Задача A. Масив і медіани підмасивів

Автор задачі: **Антон Тригуб**
Задачу підготував: **Роман Білий**
Розбір написав: **Андрій Филипюк**

В задачі необхідно було визначити, чи можливо розділити масив парної довжини на певну кількість підмасивів непарної довжини так, щоб їхні медіани були рівні.

Теза 1. Ми розбиваємо наш масив виключно на парну кількість підмасивів.

Доведення. Нехай ми розбили на непарну кількість підмасивів. Так як кожен підмасив має непарну довжину, маємо що сумарна довжина масиву це сума непарної кількості непарних чисел, з чого випливає, що довжина масиву — непарне число, що протирічить умові.

Блок 1

Додаткова умова: $n = 2$

В будь-якому разі, ми розбиваємо наш масив рівно на 2 підмасиви, що складаються з одного елементу. Медіана масиву з одного елементу — власне цей елемент. Відповідно, відповідь `Yes`, якщо ці два масиви рівні (тобто, перший елемент оригінального масиву рівний другому елементові того ж масиву), та `No` в іншому випадку.

Теза 2. Візьмемо довільний підмасив довжини m, нехай його медіана є рівна x, тоді, і тільки тоді, у нас виконуються одночасно дві нерівності:

- `[` кількість елементів строго *менших* за x `]` $\leqslant \frac{m-1}{2}$
- `[` кількість елементів строго *більших* за x `]` $\leqslant \frac{m-1}{2}$

Доведення. Це витікає напряму з визначення медіани.

Наслідок тези 2. Якщо у підмасиві непарної довжини m існує елемент x, який зустрічається строго більше ніж $\frac{m-1}{2}$ раз, то ме діана даного підмасиву є рівна x.

Блок 2

Додаткова умова: $1 \leqslant a_i \leqslant 2$, для $1 \leqslant i \leqslant n$.

Нехай c_1 та c_2 — кількість одиничок та двійок в масиві відповідно.

Розглянемо два випадки:

$c_1 \neq c_2$*:* Без втрати загальності, скажемо, що $c_1 > c_2$. Зауважимо, що в такому разі $c_1 - c_2 \geqslant 2$. (Інакше, тобто, якщо б $c_1 - c_2 = 1$, це означало б, що $c_1 = c_2 + 1$, тобто $n = c_1 + c_2 = 2 \cdot c_2 + 1$, що не може бути правдою, бо n — парне.) Візьмемо мінімальний префікс непарної довжини, такий, що кількість одиничок на ньому рівно на 1 більше ніж кількість двійок на ньому (очевидно, що такий завжди існує). Відновідно, з того, що $c_1 - c_2 \geqslant 2$, очевидно, що серед елементів поза цим префіксом одиничок теж буде більше, ніж двійок. Тобто, ми фактично розбили масив на два підмасиви, на кожному з яких одиничок більше ніж двійок, тобто згідно Тези 2, ми маємо, що на обох масивах медіана є рівна 1. Випадок, коли у всьому масиві двійок більше ніж одиничок доказується аналогічно. Іншими словами, відповідь завжди `Yes`.

$c_1 = c_2$*:* Ми розбиваємо завжди на парну кількість підмасивів непарної довжини (згідно Тези 1). Теж ми знаємо, що у нас внаслідок того, що лише два можливих елементи, медіана кожного підвідрізку, це той елемент, який там зустрічається строго частіше ніж інший (наслідок Тези 2). Якщо ми візьмемо будь-яке розбиття (без втрати загальності, нехай це буде 1), то це означає, що у кожному підмасиві кількість 1 > кількості 2, тобто загалом $c_1 > c_2$ (бо сума нерівностей це теж нерівність), що є протиріччям. Тобто, такого розбиття не існує, і відповідь завжди `No`.

Теза 3. Якщо розбиття існує, то воно складається з 2-х підмасивів.

Доведення.

