3.5 离散事件模拟

假设某银行有4个窗口营业，从早晨银行开门起不断有客户进入银行。每个窗口每一时刻只能接待一个客户，对于刚进入银行的客户，如果某个窗口的业务员正空闲，则可上前办理业务；反之，若4个窗口均有客户所占，他会排在人数最少的队伍后面。现要编制一个程序以模拟银行的这种业务活动并计算一天客户在银行逗留的平均时间。

为了计算这个平均时间，我们需要掌握每个客户到达银行和离开银行这两个时间，后者减去前者即为每个客户在银行的逗留时间。所有客户逗留时间的总合被一天内进去银行的客户数除便是所求的平均时间。

称客户到达银行和离开银行这两个时刻发生的事情为‘事件’，则整个模拟程序将按事件发生的先后顺序进行处理，这样一种模拟程序称作事件驱动模拟。算法3.6描述的正是上述银行客户的离散事件驱动模拟程序。

```
void Bank_ Simulation (int CloseTime) {
      //银行业务模拟，统计一天内客户在银行逗留的平均时间
       OpenForDay ();                                    //初始化
     while (MoreEvent) {
       EventDrived (OccurTime, EventType); //事件驱动
       switch (EventType){
       case‘ A’ :CustomerArrived(); break; //处理客户到达事件
       case‘ D’ :CustomerDeparture(); break;
                                                  //处理客户到离开事件
       default: Invalid();
      }//switch
   }//while
   CloseForDay;  //计算平均逗留时间
}// Bank_ Simulation
```

算法 3.6

算法3.6处理的主要对象是‘事件’，事件的主要信息是事件类型和事件发生的时刻。处理的事件有两类：客户到达事件，客户离开事件。前一类事件发生的时刻随客户到来自然形成；后一类事件发生时刻则由客户事务所需时间和等待所耗时间而定。由于程序驱动是按事件发生时刻的先后顺序进行，则事件表应是有序表，其主要操作是插入和删除事件。

模拟程序中需要的另一种数据结构是表示客户排队的队列，队列中有关客户的主要信息是客户到达的时刻和客户办理业务所需时间。每个队列的队头客户即为正在窗口办理事务的客户，他办完事务离开队列的时刻就是即将发生的客户离开事件的时刻。因此，在任何时刻即将发生的事件只有下列5种可能：1）新的客户到达，2）--5）1-4号窗口客户离开。

在模拟程序中只需要两种数据类型：有序链表和队列。他们的定义如下：

```
typedef struct {
        int OccurTime;  //事件发生时刻
        int NType;   //事件类型，0表示到达事件，1至
            //4表示四个窗口的离开事件
}Event, ElemType;     //事件类型，有序链表LinkList
                //的数据元素类型

Typedef LinkList EventList

                              //事件链表类型，定义为有序链表
typedef struct {
        int ArrivalTime;            //到达时刻
        int Duration;                //办理事务所需时间
}QElemType;                         //队列的数据元素类型
```

现在分析算法3.6中两个主要操作步骤是如何实现的？

先看对新客户到达事件的处理。不失一般性，假设第一个客户进门时刻为0，即是模拟程序处理的第一个事件，之后每个客户的到达时刻由前一个客户到达时设定。因此在客户到达事件发生时需先产生两个随机数：其一为到达客户办理事务所需时间durtime，其二为下一客户将到达的时间间隔intertime，假设当前事件发生的时刻为occurtime，则下一客户到达事件发生的时刻为occurtime+intertime，由此应产生一个新的客户到达事件插入事件表；刚到达的客户应插入到当前所含元素最少的队列中；若该队列在插入前为空，则还应产生一个客户离开事件插入事件表。

