

中国科学技术大学计算机学院
《数据结构》报告



实验题目：图及其应用
学生姓名：王章瀚
学生学号：PB18111697
完成日期：2019 年 12 月 8 日

计算机实验教学中心制
2019 年 09 月

1 实验要求

本次实验分为四个小实验。分别为

- 寻找关节点
- 寻找最短路径
- 无向图的可视化
- 无向图中的最长圈

其中，无向图的可视化较为简单，只需要输出对应 Graphviz 的代码，故不多加阐述，只展示代码和效果。下面逐一进行分析。

1.1 寻找关节点

1.1.1 概述

参考教材 P177-178, 算法 7.10 和 7.11，基于邻接矩阵的存储结构，使用非递归的深度优先搜索算法，求无向连通图中的全部关节点，并按照顶点编号升序输出。

1.1.2 输入与输出

输入的第一行是一个正整数 n ，表示图中的顶点数（顶点编号从 0 到 $n-1$ ）。之后的若干行是无序对 (i, j) ，表示顶点 i 与顶点 j 之间有一条边相连。

样例如下：

输入：13

0 1

0 2

0 5

0 11

1 2

1 3

1 6

1 7

1 12

3 4

6 7

6 8

6 10

7 10

9 11

9 12

11 12

输出：0 1 3 6

1.2 寻找最短路径

1.2.1 概述

基于邻接表的存储结构，依次输出从顶点 0 到顶点 1、2、……、n-1 的最短路径和各路径中的顶点信息。

1.2.2 输入与输出

输入的第一行是一个正整数 n ，表示图中的顶点数（顶点编号从 0 到 $n-1$ ）。之后的若干行是无序对 (i, j) ，表示顶点 i 与顶点 j 之间有一条边相连。

样例如下：

```
输入和前面一个问题一样。输出：1 0->1
1 0->2
2 0->1->3
3 0->1->3->4
1 0->5
2 0->1->6
2 0->1->7
3 0->1->6->8
2 0->11->9
3 0->1->6->10
1 0->11
2 0->1->12
```

1.3 无向图中的最长圈

1.3.1 概述

以邻接矩阵的形式给定带权的无向图，判断图中是否存在圈，若存在则输出图中的最长圈和圈的长度，圈的长度定义为圈上的边权值之和，而不是边的数量。

1.3.2 输入与输出

样例如下：

```

输入：
第三列是权
10
0 4 2
0 9 1
1 2 6
1 3 3
1 4 9
2 4 5
2 6 5
3 5 6
5 6 4
5 9 6
6 7 5
7 8 5
邻接矩阵
0 表示无穷
0 0 0 0 2 0 0 0 0 1
0 0 6 3 9 0 0 0 0 0
0 6 0 0 5 0 5 0 0 0
0 3 0 0 0 6 0 0 0 0
2 9 5 0 0 0 0 0 0 0
0 0 0 6 0 0 4 0 0 6
0 0 5 0 0 4 0 5 0 0
0 0 0 0 0 0 5 0 5 0
0 0 0 0 0 0 0 5 0 0
1 0 0 0 0 6 0 0 0 0

输出：
33
0->4->1->2->6->5->9->0

```

2 设计思路

2.1 寻找关节点

寻找关节点的算法，可以利用课本上的思路。即对图进行 DFS 遍历，同时维护一个数组 low 来判断是否是一个关节点。

这里

$$low[v] = Min \left\{ \begin{array}{l} visited[v], low[w], visited[k] \end{array} \right\} \left| \begin{array}{l} w \text{ 是顶点 } v \text{ 在 DFS 生成树上的孩子结点} \\ k \text{ 是顶点 } v \text{ 在 DFS 生成树上由回边联结的祖先结点} \\ (v, w) \in Edge \\ (v, k) \in Edge \end{array} \right\}.$$

这样一来，只要发现有一个根节点，它的子树中的结点它有指向祖先的回边，那么它就不是关节点。

此后只需要对深度优先搜索稍作修改来维护 low 这个数组即可。

2.2 寻找最短路径

最短路径的算法有很多。本次实验中，我使用的是 Dijkstra 算法。其具体思路就是：每次找未确定项中到起点最短的项，加入确定了项。然后对其他未确定的项进行一个遍历，看是否会缩短路径。经过所有遍历后，所有项都已经确定，此时算法结束。

与书上不同的是，这里用一个数组 front 来维护每个项的前驱。这样能够方便的找到任意项的最短路径上所有项。

2.3 无向图中的最长圈

由于没有什么较好的思路，采用比较暴力但不失有效性的方法。即，从某点出发，通过 DFS 进行遍历，同时寻找回边，如果有回边回到根节点，说明得到了一个包含该顶点的圈，这时便可以对它进行判断，确定新的最大圈。

