C1 Unofficial Tutorial

September 7, 2022

A 小水獭和主定理

A.1 坑点 1

根据题目所给的式子，我们需要比较 $\log_b a$ 与 k 的大小关系。一个直接的想法是计算 $\log_b a$，但是注意到浮点数（换言之，小数）有一定的误差，如果出现诸如 $k = 2$ 但计算得到的 $\log_b a = 1.99 \cdots 98$ 的情况时会错误地判断大小关系。这种方法不一定可行。（事实上，确实不可行。）

为了抛弃小数带来的误差，我们想让计算中所有出现的数都是整数。为了实现这个目的，我们在两侧取指数，从而将比较 $\log_b a$ 与 k 的大小关系转化为比较 a 与 b^k 的大小关系。

A.2 坑点 2

如果直接计算 b^k，注意到 b, k 都可能达到 10^9 的范围，因此有两个问题：

- k 过大，直接将 b 连乘 k 次是不可行的，在时限内并不能计算得到。
- b^k 会超出 `int` 甚至 `long long` 的范围。

A.3 可行解法

可以连乘至多 k 次 b，如果中途乘得的结果已经大于 a 则立即得到 $b^k > a$，不必继续乘下去了。可以说明这样做至多需要 $\log_2 10^9 \approx 30$ 次乘法。

如果 $b^k \leq a$，此时不必担心上述两个问题，可以直接判断 b^k 与 a 的大小关系。

注意输出时可能需要对反斜杠号转义。

A.4 吐槽

A 题作为签到题一上来就卡人合理吗？出题人真的想让选手签上到吗？作为签到题，不如弱化一下数据范围。

另一个需要注意的问题是，如果想采取快速幂之类的方法计算 b^k，对中间结果取模显然是不可行的。可以对中间结果与 $10^9 + 1$ 之类的值取最小值，从而解决溢出 `long long` 的问题。

CF 的 div2A 可能都没这么多坑。

A.5 参考实现

```cpp
#include <cstdio>
#include <algorithm>

using namespace std;

int a, b, k;

const char *solve()
{
    scanf("%d%d%d", &a, &b, &k);
    k = min(30, k);
    long long pw = 1;
    for (int i = 1; i <= k; i++)
    {
        pw *= b;
        if (a < pw)
            return "n^k";
    }
    if (a == pw)
        return "n^k\\log n";
    return "n^{\\log_ba}";
}

int main()
{
    int T;
    scanf("%d", &T);
    while (T--)
        printf("%s\n", solve());
    return 0;
}
```

B 一元多项式计算器

B.1 可行解法

对于每组给定的 X_i, Y_i，直接带入 $f(x,y)$ 的定义式，分别计算

$$\sum_{i=0}^{n} a_i x^i, \quad \sum_{i=0}^{m} b_i y^i$$

两个式子的值，最后将其相乘即为答案。

计算一组 $f(X_i, Y_i)$ 的时间复杂度为 $\mathcal{O}(n+m)$，总时间复杂度为 $\mathcal{O}(q(n+m))$。

B.2 吐槽

本题给人的感觉比较神秘，非常小的值域让本人有解法与值域相关的错觉。

上述解法并未用到二元多项式的有关性质，不免怀疑二元多项式存在的必要性。不如只给一元多项式？

C 小水獭和模运算

C.1 坑点

一个简单的想法是，我们先计算 $\prod_i a_i$ 的值，再尝试对于每个 i 消除 a_i 带来的影响，从而计算得到 $f(i)$。但是很快我们会发现这种方法不可行，原因有二：

- a_i 可能等于 0；
- 模数是 10^9+6，并非质数。

C.2 可行解法

我们先将 $f(i)$ 写成比较容易理解的形式：

$$f(i) = a_1 \times a_2 \times \cdots \times a_{i-1} \times a_{i+1} \times a_{n-1} \times a_n$$

