

Minimum Spanning Tree (MST)

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

A weighted graph is a graph where a real number called **weight** is associated with each edge.

A **spanning tree** of G is a subgraph T of G which is a tree that spans all vertices of G . In other words, T contains all of the vertices of G .

Minimum Spanning Tree

The **weight of a spanning tree** T is the sum of the weights of its edges. That is,

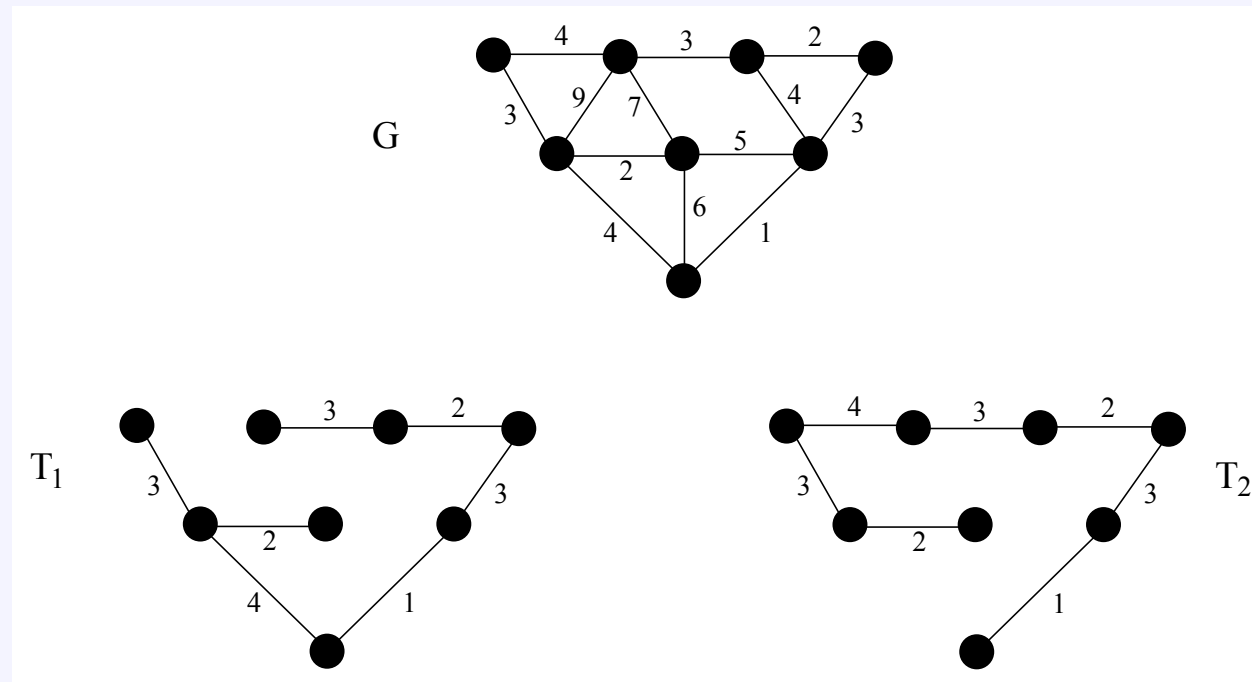
$$w(T) = \sum_{(u,v) \in T} w(u, v).$$

A **minimum spanning tree (MST)** of G is spanning tree T of minimum weight.

It should be clear that a minimum spanning tree always exists.

Minimum Spanning Tree : Examples

- Here is an example of a graph and two minimum spanning trees.



Constructing an MST

We have proved that the MST problem exhibit the optimal substructure and greedy properties.

Therefore minimum spanning trees can be constructed using greedy algorithms.

There are two common greedy algorithms to construct MSTs :

- ▶ **Kruskal's algorithm**
- ▶ **Prim's algorithm**

Both of these algorithms use the same basic ideas, but in a slightly different fashion.

Prim's Algorithm—More Details

For each node x , we store

- ▶ The predecessor $p(x)$. This is the vertex y in T which we join x to when edge (x, y) is added to T .
- ▶ The $key(x)$. The weight of the minimum weight edge that connects x to some vertex in T .

$key(r) = 0$, and $p(r) = NIL$ throughout.

Each node $x \neq r$ starts with $key(x) = \infty$.

The value $key(x)$ only changes if some node adjacent to x is added to T .

