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：图型结构的建立与搜索

实验题目：图的存储结构的建立与搜索

实验日期：2021.01.05

班级：1803501

学号：L170300901

姓名：卢兑玩

设计成绩	报告成绩	指导老师

一、实验目的

1. 掌握图的邻接矩阵、邻接表等不同存储形式的表示方法。
2. 掌握图的两种不同遍历方法的基本思想并能编程实现。
3. 掌握构造最小生成树的两种算法思想，并能编程实现。
4. 掌握求单源最短路径和任意两顶点之间的最短路径的算法。
5. 掌握求关键路径的算法，并能编程实现。
6. 能够灵活运用图的相关算法解决相应的实际问题。

二、实验要求及实验环境

实验要求：

1．分别实现图的邻接矩阵、邻接表存储结构的建立算法，分析和比较各建立算法的时间复杂度以及存储结构的空间占用情况；

2．实现图的邻接矩阵、邻接表两种存储结构的相互转换算法；

3．在上述两种存储结构上，分别实现图的深度优先搜索（递归和非递归)和广度优先搜索算法。并以适当的方式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．分析搜索算法的时间复杂度；

5．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；

6．软件功能结构安排合理，界面友好，便于使用。

实验环境

Windows10, Codeblocks20.03

三、设计思想（本程序中的用到的所有数据类型的定义，主程序的流程图及各程序模块之间的调用关系）

首先从文件读取数据建立三个结构。

① 邻接矩阵

读取顶点的个数 v，边的个数 e。读取两个顶点之间的权力。

时间复杂度：O（n2）

占用空间情况：v2

② 邻接表

读取顶点的个数 v，边的个数 e。读取两个顶点之间的权力。

时间复杂度为：O（n2）

占用空间情况：v+2e

③ 读取顶点的个数 v，边的个数 e。读取两个顶点之间的权力。

时间复杂度为：O（n2）

占用空间情况：v+e

其次从各结构转换成另外结构。这个时候利用上边读取文件时用到的建立结构方法实现转换功能。并且转换之前要判断有没有结构能转换，如果不存在 A，就不能从 A 到 B 转换。

然后对每个结构实现两种搜索算法，深度优先搜索算法用递归和非递归实现，广度优先搜索算法利用非递归实现。

④ 深度优先搜索算法 - 邻接矩阵

非递归利用栈实现算法，递归的直接调用递归函数就能实现。两个方法都需要建一个 Visit 数组，判断是不是访问过的顶点。因为邻接矩阵用数组，所以最多运行从 i 从 0 到 v 次。

非递归的时间复杂度：O（n2）

递归的时间复杂度： O（n2）

⑤ 深度优先搜索算法 - 邻接表

非递归利用栈实现算法，递归的直接调用递归函数就能实现。两个方法都需要建一个 Visit 数组，判断是不是访问过的顶点。因为邻接表利用链表，则要判断此表到最后为 NULL 的情况下结束。

非递归的时间复杂度：O（n+e）

递归的时间复杂度： O（n+e）

⑥ 深度优先搜索算法 - 邻接多重表

非递归利用栈实现算法，递归的直接调用递归函数就能实现。两个方法都需要建一个 Visit 数组，判断是不是访问过的顶点。因为邻接多重表是链表，而且是左右后续点的链表。所以每次访问结点要判断进行下一个右节点访问还是左节点访问。

非递归的时间复杂度：0（n+e）

递归的时间复杂度： 0（n+e）

⑦ 广度优先搜索算法

广度优先搜索算法利用非递归函数实现，利用队列的性质能实现广度优先搜索算法的实现。跟上边的深度优先搜索算法相似，要建立 Visit 数组判断他是不是访问过的点。而且跟上边的算法相似每个结构的算法有一些的区别，不过大概的代码相同。

