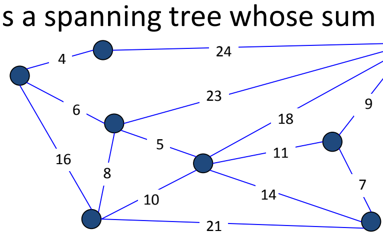
COMP 355
Advanced Algorithms

Algorithms for MSTs
Sections 4.5 (KT)

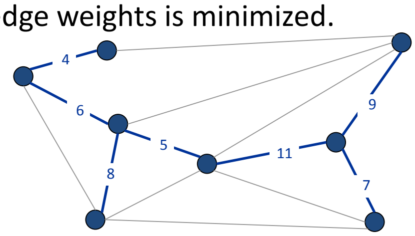
Rhodes College

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.



$G = (V, E)$



T, $\Sigma_{e \in T}\, c_e = 50$

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

can't solve by brute force

Applications

MST is fundamental problem with diverse applications.

- Network design.
  - telephone, electrical, hydraulic, TV cable, computer, road
- Approximation algorithms for NP-hard problems.
  - traveling salesperson problem, Steiner tree
- Indirect applications.
  - max bottleneck paths
  - LDPC codes for error correction
  - image registration with Renyi entropy
  - learning salient features for real-time face verification
  - reducing data storage in sequencing amino acids in a protein
  - model locality of particle interactions in turbulent fluid flows
  - autoconfig protocol for Ethernet bridging to avoid cycles in a network
- Cluster analysis.

MST Problem

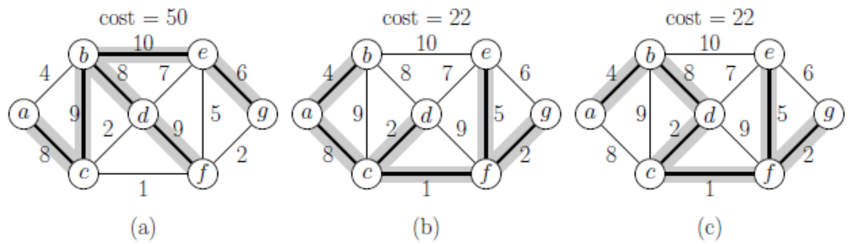
Given a connected, undirected graph G = (V,E), a *spanning tree* is an acyclic subset of edges T ⊆ E that connects all the vertices together.

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.

MST Problem



- Three spanning trees for the same graph
- (a) is not a MST
- (b) and (c) are both MSTs

Greedy Algorithms

Kruskal's algorithm. Start with T = ϕ. Consider edges in ascending order of cost. Insert edge e in T unless doing so would create a cycle.

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. Similar to Kruskal's, but easiest to implement on a parallel computer.

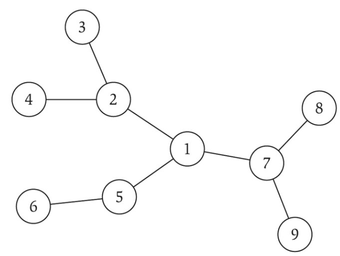
Remark. All three algorithms produce an MST.

Trees

Def. An undirected graph is a *tree* if it is connected and does not contain a cycle.

Theorem. Let G be an undirected graph on n nodes. Any two of the following statements imply the third.

– G is connected.
– G does not contain a cycle.
– G has n-1 edges.



MST Terms

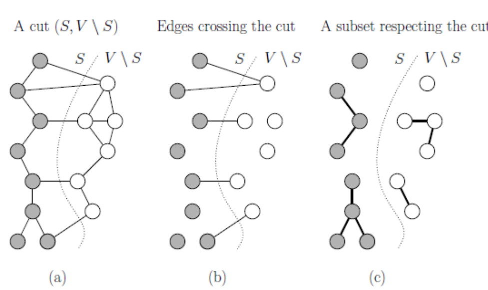
Def. We say that a subset A ⊆ E is *viable* if A is a subset of edges in some MST.

Def. We say that an edge (u, v) ∈ E \ A is *safe* if A ∪ {(u, v)} is viable. (E \A means the edges of E that are not in A.)

