УДК 004

ДОСЛІДЖЕННЯ ВИКОРИСТАННЯ ЦИКЛУ, ЗВИЧАЙНОЇ РЕКУРСІЇ ТА ХВОСТОВОЇ У МОВІ ПРОГРАМУВАННЯ SCALA

ГЛИНЧУК Л.Я. (lydmilaglin@ukr.net)
Волинський національний університет імені Лесі Українки, м. Луцьк

В тезах проаналізовано особливості хвостової рекурсії. Для дослідження характеристик рекурсій виконано дослідження продуктивності, швидкості та читабельності програмного коду з використанням циклу, звичайної та хвостової рекурсії написаного мовою програмування Scala. Для того, аби побачити та порівняти результати був вибраний приклад обчислення факторіалу числа. Отримані результати описані у висновку.

Хвостова рекурсія [1] важлива в Scala, а також в багатьох інших мовах програмування, зокрема для оптимізації роботи програм та запобігання переповнення стеку пам'яті.

У звичайній рекурсії кожен новий виклик функції додає новий запис у стек викликів, що може призвести до переповнення стеку пам'яті при великій глибині рекурсії. Хвостова рекурсія дозволяє оптимізувати рекурсивні функції, забезпечуючи те, що виклик функції буде останньою дією перед поверненням значення. Але, як сказано в [2], у Scala підтримка оптимізації хвостової рекурсії обмежена через особливості байт-коду в різних віртуальних машинах Java. Згідно з документацією Scala, хвостову рекурсію «теоретично завжди можна оптимізувати шляхом повторного використання стеку функції, яка її викликає. Проте, деякі середовища виконання, такі як Java VM, не мають достатніх засобів для ефективної імплементації повторного використання стеку викликів. Тому в Scala є лише підтримка повторного використання стеку викликів функції, остання дія якої є виклик самої себе». Оптимізація хвостової рекурсії в Scala працює тільки у випадку, коли функція x() викликає в кінці свого виконання функцію x(). Однак, якщо x() викликає y(), то виклик y() у функції x() вже не буде оптимізованим.

Використання хвостової рекурсії може значно підвищити продуктивність програми, оскільки вона уникне надмірного використання ресурсів і зменшить час виконання функцій. Scala підтримує функціональне програмування, і хвостова рекурсія є одним з важливих елементів чистого функціонального стилю. Вона дозволяє писати більш читабельний, ефективний та елегантний код. Компілятор Scala може оптимізувати хвостову рекурсію, перетворюючи рекурсивні виклики в ітерації, що може покращити продуктивність програми.

Дослідимо сказане вище на прикладах та обчислимо додатково кількість операцій та час виконання. Отже, розглянемо стандартний приклад обчислення факторіалу, де послідовно виконаємо програмування обчислення за допомогою циклу, звичайної рекурсії та з використанням хвостової рекурсії.

```
var OperationCount = 0 // Змінна для підрахунку кількості операцій
def factorial(n: Int) = { // Обчислення за допомогою циклу
var a = 1
var r: BigInt = 1 //Якщо не вказати такий тип, то більше як для 30 не порахує
while (a <= n){
 r *= a
 a += 1
 OperationCount += 1}
r }
val start = System.nanoTime()
val result = factorial(40000) // Приклад виклику функції
val end = System.nanoTime()
val time = (end - start) / 1000000000.0 // Перетворення часу в секунди
```

println(s"Кількість операцій: ${OperationCount}")
println(s"Час виконання: $time секунди")

Результат обчислення можна не виводити, бо середовище, в якому запускали на виконання код, https://scastie.scala-lang.org/ показує біля рядка коду «factorial(40000)» результат обчислення, дуже довге ціле число. Виглядає це так:

```
factorial(40000) 209169242221213236332045525676432702648837354438753434183074196798211150575121
```

Результат обчислення не поміщається, щоб скопіювати і показати. Щодо статистики, то результати наступні: «кількість операцій: 40000, час виконання: 1.040979933 секунди» (час з кожним запуском трошки відрізняється, то в більшу сторону, то в меншу). Для меншого значення, наприклад, 4000, статистика: «кількість операцій: 4000, час виконання: 0.020765382».

Наступний код – обчислення факторіалу за допомогою звичайної рекурсивної функції:

```
var OperationCount = 0 // Змінна для підрахунку кількості операцій
def factorial(n: Int): BigInt = {
  OperationCount += 1
  if (n > 0) n * factorial(n-1)
  else 1 }
```

Оскільки, решту коду аналогічна, то демонструємо тільки саму функцію.

На жаль, при обчисленні факторіалу для числа 40000, показує помилку StackOverflowError тобто переповнення стеку. При меншому значенні, наприклад 4000, помилки уже не буде і результати будуть такі: «кількість операцій: 4001, час виконання: 0.02848841 секунди».

І нарешті, за допомогою хвостової рекурсії:

```
def factorial(n: Int, acc: BigInt = 1): BigInt = {
  OperationCount += 1
  if (n > 0) factorial(n-1, acc * n)
  else acc }
```

При обчисленні факторіалу для числа 4000, результати будуть такі: «кількість операцій: 4001, час виконання: 0.026822228 секунди». Для числа 40000 тисяч помилки переповнення стеку уже не буде, а буде обчислено результат і його можна побачити так само, як і в першому прикладі. Статистика: «кількість операцій: 40001, час виконання: 1.123156151 секунди».

Отже, після проведення дослідження можна зробити такі висновки:

- для обчислення великих чисел звичайна рекурсія не завжди працює, можлива помилка переповнення стеку, але хвостова справляється з цим завданням;
- з використанням хвостової рекурсії код дійсно стає більш читабельний та елегантний;
- статистичні дані, такі як час та кількість операцій, як видно з прикладів, дають результат майже такий самий.

Тому, якщо вибирати між рекурсією та циклом, слід керуватися природою задачі. Рекурсія рекомендується для розв'язання природно рекурсивних проблем. З іншого боку, циклічні алгоритми можуть бути ефективними в сценаріях, де потрібно обробляти великі набори даних. Вони можуть бути ефективнішими і менш вимірювальними за рахунок використання менше пам'яті. [3]

Список використаної літератури

1. Хвостова рекурсія. *www.wikidata.uk-ua.nina.az*. URL: https://www.wikidata.uk-ua.nina.az/Хвостова_рекурсія.html.
2. *DSpace Repository :: Electronic Kyiv-Mohyla Academy Institutional Repository*. URL: https://ekmair.ukma.edu.ua/server/api/core/bitstreams/63160f98-a8fc-434e-b840-2c2ee3b99abd/content.
3. Рекурсія в програмуванні: що це і як застосовується?. *FoxmindEd*. URL: https://foxminded.ua/rekursiia-v-prohramuvanni/.