Лекція 0. Основні поняття обчислювальної техніки

План

1. Розвиток обчислювальної техніки (ОТ).
2. Архітектура комп’ютера. Принципи фон Неймана.
3. Поняття алгоритму.
4. Приклади алгоритмів.
5. Типи задач та алгоритмів, що будуть розглядатися.

1. Розвиток ОТ.

Традиційно еволюцію обчислювальної техніки представляють як послідовну зміну поколінь ОТ. Поява терміну «покоління» відносять до 1964 р., коли фірма IBM випустила серію комп’ютерів IBM 360 та назвала її «комп’ютерамі третього покоління».

Якщо виконувати класифікацію поколінь за технологіями, які використовувались при створенні комп’ютерів, традиційно, виділяють наступні шість поколінь.

Нульове покоління (1492-1945). Створення механічних та електромеханічних пристроїв, які були здатні виконувати арифметичні обчислення.

1492 р. Леонардо да Вінчі наводить рисунок пристрою на основі зубчатих колес, який здатен виконувати додавання десяткових чисел з тринадцяти розрядів.

1642 р. Блез Паскаль сконструював «Паскалін» - перший механічний обчислювальний пристрій який був здатен виконувати додавання та віднімання п’яти значних десяткових чисел. Було виготовлено більше десяти пристроїв.

1673 р. Готфрід Вільгельм Лейбніц створює пристрій, який здатен виконувати всі чотири арифметичні операції на 12-розрядними десятковими числами.

1801 р. Жозеф Марія Жаккард створює ткацький станок з програмним керуванням, програма якого задається за допомогою набору перфокарт.

1885 р. Дорр Фельт створює перший механічний калькулятор, в якому цифри вводяться натисканням клавіш.

1937 р. Джордж Стібітц створює перший однобітовий двійковий обчислювач на базі електромеханічного реле.

1938 р. Німецький інженер Конрад Цузе створює механічний програмований обчислювач Z1 з пам’яттю на 1000 бітів.

1943 р. Група дослідників з Гарвардського університету на чолі з Говардом Айкеном створює обчислювач ASCC Mark I – перший програмно

керований обчислювач. Довжина пристрою складала 18 м, а маса – 5 т. Машина складалась з великої кількості обчислювачів, які працювали під керуванням одного пристрою. Команди зчитувалися з паперової перфоленти та виконувались послідовно. Машина працювала з 23-розрядними числами. Додавання займало 0,3 с, множення – 4 с, а ділення – 10 с.

Перше покоління (1937 – 1953). Основною відмінністю обчислювальних пристроїв першого покоління було використання радіоламп замість механічних та електромеханічних реле. При цьому, програма здебільшого зберігалась на зовнішніх носіях – перфолентах.

Першою електронною обчислювальною машиною частіше всього називають спеціалізований калькулятор ABC (Atanasoff-Berry Computer). Він був розроблений в період з 1939 по 1942 Джоном Атанасовим та Кліффордом Беррі та призначався для розв'язання систем лінійних алгебраїчних рівнянь. ABC мав пам'ять на 50 слів довжиною 50 бітів на базі конденсаторів. Як зовнішня пам'ять використовувались перфокарти.

Другим претендентом на звання першої обчислювальної машини вважається обчислювач Colossus, який був побудований в 1943 р. в Англії Максом Ньюменом. Colossus було створено для розшифровки кодів німецької шифрувальної машини «Лоренц». До складу розробників входив також Алан Тюрінг.

Третій кандидат на роль першої електронної ОМ – програмований електронний калькулятор ENIAC (Electronic Numerical Integrator and Computer). Ідея калькулятор висунута в 1942 р. Джоном Мочлі і реалізована спільно з Преспером Еккертом в 1946 р. З самого початку калькулятор активно використовувався при створенні водородної бомби. Машина використовувалася до 1955 р. для генерування випадкових чисел, передбачення погоди і т.д.

Найбільш важлива подія цього періоду розвитку ОТ пов'язана з ім'ям Джона фон Неймана, який приймав участь у проекті ENIAC як консультант. Ще до завершення проекту Еккерт Мочлі та фон Нейман розпочали новий проект – EDVAC, особливістю якого стала ідея програми, що зберігається в пам'яті обчислювальної машини.

Технологія програмування цього періоду розвитку ОТ була примітивною – перші програми створювалися в машинних кодах – числах, які потрібно було ввести в пам'ять комп'ютера. Лише в 50-х роках почалось використання мови асемблера.

Друге покоління (1954 – 1962). Характеризується рядом досягнень в елементній базі комп'ютера та технології програмування. З технологічної

точки зору, перехід до другого покоління характеризується наступними нововведеннями:

- використанням напівпровідникових діодів та транзисторів замість радіоламп.
- використанням магнітних сердечників для пристроїв пам’яті.

Використання транзисторів дало змогу зменшити розміри обчислювальних машин та підвищити швидкість операцій.

Також змінювалась архітектура ЕОМ. В архітектуру процесора було додано індексні регістри, що дозволило спростити роботу з масивами даних. Додано блок роботи з числами з плаваючою точкою. Додано процесори вводу\виводу, що дозволило зняти це навантаження з центрального процесора.

Важливою подією стало створення мов програмування високого рівня: Фортран (1956), Алгол (1958), Кобол(1959).

Третє покоління (1963 – 1972). В перших ЕОМ третього покоління використовувались інтегральні схеми малої та середньої інтеграції, які дозволяли розмістити на одному кристалі від 10 до 100 транзисторів.

В архітектурі почали використовуватися технології паралельної обробки інформації.

У 1964 р. Сеймур Крей створив обчислювальну машину CDC 6600 яка мала швидкодію 1 MFLOPS (один мільйон операцій з плаваючою точкою в секунду).

