Graph Algorithms: MST

Joël Mathys

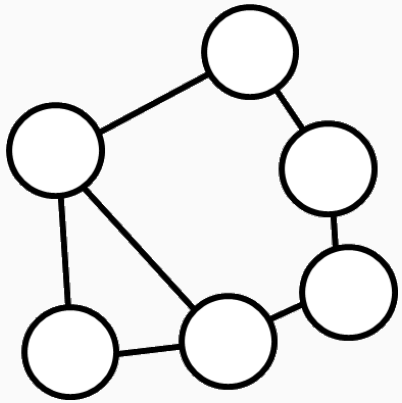
14. February 2017

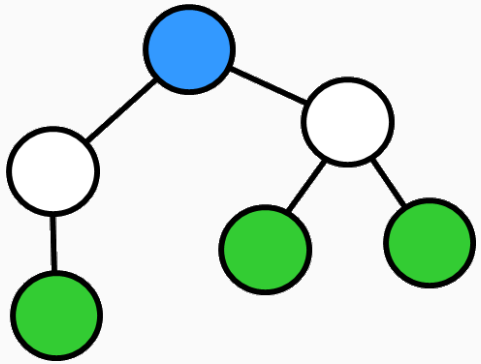
Swiss Olympiad in Informatics

Minimum Spanning Tree

What is a Tree

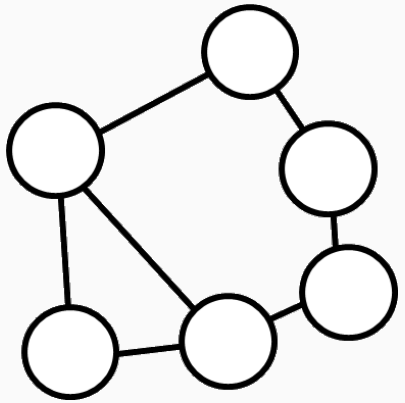
- Graph Type
  - n vertices
  - m edges
  - undirected

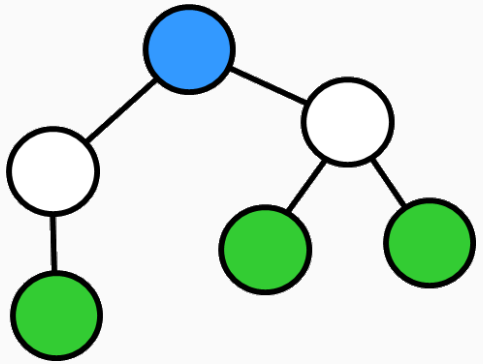




What is a Tree

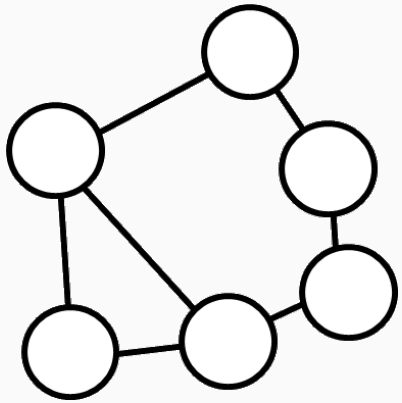
- Graph Type
  - n vertices
  - m edges
  - undirected
- Tree definition
  - connectivity
  - no cycles, unique path

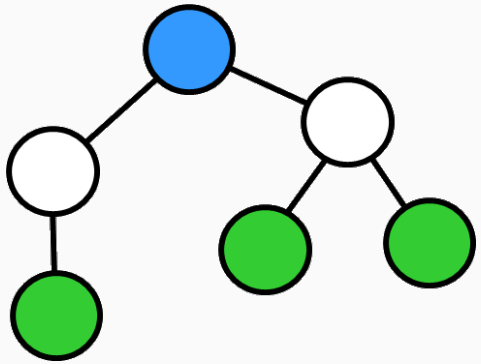




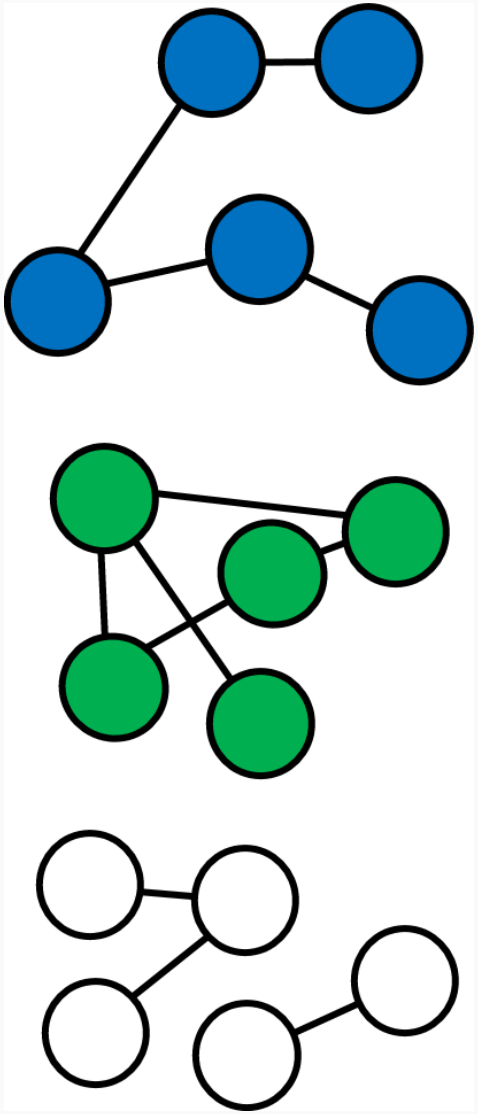
What is a Tree

- Graph Type
  - n vertices
  - m edges
  - undirected
- Tree definition
  - connectivity
  - no cycles, unique path
- Special properties
  - root
  - leaf
  - exactly $n - 1$ edges
  - height of a tree

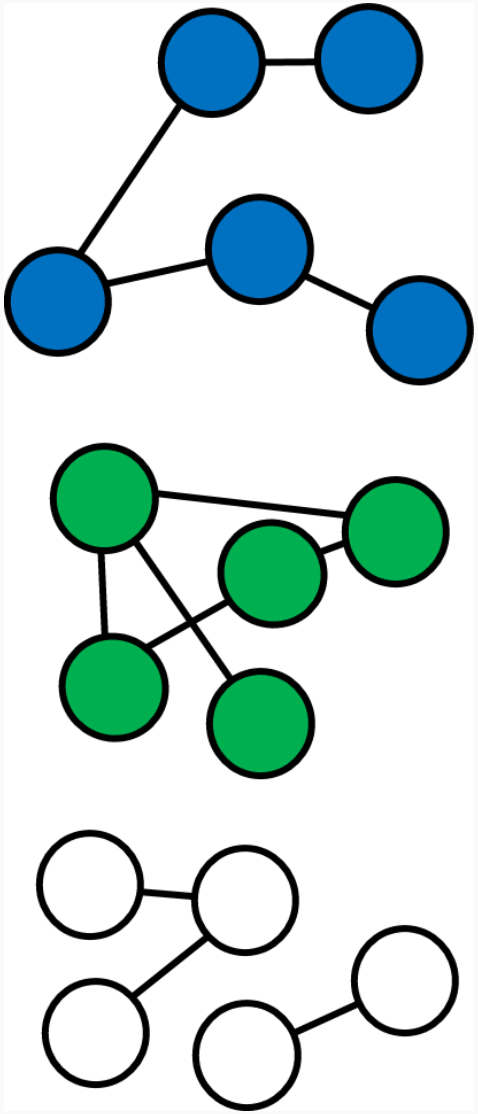




Spot the Trees



Spot the Trees



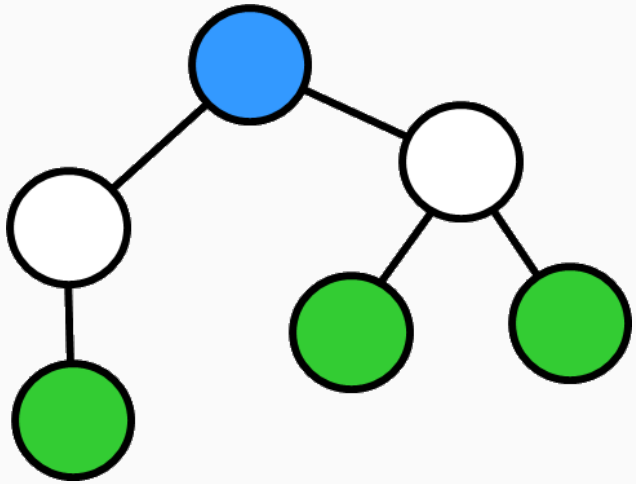
blue = tree

green $\neq$ tree - has a cycle

white $\neq$ tree - is not connected, is a forest (group of trees)

