实验二 栈与队列的应用——银行业务模拟

崔士强 PB22151743

1 问题描述

1.1 设计目标

客户业务分为两种：

1. 申请从银行得到一笔资金，即取款或借款
2. 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。

本程序通过模拟方法求出客户在银行内逗留的平均时间

1.2 输入输出

第一行输入四个数 `N`、`total`、`close_time`、`average_time`，分别表示来银行的总人数、银行开始营业时拥有的款额、今天预计的营业时长和客户交易时长之后的 N 行每行输入两个数 `a`、`b`，第一个数 `a` 为客户办理的款额，用负值和正值分别表示第一类和第二类业务。第二个数 `b` 为客户来到银行的时间

2 算法描述

2.1 数据结构描述

本程序设置两个类`Customer`和`Bank`，用队列处理顾客相关状态：

```
1  typedef Customer QElemType;
2
3  typedef int Status;
4
```

```
typedef struct QNode{
    QElemType     data;
    struct QNode  *next;
}QNode, *QueuePtr;
```

2.2 程序结构描述

程序中用到的两个类的声明如下所示：

```
class Customer
{
    public:
        int money;
        int arrivalTime;
        int waitTime;
        int leaveTime;
        int order;
};
class Bank
{
    public:
        int money;
        int closeTime;
        int serviceTime;
        int waitTimeList[100];
        int pNumber;
        int clock;
        int open;
        int avgWaitTime;
        LinkQueue Queue1, Queue2;
        Status deal(LinkQueue& queue, Customer &customer);
            // 处理queue中一位客户customer的业务
        void checkQueue1();
            // 对Queue1进行一次处理
        void checkQueue2(int benchmark);
            // 对Queue2进行检查
        void close();
            // 结束业务
};
```

3 调试分析

3.1 测试数据

测试过程中选取会导致银行持有资金无法满足需求的测试样例，以此测试第二个队列处理的正确性。另外选取时间累积超过营业时间的样例以测试结束业务的过程是否正确执行。

4 算法的时空分析

1. 初始化和输入操作：初始化银行和顾客的操作是常数时间操作 $O(1)$。然后，根据银行的顾客人数 N_p 读取顾客信息，这部分的时间复杂度是 $O(N_p)$.

2. 主循环：`while(bank.clock < bank.closeTime)`循环的次数取决于银行的关闭时间、顾客的到达时间以及服务时间。在这个循环中，有几个关键操作：

   `EnQueue(bank.Queue1, customer[i])`：队列的入队操作是常数时间 $O(1)$。

   `bank.checkQueue1()`：这个方法可能调用 `checkQueue2`。`checkQueue2` 的时间复杂度是 $O(n)$，其中 n 是队列 `Queue2` 的长度。因此，`checkQueue1` 的时间复杂度也可能受到 `Queue2` 长度的影响。

3. 输出和平均等待时间计算：输出每个顾客的等待时间并计算平均等待时间的循环时间复杂度是 $O(N_p)$。

考虑以上各点，`main` 函数的总体时间复杂度依赖于几个关键因素：顾客的数量 N_p 和队列 `Queue2` 的最大长度。假设 `Queue2` 的最大长度为 m，则 `checkQueue1` 在最坏情况下的时间复杂度是 $O(m)$。由于 `checkQueue1` 在主循环中被调用，因此 `main` 函数的总体时间复杂度为 $O(N_p \times m)$.

5 测试结果及分析

测试数据如下所示：

```
4 10000 600 10
-2000 0
1000 10
-10000 30
2000 50
```

输出结果：

```
0
0
30
0
7
```

符合预期.

6 实验体会和收获

通过本实验，熟悉了关于队列的结构、操作、算法以及适用场景.