30/3/17

MODULE -3

Greedy Technique :

A problem is solved through sequence of subproblem, each sub-problem is solved by greedy technique.

→ feasibility ⟶ limit, budget
→ locally optimal.
→ Irrevocable. ⟶ no replacement.

Optimising problem → Greedy technique.

Defination :-

Constructing a solution through sequence of steps, expanding partially construction sol^{n} obtained so far, until the complete sol^{n} for the problem is reached.

Each step should have 3 properties.

1. **Feasible** :- The subproblem should satisfy the imposed constraints.

2. **Locally optimal** :- Among the Feasible solution for the subproblem, it should choose the best sol^{n} called as optimal sol^{n}.

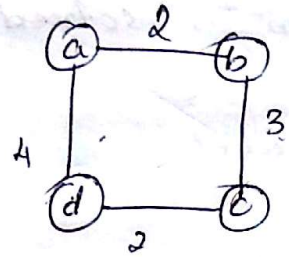
Irrevocable :- Once the subproblem is solved, it should Remain unchanged

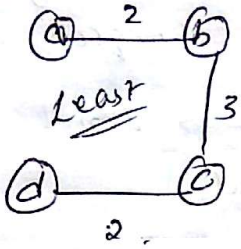
problems

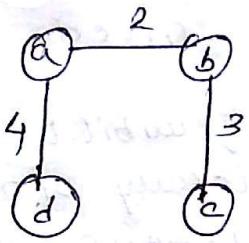
1) minimum spanning tree
2) knapsack problem
3) coin change problem.
4) single source - shortest path

1. Spanning tree [minimum]

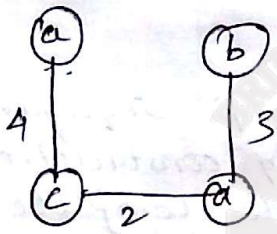
Acyclic graph which has span at all nodes

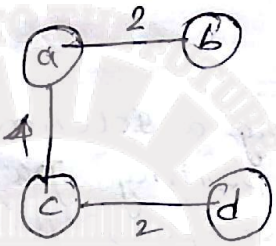






Cost = 2+3+2 = 7 4+2+3 = 9.



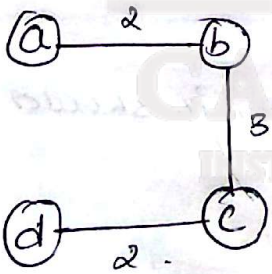


4+2+3 = 9 4+2+2 = 8.

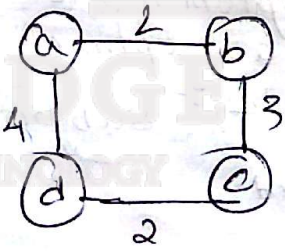
To solve minimum spanning tree we ive two algorithms.

1. Prims Algorithm :- Nearest neighbour first

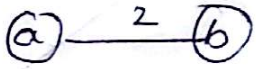
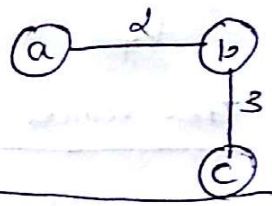
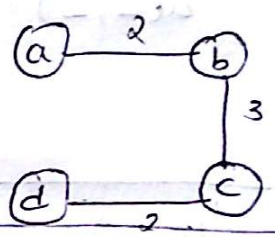
Starts with refrence node (Arbitrarily selected)



minimum spanning tree



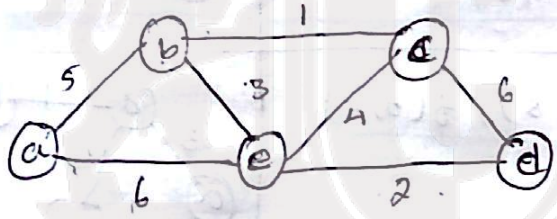
normal tree

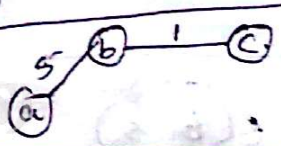
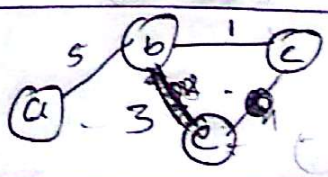
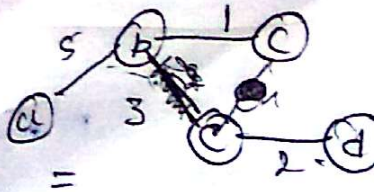
Tree vertices	Remaining vertices	Spanning tree
Initial a(–,–)	b(a,2), c(–,∞), d(a,4)	
b(a,2)	c(b,3), d(a,4)	
c(b,3)	d(c,2)	
d(c,2)	NIL	

→ Repeat these steps until there is no extra vertices.

→ The cost of the minimum edges = 2+3+2 = 7.

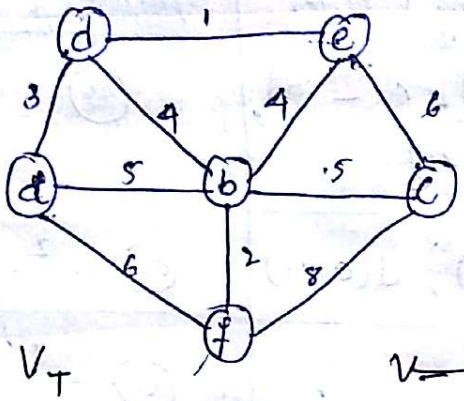
(2)

