Задача A. Ксоня та олімпіада

Автор та розробник: Святослав Бідзіля

Оскільки одна група завжди заходить в 0 секунду, а далі кожні w секунд, поки час не закінчиться. Нескладно помітити, що всього зайде $\lfloor t/w \rfloor + 1$ груп. Тут символами $\lfloor x \rfloor$ позначено округлення вниз — найбільше ціле число менше рівне за x. У кожній групі максимум m людей, а всього людей n (більше за це число людей зайти просто не може), звідки отримуємо пряму формулу для відповіді — $min(n, m \cdot (\lfloor t/w \rfloor + 1))$.

Задача B. Ксоня та алфавітне коло

Автор та розробник: Святослав Бідзіля

Спробуємо обмежити довжину відповіді. Оскільки в англійському алфавіті 26 букв, то відповідь не може бути довшою за 26. Знайдемо для кожного можливого початку довжину найдовшого алфавітного рядка, який з нього починається. Виберемо максимальний. Через обмеження довжини відповіді ми не будемо перевіряти кожний раз більше за 26 символів, отже рішення працює за $26 \cdot n$.

Для спрощення реалізації, можна занумерувати коло від 0 до $n - - - 1$, тоді наступний індекс після індексу i це $(i + 1)\ mod\ n$, де $x \mod y$ — залишок від ділення числа x на число y.

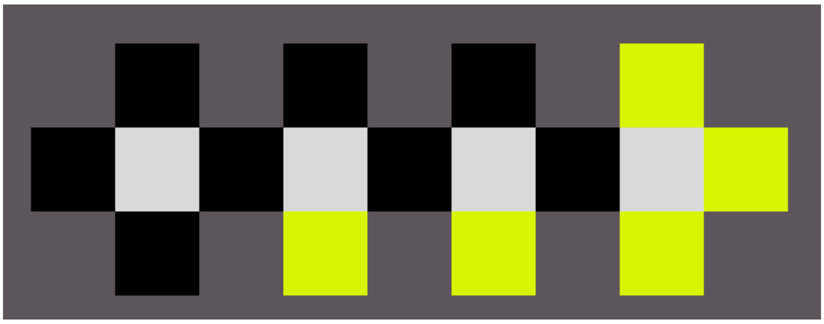
Задача C. Ксоня та двокольорова фігура

Автор та розробник: Святослав Бідзіля

Спочатку знайдемо, коли відповіді немає.

Порахуємо найбільшу площу фігури, в якій x чорних клітинок. Будемо фарбувати нашу фігуру. На першому кроці зафарбуємо будь-яку чорну клітинку в жовтий колір, а всіх її сусідів(білі клітинки) в синій. На кожному наступному колі будемо фарбувати будь-яку чорну клітинку, яка сусідня з синьою в жовтий, і всіх її сусідів в синій. Так, рівно за x кроків ми розфарбуємо всю фігуру (бо чорних клітинок рівно x). На першому колі ми фарбуємо максимум 5 клітинок, на кожному наступному - максимум по 4. Тому, максимальна площа такої фігури - $4x + 1$.

Якщо $4 \cdot min(w, b) + 1 < w + b$ відповіді немає. Інакше, покажемо одну з конструкцій, як можна досягти будь-якої відповіді. Без обмеження загальності, нехай $w < b$. Розташуємо білі клітинки в ряд, через одну. Між ними поставимо чорні клітинки. (Це можливо, бо $w < b$). Далі, всі чорні клітинки, що залишилися можна додати до будь-якої вільної чорної клітинки зверху, знизу, або збоку. Очевидно, що така фігура досягає потрібної площі і подібне рішення нескладно реалізується.



Тут зображений варіант конструкції. Жовтим відмічені місця, куди можна додавати чорні клітинки.

Задача D. Ксоня та граф

Автор та розробник: Святослав Бідзіля

Можна набрати часткові бали розв'язками за допомогою алгоритмів Дейкстри та Флойда, які шукають діаметр в довільному графі.

