# 《数据结构与算法实验》第 11 次实验

学院： 专业： 年级：
姓名： 学号： 日期： 2022 年 6 月 7 日

## 第一部分 验证实验

### 一、 实验目的

1. 掌握图的逻辑结构，验证邻接矩阵存储结构及遍历操作。

2. 掌握并验证邻接表存储结构及遍历操作。

### 二、 实验内容

1. 邻接矩阵的实现（参考实验书 p219）

   • 建立无向图的邻接矩阵存储。

   • 对建立的无向图，进行深度优先遍历（递归算法+非递归算法）。

   • 对建立的无向图，进行广度优先遍历

2. 邻接表的实现（参考实验书 p222）

   • 建立一个有向图的邻接表存储结构。

   • 对建立的有向图，进行深度优先遍历（递归算法+非递归算法）。

   • 对建立的有向图，进行广度优先遍历。

### 三、 设计编码

#### 1.理论知识

定义实现邻接矩阵的无向图类 MGraph，包括题目要求的建立、深度优先遍历、广度优先遍历等基本操作。

定义实现邻接表的有向图类 ALGraph，包括题目要求的建立、深度优先遍历、广度优先遍历等基本操作。

## 2. 代码实现

(1) MGraph. h

```
#ifndef MGraph_H
#define MGraph_H
const int MaxSize = 10;

template <class T>
class MGraph
{
    private:
        T vertex[MaxSize];
        int arc[MaxSize][MaxSize];
        int vertexNum, arcNum;
    public:
        MGraph(T a[], int n, int e);
        ~MGraph();
        void DFSTraverse(int v);
        void BFSTraverse(int v);
};

#endif
```

(1) MGraph. cpp

```
#include <iostream>
#include "MGraph.h"
using namespace std;

extern int visited[];
template <class T>
MGraph<T>::MGraph(T a[], int n, int e)
{
    int i, j, k;
    vertexNum = n, arcNum = e;
    for ( i = 0; i < vertexNum; i++)
        vertex[i] = a[i];
    for ( i = 0; i < vertexNum; i++)
        for ( j = 0; j < vertexNum; j++)
            arc[i][j] = 0;
    for ( k = 0; k < arcNum; k++)
    {
        cout<<"请输入边的两个顶点的序号：";
        cin>>i>>j;
        arc[i][j] = 1;
        arc[j][i] = 1;
    }
}

template <class T>
MGraph<T>::~MGraph()
{
    //empty
}

template <class T>
void MGraph<T>::DFSTraverse(int v)
{
    cout<<vertex[v];
    visited[v] = 1;
    for (int j = 0; j < vertexNum; j++)
        if (arc[v][j] == 1 && visited[j] == 0)
            DFSTraverse(j);
}

template <class T>
void MGraph<T>::BFSTraverse(int v)
{
    int Q[MaxSize];
    int front = -1, rear = -1;
    cout<<vertex[v];
    visited[v] = 1;
    Q[++rear] = v;
    while (front != rear)
    {
        v = Q[++front];
        for (int j = 0; j < vertexNum; j++)
            if (arc[v][j] == 1 && visited[j] == 0)
            {
                cout<<vertex[j];
                visited[j] = 1;
                Q[++rear] = j;
            }
    }
}
```

(1)MGraph_main.cpp

```
#include <iostream>
#include "MGraph.cpp"
using namespace std;
int visited[MaxSize] = {0};

int main()
{
    char ch[] = {'A','B','C','D','E','F'};
    MGraph<char> MG(ch, 6, 6);
    for (int i = 0; i < MaxSize; i++)
        visited[i] = 0;
    cout<<"深度的优先遍历序列是: ";
    MG.DFSTraverse(0);
    cout<<endl;
    for (int i = 0; i < MaxSize; i++)
        visited[i] = 0;
    cout<<"广度的优先遍历序列是: ";
    MG.BFSTraverse(0);
    cout<<endl;
    return 0;
}
```

(2) ALGraph.h

```
#ifndef ALGraph_H
#define ALGraph_H
const int MaxSize = 10;

struct ArcNode
{
    int adjvex;
    ArcNode * next;
};

template <class T>
struct VertexNode
{
    T vertex;
    ArcNode * firstedge;
};

template <class T>
class ALGraph
{
    private:
        VertexNode<T> adjlist[MaxSize];
        int vertexNum, arcNum;
    public:
        ALGraph(T a[], int n, int e);
        ~ALGraph();
        void DFSTraverse(int v);
        void BFSTraverse(int v);
};

#endif
```

(2) ALGraph.cpp

```
#include <iostream>
#include "ALGraph.h"
using namespace std;
extern int visited[];

template <class T>
ALGraph<T>::ALGraph(T a[], int n, int e)
{
    ArcNode * s;
    int i, j, k;
    vertexNum = n;
    arcNum = e;
    for ( i = 0; i < vertexNum; i++)
    {
        adjlist[i].vertex = a[i];
        adjlist[i].firstedge = NULL;
    }
    for ( k = 0; k < arcNum; k++)
    {
        cout<<"请输入边的两个顶点的序号：";
        cin>>i>>j;
        s = new ArcNode;
        s -> adjvex = j;
        s -> next = adjlist[i].firstedge;
        adjlist[i].firstedge = s;
    }

}

template <class T>
```

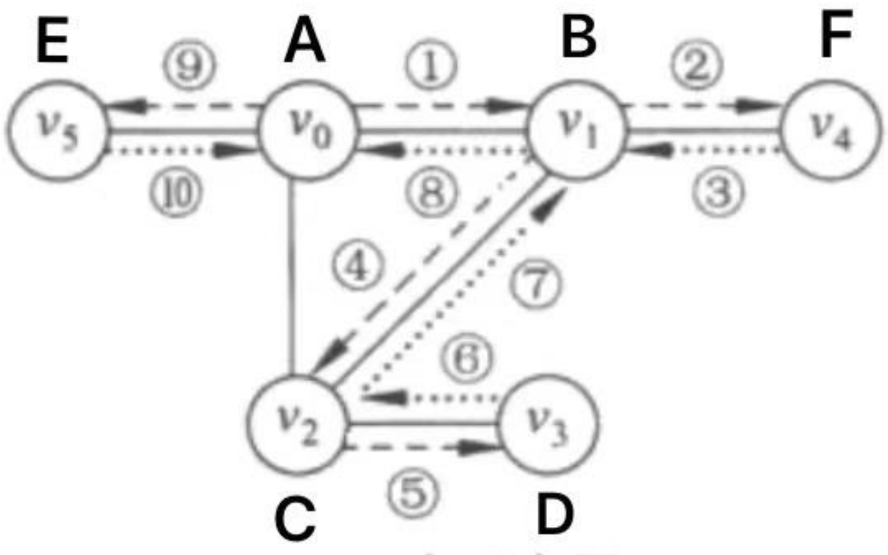
```
ALGraph<T>::~ALGraph()
{
    ArcNode * p;
    for (int i = 0; i < vertexNum; i++)
    {
        p = adjlist[i].firstedge;
        while (p != NULL)
        {
            adjlist[i].firstedge = p -> next;
            delete p;
            p = adjlist[i].firstedge;
        }
    }
}

template <class T>
void ALGraph<T>::DFSTraverse(int v)
{
    ArcNode * p = NULL;
    int j;
    cout<<adjlist[v].vertex;
    visited[v] = 1;
    p = adjlist[v].firstedge;
    while (p != NULL)
    {
        j = p -> adjvex;
        if (visited[j] == 0)  DFSTraverse(j);
        p = p -> next;
    }
}

template <class T>
void ALGraph<T>::BFSTraverse(int v)
{
    int Q[MaxSize];
    int front = -1, rear = -1;
    ArcNode * p;
    cout<<adjlist[v].vertex;
    visited[v] = 1;
    Q[++rear] = v;
    while (front != rear)
    {
        v = Q[++front];
        p = adjlist[v].firstedge;
        while (p != NULL)
        {
            int j = p -> adjvex;
            if (visited[j] == 0)
            {
                cout<<adjlist[j].vertex;
                visited[j] = 1;
                Q[++rear] = j;
            }
            p = p -> next;
        }
    }
}
```

(2)ALGraph_main.cpp

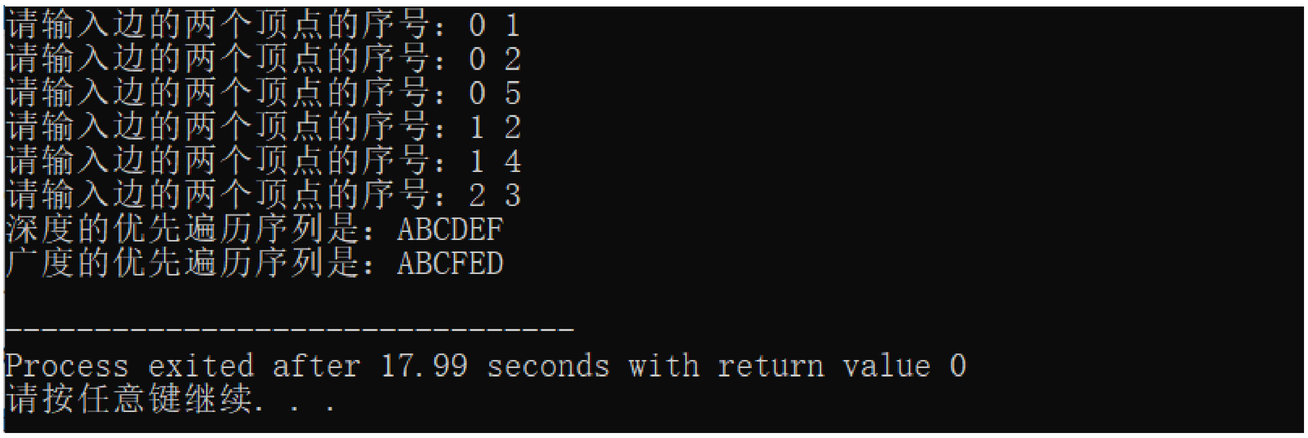
```
#include <iostream>
#include"ALGraph.cpp"
using namespace std;
int visited[MaxSize] = {0};

int main()
{
    char ch[] = {'A','B','C','D','E'};
    int i;
    ALGraph<char> ALG(ch, 5, 7);
    for (int i = 0; i < MaxSize; i++)  visited[i] = 0;
    cout<<"深度优先遍历序列是: ";
    ALG.DFSTraverse(0);
    cout<<endl;
    for (int i = 0; i < MaxSize; i++)  visited[i] = 0;
    cout<<"广度优先遍历序列是: ";
    ALG.BFSTraverse(0);
    cout<<endl;
    return 0;
}
```

