实验10图及应用（二）

实验周次：第17周 学时：2学时 地点：学院机房

学号：3190707121 姓名：武新纪 班级：人工智能191 序号：21

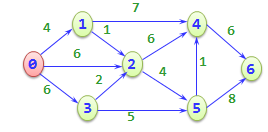
提示：请务必填写 以上个人信息。

实验每题分，共计题。

**【实验目的】**

1.领会图的两种主要存储结构和基于图的基本运算算法设计。

**【实验内容】**

****

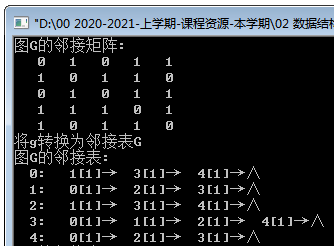
**针对以上带权有向图G。**

实验1.编写一个程序graph.cpp,设计带权图的邻接矩阵与邻接表的创建和输出运算，并在此基础上设计一个主程序exp8-1.cpp完成以下功能。

（1）建立如上图所示的有向图G的邻接矩阵，并输出（输出图的邻接矩阵二维数组形式），格式如下：

（2）销毁图G的邻接矩阵存储结构。

（3）建立如上图所示的有向图G的邻接表，并输出（输出各顶点及其邻边），格式如下：



（4）销毁图G的邻接表。

**程序**

//图的基本运算算法

#include <stdio.h>

#include <malloc.h>

//图的两种存储结构

#define INF 32767               //定义∞

#define MAXV 100                //最大顶点个数

typedef char InfoType;

//以下定义邻接矩阵类型

typedef struct

{   int no;                     //顶点编号

    InfoType info;              //顶点其他信息

} VertexType;                   //顶点类型

typedef struct

{   int edges[MAXV][MAXV];      //邻接矩阵数组

    int n,e;                    //顶点数，边数

    VertexType vexs[MAXV];      //存放顶点信息

} MatGraph;                     //完整的图邻接矩阵类型

//以下定义邻接表类型

typedef struct ANode

{   int adjvex;                 //该边的邻接点编号

    struct ANode \*nextarc;      //指向下一条边的指针

    int weight;                 //该边的相关信息，如权值（用整型表示）

} ArcNode;                      //边节点类型

typedef struct Vnode

{   InfoType info;              //顶点其他信息

    int count;                  //存放顶点入度,仅仅用于拓扑排序

    ArcNode \*firstarc;          //指向第一条边

} VNode;                        //邻接表头节点类型

typedef struct

{   VNode adjlist[MAXV];        //邻接表头节点数组

    int n,e;                    //图中顶点数n和边数e

} AdjGraph;                     //完整的图邻接表类型

//----邻接矩阵的基本运算算法----------------------------------

void CreateMat(MatGraph &g,int A[MAXV][MAXV],int n,int e) //创建图的邻接矩阵

{

    int i,j;

    g.n=n; g.e=e;

    for (i=0;i<g.n;i++)

        for (j=0;j<g.n;j++)

            g.edges[i][j]=A[i][j];

}

void DispMat(MatGraph g)    //输出邻接矩阵g

{

    int i,j;

    for (i=0;i<g.n;i++)

    {

        for (j=0;j<g.n;j++)

            if (g.edges[i][j]!=INF)

                printf("%-3d",g.edges[i][j]);

            else

                printf("%-4s","∞");

        printf("\n");

    }

}

//----邻接表的基本运算算法------------------------------------

void CreateAdj(AdjGraph \*&G,int A[MAXV][MAXV],int n,int e) //创建图的邻接表

{

    int i,j;

    ArcNode \*p;

