Міністерство освіти і науки України

Черкаський державний бізнес-коледж

Циклова комісія суспільних дисциплін

**Реферат**

З предмета Основи програмування

На тему: Обчислювальна складність алгоритмів

Виконали:

Студент 2 курсу, групи 1П-20

Зі спеціальності: Інженерія програмного забезпечення

Бондар Максим Віталійович, Куман Артем Олександрович, Мінайлова Іванна Олександрівна

Перевірив викладач:

Марченко С.В.

Черкаси-2021

**Зміст:**

ВСТУП

1. Загальна теорія алгоритмів

2. Прикладна теорія алгоритмів

3. Складність алгоритму

ВИСНОВОК

**Вступ**

Алгоритм  — це набір [інструкцій](https://uk.wikipedia.org/wiki/%D0%86%D0%BD%D1%81%D1%82%D1%80%D1%83%D0%BA%D1%86%D1%96%D1%8F_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)" \o "Інструкція (програмування)), які описують порядок дій виконавця, щоб досягти результату  [розв'язання задачі](https://uk.wikipedia.org/wiki/%D0%A0%D0%BE%D0%B7%D0%B2%27%D1%8F%D0%B7%D0%B0%D0%BD%D0%BD%D1%8F_%D0%B7%D0%B0%D0%B4%D0%B0%D1%87) за скінченну кількість дій; система правил виконання дискретного процесу, яка досягає поставленої мети за скінченний час. Для візуалізації алгоритмів часто використовують [блок-схеми](https://uk.wikipedia.org/wiki/%D0%91%D0%BB%D0%BE%D0%BA-%D1%81%D1%85%D0%B5%D0%BC%D0%B0).

Для комп'ютерних програм алгоритм є списком деталізованих інструкцій, що реалізують процес обчислення, який, починаючи з початкового стану, відбувається через послідовність логічних станів, яка завершується кінцевим станом. Перехід з попереднього до наступного стану не обов'язково детермінований — деякі алгоритми можуть містити елементи випадковості.

Поняття алгоритму належить до підвалин [математики](https://uk.wikipedia.org/wiki/%D0%9C%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0). Обчислювальні процеси алгоритмічного характеру (як-то арифметичні дії над [цілими числами](https://uk.wikipedia.org/wiki/%D0%A6%D1%96%D0%BB%D1%96_%D1%87%D0%B8%D1%81%D0%BB%D0%B0" \o "Цілі числа), знаходження [НСД](https://uk.wikipedia.org/wiki/%D0%9D%D0%B0%D0%B9%D0%B1%D1%96%D0%BB%D1%8C%D1%88%D0%B8%D0%B9_%D1%81%D0%BF%D1%96%D0%BB%D1%8C%D0%BD%D0%B8%D0%B9_%D0%B4%D1%96%D0%BB%D1%8C%D0%BD%D0%B8%D0%BA) двох чисел тощо) відомі людству з глибокої давнини. Проте чітке поняття алгоритму сформувалося лише на початку [XX століття](https://uk.wikipedia.org/wiki/XX_%D1%81%D1%82%D0%BE%D0%BB%D1%96%D1%82%D1%82%D1%8F).

В методології алгоритм є базисним поняттям і складає основу опису методів. З методології виходить якісно нове поняття алгоритму як оптимальність з наближенням до прогнозованого абсолюту. Зробивши все в послідовності алгоритму за граничних умов задачі, маємо ідеальне рішення нагальних проблем науково-практичного характеру. В сучасному світі алгоритм будь-якої діяльності у формалізованому вигляді складає основу навчання на прикладах за подібністю. На основі подібності алгоритмів різних сфер діяльності була сформована концепція (теорія) [експертних систем](https://uk.wikipedia.org/wiki/%D0%95%D0%BA%D1%81%D0%BF%D0%B5%D1%80%D1%82%D0%BD%D0%B0_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0" \o "Експертна система).

1. Загальна теорія алгоритмів

Теорія алгоритмів — це окремий розділ [математики](https://uk.wikipedia.org/wiki/%D0%9C%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0), що вивчає загальні властивості [алгоритмів](https://uk.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC). Вона виникла в 30-х роках 20 століття.

Алгоритми, проте, простежуються в [математиці](https://uk.wikipedia.org/wiki/%D0%9C%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0" \o "Математика) протягом всього часу її існування. Необхідність точного математичного уточнення інтуїтивного поняття алгоритму стала неминучою після усвідомлення неможливості існування алгоритмів розв'язку багатьох масових проблем, в першу чергу пов'язаних з [арифметикою](https://uk.wikipedia.org/wiki/%D0%90%D1%80%D0%B8%D1%84%D0%BC%D0%B5%D1%82%D0%B8%D0%BA%D0%B0) та [математичною логікою](https://uk.wikipedia.org/wiki/%D0%9C%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D1%87%D0%BD%D0%B0_%D0%BB%D0%BE%D0%B3%D1%96%D0%BA%D0%B0" \o "Математична логіка) (проблеми істинності арифметичних формул та формул першопорядкового числення предикатів, 10-та проблема Гільберта про розв'язність [діофантових рівнянь](https://uk.wikipedia.org/wiki/%D0%94%D1%96%D0%BE%D1%84%D0%B0%D0%BD%D1%82%D0%BE%D0%B2%D0%B5_%D1%80%D1%96%D0%B2%D0%BD%D1%8F%D0%BD%D0%BD%D1%8F" \o "Діофантове рівняння) та ін.). Для доведення неіснування алгоритму треба мати його точне математичне визначення, тому після сформування поняття алгоритму як нової та окремої сутності першочерговою стала проблема знаходження адекватних формальних моделей алгоритму та дослідження їх властивостей. При цьому формальні моделі були запропоновані як для первісного поняття алгоритму, так і для похідного поняття алгоритмічно обчислюваної функції.