不妨记

$$p(i) = a_1 \times a_2 \times \cdots \times a_{i-1}$$

$$s(i) = a_{i+1} \times a_{n-1} \times a_n$$

就有

$$f(i) = p(i) \times s(i)$$

因此我们可以先预处理 $p(i), s(i)$ 的值，再直接计算 $f(i)$。

时间复杂度 $\mathcal{O}(n)$。

C.3 吐槽

考虑到坑点之后，本人第一次尝试的想法是将 10^9+6 分解质因数，考虑各个质因子的影响再合并，但过于复杂遂放弃。第二次尝试的想法是使用数据结构维护给定 a_i 的区间乘积，比较繁琐，但注意到需要查询的是前缀与后缀，从而可以得到解法。

本题作为第一次上机的签到可能并不合适，如果把坑点去掉或许更好。

D 日期统计

Tag：模拟

本题属于模拟题。一种可行但比较暴力的解法是，枚举年份与月份，计算得到对应月份的天数，再枚举日期判断其是否是好的。

好日期的条件乍看上去比较复杂。具体来说好日期满足的条件是：按位写开成 8 位，这 8 位从左往右先不降后不升，或是先不升后不降。感性理解来说，这 8 位数字是单峰的。判断是否是好日期可以利用前述的前缀后缀思想，也可以在遍历这 8 个数字时额外记录当前是不降段还是不升段来辅助判断。

期待更简单的解法。

E 单词游戏 – 青春版

Tag：字符串哈希

E.1 可行解法

本题难点在于判断哪些单词是重复的。上学期数据结构课的大作业或许有一些帮助。

一种比较简单的做法是，直接将英文单词通过哈希函数对应到整数，从而可以将哈希值作为数组下标来统计单词的出现次数。

具体来说，由于字符串中只会出现小写英文字母，并且长度不超过 4，我们可以先将 `a` 对应到 1，`z` 对应到 26，将空字符对应到 0。不妨记字符 c 对应到 $f(c)$，我们可以让单词 S 对应到

$$H(S) = 27^3 f(S_1) + 27^2 f(S_2) + 27 f(S_3) + f(S_4)$$

从而可以将单词区分开。

举个例子，对于单词 `buaa` 有

$$H(\texttt{buaa}) = 27^3 f(\texttt{b}) + 27^2 f(\texttt{u}) + 27 f(\texttt{a}) + f(\texttt{a}) = 2 \times 27^3 + 21 \times 27^2 + 1 \times 27 + 1 = 54703$$

而对于单词 `a` 有

$$H(\texttt{a}) = 27^3 f(\texttt{a}) + 27^2 f(\ \ ) + 27 f(\ \ ) + f(\ \ ) = 1 \times 27^3 + 0 \times 27^2 + 0 \times 27 + 0 = 1$$

可以发现，$H(S)$ 最大不会超过 $27^4 = 531441$，是可以储存得下的。

最后我们只需要统计每个人的单词出现的次数，并累加对应的分数即可得到答案。

如果对 C++ 的 STL 比较熟悉，可以使用 `string` 储存英文单词，使用 `map` 统计字符串出现次数。

还有字典树等做法，但可能比较复杂。

E.2 吐槽

本题原题是 CF 1722E，并不懂将长度限制从恰好为 3 加强到不超过 4 除了更难写了一点之外有什么意义（更青春了?）。

F 逆序 k 倍对

Tag：归并　数据结构

F.1 可行解法

本题与求逆序对有些类似。回忆学过的求逆序对的方法，不难想到归并排序求逆序对的方法。具体来说，可以在归并时对右半边的元素乘 k，利用单调性求出这部分的贡献，再继续归并排序。时间复杂度 $\mathcal{O}(n\log n)$。

具体细节可以参考下一页的实现。

其他做法对于现在的进度而言比较超纲。可以离散化后树状数组维护大小在某个区间中的数的个数，也可以直接动态开点线段树维护。时间复杂度是 $\mathcal{O}(n\log n)$ 或是 $\mathcal{O}(n\log a_i)$。

