Тема 2. Алгоритми. Форми та засоби їх представлення

План

1. Поняття алгоритму та їх властивості
2. Способи представлення алгоритмів
3. Типи алгоритмічних процесів

2.1. Поняття алгоритму та їх властивості

Теорія алгоритмів виникла у 30-ті роки XX ст. До того часу поняття алгоритму зводилося до набору елементарних кроків: арифметичних дій, перевірки рівностей, нерівностей та інших відношень такого типу. Але на початку XX ст. об'єкти, якими оперували алгоритми, почали ускладнюватися, з'явилася потреба описувати операції над векторами, таблицями, множинами тощо. Виникла думка, що не кожна математична задача може бути розв'язана за скінчений інтервал часу.

Алгоритм є фундаментальним поняттям інформатики – це наперед задана визначена послідовність етапів, яка призводить до вирішення поставленого завдання.

Розрізняють три основні класи алгоритмів: обчислювальні, інформаційні та управляючі.

1. Обчислювальні алгоритми – алгоритми, які працюють з порівняно простими видами даних, наприклад числами, векторами, матрицями (вирішують просту арифметичну задачу);

2. Інформаційні алгоритми – це набір процедур, що працюють з великими обсягами інформації. Прикладом такої процедури може бути пошук необхідної числової, або символьної інформації, що відповідає певним ознакам. Ефективність роботи цих алгоритмів залежить від організації даних, як, наприклад, у базах даних;

3. Управляючі алгоритми характеризуються тим, що вхідні дані до них надходять від зовнішніх процесів, якими вони керують. Результатами роботи цих алгоритмів є розробка своєчасного необхідного управляючого сигналу, як реакції на швидку зміну вхідних даних.

Швидкий розвиток інформаційних технологій, їхнє використання в різноманітних наукових дослідженнях привели до створення великої кількості алгоритмів у багатьох прикладних галузях.

Кожному алгоритму відповідає конкретна задача, яку він призначений розв'язувати. Але, разом з тим, може існувати й декілька алгоритмів розв'язання даної задачі. Такі алгоритми називаються еквівалентними. Тому виникає питання вивчення ефективності цих алгоритмів, що вивчається теорію складності алгоритмів.

Складність алгоритму – це його кількісна характеристика. Вона визначається часом, за який виконується алгоритм (часова складність), та обсягом пам'яті, яка необхідна для його виконання (ємнісна складність).

З'ясувавши поняття алгоритму можна зробити висновок, що будь-який алгоритм є послідовністю інструкції. Однак, не кожну послідовність інструкцій можна назвати алгоритмом. Для цього вона повинна мати певні властивості.

Дискретність. Будь-який алгоритм записується у вигляді окремих дій. Виконання команд алгоритму повинно бути послідовним, з точною фіксацією моментів завершення виконання однієї команди й початку виконання наступної.

Скінченність. Виконання алгоритму припиняється після завершення скінченної кількості кроків.

В математиці існують обчислювальні процедури, які мають алгоритмічний характер, але не володіють властивістю скінченності. Наприклад, обчислення значення числі «Пі» - числа після коми можна розраховувати нескінченно. Однак, якщо обмежитись певною кількістю знаків після коми, то отримаємо алгоритм обчислення числа з заданою точністю.

Визначеність. Кожний крок алгоритму має бути чітко й однозначно визначений, щоб не допустити довільного трактування виконавцем. Тобто, алгоритм розрахований на механічне виконання. Якщо один й той самий алгоритм доручити для виконання різним виконавцям, то вони повинні отримати один й той самий результат.

Зрозумілість. Формулювання кожного кроку алгоритму повинно бути зрозумілим для виконавця. Якщо алгоритм буде виконуватись на ПК, то його необхідно записати відповідною мовою програмування.

Масовість. В алгоритмі повинна бути передбачена можливість його виконання для різних початкових значень. Тобто, алгоритм можна використовувати для розв'язку цілого класу однотипних задач.