Детальний аналіз поняття [«алгоритм»](https://uk.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC) виявляє, що (I) область можливих вихідних даних і область застосовності будь-якого алгоритму є перераховуваними множинами. Своєю чергою, (II) для будь-якої пари вкладених одна в другу перераховуваних множин можна підібрати алгоритм, у якого більша множина слугує областю можливих вихідних даних, а менша — областю застосовності. Мають місце такі основні теореми: (III) функція *f* обчислювана тоді і тільки тоді, коли перераховуваний її графік, тобто множина всіх пар вигляду *<х, f(x)>*.

(IV) Підмножина *А* перераховуваної множини *X* тоді і тільки тоді розв'язна відносно*X*, коли *А* і *X\A* перераховувані. (V) Якщо *А* і *В* перераховувані, то *A об'єднати B* і *A∩B* також перераховувані. (VI) В кожній нескінченній перераховуваній множині *X* існує перераховувана підмножина з неперераховуваним доповненням (в силу (IV) ця перераховувана підмножина буде нерозв'язною відносно *X*). (VII) Для кожної нескінченної перераховуваної множини*X* існує обчислювана функція, визначена на підмножині цієї множини і яка не продовжувана до обчислюваної функції, визначеної на всій *X*. Твердження (VI) і (II) в сукупності дають приклад алгоритму з нерозв'язною областю застосовуваності.

Розв'язні і перераховувані множини складають найпростіші (і найважливіші) приклади множин, структура яких задається за допомогою тих чи тих алгоритмічних процедур. Систематичне вивчення множин конструктивних об'єктів з точки зору таких властивостей цих множин, які зв'язані з наявністю тих чи тих алгоритмів, утворює так звану [алгоритмічну теорію множин](https://uk.wikipedia.org/w/index.php?title=%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC%D1%96%D1%87%D0%BD%D0%B0_%D1%82%D0%B5%D0%BE%D1%80%D1%96%D1%8F_%D0%BC%D0%BD%D0%BE%D0%B6%D0%B8%D0%BD&action=edit&redlink=1" \o "Алгоритмічна теорія множин (ще не написана)).

Також Теорію алгоритмів можна розділити на  *дескриптивну* (якісну) і *метричну* (кількісну). Перша досліджує алгоритми з точки зору встановлюваної ними відповідності між вихідними даними і результатами; до неї належать, зокрема, проблеми побудови алгоритму, що йому властиві ті чи ті властивості, — алгоритмічні проблеми. Друга досліджує алгоритми з точки зору складності як самих алгоритмів, так і обчислень, що ними задаються, тобто процесів послідовного перетворення конструктивних об'єктів. Важливо підкреслити, що як складність алгоритмів, так і складність обчислень можуть визначатися різними способами. Розробка методів оцінки складності алгоритмів і обчислень має важливе теоретичне і практичне значення.

2. Прикладна теорія алгоритмів

Основні етапи розробки алгоритму: постановка завдання і побудова моделі, розробка і реалізація алгоритму, доведення правильності та тестування алгоритму, аналіз складності алгоритму, підготовка документації. Основні інформаційні структури даних: масиви, списки, черги, стеки. Зображення дерев, графів і множин. Методи розробки алгоритмів: структурне програмування, рекурсія, обходи дерев, “поділяй і пануй”, балансування, динамічне програмування, програмування з відходом назад, метод “гілок і меж”, евристичні та наближені алгоритми. Проходження повного циклу розробки на прикладі алгоритму пошуку кістякового дерева мінімальної ваги. Алгоритми комп’ютерної математики. Алгоритми додавання і множення чисел. Аналіз та обчислення арифметичних і логічних виразів. Алгоритми на графах, пошук у глибину, пошук найкоротшого шляху. Сортування і пошук даних. Ігрові та комбінаторні алгоритми. Чисельні методи, обчислення дійсних констант і функцій. Ймовірнісні алгоритми. Алгоритми ідентифі кації текстових рядків і скінченні автомати.

3. Складність алгоритму

Інтуїтивно можна виділити такі основні складові складності алгоритму:

1. Логічна складність — кількість людино-місяців, витрачених на створення алгоритму.

2. Статична складність — довжина опису алгоритмів (кількість операторів).

3. Часова складність — час виконання алгоритму.

4. Ємнісна складність — кількість умовних одиниць пам'яті, необхідних для роботи алгоритму.

