目录

abc483 1
D.Glitchy Robot 1
E.Typo Pairs 5
F.Sum of Mod of Linear 9

abc483

D.Glitchy Robot

题目描述

数轴上有一个机器人。初始时，机器人在坐标 0 处。 另外，高桥有一个长度为 N 的整数序列 $A = (A_1, A_2, ..., A_N)$。

高桥按以下顺序执行操作：

- 选择 A 中至多 1 个元素并翻转其符号。也就是说，选择 0 个或 1 个满足 $1 \leq i \leq N$ 的下标 i，将 A_i 替换为 $-A_i$。
- 启动机器人。启动后，机器人会进行 N 次移动。在第 i 次移动（$1 \leq i \leq N$）中，设当前位置为 x，机器人沿直线移动到坐标 $x + A_i$。

高桥希望最大化机器人至少经过一次的整数坐标的个数。 求高桥采取最优行动时，「机器人至少经过一次的整数坐标的个数」的最大可能值。

这里，机器人是连续移动的。当一次移动使机器人从坐标 S 移动到坐标 T 时，认为它经过了 S 与 T 之间（含两端）的所有整数坐标。

约束条件

- $1 \leq N \leq 2 \times 10^5$
- $-10^9 \leq A_i \leq 10^9$
- $A_i \neq 0$
- 所有输入均为整数。

输入格式

输入由标准输入给出，格式如下：

```
1 N
2 A_1 A_2 ... A_N
```

输出格式

输出高桥采取最优行动时，机器人至少经过一次的整数坐标的个数的最大可能值。

样例

1.in / 1.out

```
1 3
2 -2 3 -2
```

```
1 8
```

初始时 $A = (-2, 3, -2)$。若高桥翻转 A_2 的符号，则 $A = (-2, -3, -2)$。 此时机器人的移动如下：

- 第 1 次移动：机器人从 0 移动到 -2。本次移动经过的整数坐标为 $-2, -1, 0$。
- 第 2 次移动：机器人从 -2 移动到 -5。本次移动经过的整数坐标为 $-5, -4, -3, -2$。

- 第 3 次移动：机器人从 -5 移动到 -7。本次移动经过的整数坐标为 $-7, -6, -5$。

因此，机器人至少经过一次的整数坐标为 $-7, -6, -5, -4, -3, -2, -1, 0$ 共 8 个，这也是可能达到的最大值，所以输出 8。

注意，若高桥不翻转任何元素的符号，机器人将按 $0 \to -2 \to 1 \to -1$ 移动，此时机器人至少经过一次的整数坐标为 $-2, -1, 0, 1$ 共 4 个。

2.in / 2.out

```
1 2
2 2 3
```

```
1 6
```

若不改变 A 中的任何元素，机器人按 $0 \to 2 \to 5$ 移动，机器人至少经过一次的整数坐标为 $0, 1, 2, 3, 4, 5$ 共 6 个。 这也是可能达到的最大值，所以输出 6。

题解

这个题目，我看到的第一眼是这样想的：

第 i 次到达的点，就是前缀和序列上第 i 个值，记作 s_i

能够到达的最大点和最小点，就是 s 数组的最大和最小值。

我们知道线段树可以维护区间最大值和最小值查询，同时进行区间加，所以我们就直接写一个线段树……

过了，但真的有必要吗？

cpp

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

#define maxn 2000007
typedef long long i64;

struct node {
    i64 mx, mn, lazy;
    node(i64 mx = 0, i64 mn = 0, i64 lazy = 0) : mx(mx), mn(mn), lazy(lazy) {}
} tree[maxn];

#define call int mid = (l + r) >> 1, ls = index << 1, rs = ls | 1;

inline i64 min(i64 a, i64 b) { return a < b ? a : b; }
inline i64 max(i64 a, i64 b) { return a > b ? a : b; }

void upd(int index, int l, int r) {
    call;
    tree[index].mx = max(tree[ls].mx, tree[rs].mx);
    tree[index].mn = min(tree[ls].mn, tree[rs].mn);
}

