浅谈二分策略的应用

华东师大二附中 杨俊

【摘要】本文着重讨论三种不同类型的二分问题，意在加深大家对二分的认识。它们所考虑的对象从一般有序序列，到退化了的有序序列，最后到无序序列。事实上它们也正代表了二分策略的三种不同应用。

【关键字】二分、序、应用

【正文】

“二分”，相信这个词大家都再熟悉不过了。二分是一种筛选的法则，它源于一个很简单的想法——在最坏情况下排除尽可能多的干扰，以尽可能快地求得目标。

二分算法的高效，源于它对信息的充分利用，尽可能去除冗余，减少不必要的计算，以极大化算法效率。事实上许多二分问题都可以用判定树或其它一些定理来证明，它达到了问题复杂度的下界。

尽管二分思想本身很简单，但它的扩展性之强、应用面之广，或许仍是我们所未预见的。大家也看到，近年来各类竞赛试题中，二分思想的应用不乏令人眼前一亮的例子。下面是作者归纳的二分思想的三种不同类型的应用，希望能让读者有所收获。

类型一：二分查找——应用于一般有序序列

申明：这里所指的有序序列，并不局限于我们通常所指的，按从小到大或是从大到小排好序的序列。它仅包含两层意思：第一，它是一个**序列**，一维的；第二，该序列是**有序**的，即序列中的任意两个元素都是可以比较的。也就是拥有我们平时所说的全序关系。

虽说二分查找大家都再熟悉不过了，但这里还是先简要地回顾一下二分查找的一般实现过程：

（1） 确定待查找元素所在范围
（2） 选择一个在该范围内的某元素作为基准
（3） 将待查找元素的关键字与基准元素的关键字作比较，并确定待查找元素新的更精确的范围
（4） 如果新确定的范围足够精确，输出结果；否则转至（2）

让我们看一个经典问题——**顺序统计问题**

[问题描述]

给定一个由 n 个不同的数组成的集合 S，求其中第 i 小的元素。

[分析]

相信大家对这个问题都很熟悉，让我们回顾一下二分查找是如何应用于该问题上的。

（1） 确定待查找元素在 S 中

（2） 在 n 个元素中随机取出一个记为 x，将 x 作基准

（3） 设 S 中比元素 x 小的有 p 个。

如果 i<p，表示我们所要寻找的元素比 x 小，我们就将范围确定为 S 中比 x 小的元素，求该范围内第 i 个元素；

如果 i>p，表示我们所要寻找的元素比 x 大，我们就将范围确定为 S 中比 x 大的元素，求该范围内第 i-p-1 个元素；

如果 i=p，表示第 i 小的元素就是 x。

（4） 如果找出 x，输出；否则转至（2）

因为 x 是随机选出的，由简单的概率分析，可得算法的复杂度期望值为 O(n)。

[小结]

举这个例子，是想提醒大家两点：

第一，不要想当然认为二分查找就一定与 logn 有关。算法中的第（3）步，即我们通常所说的“分”并不要求每一次都必须在 O(1)时间进行，“分”可以是建立在对序列的有效处理之上（比如上面这个例子中使用了类似于快速排序中的集合分割）。

第二，二分算法的“分”并不要求每次都必须平均（因为有时候可能很难做到这一点），只要不是每次都不平均就已经可以产生高效的算法了，这样也给使用随机化算法带来了契机。

近年来由于交互式试题的出现，也给予二分查找更多活力。相当多的二分查找问题都是以**交互式**试题的形式给出的。比如说，就上面这个例子，摇身一变就成了一道交互式的**中等硬度**问题（IOI2000）。两个题目如出一辙：你问第 i 小的，我问第(N+1)/2 小的；解法当然也就依样画葫芦：你用随机取出的 x，依照与 x 大小关系分成两段，我就随机取出 x,y，依照与 x,y 大小关系分成三段；你的复杂度期望是 O(n)，我的询问次数的期望也是 O(N)。（具体细节这里不再展开，有兴趣的同学可以参考前辈的解题报告[1]）

类型二：二分枚举——应用于退化了的有序序列

二分策略并不总是应用于上述这样显式的有序序列中，它可能借助于问题某种潜在的关联性，用于一些隐含的退化了的有序序列中。与先前介绍的二分查找相比，最大的区别在于这里的二分在判断选择哪一个部分递归调用时没有比较运算。