期待更好的解法。

F.2 需要注意的细节

答案可能会溢出 `int`，需要用 `long long` 类型储存答案。

F.3 吐槽

值域的范围是 $a_i < 2^{31}$ 会导致线段树里溢出 `int`，有些不太友好。

希望 $1 \le k \le 10$ 的限制是有作用的。

F.4 归并的参考实现

```cpp
#include <cstdio>

using namespace std;

const int N = 1e5 + 5;

int n, k;
int a[N], b[N];
long long ans;

void solve(int l, int r)
{
    if (l == r)
        return;
    int mid = (l + r) >> 1;
    solve(l, mid);
    solve(mid + 1, r);
    for (int i = l, j = mid + 1; i <= mid; i++)
    {
        while (j <= r && a[i] > 1ll * k * a[j])
            j++;
        ans += j - (mid + 1);
    }
    for (int i = l, j = mid + 1, p = l; p <= r; p++)
        if ((i <= mid && a[i] < a[j]) || j > r)
            b[p] = a[i++];
        else
            b[p] = a[j++];
    for (int i = l; i <= r; i++)
        a[i] = b[i];
}

int main()
{
    scanf("%d%d", &n, &k);
    for (int i = 1; i <= n; i++)
        scanf("%d", &a[i]);
    solve(1, n);
    printf("%lld\n", ans);
    return 0;
}
```

G　暗中观察

Tag：博弈

G.1　预备知识

首先我们需要定义状态，在本题中状态就是当前 n 的值。

我们称一个状态是**必胜态**，当且仅当在此状态下，接下来需要行动的一方存在一种策略，使得对方接下来无论如何行动，己方最终都能获得胜利。我们称一个状态是**必败态**，当且仅当在此状态下，接下来需要行动的一方无论采取何种策略，都无法获得胜利。

容易发现，必胜态的一方采取最优策略后，对方一定是必败态；必败态的一方无论采取何种策略后，对方都一定是必胜态。因此一个状态是必胜态，当且仅当存在一个行动后的状态是必败态；一个状态是必败态，当且仅当所有行动后的状态都是必胜态。

G.2　可行解法

我们考虑依次求出 $n = 0, 1, \ldots$ 时的状态时必胜态还是必败态。

不妨记录 $f(i)$，$f(i) = 1$ 表示状态 i 是必胜态，$f(i) = 0$ 表示状态 i 是必败态。当前状态为 n 时，下一轮对手可能得到的状态只有 $n - k^i \geq 0$，于是我们有

$$f(i) = \begin{cases} 1 & , \exists n - k^i \geq 0 : f(n - k^i) = 0 \\ 0 & , \text{otherwise} \end{cases}$$

边界条件是 $f(0) = 0$，这是因为 $n = 0$ 时接下来需要行动的一方已经输了。

单组数据的时间复杂度是 $\mathcal{O}(n \log k)$。

G.3　坑点

$k = 1$ 的时候，i 可以取遍 $\mathbb{N}$，但 k^i 恒等于 1。有些写法可能会炸，需要特判。

G.4　吐槽

本题是不是有一点超纲？埋坑会不会有些不太友好？

H XIAO7 想要变成猫猫

Tag：IQ Test

H.1 可行解法

本题不需要任何算法知识。

注意到最后会解密出 `0 0 0 0`，所以最后给出的四个数一定是最后一个问题的答案的相反数。

假设我们已经知道当前问题的答案是 ans，以及当前问题给出的 a_0, b_0, c_0, d_0，考虑推出当前问题的 $lastAns$，也即前一个问题的答案。根据给出的式子，有

$$\begin{aligned}(a_0 + lastAns) \times (x_{ans} \oplus ans)^2 + (b_0 + lastAns) \times (x_{ans} \oplus ans) \times x_{ans} \\ + (c_0 + lastAns) \times x_{ans}^2 + (d_0 + lastAns) = 0\end{aligned}$$

整理即可得到

$$lastAns = \frac{a_0 \times (x_{ans} \oplus ans)^2 + b_0 \times (x_{ans} \oplus ans) \times x_{ans} + c_0 \times x_{ans}^2 + d_0}{(x_{ans} \oplus ans)^2 + (x_{ans} \oplus ans) \times x_{ans} + x_{ans}^2 + 1}$$

