CSCI3160: Tutorial

Zhihan Jiang

CSE, CUHK

Outline

- Kruskal's algorithm for solving the MST problem.
- Correctness proof.

Review: the MST Problem

Let $G = (V, E)$ be a connected undirected graph. Let w be a function that maps each edge e of G to a positive integer $w(e)$ called the weight of e.

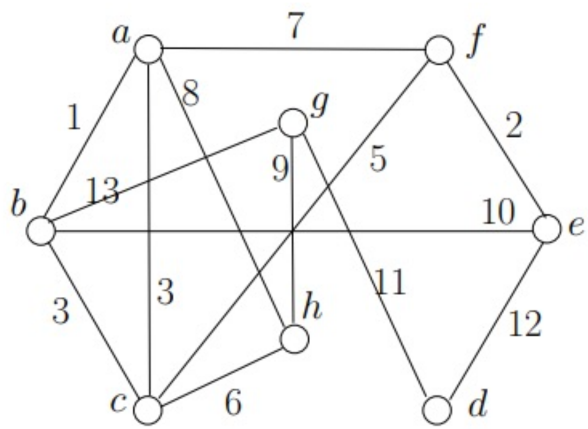
A spanning tree T is a tree satisfying the following conditions:

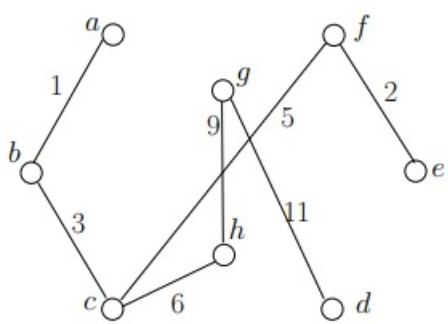
- The vertex set of T is V.
- Every edge of T is an edge in G.

The cost of T is the sum of the weights of all the edges in T.

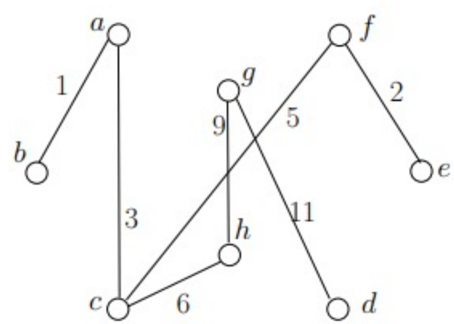
The goal of the minimum spanning tree (MST) problem is to find a spanning tree of the smallest cost.

Example

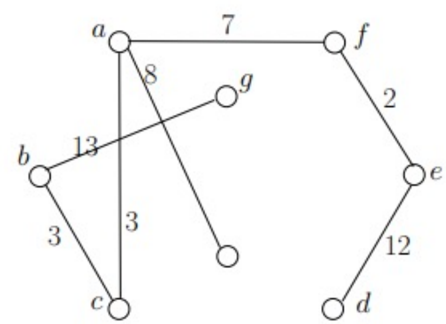




Cost 37



Cost 37



Cost 48

Kruskal's algorithm

The algorithm maintains a forest F where each vertex belongs to exactly one tree in F.

Define t as the number of trees in the current F.

At the beginning, $t = |V|$: F has $|V|$ trees each containing a single vertex.
At the end, $t = 1$: F becomes our final MST.

Cross edge: An edge $\{u, v\}$ where u and v belong to different trees in F.

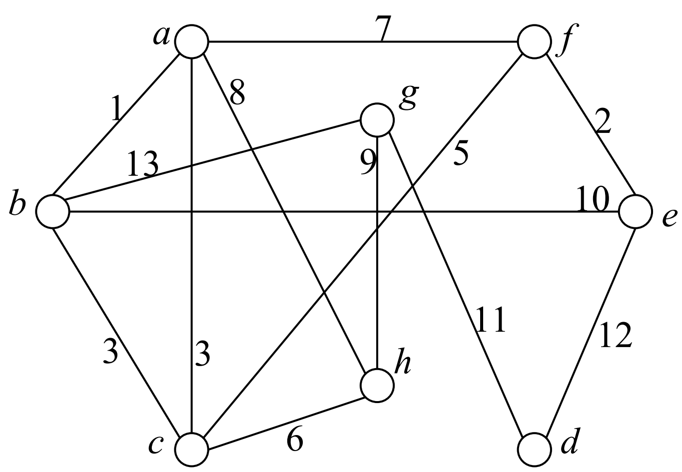
Greedy: The algorithm works by repeatedly taking the lightest cross edge.

