目录

The 17th UESTC Fun Programming Contest Round 2 1
B.Alice and the Dolls 1
C.Frogs Are Afraid at Night 3
D.Slacking Off with Git 5
E.Pensive Xiao Zhou 9

The 17th UESTC Fun Programming Contest Round 2

B.Alice and the Dolls

题目描述

Alice 有 n 个玩偶。第 i 个玩偶的初始高度为 a_i，目标高度为 b_i。

在一次操作中，Alice 选择任意一个玩偶集合（可以是空集）。

- 每个被选中的玩偶高度增加 1。
- 每个未被选中的玩偶高度减少 1。

接下来的 t 天里，Alice 的朋友 Marisa 每天都会来借书。每天，Alice 都从初始高度 a_i 开始，恰好执行 k 次操作。对于每一天，判断是否所有玩偶都能达到目标高度 b_i。

约束条件

- $1 \le t \le 10^4$;
- $1 \le n \le 2 \times 10^5$，$0 \le k \le 10^9$;
- $-10^9 \le a_i, b_i \le 10^9$;
- 所有测试数据中 n 之和不超过 2×10^5。

输入格式

第一行输入一个整数 t $(1 \le t \le 10^4)$，表示天数。

对于每组测试数据:

- 第一行包含两个整数 n 和 k;
- 第二行包含 n 个整数 $a_1, a_2, ..., a_n$;
- 第三行包含 n 个整数 $b_1, b_2, ..., b_n$。

输出格式

对于每组测试数据，如果 Alice 能让所有玩偶都达到目标高度，输出 `YES`，否则输出 `NO`。大小写不敏感。

样例

1.in / 1.out

```
5
3 3
2 -1 5
3 2 2
2 4
10 0
10 3
1 5
1
8
```

```
11  2 0
12  -5 7
13  -5 7
14  4 2
15  0 0 0 0
16  2 -2 0 1
```

```
1  YES
2  NO
3  NO
4  YES
5  NO
```

注释

在第一组测试数据中，Alice 可以在操作 1 和 2 中选择第一个玩偶，在全部三次操作中选择第二个玩偶，从不选择第三个玩偶。于是三个玩偶的高度分别变化 $+1$、$+3$、-3，最终高度为 $[3,2,2]$。因此答案是 YES。

在第二、三、五组测试数据中，不存在恰好 k 次操作的方案能让所有玩偶达到目标高度，因此答案都是 NO。

在第四组测试数据中，$k=0$，Alice 不执行任何操作。所有玩偶本来就处于目标高度，答案是 YES。

题解

每个玩偶的选择情况是完全不相关的。

如果我们一个玩偶操作了 c_i 步，它实际上就增加了 $c_i-(k-c_i)=2c_i-k$ 步，而我们实际上在解 $2c_i-k=b_i-a_i$ 这样一个整数方程。

所以先判断整除性，再看 c_i 是否在范围内。

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

typedef long long i64;

#define maxn 200007
int n;
i64 li[maxn], k, f = 1, v;
int T;
int main() {
    scanf("%d", &T);
    while (T--) {
        f = 1;

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

        for (int i = 1; i <= n; i++) {
            scanf("%lld", &v);
            int delta = v - li[i];
```

```
            if ((k + delta) % 2 != 0)
                f = 0;
            else {
                int x = (k + delta) / 2;
                if (x < 0 or x > k)
                    f = 0;
            }
        }

