МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ТЕРНОПІЛЬСЬКИЙ ДЕРЖАВНИЙ ЕКОНОМІЧНИЙ УНІВЕРСИТЕТ
ІНСТИТУТ КОМП'ЮТЕРНИХ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра інформаційно-обчислювальних систем та управління

ОПОРНИЙ КОНСПЕКТ ЛЕКЦІЙ

з дисципліни "Основи алгоритмізації" спеціальність "Комп'ютерні системи та мережі", освітньо-кваліфікаційний рівень "Бакалавр"

Тернопіль
Економічна думка
2005

Затверджено рішенням кафедри інформаційно-обчислюваних систем та управління (протокол №6 від 10 лютого 2005 року).

Васильків Н. М., Васильків Л. О. Опорний конспект лекцій з дисципліни “Основи алгоритмізації” спеціальність “Комп’ютерні системи та мережі”, освітньо-кваліфікаційний рівень “Бакалавр” – Тернопіль: Економічна думка, 2005. – 32 с.

Укладачі: Н. М. Васильків, ст. викладач;
Л. О. Васильків, стажист-дослідник.

Рецензенти: М. П. Карпінський, д. т. н., професор;
М.І.Чирка, к. т. н., доцент.

Відповідальний за випуск: А. О. Савченко, д. т. н., професор.

Опорний конспект лекцій з дисципліни “Основи алгоритмізації” містить найважливіші поняття та інформацію для розробки та складання схем алгоритмів розв’язку задач, схем даних, схем взаємозв’язку програм, схем роботи автоматизованих (комп’ютеризовани ` істем обробки інформації.

ЗМІСТ

ВСТУП

Фундаментом сучасної методології розробки програм є алгоритми. Програми, що створюються як вправи на початкових етапах навчання програмування, – це переважно нескладні завдання, записані однією з мов програмування. Мета цих вправ – допомогти студентам-першокурсникам вивчити елементи певної мови програмування; для знаходження способу їх виконання практично не потрібно великих розумових або часових затрат.

На наступних етапах навчання методи, за допомогою яких виконують завдання, набувають дедалі більшого значення, їх застосування потрібне для розроблення плану виконання завдання ще до того, як розв’язок буде виражений мовою програмування, тобто виникає необхідність створення алгоритму.

Для наочного представлення структури розв’язання складних задач особливо підходять схеми алгоритмів їх розв’язку, адже їх можна реалізувати будь-якою мовою програмування.

Загальними для всіх алгоритмів і найсуттєвішими є ознаки форми, структури і двосторонніх зв’язків між конструктивними об’єктами.

Крім цього, важливе значення мають не лише можливості нових поколінь обчислювальних засобів, а й часова складність вибраного алгоритму.

Вивчення основ алгоритмізації – необхідна умова поєднання теорії і практики програмування, частина математичної культури та загальної культури мислення.

1. ЕТАПИ РОЗВ'ЯЗУВАННЯ ЗАДАЧІ НА ЕОМ

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 розробленню 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 *відлагодженні 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'язанна 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".

2. ПОНЯТТЯ АЛГОРИТМУ

Одним з важливих понять, на яких базується застосування ЕОМ для розв'язування різноманітних задач, є поняття алгоритму.

Термін "алгоритм" звичайно використовується для позначення деякої послідовності дій, що приводять до досягнення потрібного результату.

Слово «алгоритм» є перефразуванням географічної назви місцевості Хорезм через праці відомого узбецького математика Мухамеда ібн Муса аль-Хорезмі (близько 825 року). У IX ст. великий узбецький математик Мухаммед, уродженець Хорезма (арабською "аль-Хорезмі"), розробив правила виконання чотирьох арифметичних дій над числами в десятковій системі числення. Множину цих правил назвали алгоритмом (algorithmi – від латинського написання імені аль-Хорезмі), а потім словом "алгоритм" почали позначати сукупність правил певного виду, а не тільки правил виконання арифметичних дій.

Одним із найперших алгоритмів є відомий алгоритм Евкліда для знаходження найбільшого спільного дільника натуральних чисел (III ст. до н. е.).

Тривалий час поняття алгоритму використовували лише математики при позначенні правил розв'язування різних задач. Розвиток математичної науки привів до необхідності уточнення поняття алгоритму як одного з основних математичних понять і розробки нової математичної дисципліни "Теорії алгоритмів".

Інтуїтивно алгоритм трактується як ефективна обчислювальна процедура в працях Бореля (1912 р.), Вейля (1921 р.) та ефективна обчислювальна функція у роботі Чорча (1936 р.). За Постом (1936 р.) алгоритм – це послідовність пронумерованих інструкцій. У термінах Тюрінга (1936 р.) алгоритм – непорожня послідовність команд. Еквівалентні їм інтуїтивні поняття алгоритму введені Марковим (1951 р.) і Колмогоровим (1953 р.). Подальші дослідження, здійснені Шенгаге, Ахо, Ульманом, Хопкрофтом і Криницьким, не вийшли за рамки інтуїтивного підходу.

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.

3. ВЛАСТИВОСТІ АЛГОРИТМУ

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

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.

4. СПОСОБИ ПРЕДСТАВЛЕННЯ АЛГОРИТМІВ

У процесі розроблення алгоритму можуть використовуватись різні способи його опису, які відрізняються за простотою, наочністю, компактністю, мірою формалізації, орієнтації на машинну реалізацію тощо.

Форми запису алгоритму:

- словесна або вербальна (мовна, формульно-словесна);
- псевдокод (формальні алгоритмічні мови);
- схемна:
  1) структурограми;
  2) графічна (виконується за вимогами стандарту).

4.1. Псевдокоди

Псевдокод – система позначень і правил, призначена для записування алгоритмів.

Він займає проміжне місце між звичайною і формальною мовою. Через свої особливості псевдокоди орієнтовані на людину.

У псевдокоді не вимагає дотримання синтаксичних правил для запису команд, які властиві формальним мовам, що полегшує запис алгоритму на стадії його проектування і дає можливість використати ширший набір команд, розрахований на абстрактного виконавця.

У псевдокоді звичайно є деякі конструкції, властиві формальним мовам, що полегшує перехід від запису їх псевдокодом до запису алгоритму формальною мовою.

Зокрема, у псевдокоді, як і в формальних мовах, є службові слова, зміст яких визначений раз і назавжди. Їх виділяють у друкованому тексті жирним шрифтом, а в рукописному – підкресленням:

початок, кінець, якщо, то, інакше, поки, повторювати, повторювати до.

Наприклад:

$$y = \begin{cases} x + 1, & x \geq 0 \\ 5x, & x < 0 \end{cases}$$

початок
 ввід x
 якщо $x \geq 0$ ***то*** $y = x + 1$
 інакше $y = 5x$
кінець

4.2. Структурограми

Спосіб зображення алгоритму за допомогою структурограми (схеми Нассі-Шнейдермана) реалізує в собі вимоги структурного програмування. Він дає змогу зобразити схему передачі управління не за допомогою ліній потоку, а вкладеними структурами.

