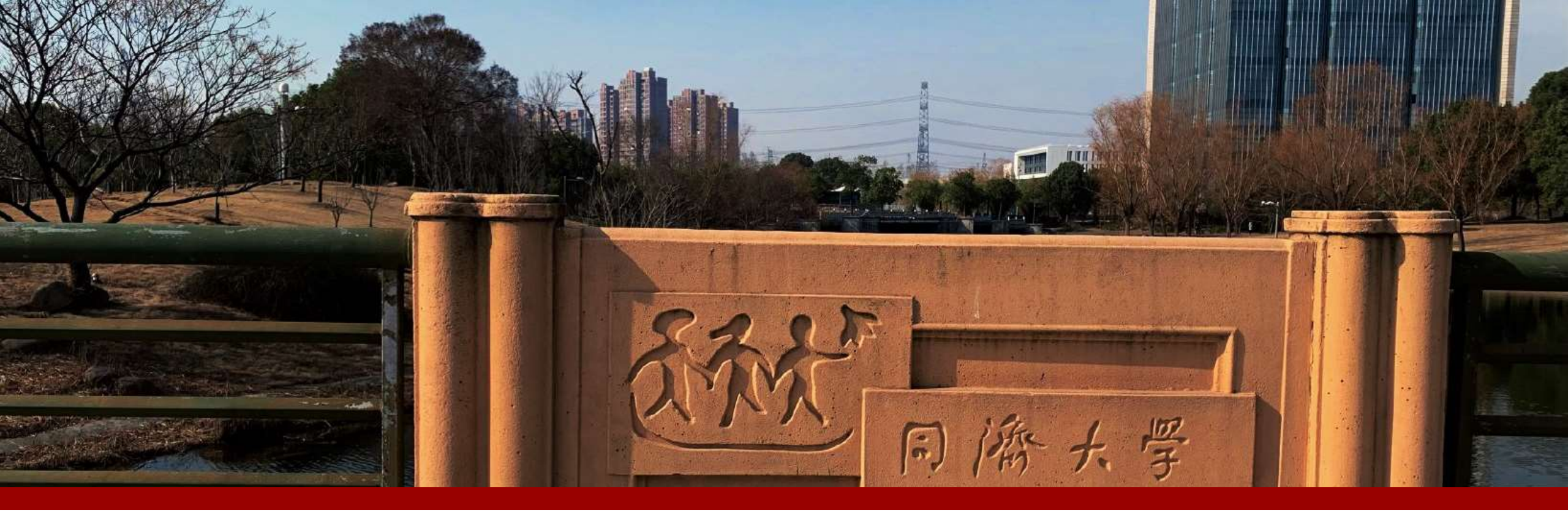
银行排队系统的设计

张嘉麟 2352595
魏余昊 2353242
周灵菲 2352570
杜泉睿 2352351
魏嘉泽 2353940
刘艺 2351305

同舟共济



01 研究问题

背景与要求

在银行或类似的服务场所，排队人系统是一个常见的应用场景。该系统的目标是按先到先服务的原则为客户提供服务，即先进先出的策略，非常适合使用队列数据结构来实现。

银行的柜合需要接照客户到达的顺宇进行服务，但有时也会有优先服务的客户（如VP客户）。因此，系统需要管理两个队列：

普通客户队列：按照客户到达的顺序进行服务，遵循FIFO (First In First Out) 原则。

VP客户队列：优先于普通客户队列中的客户进行服务。

问题分析

队列操作：两个队列分别处理不同优先级的客户。普通客户使用FIFO原则，从队首取出进行服务。VIP客户优先于普通客户得到服务，且同样遵循FIFO。

关键在于如何设计系统，使得当有VIP客户时，VIP队列先于普通客户队列得到服务。

实现流程

A. 初始化队列：创建两个队列数据结构，一个用于存储普通客户，另一个用于存储VIP客户。可以使用链表实现的队列来处理客户的入队和出队操作。

B. 客户到达：普通客户到达时，将其加入普通客户队列。VIP客户到达时，将其加入VIP客户队列。

C. 处理服务请求：当系统准备服务时，首先检查VIP客户队列是否为空。如果VIP客户队列不为空，优先从VIP客户队列中取出队首元素，进行服务。如果VIP客户队列为空，则处理普通客户队列，按照FIFO原则提供服务。如果两个队列都为空，则系统暂时没有客户需要服务。