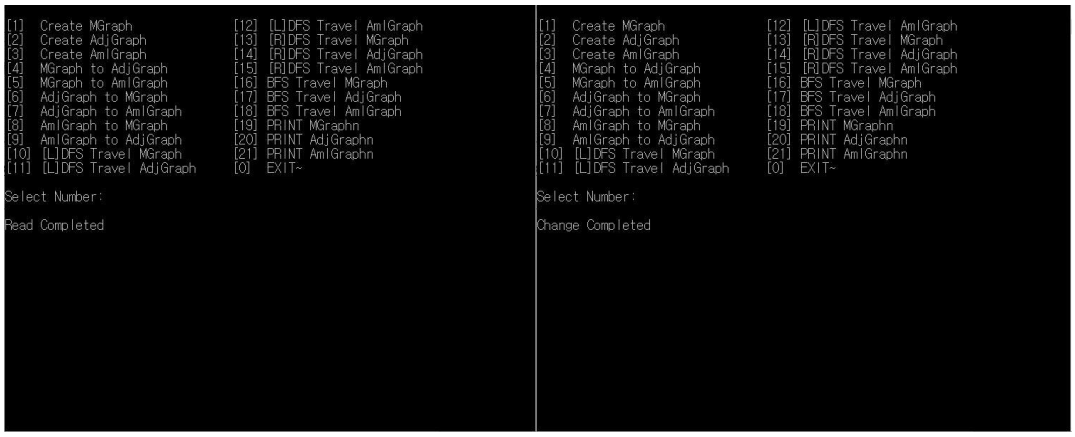
邻接矩阵的时间复杂度：0（n2）

邻接表的时间复杂度：0（n+e）

邻接多重表的时间复杂度：0（n2）

四、测试结果

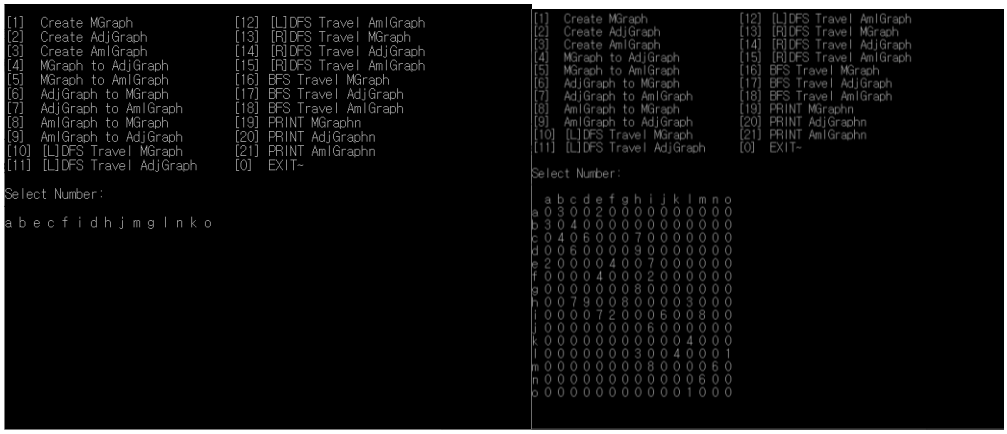
图片不全，为了看的方便只放了每个功能的一个图片。



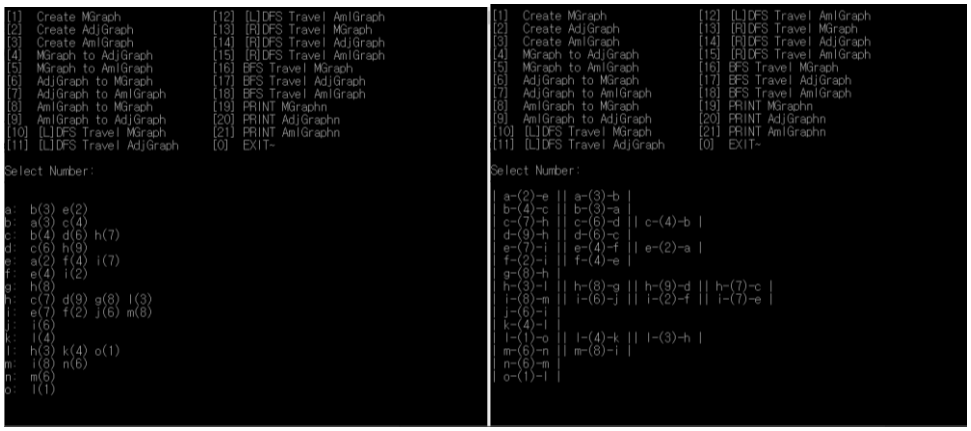
左边为读取文件建立邻接矩阵，右边是用建立的矩阵转换成邻接表

左边为用非递归算法显示邻接矩阵的深度优先搜索，右边为用递归算法显示邻接矩阵的深度优先

搜索



左边为用非递归算法显示广度优先搜索，右边为打印邻接矩阵。



左边为打印邻接表，右边为打印多重邻接表。

五、经验体会与不足

通过这个实验知道了这个的特点是特殊一个定点通往其他方向的所有最短路径的概念

具体操作方式是

1. 设定开始 node
2. 以出发 node 为准,保存各个 node 最小费用
3. 储存没有访问的 NODE 中费用最少的 NODE
4. 计算费用最少的

