Data Structures and Algorithms [1]

BITS-Pilani K. K. Birla Goa Campus

[1]Material for the presentation taken from Cormen, Leiserson, Rivest and Stein, *Introduction to Algorithms, Fourth Edition*;

Lecture plan

- Ch. 20 : Elementary Graph Algorithms (Section 20.5 (Strongly connected components) **not** part of CS F211 syllabus)

Lecture plan

- Ch. 20 : Elementary Graph Algorithms (Section 20.5 (Strongly connected components) **not** part of CS F211 syllabus)
- Ch. 21 : Finding minimum spanning trees (Algorithms of Kruskal and Prim)

Lecture plan

- Ch. 20 : Elementary Graph Algorithms (Section 20.5 (Strongly connected components) **not** part of CS F211 syllabus)
- Ch. 21 : Finding minimum spanning trees (Algorithms of Kruskal and Prim)
- Ch. 19 : Data structures for Disjoint sets (Section 19.3 will be part of CS F211 syllabus).

- Data structure to represent a collection of disjoint sets that groups n distinct elements.

Ch. 19: Data structure for Disjoint Sets

- Data structure to represent a collection of disjoint sets that groups n distinct elements.
- Suppose we have **nine** vertices labelled 1 to 9.

Ch. 19: Data structure for Disjoint Sets

- Data structure to represent a collection of disjoint sets that groups n distinct elements.
- Suppose we have **nine** vertices labelled 1 to 9.

  Possible collections of disjoint sets:

  Eg. 1 $\{3, 2, 5\}, \{4, 9\}, \{7, 8, 1\}, \{6\}$

- Data structure to represent a collection of disjoint sets that groups *n* distinct elements.
- Suppose we have **nine** vertices labelled 1 to 9.

  Possible collections of disjoint sets:

  Eg. 1 $\{3, 2, 5\}, \{4, 9\}, \{7, 8, 1\}, \{6\}$ (All vertices are present)

- Data structure to represent a collection of disjoint sets that groups n distinct elements.
- Suppose we have **nine** vertices labelled 1 to 9.

  Possible collections of disjoint sets:

  Eg. 1 $\{3, 2, 5\}, \{4, 9\}, \{7, 8, 1\}, \{6\}$ (All vertices are present)

  Eg. 2 $\{4, 6\}, \{5, 7, 9\}, \{8\}, \{3, 1\}, \{2\}$

- Data structure to represent a collection of disjoint sets that groups *n* distinct elements.
- Suppose we have **nine** vertices labelled 1 to 9.

  Possible collections of disjoint sets:

  Eg. 1 $\{3, 2, 5\}, \{4, 9\}, \{7, 8, 1\}, \{6\}$ (All vertices are present)

  Eg. 2 $\{4, 6\}, \{5, 7, 9\}, \{8\}, \{3, 1\}, \{2\}$

  (We will not have empty set in the collection.)

- Data structure to represent a collection of disjoint sets that groups n distinct elements.
- Suppose we have **nine** vertices labelled 1 to 9.

  Possible collections of disjoint sets:

  Eg. 1 $\{3,2,5\},\{4,9\},\{7,8,1\},\{6\}$ (All vertices are present)

  Eg. 2 $\{4,6\},\{5,7,9\},\{8\},\{3,1\},\{2\}$

  (We will not have empty set in the collection.)
- Suppose we have n elements. What is the maximum number of disjoint sets that we can have in the above collection?

Operations on Disjoint Sets

- Three operations :

Operations on Disjoint Sets

- Three operations :

1. MAKE-SET(X)
2. UNION(X,Y)
3. FIND-SET(X)

Operations on Disjoint Sets

- Three operations :

1. Make-set(x)
2. Union(x,y)
3. Find-set(x)

- The above operations must be performed as efficiently as possible.

Data structure for Disjoint Sets

- Suppose we have the following disjoint sets for 9 items. $\{3, 2, 5\}, \{4, 9\}, \{7, 8, 1\}, \{6\}$

Data structure for Disjoint Sets

- Suppose we have the following disjoint sets for 9 items. $\{3, 2, 5\}, \{4, 9\}, \{7, 8, 1\}, \{6\}$
- Each set will be identified by a representative element:

Data structure for Disjoint Sets

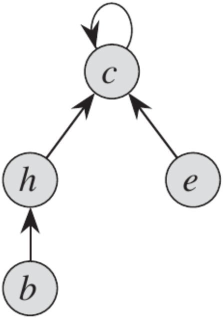
- Suppose we have the following disjoint sets for 9 items.
  $\{3, 2, 5\}, \{4, 9\}, \{7, 8, 1\}, \{6\}$

- Each set will be identified by a representative element:
  Representative for the first set can be 3, for the second set the representative can be 9 and so on.

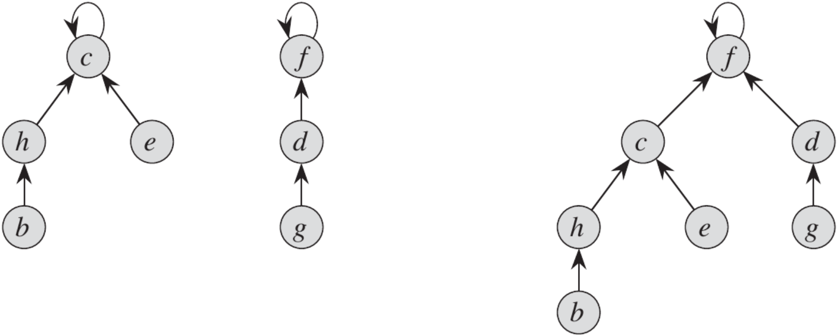
Section 19.3 : Disjoint forest Data Structure

- A disjoint set is represented as a rooted tree.

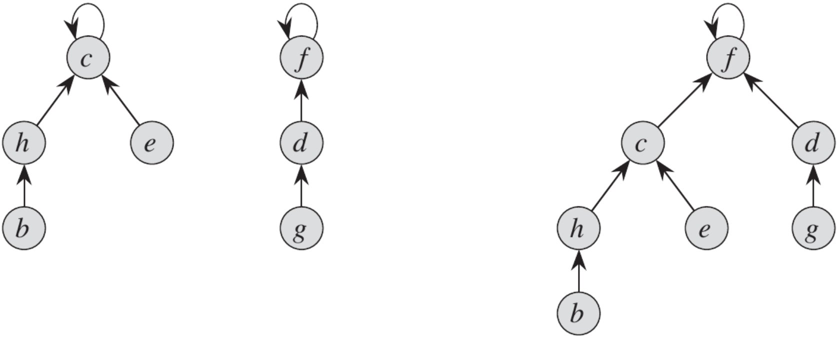
- A disjoint set is represented as a rooted tree.
- Set $\{b, h, c, e\}$



