Лекція 4. СТРУКТУРИ ДАНИХ

Зберігання послідовності значень. Багаторівневе відображення даних. Структури даних: масив, файл, рядок, множина, запис, таблиця, список, стек, черга, дек, дерево.

Зберігання послідовності значень. Обговорення матеріалу нинішньої лекції розпочнемо з прикладу.

Задача. Задано дійсні числа a_1, a_2, ..., a_{100}. Обчислити їхнє середнє арифметичне $\overline{a} = \frac{1}{100}\sum_{i=1}^{100} a_i$ і дисперсію $D = \frac{1}{99}\sum_{i=1}^{100}\left(a_i - \overline{a}\right)^2$.

Здається, нічого складного. Особливо для обчислення середнього: щоб обчислити суму, використаємо цикл з параметром *i*, що набуває значень від 1 до 100; суму, як це вже було раніше, накопичуватимемо у змінній *S*, а змінну *a* використаємо для поступового введення заданих чисел. Сказане легко зобразити блок-схемою (див. рис. 4.1, *а*).

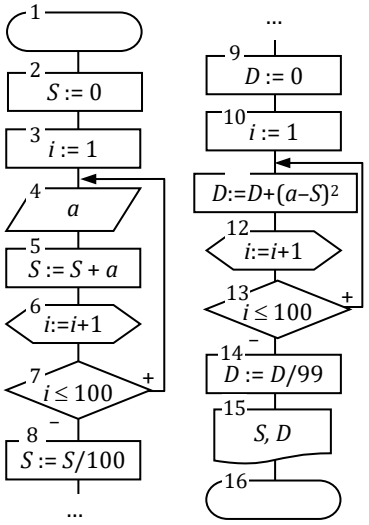

Рис. 4.1. Проект алгоритму

Зверніть увагу на використаний спосіб покрокового введення заданої послідовності чисел: їх по черзі, по одному зчитує блок 4 у змінну *a*. У блоці 5 прочитане число потрапляє до суми, а на наступному кроці циклу на його місце блок 4 зчитує наступне число.

Тепер треба скласти закінчення алгоритму. Усі задані числа введено, значення $\overline{a}$ обчислено, отже, використаємо ще один цикл і змінну *D* для накопичення суми квадратів. Закінчення алгоритму зображено на рис. 4.1, *б*.

Закінчення виглядає досить правдоподібно, але проаналізуймо його уважніше. Не важко переконатися, що маємо проблему в блоці 11: в обчисленнях бере участь те саме значення змінної *a*, адже ніде в циклі воно не змінюється і не оновлюється, а нам треба врахувати всі задані числа. Звідки ця проблема? Річ у тім, що числа *були* введені, використані і витерті, а в змінній *a* залишилось значення тільки останнього.

Як же виправити ситуацію? Можна спробувати перенести обчислення *D* в цикл (3)–(7), адже там є введення *a*, та – знову халепа: значення середнього арифметичного стає відомо тільки після закінчення цього циклу. Може вставити перед блоком 11 блок введення (як перед блоком 5)? Та чи захоче користувач алгоритму знову вводити ті самі 100 чисел, які він уже ввів? Сумнівно. Краще б наш алгоритм не забував введені значення, а якось їх запам'ятовував. Для зберігання значень всякий алгоритм використовує змінні. Наш також, адже зберігає він значення a_{100} у змінній *a*. Щоб запам'ятати решту чисел, нам треба було б ще 99 змінних. Проблему вибору такої великої кількості імен легко вирішити за допомогою цифр: *a*1, *a*2, ..., *a*100. Як автоматизувати перебір таких імен, наприклад, в циклі? На жаль, для компілятора імена *a*1, *a*55, *ai* не мають нічого спільного!

Отож, доходимо висновку, що для ефективного розв'язання задачі нам бракує або знань, або «інструментів». До тепер ми використовували в алгоритмах *прості змінні*. Вони дають змогу зберігати окремі значення. Для зберігання сукупності значень (наприклад, числової послідовності, як у цій задачі) використовують спеціальні *структуровані змінні*, або структури даних.

Структура даних складається з декількох елементів (неподільних значень або простіших структур) і має власне ім'я, єдине для цілої структури. Для неї визначено набір операцій і спосіб доступу до елементів даних.

Найпоширенішою, найбільш уживаною структурою даних є масив – сукупність фіксованої кількості однотипних елементів, кожен з яких має власний номер. Щоб звернутися до елемента, зазначають ім'я масиву і його номер, наприклад, *a*(1), *B*(5), *a*(*i*)

тощо. Масиви зручно опрацьовувати за допомогою арифметичних циклів: кількість елементів масиву відома і незмінна, тому достатньо розташувати в тілі циклу з параметром i опрацювання елемента $a(i)$.

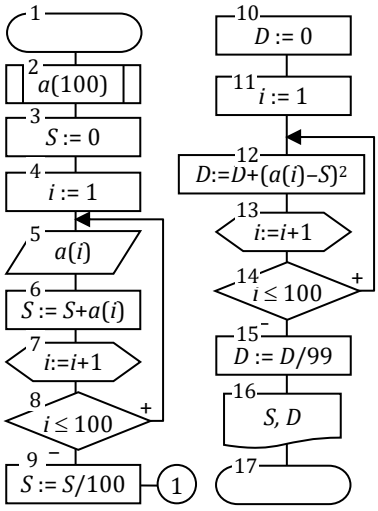

Рис. 4.2. Алгоритм обчислення середнього та дисперсії

Для розв'язання нашої задачі використаємо масив ста елементів: він зберігатиме всі числа заданої послідовності, щоб можна було ввести їх один раз і звертатися до них стільки разів, скільки буде треба.

