Волинський національний університет
імені Лесі Українки

Антонюк Б. П.

Основи алгоритмізації та програмування

курс лекцій.
Частина 1

методичні рекомендації для студентів
спеціальності 014 Середня освіта (Інформатика)

Луцьк
Вежа-Друк
2022

УДК 681.3:004.421:510.5

А 72

Рекомендовано до друку науково-методичною радою
Волинського національного університету імені Лесі Українки
(Протокол № від _ .....)

Рецензенти: ***Гуда О. В.,*** кандидат технічних наук, доцент кафедри фізики та вищої математики Луцького національного технічного університету;

Булатецька Л. В., кандидат фізико-математичних наук, доцент кафедри комп'ютерних наук та кібербезпеки Волинського національного університету імені Лесі Українки

Антонюк Б. П.
А 72 Основи алгоритмізації та програмування: курс лекцій. Частина 1 методичні рекомендації для студентів спеціальності 014 Середня освіта (Інформатика) Луцьк: Вежа-друк, 2022. 36 с.

У методичних рекомендаціях подано теоретичний матеріал першого змістового модулю вибіркового курсу «Основи алгоритмізації та програмування» для ґрунтовного високого рівня засвоєння предмету студентами. Робота розкриває базові поняття та принципи теорії алгоритмів.

Рекомендовано кафедрою загальної математики та методики навчання інформатики факультету інформаційних технологій і математики до опублікування та використання у навчальному процесі університету студентами 1 курсу другого (магістерського) рівня, спеціальності 014 Середня освіта (Інформатика).

УДК 681.3:004.421:510.5

ВСТУП

З розвитком науки інформатики та проникненню її в різні галузі людського життя слово "алгоритм" стало одним з найбільш вживаних понять у розмові для широкого кола фахівців. Завдяки переходу до інформаційного суспільства алгоритми стають одним з найважливіших чинників та здобутків цивілізації. Відомо, що математична теорія алгоритмів як наука склалася зовсім не в зв'язку з активним розвитком інформатики та обчислювальної техніки, а виникла ще в глибинах математичної логіки для вирішення її ж власних проблем. Вона, перш за все, вплинула на світогляд математиків та їх творчість. Проте є безсумнівним взаємовплив теоретичних областей, пов'язаних з обчислювальною технікою та теорією алгоритмів. Теорія алгоритмів вплинула також і на теоретичне програмування. Зокрема, велику роль в теоретичному програмуванні грають моделі обчислювальних машин, які, по суті, є обмеженнями тих представницьких обчислювальних моделей, які були створені раніше в теорії алгоритмів. Трактування програм, як об'єктів обчислення, оператори, використовувані для складання структурованих програм (послідовне виконання, розгалуження, повторення) прийшли в програмування з теорії алгоритмів. Зворотний вплив виразився, наприклад, в тому, що виникла потреба в створенні і розвитку теорії обчислювальної складності алгоритмів. Таким чином, теорія алгоритмів застосовується не тільки в інформатиці, а й в інших сферах знань. Знання узагальнених алгоритмів є дуже важливим. Їх застосування можливе для складання розкладів, вивчення впливу різних добрив на різні рослини на різних грунтах, планування експериментів у медицині та біології, розподілу ресурсів, черговості ініціювання подій, видачі завдань студентам тощо. А також і в суміжних областях математики, наприклад: в теорії груп та напівгруп, криптографії, комбінаториці, логіці, статистиці, теорії кодів тощо.

ІСТОРІЯ ВИНИКНЕННЯ ТА РОЗВИТКУ ПОНЯТТЯ «АЛГОРИТМ»

Поняття алгоритму є одним з фундаментальних понять обчислювальної математики та інформаційних технологій. Проте, виникло воно задовго до появи обчислювальних машин завдяки пошукам загальних методів розв’язку однотипних задач. Так першим відомим сьогодні алгоритмом прийнято вважати правило обчислення найбільшого спільного дільника двох натуральних чисел створене ще у III сторіччі до нашої ери грецьким математиком Евклідом. Не завжди легка задача, але суттєва в багатьох ситуаціях. Суть алгоритму у тому, щоб віднімати від більшого числа менше, підставляючи отриманий результат на місце більшого числа до того часу, поки числа не будуть однаковими. Отримане число і буде найбільшим спільним дільником. Цей набір дій і є алгоритмом, тому що виконуючи ці кроки людина отримає найбільший спільний дільник для певної пари чисел. Слово «алгоритм» пішло від імені перського математика Мухаммада ібн Мусса Аль-Хорезмі, що жив у IX сторіччі. Так от англо-норманський рукопис XIII століття згадує: «Алгоризм був вигаданий у Греції. Це частина арифметики. Вигаданий він був завдяки майстру, котрого звали Алгоризм». Вона представляє собою опис чотирьох основних правил арифметичних дій, практично ж тих, що використовуються і сьогодні. У своїх роботах Аль-Хорезмі намагався описувати правила таким чином, щоб вони були зрозумілі усім грамотним людям. В той час, коли не існувало математичної символіки такої, як дужки, знаки операцій, буквенні позначення тощо – зробити це було складно. Але йому все ж вдалось розробити власний стиль чіткого словесного написання, який не залишав шансів читачеві щось не зрозуміти чи пропустити якісь дії.

За іншим джерелом, Аль-Хорезмі описав позиційну десяткову систему запису та дій над числами та знак «порожнє коло» - тобто нуль, для відображення відсутності позиції в записі числа. І те, і інше було винайдено в Індії декількома століттями раніше. Також методи, описані у його останньому трактаті, що використовували прописані фігури або лічильні камінчики, стали відомими пізніше як «algorism» або «augrym». Наразі, ці фігури всім відомі як цифри.

В першій половині XII століття книга Аль-Хорезмі в латинському перекладі потрапила до Європи - «Algoritmi de numero Indorum» або «Індійське мистецтво обрахунку, твір Аль-Хорезмі». Поступово люди

почали забувати, що Алгоризмі це взагалі-то ім'я автора правил, тож почали просто називати їх алгорізмами. Таким чином, ім'я Аль-Хорезмі перейшло у Алгорізмі. Сам термін вживався для позначення чотирьох арифметичних операцій. Таким означення, свого часу, і ввійшло в деякі європейські мови.

Між іншим, обчислення за допомогою запису ваги розряду та роботи Аль-Хорезмі були вже добре знайомі завдяки деяким європейським вченим. Гінді-арабська система числення була популяризована в Європі середньовічним італійським математиком та торговцем Леонардо з Пізи, відомим як Фібоначчі. Дякуючи його книзі "Liber Abaci", виданій у 1202 році, прописні цифри почали замінювати таблиці підрахунку та рахунок на пальцях, як переважний спосіб обрахунку у Європі XII століття. Причиною було навіть не те, що їх простіше запам'ятати, а те, що вони дали можливість вести журнали аудиту. Загалом ж, цифри набули широкого розповсюдження у західній Європі з появою рухомого типу – технології друку, що використовує рухомі компоненти для відтворення елементів документа на папері. Але справжня популярність з'явилась завдяки появі дешевого паперу у XIX столітті. Врешті-решт, є підстави вважати, що слово algorism перетворилось вже у сучасне algorithm завдяки народній етимології - з грецького arithmos. До речі, деякі середньовічні джерела зазначають, що грецький префікс «algo» означає «мистецтво» або «вступ». Інші ж зазначають, що алгоритми були винайдені чи то грецьким філософом, чи то королем Індії або Іспанії, якого звали чи то Algus , чи то Algor, чи то Argus. Деякі джерела, включаючи Данте Аліг'єрі, навіть ідентифікували винахідника як грецького міфологічного кораблебудівника-аргонавта. Насправді не зрозуміло, чи хоч якась з цих версій була історично точною. Таким чином, до останнього часу слово алгоритм посилалося виключно на механічні прийоми арифметики, що використовували арабське числення. З плином часу, значення слова розширювалось. Вчені застосовували його не тільки для обчислень, але і для інших математичних процедур. Близько 1360 року французький філософ Ніколай Орем написав трактат «Algorismus proportionum» - «Обчислення пропорцій», в якому було вперше використано степені з дробовими показниками. Вчений майже в лоб підійшов до ідеї логарифмів. Вже у 1684 році Г.В. Лейбніц у творі «Nova Methodvs pro maximis et minimis, itemque tangentibus…» вперше використав повноформатне слово «алгоритм» в ще більш широкому сенсі: як систематичний спосіб вирішення проблем диференціального

обчислення. Користувався словом також і Л.Ейлер. Одна з його робіт має назву «Використання нового алгоритму для вирішення проблеми Пелля». Тобто стає помітним те, що розуміння Ейлером алгоритму як синоніму до способу рішення задачі стає близьким до сучасного розуміння.