我们还是先看一个问题——**BTP 职业网球赛**（USACO contest dec02）

[问题描述]

有 N 头奶牛（N 是 2^K），都是网球高手，每头奶牛都有一个 BTP 排名（恰好为 1—N）。下周将要进行一场淘汰赛，N 头奶牛分成 N/2 组，每组两头奶牛比赛，决出 N/2 位胜者；N/2 位胜者继而分成 N/4 组比赛……直至剩下一头牛是冠军。

比赛既要讲求实力，又要考虑到运气。如果两头奶牛的 BTP 排名相差大于给定整数 K，则排名靠前的奶牛总是赢排名靠后的；否则，双方都有可能获胜。现在观众们想知道，哪头奶牛是所有可能成为冠军的牛中排名最后的，并要求你列举出一个可能的比赛安排使该奶牛获胜。（限制 N≤65536）

[分析]

由于 N 很大，我们猜想可以通过贪心方法解决。

我们希望排名靠前的选手总是“爆冷”输给排名靠后的选手。于是我们让 1 输给 K+1、

[1] 2000-2002 集训队论文《中等硬度解题报告》高岳

2 输给 K+2……每一轮中每一局总是选择未比赛的排名最前的选手，输给一个排名最靠后的选手（如果有的话）。

但我们很容易找到反例，例如：N=8, K=2，由上述贪心方法我们得到对阵方式结果为 4，见图 BTP-1（其中 X→Y 表示 X 战胜 Y）。

3→1　4→2　7→5　8→6
4→3　　8→7
4→8

图 BTP-1

6→7　5→3　4→8　2→1
6→5　　4→2
6→4

图 BTP-2

但最优解为 6，见图 BTP-2。究其原因，因为我们不知道最优解是多少，所以我们是在盲目地贪心。事实上，最优解的 6 在第一轮就被我们淘汰了，当然就得不到最优解喽！

要想知道最优解？枚举！同时考虑到一个很显然的结论——**如果排名为 X 的选手最终获胜，那么排名在 X 前的选手也可以获胜**。

显然归显然，证明它我们还需要动一点小脑筋。假设排名 X 的选手最终获胜，我们通过对该对战方式的局部修改，构造出一种新的对战方式使任意排名 Y<X 的选手获胜：在 X 最终获胜的对战方式中，假设 Y 是被 Z 击败。如果 Z≠X，由 X>Y，可知 Z 一定也能击败 X，且同一轮及此后 X 所击败的选手都能被 Y 击败，所以我们只需要在此轮让 Z 击败 X，并把之后所有对战中的 X 都改成 Y 即可；如果 Z=X，由 X>Y，我们就让 Y 击败 X，同样把之后对战中的 X 都改成 Y，则最后获胜的也是 Y。

因此，我们就可以用二分枚举最终获胜的 X，大大提高了算法效率。

现在的问题是，如果我们知道最终获胜的是 X，我们能否很快构造出一种对战方式或是证明不存在吗？可以，我们仍旧使用贪心，不过因为我们知道最终谁获胜，所以我们采用倒推。

每一轮，我们都让已有选手去战胜一个排名最靠前的还未出现的选手，由该方法产生的对战方式就如图 BTP-2 所描述那样。至于最靠前的未出现选手如何找，我们可以采用静态排序二叉树实现，这里不再展开，读者可以自己考虑。

可以证明这样贪心是正确的（证明方法同前面的证明类似，这里也不再重复）。整个问题可以在 $O(N\log^2 N)$时间完成。

[小结]

对于这类需要二分枚举的问题，其实算法的根本是在一个隐含的退化了的有序序列中进行二分查找，只是这个序列仅含有 0 和 1 两种值。上例中当 X<Ans，A[X]=0；当 X≥Ans，A[X]=1。而我们所寻找的就是这两种值的分界点。有了分界点，就有了最优值，也就有了原问题的解。

让我们再看一个问题——**奖章分发**（ACM/ICPC CERC 2004）

[问题描述]

