УДК 621.3

М.В. Черкаський, В.С. Мітьков
Національний університет “Львівська політехніка”,
кафедра “Електронні обчислювальні машини”

ІСТОРИЧНИЙ АСПЕКТ СКЛАДНОСТІ АЛГОРИТМУ

© Черкаський М.В., Мітьков В.С., 2002

Розглядається історія розвитку понять "алгоритм" і "складність". Аналізується складність алгоритму Евкліда. Показано значення робіт аль-Хорезмі. Описані арифметичні алгоритми аль-Хорезмі. Висловлюється думка про надзвичайний вплив робіт аль-Хорезмі на європейський ренесанс.

The histores of development of such notion as "algorytm" and "compliting".

The compliting of Evclid’s algorythm is beeing analazed. The importance of al-Horezmi’s works are showed. The arithmetic algorytms of al-Horezmi are described. The idea that al-Horezmi’s works had influenced the european renesanse great is claimed.

Вступ

Інтуїтивне розуміння алгоритму як послідовності дій для отримання результату здавна відоме людству. І сьогодні ми користуємось такими давніми алгоритмами, як Китайська теорема про залишки або арифметичні операції, що прийшли до нас з Індії. Але теорія складності алгоритмів стала предметом активних досліджень тільки в другій половині минулого сторіччя. Перші ж кроки пошуку ефективних алгоритмів за часовою складністю були зроблені більш ніж три тисячі років тому Евклідом. Аль-Хорезмі, що працював у IX ст. н. е., описав десяткову систему числення – більш ефективну, ніж римська, що використовувалась на той час в Європі. Тому праці Евкліда і аль-Хорезмі для теорії складності алгоритмів мають особливе значення.

Фундаментальна праця Евкліда “Начала” (можна казати перші “Начала”, бо другі “Начала” – праця Ньютона) була настільною книгою математиків протягом трьох тисячоліть. І зараз аксіоми з “Начал” є необхідним елементом загальної культури людини. У “Началах” вперше описаний широковідомий алгоритм знаходження найбільшого спільного дільника двох чисел, який не втратив свого значення до наших днів.

Мухамед ібн Муса з Хорезму, або арабською мовою – аль-Хорезмі (походженням з середньоазіатського міста Хорезм), видатний багдадський вчений, у своїй книжці – трактаті “Про індійський рахунок” описав десяткову систему числення і арифметичні операції “множення і ділення, сумування, віднімання та інші”. Сьогодні збереглися лише переклади трактату, перші з яких датуються початком XII ст. Далі ми наводимо цитати з “Книги про індійський рахунок”, переклад якого був виконаний Ю. Копелевич за середньовічним латинським текстом, що зберігається у Кембриджському університеті [1].

Кожний розділ, а іноді навіть абзац трактату починався словами “Сказав Альгорізмі…”. Це словосполучення використовували у своїх лекціях і професори середньовічних університетів. Поступово ім’я аль-Хорезмі набуло звучання “алгоризм”,

Lviv Polytechnic National University Institutional Repository http://ena.lp.edu.ua

“алгоритм” і навіть перетворилися у назву нової арифметики. Пізніше термін “алгоритм” почав означати регулярний арифметичний процес (Хр. Рудольф, 1525 р.). І тільки наприкінці XVII ст. в роботах Лейбніца цей термін набув змістовності, яка не заперечує сучасному тлумаченню: “Алгоритм – це будь-який регулярний обчислювальний процес, що дозволяє за кінцеву кількість кроків розв’язувати задачі визначеного класу”. Зауважимо, що за довгу еволюцію слова “алгоритм“ було втрачено джерело його виникнення. І тільки у 1849 році сходознавець Ж. Рейно повернув нам ім’я аль-Хорезмі [2].

