Lecture 13

Instructor: Subrahmanyam Kalyanasundaram

30th September 2019

Plan

- Minimum spanning trees
- Proof of Correctness of Kruskal's Algorithm
- Union-Find Data Structure
  (Disjoint Set Data Structure)

Spanning Tree

Definition: An undirected graph G is *connected* if every vertex is reachable from every other vertex.

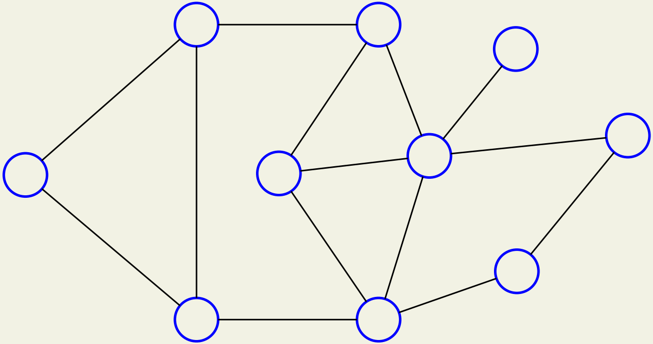
A graph $T = (V, E')$ is a spanning tree of an undirected connected graph $G = (V, E)$ if:

- $E' \subseteq E$.
- T is a *tree*. i.e., T is an acyclic and connected.

Informally: A spanning tree for G is a tree that can be found inside G which *spans* all vertices of G.

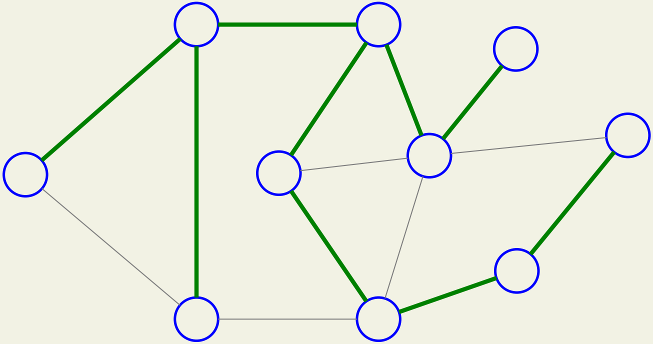
Examples

What are the possible spanning trees for this graph?



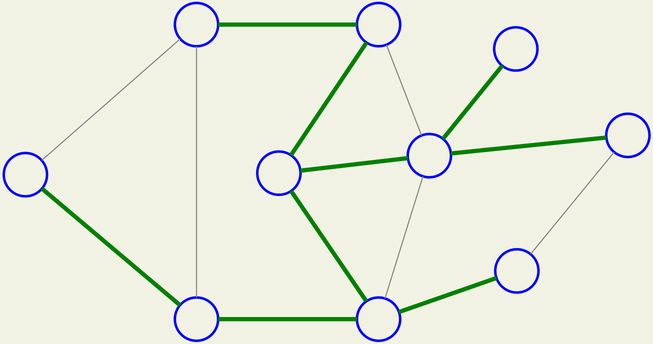
Examples

Is this a spanning tree?



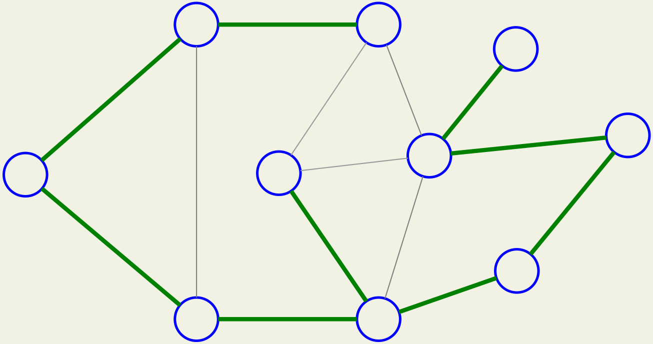
Examples

Is this a spanning tree?



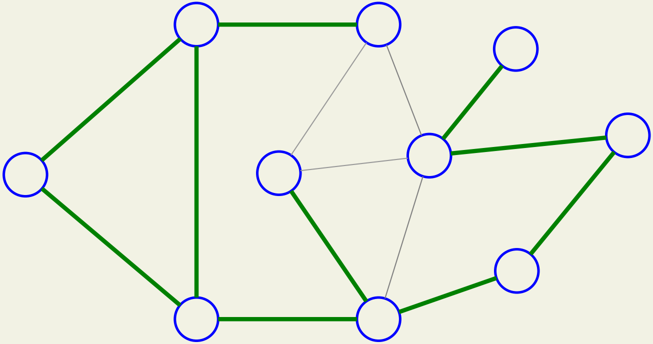
Examples

Is this a spanning tree?



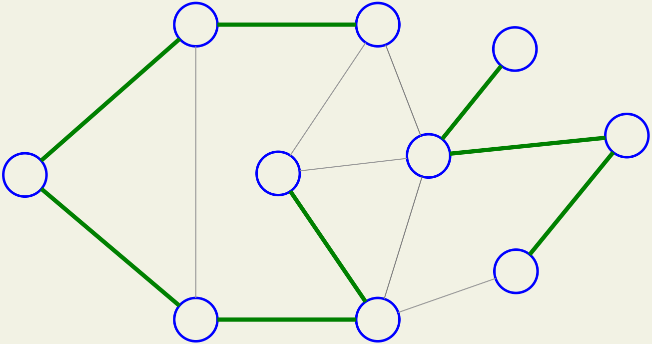
Examples

Is this a spanning tree?



Examples

Is this a spanning tree?



