A. Капсула для вакцини

Потрібно знайти найменший з вимірів ящика. Це і буде діаметром максимальної сфери, яка поміститься.

Складність розв'язку — $O(1)$.

Код розв'язку C++

Код розв'язку Python

B. Народна вакцина

Нам задано перший та тринадцятий члени арифметичної прогресії. З означення арифметичної прогресії, ми знаємо, що $A_k = A_1 + (k-1) \cdot d$, де A_i - i-тий член арифметичної прогресії, а d її різниця. З цієї формули випливає, що $d = \frac{A_k - A_1}{k-1}$, що для $k = 13$ дає $d = \frac{A_{13} - A_1}{12}$.

Зрозуміло, що усі члени арифметичної прогресії є цілими лише у випадку, якщо перший член і різниця цієї прогресії є цілими. Умова гарантує, що перший член є цілим, а тому нам треба перевірити лише різницю. Тобто, перевірити чи $B - A$ ділиться без остачі на 12.

Якщо різниця є цілою (зауважте, що вона може бути й від'ємною, це ні на що не впливає), то нам потрібно знайти суму перших тринадцяти членів цієї прогресії. Загальновідома формула для суми перших k членів — $S_k = \frac{(A_1 + A_k) \cdot k}{2}$, а тому відповідь на задачу це $\frac{(A+B) \cdot 13}{2}$.

Складність розв'язку — $O(1)$.

Код розв'язку C++

Код розв'язку Python

C. Аморальні експерименти

Усіх людей за столом можна розділити на проміжки (тут проміжком називається множина людей, що сидять підряд), такі що кожен проміжок є або повністю зараженим, або повністю не зараженим. Зауважте, що оскільки стіл круглий, то перший учасник є сусідом останнього. Наприклад, стіл з восьми учасників $\{0\ 1\ 1\ 0\ 0\ 0\ 1\ 0\}$ ділиться на 4 проміжки:

- незаражений проміжок довжини 2, що включає першого та останнього учасника,
- заражений проміжок довжини 2, що включає другого та третього учасника,
- незаражений проміжок довжини 3, що включає учасників з четвертого по шостого,
- заражений проміжок довжини один, що включає лише сьомого учасника.

.

Зрозуміло, що кожен незаражений проміжок ставатиме коротшим на два кожної хвилини, оскільки два крайні учасники заразяться від сусідніх заражених. Тобто, проміжок довжини k повністю заразиться за $\lceil \frac{k}{2} \rceil$ хвилин. Усі учасники за столом захворіють, як тільки повністю заразиться найдовший з незаражених проміжків.

Отже, все що потрібно зробити — це знайти довжину найдовшого незараженого проміжку за круглим столом. Один зі способів це зробити такий: повторимо початковий масив двічі і просто знайдемо довжину найдовшого незараженого проміжку так, якби учасники сиділи не за круглим столом, а просто в рядок.

Складність розв'язку — $O(n)$.

Код розв'язку C++

Код розв'язку Python

D. Закордонна подорож

Припустимо, що в нас є необмежений запас вакцини першого типу, але лише одна вакцина другого типу. Якщо ми всім дамо вакцину першого типу, то сумарна користь становитиме $A_1 + A_2 + ... + A_n$. Якщо i-тій людині ми дамо вакцину другого типу, замість першого, то ця користь зміниться на $B_i - A_i$. Отже, вакцину другого типу є зміст давати лише учаснику в якого $B_i - A_i$ максимальне і лише в випадку, коли $B_i - A_i$ додатне для нього.

Аналогічно, якщо в нас є лише дві дози другої вакцини, то їх може бути зміст давати лише двом учасниками з найбільшими значеннями $B_i - A_i$.

Узагальнюючи ці міркування, отримуємо повний розв'язок. Впорядкуємо всіх учасників за незростанням $B_i - A_i$. Другу вакцину отримає деяка група людей, що стоїть на початку в цьому посортованому списку. Треба лише визначити скільком людям на початку списку вигідно дати другу вакцину. Очевидно, що ідеально було б дати вакцину другого типу усім в кого значення $B_i - A_i$ додатне, а решті дати вакцину першого типу. Але у цьому випадку нам може не вистачити вакцин одного з типів. А тому треба також врахувати, що другу вакцину ми маємо дати не менше ніж $n - x$ учасникам та не більше ніж y учасникам.

Складність розв'язку — $O(n \log(n))$.

Код розв'язку C++

Код розв'язку Python

E. Весела ізоляція

Якщо сума всіх елементів масиву парна, то Зеник виграє перших ходом. В іншому випадку, якщо Зеник залишить суму елементів непарною, Марічка виграє своїм наступним ходом. Щоб цього уникнути, Зенику потрібно своїм ходом обов'язково змінити парність суми масиву. Єдиний спосіб це зробити — взяти деякий відрізок парної довжини з непарною сумою, та замінити всі елементи в ньому на одинички. Тоді сума всіх елементів масиву зміниться з непарної на парну. Це єдиний спосіб Зенику уникнути негайного програшу після наступного ходу Марічки.

