告别死循环：二分算法的“红蓝区域法”

讲师：李俊飞 (Feijoa_Li)

写在前面

很多同学在写二分时，经常纠结这几个问题：

- mid 到底是加一还是减一？
- 循环条件是 $l < r$ 还是 $l \leq r$？
- 最后的答案到底是 l 还是 r？

今天我们要介绍的**“双开区间（红蓝区域）法”**，能够让你不再纠结这些细节，用一种极其直观的物理模型，一劳永逸地解决所有二分问题。

目录

1 二分算法核心：红蓝区域法与双开区间模型

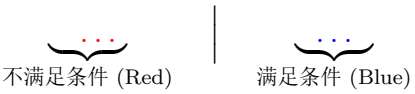
本节将介绍一种在信奥竞赛中极其实用的二分思维模型。它抛弃了复杂的边界判断，将二分问题抽象为物理上的“领地夹逼”过程。

1.1 直观模型：红蓝颜色的对抗

在做二分题时，请摒弃“我在找数字 5”这种具体数值的思维，转而建立**“寻找性质分界线”**的全局观。

假设我们面对一个有序序列或数轴区间。尽管未知具体的数值，但我们可以根据题目给出的判断条件（Check 函数），将整个定义域划分为两种颜色的领地：

- 红色区域 **(Red)**：左侧**不满足**条件的位置，视作红色（False/0）。
- 蓝色区域 **(Blue)**：右侧**满足**条件的位置，视作蓝色（True/1）。



我们的任务变得非常单纯：**定位红蓝两种颜色交界的那条唯一“缝隙”。**

1.2 执行流程：双指针哨兵机制

为了精准捕获这条分界线，我们派出两名“哨兵”指针——l（红军大将）和 r（蓝军大将），采用**双开区间** (l, r) 的策略进行搜索。

1. 定义领地（初始化）

为了防止越界并覆盖所有可能解，初始化时指针不应站在具体的格子上，而应站在**数组范围之外**：

- l **(Left)**：初始位于第一个元素的**左侧**（虚拟下标 -1）。它承诺：“我所站立及我左侧的地方，全都是红色的。”
- r **(Right)**：初始位于最后一个元素的**右侧**（虚拟下标 N）。它承诺：“我所站立及我右侧的地方，全都是蓝色的。”

2. 领地扩张（迭代过程）

只要 l 和 r 之间还有未探索的未知区域（即 $l+1<r$），探索就继续：

1. 计算中点 mid。
2. 派侦察兵查看 mid 的颜色：
   - 若 mid 是蓝色：说明 mid 及其右侧皆为蓝军领地。于是 r 左移至 mid 处守卫新边界。
   - 若 mid 是红色：说明 mid 及其左侧皆为红军领地。于是 l 右移至 mid 处守卫新边界。

核心原则： l 永远只在红色区域移动，r 永远只在蓝色区域移动，绝不越界。

3. 完美的结局（循环终止）

当 l 和 r 终于紧挨在一起（即 $l+1==r$）时，循环结束。此时的状态是唯一且确定的：

- l 指向**最后一个红色元素**。
- r 指向**第一个蓝色元素**。
- 分界线就在 l 和 r 之间。

【答案提取指南】

题目问“最大的不满足数” $\rightarrow$ 输出 l

题目问“最小的满足数” $\rightarrow$ 输出 r

无需纠结 $mid\pm1$，答案天然就在 l 或 r 手中。

1.3 数值细节：中点计算的避坑指南

在计算 mid 时，教科书常见的 $(l+r)/2$ 写法存在隐患。建议养成使用**防溢出写法**的肌肉记忆。

推荐公式： $mid=l+(r-l)/2$

该公式包含两层重要的工程意义：

1. **防止整型溢出 (Integer Overflow)：** 若 l,r 均为接近 2×10^9 的大整数，直接相加会造成数据溢出（变为负数），导致程序崩溃。先减后加则永远安全。
2. **向下的引力 (Floor Division)：** C++ 的整数除法默认向下取整。你可以理解为 mid 总是受到 l 的“引力”。在 l 和 r 相邻之前，mid 永远不会触碰到 r，这从数学上彻底杜绝了死循环的可能性。

