Тема: **Basic Algorithms and Data Structures**

Алгоритм

**Визначення**

Це набір інструкцій, які визначають послідовність дій для розв’язання певної задачі або досягнення певного результату.

**Характеристики алгоритму**

* **Детальність**
* **Неодмінність**
* **Визначеність**

**Детальність алгоритму**

**Висока детальність**: Алгоритм описує кожен маленький крок, який необхідно виконати для досягнення результату.

**Низька детальність**: Алгоритм описує загальний підхід до рішення проблеми, але може залишати деякі деталі або підкроки невизначеними.

**Залежність від контексту**: Детальність алгоритму може залежати від контексту, в якому він використовується.

**Неодміність алгоритму**

Алгоритм повинен завершитися за скінченний час - **скінченність**. Це означає, що алгоритм не повинен мати нескінченних циклів або рекурсій, які не призводять до завершення.

**Досягнення результату**. Після завершення алгоритму повинен бути отриманий певний результат або відповідь на поставлене завдання.

Результат алгоритму може залежати від вхідних даних, але сам процес виконання алгоритму повинен завжди бути неодмінним і завершуватися за скінченний час.

**Визначеність алгоритму**

**Чіткість інструкцій.** Кожна інструкція або команда в алгоритмі повинна бути такою, що її можна однозначно зрозуміти і виконати без будь-якого простору для інтерпретації.

**Відсутність неоднозначності**. Не повинно бути жодних кроків, які можна було б інтерпретувати по-різному. Наприклад, інструкція "вибери число" є неоднозначною, тоді як "вибери найбільше число зі списку" є однозначною.

**Послідовність**. Порядок виконання кроків має бути чітко визначений.

**Деталізація**. Наприклад, якщо алгоритм включає сортування списку, він повинен вказувати, який метод сортування використовується.

**Структура алгоритму**

**Вхід**: Це дані, які алгоритм отримує на початку своєї роботи. Це може бути введення користувача, дані з файлу, значення з бази даних тощо.

**Обробка**. Серце алгоритму, де відбувається вся логіка обробки. Тут виконуються різні операції з вхідними даними, такі як обчислення, сортування, пошук тощо.

**Вихід**: Після завершення обробки алгоритм видає результат. Це може бути виведення на екран, запис в файл, відправка даних в мережу тощо.

**Застосування алгоритмів**

**Пошук**

Алгоритми пошуку - це серце багатьох сучасних технологій, які ми використовуємо щодня.

Від швидкого знаходження контакту у вашому смартфоні до вибору релевантних результатів у пошукових системах - це завдяки алгоритмам пошуку.

Вони допомагають нам ефективно працювати з величезними об'ємами інформації, роблячи наше життя простішим та продуктивнішим.

**Сортування**

Алгоритми сортування - це ключовий інструмент в обробці даних.

Кожен раз, коли ви шукаєте контакт у своєму телефоні або переглядаєте список товарів у онлайн-магазині в порядку зростання ціни, ви користуєтеся результатами сортування.

Вони дозволяють нам організовувати величезні масиви інформації таким чином, щоб ми могли швидко та легко знаходити потрібну інформацію, роблячи наші цифрові інтеракції інтуїтивно зрозумілими та ефективними.

**Трансформація (Перетворення)**

Кожен раз, коли фото з вашого смартфона перетворюється на шедевр за допомогою фільтрів або коли ваші дані в Excel автоматично конвертуються в графік, відбувається трансформація.

Трансформація процес перетворення даних з одного формату чи структури в інший, що дозволяє нам візуалізувати, аналізувати та використовувати інформацію більш ефективно.

Трансформації роблять наші дані доступними, зрозумілими та корисними у різних контекстах.

**Криптографія**

У світі, де наші дані постійно пересилаються, зберігаються та обмінюються в цифровому форматі, криптографія стає ключовим інструментом для забезпечення конфіденційності, цілісності та автентичності інформації.

Від безпечних онлайн-платежів до захисту комунікацій від зловмисників - криптографія є тим щитом, який стоїть між нашими даними та потенційними загрозами, дозволяючи нам довіряти цифровому світу навколо нас.

