АНАЛИЗ НА РЕШЕНИЕТО НА ЗАДАЧА ПРОИЗВЕДЕНИЕ

1. Наивният подход е със сложност $O(n^3)$, където n е броят на елементите на дадената редица. Реализира се с тройно вложен цикъл, с който се обхождат всички елементи на редицата с индекси $i < j < k$ и за тези от тях, за които е изпълнено, че $a_i < a_j < a_k$ търсим най-голямата стойност на произведението $a_i \cdot a_j \cdot a_k$. Този подход ще вземе около 30% от точките за задачата.

2. Подход със сложност $O(n^2)$. Елементите на дадената редица записваме в масива a[i] (i = 1, 2, …, n) и пресмятане стойностите на два допълнителни масива a1[i] и a2[i], където елементът a1[i – 1] има стойност, по-малка от a[i], такава че да е най-голяма в отреза a[1], a[2], …, a[i – 1] и a2[i + 1] има стойност, по-голяма от a[i], такава че да е най-голяма в отреза a[i + 1], a[i + 2], …, a[n]. Тогава с цикъл по i = 2, …, n – 1, намираме най-голямата стойност на произведението a1[i – 1]*a[i]*a2[i + 1]. Този подход ще вземе около 50% от точките за задачата.

Пресмятането на a1[i] може да се извърши лесно за $O(n^2)$ операции, а пресмятането на a2[i] може да го реализираме даже за O(n) операции.

3. За да намалим сложността от $O(n^2)$ на O(n log n), трябва да усъвършенстваме пресмятането на a1[i]. При търсенето на най-голяма стойност (по-малка от a[i]) в отреза a[1], a[2], …, a[i-1] може да се използва двоично дърво за търсене. Подреждаме a[1], a[2], …, a[i-1] в такова дърво и търсим стойност a[i] и най-близката по-малка от нея. На всяка стъпка трябва да вмъкваме нов елемент в това дърво и за да избегнем изграждане, трябва да правим балансиране. Програмирането на класическия метод на балансирани AVL-дървета изисква писане на дълъг код и затова алтернативно решение, което е приложено в авторската програма е използване на map от STL и функциите insert и lower_bound.

Автор: Емил Келеведжиев