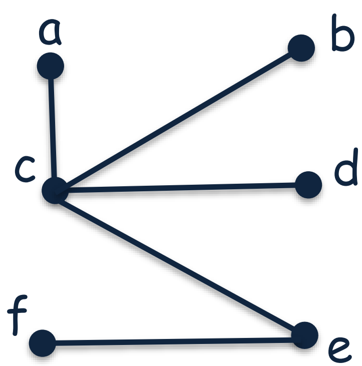
Trees

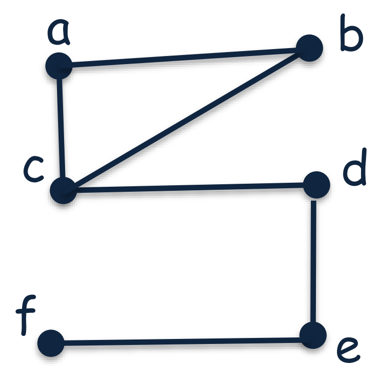
Murat Osmanoglu

Definition

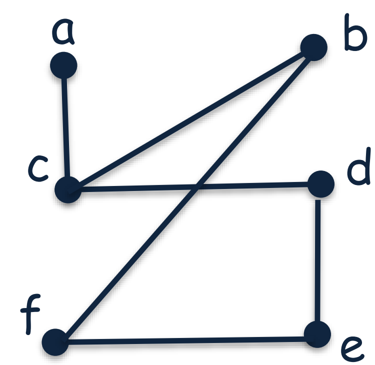
- A tree is a connected undirected graph with no simple circuit



G_1



G_2

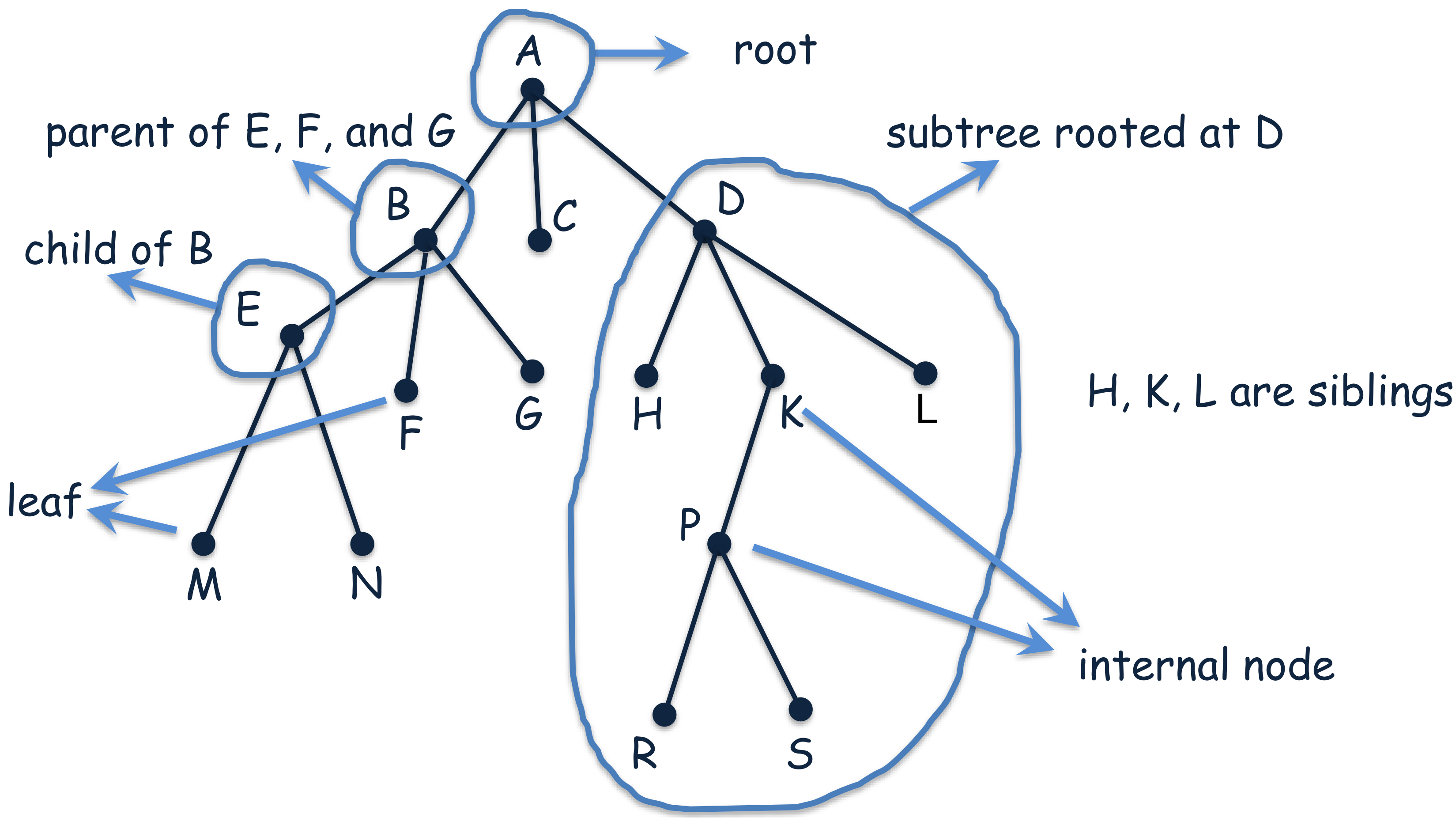


G_3

- An undirected graph is a treeif and only if there is a unique simple path between any two of its vertices

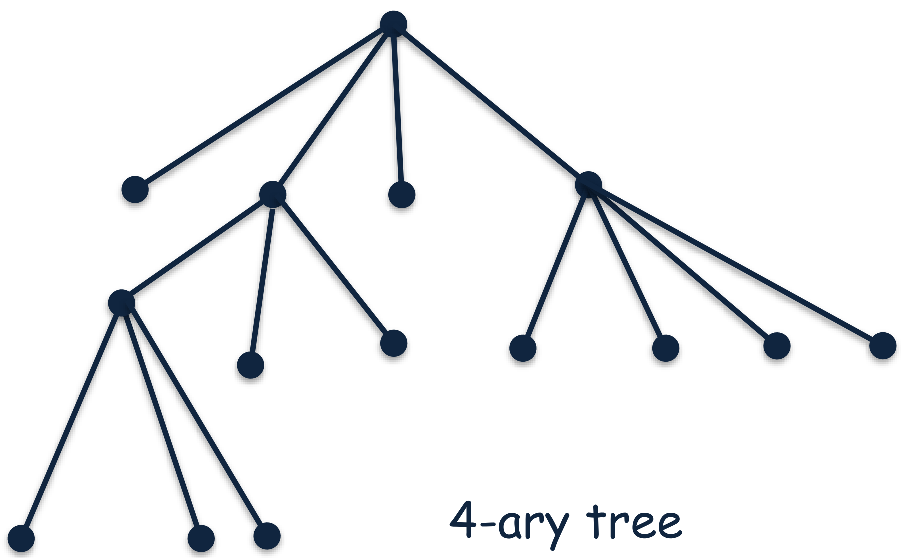
Definition

- A rooted tree is a tree in which one vertex is fixed as the root and every edge is directed away from the root

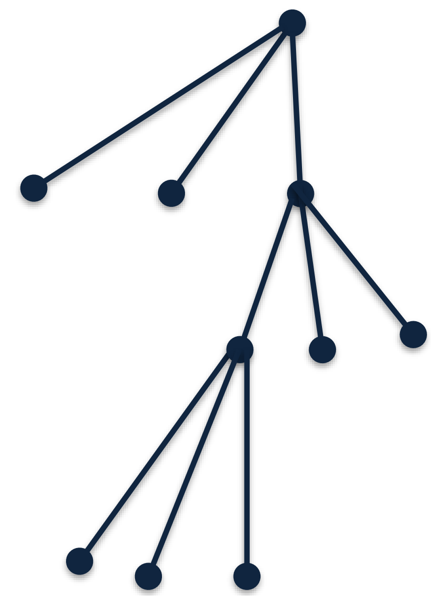


Definition

- A rooted tree is called m-ary tree if every internal vertex has no more than m children
- It's called a full m-ary tree if every internal vertex has exactly m children



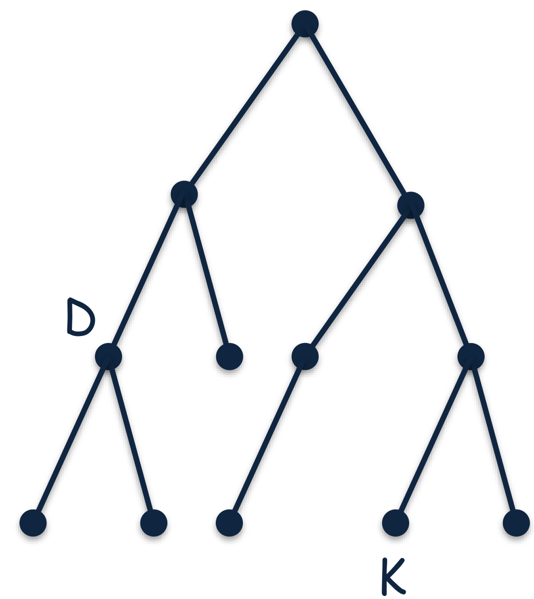
4-ary tree



full 3-ary tree

Definition

- A tree with n vertices has n – 1 edges
- A full m-ary tree with i internal vertices contains n = mi + 1 vertices
- The level (depth) of a vertex v in a rooted tree is the length of the unique path from the root to this vertex
- The height of a rooted tree is the length of the longest path from the root to any vertex
- A rooted m-ary tree of height h is balanced if all the leaves are at levels h or h – 1



