平衡规划

——浅析一类平衡思想在信息学竞赛中的应用

福建省福州市第八中学　郑 暾

【目录】

【摘要】

应用计算机解题的核心是算法设计。但算法设计方面涉及的领域十分丰富。我们不能奢求能完美地应用所有的算法，所以我们关注的通常是如何合理运用已学知识，并在所掌握算法间构建一种平衡，在限定的时间内尽可能多地解决问题。本文尝试讨论一类平衡思想应用于算法构造、算法实现的模式。

【关键字】

平衡思想、平衡点、时空效率、编程复杂度

【正文】

一、引言

平衡通常指物体或系统的一种状态。处于平衡状态的物体或系统，除非受到外界的影响，它本身不会有任何自发的变化。多种状态达到平衡通常是我们所追求的目标。

平衡思想是一种奇妙的思想，它的应用十分广泛。在算法设计，数据结构设计甚至程序设计中都能发现它的身影。计算机竞赛就是一场博弈，寻找这场博弈中的平衡点，合理应用平衡思想辅助算法设计与程序实现，往往能起到化腐朽为神奇的作用。

在信息学竞赛中，平衡思想通常有以下几个方面的运用：

1、博弈问题。有许多博弈类问题都可以转化成寻找平衡点的问题。

2、数据结构的构建。每种数据结构都能以优秀的性能支持某些操作，合理选择应用数据结构，往往能通过略微提高一些操作的复杂度，降低大多数操作的复杂度，在不同操作的效率之间构建一种平衡，以达到优化的目的。

3、时间效率 vs 空间效率。这类问题是我们经常遇到的问题。这类问题通常有这样的特性，我们能找到时间效率（或空间效率）十分优秀的算法，但代价是空间效率（或时间效率）极端低下。如何合理设计算法，组织数据，平衡二者的关系是解决这类问题的重点。

4、时空效率 vs 其他。如果面对难题难以设计出优美的算法，又或者设计

了优秀效率的算法，却无法实现或难以实现，就会出现非常尴尬的局面。合理应用平衡规划解决这类问题，往往能收到意想不到的效果。而这类问题也正是本文所要重点探讨的问题。

下文将试图论述运用平衡思想解决这类问题中的三种常见模式：经典算法的非典型实现，效果优秀的非完美算法，以及复杂问题的简单化构造。

二、应用平衡思想的几类问题

1、经典算法的非典型实现。

大多数经典算法通常是为很多人所熟知，并且能够熟悉运用。经典算法通常也有很多不同的实现方法。例如拓扑排序，如果数据范围比较大，通常使用算法复杂度为O(n)的程序，但是如果范围比较小，一个不超过10行的$O(n^2)$的程序可以使代码看起来更为简洁。而不同的实现方法中，哪怕只是细微的不同，实现后的性能与效率也可能有很大的差别。更进一步，有些算法虽然堪称经典，但是无论是思考复杂度还是实现复杂度都相对颇高，在考场上来说，使用一些相对简单实用的方法无疑是一种不错的选择。

例题一、警卫安排问题(ural 1099)

题目描述：

给定若干警卫间搭档关系，要求成对给警卫安排保卫工作，求能够安排警卫的最大值。（警卫人数不超过222）

初步分析：

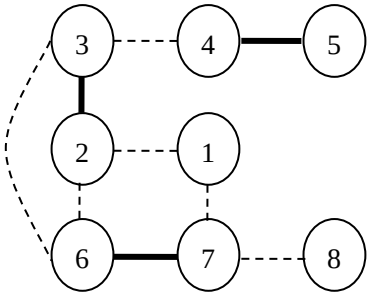
题目描述很简单，稍加分析后我们很容易看出来，这题的本质其实是要求我们求出任意图的最大匹配。

任意图的最大匹配的经典算法是应用带花匈牙利树，时间复杂度为$O(n^3)$，对付这道题来说是绰绰有余的。但是带花树本身比较复杂，思维复杂度与编程复杂度较高，而且实现起来很容易退化，考场上有限的时间内难以完成。于是我们尝试考虑替代算法予以解决。

解法分析：

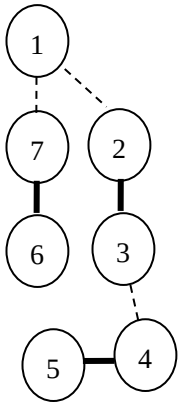
解决二分图最大匹配的时候，主要过程是不断寻找交错增广树来调整。那么这么做对于任意图是否可行？答案是否定的。

考察右边这张图（图一）。图中粗线表示当前状态下已匹配边，虚线表示未匹配边。若我们当前找的增广树如图二所示，那么我们就无法找到一条增广路。但实际上，存这样的一条增广路：$1\rightarrow 2\rightarrow 3\rightarrow 6\rightarrow 7\rightarrow 8$。为了找到增广路，我们可以采用搜索的方法，但这样寻找增广路复杂度过高。



图一

我们注意到，采用调整增广轨的方法，如果一个点之前已经被匹配到，则之后无论如何调整，这个点始终能被匹配到。因此，对于一个待匹配点，是否能找到一条以它为起点的增广路是优化解的关键。而是否能找到一棵增广树又很大程度上依赖于之前找寻的情况，若要设计数据使得无法找到增广树通常又依赖于特定的扩展顺序。这启发我们采用随机扩展顺序的方法来尽量避免形成类似图一的特殊局面。



图二

笔者的程序中生成了两个随机序列，一个作为点的初次访问序，一个作为点的拓展顺序，然后直接使用一开始所说的寻找交错树的方法来扩展。同时，采用多次随机运行的方法提高最优解的出现概率。

但是作为随机贪心算法，除了拿到AC之外，我们更应该关注这个程序通常的运行情况如何。上述算法中，在ural点数限制仅仅为222以下的情况下，通常运行次数却要设定为20至50才能基本上保证AC。相对于数据本身，这个运行次数还是比较大，说明算法本身具有比较大的不确定因素，以及出最优解概率并不是很高。所以我们需要进一步优化以提高一次运行的最优解出现概率。

进一步优化

上述解法中曾提到，采用调整增广轨的方法，如果一个点之前已经被匹配到，则之后无论如何调整，这个点始终能被匹配到。它带来的另一个信息是，若一个点之前未被匹配到，那么在本轮搜索中，这个点最终将很可能保持未匹配的状态。因此影响最终结果的，往往是某个本应该被匹配到的点因为之前增广树的查找失

