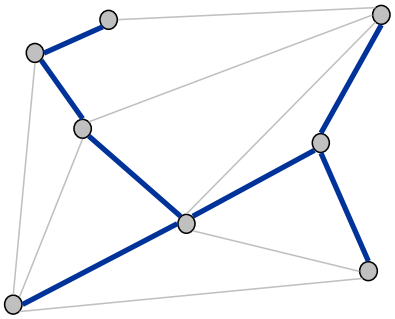
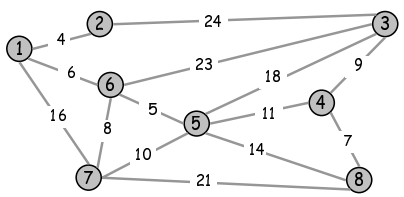
Minimum Spanning Tree



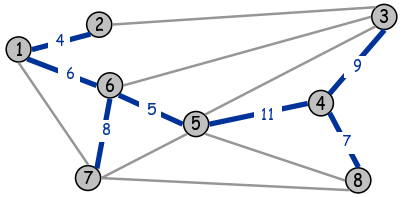
Some of these lecture slides are adapted from material in:
- *Algorithms in C, 3rd Edition, Part 5*, R. Sedgewick.

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.



G



T

w(T) = 50

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.

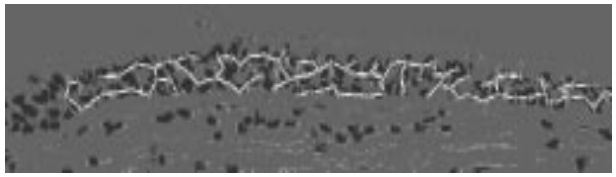
Applications

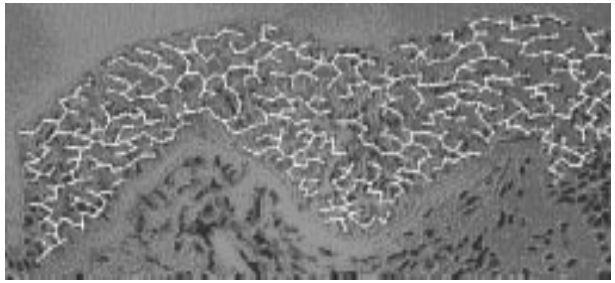
MST is fundamental problem with diverse applications.

- Nework design.
  - telephone, electrical, hydraulic, TV cable, computer, road
- Cluster analysis.
  - delete long edges leaves connected components
  - finding clusters of quasars and Seyfert galaxies
  - analyzing fungal spore spatial patterns
- Approximation algorithms for NP-hard problems.
  - metric TSP, Steiner tree
- Indirect applications.
  - max bottleneck paths
  - learning salient features for real-time face verification
  - modeling locality of particle interactions in turbulent fluid flow
  - reducing data storage in sequencing amino acids in a protein

Medical Image Processing

Arrangement of nuclei in skin cell for cancer research.





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: $\sum_{(i,j) \in T} \log(1 - p_{ij})$

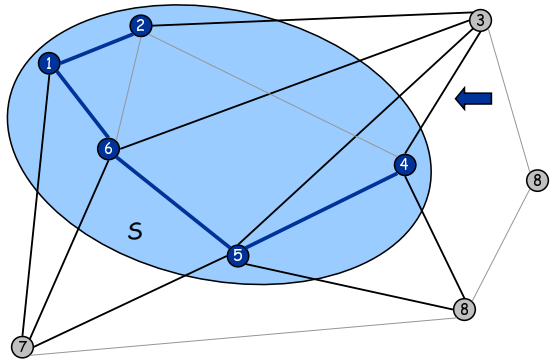
Algorithm. MST with weights = - log (1 - p_{ij}). Weights p_{ij} also work!

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
1	-
2	1-2
6	1-6
5	6-5
4	5-4