Фірмою IBM розпочато серію ЕОМ IBM 360, які мали велику популярність.

В сфері програмного забезпечення необхідно відзначити створення в 1970 р. Кеном Томпсоном з Bell Labs мови програмування B, яка була предком популярної мови програмування C, та створення першої версії операційної системи UNIX.

Четверте покоління (1972 – 1984). Відносяться обчислювальні машини, побудовані на інтегральних схемах великої та надвеликої інтеграції з кількістю транзисторів на одному кристалі від 1000 до 100000. При такому рівні інтеграції стало можливим розмістити на одну мікросхему не тільки центральний процесор, але й основну пам’ять та систему вводу\виводу.

В цей період набувають розвитку мікропроцесори та мікроЕОМ. Завдяки цьому, з’являються перші персональні комп’ютери, які могли використовувати невеликі компанії та окремі користувачі.

1973 р. – випущено декілька тисяч Xerox Alto – перший персональний комп’ютер з графічним інтерфейсом.

1975 р. – випуск комп’ютера Altair 8800 з процесором i8080.

1976 р. – комп’ютер Apple I.

1977 р. – перші масові персональні комп’ютери Apple II.

1981 р. – фірма IBM випустила комп’ютер IBM PC (IBM 5150) з відкритою архітектурою, що дозволило стороннім фірмам виконувати ремонт, обслуговування та виробляти периферійні пристрої. До 1988 було вироблено 25 мільйонів IBM-сумісних ПК.

1983 р. – випуск Apple Lisa, першого персонального комп’ютера з інтуїтивним графічним інтерфейсом та мишею.

1969 – 1973 р. – Денісом Рітчі створено мову програмування С, яка згодом використовується для написання операційної системи UNIX

П’яте покоління (1984 – …). Головною відмінністю обчислювальних систем, починаючи з другої половини 80-х років стало зростання ОТ з великою кількістю процесорів і, як наслідок, розвиток технології паралельних та розподілених обчислень. Це, здебільшого, стосувалось спеціалізованих та супер ЕОМ. Приблизно з 2005 р. почали з’являтися персональні комп’ютери з двома чи декількома ядрами.

Окремо слід виділити розвиток хмарних технологій в період 2009 – 2011 рр.

Також, до п’ятого покоління іноді відносять обчислювальні системи, побудовані на мікроконтролерах, які використовуються для керування технологічними процесами на виробництві та у побуті (системи розумний дім). А також обчислювальні системи, побудовані на базі мобільних пристроїв (вбудовані системи).

2. Архітектура комп’ютера. Принципи фон Неймана.

Уперше термін «архітектура комп’ютера» був введений фірмою IBM при розробці обчислювальних систем серії IBM 360 і застосований до тих засобів, які може використовувати програміст під час написання програм на рівні машинних команд.

Під цим терміном розуміли структуру комп’ютера з покляду програміста, так звану логічну архітектуру, що є поняттям архітектури у вузькому розумінні. Воно охоплює формати команд, форми зображення даних, методи адресації і т.д. При цьому з розгляду випадають такі вспекти, як фізична організація пристроїв комп’ютера, ієрархія пам’яті, методи паралельної обробки інформації і т.д.

Далі будемо використовувати термін «архітектура», розуміючи під цим поняттям логічну структуру у сукупності з фізичною структурною організацією.

В основу архітектури більшості сучасних комп'ютерів покладено принципи, які ще 1946 р. були сформульовані у звіті «Попереднє обговорення логічного конструювання електронного обчислювального пристрою» Джоном фон Нейманом і його колегами Г. Голдстайном та А. Берксом. Усі комп'ютери, побудовані згідно з цими принципами, тепер відомі як комп'ютери з фоннеймановською архітектурою.

Основні принципи архітектури фон Неймана такі:

- використання двійкової системи числення для кодування інформації у комп'ютері;
- програмне керування роботою комп'ютера;
- збереження програм у пам'яті комп'ютера;
- адресація пам'яті.

Зупинимось більш детально на кожному з принципів.

Принцип використання двійкової системи числення.

Інформація (команди і дані) у комп'ютері кодуються у двійковій системі числення. Система числення – це система позначення чисел. У повсякденній практиці використовується десяткова система, в якій числа записуються за допомогою десяти арабських цифр 0, 1, 2, …, 9. У двійковій системі числення є тільки дві цифри: 0 та 1. Зручність використання двійкової системи в обчислювальній техніці обумовлена тим, що електронні перемикачі можуть перебувати тільки в одному із двох станів: увімкненому чи вимкненому. Ці стани можна кодувати двома цифрами: одиницею або нулем. Так само в двох станах у кожен окремий момент часу може перебувати канал передачі даних: «рівень напруги високий», «рівень напруги низький». Крім того, логіка висловлень є двійковою логікою: будь-яке висловлення в кожен момент є істинним або хибним. Якщо висловлення є істинним, йому відповідає значення 1, якщо хибне – значення 0.

Одна двійкова цифра називається бітом. За допомогою одного біта можна закодувати два інформаційних повідомлення, що умовно позначаються символами «0» та «1». Із двох бітів можна утворити коди чотирьох інформаційних повідомлень: «00», «01», «10», «11». У загальному випадку за допомогою n бітів можна закодувати 2^n інформаційних повідомлень.

Послідовність, що складається з восьми бітів, у сучасній обчислювальній техніці прийнято називати байтом. За допомогою одного байта можна закодувати 28=256 різних повідомлень і зобразити, наприклад, значення цілих чисел від 0 до 255.

Байт є одиницею об'єму пам'яті. Похідні одиниці кілобайт (1 Кбайт=210=1024 байт), магабайт (1 Мбайт=220=1048576 байт) і гігабайт (1 Гбайт=230=1073741824 байт).

