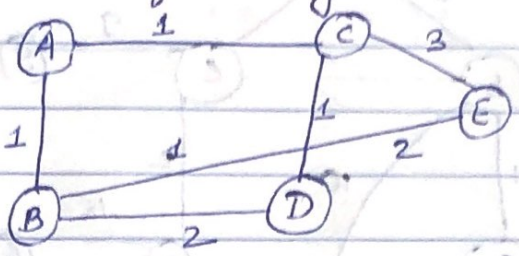
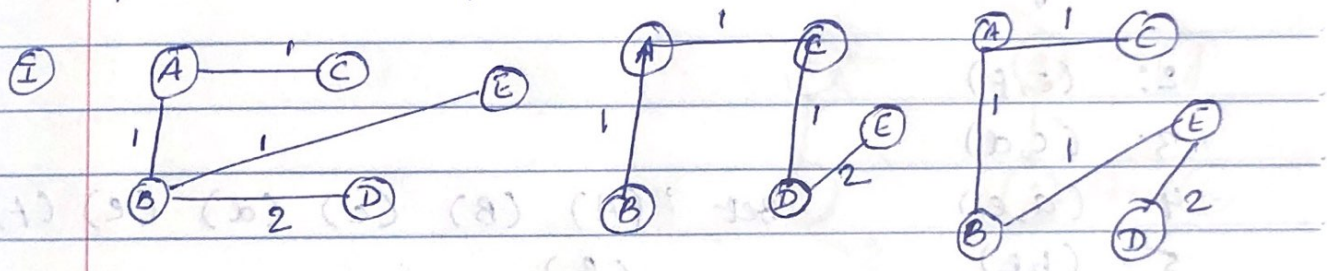
1. Show that there can be more than one minimum spanning tree in an undirected graph.

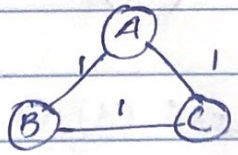
Soln: Consider the following Undirected graph:



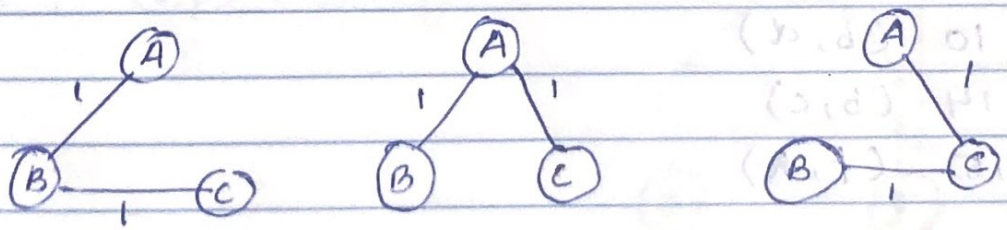
NOW, a minimum spanning tree is a spanning tree with the minimum weight. Therefore possible MSTs possible are:



Another Example:



Possible MSTs:



∴ we can have multiple MSTs for an undirected graph.

Ans-2. BFS (Breadth First Search) Algorithm traditionally cannot find all the connected components in a disconnected graph. In order to extend the BFS algo to find all the connected component in an undirected graph, we can loop in all the vertices that are unvisited and then perform BFS on these vertices.

Note: visited [i] = gray, unvisited (i) = white and visited(i) and explored (i) = Red.

```
for each vertex u do
    if colors [U] = white
    {
        BFS (G, u, white, dist, parent);
    }
for end
```

BFS

```
procedure BFS (G: graph; s: node; var
                color: carray; dist: iarray;
                parent: parray);
```

```
for each vertex u do
    color[u] = white;   dist[u] = ∞;
        parent[u] = nil;
end for

color[s] = gray;   dist[s] = 0;
init[Q];    enqueue(Q, s);

while not (empty(Q)) do
    u = head(Q);
    for each v in adj(u) do
        if color[v] = white then
            color[v] = gray;
            dist[v] = dist[u] + 1;
            parent[v] = u;
            enqueue(Q, v);
dequeue(Q);
colour[u] = Red;
print "u";

end BFS
```

3.

To prove: That a complete undirected graph G with n vertices has $n(n-1)/2$ edges.

