CMPSC 465
Data Structures and Algorithms
Spring 2022

Instructor: Chunhao Wang

Greedy algorithms

Greedy algorithms

Warm-up

Greedy algorithms

Minimum Spanning Tree

Kurskal's Algorithm

def $\textsc{Kruskal\_MST}$*(undirected* $G = (V, E)$*, weights* $w = (w_e)_{e \in E}$*)***:**

Kurskal's Algorithm

def $\textsc{Kruskal\_MST}$*(undirected $G = (V, E)$, weights $w = (w_e)_{e \in E}$)***:**

Set $A := \{\,\}$;

Kurskal's Algorithm

def $\textsc{Kruskal\_MST}$*(undirected $G = (V, E)$, weights $w = (w_e)_{e \in E}$)*:

Set $A := \{\,\}$;

for $v \in V$:

Kurskal's Algorithm

def $\textsc{Kruskal\_MST}$*(undirected $G = (V, E)$, weights $w = (w_e)_{e \in E}$)*:

  Set $A := \{\}$;

  for $v \in V$:

    make_set(v);

- make_set(v): put v into a set containing itself. $v \mapsto \{v\}$

Kurskal's Algorithm

def $\textsc{Kruskal\_MST}$*(undirected $G = (V, E)$, weights $w = (w_e)_{e \in E}$)***:**

Set $A := \{\,\}$;

for $v \in V$**:**

  $\mathrm{make\_set}$(v);

Sort E in increasing order of edge weights;

- $\mathrm{make\_set}(v)$: put v into a set containing itself. $v \mapsto \{v\}$

Kurskal's Algorithm

def $\textsc{Kruskal\_MST}$*(undirected $G = (V, E)$, weights $w = (w_e)_{e \in E}$)*:

  Set $A := \{\,\}$;

  for $v \in V$:

    $\mathrm{make\_set}(\mathsf{v})$;

  Sort E in increasing order of edge weights;

  for $(u, v) \in E$:

- $\mathrm{make\_set}(v)$: put v into a set containing itself. $v \mapsto \{v\}$

Kurskal's Algorithm

def $\textsc{Kruskal\_MST}$*(undirected $G = (V, E)$, weights $w = (w_e)_{e \in E}$)*:

  Set $A := \{\,\}$;

  for $v \in V$:

    $\mathrm{make\_set}$(v);

  Sort E in increasing order of edge weights;

  for $(u, v) \in E$:

    if $\mathrm{find\_set}(u) \neq \mathrm{find\_set}(v)$:

- $\mathrm{make\_set}(v)$: put v into a set containing itself. $v \mapsto \{v\}$
- $\mathrm{find\_set}(u)$: find which set u belongs to

Kurskal's Algorithm

def $\textsc{Kruskal\_MST}$*(undirected $G = (V, E)$, weights $w = (w_e)_{e \in E}$)***:**

```
Set A := { };
for v ∈ V:
    make_set(v);
Sort E in increasing order of edge weights;
for (u, v) ∈ E:
    if find_set(u) ≠ find_set(v):
        A := A ∪ {(u, v)};
```

- $\mathrm{make\_set}(v)$: put v into a set containing itself. $v \mapsto \{v\}$
- $\mathrm{find\_set}(u)$: find which set u belongs to

Kurskal's Algorithm

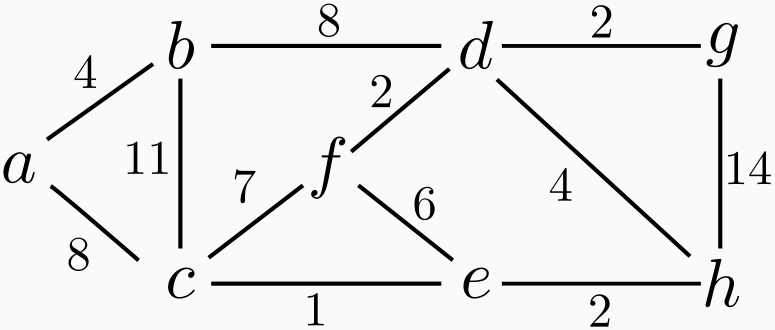
def $\textsc{Kruskal\_MST}$*(undirected $G = (V, E)$, weights $w = (w_e)_{e \in E}$)***:**

    Set $A := \{\}$;
    for $v \in V$**:**
        make_set(v);
    Sort E in increasing order of edge weights;
    for $(u, v) \in E$**:**
        if $\mathrm{find\_set}(u) \neq \mathrm{find\_set}(v)$**:**
            $A := A \cup \{(u, v)\}$;
            $\mathrm{union}(u, v)$;

- $\mathrm{make\_set}(v)$: put v into a set containing itself. $v \mapsto \{v\}$
- $\mathrm{find\_set}(u)$: find which set u belongs to
- $\mathrm{union}(u, v)$: merge the sets that u and v are in

