Graph Algorithms (Chapter 23)
Minimum Spanning Trees

University of Stavanger

Goals

Weighted graph definition and examples

The minimum spanning tree problem

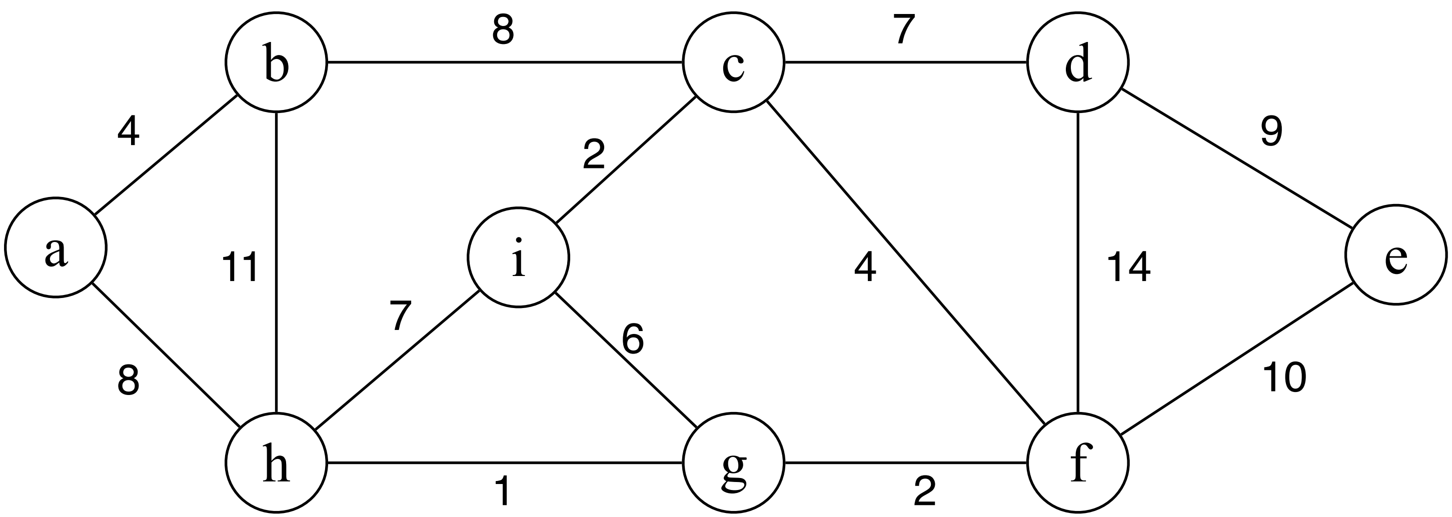
A generic solution to the problem

Algorithm of Kruskal

Algorithm of Prim

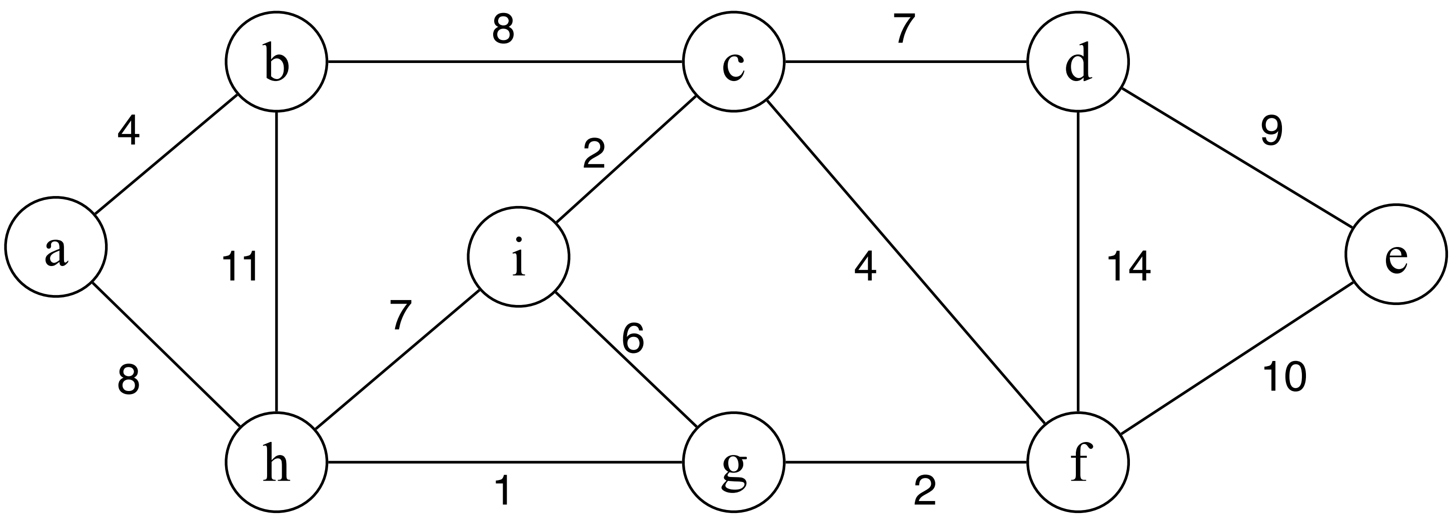
Graph Algorithms (Chapter 23)
Minimum Spanning Trees

$G = (V, E)$ is an undirected graph where every edge is associated with a weight



Graph Algorithms (Chapter 23) Minimum Spanning Trees

- A tree is a connected acyclic graph
- A spanning tree of a graph is a tree containing all vertices of G
- A minimum spanning tree (MST) is a minimum total weight spanning tree



Find the MST of this graph !

Generic greedy strategy:

1. Prior to each iteration, A is a subset of some minimum spanning tree.
2. At each step determine and edge $e = (u, v)$ such that $A \cup \{e\}$ does not violate condition 1. Meaning it is safe to add e (safe edge).

```
GENERIC-MST(G, w)
A = ∅
while A does not form a spanning tree
    find an edge (u, v) that is safe for A
    A = A ∪ {(u, v)}
return A
```

Graph Algorithms (Chapter 23)
Minimum Spanning Trees

Generic greedy strategy:

We reason in a way that there exists a spanning Tree T and $A \subseteq T$. This means that an edges $(u, v) \in T$ and $(u, v) \notin A$ is a safe edge to add to A

How to recognise that safe edge (u, v) when you don't know T yet ?!

```
GENERIC-MST(G, w)
  A = Ø
  while A does not form a spanning tree
      find an edge (u, v) that is safe for A
      A = A ∪ {(u, v)}
  return A
```

University of Stavanger

Cut definition

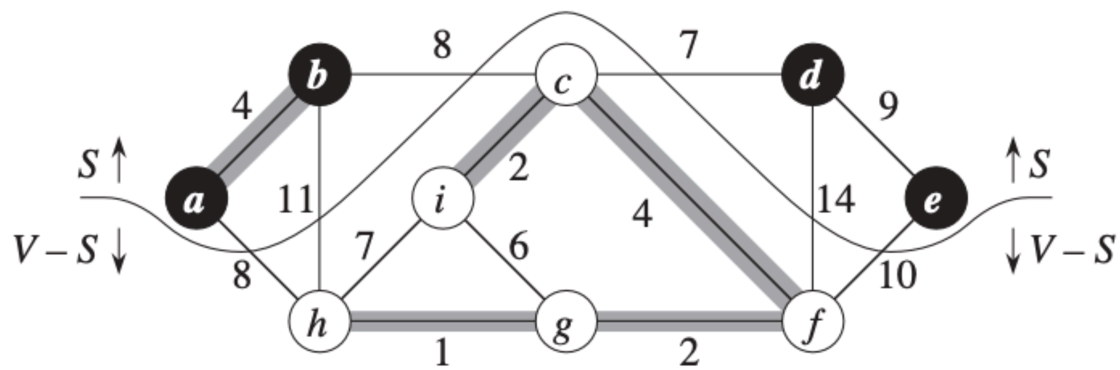
- Cut: cut $(S, S - V)$ of $G = (V, E)$ is a partition of V
- An edge $(u, v) \in E$ crosses the cut $(S, V - S)$ if one of its endpoints is in S and the other in $V - S$
- A cut respects A if no edge in A crosses the the cut
- An edge is light edge crossing a cut if its weight is minimum of any edge crossing the cut.