Minimum Spanning Tree Problem

Input

- Undirected connected graph $G = (V, E)$
- Weight function $w : E \to \mathbb{Z}^+$

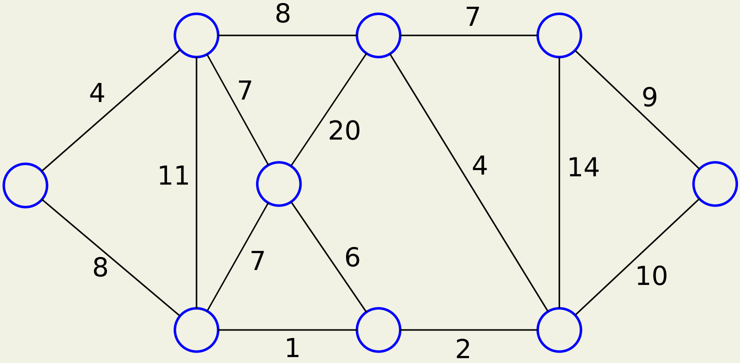
Goal

Compute a spanning tree for G with minimum total weight.

Kruskal's Algorithm (informal)

- Sort the edges in nondecreasing order by weight
- Set $T = \emptyset$
- Choose the lightest edge and add it to T as long as it does not create a cycle in T
- Terminate when T is spanning

Kruskal’s algorthm example

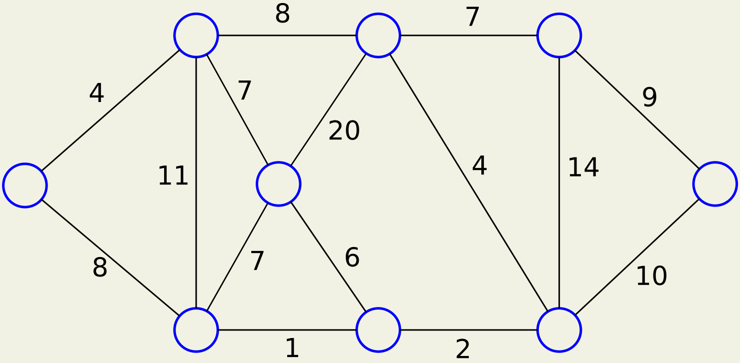


Kruskal's Algorithm Pseudocode

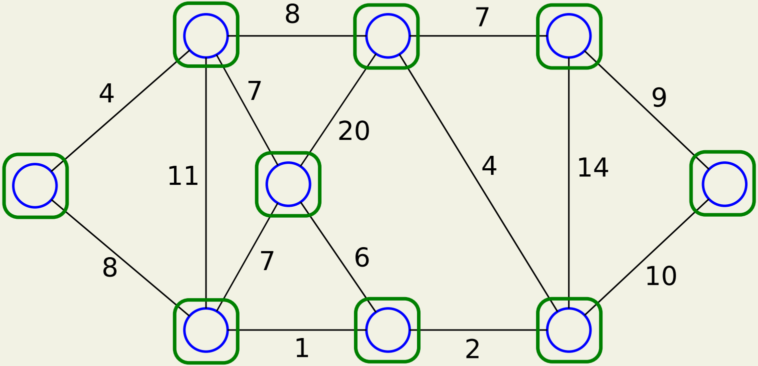
Algorithm 1 Kruskal's algorithm

```
A = ∅
for each vertex v ∈ V do
    MAKE-SET(v)
end for
Sort the edges in E into nondecreasing order by weight w
for each edge (u, v) ∈ E taken in nondecreasing order by weight do
    if FIND-SET(u) ≠ FIND-SET(v) then
        A = A ∪ {(u, v)}
        UNION(u, v)
    end if
end for
Return A
```

