Minimum Spanning Trees: Prims Algorithm

A screenshot of a computer program

Description automatically generated

Prim’s algorithm

Start with any vertex as a single-vertex tree; then add V-1 edges to it, always taking next (coloring black) the minimum weight edge that connects a vertex on the tree to a vertex not yet on the tree (a crossing edge for the cut defined by tree vertices).

Theorem: Prim’s algorithm computes the MST of any connected edge-weighted graph.

A diagram of a cross-line diagram

Description automatically generated

Data structures

->Vertices on the tree: We user a vertex-indexed Boolean array marked[], where marked[v] is true if v is on the tree.

->Edges on the tree : We use one of two data structures: a queue mst to collect MST edges or a vertex-indexed array edgeTo[] of Edge objects, where edgeTo[v] is the Edge that connects v to the tree.

->Crossing edges : We use a MinPQ<Edge> priority queue that compares edges by weight (see page 610).

Maintaining the set of crossing edges

Each time that we add an edge to the tree, we also add a vertex to the tree. To maintain the set of crossing edges, we need to add to the priority queue all edges from that vertex to any non-tree vertex (using marked[] to identify such edges). Also, any edge connecting the vertex just added to a tree vertex that is already on the priority queue now becomes ineligible (it is no longer a crossing edge because it connects two tree vertices).

An eager implementation removes ineligible edges from the priority queue. Lazy implementation checks the eligibility of the edge when it is removed.

Detailed Algorithm for lazy Prim’s MST

:start

input: Graph g

global boolean[] marked

global Queue<Edge> mst

global PriorityQueue<Edge> pq

// arbitrarily begin from 0

mark vertex 0 as visited

for edge in g.adj(0)

if the edge is not visited

pq.add(edge)

while pq is not empty

e = pq.remove()

if either of the vertices in e is not visited

mst.add(e)

for each unvisited vertex v in e

mark v

for each adjacent edge to v

if adjacent edge is not visited

pq.add(adjacent edge)

:end

A diagram of a graph

Description automatically generated with medium confidence

Runtime: The lazy version of Prim’s algorithm uses space proportional to E and time proportional to E log E (in the worst case) to compute the MST of a connected edge-weighted graph with E edges and V vertices.

Eager version of Prim’s algorithm

A few changes to the lazy version:

1 The priority queue only contains the edges between tree vertices and non-tree vertices.

2 Only keep the minimal edge for each non-tree vertex to a tree vertex.

Assume vertex v is added to the tree and vertices that form set W are adjacent to v.

Algorithm for the eager version

Input: EdgeWeightedGraph G

Output: PrimMST that gives MST