level of D is 2

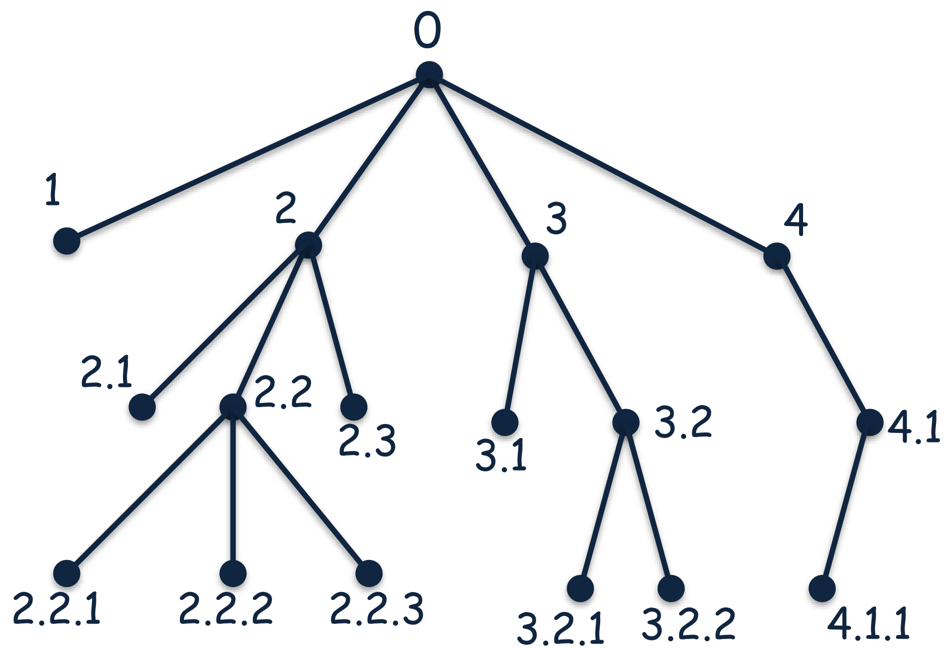
level of K is 3

height of the tree is 3

it's a balanced tree

Tree Traversal

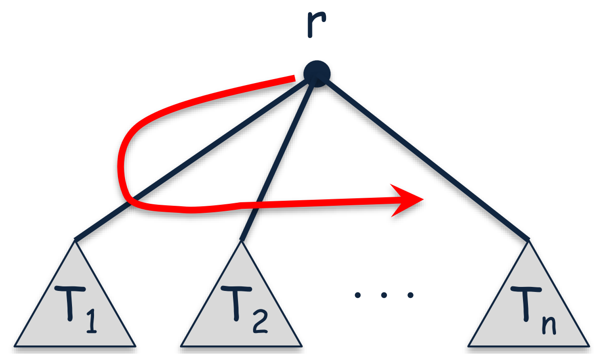
Universal Address Systems



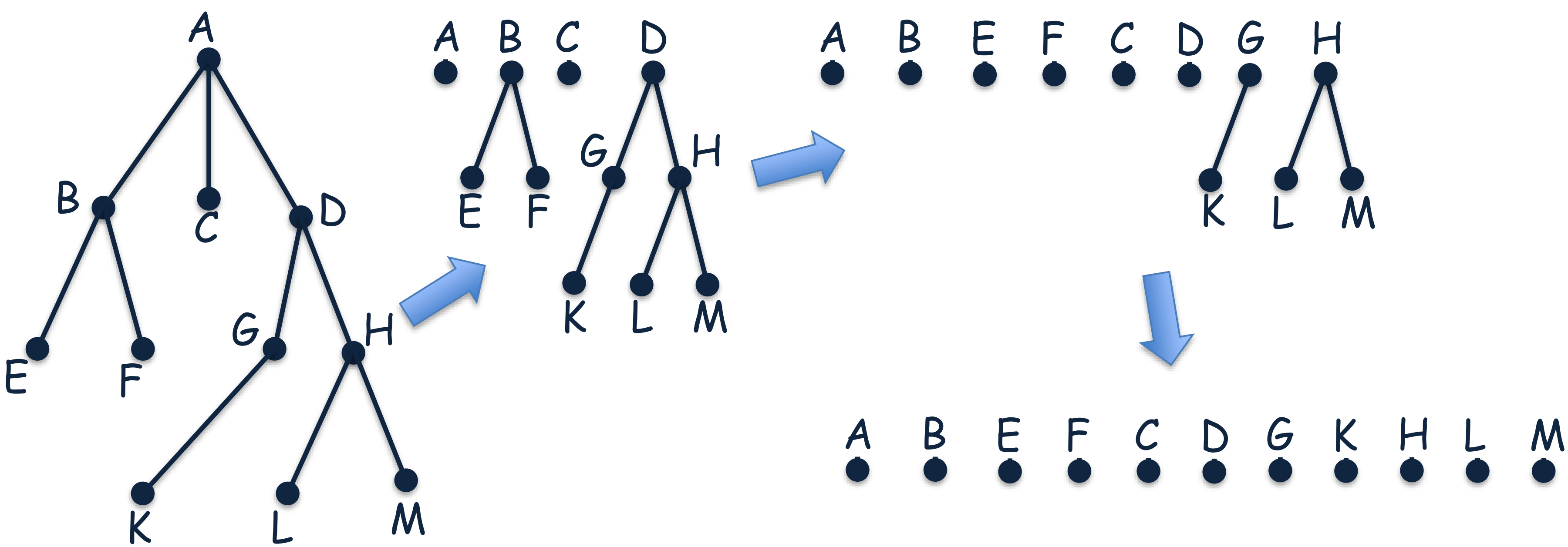
- Label the root with the integer 0, and its k children with 1, 2, . . . , k from left to right
- For each vertex v at some level with label A, label its t children with A.1, A.2, . . . , A.t from left to right

Tree Traversal

Preorder Traversal

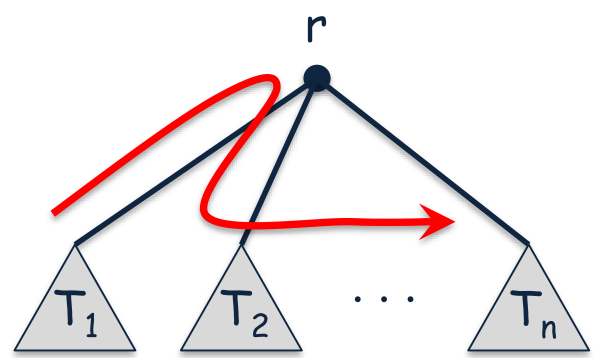


- If the tree consists of r only, visit r
- otherwise,
  - visit r
  - traverse subtrees from left to right

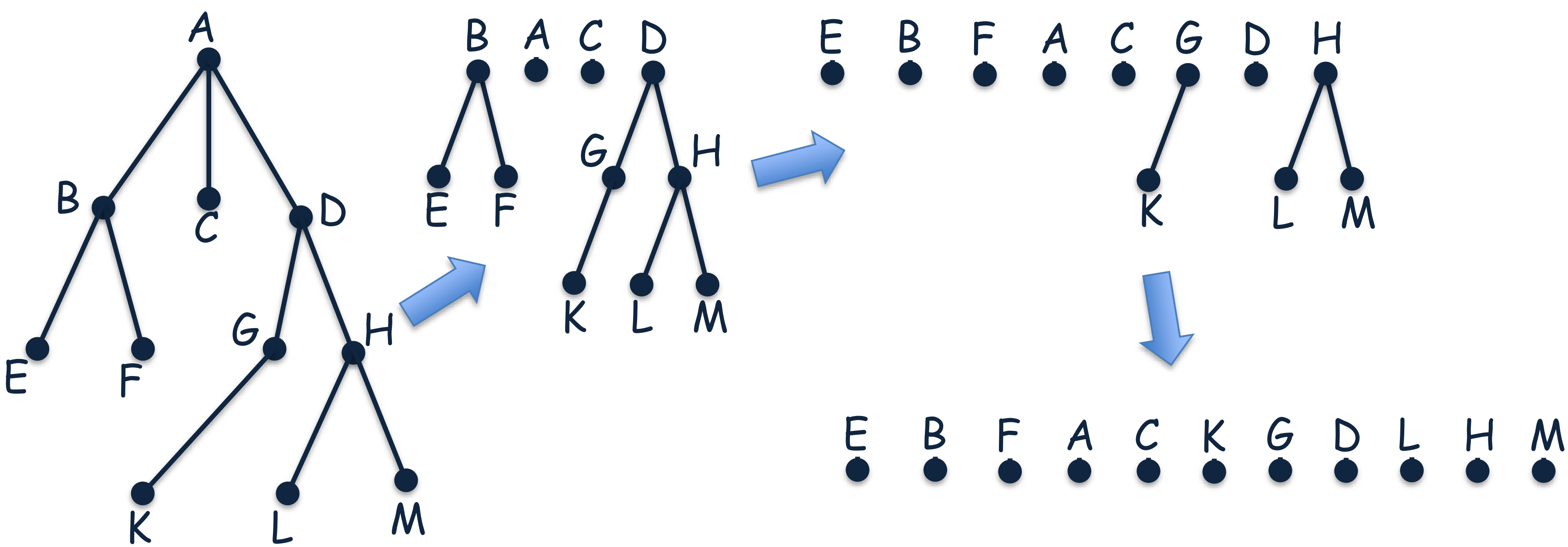


Tree Traversal

Inorder Traversal



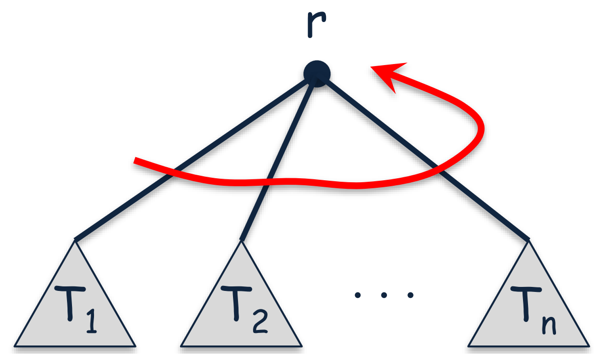
- If the tree consists of r only, visit r
- otherwise,
  - traverse leftmost subtree
  - visit r
  - traverse the remaining subtrees from left to right

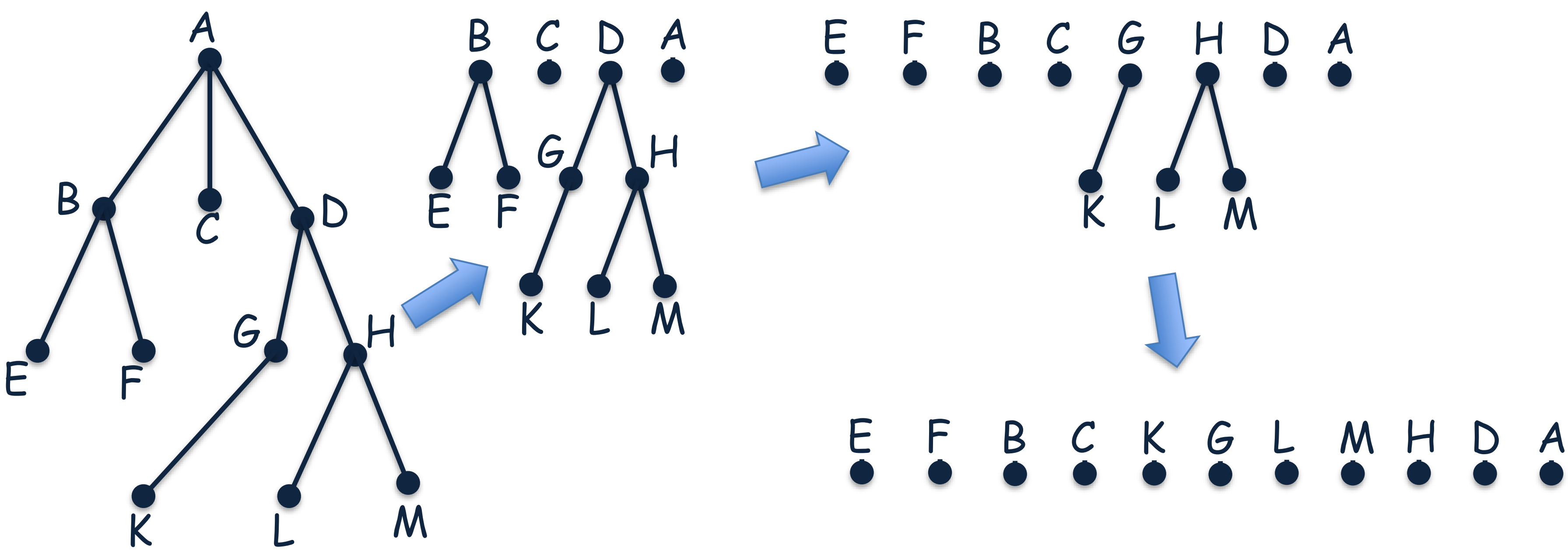


Tree Traversal

Postorder Traversal

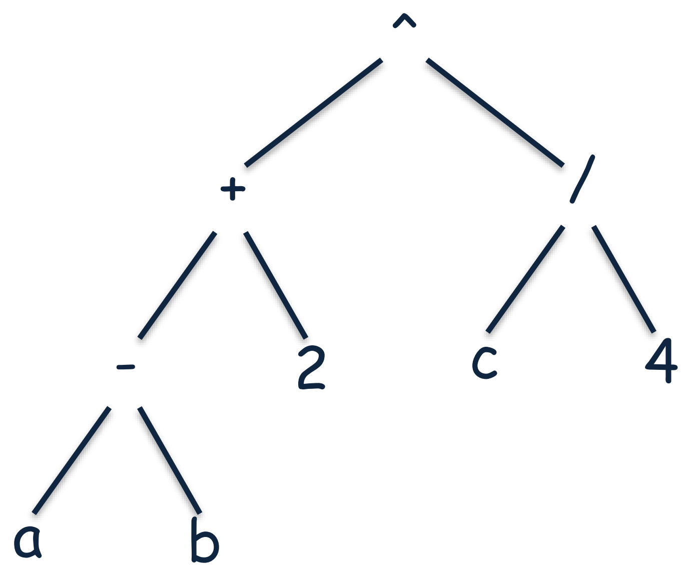
- If the tree consists of r only, visit r
- otherwise,
  - traverse subtrees from left to right
  - visit r





Tree Traversal

Infix-Prefix-Postfix Notation



- Inorder traversal of the tree

  a – b + 2 ^ c / 4

  infix form of the expression

  [(a – b) + 2] ^ (c / 4)