D. 系统终止条件：当没有客户等待时，系统进入空闲状态，可以继续接收新客户或者终止。

难点解读

优先级管理

实现中最大的难点是如何有效管理VIP客户优先于普通客户。解决办法是引入两个队列，并在服务时优先检查VIP队列是否有客户待处理，这要求系统在服务时每次都要判断两个队列的状态。

边界情况处理

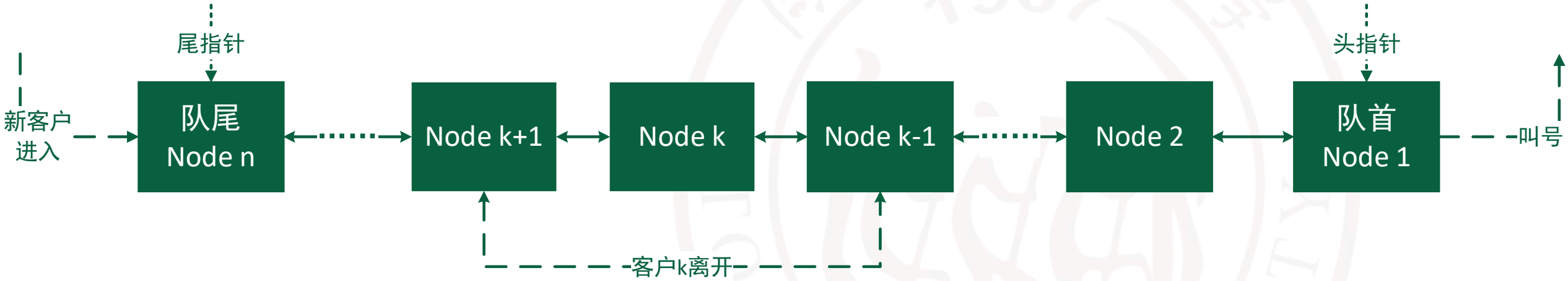
处理队列为空的情况，确保在服务客户时检查队列状态，避免在队列为空时执行无效的出队操作。当VIP和普通客户队列同时为空时，系统应当正确处理并返回合适的提示。

