Minimum Spanning Trees II: Kruskal's algorithm
COMS20017 (Algorithms and Data)

John Lapinskas, University of Bristol

Motivation

Minimum spanning tree is like the opposite of interval scheduling — almost **any** greedy approach will yield a working algorithm.

Motivation

Minimum spanning tree is like the opposite of interval scheduling — almost **any** greedy approach will yield a working algorithm.

But we already have a good algorithm: Prim's runs in $O(|E|\log|E|)$ time, and we can't beat $O(|E|)$ time since we need to read the input.

So why am I bothering to teach you Kruskal's algorithm as well?

Motivation

Minimum spanning tree is like the opposite of interval scheduling — almost **any** greedy approach will yield a working algorithm.

But we already have a good algorithm: Prim's runs in $O(|E| \log |E|)$ time, and we can't beat $O(|E|)$ time since we need to read the input.

So why am I bothering to teach you Kruskal's algorithm as well?

- It has slightly better constant factors (debatably);

Motivation

Minimum spanning tree is like the opposite of interval scheduling — almost **any** greedy approach will yield a working algorithm.

But we already have a good algorithm: Prim's runs in $O(|E| \log |E|)$ time, and we can't beat $O(|E|)$ time since we need to read the input.

So why am I bothering to teach you Kruskal's algorithm as well?

- It has slightly better constant factors (debatably);
- It's an application of a cool and useful data structure;

Motivation

Minimum spanning tree is like the opposite of interval scheduling — almost **any** greedy approach will yield a working algorithm.

But we already have a good algorithm: Prim's runs in $O(|E| \log |E|)$ time, and we can't beat $O(|E|)$ time since we need to read the input.

So why am I bothering to teach you Kruskal's algorithm as well?

- It has slightly better constant factors (debatably);
- It's an application of a cool and useful data structure;
- The **really** good algorithms use ideas from both Prim and Kruskal. (More on this next week!)

Motivation

Minimum spanning tree is like the opposite of interval scheduling — almost **any** greedy approach will yield a working algorithm.

But we already have a good algorithm: Prim's runs in $O(|E|\log|E|)$ time, and we can't beat $O(|E|)$ time since we need to read the input.

So why am I bothering to teach you Kruskal's algorithm as well?

- It has slightly better constant factors (debatably);
- It's an application of a cool and useful data structure;
- The **really** good algorithms use ideas from both Prim and Kruskal. (More on this next week!)
- Interviewers might expect you to know it...

Motivation

Minimum spanning tree is like the opposite of interval scheduling — almost **any** greedy approach will yield a working algorithm.

But we already have a good algorithm: Prim's runs in $O(|E| \log |E|)$ time, and we can't beat $O(|E|)$ time since we need to read the input.

So why am I bothering to teach you Kruskal's algorithm as well?

- It has slightly better constant factors (debatably);
- It's an application of a cool and useful data structure;
- The **really** good algorithms use ideas from both Prim and Kruskal. (More on this next week!)
- Interviewers might expect you to know it... 🤮🤮🤮

Kruskal's algorithm: The idea

Input: A connected weighted graph $G = ((V, E), w)$. **Output:** A minimum spanning tree of G.

A **minimum spanning tree** is a subtree T of G covering all of G's vertices, whose total weight $\sum_{e \in E(T)} w(e)$ is as small as possible.

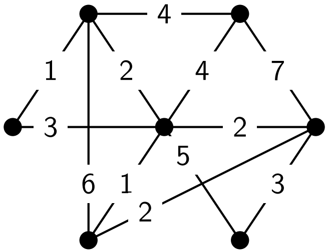
Kruskal's algorithm: The idea

Input: A connected weighted graph $G = ((V, E), w)$. **Output:** A minimum spanning tree of G.

A **minimum spanning tree** is a subtree T of G covering all of G's vertices, whose total weight $\sum_{e \in E(T)} w(e)$ is as small as possible.

We are even more greedy than in Prim's algorithm.

Rather than picking the lowest-weight edge that grows our component, we just pick the lowest-weight edge **anywhere** that doesn't make a cycle.