Зауважимо, що й слово “алгебра” бере свій початок з математичного трактату аль-Хорезмі “Книга відновлення і протиставлення”. Мухамеду ібн Мусі ще не були відомі від’ємні числа, тому в процесі обчислень він користувався операцією перенесення від’ємника з однієї частини рівняння в іншу, де той стає доданком. Цю операцію Мухамед ібн Муса називав “відновленням”. Слову “протиставлення” відповідає зміст збирання невідомих на одну сторону рівняння. Арабською “відновлення” – аль-джебр. Звідси походить слово “алгебра”[3]. Слова “алгоритм” і “алгебра” на перших кроках розвитку математики були тісно пов’язані між собою і за змістом, і за походженням. Вони виникли з одного джерела, разом пройшли багатовіковий шлях еволюції і зайняли провідні місця у сучасній науковій термінології.

Здавна найбільшу увагу приділяли дослідженням алгоритму з метою мінімізації обсягу досліджень – часовій складності розв’язання задач. Але зміст складності алгоритму не обмежується однією характеристикою. У багатьох випадках не менше значення має складність логіки побудови алгоритму, різноманітність його операцій, зв’язаність їх між собою. Ця характеристика алгоритму називається програмною складністю. В теорії алгоритмів, крім часової та програмної складності, досліджуються також інші характеристики складності, наприклад, ємнісна, але найчастіше розглядають дві з них – часову і програмну. Якщо у кінцевому результаті часова складність визначає час розв’язання задачі, то програмна складність характеризує ступінь інтелектуальних зусиль, що потрібні для синтезу алгоритму. Вона впливає на витрати часу проектування алгоритму.

У статті зроблено спробу з’ясувати, у яких наукових працях вперше згадується про часову і програмну складності процедур розв’язання задач, або хто заклав перші камені у фундамент теорії складності алгоритмів.

Перший ефективний за часовою складністю алгоритм

Алгоритм знаходження найбільшого спільного дільника, яким ми користуємось для цієї цілі і донині, був запропонований Евклідом приблизно в 1150 році до н.е. у геометричній формі. В ньому порівняння величин проводилося відрізками прямих, без використання арифметичних операцій. [4]. Доведемо, що він є ефективною за часовою складністю обчислювальною процедурою розв’язання цієї задачі. Ми будемо користуватися числовою його формою, використовувати такі операції, які в часи Евкліда були не поширені, наприклад, операція ділення.

Для того, щоб довести ефективність алгоритму, потрібно порівняти його з таким, який приймається за неефективний. Прикладом такого неефективного алгоритму є процедура послідовного перебору можливих розв’язань задачі. Ефективність, як правило, визначається часовою складністю, що вимірюється кількістю операцій, необхідних для

Lviv Polytechnic National University Institutional Repository http://ena.lp.edu.ua

розв’язання задачі. Алгоритм Евкліда є ефективним за часовою складністю порівняно з алгоритмом перебору. Мінімізація часової складності дозволяє за всіх інших рівних умов збільшити продуктивність розв’язання задачі. Якщо розв’язання задач з різними початковими даними проводиться багаторазово, мінімізація часу дозволяє одержати економічний ефект.

Дослідимо розв’язання задачі знаходження найбільшого спільного дільника двох цілих чисел (N_1,N_2, $N_1 \geq N_2$) алгоритмом перебору і алгоритмом Евкліда. Алгоритм перебору заснований на операції інкременту змінної (n) від одиниці до меншого (N_2) з двох заданих чисел і перевірці, чи ця змінна є дільником заданих чисел. Якщо це так, то значення змінної запам’ятовується і операції алгоритму продовжуються. Якщо ні, то операції алгоритму продовжуються без запам’ятовування. Операції алгоритму закінчуються видачею з пам’яті знайденого останнім спільного дільника. Блок-схему алгоритму наведено на рис. 1, а.

