Unit-3

Dynamic Programming

① Fibonacci Series

optimal substructures

overlapping subproblems

Recursive solution

```
fib(n)
    if n <= 1
        return 1

    else
        return fib(n-1) + fib(n-2)
```

Top-down approach - Memoization - storing already computed values in a list, w/o recomputing

```
fib(n, memo)
    if n <= 1
        return 1
    else if memo[n] is computed
        return memo[n]

    else
        result = fib(n-1, memo) + fib(n-2, memo)
        memo[n] = result
        return result
```

Bottom-Up Approach - maintain a table of values

```
fib(n) {
    fib[0] = 1
    fib[1] = 1
    for i ← 2 to n
        fib[i] = fib[i-1] + fib[i-2]
```

Time complexity reduces from $O(2^n)$ to $O(n)$

* Computing Binomial Coefficient

Aim: To find the coefficient of x^k in the expansion $(1+x)^n$

Recurrence: $C(n,k) = C(n-1, k-1) + C(n-1, k)$

Algorithm

```
binomial(n,k)
for i = 0 to n
    for j = 0 to min(i,k) do
        if j == 0 or j == i
            C[i,j] ← 1
        else
            C[i,j] ← C[i-1, j-1] + C[i-1, j]
return C[n,k]
```

Time Complexity: $O(nk)$

Example

1. $n=4 \quad k=2$

$C(n,k) = C(n-1,k-1) + C(n-1,k)$

n/k	0	1	2
0	1	X	
1	1	1	X
2	1	2	1
3	1	3	3
4	1	4	6

~~$C(0,1) = C$~~

$C(2,1) = C(1,0) + C(1,1)$

$= 1+1 = 2$

$C(3,1) = C(2,0) + C(2,1)$

$= 1+2 = 3$

$C(3,2) = C(2,1) + C(2,2)$

$= 2+1 = 3$

$C(4,1) = C(3,0) + C(3,1)$

$= 1+3 = 4$

$C(4,2) = C(3,1) + C(3,2)$

$= 3+3$

$= 6$

2. $n=5 \quad k=2$

n/k	0	1	2
0	1	X	
1	1	1	X
2	1	2	1
3	1	3	3
4	1	4	6
5	1	5	10

$C(5,1) = C(4,0) + C(4,1)$

$= 1+4 = 5$

$C(5,2) = C(4,1) + C(4,2)$

$= 4+6 = 10$

* Ordering of matrices - In matrix chain multiplication

→ Given a sequence of matrices to be multiplied, compute the product by minimizing the no. of basic multiplications

→ Give appropriate parenthesization

Recurrence

$$m[i,j] = \begin{cases} 0 & , \ i = j \\ \min\left(m[i,k] + m[k+1,j] + p_{i-1} p_k p_j\right) & , \ i < j \end{cases}$$

Algorithm

1. n = length of chain - 1
2. m & s are 2 tables
3. for i = 0 to n
   m[i,i] = 0
4. for l = 2 to n
   for i = 1 to n - l + 1
   j = i - l + 1
   m[i,j] = ∞
   For k = i to j - 1
   q = m[i,k] + m[k+1,j] + p(i-1) p(k) p(j)
   if q < m[i,j]
   m[i,j] = q s[i,j] = k