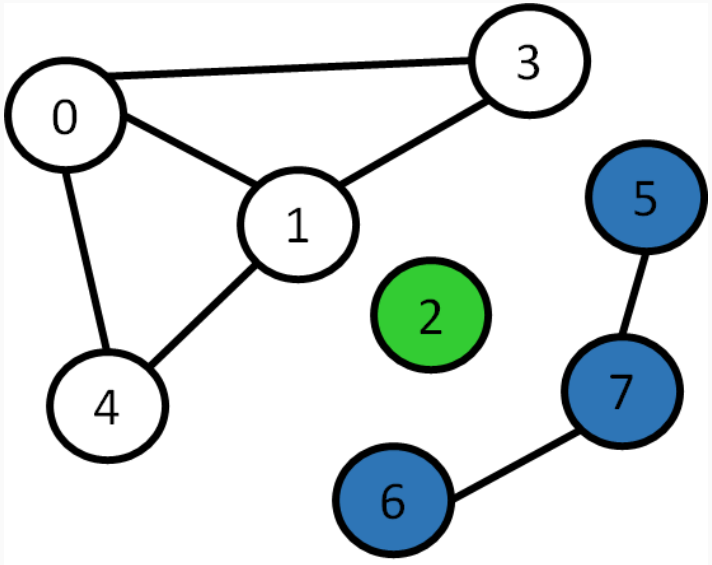
Minimum Spanning Tree

- Weight of a Spanning Tree
  - find a tree in the graph
  - sum of all edges in the tree
- Multiple Trees
  - MST = sum is minimal
- 2 Algorithms
  - Kruskal (union find)
  - Prim (dijkstra mod.)



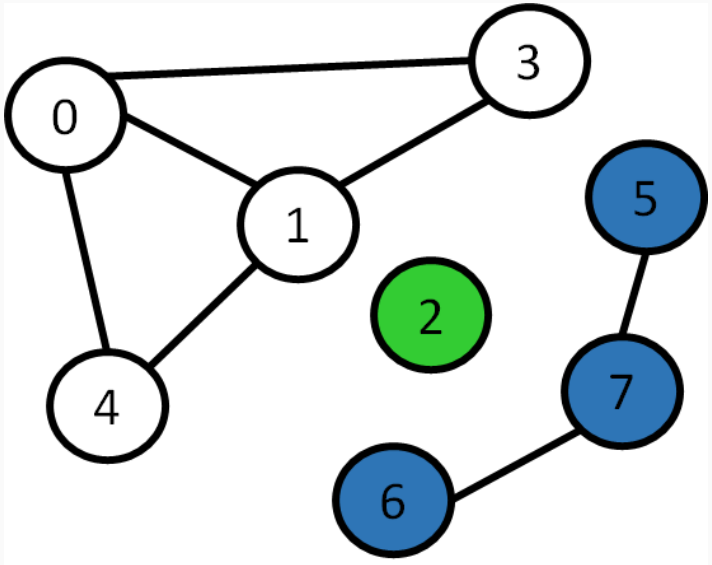
Union Find

- Datastructure
  - create new set
  - merge sets
  - ask if two elements are in the same set
- bfs, dfs
- very EASY



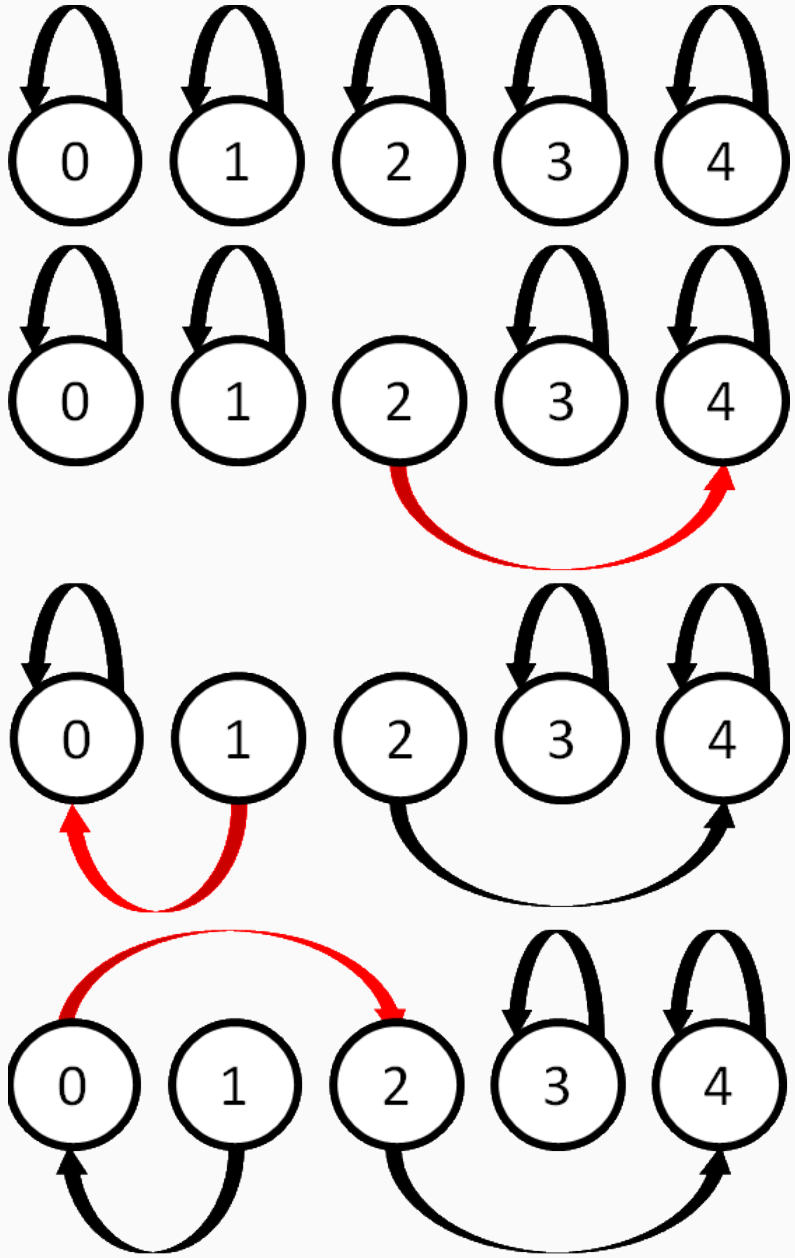
Union Find

- Datastructure
  - create new set
  - merge sets
  - ask if two elements are in the same set
- bfs, dfs
- very EASY
- Friends/Boss Idea
  - every employee has exactly one boss
  - there is a CEO
  - same company - same boss
  - company merge - new boss