Складність алгоритму дозволяє визначитися з вибором ефективного алгоритму серед тих, що побудовані для розв’язання конкретної проблеми.

Складність алгоритму – це кількісна характеристика, що відображує споживані алгоритмом ресурси під час свого виконання.

Оцінка складності

Складність алгоритмів зазвичай оцінюють за часом виконання або по використовуваній пам’яті. В обох випадках складність залежить від розмірів вхідних даних: масив з 100 елементів буде оброблений швидше, ніж аналогічний з 1000. При цьому мова йде не про точний час обчислень, який залежить від процесора, типу даних, мови програмування тощо. Оцінюється складність при прагненні розміру вхідних даних до нескінченності.

Часова складність алгори́тму — характеристика продуктивності алгоритму, що визначається кількістю елементарних операцій, які потрібно виконати для реалізації алгоритму.

При цьому вважають, що кожна елементарна операція виконується за однаковий час.

Часову складність оцінюють для найгіршого випадку і визначають як максимальний час, необхідний для обробки алгоритмом будь-якої множини з n елементів.

Часова складність алгоритму зазвичай визначається виразом O (f (n)) (або так званої О—нотації). Вираз O(f(n)) означає, що час виконання алгоритму зростає з тією ж швидкістю, що і функція f (n).

Поширені складності алгоритмів

• Якщо час роботи алгоритму не залежить від обсягу вхідних даних, то його часову складність позначають як O(1); приклад — визначення значення третього елемента масиву, для чого не потрібно ні запам’ятовувати елементи, ні проходити по ним декілька разів. Завжди потрібно просто дочекатися в потоці вхідних даних третій елемент і це буде результатом, на обчислення якого для будь—якої кількості даних потрібний один і той же час.

• Лінійна складність O (n): подвоєння розміру задачі подвоїть і необхідний час; приклади — алгоритм пошуку найбільшого елемента в невідсортованому масиві, для чого потрібно переглянути всі n елементів масиву; алгоритм додавання/віднімання чисел з n цифр.

• Квадратична складність O (n2): час роботи алгоритму зростає пропорційно квадрату кількості оброблюваних елементів, подвоєння розміру задачі вчетверо збільшує необхідний час; приклад — алгоритм сортування бульбашкою, що виконує два вкладені цикли перебору масиву.

• Кубічна складність O (n3): подвоєння розміру задачі збільшує необхідний час у вісім разів. Припустимо, певним алгоритмом потрібно виконати 4n3+7n умовних операцій, щоб обробити n елементів вхідних даних. При збільшенні n на час роботи буде значно більше впливати зведення n в куб, ніж множення його на 4 або ж додавання 7n.

Приклад:

Нехай дана послідовність з нулів та одиниць і нам потрібно з'ясувати, чи є там хоч одна одиниця. Яку складність матиме алгоритм розв’язання цієї задачі?

Розв’язання. Нехай n – кількість символів в послідовності. Алгоритм буде послідовно перевіряти, чи немає одиниці в поточному місці заданої послідовності, а потім рухатися далі, поки вхід не скінчиться. Оскільки одиниця дійсно може бути тільки одна, для отримання точної відповіді на це питання в гіршому випадку доведеться перевірити всі n символів входу. Таким чином, алгоритм має складність O (n), іншими словами, він лінійний.

**Висновок**

Взагалі процес алгоритмізації призначений для визначення елементарних дій та порядку їх виконання для розв’язання поставленого завдання. Існують різні способи запису алгоритмів (словесний, формульно-словесний, метод блок-схем, програмний та ін.), які застосовуються для представлення алгоритму у вигляді, що однозначно розуміється і розробником, і виконавцем алгоритму.

Для опису алгоритмів людина часто користується природною мовою, але для запису багатьох алгоритмів природна мова виявилась незручною, тому виникла необхідність у створенні штучних мов, наприклад мови математичних формул, хімічних процесів тощо. Існує спеціальна навчальна алгоритмічна мова, яка була створена для запису алгоритмів на папері; вона використовує слова природної мови, але має більш жорстку структуру. Найбільше поширення для запису логічної структури алгоритмів отримали графічні (структурні) схеми, які спрощують складання та аналіз алгоритму, полегшують перехід від запису алгоритму до написання програми.

Список використаних джерел

[**https://romanov.in.ua/11class/11-3/**](https://romanov.in.ua/11class/11-3/)

[**http://library.iapm.edu.ua/metod\_disc/pdf/2021.pdf**](http://library.iapm.edu.ua/metod_disc/pdf/2021.pdf)

[**https://phm.cuspu.edu.ua/theory\_of\_algorithms/markov.html**](https://phm.cuspu.edu.ua/theory_of_algorithms/markov.html)

[**https://uk.wikipedia.org/wiki/%D0%A2%D0%B5%D0%BE%D1%80%D1%96%D1%8F\_%D0%B0%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC%D1%96%D0%B2**](https://uk.wikipedia.org/wiki/%D0%A2%D0%B5%D0%BE%D1%80%D1%96%D1%8F_%D0%B0%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC%D1%96%D0%B2)