栈与队列的应用

郑悟强 PB22051082

2023.10.4

1 题目 2：括号配对检验

1.1 问题描述

假设一个表达式有英文字母（大、小写）、数字、四则运算符（+，-，*，/）和左右小括号、中括号、大括号构成，以“@”作为表达式的结束符。请编写一个程序检查表达式中的左右大中小括号是否匹配，若匹配，则返回“YES”；否则返回“NO”。

输入格式：

输入文件中第一行是表达式数目 N

之后是需要进行括号配对检测的 N 个表达式。

输出格式：

N 行输出分别对应输入的 N 行表达式，每行都为“YES”或“NO”

1.2 算法的描述

(1) 数据结构的描述：

```
class stack{//栈类
    private:
        char st[100];
        int len=0;
class parenthesis{//括号类
    private:
        char brackets[100];
        int braLen=0;
```

(2) 程序结构的描述：

核心函数：

```
void initBrackets(){//初始化这个串
    char c;
    cout<<"please input the parenthesis:"<<endl;
    while(c!='@'){//@为结束符
        cin>>c;
        brackets[braLen]=c;
        braLen++;
    }
}
```

```
bool isValid(){//判断括号是否匹配
    stack stk;
    for(int i=0;i<braLen;i++){
        switch(brackets[i]){…
    }
    if(stk.isempty()){
        return true;
    }
    return false;
}
```

主函数：

```
int main(){
    parenthesis p;
    int num;//表达式数量
    cout<<"The amount of expressions:";
    cin>>num;
    for(int i=0;i<num;i++){
        p.initBrackets();//输入这个表达式
        if(p.isValid()){//判断是否匹配
            cout<<"the parenthesis is paired"<<endl;
        }
        else{
            cout<<"the parenthesis is not paired"<<endl;
        }
    }
    system("pause");
    return 0;
}
```

1.3 调试分析

```
The amount of expressions:3
please input the parenthesis:
2*[(x+y)/(1-x)]@
the parenthesis is paired
please input the parenthesis:
(25+x)*(a*(a+b+b)}@
the parenthesis is not paired
please input the parenthesis:
{1+2[c-d(7/9)}@
the parenthesis is not paired
```

与预期一致，程序结果正确。

1.4 算法时空分析

时间复杂度上：程序采取顺序入（出）栈，每一个字符比较入栈或出栈，全程只遍历字符串一遍，时间复杂度为O(length)

空间复杂度上：程序中每次只匹配当前字符串的内容，不需引入变量，空间复杂度为 O(1)。

2 题目 3：迷宫求解

2.1 问题描述

有一个 m*n 格的迷宫 (表示有 m 行、n 列)，其中有可走的点也有不可走的点，我们用 1 表示可以走，0 表示不可以走。现在要你编程找出最短的道路，要求所走的路中没有重复的点，走时只能沿着上下左右四个方向。如果没有道路，则输出-1。

