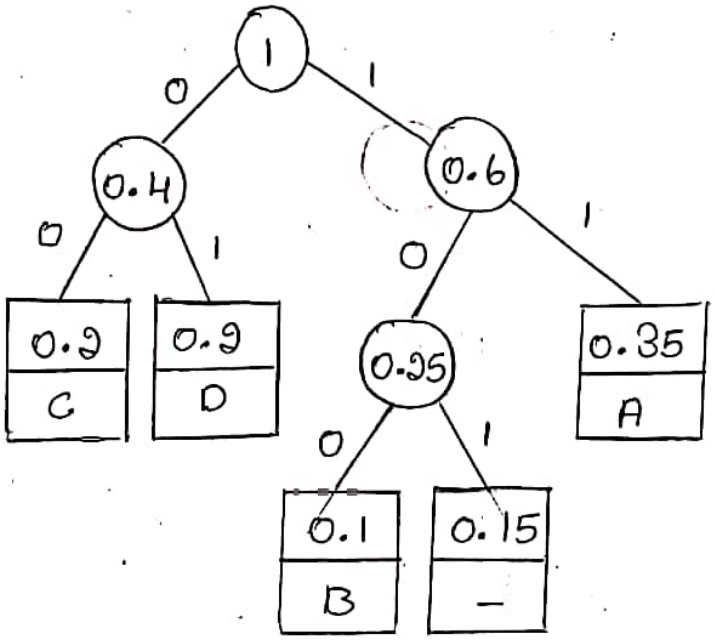
Symbol	A	B	C	D	–
frequency	0.35	0.1	0.2	0.2	0.15









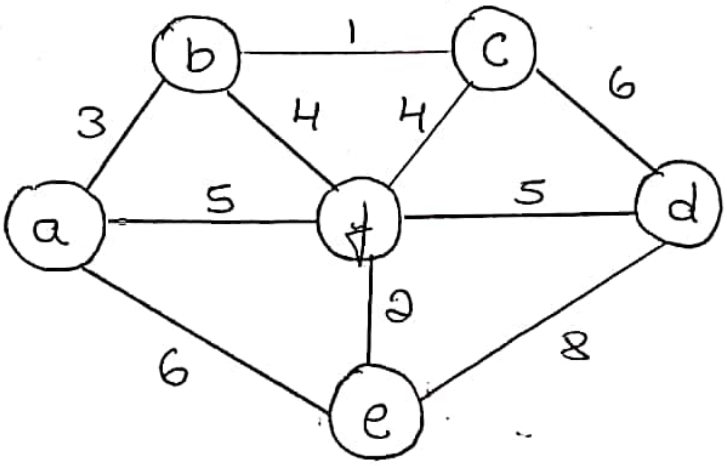


The resulting codeword are as follow

Symbol	A	B	C	D	–
Frequency	0.35	0.1	0.2	0.2	0.15
Codeword	11	100	00	01	101

05.

c) Apply prim's algorithm to obtain a minimum spanning tree for the given weighted connected graph. [06 marks]



=> Prim's algorithm

Algorithm prim(G)

// prim's algorithm for constructing a minimum spanning tree

// Input: A weighted connected graph $G = V, E$

// output: E_T, the set of edges comprising a minimum spanning tree of G.

// $V_T \leftarrow \{v_0\}$ the set of tree vertices can be initialized with any vertex.

$E_T \leftarrow \phi$

for $i \leftarrow 1$ to $|V| - 1$ do

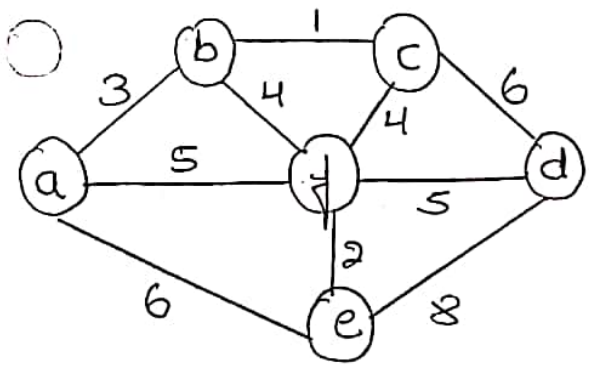
find a minimum-weight edge $e^* = (v^*, u^*)$ among all the edge (v, u) such that v is in V_T and u is in $V - V_T$

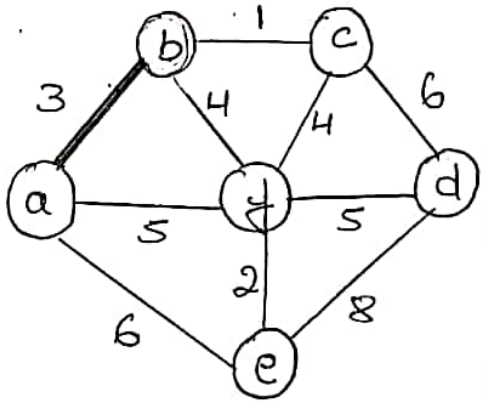
$V_T \leftarrow V_T \cup \{u^*\}$ $E_T \leftarrow E_T \cup \{e^*\}$

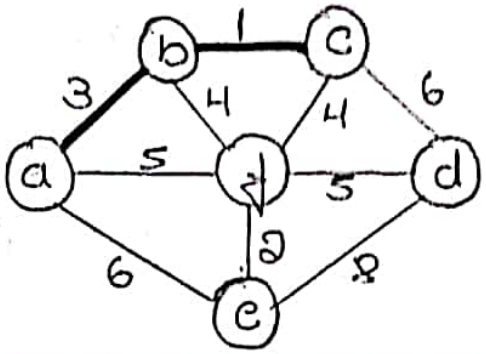
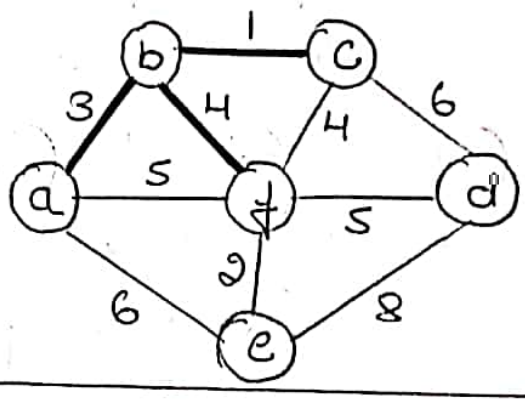
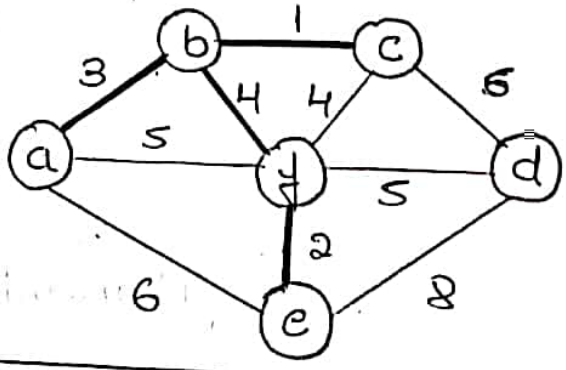
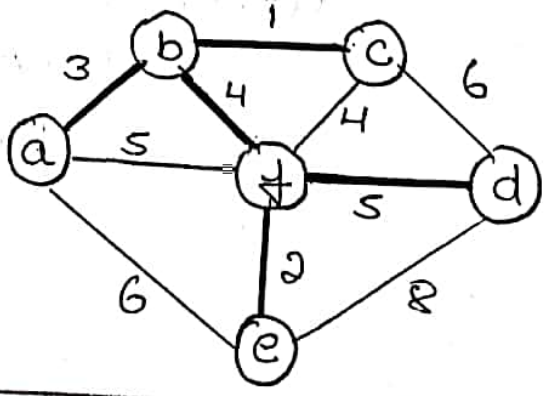
return E_T

By applying the above algorithm, let us find minimum spanning tree for given graph.

Consider the given graph.



Tree vertices	Remaining vertices	Illustration
a(-,-)	b(a,3) c(-,∞) d(-,∞) e(a,6) f(a,5)	

Tree vertex	Remaining vertex	Illustration
b(a,3)	c(b,1) d(-,∞) e(a,6) f(b,4)	
c(b,1)	d(c,6) e(a,6) f(b,4)	
f(b,4)	d(f,5) e(f,2)	
e(f,2)	d(f,5)	
d(f,5)		

06.

a) Explain the bottom up heap construction algorithm with an example. give the worst case efficiency of this algorithm.
[08 marks]

⇒ Bottom-up heap construction algorithm initialize the essentially complete binary tree with n nodes by placing keys in the order given and "heapifies" the tree as follow:-

* Starting with the last parental node, the algorithm checks whether the parental dominance holds for the key in this node. if it does not, the algorithm exchanges the nodes key K with the larger key of its children & checks whether the parental dominance holds for K in its new position. this process continues until the parental dominance for K is satisfied.

* After completing the "heapification" of the subtree rooted at the current parental

node, the algorithm proceeds to do the same for the node's immediate predecessor.

* The algorithm stops after this is done for the root of the tree.

Algorithm

Algorithm HeapBottomUp(H[1....n])

// Construct a heap from elements of a given array by the bottom up algorithm

// Input: An array H[1.....n] of orderable items

// output: A heap H[1....n]

for i ← n/2 down to 1 do

    k ← i; v ← H[k]

    heap ← false

    while not heap and 2*k ≤ n do

        j ← 2*k

        if j < n // there are two children

            if H[j] < H[j+1]

                j ← j+1

        if v ≥ H[j]

            heap ← true

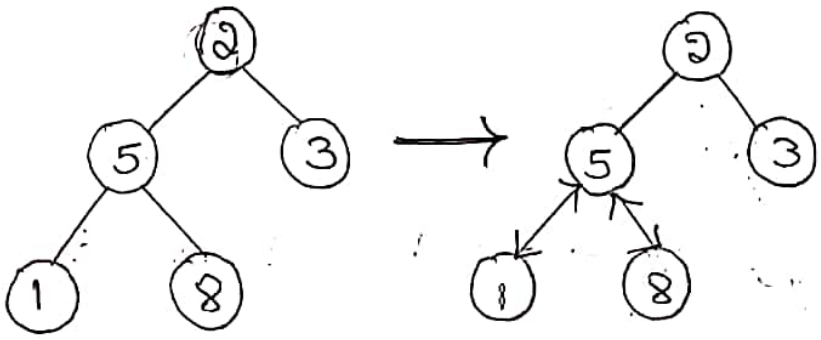
else H[k] ← H[j];

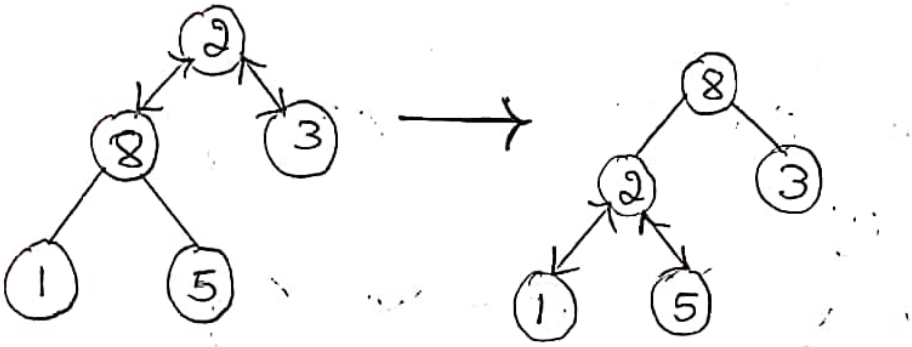
k ← j

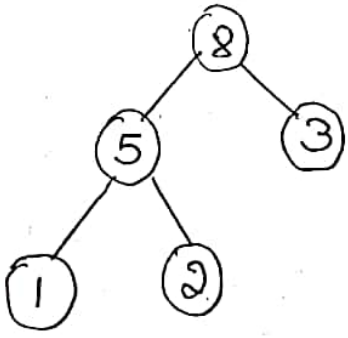
H[k] ← v

example

Apply Bottom up construction of a heap for the list 2, 5, 3, 1, 8

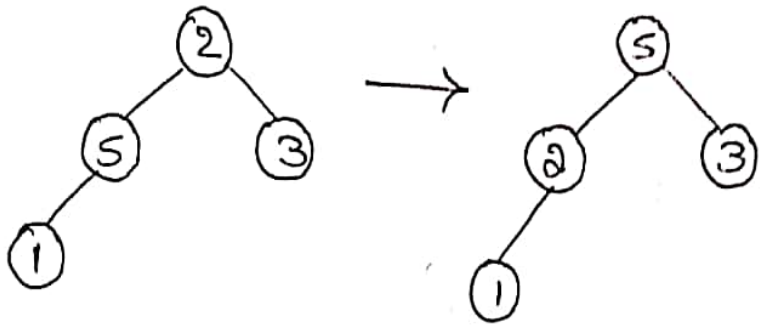




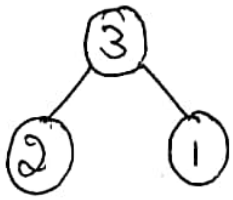


proper Heap

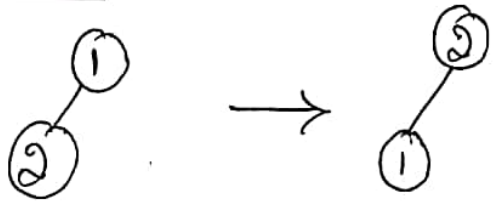
Iteration #1 :- Remove key 2



Iteration #2 :- Remove key 5



Iteration #3 :- Remove key 3



Iteration #4 :- Remove key 2

Iteration #5 :- Remove key 1

Final sorted array is 1, 2, 3, 5, 8

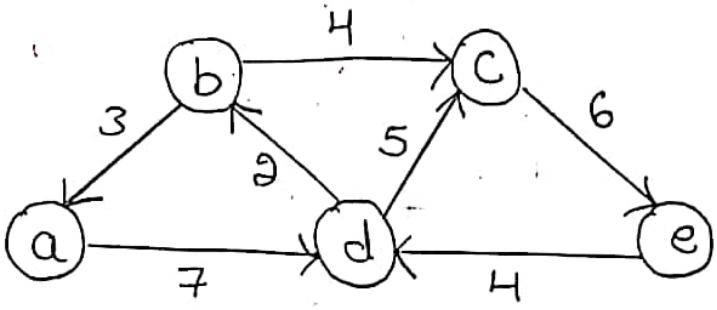
- Worst case efficiency of this algorithm is given by

$$C_{worst}(n) = \sum_{i=0}^{h-1} \sum_{\text{level } i \text{ keys}} 2(h-i) = \sum_{i=0}^{h-1} 2(h-i)2^i$$

$$= 2(n - \log_2(n+1)).$$

6)

b) Apply single source shortest path Problem assuming vertex 'a' as source [08 marks]



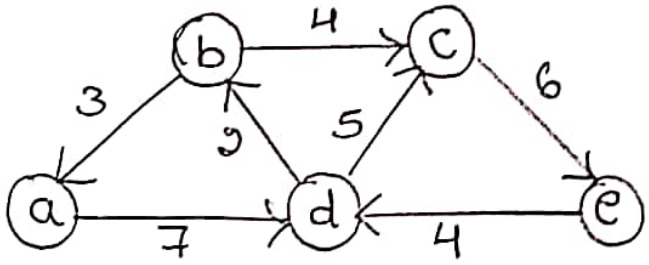
⇒ Single - source shortest path problem is also known as Dijkstra's algorithm for a given vertex called the source in a weighted connected graph, it finds shortest paths to all its other vertices.