Proof by Induction :

Base Case : For $n=1$, no. of edges $= 0$, $\frac{1(1-1)}{2} = 0$.

Hypothesis : Assume, that a complete graph with n vertices has $n(n-1)/2$ edges.

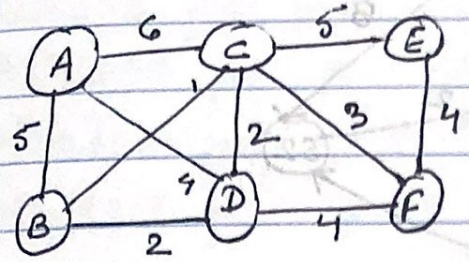
Induction : If we add $(n+1)^{st}$ vertex, we need to connect it to n original vertices, requiring n additional edges.

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

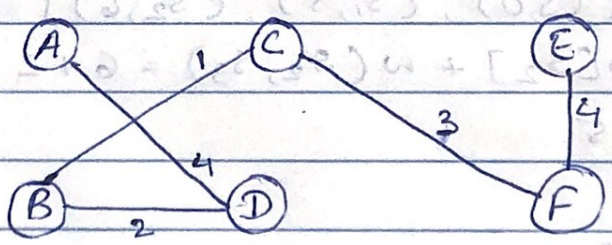
$$= \frac{(n+1)\,((n+1)-1)}{2} \text{ edges for } n+1 \text{ vertices}$$

Hence proved.

4. Prim's Algorithm:

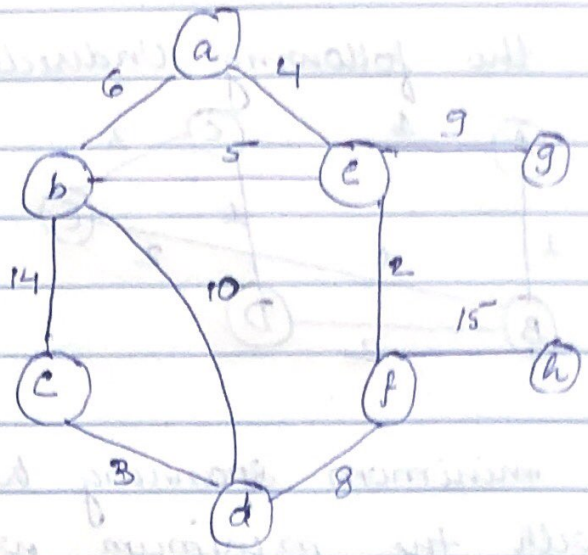


Set S	A	B	C	D	E	F
{ }	∞/nil	0/nil	∞/nil	∞/nil	∞/nil	∞/nil
B	5/B		1/B	2/B	∞/nil	∞/nil
B, C	5/B			2/B	5/C	3/C
B, C, D	4/D				5/C	3/C
B, C, D, F	4/D				4/F	
B, C, D, F, A					4/F	



5 Minimum Spanning tree using Kruskal's algorithm

Sol: given:

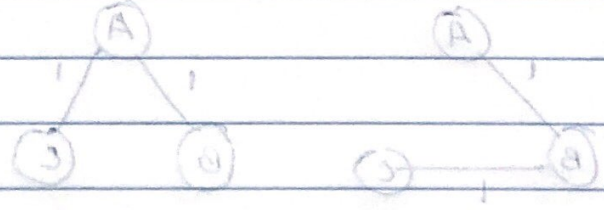


edges in sorted order

2:	(e, f)
3:	(c, d)
4	(a, e)
5	(b, e)
6	(b, a)
8	(d, f)
9	(e, g)
10	(b, d)
14	(b, c)
15	(f, h)

Set : (A) (B) (C) (D) (E) (F) (G) (H)

