《数据结构与算法》

实验 2 实验报告

班级：信安 2302

学号：202308060227

姓名：石云博

目录

1 问题分析

1.1 处理的对象

要处理的对象是一组线性表的数据，根据题目可以使用队列的方式去存储。

1.2 实现的功能

要实现的功能是能根据当前题目的 now 值去取出当前列的对头元素，并且能够自由地在队尾进行元素的插入。因为要比较 now 和当前的对头元素的值，所以要能够查找到当前列的对头元素的大小。插入的规则是找到比新元素小的最大的值，所以还要能够查找每一列的队尾元素的值的大小。最后输出存在的队列的个数。

1.3 处理后的结果

处理后，所有的元素都存到了对应的队列里面，能直接看出存在的队列数。

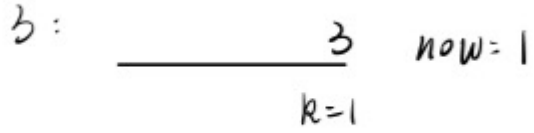
1.4 样例求解过程

样例：5 8 1 7 4 2 9 6 3

求解过程:

设定 now 为 1，使用变量 k 来记录开的队列的数量。然后读入样例，倒叙取出每个数来解题。

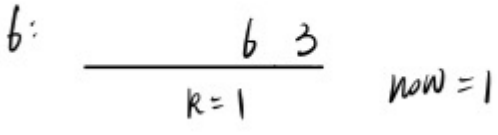
第一个数为 3，不等于当前的 now 值，所以建立新的队列来存放 3. K 的值加 1.



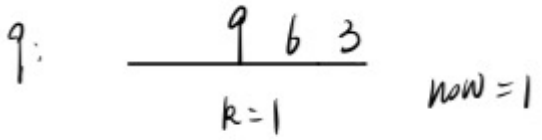
第二个数是 6，经过检索发现其大于当前的 now 值的同时 3 是当前所有队列中队尾元素比 6 小的最大元素，所以直接插入到 3 所在队列的队尾。然后遍历每列的对头元素，检查其是否等于当前的 now 值，没有等于的，所以跳过这个步骤。

这里的队尾元素检索是循环遍历当前场上的所有队列的队尾元素，使用一个临时变量 res 来储存元素大小，j 来储存列的索引，如果队尾元素大于当前元素，则取队尾元素和 res 的最大值来更新 res，同时根据比较结果更新 j 储存当前队列的索引以方便进行新元素的插入工作。

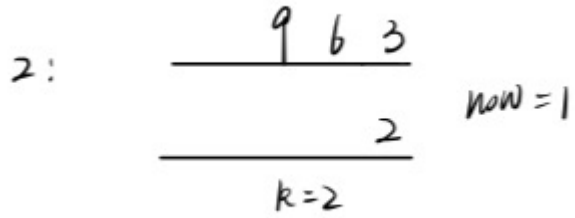
对头元素的检索则是直接循环遍历每一列的对头元素的值和当前的 now 值相比较。



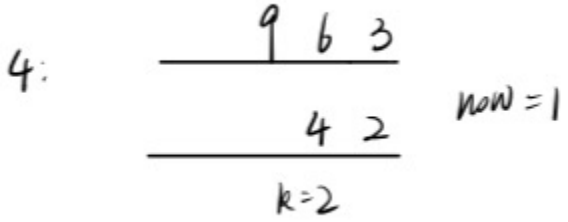
第三个数是 9，经过检索发现其大于当前的 now 值的同时 6 是当前所有队列中队尾元素比 6 小的最大元素，所以直接插入到 6 所在队列的队尾。对头元素中没有等于 now 值的。



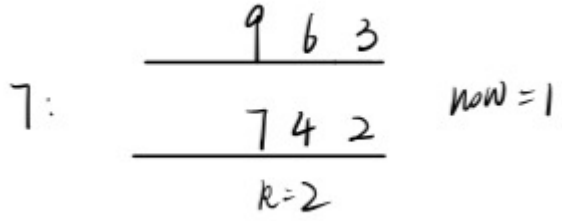
第四个数是 2，大于当前的 now 值，但是其小于当前有的所有列的队尾元素，所以需要给 2 重新开一个队列并且让 2 入队。K 的值加 1.