Example

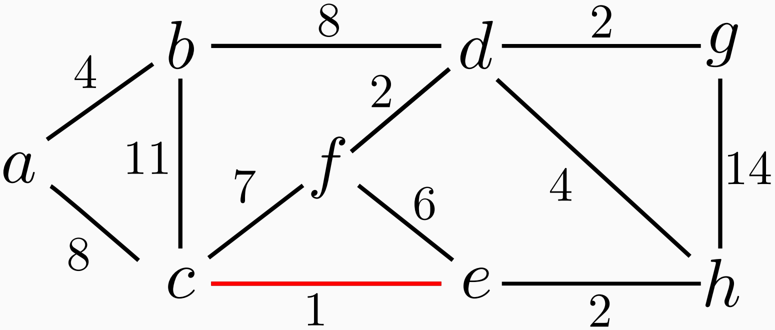
A: red edges



$$\{a\},\{b\},\{c\},\{d\},\{e\},\{f\},\{g\},\{h\}$$

Example

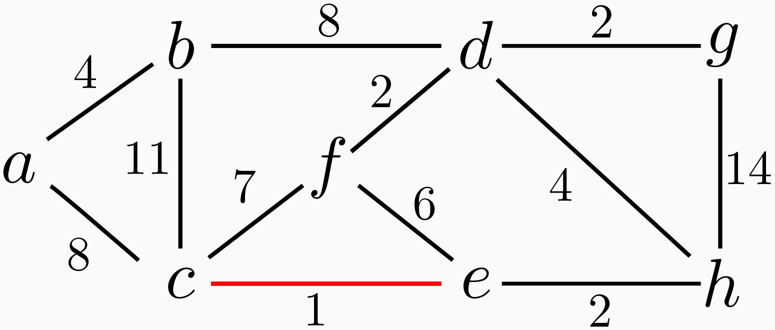
A: red edges



$$\{a\}, \{b\}, \{c\}, \{d\}, \{e\}, \{f\}, \{g\}, \{h\}$$

Example

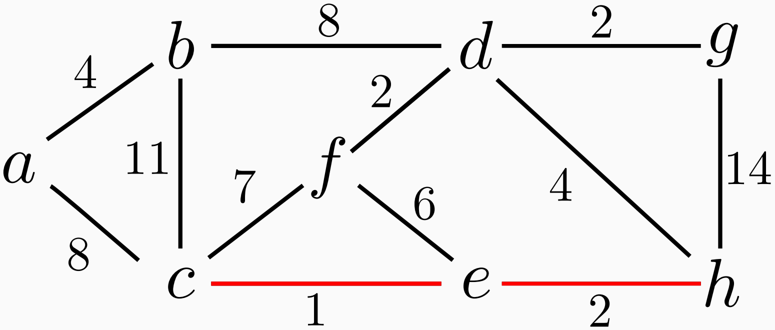
A: red edges



$$\{a\}, \{b\}, \{c, e\}, \{d\}, \{f\}, \{g\}, \{h\}$$

Example

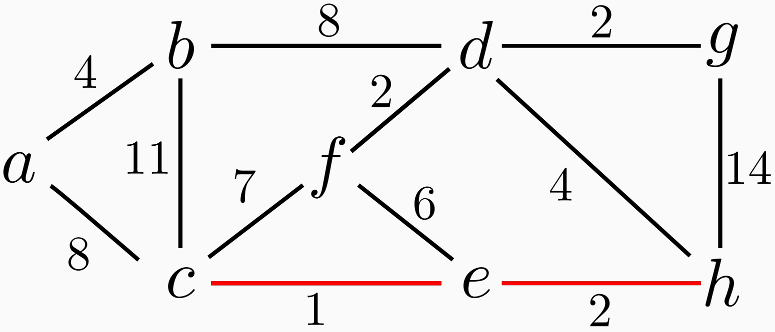
A: red edges



$$\{a\}, \{b\}, \{c, e\}, \{d\}, \{f\}, \{g\}, \{h\}$$

Example

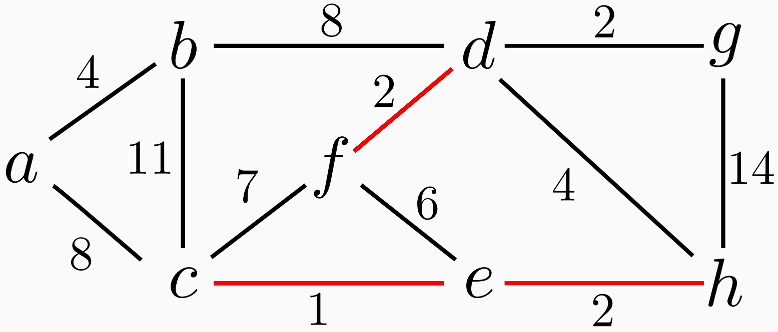
A: red edges



$$\{a\}, \{b\}, \{c, e, h\}, \{d\}, \{f\}, \{g\}$$

Example

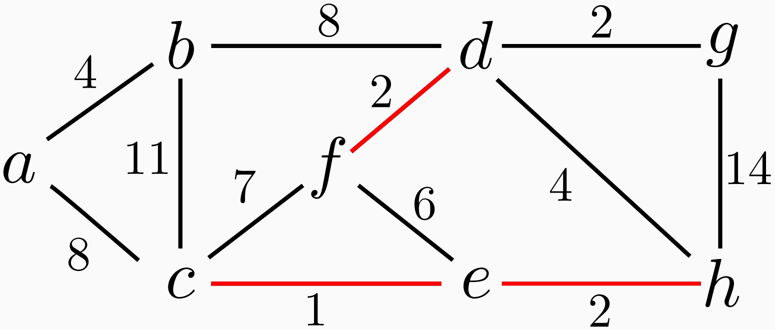
A: red edges



$$\{a\}, \{b\}, \{c, e, h\}, {\color{blue}\{d\}}, {\color{blue}\{f\}}, \{g\}$$

Example

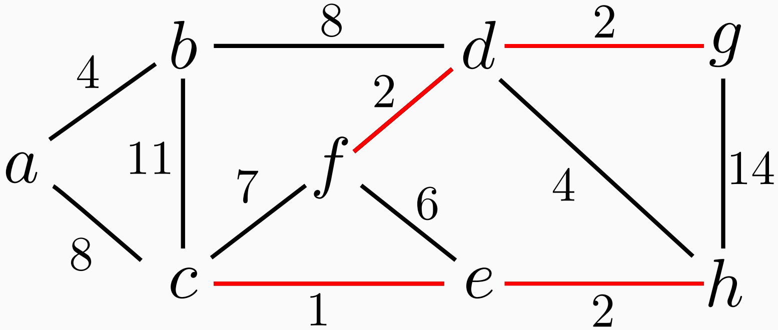
A: red edges



$$\{a\}, \{b\}, \{c, e, h\}, \{d, f\}, \{g\}$$

