Lecture 14 — Minimum Spanning Trees (Part 2)

COMPSCI 308 — Design and Analysis of Algorithms

Xing Shi Cai https://newptcai.gitlab.io

Duke Kunshan University, Kunshan, China

Jan-Mar 2026

Summary

Assignments[1]

Practice makes perfect!

📖 Required Readings:

- Section 21.2.

✏️ Required Exercises:

- Exercises 21.2 — 1–6.
- Problem 21.1.a.

[1] 💣 Assignments will not be collected; however, quiz problems will be selected from them. (This includes both Readings and Exercises.)

ITA 21.2 The Algorithms of Kruskal and Prim

⏰ Review — Generic Algorithm for MST

The algorithms of Kruskal and Prim are based on the generic algorithm for Minimum Spanning Tree (MST) —

Generic-MST(G, w)

1: $A \leftarrow \emptyset$ ← Subset of some MST
2: **while** A does not form a spanning tree **do**
3: find an edge (u, v) that is *safe* for A ← How to find a safe edge for A.
4: $A \leftarrow A \cup \{(u, v)\}$ ← add (u, v).
5: **return** A

ITA 21.2 The Algorithms of Kruskal and Prim

Kruskal's Algorithm

👀 Review — Generic Algorithm for MST

Let A be a subset of edges in some MST of G.

An edge is a safe edge for A if adding it keeps A extendable to some minimum spanning tree; equivalently, $A \cup \{(u, v)\}$ remains a subset of an MST.

Line 3 maintains the following loop invariant — A is *always* a subset of some minimum spanning tree.

Generic-MST(G, w)

1: $A \leftarrow \varnothing$
2: **while** A does not form a spanning tree **do**
3: find an edge (u, v) that is safe for A
4: $A \leftarrow A \cup \{(u, v)\}$
5: **return** A

Overview of Kruskal's Algorithm

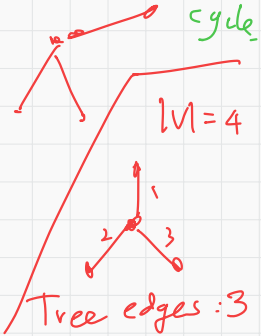
Kruskal's algorithm works as follows —

1. Sort all graph edges in non-decreasing order of weight.
2. Start with $A = \emptyset$ and each vertex in its own component.
3. Consider edges in order; add an edge (u, v) to A only if u and v are in different components.
4. Stop when $|A| = |V| - 1$.

🤔 How to show this algorithm is correct?

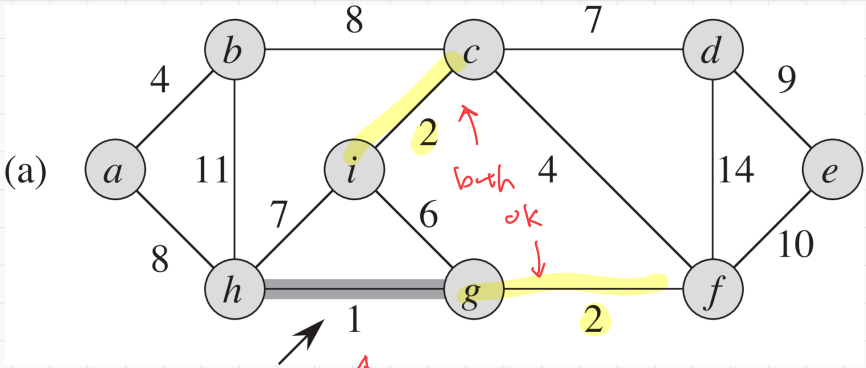
does not create a cycle

Not going to create cycle

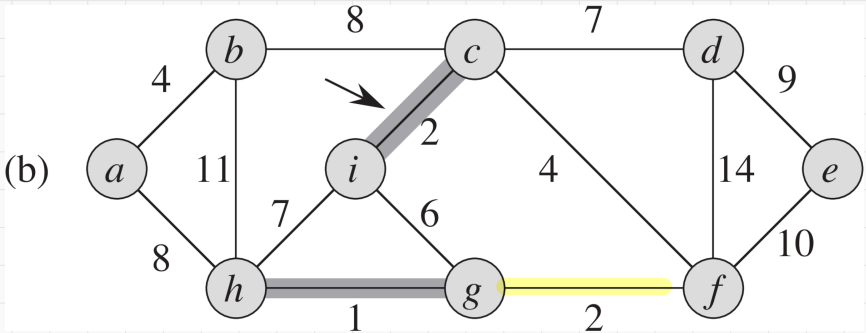


This must be a tree since it does not have any cycles.

