# ОЛЕКСАНДРІЙСЬКИЙ ПОЛІТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ

*Інженерія програмного забезпечення 121*

Алгоритми і структури даних

РЕФЕРАТ

На тему: «Теми на самостійне опрацювання»

Виконав:

Студент групи ПЗ-221 Желєзняк Єгор

Викладач:

Федоренко Наталія Олександрівна

Олександрія

**2024**

# Циклічні списки

Циклічні списки (або кругові списки) — це особливий вид зв'язаних списків, у яких останній елемент вказує на перший, утворюючи замкнене коло. Відмінність від лінійних списків полягає у тому, що у звичайному зв'язаному списку останній вузол вказує на null, тоді як у циклічному списку

він вказує на початок списку.

1. *Типи циклічних списків:*
   * Однозв’язний циклічний список (кожен вузол має лише посилання на наступний).
   * Двозв’язний циклічний список (вузли мають посилання на попередній і наступний).
2. *Переваги циклічних списків:*
   * Безперервна ітерація: легко обійти весь список, оскільки ітерація автоматично повертається до початку.
   * Ефективність у роботі з круговими структурами даних: наприклад, реалізація черги (FIFO), алгоритмів кругового планування.
   * Зменшення кількості перевірок: не потрібно перевіряти, чи досягнуто кінця списку, бо його немає.
3. *Операції з циклічними списками:*
   * додавання елементів – елемент додається, замикаючи його на початок або кінець списку;
   * видалення елементів – видаляється вузол, після чого попередній вузол замикається на наступний;
   * пошук – можна шукати елемент через обхід до повернення до початкової точки;
   * розбиття списків – циклічний список можна розділити на два, якщо знайдено середній елемент.
4. *Застосування циклічних списків:*
5. Кругові черги: наприклад, обробка запитів у багатозадачних системах.
6. Ігрові алгоритми: робота з колами гравців або подій.
7. Реалізація буферів: наприклад, кільцевий буфер (circular buffer) для роботи з потоками даних.
8. *Переваги та недоліки циклічних списків:*

Переваги:

* + простота реалізації кругових структур;
  + економія пам'яті через відсутність необхідності у зовнішньому маркуванні кінця.

Недоліки:

* + складніше налагодження через замкнутість структури;
  + ризик зациклення під час обходу без належних перевірок.

І на завершення можна сказати, що циклічні списки є ефективною і зручною структурою даних для задач, де необхідно організувати безперервний або круговий доступ до елементів. Завдяки своїй особливій структурі, вони ідеально підходять для реалізації кругових черг, ігрових алгоритмів, планування завдань та буферизації даних.

Однак, використання циклічних списків потребує ретельного підходу через ризик зациклення або складності в налагодженні. Правильне розуміння їхньої роботи та практичне опрацювання операцій, таких як додавання, видалення й пошук, допомагає оволодіти цим інструментом і застосовувати його у відповідних ситуаціях.

Вивчення циклічних списків розширює розуміння основ алгоритмів і структур даних, забезпечуючи міцну базу для вирішення більш складних задач у програмуванні.

# Нелінійні структури даних

Нелінійні структури даних — це такі структури, у яких елементи не організовані послідовно, як у лінійних структурах, а утворюють більш складні зв’язки, що дозволяють оптимізувати певні операції.

1. *Особливості:*
   * дані розташовані нелінійно, утворюючи ієрархічну або сіткову організацію;
   * забезпечують більш ефективний доступ до елементів у деяких задачах;
   * використовуються для роботи з великими й складними наборами даних.
2. *Таблиці як нелінійна структура даних*

Таблиці (або хеш-таблиці) — це структура, що використовує хеш-функції для швидкого доступу до даних.

Основні характеристики:

* + організовані у форматі ключ-значення;
  + використовують хеш-функції для визначення місця зберігання даних.

Переваги:

* + Швидкий пошук, додавання й видалення елементів (O(1) у середньому випадку).