Example

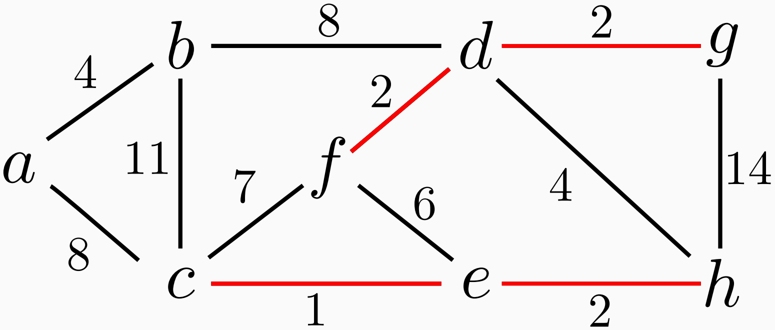
A: red edges



$$\{a\}, \{b\}, \{c, e, h\}, \{d, f\}, \{g\}$$

Example

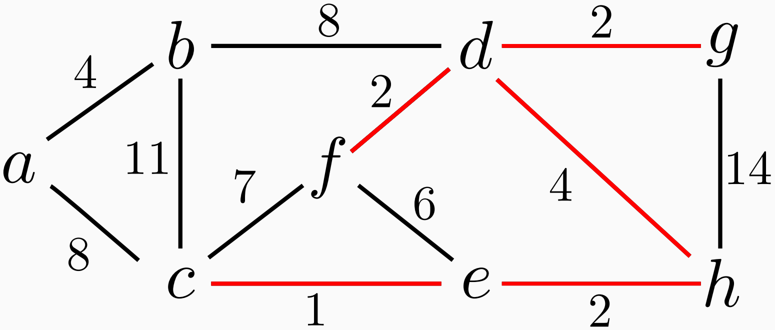
A: red edges



$$\{a\}, \{b\}, \{c, e, h\}, \{d, f, g\}$$

Example

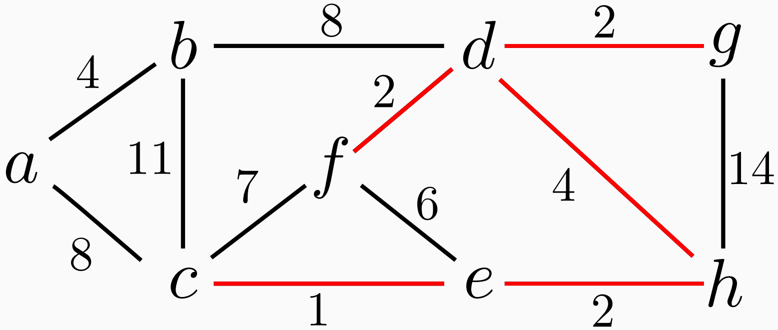
A: red edges



$$\{a\}, \{b\}, \{c, e, h\}, \{d, f, g\}$$

Example

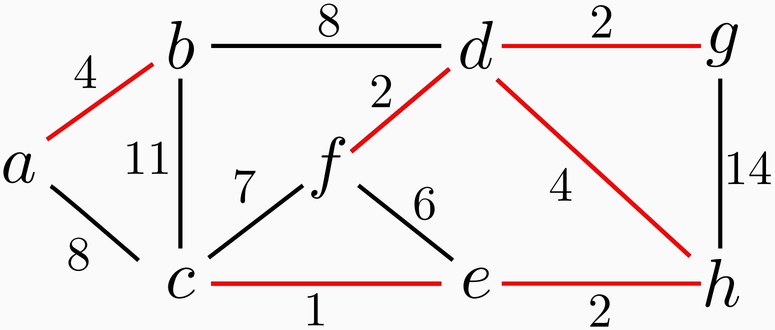
A: red edges



$$\{a\}, \{b\}, \{c, e, h, d, f, g\}$$

Example

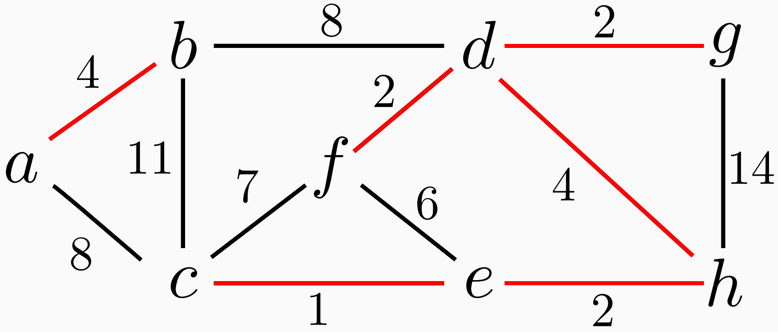
A: red edges



$$\color{blue}{\{a\}}, \color{blue}{\{b\}}, \{c, e, h, d, f, g\}$$

Example

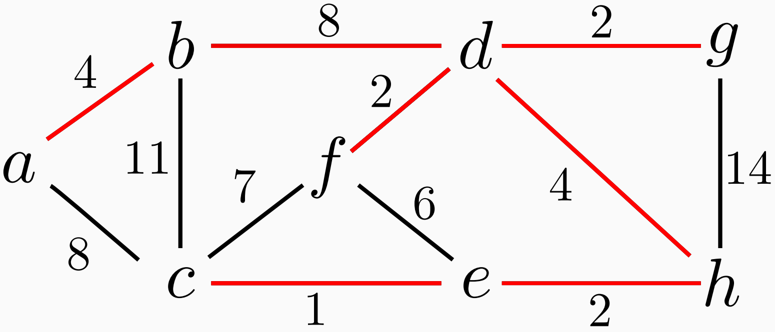
A: red edges



$$\{a, b\}, \{c, e, h, d, f, g\}$$

Example

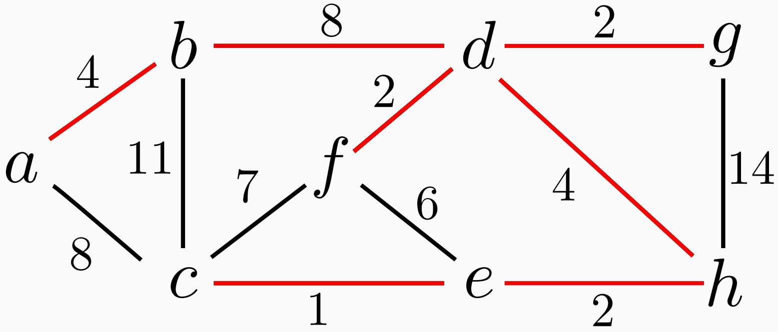
A: red edges



$$\{a, b\}, \{c, e, h, d, f, g\}$$

Example

A: red edges



$$\{a, b, c, e, h, d, f, g\}$$

Proof of correctness

We need to show two things:

Proof of correctness

We need to show two things:

1. A is a spanning tree

Proof of correctness

We need to show two things:

1. A is a spanning tree
    - it has no cycles

Proof of correctness

We need to show two things:

1. A is a spanning tree
    - it has no cycles
    - if there exists $v \in V$ that is not connected by A

Proof of correctness

We need to show two things:

1. A is a spanning tree
   - it has no cycles
   - if there exists $v \in V$ that is not connected by A
     then there exists $e \in E$ s.t. $A \cup \{e\}$ contains no cycle

Proof of correctness

We need to show two things:

1. A is a spanning tree
   - it has no cycles
   - if there exists $v \in V$ that is not connected by A then there exists $e \in E$ s.t. $A \cup \{e\}$ contains no cycle then $\textsc{Kruskal\_MST}$ will add the lightest such edge

