Росатом2024_11_Тур_1

Задача 10-1

Условие: "Великий Семафорус"

В волшебной столице Элладины уже много лет проходит известная гонка колдунов "Великий Семафорус". Заклинание, призванное защищать городские перекрестки, приняло форму магического светофора с необычной логикой работы.

Этот светофор управляется тремя магическими элементами, последовательно меняющими друг друга: *пламя Красного Дракона* горит a_0 **секунд**, *сияние Желтой Луны* — **1 секунду** и *мощь Зеленого Грифона* — a_2 **секунд**. И эта заклинательная последовательность повторяется с самого начала гонки.

Каждый год, когда наступает день гонки, колдуны и волшебницы со всего мира собираются в Элладине, чтобы испытать свои повозки на скорость и мастерство. Однако, для того, чтобы победить в гонке, необходимо не только быстро управлять магической повозкой, но и точно рассчитать моменты пересечения магических светофоров.

Легендарный гонщик Меркурий Свифт готовится стартовать. Он знает, что подъехать к первому магическому светофору сможет через n **секунд** после его старта и сможет пересечь перекресток, как только загорится зеленый. Время отсчета идет с нуля, а светофор начинает работу с Красного Дракона.

Задача для юных магов-программистов — помочь Меркурию Свифту вычислить, в какую ближайшую секунду с момента старта он сможет безопасно пересечь перекресток.

Входные данные

В первой строке через пробел вводятся 2 числа: a_0 – продолжительность сигнала *Красного Дракона*, a_2 – *Зеленого Грифона*. $1 \leq a_0, a_2 \leq 10^9$.
Во второй строке вводится одно число: n – время, к которому повозка Меркурия подъедет к светофору. ($1 \leq n \leq 10^{15}$)

Выходные данные

В единственной строке выведите одно число – ближайшую секунду с момента старта, когда Меркурий Свифт сможет пересечь светофор.

Пример

2 5
2
ans:
3
Меркурий приедет во время активации желтого цвета, ему нужно будет подождать всего одну секунду.

5 1
23
ans:
27
Меркурий приедет на третьей секунде действия красного сигнала, ему нужно будет подождать 3 секунды до конца красного и еще одну секунду действия желтого.

Решение

```cpp
#include <iostream>
#include <stdint.h>

#define iom                                                    \
  std::cin.tie(0);                                             \
  std::cout.tie(0);                                            \
  std::ios_base::sync_with_stdio(false);
#define out std::cout.flush();
#define endl "\n"

using ll = long long;
using ull = unsigned long long;

void solve(std::istream &cin, std::ostream &cout) {
  ull a0, a2;
  cin >> a0 >> a2;

  ull n;
  cin >> n;

  ull cycle = a0 + (ull)1 + a2;

  ull remained = n % cycle;

  ull ans = 0;
  if (remained < a0) { // red
    ans = n + a0 - remained + 1;
  } else if (remained < a0 + 1) { // yellow
    ans = n + 1;
  } else { // green
    ans = n;
  }

  cout << ans << endl;
}

int32_t main() {
  iom;
```

```
  int T = 1; // cin >> T;
  while (T--) {
    solve(std::cin, std::cout);
  }

  out;
  return 0;
}
```

Задача 10-2

Условие: "Тайна Пикселизии"

В волшебном Королевстве Пикселизия археологи наткнулись на древние картинные галереи, украшенные сложными мозаичными композициями. Мозаика выстроена в виде идеального квадрата размером NxN, где каждый маленький фрагмент представляет собой пиксель, интенсивность которого измеряется целым числом.

К сожалению, времена не щадили эти произведения искусства: многие изображения поблекли и стали нечеткими. Чтобы вернуть прежнее очарование этим шедеврам, хранители галерей решили прибегнуть к тайной технике восстановления — медианной фильтрации. Согласно легендам, исполнение медианного фильтра 3x3 способно очистить эти изображения, при этом не внесет искажений в первозданный смысл их линий и цветов.

