Discussion #36

Spanning Trees
Topics

• Spanning trees
• Minimal spanning trees
• Kruskal’s algorithm
• Prim’s algorithm
Spanning Trees

• A spanning tree for a connected, undirected graph $G$ is a graph $S$ consisting of the nodes of $G$ together with enough edges of $G$ such that:
  1. $S$ is connected, and
  2. $S$ is acyclic.

• Example:

  Why “tree”? Pick any node as the root — is a tree.

  Why “spanning”? Every node is (still) connected to every other node.
Algorithm for Generating a Spanning Tree

- A DFS algorithm works.
- Just keep the tree edges.

\[ O(m) = O(n^2) \] (worse case)
Spanning Trees for Weighted Graphs

• A minimal spanning tree for a weighted, connected, undirected graph $G$ is a graph $S$ consisting of the nodes of $G$ together with enough edges of $G$ such that:
  1. $S$ is connected, and
  2. $S$ is acyclic, and
  3. $S$ has minimal weight.

• Examples:

Weighted Graph

(Only) two minimal spanning trees

Note: all weights positive.
Kruskal’s Algorithm

- Sort edges by weight (smallest first)
- For each edge $e = \{x, y\}$ (in order)
  if nodes $x$ and $y$ are not in the same connected component, add $e$

Could have stopped here… we had $n-1$ edges.
Prim’s Algorithm

• Initialize a spanning tree $S$ containing a single vertex, chosen arbitrarily from the graph

• Until $n$-1 edges have been added
  – find the edge $e = \{x, y\}$ such that
    • $x$ is in $S$ and $y$ is not in $S$
    • $e$ is the smallest weighted edge left
  – Add $e$ and $y$ to $S$
Correctness Proofs
& Complexity Analysis

• Correctness
  – Detailed proofs — lengthy
  – Essential idea:
    • Prove connected: add edges until all nodes connected
    • Prove acyclic: don’t add an edge that would create a cycle
    • Prove minimal weight: only add overall or local minimal-weight edges

• Complexity
  – Straightforward implementations are easy to analyze.
  – Both algorithms have clever implementations that make them run faster.
Correctness of Prim’s Algorithm

• Prim’s algorithm produces a minimal spanning tree $S$ from a connected, weighted, undirected graph $G$.

• Proof (sketch)
  – Connected: Assume not, then there are at least two disconnected components, $C_1$ and $C_2$ in $S$. But since $G$ is connected, there is a smallest weight edge $\{x, y\}$ with $x$ in $C_1$ and $y$ in $C_2$ that would have been found. Thus, after running Prim’s algorithm $C_1$ and $C_2$ must be connected.
  – Acyclic: Assume not, then there was a first edge $\{x, y\}$ added to make a cycle. But to have been added, $x$ must have been in $S$ and $y$ must not have been in $S$. Since $y$ must not have been in $S$, there was no path from $x$ to $y$ at the time $\{x, y\}$ was added, and thus no cycle was created when $\{x, y\}$ was added.
  – Minimal weight: By induction with loop invariant: After $k$ iterations the growing spanning tree $S$ has minimal weight.
    • Basis: 0 iterations: The initialization step selects a single node and the weight of $S$ is 0, the minimum possible since all weights are assumed to be positive.
    • Induction: By the induction hypothesis, after $k$ iterations $S$ has minimal weight. Then, after $k+1$ iterations $S$ has minimal weight. For suppose not, then the chosen edge $e$ must not have been the smallest-weight edge left.
Complexity of Prim’s Algorithm

- Initialize a spanning tree $S$ containing a single vertex, chosen arbitrarily from the graph
- Until $n-1$ edges have been added
  - find the edge $e = \{x, y\}$ such that
    - $x$ is in $S$ and $y$ is not in $S$
    - $e$ is the smallest weighted edge left
  - Add $e$ and $y$ to $S$

- $O(1) + O(n(m+1)) = O(nm) = O(n^3)$ in the worst case
A Faster Prim’s Algorithm

• To make Prim’s Algorithm faster, we need a way to find the edge \( e \) faster.
• Can we avoid looking through all edges in each iteration?
  – We can if we sort them first and then make a sorted list of incident edges for each node.
  – In the initialization step this takes \( O(m\log m) \) to sort and \( O(m) \) to make lists for each node — \( O(m\log m) = O(n^2\log n^2) \) in the worst case.
  – Now for each of the \( n-1 \) iterations, we find the edge \( \{x, y\} \) by looking only at the first edge of at most \( n \) lists — \( O(n) \) over all iterations. We must, of course, discard edges on these lists as we build \( S \), so that the edge we want will be first, but with appropriate links, this is \( O(1) \).
  – Thus, the sort in the initialization dominates, which makes the algorithm \( O(m\log m) = O(n^2\log n^2) \) in the worst case.
• To make the algorithm even faster, we must somehow avoid the sort. Can we?
An Even Faster Prim’s Algorithm