Недоліки:

* + колізії при хешуванні;
  + додаткові витрати пам'яті.

Застосування: бази даних, кешування, асоціативні масиви.

1. *Дерева як нелінійна структура даних*

Дерева — це ієрархічна структура, де дані організовані у вузли, кожен з яких має підвузли (діти).

Основні поняття:

1. Корінь — головний вузол дерева.
2. Листя — вузли без дочірніх вузлів.
3. Батько та діти — зв’язки між вузлами. Види дерев:
   1. Бінарне дерево (максимум два дочірніх вузли).
   2. Бінарне дерево пошуку (ключі впорядковані).
   3. AVL-дерева (самобалансовані).
   4. B-дерева (збалансовані дерева для роботи з дисковими структурами).

Переваги:

* + - швидкий пошук та організація даних;
    - ідеально підходять для роботи з ієрархіями.

Застосування: бази даних, файлові системи, обробка виразів у програмах.

1. *Спискові структури (графи)*

Спискові структури — це подання графів, де вершини з'єднані через ребра.

Види графів:

* + орієнтовані та неорієнтовані;
  + зважені та незважені. Реалізація графів:

1. Графічно.
2. Матриця суміжності.
3. Матриця досяжності.
4. Матриця інцидентності.
5. Матриця найкоротших шляхів.

Переваги використання графів, заключається у тому, що вони підходять для моделювання реальних мереж, таких як соціальні мережі, маршрути, графи знань.

1. *Сіткові структури*

Сіткові структури — це складні нелінійні структури, які описують дані у формі багатовимірної матриці або сітки.

Характеристики:

* + підходять для представлення фізичних систем або багатовимірних масивів даних;
  + дані організовані у вузлах із сусідами, до яких є зв’язки. Приклади:
  + графи з регулярною топологією (решітки, кубічні графи);
  + картографічні сітки або комп’ютерна графіка.

1. *Переваги та недоліки нелінійних структур даних*

Переваги:

1. Гнучкість в організації великих обсягів даних.
2. Швидкий доступ до даних у певних сценаріях. Недоліки:
   1. Складність реалізації й розуміння.
   2. Витрати пам'яті через додаткові зв’язки.
   3. Вимоги до ефективних алгоритмів для роботи з такими структурами.
3. *Застосування нелінійних структур даних:*
   * таблиці: бази даних, кешування, словники;
   * дерева: файлові системи, пошукові двигуни, індекси баз даних;
   * графи: соціальні мережі, маршрутизація, алгоритми машинного навчання;
   * сітки: географічні інформаційні системи (GIS), тривимірне моделювання, симуляції.

Нелінійні структури даних є потужним інструментом для роботи зі складними наборами даних, що не піддаються лінійній організації. Таблиці, дерева, графи та сіткові структури забезпечують ефективне представлення, пошук та обробку інформації, особливо у випадках, коли дані мають

ієрархічний або зв’язний характер. Кожна з цих структур має свої переваги та сфери застосування: хеш-таблиці підходять для швидкого доступу до даних, дерева ефективні для роботи з ієрархіями, графи моделюють зв’язки між об’єктами, а сітки добре справляються з багатовимірними моделями.

Розуміння особливостей кожної структури допоможе обирати оптимальне рішення для конкретних задач для багатьох сучасних систем.

# Проходження бінарних дерев. Застосування дерев

Бінарне дерево — це ієрархічна структура даних, у якій кожен вузол має максимум два дочірніх вузли: лівий та правий.

1. *Проходження бінарних дерев*

Проходження (або обхід) бінарного дерева — це процес відвідування кожного вузла дерева у певному порядку.

Основні види проходження:

* 1. Обхід In-Order (Симетричний) – він використовується для сортування. Порядок значень у бінарному дереві пошуку буде впорядкованим.

Ліве піддерево → Корінь → Праве піддерево

* 1. Обхід Pre-Order (Прямий) – використовується для створення копій дерева або побудови виразів.