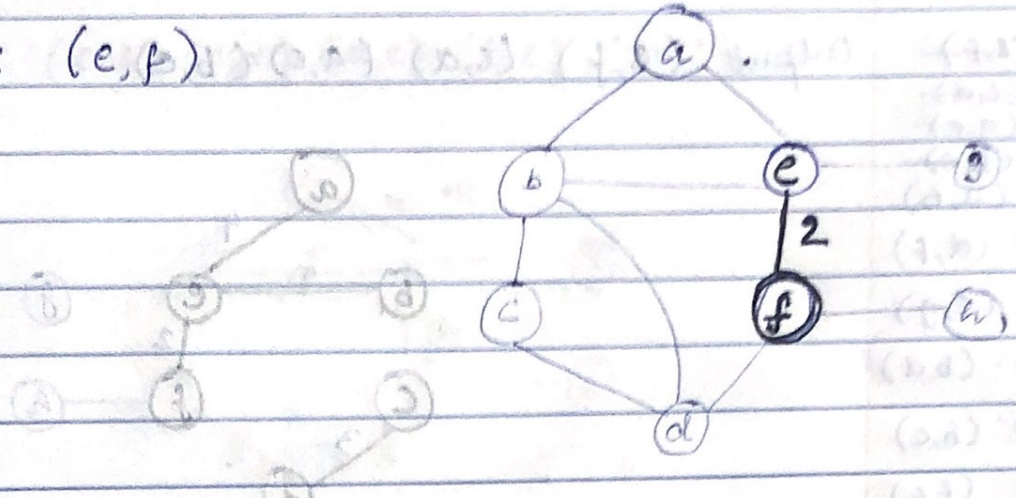
Output:



Step 1: Set : (e,f) (A) (B) (C) (d) (g) (h)

2: ~~(e,f)~~
3: (c,d)
4: (a,e)
5: (b,e)
6: (b,a)
8: (d,f)
9: (e,g)
10: (b,d)
14: (b,c)
15: (f,h)

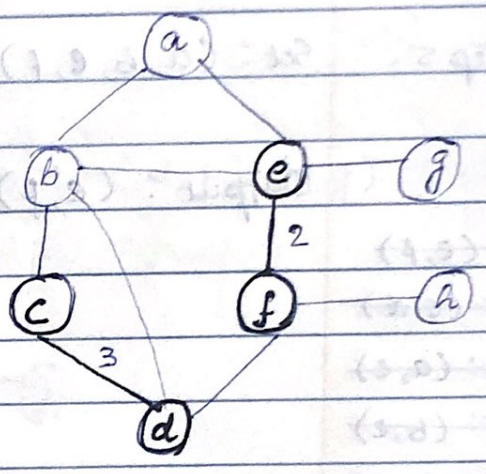
Output : (e,f)



Step 2: Set : (e,f) (c,d) (a) (b) (g) (h)

~~2: (e,f)~~
~~3: (c,d)~~
4: (a,e)
5: (b,e)
6: (b,a)
8: (d,f)
~~9: (e,g)~~
10: (b,d)
14: (b,c)
~~15: (f,h)~~

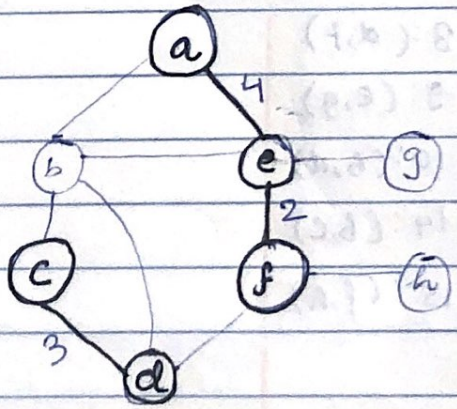
Output : (e,f) (c,d)



Step 3: Set : (a,e,f) (c,d) (B) (g) (h)

~~2: (e,f)~~
~~3: (c,d)~~
~~4: (a,e)~~
5: (b,e)
6: (b,a)
8: (d,f)
9: (e,g)
10: (b,d)
14: (b,c)
15: (f,h)

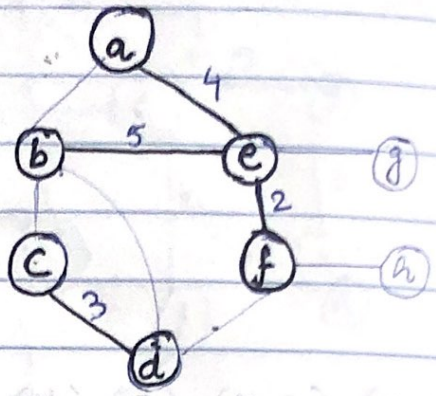
Output : (e,f) (c,d) (a,e)



