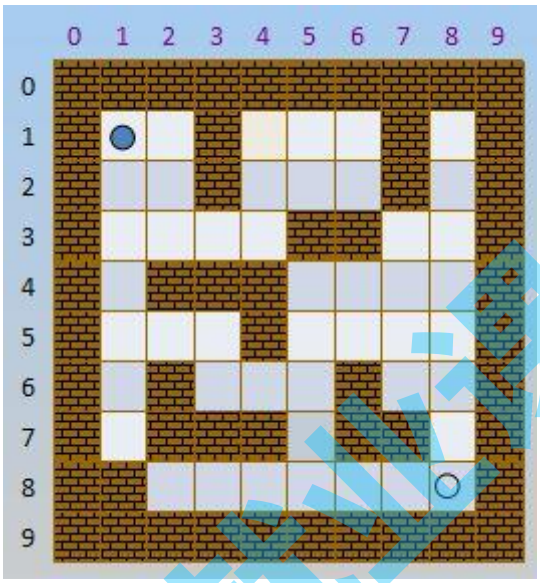
7--实践练习-迷宫问题

本节目标

- **迷宫问题求解**
- **迷宫最短路径求解**

1. 迷宫问题求解



https://www.nowcoder.com/questionTerminal/cf24906056f4488c9ddb132f317e03bc

以上迷宫OJ题曾经是百度某一年的其中一个笔试题，迷宫问题本质就是一个图的遍历问题，从起点开始不断四个方向探索，直到走到出口，走的过程中我们借助栈记录走过路径的坐标，栈记录坐标有两方面的作用，一方面是记录走过的路径，一方面方便走到死路时进行回溯找其他的通路。

```
#include<stdio.h>
#include<stdlib.h>
#include<assert.h>
#include<stdbool.h>

typedef struct Postion
{
    int row;
    int col;
}PT;
//////////////////////////////////////////////////////////////////
typedef PT STDataType;

typedef struct Stack
{
    STDataType* a;
    int top;
    int capacity;
}ST,Stack;

void StackInit(ST* ps);
```

```c
void StackDestory(ST* ps);
// 入栈
void StackPush(ST* ps, STDataType x);
// 出栈
void StackPop(ST* ps);
STDataType StackTop(ST* ps);

int StackSize(ST* ps);
bool StackEmpty(ST* ps);

void StackInit(ST* ps)
{
    assert(ps);

    ps->a = (STDataType*)malloc(sizeof(STDataType) * 4);
    if (ps->a == NULL)
    {
        printf("malloc fail\n");
        exit(-1);
    }

    ps->capacity = 4;
    ps->top = 0;
}

void StackDestory(ST* ps)
{
    assert(ps);
    free(ps->a);
    ps->a = NULL;
    ps->top = ps->capacity = 0;
}

// 入栈
void StackPush(ST* ps, STDataType x)
{
    assert(ps);

    // 满了-》增容
    if (ps->top == ps->capacity)
    {
        STDataType* tmp = (STDataType*)realloc(ps->a, ps->capacity * 2 *
sizeof(STDataType));
        if (tmp == NULL)
        {
            printf("realloc fail\n");
            exit(-1);
        }
        else
        {
            ps->a = tmp;
            ps->capacity *= 2;
        }
    }

    ps->a[ps->top] = x;
    ps->top++;
}
```

```c
// 出栈
void StackPop(ST* ps)
{
    assert(ps);
    // 栈空了，调用Pop，直接中止程序报错
    assert(ps->top > 0);

    //ps->a[ps->top - 1] = 0;
    ps->top--;
}

STDataType StackTop(ST* ps)
{
    assert(ps);
    // 栈空了，调用Top，直接中止程序报错
    assert(ps->top > 0);

    return ps->a[ps->top - 1];
}

int StackSize(ST* ps)
{
    assert(ps);

    return ps->top;
}

bool StackEmpty(ST* ps)
{
    assert(ps);

    return ps->top == 0;
}

Stack path;
//////////////////////////////////////////////////////////////

void PrintMaze(int** maze, int N, int M)
{
    for(int i = 0 ; i < N; ++i)
    {
        for(int j = 0; j < M; ++j)
        {
            printf("%d ", maze[i][j]);
        }
        printf("\n");
    }
    printf("\n");
}

// 输出栈里面的坐标路径
void PirntPath(Stack* ps)
{
    // path数据倒到rPath
    Stack rPath;
    StackInit(&rPath);
    while(!StackEmpty(&path))
```

```
    {
        StackPush(&rPath, StackTop(&path));
        StackPop(&path);
    }

    while(!StackEmpty(&rPath))
    {
        PT top = StackTop(&rPath);
        printf("(%d,%d)\n", top.row, top.col);
        StackPop(&rPath);
    }

    StackDestory(&rPath);
}

bool IsPass(int** maze, int N, int M, PT pos)
{
    if(pos.row >= 0 && pos.row < N
      && pos.col >= 0 && pos.col < M
      && maze[pos.row][pos.col] == 0)
    {
        return true;
    }
    else
    {
        return false;
    }
}

bool GetMazePath(int** maze, int N, int M, PT cur)
{
    StackPush(&path, cur);

    // 如果走到出口
    if(cur.row == N-1 && cur.col == M-1)
        return true;

    // 探测cur位置得上下左右四个方向
    PT next;
    maze[cur.row][cur.col] = 2;

    // 上
    next = cur;
    next.row -= 1;
    if(IsPass(maze, N, M, next))
    {
        if(GetMazePath(maze, N, M, next))
            return true;
    }

    // 下
    next = cur;
    next.row += 1;
    if(IsPass(maze, N, M, next))
    {
        if(GetMazePath(maze, N, M, next))
            return true;
    }
```

```
    // 左
    next = cur;
    next.col -= 1;
    if(IsPass(maze, N, M, next))
    {
        if(GetMazePath(maze, N, M, next))
            return true;
    }

    // 右
    next = cur;
    next.col += 1;
    if(IsPass(maze, N, M, next))
    {
        if(GetMazePath(maze, N, M, next))
            return true;
    }

    StackPop(&path);

    return false;
}

int main()
{
    int N = 0, M = 0;
    while(scanf("%d%d", &N, &M) != EOF)
    {
        // int a[n][m]; // vs2013 不支持
        // 动态开辟二维数组
        int** maze = (int**)malloc(sizeof(int*)*N);
        for(int i = 0; i < N; ++i)
        {
            maze[i] = (int*)malloc(sizeof(int)*M);
        }

        // 二维数组得输入
        for(int i = 0 ; i < N; ++i)
        {
            for(int j = 0; j < M; ++j)
            {
                scanf("%d", &maze[i][j]);
            }
        }

        StackInit(&path);
        // PrintMaze(maze, N, M);
        PT entry = {0, 0};
        if(GetMazePath(maze, N, M, entry))
        {
            //printf("找到通路\n");
            PirntPath(&path);
        }
        else
        {
            printf("没有找到通路\n");
```

```
        }

        StackDestory(&path);

        for(int i = 0; i < N; ++i)
        {
            free(maze[i]);
        }
        free(maze);
        maze = NULL;
    }

    return 0;
}
```