## 四、 运行与测试

1. 邻接矩阵的实现（实验样图如下）

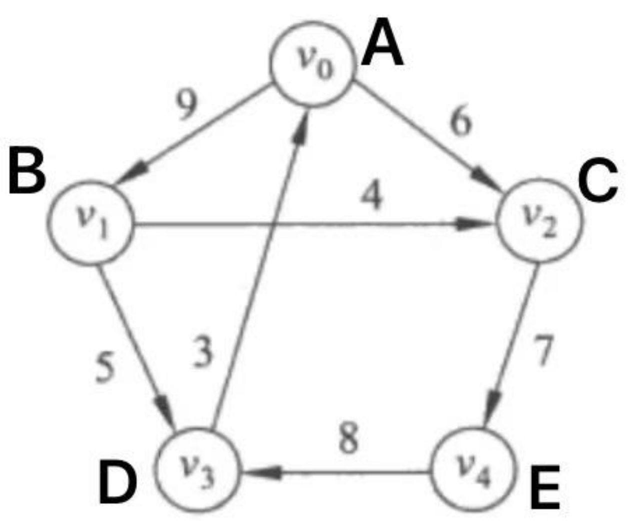


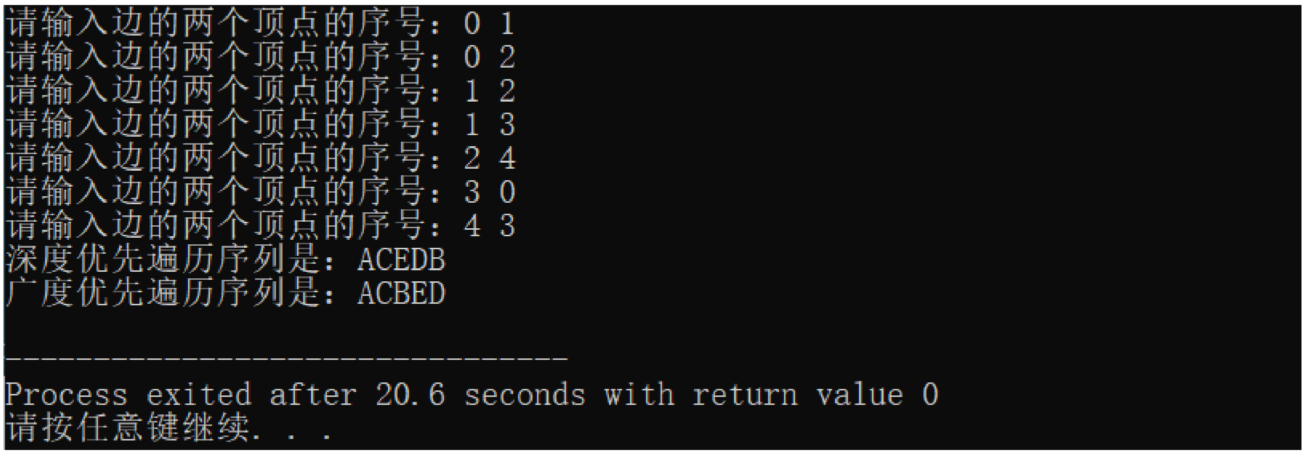
(a) 一个无向图



```
请输入边的两个顶点的序号: 0 1
请输入边的两个顶点的序号: 0 2
请输入边的两个顶点的序号: 0 5
请输入边的两个顶点的序号: 1 2
请输入边的两个顶点的序号: 1 4
请输入边的两个顶点的序号: 2 3
深度的优先遍历序列是: ABCDEF
广度的优先遍历序列是: ABCFED

--------------------------------
Process exited after 17.99 seconds with return value 0
请按任意键继续. . .
```

2.邻接表的实现（实验样图如下）




```
请输入边的两个顶点的序号: 0 1
请输入边的两个顶点的序号: 0 2
请输入边的两个顶点的序号: 1 2
请输入边的两个顶点的序号: 1 3
请输入边的两个顶点的序号: 2 4
请输入边的两个顶点的序号: 3 0
请输入边的两个顶点的序号: 4 3
深度优先遍历序列是: ACEDB
广度优先遍历序列是: ACBED

--------------------------------
Process exited after 20.6 seconds with return value 0
请按任意键继续. . .
```

# 第二部分 设计实验

## 一、实验目的

通过共计 6 个实验, 实现无向图的邻接表和邻接矩阵的互相转换，求出度为 0 的顶点个数，求泥邻接表，求路径等，熟悉邻接矩阵和邻接表的操作。

## 二、实验内容

**1. 无向图邻接矩阵转换为邻接表**：参考实验书 p102

利用前两个验证实验构造的数据结构进行实验。

使用的理论知识包括图的邻接表和邻接矩阵的存储结构以及无向图的邻接矩阵转换为邻接表的算法。

**2. 无向图邻接表转换为邻接矩阵**：参考实验书 p103

利用图的邻接表和邻接矩阵的存储结构，在邻接表上按顺序去每个边表 d 额结点，将邻接矩阵对应的单元设置为 1。

**3. 计算度为 0 的顶点个数**：参考实验书 p104

在有向图的邻接矩阵中，一行对应一个顶点，每行的非 0 元素的个数等于对应顶点的度。因此，当某行非零元素的个数为 0 时，则对应顶点的出度为 0。据此，从第一行开始，查找每行的非零元素个数是否为 0，若是则计数器加 1。

**4. 邻接表非递归深度优先遍历：**参考实验书 p104

• 需掌握图的邻接矩阵的存储结构和深度优先遍历的工作栈。

• 在邻接表的以深度优先遍历算法存储结构中，设计一个栈来模拟递归实现中系统设置的工作栈。

**5. 建立逆邻接表：**参考实验书 p104

在有向图中若邻接表中顶点 v 有邻接点 w，在逆邻接表中 w 一定有邻接点 v。由此得到本题算法思路：首先将 v 邻接表的表头结点 firstedge 域置空，然后逐行将表头结点的邻接点进行转化。

**6. 判断有向图的路径：**参考实验书 p105

分别基于深度优先和广度优先搜索编写算法，判断以邻接表存储的有向图中是否存在由顶点 vi 到顶点 vj 的路径（i≠j）。

## 三、 设计编码

### 1. 无向图邻接矩阵转换为邻接表

(1)MGraph.h

```
#ifndef MGraph_H
#define MGraph_H
//const int MaxSize = 10;

template <class T>
class MGraph
{
    private:
        T vertex[MaxSize];
        int arc[MaxSize][MaxSize];
        int vertexNum, arcNum;
    public:
        MGraph(T a[], int n, int e);
        ~MGraph();
        void DFSTraverse(int v);
        void BFSTraverse(int v);
        void PrintMat();
        void getMat(int (&mat)[MaxSize][MaxSize]);
};

#endif
```

(2)MGraph.cpp

```
#include <iostream>
#include "MGraph.h"
using namespace std;

extern int visited[];

template <class T>
MGraph<T>::MGraph(T a[], int n, int e)
{
    int i, j, k;
    vertexNum = n, arcNum = e;
    for ( i = 0; i < vertexNum; i++)
        vertex[i] = a[i];
    for ( i = 0; i < vertexNum; i++)
        for ( j = 0; j < vertexNum; j++)
            arc[i][j] = 0;
    for ( k = 0; k < arcNum; k++)
    {
        cout<<"请输入边的两个顶点的序号：";
        cin>>i>>j;
        arc[i][j] = 1;
        arc[j][i] = 1;
    }
}

template <class T>
MGraph<T>::~MGraph()
{
    //empty
}

template <class T>
void MGraph<T>::DFSTraverse(int v)
{
    cout<<vertex[v];
    visited[v] = 1;
    for (int j = 0; j < vertexNum; j++)
        if (arc[v][j] == 1 && visited[j] == 0)
            DFSTraverse(j);
}

template <class T>
void MGraph<T>::BFSTraverse(int v)
{
    int Q[MaxSize];
    int front = -1, rear = -1;
    cout<<vertex[v];
    visited[v] = 1;
    Q[++rear] = v;
    while (front != rear)
    {
        v = Q[++front];
        for (int j = 0; j < vertexNum; j++)
            if (arc[v][j] == 1 && visited[j] == 0)
            {
                cout<<vertex[j];
                visited[j] = 1;
                Q[++rear] = j;
            }
    }
}

template <class T>
void MGraph<T>::PrintMat()
{
    int i, j;
    cout<<"\n\nMartix Graph:"<<endl;
    for ( i = 0; i < vertexNum; i++)
    {
        for ( j = 0; j < vertexNum; j++)
        {
            cout<<setw(5)<<arc[i][j];
        }
        cout<<endl;
    }
}

template <class T>
void MGraph<T>::getMat(int (&mat)[MaxSize][MaxSize])
```

```
{
    for (int i = 0; i < MaxSize; i++)
    {
        for (int j = 0; j < MaxSize; j++)
        {
            mat[i][j] = arc[i][j];
        }
    }
}
```