2 进阶应用：二分答案——“猜”出来的最优解

二分答案是算法竞赛中二分思想最高频的考点，也是一种**思维方式的逆转**。

2.1 从“直接求解”到“试探验证”

很多时候，题目会要求我们求一个“最优值”（例如：求最少需要多少时间、求最大的最小距离等）。

- **正向思维（求解）：** 直接计算出最优解 X。这往往非常困难，可能涉及复杂的动态规划或枚举，时间复杂度难以接受。
- **逆向思维（验证）：** 我们不去直接算 X，而是先**猜**一个答案 mid，然后问自己：“如果答案是 mid，在这个限制下，我能满足题目的要求吗？”

这种方法被称为**“答案的判定性转化”**。只要“验证一个答案是否可行”比“直接求出答案”要容易得多（例如验证通常只需要 $O(N)$ 的贪心），且答案具有**单调性**，我们就可以用二分来地毯式搜索最优解。

2.2 两类经典模型：最大化与最小化

二分答案题目通常有着非常鲜明的提问方式，对应着红蓝模型中不同的寻找方向：

2.2.1 1. 最小值最大化 (Maximize the Minimum)

场景： “求 N 个点之间最近距离的最大值”、“求让最穷的人分到的钱尽可能多”。

单调性分析： 假设我们验证的数值为 x（例如距离）。

- 如果 x 满足条件（即距离为 x 时能放得下），那么比 x 更小的值（$x-1, x-2\dots$）肯定更能放得下（条件更宽松）。
- 我们的目标是找到**最后一个满足条件**的值。

红蓝分布：

$$\underbrace{\text{可行 (True)} \dots \text{可行}}_{\text{小数值}} \mid \underbrace{\text{不可行} \dots \text{不可行 (False)}}_{\text{大数值}}$$

搜索目标： 蓝色区域的右边界（r 或 l，取决于具体写法，双开区间法中取 l）。

2.2.2 2. 最大值最小化 (Minimize the Maximum)

场景： “求耗时最长的任务所需的最少时间”、“求段和最大值的最小值”。

单调性分析： 假设我们验证的数值为 x（例如时间限制）。

- 如果 x 满足条件（即限时 x 秒能做完），那么比 x 更大的值（$x+1, x+2\ldots$）肯定更能做完（时间更充裕）。
- 我们的目标是找到**第一个满足条件**的值。

红蓝分布：

$$\underbrace{\text{不可行 (False)} \ \ldots \ \text{不可行}}_{\text{小数值}} \ \Big| \ \underbrace{\text{可行} \ \ldots \ \text{可行 (True)}}_{\text{大数值}}$$

搜索目标： 蓝色区域的左边界（r 或 l，取决于具体写法，双开区间法中取 r）。

2.3 解题四步走

遇到这就两类问题，按以下步骤思考：

1. **确定解空间：** 答案的可能的最小值 L 和最大值 R 是多少？
2. **设计 Check 函数：** 给你一个固定的值 mid，你能否在 O(能过) 时间内判断它是否合法？（通常配合贪心算法）。
3. **画出红蓝条：** 判断单调性方向，大值可行还是小值可行？
4. **套用模板：** 使用双开区间模板，根据 Check 的结果收缩边界。

3 数学进阶：实数域上的二分

当我们的战场从离散的整数点转移到连续的实数轴时（例如计算几何中的线段交点、物理学中的抛物线时间计算），二分的“红蓝边界”依然存在，但搜索策略需要进行重要的调整。

3.1 精度与效率的博弈

在实数域中，我们通常寻找的是一个近似解。传统教科书常教授 `while (r - l > eps)` 的写法，但在竞赛实战中，这往往是一个陷阱。

3.1.1 浮点数的陷阱：为什么 eps 不靠谱？

设定一个固定的精度阈值（如 ‘eps = 1e-7‘）存在两个严重隐患：

1. **精度稀疏问题 (Precision Loss)**：计算机中的浮点数分布不是均匀的。随着数值绝对值的增大，可表示的浮点数之间的间隙（Gap）也会变大。

   例如： 当答案高达 10^{12} 时，相邻两个 ‘double‘ 类型的数值差可能已经超过了你设定的 ‘1e-7‘。这意味着 l 和 r 即使紧挨着，差值也永远大于 ‘eps‘，导致程序陷入**死循环**。