从最后一个问题出发，依次得到前一问的答案，成功解答所有问题。

H.2 题外话

本题可能是本场比赛最有趣的题之一。

样例解释写道：

第四个的问题的答案为 1，第五个问题的输入 `-1 -1 -1 -1` 解密后为 `0 0 0 0`，成功解答所有问题。

这儿的成功解答所有问题或许是双关。

数据是如何保证 ans 一定是最小的 i？这也是一个有趣的问题。

本题和 J 题交换位置可能比较合适。

I 异或游戏

Tag：分类讨论

I.1 可行解法

通过搜索得到 $n \le 33$ 的答案，找规律可以得到答案为

$$\begin{cases} n & ,n \bmod 2 = 1 \\ n-1 & ,n \bmod 4 = 0 \\ n+3 & ,n \bmod 4 = 2 \land n = 2^k - 2 \quad (k \in \mathbb{N}) \\ n+2 & ,n \bmod 4 = 2 \land n \ne 2^k - 2 \quad (k \in \mathbb{N}) \end{cases}$$

I.2 证明

我们需要先做一些准备工作。

引理 $4k \oplus (4k+1) \oplus (4k+2) \oplus (4k+3) = 0 \quad (k \in \mathbb{N})$

证 容易验证 $0 \oplus 1 \oplus 2 \oplus 3 = 0$，$k = 0$ 时成立。

当 $k > 0$ 时，$4k, 4k+1, 4k+2, 4k+3$ 右移两位后的结果均为 k，这些位相同且被异或偶数次，因此异或结果中这些位为 0。又因为 $4k, 4k+1, 4k+2, 4k+3$ 二进制最低两位分别为 $0, 1, 2, 3$，可知异或结果中最低两位也为 0，从而有 $4k \oplus (4k+1) \oplus (4k+2) \oplus (4k+3) = 0$。 □

推论 1 $0 \oplus 1 \oplus \cdots \oplus (4k+2) \oplus (4k+3) = 0 \quad (k \in \mathbb{N})$

推论 2

$$0 \oplus 1 \oplus \cdots \oplus n = \begin{cases} n & ,n \bmod 4 = 0 \\ 1 & ,n \bmod 4 = 1 \\ n+1 & ,n \bmod 4 = 2 \\ 0 & ,n \bmod 4 = 3 \end{cases}$$

接下来我们来分类讨论。

答案下界为 $n-1$。若存在 n 个互不相同的不大于 $n-1$ 的非负整数异或和为 0，当且仅当 $0, 1, \ldots, n-1$ 的异或和为 0，当且仅当 $(n-1) \bmod 4 = 3$，也即 $n \bmod 4 = 0$。因此当 $n \bmod 4 = 0$ 时答案为 $n-1$。

对应序列为

$$0,\ 1,\ 2,\ \ldots,\ n-2,\ n-1$$

若存在 n 个互不相同的不大于 n 的非负整数异或和为 0，当且仅当 $0, 1, \ldots, n$ 的异或和不大于 n，由推论 2 可知当且仅当 $n \bmod 4 \ne 2$ 时满足条件。因此当 $n \bmod 4 = 1$ 或 $n \bmod 4 = 3$ 时答案为 n。

记 $0 \oplus 1 \oplus \cdots \oplus n = x$，对应序列为

$$0,\ 1,\ 2,\ \ldots,\ x-1,\ x+1,\ \ldots,\ n-1,\ n$$

接下来我们只需要考虑 $n \bmod 4 = 2$ 的情况。

我们先说明不存在 n 个互不相同的不大于 $n+1$ 的非负整数异或和为 0。若其存在，则应存在两个不同的非负整数 $a < b \le n+1$，使得 $a \oplus b = 0 \oplus 1 \oplus \cdots \oplus (n+1)$。这实际上是在考虑我们在 $0, 1, \ldots, n+1$ 这些数中不取哪些 a, b，剩余的 n 个数能满足条件。注意到等号右侧为 0，这说明 $a = b$ 从而导出矛盾。因此答案至少为 $n+2$。

若答案为 $n+2$，则应存在三个互不相同的非负整数 $a < b < c \le n+2$，使得 $a \oplus b \oplus c = 0 \oplus 1 \oplus \cdots \oplus (n+2) = n+2$。同样地，这是在考虑我们在 $0, 1, \ldots, n+2$ 这些数中不取哪些 a, b, c，剩余的 n 个数能满足条件。

