

## Dynamic Programming :-

Dynamic Programming is a technique for solving problems with overlapping subproblems. Typically, these subproblems arise from a recurrence relating a given problem's solution to

→ Computing Binomial Coefficient :- solutions of smaller subproblems.

$$nC_k = \frac{n!}{(n-k)! k!}$$

$$C(n, 0) = nC_0 = 1$$

$$C(n, n) = nC_n = 1$$

$$C(n, k) = C_{k-1} + C_k$$

```
int binocoeff (n, k)
```

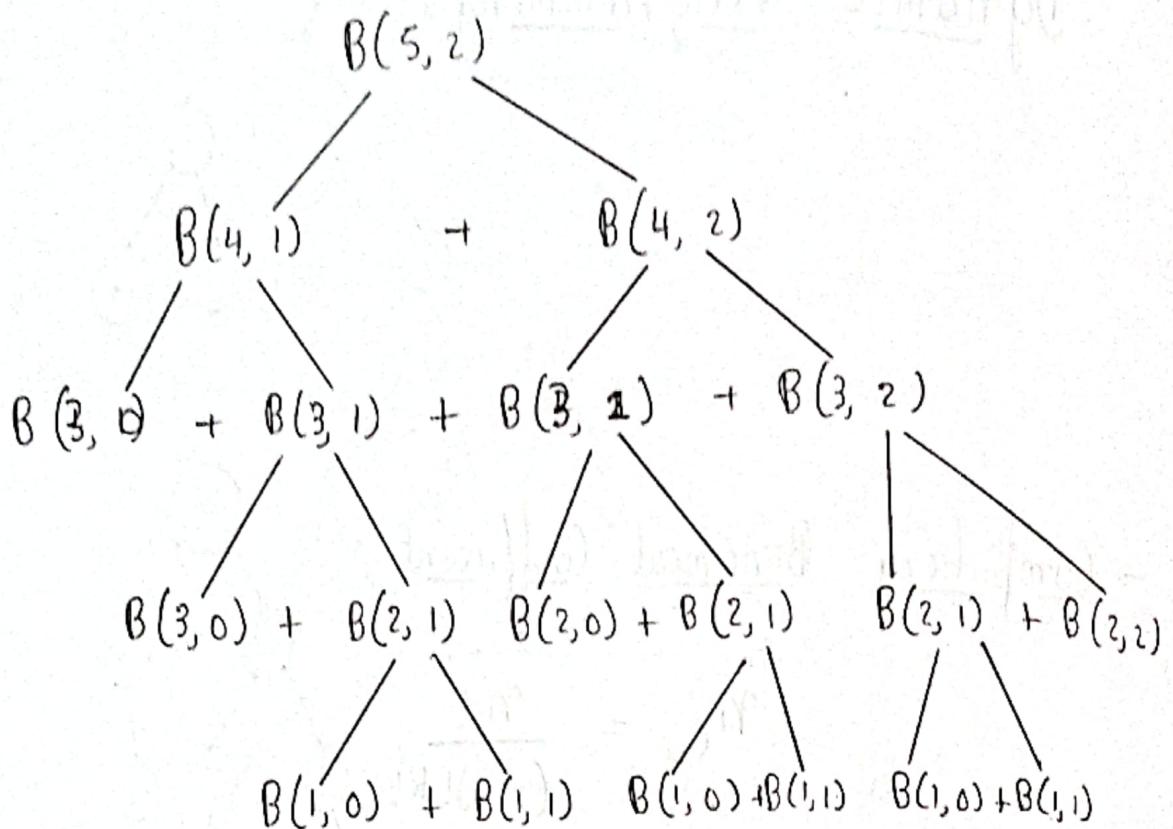
```
{
```

```
    if (k == 0 || k == n)
```

```
        return 1;
```

```
    Else
```

```
        return (binocoeff (n-1, k-1) + binocoeff (n-1, k))
```



Number of addition : 9

Here , there are some overlapping problem i.e computed more than once (Ex:-  $B(3,1)$ ,  $B(2,1)$  etc..)

→ Converting to Dynamic Programming:-

Compute  $C(5,2)$

	K		
	0	1	2
0	1		
1	1	1	
2	1	2	1
3	1	3	3
4	1	4	6
5	1	5	10

Number of additions = 7

Triangle →  $B(3,1)$

Rectangle →  $B(2,1)$

Required value →  $B(1,1)$

## Algorithm :-

Binomial ( $n, k$ )

|| Input :- A pair of non-negative integers  $n > k > 0$

|| Output :- The value of  $c(n, k)$

```
for i ← 0 to n do
    for j ← 0 to min(i, k) do
        if (j = 0 @ j = i)
            c[i,j] ← 1
        Else
            c[i,j] ← c[i-1,j-1] + c[i-1,j]
    end for
end for
return c[n,k]
```

Analysis:-

→ Basic Operation :- Addition

$$A(n, k) = \sum_{i=0}^k \sum_{j=0}^i 1 + \sum_{i=k+1}^n \sum_{j=0}^k 1$$

Triangle Part      Rectangle Part

$$= \sum_{i=0}^k (i - k + x) + \sum_{i=k+1}^n k - x + x$$

$$= \sum_{i=0}^k i + k + \sum_{i=k+1}^n 1$$

$$= \frac{k(k+1)}{2} + k(n-k)$$

$$\therefore A(n, k) \in \Theta(nk)$$

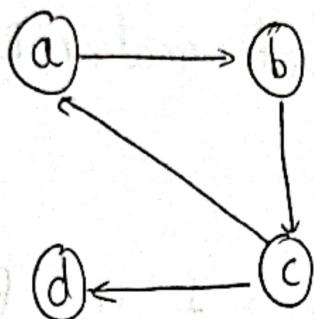
$$\text{Space Complexity} = \Theta((n+1)(k+1))$$