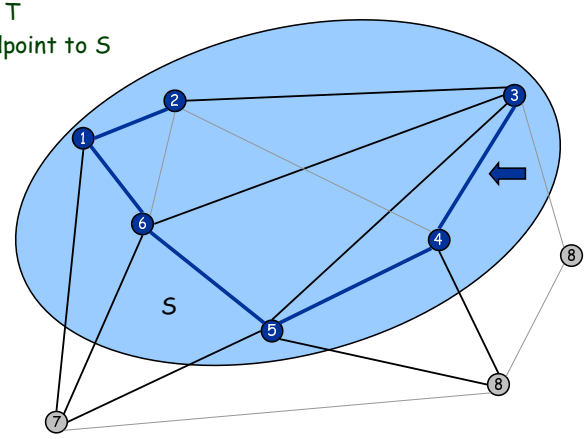


Prim's Algorithm

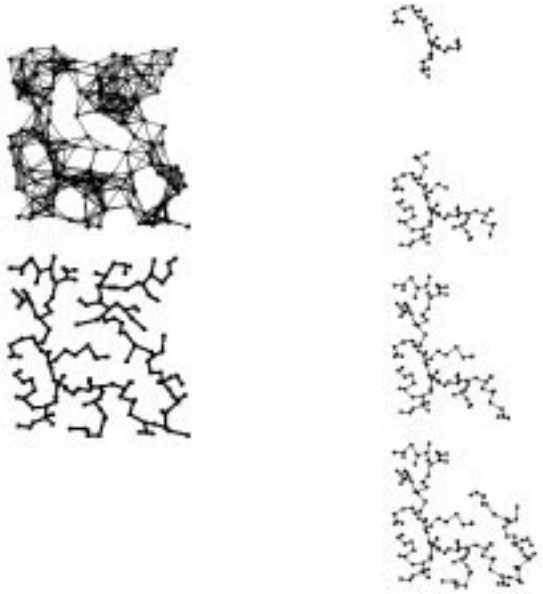
Prim's algorithm. (Jarník 1930, Dijkstra 1957, Prim 1959)

- 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
1	-
2	1-2
6	1-6
5	6-5
4	5-4
3	4-3



Prim's Algorithm: Example



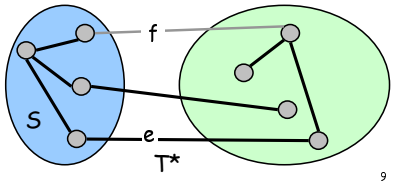
Prim's Algorithm: Proof of Correctness

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

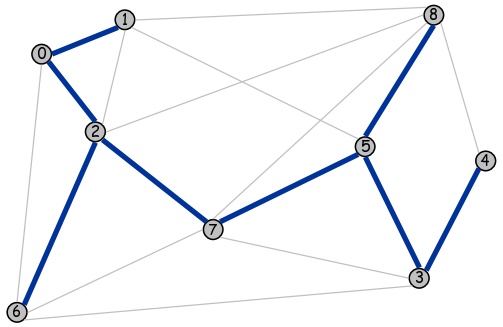


Spanning Tree Representation

How to represent a spanning tree?

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

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



Prim's Algorithm: Adjacency Matrix Implementation

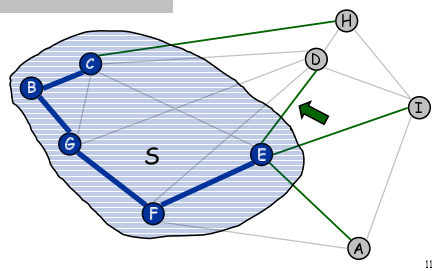
Use adjacency matrix.

- 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]`

```
if (G->wt[v][w] < dist[w]) {
   dist[w] = G->wt[v][w];
   pred[w] = v;
}
```

distance matrix

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



Prim's Algorithm: Adjacency Matrix Implementation

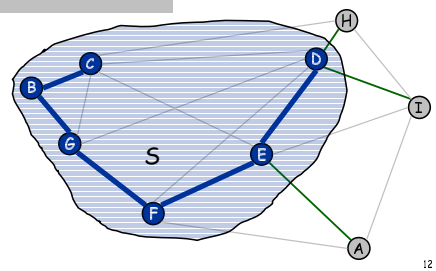
Use adjacency matrix.

- 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]`

```
if (G->wt[v][w] < dist[w]) {
   dist[w] = G->wt[v][w];
   pred[w] = v;
}
```

distance matrix

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

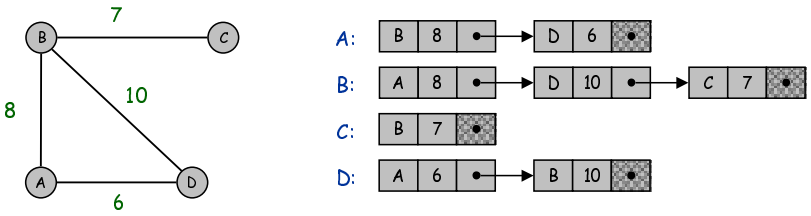


Prim's Algorithm: Adjacency List Implementation

Use adjacency list.

- Maintain extra field in each node of adjacency list to store edge weight.

```
typedef struct node *link;
 struct node {
    int w;
    double wt;
    link next;
 };
```



Prim's Algorithm: Adjacency List Implementation

Adjacency list implementation.

- Initialize $\pi[s] = 0$, pred[s] = s, $\pi[v] = \infty$, and pred[v] = -1.
- Insert all nodes onto PQ.
- Repeatedly delete node v with min $\pi[v]$ from PQ.
  - for each (v, w), if $c(v, w) < \pi[w]$ then update $\pi[w] = c(v, w)$

```
while (!PQisempty()) {
   v = PQdelmin();
   for (t = G->adj[v]; t != NULL; t = t->next) {
      w = t->w;
      if (t->wt < pi[w]) {
         pi[w] = t->wt;          <- decrease key
         pred[w] = v;            <- cost of edge (v, w)
      }
   }
}
```

Main Loop

Priority Queues for Index Items

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

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

Client passes index i of element of key to decrease.

- Index heap maintains an extra array, such that qp[i] stores the heap position of element with index i.

```
PQdeckey(int i) { fixUp(pq, qp[i]); }
```

Design issues.