Theorem 23.1

Let $G = (V, E)$ be a connected, undirected graph with a real-valued weight function w defined on E. Let A be a subset of E that is included in some minimum spanning tree for G, let $(S, V - S)$ be any cut of G that respects A, and let (u, v) be a light edge crossing $(S, V - S)$. Then, edge (u, v) is safe for A.

University of Stavanger

How to recognise a safe edge (u, v)? Option 1

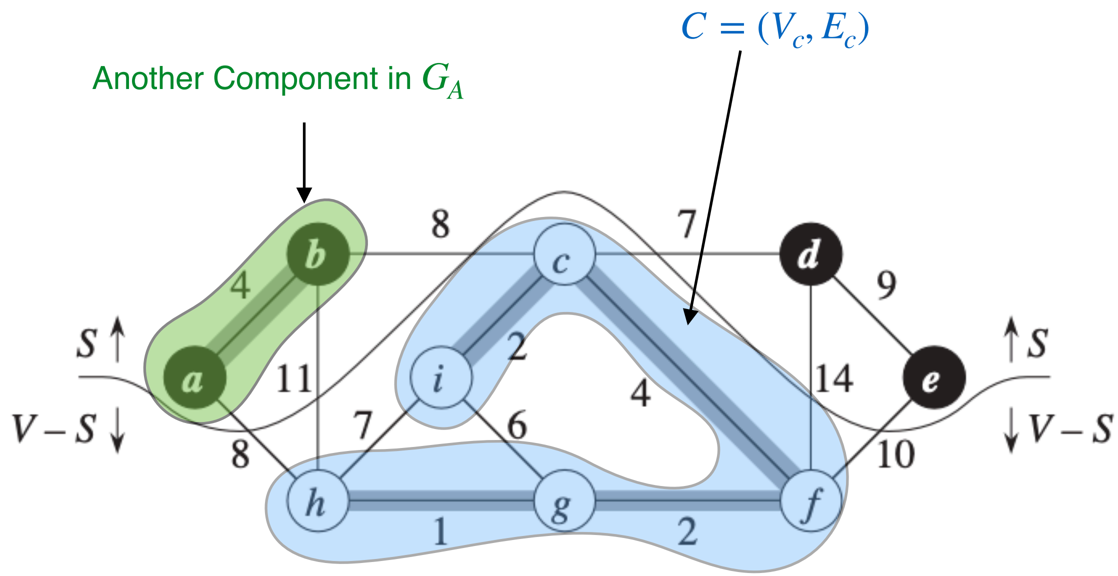


(a)

Theorem 23.1
Let $G = (V, E)$ be a connected, undirected graph with a real-valued weight function w defined on E. Let A be a subset of E that is included in some minimum spanning tree for G, let $(S, V - S)$ be any cut of G that respects A, and let (u, v) be a light edge crossing $(S, V - S)$. Then, edge (u, v) is safe for A.

University of Stavanger

How to recognise a safe edge (u, v)? Option 2



Corollary 23.2
Let $G = (V, E)$ be a connected, undirected graph with a real-valued weight function w defined on E. Let A be a subset of E that is included in some minimum spanning tree for G, and let $C = (V_C, E_C)$ be a connected component (tree) in the forest $G_A = (V, A)$. If (u, v) is a light edge connecting C to some other component in G_A, then (u, v) is safe for A.

Graph Algorithms (Chapter 23)
Minimum Spanning Trees

University of Stavanger

```
MST-KRUSKAL(G, w)
A = ∅
for each vertex v ∈ G.V
    MAKE-SET(v)
sort the edges of G.E into nondecreasing order by weight w
for each edge (u, v) ∈ G.E, taken in nondecreasing order by weight
    if FIND-SET(u) ≠ FIND-SET(v)
        A = A ∪ {(u, v)}
        UNION(u, v)
return A
```

```
MST-PRIM(G, w, r)
for each u ∈ G.V
    u.key = ∞
    u.π = NIL
r.key = 0
Q = G.V
while Q ≠ ∅
    u = EXTRACT-MIN(Q)
    for each v ∈ G.Adj[u]
        if v ∈ Q and w(u, v) < v.key
            v.π = u
            v.key = w(u, v)
```

Two algorithms following the same generic approach

```
GENERIC-MST(G, w)
A = ∅
while A does not form a spanning tree
    find an edge (u, v) that is safe for A
    A = A ∪ {(u, v)}
return A
```

University of Stavanger

Start with disjoint sets and merge them to form one single set. Merge starting from the smallest weight edge (greedy choice)

```
MST-KRUSKAL(G, w)
A = Ø
for each vertex v ∈ G.V
    MAKE-SET(v)
sort the edges of G.E into nondecreasing order by weight w
for each edge (u, v) ∈ G.E, taken in nondecreasing order by weight
    if FIND-SET(u) ≠ FIND-SET(v)
        A = A ∪ {(u, v)}
        UNION(u, v)
return A
```

? We will come back to this

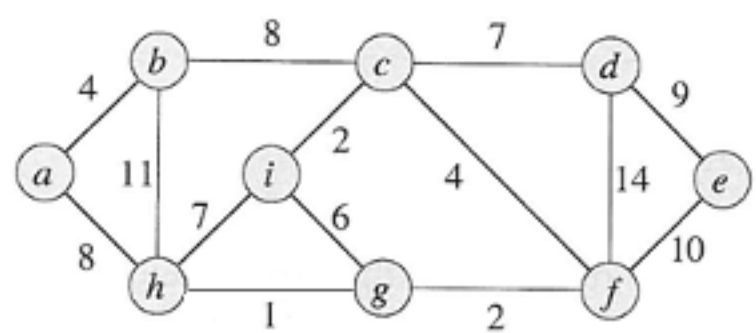
Disjoint-set operations(from chapter 21.3)

- MAKE-SET(x): Makes a set of element x where x is the set representative
- UNION(x,y): merge the set containing x with the set containing y
- FIND-SET(x): find the set (set representative) that contains the element x

Start with disjoint sets and merge them to form one single set. Merge starting from the smallest weight edge (greedy choice)

```
MST-KRUSKAL(G, w)
A = ∅
for each vertex v ∈ G.V
    MAKE-SET(v)
sort the edges of G.E into nondecreasing order by weight w
for each edge (u, v) ∈ G.E, taken in nondecreasing order by weight
    if FIND-SET(u) ≠ FIND-SET(v)
        A = A ∪ {(u, v)}
        UNION(u, v)
return A
```