Щоб отримати вищі бали знайдемо особливість графу з умови. Оскільки граф з $n - 1$ ребром утворює дерево, то граф з n ребер утворює дерево з одним циклом. Будемо розглядати граф, як цикл і кореневі дерева, з коренями в вершинах циклу. Далі є два різних випадки, де може бути діаметр. Або діаметр цього графа знаходиться в одному з дерев, або проходить через цикл.

Для першого випадку нескладно скористатися будь-яким відомим алгоритмом пошуку діаметра в дереві та для кожного дерева знайти діаметр сумарно за $O(n)$.

Для другого випадку важливо зрозуміти, що якщо діаметр проходить через дві вершини циклу (назвемо їх u і v), то та частина діаметра, яка знаходиться в деревах з цими коренями доходить до найглибшого листа в такому дереві. Дійсно, якби вона доходила не до найглибшого листа, можна було б легко покращити відповідь, пішовши від вершини до найглибшого листа.

Знайдемо для кожної вершини циклу глибину найбільшого листа в її дереві (за допомогою dfs-у). Назвемо це число для i-ї вершини h_i. Назвемо відстанню між двома вершинами в циклі $dist(x, y)$ найкоротшу відстань по ребрах циклу між вершинами x і y. Тепер задача звелася до максимізації $h_x + h_y + dist(x, y)$ по усім x, y на циклі.

Як ефективно порахувати $dist(x, y)$? Перенумеруємо вершини циклу від 1 до m. Запишемо всі ваги ребер циклу в масив w, так що вага ребра з вершини i циклу в наступну записана в w_i. Нехай $\sum_i^m w_i = S$. Тоді порахуємо відстань, якби ми йшли по одній стороні циклу $f(i, j) = \sum_{k=i}^{j-1} w_k$. В іншому випадку, ми йдемо по іншій стороні циклу, тому використовуємо всі інші ребра. Тому $dist(i, j) = min(S - f(i, j), f(i, j))$.

Вже зараз задачу можна розв'язати на високий бал за $O(n^2)$, рахуючи шукану величину по всім вершинам циклу.

Для отримання повного балу можна помітити, що при фіксованому i існує таке k, що при всіх вершинах j від i до k $f(i, j) < S - f(i, j)$, а при всіх вершинах j від k до i $f(i, j) >= S - f(i, j)$. Іншими словами, функція монотонна. Для всіх вершин з першої групи відстань дорівнює $f(i, j) = p_{j-1} - p_{i-1}$(Якщо ввести масив p - префіксних сум масиву w), а діаметр на таких вершинах зводиться до $min(h_x + h_y + p_{y-1} - p_{x-1}) = min((h_x - px - 1) + (h_y + p_{y-1}))$. При фіксованому x такий мінімум по всіх j можна легко знайти за допомогою різних структур даних - дерева відрізків, підтримуючи множину усіх кандидатів і т.д. Сума для другої групи вершин схожа і підтримується аналогічно. Для пошуку оптимального k можна використовувати бінарний пошук, або підтримувати k методом двох вказівників.

При використанні сетів або дерева відрізків з'являється додатковий логарифмічний фактор, тому асимптотика рішення $O(n \log n)$.

Задача E. Ксоня та дерево

Автор та розробник: Святослав Бідзіля

Спробуємо розв'язати задачу, якщо в нас вже множина чисел S з піддерева і нам потрібно знайти k-е число з множини всіх можливих XOR-сум підмножин.

У випадку, якщо всі числа є степеню двійки можна помітити, що якщо число є в наборі, то ми можемо або додати його у відповідь, або ні. Так можна зробити з кожним бітом, тому всього буде в множині $2^{|S|}$ чисел. Побудувати k-е число можна жадібно - будемо йти з найбільших чисел до найменших. Якщо ми можемо не ставити якийсь біт (чисел з цим бітом не встановленим $\geqslant k$), то ми не будемо його ставити. Інакше, поставимо цей біт і зменшимо k на кількість чисел, в яких цей біт не встановлений.

