Lecture 10: Minimum Spanning Trees and Prim's Algorithm

CLRS Chapter 23

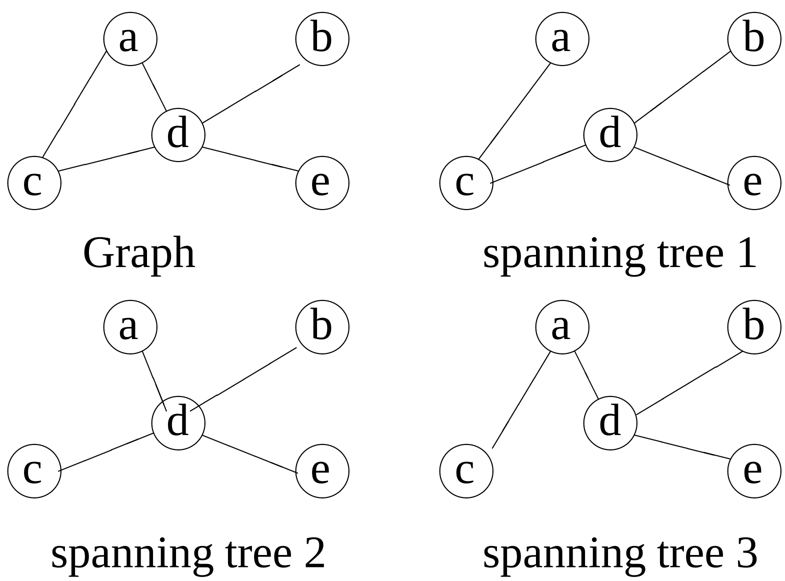
Outline of this Lecture

- Spanning trees and minimum spanning trees.
- The minimum spanning tree (MST) problem.
- Prim's algorithm for the MST problem.
  - The algorithm
  - Correctness
  - Implementation + Running Time

Spanning Trees

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.

Example:

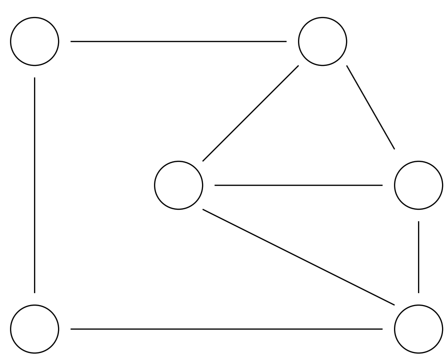


Spanning Trees

Theorem: Every connected graph has a spanning tree.

Question: Why is this true?

Question: Given a connected graph G, how can you find a spanning tree of G?

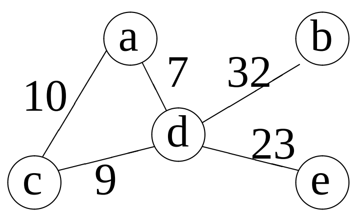


Weighted Graphs

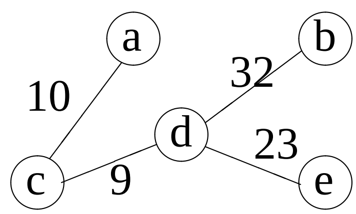
Weighted Graphs: A weighted graph is a graph, in which each edge has a weight (some real number).

Weight of a Graph: The sum of the weights of all edges.

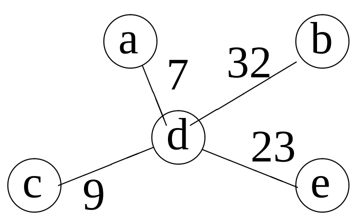
Example:

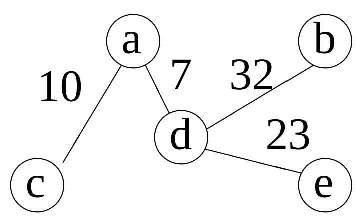


weighted graph



Tree 1. w=74





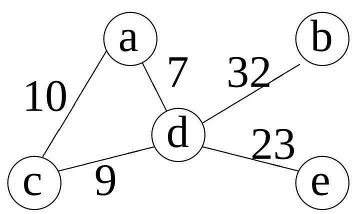
Tree 2, w=71
Minimum spanning tree

Tree 3, w=72

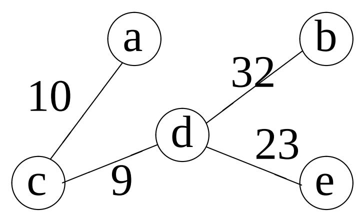
Minimum Spanning Trees

A **Minimum Spanning Tree** in an undirected connected weighted graph is a spanning tree of minimum weight (among all spanning trees).