六、附录：源代码（带注释）

```
#include <stdio.h>
```

```
#include <stdlib.h>
#include <string.h>
#define NumMax 50
#include <windows.h>

#define FileAddress "D://shiyan3.txt"

typedef struct
{
    char VData[NumMax];
    int EData[NumMax][NumMax];
    int v,e;
}MGraph;

typedef struct node
{
    int adjvex,cost;
    struct node* next;
}EdgeNode;
typedef struct
{
    char vertax;
    EdgeNode* firstedge;
}VertexNode;
typedef struct
{
    VertexNode vexlist[NumMax];
    int v,e;
}AdjGraph;
```

```
typedef struct EBox
{
    int ivex,jvex,cost;
    struct EBox *ilink,*jlink;
    char *info;
}EBox;
typedef struct VexBox
{
    char Vertex;
    EBox *firstedge;
}VexBox;
typedef struct
{
    VexBox adjmulist[NumMax];
    int v,e;
}AmlGraph;

typedef struct struct_* stack_0;
typedef struct struct_* queue_0;
typedef struct struct_
{
    int dat;
    struct struct_* next;
}struct_;
stack_0 stack_h;
queue_0 queue_h;

typedef enum {visited,unvisited}VisitIf;
```

```
VisitIf Visit[NumMax];



void Select(MGraph*,AdjGraph*,AmlGraph*);   //选择项目界面
void PrintMGraph(MGraph*);  //打印邻接矩阵
void PrintAdjGraph(AdjGraph*);  //打印邻接表
void PrintAmlGraph(AmlGraph*);  //打印邻接多重表
void CreateMGraph(MGraph*); //从文件读数据生成邻接矩阵
void CreateAdjGraph(AdjGraph*); //从文件读数据生成邻接表
void CreateAmlGraph(AmlGraph*); //从文件读数据生成邻接多重表
int LocateVex(AmlGraph*,char);  //找输入字符的位置
void MtoAdj(MGraph*,AdjGraph*); //把邻接矩阵转换成邻接表
void MtoAml(MGraph*,AmlGraph*); //把邻接矩阵转换成邻多重接表
void AdjtoM(AdjGraph*,MGraph*); //把邻接表转换成邻接矩阵
void AdjtoAml(AdjGraph*,AmlGraph*); //把邻接表转换成邻接多重表
void AmltoM(AmlGraph*,MGraph*); //把邻接多重表转换成邻接矩阵
void AmltoAdj(AmlGraph*,AdjGraph*); //把邻接多重表转换成邻接表
void pushst(int);   //进栈
int popst();    //出栈
void MGraphLoop(MGraph*);   //用循环遍历深度优先搜索邻接矩阵
void AdjGraphLoop(AdjGraph*);   //用循环遍历深度优先搜索邻接表
void AmlGraphLoop(AmlGraph*);   //用循环遍历深度优先搜索邻接多重表
void MGraphRecursive(MGraph*);  //利用递归函数深度优先搜索邻接矩阵
void DFSM(MGraph*,int); //递归型深度优先搜索邻接矩阵
void AdjGraphRecursive(AdjGraph*);  //利用递归函数深度优先搜索邻接表
void DFSAdj(AdjGraph*,int); //递归型深度优先搜索邻接表
void AmlGraphRecursive(AmlGraph*);  //利用递归函数深度优先搜索邻接多重表
void DFSAml(AmlGraph*,int); //递归型深度优先搜索邻接多重表
void ENQUEUE(int);  //进队列
```

```
int DEQUEUE();  //出队列
int EmptyQ();   //判断队列是不是空
void BFSM(MGraph*); //广度搜索邻接矩阵
void BFSAdj(AdjGraph*); //广度搜索邻接表
void BFSAml(AmlGraph*); //广度搜索邻接多重表
void gotoxy(int,int);



void gotoxy(int x,int y)
{
    COORD pos = { x - 1, y - 1 };//커서가 X 좌표에서 -1 한값. Y 좌표에서 -1 한 값으로 이동
    SetConsoleCursorPosition(GetStdHandle(STD_OUTPUT_HANDLE),   pos);//   WIN32API 함수입니다. 이건 알필요 없어요
}

void BFSAml(AmlGraph* G)
{
    if(G->v==-1)
    {
        printf("No Have AmlGraph!\n");
        return;
    }
    int i,p,j;
    EBox* m;
    for(i=0; i<G->v; i++)
        Visit[i]=1;
    for(i=0,p=i; i<G->v; i++,p=i)
    {
```

```
if(!Visit[i])
    continue;
ENQUEUE(i);
printf("%c",G->adjmulist[i].Vertex);
Visit[i]=0;
while(EmptyQ()!=-1)
{
    p=DEQUEUE();
    m=G->adjmulist[p].firstedge;
    while(m)
    {
        if(p==m->ivex)
        {
            if(Visit[m->jvex])
            {
                j=m->jvex;
                ENQUEUE(j);
                printf(" %c",G->adjmulist[j].Vertex);
                Visit[j]=0;
            }
            m=m->ilink;
        }
        else
        {
            if(Visit[m->ivex])
            {
                j=m->ivex;
                ENQUEUE(j);
                printf(" %c",G->adjmulist[j].Vertex);
```

```
                        Visit[j]=0;
                    }
                    m=m->jlink;
                }
            }
        }
    }
}

void BFSAdj(AdjGraph* G)
{
    if(G->v==-1)
    {
        printf("No Have AdjGraph!\n");
        return;
    }
    int i,p,j;
    EdgeNode* m;
    for(i=0; i<G->v; i++)
        Visit[i]=1;
    for(i=0,p=i; i<G->v; i++,p=i)
    {
        if(!Visit[i])
            continue;
        ENQUEUE(i);
        printf("%c",G->vexlist[i].vertax);
        Visit[i]=0;
        while(EmptyQ()!=-1)
        {
```

```
            p=DEQUEUE();
            m=G->vexlist[p].firstedge;
