Important

There are a few guidelines you must follow in this homework. If you fail to follow any of the following guidelines, you will recieve a **0** for the entire assignment.

1. All submitted code must compile under **JDK 7**. This includes unused code. Do not submit extra files that do not compile. Java is is backwards compatible so if it compiles under JDK 6, then it *should* compile under JDK 7.
2. Do not include any package declerations in your classes.
3. Do not modify any *existing* class headers, constructors, or method signatures. It is fine to add extra methods and classes.
4. Do not import anything that would trivialize the assignment. E.g. you should not be importing `java.util.LinkedList` for a Linked List assignment.
5. You must submit your source code, the `.java` files. Do not submit the compiled `.class` files.

After you submit your files, please redownload them and make sure that they are indeed what you intended to submit. We will not be responsible if you submit the wrong files.

Assignment

In this assignment, you will be constructing the Minimum Spanning Tree (MST) of some graph. Here are some definitions:

1. A **tree** is a type of graph where there is only one path from any one node to any other node. The number of vertices in a tree is equal to the number of edges minus one. A tree can contain no cycles.
2. A **spanning tree** is a type of tree that connects each and every vertex of the edge.
3. A **minimum spanning tree** is a spanning tree whose total edge weight is the lowest it can possibly be.

Just as in life, there are many methods of achiving our goals. In this assignment, you will be using Kruskal's algorithm achieve constructing a minimum spanning tree. To this end, you will use another data structure called a *Disjoint Sets*.

Kruskal's Algorithm

Kruskal's algorithm continuously seeks the least cost edge and adds it to the minimum spanning tree. If the edge should create a cycle, the algorithm ignores it and continues on to the next iteration.

Implementation:

1. Put every single edge in the priority queue.
2. Iterate until the priority queue is empty. At each iteration, pop off the least weight edge from the priority queue. Test to see if the edge creates a cycle. If it does, do not add it to your graph. Otherwise, add it and move on.
3. The hardest part of the algoirthm is to check for those bloody cycles. No matter, we will use Disjoint Sets to save the day!

Disjoint Sets

Disjoint sets can be used to determine if two elements belong to the same set. This will become useful for when we have to check for cycles in our MST algorithm. In Kruskal's algorithm, if two endpoints of and edge are part of the same set, then adding that edge will form a cycle in the graph. Remember that at any point in Kruskal's algorithm, different vertices of the tree might be in different sets. We have to choose edges that connect vertices in one set to another set, otherwise we're going to have cycles in our tree!

Implementation:

1. First, you must add all of the elements to the Disjoint Sets. Every element will be in its own set, since no element has been added to the MST yet. As soon as any two sets are conneted, we consider them to be part of one larger set.
2. Each vertex will have some parent. If two vertices have the same root, they are considered to be in the same set.
3. In `DisjointSets.java`, you will be given a Node class. The node class has space for a vertex and a parent pointer. You will modify these fields to construct your disjoint sets.
4. You should construct a hash map whose keys are vertices and whose values are Nodes.
5. In order to find out if two vertices belong to the same set, you should query the hash map for the appropriate nodes. You should then trace the parent pointers for these nodes and compare them. If their parent is the same they belong to the same set. If not, they do not belong to the same set.
6. To find a Node's parent, you should trace its parent pointer until you reach a node that does not have a parent of its own.
7. The disjoint sets should change whenever we add a valid edge. The valid edge makes a connection between two seperate sets. When we decide to add an edge to the graph, we have discovered the two parent pointers of the edge's endpoints to make sure they are not the same. To combine the two sets, we can simply set the parent pointer of one edge point to the parent pointer of another edge.
8. One optimization you should make is called **Path Compression**. Whenever you find the root of a node, point that node's parent pointer, as well as every node on the node's path to the root's parent poitner, point directly to the root you find. This will ***significantly*** cut down the running time of your Disjoint Sets. structure.

Prim's Algorithm

Prim's is another algorithm that we will use to find the Minimum Spanning Tree. Prim's is a greedy algorithm. Greedy algorithms are type of algorithms that take the best possible decision at the moment. Prim's starts from a starting node in the given graph and keeps looking for the smallest edge adjacent to the current node.

Implementation:

1. Put all of the neighbour edges of the current node in a priority queue.
2. Have a visited list that keeps the nodes that you have visited. This visited list will prevent you from having a cycle in your MST.

3. Iterate until the priority queue is empty. At each iteration, pop off the least weight edge from the priority queue. Test to see if the edge creates a cycle. If it does, do not add it to your graph. Otherwise, add it and move on.
4. The hardest part of the algoirthm is to check for those bloody cycles. We will use a visited list to check for the cycles.

Deliverables

You must submit the following files.

1. `DisjointSets.java`
2. `MST.java`
3. `AdjacencyList.java`

You may attach these files individually, or submit them into a zip archive.