Зауважимо, що роблячи такий хід, Зеник збільшує кількість одиничок в масиві. Якщо Марічка буде дотримуватимуться такої ж стратегії, то кількість одиничок буде збільшуватися після кожного ходу, а тому, рано чи пізно, всі елементи стануть одиничками, і переможе Марічка. Тобто, незалежно від своїх ходів, Зеник не має жодного шансу на перемогу.

Отже, переможець гри однозначно визначається парністю суми всіх елементів масиву.

Складність розв'язку — $O(n)$.

Код розв'язку C++

Код розв'язку Python

F. ДНК реп'яховірусу

Нехай n — довжина рядка T, а m — довжина рядка S.

Якщо $n < m$, то відповідь 0, адже ми не можемо знайти жодного підрядка потрібної довжини у T.

Подивимося який вклад у загальну відповідь внесе перший символ рядка S: він додасть до відповіді 1 за кожен такий же символ рядка T, де може починатися підрядок довжини m (тобто символи від 1 до $n - m + 1$). Аналогічно, другий символ рядка S додасть до відповіді 2 за кожен такий же символ S з проміжку від 2 до $n - m + 2$.

Узагальнюючи, отримуємо, що вклад i-го символу рядка S у відповідь становитиме $i \cdot count_T(S_i, i, n - m + i)$, де $count_T(x, l, r)$ — це кількість символів x у рядку T на проміжку від l до r.

Отож, якщо ми навчимося швидко шукати $count_T(x, l, r)$, то задача розв'язана. Зауважимо, що у рядка, який складається з малих літер латинської абетки може бути щонайбільше 26 різних символів.

Заведемо двовимірний масив розміром $26 \times n$. Нехай $A_{i,j} = 1$, якщо j-й символ рядка T є i-тим символом алфавіту. Тепер, щоб знайти $count_T(x, l, r)$ нам потрібно знайти суму на відповідному відрізку одновимірного масиву A_i. Для цього скористаємося відомим методом, який називається *часткові суми.*

Нехай $B_{i,j} = A_{i,1} + A_{i,2} + ... + A_{i,j} = B_{i,j-1} + A_{i,j}$. Тепер, зрозуміло, що $count_T(x, l, r) = A_{i,l} + A_{i,l+1} + ... + A_{i,r} = B_{x,r} - B_{x,l-1}$.

Отже, обчисливши один раз масив B зі складністю $O(26 \cdot n)$, ми зможемо знаходити $count_T(x, l, r)$ за $O(1)$. Загальна складність розв'язку — $O(26 \cdot n)$.

Код розв'язку C++

Код розв'язку Python

F. Розробка вакцини

Яка довжина найкоротшого шляху, що починається в корені дерева проходить через усі листки дерева, а тоді повертається у корінь? Легко бачити, що такий шлях проходить по кожному ребру рівно 2 рази, а тому його довжина — $2n - 2$, де n — кількість вершин у дереві.

Як змінюється ця довжина якщо нам не потрібно повертатися в корінь після відвідування останнього листка? Нехай D_v — відстань від кореня до вершини v (ми також називатимемо цю величину глибиною вершини v). Зрозуміло, що закінчивши у листку x, ми економимо рівно D_x ребер. Отже, вигідно закінчити наш маршрут у вершині, для якої D_v максимальне.

Таким чином, щоб відповісти на запит другого типу нам потрібно знати кількість вершин у піддереві, а також глибину найглибшого листка. Якби не було запитів першого типу, то було б достатньо обчислити ці значення для усіх вершин за допомогою пошуку в глибину. Як же бути з запитами першого типу? Від задачі з додаванням вершин перейдемо до задачі з "увімкненням"вершин. Розглянемо дерево, що вийде після виконання усіх запитів першого типу (ми називатимемо його *фінальним деревом*). Тепер вважатимемо, що кожна вершина в цьому дереві може бути або увімкненою, або ні. Коли відбувається запит першого типу, то ми вмикаємо відповідну вершину. Для запиту другого типу нам потрібно знайти кількість увімкнених вершин у піддереві та максимальну глибину серед увімкнених вершин у піддереві.

Побудуємо Ейлеровий обхід фінального дерева: випишемо усі вершини дерева в порядку обходу в глибину. Кожному піддереву відповідає деякий неперервний відрізок в цьому обході. Це дозволяє перетворити запити на піддереві у запити на відрізку. Отож, тепер нам потрібна структура даних, що ефективно виконує такі операції:

- змінити значення елемента (щоб увімкнути вершину)
- знайти суму на відрізку (щоб знайти кількість вершин у піддереві)
- знайти максимальний елемент на відрізку (щоб знайти найглибшого листка)

Одна з таких структур — дерево відрізків. Вона може виконувати кожен з цих запитів з логарифмічною складністю.

Загальна складність — $O(n + q\log(n))$.

Код розв'язку C++