

离散事件-银行业务模拟 实验报告

PB21081601 张芷苒

1.需求及分析

1.1 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。

1.2 测试数据

一天营业开始时银行拥有的款额为10000(元)，营业时间为600(分钟)。其他模拟参量自定，注意测定两种极端的情况：一是两个到达事件之间的间隔时间很短，而客户的交易时间很长，另一个恰好相反，设置两个到达事件的间隔时间很长，而客户的交易时间很短。

1.3 实现提示

事件有两类：到达银行和离开银行。初始时银行现存资金总额为total。开始营业后的第一个事件是客户到达，营业时间从0到closetime。到达事件发生时随机地设置此客户的交易时间和距下一到达事件之间的时间间隔。每个客户要办理的款额也是随机确定的，用负值和正值分别表示第一类和第二类业务。变量total、closetime以及上述两个随机量的上下界均交互地从终端读入，作为模拟参数。

两个队列和一个事件表均要用动态存储结构实现。注意弄清应该在什么条件下设置离开事件，以及第二个队列用怎样的存储结构实现时可以获得较高的效率。注意：事件表是按时间顺序有序的。

1.4 设计任务

本程序是模拟银行存取款的流程。模拟客户在银行办理存取款的排队以及办理过程，并计算出客户总的逗留时间。

程序的设计是需要实现两个排队队列（Q1、Q2），一个事件队列（Q_en）。队列1（Q1）和队列2（Q2）用于模拟客户的排队情况，客户从队尾开始排队，从队头出队。事件队列（Q_en）用于记录客户的到达/离开事件，客户到达时记录到达事件，客户办理完存取款后离开时记录离开事件。到达事件记录客户的序列号、办理金额、到达时间；离开事件记录客户的序列号、办理金额、离开时间。

客户的到达时间是随机确定的（以客户最大到达时间arriveMaxTime为上界，客户最小到达时间arriveMinTime为下界）。客户若办理完存取款离开则离开时间由客户的到达时间加上交易时间（交易时间由dealMaxTime和dealMinTime随机确定）。客户的交易额由最大交易额上限（MaxMoney）随机确定，规定负数为取款，正数为存款。

开始时,客户到达银行进入队列1（Q1）排队同时记录客户的到达事件将事件存到事件队列（Q_en）。如果客户办理存款，则更新银行的存款（total_money），然后将客户结点从队列1（Q1）删除，同时事件表（Q_en）记录客户的离开事件。如果客户办理取款，当银行当前的金额（total_money）能够满足客户的取款需求，则更新银行的存款（total_money），然后将客户结点从队列1（Q1）删除，同时事件表（Q_en）记录客户的离开事件。当银行当前的金额（total_money）不能够满足客户的取款需求，则将客户移到队列2（Q2）进行等待。每当队列1（Q1）办理完一次存款操作的时候就顺序检查队列2（Q2），查找能够完成取款操作的客户，当队列2（Q2）当前客户完成取款则更新银行存款（total_money），然后将客户从队列2（Q2）中删除，同时事件表（Q_en）记录客户的离开事件，接着转到队列1（Q1）接待客户。如果队列2（Q2）中当前客户无法完成取款则将队客户移到队列2（Q2）的队尾继续排队，接着检查下一个客户。

当银行营业结束，检查队列1（Q1）和队列2（Q2）将还在排队的客户删除同时更新客户总逗留时间和接待客户总数。最后输出客户排队情况、接待客户总人数（counter）、需要存款的人数（n_p）、成功存款的人数（s_p）、需要取款的人数（n_g）、成功取款的人数（s_g）、客户逗留总时间（totalTime）、客户逗留平均时间（totalTime/ counter）以及银行现存金额（total_money）。

1.5 输入的形式和输入值的范围

银行模拟程序需要用户输入以下几个参量：

- 银行初始存款（total_money）
- 银行营业时间(closeTime)
- 客户最大到达时间间隔(arriveMaxTime)
- 客户最小到达时间间隔(arriveMinTime)
- 业务最大处理时间(dealMaxTime)
- 业务最小处理时间(dealMinTime)
- 最大交易额上限(MaxMoney)

