Розбір задачі «Як розрізняти генератори»

Опишемо алгоритм, а після доведемо, що він справді правильний.

Будемо розв'язувати задачу окремо для кожного набору. Визначимо число $t = \lfloor \frac{30000}{K} \rfloor$ - кількість запитів, що може бути витрачена на один набір, таким чином, щоб рівномірно розподілити запити між наборами. Тепер зробимо запит для перших t елементів поточного набору. Якщо серед отриманих t чисел усі різні, визначаємо, що поточний набір - перестановка, інакше - масив.

Покажемо, чому цей алгоритм досягає бажаної точності визначення типу генерації. По-перше, якщо серед обраних елементів є однакові, то це гарантовано не перестановка, а масив. По-друге, ймовірність того, що в режимі генерації масиву буде згенерована перестановка дорівнює $\frac{n!}{n^n}$, що за обмежень задачі є меншим за 10^{-10000}, тобто можна бути повністю впевненими, що в тестах таких випадків немає. Отже, єдиний випадок, в якому ми можемо помилитися - це ситуація, коли серед перших t елементів усі різні, але десь далі все ж є повторення. Тоді масив буде визначений як перестановка. Визначимо ймовірність такої помилки.

Ідея подальших міркувань базується на відомій задачі про "парадокс днів народження". А саме ключовим фактом є таке твердження - якщо обрати t випадкових чисел від 1 до n, то ймовірність того, що усі t чисел виявляться різними дорівнює

$$\frac{A_n^t}{n^t} = (1 - \frac{1}{n}) \cdot (1 - \frac{2}{n}) \cdot ... \cdot (1 - \frac{t-1}{n})$$

Підставляючи $n = 40000$ та $t = \lfloor \frac{30000}{40} \rfloor = 750$ маємо, що ймовірність похибки менша за $8.6 \cdot 10^{-4}$. Отже, ймовірність допустити не більше двох помилок на 40 наборах дорівнює

$$(1-p)^{40} + (1-p)^{39} \cdot p \cdot 40 + (1-p)^{38} \cdot p^2 \cdot \frac{40 \cdot 39}{2} \approx 0.999994, \text{ де } p = 8.6 \cdot 10^{-4}$$

Отже, ймовірність того, що відповідь буде правильною на усіх тестах (яких за умовою не більше за 100), не менша за $(1 - 0.999994)^{100} \approx 0.9994$.

Розбір задачі «Брати чи не брати?»

Оригінальна ідея розглядається в книжці "Introduction to Algorithms, T. Cormen, Ch. Leiserson, R. Rivest, C. Stein" сторінка 97.

Опишемо алгоритм, а потім доведемо, що він досягає необхідної точності.

Зафіксуємо деяке число L ($1 \leqslant L < N$). Переглянемо перші L речей, і завжди будемо їх відкидати. Нехай найбільше значення параметра серед перших L речей дорівнює X. Тоді серед речей, що залишилися, будемо шукати першу річ, що має значення параметра більше, ніж X. Якщо жодної такої речі не знайшлося, візьмемо останню річ у магазині.

Визначимо ймовірність знайти найкращу річ таким алгоритмом. По-перше, зауважимо, що ми завжди можемо перенумерувати значення параметрів числами від 1 до N в порядку їх неспадання, і ймовірність знайти елемент N у цьому новому масиві буде дорівнювати ймовірності знайти максимум у початковому. Тобто ми розглядаємо абстрактну задачу - яка ймовірність того, що наведений алгоритм буде обирати елемент N у перестановці від 1 до N.

Розглянемо подію S_i, що алгоритм обрав елемент N при тому, що цей елемент знаходився на позиції i. Оскільки для всіх i ці події є взаємовиключними, то сумарна ймовірність успіху дорівнює $S = \sum_{i=L+1}^{n} P\{S_i\}$ (адже перші L позицій ми завжди відкидаємо).

