**数据结构实验报告**

**实验八 利用图搜索方法来解决迷宫问题**

学 生 姓 名 ：杨仕振

学 号 ：2015301500349

二○一六年十二月

**一、简介**

【问题描述】解决迷宫问题，迷宫见教材P77图3.7。

【基本要求】

设计程序建立迷宫对应的邻接表表示；

采用深度优先遍历算法输出从入口（1，1）到出口（M，N）的所有迷宫路径。

**二、算法说明**

程序中使用的数据结构：

typedef struct ANode

{

int i, j;

struct ANode \*nextarc;

}ArcNode;// 邻接表边节点

typedef struct Vnode

{

ArcNode \*firstarc;

}Vnode;// 邻接表头结点结构类型

typedef struct

{

Vnode adjlist[M + 2][N + 2];

}ALGraph;// 邻接表结构类型

typedef struct

{

int i;

int j;

}Grid;// 迷宫中的格子

typedef struct

{

Grid data[Maxsize];

int length;

}Pathrecord;// 记录一条可走的路径

int visited[M + 2][N + 2] ;// 全局数组，记录当前格子是否已走

程序中使用的主要函数：

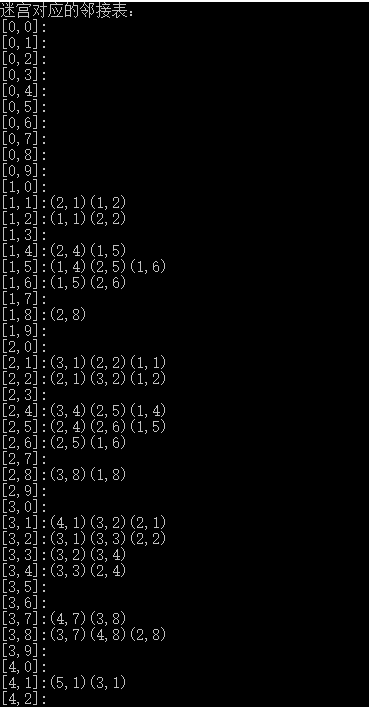
void CreateList(ALGraph \*&G, int mg[][N + 2])// 将记录迷宫信息的数组转化为邻接表

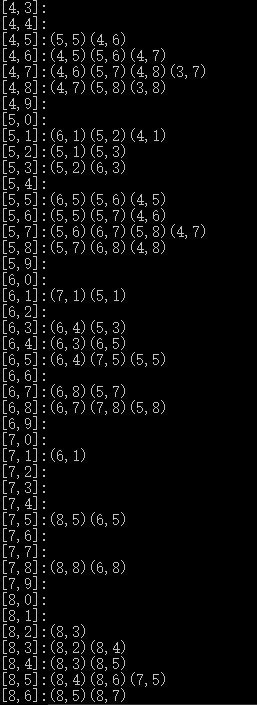
void DispAdj(ALGraph \*G)//输出邻接表

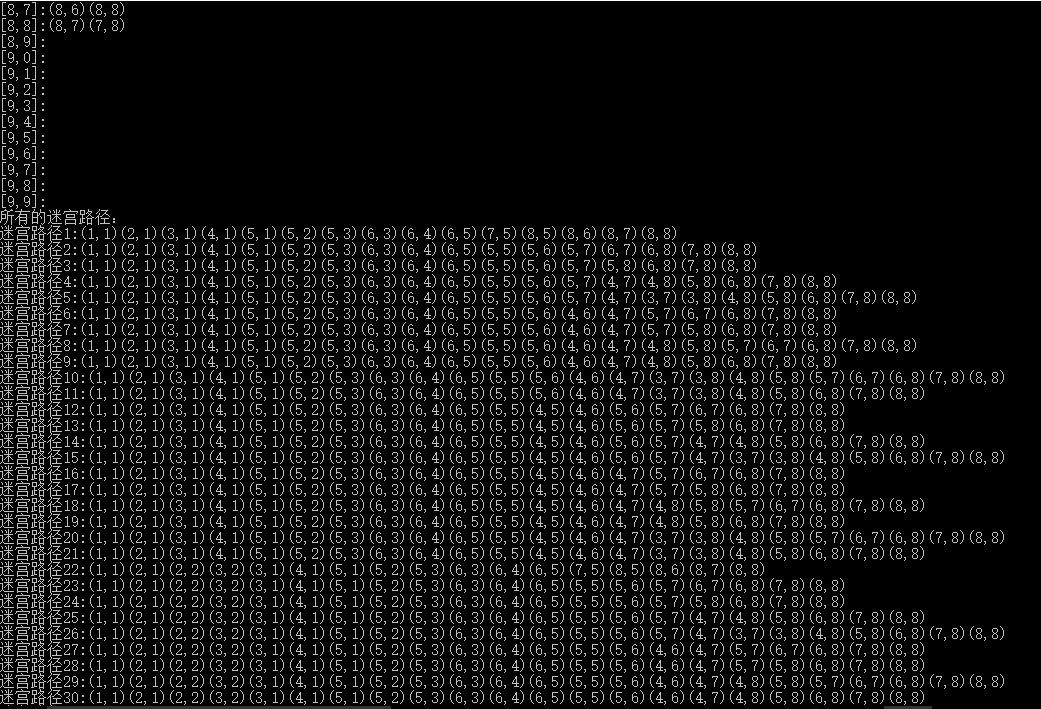
void FindPath(ALGraph \*G, int xi, int yi, int xe, int ye, Pathrecord path)//通过深度遍历找到所有路径并打印

