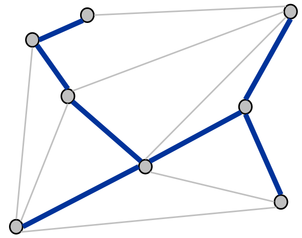
Minimum Spanning Tree

Prim's algorithm

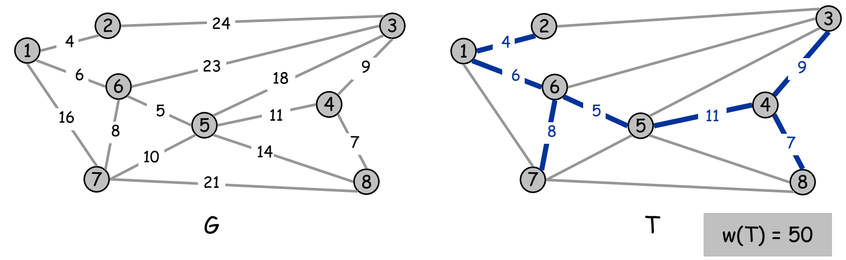
Kruskal's algorithm



Reference: Chapter 20, Algorithms in Java, 3rd Edition, Robert Sedgewick.

Minimum Spanning Tree

MST. Given connected graph G with positive edge weights, find a min weight set of edges that connects all of the vertices.



Cayley's Theorem (1889). There are V^{V-2} spanning trees on the complete graph on V vertices.

- Can't solve MST by brute force.

MST Origin

Otakar Boruvka (1926).

- Electrical Power Company of Western Moravia in Brno.
- Most economical construction of electrical power network.
- Concrete engineering problem is now a cornerstone problem in combinatorial optimization.

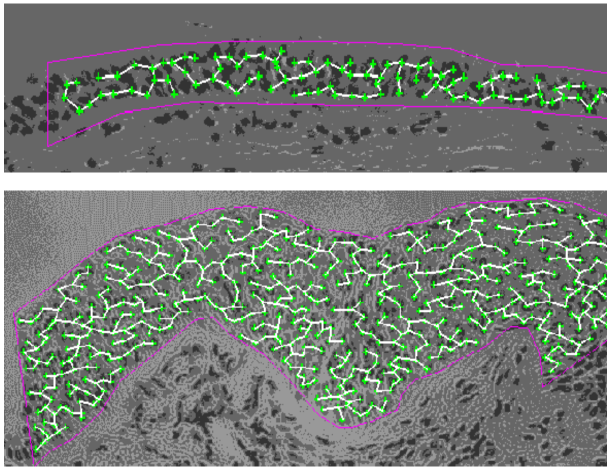
Applications

MST is fundamental problem with diverse applications.

- Network design.
  - telephone, electrical, hydraulic, TV cable, computer, road
- Cluster analysis.
  - analyzing fungal spore spatial patterns
  - microarray gene expression data clustering
  - finding clusters of quasars and Seyfert galaxies
- 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

Medical Image Processing

Arrangement of nuclei in skin cell for cancer research.

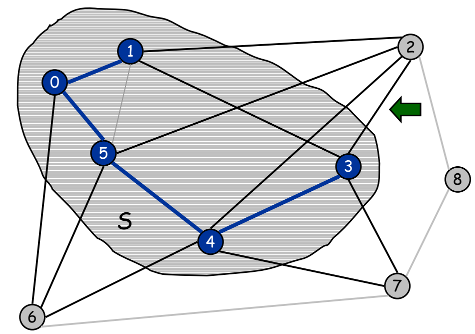


Prim's Algorithm

Prim's algorithm. (Jarník 1930, Dijkstra 1957, Prim 1959)

- Initialize T = ϕ, S = {s} for some arbitrary vertex s.
- Grow S until it contains all of the vertices:
  - let f be smallest edge with exactly one endpoint in S
  - add edge f to T
  - add other endpoint to S

