Graph Algorithms-II

Graph Algorithms

Topics

- *Spanning Trees*
- *Minimum Spanning Trees*
- *Kruskal's Algorithm for MST and Analysis*
- *Prim's Algorithm and Analysis*

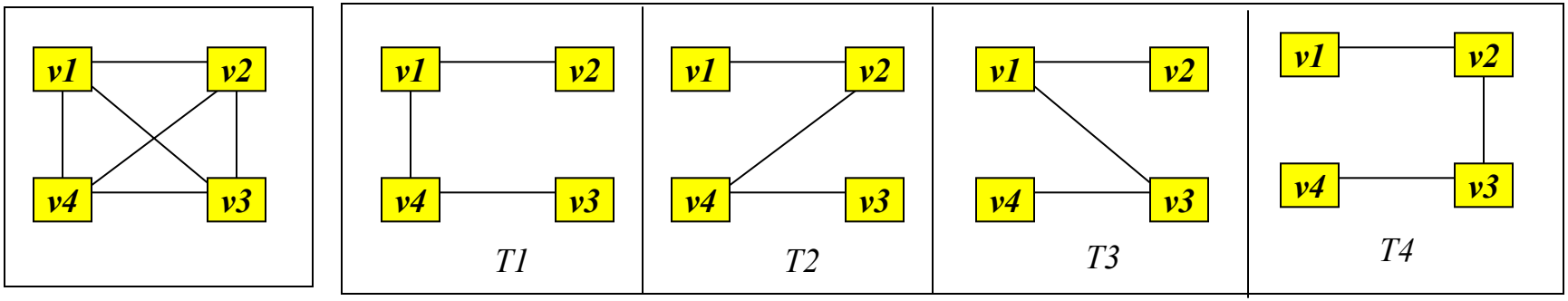
Spanning Trees

Spanning Trees

Definition

A *spanning tree* for an undirected graph is a sub-graph which includes *all vertices* but has *no cycles*.

Example: There can be *several spanning trees* for a graph. Figure *(i)* shows a sample graph. Figure *(ii)* depicts some of the trees for the graph



(i) Sample Graph

(ii) Spanning Trees

- Each spanning tree includes *all* the four vertices *(v1,v2,v3,v4)* of the parent graph

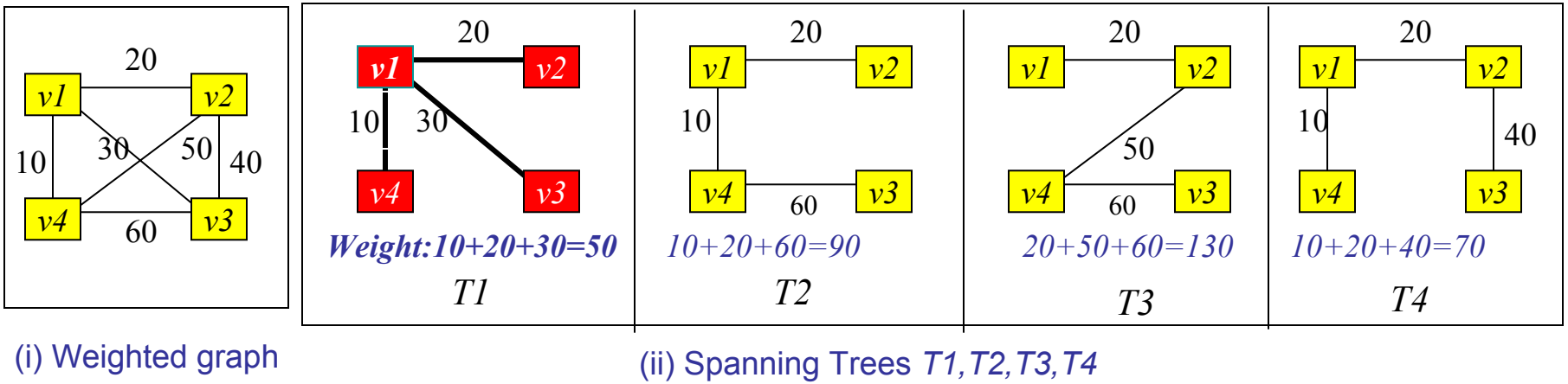
- Spanning trees can be generated by *depth-first–search* and *breadth-first-search* procedures.

Minimum Spanning Tree

Definition

A *weighted undirected graph* can have several spanning trees. One of the spanning trees has *smallest sum of all the weights* associated with the edges. This tree is called *minimum spanning tree* (*mst*).

Example **:**Figure shows a sample weighted graph, and some of the *spanning trees, with total weight of edges for each tree.* The tree *T1,* with smallest total weight is the minimum spanning tree. It is shown with vertices colored red



(i) Weighted graph

(ii) Spanning Trees *T1,T2,T3,T4*

Minimum Spanning Trees

Applications

Minimum spanning trees have many practical applications. Some typical examples are:
:

- *A telephone network can be configured, using minimum spanning tree, so that minimum cable length is used.*
- *The air travel routes can be selected so that the travel time or travel cost is least.*
- *A computer network can be set up with minimum routing distance*
- *Linking a group of islands with bridges so that total bridge span length is minimum*

➢ Two important algorithms for creating a minimum spanning tree for a weighted graph, are *Kruskal's algorithm* and *Prim's algorithm,* named after their inventors.

Kruskal's Algorithm

Minimum Spanning Tree

Kruskal's Algorithm

The *Kruskal 's algorithm* works as follows:

Step # 1: *Remove all edges of the graph*

Step #2: *Arrange edges according to their weights*

Step # 3: *Select an edge with least weight*

Step #4: *Attach the edge to the corresponding vertices, if it does not form a cycle; otherwise, drop the edge*

Step #5: *Repeat steps 3 to 4 until all the edges have been processed (added or dropped)*

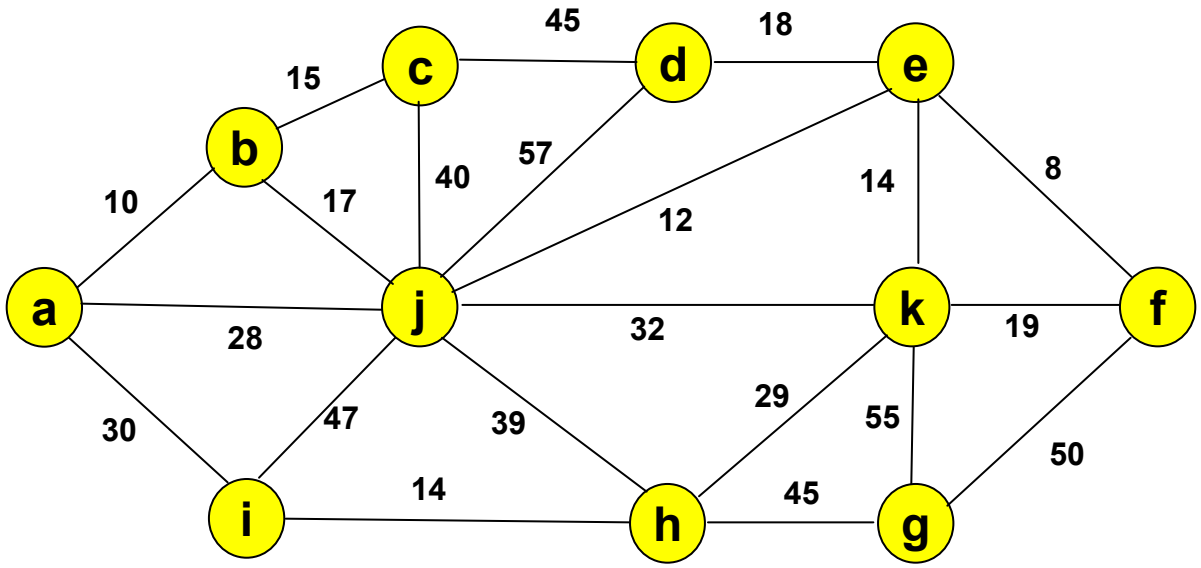
- Kruskal's algorithm is categorized as ***greedy***, because at each step it picks an edge with ***smallest weight***.

- The algorithm can be implemented in several ways. Generally, a ***priority queue*** is used to store graph edges, so that, starting with the smallest weight , edges are extracted in order of their increasing weights.

Kruskal's Algorithm

Example

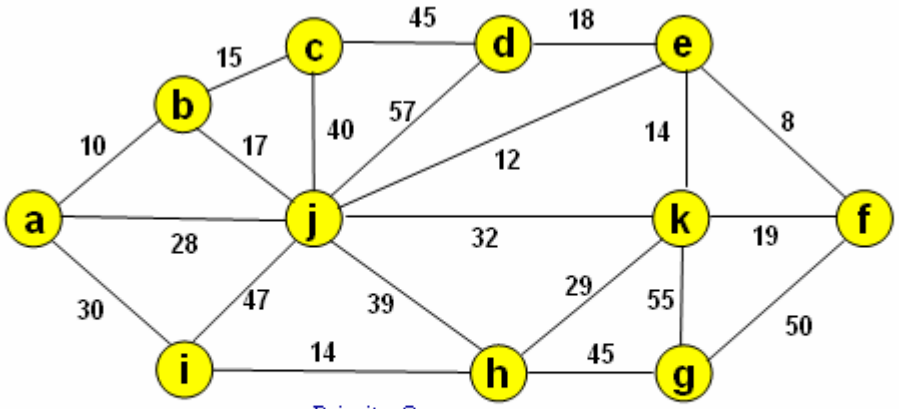
In order to examine the working of Kruskal's algorithm for building a ***minimum spanning tree*** consider the sample ***weighted undirected graph*** shown in the figure below. The number attached to the edges are the weights.



- The steps for growing the minimum spanning tree are elaborated in the next set of figures

Kruskal's Algorithm Example

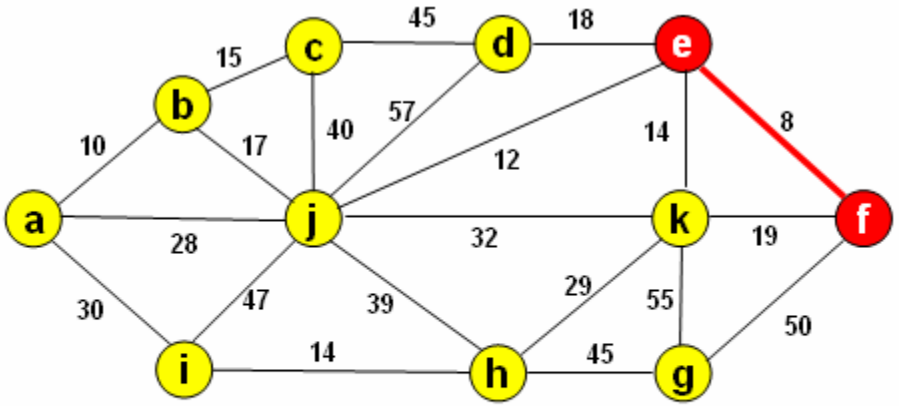
(1)In order to extract edges with increasing weights, the vertices forming edges of the graph and the corresponding weights are stored in a ***priority queue***. In the figure, the edges are shaded green, and the corresponding weights are shaded red. The ***tree set T is empty***.



Priority Queue

8	10	12	14	14	15	17	18	19	28	29	30	32	39	40	45	45	47	50	55	57
e	a	e	e	h	b	b	d	f	a	h	a	j	h	c	c	g	i	f	g	d
f	b	j	k	i	c	j	e	k	j	k	i	k	j	j	d	h	j	g	k	j

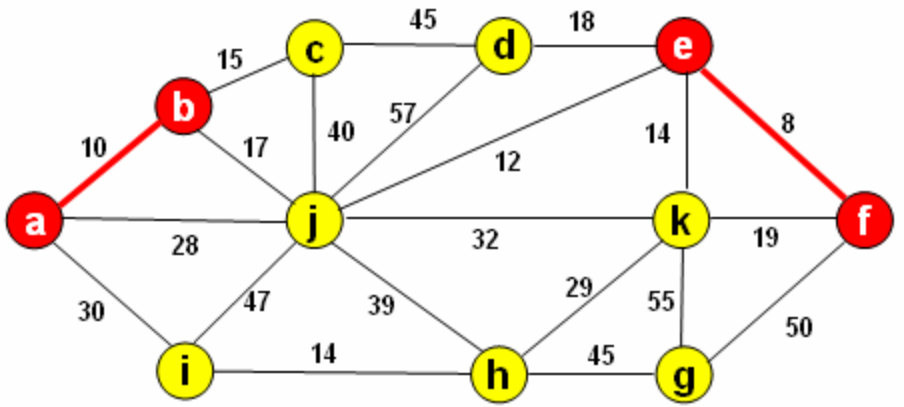
(2) The edge (*e,f,8*)is extracted from the priority queue. It is added to the spanning tree. In the diagram an added edge is shown in red color. The corresponding vertices have red background . At this stage, the tree set is ***T={(e,f,8)}***



8	10	12	14	14	15	17	18	19	28	29	30	32	39	40	45	45	47	50	55	57
e	a	e	e	h	b	b	d	f	a	h	a	j	h	c	c	g	i	f	g	d
f	b	j	k	i	c	j	e	k	j	k	i	k	j	j	d	h	j	g	k	j

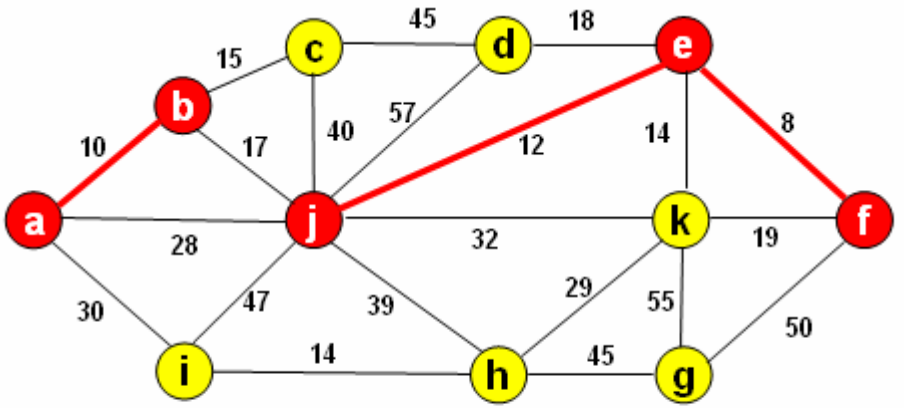
Kruskal's Algorithm Example

(3) The edge *(a,b,10)*is extracted from the priority queue. It does not form a cycle with the existing tree edge. The edge is added to the spanning tree. The total weight of edges is *18*. The tree set is ***T={(e,f,8), (a,b,10)}***



	10	12	14	14	15	17	18	19	28	29	30	32	39	40	45	45	47	50	55	57
	a	e	e	h	b	b	d	f	a	h	a	j	h	c	c	g	i	f	g	d
	b	j	k	i	c	j	e	k	j	k	i	k	j	j	d	h	j	g	k	j

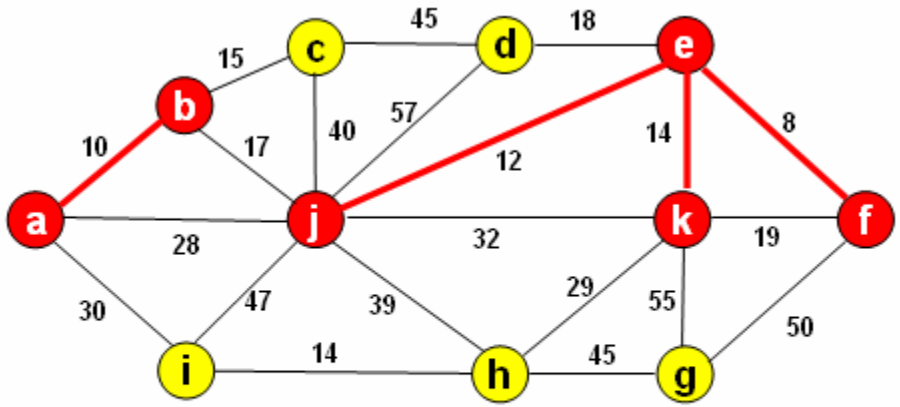
(4) The edge (*e,j,12*)is extracted from the priority queue. It does not form a cycle with any of the tree edges. The edge is added to the spanning tree. The total weight of edges is *30*. The tree set is ***T={(e,f,8), (a,b,10), (e,j,12)}***



		12	14	14	15	17	18	19	28	29	30	32	39	40	45	45	47	50	55	57
		e	e	h	b	b	d	f	a	h	a	j	h	c	c	g	i	f	g	d
		j	k	i	c	j	e	k	j	k	i	k	j	j	d	h	j	g	k	j

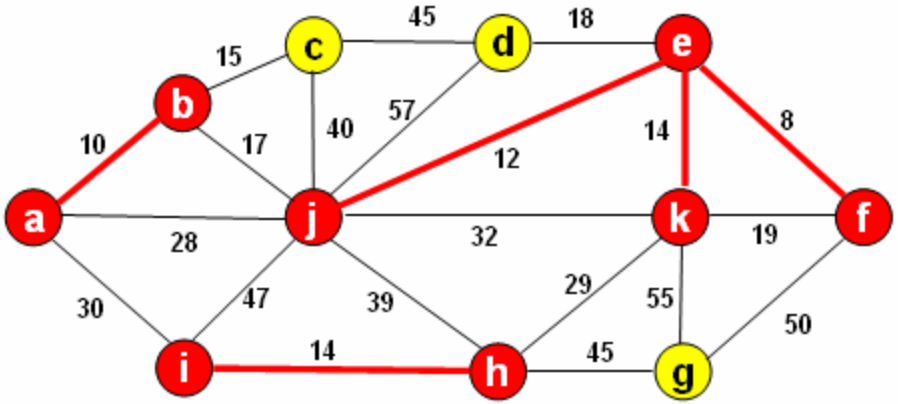
Kruskal's Algorithm Example

(5) Edge (*e,k,14*)is extracted from the priority queue. It does not form a cycle with any of the existing edges of the tree. The edge is added to the spanning tree. The accumulative weight of edges is *44*. The tree set is ***T={(e,f,8), (a,b,10), (e,j,12), (e,k,14)}***