Частина 1. Візьмемо три довільні масиви з одинаковою медіаною x. Нехай розміри масивів, це m_1, m_2, m_3 відповідно. Розглянемо медіану об'єднання цих трьох масивів, та доведемо, що вона теж рівна x. Нехай c_1, c_2, c_3 це кількість елементів, що менші за x у відповідних масивах. Тоді, з Тези 2 маємо:

- $c_1 \leqslant \frac{m_1-1}{2}$
- $c_2 \leqslant \frac{m_2-1}{2}$
- $c_3 \leqslant \frac{m_3-1}{2}$

Якщо просумуємо нерівності, то матимемо $c_1 + c_2 + c_3 \leqslant \frac{m_1+m_2+m_3-3}{2} \leqslant \frac{m_1+m_2+m_3-1}{2}$. Що значить, що кількість елементів менше x у об'єднанні є не більше половини розміру об'єднання. Аналогічні операції можна провести відносно елементів більших x. З цього відповідно випливає, що x є медіаною об'єднання.

Частина 2. Відповідно, якщо ми маємо більше 2-х підмасивів, то ми постійно можемо зменшувати їх кількість на 2 шляхом об'єднання сусідніх підмасивів. Так, як кількість підмасивів має бути парна, то ми зупинимось на 2-х підмасивах.

Блок 3

Додаткова умова: $a_i \leqslant a_{i+1}$, для $1 \leqslant i < n$.

З Тези 3 ми знаємо те, що відповідь має складатися з двох підмасивів. Розглянемо довільний елемент x. Очевидно те, що x може бути медіаною обох підмасивів, якщо вони обидва містять x. Так як у нас масив неспадний, межа між підмасивами мусить проходити так, що в першому підмасиві у нас x це буде максимальний елемент, а в другому — мінімальний. Для того, щоб максимальний/мінімальний елемент якогось набору з m елементів був медіаною (де m — непарне), необхідно щоб він зустрічався хоча б $\frac{m+1}{2}$ разів (витікає з визначення). Нехай m_1, m_2 розміри підмасивів якогось розподілу масива, та $c_{x,1}, c_{x,2}$ — кількість елементів x відповідно у першому та другому підмасиві, тоді необхідно, щоб:

- $c_{x,1} \geqslant \frac{m_1+1}{2}$
- $c_{x,2} \geqslant \frac{m_2+1}{2}$

Що, в рамках даного блоку, буде виконуватись, якщо $c_{x,1} + c_{x,2} \geqslant \frac{m_1+m_2+2}{2} > \frac{m_1+m_2}{2}$. Тобто, достатньо буде перевірити, чи існує елемент, який зустрічається більше ніж $\frac{n}{2}$ раз.

Блок 4

Додаткова умова: $1 \leqslant a_i \leqslant 3$ для $1 \leqslant i \leqslant n$ та кожне значення зустрічається в a не більше ніж $\frac{n}{2}$ разів.

Якщо у нас існує лише два унікальних значення, то блок вирішується ідентично блоку 2.

В іншому випадку, так як 1 та 3 є мінімальним та відповідно максимальним елементом, для того, щоб вони були медіанами підмасивів, вони мають зустрічатись хоча б $\frac{m}{2}$ разів (де m — розмір підмасива) там, а якщо так буде, то 1 чи 3 зустрічатимуться більше ніж $\frac{n}{2}$ разів (див. блок 3). Внаслідок чого, медіаною може бути лише число 2.

Тепер залишилось перевірити, чи таке розбиття на підмасиви існує. Спершу опишемо умову того, що число 2 є медіаною якогось підмасиву розміру m (згідно тези 2):

- [кількість 1] $\leqslant \frac{m-1}{2}$
- [кількість 3] $\leqslant \frac{m-1}{2}$

Тепер, коли ми проходимось по масиву зліва направо зберігаючи кількість 1 та 3 на певному префіксі, ми можемо бистро оприділити чи цей префікс є коректним підмасивом з медіаною 2. Залишилось перевірити чи суфікс елементів, що залишаються, теж є коректним підмасивом з медіаною 2. Для цього можемо перед початком алгоритму порахувати сумарну кількість 1 та 3, тоді на певному етапі кількість 1 та 3 на суфіксі елементів, що залишаються, можна просто порахувати як [кількість загальна] - [кількість на префіксі]. Таким чином ми зможемо теж бистро перевіряти чи суфікс елементів теж є підмасивом з медіаною 2. Під час реалізації, варто не забути про те, що нам підходять лише префікси непарного розміру.

Часова складність рішення: $O(n)$.

Блок 5

Додаткова умова: $n \leqslant 100$.

