divided by several groups. If repeat the simulation with modern communication technologies and mass media enabled, then the result will significantly differ: the party can spread its ideas over whole society and the society finds a consensus.

A case with a single political party which spread over whole society was analyzed with help of the model. It shows how resulting opinion in the society depends on the initial party parameters (number of party members and their initial opinion). The simulation showed that, in general, improving the party ideology is more important than increasing number of party members. However, there is a threshold of party members (near 3% of the society members) below which the party cannot significantly influence the society.

The situation when society already had some opinion and a political party with opposite opinion tried to change society mind was considered. It was shown, that a threshold of party members number which lets the party reach its goal exists.

Numerical experiments with two opposite political parties in a society showed, that if both parties evenly spread over the society, then such society always finds a consensus. But if each party dominates on own territory, then confrontation in the society grows. At some point it reaches its maximum and then, if the society didn't split in two, the society slowly goes towards consolidation.

Finally, obtained model can also be used for different from described in this article scenarios.

Keywords: *model, society, public opinion, consensus.*

Одержано редакцією 16.03.2020 р.
Прийнято до публікації 18.05.2020 р.
Уточнення до списку літератури (п.12) 24.06.2020 р.

УДК 519.688

DOI 10.31651/2076-5886-2020-1-44-60

PACS 02.70.-c

ПОРУБЛЬОВ Ілля Миколайович,
старший викладач, Черкаський національний університет імені Богдана Хмельницького
e-mail: ilya@vu.cdu.edu.ua
ORCID: 0000-0001-7369-3862

ДЕЯКІ АЛГОРИТМИ ПОШУКУ ЧИСЕЛ З МАКСИМАЛЬНОЮ КІЛЬКІСТЮ ДІЛЬНИКІВ

Розглянуто кілька варіантів постановки та способів розв'язування задачі пошуку чисел, що мають найбільшу кількість дільників серед чисел вказаного проміжку. Аргументовано, чому задачу для проміжку від M до N слід вважати складнішою, ніж для проміжку від 1 до N. Розглянуто узагальнення, коли максимальна кількість дільників шукається не серед усіх чисел проміжку, а лише серед не кратних деякому K. Побудовано кілька варіантів алгоритму розв'язання кількох варіантів цієї задачі, на основі істотних оптимізацій рекурсивного перебору. Наведено порівняльний аналіз швидкодії роботи розглянутих варантів алгоритму, оцінено переваги та недоліки кожного з варіантів та наведено рекомендації про доцільність застосування розглянутих варіантів при роботі з великими числами.

Ключові слова: *кількість дільників, надскладені числа, рекурсивний перебір та його оптимізації.*

Вступ

Числа з великою кількістю дільників можуть мати властивості, які, зокрема, істотно впливають на час роботи деяких алгоритмів. Тому і фактичне знаходження таких чисел, і оцінка можливої кількості дільників є цікавими та мають значення.

Оскільки у світі існує дві різні трактовки натурального числа (за однією з них 0 не є натуральним числом, за іншою – є), зафіксуємо, що в цій статті дотримуємося першого варіанту (0 не є натуральним).

Означення 1. Будемо позначати кількість натуральних дільників числа n як $\tau(n)$. (Це позначення, введене, зокрема, в [4] та багатьох інших джерелах. На жаль, воно не є цілком загальноприйнятим; як стверджують [1, сторінка oeis.org/A000005] та [3], це саме можуть позначати також $d(n)$ чи $\sigma_0(n)$.

Наприклад, $\tau(18) = 6$, і цими шістьма числами є 1, 2, 3, 6, 9, 18.) □

Означення 2. Будемо називати число n ***надскладеним***, якщо кількість його дільників строго перевищує кількість дільників будь-якого меншого числа. Інакше кажучи, n надскладене, коли

$$\forall k_{1\le k<n}\big(\tau(k) < \tau(n)\big). \qquad (1)$$

Зокрема, сайт [1] описує послідовність A002182, якій дається назва «highly composite numbers» та означення «такі числа n, що кількість дільників (n) стає рекордно великою» ([1, сторінка oeis.org/A002182]). Автору не вдалося знайти офіційного перекладу «highly composite numbers» українською та вдалося знайти російський переклад «сверхсоставные числа» ([2]), тому вжито переклад «надскладені числа». Першими десятьма надскладеними числами є: 1, 2, 4, 6, 12, 24, 36, 48, 60, 120. □

У найстандартнішому вигляді ця задача в значній мірі вже розв'язана. Зокрема, вже згадана сторінка oeis.org/A002182 сайту [1] містить посилання на список точних значень перших 10000 надскладених чисел (10000-е число перевищує 10^{481}, чого для більшості практичних застосувань достатньо). Крім того, [3] стверджує, що у [6] теоретично доведена оцінка, що кількість дільників n асимтотично зростає досить повільно, $\forall \varepsilon_{\varepsilon>0}(\tau(n) \in o(n^{\varepsilon}))$.

Але відомі теоретичні викладки не є настільки точними, щоб отримувати з них конкретні значення надскладених чисел та кількостей їхніх дільників, а дані про конкретну послідовність A002182 не містять ні добрих описів алгоритмів, якими можна їх отримувати, ні можливостей узагальнень на інші схожі задачі. (Алгоритми, наведені безпосередньо на згаданій сторінці oeis.org/A002182 сайту [1] не є досить ефективними, бо передбачають перебір усіх чисел та знаходження кількості дільників кожного; ними нереально отримати більше, ніж перших 50–200 (залежно від швидкодії комп'ютера й того, скільки часу згодні чекати на результат) членів послідовності.)

Мета статті

Описати розроблені алгоритми для розв'язання задачі пошуку чисел з великою кількістю дільників, у таких варіантах постановки:

1. Дано натуральне число N. Для проміжку всіх натуральних чисел від 1 до N (обидві межі включно), знайти число, що має максимальну кількість дільників.
2. Дані натуральні числа M, N ($M \le N$). Для проміжку всіх натуральних чисел від M до N (обидві межі включно), знайти число, що має максимальну серед чисел проміжку кількість дільників.
3. Дані натуральні числа N та K (K значно менше N, $K \ge 2$). Для проміжку всіх натуральних чисел від 1 до N (обидві межі включно), але ігноруючи чи*с*ла, кратні K, знайти число, що має максимальну кількість дільників
   а) якщо гарантовано, що число K просте;
   б) якщо такої гарантії нема.

В усіх варіантах постановки мається на увазі, що знайти треба і максимальну серед кількостей дільників усіх чисел проміжку, і те число з проміжку, яке має цю максимальну кількість дільників.

Виклад основного матеріалу

Твердження 1. Нехай відоме канонічне розкладення числа N на прості множники

$$N = p_1^{k_1} \cdot p_2^{k_2} \cdot \ldots \cdot p_m^{k_m} \qquad (2)$$

(всі p_i прості та різні, всі k_i натуральні). Тоді кількість різних дільників N дорівнює

$$\tau\left(p_1^{k_1} \cdot p_2^{k_2} \cdot \ldots \cdot p_m^{k_m}\right) = (k_1+1)\cdot(k_2+1)\cdot\ldots\cdot(k_m+1). \qquad (3)$$

(Доведення цього твердження пропустимо як загальновідоме; в разі потреби його можна знайти, наприклад, у [5, § 9].) □

Наслідок твердження 1. Якщо числа a та b взаємно прості, то $\tau(a\cdot b) = \tau(a)\cdot\tau(b)$; інакше, $\tau(a\cdot b) < \tau(a)\cdot\tau(b)$.