败而被放弃匹配。初始算法解决这个问题的方法是全局重新搜索，所以常常出现为了一个点而全部重来的尴尬场面。其实这是舍近求远。我们完全可以换一个扩展顺序，对于当前未找到交错树从而无法匹配的点，直接重新搜索一棵交错树！当然大部分的图都无法实现完美匹配，所以类似运行次数的限制，我们需要设置一个失败次数上限。当为一个未匹配点寻找匹配点时，只有失败次数超过了这个上限才放弃。

那么失败上限次数应该设置为多少比较好？进一步的，这个方法实现起来究竟效果如何呢？为此笔者进行了一系列的实验，得到了如下的实验数据表：

运行次数-失败上限	5-5	5-10	10-1	10-10	20-1	20-10	50-1
Accepted	90%	100%	0%	90%	60%	0%	100%
Wrong answer	10%	0%	100%	10%	40%	0%	0%
Time limit exceeded	0%	0%	0%	0%	0%	100%	0%
平均 AC 时间	0.0761	0.1471	--	0.2960	0.0385	--	0.0795

实验的运行平台是Ural1099的测试，时限是0.5秒。表头的N-M表示程序将重复运行N次，失败上限设置为M，例如5-5表示程序将重复运行5次，失败上限设置为5。

实验表明，这个方法能显著地提高一次运行的最优解出现概率。从表中数据可以看出，初始算法直到**20-1**才有50%以上的AC率，而优化后的方法将失败上限设置为5就可以让仅仅重复运行5次的AC率达到了90%。当然，这种方法同样存在一定的随机因素，加之Ural1099的数据比较强大（从数量到质量），所以出现了如表中5-10是100%AC但是10-10却只有90%的AC率的情况。

但正所谓有得必有失。优化后的方法有着极高的准确率，但同时时间效率并不高。实验中，5-5的时间已经逼近了50-1的时间。虽然相对于时限来说还是比较轻松，但是其增长还是较可观的（20-10的全部超时就可以说明这一点）。实际上，虽然理论上时间复杂度仅仅是多一个所设置的失败上限的常数，但由于将进一步优化中需要大量地使用了随机函数，而生成随机函数的常数比较大，造成了实际程序的常数较大，拖累了时间。

所以，两种算法都有其各自的优缺点，这就需要我们根据题目的给出的信息，根据实际算法的需求来进行选择。

进一步扩展：

随机搜索序和设置失败上限的作用并不局限于此。

对于随机搜索序，表面上看是根据克制数据或克制情况的**顺序依赖性**[1]，应用随机的顺序来进行回避。其实其本质是：通过设置一些随机权值，以改变一个当前状态的属性，从而回避特殊情况的生成（例如本题是回避克制数据的生成）。我们常用的Treap，随机快排（随机选择比较变量），其本质也是如此。

设置失败上限则是随机搜索序关于准确率的一个优化。随机算法不可避免还是有可能无法得到我们理想的状态或结果（例如本题依然可能找不到存在的增广路，又例如Treap并不完全平衡）。随机搜索序通常是全局重新运行来提高最优解的出现概率，而设置失败上限则是通过对于局部问题的精益求精来强化全局目标情况出现概率，常见的应用有大素数的测试等。

小结：

对于这道题，两种方法都可以比较轻松地通过。考虑到数据范围并不大，为了追求准确率可以增加参数上限，或者为了保证时间效率压缩参数上限，这需要我们根据实际情况合理平衡二者之间的关系。以准确率为代价我们得到了随机贪心算法，以时间复杂度为代价我们得到了准确率的进一步的优化，但不论是哪种方法，都能有效地降低了思维与编程复杂度，达到了二者之间的相对平衡。

例题二、Jackpot（PKU2103）

题目描述：

等概率选择任意整数，若其能被p1,p2,p3...pn中至少一个数整除，那么称当前情况为胜利局面。给定p1,p2,p3...pn,（n<=16）求得到胜利局面的概率（用最简分数表示）。

初步分析：

本题看起来十分复杂。题目作为参考，给了一个比较复杂的计算概率的式子。

$$P=\lim_{k\to\infty}\left(\frac{S_k}{2k+1}\right)$$

其中P表示最后结果的概率，S_k 是 -k 到 k 之间至少能被给定的$p_1,p_2...p_n$其中一个数整除的个数。

但如果直接用这个式子比较复杂。所以我们设计了下述算法代替。注意题目

[1] 即特殊情况依赖于一定的顺序，例如本题是克制数据的生成依赖于一定的搜索顺序。

中涉及高精度计算，但本题时限卡的比较紧，如果用普通的高精度容易超时，巨大的常数无法忍受。如何合理优化常数成为了解决问题的关键。

解法分析：

本题其实是数学题。题目等价于求取数区间为1~lcm(p1,p2,p3...)的时候的概率，但由于1 <= p_i <= 10^9，使得最小公倍数可能很大，直接扫描显然不现实。注意到给定的数最多只有16个，所以我们可以应用容斥原理求出区间中可以得到胜利局面的数的个数。由于题目涉及了许多求gcd，lcm的高精度，一个小技巧是开一个素数列表，当前状态以纪录素数的次数的方式记录，最后再还原成原来的整数。主算法的设计与实现并不困难。所以在主算法相同的情况下，高精度算法的实现优劣就成了左右程序效率的关键。

本题涵盖了许多的高精度算法，而几乎每一次运算都要使用到高精度运算。其中使用最多的有高精度乘以单精度与高精度加减法。

高精度运算有一个通用的优化：压位。在longint范围内通常是压4位（即万进制），在int64则可以压8位，而压位能有效地减少高精度数组的长度，从而提高效率。注意到程序实现时要大量使用div与mod运算，这两个运算的常数是比较大的。那么是否有办法绕开这两个运算呢？

答案是肯定的。

我们注意到，除法与取模的运算是为了进行压位处理的操作，（注意到若是高精度乘法，中间结果可能比较大，所以用while递减实现取模的效果更差）。但无论压位与否，我们的所采用的万进制等进行压位实质上还是沿用着通常的十进制的思维模式。但计算机处理二进制运算（位操作）显然要比十进制常数小。所以我们可以跳出思维惯性，采用二进制压位，这样，一些取模或者除法运算就能用位操作很好地实现。