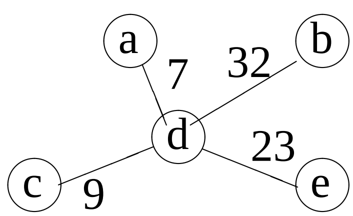
Example:



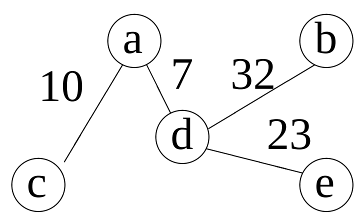
weighted graph



Tree 1. w=74



Tree 2, w=71
Minimum spanning tree

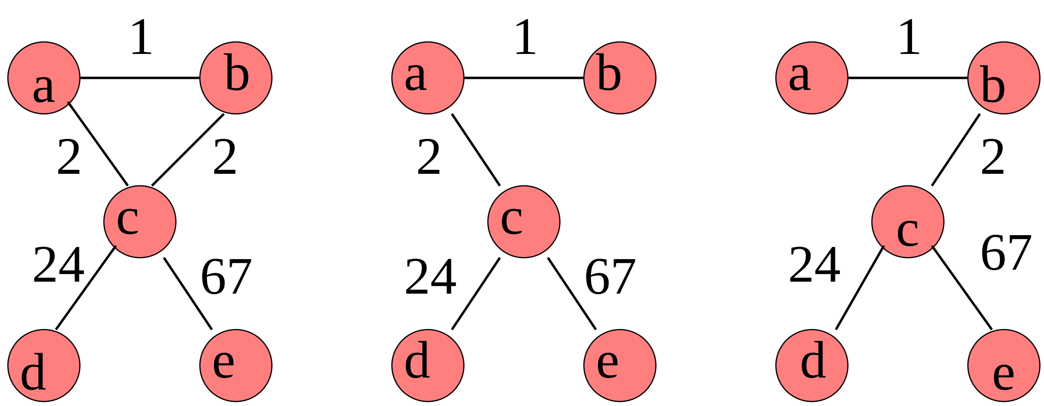


Tree 3, w=72

Minimum Spanning Trees

Remark: The minimum spanning tree may not be unique. However, if the weights of all the edges are pairwise distinct, it is indeed unique (we won't prove this now).

Example:



weighted graph   MST 1   MST 2

Minimum Spanning Tree Problem

MST Problem: Given a connected weighted undirected graph G, design an algorithm that outputs a minimum spanning tree (MST) of G.

Question: What is most intuitive way to solve?

Generic approach: A tree is an acyclic graph.
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.

We introduce two greedy algorithms (Prim's and Kruskal's algorithms) for computing a MST.
They differ in how to choose edges to add.

Greedy: make the cheapest possible choice in each step.

What is Prim's Algorithm?

- A greedy algorithm for the MST problem.

- Looks very much like Dijkstra's algorithm:
  Grow a Tree

  – Start by picking any vertex r to be the root of the tree.

  – While the tree does not contain
    all vertices in the graph
    find shortest edge leaving the tree
    and it to the tree .

- Running time is $O((|V| + |E|) \log |V|)$.

