A. Бегущая строка

Условие.

На здании университета установлена бегущая строка. Символы появляются на строке справа налево по одной букве.

Изначально был подготовлен текст X для показа на бегущей строке. Но непосредственно перед запуском были внесены правки, и теперь нужно показать текст Y.

Технически нет возможности полностью перепрограммировать строку, однако можно в любой момент отключить её, чтобы пропустить один символ из текста X.

Ваша задача — определить максимальную длину префикса текста Y, который можно показать, и минимальное количество отключений строки, необходимых для достижения этого результата.

Формат ввода.

В первой строке задана строка X $(1 \leq |X| \leq 10^6)$ только из строчных букв английского алфавита.
Во второй строке задана строка Y $(1 \leq |Y| \leq 10^6)$ только из строчных букв английского алфавита.

Формат вывода.

В первой строке выведите минимальное необходимое количество отключений бегущей строки.
Во второй строке выведите максимальную длину префикса текста Y, который можно показать.

Примеры.

Пример 1

Ввод	Вывод
aabbcc abc	3 3

Пример 2

Ввод	Вывод
abc abc	0 3

Пример 3

Ввод	Вывод
abc def	3 0

Примечание

Если в какой-то момент невозможно продолжить показ префикса Y, то строку нужно выключить, пропустить один символ из X, и продолжить показ следующих символов Y. Можно отключать строку несколько раз подряд.

Если вы показали все символы из строки Y, то нужно продолжить отключать бегущую строку пока символы в ней не закончатся.

Система оценивания

При прохождение всех тестов, ваше решение получит 100 баллов.

Решения

Язык C++

```
#include <chrono>
#include <iostream>
#include <map>
#include <random>
#include <vector>

#define sz(a) ((int)((a).size()))
#define make_unique(x)
\
  sort((x).begin(), (x).end());
\
  (x).erase(unique((x).begin(), (x).end()), (x).end())

using namespace std;
mt19937 rnd(chrono::steady_clock::now().time_since_epoch().count());

typedef long long ll;
```

```
typedef long double ld;

void solve() {
  string a, b;
  cin >> a >> b;
  int cnt = 0;
  int ans = 0;
  int i1 = 0;
  int i2 = 0;
  while (i1 < ssize(a) && i2 < ssize(b)) {
    if (a[i1] == b[i2]) {
      ans++;
      ++i1;
      ++i2;
    } else {
      ++i1;
      ++cnt;
    }
  }
  cnt += ssize(a) - i1;
  cout << cnt << '\n' << ans << '\n';
}

int32_t main() {
  ios::sync_with_stdio(0);
  cin.tie(0);
  cout.tie(0);
  int t = 1;
  while (t--) {
    solve();
  }
}
```

Язык Python

```
a = input()
b = input()
cnt = 0
ans = 0
i1 = 0
i2 = 0

while i1 < len(a) and i2 < len(b):
    if a[i1] == b[i2]:
        ans += 1
        i1 += 1
        i2 += 1
    else:
```

```
            i1 += 1
            cnt += 1

    cnt += len(a) - i1

    print(cnt)
    print(ans)
```

B. Контроль топливных стержней

Условие.

Главный инженер ядерной электростанции, профессор К., разрабатывает новую систему безопасности для мониторинга состояния топливных стержней в реакторе.

У нас есть топливный стержень длиной n единиц, представленный как отрезок $[0, n]$. По этому стержню перемещаются два контрольных нейтронных датчика. Первый датчик находится в позиции p_1 и двигается со скоростью v_1 (то есть он проходит v_1 единиц стержня за секунду). Второй датчик находится в позиции p_2 и двигается со скоростью v_2.

Начиная со своих начальных позиций, датчики могут двигаться вдоль стержня. Они не могут выходить за пределы стержня. Чтобы изменить направление движения, датчик должен сначала достичь одного из концов стержня (точки 0 или точки n), и только там он может развернуться.

Помогите профессору К. вычислить минимально возможное время, за которое каждая точка топливного стержня будет просканирована хотя бы одним нейтронным датчиком для обеспечения полного контроля радиационной безопасности.

Формат ввода.

Первая строка содержит одно целое число t $(1 \le t \le 10000)$ – количество тестовых случаев.
Каждая из следующих t строк содержит пять чисел n, p_{1_i}, v_{1_i}, p_{2_i}, v_{2_i} $(0 < n \le 10000, 0 \le p_{1_i}, p_{2_i} \le n, 0.001 \le v_{1_i}, v_{2_i} \le 1000)$. Все числа имеют не более 3 цифр после десятичной точки.

Формат вывода.

Для каждого тестового случая выведите одно число – минимальное время, за которое каждая точка топливного стержня будет просканирована хотя бы одним нейтронным датчиком.

Ваш ответ считается правильным, если его абсолютная или относительная погрешность не превышает 10^{-6}.