下面是两个程序主要部分的伪代码的对比。（高精度与单精度乘法）

采用10^n进制压位的乘法	采用2^n进制压位的乘法

for i ← 1 to a[0] + 1 do
begin
p ← a[i] * b + p div 10^n;
a[i] ← p mod 10^n;
	for i ← 1 to a[0] + 1 do
	begin
	p ← a[i] * b + p shr n;
	a[i] ← p and (2^n-1);
	end;

end;	

可以发现，两者的程序几乎相同。但采用2^n进制压位的乘法利用位运算代替了运算常数很高的div与mod，而同等情况下压位后数组长度与通常的压位差别不大（甚至稍优），所以实际应用时可以取得很好的效果。在本题中，笔者的程序应用了这个方法，在1秒内就通过了所有的数据。

当然有得必有失。这种优化因为采用2^n进制进行压位，但通常答案都是要求输出10进制的结果，所以无法类似10^n那样直接输出。在输出时需要对其进行高精度计算对10取模。这里就不再赘述了。

小结：

例题在实现中跳出了典型实现的思维框架，创新地使用了二进制压位思想。虽然修改后在最后输出时的取模操作提高了输出的时间复杂度，但毕竟输出的次数通常不多，而程序中调用高精度运算的次数却可能很大（例如本题）。这里以提高输出的复杂度为代价，降低了运算的常数复杂度，实质上是构建了二者的算法复杂度之间的平衡，所以最后能取得相当不错的效果。

经典算法还有很多，也有一大部分无论设计与实现都存在一定的困难，很多细节的操作往往也能左右算法的效率。在这样一类问题中，本来是简化问题的经典算法却成为了程序实现的障碍。根据实际需求，合理改造算法，细心设计算法细节的实现，有效地平衡算法中不同部分的复杂度关系，建立各部分之间的平衡，往往是解决这类问题的利器。

2、效果优秀的非完美算法

在信息学竞赛的赛场上，遇到不会做的题目对大多数人来说是很平常的事情。但是如果就此放弃也未免太可惜了。毕竟一道题一百分往往能直接左右竞赛的结果。如果题目有部分分，又或者题目是要求我们求最优解或者构造最优方案时，采用些非完美算法或者近似算法往往能收到不错的效果（虽然不一定能满分）。如果类似下文能使用DP、贪心算法构造出“几乎是对”的算法的话，则能为我们节省下大量的思考时间，达到了得到的分数与所消耗时间的平衡，性价比

可谓是非常高。

例题三、追捕盗贼（NOI2007 Day2 catch）

题目描述：

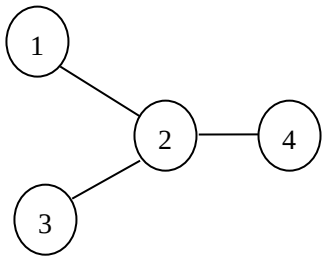
Magic Land 由N 个城市，N-1 条公路彼此连接起来，任意两个城市间都可以通过若干条公路互达。大盗Frank能够在公路上以任意速度移动。你的任务就是抓住大盗Frank。但是你完全不知道Frank躲在哪个城市，或者正在哪条公路上移动，所以需要制定一个周密的抓捕计划。每个单位时间你可以完成以下几个操作：1、在某个城市空降一位警探。2、把留在某个城市里的一位警探直接召回指挥部。3、让待在某个城市的一位警探沿着公路移动到另一个城市。若警探和大盗在同一城市或者同一条公路上移动则可以抓获大盗。请你制定一个使用人数最少的捕获计划，并且操作次数不能超过20000。

初步分析：

本题初看上去并不复杂，但却是这次比赛最难的一道题。事实上，这是一道论文题级别的题目[1]，涉及许多定理与推论，思维复杂度编程复杂度同时达到一定的高度。事实证明，考场上也没有人想出来标准做法。但这一百分也不能扔掉。考虑到这题的评分标准存在部分分，而且只要有一个可行解就有分，即使是简单的构造也能拿到一定的分数。所以各种骗分的算法蜂拥而至，也取得了一定的效果。但是这题的数据还是比较强大，寻常的骗分效果并不理想。于是经过一定的思考，我们可以发现一种比较不错的替代算法。

解法分析：

我们首先考察简单的样例数据。



城市与城市之间的连接关系如右图所示。一种可行解是：始终在城市2驻扎一个警探，然后在城市1降落1个警探并往城市2移动，到达城市2后收回，并对城市3、4进行类似的操作。

样例虽然简单，但给我们提供了一种非常有用的构造思想。我们知道，树上

[1] 本题实质上是图论中的“Search Number”问题，对于一般图，Search Number 问题是 NPC 的。对于树上的特殊问题，可以在 O(N)时间内求出 Search Number，在 O(NlogN)时间内求出相应方案。有兴趣的同学可以查找相应的资料。

的任意一个点可以将树拆分成若干个无关的部分。对应于这道题目来说，在一个点上常驻一个警探可以使大盗无法从被这个点分隔开的一个部分进入另一个部分。显然，为了使我们之前的搜索过程没有浪费，进行分叉的搜索时，在分割点上驻一个警探可以保住之前的搜索成果。

这样，我们就可以每次选择一个点将图拆分成若干个不同的部分。每个部分就是一个缩小规模版本的题目。这不是神似我们常用的动态规划的解题构造法么？根据这个思路，我们能得到一个大概的转移方程式：

$$F(V)=\max(F(V_i))+1$$

其中，$F(V)$表示树的点集为V时候的较优（最优）警探放置数量，$F(V_i)$表示第i棵子树（点集为V_i）的情况。

但是子状态的情况并不好构造，如果构造不当，这样的结果并不一定很优。

最简单的方法，我们可以搜索每次拆分的点。但是这么做时间复杂度十分高，而且输出方案、程序实现也相对麻烦。同时我们考虑这样的情况，例如一个点将这棵树分成了若干个部分，如果我们搜索行动进行到了最后一部分时，原本驻扎的警探就可以沿路径前进，这样可以节省一个驻扎在分割点的警探，结果通常会更优。

而且，一棵子树的最优解情况并不容易转移到原树上。对于这种方法，子树的若干最优解如何选择，如何与整棵树的解进行衔接从而构造最终解，如何保证正确性最优性的兼并？这些都是我们需要考虑但又十分困难的问题。

