DOI 10.31891/2307-5732-2021-303-6-102-105
УДК 004.021

БАРМАК О. В.
Хмельницький національний університет
ORCID ID: 0000-0003-0739-9678,
e-mail: alexander.barmak@gmail.com
РАДЮК П. М.
Хмельницький національний університет
ORCID ID: 0000-0003-3609-112X
e-mail: radiukpavlo@gmail.com
МОЛЧАНОВА М. О.
Хмельницький національний університет
ORCID ID: 0000-0001-9810-936X
e-mail: momolchanova@gmail.com
СОБКО О. В.
Хмельницький національний університет
ORCID ID: 0000-0001-5371-5788
e-mail: olenasobko.ua@gmail.com

ПІДХОДИ ДО ПРАКТИЧНОГО АНАЛІЗУ ОБЧИСЛЮВАЛЬНИХ АЛГОРИТМІВ

У роботі пропонується практичний підхід до визначення основних типів алгоритмів залежно від їх ефективності за зовнішнім виглядом програмного коду. Наведено приклади аналізу ефективності програмного коду для обчислювальної складності за зменшенням ефективності, що подається як (в асимптотичних позначеннях) O(1), O(logN), O(N), O(NlogN), O(N2), O(N3). Завдання дослідження полягає в аналізі програмного коду та визначенні умов, за яких алгоритм належить до того або іншого типу обчислювальної складності. Встановлено, що основними чинниками, за якими можна оцінити обчислювальну складність алгоритму за візуальним аналізом програмного коду є наявність у коді циклів, особливо вкладених, рекурсивність алгоритму тощо.

Ключові слова: обчислювальна складність алгоритму, асимптотичний аналіз алгоритму, практичний аналіз обчислювальної складності алгоритму.

OLEXANDER BARMAK, PAVLO RADIUK,MARYNA MOLCHANOVA, OLENA SOBKO
Khmelnytskyi National University

APPROACHES TO PRACTICAL ANALYSIS OF COMPUTING ALGORITHMS

The present work proposes a practical approach to determining the main types of algorithms, depending on their effectiveness in the appearance of the software code. Examples of analysis of the software code for computational complexity are given in the order of reducing the efficiency supplied as (in asymptotic designations): O(1), O(LogN), O(N), O(NlogN), O(N2), O(N2), O(N²), O(N³). The research task was to analyze the software code and specific conditions in which the algorithm refers to a particular type of computational complexity. The aim of analyzing the complexity of algorithms is to find the optimal algorithm for solving a specific problem. The criterion of optimality of the algorithm is chosen by the complexity of the algorithm, i.e., the number of elementary operations that must be performed to solve the problem using this algorithm. The complexity function is the ratio that connects the algorithm's input data with the number of elementary operations. The paper contains a description of classical computational complexity that can be revealed by visual analysis of program code. The main types of computational complexity are (listed in descending order of efficiency) constant, logarithmic, linear, linear-logarithmic, quadratic, cubic. Also, methods for the determination of computational complexity are described. It is established that the main factors that can assess the algorithm's computational complexity for the visual analysis of the software code are the presence of cycles, especially enclosed, reversibility of the algorithm, etc. Further research could usefully explore a method of semantic analysis of program code to predict the assessment of its computational complexity.

Keywords: computational complexity of an algorithm, an asymptotic analysis of an algorithm, a practical analysis of the computational complexity of an algorithm.

Постановка та аналіз задачі

Останнім часом усе частіше комп'ютерні науки (з англ. Computer Science) означують як науки в результаті вивчення яких студент набуває уміння (Skill) в інформаційних технологіях за наступними напрямками: математика, програмування та інтелектуальний аналіз даних (штучний інтелект) [1]. Зважаючи на це означення теорія алгоритмів як складова комп'ютерних наук є ланкою, яка пов'язує математику з програмуванням. На сьогодні теорія алгоритмів розвивається, головно, за трьома напрямками [2].

Класична теорія алгоритмів вивчає проблеми формулювання завдань у термінах формальних мов, запроваджує поняття розв'язуваності задачі, проводить класифікацію задач за класами складності P, NP тощо.

Теорія асимптотичного аналізу алгоритмів розглядає методи отримання асимптотичних оцінок ресурсомісткості або часу виконання алгоритмів, зокрема для рекурсивних алгоритмів. Асимптотичний аналіз дозволяє оцінити зростання потреби алгоритму в ресурсах (наприклад, часі виконання) зі збільшенням обсягу вхідних даних.

Теорія практичного аналізу обчислювальних алгоритмів вирішує завдання отримання явних функцій трудомісткості, інтервального аналізу функцій, пошуку практичних критеріїв якості алгоритмів, розробки методики вибору раціональних алгоритмів.