Медианный фильтр - это процедура в обрабатывании изображений, при которой значение каждого пикселя заменяется медианой значений яркости пикселей вокруг него, обычно определяемое по области 3x3. Медиана - это такое число в упорядоченной последовательности всех чисел, которое делит её пополам; если последовательность содержит чётное количество элементов, тогда берётся полусумма двух "срединных" чисел. Это позволяет избавиться от "шумов" и выпадающих из общего ряда значений, при этом не размывая сильно границы объектов на картине.

Обратите внимание, что для пикселей на краях мозаики полная окрестность из 3x3 не может быть сформирована, так как угловые и крайние пиксели имеют меньшее количество соседей. Эти пиксели не объединяются в фильтрацию и должны оставаться неизменными.

Вам, как признанному волшебнику программирования Пикселизии, предстоит выполнить задачу создания программы для медианной фильтрации древних изображений. Учитывайте, что изменения необходимо применять только к тем пикселям, у которых есть восемь соседей. Ваша работа поможет вернуть прошлую славу забытым мозаичным шедеврам.

Входные данные

В первой строке вводится одно целое число N ($3 \leq N \leq 1000$), обозначающее размер картинки.
Следующие N строк содержат по N целых чисел, разделенных пробелами, каждое из которых является значением яркости соответствующего пикселя мозаики и находится в диапазоне от *0* до *255* включительно.

Выходные данные

Выходные данные должны содержать *N* строк с *N* целыми числами в каждой, разделенными пробелами. Значения яркости должны быть результатом применения медианного фильтра к соответствующим пикселям входного изображения, кроме крайних пикселей, значения которых остаются без изменений (как описывалось выше).

Пример

3
1 9 10
21 43 12
43 1 3

Для среднего элемента: 1, 1, 3, 9, 10, 12, 21, 43, 43. Средний элемент = 10, поэтому 43 заменяется на 10. Остальные остаются без изменений.

1 9 10
21 10 12
43 1 3

Решение

```
#include <algorithm>
#include <iostream>
#include <string>
#include <vector>

#define iom                                                    \
  std::cin.tie(0);                                             \
  std::cout.tie(0);                                            \
  std::ios_base::sync_with_stdio(false);
#define end_flush std::cout.flush();
#define endl "\n"

using ll = long long;
using ull = unsigned long long;
using matrix = std::vector<std::vector<int>>;

int n;
const std::vector<std::pair<int, int>> diff = {
    {0, 1}, {1, 1}, {1, 0}, {1, -1}, {0, -1}, {-1, -1}, {-1, 0}, {-1, 1}};

int Calc(int i, int j, matrix &in) {
  std::vector<int> v;
  v.reserve(9);

  v.push_back(in[i][j]);
```

```
    for (auto &[dx, dy] : diff) {
      int new_i = i + dx, new_j = j + dy;
      if (0 <= new_i && new_i < n && 0 <= new_j && new_j < n) {
        v.push_back(in[new_i][new_j]);
      } else {
        return in[i][j];
      }
    }

    std::sort(v.begin(), v.end());
    return v[4];
  }

  void solve(std::istream &cin, std::ostream &cout) {
    cin >> n;

    matrix in(n, std::vector<int>(n));
    for (int i = 0; i < n; ++i) {
      for (int j = 0; j < n; ++j) {
        cin >> in[i][j];
      }
    }

    for (int i = 0; i < n; ++i) {
      for (int j = 0; j < n; ++j) {
        cout << Calc(i, j, in) << " ";
      }
      cout << endl;
    }
  }

  int32_t main() {
    iom;

    int T = 1; // cin >> T;
    while (T--) {
      solve(std::cin, std::cout);
    }

    end_flush;
    return 0;
  }
```

Задача 10-3

Условие: "Побег К."

Курица по кличке Коко решила сбежать из курятника, чтобы избежать участи оказаться в супе. Своё спасение она видит в побеге до соседнего леса, который находится в конце поля.

Поле представляет собой длинный участок прямой дороги, состоящий из **трех** параллельных полос (дорожек). Курица может стартовать с любой дорожки.