2.迷宫最短路径求解

> https://www.nowcoder.com/questionTerminal/571cfbe764824f03b5c0bfd2eb0a8ddf
>
> 本题在曾经是美团某一年的笔试题，本题在上一个题的基础上计入了体力值的概念，但是本题还有一个隐藏条件，就是要找出迷宫的最短路径，如下图的两个测试用例，需要找出最短路径，才能通过全部测试用例。
>
> 4 4 10 1 0 0 1 1 1 0 1 0 1 1 1 0 0 1 1
>
> 10 10 50 1 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 1 1 0 0 1 0 0 0 0 0 1 1 0 0 1 0 0 0 0 1 1 0 0 1 1 1 1 1 1 1 1 1 0 0 1 0 0 0 0 0 1 1 0 0 1 1 1 1 1 1 1 1 1 0 0 1 0 0 0 0 0 1 1 0 0 1 1 1 1 1 1 1

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<assert.h>
#include<stdbool.h>

typedef struct Postion
{
    int row;
    int col;
}PT;
//////////////////////////////////////////////////////////////////
typedef PT STDataType;

typedef struct Stack
{
    STDataType* a;
    int top;
    int capacity;
}ST, Stack;

void StackInit(ST* ps);
void StackDestory(ST* ps);
// 入栈
void StackPush(ST* ps, STDataType x);
// 出栈
void StackPop(ST* ps);
STDataType StackTop(ST* ps);
```

```c
int StackSize(ST* ps);
bool StackEmpty(ST* ps);

void StackInit(ST* ps)
{
    assert(ps);

    ps->a = (STDataType*)malloc(sizeof(STDataType) * 4);
    if (ps->a == NULL)
    {
        printf("malloc fail\n");
        exit(-1);
    }

    ps->capacity = 4;
    ps->top = 0;
}

void StackDestory(ST* ps)
{
    assert(ps);
    free(ps->a);
    ps->a = NULL;
    ps->top = ps->capacity = 0;
}

// 入栈
void StackPush(ST* ps, STDataType x)
{
    assert(ps);

    // 满了-》增容
    if (ps->top == ps->capacity)
    {
        STDataType* tmp = (STDataType*)realloc(ps->a, ps->capacity * 2 *
sizeof(STDataType));
        if (tmp == NULL)
        {
            printf("realloc fail\n");
            exit(-1);
        }
        else
        {
            ps->a = tmp;
            ps->capacity *= 2;
        }
    }

    ps->a[ps->top] = x;
    ps->top++;
}

// 出栈
void StackPop(ST* ps)
{
    assert(ps);
    // 栈空了，调用Pop，直接中止程序报错
```

```
    assert(ps->top > 0);