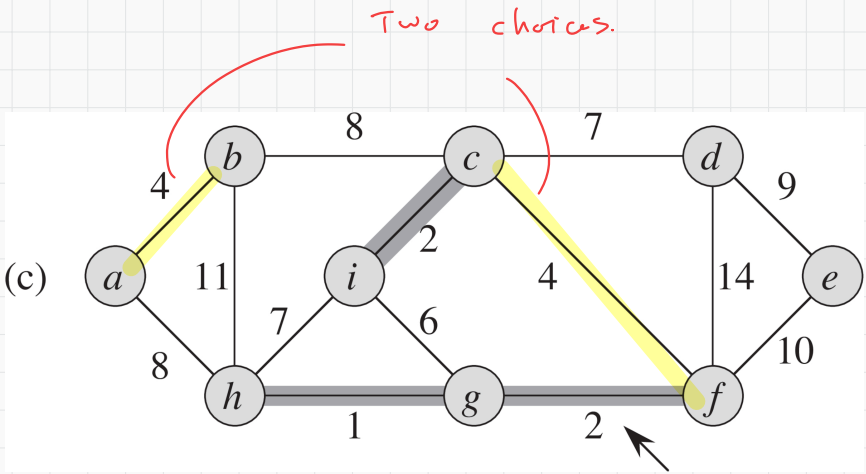
Example Run of Kruskal's Algorithm



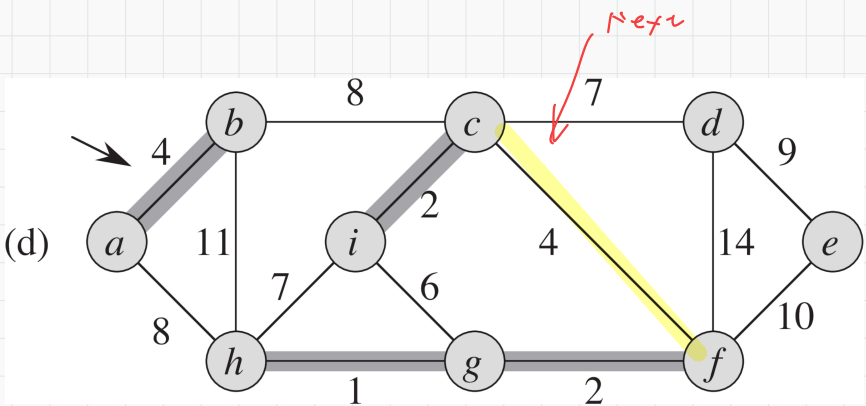
Example Run of Kruskal's Algorithm



Example Run of Kruskal's Algorithm

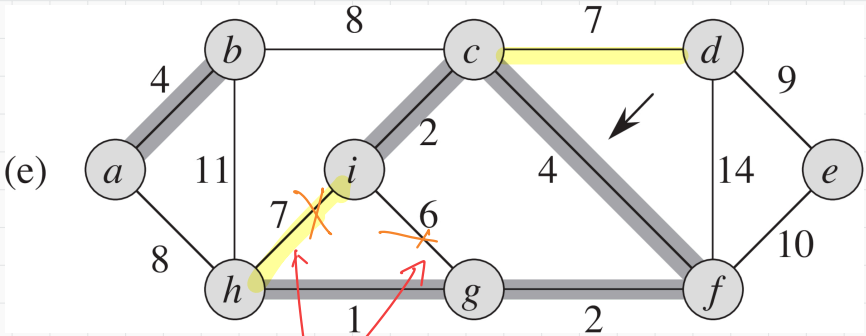


Example Run of Kruskal's Algorithm



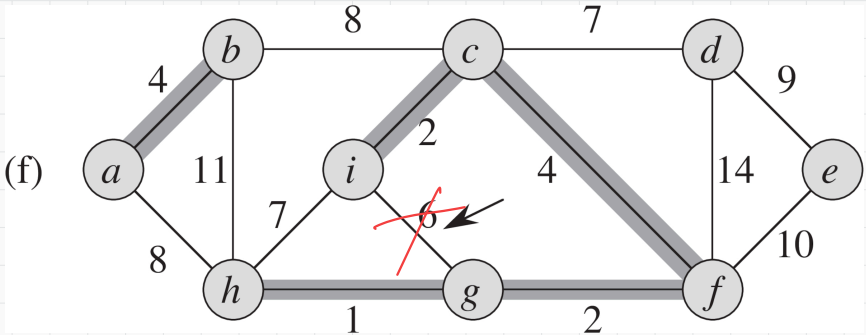
Example Run of Kruskal's Algorithm

ok.