Каждая дорожка содержит определённое количество препятствий: ямы, лужи или другие куры не дают пройти Коко. Коко может сменить полосу — то есть переместиться на соседнюю дорожку. Перемещаться можно в любой момент, в том числе прямо перед препятствием. За один раз переместиться сразу на две полосы не получится.

Ваша задача — определить, с какой дорожки Коко должна начать свой бег, чтобы прийти на финиш, минимизировав количество смен полос. Если таких дорожек несколько, укажите дорожку с наименьшим номером. Если у Коко нет возможности сбежать, выведите `-1`.

Входные данные

В первой строке вводится число N ($1 \le N \le 10^7$) — длина дороги.
В следующих N-строках вводится описание дороги, где первая строка — позиция курятника, последняя — ближайшая к лесу позиция.
В строке для дорожки содержатся символы, где `'.'` обозначает свободный участок, а `'x'` — препятствие. Все строки имеют длину 3.

Выходные данные

Если у Коко нет возможности сбежать, в единственной строке выведите `-1`.
Иначе в первой строке выведите одно число — номер начальной дорожки, с которой необходимо стартовать.
Во второй строке выведите одно число — минимальное необходимое число смен полос.

Пример

Марштрут для теста 1
6

Начали с 1ой полосы и меняли полосу трижды.

Решение

```cpp
#include <algorithm>
#include <cstdint>
#include <iostream>
#include <string>
#include <vector>

#define iom                                                                    \
  std::cin.tie(0);                                                             \
  std::cout.tie(0);                                                            \
  std::ios_base::sync_with_stdio(false);
#define out std::cout.flush();
#define endl "\n"

using ll = long long;
using ull = unsigned long long;

#define MAX_VAL (INT64_MAX - 5LL)

ll n;
std::vector<std::vector<ll>> dp, p;

void move(std::vector<std::vector<ll>> &v, ll trash) {
  for (int i = 0; i < 3; ++i) {
    v[0][i] = v[1][i];
    v[1][i] = trash;
  }
}

void solve(std::istream &cin, std::ostream &cout) {
  cin >> n;

  dp.resize(2, std::vector<ll>(3, MAX_VAL));
  p.resize(2, std::vector<ll>(3, -1));
  std::string cur, prev;

  prev = "...";
  p[0][0] = 1;
  p[0][1] = 2;
  p[0][2] = 3;
  dp[0][0] = dp[0][1] = dp[0][2] = 0;

  for (int i = 0; i < n; ++i) {
    cin >> cur;

    if (cur[0] != 'x') {
      if (dp[0][0] != MAX_VAL && dp[1][0] > dp[0][0]) {
        dp[1][0] = dp[0][0];
        p[1][0] = p[0][0];
      }
      if (prev[0] != 'x' && dp[1][0] > dp[0][1] + 1) {
        dp[1][0] = dp[0][1] + 1;
        p[1][0] = p[0][1];
      }
    }

    if (cur[1] != 'x') {
      if (dp[0][1] != MAX_VAL && dp[1][1] > dp[0][1]) {
        dp[1][1] = dp[0][1];
        p[1][1] = p[0][1];
      }
      if (prev[1] != 'x') {
        if (dp[1][1] > dp[0][0] + 1) {
          dp[1][1] = dp[0][0] + 1;
          p[1][1] = p[0][0];
        }
        if (dp[1][1] > dp[0][2] + 1) {
          dp[1][1] = dp[0][2] + 1;
          p[1][1] = p[0][2];
        }
      }
    }

    if (cur[2] != 'x') {
      if (dp[0][2] != MAX_VAL && dp[1][2] > dp[0][2]) {
        dp[1][2] = dp[0][2];
        p[1][2] = p[0][2];
      }
      if (prev[2] != 'x' && dp[1][2] > dp[0][1] + 1) {
        dp[1][2] = dp[0][1] + 1;
        p[1][2] = p[0][1];
      }
    }

    move(dp, MAX_VAL);
    move(p, -1);
    prev = cur;
  }

  ll ans = dp[0][0], idx = 0;
  if (dp[0][1] < ans) {
    ans = dp[0][1];
    idx = 1;
  }
  if (dp[0][2] < ans) {
    ans = dp[0][2];
    idx = 2;
  }
  int parent = p[0][idx];
```

```
    if (ans == MAX_VAL || parent == -1) {
      cout << -1 << endl;
    } else {
      cout << parent << endl << ans << endl;
    }
  }

  int32_t main() {
    iom;

    int T = 1; // cin >> T;
    while (T--) {
      solve(std::cin, std::cout);
    }

    out;
    return 0;
  }
```