Prim's Algorithm—More Details

Thus, when a node y is added to T , the *key* values of the nodes adjacent to y is updated

The vertices in $V - T$ are stored in a priority queue Q based on $key(x)$. This allows to pick the minimum weight edge to add to T .

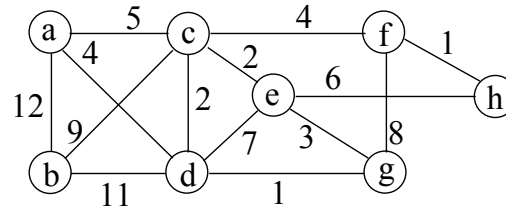
T is not stored explicitly. The MST is reconstructed using the predecessors $p(x)$ for all $p \neq r$.

Prim's Algorithm

```
Prim_MST( $G, r$ )  
   $\forall u \in G$   
     $key[u] = \text{Max\_Int};$   
   $key[r] = 0;$   
   $p[r] = \text{NIL};$   
   $Q = \text{MinPriorityQueue}(V[G])$   
  while ( $Q \neq \emptyset$ )  
     $u = \text{ExtractMin}(Q);$   
    for each  $v$  adjacent to  $u$   
      if ( $(v \in Q) \ \& \ (w(u, v) < key[v])$ )  
         $p[v] = u;$   
         $key[v] = w(u, v);$ 
```

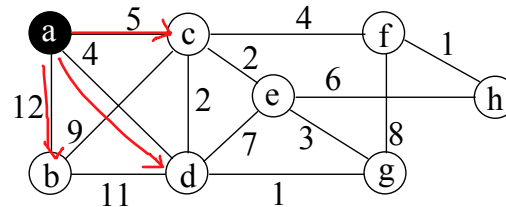
```

while ( $Q \neq \emptyset$ )
     $u = \text{ExtractMin}(Q)$ ;
    for each  $v$  adjacent to  $u$ 
        if ( $(v \in Q)$ 
            & ( $w(u, v) < \text{key}[v]$ ))
             $p[v] = u$ ;
             $\text{key}[v] = w(u, v)$ ;
    
```



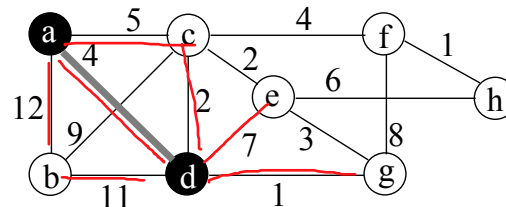
v	a	b	c	d	e	f	g	h
k	0	∞	∞	∞	∞	∞	∞	∞
p	nil	?	?	?	?	?	?	?

$Q=[a,b,c,d,e,f,g,h]$



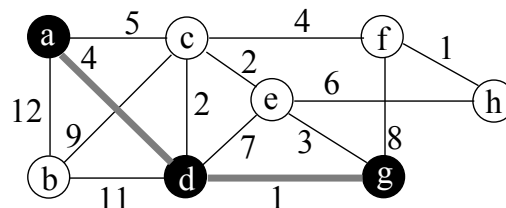
v	a	b	c	d	e	f	g	h
k	∞	12	5	9	∞	∞	∞	∞
p	nil	<u>a</u>	<u>a</u>	<u>a</u>	?	?	?	?

$Q=[d,c,b,e,f,g,h]$



v	a	b	c	d	e	f	g	h
k	∞	11	2	∞	7	∞	1	∞
p	nil	d	d	a	d	?	d	?

$Q=[g,c,e,b,f,h]$

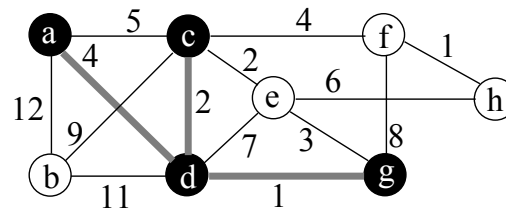


v	a	b	c	d	e	f	g	h
k	∞	11	2	∞	3	8	∞	∞
p	nil	d	d	a	g	g	d	?

