	0	1	2	3	4	5	6
Start	1	3	2	0	5	8	$\boxed{11}$
End	3	4	5	7	$\boxed{9}$	10	12

START[6] >= END[4], Selected

Start = [1, 3, 2, 0, 5, 8, 11]
finish = [3, 4, 5, 7, 9, 10, 12]

Selected = {0, 1, 4, 6}.

KNAPSACK PROBLEM

Here knapsack is like a container or a bag. Suppose, we are given some items with some given weights and profits, We have to put some items in the knapsack in such a way that the total value produces a maximum profit.

Eg: The weight of a container is 20kg. We have to select the items in such a way that the sum of the weights should be either smaller than or equal to the weight of the container, and the profit should be maximum

There are two types of knapsack problems:

i) 0/1 Knapsack Problem

ii) Fractional Knapsack Problem.

0/1 KNAPSACK PROBLEM:
It means that the items are either completely added or not at all added to fill the knapsack.

FRACTIONAL KNAPSACK PROBLEM:
It means we can divide the items to fill the knapsack.

METHOD FOR SOLVING 0/1 KNAPSACK PROBLEM
i) arrange all given items in decreasing order of Profit/Weight.
ii) Now, start selecting items from the list ensuring that the weight of the item is less than the remaining capacity of the knapsack.

ALGORITHM → On next page

knapsack (W, p)
1. Compute profit to weight ratio,
$r_i = p_i / w_i$ for $i = 1$ to n

2. Sort the items in decreasing order of ~~value to~~ profit to weight ratios.
3. For all items do
    if the weight of the current item is ≤ less than or equal to remaining weight of knapsack then
        place it in the knapsack
    else
        proceed to the next one.

EXAMPLE: Knapsack capacity = 6.

Item number	1	2	3	4
Profit (P_i)	15	20	30	14
Weight (W_i)	3	2	10	2
P_i/W_i	5	10	3	7.

Arrange according to P_i/W_i

Item Number	2	4	1	3
Profit (P_i)	20	14	15	30
Weight (W_i)	2	2	3	10
Selection	✓	✓	X	X

Total Profit = 20 + 14 = 34   Ideal profit = 15 + 20 = 25 (Item 1 & 2)

1 for i = 1 to size(P)
  calculate cost[i] = P[i] / w[i]
  Sort-Descending (cost)

2 for i = 1 to n
  do x[i] = 0

3. weight = 0

4. While weight < M
  ~~do i = best remaining items~~

5. if weight + w[i] ≤ M
  then x[i] = 1
  weight = weight + w[i]
 else
  x[i] = (M - weight) / w[i]
  weight = M.
 return x.

M = 10

Items	1	2	3	4	5
Weights (in kg)	3	3	2	5	1
Profits	10	15	10	20	8.
Pi/Wi	3.3	5	5	4	8

arrange in descending order of Pi/Wi

Items	5	2	3	4	1
weights	1	3	2	5	3
Profits	8	15	10	20	10
Knapsack	1	1	1	4/5	0

It is difficult to find the optimal solution ~~using~~ in 0/1 knapsack using greedy approach but we can find an optimal solution using Brute force technique but its time complexity is 2^n.

ALGORITHM

Knapsack (Array w, array P, int m)

1. for i = 1 to size(P),
   calculate cost[i] = P[i]/W[i].
2. Sort - Descending(cost)
3. for i = 1 to n
   do x[i] = 0
4. Weight = 0

No. of elements.

FRACTIONAL KNAPSACK

ALGORITHM:

Fractional-knapsack (W, P, M)

W: Weights of items
P: Profits of items

M: Max capacity of knapsack.

Total Profit = $49 $\{(1*8) + (1*15) + (1*10) + (4/5 * 20)\}$

We can find an optimal solution ~~o~~ for a ~~In~~ fractional knapsack problem using Greedy approach.

MINIMUM SPANNING TREE

A minimum spanning tree (MST) is the collection of edges required to connect all vertices in an undirected graph with the minimum total edge weight.

In the real world, finding the Minimum Spanning Tree can help us find the most effective way to connect houses to the internet or to find the fastest route to deliver packages.

