

1.5 MINIMUM COST SPANNING TREE (MCST) PROBLEM

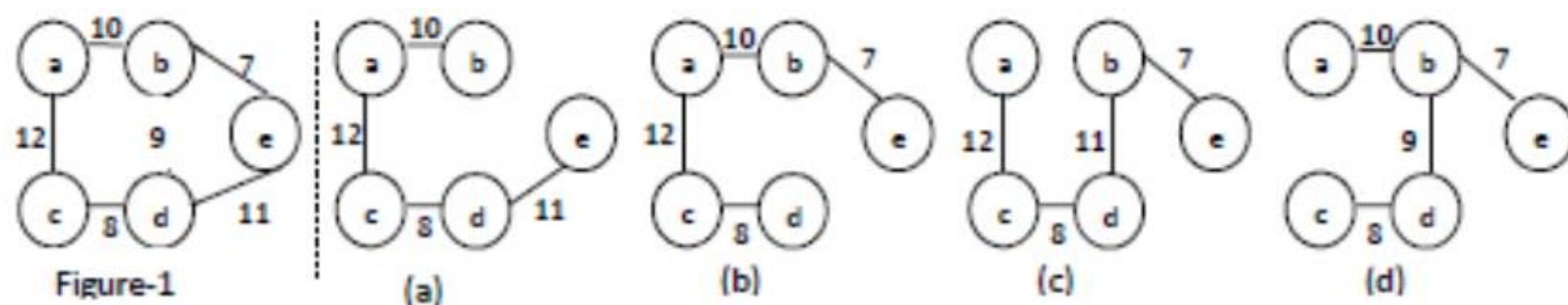
Definition: (Spanning tree): Let $G=(V,E)$ be an undirected connected graph. A *subgraph* $T=(V,E')$ of G is a spanning tree of G if and only if T is a tree (i.e. no cycle exist in T) and contains all the vertices of G .

Definition: (Minimum cost Spanning tree):

Suppose G is a weighted connected graph. A *weighted graph* is one in which every edge of G is assigned some positive weight (or length). A graph G is having several spanning tree.

In general, a *complete graph* (each vertex in G is connected to every other vertices) with n vertices has total n^{n-2} spanning tree. For example, if $n=4$ then total number of spanning tree is 16.

A *minimum cost spanning tree* (MCST) of a weighted connected graph G is that spanning tree whose sum of length (or weight) of all its edges is minimum, among all the possible spanning tree of G .



To find a MCST of a given graph G , one of the following algorithms is used:

1. Kruskal's algorithm
2. Prim's algorithm

These two algorithms use Greedy approach. A greedy algorithm selects the edges one-by-one in some given order. The next edge to include is chosen according to some optimization criteria. The simplest such criteria would be to choose an edge (u, v) that results in a minimum increase in the sum of the costs (or weights) of the edges so far included.

In General for constructing a MCST:

- We will build a set A of edges that is always a subset of some MCST.
- Initially, A has no edges (i.e. empty set).
- At each step, an edge (u, v) is determined such that $A \cup \{(u, v)\}$ is also a subset of a MCST. This edge (u, v) is called a *safe edge*.
- At each step, we always add only *safe edges* to set A .
- Termination: when all *safe edges* are added to A , we stop. Now A contains a edges of spanning tree that is also an MCST.

Thus a general MCST algorithm is:

```

GENERIC_MCST( $G, w$ )
{
     $A \leftarrow \phi$ 
    While  $A$  is not a spanning tree
    {
        find an edge  $(u, v)$  that is safe for  $A$ 
         $A \leftarrow A \cup \{(u, v)\}$ 
    }
    return  $A$ 
}

```

For solving MCST problem using Greedy algorithm, we uses the following data structure and functions, as mentioned earlier.

- C:** The set of candidates (or given values): Here $C=E$, the set of edges of $G(V, E)$.
 - S:** Set of selected candidates (or input) which is used to give optimal solution. Here the subset of edges, E' (i.e. $E' \subseteq E$) is a solution, if the graph $T(V, E')$ is a spanning tree of $G(V, E)$.
 - In case of MCST problem, the function *Solution* checks whether a solution is reached or not. This function basically checks :
 - All the edges in S form a tree.
 - The set of vertices of the edges in S equal to V .
 - The sum of the weights of the edges in S is minimum possible of the edges which satisfy (a) and (b) above.
- 1) If selection function (say *select*) which chooses the best candidate from C to be added to the solution set S ,

- iv) The *select* function chooses the best candidate from C . In case of Kruskal's algorithm, it selects an edge, whose length is smallest (from the remaining candidates). But in case of Prim's algorithm, it select a vertex, which is added to the already selected vertices, to minimize the cost of the spanning tree.
- v) A function *feasible* checks the feasibility of the newly selected candidate (i.e. edge (u,v)). It checks whether a newly selected edge (u, v) form a cycle with the earlier selected edges. If answer is "yes" then the edge (u,v) is rejected, otherwise an edge (u,v) is added to the solution set S .
- vi) Here the objective function $ObjF$ gives the sum of the edge lengths in a Solution.