$A = \{\}$

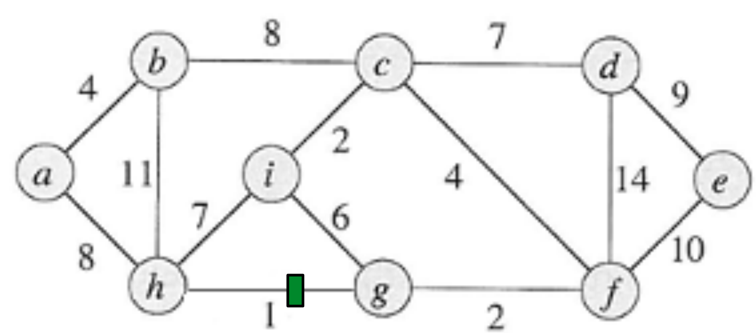


Graph Algorithms (Chapter 23)
Minimum Spanning Trees

Start with disjoint sets and merge them to form one single set. Merge starting from the smallest weight edge (greedy choice)

```
MST-KRUSKAL(G, w)
    A = ∅
    for each vertex v ∈ G.V
        MAKE-SET(v)
    sort the edges of G.E into nondecreasing order by weight w
    for each edge (u, v) ∈ G.E, taken in nondecreasing order by weight
        if FIND-SET(u) ≠ FIND-SET(v)
            A = A ∪ {(u, v)}
            UNION(u, v)
    return A
```

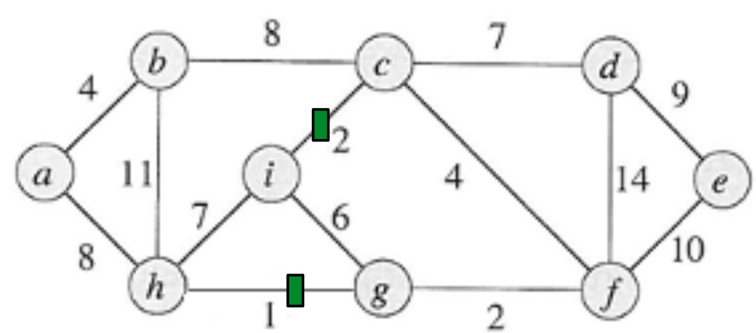
$A = \{(h, g)\}$



Start with disjoint sets and merge them to form one single set. Merge starting from the smallest weight edge (greedy choice)

```
MST-KRUSKAL(G, w)
A = Ø
for each vertex v ∈ G.V
    MAKE-SET(v)
sort the edges of G.E into nondecreasing order by weight w
for each edge (u, v) ∈ G.E, taken in nondecreasing order by weight
    if FIND-SET(u) ≠ FIND-SET(v)
        A = A ∪ {(u, v)}
        UNION(u, v)
return A
```

$A = \{(h, g), (i, c)\}$



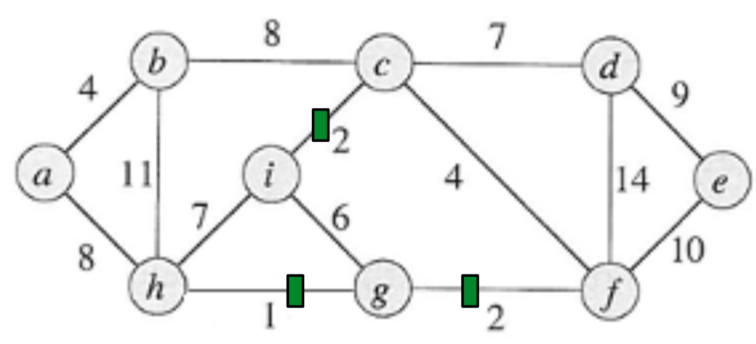
Graph Algorithms (Chapter 23)
Minimum Spanning Trees

University of Stavanger

Start with disjoint sets and merge them to form one single set. Merge starting from the smallest weight edge (greedy choice)

```
MST-KRUSKAL(G, w)
A = Ø
for each vertex v ∈ G.V
    MAKE-SET(v)
sort the edges of G.E into nondecreasing order by weight w
for each edge (u, v) ∈ G.E, taken in nondecreasing order by weight
    if FIND-SET(u) ≠ FIND-SET(v)
        A = A ∪ {(u, v)}
        UNION(u, v)
return A
```

$A = \{(h, g), (i, c), (g, f)\}$

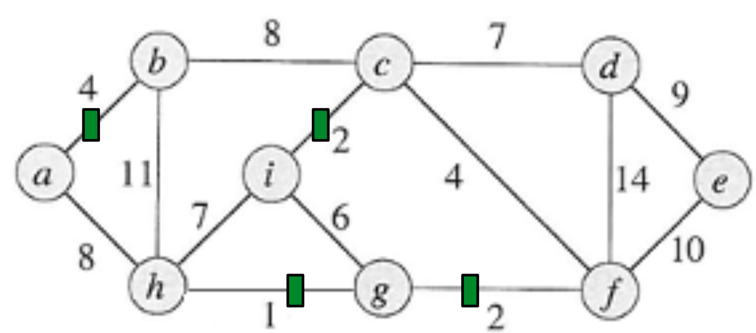


Graph Algorithms (Chapter 23)
Minimum Spanning Trees

Start with disjoint sets and merge them to form one single set. Merge starting from the smallest weight edge (greedy choice)

```
MST-KRUSKAL(G, w)
A = ∅
for each vertex v ∈ G.V
    MAKE-SET(v)
sort the edges of G.E into nondecreasing order by weight w
for each edge (u, v) ∈ G.E, taken in nondecreasing order by weight
    if FIND-SET(u) ≠ FIND-SET(v)
        A = A ∪ {(u, v)}
        UNION(u, v)
return A
```

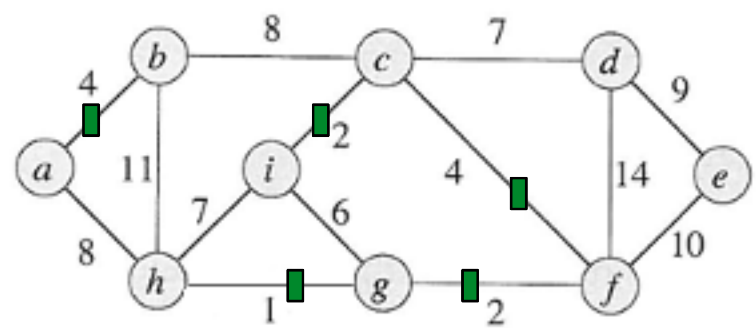
$A = \{(h,g),(i,c),(g,f),(a,b)\}$



Start with disjoint sets and merge them to form one single set. Merge starting from the smallest weight edge (greedy choice)

```
MST-KRUSKAL(G, w)
A = Ø
for each vertex v ∈ G.V
    MAKE-SET(v)
sort the edges of G.E into nondecreasing order by weight w
for each edge (u, v) ∈ G.E, taken in nondecreasing order by weight
    if FIND-SET(u) ≠ FIND-SET(v)
        A = A ∪ {(u, v)}
        UNION(u, v)
return A
```