Пример

Ввод	Вывод
2 10000.0 1.0 0.001 9999.0 0.001 4306.063 4079.874 0.607 1033.423 0.847	5001000.0000000000 3827.8370013755

Примечание

В первом примере время будет минимальным, если первый датчик поедет влево, а второй датчик поедет вправо.
Во втором примере время будет минимальным, если первый датчик поедет вправо, а второй датчик поедет влево.

Система оценивания

При прохождение всех тестов, ваше решение получит 100 баллов.

Решения

Язык C++

```cpp
#include <cmath>
#include <iomanip>
#include <iostream>

using namespace std;

using ll = long long;

ll safe_mult(ll a, ll b) {
  const ll INF = 1e18;
  if ((INF + b - 1) / b < a) {
    return INF;
  }
  return min(INF, a * b);
}

pair<ll, ll> build_seg(ll n, ll x, ll dx) {
  pair<ll, ll> seg = {min(x, x + dx), max(x, x + dx)};
  if (seg.first < 0) {
    seg.second = max(seg.second, -seg.first);
    seg.first = 0;
```

```
  }
  if (seg.second > n) {
    seg.first = max(0ll, min(seg.first, n + n - seg.second));
    seg.second = n;
  }
  return seg;
}

bool is_ok(ll n, ll x1, ll dx1, ll x2, ll dx2) {
  pair<ll, ll> first_seg = build_seg(n, x1, dx1);
  pair<ll, ll> second_seg = build_seg(n, x2, dx2);
  if (min(first_seg.first, second_seg.first) == 0 &&
      max(first_seg.second, second_seg.second) == n &&
      max(first_seg.first, second_seg.first) <=
          min(first_seg.second, second_seg.second)) {
  }
  return min(first_seg.first, second_seg.first) == 0 &&
         max(first_seg.second, second_seg.second) == n &&
         max(first_seg.first, second_seg.first) <=
             min(first_seg.second, second_seg.second);
}

void solve() {
  double dn, dp1, dv1, dp2, dv2;
  cin >> dn >> dp1 >> dv1 >> dp2 >> dv2;
  ll n = round(dn * 1e9);
  ll p1 = round(dp1 * 1e9);
  ll v1 = round(dv1 * 1000.0);
  ll p2 = round(dp2 * 1e9);
  ll v2 = round(dv2 * 1000.0);
  ll lb = 0;
  ll rb = 1e17;
  while (rb - lb > 1) {
    ll t = (lb + rb) >> 1;
    if (is_ok(n, p1, safe_mult(v1, t), p2, safe_mult(v2, t)) ||
        is_ok(n, p1, safe_mult(v1, t), p2, -safe_mult(v2, t)) ||
        is_ok(n, p1, -safe_mult(v1, t), p2, safe_mult(v2, t)) ||
        is_ok(n, p1, -safe_mult(v1, t), p2, -safe_mult(v2, t))) {
      rb = t;
    } else {
      lb = t;
    }
  }
  cout << setprecision(6) << fixed << (double)rb / 1000000.0 << '\n';
}

int main() {
  cin.tie(nullptr);
  cout.tie(nullptr);
  ios_base::sync_with_stdio(false);
```

```
  int t = 1;
  cin >> t;
  while (t--) {
    solve();
  }
  return 0;
}
```

Язык Python

```
import math

def safe_mult(a, b):
    INF = 10**18
    if ((INF + b - 1) // b < a):
        return INF
    return min(INF, a * b)

def build_seg(n, x, dx):
    seg = [min(x, x + dx), max(x, x + dx)]
    if seg[0] < 0:
        seg[1] = max(seg[1], -seg[0])
        seg[0] = 0
    if seg[1] > n:
        seg[0] = max(0, min(seg[0], n + n - seg[1]))
        seg[1] = n
    return seg

def is_ok(n, x1, dx1, x2, dx2):
    first_seg = build_seg(n, x1, dx1)
    second_seg = build_seg(n, x2, dx2)
    return (min(first_seg[0], second_seg[0]) == 0 and
            max(first_seg[1], second_seg[1]) == n and
            max(first_seg[0], second_seg[0]) <= min(first_seg[1],
second_seg[1]))

def solve():
    dn, dp1, dv1, dp2, dv2 = map(float, input().split())
    n = round(dn * 10**9)
    p1 = round(dp1 * 10**9)
    v1 = round(dv1 * 1000.0)
    p2 = round(dp2 * 10**9)
    v2 = round(dv2 * 1000.0)
    lb = 0
    rb = 10**17
    while rb - lb > 1:
        t = (lb + rb) >> 1
        if (is_ok(n, p1, safe_mult(v1, t), p2, safe_mult(v2, t)) or
```

```
            is_ok(n, p1, safe_mult(v1, t), p2, -safe_mult(v2, t)) or
            is_ok(n, p1, -safe_mult(v1, t), p2, safe_mult(v2, t)) or
            is_ok(n, p1, -safe_mult(v1, t), p2, -safe_mult(v2, t))):
            rb = t
        else:
            lb = t
    print(f"{rb / 1000000.0:.6f}")

def main():
    t = int(input())
    for _ in range(t):
        solve()

if __name__ == "__main__":
    main()
```

C. Любишь задачки порешать - люби и глину помесить

Условие.

