

# 湖南大学

数据结构

## 课程实验报告

题    目	<u>无向图中求两点间的所有简单路径</u>
学生姓名	<u>吴多智</u>
学生学号	<u>201626010520</u>
专业班级	<u>软件 1605</u>
完成日期	<u>2017 年12 月15 日</u>

## 一、需求分析

### 1. 问题描述

若用无向图表示高速公路网，其中顶点表示城市，边表示城市之间的高速公路。设计一个找路路径，获取两个城市之间的所有简单路径（找到除了起点和终点可以相同外，其它顶点均不同的路径）。

### 2. 输入数据

- (1) 从键盘上输入城市的个数；
- (2) 从键盘上输入城市之间高速公路的条数；
- (3) 从键盘上输入各城市的编号；
- (4) 从键盘上输入各高速公路的起点和终点；
- (5) 从键盘上输入要获取路径的两个城市的编号。

### 3. 输出数据

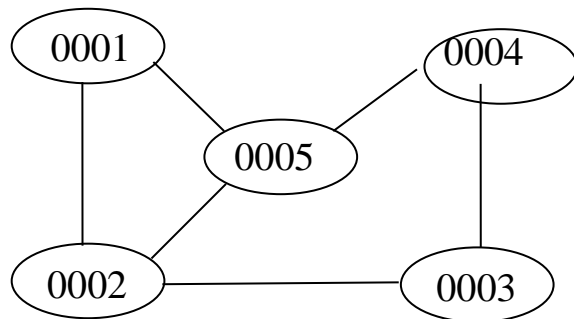
- (1) 在 Dos 界面输出两个城市间所有简单路径；
- (2) 在文件中输出两个城市间所有简单路径。

### 4. 功能

- (1) 创建一个空无向图；
- (2) 输入城市个数（顶点数）、高速公路条数（边数）和点和边的信息，用无向图表示高速公路网；
- (3) 获取两个顶点之间，除了起点和终点之外，其他顶点均不同的路径，并将所有路径输出到 Dos 界面和文件中。

### 5. 测试样例设计

- (1) 测试测试起点和终点不同的情况。



输入：

5

6

顶点信息：0001 0002 0003 0004 0005

边信息

0001 0002

0001 0005

0002 0003

0002 0005

0003 0004

0004 0005

始末顶点信息 0005 0001

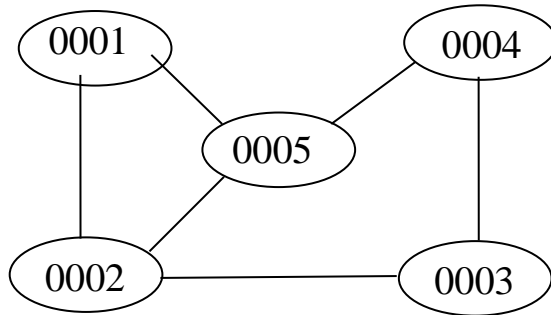
文件输出：

0005->0001

0005->0002->0001

0005->0004->0003->0002->0001

(2)



输入:

5

6

顶点信息: 0001 0002 0003 0004 0005

边信息:

0001 0002

0001 0005

0002 0003

0002 0005

0003 0004

0004 0005

始末顶点信息: 0002 0004

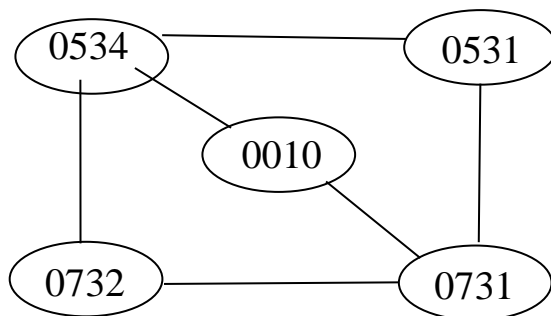
输出:

0002->0001->0005->0004

0002->0003->0004

0002->0005->0004

(3) 测试多条路径经过的城市没有重合的情况。



输入:

5

6

顶点信息: 0731 0732 0010 0531 0534

边信息:

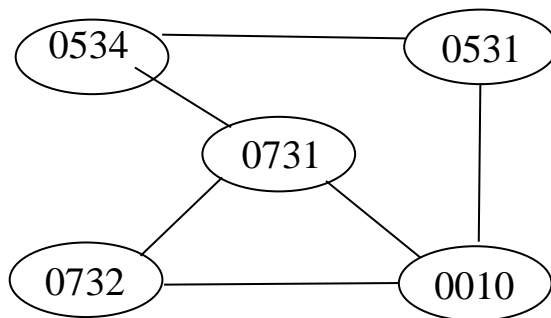
0731 0732

0731 0010  
 0731 0531  
 0732 0534  
 0010 0534  
 0531 0534  
 始末顶点信息: 0731 0534

输出:

0731->0732->0534  
 0731->0010->0534  
 0731->0531->0534

(4) 测试多条路径经过的城市有重合的情况。



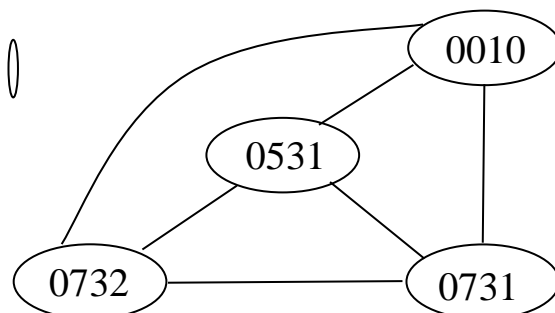
输入:

5  
 6  
 顶点信息: 0731 0732 0010 0531 0534  
 边信息:  
 0731 0732  
 0731 0010  
 0731 0534  
 0732 0010  
 0010 0531  
 0531 0534  
 始末顶点: 0731 0534

输出:

0731->0732->0010->0531->0534  
 0731->0010->0531->0534  
 0731->0534

(5)



输入:

4  
6  
顶点信息: 0731 0732 0010 0531  
边信息:  
0731 0732  
0731 0010  
0731 0531  
0732 0531  
0010 0531  
0732 0010  
始末顶点信息: 0732 0010

输出:

0732->0731->0010  
0732->0731->0531->0010  
0732->0531->0731->0010  
0732->0531->0010  
0732->0010

## 二、概要设计

### 1. 抽象数据类型

(1) **数据对象 R:** 图是由一个顶点集合  $V$  和一个弧集  $E$  构成的数据结构。  
 $Graph=(V, E)$  顶点的数据域和边的权课用于存储数据元素。

(2) **数据关系:**  $G$ 。  $VR=\{\langle v, w \rangle | v, w \in V \text{ 且 } P(v, w) \in E\}$ ,  $\langle v, w \rangle$  表示从  $v$  到  $w$  的一条弧, 并称  $v$  为弧头,  $w$  为弧尾。谓词  $P(v, w)$  定义了弧  $\langle v, w \rangle$  的意义或信息。  
复杂的非线性结构, 在图结构中, 每个元素都可以有零个或多个前驱, 也可以有零个或多个后继, 也就是说, 元素之间的关系是任意的。

### (3) 基本操作:

```
// 邻接表
class ListUDG
{
private: // 内部类
    // 邻接表中表对应的链表的顶点
    class ENode
    {
    public:
        int ivex;           // 该边所指向的顶点的位置
        int use;
        ENode *nextEdge;    // 指向下一条弧的指针
    };

    // 邻接表中表的顶点
    class VNode
    {
    public:
```

```

        string data;           // 顶点信息
        ENode *firstEdge;      // 指向第一条依附该顶点的
弧
    };

private: // 私有成员
    int mVexNum;               // 图的顶点的数目
    int mEdgNum;               // 图的边的数目
    VNode mVexs[MAX];
    ofstream outfile;
    char *fileName;

public:
    // 创建邻接表对应的图(自己输入)
    ListUDG();
    // // 创建邻接表对应的图(用已提供的数据)
    // ListUDG(char vexs[], int vlen, char edges[][2], int
elen);
    ~ListUDG();

    // 深度优先搜索遍历图
    void DFS();
    // 广度优先搜索（类似于树的层次遍历）
    void BFS();
    // 打印邻接表图
    void print();
    // 获取所有简单路径
    void getAll();

private:
    // 读取一个输入字符
    string readString();
    // 返回 ch 的位置
    int getPosition(string ch);
    // 深度优先搜索遍历图的递归实现
    void DFS(int i, int *visited);
    // 深度优先搜索遍历简单路径的递归实现
    void DFS(int i, int end, int *visited, vector<int> v);
    // 将 node 节点链接到 list 的最后
    void linkLast(ENode *list, ENode *node);

    void write(vector<int> v);

    int find(int value, vector<int> v) {

```

```

        for(int i=0;i<v.size();i++){
            if(value == v[i])
                return i;
        }

        return -1;
    }

};

```

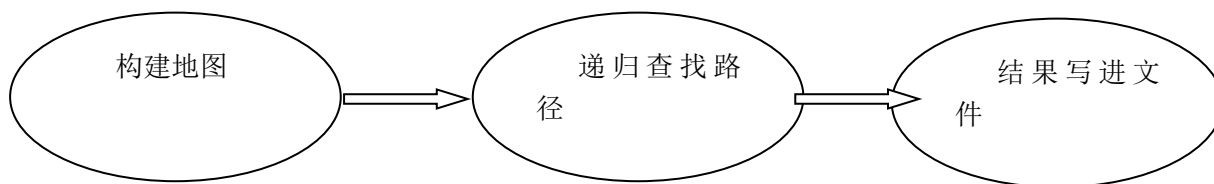
## 2. 算法的基本思想

getAll() 函数算法:

从起点开始, 每访问一个结点, 将其存入向量 path, 标记值 Mark 加一, 依次访问其相邻顶点, 如果被访问的顶点不是终点且在此之前的路径上已经访问过, 或者该顶点没有邻接顶点了, 则返回路径上的上一个顶点, 并将上一个顶点的标记取消, 即将 Mark 置为 UNVISITED, 继续查找其相邻顶点。若被访问的顶点是终点, 且路径长度大于 0, 则输出向量 path 中的所有顶点, 构成一条简单路径, 然后继续用 DFS 查找其余顶点。当找到路径时, 将 flag 标记为 1, 函数返回 flag, 以判断是否存在简单路径。

## 3. 程序的流程

程序有 3 个模块组成:



## 三、详细设计

### 1. 物理数据类型

使用邻接表实现图的 ADT, 顶点用连续的存储空间数组存储, 每个顶点分别作为一个链表的头结点, 其相邻顶点用离散的存储空间链表存储, 每建立一条边添加一个结点。

```

// 邻接表
class ListUDG
{
    private: // 内部类
        // 邻接表中表对应的链表的顶点
        class ENode
        {
        public:
            int ivex;           // 该边所指向的顶点的位置
            int use;
            ENode *nextEdge;    // 指向下一条弧的指针

```

```

};

// 邻接表中表的顶点
class VNode
{
public:
    string data;           // 顶点信息
    ENode *firstEdge;     // 指向第一条依附该顶点的弧
};

private: // 私有成员
    int mVexNum;           // 图的顶点的数目
    int mEdgNum;           // 图的边的数目
    VNode mVexs[MAX];
    ofstream outfile;
    char *fileName;

public:
    // 创建邻接表对应的图(自己输入)
    ListUDG();
    // // 创建邻接表对应的图(用已提供的数据)
    // ListUDG(char vexs[], int vlen, char edges[][2], int elen);
    ~ListUDG();

    // 深度优先搜索遍历图
    void DFS();
    // 广度优先搜索（类似于树的层次遍历）
    void BFS();
    // 打印邻接表图
    void print();
    // 获取所有简单路径
    void getAll();

private:
    // 读取一个输入字符
    string readString();
    // 返回 ch 的位置
    int getPosition(string ch);
    // 深度优先搜索遍历图的递归实现
    void DFS(int i, int *visited);
    // 深度优先搜索遍历简单路径的递归实现
    void DFS(int i, int end, int *visited, vector<int> v);
    // 将 node 节点链接到 list 的最后
    void linkLast(ENode *list, ENode *node);

```



```

void write(vector<int> v);

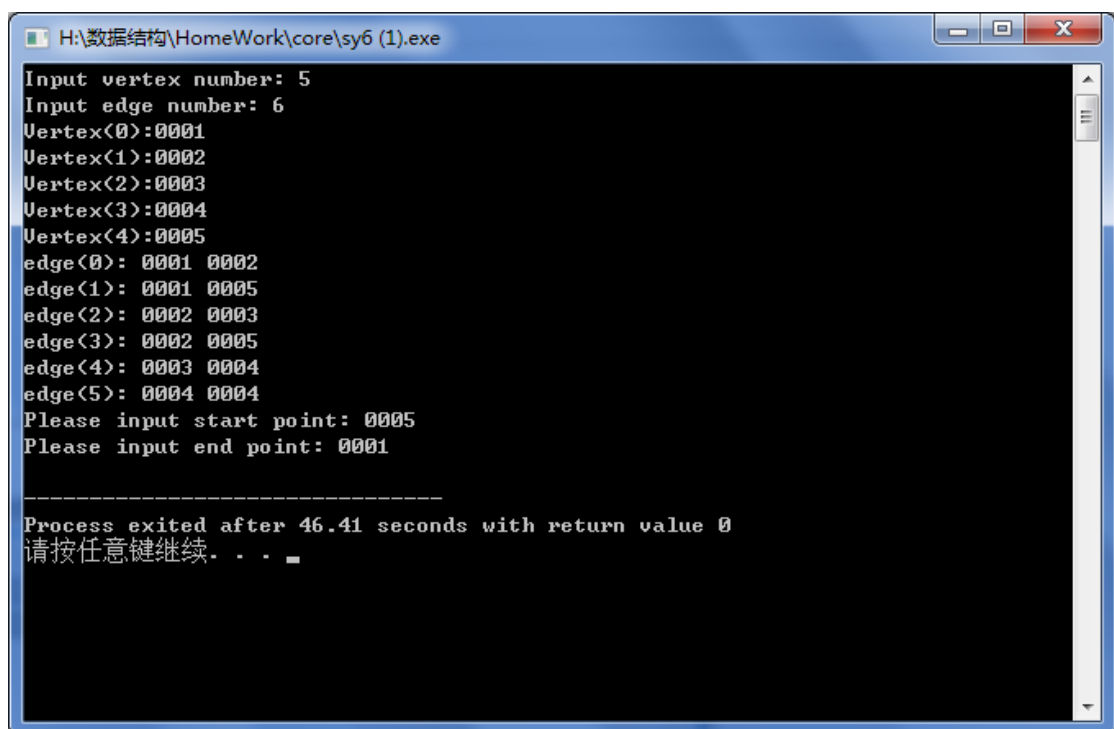
int find(int value, vector<int> v) {
    for(int i=0; i<v.size(); i++) {
        if(value == v[i])
            return i;
    }

    return -1;
}

};

```

输入和输出的格式



```

H:\数据结构\HomeWork\core\sy6 (1).exe
Input vertex number: 5
Input edge number: 6
Vertex<0>:0001
Vertex<1>:0002
Vertex<2>:0003
Vertex<3>:0004
Vertex<4>:0005
edge<0>: 0001 0002
edge<1>: 0001 0005
edge<2>: 0002 0003
edge<3>: 0002 0005
edge<4>: 0003 0004
edge<5>: 0004 0004
Please input start point: 0005
Please input end point: 0001

-----
Process exited after 46.41 seconds with return value 0
请按任意键继续. . .