            while(m)
            {
                if(Visit[m->adjvex]&&m->cost!=0)
                {
                    j=m->adjvex;
                    ENQUEUE(j);
                    printf(" %c",G->vexlist[j].vertax);
                    Visit[j]=0;
                }
                m=m->next;
            }
        }
    }
}

void BFSM(MGraph* G)
{
    if(G->v==-1)
    {
        printf("No Have MGraph!\n");
        return;
    }
    int i,p,j;
    for(i=0; i<G->v; i++)
        Visit[i]=1;
    for(i=0,p=i; i<G->v; i++,p=i)
    {
```

```
        if(!Visit[i])
            continue;
        ENQUEUE(i);
        printf("%c",G->VData[i]);
        Visit[i]=0;
        while(EmptyQ()!=-1)
        {
            p=DEQUEUE();
            for(j=0; j<G->v; j++)
            {
                if(G->EData[p][j]!=0&&Visit[j])
                {
                    ENQUEUE(j);
                    printf(" %c",G->VData[j]);
                    Visit[j]=0;
                }
            }
        }
    }
}

int EmptyQ()
{
    if(!queue_h->next)
        return -1;
    else
        return 1;
}
```

```
int DEQUEUE()
{
    if(!queue_h->next)
        return -1;
    else
    {
        int i;
        queue_0 p=queue_h->next;
        i=p->dat;
        queue_h->next=p->next;
        free(p);
        return i;
    }

}

void ENQUEUE(int i)
{
    queue_0 p=(queue_0)malloc(sizeof(struct_));
    p->dat=i;
    p->next=NULL;
    queue_0 q=queue_h;
    while(q->next)
    {
        q=q->next;
    }
    q->next=p;
}
```

```
void DFSAml(AmlGraph* G,int i)
{
    int j;
    Visit[i]=0;
    printf("%c ",G->adjmulist[i].Vertex);
    EBox* m;
    m=G->adjmulist[i].firstedge;
    while(m)
    {
        if(i==m->ivex)
        {
            j=m->jvex;
            if(Visit[j]&&m->cost!=0)
                DFSAml(G,j);
            m=m->ilink;
        }
        else
        {
            j=m->ivex;
            if(Visit[j]&&m->cost!=0)
                DFSAml(G,j);
            m=m->jlink;
        }
    }
}

void AmlGraphRecursive(AmlGraph* G)
{
    if(G->v==-1)
```

```
    {
        printf("No Have AmlGraph!\n");
        return;
    }
    int i;
    for(i=0; i<G->v; i++)
        Visit[i]=1;
    for(i=0; i<G->v; i++)
        if(Visit[i])
            DFSAml(G,i);
}

void DFSAdj(AdjGraph* G,int i)
{
    int j;
    Visit[i]=0;
    printf("%c ",G->vexlist[i].vertax);
    EdgeNode* m;
    m=G->vexlist[i].firstedge;
    while(m)
    {
        j=m->adjvex;
        if(Visit[j]&&m->cost!=0)
            DFSAdj(G,j);
        m=m->next;
    }
}

void AdjGraphRecursive(AdjGraph* G)
```

```
{
    if(G->v==-1)
    {
        printf("No Have AdjGraph!\n");
        return;
    }
    int i;
    for(i=0; i<G->v; i++)
        Visit[i]=1;
    for(i=0; i<G->v; i++)
        if(Visit[i])
            DFSAdj(G,i);
}

void DFSM(MGraph* G,int i)
{
    int j;
    Visit[i]=0;
    printf("%c ",G->VData[i]);
    for(j=0; j<G->v; j++)
        if(G->EData[i][j]!=0&&Visit[j])
            DFSM(G,j);
}

void MGraphRecursive(MGraph* G)
{
    if(G->v==-1)
    {
        printf("No Have MGraph!\n");
```

```
        return;
    }
    int i;
    for(i=0; i<G->v; i++)
        Visit[i]=1;
    for(i=0; i<G->v; i++)
        if(Visit[i])
            DFSM(G,i);
}

void AmlGraphLoop(AmlGraph* G)
{
    if(G->v==-1)
    {
        printf("No Have AmlGraph!\n");
        return;
    }
    int i,j,p,flag;
    EBox* m;
    for(i=0; i<G->v; i++)
        Visit[i]=1;
    for(i=0,p=i; i<G->v; i++,p=i)
    {
        if(!Visit[i])
            continue;
        Visit[i]=0;
        printf("%c",G->adjmulist[i].Vertex);
        while(p!=-1)
        {
```

```
flag=1;
m = G->adjmulist[p].firstedge;
while(m)
{
    if(m->ivex==p)
    {
        j=m->jvex;
        if(Visit[j])
        {
            pushst(p);
            Visit[j]=0;
            printf(" %c",G->adjmulist[j].Vertex);
            flag=0; p=j;
            break;
        }
        m=m->ilink;
    }
    else
    {
        j=m->ivex;
        if(Visit[j])
        {
            pushst(p);