В далеком будущем, когда цифровой мир полностью интегрировался с реальностью, возникла угроза, способная разрушить всю цифровую инфраструктуру планеты. Группа элитных программистов узнала, что массивы данных A и B содержат критическую информацию, необходимую для поддержания цифрового баланса.

Массивы A и B — это перестановки из n элементов, содержащие все числа от 1 до n без повторений. Ваша задача — привести массив A к массиву B, используя допустимую операцию "hop". Операция "hop" для массива A состоит в выборе индекса j от 1 до $n - 2$, затем элементы A[0], A[j], A[n - 1] становятся равны A[j], A[n - 1], A[0] соответственно.

Однако времени мало — у вас есть не более 2n операций, чтобы справиться с задачей. В противном случае, системы мирового правительства начнут завершающий цикл самоуничтожения: произойдет цифровой апокалипсис, который уничтожит всю электронику и сети, отправив человечество в Тёмные века технологии.

Из-за текущей угрозы каждое мгновение на счету. Пока массивы A и B остаются в этом хаотическом состоянии, человечество находится на грани катастрофы. Вы — последняя надежда; вселенная доверяется вашему мастерству и аналитическому мышлению.

Пожалуйста, примите этот вызов и используйте свои уникальные навыки для преобразования массива A в массив B. Помните, что у вас ограниченное число

попыток, и каждый ход имеет значение и может стать решающим.

Формат ввода.

В первой строке дано число N $(3 \leq N \leq 10^5)$.
Во второй строке дано N чисел a_i $(1 \leq a_i \leq N)$ - элементы массива A. Гарантируется, что все числа a_i - различны.
Во второй строке дано N чисел b_i $(1 \leq b_i \leq N)$ - элементы массива B. Гарантируется, что все числа b_i - различны.

Формат вывода.

В первой строке выведите число X - чисто необходимых действий для приведения массива A к массиву B.
Во второй строке выведите X чисел j_i - выбранные числа j для операций hop.
Если привести массив A к массиву B невозможно - выведите -1.

Пример.

Пример 1

Ввод	Вывод
5 1 2 3 4 5 1 3 4 2 5	10 3 2 1 3 2 1 3 2 1 3

Пример 2

Ввод	Вывод
5 1 2 3 4 5 1 4 3 2 5	-1

Примечание

Пояснения к первому тесту из условия.
hop(3): a = [4 2 3 5 1]
hop(2): a = [3 2 1 5 4]
hop(1): a = [2 4 1 5 3]
hop(3): a = [5 4 1 3 2]
hop(2): a = [1 4 2 3 5]
hop(1): a = [4 5 2 3 1]

hop(3): a = [3 5 2 1 4]
hop(2): a = [2 5 4 1 3]
hop(1): a = [5 3 4 1 2]
hop(3): a = [1 3 4 2 5]

Пояснения к второму тесту из условия:
можно доказать что невозможно привести массив A к такому массиву B.

Система оценивания

В данной задаче нет тестовых групп. Балл 100 ставится при полном решении.

