组号: 03

上海大学计算机工程与科学学院

实 验 报 告

（数据结构 1）

学　　期：2022-2023 年 1 季

组　　长：刘彦辰

学　　号：21121319

指导教师：朱能军

成绩评定：_____（教师填写）

二〇二二年 12 月 13 日

小组信息			
姓名	学号	贡献比	签名
刘彦辰	21121319	40%	
李睿凤	21121906	30%	
车心宇	21121928	30%	

实验概述	
实验零	（熟悉上机环境、进度安排、评分制度；确定小组成员）
实验一	**约瑟夫问题变种**
实验二	**列车车厢重排问题**
实验三	?
实验四	?

I 任务分配

姓名	职责
刘彦辰	项目总体架构与测试; UI设计; 项目资源和版本管理; 问题三; 多缓冲轨道问题; 部分报告撰写;
李睿凤	问题二; 部分报告撰写;
车心宇	问题一; 部分报告撰写;

II 问题解析

已知

> 有一个“丁”字型铁路调度系统，它由相互垂直的2条铁轨组成，水平方向的为主铁轨，竖直方向的为辅助铁轨。辅助铁轨用于对车厢次序进行调整，它在主铁轨中间，把主铁轨分成左右两个部分。主铁轨左边的车厢只能从左边开到右边，或者从主铁轨左边进入辅助铁轨；辅助铁轨上的车厢只可以进入主铁轨右边。

问题一

> 现在有n节火车车厢，编号为1、2、...、n，在主铁轨的左边**按1、2、...、n的顺序驶入**，要求通过这个调度系统，在主铁轨的右边**以指定次序开出**（例如：有5节车厢以1、2、3、4、5 的次序进入，要求以3、2、5、4、1的顺序出站)。请编程求解调度过程。

问题二

> 现在有n节火车车厢，编号为1、2、...、n，在主铁轨的左边以**指定的顺序驶入**，要求通过这个调度系统，在主铁轨的右边**以1、2、...、n的顺序**开出（例如：有5节车厢以 5、3、1、2、4 的次序进入，要求以1、2、3、4、5的顺序出站)。请编程求解调度过程。

这两个问题可以合并成一个较为一般的问题.

问题三

> 现在有n节火车车厢，编号为1、2、...、n，在主铁轨的左边以**指定的顺序驶入**，要求通过这个调度系统，在主铁轨的右边以**指定顺序开出**（例如：有5节车厢以 5、3、1、2、4 的次序进入，要求以3、4、1、2、5的顺序出站)。请编程求解调度过程。

III 程序实现

问题一 算法分析

依次读取目标输出序号`d`，再进行以下步骤:

1. 如果辅轨道上有车厢，即栈非空，且`stack.top() == d`,则把`d`车厢从辅轨道驶入主轨道右边，读取下一个序号。否则执行步骤二。
2. 主轨道左边有车厢，且最前面的车厢序号`No`小于`d`，则让主轨道左边最前面的车厢依次入栈，循环执行直到`No == d`，或者主轨道左边没有车厢为止。
3. 出循环时，如果主轨道左边的车厢号`No == d`，则从主轨道左边把第d号车厢开到主轨道右边，进行下一趟循环；否则，说明`d`号车厢被压入栈内无法调出，则调度失败。

由于问题一二没有本质区别, 时间复杂度在后文单独分析.

给出该算法的伪代码.

Pesudocode for Q01_stack:

```
for i from 0 to target.size - 1
    d = target[i]; // {d} is the next number to output.
    if stack_Not_Empty && stack.top == d
        Move d from stack to output rail.
        stack.pop();
    else if No <= d
        loop while No <= target.size() && No < d
```

```
            Move d from input to stack rail.
            stack.push(No++);
        if No == d
            Move d from input to output rail.
            No++;
    else
        break;
```

问题二 算法分析

主铁轨左边的车厢（$input$）以任意顺序驶入，通过调度系统后在主铁轨的右边（$procedure$）以 1、2、$...$、n 的次序开出. 利用 $stack$ 栈作辅助铁轨. 主轨道右边应得的车厢号为 j .