⇒ Compute  $C(6, 3)$  using Dynamic Programming :-

	0	1	2	3	K
0	1				
1	1	1			
2	1	2	1		
3	1	3	3	1	
4	1	4	6	4	
5	1	5	10	10	
6	1	6	15	(20)	$C(6, 3) = 20$

$\Rightarrow$  Worshall's Algorithm :-

- ⑥ Used to find Transitive Closure
- ⑥ Applicable on directed graphs.
- ⑥ Takes adjacency matrix as input



	a	b	c	d
a	0	1	0	0
b	0	0	1	0
c	1	0	0	1
d	0	0	0	0

Adjacency Matrix

$\rightarrow$  Transitive Closure Matrix :-

(Reachability matrix)

	a	b	c	d
a	1	1	1	1
b	1	0	1	1
c	0	1	1	1
d	0	0	0	0

Worshall algorithm generates a series of matrices to generate the Transitive Closure matrix

$R^{(0)}, R^{(1)}, R^{(2)}, \dots, R^{(n)}$

$R^{(0)}$  → Vertices reachable without any intermediate vertices i.e adjacency matrix

$R^{(1)}$  → Vertices reachable by using 1<sup>st</sup> node

$R^{(2)}$  → Vertices reachable using 1<sup>st</sup>, 2<sup>nd</sup> node

$R^{(n)}$  → Vertices reachable using 1<sup>st</sup>, 2<sup>nd</sup>, ..., n<sup>th</sup> node

We compute each of intermediate matrix  $R^{(k)}$  using immediate predecessor  $R^{(k-1)}$

④ i.e if for any i, j if  $r_{ij} = 1$  in  $R^{(k-1)}$  then  $r_{ij} = 1$  in  $R^{(k)}$  also

⑤

⑥ ~~if~~  $\forall i, j, r_{ij} = 1$  in  $R^{(k)}$ , if  $r_{ik}^{(k-1)} = 1$  and  $r_{kj}^{(k-1)} = 1$

	a	b	c	d
a	0	1	0	0
b	0	0	1	0
c	1	0	0	1
d	0	0	0	0

$R^{(0)}$  ⇒

	a	b	c	d
a	0	1	0	0
b	0	0	1	0
c	1	0	0	1
d	0	0	0	0

$R^{(1)}$  ⇒

	a	b	c	d
a	0	1	1	0
b	0	0	1	0
c	1	1	0	1
d	0	0	0	0

$R^{(2)}$  ⇒

	a	b	c	d
a	1	1	1	1
b	1	1	1	1
c	1	1	1	1
d	0	0	0	0

$R^{(3)}$  ⇒

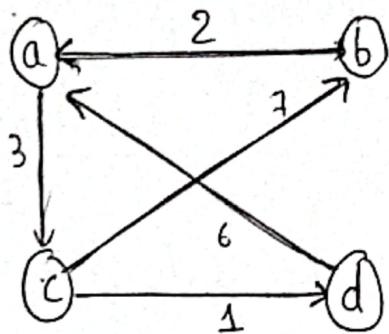
	a	b	c	d
a	1	1	1	1
b	1	1	1	1
c	1	1	1	1
d	0	0	0	0

$R^{(4)}$  ⇒

## → Floyd's Algorithm :-

- ① For finding All-Pair Shortest Path
- ② Applicable Directed / Undirected Graphs

- ③ The matrix used is called Weight matrix.



Weight Matrix

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

## → Shortest-Path Matrix :-

Distance Matrix

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

$$d_{i,j}^{(k)} = \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)})$$

	a	b	c	d
a	0	$\infty$	3	$\infty$
b	2	0	$\infty$	$\infty$
c	$\infty$	7	0	1
d	6	$\infty$	$\infty$	0

$D^{(0)}$

	a	b	c	d
a	0	$\infty$	3	$\infty$
b	2	0	(5)	60
c	$\infty$	7	0	1
d	6	$\infty$	(9)	0

$D^{(1)}$

	a	b	c	d
a	0	$\infty$	3	$\infty$
b	2	0	5	$\infty$
c	(9)	7	0	1
d	6	$\infty$	9	0

$D^{(2)}$

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

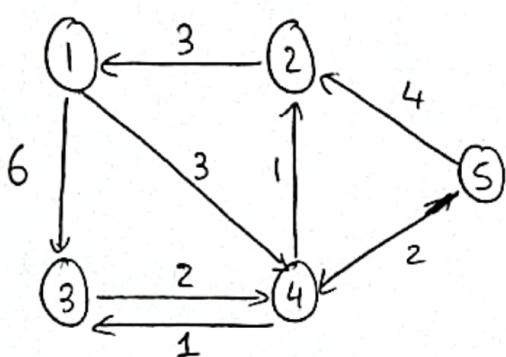
$D^{(3)}$

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

$D^{(4)}$

$\Rightarrow$  Final distance Matrix

Ex:-



Distance Matrix

	1	2	3	4	5
1	0	4	4	3	$\infty$
2	3	0	7	6	$\infty$
3	6	3	0	2	$\infty$
4	4	1	1	0	$\infty$
5	6	3	3	2	0

## Knapsack Using Dynamic Programming :-

(0-1 Knapsack)

Given,  $n \rightarrow$  items,  $W \rightarrow$  capacity,  $w_i \rightarrow$  weights of item,  $v_i \rightarrow$  values of item

$V$	0	1	2	$\dots$	$j$	$\dots$	$W$
0							
1							
?							
$i-1$							
$i$					$V[i, j]$		
:							
$n$							

Optimal Solution  
 $V[n, W]$

$$V[i, j] = V[i-1, j] \text{ when } j - w_i < 0$$

$$V[i, j] = \max \{ V[i-1, j], v_i + V[i-1, j - w_i] \} \text{ when } j - w_i \geq 0$$

Initial Condition :-  $V[i, 0] = 0 \quad \forall i \geq 0$

$$V[0, j] = 0 \quad \forall j \geq 0$$

Question:-

Item	Weight	Value
1	2	12
2	1	10
3	3	20
4	1	15

$$n = 4, W = 5$$

Sol<sup>n</sup>: -

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	12	12	12	12	12
2	0	10	12	22	22	22
3	0	10	12	22	30	32
4	0	15	25	27	37	45