        if (f)
            puts("YES");
        else
            puts("NO");
    }
    return 0;
}
```

C.Frogs Are Afraid at Night

题目描述

有一个由 n 列、m 行组成的网格。网格上恰好有 n 只青蛙。

青蛙的位置用 (x, y) 表示，其中 x 是它所在的列，y 是它所在的行。

到了晚上，青蛙会感到害怕。为了让它们有安全感，对于任意一对位于相邻列的青蛙，它们 y 坐标之差的绝对值都不能超过 1。

所有青蛙都朝向 y 坐标减小的方向。白天，一只青蛙只能向前跳一格。每次跳跃使其 y 坐标减 1，并消耗 1 点体力。因此，青蛙的 y 坐标只能减小，永远不能增大。

求让所有青蛙在夜晚到来之前满足条件所需的最小总体力消耗。

约束条件

- $1 \le n \le 2 \times 10^5$，$1 \le m \le 10^9$；
- $1 \le x_i \le n$，$1 \le y_i \le m$；
- 所有 x_i 两两不同，因此每一列恰好有一只青蛙；
- 青蛙可能按任意顺序给出。

输入格式

第一行输入两个整数 n 和 m $(1 \le n \le 2 \times 10^5, 1 \le m \le 10^9)$，分别表示网格的列数和行数。

接下来 n 行，第 i 行输入两个整数 x_i 和 y_i $(1 \le x_i \le n, 1 \le y_i \le m)$，表示第 i 只青蛙的位置。

输出格式

输出一个整数，表示所需的最小总体力消耗。

样例

1.in / 1.out

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

```
1 0
```

2.in / 2.out

```
1 5 10
2 4 10
3 1 10
4 3 1
5 5 10
6 2 10
```

```
1 30
```

注释

在第一个样例中，按列排序后青蛙的 y 坐标为 $[1,2,3]$，相邻列青蛙的 y 坐标之差都不超过 1，因此没有青蛙需要跳跃。

在第二个样例中，按列排序后青蛙的初始 y 坐标为 $[10,10,1,10,10]$。一种最优的最终配置是 $[3,2,1,2,3]$。因此所需的最小总体力为

$$(10-3)+(10-2)+(1-1)+(10-2)+(10-3)=30 \text{ 。}$$

题解

这种题目不推式子去做题意的简化，是相当危险的！

我们可以这样想，如果当前这个青蛙和相邻的这个排成了一列，其实它们两个是遵循同一个 dp 方程在相同点的状态转移的。

我写详细一点，我们来推断一下每个点的情况，为此我们需要枚举不动的点。

但是发现这样的情况（其实很多贪心什么的情况都）可以被 `1 2 10 11` 卡掉，我们思考这样一个思路：

令一列数 $z_i \le y_i, \sum(y_i - z_i)$ 就是我们的答案，显然我们希望 z_i 尽量大。

怎么去约束呢？我们发现我们实际上可以用每一个点的高度来约束当前点的最终高度 z_i，然后自然地让 z_i 受到所有约束，形式化地：

$$z_i = \min_{j \neq i}(y_j + |j-i|)$$

这样的话，如果我本身就很高，我对其他点的约束就不管用，但还是可以统一处理。

像上面 `1 2 10 11` 这个例子，`10 11` 同时被前面的 `1, 2` 所约束了，都会计算出正确的高度。

为了计算 z_i，我们需要计算它的前缀最小值和后缀最小值，然后发现这个东西实际上需要我们维护最小值所在的位置和值，或者说，往右移一步，最小值加一。

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

typedef long long i64;

#define maxn 200007
int n, m, u, v;
i64 li[maxn], lmin[maxn], rmin[maxn];
```

```cpp
std::vector<std::pair<i64, i64>> ve;
int main() {
    scanf("%d%d", &n, &m);
    for (int i = 1; i <= n; i++)
        scanf("%d%d", &u, &v), ve.push_back({u, v});
    std::sort(ve.begin(), ve.end());
    for (auto [i, v] : ve)
        li[i] = v;

    for (int i = 1; i <= n; i++)
        lmin[i] = rmin[i] = (1ll << 61);

    i64 minv = 1ll << 61;
    for (int i = 1; i <= n; i++) {
        minv++;
        if (minv > li[i])
            minv = li[i];

        lmin[i] = minv;
    }

    minv = 1ll << 61;
    for (int i = n; i >= 1; i--) {
        minv++;
        if (minv > li[i])
            minv = li[i];

        rmin[i] = minv;
    }

    i64 ans = 0;
    for (int i = 1; i <= n; i++)
        ans += li[i] - std::min(lmin[i], rmin[i]);

    printf("%lld", ans);

    return 0;
}
```

