哈尔滨工业大学计算机科学与技术学院

实验报告

课程名称： 数据结构与算法

课程类型： 必修

实验项目名称： 图型结构与应用

实验题目： 图的储存结构的建立于遍历

班级： 1303101

学号： 1130310128

姓名： 杨尚斌

设计成绩	报告成绩	指导老师
		张岩

一、 实验目的

1.熟悉图的遍历搜索方法.

2.掌握邻接矩阵的 DFS(递归和非递归)与 BFS.

3.掌握邻接表的 DFS(递归和非递归)与 BFS.

二、实验要求及实验环境

1.实验要求

(1)能够建立（有向和无向）图的邻接矩阵和邻接表存储结构。

(2)能够在邻接矩阵和邻接表存储结构上对（有向和无向）图进行深度优先（递归和非递归都要求）和广度优先搜索。

(3)能够存储和显示相应的搜索结果（深度优先或广度优先生成森林（或生成树）、深度优先或广度优先序列和编号）。

(4)以文件形式输入图的顶点和边，并显示相应的结果。要求顶点不少于 10 个，边不少于 13 个。

（5） 软件功能结构安排合理，界面友好，便于使用。

2.测试环境

操作系统：windows x64 sp1

编译器：g++ 4.7.1

二、 设计思想

1. 逻辑设计

结构体：GRAPH

用于储存邻接矩阵，里面有元素 n(点的个数)，e(边的个数)，edge[n][n]（边）， vertex(点)。

LINK

链表，储存节点的连接。

node

用于储存链表，里面有元素 vertex, *first。

LIST

用于储存在邻接表，里面有元素 n, e, a[n](node 类型)。

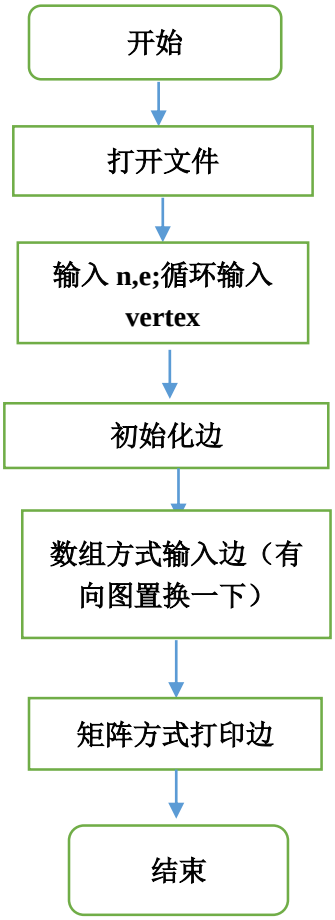
函数：

int main()	主函数
createGeaph(G)	创建并打印邻接矩阵(无向)
createGeaph1(G)	创建并打印邻接矩阵(有向)
initGraph(G)	初始化邻接矩阵
DFSGraph1(G)与 DFSGraph1Start(GRAPH	递归深度搜索