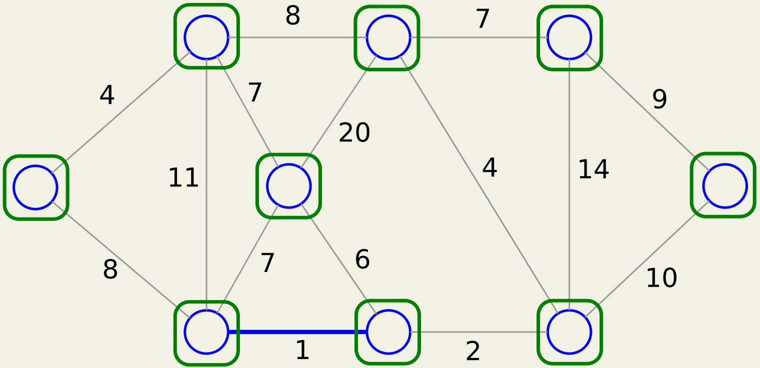
Kruskal’s algorthm example



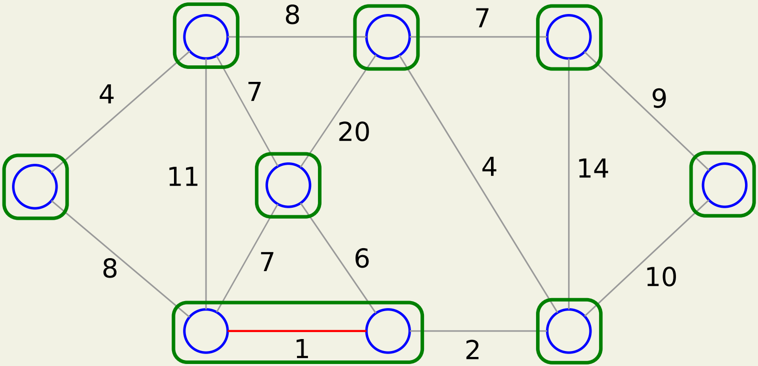
Kruskal's algorthm example



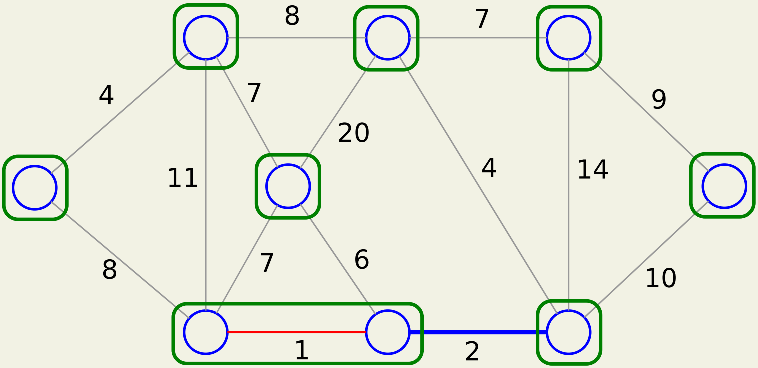
Kruskal’s algorthm example



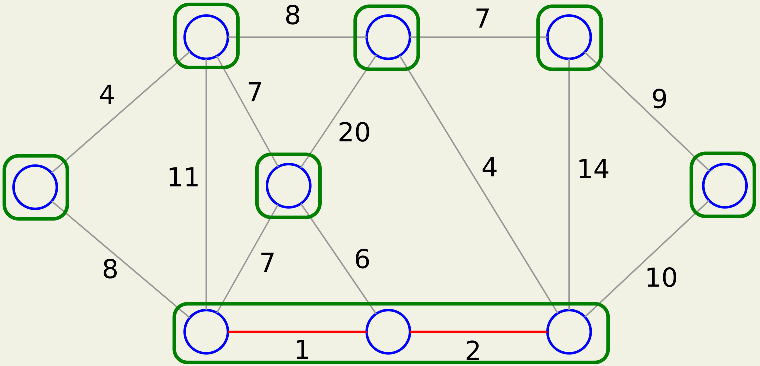
Kruskal’s algorthm example



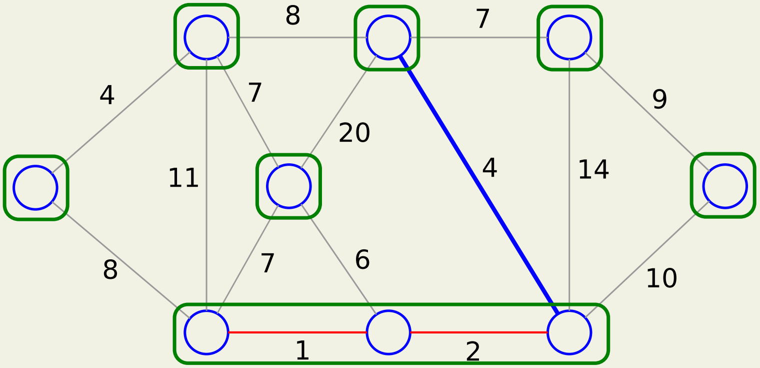
Kruskal’s algorthm example



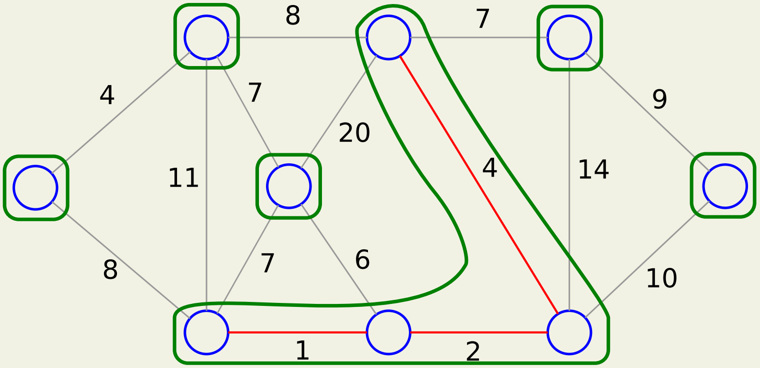
Kruskal’s algorthm example



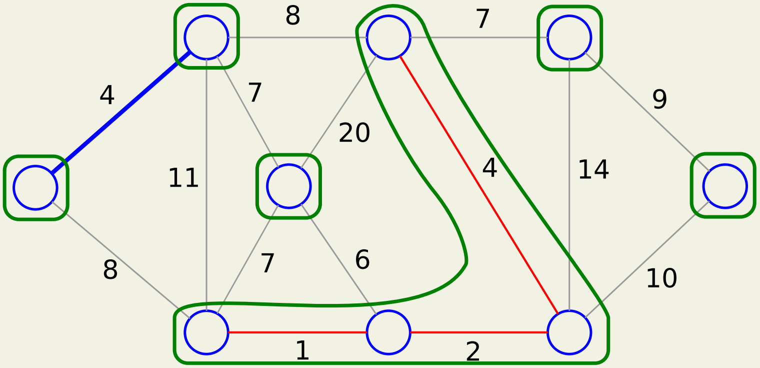
Kruskal’s algorthm example



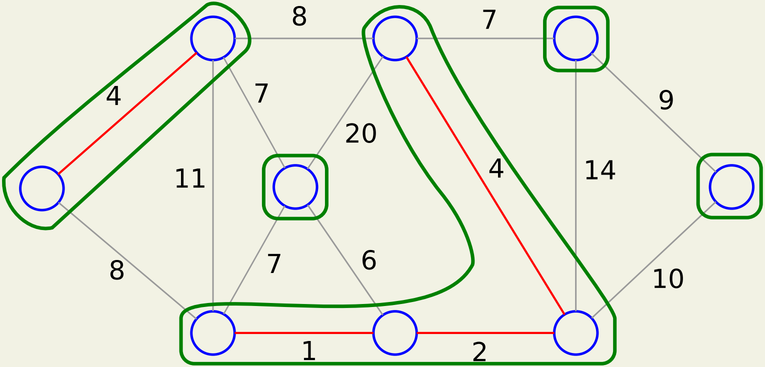
Kruskal’s algorthm example



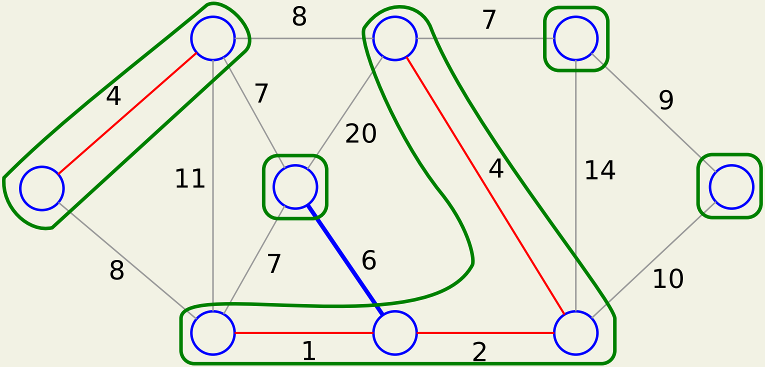
Kruskal's algorthm example



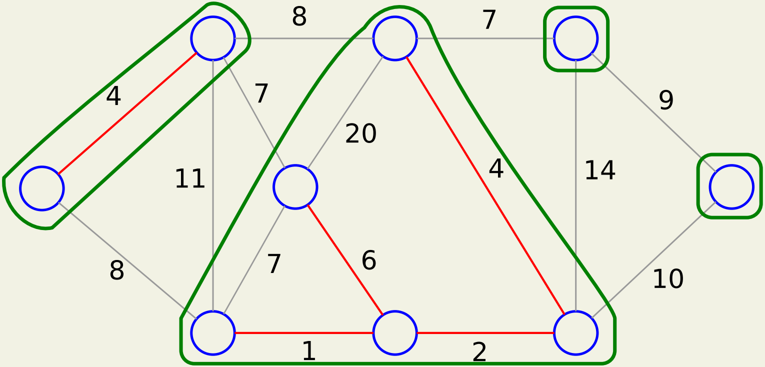
Kruskal's algorthm example



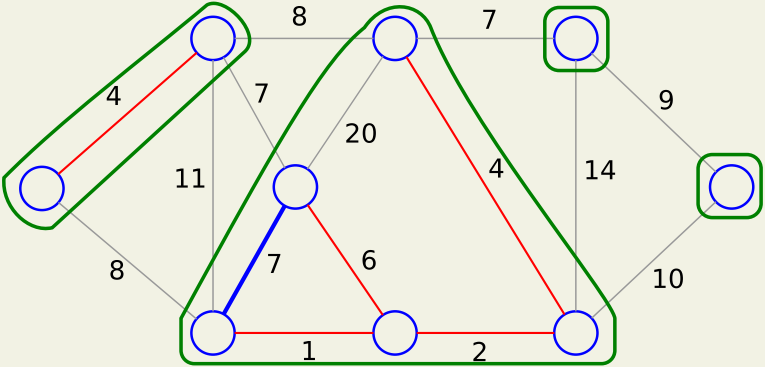
Kruskal's algorthm example



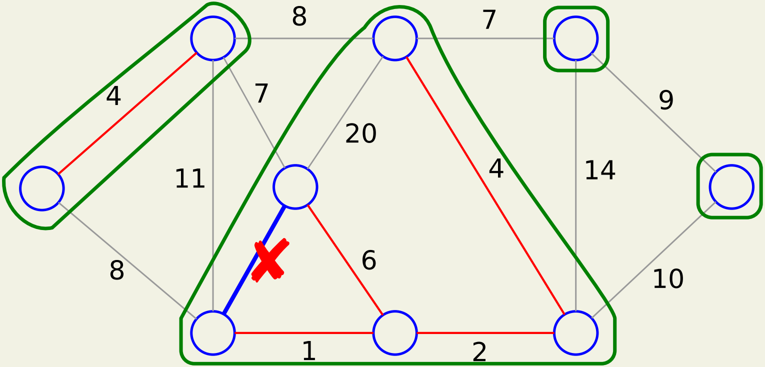
Kruskal's algorthm example



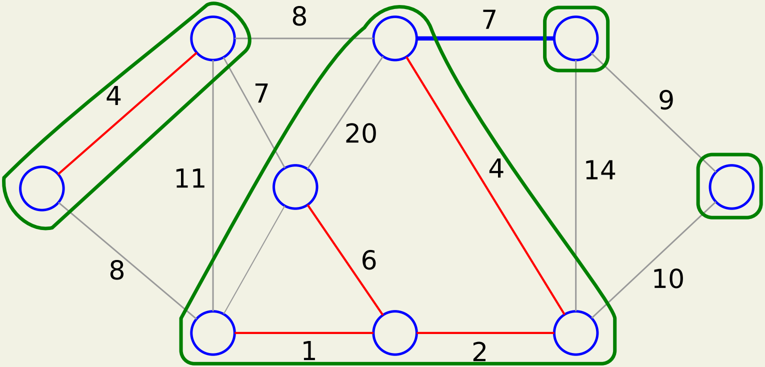
Kruskal’s algorthm example



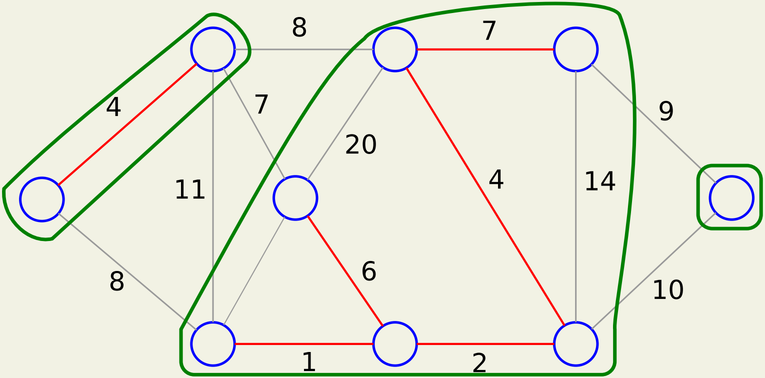
Kruskal's algorthm example



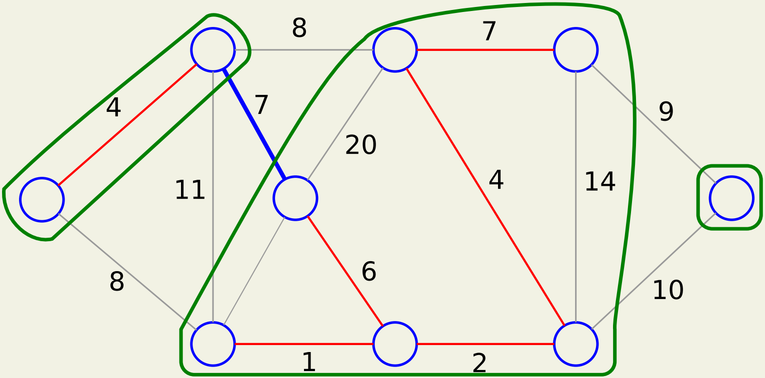
Kruskal's algorthm example



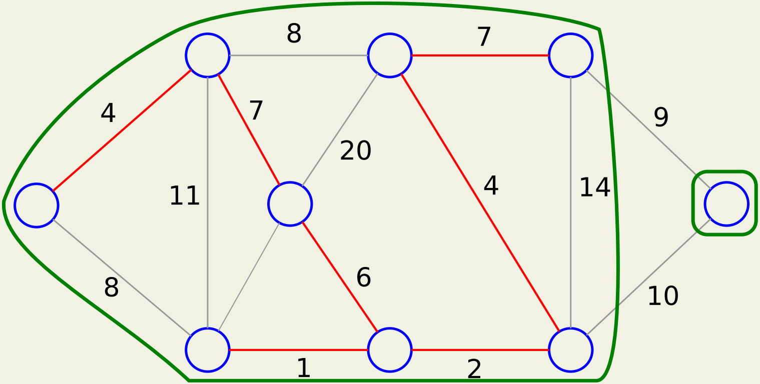
Kruskal’s algorthm example



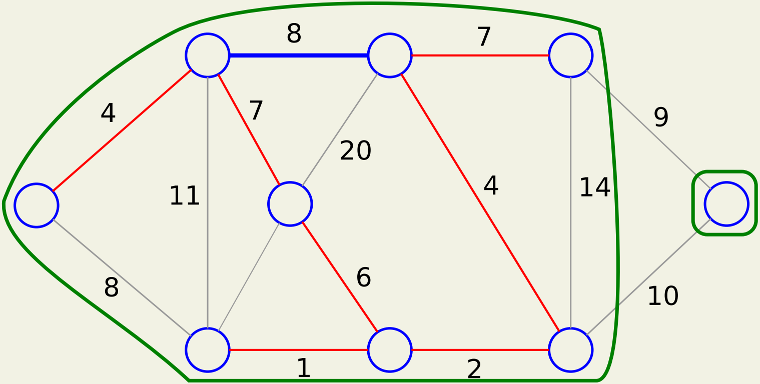