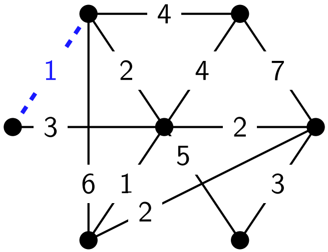
Kruskal's algorithm: The idea

Input: A connected weighted graph $G = ((V, E), w)$. **Output:** A minimum spanning tree of G.

A **minimum spanning tree** is a subtree T of G covering all of G's vertices, whose total weight $\sum_{e \in E(T)} w(e)$ is as small as possible.

We are even more greedy than in Prim's algorithm.

Rather than picking the lowest-weight edge that grows our component, we just pick the lowest-weight edge **anywhere** that doesn't make a cycle.



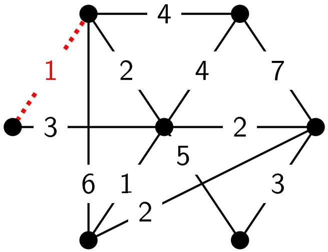
Kruskal's algorithm: The idea

Input: A connected weighted graph $G = ((V, E), w)$. **Output:** A minimum spanning tree of G.

A **minimum spanning tree** is a subtree T of G covering all of G's vertices, whose total weight $\sum_{e \in E(T)} w(e)$ is as small as possible.

We are even more greedy than in Prim's algorithm.

Rather than picking the lowest-weight edge that grows our component, we just pick the lowest-weight edge **anywhere** that doesn't make a cycle.



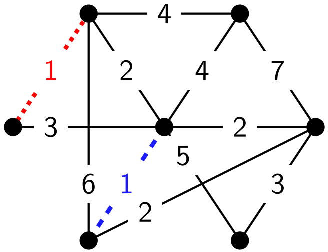
Kruskal's algorithm: The idea

Input: A connected weighted graph $G = ((V, E), w)$. **Output:** A minimum spanning tree of G.

A **minimum spanning tree** is a subtree T of G covering all of G's vertices, whose total weight $\sum_{e \in E(T)} w(e)$ is as small as possible.

We are even more greedy than in Prim's algorithm.

Rather than picking the lowest-weight edge that grows our component, we just pick the lowest-weight edge **anywhere** that doesn't make a cycle.

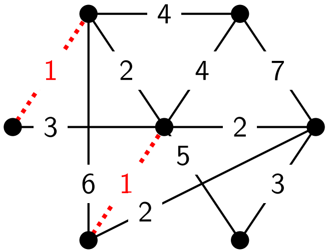


Input: A connected weighted graph $G = ((V, E), w)$. **Output:** A minimum spanning tree of G.

A **minimum spanning tree** is a subtree T of G covering all of G's vertices, whose total weight $\sum_{e \in E(T)} w(e)$ is as small as possible.

We are even more greedy than in Prim's algorithm.

Rather than picking the lowest-weight edge that grows our component, we just pick the lowest-weight edge **anywhere** that doesn't make a cycle.



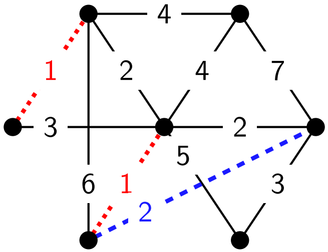
Kruskal's algorithm: The idea

Input: A connected weighted graph $G = ((V, E), w)$. **Output:** A minimum spanning tree of G.

A **minimum spanning tree** is a subtree T of G covering all of G's vertices, whose total weight $\sum_{e \in E(T)} w(e)$ is as small as possible.

We are even more greedy than in Prim's algorithm.

Rather than picking the lowest-weight edge that grows our component, we just pick the lowest-weight edge **anywhere** that doesn't make a cycle.

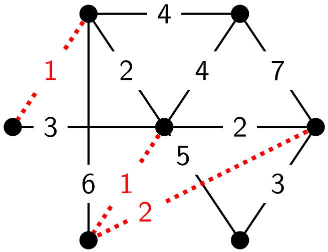


Input: A connected weighted graph $G = ((V, E), w)$. **Output:** A minimum spanning tree of G.