S	T
0	-
1	0-1
5	0-5
4	5-4
3	4-3

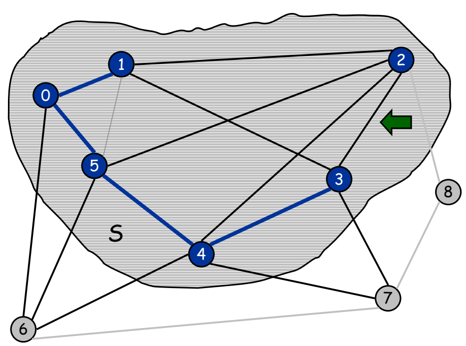


Prim's Algorithm

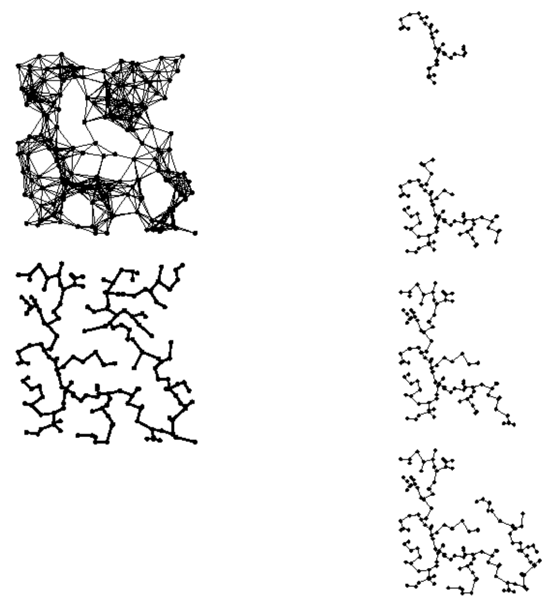
Prim's algorithm. (Jarník 1930, Dijkstra 1957, Prim 1959)

- Initialize T = ϕ, S = {s} for some arbitrary vertex s.
- Grow S until it contains all of the vertices:
  - let f be smallest edge with exactly one endpoint in S
  - add edge f to T
  - add other endpoint to S

S	T
0	-
1	0-1
5	0-5
4	5-4
3	4-3
2	3-2

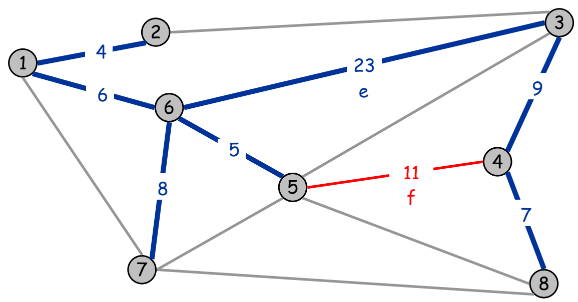


Prim's Algorithm: Example



Prim's Algorithm: Intuition of Proof of Correctness

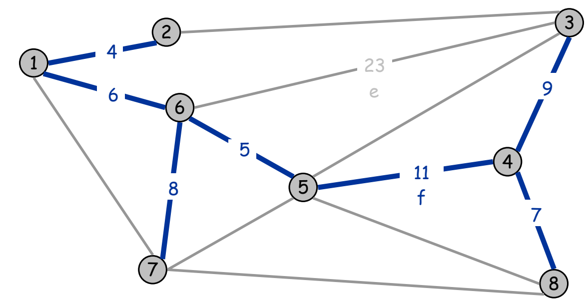
Observation. Given a spanning tree T. Let f be an edge not in T. Adding f to T creates a unique cycle. If $c_f < c_e$ for some edge e of cycle, then $T \cup \{ f \} - \{ e \}$ is a tree of lower cost.



T

Prim's Algorithm: Intuition of Proof of Correctness

Observation. Given a spanning tree T. Let f be an edge not in T. Adding f to T creates a unique cycle. If $c_f < c_e$ for some edge e of cycle, then $T \cup \{ f \} - \{ e \}$ is a tree of lower cost.