Деякі із зображуваних графічних символів відповідають зображенню символів на схемах, виконаних згідно зі стандартами Єдиної системи програмної документації (ЄСПД).

Допустимим є використання таких блоків.

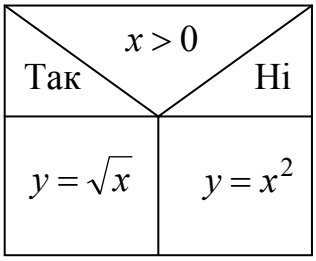
1. Блок обробки (обчислень):

$y = a + b$

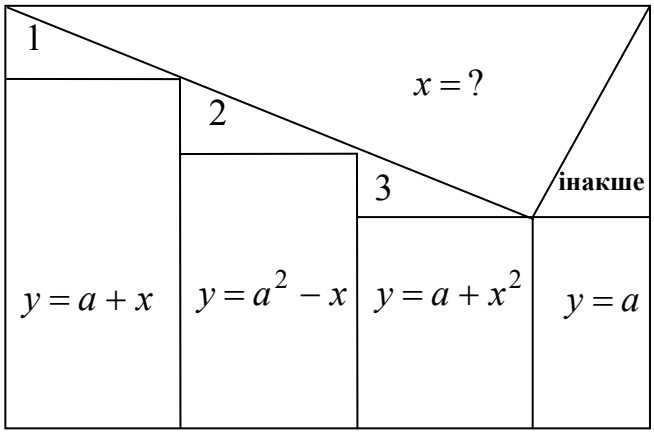
2. Блок послідовності:

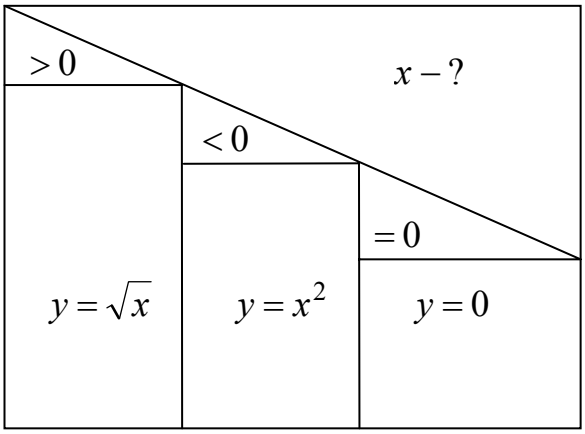
$y = a + b$
$z = y + c$

3. Блок розв’язання (для розгалужень):



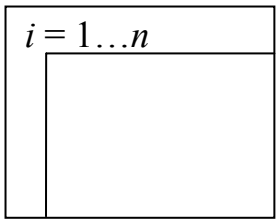
4. Блок варіанту:

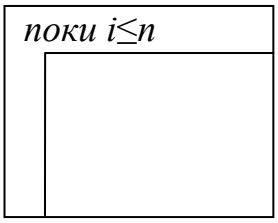




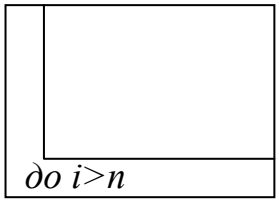
Ті варіанти, які можна точно сформулювати, розташовують зліва. Решту – об’єднують в один, що розташований справа і є виходом за недотриманням умови.

5. Блок циклу з параметром або циклу з передумовою:





6. Блок циклу з постумовою:



Кожен блок структурограми має форму прямокутника і може бути вписаний в будь-який інший. Блоки заповнюються формульно-словесно.

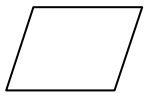
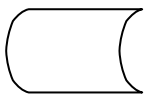
4.3. Графічне представлення алгоритмів згідно з вимогами стандартів ЄСПД

Схема в програмній документації – це графічне представлення визначення, аналізу або методу розв'язування задачі, в якому використано символи для відображення операцій, даних, потоку, обладнання тощо.

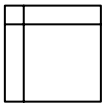
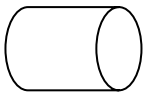
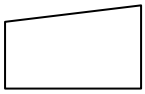
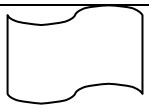
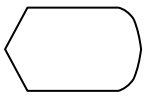
Схеми алгоритмів, програм, даних і систем складаються із символів, які мають встановлене значення (таблиця 1), короткого пояснювального тексту та з'єднувальних ліній.

Таблиця 1

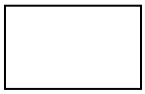
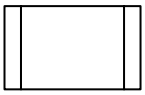
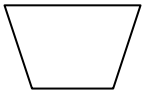
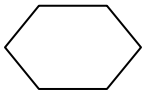
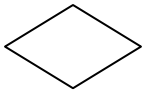
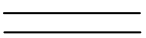
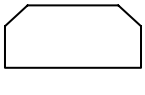
Стандартні символи

Зображення символа	Назва символа	Призначення
1	2	3
Символи даних		
	Дані	Символ відображає дані, носій яких не визначений
	Збережувані дані	Символ відображає дані, які зберігаються у вигляді, придатному для обробки, але їх носій не визначений

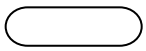
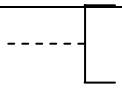
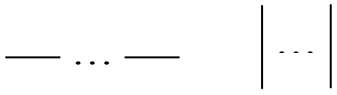
Продовження таблиці 1

1	2	3
	Оперативний запам’ятовуючий пристрій	Символ відображає дані, які зберігаються в оперативному запам’ятовуючому прист-рої
	Запам’ятовуюий пристрій з послідовним доступом	Символ відображає дані, які зберігаються в запам’ятовуючому пристрої з послідовним доступом (магнітна стрічка, магнітофонна касета тощо)
	Запам’ятовуюий пристрій з прямим доступом	Символ відображає дані, які зберігаються в запам’ятовуючому пристрої з прямим доступом (магнітний диск, магнітний барабан, гнучкий магнітний диск)
	Документ	Символ відображає дані, подані на носії в зручній для читання формі (машинограма, мікрофільм, бланки вводу даних та ін.)
	Ручний ввід	Символ відображає дані, які вводять вручну під час обробки з пристроїв будь-якого типу (клавіатура, перемикачі, кнопки, смужки зі штрих-кодом)
	Карта	Символ відображає дані, подані на носії у вигляді карти (магнітні карти, карти зі скануючими мітками та ін.)
	Паперова стрічка	Символ відображає дані, подані на носії у вигляді паперової стрічки
	Дисплей	Символ відображає дані, подані в зручній для сприйняття людиною формі на носії у вигляді відображуючого пристрою (екран для візуального спостереження, індикатори вводу інформації)