i64 li[maxn], s[maxn];
void init(int index, int l, int r) {
```

```
25      if (l == r) {
26          if (l != 1)
27              scanf("%lld", li + l);
28          s[l] = li[l] + s[l - 1];
29          tree[index].mx = tree[index].mn = s[l];
30          return;
31      }
32      call;
33      init(ls, l, mid), init(rs, mid + 1, r);
34      upd(index, l, r);
35  }
36
37  void pushlazy(int index, int l, int r) {
38      call;
39      i64 v = tree[index].lazy;
40      tree[index].lazy = 0;
41
42      tree[ls].lazy += v;
43      tree[ls].mn += v;
44      tree[ls].mx += v;
45
46      tree[rs].lazy += v;
47      tree[rs].mn += v;
48      tree[rs].mx += v;
49  }
50
51  void add(int index, int l, int r, int ql, int qr, i64 v) {
52      pushlazy(index, l, r);
53      if (ql <= l and r <= qr) {
54          tree[index].lazy += v;
55          tree[index].mn += v;
56          tree[index].mx += v;
57          return;
58      }
59      if (l > qr or r < ql)
60          return;
61      call;
62
63      add(ls, l, mid, ql, qr, v);
64      add(rs, mid + 1, r, ql, qr, v);
65      upd(index, l, r);
66  }
67
68  typedef std::pair<i64, i64> pi6;
69
70  pi6 qminmx(int index, int l, int r, int ql, int qr) {
71      pushlazy(index, l, r);
72      if (ql <= l and r <= qr)
```

```
73          return {tree[index].mn, tree[index].mx};
74      if (l > qr or r < ql)
75          return {1ll << 62, -(1ll << 62)};
76      call;
77
78      auto lv = qminmx(ls, l, mid, ql, qr), rv = qminmx(rs, mid + 1, r, ql, qr);
79      return {min(lv.first, rv.first), max(lv.second, rv.second)};
80  }
81
82  int n;
83  i64 ans;
84  int main() {
85      scanf("%d", &n);
86      n++;
87      init(1, 1, n);
88
89      auto r = qminmx(1, 1, n, 1, n);
90      ans = r.second - r.first;
91
92      for (int i = 2; i <= n; i++) {
93
94          add(1, 1, n, i, n, -2 * li[i]);
95
96          auto r = qminmx(1, 1, n, 1, n);
97          ans = max(ans, r.second - r.first);
98
99          add(1, 1, n, i, n, 2 * li[i]);
100     }
101
102     printf("%lld\n", ans + 1);
103
104     return 0;
105 }
```

显然看到这种题你拍一个雷霆线段树上去是很抽象的，我们来思考一下我们被期望怎么做。

正解也用了类似于线段树的合并性质：区间整体加减的时候，其 $\min, \max$ 也会同时加减。这条性质相当显然，不用证明。

思考我们进行虚拟意义的 $[k, n]$ 区间加 v 时，就是 $\max_{j=k}^{n}, \min_{j=k}^{n}$ 加了 v，那我们要求的 $\max_{k=1}^{n} = \max\left(\max_{j=1}^{k-1} s_j, \max(j=k)^n s_j + v\right)$，所求的 $\min$ 同理。

所以我们只需要维护前缀最大最小和后缀最大最小就可以通过此题。

本题不用线段树的原因是我们进行操作的区间非常固定，而且操作完会还原，同时只有一个操作，线段树的作用就不大了，可以用归纳讨论的方式来简化掉。

```cpp
#include <stdio.h>

typedef long long i64;
inline i64 max(i64 a, i64 b) { return a > b ? a : b; }
```

```cpp
inline i64 min(i64 a, i64 b) { return a < b ? a : b; }

#define maxn 200007

i64 li[maxn], s[maxn], lmn[maxn], lmx[maxn], rmn[maxn], rmx[maxn];
int n;

int main() {
    scanf("%d", &n);
    for (int i = 1; i <= n; i++)
        scanf("%lld", li + i), s[i] = li[i] + s[i - 1];

    lmn[0] = lmx[0] = 0;
    rmn[n + 1] = 1ll << 60;
    rmx[n + 1] = -(1ll << 60);

    for (int i = 1; i <= n; i++)
        lmn[i] = min(s[i], lmn[i - 1]), lmx[i] = max(s[i], lmx[i - 1]);
    for (int i = n; i >= 1; i--)
        rmn[i] = min(s[i], rmn[i + 1]), rmx[i] = max(s[i], rmx[i + 1]);