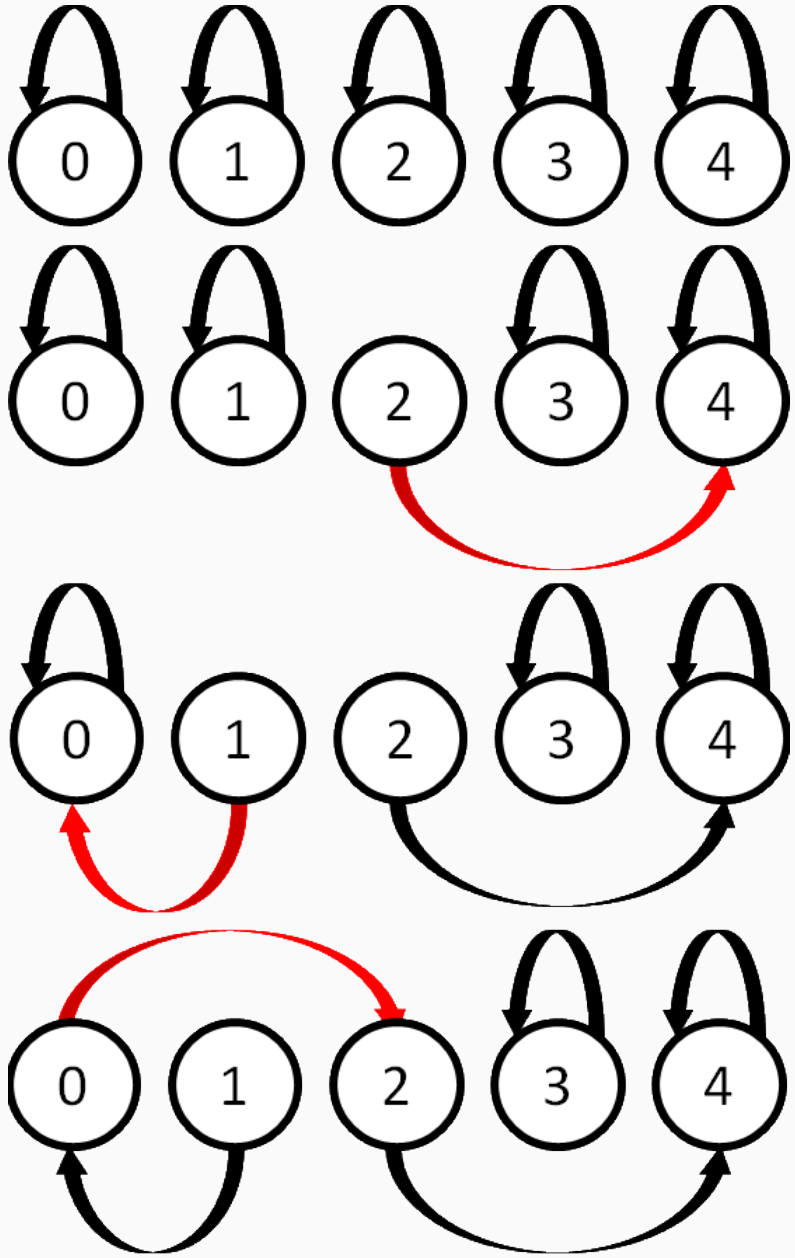
Union Find

- initzialize
  - n companies
  - own boss
- merge = change reference
- long paths!



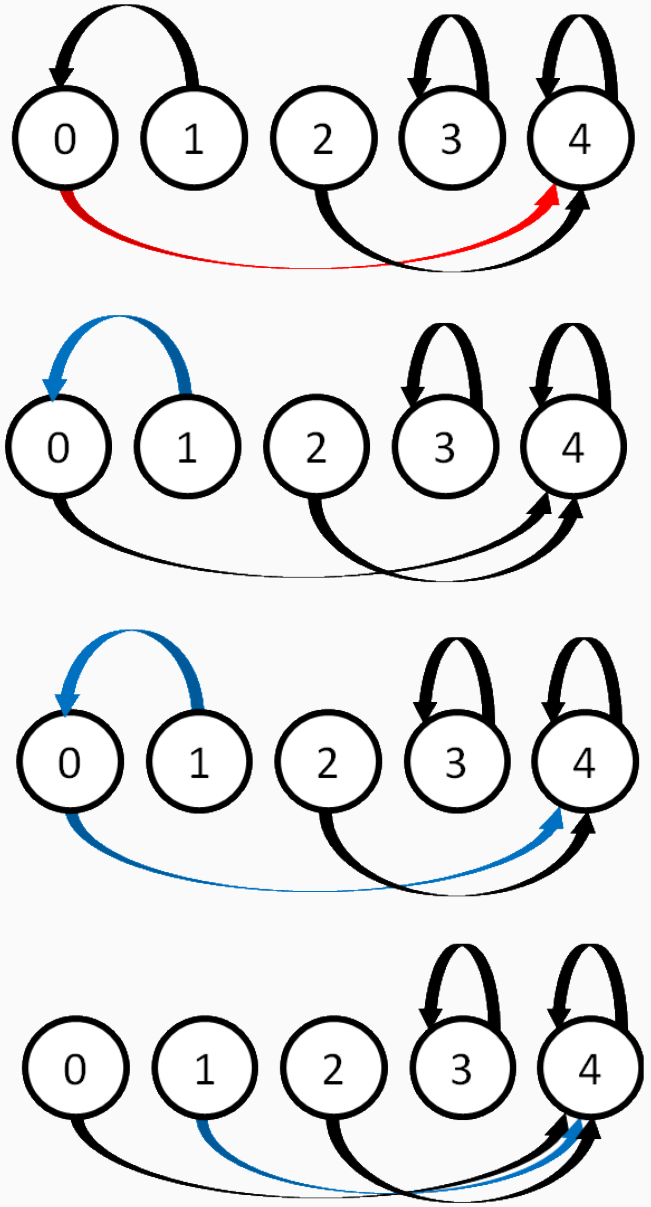
Union Find

- initzialize
  - n companies
  - own boss
- merge = change reference
- long paths!
- optimize
  - path compression
  - randomization
- Efficiency
  - $O(\alpha(n)) \approx O(1)$
- Skript



Union Find

- initzialize
  - n companies
  - own boss
- merge = change reference
- long paths!
- optimize
  - path compression
  - randomization
- Skript



Implementation

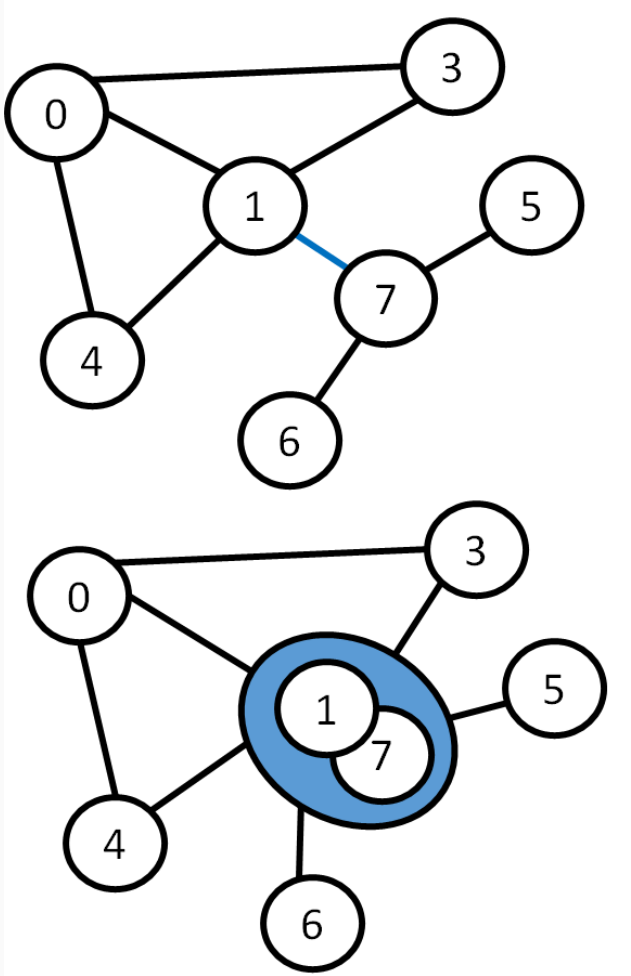
```
vector<int> parent;

int find(int x){
    if(parent[x] == x){
        return x;
    }
    return parent[x] = find(parent[x]);
}

void unite(int a, int b){
    a = find(a);
    b = find(b);
    parent[a] = b;
}
```

Kruskal

- Which edges should be part of the MST?