Proof of correctness

We need to show two things:

1. A is a spanning tree
   - it has no cycles
   - if there exists $v \in V$ that is not connected by A
     then there exists $e \in E$ s.t. $A \cup \{e\}$ contains no cycle
     then $\textsc{Kruskal\_MST}$ will add the lightest such edge
2. A has the minimum weight. Use the cut property

Proof of correctness

We need to show two things:

1. A is a spanning tree
    - it has no cycles
    - if there exists $v \in V$ that is not connected by A
      then there exists $e \in E$ s.t. $A \cup \{e\}$ contains no cycle
      then $\textsc{Kruskal\_MST}$ will add the lightest such edge
2. A has the minimum weight. Use the cut property

Theorem (The cut property)

Let A be a subset of edges of some MST of $G = (V, E)$. Let $(S, V - S)$ be a cut that respects A. Let e be the lightest edge across the cut. Then $A \cup \{e\}$ is part of some MST

Proof of correctness

We need to show two things:

1. A is a spanning tree
   - it has no cycles
   - if there exists $v \in V$ that is not connected by A
     then there exists $e \in E$ s.t. $A \cup \{e\}$ contains no cycle
     then $\textsc{Kruskal\_MST}$ will add the lightest such edge
2. A has the minimum weight. Use the cut property

Theorem (The cut property)

Let A be a subset of edges of some MST of $G = (V, E)$. Let $(S, V - S)$ be a cut that respects A. Let e be the lightest edge across the cut. Then $A \cup \{e\}$ is part of some MST

if $\textsc{Kruskal\_MST}$ adds $e = (u, v)$. Let S be the vertices reachable from u in A (the partial solution so far)

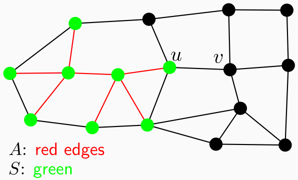
Proof of correctness

We need to show two things:

1. A is a spanning tree
   - it has no cycles
   - if there exists $v \in V$ that is not connected by A
     then there exists $e \in E$ s.t. $A \cup \{e\}$ contains no cycle
     then $\textsc{Kruskal\_MST}$ will add the lightest such edge
2. A has the minimum weight. Use the cut property

Theorem (The cut property)

Let A be a subset of edges of some MST of $G = (V, E)$. Let $(S, V - S)$ be a cut that respects A. Let e be the lightest edge across the cut. Then $A \cup \{e\}$ is part of some MST



if $\textsc{Kruskal\_MST}$ adds $e = (u, v)$. Let S be the vertices reachable from u in A (the partial solution so far)

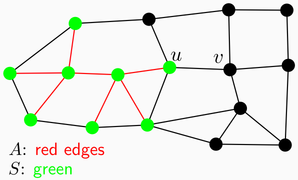
Proof of correctness

We need to show two things:

1. A is a spanning tree
   - it has no cycles
   - if there exists $v \in V$ that is not connected by A
     then there exists $e \in E$ s.t. $A \cup \{e\}$ contains no cycle
     then $\textsc{Kruskal\_MST}$ will add the lightest such edge
2. A has the minimum weight. Use the cut property

Theorem (The cut property)

Let A be a subset of edges of some MST of $G = (V, E)$. Let $(S, V - S)$ be a cut that respects A. Let e be the lightest edge across the cut. Then $A \cup \{e\}$ is part of some MST



if $\textsc{Kruskal\_MST}$ adds $e = (u, v)$. Let S be the vertices reachable from u in A (the partial solution so far)

Then $A \cup \{e\}$ is part of some MST $\square$

Proof of the cut property theorem (I)

To prove the cut property, we need a lemma

Lemma

Any connected and undirected graph $G = (V, E)$ is a tree if and only if $|E| = |V| - 1$

Proof of the cut property theorem (I)

To prove the cut property, we need a lemma

Lemma

Any connected and undirected graph $G = (V, E)$ *is a tree if and only if* $|E| = |V| - 1$

Proof.

See textbook page 129 $\square$

Proof of the theorem (II)

By assumption, A is a subset of edges of some MST T

Proof of the theorem (II)

By assumption, A is a subset of edges of some MST T

If e is part of T, then there is nothing to prove

Proof of the theorem (II)

By assumption, A is a subset of edges of some MST T

If e is part of T, then there is nothing to prove

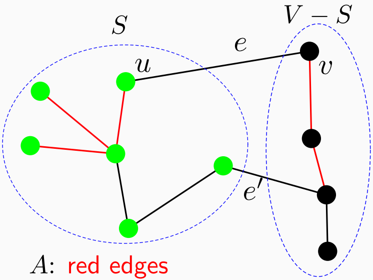
If $e \notin T$, we will construct a new MST T' with $e \in T'$:

Proof of the theorem (II)

