CS6890 HW05

Jonathan Arndt

August 7 2017

Contents

0.1 Problems

0.1.1 Problem 1

See NetSimplex.java attached.

0.1.2 Problem 2

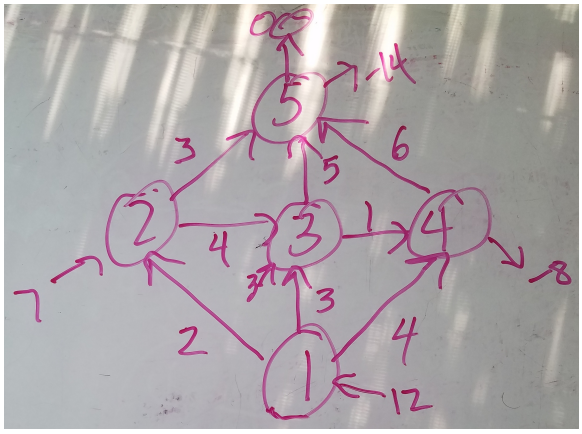

Figure 1: This is the Flow Net

Starting Basic feasible soltuion T(N,A),where N={1,2,3,4,5} and $A = \{(1,3),(1,4),(2,3),(3,5),(5,\infty)\}$

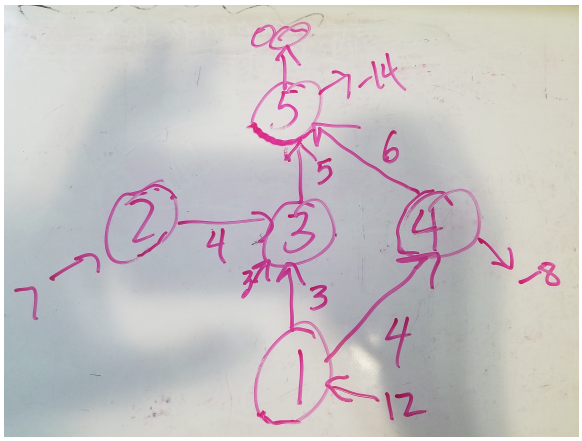

Figure 2: This is the Rooted Spanning Tree

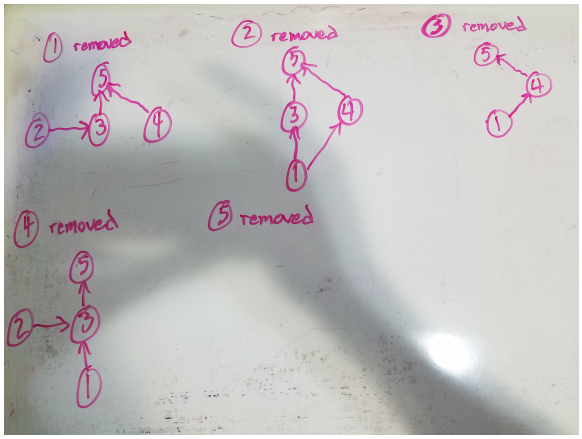

Figure 3: Removal Trees

	$(1,3)$	$(1,4)$	$(2,3)$	$(3,5)$	$(4,5)$	$RootArc$
1	-1	-1	0	0	0	0
2	0	0	-1	0	0	0
3	1	0	1	-1	0	0
4	0	1	0	0	-1	0
5	0	0	0	0	1	-1

```
Results in:
x12 = 14
x13 = 40
x34 = 20
x25 = 88
z   = 27.0
```

0.2 Source Code

0.2.1 NetSimplex.java

```
package assignment;

import utilities.spanningtree.Edge;
import utilities.spanningtree.MinimumSpanningTree;
import utilities.spanningtree.Node;

import java.util.List;

public class NetSimplex {
    public static void main(String[] args) {
        problem2();
    }
    public static void problem2(){
        String balance = "1=12,2=7,3=3,4=-8,5=-14";
        NetSimplex netSimplex = new NetSimplex(
                "{(1,2)=2,(1,3)=3,(1,4)=4,(2,3)=4,(2,5)=3," +
                        "(3,4),(3,5)=5,(4,5)=6,(5,inf)=-inf}",
                balance);
        netSimplex.setRootedSpanningTree("{(1,3),(1,4),(2,3),(3,5),(5,inf
        netSimplex.doWork();
        netSimplex.print();
    }

    private void print() {
        List<Edge> mst = minimumSpanningTree.getMinimumSpanningTree();
        MinimumSpanningTree mstNS = new MinimumSpanningTree(stringValue(m
        for(Edge e : mst)
            System.out.println(
                    (e.getEnd().getName().equals("inf")?"z   =  "+ (flow(m
                    "x"+e.getStart().getName()+e.getEnd().getName()+"  =  "
            );
    }

    private String stringValue(List<Edge> mst) {
        String s = "{";
        for(Edge e : mst)
            s+="("+e.getStart().getName()+","+e.getEnd().getName()+"="+e.
        return s;
    }
```

```
private String balance;
/**
 * algorithm network simplex;
 * begin
 *  determine an initial feasible tree structure(T,L,U);
 *  let x be the flow and pi be the node potentials associated with t
 *  while some nontree arc violates the optimality conditions do
 *  begin
 *      select an entering arc (k,l) violating its optimality conditi
 *      add arc (k,l) to the tree and determine the leaving arc (p,q)
 *      perform a tree update and update the solutions x and pi;
 *  end;
 * end;
 */
private MinimumSpanningTree minimumSpanningTree, rootedSpanningTree;
public NetSimplex(String flowNetwork, String balance){
    this.balance = balance;
    minimumSpanningTree = new MinimumSpanningTree(flowNetwork)
            .setBalance(balance);
}

public void setRootedSpanningTree(String spanningTree, String balance
    this.rootedSpanningTree = new MinimumSpanningTree(spanningTree).s
}

public void doWork(){
    double f = flow(rootedSpanningTree),
            np = nodePotential(rootedSpanningTree);
}

private double nodePotential(MinimumSpanningTree spanningTree) {
    Node rootNode = spanningTree.getNodes().get(minimumSpanningTree.g
            endNode = spanningTree.getNodes().get(minimumSpanningTree
    return _nodePotential(endNode,rootNode,0);
}

private double _nodePotential(Node endNode, Node rootNode, int i) {
    for(Edge e : rootNode.getToEdges()){
        if(e.getEnd().equals(endNode))
            i+=(int)(e.getEnd().getBalance()+e.getWeight());
        else
            i+=_nodePotential(endNode,e.getEnd(),(int)(e.getEnd().get
    }
    return i;
```

```
    }

    /**
     * Compute the flow by starting at the end nodes of the tree and work
     * @param spanningTree
     * @return
     */
    private double flow(MinimumSpanningTree spanningTree) {
        Node rootNode = spanningTree.getNodes().get(minimumSpanningTree.g
            endNode = spanningTree.getNodes().get(minimumSpanningTree.getI
        return _flow(endNode,rootNode,0);
    }

    private int _flow(Node endNode, Node rootNode, int v) {
        for(Edge n : endNode.getFromEdges()) {
            if (n.getStart().equals(rootNode))
                v += (int) (n.getStart().getBalance()+ n.getWeight());
            else
                v += _flow(n.getStart(), rootNode, (int) (n.getStart().ge
        }
        return v;
    }
}
```