2. **收敛速度的不确定性**：在某些构造的极端数据下，区间收缩到 ‘eps‘ 附近所需的迭代次数可能超出预期，导致超时（TLE）。

3.2 黄金策略：固定迭代次数法

为了彻底规避精度死循环和超时问题，我强烈建议在实数二分中采用**“固定循环次数”**的策略。

我们知道，每一次二分都会将区间长度减半。假设初始区间长度为 L，经过 K 次迭代后，区间长度变为：

$$L_{final} = L \times \left(\frac{1}{2}\right)^K$$

为什么选 100 次？ 若我们循环 100 次，精度缩减倍率为 $2^{-100} \approx 10^{-30}$。

- 哪怕初始区间是整个宇宙的大小（10^{26} 米），100 次二分后误差也小于一个原子核。
- 这个精度远远超过了 ‘double‘ 甚至 ‘long double‘ 的有效位数。

【最佳实践指南】

在处理实数二分时，请遵循以下步骤：

1. **摒弃 While**：不要写 ‘while (r - l ¿ eps)‘。
2. **使用 For 循环**：直接根据题目精度要求，写一个固定次数的循环。
   - 一般精度要求：循环 **100 次**。
   - 极高精度要求（Long Double）：循环 **200 次**。
3. **无脑收缩**：在循环体内，直接令 $mid = (l + r)/2$。如果 mid 满足条件，则 $l = mid$（或 $r = mid$），无需考虑 $+1/-1$ 的整数边界问题。

这种写法时间复杂度恒定为 $O(100)$，在保证绝对精度的同时，彻底杜绝了死循环的风险。

4 进阶武器：三分查找与单峰极值

当函数的性质不再是简单的“单调”（一路升或一路降），而是变成了**“单峰”（Unimodal）**——即先增后减（像一座山峰）或先减后增（像一个山谷）时，二分法就失效了。

此时，我们需要引入更高级的策略——**三分查找 (Ternary Search)**。

4.1 双探针逻辑：如何在黑暗中爬山？

假设我们要寻找函数 $f(x)$ 的**最大值**（山顶），且已知 $f(x)$ 是开口向下的单峰函数。

二分法之所以行不通，是因为只看中间一个点，我们不知道自己是在“上坡”还是“下坡”（除非求导，但离散函数无法求导）。因此，我们需要**两个探针**来测定地势的走向。

我们在区间 (l, r) 内部设立两个采样点 m_1 和 m_2，将区间大致三等分：

$$m_1 = l + \frac{r-l}{3}, \quad m_2 = r - \frac{r-l}{3}$$

显然有位置关系：$l < m_1 < m_2 < r$。

通过比较这两个探针的高度 $f(m_1)$ 和 $f(m_2)$，我们可以安全地**“砍掉”**一段绝对不可能包含山顶的区间：

- **情形一：** $f(m_1) < f(m_2)$
  地势分析： 右边的 m_2 比左边的 m_1 高。 **推论：** 既然我们要找最高点，且山只有一个顶，那么最高点绝不可能在 m_1 的左侧（否则 m_1 处于下坡阶段，但这违背了m_2更高的事实，或意味着有两个峰）。

  $\Rightarrow$ **操作：** 极值一定在 (m_1, r) 中。安全舍弃左侧区间，令 $l = m_1$。

- **情形二：** $f(m_1) > f(m_2)$
  地势分析： 左边的 m_1 比右边的 m_2 高。 **推论：** 同理，最高点绝不可能在 m_2 的右侧（否则 m_2 之后还在升高，那 m_1 到 m_2 就成了下坡，形成凹陷，违背单峰性质）。

