Lecture 13: Minimum Spanning Trees

Rafael Oliveira

University of Waterloo
Cheriton School of Computer Science

rafael.oliveira.teaching@gmail.com

October 26, 2023

Overview

- Minimum Spanning Trees
  - Boruvka's Algorithm
  - Prim's Algorithm
  - Kruskal's algorithm
  - Reverse-Delete
- Acknowledgements

Minimum Spanning Trees (MST)

- **Input:** undirected (connected) weighted graph $G(V, E, w)$, where $w : E \to \mathbb{R}_{>0}$

  Will assume $n = O(m)$, since our graph is connected.

- **Output:** A minimum weight spanning tree T, where

$$w(T) := \sum_{e \in T} w_e$$

Minimum Spanning Trees (MST)

- **Input:** undirected (connected) weighted graph $G(V, E, w)$, where $w : E \to \mathbb{R}_{>0}$

  Will assume $n = O(m)$, since our graph is connected.

- **Output:** A minimum weight spanning tree T, where

$$w(T) := \sum_{e \in T} w_e$$

- Cheapest way to build a connected subgraph
- **Observation:** when $w_e > 0$, note that any optimal solution must be an MST

  Property 1: Removing edge of cycle cannot disconnect the graph.

Minimum Spanning Trees (MST)

- **Input:** undirected (connected) weighted graph $G(V, E, w)$, where $w : E \to \mathbb{R}_{>0}$

  Will assume $n = O(m)$, since our graph is connected.

- **Output:** A minimum weight spanning tree T, where

$$w(T) := \sum_{e \in T} w_e$$

- Cheapest way to build a connected subgraph
- **Observation:** when $w_e > 0$, note that any optimal solution must be an MST

  Property 1: Removing edge of cycle cannot disconnect the graph.

- Very tempting to choose edge of minimum weight, will this work?

Cheapest Edge Lemma

Lemma (Cheapest Edge)

There is an MST which contains an edge of minimum weight.

Cheapest Edge Lemma

Lemma (Cheapest Edge)

There is an MST which contains an edge of minimum weight.

- Let $e = \{u, v\}$ be a cheapest edge, and T be an MST. If $e \in T$, we are done, so suppose that is not the case.

Cheapest Edge Lemma

Lemma (Cheapest Edge)

There is an MST which contains an edge of minimum weight.

- Let $e = \{u, v\}$ be a cheapest edge, and T be an MST. If $e \in T$, we are done, so suppose that is not the case.
- Let $H = T + e$. Note that H contains a unique cycle (& contains e).

Cheapest Edge Lemma

Lemma (Cheapest Edge)

There is an MST which contains an edge of minimum weight.

- Let $e = \{u, v\}$ be a cheapest edge, and T be an MST. If $e \in T$, we are done, so suppose that is not the case.
- Let $H = T + e$. Note that H contains a unique cycle (& contains e).
- Let $f \in H \setminus e$ be any other edge in the above cycle. Then we have $H - f$ is connected by property 1. Hence, $H \setminus f$ is a spanning tree.

Cheapest Edge Lemma

Lemma (Cheapest Edge)

There is an MST which contains an edge of minimum weight.

- Let $e = \{u, v\}$ be a cheapest edge, and T be an MST. If $e \in T$, we are done, so suppose that is not the case.
- Let $H = T + e$. Note that H contains a unique cycle (& contains e).
- Let $f \in H \setminus e$ be any other edge in the above cycle. Then we have $H - f$ is connected by property 1. Hence, $H \setminus f$ is a spanning tree.
- As e is a cheapest edge, we have

$$w(H \setminus f) = w(H) - w(f) = w(T) + w(e) - w(f) \leq w(T)$$

as we assumed T is MST, we must have $H \setminus f$ also MST.

Cheapest Edge on a Vertex

Lemma (Cheapest Edge on a Vertex)

For each $u \in V$, there is an MST containing cheapest edge incident on u.

- Proof is identical to previous lemma.

Greedy Algorithms

- Note that the cheapest edge lemmas give an efficient algorithm (greedy) to construct an MST

  Find cheapest edge $e = \{u, v\}$, and "contract" vertices u, v, obtaining a graph with one less vertex.

Greedy Algorithms

- Note that the cheapest edge lemmas give an efficient algorithm (greedy) to construct an MST

  Find cheapest edge $e = \{u, v\}$, and "contract" vertices u, v, obtaining a graph with one less vertex.

- **Boruvka's algorithm:**
  1. Perform the following operations until we have one vertex left
     - for each vertex in the graph, find its edge of minimum cost.
     - build a forest with these selected edges[1]
     - contract the connected components of this forest

[1] For simplicity, assuming weights are distinct, so we don't need to break ties

Greedy Algorithms

- Note that the cheapest edge lemmas give an efficient algorithm (greedy) to construct an MST

  Find cheapest edge $e = \{u, v\}$, and "contract" vertices u, v, obtaining a graph with one less vertex.