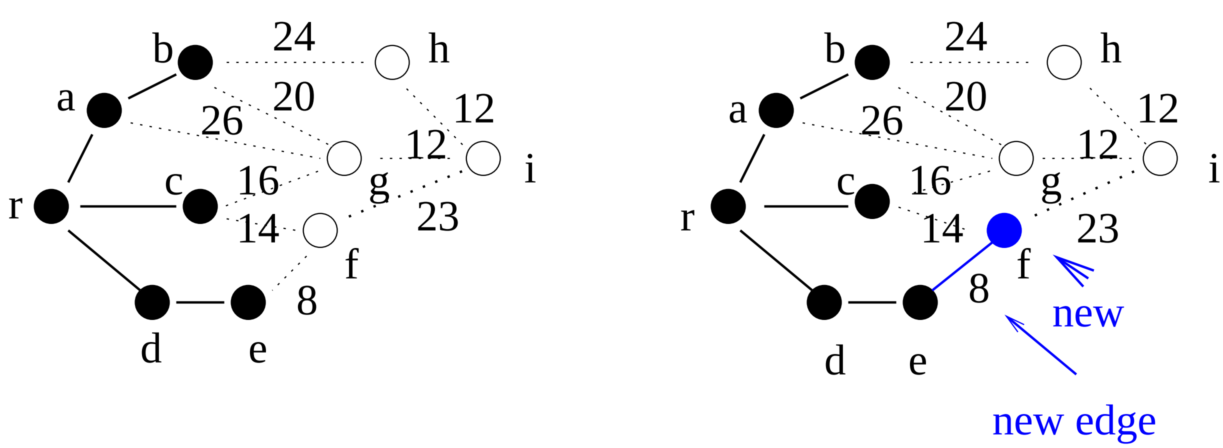
More Details

Step 0: Choose any element r; set $S = \{r\}$ and $A = \emptyset$. (Take r as the root of our spanning tree.)

Step 1: Find a lightest edge such that one endpoint is in S and the other is in $V \setminus S$. Add this edge to A and its (other) endpoint to S.

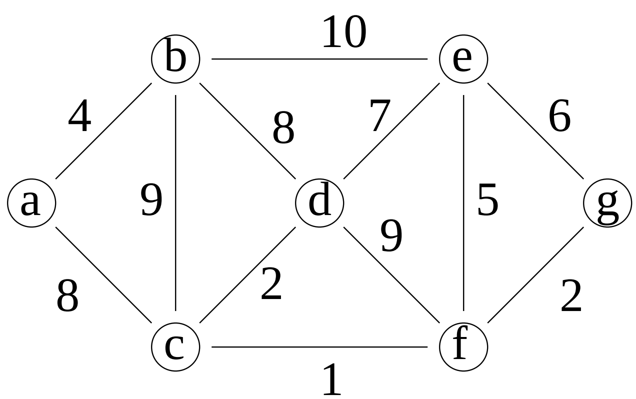
Step 2: If $V \setminus S = \emptyset$, then stop & output (minimum) spanning tree (S, A). Otherwise go to Step 1.

The idea: expand the current tree by adding the lightest (shortest) edge leaving it and its endpoint.

