# Розбір задачі «Заборонена сума»

Спочатку вирішимо цю задачу для запитів де $l = 1$ та $r = n$. Відсортуємо елементи у порядку не спадання. Перший елемент повинен бути рівним 1, бо інакше ми не зможемо отримати 1 у вигляді суми будь-якого набору. Другий елемент не може бути більшим за 2, бо з першого ми можемо отримати лише 1. Тепер, використовуючи перші два елементи, ми можемо отримати будь-яке значення від 0 до $A_1 + A_2$. Звідси, третє значення у масиві не може перевищувати $A_1 + A_2 + 1$, бо тоді ми не зможемо отримати значення $A_1 + A_2 + 1$, і воно буде вважатись забороненим. Отже, знайдемо перший момент $i$, де $A_i$ більше за $\sum_1^{i-1} A_i + 1$. Це означає, що ми можемо отримати усі значення від 1 до $\sum_1^{i-1} A_i$, а число $\sum_1^{i-1} A_i + 1$ отримати не можемо, отже це значення і буде забороненою сумою.

Таке рішення буде потребувати $O(n)$ часу на запит. Можемо це пришвидшити. Давайте будемо зберігати значення $S$, що означає, що ми можемо отримати усі числа від 0 до $S$. Спочатку $S = 0$. Отже, тепер ми хочемо додати усі числа значення яких рівне 1, бо усіх до 1 ми можемо отримати. Порахуємо значення $P$ — сума усіх одиничок в масиві, і додамо це значення до $S$. Отже, тепер ми можемо отримати усі числа від 0 до $P$, і ми можемо додати до $S$ усі числа зі значенням, що більше за 1 та не більше за $P+1$. Позначимо їх суму знову змінною $P$. Додамо це число до $S$. І продовжуємо процес за наступним алгоритмом:

1. Будемо зберігати значення $S'$ (значення $S$ на попередньому етапі), та значення $S$ для поточного етапу.
2. Отже, до $S$ треба додати суму усіх елементів значення яких більше за $S'$ та не більше за $S$.
3. Якщо їх сума рівна 0, тобто жодного такого елементу не існує, отже значення $S + 1$ буде забороненою сумою. А інакше, кажемо що $S' = S$, а до $S$ додаємо цю суму.

Очевидно, що якщо ми знайшли хоча б один елемент на деякому етапі, тоді наше нове значення $S$ буде хоча б у два рази більше ніж на попередньому. Отже, таких ітерацій буде не більше ніж $log(max(a_i))$.

Для того щоб вирішити дану задачу для будь-яких запитів, треба зберігати персистентне дерево відрізків, щоб знаходити суму усіх чисел, що лежать на проміжку від $l$ до $r$ та мають значення від $x$ до $y$.

# Розбір задачі «Запити на xor»

Зробимо персистентне дерево відрізків, що для кожного префіксу масиву буде зберігати кількість входжень кожного числа. Для цього ініціалізуємо наше дерево розміром $C = 2^{19}$ ($500000 \leqslant 2^{19}$). Воно буде повним бінарним деревом глибини 19.

- Тип 0 — скопіювати останню версію дерева та додати до неї наше число.
- Тип 2 — перейти до версії, що відповідає поточному масиву без останніх $k$ елементів.
- Тип 3 — у дереві відрізків будемо зберігати кількість разів скільки зустрічається кожне число на відповідному префіксі. Тоді відповідь можна отримати як різницю цієї кількості для префіксів $r$ та $l - 1$.
- Тип 4 — можемо застосувати стандартний пошук $k$-го найменшого елементу у дереві відрізків, де кількість елементів на відрізку буде визначатись як різниця цієї кількості для префіксу $r$ та $l - 1$. Тобто ми будемо одночасно спускатись по двох деревах, та дізнаватись так кількість потрібних чисел на відрізку $l \dots r$.
- Тип 5 — розглянемо спочатку як вирішувати задачу коли запит відповідає усьому масиву. Очевидно, що нам вигідно, щоб наші числа відрізнялись у як можна старших бітах. Будемо шукати наше число спуском по дереву. Спочатку ми стоїмо у корені. Лівий син відповідає усім числам що не мають біт із номером 19, а правий син навпаки (бо наше дерево повне бінарне). Подивимось, чи є таке число, що із даним нам буде давати ксор, що містить біт із номером

