中国科学技术大学计算机学院

《数据结构》实验报告

实验题目：离散事件模拟（银行业务模拟）

学生姓名：李骋昊

学生学号：PB20081583

完成日期：10.28

目录

实验要求

◆2.6⑤ 银行业务模拟

【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。

【基本要求】

利用动态存储结构实现模拟，即利用C语言的动态分配函数 malloc 和 free。

【附加内容】

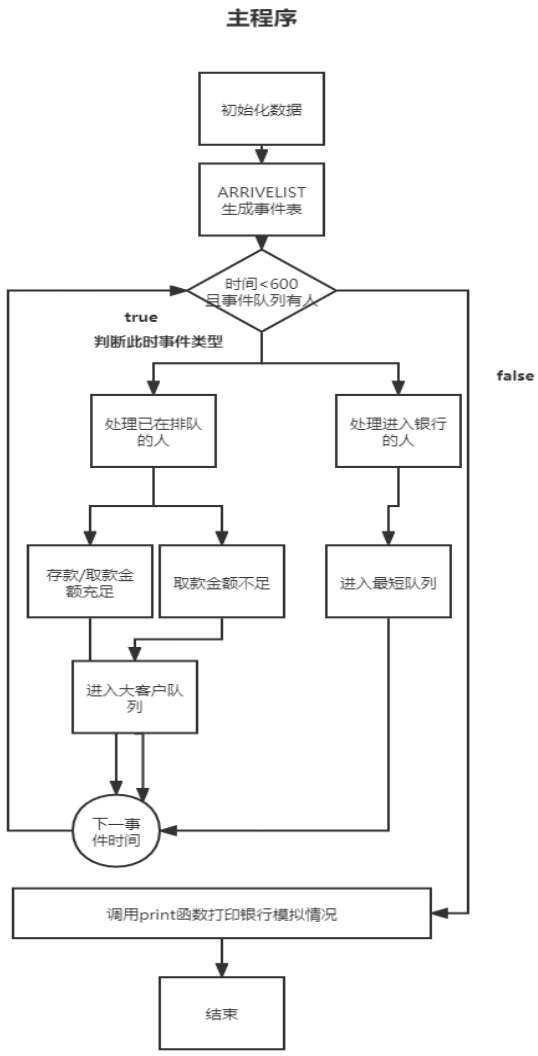
实现：多个第一种队列的服务窗口，通过输入数字确定数量。

设计思路

语言环境

因为课本的伪代码基本根据 C 语言书写，本次实验所用语言为 C11 标准下的 C 语言。

设计流程图



简单算法描述

1. 自定义银行队列数（小于 100）、最大最小存取款金额、交易间隔、交易时间。

2．生成客户到达时间表（通过结构体构造单链表）。初始化一般队列与第二类队列。

3.遍历各非空队列队首与客户到达时间表，选取到达时间最小的为下一事件。

4.判断事件是否符合循环要求，若符合，判断事件类型。若为到达时间表中事件，直接将事件写入历史事件表 history 中，链表头指针下移；若为存取款操作，判断能否完成业务，能则进行 solvelist 操作，该成员出队，不能则移入第二类用户队列。将完成的业务写入 history。返回 3.

5.打印 history 事件表

结构体说明

```
//队列中成员的状态结构
typedef struct Member {
    int status;//status为1时，代表已离开，为0代表正在排队
    int no;//用户编号
    int arrivetime, duration, endtime;
//arrive为到达时间,duration为办事所需时间，endtime为完成时间
    int waitingtime ;//等候时间
    int amount;//交易金额
    struct Member* next;
}member,*LinkMember;
```

```
//成员队列
typedef struct queue {
    int num;//队列中人数
    LinkMember front;//队首
    LinkMember rear;//队尾
}queue, *Queue;
```

代码讲解与分析

全局变量与声明

```
//设置全局状态参量
#define CLOSETIME 600 //闭店时间为600
#define initmoney 10000//初始银行账目有10000存留
```

对等候队列的操作函数

```
//预估时间
void TimePredict(Queue Q) {…

//判断空队
int EmptyLine(Queue L[]) {…

//初始化队列
Queue InitQueue() {…

//入队操作
void push(Queue q, LinkMember L) {…

//出队操作
void pop(Queue Q) {…
```

```
//出队操作
void pop(Queue Q) {…

//销毁操作，用于结束时清空存储空间
void destroy(Queue Q) {…

//寻找最小队
Queue MinQueue(Queue Q[]) {…

Queue mintime(Queue Q[]) {…

//选择各队列里下一时间发生的事件
Queue Timechoice(Queue L[],Queue Big)

//寻找可以解决的大额客户;
int search(Queue L) {…

//队伍成员复制
LinkMember Qcopy(LinkMember m) {…
```