Example

At the beginning, $|V| = 8$ trees: each tree has only one vertex.

Every edge is a cross edge at the moment.

Edge $\{a, b\}$ is the lightest cross edge.



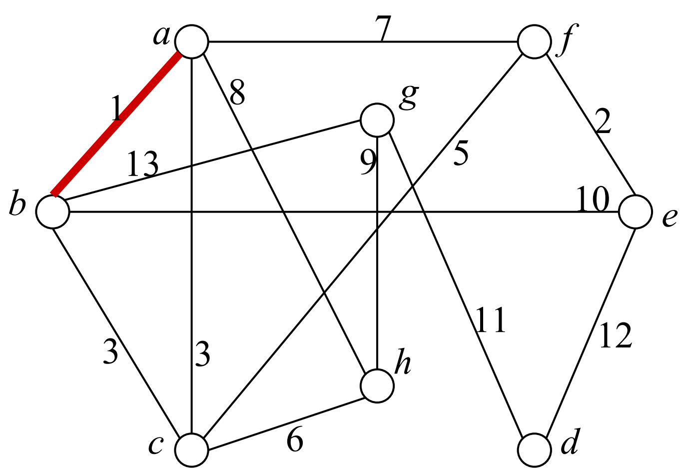
Trees	Vertices
T_1	a
T_2	b
T_3	c
T_4	d
T_5	e
T_6	f
T_7	g
T_8	h

Example

We pick $\{a, b\}$, marked red in the figure, and merge the trees of a and b.

Cross edges are shown in black.

$\{e, f\}$ is the lightest cross edge.

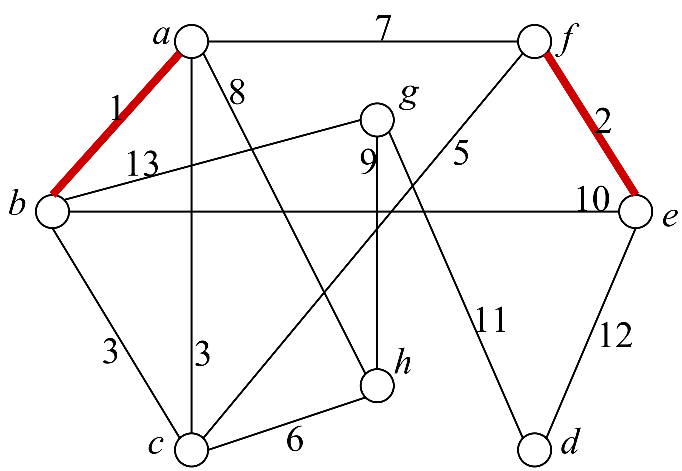


Trees	Vertices
T_1	a, b
$\cancel{T_2}$	~~b~~
T_3	c
T_4	d
T_5	e
T_6	f
T_7	g
T_8	h

Example

We pick $\{e, f\}$, merging the trees of e and f into one.

Cross edges are shown in black solid segments.
$\{a, c\}$ and $\{b, c\}$ are both the lightest cross edges.



Trees	Vertices
T_1	a, b
$\cancel{T_2}$	~~b~~
T_3	c
T_4	d
T_5	e, f
$\cancel{T_6}$	~~f~~
T_7	g
T_8	h