    G=(AdjGraph \*)malloc(sizeof(AdjGraph));

    for (i=0;i<n;i++)                       //给邻接表中所有头节点的指针域置初值

        G->adjlist[i].firstarc=NULL;

    for (i=0;i<n;i++)                       //检查邻接矩阵中每个元素

        for (j=n-1;j>=0;j--)

            if (A[i][j]!=0 && A[i][j]!=INF) //存在一条边

            {   p=(ArcNode \*)malloc(sizeof(ArcNode));   //创建一个节点p

                p->adjvex=j;

                p->weight=A[i][j];

                p->nextarc=G->adjlist[i].firstarc;  //采用头插法插入节点p

                G->adjlist[i].firstarc=p;

            }

    G->n=n; G->e=n;

}

void DispAdj(AdjGraph \*G)   //输出邻接表G

{

    ArcNode \*p;

    for (int i=0;i<G->n;i++)

    {

        p=G->adjlist[i].firstarc;

        printf("%3d: ",i);

        while (p!=NULL)

        {

            printf("%3d[%d]→",p->adjvex,p->weight);

            p=p->nextarc;

        }

        printf("∧\n");

    }

}

void DestroyAdj(AdjGraph \*&G)   //销毁图的邻接表

{

    ArcNode \*pre,\*p;

    for (int i=0;i<G->n;i++)        //扫描所有的单链表

    {   pre=G->adjlist[i].firstarc; //p指向第i个单链表的首节点

        if (pre!=NULL)

        {   p=pre->nextarc;

            while (p!=NULL)         //释放第i个单链表的所有边节点

            {   free(pre);

                pre=p; p=p->nextarc;

            }

            free(pre);

        }

    }

    free(G);                        //释放头节点数组

}

//文件exp8-1.cpp

int main()

{

    MatGraph g;

    AdjGraph \*G;

    int A[MAXV][MAXV] = {

        {0, 4, 6, 6, INF, INF, INF},

        {INF, 0, 1, INF, 7, INF, INF},

        {INF, INF, 0, INF, 6, 4, INF},

        {INF, INF, 2, 0, INF, 5, INF},

        {INF, INF, INF, INF, 0, INF, 6},

        {INF, INF, INF, INF, 1, 0, 8},

        {INF, INF, INF, INF, INF, INF, 0}

    };

    int n = 7, e = 12;

    CreateMat(g, A, n, e);

    printf("(1)图G的邻接矩阵:\n");

    DispMat(g);

    CreateAdj(G, A, n, e);

    printf("(2)图G的邻接表:\n");

    DispAdj(G);

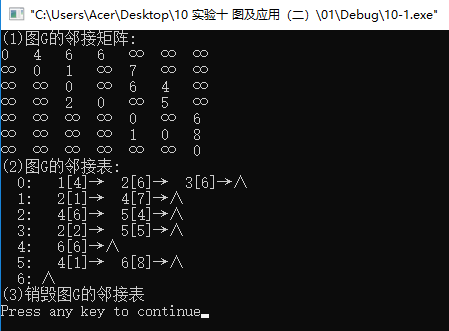
    printf("(3)销毁图G的邻接表\n");

    DestroyAdj(G);

    return 0;

}

**运行结果:**

****

实验2.编写一个程序graph\_search.cpp实现基于图的邻接表存储结构的图的两种遍历算法，深度优先遍历的递归和非递归算法、广度优先遍历的非递归算法，并在此基础上设计一个程序exp8-2.cpp完成以下功能：

（1）调用深度优先遍历的递归算法输出上图从顶点0开始的深度优先遍历序列。

（2）调用深度优先遍历的非递归算法输出上图从顶点0开始的深度优先遍历序列。

（3）调用广度优先遍历的非递归算法输出上图从顶点0开始的广度优先遍历序列。

**程序：**

// 文件graph\_search.cpp

#include "graph.cpp"         //包含图的存储结构及基本运算算法

int visited[MAXV];       //全局数组

void DFS(AdjGraph \*G, int v) //递归深度优先遍历算法

{

    ArcNode \*p;

    visited[v] = 1;             //置已访问标记

    printf("%3d", v);           //输出被访问顶点的编号

    p = G->adjlist[v].firstarc; //p指向顶点v的第一条弧的弧头结点

    while (p != NULL)

