根号算法——不只是分块

南京外国语学校 王悦同

摘 要

根号算法逐渐普及，OI题目的复杂度也变得越来越“多样”；不再仅仅是简单的O(N),O(N^2),O(NlogN)这种复杂度了。很多题目复杂度里出现了根号，其中数据结构题中的根号算法最为常见，例如分块维护或者“莫队算法”——去年已经有同学详细介绍过了。但是，根号算法绝不仅限于这些地方，它们在很多其它的算法中都会出现，伴随着不同的新奇思想。而对这些思想深入研究，更是能挖掘出比根号更优的复杂度的应用。本文介绍了几种根号算法出现在非数据结构题中的应用，并试图移植这种思想获得更优的算法。

1 算法组合

1.1 从一道水题说起

【例1】Codeforces #80 Div1 D

给出一个长度为N（⩽300000）的数列a[i]，再给出M（⩽300000）个询问，每个询问是形如(x,y)的形式，你需要输出a[x]+a[x+y]+a[x+2y]+...+a[x+ky]的和，其中x+(k+1)y>N。时限4s。

这题的询问和以往不太一样：大多数同学应该有这样的感觉，我们所学过的大多数数据结构都擅长处理“连续区间”的询问，而不擅长间隔的位置之间的询问。对于本题，线段树之类的传统数据结构难以胜任。但我们发现，假设所有询问的y都一样，那么我们就可以把原序列按下标mod y的值分成y组，询问在对应的组里就是连续的了——而且只需要部分和维护。

这样的问题就是，我们对于y=1要维护若干个数组，size总和为N；对

于y=2也要，y=3、4、一直到N都需要，难以接受。但我们也不难想到，如果y很大，那么为什么要预存呢？直接暴力不是很快么？每次暴力扫描所有位置是O(N/y)的。因此，取K= $\sqrt{N}$，我们对于1⩽y⩽K的情况预存部分和，对于y>K的情况暴力扫描，就可以保证算法的复杂度为O(N $\sqrt{N}$)了。

我们回顾一下这题。这题不能完全说不是“数据结构”，但它体现的是一种和数据结构题中的根号截然不同的思想。对于题目中的某两个约束，它们是互相制约的，并且这种制约是“乘积”关系——步长和项数，总有一个不能太大。而我们针对两个情况分别设计算法，然后将两种算法组合起来，就能获得一个完整的解决问题的算法。下面看另一道这种思路的简单应用：

【例2】Topcoder SRM589 Level-3 FlippingBits

给一个长度为N（N⩽300）的01字符串，再给一个正整数M。每次操作，你可以将一个位置取反，或者将一个长度为M的倍数的前缀取反。问最少需要多少次操作，才能使字符串成为一个循环节为M的循环串。

循环节长度为M定义为，对于任意i，如果i+M这个位置存在，那么i和i+M上的字符应该相等。

本题也不是很难。我们考虑循环节的长度和循环节的个数——这两个乘积（大约）是原串的长度，也就是他们是互相制约的，总有一个不超过 $\sqrt{N}$。因此这启示我们分别针对两种情况设计“专杀”算法。首先考虑循环节长度不是很大的情况。这里指的是循环节长度不超过17（也就是300的平方根的近似值）。这个时候我们发现，因为答案要求每一个循环节都一样，而循环节又不长，所以直接枚举：2^{17}。枚举出答案以后，每一段必须和答案相等、或者和答案的反串相等。这里很简单，只需要一个动态规划即可，因为注意到记原串每一段是否整体反转，那么操作“将一个长度为M的倍数的前缀取反”的执行次数就是有多少段连续的0和1。

下一种情况就是循环节个数不是很多。这个时候，由于操作“将一个长度为M的倍数的前缀取反”只会在不超过 $\sqrt{N}$个位置进行，故仍然可以枚举。这个时候，我们已知每一段是否反转，要构造一个答案来“迎合”它们——这也很简单，对于答案串的每一位，如果所有循环节在这一位的0比较多，这一位就是0；1比较多就是1。因此整个题目就解决了，复杂度为O($N * 2^{\sqrt{N}}$)。

我们再来简单回顾这题，和上一题还是比较相似的。通过组合两个“朴素”的算法，得到一个相对高效的做法。更重要的是，这样的“组合算法”解题，几乎是无可替代的——有些题唯一的做法就是通过组合各种算法；或者只有这样才能降低复杂度。这样的做法能扩展到更加复杂的题目上去。

1.2　进一步应用

【例3】IOI2009 Regions

N个节点的树，有R种属性，每个点属于一种属性。有Q次询问，每次询问r1,r2，回答有多少对(e1,e2)满足e1属性是r1，e2属性是r2，e1是e2的祖先。

N,Q⩽200000，R⩽25000，时限5s

我初看这题感觉是很正统静态树上的询问（幸好题目要求在线，不然我应该会想的更偏）。不过我想了很久类似$O(Nlog^2N)$之类的算法却毫无进展。后来我看了一眼题解，只看了一点点，立即恍然大悟——题解中告诉我，考虑针对询问的点的个数不同设计不同的算法。