A **minimum spanning tree** is a subtree T of G covering all of G's vertices, whose total weight $\sum_{e \in E(T)} w(e)$ is as small as possible.

We are even more greedy than in Prim's algorithm.

Rather than picking the lowest-weight edge that grows our component, we just pick the lowest-weight edge **anywhere** that doesn't make a cycle.



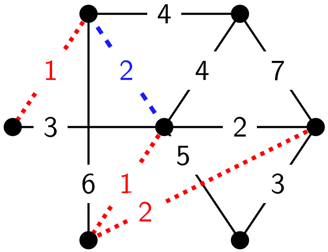
Kruskal's algorithm: The idea

Input: A connected weighted graph $G = ((V, E), w)$. **Output:** A minimum spanning tree of G.

A **minimum spanning tree** is a subtree T of G covering all of G's vertices, whose total weight $\sum_{e \in E(T)} w(e)$ is as small as possible.

We are even more greedy than in Prim's algorithm.

Rather than picking the lowest-weight edge that grows our component, we just pick the lowest-weight edge **anywhere** that doesn't make a cycle.



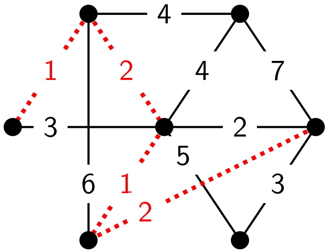
Kruskal's algorithm: The idea

Input: A connected weighted graph $G = ((V, E), w)$. **Output:** A minimum spanning tree of G.

A **minimum spanning tree** is a subtree T of G covering all of G's vertices, whose total weight $\sum_{e \in E(T)} w(e)$ is as small as possible.

We are even more greedy than in Prim's algorithm.

Rather than picking the lowest-weight edge that grows our component, we just pick the lowest-weight edge **anywhere** that doesn't make a cycle.



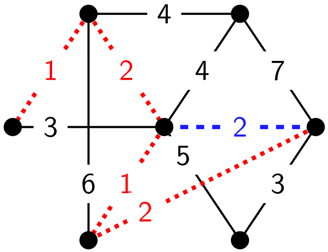
Kruskal's algorithm: The idea

Input: A connected weighted graph $G = ((V, E), w)$. **Output:** A minimum spanning tree of G.

A **minimum spanning tree** is a subtree T of G covering all of G's vertices, whose total weight $\sum_{e \in E(T)} w(e)$ is as small as possible.

We are even more greedy than in Prim's algorithm.

Rather than picking the lowest-weight edge that grows our component, we just pick the lowest-weight edge **anywhere** that doesn't make a cycle.



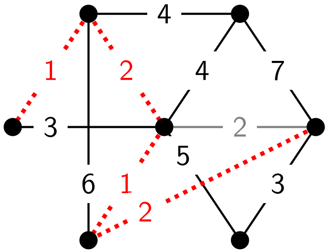
Kruskal's algorithm: The idea

Input: A connected weighted graph $G = ((V, E), w)$. **Output:** A minimum spanning tree of G.

A **minimum spanning tree** is a subtree T of G covering all of G's vertices, whose total weight $\sum_{e \in E(T)} w(e)$ is as small as possible.

We are even more greedy than in Prim's algorithm.

Rather than picking the lowest-weight edge that grows our component, we just pick the lowest-weight edge **anywhere** that doesn't make a cycle.



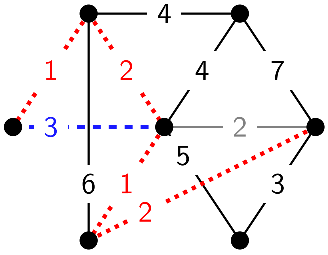
Kruskal's algorithm: The idea

Input: A connected weighted graph $G = ((V, E), w)$. **Output:** A minimum spanning tree of G.

A **minimum spanning tree** is a subtree T of G covering all of G's vertices, whose total weight $\sum_{e \in E(T)} w(e)$ is as small as possible.