By assumption, A is a subset of edges of some MST T

If e is part of T, then there is nothing to prove

If $e \notin T$, we will construct a new MST T' with $e \in T'$:



A: red edges

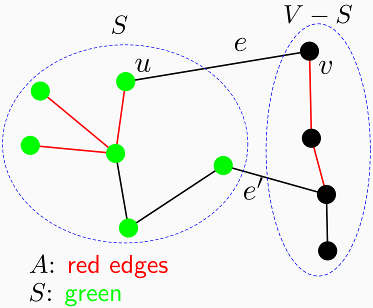
S: green

Proof of the theorem (II)

By assumption, A is a subset of edges of some MST T

If e is part of T, then there is nothing to prove

If $e \notin T$, we will construct a new MST T' with $e \in T'$:



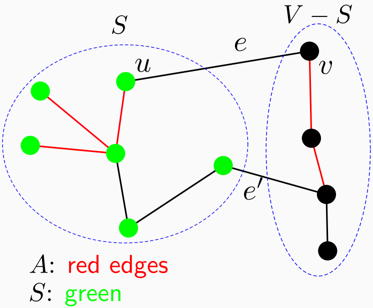
Add e to T. This will create a cycle, which must have an edge $e' \in T$ across the cut

Proof of the theorem (II)

By assumption, A is a subset of edges of some MST T

If e is part of T, then there is nothing to prove

If $e \notin T$, we will construct a new MST T' with $e \in T'$:



Add e to T. This will create a cycle, which must have an edge $e' \in T$ across the cut

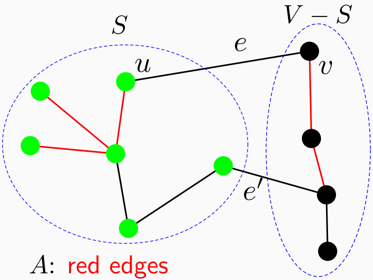
By definition of e, $w_e \leq w_{e'}$

Proof of the theorem (II)

By assumption, A is a subset of edges of some MST T

If e is part of T, then there is nothing to prove

If $e \notin T$, we will construct a new MST T' with $e \in T'$:



A: red edges
S: green

Add e to T. This will create a cycle, which must have an edge $e' \in T$ across the cut

By definition of e, $w_e \leq w_{e'}$

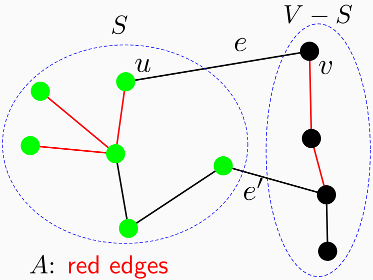
Let $T' = T \cup \{e\} - \{e'\}$.

Proof of the theorem (II)

By assumption, A is a subset of edges of some MST T

If e is part of T, then there is nothing to prove

If $e \notin T$, we will construct a new MST T' with $e \in T'$:



A: red edges

S: green

Add e to T. This will create a cycle, which must have an edge $e' \in T$ across the cut

By definition of e, $w_e \leq w_{e'}$

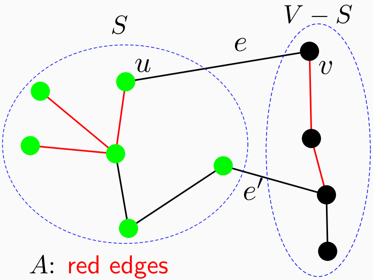
Let $T' = T \cup \{e\} - \{e'\}$. We didn't change the edge count.

Proof of the theorem (II)

By assumption, A is a subset of edges of some MST T

If e is part of T, then there is nothing to prove

If $e \notin T$, we will construct a new MST T' with $e \in T'$:



A: red edges
S: green

Add e to T. This will create a cycle, which must have an edge $e' \in T$ across the cut

By definition of e, $w_e \leq w_{e'}$

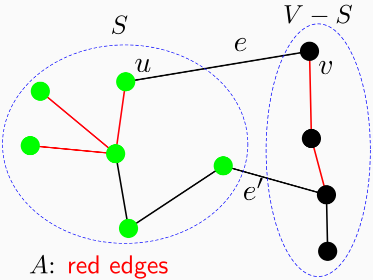
Let $T' = T \cup \{e\} - \{e'\}$. We didn't change the edge count. Lemma $\implies$ T' is a tree

Proof of the theorem (II)

By assumption, A is a subset of edges of some MST T

If e is part of T, then there is nothing to prove

If $e \notin T$, we will construct a new MST T' with $e \in T'$:



A: red edges
S: green

- Add e to T. This will create a cycle, which must have an edge $e' \in T$ across the cut
- By definition of e, $w_e \leq w_{e'}$
- Let $T' = T \cup \{e\} - \{e'\}$. We didn't change the edge count. Lemma $\implies$ T' is a tree