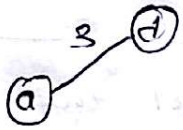
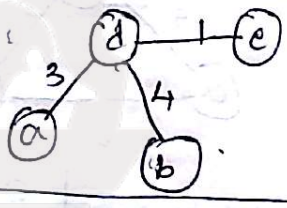
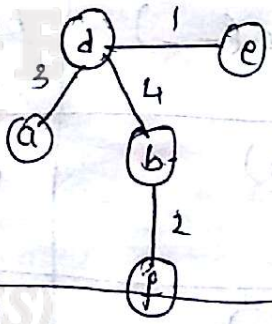
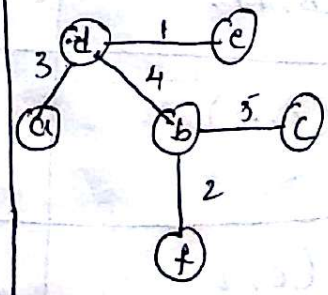


Tree vertices	Remaining vertices	Spanning tree
Initial a(–,–)	b(a,5), c(–,∞) d(–,∞) e(a,6)	
b(a,5)	c(b,1), e(b,3). d(–,∞).	
c(b,1).	d(c,6), e(c,4)	
e(c,4)	d(e,2)	
d(e,2)	NIL. minimum cost = 10.	

=

8/8/17



V_T	$V - V_T$	E_T
Tree Vertices	Remaining Vertices	Spanning tree
a(–,–)	b(a,5) c(–,∞) d(a,3) e(–,∞) f(a,6)	
d(a,3)	b(d,4), c(–,∞) e(d,1), f(a,6)	
e(d,1)	c(e,6), b(d,4) f(a,6).	
b(d,4)	c(b,5), f(b,2)	
f(b,2)	c(b,5)	
c(b,5)	NIL	

3+1+4+5+2 = 15

Prim

Algorithm Prim (G)

Input : A weighted connected graph $G = \{V, E\}$

Output : E_T set of edges constructing minimum spanning tree

$V_T \leftarrow \{v_0\}$

$E_T \leftarrow \phi$

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

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

$V_T \leftarrow V_T \cup \{v^*\}$.

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

end for

return E_T.

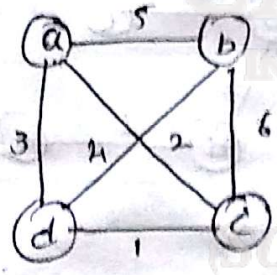
Efficiency of Prims algorithm [out of syllabus]

$O(|E| \log |V|)$

$E \rightarrow$ is the edges.

$V \rightarrow$ is the vertices.

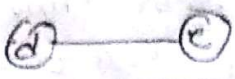
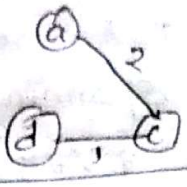
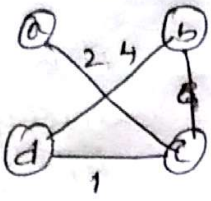
KRUSKALS Algorithm.



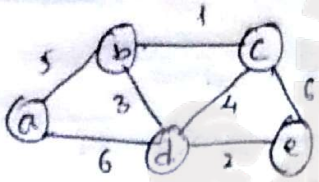
dc, ac, ad, db, ab, bc.

1 2 3 4 5 6

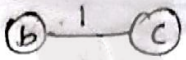
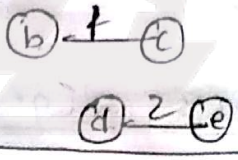
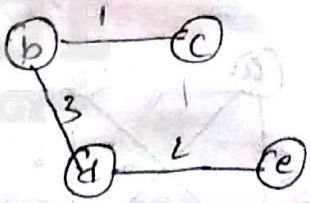
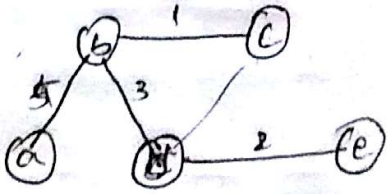
P. T. O.

Tree edge	Acyclic / cyclic	Spanning tree
de	Acyclic	
ac	Acyclic	
ad	cyclic	X
bd	Acyclic	

2.



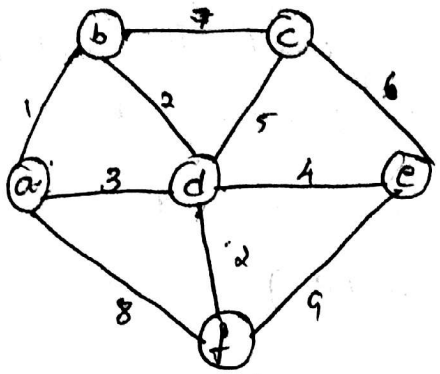
bc de bd ac ab ad or ce

Tree edge	Acyclic / cyclic	Spanning tree
bc	Acyclic	
de	Acyclic	
bd	Acyclic	
dc	cyclic	X
ab	Acyclic	
ad	cyclic	X
ce	cyclic	X

5+3+2+1 = 11

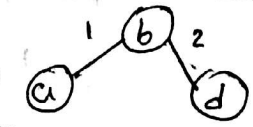
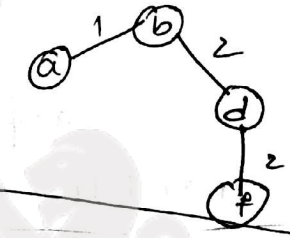
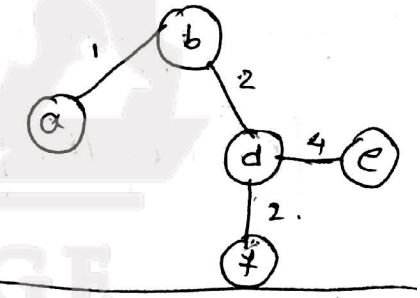
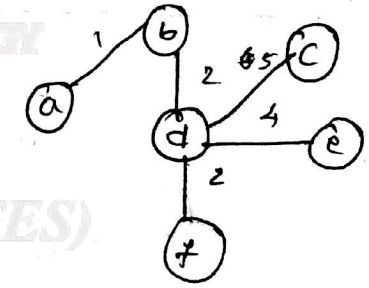
4/4/17

3.