对于 EmptyLine,InitQueue,push,pop,destroy 函数，这里采用与数据结构课本相类似的处理方法，故不多赘述。

TimePredict 函数

```
void TimePredict(Queue Q) {
    if (!Q)
        return;
    else if (Q->num&&Q->front->endtime==601)
        Q->front->endtime = TIME + Q->front->duration;
    return;
}
```

通过遍历每一条非空等候队列队首，预估结束时间为当前时间加上逗留时间。（每位成员初始结束时间全部预设为 601）

MinQueue 函数与它同类型的 MinTime、Timechoice 函数

```
//寻找最小队
Queue MinQueue(Queue Q[]) {
    Queue Min;
    int i;
    Min = Q[0];
    for (i = 0; i < MAXLINE; i++) {
        if (Q[i]->num == 0) {
            Min = Q[i];
            break;
        }
        if (Min->num > Q[i]->num)
            Min = Q[i];
    }
    return Min;
}
```

遍历每一条队列，访问队列指针指向的 num（队列人数）两两比较返回 num 最小的队列。

对于 MinTime，只是将 num 改为 front->endtime.

对于 Timechoice，则是将 MinTime 与第二类队列队首比较返回较小的队首。

这种方法时间复杂度为 O（n）.

Search 函数

```c
//寻找可以解决的大额客户;
int search(Queue L) {
    LinkMember p, q, r;
    p = q = L->front;
    while ((p->amount + MONEY) < 0) {
        L->rear->next = p;
        L->rear = p;
        r = p->next;
        p->next = NULL;
        p = r;
        if (p == q)
            return 0;
    }
    L->front = p;
    return 1;
}
```

将第二类队列的尾指针指向队首，当头指针所指成员无法完成业务时二者相应后移，直到出现可以完成交易的成员（返回 1）或完成循环一周（返回 0）.

时间复杂度为 O(n)。

Qcopy 函数

```
//队伍成员复制
LinkMember Qcopy(LinkMember m) {
    if (!m)
        return;
    LinkMember newM;
    newM = (LinkMember)malloc(sizeof(member));
    newM->status = m->status;
    newM->amount = m->amount;
    newM->arrivetime = m->arrivetime;
    newM->endtime = 601;
    newM->duration = m->duration;
    newM->no = m->no;
    return newM;
}
```

新建一个成员节点并将原节点数据复制。一般用于写入历史事件操作。

对成员表的操作函数

```
//以下为对成员表的操作

//取表尾操作
LinkMember GetRear(LinkMember L) {…

//成员表移动操作
LinkMember GoList(LinkMember L) {…

//复制操作，将某成员结构复制到链表的尾部
void copy(LinkMember m, LinkMember L) {…



//对于一般客户，我们用solvelist函数解决存取款问题
void solvelist(LinkMember L) {…
```

GetRear 函数

```
//取表尾操作
LinkMember GetRear(LinkMember L) {
    LinkMember p ;
    if (!L)
        return L;
    p = L;
    while (p->next)
        p = p->next;
    return p;
}
```

取一个指针 p 遍历链表 L 到末尾，返回 p。

时间复杂度为 O（n）。

GoList 函数

```
//成员表移动操作
LinkMember GoList(LinkMember L) {
    LinkMember p, q;
    p = L->next;
    q = L;
    q->next = NULL;
    free(q);
    return p;
}
```

简单的头指针后移。

Copy 函数

与 Qcopy 雷同，此处略过。

solvelist 函数

```
//对于一般客户，我们用solvelist函数解决存取款问题
void solvelist(LinkMember L) {
    L->status = 1;
    MONEY += L->amount;
}//作用效果是改写链表的最新项为完成
```

将成员状态标记为 1（离开银行），同时进行银行现有金额的更新。

事件生成函数 arrive

```
//arrive函数是建立事件开始链表的关键函数
LinkMember arrive() {
    LinkMember p,q,r;
    int temptime;
    p = (LinkMember)malloc(sizeof(member));
    p->arrivetime = rand() % (MAXINTER - MININTER + 1);
    p->endtime = 601;
    p->duration = rand() % (MAXPY - MINPY + 1) + MINPY;
    p->no = NO++;
    p->status = 0;
    p->amount = (rand() % (2 * MAXMONEY) - MAXMONEY);//利用rand函数随机记录交易金额
    p->next = NULL;
    q = r = p;
    TIME = p->arrivetime;//记录time的初始时间
```

