Design and Analysis of Algorithms III

Minimum Spanning Trees

Guoqiang Li
School of Software

Shanghai Jiao Tong University

Minimum Spanning Trees

Cycles

A path is a sequence of edges which connects a sequence of nodes.

Cycles

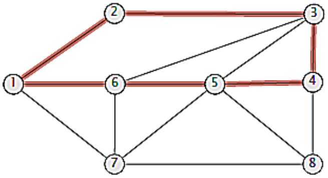
A path is a sequence of edges which connects a sequence of nodes.

A cycle is a path with no repeated nodes or edges other than the starting and ending nodes.

Cycles

A path is a sequence of edges which connects a sequence of nodes.

A cycle is a path with no repeated nodes or edges other than the starting and ending nodes.



path $P = \{(1,2),(2,3),(3,4),(4,5),(5,6)\}$
cycle $C = \{(1,2),(2,3),(3,4),(4,5),(5,6),(6,1)\}$

Cuts

A cut is a partition of the nodes into two nonempty subsets S and $V - S$.

Cuts

A cut is a partition of the nodes into two nonempty subsets S and $V - S$.

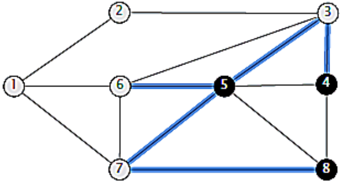
The cutset of a cut S is the set of edges with exactly one endpoint in S.

Cuts

SHANGHAI JIAO TONG UNIVERSITY

A cut is a partition of the nodes into two nonempty subsets S and $V - S$.

The cutset of a cut S is the set of edges with exactly one endpoint in S.



cut $S = \{4, 5, 8\}$

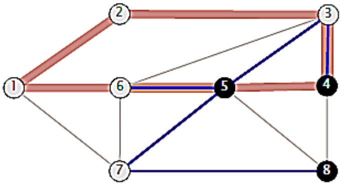
cutset $D = \{(3,4), (3,5), (5,6), (5,7), (8,7)\}$

Cycle-Cut Intersection

SHANGHAI JIAO TONG UNIVERSITY

Proposition

A cycle and a cutset intersect in an even number of edges.

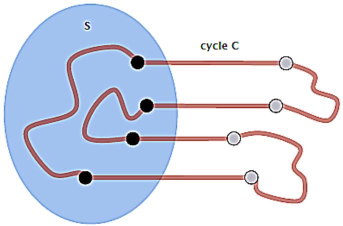


$$\begin{aligned}
\text{cycle } C &= \{(1,2),(2,3),(3,4),(4,5),(5,6),(6,1)\} \\
\text{cutset } D &= \{(3,4),(3,5),(5,6),(5,7),(8,7)\} \\
\text{intersection } C \cap D &= \{(3,4),(5,6)\}
\end{aligned}$$

Cycle-Cut Intersection

Proposition

A cycle and a cutset intersect in an even number of edges.



Spanning Tree Definition

Let $H = (V, T)$ be a subgraph of an undirected graph $G = (V, E)$. H is a spanning tree of G if H is both acyclic and connected.

Spanning Tree Properties

Proposition

Let $H = (V, T)$ be a subgraph of an undirected graph $G = (V, E)$. Then, the following are equivalent:

- H *is a spanning tree of* G.
- H *is acyclic and connected.*
- H *is connected and has* $|V| - 1$ *edges.*
- H *is acyclic and has* $|V| - 1$ *edges.*
- H *is minimally connected: removal of any edge disconnects it.*
- H *is maximally acyclic: addition of any edge creates a cycle.*

Minimum Spanning Tree (MST)

Given a connected, undirected graph $G = (V, E)$ with edge costs c_e, a minimum spanning tree (V, T) is a spanning tree of G such that the sum of the edge costs in T is minimized.

Minimum Spanning Tree (MST)

Given a connected, undirected graph $G = (V, E)$ with edge costs c_e, a minimum spanning tree (V, T) is a spanning tree of G such that the sum of the edge costs in T is minimized.

Cayley's theorem. The complete graph on n nodes has n^{n-2} spanning trees.

Applications

MST is fundamental problem with diverse applications.

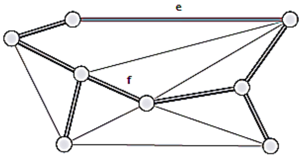
- Dithering.
- Cluster analysis.
- Max bottleneck paths.
- Real-time face verification.
- LDPC codes for error correction.
- Image registration with Renyi entropy.
- Find road networks in satellite and aerial imagery.
- Model locality of particle interactions in turbulent fluid flows.
- Reducing data storage in sequencing amino acids in a protein.
- Autoconfig protocol for Ethernet bridging to avoid cycles in a network.
- Approximation algorithms for NP-hard problems.
- Network design (communication, electrical, hydraulic, computer, road).

Fundamental Cycle

Shanghai Jiao Tong University

Fundamental cycle. Let $H = (V, T)$ be a spanning tree of $G = (V, E)$.