UNION(e,g)

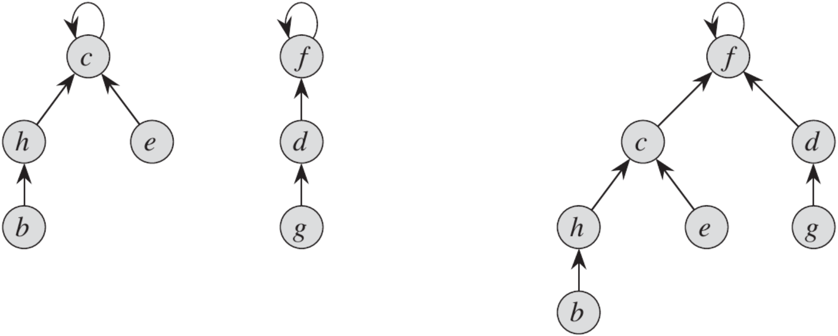


UNION(e,g)



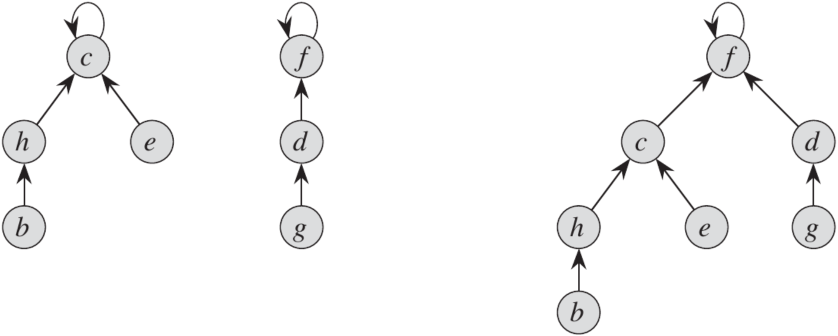
- The two sets must be replaced by the union set.

UNION(e,g)



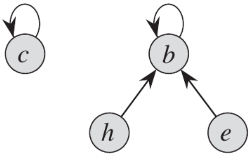
- The two sets must be replaced by the union set.
- What is the maximum number of UNION operations that we can perform if the collection of disjoint sets contain n elements?

UNION(e,g)



- The two sets must be replaced by the union set.
- What is the maximum number of UNION operations that we can perform if the collection of disjoint sets contain n elements? $(n-1)$

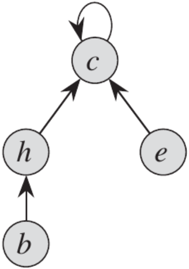
Operations on disjoint set



1. MAKE-SET(X)
2. FIND-SET(X)

Pseudocode for FIND-SET

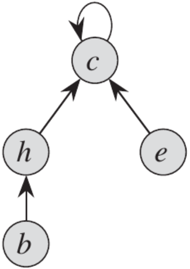
FIND-SET(b) with path compression.




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

Pseudocode for FIND-SET

FIND-SET(b) with path compression.

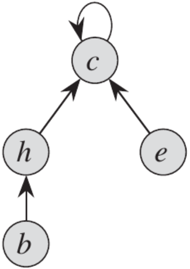


FIND-SET(x)

1 **if** $x \neq x.p$
2 $\quad x.p =$ FIND-SET$(x.p)$
3 **return** $x.p$

Rank of all the nodes remain the same.

FIND-SET(b) with path compression.

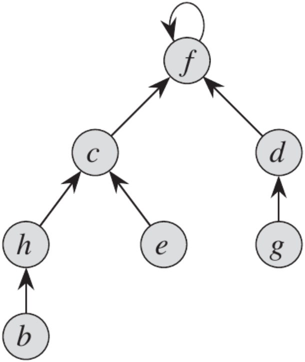


FIND-SET(x)
if $x \neq x.p$
  $x.p =$ FIND-SET($x.p$)
return $x.p$

Rank of all the nodes remain the same.
What is the advantage of path compression?

Pseudocode for FIND-SET

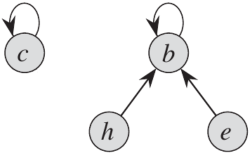
- FIND-SET(b)



FIND-SET(x)

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

UNION procedure



3. UNION(X,Y)

Union by rank heuristic

- We associate a *rank* with each node of the disjoint set tree.

Union by rank heuristic

- We associate a *rank* with each node of the disjoint set tree.
- Rank is an upperbound on the height of the tree.

Union by rank heuristic

- We associate a *rank* with each node of the disjoint set tree.
- Rank is an upperbound on the height of the tree.
- Rank is 0 when MAKE-SET creates a singleton set.

Union by rank heuristic

- We associate a *rank* with each node of the disjoint set tree.
- Rank is an upperbound on the height of the tree.
- Rank is 0 when MAKE-SET creates a singleton set.
- If the ranks are the same, we choose one of the roots as the parent and increment its rank.

Union by rank heuristic

- We associate a *rank* with each node of the disjoint set tree.
- Rank is an upperbound on the height of the tree.
- Rank is 0 when MAKE-SET creates a singleton set.
- If the ranks are the same, we choose one of the roots as the parent and increment its rank.
- Otherwise, we make a Tree with a lower rank a child of a Tree with a higher rank. Rank of the union tree remains unchanged.

Pseudocode for MAKE-SET and UNION

```
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
```

Effect of the two heuristics

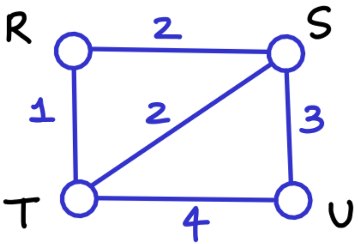
- What are the two heuristics trying to achieve?

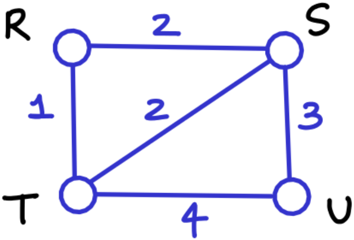
Effect of the two heuristics

- What are the two heuristics trying to achieve?
- Suppose we perform m operations and have n elements in the collection of disjoint sets.

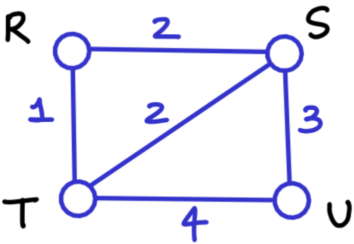
Effect of the two heuristics

- What are the two heuristics trying to achieve?
- Suppose we perform m operations and have n elements in the collection of disjoint sets.
- Overall running time will be $O(m\alpha(n))$, where $\alpha(n) = o(\lg n)$. (α is a very slowly growing function.)