Optimal Solution = 45

Items Contributing = 2, 3, 4

Subset of items = {2, 3, 4} {0 1 1 1}

(Contributing = {2, 3, 4} = desired solution)

Question-2 :-

Item	Weight	Value
1	3	25
2	2	20
3	1	15
4	4	40
5	5	50

$$n = 5 \quad W = 6$$

	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0
1	0	0	0	25	25	25	25
2	0	0	20	25	25	45	45
3	0	15	20	35	40	45	60
4	0	15	20	35	40	55	60
5	0	15	20	35	40	55	65

Optimal Solution  
= 65

Contributing Subset = {3, 5} → Composition

Items :-    1    2    3    4    5  
              0    0    1    0    1

0 → The item is not included

1 → Item is included.

$\Rightarrow$  Algorithm :-

Knapsack-bottom-up( $w[1 \dots n]$ ,  $v[1 \dots n]$ ,  $W$ )

||

for  $i \leftarrow 0$  to  $n$  do

$v[i, 0] \leftarrow 0$

for  $j \leftarrow 0$  to  $W$  do

$v[0, j] \leftarrow 0$

for  $i \leftarrow 0$  to  $n$  do

for  $j \leftarrow 0$  to  $W$  do

(condition) if  $j - w[i] \geq 0$

$v[i, j] \leftarrow \max\{v[i, j], v[i] + v[i-1, j-w]\}$

Else

$v[i, j] \leftarrow v[i-1, j]$

return  $V$  and  $V[n, W]$

→ To find composition :-

~~numofitem~~ ← 0  
numofitem ← 0  
subset[ ]

j ← w

for i ← n down to 1 do

if  $\{V[i, j] \neq V[i-1, j]\}$

numofitem ← numofitem + 1

subset[~~numofitem~~] ← i

j ← j - W[i]

end for

Note :- In Knapsack using Bottom-Up Approach (Iterative)

③ All the subproblems are solved and solved only once

But, all the solutions of subproblems do not contribute for getting solution for given problem.

→ In Knapsack using Top-Down Approach (Recursive) :-

④ All the subproblems are solved and solved more than once

But, all the solutions of subproblems contribute for getting solution for given problem.

→ Knapsack using Memory Functions :-

MFKnapsack(i, j)

|| Input :- Integer i, which is the number of items  
Integer j, which indicates the Knapsack capacity

|| Output :- The value of an optimal solution<sup>subset</sup> of the first 'i' items.

if  $v[i, j] = -1$

if  $j - w[i] \geq 0$

value  $\leftarrow \max\{MFKnapsack(i+1, j), v[i] + MFKnapsack(i-1, j-w[i])\}$

Else

value  $\leftarrow MFKnapsack(i-1, j)$

$v[i, j] \leftarrow value$

return  $v[i, j]$

Question :-

Item	Weight	Value
1	2	12
2	1	10
3	3	20
4	2	15

$$W = 5$$

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	12	+12	12	12
2	0	-1	12	22	-1	22
3	0	-1	-1	22	-1	32
4	0	-1	-1	-1	-1	37

(37)  
 $MFK(4, 5)$

$$\text{Max} \{ MFK(3, 5), 15 + MFK(3, 3) \} \quad (22)$$

$$\text{Max} \{ MFK(2, 5), 20 + MFK(2, 2) \} \quad (22)$$

$$\text{Max} \{ MFK(2, 3), 20 + MFK(2, 0) \} \quad (0)$$

$$\text{Max} \{ MFK(1, 5), 10 + MFK(1, 4) \} \quad (12)$$

$$\text{Max} \{ MFK(1, 2), 10 + MFK(1, 0) \}$$

$$\text{Max} \{ MFK(1, 3), 10 + MFK(1, 2) \} \quad (12)$$

$$\text{Max} \{ MFK(0, 4), 12 + MFK(0, 2) \}$$

$$\text{Max} \{ MFK(0, 3), 12 + MFK(0, 1) \}$$

$$\text{Max} \{ MFK(0, 5), 12 + MFK(0, 3) \}$$

$$0 \quad 12 + 0$$

$$0, 12 + 0$$

$$\text{Max} \{ MFK(0, 3), 12 + MFK(0, 1) \}$$

$\Rightarrow$  Greedy Technique :-

The solution must be

Feasible

Optimal

Irrevocable

$\rightarrow$  Change Making Problem :-

Given some coins, and an amount , select the least number of coins to make the amount.

Ex:-  $d_1 = 25$  (quonta)  $d_2 = 10$  (dime)  $d_3 = 5$  (nickle)  $d_4 = \text{penny}$  (1)

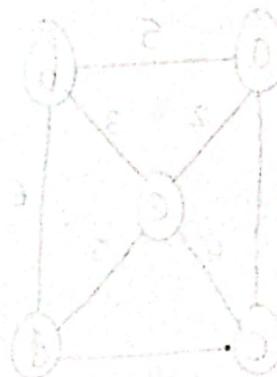
Amount : 48 cents.

$$d_1 \rightarrow 1 \Rightarrow 48 - 25 = 23$$

$$d_2 \rightarrow 2 \Rightarrow 23 - 20 = 3$$

$$d_4 \rightarrow 3 \Rightarrow 3 - 3 \times 1 = 0$$

Total :- 6 coins



Ex 2:- Amount :- 10 dollars

Denomination :-  $d_1 \rightarrow 7$   $d_2 \rightarrow 5$   $d_3 \rightarrow 1$

For this instance, greedy technique does not provide optimal solution

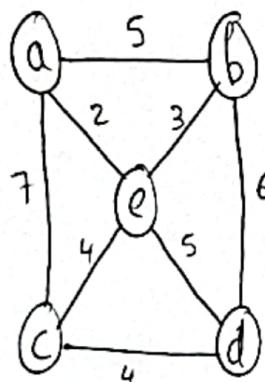
i.e.  $d_1 \rightarrow 1$  &  $d_2 \rightarrow 3 \Rightarrow 4$  coins