1.5.1 Kruskal's Algorithm

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

Kruskal's algorithm finds a minimum-cost spanning tree (MCST) of a given graph G . It uses a *greedy approach* to find MCST, because at each step it adds an edge of least possible weight to the set A . In this algorithm,

- First we examine the edges of G in order of increasing weight.
- Then we select an edge $(u, v) \in E$ of minimum weight and checks whether its end points belongs to same component or different connected components.
- If u and v belongs to different connected components then we add it to set A , otherwise it is rejected because it create a cycle.
- The algorithm stops, when only one connected components remains (i.e. all the vertices of G have been reached).

Following pseudo-code is used to constructing a MCST, using Kruskal's algorithm:

```

KRUSKAL_MCST( $G, w$ )
    /* Input: A undirected connected weighted graph  $G=(V, E)$ .
    /* Output: A minimum cost spanning tree  $T(V, E')$  of  $G$ 
    {
1.  Sort the edges of  $E$  in order of increasing weight
2.   $A \leftarrow \phi$ 
3.  for (each vertex  $v \in V[G]$ )
4.      do MAKE_SET( $v$ )
5.  for (each edge  $(u, v) \in E$ , taken in increasing order of weight
        {
6.      if (FIND_SET( $u$ )  $\neq$  FIND_SET( $v$ ))
7.           $A \leftarrow A \cup \{(u, v)\}$ 
8.          MERGE( $u, v$ )
        }
9.  return  $A$ 
    }
```

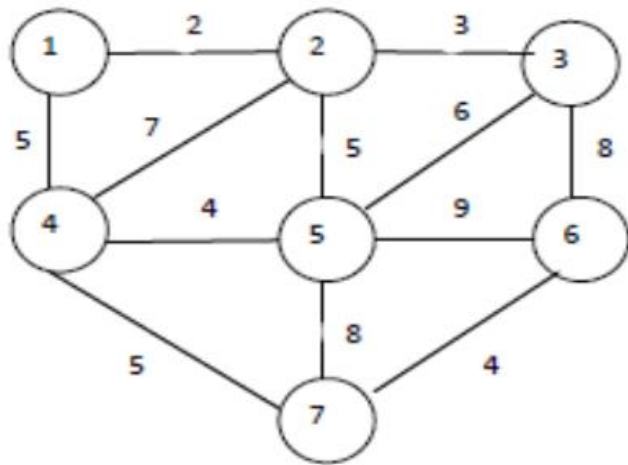
Kruskal's algorithm works as follows:

- First, we sorts the edges of E in order of increasing weight
- We build a set A of edges that contains the edges of the MCST. Initially A is empty.
- At line 3-4, the function **MAKE_SET**(v), make a new set $\{v\}$ for all vertices of G . For a graph with n vertices, it makes n components of disjoint set such as $\{1\}, \{2\}, \dots$ and so on.
- In line 5-8: An edge $(u, v) \in E$, of minimum weight is added to the set A , if and only if it joins two nodes which belongs to *different* components (to

Example: Apply Kruskal's algorithm on the following graph to find minimum-cost-spanning –
tree (MCST).

Solution: First, we sorts the edges of $G=(V,E)$ in order of increasing weights as:

Edges	(1,2)	(2,3)	(4,5)	(6,7)	(1,4)	(2,5)	(4,7)	(3,5)	(2,4)	(3,6)	(5,7)	(5,6)
weights	2	3	4	4	5	5	5	6	7	8	8	9



- check this we use a *FIND_SET()* function, which returns a same integer value, if u and v belongs to same components (In this case adding (u,v) to A creates a cycle), otherwise it returns a different integer value)
- If an edge added to A then the two components containing its end points are merged into a single component.
 - Finally the algorithm stops, when there is just a single component.

kal's Algorithm proceeds as follows:

	GE DERED	CONNECTED COMPONENTS	SPANNING FORESTS (A)
Initialization	—	{1} {2} {3} {4} {5} {6} {7} (using line 3-4)	(1) (2) (3) (4) (5) (6) (7)
1.	(1, 2)	{1, 2}, {3}, {4}, {5}, {6}, {7}	(1)–(2) (3) (4) (5) (6) (7)
2.	(2, 3)	{1,2,3}, {4}, {5}, {6}, {7}	(1)–(2)–(3) (4) (5) (6) (7)
3.	(4, 5)	{1,2,3}, {4,5}, {6}, {7}	(1)–(2)–(3) (6) (7) (4)–(5)
4.	(6, 7)	{1,2,3}, {4,5}, {6,7}	(1)–(2)–(3) (4)–(5) (6) (7)
5.	(1, 4)	{1,2,3,4,5}, {6,7}	(1)–(2)–(3) (4)–(5) (6) (7)
6.	(2, 5)	Edge (2,5) is rejected, because its end point belongs to same connected component, so create a cycle.	
7.	(4, 7)	{1,2,3,4,5,6,7}	(1)–(2)–(3) (4)–(5) (6) (7)