2 algorithms that can help us to find the Minimum Spanning Tree are:

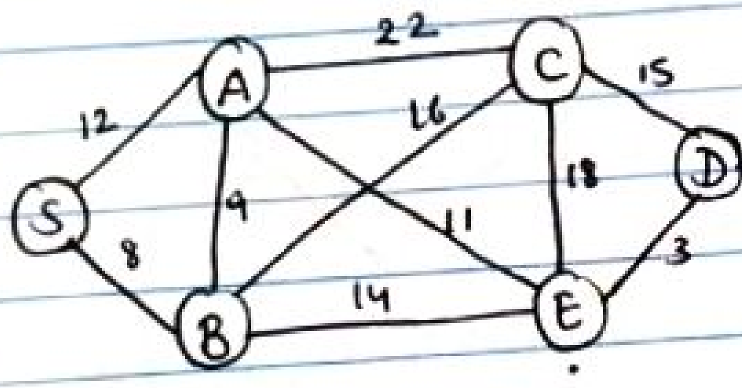
1. ~~Kr~~ Prim's algorithm
2. Kruskal's Algorithm.

Both these algorithms follow Greedy approach ~~to~~ to find the MST.

PRIM'S ALGORITHM

WORKING

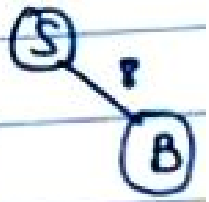
1. Choose a random vertex as the starting point, and include it as the first vertex in the MST.
2. Compare the edges going out from the selected vertex vertices and select the edge with the lowest weight which has not been visited before.
3. Add the edge and vertex to the MST.
4. Repeat steps 2 and 3 until all the vertices are not added to the MST.



Starting vertex: S

Visited = { }

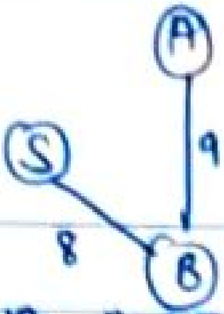
S → A = 12 , S → B = 8 (selected)



Visited = {S, B}

S→ ... B ... , B → C = 16 , B → E = 14, B → S = 8 (Already selected)

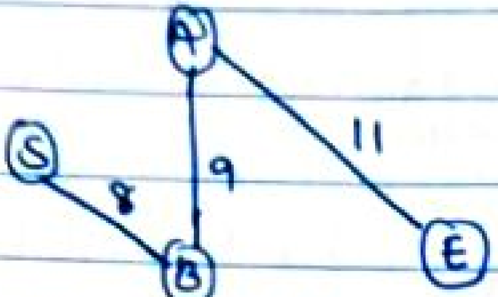
Visited = {S, B, A}



B → E = 14, B → C = 16,

A → C = 22, A → B = 9, A → E = 11 (Selected)

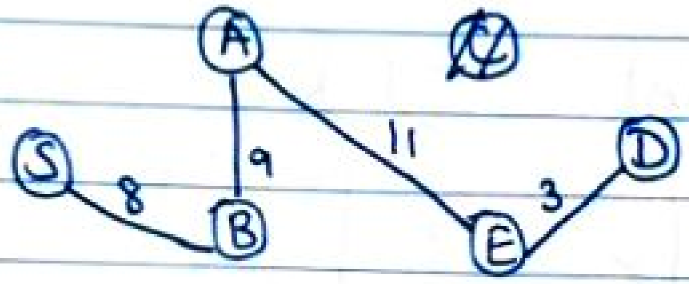
~~Vistor~~ Visited = {S, B, A, E}



B → E = 14, B → C = 16, A → C = 22, A → B = 9

E → C = 18 E → D = 3 (Selected)