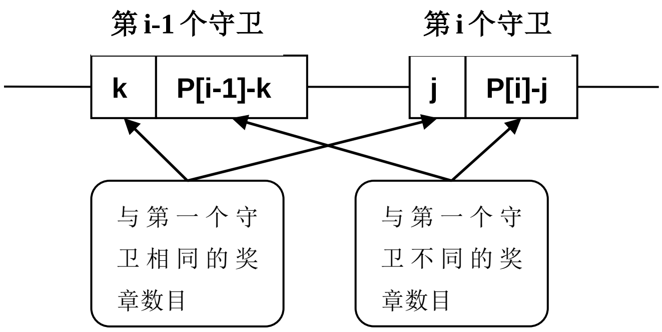
有一个环形的围墙，围墙上有一些塔，每个塔中有一个守卫。现在要发给每个守卫一些奖章，第 i 个守卫需要 P[i]个奖章。每个守卫自己的奖章必须都是不同种类的，而且相邻的两个守卫得到的奖章也不能有任何一个相同。问至少应准备多少种不同的奖章。（限制：2≤n≤10000，1≤P[i]≤100000）

[分析]

假如围墙不是环形的，我们很容易用贪心算法解决：每次发给守卫尽可能多的已准备的奖章，如果不够就再新准备若干种以满足需要。但现在围墙是环形的，最后一个守卫与第一个守卫也不能有重复的奖章，这就是问题的难点。

于是我们尝试将这一难点引入状态，用动态规划求解（动态规划是求解最优性问题的一种常用方法）。

令 a[i,j]表示前 i 个守卫已安排妥当，且第 i 个守卫有 j 个奖章与第一个守卫相同，则除第一个守卫已有的 P[1]种奖章外，至少还需要的奖章种数。



如图，假设第 i-1 个守卫有 k 个奖章与第一个守卫相同，由于这 k 个奖章与第 i 个守卫的 j 个奖章必须互不相同，易知 j+k≤P[1]；又由于第 i-1 个守卫的另外 P[i-1]-k 个奖章必须与第 i 个守卫的另外 P[i]-j 个奖章不同，设需要增加 X 种奖章，则

a[i-1,k]+X ≥ (P[i]-j)+(P[i-1]-k)

得到 **X ≥ (P[i]-j)+(P[i-1]-k) - a[i-1,k]**

于是我们有

$$\begin{aligned} a[i,j] &= \min_{0\le k\le P[i-1],k\le P[1]-j}\{a[i-1,k]+\max\{0,(P[i]-j)+(P[i-1]-k)-a[i-1,k]\}\} \\ &= \min_{0\le k\le P[i-1],k\le P[1]-j}\{\max\{a[i-1,k],P[i]+P[i-1]-j-k\}\} \end{aligned}$$

显然就这样做复杂度高达 $O(n*Pmax^2)$。即使使用了优化，也很难得到令人满意的结果，我们只得另辟蹊径。

考虑到**如果有 P[1]+X 种奖章，存在一种分发方案满足要求，那么如果我们有 P[1]+X+1 种奖章，也一定存在可行方案**（大不了其中一种我们不用）。这个性质非常重要，因为由此，我们可以就用二分枚举 X，再逐个判断是否有可行方案的方法求得结果。

所以原先问题就转化为——如果我们已经知道共有 P[1]+X 种奖章，我们能否很快判断是否存在满足要求的分发方案呢？答案是肯定的。

方法仍旧是动态规划[2]，状态表示也类似，a[i,j]表示共有 P[1]+X 种奖章，前 i 个守卫已安排妥当，且第 i 个守卫有 j 个奖章与第一个守卫相同是否可能。

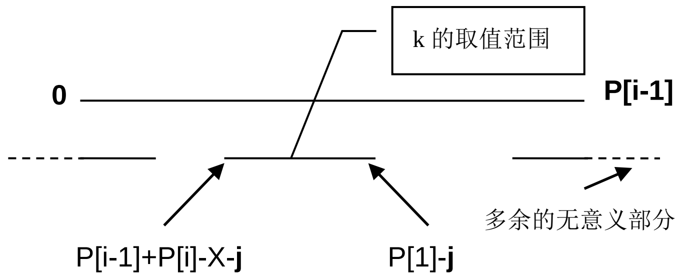
则状态转移变为：

$$a[i,j] = \mathop{\mathrm{or}}_{\substack{0\le k\le P[i-1],\ k+j\le P[1], \\ P[i-1]-k+P[i]-j\le X}} (a[i-1,k])$$

