HW4

算法设计题

陈昕琪 PB11711

1.迷宫求解问题

具体代码如下： 输出形式为，路径用5表示。

```
#include<stdio.h>
#define m 8
#define n 8
#define MAXSIZE 100

//定义迷宫
int maze[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,
        1,1,0,0,0,1,0,0,0,1,
        1,1,1,1,1,1,1,1,1,1
};

//走过的位置标记
int mark[m+2][n+2]={0};

//表示位置的结构体
typedef struct PostType
{
    int x,y;
}PostType;

//栈元素
typedef struct SElemType{
    int ord;//通道块在路径上的序号
    PostType seat;//坐标位置
    int di;//前进方向(东1，南2，西3，北4)
}SElemType;

//定义栈
typedef struct SqStack{
    SElemType *base;
    SElemType *top;
    int stacksize;
```

```
}SqStack;

//顺序栈初始化
int InitStack(SqStack &S){
    S.base=new SElemType[MAXSIZE];
    if(!S.base) return 0;
    S.top=S.base;
    S.stacksize=MAXSIZE;
    return 1;
}
//进栈
int Push(SqStack &S,SElemType e){
    if(S.top-S.base==S.stacksize) return 0;//如果栈已满则不能再压入
    //移动栈顶
    *S.top=e;
    S.top++;
    return 1;
}

//出栈
int Pop(SqStack &S){
    if(S.top==S.base) return 0; //如果栈已经为空则不能弹出
    //移动栈顶
    S.top--;
    return 1;
}

//获取栈顶元素
SElemType GetTop(SqStack stack)
{
    return *(stack.top-1);
}

//顺序栈判空
int StackEmpty(SqStack S){
    if(S.top==S.base) return 1;
    return 0;
}

//判断该位置是否是墙以及是否走过
int IfWall_NeverGo(SElemType e){
    if(maze[e.seat.x][e.seat.y]==1 || mark[e.seat.x][e.seat.y]==1){
        return 1;
    }
    return 0;
}

//判断该位置是否为出口
int IfOut(SElemType e){
    if(e.seat.x==m&&e.seat.y==n){
        printf("Find the way!\n");
        return 1;
    }
    return 0;
```

```
}
//判断该位置是否是原点
int IfGet(SElemType e){
    if(e.seat.x==1&&e.seat.y==1){
        return 1;//是原点
    }
    return 0;//不是原点
}

//移动
SElemType Move(SqStack &S){
    SElemType node;
    node=GetTop(S);
    SElemType new_node;
    switch(node.di){
        case 1:
            new_node.ord=node.ord+1;
            new_node.seat.x=node.seat.x;
            new_node.seat.y=node.seat.y+1;
            break;
        case 2:
            new_node.ord=node.ord+1;
            new_node.seat.x=node.seat.x+1;
            new_node.seat.y=node.seat.y;
            break;
        case 3:
            new_node.ord=node.ord+1;
            new_node.seat.x=node.seat.x;
            new_node.seat.y=node.seat.y-1;
            break;
        case 4:
            new_node.ord=node.ord+1;
            new_node.seat.x=node.seat.x-1;
            new_node.seat.y=node.seat.y;
            break;
    }
    new_node.di=1;
    Push(S,new_node);
    return new_node;
}

//标记走过的道路
void MarkPrint(SElemType node){
    mark[node.seat.x][node.seat.y]=1;
}

void PrintOut(SqStack &S){//输出
    SElemType *p=S.base;
    while(p<S.top){
        maze[p->seat.x][p->seat.y]=5;
        p++;
    }
    for(int i=0;i<m+2;i++){
        for(int j=0;j<m+2;j++){
```

```
            printf("%d",maze[i][j]);
        }
        printf("\n");
    }
}

//回溯算法
void Go(SqStack &S){
//  printf("begin to go\n");
    SElemType node;
    node.seat.x=1;
    node.seat.y=1;
    node.di=1;
    node.ord=1;
//  printf("successfully init\n");

    Push(S,node);

    int flag=0;
    while(!StackEmpty(S)){//非空
        SElemType new_node=Move(S);
        //不是墙
        if(!IfWall_NeverGo(GetTop(S))){//可以走
            if(IfOut(GetTop(S))){//是出口，结束
                break;
            }
            if(flag && IfGet(GetTop(S))){//是回路，结束
                break;
            }
            else{
                MarkPrint(new_node);
                flag++;//除去第一次
            }
        }
        //是墙
        else{
            Pop(S);//新位置不能走，要移出栈
            if(!StackEmpty(S)){
                //四个方向都走完了
                if(GetTop(S).di==4&&!StackEmpty(S)){
                    MarkPrint(GetTop(S)); //回溯后为了防止这个点再次被搜索，将其标记为不能访问
                    Pop(S);
                }
                if(GetTop(S).di<4){
                    (S.top-1)->di++;
                }
            }
        }
    }
    PrintOut(S);
}

int main(){
```

```
    SqStack S;
    InitStack(S);
    Go(S);
    return 0;
}
```

2.打印杨辉三角形

输入：需要打印的杨辉三角的行数 输出：杨辉三角形

```
#include<stdio.h>
#define Maxsize 50
//定义队列
typedef struct{
    int data[Maxsize];//存储行的数据
    int front,rear;//头尾指针
}SqQueue;

//初始化
void InitQueue(SqQueue &Q){
    Q.rear=Q.front=0;
}

//判空
int IsEmpty(SqQueue Q){
    if(Q.rear==Q.front)
        return 1;
    return 0;
}

//入队
int EnQueue(SqQueue &Q,int x){
    if((Q.rear+1)%Maxsize==Q.front)//判断队列是否满了
        return 0;
    Q.data[Q.rear]=x;
    Q.rear=(Q.rear+1)%Maxsize;
    return 1;
}

//出队
int DeQueue(SqQueue &Q,int &x){
    if(IsEmpty(Q))
        return 0;
    x=Q.data[Q.front];
    Q.front=(Q.front+1)%Maxsize;
    return 1;
}

//获取队头
int GetTop(SqQueue Q,int &x){
    if(IsEmpty(Q))
```

```
        return 0;
    x=Q.data[Q.front];
    return 1;
}

void YangHuiTriangle(int N){
    int n,x,i,temp;
    SqQueue Q;
    InitQueue(Q);

    EnQueue(Q,1);
    for(n=2;n<=N;n++){
        EnQueue(Q,1);//每行第一个元素是1
        for(i=1;i<=n-2;i++){
            DeQueue(Q,temp);//出队
            printf("%d ",temp);
            GetTop(Q,x);
            temp=temp+x;//将队首的两个元素相加得到新的元素
            EnQueue(Q,temp);
        }
        DeQueue(Q,x);
        printf("%d ",x);
        EnQueue(Q,1);//结尾安插一个1
        printf("\n");
    }
    //最后一行输出
    while(!IsEmpty(Q)){
        DeQueue(Q,x);
        printf("%d ",x);
    }

}

int main(){
    int N;
    printf("Input:");
    scanf("%d",&N);
    YangHuiTriangle(N);
    return 0;
}
```