30/08/19

MODULE 3

GREEDY STRATERGY

The greedy method is One of the algo design techniques. A Greedy algorithm always m the choice that looks best at that moment, wi considering whether this choice leads to a glob Optimal solution.

Greedy technique requires the criteria known as greedy choice for the solution of t solution of a local problem.

One the basis of greedy choice the optimum solution that looks best at that mom is selected and an optimum solution of the reri of the problem leads to an overall optimal so

For example :

The greedy choice in our example is receiving the fewest number of coins after paying the bill for a purchase.

Every solution has a cost or a value. Therefore the greedy choice is sometimes also known as Objective functions whose values are to be maximized or minimized.

Here we have to understand the differ between feasible and optimum solution.

There are many solution at each ste step the one which satisfies the problem const is known as feasible solution.

The best solution among all the feasible solution is known as ; Optimum solution satisfing the constraint.

There are two main property for greedy algorithm.

1. Greedy choice property.
2. Optimum Sub structure.

Greedy choice property:

The greedy approach constructs the solution through a sequence of steps, each step is chosen by making a "choice" that seems best at that particular moment to solve the problem. Such a choice is called the "greedy choice". At each step, a decision is taken based on the greedy choice. The application of the greedy choice at each step reduces the given problem into a smaller one. The choice made by greedy algorithms may depend on the choices made so far. The choices of a step once made cannot be changed in subsequent steps. If the data chosen on any step results with an optimal solution, then that data is added into the partial solution set. Each step is chosen in such a way that it is the best alternative among all feasible choices that are available.

Optimal Substructure:

A problem exhibits the Optimal substructure if the solution at each step is also Optimal, and the optimal solution of each subprogram leads to the global optimal solution of the problem, that is the local optimal solution leads to the global optimal solution.

The greedy approach should have the following functions:

1. Candidates function:

The set of all possible elements is known as a configuration set or a candidate set from which the solution is created. So, this function checks whether the selected items provide a solution or not.

Example: A finite set of all types of coins with at least one coin of each type for "Coin Change" problem.

Candidate = {10, 10, 5, 5, 5, 5, 2, 2, 2, 2, 2, 2, 2, 2, 1, 1, 1, 1}

2. Selection function:

The selection function tells which of the candidates is the most promising. The selection function is dependent on the problem at hand.

Example: Selecting the highest value coin.

3. Feasible Solution:

The set of all possible solutions satisfying the constraint at each step.

Example: The set of all possible combinations of coins in which the quantity does not exceed the total value.

- One coin of ₹10 and one coin of ₹5 = {10,5}.
- Three coins of ₹5 = {5,5,5}
- One coin of ₹10 and two coins of ₹2 and one coin of ₹1 = {10,2,2,1}
- Two coins of ₹5 and two coins of ₹ and one coin of ₹1 = {5,5,2,2,1}.
- Seven coins of ₹2 and one coin of ₹1 = {2,2,2,2,2,2,2,1}.
- Feasible = {{10,5}, {5,5,5}, {10,2,2,1}, {10,2,2,1}, {2,2,2,2,2,2,2,1}}.

4. Optimum Solution:

The best solution from the feasible solution satisfying the objective functions.

Eg: Two coins, ie; {10,5}.

5. Objective function:

The gives the final value of the solution. It is a constraint the maximizes or minimizes the profit.

Eg: Minimum number of coins used in the solution.

Objective = Two coins, ie; {10,5}.

The greedy choice may not necessarily produce the optimum solution. Greedy techniques have the following several advantages and disadvantages.

Advantages

1. Greedy methods are simple to understand and implement.
2. They are faster.
3. There are many problems where an optimal solution is sought using greedy technique.

Disadvantages.

1. They may not always produce the optimum solution
2. The correctness of the algorithm is very hard to prove
3. This approach can be disastrous for some computational task.

Applications

Many optimization problem as given in the following list can be solved using greedy algorithms.

1. Optimal merge pattern.
2. Huffman coding
3. Knapsack problem
4. Minimum spanning tree
5. Job Scheduling with deadlines
6. Single-source shorter path
7. Travelling salesman problem.

We will start with the optimal merge pattern.

Imp
03/09/19

Control Abstraction for Greedy Strategy.

```
Algorithm GreedyMethod (a,n) {
// a is an array of n inputs
    solution = ϕ
  for i=0 to n do
    {
      s:= select(a)
   if (feasible (solution,s)) then
     {
       solution := union (solution,s)
     }
   else
     reject();
  // if solution is not feasible
    }
     return solution;
}
```

Applications of Greedy Strategy.

1. Knapsack problem:

The problem is based on the concept of maximising the benefit.

A Theif robbing a general store has a knapsack of capacity w then he would try to Pack ~~Pnap~~ the knapsack with only those items that maximise the total cost of selected items.

A Theif can carry maximum weight of w in the knapsack. So there are 3 options.

1. Collect the most costly items in Order.
2. Collect the light weight items in order.
3. Collect the precious items in order. [ratio = value/weight].

So, there are 3 greedy choices.

1. Arrange the item in descending order of their value and then pick them in order.
2. Arrange the item in ascending order of their weight or size and then pick them in order.
3. Arrange the item in descending order of their ratio [value/weight] and then pick them in order.

The above mention Strategy is generalised to the selection of object from a set of n objects for the knapsack of capacity w.

Each object i has positive weight w_i and value v_i.

Objects are selected in Order to maximize the total value of selected Object in the knap sack without breaking the knapsack.