(3) ALGraph.h

```
#ifndef ALGraph_H
#define ALGraph_H
const int MaxSize = 10;

struct ArcNode
{
    int adjvex;
    ArcNode * next;
};

template <class T>
struct VertexNode
{
    T vertex;
    ArcNode * firstedge;
};

template <class T>
class ALGraph
{
    private:
        VertexNode<T> adjlist[MaxSize];
        int vertexNum, arcNum;
    public:
        ALGraph(T a[], int n, int e);
        ALGraph(T a[], int n, int e, int arc[MaxSize][MaxSize]);
        ~ALGraph();
        void DFSTraverse(int v);
        void BFSTraverse(int v);
};

#endif
```

(4) ALGraph. cpp

```
#include <iostream>
#include "ALGraph.h"
using namespace std;
extern int visited[];

template <class T>
ALGraph<T>::ALGraph(T a[], int n, int e)
{
    ArcNode * s;
    int i, j, k;
    vertexNum = n;
    arcNum = e;
    for ( i = 0; i < vertexNum; i++)
    {
        adjlist[i].vertex = a[i];
        adjlist[i].firstedge = NULL;
    }
    for ( k = 0; k < arcNum; k++)
    {
        cout<<"请输入边的两个顶点序号：";
        cin>>i>>j;
        s = new ArcNode;
        s -> adjvex = j;
        s -> next = adjlist[i].firstedge;
        adjlist[i].firstedge = s;
    }
}

template <class T>
ALGraph<T>::ALGraph(T a[], int n, int e, int arc[MaxSize][MaxSize])
{
    arcNum = e;
    vertexNum = n;
    for (int i = 0; i < vertexNum; i++)
    {
        VertexNode<T> tempvertex;
        tempvertex.vertex = a[i];
        tempvertex.firstedge = NULL;
        adjlist[i] = tempvertex;
    }
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            if (j != i && arc[i][j] != 0)
            {
                ArcNode * s = new ArcNode;
                s -> adjvex = j;
                s -> next = adjlist[i].firstedge;
                adjlist[i].firstedge = s;
            }
        }
    }
}

template <class T>
ALGraph<T>::~ALGraph()
{
    ArcNode * p;
    for (int i = 0; i < vertexNum; i++)
    {
        p = adjlist[i].firstedge;
        while (p != NULL)
        {
            adjlist[i].firstedge = p -> next;
            delete p;
            p = adjlist[i].firstedge;
        }
    }
}

template <class T>
void ALGraph<T>::DFSTraverse(int v)
{
    ArcNode * p = NULL;
    int j;
    cout<<adjlist[v].vertex;
    visited[v] = 1;
    p = adjlist[v].firstedge;
    while (p != NULL)
    {
        j = p -> adjvex;
        if (visited[j] == 0)  DFSTraverse(j);
        p = p -> next;
    }
```

```
}

template <class T>
void ALGraph<T>::BFSTraverse(int v)
{
    int Q[MaxSize];
    int front = -1, rear = -1;
    ArcNode * p;
    cout<<adjlist[v].vertex;
    visited[v] = 1;
    Q[++rear] = v;
    while (front != rear)
    {
        v = Q[++front];
        p = adjlist[v].firstedge;
        while (p != NULL)
        {
            int j = p -> adjvex;
            if (visited[j] == 0)
            {
                cout<<adjlist[j].vertex;
                visited[j] = 1;
                Q[++rear] = j;
            }
            p = p -> next;
        }
    }
}
```

(5)Graph_main.cpp

```
#include <iostream>
#include <iomanip>
using namespace std;
#include "ALGraph.cpp"
#include "MGraph.cpp"
int visited[MaxSize] = {0};

int main()
{
    char ch[]={'A','B','C','D','E'};
    MGraph<char> MG(ch, 6, 6);
    MG.PrintMat();
    int mat[MaxSize][MaxSize] = {0};
    MG.getMat(mat);
    ALGraph<char> ALG(ch, 6, 6, mat);
    return 0;
}
```

2. 无向图邻接表转换为邻接矩阵：

(1) ALGraph. h

```
#ifndef ALGraph_H
#define ALGraph_H
const int MaxSize = 10;

struct ArcNode
{
    int adjvex;
    ArcNode * next;
};

template <class T>
struct VertexNode
{
    T vertex;
    ArcNode * firstedge;
};

template <class T>
class ALGraph
{
    private:
        VertexNode<T> adjlist[MaxSize];
        int vertexNum, arcNum;
    public:
        ALGraph(T a[], int n, int e);
        ALGraph(T a[], int n, int e, int arc[MaxSize][MaxSize]);
        ~ALGraph();
        void DFSTraverse(int v);
        void BFSTraverse(int v);
        void ListToMatrix();
};

#endif
```

(2) ALGraph. cpp

```
#include <iostream>
#include <iomanip>
#include "ALGraph.h"
using namespace std;
extern int visited[];

template <class T>
ALGraph<T>::ALGraph(T a[], int n, int e)
{
    ArcNode * s;
    int i, j, k;
    vertexNum = n;
    arcNum = e;
    for ( i = 0; i < vertexNum; i++)
    {
        adjlist[i].vertex = a[i];
        adjlist[i].firstedge = NULL;
    }
    for ( k = 0; k < arcNum; k++)
    {
        cout<<"请输入边的两个顶点的序号：";
        cin>>i>>j;
        s = new ArcNode;
        s -> adjvex = j;
        s -> next = adjlist[i].firstedge;
        adjlist[i].firstedge = s;
    }
}

template <class T>
ALGraph<T>::ALGraph(T a[], int n, int e, int arc[MaxSize][MaxSize])
{
    arcNum = e;
    vertexNum = n;
    for (int i = 0; i < vertexNum; i++)
    {
        VertexNode<T> tempvertex;
        tempvertex.vertex = a[i];
        tempvertex.firstedge = NULL;
        adjlist[i] = tempvertex;
    }
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            if (j != i && arc[i][j] != 0)
            {
                ArcNode * s = new ArcNode;
                s -> adjvex = j;
                s -> next = adjlist[i].firstedge;
                adjlist[i].firstedge = s;
            }
        }
    }
}

template <class T>
ALGraph<T>::~ALGraph()
{
    ArcNode * p;
    for (int i = 0; i < vertexNum; i++)
    {
        p = adjlist[i].firstedge;
        while (p != NULL)
        {
            adjlist[i].firstedge = p -> next;
            delete p;
            p = adjlist[i].firstedge;
        }
    }
}

template <class T>
void ALGraph<T>::DFSTraverse(int v)
{
    ArcNode * p = NULL;
    int j;
    cout<<adjlist[v].vertex;
    visited[v] = 1;
    p = adjlist[v].firstedge;
    while (p != NULL)
    {
        j = p -> adjvex;
        if (visited[j] == 0)  DFSTraverse(j);
        p = p -> next;
    }
}
```

```
template <class T>
void ALGraph<T>::BFSTraverse(int v)
{
    int Q[MaxSize];
    int front = -1, rear = -1;
    ArcNode * p;
    cout<<adjlist[v].vertex;
    visited[v] = 1;
    Q[++rear] = v;
    while (front != rear)
    {
        v = Q[++front];
        p = adjlist[v].firstedge;
        while (p != NULL)
        {
            int j = p -> adjvex;
            if (visited[j] == 0)
            {
                cout<<adjlist[j].vertex;
                visited[j] = 1;
                Q[++rear] = j;
            }
            p = p -> next;
        }
    }
}

template <class T>
void ALGraph<T>::ListToMatrix()
{
    int arc[MaxSize][MaxSize];
    for (int i = 0; i < vertexNum; i++)
    {
        for (int j = 0; j < vertexNum; j++)
            arc[i][j] = 0;
    }
    ArcNode * p;
    for (int i = 0; i < vertexNum; i++)
    {
        p = adjlist[i].firstedge;
        while (p)
        {
            int j = p -> adjvex;
            arc[i][j] = 1;
            arc[j][i] = 1;
            p = p -> next;
        }
    }
    cout<<endl<<"邻接矩阵"<<endl;
    for (int i = 0; i < vertexNum; i++)
    {
        for (int j = 0; j < vertexNum; j++)
            cout<<setw(5)<<arc[i][j];
        cout<<endl;
    }
}
```

(3)ALGraph_main.cpp

```
#include <iostream>
#include <iomanip>
#include "ALGraph.cpp"
using namespace std;

int visited[MaxSize] = {0};

int main()
{
    char ch[]={'A','B','C','D','E'};
    ALGraph<char> ALG(ch, 5, 7);
    for (int i = 0; i < MaxSize; i++)
        visited[i] = 0;
    cout<<"深度优先遍历序列是: ";
    ALG.DFSTraverse(0);
    cout<<endl;
    for (int i = 0; i < MaxSize; i++)
        visited[i] = 0;
    cout<<"广度优先遍历序列是: ";
    ALG.BFSTraverse(0);
    cout<<endl;
    ALG.ListToMatrix();
    return 0;
}
```