R
2
S
1
2
3
T
4
U

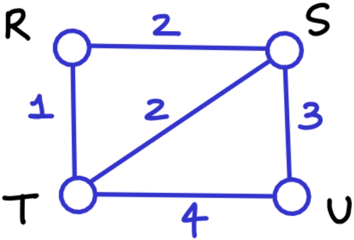


- $G(V, E)$ is a connected, undirected graph. Let w be a real valued weight function.



- $G(V, E)$ is a connected, undirected graph. Let w be a real valued weight function.

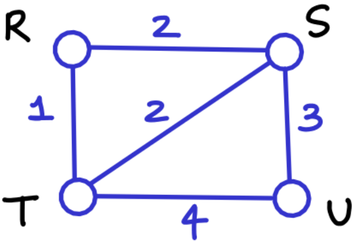
  Weight function $w : E \to \mathbb{R}$.



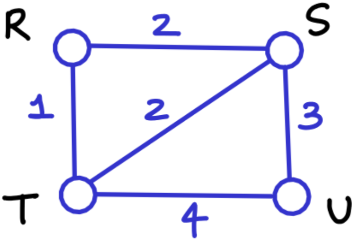
- $G(V, E)$ is a connected, undirected graph. Let w be a real valued weight function.

  Weight function $w : E \to \mathbb{R}$.

- Spanning tree is simply an acyclic subgraph where all vertices remain connected.

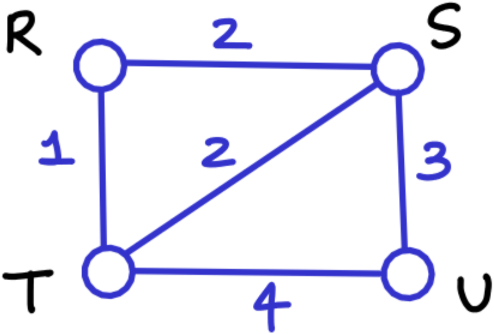


- Weight of a spanning tree:
  $$w(T) = \sum_{(u,v) \in T} w(u, v)$$

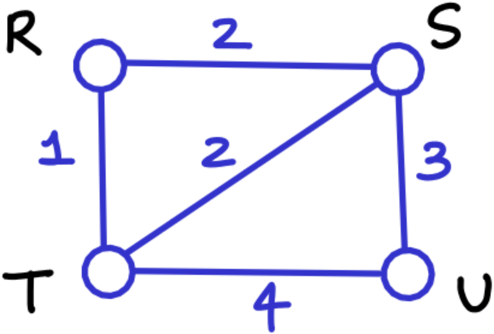


- Weight of a spanning tree:
  $$w(T) = \sum_{(u,v)\in T} w(u,v)$$
- Minimum spanning tree is a spanning tree with minimum weight.

Minimum Spanning Tree

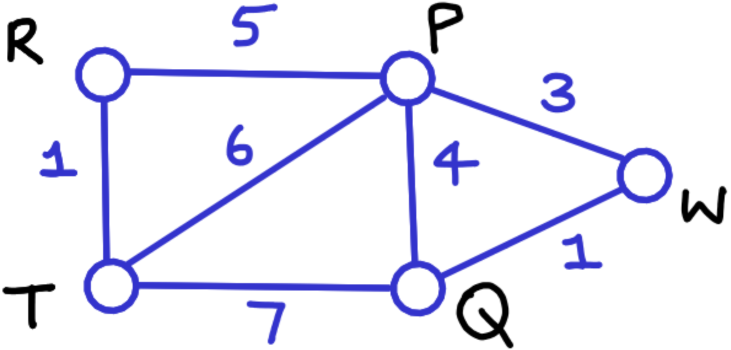


Minimum Spanning Tree



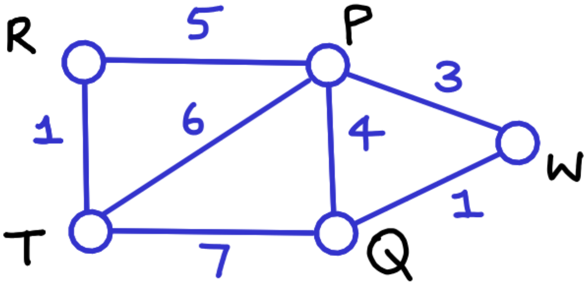
Contains $|V| - 1$ edges.

Minimum Spanning Tree

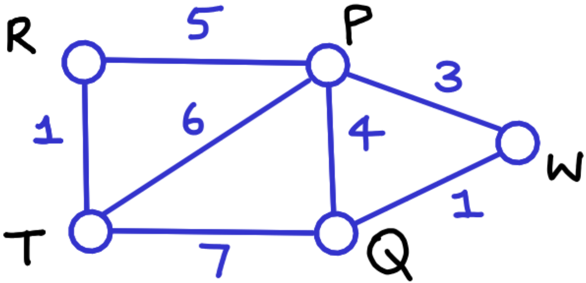


Contains $|V| - 1$ edges.

Generic method for find the MST

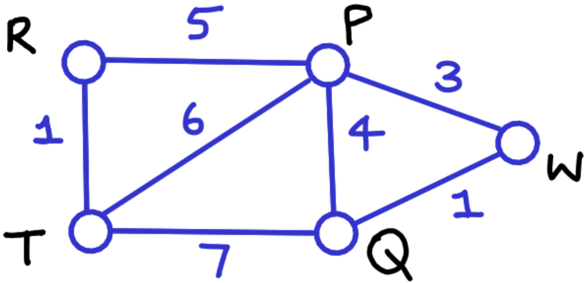


Generic method for find the MST



- Generic method manages a set of edges A, which is initially empty.

Generic method for find the MST



- Generic method manages a set of edges A, which is initially empty.
- **Invariant**: Prior to each iteration, A is a subset of some minimum spanning tree.

Generic Algorithm

- **Safe edge** : if $A \cup \{(u, v)\}$ is a subset of some minimum spanning tree.

Generic Algorithm

- **Safe edge** : if $A \cup \{(u, v)\}$ is a subset of some minimum spanning tree.