Продовження таблиці 1

1	2	3
Символи процесу		
	Процес	Символ відображає функцію обробки даних будь-якого виду
	Попередньо визначений процес	Символ відображає заздалегідь визначений процес, який складається з однієї або кількох операцій чи кроків програми, що визначені в іншому місці (в підпрограмі, модулі)
	Ручна операція	Символ відображає будь-який процес, який виконує людина
	Підготовка	Символ відображає модифікацію команди або групи команд з метою впливу на якусь наступну функцію (встановлення перемикача, модифікація індексного регістра)
	Розв’язання	Символ відображає розв’язання або функцію перемикального типу, яка має один вхід і декілька альтернативних виходів, один і тільки один з яких може бути активізований після обчислення умов, визначених всередині цього символа
	Паралельні дії	Символ відображає синхронізацію двох або більше паралельних операцій
 	Межі циклу	Символ складається з двох частин і відображає початок і кінець циклу. Обидві частини символа повинні мати один і той же ідентифікатор. Умови для приросту, завершення тощо поміщають усередині символа на початку чи в кінці залежно від розташування операції, яка перевіряє умову

Продовження таблиці 1

1	2	3
Символи ліній		
	Лінія	Символ відображає потік даних або управління.
	Передача управління	Символ відображає безпосередню передачу управління від одного процесу до іншого. Тип передачі управління вказують всередині символа.
	Канал зв’язку	Символ відображає передачу даних по каналу зв’язку.
	Пунктирна лінія	Символ відображає альтернативний зв’язок між двома чи більше символами, використовується для обведення анотованої ділянки схеми.
Спеціальні символи		
	З’єднувач	Символ відображає вихід у частину схеми і вхід з іншої частини цієї схеми. Використовується для обриву лінії і продовження її в іншому місці. Відповідні символи-з’єднувачі повинні мати одне і те ж унікальне позначення.
	Термінатор	Символ відображає вихід в зовнішнє середовище і вхід із зовнішнього середовища (початок або кінець схеми програми, зовнішнє використання і джерело чи пункт призначення даних).
	Коментар	Символ використовується для описових коментарів або пояслювальних записів.
	Пропуск	Символ (три крапки) використовується для відображення пропуску символа або групи символів, в яких не визначені ні тип, ні кількість символів.

5. ОСНОВНІ СТРУКТУРИ АЛГОРИТМІВ

Основні структури алгоритмів – це обмежений набір блоків і стандартних способів їх з'єднання для виконання типових послідовностей дій. Використання кількох основних структур дає можливість будувати різноманітні алгоритми.

До основних структур алгоритмів належать:

– **лінійна** або **послідовна** без будь-яких розгалужень конфігурація алгоритму, що нагадує форму ланцюжка (рисунок 1);

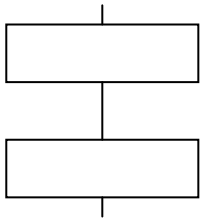


Рис. 1

– **розгалужена** конфігурація алгоритму, що містить в собі як послідовності, так і розпаралелення послідовностей. Використовується, коли залежно від умови потрібно виконати ту чи іншу дію (рис. 2, а), здійснити обхід, якщо одна вітка не містить жодних дій (рис. 2, б), здійснити множинний вибір, коли умова має більш як три можливі варіанти (рис. 2, в);

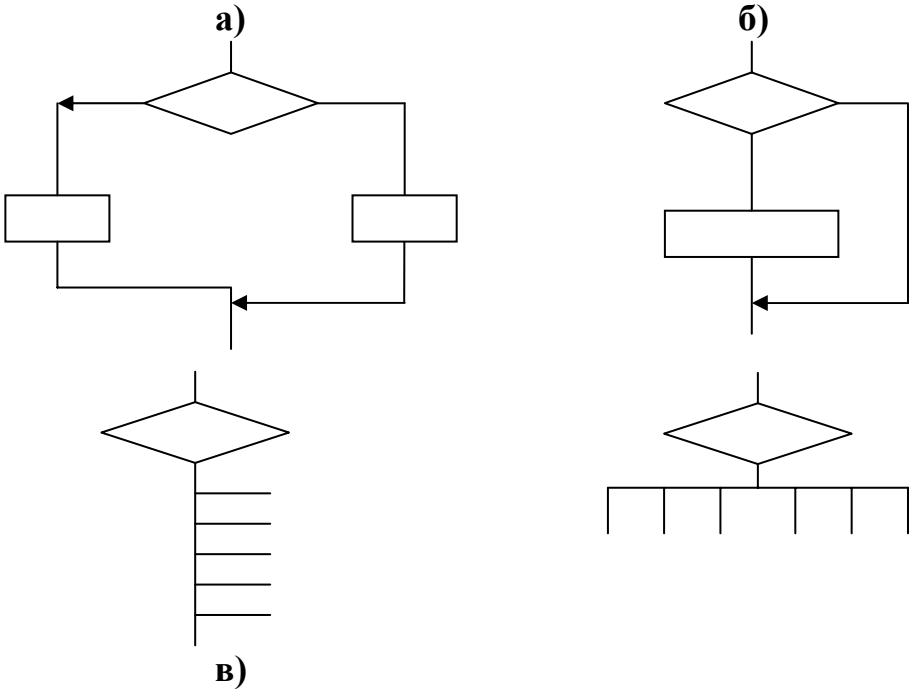


Рис. 2

– **циклічна,** щовикористовується при необхідності виконувати деякі дії кілька разів. Можливе виконання циклу ***До,*** циклу ***Поки,*** циклу за ***параметром*** (рис. 3).

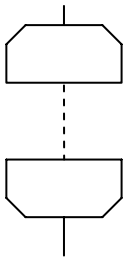
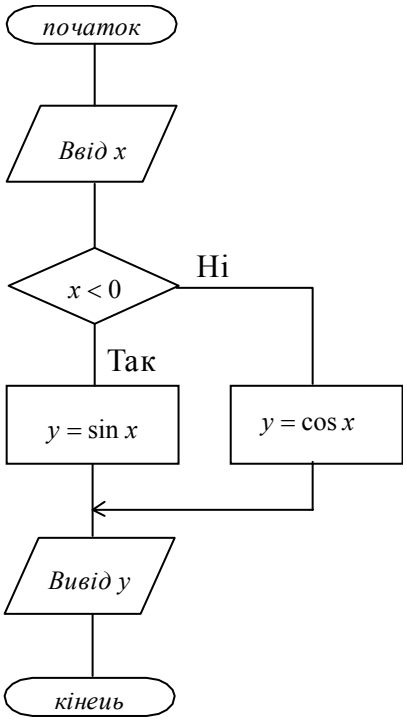


Рис. 3

Особливістю всіх наведених структур є те, що вони мають один вхід і один вихід, тому їх можна поєднувати один з одним у будь-якій послідовності. Зокрема, кожна структура може містити будь-яку іншу структуру в якості одного із блоків (рис. 4).

$$y = \begin{cases} \sin x, & x < 0 \\ \cos x, & x \geq 0 \end{cases}$$

$$S = \sum_{i=1}^{n} a_i$$



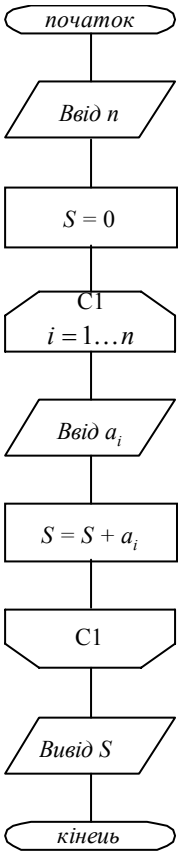


Рис. 4. Поєднання базових алгоритмічних структур у схемах алгоритмів розв'язку задач

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.