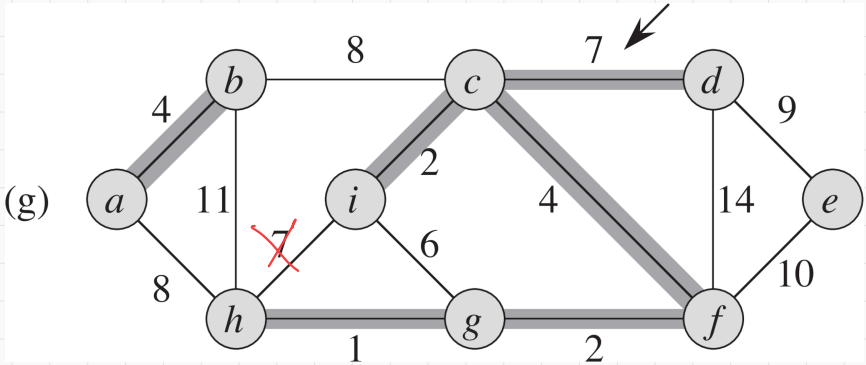


No! This creates a cycle. Don't need to check this any more

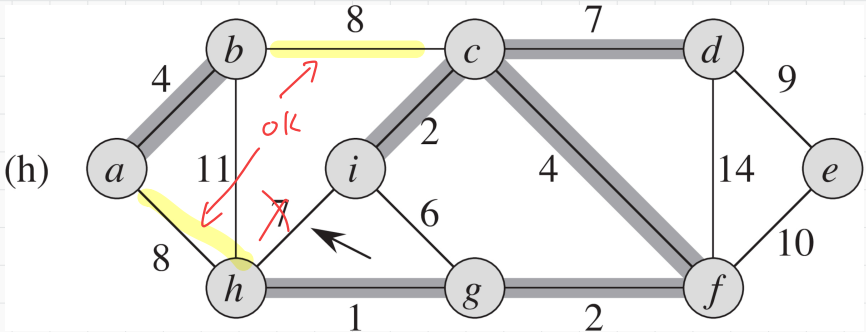
📌 Example Run of Kruskal's Algorithm



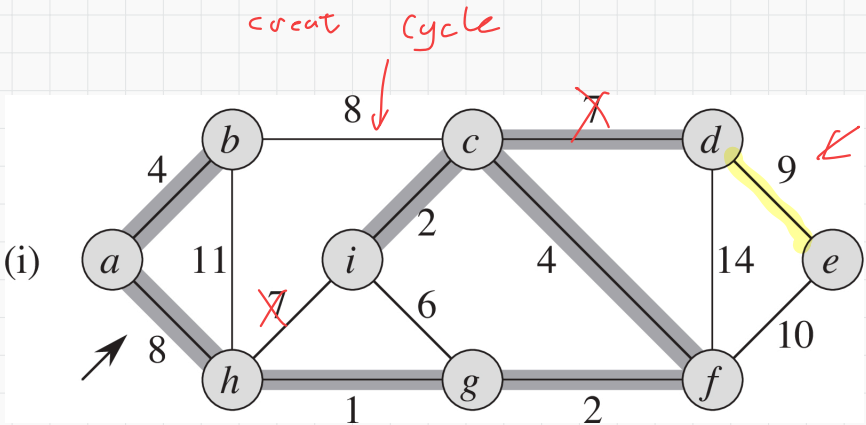
Example Run of Kruskal's Algorithm



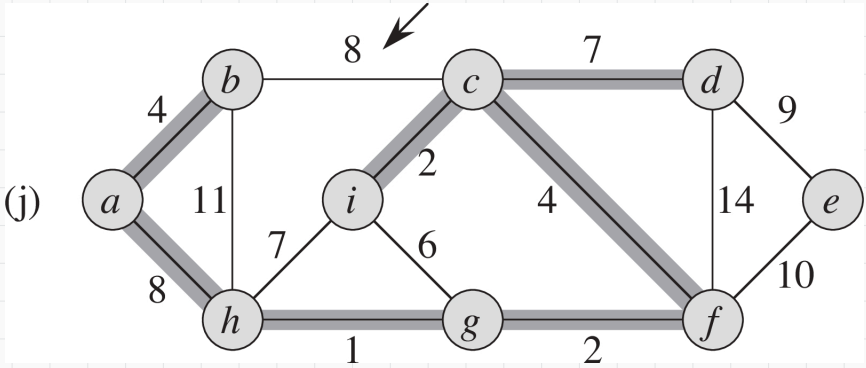
Example Run of Kruskal's Algorithm



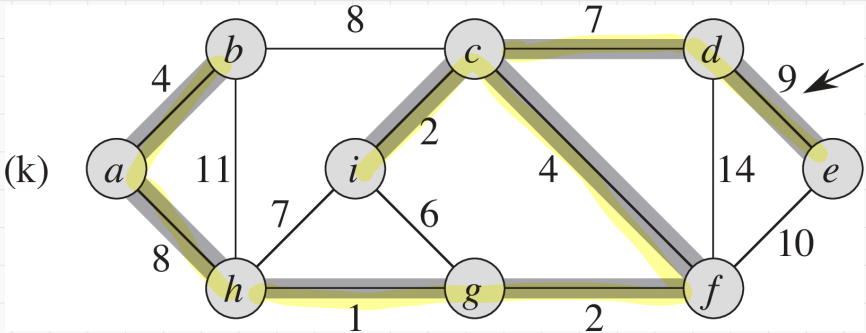
Example Run of Kruskal's Algorithm



📌 Example Run of Kruskal's Algorithm

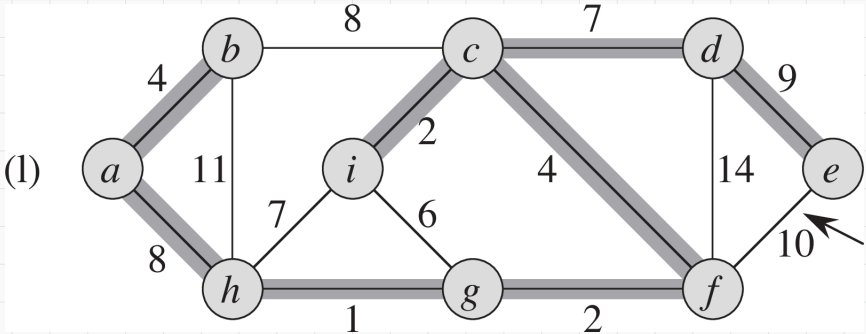


Example Run of Kruskal's Algorithm

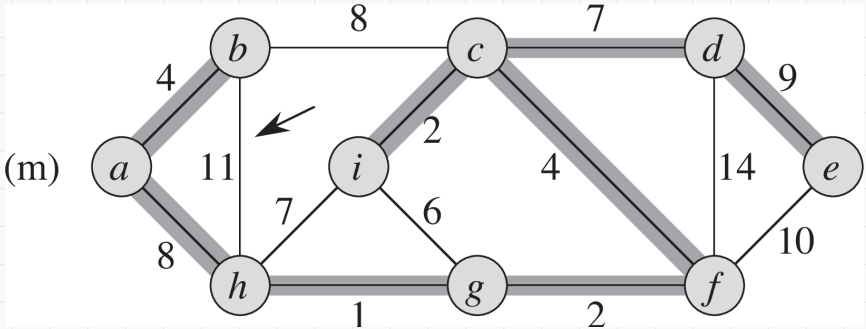


This is a greedy algorithm

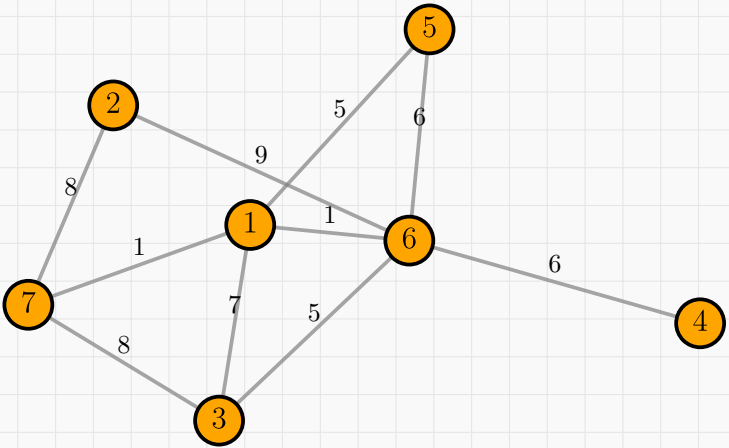
📌 Example Run of Kruskal's Algorithm



📌 Example Run of Kruskal's Algorithm



🍰 Run Kruskal's algorithm on the following graph to determine the weight of its MST?



⏰ Review — Corollary 21.2

Let A be a subset of the edges of a MST. Then A is a forest.

Take any component of A as a cut. A light edge for the cut is safe.

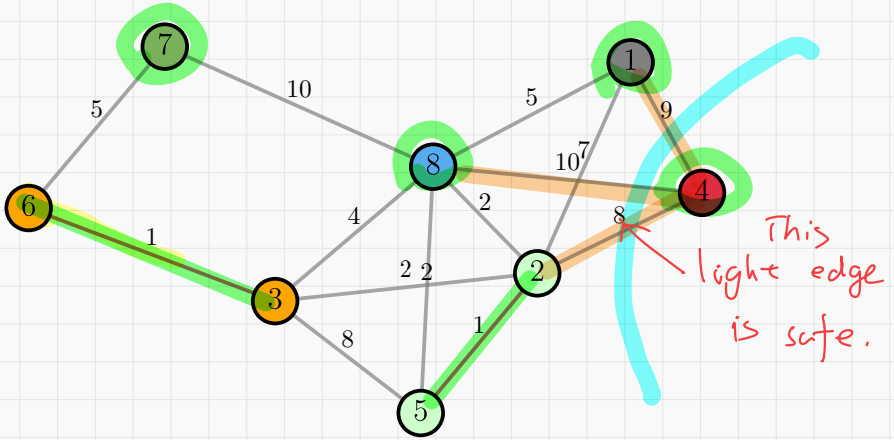


Figure 1: 🍰 What are the light edges for $S = \{5, 2\}$?

⏰ Review — Corollary 21.2

Let A be a subset of the edges of a MST. Then A is a forest.

Take any component of A as a cut. A light edge for the cut is safe.