参量输入要求用户从终端输入。

- total_money、closeTime、arriveMaxTime、dealMaxTime、MaxMoney、arriveMinTime、dealMinTime要求是一个大于零的整数
- 同时arriveMinTime不能大于 arriveMaxTime，dealMinTime不能大于dealMaxTime
- 以及arriveMinTime、arriveMaxTime、dealMinTime、dealMaxTime均不能大于closeTime

1.6 输出的形式

银行模拟程序通过dos界面输出银行营业过程的整个排队结果以及银行办理业务的结果。

输出：

- 需要存款的客户人数(n_p)
- 需要取款的客户人数(n_g)
- 成功存款的客户人数(s_p)
- 成功取款的客户人数(s_g)
- 客户逗留总时间(totalTime)
- 接待客户总数(counter)
- 客户逗留平均时间(totalTime/counter)
- 银行结余余额(total_money)

1.7 程序所能达到的功能

程序可以根据用户自定输入的合法参量（银行初始存款（total_money）、银行营业时间(closeTime)、客户最大到达时间间隔(arriveMaxTime)、客户最小到达时间间隔(arriveMinTime)、业务最大处理时间(dealMaxTime)、业务最小处理时间(dealMinTime)、最大交易额上限(MaxMoney)）来模拟银行的存取款业务。通过模拟输出结果包括客户排队情况、接待客户总人数（counter）、需要存款的人数（n_p）、成功存款的人数（s_p）、需要取款的人数（n_g）、成功取款的人数（s_g）、客户逗留总时间（totalTime）、客户逗留平均时（totalTime/ counter）以及银行现存金额（total_money）。

2. 重点代码解释

2.1 数据类型定义

2.1.1 结构体定义

```
/*客户/事件结构体*/
typedef struct event{
    int type;//事件类型到达或者离开 0为到达，1为离开
    int arriveTime;//到达时间
    int leaveTime;//离开时间
    int money;//存款数，正数为存款，复数为取款
    int num;//客户编号
```

```

    event* next;
}event,*eventLink;
/*队列结构体*/
typedef struct QNode{
    eventLink front;//队头指针
    eventLink rear; //队尾指针
}QNode,*Queue;

```

2.1.2 队列抽象数据类型的定义

```

ADT Queue{
    数据对象: D={ ai | ai∈event, i=1,2,...,n, n≥0 }
    数据关系: R1={ <ai-1, ai>|ai-1, ai∈D, i=2,...,n }
    基本操作:
        eventLink get_front(Queue &q);
            初始条件: 队列q存在
            操作结果: 返回队头结点指针
        void push(Queue &q,eventLink e);
            初始条件: 队列q存在, 结点指针e存在
            操作结果: 将e指向的结点插入到队列q的队尾
        void destoryQueue(Queue &q);
            初始条件: 队列q存在
            操作结果: 销毁队列q, 释放空间
        void pop(Queue &q);
            初始条件: 队列q存在
            操作结果: 删除对头结点并释放结点的空间
    }ADT Queue

```

2.1.3 全局变量的定义

```

int total_money;    //银行现存款
int closeTime;      //银行结束营业的时间
int arriveMaxTime;  //两个到达事件的时间间隔上限
int arriveMinTime;  //两个到达事件的时间间隔下限
int dealMaxTime;    //客户交易之间的时间上限
int dealMinTime;    //客户交易之间的时间下限
int MaxMoney;       //交易额上限
int currentTime=0;  //当前时间
int totalTime;      //客户逗留总时间
int counter=0;      //客户总数
int number=1;       //客户初始号码
int flag=1;         //判断是否有窗口在处理
int TimeOfDeal;     //交易时间
int MaxTime;        //到达时间
Queue Q_en;         //事件队列
Queue Q1;           //队列一