Step 01 :- The following two sets are considered

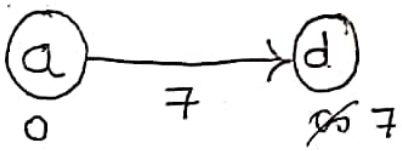
- Unvisited set = {a, b, c, d, e}
- Visited set = { }

Step 02 :- The two variables π and d are created for each vertex and initialized as

- $\pi[a] = \pi[b] = \pi[c] = \pi[d] = \pi[e] = NIL$
- $d[a] = 0$
- $d[b] = d[c] = d[d] = d[e] = \infty$



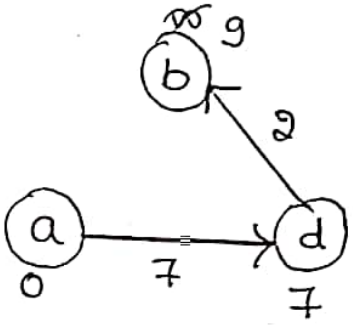
Step 3: Vertex a is chosen, because it is source vertex.



$d[a] + 7 = 0 + 7 = 7 < \infty$

$\therefore d[d] = 7$ and $\pi[d] = a$

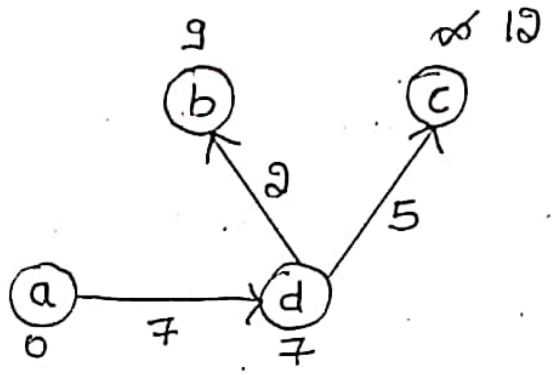
Step 4:



$d[d] + 2 = 7 + 2 = 9 < \infty$

$\therefore d[b] = 9$ and $\pi[b] = d$

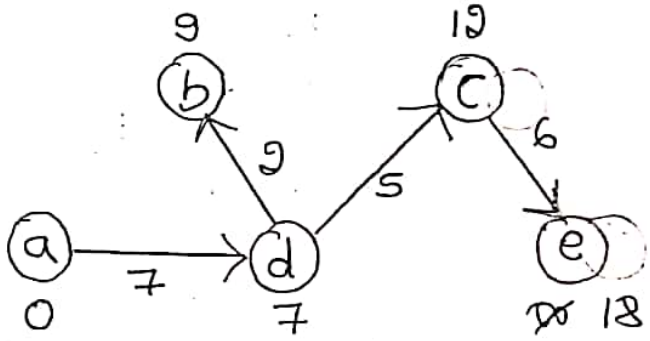
Step 05:



$d[b] + 5 = 7 + 5 = 12 < \infty$

$\therefore d[c] = 12$ and $\pi[c] = d$

Step 06:-



$d[c] + 6 = 12 + 6 = 18 < \infty$

$\therefore d[e] = 18$ and $\pi[e] = c$

By considering all the steps now the updated sets are as follow

- unvisited set = { }
- visited set = {a, b, c, d, e}

Now, all vertices of the graph are processed.

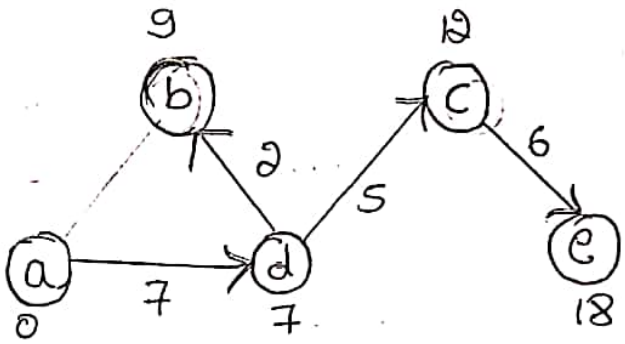
our final shortest path tree is as shown below

from a to d : length 0+7 = 7

from d to b : length 7+2 = 9

from d to c : length 7+5 = 12

from d to e : length 12+6 = 18



It represents the shortest path from source vertex 'a' to all other remaining vertices.

The order in which all the vertices are processed is :- a, d, b, c, e.

Fourth Semester B.E Degree Examination. Dec 2017/Jan 2018

Design and Analysis of Algorithms

05

a) Solve the greedy knapsack problem where $m=10$, $n=4$, $P = (40, 42, 25, 12)$ and $w = (4, 7, 5, 3)$. [06 marks]

=> given that

Knapsack Capacity $m=10$

Number of objects $n=4$

profit = $(P_1, P_2, P_3, P_4) = (40, 42, 25, 12)$

weight = $(w_1, w_2, w_3, w_4) = (4, 7, 5, 3)$

object	1	2	3	4
Profit	40	42	25	12
weight	4	7	5	3
Profit/weight (P_i/w_i)	$\frac{40}{4} = 10$	$\frac{42}{7} = 6$	$\frac{25}{5} = 3$	$\frac{12}{3} = 4$
x	1	$6/7 = 0.8571$	0	0

where x is a fraction of a object taken into knapsack

- object 1 is included in knapsack which has capacity of 10,

remaining knapsack capacity = 10 - weight of object 1

$$10 - 4 = 6$$

- the next object we considered is object 2 which has weight of 7, but if we include object 2 completely the sum of weights will be $4 + 7 = 11 > 10$, it exceeds capacity of knapsack which is only 10.
- we have only 6 weight units left in knapsack, the fraction 6/7 of object 2 is included to fill its capacity

∴ the profit obtained is

$= 1 \times 40 + 0.8571 \times 49$

$= 75.9989$

∴ the total profit earned = 76 units

5)

b) What is job sequencing with deadlines problem? let n = 5, profits [10, 3, 33, 11, 40] and deadlines [3, 1, 1, 2, 2] respectively find the optimal solution using greedy algorithm. [05 marks]

=> Job sequencing with deadline problem

Defination :- given a set of n jobs and n integer is associated with job i, deadline $d_i >= 0$ and profit $P_i >= 0$. It is required to find the jobs such that all the chosen jobs should be completed within their deadline & profit earned should be maximum with the following constraints

- only one machine is available for processing jobs.
- only one job must be processed at any point of time.

given that

number of jobs $n = 5$

profits = [10, 3, 33, 11, 40]

Deadlines = [3, 1, 1, 2, 2]

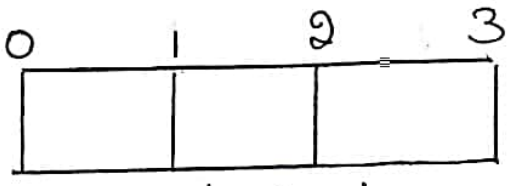
Jobs	J_1	J_2	J_3	J_4	J_5
Deadlines	3	1	1	2	2
Profits	10	3	33	11	40

Step 01: Sort all the given jobs in decreasing order of their profits

Jobs	J_5	J_3	J_4	J_1	J_2
Deadlines	2	1	2	3	1
Profits	40	33	11	10	3

Step 02:

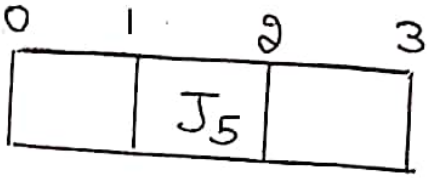
- value of maximum deadline = 3
- So, draw a gantt chart with maximum time on gantt chart = 3 units



gantt chart

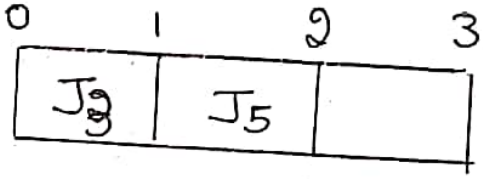
Step 03 :-

- We take job J_5
- Since its deadline is 2, so we place it in the first empty cell before deadline 2 as



Step 04:

- We take Job J_3
- Since its deadline is 1, so we place it in the first empty cell before deadline 1 as



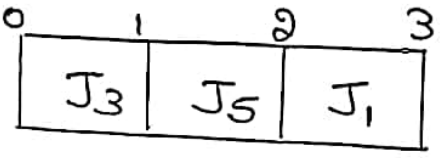
Step 05:

Now,

- We take Job J_4 whose deadline is 2 but all the slots before deadline 2 are already occupied
- Thus, J_4 can not be completed

Step 06:-

- we take Job J_1
- Since the deadline is 3, so we place it in the first empty cell before deadline 3 as



then

- the only job left is job J_2 whose deadline is 1 but all the slot before deadline 1 are already occupied.
- thus, Job J_2 can not be completed.

The optimal schedule is J_3, J_5, J_1 this is required order in which the jobs must be completed in order to obtain the maximum profit.

Maximum earned profit = sum of profit of all jobs in optimal schedule

maximum profit = profit of J_3 + profit of J_5 + profit of J_1

$= 33 + 40 + 10 = 83$ units

∴ maximum earned profit = 83 units

05.

c) Define minimum cost spanning tree (MST) write prim's algorithm to construct minimum cost spanning tree. [05 marks]

⇒ A spanning tree of a connected graph is its connected acyclic subgraph that contains all the vertices of the graph. A minimum spanning tree of a weighted connected graph is its spanning tree of the smallest weight, where the weight of a tree is defined as the sum of the weights on all its edges.

Prim's Algorithm

Algorithm prim(G)

// prim's algorithm for constructing a minimum spanning tree

// Input: A weighted connected graph $G = V, E$

// output: E_T, the set of edges composing a minimum spanning tree of G

// $V_T \leftarrow \{v_0\}$ the set of tree vertices can be initialized with any vertex.

$E_T \leftarrow \phi$

for $i \leftarrow 1$ to $|V| - 1$ do

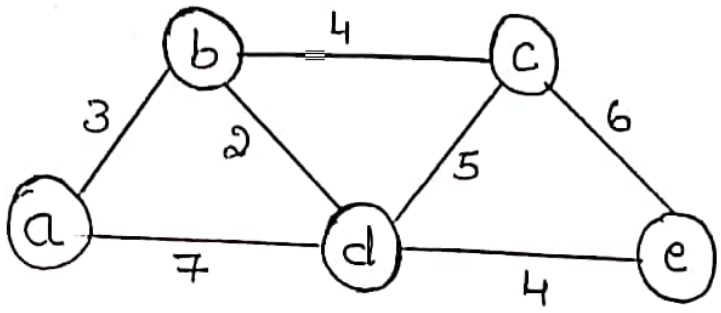
find a minimum-weight edge $e^* = (v^*, u^*)$ among all the edges (v, u) such that v is in V_T and u in $V - V_T$

$V_T \leftarrow V_T \cup \{u^*\}$ $E_T \leftarrow E_T \cup \{e^*\}$

return E_T

06.

a) Design Dijkstra's Algorithm and apply the same to find the single source shortest path your graph taking vertex 'a' as source. [08 marks]

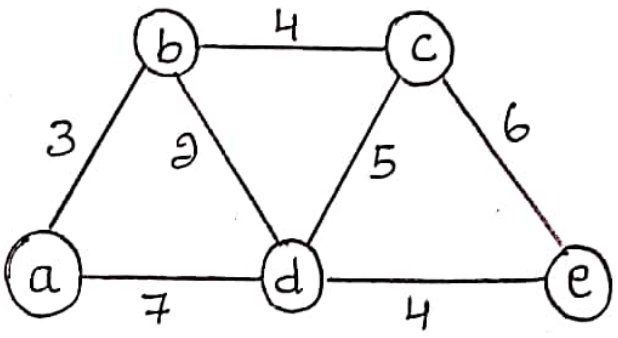


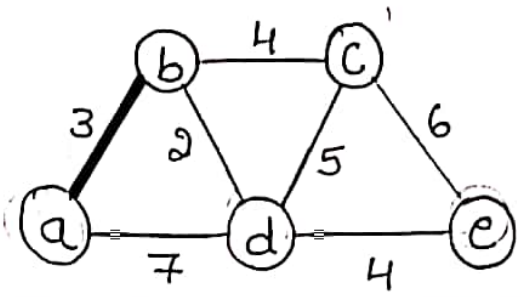
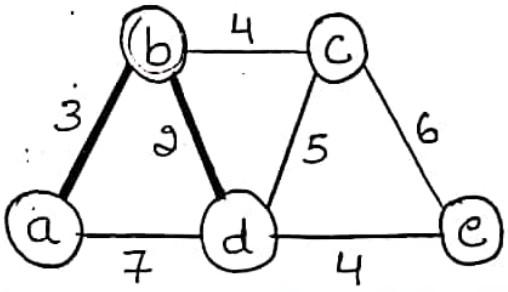
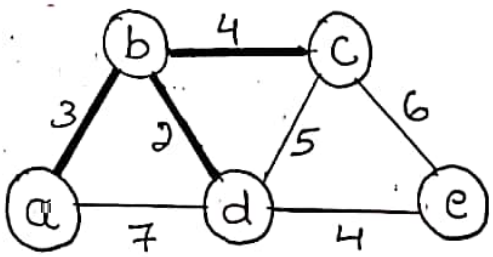
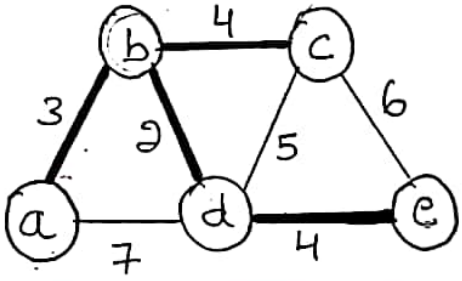
=> Dijkstra's Algorithm

```
Algorithm ShortestPaths(v, cost, dist, n)
// dist[j], 1<=j<=n is set to the length of the
 shortest
// path from vertex v to vertex j in a digraph
 g with n
// vertices dist[v] is set to zero. g is repre-
-esented by its
// cost adjacency matrix cost[1:n, 1:n].
{
   for i := 1 to n do
     {
       // initialize s
       s[i] := false ; dist[i] := cost[v,i];
     }
```

```
S[V] := true; dist[V] := 0.0; //put V in S
for num := 2 to n-1 do
{
  //determine n-1 path from V
  Choose u from among those vertices
  not in S such that dist[u] is
  minimum;
  S[u] := true; //put u in S
  for(each w adjacent to u with
              S[w] = false) do
   //update distances
  if(dist[w] > dist[u] + cost[u,w]) then
    dist[w] := dist[u] + cost[u,w];
  }
}
```

Consider the given graph.



