Задача A1. Хлопці та дівчата

Автор задачі: **Антон Тригуб**
Задачу підготував: **Олександр Гроскопф**
Розбір написав: **Павло Цікалишин**

Блок 1

Якщо в множині вже є хоча б один хлопець певного типу x, то всі інші хлопці цього ж типу також мають бути присутні. Отже, ми можемо перебирати всі підмножини типів хлопців, перевіряти, чи вони задовольняють умову задачі, та вибирати найкращий варіант.

Блок 2

Розглянемо задачу як граф, де вершини відповідають дівчатам, а хлопці є ребрами, що з'єднують ці вершини. Нам потрібно, щоб усі ребра в множині мали хоча б одну спільну вершину. Такі структури можуть бути лише двох типів:

1. **Зірка** — всі ребра виходять із однієї вершини.
2. **Трикутник** — три вершини, кожна з яких з'єднана з двома іншими.

Щоб знайти відповідь для зірки, достатньо визначити її центр x і підрахувати суму всіх ребер, що виходять із цієї вершини.

Для трикутника потрібно перебрати всі ребра (u, v) і знайти вершину, разом із якою вони утворюють трикутник.

Блок 3

Оскільки кожна вершина може мати не більше двох вихідних ребер, ми можемо перебрати всі вершини, що мають рівно два сусіди, і перевірити, чи ці сусіди також з'єднані між собою. Якщо так, то знайдено трикутник.

Блок 4

У цій групі достатньо для кожної вершини x перебрати всі пари її сусідів (u, v). Потім перевіряємо, чи існує ребро між вершинами u і v. Це працює за складністю $\mathcal{O}(n^2)$, оскільки кожна вершина може мати до n сусідів.

Блоки 5

Щоб швидко знайти всі трикутники, можна розділити вершини на **важкі** (з більш ніж $\sqrt{n}$ сусідами) і **легкі** (з меншою кількістю сусідів).

Спочатку рахуємо всі трикутники, що містять хоча б одне легке ребро. Це можна зробити так само, як у попередньому підході, і це займе $\mathcal{O}(n\sqrt{n})$ часу. Потім знаходимо трикутники, що складаються тільки з важких вершин. Для цього видаляємо всі легкі вершини та повторюємо підрахунок. Це також виконується за $\mathcal{O}(n\sqrt{n})$.

Загальна складність алгоритму — $\mathcal{O}(n\sqrt{n})$.

Блок 6-7

Щоб знайти найкращий трикутник, потрібно для кожної вершини x розглянути трикутник, утворений двома найбільшими ребрами, що виходять з x (якщо такий існує). Таких трикутників є $O(n)$, тому отримаємо розв'язок за $O(n \log n)$ або $O(n)$ в залежності від способу пошуку третього ребра таких трикутників.

Доведемо, що якщо оптимальна відповідь є трикутником, то ми її розглянемо. Дійсно, нехай оптимальна відповідь є трикутником зі сторонами, що мають ваги l_1, l_2, l_3, $l_1 \leqslant l_2 \leqslant l_3$. Нехай з вершини, що знаходиться між ребрами l_2 та l_2 (позначимо її v) виходить ще якесь ребро $r > b$. Тоді зірка з центром в v буде кращою відповіддю, ніж наш трикутник:

$$зірка \geqslant l_2 + l_3 + r > l_2 + l_3 + l_1 = трикутник$$

.

Задача A2. Манхетенське парування

Автор задачі: **Антон Тригуб**
Задачу підготувала: **Анастасія Товтин**
Розбір написала: **Анастасія Товтин**

Блок 1

Є лише дві точки, які обов'язково потрібно об'єднати в пару. Відповіддю буде манхеттенська відстань між цими точками: $d = |x_1 - x_2| + |y_1 - y_2|$

Блок 2

Спочатку потрібно поєднати попарно точки, які мають однакові y-координати (однакові точки), оскільки їх манхеттенська відстань буде рівною 0. Якщо залишаться дві різні точки, їх доведеться об'єднати, і їхня манхеттенська відстань буде 1.

Блок 3

Оскільки n не більше 4, то можна перебрати всі можливі варіанти розбиття точок на пари. Це можна уявити як множину перестановок, у яких кожні дві сусідні точки утворюють пару. Для кожної такої перестановки визначається максимальна манхеттенська відстань серед пар, і серед усіх варіантів вибирається той, у якому ця величина є найменшою.

Блок 4

Можна використати динамічне програмування по масках. Стан у цьому випадку визначається підмножиною точок, які вже поєднані у пари, і зберігається найменше можливе значення максимальної манхеттенської відстані серед усіх можливих способів парування цих точок. На кожному кроці вибираємо дві ще не використані точки, обчислюємо відстань між ними й додаємо їх у розбиття, оновлюючи значення dp.

Блок 5

Якщо всі точки мають $y = 0$, тобто вони всі лежать на горизонтальній прямій. У такому випадку достатньо впорядкувати точки за x і попарно поєднати сусідні. Максимальна відстань серед усіх пар буде визначатися найбільшою різницею між сусідніми значеннями x.

Блок 6

Манхеттенська відстань між будь-якими двома точками визначається як $|x_1 - x_2| + |y_1 - y_2|$. Оскільки всі значення x різні, найбільший внесок у відстань дає саме різниця між x-координатами, а y-координати можуть змінювати відстань лише на 0 або 1 (оскільки у може бути тільки 0 або 1).

Тоді можна спростити задачу, просто відсортувавши точки за x-координатою та поєднавши сусідні точки у списку. Це працює оптимально, тому що сортування гарантує, що точки, які знаходяться найближче одна до одної за x-координатою, будуть сусідами.

Блок 7 - 8

Давайте тимчасово проігноруємо y-координати та розрахуємо відповідь, наче всі точки лежать на одній прямій (тобто, всі $y = 0$).

Нехай отримане значення ми позначимо як ans, яке є мінімальним можливим значенням максимальної манхеттенської відстані між парними точками у цьому спрощеному випадку. Очевидно, що остаточна відповідь буде або ans, або $ans + 1$, оскільки різниця y-координатах може вплинути лише на 1.

Тепер нам потрібно перевірити, чи можливо досягти саме ans.

Для цього будемо проходити по префіксу точок, посортованих за -координатою, для яких $x_i \leqslant pref$. У процесі є три можливі ситуації:

- Всі точки з цього префіксу вже об'єднані у пари.
- Залишилася одна точка з $y = 0$.

- Залишилася одна точка з $y = 1$.

Оскільки точка, що залишилася, є крайньою правою серед оброблених, ми точно знаємо її x-координату. Для ефективного відстеження можна використовувати динамічний масив $dp[pref][0/1/2]$, де:

- $dp[pref][0]$ означає, що всі точки до pref вже об'єднані у пари.
- $dp[pref][1]$ означає, що залишилася одна точка з $y = 0$.
- $dp[pref][2]$ означає, що залишилася одна точка з $y = 1$.

Нехай cnt_0 це кількість точок з координатою $(pref, 0)$, та cnt_1 з координатою $(pref, 1)$.
Розглянемо переходи зі стану $dp[prev\_pref][0]$:

- якщо $cnt_0 \mod 2 = cnt_1 \mod 2$:
  - якщо $cnt_0 \mod 2 = 0$, то в стан $dp[pref][0]$
  - якщо $cnt_0 \mod 2 = 1$ в стан $dp[pref][0]$, але тільки якщо $ans \geqslant 1$.
- якщо $cnt_0 \mod 2 \neq cnt_1 \mod 2$:
  - якщо не існує жодної точки з координатою $(pref, 0)$, тобто $cnt_0 = 0$, то в стан $dp[pref][2]$
  - якщо не існує жодної точки з координатою $(pref, 1)$, тобто $cnt_1 = 0$, то в стан $dp[pref][1]$
  - інакше можна перейти як в стан $dp[pref][1]$ так і в стан $dp[pref][2]$.