Послідовностями двійкових цифр можна кодувати не лише числа, ай довільні інформаційні повідомлення. Зокрема, загальноприйнята система кодування ASCII ставить коди у відповідність 256 символам. Наприклад, латинській літері «a» відповідає код 9710=11000012.

Принцип програмного керування роботою комп'ютера.

Сучасний комп'ютер – це сукупність технічних і програмних засобів, які призначені для автоматизованої обробки дискретних даних відповідно до заданого алгоритму. Алгоритм – це основне поняття математики та обчислювальної техніки і згідно зі стандартом він визначається як «скінченний набір інструкцій, які описують процес розв'язування задачі за допомогою скінченної кількості операцій».

Усі обчислення, що виконує комп'ютер, мають бути зображені у пам'яті у вигляді програми, складеної з інструкцій, які прийнято називати командами. Кожна команда вказує на певну операцію, яку має виконати комп'ютер. Сукупність команд, що їх використовує конкретний комп'ютер, називається системою команд.

Програма складається з послідовності команд, які процесор виконує автоматично в певному порядку. Змігювати порядок виконання команд залежно від проміжних результатів обчислень дозволяють команди умовного та безумовного переходів. Заляки цьому можна багаторазово виконувати одну й ту саму послідовність команд програми, а отже, набагато скоротити її розміри.

Принцип збереження програм у пам'яті комп'ютера.

Цей один із найважливіших принципів, який називають також принципом однорідності пам'яті, полягає в тому, що команди та дані зберігаються в одних і тих самих областях пам'яті і нерідко кодуються тими самими станами комірок пам'яті. Комп'ютер обробляє і команди, і дані однаково, тобто над командою можна виконати ті самі операції, що й над будь-якими даними. Це розкриває нові можливості для перетворення програми безпосередньо при її виконанні. Наприклад, команди однієї частини програми можна отримати як результат виконання команд іншої. Ця можливість покладена в основу трансляторів, які перетворюють текст програми на мові високого рівня у послідовність команд конкретного комп'ютера. Принцип однорідності пам'яті, запропонований у статті Дж. фон Неймана, використовувався в обчислювальних системах, які строювались у Прінстонському університеті, тому архітектура таких обчислювальних

систем дістала назву прінстонської рахітектури. Прінстонська архітектура характерна для більшості персональних комп'ютерів. Відмінною є гарвардська архітектура, її основним принципом є різний простір для збереження програм і даних.

Принцип адресності пам'яті.

Пам'ять машини Дж. фон Неймана – це лінійна (впорядкована) і однорідна послідовність деяких елементів, які називають комірками. У будь-який елемент пам'яті інші пристрої машини можуть записати і зчитата інформацію, причому час запису та читання з будь-якої комірки однаковий для усіх комірок. Це і є принцип однорідності пам'яті.

Для виконання програми необхідно, щоб команди та дані перебували в основній пам'яті. Основна пам'ять – це послідовність комірок, кожна з яких має свій номер – адресу. Розмір комірки зазвичай дорівнює восьми двійковим розрядам – байту. Числове значення у пам'яті, як правило, займає декілька сусідніх байтів.

Адресою числа вважається адреса першого байта області пам'яті, відведеної для збереження числа. Так, якщо 32-розрядне двійкове число зберігається в комірках 200, 201, 202 та 203, то його адресою буде 200. Такий метод адресації називається адресацією молодшого байта і використовується в комп'ютерах фірми Intel. Існує й інший метод, коли адресою числа є його старший байт. Такий метод використовується в комп'ютерах фірм IBM Motorolla.

Процесор у будь-який момент може отримати доступ до будь-якої комірки пам'яті. Така пам'ять називається пам'яттю з довільним доступом.

Архітектура комп'ютерів фон Неймана.

У класичній роботі Дж. фон Неймана перелічено основні пристрої, з використанням яких може бути побудований комп'ютер. Вважають, що комп'ютери такого типу мають фоннеймановську архітектуру. Типова фоннеймановська обчислювальна машина має такі складові: арифметико-логічний пристрій, пристрій керування, пам'ять і пристрої введення-виведення (рис. 1).