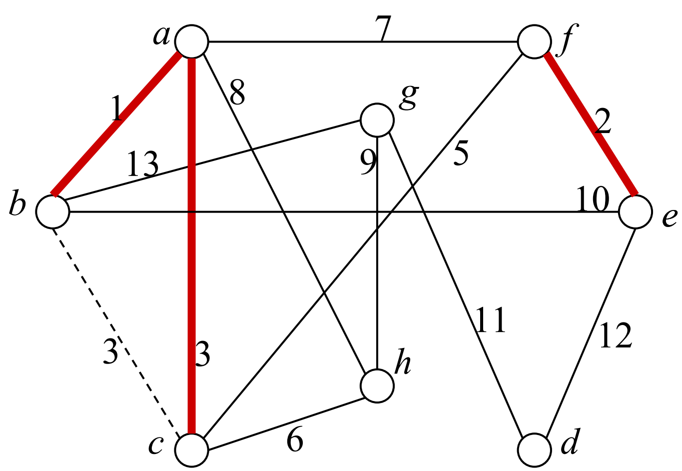
Example

We pick $\{a, c\}$ (you could also pick $\{b, c\}$), merging the trees of a and c into one.

Cross edges are shown in black solid segments.

- $\{b, c\}$ is no longer a cross edge.

$\{c, f\}$ is the lightest cross edge.



Trees	Vertices
T_1	a, b, c
$\cancel{T_2}$	~~b~~
$\cancel{T_3}$	~~c~~
T_4	d
T_5	e, f
$\cancel{T_6}$	~~f~~
T_7	g
T_8	h

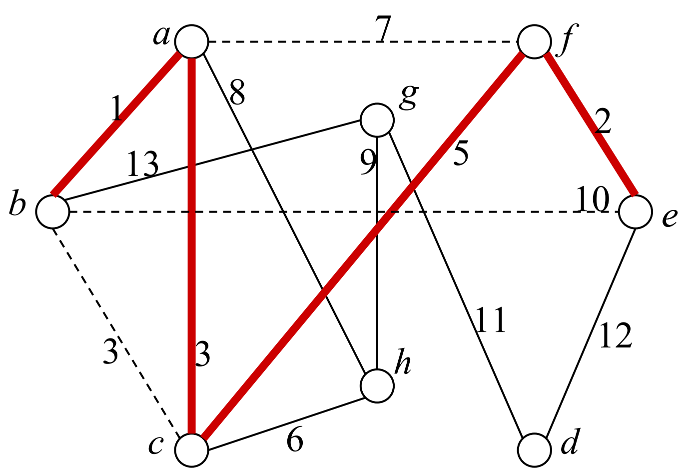
Example

We pick $\{c, f\}$, merging the trees of c and f into one.

Cross edges are shown in black solid segments.

- $\{a, f\}, \{b, e\}$ are no longer cross edges.

$\{c, h\}$ is the lightest cross edge.



Trees	Vertices
T_1	a, b, c, e, f
$\cancel{T_2}$	~~b~~
$\cancel{T_3}$	~~e~~
T_4	d
$\cancel{T_5}$	~~e, f~~
$\cancel{T_6}$	~~f~~
T_7	g
T_8	h

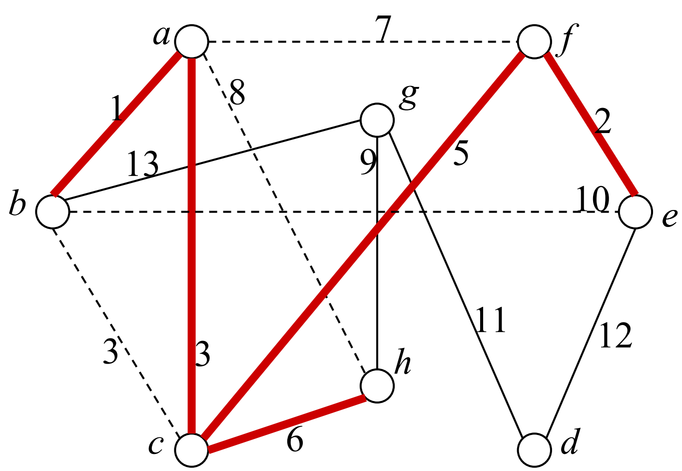
Example

We pick $\{c, h\}$, merging the trees of c and h into one.

Cross edges are shown in black solid segments.

- $\{a, h\}$ is no longer a cross edge.

$\{g, h\}$ is the lightest cross edge.