We are even more greedy than in Prim's algorithm.

Rather than picking the lowest-weight edge that grows our component, we just pick the lowest-weight edge **anywhere** that doesn't make a cycle.



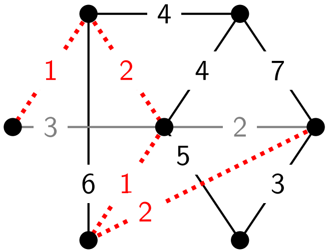
Kruskal's algorithm: The idea

Input: A connected weighted graph $G = ((V, E), w)$. **Output:** A minimum spanning tree of G.

A **minimum spanning tree** is a subtree T of G covering all of G's vertices, whose total weight $\sum_{e \in E(T)} w(e)$ is as small as possible.

We are even more greedy than in Prim's algorithm.

Rather than picking the lowest-weight edge that grows our component, we just pick the lowest-weight edge **anywhere** that doesn't make a cycle.



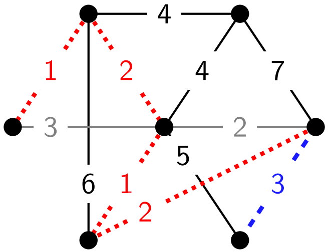
Kruskal's algorithm: The idea

Input: A connected weighted graph $G = ((V, E), w)$. **Output:** A minimum spanning tree of G.

A **minimum spanning tree** is a subtree T of G covering all of G's vertices, whose total weight $\sum_{e \in E(T)} w(e)$ is as small as possible.

We are even more greedy than in Prim's algorithm.

Rather than picking the lowest-weight edge that grows our component, we just pick the lowest-weight edge **anywhere** that doesn't make a cycle.



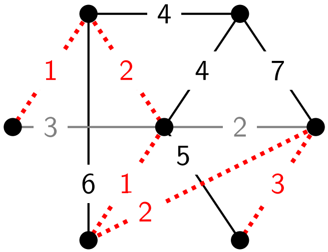
Kruskal's algorithm: The idea

Input: A connected weighted graph $G = ((V, E), w)$. **Output:** A minimum spanning tree of G.

A **minimum spanning tree** is a subtree T of G covering all of G's vertices, whose total weight $\sum_{e \in E(T)} w(e)$ is as small as possible.

We are even more greedy than in Prim's algorithm.

Rather than picking the lowest-weight edge that grows our component, we just pick the lowest-weight edge **anywhere** that doesn't make a cycle.

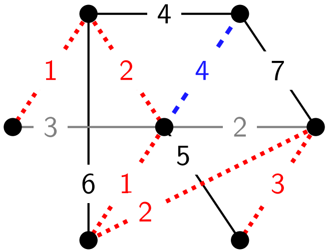


Input: A connected weighted graph $G = ((V,E),w)$. **Output:** A minimum spanning tree of G.

A **minimum spanning tree** is a subtree T of G covering all of G's vertices, whose total weight $\sum_{e \in E(T)} w(e)$ is as small as possible.

We are even more greedy than in Prim's algorithm.

Rather than picking the lowest-weight edge that grows our component, we just pick the lowest-weight edge **anywhere** that doesn't make a cycle.



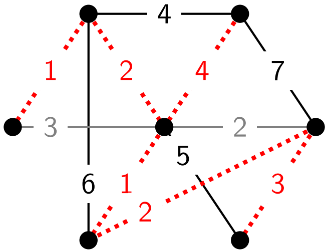
Kruskal's algorithm: The idea

Input: A connected weighted graph $G = ((V, E), w)$. **Output:** A minimum spanning tree of G.

A **minimum spanning tree** is a subtree T of G covering all of G's vertices, whose total weight $\sum_{e \in E(T)} w(e)$ is as small as possible.

We are even more greedy than in Prim's algorithm.

Rather than picking the lowest-weight edge that grows our component, we just pick the lowest-weight edge **anywhere** that doesn't make a cycle.