Багато алгоритмів було створено та записано на папір. Але перший алгоритм, що був написаний для виконання на машині, належав Аді Лавлейс – 1843 рік. Свого часу вона познайомилась з Чарльзом Беббеджом, котрого часто називають «батьком комп'ютерів» за його великий вклад у винахідництво останніх. Вона зацікавилась однією з його ідей – аналітичним двигуном. Це був механічний комп'ютер – машина, що автоматизувала обрахунки, той самий аналітичний двигун, для якого вона написала алгоритм. Робота Лавлейс полягала у написанні формули, яка показувала необхідні налаштування двигуна для обчислення певної складності послідовності чисел, званих числами Бернуллі. Наразі ця формула визнана першим комп'ютерним алгоритмом в історії. А. Лавлейс не обмежилась лише математичними розрахунками. У свій час вона була справжнім провидцем. Поки її сучасники бачили перші механічні комп'ютери лише як розбивачів чисел, вона запитувала: «А що ж стоїть за обрахунками?». Їй цікаво було розширити потенціал механічних комп'ютерів як інструментів для групових задач. Вона все ж сподівалася побачити комп'ютери, які могли б надати людям набагато більше, ніж просто автоматизовані розрахунки. На жаль, побудова аналітичного двигуна не була завершена до смерті Лавлейс, і тому вона ніколи не змогла побачити свій алгоритм у дії. Але він не побудований і на сьогоднішній день. Існують й інші часткові реалізації роботи Чарльза Беббеджа, але, на жаль, ми не можемо запустити алгоритм Ади Байрон на реальному аналітичному механізмі.

XIX століття стало епохою "алгоритмів, вбудованих у машини". Їх було багато, починала процвітати автоматизація людської діяльності. Якщо потрібен був незвичайний візерунок на шматку тканини, Жозеф Марі Жаккар, французький ткач і купець, мав рішення: жакардовий ткацький верстат. Це дозволило виробникам тканин виготовляти вишукані візерунки, використовуючи ряд перфорованих карток, які вказували ткацькому верстату, як саме потрібно ткати. Подібним чином ранні телефонні станції використовували складні механічні пристрої для підключення телефонних дзвінків. Вони автоматично виконували покрокові інструкції, щоб зрештою змусити двох людей

поговорити між собою. Ці машини, будь то ткацькі верстати чи станції, у свій час були новаторськими і досі вражають донині.

Однак усі ці пристрої все ще були суто механічними. Їх виготовляли з важелів, перемикачів, валів. Але вони все ж були дуже далекі від того, що ми сьогодні називаємо комп'ютерами.

Лише в 30-х роках ми побачили перші згадки про алгоритми в електронних (немеханічних) комп'ютерах. Алан Тьюрінг, англійський математик, був одним з перших вчених, які формально фіксували, як машини могли обчислювати. Метою Тьюрінга було зафіксувати загальний процес, а не той, який є специфічним для певного завдання, наприклад, для ідентифікації простих чисел або обчислення найбільшого спільного дільника. Науковець у 1936 році описав схему гіпотетичної машини та назвав алгоритмом те, що вміє робити така машину. Якщо щось не могло бути зроблено машиною Тюрінга, то це вже не був алгоритм. Таким чином, Тьюрінг формалізував правила виконання дій при допомозі опису роботи кількох конструкцій.

Розвиток так званої машини Тюрінга призвів до появи комп'ютерів загального призначення. На відміну від попередніх машин, нові комп'ютери мали б виконувати довільні набори інструкцій. Тобто, їх можна було б використовувати в цілях, навіть не передбачених їх творцями. Якщо іншими словами, то робота Тьюрінга призвела до розвитку комп'ютерів, на яких ми можемо встановлювати та запускати додатки.

Через десятиліття, алгоритми почали втрачати свою простоту. Настільки втрачати, що інколи люди не можуть пояснити, як вони працюють. Що ж стосовно людей, далеких від науки, то до початку 40-х років XX століття вони могли почути слово «алгоритм» лише під час навчання у школі, у купі з «алгоритмом Евкліда». Сам термін сприймався як «спеціальний» - це підтверджувала відсутність відповідних статей у невеликих енциклопедичних виданнях. Вже з середини XX століття стали розроблятися різні способи опису алгоритмів, наприклад, за допомогою спеціальних мов, які називаються алгоритмічними, і графових схем - графічного зображення алгоритму. Розвиток електронної обчислювальної техніки і методів програмування сприяло тому, що розробка алгоритмів стала необхідним етапом автоматизації. Багато хто в той час хотів вважати, що комп'ютерні алгоритми – це чорні ящики. Не потрібно було розуміти, як саме вони працювали, адже все що турбувало – це входи та виходи, що потрапило до чорного ящика, та що отримали врешті-решт. Це спрощення роботи алгоритму було вибором, котрого ми не

маємо наразі, адже люди не можуть пояснити, як саме алгоритми досягають певних результатів, то ж змушені думати про них, як про чорні, магічні ящики.

Яскравим підтвердженням цих слів є група алгоритмів штучного інтелекту. Ми можемо пояснити їх принципи: можна сказати, що алгоритм використовує штучну нейронну мережу. Ми можемо пояснити, як була створена мережа чи як вхідні дані результують у вихідні. Однак, не можемо пояснити, чому саме цей результат був результатом роботи алгоритму, за винятком чисто механічного пояснення. Ерік Луміс, тривалість ув'язнення якого залежала від алгоритму, намагався зрозуміти, чому алгоритм «COMPAS» оцінював його як злочинця високого рівня ризику, але це було просто неможливо. Складність алгоритмів часто надзвичайна. І це лише початок. Адже ми живемо у світі, де алгоритми є скрізь – не лише на папері чи в наших думках, але й у керуванні машинами, комп'ютерами та роботами.

ПОНЯТТЯ АЛГОРИТМУ

Найважливішими поняттями, на яких базується застосування ЕОМ для розв'язування різноманітних задач, є поняття алгоритм та програма. Термін "алгоритм", як вже було сказано, звичайно використовується для позначення деякої послідовності дій, що приводять до досягнення потрібного результату.

Тривалий час поняття алгоритму використовували лише математики при позначенні правил розв'язування різних задач.

Алгоритм – це набір інструкцій, що описує, як деяке завдання може бути виконане.

Іншими словами, алгоритм – система формальних правил, що визначає зміст і порядок дій над вхідними даними і проміжними результатами, необхідними для отримання кінцевого результату при розв'язуванні задачі.

Говорячи про алгоритми, необхідно розглянути джерела їх виникнення.

Перше джерело – це практика, наше повсякденне життя, що надає можливість, а іноді й вимагає отримувати алгоритми шляхом описання дій з розв'язування різних задач. Такі алгоритми називаються емпіричними.

Друге джерело – це наука. З її теоретичних положень і встановлених фактів можуть бути виведені алгоритми. Так, на основі теоретичних законів можна побудувати алгоритми для управління різними технологічними процесами.

Третім джерелом є різні комбінації і модифікації вже наявних алгоритмів. Очевидно, що тут велику роль відіграють кваліфікація та винахідливість працівників, їх знання про математичні закономірності перетворень алгоритмів.

Розв'язання будь-якої задачі на ЕОМ складається з кількох етапів, а саме:

– постановка завдання;
– формалізація (математична постановка задачі);
– вибір (або розроблення) методу розв'язування;
– розроблення алгоритму;
– складання програми;
– налагодження програми;
– обчислення та обробка результатів.

Поряд з цими етапами користувач у процесі розв'язування задачі може виконувати також наступні:

– вибір мови програмування;
– опис структури даних;
– оптимізація програми;
– тестування;
– документування та ін.