题目要求外的新增功能

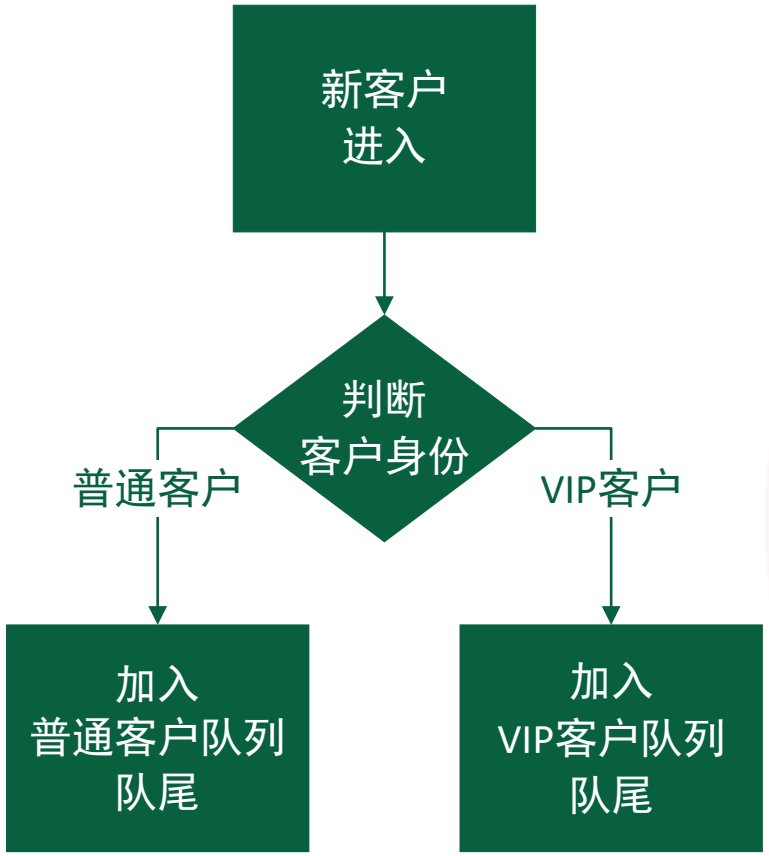
同济大学 TONGJI UNIVERSITY

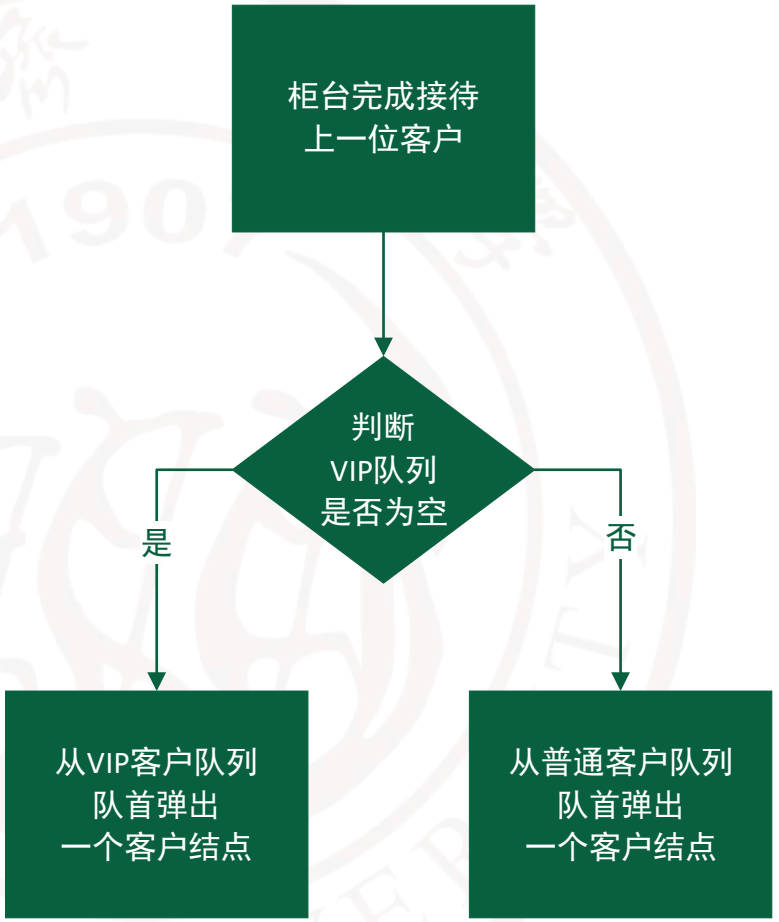
1. 重新排队功能：普通客户和VIP客户可以在不同条件下重新排队，保持队列的顺序完整性。
2. 新增叫号功能：通过增加叫号功能，可以让系统在VIP客户全部处理完后，继续处理普通客户，保证服务的顺序合理。
3. 混合数据结构管理：系统采用了队列和链表的混合结构进行客户管理，提升了数据存储和处理的灵活性。
4. 处理并发队列：系统能够同时处理两个队列，确保服务流程不冲突，且高效运行。
5. 新增的异常处理功能：处理因VIP客户加入或普通客户重排队造成的队列异常情况，保证服务流程的稳定性。

02 数据结构


尾指针
头指针
新客户
进入
队尾
Node n
Node k+1
Node k
Node k-1
Node 2
队首
Node 1
叫号
客户k离开

数据结构


新客户
进入
判断
客户身份
普通客户
VIP客户
加入
普通客户队列
队尾
加入
VIP客户队列
队尾


柜台完成接待
上一位客户
判断
VIP队列
是否为空
是
否
从VIP客户队列
队首弹出
一个客户结点
从普通客户队列
队首弹出
一个客户结点

逻辑结构

一个标准的线性队列，遵循FIFO（First In, First Out）原则，即先到达的客户先被服务。这个队列中的每个客户节点都按照到达的顺序排列，形成一个线性结构。

将这两个队列组合在一起时，我们并不宜将它们视为一个单一的树状结构，而更适合视为两个独立的线性队列，它们通过服务优先级规则相互关联。系统会根据VIP队列是否为空来决定是从VIP队列还是从普通客户队列中取出客户进行服务。

同样是一个线性队列，但具有更高的服务优先级。当VIP队列不为空时，系统会优先从VIP队列中取出客户进行服务，而不是从普通客户队列中。因此，虽然它本身也是一个线性队列，但在整体系统中，它相对于普通客户队列具有特殊的地位。

03 逻辑结构

逻辑结构

树状结构变体	两个线性结构的组合
带有优先级的树状结构或特殊形式的队列。	由两个独立的线性队列通过优先级规则相互关联而成的复合结构。
题目的直观解释	真正处理数据时采用的结构