Результативність. Алгоритм повинний забезпечувати можливість отримання результату після скінченної кількості кроків.

Ефективність. Кожний крок алгоритму повинний бути достатньо простим та виконуватись за скінченний інтервал часу.

Як бачимо, для того, щоб набір інструкцій можна було назвати алгоритмом, він повинний задовольняти низці умов.

2.2. Способи представлення алгоритмів

Ми вже знаємо, що таке алгоритм. Тепер необхідно вирішити, яким чином ми будемо його описувати.

Існує чотири способи представлення алгоритмів: словесний, за допомогою блок-схем, мовою псевдокодів та мовою програмування. *Розглянемо їх більш детально.*

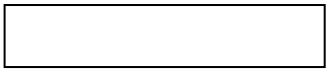
1. Словесний спосіб представлення алгоритмів – це найпростіша та найдоступніша форма представлення алгоритму. Словесна форма зазвичай використовується для алгоритмів, орієнтованих на людину-виконавця.

Прикладом словесного алгоритму може бути складання маршруту руху від крапки А до В.

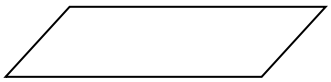
Перевірка роботи такого алгоритму є суттєвим кроком на шляху до його розуміння.

2. Представлення алгоритмів за допомогою блок-схем. Блок-схеми дозволяють зобразити алгоритм в наочній графічній формі. *Цей спосіб представлення алгоритмів потребує певної формалізації: як розуміти певні графічні зображення (блоки).*

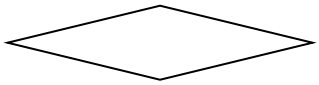
Початок та кінець алгоритму



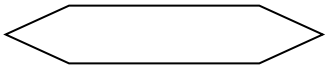
Виконання операцій, в результаті яких відбуваються розрахунки, змінюються значення змінних



Введення та виведення даних



Вибір напрямку виконання алгоритму



Початок циклічного повторення операцій

Блоки початку та кінця алгоритму використовуються при повному представленні алгоритму задачі.

Під час створення блок-схеми алгоритму, кожний з блоків із записаними в них командами з'єднуються між собою стрілками, які визначають черговість виконання етапів алгоритму.

Для запису команд всередині блоків використовується звичайна мова з елементами математичної символіки.

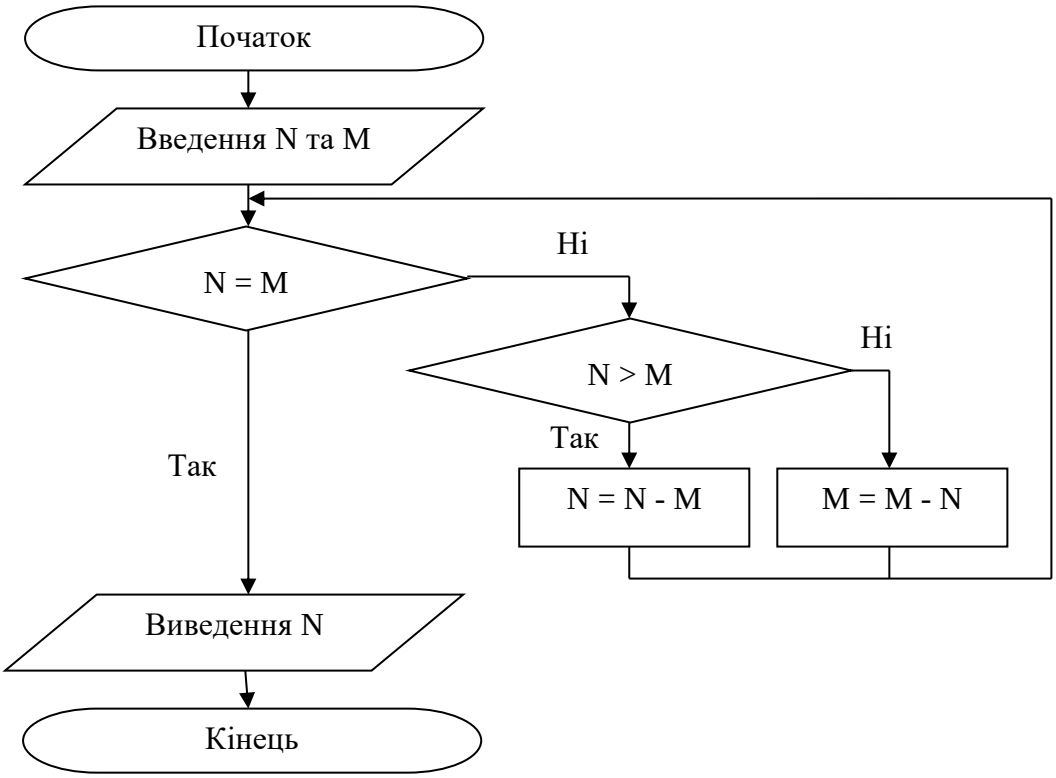
В результаті перевірки умови під час вибору напрямку виконання алгоритму, виникають два можливі шляхи для його продовження. Ці шляхи зображаються стрілками з позначеннями «так» й «ні». Перехід за стрілкою з позначенням «так» відбувається в тому разі, коли умова виконується, а перехід за стрілкою з позначенням «ні» - у протилежному випадку.