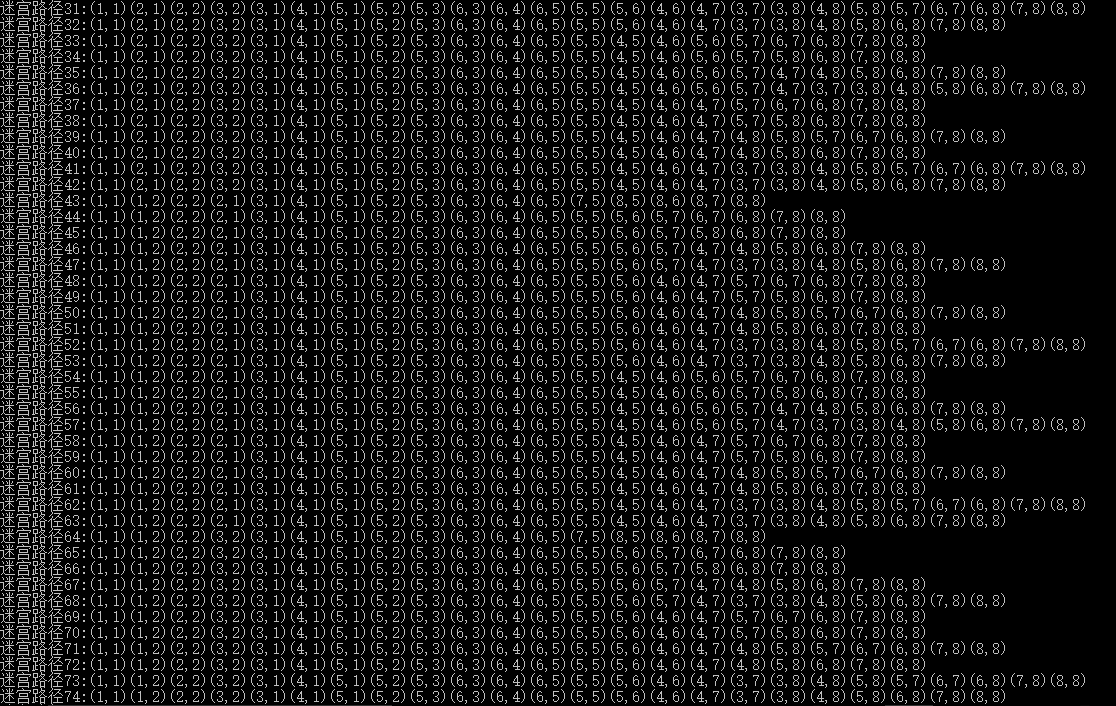
**三、测试结果**

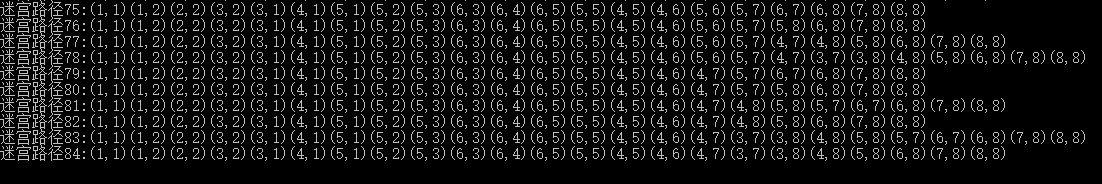
共输出83个结果，如下











**四、分析与探讨**

该程序的空间复杂度取决于创建邻接表的函数，为O(n2)