    i64 ans = (lmx[n] - lmn[n] + 1);
    for (int i = 1; i <= n; i++) {
        ans = max(ans, max(lmx[i - 1], rmx[i] - 2 * li[i]) -
                           min(lmn[i - 1], rmn[i] - 2 * li[i]) + 1);
    }

    printf("%lld", ans);

    return 0;
}
```

E.Typo Pairs

题目描述

有 N 个由小写英文字母组成、长度为 M 的字符串。 第 i 个字符串（$1 \leq i \leq N$）为 S_i。

求满足以下条件的整数对 (i, j)（$1 \leq i < j \leq N$）的个数：

- S_i 与 S_j 的汉明距离不超过 2。

什么是汉明距离？

两个长度均为 k 的字符串 A、B 的汉明距离，是指满足 $1 \leq i \leq k$、且 A 的第 i 个字符与 B 的第 i 个字符不同的整数 i 的个数。 例如，abc 与 bac 的汉明距离为 2，atcoder 与 contest 的汉明距离为 7。

约束条件

- $2 \leq N \leq 2 \times 10^5$
- $2 \leq M \leq 6$
- N、M 均为整数。

- S_i 是长度为 M、由小写英文字母组成的字符串。

输入格式

输入由标准输入给出，格式如下：

```
N M
S_1
S_2
...
S_N
```

输出格式

输出满足条件的整数对 (i, j)（$1 \leq i < j \leq N$）的个数。

样例

1.in / 1.out

```
3 3
abc
abd
bac
```

```
2
```

- $S_1 =$ `abc` 与 $S_2 =$ `abd` 的汉明距离为 1。
- $S_1 =$ `abc` 与 $S_3 =$ `bac` 的汉明距离为 2。
- $S_2 =$ `abd` 与 $S_3 =$ `bac` 的汉明距离为 3。

因此，满足条件的整数对为 $(i, j) = (1, 2)$、$(1, 3)$ 共 2 个，所以输出 2。

2.in / 2.out

```
9 5
aaaaa
aaaaa
aaaab
aaaba
aaabb
aabaa
aabab
aabba
aabbb
```

```
31
```

同一个字符串可能出现多次。

题解

不知道官方题解在写什么。我想说的是这道题就是容斥，但想到容斥是不容易的。

我们思考一下出题人期望我们干什么。

因为 M 比较小，所以其实我们可以枚举字串中不同的部分。

具体地，我们记 f_i 为和 s_i 相同的字符串个数（不含），记 $g_{i,e}$ 为 s_i 去掉 e 位置上的字符之后相同的字符串个数（不含），记 h_{i,e_1,e_2} 为 s_i 去掉 e_1, e_2 位置上的字符之后相同的字符串个数（不含）。

这三个的维护方法是显然的，但请注意 $\sum h_{i,\dots}$ 不是答案！

因为如果 s_i, s_j 本来就相同，也会被计算到 h_i 中，具体地，由于怎么剔除都满足，会被重复计算 $\binom{m}{2}$ 次。

如果 s_i, s_j 有一位不同，也会被计算到 h_i 中，具体地，当 (e_1, e_2) 中的其中一个是不同的位的时候满足条件，会被重复计算 $m-1$ 次。

所以答案分为三部分：汉明距离为 0 的 $A = \sum_{i=1}^{n} f_i$，汉明距离为 1 的 $\sum B_i = \left[\sum_{j=0}^{m-1} g_{i,j} - (m-1) \cdot f_i\right]$，汉明距离为 2 的 ... 放过我，你看到这里自己思考一下吧。

代码可以精简，但是会丢失语义，所以没有改动。

```cpp
#include <iostream>
#include <stdio.h>
#include <string>
#include <unordered_map>

