**深 圳 大 学 实 验 报 告**

**课程名称： 算法分析与设计**

**实验项目名称： 最大流问题**

**学院： 计算机与软件学院**

**专业： 软件工程**

**指导教师： 李荣华**

**报告人： 洪继耀 学号： 2014150120 班级： 02**

1. 实验目的

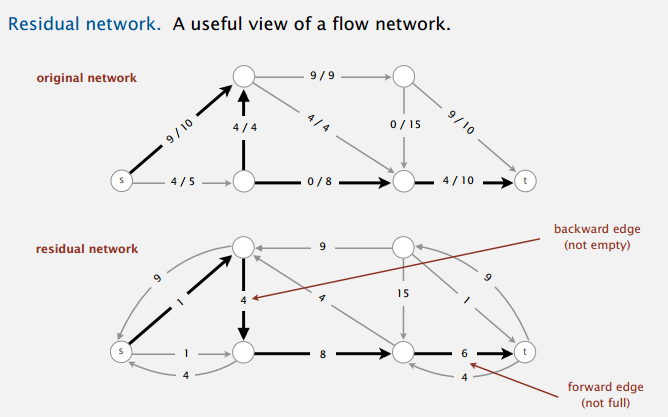
1. 掌握最短增益路径法思想。

2. 学会最大流问题求解方法。

1. 算法详解

1. ****残存网络****

给定网络G和流量f, 残存网络G\_f由那些仍有空间对流量进行调整的边构成。也就是 残留网络 = 容量网络capacity - 流量网络flow



1. 增广路径

增广路径p是残存网络中一条从源节点s到汇点t的简单路径,在一条增广路径p上能够为每条边增加的流量的最大值为路径p的残存容量

1. 残存容量

在一条增广路径p上, 要增加整条增广路径的流量, 则必须看最小能承受流量的管道, 不然水管会爆掉, 这最小承受流量就是残存容量

# 4.**最大流最小割理论**

设f为流网络G = (V, E)中的一个流, 该流网络的源节点为s, 汇点为t, 则下面的条件是等价的：

A.f是G的一个最大流

B.残存网络G\_f不包含任何增广路径

C.|f| = c(S, T), 其中(S, T)是流网络G的某个割

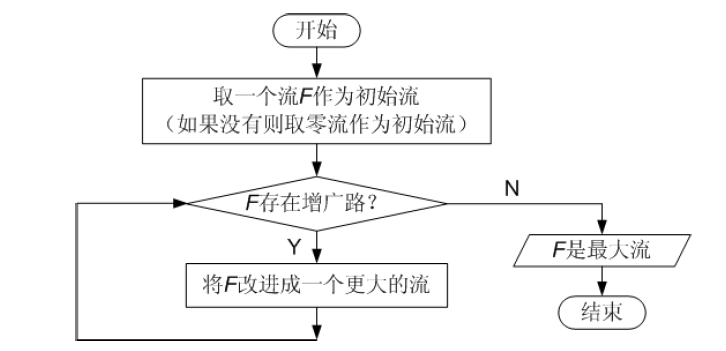
# **5.Ford-Fulkerson算法**

根据定义可以推断出来，如果一个方案f中存在增广路，那么它就不是最优的，我们需要把这条增广路给修理好。于是就有了下面增广路算法：为了得到最大流，可以从任何一个可行流开始，沿着增广路对网络流进行优化修改，直到网络中不存在增广路为止，算法的基本流程是：

(1)取一个可行流f作为初始流（如果没有给定初始流，则取零流f={ 0 }作为初始流）；

(2)寻找关于f的增广路P，如果找到，则沿着这条增广路P将f改进成一个更大的流；

(3)重复第(2)步直到f不存在增广路为止。



1. 实验代码
2. 对于流边的定义**class** FlowEdge {  
    **private final int v**, **w**; *// 边的起点和终点* **private final double capacity**; *// 容量* **private double flow**; *// 流量  
     
    /\*\*  
    \* 获取流量  
    \*  
    \** ***@return*** *流量  
    \*/* **double** getFlow() {  
    **return flow**;  
    }  
     
    */\*\*  
    \* 构造函数  
    \*  
    \** ***@param v*** *起点  
    \** ***@param w*** *终点  
    \** ***@param capacity*** *容量  
    \*/* FlowEdge(**final int** v, **final int** w, **final double** capacity) {  
    **this**.**v** = v;  
    **this**.**w** = w;  
    **this**.**capacity** = capacity;  
    }  
     