Доведення. Розглянемо довільне просте число p_i. Позначимо показник цього простого в розкладенні a як $k_{i,a}$, в розкладенні b як $k_{i,b}$. Тоді це просте привносить у $\tau(a)\cdot\tau(b)$ множники $(k_{i,a}+1)$ та $(k_{i,b}+1)$, а в $\tau(a\cdot b)$ один множник $(k_{i,a}+k_{i,b}+1)$. Оскільки $(k_{i,a}+1)\cdot(k_{i,b}+1) = k_{i,a}\cdot k_{i,b} + k_{i,a} + k_{i,b} + 1 = (k_{i,a}\cdot k_{i,b}) + (k_{i,a}+k_{i,b}+1)$, а $k_{i,a}$ та $k_{i,b}$ невід'ємні, то можуть бути два випадки. Якщо хоча б один з показників $k_{i,a}$ чи $k_{i,b}$ є нулем, то $k_{i,a}\cdot k_{i,b} = 0$ і внесок у $\tau(a)\cdot\tau(b)$ та $\tau(a\cdot b)$ виявляється однаковим. Якщо ж обидва $k_{i,a}$ та $k_{i,b}$ додатні, то внесок у $\tau(a)\cdot\tau(b)$ виявляється більшим, чим у $\tau(a\cdot b)$, на $k_{i,a}\cdot k_{i,b}$. Лишилося згадати, що всі множники у правій частині (3) додатні, а взаємно прості й не взаємно прості саме тим і відрізняються, що у взаємно простих (і лише в них) нема спільних додатних показників степеню при тому ж простому, й доведення завершене. ■

Надалі для цілей цієї статті зручніше трактувати розкладення трохи інакше, залишивши вимогу «всі p_i прості та різні», але дозволивши не лише натуральні, а й нульові степені, як-то $98 = 2^1\cdot 3^0\cdot 5^0\cdot 7^2$. Також, для цілей цієї статті зручніше вимагати, щоб розкладення на прості множники завжди містило всі прості числа $p_0 = 2$, $p_1 = 3$, $p_2 = 5$, $p_3 = 7$, $p_4 = 11$, … в порядку зростання аж до останнього реально задіяного в розкладенні (якщо якогось з цих p_i нема, вважати, що є, але з показником $k_i = 0$; нумерація з 0 не принципова, вводиться лише заради більшої схожості з більшістю сучасних мов програмування). При дотриманні всіх цих вимог, можна вважати, що розкладення однозначно задається самою лише послідовністю $(k_0, k_1, \ldots)$. Починаючи з цього місця, скрізь у статті (де не вказано інше) мається на увазі саме таке трактування. Така зміна трактування не впливає на правильність твердження 1, бо при $k_i = 0$ права частина (3) домножується на $(0+1) = 1$, що не впливає на результат.

Твердження 2. Для всіх надскладених (згідно означення 2) чисел, $k_0 \ge k_1 \ge \ldots$

Доведення. Припустимо, ніби в розкладенні є хоча б одна пара така, що $a < b$, але $k_a < k_b$. Згідно нашого трактування розкладення, $a < b$ означає, що $p_a < p_b$. Тоді можна, залишивши решту степенів простих (якщо такі в цьому розкладенні є) незмінними, замінити $p_a^{k_a}\cdot p_b^{k_b}$ на $p_a^{k_b}\cdot p_b^{k_a}$. Це зменшить добуток (число, розкладення якого розглядаємо), тому що $k_b - k_a > 0$ множників зменшаться з p_b до p_a, при тому, що всі множники залишаться додатними. Водночас, кількість дільників лишиться тією самою, бо в (3) множники тільки обміняються місцями, що не впливає на значення добутку. Отже, початкове число не є надскладеним, бо ми побудували менше число з такою ж кількістю дільників. Тобто, маємо протиріччя, джерело якого — у припущенні, ніби в розкладенні надскладеного числа може порушитися вимога $k_0 \ge k_1 \ge \ldots$ ■

Твердження 2 задає односторонню імплікацію, а не рівносильність. Наприклад, для $192 = 2^6\cdot 3^1$ вимога $k_0 \ge k_1 \ge \ldots$ виконується ($k_0 = 6 \ge 1 = k_1$, всі подальші $k_i = 0$), але 192 не надскладене: має $(6+1)\cdot(1+1) = 14$ дільників, тоді як менше $120 = 2^3\cdot 3^1\cdot 5^1$ має їх більше, а саме $(3+1)\cdot(1+1)\cdot(1+1) = 16$.

Наслідок 1 твердження 2. Якщо задачу слід розв'язати на проміжку від 1 до N, то оцінити згори, які прості множники мають шанс бути включеними у надскладені, а які точно не будуть, можна так: перебрати $\prod_{i=0}^{k} p_i$ (тобто, послідовно формувати добутки 2, потім $2 \cdot 3$, потім $2 \cdot 3 \cdot 5$, потім $2 \cdot 3 \cdot 5 \cdot 7$, …), доки добуток не перевищить N; повернутися до попереднього (останнього, при якому ще не перевищив). Прості з більшими номерами не зустрічатимуться у надскладених, бо якщо те просте з більшим номером входить у розкладення з деяким показником $k_i > 0$, то з не меншими показниками повинні входити також усі попередні прості, й добуток перевищить N. ■

Визначена таким способом кількість дещо завищена, бо не враховує, що фактично в розкладеннях більшості надскладених малі прості мають більші, чим одиниця, показники, й перевищення N стається на ще меншому простому. Але це правильна оцінка згори. Адекватно подати цю оцінку аналітично не здається можливим, бо нема простого аналітичного способу задати залежність величини простого числа від його номера. Грубо (додатково завищивши) це можна оцінити як «кожне просте $p_i \geq 2$, тож, щоб добуток був у межах N, кількість простих повинна бути бути не більшою $\log_2 N$».

Наслідок 2 твердження 2. Якщо бажаємо знайти надскладені, не маючи способу відразу лише їх і формувати (й тому потребуючи перебирати різні ч*и*сла, щоб вибрати серед них потрібні), їх можна шукати не серед взагалі всіх чисел, а лише серед тих, для розкладень яких виконується $k_0 \geq k_1 \geq \ldots$ ■

Раз ці твердження та наслідки стосуються надскладених чисел, вони можуть бути використані при розробці алгоритму вирішення задачі в постановці № 1. Також їх легко модифікувати під постановку № 3а (проміжок від 1 до N, заборонене K просте): досить викреслити K з простих, і це призведе до потрібного у цій постановці ігнорування всіх чисел, кратних K, і лише їх.

Для постановки № 3б безпосередньо вони не є правильними. Наприклад, для проміжку від 1 до 42, $K = 6$, найбільшу кількість дільників ($(3 + 1) \cdot (1 + 1) = 8$) має число $40 = 2^3 \cdot 3^0 \cdot 5^1$, для якого $k_1 = 0 < 1 = k_2$. Але далі будуть доведені твердження 4, 5 та 7, які можна вважати частковими приблизними аналогами твердження 2.

Для постановки № 2 воно теж неправильне (наприклад, на проміжку від 49 до 51 найбільшу кількість дільників ($(1 + 1) \cdot (2 + 1) = 6$) має $50 = 2^1 \cdot 3^0 \cdot 5^2$, де $k_0 = 1 < 2 = k_2$ та $k_1 = 0 < 2 = k_2$), і не вдалося знайти ніякого аналога цьому твердженню.

Загальний опис та аналіз алгоритму для постановки № 1. Можна організувати перебір рекурсивно, з такими аргументами рекурсії:

- `primeIdx` – для якого простого числа підбираємо степінь на поточному рівні рекурсії, якщо прості записані у масиві `primes`, а `primeIdx` є індексом (набуває значень 0, 1, 2, ..., внаслідок чого `primes[primeIdx]` набуває значень 2, 3, 5, ...).
- `maxDeg` – максимальний дозволений показник степені; потрібен, щоб підтримувати монотонність послідовності показників.
- `prodBefore` – який добуток усього, вже вибраного на зовнішніх рівнях рекурсії (якщо, наприклад, перебуваємо на рівні `primeIdx` = 2 і на попередніх рівнях було вибрано, що $p_0 = 2$ береться у степені 4, а $p_1 = 3$ у степені 2, то повинно бути `prodBefore` $= 2^4 \cdot 3^2 = 144$); потрібне, щоб запам'ятати один з результатів (яке число має максимальну кількість дільників).
- `divsBefore` – скільки дільників має `prodBefore` (у тому ж прикладі «$2^4, 3^2$» повинно бути $(4 + 1) \times (2 + 1) = 15$); теоретично `divsBefore` можна встановити на основі `prodBefore`, але практично для цього треба було б розкладати `prodBefore`; значно

швидше мати окремий параметр, де ця кількість готова; саме з величин `divsBefore` обирається результат-максимум.

Теоретично, аргументів `primeIdx`, `maxDeg`, `prodBefore` та `divsBefore` досить. Обмежувати перебір лише числами, меншими *N*, можна умовою `prodBefore <= N`. Але якщо *N* близьке до межі типу, дія `prodBefore * primes[primeIdx]` може давати переповнення, внаслідок чого або програма завершуватиметься аварійно, або умова `prodBefore <= N` даватиме неправильний результат. Цих проблем можна уникнути, якщо ввести ще один аргумент:

- `leftOfN` — «те, що лишилось від *N*»; при початковому нерекурсивному виклику надається значення *N* зі вхідних даних, у ході рекурсії `leftOfN` ділиться (цілочисельно) на `primes[primeIdx]` (скажімо, у тому ж прикладі «2^4, 3^2», `leftOfN` при `primeIdx` = 2 повинно дорівнювати *N* div 144; якби не було переповнень, то завжди виконувалася б рівність `leftOfN` = *N* div `prodBefore`). Тоді умовою продовження рекурсії (заглиблення у рекурсію) можна взяти `leftOfN > 0`.

Тоді саму рекурсію можна організувати, як у лістингу 1. У цій реалізації змінні `globalAnsDivs` (максимальна кількість дільників) та `globalAnsVal` (число з проміжку, на якому вона досягається) глобальні. Початковий нерекурсивний виклик цієї функції може мати, скажімо, вигляд `rec1(0, 100, N, 1, 1)`: починаємо з 0-го простого `primes[0]` = 2, дозволяємо йому великий показник степеню 100, шукаємо для проміжку до *N*, поточний добуток та поточну кількість дільників ще не накопичено (і там, і там 1, а не 0, бо це порожні добутки, а не порожні суми).

```
typedef long long LL; // Суто для скорочення назви типу, непринципово.

void rec1(int primeIdx, int maxDeg, LL leftOfN, LL prodBefore, LL divsBefore) {
  if(divsBefore > globalAnsDivs) { // Якщо поточний результат кращий за досі відомий,
    globalAnsDivs = divsBefore; globalAnsVal = prodBefore;     // то оновити.
  }
  for(int deg = 1; deg <= maxDeg; deg++) { // Поточне просте можна взяти
      //не менше 1 разу (0 разів вже враховано) і не більше разів, ніж попереднє просте.
    prodBefore *= primes[primeIdx]; // Добуток збільшується, а leftOfN зменшується
    leftOfN /= primes[primeIdx]; // відповідно до поточного простого primes[primeIdx].
    if(leftOfN == 0)  return;  // За межами потрібного проміжку, виходимо.
    else rec1(primeIdx +1, deg, prodBefore, divsBefore * (deg+1));
        // Якщо продовжуємо, то: розглядатимемо наступне просте №(primeIdx + 1);
        // обмеження на показник його степеню — кількість цього простого
        // (потрібно для монотонності, див. наслідок 2 твердження 2);
        // prodBefore вже домножений, leftOfN вже зменшений,
        // а divsBefore ще не змінений, тому домножаємо у виклику.
  }
}
```

Лістинг 1 – організація рекурсивного перебору в першому варіанті алгоритму.

Для цього рекурсивного перебору (як і для багатьох інших) важко сформулювати адекватну асимптотичну оцінку часу роботи. Можна сказати, що раз $k_0 \geq k_1 \geq \ldots$, то перебираються монотонно не зростаючі послідовності довжиною $m_{\max}$ (яку можна визначити за наслідком 1 твердження 2), значення яких є цілими числами від 0 до максимально можливого k_0 (позначимо $k_{\max}$). Як відомо, кількість таких послідовностей дорівнює кількості сполучень з повтореннями $\overline{C}^{m_{\max}}_{k_{\max}+1} = C^{m_{\max}}_{k_{\max}+m_{\max}}$ (можливих значень $k_{\max} + 1$, бо від 0 включно до $k_{\max}$ включно; з них вибирається $m_{\max}$ штук; переставляти місцями не можна, бо потрібно дотриматися монотонності; повтори можливі, бо нерівності у $k_0 \geq k_1 \geq \ldots$ нестрогі). Але, як вже вказано у наслідку 1 твердження 2, є

проблеми навіть з тим, як аналітично виразити $m_{\max}$ через N. Можна, звісно, узяти грубу верхню оцінку $m_{\max} \le \log_2 N$, а також вивести аналогічну оцінку $k_{\max} \le \log_2 N$ (якщо піднести просте $p_0 = 2$ до ще більшої степені, результат вийде більшим N). Але з такої грубої оцінки нема користі: вона створює хибне враження, ніби загальна кількість варіантів, які слід перебрати, співмірна з $C_{2\log_2 N}^{\log_2 N}$, що може бути оцінено нерівністю $\frac{N^2}{2\log_2 N+1} < C_{2\log_2 N}^{\log_2 N} < N^2$ (за наслідком з біному Ньютона, $C_p^0 + C_p^1 + \ldots + C_p^p = 2^p$; візьмемо $p = 2 \cdot \log_2 N$ і врахуємо, що доданок $C_{2\log_2 N}^{\log_2 N}$ є максимальним, але лише одним з $2 \cdot \log_2 N + 1$ додатних доданків, які в сумі дають $2^{2\log_2 N} = N^2$).

Ця оцінка сильно завищена, бо в ній не враховано, що досить перебирати лише ті послідовності, де весь добуток (а не лише 2^{k_0}) не перевищує N. Наприклад, візьмемо $N = 10^9$ і застосуємо цю саму оцінку окремо для кожної можливої пари k_0, k_1, де $2^{k_0} \cdot 3^{k_1} \le 10^9$, рахуючи також окремо для кожної такої пари оцінку $m_{\max}$. Наприклад, при $k_0 = k_1 = 1$, максимальне значення послідовності $k_{\max} = 1$, а $m_{\max}$ визначається тим, що $2 \cdot 3 \cdot 5 \cdot 7 \cdot 11 \cdot 13 \cdot 17 \cdot 19 \cdot 23 \cdot 29 \approx 6{,}47 \cdot 10^9 > 10^9$. Отже, якщо не розглядати ні 2 та 3 (вони вже зафіксовані), ні 29 (добуток разом з ним перевищує 10^9), отримаємо $m_{\max} = 7$, $k_{\max} = 1$, $\overline{C}_{1+1}^{7} = C_8^7 = 8$. Аналогічно, для $k_0 = 5$, $k_1 = 3$ максимальне значення подальших k_i визначається $k_1 = 3$, а $m_{\max}$ тим, що $2^5 \cdot 3^3 \cdot 5 \cdot 7 \cdot 11 \cdot 13 \cdot 17 \cdot 19 \approx 1{,}397 \cdot 10^9 > 10^9$, тобто $m_{\max} = 5$, що дає $\overline{C}_{3+1}^{5} = C_8^5 = 56$. Написавши для аналізу решти можливих поєднань k_0, k_1 програму (вона є очевидним узагальненням згаданих міркувань і тут не наводиться), отримаємо, що сумарна за всіма можливими парами k_0, k_1 кількість послідовностей становить 5373, що зовсім мало (як для комп'ютера) і значно менше $\frac{(10^9)^2}{2\log_2 10^9 + 1} \approx 1{,}4 \cdot 10^{16}$.

Більш того, повторивши аналогічні обчислення при $N = 2^{64}$ та $N = 2^{128}$ (що перевищують можливі значення 64-бітового та 128-бітового беззнакових типів), отримаємо сумарно $\approx 6{,}19 \cdot 10^6$ та $\approx 8{,}04 \cdot 10^{11}$ послідовностей відповідно. Ці оцінки теж завищені (враховують значення $2^{k_0} \cdot 3^{k_1}$, але не значення $2^{k_0} \cdot 3^{k_1} \cdot 5^{k_2} \cdot 7^{k_3} \cdot \ldots$, тому теж включають багато послідовностей, які не розглядатимуться, бо добуток перевищує N), тож можна стверджувати, що такий алгоритм досить ефективний при N у межах 64-бітового (виконання практично миттєве) і навіть 128-бітового (результатів реально дочекатися) типів. Прямий експеримент підтверджує це «з запасом»: було взято кілька значень порядку $10^{36} \ldots 10^{38}$ (при $2^{128} \approx 3 \cdot 10^{38}$), і для них час виконання був до 0,1 с.

Модифікація алгоритму для постановки № 2. Якщо говорити лише про правильність та ігнорувати час роботи, можна обійтися невеликими змінами коду відносно лістингу 1: аргумент `maxDeg` стає непотрібним, бо вже не треба забезпечувати монотонність показників (див. зауваження після твердження 2); при оновленні відповіді (`globalAnsDivs = divsBefore`) слід перевірити, крім `divsBefore > globalAnswer`, також `prodBefore >= M`; показник степені `deg` слід перебирати, починаючи з 0, а не 1 (там само вже наведено приклад, коли відповідь досягається на числі $50 = 2^1 \cdot 3^0 \cdot 5^2$).

Але час роботи так модифікованого алгоритму збільшується кардинально. Головним чином (але не тільки) через можливість ситуації `deg = 0`, коли всі, крім `primeIdx`, аргументи (зокрема й `leftOfN`, від якого залежить вихід з рекурсії) наступного рівня рекурсії дорівнюють відповідним аргументам попереднього рівня, тож заглиблень у рекурсію може бути дуже багато.

Це можна у значній мірі оптимізувати, використавши такі засоби.

По-перше, спробувати розв'язати задачу в постановці № 1; якщо відповідь потрапить у проміжок від M до N, то задача вирішена також і в постановці № 2. Звісно, так буде далеко не завжди.

По-друге, якщо $N - M$ досить мале, спробувати розв'язати задачу очевидним способом, тобто розкласти на прості множники кожне з чисел від M до N, знайти для кожного кількість дільників (згідно (3)) і вибрати максимум.

По-третє, ввести, «симетрично» до `leftOfN`, аргумент `leftOfM` («що лишилось від M»); якщо зменшувати нижню межу `leftOfM` діленням із заокругленням догори, наприклад, `leftOfM = (leftOfM + primes[primeIdx] - 1) / primes[primeIdx]`, то умову виходу з рекурсії можна змінити з «`leftOfN == 0`» на, наприклад, «`leftOfM > leftOfN`»; це може наставати помітно раніше, ніж «`leftOfN == 0`».

По-четверте, використати прийом відтинань гілок перебору за верхніми оцінками з методу гілок та меж (описаний в [7], [8, розд. 4.2], [9, розд. 11.3] та багатьох інших джерелах). Тобто, в кожному рекурсивному виклику будувати для поточних `prIdx`, `leftOfM`, `leftOfN`, `prodBefore`, `divsBefore` деяку евристичну оцінку вигляду «на вкладених рівнях рекурсії, набрана на зовнішніх рівнях рекурсії кількість дільників `divsBefore` не може збільшитися більш як у T разів». Тобто, потрібна додаткова функція, яка прийматиме як аргументи, наприклад, `prIdx`, `leftOfN` (чи деяку іншу підмножину параметрів рекурсії) і вертатиме деяке T, для якого виконується це твердження. Далі, якщо $\texttt{divsBefore} \cdot T \le \texttt{globalDivs}$, то поточну гілку рекурсії можна обривати як безперспективну. Було розглянуто два варіанти такої функції. (А) Коли вже дійшли до індексу `prIdx` (кількості менших простих вже зафіксовані на зовнішніх рівнях рекурсії), в числах проміжку `leftOfM` … `leftOfN` може міститися не більш як $q = \text{floor}(\log_p \texttt{leftOfN})$ нових простих, де $p = \texttt{primes[prIdx]}$, floor — заокруглення донизу (q не може бути більшим, ніж вказана кількість, бо добуток перевищив би `leftOfN`; уточнимо, що під цим q мається на увазі кількість, що враховує кратність, наприклад, $5^2 \cdot 7^1$ рахується як $2 + 1 = 3$). Враховуючи наслідок твердження 1, кількість дільників була б найбільша, якби ці не більш ніж q множників були різними простими; тоді кількість дільників була б $(1 + 1)^q = 2^q$. Таку оцінку відносно легко рахувати, вона є оцінкою згори (реальна кількість дільників справді не перевищує 2^q), але вона завищена: якщо хоча б частина простих різні, то хоча б частина з них більші за `primes[primeIdx]`, і їх, імовірно, міститься менша кількість, ніж q, пораховане через логарифм. Тому є обґрунтовані сумніви, чи добре така оцінка відтинатиме безперспективні гілки. (Б) В якості функції оцінки можна використати алгоритм з лістингу 1 для постановки № 1; аргумент `leftOfM` при цьому слід просто ігнорувати, в якості `primeIdx` передавати значення `primeIdx` з поточного виклику рекурсії перебору для постановки № 2; для такого змішаного використання двох рекурсій доведеться окремо мати змінні `globalAnsDivs2` та `globalAnsVal2`, що стосуються перебору для постановки № 2, і окремо зміні `globalAnsDivs1` та `globalAnsVal1`, що стосуються «вкладеного» перебору для постановки № 1. Це не буде щоразу те саме повне розв'язання тієї самої задачі способом з лістингу 1, бо на зовнішніх рівнях рекурсії перебору для постановки № 2 могли розглядатися послідовності показників, які були б відкинуті алгоритмом з лістингу 1, але можуть бути корисними для постановки № 2. Це даватиме верхню оцінку, бо якщо для всіх можливих продовжень не вдалося знайти кращого, ніж вже відомий (`globalAnsDivs2`) розв'язок, то його тим паче нема серед лише тієї частини можливих продовжень, які враховують межу `leftOfM`. Крім того, аналогічно оптимізації «по-перше» цього переліку, можлива ситуація, коли шукали найкращий серед усіх способів, ігноруючи `leftOfM`, а виявилося, що найкращий серед усіх задовольняє це обмеження; тоді цей спосіб тим паче найкращий серед тих, на які накладене це обмеження, тобто може знайтися не просто оцінка, а точна відповідь для відповідної гілки перебору. Однак, перебір з лістингу 1, хоч і швидкий як для перебору, все ж повільний для використання його в

кожному виклику рекурсії перебору для постановки № 2. Тому потрібен певний компроміс, коли варто використовувати такий повільний спосіб знаходження верхньої оцінки, а коли ні. Це питання поки що вирішене лише евристично, без чіткої аргументації.

По-п'яте, ідею верхніх оцінок та відкидання безперспективних варіантів можна використати не лише в рекурсивному переборі, а також і в пошуку кількості дільників поточного числа через розкладення. Традиційно (описано в [10, задачі 1.1.22–1.1.23] та багатьох інших джерелах), розкладення числа n передбачає перебір усіх потенційних дільників до $\sqrt{n}$ (включно, якщо корінь цілий), із традиційною оптимізацією «якщо число n вже зменшилося від ділень на деякі менші дільники, то далі можна брати за межу корінь зі вже зменшеного n». Однак, все це виходить із припущення «дане число n, треба за будь-яку ціну знайти точну кількість його дільників». Зараз ситуація дещо інша: коли видно, що кількість дільників числа n точно не буде більшою за максимальну відому кількість дільників `globalAnsDivs2` числ*а* `globalAnsVal2`, то ні розкладення n, ні точне значення кількості дільників n вже не важливі, тож цикл перебору дільників можна обірвати. Для скорочення формул, позначимо `globalAnsDivs2 / divs` як d (у скільки разів має збільшитися поточна кількість дільників `divs`, щоб їх стало більше, ніж наразі максимальна кількість).

З міркувань, аналогічних поясненню верхньої оцінки (А) абзацу «по-четверте», і вживаючи q в тому ж смислі, можна сказати, що для кількості дільників має виконатися

$$2^q \geq d; \tag{4}$$

при цьому, щоб ці q дільників помістилися в n, для мінімального з простих дільників (позначимо його як p) має виконуватися

$$p^q \leq n. \tag{5}$$

Оскільки p, q та n додатні, а функція $f(x) = x^q$ при додатних x та додатному $q = \text{const}$ монотонна, нерівність (5) рівносильна

$$p \leq n^{1/q}. \tag{6}$$

Логарифмуючи (4) за основою 2, отримуємо

$$q \geq \log_2 d; \tag{7}$$

взявши обернену величину від обох частин (7) (знак при цьому змінюється на протилежний), отримуємо

$$\frac{1}{q} \leq \frac{1}{\log_2 d}. \tag{8}$$

Враховуючи $n \geq 1$, (6), (8) та монотонність показникової функції, отримуємо

$$p \leq n^{\frac{1}{\log_2 d}}. \tag{9}$$

А враховуючи, що традиційне обмеження «до кореня» теж лишається в силі, маємо

$$p \leq \min(\sqrt{n}, n^{\frac{1}{\log_2 d}}). \tag{10}$$

При цьому слід розуміти, що:

1) при досить великих d, (9) накладає на p жорсткіше обмеження, ніж корінь, але при $d < 4$ меншим є, навпаки, корінь; тому, (10) обмежує перебір краще, ніж (9);
2) коли при пошуку розкладення вдається знайти новий дільник, зменшується не лише n, а й d (за рахунок збільшення `divs` при сталому `globalAnsDivs2`), і значення $n^{\frac{1}{\log_2 d}}$ при цьому, як правило, збільшується; тому, при знаходженні нового

дільника поточного числа*a*, важливо оновлювати верхню межу перебору потенційних дільників, щоб не втратити правильність (не загубити розв'язок).

По-шосте, «по-четверте» та «по-п'яте» істотно залежать від оцінок, дієвість яких істотно залежить від наразі відомого `globalAnsDivs2`, тому важливо переставити рекурсивні виклики з тими самими `leftOfM`, `leftOfN`, `prodBefore`, `divsBefore` (відповідні ситуації «поточне просте число береться в 0-му степені») на останнє місце.