Решения

Язык C++

```cpp
#include <cassert>
#include <iostream>
#include <vector>

using namespace std;

void apply_hop(vector<int> &array, vector<int> &index_by_number, int pos)
{
  assert(pos != 0);
  assert(pos + 1 != array.size());
  swap(array[0], array[pos]);
  swap(array[pos], array[array.size() - 1]);
  index_by_number[array[0]] = 0;
  index_by_number[array[pos]] = pos;
  index_by_number[array[array.size() - 1]] = array.size() - 1;
}

void solve() {
  int n;
  cin >> n;
  vector<int> a(n);
  vector<int> index_by_number_in_a(n + 1);
  for (int i = 0; i < n; ++i) {
    cin >> a[i];
    index_by_number_in_a[a[i]] = i;
  }
  vector<int> b(n);
  for (int i = 0; i < n; ++i) {
    cin >> b[i];
  }
  vector<int> actions;
  auto hop = [&a, &index_by_number_in_a, &actions](int pos) {
```

```
    apply_hop(a, index_by_number_in_a, pos);
    actions.push_back(pos);
  };
  int pos_to_fix = 1;
  while (pos_to_fix + 1 < n - 1) {
    if (index_by_number_in_a[b[pos_to_fix]] == pos_to_fix) {
      pos_to_fix += 1;
      continue;
    }
    if (index_by_number_in_a[b[pos_to_fix]] == n - 1) {
      hop(pos_to_fix);
      // b[pos_to_fix] moves from (n - 1) to pos_to_fix. b[pos_to_fix] is
fixed.
      pos_to_fix += 1;
      continue;
    }
    if (index_by_number_in_a[b[pos_to_fix]] == 0) {
      hop(pos_to_fix);
      // b[pos_to_fix] moves from 0 to (n - 1).
      hop(pos_to_fix);
      // b[pos_to_fix] moves from (n - 1) to pos_to_fix. b[pos_to_fix] is
fixed.
      pos_to_fix += 1;
      continue;
    }
    int other_problem = pos_to_fix + 1;
    while (other_problem < n - 1 &&
           index_by_number_in_a[b[other_problem]] == other_problem) {
      other_problem += 1;
    }
    if (other_problem == n - 1) {
      break;
    }
    // now we fix {pos_to_fix, other_problem} with at most 4 actions.
    if (index_by_number_in_a[b[other_problem]] == n - 1) {
      hop(other_problem);
      // b[other_problem] is fixed.
      if (index_by_number_in_a[b[pos_to_fix]] == n - 1) {
        hop(pos_to_fix);
        // b[pos_to_fix] is fixed.
        pos_to_fix = other_problem + 1;
        continue;
      }
      if (index_by_number_in_a[b[pos_to_fix]] == 0) {
        hop(pos_to_fix);
        // b[pos_to_fix] moves from 0 to (n - 1) in a.
        hop(pos_to_fix);
        // b[pos_to_fix] moves from (n - 1) to pos_to_fix. b[pos_to_fix]
is
        // fixed.
```

```
        pos_to_fix = other_problem + 1;
        continue;
      }
      hop(index_by_number_in_a[b[pos_to_fix]]);
      // b[pos_to_fix] moves from index_by_number_in_a[b[pos_to_fix]] to 0
in a.
      hop(pos_to_fix);
      // b[pos_to_fix] moves from 0 to (n - 1) in a.
      hop(pos_to_fix);
      // b[pos_to_fix] moves from (n - 1) to pos_to_fix. b[pos_to_fix] is
fixed.
      pos_to_fix = other_problem + 1;
      continue;
    }
    hop(index_by_number_in_a[b[pos_to_fix]]);
    // b[pos_to_fix] moves from index_by_number_in_a[b[pos_to_fix]] to 0
in a.
    if (index_by_number_in_a[b[other_problem]] == n - 1) {
      hop(other_problem);
      // b[pos_to_fix] moves from 0 to (n - 1).
      // b[other_problem] moves from (n - 1) to other_problem.
b[other_problem]
      // is fixed.
      hop(pos_to_fix);
      // b[pos_to_fix] moves from (n - 1) to pos_to_fix. b[pos_to_fix] is
fixed.
      pos_to_fix = other_problem + 1;
      continue;
    }
    hop(index_by_number_in_a[b[other_problem]]);
    // b[pos_to_fix] moves from 0 to (n - 1).
    // b[other_problem] moves from index_by_number_in_a[b[other_problem]]
to 0.
    hop(pos_to_fix);
    // b[pos_to_fix] moves from (n - 1) to pos_to_fix. b[pos_to_fix] is
fixed.
    // b[other_problem] moves from 0 to (n - 1).
    hop(other_problem);
    // b[other_problem] moves from (n - 1) to other_problem.
b[other_problem] is
    // fixed.
    pos_to_fix = other_problem + 1;
  }
  if (pos_to_fix != n - 1) {
    if (index_by_number_in_a[b[pos_to_fix]] == pos_to_fix) {
      // Positions 1...n-2 already correct.
    } else if (index_by_number_in_a[b[pos_to_fix]] == n - 1) {
      hop(pos_to_fix);
      // b[pos_to_fix] moves from (n - 1) to pos_to_fix. b[pos_to_fix] is
fixed.
```

```
    } else if (index_by_number_in_a[b[pos_to_fix]] == 0) {
      hop(pos_to_fix);
      // b[pos_to_fix] moves from 0 to (n - 1).
      hop(pos_to_fix);
      // b[pos_to_fix] moves from (n - 1) to pos_to_fix. b[pos_to_fix] is
fixed.
    } else {
      assert(false);
    }
  }
  for (int i = 0; i < n; ++i) {
    if (a[i] != b[i]) {
      cout << "-1\n";
      return;
    }
  }
  assert(actions.size() <= n + n);
  cout << actions.size() << '\n';
  for (auto action : actions) {
    cout << action << ' ';
  }
  cout << '\n';
}

int main() {
  cin.tie(nullptr);
  cout.tie(nullptr);
  ios_base::sync_with_stdio(false);
  int t = 1;
  while (t--) {
    solve();
  }
  return 0;
}
```