客户离开事件的处理，首先计算该客户在银行的逗留时间，然后从队列中删除该客户后察看队列是否空，若不空则设定一个新的队头客户离开事件。

```
        //程序中用到的主要变量//

EventList       ev;                    //事件表
Event           en;                    //事件
LinkQueue       q[4];                  //4个客户队列
QElemType       customer;              //客户记录
int    TotalTime, CustomerNum;
                                       //累计客户逗留时间，客户数
int com (Event a, Event b);
   //依事件a的发生时刻<或=或>事件b的发生时刻分
//别返回-1或0或1
```

```
void OpenForDay() {       //初始化操作
    TotalTime=0;   CustomerNum=0;
                    //初始化累计时间和客户数为0
    InitList (ev);      //初始化事件链表为空表
    en.OccurTime=0;  en.NType=0;
                        //设定第一个客户到达事件
    OrderInsert (ev, en, cmp);
                        //插入事件表
    for (i=1; i<=4; ++i)   InitQueue (q[i]);
                        //置空队列
}//OpenForDay
```

```
void CustomerArrived() {
  //处理客户到达事件，en.NType=0。
  ++CustomerNum;
  Random (durtime, intertime);  //生成随机数
  t=en.OccurTime+ intertime; //下一客户到达时刻
  if (t<CloseTime)        //银行尚未关门，插入事件表
    OrderInsert (ev, (t, 0), cmp);
  i=Minimum (q);                    //求长度最短队列
  EnQueue (q[i], (en.OccurTime, durtime));
  if(QueueLength (q[i])==1)
      OrderInsert (ev, (en.OccurTime + durtime, i), cmp);
              //设定第i队列的一个离开事件并插入事件表
}// CustomerArrived
```

```
void CustomerDeparture()  {
    //处理客户离开事件，en.NType>0。
    i=en.NType;  DelQueue (q[i], customer);          //
   删除第i队列的排头客户
    TotalTime+=en.OccurTime-
   customer.Arrivaltime; //累计客户逗留时间
    if (!QueueEmpty(q[i])) //设定第i队列的一个离开事
   件并插入事件表
        GetHead (q[i], customer);
        OrderInsert (ev, (en.OccurTime +
   custom.Duration, i), (*cmp)());
}// CustomerDeparture
```

```
void Bank_ Simulation (int CloseTime) {
        OpenForDay();
        while (! ListEmpty(ev) ) {    //初始化
           DelFirst (GetHead (ev), p);
           en=GetCurElem(p);
           if(en.NType==0)
                CustomerArrived();
                                      //处理客户到达事件
           else CustomerDeparture() ;
                                      //处理客户离开事件
    }                 //计算平均逗留时间
  printf( “The Average Time is %f\n”,
     (float)TotalTime/CustomerNum);
}// Bank_ Simulation
```

算法 3.7

本章学习要点

1. 掌握栈和队列类型的特点，并能在相应的应用问题中正确选用它们。

2. 熟练掌握栈类型的两种实现方法，特别应注意栈满和栈空的条件以及它们的描述方法。

3. 熟练掌握循环队列和链队列的基本操作实现算法，特别注意队满和队空的描述方法。

4. 理解递归算法执行过程中栈的状态变化过程。

本章实习题目：

一、迷宫问题

问题描述：

以一个m*n的长方阵表示迷宫，0和1分别表示迷宫中的通路和障碍。设计一个程序，对任意设定的迷宫，求出一条从入口到出口的通路，或得出没有通路的结论。**要求：**

实现一个以链表作存储结构的栈类型，然后编写一个求解迷宫的非递归程序。求得的通路以三元组(i,j,d)的形式输出，其中：(i,j)指示迷宫中的一个坐标，d表示走到下一坐标的方向（东、南、西、北）。

实现提示：

可采用‘穷举求解’方法，即从入口出发，顺着某一方向进行探索，若能走通，则继续往前进；否则沿着原路退回，换一个方向继续探索，直至出口位置，求得一条通路。假如所有可能的通路都探索到而未能到达出口，则所设定的迷宫没有通路。

可以二维数组存储迷宫数据，通常设定入口点的下标为(1,1)，出口点的下标为(m,n)。为处理方便，可在迷宫四周加一圈障碍。对于迷宫中任一位置均可约定东、南、西、北四个方向可通。

二、八皇后问题

问题描述：

以一个8*8的方阵表示棋盘，把8个皇后放在棋盘上，使得任何一个都不将其他七个的军。即同一行、同一列、同一对角线只能有一个皇后。

如何表示行、列及对角线上有无皇后？可用如下数组表示：

x[8]: x[i]表示第i列上皇后的位置（行号）；

a[8]: a[j]为true表示第j行上无皇后；

b[15]: b[k]为true表示第k条/对角线上无皇后（2<=k<=16)；

(因为在/对角线上，每个格子的坐标i和j之和是相同的）

c[15]: c[k]为true表示第k条\对角线上无皇后（ -7<=k<=7)；

(同样在\对角线上，每个格子的坐标差i-j是常数）

注意：此处，i代表列号，j代表行号

我们可以得到下面初步解法：

```
void try (int i) {
    准备选择第i个皇后的位置
    do { 选择下一个；
            if  安全
                { 置放皇后；
                  if  i<8 { try(i+1 ) ；
                                if 不成功  移去皇后
                                }
                 }
          }while 成功||位置试完
  }
```