Для розв'язання задачі в загальному вигляді потрібно розглянути числа, як набір векторів над полем Z_2. Тоді, xor двох чисел - це додавання цих векторів. Тому, для знаходження всіх можливих xor-ів підмножин, можна побудувати базис векторів, які представляють числа з множини S і подібно до рішення в минулому пункті жадібно встановлювати біти від більшого до меншого. Побудова базису може бути реалізована за $O(\log^3 A)$(Може бути пришвидшено в 32 рази використовуючи бітові операції на числах), а пошук k-го елементу в побудованому базисі - за $O(logA)$.

Тепер, повернемося до оригінальної задачі. Для перших трьох підгруп було достатньо знайти всю множину чисел пошуком вглибину, а далі використати розв'язок з першої частини.

Блок 1 Будь-який повний перебір за $O((n + q)2^n)$ проходив цей блок.

Блок 2-3 Блок проходився якщо перераховувати базиси для всіх вершин за $O(n)$, або знаходити кожний раз заново.

Блок 4 Можна реалізувати задачу для множини чисел, а не дерева.

Блок 5 Якщо в підгрупі немає запитів на зміну числа, то можемо передрахувати всі базиси для кожної вершини дерева - базис вершини це просто базис чисел з базисів її дітей. Це можна порахувати за $O(n \log^2 A)$, а далі відповідати за запити за $O(\log A)$.

Блок 6 Блок вирішувався простішим варіантом базису, достатньо було перевірити наявність потрібного біта в піддереві.

Блок 7 У загальному випадку, можемо звернутися до техніки Ейлерового обходу дерева. Побудуємо за Ейлеровим обходом дерева дерево відрізків, де в кожній вершині буде зберігатися XOR базис її підвідрізка. Для мерджа двох вершин додамо вектори одного базису в інший. Піддерево кожної вершини тепер буде цілісним відрізком на Ейлеровому обході, а отже запит базису на піддереві зводиться до одного запиту до дерева відрізків, а запит зміни числа - до одиничної заміни в дереві відрізків. Реалізація такого дерева відрізків працює за $O(\log^2 A \log n)$ на запит, тому все рішення працює за $O((n+q)\log^2 A \log n)$.

Задача A. Козак Вус та таємний лист

Автор та розробник: Андрій Романов

Є $a \cdot b \cdot c$ способи придбати один конверт, одну марку з шароварами та одну марку з вишиванкою, $a \cdot b$ способи придбати один конверт і одну марку з вишиванкою і $a \cdot c$ способи купити один конверт і одну марку з шароварами. Тому загальна кількість способів здійснити покупку рівна $a \cdot b \cdot c + a \cdot b + a \cdot c$.

Задача B. Козак Вус та чарівні черевики

Автор та розробник: Андрій Романов

Намалюємо дерево з направленими вниз ребрами та коренем у вершині 1, яке буде зображати всі можливі комбінації застосування заклять. З вершини x можна перейти вниз у вершини $2 \cdot x$ та $2 \cdot x + 1$, адже, маючи швидкість x метрів на секунду, за допомогою одного закляття швидкість може стати рівною $x + x = 2 \cdot x$ метри на секунду або $x + (x + 1) = 2 \cdot x + 1$ метри на секунду. Кожен раз, спускаючись вниз, від кореня ми переходимо або в лівий вузол, тобто застосовуємо перше закляття або в правий вузол — застосовуємо друге закляття. Кожна вершина зустрічатиметься рівно один раз, а отже порядок застосування заклять, щоб потрапити з вершини 1 у вершину x єдиний, причому необхідна кількість заклять рівна відстані від 1 до x. Помітимо, що на відстані 0 від кореня знаходиться вершина 1, на відстані 1: 2, 3, на відстані 2: 4, 5, 6, 7, на відстані 3: 8, 9, 10, 11, 12, 13, 14, 15 і так далі. Тобто на відстані n від кореня знаходяться числа від 2^n до $2^{n+1} - 1$, і щоб їх досягти ми відповідно застосуємо закляття n разів.