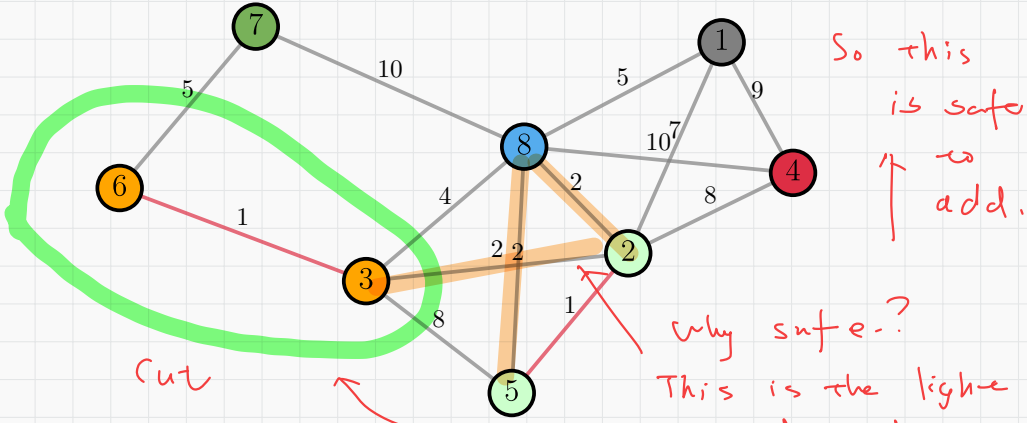


Figure 1: 🍦 What are the light edges for $S = \{3, 6\}$?

⏰ Review — Corollary 21.2

Let A be a subset of the edges of a MST. Then A is a forest.

Take any component of A as a cut. A light edge for the cut is safe.

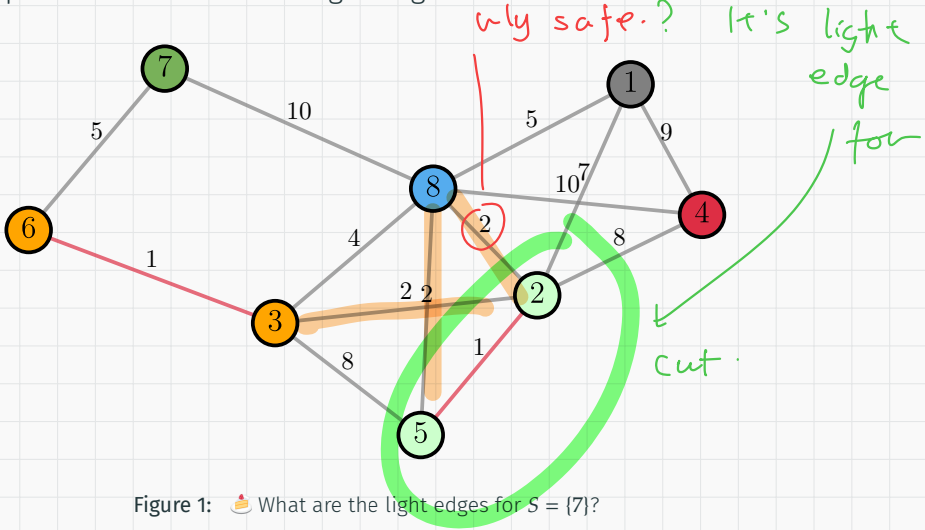


Figure 1: 🥧 What are the light edges for $S = \{7\}$?

✅ Correctness of Kruskal's Algorithm

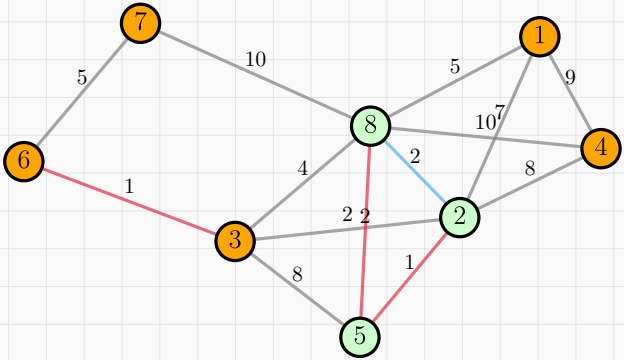


Figure 2: 🍰 Why is $(2, 8)$ *not* safe?

✅ Correctness of Kruskal's Algorithm

When an edge (u, v) added to A does not create a cycle in A, it must connect two different components in A.

If it is a *light edge* with this property, then by Corollary 21.2 it is safe.

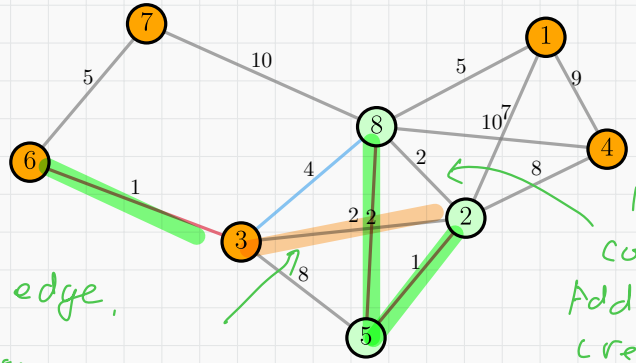


Figure 2: Why is (2, 3) safe?

Assuming G is connected, then we have

$$|V| - 1 \leq |E| \leq \binom{|V|}{2}.$$[2]

$\frac{|V|(|V|-1)}{2}$

[2] The textbook often uses V to mean $|V|$, E to mean $|E|$, etc., which should be clear from the context.

Assuming G is connected, then we have

$$|V| - 1 \le |E| \le \binom{|V|}{2}.^{2}$$

Let us assume that Kruskal's algorithm uses the disjoint-set data structures which provide —

- MAKE-SET(v),
- FIND-SET(v)
- UNION(u, v),

each of which takes $O(\log |V|) = O(\log |E|)$ time.

[2] The textbook often uses V to mean $|V|$, E to mean $|E|$, etc., which should be clear from the context.

Pseudocode for Kruskal's Algorithm

Let's show the complexity is $O(|E| \log |V|)$ for Kruskal's Algorithm.

```
Kruskal(G, w)
1:  A ← ∅                                                    ▷ O(1)
2:  for each vertex v ∈ G.V do
3:      MAKE-SET(v)                                          ▷ O(V + E)
4:  Create a single list of the edges in G.E                 ▷ O(E log E)
5:  Sort the edges of G.E into nondecreasing order by w      ▷ O(Eα(V))
6:  for each edge (u, v) ∈ G.E do
7:      if FIND-SET(u) ≠ FIND-SET(v) then
8:          A ← A ∪ {(u, v)}
9:          UNION(u, v)
10: return A                                                 ▷ Total: O(E log E)
```

Graph.

weight.

$O(|V| \cdot \log |V|) = O(|E| \log |V|)$.

At the beginning, each vertex is itself a component

$O(|E| \cdot \log |V|)$

u and v are in two different component.