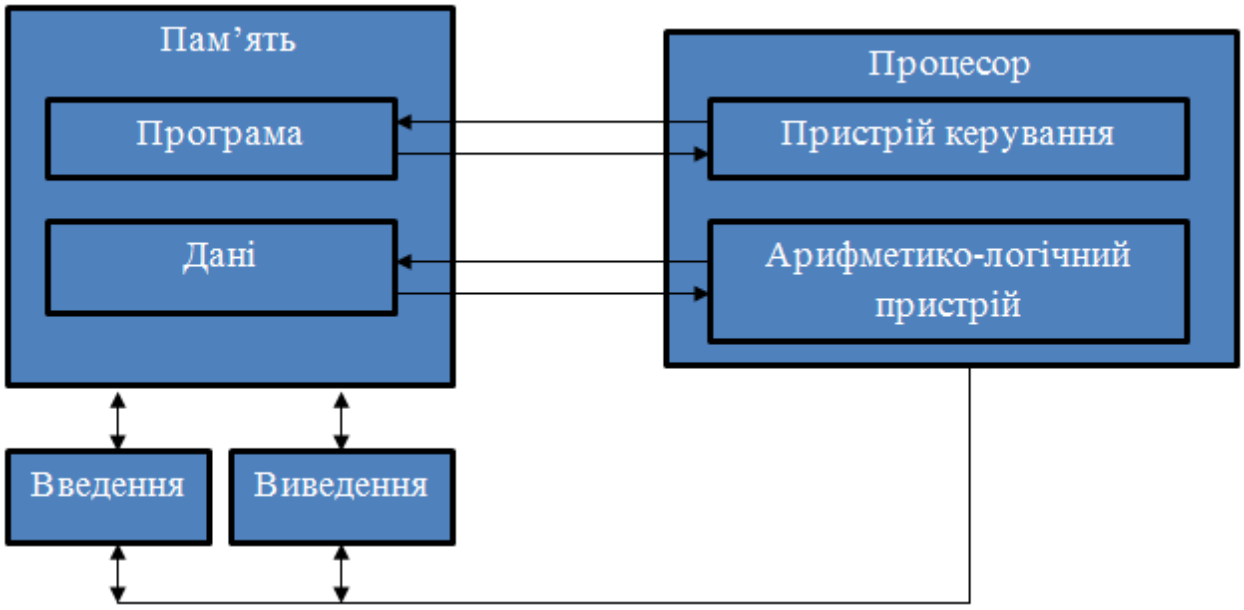

Рис. 1

В обчислювальних машинах із класичною архітектурою функції обробки інформації покладені на **арифметико-логічний пристрій** (АЛП). До функцій АЛП входить у першу чергу виконання арифметичних і логічних команд. АЛП – це не один, а ціла група операційних пристроїв, кожен з яких реалізує певну підмножину операцій.

Пристрій керування (ПК) – найголовніша частина комп’ютера, що координує роботу всіх його пристроїв та забезпечує виконання всіх програм. Основною функцією цього пристрою є формування сигналів, необхідних для вибирання команд із пам’яті в порядку що задається програмою, і подальше виконання цих команд. Крім того, ПК формує сигнали, необхідні для синхронізації та координації дій зовнішніх і внутрішніх пристроїв комп’ютера. Синхронізуючі сигнали – це сигнали, що визначають, коли має бути виконана певна операція. Реальні синхронізуючі сигнали, які керують пересиланням даних і виконанням команд, генеруються схемами керування.

АЛП і ПК дуже тісно взаємодіють між собою, і їх часто реалізують єдиним пристроєм, відомим як центральний процесор, чи просто процесор.

Пам’ять комп’ютера призначена для збереження інформації й оперативного обміну нею з іншими компонентами комп’ютера. Пам’ять поділяють на внутрішню та зовнішню. *Внутрішня пам’ять* складається з регістрів процесора, основної пам’яті та кеш-пам’яті.

Регістри процесора – це найбільш швидкодіючий, але найменший за обсягом різновид пам’яті комп’ютера. Зазвичай регістрів у процесорі небагато, і тільки в комп’ютерах з неповним набором команд (Reduced Instruction Set Computer, RISC) їх кількість може сягати кількох сотень. Регістри залежно від свого призначення можуть мати обсяг від 1 до 10 байт.

Основна пам’ять може включати пам’ять двох типів – постійну й оперативну.

Постійна пам’ять (Read Only Memory, ROM) реалізується у вигляді постійного запам’ятовуючого пристрою, дані в якому не можна модифікувати, їх можна тільки читати. Основне призначення ROM – підтримання процедур початкового завантаження операційної системи та обслуговування переривань, які припиняють виконання програми з метою реалізації спеціальних системних дій.

Оперативна пам’ять (Random Access Memory, RAM) використовується і для читання і для запису інформації. Під час роботи комп’ютера в ній зберігаються програми та дані. Оперативна пам’ять комп’ютера організована у вигляді множини байтів, або комірок, у яких зберігаються числові та символьні значення. Байти оперативної пам’яті послідовно пронумеровані починаючи з 0. Ці номери байтів є їх адресами. Залежно від типів даних, над якими виконуються дії, байти групуються у слова (два байти) або подвійні слова (чотири байти). Оперативна пам’ять сучасних персональних комп’ютерів сягає кількох гігабайтів. Особливістю RAM є неможливість зберігання інформації в ній після вимикання живлення комп’ютера. Саме цим вона відрізняється від пам’яті ROM, в якій дані зберігаються незалежно від наявності живлення.

Кеш-пам’ять – це високошвидкісна пам’ять довільного доступу, яка використовується процесором комп’ютера для тимчасового зберігання даних. Коли процесору потрібні дані, він спочатку звертається до кеш-пам’яті, і тільки якщо там потрібні дані відсутні, відбувається звертання до оперативної пам’яті. Отже, кеш-пам’ять проскорює обмін даними між процесором і оперативною пам’яттю через зменшення кількості звертань до неї процесора. Розрізняють кеш-пам’ять першого рівня (виконується на одному кристалі з процесором і має об’єм порядку декілька десятків Кбайт), другого рівня (виконується на окремому кристалі, але в межах процесора, з об’ємом в сто і більше Кбайт) та третього рівня (виконується на окремих швидкодійних мікросхемах із розташуванням на материнській платі і має обсяг один і більше Мбайт).

Пристрої введення-виведення є важливою складовою будь-якого комп’ютера. Вони забезпечують взаємодію комп’ютера з навколишнім середовищем, користувачами, об’єктами керування та іншими комп’ютерами.

Зовнішні пристрої за їх призначенням можна розділити на два основних різновиди – для тривалого зберігання інформації та для взаємодії

комп’ютера із зовнішнім середовищем, а саме користувачами, іншими комп’ютерами й об’єктами керування.

Серед них можна виділити пристрої введення (клавіатура, сканер, миша, джойстик і т.д.), виведення (принтер, плотер), засоби зв’язку та телекомунікацій (мережні плати, модеми).

Більшість зовнішніх пристроїв мають свої процесори (контролери), які є простішими за центральний процесор і виконують інші набори команд. Контролери можуть переносити дані із зовнішніх носіїв до оперативної пам’яті чи навпаки. Кожному пристрою введення-виведення виділено окрему ділянку оперативної пам’яті – порт. Із нього пристрій бере дані для зовнішнього носія, записуючи їх, наприклад, на диск або на екран комп’ютера. І саме в порт записуються дані, що надходять з клавіатури або дисковода.

