Lecture 37: Graphs IV

CS 62
Fall 2017
Kim Bruce & Alexandra Papoutsaki
Single Source Shortest Path Problem

- From a starting node $s$, find the shortest path (and its length) to all other (reachable) nodes
- The collection of all shortest paths form a tree, called... the shortest path tree!
- If all edges have the same weight, we can use BFS.
- Otherwise ...
Single Source Shortest Path Problem

• If all edges have weights $\geq 0$ then use Dijkstra’s algorithm
• Essentially BFS with priority queue
• Priorities are best known distance to a node from $s$
• We can keep track of parent nodes to get shortest path
• Example of a greedy algorithm
Dijkstra’s algorithm (1956) pseudocode

Q = {}; //set with unvisited vertices
for (every vertex v in V) {
    dist[v] = Infinity;
    parents[v] = null;
    Q.add(v);
}

dist[s] = 0;
while (!Q.isEmpty()) {
    u = vertex in Q with min dist[u];
    Q.remove(u);
    for (every edge (u,v)) {
        tentative = dist[u] + weight(u,v);
        if (tentative < dist[v]) {
            dist[v] = tentative;
            parents[v] = u;
        }
    }
}

Dijkstra’s algorithm (1984) pseudocode

```plaintext
Q = new PriorityQueue();
for (every vertex v in V) {
    dist[v] = Infinity;
    parents[v] = null;
    Q.addWithPriority(v, dist[v]);
}
dist[s] = 0;
Q.addWithPriority(s, 0);
while (!Q.isEmpty()) {
    u = Q.extractmin();
    Q.remove(u);
    for (every edge (u, v)) {
        tentative = dist[u] + weight(u, v);
        if (tentative < dist[v]) {
            dist[v] = tentative;
            parents[v] = u;
            Q.reducePriority(v, tentative);
        }
    }
}
```
Run-time of Dijkstra

- Adding and removing from priority queue: $O(\log n)$
  - Each goes on and off once, so $O(n \log n)$
- `reduce_priority`: $O(\log n)$
  - Worst case, once for each edge, so $O(m \log n)$
- Total time: $O((m + n) \log n)$
Dijkstra on sample graph
Dijkstra on sample graph