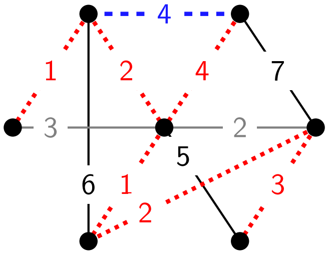


Input: A connected weighted graph $G = ((V, E), w)$. **Output:** A minimum spanning tree of G.

A **minimum spanning tree** is a subtree T of G covering all of G's vertices, whose total weight $\sum_{e \in E(T)} w(e)$ is as small as possible.

We are even more greedy than in Prim's algorithm.

Rather than picking the lowest-weight edge that grows our component, we just pick the lowest-weight edge **anywhere** that doesn't make a cycle.



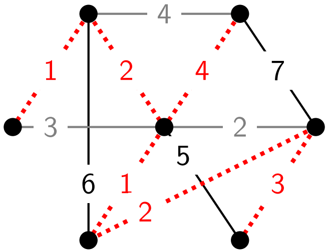
Kruskal's algorithm: The idea

Input: A connected weighted graph $G = ((V, E), w)$. **Output:** A minimum spanning tree of G.

A **minimum spanning tree** is a subtree T of G covering all of G's vertices, whose total weight $\sum_{e \in E(T)} w(e)$ is as small as possible.

We are even more greedy than in Prim's algorithm.

Rather than picking the lowest-weight edge that grows our component, we just pick the lowest-weight edge **anywhere** that doesn't make a cycle.

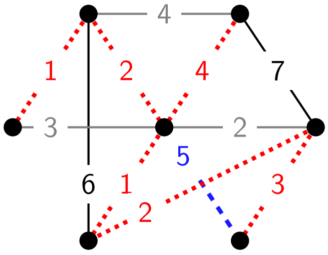


Input: A connected weighted graph $G = ((V,E),w)$. **Output:** A minimum spanning tree of G.

A **minimum spanning tree** is a subtree T of G covering all of G's vertices, whose total weight $\sum_{e \in E(T)} w(e)$ is as small as possible.

We are even more greedy than in Prim's algorithm.

Rather than picking the lowest-weight edge that grows our component, we just pick the lowest-weight edge **anywhere** that doesn't make a cycle.

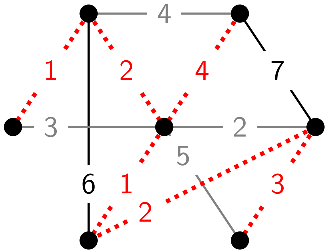


Input: A connected weighted graph $G = ((V, E), w)$. **Output:** A minimum spanning tree of G.

A **minimum spanning tree** is a subtree T of G covering all of G's vertices, whose total weight $\sum_{e \in E(T)} w(e)$ is as small as possible.

We are even more greedy than in Prim's algorithm.

Rather than picking the lowest-weight edge that grows our component, we just pick the lowest-weight edge **anywhere** that doesn't make a cycle.



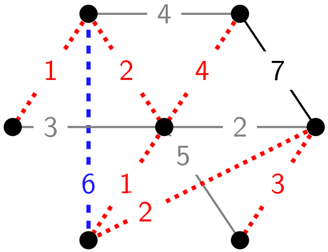
Kruskal's algorithm: The idea

Input: A connected weighted graph $G = ((V, E), w)$. **Output:** A minimum spanning tree of G.

A **minimum spanning tree** is a subtree T of G covering all of G's vertices, whose total weight $\sum_{e \in E(T)} w(e)$ is as small as possible.

We are even more greedy than in Prim's algorithm.

Rather than picking the lowest-weight edge that grows our component, we just pick the lowest-weight edge **anywhere** that doesn't make a cycle.

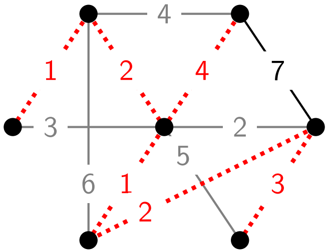


Input: A connected weighted graph $G = ((V, E), w)$. **Output:** A minimum spanning tree of G.

A **minimum spanning tree** is a subtree T of G covering all of G's vertices, whose total weight $\sum_{e \in E(T)} w(e)$ is as small as possible.