- For any non tree-edge $e \in E : T \cup \{e\}$ contains a unique cycle, say C.
- For any edge $f \in C : T \cup \{e\} - \{f\}$ is a spanning tree.

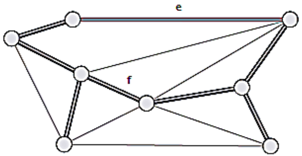


graph $G = (V, E)$
spanning tree $H = (V, T)$

Fundamental Cycle

Fundamental cycle. Let $H = (V, T)$ be a spanning tree of $G = (V, E)$.

- For any non tree-edge $e \in E : T \cup \{e\}$ contains a unique cycle, say C.
- For any edge $f \in C : T \cup \{e\} - \{f\}$ is a spanning tree.



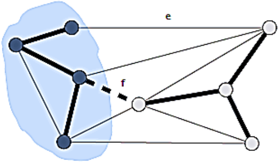
graph $G = (V, E)$
spanning tree $H = (V, T)$

Observation. If $c_e < c_f$, then (V, T) is not an MST.

Fundamental Cutset

Fundamental cutset. Let $H = (V, T)$ be a spanning tree of $G = (V, E)$.

- For any tree-edge $f \in T : T - \{f\}$ contains two connected components. Let D denote corresponding cutset.
- For any edge $e \in D : T - \{f\} \cup \{e\}$ is a spanning tree.



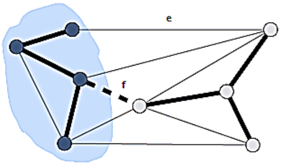
graph $G = (V, E)$
spanning tree $H = (V, T)$

Fundamental Cutset

SHANGHAI JIAO TONG UNIVERSITY

Fundamental cutset. Let $H = (V, T)$ be a spanning tree of $G = (V, E)$.

- For any tree-edge $f \in T : T - \{f\}$ contains two connected components. Let D denote corresponding cutset.
- For any edge $e \in D : T - \{f\} \cup \{e\}$ is a spanning tree.



$$\begin{aligned} \textbf{graph } G &= (V, E) \\ \textbf{spanning tree } H &= (V, T) \end{aligned}$$

Observation. If $c_e < c_f$, then (V, T) is not an MST.

The Greedy Algorithm

Red rule.

- Let C be a cycle with no red edges.
- Select an uncolored edge of C of max cost and color it red.

The Greedy Algorithm

Red rule.

- Let C be a cycle with no red edges.
- Select an uncolored edge of C of max cost and color it red.

Blue rule.

- Let D be a cutset with no blue edges.
- Select an uncolored edge in D of min cost and color it blue.

The Greedy Algorithm

Red rule.

- Let C be a cycle with no red edges.
- Select an uncolored edge of C of max cost and color it red.

Blue rule.

- Let D be a cutset with no blue edges.
- Select an uncolored edge in D of min cost and color it blue.

Greedy algorithm.

- Apply the red and blue rules (nondeterministically!) until all edges are colored. The blue edges form an MST.
- Note: can stop once $|V| - 1$ edges colored blue.

Proof of Correctness

Color invariant. There exists an $MST(V, T^*)$ containing every blue edge and no red edge.

Proof of Correctness

Color invariant. There exists an $MST(V, T^*)$ containing every blue edge and no red edge.

Proof. [by induction on number of iterations]

Proof of Correctness

Color invariant. There exists an $MST(V, T^*)$ containing every blue edge and no red edge.

Proof. [by induction on number of iterations]

Base case. No edges colored $\implies$ every MST satisfies invariant.

Proof of Correctness

Color invariant. There exists an $MST(V, T^*)$ containing every blue edge and no red edge.

Proof. [by induction on number of iterations]

Induction step (blue rule). Suppose color invariant true before blue rule.

Proof of Correctness

Color invariant. There exists an $MST(V, T^*)$ containing every blue edge and no red edge.

Proof. [by induction on number of iterations]

Induction step (blue rule). Suppose color invariant true before blue rule.

- let D be chosen cutset, and let f be edge colored blue.

Proof of Correctness

Color invariant. There exists an $MST(V, T^*)$ containing every blue edge and no red edge.

Proof. [by induction on number of iterations]

Induction step (blue rule). Suppose color invariant true before blue rule.

- let D be chosen cutset, and let f be edge colored blue.
- if $f \in T^*$, then T^* still satisfies invariant.

Proof of Correctness

Color invariant. There exists an $MST(V, T^*)$ containing every blue edge and no red edge.

Proof. [by induction on number of iterations]

Induction step (blue rule). Suppose color invariant true before blue rule.

- let D be chosen cutset, and let f be edge colored blue.
- if $f \in T^*$, then T^* still satisfies invariant.
- Otherwise, consider fundamental cycle C by adding f to T^*.

Proof of Correctness

Color invariant. There exists an $MST(V, T^*)$ containing every blue edge and no red edge.