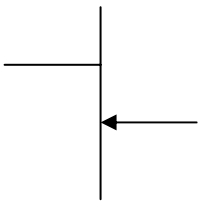
6. ПРАВИЛА ВИКОНАННЯ СХЕМ АЛГОРИТМІВ

6.1. Вимоги до побудови схем

Символи на схемі повинні бути рівномірно розташовані, мати розміри, вибрані в однакових пропорційних співвідношеннях щодо ширини та висоти, достатні для внесення тексту. Якщо обсяг тексту перевищує розмір символа, слід використовувати символ "коментар".

Потоки даних або потоки управління на схемах зображаються лініями. Напрям потоку зліва направо та зверху вниз вважається стандартним і стрілкою не позначається. В іншому випадку на лінії слід обов'язково вказувати стрілку.

Якщо дві чи більше ліній зливаються в одну, то місце з'єднання повинно бути зміщене.



Лінії мають підходити до символа зліва, або зверху, а виходити справа, або знизу. Лінії повинні бути направлені до центра символа.

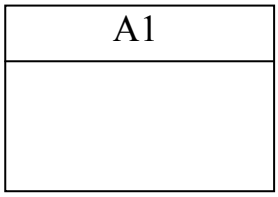
При необхідності розриву лінії (для уникнення зайвих перетинів чи надто довгих ліній) використовують символ "з'єднувач", який у випадку продовження схеми на кількох аркушах подають разом із символом "коментар".

Зовнішній з'єднувач (на аркуші 1) Внутрішній з'єднувач (на аркуші 2)

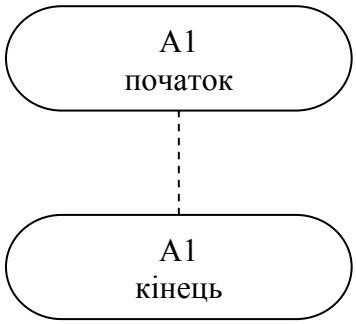
У схемах можна використовувати детальне представлення, яке позначається символом процесу або даних з горизонтальною смужкою у верхній частині.

Між цією лінією та верхньою лінією символа вказується ідентифікатор, який повторюється потім у символі початку та кінця детального представлення у тому ж комплекті документації в іншому місці.

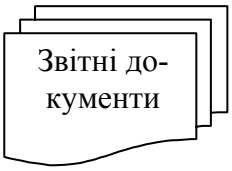
Символ зі смужкою



Детальне представлення



На схемах для зображення використання або формування кількох носіїв даних чи файлів, множини копій друкованих документів тощо замість одного символа з відповідним текстом можуть бути використані кілька символів з перекриттям зображення, які містять відповідний текст, наприклад:



Якщо кілька символів є впорядкованою множиною, то це впорядкування розташовується від першого до останнього.

Пріоритет або послідовний порядок кількох символів не змінюється через точку, в якій лінія входить чи з якої виходить.

6.2. Схема даних

Схеми даних відображають шлях даних при розв’язуванні задач та визначають етапи обробки, а також носії даних, які при цьому використовуються.

Схема даних складається із символів даних, символів процесу, символів ліній та спеціальних символів. Схема даних повинна розпочинатись і закінчуватись символами даних (рис. 5).

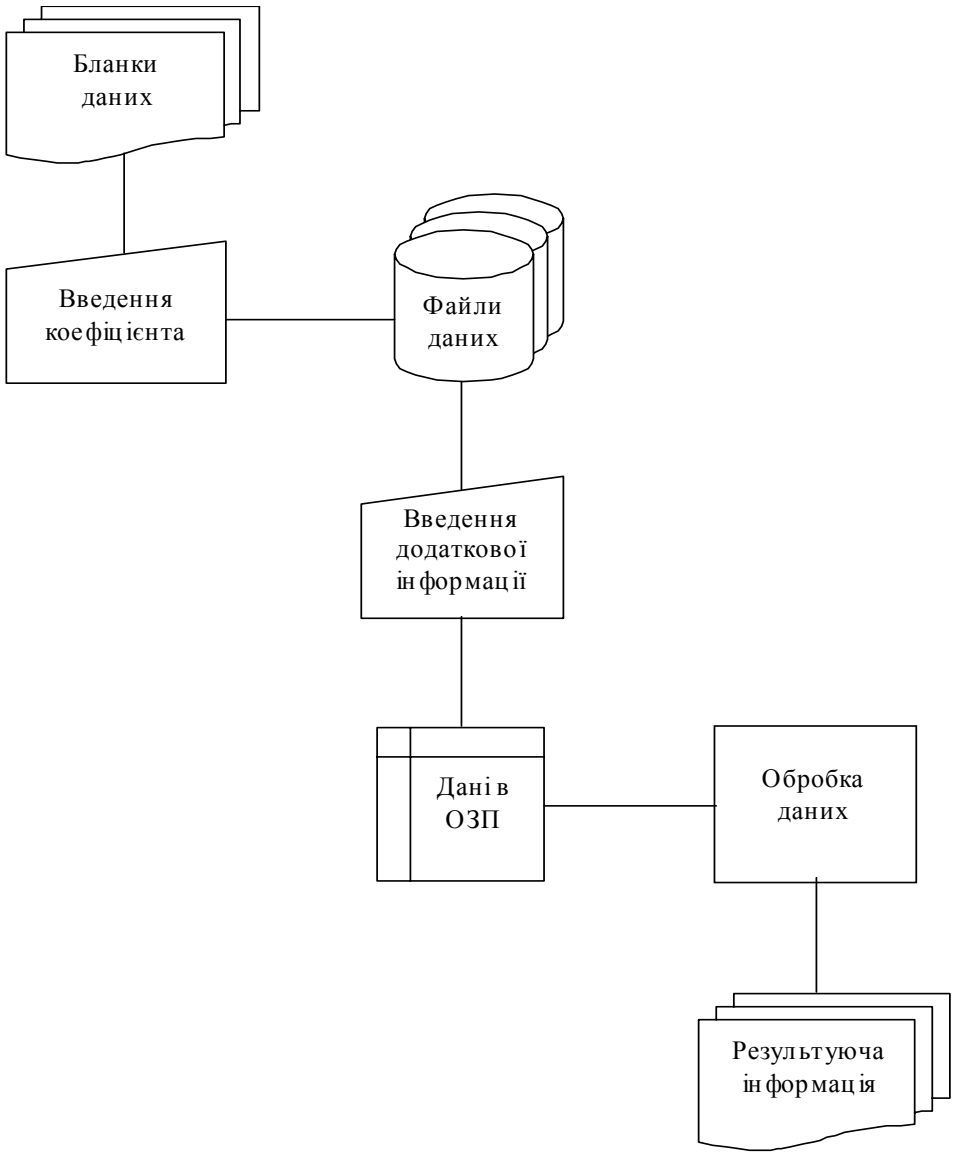


Рис. 5. Схема даних

6.3. Схема програми

Схеми програм відображають послідовність операцій у програмі.