- Preorder traversal of the tree

  ^ + - a b 2 / c 4

  this is also prefix form

- Postorder traversal of the tree

  a b – 2 + c 4 / ^

  this is also postfix form

Tree Traversal

Infix-Prefix-Postfix Notation

+ - * 2 3 5 / ^ 2 3 4

+ - 6 5 / 8 4

+ 1 2

3

7 2 3 * - 4 ^ 9 3 / +

7 6 - 4 ^ 3 +

1 4 ^ 3 +

1 3 +

4

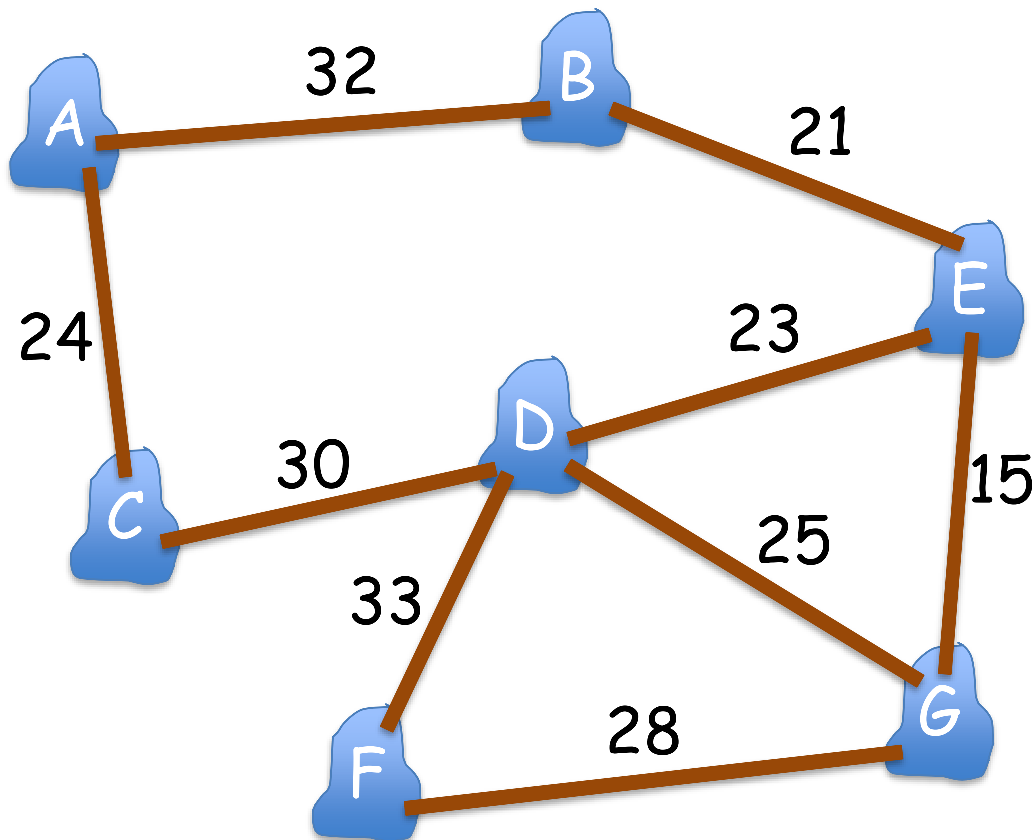
MST

- given a connected undirected graph $G = (V,E)$ with real-valued edge weight $w(e)$, an minimum spanning tree (MST) is a subset of the edges $T \subseteq E$ such that T is a tree that connects all vertices of the graphs, and T has minimum total weight.

$$w(T) = \Sigma_{e \text{ in } T}\, w(e)$$

- connecting all computers in an office buildings using least amount of cables

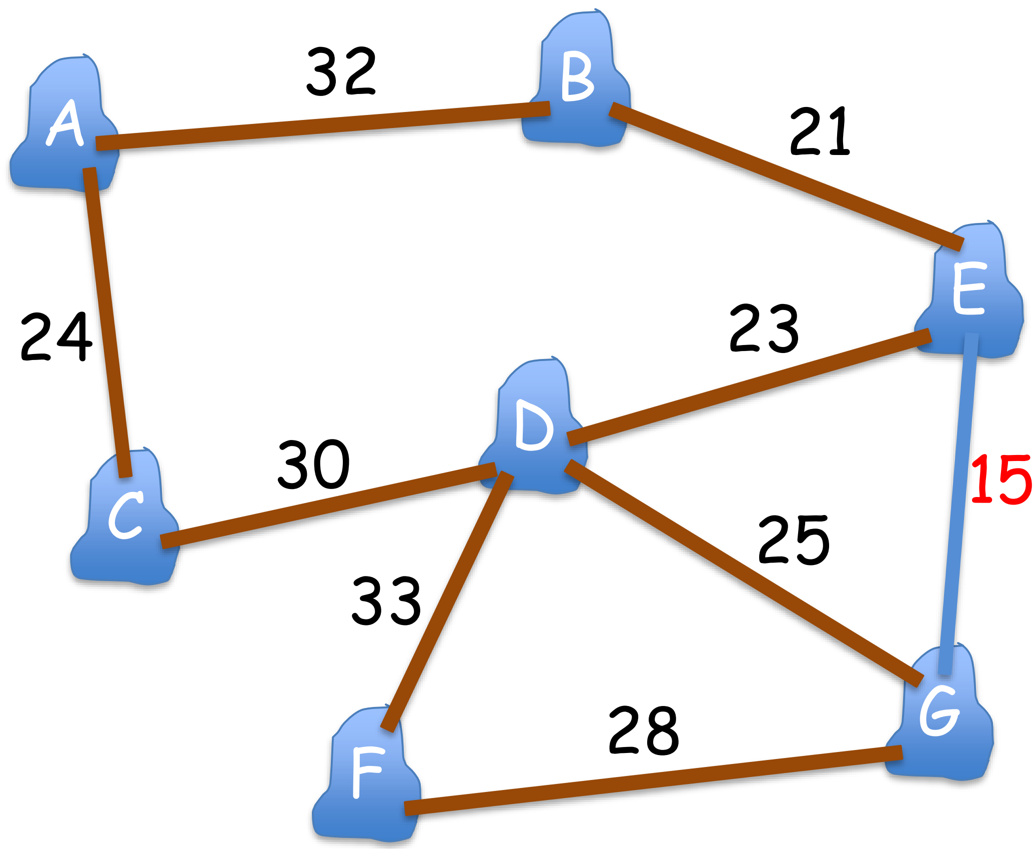
MST



n-1 edges enough to connect n nodes, one more creates a cycle

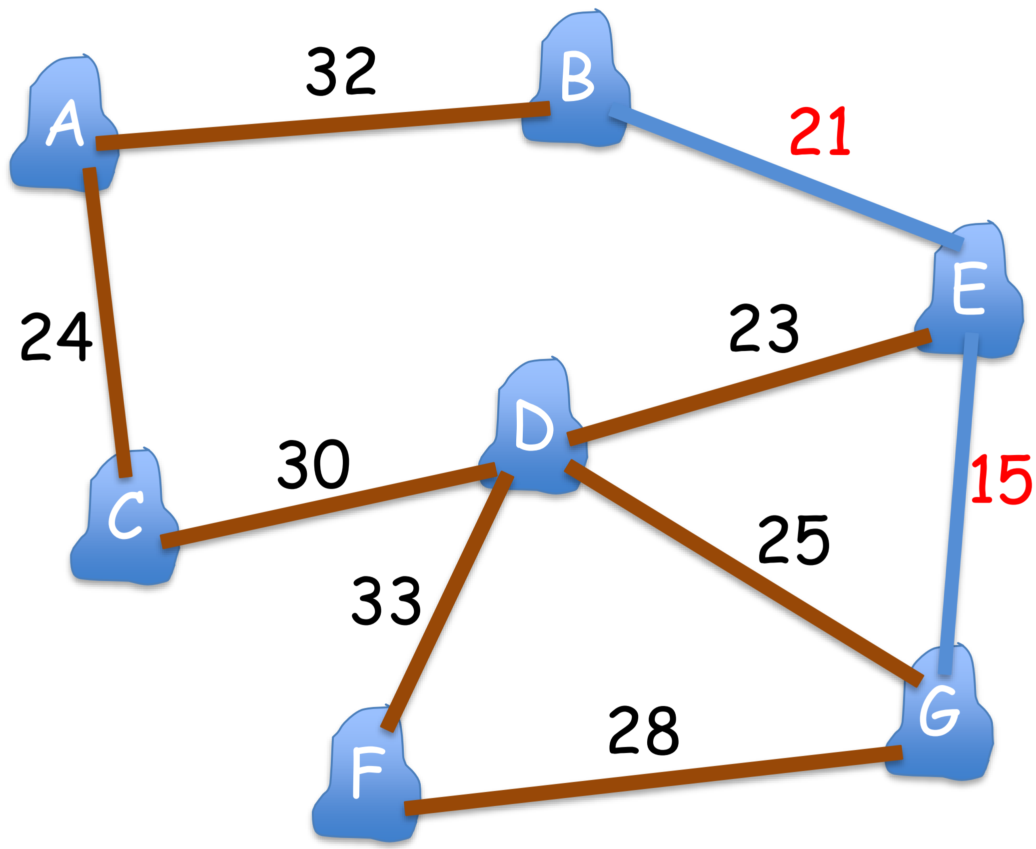
- start with an empty set of edges A
- repeat the following procedure |V| – 1 times :
  add the lightest edge that does not create a cycle to A

MST



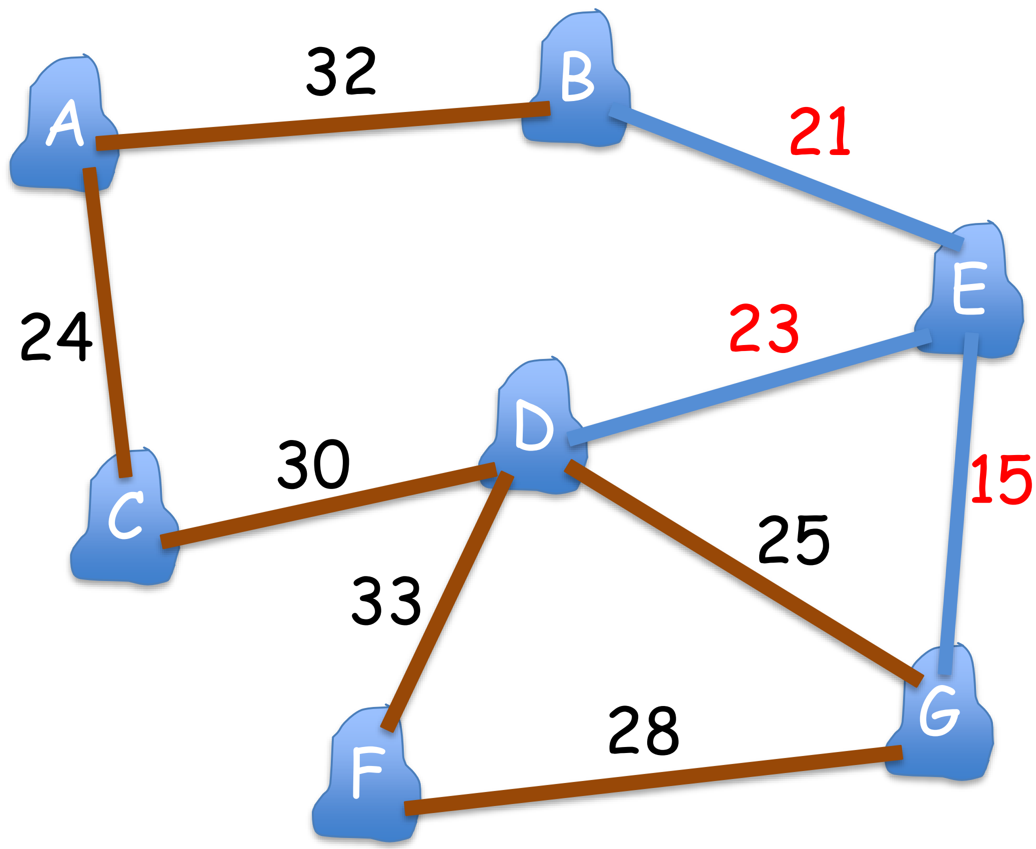
- start with an empty set of edges A
- repeat the following procedure |V| – 1 times :
  add the lightest edge that does not create a cycle to A

MST



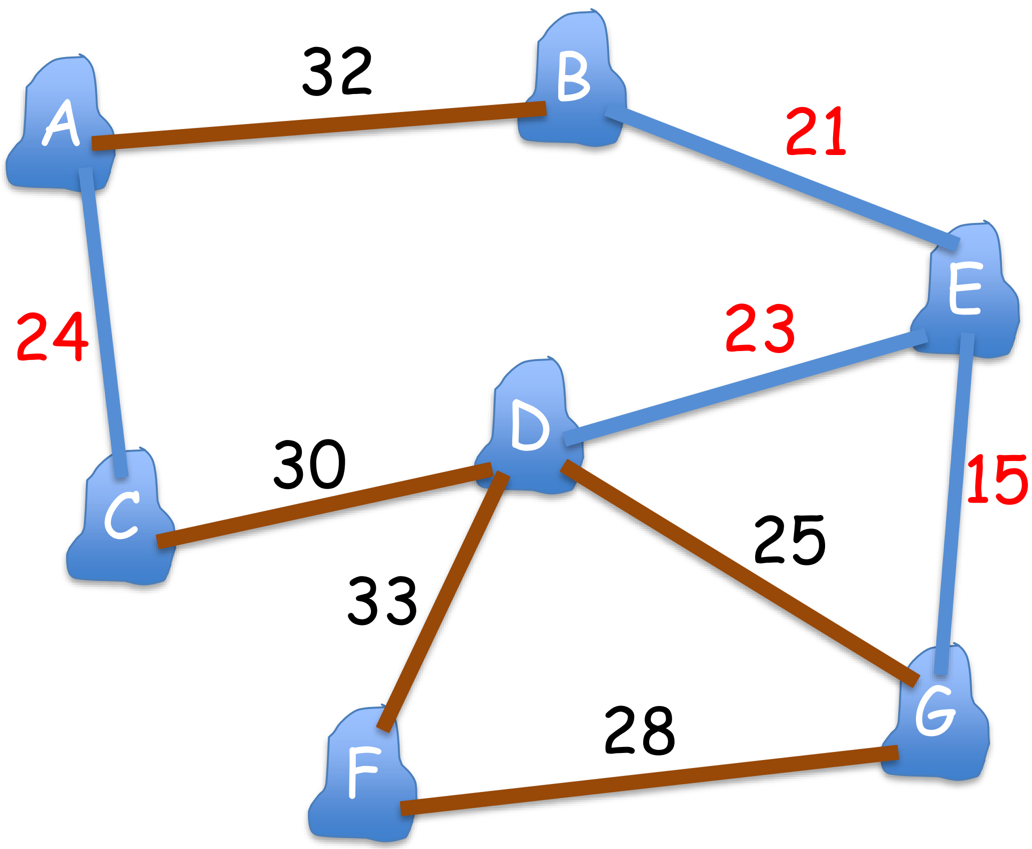
- start with an empty set of edges A
- repeat the following procedure |V| – 1 times :
  add the lightest edge that does not create a cycle to A

