Lecture 13:
• Minimum Spanning Trees

Mar 9, 2020
Midterm II

• In Class Wednesday March 25\textsuperscript{th}
  • Working on a backup plan

• Exactly the same format/rules as Midterm I

• Topics: Graph Algorithms
  • Key definitions, properties
  • Representing graphs
  • DFS and topological sort
  • Shortest Paths: BFS, Dijkstra, Bellman-Ford
  • Minimum spanning trees
  • Network flow

} this week
Minimum Spanning Trees
Network Design

• Build a cheap, well connected network
• We are given
  • a set of nodes $V = \{v_1, \ldots, v_n\}$
  • a set of potential edges $E \subseteq V \times V$
• Want to build a network to connect these locations
  • Every $v_i, v_j$ must be well connected
  • Must be as cheap as possible

• Many variants of network design
  • Recall the bus routes problem from HW2
Minimum Spanning Trees (MST)

• **Input**: a weighted graph $G = (V, E, \{w_e\})$
  - Undirected, connected, weights may be negative
  - All edge weights are distinct (makes life simpler)

• **Output**: a minimum weight spanning tree $T$
  - A *spanning tree* of $G$ is a subset of $T \subseteq E$ of the edges such that $(V, T)$ forms a tree
  - *Weight* of a tree $T$ is the sum of the edge weights
  - We’ll use $T^*$ to denote “the” minimum spanning tree
Minimum Spanning Trees (MST)
Minimum Spanning Trees (MST)
MST Algorithms

• There are at least four reasonable MST algorithms
  • **Borůvka’s Algorithm:** start with $T = \emptyset$, in each round add cheapest edge out of each connected component
  • **Prim’s Algorithm:** start with some $s$, at each step add cheapest edge that grows the connected component
  • **Kruskal’s Algorithm:** start with $T = \emptyset$, consider edges in ascending order, adding edges unless they create a cycle
  • **Reverse-Kruskal:** start with $T = E$, consider edges in descending order, deleting edges unless it disconnects
Cycles and Cuts

• **Cycle:** a set of edges \((v_1, v_2), (v_2, v_3), ..., (v_k, v_1)\)

![Diagram of a cycle with nodes and edges]

• **Cut:** a partition of the nodes into \(S, \bar{S}\)

![Diagram of a cut with highlighted nodes]

- Cycle \(C = (1,2),(2,3),(3,4),(4,5),(5,6),(6,1)\)
- Cut \(S = \{4, 5, 8\}\)
- Cutset \(= (5,6), (5,7), (3,4), (3,5), (7,8)\)
Cycles and Cuts

• **Fact:** a cycle and a cutset intersect in an even number of edges
Cycles and Cuts

• **Fact:** removing an edge from a cycle doesn’t disconnect any nodes
Properties of MSTs

• Cut Property: Let $S$ be a cut. Let $e$ be the minimum weight edge cut by $S$. Then the MST $T^*$ contains $e$
  • We call such an $e$ a safe edge

• Cycle Property: Let $C$ be a cycle. Let $f$ be the maximum weight edge in $C$. Then the MST $T^*$ does not contain $f$.
  • We call such an $f$ a useless edge
Proof of Cut Property

- **Cut Property**: Let $S$ be a cut. Let $e$ be the minimum weight edge cut by $S$. Then the MST $T^*$ contains $e$. 

```
\begin{center}
\begin{tikzpicture}
  \node at (0,0) [circle,draw,fill=black!20] (S) {$S$};
  \node at (2,0) [circle,draw,fill=black!20] (f) {$f$};
  \node at (4,0) [circle,draw,fill=black!20] (T) {$T^*$};
  \node at (-0.5,0.5) [circle,draw,fill=black!20] (e) {$e$};

  \draw (S) -- (f);
  \draw (S) -- (e);
  \draw (T) -- (e);

\end{tikzpicture}
\end{center}
```
Proof of Cycle Property

- **Cycle Property:** Let $C$ be a cycle. Let $f$ be the max weight edge in $C$. The MST $T^*$ does not contain $f$. 
Ask the Audience

• Assume $G$ has distinct edge weights

• True/False? If $e$ is the edge with the smallest weight, then $e$ is always in the MST $T^*$

• True/False? If $f$ is the edge with the largest weight, then $f$ is never in the MST $T^*$
The “Only” MST Algorithm

• **GenericMST:**
  - Let $T = \emptyset$
  - Repeat until $T$ is connected:
    - Find one or more safe edges not in $T$
    - Add safe edges to $T$

• **Theorem:** *GenericMST* outputs an MST
Borůvka’s Algorithm

• **Borůvka:**
  • Let $T = \emptyset$
  • Repeat until $T$ is connected:
    • Let $C_1, \ldots, C_k$ be the connected components of $(V, T)$
    • Let $e_1, \ldots, e_k$ be the safe edge for the cuts $C_1, \ldots, C_m$
    • Add $e_1, \ldots, e_k$ to $T$

• **Correctness:** every edge we add is safe
Borůvka’s Algorithm

Label Connected Components

Diagram of a network with labeled edges:

- Nodes: 1, 2, 3, 4, 5, 6, 7, 8
- Edges with labels: 1-2 (5), 2-3 (14), 2-4 (5), 3-4 (3), 4-1 (6), 1-6 (12), 6-7 (9), 6-5 (7), 5-8 (15), 5-3 (8), 3-4 (10)
Borůvka’s Algorithm

Add Safe Edges

Graph with nodes 1, 2, 3, 4, 5, 6, 7, 8 and edges with weights:
- 1 to 2: 6
- 1 to 6: 12
- 2 to 6: 5
- 2 to 3: 14
- 3 to 4: 3
- 3 to 5: 8
- 4 to 5: 10
- 5 to 6: 7
- 6 to 7: 9
- 7 to 8: 15
- 1 to 4: 14
- 2 to 5: 15
Borůvka’s Algorithm

Label Connected Components
Borůvka’s Algorithm

Add Safe Edges
Borůvka’s Algorithm

Done!
Borůvka’s Algorithm (Running Time)

• **Borůvka**
  
  • Let $T = \emptyset$
  
  • Repeat until $T$ is connected:
    • Let $C_1, \ldots, C_k$ be the connected components of $(V, T)$
    • Let $e_1, \ldots, e_k$ be the safe edge for the cuts $C_1, \ldots, C_m$
    • Add $e_1, \ldots, e_k$ to $T$

• **Running time**
  
  • How long to find safe edges?
  
  • How many times through the main loop?
Borůvka’s Algorithm (Running Time)

**FindSafeEdges(G,T):**

find connected components $C_1, ..., C_k$

let $L[v]$ be the component of node $v$

Let $S[i]$ be the safe edge of $C_i$

for each edge $(u,v)$ in $E$:

If $L[u] \neq L[v]$:

If $w(u,v) < w(S[L[u]])$:

$S[L[u]] = (u,v)$

If $w(u,v) < w(S[L[v]])$:

$S[L[v]] = (u,v)$

Return $\{S[1], ..., S[k]\}$
Borůvka’s Algorithm (Running Time)

• **Claim:** every iteration of the main loop halves the number of connected components.
Borůvka’s Algorithm (Running Time)

• **Borůvka**
  • Let $T = \emptyset$
  • Repeat until $T$ is connected:
    • Let $C_1, ..., C_k$ be the connected components of $(V, T)$
    • Let $e_1, ..., e_k$ be the safe edge for the cuts $C_1, ..., C_m$
    • Add $e_1, ..., e_k$ to $T$

• Running Time:
  • How long to find safe edges?
  • How many times through the main loop?
Prim’s Algorithm

• **Prim Informal**
  - Let $T = \emptyset$
  - Let $s$ be some arbitrary node and $S = \{s\}$
  - Repeat until $S = V$
    - Find the cheapest edge $e = (u, v)$ cut by $S$. Add $e$ to $T$ and add $v$ to $S$

• **Correctness**: every edge we add is safe
Prim’s Algorithm
Prim's Algorithm

$\text{Prim}(G=(V,E))$

let $Q$ be a priority queue storing $V$

$\text{value}[v] \leftarrow \infty$, $\text{last}[v] \leftarrow \bot$

$\text{value}[s] \leftarrow 0$ for some arbitrary $s$

while ($Q \neq \emptyset$):

$u \leftarrow \text{ExtractMin}(Q)$

for each edge $(u,v)$ in $E$:

if $v \in Q$ and $w(u,v) < \text{value}[v]$:

$\text{DecreaseKey}(v,w(u,v))$

$\text{last}[v] \leftarrow u$

$T = \{(1,\text{last}[1]), \ldots, (n,\text{last}[n])\}$ (excluding $s$)

return $T$
Kruskal’s Algorithm

• **Kruskal’s Informal**
  • Let $T = \emptyset$
  • For each edge $e$ in ascending order of weight:
    • If adding $e$ would decrease the number of connected components add $e$ to $T$

• **Correctness**: every edge we add is safe
Kruskal’s Algorithm
Implementing Kruskal’s Algorithm

• **Union-Find**: group items into components so that we can efficiently perform two operations:
  • **Find(u)**: lookup which component contains u
  • **Union(u,v)**: merge connected components of u,v

• Can implement **Union-Find** so that
  • Find takes $O(1)$ time
  • Any $k$ Union operations takes $O(k \log k)$ time
Kruskal’s Algorithm (Running Time)

• **Kruskal’s Informal**
  • Let $T = \emptyset$
  • For each edge $e$ in ascending order of weight:
    • If adding $e$ would decrease the number of connected components add $e$ to $T$

• Time to sort:
  • Time to test edges:
  • Time to add edges:
Comparison

- Can compute MST in time $O(m \log m)$

- **Boruvka’s Algorithm:**
  - Only algorithm worth implementing
  - Low overhead, can be easily parallelized
  - Each iteration takes $O(m)$, very few iterations in practice

- **Prim’s/Kruskal’s Algorithms:**
  - Reveal useful structure of MSTs
  - Templates for other algorithms