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
```

Generic Algorithm

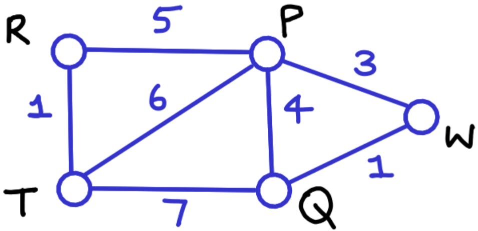
- **Safe edge** : if $A \cup \{(u, v)\}$ is a subset of some minimum spanning tree.

GENERIC-MST(G, w)

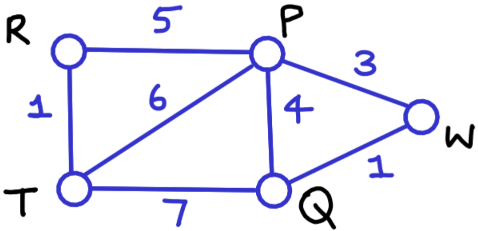
```
A = ∅
while A does not form a spanning tree
    find an edge (u, v) that is safe for A
    A = A ∪ {(u, v)}
return A
```

- How to find a safe edge?

Some terminologies

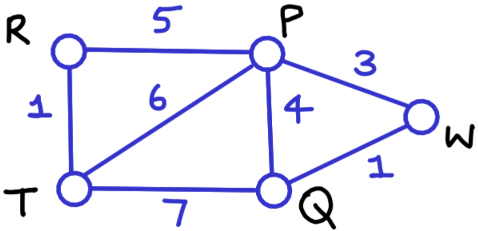


Some terminologies



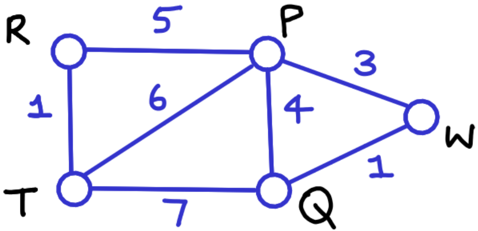
- A **cut** $(S, V - S)$ of an undirected graph $G = (V, E)$ is a partition of V.

Some terminologies



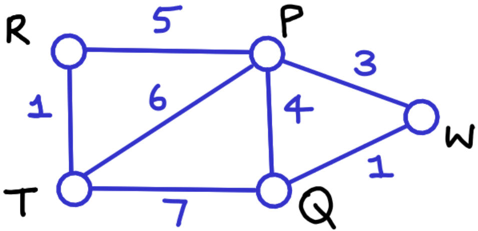
- A **cut** $(S, V - S)$ of an undirected graph $G = (V, E)$ is a partition of V. e.g. $(\{R, Q\}, \{T, P, W\})$

Some terminologies



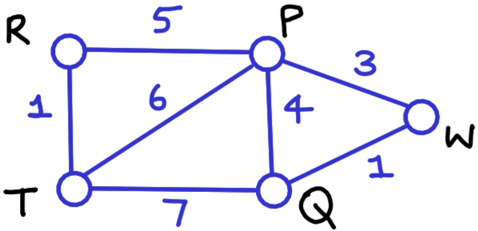
- A **cut** $(S, V - S)$ of an undirected graph $G = (V, E)$ is a partition of V. e.g. $(\{R, Q\}, \{T, P, W\})$
- An edge (u, v) **crosses** the cut $(S, V - S)$ if one of its endpoints is in S and the other is in $V - S$.

Some terminologies



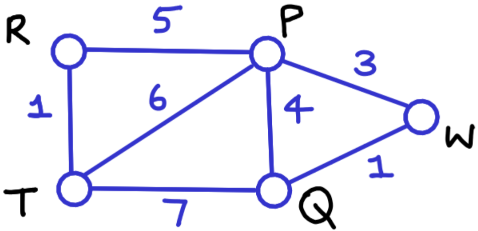
- A **cut** $(S, V - S)$ of an undirected graph $G = (V, E)$ is a partition of V. e.g. $(\{R, Q\}, \{T, P, W\})$
- An edge (u, v) **crosses** the cut $(S, V - S)$ if one of its endpoints is in S and the other is in $V - S$.
- Find an edge that crosses the cut $(\{R, Q\}, \{T, P, W\})$.

Some terminologies



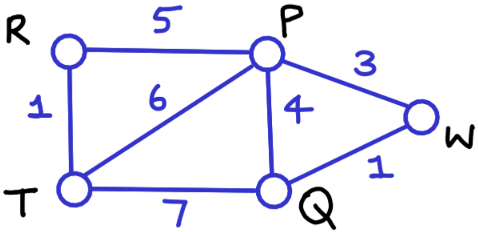
- A cut $(S, V - S)$ **respects** a set A of edges if no edge in A crosses the cut.

Some terminologies



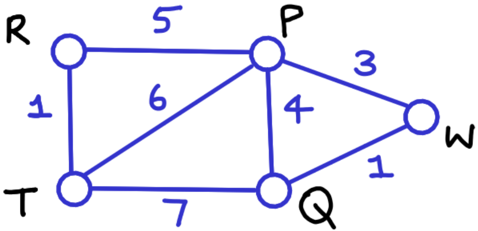
- A cut $(S, V - S)$ **respects** a set A of edges if no edge in A crosses the cut.
- Find a cut that respects $A = \{(R, T), (Q, W), (P, W)\}$?

Some terminologies



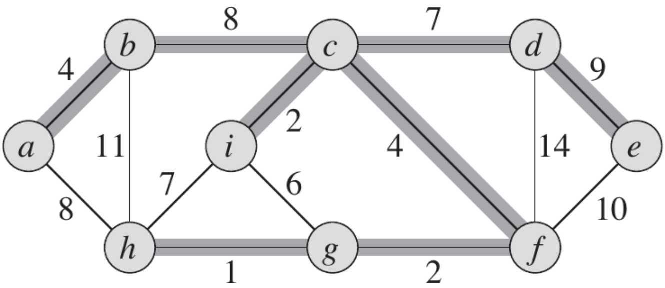
- An edge is a **light edge** crossing a cut if its weight is minimum among any edge crossing the cut.

Some terminologies



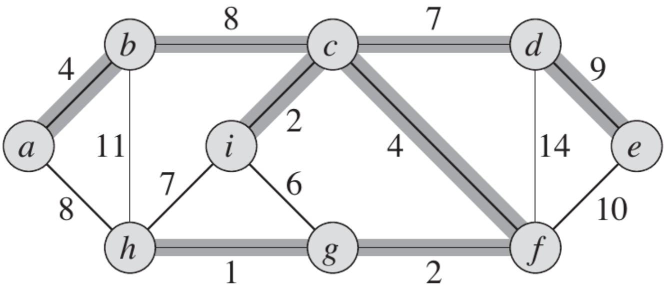
- An edge is a **light edge** crossing a cut if its weight is minimum among any edge crossing the cut.
- Find the light edge for the cut $(\{R, T\}, \{P, Q, W\})$?

Adding an edge to MST creates a cycle



- Adding an edge to a MST creates a cycle.

Adding an edge to MST creates a cycle



- Adding an edge to a MST creates a cycle.
- We can find another spanning tree by breaking the cycle.

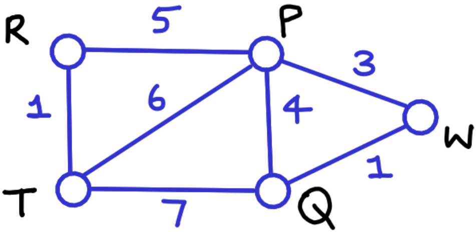
Generic Algorithm

- How to find a safe edge?

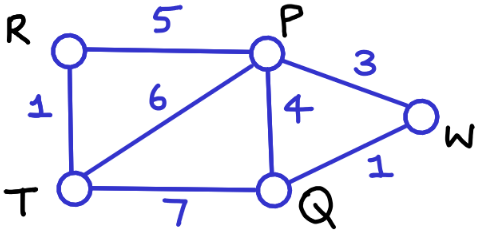
```
GENERIC-MST(G, w)
A = Ø
while A does not form a spanning tree
    find an edge (u, v) that is safe for A
    A = A ∪ {(u, v)}
return A
```