E_T

ab bd df ad de cd ce bc af ef

Tree edges	Acyclic / cyclic	Min. cost spanning tree
ab	Acyclic	
bd	Acyclic	
df	Acyclic	
ad	Cyclic	✗
dc	Acyclic	
cd	Acyclic	

= 14

P.T.O.

Algorithm : Kruskal's

//Input : A weighted - connected graph $G = \{V, E\}$

//Output : E_T - set of edges constructing MST

Sort all edges of E in non-decreasing order ie.

$w(e_1) \leq w(e_2) \leq \ldots \leq w(e_n)$

$E_T \leftarrow \phi$

ecounter $\leftarrow 0$ ..

$k \leftarrow 0$ // Index for choosing edges

while (ecounter $< |V| - 1$) do

$k \leftarrow k + 1$

if ($E_T \cup \{e_k\}$ is acyclic) then

$E_T \leftarrow E_T \cup \{e_k\}$

ecount $\leftarrow$ ecount + 1

endif

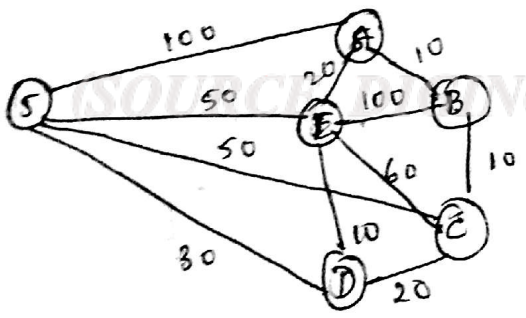
endwhile

Return E_T

Time efficiency

$O(|E| \log |E|)$

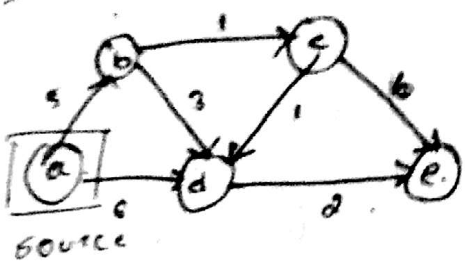
Single source shortest path (SSSP)



1. Dijkstra's Algorithm [Dia-k-stra]
2. Used in Link-state routing Algorithm

Dijkstra's algorithm is based on nearest neighbour ...

4|4|18



$a \to b = 5$

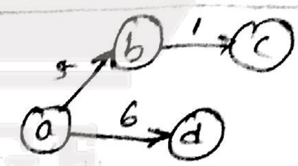
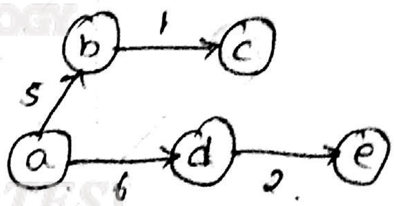
$a \to c = \infty \quad 5+1$

$a \to d = 6 \;,\; \cancel{5+3} \;,\; \cancel{6+1}$

$a \to e = \infty \;,\; \cancel{12} \;,\; 8$

a(↓ , ___) — cost

Predominate vertex

Tree vertices	Remaining vertices	Initial path
Initial a(–,0)	b(a,5), c(–,∞) d(a,6), e(–,∞)	a → a = 0.
b(a,5)	c(b,5+1), d(a,6) e(–,∞)	a → b = 5
c(b,6)	d(a,6), e(c, 6+6)	a → b → c = 6
d(a,6)	e(d, 8)	
e(d,8)	Nil	

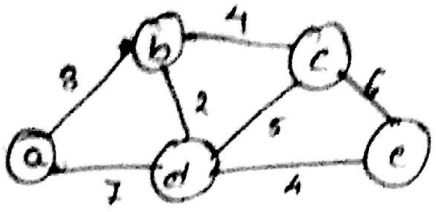
$a \to b = 5$

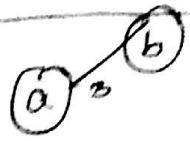
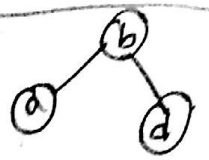
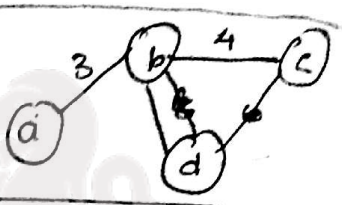
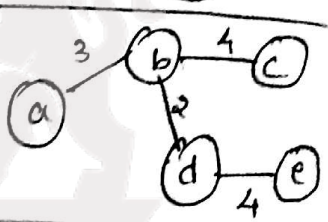
$a \to b \to c = 6$.

$a \to d = 6$.

$a \to d \to e = 8$

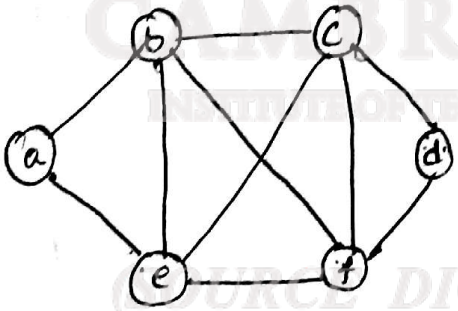
2.



Tree vertices	Remaining vertices	Shortest path
Initial a(–,0)	b(a,3) c(–,∞) d(a,7) e(–,∞)	
b(a,3)	c(b,3+4) d(b,3+2), e(–,∞)	
~~c(b,4)~~ d(b,5)	~~d(a,7)~~ e(c(b,7) e(d,11)	
c(b,7)	e(d,11)	
e(d,11)	Nil	

Shortest path = 3+4+2+4 = 13.