Під час постановки задачі першочергову увагу треба приділити з'ясуванню кінцевої мети і розроблення загального підходу до досліджуваної проблеми, а саме встановити:

1) чи зрозуміла термінологія у формулюванні задачі;
2) що дано;
3) що необхідно знайти;
4) які загальні властивості явища чи об'єкта;
5) чи існує розв'язок поставленої задачі і чи він єдиний;
6) яких даних не вистачає і чи всі вони потрібні;
7) які слід зробити припущення;
8) які можливості конкретної ЕОМ і заданої системи програмування (проаналізувати).

Формалізація – побудова математичної моделі розглядуваного явища. У результаті аналізу суті задачі визначається об'єм і специфіка даних, вводиться система умовних позначень, встановлюється

приналежність розв'язуваної задачі до одного з відомих класів задач, вибирається відповідний математичний апарат.

На перший погляд більшість задач, які зустрічаються на практиці, не мають чіткого і однозначного опису. Деякі задачі взагалі неможливо сформулювати в термінах, що допускають комп'ютерне розв'язання.

Зазвичай для формального опису задачі необхідна велика кількість різних параметрів, і часто лише в ході додаткових експериментів можна знайти інтервали зміни цих параметрів.

Якщо певні аспекти розв'язуваної задачі можна виразити в термінах якої-небудь формальної моделі, то це, безумовно, необхідно зробити, оскільки в цьому випадку в рамках формальної моделі можна дізнатись, чи існують методи й алгоритми розв'язання поставленої задачі. Навіть якщо вони не існують, то використання засобів і властивостей формальної моделі допоможе в побудові розв'язку задачі.

Практично будь-яку галузь математики чи інших наук можна використати для побудови моделі певного кола задач. Для задач, числових за своєю природою, можна побудувати моделі на основі загальних математичних конструкцій, таких як системи лінійних рівнянь, диференціальні рівняння тощо. Для задач з символьними або текстовими даними можна застосувати моделі символьних послідовностей або формальних граматик. Розв'язок таких задач містить етапи компіляції та інформаційного пошуку.

Після визначення математичного формулювання задачі слід вибрати метод її розв'язання. При цьому потрібно враховувати:

1) складність формул і співвідношень;
2) необхідну точність обчислень і характеристики самого методу.

Похибка результату визначається вибраним чисельним методом розв'язання задачі. Коли побудована (підібрана) модель поставленої задачі, то, звичайно, слід шукати розв'язок в термінах цієї моделі.

На етапі розробки алгоритму основна мета полягає в побудові розв'язання у формі алгоритму, що складається зі скінченної послідовності інструкцій, кожна з яких має чіткий зміст і може бути виконана з певними обчислювальними затратами за скінченний час. Тобто програма, написана на основі розробленого алгоритму, при будь-яких початкових даних ніколи не повинна приводити до нескінченних циклічних обчислень. З цією метою здійснюється:

1) поділ обчислювального процесу на можливі складові частини;
2) встановлення порядку їх слідування;
3) опис змісту кожної такої частини;

4) перевірка реалізації вибраного методу.

Розгляд крупноблочної структури алгоритму, дає змогу швидше і простіше розробити кілька різних його варіантів, провести їх аналіз, оцінку і вибрати оптимальний.

Поетапна деталізація алгоритму дає можливість здійснювати його розробку по частинах одночасно кількома спеціалістами.

Складання програми передбачає подання алгоритму у формі, зрозумілій ЕОМ. При налагодженні програми розробник перевіряє її візуально та виявляє помилки у процесі компіляції.

Обчислення та обробка результатів дозволяє отримати розв'язок задачі шляхом виконання завершеної програми. Цей етап є підсумком виконання всіх попередніх етапів, а іноді обумовлює необхідність повного перегляду і зміни підходу до розв'язання задачі.

Документування дає можливість людям зрозуміти програми, які написані іншими людьми. Існує так зване "золоте правило": "Оформляйте ваші програми в такому вигляді, в якому вам хотілось би бачити програми, написані іншими".

При розв'язуванні будь-якої задачі та побудові алгоритму її розв'язку звичайно беруть до уваги наявність деяких вхідних даних і мають уявлення про результат, що необхідно отримати.

ВЛАСТИВОСТІ АЛГОРИТМУ

Будь-який алгоритм повинен мати такі основні властивості:

– **детермінованість** (визначеність) – через повну однозначність правил, встановлених в алгоритмі, застосування алгоритму до однакових вхідних даних повинно приводити до однакового результату;

– **дискретність** – процес, що визначається алгоритмом, можна розчленувати (розділити) на окремі елементарні етапи (кроки), кожен з яких називається кроком алгоритмічного процесу чи алгоритму;

– **масовість** – алгоритм повинен бути придатним для розв'язування всіх задач певного типу. Наприклад, алгоритм для розв'язування системи лінійних рівнянь повинен бути придатним для системи, що складається з довільної кількості рівнянь, причому для нього існує множина даних, що допускаються в якості вхідних, тобто початкова система величин може вибиратись із деякої потенціально нескінченної множини;

– **результативність** – вказує на наявність таких варіантів вхідних даних, для яких обчислювальний процес, що реалізується за наданим алгоритмом, повинен через скінчену кількість етапів (кроків) зупинитись і дати шуканий результат або сигнал про те, що наданий алгоритм непридатний для розв'язання поставленої задачі.

Як встановлено в теорії алгоритмів, існують і такі класи задач, для розв'язування яких нема і не може бути встановлено універсального прийому – алгоритму розв'язування (хоча при окремих обмеженнях на ці розв'язування алгоритм може бути знайдено). Такі задачі називають алгоритмічно нерозв'язуваними.

Розробка алгоритму більш чи менш складної задачі вимагає високої кваліфікації виконавця і розуміння змісту задачі. З реалізацією алгоритму безпосередньо пов'язане вміння застосувати цей алгоритм до конкретних вхідних даних розв'язуваної задачі. Таке застосування називається алгоритмічним процесом. Цей процес полягає у перетворенні вхідних даних за правилами, визначеними заданим алгоритмом.

Алгоритмічний процес загалом складається із самостійних етапів, кожен з яких призначений для переведення даних з одного стану в інший. Одним із завдань кожного етапу обчислень є також визначення свого наступника.

З поняттям алгоритмічного процесу тісно пов'язане і поняття обчислювального процесу.

Обчислювальний процес в ЕОМ детермінований перетворенням даних за допомогою заданих кінцевих систем правил.

Суть алгоритмізації обчислювального процесу полягає в наступному:

–виокремлення автономних етапів обчислювального процесу;

–формальний запис змісту кожного з них;

–визначення порядку виконання виділених автономних етапів обчислювального процесу;

–перевірка правильності вибраного алгоритму для реалізації заданого методу обчислень.

Результати алгоритмізації обчислювального процесу систематизують (формалізуються) у вигляді певної обчислювальної схеми, тобто деякої послідовності операцій і форм запису результатів цих операцій, яка задає алгоритм розв'язування даної задачі.

Основними характеристиками виконавця алгоритму, тобто живого чи не живого об'єкту, що буде виконувати покроково усі команди, можна назвати наступні пункти:

- Середовище – умови, у яких діятиме виконавець.
- Елементарні дії – найпростіші дії, які виконуватиме виконавець.
- Система команд – сукупність допустимих команд для виконавця.
- Допустимі команди – команди, які зрозумілі виконавцю і можуть бути ним виконані.
- Недопустимі команди – команди, які не можуть бути виконані виконавцем з тих чи інших причин.

ФОРМИ ПРЕДСТАВЛЕННЯ АЛГОРИТМІВ

Існує декілька форм подання алгоритмів. Коротко розглянемо їх.

Словесна форма – тобто така, що може бути описана усно, за допомогою зрозумілої людям мови (наприклад, кулінарний рецепт).

Ця форма не надто поширена у науковій літературі через багатослівність та відсутність наглядності. Наприклад, розглянемо простий алгоритм порівняння двох простих чисел A та B, та збереження найбільшого з них у третій змінній C:

1. Отримати числа A та B
2. Порівняти їх
3. Якщо A більше за B, то запишемо значення A в C
4. Якщо не більше, то запишемо значення B в C
5. Вивід змінної C

Можна помітити, що такий запис має ряд недоліків: занадто багато слів, відсутня строга формалізація, деякі дії є неоднозначними. Через ці причини словесна форма не набула широкого поширення у якості запису алгоритму.