			14	14	15	17	18	19	28	29	30	32	39	40	45	45	47	50	55	57
			e	h	b	b	d	f	a	h	a	j	h	c	c	g	i	f	g	d
			k	i	c	j	e	k	j	k	i	k	j	j	d	h	j	g	k	j

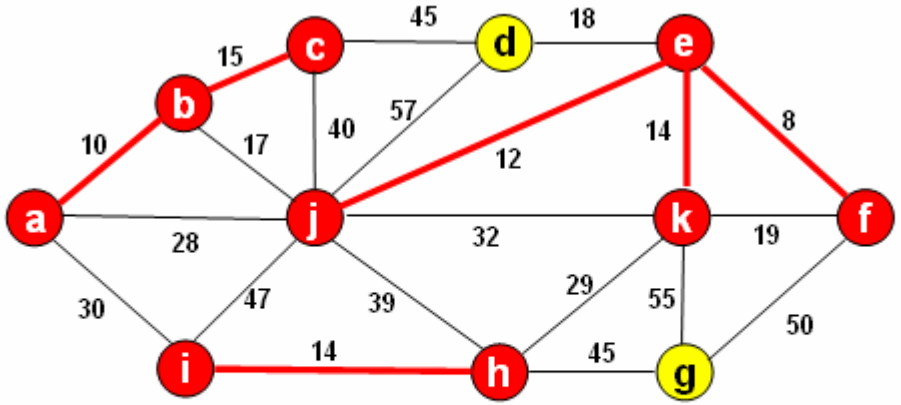
(6) Edge (*h,i,14*)is extracted from the priority queue. It does not form cycle with any of the tree edges. The edge is added to the spanning tree. The total weight of edges is *58*. The tree set is ***T={(e,f,8), (a,b,10), (e,j,12), (e,k,14), (h,i,14)}***



				14	15	17	18	19	28	29	30	32	39	40	45	45	47	50	55	57
				h	b	b	d	f	a	h	a	j	h	c	c	g	i	f	g	d
				i	c	j	e	k	j	k	i	k	j	j	d	h	j	g	k	j

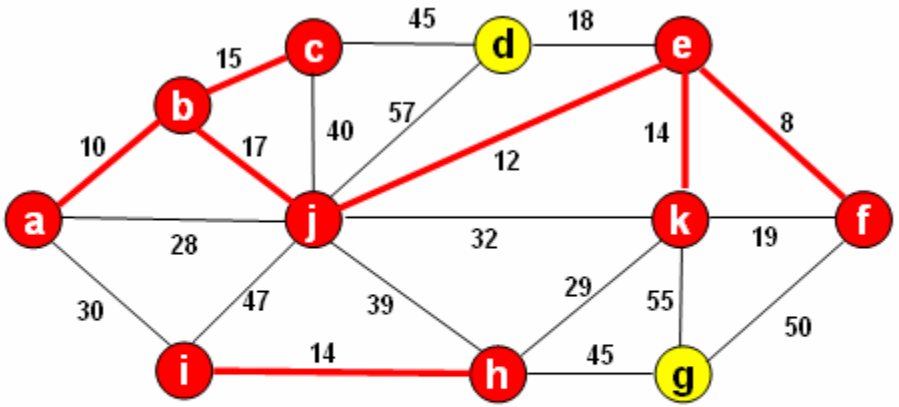
Kruskal's Algorithm Example

(7) The edge (*b,c,15*)is extracted from the priority queue. It does not form a cycle with any the tree edges. The edge is added to the spanning tree. The total weight of edges is *73*. The tree set is ***T={(e,f,8), (a,b,10), (e,j,12), (e,k,14), (h,i,14), (b,c,15)}***



					15	17	18	19	28	29	30	32	39	40	45	45	47	50	55	57
					b	b	d	f	a	h	a	j	h	c	c	g	i	f	g	d
					c	j	e	k	j	k	i	k	j	j	d	h	j	g	k	j

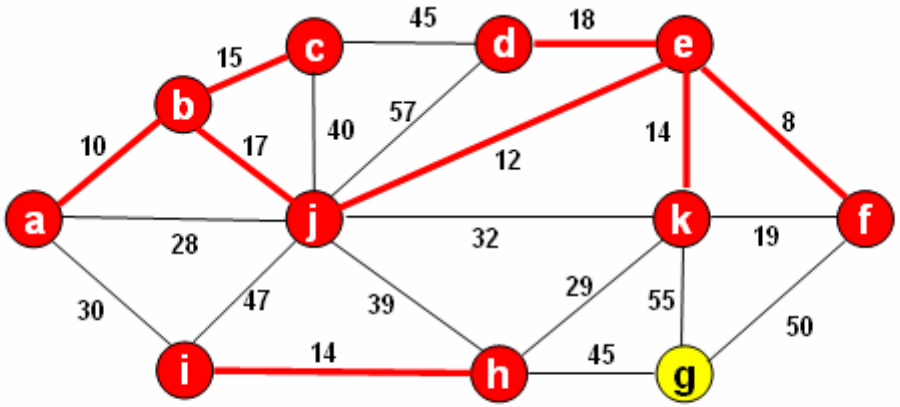
(8) The edge (*b, j ,17*) is extracted from the priority queue. It does not form a cycle with any of the existing tree edges. The edge is added to the spanning tree. The total weight of edges is *90*. The tree set is ***T={(e,f,8), (a,b,10), (e,j,12), (e,k,14), (h,i,14), (b,c,15), (b,j,17)}***



						17	18	19	28	29	30	32	39	40	45	45	47	50	55	57
						b	d	f	a	h	a	j	h	c	c	g	i	f	g	d
						j	e	k	j	k	i	k	j	j	d	h	j	g	k	j

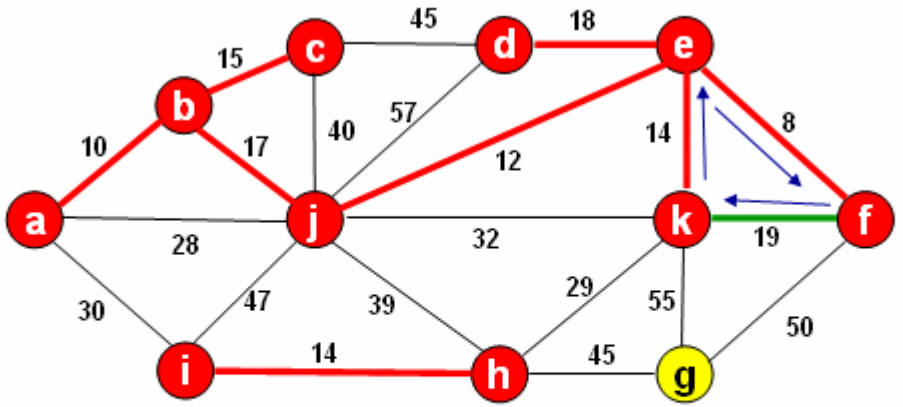
Kruskal's Algorithm Example

(9) The edge (*d,e,18.*) is extracted from the priority queue. It does not form cycle. It is added to the tree. The total weight of edges is *108*. The tree set is ***T={(e,f,8), (a,b,10), (e,j,12), (e,k,14), (h,i,14), (b,c,15), (b,j,17), (d,e,18)}***



							18	19	28	29	30	32	39	40	45	45	47	50	55	57
							d	f	a	h	a	j	h	c	c	g	i	f	g	d
							e	k	j	k	i	k	j	j	d	h	j	g	k	j

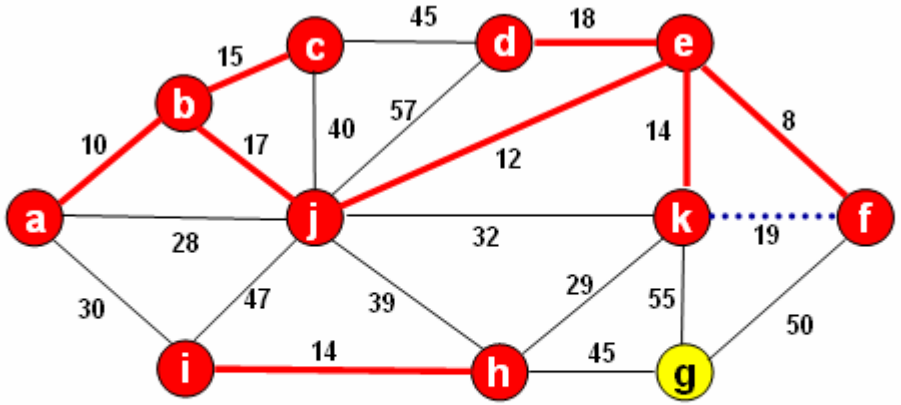
(10) The edge (*f,k,19*) is extracted from the priority queue. It forms a cycle with edges (*k,e,14*) and (*e, f,8*). The extracted edge is shown with bold green line in the figure. The cycle is identified by the arrows.



								19	28	29	30	32	39	40	45	45	47	50	55	57
								f	a	h	a	j	h	c	c	g	i	f	g	d
								k	j	k	i	k	j	j	d	h	j	g	k	j

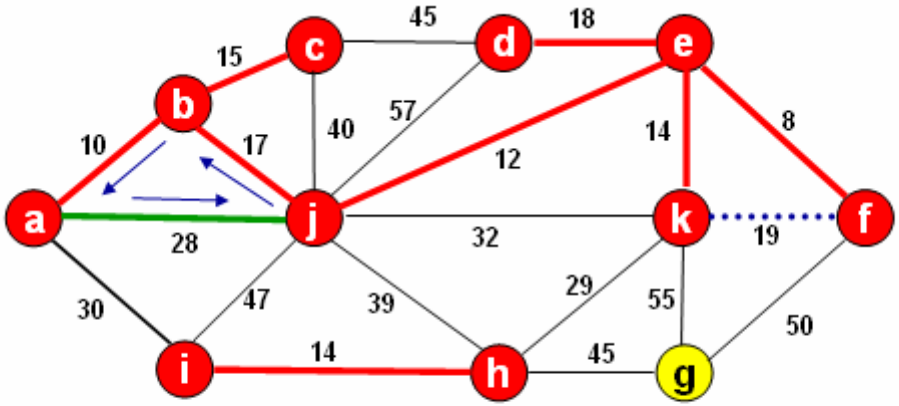
Kruskal's Algorithm Example

(11) The extracted edge is dropped. It is marked by broken line between the vertices *k* and *f*



								19	28	29	30	32	39	40	45	45	47	50	55	57
								f	a	h	a	j	h	c	c	g	i	f	g	d
								k	j	k	i	k	j	j	d	h	j	g	k	j

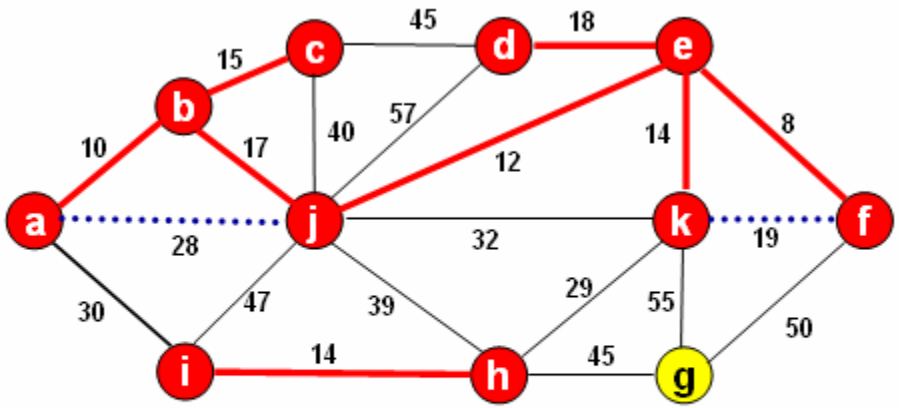
(12) The edge *(a,j,28)* is extracted from the priority queue. It forms cycle with the edges*(j,b,17)* and *(b,a,10)* which form part of the spanning tree.



									28	29	30	32	39	40	45	45	47	50	55	57
									a	h	a	j	h	c	c	g	i	f	g	d
									j	k	i	k	j	j	d	h	j	g	k	j

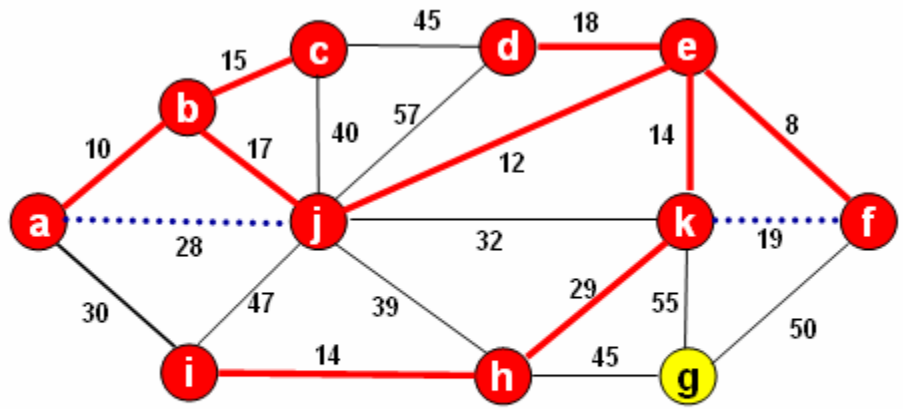
Kruskal's Algorithm Example

(13) The edge *(a,j,28)* is excluded from the tree. It is shown with broken line in the figure.



									28	29	30	32	39	40	45	45	47	50	55	57
									a	h	a	j	h	c	c	g	i	f	g	d
									j	k	i	k	j	j	d	h	j	g	k	j

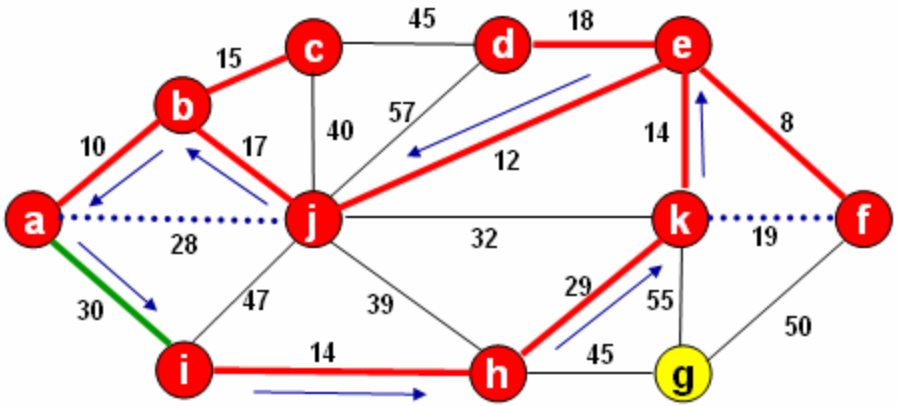
(14) The edge *(h,k,29)* is extracted from the priority queue. It does not form a cycle with any of the tree edges. It is, therefore, added to the tree. The total weight of edges is *137*. The tree set is ***T={(e,f,8), (a,b,10), (e,j,12), (e,k,14), (h,i,14), (b,c,15), (b,j,17), (d,e,18), (h,k,29)}***



										29	30	32	39	40	45	45	47	50	55	57
										h	a	j	h	c	c	g	i	f	g	d
										k	i	k	j	j	d	h	j	g	k	j

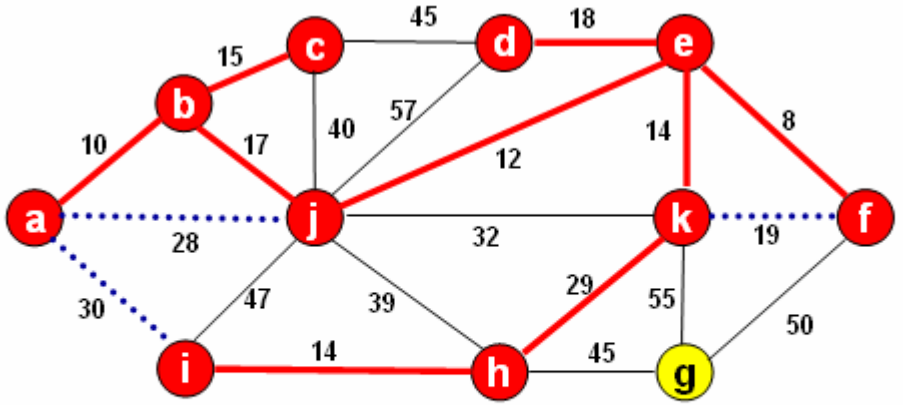
Kruskal's Algorithm Example

(15)The edge *(a,i,30)* is extracted from the queue. It forms a cycle with the tree edges *(i,h,14), (h,k,29), (k,e,14), (e,j,12), (j,b,17), (b,a,10)*. The extracted edge is shown with bold green line in the figure.