3.



Algorithm :- Dijkstras (G, s).

// Input : A weighted converted graph $G = \{V, E\}$ and source vertex s.

//output : d_v = shortest path from source vertex to $v \in V - s$.

P_v - Penultinate vertex of v.

```
Initialize (Q) // priority queue.
  for every vertex v in V
        dv ← ∞
        Pv ← null
        Insert(Q, v, dv)
  end for
  ds ← 0
  Decrease (Q, s, ds)
  VT ← ϕ.
  for i ← 0 to |V|-1 do
        u* ← Delete_min (Q).
        VT ← VT ∪ {u*}
```

for every vertex u adjacent to u* and belongs to V = V_T do

if (((d_{u^*} + w(u*, u)) < du)

du ← du* + w(u*, u)

P_u ← u*

Decrease (Q, u, du)

end if.

end for

end for.

6/11/17

Time complexity of Dijkstra's

$O(|E| \log M)$

Knapsack Problem

n - items

weight = {$w_1, w_2, w_3 \ldots w_n$}

Price / profit / value = {$P_1, P_2 \ldots P_n$}

W = capacity

feasibility condition : $w_1 + w_2 + \ldots \leq W$.

Out

Optimal sol^n :- maximum value in knapsack

Example

n = 3

Profit = {25, 15, 24}

weight = {18, 10, 14}

W = 25

Brute force

Exahustives search method {time consuming but good}

	condition	profit
{0}	0 < 25	= 0
{1}	18 < 25	= 25
{2}	10 < 25	= 15
{3}	14 < 25	= 24
{1,2}	28 ≮ 25	=> not feasible
{1,3}	32 ≮ 25	= not feasible
{2,3}	24 < 25	= 39
{1,2,3}	42 > 25	= Not feasible.

knapsack problem

- 0/1 knapsack (completely or dont take)
- Fractional knapsack → greedy technique

Fractional knapsack → greedy technique

1) Find the ratio of Profit/weight

2) Reorder the items base P_i/w_i such that $P_1/w_1 \geq P_2/w_2 \geq P_3/w_3 \geq \cdots$

items	Profit	weight	P_i/w_i	W Remaining capacity	Fraction of items	Profit of the knap sack
3	24	14	1.71	25 − 14 = 11	1	0 + 1×24 = 24
2	15	10	1.5	11 − 10 = 01	1	24 + 1×15 = 39
1	25	18	1.38	1 − 1 = 0	1/18	39 + (1/18 × 25) = 40.38

Ex1 : n=3 (1 2 3)

Profit = {25, 15, 24}

weight {18, 10, 14}

W = 25

items

Item	P	W	Pi/wi
1	25	18	1.38 III
2	15	10	1.5 II
3	24	14	1.71 I ← higher value first

2. n=4

weight = {7, 3, 4, 5}

profit = {42, 12, 40, 25}

W = 12

Item	P	W	Pi/wi
1	42	7	6
2	12	3	4
3	40	4	10
4	25	5	5

items	Profit	weight	Pi/wi	Remaining capacity (W)	fraction of item x[i]	Profit of knapsack x[i]
3	40	4	10	12-4=8	1	40
1	42	7	6	8-7=1	1	42+40=82
4	25	5	5	1-1=0	1/5	87
2	12	3	4	0	0	87+0 = 87

Profit = 87

3. $n = 7$

weight = {2, 3, 5, 7, 1, 4, 1}

profit = {10, 5, 15, 7, 6, 18, 3}

$w = 15$

Algorithm Greedy-knapsack (W, n)

//Input : P[1...n] Price W[1....n] weight of items sorted in non-decreasing order of P_i/w_i ratio .. W - capacity of knapsack.

//output : x[1...n] Solution vectors

for i ← 1 to n do

x[i] ← 0

end for

U ← W

for(i ← 1 to n do

if (w[i] > U) then

break.

~~end for~~ end if

x[i] ← 1.0

U ← U - w[i]

end for

if (i ≤ n)

x[i] ← U/w[i]

end if

return (x)

U → left out capacity or ~~capacity~~ accomodated in knapsack [capacity of knapsack]

8. n = 3

P {25, 24, 15}

W {18, 15, 10}

W = 20

Item	P	W	Pi/wi
1	25	18	1.36
2	24	15	1.7
3	15	10	1.5

capacity of knapsack

remaining weight

items	P	wi	U = 20	x
2	24	15	20 − 15 = 5	1.0
3	25	10	5 − 5 = 0	5/10 = 0
1				

return

= 0

+ 24

= 7.5

+ 0

31.5

8/4/17

Job-sequencing / sheduling with deadline.

Set of jobs - n

each job associated with deadline (time to complete).

each job is associated with price

Feasibility condition : sequenced jobs have to be completed with that deadline.

Optimal solution : sequence with maximum profit/p

Ex: n = 5

Job no :-	1	2	3	4	5
deadline	3	3	2	1	2
Price	5	1	20	10	15

Reorder

based price

Descending order

Job considered	Price	deadline	Action Taken	Total profit
3	~~20~~ 20	2	1 2 3 | | 3 | | (3, 2)	0 + 20 = 20
5	15	2	1 2 3 | 5 | 3 | | (5, 1)	20 + 15 = 35
4	10	1	not scheduled	35
1	5	3	1 2 3 | 5 | 3 | 1 | (1, 3)	40
2	1	3	Not sequenced	40

Profit = 40

x : 2 n = 7

Job no:	1	2	3	4	5	6	7
deadline	1	3	4	3	2	1	2
Price	3	5	20	18	1	6	30
	6	5	2	3	4	4	1

Job considered	Price	deadline	Action Taken	Total Profit
7	30	2	~~3~~ 7 ~~30~~ (7, 2)	30
3	20	4	7 3 (3, 4)	30+20 = 50
4	18	3	7 ~~4~~ 3 (4, 3)	50+18 = 68
6	6	1	6 7 4 3 (6, 1)	68+6 = 74
2	5	3	not scheduled	74
1	3	1	not scheduled	74
5	1	2	Not scheduled	74

Algorithm :- Job-sequencing (d, P, n).

