## 一、问题分析

### 1.1 处理的对象（数据）

处理的数据是车厢的编号，int 类型。具体需要处理的对象为：入轨道车厢的一个排列，由题意，最后输入的最先输出，符合后入先出的特点，适宜用数据结构栈储存；缓冲轨道，由题意，先进缓冲轨道的先出轨道，符合先入先出的特点，适宜用数据结构队列储存。

### 1.2 实现的功能

按车厢编号升序的顺序将车厢出轨道。将不满足出队顺序的入轨道车厢进入缓冲区，按照如下的原则来选择缓冲轨道： 如果缓冲轨道上已有的尾部车厢编号均小于当前编号；且有多个缓冲轨道都满足这一条件，则选择左端车厢编号最大的缓冲轨道；否则选择一个空的缓冲轨道。计算缓冲轨道的数量。

### 1.3 结果显示

输出缓冲轨道的数量。

### 1.4 样例求解过程

5 8 1 7 4 2 9 6 3

1.读入入轨道车厢排列：5 8 1 7 4 2 9 6 3，并将顺序存入栈中

2.nowOut=1

栈顶元素 3 出栈，无缓冲区，新建缓冲区 H1，加入缓冲区 H1：NULL->3

栈顶元素 6 出栈，加入缓冲区 H1：3->3,6

栈顶元素 9 出栈，加入缓冲区 H1：3,6->3,6,9

栈顶元素 2 出栈,无缓冲区末尾元素小于 2,新建缓冲区 H2,加入缓冲区 H2:NULL->2

栈顶元素 4 出栈，加入缓冲区 H2：2->2,4

栈顶元素 7 出栈，加入缓冲区 H2：2,4->2,4,7

栈顶元素 1 出栈，等于 nowOut，（加入缓冲区 Hk 后立即出队），nowOut++

3.nowOut=2

H2 队首元素 2 出队，H2：2,4,7->4,7，nowOut++

4.nowOut=3

H1 队首元素 3 出队，H1：3,6,9->6,9，nowOut++

5.nowOut=4

H2 队首元素 4 出队，H2：4,7->7，nowOut++

6.nowOut=5

栈顶元素 8 出栈，加入缓冲区 H2：7->7,8，nowOut++

栈顶元素 1 出栈，等于 nowOut，（加入缓冲区 Hk 后立即出队），nowOut++

7.输入车厢已全部进入缓冲区，2 条不能直接出轨道的缓冲轨道，1 条直接出轨道的缓冲轨道，输出的总缓冲轨道数量为 2+1=3。

## 二、数据结构和算法设计

**列车入轨道是先入后出，用栈实现，缓冲轨道是先入先出，用队列实现。多个队列可用线性表储存，由于不是问题的主体，直接用 STL 中的 vector 作为动态数组储存。其余栈和队列基于教材上 ADT 进行稍微修改以适应本题。**

### 1.抽象数据类型设计

```
template<typename E>class QueueADT
{
    private:
```

```
        void operator =(const QueueADT&) {}
    public:
        QueueADT() {}
        virtual ~QueueADT() {}
        virtual void clear() = 0;//清空
        virtual void init() = 0;//初始化
        virtual void enqueue(const E&it) = 0;//入队
        virtual E dequeue() = 0;//出队
        virtual const E& frontValue() const = 0;//队首元素
        virtual const E& backValue() const = 0;//队尾元素
        virtual bool empty() const = 0;//判断为空
};

template<typename E>class StackADT
{
    private:
        void operator =(const StackADT&) {}
        StackADT(const StackADT&) {}
    public:
        StackADT() {}
        virtual ~StackADT() {}
        virtual void init() = 0;
        virtual void clear() = 0;
        virtual void push(const E& it) = 0;
        virtual E pop() = 0;
        virtual const E& topValue() const = 0;
        virtual bool empty() const = 0;
};
```