Proof. [by induction on number of iterations]

Induction step (blue rule). Suppose color invariant true before blue rule.

- let D be chosen cutset, and let f be edge colored blue.
- if $f \in T^*$, then T^* still satisfies invariant.
- Otherwise, consider fundamental cycle C by adding f to T^*.
- let $e \in C$ be another edge in D.

Proof of Correctness

Color invariant. There exists an $MST(V, T^*)$ containing every blue edge and no red edge.

Proof. [by induction on number of iterations]

Induction step (blue rule). Suppose color invariant true before blue rule.

- let D be chosen cutset, and let f be edge colored blue.
- if $f \in T^*$, then T^* still satisfies invariant.
- Otherwise, consider fundamental cycle C by adding f to T^*.
- let $e \in C$ be another edge in D.
- e is uncolored and $c_e \geq c_f$ since

Proof of Correctness

Color invariant. There exists an $MST(V, T^*)$ containing every blue edge and no red edge.

Proof. [by induction on number of iterations]

Induction step (blue rule). Suppose color invariant true before blue rule.

- let D be chosen cutset, and let f be edge colored blue.
- if $f \in T^*$, then T^* still satisfies invariant.
- Otherwise, consider fundamental cycle C by adding f to T^*.
- let $e \in C$ be another edge in D.
- e is uncolored and $c_e \geq c_f$ since
  - $e \in T^* \Rightarrow$ not red

Proof of Correctness

Color invariant. There exists an $MST(V, T^*)$ containing every blue edge and no red edge.

Proof. [by induction on number of iterations]

Induction step (blue rule). Suppose color invariant true before blue rule.

- let D be chosen cutset, and let f be edge colored blue.
- if $f \in T^*$, then T^* still satisfies invariant.
- Otherwise, consider fundamental cycle C by adding f to T^*.
- let $e \in C$ be another edge in D.
- e is uncolored and $c_e \geq c_f$ since
  - $e \in T^* \Rightarrow$ not red
  - blue rule $\Rightarrow e$ not blue and $c_e \geq c_f$

Proof of Correctness

Color invariant. There exists an $MST(V, T^*)$ containing every blue edge and no red edge.

Proof. [by induction on number of iterations]

Induction step (blue rule). Suppose color invariant true before blue rule.

- let D be chosen cutset, and let f be edge colored blue.
- if $f \in T^*$, then T^* still satisfies invariant.
- Otherwise, consider fundamental cycle C by adding f to T^*.
- let $e \in C$ be another edge in D.
- e is uncolored and $c_e \geq c_f$ since
  - $e \in T^* \Rightarrow$ not red
  - blue rule $\Rightarrow e$ not blue and $c_e \geq c_f$
- Thus, $T^* \cup \{f\} - \{e\}$ satisfies invariant.

Proof of Correctness

Color invariant. There exists an $MST(V, T^*)$ containing every blue edge and no red edge.

Proof. [by induction on number of iterations]

Induction step (red rule). Suppose color invariant true before red rule.

Proof of Correctness

Shanghai Jiao Tong University

Color invariant. There exists an $MST(V, T^*)$ containing every blue edge and no red edge.

Proof. [by induction on number of iterations]

Induction step (red rule). Suppose color invariant true before red rule.

- let C be chosen cycle, and let e be edge colored red.

Proof of Correctness

Color invariant. There exists an $MST(V, T^*)$ containing every blue edge and no red edge.

Proof. [by induction on number of iterations]

Induction step (red rule). Suppose color invariant true before red rule.

- let C be chosen cycle, and let e be edge colored red.
- if $e \notin T^*$, then T^* still satisfies invariant.

Proof of Correctness

Color invariant. There exists an $MST(V, T^*)$ containing every blue edge and no red edge.

Proof. [by induction on number of iterations]

Induction step (red rule). Suppose color invariant true before red rule.

- let C be chosen cycle, and let e be edge colored red.
- if $e \notin T^*$, then T^* still satisfies invariant.
- Otherwise, consider fundamental cutset D by deleting e from T^*.

Proof of Correctness

Color invariant. There exists an $MST(V, T^*)$ containing every blue edge and no red edge.

Proof. [by induction on number of iterations]

Induction step (red rule). Suppose color invariant true before red rule.

- let C be chosen cycle, and let e be edge colored red.
- if $e \notin T^*$, then T^* still satisfies invariant.
- Otherwise, consider fundamental cutset D by deleting e from T^*.
- let $f \in D$ be another edge in C.

Proof of Correctness

SHANGHAI JIAO TONG UNIVERSITY

Color invariant. There exists an $MST(V, T^*)$ containing every blue edge and no red edge.

Proof. [by induction on number of iterations]

Induction step (red rule). Suppose color invariant true before red rule.