Просто перебираємо можливу медіану серед елементів масиву, та перевіряємо її за допомогою техніки динамічного програмування: нехай ми зафіксували якесь x — спільна медіана, тоді dp_i це значення `Так`/`Ні`, яке означає те, чи ми можемо розбити префікс масива довжиною i на будь-яку кількість підмасивів непарного розміру, так що у них всіх медіана є рівна x. Тоді, очевидно, що dp_i = `Так`, якщо i — непарне, а також:

- Медіана префіксу розміру i є рівна x, або
- Існує таке j, що dp_j = `Так` і підмасив від j до i елементу має медіану x.

Алгоритм швидкої перевірки використовується такий самий, як у блоці 4, тільки ми трактуємо:

- Числа менші за x, як 1,
- Числа рівні x, як 2,
- Числа більші за x, як 3.

Часова складність рішення: $O(n^3)$.

Теза 4. Спільна медіана повинна бути така сама, як медіана усього масиву.

Доведення. Аналогічне доведенню Тези 3, Частини 1.

Блок 6

Додаткова умова: $n \leqslant 2000$.

Рішення 1

Помітимо, що коли ми зафіксували якесь число x і хочемо перевірити чи воно може бути спільною медіаною, у нас буквально перед очима Підзадача 4, лише трактуємо:

- Числа менші за x, як 1,
- Числа рівні x, як 2,
- Числа більші за x, як 3.

Часова складність рішення: $O(n^2)$.

Рішення 2

Використаємо Тезу 4, зафіксуємо медіану та використаємо підхід динамічного програмування, аналогічний Підзадачі 5.

Часова складність рішення: $O(n^2)$.

Повне рішення.

Використаємо Тезу 4, зафіксуємо медіану (нехай це буде x), тоді ми маємо задачу, аналогічну до Підзадачі 4, лише трактуємо:

- Числа менші за x, як 1,
- Числа рівні x, як 2,
- Числа більші за x, як 3.

Задача B. Масив символів і майже паліндроми

Автор задачі: **Антон Тригуб**
Задачу підготували: **Владислав Денисюк і Владислав Заводник**
Розбір написав: **Владислав Денисюк**

Блок 1

Перше спостереження — це те, що рядок є майже паліндромом тоді і тільки тоді, коли кількість символів, що зустрічаються непарну кількість разів у цьому рядку не більша 1. Якщо є тільки один такий символ - то ми можемо поставити його в центр рядку і зробити ліву і праву половину зеркальними, якщо таких символів 0 - то просто зробити зеркальні ліву і праву половини. Можна показати, що за допомогою цих двох варіантів можна сформувати який-завгодно паліндром, а отже якщо таких символів більше 1 — то це не майже паліндром. Тоді у данному блоці є чотири випадки: 1) кількість усіх символів парна, 2) кількість `a` — непарна, 3) кількість `b` — непарна, 4) кількість усіх символів - непарна. У випадку 4 - відповідь - весь рядок, у випадках 1,2,3 - потрібно видалити якийсь префікс і якийсь суфікс, щоб кількість `a` і `b` стала непарною, для цього переберемо усі пари префіксів і суфіксів не довших за 4, якщо префікс або суфікс довший за 4, то при видаленні перших чотирьох його символів - парність кожного символу в ньому - не зміниться.

Блок 2

Блок створено для дуже неоптимальних реалізацій підтримки парності входжень конкреного символу в блоках `3` і `4`.

Блок 3

Будемо розв'язувати задачу подібно до блоку `1`, але переберемо усі пари початку і кінця, і для кожної перевіримо кількість символів з непарною кількістю входжень.

Блок 4

Оптимізуємо рішення блоку `2`: перебиратимемо початок, і потім будемо одночасно рухати кінець вправо і додавати новий символ перераховуючи кількість непарних. Ми можемо робити це підтримуючи кількість кожного символа.

Блок 5

Порахуємо на префіксі і на суфіксі найближчу позицію де на префіксі або суфіксі відповідно парність кількості `a` дорівнює x, а `b` — y. Всього є 4 пари (x, y), бо кожне з них приймає лише два значення. Будемо перебирати комбінації парностей на префіксі і суффіксі і так порахуємо відповідь. Складність: $O(n + q)$.