Ex 1

	0	1	2	3
	A_1	A_2	A_3	A_4
	2×3	3×6	6×4	4×5

	0	1	2	3
0	0	36	84	124
1		0	72	132
2			0	120
3				0

2 matrices

$m(0,1) = 2 \times 3 \times 6 = 36$

$m(1,2) = 3 \times 6 \times 4 = 72$

$m(2,3) = 64 \times 5 = 120$

3 matrices

(i) $m(0,2) = A_1 \cdot (A_2 \cdot A_3)$

$= 72 + (2 \times 3 \times 4) = \boxed{96}$

(ii) $= (A_1 \cdot A_2) \cdot A_3$

$= 36 + (2 \times 6 \times 4)$

$= 36 + 48 = \boxed{84}$ min

36
48
84

(i) $m(1,3)$

(i) $A_2 \cdot (A_3 \cdot A_4)$

$120 + (3 \times 6 \times 5) = 210$

(ii) $(A_2 \cdot A_3) \cdot A_4$

$= 72 + (3 \times 4 \times 5)$

$= 72 + 60$

$= 132$

Brackets

$((A_1 A_2) A_3)(A_4)$

4 matrices (i) $A_1 \cdot (A_2 \cdot A_3 \cdot A_4)$

$132 + (2 \times 3 \times 5) = \underline{162}$

(ii) $(A_1 \cdot A_2 \cdot A_3) \cdot A_4$

$84 + (2 \times 4 \times 5) = \underline{124}$

(iii) $(A_1 \cdot A_2) \cdot (A_3 \cdot A_4)$

$36 + 120 + (2 \times 6 \times 5)$

$= 36 + 120 + 60 = \underline{216}$

Ex 2

0	1	2	3	4
A_1	A_2	A_3	A_4	A_5
3×2	2×6	6×4	4×5	5×2

	0	1	2	3	4
0	0	36	72		
1		0	48	88	
2			0	120	
3				0	40
4					0

2 matrices

(i) $3\times2\times6$

(ii) $2\times6\times4$

(iii) $6\times4\times5$

(iv) $4\times5\times2$

3 matrices

$m(0,2) = A_1(A_2\ A_3)$

$48 + (3\times2\times4)$

$= 72$

or

$(A_1\ A_2).A_3$

$36 + (3\times6\times4)$

$36 + 72 = 108$

$m(1,3) = A_2(A_3.A_4)$

$= 120 + (6\times5\times2)$

or $= 120 + 60 = 180$

$(A_2.A_3).A_4$

$= 48 + (2\times4\times5)$

$= 48 + 40 = 88$

$m(2,4) = A_3.(A_4.A_5)$

$= 40 + (4\times2\times6) = 40 + 48 = 88$

or $(A_3.A_4).A_5 = 120 + (6\times5\times2) = 120 + 60 = 180$

4 matrices

m(0,3) $A_1 A_2 A_3 A_4$

(i) $A_1 \cdot (A_2 \cdot A_3 \cdot A_4)$

(ii) $(A_1 \cdot A_2 \cdot A_3) \cdot A_4$

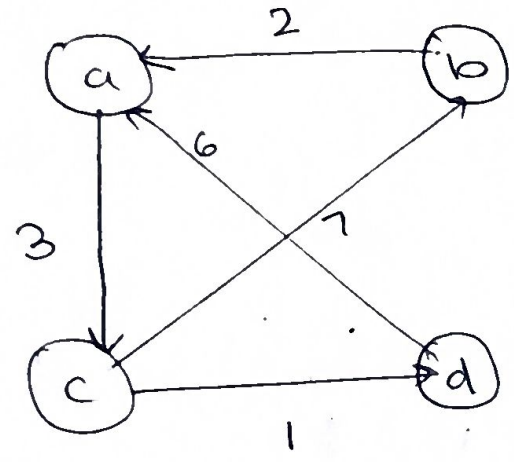
(iii) $(A_1 \cdot A_2)(A_3 A_4)$

* Floyd-Warshall Algorithm

recurrence: $d_{ij} = \min (d(i,j), d(i,k) + d(k,j))$

Time Complexity: $O(n^3)$

Example:



A_0 =

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

A_1 =

	a	b	c	d
a	0	∞	3	∞
b	2	0	5	∞
c	∞	∞	0	∞
d	6	∞	9	0

$A[cd] = A[ca] + A[ad]$

$= \infty + \infty$

$A[cb] = A[ca] + A[ab]$

$= \infty + \infty$

$A[bc] = A[ba] + A[ac]$

$= 2 + 3 = 5 < \infty$

$A[db] = A[da] + A[ab]$

$= 6 + \infty$

$A[ba] = A[ba] + A[ad]$

$= 2 + \infty = \infty$

$A[dc] = A[da] + A[ac]$

$= 6 + 3$

do 3 more time

* Dynamic Programming - Knapsack Problem

$$F(i,j) = \begin{cases} \max \{ F(i-1,j),\ v_i + F(i-1, j-w_i) \} & j - w_i \geq 0 \\ F(i-1,j) & j - w_i < 0 \end{cases}$$

Time Complexity: Pseudo-polynomial in nature

depends on value of input

if w is a polynomial fn. of n $\Rightarrow$ a polynomial time algo

depending on the value of w $\Rightarrow$ could be $O(2^n)$

np-complete.