Язык Python

```
def apply_hop(array, index_by_number, pos):
    assert pos != 0
    assert pos + 1 != len(array)
    array[0], array[pos] = array[pos], array[0]
    array[pos], array[-1] = array[-1], array[pos]
    index_by_number[array[0]] = 0
    index_by_number[array[pos]] = pos
    index_by_number[array[-1]] = len(array) - 1

def solve():
    n = int(input().strip())
```

```
    a = list(map(int, input().strip().split()))
    index_by_number_in_a = [0] * (n + 1)

    for i in range(n):
        index_by_number_in_a[a[i]] = i

    b = list(map(int, input().strip().split()))
    actions = []

    def hop(pos):
        apply_hop(a, index_by_number_in_a, pos)
        actions.append(pos)

    pos_to_fix = 1

    while pos_to_fix + 1 < n - 1:
        if index_by_number_in_a[b[pos_to_fix]] == pos_to_fix:
            pos_to_fix += 1
            continue

        if index_by_number_in_a[b[pos_to_fix]] == n - 1:
            hop(pos_to_fix)
            pos_to_fix += 1
            continue

        if index_by_number_in_a[b[pos_to_fix]] == 0:
            hop(pos_to_fix)
            hop(pos_to_fix)
            pos_to_fix += 1
            continue

        other_problem = pos_to_fix + 1

        while other_problem < n - 1 and
index_by_number_in_a[b[other_problem]] == other_problem:
            other_problem += 1

        if other_problem == n - 1:
            break

        if index_by_number_in_a[b[other_problem]] == n - 1:
            hop(other_problem)

            if index_by_number_in_a[b[pos_to_fix]] == n - 1:
                hop(pos_to_fix)
                pos_to_fix = other_problem + 1
                continue

            if index_by_number_in_a[b[pos_to_fix]] == 0:
                hop(pos_to_fix)
```

```
                hop(pos_to_fix)
                pos_to_fix = other_problem + 1
                continue

            hop(index_by_number_in_a[b[pos_to_fix]])
            hop(pos_to_fix)
            hop(pos_to_fix)
            pos_to_fix = other_problem + 1
            continue

        hop(index_by_number_in_a[b[pos_to_fix]])

        if index_by_number_in_a[b[other_problem]] == n - 1:
            hop(other_problem)
            hop(pos_to_fix)
            pos_to_fix = other_problem + 1
            continue

        hop(index_by_number_in_a[b[other_problem]])
        hop(pos_to_fix)
        hop(other_problem)
        pos_to_fix = other_problem + 1

    if pos_to_fix != n - 1:
        if index_by_number_in_a[b[pos_to_fix]] == pos_to_fix:
            pass
        elif index_by_number_in_a[b[pos_to_fix]] == n - 1:
            hop(pos_to_fix)
        elif index_by_number_in_a[b[pos_to_fix]] == 0:
            hop(pos_to_fix)
            hop(pos_to_fix)
        else:
            assert False

    if a != b:
        print("-1")
    else:
        print(len(actions))
        print(' '.join(map(str, actions)))

def main():
    t = 1  # Число тестов, может быть изменено при необходимости
    for _ in range(t):
        solve()

if __name__ == "__main__":
    main()
```

D. Круги на полях

Условие.

Когда-то давно, в живописной деревне, окруженной зелеными холмами, располагалось необъятное поле, на котором словно царствовал огромный древний камень. Местные жители верили, что этот Камень Хранителя — не просто часть ландшафта, а особое место силы, которое защищает их деревню и приносит удачу. Вокруг камня часто собирались, чтобы проводить традиционные ритуалы и рассказывать истории.

Однажды ночью над полем появились странные огни — прилетели инопланетяне. Они осветили поле яркими лазерами в поисках Камня Хранителя. Наутро поле оказалось усыпано множеством идеальных кругов и несколькими лучами, центры и начала которых загадочным образом сходились именно в священном камне.

На поверхности камня проявилось загадочное послание инопланетян. Старый мудрец, проживающий в деревне, внимательно изучил его и понял, что для расшифровки этого послания необходимо многократно вычислять расстояния между особыми точками, лежащими на пересечении кругов и прямых. Эти расстояния - ключ к пониманию смысла послания, которое могло гораздо глубже связать жителей деревни с тайнами вселенной и наследием, оставленным внеземными посетителями. Также мудрец уточнил, что путь между точками должен проходить только по кругам и прямым, оставленными инопланетянами.

Формат ввода.

В первой строке вводится число N $(1 \le N \le 1001)$ - количество кругов.
Во второй строке вводятся N целых чисел r_i $(1 \le r_i \le 10^4)$ - радиусы окружностей на полях. Гарантируется что все радиусы различны.
В третьей строке вводится число M $(1 \le M \le 381)$ - количество лучей.
В следующих M строках вводится через пробел по 2 целых числа x_i y_i $(-100000 \le x_i \le 100000, -100000 \le y_i \le 100000)$ описывающие i-тый луч, начинающийся в точке (0, 0) проходящий через точку ($x_i y_i$).
Гарантируется, что $min(|x_i|, |y_i|) \neq 0$, а также что никакие 2 луча между собой не совпадают.

В последней строке вводится 3 чисел Q a b $(1 \le Q \le 5 * 10^7)$, $(0 \le a, b < 10^9)$, - эти числа описывают точки, между которыми необходимо рассчитать кратчайшее расстояние при движении или по кругам или лучам.
Пусть t_{-1} = 1, $t_{i+1} = (t_i * a + b)\%(10^9 + 7)$. Необходимо вычислить Q расстояний, где i -ый запрос состоит в поиске кратчайшего расстояния между точками (t_{4i+1} mod N, t_{4i+2} mod M) и (t_{4i+3} mod N, t_{4i+4} mod M), где первое число в паре - номер окружности, второе - номер луча.
Обратите внимание, что запросы, прямые и круги нумеруются с нуля.
Из-за наличия камня путь не может проходить через точку (0, 0).