— 0/1 knapsack problem.

n=5

Suppose we have a knapsack of size 15 kg and 5 objects for selection and their value and weight are given below.

Size of knapsack = 15 kg

Object	A	B	C	D	E
weight (wi)	2	9	5	7	6
value (vi)	10	36	25	56	24

Method 1

Arrange the object in descending order of their value.

Object	D	B	C	E	A
Weight (wi)	7	9	5	6	2
value (vi)	56	36	25	24	10

Then, Select the most valuable item D with value 56 & weight 7.

Then;

Total size of knapsack = 15 kg.

∴ Remaining = 15 − 7

= 8//.

Then select B, B is rejected since weight = 9.

Then select the item C with weight 5 and value 25.

Remaining = 8 − 5

= 3//

Then select E, E is rejected since weight = 6.

Then select the item A with weight 2 and value

Remaining = 3 − 2

= 1//

∴

A
C
D

∴ Total profit = 56 + 25 + 10 = 91//.

It is not an optimal solution bcoz there is remaining one kg in knapsack.

Object	A	C	E	D	B
Weight (Wi)	2	5	6	7	9
value (Vi)	10	25	24	56	36

Select the least weight then select A.

Remaining = 15 − 2

= 13 //

Select the next least weight C; then

Remaining = 13 − 5

= 8

Select the next least weight E; then

Remaining = 8 − 6

= 2 //

We can insert any other weight.

Total profit = 10 + 25 + 24 = 59

It is not an optimal solution since there we can have remaining 2 knapsack.

Method 3

Object	A	B	C	D	E
weight (wi)	2	9	5	7	6
value (vi)	10	36	25	56	24
Ratio	$10/2$ 5	$36/9$ 4	$25/5$ 5	$56/7$ 8	$24/7$ 4.

Descending order.

Object	D	C	A	B	E
weight (wi)	7	5	2	9	6
value (vi)	56	25	10	36	24
Ratio	8	5	5	4	4

Select the item with most ratio the select D;

Remaining: $15 - 7$

$= 8$ //

Select the next item (C)

Remaining $= 8 - 5 = 3$ //

Select A = Remaining $= 3 - 2$

$= 1$ //

We can insert any other weight.

Total value = 56 + 25 + 10

= 91//

This is not an optimal solution Because there is remaining 1 kg in the knapsack.

Hw

I. n = 20

Objects	A	B	C
Size	18	8	6
Value	19	13	9

II n = 20

Objects	A	B	C
Size	18	15	10
Value	25	24	15

*

$n = 20$

Objects	A	B	C
Size	13	8	6
value	19	13	9

Method 1

$n = 20$

Then $20 - 13 = 7$ Remains A

B Rejected

$7 - 6 = 1$ Remains B

B rejected

1 kg

Total value $= 19 + 9 = 28$ //

Method 2

Arrange the items in the ascending order of weight

$n = 20$

Object	C	B	A
Size	6	8	13
value	9	13	19

1 kg

Select C, Remaining $= 20 - 6 = 14$

Select B, Remaining $= 14 - 8 = 6$

Total value $= 9 + 13 = 22$ //

Method 3

Arrange the items in the descending order of ra

Object	B	A	C
Size	8	13	6
value	13	19	9
Ratio	1.625	1.5	1.462

Select B,

Remaining $20 - 8 = 12$ // Skip

Select C,

Remaing $12 - 6 = 6$ //

6 kg //

Total value $= 13 + 9 = 22$ //

$n = 20$

Objects	A	B	C
Size	18	15	10
value	25	24	15

Method1

$n = 20$

$20 - 18 = 2$ A remains

B rejected

Total value = 25//

2kg

It is not optimal solution.

Method2

Object	A	B	C
Size	18	15	10
value	25	24	15

Select A,

Remaining $20 - 18 = 2$//

$2 <$ size of B, so skip B

$2 <$ size of C, so skip C

Total value = 25//

2kg

Method3

Object	B	C	A
Size	15	10	18
value	24	15	25
ratio	1.6	1.5	1.39

Select B,

Remaining $= 20 - 15 = 5$

$5 <$ size of C, so skip C

$5 <$ size of A, so skip A

Total value = 24//

Remaing 5 kg

6.09.19

o Fractional knapsack problem

In this case, Items can be broken into smaller pieces. Hence, the theif can select fraction of items according to the problem statement.

* There are n items in the store.
* Weight of the i^{th} item is $w_i > 0$.
* Profit for i^{th} item is $p_i > 0$
* Capacity of knapsack is W.
* In this version of knapsack problem items can be broken into smaller pieces. So, the theif may take a fraction of x_i of i^{th} item.

$0 \leq x_i \leq 1$.

The i^{th} item contributes the weight $x_i \cdot w_i$ to the Total weight in the knapsack and profit value $x_i \cdot p_i$ to the Total profit. Hence the Objective of this Algorithm is to maximise $\sum_{n=1}^{n} (x_i \cdot p_i)$. Subject to the constraint that; $\sum_{n=1}^{n} (x_i \cdot w_i) \leq W$.

It is clear that an Optimal solution must fill the knapsack exactly, Otherwise we could add a fraction of remaining items and increase the Overall profit.

Thus an Optimal Solution can be obtained by;

$$\sum_{n=1}^{n} (x_i \cdot w_i) = W$$

In this context first we need to sort those items according to the value of p_i/w_i. So that

$$\frac{P_{i+1}}{w_{i+1}} \leq \frac{P_i}{w_i}$$

Here; x is an array to store the fractions of items.