Задача 4

Условие "Основа основ"

В сердце каждой современной вычислительной системы лежит принцип мгновенной и безупречной синхронизации. И нет более важного компонента в этом процессе, чем генератор синхронизирующего сигнала, своевременно координирующий работу всех последовательностных частей ЭВМ.

Генератор синхросигнала обеспечивает слаженную работу всех систем и устройств, управляя моментами считывания, записи, передачи и преобразования данных.

В нашем идеализированном и упрощенном варианте ЭВМ, мы сталкиваемся с трудностью: синхросигнал от генератора должен доходить до каждого компонента одновременно, чтобы гарантировать, что все процессы в системе выполняются в пределах точно отрегулированных временных рамок и без каких-либо задержек.

Несинхронизированные компоненты могут привести к катастрофическим ошибкам в вычислениях, потере данных и сбою операций — чего мы ни в коем случае не допустим при проектировании нашей совершенной ЭВМ! Именно поэтому исключительно важно, чтобы длина каждого провода от генератора до схемы была **одинаковой**.

Схема ЭВМ представлена с некоторыми упрощениями:

- ЭВМ представляет собой прямоугольник в клетку.
- Одна из клеток - выход генератора синхронизирующей частоты.
- Все компоненты ЭВМ - прямоугольники, чьи стороны параллельны сторонам схемы ЭВМ.
- Провод с синхросигналом проходит через некоторую последовательность клеток, что каждая следующая клетка имеет хотя бы общую сторону с предыдущей (или совпадает с ней).
- В каждой клетке провод проходит через центр, возможно делая там поворот на 90 или 180 градусов.
- Провод может проходить по некоторым клеткам более одного раза.
- На пути провода любые 2 соседние клетки отличаются друг от друга. Например, клетка (1, 1) не может быть после клетки (1, 1). А последовательность клеток (1, 1), (1, 2), (1, 1), (1, 2) допустима.
- За границы схемы провода выходить не могут.

Ответом на задачу будет единственное число - минимальная сумма длин проводов, проведенных от генератора до компонентов ЭВМ.

Входные данные

В первой строке вводятся два числа: N, M - размеры схемы в клетках ($N \times M$). ($1 \le N, M \le 2000$).
Далее следует N строк, в каждой из которых по M символов $a_{i,j}$. ($a_{i,j} \in$ «.»,«X»,«I»,«B»).
Точка обозначает пустую клетку.
Символ "X" - позицию источника на схеме. Гарантируется, что ровно один символ "X" встретится на схеме.
Символ "B" обозначает, что клетка занята компонентом ЭВМ. Гарантируется, что каждый элемент схемы - прямоугольник без полостей внутри.
Элементы схемы не касаются друг друга, т.е. среди их клеток нет соседних по стороне.
Символ "I" обозначает выход схемы, куда надо подвести провод с синхросигналом. Гарантируется, что в каждом элементе схемы ровно одна клетка помечена как "I" и расположена на границе компонента, возможно в углу.
Каждый провод начинается в клетке с символом "X", заканчивается в клетке с символом "I".

Выходные данные

Выведите одно число - суммарную длину всех проводов с синхросигналом, если возможно сделать так, чтобы до каждого компонента ЭВМ сигнал доходил одновременно.
Выведите -1, если так сделать невозможно.