Переходи зі станів $dp[prev\_pref][1]$ та $dp[prev\_pref][2]$ є аналогічними, але потрібно ще розглянути обидва випадки, до якої з точок $(pref, 0)$ та $(pref, 1)$ з'єднується нез'єднана точка з префіксу $prev\_pref$ (але дозволені тільки з'єднання для яких манхеттенська відстань буде не більшою за ans), та працювати вже з оновленими значеннями cnt_0 та cnt_1 (зі значеннями які враховують нове з'єднання, і відповідно зменшення однієї з кількостей на 1).

Після обробки всіх точок, тобто коли ми дійшли до найбільшого значення $pref_{max}$, якщо $dp[pref_{max}][0]$ рівне 1, то значення ans є досяжним, інакше відповіддю є $ans + 1$. Часова складність $\mathcal{O}(n \cdot log(n))$,

Задача B1. Розбиття на три

Автор задачі: **Антон Тригуб**
Задачу підготував: **Ігнат Жаріхін**
Розбір написав: **Ігнат Жаріхін**

Блок 1 ($n = 3$)

Очевидно, існує єдине можливе розбиття на три підмасиви, кожен розміром один елемент.

Блок 2 ($a_i \leqslant 1$)

Спочатку вирішимо, які будуть суми в підмасивах. Нехай змінна cnt означає кількість одиничок в масиві. Логічно, кожен підвідрізок зробити за сумою якомога ближче до $\left\lfloor \frac{cnt}{3} \right\rfloor$. Якщо $cnt \mid 3$, то все добре і ми знайшли суми. Інакше, у нас може бути ще 1 чи 2 залишкові одинички, які ми не розподілили нікуди. Якщо вона одна, додаємо її в будь-який підмасив, а якщо їх дві — розподіляємо їх в будь-які два різні підмасиви.

Коли ми знаємо суми, просто проходимось по масиву і додаємо наступний елемент в поточний підмасив. Якщо досягли його суми(а пропустити її ми не можемо, бо $a_i \leqslant 1$), зачиняємо його і відкриваємо наступний. Таким чином знаходимо межі підмасивів.

$a + b = c$

Блок 3 (гарантується, відповідь 0**)**

Порахуємо повну суму масиву, нехай це sum. Додаткові обмеження цієї групи означають, що є таке розбиття, де всі три підмасиви мають суму $\frac{sum}{3}$. Будь-які підмасиви з іншою сумою нас взагалі не цікавлять.

Для кожного індексу i порахуємо такий мінімальний індекс $next_i \leqslant n$(якщо він існує), що сума на відрізку $[a, b)$ дорівнює $\frac{sum}{3}$. Якщо не існує, запам'ятаємо це.

Проходимось по масиву, перебираючи початок першого підмасиву — i. Якщо не існує $next_i$, то цей початок точно не підходить, пропускаємо. Тоді, нехай $j = next_i$. Аналогічно, обов'язково треба, аби $next_j$ існував, бо інакше таке розбиття не підходить. Якщо ж він існує, ми знайшли всі три межі між підвідрізками. Перші два з них мають суму $\frac{sum}{3}$, отже і третій має таку саму, тобто ми знайшли відповідь. Складність — $\mathcal{O}(n)$

Блок 4 ($n \leqslant 100$)

Повністю переберемо три початки підмасивів, як це запропоновано в «форматі вихідних даних». Коли маємо межі x, y, z, рахуємо суми просто проходом по масиву чи префіксними сумами. Складність — $\mathcal{O}(n^4)$ чи $\mathcal{O}(n^3)$, в залежності від того, який метод обрати.

Надалі, всі суми будемо рахувати за допомогою префіксних сум.

Блок 5 ($n \leqslant 2000$) **та 6** ($n \leqslant 5000$)

Переберемо лише перші два індекси. Наразі за допомогою них ми розбили масив на два підвідрізки. Один з них залишаємо, і його суму рахуємо, нехай це буде s. Тепер нам залишилось розбити другий підвідрізок ще на два: префікс і суфікс. Помітимо, що багато з цих розбиттів точно не оптимальні. Інтуїтивно, нам цікаво розбити масив на два максимально рівні за сумою. Теоретично, нехай ми розбили таким чином, що суми префікса і суфікса дорівнюють $pref$ і $suff$ відповідно. Без втрати загальності, нехай $pref \leqslant suff$. Припустимо, що один елемент масиву(той, що зараз на межі) можна перевести з суфікса в префікс(те саме, що змістити межу на один), і при цьому нерівність залишиться вірною. Формально, $pref + x \leqslant suff - x$. Нам завжди буде вигідно зробити цей перехід, тому що $\min(s, pref) \leqslant min(s, pref + x)$ і $\max(s, suff) \geqslant max(s, suff - x)$, а отже відповідь буде не гіршою. Єдиний момент, коли нам може бути не вигідно робити перехід, це коли знак в нерівності змінюється. Отже, треба фактично перевірити лише два випадки: найбільший префікс, сума якого менша за суму суфікса та навпаки, найбільший суфікс, сума якого менша за суму префіксу. Зі всіх інших можна перейти в ці, не погіршуючи відповідь.

Перший варіант реалізації: після перебору перших двох індексів бінарним пошуком шукаємо переломний момент в сумах, і перевіряємо ці два випадки. Складність — $\mathcal{O}(n^2 \cdot \log n)$, проходить п'яту групу.

Другий варіант реалізації: Переломний момент шукаємо другим вказівником, який завжди рухається праворуч по циклічному масиву. Складність — $\mathcal{O}(n^2)$, проходить шосту групу.

Блок 7 ($n \leqslant 10^5$) та 8 (без додаткових обмежень)

Нехай сума всіх елементів масиву це sum. Помітимо, що в будь-якому розбитті рівно одне з наступних двох тверджень правдиве:

- або рівно два масиви мають суму $\leqslant \left\lfloor \frac{sum}{3} \right\rfloor$ і третій має суму $\geqslant \left\lfloor \frac{sum}{3} \right\rfloor$,
- або рівно два масиви мають суму $\geqslant \left\lfloor \frac{sum}{3} \right\rfloor$ і третій має суму $\leqslant \left\lfloor \frac{sum}{3} \right\rfloor$.

Переберемо розріз масиву між двома підмасивами, що обидва мають суму, не більшу за третину sum. Аналогічно до доведення з попередньої групи, з усіх таких масивів вигідно взяти найбільші за сумою з обох сторін. Це можна зробити за допомогою бінарного пошуку або двома вказівниками. Суму третього відрізка рахуємо як залишок.

Аналогічно, перебираємо розріз між двома підмасивами, що обидва мають суму, більшу за третину sum, проте тепер мінімізуємо ці два підмасиви. Кінцева складність — $\mathcal{O}(n \cdot \log n)$ з бінпошуком (7 група) та $\mathcal{O}(n)$ зі вказівниками(повне рішення).

З точки зору реалізації, може допомогти подвоїти масив, вставивши в кінець копію початкового. Тоді замість того, щоб переходити з останнього елементу до початкового, можна просто продовжувати обхід масиву, єдине що треба буде перевірити, це те, чи не накладається кінець третього підмасиву на перший.

Задача B2. Подарунок для Антона

Автор задачі: **Костянтин Луценко**
Задачу підготували: **Костянтин Луценко**
Матвій Стречень
Павло Цікалишин
Розбір написав: **Костянтин Луценко**

Блок 0

Якщо є хоча б одна «4», то всі поруч з нею теж «4», поруч з цією «4» теж всі «4», і так далі. А таблиця скінченна, отже, такого не може бути.

Якщо є хоча б одна «3», то розглянемо найвищу з усіх «3». Тоді у всіх напрямках, окрім верхнього, вона межує з іншими «3». Розглянемо ту «3», що справа від початкової. В неї аналогічно зліва, знизу та справа інші «3». І так далі, можемо розширюватись вправо до нескінченності, а таблиця скінченна, суперечність.

Двійки утворюють циклічні структури. Одинички розташовані доміношками. Нулі є ізольованими від інших нулів. Це яскраво видно з прикладу з умови.

Описані вище роздуми не є обов'язковими, але подібний аналіз завдання дуже допомагає розв'язанню.