Формулювання мети

У роботі пропонується практичний підхід до визначення основних типів алгоритмів залежно від їх ефективності за зовнішнім виглядом програмного коду. Наведено приклади аналізу ефективності

програмного коду для обчислювальної складності за зменшенням ефективності, що подається як (в асимптотичних позначеннях) *O(1), O(logN), O(N), O(NlogN), O(N2), O(N3).* Завдання дослідження полягає в аналізі програмного коду та визначені умов, за яких алгоритм належить того або іншого типу обчислювальної складності.

Асимптотичний та практичний аналіз обчислювальних алгоритмів. Метою аналізу трудомісткості алгоритмів є знаходження оптимального алгоритму для розв'язання задачі. Як критерій оптимальності алгоритму вибирається трудомісткість алгоритму, що розуміється як кількість елементарних операцій, які необхідно виконати для розв'язання задачі за допомогою даного алгоритму. Функцією трудомісткості називається відношення, що пов'язують вхідні дані алгоритму з кількістю елементарних операцій. Зауважимо, що трудомісткість алгоритмів по-різному залежить від вхідних даних. Для деяких алгоритмів трудомісткість залежить від обсягу даних, інших алгоритмів – від значень даних, у деяких випадках порядок надходження даних може впливати на трудомісткість.

Одним зі спрощених видів аналізу, що використовуються на практиці, є асимптотичний аналіз алгоритмів. Метою асимптотичного аналізу є порівняння витрат часу та інших ресурсів різними алгоритмами, призначеними для розв'язання однієї й тієї задачі, при великих обсягах вхідних даних. Використовувана в асимптотичному аналізі оцінка функції трудомісткості, що називається складністю алгоритму, дозволяє визначити, як швидко зростає трудомісткість алгоритму зі збільшенням обсягу даних.

Підхід до практичного аналізу обчислювальної складності алгоритмів. Загальний час виконання програми залежить від двох факторів: часу виконання кожного оператора та частоти виконання кожного оператора. Час виконання кожного оператора залежить від середовища виконання, операційної системи та інших системних характеристик. Залежно від ефективності існує багато типів алгоритмів, серед яких можна виділити наступні основні типи алгоритмів (перераховані за зменшенням ефективності): константний, логарифмічний, лінійний, лінійно-логарифмічний, квадратичний, кубічний. Розглянемо кожний з перерахованих типів.

Константний (в асимптотичних позначеннях – $O(1)$). Застосунок виконує фіксовану кількість операцій, які зазвичай потребують постійного часу. Прикладом може бути наведений код (лістинг 1):

```
1   int x = 10;
2   if(x > 0) {
3       x--;
4   }
5   else {
6       x++;
7   }
```

Лістинг 1. Ілюстрація коду для константного типу

Логарифмічний тип (в асимптотичних позначеннях – $O(\log N)$). Виконується повільніше, ніж програми з постійним часом. Прикладом такого алгоритму може бути алгоритм бінарного пошуку (лістинг 2).

```
1   public static int Rank(int key, int[] numbers)
2   {
3       int low = 0;
4       int high = numbers.Length - 1;
5       while (low <= high)
6       {
7           // знаходимо середину
8           int mid = low + (high - low) / 2;
9           // якщо ключ пошуку менше значення в середині
10          // то верхньою границею буде елемент до середини
11          if (key < numbers[mid]) high = mid - 1;
12          else if (key > numbers[mid]) low = mid + 1;
13          else return mid;
14      } return -1;
15  }
```

Лістинг 2. Ілюстрація коду для логарифмічного типу

Наприклад, якщо в масиві `numbers` *8* елементів, то щоб знайти шуканий елемент, потрібно послідовно ділити кількість елементів на *2*. Тобто для пошуку потрібного елемента нам треба виконати *3* цикли `while`. І такий результат, якраз описується логарифмічною функцією: $\log_2 8 = 3$. Зростання часу виконання при зростанні N збільшуватиметься на деяку постійну величину.

Лінійний тип (в асимптотичних позначеннях – $O(N)$). Як правило, зустрічається там, де метод ґрунтується на одному циклі. Наприклад, обрахування функції факторіалу (лістинг 3).

```
1   private static int Factorial(int n) {
2       int result = 1;
3       for(int i=1; i<=n; i++) {
4           result *= i;
5       } return result;
6   }
```

Лістинг 3. Ілюстрація коду для лінійного типу

Тут виконання методу залежить від *n*. Яке значення для *n* буде передано в метод, стільки разів і виконуватиметься цикл. Тобто зростання трудомісткості алгоритму для даного методу пропорційне значенню *n*, тому його і називають лінійним.