Остаточний варіант алгоритму зображено на рис. 4.2. Блок 2 – це блок оголошення масиву, або резервування пам'яті для масиву. Блок 5 зчитує вхідні дані в різні елементи масиву. Дані в масиві зберігатимуться аж до закінчення роботи алгоритму: тепер до них можна звертатися і в блоці 6, і в блоці 12 (у разі потреби, і в інших блоках).

Отож, потреба тривалого зберігання сукупності значень та автоматичного опрацювання кожного з них привела нас до ідеї використання нового «інструменту» – масиву. А чи не можна було переформулювати спосіб обчислення D так, щоб він не використовував значення $\bar{a}$?

Відповідь на це запитання пропонуємо знайти читачеві. Детальніше розглянемо різні види структур даних, операції над ними та різні рівні відображення даних.

Багаторівневе відображення даних. Опрацювання за допомогою комп'ютера інформації про реальний світ можливе тоді, коли структура інформації точно визначена і відповідно зображена в обчислювальній машині. Носіями інформації є дані. Вони відтворюють зображення реального світу, взаємовідношення між реальними об'єктами. *Структури даних відображають структурні відношення між елементами даних*, що описують ці об'єкти. Ми вже згадували про відображення реального світу за допомогою математичних та інформаційних моделей. Творець таких моделей намагається зобразити реальний світ шляхом структуризації даних, що описують конкретну предметну область. Об'єктами предметної області можуть бути, наприклад, люди, перелічені в деякій відомості, вироби, які виготовляє підприємство, такі уявні побудови, як рахунки в банку, облікові записи користувачів бібліотеки тощо. Модель повинна точно відобразити відношення між ними. У ході моделювання відбувається абстракція дійсності, оскільки деякі властивості і характеристики реальних об'єктів ігнорують як несуттєві для розв'язуваних задач.

Інформація, яку заносять у комп'ютер, складається з певної множини даних. Довільний набір знаків, який розглядають безвідносно до його змісту, називають *даними*. Дані можна розглядати як сукупність деяких інформаційних одиниць, між якими є певні *відношення*. Ці відношення визначають *структуру даних*. Найменшу, логічно неподільну структуру, називають елементом даних. Структури даних визначають семантику даних, а також способи організації та управління даними. Використану структуру даних реалізують засобами певної мови програмування, а операції зі структурою – операторами мови.

Таблиця 4.1.

Обчислювальні структури

	Статичний аспект	Динамічний аспект
Абстрактний рівень	**Структура інформації** (пов'язана з даними задачі)	**Структура алгоритму** (пов'язана з методом розв'язування)
Конкретний рівень	**Структура пам'яті** (зумовлена можливостями реалізації)	**Структура управління** (реалізована операторами мови програмування)

Кожну обчислювальну задачу можна вивчати на двох рівнях: абстрактному та конкретному. На кожному рівні можна розглядати задачу в двох аспектах: статичному і динамічному. Всюди першочергове значення має структура (табл. 4.1). Всі види структур називають обчислювальними структурами.

Завдяки виділенню таких типів структур Ніклаус Вірт записав «Алгоритми + Структури Даних = Програми».

Розв'язуючи задачі з використанням комп'ютера, ми вирішуємо принаймні чотири взаємопов'язані підзадачі.

1. Добре зрозуміти взаємовідношення між елементами даних, суттєвими для розв'язування задачі.
2. Вибрати потрібні операції над логічно пов'язаними елементами даних.
3. Розробити методи зображення елементів даних у пам'яті машини, що дають змогу: а) якнайповніше зберегти логічні відношення, що є між елементами даних; б) легко та ефективно виконувати операції над елементами даних.
4. Вибрати, які саме засоби (зокрема мова програмування) будуть найпридатнішими для задачі і допоможуть виразити у найзручнішому вигляді ті операції, які треба виконати над даними.

Для розв'язання задачі треба створити програму, пам'ятаючи, що програма – це не мета програмування, а лише засіб для одержання результату. Програми – конкретні втілення абстрактних алгоритмів, побудовані на реальному зображенні даних. Не можна приймати рішення про подання даних, не знаючи, які алгоритми будуть до них застосовуватись. І, навпаки, часто вибір алгоритму залежить від структури даних, до яких його застосовують. Структура програми і структура даних тісно пов'язані між собою.

У процесі проектування програми зображення даних – абстракцій реальних об'єктів – поступово уточнюють і узгоджують з обмеженнями, що накладає система програмування чи можливості реалізації. Абстрактне поняття інформації проходить через декілька рівнів розуміння і перетворюється на конкретне поняття зображення даних. Тому говорять про *багаторівневе відображення даних*. Воно є спільним для різних способів проектування та побудови програм. У табл. 6.2 подано схему, яка відповідає принципові розділення логічного та фізичного зображення даних.

У багаторівневому відображенні інформації про об'єкти реального світу на машинні носії виділено п'ять головних рівнів: *змістовний*, *абстрактний*, *декларативний*, *агрегований*, *базовий*. За своїм цільовим призначенням вони стосуються проблемного (змістовний та абстрактний), процедурного (декларативний) та машинного (агрегований і базовий) середовища існування даних. Кожний з пропонованих рівнів розгляду даних відповідає певному етапу розв'язування задачі (створення програми).

