CS5800: Algorithms — Spring '21 — Virgil Pavlu

Homework 12
Submit via Gradescope

Name: Xuran Feng
Collaborators:

Instructions:

- Make sure to put your name on the first page. If you are using the LaTeX template we provided, then you can make sure it appears by filling in the `yourname` command.
- Please review the grading policy outlined in the course information page.
- You must also write down with whom you worked on the assignment. If this changes from problem to problem, then you should write down this information separately with each problem.
- Problem numbers (like Exercise 3.1-1) are corresponding to CLRS 3^{rd} edition. While the 2^{nd} edition has similar problems with similar numbers, the actual exercises and their solutions are different, so make sure you are using the 3^{rd} edition.

1. ***(20 points)*** *Exercise 26.1-3.*

Solution: If there is no path $s \sim u \sim t$, there are three cases that there is no path $s \sim u$ or there is no path $u \sim t$. If there is no path $s \sim u$, then there should be no flow coming into u for all vertices v because all vertices connected to u should also have no path with s else there would be a path $s \sim u$, there is also no water coming out of u since vertex doesn't store water. Similarly, if it is the second case where the path $u \sim t$ doesn't exist, although u may have access to source and have a positive flow coming into itself, since vertex doesn't store water, u must send water out to other vertices, but other vertices cannot have a path with t else it would violate the assumption that there is no path $u \sim t$, so there must be no water coming out of vertex u because the water has no place to be stored, accordingly there is also no water coming into vertex u. In both cases, the maximum flow will never choose u as an intermediate and for all vertices v f(u,v)==f(v,u)=0 must be valid.

2. ***(20 points)*** *Exercise 26.1-4.*

Solution: According to the textbook, a flow in G that satisfies two properties: capacity constraint, $0 \leq f(u,v) \leq c(u,v)$; flow conservation, $\sum f(v,u) = \sum f(u,v)$. We need to show that $\alpha f_1 + (1-\alpha)f_2$ also satisfies these two properties. Because $0 \leq f_1 \leq c$ and $0 \leq f_2 \leq c$, so $\alpha f_1 + (1-\alpha)f_2 \geq \alpha \cdot 0 + (1-\alpha) \cdot 0 \geq 0$, $\alpha f_1 + (1-\alpha)f_2 \leq \alpha \cdot c + (1-\alpha) \cdot c \leq c$, thus the first property is satisfied. For the second property, $\sum f_1(v,u) = \sum f_1(u,v)$ and $\sum f_2(v,u) = \sum f_2(u,v)$ are satisfied, $\sum(\alpha f_1(v,u) + (1-\alpha)f_2(v,u))$=$\alpha \cdot \sum f_1(v,u)$+$(1-\alpha)\cdot\sum f_2(v,u)$=$\alpha\cdot\sum f_1(u,v)$+$(1-\alpha)\cdot\sum f_2(u,v)$=$\sum(\alpha f_1(u,v)$+$(1-\alpha)f_2(u,v))$, thus the second property is also satisfied, so $\alpha f_1 + (1-\alpha)f_2$ is also a flow of G.

3. ***(20 points)*** *Exercise 26.2-2.*

Solution: The flow across the cut is f(s,v_1)+f(v_2,v_1)+f(v_4,v_3)+f(v_4,t)-f(v_3,v_2)=11+1+7+4-4=19, the capacity of this cut is c(s,v_1)+c(v_2,v_1)+c(v_4,v_3)+c(v_4,t)=16+4+7+4=31.

4. ***(Extra Credit)*** *Exercise 26.2-10.*

Solution:

5. ***(30 points)*** *Implement Push-Relabel for finding maximum flow.*
Extra Credit: use relabel-to-front idea from Chapter 26.5 with the Discharge procedure.