Обчислимо ймовірність $P\{S_i\}$. Подія S_i полягає в тому, що ми обрали елемент N, що знаходився на позиції i. Нехай найбільший елемент серед перших L елементів дорівнює t. Тоді подія S_i відбувається тоді і тільки тоді, коли одночасно N знаходиться на позиції i (подія A_i), та усі числа від 1 до $i-1$ не більші за t (подія B_i). Зауважимо, що події A_i та B_i - незалежні. Отже $P\{S_i\} = P\{A_i\} \cdot P\{B_i\}$.

Очевидно, ймовірність $P\{A_i\} = \frac{1}{n}$. У свою чергу ймовірність $P\{B_i\}$ дорівнює ймовірності того, що найбільший елемент серед перших L елементів (який було позначено як t) також є найбільшим і серед перших $i-1$ елементів. Але, події зустріти максимум на будь-якій позиції є рівноймовірними, отже $P\{B_i\} = \frac{L}{i-1}$.

Таким чином

$$S = \sum_{i=L+1}^{n} \frac{L}{n \cdot (i-1)} = \frac{L}{n} \cdot \sum_{i=L+1}^{n} \frac{1}{i-1} = \frac{L}{n} \cdot \sum_{i=L}^{n-1} \frac{1}{i}$$

Ця сума може бути оцінена за допомогою інтегралу:

$$\sum_{i=L}^{n-1} \frac{1}{i} \geqslant \int_L^n \frac{1}{x} dx = \ln n - \ln L$$

Отже $S \geqslant \frac{L}{n}(\ln n - \ln L)$

Максимізація цього виразу по L дає нам максимум при $L = \frac{n}{e}$, при цьому сама ймовірність дорівнює $\frac{1}{e}$.

Отже, обираючи $L = \lfloor \frac{n}{e} \rfloor$ ми отримаємо ймовірність вгадати максимум більше за $p = 36\%$. Тоді ймовірність вгадати принаймні s разів з k дорівнює:

$$X = 1 - \sum_{i=0}^{s} C_k^i \cdot p^i \cdot (1-p)^{k-i}$$

За обмежень задачі (при $k = 500, s = \frac{500}{4} = 125$) маємо $X \approx 0.99999991$. При збільшенні k ця ймовірність збільшується. Отже, ймовірність відповісти правильно на усі тести (яких за умовою не більше 100) буде $0.99999991^{100} \approx 0.99991$.

Розбір задачі «Вгадай максимум»

Розв'яжемо аналогічну задачу в припущенні, що масив - це випадкова перестановка чисел від 1 до N. Потім покажемо, як застосувати отриманий алгоритм для початкової задачі.

Спочатку з'ясуємо, як нам дізнатись точне значення елементу з заданим індексом k. Це можна зробити за допомогою бінарного пошуку, використавши $\lceil \log_2 10^9 \rceil = 30$ запитів. Дійсно, зафіксуємо два кінці l та r (початково $l = 1$, $r = 10^9$), та зробимо запит: "Чи більше елемент з індексом k ніж $m = \lfloor \frac{l+r}{2} \rfloor$?". Якщо так, то змінимо l на $m + 1$, інакше змінимо r на m. Оскільки після кожного запиту відрізок зменшується вдвічі, то і сумарна кількість запитів буде не більше ніж 30.

Використовуючи описаний вище метод, знайдемо значення першого елементу масива a_1. Будемо підтримувати значення mx, що дорівнює максимуму з усіх елементів, які ми вже передивлялись. Тобто після першого кроку $mx = a_1$.

Будемо йти праворуч масивом і запитувати: "Чи більше елемент з поточним індексом за mx?". Якщо ні, то нас не цікавить цей елемент, адже він менше за той максимум, що ми вже знайшли. Якщо так, то знайдемо значення цього елементу бінарним пошуком, як було описано вище, та оновимо mx.

Покажемо, що наведений алгоритм буде вкладатися в обмеження на кількість запитів. Насправді, це доведення інтуїтивно випливає з відомого факту, що лінійний пошук максимума у випадковому масиві робить $\ln N + O(1)$ оновлень максимуму. Нажаль, оцінити строго ймовірність відхилення від математичного сподівання цієї величини не виявляється можливим аналітично, проте експеримент показує, що значних відхилень (більше ніж в 2 рази) ніколи (в ймовірнісному сенсі) не відбувається.