Сигнали синхронізації дій усіх пристроїв передаються по керуючих лініях – шинах. Шини комп’ютера повинні забезпечити передачу інформації між:

- Процесором та оперативною пам’яттю.
- Процесором і портами введення-виведення зовнішніх пристроїв.
- Оперативною пам’яттю і портами введення-виведення пристроїв у режимі прямого доступу до пам’яті.

Шина процесор-пам’ять забезпечує безпосередній зв’язок між процесором комп’ютера та основною пам’яттю. У сучасних комп’ютерах таку шину іноді називають шиною переднього плану і позначають як FSB (Front-Side Bus). Інтенсивний обмін даними між процесором та пам’яттю вимагає, щоб кількість інформації, яка передається за одиницю часу (секунду) цією шиною (пропускна спроможність шини), була якомога більшою. Від пропускної спроможності шини процесор-пам’ять значною мірою залежить продуктивність комп’ютерів із фоннеймановською архітектурою. Функції різних шин конструктори іноді покладають на єдину системну шину, але з погляду швидкодії краще, коли дані між процесором і пам’яттю передаються окремою шиною.

Фізично шини складаються з великої кількості паралельних металевих провідників – ліній, розміщених на системній платі чи кристалі мікросхеми. Серед ліній будь-якої шини можна виділити три функціональні групи: *шина адреси*, *шина даних* і *шина керування*.

Шиною адреси передаються адреси комірок пам’яті, номери регістрів процесора, адреси портів введення-виведення і т. ін. Кількість ліній, виділених для передачі адреси, становить ширину шини адреси і визначає максимально можливий обсяг пам’яті комп’ютера, який може адресуватися.

3. Поняття алгоритму.

Алгоритм – це скінченний набір інструкцій, які описують процес розв’язування задачі за допомогою скінченної кількості операцій.

Навіщо вивчати алгоритми? Для справжнього комп'ютерного професіонала існує дві причини: практична й теоретична.

Із практичної точки зору він повинен мати уявлення про стандартний набір основних алгоритмів, що відносяться до різних областей обчислювальної техніки. Крім того, він повинен уміти розробляти нові алгоритми й аналізувати їхню ефективність.

З теоретичної точки зору процес вивчення алгоритмів, що іноді називають алгоритмикою (algorithmics), вважається наріжним каменем інформатики (computer science).

Ще одна причина для вивчення алгоритмів полягає в тім, що цей процес розвиває аналітичне мислення. Зрештою, алгоритми можна розглядати як особливий підхід до рішення задач, коли важливі не стільки самі відповіді, скільки точні інструкції для їхнього одержання.

Дональд Кнут, один з видатних учених в області інформатики й історії алгоритмики, пише наступне:

«Добре навчений в області інформатики фахівець зобов'язаний знати, як працювати з алгоритмами: як їх створювати, змінювати, розуміти й аналізувати. Ці знання дозволять не тільки писати гарні комп'ютерні програми, але й стануть основою універсального розумового апарата, що надасть неоціненну допомогу при збагненні інших наук, будь те хімія, лінгвістика, музика й т.д. Причину цього можна пояснити в такий спосіб: часто говорять, що людина нічого не розуміє, поки не пояснить це комусь іншому. Я б перефразував це так: людина глибоко не розуміє предмет доти, поки не навчить цьому комп'ютер, тобто виразить що-небудь у вигляді алгоритму... Спроба формалізувати щось у вигляді набору алгоритмів приводить до більше глибокого розуміння суті речей, чим при їхньому осмисленні традиційним способом»

Алгоритм — це послідовність чітко певних інструкцій, призначених для рішення деякої задачі.

Алгоритм має наступні властивості:

1) Дискретність - процес перетворення даних, тобто на кожному кроці алгоритму виконується чергова одна операція;

2) Результативність - алгоритм повинен давати деякий результат;

3) Закінченість - алгоритм повинен давати результат за кінцеве число кроків;

4) Визначеність - всі приписання алгоритму повинні бути однозначні, зрозумілі користувачеві;

5) Масовість - алгоритм повинен давати рішення для цілої групи задач із деякого класу, що відрізняються вихідними даними;

6) Ввід – алгоритм має деяке число вхідних даних.

7) Вивід – в алгоритму є одне або декілька вихідних даних, тобто величин, що мають певний зв'язок із вхідними даними.

Приклад алгоритму пошуку НСД.

Позначимо функцію пошуку НСД (найбільшого спільного дільника) двох ненегативних цілих чисел m і n (причому m і n не можуть одночасно рівнятися нулю) через gcd (m, n). По визначенню ця функція повинна знайти найбільше ціле число, що ділиться без остачі як на m, так і на n. Давньогрецький математик Евклід з Олександрії (III ст. до н.е.), що прославився тим, що вперше систематично виклав курс геометрії, описав алгоритм рішення цієї задачі в одній зі своїх праць за назвою Початки. Виражаючись сучасною мовою, алгоритм Евкліда заснований на рекурентному обчисленні наступної рівності:

,

Тут вираз (m mod n) є остачею від ділення m на n. Виконання алгоритму закінчується, коли вираз (m mod n) стає рівним нулю. Оскільки gcd (m, 0) = m (зрозуміло чому?), останнє отримане значення m буде також бути НСД вихідних чисел m і n.

Наприклад, обчислення НСД пари чисел (60, 24) можна виконати в такий спосіб:

.

Опишемо алгоритм Евкліда більш формально. Спочатку словесно, потім - псевдокодом.

Обчислення НСД чисел m і n за допомогою алгоритму Евкліда

Крок 1 Якщо n = 0, повернути m як відповідь і закінчити роботу; інакше перейти до кроку 2.

Крок 2 Поділити націло m на n і привласнити значення остачі змінній r.

Крок 3 Привласнити значення n змінній m, а значення r — змінній n.