$Q=[c,e,f,b,h]$

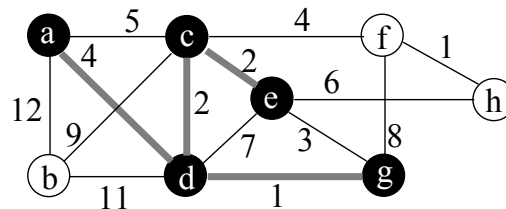

```

while ( $Q \neq \emptyset$ )
     $u = \text{ExtractMin}(Q)$ ;
    for each  $v$  adjacent to  $u$ 
        if ( $(v \in Q)$ 
            & ( $w(u, v) < \text{key}[v]$ ))
             $p[v] = u$ ;
             $\text{key}[v] = w(u, v)$ ;
    
```



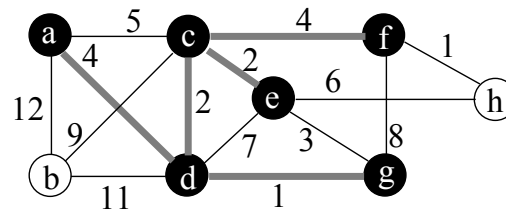
v	a	b	c	d	e	f	g	h
k	∞	9	∞	∞	2	4	∞	∞
p	nil	c	d	a	c	c	d	?

$Q = \{e, f, b, h\}$



v	a	b	c	d	e	f	g	h
k	∞	9	∞	∞	∞	4	∞	6
p	nil	c	d	a	c	c	d	e

$Q = \{f, h, b\}$

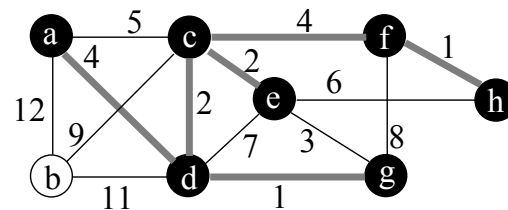


v	a	b	c	d	e	f	g	h
k	∞	9	∞	∞	∞	∞	∞	1
p	nil	c	d	a	c	c	d	f

$Q = \{h, b\}$

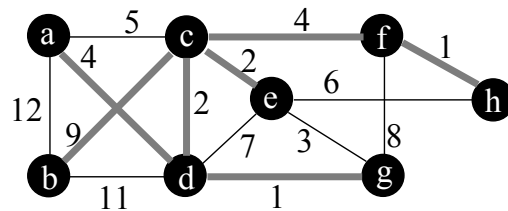
```

while ( $Q \neq \emptyset$ )
   $u = \text{ExtractMin}(Q)$ ;
  for each  $v$  adjacent to  $u$ 
    if ( $(v \in Q)$ 
      & ( $w(u, v) < \text{key}[v]$ ))
       $p[v] = u$ ;
       $\text{key}[v] = w(u, v)$ ;
  
```



v	a	b	c	d	e	f	g	h
k	∞	9	∞	∞	∞	∞	∞	∞
p	nil	c	d	a	c	c	d	f

$Q=[b]$



v	a	b	c	d	e	f	g	h
k	∞	∞	∞	∞	∞	∞	∞	∞
p	nil	c	d	a	c	c	d	f

$Q=[]$

Prim's algorithm : running time

The running time of Prim's algorithm is $O(E \lg V)$.

Building the priority queue using min heap can be done in $O(V)$.

The **while** loop is performed V times

- ▶ ExtractMin cost $O(\lg V)$, total $O(V \lg V)$
- ▶ The cost of Prim's algorithm is driven by the **for** loop
 - ▶ This **for** loop is executed in total $2|E|$ times (the sum of the length of the adjacency list)
 - ▶ Each time it is executed, it can potentially change the value of $key[v]$
 - ▶ which means an update of the priority queue that cost $O(\lg V)$
- ▶ Total cost of the **for** loop is $O(E \lg V)$
- ▶ The total cost of the **while** loop is $O(V \lg V + E \lg V) \in O(E \lg V)$

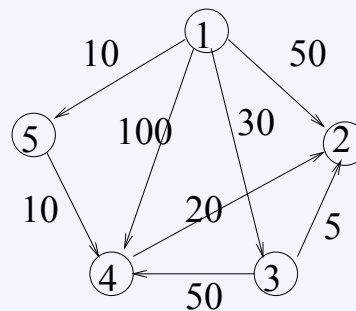
See also the analysis in Cormen, on page 636

Shortest Paths Problem

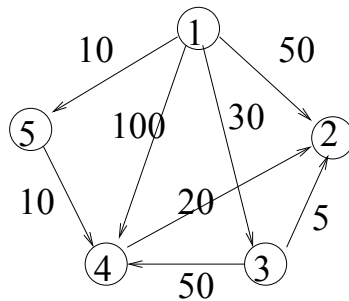
Let $G = \{V, A\}$ be a connected **directed** graph where V is the set of nodes and A is the set of arcs.