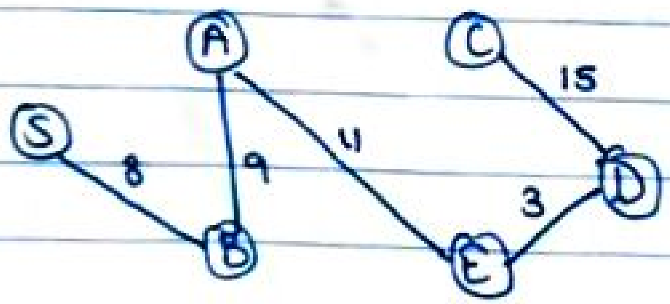
Visited = {S, B, A, E, D}

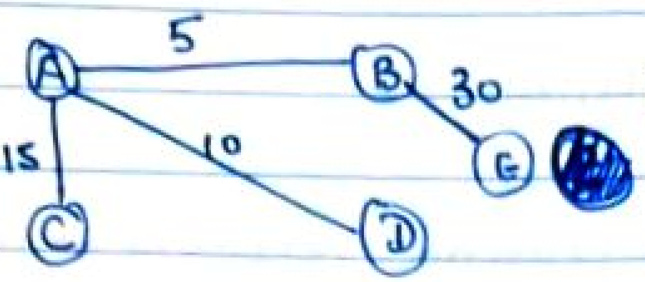


B → E = 14, B → C = 16, A → C = 22, A → B = 9

~~D → E = 3~~ , D → C = 15 (Selected)

~~Vista~~ Visited = {S, B, A, E, D, C}



4. 

Total cost = 5+10+15+30 = 60

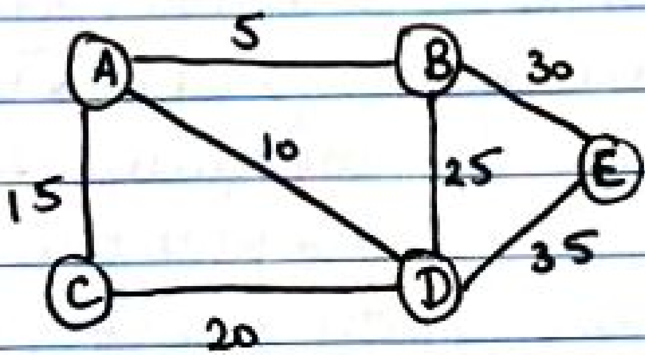
There must be exactly (n-1) edges to connect all the vertices using Kruskal's algorithm

PRIM'S ALGORITHM	KRUSKAL'S ALGORITHM
• Starts with any vertex of the graph arbitrarily. At no point of time, a forest is encountered in Prim's algorithm.	• Starts with all the vertices of the graph as a forest and every addition of the edge takes the forest one step further towards a complete tree.
• The concept of Prim's algorithm is based on the connectedness.	• The concept of Kruskal's algorithm is based on the acyclic nature of the graph.
• Saves a lot of space as no additional structure are used for storing the edges.	• More time is saved as the edges are sorted first.
• Always a new vertex	• Adding of an edge is

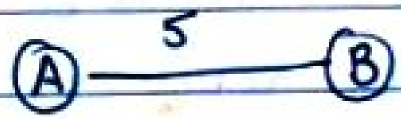
KRUSKAL'S ALGORITHM

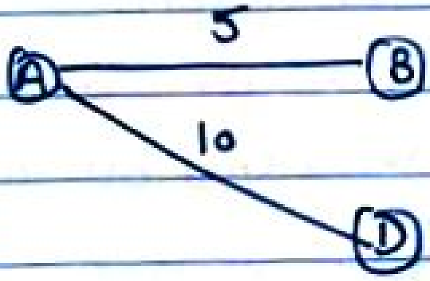
WORKING:

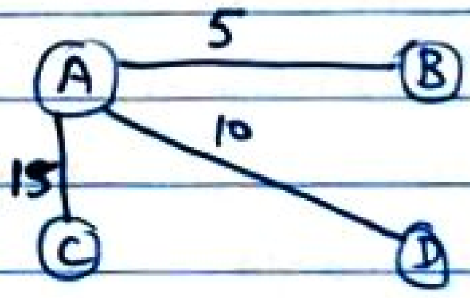
1. Arrange all the edges in an increasing order of weights.
2. Repeat the following steps until all the vertices are connected.
   a) Add an edge if the added edge do not form a cycle.
3. Add the costs of all edges in an MST to get the minimum cost.



AB	5	Accept
AD	10	Accept
AC	15	Accept
CD	20	Reject
BD	25	Reject
BE	30	Accept
DE	35	Reject

1. 