            Visit[j]=0;
            printf(" %c",G->adjmulist[j].Vertex);
            flag=0; p=j;
            break;
        }
        m=m->jlink;
```

```
                }
            }
            if(flag)
                p=popst();
        }
    }
}

void AdjGraphLoop(AdjGraph* G)
{
    if(G->v==-1)
    {
        printf("No Have AdjGraph!\n");
        return;
    }
    int i,j,p,flag;
    EdgeNode* m;
    for(i=0; i<G->v; i++)
        Visit[i]=1;
    for(i=0,p=i; i<G->v; i++,p=i)
    {
        if(!Visit[i])
            continue;
        Visit[i]=0;
        printf("%c",G->vexlist[i].vertax);
        while(p!=-1)
        {
            flag=1;
            m = G->vexlist[p].firstedge;
```

```
            while(m)
            {
                j=m->adjvex;
                if(Visit[j])
                {
                    pushst(p);
                    Visit[j]=0;
                    printf(" %c",G->vexlist[j].vertax);
                    flag=0; p=j;
                    break;
                }
                m=m->next;
            }
            if(flag)
                p=popst();
        }
    }
}

void MGraphLoop(MGraph* G)
{
    if(G->v==-1)
    {
        printf("No Have MGraph!\n");
        return;
    }
    int i,j,p,flag;
    for(i=0; i<G->v; i++)
        Visit[i]=1;
```

```
    for(i=0,p=i; i<G->v; i++,p=i)
    {
        if(!Visit[i])
            continue;
        Visit[i]=0;
        printf("%c",G->VData[i]);
        while(p!=-1)
        {
            flag=1;
            for(j=0; j<G->v; j++)
            {
                if(Visit[j]&&G->EData[p][j]!=0)
                {
                    pushst(p);
                    Visit[j]=0;
                    printf(" %c",G->VData[j]);
                    flag=0; p=j;
                    break;
                }
            }
            if(flag)
                p=popst();
        }
    }
}

int popst()
{
    int i;
```

```
    if(!stack_h->next)
        return -1;
    else
    {
        stack_0 p=stack_h->next;
        i=p->dat;
        stack_h->next=p->next;
        free(p);
        return i;
    }
}

void pushst(int i)
{
    stack_0 p=(stack_0)malloc(sizeof(struct_));
    p->dat=i;
    p->next=stack_h->next;
    stack_h->next=p;
}

void AmltoAdj(AmlGraph* x,AdjGraph* y)
{
    int i,j;
    if(x->v==-1)
    {
        printf("No Have AmlGraph!\n");
        return;
    }
    y->v=x->v; y->e=x->e;
```

```
EdgeNode* m;
for(i=0; i<y->v; i++)
{
    y->vexlist[i].vertax=x->adjmulist[i].Vertex;
    while(y->vexlist[i].firstedge)
    {
        m=y->vexlist[i].firstedge;
        y->vexlist[i].firstedge=y->vexlist[i].firstedge->next;
        free(m);
    }
}
EBox* s;
for(i=0; i<x->v; i++)
{
    s=x->adjmulist[i].firstedge;
    while(s)
    {
        EdgeNode* p=(EdgeNode*)malloc(sizeof(EdgeNode));
        if(s->ivex==i)
        {
            if(y->vexlist[i].firstedge)
            {
                EdgeNode* m=y->vexlist[i].firstedge;
                while(m->next)
                    m=m->next;
                p->adjvex=s->jvex;
                p->cost=s->cost;
                p->next=NULL;
                m->next=p;
```

```
    }
    else
    {
        p->adjvex=s->jvex;
        p->cost=s->cost;
        p->next=NULL;
        y->vexlist[i].firstedge=p;
    }
    s=s->ilink;
}
else
{
    if(y->vexlist[i].firstedge)
    {
        EdgeNode* m=y->vexlist[i].firstedge;
        while(m->next)
            m=m->next;
        p->adjvex=s->ivex;
        p->cost=s->cost;
        p->next=NULL;
        m->next=p;
    }
    else
    {
        p->adjvex=s->ivex;
        p->cost=s->cost;
        p->next=NULL;
        y->vexlist[i].firstedge=p;
    }
```

```
                s=s->jlink;
            }
        }
    }
    printf("Change Completed");
}

void AmltoM(AmlGraph* x,MGraph* y)
{
    int i,j;
    if(x->v==-1)
    {
        printf("No Have AmlGraph!\n");
        return;
    }
    y->v=x->v; y->e=x->e;
    for(i=0; i<y->v; i++)
    {
        y->VData[i]=x->adjmulist[i].Vertex;