Приклад алгоритму знаходження найбільшого спільного дільника двох цілих додатних чисел наведений на рисунку нижче.

Наочність схематичного представлення алгоритму має свої переваги. Однак ця наочність швидко втрачається, якщо зображається великий алгоритм. У таких випадках у схемі алгоритму виділяються і відокремлюються її окремі частини - модулі, основною умовою яких є один вхід і один вихід. Згодом вони включаються у схему алгоритму як окремі блоки. Такий підхід до складання алгоритму відображає ідею структурного програмування.

3. Представлення алгоритмів мовою псевдокодів. Для запису алгоритмів за допомогою мови псевдокодів використовуються службові слова та спеціальні правила запису окремих дій.

У мові псевдокодів прийняті жорсткі синтаксичні правила для запису команд, що полегшує запис алгоритму на стадії його проектування і дає можливість використання широкого набору команд, розрахованих на абстрактного виконавця.



Приклад алгоритму знаходження найбільшого спільного дільника двох цілих додатних чисел:

```
АЛГОРИТМ найбільший_спільний_дільник
  ПОЧАТОК
    ВВЕДЕННЯ «Задайте два додатні цілі числа», n, m
    ПОКИ n≠m
    ПОЧАТОК
      ЯКЩО n > m
        ТО n: = n – m
        ІНАКШЕ m: = m – n
      ВСЕ
      КІНЕЦЬ
    ВИВЕДЕННЯ «Найбільший спільний дільник
    заданих чисел:», n
  КІНЕЦЬ
```

Мова псевдокодів максимально наближена до звичайної розмовної мови. Усі службові слова, що використовуються для запису алгоритму, виділені жирним шрифтом.

4. Представлення алгоритмів мовою програмування. Алгоритми, призначені для виконання на ПК, повинні бути записані відповідними мовами програмування. Тут на перший план виходить необхідність точного запису команд, що унеможливлює довільне трактування їх виконавцем.

Більшість мов програмування (Visual Basic, Pascal, C++, dBase) називаються мовами програмування високого рівня, бо їх конструкції максимально наближені до людської мови, зручні та зрозумілі користувачам. Мова Assembler є машинно-орієнтованою, мовою низького рівня, тобто

максимально наближеною до мови самого комп'ютера. *На перший погляд ця мова здається складною, але, розібравшись у роботі комп'ютера, ви зрозумієте її простоту й доступність.*

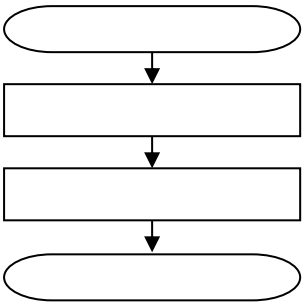
Надалі ми ознайомимося з можливостями мови програмування Visual Basic for Applications – інтерпретатором Visual Basic, впровадженим до офісних додатків.