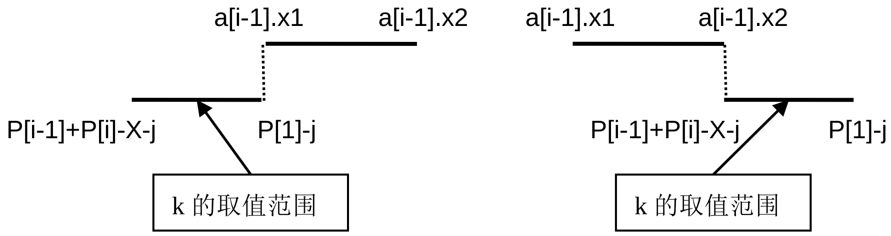
表面上看状态转移仍旧很繁琐，k 的取值有很多限制。但我们仔细观察，第二、三条合起来是 **P[i-1]+P[i]-X-j ≤ k ≤ P[1]-j**。

对于确定的 i，k 的取值范围在数轴上的表示是一条长度确定的线段，且线段的位置由 j 确定。原先第一条限制只不过是再给线段限定一个范围，去除多余无意义的部分（如图）。

[2] 准确地说应该是递推



初始状态 a[1]中只有 a[1,0]=true，其余均为 false。由归纳法很容易证明对任意 i，a[i,j]中为true的j一定是连续的一段。由此对每个i，所有状态a[i,j]可仅用两个端点表示，即a[i].x1、a[i].x2。



考虑要使 a[i,j]=true，当且仅当 k 的取值范围与线段**[**a[i-1].x1**,**a[i-1].x2**]**有交集，即满足

$$P[1]-j \ge a[i-1].x1 \quad \text{and} \quad P[i-1]+P[i]-X-j \le a[i-1].x2$$

即 $P[i-1]+P[i]-X-a[i-1].x2 \le j \le P[1]-a[i-1].x2$

由此，状态转移方程变为：

$$a[i].x1=\max(0,\ P[i-1]+P[i]-X-a[i-1].x2)$$

$$a[i].x2=\min(P[i],\ P[1]-a[i-1].x1)$$

初始状态 a[1].x1=a[1].x2=P[1]，状态总数 O(n)，状态转移 O(1)，可以说是高效的。

有了此方法，鉴于先前的分析，我们采用二分枚举 X，并利用高效的测试方法，即可在 O(n*logPmax)的时间解决整个问题。

[小结]

上面这个问题，看似难点仍旧在于动态规划，二分枚举只不过充当了一个附加手段。但实际上事先枚举的 X，极大地简便了动态规划的方程，才使得问题得以解决。应用这类二分枚举思想的最优性问题近来很是热门，而且这类试题很容易诱导选手直接采用动态规划或是贪心算法，而走入死胡同。

所以，今后我们在考虑最优性问题时，应注意问题是否隐含了一个有序的 01 序列，它是否可以用二分枚举的方法将最优性问题转化为可行性问题。切记！“退一步海阔天空”。

类型三：二分搜索——应用于无序序列

如果你认为只有在有序序列中才可以二分，那你就大错特错了，无序序列照样可以进行二分搜索。请看下面这个交互式问题——**推销员的旅行**（JSOI）

[问题描述]

作为一名推销员，Mirko 必须坐飞机访问 N 个城市一次仅一次。已知每两座城市间都有且仅有一条航线，总在整点起飞，途中花费 1 小时。每个航线的飞机总是不停地往返于两座城市，即如果有一架飞机在 5 点整从 A 市飞往 B 市，则 6 点整到达，且马上又起飞，7 点整回到 A 市……为了保证效率，Mirko 想把 N 个城市排成一个序列 $A_1,A_2,\ldots,A_N$，对于每个 i（1<i<n），Mirko 可以在到达 A_i 市后，做一小时广告，然后立即从 A_i 市出发前往 A_{i+1} 市。

可惜 Mirko 没有飞机的时刻表，所以他不得不打电话问航空公司。每通电话，他可以询问 A、B 两市之间的航线在正午 12 点从 A 飞往 B 还是从 B 飞往 A。由于 Mirko 不想在打电话上花太多钱，请你帮助他，用尽可能少的电话确定一条旅行线路。注意，交互库可能根据你的询问调整线路使你打电话的次数最多。

[分析]

