目录

Codeforces Round 1135 (Div. 2) 1
A.Composite Permutation 1
B.Diverse String 2
C.Max Substitution 4
D.Reverse Partition 6
E.Arithmetic Subarray 10

Codeforces Round 1135 (Div. 2)

A.Composite Permutation

题目描述

给定一个整数 n。请你构造一个长度为 n 的排列 p，使得对每个 $1 \le i \le n$，$i \cdot p_i$ 都是合数；如果不存在这样的排列，请判断出来。

其中「排列」指长度为 n 的数组，它由 1 到 n 的 n 个互不相同的整数按任意顺序组成。例如 $[2,3,1,5,4]$ 是排列，而 $[1,2,2]$ 不是排列（2 在数组中出现两次），$[1,3,4]$ 也不是排列（$n=3$，但数组中出现 4）。

其中「合数」指除 1 和它本身之外还有其它约数的正整数。例如 6 是合数，因为 2 是它的约数；而 1 和 5 不是合数。

约束条件

- $1 \le t \le 100$
- $1 \le n \le 100$

输入格式

输入包含多组测试数据。第一行包含测试数据组数 t，接下来是各组测试数据的描述。

每组测试数据仅有一行，包含一个整数 n。

输出格式

对于每组测试数据，如果不存在合适的排列，输出一个整数 -1。

否则输出 n 个整数 $p_1, p_2, ..., p_n$，表示一个合适的排列。

如果存在多种方案，输出任意一种即可。

样例

1.in / 1.out

```
1 3
2 1
3 4
4 5
```

```
1 -1
2 4 3 2 1
3 4 5 3 1 2
```

在第一组测试数据中，可以证明不存在合适的排列。

在第二组测试数据中，$i \cdot p_i$ 的值依次为 4、6、6、4，它们都是合数。

题解

发现只有 1 乘质数和质数乘 1 两种情况产生质数。

那很简单啊，把 1 和 4 的位置换一下就好了，看一下 1-3 的情况似乎怎么改都不满足，所以直接这样构造。

```cpp
#include <stdio.h>
int T, n;
int main() {
    scanf("%d", &T);
    while (T--) {
        scanf("%d", &n);
        if (n < 4) {
            puts("-1");
            continue;
        }
        for (int i = 1; i <= n; i++) {
            if (i == 1)
                printf("4 ");
            else if (i == 4)
                printf("1 ");
            else
                printf("%d ", i);
        }
        puts("");
    }
    return 0;
}
```

B.Diverse String

题目描述

一个 01 串 s 的「多样性」定义为：至多进行一次如下操作所能得到的不同的字符串个数：

选择整数 $0 \le x \le 1$ 以及两个整数 $1 \le l \le r \le n$，对所有 $l \le i \le r$ 令 $s_i := x$。

例如，字符串 10 的多样性为 3，因为可以得到 10、11 和 00 这三个字符串。

给定一个长度为 n 的字符串 s 以及一个整数 k。你最多可以翻转（0 变 1、1 变 0）字符串中 k 个字符的值。你的任务是求出翻转后字符串可能的最大多样性。

两个字符串 a 和 b 不同，当且仅当存在下标 i 使得 $a_i \neq b_i$。

约束条件

- $1 \le t \le 10^4$
- $1 \le n \le 2 \cdot 10^5$，$0 \le k \le n$
- 所有测试数据的 n 之和不超过 $3 \cdot 10^5$

输入格式

输入包含多组测试数据。第一行包含测试数据组数 t，接下来是各组测试数据的描述。

每组测试数据的第一行包含两个整数 n 和 k，分别表示字符串的长度和最多可以翻转的字符数。

每组测试数据的第二行包含一个长度为 n 的 01 串 s。

输出格式

对于每组测试数据，输出一行一个整数，表示翻转至多 k 个字符后字符串可能的最大多样性。

样例

1.in / 1.out

```
4
1 1
0
4 4
1001
5 1
11111
7 2
1101001
```

```
2
11
16
23
```

在第一组测试数据中，最优方案是翻转第一个字符，得到字符串 1，其多样性为 2。

在第二组测试数据中，字符串 1111 可以达到最大多样性 11。

题解

感觉自己的思维能力的确有所提升。

思考怎么构造出不同的字符串。因为我只能赋值一段区间为一个值，所以如果给定左端点为 l，则我赋值要赋不一样的值，不然就没变化了。

所以，假设某一位的值为 1，这一位贡献的答案就是这一位包括后面所有位中 1 的个数（0 也是同理）.

