Graphs:
More on Shortest Paths, Plus
Minimum Spanning Trees

CSE 373
Data Structures and Algorithms

Correctness: The Cloud Proof

Next shortest path from inside the known cloud
Better path to V? No!

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

At each step:
1) Pick vertex from queue
2) Add it to visited vertices
3) Update queue with neighbors

Breadth-first Search

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
- is minimal

$\sum_{(u,v) \in E'} c_{uv}$

Applications: wiring a house, power grids, Internet connections

$G'$ is a minimum spanning tree.
Find the MST

Two Different Approaches

Prim’s Algorithm
Almost identical to Dijkstra’s

Kruskals’s Algorithm
Completely different!

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
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 with the smallest cost from some known node
   b. Mark b as known
   c. Add (a, b) to MST
   d. Update cost of all nodes adjacent to b

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

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);
        }
    }
}
```

Find MST using Kruskal’s

| A | 2 |
| B | 3 |
| C | 4 |
| D | 5 |
| E | 6 |
| F | 7 |
| G | 8 |
| H | 9 |

Total Cost:

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