Q. Find the optimum solution of a knapsack of capacity W=60 and having 4 Objects which is listed below.

Item	A	B	C	D
(value) Profit	280	100	120	120
(size) weight	40	10	20	24
Ratio P_i/w_i	7	10	6	5

P_i/w_i of A = 280/40

= 7

P_i/w_i of B = 100/10

= 10

P_i/w_i of C = 120/20

= 6

P_i/w_i of $D = \frac{120}{24}$

$= 5$

Descending order of Ratio.

Item	B	A	C	D
Profit	100	280	120	120
weight	10	40	20	24
P_i/w_i	10	7	6	5

Select the Item ~~with~~ B into the knapsack.

ie; $60 - 10 = 50$//

Select the Item A into the knapsack.

$50 - 40 = 10$//

Select the Item C ; $10 < 20$; we have to find new weight and profit.

weight $C = 10/20 \times 20$

$= 10$//

profit of $C = 10/20 \times 120$

$= 60$

Total profit = B+A+C

= 100+280+60

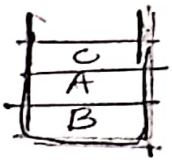
= 440

100+
280+
60
440

Total weight = B+A+C

= 10+40+10

= 60



Q. Find out the optimal solution for the following 3 items $(P_1, P_2, P_3) = (500, 400, 700)$ and $(w_1, w_2, w_3) = (1, 3, 7)$ having knapsack of capacity is 5.

Item	A	B	C
Profit	500	400	700
Weight	1	3	7
Ratio	500	300	100

Select A = 5−1 = 4

Select B = 4−3 = 1

weight = 1/7 × 7 = 1

Profit = 1/7 × 100 = 100

Total profit = 500+900+100

= 1500

Total weight = 1+3+1

= 5//

∴ Total profit = 1500

Total weight = 5 ///

Fractional knapsack problem. W=30.

Item	A	B	C	D
value	50	140	60	60
Size	5	20	10	12

Item	D	A	B	C
value	60	50	140	60
Size	2	5	20	10
ratio	30	10	7	6

$D \Rightarrow 30-2 = 28$

$A \Rightarrow 28-5 = 23$

$B \Rightarrow 23-20 = 3$

$C \rightarrow 10 \times 3)$

Size of fraction of $C = 3/10$

Size of fraction of $C = 3/10 \times 10 = 3$

Value of $C = 3/10 \times 60 = 18$

Total value $D+A+B+C$

$= 60+50+140+18$

$= 208 //$

Total Size $= D+A+B+C$

$= 2+5+20+3$

$= 30 //$

$\therefore$ Total value $= 208$

Total Size $= 30$

Algorithm Fractionalknapsack(S, W)

Input: Set $S = \{i_1, i_2, i_3, \ldots, i_n\}$ of items. Each item is assigned with the value v_i and weight w_i. The maximum weight a knapsack can handle is W.

Output: Amount x_i of each item i to maximize benefit benefit with weight at most W.

```
Repeat for each item i = 1 to n
    set xi ← 0        (selection part of item i is set to
                       zero for every item initially)
    set pi ← vi / wi  {profit value of each item}
End repeat
    set cw ← 0 and i ← 1  {set current total weight to
                           0 and start index to 1}

Repeat while cw < w and ik = n
    remove item i with highest profit value Pi from set S
    if (cw + wi) <= w then
        set xi ← 1 and cw ← cw + wi
    else
        set xi ← {w - cw} / wi
        cw = w
    endif
    set i ← i+1
    endwhile
return(x)
```

Job Sequencing problem with Deadline.

In job sequencing problem the objective is to find a sequence of jobs which is completed within their deadline and give their maximum profit.

* Each job takes one unit of time for completion.
* On the only available single processor or machine.
* The Greedy method is used to maximise the profit by finding the subset of maximum jobs to complete on the machine.
* Only one job can be done on the machine at a time and we can avail profit of the job if it is completed within the deadline.
* To solve these problem we have to find the schedule or timeslot that maximises profit.

Points to Remember

1. We have n jobs $J_1, J_2, J_3, \ldots, J_n$. This n jobs are associated with some deadline $D_1, D_2, D_3, \ldots, D_n$ and some profit $P_1, P_2, P_3, \ldots, P_n$.

Condition

~~while~~ if the job complete before the deadline then the profit is earned.

Pseudo code

```
for i=1 to n do
    set k = min(dmax, deadline(i))
    while k>=1 do
    if timeslot[k] is empty then
        timeslot[k] = job[i]
        break;
    end if
    set k=k-1
  end while
end for.
```

The following table consist of n jobs and thier corresponding deadline and profit. Find Out the maximum profit or optimal Solution if the jobs have to be completed in respective deadlines.

Arrange the table in descending order of their profit and initialise time slot has empty.

Index	1	2	3	4	5
Job	A_1	A_2	A_3	A_4	A_5
Deadline	2	1	3	2	1
Profit	60	100	20	40	20

Index	1	2	3	4	5
Job	A_2	A_1	A_4	A_3	A_5
Deadline	1	2	2	3	1
Profit	100	60	40	20	20

Value of dmax = 3 ; n = 3 (3 jobs)

Time slot (Max : 3) So, it is Timeslot

Time slot	1	2	3
Status	empty	empty	empty

Step 2 : Find the value of dmax . ie; Maximum Deadline.