```

## 2. 算法的具体步骤

### 一、 找全部简单路径

```

void ListUDG::getAll() {
    string start, end;
    cout<<"Please input start point: ";
    cin>>start;
    cout<<"Please input end point: ";
    cin>>end;

    int s = getPosition(start);
    int e = getPosition(end);
}

```

```

vector<int> v;
int visited[MAX];          // 顶点访问标记
// 初始化所有顶点都没有被访问
for (int i = 0; i < mVexNum; i++)
    visited[i] = 0;

v.push_back(s);
visited[s] = 1;
while(!v.empty()) {
    if(v[v.size()-1] == e) {
        write(v);
//        for(int i=0;i<v.size();i++) {
//            cout<<v[i]<<" ";
//        }
//        cout<<endl;
        visited[e] = 0;
        v.pop_back();
    } else {
        int n = v[v.size()-1];
        ENode * node = mVexs[n].firstEdge;
        while(node!=NULL) {
            if(visited[node->ivex] == 0 &&
find(node->ivex, v)==-1 && node->use == 0) {
                v.push_back(node->ivex);
                node->use = 1;
                visited[node->ivex] = 1;
                break;
            }
            node = node->nextEdge;
        }
        if(node == NULL) {
            visited[n] = 0;
            ENode * node = mVexs[n].firstEdge;
            while(node!=NULL) {
                if(find(node->ivex, v)==-1) { //恢复
顶点状态
                    node->use = 0;
                    visited[node->ivex] == 0;
                }
                node = node->nextEdge;
            }
            v.pop_back();
        }
    }
}

```

```
}
```

```
}
```

## 二、 写入文件

```
void ListUDG::write(vector<int> v) {
    ofstream outfile;
    outfile.open("record.txt", ios::app);
    for(int i=0; i<v.size(); i++) {
        if(i==0)
            outfile << mVexs[v[i]].data;
        else
            outfile <<"-"<< mVexs[v[i]].data;
    }

    outfile << endl;
}
```

## 3. 算法的时空分析

### 时间复杂度

- (1) 结构模块：定义邻接表图的对象：时间复杂度  $\theta(1)$ ;
- (2) 输入模块：循环输入各顶点信息：时间复杂度  $\theta(v)$ ;  
循环输入各边信息：时间复杂度  $\theta(e)$ ;
- (3) 处理模块：  
构建邻接表：时间复杂度  $\theta(v*e)$ ,  
查找简单路径：时间复杂度为 DFS 递归算法的复杂度  $\theta(v+e)$ 。

### 空间复杂度

空间复杂度为  $\theta(v*e)$ 。

结构性开销为边数  $e$  (每多一条边都要建立一个新的链表结点)。

## 四、调试分析

### 1. 调试方案设计

- (1) 调试结构模块：  
构建一个空无向图，调用邻接表输出函数，看是否为空。
- (2) 调试输入模块：  
调用构图函数输入参数信息，创建一个无向图后分别输出其邻接表信息，观察是否正确。
- (3) 调试处理模块：  
输入几组不同情况下的无向图数据和查找数据，调用查找简单路径函数，观察 Dos 界面和文件中输出的简单路径结果是否正确。

### 2. 调试过程和结果，及分析

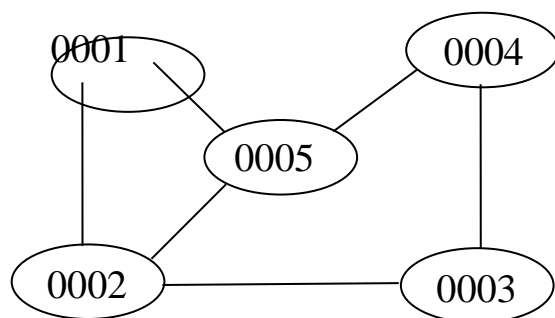
- (1) 调试结构模块：创建邻接表后，判断顶点数和边数，均为 0，且可以调用图的各个成员函数。
- (2) 调试输入模块：从输出的结果，输入的顶点和边都成功构建到图中，

输入操作正确。

(3) 调试处理模块：各种特殊类型的图（包含有向图和无向图），构图操作和基本操作的输出结果都正确。输出的邻接表与所创建的图相符，说明输出函数的操作正确。输出的路径数量和结果都正确。