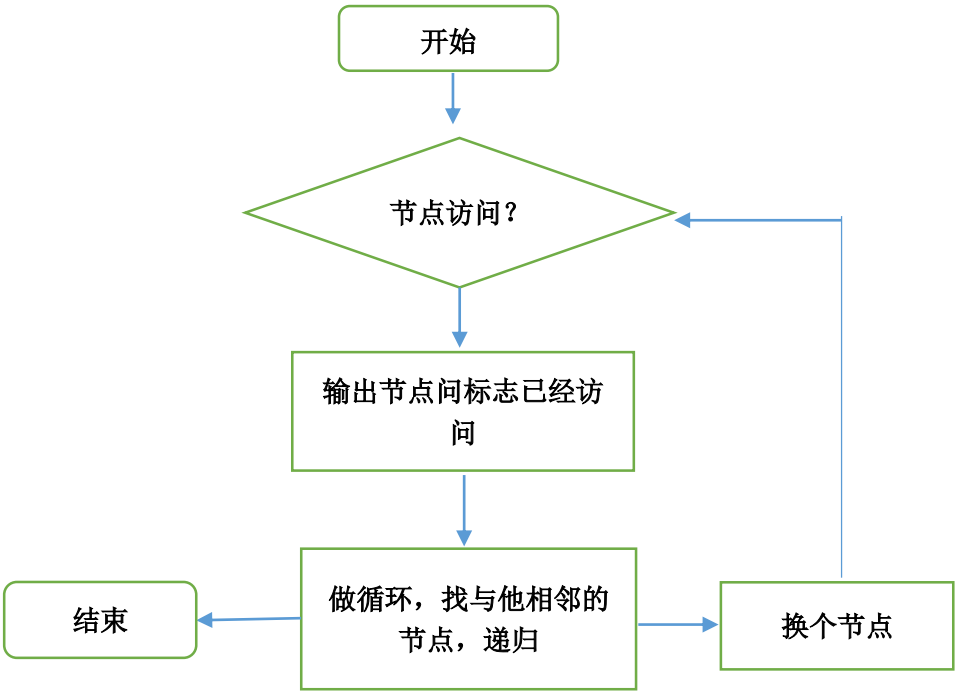
*G, int i)	
DFSGraph2（G）与 DFSGraph2Start(GRAPH *G, int i)	非递归深度搜索
BFSGraph(G)	广度搜索
createList(L)	创建并打印邻接表（无向）
createList1(L)	创建并打印邻接表（有向）
initList(L)	初始化邻接矩阵
DFSList1(L)与 DFSList1Start(LIST *L, int v)	递归深度搜索
DFSList2(L)与 DFSList2Start(LIST *L, int v)	非递归深度搜索
BFSList(L)	广度搜索