Отже для розв'язання поставленої задачі на випадковому масиві достатньо

$$n + \lceil 2 \ln N \rceil \cdot \lceil \log_2 10^9 \rceil < 200000 + 720 < 201000$$

Залишлося застосувати цей алгоритм для невипадкового масиву.

По-перше, помітимо, що наявність однакових елементів у масиві жодним чином не впливає на кількість виконань бінарного пошуку, адже ми виконуємо його лише у випадку, коли поточний елемент строго більше за поточний максимум. Отже, ми можемо виключити їх з розгляду і залишити лише масив з різних елементів.

Тепер ми можемо перенумерувати масив числами від 1 до N не впливаючи на роботу алгоритма, адже алгоритм використовує лише відносне порівняння двох елементів, а не їх реальні значення. Тобто отримуємо перестановку чисел від 1 до N.

Тепер, для того, щоб гарантувати випадковість цієї перестановки, нам було б добре її випадково перемішати, але цього ми зробити не можемо. Проте, замість цього ми можемо випадково визначити порядок, в якому ми будемо передивлятись елементи (замість руху праворуч від 1 до N). Таким

чином, наша задача зведеться до пошуку максимуму у випадковій перестановці, яка була розв'язана вище.

Розбір задачі «Арифметична прогресія»

Ідея задачі була взята з цієї задачі http://codeforces.com/problemset/problem/1114/E. Розбір можна знайти за цим посиланням - https://codeforces.com/blog/entry/65136.

Знайдемо максимальний елемент max бінарним пошуком. Для цього нам знадобиться $\lceil \log 10^9 \rceil = 30$ запитів типу ">". Використаємо останні 30 запитів типу "?", щоб знайти ще 30 випадкових елементів. Отже маємо 31 елемент, відсортуємо їх за зростанням та знайдемо попарні різниці між сусідними елементами: $s_1, s_2, \ldots, s_{30}$. Зрозуміло, що для усіх s_i виконується $s_i = k_i \cdot d$, де d - різниця арифметичної прогресії, k_i - деяке ціле додатнє число.

Обчислимо $r = gcd(s_1, s_2, \ldots, s_{30})$. Стверджується, що з великою ймовірністю $r = d$ (доведемо це далі). Тепер, знаючи max та d можемо легко відновити $x_1 = max - d(n-1)$.

Покажемо тепер, з якою ймовірністю можна стверджувати, що $r = d$. Знайдемо ймовірність P протилежної події - це означає, що для якогось p всі k_i діляться на p. Проте ймовірність того, що усі 30 k_i діляться на деяке p дорівнює $\frac{1}{p^{30}}$. При цьому ймовірність

$$P < \sum_{\substack{p=2 \\ p-prime}}^{10^9} \frac{1}{p^{30}} < \frac{1}{2^{30}} + \frac{1}{3^{30}} + \frac{1}{5^{30}} \cdot 10^9 < 10^{-9}$$

Отже, ймовірність невдачі складає менше, ніж 10^{-9}.

Розбір задачі «Знову арифметична прогресія»

Оригінальна задача була створена Тригубом Антоном для одного з турів Всеукраїнської учнівської олімпіади з інформатики 2019 року. За цим посиланням можна знайти оригінальну умову (з назвою "Козак Вус та цукерки"та розбір - https://oi.in.ua/uoi2019-results/.

Зауважимо, що можна розглядати лише цілі x_0 та d. Дійсно, якщо ми розглянемо, які елементи ми візьмемо (позначимо їх $b_1, b_2, \ldots, b_k$ у порядку зростання), то завжди можна обрати $x_0 = b_1$ та d - деякий дільник $gcd(b_2 - b_1, b_3 - b_2, \ldots, b_k - b_{k-1})$, що більший за 1. Якщо виявиться, що $gcd(b_2 - b_1, b_3 - b_2, \ldots, b_k - b_{k-1}) = 1$, то взяти такий набір взагалі неможливо. Більш того, достатньо розглядати лише прості d.