当 $n+2 \ne 2^k\ (k \in \mathbb{N})$ 时，$n+2$ 的二进制有至少两个位值为 1，因此我们可以给出如下构造：

- 记 $n+2$ 二进制下最低为 1 的位是第 l 位（从 0 开始计数）。
- 令 $a = 0,\ b = 2^l,\ c = (n+2) \oplus 2^l$。

从而说明当 $n+2 \ne 2^k\ (k \in \mathbb{N})$ 时答案为 $n+2$。

对应序列为

$$1,\ 2,\ \ldots,\ b-1,\ b+1,\ \ldots,\ c-1,\ c+1,\ \ldots,\ n,\ n+1,\ n+2$$

当 $n+2 = 2^k\ (k \in \mathbb{N})$ 时，若存在非负整数 $a < b < c \le n+2$ 使得 $a \oplus b \oplus c = n+2 = 2^k$，$a \oplus b \oplus c$ 的二进制第 k 位应为 1，这迫使 $c = n+2$，从而有 $a \oplus b = 0$，而这与 $a < b$ 矛盾。因此当 $n+2 = 2^k\ (k \in \mathbb{N})$ 时，答案至少为 $n+3$。同样地，若答案为 $n+3$，则应存在四个互不相同的非负整数 $a < b < c < d \le n+3$，使得 $a \oplus b \oplus c \oplus d = 0 \oplus 1 \oplus \cdots \oplus (n+3) = 1$。注意到 $n+3 \ge 9$，因此我们可以给出如下构造：

- $a = 0,\ b = 2,\ c = 4,\ d = 7$

从而说明当 $n+2 = 2^k\ (k \in \mathbb{N})$ 时答案为 $n+3$。

对应序列为

$$1,\ 3,\ 5,\ 6,\ 8,\ 9,\ \ldots,\ n,\ n+1,\ n+2,\ n+3$$

至此完成分类讨论。

I.3 如何合理地搜索？

在固定序列长度为 n 的情况下，直接枚举每个位置放哪个数字是非常低效的。以下是一些合理的优化：

- 我们实际上只关心序列中有哪些数字，而不关心这些数字出现的次序。因此我们可以规定序列必须是单增的。
- 不妨记当前已知可取得的满足条件的序列最大值最小为 m。在搜索的过程中可以只在 $[0, m-1]$ 的范围中枚举加入序列的数字，因为我们只关心序列最大值能不能更小，序列中出现不小于 m 的数字对这个问题毫无帮助。

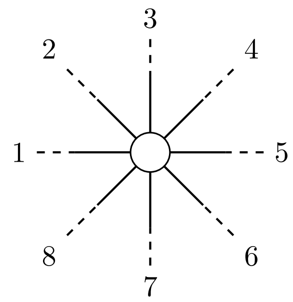
J　小水獭和二叉树

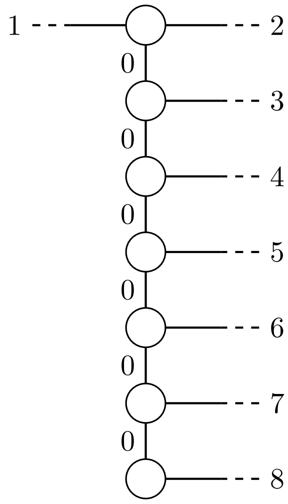
Tag：构造

J.1　可行解法

我们只需要将给定树 T 中度数超过 3 的点拆成边权为 0 的一条链，将连向原点的边接到链上的新点，并保证链上的每个点度数不超过 3 即可。

具体来说，我们可以将如下左图形态的点转为右图形态的链。原树上点与点之间的关系是独立的，对于每个点直接拆成链即可。





实现时不需要建出或遍历树，只需要记录每个点的度数以及当前拆成的链的信息即可。单组数据时间复杂度 $\mathcal{O}(n)$。

J.2　题外话

本题可能是本场比赛最有趣的题之一。

本题是 Special Judge，盲猜 checker 写起来可能都比解题代码麻烦一些些。

本题和 H 题交换位置可能比较合适。