$T \cup \{ f \} - \{ e \}$

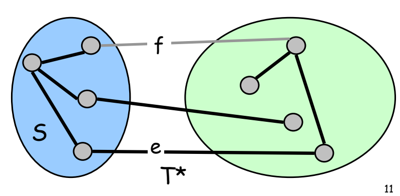
Prim's Algorithm: Proof of Correctness

Theorem. Upon termination of Prim's algorithm, T is a MST.
Proof. (by induction on number of iterations)

Invariant: There exists a MST T* containing all of the edges in T.

Base case: $T = \phi \Rightarrow$ every MST satisfies invariant.
Induction step: invariant true at beginning of iteration i.

- Let f be the edge that Prim's algorithm chooses.
- If $f \in T^*$, T* still satisfies invariant.
- Otherwise, consider cycle C formed by adding f to T*
  - let $e \in C$ be another arc with exactly one endpoint in S
  - $c_f \le c_e$ since algorithm chooses f instead of e
  - $T^* \cup \{ f \} - \{ e \}$ satisfies invariant

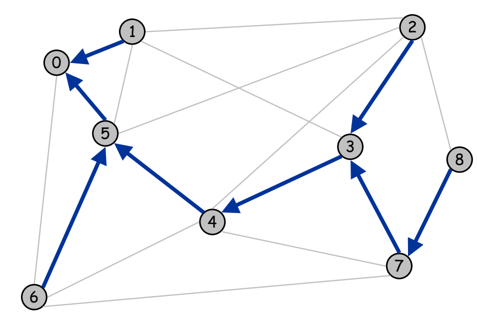


Spanning Tree Representation

How to represent a spanning tree?

- List of edges: `0-1 0-5 2-3 3-4 3-7 4-5 5-6 7-8`
- Parent-link representation: vertex indexed array `pred[v]`.

v	pred[v]
0	-
1	0
2	3
3	4
4	5
5	0
6	5
7	3
8	8

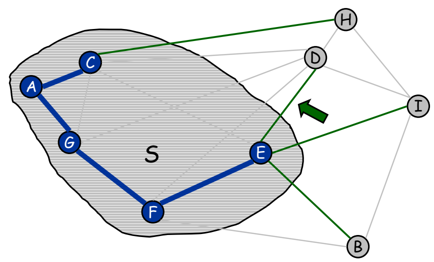


Prim's Algorithm

Maintain S = set of vertices in current tree.

- For each vertex not in S, maintain vertex in S to which it is closest.
- Choose next vertex `v` to add to S with min `dist[v]`.
- For each neighbor `w` of `v`, if `w` is closer to `v` than current neighbor in S, update `dist[w]`.

v	pred	dist
A	A	-
B	E	15
C	A	-
D	E	9
E	F	-
F	G	-
G	A	-
H	C	23
I	E	11

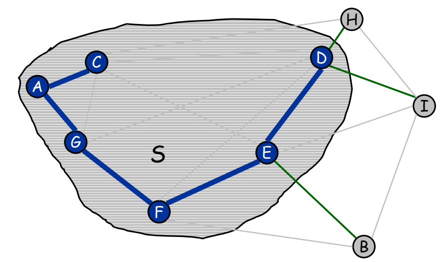


Prim's Algorithm

Maintain S = set of vertices in current tree.

- For each vertex not in S, maintain vertex in S to which it is closest.
- Choose next vertex `v` to add to S with min `dist[v]`.
- For each neighbor `w` of `v`, if `w` is closer to `v` than current neighbor in S, update `dist[w]`.

v	pred	dist
A	A	-
B	E	15
C	A	-
D	E	-
E	F	-
F	G	-
G	A	-
H	D	4
I	D	6



Weighted Graphs

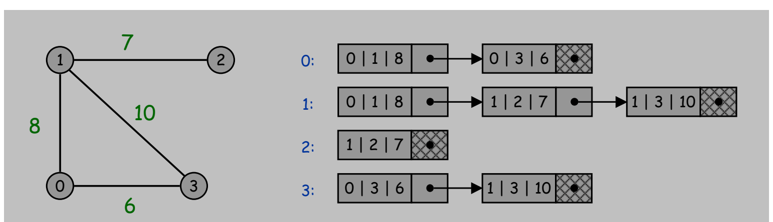
Weights.