2.2 算法的描述

(1) 数据结构的描述：

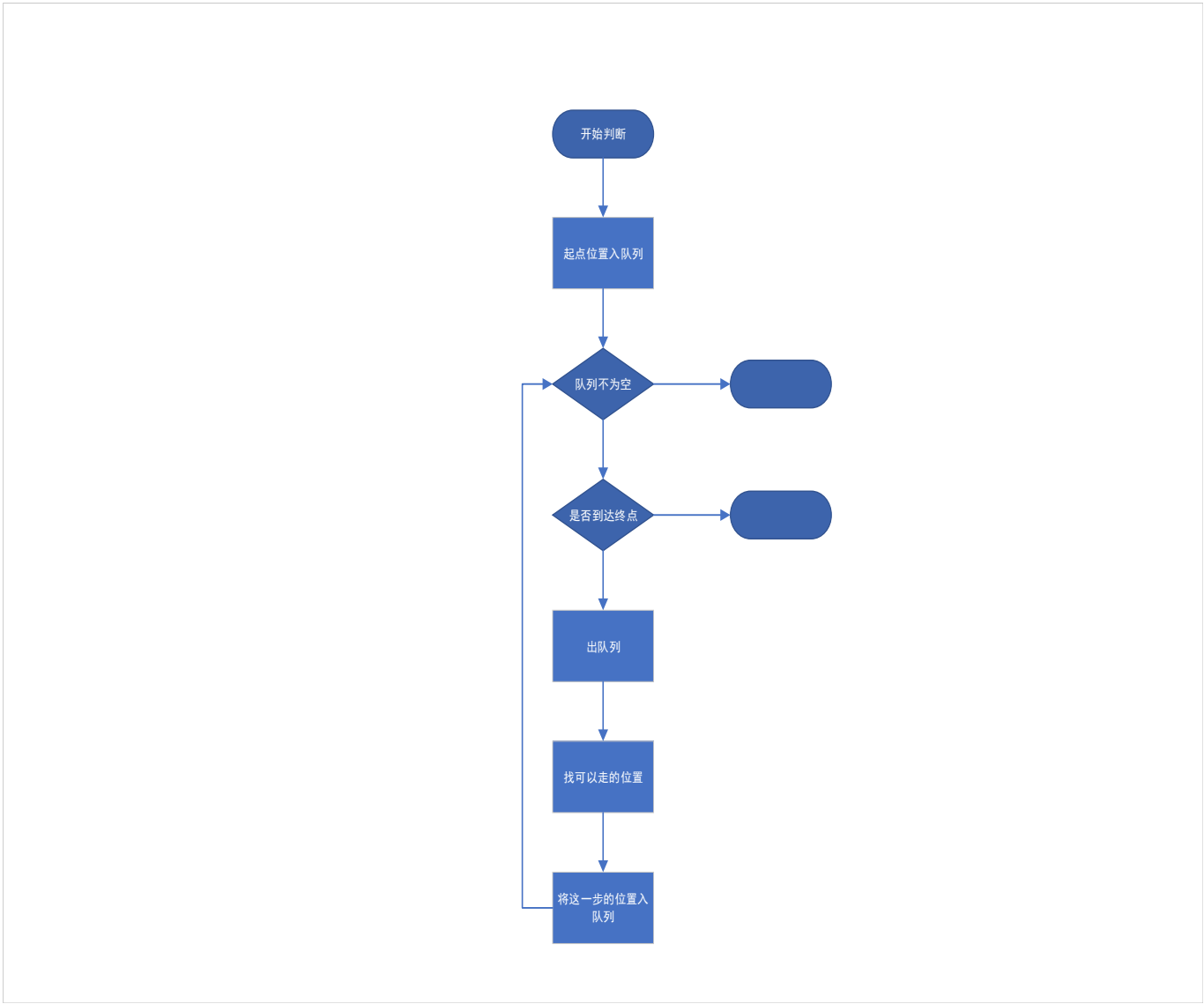
```
class maze {
private:
    //迷宫行数列数
    int col;
    int line;

    //用dx,dy表示各个方向的增量，分别对应左上右下四个方向
    int dx[4] = { -1,1,0,0 };
    int dy[4] = { 0,0,-1,1 };

public:
    int startx;//起始横纵坐标
    int endx;
    int starty;
    int endy;
    int Maze[MAXLINE][MAXCOLUMN];//迷宫
    node pathes[MAXQSIZE];//记录所有遍历过的位置

    void createMaze();//生成迷宫

    void printMaze();//打印迷宫

    int findSolution();//找到最短路径

    void printSolution(int);//打印最短路径
};
```