Пример

Группа - ($1 \le N, N \le 50$), все компоненты состоят из одной клетки "I". Также гарантируется, что до каждого компонента схемы существует путь по клеткам с длиной $|X_x - I_x| + |X_y - I_y|$, т.е. элементы ЭВМ не перекрывают все пути минимальной длины от источника до каждого элемента схемы. (20 баллов)
Группа - ($1 \le N, M \le 50$) (45 баллов)
Группа - полные ограничения (35 баллов)

Пояснения к 1 тесту.
Для подключения схемы в левом верхнем углу необходим провод длины хотя бы 5. Также два других компонента можно подключить проводом длины 5.
Для подключения компонента справа в середине (строка, считая сверху; столбец, считая слева) (5, 3)->(5, 4)->(5,5)->(4, 5)->(4, 4)->(3, 4) - один из примеров расположения провода.
А для подключения схемы слева снизу провод можно уложить так (5, 3)->(4, 3)->(4, 2)->(4, 3)->(4, 2)->(4, 1)

Решение

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

using namespace std;

void solve() {
    int n, m;
    cin >> n >> m;
    vector<vector<char>> pole(n, vector<char>(m));
    for (int x = 0; x < n; ++x) {
        for (int y = 0; y < m; ++y) {
            cin >> pole[x][y];
        }
    }

    // Search X coordinates
    int xX = -1, yX = -1;
    for (int x = 0; x < n; ++x) {
        for (int y = 0; y < m; ++y) {
            if (pole[x][y] == 'X') {
                if (xX == -1) {
                    xX = x;
                    yX = y;
                } else {
                    assert(false);
                }
            }
        }
    }
    assert(xX != -1);


    vector<vector<int>> used_check(n, vector<int>(m));
    auto isCorn = [&pole](int x, int y) {
        return pole[x][y] == 'B' || pole[x][y] == 'I';
    };
    auto valid = [&n, &m](int x, int y) {
        return (
            0 <= x && x < n &&
            0 <= y && y < m
        );
    };
    vector<pair<int, int>> targets;
    for (int x_corm = 0; x_corm < n; ++x_corm) {
        for (int y_corn = 0; y_corn < m; ++y_corn) {
            if (isCorn(x_corm, y_corn)) {
                if (used_check[x_corm][y_corn]) {
                    continue;
                }
                // New chema detected
                int wX = 0;
                while (x_corm + wX < n && isCorn(x_corm + wX, y_corn)) {
                    ++wX;
                }
                int wY = 0;
                while (x_corm + wY < m && isCorn(x_corm, y_corn + wY)) {
                    ++wY;
                }
                int prevTargetSize = targets.size();
                for (int dx = 0; dx < wX; ++dx) {
                    for (int dy = 0; dy < wY; ++dy) {
                        int x_now = x_corm + dx;
                        int y_now = y_corn + dy;
                        assert(isCorn(x_now, y_now));
                        used_check[x_now][y_now] = 1;
                        if (pole[x_now][y_now] == 'I') {
                            targets.push_back({x_now, y_now});
                        }
                    }
                }
                //validate around empty
                for (int dx : {-1, 0, wX - 1, wX}) {
                    for (int dy : {-1, 0, wX - 1, wY}) {
                        int cnt = 0;
                        if (dx == -1 || dx == wX) {
                            ++cnt;
                        }
                        if (dy == -1 || dy == wY) {
                            ++cnt;
                        }
                        if (cnt != 1) {
                            continue;
                        }
                        int x_now = x_corm + dx;
                        int y_now = y_corn + dy;
                        if (valid(x_now, y_now)) {
                            assert(used_check[x_now][y_now] == 0);
                        }
                    }
                }
                assert(prevTargetSize + 1 == targets.size());
            }
        }
```