MST



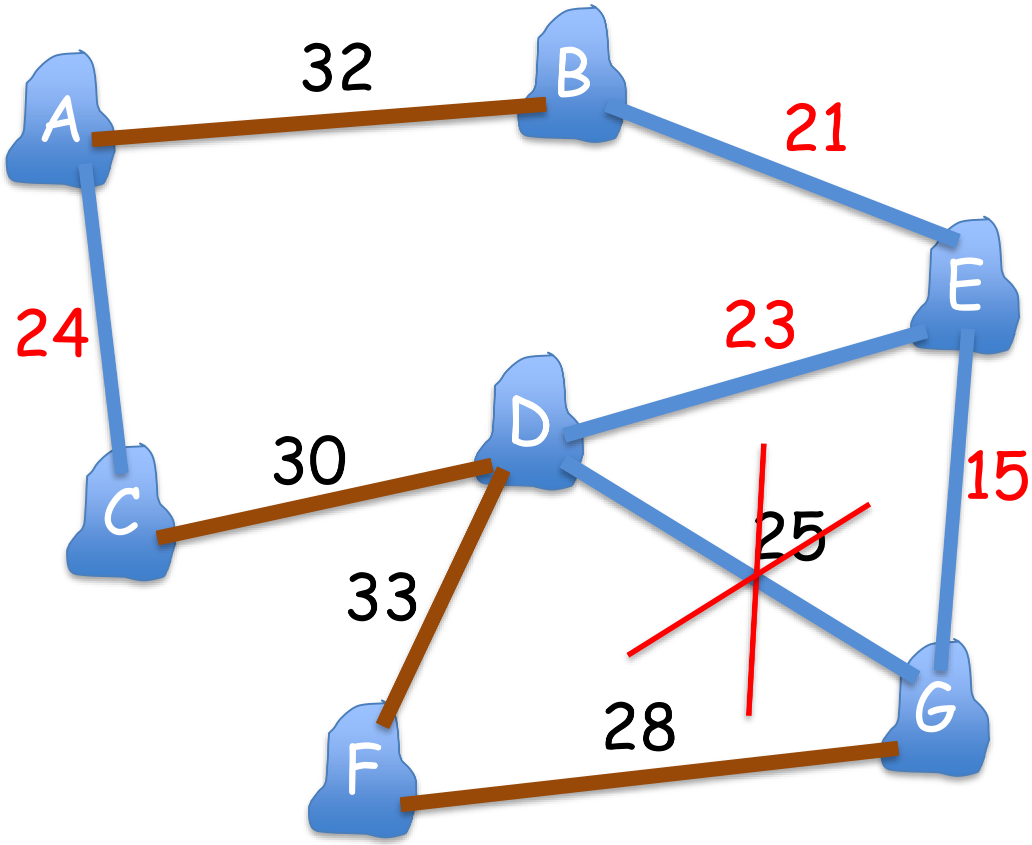
- start with an empty set of edges A
- repeat the following procedure |V| – 1 times :
  add the lightest edge that does not create a cycle to A

MST



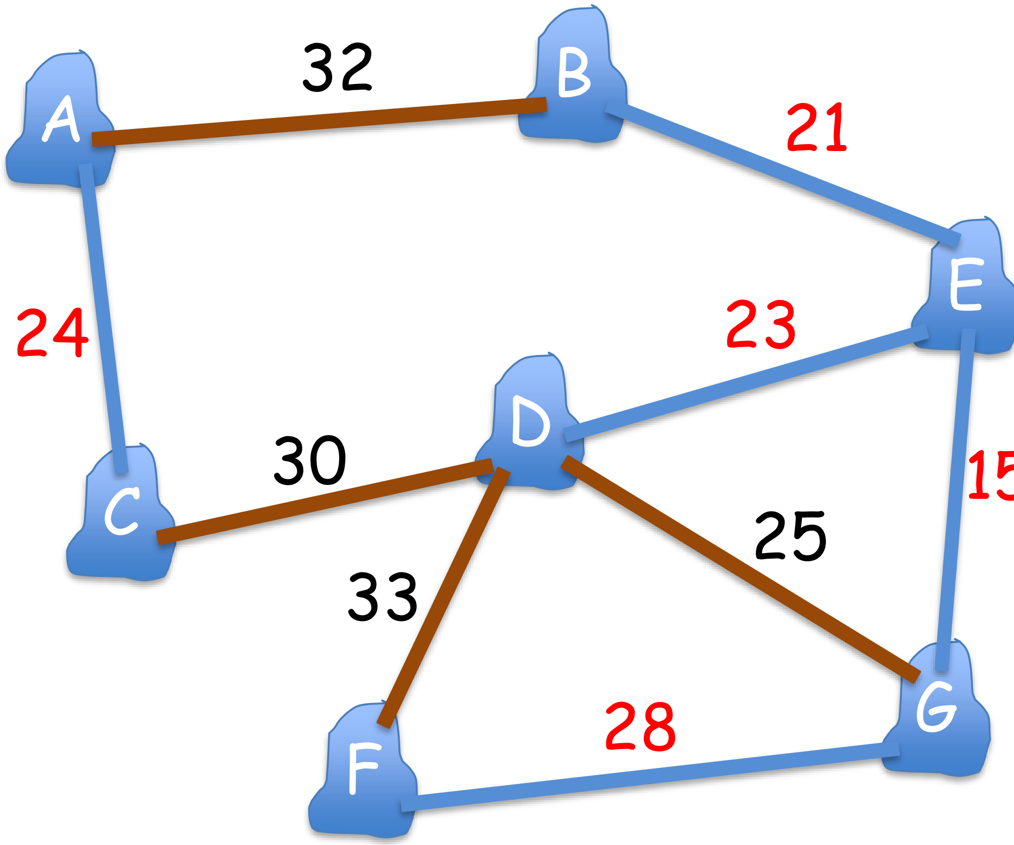
- start with an empty set of edges A
- repeat the following procedure |V| – 1 times :
  add the lightest edge that does not create a cycle to A

MST



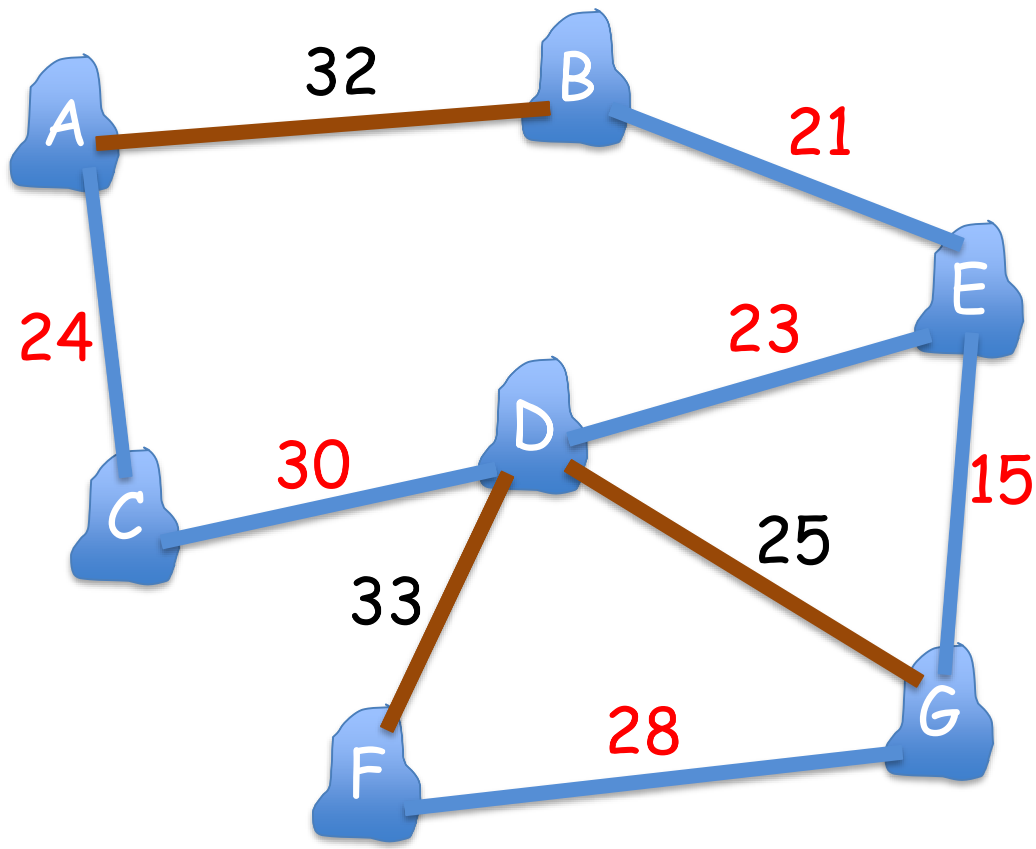
- start with an empty set of edges A
- repeat the following procedure |V| – 1 times :
  add the lightest edge that does not create a cycle to A

MST



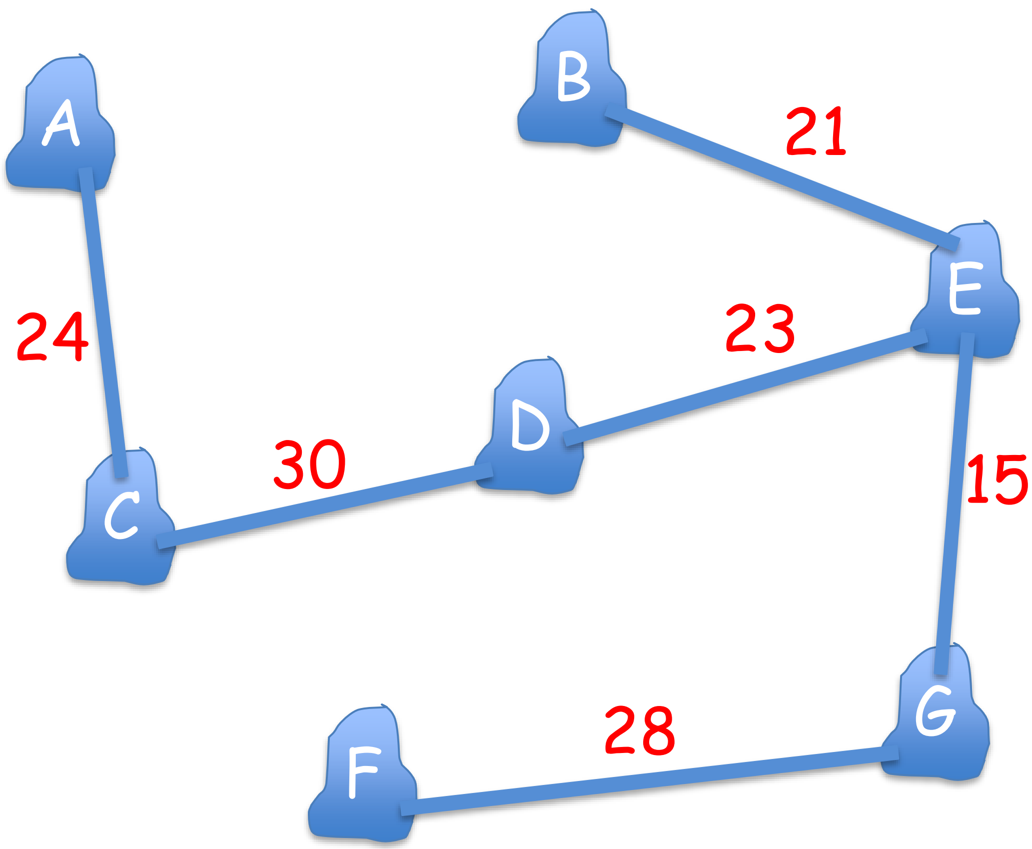
- start with an empty set of edges A
- repeat the following procedure |V| – 1 times :
  add the lightest edge that does not create a cycle to A