    {

        if (visited[p->adjvex] == 0) //若p->adjvex顶点未访问,递归访问它

            DFS(G, p->adjvex);

        p = p->nextarc; //p指向顶点v的下一条弧的弧头结点

    }

}

void DFS1(AdjGraph \*G, int v) //非递归深度优先遍历算法

{

    ArcNode \*p;

    int St[MAXV];

    int top = -1, w, x, i;

    for (i = 0; i < G->n; i++)

        visited[i] = 0; //顶点访问标志均置成0

    printf("%3d", v);   //访问顶点v

    visited[v] = 1;     //置顶点v已访问

    top++;

    St[top] = v;     //将顶点v进栈

    while (top > -1) //栈不空循环

    {

        x = St[top];                //取栈顶顶点x作为当前顶点

        p = G->adjlist[x].firstarc; //找顶点x的第一个相邻点

        while (p != NULL)

        {

            w = p->adjvex;       //x的相邻点为w

            if (visited[w] == 0) //若顶点w没有访问

            {

                printf("%3d", w); //访问顶点w

                visited[w] = 1;   //置顶点w已访问

                top++;            //将顶点w进栈

                St[top] = w;

                break; //退出循环，即再处理栈顶的顶点(体现后进先出)

            }

            p = p->nextarc; //找顶点x的下一个相邻点

        }

        if (p == NULL)

            top--; //若顶点x再没有相邻点，将其退栈

    }

    printf("\n");

}

void BFS(AdjGraph \*G, int v) //广度优先遍历算法

{

    ArcNode \*p;

    int queue[MAXV], front = 0, rear = 0; //定义环形队列并初始化

    int visited[MAXV];                    //定义存放顶点的访问标志的数组

    int w, i;

    for (i = 0; i < G->n; i++)

        visited[i] = 0; //访问标志数组初始化

    printf("%3d", v);   //输出被访问顶点的编号

    visited[v] = 1;     //置已访问标记

    rear = (rear + 1) % MAXV;

    queue[rear] = v;      //v进队

    while (front != rear) //若队列不空时循环

    {

        front = (front + 1) % MAXV;

        w = queue[front];           //出队并赋给w

        p = G->adjlist[w].firstarc; //找顶点w的第一个相邻点

        while (p != NULL)

        {

            if (visited[p->adjvex] == 0) //若相邻点未被访问

            {

                printf("%3d", p->adjvex); //访问相邻点

                visited[p->adjvex] = 1;   //置该顶点已被访问的标志

                rear = (rear + 1) % MAXV; //该顶点进队

                queue[rear] = p->adjvex;

            }

            p = p->nextarc; //找下一个相邻点

        }

    }

    printf("\n");

}

//文件exp8-2.cpp

#include " graph\_search.cpp

int main()

{

    AdjGraph \*G;

    int A[MAXV][MAXV] = {

        {0, 4, 6, 6, INF, INF, INF},

        {INF, 0, 1, INF, 7, INF, INF},

        {INF, INF, 0, INF, 6, 4, INF},

        {INF, INF, 2, 0, INF, 5, INF},

        {INF, INF, INF, INF, 0, INF, 6},

        {INF, INF, INF, INF, 1, 0, 8},

        {INF, INF, INF, INF, INF, INF, 0}};

    int n = 7, e = 12;

    CreateAdj(G, A, n, e);

    printf("图G的邻接表:\n");

    DispAdj(G);

    printf("从顶点0开始的DFS(递归算法):\n");

    DFS(G, 0);

    printf("\n");

    printf("从顶点0开始的DFS(非递归算法):\n");

    DFS1(G, 0);

    printf("从顶点0开始的BFS:\n");

