A. Міжпланетний інтернет 2

Автор задачі: Володимир Мшанецький
Кількість команд, що здали: 3 з 34 (9%)

Для короткості будемо називати колір квантових хвиль, які повинні дійти до певної планети, щоб на ній було забезпечено мережеве покриття квантового інтернету, кольором самої цієї планети. Розіб'ємо ланцюжок планет на послідовності планет одного кольору, що йдуть підряд. Очевидно, що в кожній послідовності потрібно встановити хоча б одну антену, бо із сусідніх послідовностей можуть прийти лише хвилі протилежного кольору. З іншого боку, не складно побачити, що в кожній послідовності необхідно встановити не більше двох антен. Якщо в кожній послідовності довжини 2 або більше встановити по одній антені на першій і на останній планеті, то не складно побачити, що всі планети отримають необхідні хвилі. Але така конфігурація антен не буде мінімальною.

Знайти мінімальну необхідну кількість квантових антен можна методом динамічного програмування. Нехай $f(n)$ дорівнює мінімальній необхідній кількості антен для забезпечення покриття на планетах $1...n$ за умови, що остання (сама права) антена знаходиться на планеті n. Будемо вважати, що $f(n)=0$ для всіх $n \le 0$.Тоді відповіддю задачі буде мінімальне значення $f(i)$ по всіх i, що належать останній послідовності планет одного кольору.

Розглянемо, як обчислити $f(n)$. Нехай L_{cur} — номер планети, з якої починається поточна (та, в якій знаходиться n) послідовність планет одного кольору ($L_{cur} \le n$), а L_{prev} — номер планети, з якої починається попередня послідовність ($L_{prev} < L_{cur}$). Для самої першої послідовності будемо вважати $L_{cur}=1$ і $L_{prev}=-\infty$. Тоді, якщо остання антена знаходиться на планеті n, то попередня антена може знаходитися:

- В поточній послідовності. Тоді $f(n)=min(f(i))+1$ по всім $L_{cur} \le i < n$. Очевидно, що цей варіант можливий, тільки якщо $n > L_{cur}$.
- В попередній послідовності. Тоді ця попередня антена повинна знаходитися на такій планеті x, що хвилі від неї та від антени на планеті n зустрічаються посередині між планетами L_{cur} і $L_{cur}-1$. Очевидно, що це є необхідним для забезпечення покриття в попередній та поточній послідовностях. Це означає, що повинна виконуватися рівність $(x+n)/2 = L_{cur}-0.5$. Звідси $x = 2L_{cur}-n-1$. Тоді $f(n)=f(x)+1$, але цей варіант можливий, тільки якщо $x \ge L_{prev}$.

Остаточним значенням $f(n)$ буде мінімум із цих двох варіантів. Не складно побачити, що для кожного n хоча б один із цих двох варіантів завжди буде можливим.

Отже, потрібно в циклі обчислити значення $f(i)$ для всіх i від 1 до довжини вхідного рядка. При цьому потрібно постійно підтримувати коректні значення L_{cur} і L_{prev}. Крім того, для успішної здачі задачі необхідно оптимально реалізувати обчислення мінімум $f(j)$ на відрізку $L_{cur} \le j < i$. Для цього можна постійно підтримувати значення цього мінімуму в змінній M. Не складно побачити, що в такому випадку після закінчення циклу по i відповідь задачі вже буде підраховано у змінній M.

