算设串讲

王柏森

人工智能研究院

November 14, 2021

目录

目录

如何评价算法

空间与时间，最优、平均与最差
O, Ω, Θ 渐进符号描述了一个算法时空的复杂度的无穷大
Θ 渐进符号描述了一个算法时空的复杂度的等价无穷大
例子：

$$\log_{10} n, \sin n, 10^{10}, \sum_{i=1}^{n} i^k (k \geq 1), \sum_{i=1}^{n} \frac{1}{i}$$

利用递推式计算复杂度，一般形式：

$$T(n) = aT(\lfloor n/b \rfloor) + O(n^d)$$

时间复杂度计算方法

具体方法

时间复杂度计算方法

具体方法

1. 递归树

时间复杂度计算方法

具体方法

1. 递归树
2. 代入法

时间复杂度计算方法

具体方法

1. 递归树
2. 代入法
3. 主定理

主定理

若

$$T(n) = aT(\lfloor n/b \rfloor) + O(n^d)$$

且式中 $a > 0$，$b > 1$，$d \geq 0$，则：

$$T(n) = \begin{cases} O(n^d), & d > \log_b a \\ O(n^d \log n), & d = \log_b a \\ O(n^{\log_b a}), & d < \log_b a \end{cases}$$

时间复杂度计算方法

$$T(1) = T(2) = 1$$
$$T(n) = T(n-2) + 1, \quad \text{if } n > 2$$

$$T(1) = 1$$
$$T(n) = T(n/2) + \log n, \quad \text{if } n > 1$$

目录

基本原理

分治基于主定理，构建一棵宽度、深度相适应的搜索树，使得整体问题的复杂度降低。

- 分：将问题分为若干小问题，子问题的解对整体问题的影响越小越好。

基本原理

分治基于主定理，构建一棵宽度、深度相适应的搜索树，使得整体问题的复杂度降低。

- 分：将问题分为若干小问题，子问题的解对整体问题的影响越小越好。
- 治：解决子问题。

基本原理

分治基于主定理，构建一棵宽度、深度相适应的搜索树，使得整体问题的复杂度降低。

- 分：将问题分为若干小问题，子问题的解对整体问题的影响越小越好。
- 治：解决子问题。
- 合：综合子问题的解，得到综合的、整体的答案。

逆序对问题

判断一个序列中，$a_i > a_j$ 而 $i < j$ 的 (i, j) 对数。
这个问题可以二分计算，因为将数组某一半的逆序对个数计算出来，并不影响这一半与另一半的逆序对个数。同时，对某一半进行任意操作后，不影响 i，j 在不同组时，仍然是否是逆序对。
因此可以计算前一半的逆序对后，排序，再计算另一半，最后将两半的结果合起来。
伪代码如下：

逆序对问题

Algorithm 1: Inverse

Input: 待判断的序列 a, 起始下标 i，终止下标 j
Output: 排好序的序列 a，序列 a 逆序对个数

```
if i = j then
    return 0
else
    ans ← 0,   m ← (i + j)/2
    (s1, ans1) ← Inverse(a[],i,m)
    (s2, ans2) ← Inverse(a[],m+1,j)
    这时 a[i...m], a[m + 1...j] 有序
    l ← i, r ← m + 1
    while l ≠ mandr ≠ j do
        if a[l] > a[r] then
            将 a[l] 置于 s 数组最后
            l ++
        else
            将 a[l] 置于 s 数组最后
            r ++
            ans += m - l - 1
            为此时前半段长度
        end
    end
    if l = m then
        将 a[r...j] 置于 s 数组最后
    else
```

逆序对问题

由于前后两半有序，则排序是 $O(n)$ 的。
递推式 $T(n) = 2T(\lfloor n/2 \rfloor) + O(n)$
由主定理，复杂度为 $O(n \log n)$。

字符串等价关系判定

给定两个长度为 n 的字符串 A 和 B，若称 A 与 B 是等价的，当且仅当它们满足如下关系之一：

- A 和 B 完全相同；

请你设计一个高效的算法来判断两个字符串是否等价并分析你的算法的时间复杂度。
找到子问题很简单。先注意这里的等价是有传递性的，则可以根据此优化子问题的结构。

字符串等价关系判定

给定两个长度为 n 的字符串 A 和 B，若称 A 与 B 是等价的，当且仅当它们满足如下关系之一：

- A 和 B 完全相同；
- 若将把 A 分成长度相等的两段 A1 和 A2，也将 B 分成长度相等的两段 B1 和 B2。且他们之间满足如下两种关系之一：a. A1 和 B1 等价且 A2 和 B2 等价；b. A1 和 B2 等价且 A2 和 B1 等价。

请你设计一个高效的算法来判断两个字符串是否等价并分析你的算法的时间复杂度。
找到子问题很简单。先注意这里的等价是有传递性的，则可以根据此优化子问题的结构。

平面间上的最接近点对

给定平面上 n 个点，找出其中的一对点的距离，使得在这 n 个点的所有点对中，该距离为所有点对中最小的。

平面间上的最接近点对

给定平面上 n 个点，找出其中的一对点的距离，使得在这 n 个点的所有点对中，该距离为所有点对中最小的。BruteForce: 比较，爆搜，$O(n^2)$

Algorithm 3: BruteForce

Input: 点的坐标 x_i 和 y_i
Output: 最小距离

```
ans ←
for i = 1,i ≤ n do
    for j = i + 1,j ≤ n do
        ans ← min(ans, sqrt((x_i − x_j)^2 + (y_i − y_j)^2)) end
    end
    return ans
```

平面上的最接近点对

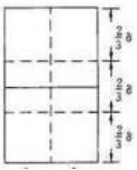
给定平面上 n 个点，找出其中的一对点的距离，使得在这 n 个点的所有点对中，该距离为所有点对中最小的。
分治算法，$O(n \log n)$。
我们先把所有点按照 x(横坐标) 从左到右升序排列。
以 X 横坐标中间的点作为分界线. 将平面的点分成左边和右边。
然后找到左边的点中最近点对的距离 d1, 和右边最近点对的距离 d2。
最后返回 $d = \min(d1, d2)$。

平面上的最接近点对

给定平面上 n 个点，找出其中的一对点的距离，使得在这 n 个点的所有点对中，该距离为所有点对中最小的。
分治算法，$O(n \log n)$。
我们先把所有点按照 x(横坐标) 从左到右升序排列。
以 X 横坐标中间的点作为分界线. 将平面的点分成左边和右边。
然后找到左边的点中最近点对的距离 d1, 和右边最近点对的距离 d2。
最后返回 $d = \min(d1, d2)$。
怎么判断左右两边的点对的距离？
我们可以利用 d 来寻找点，即横坐标与中位数差值的绝对值大于 d 的点对都不是最近点对。
这样与中位数差值的绝对值的点可能左右各一半，搜索需要 $\left(\frac{n}{2}\right)^2$ 次，这样的复杂度是 $O(n^2)$！
该怎么办？

平面上的最接近点对

1985 年 Preparata 和 Shamos 在给出该问题的一个分治算法，并且还具体分析了在 $[\mathrm{mid}-\mathrm{dis}, \mathrm{mid}+\mathrm{dis}]$ 区域中出现的情况，若 (p, q) 是 q 的最近点对，p 在带域左半部分，则 q 点必在下图所示的 $\delta * 2\delta$ 长方形上，而在该长方形上，最多只能由右边点集的 6 个点。每个点对之间的距离不小于 δ。



搜索变成了 $\mathrm{O}(6\mathrm{n}) = \mathrm{O}(\mathrm{n})$！递推式 $\mathrm{T}(\mathrm{n}) = 2\mathrm{T}(\lfloor \mathrm{n}/2 \rfloor) + \mathrm{O}(\mathrm{n})$
由主定理，复杂度为 $\mathrm{O}(\mathrm{n}\log\mathrm{n})$。由主定理，复杂度为 $\mathrm{O}(\mathrm{n}\log\mathrm{n})$。

目录

介绍

动态规划是很难、极富变化的一章。
动态规划本质是将问题不同状态的最优解记录下来，再通过状态转移方程将状态进行转移。
如果需要记录方案，则需要在转移状态时记录状态。
最后需要判断边界条件和目标状态。
即五步：设计状态、状态转移、记录方案、边界条件与目标状态。
简单起见，本章仅写状态转移方程，伪代码略。

三角形最小路径和

问题：给定一个三角形，找出自顶向下的最小路径和。一步只能移动到下一行中相邻的结点上。相邻的结点指下标与上一层结点下标相同或者等于上一层结点下标 $+1$ 的两个结点。也就是说，如果正位于当前行的下标 i，那么下一步可以移动到下一行的下标 i 或 $i+1$。

三角形最小路径和

问题：给定一个三角形，找出自顶向下的最小路径和。一步只能移动到下一行中相邻的结点上。相邻的结点指下标与上一层结点下标相同或者等于上一层结点下标 $+1$ 的两个结点。也就是说，如果正位于当前行的下标 i，那么下一步可以移动到下一行的下标 i 或 $i+1$。
设计状态：设 $f[i][j]$ 为从 1 到 i 层第 j 个元素最小路径和。
状态转移：

$$f[i][j] = \min(f[i-1][j], f[i-1][j-1]) + a[i][j]$$

记录方案：记 $pre[i][j]$ 为更新后前一个下标 j'
边界条件：$f[0][j] = 0$
目标状态：$\min_j f[n][j]$
复杂度：$O(n^2)$

背包问题

问题：一个背包容量为 W，有 n 个物品，第 i 个物品体积、价值分别为 v[i],w[i]，每个物品要么不装，要么仅装一个，问能装的最大价值为

背包问题

问题：一个背包容量为 W，有 n 个物品，第 i 个物品体积、价值分别为 v[i],w[i]，每个物品要么不装，要么仅装一个，问能装的最大价值为

设计状态：设 f[i][j] 为仅考虑前 i 个物品，背包容量为 j 时的 value 最大值。

状态转移：

$$f[i][j] = \begin{cases} \max(f[i-1][j-v[i]] + w[i], f[i-1][j]), & j \geq v[i] \\ f[i-1][j], & j < v[i] \end{cases}$$

记录方案：可以记 pre[i][j] 表示 f[i][j] 是由什么状态转移来的，如果是 $f[i-1][j-v[i]]$，记为 1，否则置 0 即可。

边界条件：$f[0][j] = f[i][0] = 0$

目标状态：f[n][W]

最后分析复杂度，应为 O(nW)。

本题可以将空间复杂度换成 O(n)，但空间复杂度不为考试要求，故略。

LIS

问题：一个序列，找出它最长上升子序列长度，其中子序列不需要连续。

LIS

问题：一个序列，找出它最长上升子序列长度，其中子序列不需要连续。
设计状态：设 f[i] 为 s 从 1 到 i 最长公共子序列长度。
状态转移：

$$f[i] = \max_{s[j]<s[i]且1\le j<i}(f[j]+1)$$

记录方案：记 pre[i] 为更新后前一个下标 j
边界条件：$f[1]=1$
目标状态：f[n]
复杂度：$O(n^2)$。

LCS

问题：两个序列，找出他们最长公共子序列长度，其中子序列不需要连续

LCS

问题：两个序列，找出他们最长公共子序列长度，其中子序列不需要连续
设计状态：设 f[i][j] 为 s1 从 1 到 i 和 s2 从 1 到 j 最长公共子序列长度。
状态转移：

$$\mathsf{f[i][j]} = \begin{cases} \mathsf{f[i-1][j-1]} + 1, & \mathsf{s1[i]} = \mathsf{s2[j]} \\ \max(\mathsf{f[i-1][j]}, \mathsf{f[i][j-1]}), & \mathsf{s1[i]} \neq \mathsf{s2[j]} \end{cases}$$

记录方案：对于三种状态分别记 pre[i][j] 为 1，2，3.
边界条件：$\mathsf{f}[0][\mathsf{j}] = \mathsf{f}[\mathsf{i}][0] = 0$
目标状态：f[n][m]
复杂度：O(nm)

MED

问题：两个串，记三种操作：加、删、改，问将第一个串编辑为第二个串的最少操作次数为？

MED

问题：两个串，记三种操作：加、删、改，问将第一个串编辑为第二个串的最少操作次数为？
设计状态：设 f[i][j] 为 s1 从 1 到 i 和 s2 从 1 到 j 最小编辑距离。
状态转移：

$$f[i] = \min\left(\underbrace{f[i-1][j]}_{\text{加}}, \underbrace{f[i][j-1]}_{\text{删}}, \underbrace{\begin{cases} f[i-1][j-1]-1, & s1[i]=s2[j] \\ f[i-1][j-1], & s1[i]\neq s2[j] \end{cases}}_{\text{改}}\right)+1$$

记录方案：一共四种状态，记 pre[i][j] 为 1，2，3，4 即可。
边界条件：$f[1][i] = f[i][1] = i$
目标状态：f[n][m]
复杂度：O(nm)

最大路径法

问题：将一句话分成若干个词语或单字，其中词语可以在词典 dic 中查询，查询可以按 $O(1)$ 计算，求分出词语和单字两者和最小的分词结果。

最大路径法

问题：将一句话分成若干个词语或单字，其中词语可以在词典 dic 中查询，查询可以按 $O(1)$ 计算，求分出词语和单字两者和最小的分词结果。
设计状态：设 f[i] 为从第一个字到 i 个字最少切分数量。
状态转移：

$$f[i] = \min_{1 \le j < i} \begin{cases} f[j] + 1, & text[j:i] \in dic \\ f[j] + (i - j), & text[j:i] \notin dic \end{cases}$$

记录方案：记 pre[i] 为第 i 个字作为分词末尾的分词头下标.
边界条件：$f[1] = 1$
目标状态：f[n]
复杂度：$O(n)$

鸡蛋掉落

问题：k 个相同鸡蛋，一栋高为 n 的楼，已知存在楼层 f，$0 \leq f \leq n$，任何高于 f 层落下的鸡蛋会碎，任意不高于 f 层落下的鸡蛋不会破。若鸡蛋在实验中碎了就不能再被使用，否则可以重复使用。求 f 与最小操作次数。

鸡蛋掉落

问题：k 个相同鸡蛋，一栋高为 n 的楼，已知存在楼层 f，$0 \le f \le n$，任何高于 f 层落下的鸡蛋会碎，任意不高于 f 层落下的鸡蛋不会破。若鸡蛋在实验中碎了就不能再被使用，否则可以重复使用。求 f 与最小操作次数。

设计状态：设 f[i][j] 为从 1 到 i 层用 j 个鸡蛋的最小操作次数。

状态转移：

$$f[i][j] = \min_{0 \le j \le n} (\max_{0 \le p \le j} (f[i-p][j] + f[i-1][j-1]) + 1)$$

记录方案：记 pre[i][j] 为更新后前一个下标 p，与具体使用的方法。

边界条件：$f[0][j] = 0, f[i][1] = i$

目标状态：f[f][k]

复杂度：$O(kn^2)$

矩阵链乘

问题：若干矩阵做乘法，怎么加括号使得乘法计算次数最少

矩阵链乘

问题：若干矩阵做乘法，怎么加括号使得乘法计算次数最少
设计状态：设 f[i][j] 为第 i 个矩阵乘到第 j 个矩阵的最少乘法计算次数。
状态转移：

$$f[i][j] = \begin{cases} 0, & i = j \\ \min\limits_{i \le k < j}(f[i][k] + f[k+1][j] + p_{i-1}p_k p_j), & i < j \end{cases}$$

式中数组 p 的意义为：矩阵 A_i 是一个 $p[i-1]$ 行 $p[i]$ 列矩阵。
记录方案：可以记 pre[i][j] 表示 f[i][j] 是由什么状态转移来的，如果是 k，记为 k。
边界条件：$f[i][i] = 0$
目标状态：$f[1][n]$
最后分析复杂度，应为 $O(n^3)$。

石子合并

问题：一排石子，每次合并将相邻两堆合成一堆，能获得新的一堆中石子数量的和的得分，问最大得分为？

石子合并

问题：一排石子，每次合并将相邻两堆合成一堆，能获得新的一堆中石子数量的和的得分，问最大得分为？
设计状态：设 f[i][j] 为区间 [i, j] 内所有石子合并到一起的最大得分
状态转移：

$$f[i][j] = \max_{i \le k < j}(f[i][k] + f[k+1][j] + \sum_{t=i}^{j} a_t)$$

记录方案：可以记 pre[i][j] 表示 f[i][j] 是由什么状态转移来的，如果是 k，记为 k。
边界条件：$f[i][i] = 0$
目标状态：$f[1][n]$
最后分析复杂度，应为 $O(n^3)$。
也可以贪心。

戳气球

问题：有 n 个气球，编号为 0 到 $n-1$，每个气球上都标有一个数字，戳破某个气球，可以获得该气球与相邻两个气球上数字的乘积的积分，求能获得的最大积分。（若戳第一个或最后一个，则只有两个数相乘）

戳气球

问题：有 n 个气球，编号为 0 到 $n-1$，每个气球上都标有一个数字，戳破某个气球，可以获得该气球与相邻两个气球上数字的乘积的积分，求能获得的最大积分。（若戳第一个或最后一个，则只有两个数相乘）

补充：记 $a[-1] = a[n] = 1$

设计状态：设 $f[i][j]$ 为区间 (i, j) 内所有石子合并到一起的最大得分

状态转移：

$$f[i][j] = \max_{i<k<j}(f[i][k] + f[k][j] + a[i]a[k]a[j])$$

记录方案：可以记 $pre[i][j]$ 表示 $f[i][j]$ 是由什么状态转移来的，如果是 k，记为 k。

边界条件：$\forall i \geq j-1,\ f[i][j] = 0$

目标状态：$f[0][n+1]$

最后分析复杂度，应为 $O(n^3)$。

目录

介绍

贪心算法难点基本有三点：一是判断一个问题是贪心问题还是其它问题。第二是找到最优方法。最后就是证明策略。这需要对最优子结构掌握较为熟练。
本部分难点在于证明，故选择了少量题目，仅展示具体题目证明的思考。
贪心算法的本质就是在每个选择中都保持最优，但最后得到的解一般不是最优解，因此贪心算法证明显得格外重要。贪心算法是个人都能找到，但证明正确性就很难了。
如何证明？一般方法如下：

贪心证明方法

1. 微扰法（邻项交换）：证明在任意状态下，任何对局部最优策略的微小改变都会造成整体结果变差。这种方法在排序贪心的正确性证明中常用。例如活动选择问题。排序问题一般使用排序不等式进行证明。有时也会用相邻项交换，证明可以参考冒泡排序正确性证明。

贪心证明方法

1. 微扰法（邻项交换）：证明在任意状态下，任何对局部最优策略的微小改变都会造成整体结果变差。这种方法在排序贪心的正确性证明中常用。例如活动选择问题。排序问题一般使用排序不等式进行证明。有时也会用相邻项交换，证明可以参考冒泡排序正确性证明。
2. 范围缩放：证明任何对局部最优策略作用范围的扩展都不会造成整体结果变差。例如合并果子（改版）

贪心证明方法

1. 微扰法（邻项交换）：证明在任意状态下，任何对局部最优策略的微小改变都会造成整体结果变差。这种方法在排序贪心的正确性证明中常用。例如活动选择问题。排序问题一般使用排序不等式进行证明。有时也会用相邻项交换，证明可以参考冒泡排序正确性证明。
2. 范围缩放：证明任何对局部最优策略作用范围的扩展都不会造成整体结果变差。例如合并果子（改版）
3. 决策包容性：证明在任意状态下，做出局部最优策略之后，在问题状态空间中的可达集合包含了作出其它任何决策后的可达集合。即：这个局部最优策略提供的可能性包含其它所有策略提供的可能性。例如跳跃游戏。

贪心证明方法

1. 微扰法（邻项交换）：证明在任意状态下，任何对局部最优策略的微小改变都会造成整体结果变差。这种方法在排序贪心的正确性证明中常用。例如活动选择问题。排序问题一般使用排序不等式进行证明。有时也会用相邻项交换，证明可以参考冒泡排序正确性证明。
2. 范围缩放：证明任何对局部最优策略作用范围的扩展都不会造成整体结果变差。例如合并果子（改版）
3. 决策包容性：证明在任意状态下，做出局部最优策略之后，在问题状态空间中的可达集合包含了作出其它任何决策后的可达集合。即：这个局部最优策略提供的可能性包含其它所有策略提供的可能性。例如跳跃游戏。
4. 反证法。例如加油站。

贪心证明方法

1. 微扰法（邻项交换）：证明在任意状态下，任何对局部最优策略的微小改变都会造成整体结果变差。这种方法在排序贪心的正确性证明中常用。例如活动选择问题。排序问题一般使用排序不等式进行证明。有时也会用相邻项交换，证明可以参考冒泡排序正确性证明。
2. 范围缩放：证明任何对局部最优策略作用范围的扩展都不会造成整体结果变差。例如合并果子（改版）
3. 决策包容性：证明在任意状态下，做出局部最优策略之后，在问题状态空间中的可达集合包含了作出其它任何决策后的可达集合。即：这个局部最优策略提供的可能性包含其它所有策略提供的可能性。例如跳跃游戏。
4. 反证法。例如加油站。
5. 数学归纳法

分数背包问题

问题：一个背包容量为 W，有 n 个物品，第 i 个物品体积、价值分别为 v[i],w[i]，每个物体可带一部分，问能装的最大价值为？

分数背包问题

问题：一个背包容量为 W，有 n 个物品，第 i 个物品体积、价值分别为 v[i],w[i]，每个物体可带一部分，问能装的最大价值为？
贪心策略：考虑价重比，每次仅需要选价重比最优的即可。

分数背包问题

问题：一个背包容量为 W，有 n 个物品，第 i 个物品体积、价值分别为 v[i],w[i]，每个物体可带一部分，问能装的最大价值为？
贪心策略：考虑价重比，每次仅需要选价重比最优的即可。
证明：不失一般性，认为物品下标以价重比由大到小排序。
设任意方案 P'，设贪心方案为 P，第 i 个物品取 x_i 份，对应的，方案 P' 的为 y_i 份，则显然有 $\sum x_i = \sum y_i - \min(n, W)$，否则随便拿一点其它物品即可得到更大的价值。两式做差，有：

$$\sum (x_i - y_i) w_i = \sum x_i w_i - \sum y_i w_i$$

由排序不等式，因为 x_i、w_i 均为由大到小排序，因此上式大于等于 0，当且仅当 y_i 由大到小排序时取等号。因此 P 为最优方案。得证。□
复杂度显然为排序复杂度，$O(n \log n)$。

活动选择问题

问题：一系列活动，给出开始结束时间 s[i],t[i]，问能参加的最多活动数为？

活动选择问题

问题：一系列活动，给出开始结束时间 s[i],t[i]，问能参加的最多活动数为？
怎样才能活动参加的最多呢？比如有许多 npy 等着圣诞节约会，如何选择尽可能多的呢？
当然选择早完事早好，早完事就有更多时间选择了。因此贪心策略如下：选择最早结束的活动。

活动选择问题

问题：一系列活动，给出开始结束时间 s[i],t[i]，问能参加的最多活动数为？
怎样才能活动参加的最多呢？比如有许多 npy 等着圣诞节约会，如何选择尽可能多的呢？
当然选择早完事早好，早完事就有更多时间选择了。因此贪心策略如下：选择最早结束的活动。
证明：不失一般性，假设活动时间下标按照结束时间前后排序。
设该策略为 P，若有一策略 P' 更优，则 P' 中至少有一个活动，结束时间比 P 中晚，否则 $P = P'$。记第一个不同的活动在 P，P' 中为 a_i，b_i，则 P' 在 b_i 后完成的所有活动皆可由 P 完成，因为 P 中 a_i 完成的时间更早。因此 P 完成的个数应该大于等于 P'，从而 $P = P'$，得证。□
伪代码略。
复杂度显然为排序复杂度，$O(n \log n)$。

合并果子（改版）

问题：一排石子，每次合并将相邻两堆合成一堆，设两堆分别为 a, b，则合成新的一堆石子数量为 $2\sqrt{ab}$，并同时获得 $2\sqrt{ab}$ 分，问最大得分为？

合并果子（改版）

问题：一排石子，每次合并将相邻两堆合成一堆，设两堆分别为 a, b，则合成新的一堆石子数量为 $2\sqrt{ab}$，并同时获得 $2\sqrt{ab}$ 分，问最大得分为？
本题策略为每次合并合并果子数最多的两堆。下证该算法的正确性。

合并果子（改版）

证明：Part 1 若每一次都向同一堆果子添加，若顺序为 $a_1, a_2, \ldots, a_n$，则易得最后一堆果子的权重为

$$2\sqrt{w_{a_n} 2\sqrt{w_{a_{n-1}} \ldots 2\sqrt{w_{a_2} w_{a_1}}}} = c \times w_{a_n}^{1/2} w_{a_{n-1}}^{1/4} \ldots w_{a_2}^{1/2^n} w_{a_2}^{1/2^n}$$

式中 c 为一常数。
取对数利用排序不等式易得：要使上式取最小值，则

$$w_{a_n} \leq w_{a_{n-1}} \leq \ldots \leq w_{a_2} \leq w_{a_1}$$

注意最后一个不等号并非必要条件，但将第一堆和第二堆相互交换没有任何意义，不失一般性，我们给出这样一个不等号。
Part 2 若在合并过程中，存在 a, b, c, d 四个权重的果子，合并方式为先合并 a, b，再合并 c, d，最后将两堆合并，则最后代价为 $2\sqrt{2\sqrt{ab}2\sqrt{cd}} = 4(abcd)^{1/4}$，这个代价与合并顺序无任何关系，不失一般性，假设 $a \leq b \leq c \leq d$。若按之前的办法，则代价为 $2\sqrt{a \times 2\sqrt{b \times 2\sqrt{cd}}} = 2^{1.75} a^{1/2} b^{1/4} c^{1/8} d^{1/8}$，两代价相除，有

$$\frac{2^2 (abcd)^{0.25}}{2^{1.75} a^{0.5} b^{0.25} c^{0.125} d^{0.125}} = 2^{0.25} \frac{(cd)^{0.125}}{a^{0.25}} \geq (\frac{\sqrt{cd}}{a})^{0.25} \geq 1$$

因此最优解合并结构中不存在两个已合并的子堆合并成一个大堆的情况，即一定是每一次向同一堆果子添加，而 Part 1 证明了以这种方式，从最大到最小合并为最优顺序。得证。□

跳跃游戏

问题：一个非负整数数组 a，初始位于 0，数组每个元素代表该位置可跳跃的最大长度，求到达终点的最少步数（保证可以到达终点）

跳跃游戏

问题：一个非负整数数组 a，初始位于 0，数组每个元素代表该位置可跳跃的最大长度，求到达终点的最少步数（保证可以到达终点）
这一步跳哪？当然是看下一步跳哪更远，才能离终点更近啊！但是好像不太够，那如果看下下一步跳得更远呢？不妨证明一下：

跳跃游戏

问题：一个非负整数数组 a，初始位于 0，数组每个元素代表该位置可跳跃的最大长度，求到达终点的最少步数（保证可以到达终点）
这一步跳哪？当然是看下一步跳哪更远，才能离终点更近啊！但是好像不太够，那如果看下下一步跳得更远呢？不妨证明一下：
证明：
设当前位置为 i，则下一步可选范围为 $[i+1, i+2, \ldots, i+a[i]]$，则下下一步范围为 $[i+2, i+3, \ldots, \max_{i<k\le a[i]}(i+k+a[i+k])]$。该范围包含了下下一步的所有可能取值，由于该算法选择 $\arg\max_{i<k\le a[i]}(i+k+a[i+k])$，则下一步的任意取值 k'，则任何 k' 的下一步，k 都能取到。因此走 k 的状态包含了其它任意点的状态，而 k 能达到最远，因此最优。□

加油站

问题：一个环路上 n 个加油站，第 i 个加油站可加 a[i] 升汽油，第 i 个加油站去下一个加油站需要 b[i] 升汽油。问是否有一个加油站，使得从这里出发，能绕环路一周。

加油站

问题：一个环路上 n 个加油站，第 i 个加油站可加 a[i] 升汽油，第 i 个加油站去下一个加油站需要 b[i] 升汽油。问是否有一个加油站，使得从这里出发，能绕环路一周。
这个问题的关键是搜索几遍所有加油站？不妨想想，如果从 x 刚好到 y 而不能去 $y+1$，那 x 到 y 之间任意一点，因为没有之前油的储备所以肯定去不了 $y+1$，因此只需要整个环搜索一遍。如果搜索次数大于 n 就不行。

加油站

证明：
若当前 x 能达到的最远加油站为 y，假设存在 x 与 y 间一座加油站 z，使得 z 能到达加油站 $y+1$。则：

$$\sum_{i=z}^{j} b[i] < \sum_{i=z}^{j} a[i],\ \forall j \in (z, y+1]$$

又因为 x 能到达加油站 z，则有：

$$\sum_{i=x}^{z} b[i] < \sum_{i=x}^{z} a[i]$$

两式相加，取 $j = y+1$ 有：

$$\sum_{i=x}^{y+1} b[i] < \sum_{i=x}^{y+1} a[i]$$

国王游戏

问题：n 个人，两只手，每只手上一正整数，从左往右站一排，最前面再额外站一个国王，国王两只手也有数字，每人得分为站在该人前面的所有人的左手上的数的乘积除以他自己右手上的数向下取整的结果。问得分最多的人最少获得多少金币？

国王游戏

问题：n 个人，两只手，每只手上一正整数，从左往右站一排，最前面再额外站一个国王，国王两只手也有数字，每人得分为站在该人前面的所有人的左手上的数的乘积除以他自己右手上的数向下取整的结果。问得分最多的人最少获得多少金币？

这个贪心策略并不明显，先考虑两人情况：设第 i 个人（国王为 0）左右手分别为 a[i]，b[i]。则：

因为第一个人更前，因此：

$$\max(a[0]/b[1], a[0]a[1]/b[2]) < \max(a[0]/b[2], a[0]a[2]/b[1])$$

又因为所有数是正整数，所以

$$1/b[1] \le a[2]/b[1], 1/b[2] \le a[1]/b[2]$$

看到乘除，其实也就两种，要么左右手乘，要么左右手除，找个数据试试发现好像是乘，那就反证法吧：

假设 $a[0]a[1]/b[2] \ge a[0]a[2]/b[1]$，则有

$$\max(a[0]/b[1], a[0]a[1]/b[2]) \le \max(a[0]/b[2], a[0]a[2]/b[1])$$

因此 $a[0]a[1]/b[2] < a[0]a[2]/b[1]$，即 $a[1]b[1] < a[2]b[2]$。

对于 n 个人的证明，只需要把 $a[0]$ 换成前 $i-1$ 个人的左手乘积即可，转化成扰动法的证明。

Thank you

Thank you for listening!

Q&A

Questions?