```

2.1.4 函数类型定义

```
/**
    结束函数
    当银行营业结束的时候调用此函数，清理释放申请的空间
*/
void closeBank();

/**
    到达函数
    当客户到达时调用此函数，随机生成客户信息以及相应的到达事件，分别
    加入队列1和事件队列。
*/
void arrive();

/**
    存款函数
    当客户办理存款的时候调用此函数
*/
void putMoney();

/**
    取款函数
    当客户办理取款的时候调用此函数
*/
void getMoney();

/**
    查找函数
    查找队列q中第一个可以满足取款的结点并返回，若是查找失败则返回
    NULL
*/
eventLink search(Queue &q,int m);

/**
    处理函数
    对通过【查找函数】找到的结点进行取款处理，生成相应的离开事件
*/
void findAndDeal();

/**
    初始化函数
    初始化三个队列，为三个队列分配空间
*/
void initQueue();
```

```
/**
    主函数
*/
int main();
```

2.2 主程序的流程

主程序首先接收用户的数据输入，并对用户输入的数据进行验证，如果输入的数据不合法则要求用户重新输入。

当用户输入的数据合法后开始主要程序的运行。首先调用initQueue()函数对队列进行初始化，以及根据客户到达时间的上下界随机生成第一个客户的到达时间。当当前时间（currentTime）小于银行结业时间（closeTime）时进行循环。每循环一次当前时间加1。

在每一次循环当中进行四次判断。

- 当交易时间（TimeOfDeal）小于当前时间（currentTime）的时候，让交易时间等于当前时间
- 当交易时间等于当前时间时，让flag为1表示当前有窗口在办理业务
- 当到达时间（MaxTime）等于当前时间（currentTime）的时候调用arrive()函数让客户到达加入队列1（Q1）排队并在事件表（Q_en）中加入到达事件，同时随机生成下一个客户到达时间（MaxTime）
- 当窗口处于交易状态（flag=1）且队列1（Q1）不为空。则进行客户交易处理，当客户是存款类型的时候调用putMoney()进行存款操作，同时调用findAndDeal()函数对队列2（Q2）进行检查是否有满足取款的客户并进行处理。当客户是取款类型的时候调用getMoney()进行取款处理

每一次循环当前时间（currentTime）加1，直到银行营业结束退出循环，输出模拟结果，最后调用closeBank()函数结束程序。

2.3 各模块的调用关系

程序分成主程序模块、存款模块、取款模块、检查处理模块、事件处理模块。主程序模块负责调用存款模块、取款模块和事件处理模块。存款模块调用事件处理模块、检查处理模块。取款模块和检查处理模块调用事件处理模块。

3.详细设计

3.1 数据类型的实现

3.1.1 队列抽象数据结构的实现

```
//入队操作
void push(Queue &q,eventLink e){
    if(e==NULL){//结点指针为空
        return;
    }
    if(q->front==NULL){//队列为空
        q->front=e;
        q->rear=e;
    }else{
        q->rear->next=e;
        q->rear=q->rear->next;
    }
}

//销毁队列
void destroyQueue(Queue &q){
    eventLink e,e1;
    e=q->front;
    if(q->front==q->rear==NULL){//队空
        return;
    }
    while(e!=NULL){
        e1=e->next;
        free(e);
        e=NULL;
        e=e1;
    }
}

//删除队首元素
void pop(Queue &q){
    eventLink e;
    e = q->front;
    if(q->front==NULL){//队列为空
        return;
    }else if(q->front->next==NULL){//队列只有一个元素
        q->front=q->rear=NULL;
    }else{
        q->front=q->front->next;
    }
    free(e);//释放空间
    e=NULL;
}

//返回队首元素
eventLink get_front(Queue &q){
```

```
        return q->front;
    }
}
```