Kruskal's algorthm example



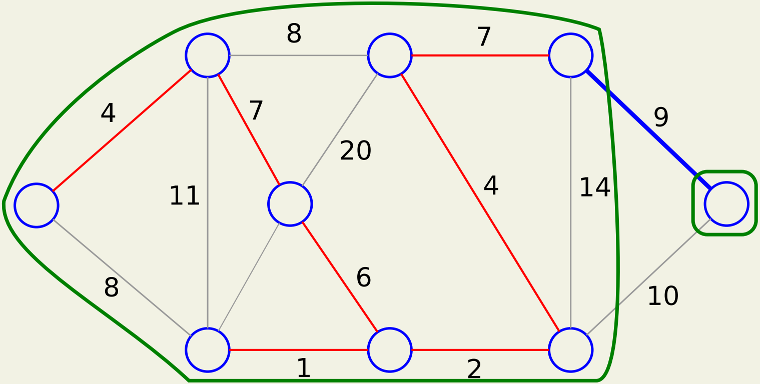
Kruskal's algorthm example



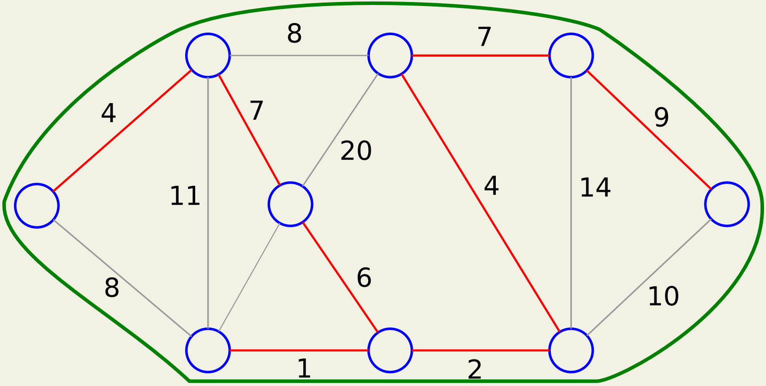
Kruskal's algorthm example



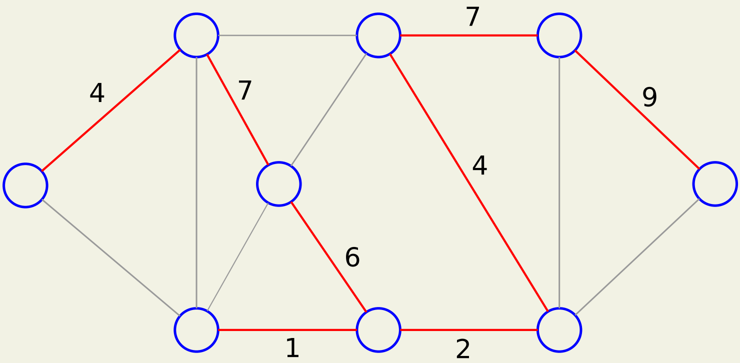
Kruskal's algorthm example



Kruskal’s algorthm example



Kruskal's algorthm example



Proof of Correctness of Kruskal's

Algorithm 2 Generic MST

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

Proof of Correctness of Kruskal's

Algorithm 3 Generic MST

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

- Kruskal's is a special case of the above generic algorithm
- (u, v) is safe for A: There is an MST that contains $A \cup \{(u, v)\}$
- How does one find a safe edge?

Proof of Correctness of Kruskal's

- (u, v) is safe for A: There is an MST that contains $A \cup \{(u, v)\}$
- The cut $(S, V - S)$ respects A: No edge in A crosses the cut $(S, V - S)$

Proof of Correctness of Kruskal's

- (u, v) is safe for A: There is an MST that contains $A \cup \{(u, v)\}$
- The cut $(S, V - S)$ respects A: No edge in A crosses the cut $(S, V - S)$

Theorem (Cut Property)