Для побудови моделі ми аналізуємо проблему на змістовому рівні, виділяємо об'єкти та їхні суттєві властивості, відношення між об'єктами й описуємо їх за допомогою *абстрактих структур даних* (АСД). Такі структури існують здебільшого тільки на папері. З їхньою допомогою створюють проект програми. Щоб записати текст програми, треба АСД відобразити засобами конкретної мови, тобто перейти до *декларативних структур даних* (ДСД). Наприклад, масив у мові Паскаль реалізують за допомогою змінної регулярного типу, у мові Бейсік – за допомогою спеціального оголошення *DIM* тощо. Проте ДСД не завжди збігаються з АСД. Наприклад, змінні спискової структури можна оголошувати лише в спеціальних мовах (Лісп, Пролог), а в універсальних мовах (Паскаль) їх моделюють за допомогою змінних інших, базових типів і наборів відповідних процедур. Зрозуміло, що кожна ДСД повинна відображатись у пам'ять ЕОМ, займати її частину. Таке відображення виконує транслятор і створює для кожної ДСД відповідну *структуру зберігання даних* (СЗД). Вона може бути послідовною, зв'язаною або розсіяною. Наприклад, масивові найліпше відповідає послідовне розміщення в пам'яті, спискові – зв'язане, таблиці – розсіяне.

Кваліфікований програміст повинен розуміти різницю між згаданими рівнями відображення, добре володіти абстрактними, декларативними структурами, структурами зберігання.

Таблиця 4.2.

Рівні відображення даних

Рівні розгляду	Етапи роз-в'язування	Носії інформації
Предметна область		**Предмети, відношення, властивості** – об'єкти реального світу, що стосуються конкретної ділянки людської діяльності.
Змістовний рівень	Формулю-вання задачі	**Неформалізовані записи фактів** у термінах категорій предметної області. Їх отримують у результаті інформаційного аналізу предметної області, виділення та опису суттєвих властивостей важливих об'єктів.
Абстрактний рівень	Побудова моделі	**Абстрактні структури даних** – дискретні математичні моделі даних змістовного рівня. Відображають певні способи комбінування елементів у структури, допускають типові операції. При переході до наступного рівня структуру можна зобразити іншою, доступною у вибраній мові.
Декларативний рівень мови програмування	Вибір засобів програму-вання	**Конструкції опису даних** у мовах про-грамування. Програміст використовує певне середовище, тому мусить розробити зображення абстрактної структури та операцій над нею засобами конкретної мови, або вибрати мову, де такі засоби вже є. Структури даних програми оголошують згідно з синтаксичними правилами, тому їх називають *декларативними*.
Агрегований рівень збереження	Створення тексту програми	**Програмно-організовані елементи пам'яті**. Залежно від можливостей використаної системи програмування, дані можуть зберігатись лише в сусідніх одиницях пам'яті (неперервне чи послідовне розміщення), розкиданих по пам'яті і з'єднаних за допомогою адрес-вказівників (зв'язане чи зчеплене), розкиданих по різних одиницях, адреси яких обчислено за спеціальною формулою (розсіяне розміщення). Таку сукупність одиниць пам'яті називають агрегатом пам'яті, а розміщену в ньому структуру даних – *структурою зберігання даних*.
Базовий рівень збереження	Реалізація програми	**Множина адресованих наборів бітів** у пам'яті. Пам'ять машини Нейманівського типу – це майже нескінченна лінійна послідовність двійкових розрядів, розділена з фіксованим кроком на адресовані одиниці: байти, слова, подвійні слова, четвірні слова.
Фізичний рі-вень реалізації		**Фізичні записи на магнітних, оптичних носіях**

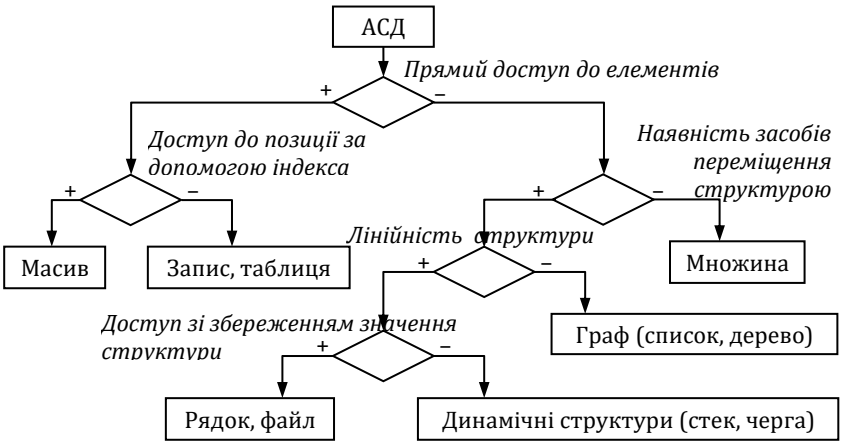

Рис. 4.3. Класифікація абстрактних структур даних

Одну з можливих класифікацій АСД можна виконати за способом доступу до складових елементів структури. Вона показана на рис. 4.3.

Розглянемо тепер детальніше будову різних АСД і можливі операції з ними.

Структури даних масив, файл, рядок. *Масив* – це сукупність фіксованої кількості однотипних елементів, кожен з яких має власний номер – індекс (або послідовність номерів). Індекси елемента однозначно задають його місце в масиві.

Масив є *регулярною* структурою: всі його елементи мають однаковий тип. Він є структурою *прямого доступу*: до кожного елемента можна звернутися безпосередньо, зазначивши ім'я масиву і номер елемента. *Розмірністю* масиву називають кількість його індексів. Одновимірний масив має один індекс, його ще називають вектором. Елементи двовимірного масиву – матриці – пронумеровано двома індексами. Тривимірний масив – куб – має три індекси. Використовують також і багатовимірні масиви.

Кожен індекс змінюється у певному *діапазоні*. Розмір масиву – це загальна кількість елементів у ньому. Вона залежить від кількості вимірів і кількості значень індекса у кожному з вимірів. Розмірність і межі індексів задають у момент оголошення масиву, надалі вони залишаються *незмінними*. Масив називають прямокутним, якщо кожен його індекс змінюється у своєму діапазоні з постійним кроком, причому межі діапазону не залежать від значень інших індексів. Прямокутні масиви вживають найчастіше, саме вони існують в багатьох мовах у явному вигляді. Різні мови надають дещо відмінні можливості щодо оголошення діапазонів зміни індексу. Наприклад, у мові Сі нижньою межею завжди є нуль, у Фортрані – одиниця, у Бейсіку – нуль або одиниця за бажанням програміста, у Паскалі – будь-яке дискретне значення.

