NOIP2015 模拟赛 题解

By nodgd

1. 好数

1.1. 题目大意

由小到大输出不小于n的m个三进制表示里数字1的个数和数字2的个数一样多的十进制数（即好数）。

1.2. 算法一

手动算出100以内的所有好数，再手动算10个出来。每次提问直接找出答案输出。可以得到50分。

1.3. 算法二

一个数的三进制表示可以将这个数不停地除以3，做除法时的余数序列就是这个数的三进制由低到高的每一位。这样就能很轻松的判断一个数是不是好数。

每次从n开始一个数一个数的枚举，每发现一个好数就输出。容易发现，好数其实很多，两个好数之间的间距也不会太大，这样就足以通过所有数据，得到100分。

2. 好文章

2.1. 题目大意

统计一个长度为n的只包含小写字母的字符串有多少个不同的长度为m的子串。

2.2. 算法一

枚举所有长度为m的子串，共有$n-m+1$个，每一个都去和以前枚举的子串进行比较，每次比较话费$O(m)$的时间，总的时间复杂度就是$O(n^2m)$，理论上可以得到30分，实际上可以得到50分或者更多。

2.3. 算法二

要比较两个字符串是否相同，一个常用的办法是字符串 Hash。字符串的常用 Hash 方法是多项式 Hash，

$$hashvalue[i] = (hashvalue[i-1] \times p + str[i])\%P$$

$$hashvalue(l,r) = (hashvalue[r] - hashval[l-1] \times p^{r-l+1})\%P$$

具体实现时随便找一个p和P，要求p,P互素，且p要比可能出现的字符数量大，也就是比26大，同时P尽量大，以减小 Hash 值出现偶然相同的的概率。具体地说，把两个本来不同的字符串判定为相同的概率是$\frac{1}{P}$。

在算法一中加入这个处理，就能快速比较两个串，时间复杂度为$O(n^2)$，可以得到50分。

2.4. 算法三

容易发现，当我们计算出 Hash 值之后，要做的事情实际上是判断$n-m+1$个数中有多少个不同的数，可以先排序再扫描依次来解决。所以，先用$O(n)$的时间复杂度求出这$n-m+1$个 Hash 值，然后排序再扫描，时间复杂度降低到了$O(n\log n)$，看似可以通过所有数据了！

然后就发现了一些问题。

第一个问题，很多人喜欢取$P=2^{64}$，也就是把$hashvalue[\,]$这个数组定义为64位无符号整数类型，从而省略取模的步骤，直接溢出就行了。这样的 Hash 是很容易卡掉的。假如一开始我们有一个长度为1的字符串 a，我们每次把上一次的字符串中的 a 和 b 相互替换之后拼接在上一次的字符串的后面，得到一个长度是上一次的两倍的字符串，例如

a
ab
abba
abbabaab
abbabaabbaababba
abbabaabbaababbabaababbaabbabaab
……

我们来计算一下 Hash 值，设长度为2^k的一个字符串是str_k，每一位的 a 和

b 取反之后是str'_k。

$$hashvalue(str_k) = hashvalue(str_{k-1}) \times p^{2^{k-1}} + hashvalue(str'_{k-1})$$

$$hashvalue(str'_k) = hashvalue(str'_{k-1}) \times p^{2^{k-1}} + hashvalue(str_{k-1})$$

两式相减

$$hashvalue(str_k) - hashvalue(str'_k)$$
$$= \left(hashvalue(str_{k-1}) - hashvalul(str'_{k-1})\right) \times \left(p^{2^{k-1}} - 1\right)$$

迭代下去可以得到

$$hashvalue(str_k) - hashvalue(str'_k)$$
$$= \left(hashvalue(str_0) - hashvalue(str'_0)\right)(p-1)(p^2-1)(p^4-1)\ldots\left(p^{2^{k-1}}-1\right)$$
$$= \left(hashvalue("a") - hashvalue("b")\right)(p-1)(p^2-1)(p^4-1)\ldots\left(p^{2^{k-1}}-1\right)$$
$$= -(p-1)(p^2-1)(p^4-1)\ldots\left(p^{2^{k-1}}-1\right)$$

要想让$P = 2^{64}$出错，就要让上面算出来这个数的素因子中含有至少 64 个 2。

$p-1$里至少有 1 个 2；
$p^2-1=(p-1)(p+2)$至少有 2 个 2；
$p^4-1=(p^2-1)(p^2+1)$至少有 3 个 2；
$p^8-1=(p^4-1)(p^4+1)$至少有 4 个 2；
……
$p^{2^{k-1}}-1=\left(p^{2^{k-2}}-1\right)\left(p^{2^{k-2}}+1\right)$至少有$k$个 2。