该程序的时间复杂度取决于输出邻接表的函数，为O(n3)

关于改进，在确认一个方块周围的可走方块时，可以通过循环和switch语句进行简化

**附录：源代码**

#include<stdio.h>

#include<stdlib.h>

#define M 8

#define N 8

#define Maxsize M\*N

typedef struct ANode

{

int i, j;

struct ANode \*nextarc;

}ArcNode;// 邻接表边节点

typedef struct Vnode

{

ArcNode \*firstarc;

}Vnode;// 邻接表头结点结构类型

typedef struct

{

Vnode adjlist[M + 2][N + 2];

}ALGraph;// 邻接表结构类型

typedef struct

{

int i;

int j;

}Grid;// 迷宫中的格子

typedef struct

{

Grid data[Maxsize];

int length;

}Pathrecord;// 记录一条可走的路径

int visited[M + 2][N + 2] = { 0 };// 全局数组，记录当前格子是否已走

int count = 0;// 统计可走的迷宫路径条数

void CreateList(ALGraph \*&G, int mg[][N + 2])// 将记录迷宫信息的数组转化为邻接表

{

int i, j;

ArcNode \*p;

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

for (i = 0; i < M + 2; i++)

for (j = 0; j < N + 2; j++)

G->adjlist[i][j].firstarc = NULL;//初始化邻接表

for (i = 1; i <= M; i++)

for (j = 1; j <= N; j++)

if (mg[i][j] == 0)

{

if (mg[i-1][j]==0)//依次检测四个格子是否可走

{

p = (ArcNode \*)malloc(sizeof(ArcNode));

p->i = i-1; p->j = j;

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

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

}

if (mg[i][j+1] == 0)

{

p = (ArcNode \*)malloc(sizeof(ArcNode));

p->i = i; p->j = j+1;

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

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

}

if (mg[i+1][j] == 0)

{

p = (ArcNode \*)malloc(sizeof(ArcNode));

p->i = i+1; p->j = j;

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

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

}

if (mg[i][j-1] == 0)

{

p = (ArcNode \*)malloc(sizeof(ArcNode));

p->i = i; p->j = j-1;

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

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

}

}

}

void DispAdj(ALGraph \*G)//输出邻接表

{

int i, j;

ArcNode \*p;

for (i = 0; i<M + 2; i++)

for (j = 0; j < N + 2; j++)

{

printf("[%d,%d]:", i, j);

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

while (p != NULL)

{

printf("(%d,%d)", p->i, p->j);

p = p->nextarc;

}

printf("\n");

}

}

void FindPath(ALGraph \*G, int xi, int yi, int xe, int ye, Pathrecord path)//通过深度遍历找到所有路径并打印

{

ArcNode \*p;

visited[xi][yi] = 1;//置当前格子为已访问

path.data[path.length].i = xi; path.data[path.length].j = yi;

path.length++;

if (xi == xe&&yi == ye)//如果到达终点，则输出路径

{

printf("迷宫路径%d:", ++count);

for (int k = 0; k < path.length; k++)

printf("(%d,%d)", path.data[k].i, path.data[k].j);

printf("\n");

}

p = G->adjlist[xi][yi].firstarc;

while (p != NULL)

{

if (visited[p->i][p->j] == 0)

FindPath(G, p->i, p->j, xe, ye, path);//未到达重点且格子未访问时递归调用FindPath函数

p = p->nextarc;

}

visited[xi][yi] = 0;

}

void main()

{

ALGraph \*G;

int mg[M + 2][N + 2] = {

{ 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 },

{ 1, 0, 0, 1, 0, 0, 0, 1, 0, 1 },

{ 1, 0, 0, 1, 0, 0, 0, 1, 0, 1 },

{ 1, 0, 0, 0, 0, 1, 1, 0, 0, 1 },

{ 1, 0, 1, 1, 1, 0, 0, 0, 0, 1 },

{ 1, 0, 0, 0, 1, 0, 0, 0, 0, 1 },

{ 1, 0, 1, 0, 0, 0, 1, 0, 0, 1 },

{ 1, 0, 1, 1, 1, 0, 1, 1, 0, 1 },

{ 1, 1, 0, 0, 0, 0, 0, 0, 0, 1 },

{ 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 },

};//储存迷宫的矩阵

CreateList(G, mg);

printf("迷宫对应的邻接表：\n"); DispAdj(G);

Pathrecord path;

path.length = 0;

printf("所有的迷宫路径：\n");

FindPath(G, 1, 1, M, N, path);

}