## 3. 计算度为 0 的顶点个数

(1)MGraph.h

```
#ifndef MGraph_H
#define MGraph_H
//int const MaxSize = 10;

template <class T>
class MGraph
{
    private:
        T vertex[MaxSize];
        int arc[MaxSize][MaxSize];
        int vertexNum, arcNum;
    public:
        MGraph(T a[], int n, int e);
        ~MGraph();
        void DFSTraverse(int v);
        void BFSTraverse(int v);
        void PrintMat();
        void getMat(int (&mat)[MaxSize][MaxSize]);
        int CountZero();
};

#endif
```

(2)MGraph.cpp

```
#include <iostream>
#include "MGraph.h"
using namespace std;

extern int visited[];

template <class T>
MGraph<T>::MGraph(T a[], int n, int e)
{
    int i, j, k;
    vertexNum = n, arcNum = e;
    for ( i = 0; i < vertexNum; i++)
        vertex[i] = a[i];
    for ( i = 0; i < vertexNum; i++)
        for ( j = 0; j < vertexNum; j++)
            arc[i][j] = 0;
    for ( k = 0; k < arcNum; k++)
    {
        cout<<"请输入边的两个顶点的序号：";
        cin>>i>>j;
        arc[i][j] = 1;
        arc[j][i] = 1;
    }
}

template <class T>
MGraph<T>::~MGraph()
{
    //empty
}

template <class T>
void MGraph<T>::DFSTraverse(int v)
{
    cout<<vertex[v];
    visited[v] = 1;
    for (int j = 0; j < vertexNum; j++)
        if (arc[v][j] == 1 && visited[j] == 0)
            DFSTraverse(j);
}

template <class T>
void MGraph<T>::BFSTraverse(int v)
{
    int Q[MaxSize];
    int front = -1, rear = -1;
    cout<<vertex[v];
    visited[v] = 1;
    Q[++rear] = v;
    while (front != rear)
    {
        v = Q[++front];
        for (int j = 0; j < vertexNum; j++)
            if (arc[v][j] == 1 && visited[j] == 0)
            {
                cout<<vertex[j];
                visited[j] = 1;
                Q[++rear] = j;
            }
    }
```

```
}

template <class T>
void MGraph<T>::PrintMat()
{
    int i, j;
    cout<<"\n\nMartix Graph:"<<endl;
    for ( i = 0; i < vertexNum; i++)
    {
        for ( j = 0; j < vertexNum; j++)
        {
            cout<<setw(5)<<arc[i][j];
        }
        cout<<endl;
    }
}

template <class T>
void MGraph<T>::getMat(int (&mat)[MaxSize][MaxSize])
{
    for (int i = 0; i < MaxSize; i++)
    {
        for (int j = 0; j < MaxSize; j++)
        {
            mat[i][j] = arc[i][j];
        }
    }
}

template <class T>
int MGraph<T>::CountZero()
{
    int count = 0;
    for (int i = 0; i < vertexNum; i++)
    {
        bool tag = 0;
        for (int j = 0; j < vertexNum; j++)
        {
            if (arc[i][j] != 0)
            {
                tag = 1;
                break;
            }
        }
        if (!tag)  count++;
    }
    return count;
}
```

(3) ALGraph.h

```
#ifndef ALGraph_H
#define ALGraph_H
const int MaxSize = 10;

struct ArcNode
{
    int adjvex;
    ArcNode * next;
};

template <class T>
struct VertexNode
{
    T vertex;
    ArcNode * firstedge;
};

template <class T>
class ALGraph
{
    private:
        VertexNode<T> adjlist[MaxSize];
        int vertexNum, arcNum;
    public:
        ALGraph(T a[], int n, int e);
        ALGraph(T a[], int n, int e, int arc[MaxSize][MaxSize]);
        ~ALGraph();
        void DFSTraverse(int v);
        void BFSTraverse(int v);
        int CountZero();
};

#endif
```

(4) ALGraph.cpp

```
#include <iostream>
```

```
#include <iomanip>
#include "ALGraph.h"
using namespace std;
extern int visited[];

template <class T>
ALGraph<T>::ALGraph(T a[], int n, int e)
{
    ArcNode * s;
    int i, j, k;
    vertexNum = n;
    arcNum = e;
    for ( i = 0; i < vertexNum; i++)
    {
        adjlist[i].vertex = a[i];
        adjlist[i].firstedge = NULL;
    }
    for ( k = 0; k < arcNum; k++)
    {
        cout<<"请输入边的两个顶点的序号：";
        cin>>i>>j;
        s = new ArcNode;
        s -> adjvex = j;
        s -> next = adjlist[i].firstedge;
        adjlist[i].firstedge = s;
    }
}

template <class T>
ALGraph<T>::ALGraph(T a[], int n, int e, int arc[MaxSize][MaxSize])
{
    arcNum = e;
    vertexNum = n;
    for (int i = 0; i < vertexNum; i++)
    {
        VertexNode<T> tempvertex;
        tempvertex.vertex = a[i];
        tempvertex.firstedge = NULL;
        adjlist[i] = tempvertex;
    }
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            if (j != i && arc[i][j] != 0)
            {
                ArcNode * s = new ArcNode;
                s -> adjvex = j;
                s -> next = adjlist[i].firstedge;
                adjlist[i].firstedge = s;
            }
        }
    }
}

template <class T>
ALGraph<T>::~ALGraph()
{
    ArcNode * p;
    for (int i = 0; i < vertexNum; i++)
    {
        p = adjlist[i].firstedge;
        while (p != NULL)
        {
            adjlist[i].firstedge = p -> next;
            delete p;
            p = adjlist[i].firstedge;
        }
    }
}

template <class T>
void ALGraph<T>::DFSTraverse(int v)
{
    ArcNode * p = NULL;
    int j;
    cout<<adjlist[v].vertex;
    visited[v] = 1;
    p = adjlist[v].firstedge;
    while (p != NULL)
    {
        j = p -> adjvex;
        if (visited[j] == 0)  DFSTraverse(j);
        p = p -> next;
    }
}

template <class T>
```

```
void ALGraph<T>::BFSTraverse(int v)
{
    int Q[MaxSize];
    int front = -1, rear = -1;
    ArcNode * p;
    cout<<adjlist[v].vertex;
    visited[v] = 1;
    Q[++rear] = v;
    while (front != rear)
    {
        v = Q[++front];
        p = adjlist[v].firstedge;
        while (p != NULL)
        {
            int j = p -> adjvex;
            if (visited[j] == 0)
            {
                cout<<adjlist[j].vertex;
                visited[j] = 1;
                Q[++rear] = j;
            }
            p = p -> next;
        }
    }
}

template <class T>
int ALGraph<T>::CountZero()
{
    int count = 0;
    ArcNode * p;
    for (int i = 0; i < vertexNum; i++)
    {
        p = adjlist[i].firstedge;
        if (!p)  count++;
    }
    return count;
}
```

(5)Graph_main.cpp

```
#include <iostream>
#include <iomanip>
using namespace std;
```

```
#include "ALGraph.cpp"
#include "MGraph.cpp"
int visited[MaxSize] = {0};

int main()
{
    char ch[]={'A','B','C','D','E','F','G'};
    MGraph<char> MG(ch, 8, 6); // 多出两个点
    int i = MG.CountZero();
    cout<<endl<<"出度为 0 的点的个数: "<<i<<endl;
    MG.PrintMat();
    int mat[MaxSize][MaxSize] = {0};
    MG.getMat(mat);
    ALGraph<char> ALG(ch, 8, 6, mat);  // 对照课本 P152
    int j = ALG.CountZero();
    cout<<endl<<"出度为 0 的点的个数: "<<j<<endl;
    return 0;
}
```

## 4. 邻接表非递归深度优先遍历

(1)MGraph.h

```
#ifndef MGraph_H
#define MGraph_H
const int MaxSize = 10;

template <class T>
class MGraph
{
    private:
        T vertex[MaxSize];
        int arc[MaxSize][MaxSize];
        int vertexNum, arcNum;
    public:
        MGraph(T a[], int n, int e);
        ~MGraph();
        void DFSTraverse(int v);
        void BFSTraverse(int v);
        void PrintMat();
        void getMat(int (&nat)[MaxSize][MaxSize]);
        int CountZero();
        void DFS_nrc(int v = 0);
};
```

```
#endif
```