По-сьоме, для якнайранішого заповнення `globalAnsDivs2` досить великою правильною оцінкою, варто, якщо не спрацювало «по-перше», то розв'язати задачу у постановці № 1 для проміжку «від 1 до $N - M + 1$», знайти число з проміжку «від M до N», кратне `globalAnsVal1` проміжку «від 1 до $N - M + 1$» (воно гарантовано існує й лише одне, дорівнює (N div `globalAnsVal1`) · `globalAnsVal1`) та знайти кількість його дільників за (3). При не дуже малих $N - M + 1$ слід очікувати, що це число має чимало дільників, бо воно кратне `globalAnsVal1`, яке є надскладеним.

По-восьме, ідею «замість рекурсії, розкладати на множники кожне з чисел діапазону й рахувати кількість дільників» можна використати не лише на початку (як у «по-друге»), а й всередині рекурсії. Це особливо корисно, коли проміжок між початковими M та N завеликий для повного перебору всіх його чисел, але після кількох ділень у рекурсивних викликах `leftOfM` та `leftOfN` стають досить близькими. Такі перевірки проміжку саме всередині рекурсії гарантують, що перевірятися будуть не всі підряд числа початкового проміжку від M до N, а лише ті з них, які кратні `prodBedore`, набраному на зовнішніх рівнях рекурсії; оскільки (завдяки «по-шосте») `prodBedore` майже завжди містить добуток невеликих простих, у таких чисел досить багато дільників. Щоправда, так можуть виникати непродуктивні витрати на перевірку одного й того ж числа в різних гілках рекурсії; але цього можна уникнути, запам'ятовуючи вже перевірені числа в одній глобальній множині (наприклад, у термінах C++ STL, `set<long long> alreadyLooked`).

Врахування всіх цих (і, насправді, ще деяких) ідей дало код, що має обсяг, який вже не варто включати у друковане видання; охочі можуть ознайомитися з ним за посиланням https://ejudge.ckipo.edu.ua/maxDivsInRange.cpp. Асимптотично оцінити час його роботи не здається можливим, бо на рекурсивний перебір накладено дуже багато різних засобів його скорочення. Але експериментально він показав себе значно кращим за очевидний підхід (пошук кількості дільників кожного числа окремо) чи за початкову версію з мінімальними відносно лістингу 1 змінами: час роботи цих алгоритмів перевищує десятки секунд вже для багатьох вхідних даних порядку $N \approx 10^7 \ldots 10^8$, а розроблений алгоритм має менший час роботи при $N \approx 10^{18}$.

У таблиці 1 наведено результати дослідження часу роботи реалізації розробленого алгоритму мовою C++ (Windows, компілятор g++, рівень оптимізацій «`-O2`») на комп'ютері тактової частоти 3,2 ГГц. Було проведено 192 000 вимірювань (по 32 000 у кожній з 6 груп). N завжди вибиралося випадково (з рівномірним розподілом) з проміжку $10^{15} \le N \le 10^{18}$. Для вибору M було використано шість різних способів, описаних у написі при таблиці; саме такі проміжки були вибрані евристично, без чіткої аргументації. Рядки були вибрані з тих міркувань, щоб наголосити в першу чергу на тривалості роботи в поганих випадках, але середнім випадкам теж приділити певну увагу. Час роботи алгоритму для кожної окремо взятої пари M, N вимірювався процесорний (засобами `GetProcessTime`), усереднений з двох спроб. Початкова похибка вимірювань кожної спроби становила 1 тік ≈ 0,015 с. Ці фактори спричинили велику кількість значень 0,015 с та 0,008 с (які фактично означають «1 тік» та «одна спроба менше 1 тіку, інша 1 тік»).

Таблиця 1 — експериментально виміряний час роботи розробленого алгоритму для розв'язування задачі в постановці № 2, при випадкових (з рівномірним розподілом) $10^{15} \le N \le 10^{18}$; стовпці (1)–(6) означають спосіб вибору M: (1) $10^9 \le M < 0{,}75 \cdot N$; (2) $0{,}75 \cdot N \le M < 0{,}99 \cdot N$; (3) $0{,}99 \cdot N \le M < 0{,}999999 \cdot N$; (4) $0{,}999999 \cdot N \le M < N - 10000$; (5) $N - 10000 \le M < N$; (6) $M = N$; у способах (1)–(5) значення M вибиралося в указаному проміжку випадково з рівномірним розподілом. Всі значення часу подані в секундах з заокругленням до тисячних.

	(1)	(2)	(3)	(4)	(5)	(6)
Середнє арифметичне	0,009	0,010	0,030	0,413	0,022	0,231
Середнє квадратичне	0,010	0,011	0,038	0,464	0,034	0,668
Максимальне значення	0,015	0,046	0,561	1,661	0,897	4,602
10-е згори значення	0,015	0,039	0,463	1,396	0,507	4,579
100-е згори значення	0,015	0,031	0,179	1,162	0,226	4,274
320-е згори значення (верхній процентиль)	0,015	0,023	0,124	1,022	0,117	3,447
3200-е згори значення (верхній дециль)	0,015	0,015	0,046	0,702	0,031	0,639
8000-е згори значення (верхній квартиль)	0,008	0,015	0,031	0,538	0,023	0,086
16000-е згори значення (медіана)	0,008	0,008	0,023	0,328	0,015	0,015
24000-е згори значення (нижній квартиль)	0,008	0,008	0,015	0,257	0,015	0,008

Мінімальність усіх характеристик у стовпці (1) легко пояснити тим, що для широких діапазонів відповідь для постановки № 1 часто виявляється остаточною для постановки № 2. Те, що величини стовпця (4) або максимальні, або другі максимальні у своїх рядках, легко пояснити тим, що для таких розмірів проміжку не дуже добре працюють хоч оптимізації на базі алгоритму для постановки № 1 (для них проміжок завузький; те, що потрапляє в нього, часто сильно відрізняється від максимального для проміжку від 1), хоч оптимізації на базі перебору від `leftOfM` до `leftOfN` (для них проміжок надто широкий). Особливість стовпця (6), де медіана та квартилі досить малі, але найбільші в своїх рядках максимум, 10-е, 100-е значення та верхній процентиль, легко пояснити тим, що з одного боку $M = N$ означає, що досить розкласти всього одне число (тому середні, медіана та квартилі досить малі), але іноді, якщо це число просте чи є добутком двох простих, близьких до кореня, для з'ясування цього слід виконати велику кількість ітерацій, і вони не відкидаються ніякими оцінками. Отже, результати експерименту відповідають очікуванням. Також вони свідчать про результативність ідей «по-п'яте» (скорочення перебору дільників на основі формули (10)) та «по-сьоме», раз аналіз проміжків у сотні чисел відбувається швидше, ніж одночислових, хоча матсподівання кількості простих (та добутків двох простих, близьких до кореня) на проміжку, очевидно, збільшується зі збільшенням довжини проміжку.

Модифікація алгоритму для постановки № 3а. Тут досить вчинити, як зазначено відразу після твердження 2, тобто робити всі ті самі дії, що у простому й ефективному варіанті цього алгоритму для постановки № 1, але викресливши K з послідовності простих. Час роботи при цьому може або зменшитися (за рахунок того, що замість самого K та всіх подальших простих використовуються більші наступні прості, й `leftOfN` зменшується швидше, від чого зменшується загальна кількість варіантів, що перебираються), або лишиться незмінним (навіть якщо вказані причини не спрацювали, ніщо не збільшує час роботи у порівнянні з постановкою № 1).

Нема принципових проблем, що істотно перешкоджали б поєднати постановки № 3а та № 2 (коли і проміжок від M до N, і розглядаються лише чи*с*ла, не кратні простому K). Однак, через поєднання технічної складності з сумнівністю значимості результату, таке поєднання не досліджувалося детально. Очевидно, крім виключення K з переліку простих, слід також змінити перебори від `leftOfM` до `leftOfN` так, щоб відкидати кратні K. Але є й менш очевидні моменти; зокрема, доведеться якось змінити оптимізацію «по-сьоме», бо навіть якщо «`globalAnsVal1` проміжку від 1 до $N - M + 1$» знайдене, враховуючи відкидання кратних K, кратність K може (але не зобов'язана) з'явитися при обчисленні (N div `globalAnsVal1`) · `globalAnsVal1`. Це робить час роботи такої недодослідженої модифікації алгоритму ще менш передбачуваним.