But, optimal solution is

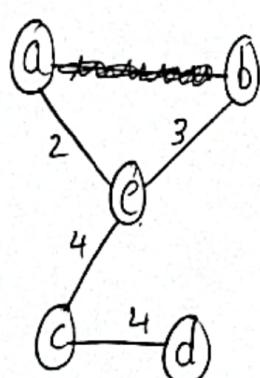
$d_2 \rightarrow 2 \Rightarrow 2$  coins

Prim's Algorithm to find Minimum Cost Spanning Tree:-

Example:-



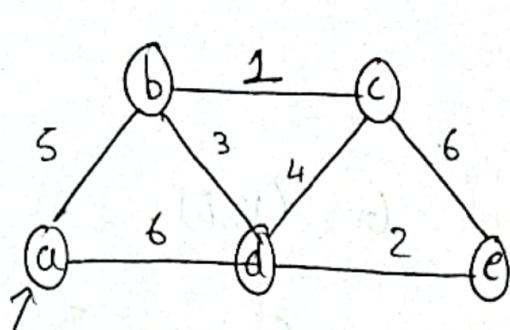
Tree Vertices	Remaining Vertices (Priority Queue)
a(-, -)	b(a, 5) e(a, 2) c(a, 7) d(-, ∞)
e(a, 2)	b(a, 5) <del>e(b, 3)</del> c(a, 7) <del>c(e, 4)</del> d(e, 5)
b(e, 3)	c(a, 7) <del>c(e, 4)</del> d(e, 5) d(b, 6)
c(e, 4)	<del>d(c, 4)</del> d(e, 5) d(b, 6)
d(c, 4)	



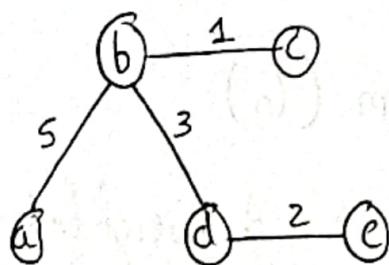
$$\Rightarrow \{ (a,e) (e,c) (c,d) (e,b) \}$$

Edge Set forming  
Minimum Spanning Tree.

Example :- 2 :-



=>



Minimum Spanning Tree

Tree Vertices

Remaining Vertices (Priority Queue)

a(-, ∞) b(a, 5), d(a, 6), c(-, ∞), e(-, ∞)

b(a, 5) c(b, 1), d(b, 3), d(a, 6), e(-, ∞)

c(b, 1) d(b, 3), d(a, 6), d(c, 4), e(c, 6)

d(b, 3) e(d, 2), e(c, 6)

e(d, 2)

Minimum Spanning Tree Edge Set

$$= \{ (a, b), (b, c), (b, d), (d, e) \}$$

$\Rightarrow$  Algorithm :-

Prim (G)

|| I/P :- A weighted connected graph  $G = (V, E)$

|| O/P :- Set of edges composing a Minimum Spanning Tree of G

$$V_T \leftarrow \{V_0\}$$

$$E_T \leftarrow \emptyset$$

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  $V - V_T$

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

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

end for

return  $E_T$

$\Rightarrow$  Kruskal's Algorithm :-

Kruskal( $G_i$ )

|| I/P :- A weighted connected graph,  $G_i = (V, E)$

|| O/P :-  $E_T$ , the set of edges composing of Minimum Spanning Tree of  $G_i$

Sort Edges in Ascending order of their weights.

$$w(e_1) \leq \dots \leq w(e_{|E|})$$

$$E_T \leftarrow \emptyset$$

$$e\text{counter} \leftarrow 0$$

$$K \leftarrow 0$$

while  $e\text{counter} < |V| - 1$  do

$$K \leftarrow K + 1$$

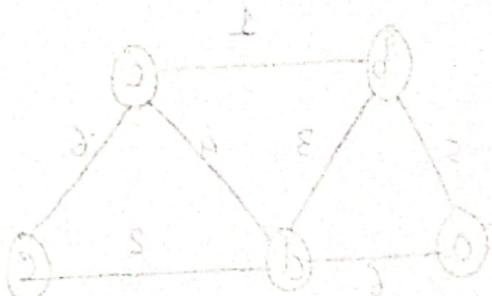
if  $E_T \cup \{e_{ik}\}$  is acyclic

$$E_T \leftarrow E_T \cup \{e_{ik}\}$$

$$e\text{counter} \leftarrow e\text{counter} + 1$$

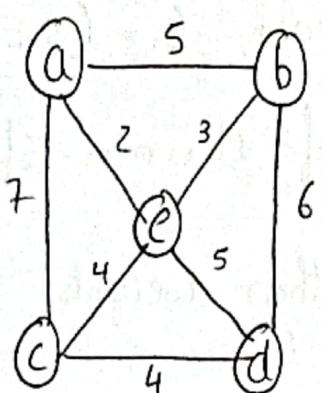
end while

return  $E_T$

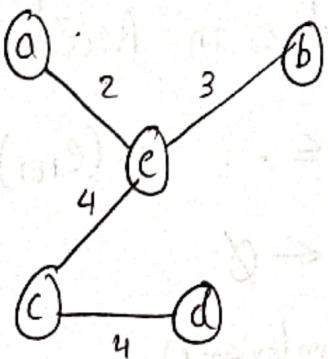


$\Rightarrow$  Kruskal's Algorithm algorithm Examples :-

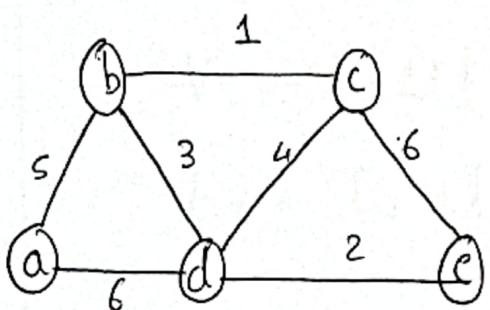
Ex:- ①



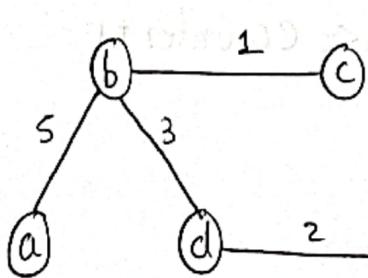
ae	be	ec	cd	ab	ed	bd	ac
2 ✓	3 ✓	4 ✓	5 ✓	5 x	5 x	6 x	7 y