Solution:

```java
import java.io.File;
import java.io.FileNotFoundException;
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

public class PushRelabel {
    public static void main(String[] args) throws FileNotFoundException {
        PushRelabel test=new PushRelabel("src/source.txt");
        test.printCapacity();
        test.printFlow();
        test.printResidual();
        test.printL();
        test.printVertices();

        test.RelabelToFront();

        test.printFlow();
        test.printResidual();
        test.printL();
        test.printVertices();
        test.printMaxFlow();
    }

    private class Vertex{
        private int name;
        private int height;
        private int excess;
        private List<Vertex> neighbours;
        private Vertex next;
        private int nextNeigh;
        public Vertex(int name){
            this.name=name;
            this.height=0;
            this.excess=0;
            this.neighbours=new ArrayList<>();
            this.next=null;
            nextNeigh=0;
        }

        public Vertex nextNeighbour(){
            if(nextNeigh<neighbours.size()) return neighbours.get(nextNeigh++);
            else {
                nextNeigh=0;
                return null;
            }
        }
    }
    private final Vertex dummyHead;
    private final int[][] capacity;
    private final int[][] flow;
    private final int[][] residual;
    private final Vertex[] vertices;

    public PushRelabel(String path) throws FileNotFoundException {
        dummyHead=new Vertex(-1);
        Scanner in =new Scanner(new File(path));
        int row= in.nextInt();
        capacity=new int[row][row];
        flow=new int[row][row];
        residual=new int[row][row];

        int i=0,j=0;
        while(in.hasNextInt()){
            capacity[i][j++]=in.nextInt();
            if(j==row){i++;j=0;}
        }

        Vertex iter=dummyHead;
        vertices=new Vertex[row];
        for(int k=0;k<row;k++) vertices[k]=new Vertex(k);
    for(int k=1;k<row-1;k++){
        for(int s=0;s<row;s++){
            if(capacity[k][s]!=0||capacity[s][k]!=0) vertices[k].neighbours.add(vertices[s]);
        }
        iter.next=vertices[k];
        iter=iter.next;
    }

    for(int k=0;k<row;k++){
        for(int s=0;s<row;s++){
            if(capacity[k][s]>0) residual[k][s]=capacity[k][s]-flow[k][s];
            else residual[k][s]=flow[s][k];
        }
    }

    vertices[0].height=row;
    for(int k=1;k<row;k++){
        flow[0][k]=capacity[0][k];
        vertices[k].excess=capacity[0][k];
        vertices[0].excess-=capacity[0][k];
        residual[0][k]-=capacity[0][k];
        residual[k][0]+=capacity[0][k];
    }
}

private void Push(Vertex u, Vertex v){
    int delta=Math.min(u.excess,residual[u.name][v.name]);
    if(capacity[u.name][v.name]>0)
        flow[u.name][v.name]+=delta;
    else
        flow[v.name][u.name]-=delta;

    residual[u.name][v.name]-=delta;
    residual[v.name][u.name]+=delta;
    u.excess-=delta;
    v.excess+=delta;
}
private void Relabel(Vertex u){
    int minHeight=Integer.MAX_VALUE;
    for(Vertex v:u.neighbours){
        if(residual[u.name][v.name]>0) minHeight=Math.min(minHeight,v.height);
    }
    u.height=minHeight+1;
}
private void Discharge(Vertex u){
    Vertex tmp;
    while(u.excess>0){
        tmp=u.nextNeighbour();
        if(tmp==null)
            Relabel(u);
        else if(residual[u.name][tmp.name]>0&&u.height==tmp.height+1)
            Push(u,tmp);
    }
}

public void RelabelToFront(){
    Vertex iter=dummyHead.next;
    Vertex prev=dummyHead;
    while(iter!=null){
        int oldHeight=iter.height;
        Discharge(iter);
        if(iter.height>oldHeight){
            prev.next=iter.next;
            iter.next=dummyHead.next;
            dummyHead.next=iter;
        }
        prev=iter;
        iter=iter.next;
    }
}

public void printResidual(){
        System.out.println("This is residual matrix:\n");
        for(int i=0;i<residual.length;i++){
            for(int j=0;j<residual.length;j++){
                System.out.print(residual[i][j]+"   ");
            }
            System.out.print("\n");
        }
    }

    public void printCapacity(){
        System.out.println("This is capacity matrix:\n");
        for(int i=0;i<capacity.length;i++){
            for(int j=0;j<capacity.length;j++){
                System.out.print(capacity[i][j]+"   ");
            }
            System.out.print("\n");
        }
    }

    public void printFlow(){
        System.out.println("This is flow matrix:\n");
        for(int i=0;i<flow.length;i++){
            for(int j=0;j<flow.length;j++){
                System.out.print(flow[i][j]+"   ");
            }
            System.out.print("\n");
        }
    }

    public void printL(){
        System.out.println("This is L and N table:");
        Vertex iter=dummyHead.next;
        while(iter!=null){
            System.out.println(iter.name+" has neighbours: ");
            for(Vertex v:iter.neighbours) System.out.print(v.name+" ");
            System.out.print("\n\n");
            iter=iter.next;
        }
    }

    public void printVertices(){
        System.out.println("This is summary of vertices:");
        for(Vertex v: vertices) System.out.println("Vertex "+v.name+" has excess "+v.excess+" and height "+v.height);
    }

    public void printMaxFlow(){
        System.out.println("Max flow is "+vertices[vertices.length-1].excess);
    }
}
```

6. ***(15 points)*** *Explain in a brief paragraph the following sentence from textbook page 737: "To make the preflow a legal flow, the algorithm then sends the excess collected in the reservoirs of overflowing vertices back to the source by continuing to relabel vertices to above the fixed height $|V|$ of the source".*

Solution: The source vertex is assigned a fixed height $|V|$ to make sure flowing back to source vertex is the last choice when flowing into sink is not possible but there is still water stored in the intermediate vertices. When intermediate vertices find no way to flow into the sink and the water simply cycle between intermediate vertices, to end the loop we need to find somewhere to clear reservoirs, and one choice is to flow back to source; to flow back to source, we need to relabel vertices above source's height which is $|V|$, since the intermedaite vertices' heights are above $|V|$, the extra flow stored in intermediate vertices can flow back into source, the flow matrix will become a valid one else the flow conservation will be violated at the end.

7. ***(Extra Credit)*** *Exercise 26.4.4.*

Solution: