C++ 算法讲义：贪心算法（中）

1. 引言

在上节课，我们解决了一个资源分配问题(部分背包)。今天，我们从资源调度、任务处理等领域来看贪心策略的应用。我们将通过“排队接水”这个经典问题，来进行探讨。

1.1 问题引入：排队接水

有 n 个人在一个水龙头前排队接水，每个人接水的时间为 T_i。请设计一种排队顺序，使得所有人的**平均等待时间**最小。注意：一个人的等待时间，是指排在他前面所有人接水时间的总和，不包括他自己的接水时间。

---

2. 贪心模型一：排序贡献问题

2.1 问题分析：从“平均”到“总和”

对于“排队接水”问题，我们的目标是“平均等待时间最小”。根据平均值的定义：
$$\text{平均等待时间} = (\text{所有人的等待时间之和})/n$$

由于人数 n 是固定的，要让平均值最小，就等价于让分子 **所有人的等待时间之和** 最小。问题被简化了。

现在，我们来分析“等待时间之和”是如何构成的。假设排队顺序为 $t_1, t_2, \ldots, t_n$：

- 第1个人等待时间：0
- 第2个人等待时间：t_1
- 第3个人等待时间：$t_1 + t_2$
- ...
- 第n个人等待时间：$t_1 + t_2 + \ldots + t_{n-1}$

将它们全部相加，得到总等待时间 $Total$。如果我们换个角度看，每个人的接水时间 t_i 对总时间的“贡献”是多少？

- t_1 的时间，被后面的 $n-1$ 个人都等了一遍，所以它在总和中被加了 $n-1$ 次。
- t_2 的时间，被后面的 $n-2$ 个人都等了一遍，所以它被加了 $n-2$ 次。
- ...
- t_i 的时间，被它后面的 $n-i$ 个人等待，所以被加了 $n-i$ 次。

于是，我们得到了总等待时间的数学表达式：
$$Total = (n-1)*t_1 + (n-2)*t_2 + \ldots + 1*t_{n-1} + 0*t_n$$

2.2 贪心策略与证明

观察这个公式，我们发现，排在越前面的人，其接水时间 t_i 所乘的**权重** $(n-i)$ **越大**。为了让这个加权和最小，一个非常自然的贪心策略应运而生：**让值小的数，乘以大的权重；值大的数，乘以小的权重。**

换句话说，**接水时间最短的人应该排在最前面。**

这个策略的正确性，可以用我们在上一讲提到的**交换论证法(反证法)**来严格证明：

1. **假设**存在一个最优解，但它不完全是按接水时间升序排列的。
2. 那么，这个队列中必然存在一对**相邻**的人，比如第 i 个人和第 $i+1$ 个人，他们的接水时间是逆序的，即 $t_i > t_{i+1}$。

3. 我们保持其他人不动，只**交换** i 和 $i+1$ 的位置。
4. 分析总等待时间的变化。交换前，这两项对总和的贡献是 $\ldots+(n-i)*t_i+(n-i-1)*t_{i+1}+\ldots$。
5. 交换后，贡献变为 $\ldots+(n-i)*t_{i+1}+(n-i-1)*t_i+\ldots$。
6. 两者之差为：$[(n-i)t_i+(n-i-1)t_{i+1}]-[(n-i)t_{i+1}+(n-i-1)t_i]=(t_i-t_{i+1})$。
7. 因为我们设 $t_i > t_{i+1}$，所以差值大于0。这意味着，交换后的总等待时间**更小**了。
8. 这与我们"假设存在一个非升序的最优解"相**矛盾**。因此，任何非升序的排列都不是最优的。最优解必须是按接水时间从小到大排列。

2.3 关键代码实现

有了被证明的策略，代码实现就非常直接了。

```
struct Person {
    int id;   // 原始编号
    int time; // 接水时间
};

// 自定义比较函数
bool cmp(const Person& a, const Person& b) {
    if (a.time != b.time) {
        return a.time < b.time; // 主要按接水时间升序
    }
    return a.id < b.id; // 时间相同，按编号升序
}

int main() {
    // total_wait_time 为总等待时间
    // current_wait_time 为前面人的接水总时长
    for (int i = 1; i < n; ++i) {
        current_wait_time += people[i - 1].time;
        total_wait_time += current_wait_time;
    }
}
```