(2)MGraph.cpp

```
#include <iostream>
#include "MGraph.h"
using namespace std;

extern int visited[];
template <class T>
MGraph<T>::MGraph(T a[], int n, int e)
{
    int i, j, k;
    vertexNum = n, arcNum = e;
    for ( i = 0; i < vertexNum; i++)
        vertex[i] = a[i];
    for ( i = 0; i < vertexNum; i++)
        for ( j = 0; j < vertexNum; j++)
            arc[i][j] = 0;
    for ( k = 0; k < arcNum; k++)
    {
        cout<<"请输入边的两个顶点的序号：";
        cin>>i>>j;
        arc[i][j] = 1;
        arc[j][i] = 1;
    }
}

template <class T>
MGraph<T>::~MGraph()
{
    //empty
}

template <class T>
void MGraph<T>::DFSTraverse(int v)
{
    cout<<vertex[v];
    visited[v] = 1;
    for (int j = 0; j < vertexNum; j++)
        if (arc[v][j] == 1 && visited[j] == 0)
            DFSTraverse(j);
}
```

```
template <class T>
void MGraph<T>::BFSTraverse(int v)
{
    int Q[MaxSize];
    int front = -1, rear = -1;
    cout<<vertex[v];
    visited[v] = 1;
    Q[++rear] = v;
    while (front != rear)
    {
        v = Q[++front];
        for (int j = 0; j < vertexNum; j++)
            if (arc[v][j] == 1 && visited[j] == 0)
            {
                cout<<vertex[j];
                visited[j] = 1;
                Q[++rear] = j;
            }
    }
}

template <class T>
void MGraph<T>::PrintMat()
{
    int i, j;
    cout<<"\n\nMartix Graph:"<<endl;
    for ( i = 0; i < vertexNum; i++)
    {
        for ( j = 0; j < vertexNum; j++)
        {
            cout<<setw(5)<<arc[i][j];
        }
        cout<<endl;
    }
}

template <class T>
void MGraph<T>::getMat(int (&mat)[MaxSize][MaxSize])
{
    for (int i = 0; i < MaxSize; i++)
    {
        for (int j = 0; j < MaxSize; j++)
        {
            mat[i][j] = arc[i][j];
        }
    }
}

template <class T>
int MGraph<T>::CountZero()
{
    int count = 0;
    for (int i = 0; i < vertexNum; i++)
    {
        bool tag = 0;
        for (int j = 0; j < vertexNum; j++)
        {
            if (arc[i][j] != 0)
            {
                tag = 1;
                break;
            }
        }
        if (!tag)  count++;
    }
    return count;
}

template <class T>
void MGraph<T>::DFS_nrc(int v)
{
    int top = -1;
    cout<<vertex[v];
    int visited[vertexNum] = {0};
    visited[v] = 1;
    int S[vertexNum] = {0};
    S[++top] = v;
    while (top != -1)
    {
        v = S[top];
        int j;
        for (j = 0; j < vertexNum; j++)
        {
            if (arc[v][j] == 1 && visited[j] == 0)
            {
                cout<<vertex[j];
                visited[j] = 1;
                S[++top] = j;
                break;
            }
        }
        if (j == vertexNum)  top--;
    }
}
```

(3)MGraph_main.cpp

```
#include <iostream>
#include <iomanip>
using namespace std;
#include "MGraph.cpp"
int visited[MaxSize] = {0};

int main()
{
    char ch[]={'A','B','C','D','E'};
    MGraph<char> MG(ch, 6, 6); //多出两个点
    MG.DFS_nrc(1);
    cout<<endl;
    MG.DFSTraverse(1);
    MG.PrintMat();
    return 0;
}
```

## 5. 建立逆邻接表

(1)ALGraph.h

```
#ifndef ALGraph_H
#define ALGraph_H
const int MaxSize = 10;

struct ArcNode
{
    int adjvex;
    ArcNode * next;
};

template <class T>
struct VertexNode
{
    T vertex;
    ArcNode * firstedge;
};

template <class T>
class ALGraph
{
    private:
        VertexNode<T> adjlist[MaxSize];
        int vertexNum, arcNum;
    public:
        ALGraph(T a[], int n, int e);
        ALGraph(T a[], int n, int e, int arc[MaxSize][MaxSize]);
        ~ALGraph();
        void DFSTraverse(int v);
        void BFSTraverse(int v);
        int CountZero();
        void List(VertexNode<T> (&Inv)[MaxSize]);
};

#endif
```

(2) ALGraph. cpp

```
#include <iostream>
#include <iomanip>
#include "ALGraph.h"
using namespace std;
extern int visited[];

template <class T>
ALGraph<T>::ALGraph(T a[], int n, int e)
{
    ArcNode * s;
    int i, j, k;
    vertexNum = n;
    arcNum = e;
    for ( i = 0; i < vertexNum; i++)
    {
        adjlist[i].vertex = a[i];
        adjlist[i].firstedge = NULL;
    }
    for ( k = 0; k < arcNum; k++)
    {
        cout<<"请输入边的两个顶点的序号：";
        cin>>i>>j;
        s = new ArcNode;
        s -> adjvex = j;
        s -> next = adjlist[i].firstedge;
        adjlist[i].firstedge = s;
    }
}

template <class T>
ALGraph<T>::ALGraph(T a[], int n, int e, int arc[MaxSize][MaxSize])
{
    arcNum = e;
    vertexNum = n;
    for (int i = 0; i < vertexNum; i++)
    {
        VertexNode<T> tempvertex;
        tempvertex.vertex = a[i];
        tempvertex.firstedge = NULL;
        adjlist[i] = tempvertex;
    }
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            if (j != i && arc[i][j] != 0)
            {
                ArcNode * s = new ArcNode;
                s -> adjvex = j;
                s -> next = adjlist[i].firstedge;
                adjlist[i].firstedge = s;
            }
        }
    }
}

template <class T>
ALGraph<T>::~ALGraph()
{
    ArcNode * p;
    for (int i = 0; i < vertexNum; i++)
    {
        p = adjlist[i].firstedge;
        while (p != NULL)
        {
            adjlist[i].firstedge = p -> next;
            delete p;
            p = adjlist[i].firstedge;
        }
    }
}

template <class T>
void ALGraph<T>::DFSTraverse(int v)
{
    ArcNode * p = NULL;
    int j;
    cout<<adjlist[v].vertex;
    visited[v] = 1;
    p = adjlist[v].firstedge;
    while (p != NULL)
    {
        j = p -> adjvex;
        if (visited[j] == 0)  DFSTraverse(j);
        p = p -> next;
    }
}

template <class T>
void ALGraph<T>::BFSTraverse(int v)
{
    int Q[MaxSize];
    int front = -1, rear = -1;
    ArcNode * p;
    cout<<adjlist[v].vertex;
    visited[v] = 1;
    Q[++rear] = v;
    while (front != rear)
    {
        v = Q[++front];
        p = adjlist[v].firstedge;
        while (p != NULL)
        {
            int j = p -> adjvex;
            if (visited[j] == 0)
            {
                cout<<adjlist[j].vertex;
                visited[j] = 1;
                Q[++rear] = j;
            }
```

```
                p = p -> next;
            }
        }
}

template <class T>
int ALGraph<T>::CountZero()
{
    int count = 0;
    ArcNode * p;
    for (int i = 0; i < vertexNum; i++)
    {
        p = adjlist[i].firstedge;
        if (!p)  count++;
    }
    return count;
}

template <class T>
void ALGraph<T>::List(VertexNode<T> (&Inv)[MaxSize])
{
    for (int i = 0; i < vertexNum; i++)
    {
        Inv[i].firstedge = NULL;
        Inv[i].vertex = adjlist[i].vertex;
    }
    ArcNode * p = new ArcNode;
    int j;
    for (int i = 0; i < vertexNum; i++)
    {
        p = adjlist[i].firstedge;
        while (p != NULL)
        {
            j = p -> adjvex;
            ArcNode * s = new ArcNode;
            s -> adjvex = i;
            s -> next = Inv[j].firstedge;
            Inv[j].firstedge = s;
            p = p -> next;
        }
    }
}
```

(3)ALGraph_main.cpp

```cpp
#include <iostream>
#include <iomanip>
using namespace std;
#include "ALGraph.cpp"
int visited[MaxSize] = {0};

int main()
{
    char ch[]={'A','B','C','D'};
    ALGraph<char> ALG(ch, 4, 4);
    VertexNode<char> Inv[MaxSize];
    ALG.List(Inv);
    return 0;
}
```

## 6. 判断有向图的路径

(1)ALGraph.h

```cpp
#ifndef ALGraph_H
#define ALGraph_H
const int MaxSize = 10;

struct ArcNode
{
    int adjvex;
    ArcNode * next;
};

template <class T>
struct VertexNode
{
    T vertex;
    ArcNode * firstedge;
};

template <class T>
class ALGraph
{
    private:
        VertexNode<T> adjlist[MaxSize];
        int vertexNum, arcNum;
    public:
        ALGraph(T a[], int n, int e);
        ALGraph(T a[], int n, int e, int arc[MaxSize][MaxSize]);
        ~ALGraph();
        void DFSTraverse(int v);
        void BFSTraverse(int v);
        int CountZero();
        void List(VertexNode<T> (&Inv)[MaxSize]);
        bool Path_DFS(int x, int y);
        bool Path_BFS(int x, int y);
};

#endif
```