Tree Vertices	Remaining Vertices	Illustration
a(-,0)	b(a,3) c(-,∞) d(a,7) e(-,∞)	
b(a,3)	c(b,3+4), d(b,3+2), e(-,∞)	
d(b,5)	c(b,7), e(d,5+4)	
c(b,7)	e(d,9)	
e(d,9)		

06.

b) Construct a Huffman code for the following data. [4 marks]

Character	A	B	C	D	-
probability	0.4	0.1	0.2	0.15	0.15

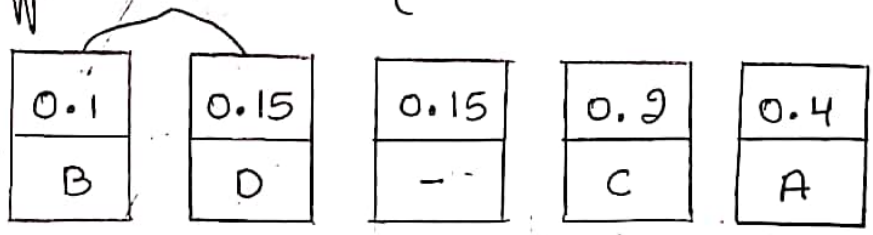
Encode the text ABACABAD and decode text 100010111001010, using the above code.

⇒ given that

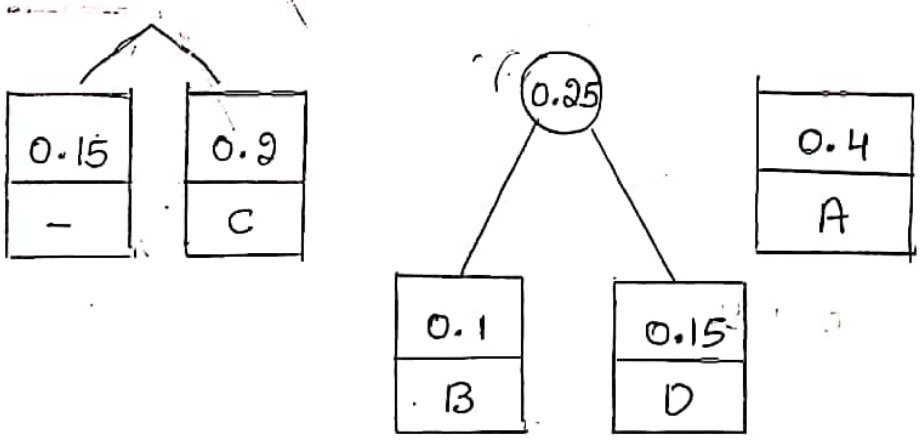
character	A	B	C	D	-
probability	0.4	0.1	0.2	0.15	0.15

Huffman tree construction

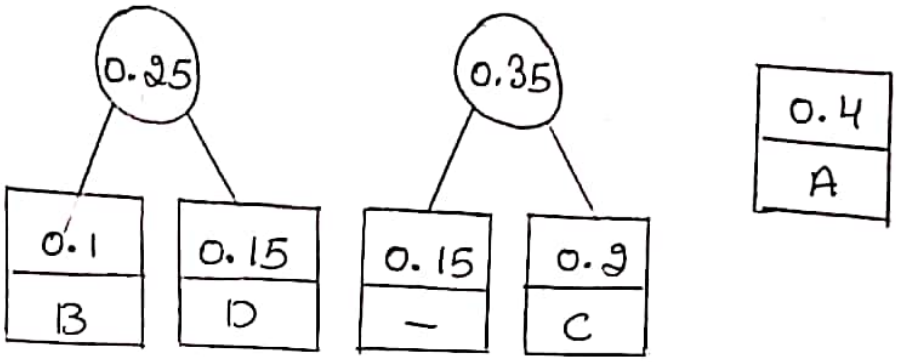
⇒



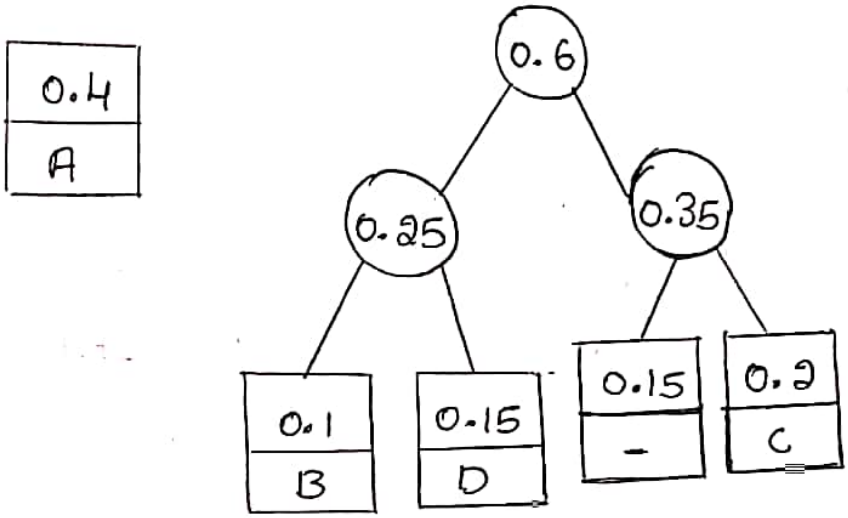
⇒



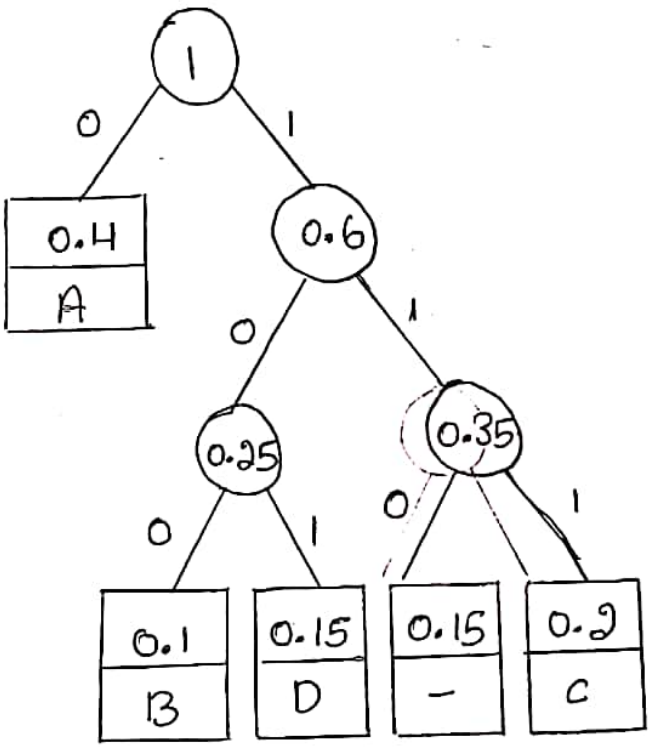
⇒



⇒



⇒



The resulting codewords are as follows

Symbol	A	B	C	D	-
Probability	0.4	0.1	0.2	0.15	0.15
Codeword	0	100	11	101	110

hence

ABACABAD is encoded as

0100011101000101

100010111001010 is decoded as

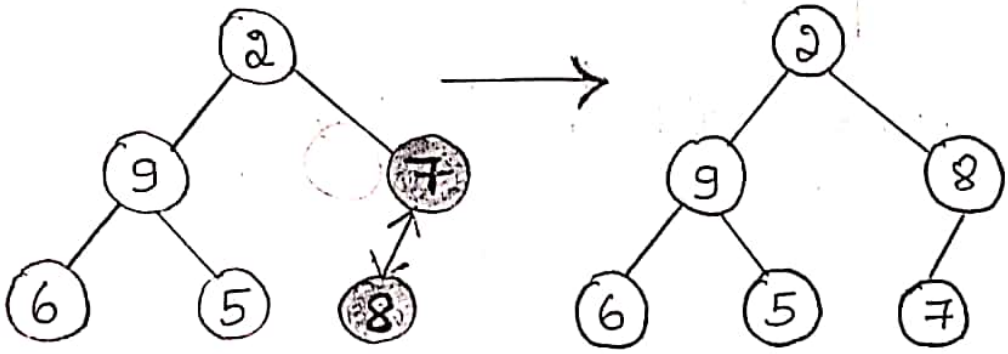
BAD_ADA

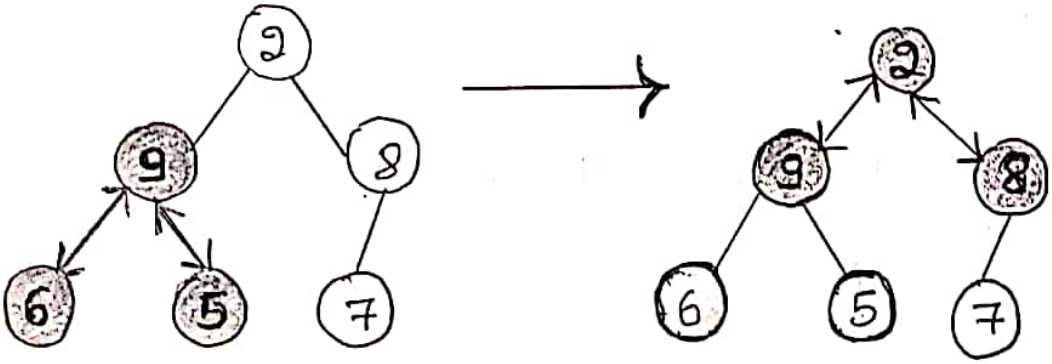
06.

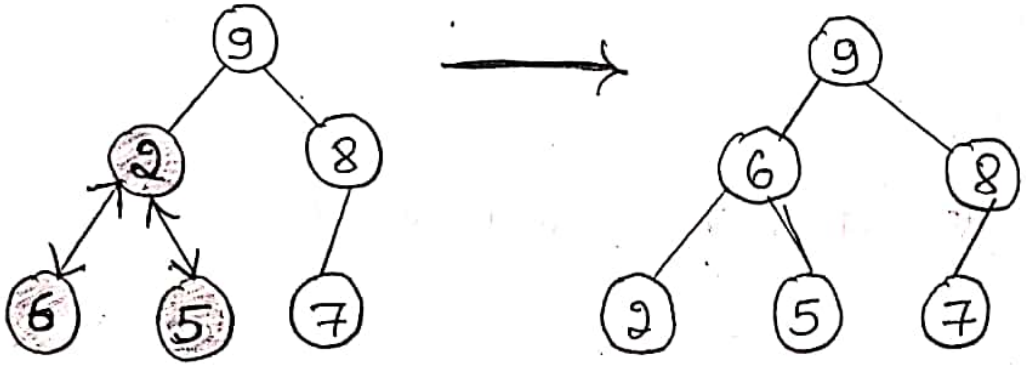
c) Construct the heap for the list 2, 9, 7, 6, 5, 8 by the bottom-up algorithm

⇒ given elements are 2, 9, 7, 6, 5, 8 [04 marks]

heap construction







Proper Heap

step 1: 2 9 [7] 6 5 8

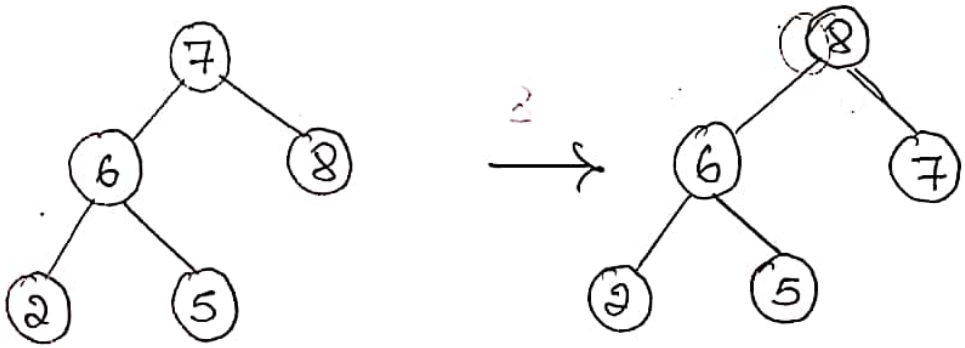
step 2: 2 [9] 8 6 5 7

step 3: [2] 9 8 6 5 7

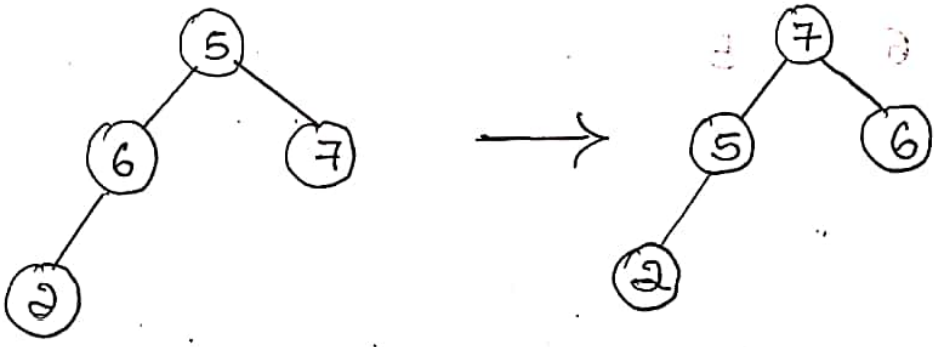
step 4: 9 [2] 8 6 5 7

step 5: 9 6 8 2 5 7

Iteration #1 : Remove key 9

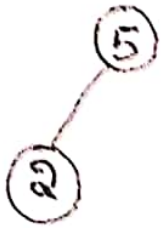


Iteration #2 : Remove key 8



Iteration #3 : Remove key 7

Iteration #4: Remove key 6



Iteration #5: Remove key 5

Iteration #6: Remove key 9

Final sorted array is 2, 5, 6, 7, 8, 9

Fourth Semester B.E. Degree Examination, June/July 2017

Design and Analysis of Algorithms

Module - 3

(a) 5) Explain Greedy criterion. Write a Prim's algorithm to find minimum cost spanning tree. [8 marks]

→ The greedy method is the straight forward design technique applicable to variety of applications.