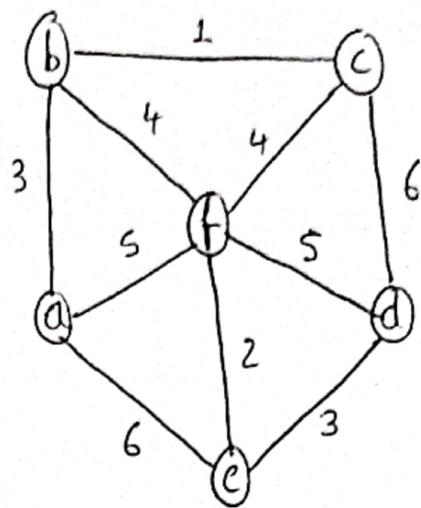
Ex:- ②



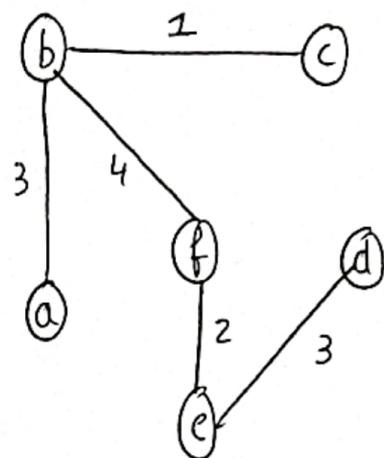
bc	de	bd	dc	ab	ad	ce
1 ✓	2 ✓	3 ✓	4 x	5 ✓	6 x	6 x



Ex - 3 :-



bc	fe	de	ab	bf	cf	af	fd	cd	ae
1	2	3	3	4	4	5	5	6	6
✓	✓	✓	✓	✓	x	x	x	x	x

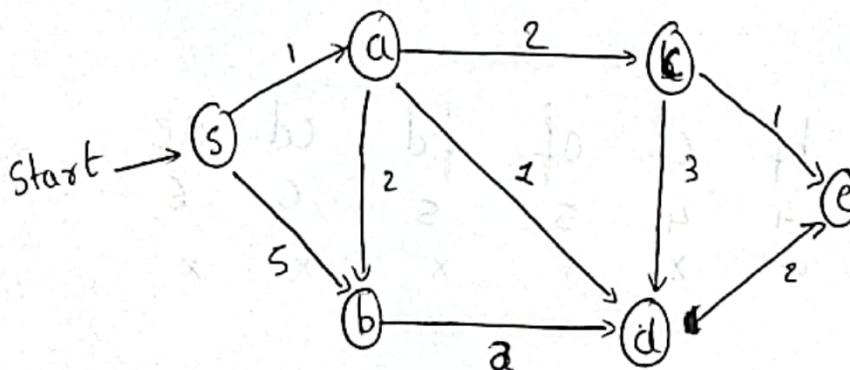


$\Rightarrow$  Dijkstra's Algorithm :- (Single Source shortest path)

① Applicable for both directed and undirected.

② Works only for positive weights.

Example :-



Initially :-

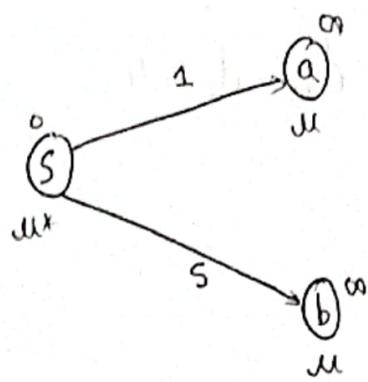
Priority Queue,

	s	a	b	c	d	e
Distance : $d_v$	0	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$
Parent : $p_v$	-	-	-	-	-	-

Unvisited Vertex Set ,  $U_T = \{s, a, b, c, d, e\}$

Visited . . . Visited Vertex Set

$V_T = \{\emptyset\}$



if  $(d_{u^*} + w(u^*, u)) < d_u$   
 $d_u \leftarrow d_{u^*} + w(u^*, u)$

For a,

$$0 + 1 < \infty \Rightarrow d_u = 1$$

For b,

$$0 + 5 < \infty \Rightarrow d_u = 5$$

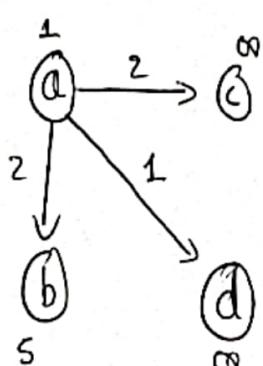
P. Queue:-

	s	a	b	c	d	e
$d_v$	0	1	5	$\infty$	$\infty$	$\infty$
$p_v$	-	s	s	-	-	-

$$S = \{b\} \subset S \cup S + E$$

$$U_T = \{a, b, c, d, e\}$$

$$V_T = \{s\}$$



For c,

$$1 + \infty < \infty \Rightarrow d_u = 3 = d_c$$

For d,

$$1 + 1 < \infty \Rightarrow d_u = 2 = d_d$$

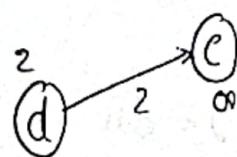
$$P = \{b\} \subset S \cup S + E$$

$$1 + 2 < s \Rightarrow d_u = d_b = 3$$

	s	a	b	c	d	e
$d_v$	0	1	3	3	2	$\infty$
$p_v$	-	s	a	a	a	-

$$U_T = \{b, c, d, e\}$$

$$V_T = \{s, a\}$$



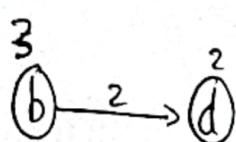
$$2+2 < \infty \Rightarrow d_u = d_e = 4$$

