

Структури даних: основні поняття

Укладач: Дмитро Попов.

<https://www.linkedin.com/in/dima1popov>

Структури даних є невід'ємною частиною інформатики і програмування. Вони описують способи організації та зберігання даних так, щоб їх було зручно обробляти та маніпулювати ними. Для того щоб краще розуміти, як працюють різні структури даних, важливо зрозуміти різницю між абстрактними структурами даних та конкретними їх реалізаціями.

Абстрактні структури даних vs. Структури даних

Абстрактна структура даних (АСД) — це модель, яка описує набір операцій над даними без зазначення конкретної реалізації цих операцій. Вона фокусується на тому, що потрібно робити з даними (набори операцій), а не на тому, як це реалізувати.

Наприклад, черга (queue) або стек (stack, стопка) можуть бути описані як абстрактні структури даних. Вони описують операції на елементах, такі як додавання чи видалення елементів, але не визначають, яким саме чином ці операції будуть виконуватися на практиці.

Наприклад, зв'язаний список або масив є конкретними структурами даних, які реалізують АСД "список". Структура даних визначає, як дані зберігаються та обробляються.

Структура даних — це конкретне втілення абстрактної структури даних (АСД). Це означає, що кожна структура даних має свою конкретну реалізацію в програмуванні, яка визначає, як саме будуть організовані дані і які будуть використовуватися алгоритми для маніпуляцій з ними.

Основні структури даних

1. Масив (Array) та Tuple (кортеж)

Масив — це структура даних, яка дозволяє зберігати кілька елементів одного типу в певному порядку. Елементи масиву зберігаються в сусідніх пам'ятних локаціях, що дозволяє швидко доступатися до елементів за індексом. Масиви мають фіксовану довжину (в більшості мов програмування).

Tuple (кортеж) схожий на масив, але з однією відмінністю — він не змінний, тобто елементи не можна змінювати після створення кортежу.

2. Список (List)

Список — це лінійна структура даних, де елементи зберігаються в порядку їх додавання. На відміну від масиву, список може змінювати свою довжину динамічно. В деяких мовах програмування списки є зв'язними структурами даних, тобто елементи можуть не зберігатися в сусідніх локаціях пам'яті.

3. Черга (Queue)

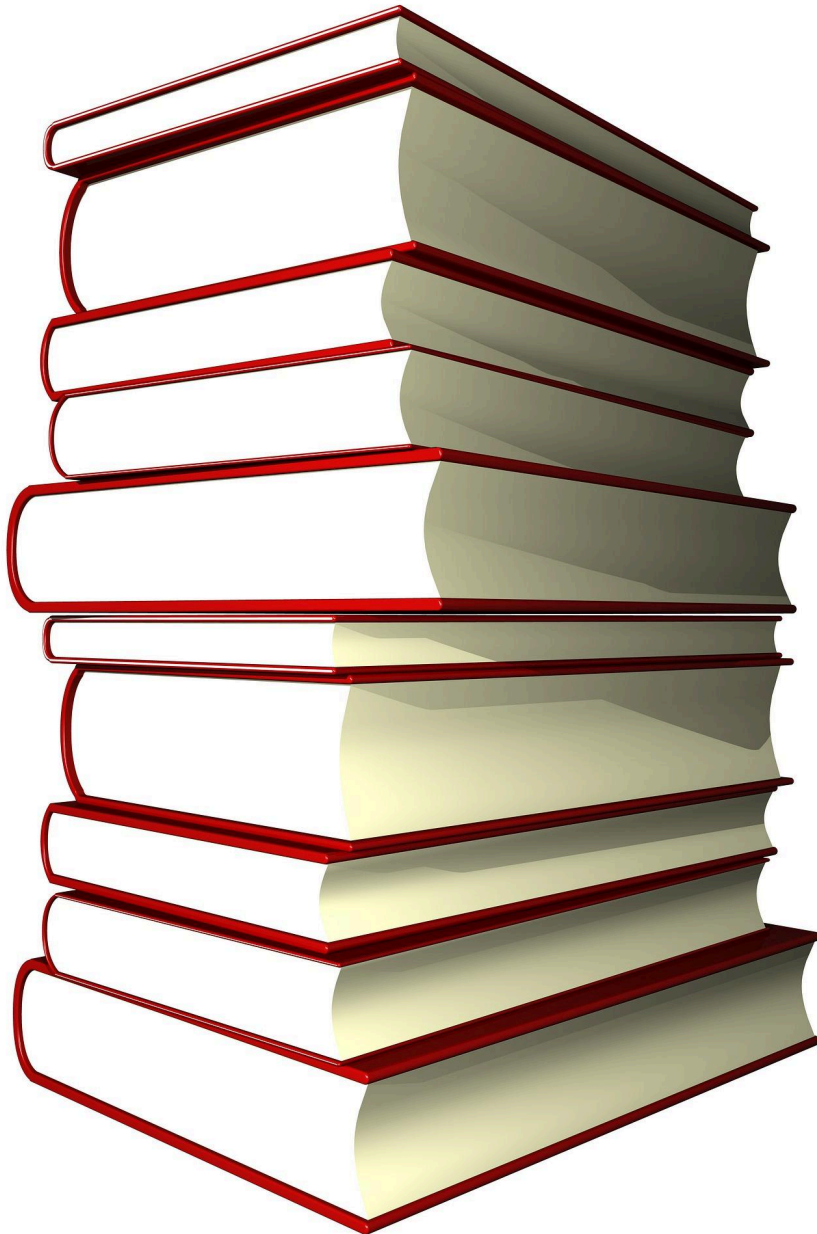
Черга — це структура даних, що працює за принципом "першим прийшов — першим пішов" (FIFO). Це означає, що елементи додаються в кінець черги і видаляються з її початку. Черга є корисною для обробки даних, де порядок обробки є критичним, наприклад, у плануванні задач.