$A = \{(h, g), (i, c), (g, f), (a, b), (c, f)\}$

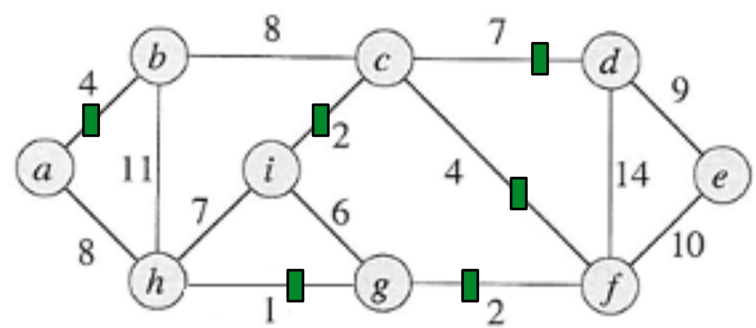


Graph Algorithms (Chapter 23) Minimum Spanning Trees

Start with disjoint sets and merge them to form one single set. Merge starting from the smallest weight edge (greedy choice)

```
MST-KRUSKAL(G, w)
A = ∅
for each vertex v ∈ G.V
    MAKE-SET(v)
sort the edges of G.E into nondecreasing order by weight w
for each edge (u, v) ∈ G.E, taken in nondecreasing order by weight
    if FIND-SET(u) ≠ FIND-SET(v)
        A = A ∪ {(u, v)}
        UNION(u, v)
return A
```

$$A = \{(h,g),(i,c),(g,f),(a,b),(c,f),(c,d)\}$$



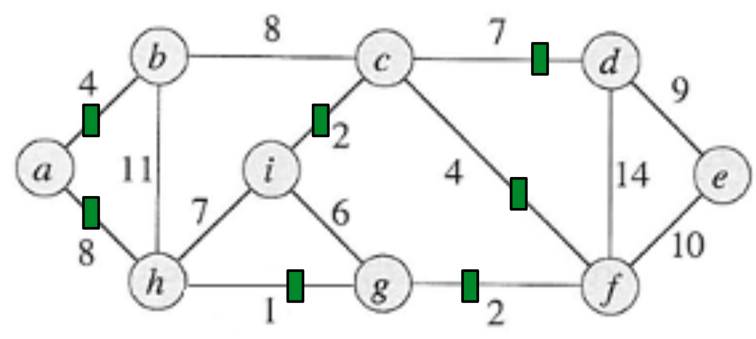
Graph Algorithms (Chapter 23)
Minimum Spanning Trees

University of Stavanger

Start with disjoint sets and merge them to form one single set. Merge starting from the smallest weight edge (greedy choice)

```
MST-KRUSKAL(G, w)
A = ∅
for each vertex v ∈ G.V
    MAKE-SET(v)
sort the edges of G.E into nondecreasing order by weight w
for each edge (u, v) ∈ G.E, taken in nondecreasing order by weight
    if FIND-SET(u) ≠ FIND-SET(v)
        A = A ∪ {(u, v)}
        UNION(u, v)
return A
```

$A = \{(h,g),(i,c),(g,f),(a,b),(c,f),(c,d),(a,h)\}$

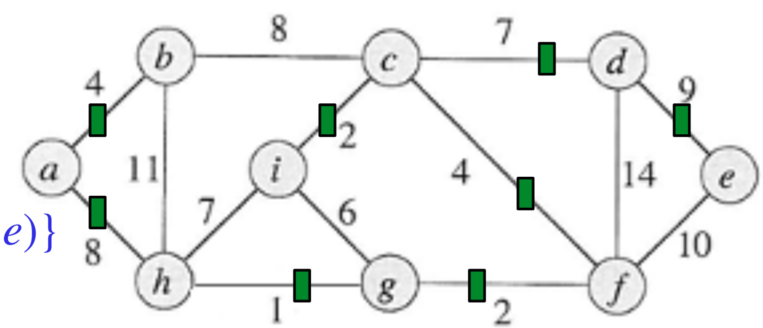


Graph Algorithms (Chapter 23)
Minimum Spanning Trees

Start with disjoint sets and merge them to form one single set. Merge starting from the smallest weight edge (greedy choice)

```
MST-KRUSKAL(G, w)
A = Ø
for each vertex v ∈ G.V
    MAKE-SET(v)
sort the edges of G.E into nondecreasing order by weight w
for each edge (u, v) ∈ G.E, taken in nondecreasing order by weight
    if FIND-SET(u) ≠ FIND-SET(v)
        A = A ∪ {(u, v)}
        UNION(u, v)
return A
```

$A = \{(h,g),(i,c),(g,f),(a,b),(c,f),(c,d),(a,h),(d,e)\}$



University of Stavanger

Disjoint-set operations(from chapter 21.3)

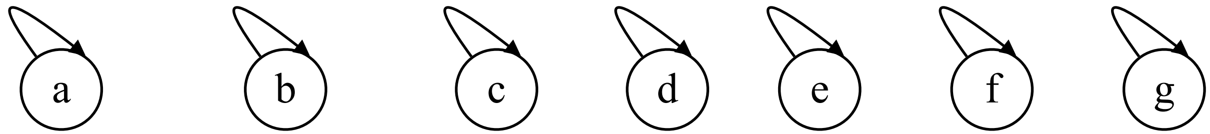
- MAKE-SET(x): Makes a set of element x where x is the set representative
- UNION(x,y): merge the set containing x with the set containing y
- FIND-SET(x): find the set (set representative) that contains the element x

Explanation

Graph Algorithms (Chapter 23) Minimum Spanning Trees

Disjoint-set operations(from chapter 21.3)

MAKE-SET(x) for {a,b,c,d,e,f,g}



MAKE-SET(x)

1 $x.p = x$

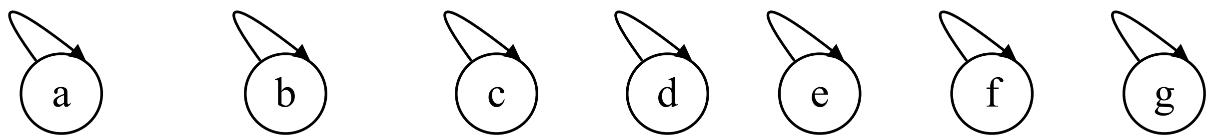
2 $x.rank = 0$

What is the runtime complexity of MAKE-SET(x)?

Graph Algorithms (Chapter 23)
Minimum Spanning Trees

Disjoint-set operations(from chapter 21.3)

MAKE-SET(x) for {a,b,c,d,e,f,g}



MAKE-SET(x)

```
1  x.p = x
2  x.rank = 0
```

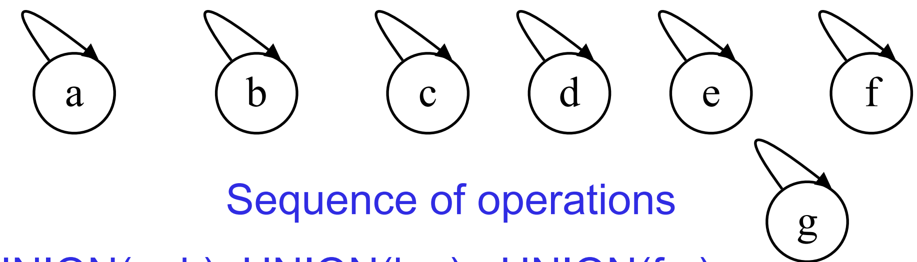
What is the runtime complexity of MAKE-SET(x)?