- ▶ Each arc $a \in A$ has a nonnegative length.
- ▶ One of the node is designated as the **source** node.
- ▶ The problem is to determine **the length of the shortest path from the source node to each of the other nodes of the graph**

This problem can be solved by a greedy algorithm called **Dijkstra's algorithm**



Dijkstra's algorithm



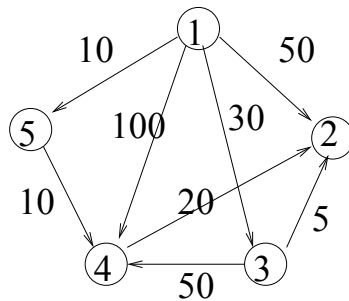
Step	v	C	D	S
init		{2,3,4,5}	{50,30,100,10}	{1}
1	5	{2,3,4}	{50,30,20,10}	{1,5}
2	4	{2,3}	{40,30,20,10}	{1,4,5}
3	3	{2}	{35,30,20,10}	{1,3,4,5}
				{1,2,3,4,5}

C is the candidate set ($V \setminus$ source node)

S be the set of selected nodes, i.e. the nodes whose minimal distance from the source is already known

A path from the source to another node is **special** if all intermediate nodes along the path belong to S

Dijkstra's algorithm

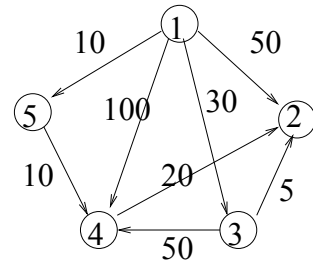


Step	v	C	D	S
init		{2,3,4,5}	{50,30,100,10}	{1}
1	5	{2,3,4}	{50,30,20,10}	{1,5}
2	4	{2,3}	{40,30,20,10}	{1,4,5}
3	3	{2}	{35,30,20,10}	{1,3,4,5}
				{1,2,3,4,5}

Let D be an array such that at each step of the algo., D contains the length of the shortest special path to each node in the graph

When a new node v is added to S , the shortest special path to v is also the shortest of all the paths to v

Dijkstra's algorithm



Step	v	C	D	S
init		{2,3,4,5}	{50,30,100,10}	{1}
1	5	{2,3,4}	{50,30,20,10}	{1,5}
2	4	{2,3}	{40,30,20,10}	{1,4,5}
3	3	{2}	{35,30,20,10}	{1,3,4,5}
				{1,2,3,4,5}

Algorithm Dijkstra(G)

$C = \{2, 3, \dots, n\}$ $\{S = V \setminus C\}$

for $i = 2$ **to** n **do** $D[i] = L[1, i]$

repeat $n - 1$ **times**

$v =$ some element of C minimizing $D[v]$

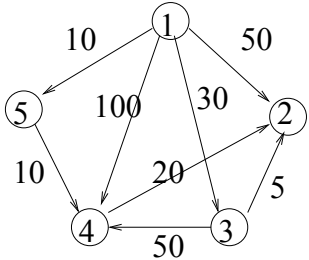
$C = C \setminus \{v\}$

for each $w \in C$ **do**

$D[w] = \min(D[w], D[v] + L[v, w])$

return D

Dijkstra's algorithm

	Step	v	C	D	S
	init		{2,3,4,5}	{50,30,100,10}	{1}
	1	5	{2,3,4}	{50,30,20,10}	{1,5}
	2	4	{2,3}	{40,30,20,10}	{1,4,5}
	3	3	{2}	{35,30,20,10}	{1,3,4,5}
					{1,2,3,4,5}

Algorithm Dijkstra(G)

$C = \{2, 3, \dots, n\}$ $\{S = V \setminus C\}$

for $i = 2$ **to** n **do** $D[i] = L[1, i]$

repeat $n - 1$ **times**

$v =$ some element of C minimizing $D[v]$

$C = C \setminus \{v\}$

for each $w \in C$ **do**

$D[w] = \min(D[w], D[v] + L[v, w])$

return D

Running time $O(|V|^2)$.