Формульно-словесна форма – коли алгоритм подається у наукових чи навчальних цілях, з використанням різноманітних формул (фізика, математика, хімія тощо)

Графічна форма – подання алгоритму у вигляді блок-схеми (структурної схеми). Такий спосіб полегшує перехід від словесної форми до написання коду програми. Він є чи не найзручнішою формою запису алгоритму, тому зміг набути широкого поширення у науковій та навчальній літературі. Власне ж блок-схема алгоритму – це графічне зображення алгоритму у вигляді схеми, де блоки (кроки алгоритму з відповідним описом) поєднані стрілками. Графічне зображення алгоритму використовується перед програмуванням завдання через його наочність, тому що зорове сприйняття зазвичай

полегшує процес написання програми, її коригування при допущених помилках, осмислення процесу обробки інформації. Інколи можна побачити таке означення: «Ззовні алгоритм являє собою схему - набір прямокутників та інших символів, всередині яких записується, що обчислюється, що вводиться в машину і що видається на друк і інші засоби відображення інформації». Тут форма подання алгоритму вже змішується із самим алгоритмом. Для створення та редагування блок-схем можна користуватися графічним редактором – наприклад «Microsoft Office Visio».

Програмна форма – подання алгоритму вже на мові програмування для подальшої його обробки машиною.

Псевдокод – подання алгоритму змішаною формою – програмною та словесною.

Зазвичай у цьому способі використовуються скорочені назви команд, змінних, загальні математичні визначення тощо. Строгих синтаксичних правил тут немає, як і єдино точного визначення, що ж таке псевдокод, хоча існують певні службові слова, які виділяються як при написанні вручну, так і на комп'ютері (наприклад, якщо…то…інакше)

Інші – специфічні способи запису алгоритмів. Наприклад, запис нот на нотному стані.

Не може здивувати те, що алгоритми теж мають переваги та недоліки. Серед переваг можна виділити те, що добре розроблений алгоритм полегшує налагодження, щоб ми могли виявити логічну помилку в програмі; алгоритм діє як проект, план програми та допомагає під час розробки програми; алгоритм не залежить від мов програмування і може бути легко кодований за допомогою будь-якої мови високого рівня. Щодо недоліків, то список буде коротшим: розробка алгоритму для складних задач буде трудомісткою і складною для розуміння; розуміння складної логіки за допомогою алгоритмів може бути дуже складним.

БАЗОВІ СТРУКТУРИ АЛГОРИТМІВ

Алгоритми, в залежності від початкових умов та мети задачі, способів її вирішення та визначення дій виконавця мають певну класифікацію, що собою відображає особливості реалізації того чи іншого алгоритму. Існує три основних типи:

- Лінійний алгоритм
- Розгалужений алгоритм

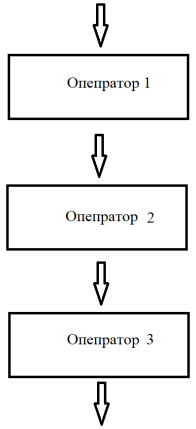
- Циклічний алгоритм

Загалом, алгоритми можна представити як певні структури, що, в свою чергу, складаються з інших елементів – базових алгоритмічних структур. Характерна особливість цих структур полягає в тому, що у них є лише один вхід та один вихід. Логічна структура будь-якого алгоритму складається з комбінації вищезгаданих базових алгоритмічних структур: розгалуження, слідування та циклу.

Лінійний алгоритм

Лінійний алгоритм – передбачає одержання розв'язку задачі одноразовим виконанням певної послідовності дій. Такі алгоритми застосовуються у найпростіших для процесу алгоритмізації задачах, в котрих перетворення інформації відбувається покроково за визначеними формулами.

Для представлення такого алгоритму використовується алгоритмічна конструкція послідовного виконання – «слідування», що передбачає собою послідовне виконання дій у визначеному порядку. У блок-схемі позначення такої конструкції здійснюється з допомогою блоків «процес».



Простим прикладом лінійного алгоритму є процес обчислення за формулою (-ами). Сам процес обчислення є досить складним, він потребує час та пам'ять для збереження як кінцевих, так і проміжних результатів. Порядок виконання обчислень у мовах програмування визначається за пріоритетом операцій, в тому числі і за дужками. Складність таких алгоритмів може викликати збої у роботі програми, або ж дуже довгу її екзикуцію. Тому для вирішення цих проблем доцільно розбити складний вираз на декілька простіших враховуючи лінійність запису виразу.

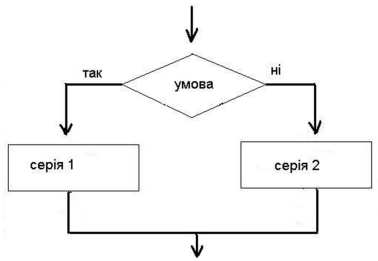
Розгалужений алгоритм

Розгалужений алгоритм – передбачає вибір однієї з кількох можливих послідовностей дій залежно від умови. Він використовується, коли перетворення інформації може відбуватись різними шляхами, залежно від властивостей вхідних даних та проміжних результатів. У розгалужених алгоритмах передбачаються всі можливі варіанти оброки інформації, кожен з яких розглядається як окрема гілка алгоритму, де вибір потрібної для виконання здійснюється за допомогою перевірки умови, що відображає усі властивості інформації, потрібної для процесу перетворення.

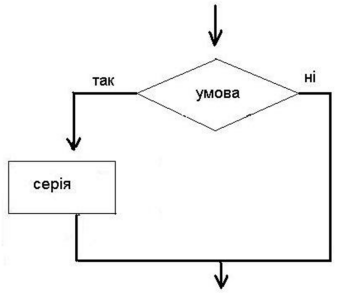
Для представлення розгалужених алгоритмів використовуються алгоритмічні конструкції розгалуження (вибору). Така структура обирає один з шляхів роботи алгоритму опираючись на результати перевірки необхідних умов. Алгоритм працюватиме незалежно від обраного шляху, адже усі вони ведуть до спільного виходу.

Структуру розгалуження можна поділити на чотири варіанти:

«якщо – то – інакше» – повне розгалуження, виконання дій незалежно від виконання заданої умови. Остання ж формулюється таким чином, щоб відповідь перевірки була «так» або «ні».



«якщо – то» – неповне розгалуження, виконання дій тільки у разі виконання або невиконання умови. Тобто є гілка, що не передбачає ніяких дій.

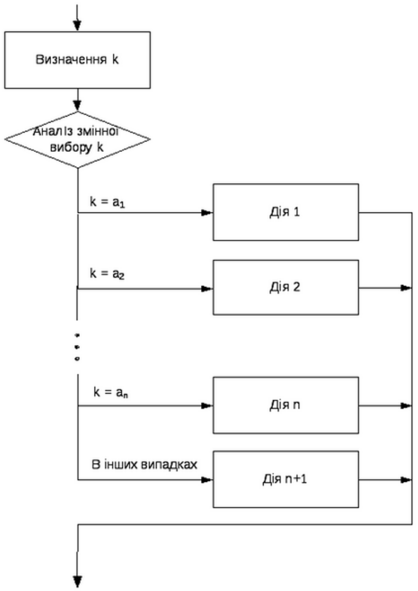


Для наступних двох варіантів необхідно ввести декілька нових означень:

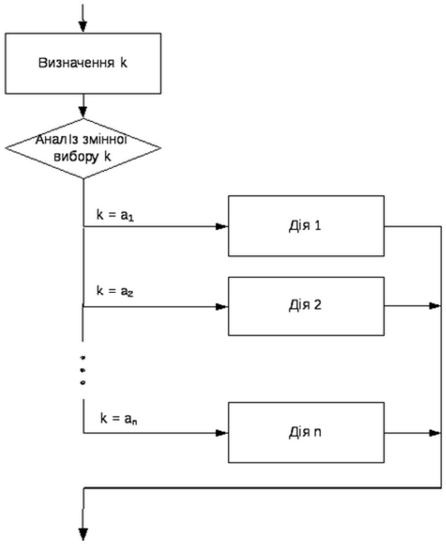
1) багатоваріантним розгалуженням називається процес, коли всі логічні блоки алгоритму поєднуються в один блок аналізу змінної, що відповідає за гілку. Такий блок матиме стільки виходів, скільки існує варіантів обробки;

2) якщо для такого розгалуження існує варіант обробки, коли значення змінної не належить до перелічених значень, то таке розгалуження називається повним багатоваріантним вибором, а інакше – неповним.

