A4. Разные алгоритмы решения одной* задачи

Демченко Георгий Павлович , БПИ-235

1.

Я не согласен с данным утверждением.

Результат работ всех 3-х алгоритмов могут отличаться в зависимости от входных данных.

Алгоритм 1:

- Находит число, которое встречается в массиве **A** наибольшее количество раз и возвращают его, если количество его вхождений $c \geq \frac{n}{2} + 1$
- В случае когда несколько различных чисел имеют одинаковое количество вхождений в массив, алгоритмы не возвращают ничего (тк $c \leq \frac{n}{2}$ в таком случае)

Алгоритм 3:

- Находит число, которое встречается в массиве **A** наибольшее количество раз и возвращают его, если количество его вхождений $c \geq \frac{n}{2} + 1$
- Если искомое число является наибольшим в массиве, то вывода не произойдет так как не встретится любого другого числа после него и не произойдет ветвление в **else** где располагается **return**

Алгоритм 2:

- **Бесполезен, ничего не решает в общем случае**
- **Если повезет с входными данными**:
    - Находит число, которое встречается в массиве **A** наибольшее количество раз и возвращает его
    - В случае когда несколько различных чисел имеют одинаковое количество вхождений в массив, алгоритм вернет то число, что было расположено ближе к концу массива наибольшее количество раз

Пример бесполезности 2-го алгоритма:

$$A = [8, 8, 8, 2, 2, 2, 5, 5]$$

Трассировка:

```
algorithm2 start

1-ая итерация
A[ind] = 8 A[i] = 8
c до проверки: 1
c после проверки: 2
--------------------
2-ая итерация
A[ind] = 8 A[i] = 8
c до проверки: 2
c после проверки: 3
--------------------
3-ая итерация
A[ind] = 8 A[i] = 2
c до проверки: 3
c после проверки: 2
--------------------
4-ая итерация
A[ind] = 8 A[i] = 2
c до проверки: 2
c после проверки: 1
--------------------
5-ая итерация
A[ind] = 8 A[i] = 2
c до проверки: 1
c после проверки: 0
новый ind: 5
--------------------
6-ая итерация
A[ind] = 2 A[i] = 5
c до проверки: 1
c после проверки: 0
новый ind: 6
--------------------
7-ая итерация
A[ind] = 5 A[i] = 5
c до проверки: 1
c после проверки: 2
--------------------

algorithm2 result:5
```

Результаты работ совпадают:

$$A = [1, 1, 2, 2, 2, 2, 4]$$

Трассировка:

Algorithm1	Algorithm2	Algorithm3

Algorithm1

```
algorithm1 start

1-ая итерация
Проверка вхождений 1 в массив
Количество вхождений 1 в массив: 2
Обновили c
--------------------
2-ая итерация
Проверка вхождений 1 в массив
Количество вхождений 1 в массив: 2
--------------------
3-ая итерация
Проверка вхождений 2 в массив
Количество вхождений 2 в массив: 4
Обновили c
--------------------
4-ая итерация
Проверка вхождений 2 в массив
Количество вхождений 2 в массив: 4
--------------------
5-ая итерация
Проверка вхождений 2 в массив
Количество вхождений 2 в массив: 4
--------------------
6-ая итерация
Проверка вхождений 2 в массив
Количество вхождений 2 в массив: 4
--------------------
7-ая итерация
Проверка вхождений 4 в массив
Количество вхождений 4 в массив: 1
--------------------
algorithm1 result:2
```

Algorithm2

```
algorithm2 start

1-ая итерация
A[ind] = 1 A[i] = 1
c до проверки: 1
c после проверки: 2
--------------------
2-ая итерация
A[ind] = 1 A[i] = 2
c до проверки: 2
c после проверки: 1
--------------------
3-ая итерация
A[ind] = 1 A[i] = 2
c до проверки: 1
c после проверки: 0
новый ind: 3
--------------------
4-ая итерация
A[ind] = 2 A[i] = 2
c до проверки: 1
c после проверки: 2
--------------------
5-ая итерация
A[ind] = 2 A[i] = 2
c до проверки: 2
c после проверки: 3
--------------------
6-ая итерация
A[ind] = 2 A[i] = 4
c до проверки: 3
c после проверки: 2
--------------------

algorithm2 result:2
```

Algorithm3

```
algorithm3 start

1-ая итерация
A[i - 1] = 1 A[i] = 1
c до проверки: 1
c после проверки: 2
--------------------
2-ая итерация
A[i - 1] = 1 A[i] = 2
c до проверки: 2
c после проверки: 1
--------------------
3-ая итерация
A[i - 1] = 2 A[i] = 2
c до проверки: 1
c после проверки: 2
--------------------
4-ая итерация
A[i - 1] = 2 A[i] = 2
c до проверки: 2
c после проверки: 3
--------------------
5-ая итерация
A[i - 1] = 2 A[i] = 2
c до проверки: 3
c после проверки: 4
--------------------
6-ая итерация
A[i - 1] = 2 A[i] = 4
c до проверки: 4
algorithm3 result:2
```

Результаты работ отличаются:

$A = [1, 1, 2, 2, 3, 3, 3]$

Трассировка:

Algorithm1 **Algorithm2** **Algorithm3**

Algorithm1

```
algorithm1 start

1-ая итерация
Проверка вхождений 1 в массив
Количество вхождений 1 в массив: 2
Обновили c
--------------------
2-ая итерация
Проверка вхождений 1 в массив
Количество вхождений 1 в массив: 2
--------------------
3-ая итерация
Проверка вхождений 2 в массив
Количество вхождений 2 в массив: 2
--------------------
4-ая итерация
Проверка вхождений 2 в массив
Количество вхождений 2 в массив: 2
--------------------
5-ая итерация
Проверка вхождений 3 в массив
Количество вхождений 3 в массив: 3
Обновили c
--------------------
6-ая итерация
Проверка вхождений 3 в массив
Количество вхождений 3 в массив: 3
--------------------
7-ая итерация
Проверка вхождений 3 в массив
Количество вхождений 3 в массив: 3
--------------------
algorithm1 no result
```

Algorithm2

```
algorithm2 start

1-ая итерация
A[ind] = 1 A[i] = 1
c до проверки: 1
c после проверки: 2
--------------------
2-ая итерация
A[ind] = 1 A[i] = 2
c до проверки: 2
c после проверки: 1
--------------------
3-ая итерация
A[ind] = 1 A[i] = 2
c до проверки: 1
c после проверки: 0
новый ind: 3
--------------------
4-ая итерация
A[ind] = 2 A[i] = 3
c до проверки: 1
c после проверки: 0
новый ind: 4
--------------------
5-ая итерация
A[ind] = 3 A[i] = 3
c до проверки: 1
c после проверки: 2
--------------------
6-ая итерация
A[ind] = 3 A[i] = 3
c до проверки: 2
c после проверки: 3
--------------------

algorithm2 result:3
```

Algorithm3

```
algorithm3 start

1-ая итерация
A[i - 1] = 1 A[i] = 1
c до проверки: 1
c после проверки: 2
--------------------
2-ая итерация
A[i - 1] = 1 A[i] = 2
c до проверки: 2
c после проверки: 1
--------------------
3-ая итерация
A[i - 1] = 2 A[i] = 2
c до проверки: 1
c после проверки: 2
--------------------
4-ая итерация
A[i - 1] = 2 A[i] = 3
c до проверки: 2
c после проверки: 1
--------------------
5-ая итерация
A[i - 1] = 3 A[i] = 3
c до проверки: 1
c после проверки: 2
--------------------
6-ая итерация
A[i - 1] = 3 A[i] = 3
c до проверки: 2
c после проверки: 3
--------------------
algorithm3 no result
```

2.

algorithm1 :

Вне зависимости от входных данных произойдет n **итераций цикла по** i**, в каждой их которых произойдет ещё** n **итераций цикла по** j

$\Rightarrow T(n) = O(n^2)\,,\, f(n) = n^2$

algorithm2 :

Вне зависимости от входных данных произойдет $n - 1$ **итерация цикла по** i

$\Rightarrow T(n) = O(n)\,,\, f(n) = n$

algorithm3 :

В случае когда наименьший элемент в массиве A и есть наиболее-повторяющийся и входит в массив более чем $\frac{n}{2}$ то произойдет $(\frac{n}{2})$ итераций цикла по i

В остальных случая произойдет $(n-1)$ итераций цикла по i

Верхняя оценка сложности алгоритма определяется сложностью применяемой сортировки, будем считать что в худшем случае она выполняется за $O(n \cdot \log_2(n))$ - merge sort

$\Rightarrow T(n) = O(n \cdot \log_2(n)) \, , f(n) = n \cdot \log_2(n)$

3.

Изменим алгоритмы 2 и 3 так, чтобы они работали аналогично алгоритму 1:

Algorithm2New.cpp

```
int32_t algorithm2New(std::vector<int32_t> &A, size_t n) {
    int32_t c = 1;
    int32_t ind = 0;
    std::unordered_map<int32_t, int32_t> countMap;
    int32_t ans = 0;
    int32_t max_cnt = 0;

    for (int32_t i = 1; i < n; ++i) {
        countMap[A[i]]++;
        if (A[ind] == A[i]) {
            c = c + 1;
        } else {
            c = c - 1;
        }

        if (c == 0) {
            ind = i;
            c = 1;
        }
        if (countMap[A[i]] > max_cnt) {
            max_cnt = countMap[A[i]];
            ans = A[i];
        }
    }

    if (countMap[ans] > n/2) {
        return ans;
    };
}
```

Algorithm3New.cpp

```
int32_t algorithm3New(std::vector<int32_t> &A, size_t n) {
    if (n == 1) {
```

```
            return A[0];
        }

        int32_t c = 1;
        std::sort(A.begin(), A.end());

        for (int32_t i = 1; i < n; ++i) {
            if (A[i - 1] == A[i]) {
                c = c + 1;
                if (c > n/2) {
                    return A[i - 1];
                }
            } else {
                c = 1;
            }
        }
    }
```

4.

algorithm2New :

Вне зависимости от входных данных произойдет $n - 1$ итерация цикла по i

$T(n)*{new} = O(n) |space, f(n)*{new} = n = f(n)$

algorithm3New :

Абсолютно аналогично изначальному алгоритму, поменяли расположение if (c > n/2) внутри ветвления.

Аналогично считаем, что применяется merge sort

$T(n)*{new} = O(n |cdot log_2(n)) |space, f(n)*{new} = n \cdot log_2(n) = f(n)$