Схема програми складається із символів процесу, символів ліній та спеціальних символів (рис. 6). Із символів даних дозволяється використовувати лише символ “дані” (див. табл.. 1).

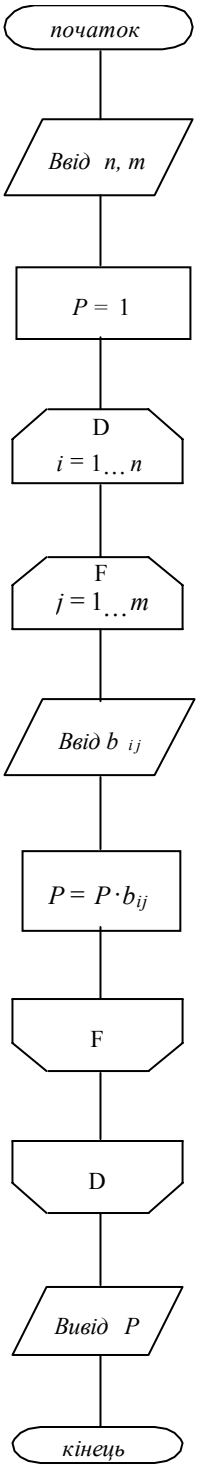


Рис. 6. Схема програми для знаходження добутку елементів матриці

6.4. Схема роботи системи

Схеми роботи системи відображають управління операціями та потік даних у системі. Ці схеми складаються із символів даних, процесу, ліній та спеціальних символів (рис. 7).

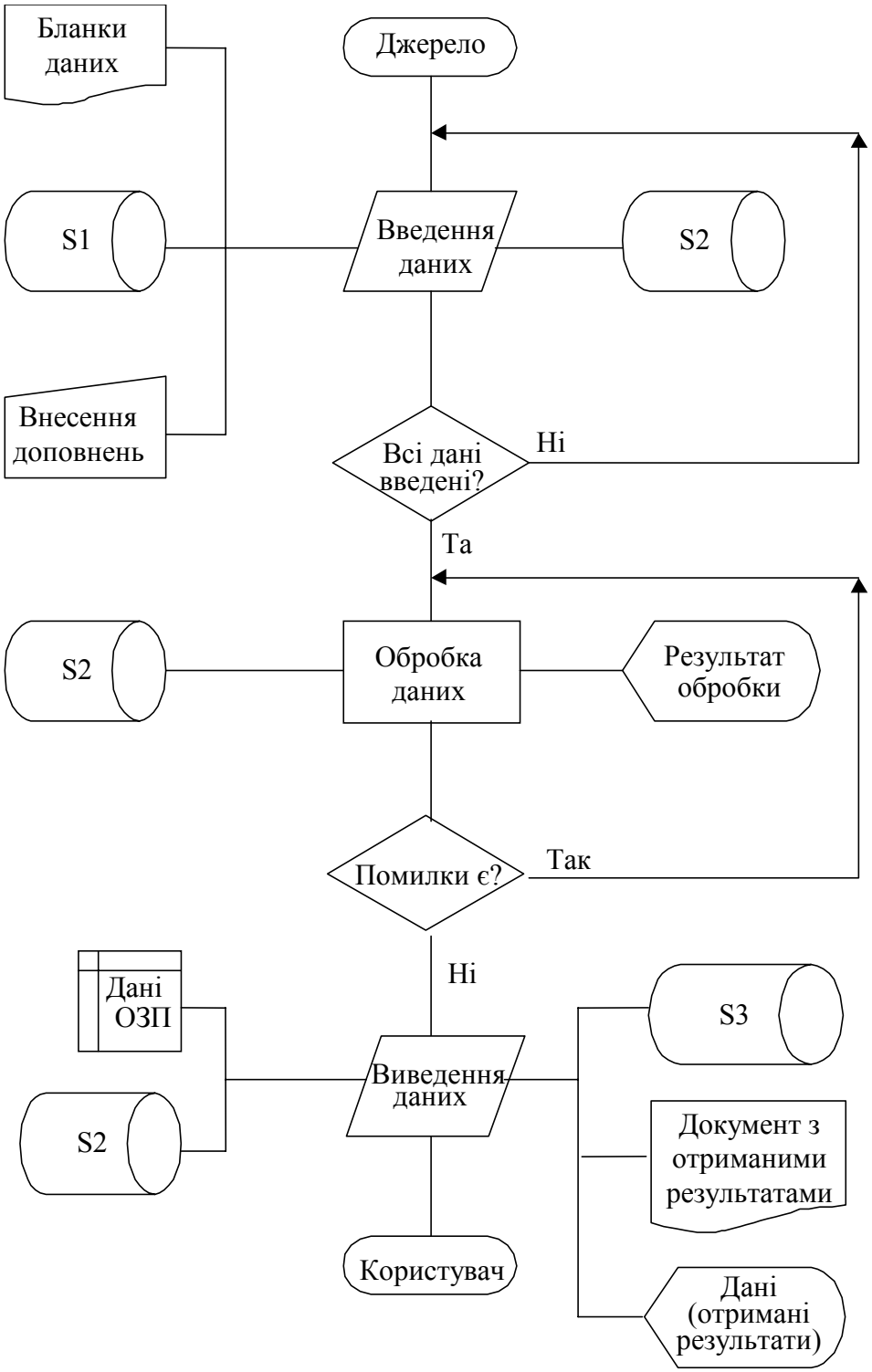


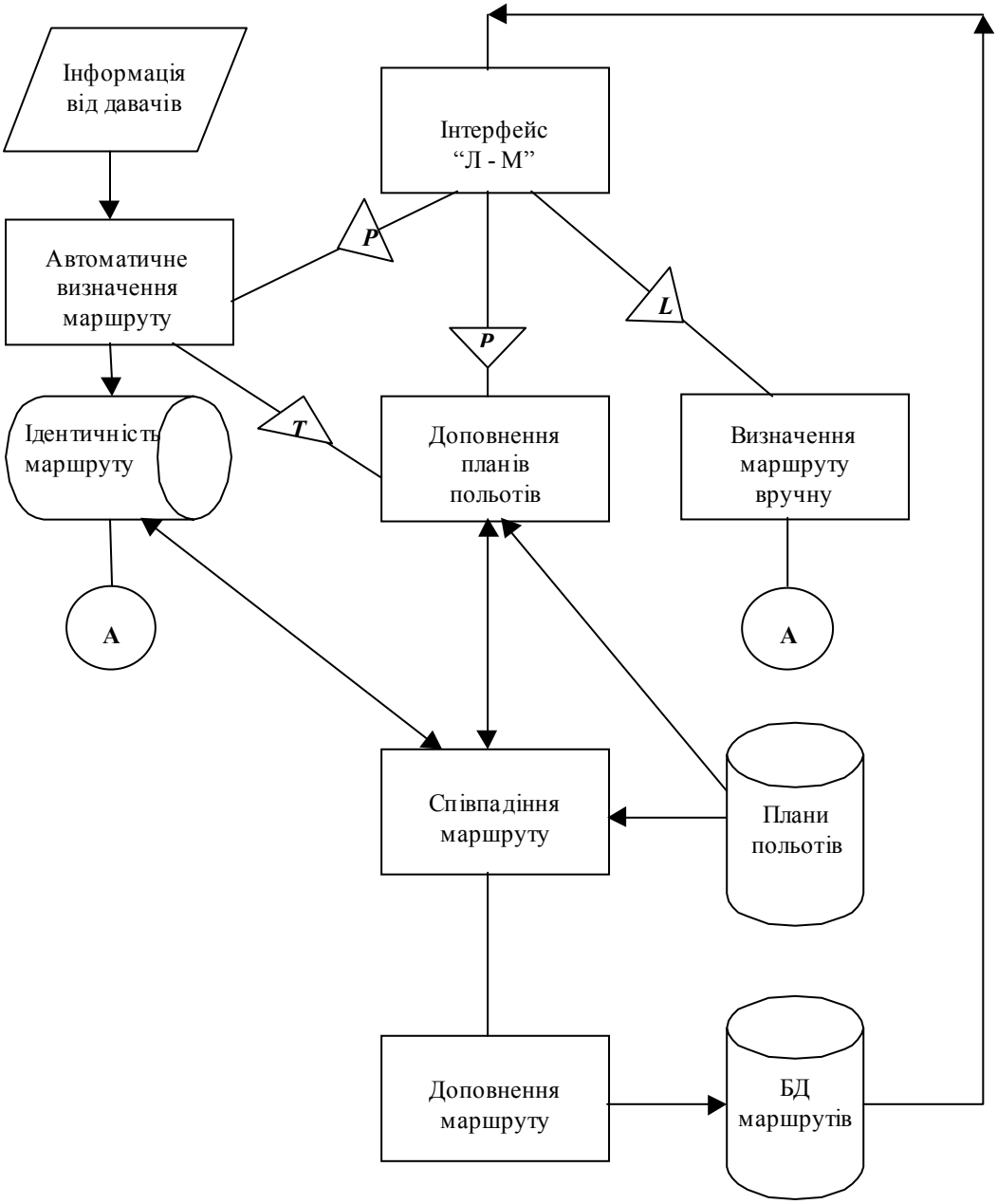
Рис. 7. Схема роботи системи

6.5. Схема взаємодії програм

Схеми взаємодії програм відображають шлях активації програм та взаємодій з відповідними даними (рис. 8).

Кожна програма на схемі взаємодії програм зображається лише один раз символом “процес”.

Лінійні символи на цій схемі можна зображати з нахилом під довільним кутом.



T – тимчасова передача управління
P – передача управління постійна
L – передача управління внаслідок переривання

Рис. 8. Схема взаємодії програм

7. АЛГОРИТМИ СОРТУВАННЯ

Доволі поширеним завданням, що виникає при роботі з масивами та лінійними списками, є завдання сортування, тобто розташування елементів в порядку зростання або зменшення значень. Існує набір алгоритмів сортування. При цьому розрізняють так зване “внутрішнє” та “зовнішнє” сортування, тобто сортування структур даних, що знаходяться в оперативній або в зовнішній пам’яті комп’ютера.

При внутрішньому сортуванні всі дані, що сортуються, поміщають в оперативну пам’ять комп’ютера, де можна отримати доступ до даних в будь-якому порядку (тобто використовується модель пам’яті з довільним доступом). Зовнішнє сортування застосовують тоді, коли об’єм впорядковуваних даних надто великий для розташування в оперативній пам’яті. При цьому вузьким місцем є механізм переміщення великих блоків даних між пристроями зовнішнього зберігання даних та оперативною пам’яттю комп’ютера. Те, що фізично неперервні дані слід для зручності переміщення організовувати у блочну структуру, змушує користувачів використовувати різні методи зовнішнього сортування.

Структура алгоритмів сортування є не дуже складною, їх доволі легко реалізовують будь-якою мовою програмування.

На сучасному етапі розвитку технології програмування існує тенденція використання готових процедур сортування, вмонтованих в інструментальне програмне забезпечення. Користуючись ними, важливо розуміти механізм їх роботи, особливості того чи іншого методу.

Розглянемо деякі поширені алгоритми внутрішнього сортування.

7.1. Сортування простим вибором

На першому кроці алгоритму, починаючи з першого елемента масиву або списку, здійснюється пошук елемента, який має найменше значення, і після цього проводиться обмін місцями знайденого елемента з елементом, який розташований на першому місці масиву або списку (рис. 9, а).

а) перший крок б) другий крок

Рис. 9. Схема алгоритму сортування простим вибором

Потім здійснюється пошук другого найменшого елемента шляхом подальшої перевірки значень елементів, починаючи з другого елемента. Елемент, який має друге найменше значення міняється місцями з елементом, що розташований на другому місці масиву чи списку (рис. 9, б). Цей процес пошуку елемента з наступним найменшим значенням і розташуванням його у відповідну позицію – на початок невідсортованої частини масиву, що залишилась, – продовжується доти, доки всі елементи не стануть відсортованими в порядку збільшення їх значень.

Загальна кількість операцій порівняння в алгоритмі сортування простим вибором дорівнює $n(n-1)/2$, тобто вона пропорційна n^2, а максимальна кількість перестановок – $n(n-1)$ також пропорційна n^2.

7.2. Сортування методом бульбашки (просте обмінне сортування)

Відмінністю цього методу від попереднього є те, що на кожному кроці замість пошуку найменшого елемента масиву з наступною його перестановкою два елементи масиву міняються місцями відразу, якщо тільки між ними порушується порядок: значення *i*-го елемента є більшим за значення (*i*+1)-го елемента. Порівняння поточного $X(i)$ елемента з усіма попередніми ($j = 1,2,\ldots,i-1$) і його перестановка при необхідності на кожному *k*-му кроці відбувається доти, доки цей мінімальний елемент не займе найбільш крайнє ліве (іноді кажуть: крайнє верхнє) положення. Тут і спостерігається аналогія з легкою бульбашкою, що здіймається вгору.