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
</tr>
</thead>
<tbody>
<tr>
<td>Init</td>
<td>0&lt;sub&gt;A&lt;/sub&gt;</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td>A</td>
<td>0&lt;sub&gt;A&lt;/sub&gt;</td>
<td>8&lt;sub&gt;A&lt;/sub&gt;</td>
<td>2&lt;sub&gt;A&lt;/sub&gt;</td>
<td>5&lt;sub&gt;A&lt;/sub&gt;</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td>C</td>
<td>0&lt;sub&gt;A&lt;/sub&gt;</td>
<td>8&lt;sub&gt;A&lt;/sub&gt;</td>
<td>2&lt;sub&gt;A&lt;/sub&gt;</td>
<td>4&lt;sub&gt;C&lt;/sub&gt;</td>
<td>7&lt;sub&gt;C&lt;/sub&gt;</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td>D</td>
<td>0&lt;sub&gt;A&lt;/sub&gt;</td>
<td>6&lt;sub&gt;D&lt;/sub&gt;</td>
<td>2&lt;sub&gt;A&lt;/sub&gt;</td>
<td>4&lt;sub&gt;C&lt;/sub&gt;</td>
<td>5&lt;sub&gt;D&lt;/sub&gt;</td>
<td>10&lt;sub&gt;D&lt;/sub&gt;</td>
<td>7&lt;sub&gt;D&lt;/sub&gt;</td>
<td>∞</td>
</tr>
<tr>
<td>E</td>
<td>0&lt;sub&gt;A&lt;/sub&gt;</td>
<td>6&lt;sub&gt;D&lt;/sub&gt;</td>
<td>2&lt;sub&gt;A&lt;/sub&gt;</td>
<td>4&lt;sub&gt;C&lt;/sub&gt;</td>
<td>5&lt;sub&gt;D&lt;/sub&gt;</td>
<td>10&lt;sub&gt;D&lt;/sub&gt;</td>
<td>6&lt;sub&gt;E&lt;/sub&gt;</td>
<td>∞</td>
</tr>
<tr>
<td>B</td>
<td>0&lt;sub&gt;A&lt;/sub&gt;</td>
<td>6&lt;sub&gt;D&lt;/sub&gt;</td>
<td>2&lt;sub&gt;A&lt;/sub&gt;</td>
<td>4&lt;sub&gt;C&lt;/sub&gt;</td>
<td>5&lt;sub&gt;D&lt;/sub&gt;</td>
<td>10&lt;sub&gt;D&lt;/sub&gt;</td>
<td>6&lt;sub&gt;E&lt;/sub&gt;</td>
<td>∞</td>
</tr>
<tr>
<td>G</td>
<td>0&lt;sub&gt;A&lt;/sub&gt;</td>
<td>6&lt;sub&gt;D&lt;/sub&gt;</td>
<td>2&lt;sub&gt;A&lt;/sub&gt;</td>
<td>4&lt;sub&gt;C&lt;/sub&gt;</td>
<td>5&lt;sub&gt;D&lt;/sub&gt;</td>
<td>8&lt;sub&gt;G&lt;/sub&gt;</td>
<td>6&lt;sub&gt;E&lt;/sub&gt;</td>
<td>12&lt;sub&gt;E&lt;/sub&gt;</td>
</tr>
<tr>
<td>F</td>
<td>0&lt;sub&gt;A&lt;/sub&gt;</td>
<td>6&lt;sub&gt;D&lt;/sub&gt;</td>
<td>2&lt;sub&gt;A&lt;/sub&gt;</td>
<td>4&lt;sub&gt;C&lt;/sub&gt;</td>
<td>5&lt;sub&gt;D&lt;/sub&gt;</td>
<td>8&lt;sub&gt;G&lt;/sub&gt;</td>
<td>6&lt;sub&gt;E&lt;/sub&gt;</td>
<td>11&lt;sub&gt;F&lt;/sub&gt;</td>
</tr>
<tr>
<td>H</td>
<td>0&lt;sub&gt;A&lt;/sub&gt;</td>
<td>6&lt;sub&gt;D&lt;/sub&gt;</td>
<td>2&lt;sub&gt;A&lt;/sub&gt;</td>
<td>4&lt;sub&gt;C&lt;/sub&gt;</td>
<td>5&lt;sub&gt;D&lt;/sub&gt;</td>
<td>8&lt;sub&gt;G&lt;/sub&gt;</td>
<td>6&lt;sub&gt;E&lt;/sub&gt;</td>
<td>11&lt;sub&gt;F&lt;/sub&gt;</td>
</tr>
</tbody>
</table>

Follow the subscripts to find shortest path from start to any vertex
Spanning Trees

• A spanning tree $T$ of a graph $G$ is a subset of the edges of $G$ such that:
  • $T$ contains no cycles and
  • Every vertex in $G$ is connected to every other vertex using just the edges in $T$
• An unconnected graph has no spanning trees.
• A connected graph will have at least one spanning tree; it may have many
Minimum Spanning Trees

• A weighted graph is a graph that has a weight associated with each edge.
• If $G$ is a weighted graph, the cost of a tree is the sum of the costs (weights) of its edges.
• A tree $T$ is a minimum spanning tree of $G$ iff:
  • it is a spanning tree and
  • there is no other spanning tree whose cost is lower than that of $T$. 
Minimum Spanning Trees

- Application:
  - The cheapest way to lay cable that connects a set of points is along a minimum spanning tree that connects those points.

- Many algorithms exist to find minimum spanning trees, most run in $O(m \log m)$ time.

- In 1995 Karger, Klein & Tarjan found a linear time randomized algorithm, but there is no known linear time deterministic algorithm
Kruskal’s Algorithm

• Create forest $F$ with no edges, using vertices in $V$
• Sort the edges in the graph by their weight (smallest to largest)
• For each edge $e$ in sorted order:
  • if $e$ connects two different trees in $F$, then add $e$ to $F$
Kruskal on sample graph

(1,2):1
(2,3):2
(4,5):3
(6,7):3
(1,4):4
(2,5):4
(4,7):4
(3,5):5
(2,4):6
(3,6):6
(5,7):7
(5,6):8
Kruskal’s Algorithm pseudocode

A = {};
for(every vertex v in V) {
    make-set(v)
    for(every edge (u, v) ordered by increasing weight) {
        if(find (u) != find (v)) {
            A.add((u, v));
            union(u, v);
        }
    }
}
return A;

make-set(v) - makes a set from a single vertex v
find(v) - finds the set that v belongs to
union(u, v) - makes the union of the sets containing u and v

Union-find structure
Graph Algorithms

- Very important in practice!
- Sophisticated data structures
- Careful analysis of correctness and complexity
- CS 140: Algorithms