Here dmax = 3

Total No: of jobs = 6

No: of jobs added to the slot = 3 .

Step 3 :

Consider Index = 1

Select job A_2, Deadline of A_2 = k(3, 1)

k = min(3, 1)

= 1 //

Put job A_2 in slot no: 1.

Timeslot	1	2	3
Status	A_2	Empty	Empty

choose job A_1, deadline of $A_1 = 2$. put job A_1 in slot 2.

Timeslot	1	2	3
Status	A_2	A_1	empty

Choose job A_4, deadline of $A_4 = 2$, since slot 2 is filled and the lowest slots are also filled neglect job A_4. (Reject)

choose job A_3, deadline of $A_3 = 3$, put A_3 in slot 3.

Time slot	1	2	3
Status	A_2	A_1	A_3.

Since all Timeslots are filled reject job 5.

Final job sequence :-

$A_2 - A_1 - A_3$.

Total profit = 100 + 60 + (40) + 20 + (20)

= 180

Jobs

* There are 5 profits $(P_1, P_2, P_3, P_4, P_5) = (20, 15, 10, 1, 6)$ and the deadline $= (2, 2, 1, 3, 3)$

Job	J_1	J_2	J_3	J_4	J_5
Deadline	2	2	1	3	3
Profit	20	15	10	1	6

Value of dmax = 3

Timeslot

Timeslot	1	2	3
Status	empty	empty	empty

~~Step~~

Job	J_1	J_2	J_3	J_5	J_4
Deadline	2	2	1	3	3
Profit	20	15	10	6	1

Total No: of Jobs = 5

Consider index = 1

Select Job J_1, Deadline of $J_1 = k(3,2)$

$= 2//$

Put job J_1 in slot no: 2

Time slot	1	2	3
Status	empty	J_1	empty

Choose Job J_2, Deadline of $J_2 = k \min(3,2)$

$= 2//$

Choose Job J_2, Deadline of $k-1$

$= 2-1$

$= 1//$

Timeslot	1	2	3
status	J_2	J_1	empty

Choose Job J_3, Deadline of $J_3 = k \min(3,1)$

$= 1//$

$k-1$

$= 0//$

Neglect J_3

Choose Job J_5, Deadline of $J_5 = k \min(3,3)$

$= 3//$.

Timeslot	1	2	3
Status	J_2	J_1	J_5

Since all Timeslots are filled reject Job J_4

Final Job Sequence

$J_2 - J_1 - J_5$

Total profit = 15 + 20 + 6

= 41.

15 +
20 +
6
41

* HW

5 jobs each with their deadline and profit value are given here. Schedule them so that maximum profit can be acheive.

Jobs	C	A	D	B	E
Profit	10	30	5	25	3
Deadline	1	2	4	4	3

Arrange table in the ascending order of the Profit.

Jobs	A	B	C	D	E
Profit	30	25	10	5	3
Deadline	2	4	1	4	3

$d_{max} = 4$

Timeslot	1	2	3	4
Status	empty	empty	empty	empty

Select Job, ie $A = k \min(4,2)$

$= 2$ //

Put job J_2 in slot no: 2

Timeslot	1	2	3	4
Status	empty	A	empty	empty

$k \min(4,4) = 4$

Timeslot	1	2	3	4
Status	empty	A	empty	B

$k(\min(4,1)) = 1$

Timeslot	1	2	3	4
Status	C	A	empty	B

$k(\min(4,4) = 4$ Already filled

$k(\min(4,3) = 3$.

Timeslot	1	2	3	4
Status	C	A	D	B

Final job sequence = C - A - D - B

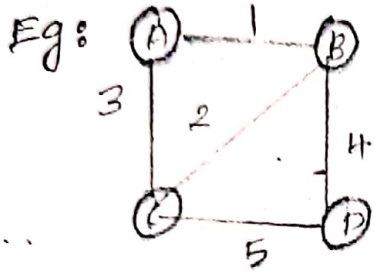
Total profit = 10 + 30 + 5 + 25 = 70 //

Prim's and Kruskal algorithm for Minimum Spanning Tree

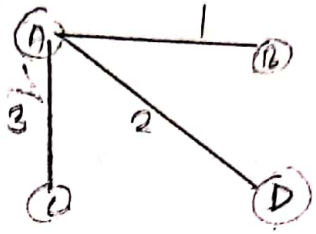
Spanning Tree of a graph consist of all vertices and some of the edges so that the graph doesnot contain a cycle.

Minimum Spanning tree is defined for weighted graph.

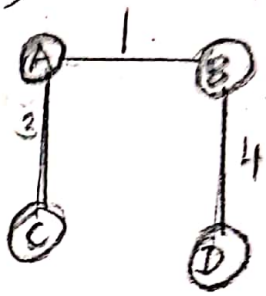
Eg:



(i)



(ii)



This is Spanning Tree.

Minimum Spanning Tree is defined for weighted graph

ie; Weight (i) = 6
Weight (ii) = 8

Weight (i) is minimum Spanning tree. Since 6 is minimum.

Prim's Algorithm

Let G = (V, E) is a connected, weighted Undirected graph.