此后，从所有项出发，执行一遍上述流程，最后即可得到最长圈。

下面流程图对从一个顶点出发找最大圈的算法稍作描述：

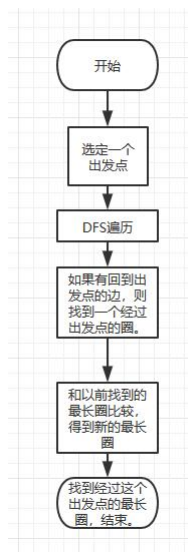


图 1: 无向图中最长圈部分算法流程图

3 关键代码讲解

3.1 邻接矩阵图类

```

1  // 图类，用邻接矩阵表示无向图
2  class MGraph {
3  private:
4      int* visited = NULL;
5      // 从顶k开始由DFS寻找最大圈。
6      void FindMaxCircleThrough(int& k, int v, int weight, vector<int>* curCircle);
7  public:
8      Vertex* vexs = NULL;
9      int** arcs = NULL; //邻接矩阵，值直接是权，若为0，则表示不相邻
10     bool edgeWeighted = false;
11     int vexNum = 0, arcNum = 0;
12     vector<int> Articul;
13
14     bool hasCircle = false;
15     int maxWeight = 0;
16     vector<int> maxCircle;
17
18     MGraph(string fileName);
19     // 打印邻接矩阵
20     void PrintMatrix();
21     // 找关节的启动函数，最终找到所有割顶放在Articul中
22     void FindArticul();
23     // 找割顶的DFS函数
  
```

```

24     void DFSArticul(int v0, int& count, int* visited, int* low);
25     // 找最大圈的函数
26     void FindMaxCircle();
27     // 打印最大圈
28     void PrintMaxCircle();
29     // 打印割顶
30     void PrintArticul();
31     // 打印可以用于生成Graphviz图的代码 ( )
32     void PrintGraphvizCode();
33 };
34

```

邻接矩阵图类

3.2 邻接表图类

其中包含弧结点类，顶类，图类。

```

1 // 弧结点类。认为弧权均为1，故不设多余信息成员变量
2 class ArcNode {
3     public:
4         int adjVex; // 该弧指向的顶点索引
5         ArcNode* nextArc; // 指向下一条弧
6
7         ArcNode(int av, ArcNode* na) : adjVex(av), nextArc(na) {};
8 };
9
10 // 顶类。
11 class VNode {
12     public:
13         int index; // 顶点序号
14         ArcNode* firstArc; // 第一条邻边
15
16         // 默认构造方法
17         VNode() : index(-1), firstArc(nullptr) {};
18         // 由序号和第一条邻边构造顶
19         VNode(int index, ArcNode* fa) : index(index), firstArc(fa) {};
20         // 添加一条关联边
21         void addAdjArc(ArcNode* arcnode);
22 };
23
24 class ALGraph {
25     public:
26         VNode* vexs; // 顶
27         int vexNum = 0, arcNum = 0; // 顶点数，边数
28         int* distance = NULL; // 由Dijkstra算法得到的最短路径距离
29         int* front = NULL; // 由Dijkstra算法得到的前驱结点
30
31         // 构造函数：通过文件名构造
32         ALGraph(string fileName);
33         // Dijkstra算法实现

```



```

34     void ShortestPath_DIJ();
35     // 通过front数组来寻找到0的最短路径
36     void printShortestPath(int i);
37     // 通过front数组来寻找到0的最短路径
38     void printAllShortestPath();
39     // 打印邻接表
40     void printAdjacentList();
41 };
42

```

邻接表图类

3.3 寻找关节点代码讲解

这里值得一说的是：为了用栈来实现，在函数内定义了一个状态类，其中包含 v,w,min。其含义与书上递归函数中变量的含义相同。此后只需要用一个状态类栈来完成程序。

```

1  void MGraph::DFSArticul(int v0, int& count, int* visited, int* low) {
2      // 状态类
3      class State {
4      public:
5          int v;
6          int w;
7          int min;
8          State(int v, int w, int min) : v(v), w(w), min(min) {};
9      };
10
11     stack<State*> DFSSStack;
12     int w = -1;
13     DFSSStack.push(new State(v0, w, count+1));
14     visited[v0] = ++count;
15
16     State* s = DFSSStack.top();
17     while (!DFSSStack.empty()) {
18         s = DFSSStack.top();
19         // 找下一个邻顶 (生成树的孩子)
20         s->w++;
21         while (s->w < vexNum && arcs[s->v][s->w] == 0) s->w++;
22         if (s->w >= vexNum) {
23             // 则说明所有孩子都访问过了, 应pop
24             // pop前设置好low
25             low[s->v] = s->min;
26             DFSSStack.pop();
27             if (DFSSStack.empty())
28                 return;
29             delete s;
30             // 对前面的结果进行一些处理, 更新min值或确认割顶
31             s = DFSSStack.top();
32             if (low[s->w] < s->min) s->min = low[s->w];