											30	32	39	40	45	45	47	50	55	57
											a	j	h	c	c	g	i	f	g	d
											i	k	j	j	d	h	j	g	k	j

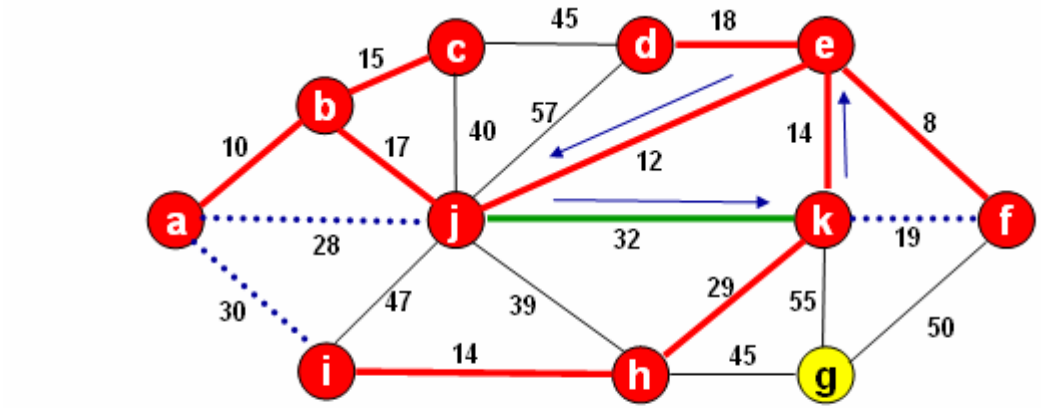
(16) The edge (*a,i,30)* is excluded from the tree. It is marked with a broken line in the figure



											30	32	39	40	45	45	47	50	55	57
											a	j	h	c	c	g	i	f	g	d
											i	k	j	j	d	h	j	g	k	j

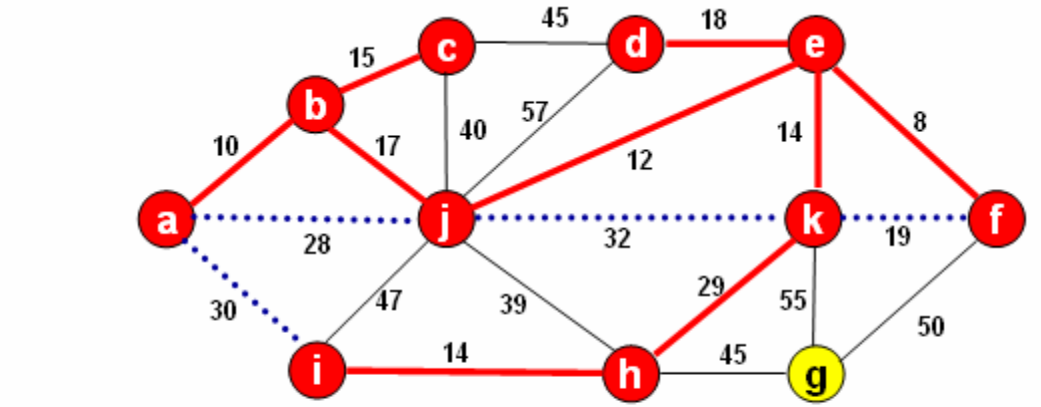
Kruskal's Algorithm Example

(17) The edge *(j,k,32)* is extracted. It forms cycle with tree edges*(k,e,14)* and *(e,j,12)*. The edge is depicted with a bold green line in the figure.



												32	39	40	45	45	47	50	55	57
												j	h	c	c	g	i	f	g	d
												k	j	j	d	h	j	g	k	j

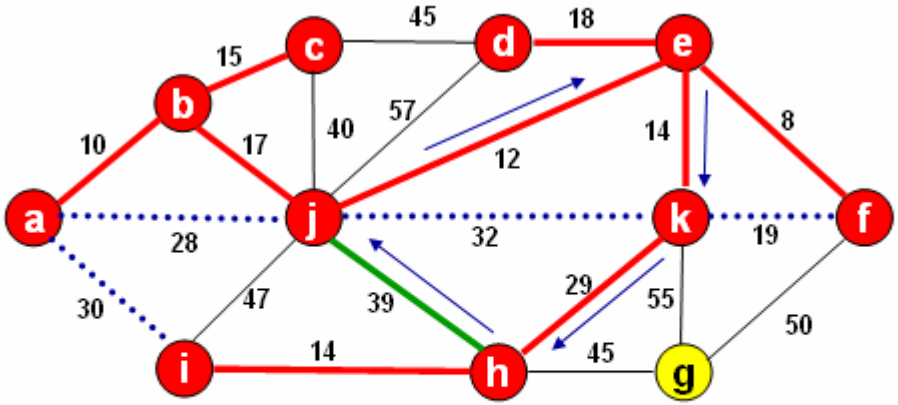
(18) The edge *(j,k,32)* is excluded from the spanning tree. It is shown with broken line in the figure



												32	39	40	45	45	47	50	55	57
												j	h	c	c	g	i	f	g	d
												k	j	j	d	h	j	g	k	j

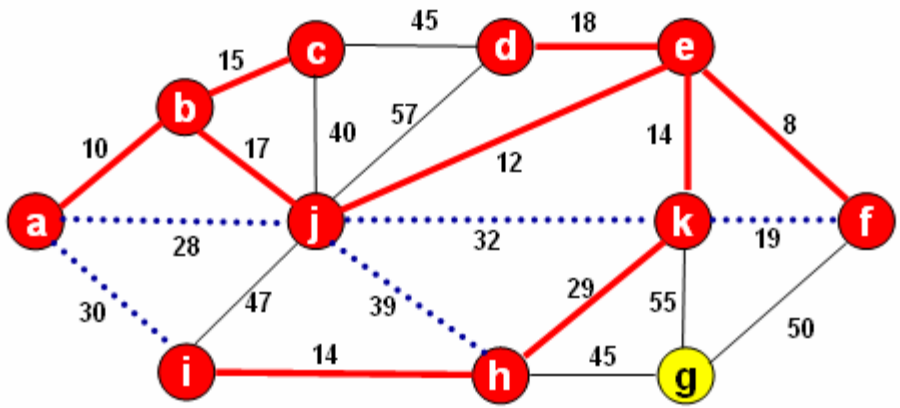
Kruskal's Algorithm Example

(19) The edge *(h,j,39)* is extracted from the queue. It forms a cycle with the tree edges *(j,e,12), (e,k,14), (k,h,29)*. The cycle is identified by arrows in the diagram



												39	40	45	45	47	50	55	57
												h	c	c	g	i	f	g	d
												j	j	d	h	j	g	k	j

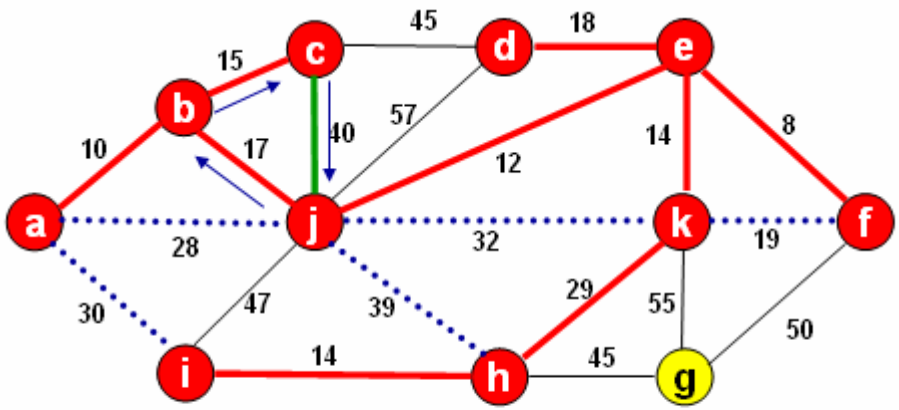
(20) The edge *(h, j, 39)* is excluded from the spanning tree configuration. It is depicted by broken line in the diagram



												39	40	45	45	47	50	55	57
												h	c	c	g	i	f	g	d
												j	j	d	h	j	g	k	j

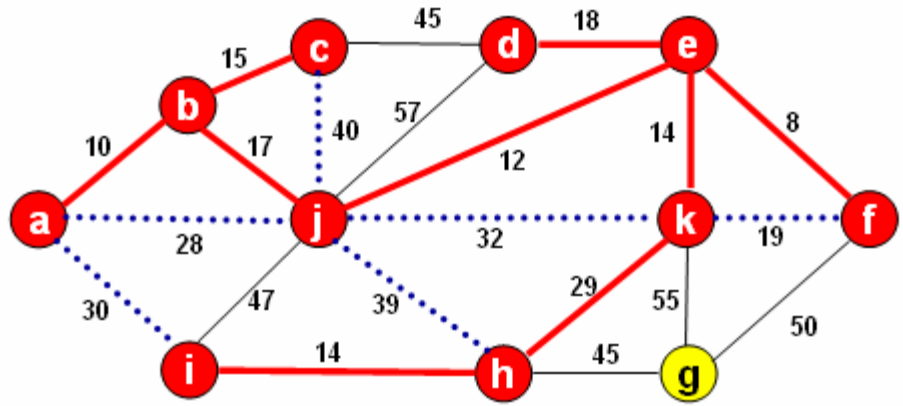
Kruskal's Algorithm Example

(21) The *(c,j,40)* is extracted from the queue. It forms cycle with the tree edges *(j,b,17)* and *(b,c,15)*. The cycle is identified by arrows in the figure. The extracted edge is shown with bold green line



													40	45	45	47	50	55	57
													c	c	g	i	f	g	d
													j	d	h	j	g	k	j

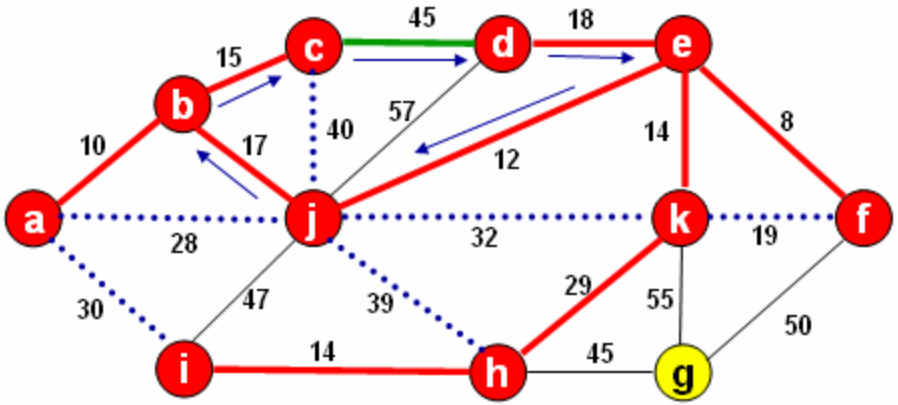
(22)The edge *(c,j,40)* is excluded from the existing tree configuration. It is shown with broken line in the figure



													40	45	45	47	50	55	57
													c	c	g	i	f	g	d
													j	d	h	j	g	k	j

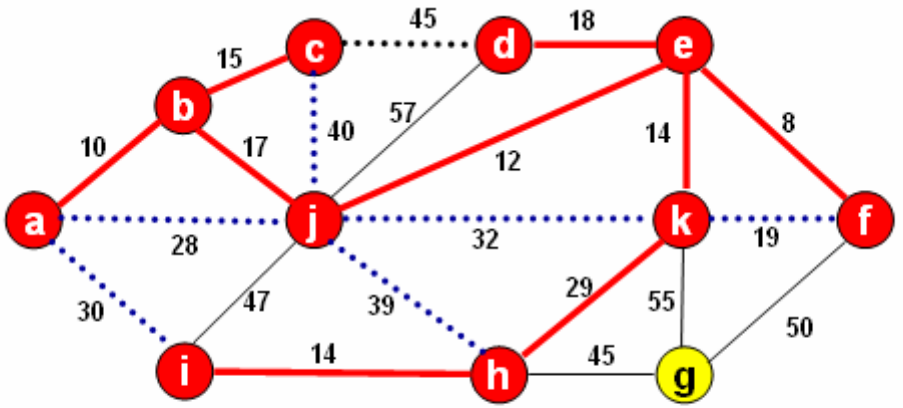
Kruskal's Algorithm Example

(23) The edge *(c,d,45)* is extracted from the queue. It forms a cycle with the tree edges *(d,e,18), ((e,j,12), (j,b,17), (b,c,15)*. The cycle is identified by arrows. The extracted edge is shown with bold green line in the diagram.



														45	45	47	50	55	57
														c	g	i	f	g	d
														d	h	j	g	k	j

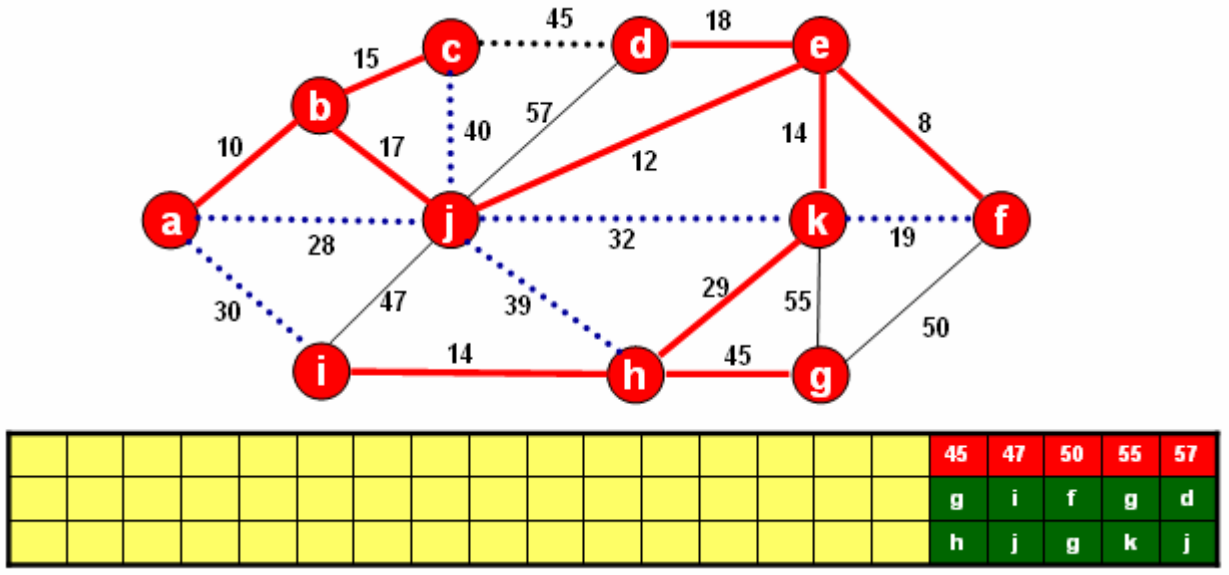
(24) The edge *(c,d,45)* is discarded. It is depicted by broken line in the figure



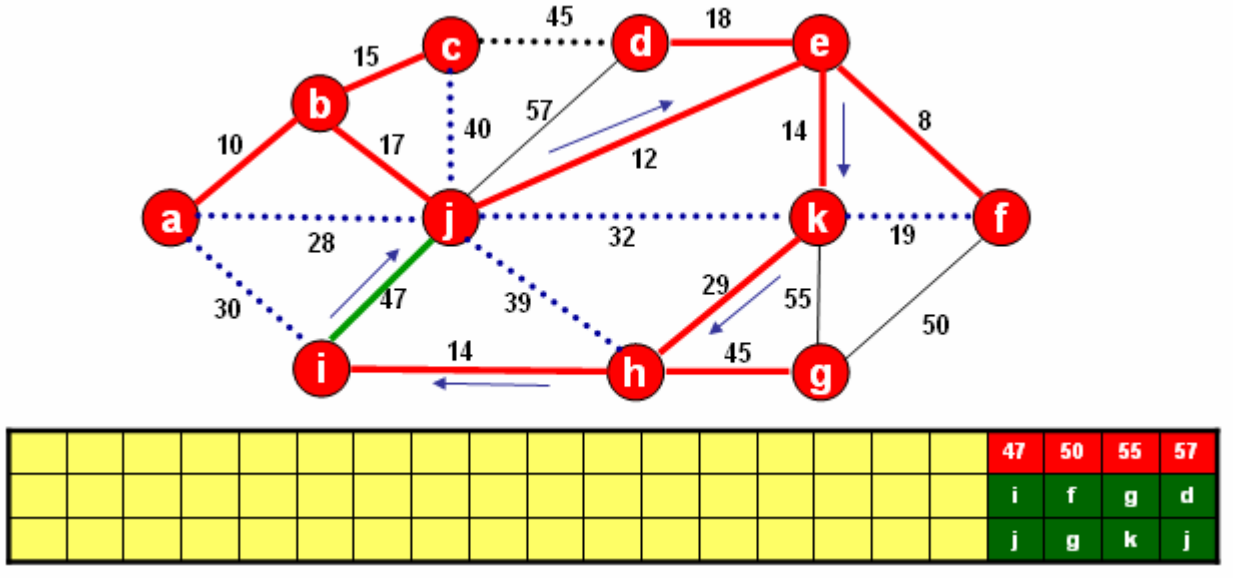
														45	45	47	50	55	57
														c	g	i	f	g	d
														d	h	j	g	k	j

Kruskal's Algorithm Example

(25)The edge *(g,h,45)* is extracted from the queue. It does not form a cycle with any of the tree edges. The edge is added to the tree. The total weight of edges is *182*.The tree set is ***T={(e,f,8), (a,b,10), (e,j,12), (e,k,14), (h,i,14), (b,c,15), (b,j,17), (d,e,18), (h,k,29), (g,h,45)}***