先如上图所示部分初始化第一个成员节点：通过 rand 函数将到达时间 arrivetime，间隔时间 duration，交易金额 amount 确定，同时客户编号从 1 开始依次排列，成员 status 编辑为 0（到达），结束时间 endtime 为 601(不可能完成)，此时时间 TIME 初始化为首节点的 arrivetime。

```
    while (r->arrivetime+r->duration < CLOSETIME) {
        temptime = r->arrivetime;
        p = (LinkMember)malloc(sizeof(member));
        p->amount = (rand() % (2 * MAXMONEY) - MAXMONEY);//利用rand函数随机记录交易金额
        p->no = NO++;
        p->status = 0;
        p->arrivetime = rand() % (MAXINTER - MININTER +1) + temptime;
        p->duration= rand() % (MAXPY - MINPY + 1) + MINPY;
        p->next = NULL;
        r->next = p;
        r = r->next;
    }
    p->next = NULL;
    return q;
}
```

之后如法炮制进入 while 循环生成后续到达成员节点，直到到达时已无法执行业务（由于初始 TIME 只能是第一个成员到达的时间，故这里额外定义一个 temptime）。

最后返回首节点。（链队结构）

历史事件打印函数 printlist

```
void PrintList(LinkMember H) {
    int S_Cus=0,IP=0;//分别为完成业务用户，进门用户
    double TotalTime=0;//所有人的用时
    double AverageTime,SPercent;//平均时长
    LinkMember Htemp=H->next;
    while (Htemp) {
        if (!Htemp->status) {
            IP++;
            printf("时间：=    用户编号：=    进入银行开始排队。\n",
Htemp->arrivetime,Htemp->no);
        }
        else {
            S_Cus++;
            Htemp->waitingtime = Htemp->endtime - Htemp->arrivetime;
            TotalTime += Htemp->waitingtime;
            if (Htemp->amount > 0)
                printf("时间：=    用户编号：=    完成存（还）款金额为
M   等待时长为=    。\n", Htemp->endtime, Htemp->no, Htemp->amount,
Htemp->waitingtime);
            else
                printf("时间：=    用户编号：=    完成取（借）款金额为
M   等待时长为=    。\n", Htemp->endtime, Htemp->no, (-1)*Htemp->
amount,Htemp->waitingtime );
        }
        Htemp = Htemp->next;
    }
    AverageTime = TotalTime / S_Cus;
    SPercent =(double) S_Cus / (double)IP;
    printf("\n时间为=    银行下班！！！", CLOSETIME);
    printf("\n\n\n\n\n");
    printf("今日营业共计有%d名客户进店，其中%d名完成业务，完成率为%.5f
,平均用时为%2.3f，当日结余金额为%d\n", IP, S_Cus,SPercent,AverageTime,
MONEY );
}
```

遍历事件表，将 status 为 1 的成员累加记录为 S_Cus（代表他们走出银行）,将他们的 waitingtime 记录为（endtime-arrivetime）并累加进入 TotalTime。将事件表中每一位 status 为 0 的成员累加记录为 IP(代表他们进入银行)。

最后，根据上述数字进行运算得到平均时长与完成业务的比例。

时间复杂度为 O（n）。

主函数

函数的初始化

自定义最大队列数（malloc 实现），最大最小时间间隔，交易时长，交易额上限。

同时初始化历史事件表头指针为 History，尾指针为 pHistory，初始化所有队列与到达事件表。

While 循环

```
while ((ArriveList || !EmptyLine(QueueLine)||Bigname->num)&&TIME<
CLOSETIME) {
```

根据流程图所说判断能否循环

判断下一时间节点

```
//预估时间
        if (!ArriveList&&EmptyLine(QueueLine))
            if (Bigname->front->endtime > CLOSETIME||Bigname->front->
amount+MONEY<0||!MONEY)
                break;
        for (int i = 0; i < MAXLINE; i++)
            if(QueueLine[i]->num)
                TimePredict(QueueLine[i]);
        if (Bigname->num&&search(Bigname)) {
            TimePredict(Bigname);
            q = Timechoice(QueueLine, Bigname);
        }
```