不妨设询问中r1的属性的点有A个，r2属性的点有B个。如果A不大、而B很大呢？我们可以允许扫描A，但是不能扫描B。一种可行的做法如下：考虑dfs序，A中每个节点的子树在dfs序中都对应了一段区间[l,r]，而B中每个节点都对应了一个dfs序中的位置p。如果l⩽p⩽r，那么可以知道A是B的祖先，反之就不是。所以，对于每种属性的节点开一个Vector，将这种属性的所有点的dfs序标号按顺序存进来，那么我们枚举A中的每个点，只要在B对应的vector里二分查找即可。这个算法并不难实现，但是却可以在O(AlogB)的时间内完成一个询问。

那么如果A很大但B不大呢？从直觉来看，我们应该设计一个O(BlogA)的算法。这样的算法确实也是能设计出来的，尽管有点复杂。例如，大概可以这么做：模仿刚刚AlogB的做法，我们扫描B，然后看这个点覆盖了多少区间。因此可以先将该组A个区间全部预处理好，把数组中对应的一段+1，例如有一个区间[5,10]，就把数组5至10这一段都+1。这样我们可以直接看这个位置的数而确定这个点被多少区间覆盖了。A个区间端点可以离散化、求和也可以用差分等

方法进行，总之也能在O(BlogA)内完成。

但是我设计出O(BlogA)的做法后发现了更严重的问题。

第一个问题就是，如果询问的A和B都很大，那么无论如何也不可能优于O(A+B)了；A和B都最大可以是N，如果每个询问都这么大，不就直接超时没有优化的余地了么？

但实际上解决这个问题的方案却异常直观。想象一下，一个询问涉及的点数非常多，那么这样的询问不可能有太多；如果有两个询问一样我们必然可以直接查询以前的答案。所以关键就是，同一个询问不做两次。那么不同的询问呢？直接做！复杂度将在后面给出证明。

第二个问题就是，现在即便已经有O(AlogB)和O(BlogA)的做法了，但仍然有这种情况：（假设N和Q是一样的，因为它们是一个数量级），有 $\sqrt{N}$种属性，每种有 $\sqrt{N}$ 个，然后N（也就是Q）个询问，把两两属性之间都问了一遍。这个时候不难发现算法复杂度退化到了O(N $\sqrt{N}$logN)，对于N=200000来说一看就是不可接受的。

通过之前的分析，我们发现，A和B一个很大一个很小的情况是不难办的。但是近似大小的时候呢？我们应该有这样一种感觉，如果我们允许A和B都扫一遍，应该可以设法避免这个log的存在。实际上确实想下去就发现不难了——把A视为区间，B视为点，要求每个点被多少区间包含了，这个实际上可以吧“区间首段点”和点一起做归并，然后顺便维护一个栈就行了，因为A中的区间互不包含，有很多显然的单调性。

最后就是复杂度分析。我们设一个阈值C，对于A⩾C的情况，使用O(BlogA)的做法，对于B⩾C的情况使用O(AlogB)的做法，复杂度最坏O(N^2logN/C)；对于其它情况用O(A+B)的做法，复杂度是O(NC)。解出来C= $\sqrt{NlogN}$。这里把N和Q混在一起分析了，但他们是一个数量级，所以这么分析没啥问题。整个算法复杂度是O(N $\sqrt{NlogN}$)，可以接受。

刚刚这题就比以往复杂多了；即便是放在当时IOI赛场，也绝非容易。多种算法的取长补短以保证复杂度的思想，在刚刚那题中得到了很好的体现。

2 另类平衡

2.1 分类维护

“分类维护”是一种数据结构题中的另类思想。同样是根号算法，它和“分块维护”在思想上还是有很大差异的。我们通过一道例题来简要的说明一下这种做法的含义和应用。

【例4】 给出一个N个点M条边的无向图，N,M⩽100000。一开始所有的点都是黑色。每次你可以进行两种操作：将一个点反色，或者查询某个点直接连边的点中有多少个黑点。时间限制1s。

本题确实不难，但体现了一种很好的思想。我们发现，在大多数情况下（随机情况下），直接暴力扫描一个点的出边是很快的。为什么？因为出边少——虽然理论上最坏是O(N)的，但是很难达到。如果一个点出边不多，那么我们就直接询问到的时候枚举即可。

