Chris Gregg
CS106B

Solution to Section #8

1. Prim's Algorithm

```
Set<Edge *>prim(BasicGraph &graph) {
    Set<Edge *> mst;
    if (graph.isEmpty()) {
        return mst;
    }
    Set<Vertex *> verticesSoFar;
    PriorityQueue<Edge*> pq;

    // Pick an arbitrary first vertex at which to start
    Vertex *lastVertexAdded = graph.getVertexSet().first();
    verticesSoFar.add(lastVertexAdded);

    // add new edges, and then pull out min edge connecting outwards
    // until we have connected all vertices
    while (verticesSoFar.size() < graph.vertexCount()) {
        // Lazily add all edges from the last vertex we added
        // outside the current tree
        Set<Edge *> edges = graph.getEdgeSet(lastVertexAdded);
        for (Edge *edge : edges) {
            if (!verticesSoFar.contains(edge->start)
                || !verticesSoFar.contains(edge->end)) {
                pq.enqueue(edge, edge->cost);
            }
        }

        Edge *nextEdge = pq.dequeue();
        while (verticesSoFar.contains(nextEdge->start)
               && verticesSoFar.contains(nextEdge->end)) {
            nextEdge = pq.dequeue();
        }

        if (!verticesSoFar.contains(nextEdge->start)) {
            lastVertexAdded = nextEdge->start;
        } else {
            lastVertexAdded = nextEdge->end;
        }

        verticesSoFar.add(lastVertexAdded);
        mst.add(nextEdge);
    }

    return mst;
}
```

2. Clustering

Main updates are:

- Instead of looping until the PQueue is empty, loop until we have found k clusters
- Make **allSets** into a Set of sets that keeps track of the unique sets we still have.

- At the end, convert each set of vertices into a set of strings of their names and return it, instead of changing a graph passed by reference.

```
Set<Set<string>> cluster(BasicGraph& graph, int k) {
    // NEW: removed mst graph because we are not building a graph anymore
    // put each vertex into a 'cluster', initially containing only itself
    Map<Vertex*, Set<Vertex*>* > clusters;
    Set<Vertex*> allVertices = graph.getVertexSet();
    Set<Set<Vertex*>* > allSets;    // NEW
    for (Vertex* v : allVertices) {
        Set<Vertex*>* set = new Set<Vertex*>();
        set->add(v);
        clusters[v] = set;
        allSets.add(set);
    }

    // put all edges into a priority queue, sorted by weight
    PriorityQueue<Edge*> pq;
    Set<Edge*> allEdges = graph.getEdgeSet();
    for (Edge* edge : allEdges) {
        pq.enqueue(edge, edge->cost);
    }

    // repeatedly pull min-weight edge out of PQ and add it to MST if its
    // endpoints are not already connected
    // NEW: removed mst set because we don't care about the edges anymore
    while (allSets.size() > k) {   // NEW
        Edge* e = pq.dequeue();
        Set<Vertex*>* set1 = clusters[e->start];
        Set<Vertex*>* set2 = clusters[e->finish];
        if (set1 != set2) {
            // merge the two sets
            set1->addAll(*set2);
            for (Vertex* v : *set1) {
                Set<Vertex*>* setv = clusters[v];
                if (setv != set1) {
                    clusters[v] = set1;
                }
            }
            delete set2;    // NEW
            allSets.remove(set2);  // NEW
        }
    }

    // ALL NEW BELOW
    Set<Set<string>> stringClusters;
    for (Set<Vertex *> * cluster : allSets) {
        Set<string> stringCluster;
        for (Vertex *v : *cluster) {
            stringCluster.add(v->name);
        }
        delete cluster;
        stringClusters.add(stringCluster);
    }
    return stringClusters;
}
```