- **Boruvka's algorithm:**
  1. Perform the following operations until we have one vertex left
     - for each vertex in the graph, find its edge of minimum cost.
     - build a forest with these selected edges
     - contract the connected components of this forest
- each iteration of the above algorithm (Boruvka step), takes $O(m)$ time to complete

Greedy Algorithms

- Note that the cheapest edge lemmas give an efficient algorithm (greedy) to construct an MST

  Find cheapest edge $e = \{u, v\}$, and "contract" vertices u, v, obtaining a graph with one less vertex.

- **Boruvka's algorithm:**
  1. Perform the following operations until we have one vertex left
     - for each vertex in the graph, find its edge of minimum cost.
     - build a forest with these selected edges
     - contract the connected components of this forest
- each iteration of the above algorithm (Boruvka step), takes $O(m)$ time to complete
- each Boruvka step at least halves the number of vertices

Greedy Algorithms

- Note that the cheapest edge lemmas give an efficient algorithm (greedy) to construct an MST

  Find cheapest edge $e = \{u, v\}$, and "contract" vertices u, v, obtaining a graph with one less vertex.

- **Boruvka's algorithm:**
  1. Perform the following operations until we have one vertex left
     - for each vertex in the graph, find its edge of minimum cost.
     - build a forest with these selected edges
     - contract the connected components of this forest
- each iteration of the above algorithm (Boruvka step), takes $O(m)$ time to complete
- each Boruvka step at least halves the number of vertices
- **Running time:** $O(m \log n)$.

Cheapest Edge in a Cut

- **Cut:** a *cut* in a graph is a bipartition of the vertex set

$$V = S \sqcup (S \setminus V)$$

The *edges* of the cut, denoted $\delta(S)$, is the set of edges $e = \{u, v\}$ with $u \in S$ and $v \notin S$

$$\delta(S) = \{\{u, v\} \in E \mid u \in S, v \notin S\}$$

Cheapest Edge in a Cut

- **Cut:** a *cut* in a graph is a bipartition of the vertex set

$$V = S \sqcup (S \setminus V)$$

The *edges* of the cut, denoted $\delta(S)$, is the set of edges $e = \{u, v\}$ with $u \in S$ and $v \notin S$

$$\delta(S) = \{\{u, v\} \in E \mid u \in S, v \notin S\}$$

Lemma (Cheapest Edge in Cut)

For every nonempty subset $\emptyset \neq S \subset V$, there is a MST containing cheapest edge in cut $(S, V \setminus S)$.

Cut Property Lemma

We will prove the following more general lemma.

Lemma (Cut Property Lemma)

Let $F \subseteq E$ be a forest which is part of some MST of G. For every nonempty subset $\emptyset \neq S \subset V$ with $\delta(S) \cap F = \emptyset$, there is a MST containing F and the cheapest edge in cut $(S, V \setminus S)$.

Cut Property Lemma

We will prove the following more general lemma.

Lemma (Cut Property Lemma)

Let $F \subseteq E$ be a forest which is part of some MST of G. For every nonempty subset $\emptyset \neq S \subset V$ with $\delta(S) \cap F = \emptyset$, there is a MST containing F and the cheapest edge in cut $(S, V \setminus S)$.

- Proof by exchange argument: let T be a MST which contains F, and let e be cheapest edge in $\delta(S)$.

Cut Property Lemma

We will prove the following more general lemma.

Lemma (Cut Property Lemma)

Let $F \subseteq E$ be a forest which is part of some MST of G. For every nonempty subset $\emptyset \neq S \subset V$ with $\delta(S) \cap F = \emptyset$, there is a MST containing F and the cheapest edge in cut $(S, V \setminus S)$.

- Proof by exchange argument: let T be a MST which contains F, and let e be cheapest edge in $\delta(S)$.
- If $e \in T$ we are done, so assume $e \notin T$.

Cut Property Lemma

We will prove the following more general lemma.

Lemma (Cut Property Lemma)

Let $F \subseteq E$ be a forest which is part of some MST of G. For every nonempty subset $\emptyset \neq S \subset V$ with $\delta(S) \cap F = \emptyset$, there is a MST containing F and the cheapest edge in cut $(S, V \setminus S)$.

- Proof by exchange argument: let T be a MST which contains F, and let e be cheapest edge in $\delta(S)$.
- If $e \in T$ we are done, so assume $e \notin T$.
- Note that $T + e$ must contain exactly one cycle, and this cycle contains e. Moreover, this cycle contains another edge from $\delta(S)$, as T connects the graph. Let $f \neq e$ be this other edge.

Cut Property Lemma

We will prove the following more general lemma.

