「グラフと組合せ」課題 **10(**解答例**)**

2008/6/30

1 Kruskal アルゴリズム

連結無向グラフ $G=(V,A)$ に対して、重み関数

$$w: A \to R \tag{1}$$

が定義されているとき、 G の木 $T \subseteq A$ のうち、重み $w(T)$ が最小となる木を求めることを考える。

Krusukal アルゴリズムは以下のような方法である。木の初期値を $T=\emptyset$ とする。

```
while( T≠A ) {
  forall ( a∈A\T ) {
    if ( w(a) is the minimum within A\T )
      if ( T∪{a} does not contain circuits ) {
          T→T∪{a}
      }
    }
  }
}
```

このプログラムを以下を参考にして作成しなさい。ここで、探索すべきグラフ G は n 変数 network で与えられる。また結果となる木 T は変数 tree である。 G の頂点から T の頂点への対応は java.util.HashMap<graphLib.Vertex,graphLib.Vertex>のインスタンス mapVertex に保存され、 $v \in V$ に対応する頂点は mapVertex.get(a)で得ることができる。

また、深さ優先探索アルゴリズムに対応するクラス graphAnalysis.SerchDepthFirst は、頂点 v から w への道が無いとき、graphAnalysis.SerchDepthFirst.search(v , w)の戻り値が null となるとする。

```
package graphAnalysis;

public class Kruskal extends AbstractSearchMinimumTree{

    /** Creates a new instance of Kruskal */
    public Kruskal(graphLib.Network network) {
        super(network);
        mkVertexMap();
    }

    /* 探索実行
     * @return 探索結果のネットワーク */
    public graphLib.Network search(){
        arcList = new java.util.Vector<graphLib.Arc>();
        boolean end=false;
        while(!end){
            graphLib.Arc a=findArc();
            if(a==null){
                end=true;
            } else {
```

```
                    /* 新しい弧を見付けたら、リストに追加し、また tree に追加する */
                    arcList.add(a);
                    graphLib.Vertex v=a.from();
                    graphLib.Vertex t=a.to();
                    tree.addArc(mapVertex.get(v),mapVertex.get(t),a.toString());
                    if(debug)System.err.println("Add:"+a.toString());
                }
            }
            return tree;
        }

        /* 弧を見付ける
         * @return 見付けた弧
         */
        private graphLib.Arc findArc(){
            int k=network.getNumArcs();
            graphLib.Arc a=null;
            double w=maxWeight;
            for(int i=0;i<k;i++){
                graphLib.Arc aa=network.getArc(i);
                if(!arcList.contains(aa)){/* 弧は未だ使われていない */
                    if(aa.getWeight()<w){
                        graphLib.Vertex vt=mapVertex.get(aa.from());
                        graphLib.Vertex tt=mapVertex.get(aa.to());
                        graphAnalysis.SearchDepthFirst search=
                            new graphAnalysis.SearchDepthFirst(tree);
                        if(search.search(vt,tt)==null){
                            /* この弧の両端は既に連結されていない */
                            a=aa;
                            w=aa.getWeight();
                        }
                    }
                }
            }
            return a;
        }
}
```

```
package graphAnalysis;

public abstract class AbstractSearchMinimumTree {

        /**
         * graph と tree の頂点の対応関係
         * 元のグラフ内頂点から Tree の頂点へ
         */
        protected java.util.HashMap<graphLib.Vertex,graphLib.Vertex> mapVertex=null;
        /* Tree で使用した弧の一覧 */
        protected java.util.Vector<graphLib.Arc> arcList=null;
        /* Tree で使用した頂点の一覧 */
        protected java.util.Vector<graphLib.Vertex> vertexList=null;
```

```
    protected graphLib.Network network=null;
    protected graphLib.Network tree=null;
    protected double maxWeight=0.;
    protected final boolean debug=true;
    /** Creates a new instance of Kruskal */
    public AbstractSearchMinimumTree(graphLib.Network network) {
        this.network=network;
        getMaxWeight();
    }

    /* 弧の重みの和を求める */
    protected void getMaxWeight(){
        int k=network.getNumArcs();
        double w=0.;
        for(int i=0;i<k;i++){
            graphLib.Arc aa=network.getArc(i);
            w+=aa.getWeight();
        }
        maxWeight=w;
    }

    /* 頂点対応関係の作成 */
    protected void mkVertexMap(){
        mapVertex = new java.util.HashMap<graphLib.Vertex,graphLib.Vertex>();
        tree= new graphLib.Network("Minimum Tree for "+network.toString());
        for(graphLib.Vertex v: network.getVertexes()){
            graphLib.Vertex w=new graphLib.Vertex(v.toString());
            w.setPoint(v.getPoint());
            mapVertex.put(v,w);
            tree.addVertex(w);
        }
        tree.setDirected(network.isDirected());
    }

    /* 探索実行
     * @return 探索結果のネットワーク */
    abstract public graphLib.Network search();

}
```