(2) 程序结构的描述：

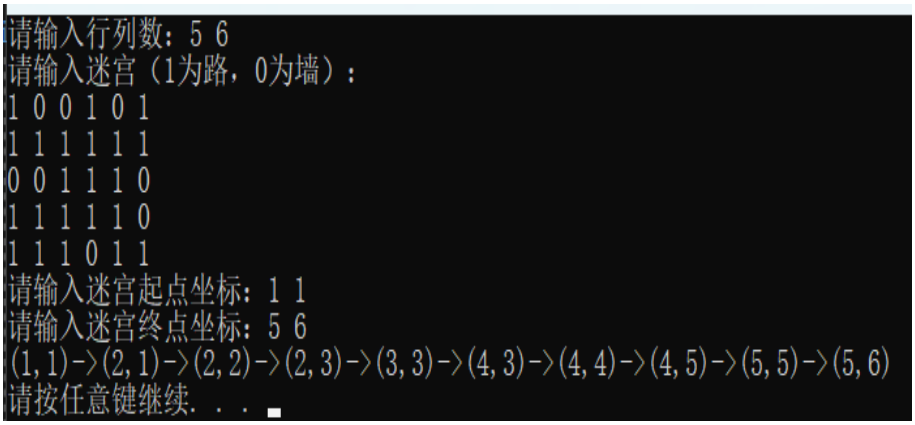
核心函数：

```
int maze::findSolution() {//使用bfs算法找到迷宫最短路径
    int No = 0;//用来记录每个结点在总结点pathes里的位置
    node current;//当前结点
    //初始化起点
    current.x = startx;
    current.y = starty;
    current.step = 0;
    current.prior = -1;
    current.no = No;
    pathes[No] = current;
    No++;
    queue path;//路径队列
    path.push(current);
    while (!path.isEmpty()) {
        current = path.front();
        path.pop();
        if (current.x == endx && current.y == endy) {
            return current.no;
        }
        for (int i = 0; i < 4; i++) {//每一个方向以此遍历
            int cx = current.x + dx[i];
            int cy = current.y + dy[i];
            if (cx<1 || cx>line || cy<1 || cy>col || Maze[cx][cy] == 0 || Maze[cx][cy] == -1) {
                continue;
            }
            node next;
            next.x = cx;
            next.y = cy;
            next.step = current.step + 1;
            next.prior = current.no;
            next.no = No;
            pathes[No] = next;//记录这个位置
            No++;
            path.push(next);
            Maze[next.x][next.y] = -1;//标记已经遍历过的位置
        }
    }
    return -1;
}
```

流程图：



2.3 调试分析



与预期一致，程序结果正确。

2.4 时空复杂度分析

由于 bfs 算法每次向外扩张一格，到能第一次找到终点停止，时间复杂度为 O(minlength)，空间复杂度为 O(minlength)。

3 题目 4：银行业务模拟

3.1 问题描述

客户业务分为两种：

第一种是申请从银行得到一笔资金，即取款或借款；

第二种是向银行投入一笔资金，即存款或还款。

银行有两个服务窗口，相应地有两个队列。客户到达银行后先排第一个队，处理每个客户业务时，如果属于第一种，且申请额超出银行现存资金总额而得不到满足，则立刻排入第二个队等候直至满足时才离开银行; 否则业务处理完后立刻离开银行, 每接待完一个第二种业务的客户，则顺序检查和处理 (如果可能) 第二个队列中的客户，对能满足的申请者予以满足, 不能满足者重新排到第二个队列的队尾。注意，在此检查过程中，一旦银行资金总额少于或等于刚才第一个队列中最后一个客户 (第二种业务) 被接待之前的数额，或者本次已将第二个队列检查或处理了一遍，就停止检查 (因为此时已不可能还有能满足者) 转而继续接待第一个队列的客户。任何时刻都只开一个窗口。假设检查不需要时间，营业时间结束时所有客户立即离开银行。