Блок 1 ($n = 1$)

Нескладно помітити, що, чергуючи доміношки «1-1» та ізольовані «0», можна отримати ланцюг довільної довжини.

Блок 2 ($n = 2$)

Якщо продублювати рішення на Блок 1 і збільшити всі числа на 1, то воно буде працювати на цей блок.

Блок 2 ($n = 3$)

Якщо обрамлення зробити з «2», то лишиться стрічка $1 \times (m - 2)$, для якої можна викликати рішення для Блоку 1.

Блок 3

Візьмемо приклад з умови. Відріжемо найлівіший і найправіший стовпець. Кути поміняємо з «1» на «0».

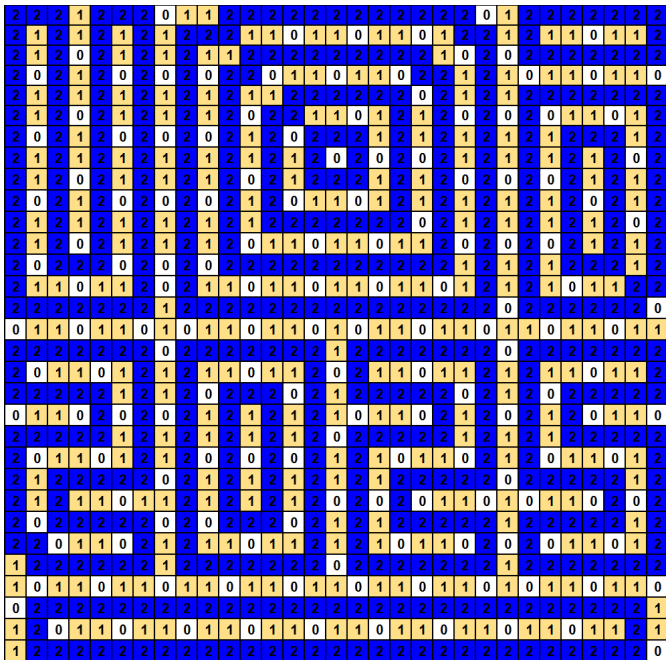
Блоки 5-11

Якщо поставити поруч з квадратиком з «2» дві доміношки з «1» та ізольований «0», то отримаємо паттерн 3×3, який можна повторювати нескінченно.

2	2	1	2	2	1	2	2	1	2	2	1	2	2	1	2	2	1
2	2	1	2	2	1	2	2	1	2	2	1	2	2	1	2	2	1
1	1	0	1	1	0	1	1	0	1	1	0	1	1	0	1	1	0
2	2	1	2	2	1	2	2	1	2	2	1	2	2	1	2	2	1
2	2	1	2	2	1	2	2	1	2	2	1	2	2	1	2	2	1
1	1	0	1	1	0	1	1	0	1	1	0	1	1	0	1	1	0
2	2	1	2	2	1	2	2	1	2	2	1	2	2	1	2	2	1
2	2	1	2	2	1	2	2	1	2	2	1	2	2	1	2	2	1
1	1	0	1	1	0	1	1	0	1	1	0	1	1	0	1	1	0
2	2	1	2	2	1	2	2	1	2	2	1	2	2	1	2	2	1
2	2	1	2	2	1	2	2	1	2	2	1	2	2	1	2	2	1
1	1	0	1	1	0	1	1	0	1	1	0	1	1	0	1	1	0
2	2	1	2	2	1	2	2	1	2	2	1	2	2	1	2	2	1
2	2	1	2	2	1	2	2	1	2	2	1	2	2	1	2	2	1
1	1	0	1	1	0	1	1	0	1	1	0	1	1	0	1	1	0
2	2	1	2	2	1	2	2	1	2	2	1	2	2	1	2	2	1
2	2	1	2	2	1	2	2	1	2	2	1	2	2	1	2	2	1
1	1	0	1	1	0	1	1	0	1	1	0	1	1	0	1	1	0

Ба більше, якщо розглянути конкретний рядок, то помітимо, що в нас завжди така ситуація, як в блоці 1: дві однакові цифри, одна інша, дві однакові, одна інша... Це значить, що так само як і в блоці 1, ми можемо обрати відрізок з m стовпців, щоб на границях не було проблем. Аналогічно можна обрати відрізок з n рядків, щоб на границях не було проблем. Відповідь — таблиця, що утворена перетином цих n рядків та m стовпців.

Блок 8



Задача C1. Випуклий масив

Автор задачі: **Антон Тригуб**
Задачу підготував: **Василь Мереніч**
Розбір написав: **Костянтин Денисов**

Твердження 1. У випуклого масиву різниці між сусідніми елементами не зменшуються. Тобто, якщо позначити $d_i = b_{i+1} - b_i$, то для кожного i виконується нерівність $d_i \leqslant d_{i+1}$.

Це безпосередньо випливає з визначення випуклості.

Оскільки послідовність різниць $d_i = b_{i+1} - b_i$ є неспадною, існує такий індекс k, що $d_i \leqslant 0$ для $i < k$ і $d_i \geqslant 0$ для $i \geqslant k$. Це означає, що масив можна розбити на дві монотонні випуклі послідовності: спадну $b_1, \ldots, b_k$ і зростаючу $b_k, \ldots, b_n$.

Тоді потрібно розбити масив на дві зростаючі випуклі послідовності, які перетинаються у найменшому елементі масиву.

Логічно будувати ці послідовності, додаючи елементи у порядку зростання. Але як визначити, до якої з них приєднати новий елемент?

Для цього достатньо зберігати по два останні елементи кожної з послідовностей. Це дозволить перевіряти умови випуклості та правильно розподіляти нові елементи.

Тоді можна написати динаміку $dp[i][a][b][c]$ ($i > a$, $i > b > c$), яка буде приймати значення `true` або `false` і позначати, чи можемо розділити перші i чисел на дві послідовності. Перша послідовність закінчується на елементах з індексами i, a, а друга — на b, c. Тоді маємо рішення за $\mathcal{O}(n^4)$.

Звернемо увагу, що в досяжних станах рівно одне з a, b, c буде рівне $i-1$. Тому насправді можна оптимізувати попередню динаміку до $\mathcal{O}(n^3)$.

Крім того, замість бінарних значень `true` та `false` для трійки i, a, b можна зберігати найбільше можливе значення c, оскільки вигідно мінімізувати різницю між b та c. І таким чином матимемо рішення за $\mathcal{O}(n^2)$.

Для того, щоб оптимізувати розв'язок, потрібно ще глибше зануритися в особливості станів. Зокрема, ми вже розглянули, що індекс $i-1$ є важливим у станах. Тепер розглянемо також індекс $i-2$.

Є три випадки станів i, a, b, c:

1. $i, i-1, b, c$;
2. $i, i-2, i-1, c$;
3. $i, c, i-1, i-2$;

Для другого і третього випадку достатньо зберегти лише максимальні можливі досяжні значення c для фіксованого i. Аналогічно маємо для першого випадку для фіксованого i та b, але тут більше параметрів, тому цей випадок є складнішим.

Розглянемо переходи для фіксованого i. Маємо не більше одного стану другого та третього типів. Станів першого типу може бути багато, тому розглянемо їх окремо, щоб оптимізувати переходи з цих станів.

Розглядаємо стан $i, i-1, b, c$. Розглянемо два випадки залежно від того, в яку послідовність ставимо елемент $i+1$:

1. Додаємо в першу підпослідовність. Тоді має виконуватись умова: $a_{i+1} + a_{i-1} \geqslant 2 \cdot a_i$. І переходимо в стан: $i+1, i, c, d$. Помітимо, що в цьому стані перехід відбувається незалежно від значень c, d, і перехід відбувається зі стану першого типу до стану першого типу.

   Тобто якщо для фіксованого i зберігати всі можливі стани як множину пар (c, d), то при переході до наступного стану $i+1$ всі ці стани залишаться в такому ж вигляді, якщо умова $a_{i+1} + a_{i-1} \geqslant 2 \cdot a_i$ виконується.