但我们怎能就此放弃这道题？花时间考虑过于复杂的标准方法显然不现实。考虑到有部分分，我们完全可以修改上述方法使之能取得一个较优解。

上述方法有一个重点是考虑将树分解成了若干个小规模的问题，但同时产生一个问题，如何衔接子问题与总问题的构造？我们不妨将初始的分割点作为树的根，构建整棵树，子问题的分割点直接设置为子树的根。每次选取一个需求最多的子树作为最后扩展部分。采用这样的思路，一个新的近似构造方法产生了。

首先枚举初始分割点，将初始分割点作为树根建树。对这棵树进行类似树状动态规划的贪心构造。我们有如下状态转移方程。

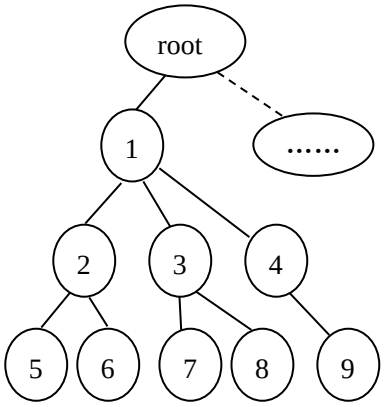
$$D(x)=\max\begin{pmatrix}\max(D(y)) & \{y\in son[x]\}\\ \max(D(z))+1 & \{z\in son[x], z\neq y\}\end{pmatrix}$$

其中D(x)表示以x为根节点的树的较优（最优）警探放置数量。son[x]表示节点x的儿子节点的集合。

构造具体解的时候，我们也完全可以按照转移来进行构造。整个过程是递归实现的。将两个警探驻扎在分割点上，派出一个警探移动向分割点的子节点，然后重复上述过程，探索完毕（即警探走到了叶子节点）就立刻将警探回收。对每一个分割点，最后探索需求警探数最大的子树，此时不用额外派遣警探，直接将分割点上警探移动到子节点，并重复上述过程直到整棵树探索完毕。

从实现方面来说，算法的主体部分无论计算还是构造，实现起来都相对比较轻松。但这个算法的最后效果如何呢？

上述算法主要弱点在于，我们除了初始的分割点是枚举的外，其余子树初始分割点直接定为子树的根。这样做虽然使得构造解变得轻松，但有可能使得子树得到的解无法保证最优，从而影响整体解的质量。例如下图的情况[1]，如果按照上述的贪心算法，我们会得到最后的答案是4，但是实际上最优解只需要3个警探就可以了。构造方案如下，root节点始终驻扎警探，分别派遣警探a，警探b到节点5，6，并同时往节点2移动，回收警探b，让警探a往节点1移动。派遣警探b到root节点并顺次移动到节点1，4，9。回收警探b，派遣a往节点3移动并派遣警探b到节点3，之后警探a，b分别往节点7，8移动并回收。对于root的其他子树采取类似的操作。



我们的贪心算法因为对于子树默认以根节点作为分割点，所以无法得到上述的构造方案。那么这个算法是否还有意义呢？

我们注意到影响到当前点解的质量只有2个因素，即子节点中需求警探数最大与次大的点。因此，其他子节点的解即使稍劣也不会影响最终结果。也就是说，实际的解给部分的解提供了一个弹性空间。同时，即使部分数据可以克制这个算

[1] 虚线与省略号部分表示节点 root 还有不少于 2 个和“节点 1 所在的子树”相同的子树。实际上，若 root 节点只有 2 个“节点 1 所在的子树”，那么这个贪心依然能得到 3 这个最优解。

法，但由于小部分克制数据比较难以构造（上述反例也至少需要大约30个节点来“配合”），随着数据量加大，通常随机的数据即使生成了克制数据也几乎无法影响最后的解。而且，即使是给定顺序的分割点，得到的解通常也是较优解甚至最优解。而枚举第一个分割点虽然把时间复杂度提高到了$O(N^2)$，但时间方面依然很宽裕，解的质量却有效地提高了。

笔者使用这个方法测试了1000组随机数据，结果表明，有百分之八十五左右的计算结果只需要5个警探，其余都是6个警探（极偶然出现过只需4个警探的情况）。对照得分规则，可以发现这样至少能获得60%的分数。（2个以下警探肯定能正确出解）。实践证明，这种方法对于考场的数据可以拿到96分的好成绩，只有一个数据点结果会比标准答案大1。

小结：

虽然没有AC掉这道题，但对于实质上属于论文难度级别的这道题，考场上能够做到96分已经足够了。考虑到这种构造法思维复杂度较低，实现起来十分轻松，时间复杂度中规中矩，时限范围内足够出解，而且解的质量相对较高。考场上，这样的程序能为我们省下大量的时间去解决其它问题。

在这道题目中，我们用解的质量为代价，极大地降低了程序思维复杂度与实现方面的难度，同时在基础的方案上，我们合理优化构造的方法，使得解的质量通常可以达到最优。

在这一类问题中，原本复杂的问题我们无法解决，一味追求AC往往得不偿失，而选择一些效果不错的近似算法就有着较高的性价比。这样实质上是通过些另类的途径，有效地建立了所得分数与所耗时间的平衡，以少量的代价取得了优秀的效果，也不失为一种优秀的解题策略。

3、复杂问题的简单化构造

当然，并不是每一道题都能够找到第二类问题那样的效果优秀的近似算法。而且如果是形如ACM之类的比赛，或者题目要求维护一系列操作时，近似算法的作用往往会被限制。此时，我们即使无法找到时空效率十分优美的算法，也需要沉着下来，仔细分析题目特征。一类常常使用的方法就是，在可行的时空限制下，

以时空效率为代价，降低思维复杂度，建立时空复杂度与思考实现复杂度的平衡，这就是复杂问题的简单化构造。

这类问题通常有一个特征。如果题目所给的条件再强大一些，把可能涉及的情况限制成某些特例时，我们能找到优秀的算法，但扩展之后却不能通用。如何有效应用特例提供的信息来简化模型，进一步地建立特例与一般的平衡，往往是解决问题的关键。

