《数据结构与算法》实验报告

实验名称	迷宫问题求解				
姓名	叶鹏	学号	20020007095	日期	2022/3/18
实验内容	用一个 M*N 的方阵表示迷宫，能够区分迷宫中的通路和障碍。编写非递归程序对给定的不同迷宫和不同起点和终点，输出一条通路或者输出没有通路的结论。以链表作为栈的存储结构类型。输出形式为连续的三元组(i, j, d)，其中(i, j)为迷宫中的一个坐标，d 表示走到下一坐标的方向（1，2，3，4，用东南西北输出）				
实验目的	利用栈的迷宫求解，编写可运行程序				

实验步骤	已实现 ✓ 自己实现链式栈，能够尝试不同迷宫和任意起点和终点 ✓ 实现自动判别起点和终点的相对位置来确定搜索方向的优先顺序 ✓ 在以上基础上能打印出二维迷宫和通路顺序 详细步骤 1. 考虑使用栈来实现深度优先搜索算法，题目要求自己手写链式栈结构，于是先定义几个数据类型 2. 需要一个表示 coordinate 的类型，可以使用 pair 类，但响应号召写一个结构体

```
10  struct PosType
11  {
12      int x;
13      int y;
14      string dir;//方向
15      int index;//序号
16      PosType() : x(0), y(0), dir(""), index(0) {}
17      PosType(int x, int y, string dir, int index) :x(x), y(y), dir(dir), index
18  };
```

picture 1 **表示坐标的结构体** PosType

3. 实现一个以 PosType 为 value 的链表结构

```
17  struct ListNode {
18      PosType val;
19      ListNode* next;
20      ListNode() : val(PosType()), next(nullptr) {}
21      ListNode(PosType p) : val(p), next(nullptr) {}
22      ListNode(PosType p, ListNode* next) : val(p), next(next) {}
23  };
```

picture 2 **链表** ListNode

4. 利用**链表实现栈结构**，支持基本的入栈(Push)、出栈(Pop)、返回栈顶(Top)、判断是否为空(Empty)、打印栈(printStack)操作

```cpp
class MyStack {
private:
    ListNode* dump;
    ListNode* head;

public:
    MyStack() {
        dump = new ListNode(PosType());
        head = nullptr;
    }

    void push(PosType p) {
        ListNode* newNode = new ListNode(p);
        if (empty()) {
            head = newNode;
            dump->next = head;
        }
        else {
            newNode->next = head;
            head = newNode;
            dump->next = head;
        }
    }

    bool pop() {
        if (empty()) return false;
        ListNode* delNode = head;
        head = head->next;
        dump->next = head;
        delete delNode;
        return true;
    }

    PosType top() {
        if (empty())return PosType(-1, -1);
        return head->val;
    }

    bool empty() {
        if (dump->next) return false;
        else return true;
    }

    void printStack() {
        if (empty())return;
        ListNode* cur = head;
        while (cur) {
            cout << "( " << cur->val.x << ", " << cur->val.y << " ) -> ";
            cur = cur->next;
        }
        cout << endl;
    }
};

/**
```

picture 3 **栈的链表实现**

5. **实现自动判别起点和终点的相对位置函数**，使用贪心算法，因为对于这种路径求解问题，是没有办法直接实现判别起点向终点的最优方向的，存在的情况太多，我们只好通过贪心实现**局部最优解**，不过这个最优解也很难是全局最优解

```
45  // 贪心求起点最优方向，以起点到终点的横纵坐标之差为准
46  int getDirection(int x1, int y1, int x2, int y2) {
47      int diff_x, diff_y;
48      diff_x = abs(x1 - x2);
49      diff_y = abs(y1 - y2);
50      if (diff_x < diff_y) {
51          if (x2 > x1)
52              return 1;
53          else
54              return 0;
55      }
56      else {
57          if (y2 > y1)
58              return 3;
59          else
60              return 2;
61      }
62  }
```

picture 4 **贪心求方向**

该函数返回的值为方向数组的索引值，其对应的分别是：

```
8      string direction[4] = { "北", "南", "西", "东" };
```

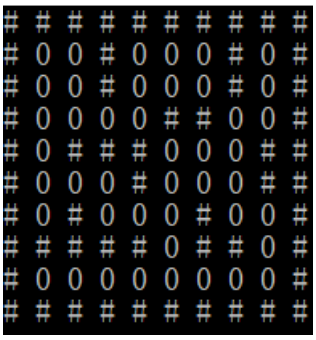
picture 5 **方向数组**

6. 接下来进入求解函数部分

```
11  void maze(vector<vector<char> >& grid, PosType current, PosType end) {
12      int nRow, nCol;
13      nRow = grid.size();
14      nCol = grid[0].size();
15      MyStack* stk = new MyStack();
16      stk->push(current);
17      grid[current.x][current.y] = '1';
18
19      while (!stk->empty()) {
20          PosType pos = stk->top();
21          stk->pop();
22
23          if (pos.x == end.x && pos.y == end.y) {
24              grid[pos.x][pos.y] = '0';
25              printMaze(grid);
26              return;
27          }
28
29          int dir = getDirection(pos.x, pos.y, end.x, end.y);
30
31          PosType newPos;
32          for (int i = 0; i < 4; ++i) {
33              newPos.x = pos.x + dx[dir % 4];
34              newPos.y = pos.y + dy[dir % 4];
35              dir++;
36              if (newPos.x >= 0 && newPos.x < nRow
37                  && newPos.y >= 0 && newPos.y < nCol
38                  && grid[newPos.x][newPos.y] == '0' ) {
39                  stk->push(newPos);
40                  grid[newPos.x][newPos.y] = '1';
41              }
42          }
43      }
44
45  }
```

常规思路，通过栈结构实现深度优先搜索，先将起始点压栈，只要栈不为空就**持续循环**，每一次循环，以当前点为中心，判断上下左右四个方向是否满足搜索条件(在图内且可走)，若**满足条件则压入栈**内。直到栈为空为止(也就是没找到可行路径)，如果当前点就是终点，则说明找到路径，返回值即可。

通过上述条件我们可以求得一条路径



picture 6 **原始迷宫**

picture 7 **解得一条路径**

可是我们发现，虽然求解到一条路径，但是其既不是最优解，也没有每一步的行动轨迹，而且不直观，这也就是普通搜索算法得局限性，没有办法保存每一步的路径，只能求得是否存在满足条件的路径，于是我们考虑改进算法(最优解无法解决，最优解需要使用 bfs+队列结构)

7. 改进算法，dfs + 回溯

说到之前的算法不能保存路径的原因就是其在压栈过程中存有**冗余信息**，也就是说，在栈中存在**不可行的点**或者**另一条路径的点**，因为栈的 FILO 结构，我们没办法去除这些点，这也导致输出路径中含有无效信息，解决办法就是在每一步可行或不可行路径搜索完后，在栈中弹出该点，也就是回溯，这样的话就可以去除无效信息。

实验步骤

```
66  void dfs(vector<vector<char> >& grid, MyStack* stk, PosType current, PosT
67      int nRow, nCol;
68      nRow = grid.size();
69      nCol = grid[0].size();
70
71      if (current.x == end.x && current.y == end.y) {
72          printMaze(grid);
73          printStack(stk);
74          return;
75      }
76
77      int nx, ny;
78
79      grid[current.x][current.y] = '1';
80
81      int dir = getDirection(current.x, current.y, end.x, end.y);
82
83      for (int i = 0; i < 4; ++i) {
84          nx = current.x + dx[dir % 4];
85          ny = current.y + dy[dir % 4];
86          dir++;
87          if (nx >= 0 && nx < nRow
88              && ny >= 0 && ny < nCol
89              && grid[nx][ny] == '0') {
90              grid[nx][ny] = '1';
91              PosType newPos(nx, ny, direction[i], stk->top().index + 1);
92              stk->push(newPos);
93
94              dfs(grid, stk, newPos, end);
95
96              stk->pop();
97              grid[nx][ny] = '0';
98          }
99      }
100
101 }
```

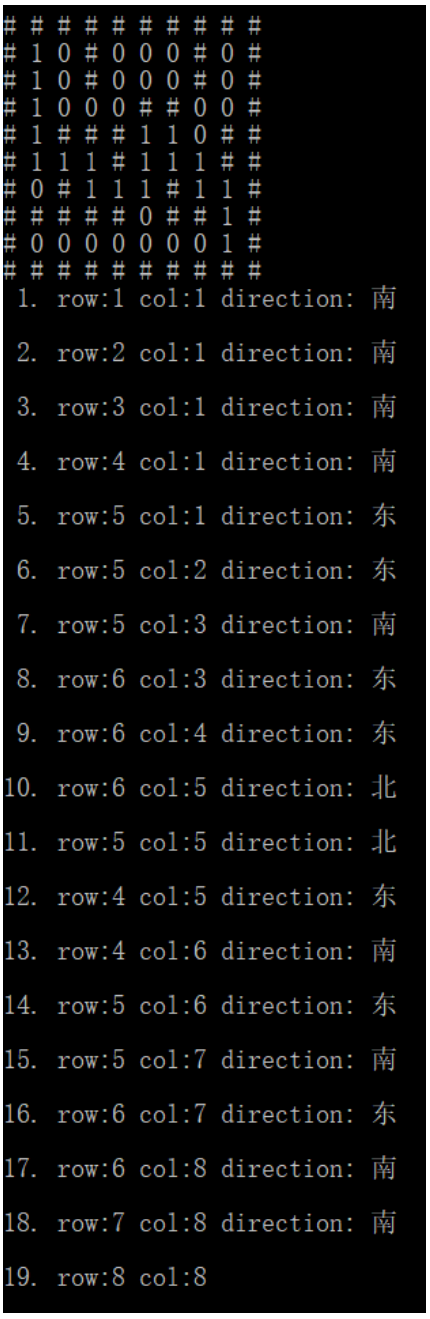
picture 8 **回溯算法**

这样一来我们即保存了路径，而且还能探索不同的路径，因此在搜索完后输出每条路径即可

8. 输出路径和迷宫，打印顺序

有了回溯算法输出路径与顺序便简单了，只需在搜索时记录即可，输出迷宫也是一样的，在我的程序内，一个迷宫以 "#" 为边界，"0" 是可行的点，"1" 是行进过的点，例如下图

picture 9 一个可行解的演示



picture 10 路径的输出，包括顺序，点的位置，以及行进的方向

```
110    void printMaze(vector<vector<char> >& grid) {
111        for (auto it = grid.begin(); it != grid.end(); ++it) {
112            for (int i = 0; i < (*it).size(); ++i) {
113                cout << (*it)[i] << " ";
114            }
115            cout << endl;
116        }
117    }
```

picture 11 迷宫输出函数

```
123    void printStack(MyStack* stk) {
124        MyStack* transfer = new MyStack();
125        ListNode* cur = stk->getHead();
126        while (cur) {
127            transfer->push(cur->val);
128            cur = cur->next;
129        }
130
131        while (!transfer->empty()) {
132            cout << setw(2) << transfer->top().index+1 << ". ";
133            cout << "row:" << transfer->top().x << " ";
134            cout << "col:" << transfer->top().y << " ";
135            transfer->pop();
136            if (transfer->empty()) {
137                cout << endl << endl;
138                break;
139            }
140            cout << "direction: " << transfer->top().dir << " ";
141            cout << endl << endl;
142        }
143    }
```

picture 12 路径输出函数(输出栈)

9. 尝试不同的起点与终点

起点：(3，4)

终点：(8，1)

```
# # # # # # # # # #
# 0 0 # 0 0 0 # 0 #
# 0 0 # 0 0 0 # 0 #
# 1 1 1 1 # # 0 0 #
# 1 # # # 0 0 0 # #
# 1 1 1 # 0 0 0 # #
# 0 # 1 1 1 # 0 0 #
# # # # # 1 # # 0 #
# 1 1 1 1 1 0 0 0 #
# # # # # # # # # #
 1. row:3 col:4 direction: 西
 2. row:3 col:3 direction: 西
 3. row:3 col:2 direction: 西
 4. row:3 col:1 direction: 南
 5. row:4 col:1 direction: 南
 6. row:5 col:1 direction: 东
 7. row:5 col:2 direction: 东
 8. row:5 col:3 direction: 南
 9. row:6 col:3 direction: 东
10. row:6 col:4 direction: 东
11. row:6 col:5 direction: 南
12. row:7 col:5 direction: 南
13. row:8 col:5 direction: 西
14. row:8 col:4 direction: 西
15. row:8 col:3 direction: 西
16. row:8 col:2 direction: 西
17. row:8 col:1
```

10. 尝试不同的迷宫

起点：(3, 2)

终点：(1, 8)

```
# # # # # # # # # # #
# 0 0 # 0 0 0 # 1 #
# 0 0 # 0 0 0 # 1 #
# 1 1 # 0 # # # 1 #
# 1 # # 0 0 0 # 1 #
# 1 1 1 1 # 0 0 1 #
# 0 # 0 1 1 # 0 1 #
# # # 0 # 1 # # 1 #
# 0 0 0 0 1 1 1 1 #
# # # # # # # # # #
 1. row:3 col:2 direction: 西
 2. row:3 col:1 direction: 南
 3. row:4 col:1 direction: 南
 4. row:5 col:1 direction: 东
 5. row:5 col:2 direction: 东
 6. row:5 col:3 direction: 东
 7. row:5 col:4 direction: 南
 8. row:6 col:4 direction: 东
 9. row:6 col:5 direction: 南
10. row:7 col:5 direction: 南
11. row:8 col:5 direction: 东
12. row:8 col:6 direction: 东
13. row:8 col:7 direction: 东
14. row:8 col:8 direction: 北
15. row:7 col:8 direction: 北
16. row:6 col:8 direction: 北
17. row:5 col:8 direction: 北
18. row:4 col:8 direction: 北
19. row:3 col:8 direction: 北
20. row:2 col:8 direction: 北
21. row:1 col:8
```

11. 源代码

链表实现：

```
struct PosType
{
    int x;
    int y;
    string dir;//方向
    int index;//序号
    PosType() : x(0), y(0), dir(""), index(0) {}
    PosType(int x, int y, string dir, int index) :x(x), y(y), dir(dir), index(index) {}
};

struct ListNode {
    PosType val;
    ListNode* next;
    ListNode() : val(PosType()), next(nullptr) {}
    ListNode(PosType p) : val(p), next(nullptr) {}
    ListNode(PosType p, ListNode* next) : val(p), next(next) {}

};
```

栈实现：

```
class MyStack {
private:
    ListNode* dump;
    ListNode* head;

public:
    MyStack() {
        dump = new ListNode(PosType());
        head = nullptr;
    }

    ListNode* getHead() { return head; }

    void push(PosType p) {
        ListNode* newNode = new ListNode(p);
        if (empty()) {
            head = newNode;
            dump->next = head;
        }
        else {
            newNode->next = head;
```

```
            head = newNode;
            dump->next = head;
        }
    }

    bool pop() {
        if (empty()) return false;
        ListNode* delNode = head;
        head = head->next;
        dump->next = head;
        delete delNode;
        return true;
    }

    PosType top() {
        if (empty())return PosType(-1, -1, "", 0);
        return head->val;
    }

    bool empty() {
        if (dump->next) return false;
        else return true;
    }

    void printStack() {
        if (empty())return;
        ListNode* cur = head;
        while (cur) {
            cout << "( " << cur->val.x << ", " << cur->val.y << " ) -> ";
            cur = cur->next;
        }
        cout << endl;
    }
};
```

方法实现(普通方法+回溯方法+输出方法)：

```
class solution {
private:
    int dx[4] = { -1,1,0,0 };
    int dy[4] = { 0,0,-1,1 };
    string direction[4] = { "北", "南", "西", "东" };

public:
```

```
8.      void maze(vector<vector<char> >& grid, PosType current, Pos
    Type end) {
9.          int nRow, nCol;
10.         nRow = grid.size();
11.         nCol = grid[0].size();
12.         MyStack* stk = new MyStack();
13.         stk->push(current);
14.         grid[current.x][current.y] = '1';
15.
16.         while (!stk->empty()) {
17.             PosType pos = stk->top();
18.             stk->pop();
19.
20.             if (pos.x == end.x && pos.y == end.y) {
21.                 grid[pos.x][pos.y] = '0';
22.                 printMaze(grid);
23.                 return;
24.             }
25.
26.             int dir = getDirection(pos.x, pos.y, end.x, end.y);

27.
28.             PosType newPos;
29.             for (int i = 0; i < 4; ++i) {
30.                 newPos.x = pos.x + dx[dir % 4];
31.                 newPos.y = pos.y + dy[dir % 4];
32.                 dir++;
33.                 if (newPos.x >= 0 && newPos.x < nRow
34.                     && newPos.y >= 0 && newPos.y < nCol
35.                     && grid[newPos.x][newPos.y] == '0' ) {
36.                     stk->push(newPos);
37.                     grid[newPos.x][newPos.y] = '1';
38.                 }
39.             }
40.         }
41.
42.     }
43.
44.     // 贪心求起点最优方向，以起点到终点的横纵坐标之差为准
45.     int getDirection(int x1, int y1, int x2, int y2) {
46.         int diff_x, diff_y;
47.         diff_x = abs(x1 - x2);
48.         diff_y = abs(y1 - y2);
49.         if (diff_x < diff_y) {
```

```
            if (x2 > x1)
                return 1;
            else
                return 0;
        }
        else {
            if (y2 > y1)
                return 3;
            else
                return 2;
        }
    }

    void dfs(vector<vector<char> >& grid, MyStack* stk, PosType current, PosType end) {
        int nRow, nCol;
        nRow = grid.size();
        nCol = grid[0].size();

        if (current.x == end.x && current.y == end.y) {
            printMaze(grid);
            printStack(stk);
            return;
        }

        int nx, ny;

        grid[current.x][current.y] = '1';

        int dir = getDirection(current.x, current.y, end.x, end.y);

        for (int i = 0; i < 4; ++i) {
            nx = current.x + dx[dir % 4];
            ny = current.y + dy[dir % 4];
            if (nx >= 0 && nx < nRow
                && ny >= 0 && ny < nCol
                && grid[nx][ny] == '0') {
                grid[nx][ny] = '1';
                PosType newPos(nx, ny, direction[dir % 4], stk->top().index + 1);
                stk->push(newPos);

                dfs(grid, stk, newPos, end);
```

```
91.
92.                 stk->pop();
93.                 grid[nx][ny] = '0';
94.             }
95.             dir++;
96.         }
97.
98.     }
99.
100.       void mazeSolution(vector<vector<char> >& grid, PosType s
    tart, PosType end) {
101.           MyStack* steps = new MyStack();
102.           steps->push(start);
103.
104.           dfs(grid, steps, start, end);
105.       }
106.
107.       void printMaze(vector<vector<char> >& grid) {
108.           for (auto it = grid.begin(); it != grid.end(); ++it)
     {
109.               for (int i = 0; i < (*it).size(); ++i) {
110.                   cout << (*it)[i] << " ";
111.               }
112.               cout << endl;
113.           }
114.       }
115.
116.       void printPosition(PosType p) {
117.           cout << "( " << p.x << ", " << p.y << " )" << endl;

118.       }
119.
120.       void printStack(MyStack* stk) {
121.           MyStack* transfer = new MyStack();
122.           ListNode* cur = stk->getHead();
123.           while (cur) {
124.               transfer->push(cur->val);
125.               cur = cur->next;
126.           }
127.
128.           while (!transfer->empty()) {
129.               cout << setw(2) << transfer->top().index+1 << ".
     ";
130.               cout << "row:" << transfer->top().x << " ";
```

```
131.                cout << "col:" << transfer->top().y << " ";
132.                transfer->pop();
133.                if (transfer->empty()) {
134.                    cout << endl << endl;
135.                    break;
136.                }
137.                cout << "direction: " << transfer->top().dir <<
    " ";
138.                cout << endl << endl;
139.            }
140.        }
141.
142.
143.    };
```

实验总结

本次实验通过栈结构实现深度优先搜索算法，解决了迷宫问题，深度优先搜索是非常经典也是非常常见的计算机学科问题的解决办法，他有很多实现方式，其中栈结构解决是一个非常经典的方案，利用了栈的先入后出的特性，在之后的学习中栈的运用会更加普遍，本次实验打了一个很好的基础