Example 1 $w = 8$

$v = \{1, 2, 5, 6\}$

$w = \{2, 3, 4, 5\}$

v_i	w_i	capacity 0	1	2	3	4	5	6	7	8
1	2	0	0	1	1	1	1	1	1	1
2	3	0	0	1	2	2	3	3	3	3
5	4	0	0	1	2	5	5	6	7	7
6	5	0	0	1	2	5	6	6	7	(8)

Ex 2

$v = \{8, 10, 12, 30\}$

$w = \{2, 2, 2, 8\}$

$W = 12$

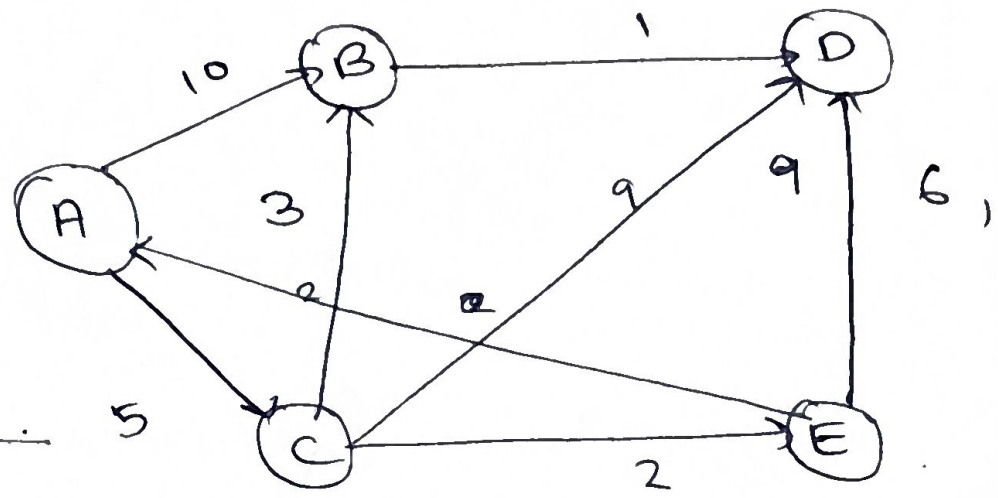
capacity

vi	wi	0	1	2	3	4	5	6	7	8	9	10	11	12
8	2	0	0	8	8	8	8	8	8	8	8	8	8	8
10	2	0	0	10	10	18	18	18	18	18	18	18	18	18
12	2	0	0	12	12	22	22	30	30	30	30	30	30	30
30	8	0	0	12	12	22	22	30	30	30	30 42	42	42	52

* Dijkstra's Algorithm

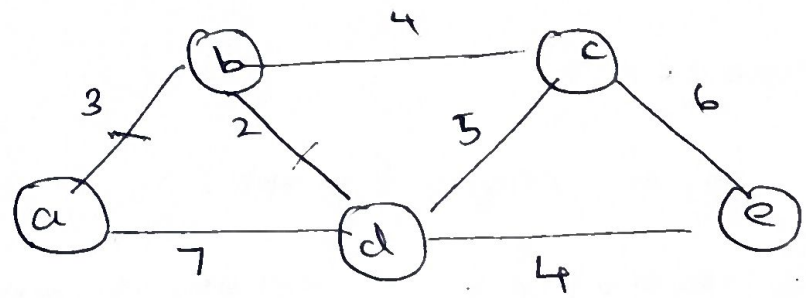
Time Complexity : $O(|V|^2)$

Ex1



	A	B	C	D	E
A	[0]	∞	∞	∞	∞.
C		10	[5]	∞	∞
E		8		14	[7]
B		[8]		13	
D				[9]	