记 0 的数量为 c_0，1 的数量为 c_1，总体答案就是

$$\frac{c_0(c_0+1)}{2}+\frac{c_1(c_1+1)}{2}$$

而 $c_0+c_1=n$，所以这就是一个二次函数，我们去求极值的时候发现对称轴是负的。

这意味着，我们要尽可能拉大 c_0 和 c_1 的差距。

```cpp
#include <cstdio>
#include <stdio.h>
#include <utility>
#define maxn 200007

int T;
int c0 = 0, c1 = 0, n, k;

void rd() {
    char c = 'a';
    do {
```

```
        c = getchar();
    } while (c != '0' and c != '1');
    if (c == '0')
        c0++;
    else
        c1++;
}

typedef long long i64;

i64 mul(i64 x) { return x * (x + 1) / 2; }
i64 cnt(i64 x) { return mul(x) + mul(n - x); }

int main() {
    scanf("%d", &T);
    while (T--) {
        scanf("%d%d", &n, &k);
        c0 = 0, c1 = 0;

        for (int i = 1; i <= n; i++)
            rd();

        if (c0 < c1)
            std::swap(c0, c1);

        if (c1 <= k) {
            printf("%lld\n", cnt(n) + 1);
        } else {
            printf("%lld\n", cnt(c1 - k) + 1);
        }
    }
    return 0;
}
```

C.Max Substitution

题目描述

给定一个长度为 n 的数组 a。一次操作中，你可以把 a 中唯一的最大值替换为任意一个严格更小的正整数（即任意满足 $1 \le x < \max(a)$ 的整数 x）。只有当 a 的最大值唯一时，才能执行该操作。

求出执行任意次操作（可以为 0 次）后能够得到的不同数组的个数。答案可能很大，请对 $10^9 + 7$ 取模。

两个数组 a 和 b 不同，当且仅当存在下标 i 使得 $a_i \neq b_i$。

约束条件

- $1 \le t \le 1000$
- $1 \le n \le 2 \cdot 10^5$
- $1 \le a_i \le 10^9$
- 所有测试数据的 n 之和不超过 $2 \cdot 10^5$

输入格式

输入包含多组测试数据。第一行包含测试数据组数 t，接下来是各组测试数据的描述。

每组测试数据的第一行包含一个整数 n，表示数组的长度。

每组测试数据的第二行包含 n 个整数 $a_1, a_2, ..., a_n$，表示数组的元素。

输出格式

对于每组测试数据，输出一行一个整数，表示执行任意次操作后能够得到的不同数组的个数，对 $10^9 + 7$ 取模。

样例

1.in / 1.out

```
5
3
2 2 1
2
1000000000 1
4
6 7 2 2
3
3 4 2
1
1
```

```
1
1000000000
32
9
1
```

在第一组测试数据中，最大元素 2 不唯一，因此只能得到初始数组 $[2, 2, 1]$ 这一个数组。

在第二组测试数据中，最大元素为 1000000000。可以证明，执行任意次操作后，可能的数组为 $[1, 1], [2, 1], [3, 1], ..., [1000000000, 1]$，因此答案为 1000000000。

题解

看了一点题解提示。

如果我们将数字从大到小排序，则一个数如果能被操作，意味着它前面的数都变成了比它更小的数。

具体变小之后有没有重复呢？我们并不在意，因为只要变小了我们就可以操作，而如果前面的数都变得比现在的数小了，自然也比前面其他的数小，前面的其他数也可以操作下去。

所以，每次计数就是 $c_i^{v_i}$，其中 c_i 是前面的数的个数 $+1$（包括现在这个），v_i 是现在这个数的值。

由于可以不对数进行修改，所以答案最终要 $+1$.

```cpp
#include <map>
#include <stdio.h>

const int mod = 1000000007;
int pow(int x, int a) {
    if (a == 0)
```

```
        return 1;
    int ans = pow(x, a >> 1);
    if (a & 1)
        return 1ll * ans * ans % mod * x % mod;
    return 1ll * ans * ans % mod;
}

int T, n, u, v, ans = 0;
std::map<int, int> m;

int main() {
    scanf("%d", &T);
    while (T--) {
        m.clear();
        ans = 1;

        scanf("%d", &n);
        for (int i = 1; i <= n; i++)
            scanf("%d", &u), m[u]++;

        int bef = 0;
        for (auto it = m.rbegin(); it != m.rend(); it++) {
            auto [u, v] = *it;
            if (v >= 2) {
                // printf("dup, counted 1\n");
                break;
            }

            // printf("counted %d * %d\n", pow(u - 1, bef + 1), 1);
            ans = (ans + 1ll * pow(u - 1, bef + 1) % mod) % mod;
            bef++;
        }

        printf("%d\n", ans);
    }
    return 0;
}
```