Найпоширеніші операції над масивами:

1) локалізація елемента у масиві;
2) пошук елемента за індексом (чи сукупністю індексів) і зчитування його значення;
3) зміна значення елемента в масиві (запис значення);
4) пошук елемента за значенням;
5) копіювання масиву;
6) впорядкування масиву;
7) введення-виведення масиву.

Локалізація елемента – це обчислення адреси конкретного елемента. Вона є основою відображення масиву в послідовну СЗД і відбувається під час кожного звертання до будь-якого елемента масиву. Найпростіше локалізувати елемент вектора, складніше це зробити для матриці. Наведемо декілька прикладів, вважатимемо, що нижньою межею індексу завжди буде одиниця.

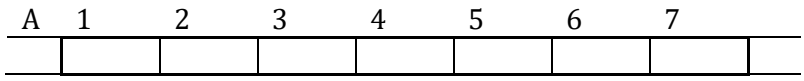

Рис. 4.4. Пам’ять одновимірного масиву

Кожен масив займає неперервну ділянку лінійної пам’яті. Вектор з семи елементів $A(7)$ схематично зображено на рис. 4.4.

Позначатимемо за допомогою літери «@» адресу змінної. Тоді $@A$ – адреса вектора, вона збігається з адресою першого елемента масиву. Адресу i-го елемента вектора обчислюють за формулою

$$@A(i) = @A + (i - 1) \times d,$$

де d – довжина елемента масиву.

Матриця є двовимірним об’єктом. Як відобразити її в одновимірну пам’ять? Наприклад, масив $B(2,3)$ зберігає значення матриці $\begin{pmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \end{pmatrix}$.

Можна сказати, що він складається з двох рядків по три елементи в кожному, або з трьох стовпців по два елементи. Обидва трактування правильні, тому відобразити матрицю в пам’ять можна двома способами: за рядками або за стовпцями.

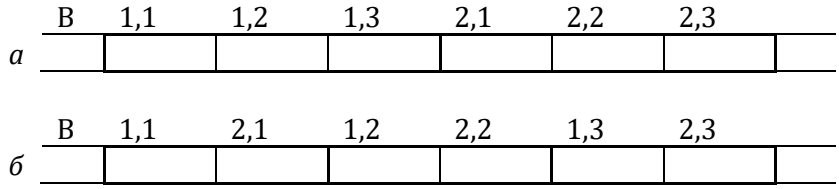

Рис. 4.5. Відображення матриці в пам’ять:
а – за рядками; *б* – за стовпцями

На рис. 4.5, *а* проілюстровано відображення за рядками. Його задає формула

$$@A(i,j) = @A + [(i - 1) \times m + (j - 1)] \times d,$$

де m – кількість елементів у рядку. Таке відображення виконує, наприклад, компілятор мови C++.

Відображення за стовпцями (рис. 4.5, *б*) задає формула

$$@A(i,j) = @A + [(j - 1) \times n + (i - 1)] \times d,$$

де n – кількість елементів у стовпці. За стовпцями відображає матрицю в пам’ять компілятор мови Фортран.

Для відображення багатовимірних масивів описані способи відрізняються тим, який з індексів – останній чи перший – змінюється швидше.

У програмах на Асемблері відображення не фіксовано: програміст резервує пам’ять потрібного розміру і сам визначає, як його трактувати – або як одновимірний масив, або багатовимірний (зі швидшою зміною першого або останнього індексу). Далі – відповідно програмує локалізацію елементів.

Для виконання операцій введення, виведення, перебору елементів масиву найзручніше застосовувати арифметичний цикл, причому параметр циклу може задавати значення індекса масиву.

Виявляється, доступ до елемента масиву не така вже й проста річ, адже для локалізації, наприклад, елемента матриці треба виконати два множення і чотири додавання. Це додаткові обчислювальні затрати, тому масиви використовують тоді, коли це справді потрібно. (Поміркуйте, яку перевагу надає те обмеження мови Сі, що найменшим значенням індексу масиву завжди є нуль?)

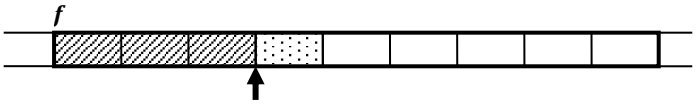

Рис. 4.6. Схематичне зображення файла

Файл – це послідовність (можливо, змінної довжини) однотипних елементів, записана на деякому носії. Елементи файла ще називають *записами*. З кожним файлом пов'язано спеціальну змінну, *вказівник файла*. Вона позначає (один!) доступний для опрацювання запис: зазначає його початок. Кожна операція доступу до файла автоматично пересуває його вказівник на наступний запис. Отож, файл є структурою *послідовного доступу*.

Найпоширеніші операції над файлом:

1) створення;
2) встановлення у початковий стан;
3) зчитування елемента файла;
4) запис значення до файла;
5) перебір усіх записів файла.

Схематично файл зображено на рис. 4.6. Штрихуванням позначено опрацьовані елементи, цятками – доступний до опрацювання елемент, вказівник файла позначає його (елемента) початок.