如果出边很多咋办呢？我们应该有这样的反应——一个多，另一个就少；出边多，这样的点就少。我们不妨以$\sqrt{N}$为界来讨论（一般都可以以$\sqrt{N}$为界讨论，等到最后卡复杂度的时候再仔细估算阈值大小）。如果一个点度数⩽$\sqrt{N}$，那么直接维护，复杂度是O(M $\sqrt{N}$)；如果度数大了，则这样的点最多$\sqrt{N}$个，我们可以直接每次修改后都暴力维护一遍这些点的答案——很简单，一旦修改，就扫描这$\sqrt{N}$个点，看看这次修改和它是否有关。这个也显然能在O($\sqrt{N}$)时间内维护。因此本题通过组合这两种情况的维护方法，就可以实现O(N $\sqrt{N}$)的复杂度。

2.2 轻松一刻

下面我们来看一个很简单有趣的题目放松一下。

【例5】烧桥计划（JSOI2013互测）

给一个长度为N的数列（N⩽100000) $A_1,A_2...A_n$ (1000⩽A_i⩽2000)。你需要从中选出若干个数（个数不限，也可以不选），记为K 个，分别为$A_{p_1},A_{p_2}...A_{p_K}$（p数列严格递增）。这次选数的代价是$A_{p_1}$*1+$A_{p_2}$*2+...+$A_{p_K}$*K。

这些选出来的数把剩下的N-K个数分成了K+1 段（可能有空段）。记每一段中A数列的和为T_i,对于所有T_i>M的段，产生额外的T_i的代价；否则没有任何额

外代价。对所有的T_i，M都是一样的，输入的数⩽2*10^8。求一种选数方案，使得总代价最小。时限7s。

我们不难设计出一个二维的状态f[i][j]，表示当前最后一个选择的位置是i，且这个位置是从左到右第j个选择的位置，此时j个位置和j段数生成的总代价，最少是多少。因此可以写出状态转移方程：记sum[i]表示前i个A数组的和，

F[i][j]=min(F[k][j-1]+(A_i)*j) for sum[i-1]-sum[k]⩽M

F[i][j]=min(F[k][j-1]+(A_i)*j+sum[i-1]-sum[k]) for sum[i-1]-sum[k]>M

其中0⩽k<i

对这个动态规划稍加分析，就可以发现利用单调队列可以将复杂度降到O(N^2)——剥离变量后直接维护，因为比较显然，所以这里不详细说明了。关键是原题是N=100000，这个做法仍然太慢。哪儿还能优化呢？

这题是我在JSOI2013的省队互测中出的一道“搞笑”题，现场没有人做出来；其实不是难题，而是没有反应过来“脑筋急转弯”。原题中有一个条件，1000⩽A_i⩽2000。上界正常，下界有什么用呢？注意到题目中选一个A_i的代价其实是越来越大的，所以我们不能选太多!

对于一个数都不选的情况，总代价⩽2000*N（只有一段）

对于选了i个数的情况，总代价⩾i*(i+1)/2*1000⩾500*i*i

要使得选i个数时答案可能成为最优解，必须有500*i*i⩽2000*N

即i⩽ $\sqrt{N}$*2

因此状态个数是O(N $\sqrt{N}$)的，即可通过本题了。

我把这题放在这儿也并非和其它题没有关联；毕竟，它也是一个O(N $\sqrt{N}$)的算法，根号的来源也同样奇特——来源于对题目中下界的分析。我当时出这题时受到topcoder某道题的启发：那题是说，给出K个点分成N层，相邻层之间有些点有边，第一层和最后一层也有一些边。求这个图的最大独立集。K⩽100，10⩽N⩽100。这题中“最大独立集”明显是需要利用二分图的性质进行匹配，但如果N是奇数则不是二分图，怎么办？注意到题目中N不小于10，而点数不多于100，所以一定有一层点数不超过10。我们枚举这一层的所有情况，就可以“破环为链”了。这题和主题关系不大，但还是顺便介绍一下；毕竟如果上界和下界可以视为一个数量级（差小常数倍，例如原题差2倍），则经常可以通过上下界的分析得到和根号有关的性质，作为解题的突破口。我们再回到原主题：

2.3 内存优化

也许卡内存的情况在OI中并不常见。但既然时间可以有这样的优化，内存为何不可以呢？通过类比的扩展，内存也可以得到类似的优化，而且幅度并不小——所以其实也是一种很有用的思路。我们通过一道题目来看一下这种应用。

【例6】Codeforces #79 Div1 E

你现在有N（⩽20000）颗石子和M（⩽20000）颗糖果，准备把它们吃完。现在有N个参数A[i]和M个参数B[j]，每次你可以选择吃一个糖果或者一个石子，吃完之后，如果剩下来x颗石子和y颗糖果，你就可以得到(A[x]+B[y])mod P的奖励（P也是给定的），然后你继续吃。你要最大化M+N次得到的所有奖励之和。你需要输出方案：每次是吃糖果还是石子。时间限制15s，内存限制45MB。