Ex2



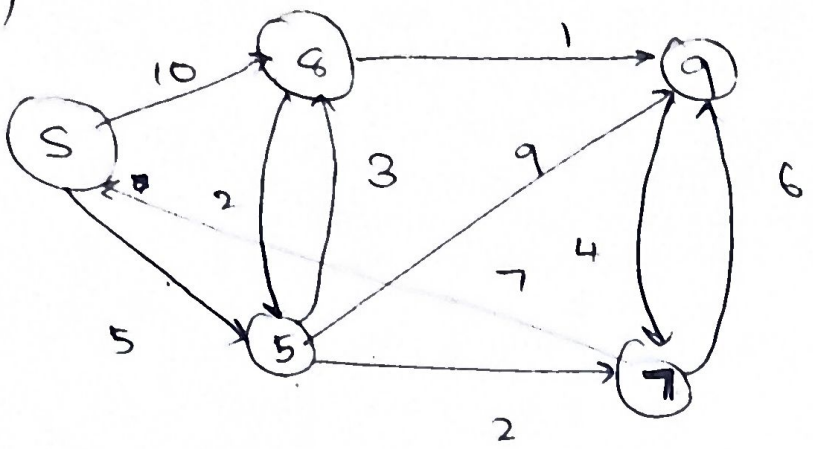
	a	b	c	d	e
a	[0]	∞	∞	∞	∞
bd			4	[2]	
b		[4]	7		6
e			7		[6]
c			[7]		

a → b → d → e

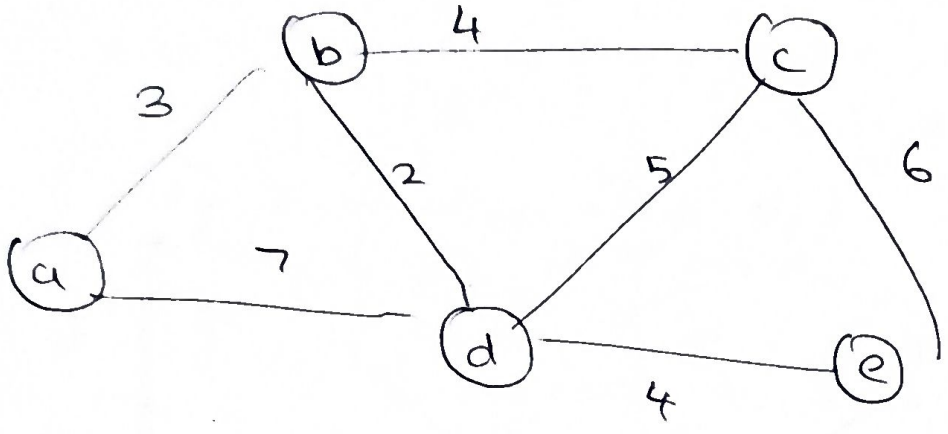
e d a

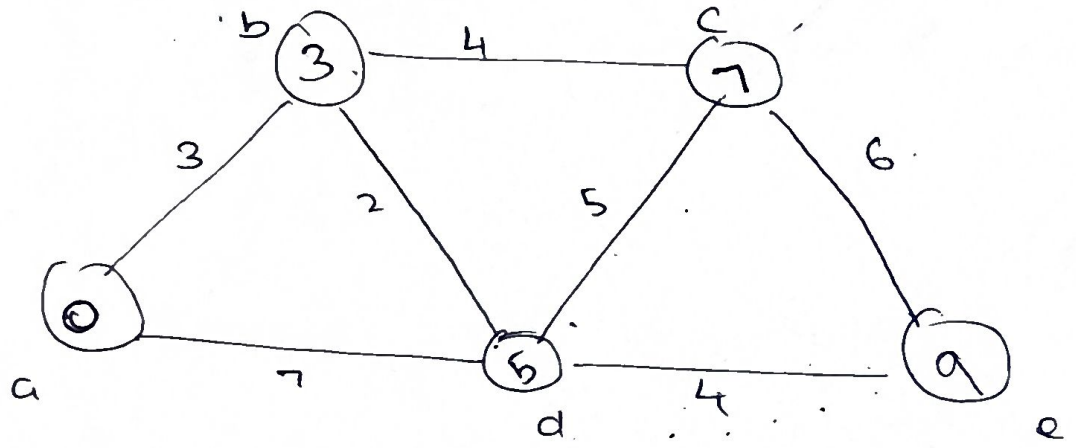
(12)