The greedy approach suggests constructing a solution through a sequence of steps, on each step the choice made must be

* feasible i.e it has to satisfy the problem's constraints.
* locally optimal i.e it has to be the best local choice among all feasible choices available on that step.
* irrevocable i.e once made, it cannot be changed on subsequent steps of the algorithm.

Prim's Algorithm :

* It is a famous greedy Algorithm.
* It is used for finding the minimum spanning tree of a given graph.

Algorithm Prim (G)

// Prim's algorithm for constructing a minimum spanning tree.
// Input : A weighted connected graph $G = V, E$.
// Output : E_T, the set of edges composing a minimum spanning tree of G.
// $V_T \leftarrow \{v_0\}$ the set of tree vertices can be initialized with any vertex.

$E_T \leftarrow \phi$

for $i \leftarrow 1$ to $|V| - 1$ do

find a minimum-weight edge $e^* = \{v^*, u^*\}$ among all the edges $\{v, u\}$ such that

v is in V_T and u is in $V - V_T$

$V_T \leftarrow V_T \cup \{u^*\}$ $\quad E_T \leftarrow E_T \cup \{e^*\}$

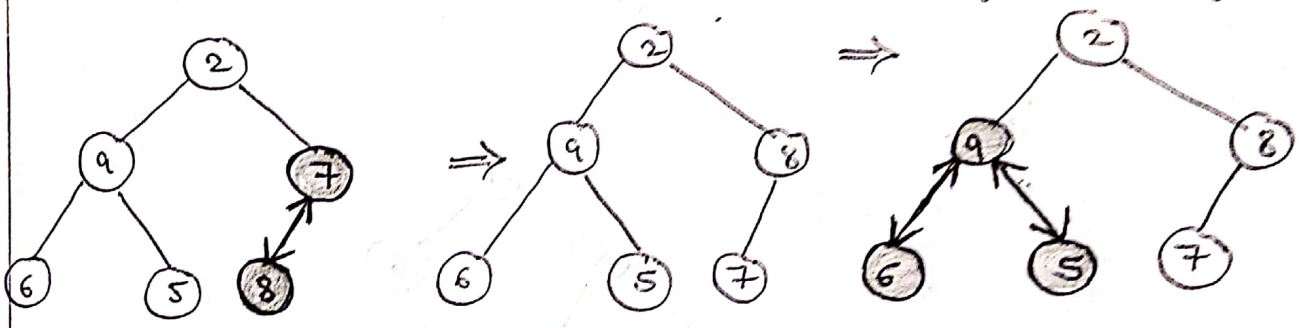
return E_T.

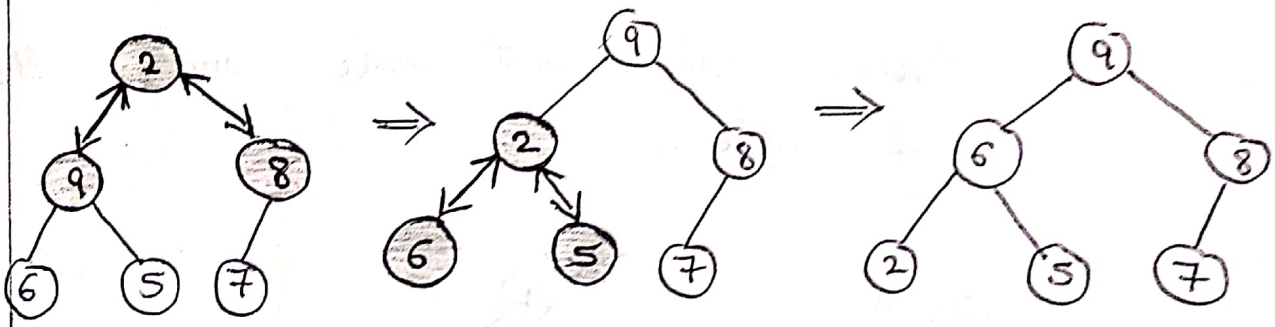
5) b) Sort the given list of numbers using heap sort : 2, 9, 7, 6, 5, 8 [8 marks]

→ Heapsort - This is a two-stage algorithm that work as follows.

Stage 1 (Heap construction) :

Construct a heap for a given array.





Step 1 : 2 9 **7** 6 5 8

Step 2 : 2 **9** 8 6 5 7

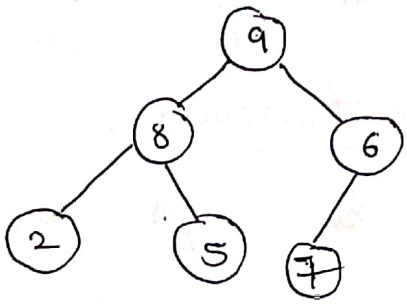
Step 3 : **2** 9 8 6 5 7

Step 4 : 9 **2** 8 6 5 7

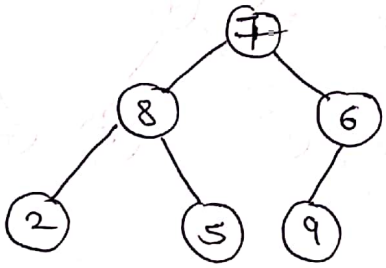
Step 5 : 9 6 8 2 5 7

Stage 2 (maximum deletions) :

Apply the root - deletion operation n-1 times to the remaining heap.

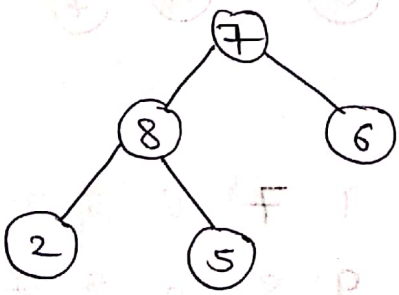


Step 1 :



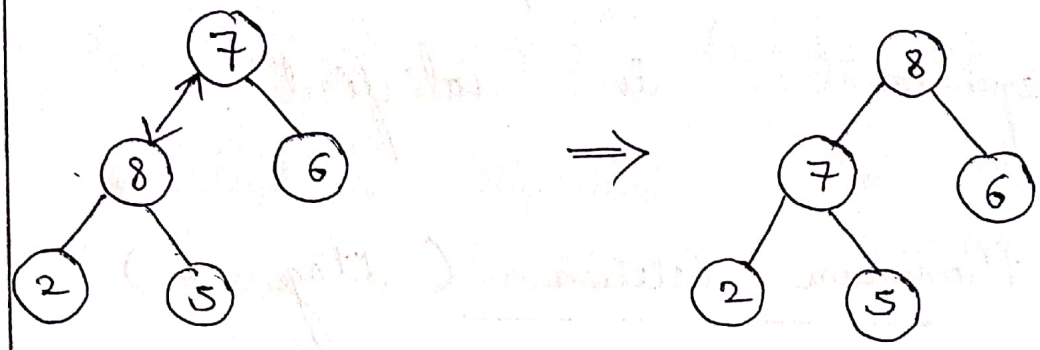
Replace the root node with the last leaf node.

Step 2 :

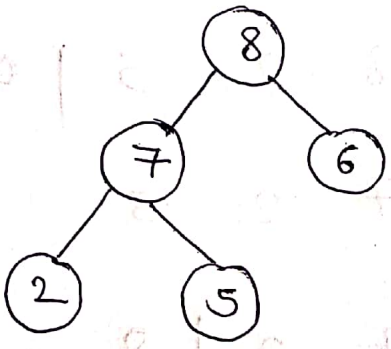


Delete or remove the last leaf node, i.e node 9 is deleted.

Step 3 : Compare the nodes and heapify the elements.



Step 4 :



Therefore we have deleted root's key from a heap. The key to be deleted is swapped with the last key. After which the smaller tree is "heapified" by exchanging the new key in its root

with larger key in its children until the parental dominance requirement is satisfied.

Maximum deletion (Stage - 2)

[9] 6 8 2 5 7

7 6 8 2 5 | 9

[8] 6 7 2 5

5 6 7 2 | 8

[7] 6 5 2

2 6 5 | 7

[6] 2 5

5 2 | 6

[5] 2

2 | 5

[2]

(or)

6 (a) Write an algorithm to find single source shortest path. [8 marks]

→ Dijkstra's Algorithm is used for finding the single source shortest path problem.

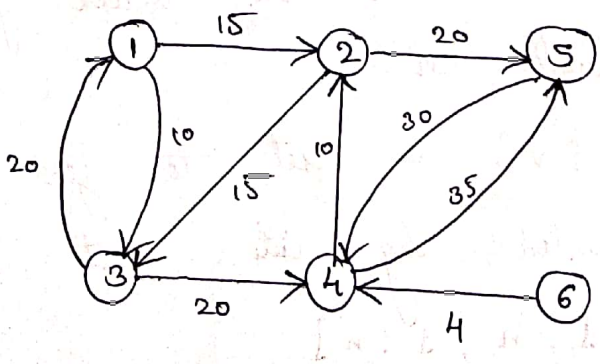
* Dijkstra Algorithm is a very famous greedy algorithm.
* It computes the shortest path from one particular source node to all other remaining nodes of the graph.

Algorithm :

```
Algorithm Dijkstra (G, S)
// dist[j], 1 <= j <= n, is set to the length of the shortest
// path from vertex v to vertex j in a digraph G with n.
// vertices, dist[v] is set to zero.
// G is represented by its cost adjacency matrix cost[1:n, 1:n].
{
  for i := 1 to n do
  {
    // Initialize S.
    S[i] := false; dist[i] := cost[v, i];
  }
```

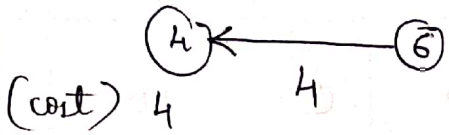
```
S[v] := true ; dist[v] := 0.0 ; // put v in S.
for num := 2 to n-1 do
{
    // Determine n-1 paths from v.
    Choose u from among those vertices not
    in S such that dist[u] is minimum;
    S[u] := true ; // put u in S.
    for (each w adjacent to u with S[w] = false) do
    // Update distances..
    if (dist[w] > dist[u] + cost[u,w])) then
    dist[w] := dist[u] + cost[u,w];
}
}
```

For example,



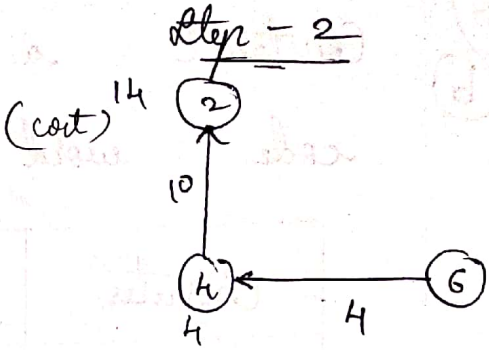
Let us solve this above example by considering 6 as the source vertex.

Step - 1

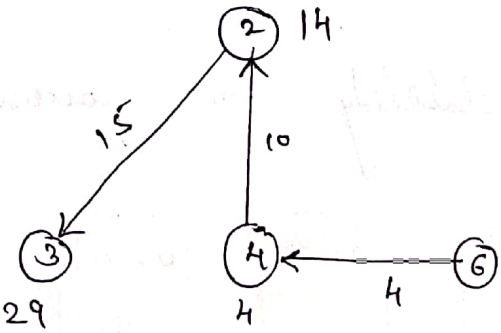


⟹

Step - 2

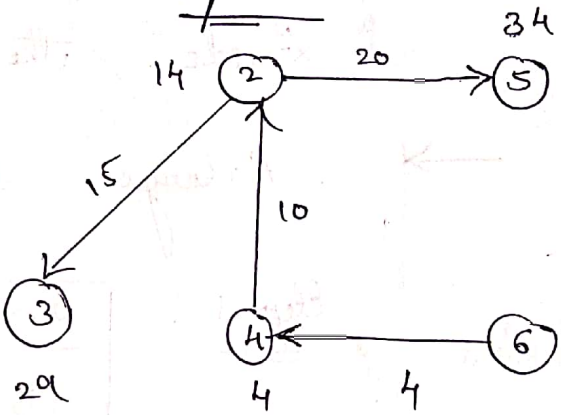


Step - 3

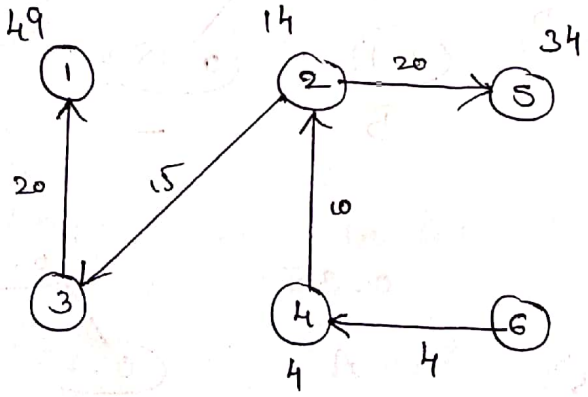


⟹

Step - 4



Step - 5



∴ This is the single source shortest path by taking 6 as source vertex.

