Алгоритмы и структуры данных-1

SET 1. Задача A4.

Осень 2024. Клычков М. Д.

```
algorithm1:
  c = 0
  ind = -1

  for i = 0 to n
      c1 = 0
      for j = 0 to n
          if A[i] = A[j]
              c1 = c1 + 1
      if c1 > c
          c = c1
          ind = i

  if c > n / 2 return A[ind]
```

```
algorithm2:
  c = 1, ind = 0

  for i = 1 to n
      if A[ind] = A[i]
          c = c + 1
      else
          c = c - 1

      if c = 0
          ind = i
          c = 1

  return A[ind]
```

```
algorithm3:
  if n = 1
      return A[0]

  c = 1
  sort(A)

  for i = 1 to n
      if A[i - 1] = A[i]
          c = c + 1
      else
          if c > n / 2
              return A[i - 1]
          c = 1
```

Пункт 1. Кратко поясним, что делает каждый из приведенных алгоритмов:

- `algorithm1`: Результат работы алгоритма — элемент массива, который встречается больше, чем $\left\lfloor \frac{n}{2} \right\rfloor$ раз. Из приведенного псевдокода нельзя сделать вывод о том, что будем выведено, если такого элемента нет.

- `algorithm2`: Тяжело описать, что это «чудо» делает и к какому результату приведет... Фиксируется элемент (изначально это A_0), затем подсчитывается таких же элементов, причем, если встречается отличающийся элемент, то счетчик уменьшается на 1. Это можно представить графиком (по оси x — количество обработанных на данный момент элементов, y — значение счетчика) (рис. 1).

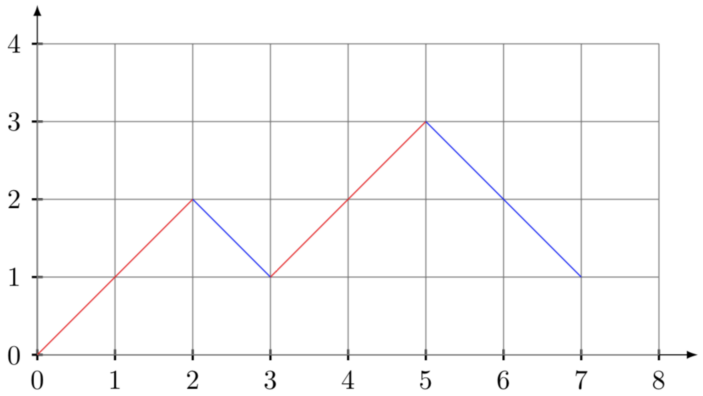

Рис. 1: Графическая интерпретация

После того как линия на графике достигает нуля, счетчик обнуляется и связывается с новым элементом массива. Возвращается элемент, с которым в последний раз был связан счетчик.

- `algorithm3`: Этот алгоритм делает то же самое, что и `algorithm1`, однако не возвращает результат в том случае, если искомый элемент максимальный (стоит в конце отсортированного массива).

Можно предположить, что требуется найти элемент, который повторяется больше $n/2$ раз, но никакой из них не реализован корректно, везде есть свои недостатки.

Приведем примеры:

1. *Результаты работы алгоритмов совпадают.* Пусть $A = \{1, 2, 1, 3, 1, 1\}$. Ожидаем результат: 1.

   Приведем частичную трассировку для каждого из алгоритма:

   `algorithm1`:

c	ind	i	c_1	j
0	-1			
0	-1	0	1	0
0	-1	0	1	1
...	...	...	...	...
4	0	0	4	6
4	0	1	0	0
...	...	...	...	...
4	0	1	1	6
...	...	...	...	...
4	0	5	4	6
4	0	6		

   `algorithm2`:

c	ind	i
1	0	
1	1	1
1	2	2
1	3	3
1	4	4
2	4	5
2	4	6

   `algorithm3`:

c	i
1	
2	1
3	2
4	3

   Действительно, результат работы каждого алгоритма — 1.

2. *Результаты работы алгоритмов отличаются.* Пусть $A = \{1, 3\}$. По сути, должен вернуться «нулевой» результат, так как искомого элемента в массиве нет.

   `algorithm1`:

c	ind	i	c_1	j
0	-1			
0	-1	0	1	0
0	-1	0	1	1
1	0	0	1	2
1	0	1	0	0
1	0	1	1	1
1	0	1	1	2
1	0	2		

   `return` не сработает

   `algorithm2`:

c	ind	i
1	0	
1	1	1
1	1	2

   Вернется 3 — некорректный ответ

   `algorithm3`:

c	i
1	
1	1
1	2

   `return` не сработает

Пункт 2. Пусть $T_1(n)$, $T_2(n)$, $T_3(n)$ — функции временной сложности для алгоритмов `algorithm1`, `algorithm2` и `algorithm3` соответственно.

Оценим приблизительно эти функции.

1. В `algorithm1` используется двойной цикл (вложенный), каждый из которых проходит от 0 до n, следовательно, $T_1(n) = O(n^2)$.

2. В `algorithm2` используется один цикл от 1 до n, следовательно, $T_2(n) = O(n)$.

3. В `algorithm2` используется сортировка и один цикл от 1 до n. Будем считать, что используется одна из сортировок, работающих за $O(n \log n)$. Тогда, $T_3(n) = O(n \log n) + O(n) = O(n \log n)$.

Пункт 3. Конечно, можно придираться к тому, что некоторые функции не возвращают значения, но будем считать, что там просто возвращается `Null`, прямо как на языке `Python`.

1. В алгоритме `algorithm2` можно определенно не хватает существенной части. После выполненного цикла нельзя утверждать, что элемент A_{ind} является искомым, можно лишь считать, что либо результат не определен (`Null` подобно `algorithm1`), либо это действительно верный ответ. Другими словами, элемент A_{ind} — единственный претендент на ответ. Оставшуюся проверку можно провести вновь за линейное время, например, так:

```
cur = 0
for i = 0 to n
  if A[i] = A[ind]
        cur = cur + 1
if cur > n / 2
  return cur
```

Рис. 2: `algorithm2` improve

Тогда, чтобы получить работающий код, нужно убрать 14 строчку и добавить вышеуказанную проверку.

2. В алгоритме `algorithm3` нужно дописать следующие строки в конце приведенного, чтобы обрабатывать те повторяющиеся элементы, которые стоят в хвосте отсортированного массива.

```
if c > n / 2
  return A[n - 1]
```

Рис. 3: `algorithm3` improve

Пункт 4. Так как в `algorithm1` никакие улучшения не предлагались, оценим только функции временной сложности улучшенных алгоритмов `algorithm2` и `algorithm3`: $T_2'(n)$ и $T_3'(n)$ соответственно.

$$T_2'(n) = T_2(n) + O(n) = O(n) + O(n) = O(n)$$

$$T_3'(n) = T_3(n) + O(1) = O(n \log n) + O(1) = O(n \log n)$$