```

```

33         if (low[s->w] >= visited[s->v]) {
34             Articul.push_back(s->v);
35         }
36         continue;
37     }
38     if (!visited[s->w]) {
39         // 如果没被访问过, 则访问它
40         State* nexts = new State(s->w, -1, -1);
41         visited[nexts->v] = nexts->min = ++count;
42         DFSSStack.push(nexts);
43         continue;
44     }
45     else if (visited[s->w] < s->min) {
46         // 否则, 尝试修改s->min (如果需要)
47         s->min = visited[s->w];
48     }
49 }
50 }
51

```

寻找关节点代码

3.4 寻找最短路径代码讲解

代码运用的是 Dijkstra 算法, 因此只稍作注释, 不多加描述。

```

1 void ALGraph::ShortestPath_DIJ()
2 {
3     // 一些初始化
4     bool* defined = new bool[vexNum];
5     for (int i = 0; i < vexNum; i++) {
6         distance[i] = INT_MAX;
7         defined[i] = false;
8     }
9     // 先考虑0项
10    int v = 0;
11    ArcNode* an = vxs[v].firstArc;
12    while (an != NULL) {
13        distance[an->adjVex] = 1;
14        front[an->adjVex] = v;
15        an = an->nextArc;
16    }
17    // 遍历所有项
18    for (int i = 0; i < vexNum; i++) {
19        int minD = INT_MAX;
20        // 找最短路径项
21        for (int j = 0; j < vexNum; j++) {
22            if (!defined[j]) {
23                if (distance[j] < minD) {
24                    v = j; minD = distance[j];
25                }
26            }
27        }
28    }
29 }

```

```

26         }
27     }
28     defined[v] = true;
29     // 更新最短路径
30     an = vexs[v].firstArc;
31     while (an != NULL) {
32         if (distance[an->adjVex] > minD + 1) {
33             distance[an->adjVex] = minD + 1;
34             front[an->adjVex] = v;
35         }
36         an = an->nextArc;
37     }
38 }
39 }
40

```

寻找最短路径代码

3.5 输出无向图的 Graphviz 代码讲解

代码内容只是对图的连接情况作一个格式化的输出，没有什么不易理解的点，注释如下：

```

1 void MGraph::PrintGraphvizCode() {
2     if (!edgeWeighted) {
3         // 非带权图
4         cout << "graph g {" << endl;
5         for (int i = 0; i < vexNum; i++) {
6             for (int j = i; j < vexNum; j++) {
7                 if (arcs[i][j] != 0) {
8                     cout << "\t" << i << "—" << j << endl;
9                 }
10            }
11        }
12        cout << "}" << endl;
13    }
14    else {
15        // 带权图
16        cout << "graph g {" << endl;
17        for (int i = 0; i < vexNum; i++) {
18            int a1 = -1;
19            int a2 = -1;
20            for (int k = 0; k < maxCircle.size(); k++) {
21                // 确定k是否是最大圈路径上的顶，如果是，用a1,a2记录其邻顶。
22                if (maxCircle[k] == i) {
23                    a1 = k == maxCircle.size() - 1 ? maxCircle[0] : maxCircle[k
24                        + 1];
25                    a2 = k == 0 ? maxCircle[maxCircle.size() - 1] : maxCircle[k
26                        - 1];
27                    break;
28                }
29            }
30        }
31    }
32 }

```

```

27         }
28         for (int j = i; j < vexNum; j++) {
29             if (arcs[i][j] != 0) {
30                 // 带颜色考虑的输出
31                 cout << "\t" << i << "—" << j
32                 << "[label = \" << arcs[i][j] << \"\""
33                 << ", color = \" << (j == a1 || j == a2 ? "red" : "black")
34                 << "\"]" << endl;
35             }
36         }
37         cout << "]" << endl;;
38     }
39 }
40

```

输出无向图的 Graphviz 代码

3.6 无向图中最长圈代码讲解

实际使用时，对每个顶都是用一次下述函数。

```

1 void MGraph::FindMaxCircleThrough(int& k, int v, int weight, vector<int>*& curCircle)
2 {
3     visited[v] = true;
4     curCircle->push_back(v);
5     for (int i = 0; i < vexNum; i++) {
6         if (arcs[v][i] != 0) {
7             if (i == k && curCircle->size() > 2) {
8                 //说明找到了一个圈
9                 if (weight + arcs[v][i] > maxWeight) {
10                     // 比之前找到的更长
11                     maxWeight = weight + arcs[v][i];
12                     maxCircle = *curCircle;
13                     continue;
14                 }
15             }
16             else if (!visited[i]) {
17                 // 未访问顶，则继续遍历
18                 FindMaxCircleThrough(k, i, weight + arcs[v][i], curCircle);
19                 continue;
20             }
21         }
22     }
23     curCircle->pop_back();
24     visited[v] = false;
25 }