Trees	Vertices
T_1	a, b, c, e, f, h
~~T_2~~	~~b~~
~~T_3~~	~~c~~
T_4	d
~~T_5~~	~~e, f~~
~~T_6~~	~~f~~
T_7	g
~~T_8~~	~~h~~

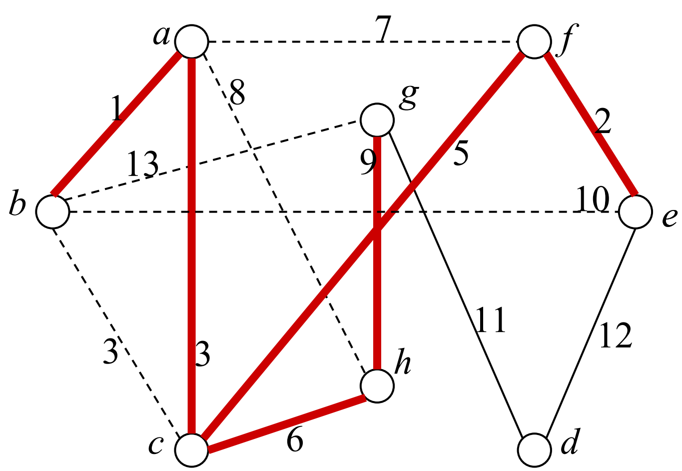
Example

We pick $\{g, h\}$, merging the trees of g and h into one.

Cross edges are shown in black solid segments.

- $\{b, g\}$ is no longer a cross edge.

$\{d, g\}$ is the lightest cross edge.



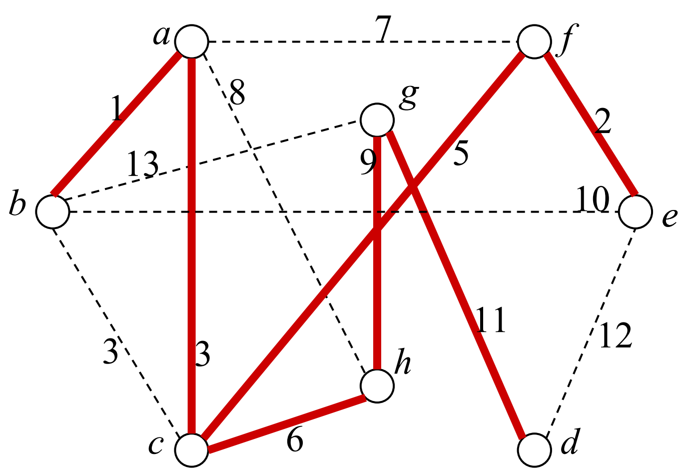
Trees	Vertices
T_1	a, b, c, e, f, g, h
$\cancel{T_2}$	~~b~~
$\cancel{T_3}$	~~c~~
T_4	d
$\cancel{T_5}$	~~e, f~~
$\cancel{T_6}$	~~f~~
$\cancel{T_7}$	~~g~~
$\cancel{T_8}$	~~h~~

Example

We pick $\{d, g\}$, merging the trees of d and g into one.

Cross edges are shown in black solid segments.

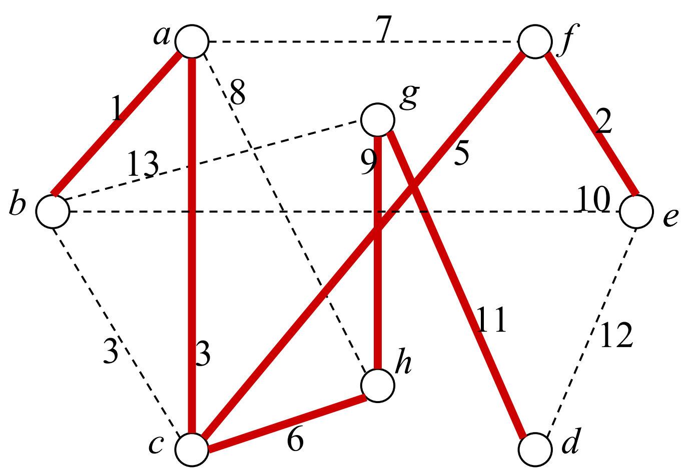
- $\{d, e\}$ is no longer a cross edge.