D.Slacking Off with Git

题目描述

本题是一道多趟（multi-pass）题。

Alice 和 Bob 想在工作时偷偷聊天。为了不被老板发现，他们决定通过修改 Git 仓库中一个无关紧要的字符串来传递消息。不过他们每次只能修改有限个字符。

具体来说，在第一次运行时，给定一个长度为 n 的字符串 s 和一个长度为 m 的消息 x。你必须修改 s 中至多 k 个字符，得到字符串 t。你不能插入或删除字符。在第二次运行时，给定第一次运行中的原始字符串 s 以及你的程序在第一次运行中输出的字符串 t。你必须还原出消息 x。

输入格式

你的程序会运行两次：先作为发送方，再作为接收方。读入第一个字符串 `alice` 或 `bob` 来判断要模拟哪个角色。

第一次运行

第一次运行的第一行是一个字符串 `alice`。第二行是一个整数 T $(1 \le T \le 1000)$，表示测试数据组数。

对于每组测试数据，第一行包含三个整数 n, m, k，分别表示 s 的长度、x 的长度以及 Alice 最多可以修改的字符数。接下来两行包含 s 和 x。所有字符串只包含小写英文字母，且 $|s| = n$、$|x| = m$。

第二次运行

第二次运行的第一行是一个字符串 `bob`。第二行是一个整数 T $(1 \le T \le 1000)$，表示测试数据组数。测试数据的组数和顺序都与第一次运行相同。

对于每组测试数据，第一行包含两个整数 n 和 m，分别表示 s 的长度和 Alice 想要传递的消息长度。接下来两行包含 s 和 t，分别表示第一次运行中的原始字符串 s 以及从第一次运行收到的字符串 t。所有字符串只包含小写英文字母，且 $|s| = n$、$|t| = n$。

在所有非样例测试中，$n = 600$、$m = 40$、$k = 20$。为了便于阅读，样例使用 $n = 40$、$m = 2$、$k = 1$。不过你的程序不需要针对样例的这种设定做特殊处理。

输出格式

第一次运行

输出 T 行，按顺序对应每组测试数据。

对于每组测试数据，输出一行，包含一个只由小写英文字母组成的字符串 t，表示发送给 Bob 的消息。你应保证 t 的长度为 n，且与对应的字符串 s 至多相差 k 个位置。

第二次运行

输出 T 行，按顺序对应每组测试数据。

对于每组测试数据，输出还原出的消息 x。它必须与第一次运行中对应测试数据的消息完全相同。

样例

第一次运行

```
alice
2
40 2 1
abcdefghijklmnopqrstuvwxyzabcdefghijklmn
hi
40 2 1
zyxwvutsrqponmlkjihgfedcbazyxwvutsrqponm
ok
```

```
zbcdefghijklmnopqrstuvwxyzabcdefghijklmn
zyxwvutsrqponmlkjihgfedcbazyxwvutsrqpona
```

第二次运行

```
bob
2
40 2
```

```
abcdefghijklmnopqrstuvwxyzabcdefghijklmn
zbcdefghijklmnopqrstuvwxyzabcdefghijklmn
40 2
zyxwvutsrqponmlkjihgfedcbazyxwvutsrqponm
zyxwvutsrqponmlkjihgfedcbazyxwvutsrqpona
```

```
hi
ok
```

注释

原题样例中的换行只是为了排版美观，实际测试时并不存在这些换行。

在第一组测试数据中，s 只有第一个字符从 a 变成了 z；在第二组测试数据中，只有最后一个字符从 m 变成了 a。每组测试数据都只用了一次修改。在第二次运行中，Bob 收到对应的原始字符串和修改后的字符串，并输出原始消息。

这些样例只演示了两次运行以及修改次数限制。所示的输出只是针对这些特定测试数据的一种可能示例，并不规定通用的通信方案。只要你的程序能在第二次运行中还原消息，输出不同的合法字符串 t 也是允许的。

本地测试

题目附带的脚本 `git-fish-chat-checker.py` 可以自动运行你的程序两次并检查结果：

```
g++ git-fish-chat.cpp -o git-fish-chat &&
python3 git-fish-chat-checker.py --input 1.txt ./git-fish-chat
```