Let $G = (V, E)$ be a connected undirected graph with weight $w : E \to \mathbb{R}$. Let $A \subseteq E$ be included in some MST of G. Let $(S, V - S)$ be a cut that respects A and let (u, v) be an edge of smallest weight that crosses this cut. Then (u, v) is safe for A.

Proof of Correctness of Kruskal's

Theorem (Cut Property)

Let $G = (V, E)$ be a connected undirected graph with weight $w : E \to \mathbb{R}$. Let $A \subseteq E$ be included in some MST of G. Let $(S, V - S)$ be a cut that respects A and let (u, v) be an edge of smallest weight that crosses this cut. Then (u, v) is safe for A.

Proof

Let T be an MST such that $A \subseteq T$. If $(u, v) \in T$, we are done. So assume $(u, v) \notin T$.
We will show that there is an MST T' such that $A \cup \{(u, v)\} \subseteq T'$. Thus (u, v) is safe for A.

Proof of Correctness of Kruskal's

Proof cont...

Consider $T \cup \{(u, v)\}$. There is a unique path p from u to v in T. When (u, v) is added to T, it forms a cycle along with the path p.

Proof of Correctness of Kruskal's

Proof cont...

Consider $T \cup \{(u, v)\}$. There is a unique path p from u to v in T. When (u, v) is added to T, it forms a cycle along with the path p.