- let C be chosen cycle, and let e be edge colored red.
- if $e \notin T^*$, then T^* still satisfies invariant.
- Otherwise, consider fundamental cutset D by deleting e from T^*.
- let $f \in D$ be another edge in C.
- f is uncolored and $c_e \geq c_f$ since

Proof of Correctness

Color invariant. There exists an $MST(V, T^*)$ containing every blue edge and no red edge.

Proof. [by induction on number of iterations]

Induction step (red rule). Suppose color invariant true before red rule.

- let C be chosen cycle, and let e be edge colored red.
- if $e \notin T^*$, then T^* still satisfies invariant.
- Otherwise, consider fundamental cutset D by deleting e from T^*.
- let $f \in D$ be another edge in C.
- f is uncolored and $c_e \geq c_f$ since
  - $f \notin T^* \Rightarrow f$ not blue

Proof of Correctness

Color invariant. There exists an $MST(V, T^*)$ containing every blue edge and no red edge.

Proof. [by induction on number of iterations]

Induction step (red rule). Suppose color invariant true before red rule.

- let C be chosen cycle, and let e be edge colored red.
- if $e \notin T^*$, then T^* still satisfies invariant.
- Otherwise, consider fundamental cutset D by deleting e from T^*.
- let $f \in D$ be another edge in C.
- f is uncolored and $c_e \geq c_f$ since
  - $f \notin T^* \Rightarrow f$ not blue
  - red rule $\Rightarrow f$ not red and $c_e \geq c_f$

Proof of Correctness

Color invariant. There exists an $MST(V, T^*)$ containing every blue edge and no red edge.

Proof. [by induction on number of iterations]

Induction step (red rule). Suppose color invariant true before red rule.

- let C be chosen cycle, and let e be edge colored red.
- if $e \notin T^*$, then T^* still satisfies invariant.
- Otherwise, consider fundamental cutset D by deleting e from T^*.
- let $f \in D$ be another edge in C.
- f is uncolored and $c_e \geq c_f$ since
  - $f \notin T^* \Rightarrow f$ not blue
  - red rule $\Rightarrow f$ not red and $c_e \geq c_f$
- Thus, $T^* \cup \{f\} - \{e\}$ satisfies invariant.

Proof of Correctness

Theorem

The greedy algorithm terminates. Blue edges form an MST.

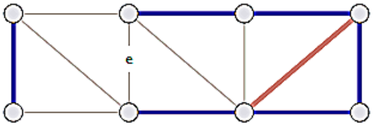
Proof of Correctness

Theorem

The greedy algorithm terminates. Blue edges form an MST.

Proof. We need to show that either the red or blue rule (or both) applies.

- Suppose edge e is left uncolored.
- Blue edges form a forest.
- Case 1: both endpoints of e are in same blue tree.
  $\Rightarrow$ apply red rule to cycle formed by adding e to blue forest.



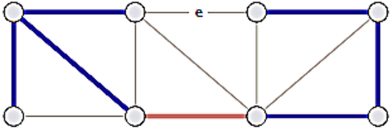
Proof of Correctness

Theorem

The greedy algorithm terminates. Blue edges form an MST.

Proof. We need to show that either the red or blue rule (or both) applies.

- Suppose edge e is left uncolored.
- Blue edges form a forest.
- Case 1: both endpoints of e are in same blue tree.
  $\Rightarrow$ apply red rule to cycle formed by adding e to blue forest.
- Case 2: both endpoints of e are in different blue trees.
  $\Rightarrow$ apply blue rule to cutset induced by either of two blue trees.



Prim, Kruskal, Borůvka

Prim's Algorithm

Prim's Algorithm

Shanghai Jiao Tong University

Initialize $S =$ any node, $T = \varnothing$.

Prim's Algorithm

Initialize $S =$ any node, $T = \varnothing$.
Repeat $|V| - 1$ times:

Prim's Algorithm

Initialize $S =$ any node, $T = \varnothing$.

Repeat $|V| - 1$ times:

- Add to T a min-cost edge with one endpoint in S.

Prim's Algorithm

Initialize $S =$ any node, $T = \varnothing$.
Repeat $|V| - 1$ times:

- Add to T a min-cost edge with one endpoint in S.
- Add new node to S.

Prim's Algorithm

Initialize $S =$ any node, $T = \varnothing$.
Repeat $|V| - 1$ times:

- Add to T a min-cost edge with one endpoint in S.
- Add new node to S.

Theorem

Prim's algorithm computes an MST.

Prim's Algorithm

Initialize $S =$ any node, $T = \varnothing$.
Repeat $|V| - 1$ times:

- Add to T a min-cost edge with one endpoint in S.
- Add new node to S.

Theorem

Prim's algorithm computes an MST.

Proof. Special case of greedy algorithm

Prim's Algorithm

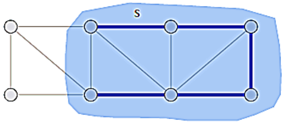
Initialize $S =$ any node, $T = \varnothing$.
Repeat $|V| - 1$ times:

- Add to T a min-cost edge with one endpoint in S.
- Add new node to S.