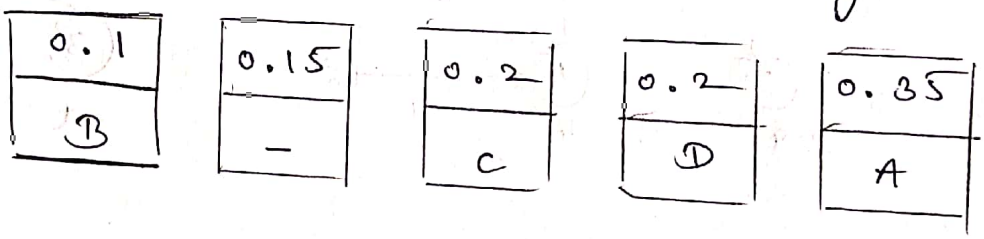
6 (b) Construct a Huffman tree and resulting code word for the following.

Character	A	B	C	D	-
Probability	0.35	0.1	0.2	0.2	0.15

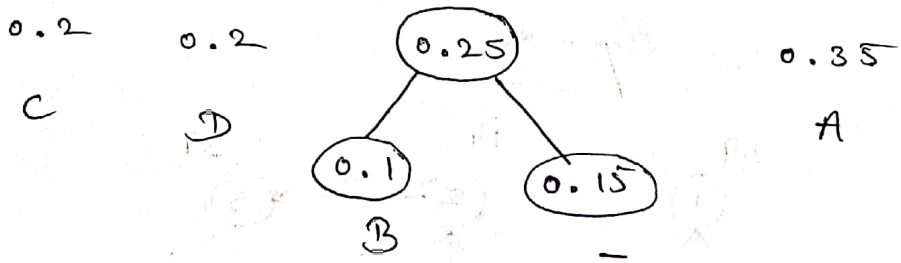
Encode the words DAD and ADD. [8 marks]

→ Arrange the probability in ascending order.

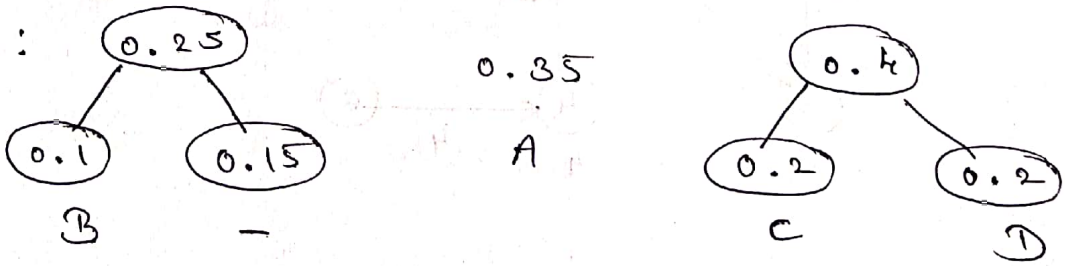
Step 1:



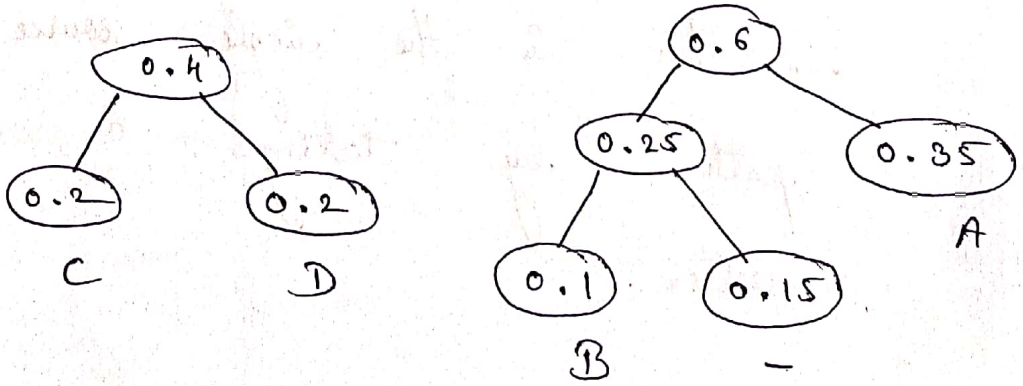
Step 2:



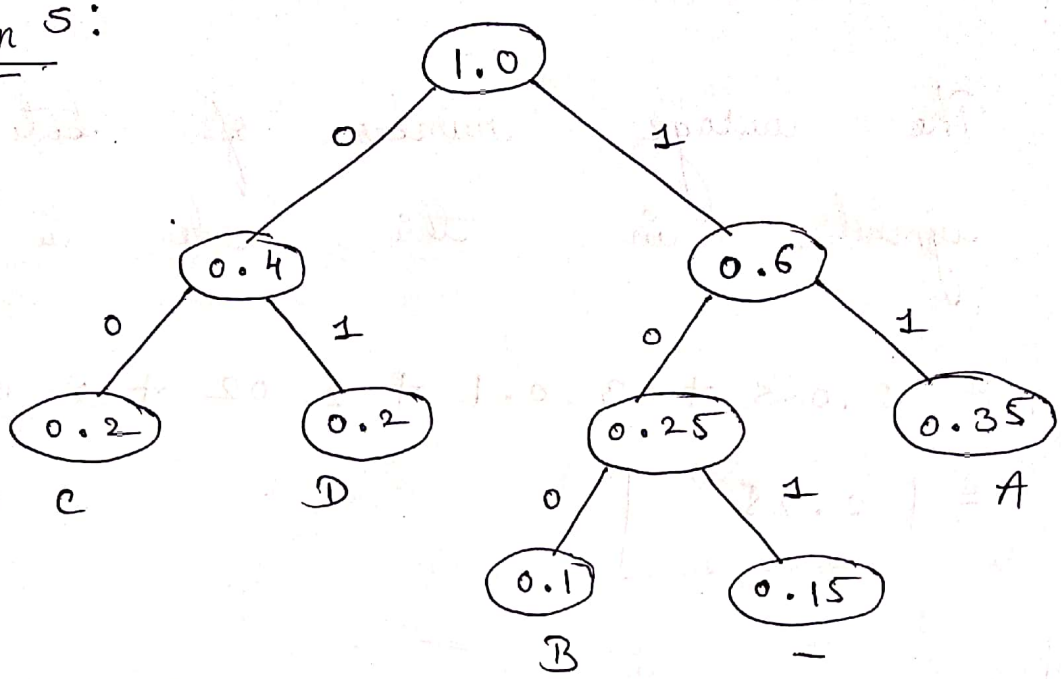
Step 3:



Step 4:



Step 5:



The resulting codewords are as follows,

Symbol	A	B	C	D	—
frequency	0.35	0.1	0.2	0.2	0.15
Codeword	11	100	00	01	101

* DAD is encoded as 011101
* ADD is encoded as 110101.

DAD and ADD

01 11 01 and 11 01 01

The average number of bits per symbol in this code is,

$= 2.035 + 3.0.1 + 2.02 + 3.0.15$

$= \boxed{2.25}$

Fourth Semester B.E. Degree

Examination, June/July 2019

Design and Analysis of Algorithms

Module - 3

5 (a) Write an algorithm to solve knapsack problem using Greedy technique. Find the optimal solution to the knapsack instance $n = 7$, $m = 15$. [10 marks]

$(P_1, P_2, \ldots P_7) = (10, 5, 15, 7, 6, 18, 3)$

$(w_1, w_2, \ldots w_7) = (2, 3, 5, 7, 1, 4, 1)$

→ Algorithm:

```
void GreedyKnapSack (int m, int n, float w[],
                     float p[])
// Input : p[1:n] and w[1:n] -
           contains profits and weights.
// p[i]/w[i] >= p[i+1]/w[i+1] -
   decreasing order of ratio.
// m is knapSack capacity.
// x[1:n] is solution vector.
```

```
{
for ( i = 1 ; i <= n ; i++)
    x[i] = 0
for ( i = 1 ; i <= n ; i++)
{
    if ( w[i] > m)
        break ;
    else
    {
        x [i] = 1 ;
        tp = tp + p[i] ;
        m = (int)(m - w[i])
    }
}
if (i < n)
    x[i] = m/ w[i] ;
tp = tp + (x[i] * p[i]);
// Display the total profit.
```

By considering the above algorithm, we can apply the algorithm to find optimal solution.

By given data,

* Knapsack capacity, $m = 15$
* number of objects, $n = 7$
* Profit, P

$(P_1, P_2, \ldots P_7) = (10, 5, 15, 7, 6, 18, 3)$

* Weight, w

$(w_1, w_2, \ldots w_7) = (2, 3, 5, 7, 1, 4, 1)$

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

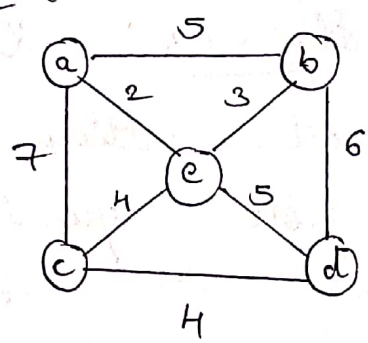
Objects	1	2	3	4	5	6	7
Profit	10	5	15	7	6	18	3
Weight	2	3	5	7	1	4	1
Profit / Weight (P_i/w_i)	$\frac{10}{2}$ = 5	$\frac{5}{3}$ = 1.66	$\frac{15}{5}$ = 3	$\frac{7}{7}$ = 1	$\frac{6}{1}$ = 6	$\frac{18}{4}$ = 4.5	$\frac{3}{1}$ = 3
x	1	0.6	1	0	1	1	1

Maximum profit = $1 \times 10 + 0.6 \times 5 + 1 \times 15 + 0 + 1 \times 6 + 1 \times 18 + 1 \times 3$

= 55 units.

∴ The maximum profits is 55 units.

5 (b) Apply Prim's algorithm and Kruskal's method to find the minimum cost spanning tree to the graph shown in fig. [10 marks]



→ By applying Prim's algorithm we can find minimum cost spanning tree by following steps,

Step -01 :

* Randomly choose any vertex.
* The vertex connecting to edge having least weigh is usually selected.

Step -02 :

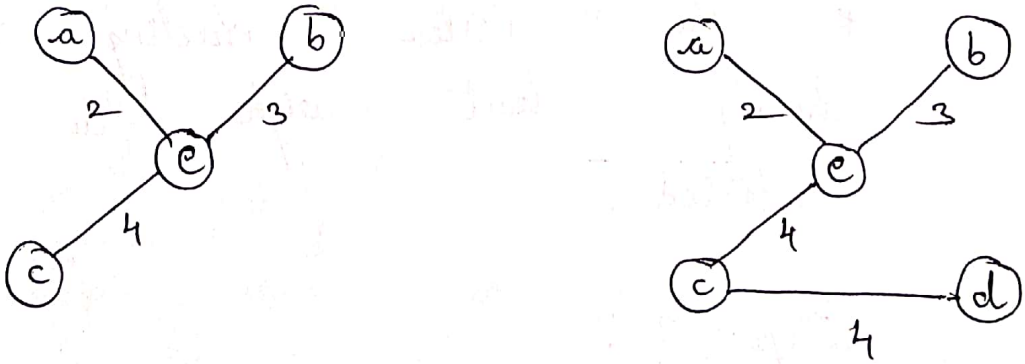
* Find all edges that connect the tree to new vertices.
* Find least weight edge among those edges and include it in existing tree.

* If including edge creates a cycle, then reject that edge and look for next least weight edge.

Step - 03 : Keep repeating step-02 until all vertices are included and minimum cost spanning tree is obtained.

Step - 1 ⟹ Step - 2

Step - 3 ⟹ Step - 4



∴ Now cost of minimum spanning tree
= Sum of all edge weights
= 2 + 3 + 4 + 4
= 13 units

By applying Kruskal's algorithm we can find minimum cost spanning tree by following steps,

Step - 1 :

Sort all the edges from low weight to high weight.

Step - 2 :

* Take the edge with lowest weight and use it to connect the vertices of graph.
* If adding an edge creates a cycle, then reject that edge and go for next least weight edge.

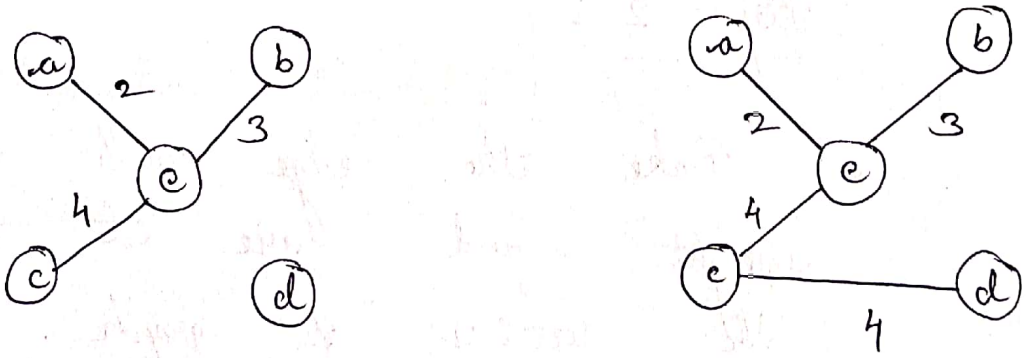
Step - 3 :

Keep adding edges until all vertices are connected and a

minimum spanning tree is obtained.