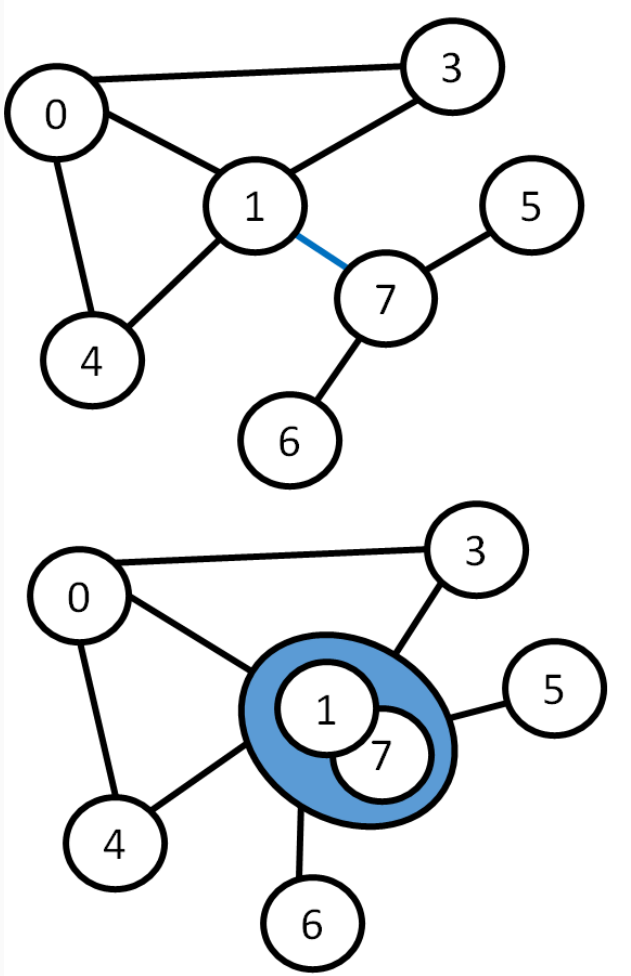
Kruskal

- Which edges should be part of the MST?
- search cheapest edge
- ”merge” vertices
- same problem, repeat again
- ”merge process”
  - check if same component
  - do efficiently with Union Find



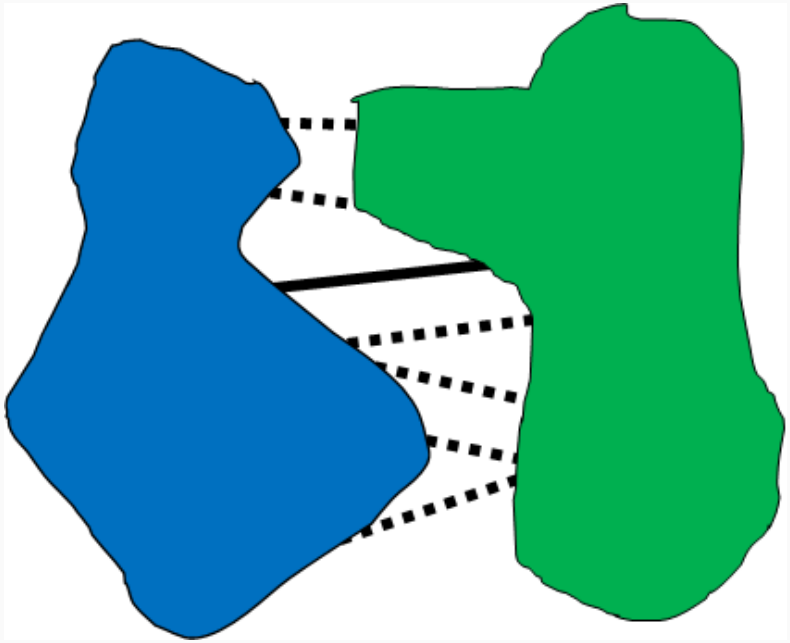
Kruskal

- init Union Find
- Determine cheapest edge
- Sort all edges
- Loop through all edges
- Check if part of the MST
  - yes - unite, add to cost
  - no - do nothing



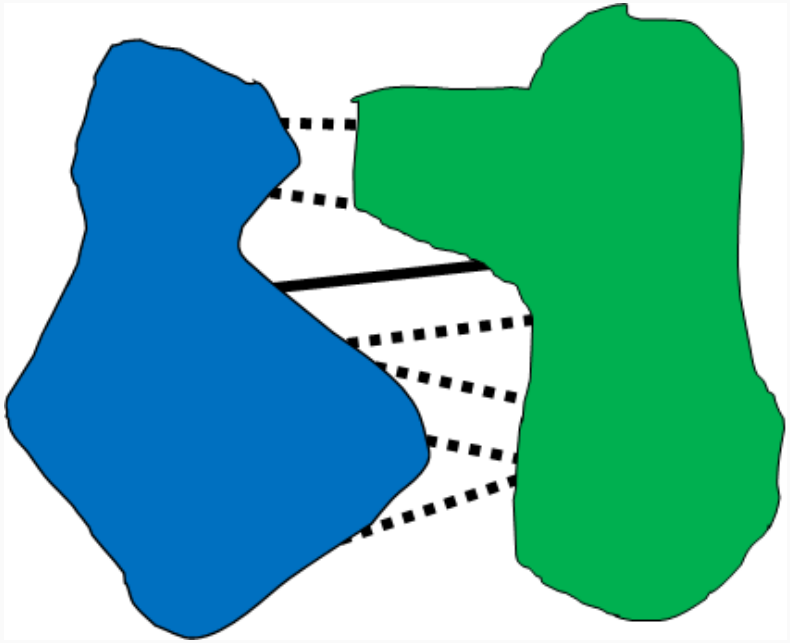
Kruskal Correct

- Why is it correct?
- Imagine optimal solution
- Delete smallest edge
- Two seperate components
- Merge again
  - Look at all edges between
  - which one is the best?
- Efficiency