Жива черга – це неформальна система, де людина має бути фізично присутньою, щоб зайняти своє місце в черзі, і порядок визначається зазвичай за принципом "хто прийшов раніше". На відміну від черги за записом (наприклад, через онлайн-систему чи попередню

реєстрацію), жива черга не має формалізованого механізму фіксації порядку, і люди самостійно стежать за своєю позицією. Це часто призводить до хаотичності, адже хтось може "зайняти чергу" для іншої особи або порушити порядок.

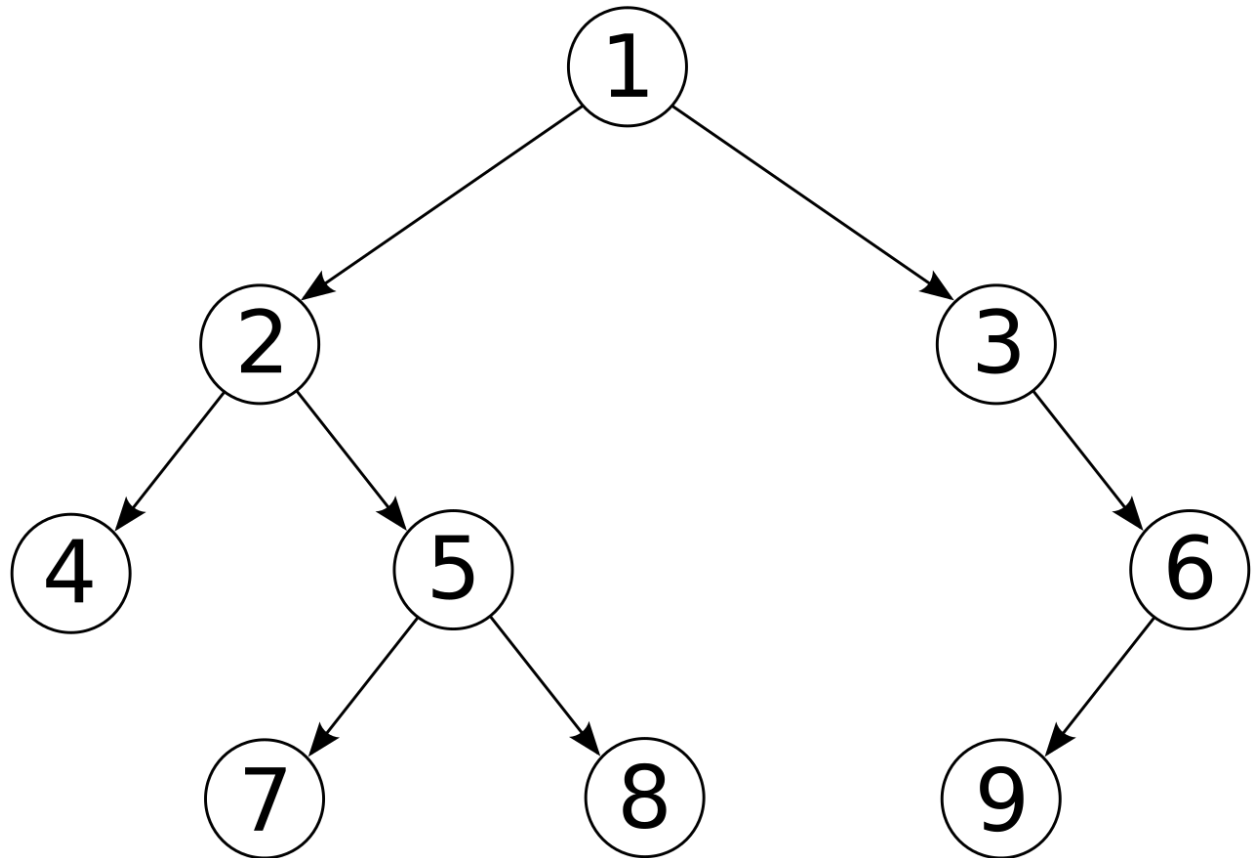
4. Стек (Stack)