MST



- start with an empty set of edges A
- repeat the following procedure |V| – 1 times :
  add the lightest edge that does not create a cycle to A

MST



- start with an empty set of edges A
- repeat the following procedure |V| – 1 times :
  add the lightest edge that does not create a cycle to A

Prim's Algorithm

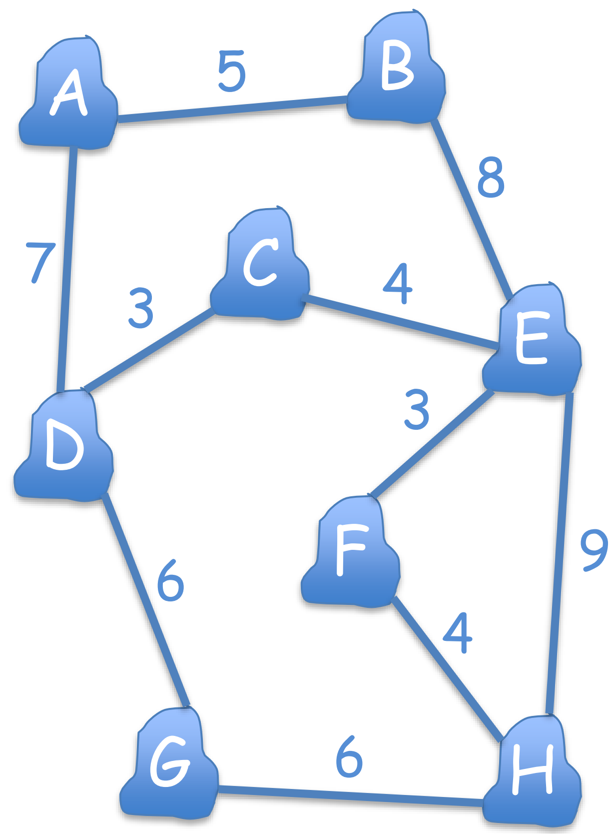
<u>MST-Prim(G,s)</u>

```
for each u of V
    u.key = ∞                                O(|V|)
    u.par = nil
s.key = 0
create a minimum priority Q on V             O(|V|)
while Q ≠ { }
    u = ExtractMin(Q)                        O(|V|.log|V|)
    for each v of Adj(u)
        if v in Q and w(u,v) < v.key
            v.par = u                        O(|E|.log|V|)
            v.key = w(u,v)
```

Prim's Algorithm

MST-Prim(G,s)

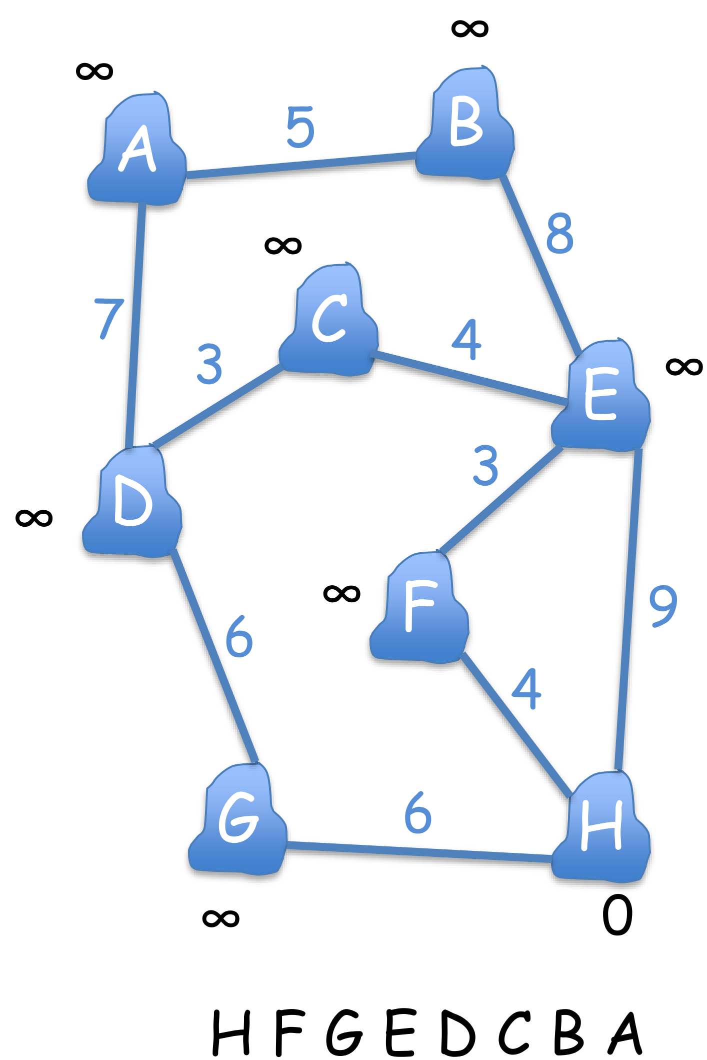
```
for each u of V
      u.key = ∞
      u.par = nil
s.key = 0
create a minimum priority Q on V
while Q ≠ { }
      u = ExtractMin(Q)
      for each v of Adj(u)
            if v in Q and w(u,v) < v.key
                  v.par = u
                  v.key = w(u,v)
```



Prim's Algorithm

MST-Prim(G,s)

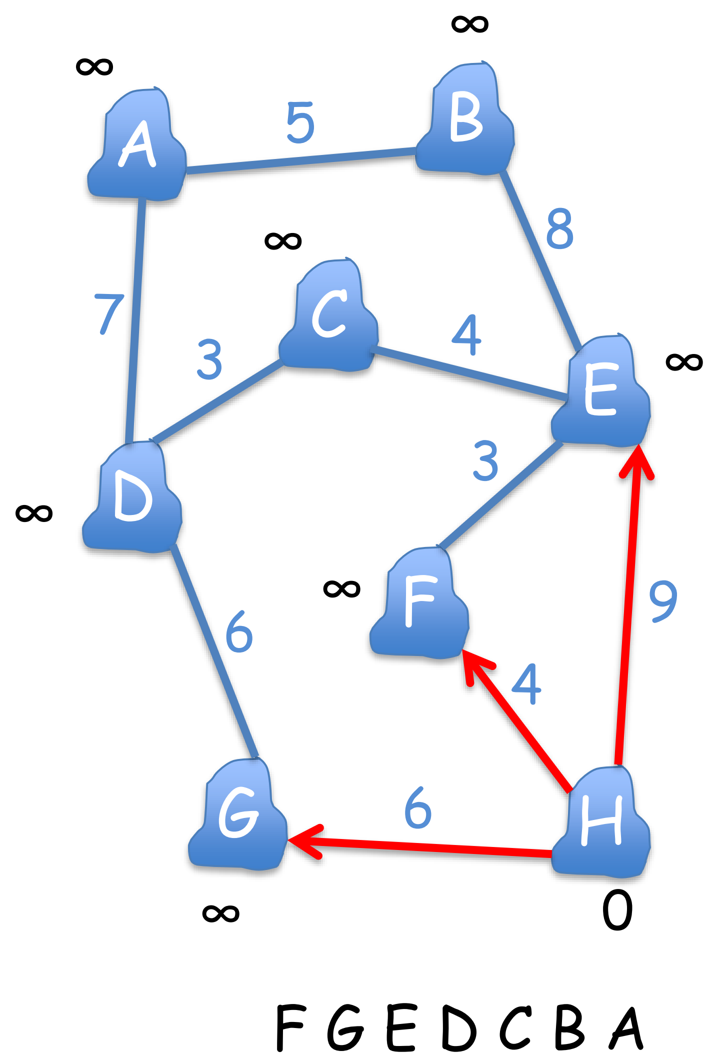
```
for each u of V
      u.key = ∞
      u.par = nil
s.key = 0
create a minimum priority Q on V
while Q ≠ { }
      u = ExtractMin(Q)
      for each v of Adj(u)
            if v in Q and w(u,v) < v.key
                  v.par = u
                  v.key = w(u,v)
```



Prim's Algorithm

MST-Prim(G,s)

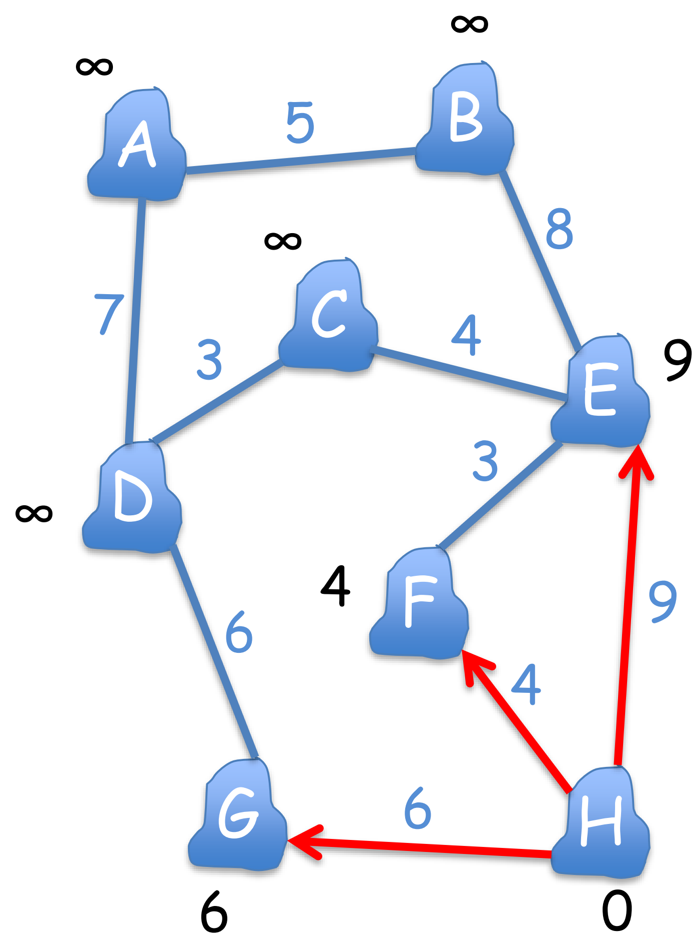
```
for each u of V
      u.key = ∞
      u.par = nil
s.key = 0
create a minimum priority Q on V
while Q ≠ { }
      u = ExtractMin(Q)
      for each v of Adj(u)
            if v in Q and w(u,v) < v.key
                  v.par = u
                  v.key = w(u,v)
```



Prim's Algorithm

MST-Prim(G,s)

```
for each u of V
      u.key = ∞
      u.par = nil
s.key = 0
create a minimum priority Q on V
while Q ≠ { }
      u = ExtractMin(Q)
      for each v of Adj(u)
            if v in Q and w(u,v) < v.key
                  v.par = u
                  v.key = w(u,v)
```

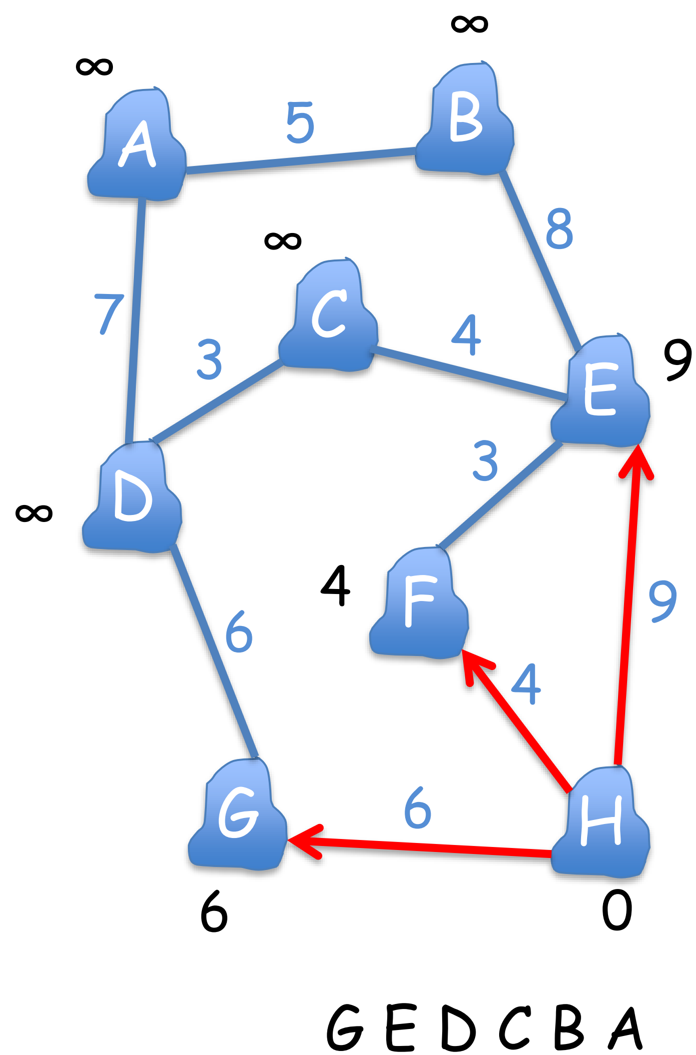


F G E D C B A

Prim's Algorithm

MST-Prim(G,s)