Довжина файла (кількість записів) може змінюватися. Після опрацювання останнього елемента вказівник файла автоматично зміщується на довжину цього елемента і позначає кінець файла. Якщо в такій ситуації виконати операцію запису до файла, то у ньому з'явиться новий (вже опрацьований) елемент, а вказівник файла знову встановиться на його кінець. Усі наступні операції запису додаватимуть до файла нові елементи. Довжина файла може і зменшитися, якщо виконати спеціальну операцію втинання, у результаті якої з файла вилучають всі записи, починаючи з позначеного вказівником і аж до останнього. Після втинання вказівник позначає кінець файла. Зрозуміло, що так легко вилучити з файла останній запис (чи декілька прикінцевих). Вилучити перший чи будь-який внутрішній запис можна тільки за допомогою «стиску» – переміщення наступних за ним записів.

Для перебору елементів файла використовують ітераційні цикли, умовою закінчення яких є досягнення вказівником файла його закінчення.

Файл можна порівняти з одновимірним масивом: їхньою спільною рисою є однотипність елементів, а різниця полягає у різному способі доступу до елементів, у можливості зміни довжини файла.

Рядок – це скінченна послідовність літер деякого алфавіту. У теорії алгоритмів абстрактним алфавітом є скінченна непорожня множина довільних символів, наприклад, латинський алфавіт, двійковий алфавіт {0, 1}, множина нотних знаків, ієрогліфів тощо. ЕОМ використовує стандартизований алфавіт – множину символів, визначену таблицею кодів.

Сучасні мови програмування володіють засобами обробки рядків, які реалізують основні операції над рядками:

1) створення;
2) конкатенація двох рядків для одержання нового (конкатенація – «склеювання», дописування літер другого рядка одразу після останньої літери першого);
3) підстановка (пошук у рядку заданого підрядка і заміна його іншим);
4) перевірка двох рядків на тотожність;
5) визначення довжини рядка.

Розрізняють три види рядків: фіксованої, змінної і необмеженої довжини. Для відображення рядка у СЗД зазвичай вибирають вектори і ланцюгові списки.

Рядок фіксованої довжини відображають у вектор – одновимірний масив літер, розмір такого вектора дорівнює довжині рядка.

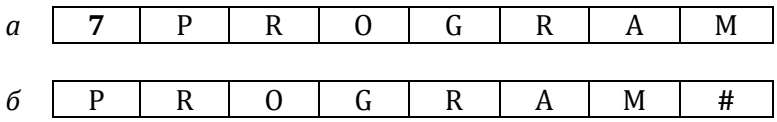

Рис. 4.7. Відображення рядка змінної довжини в пам'ять:
а – з дескриптором; *б* – з граничним маркером

Рядок змінної довжини відображають у вектор за допомогою дескриптора – першого елемента такого вектора, що описує довжину рядка (рис. 4.7, *а*), або за допомогою граничного маркера – символа, що не входить до алфавіту рядка. Такий маркер позначає закінчення рядка (рис. 4.7, *б*). Гнучкішим є відображення рядків у ланцюгові списки. Його використовують і для рядків необмеженої довжини.

Рядки фіксованої довжини опрацьовують як масиви – за допомогою арифметичних циклів, а змінної довжини – за допомогою ітераційних.

Структури даних множина, запис, таблиця. *Множина* – це невпорядкована сукупність деяких об'єктів, які називають елементами множини. Усі елементи множини різні, над ними не визначено жодного відношення.

Головні операції над множинами:

1) перевірка належності елемента множині (результатом є істина або хиба);
2) додавання елемента до множини (множина залишиться незмінною, якщо вона вже містить такий елемент);
3) вилучення елемента з множини;
4) об'єднання, перетин, різниця двох множин.

У програмах множини використовують для зберігання даних тоді, коли порядок цих даних є несуттєвим, і кожне значення може траплятися не більше, ніж один раз.

Для відображення множини в пам'ять ЕОМ використовують один з двох можливих способів. Перший описує кожен елемент множини і розташовує такий опис у векторі або лінійному (ланцюговому) списку. Другий визначає всі потенційно можливі елементи множини, а потім для всякої підмножини такої універсальної множини зазначає для кожного можливого елемента, чи належить він цій підмножині, чи ні. СЗД у цьому випадку – бітовий вектор.

Запис (від англійського record, не плутати з записом – елементом файла), або структура – це сукупність фіксованої кількості різнотипних елементів, які називають полями. Кожне поле має власне унікальне ім'я. Доступ до поля – за комбінацією імен структури та самого поля.

Записи використовують для відображення інформації про властивості реальних об'єктів. Наприклад, дані про деякого студента могли б міститися у структурі з полями для повного імені, дати народження, адреси проживання, оцінок за іспити, розміру стипендії тощо. Дату й адресу також зручно зображати структурами.

Головні операції над структурами: читання і запис значень полів, копіювання. Перебір полів структури виконують вручну, зазначивши їхні імена.

	Ключ	Поле1	Поле2	...	ПолеN
Шапка					
	1				
Записи	2				
	...				
	M				

Рис. 4.8. Схематичне зображення таблиці

Таблиця – це набір елементів, з кожним з яких пов'язано унікальну ознаку, ключ. Ключ визначає позицію елемента в таблиці і забезпечує прямий доступ до нього. Усі елементи

таблиці є парами вигляду {Ключ (аргумент), Тіло (значення)}. Вони утворюють *рядки* таблиці. Тілом рядка найчастіше є структура (запис). У всіх рядках набори полів структури однакові. Однойменні поля утворюють *стовпці* таблиці. Описи полів розташовано у спеціальному рядку – шапці таблиці (рис. 4.8).

Ключем елемента таблиці може бути просте значення (наприклад, номер, ім'я чи ідентифікатор) або сукупність кількох атрибутів. Кожен ключ задовольняє такі вимоги: 1) однозначно ідентифікує елемент; 2) не має надлишковості – ніякий атрибут не можна вилучити з ключа, щоб не порушити однозначності ідентифікації. Для одного елемента може знайтися декілька ключів. Один з них (здебільшого найпростіший) вибирають первинним ключем і використовують для зазначення елемента.