Theorem

Prim's algorithm computes an MST.

Proof. Special case of greedy algorithm (blue rule repeatedly applied to S).



Prim's Algorithm: Implementation


```
PRIM(G, w)
input  : A connected undirected graph G = (V, E), with edge weights w_e
output: A minimum spanning tree defined by the array prev

for all u ∈ V do
    cost(u) = ∞;
    prev(u) = nil;
end
pick any initial node u_0;
cost(u_0) = 0;
H =makequeue(V)\\ using cost-values as keys;
while H is not empty do
    v=deletemin(H);
    for each (v, z) ∈ E do
        if cost(z) > w(v, z) then
            cost(v) = w(v, z); prev(z) = v;
            decreasekey (H,z);
        end
    end
end
```

Prim's Algorithm: Analysis

SHANGHAI JIAO TONG UNIVERSITY

Theorem

Prim's algorithm can be implemented to run in $O(|E| \log |V|)$ *time.*

Prim's Algorithm: Analysis

Theorem

Prim's algorithm can be implemented to run in $O(|E| \log |V|)$ time.

Proof.

Prim's Algorithm: Analysis

Theorem

Prim's algorithm can be implemented to run in $O(|E| \log |V|)$ *time.*

Proof.

By priority queue implementation.

Kruskal's Algorithm

Kruskal's Algorithm

Consider edges in ascending order of cost:

Kruskal's Algorithm

Consider edges in ascending order of cost:

- Add to tree unless it would create a cycle.

Consider edges in ascending order of cost:

- Add to tree unless it would create a cycle.

Theorem

Kruskal's algorithm computes an MST.

Kruskal's Algorithm

Consider edges in ascending order of cost:

- Add to tree unless it would create a cycle.

Theorem

Kruskal's algorithm computes an MST.

Proof. Special case of greedy algorithm.

Kruskal's Algorithm

SHANGHAI JIAO TONG UNIVERSITY

Consider edges in ascending order of cost:

- Add to tree unless it would create a cycle.

Theorem

Kruskal's algorithm computes an MST.

Proof. Special case of greedy algorithm.

- Case 1: both endpoints of e in same blue tree.

Kruskal's Algorithm

Consider edges in ascending order of cost:

- Add to tree unless it would create a cycle.

Theorem

Kruskal's algorithm computes an MST.

Proof. Special case of greedy algorithm.

- Case 1: both endpoints of e in same blue tree.
  $\Rightarrow$ color e red by applying red rule to unique cycle.

Kruskal's Algorithm

Consider edges in ascending order of cost:

- Add to tree unless it would create a cycle.

Theorem

Kruskal's algorithm computes an MST.

Proof. Special case of greedy algorithm.

- Case 1: both endpoints of e in same blue tree.
  $\Rightarrow$ color e red by applying red rule to unique cycle.
- Case 2: both endpoints of e in different blue trees.

Kruskal's Algorithm

Consider edges in ascending order of cost:

- Add to tree unless it would create a cycle.

Theorem

Kruskal's algorithm computes an MST.

Proof. Special case of greedy algorithm.

- Case 1: both endpoints of e in same blue tree.
  $\Rightarrow$ color e red by applying red rule to unique cycle.
- Case 2: both endpoints of e in different blue trees.
  $\Rightarrow$ color e blue by applying blue rule to cutset defined by either tree.

Kruskal's Algorithm

KRUSKAL(V, E, c)

SORT m edges by cost and renumber so that
$c(e_1) \leq c(e_2) \leq \ldots \leq c(e_m)$;
$T \leftarrow \varnothing$;
for *each* $v \in V$ **do** MAKESET(v);
for $i = 1$ TO m **do**
  $(u, v) \leftarrow e_i$;
  if FINDSET(u) $\neq$ FINDSET(v) **then**
    $T \leftarrow T \cup \{e_i\}$;
    UNION(u,v);
  end
end
RETURN T;

Kruskal's Algorithm: Analysis

Theorem

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

Kruskal's Algorithm: Analysis

Theorem

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

- Sort edges by cost.

Kruskal's Algorithm: Analysis

SHANGHAI JIAO TONG UNIVERSITY

Theorem

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

- Sort edges by cost.
- Use disjoint set data structure to dynamically maintain connected components.

Reverse-Delete Algorithm

Start with all edges in T and consider them in descending order of cost:

Reverse-Delete Algorithm

Shanghai Jiao Tong University

Start with all edges in T and consider them in descending order of cost:

- Delete edge from T unless it would disconnect T.

Reverse-Delete Algorithm

Start with all edges in T and consider them in descending order of cost:

- Delete edge from T unless it would disconnect T.

Theorem

The reverse-delete algorithm computes an MST.

Reverse-Delete Algorithm

Start with all edges in T and consider them in descending order of cost:

- Delete edge from T unless it would disconnect T.

Theorem

The reverse-delete algorithm computes an MST.

Proof. Special case of greedy algorithm.