19. Для цього просто треба перевірити що певний син містить хоча б один елемент. Якщо таке число є, перейдемо до відповідного сину, інакше перейдемо до протилежного. Таким жадним рішенням ми можемо знайти відповідь. Щоб дізнатись відповідь для відрізка застосуємо той самий прийом, що ми застосовували у попередніх запитах — будемо віднімати від одного дерева значення іншого.

# Розбір задачі «Соня та функція»

Будемо застосовувати бінарний пошук по відповіді. Нам треба вміти рахувати кількість відрізків значення функції $f$ для яких, не більше за $W$.

Замінимо наш масив на масив префіксних ксорів. Тобто значення $B_i$ буде рівне $A_1$ $xor$ $A_2$ $xor$ $\dots$ $xor$ $A_i$.

Спочатку побудуємо персистентне дерево відрізків, що буде зберігати кількість входжень кожного числа $B_i$ на кожному префіксі, за прикладом, як ми його будували для попередньої задачі. Тепер, застосовуючи дане дерево, ми можемо за $log(max(a_i))$ знаходити кількість чисел на відрізку $l \dots r$, ксор яких із числом $X$ не буде перевищувати число $W$. Для цього будемо спускатись по нашому дереву, та припустимо що поточна вершина відповідає найбільшому спільному префіксу двійкового представлення числа $W$ та ксору числа $X$ із певним іншим числом (кількість якого ми хочемо підрахувати). Тоді, розглянемо можливі варіанти, щоб обрати наступний біт, та якщо новий префікс вже буде менше за відповідний префікс числа $W$, додамо до відповіді розмір піддерева, що відповідає за цей префікс. Далі перейдемо до сина що відповідає наступному префіксу.

Як знайти кількість відрізків, значення $f$ на яких не більше за $W$:

1. Розглянемо функцію $solve(l, r)$, що буде знаходити кількість відрізків $(ll, rr)$, таких, що $l \leqslant ll \leqslant rr \leqslant r$ та $f(ll, rr) \leqslant w$.
2. Знайдемо позицію $pos$ найменшого числа на відрізку $l \dots r$.
3. Будемо шукати усі потрібні відрізки, що ще й проходять через цю позицію. Тоді ми зможемо зафіксувати значення $MIN(ll, rr) = A_{pos}$ для нашої функції $f$.
4. Припустимо, що $pos - l \leqslant r - pos$. Тоді переберемо значення $ll$ $(l \leqslant ll \leqslant pos)$. Знаючи $B_{ll-1}$, нам потрібно знайти кількість таких $rr$, що $pos \leqslant rr \leqslant r$ та $B_{ll-1}$ $xor$ $B_{rr} \leqslant W/MIN(ll, rr)$. Права частина у нас зафіксована та є константою, а кількість таких чисел $B_{rr}$ ми вже вміємо шукати. Випадок коли $r - pos \leqslant pos - l$ аналогічний, лише треба перебирати позиції справа від $pos$.
5. Викличемо $solve(l, pos - 1)$ та $solve(pos + 1, r)$, щоб знайти відрізки, що ми не врахували, оскільки вони не проходять через позицію $pos$.

Асимптотика рішення $O(n \cdot log^3(n))$.

# Розбір задачі «Максимальний рядок»

Побудуємо над рядком $S_0$ дерево відрізків, кожна вершина якого буде зберігати наступну інформацію про відповідний їй відрізок:

- Хеш рядку на даному відрізку.
- Хеш для інвертованого рядка на даному відрізку.
- Обіцянка — чи треба інвертувати інформацію для синів даної вершини.

Усю інформацію для вершини можна легко отримувати через інформацію про її синів та через обіцянки з вершини-батька.

Зробимо це дерево відрізків персистентним. Тепер ми можемо легко оброблювати усі зміни, та зберігати одночасно усі рядки.