Щоб описати основну ідею цього методу, уявіть, що записи, які слід сортувати, зберігаються у масиві, розташованому вертикально.

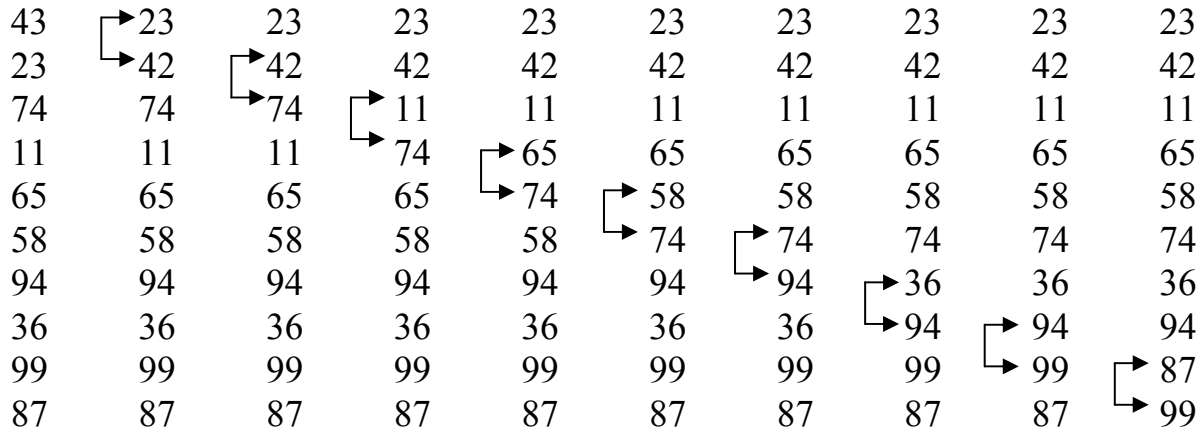
Розглянемо цей алгоритм на числовому прикладі. Нехай задано масив чисел (42, 23, 74, 11, 65, 58, 94, 36, 99, 87). На першому кроці ($k = 1$) елементи масиву будуть переставлятися так, як подано на рисунку 10,а.

Така процедура повторюється на кожному кроці. У результаті цього в кінці кожного кроку масив буде мати вигляд, зображений на рис. 10 б.

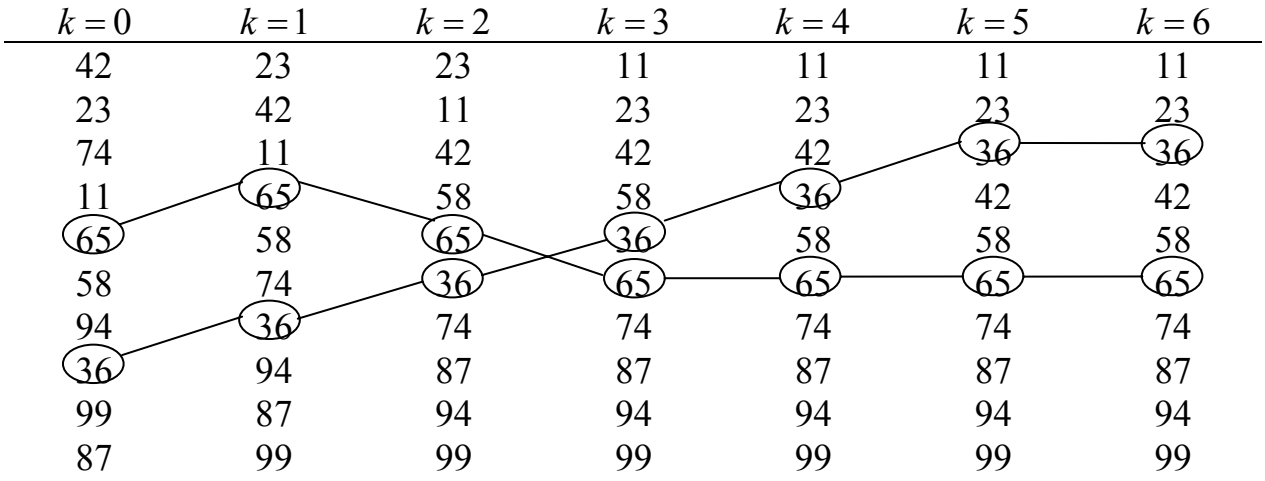
Як бачимо, дійсно, елементи з меншим значенням – «легкі елементи» – піднімаються до верхньої частини масиву, а “важкі” – опускаються вниз. Однак траєкторія переміщення елементів у масиві не завжди одностороння, а тому неоптимальна (прикладом є елемент зі значенням “65”).

Кількість проходів методу бульбашкового сортування може бути змінною. Його загальна ефективність залежить від початкового розташування елементів. Максимальна кількість порівнянь і кількість перестановок пропорційні n^2.

$j=1$	$j=2$	$j=3$	$j=4$	$j=5$	$j=6$	$j=7$	$j=8$	$j=9$	$j=10$



а) перший крок



б) *k*-ий крок

Рис. 10. Схема алгоритму сортування методом бульбашки

Удосконалення методу бульбашки – метод *шейкер-сортування*, при якому перегляд масиву відбувається поперемінно з двох напрямків.

7.3. Сортування вставками

Нехай $1 < j \le N$ та елементи $X_1,\ldots,X_{j-1}$ уже розміщені так, що $X_1 \le X_2 \le \ldots \le X_{j-1}$. Тоді порівнюють по черзі X_j з X_{j-1}, X_{j-2}, поки не виявиться, що елемент X_j потрібно вставити між X_i та X_{i+1}. Для цього зсувають елементи $X_{i+1},\ldots,X_{j-1}$ на одну позицію праворуч і розташовують новий елемент в позицію $i+1$.

Процес “просіювання” може закінчитись при виконанні однієї з умов:

1) знайдено елемент зі значенням, меншим, ніж X_j;

2) досягнуто лівої межі готової послідовності.

Алгоритм з прямими вставками можна вдосконалити, якщо звернути увагу на те, що готова послідовність, у яку потрібно вставити новий елемент, уже впорядкована. Використовується так званий подвійний пошук, при якому робиться спроба порівняння з серединою готової послідовності, а потім процес поділу на дві частини продовжується, поки не буде знайдена точка включення (вставки). Такий модифікований алгоритм сортування називається методом з двійковим включенням. Він не є ефективнішим за пряме включення.

Різновид сортування вставками – сортування Шелла, коли на початку весь масив ділиться, наприклад, на 8 груп по два елементи в кожній: (X_1, X_9); (X_2, X_{10}); ... ; (X_8, X_{16}). Після сортування кожної групи окремо всі елементи знову ділять, але тепер уже на 4 групи по чотири елементи в кожній: (X_1, X_5, X_9, X_{13}); ... ; $(X_4, X_8, X_{12}, X_{16})$, тобто об'єднують деякі групи в одну (X_1, X_9) та (X_5, X_{13}) в (X_1, X_5, X_9, X_{13}) тощо, і продовжують процес. Спосіб розбивки на групи може бути й іншим.