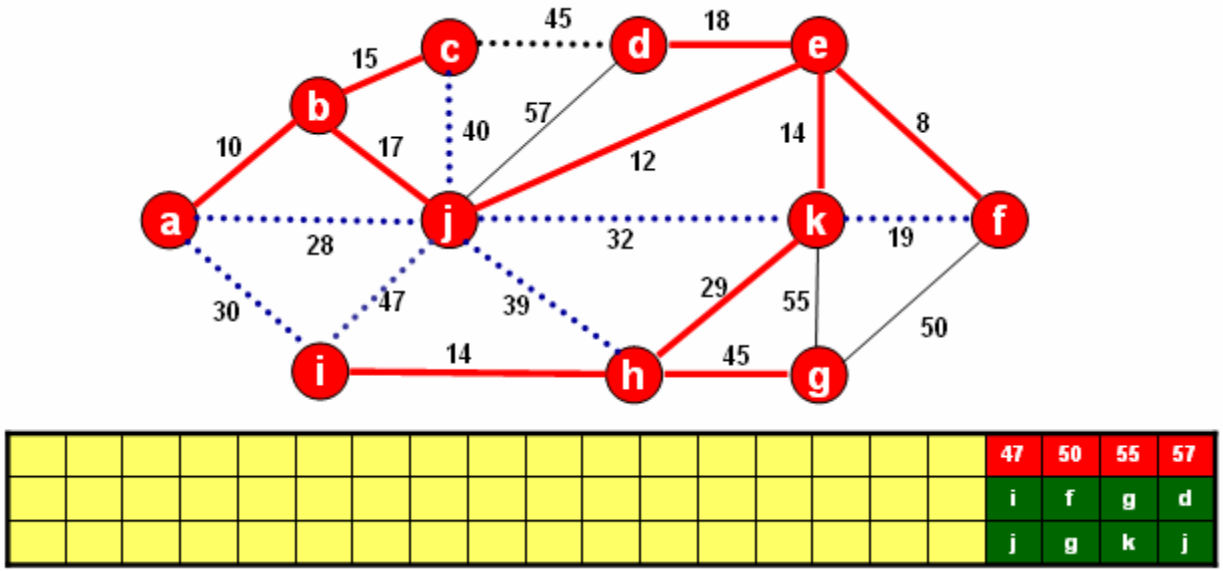


(26)The edge*(i,j,47)* is extracted from the queue. It forms a cycle with tree edges *(j,e,12), (e,k,14), (k,h,29), (h,i,14)*. The cycle is marked by the arrows. The extracted edge is shown with bold green line in the figure

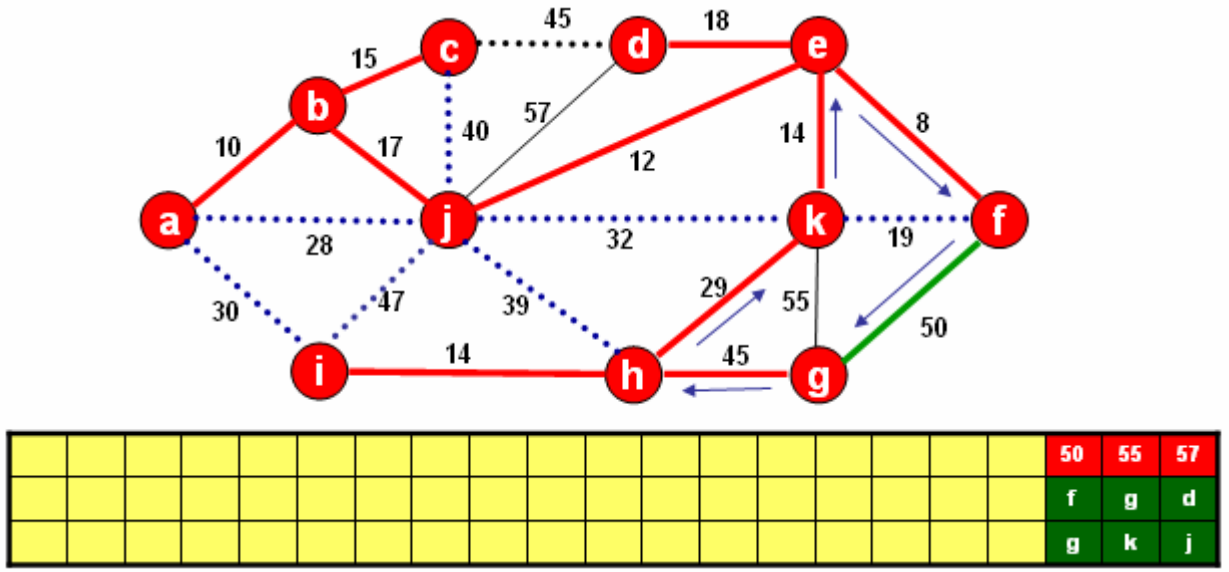


Kruskal's Algorithm Example

(27) The edge *(i,j,47)* is dropped. It is marked by a dotted line in the figure.

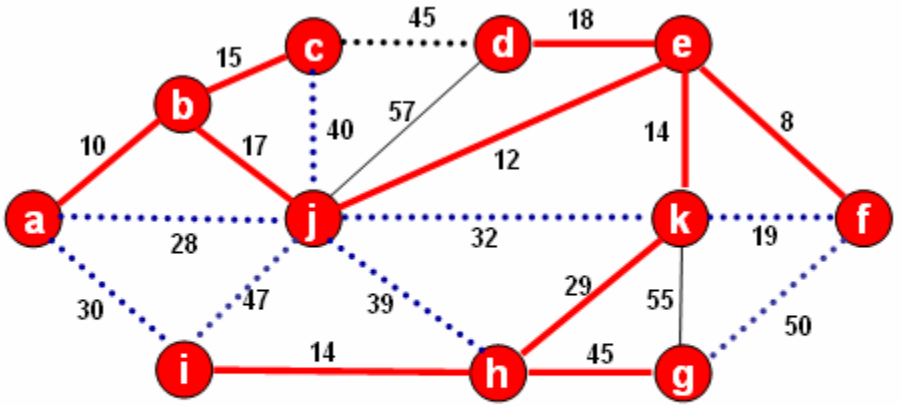


(28) The edge *(f,g,50)* is extracted from the queue. It forms a cycle with the tree edges *(g,h,45), (h,k,29), (k,e,14), (e,f,8)*. The cycle is identified by arrows. The extracted edge is shown by bold green line in the figure



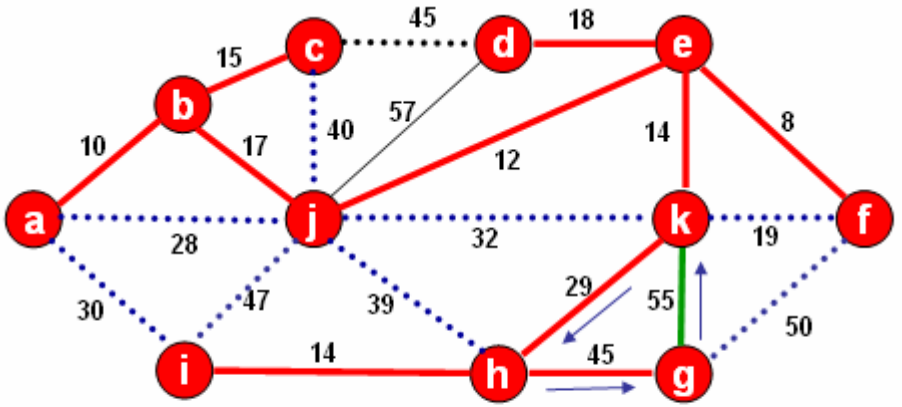
Kruskal's Algorithm Example

(29) The edge *(f,g,50)* is excluded from the tree configuration. It is marked by dotted line in the figure



																				50	55	57
																				f	g	d
																				g	k	j

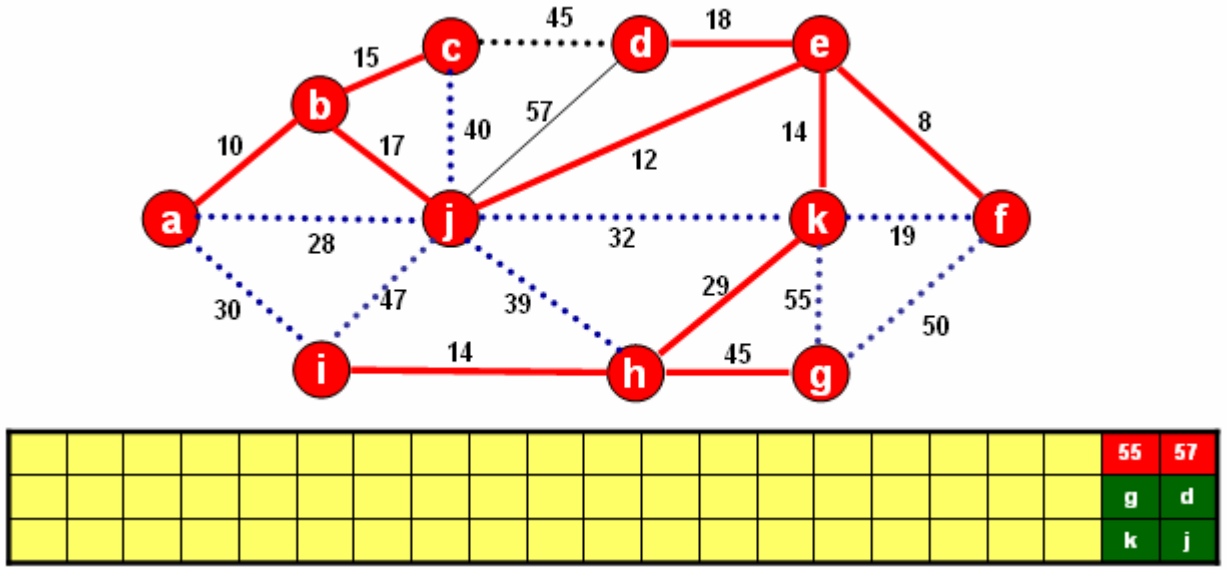
(30)The edge *(g,k,55)* is extracted from the queue. It forms cycle with the tree edges *(k,h,29)* and *(h,g,45)*. The cycle is marked by the arrows. The extracted edge is shown with bold green line



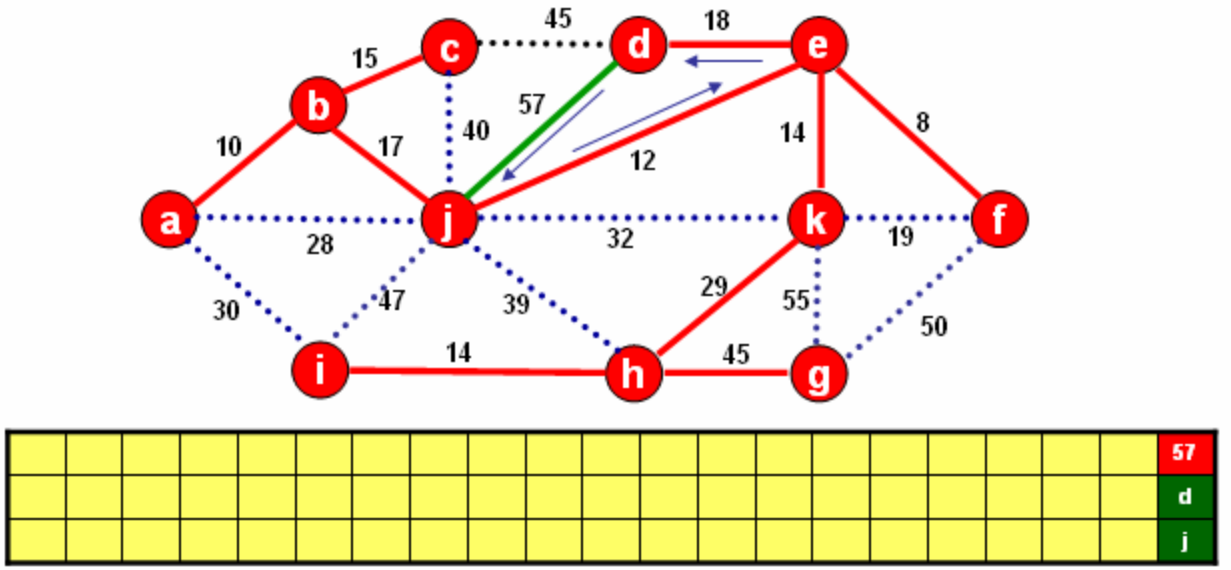
																					55	57
																					g	d
																					k	j

Kruskal's Algorithm Example

(31) The edge *(g,k,55)* is dropped. It is marked by dotted line in the figure

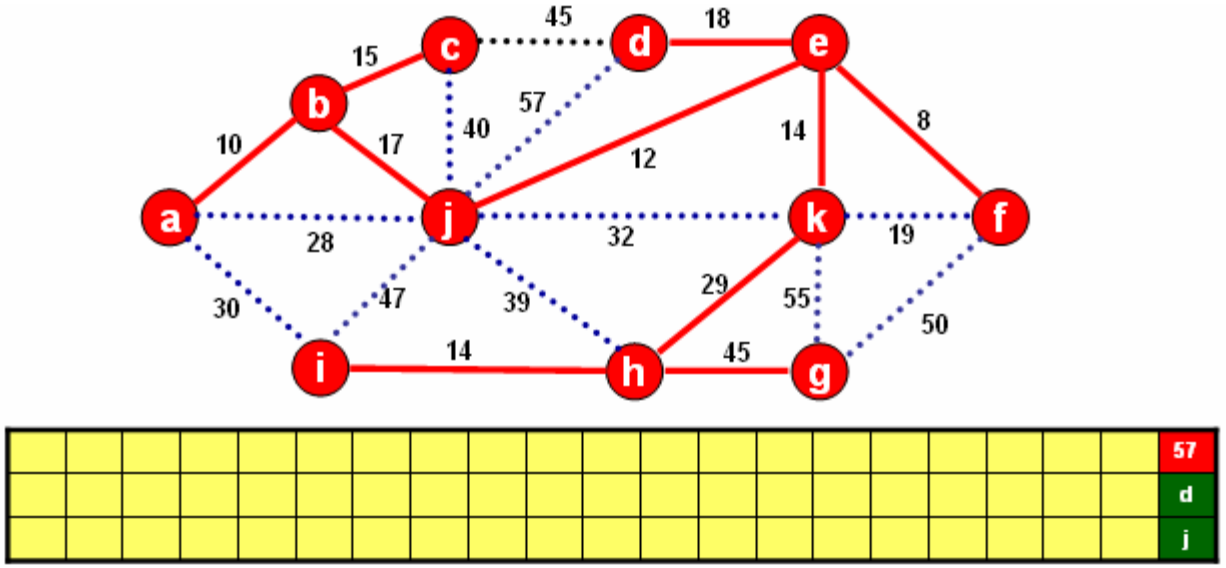


(32) The edge *(d,j,57)* is extracted from the queue. It forms a cycle with tree edges *(j,e,12)* and *(e,d,18)*. The cycle is marked by arrows. The extracted edge is shown with bold green line in the figure

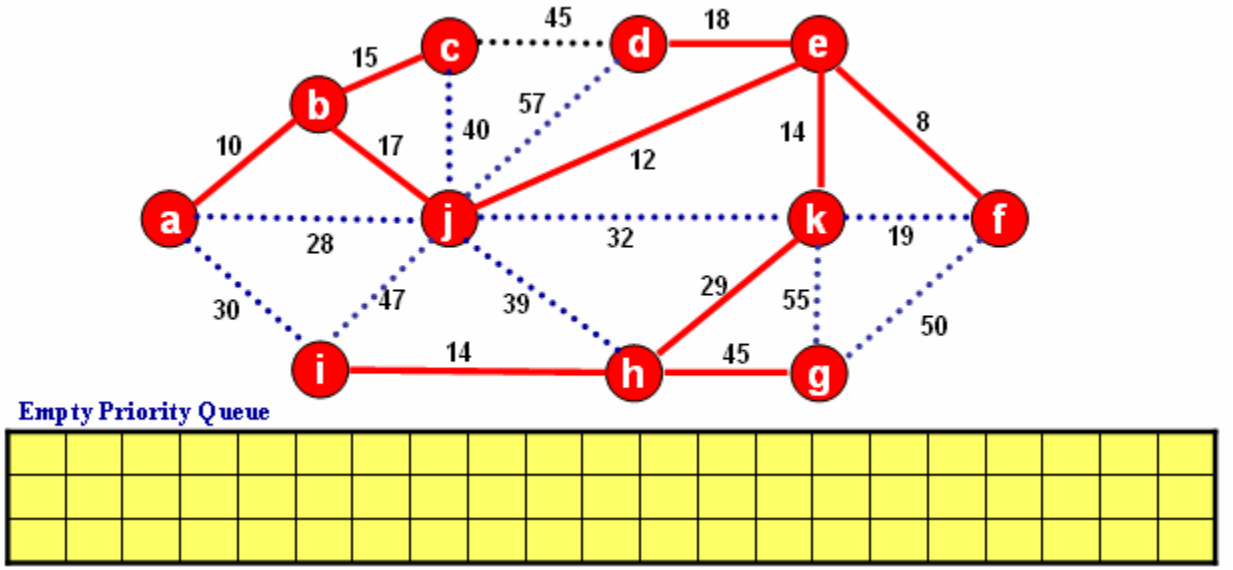


Kruskal's Algorithm Example

(33) The edge*(d,j,57)* is excluded from the tree. It is shown with broken line in the figure.