- **Theorem:** Let A be a set of edges which are included in some minimum spanning tree. Let $(S, V - S)$ be any cut that respects A, and let (u, v) be a light edge crossing $(S, V - S)$. Then, edge (u, v) is safe for A.

1. Set A is empty

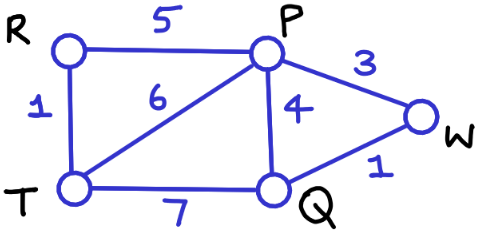


1. Set A is empty



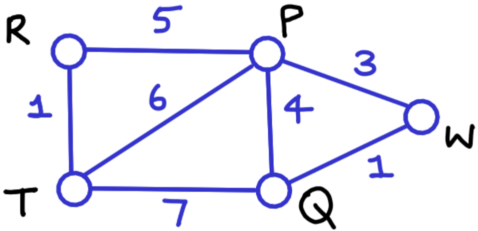
- Let $A = \{\}$.

1. Set A is empty



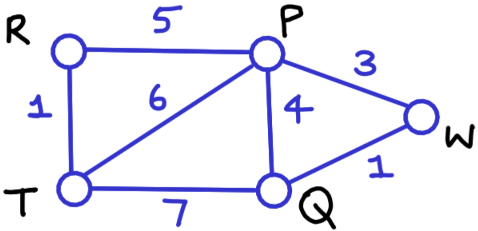
- Let $A = \{\}$.
- Find any cut $(S, V - S)$ that respects A.

1. Set A is empty



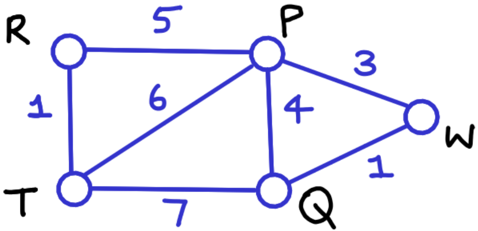
- Let $A = \{\}$.
- Find any cut $(S, V - S)$ that respects A.
  Let us say $(\{R, T, P\}, \{Q, W\})$

1. Set A is empty



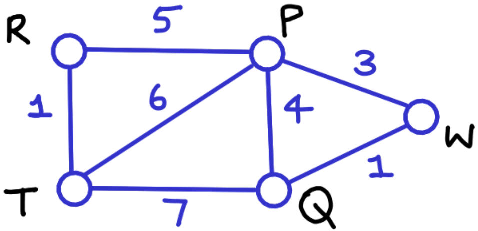
- Let $A = \{\}$.
- Find any cut $(S, V - S)$ that respects A.
  Let us say $(\{R, T, P\}, \{Q, W\})$
- Find a light edge crossing the cut $(S, V - S)$.

1. Set A is empty



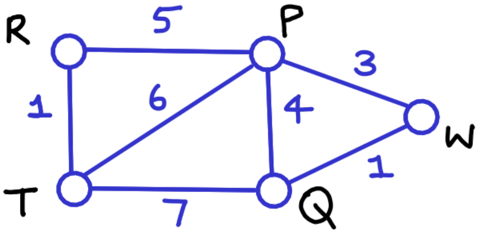
- Let $A = \{\}$.
- Find any cut $(S, V - S)$ that respects A.
  Let us say $(\{R, T, P\}, \{Q, W\})$
- Find a light edge crossing the cut $(S, V - S)$. Edge (P, W)

1. Set A is empty

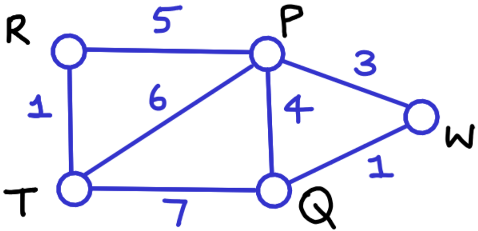


- Let $A = \{\}$.
- Find any cut $(S, V - S)$ that respects A.
  Let us say $(\{R, T, P\}, \{Q, W\})$
- Find a light edge crossing the cut $(S, V - S)$. Edge (P, W)
- The light edge (P, W) will be a safe edge.

2. Set A is non-empty

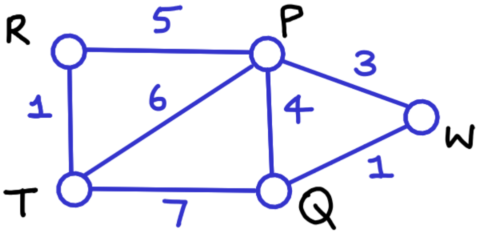


2. Set A is non-empty



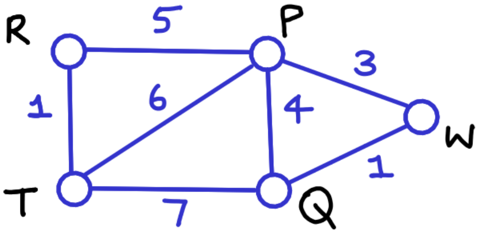
- $A = \{(P, W)\}$.

2. Set A is non-empty



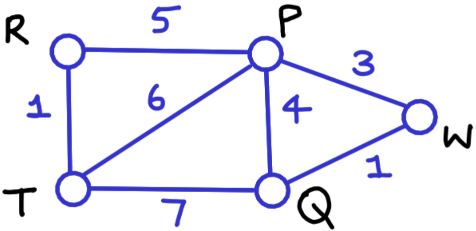
- $A = \{(P, W)\}$.
- Find any cut $(S, V - S)$ that respects A.

2. Set A is non-empty



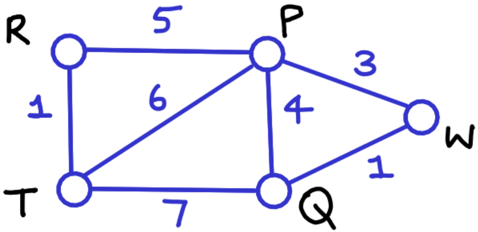
- $A = \{(P, W)\}$.
- Find any cut $(S, V - S)$ that respects A. Let us say $(\{R, P, W\}, \{T, Q\})$

2. Set A is non-empty



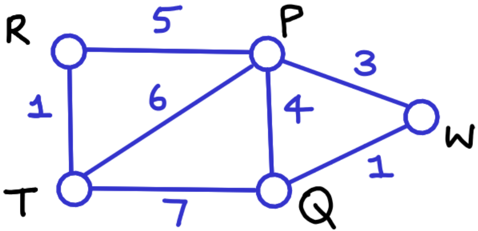
- $A = \{(P, W)\}$.
- Find any cut $(S, V - S)$ that respects A.
  Let us say $(\{R, P, W\}, \{T, Q\})$
- Find a light edge crossing the cut $(S, V - S)$.

2. Set A is non-empty



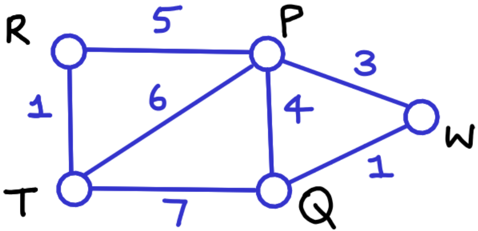
- $A = \{(P, W)\}$.
- Find any cut $(S, V - S)$ that respects A.
  Let us say $(\{R, P, W\}, \{T, Q\})$
- Find a light edge crossing the cut $(S, V - S)$. Edge (R, T)

2. Set A is non-empty

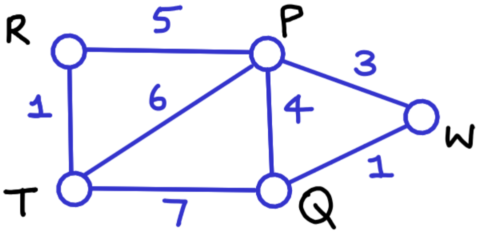


- $A = \{(P, W)\}$.
- Find any cut $(S, V - S)$ that respects A.
  Let us say $(\{R, P, W\}, \{T, Q\})$
- Find a light edge crossing the cut $(S, V - S)$. Edge (R, T)
- The light edge (R, T) will be a safe edge.

3. Set A is non-empty

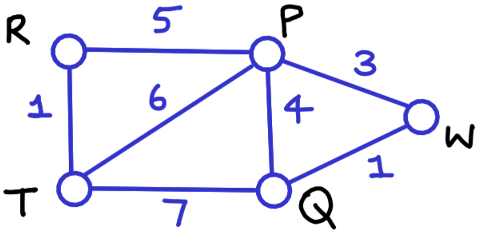


3. Set A is non-empty



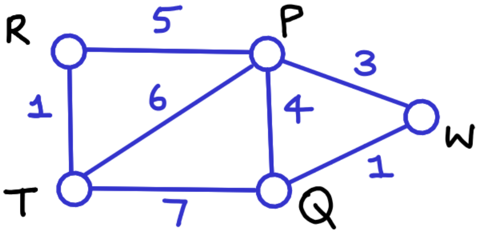
- $A = \{(P, W), (R, T)\}$.

3. Set A is non-empty



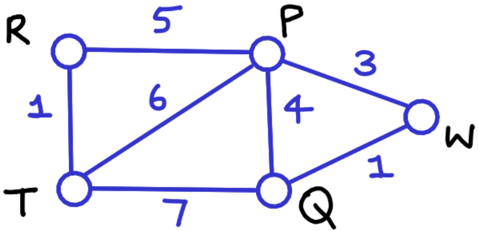
- $A = \{(P, W), (R, T)\}$.