Time Cmplxity
 $O(E \log |E|)$

Total Cost of Spanning tree, $T = 2+3+5+4+5+4=23$

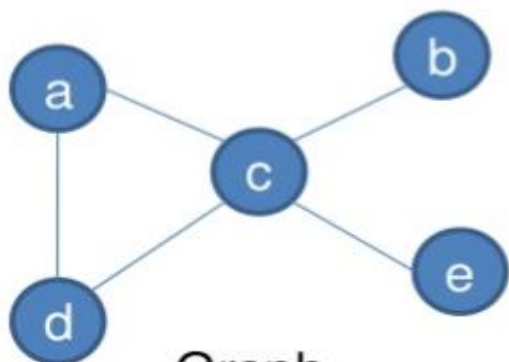
PRIM'S ALGORITHM

- Prim's algorithm is a greedy algorithm that finds a minimum spanning tree for a weighted undirected graph.

- Developed in 1930 by Czech mathematician Vojtěch Jarník and later rediscovered and republished by computer scientists Robert C. Prim in 1957 and E. W. Dijkstra in 1959.

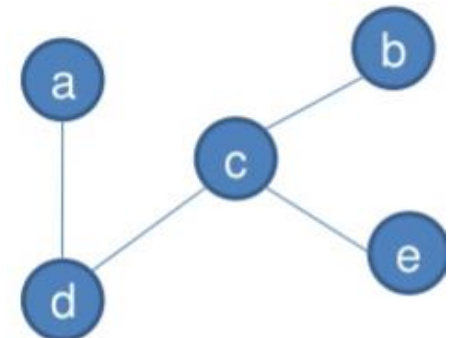
- Therefore, it is also sometimes called the **DJP algorithm**, **Jarník's algorithm**, the **Prim–Jarník algorithm**, or the **Prim–Dijkstra algorithm**.

Spanning Trees: A subgraph T of a undirected graph $G = (V, E)$ is a spanning tree of G if it is a tree and contains every vertex of G .



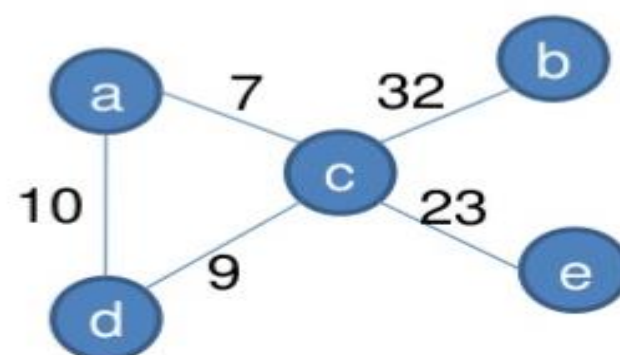
Graph

- Every connected graph has a spanning tree.
- May have multiple spanning tree.
- For example see this graph.

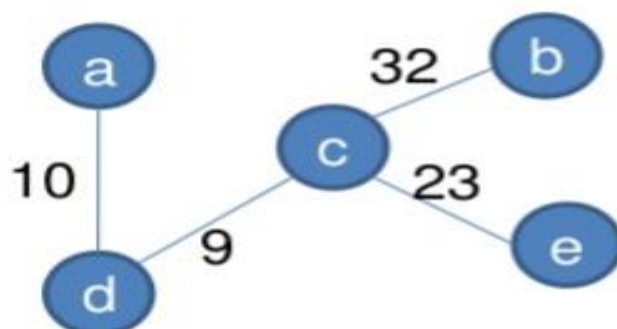


Spanning Tree 1

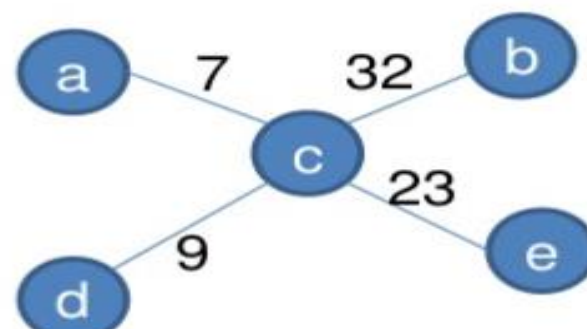
Minimum Spanning Tree in an undirected connected weighted graph is a spanning tree of minimum weight. Example:



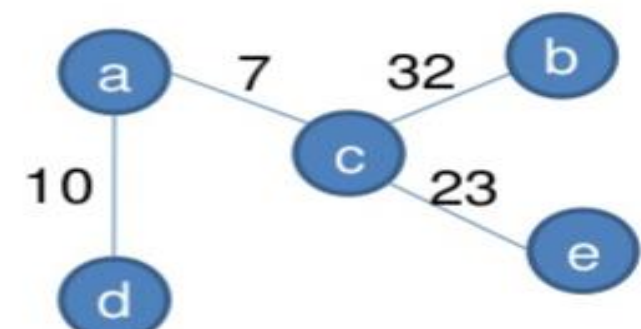
Weighted Graph



Spanning Tree 1,
 $w=74$



Spanning Tree 2,
 $w=71$
(Minimum Spanning Tree)



Spanning Tree 3,
 $w=72$

GENERIC-MST Algorithm

GENERIC-MST (G, w)

1	$A = \emptyset$
2	while A does not form a spanning tree
3	find an edge (u, v) that is safe for A
4	$A = A \cup \{ (u, v) \}$
5	return A

- The idea is to start with an empty graph and try to add edges one at a time, always making sure that what is built remains acyclic.

- Gives us an idea how to grow a MST.

- An edge (u, v) is safe for A if and only if $A \cup \{(u, v)\}$ is also a subset of some MST

PRIM's Algorithm

MST-PRIM(G, w, r)

1	for each $u \in V[G]$
2	do $key[u] \leftarrow \infty$
3	$\Pi[u] \leftarrow NIL$
4	$key[r] \leftarrow 0$
5	$Q \leftarrow V[G]$
6	while Q is not Empty
7	do $u \leftarrow \text{EXTRACT-MIN}(Q)$
8	for each $v \in \text{Adj}[u]$
9	do if $v \in Q$ and $w(u, v) < key[v]$
10	then $\Pi[v] \leftarrow u$
11	$key[v] \leftarrow w(u, v)$

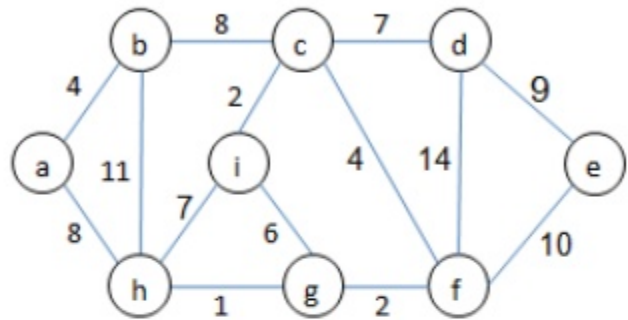
- A special case of generic minimum-spanning-tree algorithm and operates much like Dijkstra's algorithm.
 - Edges in the set A always form a single tree.
 - Greedy algorithm since at each step it adds to the tree an edge that contributes the minimum amount possible to the tree's weight.
 - Connected graph G and the root r of the MST to be drawn are inputs.
 - During execution of the algorithm, all vertices that are not in the MST reside in a min-priority queue Q based on a key attribute.
 - For each vertex v, the attribute v.key is the minimum weight of any edge connecting v to a vertex in the tree.
 - The attribute v.Π names parent of v in the tree.
 - Maintains the set A from GENERIC-MST as $A = \{ (v, v.\Pi) : v \in V - \{r\} - Q \}$.
- When the algorithm terminates, the min-priority queue Q is empty.
- The MST A for G is thus
- $A = \{ (v, v.\Pi) : v \in V - \{r\} \}$.

PRIM's Algorithm (Steps 1-5 : Initialization)

MST-PRIM(G,w,r)

1	for each $u \in V[G]$	} Initialization	
2	do $key[u] \leftarrow \infty$		
3	$\Pi[u] \leftarrow NIL$		
4	$key[r] \leftarrow 0$		
5	$Q \leftarrow V[G]$		
6	while Q is not Empty		
7	do $u \leftarrow EXTRACT-MIN(Q)$		
8	for each $v \in Adj[u]$		
9	do if $v \in Q$ and $w(u,v) < key[v]$		
10	then $\Pi[v] \leftarrow u$		
11	$key[v] \leftarrow w(u, v)$		

Example Graph



u	a	b	c	d	e	f	g	h	i
key[u]	∞	∞	∞	∞	∞	∞	∞	∞	∞
$\Pi[u]$	NIL	NIL	NIL	NIL	NIL	NIL	NIL	NIL	NIL

After Steps 1-3

u	a	b	c	d	e	f	G	H	i
key[u]	0	∞	∞	∞	∞	∞	∞	∞	∞
$\Pi[u]$	NIL	NIL	NIL	NIL	NIL	NIL	NIL	NIL	NIL