2. Додаємо в другу підпослідовність. Тоді має виконуватись умова: $d + a_{i+1} \geqslant 2 \cdot c$, що еквівалентно $a_{i+1} \geqslant 2 \cdot c - d$. І переходимо в стан: $i+1, c, i, i-1$, тобто у стан третього типу.

   Для станів третього типу потрібно максимізувати c, тому необхідно знайти максимальне c серед усіх станів, що задовольняють умову $a_{i+1} \geqslant 2 \cdot c - d$.

Переходи зі станів другого і третього типу оптимізувати не треба, бо їх не більше одного.

Тепер залишилося оптимізувати переходи зі станів першого типу до третього. Для цього можна відсортувати стани (c, d) за значенням $2 \cdot c - d$. Після цього для кожного нового елемента a_{i+1} потрібно знаходити максимальне значення c серед тих станів, де $2 \cdot c - d \leqslant a_{i+1}$.

Щоб ефективно знаходити цей максимум, немає необхідності використовувати складні структури даних, такі як дерево відрізків або дерево Фенвіка. Достатньо `std::set`, якщо помітити, що стани, для яких існує інший стан з меншим $2 \cdot c - d$ і водночас більшим c, не потрібно зберігати.

Таким чином, якщо відсіяти невигідні стани, то отримані стани будуть відсортовані як за $2 \cdot c - d$, так і за c. Тоді для знаходження максимуму на префіксі достатньо одного виклику `lower_bound`.

Фінальна складність рішення: $\mathcal{O}(n \log n)$.

Задача C2. Розворот ABC

Автор задачі: **Антон Тригуб**
Задачу підготували: **Роман Піцура**
Розбір написали: **Роман Піцура, Ігор Баренблат**

Блок 1

У цьому блоці відповідь може бути 0 або 1. Щоб його розв'язати, потрібно перевірити, чи можливо виконати хоча б одну операцію. Якщо так — вивести 1, інакше — 0.

Необхідно з'ясувати, чи існує таке `i`, що підрядок $s_i s_{i+1}$ дорівнює одному з трьох: `AB`, `BC` або `CA`.

Блок 2

У цьому блоці потрібно перебрати всі можливі рядки довжини до 3 і побудувати дерево розгалужень.

Блок 3

У цьому блоці рядок складається лише з символів `A` та `B`, отже, можлива тільки одна операція — над підрядком `AB`.

Фінальний вигляд рядка — це всі `B` ліворуч, а всі `A` праворуч: `BB...BBAA...AA`, бо більше неможливо виконати жодну операцію.

Щоб порахувати кількість операцій, для кожного символу `A` потрібно підрахувати кількість `B`, що йдуть після нього, та просумувати для всіх `A`.

Блок 4

Для рядка виду `AA...AABB...BBCC...CC`, нехай n — кількість `A`, m — кількість `B`, k — кількість `C`.

Вигідно або замінити всі `A` та `B`, або всі `B` та `C`.

Відповідь: $\max(n \cdot m, m \cdot k)$.

Блок 5

Погрупуємо однакові підрядки символів у початковому рядку — вони виконуватимуть операції разом.

Якщо підрядок `CA` ніколи не з'являвся, він і не з'явиться після жодної кількості операцій.

Доведення: якщо припустити, що `CA` з'явився, то до цієї операції рядок мав містити `CBA` або `AC`, але з жодного з цих рядків не утворюється `CA`.

Можливі лише операції над `AB` та `BC`. Для кожного символу `B` розглянемо такі випадки:

1. `B` має зліва лише `A`: взаємодіє з усіма `A` до першого `C` зліва.
2. `B` має справа лише `C`: взаємодіє з усіма `C` до першого `A` справа.
3. `B` має `A` зліва і `C` справа: обираємо максимум з випадків 1 і 2.

Блок 6

Якщо довжина рядка $\leqslant 12$, можна запускати повну рекурсію.

Якщо операції виконати неможливо — відповідь 0.

Інакше перебираємо всі можливі операції, запускаємо рекурсію для кожного варіанту і вибираємо максимальну відповідь.

Складність: $O(n \cdot 3^n)$, оскільки кількість станів — 3^n, і з кожного стану — до n переходів.

Блок 7

Щоб розв'язати цей блок, попередньо порахуємо відповідь для всіх 3^n станів (як у блоці 6).

Після цього на кожен запит відповідатимемо за $O(1)$.

Блок 9

Погрупуємо однакові підрядки символів у початковому рядку — вони діятимуть разом.

Результуючий рядок матиме вигляд `...ACBACBACB...`.

Введемо *dp*[перший символ][суфікс] — відповідь для суфікса, якщо перший символ у результаті — вказаний.

Тоді шукаємо $\max(dp['A'][0], dp['B'][0], dp['C'][0])$.

Переходи в dp, для прикладу $first\_symbol = \texttt{A}$ (аналогічно для інших):

- Якщо $s[i] = \texttt{A}$, то $dp[\texttt{A}][i] = \max(dp[\texttt{A}][i+1], dp[\texttt{C}][i+1])$ (після A не може бути B). - Якщо $s[i] = \texttt{B}$, то $dp[\texttt{A}][i] = -\infty$. - Якщо $s[i] = \texttt{C}$: - Якщо далі йде B або кінець рядка, то $dp[\texttt{A}][i] = -\infty$. - Якщо рядок виглядає як `CACACACA[C]`, то переміщуємо всі A ліворуч:

$dp[\texttt{A}][i]$ = value_of_swapping_all_A_to_left

- Якщо рядок виглядає як `CACACACACB...`:

$dp[\texttt{A}][i]$ = value_of_swapping_all_A_to_left + $dp[\texttt{B}][B...]$

- Якщо рядок має вигляд `CACACACAB...`, то:

$dp[\texttt{A}][i]$ = max(value_of_swapping_all_A_to_left + $dp[\texttt{B}][B...]$, value_of_swapping_bold_A_to_left + $dp[\texttt{B}][AB...]$)

Задача D1. Проста підпослідовність

Автор задачі: **Антон Тригуб**
Задачу підготували: **Андрій Столітній, Костянтин Денисов**
Розбір написав: **Костянтин Луценко**

Блок 1

```
Ans = R - L + 1 - (L%2==0) - (R%2==0) + ((L==R)&&(L%2==0))
```

Блок 2

Можна перебрати всі 2^n підпослідовностей і перевірити, чи вони є хорошими, порахувавши всі префіксні і суфіксні суми. Можна для кожного з $\frac{n(n+1)}{2}$ можливих запитів порахувати відповідь, перебравши всі її підпослідовності. Складність $\mathcal{O}(2^n \cdot n)$.

Блок 3

Нехай $\sum[L, R]$ означає суму на підвідрізку $[L, R]$. Скажемо, що $\sum[L, R] \stackrel{def}{=} 0$ при $L > R$ (порожній відрізок).

Нехай $\sum[1, n] = S$.

Визначення хорошого масиву переписується як $\sum[1, a] \geqslant 0$, $\sum[a, n] \geqslant 0$. Але $\sum[a, n] = \sum[1, n] - \sum[1, a-1]$. Тому визначення хорошого масиву можна записати як

$$\sum[1, a] \in [0, S]$$

Тому можна обрахувати `dp[позиція][поточна сума][максимальна префіксна сума яка зустрічалася]`, де значення динаміки — це максимальна кількість елементів (або `-INF`, якщо стан недосяжний). Складність $\mathcal{O}(qn^3)$.

Також помітимо, що оптимальна відповідь отримується з заданого відрізка видаленням деякої кількості мінус одиничок. Наша мета — з'ясувати, скільки мінус одиничок ми маємо видалити.

Блок *

Твердження. Масив є хорошим тоді і тільки тоді, коли всі суми його підвідрізків є не більшими за суму всього масиву.

Доведення.

$\Leftarrow$: Від супротивного, припустимо, всі відрізки мають суму, меншу за S, але масив не є хорошим. Через те, що масив не є хорошим, то існує або його префікс, або суфікс, з від'ємною сумою. Без обмеження загальності, нехай це префікс, тобто існує R таке, що $\sum[1, R] < 0$. Але тоді $\sum[R+1, n] = S - \sum[1, R] > S$, тобто відрізок $[1, R]$ має суму більшу за суму всього масиву.