2. 物理设计

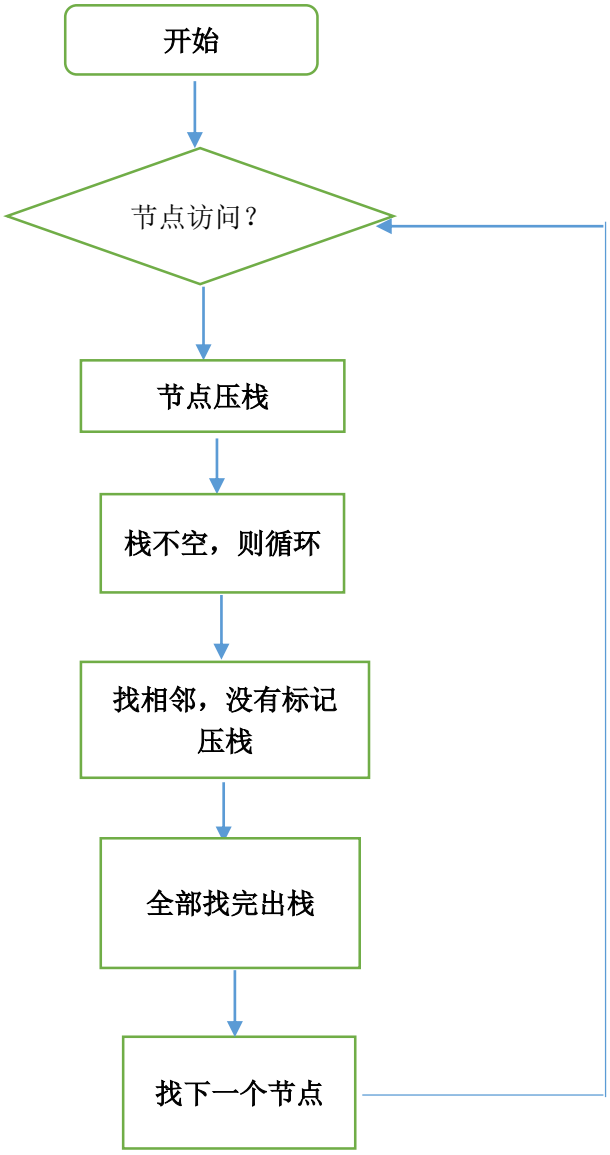
1. 创建邻接矩阵



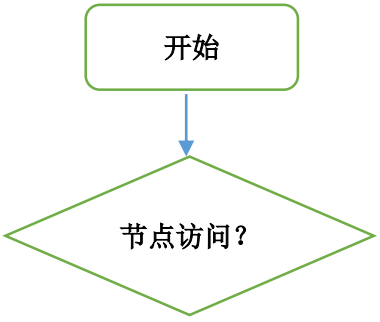
2. 递归深度搜索邻接矩阵

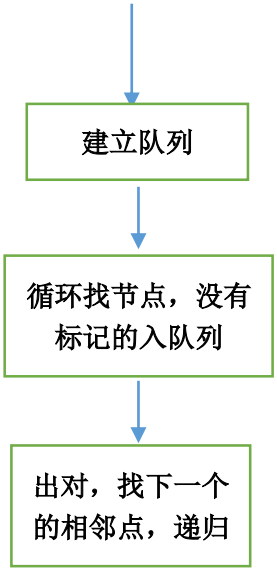


3. 非递归深度搜索邻接矩阵

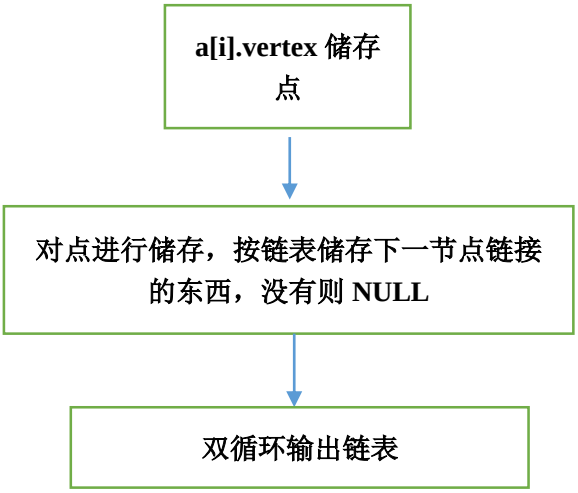


4. 广度搜索邻接矩阵





5. 创建邻接表



6. 递归深度搜素邻接表

1.访问第源元素标记为访问；