```

无向图中最长圈代码

4 调试分析

4.1 寻找关节点

4.1.1 问题发现与解决

编写代码的时候，发现并不那么容易用栈来实现这样一个函数。因此最后无可奈何地创建了一个状态类来表示状态，进而使得整个代码和递归极其相似。

4.1.2 算法的时空复杂度分析

由于采用邻接矩阵，故空间复杂度几乎为 $o(v^2)$

时间复杂度上，由于只是进行了一次深度优先搜索，由于查找下一个结点需要 $o(n)$ ，而类似递归需要 $o(n)$ ，综合起来应该为 $o(n^2)$

4.2 寻找最短路径

4.2.1 问题发现与解决

在这个代码中，由于使用的是 Dijkstra 算法，原理上比较清晰，没有什么大问题。

4.2.2 算法的时间复杂度分析

Dijkstra 算法中，每次为了确定一个顶来寻找有最小路径的顶用了 $o(n)$ ，此外，对所有顶都要执行一次这样的操作，因此时间复杂度应该为 $o(n^2)$ 。

4.3 无向图中的最长圈

4.3.1 问题发现与解决

使用深度优先搜索时，应该注意返回之后对 visited 等数据结构的回溯操作，否则将产生错误。

4.3.2 算法的时间复杂度分析

每一次深度优先搜索去寻找圈的时候，都用了 $o(n^2)$ 的时间复杂度，而对每个顶执行一次深度优先搜索，因此总的时间复杂度是 $o(n^3)$

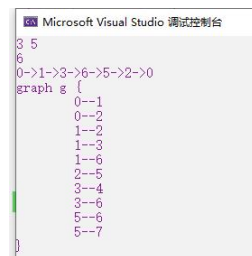
5 代码测试

5.1 寻找关节点

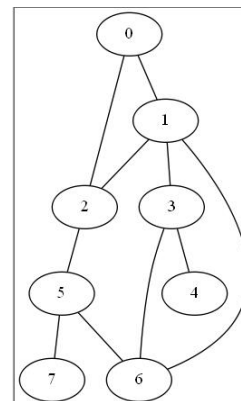
5.1.1 输入 1:

```
8
0 1
0 2
1 2
1 3
1 6
2 5
3 4
3 6
5 6
5 7
```

输出截图 1:



```
Microsoft Visual Studio 调试控制台
3 5
6
0->1->3->6->5->2->0
graph g {
    0--1
    0--2
    1--2
    1--3
    1--6
    2--5
    3--4
    3--6
    5--6
    5--7
}
```

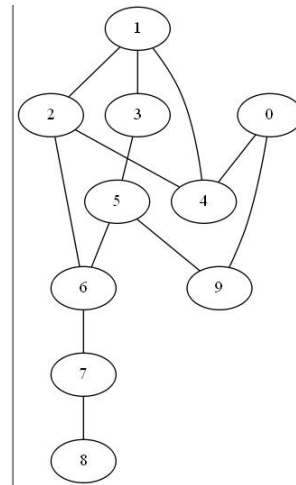


5.1.2 输入 2:

```
10
0 4
0 9
1 2
1 3
1 4
2 4
2 6
2 6
3 5
5 6
5 9
6 7
7 8
```

输出截图 2:

```
Microsoft Visual Studio 调试控制台
6 7
7
0->4->1->2->6->5->9->0
graph g {
    0--4
    0--9
    1--2
    1--3
    1--4
    2--4
    2--6
    2--6
    3--5
    5--6
    5--9
    6--7
    7--8
}
```

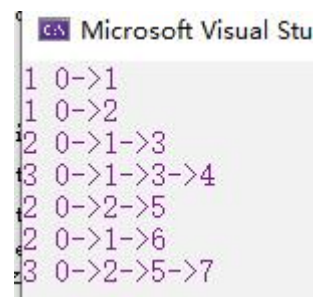


5.2 寻找最短路径

5.2.1 输入 1:

```
8
0 1
0 2
1 2
1 3
1 6
2 5
3 4
3 6
5 6
5 7
```

输出截图 1:



```
Microsoft Visual Stu
1 0->1
1 0->2
2 0->1->3
3 0->1->3->4
2 0->2->5
2 0->1->6
3 0->2->5->7
```


5.2.2 输入 2:

```
10
0 4
0 9
1 2
1 3
1 4
2 4
2 6
3 5
5 6
5 9
6 7
7 8
```

输出截图 2:



```
Microsoft Visual Studio 调试控制台
2 0->4->1
2 0->4->2
3 0->4->1->3
1 0->4
2 0->9->5
3 0->4->2->6
4 0->4->2->6->7
5 0->4->2->6->7->8
1 0->9
```