Отже, якщо $2^n \leqslant k \leqslant 2^{n+1} - 1$, то відповідь n. Інакше кажучи, $n = \lfloor \log k \rfloor$. Складність розв'язку $O(\log k)$.

Задача C. Козак Вус та залізнична дорога

Автор та розробник: Андрій Романов

Цю задачу можна вирішувати багатьма способами, я наведу один з них.

Блок 1. Можемо зробити повний перебір по всіх розстановках ліхтарів і перевірити, чи хоча б одна з них задовольняє умові задачі. Складність такого розв'язку $O(n! \cdot n)$.

Блок 2. Для зручності будемо називати ті $n/2$ ліхтарі, що мають один і той самий колір червоними. Розставимо $n/2 - 1$ червоні ліхтарі на верхню гілку і між ними поставимо $n/2 - 1$ не червоних ліхтарів, в тому числі один з них на перехресті. В нас лишився один червоний і один не червоний ліхтар(назвемо його особливим). Якщо $k \geqslant 3$, то за потреби замінимо особливий ліхтар на не червоний такий, колір якого відрізняється від кольору ліхтаря на перехресті та два ліхтарі, що в нас лишилися поставимо на нижню гілку. Якщо ж $k = 2$, то обов'язково кількість ліхтарів кольору 1 і кольору 2 буде рівна як на верхній, так і на нижній гілках, але через те, що ліхтар на перехресті враховується двічі, загалом кількості ліхтарів цих двох кольорів будуть відрізнятися. Отже, при $k = 2$ відповідь в цьому блоці завжди -1.

Блок 3. З обмежень блоку випливає, що будь-які два ліхтарі будуть мати різні кольори, відповідно й будь-які два сусідні ліхтарі точно будуть різних кольорів. Розставимо ліхтарі довільним чином, наприклад $n - 3$ з них поставимо на верхню гілку, 1 на перехрестя і 2 на нижню гілку. Таке розташування задовольняє умові.

Блок 4. Як і в першому блоці можемо перебрати всі можливі розстановки. Складність такого розв'язку $O(2^n \cdot n)$.

Блок 5. Як на верхній, так і на нижній гілці кількість одиниць і двійок буде однаковою, але одиниця/двійка на перехресті буде врахована два рази, а отже кількість одиниць/двійок на один більша за кількість двійок/одиниць. Отже, якщо n парне або кількості одиниць і двійок відрізняються хоча б на 2, то відповідь -1. Інакше розставимо $(n+1)/2$ одиничок і $(n-1)/2$ двійок. Випадок, коли двійок більше розглядається аналогічно. Якщо $n = 5$, то нам не вдасться розставити ці ліхтарі вздовж залізничної дороги, адже завжди знайдуться два сусідні ліхтарі одного кольору. Інакше поставимо ліхтар 2 на перехрестя. Розставимо $n - 4$ ліхтарі в порядку $121 \ldots 21$ на верхній гілці. Тепер розставимо 3 ліхтарі 121 на нижній гілці. Задане розташування задовольняє умові задачі.

Блок 6. Назвемо ліхтар максимальним, якщо ліхтарів такого ж кольору як в нього найбільше серед інших ліхтарів, мінімальним, якщо ліхтарів його кольору найменше серед інших. На обидвох гілках максимальних ліхтарів не може бути більше ніж сумарно ліхтарів інших кольорів, бо інакше б якісь два максимальні ліхтарі стояли поруч. При цьому перехрестя враховувалося два рази, тому якщо на перехресті стоїть не максимальний ліхтар, то кількість максимальних ліхтарів може бути більшим на 1 за кількість ліхтарів всіх інших кольорів. Отже, якщо кількість максимальних ліхтарів хоча б на 2 більша за сумарну кількість всіх інших ліхтарів, то відповідь -1, інакше розв'язок існує.