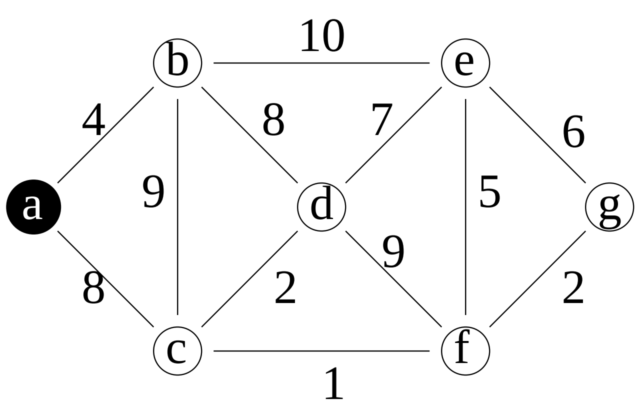


Prim’s Algorithm

Worked Example



Connected graph



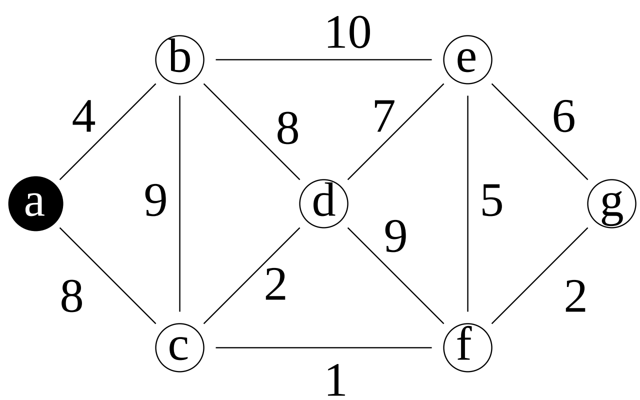
Step 0

S={a}

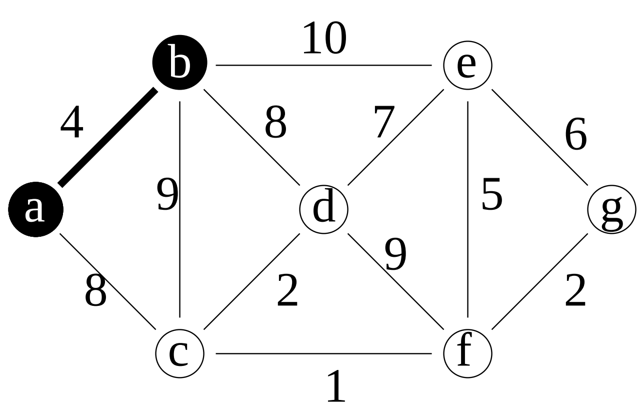
V \ S = {b,c,d,e,f,g}

lightest edge = {a,b}

Prim's Example – Continued



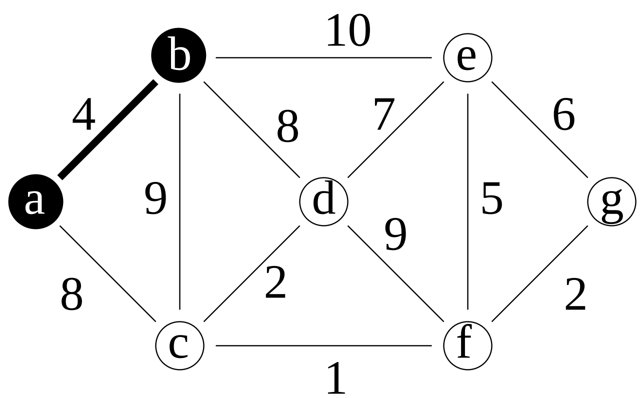
Step 1.1 before
S={a}
V \ S = {b,c,d,e,f,g}
A={}
lightest edge = {a,b}



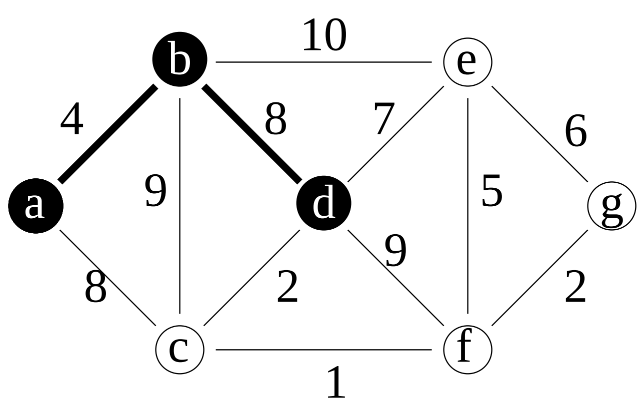
Step 1.1 after
S={a,b}
V \ S = {c,d,e,f,g}
A={{a,b}}
lightest edge = {b,d}, {a,c}

Prim’s Algorithm

Prim’s Example – Continued



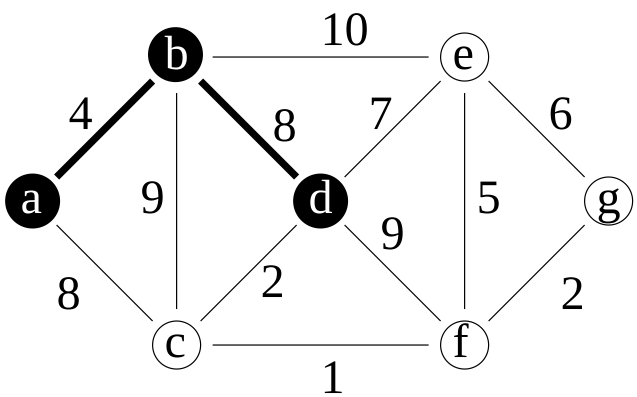
Step 1.2 before
S={a,b}
V \ S = {c,d,e,f,g}
A={{a,b}}
lightest edge = {b,d}, {a,c}