$\Theta(1)$

Graph Algorithms (Chapter 23)
Minimum Spanning Trees

Disjoint-set operations(from chapter 21.3)

```
MAKE-SET(x)
    x.p = x
    x.rank = 0

UNION(x, y)
    LINK(FIND-SET(x), FIND-SET(y))

LINK(x, y)
    if x.rank > y.rank
        y.p = x
    else x.p = y
        if x.rank == y.rank
            y.rank = y.rank + 1

FIND-SET(x)
    if x ≠ x.p
        x.p = FIND-SET(x.p)
    return x.p
```



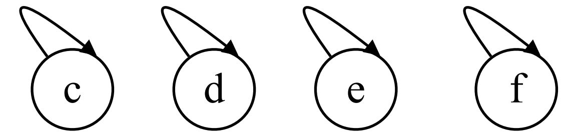
Sequence of operations

UNION(a, b), UNION(b,c), ,UNION(f,g), UNION(c, d), UNION(d,f)?

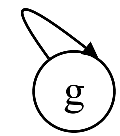
University of Stavanger

Disjoint-set operations(from chapter 21.3)

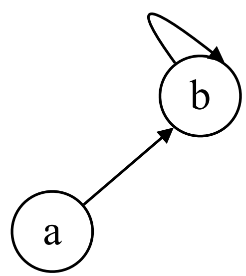
```
MAKE-SET(x)
  x.p = x
  x.rank = 0

UNION(x, y)
  LINK(FIND-SET(x), FIND-SET(y))

LINK(x, y)
  if x.rank > y.rank
      y.p = x
  else x.p = y
      if x.rank == y.rank
          y.rank = y.rank + 1

FIND-SET(x)
  if x ≠ x.p
      x.p = FIND-SET(x.p)
  return x.p
```



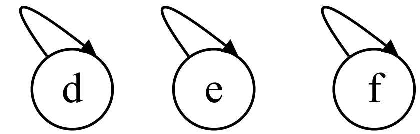
Sequence of operations



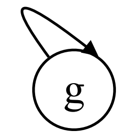
~~UNION(a, b)~~, UNION(b,c), ,UNION(f,g), UNION(c, d), UNION(d,f)?



Disjoint-set operations(from chapter 21.3)

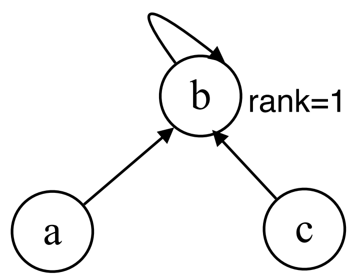


Sequence of operations



~~UNION(a, b)~~, ~~UNION(b,c)~~, ,UNION(f,g), UNION(c, d), UNION(d,f)?

```
MAKE-SET(x)
    x.p = x
    x.rank = 0

UNION(x, y)
    LINK(FIND-SET(x), FIND-SET(y))

LINK(x, y)
    if x.rank > y.rank
        y.p = x
    else x.p = y
        if x.rank == y.rank
            y.rank = y.rank + 1

FIND-SET(x)
    if x ≠ x.p
        x.p = FIND-SET(x.p)
    return x.p
```

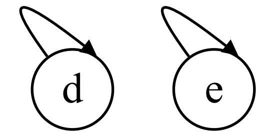


Graph Algorithms (Chapter 23)
Minimum Spanning Trees

University of Stavanger

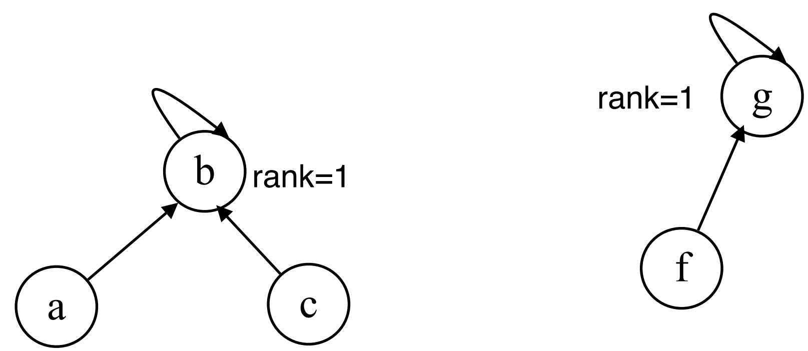
Disjoint-set operations(from chapter 21.3)

```
MAKE-SET(x)
    x.p = x
    x.rank = 0

UNION(x, y)
    LINK(FIND-SET(x), FIND-SET(y))

LINK(x, y)
    if x.rank > y.rank
        y.p = x
    else x.p = y
        if x.rank == y.rank
            y.rank = y.rank + 1

FIND-SET(x)
    if x ≠ x.p
        x.p = FIND-SET(x.p)
    return x.p
```



Sequence of operations

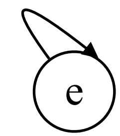
~~UNION(a, b)~~, ~~UNION(b,c)~~, ,~~UNION(f,g),~~
UNION(c, d), UNION(d,f)?



Graph Algorithms (Chapter 23)
Minimum Spanning Trees

University of Stavanger

Disjoint-set operations(from chapter 21.3)



MAKE-SET(x)
```
x.p = x
x.rank = 0
```

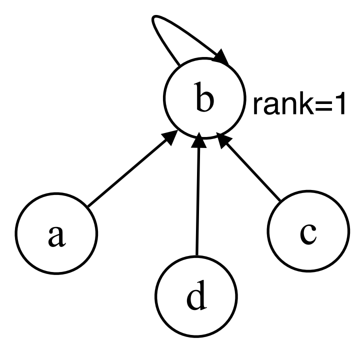
UNION(x, y)
```
LINK(FIND-SET(x), FIND-SET(y))
```

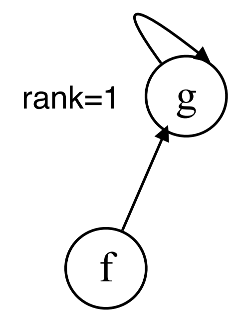
LINK(x, y)
```
if x.rank > y.rank
    y.p = x
else x.p = y
    if x.rank == y.rank
        y.rank = y.rank + 1
```

FIND-SET(x)
```
if x ≠ x.p
    x.p = FIND-SET(x.p)
return x.p
```

Sequence of operations

~~UNION(a, b)~~, ~~UNION(b,c)~~, ,~~UNION(f,g),~~ ~~UNION(c, d)~~, UNION(d,f)?