After Step 4

Q	A	b	c	d	e	f	g	h	i
A	EMPTY								

After Step 5

Before Step 6

Q	a	b	c	d	e	f	g	h	i
key[u]	0	∞	∞	∞	∞	∞	∞	∞	∞
$\Pi[u]$	NIL	NIL	NIL	NIL	NIL	NIL	NIL	NIL	NIL

Steps 6-11 (for u=a)

u	v	$v \in Q$ AND $w(u,v) < Key[v]$	$\Pi[v] \leftarrow u,$ $Key[v] \leftarrow w(u,v)$
a	b	YES	$\Pi[b] \leftarrow a,$ $Key[b] \leftarrow 4$
a	h	YES	$\Pi[h] \leftarrow a,$ $Key[h] \leftarrow 8$

After Step 6-11 (for u=a)

Q	a	b	c	d	e	f	g	h	i
key[u]	0	4	∞	∞	∞	∞	∞	8	∞
$\Pi[u]$	NIL	a	NIL	NIL	NIL	NIL	NIL	a	NIL

Q	a	b	c	d	E	f	g	h	i
A	EMPTY								

Q	b	c	d	e	f	g	h	i	
A	a								

2

Status of Q before using u=b

Q	a	b	c	d	e	f	g	h	i
key[u]	0	4	∞	∞	∞	∞	∞	8	∞
Π[u]	NIL	a	NIL	NIL	NIL	NIL	NIL	a	NIL

Q	b	c	d	e	f	g	h	i
A	a							

Steps 6-11(for u=b)

u	v	v ∈ Q AND w(u, v) < Key[v]	Then Π[v] ← u, Key[v] ← w(u, v)
b	c	YES	Π[c] ← b, Key[c] ← 8
b	h	NO	do nothing
b	a	NO	do nothing

5

Status of Q before using u=f

Q	a	b	c	d	e	f	g	h	i
key[u]	0	4	8	7	∞	4	6	7	2
Π[u]	NIL	a	b	c	NIL	c	i	i	c

Q	d	e	f	g	h
A	a	b	c	i	

Steps 6-11(for u=f)

u	v	v ∈ Q AND w(u, v) < Key[v]	Then Π[v] ← u, Key[v] ← w(u, v)
f	d	NO	Do nothing
f	e	YES	Π[e] ← f, Key[e] ← 10
f	g	YES	Π[g] ← f, Key[g] ← 6
f	c	NO	Do nothing

8

Status of Q before using u=d

Q	a	b	c	d	e	f	g	h	i
key[u]	0	4	8	7	10	4	2	7	2
Π[u]	NIL	a	b	c	f	c	f	i	c

Q	d	e
A	a	b

Steps 6-11(for u=d)

u	v	v ∈ Q AND w(u, v) < Key[v]	Then Π[v] ← u, Key[v] ← w(u, v)
d	c	NO	Do Nothing
d	f	NO	Do Nothing
d	e	YES	Π[e] ← d, Key[e] ← 9

3

Status of Q before using u=c

Q	a	b	c	d	e	f	g	h	i
key[u]	0	4	8	∞	∞	∞	∞	8	∞
Π[u]	NIL	a	b	NIL	NIL	NIL	NIL	a	NIL

Q	c	d	e	f	g	h	i
A	a	b					

Steps 6-11(for u=c)

u	v	v ∈ Q AND w(u, v) < Key[v]	Then Π[v] ← u, Key[v] ← w(u, v)
c	i	YES	Π[i] ← c, Key[i] ← 2
c	f	YES	Π[f] ← c, Key[f] ← 4
c	d	YES	Π[d] ← c, Key[d] ← 7
c	b	NO	Do Nothing

6

Status of Q before using u=g

Q	a	b	c	d	e	f	g	h	i
key[u]	0	4	8	7	10	4	2	7	2
Π[u]	NIL	a	b	c	f	c	f	i	c

Q	d	e	g	h
A	a	b	c	i

Steps 6-11(for u=g)

u	v	v ∈ Q AND w(u, v) < Key[v]	Then Π[v] ← u, Key[v] ← w(u, v)
g	h	YES	Π[h] ← g, Key[h] ← 1
g	i	NO	Do Nothing
g	f	NO	Do Nothing

9

Status of Q before using u=e

Q	a	b	c	d	e	f	g	h	i
key[u]	0	4	8	7	9	4	2	1	2
Π[u]	NIL	a	b	c	d	c	f	g	c

Q	e
A	a

Steps 6-11(for u=e)

u	v	v ∈ Q AND w(u, v) < Key[v]	Then Π[v] ← u, Key[v] ← w(u, v)
e	d	NO	Do Nothing
e	f	NO	Do Nothing