所以$hashvalue(str_k) - hashvalue(str'_k)$里有$\frac{k(k+1)}{2}$个 2，取$k=11$就有 66 个 2 了。所以只要构造这样的字符串，长度不小于$2^{11}=2048$，同时M也不小于2048，就能够于是卡掉了$P=2^{64}$。所以这样做只能得到70分。

第二个问题，很多人喜欢取常见的质数，例如10^9+7，10^9+9之类的数。这样也很容易被卡掉，不是因为太常见，而是因为这个质数不够大。

不妨认为每个字符串的 Hash 值是一个在区间$[0,P-1]$里的随机整数。这$n-m+1$个字符串互不相同，所以问题的答案应该是$n-m+1$。于是我们写出在区间$[0,P-1]$里k个随机整数互不相同的概率

$$1 \times \frac{P-1}{P} \times \frac{P-2}{P} \times \ldots \times \frac{P-k+1}{P} = \frac{P!}{P^k \times (P-k)!}$$

有一个著名的斯特林公式

$$n! \approx \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$$

带入得到

$$\frac{P!}{P^k \times (P-k)!} \approx \frac{\sqrt{2\pi P}\left(\frac{P}{e}\right)^P}{P^k \times \sqrt{2\pi(P-k)}\left(\frac{P-k}{e}\right)^{P-k}}$$

$$= \frac{\sqrt{P} \times P^{P-k}}{\sqrt{P-k} \times (P-k)^{P-k} \times e^k}$$

$$= e^{\frac{1}{2}\ln P + (P-k)\ln P - \frac{1}{2}\ln(P-k) - (P-k)\ln(P-k) - k}$$

取$P = 10^9, k = 10^5$，带入计算得到≈ 0.00673717，也就是说，这时只有 0.0067373717 的概率答案正确，这显然是不能接受的。

取$P = 10^{10}, k = 10^5$，带入计算得到≈ 0.60654916，比刚才大得多了，但是仍然不能接受。

取$P = 10^{12}, k = 10^5$，带入计算得到≈ 0.99610136，几乎是可以接受了。如果把P取到比10^{15}级别，出错的概率就足够小额，可以通过本题。另外，为了在计算$hashvalue[i] \times p$的时候不超出 64 位整数类型的范围,就需要$p \times P < 2^{64}$，所以取到10^{15}到10^{16}之间最佳。

这样的做法，根据P的大小和选手的运气，可以得到$50 \sim 100$ 分。但无论如何，$10^9 + 7, 10^9 + 9$之类的常用素数都只有 50 分。

2.5. 算法四

为了防止 Hash 被卡，我们也可以不适用 Hash。编写代码最简单的就是开一个 `set<string>`，每次把一个子串插入到 `set` 里面，最后 `set.size()`就是答案。时间复杂度$O(nm\log n)$，还是只有 50 分。

2.6. 算法五

除了 set，我们还有 Trie 呢！如果采用普通方法来保存 Trie 的边，时间复杂度$O(nm)$，空间复杂度$O(nm|\sum|)$，其中$\sum$表示字符集，$|\sum| = 26$，还是只有 50 分，而且可能会爆内存。如果采用链表来保存 Trie 的边，时间复杂度变成$O(nm|\sum|)$，空间复杂度变成了$O(nm)$，可以得到 70 分。

2.7. 算法六

算法三其实效率很高，就是可能会因为选择 Hash 参数时挂掉，我们有什么简单的办法补救呢？其实很简单，我们可以同时运行两个或者更多的算法三，比如第一个里面$p_1 = 29, P_1 = 10^9 + 7$，第二个里面$p_2 = 31, P_2 = 10^9 + 9$。如果两个子串的两种 Hash 值都相等，我们才认为它们是一样的，那么出错的概率就是$\frac{1}{P_1P_2}$，而k个串的 Hash 值互不相同的概率是

$$\frac{(P_1P_2)!}{(P_1P_2)^k(P_1P_2-k)!}$$

就相当于算法三中找了一个10^{18}级别的P，出错的概率足够小，可以轻松地得到100分。

2.8. 算法七（知识拓展）

以上算法都是基于 Hash 的，无论概率多小，都有一定的概率出错。那么，能不能找到一个算法，保证复杂度的同时能够确保不会出错呢？

答案是肯定的，那就是后缀数据结构大家族——后缀数组、后缀自动机、后缀平衡树、后缀树。除了后缀平衡树是$O(n\log n)$以外，其他三个都能做到$O(n)$的时间复杂度。由于这四个数据结构都比较复杂，超出了 NOIP 考试的难度范围，