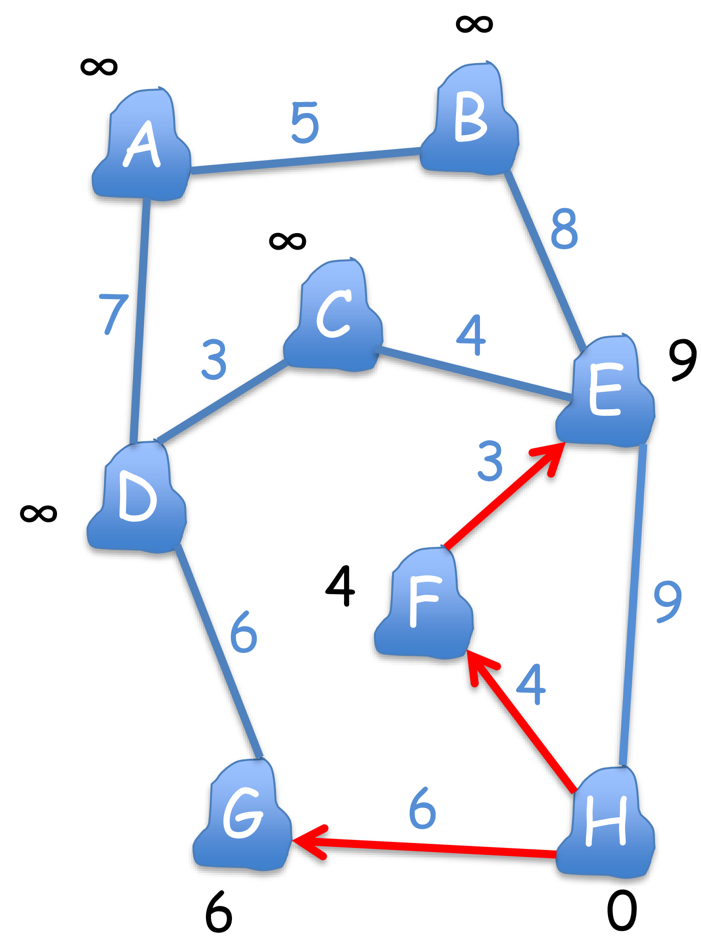
```
for each u of V
      u.key = ∞
      u.par = nil
s.key = 0
create a minimum priority Q on V
while Q ≠ { }
      u = ExtractMin(Q)
      for each v of Adj(u)
            if v in Q and w(u,v) < v.key
                  v.par = u
                  v.key = w(u,v)
```



Prim's Algorithm

MST-Prim(G,s)

```
for each u of V
      u.key = ∞
      u.par = nil
s.key = 0
create a minimum priority Q on V
while Q ≠ { }
      u = ExtractMin(Q)
      for each v of Adj(u)
            if v in Q and w(u,v) < v.key
                  v.par = u
                  v.key = w(u,v)
```

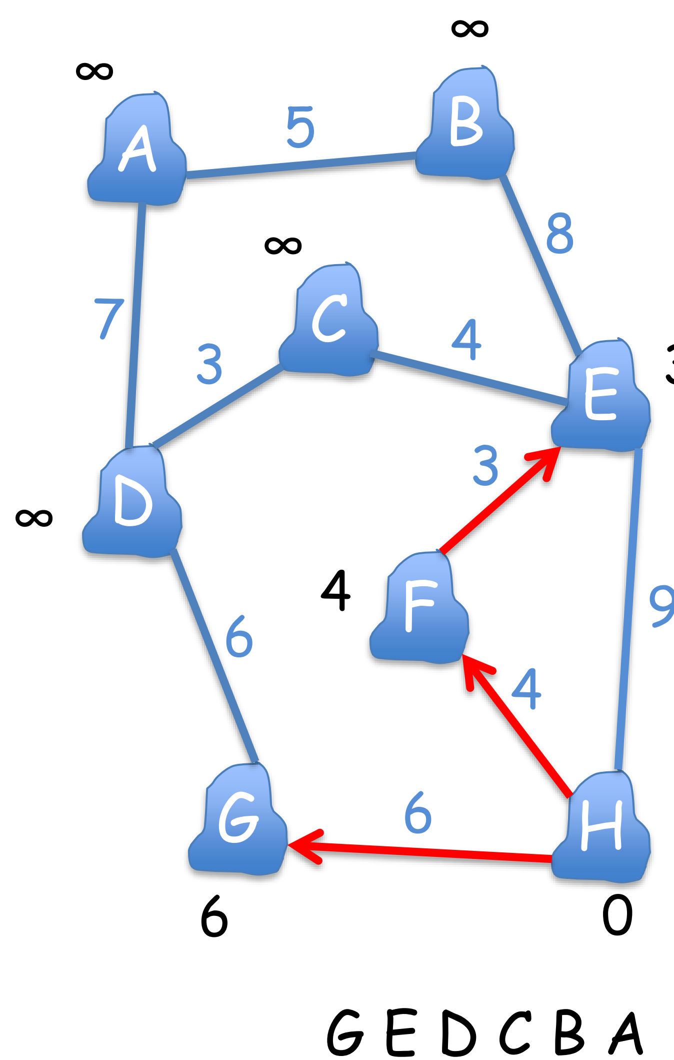


G E D C B A

Prim's Algorithm

MST-Prim(G,s)

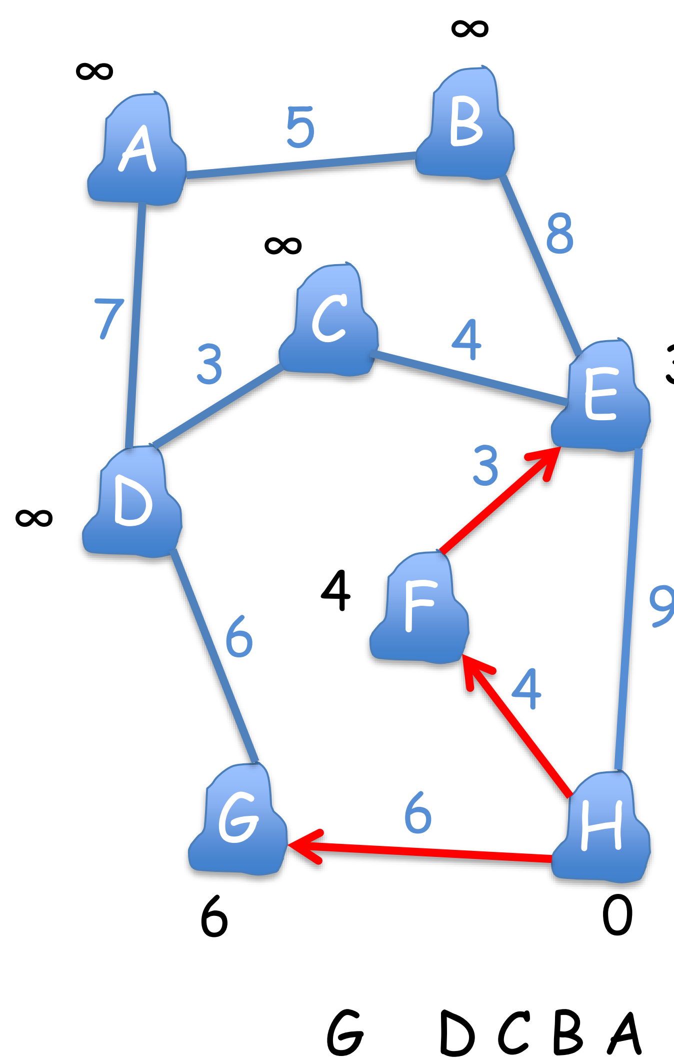
```
for each u of V
      u.key = ∞
      u.par = nil
s.key = 0
create a minimum priority Q on V
while Q ≠ { }
      u = ExtractMin(Q)
      for each v of Adj(u)
            if v in Q and w(u,v) < v.key
                  v.par = u
                  v.key = w(u,v)
```



Prim's Algorithm

<u>MST-Prim(G,s)</u>

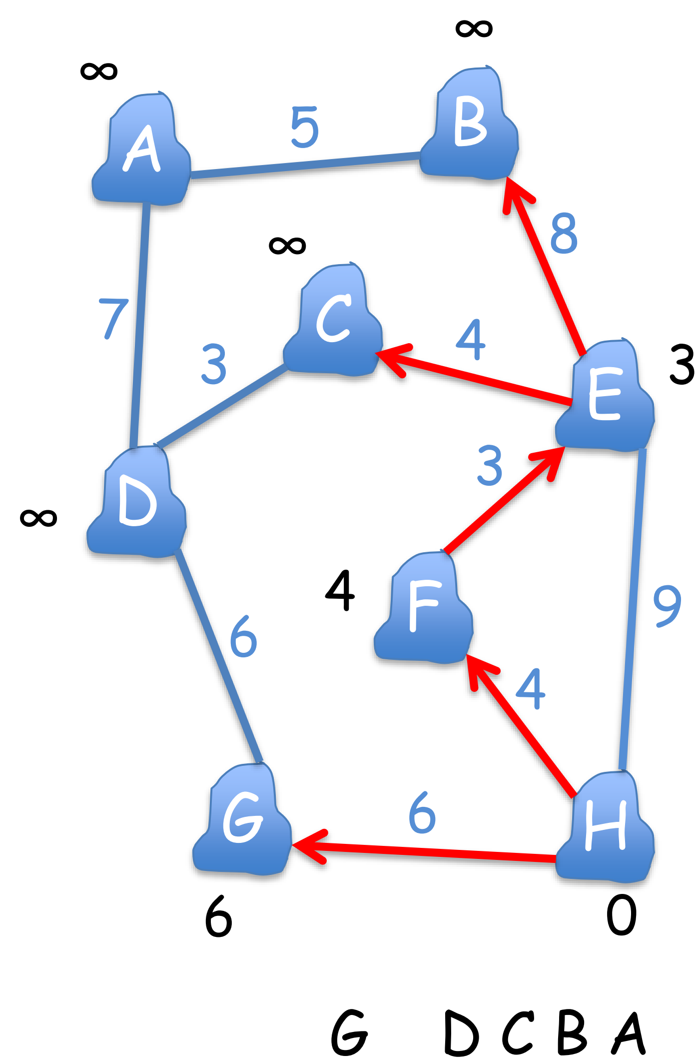
```
for each u of V
      u.key = ∞
      u.par = nil
s.key = 0
create a minimum priority Q on V
while Q ≠ { }
      u = ExtractMin(Q)
      for each v of Adj(u)
            if v in Q and w(u,v) < v.key
                  v.par = u
                  v.key = w(u,v)
```



Prim's Algorithm

MST-Prim(G,s)

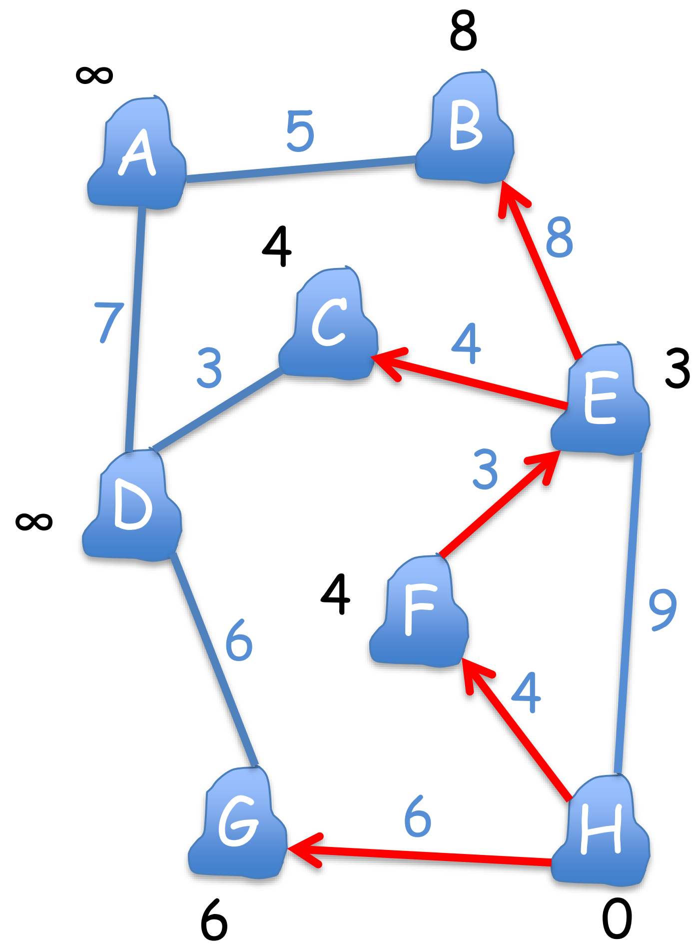
```
for each u of V
    u.key = ∞
    u.par = nil
s.key = 0
create a minimum priority Q on V
while Q ≠ { }
    u = ExtractMin(Q)
    for each v of Adj(u)
        if v in Q and w(u,v) < v.key
            v.par = u
            v.key = w(u,v)
```



