Лекція 1. Основні поняття про алгоритми.

План.

1. Інтуїтивні визначення алгоритму і його властивості.
2. Математичні визначення алгоритму.
3. Приклад алгоритму НСД.

1.

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 в студентів уміння аналітично мислити. 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.

Ілюстрація поняття алгоритм.

Алгоритмізація – це загальна послідовність дій, які необхідно виконати для побудови алгоритму рішення задачі, у тому числі – виділення конкретних

кроків алгоритмічного процесу, визначення виду формального запису для кожного кроку й установлення певного порядку виконання кожного кроку.

2.

Описане вище визначення алгоритму називається інтуїтивним, тому що воно розраховано на людське розуміння. В 30-х роках минулого сторіччя, математикам стало зрозуміло, що необхідно уточнити поняття алгоритму для рішення власних проблем математики. Математичні визначення алгоритму були отримані в середині 30-х років у роботах Д. Гильберта, К. Геделя, А. Черча, С. Клини, Э. Поста, А. Тьюринга, А. Маркова. Було розроблено три типи моделей.

Перший тип моделей заснований на понятті *рекурсивної функції.*

Другий – на основі опису детермінованого пристрою, що працює по кроках і виконуючому на кожному кроці заздалегідь визначені операції з елементами пристрою й даними (машини Поста, Тьюринга).

Третій тип моделей пов'язаний з роботою зі словами в деякому фіксованому алфавіті, які за допомогою підстановок переходять в інші слова (нормальний алгоритм Маркова).

Визначення алгоритму за допомогою рекурсивних функцій було зроблено А. Черчем і С. Клини. Воно достатньо складно й пов'язане з розділом математики - теорією вычислимых функцій.

Математики Э. Пост (США) і А. Тьюринг (Англія) незалежно друг від друга в 1936 році й майже в той же час, що й А. Черч, і С. Клини спробували уточнити поняття алгоритму й потім при його допомозі визначити точно й клас вычислимых функцій. Основна ідея Э. Поста й А. Тьюринга полягала в тім, що алгоритмічні процеси - це процеси, які може робити спеціально побудована «машина» (пристрій, автомат). Відповідно до цієї ідеї ними були описані в точних математичних термінах досить вузькі класи машин, але на цих машинах виявилося можливим здійснити (промоделювати) всі алгоритмічні процеси, які коли-небудь, описувалися математиками. Моделі, представлені цими

машинами, здійсненні на згаданих машинах було запропоновано розглядати як математичні визначення взагалі всіх алгоритмів.

Машинами ці математичні побудови називаються тому, що при побудові використаються деякі поняття реальних електронно-обчислювальних машин - пам'ять, команда, програма.

Машина Поста

Машина Поста складається з необмеженої в обидва боки стрічки, розділеної на осередки, які послідовно пронумеровані цілими числами, як позитивними, так і негативними. Стрічка відіграє роль пам'яті. У кожному осередку стрічки коштує або ознака того, що в осередку записаний влучна, або осередок порожня. Стан стрічки - це дані про те, які осередки зайняті, а які порожні.

Крім стрічки є голівка читання\запису, що:

- Уміє рухатися вперед, назад і стояти на місці;
- Уміє читати вміст, стирати й записувати мітку;
- Управляється програмою, у яку можуть входити в будь-якій комбінації й будь-якій кількості шість команд:

1. Вправо
2. Уліво
3. Поставити мітку
4. Стерти мітку
5. Передачі керування на один номер команди в програмі, якщо в поточному осередку є мітка, якщо мітки ні, те передача керування на інший номер команди.
6. Припинення роботи.

Стан машини - це стан стрічки й положення голівки читання\запису.

Машина Поста, незважаючи на зовнішню простоту, може робити різні обчислення, для чого треба задати початковий стан машини й програму, що ці обчислення зробить.

Машина вирішує наступну проблему: якщо для рішення задачі можна побудувати машину Поста, то вона алгоритмічно розв'язна, тобто для цієї задачі побудований алгоритм.