2.3. Типи алгоритмічних процесів

Після ознайомлення з різними формами запису алгоритмів, перейдемо до вивчення типів алгоритмічних процесів.

Існує декілька типів алгоритмічних процесів: лінійні, розгалужені, циклічні та допоміжні алгоритми.

Лінійні алгоритми. Лінійним алгоритмом називається такий алгоритм, в якому всі його команди виконуються послідовно в часі, одна за одною:

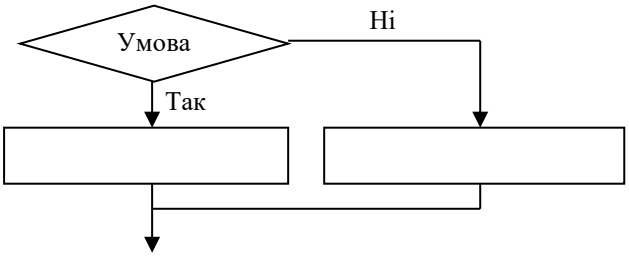


Кожна наступна команда починає виконуватись лише при повному завершення попередньої команди.

Приклад лінійного алгоритму: розпорядок дня, розпорядження керівника з переліком завдань, етапи виконання робочого процессу (створення сайту тощо).

При вирішенні реальних задач, лінійні алгоритми зустрічаються дуже рідко. Найчастіше можна зустріти лінійні фрагменти складних алгоритмів.

Розгалужені алгоритми. Алгоритм, що містить хоча б одну умову, в результаті перевірки якої здійснюється перехід до одного з можливих кроків, називається розгалуженим.

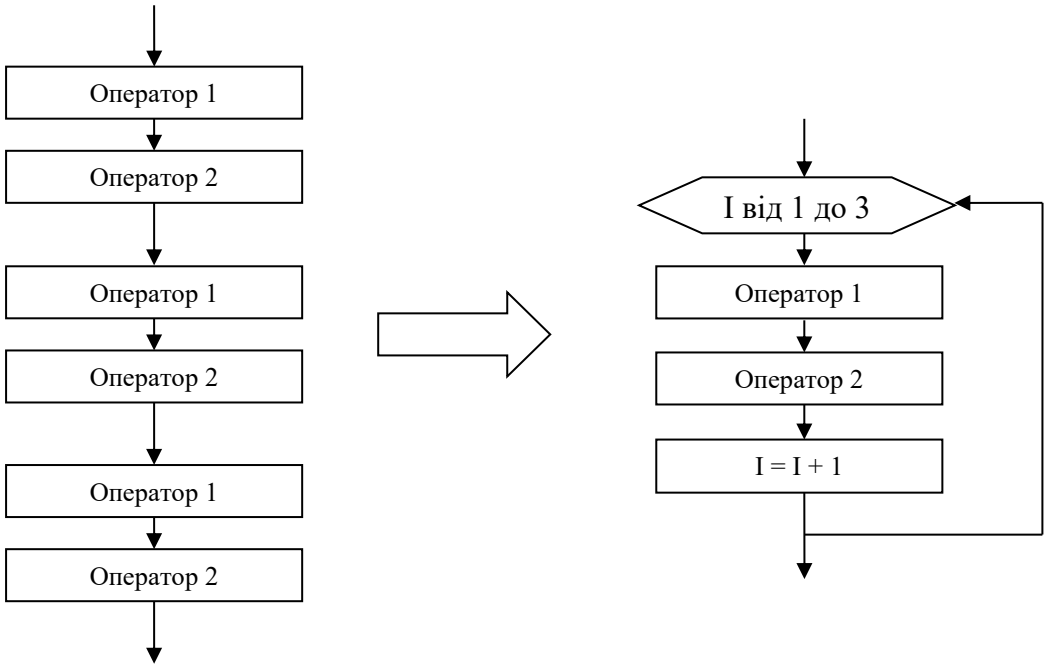


Досить часто вибір тих чи інших дій для продовження алгоритму залежить від виконання, або невиконання певних умов. Команди, які аналізують ці умови називаються командами розгалуження.