Основна операція в таблиці – пошук. Вона полягає в тому, щоб за заданим ключем визначити адресу зберігання запису, якщо він є. Важливо вибрати таку організацію таблиці, яка допускала б ефективний пошук.

Таблиці з прямим доступом використовують ключем адресу елемента, або обчислюють цю адресу як взаємно однозначну функцію від ключа. Таблиці з обчислюваними входами для визначення адреси елемента за ключем використовують функції розстановки (перемішування) або хеш-функції.

Структури даних список, стек, черга, дек, дерево. *Список* – це скінчена впорядкована послідовність пов'язаних між собою об'єктів довільної природи та розміру. Ці об'єкти прийнято називати ланками. Кількість ланок може змінюватися. Впорядкованість ланок залежить від призначення списку, і нею може керувати програміст.

Поняття «список» природно узагальнює звичні для нашого повсякденного життя структури, наприклад, список студентів академічної групи, список абонентів телефонної мережі тощо. За допомогою списків можна імітувати схему залізничних сполучень, роботу станції технічного обслуговування та ін.

Ланку списку умовно можна поділити на дві частини: тіло (власне елемент списку); довідкова інформація про розмір і тип конкретної ланки, взаємозв'язок з іншими ланками тощо.

Зі списком можна виконувати такі основні операції:

1) створення (порожнього) списку;
2) перехід до сусідньої ланки, перебір цілого списку;
3) локалізація елемента (встановлення вказівника на елемент);
4) включення у довільному місці нової ланки (підсписку);
5) вилучення зі списку довільної ланки (підсписку);
6) зміна порядку ланок у списку.

Зазначимо, що перелічені вище дії не вимагають фізичного переміщення ланок, а лише передбачають зміну довідкової частини окремих ланок. Список дуже гнучка структура, оскільки її можна збільшувати чи зменшувати за рахунок додавання чи вилучення нових елементів у довільній позиції списку. Списки можна об'єднувати або розбивати на менші.

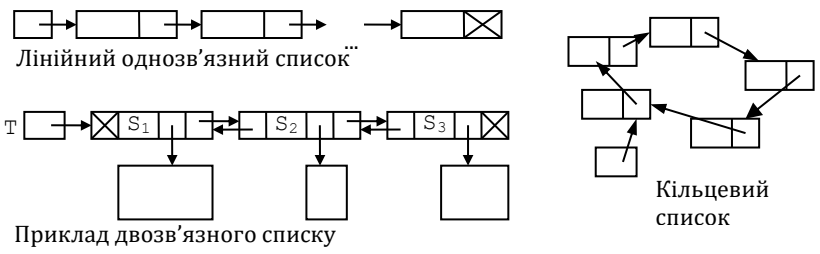

Рис. 4.9. Схематичне зображення списків

Порядок елементів визначено не індексами (номерами), як у масиві, а вказівниками. Якщо елемент не має попередника, то це голова списку, а інші ланки – його хвіст.

Залежно від способу пов'язування ланок розрізняють типи списків. Найрозповсюдженішими з них є такі:

- лінійний однозв'язний список;
- лінійний двозв'язний список;
- кільцевий, як частковий випадок лінійного;
- ієрархічний;
- асоціативний.

У лінійному списку визначено відношення сусідства ланок, а в списках інших типів – ні. Списки відображають у послідовні або спискові СЗД.

Є три динамічні структури – стек, черга та дек. Особливістю всіх динамічних структур є те, що елементи перебувають у них до першого звертання. Додавання і вилучення елементів відбувається не в довільному місці структури, а в місці, визначеному саме структурою даних. Будь-яке звертання до структури змінює її вміст.

Стек (магазин) – це впорядкована лінійна динамічно змінювана послідовність елементів, для якої виконуються такі умови:

1) новий елемент приєднується завжди до того самого краю послідовності;
2) доступ до елементів відбувається завжди з того краю послідовності, до якого елементи приєднуються;
3) елемент зберігається в послідовності до моменту звертання до нього.

Дані заносять у стек і вилучають лише з одного кінця – *вершини стека*. Стек працює за принципом «останнім прийшов – першим пішов». Тому його називають послідовністю типу LIFO (last in firs out). Деякий пристрій за цим принципом працюватиме так: замовлення, яке надійшло першим, заносять у стек і одразу ж починають виконувати. Якщо під час виконання чергового замовлення надходить наступне, то робота механізму переривається, нове замовлення заносять у стек і механізм розпочинає роботу з цим новим замовленням. Попереднє ж залишається в стеку, очікуючи завершення опрацювання нового замовлення. Як тільки опрацювання замовлення завершується, воно вилучається зі стека, і механізм повертається до обслуговування попереднього. Залежно від часу надходження замовлень і потрібної тривалості їхньої обробки стек може видовжуватись, скорочуватись, залишатись деякий час порожнім.

Друга назва цієї структури даних – магазин – від аналогії зі способом функціонування магазину стрілецької зброї: набій, який вклали до нього останнім, вистрілює першим.

Практична потреба в організації стека виникає, наприклад, під час перетворення транслятором арифметичних виразів програми з інфіксної форми запису до постфіксної (до так званого польського запису), простежування послідовностей викликів процедур тощо.

Розрізняють стеки з проштовхуванням (push down) та без проштовхування. У стеках першого типу закріплена вершина, і в момент занесення нового елемента всі дані в стеку опускаються на одну позицію нижче, щоб звільнити місце для нового значення (справді, як набої у магазині автомата). У стеках без проштовхування закріплене дно. У випадку занесення елемента вершина (вказівник вершини) пересувається на вільне місце у стеку, а у випадку звертання до елемента – назад до останнього занесеного значення.