Graph Algorithms (Chapter 23) Minimum Spanning Trees

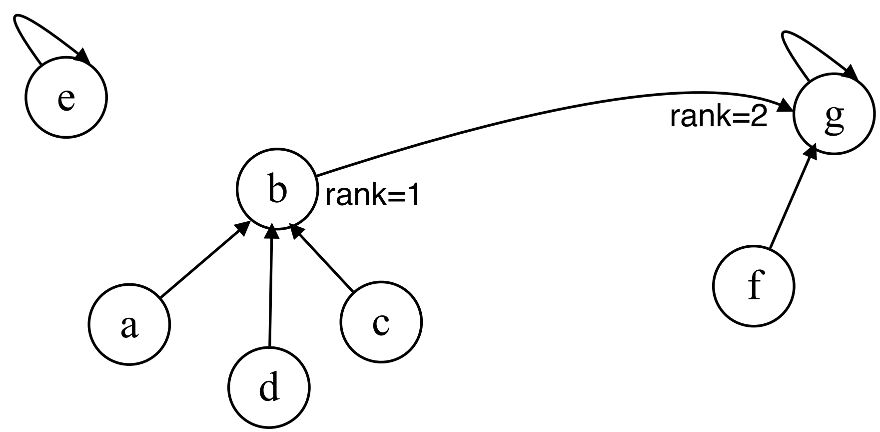
University of Stavanger

Disjoint-set operations(from chapter 21.3)

```
MAKE-SET(x)
  x.p = x
  x.rank = 0

UNION(x, y)
  LINK(FIND-SET(x), FIND-SET(y))

LINK(x, y)
  if x.rank > y.rank
      y.p = x
  else x.p = y
      if x.rank == y.rank
          y.rank = y.rank + 1

FIND-SET(x)
  if x ≠ x.p
      x.p = FIND-SET(x.p)
  return x.p
```

Sequence of operations

~~UNION(a, b)~~, ~~UNION(b,c)~~, ,~~UNION(f,g),~~ ~~UNION(c, d)~~, ~~UNION(d,f)~~?



Disjoint-set operations(from chapter 21.3)

MAKE-SET(x)
```
x.p = x
x.rank = 0
```

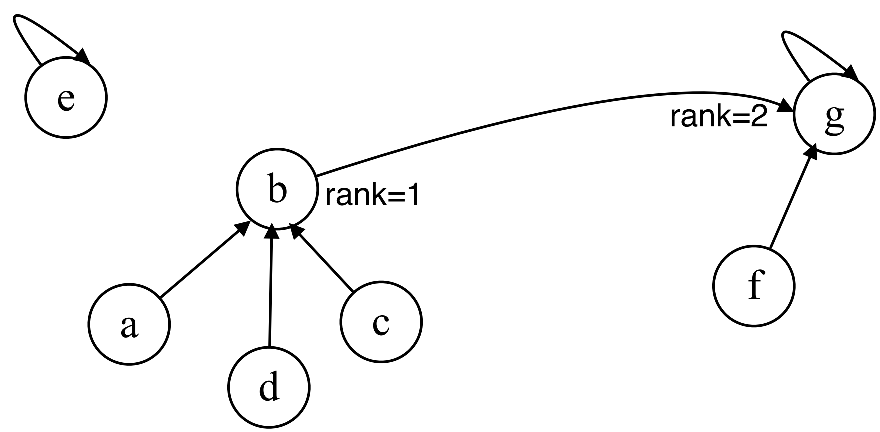
UNION(x, y)
```
LINK(FIND-SET(x), FIND-SET(y))
```

LINK(x, y)
```
if x.rank > y.rank
      y.p = x
else x.p = y
      if x.rank == y.rank
            y.rank = y.rank + 1
```

FIND-SET(x)
```
if x ≠ x.p
      x.p = FIND-SET(x.p)
return x.p
```

What is the time complexity of

UNION(x,y)?



Graph Algorithms (Chapter 23)
Minimum Spanning Trees

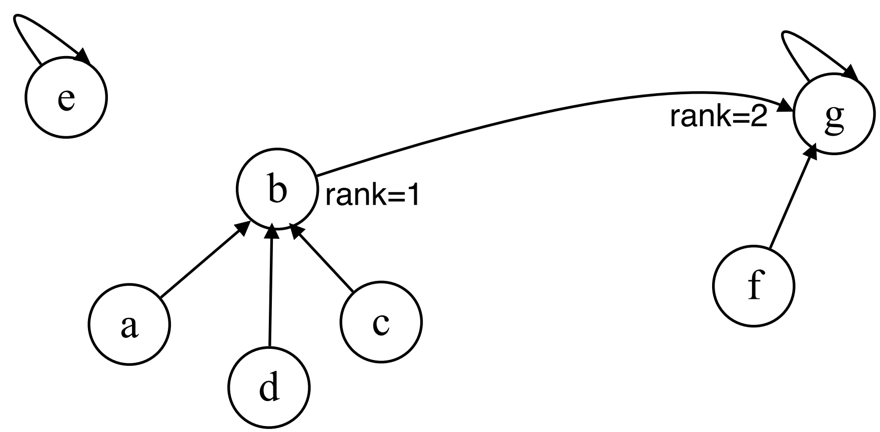
Disjoint-set operations(from chapter 21.3)

```
MAKE-SET(x)
  x.p = x
  x.rank = 0

UNION(x, y)
  LINK(FIND-SET(x), FIND-SET(y))

LINK(x, y)
  if x.rank > y.rank
      y.p = x
  else x.p = y
      if x.rank == y.rank
          y.rank = y.rank + 1

FIND-SET(x)
  if x ≠ x.p
      x.p = FIND-SET(x.p)
  return x.p
```

What is the time complexity of

UNION(x,y)?

Depends on FIND-SET



Graph Algorithms (Chapter 23) Minimum Spanning Trees

Disjoint-set operations(from chapter 21.3)

MAKE-SET(x)

$x.p = x$
$x.rank = 0$

UNION(x, y)

LINK(FIND-SET(x), FIND-SET(y))

LINK(x, y)

if $x.rank > y.rank$
    $y.p = x$
else $x.p = y$
    if $x.rank == y.rank$
        $y.rank = y.rank + 1$

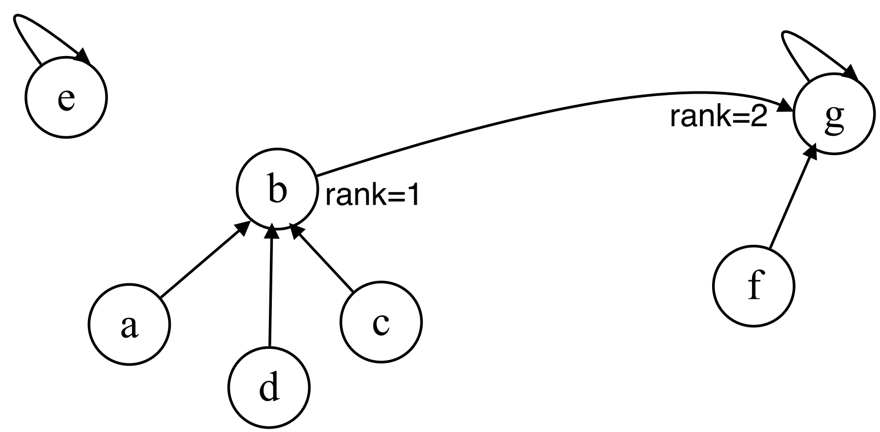
FIND-SET(x)

if $x \neq x.p$
    $x.p =$ FIND-SET$(x.p)$
return $x.p$

What is the time complexity of

UNION(x,y)?