  $\Rightarrow$ **操作：** 极值一定在 (l, m_2) 中。安全舍弃右侧区间，令 $r = m_2$。

4.2 工程实现：离散域的“暴力收尾”

在实数域上三分很简单（使用固定循环次数即可），但在**整数域**上三分极其容易写挂。

当 l 和 r 距离很近时（例如 $r-l<3$），m_1 和 m_2 可能会计算重合，或者陷入死循环。为了在考场上万无一失，建议采用**“三分缩范围 + 暴力定答案”**的混合策略：

1. **主循环收缩：** 当区间长度比较大时（例如 $l+2<r$），正常执行三分逻辑，不断缩小范围。
2. **暴力扫尾：** 当循环结束时，区间内只剩下 3 到 5 个整数点。此时不要再纠结边界了，直接一个 `for` 循环遍历 $[l,r]$ 这一小段，算出最大值即可。

这种写法既保留了 $O(\log N)$ 的高效，又完全规避了小区间内的数学边界陷阱。

5 0/1 分数规划：化除为减的魔法

这是二分思想在代数领域最惊艳的应用之一。它解决了一类让人头疼的“最优比率”问题。

5.1 问题的本质

假设你有一堆物品，每个物品有两个属性：价值 a_i 和代价 b_i。我们需要从中选出一组物品（集合 S），在满足题目特定约束（比如选 k 个，或者连通）的前提下，让总价值与总代价的比值最大：

$$\text{Maximize} \quad V=\frac{\sum_{i\in S}a_i}{\sum_{i\in S}b_i} \tag{1}$$

直接求这个比率非常困难，因为分子和分母都会随着选择变化，且互相牵制。

5.2 核心推导：不等式的变形

我们换个思路：与其直接求最大值，不如去**猜**这个比率是多少。

假设我们猜测最终答案能达到 mid。如果存在一种选择方案 S，使得比率 $\geq mid$，那么数学上意味着：

$$\frac{\sum a_i}{\sum b_i}\geq mid$$

我们将分母乘过去（注意 b_i 通常为正，不改变符号）：

$$\sum a_i \geq mid\cdot\sum b_i$$

移项，将含 mid 的项挪到左边：

$$\sum a_i - \sum (mid \cdot b_i) \geq 0$$

合并求和符号：

$$\sum_{i \in S} (a_i - mid \cdot b_i) \geq 0 \tag{2}$$

5.3 算法流程：新权值的诞生

通过上面的变形，我们发现了一个惊人的事实：

如果我们把每个物品的新权值定义为 $w_i = a_i - mid \cdot b_i$，那么问题就变成了：能否选出一组物品，使得它们的新权值之和非负？

这彻底消除了讨厌的除法！现在的解题套路变成了标准的三步走：

1. **二分答案**：在实数域上二分比率 mid（建议迭代 100 次）。
2. **重置权值**：在 ‘check(mid)‘ 函数中，将所有物品的权值重置为 $w_i = a_i - mid \cdot b_i$。
3. **判定可行性**：
   - 如果题目要求选 K 个：那就贪心地选 w_i 最大的 K 个，看和是否 ≥ 0。
   - 如果题目要求生成树：那就用 w_i 跑最大生成树，看树权是否 ≥ 0。
   - 如果题目要求找环：那就看图中是否存在正环。

   如果最大权值和 ≥ 0，说明 mid 定小了（或者刚好），答案在右边；否则答案在左边。

5.4 总结

0/1 分数规划的本质，就是通过二分答案，将非线性的**“比率最大化”**问题，转化为线性的**“判定权值和非负”**问题。掌握了这个转化，你就能把以前学过的所有算法（贪心、DP、最短路）都套上“最优比率”的外壳。

6 高阶模型：维度变换与 WQS 二分

当题目要求“恰好选 K 个物品”时，朴素的 DP 往往需要一维状态来记录个数，导致复杂度飙升到 $O(N^2)$。

WQS 二分（又称 Aliens Trick / 带权二分）提供了一种降维打击的思路：通过引入一个“隐形的手”，把刚性的数量限制转化为弹性的价格调节。

6.1 直观理解：用“价格”控制“销量”