Щоб порівняти два рядки лексикографічно, можна знайти першу позицію де вони відрізняються, та порівняти лише пару символів. Для цього будемо робити спуск по дереву:

- Якщо вершина лист — порівнюємо за значенням обіцянки. Якщо треба інвертувати значення синів, це значить що на поточній позиції стоїть одиничка, інакше нулик.
- Якщо ліві частини рядків відрізняються (це можна дізнатись за допомогою хешів які ми зберігаємо), переходимо до лівого сина.
- Інакше переходимо до правого сина.
- Продовжуємо спуск поки не дійдемо до листа.

# Розбір задачі «Stack»

Розв'язок задачі розглянуто у матеріалах лекції.

# Розбір задачі «Оголошення на паркан»

Перше що потрібне для розв'язку задачі, будемо використовувати бін пошук по відповіді. Припустимо що ми можемо отримати висоту оголошення рівну $h$. Поставимо одиничку на позиціях, де висота дошки паркану не менше за $h$, та нулик там де висота менше. Отже, якщо висота $h$ може бути відповіддю, тоді існує неперервний відрізок, що повністю лежить на відрізку $l \dots r$, має довжину рівну $w$, та складається цілком з одиничок.

Поставимо на кожну позицію нулик, та будемо додавати числа по-одному у порядку незростання значення (ставити на їх позиції одинички). Цей масив зробимо персистентним, тобто кожній зміні буде відповідати нова версія масиву. Для цього побудуємо на масиві дерево відрізків.

Тепер, щоб звернутись до певного моменту, коли були додані усі числа більше рівні за $h$, нам достатньо лише звернутись до певної версії дерева відрізків. Для того щоб перевірити чи існує потрібний нам відрізок, у кожній вершині дерева достатньо зберігати наступні значення:

- Максимальна довжина відрізку з одиничок, що лежить цілком на відрізку за який відповідає дана вершина.
- Максимальний префікс даного відрізку, що складається цілком з одиничок.
- Максимальний суфікс даного відрізку, що складається цілком з одиничок.

Асимптотика розв'язку $O(n \cdot log^2(n))$, оскільки бін пошук можна використовувати лише на числах, що зустрічаються у масиві.

# Розбір задачі «Соня та книжки»

Зробимо новий масив $B$ довжини $n$, усі елементи якого спочатку будуть рівні 0. Будемо йти справа наліво. Припустимо, що ми зараз стоїмо на позиції $i$, де знаходиться число $x = a_i$. Поставимо на позицію $i$ у масиві $B$ одиничку, та якщо ми вже зустрічали число $x$, поставимо 0 на попередній позиції, де ми його зустрічали. Це означає, що одиничка стоїть на позиціях, де вперше зустрічається якесь число, починаючи від позиції $i$. А значення суми на певному відрізку $i \dots r$ $(i \leqslant r \leqslant n)$ буде відповідати кількості різних чисел на цьому відрізку. Зробимо масив $B$ персистентним, та запам'ятаємо його версію для кожної позиції $i$.

Тепер, щоб вирішити нашу задачу при фіксованому значенні $k$, знайдемо перший префікс нашого масиву, де зустрічається рівно $k + 1$ різних чисел, нехай це позиція $p$. Тоді префікс $p - 1$ буде відповідати найбільшому префіксу який ми можемо взяти. Тож кожен раз будемо брати найбільший префікс, що містить не більше ніж $k$ різних чисел. Таке жадібне рішення є правильним.

Як знайти найбільший префікс, що містить не більше $k$ різних чисел? Візьмемо версію дерева відрізків, що відповідає за початок нашого масиву, та спуском по дереву знайдемо позицію $k + 1$-ї одиничи. Тоді попередня до неї позиція і буде шуканою.

Для фіксованого $k$ таке рішення буде працювати $O((n/k) \cdot log(n))$.

Тобто сумарно для усіх $k$ дане рішення буде працювати $O((n/1) \cdot log(n) + (n/2) \cdot log(n) + \ldots + (n/n) \cdot log(n))$, а як відомо, $n/1 + n/2 + n/3 + \ldots + n/n = O(n\ log\ n)$, тож асимптотика рішення $O(n\ log^2(n))$.