- Case 1. [deleting edge e does not disconnect T]

Reverse-Delete Algorithm

Start with all edges in T and consider them in descending order of cost:

- Delete edge from T unless it would disconnect T.

Theorem

The reverse-delete algorithm computes an MST.

Proof. Special case of greedy algorithm.

- Case 1. [deleting edge e does not disconnect T]
  $\Rightarrow$ apply red rule to cycle C formed by adding e to another path in T between its two endpoints

Reverse-Delete Algorithm

SHANGHAI JIAO TONG UNIVERSITY

Start with all edges in T and consider them in descending order of cost:

- Delete edge from T unless it would disconnect T.

Theorem

The reverse-delete algorithm computes an MST.

Proof. Special case of greedy algorithm.

- Case 1. [deleting edge e does not disconnect T]
  $\Rightarrow$ apply red rule to cycle C formed by adding e to another path in T between its two endpoints
- Case 2. [deleting edge e disconnects T]

Reverse-Delete Algorithm

Start with all edges in T and consider them in descending order of cost:

- Delete edge from T unless it would disconnect T.

Theorem

The reverse-delete algorithm computes an MST.

Proof. Special case of greedy algorithm.

- Case 1. [deleting edge e does not disconnect T]
  $\Rightarrow$ apply red rule to cycle C formed by adding e to another path in T between its two endpoints
- Case 2. [deleting edge e disconnects T]
  $\Rightarrow$ apply blue rule to cutset D induced by either component

Reverse-Delete Algorithm

SHANGHAI JIAO TONG UNIVERSITY

Start with all edges in T and consider them in descending order of cost:

- Delete edge from T unless it would disconnect T.

Theorem

The reverse-delete algorithm computes an MST.

Proof. Special case of greedy algorithm.

- Case 1. [deleting edge e does not disconnect T]
  $\Rightarrow$ apply red rule to cycle C formed by adding e to another path in T between its two endpoints
- Case 2. [deleting edge e disconnects T]
  $\Rightarrow$ apply blue rule to cutset D induced by either component

Fact. [Thorup 2000] Can be implemented to run in $O(|E| \log |V| (\log \log |V|)^3)$ time.

Review: the Greedy MST Algorithm

Shanghai Jiao Tong University

Red rule.

- Let C be a cycle with no red edges.
- Select an uncolored edge of C of max cost and color it red.

Blue rule.

- Let D be a cutset with no blue edges.
- Select an uncolored edge in D of min cost and color it blue.

Greedy algorithm.

- Apply the red and blue rules (nondeterministically!) until all edges are colored. The blue edges form an MST.
- Note: can stop once $|V| - 1$ edges colored blue.

Review: the Greedy MST Algorithm

Theorem

The greedy algorithm is correct.

Review: the Greedy MST Algorithm

Theorem

The greedy algorithm is correct.

Special cases. Prim, Kruskal, reverse-delete, ...

Borůvka's Algorithm

Shanghai Jiao Tong University

Repeat until only one tree.

- Apply blue rule to cutset corresponding to each blue tree.
- Color all selected edges blue.

Borůvka's Algorithm

Repeat until only one tree.

- Apply blue rule to cutset corresponding to each blue tree.
- Color all selected edges blue.

Theorem

Borůvka's algorithm computes the MST. ⟵ *assume edge costs are distinct*

Borůvka's Algorithm

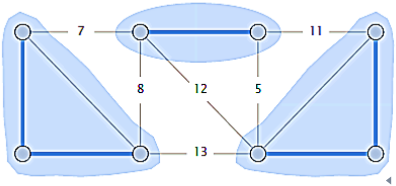
Repeat until only one tree.

- Apply blue rule to cutset corresponding to each blue tree.
- Color all selected edges blue.

Theorem

Borůvka's algorithm computes the MST. ← *assume edge costs are distinct*

Proof. Special case of greedy algorithm (repeatedly apply blue rule).



Borůvka's Algorithm

Theorem

Borůvka's algorithm can be implemented to run in $O(|E| \log |V|)$ *time.*

Borůvka's Algorithm

Theorem

Borůvka's algorithm can be implemented to run in $O(|E| \log |V|)$ *time.*

Proof.

Borůvka's Algorithm

SHANGHAI JIAO TONG UNIVERSITY

Theorem

Borůvka's algorithm can be implemented to run in $O(|E| \log |V|)$ time.

Proof.

To implement a phase in $O(|E|)$ time:

- compute connected components of blue edges
- for each edge $(u, v) \in E$, check if u and v are in different components; if so, update each component's best edge in cutset

Borůvka's Algorithm

Theorem

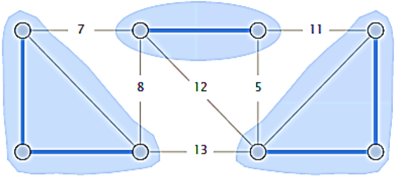
Borůvka's algorithm can be implemented to run in $O(|E| \log |V|)$ time.