«вибір – інакше» – повний багатоваріантний вибір. При великій кількості умов для кожної гілки обирається змінна, що зберігає значення або проміжки значень даної гілки.



«вибір» – неповний багатоваріантний вибір



Циклічний алгоритм

Циклічний алгоритм – передбачає одержання розв’язку задачі багаторазовим виконанням певної послідовності дій. Повторювану

частину таких циклів називають тілом циклу. Для побудови циклічного алгоритму необхідно виконати наступні кроки:

- визначити всі дії, що потрібні для входу до циклу, підготувати його;
- визначити всі операції у циклі;
- визначити умову виходу із циклу.

Для представлення таких алгоритмів використовуються алгоритмічні конструкції повторення, які реалізуються одним із трьох способів:

Прості цикли з параметром. Якщо під час перетворення інформації є змінна, що змінюється за певним визначеним правилом, то можна визначити кількість ітерацій циклу та організувати вихід з циклу.

Ітераційні цикли. «Розв'язання систем лінійних алгебраїчних рівнянь з десятками чи сотнями невідомих, пошук коренів алгебраїчних рівнянь високих степенів, розв'язання систем диференційних рівнянь, інтерполяція та екстраполяція функцій, обчислення значень функції за допомогою рядів, інтегрування тощо – усі ці задачі розв'язуються за допомогою циклічних алгоритмів, що реалізують циклічні ітераційні процеси, для яких заздалегідь неможливо визначити кількість повторень циклу». Тому умова виходу з циклу формулюється залежно від самої задачі. Також існує два види ітераційних циклів – з передумовою та післяумовою. У циклі з передумовою, власне, умова знаходиться перед ітерацією, а з післяумовою – після ітерації циклу.

Складні циклічні процеси. З вищезгаданих структур можна будувати різні циклічні процеси із вкладеними циклами у них. Останні можуть бути внутрішні та зовнішні. Для зміни параметру у зовнішньому циклі відбувається багаторазове виконання визначених дій у внутрішньому (вкладеному) циклі. Кількість таких вкладених циклів не є обмеженою. Так, як особливістю базових алгоритмічних конструкцій є те, що вони мають лише один вхід та один вихід, то вони можуть вкладатися одна в одну довільним чином.

Так, комбінуючи різні алгоритмічні структури, можна отримати алгоритми будь-якої складності. За допомогою них можна створити навіть найскладніший алгоритм.

Так, комбінуючи різні алгоритмічні структури, можна отримати алгоритми будь-якої складності. За допомогою них можна створити навіть найскладніший алгоритм. Досвід практичної алгоритмізації привів до формування особливої методики структурної організації

алгоритмів, використання якої зменшує ймовірність помилок у процесі розробки і запису алгоритмів, спрощує їх розуміння і модифікацію.

Цю методику алгоритмізації називають структурним підходом. При структурному підході до конструювання алгоритмів їх ніби "збирають" із трьох основних (базових) структур.

ГРАФІЧНЕ ПРЕДСТАВЛЕННЯ АЛГОРИТМІВ

Опис алгоритмів у вигляді блок-схем є найбільш наочним і найпоширенішим графічним способом представлення алгоритмів. Схеми алгоритмів відображають шлях даних при розв'язку задач і визначають етапи обробки, а також різні носії даних, які застосовуються. Схема складається з наступних символів (будемо їх традиційно називати блоками):

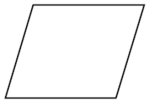
1) блоки даних, які можуть також вказувати вид носія даних;

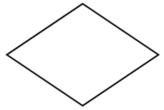
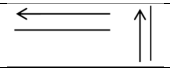
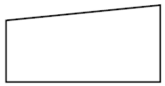
2) блоки процесу, який слід виконати над даними;

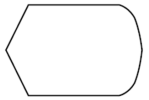
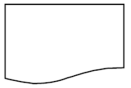
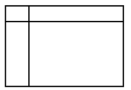
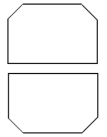
3) лінії, що вказують потоки даних між процесами й (або) носіями даних;

4) спеціальні символи, використовувані для полегшення написання й читання схеми.

У стандарті визначені умовні позначки в схемах алгоритмів і встановлені правила виконання схем. Наведемо графічні символи (блоки), які найчастіше використовуються при описі алгоритмів.

Найменування блоку	***Графічне зображення***	***Функція блоку***
1. Початок / Кінець		Блок відображає вихід у зовнішнє середовище й вхід із зовнішнього середовища (початок або кінець схеми)
2.Ввід / Вивід		Блок відображає дані, носій даних не визначений
3.Обчислювальний		Блок відображає функцію обробки даних будь-якого виду (виконання певної операції або групи операцій, що приводить до зміни значення, форми або розміщення інформації)

4. Логічний		Блок відображає розв'язок або функцію комутаційного типу, що має один вхід і ряд альтернативних виходів, один і тільки один з яких може бути активізований після обчислення умов, визначених усередині цього блоку. Відповідні результати обчислення можуть бути записані по сусідству з лініями, що відображають ці шляхи.
5.Підпрограма		Блок відображає визначений процес, що складається з однієї або декількох операцій або кроків програми, які визначені в іншому місці
6.Переадрасація		Блок відображає модифікацію команди або групи команд із метою впливу на деяку наступну функцію (установка перемикача, модифікація індексного регістру або ініціалізація програми)
7. Коментар		Блок використовують для додавання описових коментарів або пояснювальних записів з метою пояснення або приміток. Пунктирні лінії в блоці коментаря пов'язані з відповідним блоком або можуть обводити групу блоків. Текст коментарів або приміток має бути поміщений біля обмежуючої фігури.
8. Лінія		Блок відображає потік даних або керування
9. З'єднувач		Блок відображає вихід у частину схеми й вхід з іншої частини цієї схеми й використовується для обриву лінії й продовження її в іншому місці. Відповідні блоки-з'єднувачі повинні містити одне і те ж саме унікальне позначення.
10. Ручне введення		Блок відображає дані, що вводяться вручну під час обробки із пристроїв будь-якого типу. Будемо використовувати для позначення введення даних із клавіатури

11. Дисплей		Блок відображає дані, представлені в формі, яка читається людиною, на носії у вигляді пристрою, що відображає інформацію (екран для візуального спостереження)
12. Документ		Блок відображає дані, представлені на носії в зручній для читання формі (машинограма, документ для оптичного або магнітного зчитування, рулон стрічки з підсумковими даними). Будемо використовувати для позначення виводу даних на принтер
13. Оперативний запам'ятовувальний пристрій		Блок відображає дані, що зберігаються в оперативному запам'ятовувальному пристрої.
14. Границя циклу		Блок, що полягає із двох частин, відображає початок і кінець циклу. Обидві частини блоку мають той самий ідентифікатор. Умови для ініціалізації, збільшення, завершення і т.ін. містяться усередині блоку на початку або наприкінці залежно від розташування операції, що перевіряє умову. Будемо використовувати для позначення границь циклу

Правила застосування блоків

1. Блок призначений для графічної ідентифікації функції, яку він відображає, незалежно від тексту усередині цього блоку.

2. Блоки в схемі повинні бути розташовані рівномірно. Слід дотримуватися розумної довжини з'єднань і мінімального числа довгих ліній.

3. Кути й інші параметри, що впливають на відповідну форму символів не повинні змінюватися. Блоки повинні бути, по можливості, одного розміру.

4. Більшість блоків задумана так, щоб дати можливість включення тексту усередині блоку. Якщо обсяг тексту, що міститься усередині блоку, перевищує його розміри, слід використовувати блок коментаря.

5. У схемах може використовуватися ідентифікатор (номер, ім'я) блоків (блоку), який повинен розташовуватися ліворуч над блоком.

6. У схемах може використовуватися розгорнуте представлення, яке позначається за допомогою блоку зі смугою для процесу або даних. Блок зі смугою вказує, що в цьому ж комплекті документації в іншому місці є більш детальне представлення.