## 五、测试结果

(1) 测试测试起点和终点不同的情况：结果正确输出了各条简单路径，每条路径各顶点均不相同。



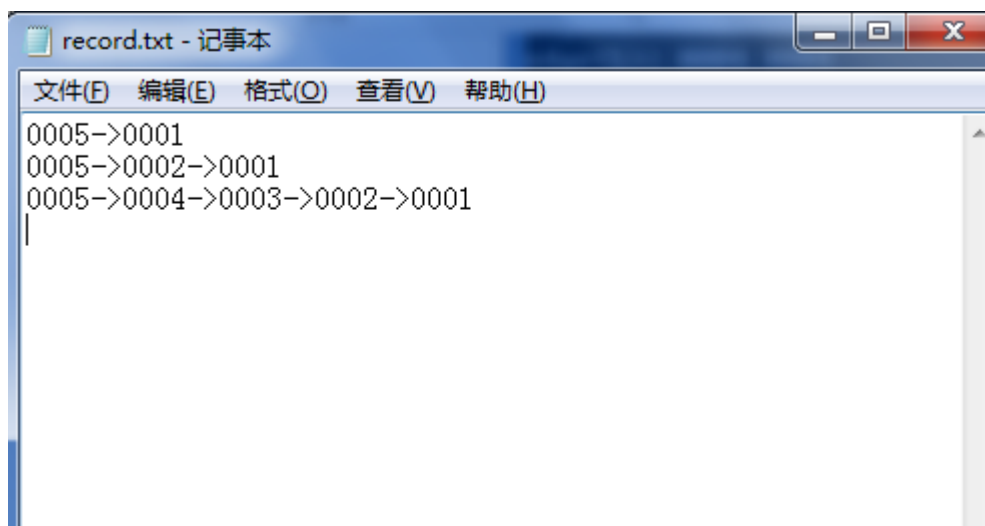
Dos 界面：

```

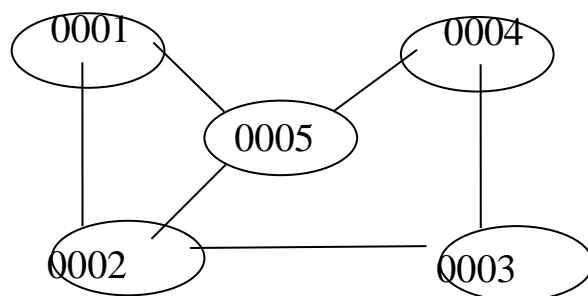
H:\数据结构\HomeWork\core\sy6 (1).exe
Input vertex number: 5
Input edge number: 6
Vertex(0):0001
Vertex(1):0002
Vertex(2):0003
Vertex(3):0004
Vertex(4):0005
edge(0): 0001 0002
edge(1): 0001 0005
edge(2): 0002 0003
edge(3): 0002 0005
edge(4): 0003 0004
edge(5): 0004 0004
Please input start point: 0005
Please input end point: 0001

-----
Process exited after 46.41 seconds with return value 0
请按任意键继续. . .
    
```

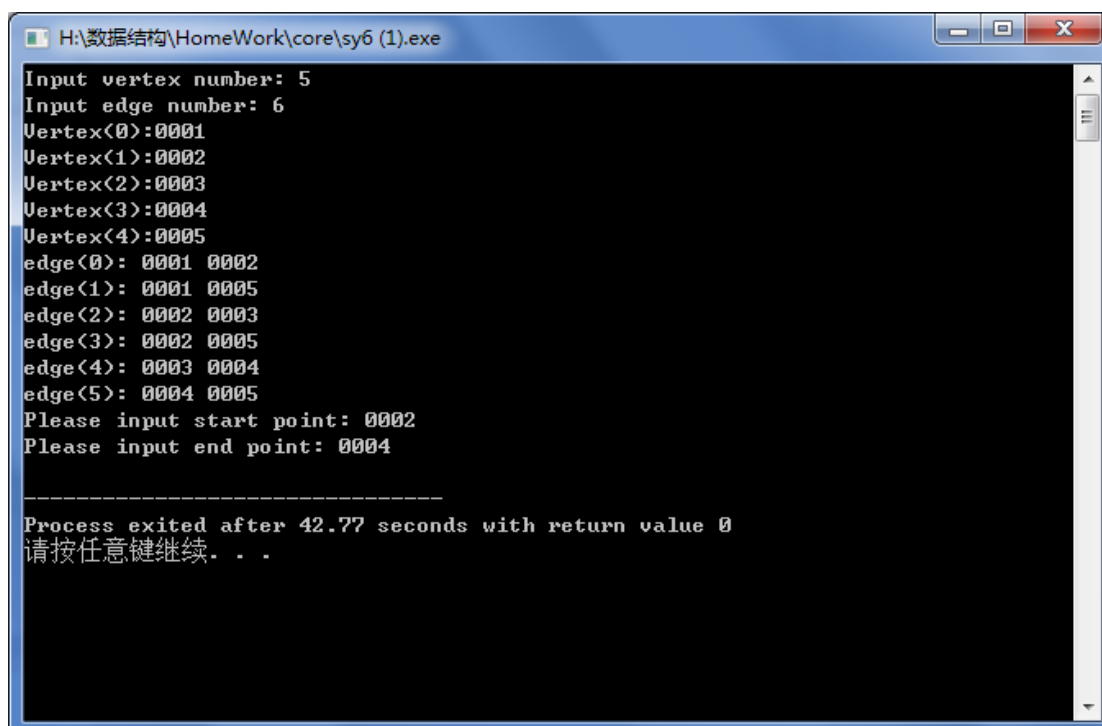
文件：



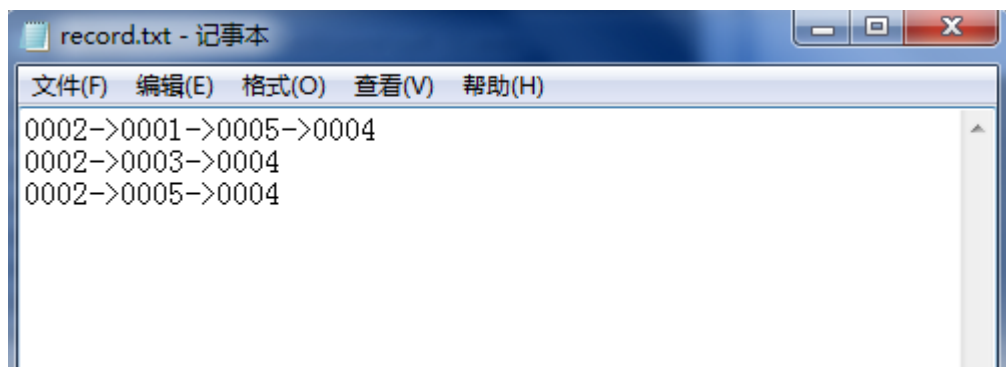
(2)



