数据结构试验二　　老鼠走迷宫

学生姓名：岳宇轩

学号：19020011038

专业：19 慧与

指导老师：纪筱鹏

一、实验思路（调试分析）：

用深度优先搜索实现，使用堆栈存储老鼠走过的路径。**最终实现最短路径的求解。**

我用一个二维数组 map[10][12]来表示地图。其元素有三种取值：0 代表

无墙、没走过；1 代表墙；2 代表无墙、已走过。

首先输入 start 点坐标(即为其在数中的行列坐标)，若输入不合法（越界，或 start 在墙中）则需重新输入。

1.先判断 start 是否有后继。这里我做的比较繁琐，是把每种情况都判断了。其实可以在地图外再包一层墙，这样就可以所有点用一种方式来判断了。

若 start 点无问题，则把它压入栈 s。

2.定义一个栈 minPath 用来存放最短路径

3.用 FillStack 函数填充 s。若失败则无出口。

4.若填充成功，判断路径长度。若为 2，则已经是最短路径，直接输出。

5.若其它情况，则先把 s 的内容赋给 minPath 作为起始最小路径，对 s 修改方向后再进行填充。若填充后的路径更小，则更新 minPath。重复上述操作直至遍历全部路径。

6.释放空间

具体细节见注释

二、心得(实验总结)：

1.堆栈不能直接复制，比如 s1 = s2 这样。要用一个额外的栈，把 s2 的内容输出到额外栈，额外栈此时倒序存放复制的内容，再将额外栈输出到 s1 和 s2（注意这里也要输出到 s2，否则 s2 就是空的了）。

2.用栈来代表路径。栈中元素就是地图上的一个点

3.我觉得我做的比较好的一个地方是提炼了 FillStack 和 RidirectStack 这两个函数。

FillStack 的起始条件是一条路径，然后这个函数它会填充这条路径，如果能找到出口，则返回 true，如果找

不到出口，则返回 false。

RidirectStack 的作用是修改路径。起始条件是一条路径，它拿出路径的最后一个节点，并修改最后一个节点的方向。如果这个节点不能再进行修改了，则删除该点，再倒数第二个，修改方向.....依次直到拿出了起始点。如果起始点也不能修改方向了，则是空栈。

这个函数还是比较关键的，在填充路径和求取最小路径时都有用到

4.在 FillStack 里用 dowhile 循环是因为，循环截止的条件 start 点也可能满足，所以要先执行一次。

5.地图上的点需要有三种表示：有墙，

无墙走过，无墙没走过。如果没有标记已经走过的路径，当地图有形如：

1 1 0 0 0 0 0 1

1 1 0 1 1 1 0 1

1 1 0 0 0 0 0 1

这样的圈时，就停不下来了。

7.很多地方各种边界条件都要卡（比如判断下一个点的方向时，回退修改路径时等等），有时候一个地方忘记限制了（比如没卡数组边界，起始点要单独拿出来讨论）程序就跑半天跑不完了*崩溃*。还是要细致一点。

三、实验结果

```
D:\CLionFiles\cmake-build-debug\CLionFiles.exe
1 0 1 1 0 1 1 1 1 1 1 1
1 0 1 1 0 1 1 1 1 1 1 1
0 0 0 0 0 0 0 0 1 1 1 1
1 0 1 1 1 0 1 0 1 1 1 1
1 0 1 1 1 0 1 0 0 0 0 0
1 0 0 0 1 0 1 1 1 0 1 1
1 0 1 0 1 0 0 1 1 0 1 1
1 0 1 0 1 1 0 1 1 0 1 1
1 0 0 0 1 1 0 1 1 0 1 1
1 0 1 0 1 1 0 1 1 0 1 1
Please enter the position of the init point : x(0<=x<=9),y(0<=y<=11):5 2
The min size path is:START--RIGHT--DOWN--DOWN--DOWN
--DOWN--DOWN--Export
Process finished with exit code 0
```

```
D:\CLionFiles\cmake-build-debug\CLionFiles.exe
1 0 1 1 0 1 1 1 1 1 1 1
1 0 1 1 0 1 1 1 1 1 1 1
0 0 0 0 0 0 0 0 1 1 1 1
1 0 1 1 1 0 1 0 1 1 1 1
1 0 1 1 1 0 1 0 0 0 0 0
1 0 0 0 1 0 1 1 1 0 1 1
1 0 1 0 1 0 0 1 1 0 1 1
1 0 1 0 1 1 0 1 1 0 1 1
1 0 0 0 1 1 0 1 1 0 1 1
1 0 1 0 1 1 0 1 1 0 1 1
Please enter the position of the init point : x(0<=x<=9),y(0<=y<=11):5 2
The min size path is:START--RIGHT--DOWN--DOWN--DOWN
--DOWN--DOWN--Export
Process finished with exit code 0
```

```
D:\CLionFiles\cmake-build-debug\CLionFiles.exe
1 0 1 1 0 1 1 1 1 1 1 1
1 0 1 1 0 1 1 1 1 1 1 1
0 0 0 0 0 0 0 0 1 1 1 1
1 0 1 1 1 0 1 0 1 1 1 1
1 0 1 1 1 0 1 0 0 0 0 0
1 0 0 0 1 0 1 1 1 0 1 1
1 0 1 0 1 0 0 1 1 0 1 1
1 0 1 0 1 1 0 1 1 0 1 1
1 0 0 0 1 1 0 1 1 0 1 1
1 0 1 0 1 1 0 1 1 0 1 1
Please enter the position of the init point : x(0<=x<=9),y(0<=y<=11):2 1
The min size path is:START--LEFT--LEFT--Export
Process finished with exit code 0
```

```
D:\CLionFiles\cmake-build-debug\CLionFiles.exe
1 0 1 1 0 1 1 1 1 1 1 1
1 0 1 1 0 1 1 1 1 1 1 1
0 0 0 0 0 0 0 0 1 1 1 1
1 0 1 1 1 0 1 0 1 1 1 1
1 0 1 1 1 0 1 0 0 0 0 0
1 0 0 0 1 0 1 1 1 0 1 1
1 0 1 0 1 0 0 1 1 0 1 1
1 0 1 0 1 1 0 1 1 0 1 1
1 0 0 0 1 1 0 1 1 0 1 1
1 0 1 0 1 1 0 1 1 0 1 1
Please enter the position of the init point : x(0<=x<=9),y(0<=y<=11):2 1
The min size path is:START--LEFT--LEFT--Export
Process finished with exit code 0
```

分析上述结果，如果按照上右下左的顺序，则路径应该是：

START--UP--UP--UP--EXPORT

找寻最短路径，则是 START--LEFT--LEFT--EXPORT

四、CODE：

```
#include <iostream>


enum direction {
```

```
    UP, RIGHT, DOWN, LEFT
};

typedef struct {
    int x;  // 节点横坐标
    int y;  // 节点纵坐标
    enum direction dir;  // 节点下一步移动方向
} Elem;

typedef struct Node {
    Elem data;
    struct Node *next;
} Node;

typedef Node *Stack;

void InitStack(Stack *s) {  // 初始化堆栈
    *s = (Node *) malloc(sizeof(Node));
    (*s)->next = nullptr;
}

bool StackEmpty(Stack s) {  // 判断堆栈是否为空
```

```
    if (s->next == nullptr)
        return true;
    else
        return false;
}

void Push(Stack s, Elem e) {  // 压栈
    Node *t = (Node *) malloc(sizeof(Node));
    t->data = e;
    t->next = s->next;
    s->next = t;
}

void Pop(Stack s) {  // 出栈
    if (StackEmpty(s))
        return;
    Node *t = s->next;
    s->next = t->next;
    free(t);
}

Elem Top(Stack s) {  // 获取栈顶元素
```

```
    return s->next->data;
}

void DeleteStack(Stack s) {  // 删除栈
    while (s != nullptr) {
        Node *t = s;
        s = s->next;
        free(t);
    }
}

void ClearStack(Stack s) {  // 清空栈
    while (!StackEmpty(s)) {
        Pop(s);
    }
}

int getStackLength(Stack s) {  // 获得栈长度
    int num = 0;
    while (s->next != nullptr) {
        num += 1;
        s = s->next;
```

```
    }
    return num;
}

void PrintStack(Stack s) {  // 打印路径
    Stack t;
    InitStack(&t);
    Elem temp_elem;
    while (!StackEmpty(s)) {
        temp_elem = Top(s);
        Pop(s);
        Push(t, temp_elem);
    }
    printf("START--");
    while (!StackEmpty(t)) {

        temp_elem = Top(t);
        switch (temp_elem.dir) {
            case UP:
                printf("UP");
                break;
            case RIGHT:
```

```
                printf("RIGHT");
                break;
            case DOWN:
                printf("DOWN");
                break;
            case LEFT:
                printf("LEFT");
                break;
        }
        temp_elem = Top(t);
        Pop(t);
        Push(s, temp_elem);
        printf("--");
    }
    printf("Export");
}

void CopyStack(Stack s1, Stack s2) {  // 栈复制
    Stack temp;
    InitStack(&temp);
    Elem elem;
    while (!StackEmpty(s2)) {
```

```
        elem = Top(s2);
        Pop(s2);
        Push(temp, elem);
    }
    Elem t;
    while (!StackEmpty(temp)) {
        elem = Top(temp);
        t = elem;
        Pop(temp);
        Push(s1, elem);
        Push(s2, t);
    }
}


  // 初始化地图
int map[10][12] = {{1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1},
                   {1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1},
                   {0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1},
                   {1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1},
                   {1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0},
                   {1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1},
                   {1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1},
```

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

void RedirectStack(Stack s) {  // 修改路径。初始条件：一条路径；函数结果：修改路径尾部的方向
    bool flag_stop = false;  // 停止循环的标志
    Elem top, next;
    while (!StackEmpty(s) && !flag_stop) {
        top = Top(s);
        Pop(s);  // 拿出栈顶元素
        if (StackEmpty(s)) {  // 如果拿出了起点
            switch (top.dir) {  // 判断起点的指向
                case UP:  // 如果向上，则看是否可以向右
                    if (top.y != 11 && !map[top.x][top.y + 1]) {  // 该点不在地图右边缘而且该点右侧不是墙且没有走过
                        top.dir = RIGHT;  // 修改方向向右
                        Push(s, top);  // 将该点压栈
                        flag_stop = true;  // 停止循环
                    } else if (top.x != 9 && !map[top.x +
```

```
1][top.y]) {
                        top.dir = DOWN;
                        Push(s, top);
                        flag_stop = true;
                    } else if (top.y != 0
&& !map[top.x][top.y - 1]) {
                        top.dir = LEFT;
                        Push(s, top);
                        flag_stop = true;
                    }
                    break;
                case RIGHT:  // 如果向右，看是否可以向
下
                    if (top.x != 9 && !map[top.x +
1][top.y]) {
                        top.dir = DOWN;
                        Push(s, top);
                        flag_stop = true;
                    } else if (top.y != 0
&& !map[top.x][top.y - 1]) {
                        top.dir = LEFT;
                        Push(s, top);
```

```
                    flag_stop = true;
                }
                break;
            case DOWN:  // 如果向下，看是否可以向
左
                if (top.y != 0 && !map[top.x][top.y
- 1]) {
                    top.dir = LEFT;
                    Push(s, top);
                    flag_stop = true;
                }
                break;
            case LEFT:  // 如果向左，删除该点即可
                map[top.x][top.y] = 0;  // 将该点
置为没走过（0）
                break;
        }
    } else {  // 拿出的不是起点
        next = Top(s);
        switch (top.dir) {
            case UP:
                if (top.y != 11
```

```
&& !map[top.x][top.y + 1] && next.dir != LEFT) {  //
该点不在地图右边缘且该点右边不是墙且没有走过且上一步不
是向左走
                        top.dir = RIGHT;
                        Push(s, top);
                        flag_stop = true;
                    } else if (top.x != 9 && !map[top.x +
1][top.y] && next.dir != UP) {
                        top.dir = DOWN;
                        Push(s, top);
                        flag_stop = true;
                    } else if (top.y != 0
&& !map[top.x][top.y - 1] && next.dir != RIGHT) {
                        top.dir = LEFT;
                        Push(s, top);
                        flag_stop = true;
                    }
                    break;
                case RIGHT:
                    if (top.x != 9 && !map[top.x +
1][top.y] && next.dir != UP) {
                        top.dir = DOWN;
```

```
                    Push(s, top);
                    flag_stop = true;
                } else if (top.y != 0
&& !map[top.x][top.y - 1] && next.dir != RIGHT) {
                    top.dir = LEFT;
                    Push(s, top);
                    flag_stop = true;
                }
                break;
            case DOWN:
                if (top.y != 0 && !map[top.x][top.y
- 1] && next.dir != RIGHT) {
                    top.dir = LEFT;
                    Push(s, top);
                    flag_stop = true;
                }
                break;
            case LEFT:
                map[top.x][top.y] = 0;  //
                break;
        }
    }
```

```
    }
}

bool FillStack(Stack s) {
  // 填充路径。起始条件：非空栈。结果：将路径填充至出口并返回 true，若当前条件下无出口则返回 false
    bool has_export = true;  // 返回标志
    Elem top = Top(s);  // 获取栈顶元素

    do {
        Elem nextPosition;  // 定义一变量代表下一个点
        if (top.dir == UP) {  // 如果路径最后是向上的
            nextPosition.x = top.x - 1;
            nextPosition.y = top.y;  // 给下一个点赋值
        } else if (top.dir == RIGHT) {
            nextPosition.x = top.x;
            nextPosition.y = top.y + 1;
        } else if (top.dir == DOWN) {
            nextPosition.x = top.x + 1;
            nextPosition.y = top.y;
        } else {
            nextPosition.x = top.x;
```

```
            nextPosition.y = top.y - 1;
        }


        if (nextPosition.x == 0 || nextPosition.x == 9 ||
nextPosition.y == 0 || nextPosition.y == 11) {  // 下一
个点在地图的边界上
            if (nextPosition.x == 0)  // 在上边界
                nextPosition.dir = UP;  // 向上走
            else if (nextPosition.y == 11)
                nextPosition.dir = RIGHT;
            else if (nextPosition.x == 9)
                nextPosition.dir = DOWN;
            else
                nextPosition.dir = LEFT;
            Push(s, nextPosition);
            map[nextPosition.x][nextPosition.y] = 2;
        } else {  // 不在边界
            if (!map[nextPosition.x - 1][nextPosition.y]
&& top.dir != DOWN) {  // 先看能不能向上走，条件为：
向上无墙且没走过，且该点的方向不是向下
```

```
                nextPosition.dir = UP;
                Push(s, nextPosition);
                map[nextPosition.x][nextPosition.y] =
2;  // 置地图上该点为 2，表示已走过
            } else if
(!map[nextPosition.x][nextPosition.y + 1] && top.dir !=
LEFT) {

                nextPosition.dir = RIGHT;
                Push(s, nextPosition);
                map[nextPosition.x][nextPosition.y] =
2;
            } else if (!map[nextPosition.x +
1][nextPosition.y] && top.dir != UP) {

                nextPosition.dir = DOWN;
                Push(s, nextPosition);
                map[nextPosition.x][nextPosition.y] =
2;
            } else if
(!map[nextPosition.x][nextPosition.y - 1] && top.dir !=
RIGHT) {
```

```
                nextPosition.dir = LEFT;
                Push(s, nextPosition);
                map[nextPosition.x][nextPosition.y] =
2;
            } else {

                RedirectStack(s);  // 该点走不通，修改路
径
            }
        }
        if (!StackEmpty(s))
            top = Top(s);
    } while (top.x != 0 && top.x != 9 && top.y != 0 &&
top.y != 11 && !StackEmpty(s));  // 循环截止条件为到
达边界（出口）或者找不到出口（空栈）


    if (StackEmpty(s))  // 空栈，即找不到出口
        has_export = false;  // 置标志为 false
    return has_export;
```

```
}


int main() {

    // 输出地图
    for (int i = 0; i < 10; i++) {
        for (int j = 0; j < 12; j++) {
            printf("%d ", map[i][j]);
        }
        printf("\n");
    }

    // 初始化栈
    Stack s;
    InitStack(&s);


    int init_X, init_Y;
```

```
  // 输入 start 点，并判断是否合法
    printf("Please enter the position of the init point :
x(0<=x<=9),y(0<=y<=11):");
    scanf_s("%d %d", &init_X, &init_Y);
    while (map[init_X][init_Y] || init_X < 0 || init_X > 9
|| init_Y < 0 || init_Y > 11) {
        printf("Wrong!The init point should not in the
wall or beyond the map!");
        printf("Please enter the position of the init
point : x(0<=x<=9),y(0<=y<=11):");
        scanf_s("%d %d", &init_X, &init_Y);
    }

  // 得到 start 点，判断其是否可以走出下一步
    Elem init_point;
    init_point.x = init_X;
    init_point.y = init_Y;


    bool flag_init = true;
    if (init_X == 0 && init_Y == 0) {
```

```
            if (map[1][0] + map[0][1] == 2)
                flag_init = false;
        } else if (init_X == 9 && init_Y == 0) {

            if (map[8][0] + map[9][1] == 2)
                flag_init = false;
        } else if (init_X == 0 && init_Y == 11) {

            if (map[0][10] + map[1][11] == 2)
                flag_init = false;
        } else if (init_X == 9 && init_Y == 11) {

            if (map[9][10] + map[8][11] == 2)
                flag_init = false;
        } else if (init_X == 0) {

            if (map[0][init_Y - 1] + map[1][init_Y] +
map[0][init_Y + 1] == 3)
                flag_init = false;
        } else if (init_Y == 0) {

            if (map[init_X - 1][0] + map[init_X][1] +
```

```
map[init_X + 1][0] == 3)
                flag_init = false;
        } else if (init_X == 9) {

            if (map[9][init_Y - 1] + map[8][init_Y] +
map[9][init_Y + 1] == 3)
                flag_init = false;
        } else if (init_Y == 11) {

            if (map[init_X - 1][11] + map[init_X][10] +
map[init_X + 1][11] == 3)
                flag_init = false;
        } else {

            if (map[init_X][init_Y - 1] + map[init_X -
1][init_Y] + map[init_X][init_Y + 1] + map[init_X +
1][init_Y] == 4)
                flag_init = false;
        }

        if (!flag_init) {  // 起始点没有后继
            printf("No Export!\n");
```

```
        exit(0);
    }

      // start 点有后继
    if (init_X - 1 >= 0 && map[init_X - 1][init_Y] == 0)
{
          // 向上走不越界且向上无墙且没走过
        init_point.dir = UP;  // 向上走
    } else if (init_Y + 1 <= 11 && map[init_X][init_Y + 1]
== 0) {

        init_point.dir = RIGHT;
    } else if (init_X + 1 <= 9 && map[init_X + 1][init_Y]
== 0) {

        init_point.dir = DOWN;
    } else {

        init_point.dir = LEFT;
    }
    Push(s, init_point);  // 将 start 点压栈
    Stack minPath;  // 定义一个栈 minPath 用于存放最
```

```
小路径
    InitStack(&minPath);
    if (FillStack(s)) {  // 如果可以根据 start 点填充出一条
抵达出口的路径
        printf("The min size path is:");
        if (getStackLength(s) == 2) {  // 路径长度为 2，
已经是最小路径
            PrintStack(s);
            exit(0);
        } else {  // 路径长度>2
            CopyStack(minPath, s);  // 将 s 作为初始最
小路径
            int minLength = getStackLength(s);
            int tempLength;
            while (!StackEmpty(s)) {  // 循环截止条件为
空栈，即遍历了所有可能
                RedirectStack(s);  // 修改路径
                if (StackEmpty(s))  // 空栈则退出循环
                    break;
                else {
                    if (FillStack(s)) {  // 可以填充路径
                        tempLength =
```

```
getStackLength(s);
                        if (tempLength < minLength)
{  // 比较填充后的路径长度和最小路径的长度，如果更小
                        ClearStack(minPath);  //
清空最小路径栈
                        CopyStack(minPath, s);
// 将填充后的路径赋给 minPath
                        minLength = tempLength;
// 更新最小长度
                    }
                }
            }
        }
        PrintStack(minPath);  // 输出最小路径
    }
} else
    printf("NO EXPORT");
  // 释放空间
    DeleteStack(s);
    DeleteStack(minPath);

    return 0;
```