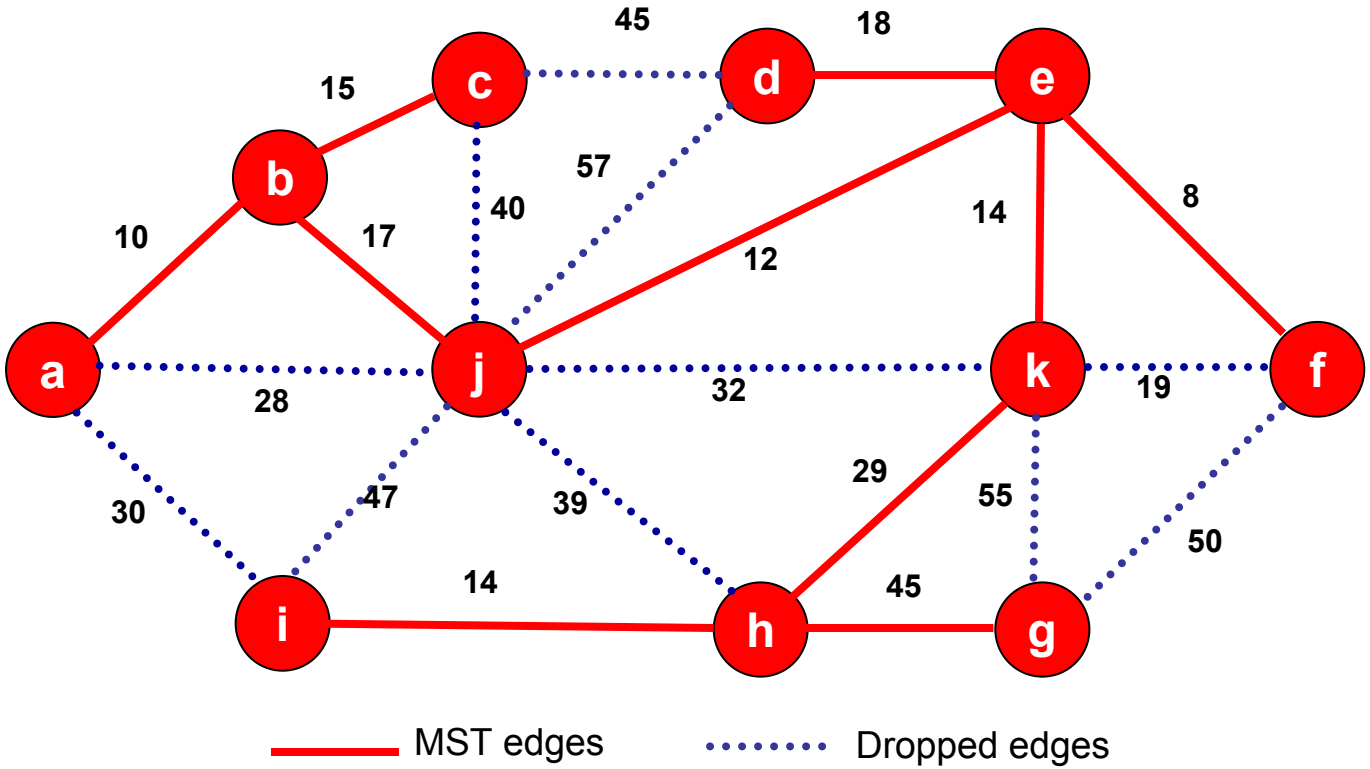
(34) After removing the edge*(d,j,57),* the priority queue becomes empty. The Kruskal's algorithm terminates.



Kruskal's Algorithm

Example

The ***Minimum Spanning Tree (MST),*** generated by the Kruskal's algorithm, is shown below. The dotted lines indicate the edges that are in the sample graph but excluded from the MST. The tree includes all of the vertices in the original graph. It does not have any cycles. The total weight of all the tree edges is ***182.***



Kruskal's Algorithm

Implementation

It follows that the Kruskal's algorithm is implemented in two phases

(1) *Sorting edges in non-decreasing order*

(2) *Constructing spanning tree (a sub-graph which has no cycles)*

- In actual implementation, the edges can be sorted by using an efficient algorithm such as *Quick sort*. Alternatively, a *Priority Queue* can be used. The edges are retrieved in non-decreasing order by *dequeue* operation, as shown in the preceding example.

- For *detection of cycles Union Sets are used*

Kruskal's Algorithm

Detecting Cycles

To check whether or not an extracted edge forms a cycle, a *Union Set* can be used. A Union Set is defined as a set of *disjoint sets of vertices in a graph*

Example : Consider the set of vertices *V={a ,b, c, d ,e}* in a sample graph. The Union Set *UV* is defined as the set consisting of members *{a}, {b}, {c}, {d}*. In set notation *UV* can be denoted as *set of sets*, as under

UV={ {a}, {b}, {c}, {c}, {d}, {e} }

- The rule for accepting or discarding an extracted edge is as follows. Suppose that an edge *(u, v)* is extracted from a priority queue. Let *UV* be the Union Set of vertices

  (i) *If the vertices u, v belong to the **same member** S of the Union Set UV then the edge (u, v) forms **cycle**, and **discarded***

  (i) *If the vertices u, v belong to two different members, S1 and S2, of the Union Set UV then the edge (u, v) is **accepted**. Further, the sets S1 and S2 are **merged together to form a single set S** which is placed in the Union Set UV*

Detecting Cycles

Example

Suppose at any stage of the execution of Krukal's algorithm , the ***UV Union Set of vertices*** consists of members *{a}, {b}, { c, e}, {d} ,i.e UV= { {a}, {b}, {c, e}, {d} }* . Consider the following possibilities:

(1) Suppose an edge (*e, d*) is extracted from the priority queue. Since vertices *e, d* belong to two *different* members *{ c, e}* and *{ d}* of *UV*, the edge *(e, d)* does not form a cycle. The sets *{e, d}* and *{ d }* will be merged into the set *{ c, d, e}* which will replace the sets *{ c,e} { d}* in the original set *UV*. Thus, the updated *UV* would be *UV= { {a}, {c, d ,e} }*

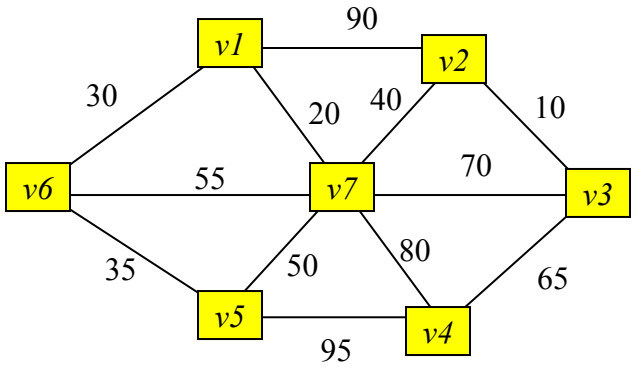
(2) Assume again that an edge *(c, e)* is extracted. Since the vertices *e* and *d* belong to the *same* set *{c, d, e}* of the *UV*, the edge *(c, e)* will be discarded

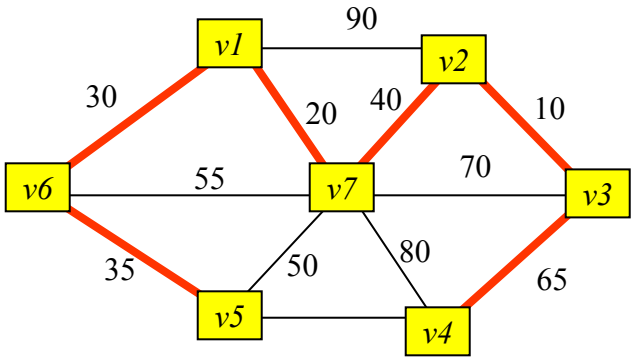
(3) Again, assume that the edge (*a, b)* is extracted. Since the vertices *a, b* belong to *different* sets, the corresponding disjoints set, namely, *{a}, {b}* will be merged. Thus, the updated *UV* would be *UV={{a, b}, {c, d, e} }*.

(4) Next , assume that edge *(b ,c)* is extracted, Since *b* belongs to *{a,b}* and *c* belongs to *{c, d, e}* the sets *{a, b}* and *{c, d, e}* would be merged i.e *UV={a, b, c, d, e}*

Kruskal's Algorithm

Using Set for MST





Edge	Weight
(v2,v3)	10
(v1,v7)	20
(v1,v6)	30
(v5,v6)	35
(v2,v7)	40
(v5,v7)	50
(v6,v7)	55
(v3,v4)	65
(v3,v7)	70
(v4,v7)	80
(v1,v2)	90
(v4,v5)	95

Edge	Action	UV={ {v1}, {v2}, {v3}, {v4}, {v5}, {v6}, {v7} }
(v2,v3)	added	{ {v1}, **{v2, v3}**, {v4}, {v5}, {v6}, {v7} }
(v1,v7)	added	{ **{v1,v7}**, {v2,v3},{v4}, {v5},{v6}}
(v1,v6)	added	{ **{v1,v6,v7}**, {v2,v3},{v4},{v5} }
(v5,v6)	added	{ **{v1, v5,v6,v7}**, {v2,v3},{v4} }
(v2,v7)	added	{ **{v1,v2,v3,v5,v6,v7}**,{v4} }
(v5,v7)	rejected	V5,v7 already in the first set
(v6,v7)	rejected	V6,v7 already in the first set
(v3,v4)	added	{ **{v1,v2,v3,v4,v5,v6,v7}** }
(v3,v7)	rejected	V3,v7 already included
(v4,v7)	rejected	V4,v7 already included
(v1,v2)	rejected	V1,v2 already included
(v4,v5)	rejected	V4,v5 already included

Kruskal's Algorithm

Implementation

Following is pseudo code for ***Kruskal's algorithm***.. The Union Set *UV* initially holds vertices of the graph. The input to the procedure includes a matrix W of edge weights. A priority queue is created to hold the edges and weights.

```
MST-KRUSKAL(G,W)
1  T←φ                                  ►Initialize Minimum Spanning Tree
2  UV←φ                                 ► Initialize Union set UV
3  for each vertex v ∈ V
4       do UV ← UV  ∪   {v}             ► Add Vertex sets to union set UV
5  for each (u, v)  ∈ E                 ►Create a priority queue of edges and weights
6       do   ENQUEUE(Q,u,v,w)           ► weights are  keys (priorities)
7  while |UV| >1 do                     ► Continue until all vertices form a single set of cardinality 1
8       (u, v) ←DEQUEUE(Q)              ►Remove an edge (u,v) from the queue
9        if u, v belong to different sets US1,US2 ∈ UV
10            then  UV ←UV – US1        ► remove set US1 from UV
11                  UV←UV  - US2        ►remove set US2 from UV
12                  UV ←UV ⋃( US1 ⋃US2) ►Join sets US1 and US2 , add the union to UV
13                  T←T ⋃(u, v)         ►Add the selected edge to the spanning tree
14 return T
```

Kruskal Visualization

Kruskal's Algorithm Visualization

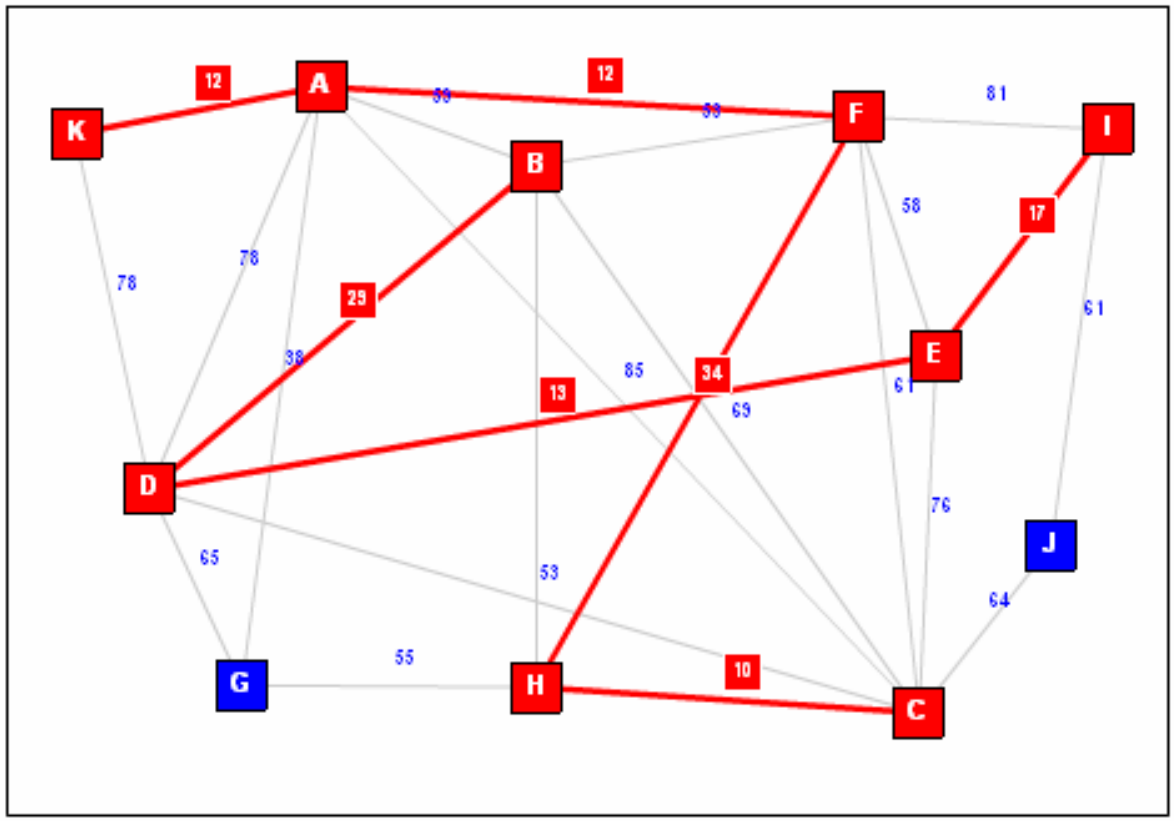
Reset the graph to the initial state

Visualization

- Slow
- Medium
- Fast

Start | Reset Graph | Draw Graph | stop

DEQUEUING: Edge(F,H)=34 added to MST



Priority Queue

Wt	Edge		Wt	Edge		Wt	Edge	
35	B	H						
38	A	G						
53	B	F						
53	C	D						
55	G	H						
58	E	F						
59	A	B						
61	C	F						
61	I	J						
64	C	J						
65	D	G						
69	B	C						
76	C	E						
78	D	K						
78	A	D						
81	F	I						
85	A	C						

Analysis of Kruskal's Algorithm

Running Time

Assume that edges are processed by using ***priority queue,*** graph is represented by linked lists, and Union Set is used to detect cycles. The main contributions to the running time are

(1) T_{sort} *, time to sort edges*

(2) T_{ini} *, time to initialize the sets*

(3) T_{scan} *, time to scan linked lists*

It was shown earlier that the time to sort n data items, using quick sort is $n \lg n$. Since edge set contains $|E|$ edges, time to sort edges will be

$$T_{sort} = O(|E|. \lg| E|).$$

The time for initialization of sets will be

$$T_{ini} = O(|V|).$$, where $|V|$ is number of vertices in the graph

The total time to scan the linked lists representing the graph is equal to number of edges in the graph. This time is at most

$$T_{scan} = O(|E|)$$

Thus, total running time is $\boldsymbol{O(|E|. \lg| E|) + O(|E|)+O(|V|)}$. Ignoring $|E|$ compared to $|E|.|\lg |E|$, the running time for Kruskal's algorithm is

$$\boldsymbol{T_{kruskal} = O(|E|.\lg |E|) +O(|V|)}$$

Prim's Algorithm

Prim's Algorithm

Strategy

The ***Prim's algorithm*** is an alternative method for creating minimum spanning tree for a ***weighted undirected graph***. Unlike the Kruskal's algorithm, it makes a systematic selection of vertices. It proceeds by choosing some arbitrary initial vertex, and then examines all neighbors. It selects the neighbor which has the shortest distance. Next, the neighbors of selected vertices are examined for shortest link. This process is continued till all the vertices have been explored. The selected shortest links form ***minimum spanning tree.***

Prim's Algorithm

Procedure

Let *V* be the vertex set for a graph *G*. Let *T* be the ***minimum spanning tree*** for *G*. The Prim's algorithm proceeds as follows:

Step #1: *Select some vertex s in V , as the start vertex.*

Step # 2: *Add vertex* ***s*** *to an empty set S. Remove* ***s*** *from V.*

Step # 3: *Repeat Step #:4 through Step #:6 until the set V is empty.*

Step # 4: *Examine all vertices in S which are linked to vertices in V.*

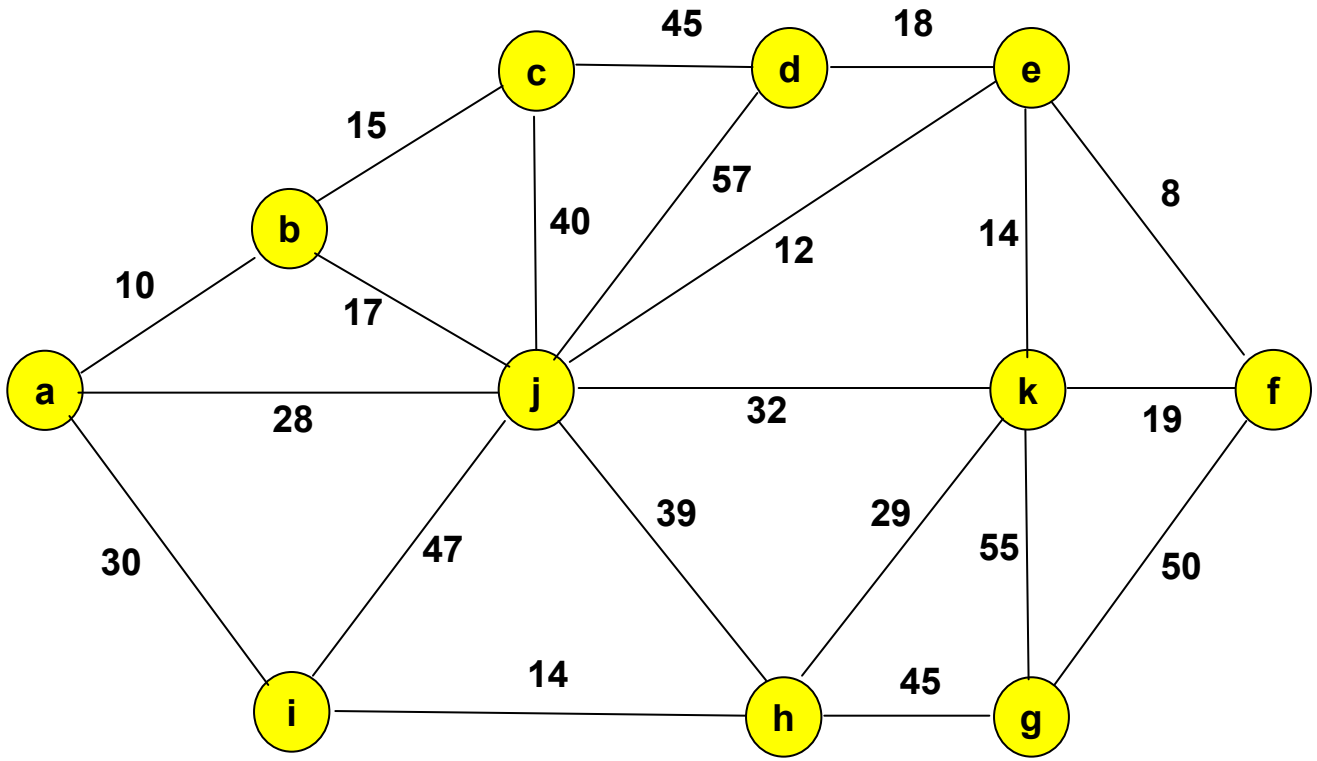
Step #5 :*Choose the vertex* ***u*** *in V which has the* ***minimum distance from vertex v*** *in S.*