The graph uses adjacent list.

$O(|E| \log |E|) = O(|E| \log |V|)$.

ITA 21.2 The Algorithms of Kruskal and Prim

Prim's Algorithm

Overview of Prim's Algorithm

Prim's algorithm constructs the MST as follows —

- Start with a single root vertex r and let T be the current tree.
- At each step, consider the cut $(T, V \setminus T)$ and add the minimum-weight edge that crosses the cut.
- The added edge brings a new vertex into T.
- Repeat until T spans all vertices.

Eventually a MST

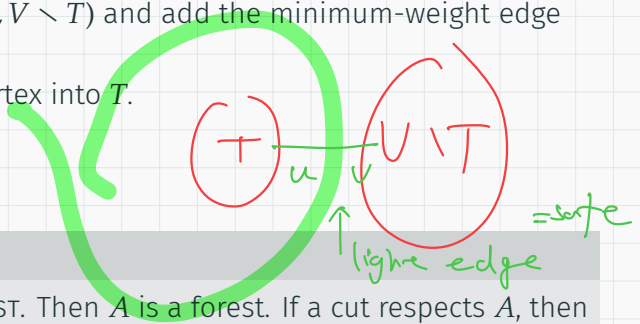
light edge

safe edge

Overview of Prim's Algorithm

Prim's algorithm constructs the MST as follows —

- Start with a single root vertex r and let T be the current tree.
- At each step, consider the cut $(T, V \setminus T)$ and add the minimum-weight edge that crosses the cut.
- The added edge brings a new vertex into T.
- Repeat until T spans all vertices.

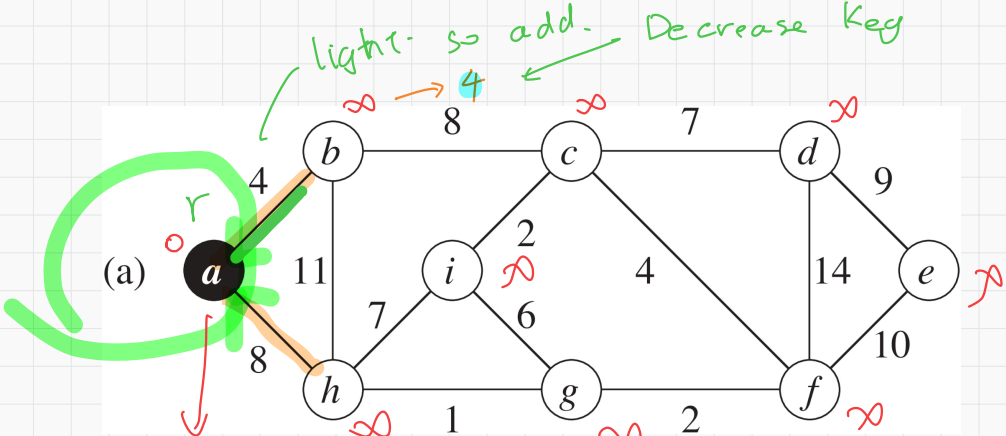


The correctness is guaranteed by —

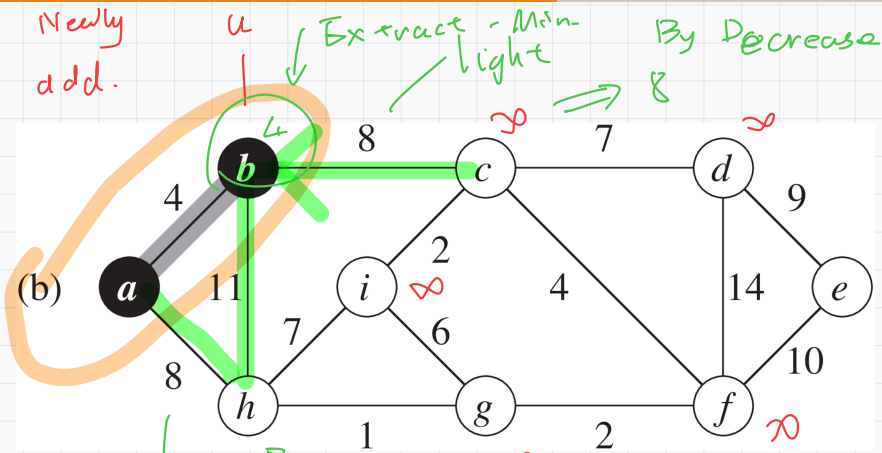
Corollary 21.3

Let A be a subset of the edges of a MST. Then A is a forest. If a cut respects A, then a light edge crossing the cut is safe.