4

Status of Q before using u=i

Q	a	b	c	d	e	f	g	h	i
key[u]	0	4	8	7	∞	4	∞	8	2
Π[u]	NIL	a	b	c	NIL	c	NIL	a	c

Q	d	e	f	g	h	i
A	a	b	c			

Steps 6-11(for u=i)

u	v	v ∈ Q AND w(u, v) < Key[v]	Then Π[v] ← u, Key[v] ← w(u, v)
i	h	YES	Π[h] ← i, Key[h] ← 7
i	g	YES	Π[g] ← i, Key[g] ← 6
i	c	NO	Do Nothing

7

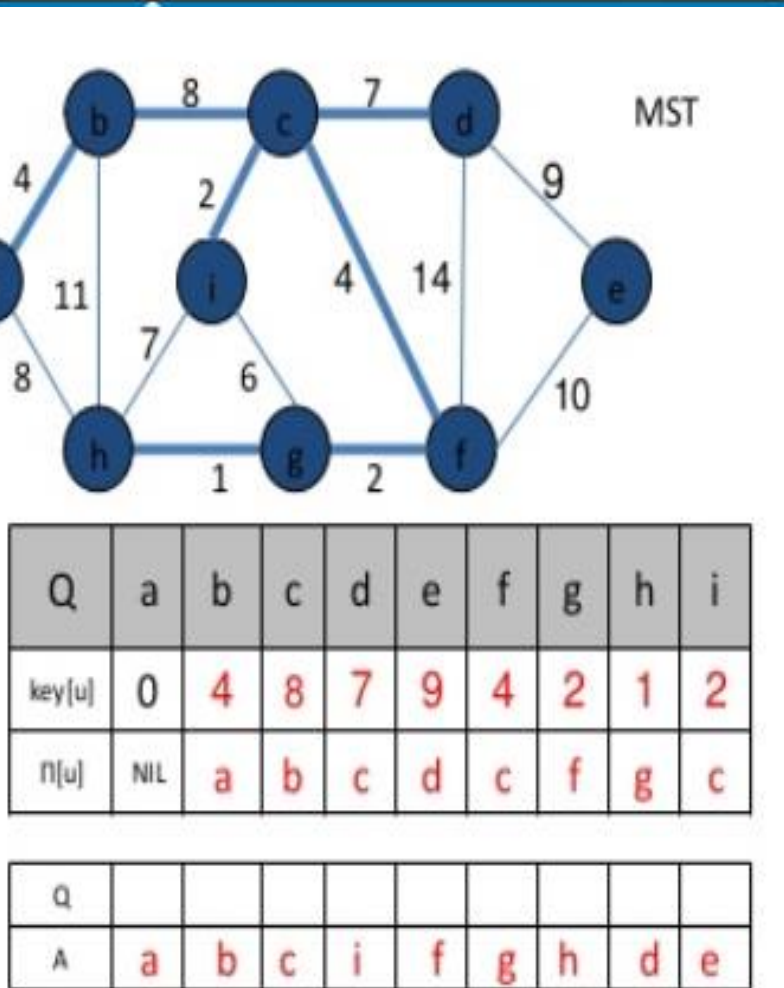
Status of Q before using u=h

Q	a	b	c	d	e	f	g	h	i
key[u]	0	4	8	7	10	4	2	7	2
Π[u]	NIL	a	b	c	f	c	f	i	c

Q	d	e	h
A	a	b	c

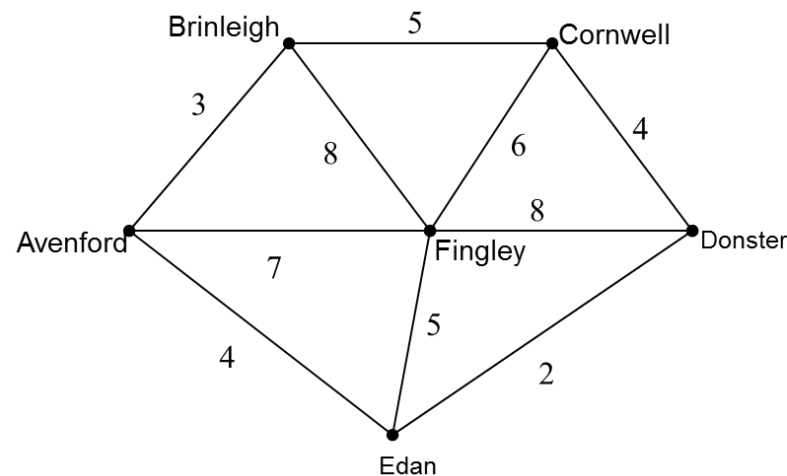
Steps 6-11(for u=h)

u	v	v ∈ Q AND w(u, v) < Key[v]	Then Π[v] ← u, Key[v] ← w(u, v)
h	a	NO	Do Nothing
h	b	NO	Do Nothing
h	i	NO	Do Nothing
h	g	NO	Do Nothing



Prim's algorithm with an Adjacency Matrix

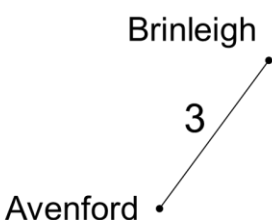
A cable company want to connect five villages to their network which currently extends to the market town of Avenford. What is the minimum length of cable needed?