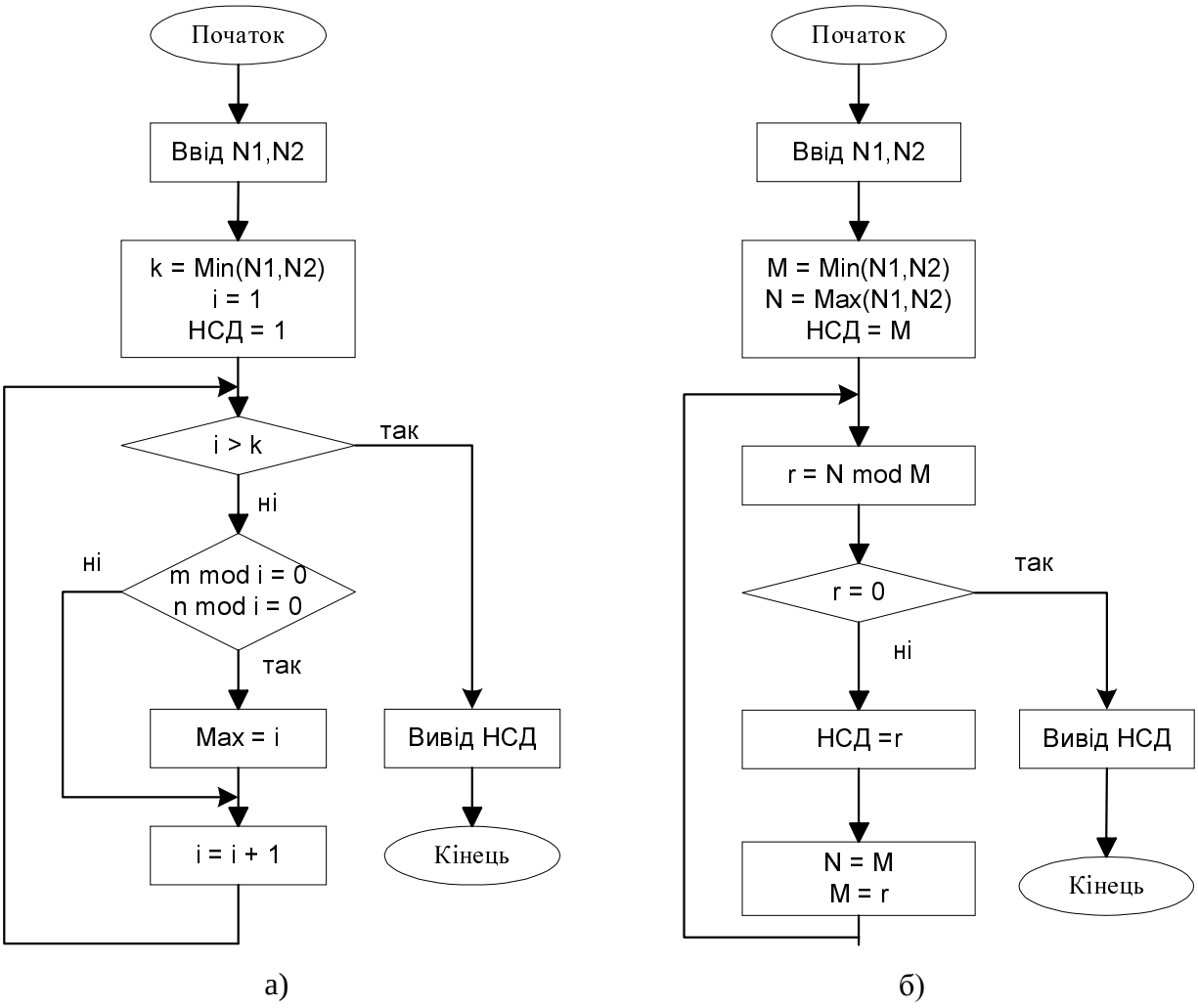

Рис. 1. Блок-схема алгоритму перебору (а) і Евкліда (б)

Адаптований до сучасної арифметики алгоритм Евкліда використовує циклічну операцію ділення більшого числа на менше, знаходження остачі (r) і заміну числа, яке було більшим, на число, яке було меншим, а менше число на остачу. Всі перераховані операції

Lviv Polytechnic National University Institutional Repository http://ena.lp.edu.ua

виконуються в циклі. Операції циклу закінчуються, коли ділення не дає остачу. Останній дільник є найбільшим спільним дільником. Блок-схему алгоритму наведено на рис. 1, б.