**2.物理数据类型设计**

```
template<typename E>class Queue:public QueueADT<E>
{
    private:
        Link<E>* front;
        Link<E>* rear;
        int size;
    public:
        Queue();
        Queue(const Queue<E>&another);//vector 会用到拷贝构造函数，这里也是进行初始化
        //Queue(Queue<E>&another);见三.4.1 保留了原先拷贝构造函数的实现，实际上因为参数类型限制不会执行到
        ~Queue();
        void init();
```

```
        void clear();
        void enqueue(const E&it);
        E dequeue();
        const E&frontValue()const;
        const E&backValue()const;
        bool empty()const;
};

template<typename E>class Stack:public StackADT<E>
{
    private:
        Link<E>*top;
        int size;
    public:
        Stack();
        ~Stack();
        void init();
        void clear();
        void push(const E&it);
        E pop();
        const E& topValue() const;
        bool empty()const;
};
```

**3.算法思想的设计**

模拟列车入轨道和出轨道的过程，直到所有入轨道的车厢都进入缓冲区为止。判断当前入轨道车厢和缓冲区车厢是否有等于 nowOut，若有则模拟 nowOut++，若无则将当前入轨道车厢加入缓冲队列，按缓冲队列原则选择，无符合要求的则开辟新的缓冲队列，从而计算出缓冲队列总个数。

**4.关键功能的算法步骤**

1.初始化 nowOut=1,k=0

2.输入入轨道编号，存入栈 train 中

3.当栈 train 不为空，即 train 未全部进入缓冲队列时

3.1 当栈顶元素=nowOut，（加入 Hk 缓冲队列），直接出栈并 nowOut++

3.2 当栈顶元素不等于 nowOut，考察每一个缓冲队列

3.2.1 若有队列队首等于 nowOut，则出队且 nowOut++

3.2.2 若无队列队首等于 nowOut,找到队尾元素小于栈首元素且最大的队列入队；若无，新建一个缓冲队列并入队

4.输出缓冲队列的个数为共建的队列个数加单独的轨道

## 三、算法性能分析

**1.时间复杂度**

依次处理入轨中的每个车厢编号：处理 n 个车厢，其中 n 是入轨中车厢的数量。对于每个车厢，执行的操作是常数时间，因此这部分的时间复杂度是 O(n)。

考察每个缓冲轨道队列：缓冲队列数量 k 由数据特点动态确定，k 一般来说比 n 小得多，但

也无法排除 k 与 n 相差无几的情况。
综上所述，算法的总体时间复杂度是 O(kn)，其中 n 是入轨中车厢的数量，k 是最终缓冲队列的数量。最差情况下 $O(n^2)$。

**2.空间复杂度**

共开辟一个栈和 k 个队列，由于每个车厢出栈后最多存储一次，空间复杂度 O(n)

**3.总结**

时间复杂度：O(kn)（k 为缓冲队列的数量，由数据动态决定）
空间复杂度：O(n)

**4.不足之处的思考与补充**

4.1.STL 中 vector 进行 push_back 时会拷贝构造，而用链表实现，指针在拷贝构造的时候有风险。包括容量不足时 vector 会整体拷贝到新的内存进行扩容，经常会出现一些问题。这里因为没有真正拷贝已有元素队列的需要，就在 Queue 类的拷贝构造函数使用和默认构造函数相同的方法，忽视原队列，并且将 vector 预置容量，以防扩容时整体全部初始化。
这种方法毕竟是暂时的特殊处理，vector 容量增大就不够灵活。但拷贝构造的参数限制是常量，无法对原队列进行出队操作也因为队列的限制无法直接访问元素，暂时没有找到普遍的处理方法。更好的选择是重新自己实现一个线性表，其中包含 append 方法，直接 new Queue 而避免拷贝构造。但由于此题的重点还是 stack 和 queue 的应用，再添加一个 list 的 ADT 比较多余，所以选择了特殊的处理方法适应本题。
4.2.包含缓冲轨道的特判，如果前面有空轨道，是可以优先选取而不用再开辟新缓冲轨道，这是在缓冲轨道的选择原则中没直接提到的。