	A	B	C	D	E	F
A	-	3	-	-	4	7
B	3	-	5	-	-	8
C	-	5	-	4	-	6
D	-	-	4	-	2	8
E	4	-	-	2	-	5
F	7	8	6	8	5	-

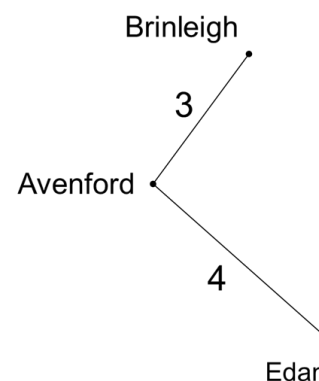
- Start at vertex A. Label column A "1" .
- Delete row A
- Select the smallest entry in column A (AB, length 3)

	1	A	B	C	D	E	F
A	-	3	-	-	-	4	7
B	3	-	5	-	-	-	8
C	-	5	-	4	-	-	6
D	-	-	4	-	2	-	8
E	4	-	-	-	2	-	5
F	7	8	6	8	5	-	-



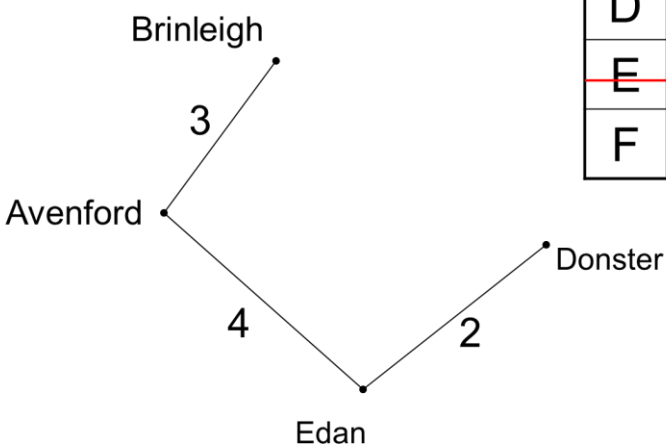
- Label column B "2"
- Delete row B
- Select the smallest uncovered entry in either column A or column B (AE, length 4)

	1	2	A	B	C	D	E	F
A	-	3	-	-	-	-	4	7
B	3	-	5	-	-	-	-	8
C	-	5	-	4	-	-	-	6
D	-	-	4	-	2	-	-	8
E	4	-	-	-	2	-	-	5
F	7	8	6	8	5	-	-	-



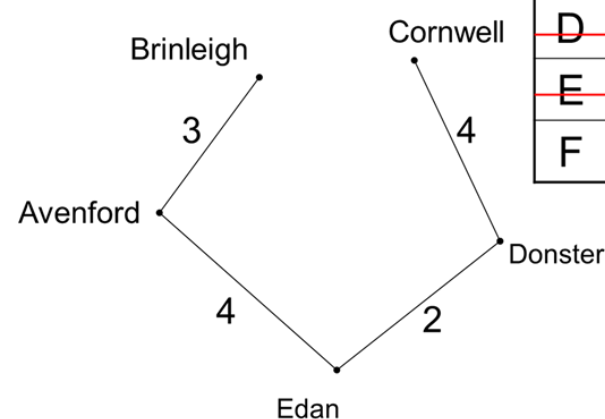
- Label column E "3"
- Delete row E
- Select the smallest uncovered entry in either column A, B or E (ED, length 2)

	1	2	3	A	B	C	D	E	F
A	-	3	-	-	-	-	-	4	7
B	3	-	5	-	-	-	-	-	8
C	-	5	-	4	-	-	-	-	6
D	-	-	4	-	-	2	-	-	8
E	4	-	-	-	2	-	-	-	5
F	7	8	6	8	5	-	-	-	-



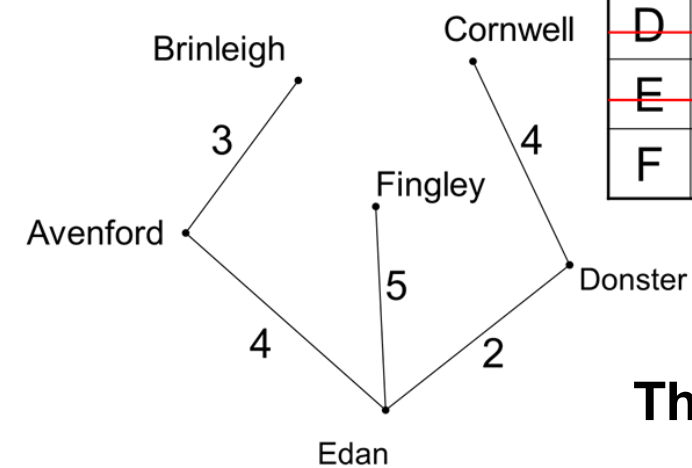
- Label column D "4"
- Delete row D
- Select the smallest uncovered entry in either column A, B, D or E (DC, length 4)