Формат вывода.

Выведите одно число - сумму ответов на все Q запросов.
Пусть X - сумма ответов Жюри, Y - сумма Ваших ответов, которая будет считаться правильной, если
$\frac{|X-Y|}{2*Q} < 10^{-4}$.

Пример.

Пример 1

Ввод	Вывод
1 1 5 1 0 0 1 -1 0 0 -1 1 1 2 1 2	2.3561944902

Пример 2

Ввод	Вывод
3 1 2 3 5 1 0 0 1 -1 0 0 -1 1 1 4 1 2	12.8539816340

Примечание

При непройденных тестах из условия Ваше решение будет тестироваться дальше.

Пояснения к тесту 1:

В данном тесте 2 запроса.

- Для первого запроса необходимо вычислить $t_0 = 1 * 1 + 2 = 3$, $t_1 = 3 * 1 + 2 = 5$, $t_2 = 5 * 1 + 2 = 7$, $t_3 = 7 * 1 + 2 = 9$.
- t_0%N=3%1=0, t_1%M=5%5=0, t_2%N=7%1=0, t_3%M=9%5=4.
- Значит необходимо вычислить рассятоние между точками на пересечении круге номер 0 (радиуса 1) и лучей номер 0(1, 0) и 4(1, 1).
- Ответом будет длина дуги $pi/4$ окружности радиуса 1, что равняется 0.785398.

- Для второго запроса необходимо вычислить $t_4 = 11$, $t_5 = 13$, $t_6 = 15$, $t_7 = 17$.
- t_4%N=11%1=0, t_5%M=13%5=3, t_6%N=15%1=0, t_7%M=17%5=2.
- Значит необходимо вычислить рассятоние между точками на пересечении круге номер 0 (радиуса 1) и лучей номер 3(0, -1) и 2(-1, 0).
- Ответом будет длина дуги $pi/2$ окружности радиуса 1, что равняется 1.5708.

Пояснения к тесту 2:

- Ответ на запрос 1 будет: 1.785398 (R=1, луч (1, 0)) и (R=2, луч (1, 1)) - расстояние между точкой на пересечении окружности радиуса 1 и луча(1, 0) и точкой на пересечении окружности радиуса 2 и луча (1, 1).
- Ответ на запрос 2 будет: 3.570796 (R=3, луч (0, -1)) и (R=1, луч (-1, 0)).
- Ответ на запрос 3 будет: 4.141592 (R=2, луч (0, 1)) и (R=3, луч (1, 0))
- Ответ на запрос 4 будет: 3.356194 (R=1, луч (1, 1)) и (R=2, луч (0, -1))

Система оценивания

Группа 1. Гарантируется, что $Q = 1$, $N = 1$, $M \leq 2$. За эту группу вы получите 2 балла.
Группа 2. Гарантируется, что $Q = 1$, $N = 1$, $M \leq 100$. За эту группу вы получите 4 баллов. Будет тестироваться только при прохождении группы 1.
Группа 3. Гарантируется, что $Q = 1$, $N = 10$, $M \leq 10$. За эту группу вы получите 10 баллов. Будет тестироваться только при прохождении группы 2.
Группа 4. Гарантируется что $Q \leq 1000$. За эту группу вы получите 17 баллов. Будет тестироваться только при прохождении группы 3.
Группа 5. Гарантируется что $Q \leq 10^6$. За эту группу вы получите 29 баллов. Будет тестироваться только при прохождении группы 4.
Группа 6. За эту группу вы получите 38 баллов. Будет тестироваться только при прохождении группы 5.