```
    }

    queue<pair<int, int>> q;
    vector<pair<int, int>> d_coord = {{-1, 0}, {1, 0}, {0, 1}, {0, -1}};
    const int INF = 1e9 + 7;
    vector<vector<int>> dist(n, vector<int>(m, INF));
    q.push({xX, yX});
    dist[xX][yX] = 0;
    while (q.size()) {
        auto [x_tmp, y_tmp] = q.front();
        q.pop();
        for (auto [dx, dy] : d_coord) {
            int x_now = x_tmp + dx;
            int y_now = y_tmp + dy;
            if (valid(x_now, y_now) && dist[x_now][y_now] == INF) {
                if (pole[x_now][y_now] == '.' || pole[x_now][y_now] == 'I') {
                    dist[x_now][y_now] = dist[x_tmp][y_tmp] + 1;
                    if (pole[x_now][y_now] == '.') {
                        q.push({x_now, y_now});
                    }
                }
            }
        }
    }
    int max_dist = 0;
    set<int> osts;
    int ans = 0;
    for (auto [x_fin, y_fin] : targets) {
        if (dist[x_fin][y_fin] == INF) {
            ans = -1;
        } else {
            max_dist = max(max_dist, dist[x_fin][y_fin]);
            osts.insert(dist[x_fin][y_fin] % 2);
        }
    }
    if (ans == -1 || osts.size() == 2) {
        cout << -1 << endl;
        return;
    } else {
        cout << targets.size() * max_dist << endl;
        return;
    }
}

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

Задача 5

Условие

В далекой галактике живет юный исследователь Вовочка. В один прекрасный день, при изучении новой планеты, он наткнулся на загадочную древнюю цивилизацию. Было обнаружено, что эта цивилизация обладала способностью оставлять послания, состоящие из трех камней: `1, 2, 3`.

Послания можно разделить на два типа: "Правый взлет" (1-2-3) и "Левый взлет" (3-2-1). Вовочка также располагает специальной аппаратурой, которая позволяет ему изменять значение определенного камня, что помогает ему уточнить информацию или изучить новые варианты посланий.

Вовочке, как истинному исследователю, интересно, сколько таких посланий содержится на конкретном участке местности, который они обозначали в виде последовательного числового массива.

Существует два типа запросов.

- Запрос первого типа имеет вид `(? l r t)`. Необходимо искать количество "Правых взлетов" в отрезке [l, r], если t = 1. Если же t = 2, то запрос будет искать общее число взлетов: Левых и Правых. Одно число может быть частью двух взлетов одновременно. Например 1 2 3 2 1 содержит 2 взлета, где 3 - это последний элемент правого взлета и первый элемент левого.
- Запрос второго типа имеет вид `(! pos val)`. Необходимо изменить значение камня на позиции *pos* на заданное значение *val*.

  Помогите Вовочке расшифровать послания древних! Напишите программу, которая выполнит все запросы первого типа и выведет результат. Вывод каждого нового запроса должен начинаться с новой строки.

Входные данные

Первая строка содержит натуральное число N $(3 \le N \le 10^5)$ - длина массива.
Вторая строка содержит N натуральных чисел a_i $(1 \le a_i \le 3)$ - элементы исходного массива.
Третья строка содержит натуральное число Q $(1 \le Q \le 10^5)$ - число запросов.
Следующие Q строк описывают запросы в одном из 2 форматов.
Запрос первого типа имеет вид `? l r t` $(1 \le l < r \le N), (t \in \{1, 2\})$.
Запрос второго типа имеет вид `! pos val` $(1 \le pos \le N)$, $(1 \le val \le 3)$.

Гарантируется, что будет хотя бы один запрос первого типа.

Выходные данные

На каждый запрос одно вида выведите одно число - ответ на запрос. Каждый новый ответ должен начинаться с новой строки.

Пример

Группа 1 - гарантируется, что $(N, Q \leq 1000)$. За эту группу вы получите 25 баллов.
Группа 2 - нет запросов второго типа. За эту группу вы получите 35 баллов. Будет тестироваться только при прохождении группы 1.
Группа 3 - полные ограничения. За эту группу вы получите 40 баллов. Будет тестироваться только при прохождении группы 2.