Trees	Vertices
T_1	a, b, c, d, e, f, g, h
$\cancel{T_2}$	~~b~~
$\cancel{T_3}$	~~c~~
$\cancel{T_4}$	~~d~~
$\cancel{T_5}$	~~e, f~~
$\cancel{T_6}$	~~f~~
$\cancel{T_7}$	~~g~~
$\cancel{T_8}$	~~h~~

Example

Now, there is only one tree T_1 in forest F, which is our final MST.



Trees	Vertices
T_1	a, b, c, d, e, f, g, h
~~T_2~~	~~b~~
~~T_3~~	~~c~~
~~T_4~~	~~d~~
~~T_5~~	~~e, f~~
~~T_6~~	~~f~~
~~T_7~~	~~g~~
~~T_8~~	~~h~~

Correctness Proof

Next, we will prove that Kruskal's algorithm returns an MST.

Let e_i ($i \in [1, |V| - 1]$) be the i-th edge picked, that is, the algorithm picks edges in this order: $e_1, e_2, \ldots, e_{|V|-1}$.

Claim: For any $k \in [1, |V| - 1]$, there is an MST containing $e_1, e_2, \ldots, e_k$.

We will prove the claim by induction.

Base Case: $k = 1$. We have proved this in class.

Correctness Proof

Claim: For any $k \in [1, |V| - 1]$, there is an MST containing $e_1, e_2, \ldots, e_k$.

Inductive Case: Assuming the claim's correctness for $k = i - 1$ $(i \geq 2)$, we will prove it for $k = i$.

By the inductive assumption, there is an MST T that includes $e_1, \ldots, e_{i-1}$.

If T includes e_i, the claim already holds and we are done.

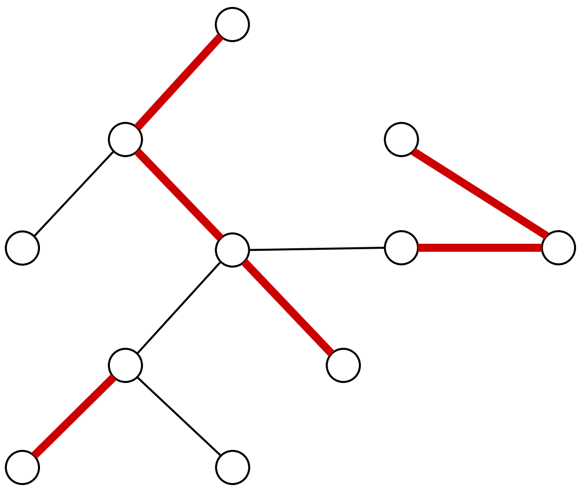
Next, we will focus on the case where T does not include e_i.

Correctness Proof

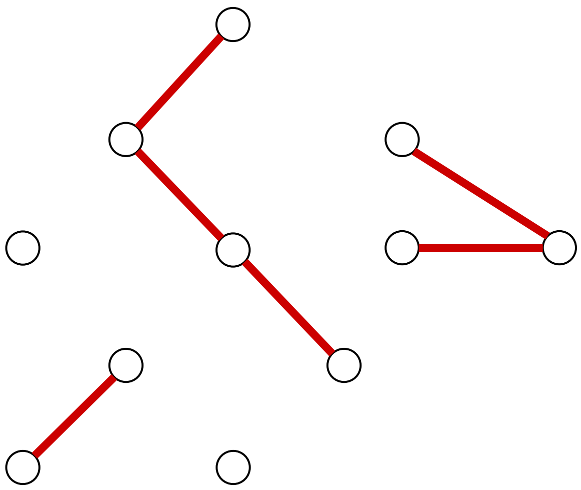
By the inductive assumption, there is an MST T that includes $e_1, \ldots, e_{i-1}$.

Consider the graph $G' = (V, \{e_1, \ldots, e_{i-1}\})$; this is the forest maintained by the algorithm after picking e_{i-1}.

Here is an example of T and G' where $i = 7$, and $e_1, \ldots, e_{i-1}$ are shown in red.