(1)



(2)





Algorithm:

n ← length (graph)

initialize an empty distance matrix

set start vertex's dist to 0

create a visited list, set to 0

iterate to n

set distance to infinity

min_dist_vertex ← 1

```
for j in range n
    if not visited and distances[j] < min_dist
        min_dist = distance[j]
        min_dist_vertex ← j
visited[min_dist_vertex] ← 1

for j in range (n)
    if not visited[j] and graph[min_dist_vertex][j]
                                                > 0 :
        newdist ← distance[min_dist_vertex] + 1
                  graph[min_dist_vertex][j]
        if newdist < distance[j]
            assign
return distances
```

Complexity : $O(n^2)$

* Prim's Algorithm for MST

Algorithm

$V_T = \{\}$

$E_T = \phi$

for i ← 1 to no. of vertices - 1

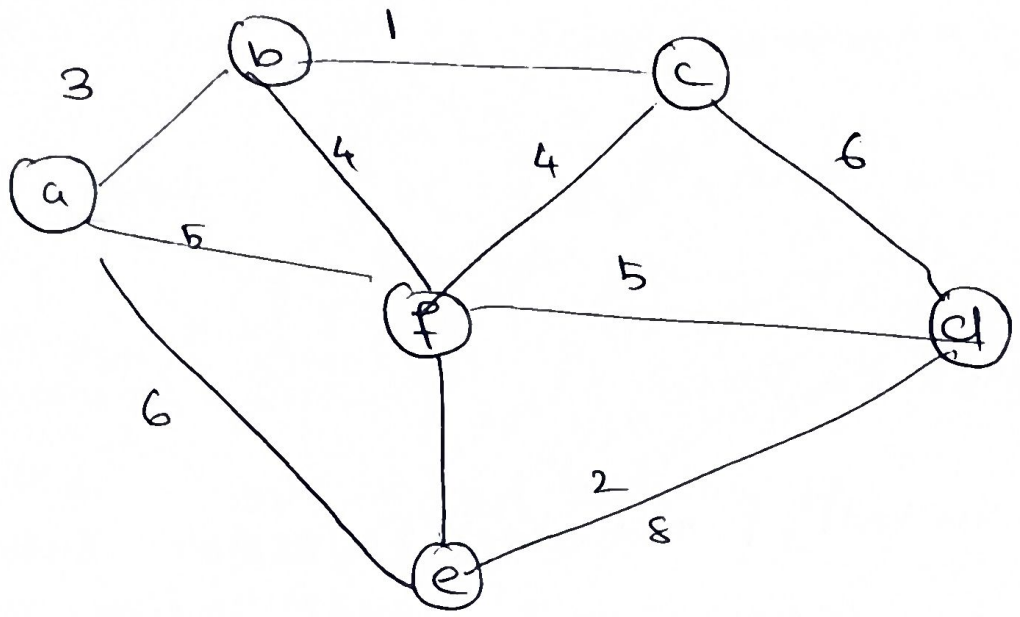
find a minimum edge (v^*, u^*) such that v is in V_T and u is in $V - V_T$

