# 实验报告

题目：迷宫问题——栈的应用

班级：网络工程 姓名：王硕 学号：1625131037

一、需求分析

1. 实现一个链栈，并完成以下操作：初始化、入栈、出栈、取栈的长度、取栈顶元素的值、删除栈顶元素。实现一个用二维数组表示的迷宫。
2. 演示程序以用户和计算机的对话方式进行，即在计算机终端上显示“提示信息“之后，由用户在键盘上输入演示程序中规定的运算命令；相应的输入数据和运算结果显示在后。
3. 程序根据迷宫，用穷举法求出通过迷宫的路径，输出每一步的位置和走向。
4. 测试数据：迷宫：

q[11][10]={{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},{1,0,0,0,0,1,1,0,1,1},{1,0,1,1,1,0,0,1,0,1},{1,0,0,0,1,0,0,0,0,1},{1,0,1,0,0,0,1,0,1,1},{1,0,1,1,1,1,0,0,1,1},{1,1,1,0,0,0,1,0,1,1},{1,1,1,0,0,0,0,0,0,1},{1,1,1,1,1,1,1,1,1,1}};

（1表示障碍，0表示通路，迷宫已经在四周加上障碍）

二．概要实现

1.为实现上述程序功能，需要链栈抽象数据类型。

**ADT** linkstack{

数据对象：D={ai| ai∈I，i=1,2,3,4…n，n≥0}

数据关系：{< ai-1, ai>| ai-1, ai ∈I}

基本操作：

Init（&S）

初始化链栈S；

Pop（&S，&date）

对S出栈，栈顶元素赋值给date；

Push（&s，date）

对S入栈，入栈元素的数据为date；

Del（&s）

删除s的栈顶元素；

Length（s）

求s的长度并返回其值；

2.实现程序功能的函数。

judge(linkstack &s,int &i,int &j)

根据s中保存的位置和前进方向判断按该方向前进能否走通；

guide(linkstack &s,int &i,int &j)

按照s中保存的位置和前进方向在迷宫中移动，如果满足路径条件对s执行入栈操作，若进入迷宫死胡同对s进行出栈操作，移动一次则对曾移动到的地块标记为障碍1，根据judge的返回数据更改前进方向判别是否进入死胡同。

三、详细设计

void init(linkstack &s) //初始化链栈

{

lsnode \*t=new lsnode;

t->next=NULL;

s=t;

}

void push(linkstack &s,int e1,int e2,int e3) //入栈

{

lsnode \*t=new lsnode;

t->h=e1;

t->l=e2;

t->d=e3;

t->next=s->next;

s->next=t;

}

void pop(linkstack &s,int &e1,int &e2,int &e3) //出栈

{

lsnode \*t;

t=s->next;

e1=t->h;

e2=t->l;

e3=t->d;

s->next=t->next;

delete t;

}

void del(linkstack &s)

{

lsnode \*t;

t=s->next;

s->next=t->next;

delete t;

}

int length(linkstack s)

{

lsnode \*t;

t=s;

int i=0;

while(s->next!=NULL)

{

i++;

s=s->next;

}

s=t;

return i;

}

int judge(linkstack &s,int &i,int &j)

{

if(s->next->d==1 && q[i][j+1]==0)

{

j++;

return 1;

}

if(s->next->d==2 && q[i+1][j]==0)

{

i++;

return 1;

}

if(s->next->d==3 && q[i][j-1]==0)

{

j--;

return 1;

}

if(s->next->d==4 && q[i-1][j]==0)

{

i--;

return 1;

}

return 0;

}

void guide(linkstack &s,int &i,int &j)

{

if(judge(s,i,j))

{

push(s,i,j,1);

q[i][j]=1;

}

else

if(s->next->d<=4)

s->next->d=s->next->d+1;

else

{

del(s);

i=s->next->h;

j=s->next->l;

q[i][j]=1;

}

}

四、调试分析

本次实验实现较为复杂，走迷宫的路径按照先入后出的顺序，符合栈的特性。但实现具体算法实现需要考虑每次走迷宫的方向的变换，对应方向能否走通的判定，对走过路径的处理。

对于复杂算法，善于使用调试debug非常重要，可以很快找到算法中的漏洞。

附完整代码：

#include<iostream>

using namespace std;

int q[11][10]={{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},

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

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

typedef struct lsnode

{

int h,l,d;

lsnode \*next;

}\*linkstack;

void init(linkstack &s)

{

lsnode \*t=new lsnode;

t->next=NULL;

s=t;

}

void push(linkstack &s,int e1,int e2,int e3)

{

lsnode \*t=new lsnode;

t->h=e1;

t->l=e2;

t->d=e3;

t->next=s->next;

s->next=t;

}

void pop(linkstack &s,int &e1,int &e2,int &e3)

{

lsnode \*t;

t=s->next;

e1=t->h;

e2=t->l;

e3=t->d;

s->next=t->next;

delete t;

}

void del(linkstack &s)

{

lsnode \*t;

t=s->next;

s->next=t->next;

delete t;

}

int length(linkstack s)

{

lsnode \*t;

t=s;

int i=0;

while(s->next!=NULL)

{

i++;

s=s->next;

}

s=t;

return i;

}

int judge(linkstack &s,int &i,int &j)

{

if(s->next->d==1 && q[i][j+1]==0)

{

j++;

return 1;

}

if(s->next->d==2 && q[i+1][j]==0)

{

i++;

return 1;

}

if(s->next->d==3 && q[i][j-1]==0)

{

j--;

return 1;

}

if(s->next->d==4 && q[i-1][j]==0)

{

i--;

return 1;

}

return 0;

}

void guide(linkstack &s,int &i,int &j)

{

if(judge(s,i,j))

{

push(s,i,j,1);

q[i][j]=1;

}

else

if(s->next->d<=4)

s->next->d=s->next->d+1;

else

{

del(s);

i=s->next->h;

j=s->next->l;

q[i][j]=1;

}

}

int main()

{

linkstack s;

init(s);

int i=1,j=1,g=1;

push(s,i,j,g);

do

{

guide(s,i,j);

}while(i!=9 || j!=8);

linkstack t;

init(t);

do

{

pop(s,i,j,g);

push(t,i,j,g);

}while(s->next);

while(t->next!=NULL)

{

i=t->next->h;

j=t->next->l;

g=t->next->d;

cout<<"("<<i<<","<<j<<","<<g<<")"<<" ";

del(t);

cout<<endl;

}

return 0;

}