Над стеком визначено дві операції: включення (занесення) елемента в стек і зчитування (вилучення) елемента зі стека. Якщо стек реалізувати за допомогою однонапрямленого списку, то над ним можна виконувати операції додавання нової ланки лише на початок списку і вилучення лише першої ланки списку. Стек можна відобразити також у послідовну СЗД і реалізувати його за допомогою вектора з індексом вільного елемента як рухомої вершини.

Черга – це впорядкована лінійна динамічно змінювана послідовність елементів, для якої виконуються такі умови:

1) новий елемент приєднується завжди до того самого краю послідовності;

2) доступ до елементів, тобто їх вилучення, відбувається завжди з іншого краю послідовності.

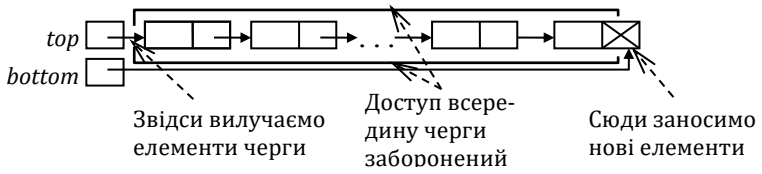

Рис. 4.10. Схематичне зображення черги

Місце вилучення називають *головою* черги, а місце включення – *хвостом*. У структурі черги потрібні два вказівники: на голову послідовності для вилучення елементів і на хвіст – для занесення (рис. 4.10). При читанні з черги завжди вилучають «найстаріший» елемент. Говорять, що черга працює за принципом «першим прийшов, першим пішов», тобто є послідовністю типу FIFO (first in first out). Всередині черги елементи рухаються від хвоста до голови.

Черги широко застосовують в імітаційних алгоритмах і моделюванні складних систем обслуговування, в операційних системах. Крім класичних черг, які ще називають *лінійними*, використовують черги *з пріоритетом* та *циклічні*. Черги з пріоритетом дають змогу вилучати будь-який елемент залежно від їхніх додаткових характеристик. Наприклад, в операційних системах окремі задачі можуть мати вищий пріоритет і після надходження до загальної черги завдань позачергово потрапляти на виконання. У циклічній черзі перший елемент розташовується одразу за останнім. Якщо операційна система обробляє задачі однакового пріоритету в режимі розділення часу, то всіх їх вона поміщає в кільцеву чергу і виділяє їм кванти часу.

Черги відображають у спискові структури або вектори.

Дек схожий до обох попередніх АСД – це також впорядкована послідовність елементів, але додавати і вилучати елементи можна з обох її кінців.

Найважливішими нелінійними структурами, які використовують в обчислювальних процесах, є дерева. Вони найліпше пристосовані для розв'язування задач штучного інтелекту (побудова таблиць рішень), синтаксичного аналізу (побудова дерева виведення, розпізнавання ієрархічних структур), аналітичних перетворень (зображення виразів), організації баз даних та інших задач.

Деревом (деревовидною структурою) називають множину взаємопов'язаних об'єктів (які називають також *вершинами* або вузлами), розташованих за рівнями за такими правилами:

1) на першому рівні – один вузол, *корінь* дерева;
2) будь-який вузол X наступного, i-го ($i \neq 0$) рівня пов'язаний лише з одним вузлом Y попереднього, (i – 1)-го рівня.

У такому випадку Y називають безпосереднім предком вузла X, а X – безпосереднім нащадком Y. Якщо вузол не має нащадків, то він називається *листком*, або термінальним елементом. Всі вузли, крім кореневого, які мають нащадків, називаються внутрішніми вузлами, або *вершинами*.

Усі вузли дерева, пов'язані з вершиною X безпосередньо або через інші вузли, називаються її нащадками. Довільна вершина X дерева разом з усіма своїми нащадками називається *піддеревом* цього дерева.

Дерево можна визначити також рекурсивно – це скінченна множина T, що складається з одного чи більше вузлів таких, що:

1) є один спеціальний вузол, який називають коренем цього дерева;
2) решта вузлів входять до складу пов'язаних з коренем $m \geq 0$ множин T_1, ..., T_m, які попарно не перетинаються і кожна з яких сама є деревом (T_1, ..., T_m називають піддеревами).

Ізольована вершина утворює окреме дерево.

Дерева застосовують для зображення об’єктів, які мають ієрархічну внутрішню будову. Наприклад, адміністративну структуру університету можна зобразити у вигляді дерева, коренем якого є ректор, а листками – зокрема, викладачі та студенти; математичну формулу – деревом, листками якого є операнди, а вершинами – знаки операцій тощо.

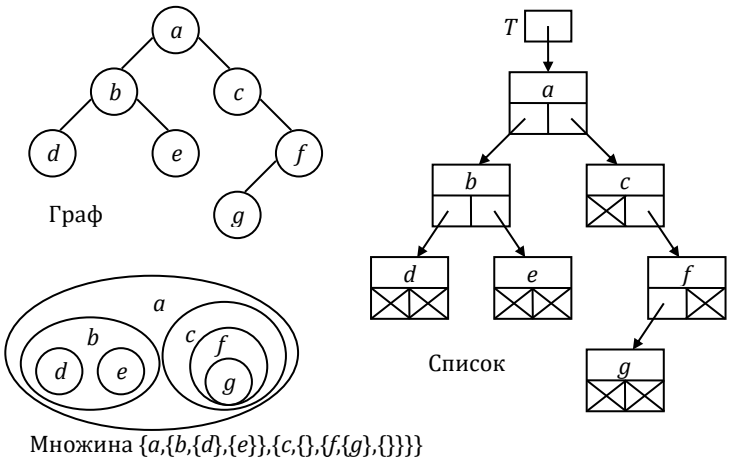

Рис. 4.11. Різні способи зображення дерева

Кожна вершина дерева складається з полів зв’язку з нащадками та інформативної частини, яка залежно від потреби містить інформацію довільного типу. Інформативну частину вершини дерева називають елементом дерева.