写一个上述银行业务的事件驱动模拟系统，通过模拟方法求出客户在银行内逗留的平均时间。

3.2 算法的描述

(1) 数据结构的描述

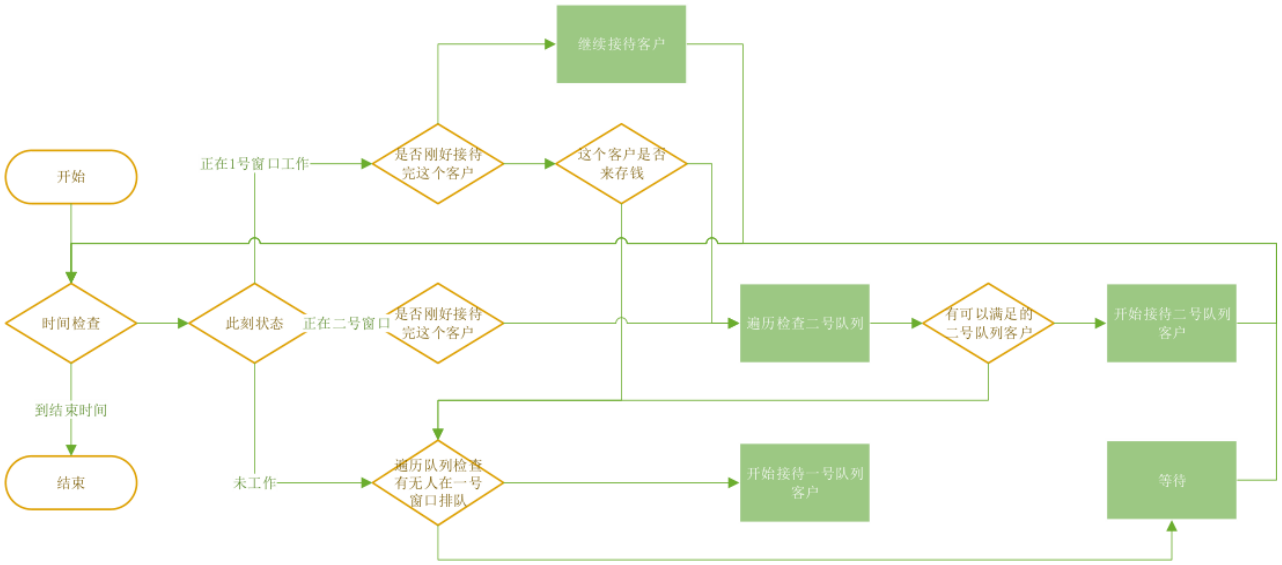
队列的基本功能：

```
class queue {

public:
    int top = 0;
    int base = 0;
    int no[MAXCLIENT];//表示客户编号
    bool isEmpty();

    void push(int num);

    void pop();

    int front();

    int size();
};
```

银行类的基本功能：

```
#pragma once
#include"queue.h"
#include<windows.h>
using namespace std;

class bank {
private:
    //两个队列分别为1队和2队
    queue q1;
    queue q2;
    int N;//银行总人数
    int total;//银行的钱
    int close_time;//营业时长
    int average_time;//客户交易时长
    int client[MAXCLIENT][3];//表示每个客户办理的款额、到达时间、离开时间
    int clock;//时钟

public:
    void initBank();//初始化银行

    void showBank();//输出银行工作状态

    void bankWorking();//银行具体工作
};
```

(2) 程序结构的描述：



3.3 调试分析

```
依次输入 客户总数 初始钱数 营业时间 交易时长
4 10000 600 10
输入第1个客户存（取）款金额 到达时间
-2000 0
输入第2个客户存（取）款金额 到达时间
-11000 10
输入第3个客户存（取）款金额 到达时间
-10000 30
输入第4个客户存（取）款金额 到达时间
2000 50
```

```
time:  600      total money:  0
窗口一
窗口二     2号客户(-11000)

1 号客户等待时间为: 0
2 号客户等待时间为: 590
3 号客户等待时间为: 30
4 号客户等待时间为: 0
平均等待时间为:  155请按任意键继续. . .
```

3.4 时空复杂度分析

模拟实际过程，时空复杂度都为 O(n(人数))。

4 实验体会与分析

通过本次实验，熟练掌握队列、栈的实现和使用。同时也学会了应用队列和栈解决各种实际问题。其中迷宫问题掌握了借助队列实现 bfs 算法进行优先广度搜索。