Step 1 : Create two set V and V' such that :

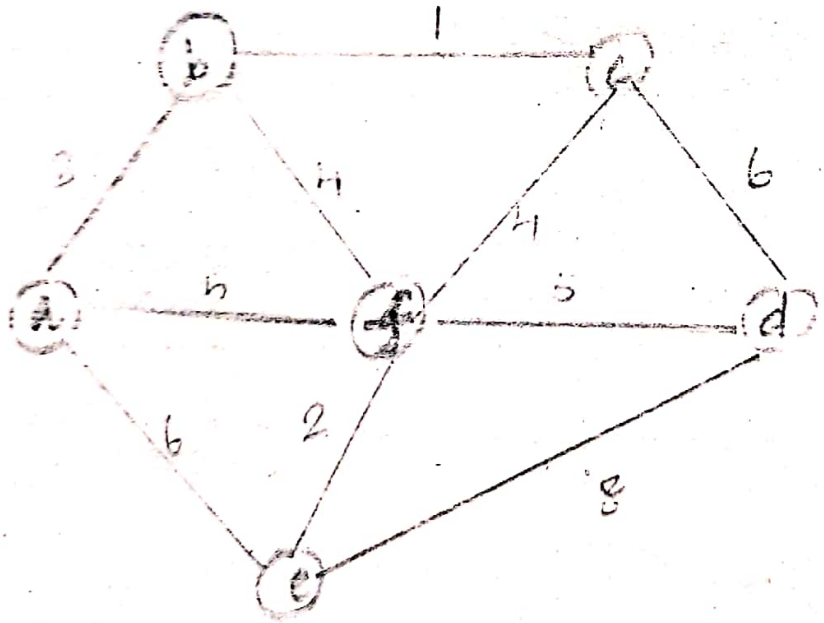
- V' is empty
- V contains all vertices of graph
- select Minimum weighted edge i, j from graph G and insert i and j into the set V'.

Step 2 : Repeat step 3 while V is not equal to V'.

Step 3 : Find all neighbours of all vertices which are in set V such that : one end point of neighbouring edges in V' and another not in V'.

Step 4 : sum up all selected edges weight (s).

Step 5 : Stop



$V = \{a, b, c, d, e, f\}$

$V' = \{$

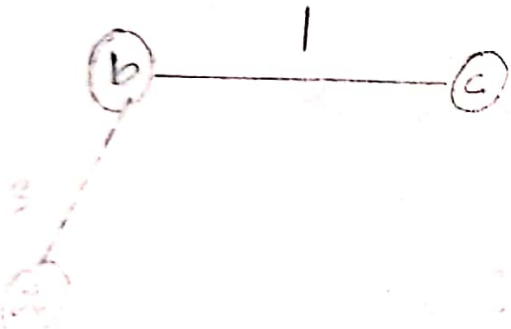
Conditions

1. Remove all loops
2. Remove all parallel edges between 2 vertex except the one with least way.
3. Select the vertex with minimum weighted edge as first vertex.