$V_T \leftarrow V_T \cup \{u_*\}$

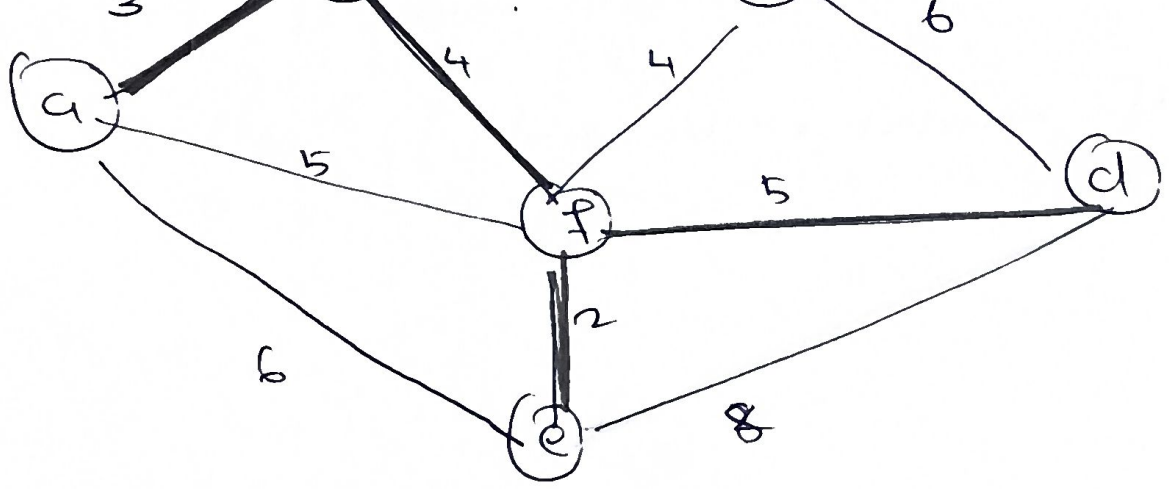
$E_T \leftarrow E_T \cup \{e_*\}$

return E_T

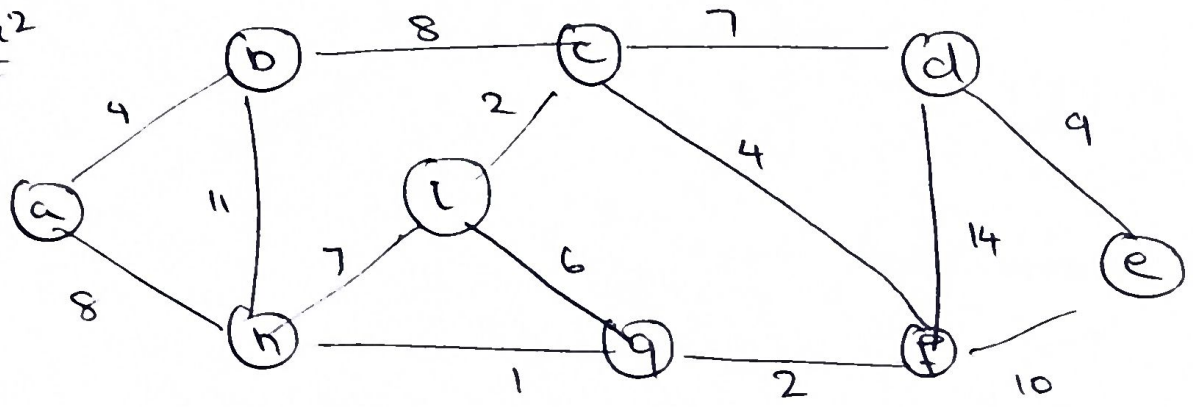
Ex

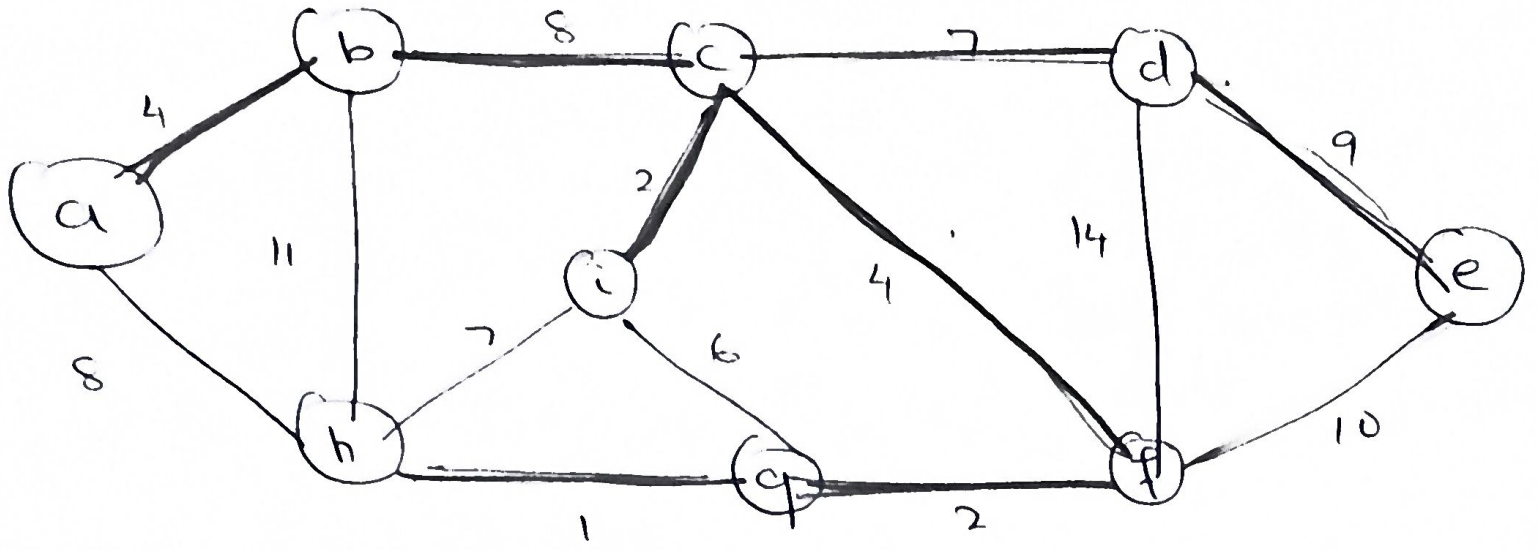


V



Ex2