- Method 1: graph access function `G.cost(v, w)`.
- Method 2: modify adjacency list iterator to return `Edge`.

Tradeoffs.

map routing assignment

- Method 1 is easier with adjacency matrix or Euclidean weights. Method 2 is more general.



adjacency list of Edge objects

Prim's Algorithm

Adjacency list implementation.

- Initialize, `dist[v]` $= \infty$ and `dist[s] = 0`.
- Insert all vertices onto PQ.
- Repeatedly delete vertex v from PQ with min `dist[v]`.
  - for each v-w, if `(dist[w] > G.cost(v, w))`, update `dist[w]`

```
while (!pq.isEmpty()) {
   int v = pq.delMin();
   IntIterator i = G.neighbors(v);
   while (i.hasNext()) {
      int w = i.next();
      if (dist[w] > G.cost(v, w)) {
          dist[w] = G.cost(v, w);
          pq.decrease(w, dist[w]);
          pred[w] = v;
      }
   }
}
```

cost of edge v-w

decrease key

main loop

Priority Queues for Index Items

Index heap-based priority queue.

- Insert, delete min, test if empty.
- Decrease key.

Brute force array implementation.

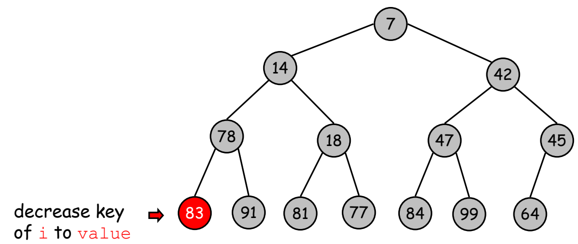
- Maintain vertex indexed array `dist[w]`.
- Decrease key: change `dist[w]`.
- Delete min: scan through `dist[w]` for each vertex `w`.

Operation	Prim	Array
insert	V	V
delete-min	V	V
decrease-key	E	1
is-empty	V	1
total		V^2

Priority Queues for Index Items

Index heap-based priority queue. (Sedgewick Program 9.12)

- Assumes elements are named `0` to `N-1`.
- Assumes priorities are of type `double`.
- Client: `pq.decrease(i, value)`.



How to decrease key of vertex i? Bubble it up.
How to know which heap node to bubble up? Maintains an extra array `qp[i]` that stores the heap index of vertex i.

Priority Queues for Index Items

Design issues.

- PQ maintains priorities; client accesses through PQ interface.
- Client maintains priorities; PQ accesses through client.
- Both maintain their own copy.

```
public void insert(int k, double value) {
   N++;
   pq[N] = k;
   qp[k] = N;
   priority[k] = value;
   fixUp(pq, N);
}

public void decrease(int k, double value) {
   priority[k] = value;
   fixUp(pq, qp[k]);
}

private void exch(int i, int j) {
   int swap = qp[i]; qp[i] = qp[j]; qp[j] = swap;
   pq[qp[i]] = i; pq[qp[j]] = j;
}
```


Prim's Algorithm: Implementation Cost Summary

		Priority Queue			
Operation	Prim	Array	Binary heap	d-way Heap	Fibonacci heap †
insert	V	V	log V	$d \log_d V$	1
delete-min	V	V	log V	$d \log_d V$	log V
decrease-key	E	1	log V	$\log_d V$	1
is-empty	V	1	1	1	1
total		V^2	E log V	$E \log_{E/V} V$	E + V log V

optimize parameters $\Rightarrow d = E/V$

† Individual ops are amortized bounds

Prim's Algorithm: Priority Queue Choice

The choice of priority queue matters in Prim implementation.

- Array: $\Theta(V^2)$.
- Binary heap: $O(E \log V)$.
- Fibonacci heap: $O(E + V \log V)$.

Best choice depends on whether graph is SPARSE or DENSE.