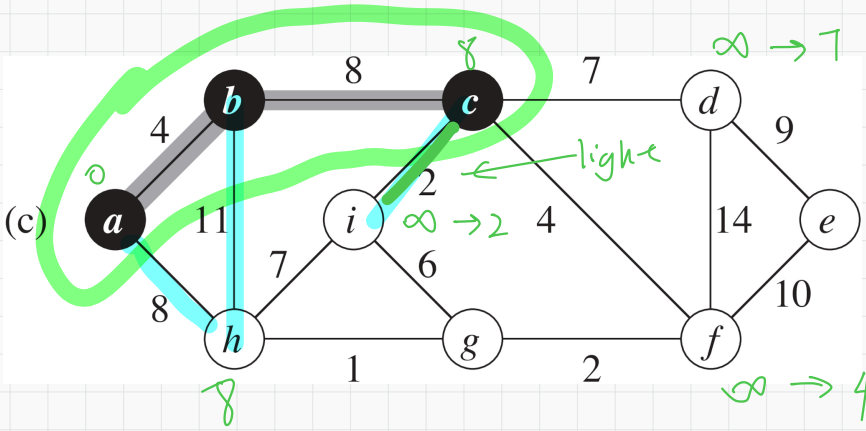
Example Run of Prim's Algorithm



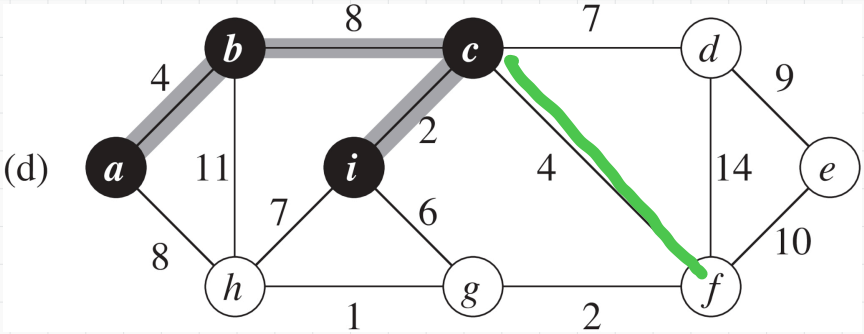
Example Run of Prim's Algorithm



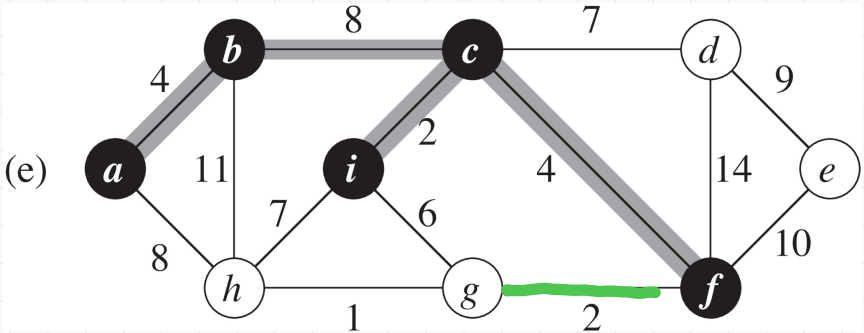
Example Run of Prim's Algorithm



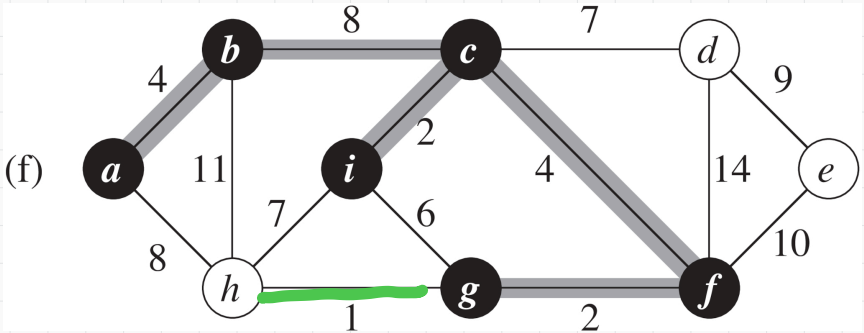
📌 Example Run of Prim's Algorithm



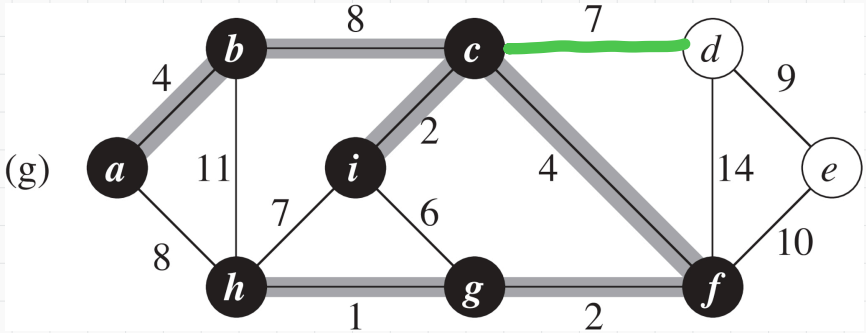
Example Run of Prim's Algorithm



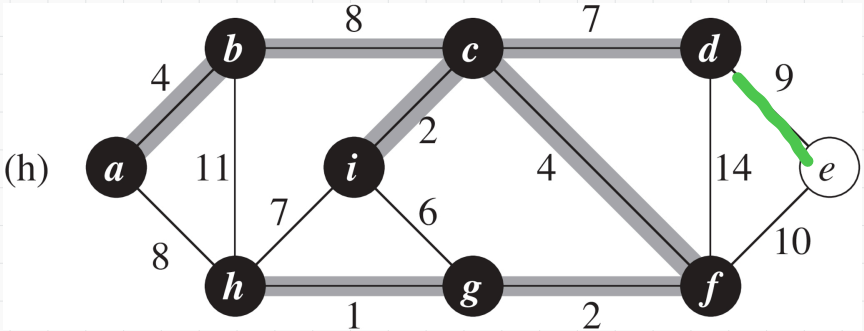
Example Run of Prim's Algorithm



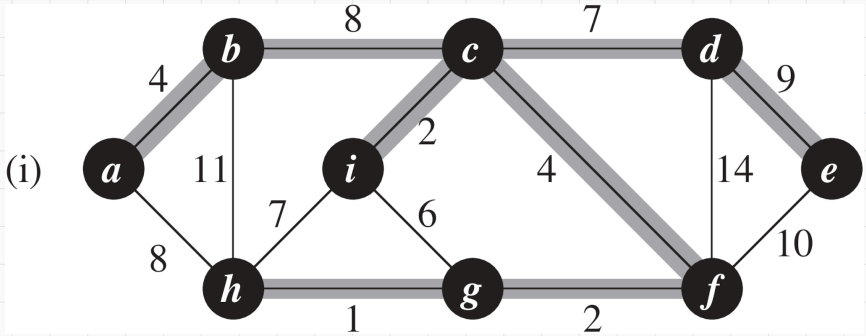
Example Run of Prim's Algorithm



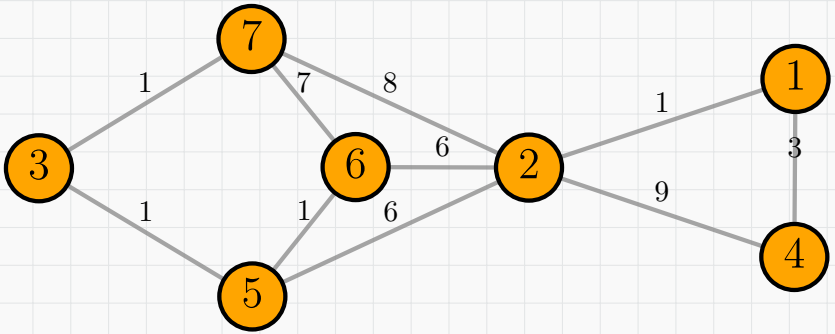
📌 Example Run of Prim's Algorithm



📌 Example Run of Prim's Algorithm



🍰 Run Prim's algorithm on the following graph with $r = 7$.



🔧 Min-Priority Queue

A min-priority queue (with a binary min-heap implementation) is a data structure that provides the following operations —

- Build-Queue(V) — build a queue in $O(|V|)$ time.
- Extract-Min(Q) — remove and return the element with the lowest `key` in $O(\log |V|)$ time.
- Decrease-Key($Q, u,$ `newKey`) — reduce the `key` of u to `newKey` in $O(\log |V|)$ time.

💡 The min-priority queue helps us to find the light edge to add.