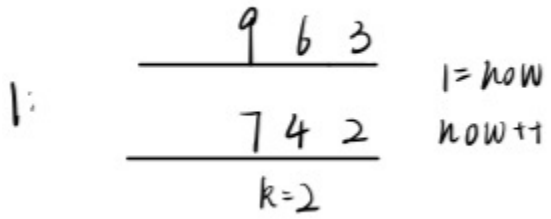
第五个数是 4，大于当前的 now 值，检索当前有的所有队列的队尾元素，发现 2 所在的队列是队尾元素小于 4 的最大元素，所以将 4 插入到 2 所在队列的队尾。



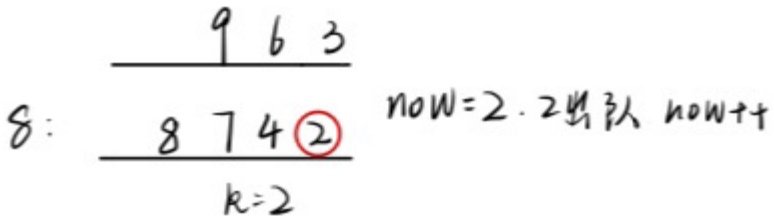
第 6 个数是 7，大于当前的 now 值检索当前有的所有队列的队尾元素，发现 4 所在的队列是队尾元素小于 7 的最大元素，所以将 7 插入到 4 所在队列的队尾。



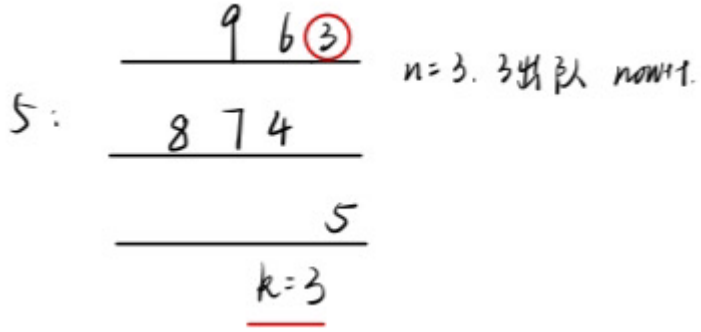
第 7 个数是 1，等于当前的 now 值，所以不需要重新分配队列，同时 now 的值加 1.



第 8 个数是 8，大于当前的 now 值，遍历每一列的队尾元素，找到小于 8 的最大队尾元素，是 7，则将 8 插入到 7 所在队列的队尾。遍历每个队列的对头元素，发现有一列的对头元素是 2，等于当前的 now 值，则 2 出队，同时 now 的大小加 1.



第 9 个数是 5，大于当前的 now 值，检索每列的队尾元素，发现没有队尾元素小于 5 的，所以开一个新的队列来存放 5. K 的值加 1.



至此所有的数遍历完了，可以输出 k，即本题的答案了，样例中的 k 值为 3.

2 数据结构与算法分析

2.1 抽象数据类型设计

使用的抽象数据类型是线性表中的队列。

数据对象：D = { ai|ai ∈ 集合中元素， i=1,2,3……,n ,n>=0}

数据关系：R1={|ai-1 ,ai ∈D,i=2,……,n}

具体实现：ADT 的定义和声明写在项目中的 BaseList.h 中

接下来是一些函数：包括构造函数，析构函数，以及一些操作函数（对头弹出元素，查找对头队尾元素的值等等）。

Public:

```
virtual ~BaseList() {} {} // Base destructor 析构函数
virtual void push(const E& item)=0; // 在队尾插入元素
virtual void pop_front()=0; // 弹出对头元素
virtual E front()=0; // 返回对头元素的值
virtual bool empty()=0; // 检查队列是否为空
```

virtual E back()=0; // 返回队尾元素的值。

virtual E length ()=0; // 返回队列的元素个数。

2.2 物理数据对象设计

ADT 的物理实现设计通过顺序表的存储结构来实现的，私有部分是一些参数，公有部分是一些操作函数的具体实现，各个函数的函数原型和功能如下：

private：

int maxsize;

int size;