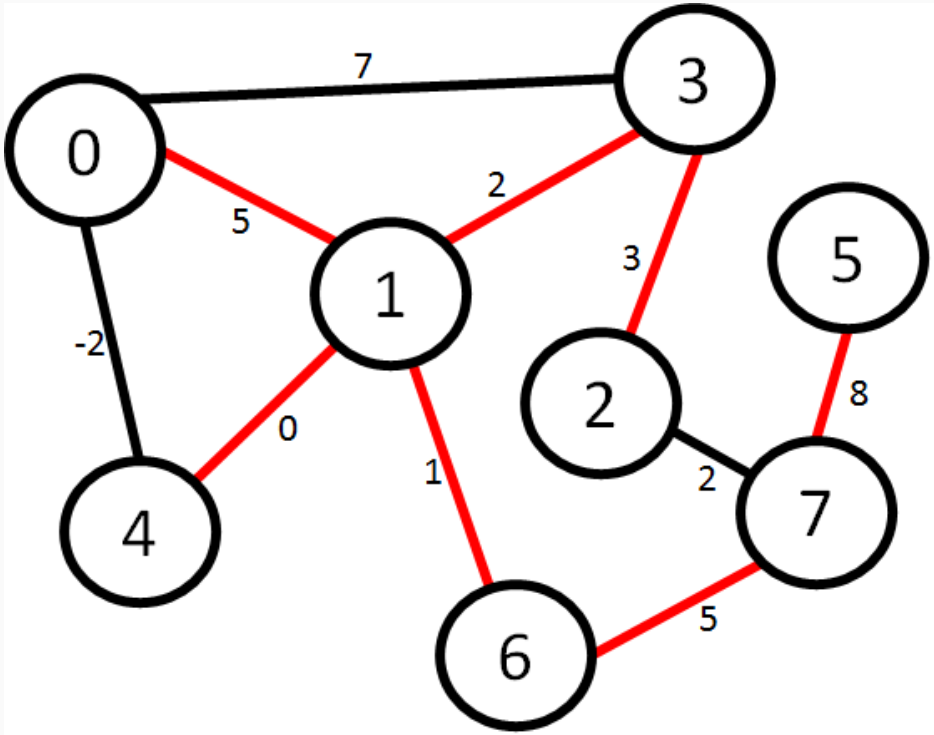


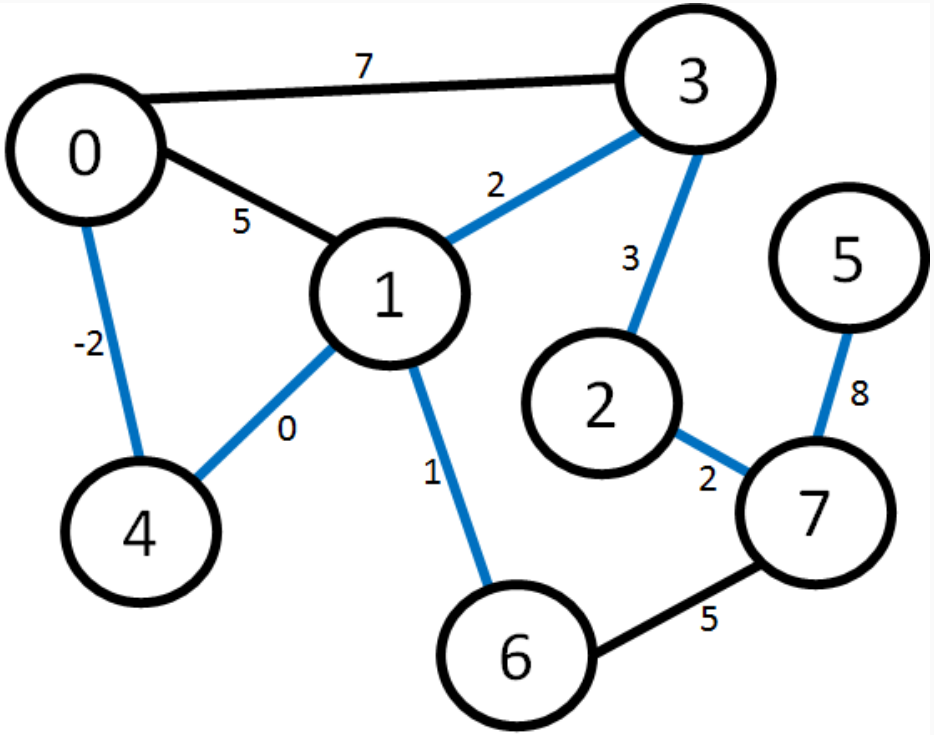
Kruskal Correct

- Why is it correct?
- Imagine optimal solution
- Delete smallest edge
- Two seperate components
- Merge again
  - Look at all edges between
  - which one is the best?
- Efficiency
- $O(m \cdot log(m))$
  - sort
  - union find