2.遍历所有的和源元素相关的元素，如果为被访问，递归的搜索该顶点；

7. 非递归深度搜索邻接表

1.建立空栈，第一个元素进栈；

2.循环到栈空

3.出栈

4.如果出栈的元素没有访问，访问，标记为已经访问；

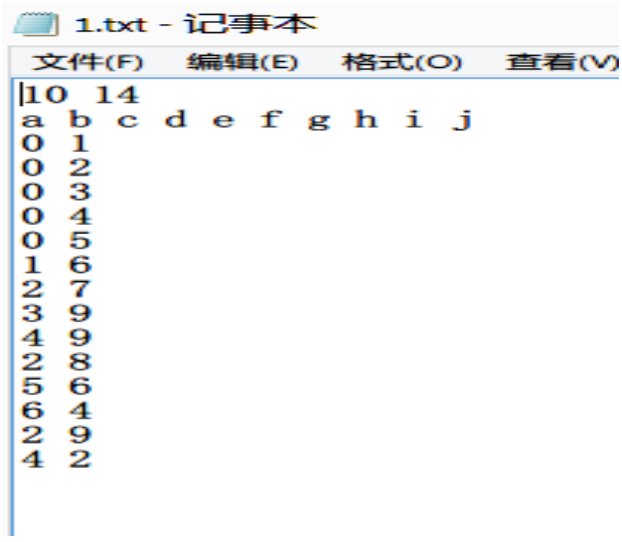
5.遍历和出栈元素有关系的顶点而且没有被访问，压栈；

8. 广度搜索邻接表

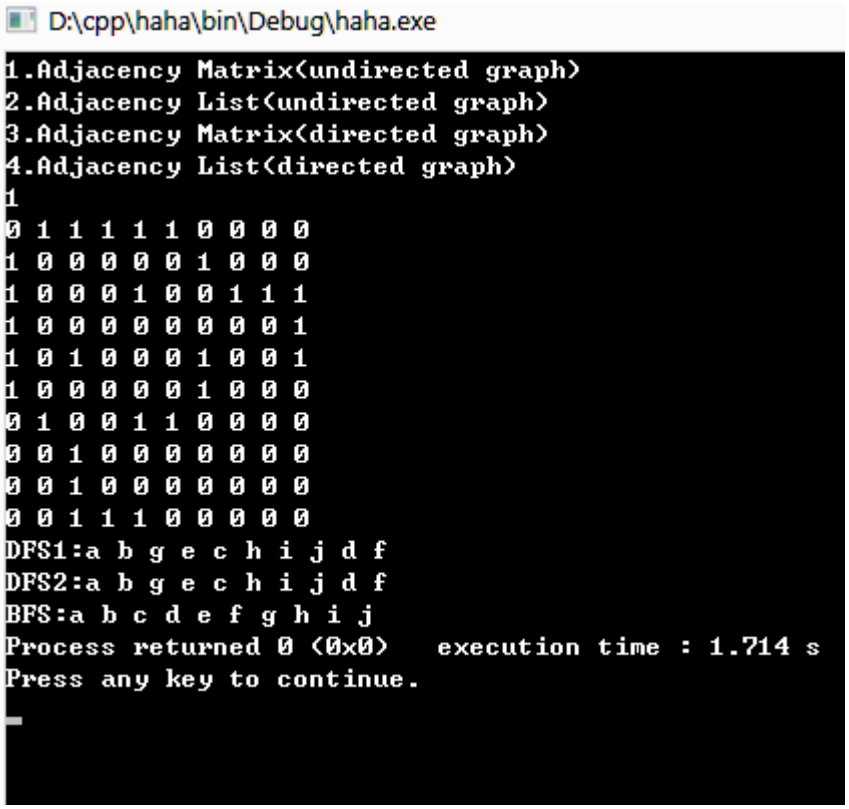
1. 初始化队列
2. 访问顶点，标记，入队列
3. 出队列，得第一个邻接点
4. 如果为访问则访问标记，否则入队