2. 

3. 

is joined to an old vertex.

- Addition of nodes is based on the concept of shortest distance or weight.

performed by selecting the sorted edge.

- Next edge is always determined by the edge list.

	0	1	2	3	4	5	6
Start	1	3	2	0	5	8	$\boxed{11}$
End	3	4	5	7	$\boxed{9}$	10	12

START[6] >= END[4], Selected

Start = [1, 3, 2, 0, 5, 8, 11]
finish = [3, 4, 5, 7, 9, 10, 12]

Selected = {0, 1, 4, 6}.

KNAPSACK PROBLEM

Here knapsack is like a container or a bag. Suppose, we are given some items with some given weights and profits, We have to put some items in the knapsack in such a way that the total value produces a maximum profit.

Eg: The weight of a container is 20kg. We have to select the items in such a way that the sum of the weights should be either smaller than or equal to the weight of the container, and the profit should be maximum.

There are two types of knapsack problems:

i) 0/1 Knapsack Problem

ii) Fractional Knapsack Problem.

0/1 KNAPSACK PROBLEM:
It means that the items are either completely added or not at all added to fill the knapsack.

FRACTIONAL KNAPSACK PROBLEM:
It means we can divide the items to fill the knapsack.

METHOD FOR SOLVING 0/1 KNAPSACK PROBLEM
i) arrange all given items in decreasing order of Profit/Weight.
ii) Now, start selecting items from the list ensuring that the weight of the item is less than the remaining capacity of the knapsack.

ALGORITHM → On next page

knapsack (W, p)
1. Compute profit to weight ratio,
$r_i = p_i/w_i$ for $i = 1$ to n

2. Sort the items in decreasing order of ~~value to~~ profit to weight ratios.

3. For all items do

    if the weight of the current item is ≤ ~~less than or equal to~~ remaining weight of knapsack then

        place it in the knapsack

    else

        proceed to the next one.

EXAMPLE: Knapsack capacity = 6.

Item number	1	2	3	4
Profit (P_i)	15	20	30	14
Weight (W_i)	3	2	10	2
P_i/W_i	5	10	3	7.

Arrange according to P_i/W_i

Item Number	2	4	1	3
Profit (P_i)	20	14	15	30
Weight (W_i)	2	2	3	10
Selection	✓	✓	X	X

(Item 1 + 2)

Total Profit = 20 + 14 = 34 Ideal profit = 15 + 20 = 25

It is difficult to find the optimal solution ~~using~~ in 0/1 knapsack using greedy approach but we can find an optimal solution using Brute force technique but its time complexity is 2^n.

ALGORITHM

Knapsack (Array w, array P, int m)

1. for i = 1 to size(P),
   calculate cost[i] = P[i]/W[i].
2. Sort - Descending(cost)
3. for i = 1 to n
   do x[i] = 0
4. Weight = 0

No. of elements.

FRACTIONAL KNAPSACK

ALGORITHM:

Fractional-knapsack (w, P, M)

W: Weight of items
P: Profits of items

M: Max capacity of knapsack.

1 for i = 1 to size(P)
calculate cost [i] = P[i] / w[i]
Sort-Descending (cost)

2 for i = 1 to n
do x[i] = 0

3. weight = 0

4. While weight < M
~~do i = best remaining items~~

5. if weight + w[i] ≤ M
then x[i] = 1
weight = weight + w[i]
else
x[i] = (M - weight) / w[i]
weight = M.
return x.

M = 10

Items	1	2	3	4	5
Weights (in kg)	3	3	2	5	1
Profits	10	15	10	20	8.
Pi / Wi	3.3	5	5	4	8

arrange in descending order of Pi / Wi

Items	5	2	3	4	1
weights	1	3	2	5	3
Profits	8	15	10	20	10
Knapsack	1	1	1	4/5	0

Total Profit = 49 $\{(1*8) + (1*15) + (1*10) + (4/5 * 20)\}$

We can find an optimal solution for a ~~In~~ fractional knapsack problem using Greedy approach.

MINIMUM SPANNING TREE

A minimum spanning tree (MST) is the collection of edges required to connect all vertices in an undirected graph with the minimum total edge weight.