Аналіз цих двох алгоритмів показує, що часова складність алгоритму перебору значно перевищує часову складність алгоритму Евкліда. Для обох алгоритмів часова складність є функцією від вхідних даних, а не їх розміру. В таких випадках при порівнянні ефективності алгоритмів користуються порівнянням часових складностей, визначених для найгіршого випадку.[5] Часова складність для найгіршого випадку (L_{max}) являє собою максимальну часову складність серед всіх вхідних даних розміру N.

Очевидно, що часова складність L_{max} для алгоритму перебору

$$L_{max} = C \cdot N_2, \tag{1}$$

де C – *const*, яка дорівнює кількості операцій в кожній ітерації.

Доведемо, що для цілих чисел n ($1 \le n < r_i$) алгоритм Евкліда знаходження найбільшого спільного дільника має найбільшу часову складність для пари чисел r_{i-2} і r_{i-1} де 1, 2, 3, ... , r_{i-2} , r_{i-1} , r_i – числа Фібоначчі.

Проаналізуємо алгоритм Евкліда, починаючи з передостаннього циклу (в останньому циклі остача дорівнює 0, що є ознакою завершення алгоритму). Будемо рахувати цикли в зворотному порядку, тобто останній цикл рахуємо нульовим , передостанній – першим і т.д.

Найменша остача, яку можна отримати в першому (передостанньому) циклі дорівнює 1. Це означає, що найбільшою часова складність буде при знаходженні найбільшого спільного дільника двох взаємно простих чисел.

Найменша остача, яку можна отримати в другому циклі, дорівнює 2. Числа 2 і 1 є найменшими, які при діленні першого на друге дають в остачі 0.

Найменша остача, яку можна отримати в третьому циклі, дорівнює 3. Числа 3 і 2 є найменшими, які при діленні першого на друге дають в остачі 1.

Очевидно, що найменша остача r_i, яку можна отримати на і-му циклі, дорівнює сумі остач (і-1)-го та (і-2)-го циклів, а числа r_i та r_{i-1} є найменшими, які при діленні першого на друге дають в остачі r_{i-2}

$$r_i = r_{i-1} + r_{i-2} . \tag{2}$$

Той факт, що на кожній ітерації ми отримуємо мінімально можливу остачу, є доказом того, що кількість ітерацій буде максимальною.

Остачі 1, 2, 3, 5, … , r_{i-2} , r_{i-1} , r_i , які отримуються послідовно в кожній ітерації, є числами Фібоначчі.

При визначенні степеня зростання часової складності двох алгоритмів можемо знехтувати коефіцієнтом пропорційності C, який визначає кількість операцій на кожному кроці ітерації. Прийнявши C=1, отримаємо результати, наведені в таблиці.

Алгоритм перебору є найбільш простим для розуміння, запам’ятовування і програмування послідовності операцій розв’язання задачі. Його програмна складність є відносно малою. Але цей алгоритм має велику часову складність відносно алгоритму, в

Lviv Polytechnic National University Institutional Repository http://ena.lp.edu.ua

процесі синтезу якого було використано інтелектуальні зусилля, знання, досвід людини. В даному прикладі часова і програмна складності взаємозалежні.

Отже, вже три тисячі років тому замість бездумного способу розв'язання задач перебором були зроблені перші кроки пошуку ефективних алгоритмів за часовою складністю.