Перейти до кроку 1.

Алгоритм у вигляді псевдокоду.

Алгоритм Euclid (m, n)
// Алгоритм Евклида обчислює значення функції gcd (m, n)
// Вхідні дані: два ненегативних цілих числа m і n,
// які не можуть одночасно бути дорівнюють нулю
// Вихідні дані: найбільший загальний дільник чисел m і n

while $n \neq 0$ **do**
  $r \leftarrow m \mod n$
  $m \leftarrow n$
  $n \leftarrow r$
return m

Чому алгоритм Евкліда завжди завершується після кінцевого числа ітерацій?

Як і при рішенні більшості інших задач, існує кілька алгоритмів обчислення НСД.

Розглянемо два інших способи рішення цієї задачі.

Перший з них заснований на підборі найбільшого цілого числа - такого, щоб числа m і n ділилися на нього без остачі. Очевидно, що такий загальний дільник не може бути більше найменшого із чисел пари, яке можна записати як t = min{m, n}. Тому виконання алгоритму можна почати з перевірки того, чи діляться обоє числа, m і n, на t без остачі. Якщо це так, то число t є відповіддю; якщо ні, потрібно зменшити значення t на одиницю й знову виконати перевірку. (чи можемо ми переконатися, що в остаточному підсумку цей процес завершиться?). Наприклад, для розглянутої вище пари чисел (60,24), виконання алгоритму починається з перевірки числа 24, потім - 23 і т.д. доти, поки значення числа t не стане рівним 12, після чого алгоритм повинен завершити свою роботу.

Обчислення НСД чисел m і n методом послідовного перебору

Крок 1 Привласнити значення функції min {m, n} змінній t.

Крок 2 Розділити m на t. Якщо остача дорівнює нулю, перейти до кроку 3; інакше перейти до кроку 4.

Крок 3 Розділити n на t. Якщо остача дорівнює нулю, повернути t як відповідь і закінчити роботу; інакше перейти до кроку 4.

Крок 4 Відняти 1 з t. Перейти до кроку 2.

У чому відмінності розглянутого алгоритму від Алгоритму Евклида.

Третій спосіб пошуку НСД називається «шкільний метод».

Обчислення НСД чисел m й n розкладенням на прості множники.

Крок 1 Розкласти на прості множники число m.

Крок 2 Розкласти на прості множники число n.

Крок 3 Для простих множників чисел m і n, знайдених на кроці 1 і 2, виділити їхні загальні дільники. (Якщо p є загальним дільником чисел m і n і зустрічається в їхньому розкладанні на прості множники, відповідно, pm і pn раз, те при виділенні потрібно повторити це min {pm,pn} раз)

Крок 4 Обчислити добуток всіх виділених загальних дільників і повернути його як результат пошуку НСД двох зазначених чисел.

Таким чином, для розглянутої вище пари чисел (60,24), одержимо:

60 = 2 * 2 * 3 * 5

24 = 2 * 2 * 2 * 3

gcd(60,24)=2*2*3 = 12.

Чи є описаний «шкільний» метод знаходження НСД дійсно алгоритмом? Перевірте по властивостях алгоритму.

Розглянемо нескладний алгоритм генерації послідовності простих чисел, що не перевищують довільно заданого цілого числа n.

Найімовірніше він був придуманий у древній Греції й тому названо решетом Ератосфена (прибл. 200 рік до н.е.). Для початку складемо список, що містить послідовність цілих чисел від 2 до n, з якого потім ми повинні будемо отримати прості числа. Далі, на першому проході алгоритму потрібно видалити зі списку всі числа, які діляться на 2, тобто 4, 6 і т.д. Потім необхідно вибрати зі списку наступний елемент (у цьому випадку це 3) і видалити зі списку всі числа, які діляться на нього. (В описуваної простої версії алгоритму існує невелика накладка, оскільки деякі із чисел, наприклад 6, повинні видалятися зі списку більше одного разу.) Для числа 4 не потрібно виконувати спеціальний прохід за списком, тому що число 4 кратне 2, тому воно вже буде вилучено зі списку на першому проході. (Точно так само в цьому алгоритмі не потрібно виконувати прохід і для всіх інших чисел, які були вилучені зі списку на попередніх проходах.) Наступним числом, що залишилося в списку й використається на третьому проході, є 5. Робота

алгоритму триває так доти, поки в списку існують числа, які можна видалити. Числа, що залишилися в списку після виконання алгоритму, є простими.

Як приклад використання описаного вище алгоритму розглянемо процес пошуку простих чисел, що не перевищують n = 25:

Приведемо алгоритм у вигляді псевдокоду.

Алгоритм Sieve (n) (решето Эратосфена)

// Реалізація решета Эратосфена

// Вхідні дані: Позитивне ціле число n>=2

// Вихідні дані: Масив L простих чисел, менших або

// рівних n

```
for p ← 2 to n do
    A[p] ← p
for p ← 2 to ⌊√n⌋ do
    if A[p] ≠ 0
        j ← p * p
        while j ⩽ n
            A[j] ← 0
            j ← j + p
```

У чому різниця наведеного псевдокоду від словесного опису алгоритму?

Схема розв'язання задач на ЕОМ

Алгоритми можна вважати процедурним рішенням задач. Причому ці рішення є не стільки відповідями, скільки точно певними інструкціями для одержання відповідей. Саме цей акцент на точності визначення конструктивних процедур відрізняє інформатику від інших дисциплін, зокрема від теоретичної математики.

Перелічимо й коротко опишемо послідовність етапів проектування й аналізу алгоритмів

Розуміння задачі

Існує кілька типів задач, які при створенні комп'ютерних додатків зустрічаються найчастіше. Їхній огляд буде зроблений далі.