Нехай d - деякий простий дільник $gcd(b_2 - b_1, b_3 - b_2, \ldots, b_k - b_{k-1})$. Це означає, що усі b_i дають однаковий залишок при діленні на d. Тобто, якщо зафіксвати d та розбити початковий набір a_i на групи залежно від залишку елементів при діленні на d, то найбільша група і буде найкращою відповіддю для заданого d. Пошук найбільшої групи можна зробити за $O(n \log n)$ шляхом перетворення початкового масиву в map, де ключ - залишок від ділення на d, а значення - кількість таких елементів.

Залишилось зрозуміти, які саме прості числа d перевіряти, щоб задовольнити обмеження на час виконання. Насправді, можна помітити, що відповідь завжди не менша за $\lceil \frac{n}{2} \rceil$, адже при $d = 2$ принаймні половина елементів виявиться або парною, або непарною. Отже, якщо навмання обрати 50 пар, з імовірністю більшою за $1 - \frac{1}{4^{50}} > 1 - 10^{-6}$ принаймні одна з них має входити повністю до правильної відповіді. Отже, можна розглянути лише прості, що трапляються в розкладі на прості множники різниць елементів цих 50 пар чисел. Кожна різниця не більше за A, тоді їх добуток не більший за A^{50}. Можна показати, що при $A = 10^9$ кількість різних простих дільників чисел не більших за A^{50} не перевищує 180.

Для розкладання на прості можна використовувати алгоритм за $O(\sqrt{d})$

Отже кінцева асимптотика: $O(180n \log n + 50\sqrt{10^9})$

Розбір задачі «Все по 1000»

Розглянемо випадкову величину x_{ij} - елемент таблиці на перетині i-го рядка та j-го стовпця. За умовою всі такі величини рівномірно і незалежно розподілені на деякому відрізку $[a; b]$. Тоді $E[x_{ij}] = \frac{a+b}{2}$, де $E[x]$ - математичне сподівання випадкової величини x.

Розглянемо деякий шлях у таблиці. Позначимо випадкові величини, що відповідають коміркам, через які проходить цей шлях, як y_1, y_2, ..., y_{n+m-1} (дійсно кожен шлях складається з $n+m-1$ комірок). Математичне сподівання довжини шляху дорівнює

$$E\left[\sum_{i=1}^{n+m-1} y_i\right] = \sum_{i=1}^{n+m-1} E[y_i] = \sum_{i=1}^{n+m-1} \frac{a+b}{2} = (n+m-1)\frac{a+b}{2}$$

Оскільки шлях обирався довільно, і значення математичного сподівання його довжини не залежить від самого шляху, то і математичне сподівання середнього арифметичного всіх довжин дорівнює $S = (n+m-1)\frac{a+b}{2}$.

Залишилося визначити a та b. Точніше, нам достатньо знайти значення $\frac{a+b}{2}$. Для цього дізнаємося 1500 довільних елементів таблиці z_i за допомогою запитів та обчислимо їх середнє арифметичне Z. В якості відповіді виведемо $1999 \cdot Z$.

Покажемо, що знайдена відповідь потрапляє в обмеження на відхилення від правильної відповіді. По-перше зауважимо, що розподіли і Z, і S дуже близьks до нормального, адже обидві ці величини є сумою великої кількості незалежних рівномірно розподілених випадкових величин. А одже з великою ймовірністю (більшою за $1 - 10^{-4}$) реальне значення середнього арифметичного довжин всіх шляхів $\hat{S}$ буде лежати в межах $E[S] \pm 4\sqrt{Var[S]}$, а реальне значення $\frac{a+b}{2}$ буде лежати в межах $E[Z] \pm 4\sqrt{Var[Z]}$, де $Var[x]$ - дисперсія випадкової величини x.