如果不考虑内存限制的问题，这题还是很容易的。一个动态规划：F[i][j]表示吃了i个糖果、j个石子时的最优答案。F[i][j]=(A[N-i]+B[M-j])mod P+max(F[i-1][j],F[i][j-1])。这应该大家都会；至于记录方案，只要另一个数组G记录每个位置从哪儿转移来的就行了。

那么回来看内存限制呢？大家应该都能想到F数组可以滚动数组、而G可以压位。不过算了一下发现这样也需要五十多兆内存……所以考虑另辟蹊径。

直接存下这个表内存是不够的。但我们为啥要全部存下来呢？例如，我们如果记录了倒数第K行的答案，那么从第K+1行到第N行的空间就可以和前面“公用”，因为我把这段的答案得出来后直接输出，以后就规约为从第K行往前的答案的问题了。受此启发，我们考虑适当存一些行的答案，以尽可能公用空间。

想到这里就不难想到根号了；凭直觉来也应该是每$\sqrt{N}$行存一次答案（也就是F数组，这里不滚动了）。这样我们只需要O(N$\sqrt{N}$)的空间存储这些信息，就可以分段恢复答案了！我们要f[M][N]的答案，但我们既然有了M-$\sqrt{N}$这一行的答案，就可以从这一行开始dp，然后得出这一段的“最优路径”；然后同样的空间还可以恢复上面$\sqrt{N}$行的答案，再往上，以此类推。这是一种另类的平衡。虽然理解起来不困难，但思维还是很新奇的，是根号思想的一种扩展应用。整个算法复杂度仍然是O(N^2)，但空间是O(N$\sqrt{N}$)了。

3 自然出现

也有一些时候，我们并不是刻意去让根号出现在算法里，但是它却会无意中就出现了。举个例子：N div i（整除）当i取遍1至N时，只有O($\sqrt{N}$)种不同的取值。这个并不难证明：当i⩽$\sqrt{N}$时，最多也就$\sqrt{N}$个不同的答案；而当i>$\sqrt{N}$时，答案落在[1, $\sqrt{N}$]的区间内，也最多$\sqrt{N}$个。这个性质有一些应用；下一个例子介绍了一下这一类"自然出现"的$\sqrt{N}$如何应用起来。

【例7】POI2007 Queries

有N（⩽50000）组询问，对于给定的整数a,b和d，有多少正整数对x,y，满足x⩽a，y⩽b，并且gcd(x,y)=d。

本题是一个数学类问题。首先分析一下题目，gcd(x,y)=d这个可以直接转化为gcd(x,y)=1，并且把a和b的范围同时缩小为a/d和b/d。也就是说，我们要求出有多少互质的数对，一个不超过a，一个不超过b。这和NOI2010的能量采集十分像，可以直接通过容斥原理解决。具体做法就是，设F(x)表示有多少对数，一个不超过a，一个不超过b，并且它们的gcd 是x的倍数。这很简单——F(x)=(a div x)*(b div x)。接下来用容斥即可，答案=F(1)-F(2)-F(3)-F(5)+F(6)-F(7)……，其中F(x)加到答案中前面有一个系数p(x)，如果x中某种质因子含有超过一个，则p(x)=0，否则p(x)=$(-1)^G$，其中G表示x中不同质因子的个数。直接这么做是每个询问O(N)的，相比O(N^2)的暴力确实好得多，但仍然复杂度不满足要求。

我们观察式子，发现我们要做的其实就是，尽快求出∑F(i)*p(i)。p(i)规律并不明显，但F(i)呢？之前我们说过，N div i的取值个数是O($\sqrt{N}$)级别的，那么F(x)和它有直接的关联——a div x和b div x都只有O($\sqrt{N}$)级别的不同取值，所以F(x)取值也是同一个数量级的!

对于连续一段相邻的F(x)，如果他们相等，就可以一并计算。这里仅仅需要维护一个p(x)数组的部分和就行了。通过维护p(x)的部分和和将相同的F(x)一起计算，我们成功的做到了在O($\sqrt{N}$)的时间内回答一次询问，因此可以通过本题。

本题中所说的p(x)这个函数还有一个名字叫"Mobius函数"，去年集训队王迪同学的论文里讲到过。这里并没有深入挖掘它在数学上的更多性质和广泛应用，只是以此为例，指出了一类利用"自然"存在的性质对算法进行优化的思路。

参考文献

[1] 《21-st International Olympiad In Informatics TASKS AND SOLUTIONS》from www.ioi2009.org

[2] 《POI.XIV.sol》2008国家集训队作业，成都七中-周梦宇

[3] 《浅谈容斥原理》2013国家集训队论文，成都七中-王迪