Step - 1 ⟹ Step - 2

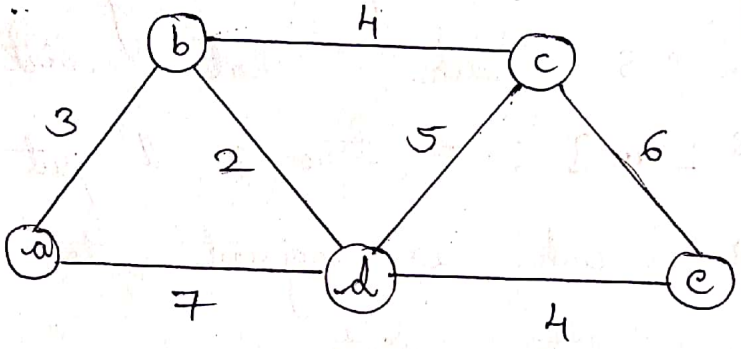
Step - 3 ⟹ Step - 4



∴ Weight of the MST
= Sum of all edge weights.
= 2 + 3 + 4 + 4
= 13 units.

Or

6 (a) Write an algorithm to solve single source shortest path problem. Apply the algorithm to the graph shown in fig. by considering 'a' as source.. [10 marks]



→ Algorithm :

Algorithm Dijkstra (G, S)

// dist [j] , 1 <= j <= n , is set to the length of the shortest.

// path from vertex v to vertex j in a diagraph G with n.

// vertices, dist [v] is set to zero.

// G is represented by its cost adjacency matrix cost [1 : n, 1 : n]

{

for i := 1 to n do

{

// Initialize S.

S [i] := false ; dist [i] := cost [v, i];

}

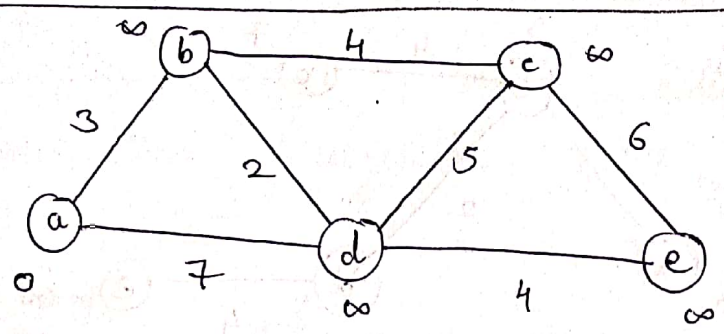
```
S[v] := true; dist[v] := 0.0; // put v in S.
for num := 2 to n-1 do
{
    // Determine n-1 paths from v.
    Choose u from among those vertices not
    in S such that dist[u] is minimum;
    S[u] := true; // put u in S.
    for (each w adjacent to u with S[w] = false) do
    // update distances.
    if (dist[w] > dist[w] + cost[u,w])) then
    dist[w] := dist[u] + cost[u,w];
    }
}
```

Step 1 : The following two sets are created.

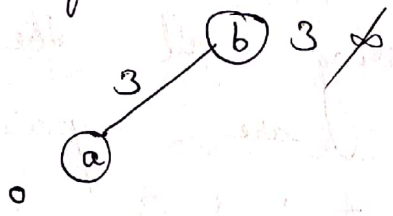
- Unvisited set : {a, b, c, d, e}
- Visited set : {}

Step 2 : The two variable π and d are created for each vertex and initialized as -

- d[a] = 0
- π[a] = π[b] = π[c] = π[d] = π[e] = NIL.
- d[b] = d[c] = d[d] = d[e] = ∞



Step 3 : Vertex a is chosen, this is because we have to estimate the path from vertex a to vertex e.

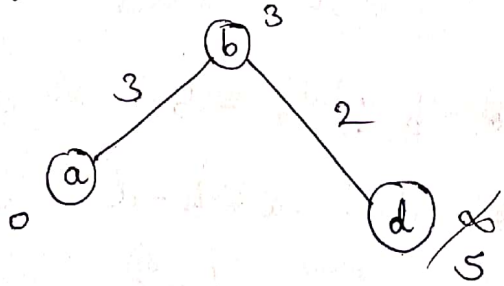


$d[a] + 3 = 0 + 3 = 3 < \infty$

$\therefore d[b] = 3$ and $\pi[b] = a$.

Step 4 :

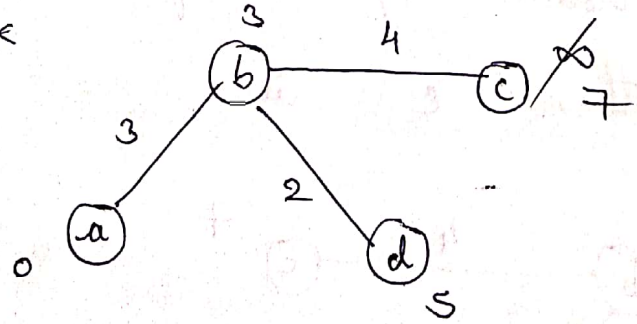
*



$d[b] + 2 = 3 + 2 = 5 < \infty$

$\therefore d[d] = 5$ and $\pi[d] = b$.

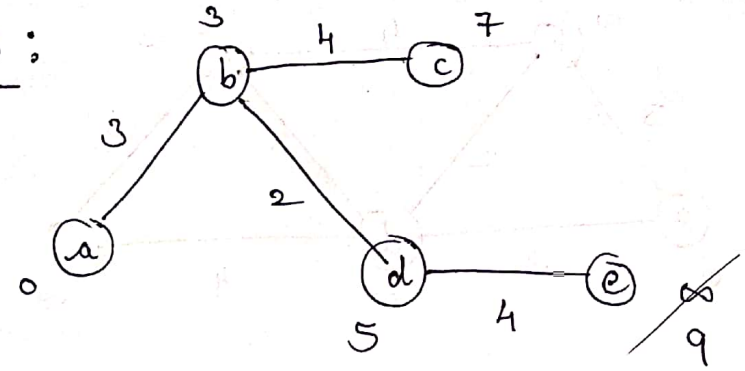
*



$d[b] + 4 = 3 + 4 = 7 < \infty$

$\therefore d[c] = 7$ and $\pi[c] = b$.

Step 5 :



$d[d] + 4 = 5 + 4 = 9 < \infty$

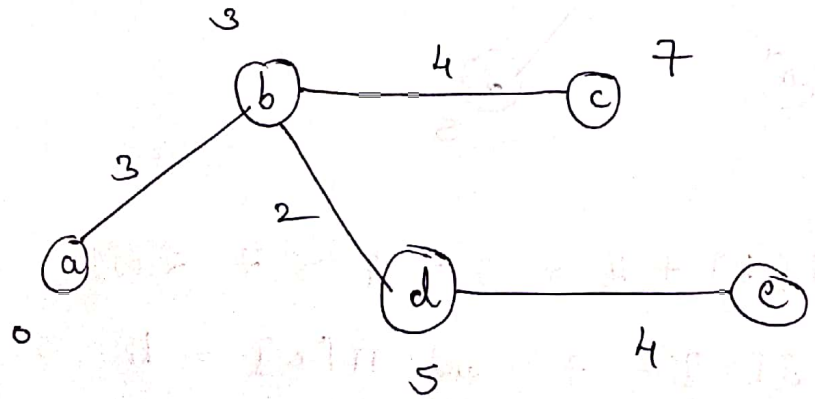
$\therefore d[e] = 9$ and $\pi[e] = d$.

By considering all the steps, now the sets are updated as,

- Unvisited set : { }
- Visited set : { a, b, c, d, e }

from a to b : a - b length 3
from a to d : a - b - d length 5
from a to c : a - b - c length 7
from a to e : a - b - d - e length 9

The shortest path from vertex a to vertex e is,



6) b) Define heap. Write bottom-up heap construction algorithm. Construct heap for the list 1, 8, 6, 5, 3, 7, 4 using bottom-up algorithm and successive key insertion method. [10 marks]

→ Heap -

A heap can be defined as binary tree with keys assigned to its nodes, one key per node, provided following conditions are met.

* The shape property -
The binary tree is essentially complete.

* The parental dominance or heap property -
The key in each node is greater than or equal to the keys in its children.

Algorithm :

Algorithm HeapBottomUp (H[1..n])

// Constructs a heap from elements of a given array

// by the bottom-up algorithm

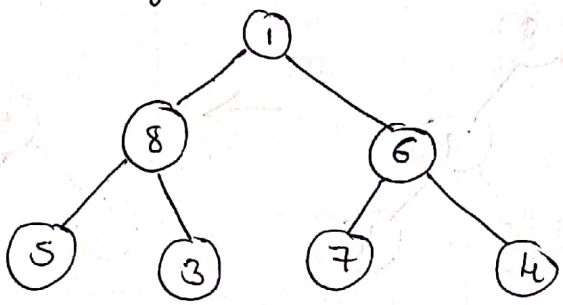
// Input : An array H[1..n] of orderable items.

// Output : A heap H[1..n]

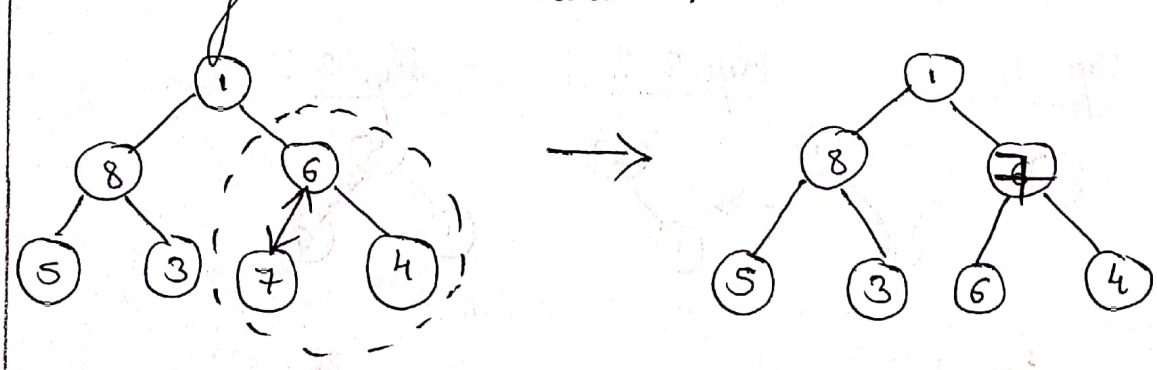
```
for i ← n/2 down to 1 do K ← i;
v ← H[K] heap ← false.
while not heap and 2*K ≤ n do
j ← 2*K
if j < n // there are two children
    if H[j] < H[j+1]
        j ← j+1
if v ≥ H[j]
heap ← true
else
    H[K] ← H[j]; K ← j H[K] ← v
```

Heap Construction : 1, 8, 6, 5, 3, 7, 4

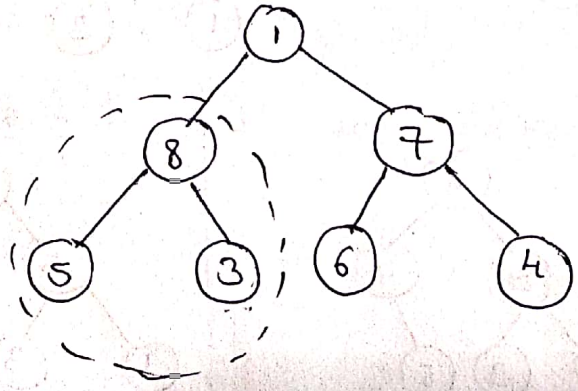
Step 1 : Initialize the structure with keys in given order.



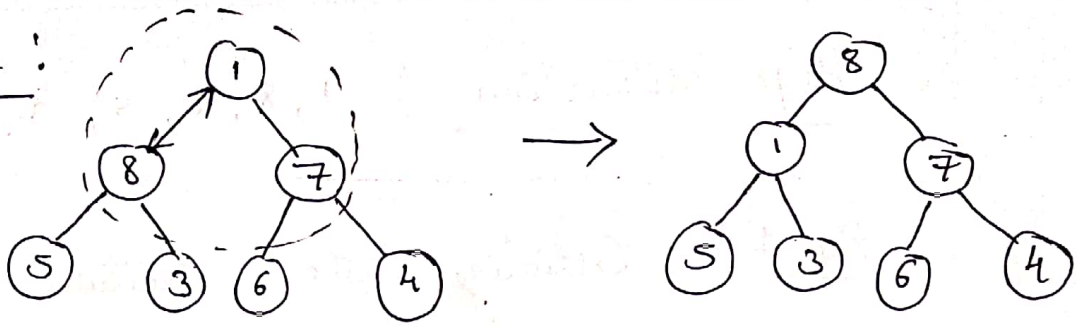
Step 2 : Check whether parent node is less than children node, if condition satisfies exchange the nodes.



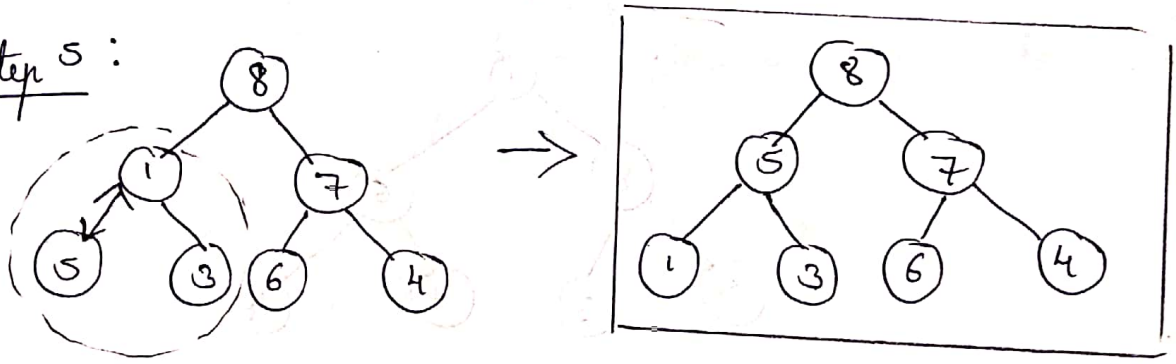
Step 3 : Repeat the step 2 if condition does not satisfies then there are no changes.



Step 4 :



Step 5 :



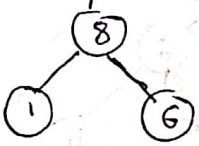
Successive key insertion (Top - down Construction)

1, 8, 6, 5, 3, 7, 4.