例题四、数列维护

题目描述：

给一个长度为n(n<=100000)的整数数列，要求维护以下几个操作。

1、数列第i项到第j项同时加上一个整数。

2、询问第i项到第j项中比整数c小的数有多少个。

最大操作数不超过10000。

初步分析：

本题要求我们应用数据结构维护关于序列区间的一些操作。由于同时涉及区间的比大小，询问，增减操作，我们的第一反应通常是用树套树的数据结构来解决这样的问题。但注意到题目中有变换区间序列的操作，简单地套用模型会使得区间合并的时候时间复杂度并不理想。再者，树套树有着复杂的代码量，调试起来也并不容易，如果不小心就可能造成巨大的损失。注意到题目中最大操作数仅10000，时限方面也并不紧张（5s），我们可以放弃树套树的思想，考虑采用实现简单、代码量低、调试方便的算法来代替。

解法分析：

对于区间操作，还有一种数据结构也常常为我们所用，那就是块状链表。不过，直接写普通的块状链表复杂度是$O(m\sqrt{n}\log n)$，但代码量依然巨大。但块状链表的思想可以为我们所用。我们注意到，题目中的区间操作仅局限于区间增减或者询问，并不会打乱数列的顺序，所以，我们不妨稍微改造下块状链表——块状数组。

块状数组是这样的一个数组，它直接对原数组进行划分，分为$\sqrt{n}$个部分，

每个部分长度为$\sqrt{n}$，每个部分开相应的域以记录类似增减的操作信息（当然也可以是一些统计的信息，但这不在本题考虑范围之内）。这样的结构实现起来十分简单（使用原数组即可），适合的操作包括一些区间大小询问，区间增减操作，但不支持区间插入删除操作，原因是块状数组适合这样的序列操作，序列项与项之间隐含着严格不变的次序关系。但本题所要求的操作正好能满足这个条件。

在这道题目中，我们需要维护的操作有两个：区间增减操作，区间询问操作。注意到增减操作时，如果完全覆盖了一个划分区间，那么这个操作就不会影响这个划分区间中数的大小关系。也就是说，对于一个区间增减操作，我们至多只需要更新“操作区间”的头尾所在的划分区间的序列关系。而值得注意的是，如果我们能保证时时维护划分区间使之有序，那么区间询问操作就可以应用二分法轻松完成。基于上述思想，我们可以设计出如下在线算法。

设数组A为原数组，令数组$B=A$。预处理为：将数组B划分为$\sqrt{n}$块，每块长度为$\sqrt{n}$（块状数组）。对于数组B的每个部分，应用快速排序将其按从小到大排序，并开额外的域来记录区间增减情况。初始时所有额外域的数值为0。

对于操作1（数列第i项到第j项同时加上一个整数），用常数时间求出操作区间的连续部分。对于完全被操作区间覆盖的部分，只需修改相应的标记域值即可。对于未被完全覆盖的部分（只可能是头尾，最多两个部分），直接对该部分内相应数值进行修改。最多修改$\sqrt{n}$个区间，头尾区间最多修改$\sqrt{n}$个数值，修改后重排序代价为$\sqrt{n}\log\sqrt{n}=0.5\sqrt{n}\log n=O(\sqrt{n}\log n)$，所以该操作最坏情况下复杂度为$O(\sqrt{n}\log n)$。

对于操作2（询问第i项到第j项中比整数c小的数有多少个），方法是类似的。首先用常数时间求出操作区间的连续部分，对于完全被操作区间覆盖的部分。因为其中是有序的，可以使用二分法求出比c小的数的个数。对于头尾所在的两个区间，只需直接扫描一遍并统计即可。这样最多访问$\sqrt{n}$个区间，每个区间询问复杂度为$\sqrt{n}\log\sqrt{n}=0.5\sqrt{n}\log n=O(\sqrt{n}\log n)$，头尾区间最多扫描$\sqrt{n}$个数值，所以总的时间复杂度最坏为$O(\sqrt{n}\log n)$。

综上所述，总共m个操作，时间复杂度最坏为$O(m\sqrt{n}\log n)$，最坏情况复杂度在千万级别，足够通过所有数据。

小结：

实际上，本题虽然有采用树套树，时间复杂度为$O(m\log n\log n)$的算法，但其思维复杂度比较高，实现也并不容易。考虑到题目的特殊性，我们可以发现序列中隐含的不变的次序关系，并根据数据范围和时限选择合适的算法。本题中，我们并没有硬性追求时间复杂度十分优美的算法。由于题设对操作的限制，使得我们能通过改造经典数据结构的实现方法，有效地降低了思维与编程复杂度，还在一定程度上优化了常数。这实质上正是在时间复杂度与思维复杂度之间找到了一个平衡点，并借此有效地解决了问题。

例题五、树的维护(PKU 3237)

题目描述：

给定一棵有N个节点的树，点编号为1～N，边按给定顺序编号为1～N-1，每一条边有一个边权。需要对这棵树维护三个操作：

1、将某一条边的权值修改为V。

2、将点A到B的路径上的边权值取负号。

3、询问点A到B的路径上的权值最大值。

数据组数不超过20，点的个数不超过10000。

初步分析：

本题主要是对树上的路径进行操作，这类问题我们往往可以采用动态树来解决。并且对于这道题，动态树有“看着十分舒服”的对数级别复杂度。但问题并没有解决。首先，对这题使用动态树有种“大材小用”的感觉，因为动态树功能最强大的地方在于它能维护一棵不断改变形状的树，但本题中树的形状是固定的。其次，动态树实现起来比较复杂，巨大的常系数也不得不为我们所考虑。所以，我们尝试考虑一些其他算法来代替动态树，以便更有效地解决这个问题。

解法分析：

我们不妨先考虑题目的一种特例情况，即树退化成了一条链的情况。这个问

题便是我们常常见到一类应用线段树解决的问题。那么是否能将一条链推广成树的情况呢？

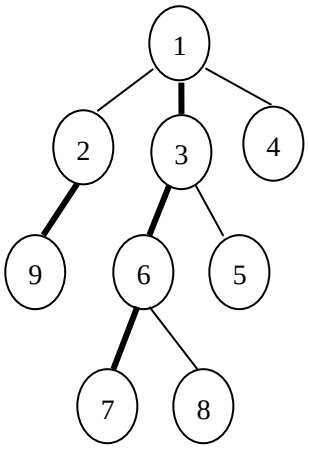
笔者首先考虑的情况，是将树按照某种规则（例如深搜序）进行遍历，将遍历时产生的访问序列当成一条链的情况来处理。遗憾的是，经过若干次尝试后，发现情况并没有想象中那么简单，故放弃了这种想法。