Step 4: Set : (a, b, e, f) (c, d) (g) (h)

~~2 : (e, f)~~
~~3 : (c, d)~~
~~4 : (a, e)~~
~~5 : (b, e)~~
6 : (a, b)
8 : (d, f)
9 : (e, g)
10 : (b, d)
14 : (b, c)
15 : (f, h)

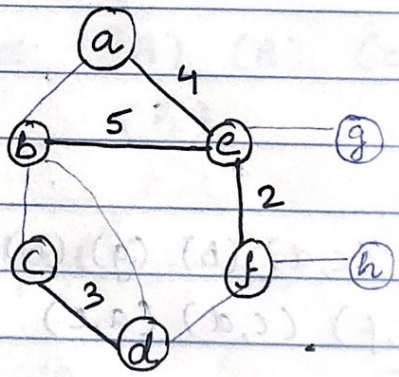
Output : (e, f) (c, d) (a, e) (b, e)



Step 5: Set : (a, b, e, f) (c, d) (g) (h)

Output : (e, f) (c, d) (a, e) (b, e)

~~2 : (e, f)~~
~~3 : (c, d)~~
~~4 : (a, e)~~
~~5 : (b, e)~~
~~6 : (a, b)~~
8 : (d, f)
9 : (e, g)
10 : (b, d)
14 : (b, c)
15 : (f, h)



step 6: Set : (a, b, c, d, e, f) (g) (h)

~~2: (e,f)~~ Output : (e,f) (c,d) (a,e) (b,e) (d,f)

~~3: (c,d)~~

~~4: (a,e)~~

~~5: (b,e)~~

~~6: (a,b)~~

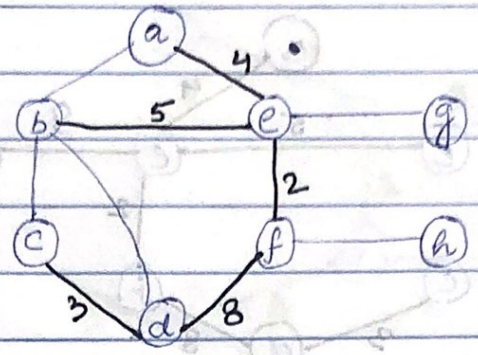
~~8: (d,f)~~

9 : (e,g)

10 : (b,d)

~~14: (b,c)~~

~~15: (f,h)~~



step 7 : Set : (a, b, c, d, e, f, g) (h)

Output : (e,f) (c,d) (a,e) (b,e) (d,f) (e,g)

~~2: (e,f)~~

~~3: (c,d)~~

~~4: (a,e)~~

~~5: (b,e)~~

~~6: (a,b)~~

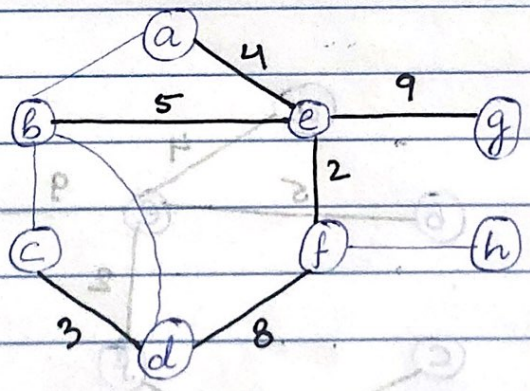
~~8: (d,f)~~

~~9: (e,g)~~

10 : (b,d)

14 : (b,c)

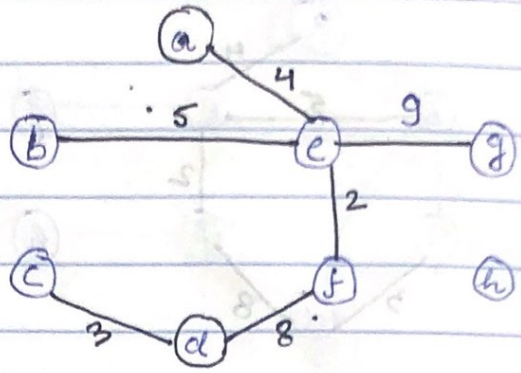
15 : (f,h)