B. Christmas lights

Автор задачі: Дмитро Садовий
Кількість команд, що здали: 10 з 34 (29%)

Спочатку розглянемо математичну частину розв'язку задачі. Припустимо, що гірлянда розбита на n сегментів, тобто між точками кріплення знаходиться n відрізків. Тоді ми можемо обчислити горизонтальну довжину одного сегмента як W/n, а довжину гірлянди на один сегмент як L/n. Через рівнобедрений трикутник, в якому бокова сторона $d=L/(2n)$ та половина основи $W/(2n)$, провис гірлянди обчислюється за теоремою Піфагора:

$$h=\sqrt{\left(\frac{L}{2n}\right)^2-\left(\frac{W}{2n}\right)^2}=\frac{1}{2n}\sqrt{L^2-W^2}$$

Щоб гірлянда не торкалася підлоги, потрібно, щоб провис h був **строго** меншим за H:

$$\frac{1}{2n}\sqrt{L^2-W^2}<H$$

Звідси:

$$n>\frac{\sqrt{L^2-W^2}}{2H}$$

Мінімальне ціле число сегментів, яке задовольняє умову, буде:

$$n_{min}=\lfloor\frac{\sqrt{L^2-W^2}}{2H}\rfloor+1$$

Зверніть увагу, що це — не те ж саме, що округлити догори. Оскільки завдяки початковим кріпленням у кутах вже буде 1 сегмент, кількість додаткових кріплень дорівнює:

$$answer=n_{min}-1=\lfloor\frac{\sqrt{L^2-W^2}}{2H}\rfloor+1-1=\lfloor\frac{\sqrt{L^2-W^2}}{2H}\rfloor$$

Тепер розглянемо технічну реалізацію цього розв'язку. На перший погляд може здаватися, що потрібно просто обчислити відповідь за вказаною вище формулою. Але через похибку при обчисленнях у числах із рухомою комою та залежно від того, як конкретно було записано формулу відповіді, при цьому можна отримати відповідь на один більше чи менше правильної. Теоретично, можна довести, що точності типу `long double` в C++ на x86 у компіляторі GCC достатньо при обчисленні відповіді за формулою, записаною в точності так, як зазначено вище. Але може бути безпечніше знайти n_{min} за допомогою пошуку.

Візьмемо знайдену раніше нерівність для n, та перетворимо її таким чином, щоб усі обчислення можна було виконати у цілих числах:

$$n>\frac{\sqrt{L^2-W^2}}{2H}$$
$$2Hn>\sqrt{L^2-W^2}$$
$$4H^2n^2>L^2-W^2$$

Тепер n_{min} можна знайти за допомогою лінійного (тобто, циклом) або бінарного пошуку. З нерівності очевидно, що шукати потрібно у діапазоні від 0 до $\lceil L/2\rceil$. Відповіддю задачі, як зазначено вище, буде $n_{min}-1$.

При реалізації пошуку слід звернути увагу на діапазони можливих значень обох частин нерівності. При лінійному пошуку від 0 до $\lceil L/2\rceil$ можна довести, що діапазону значень 64-бітових цілих буде достатньо. При бінарному пошуку діапазону 64-бітових цілих достатньо не буде, тому потрібно використовувати або тип `__int128` у C++, або довгу арифметику в інших мовах.

C. ElectroTransport

Автор задачі: Олександр Нестерюк
Кількість команд, що здали: 17 з 34 (50%)

У розв'язку цієї задачі використовується модифікований алгоритм Дейкстри, що працює не просто по станціях, а з розширеним простором станів. Опис базового алгоритму Дейкстри виходить за межі цього розбору, але про нього можна прочитати у Вікіпедії або запитати у вашої улюбленої LLM. Основна особливість задачі полягає в тому, що ребра між станціями можна використовувати лише за умови, що тип транспорту чергується: після трамвая має бути тролейбус і навпаки.

Зазвичай алгоритм Дейкстри реалізовують на основі черги з пріоритетом, в яку додають стани (ціна, поточна вершина). У цій задачі до стану додається третій компонент: (ціна, поточна вершина, тип останнього використаного транспорту). При ініціалізації алгоритму Дейкстри замість єдиного стартового стану існує одразу два стартових стани: $(0, A, 0)$ та $(0, A, 1)$. Поточна найкраща вартість проїзду до станції також зберігається окремо для кожного типу останнього використаного транспорту.