We are even more greedy than in Prim's algorithm.

Rather than picking the lowest-weight edge that grows our component, we just pick the lowest-weight edge **anywhere** that doesn't make a cycle.

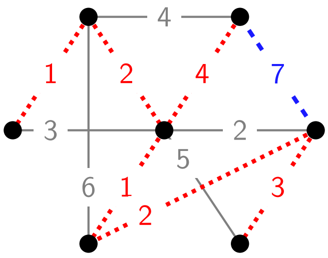


Input: A connected weighted graph $G = ((V, E), w)$. **Output:** A minimum spanning tree of G.

A **minimum spanning tree** is a subtree T of G covering all of G's vertices, whose total weight $\sum_{e \in E(T)} w(e)$ is as small as possible.

We are even more greedy than in Prim's algorithm.

Rather than picking the lowest-weight edge that grows our component, we just pick the lowest-weight edge **anywhere** that doesn't make a cycle.



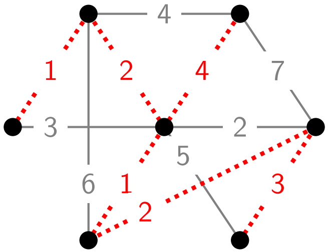
Kruskal's algorithm: The idea

Input: A connected weighted graph $G = ((V, E), w)$. **Output:** A minimum spanning tree of G.

A **minimum spanning tree** is a subtree T of G covering all of G's vertices, whose total weight $\sum_{e \in E(T)} w(e)$ is as small as possible.

We are even more greedy than in Prim's algorithm.

Rather than picking the lowest-weight edge that grows our component, we just pick the lowest-weight edge **anywhere** that doesn't make a cycle.



Kruskal's algorithm: Formal version and correctness

Input: A connected weighted graph $G = ((V, E), w)$. **Output:** A minimum spanning tree of G.

A **minimum spanning tree** is a subtree T of G covering all of G's vertices, whose total weight $\sum_{e \in E(T)} w(e)$ is as small as possible.

Kruskal's algorithm: Formal version and correctness

Input: A connected weighted graph $G = ((V, E), w)$. **Output:** A minimum spanning tree of G.

A **minimum spanning tree** is a subtree T of G covering all of G's vertices, whose total weight $\sum_{e \in E(T)} w(e)$ is as small as possible.

Formally: Let $e_1, \ldots, e_m$ be the edges of G, with $w(e_1) \leq \cdots \leq w(e_m)$.

Let $T_0 = (V, \emptyset)$ be the empty graph on V.
Given T_i, let $T_{i+1} = T_i + e_{i+1}$ if this is a forest, or T_i otherwise.

Kruskal's algorithm is to calculate and return T_m. Why does this work?

Kruskal's algorithm: Formal version and correctness

Input: A connected weighted graph $G = ((V, E), w)$. **Output:** A minimum spanning tree of G.

A **minimum spanning tree** is a subtree T of G covering all of G's vertices, whose total weight $\sum_{e \in E(T)} w(e)$ is as small as possible.

Formally: Let $e_1, \dots, e_m$ be the edges of G, with $w(e_1) \leq \dots \leq w(e_m)$.

Let $T_0 = (V, \emptyset)$ be the empty graph on V.
Given T_i, let $T_{i+1} = T_i + e_{i+1}$ if this is a forest, or T_i otherwise.

Kruskal's algorithm is to calculate and return T_m. Why does this work?

$\boldsymbol{T_m}$ **is a spanning tree:** Suppose not, for a contradiction.

By construction, T_m has no cycles and $V(T_m) = V$, so T_m must have at least two components C_1 and C_2 (both of which are trees).

Since G is connected, it must contain an edge e_i between C_1 and C_2.
$T_m + e_i$ contains no cycles since C_1 and C_2 are trees, so nor does $T_{i-1} + e_i$, so we should have $e_i \in E(T_i)$ — a contradiction. $\square$

Kruskal's algorithm: Correctness II

$\boldsymbol{T_m}$ **is minimum:** Again we will use an exchange argument.