Вхідні дані, оброблювані алгоритмом, визначають екземпляр задачі (problem's instance), розв'язуваної за допомогою обраного алгоритму. При цьому дуже важливо точно вказати границі прикладів, які повинні враховуватися в алгоритмі.

Визначення можливостей обчислювального пристрою

Повністю усвідомивши суть поставленої задачі, необхідно оцінити можливості обчислювального пристрою, для якого створюється алгоритм. Переважна більшість сучасних алгоритмів призначено для створення на їхній основі програми, що працює на комп'ютері, що сильно нагадує за структурою машину фон Неймана. Суть цієї архітектури виражена в її назві - машина з довільним доступом (random-access machine, або RAM). Передбачалося, що команди повинні послідовно вибиратися з пам'яті й виконуватися спеціальним пристроєм, називаним центральним процесором, причому в кожний момент часу в машині Неймана могла виконуватися тільки одна команда. Тому алгоритми, розроблені для виконання на такій машині, назвали послідовними (sequential algorithms).

Поява машини Неймана не зупинила прогрес в області обчислювальної техніки. Незабаром були придумані комп'ютери, які могли одночасно (тобто паралельно) виконувати кілька команд. Тому алгоритми, розроблені для таких машин, назвали паралельними (parallel algorithms). Проте вивчення класичних методів проектування й аналізу алгоритмів для машини Неймана ще довго буде залишатися наріжним каменем алгоритмики.

Вибір між точним або наближеним методом розв'язання задачі.

Наступне принципове питання - вибір точного або наближеного методу рішення задачі. У першому випадку алгоритм називається точним (exact algorithm), а в другому - наближеним (approximation algorithm). Чому для рішення задачі іноді вибираються наближені алгоритми? По-перше, існують задачі, які не можна вирішити точно. Як приклад можна привести добування квадратного кореня, рішення нелінійних рівнянь або обчислення певних інтегралів. По-друге, що існують алгоритми для точного рішення задачі можуть бути неприпустимо повільними, якщо її складність досить висока. Найбільш відомою з таких задач є задача комівояжера (traveling salesman problem), що полягає в пошуку найкоротшого маршруту між n містами.

Вибір відповідних структур даних

У деяких алгоритмах не потрібно, щоб вхідні дані були подані в якомусь специфічному форматі. Однак так буває не завжди, більше того, для роботи багатьох алгоритмів потрібні зовсім певні структури даних.

Методи проектування алгоритмів

Метод проектування алгоритму (algorithm design technique) (або "стратегія", або "принцип") - це універсальний підхід, застосовуваний для алгоритмічного рішення широкого кола задач, що ставляться у різних областях обчислювальної техніки.

Методи подання алгоритмів

Як приклад ми описали алгоритм Евклида як словами (тобто у вільній формі у вигляді послідовності виконуваних дій), так і у вигляді псевдокоду. У нашому курсі для подання алгоритмів найчастіше використаються ці дві форми.

Псевдокод (pseudocode) являє собою суміш однієї із природних мов і конструкцій, характерних для мови програмування.

Які ще способи подання алгоритмів вам відомі?

Оцінка коректності алгоритму

Після подання алгоритму в якій-небудь формі, необхідно оцінити його коректність (correctness). Це означає, що ви повинні довести, що обраний алгоритм за обмежений проміжок часу видає необхідний результат для будь-яких коректних значень вхідних даних.

Аналіз алгоритму

Звичайно творці алгоритмів намагаються, щоб вони задовольняли декільком вимогам. Після перевірки коректності алгоритму однією з найважливіших характеристик є оцінка його ефективності. На практиці існує два види оцінки ефективності алгоритму: часова й просторова.

Часова ефективність (time efficiency) є індикатором швидкості роботи алгоритму. Що стосується просторової ефективності (space efficiency), то ця

оцінка показує кількість додаткової оперативної пам'яті, необхідної для роботи алгоритму. Загальна ідея й конкретні методи аналізу ефективності алгоритмів будуть розглянуті на наступній лекції.

Ще однією важливою характеристикою алгоритму є його простота (simplicity). На відміну від ефективності, яку можна точно визначити й оцінити за допомогою строгих математичних виражень, простота алгоритму чимсь нагадує людську красу, поняття якої настільки суб'єктивно, що виробити об'єктивні критерії її оцінки досить непросто.

Наступною важливою характеристикою алгоритму є його спільність, або універсальність (generality). По суті, тут можна виділити два моменти: спільність задачі, для рішення якої розроблений алгоритм, і діапазон припустимих значень його вхідних даних.

Важливі типи задач

Серед величезної кількості задач, що зустрічаються в обчислювальній техніці, можна виділити кілька типів, яким вчені завжди приділяли особливу увагу. Подібний інтерес викликаний або практичним значенням задачі, або якимись її властивостями, що представляють особливий інтерес для дослідження. На щастя, у більшості випадків ці причини взаємозалежні, що тільки підсилює інтерес вчених.

Коротко розглядаються найбільш важливі типи задач:

- сортування;
- пошук;
- обробка рядків;
- задачі з теорії графів;
- комбінаторні задачі;
- геометричні задачі;
- чисельні задачі.

Ці задачі будуть використатися надалі при вивченні алгоритмів, для ілюстрації різних методів проектування й аналізу алгоритмів.

Сортування

Задача сортування (sorting problem) полягає в упорядкуванні заданого списку яких-небудь елементів у зростаючому (убутному) порядку.

Пошук

Задача пошуку пов'язана зі знаходженням заданого значення, називаного ключем пошуку (search key), серед заданої множини (або мультимножини).