// Input :- Jobs ordered based on price in non-increasing order d - deadline, P - Price, n - no of jobs

// output :- sequenced Job - J.

```
J ← { 1 }
for i ← 2 to n do
    if (all the jobs in J ∪ {i} can be completed with deadline) then
        J ← J ∪ {i}
    end if
end for
return J
```

X

General method of Greedy Technique

Note by default need to be included in greedy technique explaination.

Algorithm Greedy (a, n)

// Input : An array a[1...n] of subproblems

// output : Feasible and optimal solution.

```
solution ← ϕ
for i ← 1 to n do                    // solution of subproblem i to x.
    x ← select (a)
    if (feasible (solution, x)) then
        solution ← Union (solution, x)
    end if
end for
return (solution)
```

(F)

15/4/17

ASCII / EBCDIC.

Fixed length coding.

Huffmann coding.

coding Technique to represent non-numeric elements with binary value.

→ variable length coding.

→ Based on the frequency of occurence of the character, the no of bits to represent varies. ie frequently occuring characters will be assigned with less no of bits. and rarely occuring characters with maximum no of bits

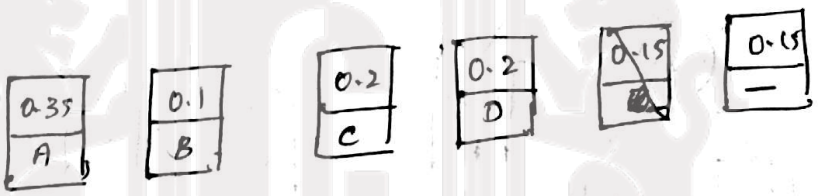
→ Huffmann coding is also called as one of the loss less compression technique.

Ex:-

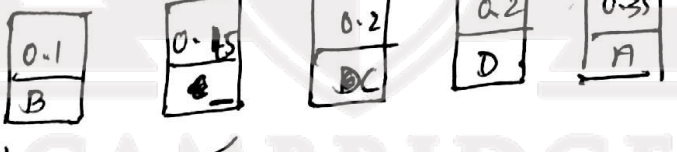
Character	A	B	C	D	E
Probability	0.35	0.1	0.2	0.2	0.15

frequency

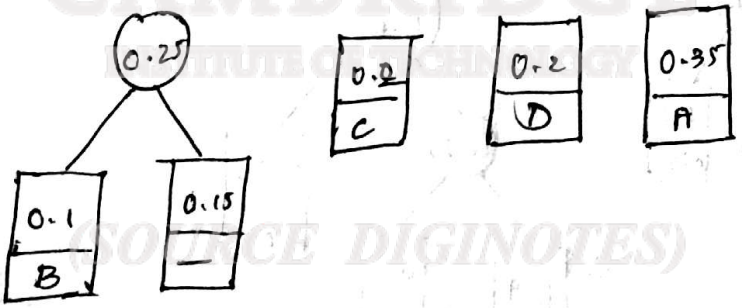
construct a tree for every character



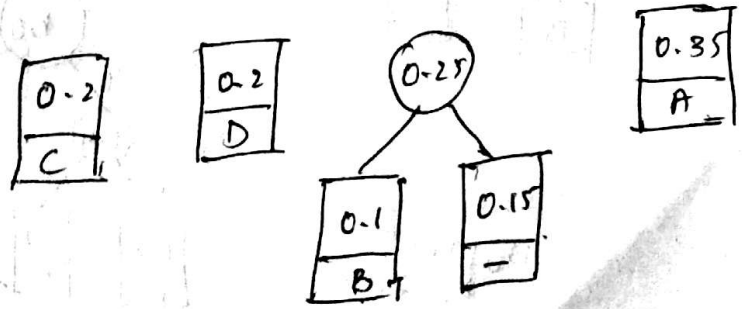
Reorder (ascending)