(2) ALGraph. cpp

```
#include <iostream>
#include <iomanip>
#include "ALGraph.h"
using namespace std;
extern int visited[];

template <class T>
ALGraph<T>::ALGraph(T a[], int n, int e)
{
    ArcNode * s;
    int i, j, k;
    vertexNum = n;
    arcNum = e;
    for ( i = 0; i < vertexNum; i++)
    {
        adjlist[i].vertex = a[i];
        adjlist[i].firstedge = NULL;
    }
    for ( k = 0; k < arcNum; k++)
    {
        cout<<"请输入边的两个顶点的序号：";
        cin>>i>>j;
        s = new ArcNode;
        s -> adjvex = j;
        s -> next = adjlist[i].firstedge;
        adjlist[i].firstedge = s;
    }
}
```

```
template <class T>
ALGraph<T>::ALGraph(T a[], int n, int e, int arc[MaxSize][MaxSize])
{
    arcNum = e;
    vertexNum = n;
    for (int i = 0; i < vertexNum; i++)
    {
        VertexNode<T> tempvertex;
        tempvertex.vertex = a[i];
        tempvertex.firstedge = NULL;
        adjlist[i] = tempvertex;
    }
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            if (j != i && arc[i][j] != 0)
            {
                ArcNode * s = new ArcNode;
                s -> adjvex = j;
                s -> next = adjlist[i].firstedge;
                adjlist[i].firstedge = s;
            }
        }
    }
}

template <class T>
ALGraph<T>::~ALGraph()
{
    ArcNode * p;
    for (int i = 0; i < vertexNum; i++)
    {
        p = adjlist[i].firstedge;
        while (p != NULL)
        {
            adjlist[i].firstedge = p -> next;
            delete p;
            p = adjlist[i].firstedge;
        }
    }
}

template <class T>
```

```cpp
void ALGraph<T>::DFSTraverse(int v)
{
    ArcNode * p = NULL;
    int j;
    cout<<adjlist[v].vertex;
    visited[v] = 1;
    p = adjlist[v].firstedge;
    while (p != NULL)
    {
        j = p -> adjvex;
        if (visited[j] == 0)  DFSTraverse(j);
        p = p -> next;
    }
}

template <class T>
void ALGraph<T>::BFSTraverse(int v)
{
    int Q[MaxSize];
    int front = -1, rear = -1;
    ArcNode * p;
    cout<<adjlist[v].vertex;
    visited[v] = 1;
    Q[++rear] = v;
    while (front != rear)
    {
        v = Q[++front];
        p = adjlist[v].firstedge;
        while (p != NULL)
        {
            int j = p -> adjvex;
            if (visited[j] == 0)
            {
                cout<<adjlist[j].vertex;
                visited[j] = 1;
                Q[++rear] = j;
            }
            p = p -> next;
        }
    }
}

template <class T>
int ALGraph<T>::CountZero()
```

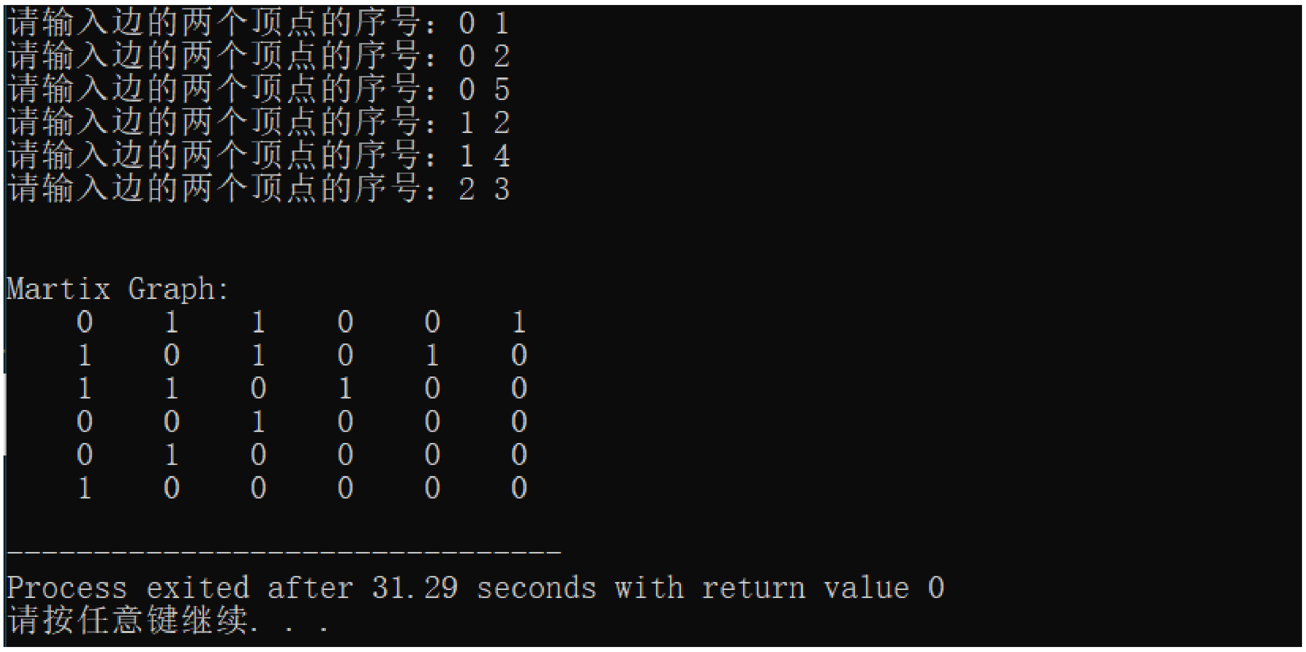
```
{
    int count = 0;
    ArcNode * p;
    for (int i = 0; i < vertexNum; i++)
    {
        p = adjlist[i].firstedge;
        if (!p)  count++;
    }
    return count;
}

template <class T>
void ALGraph<T>::List(VertexNode<T> (&Inv)[MaxSize])
{
    for (int i = 0; i < vertexNum; i++)
    {
        Inv[i].firstedge = NULL;
        Inv[i].vertex = adjlist[i].vertex;
    }
    ArcNode * p = new ArcNode;
    int j;
    for (int i = 0; i < vertexNum; i++)
    {
        p = adjlist[i].firstedge;
        while (p != NULL)
        {
            j = p -> adjvex;
            ArcNode * s = new ArcNode;
            s -> adjvex = i;
            s -> next = Inv[j].firstedge;
            Inv[j].firstedge = s;
            p = p -> next;
        }
    }
}

template <class T>
bool ALGraph<T>::Path_DFS(int x, int y)
{
    int v = x;
    int top, j;
    ArcNode * p;
    for (int i = 0; i < vertexNum; i++)
    {
        visited[0] = 0;
    }
    int S[MaxSize];
    top = -1;
    visited[v] = 1;
    S[++top] = v;
    while (top != -1)
    {
        v = S[top];
        p = adjlist[v].firstedge;
        while (p != NULL)
        {
            j = p -> adjvex;
            if (visited[j] == 0)
            {
                cout<<adjlist[j].vertex<<" ";
                if (j == y)  return true;
                visited[j] == 1;
                S[++top] = j;
                break;
            }
            p = p -> next;
        }
        if (p == NULL)  top--;
    }
    return false;
}

template <class T>
bool ALGraph<T>::Path_BFS(int x, int y)
{
    int v = x;
    int front, rear, j;
    ArcNode * p;
    for (int i = 0; i < vertexNum; i++)
    {
        visited[0] = 0;
    }
    int Q[MaxSize];
    front = rear = -1;
    visited[v] = 1;
    Q[++rear] = v;
    while (front != rear)
    {
        v = Q[++front];
        p = adjlist[v].firstedge;
        while (p != NULL)
        {
            j = p -> adjvex;
            if (visited[j] == 0)
            {
                cout<<adjlist[j].vertex<<" ";
                if (j == y)  return true;
                visited[j] == 1;
                Q[++rear] = j;
            }
            p = p -> next;
        }
    }
    return false;
}

void ifs(bool flag)
{
    if (flag)  cout<<"has path"<<endl;
    else  cout<<"no path"<<endl;
}
```

(3)ALGraph_main.cpp

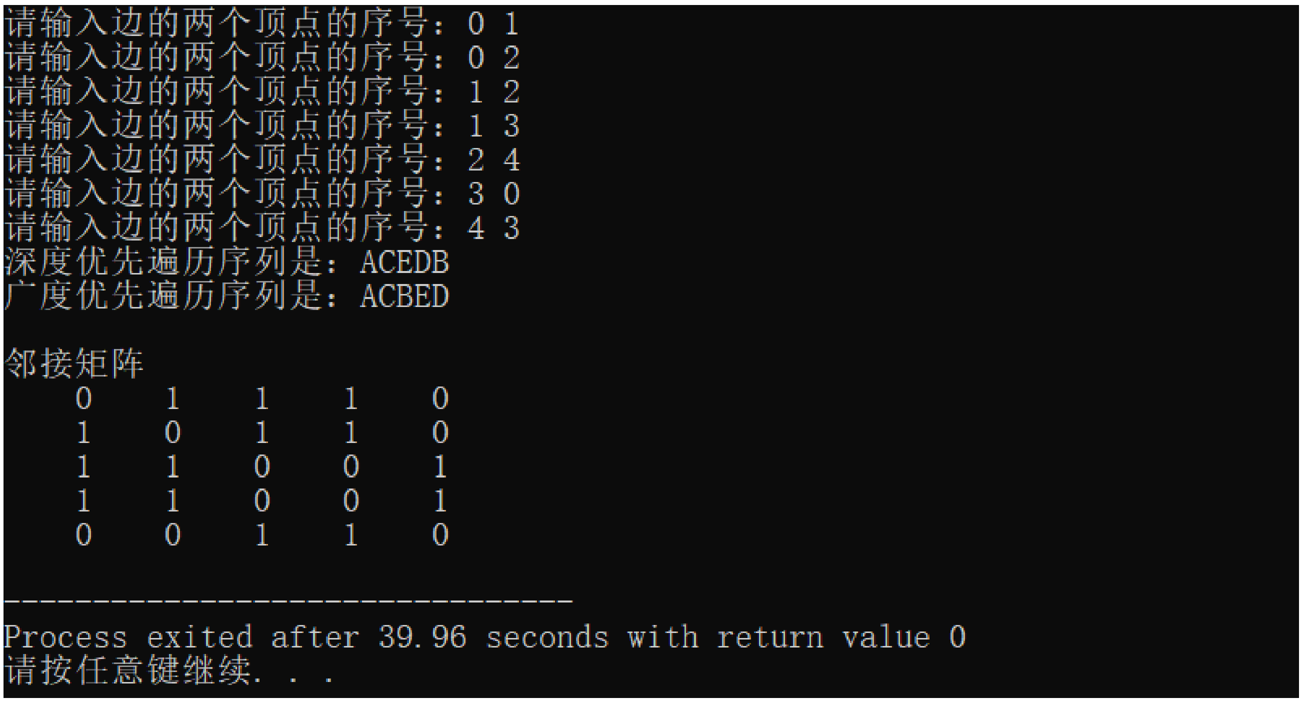
```
#include <iostream>
#include <iomanip>
using namespace std;
#include "ALGraph.cpp"
int visited[MaxSize] = {0};
int main()
{
    char ch[]={'A','B','C','D'};
    ALGraph<char> ALG(ch, 4, 4);
    bool flag;
    flag = ALG.Path_BFS(2,0); ifs(flag);
    flag = ALG.Path_DFS(2,0); ifs(flag);
    flag = ALG.Path_BFS(2,1); ifs(flag);
    flag = ALG.Path_DFS(2,1); ifs(flag);
    flag = ALG.Path_BFS(1,3); ifs(flag);
    flag = ALG.Path_DFS(1,3); ifs(flag);
    return 0;
}
```

