COMP 355
Advanced Algorithms
Prim’s and Boruvka’s Algorithms for MSTs
Section 23.2 (CLRS): Sections 4.5 (KT)
Minimum Spanning Tree

Minimum spanning tree. Given a connected graph $G = (V, E)$ with real-valued edge weights $c_e$, an MST is a subset of the edges $T \subseteq E$ such that $T$ is a spanning tree whose sum of edge weights is minimized.

Cayley's Theorem. There are $n^{n-2}$ spanning trees of $K_n$.

Can't solve by brute force
MST Problem

Given a connected, undirected graph $G = (V,E)$, a spanning tree is an acyclic subset of edges $T \subseteq E$ that connects all the vertices together.

We define the cost of a spanning tree $T$ to be the sum of edges in the spanning tree $w(T) = \sum_{(u,v) \in T} w(u,v)$.

A minimum spanning tree (MST) is a spanning tree of minimum weight.
Greedy Algorithms

**Kruskal’s algorithm.** Start with $T = \emptyset$. Consider edges in ascending order of cost. Insert edge $e$ in $T$ unless doing so would create a cycle.

**Reverse-Delete algorithm.** Start with $T = E$. Consider edges in descending order of cost. Delete edge $e$ from $T$ unless doing so would disconnect $T$.

**Prim’s algorithm.** Start with some root node $s$ and greedily grow a tree $T$ from $s$ outward. At each step, add the cheapest edge $e$ to $T$ that has exactly one endpoint in $T$.

**Boruvka’s algorithm.**

**Remark.** All three algorithms produce an MST.
Prim's algorithm. [Jarník 1930, Dijkstra 1957, Prim 1959]

- Initialize $S =$ any node.
- Apply cut property to $S$.
- Add min cost edge in cutset corresponding to $S$ to $T$, and add one new explored node $u$ to $S$. 
Implementation: Prim's Algorithm

Implementation. Use a priority queue

- Maintain set of explored nodes S.
- For each unexplored node v, maintain attachment cost $a[v] = \text{cost of cheapest edge } v \text{ to a node in } S$.
- Runtime is $O(m \log n)$

```java
 PrimMST(G=(V,E), v, s) { 
   for each (u in V) { // initialization
     key[u] = +infinity
     color[u] = undiscovered
   }
   key[s] = 0 // start at root
   pred[s] = null
   add all vertices to priority queue Q
   while (Q is nonEmpty) { // until all vertices in MST
     u = extract-min from Q // vertex with lightest edge
     for each (v in Adj[u]) {
       if ((color[v] == undiscovered) && (w(u,v) < key[v])) {
         key[v] = w(u,v) // new lighter edge out of v
         decrease key value of v to key[v]
         pred[v] = u
       }
     }
     color[u] = finished
   } // finish
   [The pred pointers define the MST as an inverted tree rooted at s]
 }
```

$$T(n, m) = \sum_{u \in V} (\log n + \text{degree}(u) \log n) = \sum_{u \in V} (1 + \text{degree}(u)) \log n$$

$$= \log n \sum_{u \in V} (1 + \text{degree}(u)) = (\log n)(n + 2E) = \Theta((n + m) \log n).$$
Fig. 23: Prim’s algorithm example.
Prim's Algorithm Demo

Initialize $S = \text{any node.}$
Repeat $n - 1$ times:
• Add to tree the min weight edge with one endpoint in $S$.
• Add new node to $S$. 

![Graph showing Prim's algorithm](image-url)
Initialize $S = \text{any node.}$
Repeat $n - 1$ times:
• Add to tree the min weight edge with one endpoint in $S$.
• Add new node to $S$. 
Prim's Algorithm Demo

Initialize $S = \text{any node.}$
Repeat $n - 1$ times:

- Add to tree the min weight edge with one endpoint in $S$.
- Add new node to $S$. 
Prim's Algorithm Demo

Initialize \( S \) = any node.
Repeat \( n - 1 \) times:
- Add to tree the min weight edge with one endpoint in \( S \).
- Add new node to \( S \).
Prim's Algorithm Demo

Initialize $S =$ any node.
Repeat $n - 1$ times:
- Add to tree the min weight edge with one endpoint in $S$.
- Add new node to $S$.
Prim's Algorithm Demo

Initialize $S = \text{any node}$. Repeat $n - 1$ times:

- Add to tree the min weight edge with one endpoint in $S$.
- Add new node to $S$. 

![Graph with Prim's Algorithm shaded](image)
Prim's Algorithm Demo

Initialize $S = \text{any node.}$
Repeat $n - 1$ times:

• Add to tree the min weight edge with one endpoint in $S$.
• Add new node to $S$. 
Prim's Algorithm Demo

Initialize $S = \text{any node.}$
Repeat $n - 1$ times:
• Add to tree the min weight edge with one endpoint in $S$.
• Add new node to $S$. 

![Graph diagram]
Initialize $S =$ any node.
Repeat $n - 1$ times:
• Add to tree the min weight edge with one endpoint in $S$.
• Add new node to $S$. 