Step #6: *Remove vertex* ***u*** *from V and add it to S. Move edge* ***(v ,u)*** *to T.*

Prim's Algorithm

Example

In order to study the working of Prim's algorithm, consider the ***weighted undirected graph shown*** in the figure below

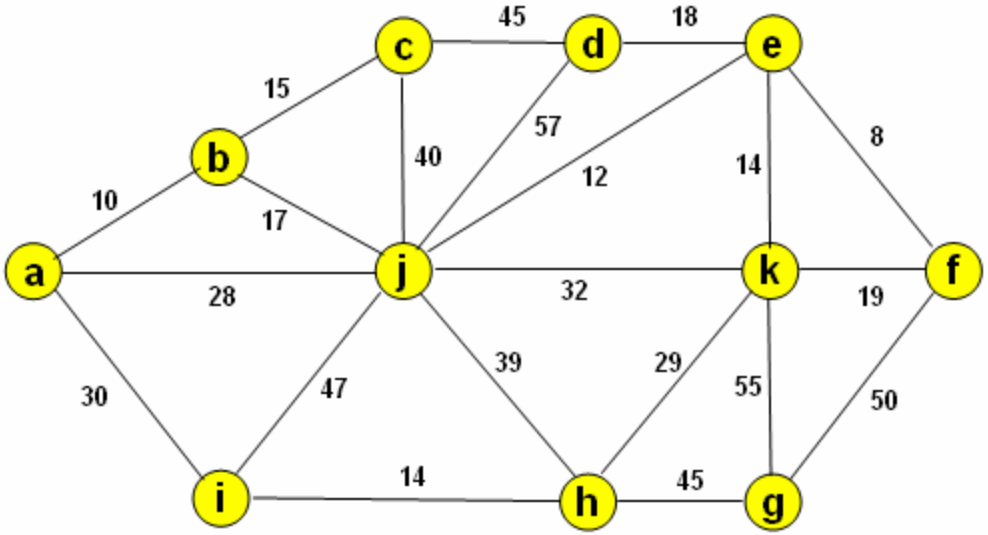


The working of algorithm is illustrated by the next set of figures with exploratory notes

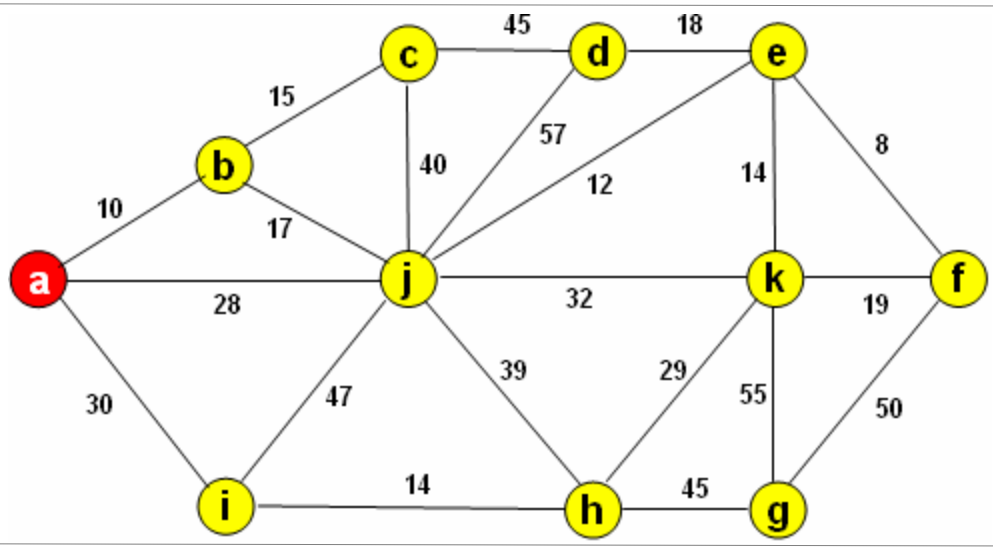
Prim's Algorithm

Example

(1) Figure shows a sample *weighted undirected graph*. The vertex set for the graph is *V={a, b, c, d, e, f, g, h, i, j, k}*. A set *S* is used to store vertices, which are selected to form minimum spanning tree. Another set *T* is defined to contain the edges of minimum spanning tree. Initially, *S* and *T* are empty: *S={}, T={}*



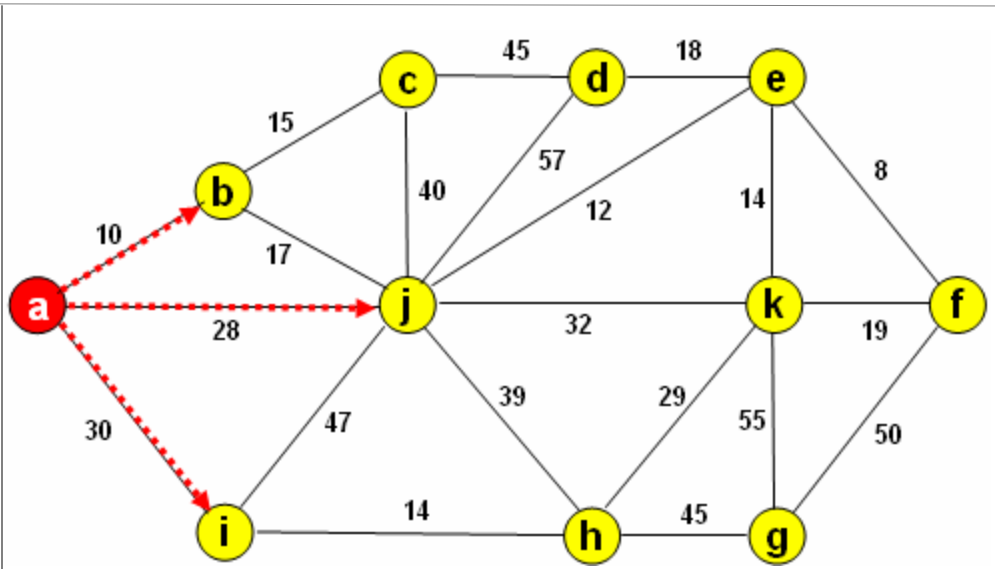
(2) First, vertex *a* is selected for the start of Prim's algorithm. In the figure it is shaded red. The selected vertex is removed from set *V* and placed in set *S*. Thus,
V={ b, c, d, e, f, g, h, i, j, k}
S={a}



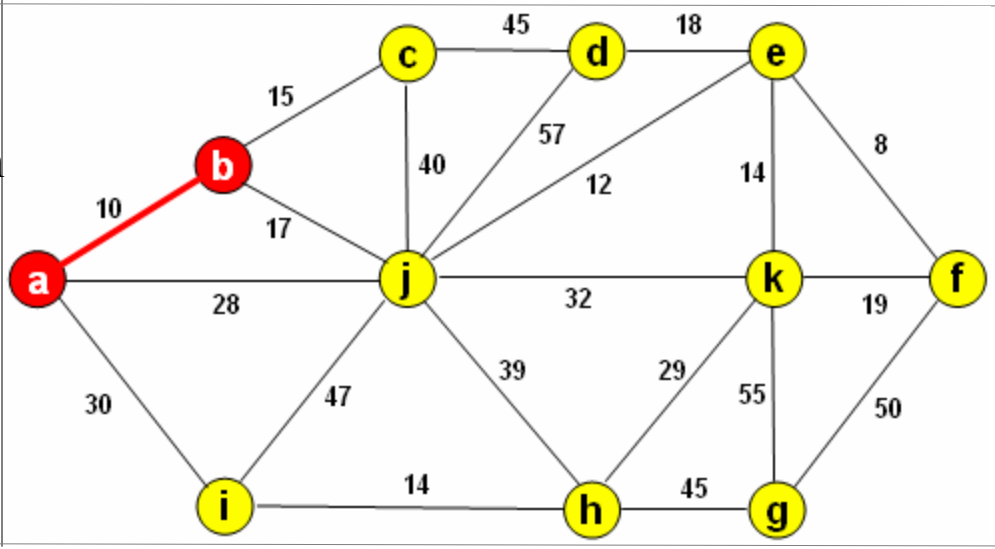
Prim's Algorithm

Example

(3) Initially, *V={ b, c, d, e, f, g, h, i, j, k}*
S={a}
The distances of vertices in set *S* to the *linked* vertices in set *V* are as follows:
a→b=10 (minimum)
a→j=28
a→i=30
The minimum distance 10 is from *a* to *b*



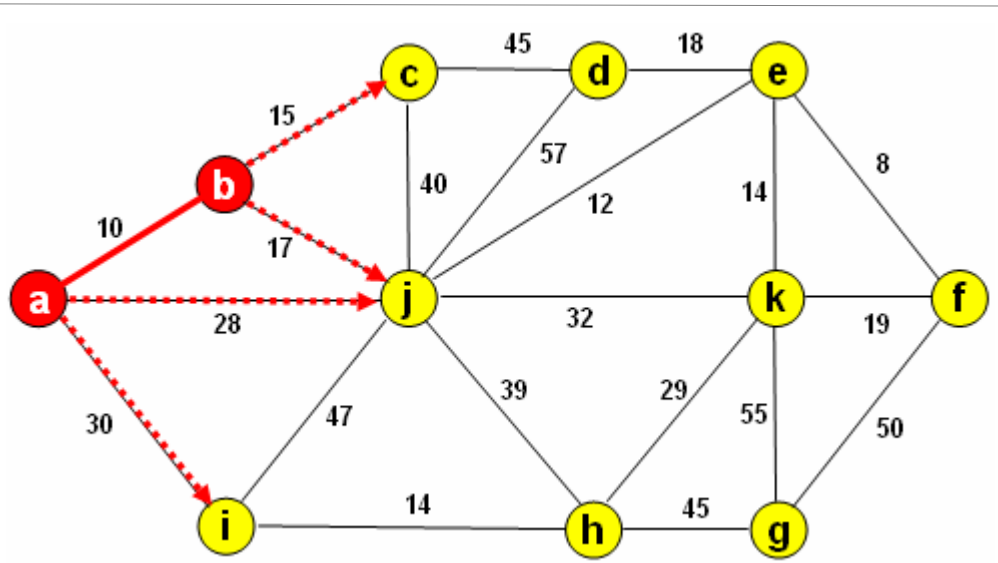
(4) Vertex *b,* which has the *minimum distance,* is selected and placed in set *S*.
Thus, The *edge (a,b,10)* of MST is placed in set *T :*
V={c, d, e, f, g, h, i, j, k}
S={a, b},
T= {(a,b,10)}
The total distance so far is*10*



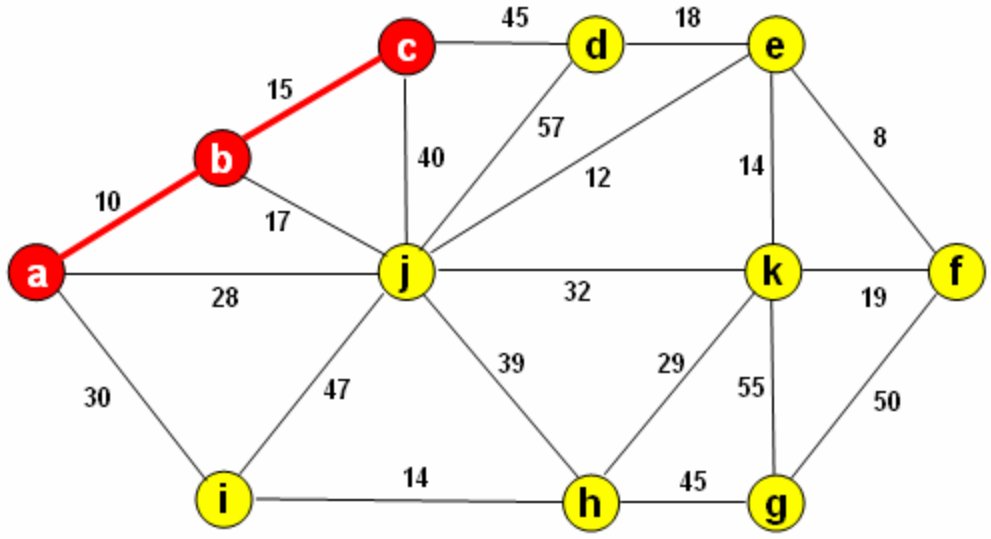
Prim's Algorithm

Example

(5) Initially, *V={ c, d, e, f, g, h, i, j, k}*
S={a, b}
The distances of vertices in set *S* to the *linked* vertices in set *V* are as follows:
a→j=28
a→i=30
b→j=17
b→c=15 (minimum)
The minimum distance *15* is from *b* to *c*



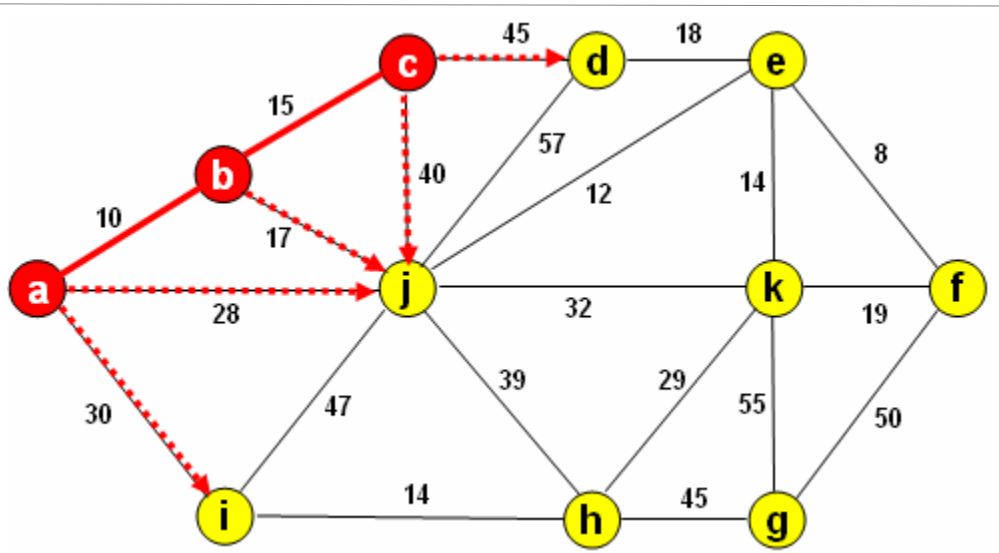
(6) Vertex *c,* which has the *minimum distance,* is selected and placed in set *S*.
Thus, The *edge (b,c,15)* of MST is placed in set *T :*
V={d, e, f, g, h, i, j, k}
S={a, b, c},
T={(a,b,10), (b,c,15)}
The total distance so far is *25*



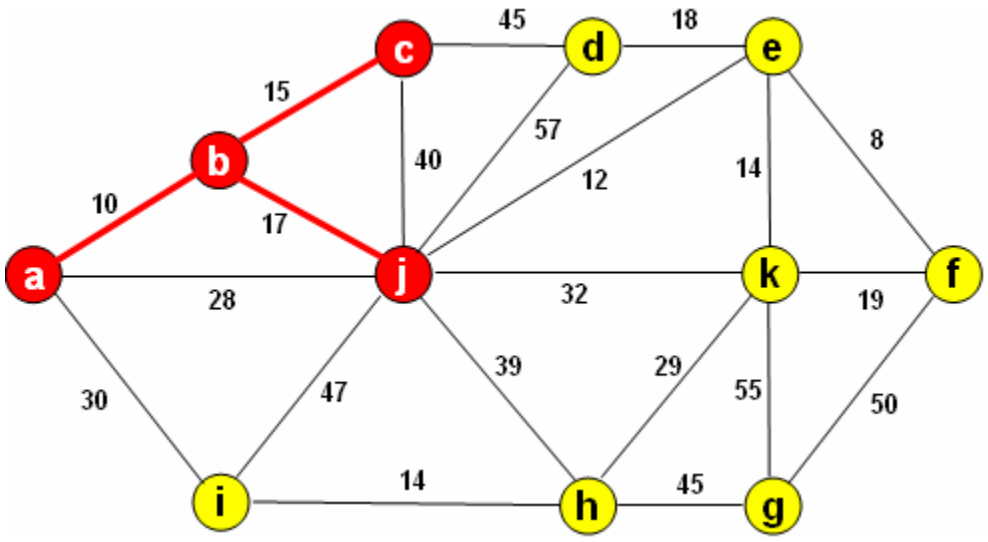
Prim's Algorithm

Example

(7) Initially, *V={ d, e, f, g, h, i, j, k}*
S={a, b, c}
The distances of vertices in set *S* to the *linked* vertices in set *V* are as follows:
a→j=28, a→i=30
b→j=17 (minimum)
c→d= 45, c→j=40
The minimum distance 17 is from *b* to *j*