Step 1

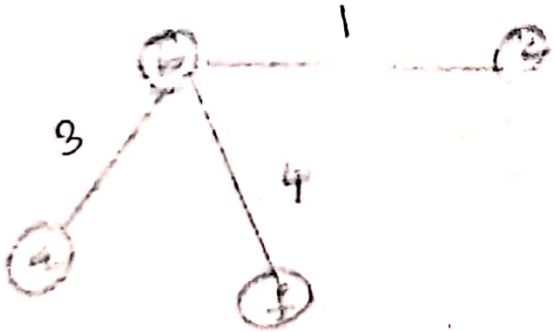
minimum spanning

Step 2

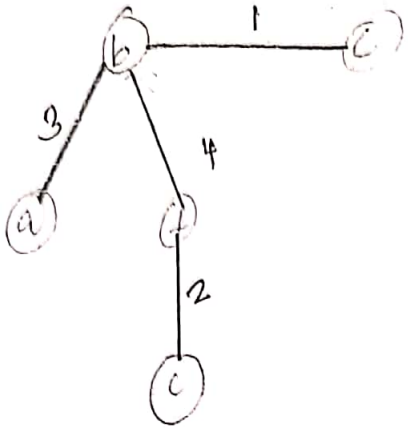


minimum spanning edges

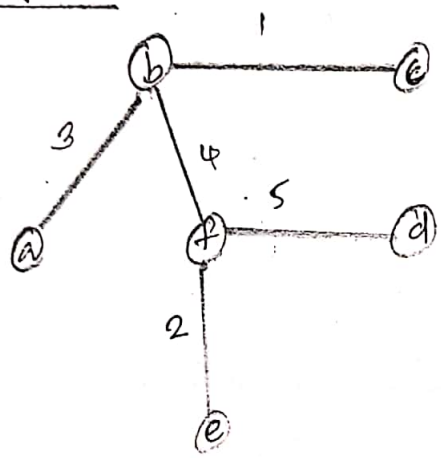
Step 3



Step 4



Step 5

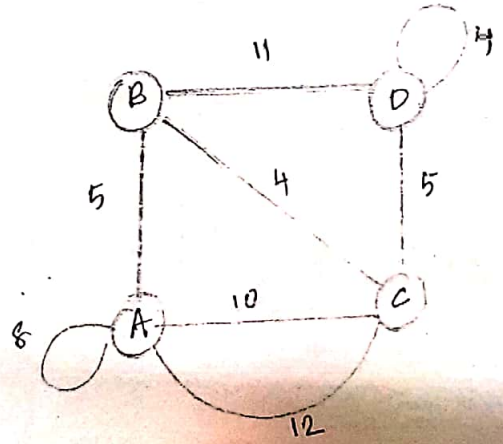


Total or sum = 1+3+4+2+5

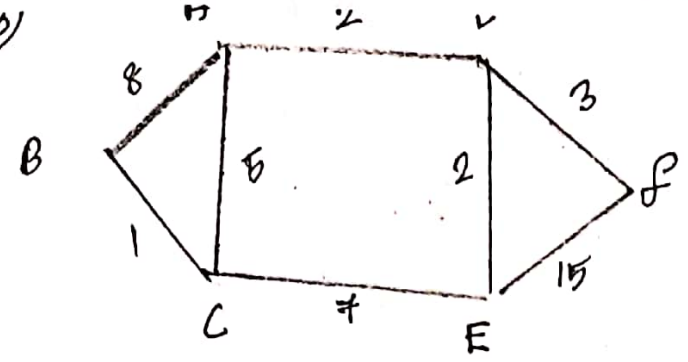
= 15

∴ Minimum weight = 15 //

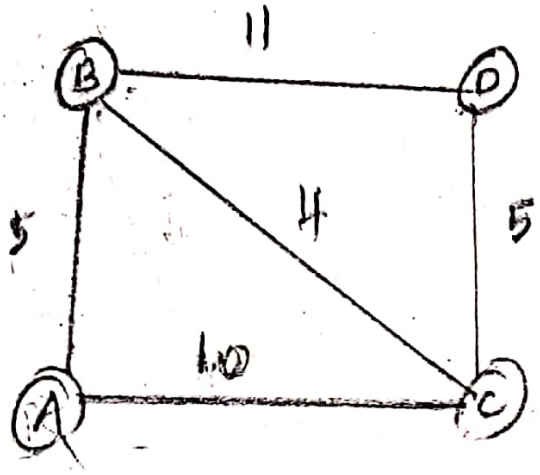
Q. (2)



(1)

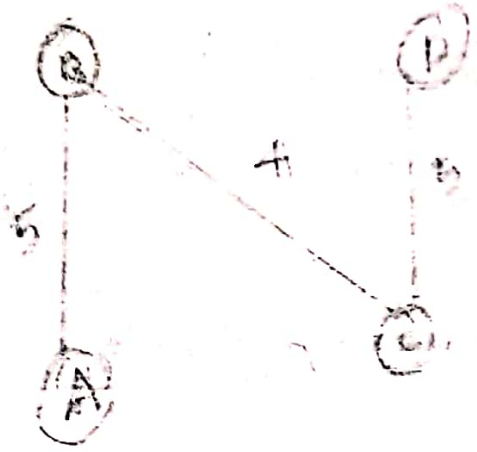


(2) Ans:-



Remaining Step:

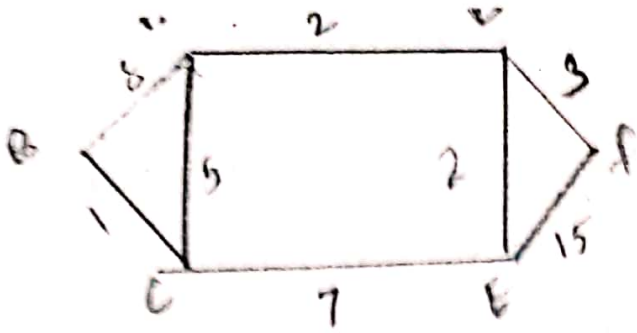
(3) Ans:-

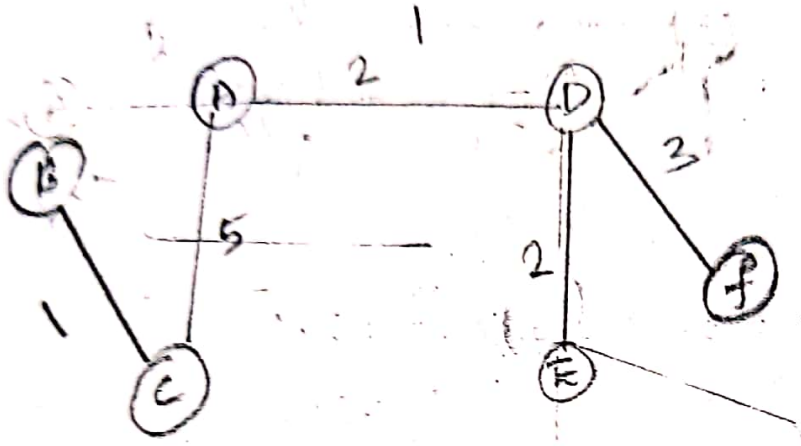


$\therefore$ Sum = 5+4+5

= 14

(3)





Sum = 1+5+2+2+3

= 13

19/09/19 Kruskal's Algorithm for Minimum Spanning Tree.

Kruskal's Algorithm is a minimum spanning tree algorithm which finds an edge of the least possible weight that connects any two trees in the forest.

* It finds a subset of edges that form a tree that include every vertex where the total weight of the edges in the tree is minimized.

Method

Step 1 : Remove all loops.

Step 2 : Remove all parallel edges

Step 3 : Sort all the edges in the ascending order of the weight.

step 4 : Pick the smallest edge, check if it forms a cycle with the spanning tree form so far. If cycle is not form include this edge. else, discard it.

Step 5 : Repeat Step 4 until there are $V-1$ edge in the spanning tree.

Pseudo code

```
MST_KRUSKAL (G,w)
1) A = ϕ
2) For each vertex v ∈ G,V
3) MAKE_SET (v)
4) Sort edges of the graph G into ascending
```

order by weight w.

5) For each edge (u,v) ∈ G1 taking in ascending order by weight.