Let S be a minimum spanning tree with $S \neq T_m$. We will turn S into a tree S^+ with one more edge in common with T_m, and $w(S^+) \leq w(S)$.

By repeating the process, we prove: $w(S) \geq w(S^+) \geq \cdots \geq w(T_m)$, and we're done.

Kruskal's algorithm: Correctness II

Goal: Turn an arbitrary minimum spanning tree S into a new tree S^+, with one more edge in common with T_m and with $w(S^+) \le w(S)$.

Kruskal's algorithm: Correctness II

Goal: Turn an arbitrary minimum spanning tree S into a new tree S^+, with one more edge in common with T_m and with $w(S^+) \leq w(S)$.

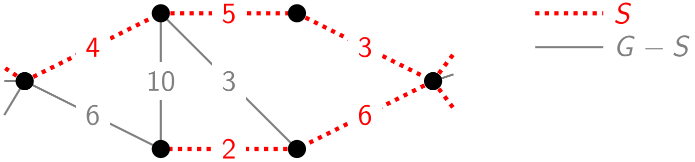
Key fact: If we add an edge to S, we create exactly one cycle C.
If we then remove any other edge from C, the result is a tree.
(See problem sheet.)

Kruskal's algorithm: Correctness II

Goal: Turn an arbitrary minimum spanning tree S into a new tree S^+, with one more edge in common with T_m and with $w(S^+) \leq w(S)$.

Key fact: If we add an edge to S, we create exactly one cycle C.
If we then remove any other edge from C, the result is a tree.
(See problem sheet.)

Since $T_m \neq S$ and both have $|V| - 1$ edges by the FLoT, there must be an edge $e \in E(T_m) \setminus E(S)$. Let C be the unique cycle in $S + e$.

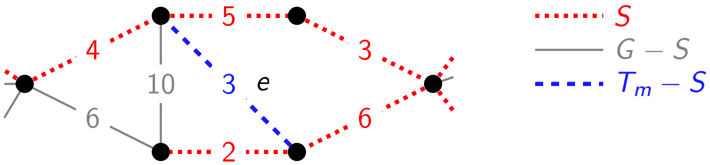


Kruskal's algorithm: Correctness II

Goal: Turn an arbitrary minimum spanning tree S into a new tree S^+, with one more edge in common with T_m and with $w(S^+) \leq w(S)$.

Key fact: If we add an edge to S, we create exactly one cycle C. If we then remove any other edge from C, the result is a tree. (See problem sheet.)

Since $T_m \neq S$ and both have $|V| - 1$ edges by the FLoT, there must be an edge $e \in E(T_m) \setminus E(S)$. Let C be the unique cycle in $S + e$.

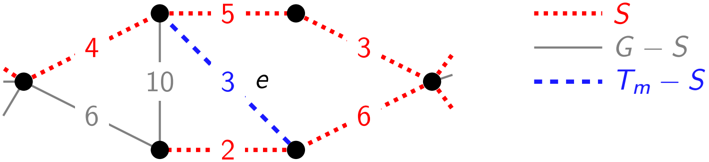


Kruskal's algorithm: Correctness II

Goal: Turn an arbitrary minimum spanning tree S into a new tree S^+, with one more edge in common with T_m and with $w(S^+) \leq w(S)$.

Key fact: If we add an edge to S, we create exactly one cycle C. If we then remove any other edge from C, the result is a tree. (See problem sheet.)

Since $T_m \neq S$ and both have $|V| - 1$ edges by the FLoT, there must be an edge $e \in E(T_m) \setminus E(S)$. Let C be the unique cycle in $S + e$.



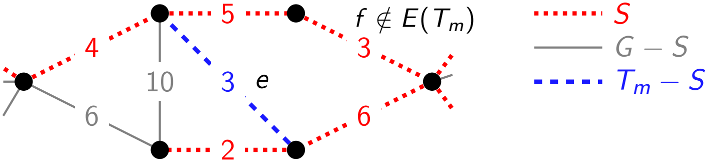
Since T_m has no cycles, there must be some edge $f \in E(C) \setminus E(T_m)$.