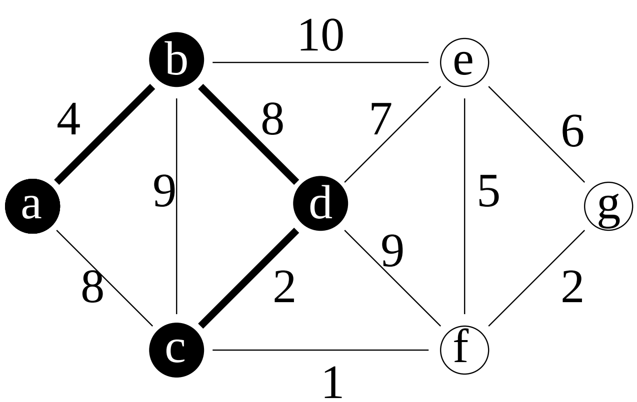
Step 1.2 after
S={a,b,d}
V \ S = {c,e,f,g}
A={{a,b},{b,d}}
lightest edge = {d,c}

Prim's Algorithm

Prim's Example – Continued



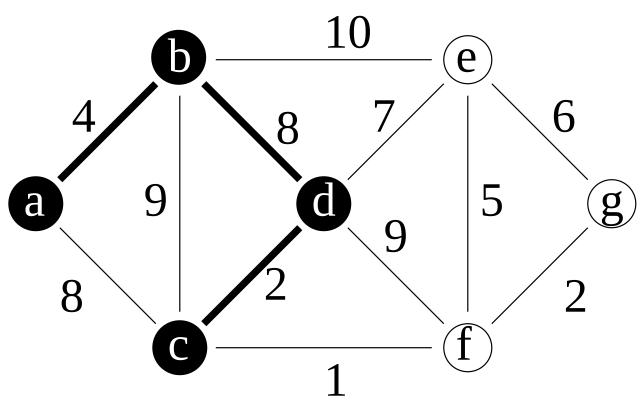
Step 1.3 before
S={a,b,d}
V \ S = {c,e,f,g}
A={{a,b},{b,d}}
lightest edge = {d,c}



Step 1.3 after
S={a,b,c,d}
V \ S = {e,f,g}
A={{a,b},{b,d},{c,d}}
lightest edge = {c,f}