Pseudocode for Prim's Algorithm

```
Prim(G, w, r)
1:  let B[1..|V|] be a new bit array
2:  for each u ∈ G.V do                                   ▷ O(V)
3:      u.key = ∞
4:      u.π = NIL
5:      B[u] = True
6:  r.key = 0
7:  Q = BUILD-QUEUE(G.V)                                  ▷ O(V)
8:  while Q ≠ ∅ do
9:      u = EXTRACT-MIN(Q)                                ▷ O(V log V)
10:     let B[u] = False
11:     for each v ∈ G.Adj[u] do
12:         if B[v] == True and w(u, v) < v.key then
13:             v.π = u
14:             DECREASE-KEY(Q, v, w(u, v))               ▷ O(E log V)
                                                          ▷ Total: O(E log V)
```

Stores if v is in T.

Initial estimate of dist to T.

u.π will be the vertex connect u to T.

u is not in T.

r is in T

All vertexes are in T

The light edge.

u is in T now.

for every neighbour of u, update estimate to T.

not in T

edge weight from u to

Time Complexity of Prim's Algorithm

The running time depends on the implementation of the priority queue Q.

- Using a binary min-heap:
  - Build-Queue: $O(V)$.
  - Extract-Min: Executed $|V|$ times, each takes $O(\log V)$. Total: $O(V \log V)$.
  - Decrease-Key: Executed $O(E)$ times, each takes $O(\log V)$. Total: $O(E \log V)$.
- Total time: $O(V \log V + E \log V) = O(E \log V)$.
- This is asymptotically the same as Kruskal's algorithm.

Implementation details:

- Map between vertices and heap elements.
- Membership check in Q takes $O(1)$ using a bit array.

✅ Correctness of Prim's Algorithm

Prior to each iteration of the `while` loop, three things are always true —

1. $A = \{(v, v.\pi) : v \in V - \{r\} - Q\}$.

✅ Correctness of Prim's Algorithm

Prior to each iteration of the `while` loop, three things are always true —

1. $A = \{(v, v.\pi) : v \in V - \{r\} - Q\}$.
2. The vertices already placed into the MST are those in $V - Q$.

✅ Correctness of Prim's Algorithm

Prior to each iteration of the `while` loop, three things are always true —

1. $A = \{(v, v.\pi) : v \in V - \{r\} - Q\}$.
2. The vertices already placed into the MST are those in $V - Q$.
3. For all vertices v in Q, if $v.\pi \neq$ `NIL`, then

✅ Correctness of Prim's Algorithm

Prior to each iteration of the `while` loop, three things are always true —

1. $A = \{(v, v.\pi) : v \in V - \{r\} - Q\}$.
2. The vertices already placed into the MST are those in $V - Q$.
3. For all vertices v in Q, if $v.\pi \neq$ `NIL`, then
   - $v.key < \infty$

✅ Correctness of Prim's Algorithm

Prior to each iteration of the `while` loop, three things are always true —

1. $A = \{(v, v.\pi) : v \in V - \{r\} - Q\}$.
2. The vertices already placed into the MST are those in $V - Q$.
3. For all vertices v in Q, if $v.\pi \neq$ `NIL`, then
   - $v.key < \infty$
   - and $v.key$ is the weight of $(v, v.\pi)$ connecting v to a vertex which is in the MST,

✅ Correctness of Prim's Algorithm

Prior to each iteration of the `while` loop, three things are always true —

1. $A = \{(v, v.\pi) : v \in V - \{r\} - Q\}$.
2. The vertices already placed into the MST are those in $V - Q$.
3. For all vertices v in Q, if $v.\pi \neq$ `NIL`, then
   - $v.key < \infty$
   - and $v.key$ is the weight of $(v, v.\pi)$ connecting v to a vertex which is in the MST,
   - and $(v, v.\pi)$ has the lightest weight among all such edges.

✅ Correctness of Prim's Algorithm

Prior to each iteration of the `while` loop, three things are always true —

1. $A = \{(v, v.\pi) : v \in V - \{r\} - Q\}$.
2. The vertices already placed into the MST are those in $V - Q$.
3. For all vertices v in Q, if $v.\pi \neq$ `NIL`, then
   - $v.key < \infty$
   - and $v.key$ is the weight of $(v, v.\pi)$ connecting v to a vertex which is in the MST,
   - and $(v, v.\pi)$ has the lightest weight among all such edges.

💡 So each time a vertex u is dequeued, it is guaranteed that $(u.\pi, u)$ is a safe edge.

ITA 21.2 The Algorithms of Kruskal and Prim

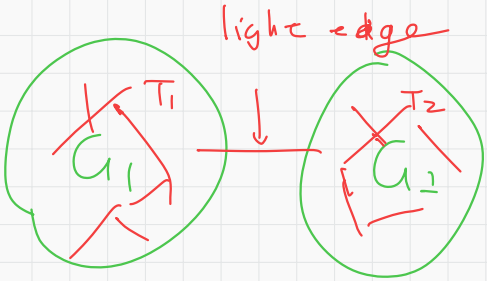
Exercise 21.2

Exercise 21.2-6

Consider a divide-and-conquer algorithm for MST

1. Partition vertex set V of graph $G = (V, E)$ into V_1 and V_2 such that $|V_1|$ and $|V_2|$ differ by at most 1.
2. Define E_1 as edges incident only on V_1, and E_2 as edges incident only on V_2.
3. Recursively compute MST for subgraphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$.
4. Select the lightest edge in E that connects V_1 and V_2 to merge two MSTs.

🍰 Do you think this algorithm is correct?



Exercise 21.2-6

- What are the MSTs for the green and orange subgraphs?
- What is the MSTs for the whole graph?