Lemma (Cut Property Lemma)

Let $F \subseteq E$ be a forest which is part of some MST of G. For every nonempty subset $\emptyset \neq S \subset V$ with $\delta(S) \cap F = \emptyset$, there is a MST containing F and the cheapest edge in cut $(S, V \setminus S)$.

- Proof by exchange argument: let T be a MST which contains F, and let e be cheapest edge in $\delta(S)$.
- If $e \in T$ we are done, so assume $e \notin T$.
- Note that $T + e$ must contain exactly one cycle, and this cycle contains e. Moreover, this cycle contains another edge from $\delta(S)$, as T connects the graph. Let $f \neq e$ be this other edge.
- By minimality of e, we have

$$w(T + e - f) = w(T) + w(e) - w(f) \leq w(t)$$

Cut Property Lemma

We will prove the following more general lemma.

Lemma (Cut Property Lemma)

Let $F \subseteq E$ be a forest which is part of some MST of G. For every nonempty subset $\emptyset \neq S \subset V$ with $\delta(S) \cap F = \emptyset$, there is a MST containing F and the cheapest edge in cut $(S, V \setminus S)$.

- Proof by exchange argument: let T be a MST which contains F, and let e be cheapest edge in $\delta(S)$.
- If $e \in T$ we are done, so assume $e \notin T$.
- Note that $T + e$ must contain exactly one cycle, and this cycle contains e. Moreover, this cycle contains another edge from $\delta(S)$, as T connects the graph. Let $f \neq e$ be this other edge.
- By minimality of e, we have

$$w(T + e - f) = w(T) + w(e) - w(f) \leq w(t)$$

- $F \subset T + e - f$, since $F \subset T$ and $F \cap \delta(S) = \emptyset$

Prim's algorithm

- **Idea:** start from arbitrary vertex s and grow connected component one vertex at a time

Prim's algorithm

- **Idea:** start from arbitrary vertex s and grow connected component one vertex at a time
- Algorithm
  1. $F = \emptyset$, $S = \{s\}$
  2. While $S \neq V$:
     - let $e = \{u, v\} \in \delta(S)$ be a cheapest edge, with $u \in S$, $v \notin S$
     - $F \leftarrow F + e$, $S \leftarrow S \cup \{v\}$
  3. return F

Prim's algorithm

- **Idea:** start from arbitrary vertex s and grow connected component one vertex at a time
- Algorithm
  1. $F = \emptyset$, $S = \{s\}$
  2. While $S \neq V$:
     - let $e = \{u, v\} \in \delta(S)$ be a cheapest edge, with $u \in S$, $v \notin S$
     - $F \leftarrow F + e$, $S \leftarrow S \cup \{v\}$
  3. return F
- **Correctness:** follows from cut property lemma

Prim's algorithm

- **Idea:** start from arbitrary vertex s and grow connected component one vertex at a time
- Algorithm
  1. $F = \emptyset$, $S = \{s\}$
  2. While $S \neq V$:
     - let $e = \{u, v\} \in \delta(S)$ be a cheapest edge, with $u \in S$, $v \notin S$
     - $F \leftarrow F + e$, $S \leftarrow S \cup \{v\}$
  3. return F
- **Correctness:** follows from cut property lemma
- **Runtime:** need to find cheapest edge fast. How can we do that? Via priority-queue (a balanced BST).
  Using such a priority-queue, runtime is given by $O(m \log n)$.

Prim - Full Implementation

- Full Algorithm
  1. $F = \emptyset$, $S = \{s\}$, $p[u] = NULL$ for all $u \in V$
     - $D[u] = \infty$ for all $u \in V \setminus \{s\}$, $D[s] = 0$ (distance to set S)
     - $Q = V$ priority-queue (balanced BST with keys given by D)
  2. While $Q \neq \emptyset$:
     - $u = \text{EXTRACT-MIN}(Q)$
     - For $v \in N(u)$:
       if $w_{uv} < D[v]$, then:
       set $D[v] = w_{uv}$,
       $p[v] = u$ and do
       $\text{DECREASE-KEY}(Q, v)$
     - $F \leftarrow F + \{u, p[u]\}$, $S \leftarrow S \cup \{u\}$
  3. return F

Kruskal's Algorithm

- **Idea:** consider edges from cheapest to most expensive, and add edge to the solution as long as it doesn't create a cycle

Kruskal's Algorithm

- **Idea:** consider edges from cheapest to most expensive, and add edge to the solution as long as it doesn't create a cycle
- Algorithm
  1. $F = \emptyset$
  2. Sort edges in non-decreasing weights, so $w(e_1) \leq w(e_2) \leq \cdots \leq w(e_m)$
  3. For $1 \leq i \leq m$:
     If $F \cup \{e_i\}$ doesn't create a cycle, then $F \leftarrow F \cup \{e_i\}$
  4. return F

