# Data Structures
# Intro To Minimum Spanning Trees

Cain Susko

Queen's University
School of Computing

March 30, 2022

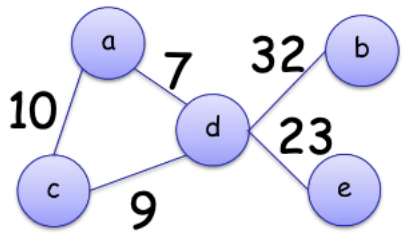
# Spanning Trees

A spanning Tree is, given the graph $G$, a sub-graph of $G$: $T$ which contains all vertices within $G$. It is known that every connected graph (there is a path from any node to any other node) has at least one spanning tree.

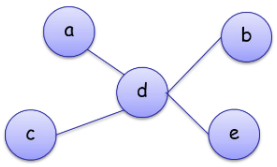
**Tree Generation** There are a few ways to convert a (connected) graph into a spanning tree. These include Depth First and Breadth First search which are both ways of creating a tree by traversing it.

# Minimum Spanning Tree

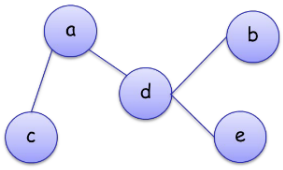
The idea of a minimum spanning tree is, given a graph with weighted edges, what is the lowest 'cost' (sum of weights) for a spanning tree. For example, given the graph:




If we were to create a few spanning trees we could then find the minimum one:




Tree2 w= 71




Tree3 w= 72

And thus, tree 2 is the MST for the given graph. Note: there is another tree not shown which has a weight of 74.

### Definition

A minimum Spanning Tree from an undirected, connected, weighted graph is a spanning tree of minimum weight. There may be more than one MST for a given graph.

**Applications** The concept of the MST is intriguing for the computer scientist as it allows for the association of data with the least amount of memory required for maintaining the relationships. The applications can include minimizing communication networks to optimizing transit networks.

## Kruskal's Algorithm

A basic algorithm for growing an MST from a graph is as follows: given a graph $G$,

```
def Kruskal(G) {
        S=[]
        while |S| < n-1{
                Let (u,v) be the minimum weighted edge in G
                        such that S.append((u,v)) has no cycles

                S.append((u,v))
        }
        return S
}
```

Which is to say, one adds edges in increasing weight, skipping those who's addition would create a cycle, thus making it *not* a tree.

**Checking For Cycles** by intuition, if one is to connect 2 vertices which are already in the same graph, this creates a cycle. In order to avoid a vertex being added which is already in the tree we must give each vertex a label and keep track of such labels using the Union-Find data structure.

### Union-Find

This is a type of data structure which uses the tree ADT. The root of the tree is a representative item of the entire tree. Each node below the root

points back to its parent; the root points to itself.
There are 3 operations for Union-Find:

- `Create-Set(u)` - create a set containing the single element `u`
- `Find-Set(u)` - find the set containing the element `u`
- `Union(u,v)` - merge the sets respectively containing `u` and `v` into a common set `a`

Thus, Using this data-structure, Kruskal's algorithm is the following:

```
^^ Sorted edgeList By Weight ^^

A = []

for each u in V{
        create-set(u)
}

for (u,v) in edgeList{

        if (find-set(u) != find-set(v)){
                A.append((u,v))

                Union(u,v)
        }
}
```

**Implementation** For each of the tree operations, the general implementation is the following:

- create: `u.parent = u`
- find:

```
temp = u.parent
while temp.parent != temp{
        temp = temp.parent
}
return temp
```

- union:

```
a = find-set(x)
b = find-set(y)

if (a.height <= b.height){
        if (a.height == b.height){
                b.height++
        }
        a.parent = b
} else {
        b.parent = a
}
```

The next note goes over a different algorithm for MST generation.