0.2.2 MinimumSpanningTree.java

```
package utilities.spanningtree;

import utilities.Ratio;

import java.util.*;

public class MinimumSpanningTree {
    private HashMap<String,Node> nodes = new HashMap<>();
    private List<Edge> edgeList = new ArrayList<>(),
            spanningTreeEdges = new ArrayList<>();
    private Node rootNode;
    private Node endNode;


    public MinimumSpanningTree(String flowNetwork) {
        parse(flowNetwork);
    }

    public MinimumSpanningTree setBalance(String balance) {
        for(String s : balance.split(",")){
            String[] v = s.split("=");
            nodes.get(v[0]).setBalance(Integer.parseInt(v[1]));
        }

        return this;
    }

    private void parse(String flowNetwork) {
        String[] edges = flowNetwork.replaceAll("[{|}␣]", "").split(",\\(
        for(String ss : edges) {
            String[] v = ss.replaceAll("[\\(|\\)]", "").split("=");
            double weight = 1;
            if(v.length == 2) {
                v[1] = v[1].replace("inf",Integer.MAX_VALUE+"").replace("
                weight = Double.parseDouble(v[1]);
            }
            Edge e = new Edge(v[0]);
            e.setWeight(weight);
            Node start = e.getStart(), end = e.getEnd();
            if(nodes.containsKey(e.getStart().getName()))
                start = nodes.get(e.getStart().getName());
            else
```

```
                nodes.put(start.getName(),start);
            if(nodes.containsKey(e.getEnd().getName()))
                end = nodes.get(e.getEnd().getName());
            else
                nodes.put(end.getName(),end);
            e.setStart(start);
            e.setEnd(end);
            start.addToEdge(e);
            end.addFromEdge(e);
            edgeList.add(e);
        }
    }

    public HashMap<String, Node> getNodes() {
        return nodes;
    }

    public void setNodes(HashMap<String, Node> nodes) {
        this.nodes = nodes;
    }

    public List<Edge> getEdgeList() {
        return edgeList;
    }

    public void setEdgeList(List<Edge> edgeList) {
        this.edgeList = edgeList;
    }

    /**
     * Implementation using Prim's Algorithm to get the Minimum Spanning
     * Select Random Arbitrary node  A add A to visited list
     * Select smallest node from A->B  add B to visited list
     * Continue this process
     */
    public List<Edge> getMinimumSpanningTree(){
        Node a = getNodes().get("1");//getRandomNode();
        visited.add(a);
        while (visited.size() < nodes.size()) {
            List<Edge> edges = new ArrayList<>();
            visited.forEach(aa->edges.addAll(aa.getToEdges()));
            Collections.sort(edges);
            if(edges.size() == 0)
                break;
```

```
            for(Edge e : edges)
                if(!visited.contains(e.getEnd())){
                    visited.add(e.getEnd());
                    spanningTreeEdges.add(e);
                    break;
                }
        }
        return spanningTreeEdges;
    }
    private Set<Node> visited = new HashSet<>();
    private Random r = new Random();
    public Node getRandomNode() {
        return new ArrayList<>(nodes.values()).get(r.nextInt(nodes.size()
    }

    /**
     * The root node has no ins only outs
     * @return
     */
    public Node getRootNode() {
        for(Node n : nodes.values())
            if(n.getFromEdges().isEmpty())
                rootNode = n;
        return rootNode;
    }

    public Node getEndNode() {
        for(Node n : nodes.values())
            if(n.getToEdges().isEmpty())
                endNode = n;
        return endNode;
    }
}
```