- Find any cut $(S, V - S)$ that respects A.

3. Set A is non-empty



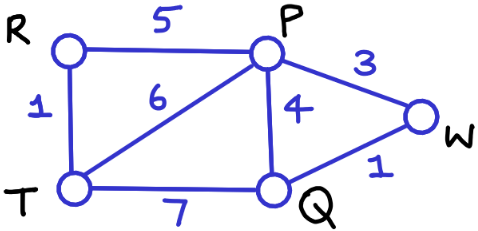
- $A = \{(P, W), (R, T)\}$.
- Find any cut $(S, V - S)$ that respects A. Let us say $(\{R, T\}, \{P, Q, W\})$

3. Set A is non-empty



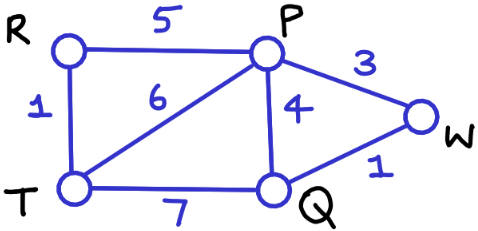
- $A = \{(P, W), (R, T)\}$.
- Find any cut $(S, V - S)$ that respects A.
  Let us say $(\{R, T\}, \{P, Q, W\})$
- Find a light edge crossing the cut $(S, V - S)$.

3. Set A is non-empty



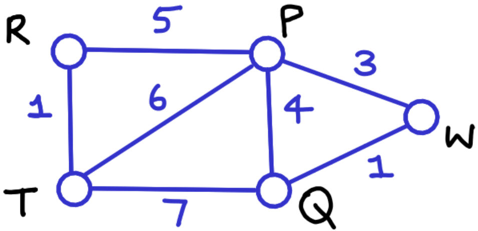
- $A = \{(P, W), (R, T)\}$.
- Find any cut $(S, V - S)$ that respects A.
  Let us say $(\{R, T\}, \{P, Q, W\})$
- Find a light edge crossing the cut $(S, V - S)$. Edge (R, P)

3. Set A is non-empty

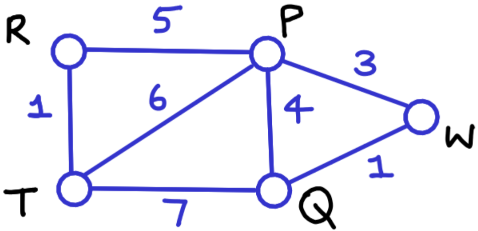


- $A = \{(P, W), (R, T)\}$.
- Find any cut $(S, V - S)$ that respects A.
  Let us say $(\{R, T\}, \{P, Q, W\})$
- Find a light edge crossing the cut $(S, V - S)$. Edge (R, P)
- The light edge (R, P) will be a safe edge.

4. Set A is non-empty

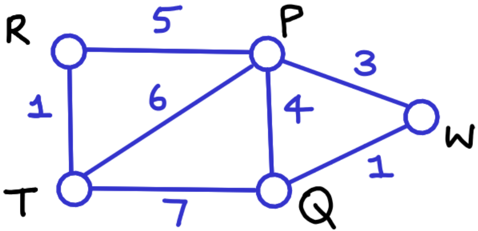


4. Set A is non-empty



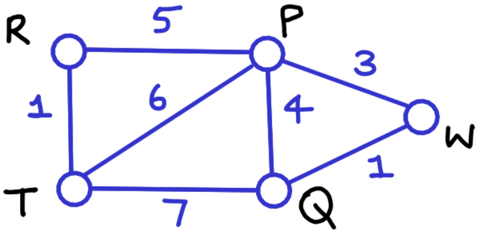
- $A = \{(P, W), (R, T), (R, P)\}$.

4. Set A is non-empty



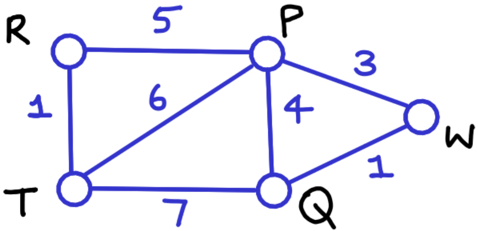
- $A = \{(P, W), (R, T), (R, P)\}$.
- Find a cut $(S, V - S)$ that respects A.

4. Set A is non-empty



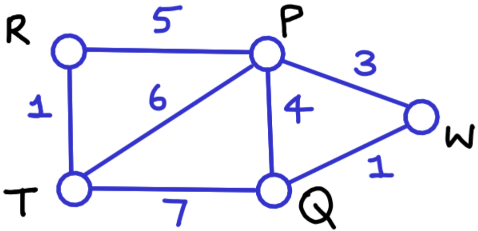
- $A = \{(P, W), (R, T), (R, P)\}$.
- Find a cut $(S, V - S)$ that respects A.
  Only possibility $(\{R, T, P, W\}, \{Q\})$

4. Set A is non-empty



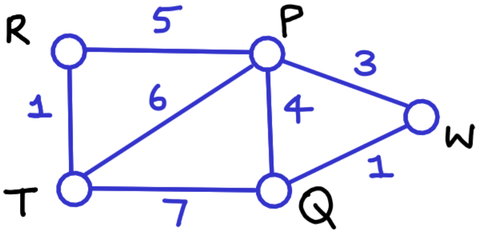
- $A = \{(P, W), (R, T), (R, P)\}$.
- Find a cut $(S, V - S)$ that respects A.
  Only possibility $(\{R, T, P, W\}, \{Q\})$
- Find a light edge crossing the cut $(S, V - S)$.

4. Set A is non-empty



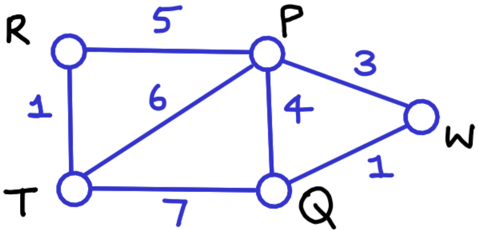
- $A = \{(P, W), (R, T), (R, P)\}$.
- Find a cut $(S, V - S)$ that respects A.
  Only possibility $(\{R, T, P, W\}, \{Q\})$
- Find a light edge crossing the cut $(S, V - S)$. Edge (Q, W)

4. Set A is non-empty

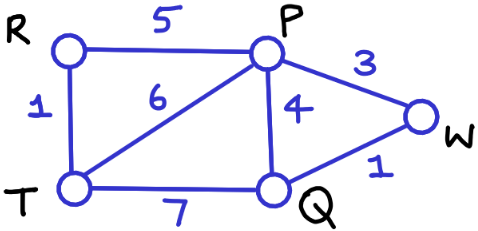


- $A = \{(P, W), (R, T), (R, P)\}$.
- Find a cut $(S, V - S)$ that respects A.
  Only possibility $(\{R, T, P, W\}, \{Q\})$
- Find a light edge crossing the cut $(S, V - S)$. Edge (Q, W)
- The light edge (Q, W) will be a safe edge.

5. Set A is non-empty

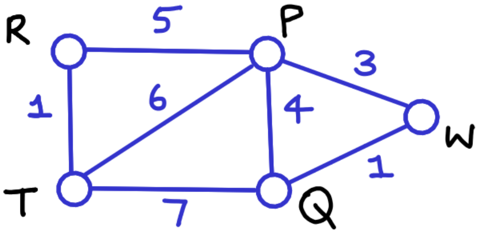


5. Set A is non-empty



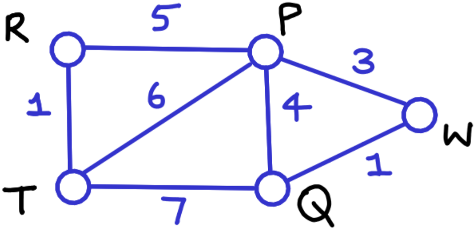
- $A = \{(P, W), (R, T), (R, P), (Q, W)\}$.

5. Set A is non-empty



- $A = \{(P,W),(R,T),(R,P),(Q,W)\}$.
- Set A forms a spanning tree. The generic algorithm terminates

5. Set A is non-empty



- $A = \{(P, W), (R, T), (R, P), (Q, W)\}$.
- Set A forms a spanning tree. The generic algorithm terminates

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
```