$\Rightarrow$: Від супротивного, припустимо, що масив є хорошим, але існує $\sum[L, R] > \sum[1, n]$. Тому $\sum[1, n] - \sum[L, R] = \sum[1, L-1] + \sum[R+1, n] < 0$, тому хоча б одне з чисел $\sum[1, L-1]$ або $\sum[R+1, n]$ є від'ємним, бо їх сума від'ємна. Отже, масив не є хорошим.

Повернемось до задачі — потрібно визначити мінімальну кількість «-1», яку потрібно видалити, щоб суми на всіх підвідрізках не перевищували суму на всьому масиві.

Нехай $\sum[1, n] = S_0$. Нехай $[L, R]$ — це підвідрізок з максимальною сумою (якщо таких декілька, то довільний з них). Нехай $\sum[L, R] = M_0$. Хочемо зробити, щоб $S = M$, але спочатку маємо $M_0 \geqslant S_0$.

За одне видалення мінус одинички S збільшується на 1, а M не зменшується. Отже, різниця $M - S$ зменшується не більше, ніж на 1. Отже, ми маємо видалити щонайменше $M_0 - S_0$ елементів.

Доведемо, що $M_0 - S_0$ елементів видалити завжди достатньо. Нехай $\sum[1, L-1] = -P$ і $\sum[R+1, n] = -Q$. Помітимо, що $M_0 - S_0 = \sum[L, R] - \sum[1, n] = -(\sum[1, L-1] + \sum[R+1, n]) = P + Q$. Помітимо також, що $P, Q \leqslant 0$ (тобто $\sum[1, L-1] \leqslant 0$ та $\sum[R+1, n] \leqslant 0$), бо інакше $[L, R]$ можна розширити зі збільшенням суми.

Нехай $\sum'[L, R]$ — це сума на підвідрізку після нашого видалення. Ми хочемо зробити, щоб тепер всі суми $\sum'[1, a]$ та $\sum'[a, n]$ були невід'ємнии, видаливши $P + Q$ мінус одиничок.

Для цього видалимо на префіксі $[1, L-1]$ деякі P мінус одиничок, а на суфіксі $[R+1, n]$ деякі Q мінус одиничок. Які саме — з'ясуємо пізніше. Тоді $\sum'[1, L-1] = \sum[1, L-1] + P = -P + P = 0$, аналогічно $\sum'[R+1, n] = 0$.

Помітимо, що $\sum[a+1, L-1] \leqslant 0$, бо інакше $\sum[a+1, R] = \sum[L, R] + \sum[a+1, L-1] > \sum[L, R]$, а ми позначили $[L, R]$ відрізок з найбільшою сумою.

Помітимо, що $\sum[L, a] \geqslant 0$, бо інакше $\sum[a+1, R] = \sum[L, R] - \sum[L, a] > \sum[L, R]$, а ми позначили $[L, R]$ відрізок з найбільшою сумою.

Визначимо, які додаткові умови необхідні для того, щоб отриманий масив був хорошим:

1. $a < L$:

   З'являється умова $\sum'[1, a] \geqslant 0$.

2. $L \leqslant a \leqslant R$:

$$\sum'[1, a] = \sum'[1, L] + \sum'[L, a] = 0 + \sum'[L, a] \geqslant 0,$$

   тобто це виконується автоматично.

3. $a > R$:

$$\begin{gathered}\sum'[1, a] = \sum'[1, L-1] + \sum'[L, R] + \sum'[R+1, a] = \\ = 0 + M_0 + \sum'[R+1, a] = M_0 + \sum'[R+1, n] - \sum'[a+1, n] = \\ = M_0 + 0 - \sum'[a+1, n],\end{gathered}$$

   тобто з'являється умова $\sum'[a+1, n] \leqslant M_0$.

Отже, необхідно і достатньо, щоб для всіх $a < L$ виконувалось $\sum[1, a] \in [0, M_0]$, а також аналогічна умова на суфікси, тобто $\sum[a, n] \in [0, M_0]$ для всіх $a > R$. Цього, наприклад, можна добитися, якщо видаляти «-1» жадібно: йдемо по $[1..a]$ збільшуючи a від 0 до $L-1$ і, якщо префіксна сума стала від'ємна, то видаляємо поточну мінус одиницю. Тоді всі префіксні суми стануть невід'ємні. Припустимо, що при видаленні одиниці на позиції a виявилось, що $\sum'[1, b]$ стала більшою за $M_0 + 1$ для деякого $b < L$. Помітимо, що в цей момент змінились тільки префіксні суми, що правіше за a, тобто $b \geqslant a$. Але $\sum'[1, a]$ у цей момент стала рівною 0. Тому $\sum'[a+1, b] = \sum'[1, b] - \sum'[1, a] = M_0 + 1$. Але на цьому відрізку ми нічого не змінювали, тобто

$$\sum'[a+1, b] = \sum[a, b] = M_0 + 1 > M_0 = \sum[L, R].$$

Але ми обирали M_0 як максимальну суму на підвідрізку. Тому такого не може бути, а значить, всі префіксні суми лежать у потрібних межах. Аналогічно робимо з суфіксами.

Також варто зауважити, що ми дійсно видалили рівно P мінус одиничок, тобто що сума на всьому префіксі стала рівною нулеві. Дійсно, якщо при видаленні мінус одиничка з позиції a виявилось, що $\sum'[1, L-1] > 0$, то $\sum[a+1, L-1] = \sum'[a+1, L-1] = \sum'[1, L-1] - \sum'[1, a] = \sum'[1, L-1] - 0 > 0$, а ми доводили, що суми на відрізках виду $[a, L-1]$ завжди невід'ємні.

Отже, ми звели задачу до пошуку M_0, S_0 і виведення на екран величини $R - L + 1 - (M_0 - S_0)$.

Блок 4

В розборі цього і наступних блоків описано, як знайти $M = \max\limits_{l \leqslant L \leqslant R \leqslant r} \sum[L, R]$. Як показано вище, це розв'яже задачу.

Переберемо R. Маємо обрати L таке, що $\sum[L, R]$ максимальне, тобто $\sum[1, L-1]$ мінімальне. Можемо підтримувати це L одночасно з перебором R.

Складність: $\mathcal{O}(qn)$

Блоки 5-6

Нехай $node[L, R]$ — це четвірка значень:

- S — сума на цьому відрізку;

- A — максимальна сума префіксу цього відрізку;
- B — максимальна сума суфікса цього відрізку;
- M — максимальна сума деякого підвідрізку цього відрізку.

Тоді можна рахувати $node[L, R]$ через $node[L, a]$ та $node[a + 1, R]$ для деякого a: нехай

$$(S_1, A_1, B_1, M_1) + (S_2, A_2, B_2, M_2) = (S, A, M, B),$$

тоді:

- $S = S_1 + S_2$;
- $A = \max(A_1, S_1 + A_2)$;
- $B = \max(B_2, S_2 + B_1)$;
- $M = \max(M_1, M_2, B_1 + A_2)$.

Якщо ви дуже хочете здати ці блоки, але не здати наступний, можна використати sqrt-декомпозицію за $\mathcal{O}(q\sqrt{n})$. Для цього слід розділити масив на блоки довжини C (де $C \sim \sqrt{n}$) і підрахувати $node[kC, (k + 1)C - 1]$. Далі для кожного запиту пошуку відповіді можна виконувати $\mathcal{O}(n/C)$ описаних вище злиттів, а для запитів оновлення потрібно перераховувати суму для зміненого блоку за $\mathcal{O}(C)$.

Блок 7

Описане в попередньому блоці злиття можна обраховувати в дереві відрізків, у вершинах якого будуть зберігатись числа S, A, B, M для відповідних відрізків. Запит `get` також буде використовувати дане злиття і поверне четвірку чисел, з якої ми візьмемо M та S і виведемо $R - L + 1 - (M - S)$.