PO:

	s	a	b	c	d	e
$d_v$	0	1	3	3	2	4
$P_v$	-	s a a a a d				

$$U_T = \{b, c, e\}$$

$$V_T = \{s, a, d\}$$

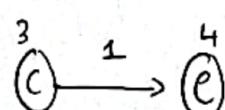
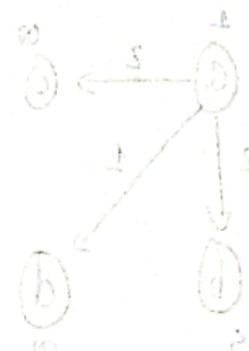


$$3+2 < 2 \Rightarrow d_u = d_d = 2$$

PO:

	s	a	b	c	d	e
$d_v$	0	1	3	3	2	4
$P_v$	-	s a a a d				

$$U_T = \{b, c, e\}$$



$$3+1 < 4 \Rightarrow d_u = d_e = 4$$

PO:

	s	a	b	c	d	e
$d_v$	0	1	3	3	2	4
$P_v$	-	s a a a d				

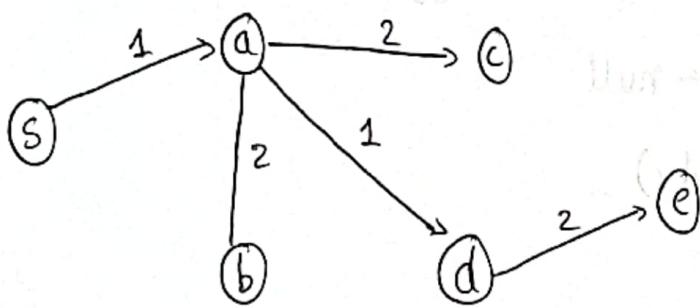
$$U_T = \{e\} \quad V_T = \{s, a, d, b, c\}$$

No adjacent vertex from e :-

	s	a	b	c	d	e
$\therefore PA :-$	$d_v$	0	1	3	3	2

$P_v$	-	s	a	a	a	d
-------	---	---	---	---	---	---

$$U_T = \{\emptyset\} \quad V_T = \{s, a, d, b, c, e\}$$



$$s-a = 1$$

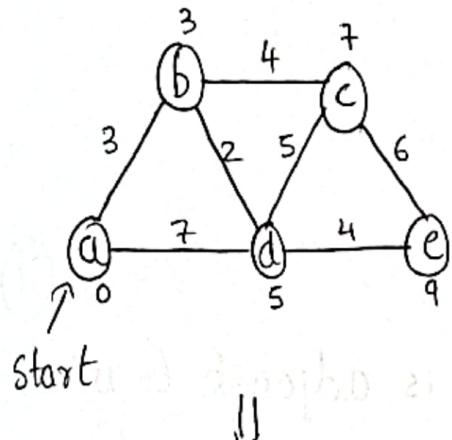
$$s-b = 3$$

$$s-c = 3$$

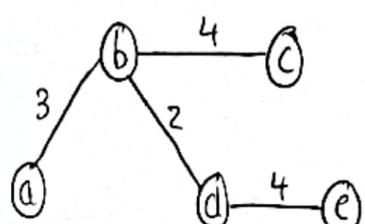
$$s-d = 2$$

$$s-e = 4$$

$\Rightarrow$  Another Method :-



Vertices Explored	Remaining Vertices (Priority Queue)
$a(-, 0)$	$b(a, 3), d(a, 7)$
$b(a, 3)$	$(b, 3+4), d(b, 2+3), d(b, 7)$
$d(b, 5)$	$c(b, 7), c(d, 5+5), e(d, 5+4)$
$c(b, 7)$	$e(d, 9), e(c, 7+6)$
$e(d, 9)$	



Answer :-

$$a-b = 3$$

$$a-c = 7 \text{ (Through } b\text{)}$$

$$a-d = 5 \text{ (Through } b\text{)}$$

$$a-e = 9 \text{ (Through } b, d\text{)}$$

Algorithm :-

Dijkstra( $G, s$ )

// G I/P :-  $G$  is a connected, weighted @ undirected graph and starting vertex  $s$ .

// O/P :-

for every vertex  $v$  in  $V$  do

$d_v \leftarrow \infty$ ,  $p_v \leftarrow \text{null}$

Insert  $(\emptyset, v, d_v)$

$d_s \leftarrow 0$

Decrease( $\emptyset, s, d_s$ )

$V_I \leftarrow \emptyset$

for  $i \leftarrow 0$  to  $|V|-1$  do

$u^* \leftarrow \text{DeleteMin}(\emptyset)$

$V_I \leftarrow V_I \cup \{u^*\}$

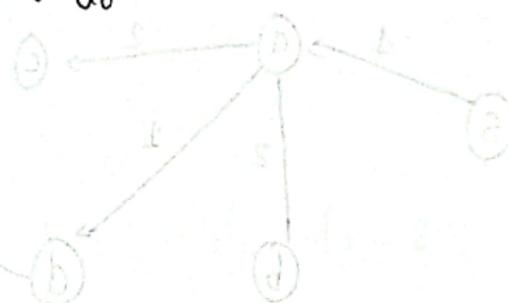
for every vertex  $u$  in  $V - V_I$  that is adjacent to  $u^*$  do