Step 8: set : (a, b, c, d, e, f, g) (h)

~~2 : (e,f)~~
~~3 : (c,d)~~
~~4 : (a,e)~~
~~5 : (b,e)~~
~~6 : (a,b)~~
~~8 : (d,f)~~
~~9 : (e,g)~~
~~10 : (b,d)~~
14 : (b,c)
15 : (f,h)

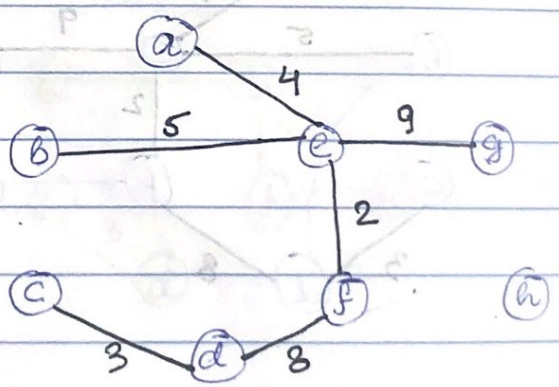
output : (e,f) (c,d) (a,e) (b,e) (d,f) (e,g)



Step 9: set : (a, b, c, d, e, f, g) (h)

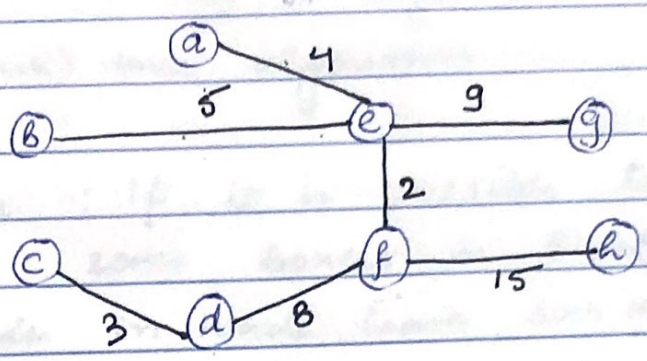
output : (e,f) (c,d) (a,e) (b,e) (d,f) (e,g)

~~2 : (e,f)~~
~~3 : (c,d)~~
~~4 : (a,e)~~
~~5 : (b,e)~~
~~6 : (a,b)~~
~~8 : (d,f)~~
~~9 : (e,g)~~
~~10 : (b,d)~~
~~14 : (b,c)~~
~~15 : (f,h)~~

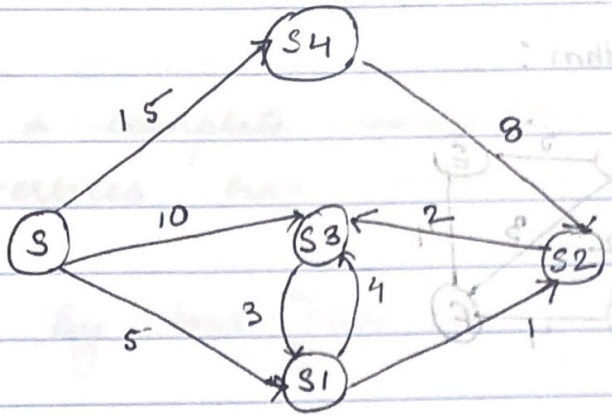


Step 10: Set: (a, b, c, d, e, f, g, h)

output: (e,f) (c,d) (a,e) (b,e) (d,f) (e,g) (f,h)



6



Solution = $\{(S,0)\}$

$D[S_1] = 5 \quad D[S_2] = \infty \quad D[S_3] = 10 \quad D[S_4] = 15$

Solution = $\{(S,0), (S_1,5)\}$

$D[S_2] = D[S_1] + w(S_1, S_2) = 5+1 = 6$

$D[S_3] = D[S_1] + w(S_1, S_3) = 5+4 = 9$

$D[S_4] = 15$

Solution = $\{(S,0), (S_1,5), (S_2,6)\}$

$D[S_3] = D[S_2] + w(S_2, S_3) = 6+2 = 8$

$D[S_4] = 15$

Solution = $\{(S,0), (S_1,5), (S_2,6), (S_3,8), (S_4,15)\}$

Solution = {(s, 0)}

D[s_1] = 5 for path [s, s_1]