When is an edge safe?

Let S be a subset of the vertices S ⊆ V .

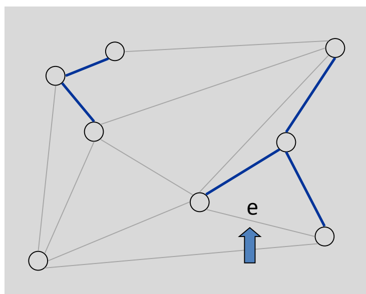
- A cut (S, V \ S) is a partition of the vertices into two disjoint subsets (a)
- An edge (u, v) crosses the cut if u ∈ S and v /∈ S (b)
- Given a subset of edges A, we say that a cut respects A if no edge in A crosses the cut(c)



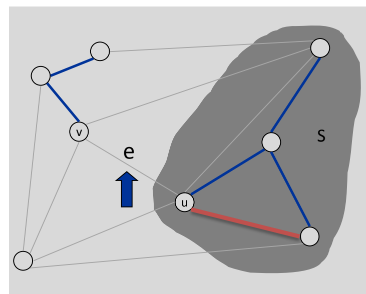
Kruskal's Algorithm: Proof of Correctness

Kruskal's algorithm. [Kruskal, 1956]

- Consider edges in ascending order of weight.
- Case 1: If adding e to T creates a cycle, discard e according to cycle property.
- Case 2: Otherwise, insert e = (u, v) into T according to cut property where S = set of nodes in u's connected component.



Case 1



Case 2

Implementation: Kruskal's Algorithm

Implementation. Use the union-find data structure.

- Build set T of edges in the MST.
- Maintain set for each connected component.
- O(m log n) for sorting and O(m $\underbrace{\alpha}_{\text{essentially a constant}}$ (m, n)) for union-find.

$m \le n^2 \Rightarrow$ log m is O(log n)

Kruskal's Algorithm

```
KruskalMST(G=(V,E), w) {
    A = {}                              // initially A is empty
    Place each vertex u in a set by itself
    Sort E in increasing order by weight w
    for each ((u, v) in this order) {
        if (find(u) != find(v)) {       // u and v in different trees
            add (u, v) to A             // join subtrees together
            union(u, v)                 // merge these two components
        }
    }
    return A
}
```

Kruskal's Algorithm Example

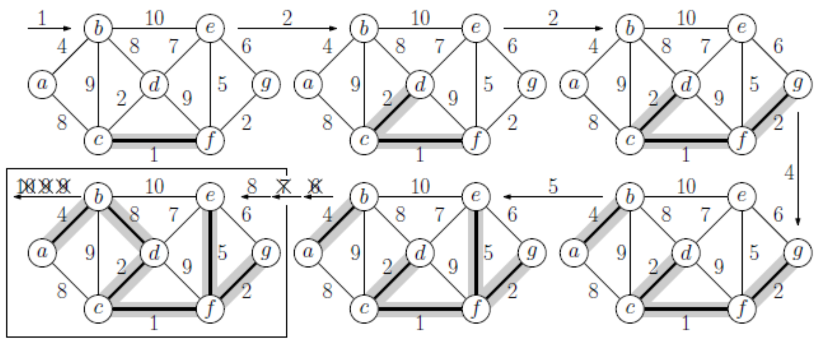
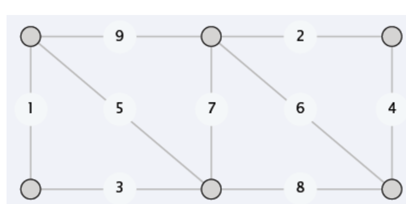
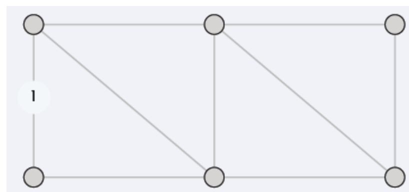


Fig. 21: Kruskal's Algorithm. Each vertex is labeled according to the set that contains it.