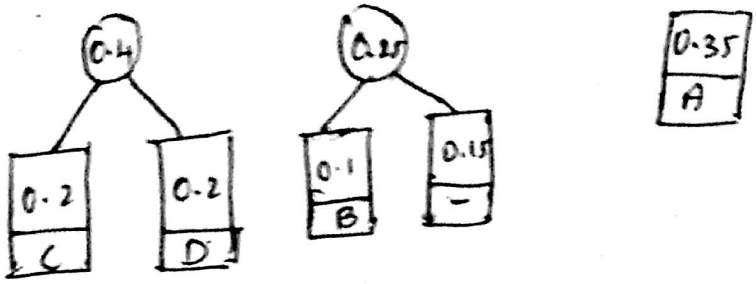
merge



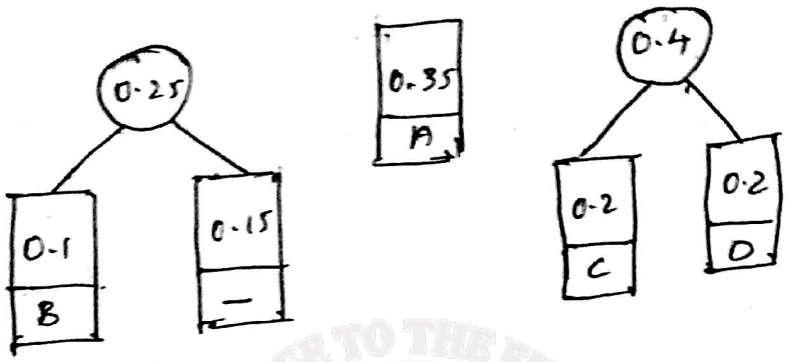
Reorder



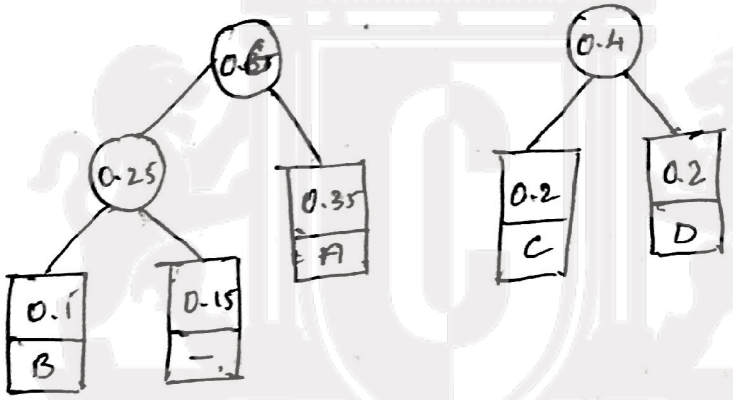
merge.



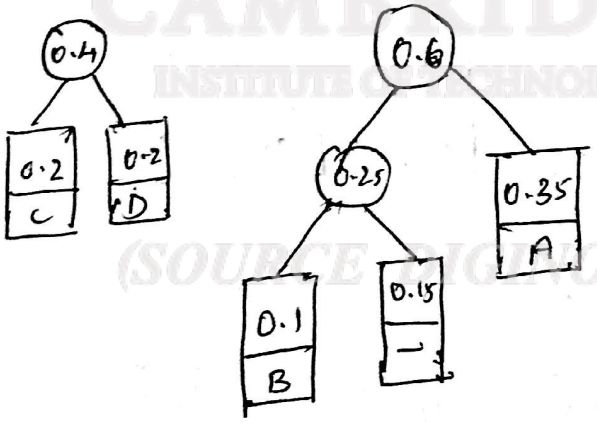
Reorder



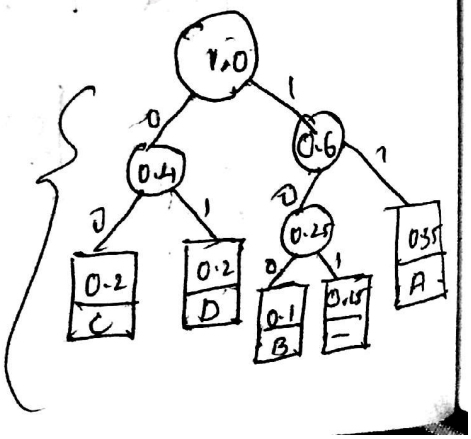
merge



Reorder



merge



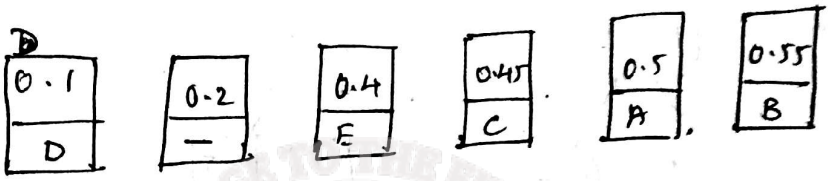
Huffman tree

A	B	C	D	—
11	100	00	01	101

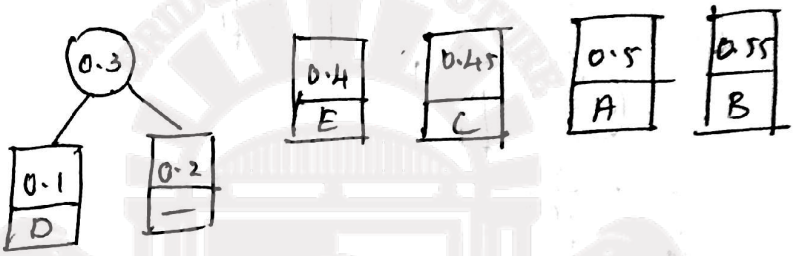
example :

character	A	B	C	D	E	—
Probability	0.5	0.55	0.45	0.1	0.4	0.2

Reorder

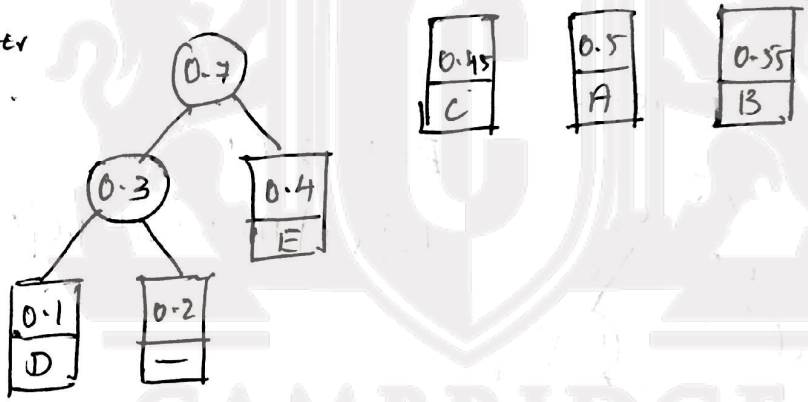


merge.