FIND-SET(x)= $\Theta(log(n))$



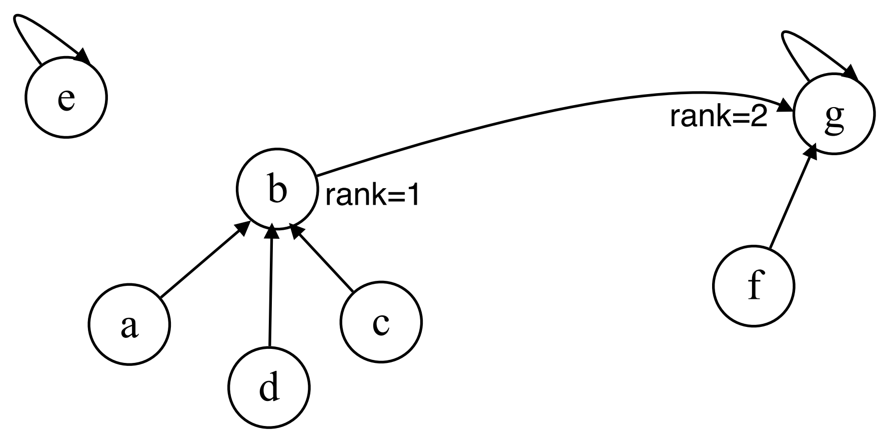
Graph Algorithms (Chapter 23)
Minimum Spanning Trees

Disjoint-set operations(from chapter 21.3)

```
MAKE-SET(x)
  x.p = x
  x.rank = 0

UNION(x, y)
  LINK(FIND-SET(x), FIND-SET(y))

LINK(x, y)
  if x.rank > y.rank
      y.p = x
  else x.p = y
      if x.rank == y.rank
          y.rank = y.rank + 1

FIND-SET(x)
  if x ≠ x.p
      x.p = FIND-SET(x.p)
  return x.p
```

Can we do better?

FIND-SET(x)= $\Theta(log(n))$



Graph Algorithms (Chapter 23)
Minimum Spanning Trees

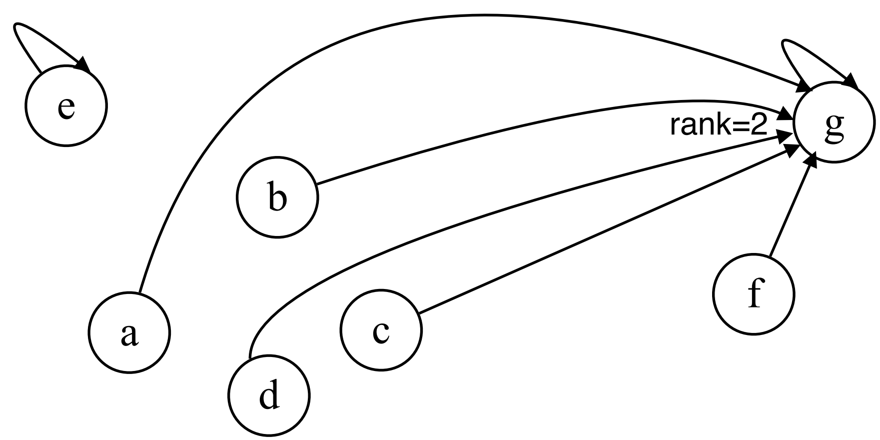
Disjoint-set operations(from chapter 21.3)

```
MAKE-SET(x)
    x.p = x
    x.rank = 0

UNION(x, y)
    LINK(FIND-SET(x), FIND-SET(y))

LINK(x, y)
    if x.rank > y.rank
        y.p = x
    else x.p = y
        if x.rank == y.rank
            y.rank = y.rank + 1

FIND-SET(x)
    if x ≠ x.p
        x.p = FIND-SET(x.p)
    return x.p
```

Can we do better?

Path compression

On FIND-SET each element on the path gets as parent the set representative



University of Stavanger

What is the runtime complexity of Kruskal Algorithm?

```
MST-KRUSKAL(G, w)
A = ∅
for each vertex v ∈ G.V
    MAKE-SET(v)
sort the edges of G.E into nondecreasing order by weight w
for each edge (u, v) ∈ G.E, taken in nondecreasing order by weight
    if FIND-SET(u) ≠ FIND-SET(v)
        A = A ∪ {(u, v)}
        UNION(u, v)
return A
```

What is the runtime complexity of Kruskal Algorithm?

$O(E\,.\,lgE)$ and since $|V^2| \geq |E| \geq |V| - 1$ we have $lg|E| = O(lgV)$,

we can also say that the runtime time is $O(E\,.\,lgV)$

MST-KRUSKAL(G, w)

$A = \emptyset$
for each vertex $v \in G.V$ — $O(V)$
  MAKE-SET(v) — $O(1)$
sort the edges of $G.E$ into nondecreasing order by weight w — $O(E\,.\,lgE)$
for each edge $(u, v) \in G.E$, taken in nondecreasing order by weight — $O(E)$
  if FIND-SET$(u) \neq$ FIND-SET(v) — $O(lgV)$
    $A = A \cup \{(u, v)\}$ — $O(1)$
    UNION(u, v) — $O(lgV)$
return A

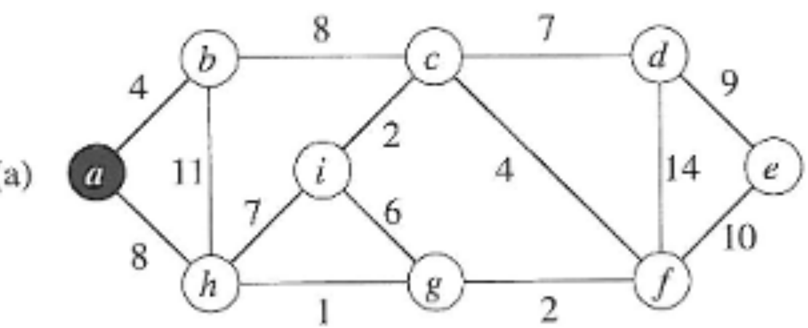
or less by compressed path

Graph Algorithms (Chapter 23)
Minimum Spanning Trees

University of Stavanger

Use a priority queue holding the vertices to be extracted by minimum edge weight. Update the parents of the vertices to construct a tree

```
MST-PRIM(G, w, r)
for each u ∈ G.V
    u.key = ∞
    u.π = NIL
r.key = 0
Q = G.V
while Q ≠ ∅
    u = EXTRACT-MIN(Q)
    for each v ∈ G.Adj[u]
        if v ∈ Q and w(u, v) < v.key
            v.π = u
            v.key = w(u, v)
```



Start at r, and follow the minimum weighted edge, update the keys in Q.

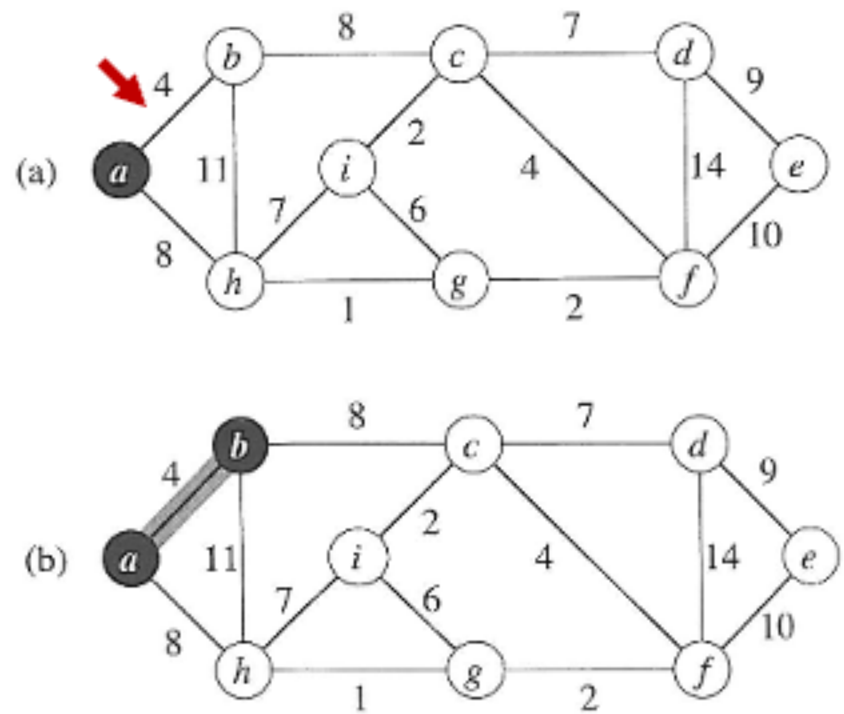
After a, b will have b.key=4 and h.key=8.