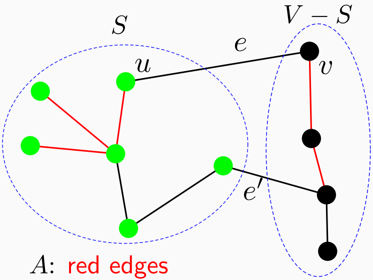
$$\mathrm{weight}(T') = \mathrm{weight}(T) + w_e - w_{e'}$$

Proof of the theorem (II)

By assumption, A is a subset of edges of some MST T

If e is part of T, then there is nothing to prove

If $e \notin T$, we will construct a new MST T' with $e \in T'$:



A: red edges
S: green

Add e to T. This will create a cycle, which must have an edge $e' \in T$ across the cut

By definition of e, $w_e \le w_{e'}$

Let $T' = T \cup \{e\} - \{e'\}$. We didn't change the edge count. Lemma $\implies$ T' is a tree

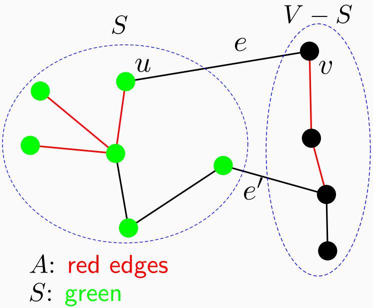
$$\begin{aligned} \mathrm{weight}(T') &= \mathrm{weight}(T) + w_e - w_{e'} \\ &\le \mathrm{weight}(T) + w_{e'} - w_{e'} \end{aligned}$$

Proof of the theorem (II)

By assumption, A is a subset of edges of some MST T

If e is part of T, then there is nothing to prove

If $e \notin T$, we will construct a new MST T' with $e \in T'$:



Add e to T. This will create a cycle, which must have an edge $e' \in T$ across the cut

By definition of e, $w_e \leq w_{e'}$

Let $T' = T \cup \{e\} - \{e'\}$. We didn't change the edge count. Lemma $\implies$ T' is a tree

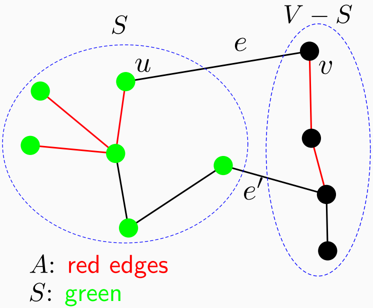
$$\begin{aligned} \mathrm{weight}(T') &= \mathrm{weight}(T) + w_e - w_{e'} \\ &\leq \mathrm{weight}(T) + w_{e'} - w_{e'} \\ &= \mathrm{weight}(T) \end{aligned}$$

Proof of the theorem (II)

By assumption, A is a subset of edges of some MST T

If e is part of T, then there is nothing to prove

If $e \notin T$, we will construct a new MST T' with $e \in T'$:



Add e to T. This will create a cycle, which must have an edge $e' \in T$ across the cut

By definition of e, $w_e \leq w_{e'}$

Let $T' = T \cup \{e\} - \{e'\}$. We didn't change the edge count. Lemma $\implies$ T' is a tree

$$\begin{aligned} \text{weight}(T') &= \text{weight}(T) + w_e - w_{e'} \\ &\leq \text{weight}(T) + w_{e'} - w_{e'} \\ &= \text{weight}(T) \end{aligned}$$

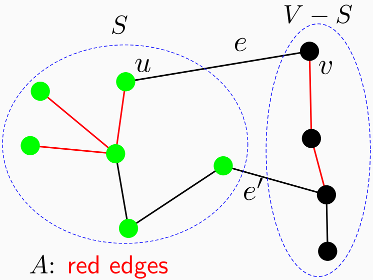
T is MST $\implies$ $\text{weight}(T') = \text{weight}(T)$

Proof of the theorem (II)

By assumption, A is a subset of edges of some MST T

If e is part of T, then there is nothing to prove

If $e \notin T$, we will construct a new MST T' with $e \in T'$:



A: red edges

S: green

Add e to T. This will create a cycle, which must have an edge $e' \in T$ across the cut

By definition of e, $w_e \leq w_{e'}$

Let $T' = T \cup \{e\} - \{e'\}$. We didn't change the edge count. Lemma $\implies$ T' is a tree

$$\begin{aligned} \text{weight}(T') &= \text{weight}(T) + w_e - w_{e'} \\ &\leq \text{weight}(T) + w_{e'} - w_{e'} \\ &= \text{weight}(T) \end{aligned}$$

T is MST $\implies$ $\text{weight}(T') = \text{weight}(T)$ $\implies$ T' is also an MST $\square$

Running time of Kruskal's algorithm (I)

Depends on how we implement make_set, find_set, and union

Running time of Kruskal's algorithm (I)

Depends on how we implement make_set, find_set, and union

Using linked list:

$$\{a, b, c\} \quad \text{head} \to a \to b \to c$$

Running time of Kruskal's algorithm (I)