Kruskal's Algorithm

- **Idea:** consider edges from cheapest to most expensive, and add edge to the solution as long as it doesn't create a cycle
- Algorithm
  1. $F = \emptyset$
  2. Sort edges in non-decreasing weights, so $w(e_1) \leq w(e_2) \leq \cdots \leq w(e_m)$
  3. For $1 \leq i \leq m$:
     If $F \cup \{e_i\}$ doesn't create a cycle, then $F \leftarrow F \cup \{e_i\}$
  4. return F
- **Correctness:** follows from cut property lemma

Kruskal's Algorithm

- **Idea:** consider edges from cheapest to most expensive, and add edge to the solution as long as it doesn't create a cycle
- Algorithm
  1. $F = \emptyset$
  2. Sort edges in non-decreasing weights, so $w(e_1) \leq w(e_2) \leq \cdots \leq w(e_m)$
  3. For $1 \leq i \leq m$:

     If $F \cup \{e_i\}$ doesn't create a cycle, then $F \leftarrow F \cup \{e_i\}$
  4. return F
- **Correctness:** follows from cut property lemma
- **Running Time:** need to check if the two endpoints of edges e_i belong to same component in forest F.

UNION-FIND

Kruskal's Algorithm - Full Implementation

- UNION-FIND data-structure
  1. MAKESET(x): creates singleton set containing just x
  2. FIND(x): returns which set x belongs to
  3. UNION(x, y): merge sets containing x and y

Kruskal's Algorithm - Full Implementation

- UNION-FIND data-structure
  1. MAKESET(x): creates singleton set containing just x
  2. FIND(x): returns which set x belongs to
  3. UNION(x, y): merge sets containing x and y
- Can implement all these operations in $O(\log n)$ time when there are at most n elements[1]

[1] And in CS 466 we see how to do it even faster! :)

Kruskal's Algorithm - Full Implementation

- UNION-FIND data-structure
  1. MAKESET(x): creates singleton set containing just x
  2. FIND(x): returns which set x belongs to
  3. UNION(x, y): merge sets containing x and y
- Can implement all these operations in $O(\log n)$ time when there are at most n elements[1]
- Algorithm:
  - $F := \emptyset$, MAKESET(u) for each $u \in V$
  - Sort edges in non-decreasing weights, so $w(e_1) \le w(e_2) \le \cdots \le w(e_m)$
  - For $1 \le i \le m$: let $e_i = \{u, v\}$

    If FIND(u) $\neq$ FIND(v) (i.e. $F \cup \{e_i\}$ doesn't create a cycle):

    $F \leftarrow F \cup \{e_i\}$ and UNION(u, v)
  - return F

[1] And in CS 466 we see how to do it even faster! :)

Kruskal's Algorithm - Full Implementation

- UNION-FIND data-structure
  1. MAKESET(x): creates singleton set containing just x
  2. FIND(x): returns which set x belongs to
  3. UNION(x, y): merge sets containing x and y
- Can implement all these operations in $O(\log n)$ time when there are at most n elements[1]
- Algorithm:
  - $F := \emptyset$, MAKESET(u) for each $u \in V$
  - Sort edges in non-decreasing weights, so $w(e_1) \leq w(e_2) \leq \cdots \leq w(e_m)$
  - For $1 \leq i \leq m$: let $e_i = \{u, v\}$

    If FIND(u) $\neq$ FIND(v) (i.e. $F \cup \{e_i\}$ doesn't create a cycle):

    $F \leftarrow F \cup \{e_i\}$ and UNION(u, v)
  - return F
- Each data structure operation can be done in $O(\log n)$ time, then total running time is $O(m \log n)$.

[1] And in CS 466 we see how to do it even faster! :)

Reverse-Delete Algorithm

- **Idea:** keep removing heaviest edge as long as remaining graph still connected.

Reverse-Delete Algorithm

- **Idea:** keep removing heaviest edge as long as remaining graph still connected.
- Correctness of this algorithm follows from the following lemma

Lemma (Cycle Property)

If C is any cycle in G and $e \in C$ is a most expensive edge belonging to C, then there is T MST of G such that $e \notin T$.
If all edges have distinct weights, then e does not belong to any MST of G.

Acknowledgement

- Based on Prof. Lau's Lecture 10

  `https://cs.uwaterloo.ca/~lapchi/cs341/notes/L10.pdf`
- Also based on [**?**, Chapters 2 and 4]KT

References I

Cormen, Thomas and Leiserson, Charles and Rivest, Ronald and Stein, Clifford. (2009)
Introduction to Algorithms, third edition.
MIT Press

Kleinberg, Jon and Tardos, Eva (2006)
Algorithm Design.
Addison Wesley