Задача D2. Проста задача

Автори задачі: **Ігор Баренблат, Матвій Асландуков**
Задачу підготували: **Ігор Баренблат, Матвій Асландуков**
Розбір написав: **Матвій Асландуков**

Введемо наступні позначення:

- $x = \max\limits_{i=1}^{n} a_i$;
- $N = \sum\limits_{i=1}^{T} n_i$, де n_i — значення n в i-му наборі вхідних даних;
- $X = \max\limits_{i=1}^{T} x_i$, де x_i — значення x в i-му наборі вхідних даних;
- ans — мінімально можлива кількість *дивних* підпослідовностей серед k утворених.

Блок 1 ($k = 1$).

Існує єдиний спосіб розбити масив a на одну підпослідовність. Щоб знайти кількість *дивних* підпослідовностей серед однієї утвореної, достатньо перевірити, чи є сума елементів масиву a простим числом — це можна зробити стандартним алгоритмом за $O(\sqrt{s})$, де $s = \sum_{i=1}^{n} a_i \leqslant nx$.

Складність обробки одного набору вхідних даних — $O(n + \sqrt{nx})$.

Складність обробки всіх наборів вхідних даних — $O(N + N\sqrt{X})$.

Блоки 2, 3, 4, 5, 6, 7, 8.

Для розв'язку наступних блоків доведемо дві допоміжні леми.

Лема 1.

Твердження. Нехай $q = n - k$. Розглянемо наступний процес:

- Створимо n підпослідовностей, де i-та початково складається з одного елементу a_i.
- Виконаємо q операцій «об'єднання»: під час кожної операції виберемо дві певні підпослідовності, та об'єднаємо їх в одну.

Тоді задача розбиття масиву на k непорожніх підпослідовностей еквівалентна задачі об'єднання n одноелементних підпослідовностей за допомогою q описаних вище операцій.

Доведення. Дійсно, при виконанні однієї операції, кількість підпослідовностей зменшується на 1, а отже після виконання q операцій, кількість підпослідовностей буде дорівнювати $n - q = n - n + k = k$. Також легко помітити, що будь-яке розбиття можна утворити за допомогою q описаних вище операцій: для цього під час виконання чергової операції потрібно вибрати дві довільні підпослідовності, елементи яких належать одній у відповідному розбитті.

Лема 2.

Твердження. $ans \geqslant n - 2q$.

Доведення. Дійсно, в описаному вище процесі початково існувало n *дивних* підпослідовностей, а кожна операція «об'єднання» зменшувала кількість *дивних* підпослідовностей не більше ніж на 2. Тому після виконання будь-яких q операцій, все ще мало б залишитися хоча б $(n - 2q)$ *дивних* підпослідовностей.

Блок 6 ($2k \geqslant n + 1$).

Оскільки $k = n - q$, то $2 \cdot (n - q) \geqslant n + 1 \Rightarrow n - 2q \geqslant 1$. В такому випадку завжди можна утворити відповідь рівно з $(n - 2q)$ *дивними* підпослідовностями, що є оптимальною відповіддю згідно з другою лемою. А саме, під час виконання чергової операції «об'єднання», виберемо дві одноелементні підпослідовності з однаковою парністю елементів. Неважко помітити, що це завжди можна буде зробити:

- для непарного n, існує парна кількість або парних, або непарних елементів, а тому максимум один елемент може залишитися без пари, що відповідає умові $(n - 2q) \geqslant 1$.

- для парного n, максимум два елементи можуть залишитися без пари (один парний, інший непарний), що відповідає умові $(n - 2q) \geqslant 2$, яка виконується, оскільки $(n - 2q)$ є парним числом та $(n - 2q) \geqslant 1$.

Оскільки сума двох чисел з однаковою парністю є парним числом більшим за два, то у результаті «об'єднання» замість двох *дивних* підпослідовностей утвориться одна *недивна*. Таким чином, кожна операція буде зменшувати кількість *дивних* підпослідовностей на 2, що в підсумку зробить $ans = n - 2q$.

Складність обробки всіх наборів вхідних даних — $O(N)$.

Блоки 2, 3, 7.

Перед обробкою всіх наборів вхідних даних підрахуємо допоміжний масив `isPrime[1..2X]` (`isPrime[1..10X]` у третьому блоці), де `isPrime[i]` позначає, чи є число i простим. Це можна зробити за $O(X \log X)$ за допомогою алгоритму «Решето Ератосфена».

Блок 2 ($n \leqslant 4$).

При наявності рішення на перший та шостий блоки, потрібно розглянути тільки випадок $n = 4, k = 2$, оскільки для всіх інших випадків виконується хоча б одна з умов $2k \geqslant n + 1$, $k = 1$.

При $n = 4, k = 2$ достатньо розглянути лише випадки, де обидві підпослідовності складаються з двох елементів. Це можна зробити, перебравши один з трьох елементів, який буде в підпослідовності разом з $a[1]$. Якщо в якомусь з трьох випадків немає *дивних* підпослідовностей (це можна перевірити за допомогою масиву `isPrime`), то $ans = 0$. Інакше серед цих випадків треба знайти будь-який з однією *дивною* підпослідовністю. Такий завжди знайдеться, оскільки в масиві довжини 4 існує пара елементів однакової парності (а отже їхня сума не є простим числом). Саме тому не потрібно розглядати випадки, де одна підпослідовність складається з одного елементу, а інша — з трьох, оскільки в них гарантовано буде хоча б одна *дивна* підпослідовність (яка складається з одного елементу, що є простим числом).

Не маючи рішень на перший та шостий блоки, можна окремо розглянути усі можливі комбінації значень n, k, та вирішити задачу для кожної з них окремо, перебравши усі необхідні варіанти утворення підпослідовностей.

Складність обробки всіх наборів вхідних даних — $O(X \log X + N)$.

Блок 3 ($T \leqslant 20, n \leqslant 10$).

У цьому блоці достатньо було перебрати всі можливі розбиття масиву a на k підпослідовностей. А саме, будь-яке розбиття можна задати за допомогою масиву ids довжини n, де $ids[i]$ позначає номер підпослідовності, до якої належить $a[i]$.

Щоб розбиття однозначно задавалося масивом ids, достатньо розглядати лише такі ids, де $ids[1] = 1$ та $1 \leqslant ids[i] \leqslant (\max\limits_{j=1}^{i-1} ids[j] + 1)$ при $i > 1$. Дійсно, оскільки порядок підпослідовностей не має значення, без втрати загальності можна вважати $ids[1] = 1$, а для кожного наступного i, i-й елемент або буде входити до підпослідовності, в якій вже зустрічався якийсь попередній елемент масиву a (тоді $1 \leqslant ids[i] \leqslant \max\limits_{j=1}^{i-1} ids[j]$), або буде найлівішим елементом нової підпослідовності (тоді без втрати загальності можна вважати $ids[i] = \max\limits_{j=1}^{i-1} ids[j] + 1$).

Усі розбиття можна перебрати за допомогою рекурсивної генерації масиву ids. Наприклад, наступний код на `Python3` перебирає та рахує кількість розбиттів масиву довжини 10:

```
def countPartitions(ids):
    if len(ids) == 10:
        return 1
    return sum(countPartitions(ids + [id]) for id in range(1, max(ids) + 2))
print(countPartitions([1]))
```

Експериментальним шляхом можна побачити, що при $n = 10$ кількість розбиттів $C = 115975$. Для кожного з них із загальною кількістю підпослідовностей k у розбитті, можна підрахувати кількість *дивних* підпослідовностей за допомогою масиву `isPrime`, та вибрати розбиття з мінімальною кількістю.

Складність обробки всіх наборів вхідних даних — $O(X \log X + C \cdot N)$.

Блоки 4, 5, 7, 8.

Для цих блоків буде наведено рішення для $1 < k$, $5 \leqslant n$ та $2k < n + 1 \Rightarrow 2k \leqslant n \Rightarrow n \leqslant 2q$. При невиконанні цих умов, дивіться рішення для блоків 1, 2 та 6.

