离散事件模拟

大数据学院1班 和泳毅 PB19010450

实验要求

数据结构部分

实现操作

在**queue.h** 和**priority_queue.h**中需要填写这两个数据结构的操作。

1. **init**函数对数据结构进行初始化。
2. **destroy**函数负责清除分配堆内存的空间。
3. **enqueue**函数会将一个元素加入到队列中。如果队列已满，返回false；否则，返回true。
4. **dequeue**函数将队头元素从队列中取出。如果队列中没有数据，返回false；否则，返回true。
5. **top**函数会访问队列头部中的元素，但不会改变队列。如果队列中没有数据，返回false；否则，返回true。
6. **length**函数返回队列中元素的个数。
7. **full**和**empty**函数分别判断队列是否为满或者为空。

实现细节

1. queue 和 priority queue 最大的不同点在于：queue 要求先进入队列的元素最先出来，而 priority queue 要求把在队列数据中最小的元素放入队头，让其成为最先出来的元素。
2. 两个数据结构中的 data 域是存放数据的数组， capacity 域表示队列能容纳的最大元素个数。由于队空和队满之间存在 capacity + 1 种状态，data 也需要分配 capacity + 1 的容量以区分。
3. queue 的 front 指向队头元素，而 rear 指向下一个元素插入的位置。
4. priority queue 的实现：数据只能存放在 data[1] 和 data[rear] 之间，data[1] 一定是优先队列中元素最小的，且满足 data[i/2] <= data[i] (i > 1)。元素入队时，放入队列末尾，判断是否满足 data[i/2] <= data[i]。若满足可返回；若不满足则交换 data[i/2] 和 data[i] 的位置，继续判断交换直到满足条件。例如，在原来有4个元素的优先队列中，插入比原队列元素都要小的元素，则会发生交换 data[5] <-> data[2], data[2] <-> data[1]。元素出队时，取出对头元素，将队尾元素放入队头位置 data[1]，从 data[1] 处开始判断 data[i/2] <= data[i] 这一条件。（注意：满足 i/2 = j 的 i 有 2j 和 2j+1， 并判断是否越界，操作的时间复杂度应该为 O(lgn) ）
5. 觉得上面实现有困难的。可以用遍历找出最小元素放入队头。（时间复杂度是线性的，但测试不卡时间）
6. 为了通用性，priority queue 需要比较元素大小的函数指针 cmp。cmp(a, b) >= 0 表示 a >= b。

队列模拟部分

总体需求

银行共有 num_servers 个柜台。顾客达到银行后，如果柜台都被占满，就会按它们的进入时间排成一个队列，先到先服务。如果排队中的人数超过 queue_capacity，后来的顾客就不进入银行办理业务了。

我们已随机生成了达到银行的顾客的到达时间和业务办理时间。你需要在**simulation.cc**中填写相关操作。最终得到**办理过业务总人数 （不包括没有进入银行的顾客）**，和他们在**排队等待的总时间 （不包括办理业务的时间）**。返回排队等待的平均时间。

实现细节

Customer中分别有3个数据，分别是：arrive_time，service_time, leave_time。它们分别表示客户到达银行的时间点，客户办理业务需要的时间，客户最终离开的时间点。

simulate函数中有3个队列数据结构，都存储Customer类型，分别是：arrive_flow、queue、leave_flow。

arrive_flow已生成好并作为simulate函数的参数，已经**根据进入银行的时间做了从小到大的排序**，且arrive_time，service_time已经根据指数分布随机生成。

queue是客户在银行内排队等待的一个队列。如果所有的服务台都被占用，就会有客户在等待。如果等待人数超过队列容量queue_capacity，新来的客户无法排队，直接离开。

leave_flow是所有正在办理业务的客户的优先队列，根据客户离开时间进行排序。容量大小就是服务台数量num_servers。你需要自己**设定好leave_time和Customer在优先队列中的比较函数。**

从客户视角上看。客户在arrive_time进入银行（arrive_flow 出队）。如果银行排队的人太多了，就直接走了；否则加入排队（queue 入队）。如果有一个服务台没有人办业务，自己在队头，就能结束排队（queue 出队），用service_time的时间办业务（leave_flow 入队）。到了leave_time后就离开银行 （leave_flow 出队）。

从上帝视角上看。只会发生3种事件，其中只有2种事件会改变系统，一个是**客户进入银行**，另一个是**客户离开银行**。也只有在这两个事件发生后，**排队等待中的客户开始办理业务**的事情才会发生。因此~~作为膜法师~~，只需要~~操控时间~~，选择做最近的这两类事件，然后判断能不能做第三种事件。