	1	2	4	3	A	B	C	D	E	F
A	-	3	-	-	-	-	-	-	4	7
B	3	-	5	-	-	-	-	-	-	8
C	-	5	-	4	-	-	-	-	-	6
D	-	-	4	-	2	-	-	-	-	8
E	4	-	-	-	2	-	-	-	-	5
F	7	8	6	8	5	-	-	-	-	-



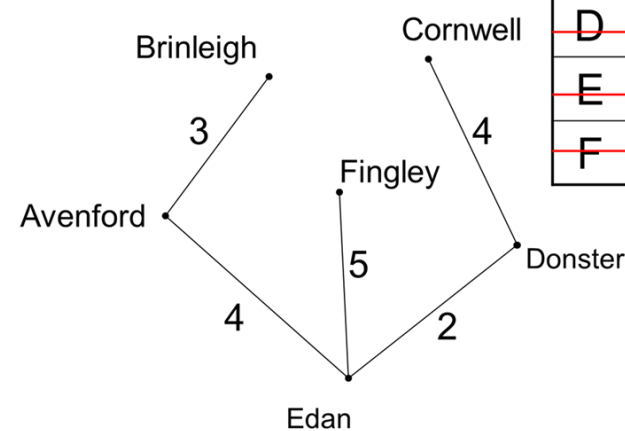
- Label column C "5"
- Delete row C
- Select the smallest uncovered entry in either column A, B, D, E or C (EF, length 5)

	1	2	5	4	3	A	B	C	D	E	F
A	-	3	-	-	-	-	-	-	-	4	7
B	3	-	5	-	-	-	-	-	-	-	8
C	-	5	-	4	-	-	-	-	-	-	6
D	-	-	4	-	2	-	-	-	-	-	8
E	4	-	-	-	2	-	-	-	-	-	5
F	7	8	6	8	5	-	-	-	-	-	-



- FINALLY
- Label column F "6"
- Delete row F

	1	2	5	4	3	6	A	B	C	D	E	F
A	-	3	-	-	-	-	-	-	-	-	4	7
B	3	-	5	-	-	-	-	-	-	-	-	8
C	-	5	-	4	-	-	-	-	-	-	-	6
D	-	-	4	-	2	-	-	-	-	-	-	8
E	4	-	-	-	2	-	-	-	-	-	-	5
F	7	8	6	8	5	-	-	-	-	-	-	-






The spanning tree is shown in the diagram

$$\text{Length } 3 + 4 + 4 + 2 + 5 = 18\text{Km}$$

Complexity Analysis of Prim's algorithm

MST-PRIM(G, w, r)

1	for each $u \in V[G]$		Total time: $O(V \lg V + E \lg V) = O(E \lg V)$
2	do $key[u] \leftarrow \infty$		
3	$\Pi[u] \leftarrow NIL$		
4	$key[r] \leftarrow 0$		
5	$Q \leftarrow V[G]$		
6	while Q is not Empty		Body of while loop is executed V times. Takes $O(\lg V)$ times. Takes $O(V \lg V)$ times.
7	do $u \leftarrow \text{EXTRACT-MIN}(Q)$		
8	for each $v \in \text{Adj}[u]$		
9	do if $v \in Q$ and $w(u, v) < key[v]$		Constant Executed $O(E)$ times total. $O(E \lg V)$
10	then $\Pi[v] \leftarrow u$		
11	$key[v] \leftarrow w(u, v)$		

Takes $O(\lg V)$ times.

Total time:

This time complexity can be improved and reduced to $O(E + V \log V)$ using Fibonacci heap.

Prim's Algorithm	Kruskal's Algorithm
The tree that we are making or growing always remains connected.	The tree that we are making or growing usually remains disconnected.
Prim's Algorithm grows a solution from a random vertex by adding the next cheapest vertex to the existing tree.	Kruskal's Algorithm grows a solution from the cheapest edge by adding the next cheapest edge to the existing tree / forest.
Prim's Algorithm is faster for dense graphs.	Kruskal's Algorithm is faster for sparse graphs.
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 forest ,every addition of edge takes a forest one step further towards a complete tree
Based on acyclic graphs	Based on connectedness
Time Complexity: $O(E \lg V)$	Time Complexity : $O(E \log E)$
Addition of Nodes is based on shortest distance weight	Next Edge is determined by the edge list