    */\*\*  
    \* 获取起点  
    \*  
    \** ***@return*** *起点  
    \*/* **int** from() {  
    **return v**;  
    }  
     
    */\*\*  
    \* 获取终点  
    \*  
    \** ***@return*** *终点  
    \*/* **int** to() {  
    **return w**;  
    }  
     
    */\*\*  
    \* 获取另外一个端点  
    \*  
    \** ***@param vertex*** *其中一个端点  
    \** ***@return*** *另外一个端点  
    \*/* **int** other(**final int** vertex) {  
    **if** (vertex == **v**) {  
    **return w**;  
    } **else if** (vertex == **w**) {  
    **return v**;  
    } **else** {  
    **throw new** RuntimeException(**"Inconsistent edge"**);  
    }  
    }  
     
    */\*\*  
    \* 获取残留流量  
    \*  
    \** ***@param vertex*** *流向终点端点  
    \** ***@return*** *残余容量  
    \*/* **double** residualCapacityTo(**final int** vertex) {  
    **if** (vertex == **v**) { *//反向边* **return flow**;  
    } **else if** (vertex == **w**) { *//正向边* **return capacity** - **flow**;  
    } **else** {  
    **throw new** IllegalArgumentException(**"点错乱了"**);  
    }  
    }  
     
    */\*\*  
    \* 压入流量delta  
    \*  
    \** ***@param vertex*** *流向终点端点  
    \** ***@param delta*** *流大小  
    \*/* **void** addResidualFlowTo(**final int** vertex, **final double** delta) {  
    **if** (vertex == **v**) { *// 倒流* **flow** -= delta;  
    } **else if** (vertex == **w**) { *// 正流* **flow** += delta;  
    } **else** {  
    **throw new** IllegalArgumentException();  
    }  
    }  
     
   }
3. 对于流图的定义

**import** java.util.ArrayList;**class** FlowNetwork {  
 **private final int V**; *// 顶点个数* **private** ArrayList<FlowEdge>[] **adj**;  
 */\*\*  
 \* 构造函数  
 \*  
 \** ***@param V*** *顶点的个数  
 \*/* FlowNetwork(**final int** V) {  
 **this**.**V** = V;  
 **adj** = **new** ArrayList[V];  
 **for** (**int** v = 0; v < V; v++) {  
 **adj**[v] = **new** ArrayList<>();  
 }  
 }  
  
 */\*\*  
 \* 增加边  
 \*  
 \** ***@param e*** *边  
 \*/* **void** addEdge(**final** FlowEdge e) {  
 **int** v = e.from();  
 **int** w = e.to();  
 **adj**[v].add(e); *// 正向边v->w* **adj**[w].add(e); *// 反向边w->e* }  
  
 */\*\*  
 \* 返回顶点数量  
 \*  
 \** ***@return*** *顶点数量  
 \*/* **int** V() {  
 **return V**;  
 }  
  
 */\*\*  
 \* 返回邻接边  
 \*  
 \** ***@param v*** *某个顶点  
 \** ***@return*** *它的领接边们  
 \*/* Iterable<FlowEdge> adj(**final int** v) {  
 **return adj**[v];  
 }  
}

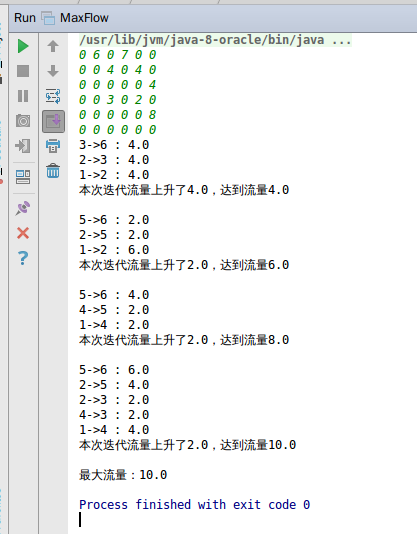
1. 核心工具类

**import** java.util.LinkedList;  
**import** java.util.Queue;**class** FordFulkerson {  
 **private** FlowEdge[] **edgeTo**; *// s->t路径的边的集合（逆增广路径）* **private double value**; *// 流大小  
  