Модифікація алгоритму для постановки № 3б. Покажемо, що для постановки № 3б не можна просто відкинути якісь із простих чисел. Розглянемо $K = 40 = 2^3 \cdot 3^0 \cdot 5^1$ і побудуємо (очевидним неефективним алгоритмом, не вартим детальнішої згадки) послідовність «рекордних» за кількістю дільників чисел. (Аналогічно (1) з означення 2, але з умовою ігнорування чисел, кратних $K = 40$; формально це можна виразити як $\left\{n \mid (n \bmod 40 \neq 0) \wedge \left(\forall j_{(1 \le j < n) \wedge (j \bmod 40 \neq 0)} \left(\tau(j) < \tau(n)\right)\right)\right\}$ у порядку зростання.) Перші 16 таких чисел наведено у таблиці 2, де значення розподілені у групи за причиною, чому це число не кратне 40: чи тому, що не кратне $5 = 5^1$, чи тому, що не кратне $8 = 2^3$; чи*с*ла, не кратні ні 5, ні 8 включені в обидві групи.

Таблиця 2 — перші 16 значень, «рекордних» за кількістю дільників серед чисел, не кратних 40.

не кратні 5	2	4	6	12	24	36	48		144	168		336			1008	
не кратні 8	2	4	6	12		36		60			180		420	900		1260

Бачимо, що для значень, «рекордних» серед не кратних 40, ця не кратність 40 забезпечується то не кратністю $5 = 5^1$, то не кратністю $8 = 2^3$, і нема очевидного закону, який виражав би, коли яка з цих умов має місце. Природно, щоб аналогічна ситуація траплялася і для деяких інших складених, в тому числі й тих, розкладення яких мають більше простих у ненульових степенях. Тобто, гарантовано відкинути якісь із простих, що входять у розкладення K в ненульових степенях, не здається можливим. Тому, сформулюємо інші знайдені обмеження, корисні для оптимізацій перебору (плануючи його загальну логіку схожою на перебір з лістингу 1, але зі значними відмінностями). Почнемо з простих, а все ж корисних означення 3 та тверджень 3 і 4.

Означення 3. Будемо позначати $z_0, z_1, \ldots$ показники степенів, з якими прості $p_0 = 2$, $p_1 = 3$, $p_2 = 5, \ldots$ входять у розкладення K, кратність якому заборонена: $K = p_0^{z_0} \cdot p_1^{z_1} \cdot \ldots$ Очевидно, значення $z_0, z_1, \ldots$ цілі невід'ємні (можуть дорівнювати 0). □

Твердження 3. Якщо в деякому рекурсивному виклику `prodBefore` кратний K, всю поточну гілку можна обривати виходом з рекурсії (чи, що практично те само, не заходити туди, перевіряючи кратність не всередині виклику, а ззовні (перед)).

Доведення. Перебір організований так, що якщо `prodBefore` поточного виклику кратний K, він буде кратний і в усьому піддереві рекурсії; все це треба проігнорувати. ■

Твердження 4. Якщо рекурсивний перебір розглядає випадок $k_j < z_j$ (бере просте p_j у степені, меншій, ніж степінь його входження у K; технічно це визначається тим, що локальна змінна `deg` на рівні рекурсії `prIdx` = j набуває значення, менше z_j, і саме з ним відбувається рекурсивний виклик; все це можливо лише для тих простих, де $z_j > 0$, тобто p_j є дільником K), то на вкладених рівнях рекурсії можна застосовувати перебір з лістингу 1, призначений для постановки № 1.

Доведення. Ми плануємо організувати рекурсію для постановки № 3б так, щоб мати всі потрібні для перебору з лістингу 1 параметри `prIdx`, `leftOfN`, `prodBefore`, `divsBefore` — отже, маємо технічну можливість у одному з вузлів однієї рекурсії запустити іншу рекурсію (тобто, організувати перебір так, щоб з деяких вузлів д*е*рева нової рекурсії далі продовжувалися піддер*е*ва старої рекурсії, а не гілки тієї (нової) рекурсії, яка привела до цього вузла).

Єдина причина, навіщо нова рекурсія має відрізнятися від лістингу 1 — треба перебирати лише ч*и*сла, не кратні K. На всіх подальших рівнях розглядатимуться лише більші прості, й результат домноження `prodBefore` на них ніколи не стане кратним K, бо степінь p_j так і лишатиметься $p_j^{k_j}$, що менше $p_j^{z_j}$, потрібного для кратності K. В цих умовах, ніщо не заважає застосувати до степенів подальших (більших) простих доведення, цілком аналогічне доведенню твердження 2. ■

Примітка до твердження 4. Йдеться лише про монотонність **подальших** показників $k_{j+1} \ge k_{j+2} \ge \ldots$ **між собою**; відношення ж між k_j та k_{j+1} може бути будь-яким ($k_j > k_{j+1}$, $k_j = k_{j+1}$ чи $k_j < k_{j+1}$). □

А як перебирати, коли ще не настав перехід, описаний у твердженні 4? Щоб визначити це, сформулюємо кілька тверджень-аналогів твердження 2, які теж дозволять обмежити перебір (гірше, ніж твердження 2, але хоч якось).

Твердження 5. Для розв'язування задачі у постановці № 3б досить перебирати ч*и*сла, в розкладенні яких, для будь-яких простих p_a, p_b, таких, що $p_a < p_b$ та $z_a = 0$ (K не кратне p_a), завжди (незалежно від значення z_b) виконується $k_b \le k_a$.

Доведення. Аналогічно доведенню твердження 2, якщо розкладення деякого числ*а* містить $p_a^{k_a} \cdot p_b^{k_b}$, де $p_a < p_b$ та $k_a < k_b$, то заміна $p_a^{k_a} \cdot p_b^{k_b}$ на $p_a^{k_b} \cdot p_b^{k_a}$ зменшує значення, не змінюючи кількості дільників, і нема потреби розглядати більше (початкове) E, коли ту ж кількість дільників можна здобути й на меншому (отриманому цією заміною) E'. Але треба ще показати неможливість поєднання умов цього твердження із ситуацією $(E \vdots K) \wedge (E' \nvdots K)$, тобто «отримане внаслідок заміни E' стає кратним K і його слід відкинути, хоча початкове E не кратне K». А неможливо це тому, що вказана заміна збільшує показник лише степеню з основою p_a, для якої $z_a = 0$ (тобто, якій K не кратне), тож збільшення показника саме цього степеню не може створити кратності K; зменшення якого б не було показника теж не може створити кратності. ■

Примітка до твердження 5. Умова $z_a = 0$ важлива, бо при $z_a > 0$ існують приклади, коли менше E' стає кратним K, хоча початкове E не було. Причому, це можливо як у випадку $z_b = 0$, так і $z_b > 0$. Наприклад, $E = 18 = 2^1 \cdot 3^2$ ($k_0 = 1$, $k_1 = 2$) не можна зменшувати до $E' = 12 = 2^2 \cdot 3^1$ ($k_0 = 2$, $k_1 = 1$) ні при $K = 4 = 2^2$ ($z_0 = 2$, $z_1 = 0$), ні при $K = 12 = 2^2 \cdot 3^1$ ($z_0 = 2$, $z_1 = 1$), бо в обох випадках заміна $2^1 \cdot 3^2$ на $2^2 \cdot 3^1$ призводить до того, що число стає кратним K і його слід відкинути. І (як можна встановити перебором), хоч при $K = 4$, хоч при $K = 12$, саме 18 з 6-ма дільниками є відповіддю для всіх N з проміжку $18 \le N < 30$, тож його пропускати не можна. □

Твердження 6. Якщо розкладення K містить $p_a^{z_a} \cdot p_b^{z_b}$ при $p_a < p_b$ та $1 \le z_a \le z_b$, то $p_a^{z_a}$ можна відкинути (розв'язати задачу для $K' = K / p_a^{z_a}$), і результат буде тим самим.