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

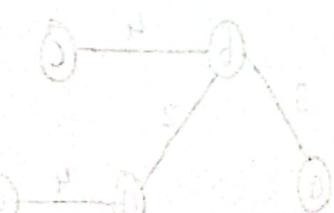
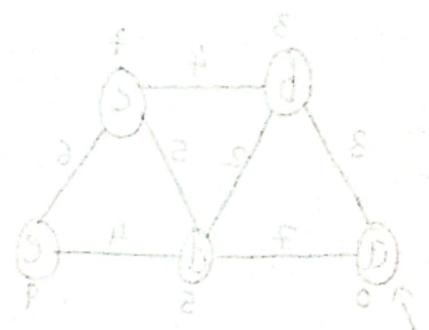
$d_u \leftarrow d_{u^*} + w(u^*, u)$

$p_u \leftarrow u^*$

Decrease( $\emptyset, u, d_u$ )



→ bottom right



## Huffman Coding :-

⊗ It is a greedy lossless data compression algorithm

→ Two types of Encoding :-

i) Fixed length Encoding

ii) Variable length Encoding

⊗ Huffman Coding uses variable length encoding

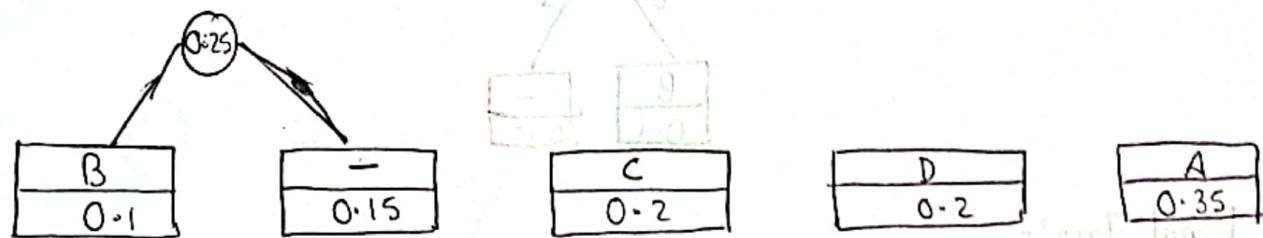
Ex:-

Given, set of 5 alphabets {A, B, C, D, -} :-

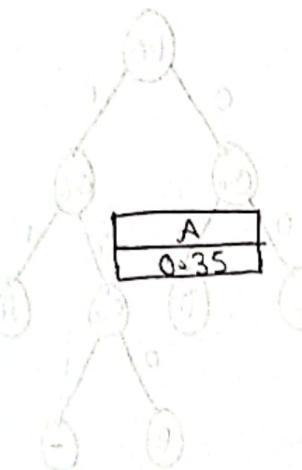
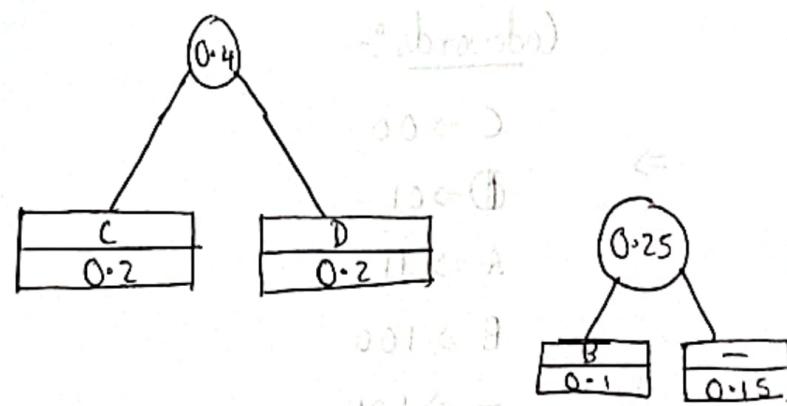
Characters	A	B	C	D	-
Probability	0.35	0.1	0.2	0.2	0.15

Encode :-

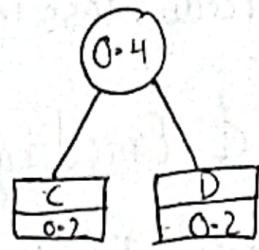
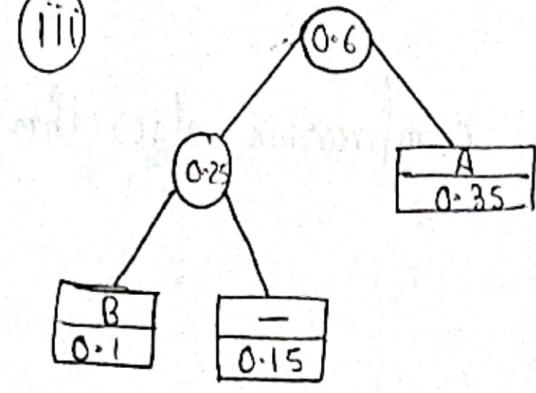
BAD-AC