Since u and v are on the opposite sides of the cut $(S, V - S)$, at least one of the edges in p crosses the cut $(S, V - S)$. Let (x, y) be such an edge. Since $(S, V - S)$ respects A, $(x, y) \notin A$. Removing (x, y) breaks T into two components.

Proof of Correctness of Kruskal's

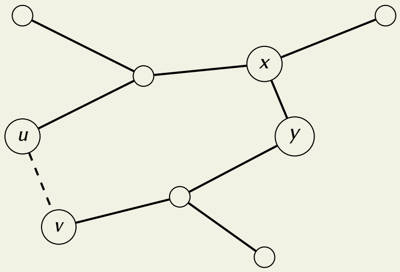
Proof cont...

Consider $T \cup \{(u, v)\}$. There is a unique path p from u to v in T. When (u, v) is added to T, it forms a cycle along with the path p.

Since u and v are on the opposite sides of the cut $(S, V - S)$, at least one of the edges in p crosses the cut $(S, V - S)$. Let (x, y) be such an edge. Since $(S, V - S)$ respects A, $(x, y) \notin A$. Removing (x, y) breaks T into two components.

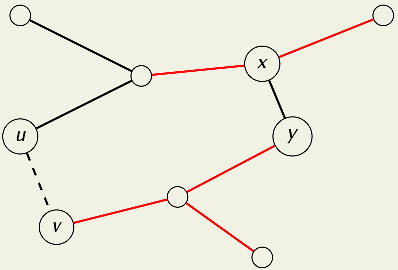
Consider $T' = T - \{(x, y)\} \cup \{(u, v)\}$. Adding (u, v) to $T - \{(x, y)\}$, reconnects the two components. Hence T' is a spanning tree.

Proof of Correctness of Kruskal's



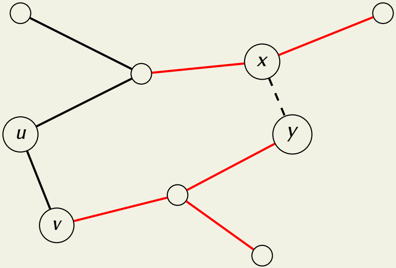
- Solid edges are in T

Proof of Correctness of Kruskal's



- Solid edges are in T
- Red edges are in A

Proof of Correctness of Kruskal's



- Solid edges are in T
- Red edges are in A
- (u, v) is safe!