6.1.1 硬约束 vs. 软调节

假设你在经营一家商店，题目要求你**“必须卖出恰好 K 个苹果”**以获得最大利润。这很难，因为你得时刻盯着销量。

但如果你换一种思路：

1. 取消“恰好 K 个”的限制，允许随便卖（这就变成了简单的贪心或 $O(N)$ DP）。
2. 但是，每卖出一个苹果，必须向系统缴纳 λ **元的“手续费”**。

这就引入了博弈：

- 如果手续费 λ **太贵**（例如 100 万/个），为了利润最大化，你可能一个都不卖（选出的个数 $< K$）。
- 如果手续费 λ **太便宜**（甚至倒贴钱），你会疯狂卖苹果（选出的个数 $> K$）。
- **中间必然存在一个神奇的价格** λ_{mid}，使得你在追求利润最大化的无拘无束的决策中，**“自愿”**且**“恰好”**卖出了 K 个苹果！

这就是 WQS 二分的本质：**二分这个手续费 λ，直到最优策略自动选出 K 个物品。**

6.2 适用前提：边际效应递减（凸性）

不是所有问题都能用这招。WQS 二分生效的前提是问题的解具有**凸性（Convexity）**。

通俗地说，就是**“边际效应递减”**：

> 当你选第 1 个物品时，获利巨大；选第 2 个时，获利稍少……选第 100 个时，获利微乎其微。

如果我们画出图表：横坐标是“选择物品的个数 x”，纵坐标是“最大收益 $f(x)$”。如果 $f(x)$ 的图像是一个**上凸的形状**（斜率逐渐变小），那么我们就可以用 WQS 二分。

6.3 几何意义：斜率切线法

6.3.1 为什么要减去 $\lambda \cdot x$？

我们在无限制的情况下求解的最大利润，实际上是在求：

$$g(\lambda) = \max_{x}\{f(x) - \lambda \cdot x\}$$

移项变形一下：

$$f(x) = \lambda \cdot x + g(\lambda)$$

这看起来像什么？$y = kx + b$!

- $f(x)$ 是 y 坐标（总收益）。
- x 是 x 坐标（选择个数）。
- λ 是**斜率**。
- $g(\lambda)$ 是**截距**。

当我们固定斜率 λ 并试图最大化截距 $g(\lambda)$ 时，几何上等价于：**用一条斜率为 λ 的直线去切 $f(x)$ 的图像。** 切点对应的横坐标，就是在该手续费下的最优选择个数。

6.4 解题三步走

1. **二分斜率 λ**：范围通常在 $[-\text{inf}, \text{inf}]$。
2. **带权 DP / 贪心 (Check 函数)**：在 ‘check(mid)‘ 中，忽略个数 K 的限制，但把每个物品的权值减去 mid（手续费）。跑一遍基础算法，算出此时的**最优总价值** val 和**最优选取的物品数** cnt。
3. **调整边界与还原答案：**
   - 如果 $cnt \geq K$：说明手续费太便宜了，或者刚好。我们记录答案，并尝试增加手续费（让 λ 变大，cnt 变小），即 $l = mid + 1$（或根据写法调整）。
   - 如果 $cnt < K$：说明手续费太贵了，需要降价。

   最终答案还原公式：

   $$\text{真实答案} = \text{带权最优值} + K \cdot \text{最终的手续费}$$

   即：$f(K) = g(\lambda_{ans}) + K \cdot \lambda_{ans}$。

7 整体二分（Parallel Binary Search）：批发式的高效分治

这是二分算法的最终形态。它解决的是这样一种困境：

你有 10 万个询问，每个询问都可以用二分答案解决。但是，每次 'check' 函数都要跑一遍 $O(N)$ 的操作。如果一个个做（零售），总复杂度是 $O(Q \cdot N \cdot \log V)$，直接超时。