В основному циклі алгоритму при обробці кожного елемента черги перевіряється, чи не було вже знайдено кращий спосіб досягти даної станції з таким же типом транспорту — якщо так, поточний елемент ігнорується. Потім для кожного ребра, що виходить із поточної станції, перевіряється умова чергування: наступний транспорт повинен відрізнятися від попереднього. Якщо ця умова виконується, обчислюється нова сума вартості проїзду до відповідного сусіду, і якщо вона є кращою за збережену для цієї комбінації станції та транспорту, робиться релаксація та новий стан додається до черги.

Як тільки досягаємо кінцевої станції, алгоритм може одразу повернути результат — поточну суму вартості, адже вона буде гарантовано мінімальною завдяки властивостям алгоритму Дейкстри. Якщо по всьому графу не знайдено допустимого маршруту, слід вивести -1.

При реалізації розв'язку слід перевірити, що програма коректно обробляє випадок $A = B$. У ньому відповідь дорівнює нулю. Також слід звернути увагу на те, що в графі можуть бути петлі та кратні ребра.

D. Послідовні числа

Автор задачі: Денис Остапенко
Кількість команд, що здали: 0 з 34 (0%)

Ця задача на акуратний аналіз випадків. У ній легко перевірити, чи підходить відповідь, і багато хто так і робив, але перебір до 10^{1000} отримує TLE13 або PE2 (якщо у циклу верхня межа нижче другого тесту з умови `1011?214 -> 10110214`).

Випадок 1: якщо відповідь — невелике число, наприклад, до 1000. Його можна перебрати в лоб і результат перебору гарантовано мінімальний.

Випадок 2: якщо відповідь більше або дорівнює 1000. Тоді не складно довести, що зміна довжини поточного числа відбувається максимум один раз.

Випадок 2.1: зміна довжини відбувається рівно один раз. Перебираємо довжину першого числа L та позицію P, у якій довжина поточного числа змінюється на $L+1$ (з очевидними обмеженнями на кратність довжини лівого та правого шматка шаблону). Позиція P і довжина L однозначно задають, що там за числа: $999\ldots9$ до P, $1000\ldots0$ наступним числом. Можна цю потенційну відповідь $10^L-P/L$ просто перевірити, пройшовши по шаблону. Для мінімізації відповіді в цьому випадку перебираємо спочатку L, потім P з кінця, але більш оптимальна відповідь може бути отримана у випадку 2.2, тому зберігаємо всі знайдені кандидати в масив можливих відповідей, мінімум якого ми потім виведемо.

Випадок 2.2: немає зміни довжини. Тоді перебираємо довжину першого числа L. Вона повинна бути дільником довжини шаблону N, отже, перебираємо всі дільники N (для $N\le1000$ їх не більше 32), і записуємо шаблон в прямокутну таблицю розміром $R\times L$, де $R=N/L$, по одному числу на рядок.

Зауважимо, якщо в якомусь рядку таблиці в першому символі зустрічається 0, то поточна довжина L точно не підходить. Умова гарантує, що першим символом всього шаблону не може бути 0. Тому для всіх шаблонів завжди підходить мінімальне число довжини N, отримане з шаблону заміною всіх «?» крім першого на 0, а першого «?» на 1. Повернемо це число як потенційну відповідь для $L=N$.

Назвемо зміною розряду ситуацію, коли у деякій колонці йде спочатку одна цифра, а потім наступна. Для будь-якого $L<N$ зміна розряду присутня. Більше того, існує і є єдиною сама ліва і верхня зміна розряду. Не складно довести, що при даних обмеженнях на довжину шаблону в усіх розрядах, крім трьох наймолодших, зміна розряду може статися не більше одного разу. Розглянемо два випадки: випадок 2.2.1, коли перша (сама ліва і верхня) зміна розряду знаходиться в «хвості» чисел (у трьох наймолодших розрядах), і випадок 2.2.2, коли перша зміна розряду знаходиться в «голові» чисел.

Для оптимальної обробки наступних випадків потрібно спочатку підрахувати кілька допоміжних структур.

Для швидкості перевірки наявності цифр у прямокутному шаблоні підрахуємо 2D суму динамічним програмуванням `dp[r][l][D]` — скільки цифр D знаходиться у прямокутнику $0..r,0..l$. Використовуючи цей масив `dp` та принцип включення-виключення, ми зможемо за $O(1)$ відповідати на запитання, скільки цифр D знаходиться у прямокутнику $r_1..r_2,l_1..l_2$.

Далі можна підрахувати, використовуючи 2D суму, для якого максимального `const_powers` гіпотеза «перші `const_powers` цифр константні для всіх R чисел» не суперечить шаблону. Це означає, що у кожній колонці, тобто прямокутнику $0..R-1,c..c$ для кожного $c=0..\text{const\_powers}-1$, буде зустрічатися не більше одної різної цифри. Також можна заразом і записати в масив `const_digits`, які це цифри (якщо відома хоча б одна).

Далі можна підрахувати, які хвости чисел довжини 3 допускаються хвостом (останніми трьома колонками) прямокутного шаблону. Для ефективності можна використовувати такий підхід: для кожного степеня p (p останнього розряду вважаємо 0), для кожної відомої цифри D у хвості прямокутного шаблону (за координатами i,p) додамо умову на залишок за модулем 10^{p+1} хвоста першого числа. Цей залишок знаходиться всередині відрізка $[D\cdot10^p-i..(D+1)\cdot10^p-1-i]$ за

модулем 10^{p+1}. Відрізок може містити 0 (але не може містити більше 10% всіх залишків), тоді його слід розбити на два, $[min, 10^{p+1}-1] \cup [0, max]$. Відсортуємо межі відрізків як події та перетнемо всі ці відрізки. Дозволені лише залишки, накриті всіма відрізками. Отримавши всі можливі залишки від ділення хвоста першого числа на 10, 100 та 1000, перебором знайдемо всі дозволені хвости. Якщо отриманий масив дозволених хвостів порожній, немає жодного числа довжини L, що підходить під шаблон.

Випадок 2.2.1: ліва-верхня зміна розряду відбувається в хвості чисел довжини 3. Тоді голови всіх чисел однакові. Це можливо, якщо $\text{const\_powers}+3 \geq L$ та найменший дозволений хвіст t задовільняє нерівність $t+R-1<1000$. Кандидата на відповідь можна отримати заміною в розрядах з масиву `const_digits` повністю невідомих на $1000\ldots$, а наприкінці поставити мінімальний дозволений хвіст.

Випадок 2.2.2: сама ліва-верхня зміна розряду може відбуватися в перших $\text{const\_powers}+1$ розрядах, але в голові. Переберемо колонку p (від 0 до `const_powers`), в якій відбувається ця зміна, та рядок i (від 0 до $R-1$), після якого відбувається ця зміна. Зауважимо, що це не може бути зміна з 9 на 0, тому що тоді це була би не найлівіша зміна розряду, бо в розряді зліва від p також відбувалася би зміна. Можлива зміна розряду p після рядка i ділить прямокутник на 6 частин (іноді порожніх), кожна з яких повинна задовольняти свою умову, яку можна швидко перевірити, використовуючи підраховані раніше структури:

1. частина ліворуч від колонки зміни розряду p: повинна бути сумісна з гіпотезою, що всі ці розряди константні; автоматично виходить, якщо перебирати колонку зміни розряду p до `const_powers` включно
2. частина в колонці зміни розряду p, від рядка 0 до i включно: там повинна зустрічатися не більше ніж єдина цифра, і вона не має дорівнювати 9
3. частина в колонці зміни розряду p, нижче i: там повинна зустрічатися не більше ніж єдина цифра, і вона, якщо вона відома, повинна не дорівнювати 0 та бути на 1 більша за цифру з умови 2
4. частина в колонках голови правіше колонки зміни розряду p, від рядка 0 до i включно: там має зустрічатися лише 9
5. частина в колонках голови правіше колонки зміну розряду p, нижче i: там має зустрічатися тільки 0
6. хвіст: хвіст $999-i$ має бути дозволеним