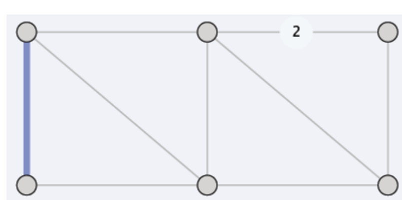
Kruskal's Algorithm Demo



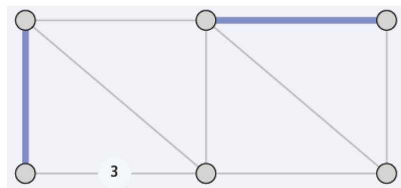
Kruskal's Algorithm Demo



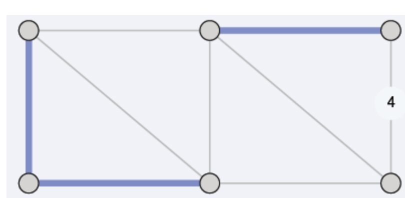
Kruskal's Algorithm Demo



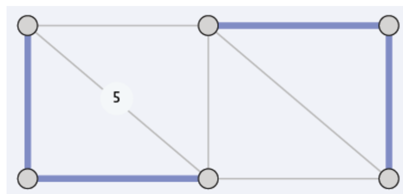
Kruskal's Algorithm Demo



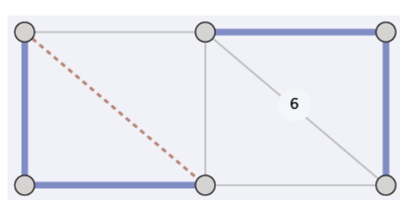
Kruskal's Algorithm Demo



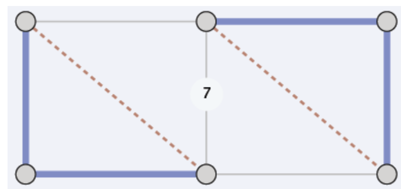
Kruskal's Algorithm Demo



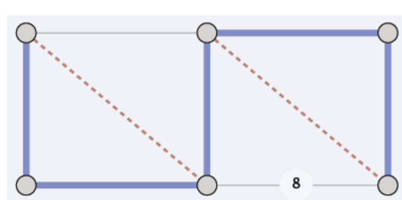
Kruskal's Algorithm Demo



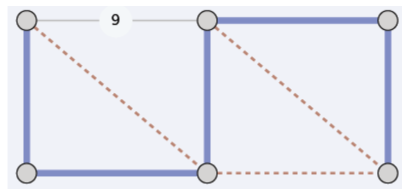
Kruskal’s Algorithm Demo



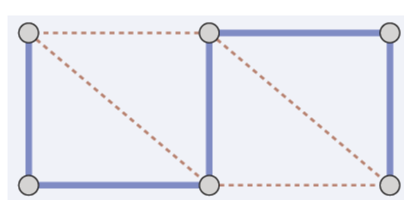
Kruskal’s Algorithm Demo



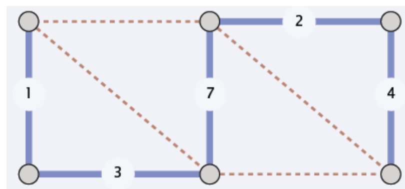
Kruskal's Algorithm Demo



Kruskal's Algorithm Demo



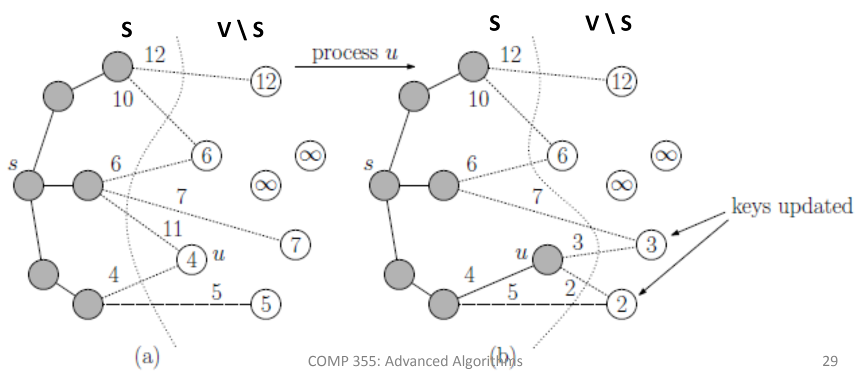
Kruskal's Algorithm Demo



Prim's Algorithm: Proof of Correctness

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] = cost of cheapest edge v to a node in S.
- Runtime is O(m log n)