Prim's Algorithm

Prim's Example – Continued

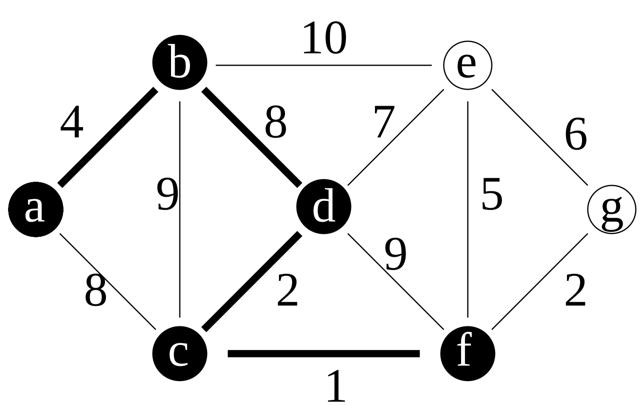


Step 1.4 before

S={a,b,c,d}

V \ S = {e,f,g}

A={{a,b},{b,d},{c,d}}
lightest edge = {c,f}



Step 1.4 after

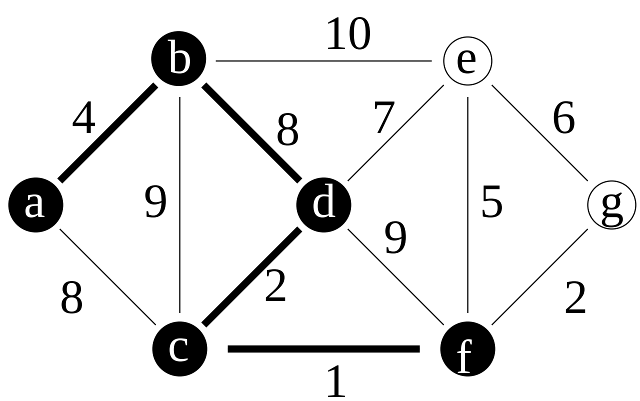
S={a,b,c,d,f}
V \ S = {e,g}

A={{a,b},{b,d},{c,d},{c,f}}

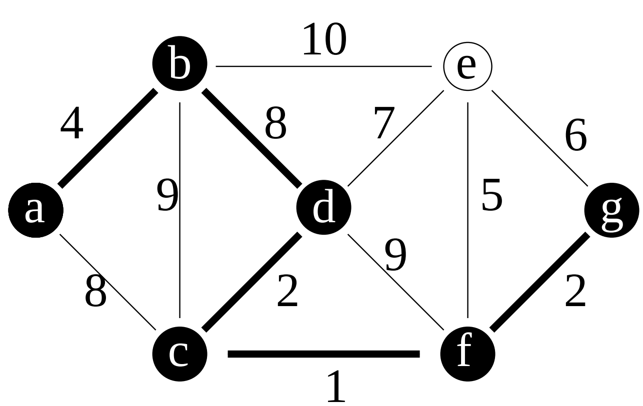
lightest edge = {f,g}

Prim's Algorithm

Prim's Example – Continued



Step 1.5 before
S={a,b,c,d,f}
V \ S = {e,g}
A={{a,b},{b,d},{c,d},{c,f}}
lightest edge = {f,g}



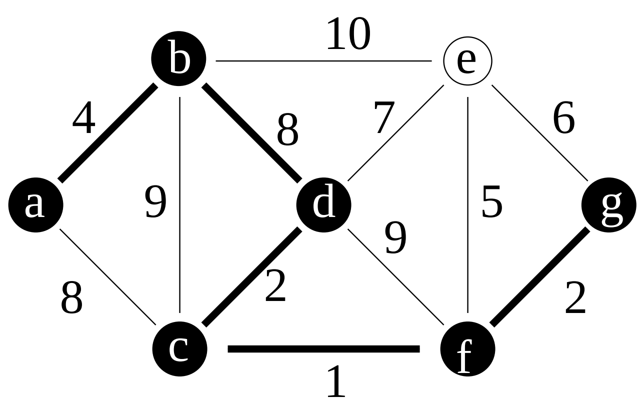
Step 1.5 after

S={a,b,c,d,f,g}
V \ S = {e}
A={{a,b},{b,d},{c,d},{c,f},
{f,g}}

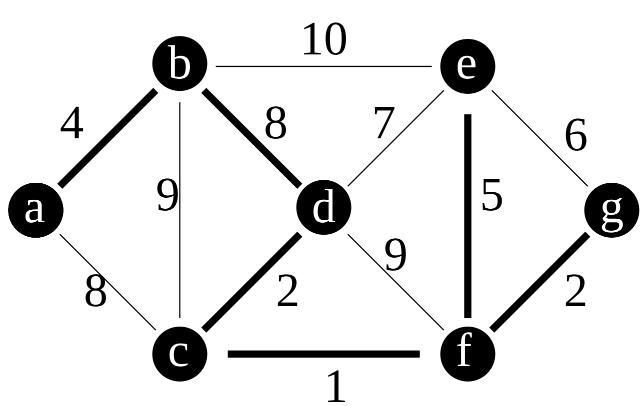
lightest edge = {f,e}

Prim's Algorithm

Prim's Example – Continued



Step 1.6 before
S={a,b,c,d,f,g}
V \ S = {e}
A={{a,b},{b,d},{c,d},{c,f},
{f,g}}
lightest edge = {f,e}



Step 1.6 after
S={a,b,c,d,e,f,g}
V \ S = {}
A={{a,b},{b,d},{c,d},{c,f},
{f,g},{f,e}}
MST completed

Recall Idea of Prim's Algorithm

Step 0: Choose any element r and set $S = \{r\}$ and $A = \emptyset$. (Take r as the root of our spanning tree.)

Step 1: Find a lightest edge such that one endpoint is in S and the other is in $V \setminus S$. Add this edge to A and its (other) endpoint to S.

Step 2: If $V \setminus S = \emptyset$, then stop and output the minimum spanning tree (S, A).
Otherwise go to Step 1.

Questions:

- Why does this produce a **Minimum** Spanning Tree?

- How does the algorithm find the lightest edge and update A efficiently?

- How does the algorithm update S efficiently?

Correctness of Prim's Algorithm

Lemma: Let (S, A) be a subtree of a MST of an undirected graph $G = (V, E)$, where $S \subset V$ and $A \subset E$. Let $e = \{u, v\}$ be an edge such that

(1) $u \in S$ and $v \in V \setminus S$;
(2) e has lowest weight among all the edges between a vertex in S and a vertex in $V \setminus S$.

Then $(S \cup \{v\}, A \cup \{e\})$ is a subtree of a MST.

Proof: Let T be a MST of G that contains (S, A).
If e is an edge of T, we are done.