        for(j=0; j<y->v; j++)
            y->EData[i][j]=0;
    }
    EBox* s;
    for(i=0; i<x->v; i++)
    {
        s=x->adjmulist[i].firstedge;
        while(s)
        {
            if(s->ivex==i)
```

```
            {
                y->EData[i][s->jvex]=s->cost;
                y->EData[s->jvex][i]=s->cost;
                s=s->ilink;
            }
            else
            {
                y->EData[i][s->ivex]=s->cost;
                y->EData[s->ivex][i]=s->cost;
                s=s->jlink;
            }
        }
    }
    printf("Change Completed");
}

void AdjtoAml(AdjGraph* x,AmlGraph* y)
{
    int i,j;
    if(x->v==-1)
    {
        printf("No Have AdjGraph!\n");
        return;
    }
    y->v=x->v; y->e=x->e;
    for(i=0; i<y->v; i++)
    {
        y->adjmulist[i].Vertex=x->vexlist[i].vertax;
        y->adjmulist[i].firstedge=NULL;
```

```
    }
    for(i=0; i<y->v; i++)
    {
        EdgeNode* s=x->vexlist[i].firstedge;
        while(s)
        {
            j=s->adjvex;
            EBox* p=(EBox*)malloc(sizeof(EBox));
            p->ivex=i; p->jvex=j; p->cost=s->cost;
            p->ilink=y->adjmulist[i].firstedge;
            p->jlink=y->adjmulist[j].firstedge;
            y->adjmulist[i].firstedge=y->adjmulist[j].firstedge=p;
            s=s->next;
        }
    }
    printf("Change Completed");
}

void AdjtoM(AdjGraph* x,MGraph* y)
{
    int i,j;
    if(x->v==-1)
    {
        printf("No Have AdjGraph!\n");
        return;
    }
    y->v=x->v; y->e=x->e;
    for(i=0; i<y->v; i++)
    {
```

```
        y->VData[i]=x->vexlist[i].vertax;
        for(j=0; j<y->v; j++)
            y->EData[i][j]=0;
    }
    EdgeNode* p;
    for(i=0; i<y->v; i++)
    {
        p=x->vexlist[i].firstedge;
        while(p)
        {
            y->EData[i][p->adjvex]=p->cost;
            p=p->next;
        }
    }
    printf("Change Completed");
}

void MtoAml(MGraph* x,AmlGraph* y)
{
    int i,j;
    if(x->v==-1)
    {
        printf("No Have MGraph!\n");
        return;
    }
    y->v=x->v; y->e=x->e;
    for(i=0; i<y->v; i++)
    {
        y->adjmulist[i].Vertex=x->VData[i];
```

```
        y->adjmulist[i].firstedge=NULL;
    }
    for(i=0; i<y->v; i++)
    {
        for(j=i; j<y->v; j++)
        {
            if(x->EData[i][j]!=0)
            {
                EBox* p=(EBox*)malloc(sizeof(EBox));
                p->ivex=i; p->jvex=j; p->cost=x->EData[i][j];
                p->ilink=y->adjmulist[i].firstedge;
                p->jlink=y->adjmulist[j].firstedge;
                y->adjmulist[i].firstedge=y->adjmulist[j].firstedge=p;
            }
        }
    }
    printf("Change Completed");
}

void MtoAdj(MGraph* x,AdjGraph* y)
{
    int i,j;
    if(x->v==-1)
    {
        printf("No Have MGraph!\n");
        return;
    }
    y->v=x->v; y->e=x->e;
    EdgeNode* m;
```

```
for(i=0; i<y->v; i++)
{
    y->vexlist[i].vertax=x->VData[i];
    while(y->vexlist[i].firstedge)
    {
        m=y->vexlist[i].firstedge;
        y->vexlist[i].firstedge=y->vexlist[i].firstedge->next;
        free(m);
    }
}
for(i=0; i<x->v; i++)
{
    for(j=i; j<x->v; j++)
    {
        if(x->EData[i][j]!=0)
        {
            EdgeNode* p=(EdgeNode*)malloc(sizeof(EdgeNode));
            if(y->vexlist[i].firstedge)
            {
                EdgeNode* s=y->vexlist[i].firstedge;
                while(s->next)
                    s=s->next;
                p->adjvex=j;
                p->cost=x->EData[i][j];
                p->next=NULL;
                s->next=p;
            }
            else
            {
```

```
                    p->adjvex=j;
                    p->cost=x->EData[i][j];
                    p->next=NULL;
                    y->vexlist[i].firstedge=p;
                }
                EdgeNode* q=(EdgeNode*)malloc(sizeof(EdgeNode));
                if(y->vexlist[j].firstedge)
                {
                    EdgeNode* s=y->vexlist[j].firstedge;
                    while(s->next)
                        s=s->next;