Блок 6

Якщо рядок не є майже паліндромом — виведемо всю його довжину. Інакше — ми можемо видалити або лівий або правий або лівий і правий кінці і рядок перестане бути майже паліндромом. Це так тому, що якщо обидва з них входять парну кількість раз - то після видалення - обидва будуть входити непарну кількість раз і це не буде майже паліндромом. А якщо хоча б одне з них входить непарну кількість раз — то потрібно видалити те, що входить парну кількість раз і ми знову матимемо два символи які входять непарну кількість раз.

Блок 7

Розв'язок схожий до блоку `6`. Якщо символи на кінцях відрізку рівні, то при видалені символу з одного з кінців вони не будуть рівні і ми отримаємо блок `6`. Тому якщо у блоці `6` ми перебирали усі суфікси і префікси не довше одного, то тут — не довше за 2.

Блок 9

Оскільки довжина рядку непарна - один з символів точно входить в нього непарну кількість разів. Нам потрібно видалити хоча б один інший символ і парну кількість данного. Для цього знайдемо найближчий до одного з кінців відрізка символ, що зустрічається парну кількість раз. Це кінець відрізка якщо хоча б один з них не є тим символом, що зустрічається непарну кількість раз, або якщо обидва кінця відрізка — це непарний символ, ми можемо до обробки всіх запитів порахувати на префіксі і на суфіксі останню позицію де стояв інший символ, тоді ми зможемо знайти найближчий до кінців відрізка парний символ за дві перевірки. Якщо для того щоб видалити його потрібно буде видалити непарний символ непарну кількість раз, просто видалимо його ще раз з іншого кінця відрізка. Складність: $O(n + q)$.

Блок 10

По черзі видалимо початок і кінець звівши задачу до блоку 9 і візьмемо максимальну відповідь.

Задача C. Масив і знову додавання

Автор задачі: **Valerio Stancanelli**
Задачу підготував: **Tommaso Dossi**
Розбір написав: **Valerio Stancanelli**

Блок 1

$n \leqslant 100$. Потрібно заповнити позиції $[2, 100]$ одиницями, потім n разів додати $a_2 = 1$ до a_1.

Кількість операцій: $98 + n \leqslant 198 \leqslant 300$.

Блок 2

$n = k^2, k \leqslant 100$. Потрібно заповнити позиції $[3, 100]$ одиницями, потім k разів додати $a_3 = 1$ до a_2, потім k разів додати $a_2 = k$ до a_1.

Кількість операцій: $97 + k + k \leqslant 297 \leqslant 300$.

Блок 3

$n = 2^k - 1$. Спочатку заповнимо позиції $[1, 100]$ одиницями. Мета: $a_1 = 2^k - 1$.

$>$ Спостереження: $2^k - 1 = 2(2^{k-1} - 1) + 1$.

$>$ Нова мета: $a_i = 2^{k-i+1} - 1$.

Отже, коли ви отримаєте правильне значення a_i, ви можете додати a_i двічі до a_{i-1}, щоб отримати правильне значення a_{i-1}.

Приклад послідовності операцій для $n = 31$:

```
 1   1   1   1   1   1   1   ...
 1   1   1   3   1   1   1   ...
 1   1   7   3   1   1   1   ...
 1  15   7   3   1   1   1   ...
31  15   7   3   1   1   1   ...
```

Кількість операцій: $99 + 2k \leqslant 99 + 2 \cdot 59 = 217 \leqslant 300$.

Блок 4

n — число Фібоначчі. Використовуючи основну властивість чисел Фібоначчі $f_{i+2} = f_{i+1} + f_i$, можемо перетворити послідовність $\ldots, 0, f_{i+1}, f_i, \ldots$ на $\ldots, f_{i+3}, f_{i+2}, f_i$ за 3 операції:

```
...          0  f[i + 1]  f[i]  ...
...  f[i + 1]  f[i + 1]  f[i]  ...
...  f[i + 1]  f[i + 2]  f[i]  ...
...  f[i + 3]  f[i + 2]  f[i]  ...
```

Використовуючи дані 3 операції необхідну кількість разів, можемо отримати будь-яке число Фібоначчі.