Визначимо $Var[Z]$.

$$Var[Z] = Var\left[\frac{\sum_{i=1}^{1500} z_i}{1500}\right] = \frac{\sum_{i=1}^{1500} Var[z_i]}{1500^2} = \frac{\sum_{i=1}^{1500} \frac{(b-a+1)^2-1}{12}}{1500^2} = \frac{(b-a+1)^2-1}{18000}$$

Знайдемо тепер $Var[S]$. Для цього визначимо, у якій частці шляхів трапляється кожна з комірок. Тобто зафіксуємо деяку комірку (i,j) і порухаємо відношення r_{ij} кількості шляхів, які проходить через цю комірку, до кількості всіх шляхів. Зауважимо, що загальна кількість шляхів в таблиці $n \times m$ дорівнює C_{n+m-2}^{n-1}. Отже $r_{ij} = \frac{C_{i+j-2}^{i-1} \cdot C_{n+m-i-j}^{n-i}}{C_{n+m-2}^{n-1}}$.

Тоді

$$S = \sum_{i=1}^{n}\sum_{j=1}^{m} r_{ij} \cdot x_{ij}$$

$$Var[S] = \sum_{i=1}^{n}\sum_{j=1}^{m} r_{ij}^2 \cdot Var[x_{ij}] = Var[x_{ij}] \cdot \sum_{i=1}^{n}\sum_{j=1}^{m} r_{ij}^2 = \frac{(b-a+1)^2-1}{12} \cdot \sum_{i=1}^{n}\sum_{j=1}^{m} r_{ij}^2$$

Можна показати, що за обмежень задачі $\sum_{i=1}^{n}\sum_{j=1}^{m} r_{ij}^2 < 80$, отже,

$$Var[S] < \frac{80}{12} \cdot ((b-a+1)^2-1) = \frac{20}{3} \cdot ((b-a+1)^2-1)$$

.

Для спрощення обчислень знехтуємо доданком -1 у $Var[S]$ та $Var[Z]$ і отримаємо

$$\sigma_S = \sqrt{Var[S]} < 2.6(b-a+1)$$

$$\sigma_Z = \sqrt{Var[Z]} < \frac{b-a+1}{\sqrt{18000}}$$

Отже з великою ймовірністю:

$$|\hat{S} - 1999\frac{a+b}{2}| < 4 \cdot 2.6(b-a+1)$$

Також:

$$|\frac{a+b}{2} - Z| < 4\frac{b-a+1}{\sqrt{18000}}$$

Отже

$$|\hat{S} - 1999Z| < 4(b-a+1) \cdot (2.6 + \frac{1999}{\sqrt{18000}}) < 70(b-a+1)$$

з ймовірністю, більшою за $1 - 10^{-4}$.

Розбір задачі «Майже все по 1000»

Позначимо як x_{ij} значення у комірці (i, j). Обчислимо, який внесок до результату робить кожна з комірок. Тобто зафіксуємо деяку комірку (i, j) і порахуємо відношення r_{ij} кількості шляхів, які проходить через цю комірку, до кількості всіх шляхів. Зауважимо, що загальна кількість шляхів в таблиці $n \times m$ дорівнює C_{n+m-2}^{n-1}. Отже $r_{ij} = \frac{C_{i+j-2}^{i-1} \cdot C_{n+m-i-j}^{n-i}}{C_{n+m-2}^{n-1}}$.

А шукана величина S обчислюється як:

$$S = \sum_{i=1}^{n} \sum_{j=1}^{m} r_{ij} \cdot x_{ij}$$

Для обмежень, заданих в умові задачі, можна обчислити r_{ij} та помітити, що серед них багато таких, що дуже близькі до нуля, через що незалежно від значення x_{ij}, їх вклад у величину S буде близький до нуля. Покажемо, як це застосувати.

Розіб'ємо множину усіх комірок таблиці на дві групи A_1 та A_2 таким чином, щоб до групи A_1 увійшли комірки з першими 115000 найбільшими значеннями r_{ij}, а до групи A_2 - решта комірок.

