«Алгоритмы и структуры данных»

Отчет по практической работе №5

Мустафаев Алихан

A. Бинарный поиск

ограничение по времени на тест

1 секунда

ограничение по памяти на тест

256 мегабайт

У Айбара есть массив 𝑎 из 𝑛 целых чисел, отсортированный по неубыванию. Учительница задала ему q вопросов. В каждом вопросе она спрашивает у Айбара, есть ли в массиве число x𝑥. Айбар, разумеется, без проблем справился с заданием. А сможете ли вы?

**Входные данные**

В первой строке входных данных даны два целых числа n𝑛 и q𝑞 (1 ≤ n, q ≤ 105) (1 ≤ 𝑛, 𝑞 ≤ 105).

Во второй строке заданы n из 𝑛 целых чисел a1, a2…, an 𝑎1, 𝑎2…, 𝑎𝑛 (−2 ∗ 109 ≤ ai ≤ 2 ∗ 109) (−2 ∗ 109 ≤ 𝑎𝑖 ≤ 2 ∗ 109) – элементы массива.

В третьей строке заданы q𝑞 целых чисел – x𝑥 ( −2 ∗ 109 ≤ x ≤ 2 ∗ 109) (−2 ∗ 109 ≤ 𝑥 ≤ 2 ∗ 109) – вопросы.

**Выходные данные**

Нужно вывести 𝑞 строк – есть ли в массиве число x.

#include <iostream>

#include <vector>

#include <algorithm>  // Для функции binary\_search

using *namespace* std;

*int* main() {

*int* n, q;

    cin >> n >> q;  // Вводим n (размер массива) и q (количество запросов)

    vector<*int*> a(n);  // Создаем вектор размера n для хранения элементов массива

    for (*int* i = 0; i < n; ++i) {

        cin >> a[i];  // Заполняем массив числами

    }

    for (*int* i = 0; i < q; ++i) {  // Обрабатываем каждый запрос

*int* x;

        cin >> x;  // Вводим число, которое нужно найти

        if (binary\_search(a.begin(), a.end(), x)) {  // Проверяем, есть ли x в массиве

            cout << "YES" << endl;  // Если есть, выводим "YES"

        } else {

            cout << "NO" << endl;  // Если нет, выводим "NO"

        }

    }

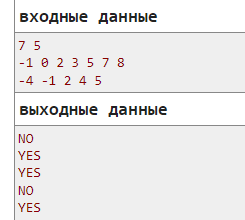
    return 0;

}

**Описание задачи**

У вас есть отсортированный массив a из n целых чисел, и вам нужно ответить на q запросов. В каждом запросе дается число x, и нужно выяснить, присутствует ли это число в массиве.   
 Если число есть — выводим "YES", если нет — "NO".  
**Описание кода**

**Пошаговое объяснение**

1. **Ввод чисел**:
   * Сначала вводятся два целых числа:
     + n — количество элементов в массиве.
     + q — количество запросов.
   * Затем вводится n чисел, которые будут храниться в массиве a.
   * Массив **уже отсортирован** (как указано в условии задачи).
2. **Выполнение запросов**:
   * Для каждого из q запросов вводится число x.
   * Используется функция binary\_search из библиотеки <algorithm>, которая проверяет, присутствует ли число x в массиве.
     + binary\_search выполняется за **логарифмическое время O(log n)**, что очень эффективно для больших массивов.
3. **Вывод результата**:
   * Если число найдено в массиве, выводится **"YES"**.  
     Если нет **"NO"**.  
     

B. Ближайшее число

ограничение по времени на тест

1 секунда

ограничение по памяти на тест

256 мегабайт

#include <iostream>

#include <vector>

#include <algorithm>

using *namespace* std;

*int* main() {

*int* n, q;

    cin >> n >> q;

    vector<*int*> a(n);  // Массив из n чисел

    for (*int* i = 0; i < n; ++i) {

        cin >> a[i];  // Вводим элементы массива

    }

    for (*int* i = 0; i < q; ++i) {  // Обрабатываем каждый запрос

*int* x;

        cin >> x;  // Вводим число из запроса

*auto* it = lower\_bound(a.begin(), a.end(), x);  // Ищем позицию x или ближайшего большего элемента

        if (it == a.begin()) {

            // Если x меньше или равен первому элементу массива

            cout << \*it << endl;