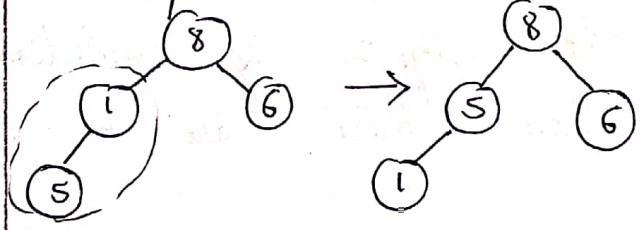
Step 1 :

Step 2 :

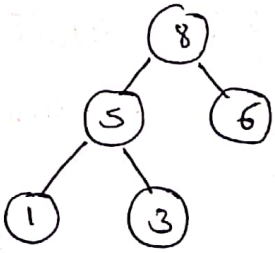
Step 3 :



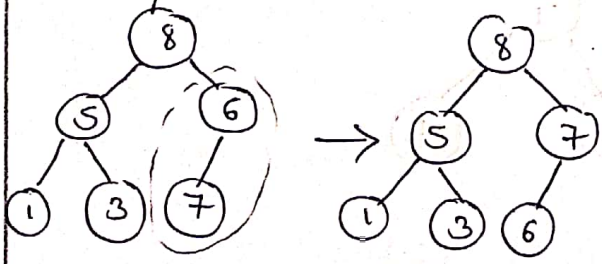
Step 4 :



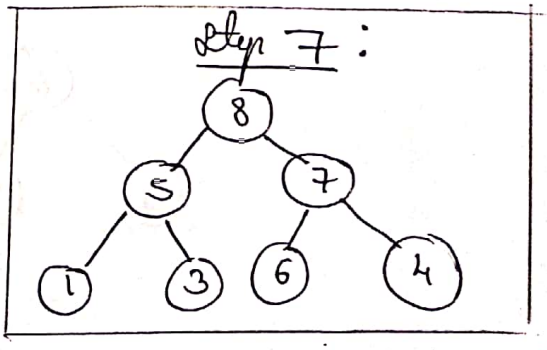
Step 5 :



Step 6 :



Step 7 :



Fourth Semester B.E. Degree Examination, Dec. 2019 / Jan. 2020

Design and Analysis of Algorithm

Module - 3

5 (a) Find the optimal solution to the knapsack instant $n = 7$, $m = 15$ using greedy method. [10 marks]

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

→ // As it is for 10 marks include algorithm as a valid content.

Algorithm :

```
void GreedyKnapSack (int m, int n, float w[ ],
                     float p[ ])
// Input: P[1:n] and w[1:n] - contains
        profits and weights.
// p[i]/w[i] >= p[i+1]/w[i+1] - decreasing
        order of ratio.
// m is knapSack capacity.
// x[1:n] is solution vector.
{
  for (i = 1; i <= n; i++)
    x[i] = 0
```

```
for (i = 1; i <= n; i++)
{
    if (w[i] > u)
        break;
    else
    {
        x[i] = 1;
        tp = tp + p[i];
        u = (int)(u - w[i])
    }
}
if(i < n)
    x[i] = u/ w[i];
tp = tp + (x[i] * p[i]);
// Display the total profit.
```

By considering this above algorithm we can apply this algorithm to find optimal solution.

By Given data,

Knapsack capacity m = 15

number of objects n = 7

Profit = P = (10, 5, 15, 7, 6, 18, 3)

Weight = W = (2, 3, 5, 7, 1, 4, 1)

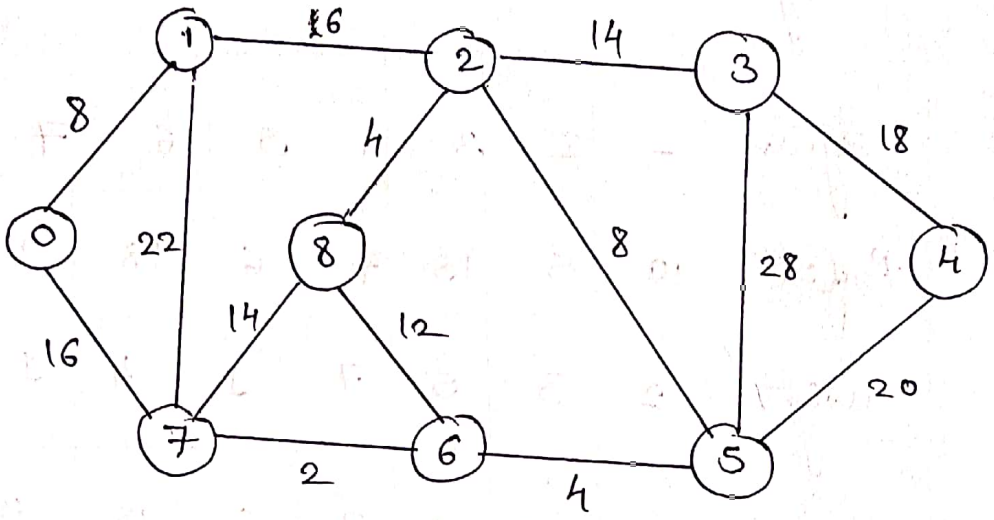
Objects	1	2	3	4	5	6	7
Profit	10	5	15	7	6	18	3
Weight	2	3	5	7	1	4	1
Profit/Weight (Pi/Wi)	10/2 = 5	5/3 = 1.66	15/5 = 3	7/7 = 1	6/1 = 6	18/4 = 4.5	3/1 = 3
x	1	0.6	1	0	1	1	1

Maximum profit = $1 \times 10 + 0.6 \times 5 + 1 \times 15 + 0 + 1 \times 6 + 1 \times 18 + 1 \times 3$

= 55 units

∴ The maximum profit is 55 units.

5) b) Find the minimum spanning tree using Kruskal's algorithm. [10 marks]



→ Kruskal's algorithm for constructing a minimum spanning tree.

// input : A weighted connected graph $G = (V, E)$

// output : E_T, the set of edges comprising a minimum spanning tree of G,

sort E in nondecreasing order of the edge weights $w(e_{i_1}) \leq \ldots \leq w(e_{i_{|E|}})$

$E_T \leftarrow \phi$; ecounter $\leftarrow 0$ // initialize the set of tree edges and its size.

$k \leftarrow 0$ // initialize the number of processed edges.

while ecounter $< |V| - 1$ do

$k \leftarrow k + 1$

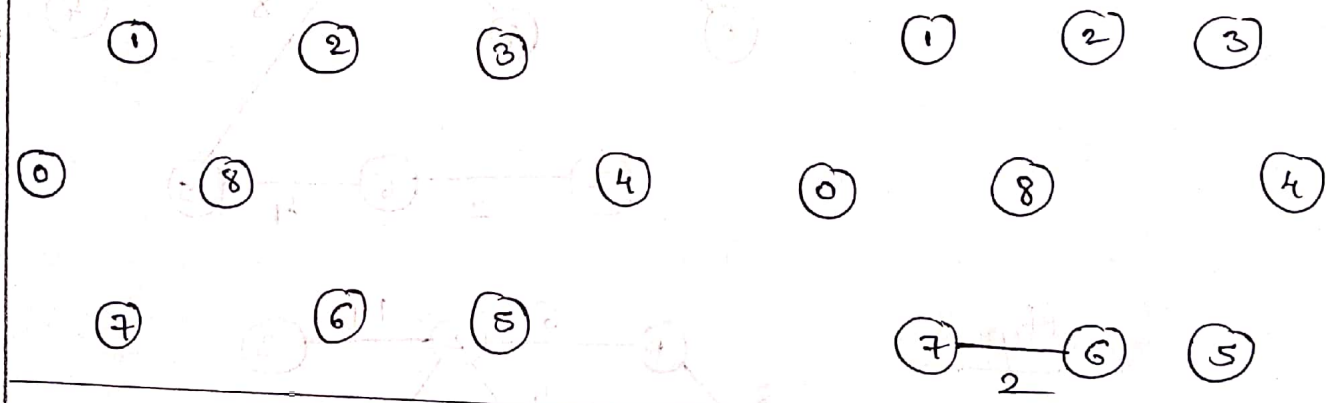
if $\mathcal{E}_T \cup \{e_{ik}\}$ is acyclic

$\mathcal{E}_T \leftarrow \mathcal{E}_T \cup \{e_{ik}\}$; ecounter ← ecounter + 1

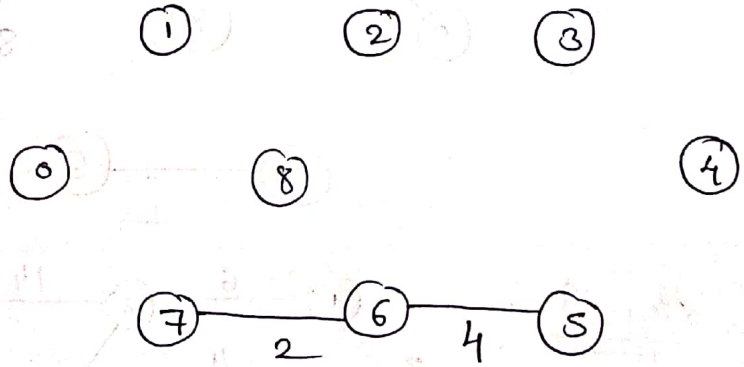
return $\mathcal{E}_T$

By applying the above algorithm, let us find minimum spanning tree for given graph.

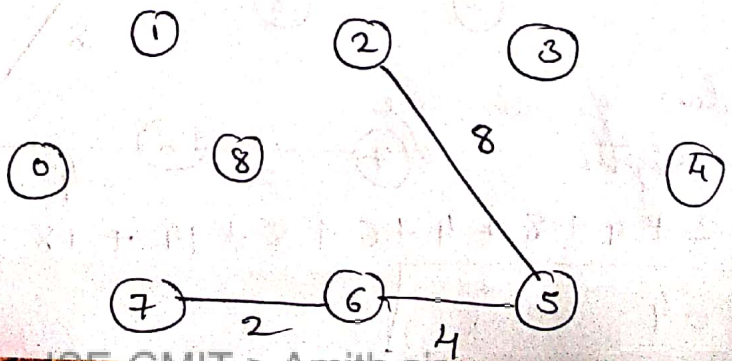
Step 1: ⟹ Step : 2



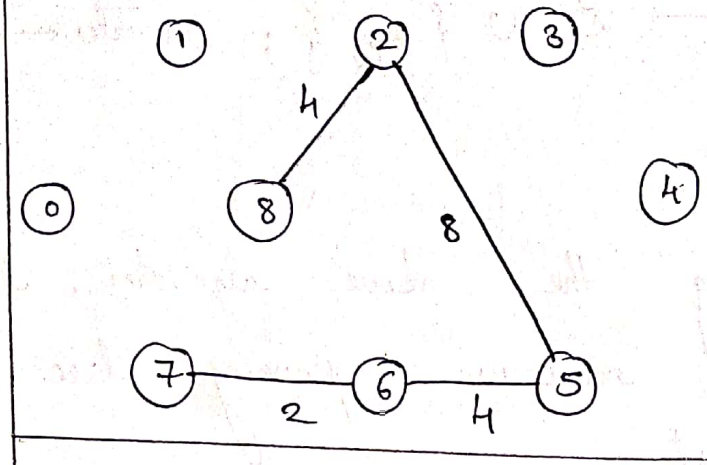
Step 3 :



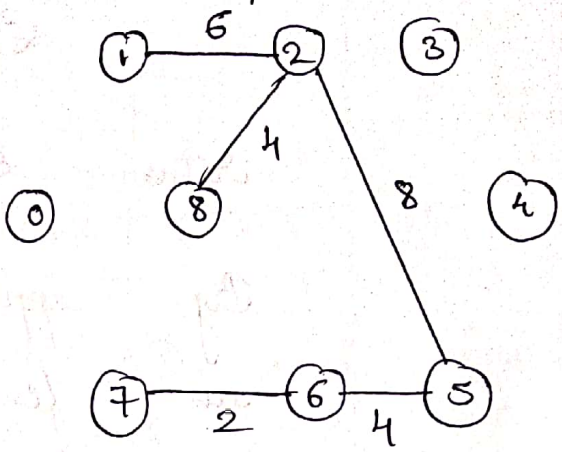
Step 4 :



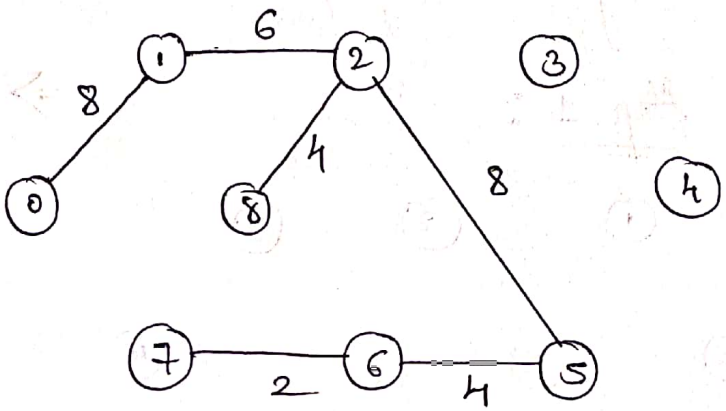
Step 5 :



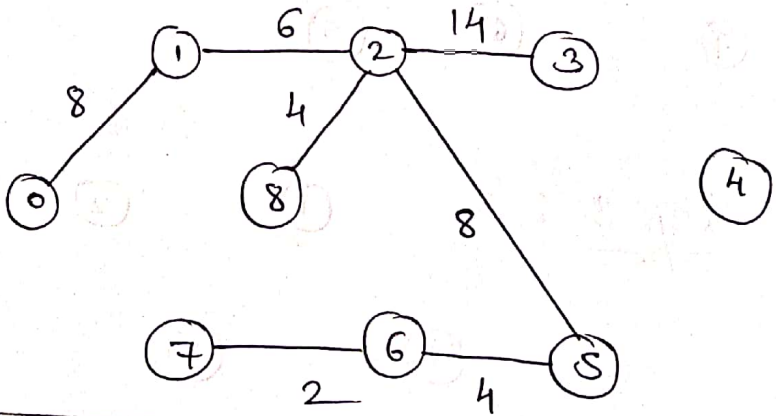
Step 6 :