    //ps->a[ps->top - 1] = 0;
    ps->top--;
}

STDataType StackTop(ST* ps)
{
    assert(ps);
    // 栈空了，调用Top，直接中止程序报错
    assert(ps->top > 0);

    return ps->a[ps->top - 1];
}

int StackSize(ST* ps)
{
    assert(ps);

    return ps->top;
}

bool StackEmpty(ST* ps)
{
    assert(ps);

    return ps->top == 0;
}

Stack path;
Stack minpath;
////////////////////////////////////////////////////////////////

void PrintMaze(int** maze, int N, int M)
{
    for (int i = 0; i < N; ++i)
    {
        for (int j = 0; j < M; ++j)
        {
            printf("%d ", maze[i][j]);
        }
        printf("\n");
    }
    printf("\n");
}

// 输出栈里面的坐标路径
void PirntPath(Stack* ps)
{
    // path数据倒到rPath
    Stack rPath;
    StackInit(&rPath);
    while (!StackEmpty(ps))
    {
        StackPush(&rPath, StackTop(ps));
        StackPop(ps);
    }
```

```
    while (StackSize(&rPath) > 1)
    {
        PT top = StackTop(&rPath);
        printf("[%d,%d],", top.row, top.col);
        StackPop(&rPath);
    }

    PT top = StackTop(&rPath);
    printf("[%d,%d]", top.row, top.col);
    StackPop(&rPath);

    StackDestory(&rPath);
}

bool IsPass(int** maze, int N, int M, PT pos)
{
    if (pos.row >= 0 && pos.row < N
        && pos.col >= 0 && pos.col < M
        && maze[pos.row][pos.col] == 1)
    {
        return true;
    }
    else
    {
        return false;
    }
}

void StackCopy(Stack* ppath, Stack* pcopy)
{
    pcopy->a = (STDataType*)malloc(sizeof(STDataType*)*ppath->capacity);
    memcpy(pcopy->a, ppath->a, sizeof(STDataType)*ppath->top);
    pcopy->top = ppath->top;
    pcopy->capacity = ppath->capacity;
}

void GetMazePath(int** maze, int N, int M, PT cur, int P)
{
    StackPush(&path, cur);

    // 如果走到出口
    if (cur.row == 0 && cur.col == M - 1)
    {
        // 找到了更短的路径，更新minpath;
        if (P >= 0 && StackEmpty(&minpath)
            || StackSize(&path) < StackSize(&minpath))
        {
            StackDestory(&minpath);
            StackCopy(&path, &minpath);
        }
    }

    // 探测cur位置得上下左右四个方向
    PT next;
    maze[cur.row][cur.col] = 2;

    // 上
    next = cur;
```

```
    next.row -= 1;
    if (IsPass(maze, N, M, next))
    {
        GetMazePath(maze, N, M, next, P - 3);

    }

    // 下
    next = cur;
    next.row += 1;
    if (IsPass(maze, N, M, next))
    {
        GetMazePath(maze, N, M, next, P);
    }


    // 左
    next = cur;
    next.col -= 1;
    if (IsPass(maze, N, M, next))
    {
        GetMazePath(maze, N, M, next, P - 1);
    }

    // 右
    next = cur;
    next.col += 1;
    if (IsPass(maze, N, M, next))
    {
        GetMazePath(maze, N, M, next, P - 1);
    }

    // 恢复一下，可能两条通路有一段路是重叠的，不恢复，另一条就走不通了
    maze[cur.row][cur.col] = 1;
    StackPop(&path);
}

int main()
{
    int N = 0, M = 0, P = 0;
    while (scanf("%d%d%d", &N, &M, &P) != EOF)
    {
        // int a[n][m]; // vs2013 不支持
        // 动态开辟二维数组
        int** maze = (int**)malloc(sizeof(int*)*N);
        for (int i = 0; i < N; ++i)
        {
            maze[i] = (int*)malloc(sizeof(int)*M);
        }

        // 二维数组得输入
        for (int i = 0; i < N; ++i)
        {
            for (int j = 0; j < M; ++j)
            {
                scanf("%d", &maze[i][j]);
            }
        }
```

```
        StackInit(&path);
        StackInit(&minpath);
        // PrintMaze(maze, N, M);
        PT entry = { 0, 0 };
        GetMazePath(maze, N, M, entry, P);

    if(!StackEmpty(&minpath))
    {
        PirntPath(&minpath);
    }
    else
    {
        printf("Can not escape!\n");
    }


        StackDestory(&path);
        StackDestory(&minpath);

        for (int i = 0; i < N; ++i)
        {
            free(maze[i]);
        }
        free(maze);
        maze = NULL;
    }

    return 0;
}
```