Блок 4 (n — парне, $a_i > 2$).

Оскільки всі $a_i > 2$, то всі елементи масиву a — непарні. Оскільки $q \geqslant \frac{n}{2}$, то спочатку розіб'ємо усі елементи по парам, тим самим виконавши $\frac{n}{2}$ операцій «об'єднання», та зробивши всі підпослідвності *недивними*, оскільки суми їхніх елементів будуть парними числами більше за два. Після цього залишиться виконати $(q - \frac{n}{2})$ операцій «об'єднання», що можна зробити довільним чином, оскільки сума будь-яких двох парних чисел є парним числом. Кількість *дивних* підпослідовностей у результаті буде дорівнювати нулю, що очевидно є мінімально можливою кількістю.

Складність обробки всіх наборів вхідних даних — $O(N)$.

Блоки 5, 7, 8.

Для розв'язку наступних блоків доведемо допоміжну лему.

Лема 3.

Твердження. Серед п'яти простих чисел завжди можна знайти три числа, сума яких ділиться на 3 та не є простим числом.

Доведення. Помітимо, що сума будь-яких трьох простих чисел перевищує 3, а отже ця сума не є простим числом у випадку, якщо вона ділиться на 3. Тепер доведемо, що можна знайти три числа, сума яких ділиться на 3. Припустимо, що це не так. Розглянемо залишки від ділення на 3 всіх заданих чисел. Якщо якийсь залишок зустрічається хоча б 3 рази, то сума трьох відповідних чисел буде ділитися на 3, що по припущенню неможливо. Також якщо усі залишки (0, 1, 2) зустрічаються хоча б один раз, то сума трьох відповідних чисел буде ділитися на 3, що по припущенню неможливо. Це означає, що у масиві залишків зустрічається лише два різних числа, кожне з яких зустрічається максимум два рази, а тому загальна довжина масиву не перевищує 4. Отримали протиріччя, оскільки лема доводилася для п'яти чисел.

Блок 5 (n — непарне, $a_i > 2$).

Оскільки q — ціле число, то для непарного n нерівність $q \geqslant \frac{n}{2}$ можна посилити: $q \geqslant \frac{n+1}{2}$. Оскільки $n \geqslant 5$, то згідно з третьою лемою серед перших п'яти елементів масиву a можна знайти трійку з сумою, що не ділиться на 3. Знайдемо таку трійку шляхом повного перебору всіх $C_5^3 = 10$ трійок, та об'єднаємо відповідні елементи в одну підпослідовність, виконавши дві операції «об'єднання». Інші $(n - 3)$ елементи розіб'ємо по парам довільним чином, виконавши $\frac{n-3}{2}$ операції «об'єднання». Після цих $(2 + \frac{n-3}{2} = \frac{n+1}{2})$ операцій «об'єднання» усі підпослідовністі будуть *недивними*, і тільки перша матиме непарну суму елементів.

Оскільки $k > 1$, то маючи лише одну *недивну* підпослідовність з непарною сумою та решту *недивних* підпослідовностей з парною, фінальні $c = (q - \frac{n+1}{2})$ операцій «об'єднання» можна витратити на об'єднання $(c+1)$ підпослідовностей з парною сумою в одну підпослідовність також з парною сумою. Кількість *дивних* підпослідовностей у результаті буде дорівнювати нулю, що очевидно є мінімально можливою кількістю.

Складність обробки всіх наборів вхідних даних — $O(N)$.

Блок 8 (n — непарне).

Аналогічно п'ятому блоку, тут буде виконуватися нерівність $q \geqslant \frac{n+1}{2}$.

Поступово розглянемо всі можливі «типи» масиву a:

1. Кількість двійок в масиві a — непарна.

Тоді кількість непарних елементів в масиві a — парна. Залишимо одну двійку в окремій одноелементній підпослідовності та використаємо $\frac{n-1}{2}$ операцій «об'єднання» для розбиття решти $(n-1)$ елементів масиву a на пари: парні елементи з парними, непарні — з непарними. Використаємо додаткову операцію «об'єднання», щоб об'єднати одноелементну підпослідовність з двійкою з будь-якою іншою підпослідовністю. Після цих $(1+\frac{n-1}{2} = \frac{n+1}{2})$ операцій «об'єднання» усі підпослідовності будуть *недивними* та матимуть парну суму елементів. Фінальні $(q - \frac{n+1}{2})$ операцій «об'єднання» можна зробити аналогічно фінальній частині розв'язку п'ятого блоку, отримавши нульову кількість *дивних* підпослідовностей у результаті.

2. Кількість двійок в масиві a — парна.

   Тоді кількість непарних елементів в масиві a — непарна. Спробуємо знайти трійку непарних елементів з сумою, що ділиться на 3 (це можна зробити так само, як у блоці 5):

   - Якщо така трійка знайшлася, то об'єднаємо відповідні елементи в одну підпослідовність, виконавши дві операції «об'єднання». Виконаємо $\frac{n-3}{2}$ операцій «об'єднання» для розбиття решти $(n-3)$ елементів масиву a на пари: парні елементи з парними, непарні — з непарними. Після цих $(2 + \frac{n-3}{2} = \frac{n+1}{2})$ операцій «об'єднання» усі підпослідовності будуть *недивними*, і тільки перша матиме непарну суму елементів. Фінальні $(q - \frac{n+1}{2})$ операцій «об'єднання» можна зробити аналогічно фінальній частині розв'язку п'ятого блоку, отримавши нульову кількість *дивних* підпослідовностей у результаті.
   - Якщо така трійка не знайшлася, то з третьої леми випливає, що кількість непарних елементів не перевищує 3, а отже кількість двійок хоча б 2. Розглянемо два випадки:
     (a) Серед непарних елементів існує якийсь елемент $x \neq 3$. Залишимо x та дві двійки в окремих одноелементних підпослідовностях. Виконаємо $\frac{n-3}{2}$ операцій «об'єднання» для розбиття решти $(n-3)$ елементів масиву a на пари: парні елементи з парними, непарні — з непарними. Оскільки x — просте число більше за три, x може мати лише два можливих залишка при діленні на 3:
        i. $x \mod 3 = 2$, тобто $(x+4)$ ділиться на 3. Виконаємо дві операції «об'єднання» одноелементних підпослідовностей в одну: $[x, 2, 2]$.
        ii. $x \mod 3 = 1$, тобто $(x+2)$ ділиться на 3. Виконаємо одну операцію «об'єднання», щоб створити двоелементну підпослідовність: $[x, 2]$, а другу операцію, щоб об'єднати останню двійку з будь-якою іншою з перших $\frac{n-3}{2}$ пар.

     Після цих $(2 + \frac{n-3}{2} = \frac{n+1}{2})$ операцій «об'єднання» усі підпослідовності будуть *недивними*, і тільки одна матиме непарну суму елементів. Фінальні $(q - \frac{n+1}{2})$ операцій «об'єднання» можна зробити аналогічно фінальній частині розв'язку п'ятого блоку, отримавши нульову кількість *дивних* підпослідовностей у результаті.

