1) Student Name: Venkatesh G D

2) USN : 2GI19CS175

3) BE/MBA : BE

4) Semester : 6

5) Course Name : DAA

6) Course Code :

7) Name of Colg : KLS GIT

8) Date & Time : 20-07-21 ; 3:00PM

9) Mob No : 9972287030

10) Signature : Venkatesh G D

1.) Merge Sort algorithm

Merge sort is Divide & Conquer algorithm.

→ MergeSort (arr[], l, r)
If r > l.
1) Find the middle point to divide the array into 2 halves. middle m = l + (r-l)/2
2) Call mergesort for first half:
Call MergeSort (arr, l, m).
3.) Call Mergesort for second half:
Call mergesort (arr, m+1, r)
a.) Merge the two halves sorted in step 2 & 3:
Call the merge (arr, l, m, r)

→ It divides the input array into 2 halves, calls itself for the two halves & then merges the 2 sorted halves. The merge() function is used for merging 2 halves. The merge (arr, l, m, r) is a key process that assumes that arr[l..m] & arr[m+1..r] are sorted & the 2 sorted sub arrays into one

→ Time Complexity: MergeSort is recursive algorithm & TC can be expressed as recurrance relation
$T(n) = 2T(n/2) + \theta(n)$.
The time complexity of Merge Sort is $\theta(n \log n)$ in all 3 cases (worst, avg & best).

3.) Heap - A heap is a tree based data structure in which all nodes of a tree are in specific order.

→ Bottom-up Heap construction algorithm

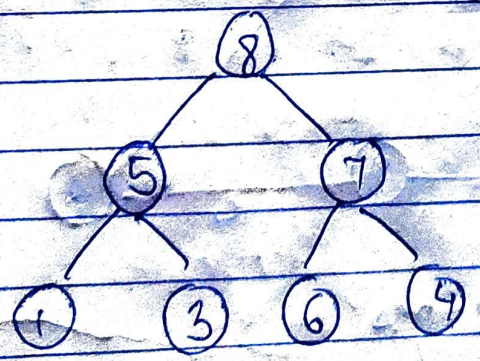
// Construct a heap from elements of given array by
// bottom up approach
// Input : An array A[1,n] of orderable items
// Output : A heap array A[1,n]

Begin

for i = n/2 down to 1 do
    Heapify(A, n, i)

for i = n down to 2 do
    exchange(A[1], A[i])
    Heapify(A, i-1, 1)

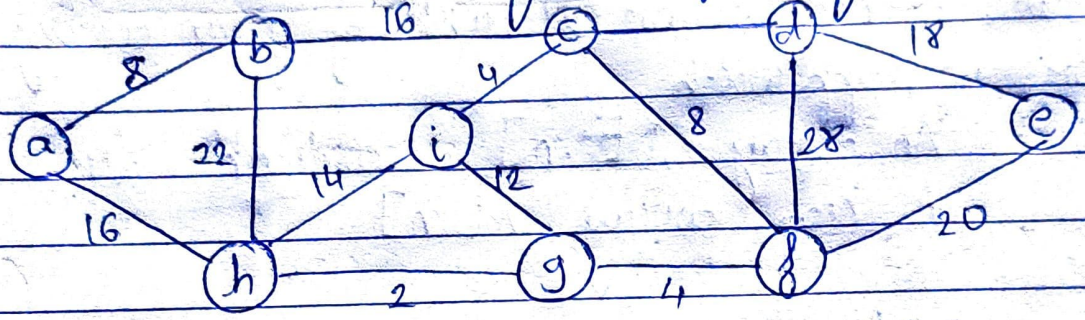
End.