## 四、运行结果

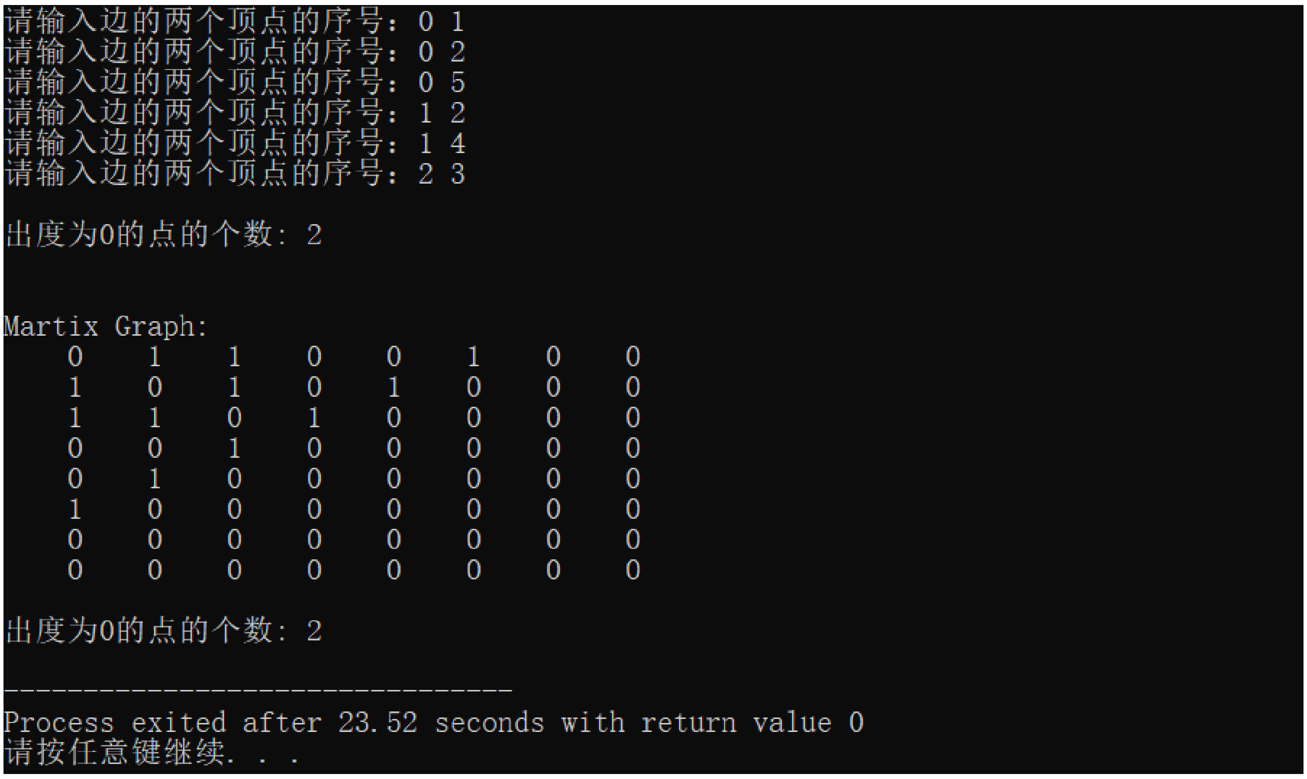
### 1. 无向图邻接矩阵转换为邻接表


```
请输入边的两个顶点的序号: 0 1
请输入边的两个顶点的序号: 0 2
请输入边的两个顶点的序号: 0 5
请输入边的两个顶点的序号: 1 2
请输入边的两个顶点的序号: 1 4
请输入边的两个顶点的序号: 2 3


Martix Graph:
    0    1    1    0    0    1
    1    0    1    0    1    0
    1    1    0    1    0    0
    0    0    1    0    0    0
    0    1    0    0    0    0
    1    0    0    0    0    0

--------------------------------
Process exited after 31.29 seconds with return value 0
请按任意键继续. . .
```

### 2. 无向图邻接表转换为邻接矩阵

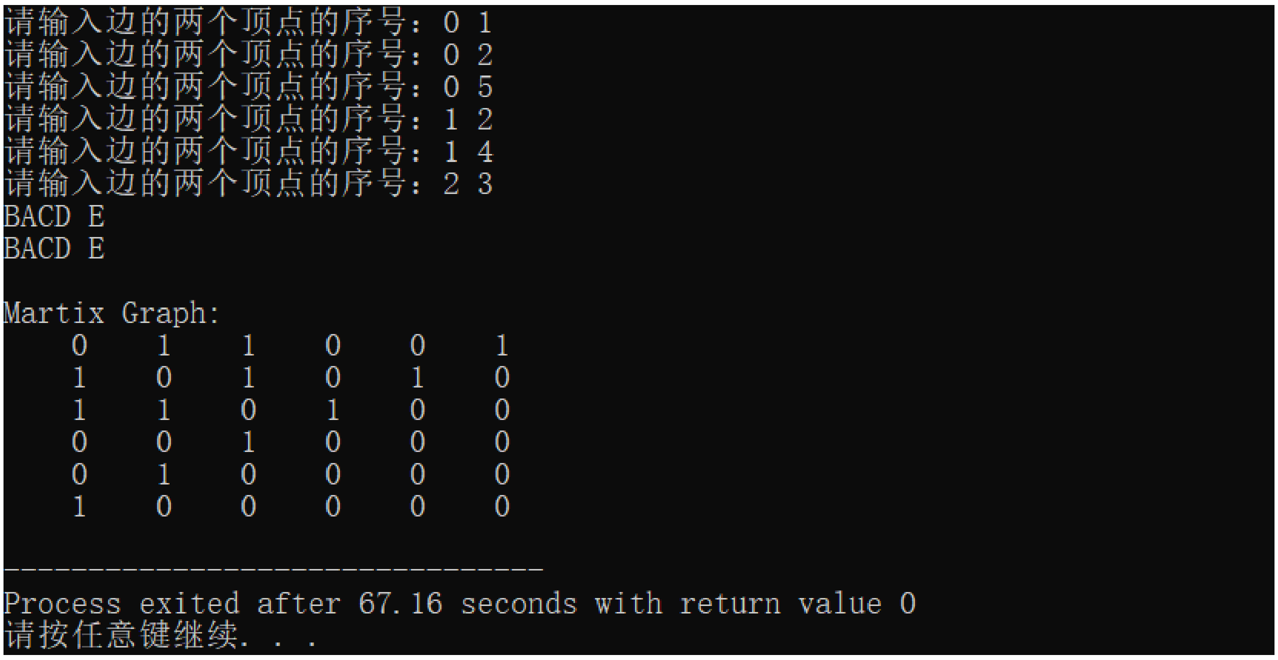

```
请输入边的两个顶点的序号: 0 1
请输入边的两个顶点的序号: 0 2
请输入边的两个顶点的序号: 1 2
请输入边的两个顶点的序号: 1 3
请输入边的两个顶点的序号: 2 4
请输入边的两个顶点的序号: 3 0
请输入边的两个顶点的序号: 4 3
深度优先遍历序列是: ACEDB
广度优先遍历序列是: ACBED

邻接矩阵
    0    1    1    1    0
    1    0    1    1    0
    1    1    0    0    1
    1    1    0    0    1
    0    0    1    1    0

--------------------------------
Process exited after 39.96 seconds with return value 0
请按任意键继续. . .
```

3. 计算度为 0 的顶点个数



```
请输入边的两个顶点的序号: 0 1
请输入边的两个顶点的序号: 0 2
请输入边的两个顶点的序号: 0 5
请输入边的两个顶点的序号: 1 2
请输入边的两个顶点的序号: 1 4
请输入边的两个顶点的序号: 2 3

出度为0的点的个数: 2


Martix Graph:
    0    1    1    0    0    1    0    0
    1    0    1    0    1    0    0    0
    1    1    0    1    0    0    0    0
    0    0    1    0    0    0    0    0
    0    1    0    0    0    0    0    0
    1    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0

出度为0的点的个数: 2

--------------------------------
Process exited after 23.52 seconds with return value 0
请按任意键继续. . .
```