Suppose that e is not an edge of T.
There is a unique path from u to v in T. There must be at least one edge $e' = \{u', v'\}$ in the path such that $u' \in S$ and $v' \in V \setminus S$. By (2) above,

$$W(e) \leq W(e'). \qquad (*)$$

Consider the new tree $T' := (T \cup \{e\}) \setminus \{e'\}$. Since T is MST,

$$W(T) \leq W(T') = W(T) - W(e') + W(e)$$

and so $W(e') \leq W(e)$. Combined with $(*)$, this proves that $W(e') = W(e)$, and so $W(T') = W(T)$. Therefore T' is also a MST, and T' contains $(S \cup \{v\}, A \cup \{e\})$.

Correctness of Prim's Algorithm

Lemma: Let (S, A) be a subtree of a MST of an undirected graph $G = (V, E)$, where $S \subset V$ and $A \subset E$. Let $e = \{u, v\}$ be an edge such that

(1) $u \in S$ and $v \in V \setminus S$;
(2) e has the lowest weight among all the edges between a vertex in S and a vertex in $V \setminus S$.

Then $(S \cup \{v\}, A \cup \{e\})$ is a subtree of a MST.

We can now prove the correctness of Prim's algorithm by induction.

When the algorithm starts, $(\{r\}, \emptyset)$ is definitely a subtree of a MST of G (why).

At each step the algorithm chooses an edge $e = \{u, v\}$ that satisfies (1) and (2) so, from the lemma, $(S \cup \{v\}, A \cup \{e\})$ remains a subtree of some MST of G.

In particular, when the algorithm ends, $S = V$ and A is a tree on V. We know from above that (S, A) is a subtree of some MST of G but, since A itself is a tree on G, this means that A itself is a MST.

Prim's Algorithm

Question: How does the algorithm update S efficiently?

Answer: Color the vertices. Initially all are white. Change the color to black when the vertex is moved to S. Use color[v] to store color.

Question: How does the algorithm find the lightest edge and update A efficiently?

Answer:
(a) Use a priority queue to find the lightest edge.
(b) Use pred[v] to update A.

Reviewing Priority Queues

Priority Queue is a data structure (can be implemented as a heap) which supports the following operations:

insert(u, key):

Insert u with the key value key in Q.

u = extractMin():

Extract the item with the minimum key value in Q.

decreaseKey(u, new-key):

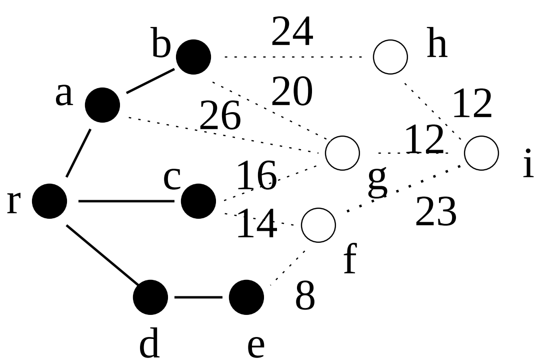
Decrease u's key value to new-key.

Remark: Priority Queues can be implemented so that each operation takes time $O(\log |Q|)$. See CLRS!

Using a Priority Queue to Find the Lightest Edge

Each item of the queue is a triple $(u, pred[u], key[u])$, where

- u is a vertex in $V \setminus S$,
- $key[u]$ is the weight of the lightest edge from u to any vertex in S, and
- $pred[u]$ is the endpoint of this edge in S. The array is used to build the MST tree.



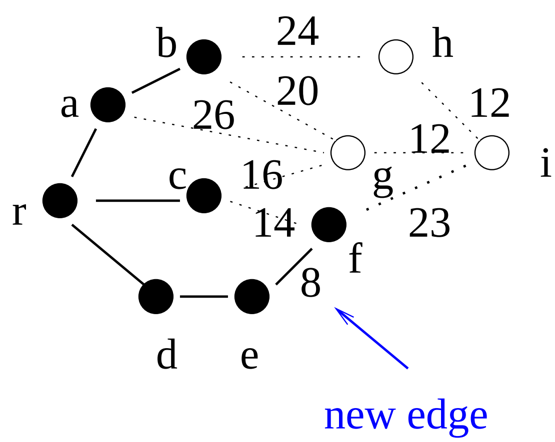
key[f] = 8, pred[f] = e

key[i] = infinity, pred[i] = nil

key[g] = 16, pred[g] = c

key[h] = 24, pred[h] = b

$\longrightarrow$ f has the minimum key



key[i] = 23, pred[i] = f

After adding the new edge and vertex f, update the key[v] and pred[v] for each vertex v adjacent to f

Description of Prim's Algorithm

Remark: G is given by adjacency lists. The vertices in $V \setminus S$ are stored in a priority queue with key=value of lightest edge to vertex in S.