问题比较长，让我们先分析一下它到底要我们做什么。

假设我们在正午 12 点从 A_1 市飞往 A_2 市，1 点整到达，又做了一小时广告，2 点整从 A_2 市出发前往 A_3 市……显然，我们总是在偶数整点时出发从一个城市前往另一个城市。

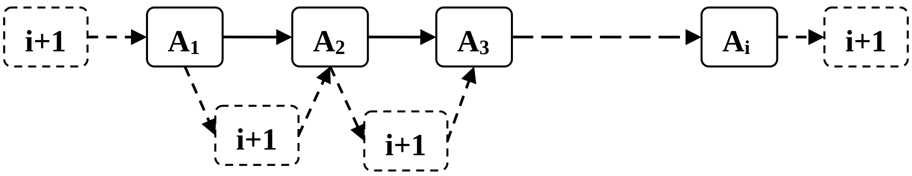
因为飞机往返时间恰为 2 小时，中间又没有停顿，所以如果正午 12 点的飞机是从 A 飞往 B，则任意偶数整点的飞机也一定是从 A 飞往 B 的。

由此，我们可以将先前的问题转化（注意：是转化而不是等价）为在一个竞赛图[3]中寻找一条哈密尔顿路的问题，我们的目标就是尽可能少地进行询问。

为了避免询问的盲目性，我们尝试使用增量法逐步扩展序列。

我们先任意询问两座城市，方便起见就选择城市 1 和城市 2。不妨设 1 到 2 有边，我们就得到一条长度为 1 的线路 1→2。

假设我们已设计了一条含有前 i 座城市、长度为 i-1 的线路 $A_1 \rightarrow A_2 \rightarrow \cdots \rightarrow A_i$，我们希望再加入一座城市 i+1，变成长度为 i 的线路。



如图，我们可先询问 i+1 到 A_1 是否有边。如果有，我们就将线路改作 $i+1 \rightarrow A_1 \rightarrow A_2 \rightarrow \cdots \rightarrow A_i$；否则，我们再询问 A_i 到 i+1 是否有边，如果有，我们将线路改作 $A_1 \rightarrow A_2 \rightarrow \cdots \rightarrow A_i \rightarrow i+1$。如果还没有，就表示两端不能插入，i+1 只能插在中间。

注意到 i+1 与 A_1 间的边是从 A_1 到 i+1。如果 i+1 有边到 A_2，我们就可以把 i+1 插在 A_1 与 A_2 间；如果 i+1 仍没有边到达 A_2，就表示 A_2 有边到 A_{i+1}，我们可再询问 A_3……

这样的询问会不会问到底也没法插入 i+1？不会，因为 i+1 有边到 A_i。

由此，我们得到了一种询问方法，但最坏情况下总的询问次数可能是 $O(N^2)$的。

由于**对每个点 $A_k(1 \le k \le i)$，要么 A_k 有边到 i+1，要么 i+1 有边到 A_k**。且 A_1 有边到 i+1，i+1 有边到 A_i。于是，我们可以用**二分搜索**寻找 A_k，使 i+1 可以插入 A_k 与 A_{k+1} 之间。

假设我们已知 i+1 可以插入 A_p 与 A_r 之间，我们询问 i+1 与 $A_{(p+r)/2}$ 间边的方式。

如果 i+1 有边到 $A_{(p+r)/2}$，则表示 A_p 到 $A_{(p+r)/2}$ 间可以插入 i+1；否则表示 $A_{(p+r)/2}$ 到 A_r 间可以插入 i+1。不断二分，直至 p+1=r。

这样，我们所需要的询问次数仅为 O(nlogn)。

[小结]

对于这类二分搜索问题，其实从根本上讲我们是在一个无序的 01 序列中进行查找，查找的对象是一个特殊的子串‘01’。有了这个子串，再利用构造法，就可以将结果转化为原先问题的一组可行解了。

当我们询问 i+1 到 $A_{(p+r)/2}$ 之间的边，如果 i+1 有边到 $A_{(p+r)/2}$，我们就设法将 i+1 插入 A_p 与 $A_{(p+r)/2}$ 之间。其实，这并不表示 i+1 就一定不能插入到 $A_{(p+r)/2}$ 与 A_r 之间，只是不一定能。而 i+1 一定能插入到 A_p 与 $A_{(p+r)/2}$ 之间。由此可见，这样的二分搜索方法往往应用于那些可行解很多，但需要高效地构造一组可行解的问题。