Time Complexity : $O(E \log V)$

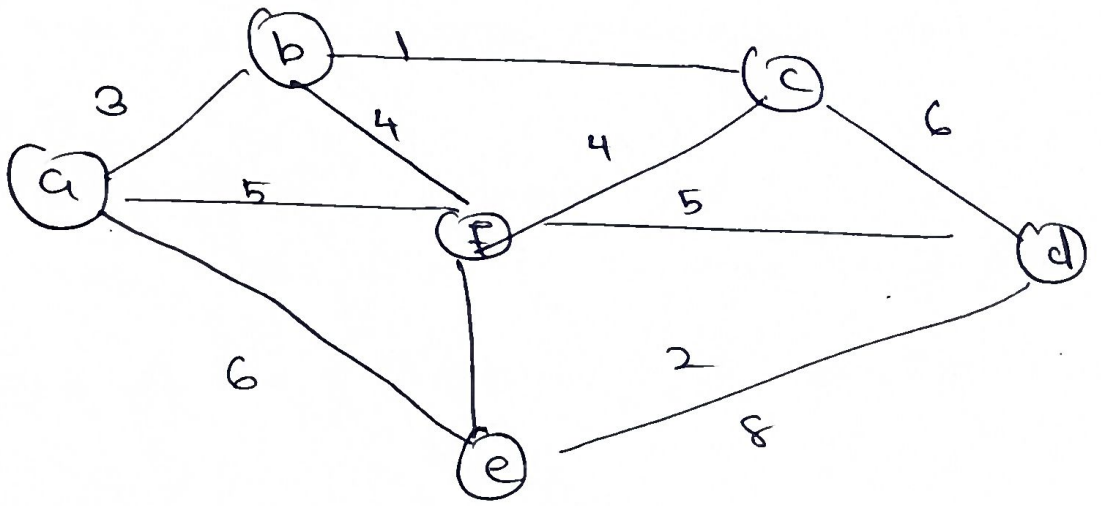
* Kruskal's Algorithm

to find the MST, w/ disjoint sets — $O(E \log E)$

sort wts in non-decreasing order

add edges in that order as long as there is no cycle

Ex.



Ans =