因此，从逻辑结构的角度来看，这个银行排队系统更适合视为由两个具有不同服务优先级的线性队列组成的复合结构。

这样，既保留了线性队列的先进先出特性，又通过引入优先级规则来实现对不同类型客户的差异化服务。

04 存储结构

双向链表

同济大学 TONGJI UNIVERSITY

链表链接	链表结点内容

class Node {
friend class Queue;
private:
Customer* data; // 节点存储的客户数据
Node* prev; // 前一个节点指针
Node* next; // 后一个节点指针
 	
// 构造函数
Node(Customer* data) : data(data), prev(nullptr), next(nullptr) {}
};
	lass Customer {
	public:
	string id; // 客户ID
	string name; // 客户姓名
	int priority; // 优先级（对于VIP客户来说更高）
	};

双向链表优点

同济大学 TONGJI UNIVERSITY

动态存储	高效插入/删除	灵活的客户优先级处理
不需要预先定义大小，不设上限。能灵活应对客户数量变化。	在任意位置插入和删除操作时间复杂度为O(1)，VIP客户或因故离开的普通客户可以迅速从队列中移除	可以在链表的任意位置插入VIP客户，保持优先服务的原则。

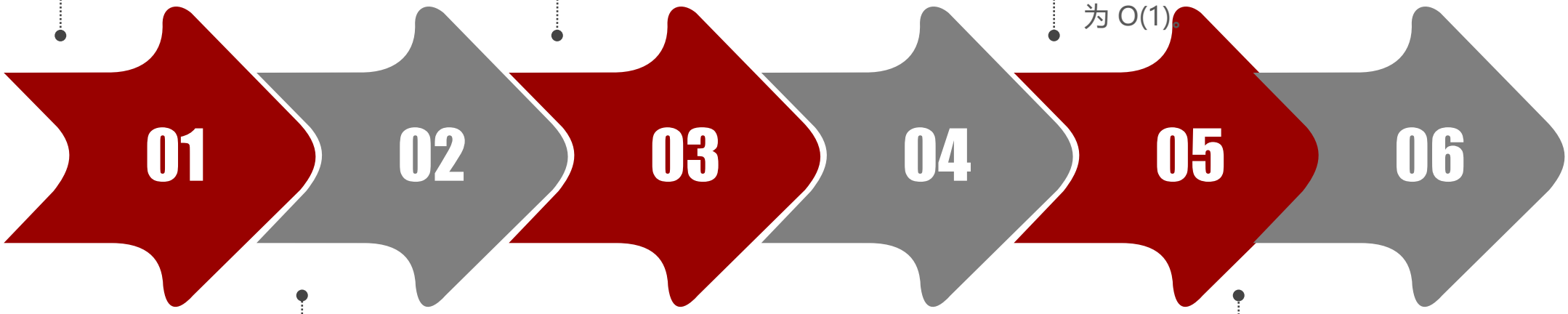
05 时间复杂度分析

时间复杂度

普通客户入队 (enqueueNormal)：
将普通客户加入普通客户队列，实际上是在队列尾部插入客户。由于我们使用的是链表实现的队列，插入操作的时间复杂度为 O(1)。

3.服务客户 (serveCustomer)：
在服务时，我们首先检查VIP客户队列是否为空。如果VIP队列不为空，从队首取出一个客户进行服务，否则处理普通客户队列。
取出队首元素的时间复杂度为 O(1)，无论是VIP客户还是普通客户，出队操作在链表头部进行，时间复杂度均为 O(1)。

队列状态检查 (isEmpty)：
检查VIP队列或普通队列是否为空，只需检查链表的头指针是否为空，时间复杂度为 O(1)。



2.VIP客户入队 (enqueueVIP)：
同样地，VIP客户的入队也是在链表尾部插入节点，时间复杂度为 O(1)。

4.重新排队 (reEnQueueCustomer)：
在某些情况下，客户可能需要重新排队。这会涉及在链表中间插入客户，或者从中删除客户并重新插入指定位置。
在链表中查找指定位置的时间复杂度为 O(n)，其中 n 是队列中客户的数量。
插入和删除操作的时间复杂度为 O(1)

查找客户 (findCustomerByQueueNumber)：
该操作会遍历普通客户和VIP客户队列，根据客户的队列号码进行查找。最坏情况下，需要遍历整个队列，时间复杂度为 O(n)。