Prim's Algorithm

```
PrimMST(G=(V,E), w, 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
    }
    [The pred pointers define the MST as an inverted tree rooted at s]
}
```

$$
\begin{aligned}
T(n,m) &= \sum_{u \in V} (\log n + \text{degree}(u) \log n) = \sum_{u \in V} (1 + \text{degree}(u)) \log n \\
&= \log n \sum_{u \in V} (1 + \text{degree}(u)) = (\log n)(n + 2E) = \Theta((n+m)\log n).
\end{aligned}
$$

Prim's Algorithm Example

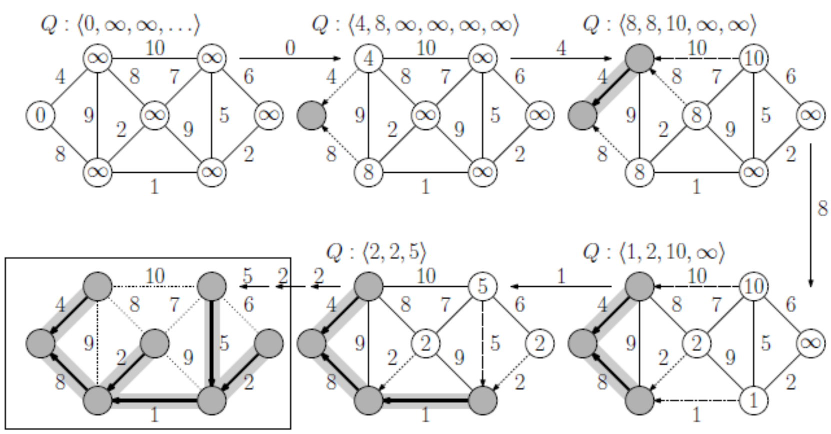
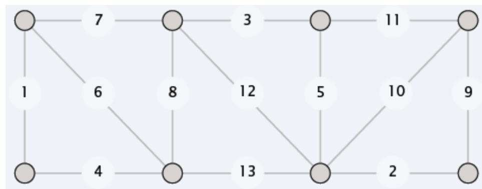


Fig. 23: Prim's algorithm example.

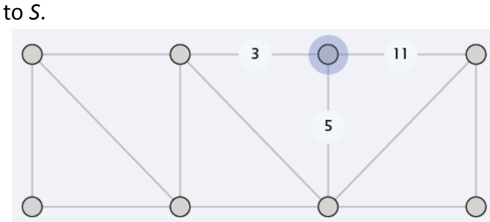
Prim's Algorithm Demo

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*.



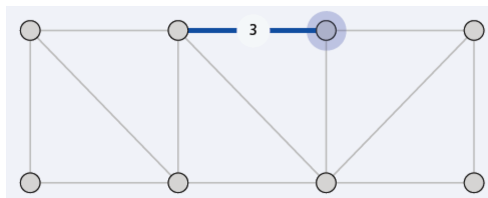
Prim's Algorithm Demo

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*.



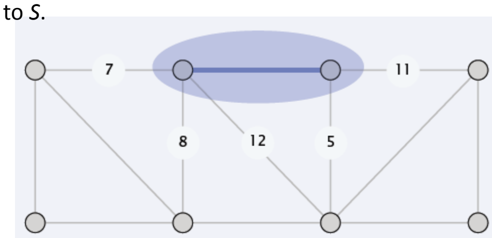
Prim's Algorithm Demo

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



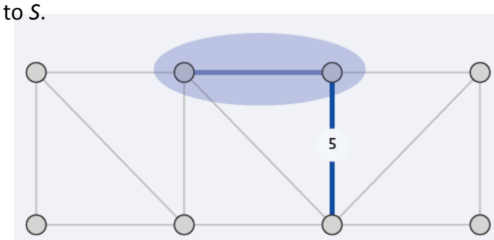
Prim's Algorithm Demo

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.