三、 测试结果

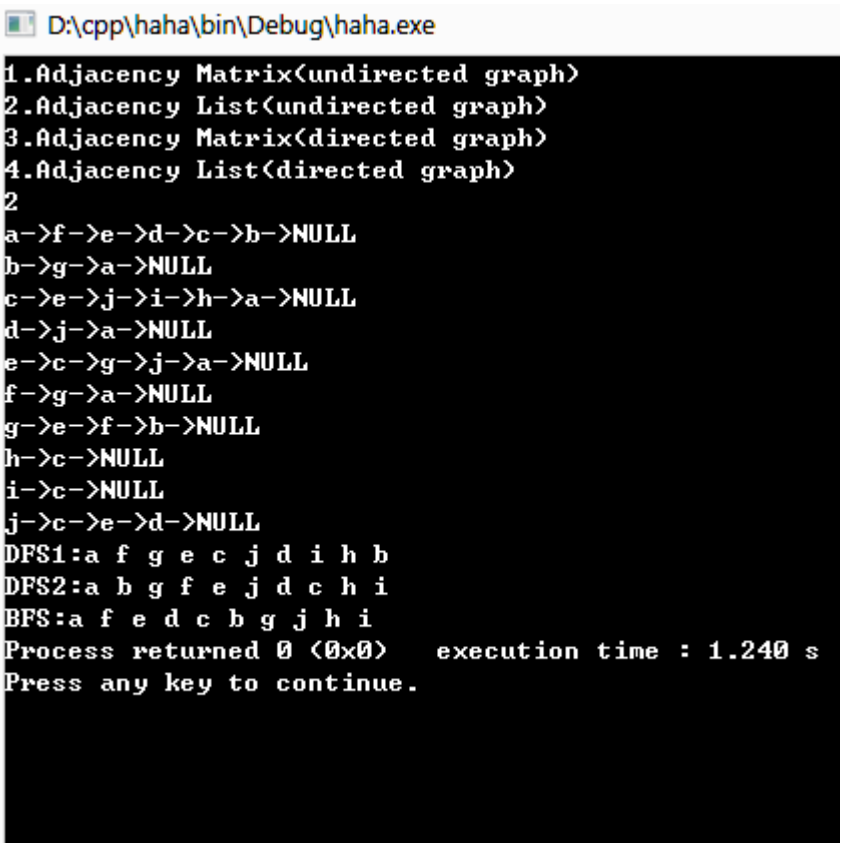
测试数据：



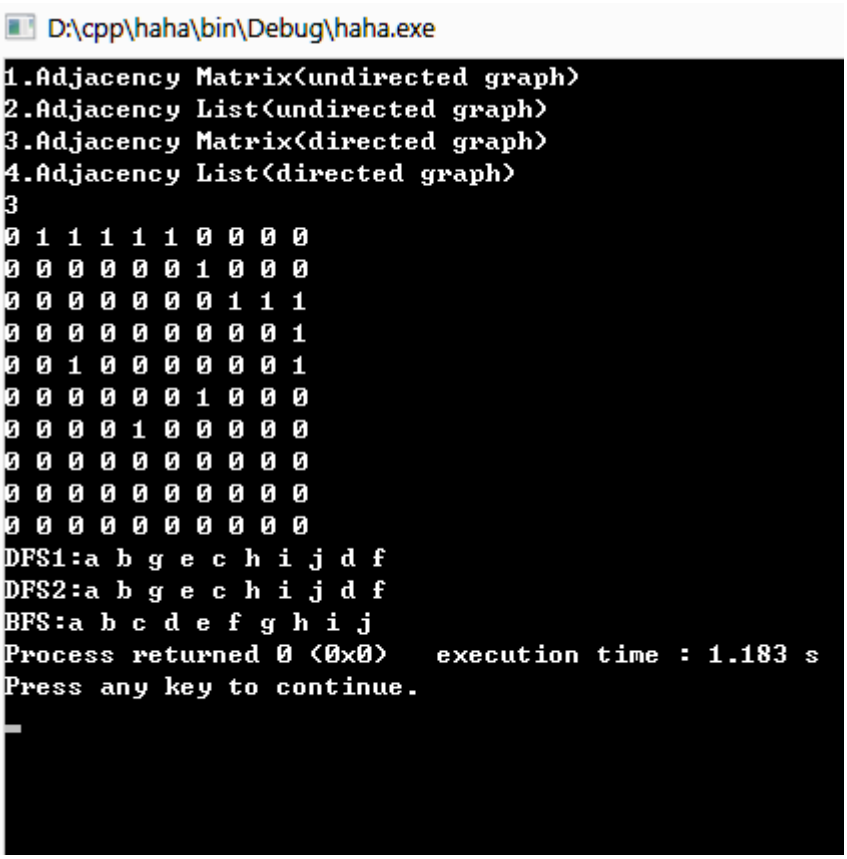
邻接矩阵测试结果（无向图）



邻接表测试效果（无向图）



邻接矩阵测试结果（有向图）



邻接表测试效果（有向图）

四、 系统不足与经验体会

系统不足：

1. 在输入菜单选择阶段，如果写一个 while 循环，会造成程序的报错（应该是输入流的问题），目前没有找到相应解决方法。
2. 由于时间关系没有对输出的东西进行文件保存。