Доведення. Очевидно, що кожне число потрапляє в одну з чотирьох груп: (А) не кратні ні $p_a^{z_a}$, ні $p_b^{z_b}$; (Б) кратні $p_a^{z_a}$, але не кратні $p_b^{z_b}$; (В) кратні $p_b^{z_b}$, але не кратні $p_a^{z_a}$; (Г) кратні і $p_a^{z_a}$, і $p_b^{z_b}$. Враховуючи, що p_a та p_b — різні прості, кратними $p_a^{z_a} \cdot p_b^{z_b}$ є всі ч*и*сла з групи (Г) і лише вони. Розглянемо, чим відрізняються задачі в

постановці № 3б для початкового K і для зменшеного $K' = K / p_a^{z_a}$. Чи*с*ла з групи (Г) і там, і там треба ігнорувати; чи*с*ла з груп (А) та (Б) і там, і там не треба ігнорувати (саме серед них, одних чи інших, буде шукане). Отже, створити відмінність могли б лише чи*с*ла з групи (В), які треба ігнорувати для зменшеного K', але не треба для початкового K. Але належність чис*л*а групі (В) означає, що воно містить у розкладенні $p_a^{k_a} \cdot p_b^{k_b}$, де $k_a < z_a$ (це і є не кратністю $p_a^{z_a}$) та $k_b \ge z_b$ (це і є кратністю $p_b^{z_b}$). Оскільки за припущенням цього твердження $z_a \le z_b$, маємо $k_a < z_a \le z_b \le k_b$, звідки $k_a < k_b$. Аналогічно доведенню твердження 2, вигідно замінити $p_a^{k_a} \cdot p_b^{k_b}$ на $p_a^{k_b} \cdot p_b^{k_a}$ (лишаючи степені інших простих, якщо такі є, незмінними): внаслідок заміни число стає меншим, маючи ту саму кількість дільників. Щоправда, тепер слід перевірити додаткову умову: раптом зменшене число слід відкинути, бо воно кратне початковому K? Але це не так, враховуючи $k_a < z_b$ (слідує з виписаного на кілька рядків вище ланцюжка нерівностей). Тобто, не розгляд чисел з групи (В) не заважає правильності (для кожного з них існує інше число, яке буде розглянуте й дасть не гірший результат). ■

Наслідок 1 твердження 6. Якщо розкладення K містить багато простих і ситуація $p_a < p_b$ та $1 \le z_a \le z_b$ з'являється кілька разів для різних пар a, b, твердження 6 можна застосувати багатократно, і гарантовано прийти до однієї з двох ситуацій: або ненульовий показник лишиться у степені тільки одного простого, або послідовність показників z_j (рахуючи лише серед ненульових) стане строго спадною (лише ці випадки роблять поєднання $p_a < p_b$ та $1 \le z_a \le z_b$ неможливим ні для яких a, b). Наприклад, $K = 3^3 \cdot 5^2 \cdot 11^5$: прибрати 3^3, бо є 11^5; прибрати 5^2, бо є 11^5; лишиться 11^5. Та, наприклад, $K = 2^5 \cdot 3^3 \cdot 5^4 \cdot 7^1 \cdot 13^1$: прибрати 3^3, бо є 5^4; прибрати 7^1, бо є 13^1; лишиться $2^5 \cdot 5^4 \cdot 13^1$. ■

Наслідок 2 твердження 6. Враховуючи наслідок 1 твердження 6, навіть не маючи конкретного K та його розкладення, а лише $K_{\max}$ (таке, що розглядатимуться лише K з проміжку $2 \le K < K_{\max}$), можливу кількість **різних** простих у розкладенні зменшеного K' можна оцінити згори так: послідовно будувати $2^1 = 2$, $2^2 \cdot 3^1 = 12$, $2^3 \cdot 3^2 \cdot 5^1 = 360$, $2^4 \cdot 3^3 \cdot 5^2 \cdot 7^1 = 75600$, …, доки добуток не перевищить $K_{\max}$; коли перевищить, повернутися на один крок назад. Адже, враховуючи наслідок 1 та монотонність (при додатних аргументах) добутку, степеневої та показникової функцій, саме такими є мінімальні можливі добутки степенів 1-го, 2-х, 3-х, 4-х, … простих, що задовольняють наслідку 1 твердження 6. Аналогічно наслідку 1 твердження 2, не здається можливим виразити це аналітично скільки-небудь точно, а, завищуючи оцінку, можна говорити про не більш як $\sqrt{2 \cdot \log_2(K_{\max})}$ штук (логарифм з міркувань, аналогічних згаданим у наслідку 1 твердження 2; корінь — з того, що j різних простих повинні входити у розкладення зі степенями j, $j-1$, …, 1, сума становить $j(j+1)/2$, тож зворотня дія потребує (приблизно) домноження на 2 й кореню). ■

Саме тому, що твердження 5 не працює при $z_a > 0$, важливо, що твердження 6 та його наслідки дозволяють побудувати на основі K рівноцінне йому K', яке містить мало різних простих: це зменшує кількість можливих місць порушення твердження 5). Але корисність твердження 6 та його наслідків цим не обмежується: крім того, вони також дають можливість довести наступне твердження 7.

Твердження 7. Нехай перед початком основного перебору до K застосовані дії, описані в наслідку 1 твердження 6, й перебір відбувається для зменшеного K'. Нехай перебір організований з урахуванням твердження 4. Нехай у розкладення K' входять степені $p_a^{z_a} \cdot p_b^{z_b}$, де $p_a < p_b$, $z_a > 0$, $z_b > 0$. Нехай рівень рекурсії, що вибирає показник

степеню простого p_a, розглядає випадок $k_a \geq z_a$. Тоді діапазон перебору показників степеню простого p_b можна обмежити умовою $k_b \leq k_a$.

Доведення. Припустимо, ніби розглядаємо $k_b > k_a$. Оскільки маємо не довільне K, а K', отримане за наслідком 1 твердження 6, то з $p_a < p_b$, $z_a > 0$, $z_b > 0$ випливає $z_a > z_b$. Отже, можна зібрати ланцюжок нерівностей $k_b > k_a \geq z_a > z_b$, з якого випливає $k_b > z_b$. Тобто, степінь простого p_b не створює некратності K'. Оскільки перебір організований з урахуванням твердження 4, потрібна некратність K' не могла з'явитися ще раніше (при ще меншому простому): якби вона з'явилася вже там, то показники степенів простих p_a та p_b підбиралися б уже перебором з лістингу 1, а не тим перебором, який враховує K. Значить, можливі два варіанти: або потрібна некратність K' надалі так і не утвориться, або утвориться завдяки тому, що є хоча б одне z_c, таке, що $c > b$ та $z_c > 0$, і для нього пізніше буде вибрано $k_c < z_c$. Якщо некратність K' так і не утвориться, нема потреби перебирати зовсім безперспективні варіанти. Якщо утвориться за рахунок вказаної причини, ніщо не заважає повторити прийом з доведення твердження 2, бо некратність K' і так забезпечується степенем $k_c < z_c$ третього (не p_a і не p_b) простого p_c, і заміна $p_a^{k_a} \cdot p_b^{k_b}$ на $p_a^{k_b} \cdot p_b^{k_a}$ на це ніяк не впливає. ■

Твердження 8. Підсумовуючи твердження 3—7, перебір для постановки № 3б можна і варто організувати так:

1. Розпочати з того, що перетворити задане у вхідних даних даних K згідно наслідку 1 твердження 6, й надалі працювати з K'. При цьому заодно формується масив `Z_` (показники $z_0, z_1, \ldots$ згідно означення 3), який робиться глобальним, щоб могти легко доступатися до нього з рекурсії.
2. Мати дві окремі рекурсії, одна цілком відповідає лістингу 1 й використовується тоді, коли вже забезпечено некратність K, інша — коли ще не збезпечено.
3. Друга рекурсія має всі ті самі параметри, що перша, та параметр `maxDegForK` (максимальний показник степені для тих простих, де $z_j > 0$; використовується там, де виконуються умови твердження 7), а також використовує масив `Z_`.
4. Друга рекурсія може:
   а) Обривати свої гілки згідно твердження 3.
   б) Продукувати замість власних викликів виклики першої рекурсії, згідно з твердженням 4; максимальний степінь `deg` при цьому дорівнює `Z_[prIdx]` – 1.
   в) Продукувати власні виклики для тих простих, які входять у K' (`Z_[prIdx]` > 0), значення `deg` мають перебувати у проміжку від `Z_[prIdx]` до `min(maxDegForK, maxDeg)` (обидві межі включно), бо саме цей проміжок диктується твердженнями 5 та 7, якщо врахувати, що `deg` < `Z_[prIdx]` підпадають під твердження 4.
   г) Продукувати власні виклики для тих простих, які не входять у K' (`Z_[prIdx]` = 0), значення `deg` мають перебувати у проміжку від 0 до `maxDeg` (обидві межі включно, але 0 означає припинення подальших заглиблень), бо це визначається твердженням 5.

Доведення, як такого, нема, бо воно зводиться до того, щоб перевірити, що наведені пункти покривають собою всі можливі ситуації. ■

Отриманий код завеликий для включення у друковане видання; охочі можуть ознайомитися з ним тут: https://ejudge.ckipo.edu.ua/maxDivsExceptDivK.cpp.

Оцінити асимптотично час виконання цього алгоритму не здається можливим (що природно, враховуючи, що він включає в себе алгоритм з лістингу 1, адекватно оцінити який теж не вдалося), тому знову наведемо експериментальні оцінки.