Є різні способи зображення дерев: за допомогою словесного опису, у вигляді графа (див. рис. 4.11), вкладених множин тощо. Дерево можна інтерпретувати як список, ланки якого можуть мати по декілька наступників. Розглянуті вище однонапрямлені списки є, по суті, «виродженими» деревами, вершини якого мають тільки одного наступника.

Графічно зв’язок між вершинами зображають за допомогою ліній – ребер. Кількість ребер між вузлами називають довжиною шляху між цими вузлами. Довжина шляху від кореня до деякого вузла називається рівнем цього вузла. Рівень кореня – нуль. Найбільший рівень дерева називається його *висотою*. Внутрішнім шляхом дерева називають суму довжин шляхів від кореня до кожного вузла.

Якщо з дерева вилучити корінь і ребра, що зв’язують його з вершинами першого рівня, то отримаємо множину незв’язаних дерев, яку називають *лісом*. Оскільки кожен вузол визначає піддерево, то можна вважати, що піддерева, розташовані під вузлом, утворюють ліс.

Максимальна кількість безпосередніх нащадків того самого предка називається *степенем дерева*. Наприклад, на рис. 4.11 зображено дерево степеня два. Такі дерева називають *двійковими* (бінарними) деревами. Дерево називається впорядкованим, якщо має значення відносний порядок його піддерев. Якщо на i-му рівні дерева є n вузлів, то можна утворити $n!$ різних впорядкованих дерев, перестановкою значень (елементів) цих вузлів.

Вузли дерева можуть містити спеціальне поле для їхнього імені – так званого *ключа*. Ключами можуть бути довільні коди (числові, літерні), що допускають порівняння. Всі ключі в дереві повинні бути різними. Вони слугують для ідентифікації вузлів дерева. Завдяки використанню ключів алгоритми обробки деревовидних структур стають простішими, адже опрацьовують не самі елементи, а їхні ключі.

Окремий важливий клас становлять двійково-пошукові дерева (дерева пошуку). Їх визначають так: з кожного вузла виходить не більше двох гілок, і кожну гілку ідентифікують як ліву або як праву.

У дереві пошуку для кожної вершини правильним є твердження про те, що ліве її піддерево містить лише вершини з меншими ключами, а праве – з більшими. Дерева пошуку зручно використовувати для організації швидкого доступу до потрібних частин великих масивів даних. Для цього даним приписують певні ключі і розташовують їх у вузлах відповідного дерева. Потрібного ключа можна знайти, скориставшись алгоритмом

бінарного пошуку. Час його виконання – величина порядку $O(\log_2 n)$, де n – кількість ключів у дереві.

Найефективніші для пошуку – ідеально збалансовані дерева. Дерево пошуку називається *ідеально збалансованим*, якщо кількість вершин у всіх його відповідних лівих і правих піддеревах відрізняється не більше, ніж на одиницю. Ідеальне збалансування дерева пошуку виконати складно і дорого, тому частіше використовують просте збалансування. Дерево називається *збалансованим* тоді і лише тоді, коли висоти двох піддерев кожної з його вершин відрізняються не більше, ніж на одиницю. Ця вимога є суттєво слабшою порівняно з умовами ідеальної збалансованості. Дерево, яке має однакову *кількість* вершин у піддеревах, має піддерева однакової *висоти*, але не навпаки. Збалансовані дерева називають ще АВЛ-деревами за іменами їхніх винахідників Г. М. Адельсон-Вельського та Є. М. Ландіса (1962). Для АВЛ-дерев час пошуку теж є величиною порядку $O(\log_2 n)$, а довжина АВЛ-дерева в найгіршому випадку не перевищує 1,45 довжини ідеально збалансованого дерева.

Над деревом виконують такі головні операції:

- обхід дерева;
- вставка нової вершини (піддерева);
- вилучення вершини (піддерева).

Найчастіше виконують обхід дерева. Під час обходу алгоритм повинен опрацювати всі вершини дерева, кожну – один раз. Дерево обходять, щоб відшукати певний елемент, роздрукувати всі елементи, перетворити їх тощо.

Найпоширенішими способами обходу дерева є *прямий* (нисхідний, preorder), *зворотний* (змішаний, inorder), *симетричний* (висхідний, postorder), порівневий. Кожен з них має два різновиди: лівостороннній і правостороннній. Під час лівостороннього обходу першим опрацьовують крайній лівий листок, а останнім – крайній правий. Прямий обхід задає такий порядок: опрацювання кореня, прямий обхід лівого піддерева, прямий обхід правого піддерева. Зворотний обхід: зворотний обхід лівого піддерева, опрацювання кореня, зворотний обхід правого піддерева. Симетричний обхід: симетричний обхід лівого піддерева, симетричний обхід правого піддерева, опрацювання кореня.

Деревовидні структури відображають у послідовні, спискові або розсіяні СЗД. Під час відображення у послідовні структури враховують той факт, чи містить дерево інформацію (навантажене дерево), чи тільки описує підпорядкованість, ієрархію вузлів (топологічне дерево). Відображенням у рядок, зображеного на рисунку навантаженого дерева, є запис {a,{b,{d},{e}},{c,{},{f,{g},{}}}}. Дужки позначають піддерева. Для відображення топологічних дерев використовують коди, що ґрунтуються на обходах дерева. Наприклад, нехай 1 позначає спуск, а 0 – підйом, тоді результатом прямого обходу того ж дерева (без врахування літер) є код 110100111000.