Решения

Язык C++

```
#include <algorithm>
#include <cmath>
#include <iomanip>
#include <iostream>
```

```cpp
#include <vector>

using namespace std;

using ll = long long;
using ld = double;

const ll MOD = 1e9 + 7;
const ld two_pi = atan2(/*y=*/1, /*x=*/0) * (ld)4.0;

ld get_circle_dist(ld lhs, ld rhs) {
  ld diff = fabs(lhs - rhs);
  return min(diff, two_pi - diff);
}

void solve() {
  int n;
  cin >> n;
  vector<int> r(n);
  vector<int> get_sorted_number(n);
  {
    vector<pair<int, int>> r_input(n);
    for (int i = 0; i < n; ++i) {
      cin >> r_input[i].first;
      r_input[i].second = i;
    }
    sort(r_input.begin(), r_input.end());
    for (int i = 0; i < n; ++i) {
      r[i] = r_input[i].first;
      get_sorted_number[r_input[i].second] = i;
    }
  }
  int m;
  cin >> m;
  vector<pair<int, int>> rays(m);
  vector<ld> ray_angles(m);
  for (int i = 0; i < m; ++i) {
    cin >> rays[i].first >> rays[i].second;
    ray_angles[i] = atan2(rays[i].second, rays[i].first);
  }
  vector<vector<vector<ld>>> dp(n, vector<vector<ld>>(m, vector<ld>(m)));

  for (int i = 0; i < m; ++i) {
    for (int j = i; j < m; ++j) {
      dp[0][i][j] = get_circle_dist(ray_angles[i], ray_angles[j]) *
(ld)r[0];
    }
  }
  for (int circle_id = 1; circle_id < n; ++circle_id) {
    for (int i = 0; i < m; ++i) {
```

```
      for (int j = i; j < m; ++j) {
        dp[circle_id][i][j] = min(
            get_circle_dist(ray_angles[i], ray_angles[j]) *
(ld)r[circle_id],
            dp[circle_id - 1][i][j] +
                (ld)2 * (r[circle_id] - r[circle_id - 1]));
      }
    }
  }
  int q;
  ll a, b, t_actual = 1;
  cin >> q >> a >> b;
  ld sum_ans = 0;
  while (q--) {
    t_actual = (t_actual * a + b) % MOD;
    int first_circle_id = get_sorted_number[t_actual % n];
    t_actual = (t_actual * a + b) % MOD;
    int first_ray_id = t_actual % m;
    t_actual = (t_actual * a + b) % MOD;
    int second_circle_id = get_sorted_number[t_actual % n];
    t_actual = (t_actual * a + b) % MOD;
    int second_ray_id = t_actual % m;
    ld tmp_ans =
        dp[min(first_circle_id, second_circle_id)]
          [min(first_ray_id, second_ray_id)][max(first_ray_id,
second_ray_id)] +
        r[max(first_circle_id, second_circle_id)] -
        r[min(first_circle_id, second_circle_id)];
    sum_ans += tmp_ans;
  }
  cout << setprecision(10) << fixed << sum_ans << '\n';
}

signed main() {
  cin.tie(nullptr);
  cout.tie(nullptr);
  ios_base::sync_with_stdio(false);
  int t = 1;
  while (t--) {
    solve();
  }
  return 0;
}
```

Язык Python

```
import math

MOD = 10**9 + 7
two_pi = math.pi * 2

def get_circle_dist(lhs, rhs):
    diff = abs(lhs - rhs)
    return min(diff, two_pi - diff)

def solve():
    n = int(input())
    r = list(map(int, input().split()))
    r_input = [(r[i], i) for i in range(n)]
    r_input.sort()

    r = [r_input[i][0] for i in range(n)]
    get_sorted_number = [0] * n
    for i in range(n):
        get_sorted_number[r_input[i][1]] = i

    m = int(input())
    rays = []
    ray_angles = []
    for _ in range(m):
        x, y = map(int, input().split())
        rays.append((x, y))
        ray_angles.append(math.atan2(y, x))

    dp = [[[0.0 for _ in range(m)] for _ in range(m)] for _ in range(n)]

    for i in range(m):
        for j in range(m):
            dp[0][i][j] = get_circle_dist(ray_angles[i], ray_angles[j]) *
r[0]

    for circle_id in range(1, n):
        for i in range(m):
            for j in range(m):
                dp[circle_id][i][j] = min(
                    get_circle_dist(ray_angles[i], ray_angles[j]) *
r[circle_id],
                    dp[circle_id - 1][i][j] + 2 * (r[circle_id] -
r[circle_id - 1])
                )

    q, a, b = map(int, input().split())
    t_actual = 1
    sum_ans = 0.0
    for _ in range(q):
```

```
        t_actual = (t_actual * a + b) % MOD
        first_circle_id = get_sorted_number[t_actual % n]
        t_actual = (t_actual * a + b) % MOD
        first_ray_id = t_actual % m
        t_actual = (t_actual * a + b) % MOD
        second_circle_id = get_sorted_number[t_actual % n]
        t_actual = (t_actual * a + b) % MOD
        second_ray_id = t_actual % m

        tmp_ans = dp[min(first_circle_id, second_circle_id)][first_ray_id]
[second_ray_id] + \
                  r[max(first_circle_id, second_circle_id)] - \
                  r[min(first_circle_id, second_circle_id)]

        sum_ans += tmp_ans

    print(f"{sum_ans:.10f}")

if __name__ == "__main__":
    solve()
```

E. Лингвистический феномен

Условие.

Поликарп увлекается лингвистикой и исследует особые свойства текстов. У него есть строка длиной N символов — результат его последних исследований. Однажды ему написал известный коллекционер редких лингвистических феноменов и предложил сделку: он готов купить у Поликарпа особую подстроку, но только если она соответствует четырём строгим требованиям:

1. Эта подстрока должна являться префиксом всей строки Поликарпа;
2. Эта же подстрока должна являться суффиксом всей строки Поликарпа;
3. Эта же подстрока должна встречаться ещё как минимум дважды где-то в середине строки;
4. Все четыре вхождения этой подстроки (префикс, суффикс и два средних вхождения) не должны пересекаться друг с другом.

Другими словами, коллекционер хочет приобрести такую подстроку, которая входит в строку Поликарпа четыре раза без пересечений, причём первое вхождение начинается с начала строки, а последнее заканчивается в конце строки.