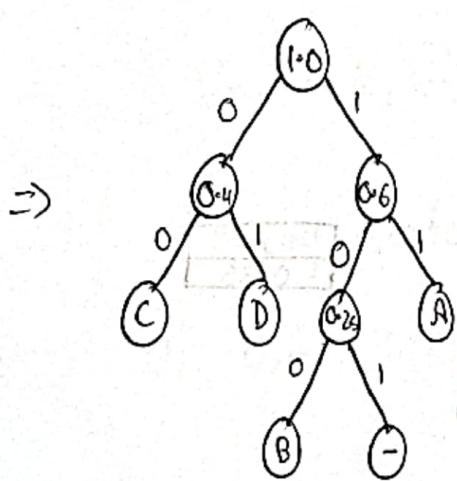
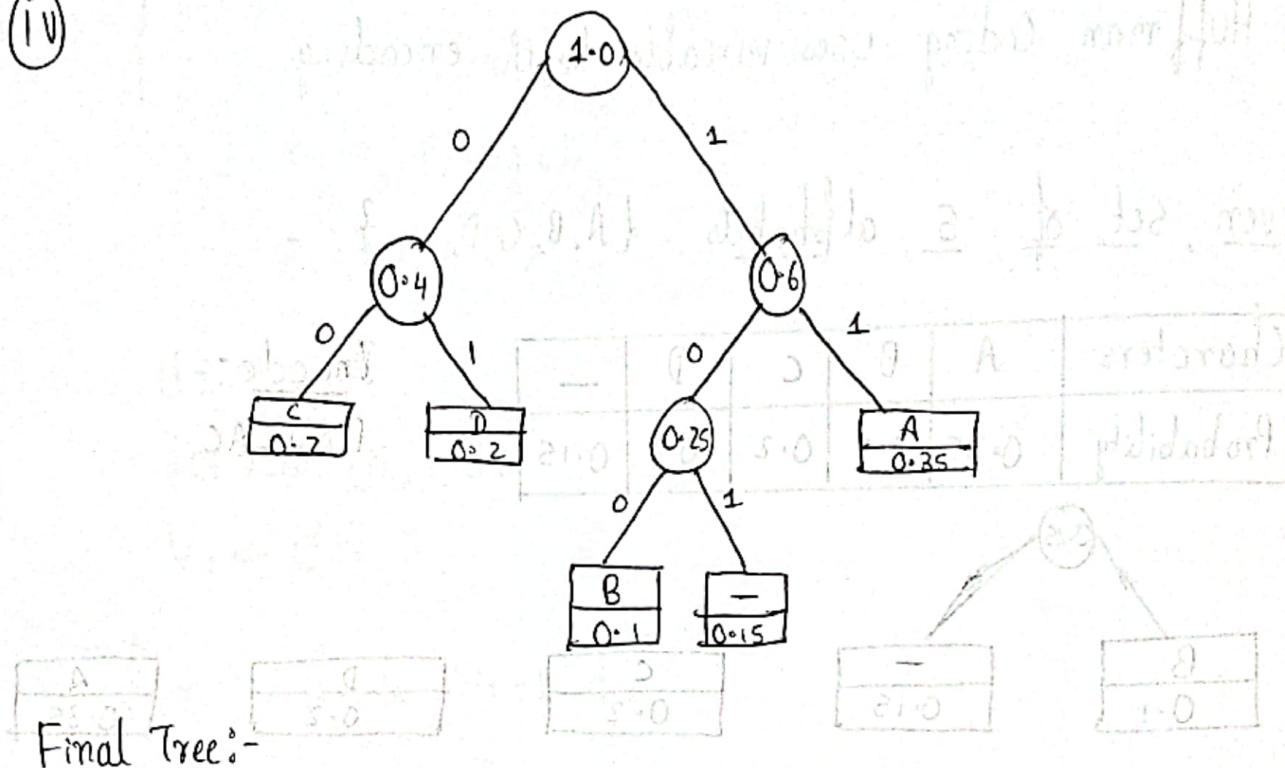
(ii)



(III)



(IV)

Codewords :-

$$C \Rightarrow 00$$

$$D \Rightarrow 01$$

$$A \Rightarrow 11$$

$$B \Rightarrow 100$$

$$- \Rightarrow 101$$

$$\Rightarrow BAD\_AC = 1001101101100$$

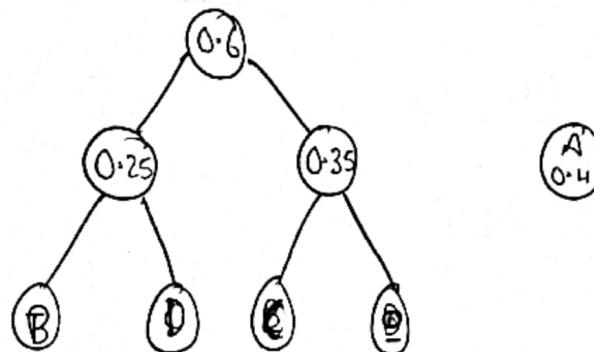
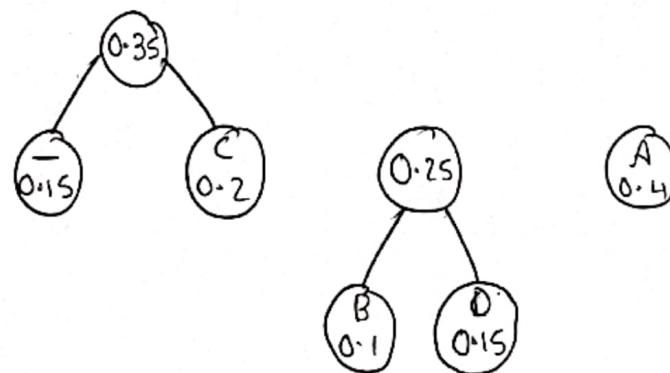
Q) Construct a Huffman code for following Data :-

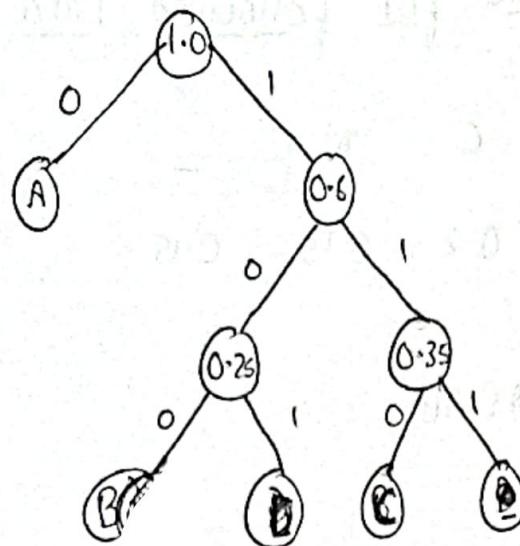
Characters	A	B	C	D	-
Probability	0.4	0.1	0.2	0.15	0.15

Encode the text ABACABAB

Decode 10010111001010

Soln:-





$A \Rightarrow 0 \quad B = 110 \quad C \Rightarrow 101 \quad - \Rightarrow 100 \quad D \Rightarrow 111$