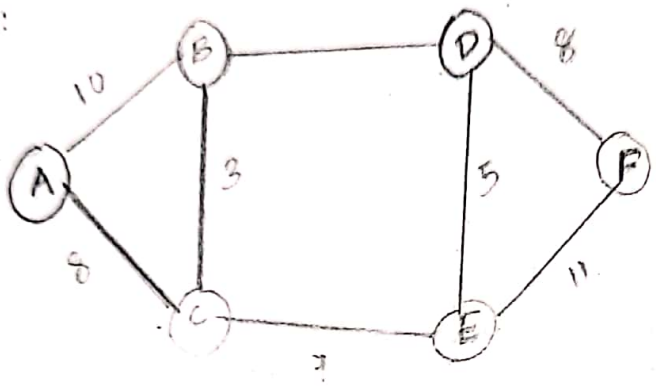
6) If FIND_SET (u) ≠ FIND_SET(v)

7) A = A ∪ {(u,v)}

8) UNION (u,v)

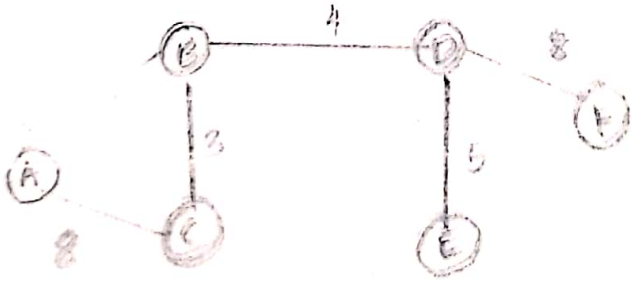
9) Return A.

Eg:



Edge in ascending order.

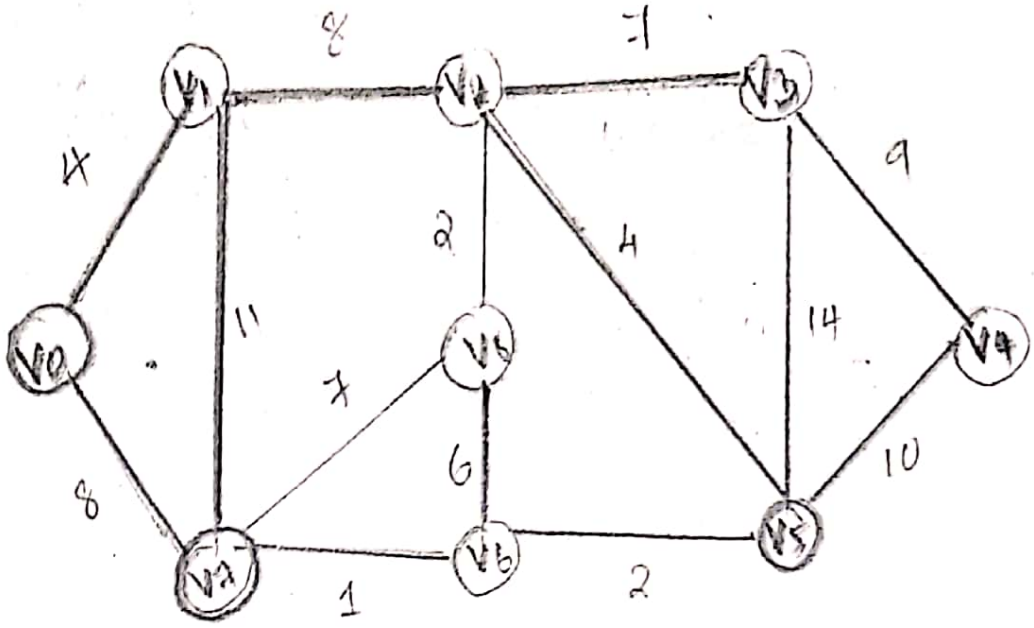
Edge	BC	BD	DE	CE	DF	AC	AB	EF
weight	3	4	5	7	8	8	10	11



Total weight =

8+3+4+5+8

= 28.