经验体会：

1. 课堂上对内容进行较好的消化，在邻接表阶段花费时间太长。
2. 需要再度复习 C++的相关操作，尤其表现文件操作模块。

六、附录：源代码（带注释）

```
#include <iostream>
#include <stdlib.h>
#include <fstream>
#define N 50
using namespace std;
struct GRAPH
{
    int n, e;
    int edge[N][N];
    char vertex[N];
};
struct LINK
{
    int v;
    LINK *next;
};
struct node
{
    char vertex;
    struct LINK * first;
};
struct LIST
{
    int n, e;
    node a[N];
};
//the point is visited or not
bool visit[N];
void createGraph(GRAPH &G)
{
    int a, b;
    cin >> G.n >> G.e;
    //input the vertex
    for(int i = 0; i < G.n; i++)
    {
        cin >> G.vertex[i];
    }
    //Init the edge
    for(int i = 0; i < G.n; i++)
    {
        for(int j = 0; j < G.n; j++)
```

```
        {
            G.edge[i][j] = 0;
        }
    }
    //input the edge
    for(int i = 0; i < G.e; i++)
    {
        //undirected graph
        cin >> a >> b;
        G.edge[a][b] = 1;
        G.edge[b][a] = 1;
    }
    //print the graph with Matrix
    for(int i = 0; i < G.n; i++)
    {
        for(int j = 0; j < G.n; j++)
        {
            cout << G.edge[i][j] << " ";
        }
        cout << endl;
    }
}
void createGraph2(GRAPH &G)
{
    int a, b;
    cin >> G.n >> G.e;
    //input the vertex
    for(int i = 0; i < G.n; i++)
    {
        cin >> G.vertex[i];
    }
    //Init the edge
    for(int i = 0; i < G.n; i++)
    {
        for(int j = 0; j < G.n; j++)
        {
            G.edge[i][j] = 0;
        }
    }
    //input the edge
    for(int i = 0; i < G.e; i++)
    {
        //directed graph
        cin >> a >> b;
```

```
        G.edge[a][b] = 1;
    }
    //print the graph with Matrix
    for(int i = 0; i < G.n; i++)
    {
        for(int j = 0; j < G.n; j++)
        {
            cout << G.edge[i][j] << " ";
        }
        cout << endl;
    }
}
//init the visiting, with false
void initGraph(GRAPH & G)
{
    for(int i = 0; i < G.n; i++)
    {
        visit[i] = false;
    }
}
void initList(LIST & L)
{
    for(int i = 0; i < L.n; i++)
    {
        visit[i] = false;
    }
}
void DFSGraph1Start(GRAPH * G, int i)
{
    cout << G->vertex[i] << " ";
    visit[i] = true;
    for(int j = 0; j < G->n; j++)
    {
        if(G->edge[i][j] == 1 && !visit[j])
        {
            DFSGraph1Start(G, j);
        }
    }
}
//start DFS in the no visiting
void DFSGraph1(GRAPH &G)
{
    for(int i = 0; i < G.n; i++)
    {
```

```
        if(!visit[i])
        {
            DFSGraph1Start(&G, i);
        }
    }
}
//Non-recursive DFS
void DFSGraph2Start(GRAPH *G, int i)
{
    int STACK[N];
    int top = N;
    top --;
    //push stack
    STACK[top] = i;
    while(top != N)
    {
        int k = STACK[top];
        top ++;
        if(!visit[k])
        {
            cout << G->vertex[k] << " ";
            visit[k] = true;
        }
        for(int i = G->n; i >=0; i--)
        {
            if(!visit[i] && G->edge[k][i])
            {
                top --;
                STACK[top] = i;
            }
        }
    }

}
void DFSGraph2(GRAPH &G)
{
    for(int i = 0; i < G.n; i++)\
    {
        if(!visit[i])
        {
            DFSGraph2Start(&G, i);
        }
    }
```

```
}
void BFSGraphStart(GRAPH *G, int i)
{
    int Q[N], f = 0, r = 0;
    visit[i] = true;
    Q[r] = i;
    r++;
    cout << G->vertex[i] << " ";
    while(f!=r)
    {
        i = Q[f];
        f++;
        for(int k = 0; k < G->n; k++)
        {
            if(!visit[k] && G->edge[i][k])
            {
                cout << G->vertex[k] << " ";
                visit[k] = true;
                Q[r] = k;
                r++;
            }
        }
    }
}
void BFSGraph(GRAPH &G)
{
    for(int i = 0; i < G.n; i++)
    {
        if(!visit[i])
        {
            BFSGraphStart(&G, i);
        }
    }
}
void createList(LIST *G)
{
    int j, k;
    cin >> G->n >> G->e;
    for(int i = 0; i < G->n; i++)
    {
        //input the point, and the first is null
        cin >> G->a[i].vertex;
        G->a[i].first = NULL;
    }
```

```
    for(int i = 0; i < G->e; i++)
    {
        cin >> j >> k;
        LINK *p = new LINK;
        p->v = k;
        p->next = G->a[j].first;
        G->a[j].first = p;
        // undirected graph
        p = new LINK;
        p->v = j;