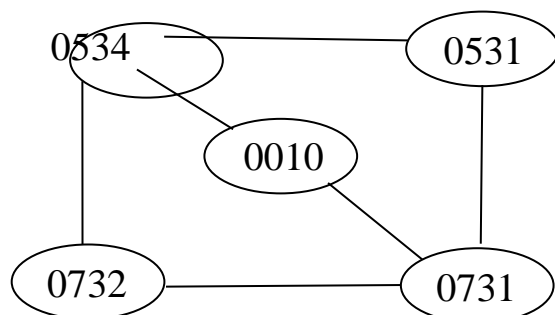
Dos 界面:



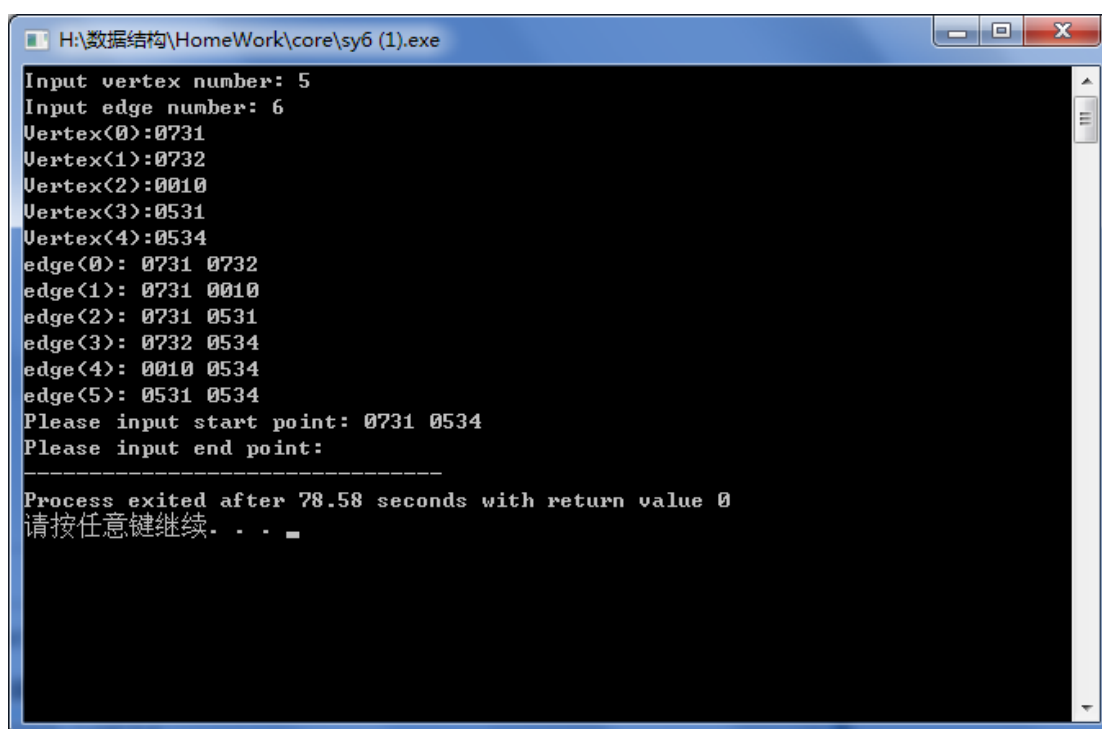
文件:



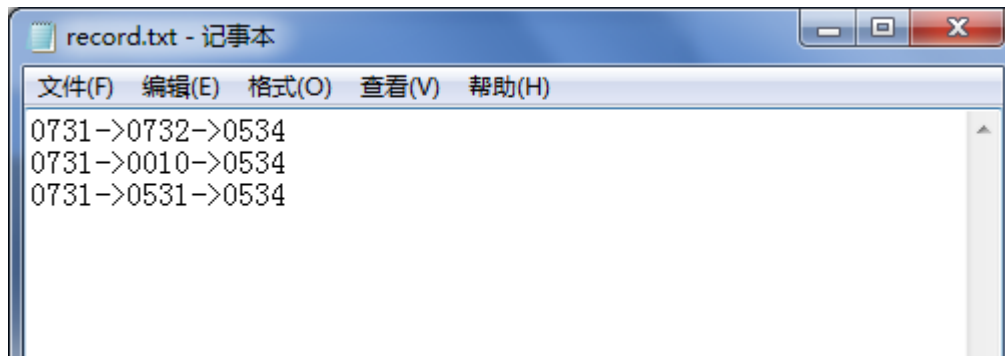
(3)测试多条路径经过的城市没有重合的情况：输出的各条路径均为简单路径，且输出的所有顶点除起点和终点外无重复。



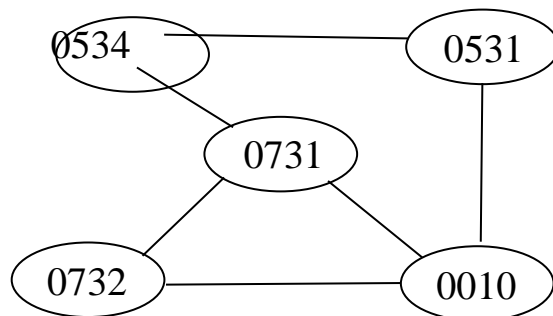
Dos 界面:



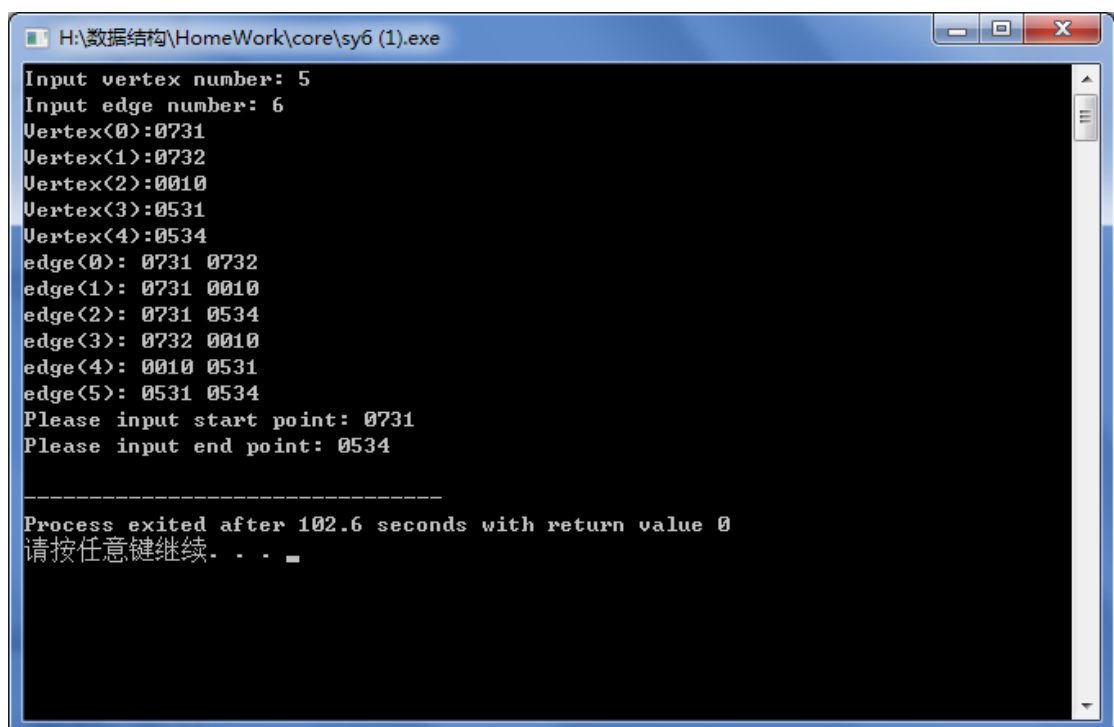
文件:



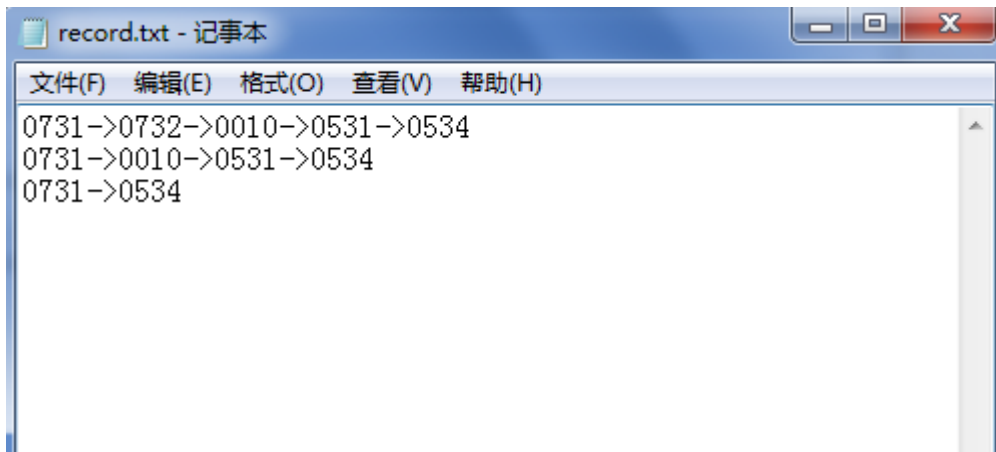
(4)测试多条路径经过的城市有重合的情况：同一顶点在不同路径中可以多次被输出，但在同一条路径中不能重复。



Dos 界面:

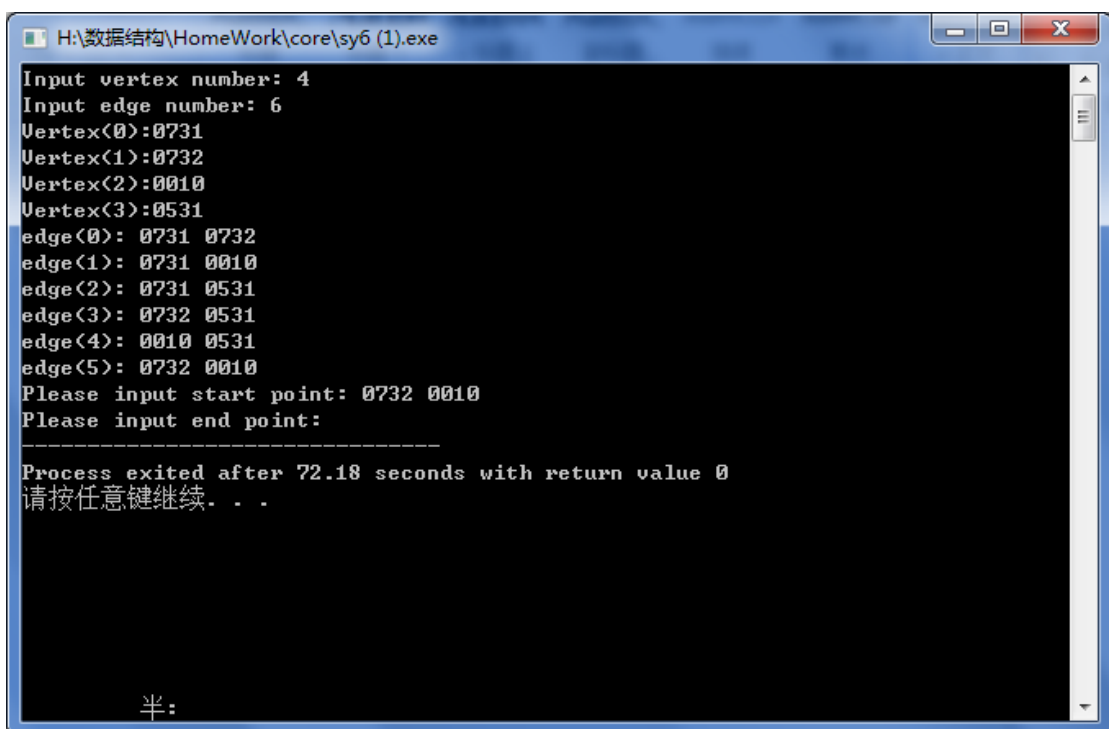


文件:



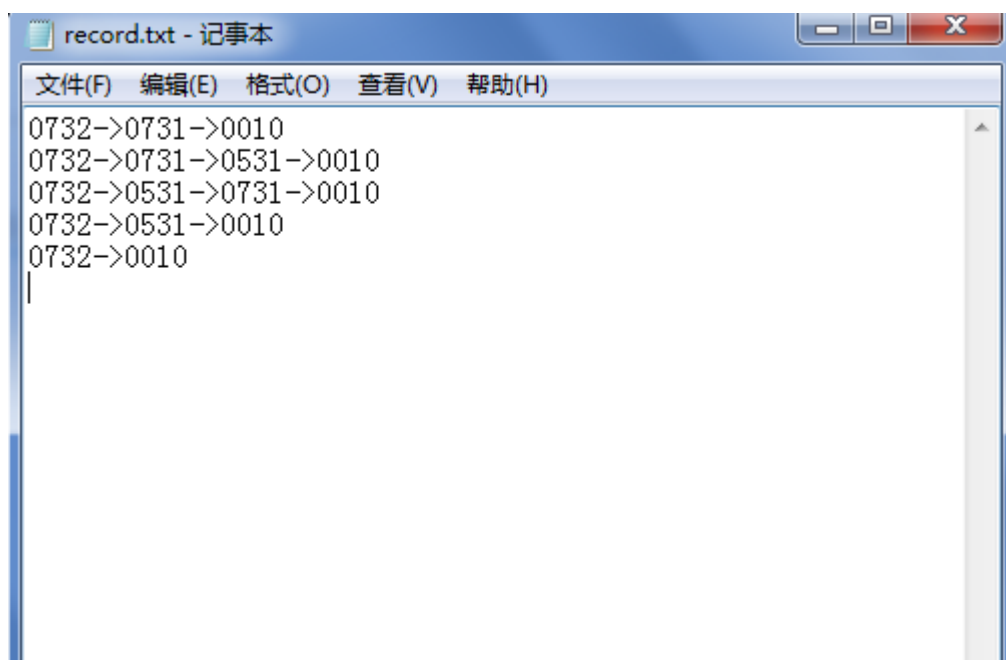
(5)

Dos 界面:



文件:





```
record.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
0732->0731->0010
0732->0731->0531->0010
0732->0531->0731->0010
0732->0531->0010
0732->0010
|
```

## 六、实验心得

本次实验是基于图的物理实现，求解简单路径，算法本身不是很复杂。

在做实验的过程中，遇到了一点问题，一开始没有看全需求，所以非常顺利地完成了实验代码。在设计测试数据时重新读了一遍问题描述和需求分析，发现起点和终点可以重复。在解决这个问题时还遇到了一些波折，一开始想在找路径函数中做修改，区分起点和终点是否重复的情况，但是基于 DFS 的简单路径求解过程是一个递归函数，因此在递归调用的函数中起点是不断变化的。考虑后想到，可以在标记值上做改变，起点和终点与其他顶点的唯一不同点就是可以被遍历两次，而第一次标记到的一定是起点，因此只需要让起点的标记值比其他顶点小 1，标记时标记遍历的次数即可。这启发了我们，想问题不一定要太直接，可以通过改变其他操作实现想要的功能。

另外，文件输出也是之前实验中没有用过的知识。

## 七、用户使用说明

结果的输出是在同目录下的 record.txt 文件