Proof.

To implement a phase in $O(|E|)$ time:

- compute connected components of blue edges
- for each edge $(u, v) \in E$, check if u and v are in different components; if so, update each component's best edge in cutset

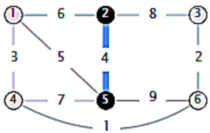
$\leq \log_2 |V|$ phases since each phase (at least) halves total $\#$ components.



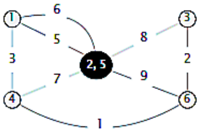
Contraction version.

- After each phase, contract each blue tree to a single supernode.
- Delete self-loops and parallel edges (keeping only cheapest one).
- Borůvka phase becomes: take cheapest edge incident to each node.

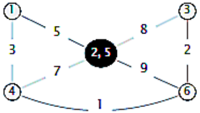
graph G



contract edge 2–5



delete self–loops and parallel edges



A Question

Q. How to contract a set of edges?

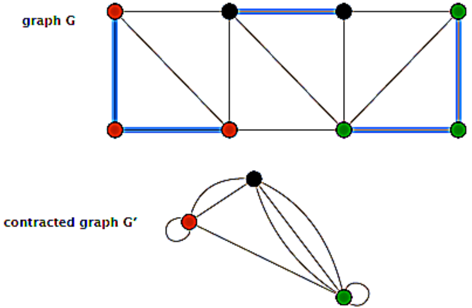
Contract a Set of Edges

Problem. Given a graph $G = (V, E)$ and a set of edges F, contract all edges in F, removing any self-loops or parallel edges.

Contract a Set of Edges

Problem. Given a graph $G = (V, E)$ and a set of edges F, contract all edges in F, removing any self-loops or parallel edges.

Goal. $O(|V| + |E|)$ time.



Contract a Set of Edges

1. mark the edges to be contracted;
2. determine the connected components formed by the marked edges;
3. replace each connected component by a single vertex;
4. finally, eliminate the self-loops and multiple edges created by these contractions.

Contract a Set of Edges

1. mark the edges to be contracted;
    - To find the minimum weight edge incident on each node, takes $O(|E| + |V|)$ time;
2. determine the connected components formed by the marked edges;
3. replace each connected component by a single vertex;
4. finally, eliminate the self-loops and multiple edges created by these contractions.

Contract a Set of Edges

1. mark the edges to be contracted;
   - To find the minimum weight edge incident on each node, takes $O(|E| + |V|)$ time;
2. determine the connected components formed by the marked edges;
   - Use DFS to find the connected components, take $O(|E| + |V|)$ time;
3. replace each connected component by a single vertex;
4. finally, eliminate the self-loops and multiple edges created by these contractions.

Contract a Set of Edges

SHANGHAI JIAO TONG UNIVERSITY

1. mark the edges to be contracted;
   - To find the minimum weight edge incident on each node, takes $O(|E| + |V|)$ time;
2. determine the connected components formed by the marked edges;
   - Use DFS to find the connected components, take $O(|E| + |V|)$ time;
3. replace each connected component by a single vertex;
   - Associate each connected component with that new vertex, take $O(|E| + |V|)$ time (in the above loop);
4. finally, eliminate the self-loops and multiple edges created by these contractions.

Contract a Set of Edges

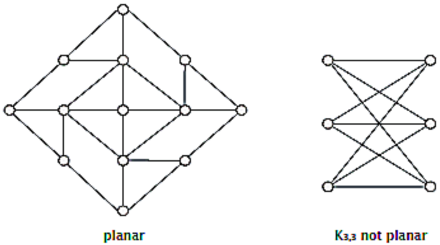
1. mark the edges to be contracted;
   - To find the minimum weight edge incident on each node, takes $O(|E| + |V|)$ time;
2. determine the connected components formed by the marked edges;
   - Use DFS to find the connected components, take $O(|E| + |V|)$ time;
3. replace each connected component by a single vertex;
   - Associate each connected component with that new vertex, take $O(|E| + |V|)$ time (in the above loop);
4. finally, eliminate the self-loops and multiple edges created by these contractions.
   - To eliminate edges, takes $O(|E|)$ time.

Borůvka's on Planar Graphs

SHANGHAI JIAO TONG UNIVERSITY

Theorem

Borůvka's algorithm (contraction version) can be implemented to run in $O(|V|)$ time on planar graphs.



Borůvka's on Planar Graphs

Theorem

Borůvka's algorithm (contraction version) can be implemented to run in $O(|V|)$ time on planar graphs.

Proof.

Borůvka's on Planar Graphs

Theorem

Borůvka's algorithm (contraction version) can be implemented to run in $O(|V|)$ time on planar graphs.

Proof.

Each Borůvka phase takes $O(|V|)$ time:

Borůvka's on Planar Graphs

SHANGHAI JIAO TONG UNIVERSITY

Theorem

Borůvka's algorithm (contraction version) can be implemented to run in $O(|V|)$ time on planar graphs.