![Graph Illustration]
Prim's Algorithm Demo

Initialize $S =$ any node.
Repeat $n - 1$ times:
  • Add to tree the min weight edge with one endpoint in $S$.
  • Add new node to $S$.
Prim's Algorithm Demo

Initialize $S = \text{any node.}$
Repeat $n - 1$ times:
  • Add to tree the min weight edge with one endpoint in $S$.
  • Add new node to $S$. 

```markdown
Prim's Algorithm Demo

Initialize $S = \text{any node.}$
Repeat $n - 1$ times:
  • Add to tree the min weight edge with one endpoint in $S$.
  • Add new node to $S$.
```
Prim's Algorithm Demo

Initialize $S = \text{any node.}$
Repeat $n - 1$ times:
• Add to tree the min weight edge with one endpoint in $S$.
• Add new node to $S$. 

![Graph Diagram]
Prim's Algorithm Demo

Initialize $S = \text{any node}$.
Repeat $n - 1$ times:
  • Add to tree the min weight edge with one endpoint in $S$.
  • Add new node to $S$. 

![Diagram of Prim's Algorithm](image)
Prim's Algorithm Demo

Initialize $S = \text{any node.}$
Repeat $n - 1$ times:
• Add to tree the min weight edge with one endpoint in $S$.
• Add new node to $S$. 
Prim's Algorithm Demo

Initialize $S =$ any node.
Repeat $n - 1$ times:
- Add to tree the min weight edge with one endpoint in $S$.
- Add new node to $S$. 
Prim's Algorithm Demo

Initialize $S = \text{any node.}$
Repeat $n - 1$ times:
- Add to tree the min weight edge with one endpoint in $S$.
- Add new node to $S$. 
Prim's Algorithm Demo

Initialize $S = $ any node.
Repeat $n - 1$ times:
• Add to tree the min weight edge with one endpoint in $S$.
• Add new node to $S$. 

Diagram of a graph with nodes 1, 2, 3, 4, 5, 7, and 9 showing connections.
Boruvka’s Algorithm

The Boruvka algorithm can be summarized in one line:

Borůvka: Add ALL the safe edges and recurse.

Image compliments of Jeff Erickson at University of Illinois, Urbana-Champaign, who modified an existing image drawn by and available on Allie Brosh’s, “This is Why I’ll Never be an Adult”, Hyperbole and a Half website
Boruvka’s Algorithm

Add edges to a growing forest of trees (Prim’s algorithm in parallel)
• At each stage, find the minimum-weight edge that connects each tree to a different one, then add all such edges to the MST.
• Assume that the edge weights are all different, to avoid cycles.
• Hint: Maintain in a vertex-indexed array to identify the edge that connects each component to its nearest neighbor, and use the union-find data structure.

```
BoruvkaMST(G=(V,E), w) {
    initialize each vertex to be its own component
    A = {} // A holds edges of the MST
    while (there are two or more components) {
        for (each component C) {
            find the lightest edge (u,v) with u in C and v not in C
            add {u,v} to A (unless it is already there)
        }
        apply DFS to graph (V, A), to compute the new components
    }
    return A // return final MST edges
}
```
Boruvka’s Algorithm Example

Fig. 24: Boruvka’s Algorithm.
Boruvka’s Algorithm: Analysis

How long does Boruvka’s algorithm take?

Each iteration (of the outer while loop) can be performed in $O(n + m)$ (DFS search)

But how many iterations are required in general?

Claim: There are never more than $O(\log n)$ iterations needed.
- Let $m$ denote the number of components at some stage
- Each component merges with at least 1 other component, so at most we have $m/2$ components, and at least we have 1
- Therefore, the number of components decreases by at least half each time

Total running time is $O((n+m) \log n)$ time $\approx O(m \log n)$ (since $n$ is asymptotically no larger than $m$)
Lexicographic Tiebreaking

To remove the assumption that all edge costs are distinct: perturb all edge costs by tiny amounts to break any ties.

Impact. Kruskal and Prim only interact with costs via pairwise comparisons. If perturbations are sufficiently small, MST with perturbed costs is MST with original costs.

e.g., if all edge costs are integers, perturbing cost of edge $e_i$ by $i / n^2$

Implementation. Can handle arbitrarily small perturbations implicitly by breaking ties lexicographically, according to index.

```java
boolean less(i, j) {
    if (cost(e_i) < cost(e_j)) return true
    else if (cost(e_i) > cost(e_j)) return false
    else if (i < j) return true
    else return false
}
```
Deterministic comparison based algorithms.

- $O(m \log n)$: [Jarník, Prim, Dijkstra, Kruskal, Boruvka]
- $O(m \log \log n)$: [Cheriton-Tarjan 1976, Yao 1975]
- $O(m \beta(m, n))$: [Fredman-Tarjan 1987]
- $O(m \log \beta(m, n))$: [Gabow-Galil-Spencer-Tarjan 1986]
- $O(m \alpha(m, n))$: [Chazelle 2000]

Holy grail. $O(m)$.

Notable.

- $O(m)$ randomized: [Karger-Klein-Tarjan 1995]
- $O(m)$ verification: [Dixon-Rauch-Tarjan 1992]

Euclidean.

- 2-d: $O(n \log n)$. compute MST of edges in Delaunay
- k-d: $O(k n^2)$. dense Prim
For the following graph:

1. List the edges of the minimum spanning tree in the order that they are added by Kruskal’s algorithm. (List only the edges that are in the MST.) You may list edges either by their weight (e.g., “7”) or by their endpoints (e.g., “(b, d)”).

2. Assuming that ‘a’ is the start vertex, list the edges of the minimum spanning tree in the order that they are added by Prim’s algorithm. (List only the edges that are in the MST.)
Next Time

• Dijkstra’s Algorithm for Shortest Paths
• Section 24.3 (CLRS)
• Section 4.4(KT)