                    q->adjvex=i;
                    q->cost=x->EData[i][j];
                    q->next=NULL;
                    s->next=q;
                }
                else
                {
                    q->adjvex=i;
                    q->cost=x->EData[i][j];
                    q->next=NULL;
                    y->vexlist[j].firstedge=q;
                }
            }
        }
    }
    printf("Change Completed");
}
```

```
int LocateVex(AmlGraph* G, char v)
{
    int i;
    for(i=0; i<G->v; ++i)
    {
        if(G->adjmulist[i].Vertex==v)
            return i;
    }
    return -1;
}

void CreateAmlGraph(AmlGraph* G)
{
    char v1,v2;
    int i,j,k,num;
    FILE* fp;
    G->e=0;

    fp=fopen(FileAddress,"r");
    if(!fp)
        exit(-1);
    if(!feof(fp))
    {
        fscanf(fp,"Vertex:%d\nEdge:%d\n",&G->v,&G->e);
        for(i=0; i<G->v; i++)
        {
            G->adjmulist[i].Vertex=fgetc(fp);
            G->adjmulist[i].firstedge=NULL;
        }
```

```
        for(k=0; k<G->e; k++)
        {
            fscanf(fp,"\n[%c][%c]: %d",&v1,&v2,&num);
            i=LocateVex(G,v1); j=LocateVex(G,v2);
            EBox* p=(EBox*)malloc(sizeof(EBox));
            p->ivex=i; p->jvex=j; p->cost=num;
            p->ilink=G->adjmulist[i].firstedge;
            p->jlink=G->adjmulist[j].firstedge;
            G->adjmulist[i].firstedge=G->adjmulist[j].firstedge=p;
        }
    }
    printf("Read Completed");
    fclose(fp);

}

void CreateAdjGraph(AdjGraph* G)
{
    int i,j,k,num,d;
    char v1,v2;
    FILE* fp;
    fp=fopen(FileAddress,"r");
    if(!fp)
        exit(-1);
    if(!feof(fp))
    {
        fscanf(fp,"Vertex:%d\nEdge:%d\n",&G->v,&G->e);
        EdgeNode* m;
        for(i=0; i<G->v; i++)
```

```
{
    G->vexlist[i].vertax=fgetc(fp);
    while(G->vexlist[i].firstedge)
    {
        m=G->vexlist[i].firstedge;
        G->vexlist[i].firstedge=G->vexlist[i].firstedge->next;
        free(m);
    }
}
for(k=0; k<G->e; k++)
{
    fscanf(fp,"\n[%c][%c]: %d",&v1,&v2,&num);
    for(d=0; d<G->v; d++)
    {
        if(v1==G->vexlist[d].vertax)
            i=d;
        if(v2==G->vexlist[d].vertax)
            j=d;
    }
    EdgeNode* p=(EdgeNode*)malloc(sizeof(EdgeNode));
    if(G->vexlist[i].firstedge)
    {
        EdgeNode* s=G->vexlist[i].firstedge;
        while(s->next)
            s=s->next;
        p->adjvex=j;
        p->cost=num;
        p->next=NULL;
        s->next=p;
```

```
        }
        else
        {
            p->adjvex=j;
            p->cost=num;
            p->next=NULL;
            G->vexlist[i].firstedge=p;
        }

        EdgeNode* q=(EdgeNode*)malloc(sizeof(EdgeNode));
        if(G->vexlist[j].firstedge)
        {
            EdgeNode* s=G->vexlist[j].firstedge;
            while(s->next)
                s=s->next;
            q->adjvex=i;
            q->cost=num;
            q->next=NULL;
            s->next=q;
        }
        else
        {
            q->adjvex=i;
            q->cost=num;
            q->next=NULL;
            G->vexlist[j].firstedge=q;
        }
    }
}
```

```
    printf("Read Completed");
    fclose(fp);
}

void CreateMGraph(MGraph* G)
{
    int i,j,k,d,num;
    char v1,v2;
    FILE* fp;
    G->e=0;

    fp=fopen(FileAddress,"r");
    if(!fp)
        exit(-1);
    if(!feof(fp))
    {
        fscanf(fp,"Vertex:%d\nEdge:%d\n",&G->v,&G->e);
        for(i=0; i<G->v; i++)
            G->VData[i]=fgetc(fp);
        for(i=0; i<G->v; i++)
            for(j=0; j<G->v; j++)
                G->EData[i][j]=0;
        for(k=0; k<G->e; k++)
        {
            fscanf(fp,"\n[%c][%c]: %d",&v1,&v2,&num);
            for(d=0; d<G->v; d++)
            {
                if(v1==G->VData[d])
                    i=d;
```

```
                if(v2==G->VData[d])
                    j=d;
            }
            G->EData[i][j]=num;
            G->EData[j][i]=num;
        }
    }
    printf("Read Completed");
    fclose(fp);
}

void PrintAmlGraph(AmlGraph* G)
{
    int i,j;
