Lecture 28: Minimum Spanning Trees

Prof. John Wenskovitch

04/06/2016
Last Time

- Directed graphs and terminology
- Depth- and breadth-first search
- Finding directed cycles
- Topological sort
- Strongly connected components
Edge-Weighted Graphs

• To this point, we were assuming that all edges are equal.

• An edge-weighted graph is a graph model where we associate weights with each edge.
  – Weights are not necessarily distances (but can be) or costs (but also can be).
  – Weights may be zero or negative.
Minimum Spanning Tree

- A **spanning tree** of a graph is a connected subgraph with no cycles that includes all the vertices.
- A **minimum spanning tree (MST)** of an edge-weighted graph is a spanning tree whose total weight is no larger than the weight of any other spanning tree.
Storing Edge-Weighted Graphs
More terminology

• A **cut** of a graph is a partition of its vertices into two nonempty, disjoint sets.
  – Typically specified as a set of vertices, with the implicit assumption that the missing vertices are a part of the other set.

• A **crossing edge** of a cut is an edge that connects a vertex in one set with a vertex in the other.

• **Greedy algorithms** are a class of algorithms that always take the best available solution at each step, regardless of future consequences.
Prim’s Algorithm

- Basic idea:
  - Attach a new edge to a single growing tree at each step.
  - Start with any vertex as a single vertex tree, then add $V - 1$ edges to it.
  - Always take the remaining minimum weight edge that connects a vertex on the tree to a vertex not yet on the tree.
LazyPrimMST(EdgeWeightedGraph G) {
    pq = new MinPQ<Edge>();
    marked = new boolean[G.V()];
    mst = new Queue<Edge>();

    visit(G, 0);
    while (!pq.isEmpty()) {
        Edge e = pq.delMin();
        int v = e.either();
        int w = e.other(v);
        if (marked(v) && marked(w))     continue;
        mst.enqueue(e);
        if (!marked(v))        visit(G, v);
        if (!marked(w))        visit(G, w);
    } //while
} //LazyPrimMST

void visit(EWG G, int v) {
    marked[v] = true;
    for (Edge e: G.adj(v))
        if (!marked(e.other(v))
            pq.insert(e);
} //visit
Prim’s Algorithm – Visual

* marks new entries
* crossing edges (ordered by weight)

ineligible edges (gray)

04/06/2016
Minimum Spanning Trees

ALLEGHENY COLLEGE
Prim’s Algorithm – Analysis

• **Theorem:** Lazy Prim’s uses space proportional to $E$ and time proportional to $E \times \log(E)$ (in the worst case).

• **Proof:**
  – For space, in the worst case, all edges will be in the MinPQ at the same time (vertex 0 is connected to every other vertex, which in turn are not connected to each other).
  – For time, in the worst case we need to consider every edge, and the MinPQ takes $\log(E)$ time for insertion and retrieval. ■
Eager Prim

• **Observation:** We don’t need ALL of those edges in our priority queue. We only need the fastest (current) way to access each of the non-tree vertices.
  
  – Let’s put vertices on the priority queue instead of edges, and update their weights as necessary.
Eager Prim

```java
PrimMST(EdgeWeightedGraph G) {
    edgeTo = new Edge[G.V()];
    distTo = new double[G.V()];
    marked = new boolean[G.V()];
    for (int v = 0; v < G.V(); v++)
        distTo[v] = Double.POSITIVE_INFINITY;
    pq = new IndexMinPQ<Double>(G.V());
    distTo[0] = 0.0;
    pq.insert(0, 0.0);
    while (!pq.isEmpty())
        visit(G, pq.delMin());
} //PrimMST
```
Eager Prim

```java
void visit(EdgeWeightedGraph G, int v) {
    marked[v] = true;
    for (Edge e: G.adj(v)) {
        int w = e.other(v);
        if (marked[w])
            continue;
        if (e.weight() < distTo[w]) {
            edgeTo[w] = e;
            distTo[w] = e.weight();
            if (pq.contains(w))
                pq.changeKey(w, distTo[w]);
            else
                pq.insert(w, distTo[w]);
        } //if
    } //for
} //visit
```
Eager Prim – Visual

black: on MST
red: on pq
gray: not on MST

thick red: smallest on pq, next to add to MST
Eager Prim – Analysis

• **Theorem:** Eager Prim uses extra space proportional to $V$ and extra time proportional to $E \times \log(V)$ (in the worst case).

• **Proof:**
  – The number of vertices in the priority queue is at most $V$, and there are 3 vertex-indexed arrays.
  – The IndexMinPQ implements operations in time $\log(V)$. ■
Kruskal’s Algorithm

• **Observation:** Instead of worrying about building a single tree vertex-by-vertex, let’s just consider the edges from smallest to largest.
  
  – Add each edge to the MST as long as it doesn’t form a cycle, and stop after $V - 1$ edges are added.
Kruskal’s Algorithm

KruskalMST(EdgeWeightedGraph G) {
    mst = new Queue<Edge>();
    MinPQ<Edge> pq = new MinPQ<Edge>();
    for (Edge e: G.edges()) {
        pq.insert(e);
        UF uf = new UF(G.V());

        while (!pq.isEmpty() && mst.size() < G.V()-1) {
            Edge e = pq.delMin();
            int v = e.either(), w = e.other(v);
            if (uf.connected(v, w))
                continue;
            uf.union(v, w);
            mst.enqueue(e);
        } //while
    } //KruskalMST
}
Kruskal’s Algorithm – Visual
Kruskal’s Algorithm – Analysis

- **Theorem:** Kruskal’s algorithm uses space proportional to $E$ and time proportional to $E \times \log(E)$ (in the worst case).

- **Proof:**
  - We still need to compare all edges.
  - We still have a log-time priority queue.
Can We Do Better?

• No theoretical results say that there can’t be a linear-time MST algorithm, but we certainly haven’t found one yet.
• But we’re getting close.
  – Chazelle introduced an algorithm in 1997 that “could never be distinguished from a linear time algorithm in any conceivable practical situation (even though it is provably nonlinear), but it is so complicated that no one would use it in practice.”
• Practically speaking, Prim and Kruskal are also effectively linear algorithms for sparse graphs in most practical situations.
• Also, Prim and Kruskal don’t work for directed graphs. This becomes a much more difficult problem called “minimum cost arborescence.”
Any Questions?

http://goo.gl/forms/dDI9Rz5Gy9