设计思路

数据结构部分

queue.h按照课本对循环队列的建议来设计，使空间为capacity+1(容量为capacity)的链中只存储capacity个结点以区分队满与队空。

1. **init**：先分配空间，指针初始化，容量赋值。
2. **destroy**：释放空间，指针初始化，容量清零。
3. **enqueue**：先检查队满，在队尾插入元素，尾指针增一。
4. **dequeue**：先检查队空，保存队头元素，头指针增一。
5. **length**：先检查队空，头尾指针之差对总空间取余。
6. **full**：尾指针增一对总空间取余与头指针比较。
7. **empty**：头尾指针比较。

priority_queue.h主要不同点为插入与删除，对比小顶堆概念每次把最小的结点放在队头。

1. **enqueue**：先检查队满，在队尾插入元素，尾指针增一。令i=q.rear,让q.data[i]与q.data[i/2]比较使得队头元素最小，注意边界。
2. **dequeue**：先检查队空，保存队头元素，队头元素与队尾元素交换，尾指针减一。令i=1,让q.data[i]与q.data[i*2]比较使得队头元素最小，注意边界。

队列模拟部分

simulation.cc注意到达流，队伍，离开流的关系。离开流是优先队列，队头元素到时间最先离开，而当离开流不满时，队伍出元素加入离开流。到达流已有序，当队伍不满时，到达流出元素加入队伍。

离开时刻=出队时刻+服务时间。排队时间=出队时刻-到达时刻。区分时间与时刻。

关键代码讲解

分析本实验核心为队列模拟部分代码，采用代码+注释的方式讲解：

```
double simulate(const size_t queue_capacity, const size_t num_servers, Queue<Customer>&
arrival_flow)
{
    Queue<Customer> queue;
    Priority_Queue<Customer> leave_flow;

    int num_customers = 0;
    double total_queue_time = 0;

    Queue<Customer>::init(queue, queue_capacity);
    Priority_Queue<Customer>::init(leave_flow, num_servers, customer_compare);//初始化

    while (!Queue<Customer>::empty(arrival_flow) || !Queue<Customer>::empty(queue) ||
!Priority_Queue<Customer>::empty(leave_flow))
        //三队不都空
    {
        double current_time = 0;
        Customer customer_to_arrive;
        Customer customer_to_leave;
        Queue<Customer>::top(arrival_flow, customer_to_arrive);//到达流队头
        Priority_Queue<Customer>::top(leave_flow, customer_to_leave);//离开流队头
        if (!Queue<Customer>::empty(arrival_flow) &&
(Priority_Queue<Customer>::empty(leave_flow) || customer_to_arrive.arrive_time <
customer_to_leave.leave_time))
            //到达流非空 且 离开流空或到达时刻<离开时刻
        {
            Queue<Customer>::dequeue(arrival_flow, customer_to_arrive);
            if (!Queue<Customer>::full(queue))//队伍未满则排队
                Queue<Customer>::enqueue(queue, customer_to_arrive);
            current_time = customer_to_arrive.arrive_time;//更新时刻
        }
        else if (!Priority_Queue<Customer>::empty(leave_flow) &&
(Queue<Customer>::empty(arrival_flow) || customer_to_leave.leave_time <=
customer_to_arrive.arrive_time))
        {
            Priority_Queue<Customer>::dequeue(leave_flow, customer_to_leave);//离开流队头离开
            current_time = customer_to_leave.leave_time;//更新时刻
        }
        if (!Queue<Customer>::empty(queue) && !Priority_Queue<Customer>::full(leave_flow))
            //队伍非空 且 到达流未满
        {
            Customer customer;
            Queue<Customer>::dequeue(queue, customer);
            customer.leave_time = current_time + customer.service_time;//离开时刻
            Priority_Queue<Customer>::enqueue(leave_flow, customer);//队伍出元素入离开流
            double queue_time = current_time - customer.arrive_time;//一人排队时间
            num_customers++;
            total_queue_time += queue_time;//总排队时间
        }
    }

    Queue<Customer>::destroy(arrival_flow);
    Queue<Customer>::destroy(queue);
    Priority_Queue<Customer>::destroy(leave_flow);

    return total_queue_time / num_customers;
}

```

调试分析

需求分析

得到模拟银行业务排队的人均时间，查看是否在置信区间内。

时空分析

队列操作都是基本操作，队伍模拟时间复杂度为O(n)。

代码测试

```
ubuntu@VM2118-Yorick:/home/ubuntu/desktop/ds-lab/queue$ ./run.sh
===============================================================================
All tests passed (286 assertions in 2 test cases)

===============================================================================
All tests passed (329 assertions in 2 test cases)

==20849== Memcheck, a memory error detector
==20849== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==20849== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==20849== Command: ./build/test_memory
==20849==
Expectation: 0.49143
Simulation: [0.463622, 0.556841]
==20849==
==20849== HEAP SUMMARY:
==20849==     in use at exit: 0 bytes in 0 blocks
==20849==   total heap usage: 10 allocs, 10 frees, 98,472 bytes allocated
==20849==
==20849== All heap blocks were freed -- no leaks are possible
==20849==
==20849== For lists of detected and suppressed errors, rerun with: -s
==20849== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
Expectation: 0.338462
Simulation: [0.33545, 0.339548]
Expectation: 0.0325339
Simulation: [0.0321081, 0.0327626]
Expectation: 0.0402575
Simulation: [0.0357406, 0.0406599]
===============================================================================
All tests passed (3 assertions in 3 test cases)
```

实验总结

对Linux虚拟机的相关操作有了一定认识，对c++有了一定的了解。同时发现优先队列可以在找最小值时提高性能。