Чи можна будь-який алгоритм представити у формі машини Поста, тобто чи можна умова задачі записати в термінах машини Поста, і побудувати систему команд, що приводить до результату? Відповідь на це питання дається у вигляді так званої тези Поста.

Теза Поста. Усякий алгоритм представимо у формі машини Поста.

Це теза тому, що це твердження неможливо довести (чому?)

Машина Поста - це модель комп'ютера.

Машина Тьюринга

Машина Тьюринга складається з необмеженої в обидва боки стрічки, розділеної на осередки, які послідовно пронумеровані цілими числами, як позитивними, так і негативними.

У кожному осередку стрічки може стояти будь-який символ із заданого алфавіту, у якому виділений «порожній» символ - ознака того, що осередок порожня.

Машина має кінцеву множину внутрішніх станів, початкове (з його починається робота машини) і кінцевий стан, потрапивши в яке, машина припиняє роботу.

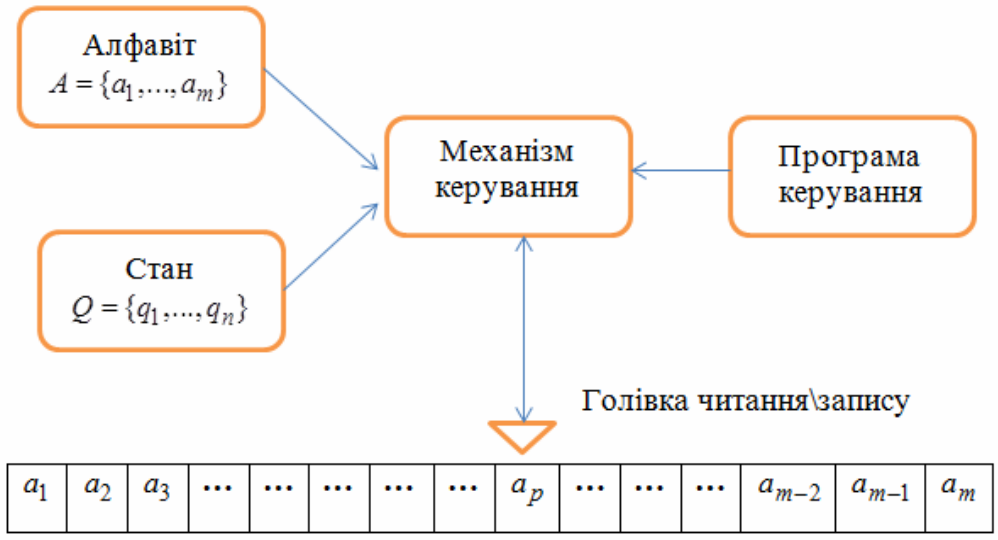
Крім стрічки є голівка читання\запису, що, по-перше, уміє рухатися вперед, назад і стояти на місці, по-друге, уміє читати вміст, стирати й записувати символи з даного алфавіту; у третіх, управляється програмою.

Програма являє собою таблицю, у якій у кожній клітці записана команда. Кожна клітка визначається двома параметрами - символом алфавіту й станом машини. Команда являє собою вказівку, куди пересунути голівку читання\запису з поточного стану, який символ записати в поточний осередок і в який стан перейде машина.

Машина Тьюринга, як і машина Поста - це модель комп'ютера.

Машина Тьюринга, аналогічно машині Поста, вирішує наступну проблему: якщо для рішення задачі можна побудувати машину Тьюринга, те вона алгоритмічно розв'язна.

І машина Тьюринга, і машина Поста еквівалентні по своїх можливостях.



Схематичне зображення машини Тюрінга

Теза Тьюринга. Усякий алгоритм представимо у формі машини Тьюринга.

Тези Тьюринга й Поста підтверджуються практикою.

Основні відмінності машини Поста й машини Тьюринга чисто технічні. Доведено їхню еквівалентність.

Нормальний алгоритм Маркова

Нормальний алгоритм Маркова - це математична побудова для уточнення поняття «Алгоритм», названо по імені автора А.А. Маркова.