遍历所有非空队首节点（对于第二类队列 Bigname，需要 search 返回 1），将它们的结束时间进行预先估计同时进行选择。

对该时间节点进行操作

一般情况

我们将能顺利进入历史事件表称之为一般情况，其情形如下图（以下一事件发生在到达事件表为例说明）。

```
//下一事件发生在到达事件表中
else {
    if (!ArriveList)
        break;
    TIME = ArriveList->arrivetime;
    if (TIME > CLOSETIME)
        break;
    copy(ArriveList, History);
    push(MinQueue(QueueLine), Qcopy(ArriveList));
    ArriveList=GoList(ArriveList);
}
```

首先更新时间并判断是否下班，将事件写入历史事件表中。

如果是到达则进入最短队列，到达事件表头指针后移。

如果是离开则更新银行金额以及成员状态（status 改为 1），同时出队。

出现资金不足

```
//是大客户
if (p->amount + MONEY < 0) {
    push(Bigname, Qcopy(p));
    pop(q);
}
```

将数据拷贝进第二类队列同时将客户从第一类队列中出队（本次操作不成功）

打印历史事件表，程序执行完毕

```
//打印结果
PrintList(History);

system("pause");

return 0;
```

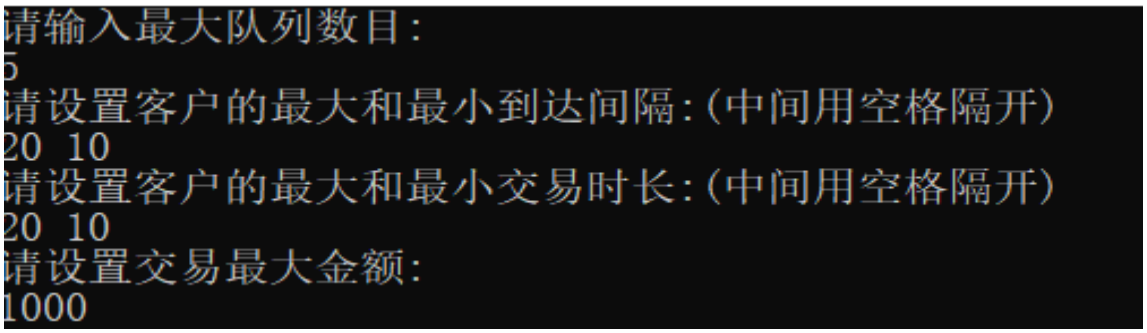
主函数的时空复杂度

由于每次事件发生都需要遍历所有成员判断事件，故时间复杂度为 O（n^2）,空间复杂度为 O（n）

代码测试与问题解决

一般条件下的代码测试

将交易金额设置为 1000，仅有 5 条队列，交易间隔与到达上下限为 20 和 10，测试结果如下。



执行成功。

较为复杂情况下代码测试

将交易金额设置为 10000，初始金额设置为 0，交易间隔与到达上下限为 2 和 1.，队列为 15。结果如下。

编程中问题解决

如何实现队列的动态设置。

这里（C 语言）有 2 种解决思路。一是定义一个足够大的数组（像本题一样），二是通过 malloc*n（n 是自定义队列长度）。由于两者实际上（在这题）差异不大，这里采用第二种方法。

如何实现时间的驱动

这里同样有两种方法。一是预先生成一个顺序事件表，通过顺序读取到达事件表来实现时间的变换。二是随机生成到达事件，同时与排序后的几条队列队首进行比较判断下一事件发生。

如何实现队列中的事件记录

可以在入队出队等事件发生时逐次打印，也可以和本题一样编写历史事件表一次性打印。

对于结束时间的初始化

初始化结束时间统一设置为 601，可以判断是否未执行时间预测函数，也可以作为打印时的依据，比 malloc 后的乱码更优越

实验总结

使用到的数据结构

各成员等候队列、到达事件表、历史事件表均为链队。

所以本次实验使用了单链表和队两种数据结构类型。

实验思考

大部分函数并没有得到优化，像 Qcopy 和 copy 这类函数具有一定的重合，导致代码尽管很简单却写了 400 余行，过于冗长。

由于使用纯 C 语言，C++的类无法运用，也是代码过长的原因之一。

通过做这次实验加深了对于课本前几章所学内容的印象与这几种基本数据结构类型的应用。同时通过对较长代码的 debug，也加深了我对养成良好编程习惯的深刻认识。需要进行适当的注释以及恰当的缩进，同时粗糙的复制粘贴可能是许多 bug 的成因。

附录（文件内容）

解决方案 banklist 文件夹

图片 picture 文件夹（包括上述所有实验）