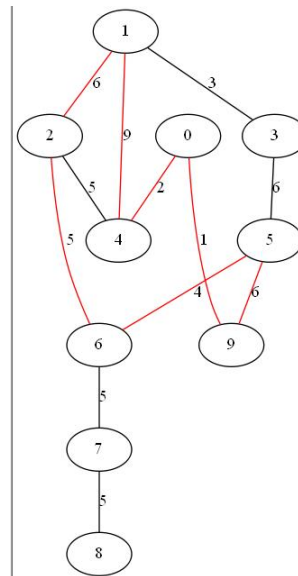
5.3 无向图中的最长圈

5.3.1 输入 1:

```
10
0 4 2
0 9 1
1 2 6
1 3 3
1 4 9
2 4 5
2 6 5
3 5 6
5 6 4
5 9 6
6 7 5
7 8 5
```

输出截图 1:

```
Microsoft Visual Studio 调试控制台
6 7
33
0->4->1->2->6->5->9->0
graph g
{
0--4[label = "2", color = "red"]
0--9[label = "1", color = "red"]
1--2[label = "6", color = "red"]
1--3[label = "3", color = "black"]
1--4[label = "9", color = "red"]
2--4[label = "5", color = "black"]
2--6[label = "5", color = "red"]
3--5[label = "6", color = "black"]
5--6[label = "4", color = "red"]
5--9[label = "6", color = "red"]
6--7[label = "5", color = "black"]
7--8[label = "5", color = "black"]
}
```

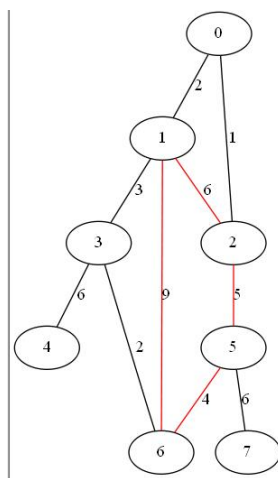


5.3.2 输入 2:

```
8
0 1 2
0 2 1
1 2 6
1 3 3
1 6 9
2 5 5
3 4 6
3 6 2
5 6 4
5 7 6
```

输出截图 2:

```
Microsoft Visual Studio 调试控制台
3 5
24
1->2->5->6->1
graph g {
    0--1[label = "2", color = "black"]
    0--2[label = "1", color = "black"]
    1--2[label = "6", color = "red"]
    1--3[label = "3", color = "black"]
    1--6[label = "9", color = "red"]
    2--5[label = "5", color = "red"]
    3--4[label = "6", color = "black"]
    3--6[label = "2", color = "black"]
    5--6[label = "4", color = "red"]
    5--7[label = "6", color = "black"]
}
```



6 实验总结

本次实验中，主要运用了邻接矩阵表示图和邻接表表示图。通过实验可以发现，邻接矩阵便于查找边权关系，而使用邻接表图时，可以方便地查找下一个邻边，以较好地完成遍历操作。另外，图论中的各种定理也比较丰富，应该尽快熟悉各大常用定理，并熟练编写其程序。

7 附录

7.1 附录 A. 关节点寻找、最长圈寻找及 Graphviz 代码的输出程序

```
1 // PB18111697_王章瀚_4_AdjacentMatrix.cpp : 此文件包含 "main" 函数。程序执行将在此处
  开始并结束。
2 //
3
4 #include <iostream>
5 #include <fstream>
6 #include <sstream>
7 #include <string>
8 #include <climits>
9 #include <vector>
10 #include <stack>
11 #include <algorithm>
12
13 using namespace std;
14
15 class Vertex {
16 public:
17     int index;
18 };
19
20 // 图类，用邻接矩阵表示无向图
21 class MGraph {
22 private:
23     int* visited = NULL;
24     // 从顶k开始由DFS寻找最大圈。
25     void FindMaxCircleThrough(int& k, int v, int weight, vector<int>* curCircle);
26 public:
27     Vertex* vexs = NULL;
28     int** arcs = NULL; //邻接矩阵，值直接是权，若为0，则表示不相邻
29     bool edgeWeighted = false;
30     int vexNum = 0, arcNum = 0;
31     vector<int> Articul;
32
33     bool hasCircle = false;
34     int maxWeight = 0;
35     vector<int> maxCircle;
36
37     MGraph(string fileName);
38     // 打印邻接矩阵
39     void PrintMatrix();
40     // 找关节点的启动函数，最终找到所有割顶放在Articul中
41     void FindArticul();
42     // 找割顶的DFS函数
43     void DFSArticul(int v0, int& count, int* visited, int* low);
44     // 找最大圈的函数
45     void FindMaxCircle();
```

```

46 // 打印最大圈
47 void PrintMaxCircle();
48 // 打印割顶
49 void PrintArticul();
50 // 打印可以用于生成Graphviz图的代码 ( )
51 void PrintGraphvizCode();
52 };
53
54 void MGraph::FindMaxCircleThrough(int& k, int v, int weight, vector<int>* curCircle)
55 {
56     visited[v] = true;
57     curCircle->push_back(v);
58     for (int i = 0; i < vexNum; i++) {
59         if (arcs[v][i] != 0) {
60             if (i == k && curCircle->size() > 2) {
61                 //说明找到了一个圈
62                 if (weight + arcs[v][i] > maxWeight) {
63                     // 比之前找到的更长
64                     maxWeight = weight + arcs[v][i];
65                     maxCircle = *curCircle;
66                     continue;
67                 }
68             }
69             else if (!visited[i]) {
70                 // 未访问顶, 则继续遍历
71                 FindMaxCircleThrough(k, i, weight + arcs[v][i], curCircle);
72                 continue;
73             }
74         }
75     }
76     curCircle->pop_back();
77     visited[v] = false;
78 }
79
80 MGraph::MGraph(string fileName) {
81     ifstream fs(fileName, ios::in | ios::out);
82     // 读入顶点数
83     string line;
84     fs >> vexNum;
85     getline(fs, line);
86     // 分配顶点的空间
87     veks = new Vertex[vexNum];
88     if (veks == NULL) return;
89     for (int i = 0; i < vexNum; i++) {
90         veks[i].index = i;
91     }
92     // 分配visited的空间
93     visited = new int[vexNum];
94     if (visited == NULL) return;
95     memset(visited, 0, sizeof(int) * vexNum);
96     // 分配边矩阵空间
97     arcs = new int* [vexNum];
98     if (arcs == NULL) return;
99     for (int i = 0; i < vexNum; i++) {