     (b) Всі непарні елементи дорівнюють трьом. Це означає, що в масиві a існує лише один непарний елемент, оскільки інакше б на попередньому кроці знайшлася трійка з сумою, що ділиться на 3 $((3+3+3) \mod 3 = 0)$. Іншими словами, $a = [2, 2, \ldots, 2, 2, 3]$. Розглянемо два випадки:
        i. $q = \frac{n+1}{2}$. В такому випадку хоча б одна з утворених k підпослідовностей буде *дивною*: або підпослідовність з трійкою ($[3]$, $[3, 2]$, $[3, 2, 2]$), або одноелементна підпослідовність $[2]$ (така завжди знайдеться, якщо підпослідовність з трійкою складається хоча б з чотирьох елементів).
        Виконаємо $\frac{n-1}{2}$ операцій «об'єднання» для розбиття всіх $(n-1)$ двійок на пари, та ще одну операцію, щоб об'єднати $[2, 2]$ та $[3]$. Отримаємо розбиття з однією дивною підпослідовністю.
        ii. $q > \frac{n+1}{2} \Rightarrow q \geqslant \frac{n+3}{2}$. В такому випадку $n \geqslant 7$, оскільки при $n = 5$ єдине можливе значення $q = \frac{n+3}{2} + 1 = 4$, що вже розглянуто окремо у випадку $k = n - q = 5 - 4 = 1$.
        Об'єднаємо останні чотири елементи в підпослідовність $[2, 2, 2, 3]$, виконавши три операції «об'єднання». Об'єднаємо три двійки в підпослідовність $[2, 2, 2]$, виконавши дві операції «об'єднання». Виконаємо $\frac{n-7}{2}$ операцій «об'єднання» для розбиття

решти $(n-7)$ двійок на пари. Після цих $(5+\frac{n-7}{2}=\frac{n+3}{2})$ операцій «об'єднання» усі підпослідовності будуть *недивними*, і тільки одна з них матиме непарну суму елементів. Фінальні $(q-\frac{n+3}{2})$ операцій «об'єднання» можна зробити аналогічно фінальній частині розв'язку п'ятого блоку, отримавши нульову кількість *дивних* підпослідовностей у результаті.

Складність обробки всіх наборів вхідних даних — $O(N)$.

Блок 7 (n — парне).

Нагадаємо, що тут розглядається розв'язок для $1 < k, 5 \leqslant n, q \geqslant \frac{n}{2}$. Оскільки n — парне, нерівність $n \geqslant 5$ можна посилити: $n \geqslant 6$.

Поступово розглянемо всі можливі «типи» масиву a:

1. Кількість двійок в масиві a — парна.

   Тоді кількість непарних елементів в масиві a — парна. Використаємо $\frac{n}{2}$ операцій «об'єднання» для розбиття усіх n елементів масиву a на пари: парні елементи з парними, непарні — з непарними. Після цих $\frac{n}{2}$ операцій «об'єднання» усі підпослідовності будуть *недивними* та матимуть парну суму елементів. Фінальні $(q-\frac{n}{2})$ операцій «об'єднання» можна зробити аналогічно фінальній частині розв'язку п'ятого блоку, отримавши нульову кількість *дивних* підпослідовностей у результаті.

2. Кількість двійок в масиві a — непарна.

   Тоді кількість непарних елементів в масиві a — непарна. Спробуємо знайти непарний елемент x такий, що $(x+2)$ не є простим числом (це можна перевірити за допомогою `isPrime[x + 2]`):

   - Якщо такий елемент знайшовся, то об'єднаємо його з двійкою в одну підпослідовність $[x, 2]$, виконавши одну операцію «об'єднання». Виконаємо $\frac{n-2}{2}$ операцій «об'єднання» для розбиття решти $(n-2)$ елементів масиву a на пари: парні елементи з парними, непарні — з непарними. Після цих $(1+\frac{n-2}{2}=\frac{n}{2})$ операцій «об'єднання» усі підпослідовності будуть *недивними*, і тільки перша матиме непарну суму елементів. Фінальні $(q-\frac{n}{2})$ операцій «об'єднання» можна зробити аналогічно фінальній частині розв'язку п'ятого блоку, отримавши нульову кількість *дивних* підпослідовностей у результаті.
   - Якщо такого елементу не знайшлося, це означає, що для всіх непарних елементів x виконується $(x+2) \mod 3 \neq 0 \Rightarrow x \mod 3 \neq 1$. Розглянемо два випадки:
     (a) $q = \frac{n}{2}$. В такому випадку хоча б одна з утворених k підпослідовностей буде *дивною*: або одноелементна підпослідовність (яка зажди є *дивною*), або підпослідовність $[2, x]$, де $x \neq 2$ (така завжди знайдеться, оскільки двійок непарна кількість, а тому хоча б одна двійка мала б об'єднатися з непарним елементом x).
     Об'єднаємо двійку з будь-яким непарним елементом, та виконаємо ще $\frac{n-2}{2}$ операцій «об'єднання» для розбиття решти $(n-2)$ елементів масиву a на пари: парні елементи з парними, непарні — з непарними. Отримаємо розбиття з однією дивною підпослідовністю.
     (b) $q > \frac{n}{2} \Rightarrow q \geqslant \frac{n+2}{2}$. Спочатку виконаємо перші $\frac{n+2}{2}$ операцій «об'єднання» в залежності від «типу» масиву a:
       i. В масиві a існує хоча б три двійки та одна трійка. Об'єднаємо ці чотири елементи в підпослідовність $[2, 2, 2, 3]$, виконавши три операції «об'єднання». Виконаємо $\frac{n-4}{2}$ операцій «об'єднання» для розбиття решти $(n-4)$ елементів масиву a на пари: парні елементи з парними, непарні — з непарними. Кількість виконаних операцій «об'єднання» — $(3+\frac{n-4}{2}=\frac{n+2}{2})$.
       ii. В масиві a існує хоча б п'ять двійок. В такому випадку для кожного непарного елементу x виконується $x \mod 3 = 2$ (інакше б розглянувся попередній випадок). Об'єднаємо дві двійки та будь-який непарний елемент x в підпослідовність $[2, 2, x]$, виконавши дві операції «об'єднання». Об'єднаємо три двійки в підпослідовність $[2, 2, 2]$, виконавши дві операції «об'єднання». Виконаємо $\frac{n-6}{2}$ операцій

«об'єднання» для розбиття решти $(n-6)$ елементів масиву a на пари: парні елементи з парними, непарні — з непарними. Кількість виконаних операцій «об'єднання» — $(2 + 2 + \frac{n-6}{2} = \frac{n+2}{2})$.

iii. В масиві a існує рівно три двійки. Оскільки $n \geqslant 6$, то в масиві a існує хоча б три непарні елементи. Аналогічно попередньому випадку, для кожного непарного елементу x виконується $x \mod 3 = 2$.
Об'єднаємо три двійки в підпослідовність $[2, 2, 2]$, виконавши дві операції «об'єднання». Об'єднаємо три довільні непарні елементи x, y, z в підпослідовність $[x, y, z]$, виконавши дві операції «об'єднання» (їхня сума буде ділитися на 3, оскільки $x \mod 3 = y \mod 3 = z \mod 3 = 2$). Виконаємо $\frac{n-6}{2}$ операцій «об'єднання» для розбиття решти $(n-6)$ елементів масиву a на пари: парні елементи з парними, непарні — з непарними. Кількість виконаних операцій «об'єднання» — $(2 + 2 + \frac{n-6}{2} = \frac{n+2}{2})$.

iv. В масиві a існує рівно одна двійка. Оскільки $n \geqslant 6$, то в масиві a існує хоча б п'ять непарних елементів. Згідно з третьою лемою, серед них можна знайти три елементи, сума яких ділиться на 3.
Знайдемо їх шляхом повного перебору всіх $C_5^3 = 10$ трійок, та об'єднаємо відповідні елементи в одну підпослідовність, виконавши дві операції «об'єднання».
Об'єднаємо двійку з двома іншими непарними елементами x, y в підпослідовність $[2, x, y]$, виконавши дві операції «об'єднання». Виконаємо $\frac{n-6}{2}$ операцій «об'єднання» для розбиття решти $(n-6)$ елементів масиву a на пари: парні елементи з парними, непарні — з непарними. Кількість виконаних операцій «об'єднання» — $(2 + 2 + \frac{n-6}{2} = \frac{n+2}{2})$.

Після цих $\frac{n+2}{2}$ операцій «об'єднання» усі підпослідовності будуть *недивними*, і тільки одна з них матиме непарну суму елементів. Фінальні $(q - \frac{n+2}{2})$ операцій «об'єднання» можна зробити аналогічно фінальній частині розв'язку п'ятого блоку, отримавши нульову кількість *дивних* підпослідовностей у результаті.

Складність обробки всіх наборів вхідних даних — $O(X \log X + N)$.

Авторська реалізація

Для кращого розуміння описаного вище розв'язку, його авторську реалізацію на `Python3` можна знайти за посиланням `https://ideone.com/owiZT3`.