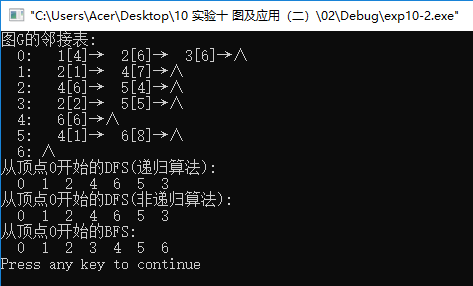
    BFS(G, 0);

    DestroyAdj(G);

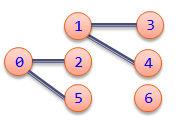
    return 0;

}

**运行结果:**



给出以下非连通图：



实验3.编写一个程序unconnected\_graph\_search.cpp实现基于非连通图的邻接表存储结构的图的两种遍历算法：深度优先遍历算法和广度优先遍历算法，并在此基础上设计一个程序exp8-3.cpp完成以下功能：

（1）输出以上非连通图的深度优先遍历序列。

（2）输出以上非连通图的广度优先遍历序列。

**程序：**

//文件unconnected\_graph\_search.cpp

#include <stdio.h>

#include <malloc.h>

#define MAXV 100 //最大顶点个数

typedef char InfoType;     //定义图结点数据域类型

int visited1[MAXV] = {0};  //全局数组1,为深度优先遍历算法使用

int visited2[MAXV] = {0};  //全局数组2,为广度优先遍历算法使用

//以下定义邻接表类型

typedef struct ANode

{

    int adjvex;            //该边的邻接点编号

    struct ANode \*nextarc; //指向下一条边的指针

    int weight;            //该边的相关信息，如权值（用整型表示）

} ArcNode;                 //边节点类型

typedef struct Vnode

{

    InfoType info;         //顶点其他信息

    int count;             //存放顶点入度,仅仅用于拓扑排序

    ArcNode \*firstarc;     //指向第一条边

} VNode;                   //邻接表头节点类型

typedef struct

{

    VNode adjlist[MAXV];   //邻接表头节点数组

    int n, e;              //图中顶点数n和边数e

} AdjGraph;                //完整的图邻接表类型

//创建图的邻接表

void CreateAdj(AdjGraph \*&G, int A[MAXV][MAXV], int n, int e)

{

    int i, j;

    ArcNode \*p;

    G = (AdjGraph \*)malloc(sizeof(AdjGraph));

    for (i = 0; i < n; i++) //给邻接表中所有头节点的指针域置初值

        G->adjlist[i].firstarc = NULL;

    for (i = 0; i < n; i++) //检查邻接矩阵中每个元素

        for (j = n - 1; j >= 0; j--)

            if (A[i][j] != 0 && A[i][j] != 0) //存在一条边

            {

                p = (ArcNode \*)malloc(sizeof(ArcNode)); //创建一个节点p

                p->adjvex = j;

                p->weight = A[i][j];

                p->nextarc = G->adjlist[i].firstarc; //采用头插法插入节点p

                G->adjlist[i].firstarc = p;

            }

    G->n = n;

    G->e = n;

}

//输出邻接表G

void DispAdj(AdjGraph \*G)

{

    ArcNode \*p;

    for (int i = 0; i < G->n; i++)

    {

        p = G->adjlist[i].firstarc;

        printf("%3d: ", i);

        while (p != NULL)

        {

            printf("%3d[%d]→", p->adjvex, p->weight);

            p = p->nextarc;

        }

        printf("∧\n");

    }

}

//销毁图的邻接表

void DestroyAdj(AdjGraph \*&G)

{

    ArcNode \*pre, \*p;

    for (int i = 0; i < G->n; i++) //扫描所有的单链表

    {

        pre = G->adjlist[i].firstarc; //p指向第i个单链表的首节点

        if (pre != NULL)

        {

            p = pre->nextarc;

            while (p != NULL) //释放第i个单链表的所有边节点

            {

                free(pre);

                pre = p;

                p = p->nextarc;

            }

            free(pre);

        }

    }

    free(G); //释放头节点数组

}

//递归深度优先遍历算法

void DFS(AdjGraph \*G, int v)