**Суть використання алгоритмів**

**Ми не шукаємо "ідеального" рішення. Ми шукаємо "прийнятне" рішення, яке відповідає нашим потребам або обмеженням.**

Суть використання алгоритмів полягає в тому, щоб отримати відповідь, яка відображає баланс між точністю, швидкістю та витратами ресурсів.

**Створення алгоритму**

**Визначення проблеми**

Перший крок у створенні алгоритму - це чітке розуміння та визначення проблеми, яку потрібно вирішити.

Задайте собі питання: Що саме я хочу досягти?

Які вхідні дані будуть доступні? Який очікуваний результат?

Чітке визначення проблеми полегшує подальші етапи розробки алгоритму.

**Розробка стратегії вирішення**

На цьому етапі ви розробляєте план або стратегію, як вирішити визначену проблему.

Розгляньте різні підходи до вирішення проблеми та виберіть найбільш оптимальний.

Створіть крок за кроком інструкції, які допоможуть досягти бажаного результату.

**Кодування стратегії**

Перетворіть вашу стратегію вирішення в конкретний код на обраній вами мові програмування.

Зосередьтеся на ефективності та читабельності коду.

Використовуйте коментарі та дотримуйтеся стандартів кодування, щоб ваш код було легко розуміти іншим розробникам.

**Тестування алгоритму**

Перевірте свій алгоритм на різних вхідних даних, включаючи крайні випадки.

Переконайтеся, що ваш алгоритм працює правильно та ефективно.

Якщо виникають проблеми або помилки, поверніться до етапу кодування та внесіть необхідні корективи.

**Структурування даних**

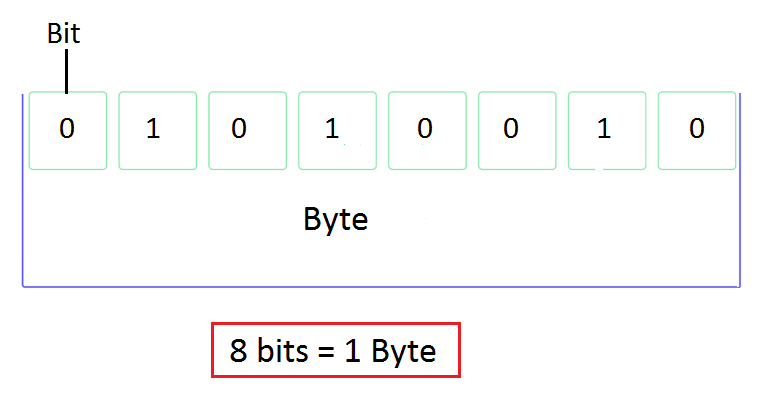
Там, де ми бачимо зображення, текст або відео, комп'ютер "бачить" лише послідовності двійкового коду.

**Біт та Байт**

**Біт** - це основна одиниця інформації в комп'ютерах, яка може мати два значення: **0** або **1**.

**Байт** - це група з **8** *бітів*, яка використовується в комп'ютерах для збереження одного символу інформації. Один байт може представляти цифру, букву, знак пунктуації або інший знак у відповідності до визначеного кодування.

