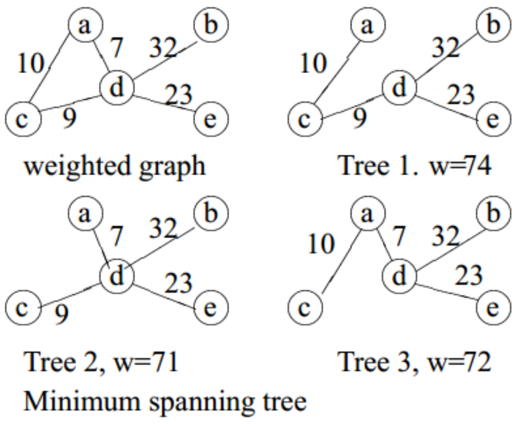
Minimum Spanning Trees

Hariank Muthakana

October 23, 2014

1 Introduction

A *spanning tree* is a subgraph that contains all the vertices of the original graph and is also a tree. While a graph may have several spanning trees, we are generally interested in the *minimum spanning tree*. Given a weighted graph, we aim to find a spanning tree with minimum total edge-weight-sum.



Prim's algorithm and Kruskal's algorithm are two important greedy algorithms used to find minimum spanning trees.

2 Kruskal's Algorithm

Kruskal's is generally the easier of the two to understand. The idea is that we start from an empty set and build up the solution tree one edge at a time. At each step, we want to add an edge that is going to be in the final MST. The key is how we choose which edges to add. The objective when each edge is added is to avoid cycles. If an edge between vertices a and b does not contribute a cycle, then a and b should not already be part of the same tree.

```
S = {}
for all edges in sorted order:
    if edge {a, b} does not contribute a cycle:
    add {a, b} to S
    S.add( (a, b) )
return S
```

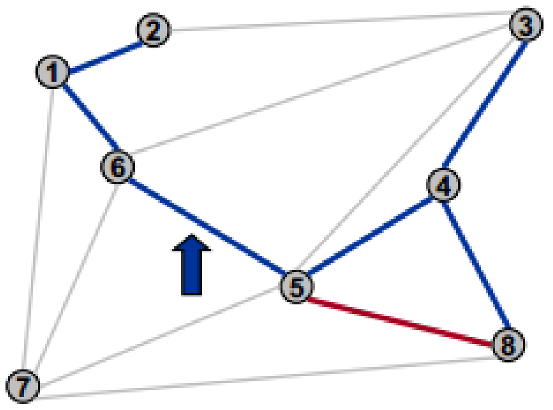


Figure 1: The 5-8 edge incorrectly adds a cycle

Clearly, while the algorithm itself is rather simple, the tricky part is determining if a and b are in the same tree before the edge is added. For this, we can use a *union-find*, or a disjoint set data structure. We can use the $Find()$ operation to check whether a and b are part of the same "set" (in this case, the same tree). The time complexity will therefore be $O(E \log V)$.

3 Prim's Algorithm

Prim's is similar to Kruskal's, except now we build up the tree one vertex at a time. This is also a greedy algorithm. We start with an arbitrary vertex as the vertex. At each step, we add the lightest, or shortest, edge leaving our current tree to our edge set, and its other endpoint to the vertex set.

```
root = arbitrary vertex
S = {root}
while len(S) < n:
    (a, b) = lightest edge connecting S to vertices not in S
    S.add( (a, b) )
```

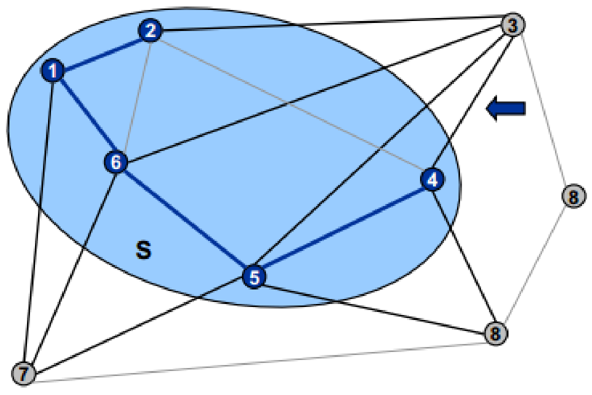


Figure 2: Edge to be added

To find the lightest edge, we can use a priority queue or heap. The key of the priority q is the weight of the lightest edges to each vertex. At each step, we pop a vertex off the queue, update the key values of all its neighbors, and add it to our vertex set. With the priority queue or heap, Prim runs in $O(E \log V)$.

4 Comparisons

Why are both of these algorithms important if they have the same time complexity?. Prim's algorithm and Kruskal's algorithm run differently based on how sparse or dense the graph is. Prim's actual time complexity is $O((E+V)\log V)$, which can be improved to $O(E+V\log V)$ with a Fibonacci heap. So, when you have many more edges than vertices (dense), Prim will perform better. When the graph has few edges (sparse), Kruskal will perform better.

5 Other

- Boruvka's Algorithm
- Phone/Computer Networks
- Euclidean Space
- TSP Heuristic
- Steiner Trees

5.1 References

http://www.cse.ust.hk/~dekai/271/notes/L07/L07.pdf
http://www.cs.princeton.edu/courses/archive/spr02/cs226/lectures/mst-4up.pdf