Корінь → Ліве піддерево → Праве піддерево

* 1. Обхід Post-Order (Зворотний) – використовується для видалення вузлів або оцінки виразів.

Ліве піддерево → Праве піддерево → Корінь

* 1. Обхід Level-Order (Рівневий) – проходження дерева рівень за рівнем (ширина дерева). Використовується для аналізу структури дерева.

1. *Застосування дерев:*

* бінарні дерева пошуку: дозволяють швидкий пошук, вставку та видалення елементів;
* AVL–дерева: самобалансовані бінарні дерева, що забезпечують O(log n) для всіх основних операцій;
* B-дерева: збалансовані дерева, спеціально створені для зберігання великої кількості даних на дисках;
* дерева виразів: математичні вирази подаються у вигляді дерева, де вузли — це операції, а листя — операнди;
* трійкові дерева (Trie): дерева для роботи з рядками, де кожен рівень відповідає за символ рядка;
* ієрархічні дерева: організація даних у вигляді дерев дозволяє ефективно моделювати ієрархії;
* двійкові купи (Heap): дерева, що використовуються для побудови черг із пріоритетами.

1. *Переваги використання дерев:*
   1. Ефективність: швидкий пошук, вставка й видалення даних (O(log n) у збалансованих деревах);
   2. Ієрархічність: зручна структура для представлення ієрархій;
   3. Гнучкість: використання у різних алгоритмах і системах.

Бінарні дерева є важливою структурою даних, що дозволяє організовувати інформацію для швидкого пошуку, сортування та інших операцій. Різноманітні способи проходження дерев (in-order, pre-order, post- order, level-order) забезпечують гнучкість у роботі з даними залежно від поставленої задачі. Різні типи дерев, як-от бінарні дерева пошуку, AVL-дерева, B-дерева та трійкові дерева, мають свої переваги та знаходять застосування у таких галузях, як бази даних, файлові системи, алгоритми оптимізації та обчислення математичних виразів.

Вивчення дерев не лише поглиблює розуміння структур даних, але й розвиває навички оптимального вирішення складних задач. Практика роботи з деревами допомагає ефективно використовувати їх у реальних системах і алгоритмах, зберігаючи баланс між швидкістю виконання операцій і споживанням ресурсів.

# Маршрути, ланцюги та цикли. Зв'язність. Дерево графів

Граф — це структура даних, яка складається з вершин і ребер, що

з'єднують ці вершини. Графи дозволяють моделювати різноманітні задачі, пов’язані зі зв’язками між об’єктами.

1. *Маршрути, ланцюги та цикли у графах*

Маршрут — це послідовність ребер, яка дозволяє перейти від однієї вершини до іншої. Маршрут може повторювати вершини й ребра.

Ланцюг — це маршрут, у якому жодне ребро не повторюється.

Цикл — це замкнений ланцюг, у якому перша й остання вершини збігаються, але інші вершини не повторюються.

Цикли бувають:

* простий цикл: цикл без повторення ребер або вершин (окрім початкової/кінцевої вершини);
* Гамільтонів цикл: проходить через кожну вершину графа рівно один раз;
* Ейлерів цикл: Проходить через кожне ребро графа рівно один раз.

1. *Зв'язність графів*

Зв'язність у неорієнтованих графах:

* + зв’язний граф: існує шлях між будь-якими двома вершинами;
  + компонента зв’язності: максимальний підграф, у якому всі вершини зв’язані.

Зв'язність в орієнтованих графах:

* + сильна зв’язність: існує шлях між будь-якою парою вершин у обох напрямах;
  + слабка зв’язність: існує зв’язок між вершинами, якщо ігнорувати орієнтацію ребер;
  + косильно-зв’язний граф: містить сильні компоненти зв’язності, з'єднані між собою.