但上述想法也并非一无是处。其实上述想法的主要目的是将树的情况转化成链的情况。既然转化不成功，不如直接把树拆成一条一条的链来做。于是，下述算法产生了。

首先我们任选一点为根构建整棵树，并将每条边的权值下放到边连接的两点中的儿子节点上。（整棵树的根取值可以定为任意值，因为并不影响任何操作）对于任意一个节点x，假设它的儿子们都属于一条自底向上，并且以自己为结尾的链上，那么我们选择其中最长的一条链连到x上。这样树上的所有点都属于不同的链上（有些链只有一个节点），我们称这些链为**拆分链**。注意之前对边下放权值的好处这里就体现出来了——链与链之间断开的边不用作为单独的边而特殊考虑了。

例如右图的情况，我们就将这棵树拆成了如下5条拆分链（图中用粗线表示拆分链的边）：

$1\to3\to6\to7$，$2\to9$，4，5，8。



对于每条链，建立一棵线段树，由于取负操作的存在，需要同时维护这条链中的最大边与最小边。链与链的拆分关系可以用并查集存储。

对于操作1，直接修改存储该边权值的点的权值，并维护该点所在线段树中的最大值与最小值。复杂度为$O(\log m)$。显然m的最大值为n。

对于操作2，我们先求出点A与B的最近公共祖先root，然后分别对A到root，B到root的两条链进行操作。每次操作就是依次访问他们之间的拆分链，在相应的线段树上进行取负操作。

对于操作3，类似操作2，求出A与B的最近公共祖先root，分别求出A到root，B到root的两条路径的结果，取较大的输出。每次操作依然是依次查找当前路径

所在的拆分链，然后对这一段在这条链的线段树上进行询问操作。

以上所有关于最近公共祖先，路径与拆分链的查找，都可以在常数时间内完成。所以这个算法复杂度的瓶颈就在于，操作2与操作3中最多可能访问的树的拆分链的个数。

从直观感觉上来说，我们的拆链方法对于普通的数据，都能使得很多的路径基本落在拆分链上，所以这个数通常并不会很大。

但这样总是不够保险。实际上，我们也可以估算出，操作2与操作3的复杂度最多不超过$O\left(\sqrt{n}\log n\right)$。

[引理]按照以上贪心思想对任意一棵 n 个节点的树进行拆链，任意点 x 到其任意祖先 root 之间的拆分链数目不超过$O(\sqrt{n})$条。

[证明]定义一个集合 S，它的元素是 x 到 root 之间所有的边。设 S 中的边属于 k 条拆分链，这些拆分链与路径公共部分的最上层节点为$P_1,P_2\ldots P_k$，其中$P_k=root$。那么 P_2 必然连接着另外一条向下走的拆分链，这条拆分链除去属于 S 的部分至少还有 1 个点；同时 P_3 也必然连接着另外一条向下走的拆分链，这条拆分链除去属于 S 的部分至少还有 2 个点；……；同理 P_k 连接的向下走的拆分链除去属于 S 的部分至少还有 k-1 个点。这样以 root 为根的子树至少要有$1+2+3+\ldots+k-1+k=\frac{k(k+1)}{2}$个点。显然 $\frac{k(k+1)}{2}$ 最大为 n，这时$k=\frac{\sqrt{8n+1}-1}{2}=O\left(\sqrt{n}\right)$。引理得证。

[定理]理论上操作2与操作3的时间复杂度最坏为$O\left(\sqrt{n}\log n\right)$。

[证明]根据[引理]，我们知道每次操作2与操作3最多访问$O(\sqrt{n})$条拆分链，而每次访问拆分链需要维护拆分链对应的线段树，维护的复杂度为$O\left(\log m\right)$，其中m最多不超过n。所以操作2与操作3的时间复杂度最坏为$O\left(\sqrt{n}\log n\right)$。

以上我们证明了操作2与操作3的复杂度最多不超过$O\left(\sqrt{n}\log n\right)$。从而对于一

组数据，整个算法的时间复杂度为$O\left(n\log n+q\sqrt{n}\log n\right)$，看起来时限有些吃紧。但实际上这个上限是一个十分宽松的上限，而且程序实现的常数比较低，所以运行起来效果相当不错。PKU上，笔者的程序用了951MS就通过了所有的数据。

小结：

本题中，我们放弃了看上去十分优美的对数复杂度的动态树(复杂度为$O(n\log n+q\log n\log n)$)，转而通过对树的拆链操作来完成题目要求的操作。我们通过对特例情况的考虑，寻找到了最后的解法，这实际上是建立了特例与推广的平衡，从而有效地解决了这个问题。虽然我们估计的最坏复杂度比较庞大，但是实际运行效果相当优秀，而且程序实现方面也相对简单。虽然没有精细地计算出时间复杂度的上限，但这并不妨碍这个算法的实用价值。

实际上，ural某次比赛中也曾出现过这题（ural1553)，在比赛中，笔者也正是用这种方法第一个AC此题，也正说明这种方法是行之有效的。

三、总结

我们知道有一个经典的木桶：一个木桶由许多块木板组成，如果组成木桶的这些木板长短不一，那么这个木桶的最大容量不取决于长的木板，而取决于最短的那块木板。如果把时间复杂度，空间复杂度等都看作是解决问题这个木桶的一个木板，那么我们常常忽略的便是思维与实现复杂度这块木板。为了增加容量，不妨截取较长的木板来拼接到短的木板上，所有木板长度相同时容量也达到了最大。平衡思想正是这样一种取长补短的思想。

应用平衡思想解决的问题有一个特征：存在瓶颈。这瓶颈通常是阻碍我们解决问题的主要障碍。本文就对可应用于这类的平衡思想作了一定的介绍，同时较深入地讨论了一类以其他方面为代价，降低问题的思维复杂度与程序的实现复杂度的模型。通过构建各方面复杂度的平衡，我们能有效地在有限的时间里尽可能多地解决问题。赛场上，我们不必追求最完美的时空复杂度，因为高分才是硬道理，AC才是硬道理！