Коллекционер предложил платить за подстроку по одной монете за каждый символ. Поликарп, будучи практичным программистом, хочет найти максимально длинную подстроку, удовлетворяющую всем требованиям, чтобы получить наибольшую выгоду.

Помогите Поликарпу найти эту подстроку!

Формат ввода.

В первой строке дано число N - длина строки Поликарпа.
Далее дана строка S из латинских символов в нижнем регистре $(4 \leq |S| \leq N)$

Формат вывода.

В единственной строке выведите самую длинную подстроку, которая удовлетворяет всем заданным условиям коллекционера.
Выведите -1, если такой подстроки нету.

Пример.

Пример 1

Ввод	Вывод
9 aaaaaaaaa	aa

Пример 2

Ввод	Вывод
8 abcdefgh	-1

Пример 3

Ввод	Вывод
9 abababababa	a

Примечание

В первом примере ответом является подстрока "aa" (например, с вхождениями на позициях 1, 3, 5, 8)

Во втором примере нет подстроки, подходящей под условия.
В третьем примере ответом является подстрока "a" (например, с вхождениями на позициях 1, 3, 5, 9)

Система оценивания

Каждая группа будет тестироваться только при успешном прохождении предыдущей.

Группа	N	Баллы
1	$1 \leq N \leq 800$	7
2	$1 \leq N \leq 2 * 10^4$	10
3	$1 \leq N \leq 3 * 10^6$	14
4	$1 \leq N \leq 10^7$	69

Решения

Язык C++

```
#include <algorithm>
#include <chrono>
#include <iostream>
#include <map>
#include <random>
#include <vector>

#define sz(a) ((int)((a).size()))
#define make_unique(x)
\
  sort((x).begin(), (x).end());
\
  (x).erase(unique((x).begin(), (x).end()), (x).end())

using namespace std;
mt19937 rnd(chrono::steady_clock::now().time_since_epoch().count());

typedef long long ll;
typedef long double ld;

vector<int> z_f(const string &s) {
  vector<int> z(s.size());
  int l = 0, r = 0;
  for (int i = 1; i < ssize(s); ++i) {
    if (i <= r) {
      z[i] = min(z[i - l], r - i + 1);
    }
```

```
    while (i + z[i] < ssize(s) && s[z[i]] == s[i + z[i]]) {
      ++z[i];
    }
    if (i + z[i] - 1 > r) {
      r = i + z[i] - 1;
      l = i;
    }
  }
  z[0] = s.size();
  return z;
}

void solve() {
  int n;
  cin >> n;
  string s;
  cin >> s;
  auto z = z_f(s);
  vector<int> original_z = z;
  for (int i = 0; i < n; ++i) {
    // remove overlap with prefix
    z[i] = min(z[i], i);

    // remove overlap with suffix
    z[i] = min(z[i], (n - i) / 2);

    // add higher bound
    z[i] = min(z[i], n / 4);
  }
  vector<vector<int>> pos(n / 4 + 1);
  for (int i = 0; i < n; ++i) {
    pos[z[i]].emplace_back(i);
  }
  // leftmost and rightmost occurance
  int l = 1e9;
  int r = -1;
  for (int len = n / 4; len > 0; --len) {
    for (auto p : pos[len]) {
      l = min(l, p);
      r = max(r, p);
    }
    if (original_z[n - len] != len)
      continue;
    if (r - l >= len) {
      for (int i = 0; i < len; ++i) {
        cout << s[i];
      }
      cout << endl;
      return;
    }
```

```
  }
  cout << -1 << endl;
}

int32_t main() {
  ios::sync_with_stdio(0);
  cin.tie(0);
  cout.tie(0);
  int t = 1;
  while (t--) {
    solve();
  }
}
```

Язык Python

```
import time
from typing import List

def z_function(s: str) -> List[int]:
    n = len(s)
    z = [0] * n
    l, r = 0, 0
    for i in range(1, n):
        if i <= r:
            z[i] = min(z[i - l], r - i + 1)
        while i + z[i] < n and s[z[i]] == s[i + z[i]]:
            z[i] += 1
        if i + z[i] - 1 > r:
            r = i + z[i] - 1
            l = i
    z[0] = n
    return z

def solve():
    n = int(input())
    s = input()

    z = z_function(s)
    original_z = z.copy()

    for i in range(n):
        # remove overlap with prefix
        z[i] = min(z[i], i)

        # remove overlap with suffix
        z[i] = min(z[i], (n - i) // 2)
```

```
        # add higher bound
        z[i] = min(z[i], n // 4)

    pos = [[] for _ in range(n // 4 + 1)]
    for i in range(n):
        pos[z[i]].append(i)

    # leftmost and rightmost occurrence
    l = float('inf')
    r = -1

    for length in range(n // 4, 0, -1):
        for p in pos[length]:
            l = min(l, p)
            r = max(r, p)

        if original_z[n - length] != length:
            continue

        if r - l >= length:
            print(s[:length])
            return

    print(-1)

def main():
    t = 1
    for _ in range(t):
        solve()

if __name__ == "__main__":
    main()
```