 /\*\*  
 \* 构造函数，构造的同时就不断寻找增广路径并更新最大流  
 \*  
 \** ***@param G*** *图  
 \** ***@param s*** *起点（源点）  
 \** ***@param t*** *终点（汇点）  
 \*/* FordFulkerson(**final** FlowNetwork G, **final int** s, **final int** t) {  
 **value** = 0.0;  
 **while** (hasAugmentingPath(G, s, t)) {  
 **double** bottle = Double.***POSITIVE\_INFINITY***; *// 最大值* **for** (**int** v = t; v != s; v = **edgeTo**[v].other(v)) { *// 沿着t->s计算最大流量* bottle = Math.*min*(bottle, **edgeTo**[v].residualCapacityTo(v));  
 }  
 **for** (**int** v = t; v != s; v = **edgeTo**[v].other(v)) {  
 **edgeTo**[v].addResidualFlowTo(v, bottle); *// 给边重新计算流量* }  
 **value** += bottle; *// 最大流量上升* **for** (**int** v = t; v != s; v = **edgeTo**[v].other(v)) {  
 **int** from = **edgeTo**[v].from() + 1;  
 **int** to = **edgeTo**[v].to() + 1;  
 System.***out***.println(from + **"->"** + to + **" : "** + **edgeTo**[v].getFlow());  
 }  
 System.***out***.println(**"本次迭代流量上升了"** + bottle + **"，达到流量"** + **value**);  
 System.***out***.println();  
 }  
 }  
  
 */\*\*  
 \* 图的遍历算法，判断是不是有增广路径  
 \*  
 \** ***@param G*** *图  
 \** ***@param s*** *起点（源点）  
 \** ***@param t*** *终点（汇点）  
 \** ***@return*** *是和否  
 \*/* **private boolean** hasAugmentingPath(**final** FlowNetwork G, **final int** s, **final int** t) {  
 **edgeTo** = **new** FlowEdge[G.V()];  
 **boolean**[] marked = **new boolean**[G.V()]; *// 记录在残留网络中s（源点）->t是否可达* Queue<Integer> q = **new** LinkedList<>(); *// 用队列组织广度优先搜索算法* q.offer(s);  
 marked[s] = **true**; *// s->s必然可达* **while** (!q.isEmpty()) {  
 **int** v = q.poll(); *// 每次取出队首v（访问之）* **for** (FlowEdge e : G.adj(v)) { *// v的所有领接边* **int** w = e.other(v); *// 领接点w  
 // 如果还有残余流量并且这条路并未被使用* **if** (e.residualCapacityTo(w) > 0 && !marked[w]) { *// e到w的流量大于0且s->w未判断* **edgeTo**[w] = e; *// 放入增广路径* marked[w] = **true**; *// 判断s->w可达* q.offer(w); *// w放入队首（发现之）* }  
 }  
 }  
 *// 顺利到达终点的话 marked[t]==true s可达t于是存在增广路径* **return** marked[t];  
 }  
  
 */\*\*  
 \* 获取最大流大小  
 \*  
 \** ***@return*** *大小  
 \*/* **double** value() {  
 **return value**;  
 }  
  
}

1. 测试类

**import** java.util.Scanner;**public class** MaxFlow {  
 **private static final int *V*** = 6; *// 点的个数* **public static void** main(**final** String[] args) {  
  
 FlowNetwork flowNetwork = **new** FlowNetwork(***V***);  
  
 Scanner in = **new** Scanner(System.***in***);  
  
 **double** capacity;  
 **for** (**int** i = 0; i < ***V***; i++) {  
 **for** (**int** j = 0; j < ***V***; j++) {  
 capacity = in.nextDouble();  
 **if** (capacity > 0) {  
 FlowEdge e = **new** FlowEdge(i, j, capacity);  
 flowNetwork.addEdge(e);  
 }  
 }  
 }  
 FordFulkerson fordFulkerson = **new** FordFulkerson(flowNetwork, 0, ***V*** - 1);  
 System.***out***.println(**"最大流量："** + fordFulkerson.value());  
 }  
}

1. 运行截图



1. 实验总结

研究最大流问题很有意义。

|  |
| --- |
| 指导教师批阅意见：  成绩评定：  指导教师签字：  年 月 日 |
| 备注： |

注：1、报告内的项目或内容设置，可根据实际情况加以调整和补充。

2、教师批改学生实验报告时间应在学生提交实验报告时间后10日内。