3.1.2 函数类型的实现

```
/**
    结束函数
    当银行营业结束的时候调用此函数，清理释放申请的空间
*/
void closeBank(){

    //调用destoryQueue()函数销毁三个队列Q1、Q2、Q_en
    destoryQueue(Q1);
    destoryQueue(Q2);
    destoryQueue(Q_en);
}

/**
    到达函数
    当客户到达时调用此函数，随机生成客户信息以及相应的到达事件，
    分别加入队列1和事件队列。
*/
void arrive(){

    //声明两个eventLink结点指针e、e1
    //用malloc函数为两个结点指针分配空间

    if(e为空或者e1为空){
        //退出函数
    }

    //用(rand()%(2*MaxMoney)-MaxMoney)随机生成e和e1的办理金额
    //将当前时间（currentTime）作为e和e1的到达时间
    //将（number）作为e和e1的序列号，number++
    //将e1的类型设置为0（到达）

    //将e加入队列1（Q1）
    //将e1加入事件队列（Q_en）
}

/**
    存款函数
    当客户办理存款的时候调用此函数
*/
void putMoney(){

    //声明两个eventLink结点指针e、e1
```



```

//用malloc函数为结点指针e1分配空间

if(e1为空){
    //退出函数
}
//用get_front()函数获取队列1的队头元素
//更新银行金额total_money
//利用获得到的队头元素信息设置离开事件e1
//e1的离开时间为客户的到达时间+随机的处理时间
//e1加入事件队列
//删除队列1队头元素
//增加客户的总逗留时间（客户的离开时间-客户的到达时间）
//接待客户人数加1
//客户的离开时间为交易时间
//将窗口设置为空闲状态
}

/**
    取款函数
    当客户办理取款的时候调用此函数
*/
void getMoney(){
    //声明两个eventLink结点指针e、e1
    //用malloc函数为结点指针e1分配空间
    if(e1为空){
        //退出函数
    }
    //用get_front()函数获取队列1的队头元素
    if(客户的取款金额大于银行现存金额){
        //取款失败
        //e1为队列2客户元素
        //将客户从队列1删除，让客户进入队列2继续排队
    }else{
        //取款成功
        //更新银行现存金额
        //向事件队列添加相应的离开事件e1
        //将客户从队列1删除
        //接待客户人数加1
        //增加客户的总逗留时间（客户的离开时间-客户的到达时间）
        //客户的离开时间为交易时间
        //将窗口设置为空闲状态
    }
}

/**
    查找函数
    查找队列q中第一个可以满足取款的结点并返回，若是查找失败则返回
    NULL
*/

```

```

eventLink search(Queue &q,int m){
    //声明两个eventLink结点指针e、e1
    //取队列的队头元素e
    while(队头元素不为空){//遍历队列
        if(客户办理金额小于等于银行现存金额){
            if(队列只有一个元素){
                //队列头尾指针置空
                //返回客户指针e1
            }else{
                //队头指针移动一位
                //返回客户指针e1
            }
        }else{//资金不足，查找下一个客户
            if(队列只有一个元素){
                //不做操作
            }else{
                //将队头结点放到队尾
            }
        }
        if(队列遍历了一遍){
            //返回NULL
        }
    }
    //返回NULL
}

/**
    处理函数
    对通过【查找函数】找到的结点进行取款处理，生成相应的离开事件
*/

void findAndDeal(){
    //声明两个eventLink结点指针e、e1
    while(调用search函数查找客户结点e){
        //用malloc函数为结点指针e1分配空间
        //更新银行现存款total_money
        //离开事件e1的离开时间为当前时间+随机处理时间
        //设置离开事件e1加入事件表(Q_en)
        //接待客户人数增加
        //交易时间加上处理时间
        //增加客户总逗留时间（客户的离开时间-客户的到达时间）
        //释放结点e的空间
    }
    //将交易状态设为空闲状态
}

/**
    初始化函数
    初始化三个队列，为三个队列分配空间
*/

void initQueue(){

```

```

//为三个队列Q1、Q2、Q_en分配空间
if(Q1、Q2、Q_en其中一个为空){
    //退出函数
}
//将三个队列的头尾指针置空
}