T

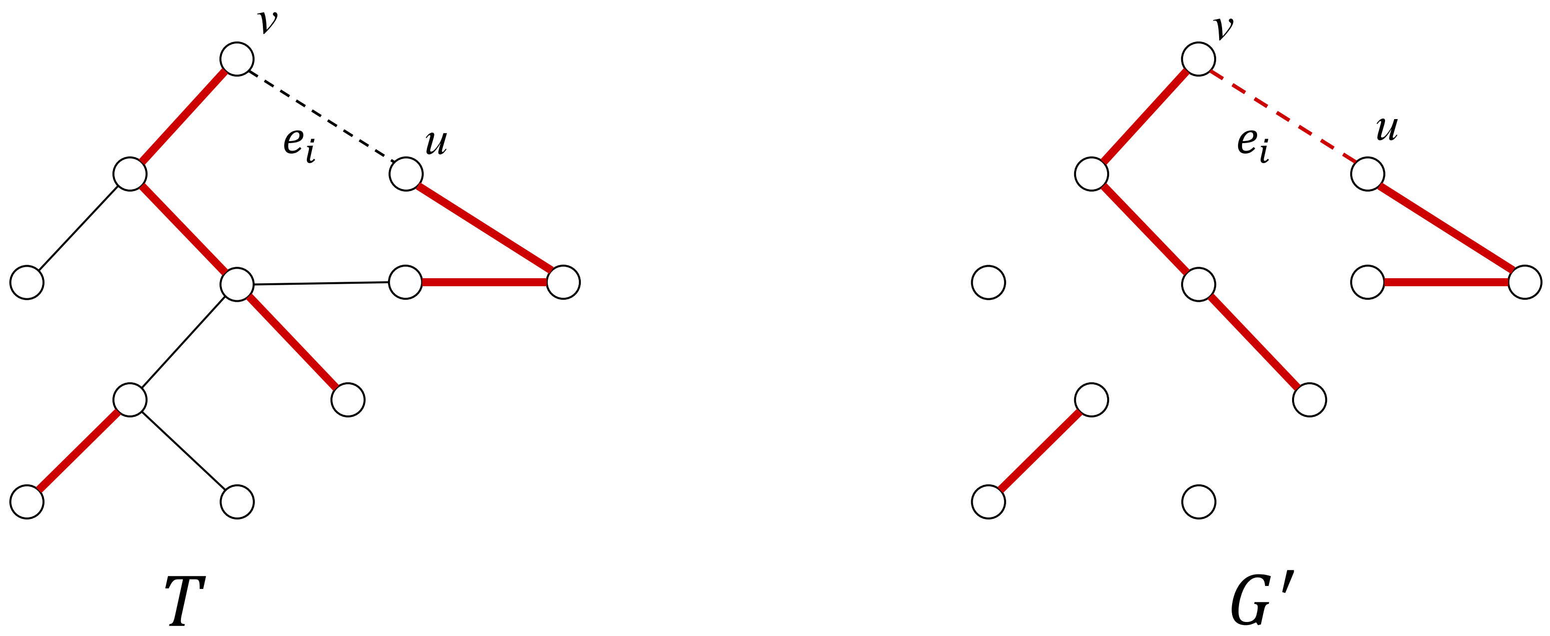


G'

Correctness Proof

By how the algorithm runs, the edge $e_i = \{u, v\}$ must be a cross edge in G', i.e., u and v are in different trees.

Since T does not include e_i, adding e_i to T creates a cycle.