• Start with the smallest-weight edge \( e = \{x, y\} \) in \( S \)
  – discard \( x \) and \( y \) from the list of nodes to consider
  – for both \( x \) and \( y \), find the closest node, if any (among the non-discard nodes), and keep them and their edge weights

• Until \( n-1 \) edges have been added
  – find the edge \( e = \{x, y\} \) such that
    • \( x \) is in \( S \) and \( y \) is not in \( S \)
    • \( e \) is the smallest weighted edge left
  – add \( e \) and \( y \) to \( S \)
  – discard \( y \) from the list of nodes to consider, and for both \( x \) and \( y \), find the closest node, if any (among the non-discarded nodes), and keep them and their edge weights

• \( O(m) + O(n(n+1+n)) = O(n^2) \) in the worst case
Correctness of Kruskal’s Algorithm

• Kruskal’s algorithm produces a minimal spanning tree $S$ from a connected, weighted, undirected graph $G$.

• Proof (sketch)
  – Connected: Assume not, then there are at least two disconnected components, $C_1$ and $C_2$ in $S$. But since $G$ is connected, there is a smallest weight edge $\{x, y\}$ with $x$ in $C_1$ and $y$ in $C_2$ that would have been added. Thus, after running Kruskal’s algorithm $C_1$ and $C_2$ must be connected.
  – Acyclic: Assume not, then there was a first edge $\{x, y\}$ added to make a cycle. But to have been added, $x$ and $y$ must have not been connected in $S$. Since $x$ and $y$ must not have been connected in $S$, there was no path from $x$ to $y$ at the time $\{x, y\}$ was added, and thus no cycle was created when $\{x, y\}$ was added.
  – Minimal weight: …
Minimal Weight Proof

- Assume the weights in $G$ are unique, so that there is a unique minimal spanning tree $T$. (We can make the weights unique without changing $S$ or $T$ by adding different infinitesimal amounts to edges with equal weights.)

- We can show that $S$ and $T$ are equivalent by assuming otherwise. If $S$ and $T$ differ, there must be at least one edge that is in one but not the other. Let $e = \{x, y\}$ be the first such edge considered by Kruskal’s algorithm.
  - Case 1: $e$ is in $T$ but not $S$. Since Kruskal’s algorithm rejects $e$, $x$ and $y$ must be connected, but then since these edges must also be in $T$, $T$ has a cycle — a contradiction.
  - Case 2: $e$ is in $S$ but not $T$. Since $T$ is connected and acyclic, there is an acyclic path $P = \langle y, \ldots, v, w, \ldots, x \rangle$ in $T$ connecting $y$ and $x$ ($y = v$ and/or $w = x$ is possible). If all the edges in $P$ have lower weight than $e$, $S$ has a cycle $\langle y, \ldots, v, w, \ldots, x, y \rangle$ — a contradiction. If not, there is an edge, $f = \{v, w\}$, in $P$ that has a higher weight than $e$. But then removing $f$ and adding $e$ in $T$ results in a spanning tree with lower aggregate weight — a contradiction.
Complexity of Kruskal’s Algorithm

- Sort edges by weight (smallest first)
- For each edge $e = \{x, y\}$, until $n-1$ edges added
  if nodes $x$ and $y$ are not in the same connected component, add $e$

- $O(m \log m) + O(nm) = O(n^2 \log n^2) + O(n^3) = O(n^3)$

- How can Kruskal’s algorithm be improved?
  - Often already sorted (omit first step)
  - Find a faster way to check “in same connected component”

* $O(n) —$ not $O(m) —$ because the edges for the DFS check are the edges added to the spanning tree so far, and there can never be more than $n-1$ edges in the spanning tree.
Complexity of Kruskal’s Algorithm

• Assume edges sorted by weight in descending order.
• Initialize pointers to trees of height 0
• For each edge $e = \{x, y\}$, until $n-1$ edges added
  
  if $x$ and $y$ are not in the same tree (i.e. don’t have the same root),
  
  add $e$
  
  merge smaller tree (of $x$ or $y$) into

$$O(n) + O(m \log n) = O(n^2 \log n)$$
Kruskal’s Algorithm: Tree Construction

\begin{itemize}
  \item \{A,B\} 1
  \item \{D,F\} 2
  \item \{B,C\} 4
  \item \{A,D\} 5
  \item \{C,D\} 5
  \item \{E,D\} 6
  \item \{A,E\} 7
  \item \{B,D\} 8
\end{itemize}