Prim's Algorithm

MST-Prim(G,s)

```
for each u of V
      u.key = ∞
      u.par = nil
s.key = 0
create a minimum priority Q on V
while Q ≠ { }
      u = ExtractMin(Q)
      for each v of Adj(u)
            if v in Q and w(u,v) < v.key
                  v.par = u
                  v.key = w(u,v)
```

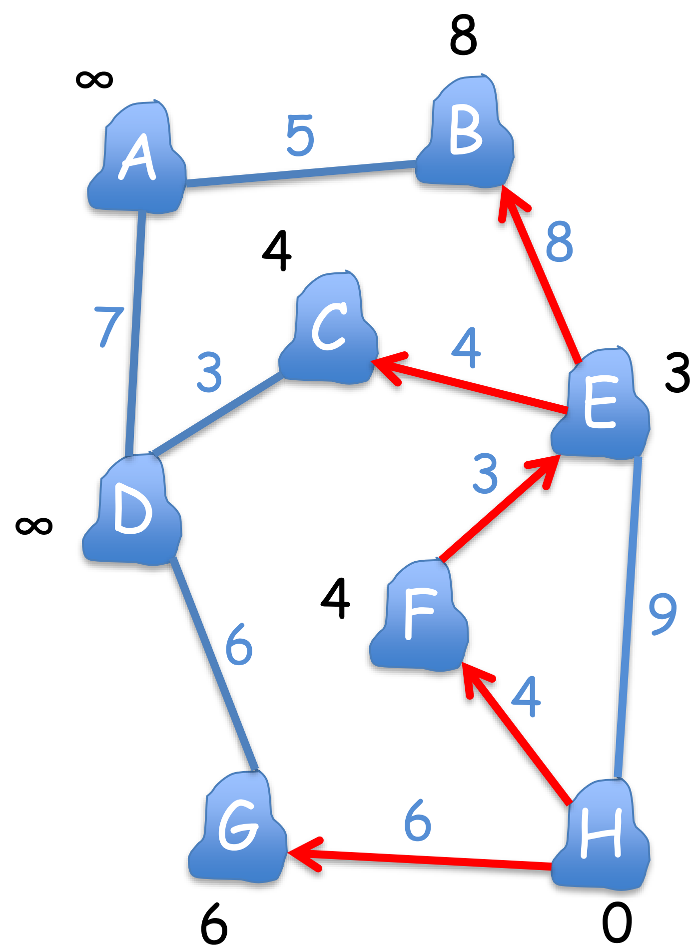


G D C B A

Prim's Algorithm

MST-Prim(G,s)

```
for each u of V
      u.key = ∞
      u.par = nil
s.key = 0
create a minimum priority Q on V
while Q ≠ { }
      u = ExtractMin(Q)
      for each v of Adj(u)
            if v in Q and w(u,v) < v.key
                  v.par = u
                  v.key = w(u,v)
```

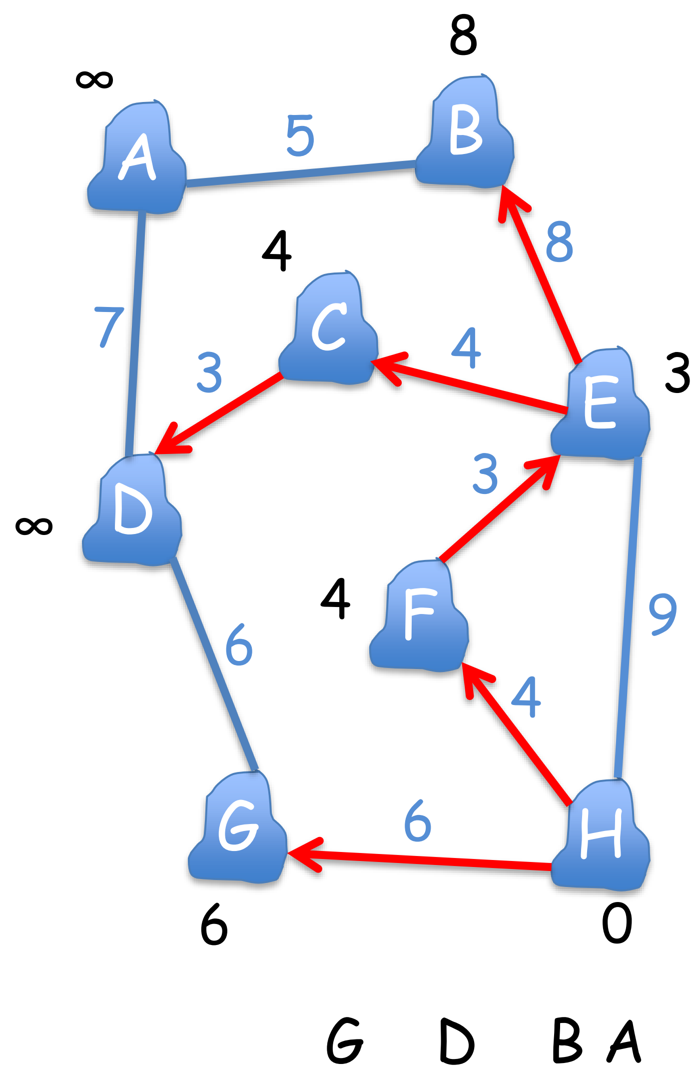


G D B A

Prim's Algorithm

MST-Prim(G,s)

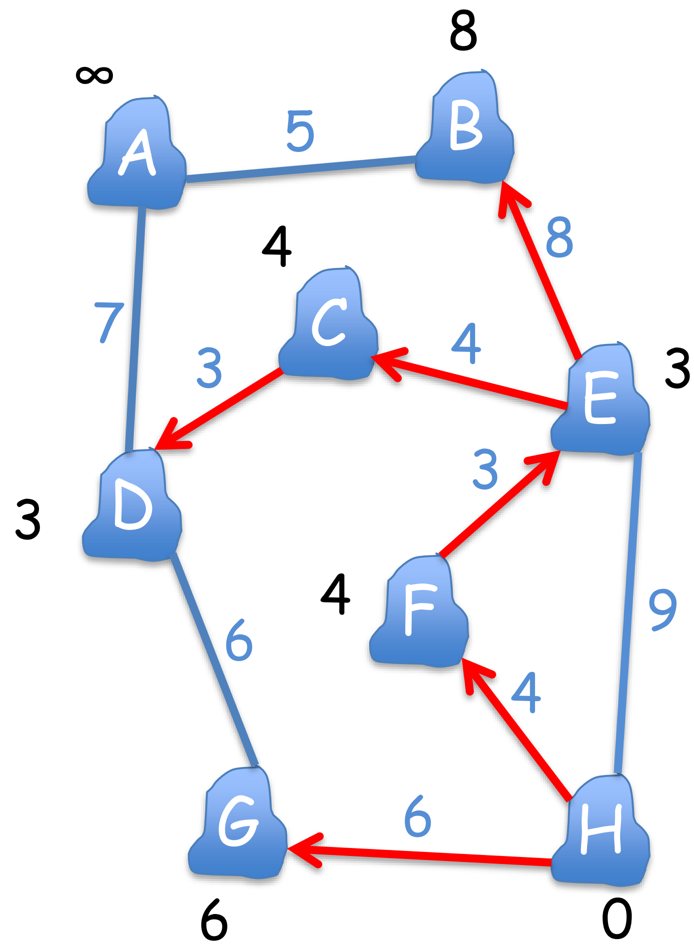
```
for each u of V
      u.key = ∞
      u.par = nil
s.key = 0
create a minimum priority Q on V
while Q ≠ { }
      u = ExtractMin(Q)
      for each v of Adj(u)
            if v in Q and w(u,v) < v.key
                  v.par = u
                  v.key = w(u,v)
```



Prim's Algorithm

MST-Prim(G,s)

```
for each u of V
    u.key = ∞
    u.par = nil
s.key = 0
create a minimum priority Q on V
while Q ≠ { }
    u = ExtractMin(Q)
    for each v of Adj(u)
        if v in Q and w(u,v) < v.key
            v.par = u
            v.key = w(u,v)
```

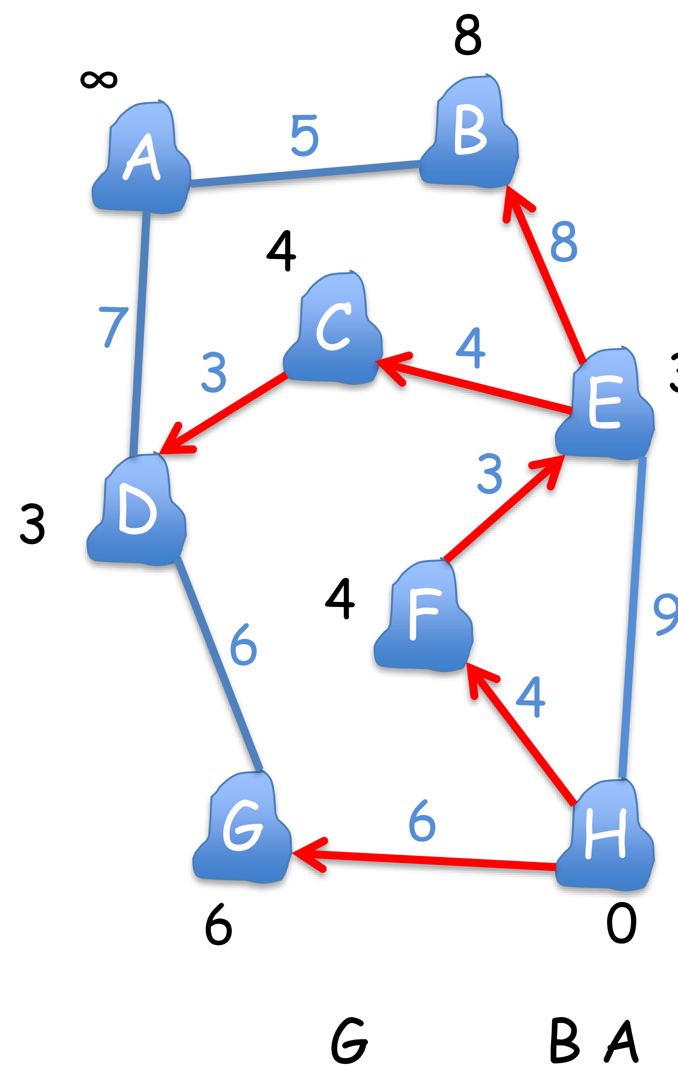


G D B A

Prim's Algorithm

MST-Prim(G,s)

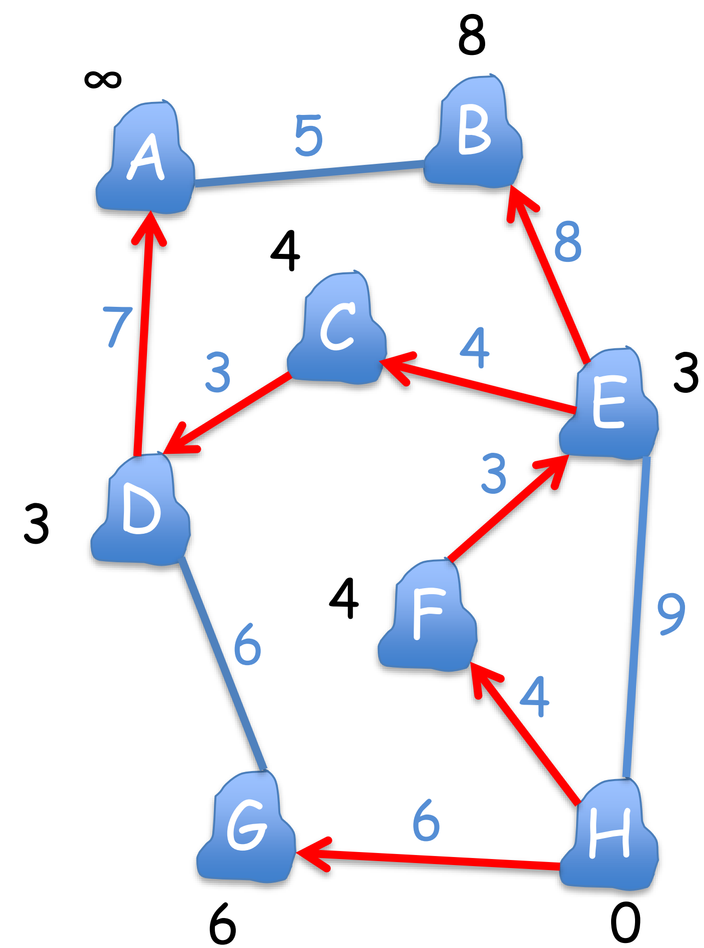
```
for each u of V
      u.key = ∞
      u.par = nil
s.key = 0
create a minimum priority Q on V
while Q ≠ { }
      u = ExtractMin(Q)
      for each v of Adj(u)
            if v in Q and w(u,v) < v.key
                  v.par = u
                  v.key = w(u,v)
```