D.Reverse Partition

题目描述

给定两个正整数 n 和 m。

一个「划分」是一个由 m 个正整数组成的数组 p，满足 $p_1 + p_2 + ... + p_m = n$。

对于任意长度为 n 的数组 b 和一个划分 p，定义 $f(b, p)$ 为一个长度为 m 的数组 c，其中：

- c_1 是 b 的前 p_1 个元素之和；
- c_2 是 b 接下来的 p_2 个元素之和；
- ……

- c_m 是 b 最后 p_m 个元素之和。

对于一个长度为 n 的数组 d，设 e 为将 d 反转后得到的数组。如果 $f(d,p)=f(e,p)$，则称划分 p 是 d 的一个「特殊划分」。

给定一个长度为 n 的数组 a，你需要回答 q 个相互独立的询问。每个询问给定两个整数 i 和 v，先令 $a_i := a_i + v$，然后输出 a 的特殊划分个数。答案可能很大，请对 10^9+7 取模。

注意询问相互独立，即每个询问结束后 a 会恢复为原始值。

约束条件

- $1 \le t \le 10^4$
- $1 \le m \le n \le 2 \cdot 10^5$，$1 \le q \le 2 \cdot 10^5$
- $|a_i| \le 10^9$
- 每个询问中 $1 \le i \le n$，$|v| \le 10^9$
- 所有测试数据的 n 之和与 q 之和均不超过 $2 \cdot 10^5$

输入格式

输入包含多组测试数据。第一行包含测试数据组数 t，接下来是各组测试数据的描述。

每组测试数据的第一行包含三个整数 n、m 和 q，分别表示 a 的长度、划分的长度和询问的个数。

第二行包含 n 个整数 $a_1, a_2, ..., a_n$。

接下来 q 行，每行包含两个整数 i 和 v。

输出格式

对于每个询问，输出一个整数，表示该询问下 a 的特殊划分个数，对 10^9+7 取模。

样例

1.in / 1.out

```
2
5 2 3
1 2 3 2 1
3 0
2 1
5 -1
8 3 6
3 1 4 5 1 7 2 3
1 0
4 2
8 -2
2 1
5 -1
7 3
```

```
4
2
0
3
1
0
6
```

```
8 1
9 1
```

在第一组测试数据中有 3 个询问：

- 第一个询问中，数组为 $a = [1, 2, 3, 2, 1]$，特殊划分为 $[1, 4]$、$[2, 3]$、$[3, 2]$ 和 $[4, 1]$。
- 第二个询问中，数组为 $a = [1, 3, 3, 2, 1]$，特殊划分为 $[1, 4]$ 和 $[4, 1]$。
- 第三个询问中，数组为 $a = [1, 2, 3, 2, 0]$，可以证明不存在特殊划分。

题解

这个题是灵机一动的典型吧。

因为我们的划分（partition）位置是固定的，而序列被反转了，所以我们首先考虑怎么知道前几个数后几个数的和相等呢？

前缀和可以优化，然后从前到后扫一下，对吧？

我们还可以更进一步，这道题最技巧性的一步——**我们令正序前缀和减去倒序前缀和。**

这样的话，我们的可以划分的点就是 $[0, n-1]$ 之中值为 0 的点，记有 k 个 0，最终的答案就是 C_k^{m-1}.

但是题目还有 q 次独立的修改，我们如果每次都前缀和然后去统计 0，会相当麻烦。

因为修改是独立的，而且只改一个值。我们发现，这就是在前缀和数组上进行了一个区间加 k，所以我们只需要在这一段区间上把原来的统计 0 变成统计 $-k$ 即可。

怎么实现呢？离线询问，拆分端点！

离线询问后，我们可以用 `map` 维护出给定左端点到 n 中任意数字的个数，用类似前缀和的方法就能得到区间内的答案。

这里有一个实现细节，如果你的左右端点反转了，意味着我们加减的值也反转了，此时我们需要统计 k（下面代码的判断）.

```cpp
for (int i = 1; i <= q; i++) {
    scanf("%d%d", &u, &v);
    if (u == n - u + 1)
        continue;

    int l = u;
    int r = n - u + 1;
    if (l < r) {
        edge[l].push_back({-v, i, 1});
        edge[r].push_back({-v, i, -1});
    } else {
        edge[r].push_back({v, i, 1});
        edge[l].push_back({v, i, -1});
    }
}
```