Theorem

- **Theorem:** Let A be a set of edges which are included in some minimum spanning tree. Let $(S, V - S)$ be any cut that respects A, and let (u, v) be a light edge crossing $(S, V - S)$. Then, edge (u, v) is safe for A.

Corollary

- **Corollary:** Let A be a set of edges which are included in some minimum spanning tree. 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.

Corollary

- Let $A = \{(R, T), (P, W)\}$.

Corollary

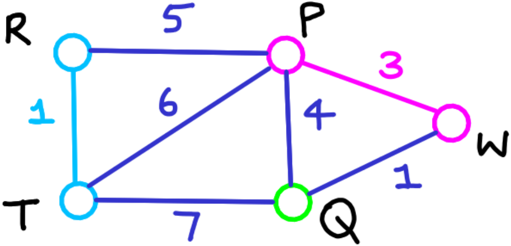
- Let $A = \{(R, T), (P, W)\}$.
- $G_A = (V, A)$ is a forest containing three trees.

Corollary

- Let $A = \{(R, T), (P, W)\}$.
- $G_A = (V, A)$ is a forest containing three trees.
- Let C be the tree containing the vertices P and W.

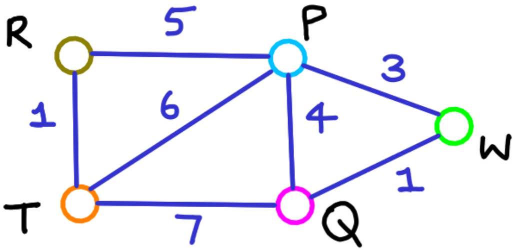
Corollary

- Let $A = \{(R, T), (P, W)\}$.
- $G_A = (V, A)$ is a forest containing three trees.
- Let C be the tree containing the vertices P and W.