Нормальний алгоритм Маркова задається алфавітом *A* и нормальною схемою підстановок.

Алфавіт - кінцева, непуста множина елементів називаних буквами. Різні сполучення букв утворять слова.

Нормальна схема підстановок - це кінцевий набір, що складається з пар слів, де ліве слово переходить у праве (але не навпаки).

Нормальним алгоритмом в алфавіті *A* називається наступний алгоритм побудови послідовності слів: як початкове слово береться саме слово *P* и к йому застосовують один по одному кожну пару зі схеми підстановок. Якщо підстановка можлива, то неї здійснюють і починають підстановки спочатку. Якщо процес обривається (немає ні однієї припустимої підстановки) на слові *Q* або приходить у кінцеву підстановку, то даний нормальний алгоритм перетворив *P* в *Q*.

Если є задача: від *P* перейти до *Q* і доведене, що не можна побудувати нормальну схему, то має місце алгоритмічно нерозв'язна задача.

Задаючи те ж питання: чи можна будь-який алгоритм представити у вигляді нормального алгоритму Маркова? Одержуємо аналогічну відповідь у вигляді так званої тези Маркова.

Теза Маркова. Усякий алгоритм в алфавіті *A* представимо у вигляді нормального алгоритму в цьому ж алфавіті.

Клас нормальних алгоритмів Маркова й клас алгоритмів представлених у формі машин Поста й Тьюринга збігаються.

Алгоритмічно нерозв'язні задачі

Однієї із причин математиків, що змусили, зайнятися питаннями формалізації поняття алгоритму, була необхідність одержати спосіб, що дозволяє визначити для деяких проблем існування або відсутність алгоритму для їхнього рішення.

Якщо для задачі побудований алгоритм - виходить, задача алгоритмічно розв'язна. А якщо не побудований, тобто два варіанти:

- Алгоритм поки не побудований.
- Алгоритм неможливо побудувати, і задача називається алгоритмічно нерозв'язної.

Приведемо приклад алгоритмічно нерозв'язних задач.

Проблема зупинки алгоритму. Для будь-якого алгоритму *A* и вихідних даних *a* потрібно визначити чи приведе виконання алгоритму *A* с вихідними даними *a* до результату. Більш формально: чи можна побудувати алгоритм *B*, що будучи застосуй до *A(a)* повертає Істину, якщо *A(a)* дає результат, і повертає Неправда в противному випадку.

Доказано, що ця задача алгоритмічно нерозв'язна, тобто побудувати алгоритм *У* неможливо.

3.

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 Евклид з Олександрії (III у 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 Евклида** заснований на рекуррентному обчисленні наступної рівності:

$$\gcd(m,n) = \gcd(n, m \bmod n),$$

Тут вираження (m mod n) є остачею від ділення m на n. Виконання алгоритму закінчується, коли вираження (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:

$$\gcd(60,24) = \gcd(24,12) = \gcd(12,0) = 12.$$

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

Обчислення НСД чисел m і n за допомогою алгоритму Евклида

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

Крок 2 Поділити нацело m на n і привласнити значення остачі змінної r.

Крок 3 Привласнити значення n змінної m, а значення r — змінної 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 ≠ 0 do
    r ← m mod n
    m ← n
    n ← r
return m
```

Чому алгоритм Евклида 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} змінної 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 «шкільним» методом

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, p_m і p_n раз, те при виділенні потрібно повторити це min $\{p_m,p_n\}$ 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 Эратосфена (прим. 200 рік до н.е.). Для початку складемо список, що містить послідовність цілих чисел від 2 до n, з якого потім ми повинні будемо вибрати 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 віддалятися 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

i ← 0
for p ← 2 to n do
    if A[p] ≠ 0
        L[i] ← A[p]
        i ← i + 1
return L
```

У чому різниця наведеного псевдокоду від словесного опису алгоритму?

Резюме

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.

ЛІТЕРАТУРА

Основна

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 C++. - 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.