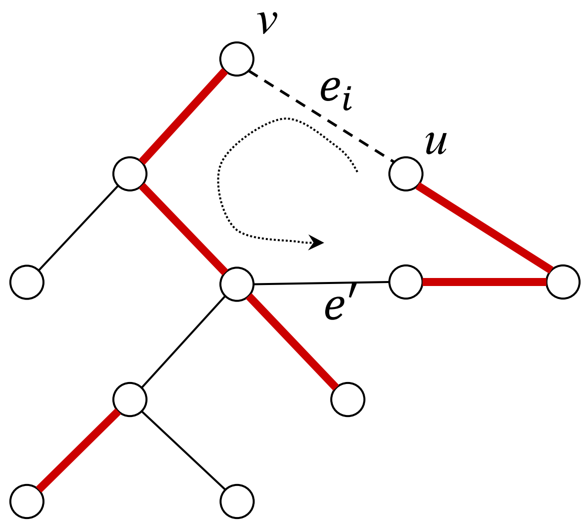


Correctness Proof

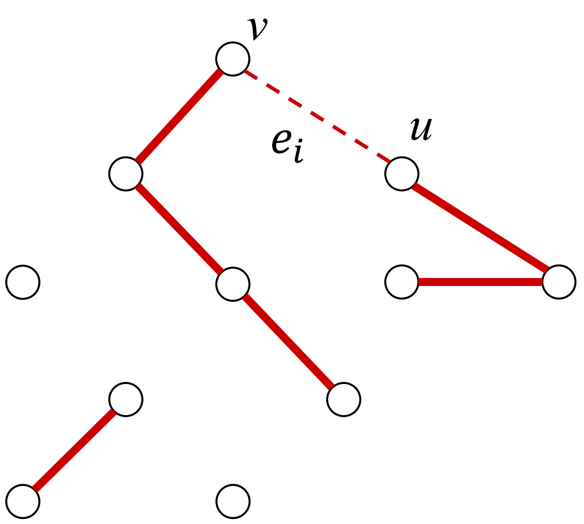
Walk on this cycle in the following manner:

- start from u;
- cross e_i to reach v and continue in this direction;
- stop right after having crossed an edge e' that takes us back to the tree of u.

Both e_i and e' are cross edges before the algorithm picks the i-th edge. Hence e_i cannot be heavier than e'.



T



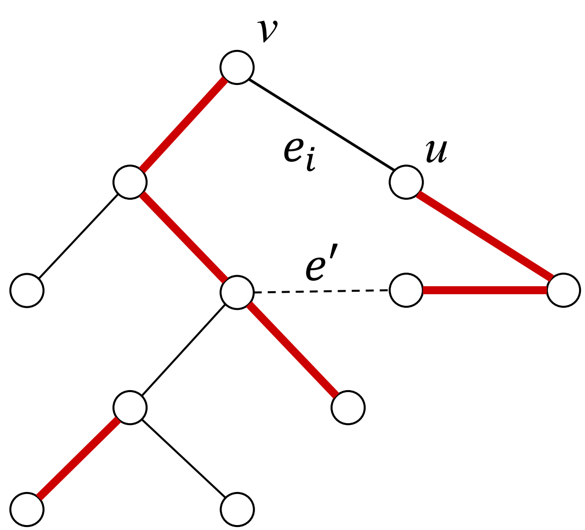
G'

Correctness Proof

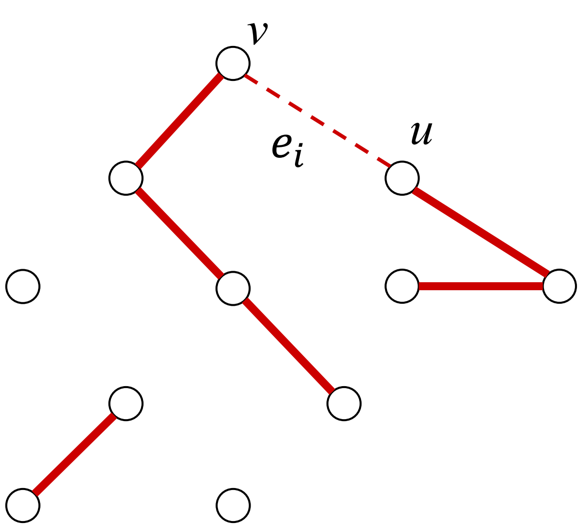
Remove e' from T and add e_i.

This yields another MST T', which contains $e_1, \ldots, e_i$.

We thus have proved the claim for $k = i$.



T'



G'

Running Time

Kruskal's algorithm can be implemented in $O(|E| \cdot \log|E|)$ time.

- This is not trivial
  (but you have learned all the data structures required in the implementation).