Тоді вираз для S можна переписати як:

$$S = \sum_{(i,j)\, in\, A_1} r_{ij} \cdot x_{ij} + \sum_{(i,j)\, in\, A_2} r_{ij} \cdot x_{ij}$$

У якості відповіді будемо надавати значення $\hat{S}$, що обчислюється як:

$$\hat{S} = \sum_{(i,j)\, in\, A_1} r_{ij} \cdot x_{ij} + 0.5 \cdot \sum_{(i,j)\, in\, A_2} r_{ij}$$

Покажемо, що така оцінка дійсно задовольняє умову задачі:

$$|S - \hat{S}| < \sum_{(i,j)\, in\, A_2} r_{ij} \cdot |x_{ij} - 0.5| < 0.5 \cdot \sum_{(i,j)\, in\, A_2} r_{ij}$$

При цьому, безпосереднє обчислення значення $\sum_{(i,j)\, in\, A_2} r_{ij}$ показує, що воно є меншим за 2.

Таким чином $|S - \hat{S}| < 1$.

Важливою частиною розв'язку є обчислення r_{ij} з потрібною точністю та швидкістю. Для цього можна застосувати динамічне програмування. Відомо, що $r_{11} = 1$. Далі, згідно з формулою для r_{ij} можна записати:

$$r_{ij} = r_{i-1j} \frac{(i+j-2)(n-i+1)}{(i-1)(n-m-i-j+1)}$$

або

$$r_{ij} = r_{ij-1} \frac{(i+j-2)(m-j+1)}{(j-1)(n-m-i-j+1)}$$

Для того, щоб отримати достатню точність, необхідно при обчисленні кожного наступного елемента, спиратися на більший з попередніх. Так ми уникаємо спроби відтворити ненульове значення з такого, що вже стало дуже близьким до нуля.

Розбір задачі «Дерево»

Посилання на оригінал ідеї - http://codeforces.com/problemset/problem/1061/F. Розбір оригінальної задачі - https://codeforces.com/blog/entry/63384.

Розіб'ємо задачу на три частини:

1. Знайти будь-який лист $leaf_1$
2. Знайти будь-який інший лист в іншому, відносно кореня, піддереві $leaf_2$
3. Знайти корінь $root$

Нехай ми зафіксували вершину u. Можна перевірити, чи є вона листом за n запитів. Для цього оберемо довільну іншу вершину v, та для всіх інших вершин w запитаємо, чи проходить шлях $v \rightsquigarrow w$ через нашого кандидата на лист u. Маємо, що u - лист тоді і тільки тоді, коли на всі запити буде отримана відповідь "ні".

Тепер помітимо, що в повному k-ічному дереві кількість листів дорівнює $\frac{N(k-1)+1}{k}$. Отже, ймовірність отримати лист, обираючи вершину навмання, більше за $\frac{k-1}{k}$. При $k = 2$ досягається найменше значення $\frac{1}{2}$. Таким чином, обираючи навмання вершину, та перевіряючи її за n запитів, ми з ймовірністю невдачі меншою за $\left(\frac{1}{20}\right)^{20}$ знайдемо лист за $20n$ запитів.

Отже, ми знаємо лист $leaf_1$. З величин n та k ми можемо однозначно визначити висоту дерева (кількість ребер на шляху від кореня до листа) h. Тепер нам необхідно знайти будь-який лист $leaf_2$, що знаходиться в іншому, відносно кореня, піддереві, ніж $leaf_1$. Для цього достатньо знайти будь-яку вершину u, щоб на шляху $leaf_1 \rightsquigarrow u$ було рівно $2h - 1$ вершин. Це твердження виконується тоді і тільки тоді, коли u - лист у іншому піддереві, відносно кореня, ніж $leaf_1$.

Для перевірки такого твердження достатньо n запитів. А саме, для кожної іншої вершини v запитаємо, чи проходить шлях $leaf_1 \rightsquigarrow u$ через v. Якщо кількість таких v рівно $2h - 1$, то це і є наш $leaf_2$.

Оскільки кількість листів в обох піддеревах корня є рівною, а ймовірність знайти лист серед усіх вершин навмання не менша за $\frac{1}{2}$, то ймовірність знайти лист з іншого піддерева не менша за $\frac{1}{4}$. Тоді за $40n$ запитів ми з ймовірністю невдачі меншою за $\left(\frac{1}{20}\right)^{20}$ знайдемо шуканий лист $leaf_2$.