Kruskal's algorithm: Correctness II

Goal: Turn an arbitrary minimum spanning tree S into a new tree S^+, with one more edge in common with T_m and with $w(S^+) \leq w(S)$.

Key fact: If we add an edge to S, we create exactly one cycle C. If we then remove any other edge from C, the result is a tree. (See problem sheet.)

Since $T_m \neq S$ and both have $|V| - 1$ edges by the FLoT, there must be an edge $e \in E(T_m) \setminus E(S)$. Let C be the unique cycle in $S + e$.



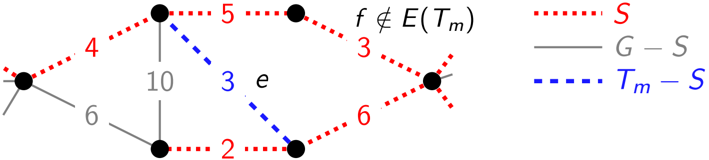
Since T_m has no cycles, there must be some edge $f \in E(C) \setminus E(T_m)$.

Kruskal's algorithm: Correctness II

Goal: Turn an arbitrary minimum spanning tree S into a new tree S^+, with one more edge in common with T_m and with $w(S^+) \leq w(S)$.

Key fact: If we add an edge to S, we create exactly one cycle C. If we then remove any other edge from C, the result is a tree. (See problem sheet.)

Since $T_m \neq S$ and both have $|V| - 1$ edges by the FLoT, there must be an edge $e \in E(T_m) \setminus E(S)$. Let C be the unique cycle in $S + e$.



Since T_m has no cycles, there must be some edge $f \in E(C) \setminus E(T_m)$.

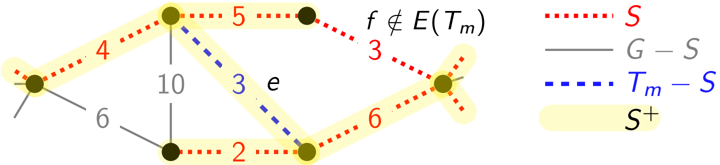
Since Kruskal's algorithm added e instead of f, we have $w(e) \leq w(f)$. We therefore take $S^+ = S - f + e$.

Kruskal's algorithm: Correctness II

Goal: Turn an arbitrary minimum spanning tree S into a new tree S^+, with one more edge in common with T_m and with $w(S^+) \leq w(S)$.

Key fact: If we add an edge to S, we create exactly one cycle C.
If we then remove any other edge from C, the result is a tree.
(See problem sheet.)

Since $T_m \neq S$ and both have $|V| - 1$ edges by the FLoT, there must be an edge $e \in E(T_m) \setminus E(S)$. Let C be the unique cycle in $S + e$.



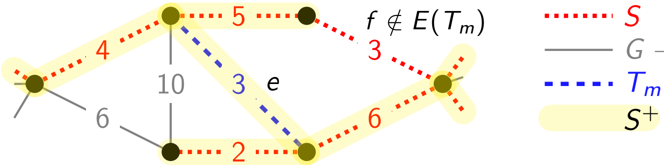
Since T_m has no cycles, there must be some edge $f \in E(C) \setminus E(T_m)$.

Since Kruskal's algorithm added e instead of f, we have $w(e) \leq w(f)$.
We therefore take $S^+ = S - f + e$.

Goal: Turn an arbitrary minimum spanning tree S into a new tree S^+, with one more edge in common with T_m and with $w(S^+) \leq w(S)$.

Key fact: If we add an edge to S, we create exactly one cycle C. If we then remove any other edge from C, the result is a tree. (See problem sheet.)

Since $T_m \neq S$ and both have $|V| - 1$ edges by the FLoT, there must be an edge $e \in E(T_m) \setminus E(S)$. Let C be the unique cycle in $S + e$.



Since T_m has no cycles, there must be some edge $f \in E(C) \setminus E(T_m)$.

Since Kruskal's algorithm added e instead of f, we have $w(e) \leq w(f)$. We therefore take $S^+ = S - f + e$. □