Graphs: More on Shortest Paths, Plus Minimum Spanning Trees

CSE 373
Data Structures and Algorithms

Today’s Outline

• Announcements
  – Homework #5 – due Thurs June 4

• Graphs
  – Shortest Paths Algorithms
  – Minimum Spanning Tree

Correctness: The Cloud Proof

How does Dijkstra’s decide which vertex to add to the Known set next?
• If path to V is shortest, path to W must be at least as long
  (or else we would have picked W as the next vertex)
• So the path through W to V cannot be any shorter!

Correctness: Inside the Cloud

Prove by induction on # of nodes in the cloud:
  Initial cloud is just the source with shortest path 0
  Assume: Everything inside the cloud has the correct shortest path
  Inductive step: Only when we prove the shortest path to some node v (which is not in the cloud) is correct, we add it to the cloud

When does Dijkstra’s algorithm not work?

Dijkstra’s vs BFS

At each step:
1) Pick closest unknown vertex
2) Add it to finished vertices
3) Update distances

Breadth-first Search

Dijkstra’s Algorithm

Some Similarities:

The Trouble with Negative Weight Cycles

A 2 B
C 2 D
E

What’s the shortest path from A to E?

Problem?
Minimum Spanning Trees
Given an undirected graph \( G = (V, E) \), find a graph \( G' = (V', E') \) such that:
- \( E' \) is a subset of \( E \)
- \( |E'| = |V| - 1 \)
- \( G' \) is connected
- \( G' \) is minimal
\[
\sum_{(u, v) \in E'} c_{uv}
\]
Applications: wiring a house, power grids, Internet connections

Find the MST

Two Different Approaches
Prim’s Algorithm
Almost identical to Dijkstra’s
Kruskal’s Algorithm
Completely different!

Prim’s Algorithm for MST
A node-based greedy algorithm
Builds MST by greedily adding nodes
1. Select a node to be the “root”
   - mark it as known
   - update cost of all its neighbors
2. While there are unknown nodes left in the graph
   a. Select an unknown node \( b \) with the smallest cost from some known node \( a \)
   b. Mark \( b \) as known
   c. Add \((a, b)\) to MST
   d. Update cost of all nodes adjacent to \( b \)

Find MST using Prim’s

<table>
<thead>
<tr>
<th>V</th>
<th>Kwn</th>
<th>Distance</th>
<th>path</th>
</tr>
</thead>
<tbody>
<tr>
<td>v1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>v2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>v3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>v4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>v5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>v6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>v7</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

OrderDeclaredKnown: \( V_1 \)
Prim’s Algorithm Analysis

Running time:
- Same as Dijkstra’s: $O(|E| \log |V|)$

Correctness:
- Proof is similar to Dijkstra’s

Kruskal’s Algorithm for MST

An edge-based greedy algorithm
- Builds MST by greedily adding edges

1. Initialize with
   - empty MST
   - all vertices marked unconnected
   - all edges unmarked
2. While there are still unmarked edges
   a. Pick the lowest cost edge $(u, v)$ and mark it
   b. If $u$ and $v$ are not already connected, add $(u, v)$ to the MST and mark $u$ and $v$ as connected to each other

Doesn’t it sound familiar?

Kruskal’s MST Algorithm

Idea: Grow a forest out of edges that do not create a cycle. Pick an edge with the smallest weight.

$G=(V,E)$

Kruskal code

```cpp
void Graph::kruskal(){
    int edgesAccepted = 0;
    DisjSet s(NUM_VERTICES);
    while (edgesAccepted < NUM_VERTICES - 1){
        e = smallest weight edge not deleted yet;
        // edge e = (u, v)
        uset = s.find(u);
        vset = s.find(v);
        if (uset != vset){
            edgesAccepted++;
            s.unionSets(uset, vset);
        }
    }
}
```

2|E| finds $|V|$ unions $|E|$ heap ops

Find MST using Kruskal’s

- Now find the MST using Prim’s method.
- Under what conditions will these methods give the same result?