Блок зі смугою являє собою будь-який блок, усередині якого у верхній частині проведена горизонтальна лінія. Між цією лінією й верхньою лінією блоку розміщений ідентифікатор, що вказує на розгорнуте представлення даного блоку.

У якості першого й останнього блоку розгорнутого представлення має бути використаний блок покажчика кінця. Перший блок покажчика кінця повинен містити посилання, яке є також у блоці зі смугою.

Правила виконання з'єднань:

1. Потоки даних у схемах показуються лініями, які проводять горизонтально або вертикально. Напрямок потоку зліва направо і зверху вниз вважається стандартним. У випадках, коли необхідно внести більшу ясність у схему (наприклад, при з'єднаннях), на лініях використовуються стрілки. Якщо потік має напрямок, відмінний від стандартного, стрілки повинні вказувати цей напрямок.

2. У схемах слід уникати перетинання ліній. Лінії, що перетинаються, не мають логічного зв'язку між собою, тому зміни напрямку в точках перетину не допускаються.

3. Дві або більше вхідних ліній можуть поєднуватися в одну вихідну лінію. Якщо дві або більше ліній поєднуються в одну лінію, то місце об'єднання повинне бути зміщене.

4. Лінії в схемах повинні підходити до блоку або зліва, або зверху, а виходити або справа, або знизу. Лінії повинні бути спрямовані до центру блоку.

5. При необхідності лінії в схемах слід розривати для запобігання зайвих перетинань або занадто довгих ліній, а також, якщо схема складається з декількох сторінок. З'єднувач на початку розриву називається зовнішнім з'єднувачем, а з'єднувач наприкінці розриву – внутрішнім з'єднувачем. Посилання до сторінок можуть бути наведені разом із блоком коментаря для їхніх з'єднувачів.

6. Кілька виходів із блоку Рішення слід показувати двома лініями від даного блоку до інших блоків

Довжину блоку треба вибирати з ряду 10, 15, 20 мм і можна збільшувати на число, кратне 5. Висота блоку в півтора раза менше довжини, крім блоків 1, 10, у яких висота дорівнює половині довжини.

У цілому спосіб запису алгоритму у вигляді схеми можна розглядати як певну алгоритмічну мову зі своєю системою позначень (словник мови) і правил для однотипного запису алгоритмів і їх виконання (синтаксис мови). Оскільки в навчальному курсі «Інформатика» студенти вирішують відносно нескладні завдання, то доцільно в навчальних цілях розробляти алгоритми з високим ступенем деталізації, що полегшує складання програми для розв'язку задачі на комп'ютері. Із цією метою необхідно уточнити (деталізувати) і «тлумачний» словник і синтаксис графічної мови зображення алгоритму у вигляді схеми.

Блок Початок/Кінець уточнень не вимагає, будемо використовувати його для позначення початку й кінця процесу обробки даних. Блок Введення/Вивід будемо використовувати для позначення послідовного введення або виводу даних, носій яких не визначений. Для конкретизації функції усередині блоку можна записувати слово Введення або Вивід, нижче якого будемо записувати список введення або виводу відповідно. Список введення або виводу складається з елементів, що відділяються один від одного символом «кома». Елементом списку введення може бути:

- ім'я простої змінної;
- ім'я змінної з індексом (елемент масиву).

Елементом списку виводу може бути:

- ім'я простої змінної;
- ім'я змінної з індексом (елемент масиву);
- апостроф (послідовність символів, взята в лапки «»).

Під простою змінною, або просто змінною будемо розуміти деяку комірку пам'яті, тобто окреме місце для зберігання одного значення. В окремій комірці за час роботи алгоритму може побувати безліч різних констант (звідси назва – змінна). Такими комірками (електронними, магнітними, оптичними) оснащений комп'ютер. Змінні мають буквено-символьне позначення, наприклад, *S, ABA, F, n, a1, b, B2*. Водночас позначення змінної є адресою (номером, індексом) комірки, у якій будуть записуватися константи. Кожна з таких констант називається значенням змінної. Наприклад, *S* є змінною й адресою комірки *S* одночасно. З алгоритмічної точки зору поняття «змінна» і «адреса комірки» пам'яті є ідентичними.

Іншим різновидом змінних є так звана змінна з індексом або елемент масиву. Масив – це деяка сукупність комірок, об'єднана одним позначенням (іменем). Будь-який масив обов'язково має розмірність. Масиви бувають одномірними, двовимірними, тривимірними і т.ін. Для того щоб можна було відрізнити одну комірку масиву від іншої комірки цього ж масиву, їх нумерують. Нумерація комірок звичайно починається з 1 (необов'язково). Номер комірки масиву називається його індексом, а константа в комірці – елементом масиву. При введенні функцію блоку Введення/Вивід можна визначити наступними ключовими словами: прочитати дані з невизначеного носія інформації й записати (запам'ятати) у комірках запам'ятовувального пристрою.

ЕТАПИ РОЗРОБКИ АЛГОРИТМУ

Для вирішення будь-якої проблеми необхідно створити план – алгоритм дій. Є чимало шляхів, як це можна зробити. Деякі є занадто неформальними, інші навпаки – формальні та мають математичну природу, деякі ж можуть бути просто графічними. Розробка алгоритму – це ключовий крок у вирішенні задачі. То ж розробка алгоритму складається з п’яти основних кроків:

Опис проблеми

Досить часто опис проблеми є найслабшою частиною усього процесу, адже він може страждати від декількох типів помилок: неоднозначність опису, його неповнота, внутрішні суперечності. Завданням розробника алгоритму є виявлення дефекту в описі та правильна обробка помилок.

Аналіз проблеми

Метою цього кроку є визначення як початкової, так і кінцевої точок для вирішення проблеми. Цей процес аналогічний математику, який визначає, що дається і що має бути доведено. Хороший опис проблеми полегшує виконання цього кроку.

Визначаючи вихідну точку, слід почати з пошуку відповідей на такі запитання: які дані доступні, де ці дані, які формули стосуються проблеми, які взаємозв’язки між даними тощо. Визначаючи кінцеву точку, нам потрібно описати характеристики рішення. Іншими словами: «Як ми маємо дізнатись, коли закінчимо?»

Розробка алгоритму високого рівня

Алгоритм - це план вирішення проблеми, але плани мають кілька рівнів деталізації. Зазвичай краще починати з високорівневого

алгоритму, який включає основну частину рішення, але залишає деталі на наступний крок.

Деталізація алгоритму

Алгоритм високого рівня показує основні кроки, яких потрібно дотримуватися для вирішення проблеми. Тепер потрібно додати деталей до цих кроків, але скільки деталей потрібно додати? Відповідь на це питання залежить від ситуації. Необхідно розглянути, хто (або що) збирається впровадити алгоритм і скільки ця людина (або річ) вже знає, як це зробити. Коли мета - розробити алгоритми, які ведуть до комп’ютерних програм, потрібно врахувати можливості комп’ютера та надати достатньо деталей, щоб хтось інший міг використати наш алгоритм для написання комп’ютерної програми, яка відповідає крокам нашого алгоритму. Для складних проблем, як правило, кілька разів проходять цей процес, розробляючи алгоритми середнього рівня. Кожного разу потрібно додавати більше деталей до попереднього алгоритму, зупиняючись, коли не видно користі для подальшого вдосконалення. Цю техніку поступового переходу від високого рівня до детального алгоритму часто називають поетапним вдосконаленням - процесом розробки детального алгоритму шляхом поступового додавання деталей до алгоритму високого рівня

Перегляд алгоритму

Останній крок - перегляд алгоритму. Що шукається? По-перше, потрібно поетапно проробити алгоритм, щоб визначити, чи він вирішить вихідну проблему чи ні. Після того, як точно встановлено, що алгоритм надає рішення проблеми, потрібно почати шукати інші речі. Наступні запитання типові для тих, які слід задавати, коли ми переглядаємо алгоритм:

- Цей алгоритм вирішує дуже конкретну задачу чи вирішує більш загальну задачу?
- Якщо він вирішує цілком конкретну проблему, чи слід його узагальнювати?
- Чи можна спростити цей алгоритм?
- Чи подібне рішення до вирішення іншої проблеми?
- Чим вони схожі? Чим вони відрізняються?

Задавання цих питань та пошук їх відповідей - це хороший спосіб розвинути навички, які можна застосувати до будь-якої проблеми.