Решение

```
#include <iostream>
#include <vector>

using namespace std;

class TSegtree {
public:
    TSegtree(int n) {
        MAXN = 1;
        while (MAXN < n) {
            MAXN += MAXN;
        }
        values.resize(MAXN + MAXN);
    }
    void set_value(int pos, int val) {
        set_value(pos, val, 1, 0, MAXN);
    }
    int get_sum(int le, int re) {
        return get_sum(le, re, 1, 0, MAXN);
    }

private:
    void set_value(int pos, int val, int v, int lb, int rb) {
        if (lb + 1 == rb) {
            values[v] = val;
            return;
        }
        int mid = (lb + rb) / 2;
        if (pos < mid) {
            set_value(pos, val, v + v, lb, mid);
        } else {
            set_value(pos, val, v + v + 1, mid, rb);
        }
        values[v] = values[v + v] + values[v + v + 1];
    }

    int get_sum(int le, int re, int v, int lb, int rb) {
        if (le <= lb && rb <= re) {
            return values[v];
        }
        if (re <= lb || rb <= le) {
            return 0;
        }
        int mid = (lb + rb) / 2;
        int left_ans = get_sum(le, re, v + v, lb, mid);
        int right_ans = get_sum(le, re, v + v + 1, mid, rb);
        return left_ans + right_ans;
    }

    int MAXN;
    vector<int> values;
};

int normalise(int val, int val_min, int val_max) {
    if (val <= val_min) {
        return val_min;
    }
    if (val >= val_max) {
        return val_max;
    }
    return val;
}

int get_sum(int l, int r, int n, TSegtree& segtree) {
    l = normalise(l, 0, n);
    r = normalise(r, 0, n);
    if (l >= r) {
        return 0;
    }
    int res = segtree.get_sum(l, r);
    return res;
}

bool my_compare(int pos_to_check, const vector<int>& numbers, int value_equal_to) {
    if (0 <= pos_to_check && pos_to_check < numbers.size()) {
        return numbers[pos_to_check] != value_equal_to;
    }
    return true;
}

void count_left(int pos, const vector<int>& numbers, TSegtree& segtree) {
    if (pos < 0 || pos >= numbers.size()) {
        return;
```

```
    }
    segtree.set_value(pos, 0);
    if (my_compare(pos + 2, numbers, 1) || my_compare(pos + 1, numbers, 2) || my_compare(pos, numbers, 3)) {
        return;
    }
    segtree.set_value(pos, 1);
}

void count_right(int pos, const vector<int>& numbers, TSegtree& segtree) {
    if (pos < 0 || pos >= numbers.size()) {
        return;
    }
    segtree.set_value(pos, 0);
    if (my_compare(pos - 2, numbers, 1) || my_compare(pos - 1, numbers, 2) || my_compare(pos, numbers, 3)) {
        return;
    }
    segtree.set_value(pos, 1);
}

void solve() {
    int n;
    cin >> n;
    vector<int> numbers(n);
    for (int pos_read = 0; pos_read < n; ++pos_read) {
        cin >> numbers[pos_read];
    }
    TSegtree left_side(n), right_side(n);
    for (int pos_init = 0; pos_init < n; ++pos_init) {
        count_left(pos_init, numbers, left_side);
        count_right(pos_init, numbers, right_side);
    }
    int q;
    cin >> q;
    for (int ask_id = 0; ask_id < q; ++ask_id) {
        char type;
        cin >> type;
        if (type == '!') {
            int pos, val;
            cin >> pos >> val;
            --pos;
            numbers[pos] = val;
            for (int pos_left_update = pos - 2; pos_left_update <= pos; ++pos_left_update) {
                count_left(pos_left_update, numbers, left_side);
            }
            for (int pos_right_update = pos; pos_right_update <= pos + 2; ++pos_right_update) {
                count_right(pos_right_update, numbers, right_side);
            }
        } else {
            int le, re, ask_type;
            cin >> le >> re >> ask_type;
            --le;
            int ans_right = get_sum(le + 2, re, n, right_side);
            int ans_left = get_sum(le, re - 2, n, left_side);
            int ans = ans_right;
            if (ask_type == 2) {
                ans += ans_left;
            }
            cout << ans << '\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;
}
```