Перевірка зв’язності:

* + для неорієнтованих графів: пошук компонент зв’язності за допомогою DFS або BFS;
  + для орієнтованих графів: алгоритм Косарайю для пошуку сильних компонент зв’язності.

1. *Дерево графів*

Дерево — це зв’язний ациклічний граф, у якому між будь-якими двома вершинами існує тільки один шлях.

Властивості дерев:

* + якщо граф є деревом, то кількість ребер завжди дорівнює ∣V∣ - 1|, де ∣V∣ — кількість вершин;
  + граф без циклів з |V| - 1 ребрами завжди є деревом;
  + дерева мають кореневу вершину, від якої починається ієрархія. Операції з деревами у графах:

1. Пошук мінімального кістякового дерева: Знаходження дерева, що з'єднує всі вершини з мінімальною загальною вагою. Для цього використовуються 2 основних алгоритми:
   * алгоритм Крускала (основа: сортування ребер);
   * алгоритм Прима (основа: розширення дерева).
2. Обхід дерева: Застосовуються методи обходу дерев (DFS, BFS).
3. *Застосування маршрутів, дерев та циклів*

Маршрути та цикли:

* + оптимізація транспортних маршрутів;
  + планування мереж, таких як дороги, електричні лінії;
  + пошук маршрутів у GPS-навігаторах. Зв'язність:
  + аналіз соціальних мереж (пошук груп користувачів);
  + виявлення ізольованих компонент у системах зв’язку.

Дерева графів:

* + оптимізація витрат (наприклад, прокладка кабелів);
  + ієрархічні системи в організаціях або базах даних.

Тема маршрутів, ланцюгів, циклів, зв’язності та дерев у графах є ключовою для розуміння роботи з графовими структурами. Вивчення цих понять дозволяє моделювати й вирішувати складні задачі, пов’язані з мережею зв’язків, транспортними системами, плануванням та оптимізацією.

Маршрути та ланцюги визначають, як можна проходити граф, а цикли допомагають знайти повторювані шляхи, такі як Ейлерові або Гамільтонові. Зв’язність показує, як вершини графа пов’язані між собою, що є критичним для аналізу цілісності систем, таких як мережі зв’язку або соціальні мережі. Дерева графів забезпечують ефективний спосіб організації й оптимізації даних, наприклад, у задачах пошуку мінімального кістякового дерева.

Розуміння цих концепцій у поєднанні з алгоритмами роботи з графами відкриває широкі можливості для їх практичного застосування, від навігації до побудови складних ієрархічних систем.

# Пошук маршруту у графі. Пошук відстані між вершинами графа. Мінімальні шляхи та цикли у зважених графах

Маршрут у графі — це послідовність вершин, зв’язаних ребрами.

Зважений граф — граф, у якому кожному ребру присвоєна вага, що може відображати відстань, вартість або час.

1. *Пошук маршруту у графі*

Пошук у ширину (BFS)

* + використовується для знаходження найкоротшого шляху у незважених графах;
  + BFS досліджує всі сусідні вершини поточного вузла перед переходом на наступний рівень.

Пошук у глибину (DFS)

* + використовується для перевірки існування шляху між двома вершинами;
  + DFS досліджує одну гілку до кінця перед переходом до іншої.

1. *Пошук відстані між графами*

Незважені графи:

Використовується BFS для визначення найкоротшої кількості ребер між вершинами.

Зважені графи:

Використовується алгоритм Дейкстри, він знаходить найкоротший шлях від однієї вершини до всіх інших у зваженому графі без від’ємних ваг.

1. *Мінімальні шляхи у зважених графах*

Алгоритм Беллмана-Форда:

* + використовується для знаходження найкоротших шляхів у графах із від’ємними вагами;
  + перевіряє всі ребра |V| - 1 разів (|V| — кількість вершин).

Алгоритм Флойда-Воршелла:

* + обчислює найкоротші шляхи між усіма парами вершин;
  + працює як для орієнтованих, так і для неорієнтованих графів.

