云南大学数学与统计学院  
《算法图论实验》上机实践报告

|  |  |  |
| --- | --- | --- |
| 课程名称：算法图论实验 | 年级：2015级 | 上机实践成绩： |
| 指导教师：李建平 | 姓名：刘鹏 | 专业：信息与计算科学 |
| 上机实践名称：有向图的弧连通度 | 学号：20151910042 | 上机实践日期：2019-01-01 |
| 上机实践编号：6 | 组号： |  |

# 实验目的

1. 了解图的点连通度和弧连通度的定义；
2. 了解Menger定理的描述以及证明。

# 实验内容

1. 写出求图的弧连通度的算法；
2. 用C语言实现上述算法。

# 实验平台

Windows 10 Pro 1803；

MacOS Mojave。

# 算法设计

## 预备知识

为了刻画图的连通程度，引入图的连通度的概念。设连通图不是完全图，如果，使得是不连通的，则称是图的**点截集**（vertex-cut）。如果，也称之为-点截集。给定图的两个顶点和，假如点截集使得顶点和彼此不可到达，那么称点截集是图中**分离的点截集**（vertex-cut separating and ）。

定义图的点连通度（vertex-connectivity），记为，

对于非负整数，若，则称是-点连通图（-vertex-connected Graph），简称-连通图。

类似地可以定义边截集。给定连通图，如果，使得是不连通的，则称是图的**边截集**（edge-cut）。如果，也称之为-边截集。类似地定义**分离的边截集**（edge-cut separating and ）。定义图的边连通度（edge-connectivity），记为，

设是一个图，和是的任意两个非空真子集。给定一个链，如果它的两个端点分别属于和，而且链上的其他点都不属于，则称这个链是-**链**（-chain）。特别地，若，则本身就是一个-链。若，且任何一个-链都与相交，则称是一个-**分离集**（-separator），与也是-分离集。

**Menger定理** 和是的任意两个子集，则中存在个点不交的-链，当且仅当每个-分离集至少包含个点。

**Menger推论1** 设，，则中存在个点不交的-链，当且仅当中的每个分离的点截集至少包含个点。

**Menger推论2** 设，，则中存在个边不交的-链，当且仅当中的每个分离的边截集至少包含个边。

（**该推论的证明比较有启发性**，可以利用图的线图来证明，线图必然满足Menger定理，所以根据这个观察结果，直接证明本推论。）

**关于点的Menger型极大极小定理** 设是的两个不相邻的点，则内点不交的-链的最大个数等于局部点连通度。

**关于边的Menger型极大极小定理** 设是的两个不相邻的点，则内边不交的-链的最大个数等于局部点边通度。

## 算法解析

有了Menger定理的铺垫，就可以很顺利地写出求图的点连通度的算法，其核心思想是网络流算法。因为有点不交这个限制条件，所以不能单纯地讲所有边的容量设置为1。这直接导致了网络流算法中出现了节点的流量限制，但是Edmonds-Karp算法中并没有对于节点进行限制，所以需要对原图进行一定的变换。-链集合如图(*a*)所示，对于其中的蓝色节点来说，一定是偶数，可以将之拆分为两个节点，中间加一个边，且该边的容量设置为，这样就可以解除节点容量限制，直接在原图中进行网络最大流算法。(*a*)图是原先的，(*b*)图是经过变换之后的图。这样就可以解决点连通度的求解问题。对于边连通度问题而言要简单一点，直接在原图上采取网络流算法即可，通过遍历所有的点对，找出最小值即可。



## 求有向图的弧连通度的算法

假设已经有Edmonds-Karp算法，可以用来求解有向图的顶点与之间的最大网络流，且记这个形式为，这个算法是下面这个算法的子算法。

|  |  |
| --- | --- |
| **Algorithm**  **Input**  **Output**  **Begin**  **Step 1**  **Step 2**  **Step 3**  **Step 4**  **End** | 求有图的弧连通度，记此算法为  连通的有向图  图的一个弧连通度，记  // 初始化一个空的列表  // 初始化弧的容量  **for** **each** edge :    // 寻找任意有序点对之间的最大流量  **for each** vertex :  **for each** vertex **and** :    **output** |