Depends on how we implement make_set, find_set, and union

Using linked list:

$$\{a, b, c\} \quad \text{head} \to a \to b \to c \qquad \text{find\_set}(b):$$

Running time of Kruskal's algorithm (I)

Depends on how we implement make_set, find_set, and union

Using linked list:

$$\{a, b, c\} \quad \text{head} \to a \to b \to c \qquad \text{find\_set}(b)\text{: } O(1)$$

Running time of Kruskal's algorithm (I)

Depends on how we implement make_set, find_set, and union

Using linked list:

$\{a, b, c\} \quad \text{head} \to a \to b \to c$

$\text{find\_set}(b)$: $O(1)$

$\text{make\_set}(v)$:

Running time of Kruskal's algorithm (I)

Depends on how we implement make_set, find_set, and union

Using linked list:

$$\{a, b, c\} \quad \text{head} \to a \to b \to c \qquad \text{find\_set}(b)\text{: } O(1)$$

$$\text{make\_set}(v)\text{: } O(1)$$

Running time of Kruskal's algorithm (I)

Depends on how we implement make_set, find_set, and union

Using linked list:

$$\{a, b, c\} \quad \text{head} \to a \to b \to c \qquad \text{find\_set}(b)\text{: } O(1)$$

$$\text{make\_set}(v)\text{: } O(1)$$

$$\{d, e\} \quad \text{head} \to d \to e$$

Running time of Kruskal's algorithm (I)

Depends on how we implement make_set, find_set, and union

Using linked list:

$\{a, b, c\}$ $\quad \text{head} \to a \to b \to c$ $\qquad \text{find\_set}(b)$: $O(1)$

$\text{make\_set}(v)$: $O(1)$

$\{d, e\}$ $\quad \text{head} \to d \to e$

$\text{union}(a, b)$

Running time of Kruskal's algorithm (I)

Depends on how we implement make_set, find_set, and union

Using linked list:

$\{a, b, c\}$ $\quad \text{head} \to a \to b \to c$ $\qquad$ find_set(b): $O(1)$

make_set(v): $O(1)$

$\{d, e\}$ $\quad \text{head} \to d \to e$

union(a, b) $\qquad \text{head} \to a \to b \to c \to d \to e$

Running time of Kruskal's algorithm (I)

Depends on how we implement make_set, find_set, and union

Using linked list:

$\{a, b, c\}$ head $\to a \to b \to c$ find_set(b): $O(1)$

make_set(v): $O(1)$

$\{d, e\}$ head $\to d \to e$

union(a, b) head $\to a \to b \to c \to d \to e$

Cost of union:

Running time of Kruskal's algorithm (I)

Depends on how we implement make_set, find_set, and union

Using linked list:

$\{a, b, c\}$ $\quad \text{head} \to a \to b \to c$ $\qquad \text{find\_set}(b)$: $O(1)$

$\text{make\_set}(v)$: $O(1)$

$\{d, e\}$ $\quad \text{head} \to d \to e$

$\text{union}(a, b)$ $\qquad \text{head} \to a \to b \to c \to d \to e$

Cost of union: O(length of the shorter list)

Running time of Kruskal's algorithm (I)

Depends on how we implement make_set, find_set, and union

Using linked list:

$\{a, b, c\}$ $\quad \text{head} \to a \to b \to c$ $\qquad$ $\text{find\_set}(b)$: $O(1)$

$\text{make\_set}(v)$: $O(1)$

$\{d, e\}$ $\quad \text{head} \to d \to e$

$\text{union}(a, b)$ $\qquad \text{head} \to a \to b \to c \to d \to e$

Cost of union: O(length of the shorter list)

Using an array to implement it:

vertex	1	2	3	4	5
head	1	1	1	4	4

$\xrightarrow{\text{union}}$

Running time of Kruskal's algorithm (I)

Depends on how we implement make_set, find_set, and union

Using linked list:

$\{a, b, c\}$ $\quad \text{head} \to a \to b \to c$ $\qquad$ $\text{find\_set}(b)$: $O(1)$

$\text{make\_set}(v)$: $O(1)$

$\{d, e\}$ $\quad \text{head} \to d \to e$

$\text{union}(a, b)$ $\qquad \text{head} \to a \to b \to c \to d \to e$

Cost of union: O(length of the shorter list)

Using an array to implement it:

$$
\begin{array}{r|c|c|c|c|c|}
\hline
\text{vertex} & 1 & 2 & 3 & 4 & 5 \\
\hline
\text{head} & 1 & 1 & 1 & 4 & 4 \\
\hline
\end{array}
\xrightarrow{\text{union}}
\begin{array}{|c|c|c|c|c|}
\hline
1 & 2 & 3 & 4 & 5 \\
\hline
1 & 1 & 1 & 1 & 1 \\
\hline
\end{array}
$$