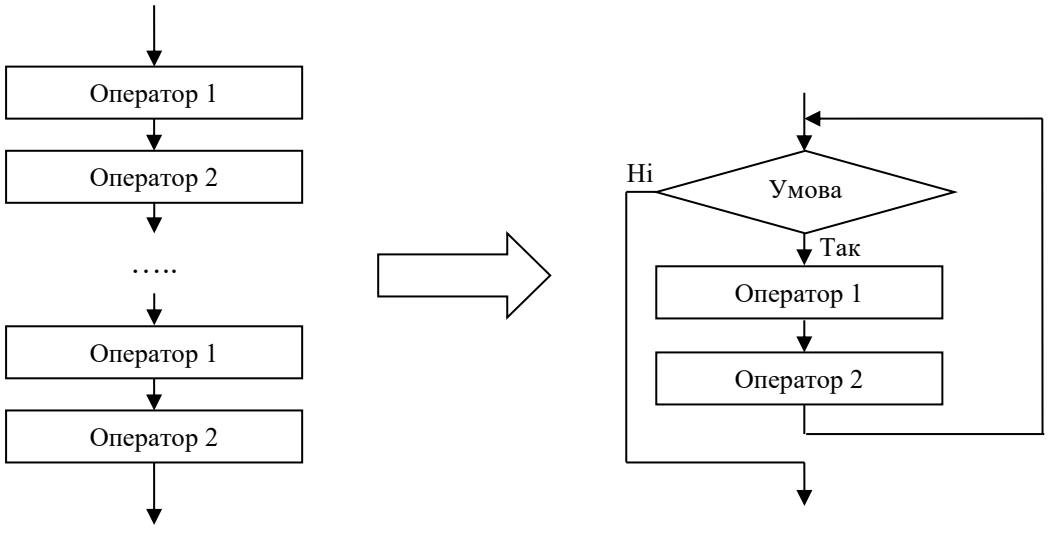
За аналогією з лінійними алгоритмами, мабуть, доцільніше говорити про розгалужені фрагменти алгоритмів, ніж про самі розгалужені алгоритми.

Циклічні алгоритми. Алгоритм, в якому певна послідовність команд повторюється декілька разів зі старими, або новими вхідними даними, називається циклічним.

Приклад 1: лінійний алгоритм перетворюється до циклічного вигляду, кількість повторень блоку команд є відомою.



Приклад 2: лінійний алгоритм перетворюється до циклічного вигляду, кількість повторень блоку команд є невідомою.



Необхідність у використанні циклічних команд виникає, коли необхідно декілька разів використовувати для обчислення одні й ті самі формули з різними значеннями змінних, або якісь інші однотипні команди.

Допоміжні алгоритми. При складанні алгоритмів часто виникають ситуації, коли одна й та сама послідовність операцій повинна неодноразово виконуватись в різних частинах алгоритму. Доцільно записати таку

послідовність команд лише один раз й звертатись до нєї у разі виникнення необхідності.

Допоміжним називається алгоритм, який створюється наперед, оформлюється у вигляді окремої процедури (підпрограми) та запускається у разі необхідності.

Використання допоміжних алгоритмів дозволяє:

– скоротити довжину первісного алгоритму;

– полегшити процес відладки програми.

Допоміжні алгоритми можуть бути:

– внутрішніми, локальними – створюються у межах початкового алгоритму та доступні для використання тільки в цьому алгоритмі;

– зовнішніми, глобальними – їх можна використовувати в різних незалежних алгоритмах. Такі допоміжні алгоритми найчастіше об'єднуються у бібліотеки (dll-файли).