Результати дослідження тривалості роботи розробленого алгоритму наведені у таблиці 3. Було проведено 180 000 (по 30 000 у кожній з 6 груп) вимірювань часу роботи реалізації цього алгоритму. Мова, компілятор, комп'ютер, засоби вимірювання часу та статистичні характеристики ті самі, що в табл. 1.

Таблиця 3 – експериментально виміряний час роботи розробленого алгоритму для розв'язування задачі в постановці № 3б, при випадкових $10^{35} \le N \le 3 \cdot 10^{35}$; стовпці означають спосіб вибору K: (1) $2 \le K \le 50$; (2) $51 \le K \le 1000$; (3) $1001 \le K \le 2 \cdot 10^9$; (4) K мають вигляд $2^{m_0} \cdot 3^{m_1} \cdot 5^{m_2} \cdot 7^{m_3} \cdot 11^{m_4}$; (5) вигляд $2^{m_0} \cdot 3^{m_1} \cdot 5^{m_2} \cdot 7^{m_3} \cdot 11^{m_4} \cdot 13^{m_5} \cdot 17^{m_6}$; (6) вигляд $2^{m_0} \cdot 5^{m_2} \cdot 11^{m_4} \cdot 17^{m_6}$.

	(1)	(2)	(3)	(4)	(5)	(6)
Середнє арифметичне	0,038	0,061	0,041	0,094	0,077	0,080
Середнє квадратичне	0,041	0,071	0,042	0,111	0,089	0,099
Максимальне значення	0,140	0,226	0,234	1,248	1,404	1,263
10-е згори значення	0,125	0,203	0,172	0,390	0,250	0,639
100-е згори значення	0,094	0,179	0,070	0,288	0,218	0,304
300-е згори значення (верхній процентиль)	0,086	0,171	0,070	0,250	0,195	0,249
3000-е згори значення (верхній дециль)	0,055	0,117	0,054	0,171	0,133	0,164
7500-е згори значення (верхній квартиль)	0,047	0,078	0,047	0,125	0,101	0,101
15000-е згори значення (медіана)	0,039	0,047	0,039	0,094	0,078	0,078
22500-е згори значення (нижній квартиль)	0,031	0,039	0,039	0,039	0,039	0,023

Результати та висновки

Для кожного з трьох описаних у статті варіантів постановки задачі розроблено алгоритм, значно ефективніший за очевидні. Хоча для жодного з розроблених алгоритмів не вдалося побудувати адекватну асимптотичну оцінку, але швидкодія кожного з них досліджена експериментально. Для постановок № 1 та № 3а час не перевищував десятих секунди при $N \approx 10^{38}$; для постановки № 2, час не перевищував кількох секунд при $N \approx 10^{18}$ (детальні результати подано у таблицях, наведених у статті); для постановки № 3б час не перевищував кількох секунд при $N \approx 10^{35}$.

Список використаної літератури:

1. Сайт «The On-Line Encyclopedia of Integer Sequences». Режим доступу https://oeis.org/A002182
2. Стаття «Сверхсоставное число» в російськомовному розділі вікіпедії. Режим доступу https://ru.wikipedia.org/wiki/Сверхсоставное_число
3. Стаття «Divisor_function» в англомовному розділі вікіпедії. Режим доступу https://en.wikipedia.org/wiki/Divisor_function
4. Бухштаб А. А. Теория чисел. / А. А. Бухштаб – М.: Просвещение, 1966 – 392 с.
5. Дирихле П. Г. Л. Лекции по теории чисел. В обработке и с добавлениями Р. Дедекинда. / П. Г. Л Дирихле. – М.: Объединённое научно-техническое издательство НКТП СССР, 1936. – 404 с.
6. Apostol, Tom M. (1976), *Introduction to analytic number theory*, Undergraduate Texts in Mathematics, New York-Heidelberg: Springer-Verlag.
7. Little J. D. C. An algorithm for the traveling salesman problem. / Little J. D. C., Murty K. G., Sweeney D. W., Karel C. // Operations Research – Vol. 11 (1963) – P. 972-989.
8. Гудман, С. Введение в разработку и анализ алгоритмов. / С. Гудман, С. Хидетниеми. – М.: Мир, 1981 – 368 с.
9. Порублёв И. Н., Алгоритмы и программы. Решение олимпиадных задач / И. Н. Порублёв, А. Б. Ставровский – М.,СПб-К., Диалектика, 2007. – 480 с.

10. Шень А., Программирование: теоремы и задачи / А. Шень. – М., МЦНМО, 2007 – 296 с.

References:

1. Web-site "The On-Line Encyclopedia of Integer Sequences". Retrieved from: https://oeis.org/A002182
2. The Article "Highly composite number". Retrieved from: https://ru.wikipedia.org/wiki/Сверхсоставное_число. [in Russian]
3. The Article "Divisor_function". Retrieved from: https://en.wikipedia.org/wiki/Divisor_function
4. Bukhshtab, A. A. (1966). Number theory. – M.: Prosveshcheniie. [in Russian]
5. Dirichlet, P. G. L. (1936). Lectures on number theory. In processing and with additions by R. Dedekind. – M.: NKTP of the USSR. [in Russian]
6. Apostol, Tom M. (1976). Introduction to analytic number theory // Undergraduate Texts in Mathematics, New York-Heidelberg: Springer-Verlag.
7. Little, J. D. C., Murty, K. G., Sweeney, D. W., Karel, C. (1963). An algorithm for the traveling salesman problem. Operations Research, Vol. 11.
8. Goodman, S., Hideniemi, S. (1981). Introduction to the development and analysis of algorithms. – M.: Mir. [in Russian]
9. Porublyov, I. N., Stavrovsky, A. B. (2007). Algorithms and programs. Solving Olympiad problems. – M.-St. Petersburg-K., Dialektika. [in Russian]
10. Shen', A. (2007). Programming: theorems and problems. – M., MTsNMO. [in Russian]

PORUBLYOV Ilya,
senior lecturer, Bohdan Khmelnytsky National University of Cherkasy, Ukraine

SOME ALGORITHMS FOR SEARCHING NUMBERS WITH MAXIMAL QUANTITY OF DIVISORS

Summary. Introduction. *Introduced some versions of the problem of searching, among numbers of some range, numbers with maximal quantity of divisors. Itroduced reasons, why searching such numbers among range between arbitrary M and N should be considered harder, that among range between 1 and N. Considered generalizations, when maximal quantity of divisors is searched not among all numbers in range, but among numbers, which aren't multiples of some given K, only. Some versions of algorithms, solving some versions of the problem, were designed, based on significant optimizations of brute force approach. A comparative analysis of performance of the algorithms was carried out, the differences and shortcomings for the each of the variants were estimated, and the recommendations about the efficiency of the algorithms were made.*

Purpose. *The purpose of the article is to describe the designed algorithms for solving problem of searching numbers with maximal quantity of divisors, in such problem formulation versions:*

1. Given a positive integer N, for all integers in range between 1 and N (both inclusive), find number having maximal (among numbers in the range) quantity of divisors.

2. Given positive integers M, N ($M \le N$), for all integers in range between M and N (both inclusive), find number having maximal (among numbers in the range) quantity of divisors.

3. Given positive integers N and K (K significantly less N, $K \ge 2$), for integers in range between 1 and N (both inclusive), but excluding multiples of K, find number having maximal (among the said numbers) quantity of divisors.

To achieve the goal, a number of definitions and statements are given that allow to implement more efficient algorithms compared to the standard ones. Based on the given theoretical information, algorithms are developed and described, which are further implemented in C++.

Results and conclusions. *For each problem formulation mentioned in previous section, an algorithm, significantly more effective than obvious one, was developed. Attempts to build adequate asymptotic estimate of their work duration failed (for each of them), but work time duration of each of them was studied experimentally. For searching the divisors on the interval from 1 to N, duration didn't exceed some decimals of second for $N \approx 10^{38}$; for searching the divisors on the interval from some integers M to N, duration didn't exceed some seconds for $N \approx 10^{18}$ (the article provides detailed tables with the results of the algorithms); for searching the divisors on the interval from some integers M to N, if it is not known whether the number M is prime, duration didn't exceed some seconds for*

$N \approx 10^{35}$. *Thus, the obtained results allow us to assert the greater efficiency of the developed algorithms compared to the standard ones.*

Keywords: *quantity of divisors, highly composite numbers, brute force and its optimizations.*

Одержано редакцією 24.01.2020 р.
Прийнято до публікації 30.03.2020 р.