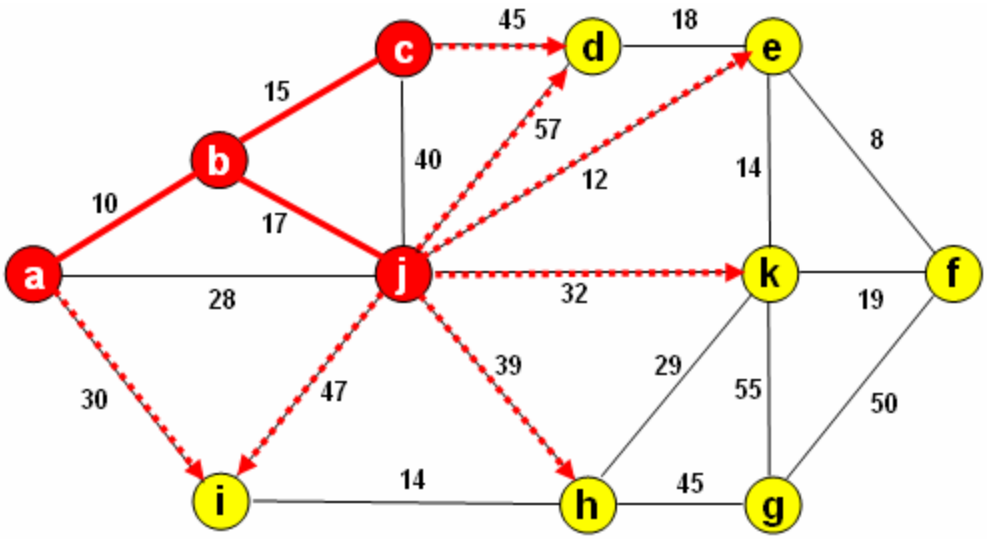
(8) Vertex *j,* which has the *minimum distance,* is selected and placed in set *S*.
Thus, The *edge (b,j,17)* of MST is placed in set *T :*
V={d, e, f, g, h, i, k}
S={a, b, c, j}
T={(a,b,10), (b,c,15), (b,j,17)}
The total distance so far is *42*



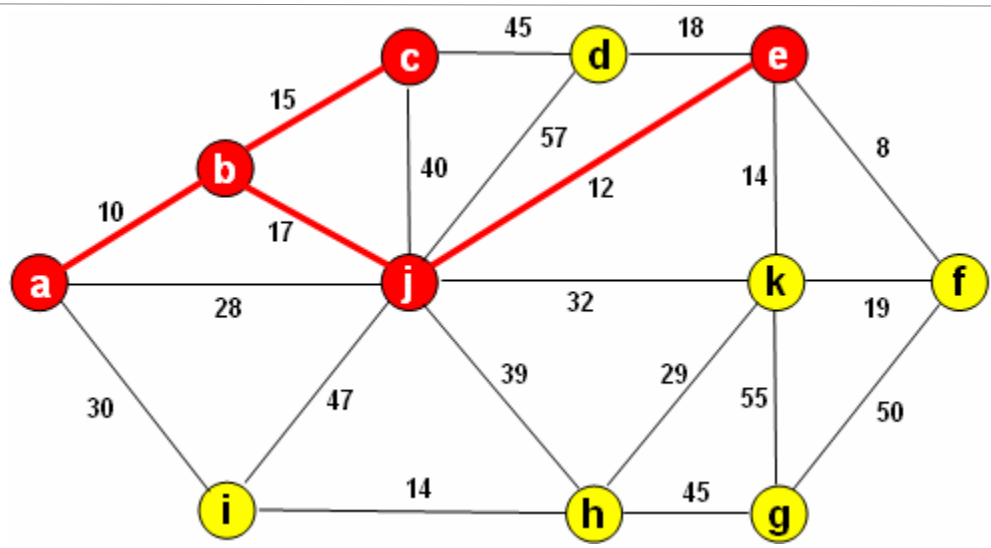
Prim's Algorithm

Example

(9) Initially, *V={ d, e, f, g, h, i, k}*
S={a, b, c, j}
The distances of vertices in set *S* to the *linked* vertices in set *V* are as follows:
a→i=30 , c→d= 45,
j→d=57, j→e=12 (minimum),
j→k=32, j→i=47,
j→h=39
The minimum distance 12 is from *j* to *e*



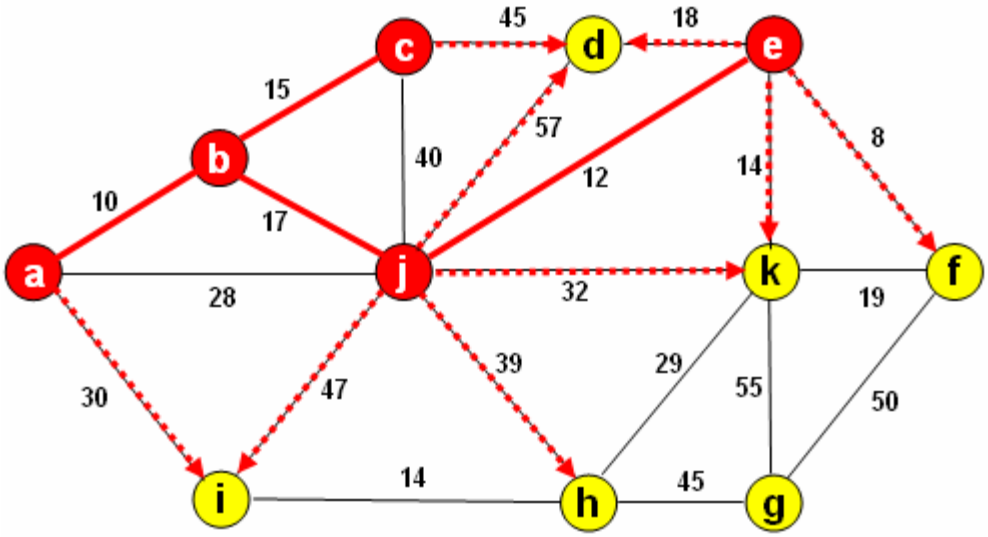
(10) Vertex *e,* which has the *minimum distance,* is selected and placed in set *S*. Thus, The *edge (j,e,12)* of MST is placed in set *T :*
V={d, f, g, h, i, k}
S={a, b, c, e, j}
T={(a,b,10), (b,c,15), (b,j,17), (j,e,12)}
The total distance so far is *54*



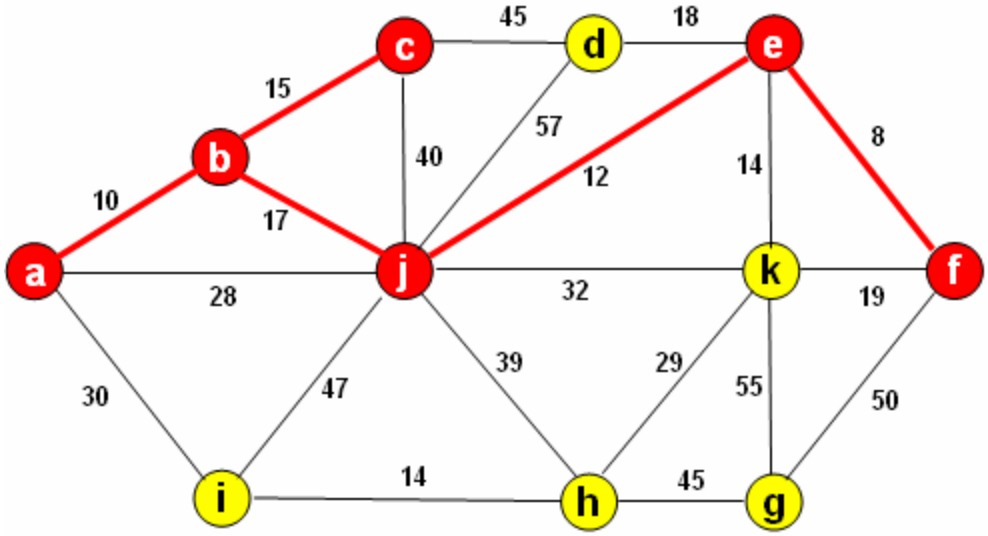
Prim's Algorithm

Example

(11) Initially, *V={ d, f, g, h, I, k}*
S={a, b, c, e, j}
The distances of vertices in set *S* to the *linked* vertices in set *V* are as follows:
a→i=30 , c→d= 45,
j→d=57, j→k=32,
j→i=47, j→h=39
e→d=18, e→f=8 (minimum)
e→k=14
The minimum distance *8* is from *e* to *f*



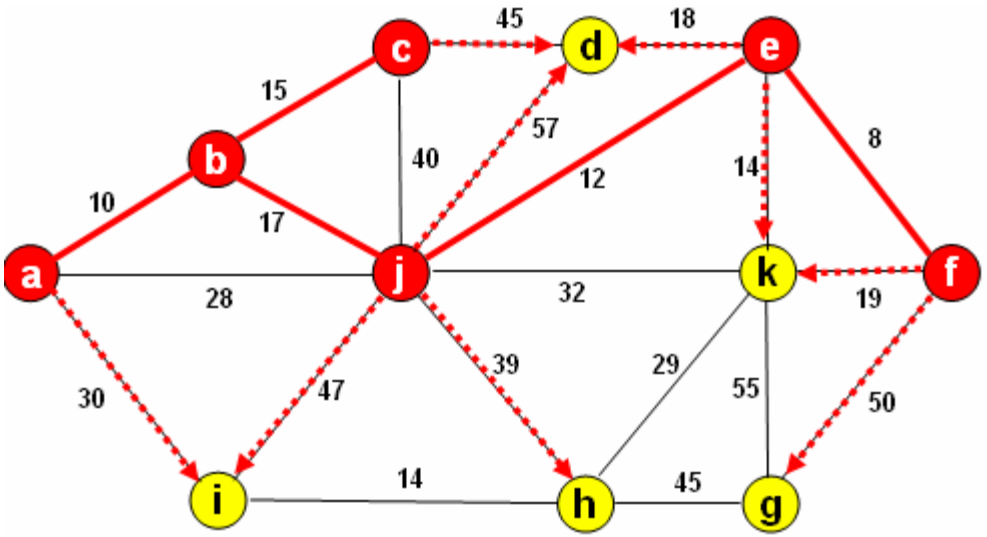
(12) Vertex *f,* which has the *minimum distance,* is selected and placed in set *S*.
Thus, The *edge (e,f,8)* of MST is placed in set *T :*
V={d, g, h, i k}
S={a, b, c, e, f, j}
T={(a,b,10), (b,c,15), (b,j,17), (j,e,12), (e,f,8)}
The total distance so far is *62*



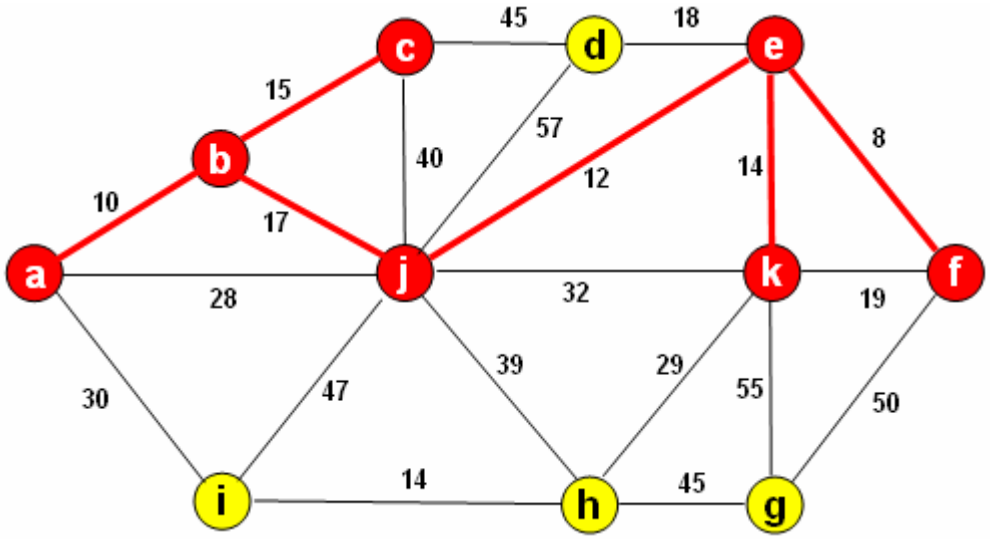
Prim's Algorithm

Example

(13) Initially, *V={ d, g, h, i, k}*
S={a, b, c, e, f, j}
The distances of vertices in set *S* to the *linked* vertices in set *V* are as follows:
a→i=30 , c→d= 45,
j→d=57, j→k=32,
j→i=47, j→h=39
e→d=18, f→k=19
f→g=50, e→k=14 (minimum)
The minimum distance *14* is from *e* to *k*



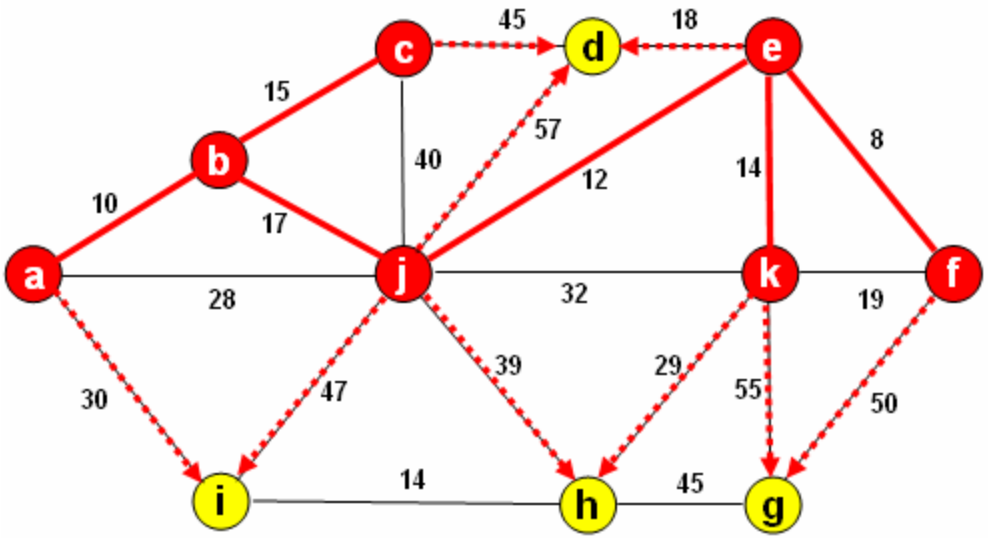
(14) Vertex *k,* which has the *minimum distance,* is selected and placed in set *S*.
Thus, The *edge (e,k,14)* of MST is placed in set *T :*
V={d, g, h, i}
S={a, b, c, e, f, j, k}
T={(a,b,10), (b,c,15), (b,j,17), (j,e,12), (e,f,8), (e,k,14)}
The total distance so far is *76*



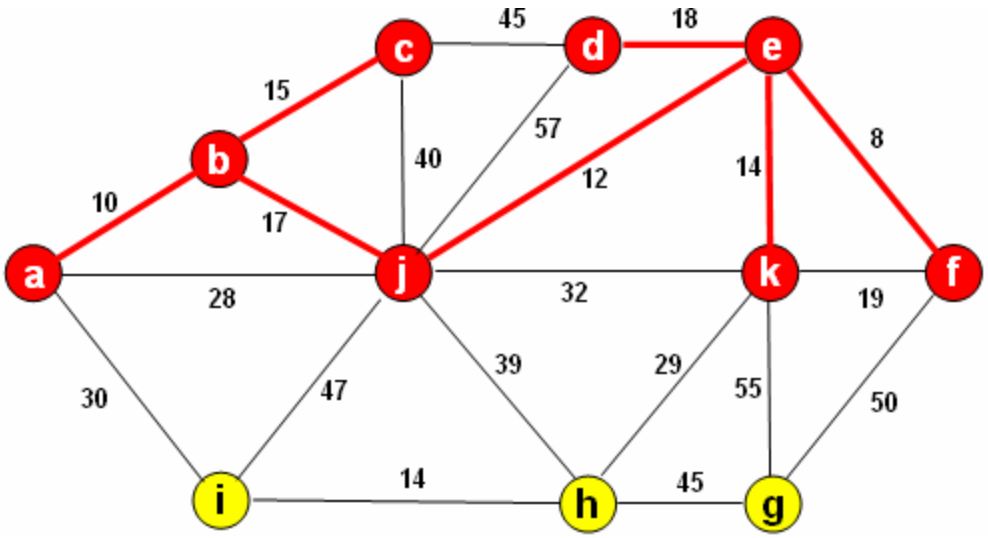
Prim's Algorithm

Example

(15) Initially, *V={ d, g, h, i}*
S={a, b, c, e, f, j, k}
The distances of vertices in set *S* to the *linked* vertices in set *V* are as follows:
a→i=30 , c→d= 45,
j→d=57, j→i=47,
j→h=39, e→d=18 (minimum)
f→g=50, k→h=29, k→g=55
The minimum distance *14* is from *e* to *k*



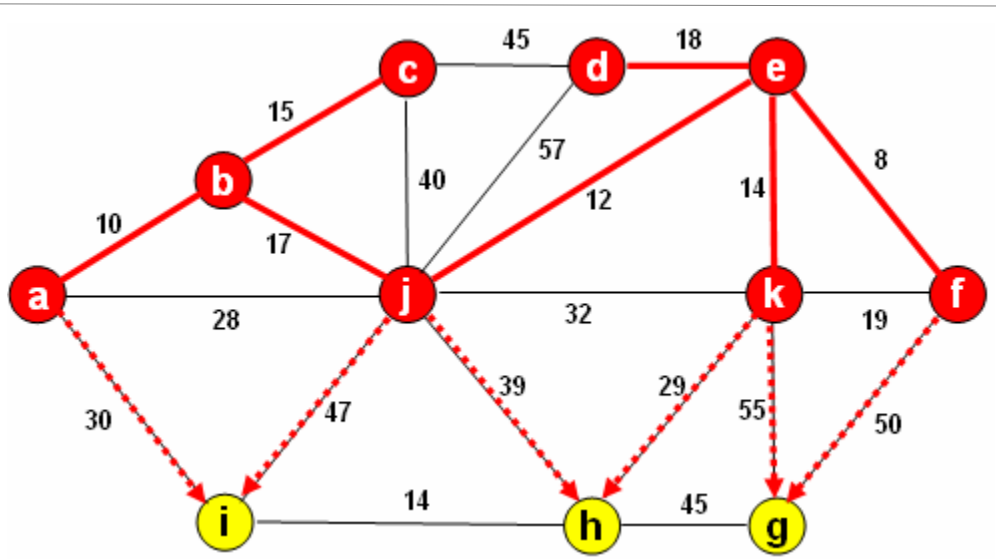
(16) Vertex *d,* which has the *minimum distance,* is selected and placed in set *S*.
Thus, The *edge (e,d,18)* of MST is placed in set *T :*
V={ g, h, i}
S={a, b, c, d, e, f, j, k}
T={(a,b,10), (b,c,15), (b,j,17), (j,e,12), (e,f,8), (e,k,14), (e,d,18)}
The total distance so far is *94*