#define maxn 200007

using namespace std;
unordered_map<string, int> dif0, dif1, dif2;
string ss[maxn];
int f[maxn][6], g[maxn][6][6], e0[maxn], a1[maxn];
int n, m;

inline string genex1(const string &s, int ex) {
    string x;
    for (int j = 0; j < m; j++)
        if (j != ex)
            x.push_back(s[j]);
    return x;
}

inline string genex2(const string &s, int ex1, int ex2) {
    string x;
    for (int j = 0; j < m; j++)
        if (j != ex1 and j != ex2)
            x.push_back(s[j]);
    return x;
}

typedef long long i64;
i64 ans = 0;

int main() {
    scanf("%d%d", &n, &m);
    for (int i = 1; i <= n; i++)
```

```
36          cin >> ss[i], dif0[ss[i]]++;
37
38      for (int ex = 0; ex < m; ex++) {
39          dif1.clear();
40          for (int i = 1; i <= n; i++)
41              dif1[genex1(ss[i], ex)]++;
42
43          for (int i = 1; i <= n; i++) {
44              // exclude self
45              f[i][ex] = dif1[genex1(ss[i], ex)] - 1;
46          }
47      }
48
49      for (int ex1 = 0; ex1 < m; ex1++)
50          for (int ex2 = ex1 + 1; ex2 < m; ex2++) {
51              dif2.clear();
52              for (int i = 1; i <= n; i++)
53                  dif2[genex2(ss[i], ex1, ex2)]++;
54
55              for (int i = 1; i <= n; i++) {
56                  // exclude self
57                  g[i][ex1][ex2] = (dif2[genex2(ss[i], ex1, ex2)] - 1);
58              }
59          }
60
61      for (int i = 1; i <= n; i++)
62          e0[i] = dif0[ss[i]] - 1;
63
64      for (int i = 1; i <= n; i++) {
65          ans += e0[i];
66      }
67
68      for (int i = 1; i <= n; i++) {
69          i64 ta = 0;
70          for (int ex = 0; ex < m; ex++)
71              ta += f[i][ex];
72          ta -= m * e0[i];
73          a1[i] = ta;
74          ans += ta;
75      }
76
77      for (int i = 1; i <= n; i++) {
78          i64 ta = 0;
79          for (int ex1 = 0; ex1 < m; ex1++)
80              for (int ex2 = ex1 + 1; ex2 < m; ex2++)
81                  ta += g[i][ex1][ex2];
82
83          ta -= 1ll * m * (m - 1) / 2 * e0[i];
```

```
84          ta -= (m - 1) * a1[i];
85          ans += ta;
86      }
87
88      printf("%lld", ans / 2);
89
90      return 0;
91  }
```

F.Sum of Mod of Linear

题目描述

给定整数 N, Q，以及长度为 N 的整数序列 $A = (A_1, A_2, ..., A_N)$、$B = (B_1, B_2, ..., B_N)$。

给定 Q 个查询，请依次回答。在第 i 个查询中，给定一个整数 x，求 $\sum_{i=1}^{N}((x + A_i) \bmod B_i)$ 的值。

约束条件

- $1 \leq N \leq 5 \times 10^5$
- $0 \leq A_i < B_i \leq 5 \times 10^5$
- $1 \leq Q \leq 2 \times 10^5$
- $0 \leq x \leq 5 \times 10^5$
- 所有输入均为整数。

输入格式

输入由标准输入给出，格式如下：

```
N
A_1 B_1
A_2 B_2
...
A_N B_N
Q
query_1
query_2
...
query_Q
```

第 i 个查询的输入格式如下：

```
x
```

输出格式

输出 Q 行。

第 i 行（$1 \leq i \leq Q$）输出第 i 个查询的答案。

样例

1.in / 1.out

```
3
1 3
2 5
```

```
4   0 1
5   5
6   0
7   1
8   2
9   3
10  200000
```

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

每个查询的答案如下：

- 第 1 个查询中 $x = 0$，答案为 $((0 + 1) \bmod 3) + ((0 + 2) \bmod 5) + ((0 + 0) \bmod 1) = 1 + 2 + 0 = 3$。
- 第 2 个查询中 $x = 1$，答案为 $((1 + 1) \bmod 3) + ((1 + 2) \bmod 5) + ((1 + 0) \bmod 1) = 2 + 3 + 0 = 5$。
- 第 3 个查询中 $x = 2$，答案为 $((2 + 1) \bmod 3) + ((2 + 2) \bmod 5) + ((2 + 0) \bmod 1) = 0 + 4 + 0 = 4$。
- 第 4 个查询中 $x = 3$，答案为 $((3 + 1) \bmod 3) + ((3 + 2) \bmod 5) + ((3 + 0) \bmod 1) = 1 + 0 + 0 = 1$。
- 第 5 个查询中 $x = 200000$，答案为 $((200000 + 1) \bmod 3) + ((200000 + 2) \bmod 5) + ((200000 + 0) \bmod 1) = 0 + 2 + 0 = 2$。

2.in / 2.out