Стек працює за принципом "останнім прийшов — першим пішов" (LIFO). Операції додають елемент в верхівку стеку і видаляють його з верхівки. Стек корисний для задач, де потрібно зберігати тимчасову інформацію або обробляти дані в зворотньому порядку, наприклад, при зворотному обході виразів.

Стек можна уявити як стопку тарілок, з якої можна взяти верхню, і на яку можна покласти верхню тарілку. Інша назва стека — “магазин”, за аналогією з принципом роботи магазину в автоматичній зброї.

5. Дерево (Tree) та граф



(Картинка: структура даних “дерево”)

Дерево — це ієрархічна структура даних, де елементи (вузли) з’єднані між собою через різьки. Найпоширеніший тип дерева — бінарне дерево, де кожен вузол може мати не більше двох нащадків. Деревя використовуються для зберігання ієрархічних зв’язків, таких як файлові системи або організаційні структури.

Приклади дерев включають бінарні дерева, AVL-деревя та B-деревя, які використовуються для пошуку, сортування та індексації.

У програмуванні збалансоване дерево в загальному розумінні цього слова — це такий різновид двійкового дерева пошуку, яке автоматично підтримує свою висоту, тобто кількість рівнів вершин під коренем є мінімальною.

Купа (англ. heap) в інформатиці — спеціалізована деревоподібна структура даних, в якій існують певні властивості впорядкованості: якщо B — вузол нащадок A — тоді $ключ(A) \geq ключ(B)$. З цього випливає, що елемент з найбільшим ключем завжди є кореневим вузлом. Не існує ніяких обмежень щодо максимальної кількості елементів-нащадків, яку повинна мати кожна ланка, однак, на практиці, зазвичай, кожен елемент має не більше двох нащадків. Купа є однією із найефективніших реалізацій абстрактного типу даних, який має назву черга з пріоритетом. Купи відіграють критичну роль у низці ефективних алгоритмів роботи з графами, як то в алгоритмі Дейкстри та в алгоритмі сортування пірамідальне сортування. Найуживанішим класом куп є бінарні купи.

Граф — це математична модель, яка складається з множини вершин (вузлів) та множини ребер (з'єднань між вершинами). Графи використовуються для представлення різних структур, де об'єкти пов'язані між собою, наприклад, у комп'ютерних мережах, соціальних мережах, маршрутах транспорту тощо.