Proof.

Each Borůvka phase takes $O(|V|)$ time:

- Fact 1: $|E| \leq 3|V|$ for simple planar graphs.

Borůvka's on Planar Graphs

Theorem

Borůvka's algorithm (contraction version) can be implemented to run in $O(|V|)$ time on planar graphs.

Proof.

Each Borůvka phase takes $O(|V|)$ time:

- Fact 1: $|E| \leq 3|V|$ for simple planar graphs.
- Fact 2: planar graphs remains planar after edge contractions/deletions.

Borůvka's on Planar Graphs

SHANGHAI JIAO TONG UNIVERSITY

Theorem

Borůvka's algorithm (contraction version) can be implemented to run in $O(|V|)$ time on planar graphs.

Proof.

Each Borůvka phase takes $O(|V|)$ time:

- Fact 1: $|E| \leq 3|V|$ for simple planar graphs.
- Fact 2: planar graphs remains planar after edge contractions/deletions.

Number of nodes (at least) halves in each phase.

Borůvka's on Planar Graphs

Theorem

Borůvka's algorithm (contraction version) can be implemented to run in $O(|V|)$ time on planar graphs.

Proof.

Each Borůvka phase takes $O(|V|)$ time:

- Fact 1: $|E| \leq 3|V|$ for simple planar graphs.
- Fact 2: planar graphs remains planar after edge contractions/deletions.

Number of nodes (at least) halves in each phase.

Thus, overall running time $\leq c \cdot |V| + c \cdot |V|/2 + c \cdot |V|/4 + c \cdot |V|/8 + \cdots = O(|V|)$.

A Hybrid Algorithm

Borůvka-Prim algorithm.

- Run Borůvka (contraction version) for $\log_2 \log_2 |V|$ phases.
- Run Prim on resulting, contracted graph.

A Hybrid Algorithm

Borůvka-Prim algorithm.

- Run Borůvka (contraction version) for $\log_2 \log_2 |V|$ phases.
- Run Prim on resulting, contracted graph.

Theorem

Borůvka-Prim computes an MST.

A Hybrid Algorithm

Borůvka-Prim algorithm.

- Run Borůvka (contraction version) for $\log_2 \log_2 |V|$ phases.
- Run Prim on resulting, contracted graph.

Theorem

Borůvka-Prim computes an MST.

Proof. Special case of the greedy algorithm.

A Hybrid Algorithm

Shanghai Jiao Tong University

Theorem

Borůvka-Prim can be implemented to run in $O(|E| \log \log |V|)$ *time.*

A Hybrid Algorithm

SHANGHAI JIAO TONG UNIVERSITY

Theorem

Borůvka-Prim can be implemented to run in $O(|E| \log \log |V|)$ *time.*

Proof.

A Hybrid Algorithm

Theorem

Borůvka-Prim can be implemented to run in $O(|E| \log\log|V|)$ *time.*

Proof.

- The $\log_2 \log_2 |V|$ phases of Borůvka's algorithm take $O(|E| \log\log|V|)$ time; resulting graph has $\leq |V|/\log_2|V|$ nodes and $\leq |E|$ edges.
- Prim's algorithm (using Fibonacci heaps) takes $O(|E| + |V|)$ time on a graph with $|V|/\log_2|V|$ nodes and $|E|$ edges.

Linear-Time Algorithm?

year	worst case	discoverec by
1975	$O(\lvert E\rvert \log\log \lvert V\rvert)$	Yao
1976	$O(\lvert E\rvert \log\log \lvert V\rvert)$	Cheriton-Tarjan
1984	$O(\lvert E\rvert \log^* \lvert V\rvert), O(\lvert E\rvert + \lvert V\rvert \log \lvert V\rvert)$	Fredman-Tarjan
1986	$(\lvert E\rvert \log(\log^* \lvert V\rvert))$	Gabow-Galil-Spencer-Tarjan
1997	$O(\lvert E\rvert \alpha(\lvert V\rvert) \log \alpha(\lvert V\rvert))$	Chazelle
2000	$O(\lvert E\rvert \alpha(\lvert V\rvert))$	Chazelle
2002	asymptotically optimal	Pettie-Ramachandran
20xx	$O(\lvert E\rvert)$	???

deterministic compare-based MST algorithms

iterated logarithm function

$$\lg^* n = \begin{cases} 0 & \text{if } n \le 1 \\ 1 + \lg^*(\lg n) & \text{if } n > 1 \end{cases}$$

n	$\lg^* n$
$(-\infty, 1]$	0
$(1, 2]$	1
$(2, 4]$	2
$(4, 16]$	3
$(16, 2^{16}]$	4
$(2^{16}, 2^{65536}]$	5

Minimum Bottleneck Spanning Tree

Shanghai Jiao Tong University

Problem. Given a connected graph G with positive edge costs, find a spanning tree that minimizes the most expensive edge.

Goal. $O(|E| \log |E|)$ time or better.