        } else if (it == a.end()) {

            // Если x больше всех элементов массива

            cout << \*(it - 1) << endl;

        } else {

            // Если нашли элемент посередине

*int* upper = \*it;       // Текущий элемент (ближайший больший или равный x)

*int* lower = \*(it - 1);  // Предыдущий элемент

            // Проверяем, какой элемент ближе к x

            if (x - lower <= upper - x) {

                cout << lower << endl;  // Выводим меньший при равенстве расстояний

            } else {

                cout << upper << endl;

            }

        }

    }

    return 0;

У Айбара есть массив a𝑎 из n𝑛 целых чисел, отсортированный по неубыванию. Учительница задала ему q𝑞 вопросов. В каждом вопросе она задает число x𝑥, и Айбар должен сказать ей ближайшее число к x𝑥 по значению в массиве. Если таких несколько, вывести меньшее число. Айбар, разумеется, без проблем справился с заданием. А сможете ли вы?

**Входные данные**

В первой строке входных данных даны два целых числа n𝑛 и q𝑞 (1≤n,q≤105)(1≤𝑛,𝑞≤105).

Во второй строке заданы n𝑛 целых чисел a1,a2,…,an𝑎1,𝑎2,…,𝑎𝑛 (−2∗109≤ai≤2∗109)(−2∗109≤𝑎𝑖≤2∗109) – элементы массива.

В третьей строке заданы q𝑞 целых чисел – x𝑥 (−2∗109≤x≤2∗109)(−2∗109≤𝑥≤2∗109) – вопросы.

**Выходные данные**

Для каждого из q𝑞 чисел выведите в отдельную строку число из массива a𝑎, наиболее близкое к данному. Если таких несколько, выведите меньшее из них.

**Описание задачи**

У вас есть **отсортированный массив** a из n целых чисел. Для каждого из q запросов задается число x, и нужно найти **наиболее близкое по значению число** в массиве a. Если есть два числа на одинаковом расстоянии от x, следует выбрать **меньшее из них**.

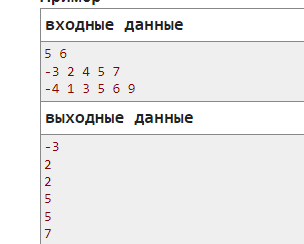
**Уточнение задачи**

* **Вход**:
  + n и q — количество элементов массива и запросов соответственно.
  + n целых чисел в массиве a, отсортированных по неубыванию.
  + q чисел x — числа из запросов.
* **Выход**:
  + Для каждого запроса нужно вывести **наиболее близкое** число из массива. Если два числа одинаково близки, выводим **меньшее** из них.

**Пошаговое объяснение**

1. **Ввод данных**:
   * Сначала вводим два целых числа:
     + n — количество элементов массива.
     + q — количество запросов.
   * Затем вводим n целых чисел массива a, который **отсортирован** по неубыванию.
2. **Обработка каждого запроса**:
   * Для каждого из q запросов читаем число x.
   * Используем функцию lower\_bound из библиотеки <algorithm>. Она находит **первую позицию**, где элемент массива **больше или равен** x.
3. **Проверка позиции it**:
   * **Если it указывает на начало массива** (a.begin()), то выводим первый элемент, так как x меньше или равен всем элементам.
   * **Если it указывает на конец массива** (a.end()), то выводим последний элемент, так как x больше всех элементов.
   * **Если it находится посередине**, проверяем, какой из двух соседних элементов — ближний к x. Если расстояния равны, выбираем **меньший** элемент.
4. **Вывод**:
   * Для каждого запроса выводится наиболее близкое число из массива.

**Временная сложность**

* **lower\_bound** работает за **O(log n)** для каждого запроса.
* Обработка всех запросов занимает **O(q \* log n)**, что достаточно быстро для больших значений n и q (до 100,000).  
    
  

C. Левая и правая позиция числа

ограничение по времени на тест

1 секунда

ограничение по памяти на тест

256 мегабайт

У Айбара есть массив a𝑎 из n𝑛 целых чисел, отсортированный по неубыванию. Учительница задала ему q𝑞 вопросов. В каждом вопросе она задает число x𝑥, и Айбар должен сказать ей индекс самого левого и самого правого вхождения данного числа в массиве. Айбар, разумеется, без проблем справился с заданием. А сможете ли вы?