D[s_2] = ∞ for path [s, s_2]

D[s_3] = 10 for path [s, s_3]

D[s_4] = 15 for path [s, s_4]

Solution = {(s, 0), (s_1, 5)}

D[s_2] = 6 for path [s, s_1, s_2]

D[s_3] = 9 for path [s, s_1, s_3]

D[s_4] = 15 for path [s, s_4]

Solution = {(s, 0), (s_1, 5), (s_2, 6)}

D[s_3] = 8 for path [s, s_1, s_2, s_3]

D[s_4] = 15 for path [s, s_4]

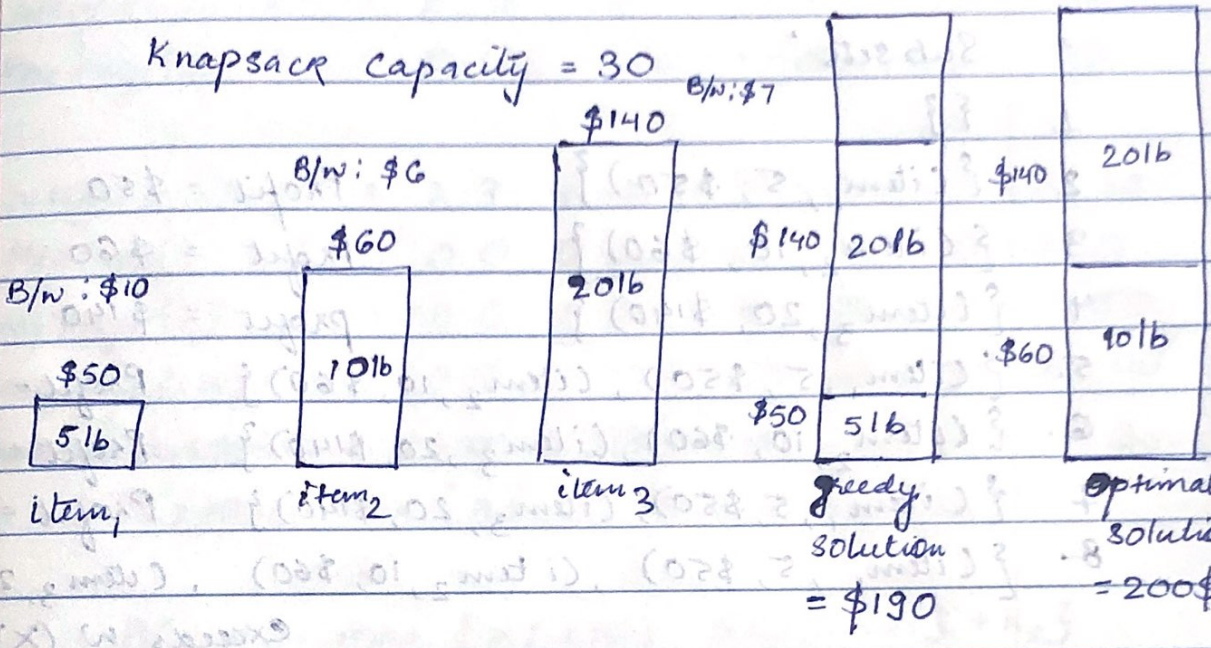
Solution = {(s, 0), (s_1, 5), (s_2, 6), (s_3, 8), (s_4, 15)}

7.

a. Using Greedy 4 (highest Benefit per unit weight First).

$S = \{(item_1, 5, \$50), (item_2, 10, \$60), (item_3, 20, \$140)\}$

Knapsack capacity = 30



Using Greedy 4 approch i.e. highest Benefit per unit weight First, we'll pick 5 lb 1st as it has B/w: $10 and than 20lb as B/w: $7, now our Knapsack has capacity 30 ∴ no more can be filled ∴ Greedy solution = $190 ($50 + $140).