1. PQ implementation maintains Keys; client accesses Keys only through handles provided by implementation.
2. Client maintains Keys; PQ implementation accesses Keys only through handles provided by client (array indices).

Prim's Algorithm: Implementation Cost Summary

		Priority Queue		
Operation	Prim	Array	Binary heap	Fibonacci heap †
insert	V	V	log V	1
delete-min	V	V	log V	log V
decrease-key	E	1	log V	1
is-empty	V	1	1	1
total		V^2	E log V	E + V log V

† Individual ops are amortized bounds

Prim's Algorithm: Priority Queue Choice

The choice of priority queue matters in Prim implementation.

- Array: $O(V^2)$.
- 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.

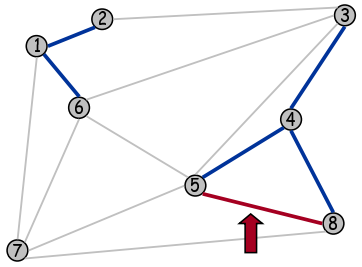
Bottom line.

- Classic Prim is optimal for dense graphs.
- Binary heap far better for sparse graphs.
- Fibonacci heap best in theory, but not in practice.

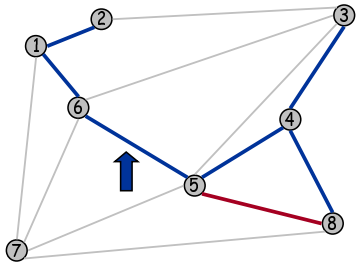
Kruskal's Algorithm

Kruskal's algorithm (1956).

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



Case 1: {5, 8}



Case 2: {5, 6}

Kruskal's Algorithm: Example

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.

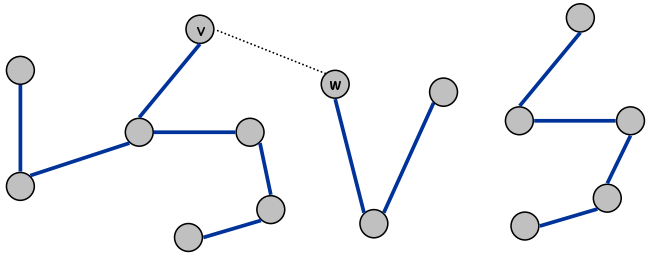
Corollary. "Greed is good. Greed is right. Greed works. Greed clarifies, cuts through, and captures the essence of the evolutionary spirit."

Gordon Gecko
(Michael Douglas)

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 $O(\log^* V)$.
  - each tree in forest corresponds to a set
  - to see if adding edge between v and w creates a cycle, check if v and w are already in same component
  - when adding v-w to forest F, merge sets containing v and w



Kruskal's Algorithm: Implementation

Kruskal's Algorithm

```
void GRAPHmst(Graph G, Edge F[]) {
   int i, k, v, w;
   Edge *edges = GRAPHedges(G);      <- list of all edges in G

   sort(edges, 0, G->E - 1);         <- sort edges by weight

   UFinit(G->V);
   for (i = k = 0; i < E && k < G->V - 1; i++) {
      v = edges[i].v;
      w = edges[i].w;

      if (!UFfind(v, w)) {           <- if v-w does not create a cycle,
         UFunion(v, w);                 add it to forest F and merge
         F[k++] = edges[i];             components containing v and w
      }
   }                                 <- array of k edges in 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: in this universe $\log^* V \leq 5$.

Advanced MST Algorithms

Deterministic comparison based algorithms.

- $O(E \log V)$ Prim, Kruskal, Boruvka.
  $O(V^2)$ Prim, Boruvka.
- $O(E \log \log V)$. Cheriton-Tarjan (1976), Yao (1975).
- $O(E \log^* V)$, $O(E + V \log V)$ Fredman-Tarjan (1984).
- $O(E \log (\log^* V))$. Gabow-Galil-Spencer-Tarjan (1986).
- $O(E \alpha (E, V) \log \alpha (E, V))$. Chazelle (1997).
- $O(E \alpha (E, V))$. Chazelle (2000).
- $O(E)$. Holy grail.

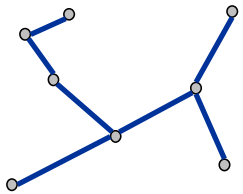
Worth noting.

- $O(E)$ verification. Dixon-Rauch-Tarjan (1992).
- $O(E)$ randomized. Karger-Klein-Tarjan (1995).

Euclidean MST

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

- Distances between point pairs are Euclidean distances.



Compute $\Theta(N^2)$ distances and run Prim's algorithm on complete graph.

- Memory and running time are quadratic in input size.
- Can use squares of distances to avoid taking square roots.

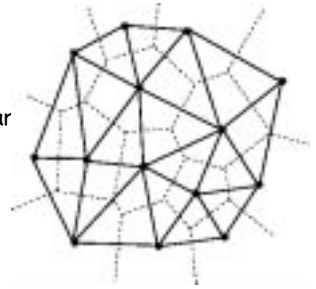
Is it possible to do better by exploiting the geometry?

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 $\Omega(N \log N)$ comparisons.