In the real world, finding the minimum Spanning Tree can help us find the most effective way to connect houses to the internet or to find the fastest route to deliver packages.

2 algorithms that can help us to find the Minimum Spanning Tree are:

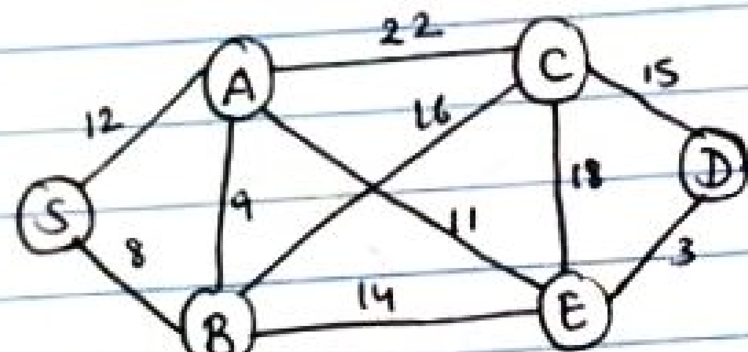
1. ~~Kru~~ Prim's algorithm
2. Kruskal's Algorithm.

Both these algorithms follow Greedy approach ~~to~~ to find the MST.

PRIM'S ALGORITHM

WORKING

1. Choose a random vertex as the starting point, and include it as the first vertex in the MST.
2. Compare the edges going out from the selected vertex/vertices and select the edge with the lowest weight which has not been visited before.
3. Add the edge and vertex to the MST.
4. Repeat steps 2 and 3 until all the vertices are not added to the MST.

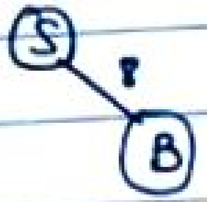


Starting vertex: S

Visited = { }

S → A = 12 , S → B = 8 (selected)

visited = {S, B}



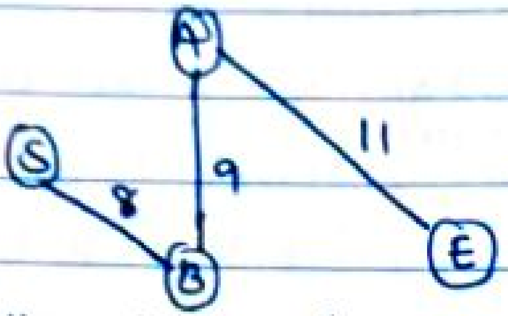
S—
B

, B → C = 16 , B → E = 14, B → S = 8 (Already selected)

Visited = {S, B, A}

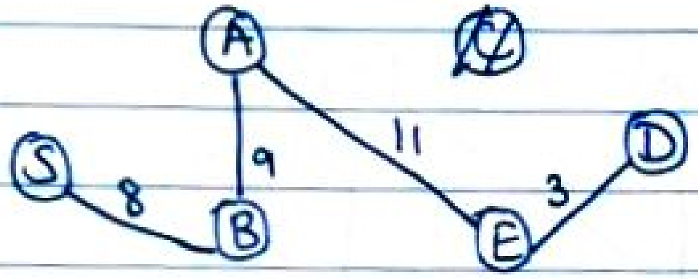
B → E = 14, B → C = 16,
A → C = 22, A → B = 9, A → E = 11
(Selected)

~~vistor~~ Visited = {S, B, A, E}



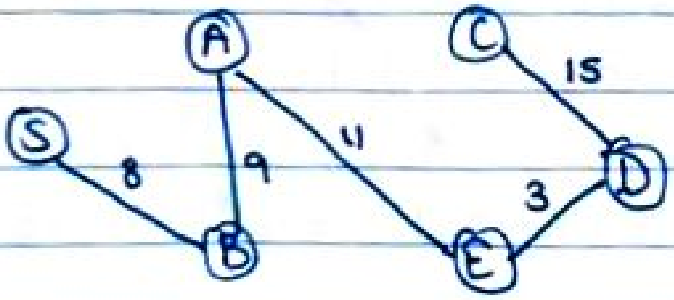
B → E = 14, B → C = 16, A → C = 22, A → B = 9
E → C = 18 E → D = 3 (Selected)