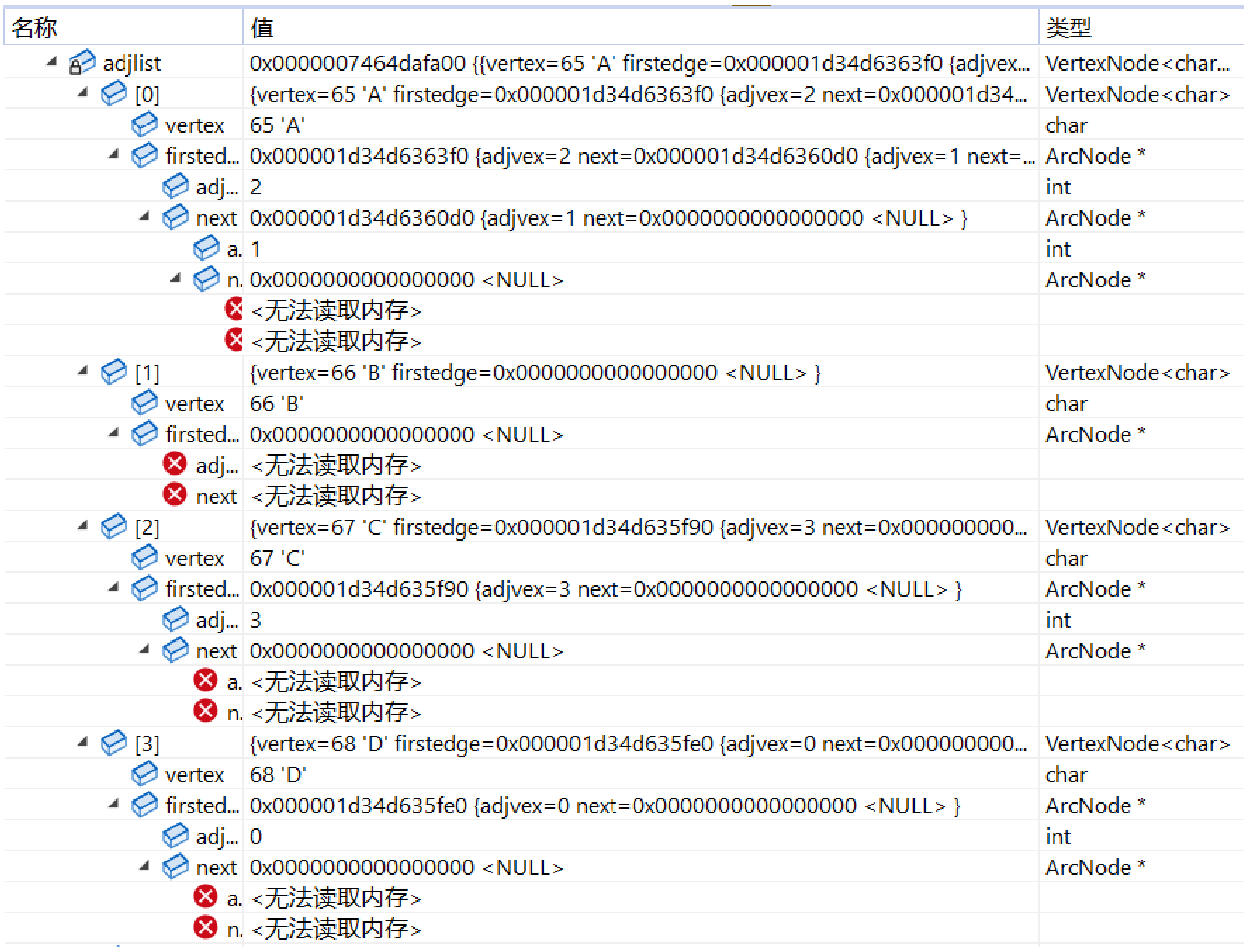
4. 邻接表非递归深度优先遍历



```
请输入边的两个顶点的序号: 0 1
请输入边的两个顶点的序号: 0 2
请输入边的两个顶点的序号: 0 5
请输入边的两个顶点的序号: 1 2
请输入边的两个顶点的序号: 1 4
请输入边的两个顶点的序号: 2 3
BACD E
BACD E

Martix Graph:
    0    1    1    0    0    1
    1    0    1    0    1    0
    1    1    0    1    0    0
    0    0    1    0    0    0
    0    1    0    0    0    0
    1    0    0    0    0    0

--------------------------------
Process exited after 67.16 seconds with return value 0
请按任意键继续. . .
```

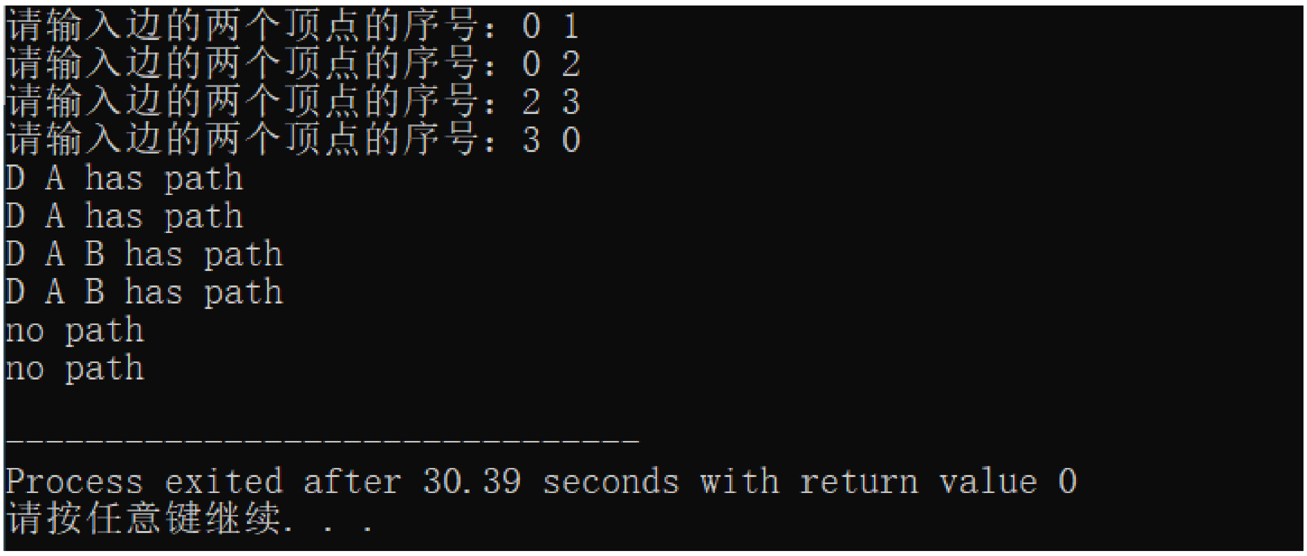
5. 建立逆邻接表

监视器视角如图所示，符合样例。



| 名称 | 值 | 类型 |
|---|---|---|
| adjlist | 0x0000007464dafa00 {{vertex=65 'A' firstedge=0x000001d34d6363f0 {adjvex... | VertexNode<char... |
| [0] | {vertex=65 'A' firstedge=0x000001d34d6363f0 {adjvex=2 next=0x000001d34... | VertexNode<char> |
| vertex | 65 'A' | char |
| firsted... | 0x000001d34d6363f0 {adjvex=2 next=0x000001d34d6360d0 {adjvex=1 next=... | ArcNode * |
| adj... | 2 | int |
| next | 0x000001d34d6360d0 {adjvex=1 next=0x0000000000000000 <NULL> } | ArcNode * |
| a. | 1 | int |
| n. | 0x0000000000000000 <NULL> | ArcNode * |
|  | <无法读取内存> |  |
|  | <无法读取内存> |  |
| [1] | {vertex=66 'B' firstedge=0x0000000000000000 <NULL> } | VertexNode<char> |
| vertex | 66 'B' | char |
| firsted... | 0x0000000000000000 <NULL> | ArcNode * |
| adj... | <无法读取内存> |  |
| next | <无法读取内存> |  |
| [2] | {vertex=67 'C' firstedge=0x000001d34d635f90 {adjvex=3 next=0x000000000... | VertexNode<char> |
| vertex | 67 'C' | char |
| firsted... | 0x000001d34d635f90 {adjvex=3 next=0x0000000000000000 <NULL> } | ArcNode * |
| adj... | 3 | int |
| next | 0x0000000000000000 <NULL> | ArcNode * |
| a. | <无法读取内存> |  |
| n. | <无法读取内存> |  |
| [3] | {vertex=68 'D' firstedge=0x000001d34d635fe0 {adjvex=0 next=0x000000000... | VertexNode<char> |
| vertex | 68 'D' | char |
| firsted... | 0x000001d34d635fe0 {adjvex=0 next=0x0000000000000000 <NULL> } | ArcNode * |
| adj... | 0 | int |
| next | 0x0000000000000000 <NULL> | ArcNode * |
| a. | <无法读取内存> |  |
| n. | <无法读取内存> |  |

## 6. 判断有向图的路径



```
请输入边的两个顶点的序号: 0 1
请输入边的两个顶点的序号: 0 2
请输入边的两个顶点的序号: 2 3
请输入边的两个顶点的序号: 3 0
D A has path
D A has path
D A B has path
D A B has path
no path
no path

--------------------------------
Process exited after 30.39 seconds with return value 0
请按任意键继续. . .
```

## 六、总结与心得

这次的实验主要围绕无向图和有向图的性质以及他们的深度遍历和广度遍历实现，每一个实验项目的头文件和函数文件大同小异，主要根据实验目的进行函数的补充和删除，但是自己编程实现这些带有特定目的的函数，还是有一定的难度。

另外，本章节的实验对于调试的能力要求较高，需要自己对调试功能进行再次的熟悉和学习。