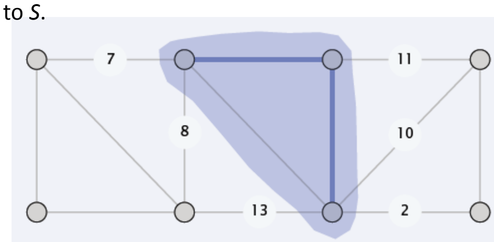
Prim's Algorithm Demo

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.



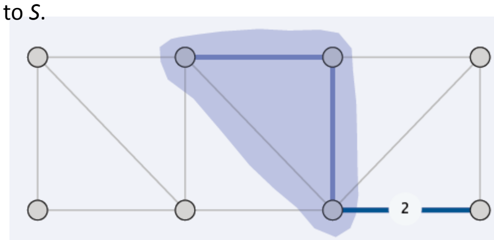
Prim's Algorithm Demo

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



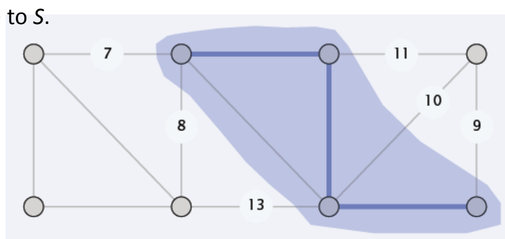
Prim's Algorithm Demo

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.



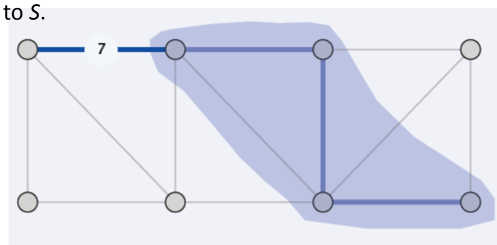
Prim's Algorithm Demo

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.



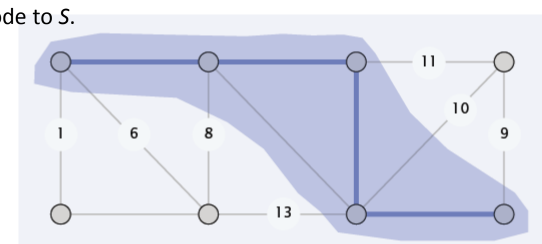
Prim's Algorithm Demo

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



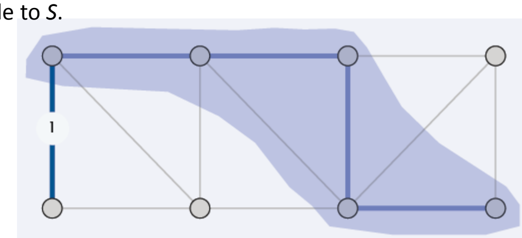
Prim's Algorithm Demo

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.



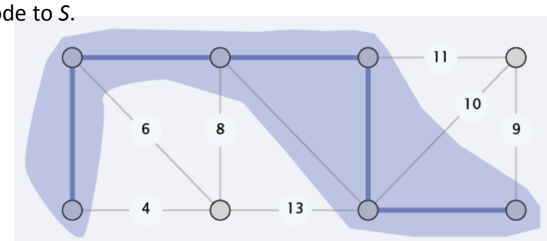
Prim's Algorithm Demo

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*.