Лінійно-логарифмічний тип (в асимптотичних позначеннях – $O(N\log N)$). Прикладом такого алгоритму може бути сортування злиттям (merge sort) [3], тобто алгоритм, у реалізації якого використано рекурсію.

Квадратичний тип (в асимптотичних позначеннях – $O(N^2)$). Як правило, методи, які відповідають такому типу алгоритму, містить два цикли – зовнішній і вкладений, які виконуються для всіх значень аж до *N*. Як приклад можна навести програму сортування бульбашкою (bubble sort) масиву з *N* елементів, в якій у гіршому випадку нам треба здійснити обхід *N* * *N* елементів за допомогою двох циклів (лістинг 4).

```
1   private static void BubbleSort(int[] nums) {
2       int temp;
3       for (int i = 0; i < nums.Length - 1; i++) {
4           for (int j = i + 1; j < nums.Length; j++) {
5               if (nums[i] > nums[j]) {
6                   temp = nums[i];
7                   nums[i] = nums[j];
8                   nums[j] = temp;
9               }
10          }
11      }
12  }
```

Лістинг 4. Ілюстрація коду для квадратичного типу

Кубічний тип (в асимптотичних позначеннях – $O(N^3)$). У програмах, які відповідають цьому типу алгоритму, використовуються три вкладені цикли, наприклад – лістинг 5.

```
1  char[] chars = new char[] { 'A', 'B', 'C' };
2  for(int i=0; i<chars.Length; i++) {
3      for(int j=0; j<chars.Length;j++) {
4          for(int k=0; k<chars.Length;k++) {
5              Console.WriteLine($"{chars[i]}{chars[j]}{chars[k]}");
6          }
7      }
8  }
```

Лістинг 5. Ілюстрація коду для кубічного типу

Тут розглянуто лише деякі основні типи алгоритмів, яких насправді набагато більше.

До наведеного треба зробити декілька зауважень. Була розглянута ідеальна модель визначення складності алгоритму. Але варто зазначити, що вплив оточення (операційної системи) на виконання програми мізерно малий. Хоча насправді оточення може робити свій суттєвий внесок у кінцеву продуктивність алгоритму.

З іншого боку, здебільшого складність алгоритму залежить від наявності циклів. Метод без циклу з простими виразами має складність $O(1)$, тоді як метод з одним циклом – вже $O(N)$, що теоретично гірше, ніж $O(1)$. Однак насправді наявність простих операторів навіть без циклу може суттєво впливати на продуктивність. Багато чого тут залежить знову ж таки від конкретної логіки програми.

І ще один аспект, який може вплинути на виконання програми – це кешування. Використання кешування дозволяє повторно виконати операцію швидше. Тим самим час виконання однієї й тієї операції може бути непостійним.

Висновки

У дослідженні пропонується практичний підхід до визначення обчислювальної складності алгоритму за виглядом програмного коду. Наведені типові обчислювальні складності, що виявляються

візуальним аналізом програмного коду. Наведені основні типи обчислювальних складностей та способи їх визначення. Подальші дослідження можуть бути направлені на розробку методу семантичного аналізу програмного коду для прогнозу оцінки його обчислювальної складності.

Література

1. Gadanidis G. Artificial intelligence, computational thinking, and mathematics education. The International Journal of Information and Learning Technology. 2017. Vol. 34, No. 2. P. 133–139.

2. Тверитникова О.Є. Базові алгоритми та основи програмування. Теорія і практика : навч.-метод. посіб. / Тверитникова О.Є., Крилова В.А., Васильченков О.Г. – Харків : НТУ «ХПІ», 2020. – 264 с.

3. Sharma A. Merge sort algorithm in C#. C# Corner.URL: https://www.c-sharpcorner.com/blogs/merge-sort-algorithm-in-c-sharp (10.11.2021).

References

1. Gadanidis G. Artificial intelligence, computational thinking, and mathematics education. The International Journal of Information and Learning Technology. 2017. Vol. 34, No. 2. P. 133–139.

2. Tverytnykova O.Ie., Krylova V.A., Vasylchenkov O.H. Bazovi alhorytmy ta osnovy prohramuvannia. Teoriia i praktyka : navch.-metod. posib. – Kharkiv : NTU «KhPI», 2020. – 264 s.

3. Sharma A. Merge sort algorithm in C#. C# Corner.URL: https://www.c-sharpcorner.com/blogs/merge-sort-algorithm-in-c-sharp (10.11.2021).

Рецензія/Peer review : 24.11.2021 Надрукована/Printed :30.12.2021