Proof of Correctness of Kruskal's

Proof cont...

Consider $T' = T - \{(x, y)\} \cup \{(u, v)\}$. Adding (u, v) to $T - \{(x, y)\}$, reconnects the two components. Hence T' is a spanning tree.

Proof of Correctness of Kruskal's

Proof cont...

Consider $T' = T - \{(x,y)\} \cup \{(u,v)\}$. Adding (u,v) to $T - \{(x,y)\}$, reconnects the two components. Hence T' is a spanning tree.

$$\begin{aligned} w(T') &= w(T) - w((x,y)) + w((u,v)) \\ &\leq w(T) \qquad\qquad [\text{since } w((u,v)) \leq w((x,y))] \end{aligned}$$

Proof of Correctness of Kruskal's

Proof cont...

Consider $T' = T - \{(x, y)\} \cup \{(u, v)\}$. Adding (u, v) to $T - \{(x, y)\}$, reconnects the two components. Hence T' is a spanning tree.

$$\begin{aligned} w(T') &= w(T) - w((x, y)) + w((u, v)) \\ &\leq w(T) \qquad\qquad [\text{since } w((u, v)) \leq w((x, y))] \end{aligned}$$

Hence T' is a minimum spanning tree!

Kruskal's Algorithm Running Time

Algorithm 4 Kruskal's algorithm

```
A = ∅
for each vertex v ∈ V do
    MAKE-SET(v)
end for
Sort the edges in E into nondecreasing order by weight w
for each edge (u, v) ∈ E taken in nondecreasing order by weight do
    if FIND-SET(u) ≠ FIND-SET(v) then
        A = A ∪ {(u, v)}
        UNION(u, v)
    end if
end for
Return A
```

Kruskal's Algorithm Running Time

- $|V|$ Make-Set(v) operations
- At most $2|E|$ Find-Set(v) operations
- $|V| - 1$ Union(u, v) operations

Kruskal's Algorithm Running Time

- $|V|$ MAKE-SET(v) operations
- At most $2|E|$ FIND-SET(v) operations
- $|V| - 1$ UNION(u, v) operations

- And a sort the edges – takes $O(|E| \log |E|)$ time

Kruskal's Algorithm Running Time

- $|V|$ Make-Set(v) operations
- At most $2|E|$ Find-Set(v) operations
- $|V| - 1$ Union(u, v) operations

- And a sort the edges – takes $O(|E| \log |E|)$ time
- What if edges were already sorted? What if we can sort them in linear time?

Disjoint Set

Maintain a collection $\mathcal{F} = \{S_1, S_2, \ldots, S_k\}$ of disjoint sets.
One element from each set serves as a 'representative' for that set.

Disjoint Set supports the following procedures:

- $\textsc{MakeSet}(x)$ – Creates a singleton set with element x.
- $\textsc{Union}(x, y)$ – Performs union on sets containing x and y.
- $\textsc{FindSet}(x)$ – Find the set containing x.

MakeSet(x)

Creates a singleton set containing x.

We assume that x is not an element of any other set in $\mathcal{F}$.
We assume x as the representative for the set just created.

Union

Union(x, y)

Performs union on sets containing x and y.

Let $S, T \in \mathcal{F}$ such that $x \in S$ and $y \in T$.
Create a new set $U = S \cup T$.
Choose and assign a representative for U.
Remove S and T from $\mathcal{F}$.

FindSet(x)
Find the set containing x.

Let $S \in \mathcal{F}$ such that $x \in S$. (Note: exactly one set contains x.)
Return a pointer to the representative element of S.

Implementation

Disjoint Set using linked lists:

Implementation

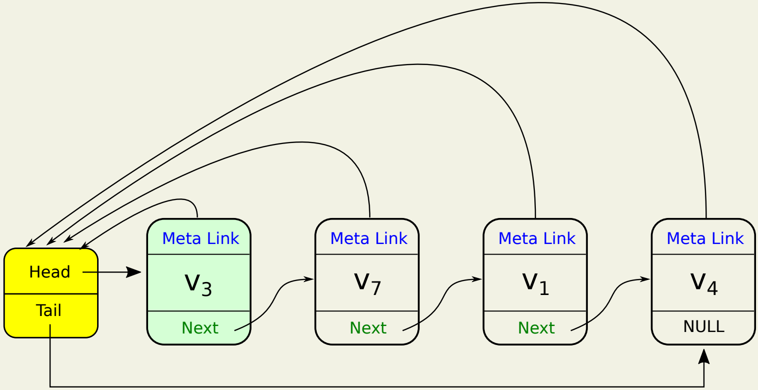
Disjoint Set using linked lists:

- For each set S, maintain:
  - a node with metadata
  - a linked list L_S with the objects in the set.
- The "Metadata Node" stores head and tail pointers to the linked list.
- Each node in the linked list consists of:
  - The value of the element.
  - A pointer to the next element.
  - A pointer to the Metadata Node.

The head of L_S is the representative of S.

Implementation

Linked list for set $\{v_1, v_3, v_4, v_7\}$.



Implementation

$\textsc{MakeSet}(x)$

Creates a singleton set with element x

- Create a new node for metadata
- Create a linked list containing just x.
- Node x is the head and tail of the list.
- Representative for this set is x itself.

Implementation

FindSet(x)

Find the set containing node x.

- Return a pointer to the representative.