Обробка рядків

У зв'язку зі швидким збільшенням останнім часом кількості додатків, пов'язаних з обробкою нечислових даних, інтерес учених і фахівців-практиків усе більше залучають алгоритми обробки рядків. Рядком (string) називається послідовність символів, узятих із заздалегідь певного алфавіту. Практичний інтерес представляють, наприклад, текстові рядки, що складаються з букв, цифр і спеціальних символів; бітові рядки, що складаються з нулів і одиниць; послідовності генів, які можуть бути змодельовані за допомогою рядків символів, узятих із чотирьохсимвольного алфавіту {A, C, G, T}.

Проте варто відзначити, що алгоритми обробки рядків стали важливі для обчислювальної техніки дуже давно - з тих пір, як з'явилися перші мови програмування й відповідні їм програми - компілятори.

Існує одна специфічна задача, що привернула особливу увагу фахівців з інформатики. Мова йде про пошук заданого слова в рядку текста. Її назвали пошуком підстрок (string matching). Для виконання такого специфічного пошуку розроблено кілька алгоритмів.

Задачі з теорії графів

Однієї із самих старих і, мабуть, найцікавіших областей алгоритмики є обробка графів. Нестрого граф можна визначити як набір точок, називаних вершинами, частина з яких з'єднана відрізками, називаними ребрами.

Графи є досить цікавим об'єктом для вивчення як з теоретичної, так і із практичної точок зору. За допомогою графів можна змоделювати досить велику кількість процесів, що відбуваються в реальному житті, наприклад функціонування транспортних і комунікаційних мереж, календарне планування проекту й т.д.

Комбінаторні задачі

Суть цих задач в остаточному підсумку зводиться до знаходження такого комбінаторного об'єкта, як перестановка, сполучення або підмножина,

який би задовольняв певним обмеженням і мав задані властивості (наприклад, дозволяв максимізувати прибуток або мінімізувати витрати.)

Загалом кажучи, комбінаторні задачі відносять до класу самих складних як з теоретичної, так і із практичної точок зору. Їхня складність обумовлена наступним. По-перше, кількість комбінаторних об'єктів звичайно дуже швидко росте при збільшенні масштабу задачі й досягає неуявних значень уже при досить скромних її масштабах. По-друге, поки не існує алгоритмів для пошуку точного рішення подібних задач за прийнятний час.

Геометричні задачі

Геометричні алгоритми пов'язані з такими геометричними об'єктами, як точки, лінії, багатокутники. Ми розглянемо алгоритми рішення тільки двох класичних задач обчислювальної геометрії: пошуку пари найближчих точок і визначення опуклої оболонки.

Чисельні задачі

Чисельні задачі відносяться до ще однієї досить великої області практичного використання алгоритмів. Вони мають справу з математичними об'єктами, які по своїй суті є безперервними. От приклади типових чисельних задач: рішення рівнянь і систем рівнянь, обчислення певних інтегралів і значень функцій і т.д. Переважна більшість таких задач може бути вирішено тільки приблизно. Ще одні принципові труднощі полягають у тім, що при рішенні подібних задач звичайно потрібно виконувати операції з дійсними числами, які в комп'ютері можуть бути представлені тільки з певною погрішністю.

Резюме лекції

Таким чином, на цій лекції були розглянуті такі питання:

1. Інтуїтивне поняття алгоритму.
2. Формалізоване поняття алгоритму.
- Машина Поста.
- Машина Тюрінга.
- Нормальний алгоритм Маркова.
3. Приклад алгоритмів пошуку НСД.
4. Розглянуто основні типи задач.

ЛІТЕРАТУРА

1. Ахо А., Хопкрофт В., Ульман Д. Структуры данных и алгоритмы. : Пер. с англ. : Уч. пос. – М. : «Вильямс», 2010. – 384 с.
2. Вирт Н. Алгоритмы и структуры данных. – «Невский Диалект», 2008. – 352 с.
3. Каррано Ф.М., Причард Д.Д. Абстракция данных и решение задач на С++. - М. : «Вильямс», 2003. – 848 с.
4. Ковалюк Т.В. Алгоритмізація та програмування. Львів: «Магнолія 2006», 2013. – 400 с.
5. Кормен Т., Лейзерсон Ч., Ривест Р. Алгоритмы. Построение и анализ. – М. : «Вильямс», 2007. – 1296 с.
6. Логинов В.И., Шемагина Л.Н. Основы алгоритмизации. – Н. Новгород: «ВГАВТ», 2010. – 81 с.
7. Левитин А. Алгоритмы: введение в разработку и анализ. : Пер. с англ. – М. : «Вильямс», 2006. – 576 с.
8. Потопахин В.В. Современный самоучитель по алгоритмам. – М.: ДМК Пресс, 2012. – 320 с.
9. Скиена С. Алгоритмы. Руководство по разработке. – СПб.: БХВ-Петербург, 2011. – 720 с.
10. Сэджвик Р. Функдаментальные алгоритмы на С. – К.: «ДиаСофт», 2001. – 688 с.

Додаткова:

1. Кнут Д. Искусство программирования, том 1. Основные алгоритмы. 3-е изд.: Пер. с англ.: Уч. пос. -М.: «Вильямс», 2000. – 720 с.
2. Кнут Д. Искусство программирования, том 3. Сортировка и поиск. 2-е изд.: Пер. с англ.: Уч. пос. -М.: «Вильямс», 2000. – 824 с.
3. Мартинюк Тетяна Борисівна. Рекурсивні алгоритми багатооперандної обробки інформації.- Вінниця: Універсум, 2000.- 216с.
4. Лекції курсу "Розробка та аналіз алгоритмів".- Запоріжжя: ЗДУ, 2000.- 54с.
5. Караванова, Тетяна Петрівна Основи алгоритмізації та програмування: 750 задач з рекомендаціями та прикладами: Посіб.- К.: ФОРУМ, 2002.- 287 с.
6. Кулик А.Я. Адаптивні алгоритми передавання інформації: Монографія. - Вінниця: Універсум, 2003.- 214с.