```

3.1.3 主函数的实现

```

int main(){
    while（用户选择进入模拟程序）{
        //用户输入模拟数据
        //初始化三个队列
        //随机产生到达时间（MaxTime）用来确定第一个到达的客户
        while(当前时间<银行结业时间){
            //当前时间加1
            if(当交易时间<当前时间){
                //让交易时间等于当前时间
            }
            if(交易时间==当前时间){
                //让flag为1
            }
            if(到达时间==当前时间){
                //调用到达函数arrive()
                //随机生成下一个到达时间（MaxTime）
            }
            if(flag==1并且队列1不为空){
                if(队列1队头客户办理存款){
                    //调用存款函数putMoney()
                    //调用查找函数findAndDeal()
                }else{//办理取款
                    //调用取款函数getMoney()
                }
            }
        }
        //输出事件队列信息
        while(事件队列队头不为空){//遍历事件队列
            if(事件为离开类型){
                //输出事件信息
                if(事件为存款类型){
                    //成功存款人数加1
                }else{
                    //事件为取款类型
                    //成功取款人数加1
                }
            }else{
                //事件为到达类型
            }
        }
    }
}

```

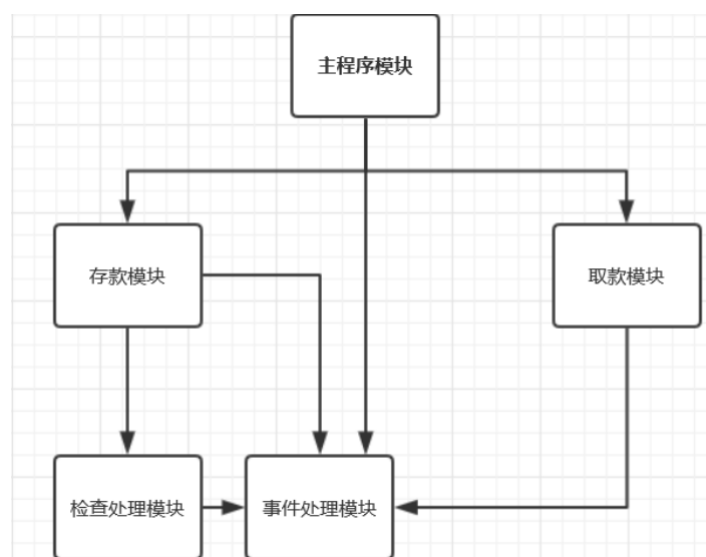
```

        //输出事件信息
        if(事件为存款类型){
            //需要存款人数加1
        }else{
            //事件为取款类型
            //需要存款人数加1
        }
    }
    //删除事件队列队头元素
}
//事件队列遍历结束
while(队列1队头元素不为空){
    //更新总逗留时间
    //接待客户人数加一
    //删除事件队列1队头元素
}
while(队列2队头元素不为空){
    //更新总逗留时间
    //接待客户人数加一
    //删除事件队列2队头元素
}
//调用结束函数closeBank()
//输出最终模拟营业结果
} //退出循环
} //main

```

3.2 程序模块调用关系图

程序分成主程序模块、存款模块、取款模块、检查处理模块、事件处理模块。



4. 算法的时空分析

下面给出程序中各个操作的时间复杂度和空间复杂度的分析。

4.1 get_front(Queue &q)//返回队首元素

由函数的操作可以看出此操作只需要返回队列头指针，所以时间复杂度为 $O(1)$ ，空间复杂度也为 $O(1)$ 。

4.2 push(Queue &q,eventLink e)//入队操作

由函数的操作可以看出来函数里没有循环语句只有简单的判断语句，没有额外的空间，只有指针的赋值操作。所以时间复杂度为 $O(1)$ ，空间复杂度也为 $O(1)$ 。

4.3 destoryQueue(Queue &q)// 销毁队列

在这个函数中存在循环用以遍历队列，所以这个操作的时间复杂度为 $O(n)$ ，函数没有额外的空间，所以空间复杂度为 $O(1)$ 。

4.4 pop(Queue &q)// 删除队首元素

通过分析可以知道函数的操作可以看出来函数里没有循环语句只有简单的判断语句，没有额外的空间，只有指针的操作。所以时间复杂度为 $O(1)$ ，空间复杂度也为 $O(1)$ 。