Існує кілька типів графів. Залежно від наявності напрямку ребер, графи поділяються на орієнтовані, де кожне ребро має напрямок (тобто йде від однієї вершини до іншої), та неорієнтовані, де ребра не мають напрямку і просто з'єднують дві вершини. За кількістю ребер між парами вершин графи можуть бути простими, тобто без кратних ребер і петель, або мультиграфами, які допускають кілька ребер між одними й тими ж вершинами. Якщо граф містить ребра, що з'єднують вершину саму з собою, то такі графи називають графами з петлями.

Також графи бувають зваженими — коли кожне ребро має числове значення (вагу), яке може означати довжину, вартість або будь-який інший параметр — або незваженими, коли всі ребра вважаються однаковими. За зв'язністю графи бувають зв'язними — коли між будь-якими двома вершинами існує шлях — та незв'язними — коли деякі вершини не мають з'єднань одна з одною.

Серед окремих типів графів виділяють дерево — це зв'язний граф без циклів, цикл — граф, у якому ребра утворюють замкнене кільце, DAG (ациклічний орієнтований граф) — це орієнтований граф без циклів, повний граф — у якому кожна пара вершин з'єднана ребром, а також двочастковий граф — у якому всі вершини можна розділити на дві множини, і кожне ребро з'єднує вершини з різних множин.

Таким чином, графи — це гнучкий інструмент для моделювання зв'язків у найрізноманітніших галузях, від математики до комп'ютерних наук.

Розфарбування графа (вершинне)

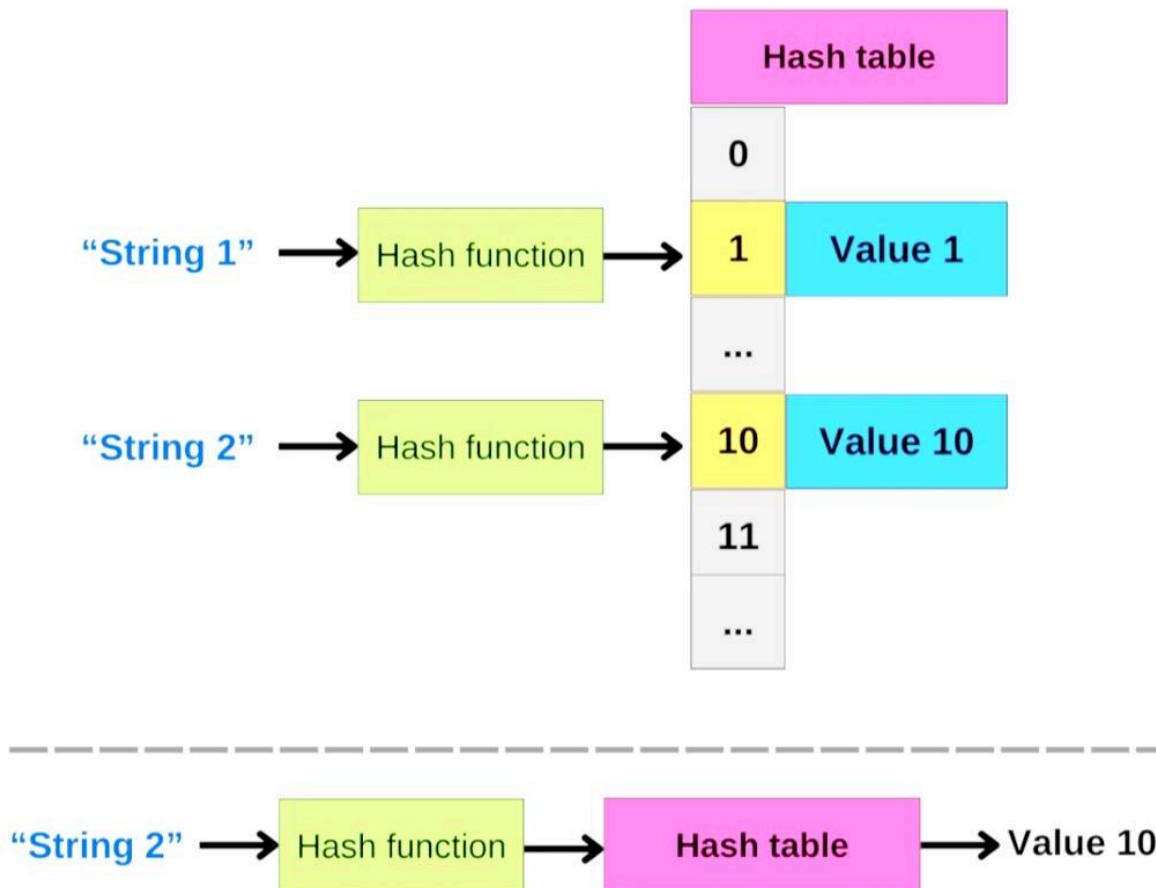
Це спосіб призначити кольори вершинам графа так, щоб жодні дві суміжні вершини не мали одного кольору.