    if(G->v==-1)
    {
        printf("No Have AmlGraph!\n");
        return;
    }
    EBox* s;
    for(i=0; i<G->v; i++)
    {
        s=G->adjmulist[i].firstedge;
        while(s)
        {
            if(s->ivex==i)
            {
                printf("|                                                %c-(%d)-%c
|",G->adjmulist[i].Vertex,s->cost,G->adjmulist[s->jvex].Vertex);
```

```
                s=s->ilink;
            }
            else
            {
                printf("|                                                    %c-(%d)-%c
|",G->adjmulist[i].Vertex,s->cost,G->adjmulist[s->ivex].Vertex);
                s=s->jlink;
            }
        }
        printf("\n");
    }
}

void PrintAdjGraph(AdjGraph* G)
{
    int i,j;
    if(G->v==-1)
    {
        printf("No Have AdjGraph!\n");
        return;
    }
    EdgeNode* S;
    for(i=0; i<G->v;i++)
    {
        S=G->vexlist[i].firstedge;
        printf("\n%c: ",G->vexlist[i].vertax);
        while(S)
        {
            printf(" %c(%d)",G->vexlist[S->adjvex].vertax,S->cost);
```

```
            S=S->next;
        }
    }
}

void PrintMGraph(MGraph* G)
{
    int i,j;
    if(G->v==-1)
    {
        printf("No Have MGraph!\n");
        return;
    }
    for(i=0,j=0; i<G->v;i++)
    {
        printf(" ");
        while(j<G->v)
        {
            printf(" %c",G->VData[j++]);
        }
        printf("\n");
        printf("%c",G->VData[i]);
        for(j=0; j<G->v; j++)
        {
            printf(" %d",G->EData[i][j]);
        }
    }
}
```

```
void Select(MGraph* A,AdjGraph* B,AmlGraph* C)
{

    int i,j,num;
            gotoxy(1,1);
    printf("\n[1]  Create MGraph\t\t[12] [L]DFS Travel AmlGraph\n");
    printf("[2]  Create AdjGraph\t\t[13] [R]DFS Travel MGraph\n");
    printf("[3]  Create AmlGraph\t\t[14] [R]DFS Travel AdjGraph\n");
    printf("[4]  MGraph to AdjGraph\t\t[15] [R]DFS Travel AmlGraph\n");
    printf("[5]  MGraph to AmlGraph\t\t[16] BFS Travel MGraph\n");
    printf("[6]  AdjGraph to MGraph\t\t[17] BFS Travel AdjGraph\n");
    printf("[7]  AdjGraph to AmlGraph\t[18] BFS Travel AmlGraph\n");
    printf("[8]  AmlGraph to MGraph\t\t[19] PRINT MGraphn\n");
    printf("[9]  AmlGraph to AdjGraph\t[20] PRINT AdjGraphn\n");
    printf("[10] [L]DFS Travel MGraph\t[21] PRINT AmlGraphn\n");
    printf("[11] [L]DFS Travel AdjGraph\t[0]  EXIT~\n");
    printf("\nSelect Number:");
    do{
    gotoxy(1,1);
    gotoxy(15,14);
    scanf("%d",&num);
    gotoxy(15,14);
    for(i=0; i<NumMax;i++)
        printf("\t\t\t\t\t\t\t\n");
    gotoxy(1,16);
    switch(num)
    {
    case 1:
        CreateMGraph(A);
```

```
    break;
case 2:
    CreateAdjGraph(B);
    break;
case 3:
    CreateAmlGraph(C);
    break;
case 4:
    MtoAdj(A,B);
    break;
case 5:
    MtoAml(A,C);
    break;
case 6:
    AdjtoM(B,A);
    break;
case 7:
    AdjtoAml(B,C);
    break;
case 8:
    AmltoM(C,A);
    break;
case 9:
    AmltoAdj(C,B);
    break;
case 10:
    MGraphLoop(A);
    break;
case 11:
```

```
        AdjGraphLoop(B);
        break;
    case 12:
        AmlGraphLoop(C);
        break;
    case 13:
        MGraphRecursive(A);
        break;
    case 14:
        AdjGraphRecursive(B);
        break;
    case 15:
        AmlGraphRecursive(C);
        break;
    case 16:
        BFSM(A);
        break;
    case 17:
        BFSAdj(B);
        break;
    case 18:
        BFSAml(C);
        break;
    case 19:
        PrintMGraph(A);
        break;
    case 20:
        PrintAdjGraph(B);
        break;
```

```
    case 21:
        PrintAmlGraph(C);
        break;
    case 0:
        system("cls");
    default:break;
    }
    }while(num!=0);
}

int main()
{
    int i=0;
    stack_h=(stack_0)malloc(sizeof(struct_));
    queue_h=(queue_0)malloc(sizeof(struct_));
    stack_h->next=NULL;
    queue_h->next=NULL;
    MGraph A;
    AdjGraph B;
    for(i=0; i<NumMax; i++)
        B.vexlist[i].firstedge=NULL;
    AmlGraph C;
    A.v=B.v=C.v=-1;
    Select(&A,&B,&C);
    return 0;
}
```