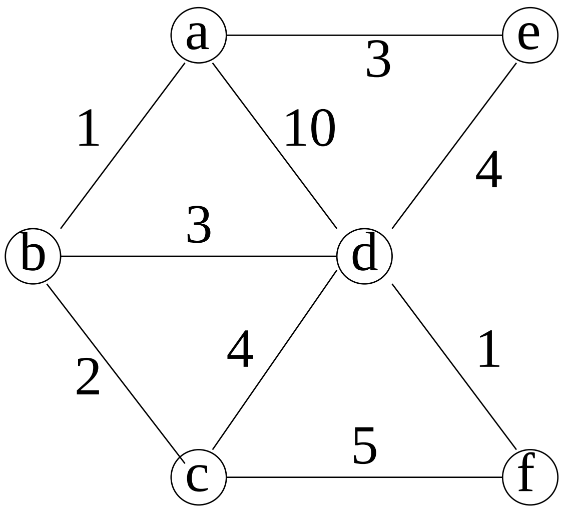
```
Prim(G, w, r)
{   for each u ∈ V                                 initialize
    {   key[u] = +∞;
        color[u] = W;
    }
    key[r] = 0;                                    start at root
    pred[r] = NIL;
    Q = new PriQueue(V);                           put vertices in Q
    while(Q is nonempty)                           until all vertices in MST
    {   u=Q.extraxtMin();                          lightest edge
        for each (v ∈ adj[u])
        {   if ((color[v] == W)&(w[u, v] < key[v]))
                key[v] = w[u, v];                  new lightest edge
                Q.decreaseKey(v, key[v]);
                pred[v] = u;
        }
        color[u] = B;
    }
}
```

When the algorithm terminates, $Q = \emptyset$ and the MST is

$$T = \{\{v, pred[v]\} : v \in V \setminus \{r\}\}.$$

The pred pointers define the MST as an inverted tree rooted at r.

Example for Running Prim's Algorithm



u	a	b	c	d	e	f
key[u]						
pred[u]						

Analysis of Prim's Algorithm

Let $n = |V|$ and $e = |E|$. The data structure PriQueue supports the following two operations: (See CLRS)

- $O(\log n)$ to extract each vertex from the queue.
  Done once for each vertex $= O(n \log n)$.

- $O(\log n)$ time to decrease the key value of neighboring vertex.
  Done at most once for each edge $= O(e \log n)$.

Total cost is then

$$O((n + e) \log n)$$

Analysis of Prim's Algorithm – Continued

```
Prim(G, w, r)  {
   for each (u in V)                                    2n
   {
      key[u] = +infinity;
      color[u] = white;
   }

   key[r] = 0;                                          1
   pred[r] = nil;                                       1
   Q = new PriQueue(V);                                 n

   while (Q. nonempty())                        1
   {
      u = Q.extractMin();                   O(log n)
      for each (v in adj[u])
      {
         if ((color[v] == white) &         1
              (w(u,v) < key[v])             1   O(deg(u) log n)
         {
            key[v] = w(u, v);              1
            Q.decreaseKey(v, key[v]);      O(log n)
            pred[v] = u;                   1
         }
      }
      color[u] = black;                          1
   }
}
```

$$\sum_{u \in V} [O(\log n) + O(deg(u) \log n)]$$

Analysis of Prim's Algorithm – Continued

So the overall running time is

$$
\begin{aligned}
& T(n,e) \\
&= 3n + 2 + \sum_{u \in V} [O(\log n) + O(\deg(u) \log n)] \\
&= 3n + 2 + O\left[(\log n) \sum_{u \in V} (1 + \deg(u))\right] \\
&= 3n + 2 + O[(\log n)(n + 2e)] \\
&= O[(\log n)(n + 2e)] \\
&= O[(\log n)(n + e)] \\
&= O[(|V| + |E|) \log |V|].
\end{aligned}
$$