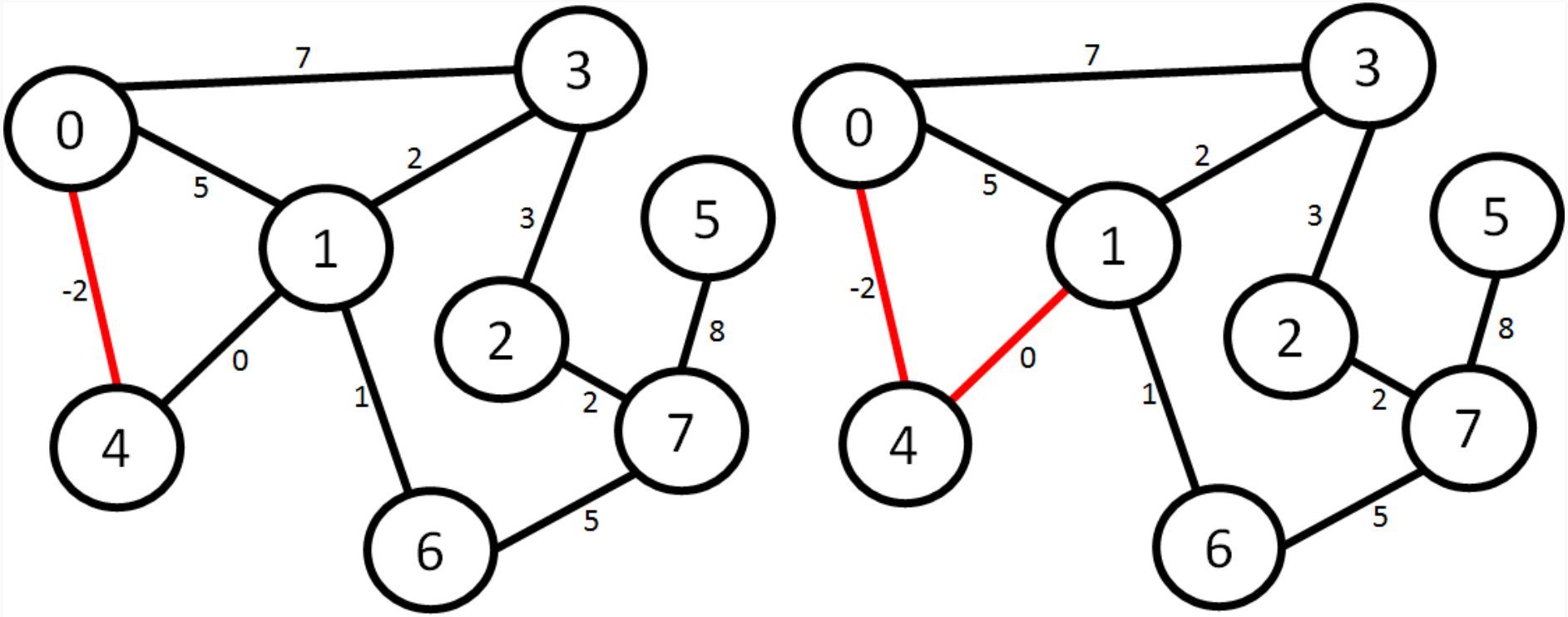


Kruskal Visualization

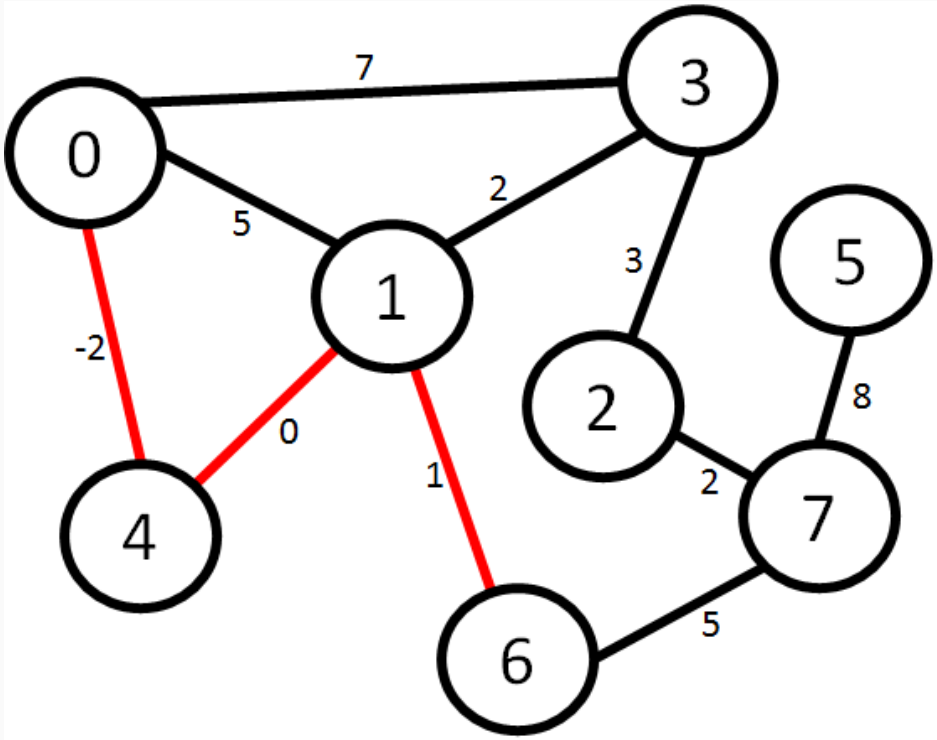


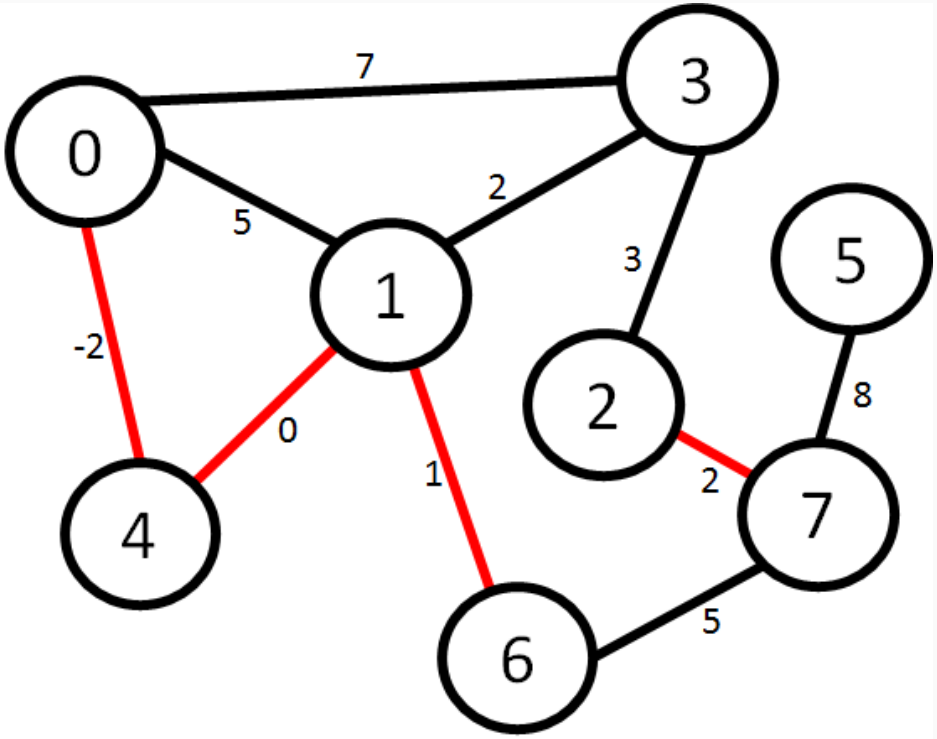


Kruskal Visualization

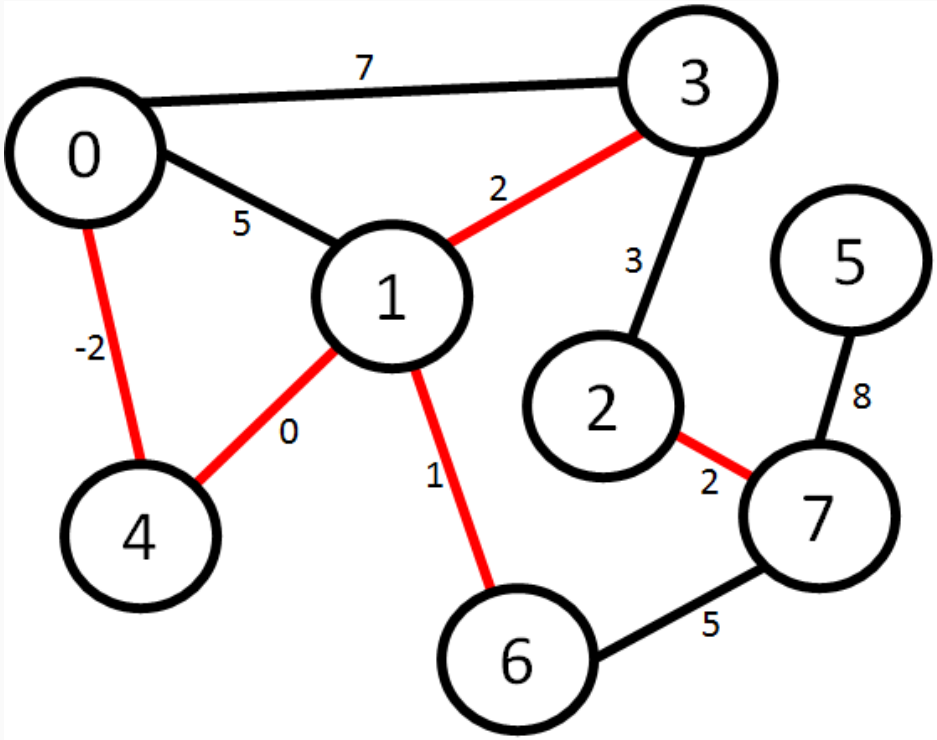


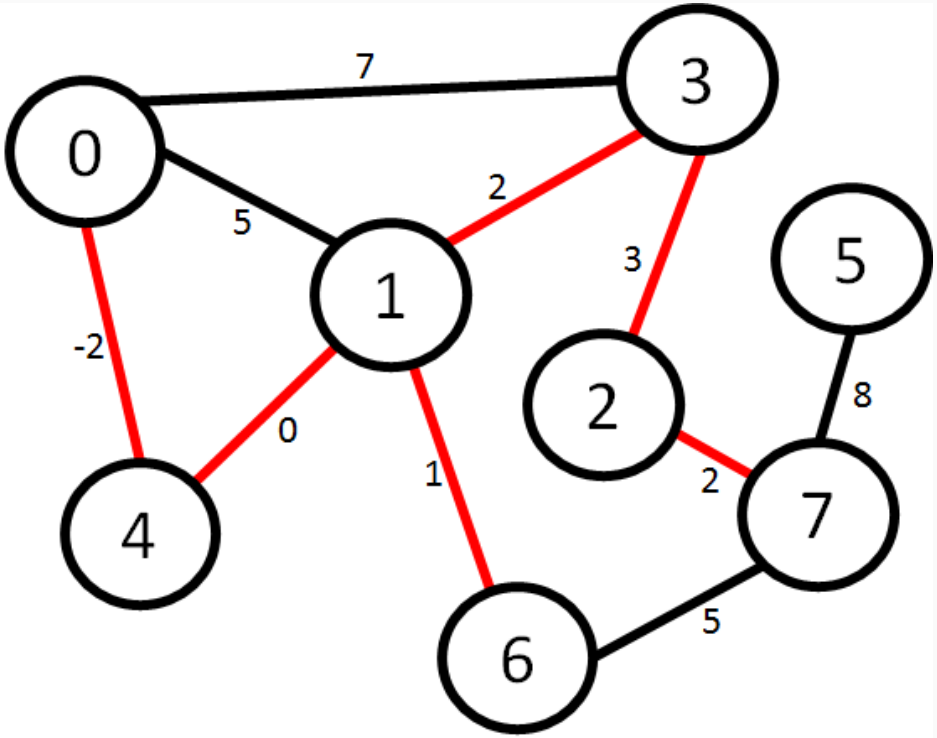
Kruskal Visualization



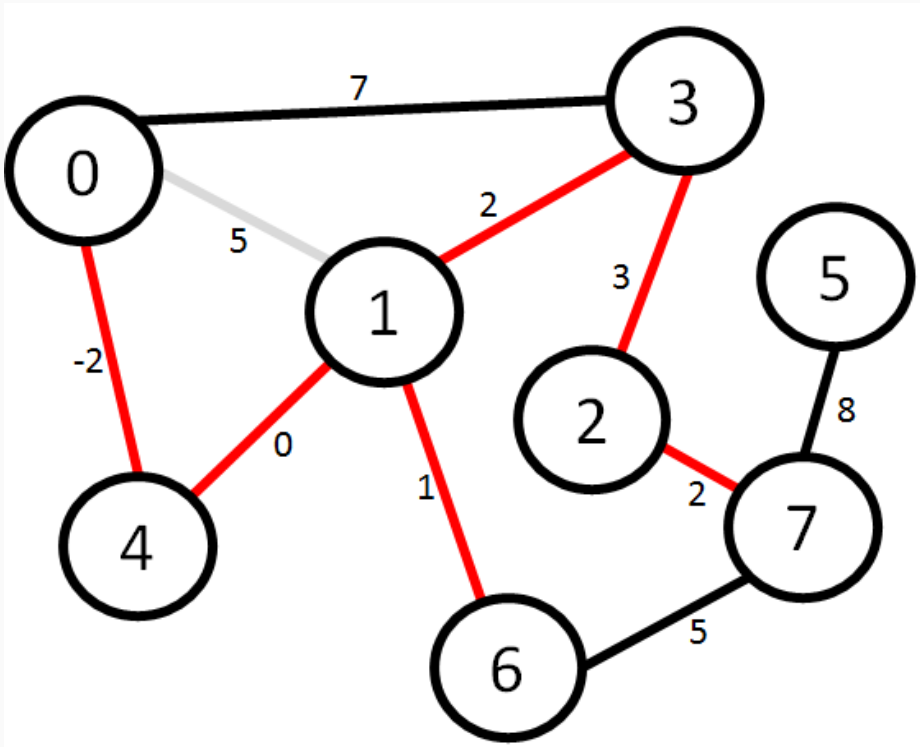


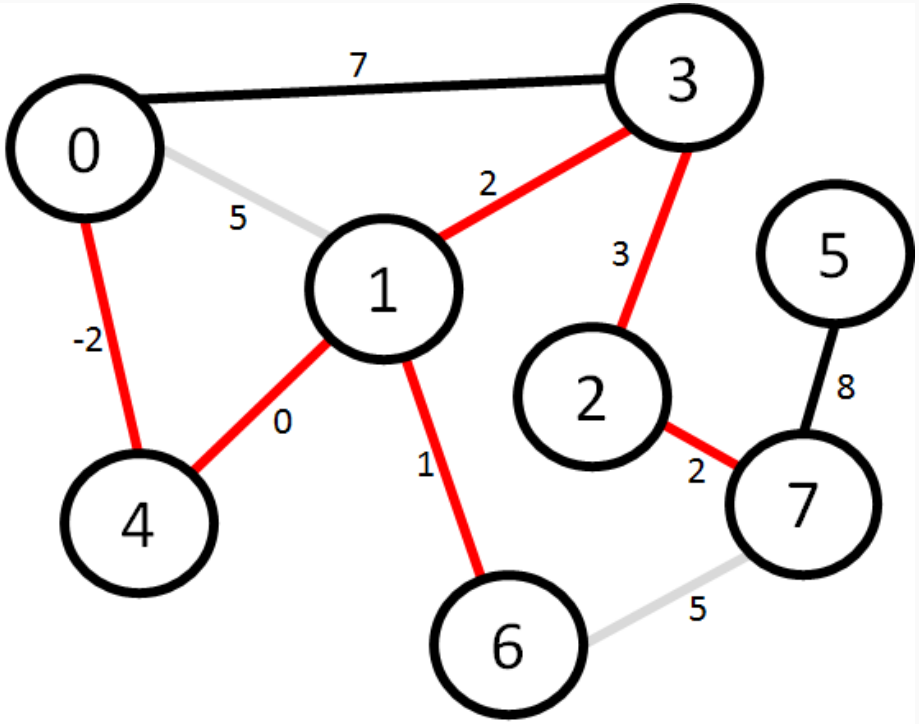
Kruskal Visualization



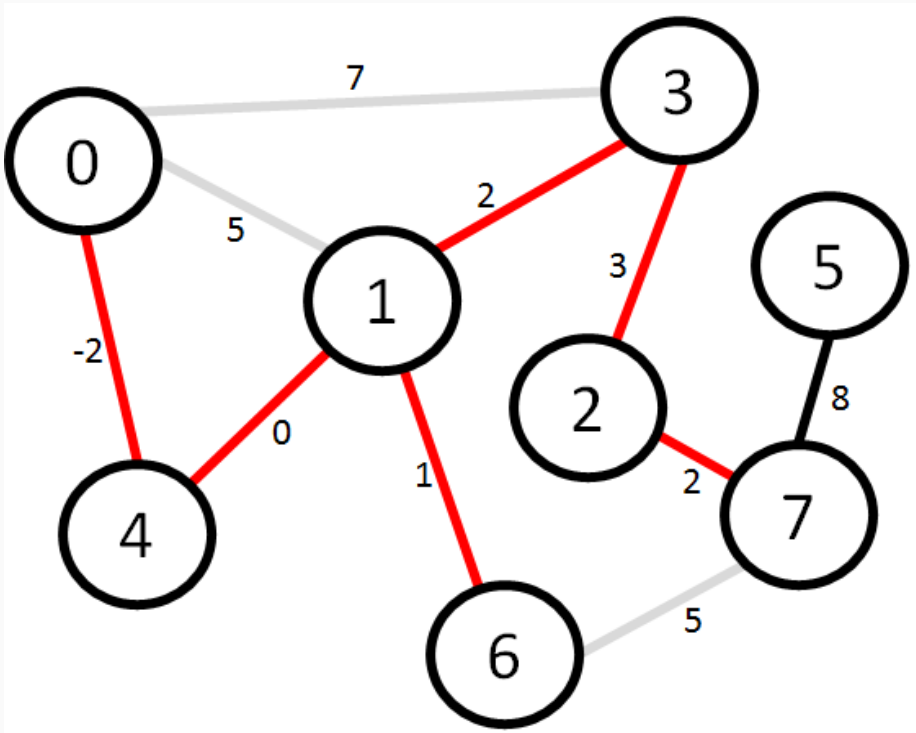


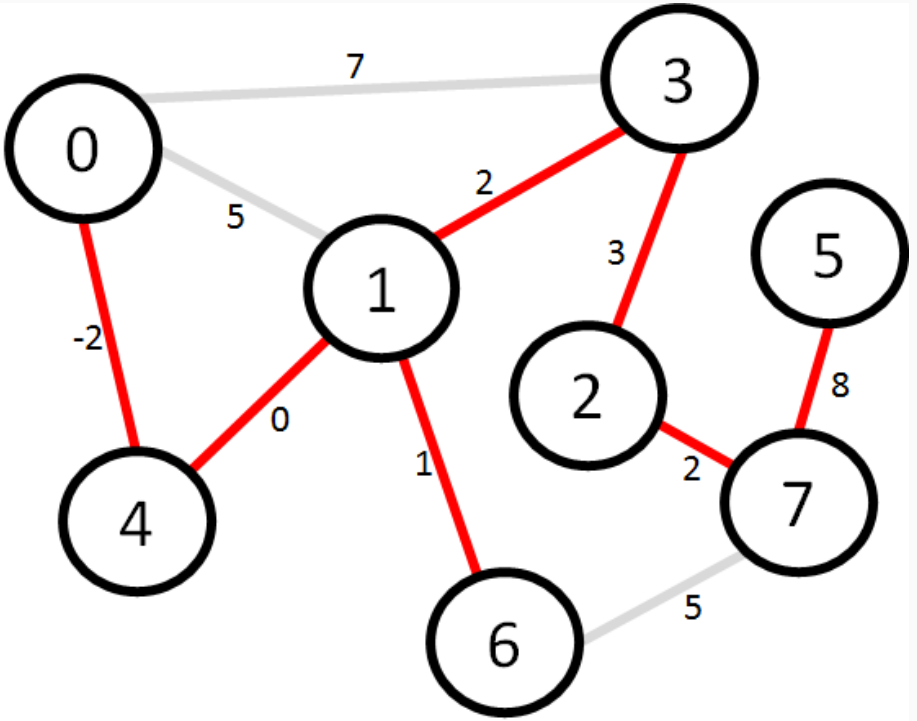
Kruskal Visualization





Kruskal Visualization



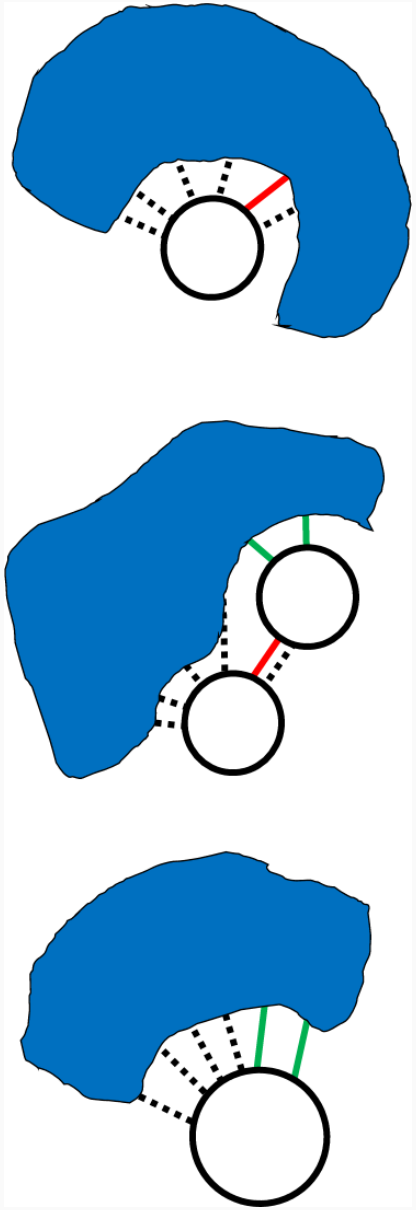


Kruskal Code

```