# Розбір задачі «Дужки»

Спочатку, пригадаємо алгоритм для пошуку кількості різних підрядків рядку $S$. Для цього побудуємо суфіксний масив $P$, та для кожної пари сусідніх суфіксів знайдемо найбільший спільний префікс (масив LCP). Тоді, будемо додавати усі різні підрядки, які є префіксами відповідного суфіксу, у порядку їх розташування у суфіксному масиві. Додамо до відповіді довжину першого суфіксу. Припустимо, що ми зараз розглядаємо суфікс на позиції $i$ ($i > 1$), та $w$ це довжина найбільшу спільного префіксу для суфіксів на позиції $i$ та $i - 1$. Тоді, перші $w$ префіксів поточного суфікса вже додані до відповіді, бо ми їх розглядали до цього моменту. А усі інші префікси ми зустрічаємо вперше, бо інакше $w$ було б більшим, бо усі суфікси розташовані у лексикографічному порядку. Звідси, до відповіді треба додати значення $N - P[i] + 1 - w$.

Перейдемо тепер до розв'язку оригінальної задачі. Він буде основаним на рішенні яке ми розглянули до цього, але тепер, нам треба додавати лише ті префікси, що є правильними дужковими послідовностями. Зробимо новий масив, у якому на позиціях де зустрічається відкрита дужка ми поставимо 1, а на позиціях де є закрита дужка, поставимо $-1$. Побудуємо для цього масиву масив часткових сум. Як перевірити що відрізок $l \dots r$ утворює правильну дужкову послідовність:

1. Різниця часткових сум у позиції $r$ та $l-1$ рівна нулю. Тобто усі символи зустрічаються однакову кількість разів.
2. Кожне значення часткової суми на відрізку від $l$ до $r$ не менше за значення часткової суми у позиції $l - 1$, бо інакше на відповідному префіксі даного відрізку кількість закритих дужок більша за кількість відкритих.

Тепер, коли ми хочемо дізнатись кількість правильних дужкових послідовностей, що починаються на позиції $l$, та закінчуються десь на позиції між $r$ та $n$, нам в першу чергу треба прибрати всі відрізки, де кількість закритих дужок більша за кількість відкритих. Для цього можна просто знайти першу позицію на відрізку $r \dots n$, де значення часткової суми менше за це значення на позиції $l - 1$. Це можна зробити звичайним деревом відрізків. Тепер нам потрібно дізнатись кількість відрізків, де кількість відкритих дужок рівна кількості закритих, тобто значення часткової суми рівне цьому значенню у позиції $l - 1$. Тобто треба сказати скільки разів якесь число зустрічається на певному відрізку. Це можна робити не використовуючи дерево відрізків.

# Розбір задачі «Соня та камінці»

Припустимо, що ми вирішували задачу $offline$ (спочатку зчитали усі запити та потім починаємо знаходити відповіді). Пройдемось по масиву зліва на право. Зараз ми стоїмо на позиції $i$. Будемо розглядати лише перші $i$ чисел з масиву. Нам знадобиться додатковий масив $b_i$. Для кожної країни, в останні $k$ позицій де зустрічались камінці з цієї країни (серед перших $i$ камінців), поставимо одинички, а в усі інші поставимо нулик. Тоді, відповідь для будь-якого відрізку $l \dots i$ ($1 \leqslant l \leqslant i$) буде рівна кількості одиничок на цьому відрізку, бо тоді ми зарахуємо кожне число не більше ніж $k$ разів, та нам завжди вигідно зараховувати останні позиції де воно зустрічається. При переході на наступну позицію, ми туди обов'язково поставимо одиничку, а також, можливо, замінимо якусь одну одиничку на нулик.

Щоб знаходити відповідь $online$, тобто одразу після зчитування запиту, зробимо наш масив персистентним та запам'ятаємо його версію для кожної позиції $i$. Тоді нам буде достатньо взяти суму на відрізку $l \dots r$ для версії, що відповідає моменту $i = r$.