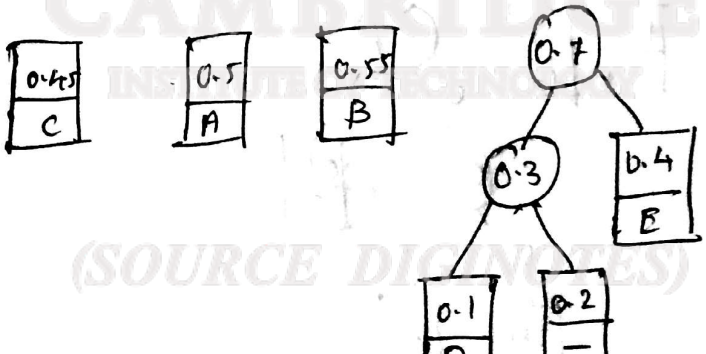


~~Reorder~~

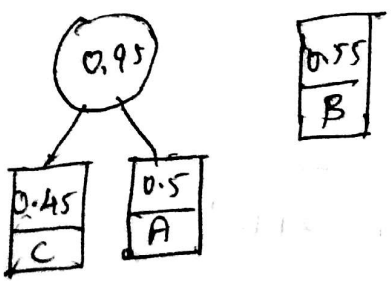
merge.



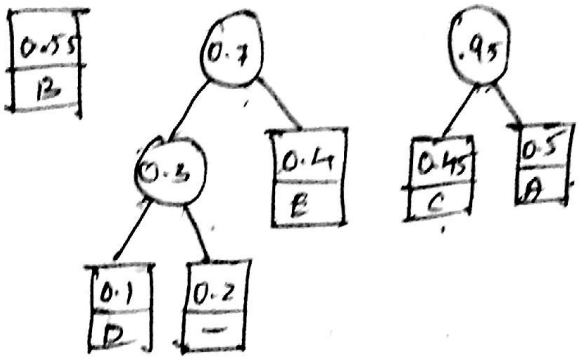
Reorder



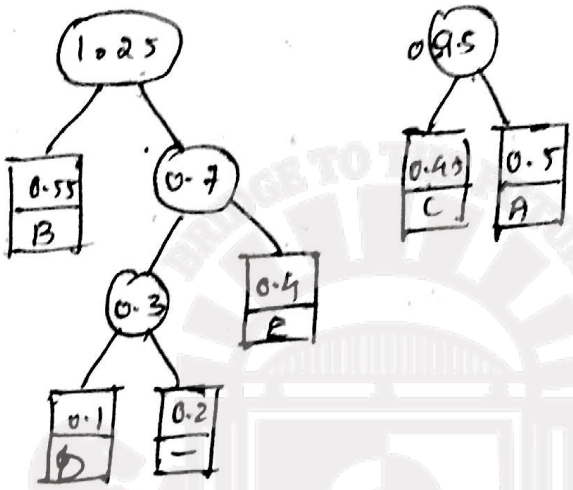
Merge



Reorder



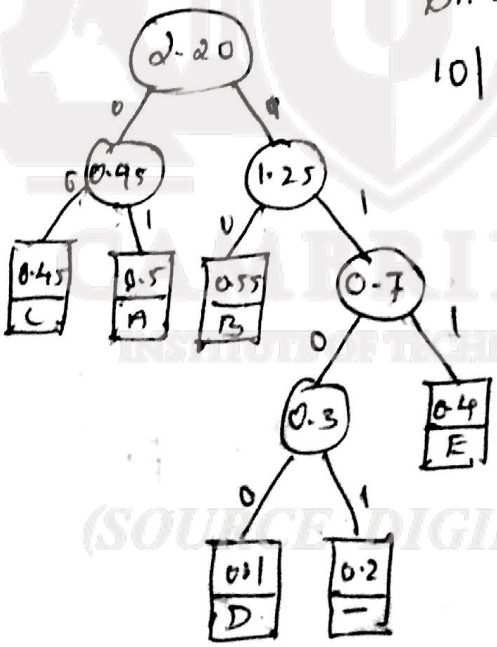
merge.



Reorder and merge.

BAD DAA

10|01|1100|1101|1100|01|01



A	B	C	D	E	_
01	10	00	1100	111	1101

Algorithm :- Huffman tree.

//Input : A (array) vector of characters and their frequency.

//Output : Huffman tree.

step 1 : Initialize 'n' node trees and label them with the character of the alphabet. Record the frequency of each character in its tree root to indicate the tree's weight

Step 2 : Repeat the following operation until a single tree is obtained.

a) Find the two trees with smallest weight.

b) make them as left and Right subtree of a new tree.

c) Record the sum of their weights in the root of the new tree as its weight.

step 3 :- Mark the left edge with '0' and right edge with '1'.

18/4/17.

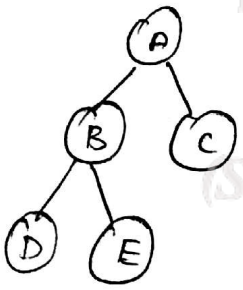
Heap sort

uses : Transform and conquer design technique.

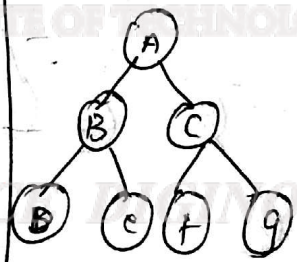
Heap tree

* Binary tree.
* Essentially complete Binary tree.

Strictly B.T



complete B.T.



→ 2^d.

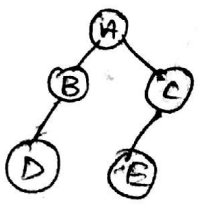
→ children same level

→ level completely fill.

→ $2^{d+1} - 1$ → total no of node.

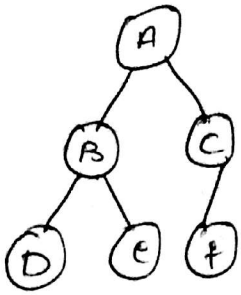
height = 1 + depth.