Prim's Algorithm

<u>MST-Prim(G,s)</u>

```
for each u of V
    u.key = ∞
    u.par = nil
s.key = 0
create a minimum priority Q on V
while Q ≠ { }
    u = ExtractMin(Q)
    for each v of Adj(u)
        if v in Q and w(u,v) < v.key
            v.par = u
            v.key = w(u,v)
```

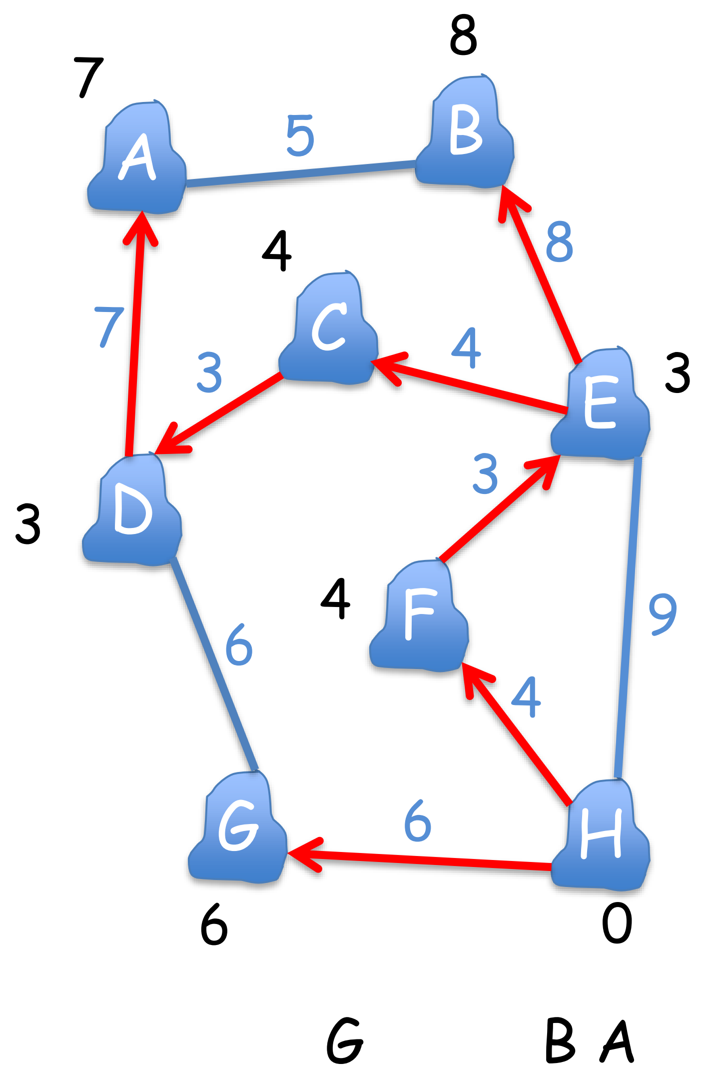


G B A

Prim's Algorithm

MST-Prim(G,s)

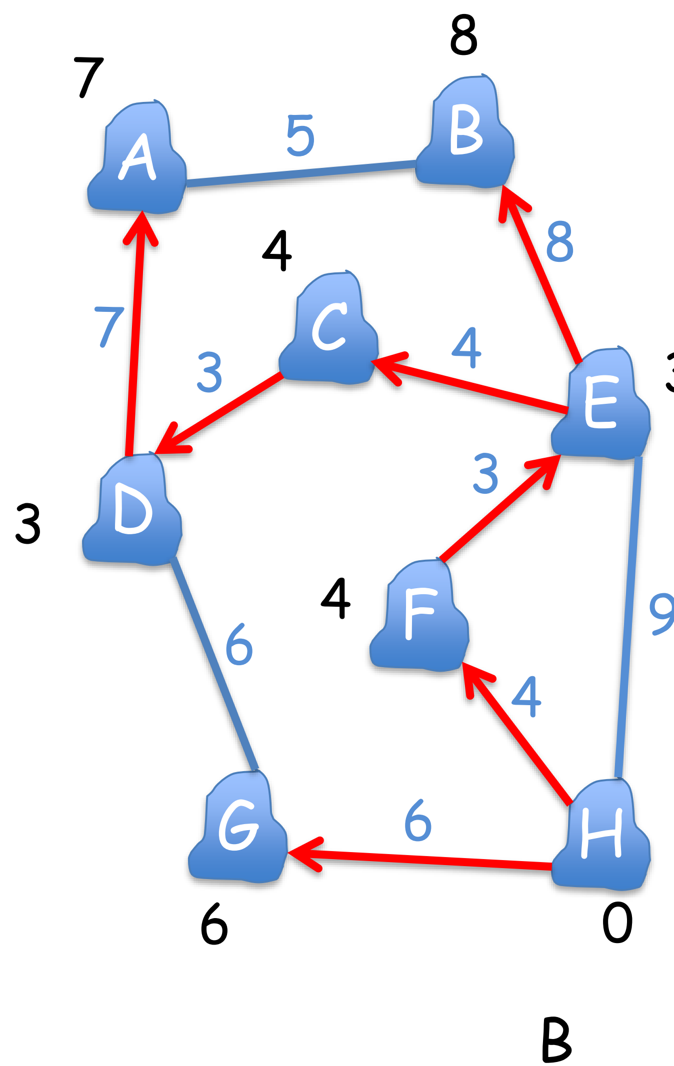
```
for each u of V
    u.key = ∞
    u.par = nil
s.key = 0
create a minimum priority Q on V
while Q ≠ { }
    u = ExtractMin(Q)
    for each v of Adj(u)
        if v in Q and w(u,v) < v.key
            v.par = u
            v.key = w(u,v)
```



Prim's Algorithm

MST-Prim(G,s)

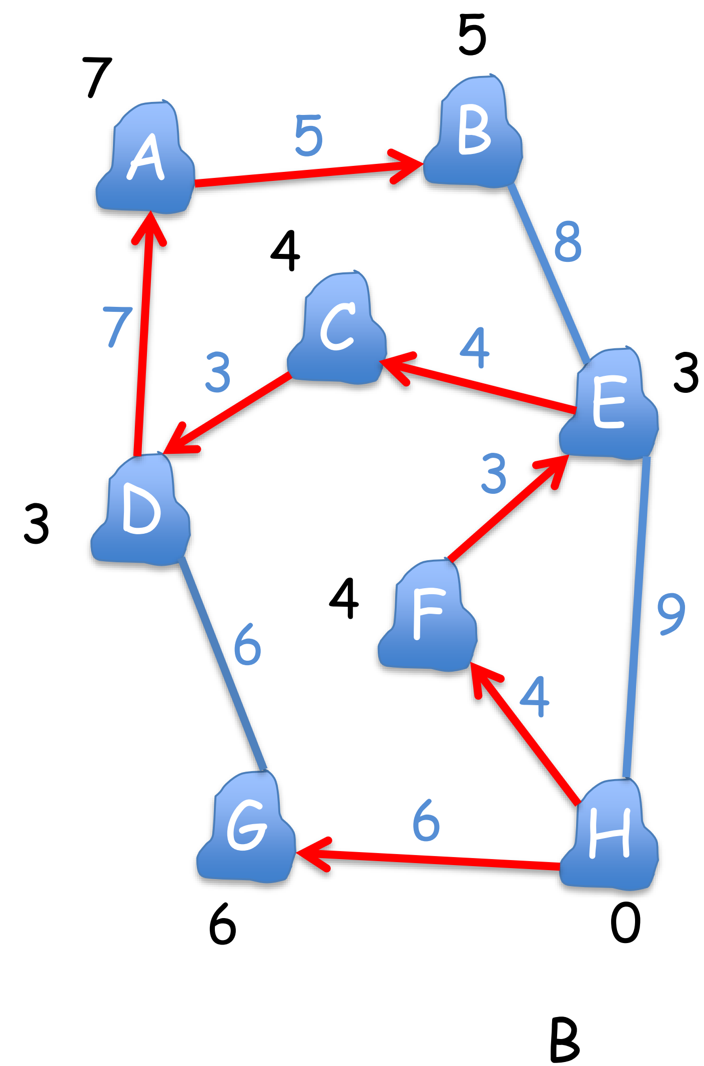
```
for each u of V
      u.key = ∞
      u.par = nil
s.key = 0
create a minimum priority Q on V
while Q ≠ { }
      u = ExtractMin(Q)
      for each v of Adj(u)
            if v in Q and w(u,v) < v.key
                  v.par = u
                  v.key = w(u,v)
```



Prim's Algorithm

MST-Prim(G,s)

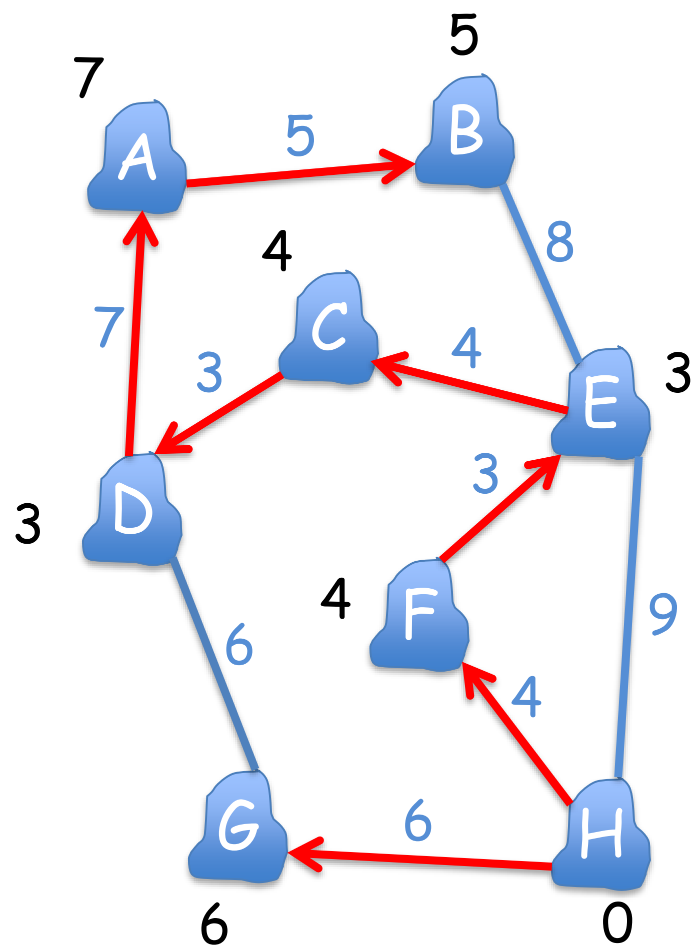
```
for each u of V
      u.key = ∞
      u.par = nil
s.key = 0
create a minimum priority Q on V
while Q ≠ { }
      u = ExtractMin(Q)
      for each v of Adj(u)
            if v in Q and w(u,v) < v.key
                  v.par = u
                  v.key = w(u,v)
```



Prim's Algorithm

MST-Prim(G,s)

```
for each u of V
    u.key = ∞
    u.par = nil
s.key = 0
create a minimum priority Q on V
while Q ≠ { }
    u = ExtractMin(Q)
    for each v of Adj(u)
        if v in Q and w(u,v) < v.key
            v.par = u
            v.key = w(u,v)
```



Prim's Algorithm

MST-Prim(G,s)

```
for each u of V
      u.key = ∞
      u.par = nil
s.key = 0
create a minimum priority Q on V
while Q ≠ { }
      u = ExtractMin(Q)
      for each v of Adj(u)
            if v in Q and w(u,v) < v.key
                  v.par = u
                  v.key = w(u,v)
```