外在看来，平衡规划更像是一种“骗分”思想。但笔者认为，解决信息学问题并不能只是单纯追求时间复杂度的完美，在有限的时间内要综合考虑各方面因

素所带来的问题，这也正是信息学竞赛的魅力所在。合理应用我们所学的知识并将他们有效结合，合理决策对问题的数学模型，算法各方面（包括设计，实现）的精力分配，才能有效地解决问题。当然，即使如此，也需要我们有扎实的基本功，才能在赛场上能有更多的选择。而且这类思想需要我们不断地练习才能较好地掌握，实践才是感悟这一类问题最有效的途径。毕竟，纸上得来终觉浅，绝知此事要躬行。

四、感谢

感谢 陈丹琦 同学 提供部分资料并对本文的实现与修改提出建议；
感谢 董华星 同学 为本文的修改提出建议；
感谢 刘　弈 同学 为本文的修改提出建议；
感谢 刘汝佳 教练 对本文的实现与修改提出建议；
感谢 陈　光 老师 为本文的修改提出建议；
感谢 胡山立 老师 为本文的修改提出建议。

五、参考文献

[1]《算法艺术与信息学竞赛》　刘汝佳 黄亮 著

[2]《<Query on a tree（spoj375）>解题报告》　周戈林

[3]《Catch 题目分析》　许智磊

六、附录

例一的原题：ural1099. Work scheduling

There is certain amount of night guards that are available to protect the local junkyard from possible junk robberies. These guards need to scheduled in pairs, so that each pair guards at different night. The junkyard CEO ordered you to write a program which given the guards characteristics determines the maximum amount of scheduled guards (the rest will be fired). Please note that each guard can be scheduled with only one of his colleagues and no guard can work alone.

Input

The first line of input contains one number $N \leqslant 222$ — this is the amount of night

guards. Unlimited number of lines consisting of unordered pairs (*i*, *j*) follow, each such pair means that guard #*i* and guard #*j* can work together, because it is possible to find uniforms that suit both of them (The junkyard uses different parts of uniforms for different guards i.e. helmets, pants, jackets. It is impossible to put small helmet on a guard with a big head or big shoes on guard with small feet). The input ends with Eof.

Output

You should output one possible optimal assignment. On the first line of the output write the even number *C* — the amount of scheduled guards. Then output *C*/2 lines, each containing 2 integers (*i*, *j*) that denote that *i* and *j* will work together.

Sample

Input	Output
3 1 2 2 3 1 3	2 1 2

Problem Author: Jivko Ganev

例二的原题：PKU2103 Jackpot

Description

The Great Dodgers company has recently developed a brand-new playing machine. You put a coin into the machine and pull the handle. After that it chooses some integer number. If the chosen number is zero you win a jackpot. In the other case the machine tries to divide the chosen number by the lucky numbers p1 , p2 , . . . , pn . If at least one of the remainders is zero --- you win.

Great Dodgers want to calculate the probability of winning on their machine. They

tried to do it, but failed. So Great Dodgers hired you to write a program that calculates the corresponding probability.

Unfortunately, probability theory does not allow you to assume that all integer numbers have equal probability. But one mathematician hinted you that the required probability can be approximated as the following limit:

$$\lim_{k\to\infty}\left(S_k / 2k+1\right)$$

Here Sk is the number of integers between -k and k that are divisible by at least one of the lucky numbers.

Input

Input file contains n --- the number of lucky numbers (1 <= n <= 16), followed by n lucky numbers (1 <= pi <= 10^9).

Output

It is clear that the requested probability is rational. Output it as an irreducible fraction.

On the first line of the output file print the numerator of the winning probability. On the second line print its denominator. Both numerator and denominator must be printed without leading zeroes. Remember that the fraction must be irreducible.

Sample

Input	output
2 4 6	1 3

Source

Northeastern Europe 2004, Northern Subregion

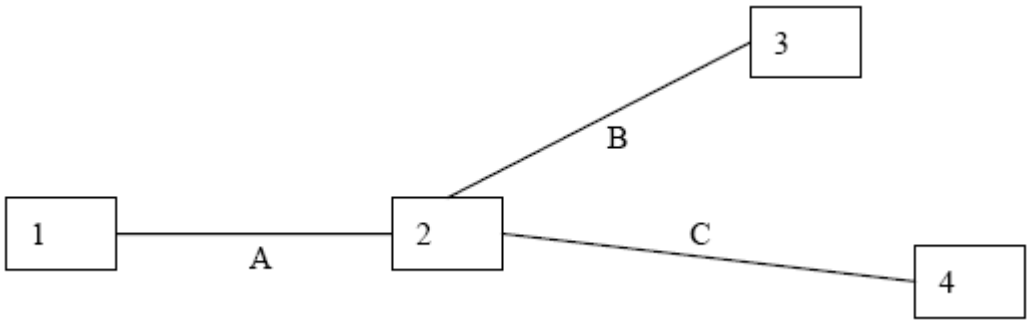
例三的原题：追捕盗贼

问题描述

魔法国度Magic Land 里最近出现了一个大盗Frank，他在Magic Land 四处作案，专门窃取政府机关的机密文件（因而有人怀疑Frank 是敌国派来的间谍）。为了捉住Frank，Magic Land 的安全局重拳出击！

Magic Land 由N 个城市组成，并且这N 个城市又由恰好N-1 条公路彼此连接起来，使得任意两个城市间都可以通过若干条公路互达。从数据结构的角度我们也可以说，这N 个城市和N-1 条公路形成了一棵树。

例如，下图就是Magic Land 的一个可能格局（4 个城市用数字编号，3 条公路用字母编号）：



大盗Frank 能够在公路上以任意速度移动。

比方说，对于上图给出的格局，在0.00001 秒钟内（或者任意短的一段时间内），Frank 就可以从城市1 经过城市2 到达城市4，中间经过了两条公路。

想要生擒Frank 困难重重，所以安全局派出了经验丰富的警探，这些警探具有非凡的追捕才能：

1.只要有警探和Frank 同处一个城市，那么就能够立刻察觉到Frank，并且将其逮捕。

2.虽然Frank 可以在公路上以任意快的速度移动，但是如果有警探和Frank在同一条公路上相遇，那么警探也可以立刻察觉到Frank并将其逮捕。

安全局完全不知道Frank躲在哪个城市，或者正在哪条公路上移动，所以需要制定一个周密的抓捕计划，计划由若干步骤组成。在每一步中，可以做如下几件事中的一个：

1.在某个城市空降一位警探。警探可以直接从指挥部空降到Magic Land的任

意一个城市里。此操作记为“L x”，表示在编号为x的城市里空降一位警探。耗时1秒。

2.把留在某个城市里的一位警探直接召回指挥部。以备在以后的步骤中再度空降到某个城市里。此操作记为“B x”。表示把编号为x 的城市里的一位警探召回指挥部。耗时1秒。