文件 `1.txt` 必须使用第一次运行的输入格式。

题解

给你那么长一个字符串……只允许修改一半信息，但发现我们可以通过修改下标来传递信息。

具体地，比如把字符串切成 30 个一份的，每组传递两个字符的信息。我们修改下标 0-25 的某个字符，使其变得和原来不一样（例如循环加），这就能传递两个信息（下标，加上的值）。

唯一有一点，如果循环加的值是 0，怎么办？我们的接收端就得不到更改。

此时我们可以在下标 27（当时随便写的）的位置将答案直接修改成另一个字符，读出即可。

这样即使最终没有修改这个字符，程序还是能读出正确的值。 这个实现真是丑死我了！

```cpp
#include <iostream>
#include <stdio.h>
#include <string>
using std::cin;
using std::cout;
using std::endl;
using std::string;

int T;
string t, s, x;

int de(char c) { return c - 'a'; }
int mv(int x) { return (x + 260) % 26; }
char en(int x) { return x + 'a'; }
```

```
15
16  int n, m, k;
17  void run_alice() {
18      cin >> n >> m >> k >> s >> t;
19      int d = m / 2;
20
21      for (int i = 0; i < d; i++) {
22          int a = de(t[2 * i]), b = de(t[2 * i + 1]);
23          int base = 30 * i;
24
25          if (b != 0) {
26              int ind = base + a;
27              s[ind] = en(mv(de(s[ind]) + b));
28          } else {
29              s[base + 27] = en(a);
30          }
31      }
32
33      cout << s << endl;
34  }
35
36  void run_bob() {
37      cin >> n >> m >> s >> t;
38      int d = m / 2;
39
40      for (int i = 0; i < d; i++) {
41          // cout << "group " << i << " / " << d << endl;
42          int base = 30 * i;
43          int f = 0;
44          for (int d = 0; d < 26; d++)
45              if (s[base + d] != t[base + d])
46                  f = 1,
47                  cout << en(d) << en(mv(de(t[base + d]) - de(s[base + d])));
48
49          if (f == 0)
50              // cout << "[" << s[base + 27] << 'a' << "]";
51              cout << t[base + 27] << 'a';
52      }
53      cout << endl;
54  }
55
56  int main() {
57      cin >> x;
58      if (x == "alice") {
59          cin >> T;
60          while (T--)
61              run_alice();
62      }
```

```
    if (x == "bob") {
        cin >> T;
        while (T--)
            run_bob();
    }
    return 0;
}
```

E.Pensive Xiao Zhou

题目描述

来自 UESTC-XCPC 的小周想在他的第一次 ICPC 区域赛中拿到金牌。他计划训练 n 天。他选好了 n 场过去的比赛，编号为 $1, 2, ..., n$。他计划每天恰好虚拟参加（VP）其中一场比赛，每场比赛恰好参加一次。

第 i 场比赛有两个参数 a_i 和 b_i。如果他第 j 天参加第 i 场比赛（$1 \le i, j \le n$），那么他当天的兴奋度为 $a_i + j \cdot b_i$。他的兴奋度只取决于所选的比赛和天数，与之前的日子无关。

如果他过于兴奋，他就会捂着肚子在 A232 教室放声大笑。这会暴露他的真实物种，并让他被 VegiCat 封禁。所以他需要保持“沉思”。然而，他每天的最大兴奋度越高，就越难维持这种状态。

他可以按任意顺序参加这 n 场比赛。求可能的最大每日兴奋度的最小值。只需要输出这个值，不需要输出比赛顺序。

形式化地，求

$$\min_{p_1,p_2,\ldots,p_n} \max_{1\le j\le n} \left(a_{p_j} + j \cdot b_{p_j}\right),$$