- 2,000 vertices, 1 million edges. Heap: 2-3x slower.
- 100,000 vertices, 1 million edges. Heap: 500x faster.
- 1 million vertices, 2 million edges. Heap: 10,000x faster.

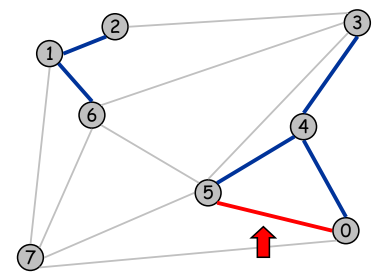
Bottom line.

- Array implementation optimal for dense graphs.
- Binary heap far better for sparse graphs.
- Fibonacci heap best in theory, but not in practice.

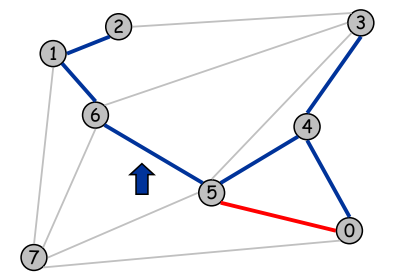
Kruskal's Algorithm

Kruskal's algorithm (1956).

- Initialize forest F = ϕ.
- Consider edges in ascending order of weight.
- If adding edge e to forest F does not create a cycle, then add it. Otherwise, discard e.

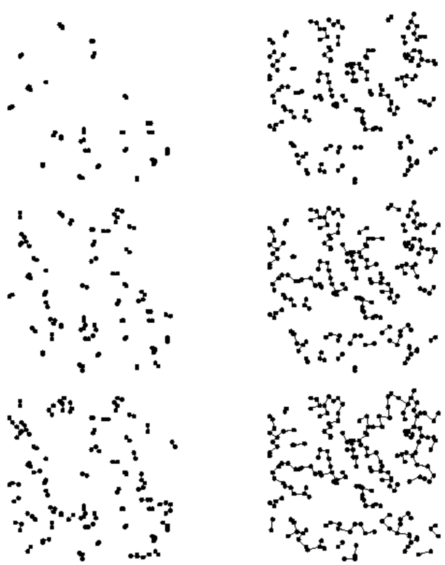


Case 1: adding `5-0` creates a cycle



Case 2: adding `5-6` connects 2 components

Kruskal's Algorithm: Example



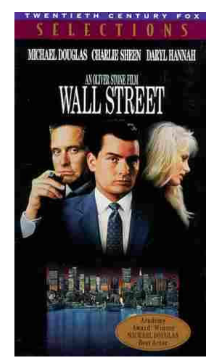
clusters

Kruskal's Algorithm: Proof of Correctness

Theorem. Upon termination of Kruskal's algorithm, F is a MST.

Proof. Identical to proof of correctness for Prim's algorithm except that you let S be the set of nodes in component of F containing v.

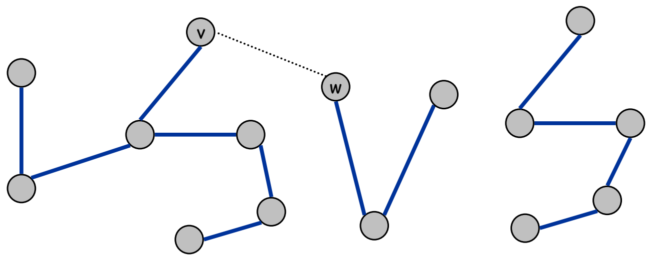
Greed is good. Greed is right. Greed works. Greed clarifies, cuts through, and captures the essence of the evolutionary spirit." - Gordon Gecko



Kruskal's Algorithm: Implementation

How to check if adding an edge to F would create a cycle?