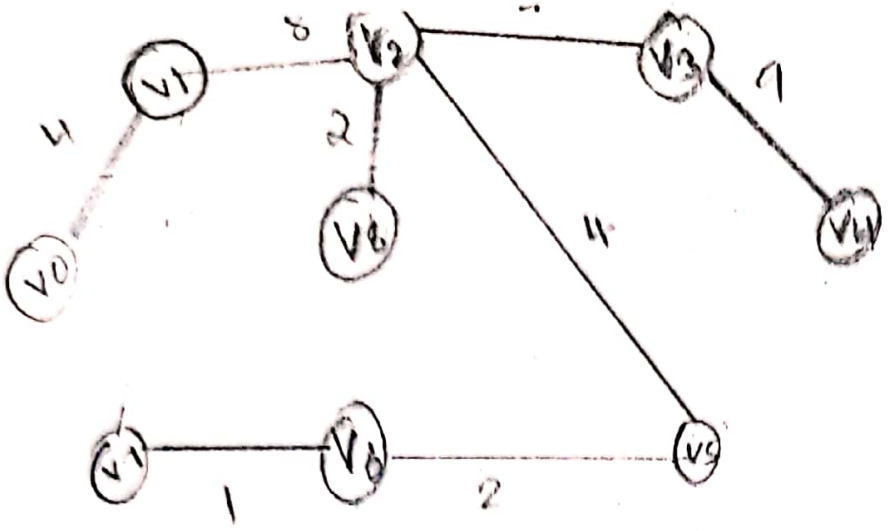


Edges in ascending order

Edges	V_7, V_6	V_6, V_5	V_2, V_8	V_0, V_1	V_5, V_2	V_8, V_6
weight	1	2	2	4	4	6

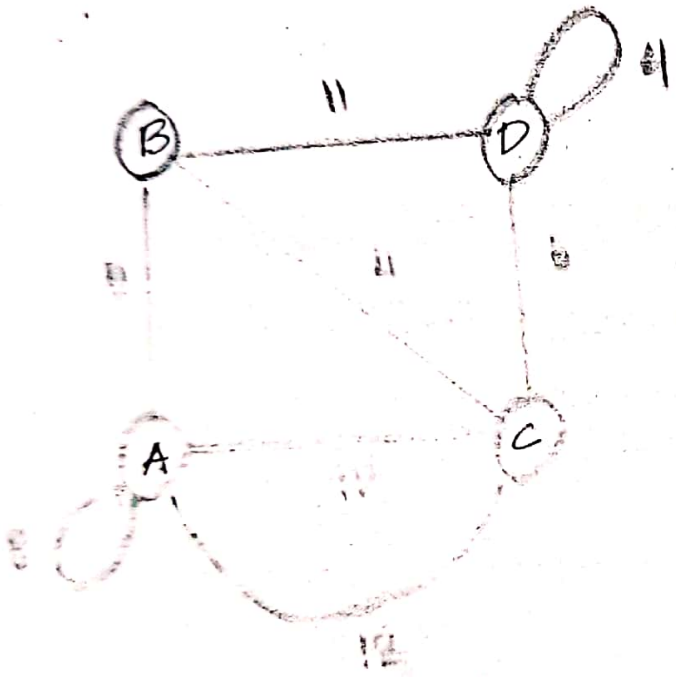
Edges	V_2, V_3	V_1, V_8	V_1, V_2	V_0, V_7	V_3, V_4
weight	7	7	8	8	9

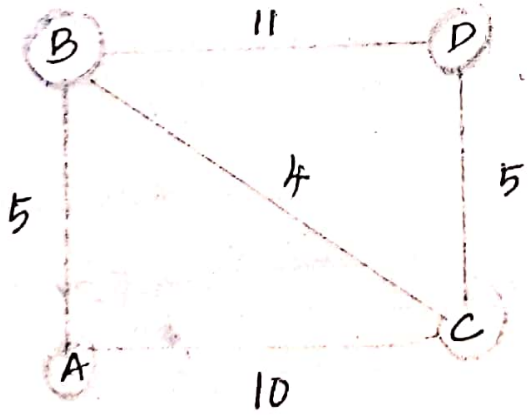
edges	V_5, V_4	V_1, V_7
weight	10	11



Total weight = 4+8+2+7+9+4+2+1

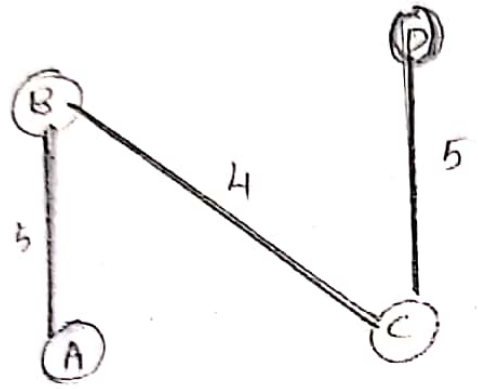
= 37





Remove all loops & parelled edges.

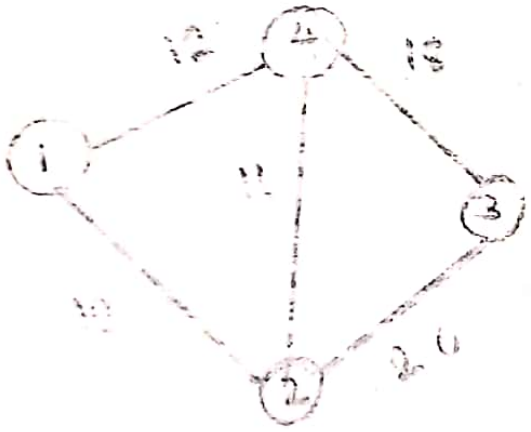
Edges	BC	CD	AB	AC	BD
weight	4	5	5	10	11



Total weight = 5+4+5

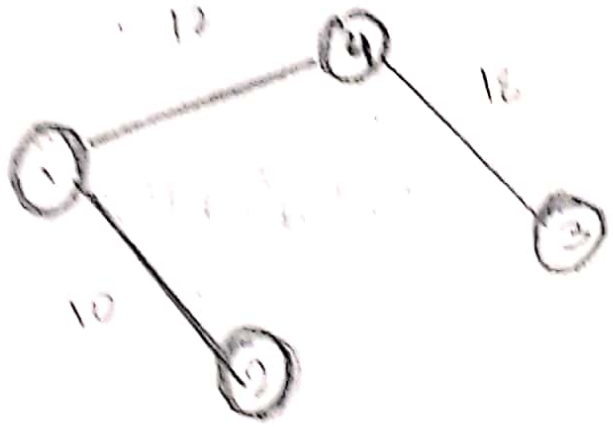
= 14

*



Edges in ascending order.

Edges	1,2	1,4	4,2	4,3	3,2
weight	10	12	16	18	20



Total weight = 10+12+18

= 40

1
10+
12+
18
40