Степінь зростання часової складності

№ п/п	Числа Фібоначчі	Степінь зростання часової складності алгоритму Евкліда	Степінь зростання часової складності алгоритму перебору
1	121393	24	75025
2	75025	23	46368
3	46368	22	28657
4	28657	21	17711
5	17711	20	10946
6	10946	19	6765
7	6765	18	4181
8	4181	17	2584
9	2584	16	1597
10	1597	15	987
11	987	14	610
12	610	13	377
13	377	12	233
14	233	11	144
15	144	10	89
16	89	9	55
17	55	8	34
18	34	7	21
19	21	6	13
20	13	5	8
21	8	4	5
22	5	3	3
23	3	2	2
24	2	1	1
25	1	0	0

Lviv Polytechnic National University Institutional Repository http://ena.lp.edu.ua

Перші згадки про програмну складність

Вперше значення зменшення програмної складності продемонстрував аль-Хорезмі у своєму трактаті “Про індійський рахунок”. У часи аль-Хорезмі для розрахунків користувались непозиційною римською системою числення. Її вузловими числами є I, V, X, L, C, D, M, всі решта чисел утворюються сумуванням і відніманням вузлових. Аль-Хорезмі, мабуть, першим звернув увагу на складність римської системи числення порівняно з позиційною десятковою з точки зору простоти операцій, їх послідовного виконання та засвоєння. Він писав: “…ми вирішили розтлумачити про індійський рахунок за допомогою .IX. літер, якими вони виражали будь-яке своє число для *легкості* і *стислості*, *полегшуючи* справу тому, хто вивчає арифметику, тобто число найбільше і найменше, і все, що є в ньому від множення і ділення, сумування, віднімання та інше.”[1]. Виокремлені слова – *легкість, стислість, полегшення* свідчать перш за все про те, що мова йде про програмну складність алгоритмів арифметичних операцій з використанням двох систем числення. Мабуть, ці слова аль-Хорезмі про складність алгоритмів при їх порівнянні були першими в історії арифметики.

Алгоритм сумування

Алгоритми реалізації арифметичних операцій, описані аль-Хорезмі у словесній формі, були першими у позиційній десятковій системі числення. Цікаво спостерігати, як точно і послідовно описує він алгоритм сумування, користуючись арабською системою числення і кільцем (нулем). Наведемо повністю цей алгоритм [1].

“Сказав Алгорізмі: Якщо ти хочеш додати число до числа або відняти число від числа, постав обидва числа в два ряди, тобто одне над другим, і нехай буде розряд одиниць під розрядом одиниць і розряд десятків під розрядом десятків. Якщо захочеш скласти обидва числа, тобто додати одне до другого, то додай кожний розряд до розряду того ж роду, який над ним, тобто одиниці до одиниць, десятки до десятків. Якщо в якому-небудь із розрядів, тобто в розряді одиниць або десятків, або якому-небудь іншому набереться десять, став замість них одиницю і висувай її в верхній ряд, тобто, якщо ти маєш в першому розряді, який є розряд одиниць, десять, зроби з них одиницю і підніми її в розряд десятків, і там вона буде означати десять. Якщо від числа залишилось що-небудь, що нижче десяти, аби якщо саме число нижче десяти, залиши його в тому ж розряді. А якщо нічого не залишиться, постав кружок, щоби розряд не був пустим; але нехай буде в ньому кружок, який займе його, аби не сталося так, що якщо він буде пустим, розряди зменшаться і другий буде прийнятий за перший, і ти обманешся в своєму числі. Те ж саме ти зробиш у всіх розрядах. Подібним же чином, якщо збереться у другому розряді .X., зробиш з них одиницю і піднімеш її в третій розряд, і там вона буде означати сто, а що лишається нижче .X., залишиться тут. Якщо ж нічого в інших не залишається , ставиш тут кружок, як вище. Так ти зробиш в інших розрядах, якщо буде більше”. Можна бачити, що в цьому опису є всі параметри алгоритму. Це один з перших відомих у світі вербальних арифметичних алгоритмів.