3.让待在城市x 的一位警探沿着公路移动到城市y，此操作记为“M x y”。耗时1 秒。当然，前提是城市x 和城市y 之间有公路。如果在警探移动的过程中，大盗Frank 也在同一条公路上，那么警探就抓捕到了Frank。

现在，由你来制定一套追捕计划，也就是给出若干个步骤，需要保证：无论大盗Frank一开始躲在哪儿，也无论Frank 在整个过程中如何狡猾地移动（Frank 大盗可能会窃取到追捕行动的计划书，所以他一定会想尽办法逃避），他一定会被缉拿归案。

希望参与的警探越少越好，因为经验丰富的警探毕竟不多。

例如对于前面所给的那个图示格局，一个可行的计划如下：

1. L 2 在城市2 空降一位警探。注意这一步完成之后，城市2里不会有Frank，否则他将被捉住。

2. L 2 再在城市2 空降一位警探。

3. M 2 1 让城市2 的一位警探移动到城市1。注意城市2 里还留有另一位警探。这一步完成之后，城市1 里不会有Frank，公路A 上也不会有Frank。也就是说，假如Frank 还没有被逮捕，那么他只能是在城市3 或城市4 里，或者公路B 或公路C 上。

4. B 1 召回城市1 的一位警探。注意虽然召回了这位警探，但是由于我们始终留了一位警探在城市2 把守，所以Frank 仍然不可能跑到城市1 或者是公路A 上。

5. L 3 在城市3 空降一位警探。注意这一步可以空降在此之前被召回的那位警探。这一步完成之后，城市3 里不会有Frank，否则他会被捉住。

6. M 3 2 让城市3 里的一位警探移动到城市2。这一步完成之后，如果Frank 还没有被捉住，那他只能是在公路C 上或者城市4 里。注意这一步之后，城市2 里有两位警探。

7. M 2 4 让城市2 里的一位警探移动到城市4。这一步完成之后，Frank 一定会被捉住，除非他根本就没来Magic Land。

这个计划总共需要2 位警探的参与。可以证明：如果自始至终只有1 名或者更少的警探参与，则Frank 就会逍遥法外。

你的任务很简单：对于一个输入的Magic Land 的格局，计算S，也就是为了追捕Frank 至少需要投入多少位警探，并且给出相应的追捕计划步骤。

输入文件

输入文件给出了Magic Land 的格局。

第一行一个整数N，代表有N 个城市，城市的编号是1~N。

接下来N-1 行，每行有两个用空格分开的整数x_i，y_i，代表城市x_i，y_i 之间有公路相连。保证$1 \leqslant x_i, y_i \leqslant N$。

输出文件

向输出文件输出你所给出的追捕计划。

第一行请输出一个整数S，代表追捕计划需要多少位警探。

第二行请输出一个整数T，代表追捕计划总共有多少步。

接下来请输出T 行，依次描述了追捕计划的每一步。每行必须是以下三种形式之一：

“L x”，其中L 是大写字母，接着是一个空格，再接着是整数x，代表在城市x 空降一位警探。你必须保证$1 \leqslant x \leqslant N$。

“B x”，其中B 是大写字母，接着是一个空格，再接着是整数x，代表召回城市x 的一位警探。你必须保证$1 \leqslant x \leqslant N$，且你的计划执行到这一步之前，城市x 里面确实至少有一位警探。

“M x y”，其中M 是大写字母，接着是一个空格，再接着是整数x，再跟一个空格，最后一个是整数y。代表让城市x 的一位警探沿着公路移动到城市y。你必须保证$1 \leqslant x, y \leqslant N$，且你的计划执行到这一步之前，城市x 里面确实至少有一位警探，且城市x, y 之前确实有公路。

必须保证输出的S 确实等于追捕计划中所需要的警探数目。

样例输入	样例输出
4	2

1 2 3 2 2 4	7 L 2 L 2 M 2 1 B 1 L 3 M 3 2 M 2 4

评分标准

对于任何一个测试点：

如果输出的追捕计划不合法，或者整个追捕计划的步骤数T 超过了20000，

或者追捕计划结束之后，不能保证捉住Frank，则不能得分。

否则，用你输出的S 和我们已知的标准答案S*相比较：

1. 若S<S*，则得到120%的分。
2. 若S=S*，则得到100%的分。
3. 若S*<S≤S*+2，则得到60%的分。
4. 若S*+2<S≤S*+4，则得到40%的分。
5. 若S*+4<S≤S*+8，则得到20%的分。
6. 若S>S*+8，则得到10%的分。

数据规模和约定

输入保证描述了一棵连通的N 结点树，1≤N≤1 000。

例题五原题：PKU3237 tree

Description

You are given a tree with N nodes. The tree's nodes are numbered 1 through N and its edges are numbered 1 through $N - 1$. Each edge is associated with a weight. Then you are to execute a series of instructions on the tree. The instructions can be one of the following forms:

CHANGE *i v*	Change the weight of the i_{th} edge to *v*
NEGATE *a b*	Negate the weight of every edge on the path from *a* to *b*
QUERY *a b*	Find the maximum weight of edges on the path from *a* to *b*

Input

The input contains multiple test cases. The first line of input contains an integer *t* (*t*<=20), the number of test cases. Then follow the test cases.

Each test case is preceded by an empty line. The first nonempty line of its contains *N* (*N* <= 10,000). The next *N* − 1 lines each contains three integers *a*, *b* and c, describing an edge connecting nodes *a* and *b* with weight *c*. The edges are numbered in the order they appear in the input. Below them are the instructions, each sticking to the specification above. A lines with the word "DONE" ends the test case.

Output

For each "QUERY" instruction, output the result on a separate line.

Sample Input

```
1
3
1 2 1
2 3 2
QUERY 1 2
CHANGE 1 3
QUERY 1 2
DONE
```

Sample Output

```
1
3
```

Source

POJ Monthly--2007.06.03, Lei, Tao