---

[3] 即有 N 个顶点，两两之间恰只有一条边的有向图

有了上述思想，我们再看一个例子——**非与门电路**（ACM/ICPC CERC 2001）

[问题描述]

有一种门电路叫做非与门（NAND），每个 NAND 有两个输入端，输出为两个输入端非与运算的结果，即 not(A and B)。给出一个由 N 个 NAND 组成的无环电路（这一点对于一个逻辑电路来说很重要），电路的 M 个输入全部连接到一个相同的输入 x，如图 NAND-1 所示。请把其中一些输入设置为常数，用最少的 x 完成相同功能。图 NAND-2 是一个只用一个 x 输入但可以得到同样结果的电路。

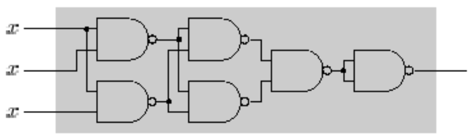


图 NAND-1

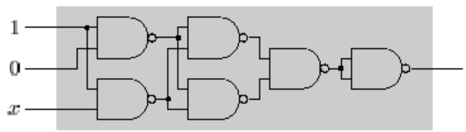


图 NAND-2

[分析]

首先想到的是使用表达式，将各个门电路的输出表示成输入的某个函数运算结果，以此连接整个电路的输入和输出。但由于 NAND 运算不满足结合律，也就无法进行表达式的运算。我们只得另辟蹊径。

事实上，我们很容易算出 x 分别为 0 和 1 时电路的输出结果 ans0、ans1。如果 ans0=ans1，显然我们一个 x 都不需要，将所有输入均设为 0 或 1 即可满足要求；如果 ans0≠ans1 呢？一个都不用是不可能的了，那么只用一个呢？

考虑到，即使只有一个 x，我们能够选择的输入序列仍非常多，只要有一个满足要求即可。因此，我们尽可大胆猜想，最多只要一个未知输入 x，即可满足要求。

只用一个 x，这表示当这个 x=0 时，电路输出恰好为 ans0；当这个 x=1 时，电路输出恰好为 ans1。我们只改变了一个输入端口的值，而且是从 0 改为 1。

由此，我们假想一个全为 0 的输入序列，经过若干次这样的修改后，最终将变为一个全为 1 的输入序列；另一方面，开始时电路输出 ans0，而最后电路输出 ans1。所以，必定存在某个时刻，当一个输入由 0 变为 1 时，输出恰从 ans0 变为 ans1。

因此这就证明了我们刚才的最多只用一个 x 的猜想是正确的。同时，算法也浮出水面：

假想我们构造了如下 M+1 个不同的输入：

0:　0 0 0 0 0 0 … 0
1:　1 0 0 0 0 0 … 0
2:　1 1 0 0 0 0 … 0
…　…
M:　1 1 1 1 1 1 … 1

要求我们找出相邻两组输入，前一个输出 ans0，后一个输出 ans1。

我们仍旧可以由**二分搜索**高效完成：已知第 p 组输出 ans0、第 r 组输出 ans1，我们计算第(p+r)/2 组输出 ans’。如果 ans’=ans0，取 r=(p+r)/2；否则 p=(p+r)/2，直至 p+1=r。

于是问题在 O(NlogM)时间被很好地解决。

[小结]

可以看出，这类二分搜索问题的难点在于构造。如何将一个完全不相干的问题引入二分搜索的应用范围，利用一个不起眼的 0 到 1 的改变构造出原问题的一个解决方案。构造法没有统一的格式或者规律可循，只能具体问题具体分析。我们所能做的，就是在平时多接触、多思考、多积累。

【总结】

本文这里仅简单地介绍了几个例子，要涵盖二分策略的所有应用甚至是大部分应用都是困难的。我想指出的是二分思想虽然简单，但是它的内容还是非常丰富的。我们不能让二分思想仅仅停留在一般有序数组上的最基本的二分查找，而应该扩展到更广泛的应用上。

二分策略常与其它一些算法相结合，并借以隐蔽自己，以逃离我们的视线。这就要求我们能有扎实的基本功、丰富的解题经验、大胆合理的猜测以及活跃的创造思维，方可“以不变应万变”。

【参考文献】

《实用算法的分析与程序设计》吴文虎 王建德

《算法艺术与信息学竞赛》刘汝佳 黄亮