Розглянемо логіку побудови арифметичних процедур з використанням римської та арабської систем числення з метою порівняння їх за програмною складністю. Розглянемо

Lviv Polytechnic National University Institutional Repository http://ena.lp.edu.ua

приклад операції сумування. Будемо користуватися алгоритмом на основі табличного методу. У арабській системі з порозрядними операціями розмір таблиці 10*10. Визначення суми чергових розрядів двох чисел, наприклад, 2 і 3 за таблицею дорівнює 5, або 7 і 9 дорівнює 16. Ці таблиці ми пам'ятаємо з дитинства.

Інша ситуація є з римською системою числення. Крім таблиці (I,II,…X)*(I,II,…X), що є еквівалентом таблиці 10*10 у арабській позиційній системі, додатково потрібно ще чотири таблиці з вузловими числами римської системи L,C,D,M та доповнення табличного методу логічними процедурами. Наприклад, для операції сумування двох чисел CMLIX + XCIV потрібні таблиці більшого об'єму, ніж 10*10 кожна. Один з варіантів рахунку полягає в представленні чисел, по-перше, розділеними на окремі цифри, по-друге, проведенням операцій віднімання і сумування окремих цифр з використанням таблиць, по-третє, об'єднанням цифр, що залишилися, в єдине число.

$$CMLIX + XCIV = (-C) + M + L + (-I) + X + (-X) + C + (-I) + V.$$

Оскільки

$$-C + C = 0;\ -X + X = 0;\ -I - I = -II,\ V - II = III,$$

то (1) перепишеться у вигляді

$$CMLIX + CIV = M + L + III = MLIII;$$

Як бачимо, для проведення розрахунків у римській системі необхідно виконувати більше типів операцій, ніж у десятковій арабській позиційній системі числення. Крім того, у римській системі потрібні додаткові логічні перетворення, що суттєво ускладнюють зв'язки між окремими операціями обчислювального процесу.

Часова складність операцій сумування і віднімання в десятковій системі визначається кількістю розрядів у взаємодіючих числах. У римській системі часова складність залежить від порядку розташування цифр у числах. У тих випадках, коли не потрібно утворювати ланцюги логічних операцій, часова складність не перевищує часову складність операцій з десятковими числами. У протилежному випадку, як у розглянутому прикладі, зростання невелике.

Отже, за логікою побудови і різноманітністю операцій – програмною складністю, арабська система суттєво простіша за римську, що й довів аль-Хорезмі. За часовою складністю вони майже однакові.

Десяткова система в Європі

Лише через три століття трактат "Про індійський рахунок" був перекладений в Європі латинською, іспанською, італійською. Були і інші шляхи розповсюдження індійської системи, але найпотужнішим поштовхом для її опанування Європою були праці аль-Хорезмі. Перехід від римської системи числення до позиційної десяткової був видатним кроком не тільки у математиці. Революційне значення мало використання десяткової системи числення в Європі у всіх сферах життя – побуті, навчанні, торгівлі, суднобудуванні, техніці, географії, астрономії та багато інших. І те, що цей процес припав на епоху Відродження, не є випадковим. На нашу думку, введення десяткової системи числення

Lviv Polytechnic National University Institutional Repository http://ena.lp.edu.ua

відіграло вирішальну роль прискорювача у поступових процесах Ренесансу. Значна роль в цьому належить видатному арабському вченому аль-Хорезмі.

1. Мухаммад ибн Муса ал-Хорезми. Математические трактаты. – Ташкент, 1983. 2. Юшкевич А.П. История математики в средние века. – М., 1961. 3. Выгодский М.Я. Справочник по элементарной математике. – М., 1955. 4. Кальман Э. История математики в древности. – М., 1961. 5. Албфред В. Ахо. Структуы данных и алгоритмы. – М., 2000.

Lviv Polytechnic National University Institutional Repository http://ena.lp.edu.ua