```
1 3
2 2 3
3 5 7
4 0 9
5 4
6 9
7 3
8 6
9 17
```

```
1 2
2 6
3 12
4 10
```

3.in / 3.out

```
1   8
2   0 10
3   3 8
4   1 3
```

```
5  5 7
6  4 8
7  4 9
8  2 6
9  5 7
10 7
11 130140
12 70510
13 22719
14 104735
15 119072
16 127841
17 191631
```

```
1 16
2 21
3 31
4 29
5 19
6 21
7 27
```

题解

哇这个题我竟然能看懂题解……？好像有点走马灯了。

这里必须把 mod 拆分开，不然的话每个模数都不同，肯定是做不了的，所以：

$$\begin{aligned}&\sum_{i=1}^{n}(x+A_i)\bmod b_i\\=&\sum_{i=1}^{n}\left[x+A_i-\left\lfloor\frac{x+A_i}{B_i}\right\rfloor\cdot B_i\right]\\=&\,n\cdot x+\sum_{i=1}^{n}A_i-\sum_{i=1}^{n}B_i\cdot\left\lfloor\frac{x+A_i}{B_i}\right\rfloor\end{aligned}$$

然后我们只用求最后一段就好了。

但是复杂度优化在哪呢？回到题目要求发现我们的 $A_i,B_i,x\le 10^5$，这意味着我们的值域处理是可以接受的。

看到这个整除运算，你就应该想到：数论分块！

经典的数论分块是让你算 $\sum_{i=1}^{n}\left\lfloor\frac{n}{i}\right\rfloor$ 这样的形式，而这里的话，发现 $f(x)=B_i\left\lfloor\frac{x+A_i}{B_i}\right\rfloor$ 这个函数是符合数论分块性质的！

具体地，给定一个数 k，我们发现在 $[k\cdot B_i-A_i,(k+1)\cdot B_i-A_i)$ 这个区间内，$f(x)=k$.

这启示我们：预处理 x 的值域，然后使用差分运算来进行区间加，最后统一前缀和。

但是到现在，我们的复杂度还有一个问题。如果我给出很多对 $(A_i,B_i)=(0,1)$ 这样 B_i 相当小的最坏情况，我们处理它需要 $\frac{\mathrm{MAXN}}{B_i}\sim\mathrm{MAXN}$ 的时间，复杂度直接飞起来。

然后就是本题比较关键的优化：把相同的 (A_i,B_i) 合并。

在这种情况下，我们的最坏情况是：$(A_i,B_i)=(0,1),(0,2),(1,2),(0,3),(1,3),(2,3),...$

记 $N = \mathrm{MAXN}$ 复杂度大概是 $\sum_{i=1}^{\sqrt{n}} i \cdot \left\lfloor \frac{N}{i} \right\rfloor \sim N\sqrt{N}$，计算量在 3×10^8，就过去了。

cpp

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

#define maxn 500107

typedef long long i64;
typedef std::pair<int, int> pii;

int n, u, v, q, x;

std::map<pii, int> mp;

i64 diff[maxn], s;

inline int max(int a, int b) { return a > b ? a : b; }

int main() {
    scanf("%d", &n);
    for (int i = 1; i <= n; i++)
        scanf("%d%d", &u, &v), mp[{u, v}]++, s += u;

    for (auto [A, r] : mp) {
        auto [a, b] = A;

        for (int v = 0;; v++) {
            int lb = max(0, v * b - a);
            int rb = (v + 1) * b - a - 1;

            if (rb < 0)
                continue;

            if (lb >= maxn)
                break;

            diff[lb] += 1ll * v * r * b;
            if (rb + 1 < maxn)
                diff[rb + 1] -= 1ll * v * r * b;
        }
    }

    for (int i = 1; i < maxn; i++)
        diff[i] += diff[i - 1];

    scanf("%d", &q);
    for (int i = 1; i <= q; i++) {
        scanf("%d", &x);
```

```
47          printf("%lld\n", 1ll * n * x + s - diff[x]);
48      }
49
50      return 0;
51  }
```