所以我们就得到了本题的 AC 代码。

```cpp
#include <array>
#include <map>
#include <stdio.h>
```

```cpp
#include <vector>
#define maxn 200007

const int mod = 1000000007;

int T, n, k, v, m, q, c0, u;
int li[maxn];
long long s[maxn], p[maxn];
int invs[maxn], fac[maxn], facinv[maxn], ans[maxn];

inline int C(int n, int m) {
    if (n < m)
        return 0;
    return 1ll * fac[n] * facinv[m] % mod * facinv[n - m] % mod;
}

std::vector<std::array<int, 3>> edge[maxn];
std::map<long long, int> mp; // FIX1: key 与 p[] 同类型

int main() {
    invs[1] = 1;
    fac[0] = 1;
    fac[1] = 1;
    facinv[0] = 1;
    facinv[1] = 1;
    for (int i = 2; i < maxn; i++)
        invs[i] = 1ll * (mod - mod / i) * invs[mod % i] % mod,
        fac[i] = 1ll * i * fac[i - 1] % mod,
        facinv[i] = 1ll * invs[i] * facinv[i - 1] % mod;

    scanf("%d", &T);
    while (T--) {
        scanf("%d%d%d", &n, &m, &q);

        mp.clear();
        for (int i = 1; i <= n; i++)
            edge[i].clear();

        for (int i = 1; i <= n; i++)
            scanf("%d", li + i);

        for (int i = 1; i <= n; i++)
            s[i] = (long long)li[i] - li[n - i + 1];

        for (int i = 1; i <= n; i++) {
            p[i] = s[i] + p[i - 1];
            mp[p[i]]++;
        }
```

```
52
53          c0 = 0;
54          for (int i = 1; i < n; i++)
55              c0 += (p[i] == 0);
56
57          for (int i = 1; i <= q; i++)
58              ans[i] = c0;
59
60          for (int i = 1; i <= q; i++) {
61              scanf("%d%d", &u, &v);
62              if (u == n - u + 1)
63                  continue;
64
65              int l = u;
66              int r = n - u + 1;
67              if (l < r) {
68                  edge[l].push_back({-v, i, 1});
69                  edge[r].push_back({-v, i, -1});
70              } else {
71                  edge[r].push_back({v, i, 1});
72                  edge[l].push_back({v, i, -1});
73              }
74          }
75
76          for (int i = 1; i <= n; i++) {
77              for (auto [val, id, f] : edge[i]) {
78                  ans[id] += mp[val] * f;
79                  ans[id] -= mp[0] * f;
80              }
81
82              mp[p[i]]--;
83          }
84
85          for (int i = 1; i <= q; i++)
86              printf("%d\n", C(ans[i], m - 1));
87      }
88      return 0;
89  }
```

E.Arithmetic Subarray

题目描述

给定一个长度为 n 的数组 a 和一个正整数 k。

如果子数组 $a[l..r]$ 中相邻元素的差是常数，即 $a_{l+1} - a_l = a_{l+2} - a_{l+1} = ... = a_r - a_{r-1}$，则称 $a[l..r]$ 是一个「等差子数组」。注意长度为 1 的子数组都是等差子数组。

你最多可以选择 a 中的 k 个元素，把它们改成任意整数（不要求在 $[1, 10^9]$ 内）。你的任务是求出修改后 a 的等差子数组的最大可能长度。

数组 a 的一个子数组指 $[a_l, a_{l+1}, ..., a_r]$，其中 $1 \le l \le r \le n$。

约束条件

- $2 \le n \le 2 \cdot 10^5$
- $1 \le k \le \min(n, 1500)$
- $1 \le a_i \le 10^9$
- 所有测试数据的 n 之和不超过 $2 \cdot 10^5$
- 所有测试数据的 $n \cdot k$ 之和不超过 10^7

输入格式

输入包含多组测试数据。第一行包含测试数据组数 t，接下来是各组测试数据的描述。

每组测试数据的第一行包含两个整数 n 和 k。

每组测试数据的第二行包含 n 个整数 $a_1, a_2, ..., a_n$。

输出格式

对于每组测试数据，输出一个整数，表示修改至多 k 个元素后 a 的等差子数组的最大可能长度。

样例

1.in / 1.out

```
4
5 1
1 2 6 4 5
5 1
1 2 3 100 200
3 1
1 1000000000 2
6 2
1 2 4 8 16 32
```

```
5
4
3
4
```

在第一组测试数据中，把数组改为 $a = [1, 2, 3, 4, 5]$ 即可得到答案 5。

在第三组测试数据中，把数组改为 $a = [1999999998, 1000000000, 2]$ 即可得到答案 3。

题解

TODO