E* listArray;

Public：

List(int s) {} ； // 构造函数

~List() { delete [] listArray;} // 析构函数

void push(const E& it) {} // 在队列的队尾插入元素

E back() {} // 返回队尾元素的值

bool empty() {} // 判断队列是否为空

void pop_front() {} // 弹出对头元素

E front() {} // 返回对头元素的值

int length() {}； // 返回队列的长度

2.3 算法设计的思想

根据题目描述和样例求解过程，我们需要找到针对每一组数据的最小的需要用到的队列数。想找出这个最小值，我们可以模拟题目的要求，每次面对一个新的元素，首先判断其是否等于当前的 now 的值，不等于的情况去找到符合要求的能让该元素入队的队列，没有符合要求的队列则开一个新的队列以存放该元素。期间注意每次新元素进来时要检查当前存在的队列的对头元素是否等于 now 值，如果等于则要让 now 值加 1。所有数据都处理完之后得到的 k 就是答案。

2.4 关键功能算法步骤

怎样将元素插入的其对应符合要求的队列：

1 找小于当前元素的最大元素，使用临时变量 res 存遍历过程中符合要求的队尾元素的最大值，j 存队列的索引，这样就能在遍历完后直接将元素插入到 j 列的队尾了。

2 利用 push 函数将新的元素入队。

3 使用 back()函数返回队尾值来比较是否能入队，用 front()函数返回对头值来判断其是否需要出队，now 的值是否需要加 1。

ADT 部分关键函数的实现：

1，push（）函数：ListArray[size++]=it;但是需要考虑到数组的 size 会不会溢出其 maxsize，当 size==maxsize 时，想往里插入元素则需要对数组进行扩容，首先将 maxsize*2,然后以现在的 maxsize 创建一

个新的数组并使用循环将旧数组的元素都复制到新数组中，然后使用新数组代替旧数组

2 back()函数 ：return listArray[size - 1];

3 front()函数：return listArray[0];

4 pop_front()函数：从 1 到 size-2 位，循环让数组的元素往前的移位，然后让 size-1;

5 empty()函数：return size==0;

3 算法性能分析

3.1 时间复杂度

这个程序的主要部分是一个循环，它遍历输入的每一个数字。在这个循环中，它执行以下操作：

检查当前的数字是否等于 now。如果是，它就增加 now 的值并继续下一个数字。这个操作的时间复杂度是

O(1)。

如果当前的数字不等于 now，它就会遍历所有的 List 对象来找到一个合适的位置插入当前的数字。这个操作的时间复杂度是

O(n)，其中

n 是 List 对象的数量。

在找到合适的位置后，它会将当前的数字插入到 List 对象中。这个操作的时间复杂度是 O(1)。

因此，对于每一个输入的数字，它需要执行 O(n)的操作。由于这个过程会对每一个输入的数字重复，所以总的时间复杂度是 O(n^2)

3.2 空间复杂度

这段代码的空间复杂度主要取决于以下几个部分：

1. List<int>* l[N];：这里创建了一个指针数组，数组的大小为常数 N，所以这部分的空间复杂度为 O(N)。
2. int res[N];：这里创建了一个整型数组，数组的大小也为常数 N，所以这部分的空间复杂度也为 O(N)。
3. for (int i=0;i<N;i++) { l[i]=new List<int>(N); }：这里为每个指针分配了一个大小为 N 的列表，所以这部分的空间复杂度为 O(N2)。

综上，这段代码的总空间复杂度为

O(N^2)