7.4. Швидке сортування

В алгоритмі швидкого сортування метою кожного кроку є розміщення наступного елемента на його кінцеву позицію у середині масиву. Масив завжди ділиться на дві частини, тому бажано вибрати опорний елемент, відносно якого перевпорядковуються елементи масиву, близький до значення медіани розподілу значень. Надалі аналогічний процес застосовується для кожної частини масиву доти, доки всі елементи не займуть свої кінцеві позиції. Розглянемо приклад:

$$X_i = \{42 \quad 23 \quad 74 \quad 11 \quad 65 \quad 58 \quad 94 \quad 36 \quad 99 \quad 87\}.$$

В алгоритмі використовуються два індекси *i* та *j* з початковими значеннями для даного прикладу 1 та 10 відповідно.

Спочатку порівнюються елементи X_i та X_j і, якщо перестановка не потрібна, то *j* зменшується на одиницю й процес повторюється. У разі, коли $X_i > X_j$, елементи X_i та X_j міняються місцями, і після цього процес повторюється, але вже зі збільшенням величини *i*, а не *j*, на одиницю; *j* залишається фіксованим, поки не виникне наступна перестановка. У тому випадку знову *j* зменшиться на 1, а *i* залишиться фіксованим, і так далі (рис. 11).

$k=1$	$k=2$	$k=3$	$k=4$	$k=5$	$k=6$	$k=7$	$k=8$	$k=9$	$k=10$
42	23	74	11	65	58	94	36	99	87
42	23	74	11	65	58	94	36	99	87
42	23	74	11	65	58	94	36	99	87
36	23	74	11	65	58	94	42	99	87
36	23	74	11	65	58	94	42	99	87
36	23	42	11	65	58	94	74	99	87
36	23	42	11	65	58	94	74	99	87
36	23	42	11	65	58	94	74	99	87
36	23	42	11	65	58	94	74	99	87
36	23	11	42	65	58	94	74	99	87

Рис. 11. Схема алгоритму швидкого сортування

У результаті проходження елемент “42” став на свою кінцеву позицію, а початковий масив розбився на два підмасиви: (36, 23, 11) та (65, 58, 94, 74, 99, 87). Цей процес застосовується в подальшому для кожного з них окремо.

8. КРИТЕРІЙ ЕФЕКТИВНОСТІ АЛГОРИТМІВ

Алгоритм для розв’язання будь-якого завдання або проблеми розробити здебільшого неважко. Важко розробити ефективний алгоритм, а ще важче найкращий з можливих.

Найчастіше відмінність між ефективним і неефективним алгоритмами полягає у відсутності чіткого уявлення не лише про саму задачу, а й про відповідні математичні властивості структур, які використовуються для її моделювання.

Нехай A – алгоритм для розв’язування деякого класу задач;

n – розмірність окремої задачі з цього класу (в загальному випадку n може бути масивом або довжиною вхідної послідовності).

Визначимо $f_A(n)$ як робочу функцію, яка дає верхню границю для максимальної кількості основних операцій (додавання, порівняння та ін.), які повинен виконати алгоритм A для розв’язання будь-якої задачі розмірності n.

Використовується наступний критерій для оцінки якості алгоритму:

Алгоритм поліноміальний, якщо $f_A(n)$ росте не швидше, ніж поліном від n, в іншому випадку – алгоритм A експоненціальний.

Інше позначення, стандартне в математиці: функцію $f(n)$ визначають як $O[g(n)]$ і кажуть, що вона порядку $g(n)$ для великих n, якщо:

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

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.

Один зі способів визначення часової ефективності алгоритму полягає в наступному: на основі поданого алгоритму слід написати програму і виміряти час її виконання на певному комп'ютері для вибраної множини вхідних даних. Цей спосіб, хоча й доволі популярний, має певні недоліки. Час виконання залежить не лише від використовуваного алгоритму, а й від архітектури та набору внутрішніх інструкцій певного комп'ютера, від якості компілятора, від кваліфікації програміста, який реалізував алгоритм. Час виконання також може суттєво залежати від вибраної множини тестових вхідних даних. Ця залежність стає цілком очевидною при реалізації одного і того ж алгоритму з використанням різних комп'ютерів, різних компіляторів, програмістів різного рівня і при використанні різних тестових вхідних даних. Для збільшення об'єктивності оцінки алгоритмів приймається асимптотична часова складність як основна міра ефективності виконання алгоритму. Термін "ефективність" є синонімом цієї міри і стосується переважно часу виконання у найгіршому випадку (на відміну від середнього часу виконання).

Можливі ситуації, коли доцільно відійти від часу виконання в найгіршому випадку як міри ефективності алгоритму. Зокрема, якщо відомий очікуваний розподіл вхідних даних, на яких виконується алгоритм, то в цьому випадку середньостатистичний час виконання алгоритму є більш розумною мірою ефективності порівняно з оцінкою часу виконання в найгіршому випадку.

9. АЛГОРИТМІЧНА КУЛЬТУРА

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.

ЛІТЕРАТУРА

1. Овсяк В. Алгоритми: аналіз методів, алгебра впорядкувань, моделі, моделювання. – Львів, 1996. – 132 с.

2. Ахо А., Хопкрофт Д., Ульман Д. Структуры данных и алгоритмы: Пер. с англ. – М.: Изд. дом “Вільямс”, 2001. – 384 с.

3. Гудман С., Хидетниеми С. Введение в разработку и анализ алгоритмов. – М.: Мир, 1981. – 368 с.

4. Кормен Т., Лейзерсон Ч., Ривест Р. Алгоритмы: построение и анализ. – М.: МЦНМО, 2000. – 960 с.

5. ГОСТ 19.701-90. ЕСПД. Схемы алгоритмов, программ, данных и систем. Условные обозначения и правила выполнения. – Введ 01.01.92. – М.: Изд-во стандартов, 1991. – 25 с.

6. Колодницький М. М. Технічне та програмне забезпечення комп’ютерних інформаційних технологій: Навч. посібник. – Житомир: ЖІТІ, 1995. – 231 с.

Редактор *І. В. Галицька*
Технічний редактор *Л. Ф. Щербак*
Верстка *І. І. Барського*

Підписано до друку 18.05.2005 р.
Формат 60x84 $^{1}/_{16}$. Гарнітура Times.
Папір офсетний. Друк на дублікаторі.
Облік.-видавн. арк. 1,6. Умовн. друк. арк. 1,9. Зам. № 1-30.
Тираж 100 прим.

Віддруковано у видавництві ТДЕУ
“Економічна думка”
46000 Тернопіль, вул. Львівська, 11,
тел. (0352) 43-22-18, факс (0352) 43-24-40
E-mail: edition@tane.edu.ua