Поставимо на перехрестя мінімальний ліхтар. Тепер будемо виставляти на верхню гілку максимальний і мінімальний ліхтарі на цей час по черзі. Будемо продовжувати цей процес доки в нас не залишаться ліхтарі лише одного кольору. З обмежень, які ми розглянули в минулому абзаці в нас може залишитися не більше двох ліхтарів одного кольору. Можна показати, що останній ліхтар на верхній гілці відмінного кольору від ліхтаря на перехресті, адже такі ліхтарі були використані в першу чергу, а $k \geqslant 3$ (при $k = 1$ відповідь -1). Якщо в нас на руках два ліхтаря одного кольору, то поставимо їх на нижню гілку і між ними поставимо останній ліхтар з верхньої гілки. Якщо в нас на руках один ліхтар, то поставимо його на нижню гілку разом з останнім ліхтарем з верхньої гілки. Якщо в нас на руках не лишилося жодного ліхтаря, то переставимо останні два ліхтарі з верхньої гілки на нижню гілку. Передостанній ліхтар з верхньої гілки був максимальним, отже його колір точно відмінний від ліхтаря на перехресті. Таким чином ми отримаємо розстановку, що задовольняє умові задачі. Максимальний і мінімальний ліхтарі в кожен момент часу можна шукати проходом по масиву. Тоді складність алгоритму $O(n^2)$.

Блок 7. Об'єднуємо 5 і 6 блоки разом.

Блок 8. Будемо підтримувати сет, де будемо зберігати пари поточна кількість ліхтарів кольору x, x. Таким чином зможемо шукати максимальний і мінімальний ліхтарі за $\log(n)$. Загальна асимптотика $O(n \log n)$.

Задача D. Козак Вус та дерево

Автори та розробники: Андрій Романов, Антон Ципко, Костянтин Денисов

Блок 1. Перевіримо, чи сума ваг ребер графа не перевищує k_1. Якщо ні, то відповідь 0, інакше відповідь 1.

Блок 2. Знайдемо для кожної вершину вигоду — суму ваг ребер, які з неї виходять. Тоді нам вигідно або зовсім не фарбувати вершини в чорний колір, або пофарбувати в чорний колір вершину з найбільшою вигодою. Перевіримо, чи підійде нам один з цих двох варіантів. Якщо жоден не підійшов, то відповідь 2.

Блок 3. Напишемо динаміку `dp[x][y]` — найменша можлива сума ваг яскравих ребер, якщо ми розглядаємо вершини від 1 до x і пофарбували y з них в чорний колір. Перехід `dp[x][y] = min(dp[x-1][y-1]+edge[x-1][x], dp[x-2][y-1]+edge[x-2][x-1]+edge[x-1][x], dp[x-1][y])`, де `edge[x-1][x]` — вага ребра між вершинами $x-1$ та x. Тоді `dp[n][1]`, `dp[n][2]`, ..., `dp[n][n]` — значення мінімальних можливих сум ваг яскравих ребер, якщо пофарбовано i вершин. Застосувавши бінарний пошук на цьому масиві зможемо відповідати на запит за $O(\log n)$. Загальна складність $O(n^2 + m \log n)$.

Блок 4. Через те, що заданий всього один запит і n маленьке, можемо перебрати всі можливі комбінації вибору набору вершин, які ми пофарбуємо в чорний і знайти найкращу з них (тобто ту в якій залучена найменша кількість вершин). Складність $O(n! \cdot n)$.

