Today’s Outline

• Announcements
  – HW #6-7
  • Partner Selection due Thurs May 29 (last night)
  • Assignment due Thurs June 5th.

• Graphs
  – Dijkstra’s (Solves the SSSP problem)
  – Minimum Spanning Trees (MSTs)

Dijkstra’s 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

Dijkstra’s Algorithm

Some Similarities:

The Trouble with Negative Weight Cycles

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
- $\sum c_{uv}$ is minimal

Applications: wiring a house, power grids, Internet connections

Two Different Approaches
- Prim's Algorithm
  - Idea: Grow a tree by adding an edge from the “known” vertices to the “unknown” vertices. Pick the edge with the smallest weight.

Prim's Algorithm for MST
A node-based greedy algorithm

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 $b$

Prim’s Algorithm

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 $b$

Applications: wiring a house, power grids, Internet connections

Prim’s algorithm
Idea: Grow a tree by adding an edge from the “known” vertices to the “unknown” vertices. Pick the edge with the smallest weight.

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>

Order Declared Known: $V_1$
Prim’s Algorithm Analysis

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

Correctness:
Proof is similar to Dijkstra’s

Kruskal’s 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’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

\[ \text{Doesn’t it sound familiar?} \]

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);
        }
    }
}
```

Kruskal’s Algorithm: Correctness

It clearly generates a spanning tree. Call it \(T_K\).

Suppose \(T_K\) is not minimum:

1. Pick another spanning tree \(T_{\text{min}}\) with lower cost than \(T_K\)
2. Pick the smallest edge \(e_1=(u,v)\) in \(T_K\) that is not in \(T_{\text{min}}\)
3. \(T_{\text{min}}\) already has a path \(p\) in \(T_{\text{min}}\) from \(u\) to \(v\)
   - Adding \(e_1\) to \(T_{\text{min}}\) will create a cycle in \(T_{\text{min}}\)
4. Pick an edge \(e_2\) in \(p\) that Kruskal’s algorithm considered after adding \(e_1\) (must exist: \(u\) and \(v\) unconnected when \(e_1\) considered)
   - \(\text{cost}(e_2) \geq \text{cost}(e_1)\)
   - can replace \(e_2\) with \(e_1\) in \(T_{\text{min}}\) without increasing cost!

Keep doing this until \(T_{\text{min}}\) is identical to \(T_K\).

\(T_K\) must also be minimal – contradiction!