**Входные данные**

В первой строке входных данных даны два целых числа n𝑛 и q𝑞 (1≤n,q≤105)(1≤𝑛,𝑞≤105).

Во второй строке заданы n𝑛 целых чисел a1,a2,…,an𝑎1,𝑎2,…,𝑎𝑛 (−2∗109≤ai≤2∗109)(−2∗109≤𝑎𝑖≤2∗109) – элементы массива.

В третьей строке заданы q𝑞 целых чисел – x𝑥 (−2∗109≤x≤2∗109)(−2∗109≤𝑥≤2∗109) – вопросы.

**Выходные данные**

Программа должна вывести q𝑞 строчек. Для каждого числа из второго списка нужно вывести индекс самого левого и самого правого вхождения данного числа в массиве. Нумерация начинается с единицы. Если число не входит в первый список, нужно вывести одно число 0.

**Описание задачи**

У вас есть **отсортированный массив** a из n целых чисел. Для каждого из q запросов задается число x. Нужно найти:

1. **Индекс самого левого вхождения** числа x в массив.
2. **Индекс самого правого вхождения** числа x в массив.

Если число **не присутствует** в массиве, нужно вывести **0**.

**Уточнение задачи**

* **Нумерация индексов начинается с 1**.
* Если число встречается несколько раз, нужно указать **первую и последнюю позицию** его вхождения.
* Если число отсутствует в массиве, вывести **0**.

#include <iostream>

#include <vector>

#include <algorithm>

using *namespace* std;

*int* main() {

*int* n, q;

    cin >> n >> q;

    vector<*int*> a(n);  // Создаем массив размера n

    for (*int* i = 0; i < n; ++i) {

        cin >> a[i];  // Вводим элементы массива

    }

    for (*int* i = 0; i < q; ++i) {  // Обрабатываем каждый запрос

*int* x;

        cin >> x;  // Вводим число x из запроса

*auto* lower = lower\_bound(a.begin(), a.end(), x);  // Находим первое вхождение числа x

*auto* upper = upper\_bound(a.begin(), a.end(), x) - 1;  // Находим последнее вхождение числа x

        if (lower == a.end() || \*lower != x) {  // Если число не найдено

            cout << 0 << endl;

        } else {  // Если число найдено

            cout << (lower - a.begin() + 1) << " " << (upper - a.begin() + 1) << endl;

        }

    }

    return 0;

}

Пошаговое объяснение

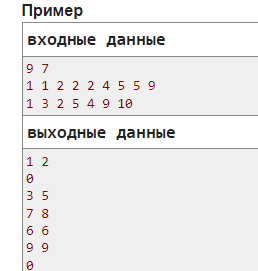
1. Ввод данных:
   * Вводим два целых числа: n (размер массива) и q (количество запросов).
   * Затем вводим n чисел в массив a. Массив уже отсортирован по неубыванию.
2. Обработка каждого запроса:
   * Для каждого из q запросов читаем число x.
   * Используем:
     + lower\_bound: Находит первую позицию числа x или ближайшего большего числа.
     + upper\_bound: Находит первую позицию, где элемент больше x. Чтобы получить последнюю позицию x, берем upper\_bound - 1.
3. Проверка наличия числа:
   * Если lower указывает на конец массива или элемент не равен x, значит, числа нет в массиве, и выводим 0.
   * Если число найдено:
     + Выводим индекс первого вхождения (lower - a.begin() + 1).
     + Выводим индекс последнего вхождения (upper - a.begin() + 1).

**Пояснение:**

* Для запроса x = 2: первое вхождение на 2-й позиции, последнее на 3-й позиции.
* Для запроса x = 4: число отсутствует, выводим **0**.
* Для запроса x = 1: первое и последнее вхождение на 1-й позиции.

**Временная сложность**

* **lower\_bound** и **upper\_bound** работают за **O(log n)**.
* Поэтому обработка всех запросов занимает **O(q \* log n)**, что эффективно для больших значений n и q (до 100,000).

****