{

    ArcNode \*p;

    visited1[v] = 1;            //置已访问标记

    printf("%3d", v);           //输出被访问顶点的编号

    p = G->adjlist[v].firstarc; //p指向顶点v的第一条弧的弧头结点

    while (p != NULL)

    {

        if (visited1[p->adjvex] == 0) //若p->adjvex顶点未访问,递归访问它

            DFS(G, p->adjvex);

        p = p->nextarc; //p指向顶点v的下一条弧的弧头结点

    }

}

//广度优先遍历算法

void BFS(AdjGraph \*G, int v)

{

    ArcNode \*p;

    int queue[MAXV], front = 0, rear = 0; //定义环形队列并初始化

    int visited[MAXV];                    //定义存放顶点的访问标志的数组

    int w, i;

    for (i = 0; i < G->n; i++)

        visited[i] = 0; //访问标志数组初始化

    printf("%3d", v);   //输出被访问顶点的编号

    visited[v] = 1;     //置已访问标记

    rear = (rear + 1) % MAXV;

    queue[rear] = v;      //v进队

    while (front != rear) //若队列不空时循环

    {

        front = (front + 1) % MAXV;

        w = queue[front];           //出队并赋给w

        p = G->adjlist[w].firstarc; //找顶点w的第一个相邻点

        while (p != NULL)

        {

            if (visited[p->adjvex] == 0) //若相邻点未被访问

            {

                printf("%3d", p->adjvex); //访问相邻点

                visited[p->adjvex] = 1;   //置该顶点已被访问的标志

                rear = (rear + 1) % MAXV; //该顶点进队

                queue[rear] = p->adjvex;

            }

            p = p->nextarc; //找下一个相邻点

        }

    }

    printf("\n");

}

// 深度优先遍历非连通无向图算法

void DFS\_NC(AdjGraph \*G)

{

    int i;

    for (i = 0; i < G->n; i++) //遍历所有未访问过的顶点

        if (visited1[i] == 0)  //未被访问过的顶点，则调用DFS

            DFS(G, i);

}

// 广度优先遍历非连通无向图算法

void BFS\_NC(AdjGraph \*G)

{

    int i;

    for (i = 0; i < G->n; i++) //遍历所有未访问过的顶点

        if (visited2[i] == 0)  //未被访问过的顶点，则调用BFS

            BFS(G, i);

}

//文件exp8-3.cpp

#include "unconnected\_graph\_search.cpp"

int main()

{

    AdjGraph \*G;

    int A[][MAXV] =

        {{0, 0, 1, 0, 0, 1, 0},

         {0, 0, 0, 1, 1, 0, 0},

         {1, 0, 0, 0, 0, 0, 0},

         {0, 1, 0, 0, 0, 0, 0},

         {0, 1, 0, 0, 0, 0, 0},

         {1, 0, 0, 0, 0, 0, 0},

         {0, 0, 0, 0, 0, 0, 0}};

    int n = 7, e = 4;

    CreateAdj(G, A, n, e);

    printf("图G的邻接表:\n");

    DispAdj(G);

    printf("深度优先遍历序列为:\n");

    DFS\_NC(G);

    printf("\n");

    printf("广度优先遍历序列为:\n");

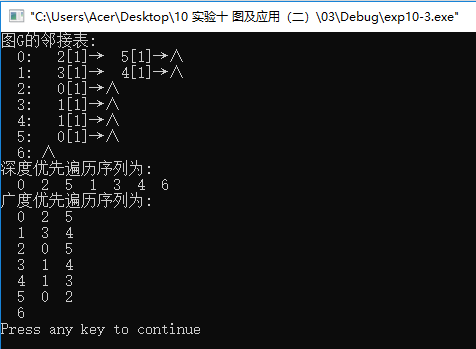
    BFS\_NC(G);

    DestroyAdj(G);

    return 0;

}

**运行结果：**

****