Homework 3 Report

Qingyuan Zhang

My MST finding algorithm use the basic prim algorithm to finding the minimum spanning tree. However, I add a recursive method to let the prim algorithm to recursively finding the minimum spanning tree when we have the situation that edges have the same weight.

**public** List<SpanningTree> getMinimumSpanningTrees(Graph graph) {

List<SpanningTree> result = **new** ArrayList<SpanningTree>();

PriorityQueue<Edge> queue = **new** PriorityQueue<>();

SpanningTree tree = **new** SpanningTree();

Set<Integer> set = **new** HashSet<>();

add(queue, set, graph, 0);

getMinimumSpanningTreesAux(graph,tree,queue,set,result);

**return** result;

}

The main getMinimumSpanningTrees method is pretty simple. It is basically the same thing as prim method. We add the first vertex 0 to the getMinimumSpaningTreesAux method.

**public** **void** getMinimumSpanningTreesAux(Graph graph, SpanningTree tree,PriorityQueue<Edge> queue,Set<Integer> set,List<SpanningTree> trees){

**if** (tree.size()+1 == graph.size()){

trees.add(tree);

**return**;

}

**if**(clean(queue,set)){

**return**;

}

Edge edge;

SpanningTree copytree;

PriorityQueue<Edge> copyqueue;

Set<Integer> copyset;

**do**{

edge = queue.poll();

copytree = **new** SpanningTree(tree);

copyqueue = **new** PriorityQueue<Edge>(queue);

copyset = **new** HashSet<Integer>(set);

copytree.addEdge(edge);

add(copyqueue, copyset, graph, edge.getSource());

getMinimumSpanningTreesAux(graph,copytree,copyqueue,copyset,trees);

}

**while**(!queue.isEmpty()&& edge.compareTo(queue.peek())==0);

}

The main getMinimumSpanningTreesAux method is an advanced version of prim algorithm. It will add the minimum spanning tree to the list if the tree length plus 1 is equal to the graph size. Cleaning method is the method to clean up the repetitive edges when recursive call and also check whether the edges will form cycle. It will prevent the redundant spanning tree. Then we comes to the loop that very similar to the prim algorithm. Every time we meet the situation that we should choose between the same weighted edges. We have to make a recursive call to get a different MST. In order to pass the tree, queue and set value to the getMinimumSpanningTreesAux method and also keep the original value to recursively run, we have to make copies of tree, queue and set. And then we add polling out edge to the spanning tree since we already check it in the clean method. Finally we make the recursive call.

If we are considering a 4 vertices full graph with all edges weighted 1. My method is like the figure below. It starts with vertex 0 and then comes to peek the queue to find whether there is same weight minimum edges. If the queue have the same weight minimum edges we first pick one to find the minimum spanning tree with our recursive prim algorithm again. So if the red edges is the edges that we chose, we go to another vertices and try to find the edges again. So we keep the same process until we find the minimum spanning tree. Afterward, we trackback to the previous step which is a tree without the last edges and choose another same weighted edge. After we track back to the first edges we choose, we can eliminate from the graph since we already find all the minimum spanning tree including the edge. And since we are recursive calling, we start with the other same weight minimum edge and keep the same process.

…………….

**private** **boolean** clean(PriorityQueue<Edge> queue,Set<Integer> set){

List<Edge> remove = **new** ArrayList<Edge>();

**for**(Edge edge: queue)

{

**if**(set.contains(edge.getSource()))

remove.add(edge);

}

queue.removeAll(remove);

**return** queue.isEmpty();

}

The clean method as we discussed above is aim at clean the redundant edges in the queue and also remove the edges that forms cycle. It create a new copy of queue and add all the edges that will form the cycles to the copy queue and remove it from queue. This method also can stop the recursive prim method when the queue is empty.

The worst-case complexity of this algorithm is O(V^(V)). N represent the number of minimum spanning tree and V represent the number of vertex. We have to first do recursive call with V times and in each recursive call do the recursive again. So the recursive complexity it will be O(V^(V)).It is a very slow running time. I think it will become quicker if we try dynamic programming,

The proof of finding all the minimum spanning tree:

We start with a vertex and consider all the incoming edge of this vertex and choose the minimum weight. If there are same minimum weighted edges, we do recursive call and we keep the same process. Each incoming edge of every nodes will be consider in this algorithm and if there is situation of same weighted minimum edge we try to find the different spanning tree. So it will consider all the cases of minimum spanning tree.