Якщо всі умови на частини прямокутника задоволені, можна конструювати потенційну відповідь. Старші p розрядів отримуємо тією самою заміною «?» у масиві `const_digits` на $1000\ldots$ Для колонки зміни розряду p беремо або цифру з умови 2 (якщо відома), або цифру, що йде перед цифрою умови 3 (якщо відома). Якщо обидві цифри невідомі, з міркувань мінімальності знову беремо цифру з $1000\ldots$ Нарешті, для цифр правіше колонки зміни розряду беремо $999\ldots 9-i$.

Розглянуті випадки повністю покривають всі можливі ситуації і кожне побудоване в них число задовольняє шаблону і мінімально для параметрів, що перебираються, тому загальний мінімум буде дорівнювати мінімуму зі списку всіх знайдених потенційних кандидатів.

E. AI замінить математиків

Автор задачі: Володимир Мшанецький
Кількість команд, що здали: 25 з 34 (74%)

У задачі було потрібно порівняти два числа за наступним математично-неправильним алгоритмом:

1. Беремо перше число A як рядок та розбиваємо його на дві частини: перед та після крапки. Інтерпретуємо кожну частину як окреме ціле число — A_L і A_R.
2. Аналогічно друге число B розбиваємо на B_L і B_R.
3. Порівнюємо A_L і B_L. Якщо одно з них більше, то відповідне дробове число вважається більшим.
4. Порівнюємо A_R і B_R. Якщо одно з них більше, то відповідне дробове число вважається більшим.
5. Інакше дробові числа вважаються рівними.

Хоча в умові цього явно не зазначено, з останнього прикладу видно, що знак мінус впливає тільки не частину перед десятковою крапкою. Наприклад, в числі -9.11 ліва частина інтерпретується як -9, а права — як $+11$.

Алгоритм розв'язку цієї задачі є тривіальним, тому все зводиться до його реалізації. По-перше, числа необхідно зчитувати як рядки. Якщо зчитувати числа у змінні типу число з рухомою комою, то частину необхідної інформації буде втрачено. Наприклад, «`009.900`» буде зчитано як 9.9. При цьому, по-перше, частина після крапки стане 9 замість 900, а по-друге, ми втратимо незначущі нулі, які згідно з умовою задачі потрібно вивести.

По-друге, необхідно розбити кожне число (все ще як рядок) на частини перед та після крапки. В мовах Python, Java та Kotlin це можна зробити за допомогою методу `split()`. У Python і Kotlin можна написати `str.split('.')`, а в Java потрібно писати `str.split("\\.")`, бо аргументом `split()` у Java є регулярний вираз. У C++ зручного методу `split()` досі немає, тому розбиття необхідно робити вручну. Наприклад, можна використати `str.find('.')`, щоби знайти індекс крапки, а затим метод `substr()`, щоби розбити рядок на два.

По-третє, необхідно конвертувати отримані раніше рядки у цілі числа. У всіх мовах для цього є зручні функції. У C++ є функція `std::stoi()`, у Java є статичний метод `Integer.parseInt()`, у Kotlin є метод `String.toInt()`, а у Python — функція `int()`.

Альтернативно, у C++ можна одразу розібрати числа з початкової строки, не виконуючи розбиття на частини до та після крапки. Для цього можна використати функцію `std::sscanf()` або клас `std::istringstream`. У першому випадку потрібно використовувати маску `"%d.%d"`, а у другому — зчитувати `int`, `char` і знову `int`.

Нарешті, потрібно виконати порівняння та вивести відповідь. Тут можна написати кілька вкладених `if`. Під час виводу відповіді необхідно обов'язково виводити числа з рядкових змінних, в які їх було зчитано на самому початку. Як вже зазначалося, це необхідно для того, щоб зберегти незначущі нулі, як того вимагає умова задачі.

До речі, хто-небудь помітив, що відповідь M47h-bot на перше запитання також була очевидно неправильною?