4.5 closeBank()//结束函数

函数只是调用了三个函数而调用的函数时间复杂度为 $O(n)$ ，所以函数时间复杂度为 $O(n)$ ，空间复杂度为 $O(1)$ 。

4.6 arrive();//到达函数

通过对函数操作的分析可以知道，这个函数与问题规模无关所以时间复杂度为 $O(1)$ ，空间复杂度也为 $O(1)$ 。

4.7 putMoney()//存款函数

通过对函数操作的分析可以知道，这个函数与问题规模无关所以时间复杂度为 $O(1)$ ，空间复杂度也为 $O(1)$ 。

4.8 getMoney()//取款函数

通过对函数操作的分析可以知道，这个函数与问题规模无关所以时间复杂度为 $O(1)$ ，空间复杂度也为 $O(1)$ 。

4.9 search(Queue &q,int m)// 搜索函数

通过分析可以知道函数里面存在一个循环用来遍历队列，所以这个操作的时间复杂度为 $O(n)$ ，函数没有额外的空间，所以空间复杂度为 $O(1)$ 。

4.10 findAndDeal()//处理函数

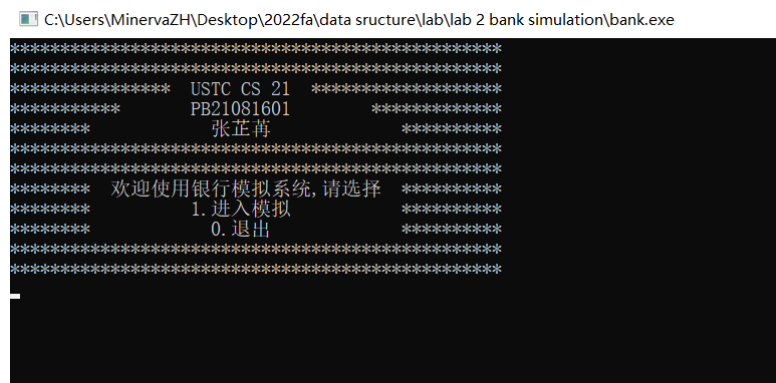
通过分析可以知道函数里面存在一个循环用来调用查找函数，所以这个操作的时间复杂度为 $O(n)$ ，函数没有额外的空间，所以空间复杂度为 $O(1)$ 。

4.11 initQueue()//队列初始化

通过对函数操作的分析可以知道，这个函数与问题规模无关所以时间复杂度为 $O(1)$ ，空间复杂度也为 $O(1)$ 。

5. 实际效果

下面是本程序运行的实际效果截图：



```
C:\Users\MinervaZH\Desktop\2022fa\data sructure\lab\lab 2 bank simulation\bank.exe
*****
*****  USTC CS 21  *****
*****  PB21081601  *****
*****    张芷苒    *****
*****
*****  欢迎使用银行模拟系统, 请选择  *****
*****    1. 进入模拟  *****
*****    0. 退出    *****
*****
*****
```

85	到达	547	-241
83	离开	551	-789
86	到达	550	-149
84	离开	553	243
85	离开	552	-241
86	离开	566	-149
87	到达	558	-750
88	到达	562	891
87	离开	561	-750
89	到达	567	-81
88	离开	566	891
89	离开	579	-81
90	到达	576	307
90	离开	584	307
91	到达	582	-676
91	离开	589	-676
92	到达	590	657
92	离开	609	657
93	到达	599	454
需要存款的客户人数: 41 (人)			
需要取款客户人数: 52 (人)			
成功存款的客户人数: 40 (人)			
成功取款客户人数: 52 (人)			
客户逗留总时间: 981 (分钟)			
接待客户总数: 93 (人)			
客户逗留平均时间: 10 (分钟)			
银行结余余额: 91691 (元)			
以上为银行模拟结果			
请按任意键继续. . .			