Visited = {S, B, A, E, D}



B → E = 14, B → C = 16, A → C = 22, A → B = 9
~~D → E = 3~~ , D → C = 15 (Selected)

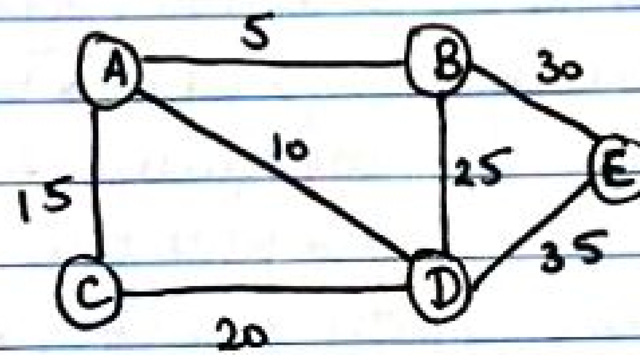
~~Vista~~ Visited = {S, B, A, E, D, C}



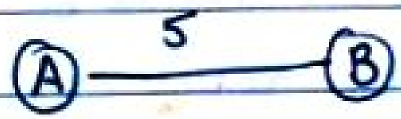
KRUSKAL'S ALGORITHM

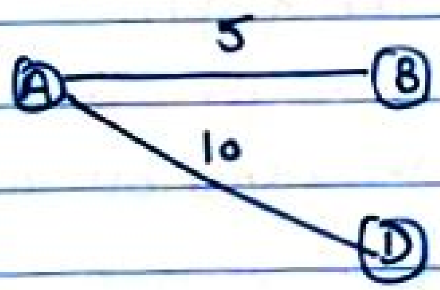
WORKING:

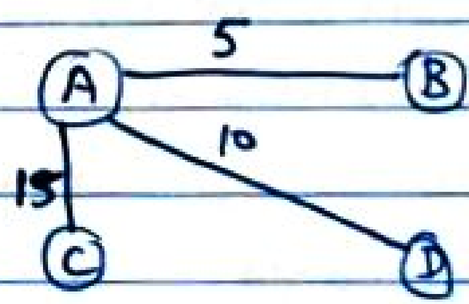
1. Arrange all the edges in an increasing order of weights.
2. Repeat the following steps until all the vertices are connected.
   a) Add an edge if the added edge do not form a cycle.
3. Add the costs of all edges in an MST to get the minimum cost.

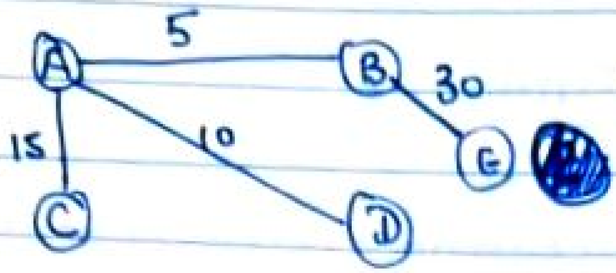


AB	5	Accept
AD	10	Accept
AC	15	Accept
CD	20	Reject
BD	25	Reject
BE	30 - Accept	
DE	35 - Reject	

1. 

2. 

3. 

4. 

Total cost = 5+10+15+30 = 60

There must be exactly (n-1) edges to connect all the vertices using Kruskal's algorithm

PRIM'S ALGORITHM	KRUSKAL'S ALGORITHM
• Starts with any vertex of the graph arbitrarily. At no point of time, a forest is encountered in Prim's algorithm.	• Starts with all the vertices of the graph as a forest and every addition of edge takes the forest one step further towards a complete tree.
• The concept of Prim's algorithm is based on the connectedness.	• The concept of Kruskal's algorithm is based on the acyclic nature of the graph.
• Saves a lot of space as no additional structures are used for storing the edges.	• More time is saved as the edges are sorted first.
• Always a new vertex	• Adding of an edge is

is joined to an old vertex.

- Addition of nodes is based on the concept of shortest distance or weight.

performed by selecting the sorted edge.

- Next edge is always determined by the edge list.