Приклад послідовності операцій для $n = 55$:

```
 0   0   0   0   1   1   ...
 0   0   0   1   1   1   ...
 0   0   0   1   2   1   ...
 0   0   0   3   2   1   ...
 0   0   3   3   2   1   ...
 0   0   3   5   2   1   ...
 0   0   8   5   2   1   ...
 0   8   8   5   2   1   ...
 0   8  13   5   2   1   ...
 0  21  13   5   2   1   ...
21  21  13   5   2   1   ...
21  34  13   5   2   1   ...
55  21  13   5   2   1   ...
```

Кількість операцій: менше $99 + \frac{3}{2} \cdot \log_{\varphi}(10^{18}) \leqslant 300$.

Блок 5, 1927 операцій

Представимо n у двійковому записі. Знайдемо таку послідовність операцій, під час якої a_2 буде приймати значення $1, 2, \ldots, 2^{59}$ в певні моменти часи.

Заповнимо позиції $[2, 100]$ одиницями. Тепер для кожного j від 1 до 59, виконаємо операції $(j+1), j, \ldots, 2$, щоб виконувалися рівності $a_{j+1} = 2^1, a_j = 2^2, \ldots, a_2 = 2^j$.

Якщо двійковий запис числа n містить i-й біт, то виконаємо операцію `a[1] += a[2]`, коли $a_2 = 2^i$.

Приклад послідовності операцій для $n = 42 = 2 + 8 + 32$:

```
 0   1   1   1   1   1   1   ...
 2   2   1   1   1   1   1   ...
 2   4   2   1   1   1   1   ...
10   8   4   2   1   1   1   ...
10  16   8   4   2   1   1   ...
42  32  16   8   4   2   1   ...
```

Кількість операцій: $98 + \frac{59 \cdot 60}{2} + 59 = 1927$.

Блок 5, 1086 операцій

У попередньому алгоритмі отримання великих степеней двійки потребує великої кількості операцій. Виявляється, що є кращий спосіб створення степенів 2. У другій половині процесу ми можемо створювати степені 4 замість того, щоб створювати степені 2.

Приклад послідовності операцій для $n = 426 = 2 + 8 + 32 + 128 + 256$:

```
  0   1   1   1   1   1   1   ...
  2   2   1   1   1   1   1   ...
  2   4   2   1   1   1   1   ...
 10   8   4   2   1   1   1   ...
 10  16   8   4   2   1   1   ...

 42  32  16   8   4   1   1   ...
 42  64  32  16   4   1   1   ...
170 128  64  16   4   1   1   ...
426 256  64  16   4   1   1   ...
```

Зауважте, що нам потрібна одна додаткова операція для кожного рядка у другій половині процесу (оскільки нам потрібно додати 4^k двічі до $2 \cdot 4^k$, щоб отримати 4^{k+1}).

Кількість операцій: $98 + \frac{30 \cdot 31}{2} + \frac{29 \cdot 30}{2} + 29 + 59 = 1086$.

Блок 5, 929 операцій

Краще використовувати систему числення з основою 3.

Цього разу нам потрібні 2 операції для того, щоб отримати 3^{k+1}, починаючи з 3^k. Також необхідно виконати 4 додаткові операції для кожного рядка у другій половині процесу (оскільки нам потрібно додати 9^k шість разів до $3 \cdot 9^k$, щоб отримати 9^{k+1}). Проте система числення з основою 3 є більш ефективною, ніж система числення з основою 2, оскільки $log_3(10^{18}) \approx 38$ є «набагато» меншим за $log_2(10^{18}) \approx 60$.

Кількість операцій: $98 + 2 \cdot \frac{19 \cdot 20}{2} + 2 \cdot \frac{18 \cdot 19}{2} + 4 \cdot 18 + 37 = 929$.

Блок 5, 276 операцій

Попередні рішення використовують $O(\log^2 n)$ операцій. Чи можемо ми використати $O(\log n)$ операцій?