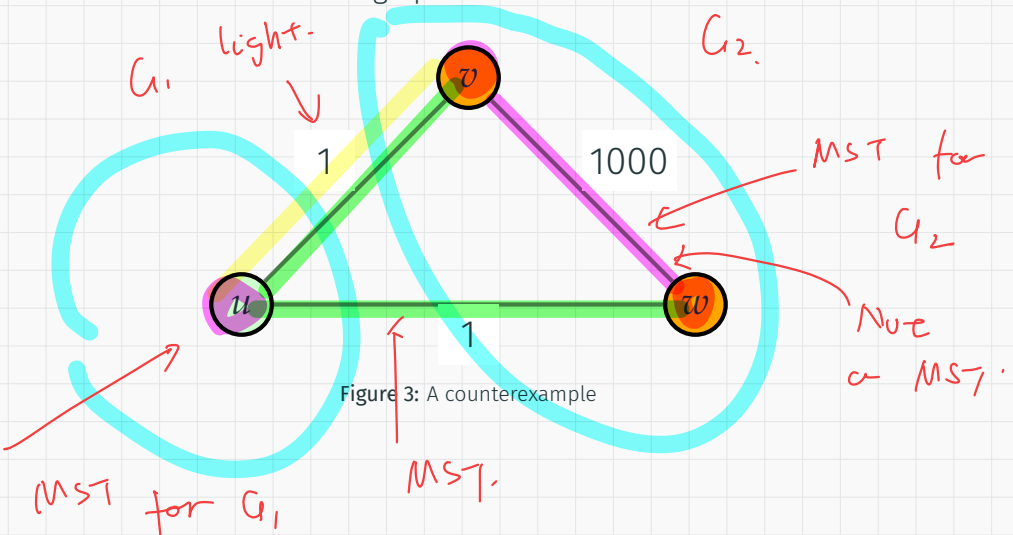


Figure 3: A counterexample

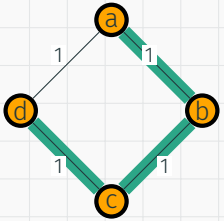
Exercise 21.2-1

Kruskal's algorithm can return different spanning trees for the same graph G when the edge weights tie. Show that for every minimum spanning tree T of G, there is a way to order the edges so Kruskal's algorithm returns T.

Exercise 21.2-1

Kruskal's algorithm can return different spanning trees for the same graph G when the edge weights tie. Show that for every minimum spanning tree T of G, there is a way to order the edges so Kruskal's algorithm returns T.

Answer: Order the edges by nondecreasing weight, breaking ties so that edges of T are considered before any other edges with the same weight.



(a,b), (b,c), (c,d), (a,d)

One order to get this tree.

Any order with (a,d) in the end works

Figure 4: How to order the edges in this graph to get the highlighted MST?

ITA 23 Problems

Problem 23-1 — Uniqueness of mst

Let $G = (V, E)$ be a connected graph with distinct edge weights. Show that T, the mst of G, is unique.

Proof: Recall that —

Exercise 21.1-8

Let T and T' be the mst of some graph G. Let L and L' be *sorted* list of the weights of the edges of T and T', respectively. Then $L = L'$.

If G has two distinct mst T and T', then they must differ by at least one edge.

As the edge weights are unique, it is not possible that L and L' are the same. This is a contradiction.

Problem 23-1: Non-uniqueness of second-best mst

Let $\mathcal{T}$ be the set spanning trees of G.

The second-best MST of G is the spanning tree T' with

$$w(T') = \min_{T'' \in \mathcal{T} - \{T\}} w(T'')$$

where T is the unique MST of G.

Show that the second-best MST of G is not necessarily unique.

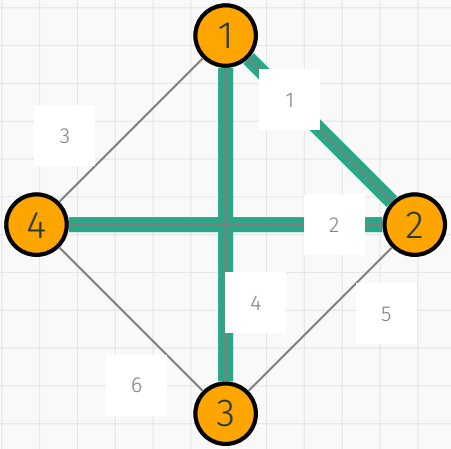


Figure 5: 🍰

Problem 23-1: Non-uniqueness of second-best MST

Let $\mathcal{T}$ be the set spanning trees of G.

The second-best MST of G is the spanning tree T' with

$$w(T') = \min_{T'' \in \mathcal{T} - \{T\}} w(T'')$$

where T is the unique MST of G.

Show that the second-best MST of G is not necessarily unique.

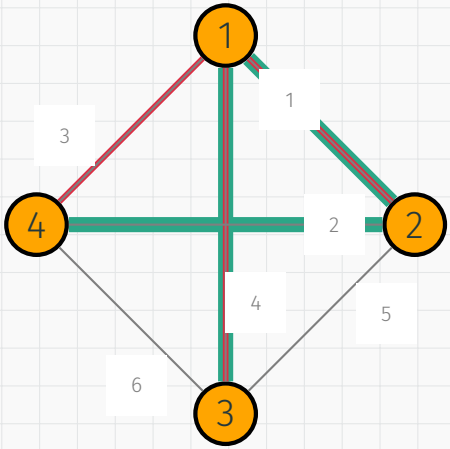


Figure 5: What is the MST of G?

Problem 23-1: Non-uniqueness of second-best MST

Let $\mathcal{T}$ be the set spanning trees of G.

The second-best MST of G is the spanning tree T' with

$$w(T') = \min_{T'' \in \mathcal{T} - \{T\}} w(T'')$$

where T is the unique MST of G.

Show that the second-best MST of G is not necessarily unique.

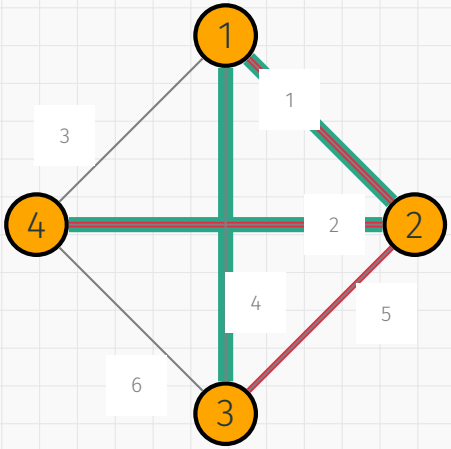


Figure 5: What is one second-best MST of G?

Problem 23-1: Non-uniqueness of second-best MST

Let $\mathcal{T}$ be the set spanning trees of G.

The second-best MST of G is the spanning tree T' with

$$w(T') = \min_{T'' \in \mathcal{T} - \{T\}} w(T'')$$

where T is the unique MST of G.

Show that the second-best MST of G is not necessarily unique.

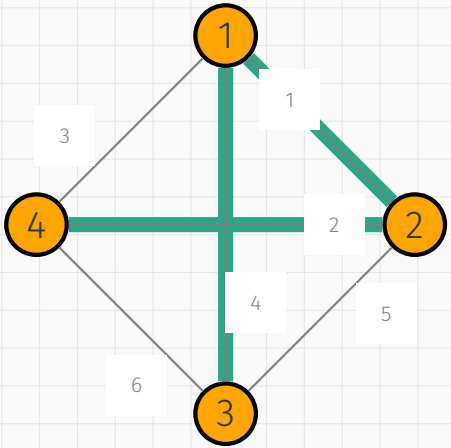


Figure 5: What is another second-best MST of G?

What are your main takeaways today? Any questions?