18-0: **Spanning Trees**

- 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

18-1: **Spanning Tree Examples**

- Graph

![Graph](image1)

18-2: **Spanning Tree Examples**

- Spanning Tree

![Spanning Tree](image2)

18-3: **Spanning Tree Examples**

- Graph

![Graph](image3)
18-4: Spanning Tree Examples

- Spanning Tree

18-5: 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

18-6: MST Example

18-7: MST Example
18-8: **Minimal Cost Spanning Trees**

- Can there be more than one minimal cost spanning tree for a particular graph?

18-9: **Minimal Cost Spanning Trees**

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

18-10: **Minimal Cost Spanning Trees**

- 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

18-11: **Calculating MST**

- 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

18-12: **Kruskal’s Algorithm**

- Start with an empty graph (no edges)
- Sort the edges by cost
- For each edge $e$ (in increasing order of cost)
  - Add $e$ to $G$ if it would not cause a cycle
18-13: **Kruskal’s Algorithm Examples**

![Graph Example]

18-14: **Kruskal’s Algorithm**

- 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

18-15: **Kruskal’s Algorithm**

- 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?

18-16: **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)

18-17: **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?

18-18: **Kruskal’s Algorithm**

- Disjoint sets!
- Create a list of all edges
- Sort list of edges
- For each edge $e = (v_1, v_2)$ in the list
  - if $\text{FIND}(v_1) \neq \text{FIND}(v_2)$
    - Add $e$ to spanning tree
    - $\text{UNION}(v_1, v_2)$

18-19: **Prim’s Algorithm**

- Grow that spanning tree out from an initial vertex
- Divide the graph into two sets of vertices
  - vertices in the spanning tree
  - vertices not in the spanning tree
- Initially, Start vertex is in the spanning tree, all other vertices are not in the tree
  - Pick the initial vertex arbitrarily

18-20: **Prim’s Algorithm**

- While there are vertices not in the spanning tree
  - Add the cheapest vertex to the spanning tree

18-21: **Prim’s Algorithm Examples**

```
0      2
    1
4  1  3  10
  2
2  2  3  7
  7
5  8  4
  6
5  1
  6
```

18-22: **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 $v_k$)

18-23: 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

18-24: Dijkstra Code

```c
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].known = false;
        T[i].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;
                }
            }
        }
    }
}
```

18-25: Prim Code

```c
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[i].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;
                }
            }
        }
    }
}
```