ПОНЯТТЯ АЛГОРИТМІЧНОЇ КУЛЬТУРИ

Для того, щоб розв'язувати задачі за допомогою ЕОМ, складати програми і користуватись ними, необхідна алгоритмічна культура.

Алгоритмічна культура – це частина загальної математичної культури і загальної культури мислення, яка зумовлює формування вмінь, пов'язаних з розумінням суті поняття алгоритму і його властивостей, тобто це сукупність знань, умінь і навичок, які дозволяють успішно розв'язувати задачі.

У процесі розв'язування прикладних задач вибір алгоритму має певні труднощі. Одна з основних проблем, що виникають при переході від алгоритмів безкомп'ютерних до алгоритмів, представлених у вигляді комп'ютерної програми, пов'язана з формою їх запису, яка повинна бути зрозумілою комп'ютеру. Алгоритм повинен відповідати наступним вимогам:

1) бути простим для розуміння, переведення в програмний код та налагодження;

2) ефективно використовувати комп'ютерні ресурси і виконуватись якнайшвидше.

Оволодіння алгоритмічною культурою включає необхідність не тільки інтуїтивного розуміння суті поняття алгоритму та його властивостей, уявлення про можливість автоматизації тієї сфери діяльності, до якої цей алгоритм застосовується, а й уміння представляти алгоритм за допомогою певних засобів і методів опису, зрозумілих виконавцю, знання основних типів алгоритмів і способів їх використання.

Із практичного досвіду побудови алгоритмів і реалізації їх у вигляді програм випливають наступні висновки та рекомендації:

1. Планування етапів розробки програми (спочатку чорновий варіант алгоритму у неформальному стилі, потім псевдопрограма, далі – послідовна формалізація псевдопрограми, тобто перехід до рівня виконуваного коду). Ця стратегія організовує і дисциплінує процес створення кінцевої програми, яка буде простою для налагодження і для подальшої підтримки та супроводу.

2. Використання інкапсуляції. Усі процедури, що реалізують абстрактні типи даних (АТД), подаються в одному місці програмного тексту. Надалі, якщо виникне необхідність змінити реалізацію АТД, можна буде коректно і без особливих затрат внести якісь зміни, оскільки всі необхідні процедури локалізовані в одному місці програми.

3. Використання та модифікація вже існуючих програм. Один із неефективних підходів до процесу програмування полягає в тому, що кожний новий проект розглядається з "нуля", без урахування вже існуючих програм. Як правило, серед програм, реалізованих на момент початку проекту, можна знайти такі, котрі якщо вирішують не всю поставлену задачу, то хоча б її частину. Після створення завершеної програми слід передбачити сфери, де її ще можна використати.

ОЦІНКИ СКЛАДНОСТІ АЛГОРИТМІВ

Одними з найбільш значущих властивостей алгоритмів є їх складність та ефективність. Тому при різноманітному застосуванні алгоритмів дуже важливо вміти правильно оцінювати їх. У програмуванні ж, обчислювальну складність алгоритмів оцінюють за допомогою кількості дій, що виконує алгоритм та за обсягом при цьому задіяної пам'яті. Перед написанням алгоритму потрібно визначити необхідні йому ресурси, оцінити те, як складність буде залежати від кількості вхідних даних, щоб алгоритм не працював нескінченну кількість часу. Для досягнення максимальної ефективності рекомендовано зменшити використання ресурсів. Однак час та пам'ять не можна порівняти, тому який із двох алгоритмів вважати більш ефективним залежить від того, який фактор для конкретної задачі важливіший – вимога високої швидкості, мінімального використання пам'яті чи інша міра ефективності.

Складність алгоритму – це «кількісна оцінка ресурсів, що витрачаються алгоритмом. В якості ресурсів можна розглядати людські (на створення і розуміння алгоритму) і обчислювальні (на виконання алгоритму) ресурси. Тому розрізняють описову (інтелектуальну) і обчислювальну складність алгоритму» [11].

Описова складність визначається з самого запису алгоритму та характеризується довжиною цього запису. Відомо, що єдиного критерію оцінки такої складності не існує.

Як обчислювальні ресурси для виконання алгоритму розглядають пам'ять і процесорний час. Тому основними мірами обчислювальної складності алгоритмів є:

- ємнісна (або просторова) складність – кількість пам'яті, необхідної для виконання алгоритму;
- часова складність – кількість часу, необхідного на виконання алгоритму (цей час, як правило, визначається кількістю

елементарних операцій, які потрібно виконати для реалізації алгоритму).

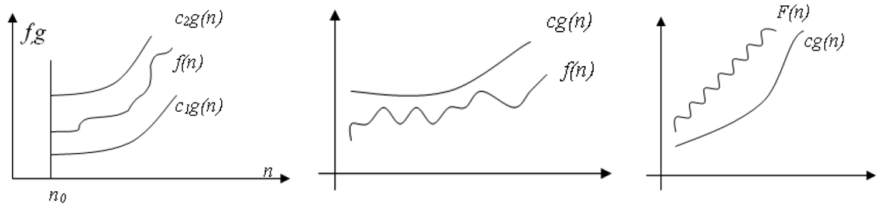
Комп'ютери мають обмежений обсяг пам'яті. «Якщо дві програми реалізують ідентичні функції, то та, яка використовує менший обсяг пам'яті, характеризується меншою ємнісною складністю. Іноді пам'ять є домінуючим чинником в оцінці ефективності програм. Однак в останні роки у зв'язку із швидким її здешевленням ця складова ефективності поступово втрачає своє значення».

Для оцінки часової складності можна просто запустити кожен алгоритм на декількох задачах і порівняти час виконання. Інакший спосіб – це оцінити час виконання підрахунком операцій математично. У цьому випадку складність алгоритму описується функцією $f(n)$, де n - розмір вхідних даних (це може бути розмірність масиву, кількість слів в тексті, довжина послідовності в алгоритмі, число вершин оброблюваного графа тощо).

Знаходження точної залежності $f(n)$ для алгоритму - задача доволі складна. Часто буває, що така докладна оцінка не потрібна. З цієї причини зазвичай обмежуються лише асимптотичними оцінками швидкості росту функції, які описують граничну поведінку складності алгоритму при збільшенні n (тобто те, наскільки швидко або повільно зростає ця функція).

Загалом використовується декілька підходів, які виражають порядок складності алгоритму. В їх основі лежить порівняння функції $f(n)$ з якою-небудь функцією, поведінка якої є дослідженою.

Отже, розрізняють верхні (O), нижні (Ω) і ефективні (Θ) асимптотичні оцінки.



Асимптотичні оцінки складності алгоритму ефективна, верхня та нижня відповідно

Найбільш популярною для оцінювання складності алгоритмів є верхня оцінка - O -нотація (читають «О велике»). Її позначення:

$$f(n)=O(g(n))$$

що фактично означає

$$\lim_{n\to\infty}\frac{f(n)}{g(n)} = const \neq 0.$$

При цьому кажуть, що f(n) є функцією порядку g(n) для великих n.

f(n)= Θ (g(n)), якщо ∃ c_1 >0, c_2 >0, n_0 >0, такі, що:

$$c_1*g(n) \le f(n) \le c_2*g(n), \text{ при } n > n_0$$

Очевидно, що алгоритм, який розв'язує задача розмірності n за $O(2^n)$ кроків, кращий за алгоритм, який розв'язує її за O(n!) чи $O(n^n)$ кроків.

Така нотація дозволяє враховувати у функції лише значущі елементи, відкидаючи ті, які менш важливі.

Розглянемо алгоритм обчислення значення багаточлена степеня n в заданій точці x

$$P_n(x)=a_n x^n+ a_{n-1} x^{n-1} \ldots+a_i x_i+\cdots+a_1 x_1+a_0$$

- Алгоритм 1. Для кожного доданка, окрім a_0, піднести x в задану степінь послідовним множенням і домножити на коефіцієнт. Потім доданки скласти.

Значить, всього $1+2+3+\cdots+n=n(n+1)/2$ множень. Додатково потрібно $n+1$ додавання. Всього $n(n+1)/2+n+1=n^2/2+3n/2+1$ операцій.