时间复杂度

总结：

对于入队和出队操作，时间复杂度均为 O(1)，这也是队列数据结构的一大优势。

在涉及重新排队和查找客户的操作中，时间复杂度为 O(n)，因为这些操作需要遍历队列。

06 代码框架展示

代码展示

同济大学 TONGJI UNIVERSITY

1.客户和队列节点的表示

```
Class Customer {
    String id;      // 客户ID
    String name;    // 客户姓名
    Integer priority;   // 优先级（0表示普通客户，1表示VIP客户）
}。
Class Node {
    Customer data;      // 节点存储的客户数据
    Node prev;          // 前一个节点指针
    Node next;          // 后一个节点指针
}
Class Queue {
    Node head;          // 队列头节点
    Node tail;          // 队列尾节点
}
```

2. 队列操作

Method enqueue (){}//队列末尾添加新客户。如果队列为空，新节点既是头节点也是尾节点；否则，新节点成为新的尾节点。

Method dequeue() {}
//从队列头部移除客户。如果队列只有一个节点，则头尾节点都设为nullptr；否则，新的头节点是原头节点的下一个节点。

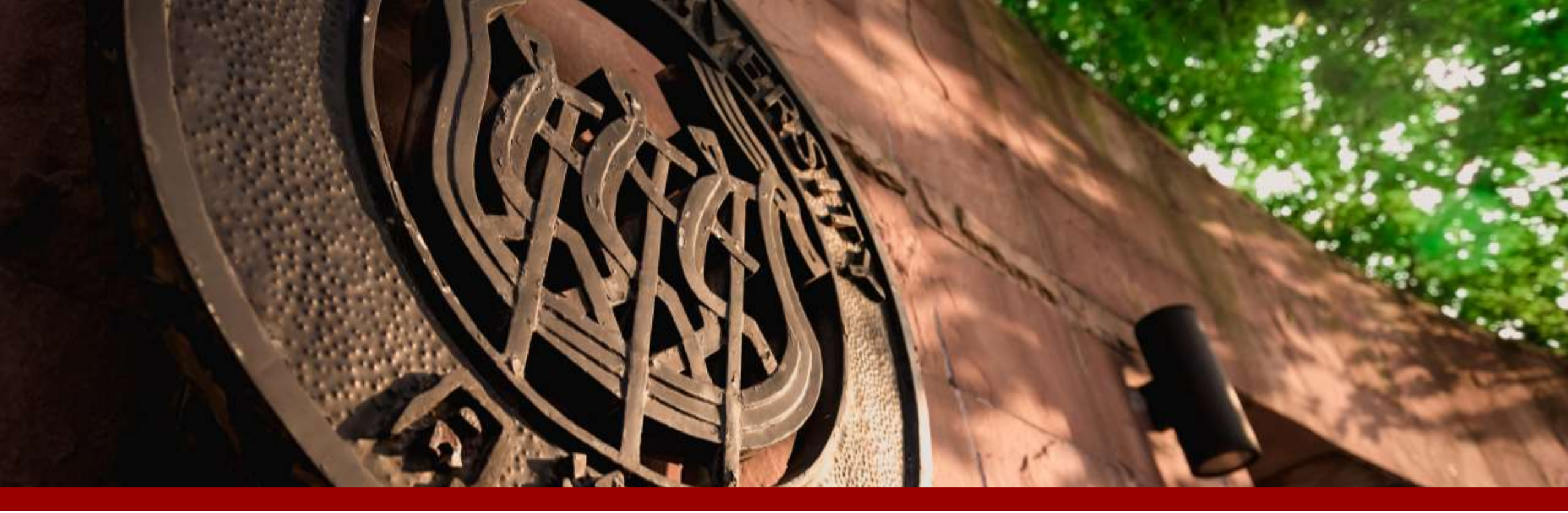
bool isEmpty() const;
//检查队列是否为空，即头节点是否为nullptr。

Method front() {}
//获取队列头部的客户数据，但不从队列中移除

代码展示

3. 银行排队系统管理

```
Class BankQueueManager {
    Queue normalQueue;    // 普通客户队列
    Queue vipQueue;       // VIP客户队列

//构造函数
Constructor() {
        normalQueue = Create Queue;
        vipQueue = Create Queue;
    }

Method enqueueNormal(customer) {
        normalQueue.enqueue(customer);
    }

Method enqueueVIP(customer) {
        vipQueue.enqueue(customer);
    }

    Method serveCustomer() {
}

    Method tempServeCustomer() {
        If vipQueue is not empty Then
            Return vipQueue.tempDequeue();
        Return normalQueue.tempDequeue();
    }
}
```



汇报结束 感谢聆听！

同舟共济