Data Structures and Algorithms
Spanning Trees

Chris Brooks

Department of Computer Science
University of San Francisco
Given a connected, undirected graph $G$

- A *subgraph* of $G$ contains a subset of the vertices and edges in $G$
- A *Spanning Tree* $T$ of $G$ is:
  - subgraph of $G$
  - contains all vertices in $G$
  - connected
  - acyclic
21-1: Spanning Tree Examples

Graph

```
0 ---- 3 ---- 1
 |
|
2 3 4
 |
|
5 ---- 6
```
21-2: Spanning Tree Examples

Spanning Tree

0 -- 1

0 -- 2 -- 3

1 -- 4

2 -- 3

5 -- 6
21-3: Spanning Tree Examples

6 Graph

- Graph

Department of Computer Science — University of San Francisco — p.5/27
Spanning Tree Examples

Spanning Tree

0 1
2 3 4
5 6

Department of Computer Science — University of San Francisco — p.6/27
21-5: **Minimal Cost Spanning Tree**

- Minimal Cost Spanning Tree
  - Given a weighted, undirected graph \( G \)
  - Spanning tree of \( G \) which minimizes the sum of all weights on edges of spanning tree
21-6: MST Example
Can there be more than one minimal cost spanning tree for a particular graph?
Can there be more than one minimal cost spanning tree for a particular graph?

YES!

What happens when all edges have unit cost?
Can there be more than one minimal cost spanning tree for a particular graph?

YES!

- What happens when all edges have unit cost?
- All spanning trees are MSTs
Two algorithms to calculate MST:

- **Kruskal’s Algorithm**
  - Build a “forest” of spanning trees
  - Combine into one tree

- **Prims Algorithm**
  - Grow a single tree out from a start vertex
21-12: Kruskal’s Algorithm

1. Start with an empty graph (no edges)
2. Sort the edges by cost
3. For each edge, add to graph if it would not cause a cycle.
21-13: Kruskal’s Algorithm Examples
Proof (by contradiction)

Assume that no optimal MST $T$ contains the minimum cost edge $e$

Add $e$ to $T$, which causes a cycle

Remove an edge other than $e$ to break the cycle

$\text{cost } T' \leq T$, a contradiction
Coding Kruskal’s Algorithm:
- Place all edges into a list
- Sort list of edges by cost
- For each edge in the list
  - Select the edge if it does not form a cycle with previously selected edges
  - How can we do this?
21-16: Kruskal’s Algorithm

Determining of adding an edge will cause a cycle

- Start with a forest of $V$ trees (each containing one node)
- Each added edge merges two trees into one tree
- An edge causes a cycle if both vertices are in the same tree
  - (examples)
21-17: Kruskal’s Algorithm

We need to:

△ Put each vertex in its own tree
△ Given any two vertices $v_1$ and $v_2$, determine if they are in the same tree
△ Given any two vertices $v_1$ and $v_2$, merge the tree containing $v_1$ and the tree containing $v_2$

... sound familiar?
Disjoint sets!

Create a list of all edges

Sort list of edges

For each edge \( e = (v_1, v_2) \) in the list

\( \triangledown \) if \( \text{FIND}(v_1) \neq \text{FIND}(v_2) \)

- Add \( e \) to spanning tree
- \( \text{UNION}(v_1, v_2) \)
21-19: Prim’s Algorithm

6 Grow that spanning tree out from an initial vertex

6 Divide the graph into two sets of vertices
   △ vertices in the spanning tree
   △ vertices not in the spanning tree

6 Initially, Start vertex is in the spanning tree, all other vertices are not in the tree
   △ Pick the initial vertex arbitrarily
While there are vertices not in the spanning tree
   - Add the cheapest vertex to the spanning tree
21-21: Prims’s Algorithm Examples
21-22: Prim’s Algorithm

- Use a table – much like Dijkstra table
- Path has the same meaning
- Cost is for vertex \( v_k \)
  - cost to add \( v_k \) to the tree
  - (instead of length of path to \( v_k \) )
21-23: Prim’s Algorithm

- Code for Prim’s algorithm is very similar to the code for Dijkstra’s algorithm
- Make *one small change* to Dijkstra’s algorithm to get Prim’s algorithm
void Dijkstra(Edge G[], int s, tableEntry T[]) {
    int i, v;
    Edge e;
    for(i=0; i<G.length; i++) {
        T[i].distance = Integer.MAX_VALUE;
        T[i].path = -1;
        T[i].known = false;
    }
    T[s].distance = 0;
    for (i=0; i < G.length; i++) {
        v = minUnknownVertex(T);
        T[v].known = true;
        for (e = G[v]; e != null; e = e.next) {
            if (T[e.neighbor].distance >
                T[v].distance + e.cost) {
                T[e.neighbor].distance = T[v].distance + e.cost;
                T[e.neighbor].path = v;
            }
        }
    }
}
void Dijkstra(Edge G[], int s, tableEntry T[]) {
    int i, v;
    Edge e;
    for(i=0; i<G.length; i++) {
        T[i].distance = Integer.MAX_VALUE;
        T[i].path = -1;
        T[i].known = false;
    }
    T[s].distance = 0;
    for (i=0; i < G.length; i++) {
        v = minUnknownVertex(T);
        T[v].known = true;
        for (e = G[v]; e != null; e = e.next) {
            if (T[e.neighbor].distance > e.cost) {
                T[e.neighbor].distance = e.cost;
                T[e.neighbor].path = v;
            }
        }
    }
}