int kruskal(vector<vector<pair<int,int> > > &graph){
    vector<pair<int,pair<int,int> > > edges;
    int weight = 0;

    for(int i=0;i<(int)graph.size();i++){
        parent.push_back(i);
        for(int j=0;j<(int)graph[i].size();j++){
            int from = i;
            int to = graph[i][j].first;
            int w = graph[i][j].second;
            if(from > to){continue;}
            edges.push_back({w, {from, to}});
        }
    }
    sort(edges.begin(), edges.end());

    for(int i=0; i<(int)edges.size();i++){
        int from = edges[i].second.first;
        int to = edges[i].second.second;
        int w = edges[i].first;

        if(find(to) == find(from)){
            continue;
        }
        unite(to, from);
        weight += w;
    }
    return weight;
}
```

Prim

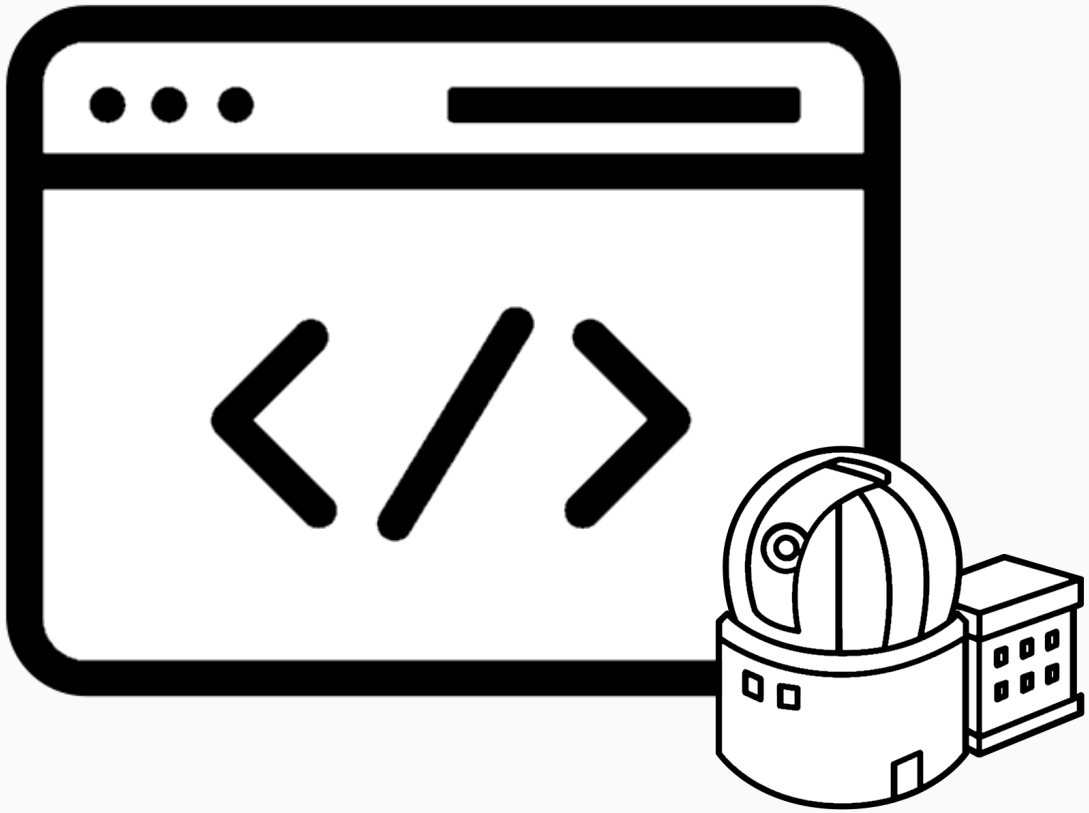
- Dijkstra
  - don't add to total distance
- construct from a single point
- keep track of the costs
- no union find



Additional Resources

- SOI Scripts with Code-Samples
- Union Find
- `https://soi.ch/wiki/union-find_de/`
- Minimum Spanning Tree
- `https://soi.ch/wiki/mst_de/`

Questions



Much of the excitement we get out of our work is that we don't really know what we are doing