Class 25

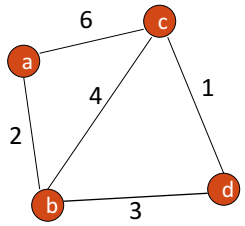
MINIMUM SPANNING TREE PROBLEM

PRIM'S ALGORITHM

KRUSKAL'S ALGORITHM

Spanning Tree

Consider the following graph:



A *spanning tree* of a connected graph *G* is a a connected acyclic subgraph of *G* that includes all of *G*'s vertices.

It takes on the form of a tree.

The above graph has several spanning trees, including:

{ab, ac, cd}

{ac, bc, cd}

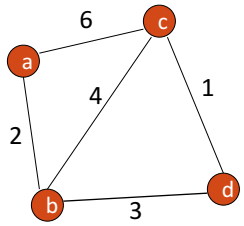
Minimum Spanning Tree

A *minimum spanning tree* of a weighted, connected graph *G* is a spanning tree of minimum total weight.

If there is a spanning tree, then there is a minimum spanning tree.

For the graph from the prior slide:

{ab, bd, cd} [value 2 + 3 + 1 = 6]



Small group exercise:

How could we find a minimum spanning tree?

Prim's Algorithm

Grow a tree.

Pick a node.

Pick the lowest cost edge out of that node, and connect those two nodes.

From now on, pick the lowest cost edge of all the nodes already forming the tree, ensuring that adding that node does not create a cycle.

Repeat until all nodes are included.

Animation: https://en.wikipedia.org/wiki/File:PrimAlgDemo.gif

Explanation: https://en.wikipedia.org/wiki/File:Prim%27s_algorithm.svg

Animation with a larger graph: https://visualgo.net/en/mst

Kruskal's Algorithm

Grow forests.

Sort the edges from lowest to highest edge cost.

Create a forest in which each node forms a tree.

Pick the lowest cost edge and join the two trees, if that does not cause a cycle.

Continue until all trees are merged into a single tree.

Animation: https://en.wikipedia.org/wiki/File:KruskalDemo.gif

Explanation: https://en.wikipedia.org/wiki/Kruskal%27s_algorithm#Example

Animation with a larger graph: https://visualgo.net/en/mst

Proof of Optimality: Warm-up

Suppose the graph has just one node.

That node forms an MST

Suppose the graph has just two nodes (with one edge)

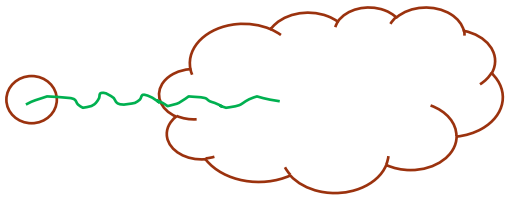
The graph forms an MST

Proof of Optimality: Warm-up

Suppose the graph has many nodes.

Suppose all nodes except for one have been formed into an MST.

How would you connect the remaining node?



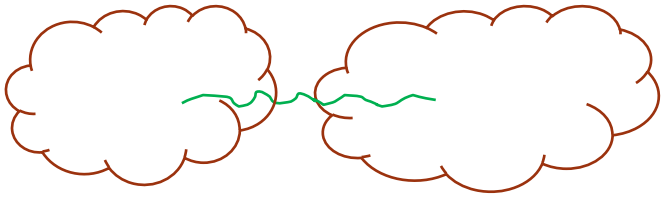
Presumably with the lowest cost edge.

Proof of Optimality: Warm-up

Suppose the graph has many nodes.

Suppose about half the nodes have been formed into MSTs each.

How would you connect the tow halves?



Again, presumably with the lowest cost edge.

Proof of Optimality

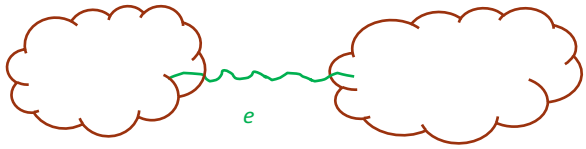
Proof by contradiction.

We know there is an MST for the graph G.

For sake of argument, suppose there is only one MST.

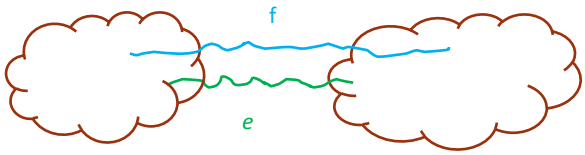
Let *T* be that MST.

Let *e* be the lowest cost edge that is used to form the MST, but *e* is not on *T*



Proof of Optimality

Let *f* be an edge that connects the two parts and is on *T*.



If we add *f* to the MST we are building, then the MST forms a cycle.

Let's remove *e*.

Now we have a spanning tree that has a higher cost than the one with *e*.

A contradiction.

Prim's Algorithm in detail

ALGORITHM *Prim*(G)

//Prim's algorithm for constructing a minimum spanning tree
//Input: A weighted connected graph $G = \langle V, E \rangle$
//Output: E_T, the set of edges composing a minimum spanning tree of G
$V_T \leftarrow \{v_0\}$ //the set of tree vertices can be initialized with any vertex
$E_T \leftarrow \varnothing$
for $i \leftarrow 1$ **to** $|V| - 1$ **do**
    find a minimum-weight edge $e^* = (v^*, u^*)$ among all the edges (v, u)
    such that v is in V_T and u is in $V - V_T$
    $V_T \leftarrow V_T \cup \{u^*\}$
    $E_T \leftarrow E_T \cup \{e^*\}$
return E_T

Prim's Algorithm

Small team time: Use data structures that ensure Prim's algorithm is maximally efficient.

Analyze the computational complexity of Prim's algorithm.

Kruskal's Details and Analysis

ALGORITHM *Kruskal(G)*

```
//Kruskal's algorithm for constructing a minimum spanning tree
//Input: A weighted connected graph G = ⟨V, E⟩
//Output: E_T, the set of edges composing a minimum spanning tree of G
sort E in nondecreasing order of the edge weights w(e_i1) ≤ ··· ≤ w(e_i|E|)
E_T ← ∅;  ecounter ← 0        //initialize the set of tree edges and its size
k ← 0                         //initialize the number of processed edges
while ecounter < |V| − 1 do
    k ← k + 1
    if E_T ∪ {e_ik} is acyclic
        E_T ← E_T ∪ {e_ik};   ecounter ← ecounter + 1
return E_T
```

Kruskal's Details and Analysis

Small team time: Use data structures that ensure Kruskal's algorithm is maximally efficient.

Analyze the computational complexity of Kruskal's algorithm.