ID1020 2018-10-09 #13

Minimum spanning trees

A spanning tree of G is a subgraph T that is
- Connected
- Acyclic
- Includes all vertices

In: Connected undirected graph G with positive edge weights.
Out: Min weight spanning tree.

Greedy algorithm

Connected graph with distinct edge weights ⇒ MST exists and is unique
A cut in a graph is a partition of its vertices into two non-empty sets
A crossing edge connects vertices in different sets.

Given any cut, the crossing edge of min weight is in the MST
Proof in presentation

- Start with all edges colored gray
- Find cut with no black crossing edges; color its min-weight edge black.
- Repeat until V-1 edges are colored black. (E = V-1)

This computes the MST, Proof in presentation.

If the weights are not distinct, this algorithm still works.
If all edges are not connected, a single MST does not exist.

Weighted edge API
- Edge(int v, int w, double weight)
- int either()
- int other(int v)
- compareTo(Edge that)
- double weight()

Edge-weighted graph: vertex-indexed array of edges

Minimum spanning tree API
- MST(EdgeWeightedGraph G)
- Iterable<Edge> edges()
- double weight()

Kruskal's algorithm
- Consider edges in ascending order of weight.
- Add next edge to tree T unless doing so would create a cycle.

Union-find can be used to detect cycles.
(and should)

computes MST in time proportional to $E \log E$ (worst case)

Prim's algorithm
- Start with vertex 0 and greedily grow tree T
- Add to T the min weight edge with exactly-one endpoint in T
- Repeat until V-1 edges

Lazy and eager implementation in presentation

## Shortest path

- Single source, single sink, source-sink, all pairs.

Restrictions on edge weights
- Nonnegative weights
- Euclidean weights
- Arbitrary weights

Api
DirectedEdge(int v, int w, double weight)
int from()
int to()
double weight()
String toString()

---

Shortest-paths tree (SPT), represented by two arrays
- distTo[v] is length of shortest path from s to v.
- edgeTo[v] is last edge on shortest path from s to v

Relax edge $e = v \to w$
- distTo[v] is length of shortest known path from s to v.
- distTo[w] is length of shortest known path from s to w.
- edgeTo[w] is last edge on shortest known path from s to w.
- If $e = v \to w$ gives shorter path to w through v, update both distTo[w] and edgeTo[w].

Let G be an edge-weighted digraph.
Then distTo[] are the shortest path distances from s iff
- distTo[s] = 0
- For each vertex v, distTo[v] is the length of some path from s to v.
- For each edge $e = v \to w$, distTo[w] ≤ distTo[v] + e.weight().

Proof in presentation

Generic algorithm (to compute SPT from s)
- Initialize distTo[s] = 0 and distTo[v] = ∞ for all other vertices.
- Repeat until optimality conditions are satisfied:
  - Relax any edge.

Dijkstra's algorithm
- Consider vertices in increasing order of distance from s.
  (non-tree vertex with the lowest distTo[] value)
- Add vertex to tree and relax all edges pointing from that vertex.

Acyclic shortest paths
- Consider vertices in topological order.
- Relax all edges pointing from that vertex.