        p->next = G->a[k].first;
        G->a[k].first = p;
    }
    for(int i = 0; i < G->n; i++)
    {
        cout << G->a[i].vertex;
        LINK *m = G->a[i].first;
        while(m != NULL)
        {
            cout << "->" << G->a[m->v].vertex;
            m = m -> next;
        }
        cout << "->NULL" <<endl;
    }
}
void createList2(LIST *G)
{
    int j, k;
    cin >> G->n >> G->e;
    for(int i = 0; i < G->n; i++)
    {
        //input the point, and the first is null
        cin >> G->a[i].vertex;
        G->a[i].first = NULL;
    }
    for(int i = 0; i < G->e; i++)
    {
        //directed graph
        cin >> j >> k;
        LINK *p = new LINK;
        p->v = k;
        p->next = G->a[j].first;
        G->a[j].first = p;
    }
```

```
    for(int i = 0; i < G->n; i++)
    {
        cout << G->a[i].vertex;
        LINK *m = G->a[i].first;
        while(m != NULL)
        {
            cout << "->" << G->a[m->v].vertex;
            m = m -> next;
        }
        cout << "->NULL" <<endl;
    }
}

void DFSList1Start(LIST *L, int i)
{
    LINK *p;
    cout << L->a[i].vertex << " ";
    visit[i] = true;
    p = L->a[i].first;
    while(p != NULL)
    {
        if(!visit[p->v])
        {
            DFSList1Start(L, p->v);
        }
         p = p->next;
    }
}
void DFSList1(LIST &L)
{
    for(int i = 0; i < L.n; i++)
    {
        if(!visit[i])
        {
            DFSList1Start(&L, i);
        }
    }
    cout << endl;
}

void DFSList2Start(LIST *L, int v)
{
    int STACK[N];
    int top = N;
```

```
    LINK *p = NULL;
    top--;
    STACK[top] = v;
    while(top != N)
    {
        int j = STACK[top];
        top++;
        if(!visit[j])
        {
            cout << L->a[j].vertex << " ";
            visit[j] = true;
        }
        for(p = L->a[j].first; p !=NULL; p=p->next)
        {
            if(!visit[p->v])
            {
                top--;
                STACK[top] = p->v;
            }
        }
    }

}
void DFSList2(LIST &L)
{
    for(int i = 0; i < L.n; i++)
    {

        if(!visit[i])
        {
            DFSList2Start(&L, i);
        }
    }
    cout << endl;
}
void BFSListStart(LIST * L, int v)
{
    int Queue[N];
    int front = 0, rear = 0;
    LINK *p = NULL;
    visit[v] = true;
    Queue[rear] = v;
    rear++;
    cout << L->a[v].vertex << " ";
```

```
    while(front != rear)
    {
        v = Queue[front];
        front++;
        p = L->a[v].first;
        while(p != NULL && !visit[p->v])
        {
            cout << L->a[p->v].vertex << " ";
            visit[p->v] = true;
            Queue[rear] = p->v;
            rear++;
            p = p->next;
        }
    }
}
void BFSList(LIST &L)
{
    for(int i = 0; i < L.n; i++)
    {
        if(!visit[i])
        {
            BFSListStart(&L, i);
        }
    }
}
int main()
{
    struct GRAPH G;
    struct LIST L;
    int n, o=1;
    cout << "1.Adjacency Matrix(undirected graph) " << "\n" <<
"2.Adjacency List(undirected graph)"
    << "\n" << "3.Adjacency Matrix(directed graph) " << "\n" <<
"4.Adjacency List(directed graph)"<< endl;
    cin >> n;
        if(n == 1)
        {
        freopen("1.txt", "r", stdin);
        createGraph(G);
        cout << "DFS1:";
        initGraph(G);
        DFSGraph1(G);
        cout << endl << "DFS2:";
        initGraph(G);
```

```
DFSGraph2(G);
cout << endl << "BFS:";
initGraph(G);
BFSGraph(G);
fclose(stdin);
}
else if(n ==2)
{
freopen("1.txt", "r", stdin);
createList(&L);
cout << "DFS1:";
initList(L);
DFSList1(L);
cout << "DFS2:";
initList(L);
DFSList2(L);
cout << "BFS:";
initList(L);
BFSList(L);
fclose(stdin);
}
else if(n == 3)
{
freopen("1.txt", "r", stdin);
createGraph2(G);
cout << "DFS1:";
initGraph(G);
DFSGraph1(G);
cout << endl << "DFS2:";
initGraph(G);
DFSGraph2(G);
cout << endl << "BFS:";
initGraph(G);
BFSGraph(G);
fclose(stdin);
}
else if(n == 4)
{
freopen("1.txt", "r", stdin);
createList2(&L);
cout << "DFS1:";
initList(L);
DFSList1(L);
cout << "DFS2:";
```

```
        initList(L);
        DFSList2(L);
        cout << "BFS:";
        initList(L);
        BFSList(L);
        fclose(stdin);
        }
        else
        {
            cout << "Thank you!" << endl;
            o = 0;
        }
}
```