Для того, щоб відновити корінь ми використаємо інформацію про шлях між листами $leaf_1$ та $leaf_2$. Будемо вважати, що ми запам'ятали усі вершини, через які проходив шлях $leaf_1 \rightsquigarrow leaf_2$ - $v_1, v_2, \ldots, v_{2h-1}$. Знайдемо, яка з них є коренем. Нехай поточний кандидат - v_i. Будемо запитувати для всіх інших v_j, чи лежить v_j на шляху $leaf_1 \rightsquigarrow v_i$. Якщо таких v_j виявиться рівно $h - 1$, то вершина v_i - корінь. На такий пошук достатньо h^2, тобто $O(\log^2 N)$ запитів. Для того, щоб до цього моменту залишилось h^2 запитів достатньо не задавати зайвих запитів під час попередніх кроків. Наприклад, під час перевірки на лист, якщо отримали, що якийсь шлях проходить через кандидата, одразу зупинятись і дивитись іншу вершину.

Розбір задачі «Цікаві картки»

Посилання на оригінал ідеї - http://codeforces.com/problemset/problem/843/B. Розбір оригінальної задачі - http://codeforces.com/blog/entry/54029.

Запитаємо значення найлівішої картки та ще випадкових 999 карток. Серед усіх отриманих карток, оберемо таку, що має найбільше значення, що менше за X. Після цього, запитаємо від цієї картки 1000 наступних. Першу серед них, що опиниться більша за X, і є відповіддю. Якщо таких не знайшлося, то виведемо -1. Окремо слід розглянути випадки, коли X менше за значення першої картки.

Покажемо, чому цей алгоритм знаходить відповідь з великою ймовірністю. Нехай картка з відповіддю знаходиться на позиції k зліва. Якщо ми не знайшли цю картку, це означає, що жодна з наших випадково обраних позицій не потрапила у відрізок $[k - 1000; k]$. Ймовірність цього є $\left(\frac{50000-1000}{50000}\right)^{1000} < 1.7 \cdot 10^{-9}$.

Розбір задачі «Встигни вгадати»

Посилання на оригінал ідеї - http://codeforces.com/problemset/problem/1039/B. Розбір оригінальної задачі - https://codeforces.com/blog/entry/61668.

Будемо підтримувати відрізок $[l;r]$, на якому точно буде знаходитись число після його наступної зміни. Спочатку $l = 1, r = N$. Нехай, нам відомо, що зараз число на відрізку $[l;r]$, тоді запитаємо про відрізок $[l;m]$, де $m = \lfloor \frac{l+r}{2} \rfloor$. Якщо отримаємо відповідь так, то новий відрізок буде $[l-k, m+k]$, якщо ні - $[m-k, r+k]$. Так на кожному кроці ми зменшуємо відрізок вдвічі і збільшуємо на $2k$. Можемо таким чином дійти до відрізка довжиною $4k$. Для цього нам знадобиться не більше за $\log N + 6$ запитів.

Нехай ми маємо відрізок $[l;r]$ довжиною $4k$. Оберемо випадковим чином з нього деяке число t та запитаємо, чи є число на відрізку $[t;t]$. Якщо так - то відповідь t. Якщо ні, за допомогою бінарного пошуку (починаючи з відрізку $[l-k, r+k]$) отримаємо знову відрізок довжини $4k$ та повторимо спробу. На кожну спробу нам необхідно не більше 6 запитів. Отже, можна зробити не менше ніж $\frac{1500-(\log N+6)}{6} > 238$ спроб.

Ймовірність не вгадати число на відрізку довжини $4k$ за наших обмежень є не більшою за $p = \frac{39}{40}$. Ймовірність не вгадати число 238 разів є не більшою за $p^{238} < 0.0025$.

Можна отримати навіть меншу ймовірність помилки, якщо в якості порогу обрати не $4k$, а, наприклад, $4k + 5$, тоді ймовірність помилки буде не більше за $4 \cdot 10^{-4}$.