**Біт та Байт**

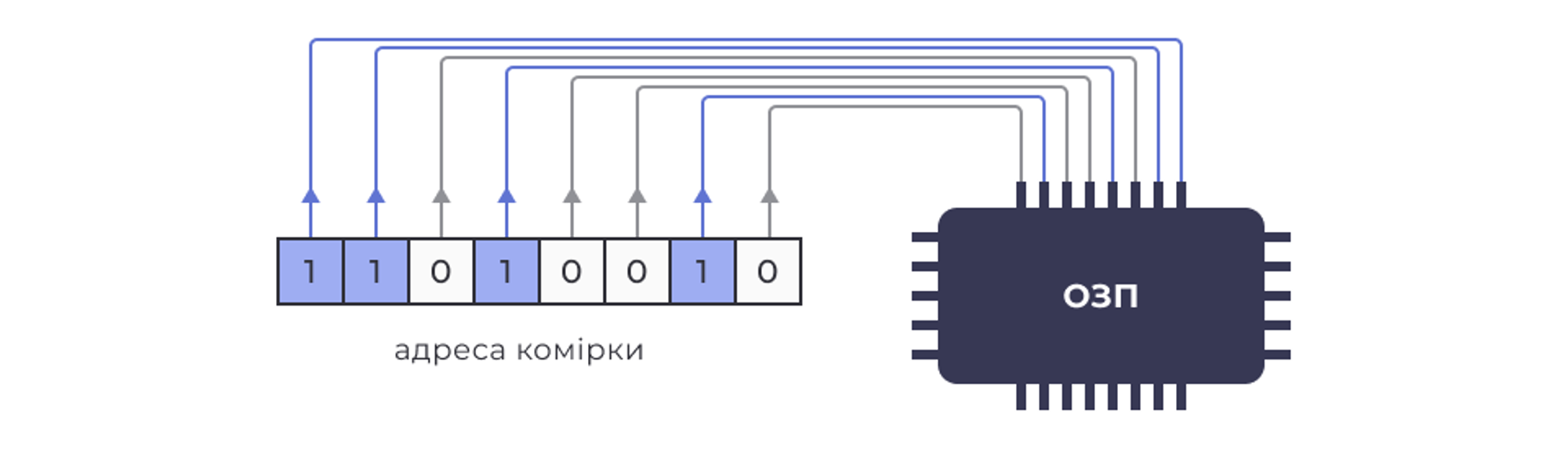


**Системи обчислення**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **десяткова** | **двійкова** | **шістнадцят-кова** | **десяткова** | **двійкова** | **шістнадцят-кова** |
| 0 | 0000 | 0 | 8 | 1000 | 8 |
| 1 | 0001 | 1 | 9 | 1001 | 9 |
| 2 | 0010 | 2 | 10 | 1010 | A |
| 3 | 0011 | 3 | 11 | 1011 | B |
| 4 | 0100 | 4 | 12 | 1100 | C |
| 5 | 0101 | 5 | 13 | 1101 | D |
| 6 | 0110 | 6 | 14 | 1110 | E |
| 7 | 0111 | 7 | 15 | 1111 | F |

**Пам’ять**

Комп'ютерна пам'ять розділена на багато окремих "клітин" або "комірок", кожна з яких може зберігати невеликий обсяг даних і має свою унікальну числову адресу. Для читання або запису даних в пам'ять використовуються операції, які впливають на одну клітину за один раз. Для доступу до конкретної клітини пам'яті необхідно передати її числову адресу.



**Оцінювання якості алгоритму**

**Метрики**

**Часова складність**

Вимірює кількість операцій, які алгоритм виконує в залежності від розміру вхідних даних. Це допомагає оцінити, наскільки швидко алгоритм може виконуватися.

**Просторова складність**

Вимірює обсяг пам'яті, який алгоритм використовує в залежності від розміру вхідних даних. Це допомагає зрозуміти, наскільки ефективно алгоритм використовує ресурси пам'яті.

**Часова складність алгоритму**

* **Найкращий випадок**, коли для вхідних даних встановленого обсягу потрібна мінімальна кількість операцій виконання алгоритму.
* **Найгірший випадок**, коли для вхідних даних встановленого обсягу потрібна максимальна кількість операцій виконання алгоритму.
* **Середній випадок** підносить середню кількість операцій, зазвичай потрібних для опрацювання вхідних даних встановленого обсягу.

**Big O notation**

Символічне представлення, яке описує, як час виконання алгоритму зростає зі збільшенням розміру вхідних даних. Воно фокусується на "найгіршому" випадку виконання алгоритму.

* ***O(1)* константний час**. Незалежно від розміру вхідних даних, алгоритм завжди виконується за однаковий час.
* ***O(n)*** **лінійний час**. Час виконання зростає лінійно зі збільшенням розміру вхідних даних.
* ***O(n²)*** **квадратичний час**. При збільшенні розміру вхідних даних вдвічі, час виконання зростає в чотири рази.
* ***O(log n)*****логарифмічний час**. Зазвичай зустрічається в алгоритмах пошуку.

***O(1)* константний час**

Функція завжди повертає перший елемент масиву.

Незалежно від розміру масиву, операція завершується за один крок, тому часова складність є константою.

def get\_first\_element(arr):

return arr[0]

***O(n)* лінійний час**

Функція перевіряє кожен елемент масиву, щоб знайти цільовий елемент. У найгіршому випадку нам потрібно буде перевірити весь масив, тому часова складність є лінійною.

def find\_element(arr, target):

for item in arr:

if item == target:

return True

return False