Implementation

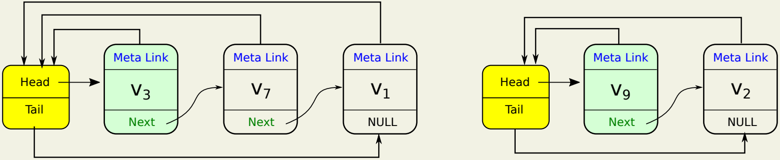
UNION(X,Y)

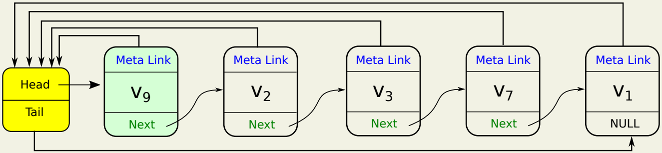
Union of sets containing x and y.

- Append linked list of set S containing x to set T containing y.
- Representative of new set is same as representative of T.
- Update meta pointers of nodes in S to the correct metadata node.
- Update tail pointer in metadata node of T.

Implementation

Union of sets $S = \{v_1, v_3, v_7\}$ and $T = \{v_2, v_9\}$.





Union(v_1,v_2)

Analysis

Running time under Linked List implementation:

- $\textsc{MakeSet}(x)$ – $O(1)$
- $\textsc{FindSet}(x)$ – $O(1)$
- $\textsc{Union}(x, y)$ – ?

Analysis

UNION(x, y) –

- $S \leftarrow$ FINDSET(x) and $T \leftarrow$ FINDSET(Y) – $O(1)$ time.
- Appending linked list of S to tail end of T – $O(1)$ time.
- Updating the new metadata (tail) – $O(1)$ time.
- Updating the backward pointers of nodes in S takes $O(n)$ time.

We can show a case where after $O(n)$ operations, time taken would be $O(n^2)$.

Recap: List Implementation

Disjoint Set using linked lists:

- For each set S, maintain:
  - a node with metadata
  - a linked list L_S with the objects in the set.
- The "Metadata Node" stores:
  - Head and tail pointers to the linked list.
- Each node in the linked list consists of:
  - The value of the element.
  - A pointer to the next element.
  - A pointer to the Metadata Node.

The head of L_S is the representative of S.

List Implementation - Union by Rank heuristic

Disjoint Set using linked lists, union by rank:

- For each set S, maintain:
  - a node with metadata
  - a linked list L_S with the objects in the set.
- The "Metadata Node" stores:
  - Head and tail pointers to the linked list.
  - Size of the set.
- Each node in the linked list consists of:
  - The value of the element.
  - A pointer to the next element.
  - A pointer to the Metadata Node.

The head of L_S is the representative of S.

List Implementation - Union by Rank heuristic

UNION(X,Y)

Union of sets containing x and y.

Let $x \in S$ and $y \in T$.

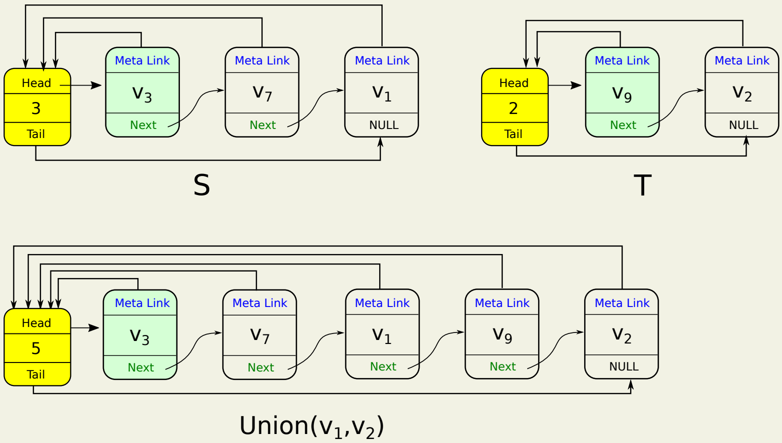
If $|S| \leq |T|$,

- Append list of S to tail end of list of T.
- Representative of new set is same as that of T.
- Update meta pointers of nodes in S
- Update tail pointer in metadata node of T.
- Update size of set in the metadata node.

Else, do the opposite.

Implementation - Union by Rank heuristic

Union of sets $S = \{v_1, v_3, v_7\}$ and $T = \{v_2, v_9\}$.



Analysis - Union by Rank heuristic

Theorem

A sequence of m operations in total, n of which are MakeSet takes $O(m + n \log n)$ time.

Analysis - Union by Rank heuristic

Observation 1

Updating the meta pointers takes the most time.

Observation 2

The meta pointer of a node x is updated only when union happens with a bigger set.

Proof strategy

- Fix an element x.
- Count number of times the meta pointer on node x is updated.

Analysis - Union by Rank heuristic

Observation 2 (informal)

If x lived inside a set of size s,
and a union operation updated its meta pointer,
then x now lives inside a set of size at least $2s$.

- Initially, x starts off as a singleton set.
- After k many updates to its meta pointer, it lives inside a set of size at least 2^k.
- Total number of elements is n. So $2^k \leq n$.
- This means $k \leq \log n$

Hence, for each element the meta pointer can be updated at most $k \leq \log n$ many times.
Worst case total number of updates to meta pointer across all n elements is $n \log n$.

□

Implementation - disjoint forests

The disjoint forest implementation:

- Each set S is implemented as a rooted tree.

A node corresponding to an $x \in S$ contains:

- The value (or pointer to) x.
- A pointer to its parent.

Implementation - disjoint forests

The disjoint forest implementation:

- Each set S is implemented as a rooted tree.

A node corresponding to an $x \in S$ contains:

- The value (or pointer to) x.
- A pointer to its parent.

Note:

- There are no pointers to children nodes!
- There is no dedicated metadata node for each set.
- Convention: Parent of root will be itself.
- Root node is also the representative.