整体二分的核心思想是**“批发”**：既然所有询问都在同一个答案值域 $[Min, Max]$ 内找答案，不如把它们捆在一起，通过一次遍历，同时更新所有人的进度。

7.1 核心逻辑：数据流的分拣

我们要定义一个递归过程 `solve(L, R, QueryList)`，含义是：“我知道这堆询问 *QueryList* 的最终答案，一定都在数值区间 $[L, R]$ 里，请帮我进一步缩小范围。”

这个过程就像在一个巨大的**分拣工厂**里工作：

1. **设定中点（切割）：** 取值域中点 $mid = (L + R)/2$。我们将把所有操作和询问按照“是否 $\leq mid$”这一标准分为两类。
2. **发放物资（贡献计算）：** 遍历时间轴上所有的修改操作（例如“加入一个数 x”）：
   - 如果 $x \leq mid$：说明这个数属于“左半边”的较小值。它会对当前的判定产生影响（比如贡献了 1 个排名）。我们将它加入数据结构（如树状数组）。
   - 如果 $x > mid$：说明这个数太大了，暂时不考虑，之后会归入右半边递归。
3. **客户分流（询问判定）：** 遍历 `QueryList` 中的每一个询问（例如“查区间第 K 小”）：查询数据结构，看在只考虑 $\leq mid$ 的数值时，当前区间里有多少个数（记为 cnt）。
   - **左侧分流 (Happy List)**：若 $cnt \geq K$：说明答案 $\leq mid$（光靠左半边的数就够凑齐 K 个了）。 $\Rightarrow$ 该询问归入 `left_list`，继续在 $[L, mid]$ 里找。
   - **右侧分流 (Hungry List)**：若 $cnt < K$：说明答案 $> mid$（左半边的数不够，还需要在右半边再找 $K - cnt$ 个）。 $\Rightarrow$ **关键操作：** 令 $K \leftarrow K - cnt$（扣除已有的贡献），然后该询问归入 `right_list`，继续在 $[mid + 1, R]$ 里找。
4. **还原现场（Rollback）：** **这是极其重要的一步！** 在进入下一层递归之前，必须把步骤 2 中加入数据结构的操作全部撤销（清空树状数组）。因为下一层递归是独立的，不能受当前层的残余影响。
5. **递归分治：** 分别调用 `solve(L, mid, left_list)` 和 `solve(mid+1, R, right_list)`。

7.2 复杂度分析：为什么它快？

虽然看起来很复杂，但我们算一算账：

- **深度：** 答案的值域为 V，递归层数只有 $\log V$ 层。
- **广度：** 在每一层递归中，每个修改操作和每个询问只会被处理一次（要么去左边，要么去右边，类似归并排序）。

配合 $O(\log N)$ 的数据结构，总的时间复杂度为 $O((N+Q)\log N \log V)$。这比单独二分快了整整 N 倍！

【总结：整体二分的三个特征】

1. **答案可二分：**单个询问具有单调性。
2. **修改独立：**操作之间互不影响，可以按值域划分。
3. **贡献可加：**判定时可以把一部分贡献先扣除（如 $K \leftarrow K - cnt$），去子问题里找剩下的。

8 二分核心知识点全景汇总

层次	知识点	核心逻辑	典型应用
基础	整数二分	双开边界 $l+1<r$	有序数组定位
基础	实数二分	固定迭代 100 次	几何/计算几何
核心	二分答案	判定性转化	资源分配/最大最小化
进阶	三分查找	三等分点采样对比	单峰函数求极值
进阶	分数规划	权值重定义为 $a-kb$	效益比最优化
高阶	WQS 二分	凸优化与代价抵消	带约束的最优选法
高阶	整体二分	全局询问分治	动态区间第 K 大

9 结语：二分的灵魂是“舍弃”

二分查找之所以高效，是因为它每一步都在果断地舍弃一半的解空间。在学习过程中，大家应重点体会这种“分而治之”的确定性。希望本讲义能帮助大家彻底掌握二分算法，在竞赛中做到边界不乱、逻辑清爽。