```

```

99         arcs[i] = new int[vexNum];
100         if (arcs[i] == NULL) return;
101         memset(arcs[i], 0, sizeof(int)*vexNum);
102     }
103     // 读入数据
104     while(!fs.eof()) {
105         getline(fs, line);
106         stringstream ss(line);
107         int h, t;
108         ss >> h >> t;
109         // 看该行是否有第三个数，如果有说明输入了边权
110         if (ss.eof()) {
111             arcs[h][t] = 1;
112             arcs[t][h] = 1;
113         }
114         else {
115             edgeWeighted = true;
116             ss >> arcs[h][t];
117             arcs[t][h] = arcs[h][t];
118         }
119         arcNum++;
120     }
121     if (arcNum >= vexNum) hasCircle = true;
122 }
123
124 void MGraph::PrintMatrix() {
125     for (int i = 0; i < vexNum; i++) {
126         for (int j = 0; j < vexNum; j++) {
127             cout << arcs[i][j] << " ";
128         }
129         cout << endl;
130     }
131     cout << endl;
132 }
133 // 这里假设图连通
134 void MGraph::FindArticul() {
135     // 全局计数
136     int count = 1;
137     // 定义访问过标记数组
138     int* visited = new int[vexNum];
139     if (visited == NULL) return;
140     memset(visited, 0, sizeof(int) * vexNum);
141     // 定义low数组
142     int* low = new int[vexNum];
143     if (low == NULL) return;
144     memset(low, 0, sizeof(int) * vexNum);
145
146     // 第一个结点vexs[0]已经访问
147     visited[0] = 1;
148     int v = -1;
149     while (arcs[0][++v] == 0);
150     DFSArticul(v, count, visited, low);
151     if (count < vexNum) {
152         // 说明生成树中至少有两棵子树，这时根vexs[0]是割顶

```

```

153     Articul.push_back(0);
154     // 继续访问其他子树
155     v++;
156     while (v < vexNum && arcs[0][v] == 0) v++;
157     DFSArticul(v, count, visited, low);
158 }
159 // 对割顶进行排序
160 sort(Articul.begin(), Articul.end(), less<int>());
161 }
162
163 void MGraph::DFSArticul(int v0, int& count, int* visited, int* low) {
164     // 状态类
165     class State {
166     public:
167         int v;
168         int w;
169         int min;
170         State(int v, int w, int min) : v(v), w(w), min(min) {};
171     };
172
173     stack<State*> DFSStack;
174     int w = -1;
175     DFSStack.push(new State(v0, w, count+1));
176     visited[v0] = ++count;
177
178     State* s = DFSStack.top();
179     while (!DFSStack.empty()) {
180         s = DFSStack.top();
181         // 找下一个邻顶 (生成树的孩子)
182         s->w++;
183         while (s->w < vexNum && arcs[s->v][s->w] == 0) s->w++;
184         if (s->w >= vexNum) {
185             // 则说明所有孩子都访问过了, 应pop
186             // pop前设置好low
187             low[s->v] = s->min;
188             DFSStack.pop();
189             if (DFSStack.empty())
190                 return;
191             delete s;
192             // 对前面的结果进行一些处理, 更新min值或确认割顶
193             s = DFSStack.top();
194             if (low[s->w] < s->min) s->min = low[s->w];
195             if (low[s->w] >= visited[s->v]) {
196                 bool isIn = false;
197                 for (auto a : Articul) {
198                     if (a == s->v) {
199                         isIn = true;
200                         break;
201                     }
202                 }
203                 if (!isIn) Articul.push_back(s->v);
204             }
205             continue;
206         }