However an optimal solution would have been 10 lb + 20lb i.e. 60$ + $140 = $200.

b. Brute Force Method :

$S = \{(item_1, 5, \$50), (item_2, 10, \$60), (item_3, 20, \$140)\}$ and $W = 30$.

a. Subsets:

1. $\{\}$
2. $\{(item_1, 5, \$50)\}$ Profit = \$50
3. $\{(item_2, 10, \$60)\}$ Profit = \$60
4. $\{(item_3, 20, \$140)\}$ Profit = \$140
5. $\{(item_1, 5, \$50), (item_2, 10, \$60)\}$ Profit = \$110
6. $\{(item_2, 10, \$60), (item_3, 20, \$140)\}$ Profit = \$200
7. $\{(item_1, 5, \$50), (item_3, 20, \$140)\}$ Profit = \$190
8. $\{(item_1, 5, \$50), (item_2, 10, \$60), (item_3, 20, \$140)\}$ exceeds W (X)

c Dynamic approch:

$W = 30, \quad S = \{(i_1, 5, \$50), (i_2, 10, \$60), (i_3, 20, \$140)\}$

Weight :	0	1	2	3	....	30
Max Profit{}	0	0	0	0	....	0

Weight	0	1	2	3	4	5	... 9	... 10	... 14	15 ... 30
Max Profit{}	0	0	0	0	0	0	... 0	... 0	0	0 ... 0
Max Profit{i_1}	0	0	0	0	0	50	.. 50	.. 50	... 50	50 ... 50
Max Profit{i_1, i_2}	0	0	0	0	0	50	... 50	60 ... 60	110 ... 110	

~~Max Profit{i_1, i_2, i_3}~~ ~~0 0 0 0 0 50 50~~

$B[2,10] = \max\{B[1,10], B[1, 10-10] + b_2\}$

$= 60$

$B[2,15] = \max\{B[1,15], B[1, 15-10] + b_2\}$

$= \max\{50, 50+60\}$

$= 110$

Wt:	0...4	5...9	10...14	15...19	20...24	25...29	30
MaxP{}	0...0	0...0	0...0	0...0	0...0	0...0	0
MaxP{i_1}	0...0	50...50	50...50	50...50	50...50	50...50	50
MaxP{i_1, i_2}	0...0	50...50	60...60	110...110	110....	110..	110
MaxP{i_1, i_2, i_3}	0...0	50...50	60...60	110...110	140..140	190...190	200

- $B[3,20] = \max\{B[2,20], B[2,20-20] + b_3\} = 140$
- $B[3,25] = \max\{B[2,25], B[2,25-20] + 140\} = 190$
- $B[3,30] = \max\{B[2,30], B[2,30-20] + 140\} = 200.$

8.

Lets consider there are

'n' → vertices (boxers) &

'K' → edges (rivalaries)

given O(n+K) time algorithm

To determine : If it is possible to place some boxers in Black team & remainder in white team such that each rivalry is between a Black & white team member.

Solution : This problem can be solved using the concept of bipartite graph.

For a bipartite graph, we can split the graph such that all edges have one node on each side of the split.

Algorithm :

- choose an arbitary Boxer node and label it as Black team member and place it in a set {U}. This our start node.
- perform BFS on the choosen start node boxer.

- Color all the neighbours of the choosen boxer with white colour and place it in Set $\{V\}$

- Now, consider a boxer b at the head of Queue and suppose that it has label as black. To continue the search, we look through b's edges. At this time we'll also look for neighboring nodes labled as boxers with same label as black. If found, it is a conflict as we have black-black rivalaries.

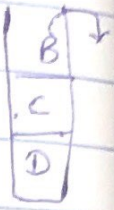
- If not, we have successfully partioned the boxers.

The major time-related component in this algorithm is breadth first Search which needs $O(n+k)$ where $O(n)$ time is required to allocate each boxer in a black team or white team and $O(k)$ time to check edges.

$\therefore$ Over all time for algorithm $O(n+k)$

9. LCS

	y_i	B	C	A	D
x_i	0	0	0	0	0
B	0	1↖	1←	1←	1←
C	0	1↑	2↖	2←	2←
D	0	1↑	2↑	2↑	3↖
E	0	1↑	2↑	2↑	3↑



∴ Our LCS is BCD

10.

BFS : 1, 2, 3, 4, 5, 6, 7

DFS : 1, 2, 5, 6, 3, 7, 4