F. Усихання грибів (завдання від хіміків)

Автор задачі: Вікторія Рувінська
Кількість команд, що здали: 34 з 34 (100%)

Результат можна розрахувати за формулою:

$$X = W \times \frac{100 - H_1}{100 - H_2},$$

де: W — початкова вага грибів, H_1 — початковий відсоток вологості (в процентах), H_2 — результуючий відсоток вологості (в процентах), X — результуюча вага.

В дужках зверху і знизу — процент сухої речовини в грибах до і після усихання. Тобто, початкова вага зменшується в стільки разів в скільки разів збільшується процент сухої речовини в грибах.

Далі — трохи інший підхід та більш детальний розбір.

Розглянемо приклад: вхідні дані — «`100 99 98`», відповідь — 50.0000. У цьому прикладі: зі 100 кг грибів 99% води, тобто, 99 кг води; отже, $100 - 99 = 1$ кг сухої речовини. При зменшенні води до 98% кількість сухої речовини не змінюється, зменшиться лише кількість води. Нехай X — шукана змінена вага грибів. Складаємо рівняння, в якому ліворуч і праворуч — вага сухої речовини, яка не змінюється при усиханні: праворуч — 1 кг, а ліворуч — $100\% - 98\% = 2\%$ від нової невідомої ваги X.

$$X \cdot (1 - 98/100) = 1$$
$$X \cdot 0.02 = 1$$
$$X = 1 / 0.02 = 50 \text{ кг}$$

Таким чином, загальний алгоритм розрахунку наступний:

1. Знаходимо вагу сухої речовини $W_{\text{без\_води}}$ в початкових грибах:
   $W_{води} = W \cdot H_1 / 100$
   $W_{\text{без\_води}} = W - W_{води}$
2. Знаходимо шукану змінену вагу грибів X (при цьому $W_{\text{без\_води}}$ не змінюється):
   $W_{\text{без\_води}} = X \cdot (1 - H_2/100)$
   Звідси, $X = W_{\text{без\_води}} / (1 - H_2/100)$

Якщо підставити у останню формулу для X попередні формули для $W_{\text{без\_води}}$ та $W_{води}$ і скоротити, то отримуємо формулу з початку цього розбору.

G. Driving exam

Автор задачі: Дмитро Садовий
Кількість команд, що здали: 23 з 34 (68%)

Мета задачі — знайти найменше значення P від 1 до 86400, при якому усі перевірки сайту (у моменти $P, 2P, 3P, \ldots$) відбуваються у «безпечних» секундах, тобто в ті моменти, коли на сайті немає доступних талонів. При розв'язанні задачі варто було звернути увагу на те, що прямий перебір всіх інтервалів для кожної секунди при кожному P може бути неефективним, якщо не застосувати підхід із попередньою обробкою інтервалів.

Щоб знайти найменше P, необхідно:

1. Знайти спосіб швидко визначати, для кожної секунди дня, чи є в цей момент хоча б один талон доступним.
2. Перебрати можливі значення P і перевірити, чи для кожного з них усі кратні $P, 2P, 3P, \ldots$ не припадають на моменти, коли талони доступні.

Для того, щоб мати можливість для кожної секунди дня швидко визначати, чи є в цей момент хоча б один талон доступним, побудуємо масив доступності розміром 86400. Замість того, щоб для кожної секунди окремо перевіряти, чи входить вона до інтервалу кожного талону (що було б надто повільно), застосуємо метод різниць. Створимо допоміжний масив розміром, наприклад, 86402 (адже секунди нумеруються від 1 до 86400, і потрібен додатковий елемент для роботи з кінцями інтервалів). Для кожного талону з інтервалом $[x_i; y_i]$ збільшуємо значення в позиції x_i на 1, а в позиції y_i+1 зменшуємо на 1. Далі проходимо циклом по цьому допоміжному масиву і накопичуємо суму його елементів до поточного (префіксну суму). Тоді для кожного моменту часу, якщо поточна сума дорівнює 0, то цей момент часу є безпечним, інакше — небезпечним. Це дозволяє заповнити масив доступності талонів.