Ми знаємо, як отримати значення $2x$ починаючи з x:

```
 0   x   ?   ?   ?   ?   ?   ...
 x   x   ?   ?   ?   ?   ?   ...
2x   x   ?   ?   ?   ?   ?   ...
```

Ми також хочемо мати можливість отримати $(2x+1)$ починаючи з x. Одним з можливих способів є додавання 1 до всіх позицій спочатку:

```
      1   1   1   1   1   1   1   ...
      1   x   ?   ?   ?   ?   ?   ...
  x + 1   x   ?   ?   ?   ?   ?   ...
 2x + 1   x   ?   ?   ?   ?   ?   ...
```

Але тепер ми не можемо отримати $2x$ з x. Виявляється, що замість цього ми можемо отримати $2x + 2$.

```
      1   1   1   1   1   1   1   ...
      2   1   1   1   1   1   1   ...
      2   x   ?   ?   ?   ?   ?   ...
  x + 2   x   ?   ?   ?   ?   ?   ...
 2x + 2   x   ?   ?   ?   ?   ?   ...
```

Цього достатньо, щоб обчислити цільові значення a_i:

1. $a_1 = n$;
2. $a_i = \frac{a_{i-1}-1}{2}$ якщо a_{i-1} непарне;
3. $a_i = \frac{a_{i-1}-2}{2}$ якщо a_{i-1} парне.

Приклад послідовності операцій у зворотньому порядку для $n = 150$:

```
150  74  36  17   8   3   1   ...
  2  74  36  17   8   3   1   ...
  2   2  36  17   8   3   1   ...
  2   2   2  17   8   3   1   ...
  2   2   2   1   8   3   1   ...
  2   2   2   1   2   3   1   ...
  2   2   2   1   2   1   1   ...
  1   1   1   1   1   1   1   ...
```

Таким чином, нам просто потрібно 99 операцій, щоб заповнити масив одиницями, 59 операцій, щоб поставити 2 там, де це необхідно, а потім $2 \cdot 59$ операцій, щоб отримати цільові значення a_i.

Кількість операцій: $99 + 59 + 2 \cdot 59 = 276 \leqslant 300$.

Задача D. Масив і XOR

Автор задачі: **Денисов Костянтин**
Задачу підготував: **Денисов Костянтин**
Розбір написав: **Денисов Костянтин**

Блок 1

Напишемо повний перебір. Порахуємо $f_i(x)$ для кожного x від 0 до 2^m-1 проходимо по елементам масиву з відповідного відрізка запиту.

Блок 2

Будемо поступово будувати відповідь на запит від більших бітів до менших. Дізнаємося чи буде $m-1$ біт у відповіді. Поділимо наш вхідний массив a на 2 інші масиви b_0 та b_1, де у b_0 знаходяться всі числа вхідного масиву, у яких $m-1$ біт дорівнює 0, аналогічно b_1 містить всі числа вхідного масиву, у яких $m-1$ біт дорівнює 1. Тоді очевидно, що якщо хоча б один з масивів b_0, b_1 є порожнім, то $m-1$ біт буде дорівнювати 1 у відповіді. Розглянемо випадок, коли обидва масива непорожні. Розглянемо x, у яких $m-1$ біт дорівнює 0, тоді $f(x) = \min_{y\in a} y \oplus x = \min_{y\in b_0} y \oplus x$, оскільки для чисел з масиву b_1 число $y \oplus x$ буде мати $m-1$ біт, отже, на ньому мінімум не буде досягатися. Аналогічно, якщо розглянути x, у яких $m-1$ біт дорівнює 1, то отримаємо $f(x) = \min_{y\in a} y \oplus x = \min_{y\in b_1} y \oplus x$. Отже, фактично ми розділили задачу на дві незалежні задачі (в залежності від $m-1$ біта в x), в яких відрізняються масиви на яких шукаємо відповідь.

З цих міркувань маємо наступний алгоритм (псевдокод), що поступово знаходить відповідь.

```
solve (m, a) {
    if (m == 0) return 0
    build b[0], b[1] from a
    if (b[0] is empty)
        return solve(m - 1, b[1]) ^ (1 << (m - 1));
    if (b[1] is empty)
        return solve(m - 1, b[0]) ^ (1 << (m - 1));
    return max(solve(m - 1, b[0]), solve(m - 1, b[1]));
}
```

Блок 4

Помітимо, що рекурсивне розділення масиву на b_0 та b_1 аналогічне побудові двійково бору на числах масиву. У вершинах бору можемо зберігати кількість чисел, що проходить через поточну вершину бора та підтримувати відповідь для цього піддерева бора. Отже, знаючи значення у синах вершини можемо просто порахувати значення у батьківській вершині бора. Таким чином ми можемо додадавати та видаляти числа з масиву при цьому оновлюючи всі значення у вершинах за $O(m)$. Таким чином можемо використати алгоритм Мо. Складність рішення $O(n \cdot \sqrt{q} \cdot m)$.