- Naïve solution: DFS in O(V) time.
- Clever solution: union-find in O(log* V) amortized time.
  - each tree in forest F corresponds to a set
  - adding `v-w` creates a cycle if `v` and `w` are in same component
  - when adding `v-w` to forest F, merge sets containing `v` and `w`



F

Kruskal's Algorithm: Implementation

```
public class MST {
   private Edge[] mst;           // list of all edges in mst

   public MST(Graph G) {
      mst = new Edge[G.V()];
      Edge[] edges = G.edges();  // list of all edges in G
      Arrays.sort(edges);        // sort them by weight

      UnionFind uf = new UnionFind(G.V());
      for (int i = 0, k = 1; i < G.E(); i++) {
         int v = edges[i].v();
         int w = edges[i].w();
         if (!uf.find(v, w)) {   // v-w does not create a cycle
            uf.unite(v, w);      // merge v and w components
            mst[k++] = edges[i]; // add edge to mst
         }
      }
   }
}
```


Kruskal's Algorithm: Running Time

Operation	Frequency	Cost
sort	1	E log E
union	V - 1	log* V †
find	E	log* V †

† Amortized bound using weighted quick union with path compression.

Kruskal running time: O(E log V).

If edges already sorted. O(E log* V) time.

recall: log* V ≤ 5 in this universe

Advanced MST Algorithms

Year	Worst Case	Discovered By
1975	E log log V	Yao
1976	E log log V	Cheriton-Tarjan
1984	E log* V, E + V log V	Fredman-Tarjan
1986	E log (log* V)	Gabow-Galil-Spencer-Tarjan
1997	E α (E, V) log α (E, V)	Chazelle
2000	E α (E, V)	Chazelle
20??	E	???

Deterministic Comparison Based MST Algorithms

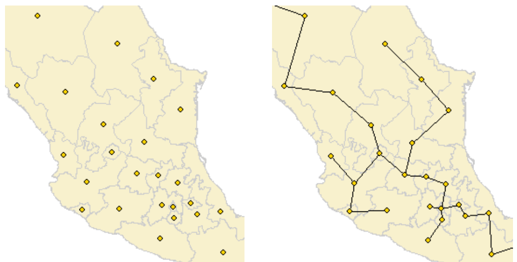
Year	Problem	Time	Discovered By
1976	Planar MST	E	Cheriton-Tarjan
1992	MST Verification	E	Dixon-Rauch-Tarjan
1995	Randomized MST	E	Karger-Klein-Tarjan

Related Problems

Euclidean MST

Given N points in the plane, find MST connecting them.

- Distances between point pairs are Euclidean distances.



Brute force: compute $\Theta(N^2)$ distances and run Prim's algorithm.

- Memory and running time are $\Theta(N^2)$, which is quadratic in input size.
- Can use squares of distances to avoid taking square roots.

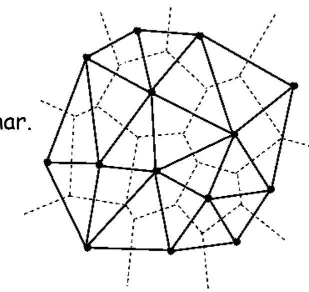
Is it possible to do better by exploiting the geometry?

Euclidean MST

Key geometric fact. Edges of the Euclidean MST are edges of the Delaunay triangulation.

Euclidean MST algorithm.

- Compute Voronoi diagram to get Delaunay triangulation.
- Run Kruskal's MST algorithm on Delaunay edges.



Running time: O(N log N).

- Fact: $\leq$ 3N - 6 Delaunay edges since it's planar.
- O(N log N) for Voronoi.
- O(N log N) for Kruskal.

Lower bound. Any comparison-based Euclidean MST algorithm requires Ω(N log N) comparisons.

Optimal Message Passing

Optimal message passing.

- Distribute message to N agents.
- Each agent i can communicate with some of the other agents j, but their communication is (independently) detected with probability p_{ij}.
- Group leader wants to transmit message to all agents so as to minimize overall probability of detected.

Objective.

- Find tree T that minimizes: $1 - \prod_{(i,j) \in T} (1 - p_{ij})$
- Or equivalently, that maximizes: $\prod_{(i,j) \in T} (1 - p_{ij})$
- Or equivalently, that maximizes: $\sum_{(i,j) \in T} \log (1 - p_{ij})$

Algorithm. MST with weights = - log (1 - p_{ij}). Weights p_{ij} also work!