Далі виконаємо перевірку всіх кандидатів для мінімального P. Для кожного кандидата P від 2 до 86400 перебираємо всі моменти часу, коли учень перевіряє сайт: $P, 2P, 3P, \ldots$ до кінця дня (тобто, не більше 86400). Якщо хоч один з моментів перевірки потрапляє у «небезпечну» секунду (тобто, коли талон доступний), поточне P відкидається. Якщо ж всі моменти перевірки виявляються безпечними, то P є відповіддю, і алгоритм завершується.

Якщо ми перевірили всі можливі P від 2 до 86400 і не знайшли відповідь, то відповіді не існує і потрібно вивести -1.

H. Турова в'язниця

Автор задачі: Денис Остапенко
Кількість команд, що здали: 11 з 34 (32%)

Розглянемо, які клітини на нескінченній дошці «безпечні» для королів. Рухається король за правилами шахів, але він не може ступати на клітини, де знаходиться тура або які вона б'є. Оскільки тура б'є по горизонталі та вертикалі, то усі цілі ряди та колонки, в яких стоять тури, недоступні. Таким чином, «безпечними» є рівно ті клітини, де і координата по x не збігається з жодною з координат тур, і координата по y теж не збігається з жодною з координат тур.

При цьому, з цієї множини безпечних клітин видно, що вони розбиваються на зв'язні компоненти (тобто, «окремі камери») — кожна така компонента являє собою прямокутник (може бути як скінченним, так і нескінченним) між «битими» (атакованими) рядами та колонками.

Якщо два королі знаходяться в одній зв'язній компоненті (тобто, в одній «камері»), то за допомогою послідовності допустимих ходів кожен король може переміщуватись по всій компоненті. Отже, з часом один із королів зможе підійти так близько до іншого, що їхні позиції стануть сусідніми (тобто, через один хід один з них потрапить у клітину з іншим). Щоб цього не могло статись, в кожній зв'язній компоненті безпечних клітин можна поставити не більше одного короля.

Отже, максимальна кількість королів — це рівно кількість зв'язних компонент безпечних клітин. Оскільки безпечні клітини — це ті, що не лежать на атаці тур, вони визначаються як всі пари (x, y), де:

- $x \notin X$ (X — множина всіх координат x тур)
- $y \notin Y$ (Y — множина всіх координат y тур)

Якщо розглянути ці осі окремо, то усі «биті» колонки (чи ряди) розбивають відповідну пряму на проміжки. При цьому, незалежно від того, чи проміжок скінченний чи ні, він є однією зв'язною компонентою в напрямку x (аналогічно по y). Тому кількість зв'язних компонент на дошці — це добуток (кількість проміжків по x) на (кількість проміжків по y).

Якщо X — відсортований список атакованих колонок, то «безпечні» проміжки по x виглядають так:

- усі цілі x менші за найменше x з X — це один проміжок
- для кожної пари сусідніх x_i і x_{i+1}, якщо $x_{i+1} - x_i > 1$, маємо відрізок $[x_i + 1, x_{i+1} - 1]$
- усі x більші за найбільше x з X — це ще один проміжок

Аналогічно для y.

Таким чином, остаточний алгоритм є наступним:

1. Зчитуємо усі координати x та y в окремі масиви X та Y
2. Сортуємо масиви X та Y
3. Проходимо циклом по масиву X і рахуємо кількість «безпечних» проміжків, отримуємо C_x
4. Аналогічно обчислюємо C_y
5. Обчислюємо відповідь $C_x \times C_y$

Випадок $N = 0$ слід обробити окремо, у ньому відповідь — 1.

Окремо слід звернути увагу, що остаточна відповідь $C_x \times C_y$ може виходити за діапазон припустимих значень 32-бітових цілих, тому для обчислень слід використовувати 64-бітові цілі.