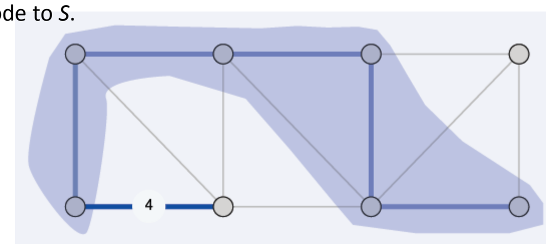
Prim's Algorithm Demo

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



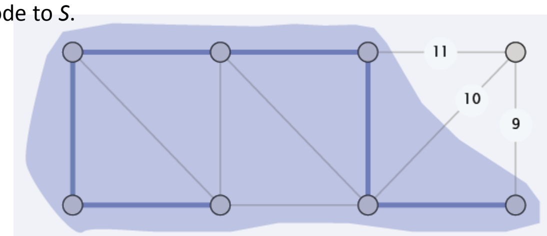
Prim's Algorithm Demo

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.



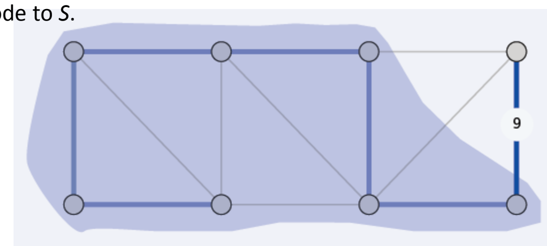
Prim's Algorithm Demo

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.



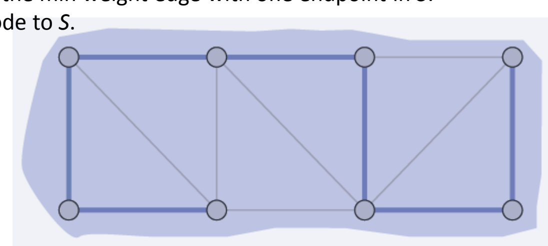
Prim's Algorithm Demo

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



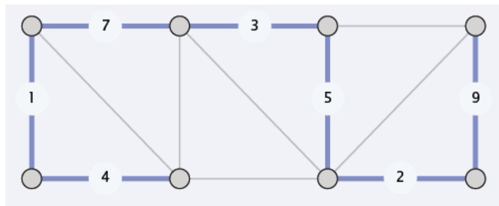
Prim's Algorithm Demo

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*.



Boruvka's Algorithm

The Boruvka algorithm can be summarized in one line:

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 (Kruskal'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.

Boruvka's Algorithm

```
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

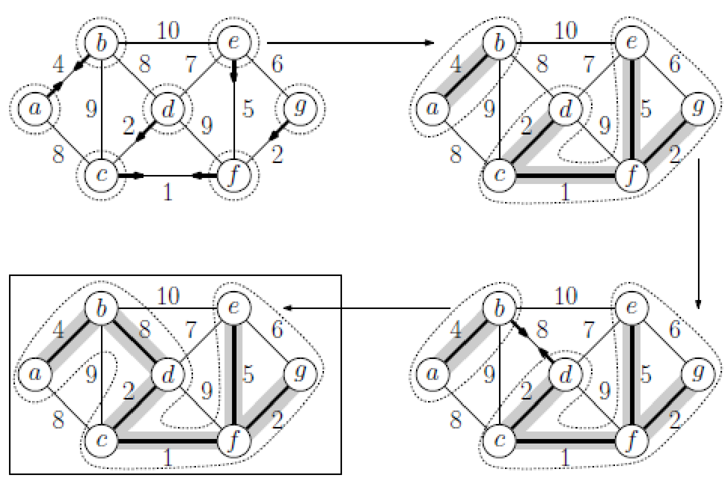


Fig. 24: Boruvka's Algorithm.

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 ≈ 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.

```
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
}
```

MST Algorithms: Theory

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 β(m, n)). [Fredman-Tarjan 1987]
- O(m log β(m, n)). [Gabow-Galil-Spencer-Tarjan 1986]
- O(m α (m, n)). [Chazelle 2000]

Holy grail. O(m).

Notable.

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

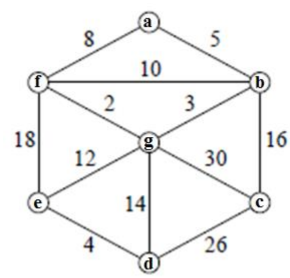
Euclidean.

- 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 4.4(KT)