The next to extract will be b because b.key is the minimum

Graph Algorithms (Chapter 23)
Minimum Spanning Trees

Use a priority queue holding the vertices to be extracted by minimum edge weight. Update the parents of the vertices to construct a tree

```
MST-PRIM(G, w, r)
for each u ∈ G.V
    u.key = ∞
    u.π = NIL
r.key = 0
Q = G.V
while Q ≠ ∅
    u = EXTRACT-MIN(Q)
    for each v ∈ G.Adj[u]
        if v ∈ Q and w(u, v) < v.key
            v.π = u
            v.key = w(u, v)
```

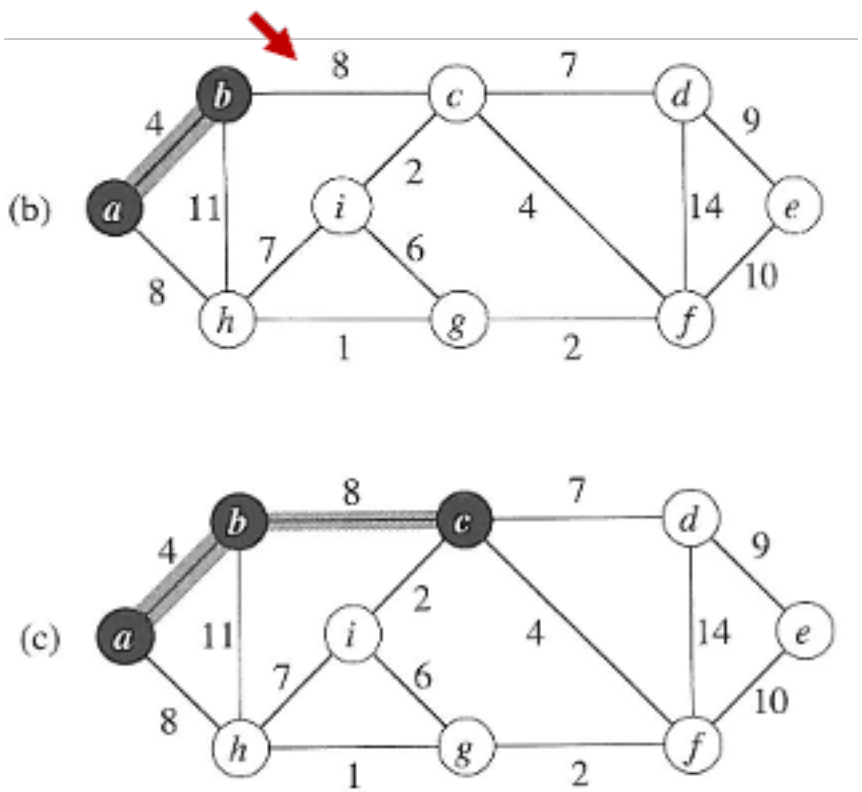


Graph Algorithms (Chapter 23)
Minimum Spanning Trees

Use a priority queue holding the vertices to be extracted by minimum edge weight. Update the parents of the vertices to construct a tree

MST-PRIM(G, w, r)

```
for each u ∈ G.V
    u.key = ∞
    u.π = NIL
r.key = 0
Q = G.V
while Q ≠ ∅
    u = EXTRACT-MIN(Q)
    for each v ∈ G.Adj[u]
        if v ∈ Q and w(u, v) < v.key
            v.π = u
            v.key = w(u, v)
```



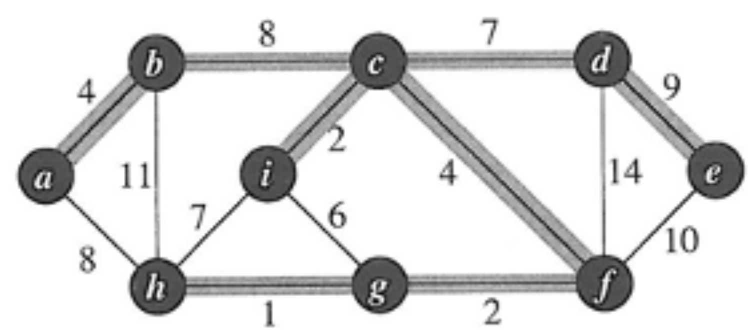
Graph Algorithms (Chapter 23)
Minimum Spanning Trees

Use a priority queue holding the vertices to be extracted by minimum edge weight. Update the parents of the vertices to construct a tree

What the runtime of MST-PRIM?

```
MST-PRIM(G, w, r)
for each u ∈ G.V
    u.key = ∞
    u.π = NIL
r.key = 0
Q = G.V
while Q ≠ ∅
    u = EXTRACT-MIN(Q)
    for each v ∈ G.Adj[u]
        if v ∈ Q and w(u, v) < v.key
            v.π = u
            v.key = w(u, v)
```

(i)



Graph Algorithms (Chapter 23)
Minimum Spanning Trees

Use a priority queue holding the vertices to be extracted by minimum edge weight. Update the parents of the vertices to construct a tree

What the runtime of MST-PRIM?

$$O(V\,.\,lgV + E\,.\,lgV) = O(E\,.\,lgV)$$

because $|E| \geq |V| - 1$

```
MST-PRIM(G, w, r)
for each u ∈ G.V
    u.key = ∞
    u.π = NIL
r.key = 0
Q = G.V
while Q ≠ ∅
    u = EXTRACT-MIN(Q)
    for each v ∈ G.Adj[u]
        if v ∈ Q and w(u, v) < v.key
            v.π = u
            v.key = w(u, v)
```

$O(V)$

$O(VlgV)$ total times we extract min

$O(E)$ total times we enter the loop

$O(lgV)$ from chapter 6.5

```
HEAP-INCREASE-KEY(A, i, key)
if key < A[i]
    error "new key is smaller than current key"
A[i] = key
while i > 1 and A[PARENT(i)] < A[i]
    exchange A[i] with A[PARENT(i)]
    i = PARENT(i)
```

Summary Chapter 23

- Minimum spanning tree applies to undirected graphs
- A generic solution focuses on always picking a safe edge to add
- A safe edge to add to a minimum spanning tree is the greedy choice that can be taken
- Kruskal starts with a forest of trees and builds a minimum spanning tree by linking the trees with safe edges
- Prims grows a minimum spanning tree by maintaining a min-priority queue
- Both algorithms run in $O(E \,.\, lgV)$ depending on how the priority queue and disjoint set operations are implemented.

Exercices

Will come