其中 $p_1, p_2, ..., p_n$ 是 $1, 2, ..., n$ 的一个排列。

约束条件

- $1 \le n \le 2 \times 10^5$;
- $-10^9 \le a_i, b_i \le 10^9$。

输入格式

第一行输入一个整数 n $(1 \le n \le 2 \times 10^5)$，表示天数，同时也是比赛数量。

接下来 n 行，第 i 行输入两个整数 a_i 和 b_i $(-10^9 \le a_i, b_i \le 10^9)$，表示第 i 场比赛的参数。

输出格式

输出一个整数，表示最大每日兴奋度的最小可能值。答案可能为负数。

样例

1.in / 1.out

```
3
8 1
0 4
3 -1
```

```
9
```

2.in / 2.out

```
4
```

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

```
1 4
```

注释

在第一个样例中，一种最优安排是按给定顺序参加比赛：

- 第 1 天参加比赛 1，兴奋度为 $8+1\cdot 1=9$。
- 第 2 天参加比赛 2，兴奋度为 $0+2\cdot 4=8$。
- 第 3 天参加比赛 3，兴奋度为 $3+3\cdot(-1)=0$。

最大值为 9。如果交换前两场比赛，兴奋度变为 $4,10,0$，最大值为 10，更差。可以证明 9 是最优的。

注意答案可能超出 32 位整数范围，因此应使用 64 位整数（如 C++ 中的 `long long`）以避免整数溢出。

题解

还是不难想的……补题的时候还以为是 f 激动了半天……

第一个转换，$a_k+i\cdot b_k$ 是一条直线。

然后发现答案显然具有单调性，$a_k+i\cdot b_k\le x$ 对更大的 x 是显然成立的。

那还说啥了，二分答案！

然后呢？这个时候我灵机一动，我们要检查答案，就是在把 $[1,n]$ 的值一个个分配给直线，让它去取值。

什么样的值是合适的呢？我们反解直线方程！

$$a+ib\le x\to\begin{cases}[a\le x],b=0\\ i\le\left\lfloor\frac{x-a}{b}\right\rfloor,b>0\\ i\ge\left\lceil\frac{x-a}{b}\right\rceil,b<0\end{cases}$$

注意 `ceil` 在 $b<0$ 时的实现。

然后我们就得到 n 个区间（超出范围显然可以截断到端点），然后我们去做纯粹的贪心区间匹配。

把区间按照左端点降序为第一关键字，右端点降序为第二关键字排序，从 n 到 1 依次判断每个点是否落在每个区间里。

由于出现了负数，请注意二分的实现，以及向 0 舍入的性质。

cpp
```cpp
#include <algorithm>
#include <functional>
#include <stdio.h>
#define maxn 200007

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

int n;

```

```cpp
pii li[maxn];

inline i64 ceildiv(i64 a, i64 b) {
    if (a >= 0)
        return (a + b - 1) / b;
    return a / b;
}

// a + bk <= x [kmin, kmax]
pii cal_range(pii p, i64 x) {
    i64 a = p.first, b = p.second, L, R;
    if (b == 0) {
        if (a > x)
            return {1, -1};
        L = 1, R = n;
    } else if (b > 0) {
        L = 1;
        R = (x - a) / b;
    } else {
        i64 c = -b;
        L = ceildiv(a - x, c);
        R = n;
    }

    if (L < 1)
        L = 1;
    if (R > n)
        R = n;
    return {L, R};
}

pii iv[maxn];
bool check(i64 x) {
    for (int i = 1; i <= n; i++) {
        iv[i] = cal_range(li[i], x);
        if (iv[i].first > iv[i].second)
            return false;
    }

    std::sort(iv + 1, iv + n + 1, std::greater<pii>());

    for (int k = 0; k < n; k++) {
        i64 i = n - k;
        if (i < iv[k + 1].first or i > iv[k + 1].second)
            return false;
    }
    return true;
}
```

```
59
60  int main() {
61      scanf("%d", &n);
62      for (int i = 1; i <= n; i++)
63          scanf("%lld%lld", &li[i].first, &li[i].second);
64
65      i64 l = -(1ll << 60), r = (1ll << 60);
66      while (l < r) {
67          i64 mid = l + (r - l) / 2;
68          if (check(mid))
69              r = mid;
70          else
71              l = mid + 1;
72      }
73      printf("%lld\n", l);
74      return 0;
75  }
```