- Алгоритм 2. Винесемо x за дужки і перепишемо багаточлен у вигляді

$$P_n(x)=a_0+x(a_1+x(a_2+\cdots x(a_i+\cdots x(a_{n-1}+a_n x)))).$$

Наприклад

$$P_3(x)=a_3x_3+a_2x_2+a_1x_1+a_0=a_0+x(a_1+x(a_2+a_3x)).$$

Будемо обчислювати вираз зсередини. Сама внутрішня дужка вимагає одне множення і одне додавання. Її значення використовується для наступної дужки. І так, одне множення і одне додавання на кожну дужку, яких $n-1$ штука. І ще після обчислення самої зовнішньої дужки помножити на x і додати a_0. Всього n множень $+n$ додавань $=2n$ операцій.

У функції $f(n)=n^2/2+3n/2+1$ при досить великих n компонента n^2 буде значно перевершувати інші складові, і тому характерна поведінка цієї функції визначається саме цією компонентою. Інші компоненти (порівняно повільно зростаючий доданок 3n/2 +1 і константний

множник 1/2) можна відкинути і умовно записати, що дана функція має оцінку поведінки (в сенсі швидкості росту її значень) виду $O(n^2)$ (читається як "О велике від ен квадрат"). Фраза «складність алгоритму є $O(n^2)$» означає, що при великих n час роботи алгоритму (або загальна кількість операцій) не більше ніж cn^2, де c - якась додатна константа.

Таким чином, O() - "урізана" оцінка часу роботи алгоритму, яку часто набагато простіше отримати, ніж точну формулу для кількості операцій.

Для однієї й тієї ж задачі можуть існувати алгоритми різної складності. Наприклад, для попередньої задачі алгоритм 1 має складність $O(n^2)$, алгоритм 2 - $O(n)$.

Часто буває і так, що більш повільний алгоритм працює завжди, а більш швидкий - лише за певних умов.

Розглянемо правила формування оцінки O( ):

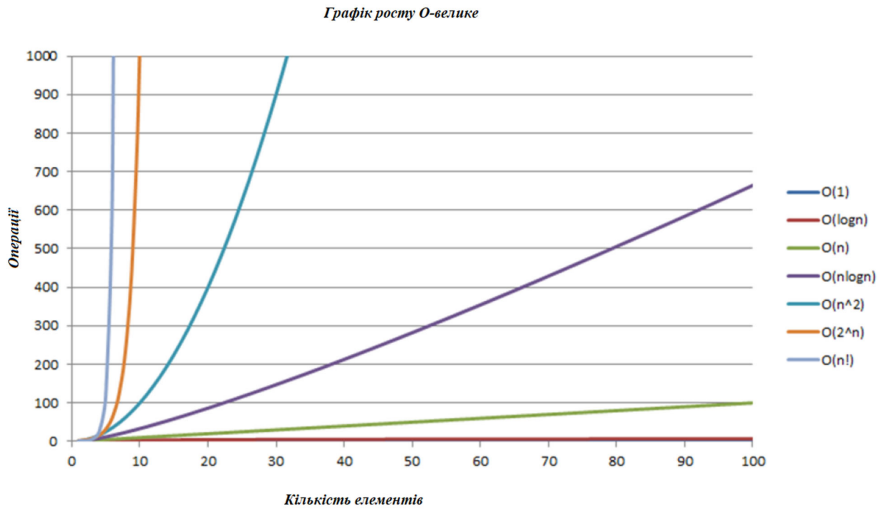
- При оцінці за функцію береться кількість операцій, що зростає найшвидше.

Тобто, якщо в програмі одна функція, наприклад, множення, виконується $O(n)$ разів, а додавання - $O(n^2)$ разів, то загальна складність програми - $O(n^2)$, оскільки при збільшенні n додавання стануть виконуватися настільки часто, що будуть впливати на швидкодію куди більше, ніж множення.

- При оцінці O( ) константи не враховуються.

Нехай один алгоритм робить $2500n+1000$ операцій, а інший - $2n+1$ Обидва вони мають оцінку $O(n)$, так як їхній час виконання зростає лінійно.

Інший наслідок опускання константи - алгоритм з часом $O(n^2)$ може працювати значно швидше алгоритму $O(n)$ при малих n. За рахунок того, що реальна кількість операцій першого алгоритму може бути $n^2+10n+6$, а другого – $1000000n+5$. Втім, другий алгоритм рано чи пізно обжене перший, оскільки n^2 росте куди швидше за $1000000n$.



Важливе теоретичне і практичне значення має класифікація алгоритмів, яка бере до увагу швидкість зростання цієї функції.

Отже, залежно від складності, виділяють такі основні класи алгоритмів:

O(1)	Сталий час роботи не залежно від розміру задачі. Наприклад, прочитати деякий елемент масиву.
$O(\log \log n)$	Дуже повільне зростання необхідного часу. Наприклад, коли задача розбивається на незалежні підзадачі пропорційно до квадратного кореня від розміру вхідних даних.
O(log n)	Логарифмічне зростання – подвоєння розміру задачі збільшує час роботи на сталу величину. Наприклад, коли задача розбивається на незалежні підзадачі діленням їх навпіл.
O(n)	Лінійне зростання – подвоєння розміру задачі подвоїть і необхідний час. Наприклад прочитати деякий елемент списку, або знайти необхідний елемент в масиві.
O(n log n)	Лінеаритмічне зростання – подвоєння розміру задачі збільшить необхідний час трохи більше ніж вдвічі. Наприклад, коли задача розбивається на підзадачі

поділом навпіл, а потім окремі результати збираються в єдиний.

$O(n^2)$	Квадратичне зростання – подвоєння розміру задачі вчетверо збільшує необхідний час. Наприклад для простих алгоритмів сортування, де зустрічається вкладений цикл по всіх вхідних даних.
$O(n^3)$	Кубічне зростання – подвоєння розміру задачі збільшує необхідний час у вісім разів. Наприклад, просте множення матриць, або алгоритми де зустрічається потрійне вкладення циклів по всіх елементах.
$O(cn)$	Експоненціальне зростання – збільшення розміру задачі на 1 призводить до c-кратного збільшення необхідного часу; подвоєння розміру задачі підносить необхідний час у квадрат. Наприклад, алгоритми повного перебору.

Література

1. Erickson J. Algorithms, paperback, 2019. 472p. URL: https://jeffe.cs.illinois.edu/teaching/algorithms/book/Algorithms-JeffE.pdf (Last accessed: 21.05.2022)
2. Kowalkiewicz M. How did we get here? The story of algorithms. 2019. URL: https://towardsdatascience.com/how-did-we-get-here-the-story-of-algorithms-9ee186ba2a07 (Last accessed: 21.05.2022)
3. Малярчук С. М. Основи інформатики у визначеннях, таблицях і схемах: Довідково-навчальний посібник. «Ранок», 2007. 112 с.
4. Базові алгоритмічні структури. Національний технічний університет України «Київський політехнічний інститут». Київ, 2016. URL: https://studfile.net/preview/5994725/page:4/ (Дата звернення: 20.05.2022)
5. Оцінка ефективності алгоритму. Національний технічний університет України «Київський політехнічний інститут». Київ, 2016. URL: https://studfile.net/preview/5994725/page:2/ (Дата звернення: 03.05.2022)
6. van Lint J. H., Wilson R. M. A Course in Combinatorics. — Cambridge University Press, 1992.
7. Андрійченко К. А., Вєтров О. С. Аналіз алгоритму «Перебір з поверненням» на прикладі задачі побудови латинського квадрату Матеріали всеукраїнської науково-практичної конференції для студентів, аспірантів та молодих вчених (29 квітня 2020 р.). Вінниця, 2020. С. 161-163.

Зміст

Навчально-методичне видання

Антонюк Богдан Петрович

Основи алгоритмізації та програмування

курс лекцій.
Частина 1

методичні рекомендації для студентів
спеціальності 014 Середня освіта (Інформатика)

Друкується в авторській редакції

Формат 60x84 $^{1}/_{16}$. Обсяг 1,5 ум. друк, арк., 1,4 обл.-вид. арк.
Наклад 50 пр. Зам. 63. Видавець і виготовлювач - Вежа-Друк
(м. Луцьк, вул, Винниченка, 14, тел. (0332) 29-90-65).
Свідоцтво Держ. комітету телебачення та радіомовлення України
ДК № 4607 від 30.08.2013 р