---

3. 贪心模型二：区间选择问题

现在我们来看另一大类问题：从一堆带有起止时间（或左右端点）的事件（或区间）中，按某种规则进行选择。

3.1 问题引入：活动安排 (凌乱的yyy)

这个问题其目标是在大量有时间冲突的活动中，选择出**数量最多**的、互不冲突的活动。

我们已经知道，正确的贪心策略是：**将所有活动按结束时间升序排序，然后依次选择与已选活动不冲突的活动。**

这个策略的直觉是：**尽快结束当前活动，为未来留下最大的选择空间。** 它的正确性，也可以用 反证法/交换论证法 严格证明，篇幅限制这里就不证明了。

3.2 模型抽象：最多不相交区间问题

“活动安排”问题可以被抽象成一个数学模型：**最多不相交区间问题。**

给定 n 个闭区间 $[l_i, r_i]$，选择最多的区间，使得它们两两不相交。

这里的映射关系是：

- 一场活动 <==> 一个区间 $[l, r]$
- 开始时间 <==> 左端点 l
- 结束时间 <==> 右端点 r
- 活动不冲突 <==> 区间不相交

只要能抽象出问题的核心，无论它被包装成什么样子，解法都是一样的：**按右端点排序，然后贪心选取。**

3.3 模型应用与变种

掌握了这个模型，我们就能解决一系列看似不同的问题。

变种1：最少资源覆盖所有目标 (射气球)

问题：在 X 轴上，有一些水平方向的气球。每个气球可以由一个开始和结束坐标去表示，要求用最少的箭射爆所有气球。一支箭可以射爆所有穿过其发射位置的气球。
分析：这个问题要求“最少”的资源（箭）来“覆盖”所有目标（气球）。我们转换一下思路：每当我们**不得不**使用一支新箭时，才用。什么时候是“不得不”呢？就是当出现一个新气球，它无法被上一支箭覆盖时。
贪心策略：

1. 将所有气球（区间）按**右端点**从小到大排序。
2. 初始化箭数 $arrows = 1$。取第一个气球，并假想在它的右端点 r_1 处射出第一支箭。这个位置是在能射爆第一个气球的前提下，能“照顾”到后续气球范围最广的位置。
3. 遍历剩下的气球。如果某个气球 $[l_i, r_i]$ 的左端点 $l_i r_1$，说明它已经逃出了第一支箭的范围，我们**不得不**发射一支新箭。
4. 此时，增加箭数 $arrows++$，并更新我们的“箭靶”位置为这个新气球的右端点 r_i。
5. 重复此过程，直到所有气球都被检查完毕。

变种2：最少点覆盖所有区间 (区间选点)

问题：假设有 n 个区间，分别是 `[l₁, r₁], [l₂, r₂], ..., [lₙ, rₙ]`。 我们需要从这 n 个区间中选择出若干个区间，要求选出的这些区间**两两不相交**（端点相交的情况，如 [1,3] 和 [3,5]，不算相交）。最多能选出多少个这样的区间呢?

分析：这和射气球问题几乎完全一样！因为选一个点 就等价于 射一支箭。
贪心策略：

1. 将所有区间按**右端点**从小到大排序。
2. 选择第一个区间的**右端点**作为第一个点。
3. 遍历后续区间，跳过所有已经被这个点覆盖的区间。
4. 找到第一个未被覆盖的区间，选择它的右端点作为第二个点。
5. 重复此过程。

---

4. 总结

- **贪心模型一 (加权和贪心排序)**：当总成本是每个元素按其排序位置的加权和时，通常的策略是将“最优”的元素放在“贡献最大”的位置。
- **贪心模型二 (区间选择)**：处理区间冲突、覆盖、选择问题时，**按右端点排序**是常见的贪心标准。因为它总是为**未来保留最大的可能性**。
- **识别模型**：算法学习的关键在于识别问题的数学模型。学会将具体问题抽象化，这样就能通过我们学习的算法去解决各种实际问题。