```

```

207         if (!visited[s->w]) {
208             // 如果没被访问过，则访问它
209             State* nexts = new State(s->w, -1, -1);
210             visited[nexts->v] = nexts->min = ++count;
211             DFSStack.push(nexts);
212             continue;
213         }
214         else if (visited[s->w] < s->min) {
215             // 否则，尝试修改s->min (如果需要)
216             s->min = visited[s->w];
217         }
218     }
219 }
220
221 }
222
223 void MGraph::FindMaxCircle() {
224     for (int k = 0; k < vexNum; k++) {
225         memset(visited, 0, sizeof(int) * vexNum);
226         FindMaxCircleThrough(k, k, 0, new vector<int>);
227     }
228 }
229
230 void MGraph::PrintMaxCircle()
231 {
232     if (hasCircle) {
233         cout << maxWeight << endl << maxCircle[0];
234         int vsize = maxCircle.size();
235         for (int i = 1; i < vsize; i++) {
236             cout << "<span style='color: purple;'>->" << maxCircle[i];
237         }
238         cout << "<span style='color: purple;'>->" << maxCircle[0] << endl;
239     }
240     else {
241         cout << "没有圈" << endl;
242     }
243 }
244
245 void MGraph::PrintArticul() {
246     for (auto e : Articul) {
247         cout << e << ' ';
248     }
249     cout << endl;
250 }
251
252 void MGraph::PrintGraphvizCode() {
253     if (!edgeWeighted) {
254         // 非带权图
255         cout << "graph g {" << endl;
256         for (int i = 0; i < vexNum; i++) {
257             for (int j = i; j < vexNum; j++) {
258                 if (arcs[i][j] != 0) {
259                     cout << "\\t" << i << "<span style='color: purple;'>—" << j << endl;
260                 }

```



```

261         }
262     }
263     cout << "}" << endl;
264 }
265 else {
266     // 带权图
267     cout << "graph g {" << endl;
268     for (int i = 0; i < vexNum; i++) {
269         int a1 = -1;
270         int a2 = -1;
271         for (int k = 0; k < maxCircle.size(); k++) {
272             // 确定k是否是最大圈路径上的顶, 如果是, 用a1,a2记录其邻顶。
273             if (maxCircle[k] == i) {
274                 a1 = k == maxCircle.size() - 1 ? maxCircle[0] : maxCircle[k
+ 1];
275                 a2 = k == 0 ? maxCircle[maxCircle.size() - 1] : maxCircle[k
- 1];
276                 break;
277             }
278         }
279         for (int j = i; j < vexNum; j++) {
280             if (arcs[i][j] != 0) {
281                 // 带颜色考虑的输出
282                 cout << "\t" << i << "—" << j
283                 << "[label = \" << arcs[i][j] << "\"\"
284                 << ", color = \" << (j == a1 || j == a2 ? "red" : "black")
<< "\" ]" << endl;
285             }
286         }
287     }
288     cout << "}" << endl;;
289 }
290 }
291
292 int main()
293 {
294     MGraph mg("optional_input2.txt");
295     mg.PrintMatrix();
296     mg.FindArticul();
297     mg.PrintArticul();
298     mg.FindMaxCircle();
299     mg.PrintMaxCircle();
300     mg.PrintGraphvizCode();
301 }
302

```

关节点寻找、最长圈寻找及 Graphviz 代码的输出程序

7.2 附录 B. 最短路径搜寻程序

```

1 // PB18111697_王章瀚_4_AdjacentList.cpp : 此文件包含 "main" 函数。程序执行将在此处开
  始并结束。
2 //
3
4 #include <iostream>
5 #include <fstream>
6 #include <sstream>
7 #include <list>
8 #include <string>
9 #include <queue>
10 #include <stack>
11
12 using namespace std;
13
14 // 弧结点类。认为弧权均为1，故不设多余信息成员变量
15 class ArcNode {
16 public:
17     int adjVex; // 该弧指向的顶点索引
18     ArcNode* nextArc; // 指向下一条弧
19
20     ArcNode(int av, ArcNode* na) : adjVex(av), nextArc(na) {};
21 };
22
23 // 顶点类。
24 class VNode {
25 public:
26     int index; // 顶点序号
27     ArcNode* firstArc; // 第一条邻边
28
29     // 默认构造方法
30     VNode() : index(-1), firstArc(nullptr) {};
31     // 由序号和第一条邻边构造顶点
32     VNode(int index, ArcNode* fa) : index(index), firstArc(fa) {};
33     // 添加一条关联边
34     void addAdjArc(ArcNode* arcnode);
35 };
36
37 class ALGraph {
38 public:
39     VNode* vexs; // 顶点
40     int vexNum = 0, arcNum = 0; // 顶点数，边数
41     int* distance = NULL; // 由Dijkstra算法得到的最短路径距离
42     int* front = NULL; // 由Dijkstra算法得到的前驱结点
43
44     // 构造函数：通过文件名构造
45     ALGraph(string fileName);
46     // Dijkstra算法实现
47     void ShortestPath_DIJ();
48     // 通过front数组来寻找0的最短路径
49     void printShortestPath(int i);
50     // 通过front数组来寻找0的最短路径
51     void printAllShortestPath();
52     // 打印邻接表
53     void printAdjacentList();