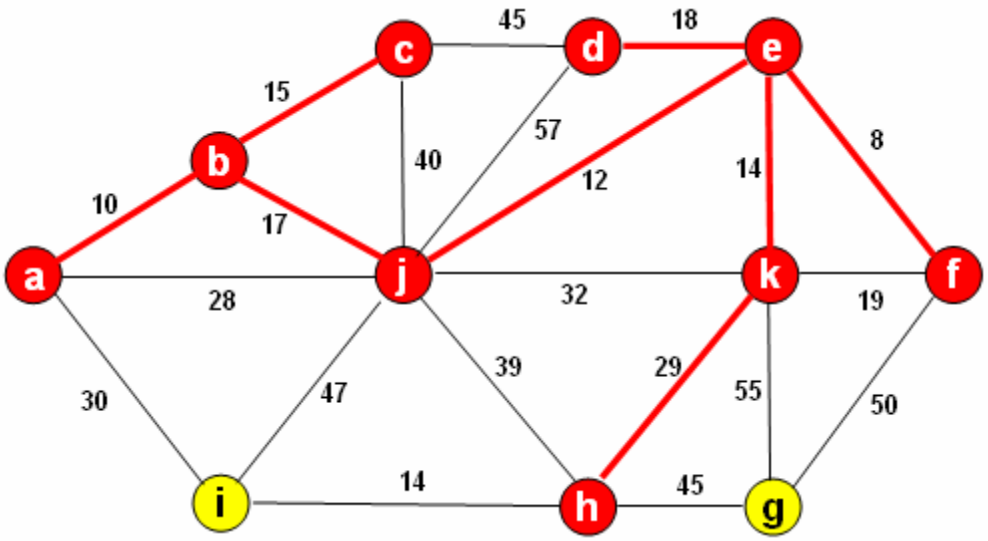
Prim's Algorithm

Example

(17) Initially, *V={ g, h, i}*
S={a, b, c ,d, e, f, j, k}
The distances of vertices in set *S* to the *linked* vertices in set *V* are as follows:
a→i=30 ,
j→i=47, *j→h=39*,
f→g=50, *k→h=29* *(minimum)*
k→g=55
The minimum distance *29* is from *k* to *h*



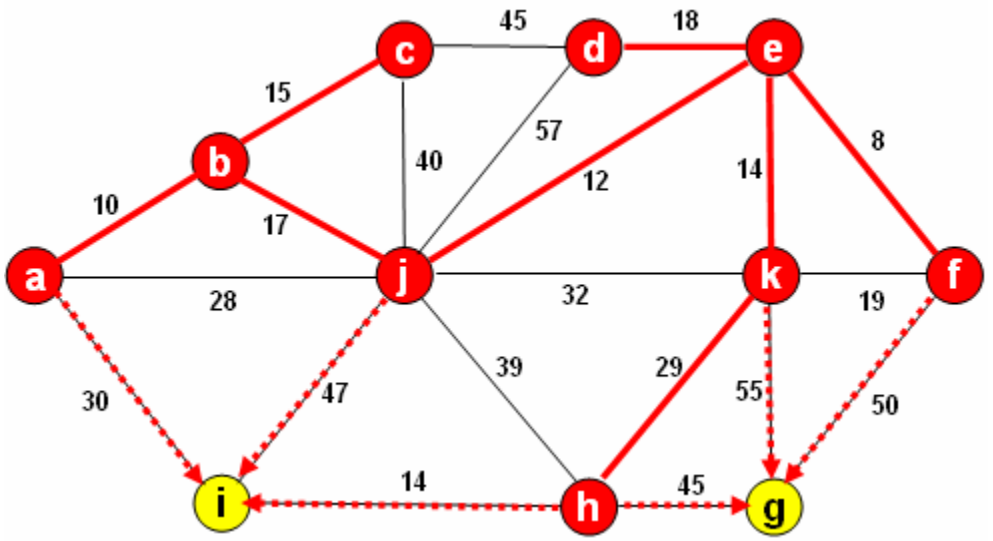
(18) Vertex *h,* which has the *minimum distance,* is selected and placed in set *S*.
Thus, The *edge (k,h,29)* of MST is placed in set *T* :
V={ g, i}
S={a, b, c, d, e, f, h, j, k}
T={(a,b,10), (b,c,15), (b,j,17), (j,e,12), (e,f,8), (e,k,14), (e,d,18), (k,h,29)}
The total distance so far is *123*



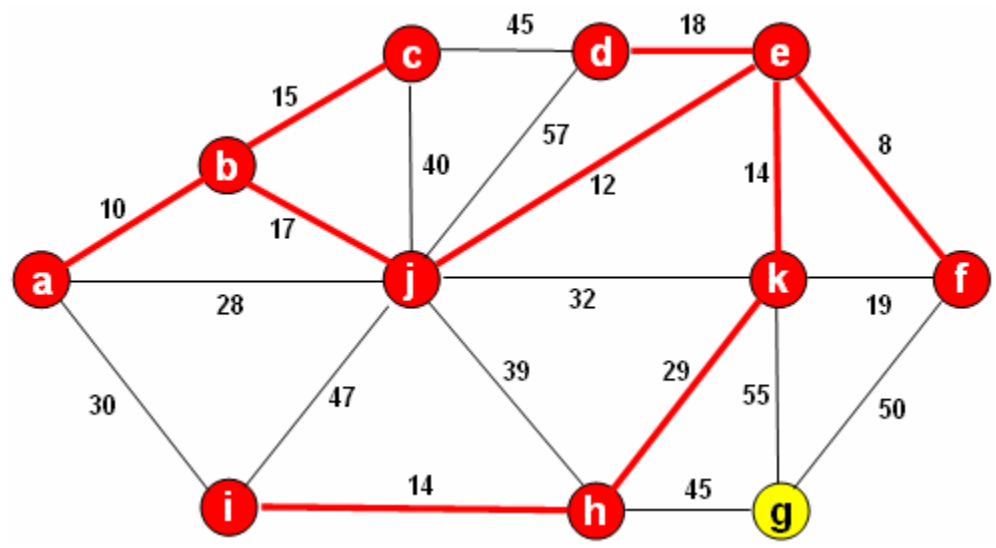
Prim's Algorithm

Example

(19) Initially, *V={ g, i}*
S={a, b, c ,d, e, f, h, j, k}
The distances of vertices in set *S* to the *linked* vertices in set *V* are as follows:
a→i=30 , j→i=47,
h→i=14 (minimum)
f→g=50, h→g=45
k→g=55
The minimum distance *30* is from *k* to *h*



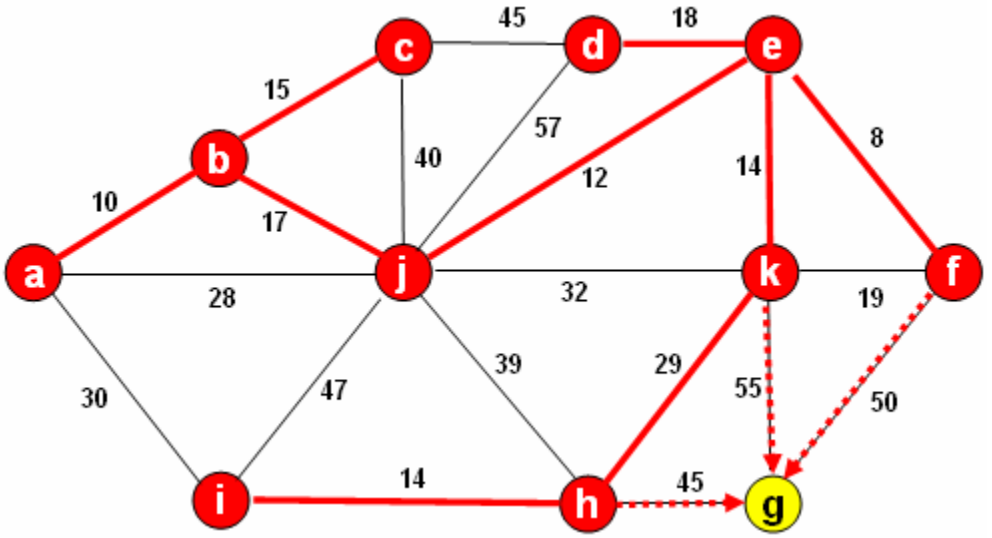
(20) Vertex *i,* which has the *minimum distance,* is selected and placed in set *S*.
Thus, The *edge (h,i,14)* of MST is placed in set *T :*
V={ g}
S={a, b, c, d, e, f, h, i, j, k}
T={(a,b,10), (b,c,15), (b,j,17), (j,e,12), (e,f,8), (e,k,14), (e,d,18), (k,h,29),(h,i,14)}
The total distance so far is *137*



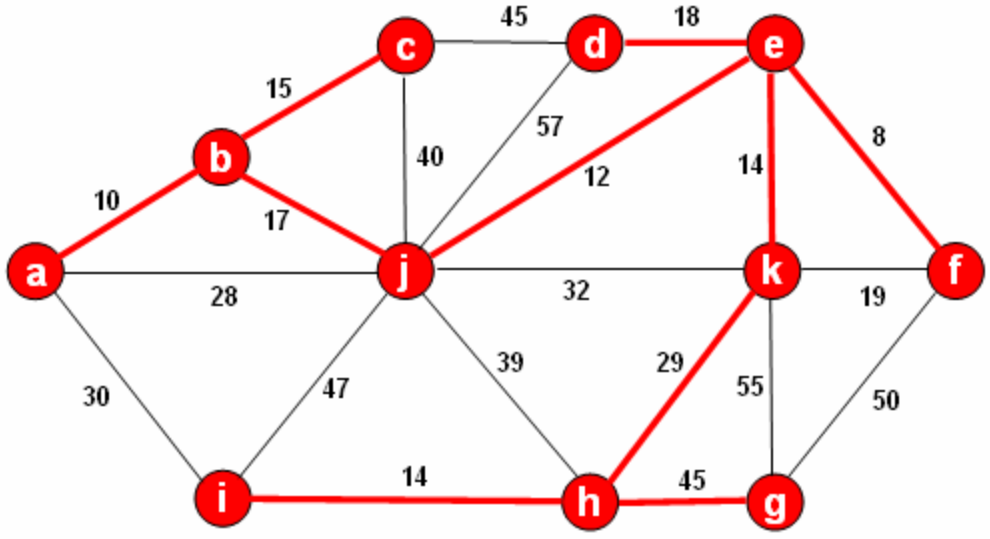
Prim's Algorithm

Example

(21) Initially, *V={ g}*
S={a, b, c ,d, e, f, h, j, k}
The distances of vertices in set *S* to the *linked* vertices in set *V* are as follows:
f→g=50,
h→g=45 (minimum)
k→g=55
The minimum distance *30* is from *k* to *h*



(22) Vertex *g,* which has the *minimum distance,* is selected and placed in set *S*.
Thus, The *edge (h,g,45)* of MST is placed in set *T* .
Set *V* is *empty*. *S={a, b, c, d, e, f, g, h, i, j, k}*
T={(a,b,10), (b,c,15), (b,j,17), (j,e,12), (e,f,8), (e,k,14), (e,d,18), (k,h,29), (h,i,14), (h,g,45)}.The total distance so far is *182*



Prim's Algorithm

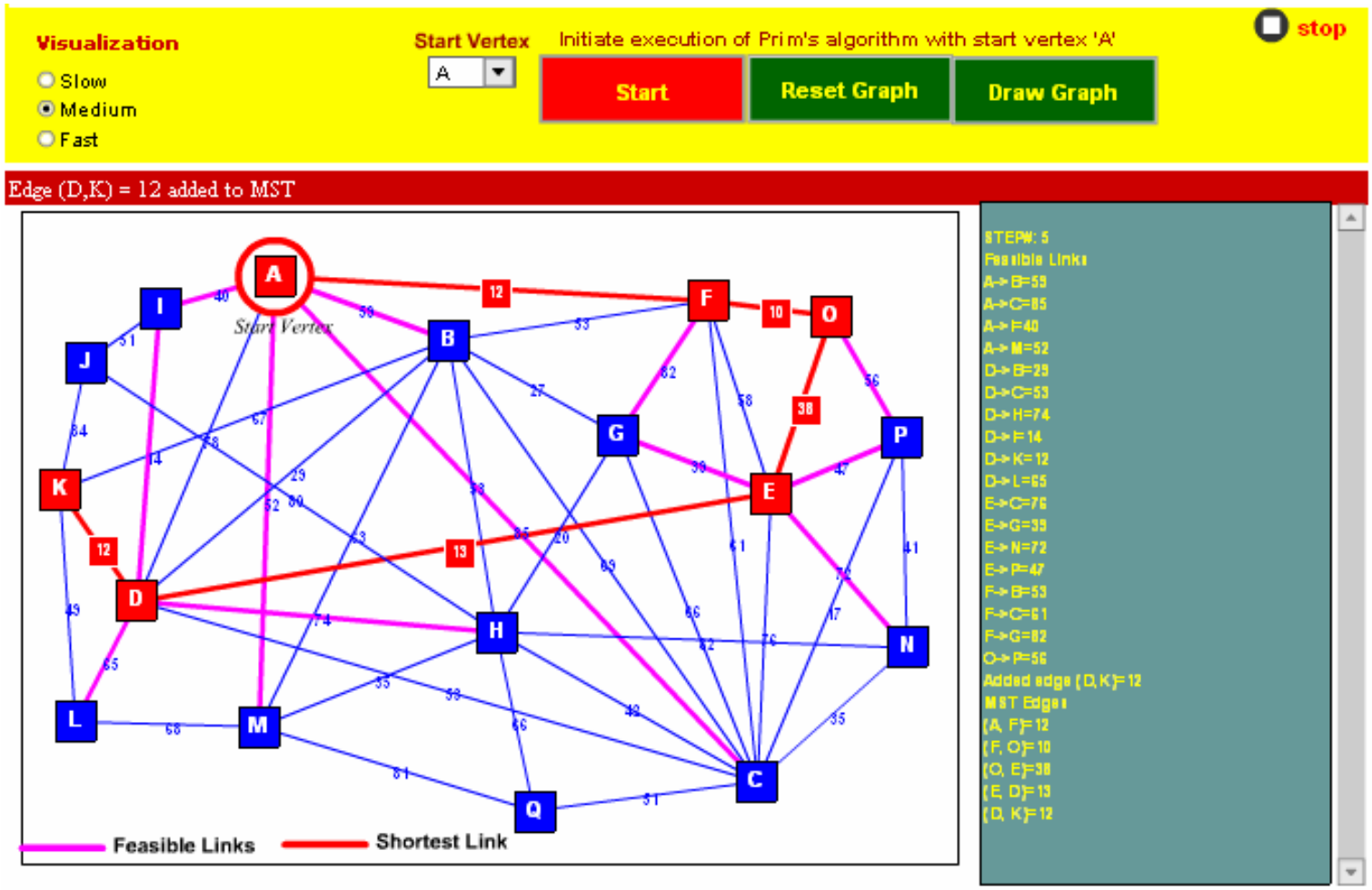
Implementation

The MST-PRIM method implements Prim's algorithm for a graph G. A start vertex **s** is passed to the method.

```
MST-PRIM(G, s)
1  T ← φ                          ► T holds minimum spanning tree
2  S ← φ                          ► S contains vertices added to the tree
3  S ← S ∪ {s}                    ► Start vertex is added to S
4   V ← V-{s}                     ► Start vertex is removed from V
5   while V ≠ φ do
6  d ← ∞  ►d is used to store minimum distance . It is initialized to some very large number
7       for each vertex u ∈ S  do          ►Select a vertex  u in S
8           for each vertex v  ∈ Adj[u] do ►Examine all vertices which are linked to u
9               if d > w(u,v)              ► Check if the vertex u has shorter distance
10                  then  d ← w(u,v)       ► If yes, replace d with weight w (u, v)
11                        w1 ← v           ► store vertex v in w1
12                        w2 ← u           ►store vertex u in w2
13    V ← V - {w1}                         ►remove vertex w1 from V
14    S  ←  S ∪ {w1}                       ►add vertex w1 to S
15  T ← T ∪ {(w1,w2)}                      ►Add edge (w1,w2) to T
16 return T
```

Prim Visualization

Prim's Algorithm Visualization



Analysis of Prim's Algorithm

Running Time

The *while loop* in the implementation is executed $|V|-1$ times, where $|V|$ represents the number of vertices in the graph. In each iteration, distances of a given vertex from other vertices is explored at most $|V|-1$ *times*, in worst case. Thus, the running time of Prim's algorithm is

$T_{prim} = O((|V|-1).(|V|-1))$

$= O(|V|^2)$

$= O(n^2)$, n being the total number of vertices in the graph.

A more efficient implementation of Prim's algorithm (*Ref. T.Cormen et al*) uses a *min-heap* to extract an edge with smallest weight from a given vertex. In this case, it can be shown that the running time is

$T_{prim} = O(V \lg V + E \lg V)$

$= O(E \lg V)$

.