Atcoder abc 386

E - Maximize XOR

这个又吃了一个教训，怎么说呢，我是不是因为没关注组合数学那个简单公式，就不会意识到这个教训，从而自然继续吃这个教训了。

总结是：不熟练数学导致的反应过慢以及无法继续推进。

这题是说，给你一个非负整数的数组 A ，长度为 N ，然后给你一个数字 K ，你需要在数组中选 K 个数（也就是选一个大小为 K 的子集），使得这 K 个数的异或和是最大的，输出最大的异或和。而且重点信息：题目告诉你，虽然 N 和 K 又数据范围，但是数据集中给的 N 和 K 的大小保证 $\binom{N}{K} \leq 10^6$ ，也就是说组合数不会超过 10^6 。

那么我们可以枚举所有的组合嘛，然后去计算异或和，更新答案。这样涉及到遍历 K 个你选择的数，K 如果太大了就会超时。

怎么办呢。

首先就是组合数学书上可能第一个公式或者常识我就给忘了：选 K 个的组合数等于选 $N-K$ 个的组合数，也就是 $\binom{N}{K} = \binom{N}{N-k}$ 。

可以看出一个大一个小，那么我们枚举所有排列是固定的 $O(\binom{N}{K}) = O(\binom{N}{N-K})$ ，在对一个排列计算异或和的时候的复杂度是 $O(K)$ 或者 $O(N-K)$ ，那么我们选择更小的那个去做不就好了。

也就是说

- 如果 K 比较小，我们就直接去求选 K 个答案。
- 如果 K 比较大，我们就去求排除 $N-K$ 个的答案，每次对于一个要排除的组合，假设异或和为 x ，我们要求的是排除掉它之后的异或和，那么答案为 $max(total\_sum \oplus x)$ 。
- 时间复杂度度就变成了 $O(\binom{N}{K} * min(K, N-K))$ 。

是否担心这个 $min(K, N-K)$会超时?

其实就可以大概算一下 K 是多少。比如 $\binom{2*10^5}{K} \leq 10^6$ 的话，K 其实只能选择 1 或者 $2*10^5$ ，那么 N 为其他数呢？可以想办法去数学证明，也可以在草稿纸上或者计算器自己随便试一些数量级的数，你会发现 $min(K, N-K)$ 会很小很小，就是一个常数级别。代码实现可以迭代的去枚举组合，我用的搜索方式，迭代的时间复杂度会更加优秀。搜索过程中加了一点点剪枝。

```
#include<bits/stdc++.h>
using namespace std;

long long res = 0, total = 0;;
int n, k;
bool ok = true;
void dfs(vector<long long>& a, int start, int len, long long sum) {
    if(ok) {
        if(len == k) {
            res = max(res, sum);
            return ;
        }
```

```
    } else  {
        if(len == n - k) {
            res = max(res, total ^ sum);
            return ;
        }
    }

    if(ok && len + n - start < k) return ;
    if(!ok && len + n - start < n - k) return ;
    for(int i = start; i < n; i ++) {
        long long new_sum = sum ^ a[i];
        dfs(a, i + 1, len + 1, new_sum);
    }
}

int main() {
    cin >> n >> k;
    vector<long long> a(n);
    for(int i = 0; i < n; i ++) {
        cin >> a[i];
        total ^= a[i];
    }
    if(k > n / 2) ok = false;
    dfs(a, 0, 0, 0);
    cout << res << endl;
    return 0;
}
```

F - Operate K

总感觉这两把 abc 的 dp 都是我应该能想出来的 dp，但是俺 dp 真的菜的离谱啊。。。。。哎，继续积累吧。

这题在前面一题上把 K 的范围从 $K = 1$ 改成了 $1 \leq K \leq 20$，其他不变。

就是一个经典题目来的。。。$leetcode$ 编辑距离？给你字符串 S 和字符串 T，有插入一个字母、删除一个字母、替换一个字母三种操作，问给你 S 和 T 还有 K，问你在不超过 K 次操作的情况下能不能把 S 变成 T。

朴素做法就是 $O(n * m)$，其中 $n = len(S)$， $m = len(T)$。

- 补充朴素做法：$f[i][j]$ 表示将 S 的 $1 \ldots i$ 和 T 的 $1 \ldots j$ 变成一样的操作次数最少是 $f[i][j]$。
- 那么对于添加操作：$f[i][j] = min(f[i][j], f[i][j-1] + 1)$，也就是说在 S 的 i 位置我要往后添加一个使得和 T 的 $1 \ldots j$ 一样，那么 S 的 $1 \ldots i$ 和 T 的 $1 \ldots j-1$ 已经经过 $f[i][j-1]$ 次操作匹配了。
- 同理对于删除操作：$f[i][j] = min(f[i][j], f[i-1][j])$。
- 对于修改操作：$f[i][j] = min(f[i][j], f[i-1][j-1] + (S[i]! = T[j]))$。

时间复杂度太大，那咋整了，注意到 K 这么小，肯定是从 K 开始考虑啊，我咋不往这里考虑呢？这个也吃了好多次教训了吧。

$1 \leq K \leq 20$，意思是你操作的次数不会超过 20，那么也就是说其实我只需要考虑 $|i-j| <= 20$ 部分的 dp 就好了！因为你要么修改，涉及的长度不变，要么一直添加或者一直删除，对于 S 的位置 i 来说，它需要考虑的 T 的 j 的范围只需要满足 $|i-j| <= 20$ 。时间复杂度就变了 $O(N*41)$ 。

然后写递推或者写记忆化搜索就行了，注意小技巧用偏移量控制不要爆负数的坐标。我喜欢记忆化搜索。

```
#include<bits/stdc++.h>
using namespace std;
int k;
string s, t;
int cache[500010][50];

int f(int i, int j) {
    if(i == 0) return j;
    if(j == 0) return i;
    if(abs(i - j) > k) return 0x3f3f3f3f;
    if(cache[i][i - j + 25] != -1) return cache[i][i - j + 25];

    int res = 0x3f3f3f3f;
    res = min(res, f(i - 1, j) + 1);
    res = min(res, f(i, j - 1) + 1);
    res = min(res, f(i - 1, j - 1) + (s[i] != t[j]));
    cache[i][i - j + 25] = res;
    return res;
}

void solve() {
    memset(cache, -1, sizeof cache);
    int n = s.size(), m = t.size();
    s = " " + s;
    t = " " + t;
    if(f(n, m) <= k) cout << "Yes" << endl;
    else cout << "No" << endl;
    return ;
}

int main() {
    cin >> k >> s >> t;
    solve();
    return 0;
}
```

G 哥们先放放 过个元旦再说