1. *Мінімальні цикли у зважених графах*

Цикли у графах:

* + Ейлерів цикл: проходить через кожне ребро рівно один раз (граф зв’язний, усі вершини мають парний ступінь);
  + Гамільтонів цикл: проходить через кожну вершину рівно один раз. Алгоритм пошуку циклів: пошук Ейлерового циклу (Алгоритм Флері)

полягає у тому, що він будує маршрут за рахунок вибору ребер, які не руйнують зв’язність графа.

1. *Застосування алгоритмів пошуку маршрутів і мінімальних шляхів*
2. Навігаційні системи: пошук оптимальних маршрутів у містах.
3. Мережі зв’язку: визначення мінімальних шляхів для передачі даних.
4. Транспортна логістика: оптимізація доставок і зменшення витрат.
5. Оптимізація в бізнесі: планування маршрутів для мінімізації часу чи витрат.

Пошук маршрутів і мінімальних шляхів у графах є важливою складовою роботи з графовими структурами даних. Ці алгоритми дозволяють вирішувати задачі, пов’язані з пошуком оптимальних шляхів між вершинами, оцінкою відстаней та знаходженням мінімальних циклів. Основні методи, такі як пошук у ширину, пошук у глибину, алгоритм Дейкстри, Беллмана-Форда та Флойда- Воршелла, застосовуються для різних типів графів – зважених і незважених, з від’ємними або додатними вагами. Алгоритми оптимізації, наприклад, для пошуку Ейлерових або Гамільтонових циклів, використовуються для складних задач у логістиці, мережах зв’язку та навігації. Їх застосування сприяє ефективному плануванню, зменшенню витрат і підвищенню продуктивності в різних сферах.

# Організація даних для прискорення пошуку за вторинними ключами. Пошук хешуванням

Хешування — це метод, який забезпечує швидкий доступ до даних через використання хеш-функцій.

Вторинні ключі — це атрибути, які використовуються для пошуку записів, але не є унікальними ідентифікаторами, як первинний ключ.

1. *Основи хешування*

Хеш-функція: математична функція, яка перетворює вхідне значення (вторинний ключ) на числове значення фіксованого розміру — хеш-код.

Хеш-таблиця: структура даних, у якій значення зберігаються за індексами, обчисленими за допомогою хеш-функції.

Переваги хешування:

* швидкість пошуку: доступ за O(1) в ідеальному випадку;
* простота реалізації. Недоліки:
* колізії (випадки, коли різні ключі мають однаковий хеш-код);
* залежність від якості хеш-функції.

1. *Колізії у хешуванні*

Причини їх виникнення пов’язані з обмеженою кількістю можливих хеш- кодів для нескінченного числа ключів.

Методи розв'язання:

* + метод відкритої адресації: зберігання нового елемента у наступній доступній комірці таблиці;
  + метод ланцюгів: використання зв'язаних списків для збереження всіх елементів з однаковим хеш-кодом;
  + метод псевдовипадкового пошуку: знаходження нового індексу за певною формулою.

1. *Організація пошуку за вторинними ключами*

Використання хешування для вторинних ключів передбачає створення додаткової структури — хеш-індексів.

Хеш-індексація:

* + вторинний ключ обробляється хеш-функцією;
  + результуючий індекс використовується для швидкого доступу до записів.

Переваги:

* + можливість швидкого пошуку навіть у великих наборах даних;
  + ефективне управління пам'яттю.

1. *Приклади використання хешування*

Бази даних: індексація великих обсягів даних за атрибутами, які не є унікальними (наприклад, категорія товару).

Мережеві системи: зберігання і пошук інформації в кешах. Файлові системи: швидкий доступ до метаданих файлів.

І як висновок, можна сказати, що хешування є ефективним методом організації пошуку за вторинними ключами, особливо для завдань, що потребують миттєвого доступу до даних. Однак його застосування слід враховувати з урахуванням характеристик даних, кількості записів та можливості колізій.