在此不再赘述，有兴趣的小伙伴可以去网上找相应的论文、博客来观摩和学习。

2.9. 几句废话

其实，这道题是我当年膜拜完了这篇博客之后，一直心有余悸的从而搬出来的。在看过这篇博客之前，我一直写的是自然溢出的 Hash，后来就改成了取两个P的双 Hash，从此以后，妈妈再也不用担心我被卡 Hash。

希望大家做过这道题之后，不一定要知道每种 Hash 怎么卡，为什么被卡，但是一定要知道什么 Hash 可以轻松的被卡掉，以后自己写代码时注意避免。

3. 好路线

3.1. 题目大意

给定一个$n \times m$的数字矩阵，找出一条从左上角到右下角，只能向右或者向下走的路径，使路径上的$n+m-1$个数的方差最小。$1 \le n, m \le 50$, $0 \le h(x,y) \le 50$。

3.2. 算法一

用深度优先搜索来枚举所有的符合条件的路径，计算出方差并记录最小值。当$n, m \le 10$时，路径条数不超过组合数$\binom{19}{10}=92378$，显然可以很快的找出最小值。这样做可以得到 30 分。

3.3. 算法二

同样是深度优先搜索，但是我们希望能够找到方法可以剪枝。要想剪枝，就需要找到一个方法来评估当前的局面。我们知道，方差是衡量一些数与平均值的偏离程度的，偏离程度越高，方差就越大。假设现在已经用搜索确定了路径上的前k个数，当我们确定了所有数之后，方差至少是多少呢？显然，方差最小的情况就是后面的每个数都等于当前k个数的平均数的时候。于是我们找出了一个估算方法，可以算出方差的下界。计算出下界，就能够进行可行性剪枝了，如果这个下界已经大于之前已经求出的最小值，就不再枚举下去。加上这样的剪枝之后可以得到50分。

3.4. 算法三

我们发现，再往下优化很难用搜索在短时间内计算出$n, m \le 50$的答案，这说明我们不应该再往这方面思考，应该换一个角度寻求突破。我们先考虑从方差的表达式入手，设$N = n+m-1$

$$\begin{aligned}
& N^2\sigma^2 \\
&= N\left((x_1-\bar{x})^2+\cdots+(x_N-\bar{x})^2\right) \\
&= N\left((x_1^2-2x_1\bar{x}+\bar{x}^2)+\cdots+(x_N^2-2x_N\bar{x}+\bar{x}^2)\right) \\
&= N\left((x_1^2+\cdots+x_N^2)-2\bar{x}(x_1+\cdots+x_N)+N\bar{x}^2\right) \\
&= N(x_1^2+\cdots+x_N^2)-(x_1+\cdots+x_N)^2
\end{aligned}$$

设$a_k = x_1+\cdots+x_k, b_k = x_1^2+\cdots+x_k^2$，带入

$$\begin{aligned}
&= Nb_N - a_N^2 \\
&= N(b_{N-1}+x_N)-(a_{N-1}+x_N)^2 \\
&= (Nb_{N-1}-a_{N-1}^2)+Nx_N-2a_{N-1}x_N+x_N^2
\end{aligned}$$

所以，我们可以设定一个 DP 状态$f[x][y][sum]$，表示从左上角走到(x,y)处，路径上的$x+y-1$数的总和为sum时，$Nb_{x+y-1}-a_{x+y-1}^2$的值。容易写出状态转移方程

$$\begin{aligned}
& f[x][y][sum] \\
&= N \times h(x,y) - 2\left(sum-h(x,y)\right)\times h(x,y) + h^2(x,y) \\
&\quad + \min\{f[x-1][y][sum-h(x,y)], f[x][y-1][sum-h(x,y)]\}
\end{aligned}$$

转移的时间复杂度是$O(1)$的，所以 DP 的总时间复杂度就是状态的数量。$1 \le x \le n, 1 \le y \le m, 0 \le sum \le (n+m-1)\times \max\{h\}$，所以总的时间复杂度就是$O(nm(n+m)\max\{h\})$，可以得到 100 分。