A = [1, 8, 6, 5, 3, 7, 4] (Before)

A = [8, 5, 7, 1, 3, 6, 4] (After heaping)

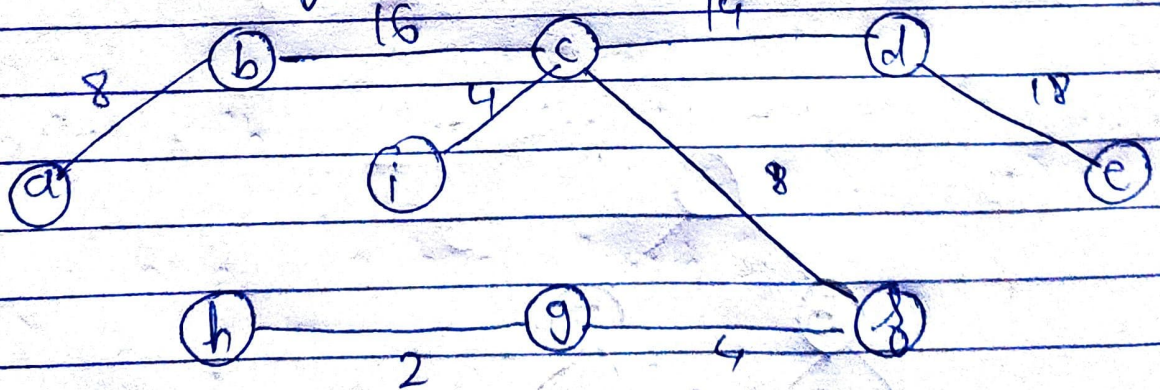
4.) The cost of spanning tree is sum of weights of all edges of the tree. There can be many spanning trees. Minimum cost spanning is a tree cost is minimum among all spanning trees



Implementation of Kruskal's algorithm:

1) Sort all the edges from low weight to high
2) Take the edge which the lowest weight & add it to the spanning tree. If adding edge created a cycle, the reject this edge
3) Keep adding edges until we reach all vertices

→



9-1 = 8 edges

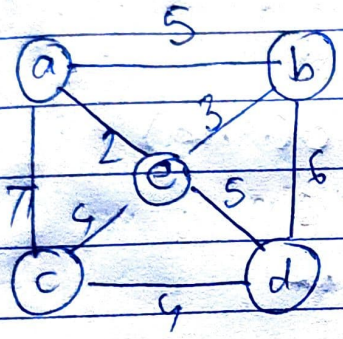
Minimum cost : 8 + 16 + 14 + 8 + 4 + 8 + 2 + 4

= 74

5) Greedy technique : General algorithm design strategy, built on following elements.

* Configuration: Different choices, values to find
* Objective function: Some configuration to be either maximized or minimized.

→



Tree vertices	Remaining vertices
e(-,-)	a(e,2) b(e,3) c(e,4) d(e,5)
a(e,2)	b(e,3) c(e,4) d(e,5)
b(e,3)	c(e,4) d(e,5)
c(e,4)	d(e,4)
d(e,4)	

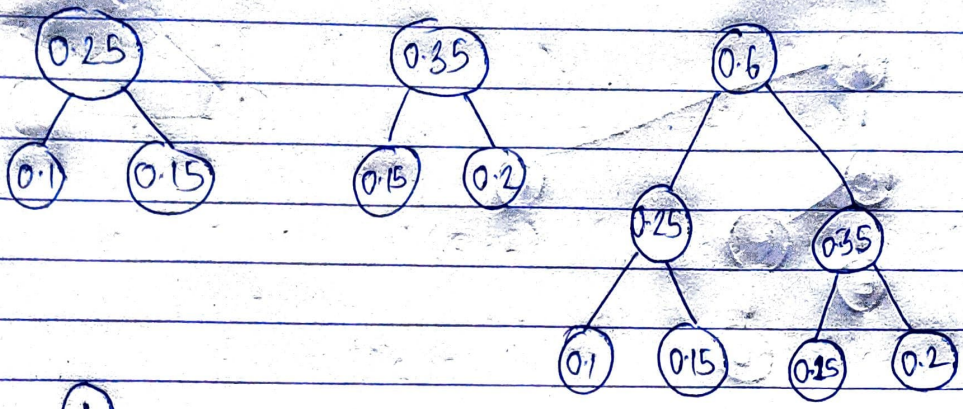
Minimum cost of Spanning tree = 13.