Блок 3

Побудуємо бор на всьому вхідному масиві. Навчимося відповідати на запит на відрізку $[l, r]$. Для цього будемо ітеруватися вглиб бора поки всі числа з поточного відрізку проходять через поточну вершину бора. Спочатку ми знаходимся у корені бора. Далі розглянемо два варіанти:

1. Якщо всі числа з поточного відрізка запиту проходять через один з синів поточної вершини, то перейдемо до цього сина і продовжимо цю ж процедуру.

2. В іншому випадку числа з поточного відрізка знаходяться у двох синах вершини. Можна помітити, що якщо подивитися на взагалі всі числа масиву, що проходять через якогось сина поточної вершини, то числа з відрізка запиту будуть утоворювати якийсь його префікс або суфікс. Тоді можна зберегти відповіді для кожного префікса (суфікса) у кожній вершині бору поки його будуємо.

Блок 5

Цей блок було створено для неоптимальних рішень, які використовують дерево відрізків або дерева фенвіка у заключній частині рішення з блоку 6.

Блок 6

Можемо помітити, що, неформально некажучи, всі попередні алгоритми обирають якийсь шлях у борі і «максимізують» значення на цьому шляху. А саме, якщо ми пішли на шляху до якогось сина вершини у борі, якій відповідає k-ий біт, то значення на цьому шляху буде мати k-ий біт рівний 1 тоді і тільки тоді, коли немає другого сина у поточній вершині.

Тоді оберемо якийсь шлях у нашому борі, розглянемо як інші шляхи впливають на його значення. Оберемо якийсь інший шлях. Нехай вони перетинаються по k найбільших бітах. Тоді значення на першому шляху буде мати $k+1$ найбільший біт рівний 0. Отже, нас цікавить тільки довжина перетину двох шляхів. Є $m+1$ різних довжин перетину. Шлях відповідає певному числа з масива a. Нехай воно має індекс j. Тоді для кожної можливої довжини len перетину шляхів знайдемо найбільший індекс l_len, що $l_len < j$ та шлях числа a_{l_len} має довжину перетину зі шляхом a_j рівну len. Аналогічно знайдемо такі найменші значення r_len, що $r_len > j$. Інші шляхи нас не цікавлять, бо вони вже ніяк не змінять значення на шляху для числа a_j.

Тоді для кожного шляху, що відповідає числу a_j маємо не більше m шляхів зліва, що йому змінять значення на цьому шляху і аналогічно не більше m шляхів справа. Тоді маємо, що є не більше m^2 різних відрізків, де поточний шлях приймає різні значення. Тоді кожного шляху будемо мати не більше m^2 таких «оновлень»: $[c_i, d_i, b_i]$, що означає, що значення цього шляху на відрізку $[c_i, d_i]$ дорівнює b_i. Іншими словами, якщо в нає є запит, який міститься у відрізку $[c_i, d_i]$, то в нього буде значення принаймі b_i.

Тобто зараз ми маємо таку задачу: маємо не більще $n \cdot m^2$ $[c_i, d_i]$ відрізків кожен зі значенням b_i і для кожного запиту нам потрібно дізнатися максимальне значення серед відрізків, які повністю його містять.

Цю задачу будемо вирішувати в офлайні, тобто спершу зчитаємо всі запити і будемо відповідати на них у довільному порядку, а не в тому, що їх задають.

Заведемо новий масив e довжиною n, початково $e_i = 0$. Відсортуємо наші запити $[l_i, r_i]$ за зростанням l_i. Далі розглядаємо наші запити у порядку збільшення l_i. Розглянемо запит $[l_i, r_i]$. Тоді для всіх відрізків j, що $c_j \leqslant l_i$ оновимо $e_{d_j} := max(e_{d_j}, b_j)$. Тоді відповіддю на запит $[l, r]$ буде максимальне значення на відрізку $[r_i, n]$. Оскільки запитів оновлення значно більше ніж запитів знаходження максимуму, то будемо підтримувати це за допомогою sqrt-декомпозиції.

Отже, маємо рішення за $O(n \cdot m^2 + q \cdot \sqrt{n})$.