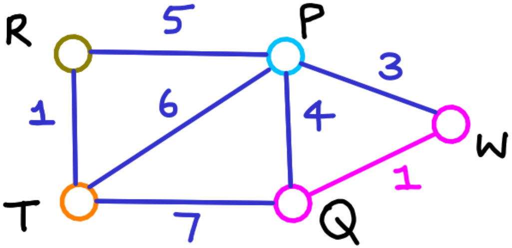
Kruskal's Algorithm : Step 1

- $A = \{\}$
- G_A contains 5 trees each having one vertex.



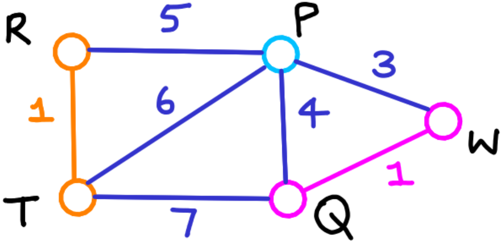
Kruskal's Algorithm : Step 2

- $A = \{(Q, W)\}$
- G_A contains 4 trees.



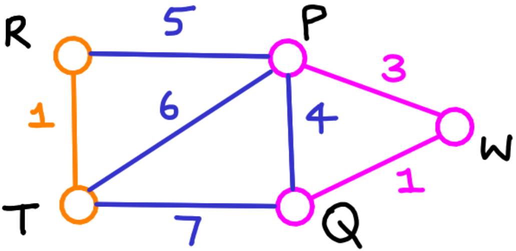
Kruskal's Algorithm : Step 3

- $A = \{(Q, W), (R, T)\}$
- G_A contains 3 trees.



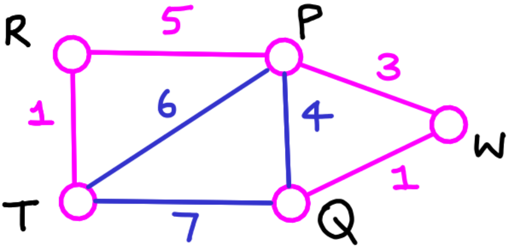
Kruskal's Algorithm : Step 4

- $A = \{(Q, W), (R, T), (P, W)\}$
- G_A contains 2 trees.



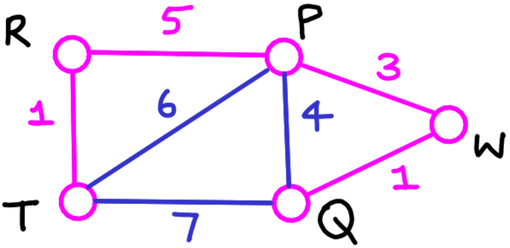
Kruskal's Algorithm : Step 5

- $A = \{(Q, W), (R, T), (P, W), (R, P)\}$
- G_A contains 1 tree.



Kruskal's Algorithm : Step 5

- $A = \{(Q, W), (R, T), (P, W), (R, P)\}$
- G_A contains 1 tree.

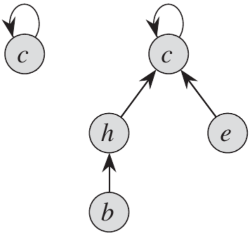


- If we keep considering more edges, it will always connect the same component.

Kruskal's 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
```

Disjoint forest data structure



MAKE-SET(X)

UNION(X,Y)

FIND-SET(X)

Running time for Kruskal's 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
```

Prim's Algorithm

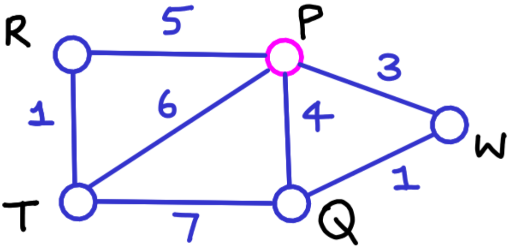
- Edges in set A always form a single component in the forest G_A.

Prim's Algorithm

- Edges in set A always form a single component in the forest G_A.
- We start growing the tree from an initial root vertex r.

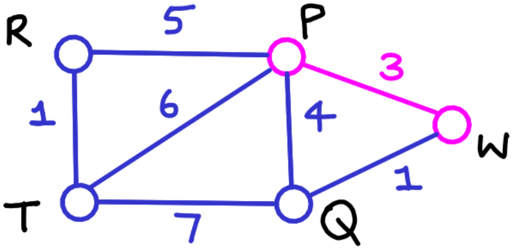
Prim's Algorithm : Step 1

- $A = \{\}$



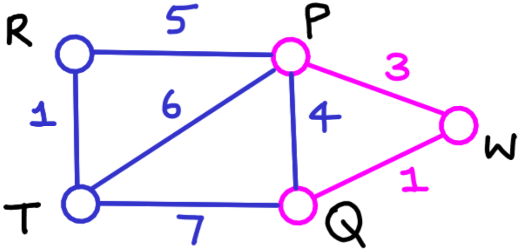
Prim's Algorithm : Step 2

- $A = \{(P, W)\}$



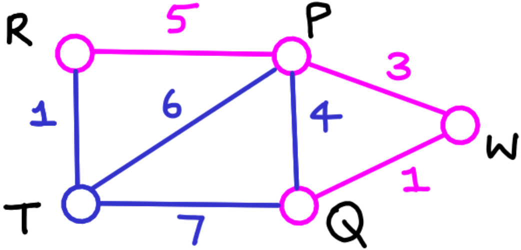
Prim's Algorithm : Step 3

- $A = \{(P, W), (Q, W)\}$



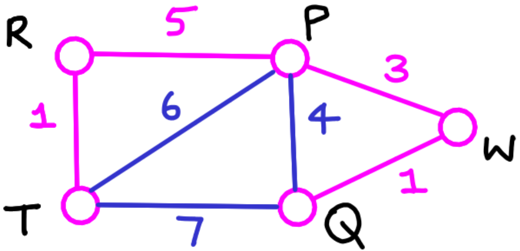
Prim's Algorithm : Step 4

- $A = \{(P, W), (Q, W), (R, P)\}$



Prim's Algorithm : Step 5

- $A = \{(P, W), (Q, W), (R, P), (R, T)\}$



Prim's Algorithm

```
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)
```

Max-priority Queue / Min-priority Queue

- Build-Min-Heap

Max-priority Queue / Min-priority Queue

- BUILD-MIN-HEAP
- HEAP-EXTRACT-MIN

Max-priority Queue / Min-priority Queue

- BUILD-MIN-HEAP
- HEAP-EXTRACT-MIN
- HEAP-DECREASE-KEY

Running time of Prim's Algorithm

```
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)
```

Running time of Prim's Algorithm