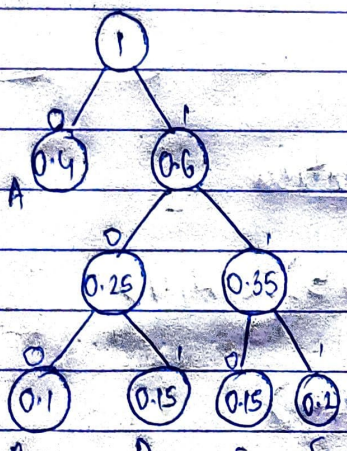
7)

Symbol	A	B	C	D	-
Frequency	0.4	0.1	0.2	0.15	0.15

Arrange in ascending order

Symbol	B	D	-	C	A
Frequency	0.1	0.15	0.15	0.2	0.4





A = 0
B = 100
C = 111
D = 101
- = 110

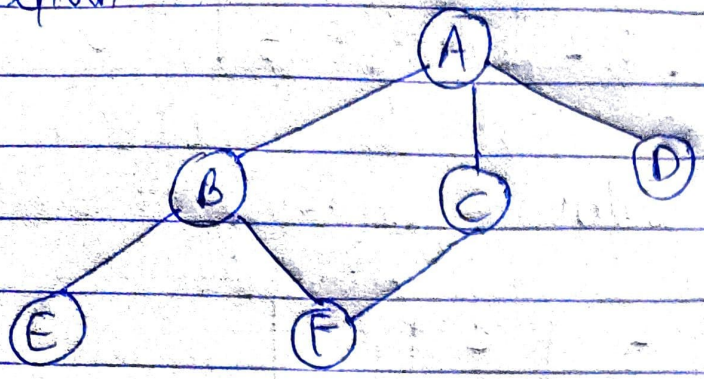
i) 1000101 1100 101 1101 0101.
ii) BAD - ADA
iii) $\left(\frac{3-2.2}{8}\right) \times 100$

CR = 0.26666 × 100
C.R = 26.66%

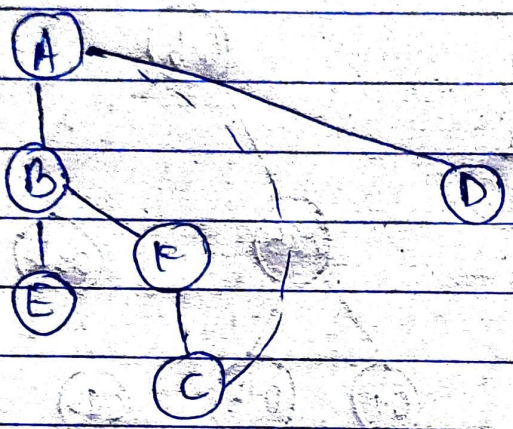
Compression Ratio

$R_1 = 0.4 \times 1 + 0.1 \times 3 + 0.2 \times 3 + 0.15 \times 3 + 0.15 \times 3$
$R_1 = 2.2$

2) Given:



DFS Traversal



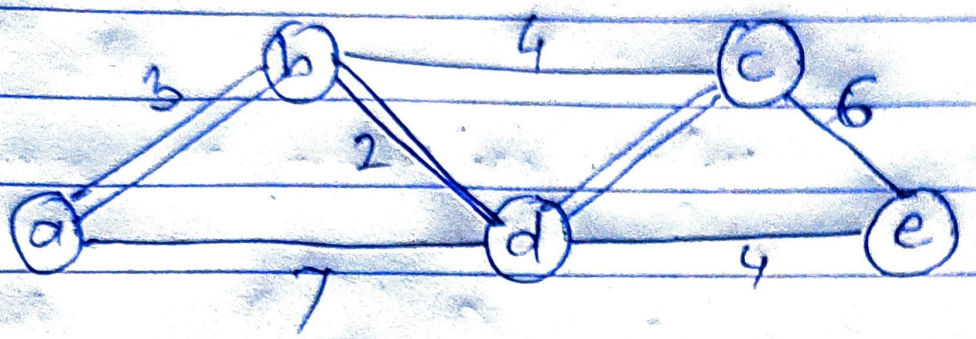
---

6) Vertices Remaining vertices

d(-,0)	a(d,7) b(d,2) c(d,5) e(d,4)
b(d,2)	a(b,5) c(d,5) e(d,4)
e(d,4)	a(b,5) c(d,5)
a(b,5)	c(d,5)
c(d,5)	

The resultant graph

6) after applying algorithm



Length da = d - b - a ⇒ 2 + 3 ⇒ 5

db = d - b ⇒ 2

dc = d - c ⇒ 5

de = d - e ⇒ 4