Блок 5. Напишемо квадратну динаміку `dp[v][cnt][flag]` — мінімальна можлива вага яскравих ребер, якщо розглядати лише вершини піддерева v, cnt з них пофарбували в чорний колір, `flag=1`, якщо v чорна і `flag=0`, якщо v біла. Будемо поступово приєднувати до вершини v піддерева її синів. Коли ми приєднуємо піддерево вершини to, то переберемо всі можливі комбінації кількості вершин, які ми пофарбуємо. В піддереві вершини v ми можемо пофарбувати в чорний колір від $\mathrm{I} = 0$ до $\mathrm{I} = \mathrm{n}$ вершин, в піддереві вершини to від $J = 0$ до $J = n$ вершин. Переходами будуть:

1. `dp[v][I+J][0]=min(dp[v][I+J][0], dp[v][I][0]+dp[to][J][1]);`
2. `dp[v][I+J][0]=min(dp[v][I+J][0], dp[v][I][0]+dp[to][J][0]+edge[to][v]);`
3. `dp[v][I+J][1]=min(dp[v][I+J][1], dp[v][I][1]+dp[to][I][0]);`
4. `dp[v][I+J][1]=min(dp[v][I+J][1], dp[v][I][1]+dp[to][J][1]);`

Тепер побудуємо масив `ans[cnt]=min(dp[root][cnt][0], dp[root][cnt][1])`, де $root$ — корінь дерева. `ans[cnt]` — мінімальна можлива сума ваг яскравих ребер, якщо в чорний колір пофарбовано cnt вершин. Побудувавши даний масив можемо бінарним пошуком знаходити відповідь на кожен запит — знаходимо мінімальне таке cnt, що $ans[cnt] \leqslant k_i$. Таким чином складність цього розв'язку $O(n^3 + m \log n)$.

Блок 6. Помітимо, що I та J можемо перебирати не до n, а до `cnt[v]` і `cnt[to]` відповідно. Тут `cnt[v]` — поточний розмір піддерева вершини v, `cnt[to]` — розмір піддерева вершини to. На перший погляд, може здатися, що складність розв'язку не змінилася, але насправді тепер ми підраховуємо dp за $n \cdot n$ — перевірте це самостійно. Асимптотика рішення $O(n^2 + m \cdot \log n)$.

Задача E. Козак Вус та граф

Автор та розробник: Антон Ципко

Блок 1. Якщо $k_i = 1$, то потрібно вивести цю вершину. Інакше — -1.

Блок 2. В операціях другого типу можемо використовувати СНМ (систему неперетинних множин). В операціях першого типу можемо проходитися по всім вершинах і перевіряти, чи ця вершина належить множині. Для того, щоб перевірити, чи вершина знаходиться у множині, потрібно перевірити чи корінь цієї вершини у СНМ збігається з коренем запитаної вершини.

Блок 3. Можемо для кожного стану зберігати масив СНМ. Після кожної операції нам потрібно запам'ятати масив СНМ. Тоді, коли є операція другого типу, то нам потрібно скопіювати повністю масив, а потім виконати на ньому операцію. А коли є операція третього типу, то потрібно лише скопіювати правильний масив.

Блок 4. Обмеження означає, що компонента буде відрізком на масиві. Коли ми отримуємо запит другого типу, то нам потрібно знайти крайню вершину зліва, це можна зробити бінарним пошуком, а потім перевірити чи k-та вершина справа також знаходиться в цій же компоненті. Також можна зберігати відрізки у `set< pair<int, int> >`, а потім використовувати lower_bound для знаходження відповіді.

Блок 5. Можемо для кожної компоненти зберігати множину вершин в ній. Коли у нас є операція другого типу, нам потрібно додати з меншої за розміром множини у більшу. Цей метод називається «від меншого до більшого». Він дасть можливість виконати всі ці операції сумарно за $O(n \log^2 n)$. На операції першого типу потрібно знаходити k-ий елемент у множині.