Хроматичне число графа ($\chi(G)$) — мінімальна кількість кольорів, потрібна для правильного розфарбування графа.

Наприклад, якщо $\chi(G) = 3$, значить, достатньо трьох кольорів, щоб розфарбувати всі вершини без конфліктів.

6. Хеш таблиця (Hash Table)

Хеш таблиця використовує хеш-функцію для перетворення ключа в індекс, за яким можна знайти відповідний елемент. Вона забезпечує дуже швидкий доступ до елементів, а також є основою для реалізації асоціативних масивів або словників.



7. Зв'язаний список (Linked List)

Зв'язаний список — це структура даних, в якій кожен елемент містить вказівник на наступний елемент. Це дозволяє динамічно розширювати або скорочувати список без необхідності зміщувати елементи. Зв'язаний список є корисним для додавання або видалення елементів на початку чи в середині списку.

8. Асоціативний масив (англ. associative array) (або словник, хеш, в англійській літературі також застосовуються терміни mapping, hash, dictionary) — абстрактний тип даних, що дозволяє зберігати дані у вигляді набору пар ключ — значення та доступом до значень за їх ключем.

Реалізації асоціативних масивів зазвичай підтримують операції додавання пари, а також пошуку та видалення пари за ключем:

-вставити (ключ, значення)

-шукати (ключ)

-вилучити (ключ)

Передбачається, що асоціативний масив не може містити дві пари з однаковими ключами. У парі (k, v) значення v називається значенням, що асоціюється з ключем k . Залежно від реалізації, ключі та значення можуть задаватися і множинами значень.

Асоціативний масив з погляду інтерфейсу зручно розглядати як звичайний масив, в якому як індекси можна використовувати не тільки цілі числа, але і значення інших типів, наприклад, рядка.

Різниця між чергою в реальному житті та чергою як структурою даних

У реальному житті черга — це лінійне чергування людей або об'єктів для отримання певної послуги, наприклад, в банку чи на касі. Принцип черги в житті також ґрунтується на FIFO (First In, First Out), тобто перші, хто встане в чергу, перші і обслуговуються.

В контексті програмування, черга є абстракцією цього процесу. Програміст може реалізувати чергу різними способами, наприклад, використовуючи масиви або списки, і застосовувати операції додавання елементів у кінець черги та видалення з початку.

Черга в повсякденному житті

Куди преш без черги!?



Як насправді працює черга у житті та в програмуванні.

В живій черзі кожен учасник намагається слідкувати за іншими, бо всі мають спільний інтерес — не пропустити свою чергу. Люди, які стоять позаду, часто відчують певний соціальний тиск, щоб стежити за тим, чи не влізає хтось перед тобою без черги. Але це надлишковий механізм, який, на жаль, часто вимагає практика, через те, що є непорядні

кадри. (Тому, в ідеалі, має бути все по запису, та контролюватися супервізором, а не якась жива черга, яка сама організовується. Наприклад, за номерками. Якщо пропустив свій номер, стаєш на початок черги.).

Жива черга працює за принципом "перший прийшов — перший пішов" (FIFO) і часто реалізується як зв'язаний список (linked list). Це означає, що кожна людина в черзі знає, хто йде після неї, але не обов'язково знає, хто був перед нею. Тобто, друга людина знає, хто перша, третя знає, хто друга, і так далі.

Іноді буває, що хтось тимчасово відходить із черги і просить наступного зберегти йому місце. Проблема в тому, що це порушує логіку черги. Якщо кожен в черзі знає тільки про наступного, то для того, щоб таке працювало, треба додавати можливість знати й про попередніх. На перший погляд, це здається незначною зміною, але насправді виникає проблема, бо для нормальної роботи черги, кожна людина повинна мати чітке уявлення, хто йде перед нею, але в даному випадку це не можливо, бо особа відсутня фізично (тобто немає чіткого знання про те, хто ж зарезервував місце).