判断主轨道左边最前端的车厢是否和 j 相同，若相同则进入右端；若不同则看辅助轨道最前端和 j 相同，若相同则从辅助轨道进入右端；若仍无和 j 相同的车辆输出到主轨道右端，则将主轨道左端的最前端车辆输入辅轨道。以上完成一次循环。

如果主轨道左边有车厢且最前面的车厢号 $input[i]$ 等于 j ，让主轨道左边最前面的车厢进入辅轨道.

```
 if i < input.size() && input[i] == j
    procedure += string("* Move ") + to_string(input[i])
+ " from input rail to output rail.\n";
    ++i;
    ++j;
}
```

如果辅助轨道上有车厢且辅轨道最前面的车厢号等于 j , 从辅轨道把第 j 号车厢开到主轨道右边.

```
 else if (!stack.empty() && stack.top() == j) {
    procedure += string("* Move ") +
to_string(stack.top()) + " from stack rail to output
rail.\n";
    stack.pop();
    ++j;
}
```

如果主轨道左边有车厢且最前面的车厢号 $input[i]$ 小于 j，让主轨道左边最前面的车厢进入辅轨道.

```
else if (i < input.size()) {
    procedure += string("* Move ") + to_string(input[i])
+ " from input rail to stack rail.\n";
    stack.push(input[i]);
    ++i;
}
```

其他情况发生时，车厢号为 j 的车厢被压在辅轨道内部，无法从辅轨道进入主轨道右边，则此次车厢调度失败，返回 `IMPOSSIBLE`.

问题三 算法分析

假设目标输出序列储存在数组 `target` 中; 用户输入序列储存在数组 `input` 中; 缓存轨道以栈实现, 并令其为对象 `stack`.

定义整数 `i`, 赋予初始值 0, 用以遍历 `input` (`input[i]` 表示主轨道左侧**最前**车厢的编号);

定义整数 `j`, 赋予初始值 0, 用以遍历 `target` (`target[j]` 表示主轨道右侧**待输出**车厢的编号).

关键问题为: **按照顺序在 `input` 或 `stack` 中找到当前输出目标 `target[j]`**.

以 `j` 的迭代作为外部循环 (指明当前待输出的车厢编号为 `target[j]`); 接着将所有可能情况分为四类处理:

1. `input` 不为空且 `input[i]` 等于 `target[j]`; 此时直接将 `input[i]` "移动"到右侧输出铁轨.

   > 这里的"移动" 不需要做物理结构的改变, 只需要将 `i` 和 `j` 分别增加 1 即可.

2. (`input` 为空) 或 (`input` 不为空但`input[i]` 不等于 `target[j]`), 又有 `stack` 不为空且 `stack.top` 等于 `target[j]`; 此时直接将 `stack.top()` "移动" 到右侧输出铁轨.

   > 这里的"移动"需要将 pop 栈顶元素, 且将 `j` 增加1.

3. `input` 不为空, 但 `input[i]` 不等于 `target[j]`, 且 (`stack` 为空) 或 (`stack` 不为空但 `stack.top` 不等于target[j]); 此时将input[i]"移动"入stack`.

   > 这里的"移动"需要将 `input[i]` push 入 `stack`, 且将 `i` 增加 1.

4. 其他情况, 即 `input` 为空, 但也无法将 `stack.top` 移动入输出轨道; 此时可知不可能完成调度.

下面给出程序实现的伪代码.

Pesudocode for Q03_stack:

```
i = 0 // The index of the first remaining carriage (the carrige to pop next time) on the input rail.
j = 0 // The index of the requiring carriage on the output rail.
while j < target.size
    if i < input.size and input[i] == target[j]
        Move input[i]from input rail to output rail
        ++i
        ++j
    else if stack_Not_Empty and stack.top == target[j]
        Move stack.top from stack rail to output rail
        stack.pop()
        ++j
```

```
    else if i < input.size
        Move input[i] from input rail to stack rail
        stack.push(input[i])
        ++i;
    else
        Resort IMPOSSIBLE
```

IV 复杂度分析

时间复杂度

三个问题在实现过程中没有区别.

考察问题三的伪代码:

Pesudocode for Q03_stack:

```
...
while j < target.size   // O(n)
    ...
```

对 `n` 次的目标进行操作, 而所有操作均可视为常数步, 因此算法总时间复杂度为:

$$O(n)$$

空间复杂度

考虑空间复杂度, 对于问题三, 最坏情况下需要 $3 * n$ 的空间 (分别给 `input`, `target` 和 `stack`)

因此空间复杂度为:

$$O(n)$$

V 多缓冲轨道问题

本次课程项目的列车车厢重排问题输入最简单的 (只有一个缓冲轨道) 的列车车厢重排问题;

更一般的情况是, 当有 `k` 个轨道, 如何按照要求对列车车厢进行重排.

对于这个问题解答, 我已将代码开源于以下连接:

https://github.com/jamesnulliu/DataStructure_SHU/blob/main/08_Stack/source/reSortRailCars.cpp

以上代码单纯使用栈和线性查找实现了多缓冲轨道的列车车厢重排问题, 最坏情况下的时间复杂度为:

$$O(numOfCars * numOfTracks)$$

而使用二分查找或者用平衡二叉搜索树储存缓冲轨道顶部的车厢编号, 可以将时间复杂度降低为:

$$O(numOfCars * \log{(numOfTracks)})$$

日后我将会对这些代码进行进一步更新.

by JamesNULLiu, Dec, 27, 2022