Множину можна підтримувати за допомогою декартового дерева, або за допомогою спеціальної структури даних в C++, яка дуже схожа на set, але крім операцій вставки, також вміє шукати k-ий елемент.

```
#include <ext/pb_ds/assoc_container.hpp>
using namespace __gnu_pbds;
typedef tree<int, null_type, less<int>, rb_tree_tag,
tree_order_statistics_node_update> ordered_set;
// s.find_by_order(num) - returns iterator to the num-th element (0 <= num < s.size())
// s.order_of_key(key) - returns number of elements smaller than key
```

Блок 6. Можемо підтримувати персистентне дерево відрізків. Для операцій другого типу нам потрібно вміти присвоювати на відрізку. А для операцій першого типу потрібно вміти знаходити перший елемент, який має таке ж значення (початок компоненти). Дуже схожий метод до четвертого блоку.

Блоки 7-9. Ми можемо відповідати на запити оффлайн, тому ми можемо побудувати dfs на запитах. Тобто, у нас будуть вершини - запити. У нас в кожну вершину типу 1 чи 2 можна потрапити через попередню за номером вершину, а у i-ту вершину третього типу, можна потрапити лише з вершини x_i. Тобто, якщо $t_i = 3$, то у нас буде ребро (x_i, i), якщо ж $t_i \neq 3$, то буде ребро $(i-1, i)$. Такий dfs дозволить підтримувати певний стан, проте такий стан потрібно вміти оновлювати та відкочувати. Основна проблема саме у відкочуванні.

Давайте для початку розберемося, як це робити за $O(nq \log n)$. Будемо підтримувати СНМ, проте ми не будемо використовувати оптимізацію, у якій усі вершини будуть вести у корінь. Кожна вершина буде вести у свого прямого батька, а не в корінь. Коли у нас є операція другого типу, то, щоб об'єднати компоненти, нам потрібно під'єднати меншу компоненту до більшої. Тобто, нехай ми об'єднуємо компоненти вершин v_i та u_i. Хай v'_i — корінь компоненти v_i в СНМ. u'_i — аналогічно для u_i. Також нехай компонента v'_i менша за компоненту u'_i. Тоді нам потрібно сказати, що батьком вершини v'_i є вершина u'_i. Потім, щоб скасувати це об'єднання, нам потрібно сказати, що батьком вершини v'_i є сама вершина v'_i (тобто, вона сама є коренем). Для операцій першого типу, ми можемо пройтися по всім вершинам і перевірити, у якій компоненті вона знаходиться. Потрібно буде вивести k-ту таку вершину. Для кожної вершини ми можемо знайти корінь компоненти за $O(\log n)$ (згадуємо метод «від меншого до більшого» з п'ятого блоку). Отже, ми вміємо обробляти запит першого типу за $O(n \log n)$, а другого за $O(\log n)$.

Розіб'ємо усі вершини на $\sqrt{n}$ блоків довжини $\sqrt{n}$. Для i-го блоку будемо зберігати кількість вершин у цьому блоці, які знаходяться в компоненті вершини j. Нехай це буде зміна c_{ij}. Якщо у нас є операція другого типу, то ми можемо кожен блок оновити за $O(1)$, додавши одне значення до іншого. Отже, за одну операцію другого типу нам потрібно $O(\sqrt{n})$ часу, щоб оновити масив c. Стільки ж часу потрібно, щоб роз'єднати компоненти. Для виконання операції першого типу, будемо спочатку проходитися по всім блокам і дивитися, у якому блоці знаходиться наша шукана відповідь. Тут ми витратимо $O(\sqrt{n})$ часу. Потім ми будемо перебирати усі вершини, так само як у попередньому абзаці, проте будемо перебирати не абсолютно всі вершини, а лише ті, що в цьому блоці. Отже, для такого нам потрібно $O(\sqrt{n} \log n)$ часу. Сумарна асимптотика виходить $O(q\sqrt{n} \log n)$.

Зверніть увагу, що необов'язково використовувати саме $\sqrt{n}$ як розмір блока. Нехай s — розмір блока. Тоді асимптотика вийде $O(q \max(\frac{n}{s}, s \log n))$.