Наприклад, уявімо, що нова людина стає в чергу не за вами, а за вашим другом, який відходить, але був позаду вас. Це означає, що новий учасник черги не має можливості правильно контролювати своє місце, бо він не бачить, хто був перед ним. А якщо кілька людей так забронюють собі місце, черга стає хаотичною і вже не може працювати належним чином. Тим більше, якщо людина, яка резервувала інший місце вже вийшла з черги. Що тоді робити, вклинюватися в чергу і щось доводити, чи ставати в кінець?! Інтуїтивно, людина, яка просить тимчасово потримати місце у черзі, просить того, хто попереду, а не позаду, бо нібито той, хто позаду, може обманути й піти перед нею. Коротше, тому такі дії не варто робити.

Звичайна жива черга — це свого роду ланцюг людей, де кожен знає своє місце і хто стоїть перед ним. Це створює певний порядок, і всі чітко розуміють, де вони знаходяться в черзі. Однак, на практиці часто виникають ситуації, коли люди починають спостерігати за чергою, щоб запобігти порушенню черговості.

Коли одній людині дозволяється "залишити місце", це може створити прецедент для інших. І ось ми маємо ситуацію, де кожен може почати "резервувати" місце для друга або просто попросити когось зберегти місце, що в підсумку призводить до хаосу. Люди почнуть відчувати, що їх можуть обігнати без всяких наслідків, і це може привести до того, що всі спробують "перехитрити" один одного, що знову ж таки порушить баланс черги.

Черга з пріоритетом — це абстрактний тип даних, де елементи обслуговуються на основі їхнього пріоритету, а не порядку їх додавання. Елементи з вищим пріоритетом обробляються раніше за елементи з нижчим пріоритетом. Це схоже на звичайну чергу, але з додатковим виміром пріоритету.

Уявіть собі відділення невідкладної допомоги лікарні. Пацієнтів не обов'язково приймають у порядку їх надходження. Натомість їх сортують (призначають пріоритетом) на основі тяжкості їхнього стану. Пацієнта з найвищим пріоритетом (найкритичніший стан) приймають першим, незалежно від того, чи прибув він раніше чи пізніше за інших пацієнтів.

Правило балансу для масиву або черги (аналог правила Кірхгофа):

Сума всіх доданих елементів дорівнює сумі всіх видалених елементів та тих, що залишились у структурі.

Формулою:

Додано = Видалено + Залишилось;

Закон Літтла

Закон Літтла (англ. Little's Law) — це фундаментальний закон у теорії масового обслуговування (черг, логістики, систем виробництва), який встановлює просте, але дуже потужне співвідношення між трьома ключовими характеристиками системи:

 Формула:


$$L = \lambda * W,$$

де:

L — середня кількість елементів у системі (наприклад, клієнтів у черзі або деталей у виробництві),

λ (лямбда) — середня інтенсивність надходження (вхідний потік), тобто скільки об'єктів надходить до системи за одиницю часу,

W — середній час перебування одного елемента в системі.

 Що це означає?

Середня кількість об'єктів у системі = швидкість їх надходження × середній час перебування.

Цей закон працює незалежно від розподілу вхідного потоку чи часу обслуговування, тобто дуже загальний і універсальний.

Висновок

Структури даних — це основа ефективної обробки даних у програмуванні. Знання основних типів структур даних дозволяє вибирати найбільш підходящий спосіб організації та маніпулювання інформацією в залежності від конкретних вимог задачі. Абстрактні структури даних дають можливість розглядати загальні принципи роботи з даними, а конкретні реалізації в програмуванні дозволяють ефективно вирішувати реальні задачі.

Джерела:

1. Ахо, А.В., Хопкрофт, Дж.Е., Ульман, Д.Д. — "Алгоритми та структури даних".
2. Вірт, Н. "Алгоритми та структури даних".
3. Кнут, Д. "Мистецтво програмування", томи 1-4.