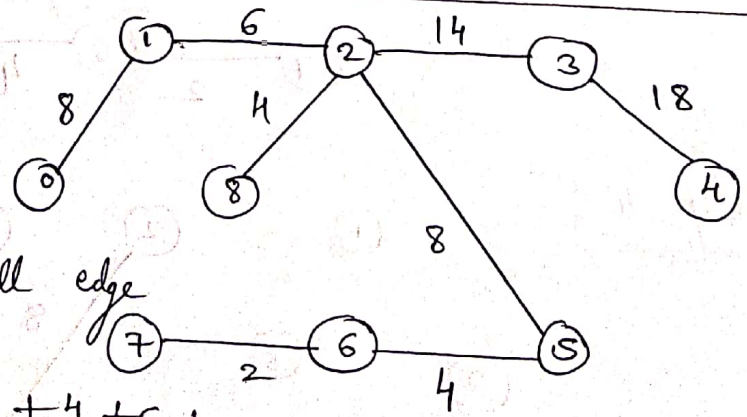
Step 7 :



Step 8 :



Step 9 :



= Sum of all edge weights

= 2+4+8+4+6+8+14+18

= 64 units

6 (a) Construct a Huffman code for the following data :

Characters	A	B	C	D	-
Probability	0.4	0.1	0.2	0.15	0.15

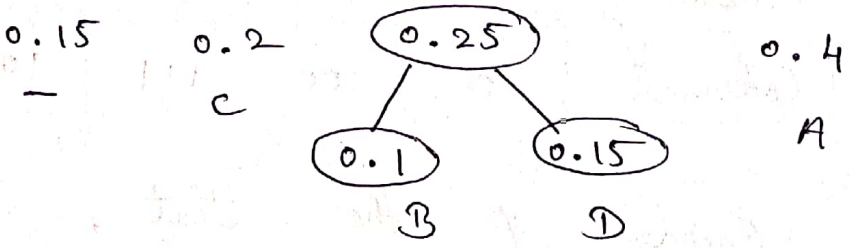
Encode the text ABACABAD and decode 100010111001010. [10 marks]

→ Arrange the probability in ascending order.

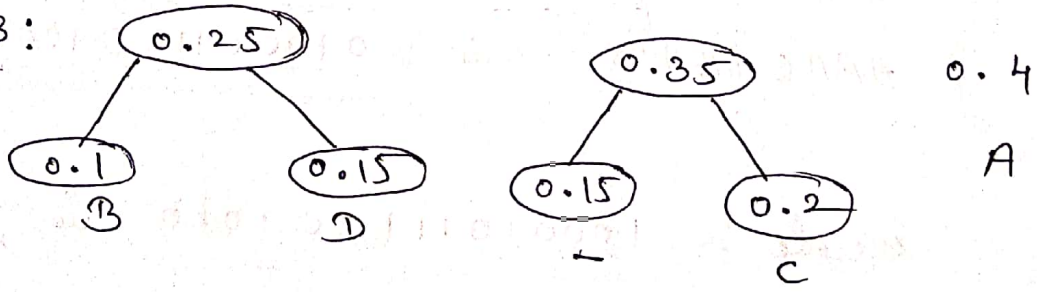
Step 1:

0.1	0.15	0.15	0.2	0.4
B	D	-	C	A

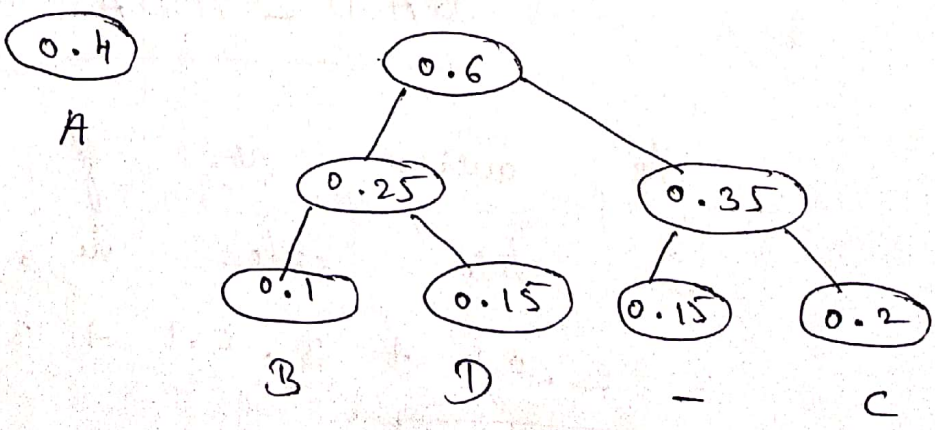
Step 2:



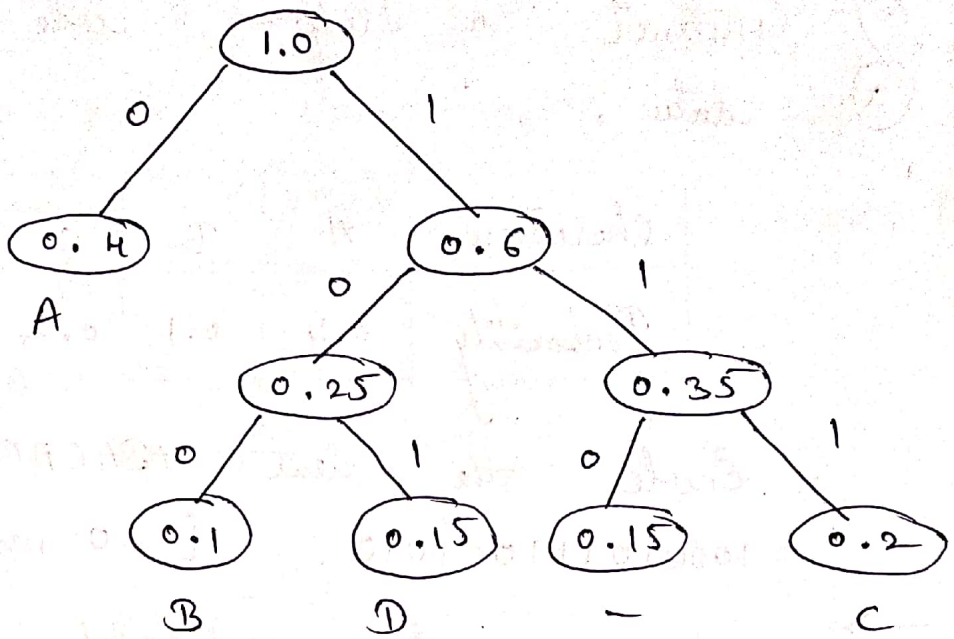
Step 3:



Step 4:



Step 5:



The resulting codewords are as follows,

Character	A	B	C	D	—
Probability	0.4	0.1	0.2	0.15	0.15
Codeword	0	100	111	101	110

Encode of the text, ABACABAD is | 01000111010001010 |

Decode: 100010111001010 is,

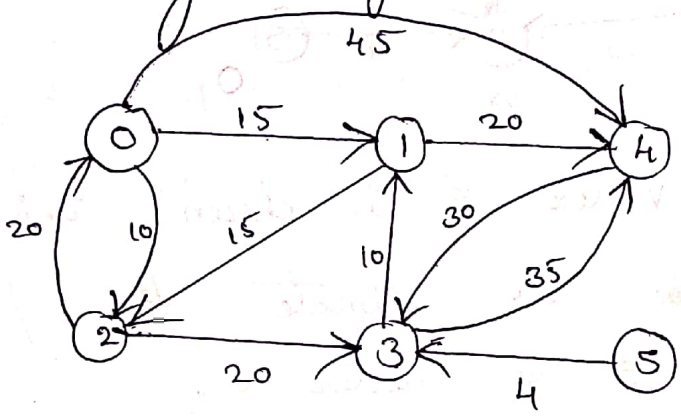
| BAD_ADA |

The average no. of bits per symbol in this code is,

= 1.0.4 + 3.0.1 + 3.0.2 + 3.0.15 + 3.0.15

= | 2.2 |

6 b) Calculate the shortest distance and shortest path from vertex 5 to vertex 0 using Dijkstra's. [10 marks].



→ * Dijkstra Algorithm is a very famous greedy algorithm.

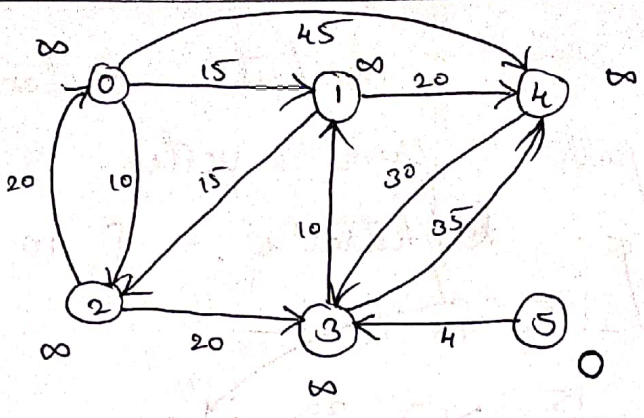
* It is used for solving single source shortest path problem.

Step 1: The following two sets are created.

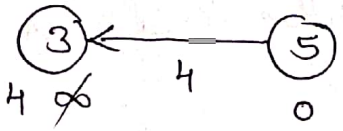
- Unvisited set: {0, 1, 2, 3, 4, 5}
- Visited set: { }

Step 2: The two variable Π and d are created for each vertex and initialized as —

- $\Pi[0] = \Pi[1] = \Pi[2] = \Pi[3] = \Pi[4] = \Pi[5] = NIL$
- $d[5] = 0$
- $d[0] = d[1] = d[2] = d[3] = d[4] = \infty$



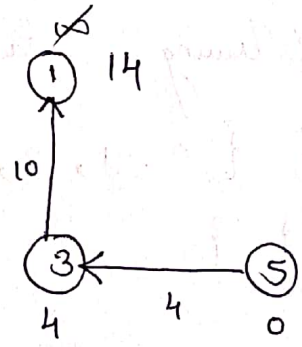
Step 3 : Vertex 5 is chosen, this is because we have to estimate the path from vertex 5 to vertex 0.



$d[5] + 4 = 0 + 4 = 4 < \infty$

$\therefore d[3] = 4$ and $\Pi[3] = 5$

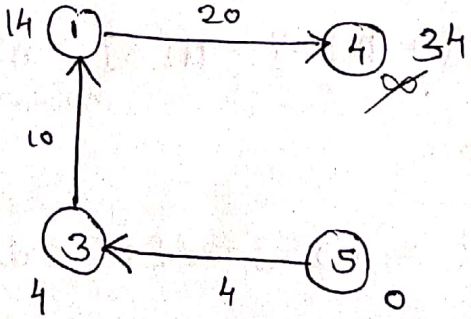
Step 4 ;



$d[3] + 10 = 4 + 10 = 14 < \infty$

$\therefore d[1] = 14$ and $\Pi[1] = 3$

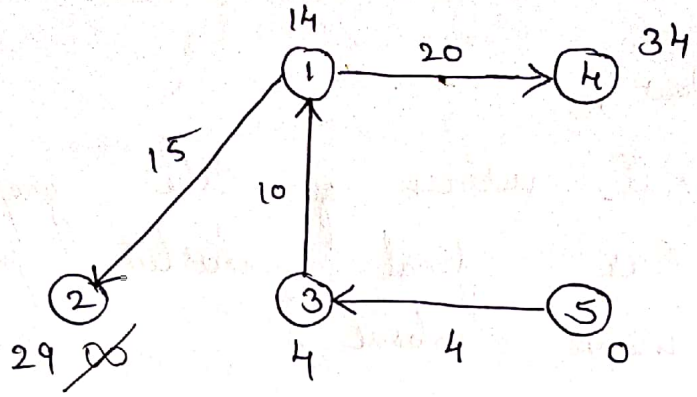
Step 5 :



$d[1] + 20 = 14 + 20 = 34 < \infty$

$\therefore d[4] = 34$ and $\Pi[4] = 1$

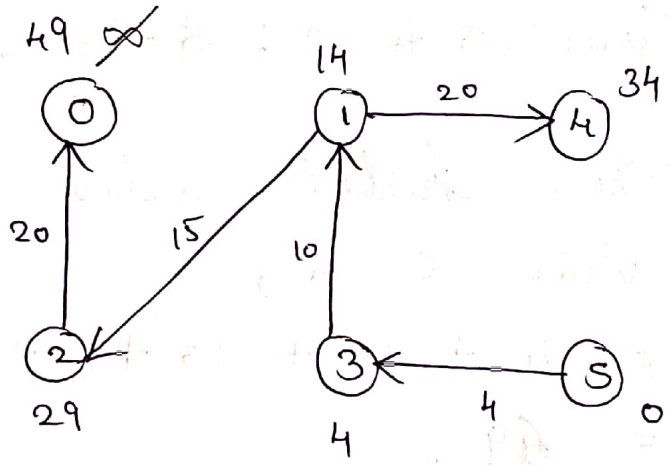
Step 6 :



$d[1] + 15 = 14 + 15 = 29 < \infty$

$\therefore d[2] = 29$ and $\Pi[2] = 1$

Step 7 :



$d[2] + 20 = 29 + 20 = 49 < \infty$

$\therefore d[0] = 49$ and $\Pi[0] = 2$.

By considering all the steps, now the sets are updated as,

- Unvisited set : { }
- Visited set : {0, 1, 2, 3, 4, 5}

Now,

- All vertices of the graph are processed.
- Our final shortest path tree is as shown above.

∴ from 5 to 3 : 5 - 3 length 4
from 5 to 1 : 5 - 3 - 1 length 14
from 5 to 4 : 5 - 3 - 1 - 4 length 34
from 5 to 2 : 5 - 3 - 1 - 2 length 29
from 5 to 0 : 5 - 3 - 1 - 2 - 0 length 49

The shortest distance from vertex 5 to vertex 0 is,

= 4 + 10 + 15 + 20 + 20
= 69.

The shortest path from vertex 5 to vertex 0 is,