```

```

54     };
55
56     void VNode::addAdjArc(ArcNode* arcnode)
57     {
58         if (firstArc == NULL) {
59             firstArc = arcnode;
60             return;
61         }
62         ArcNode* p = firstArc;
63         while (p->nextArc != NULL) p = p->nextArc;
64         p->nextArc = arcnode;
65         return;
66     }
67
68     void ALGraph::ShortestPath_DIJ()
69     {
70         // 一些初始化
71         bool* defined = new bool[vexNum];
72         for (int i = 0; i < vexNum; i++) {
73             distance[i] = INT_MAX;
74             defined[i] = false;
75         }
76         // 先考虑0顶
77         int v = 0;
78         ArcNode* an = vexs[v].firstArc;
79         while (an != NULL) {
80             distance[an->adjVex] = 1;
81             front[an->adjVex] = v;
82             an = an->nextArc;
83         }
84         // 遍历所有顶
85         for (int i = 0; i < vexNum; i++) {
86             int minD = INT_MAX;
87             // 找最短路径顶
88             for (int j = 0; j < vexNum; j++) {
89                 if (!defined[j]) {
90                     if (distance[j] < minD) {
91                         v = j; minD = distance[j];
92                     }
93                 }
94             }
95             defined[v] = true;
96             // 更新最短路径
97             an = vexs[v].firstArc;
98             while (an != NULL) {
99                 if (distance[an->adjVex] > minD + 1) {
100                     distance[an->adjVex] = minD + 1;
101                     front[an->adjVex] = v;
102                 }
103                 an = an->nextArc;
104             }
105         }
106     }
107

```

```

108 ALGraph::ALGraph(string fileName) {
109     fstream fs(fileName, ios::in | ios::out);
110     // 读入顶点数
111     string line;
112     fs >> vexNum;
113     getline(fs, line);
114     // 分配顶点的空间
115     vexs = new VNode[vexNum];
116     if (vexs == NULL) return;
117     for (int i = 0; i < vexNum; i++) {
118         vexs[i].index = i;
119     }
120     // 分配distance空间
121     distance = new int[vexNum];
122     if (distance == NULL) return;
123     memset(distance, 0, sizeof(int) * vexNum);
124     // 分配front空间
125     front = new int[vexNum];
126     if (front == NULL) return;
127     memset(front, 0, sizeof(int) * vexNum);
128
129     // 读入数据
130     while (!fs.eof()) {
131         getline(fs, line);
132         stringstream ss(line);
133         int h, t;
134         ss >> h >> t;
135         vexs[h].addAdjArc(new ArcNode(t, nullptr));
136         vexs[t].addAdjArc(new ArcNode(h, nullptr));
137         arcNum++;
138     }
139 }
140
141 void ALGraph::printShortestPath(int i)
142 {
143     int t = i;
144     stack<int> s;
145     while (t != 0) {
146         s.push(t);
147         t = front[t];
148     }
149     int ssize = s.size();
150     cout << distance[i] << " 0";
151     for (int t = 0; t < ssize; t++) {
152         cout << ">" << s.top();
153         s.pop();
154     }
155     cout << endl;
156 }
157
158 void ALGraph::printAllShortestPath()
159 {
160     for (int i = 1; i < vexNum; i++) {
161         printShortestPath(i);

```

```

162     }
163 }
164
165 void ALGraph::printAdjacentList()
166 {
167     for (int i = 0; i < vexNum; i++) {
168         cout << i << ": ";
169         ArcNode* p = vexs[i].firstArc;
170         while (p != NULL) {
171             cout << p->adjVex << " ";
172             p = p->nextArc;
173         }
174         cout << endl;
175     }
176 }
177
178
179
180 int main()
181 {
182     ALGraph alg("input2.txt");
183     alg.ShortestPath_DIJ();
184     alg.printAllShortestPath();
185 }
186

```

最短路径搜寻程序