Almost complete B.T



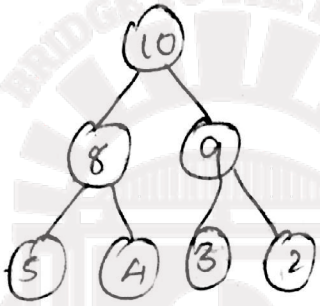
$\frac{2*k}{2*k+1}$

* Essentially complete Binary tree.
  is an almost complete B.T.
  missing nodes are from Right to left.



* Parental Dominance.

  Parent node value should be larger than children (Descending heap tree

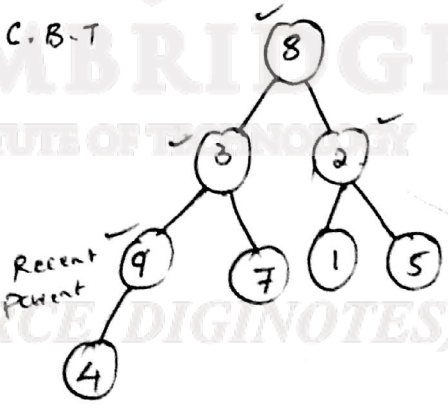


18/4/17

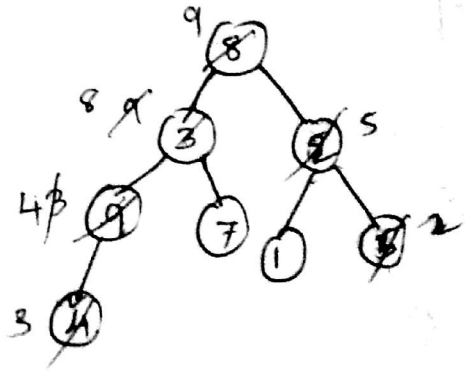
Construction of Heap tree (& Heapification)

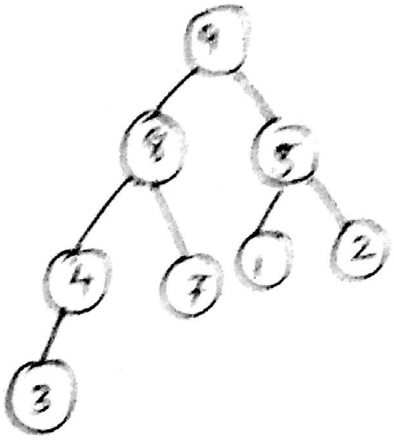
	1	2	3	4	5	6	7	8	
H[1....n]	8	3	2	9	7	1	5	4	\ \ \ \

Represent Array into essentially C.B.T



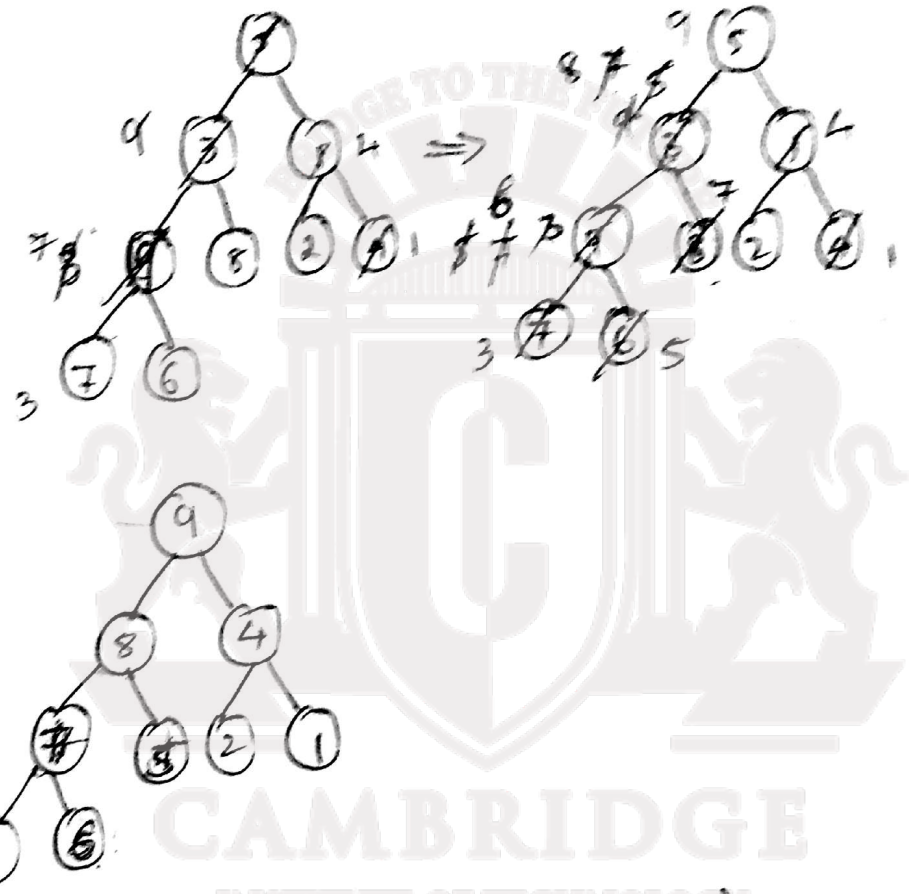
From recent Parent try to satisfy parental dominance.





2.

5	3	1	9	8	2	4	7	6



Algorithm :- Heap_Bottom_up (H[1....n])

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

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

for $i \leftarrow \lfloor n/2 \rfloor$ downto 1 do. ($n/2$ → recent parent)

$k \leftarrow i$

$v \leftarrow H[k]$

Heap ← false.

while (not heap and $2*k \leq n$) do.

$j \leftarrow 2*k$.

```
if (j<n) then // there are 2 children
  if (H[j] < H[j+1]) then
      j ← j+1
  end if
end if
if (v ≥ H[j])
    heap ← true
else
    H[k] ← H[j]
    k ← j
  end if
end while
  H[k] ← v
end for
```