# 程序代码

源代码采用了开源社区写的图论算法库，这里仅仅展示我编写的一个核心文件graph.cpp。

## 程序代码

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127  128  129  130  131  132  133  134  135  136  137  138  139  140  141  142  143 | #include "graph.hpp"  // Operator overloads  std**::**ostream **&** **operator<<(**std**::**ostream **&**stream**,** const Graph **&**g**)**  **{**  stream **<<** "Number of vertices in graph: " **<<** g**.**size**()** **<<** std**::**endl**;**  **for** **(**uint32\_t i **=** 0**;** i **<** g**.**size**();** **++**i**)**  **{**  std**::**cout **<<** "Vertex: " **<<** i **<<** " has neighbors: " **<<** std**::**endl**;**  **for** **(**auto **&**neighbor **:** g**[**i**])**  **{**  stream **<<** neighbor **<<** ", "**;**  **}**  stream **<<** std**::**endl**;**  **}**  **return** stream**;**  **}**  template**<**typename Vec**,** typename VecType**>**  void fill1DVec**(**Vec **&** vec**,** VecType vt**)**  **{**  vec**.**assign**(**vec**.**size**(),** vt**);**  **}**  template**<**typename Vec**,** typename VecType**>**  void fill2DVec**(**Vec **&** vec**,** VecType vt**)**  **{**  **for** **(**auto **&** row **:** vec**)**  **{**  fill1DVec**(**row**,** vt**);**  **}**  **}**  uint32\_t EdgeConnectivityCalculator**::**bfs**(**uint32\_t u**,** uint32\_t v**)**  **{**  uint32\_t vertexCount **=** g\_**.**size**();**  std**::**queue**<**uint32\_t**>** queue**;**  queue**.**push**(**u**);**  **while** **(!**queue**.**empty**())**  **{**  const uint32\_t i **=** queue**.**front**();**  queue**.**pop**();**  **for** **(**uint32\_t j **=** 0**;** j **<** g\_**[**i**].**size**();** **++**j**)**  **{**  uint32\_t neighbor **=** g\_**[**i**][**j**];**  uint32\_t currentResidualCapacity **=** edgeCapacity\_**[**i**][**j**]** **-** currentFlow\_**[**i**][**j**];**  **if** **(**path\_**[**neighbor**]** **==** **-**1 **&&** **(**currentResidualCapacity **>** 0**))**  **{**  path\_**[**neighbor**]** **=** i**;**  residualCapacity\_**[**neighbor**]** **=** std**::**min**(**residualCapacity\_**[**i**],** currentResidualCapacity**);**  **if** **(**neighbor **!=** v**)**  **{**  queue**.**push**(**neighbor**);**  **}**  **else**  **{**  **return** residualCapacity\_**[**v**];**  **}**  **}**  **}**  **}**  **return** 0**;**  **}**  uint32\_t EdgeConnectivityCalculator**::**edmondsKarp**(**uint32\_t source**,** uint32\_t sink**)**  **{**  clearData**();**  uint32\_t pathFlow **=** 0**,** maxFlow **=** 0**;**  **while** **(true)**  **{**  fill1DVec**(**residualCapacity\_**,** 0**);**  fill1DVec**(**path\_**,** **-**1**);**  residualCapacity\_**[**source**]** **=** std**::**numeric\_limits**<**uint32\_t**>::**max**();**  path\_**[**0**]** **=** **-**2**;**  pathFlow **=** bfs**(**source**,** sink**);**  **if** **(**pathFlow **<=** 0**)**  **{**  **break;**  **}**  maxFlow **+=** pathFlow**;**  uint32\_t s **=** sink**;**  **while** **(**s **!=** source**)**  **{**  uint32\_t prev **=** path\_**[**s**];**  // Workaround, maybe we should use a matrix instead of a 2d list after all?  uint32\_t indexOfPrev **=** std**::**distance**(**g\_**[**s**].**begin**(),** std**::**find**(**g\_**[**s**].**begin**(),** g\_**[**s**].**end**(),** prev**));**  uint32\_t indexOfS **=** std**::**distance**(**g\_**[**prev**].**begin**(),** std**::**find**(**g\_**[**prev**].**begin**(),** g\_**[**prev**].**end**(),** s**));**  currentFlow\_**[**prev**][**indexOfS**]** **+=** pathFlow**;**  currentFlow\_**[**s**][**indexOfPrev**]** **-=** pathFlow**;**  s **=** prev**;**  **}**  **}**  **return** maxFlow**;**  **}**  uint32\_t EdgeConnectivityCalculator**::**findEdgeConnectivity**()**  **{**  **if** **(**g\_**.**size**()** **<** 1**)**  **return** 0**;**  uint32\_t result **=** g\_**[**0**].**size**();**  **for** **(**uint32\_t i **=** 1**;** i **<** g\_**.**size**();** i**++)**  **{**  result **=** std**::**min**(**result**,** edmondsKarp**(**0**,** i**));**  **}**  **return** result**;**  **}**  void EdgeConnectivityCalculator**::**clearData**()**  **{**  fill2DVec**(**currentFlow\_**,** 0**);**  fill2DVec**(**edgeCapacity\_**,** 1**);**  fill1DVec**(**path\_**,** **-**1**);**  fill1DVec**(**residualCapacity\_**,** 0**);**  residualCapacity\_**[**0**]** **=** std**::**numeric\_limits**<**uint32\_t**>::**max**();** // not necessary here?  path\_**[**0**]** **=** **-**2**;** // so we can visit the first vertex in BFS  **}**  void EdgeConnectivityCalculator**::**initializeDataFromGraph**(**Graph **&** g**)**  **{**  // Copy graph structure  g\_ **=** Graph**(**g**);**  edgeCapacity\_ **=** Graph**(**g**);**  currentFlow\_ **=** FlowVecType**(**g**.**size**());**  **for** **(**uint32\_t i **=** 0**;** i **<** currentFlow\_**.**size**();** i**++)**  **{**  currentFlow\_**[**i**].**resize**(**g\_**[**i**].**size**(),** 0**);**  **}**  residualCapacity\_ **=** std**::**vector**<**uint32\_t**>(**g**.**size**());**  path\_ **=** std**::**vector**<**int**>(**g**.**size**());**  //clearData();  **}**  EdgeConnectivityCalculator**::**EdgeConnectivityCalculator**()**  **{**  **}**  uint32\_t EdgeConnectivityCalculator**::**findEdgeConnectivity**(**Graph **&** g**)**  **{**  initializeDataFromGraph**(**g**);**  **return** findEdgeConnectivity**();**  **}** |

## 运行结果



这里的图结构是用迭代器模式实现的邻接表。如下所示为1.in的文件内容

0,1,2,3

1,0,2

2,0,1,3

3,0,2,4,5

4,3,5

5,3,4

转化为图片为：



代码库[3]是里斯本大学的一位同学写的实验报告，他详细地分析了三个算法的性能，在实验中有所参考他的一些代码。

# 参考文献

[1] **林锐**. 高质量 C++/C 编程指南 [M]. 1.0 ed., 2001.

[2] <https://github.com/matrackif/EdgeConnectivityUsingFlow/tree/master/EdgeConnectivity>

[3] <https://github.com/miferrei/Edge-Connectivity-in-Graphs>