УДК 004.021

АНАЛІЗ ЕФЕКТИВНОСТІ РЕАЛІЗАЦІЇ АРИФМЕТИЧНИХ АЛГОРИТМІВ НА МОВАХ ПРОГРАМУВАННЯ C++ ТА PYTHON

А.К. Новокшонов

У даній роботі представлені результати обчислювального експерименту, метою якого є уточнення реальної швидкодії арифметичних алгоритмів з використанням арифметики довільної точності при реалізації на мовах програмування C++ та Python. Як математичну модель для арифметичних алгоритмів обрана цілочисельна «машина, що складає». «Машина, що складає» – це математична абстракція, введена Р. Флойдом та Д. Кнутом, суть якої полягає у тому, що лише за допомогою операцій додавання, віднімання, порівняння, присвоювання та обмеженої кількості регістрів можна з прийнятною обчислювальною ефективністю виразити більш складні операції, такі як знаходження лишку за модулем, множення, знаходження найбільшого спільного дільника, піднесення до степеня за модулем. Особливістю даної реалізації є використання арифметики довільної точності, що може бути корисним для використання у криптографічних алгоритмах.
Ключові слова: C++, Python, GMP, додавання, віднімання, найбільший спільний дільник, піднесення до степеня, числа Фібоначчі.

В данной работе представлены результаты вычислительного эксперимента, целью которого является уточнение реальной производительности арифметических алгоритмов с использованием арифметики произвольной точности при реализации на языках программирования C++ и Python. Как математическая модель для арифметических алгоритмов выбрана целочисленная «складывающая машина». «Складывающая машина» – это математическая абстракция, введённая Р. Флойдом и Д. Кнутом, суть которой заключается в том, что с помощью только операций сложения, вычитания, сравнения, присваивания и ограниченного количества регистров можно с приемлемой вычислительной эффективностью выразить более сложные операции, такие как нахождение остатка по модулю, умножение, нахождение наибольшего общего делителя, возведение в степень по модулю. Особенностью данной реализации является использование арифметики произвольной точности, что может быть полезным для использования в криптографических алгоритмах.
Ключевые слова: C++, Python, GMP, сложение, вычитание, наибольший общий делитель, возведение в степень, числа Фибоначчи.

This paper presents the results of the numerical experiment, which aims to clarify the actual performance of arithmetic algorithms implemented in C ++ and Python programming languages using arbitrary precision arithmetic. "Addition machine" has been chosen as a mathematical model for integer arithmetic algorithms. "Addition machine" is a mathematical abstraction, introduced by R. Floyd and D. Knuth. The essence of "addition machine" is the following: using only operations of addition, subtraction, comparison, assignment and a limited number of registers it is possible to calculate more complex operations such as finding the residue modulo, multiplication, finding the greatest common divisor, exponentiation modulo with reasonable computational efficiency. One of the features of this implementation is the use of arbitrary precision arithmetic, which may be useful in cryptographic algorithms.
Key words: C++, Python, GMP, addition, subtraction, greatest common divisor, exponentiation, Fibonacci numbers.

Вступ

Сучасний світ за умов високого рівня розвитку комп'ютерних технологій та високої конкуренції ставить перед науковцями та розробниками програмного забезпечення все більш складні задачі. Відповідно, виникає питання про дослідження ефективності інструментів, які можуть бути використані для автоматизації розв'язання таких задач. У даній роботі для порівняння були вибрані два лідери, які входять до першої п'ятірки світового рейтингу мов програмування [1], – це уже «класична» мова C++, програми на якій компілюються, та мова Python, програми на якій інтерпретуються і яка набрала найбільшу популярність за останні 5 років [2]. Як тестові задачі вибрані арифметичні алгоритми, що можуть бути промодельовані за допомогою «машини, що складає», яка має наступні цікаві властивості.

Розглянемо обмежений набір операцій: присвоєння, додавання, віднімання та порівняння двох цілих чисел. Виникає задача ефективної реалізації інших, більш складних операцій, використовуючи тільки наведені вище прості операції.

У роботі [3] Р. Флойдом та Д. Кнутом була запропонована така математична абстракція, як «машини, що складають» (англ. addition machines), і було доведено, що з їх допомогою можлива реалізація складних операцій з лінійним уповільненням. До таких складних операцій відносять знаходження лишку за модулем, множення, ділення, знаходження найбільшого спільного дільника та піднесення до степеня за модулем. Варто зазначити, що за допомогою цих складних операцій виражається значна частина арифметичних алгоритмів, які використовуються у сучасних криптографічних протоколах.

Таким чином, «машини, що складають» – це математична абстракція, яка являє собою обчислювальний пристрій з обмеженою кількістю регістрів, над якими можна здійснювати лише такі операції:

1) введення: *read x*;

2) виведення: *write x*;

3) присвоювання: $x \leftarrow y$;

4) додавання: $x \leftarrow x + y$;

5) віднімання: $x \leftarrow x - y$;

6) порівняння: $x \geq y$.

«Машини, що складають» можуть працювати як з цілими числами, так і з дійсними. У даній роботі усі операції вважаємо такими, що здійснюються над цілими числами, тому описаний пристрій називається цілочисельною «машиною, що складає».

Метою даної роботи є дослідження ефективності алгоритмів цілочисельної «машини, що складає» при її реалізації за допомогою мов програмування C++ і Python, використовуючи арифметику довільної точності. Використання арифметики довільної точності було вибране у зв'язку із можливим подальшим застосуванням результатів даного дослідження у криптографії.

Методика дослідження

У даній роботі обмежимося розглядом і порівнянням таких операцій над цілими числами:

1) множення двох чисел;

2) піднесення до степеня за модулем;

3) знаходження лишку за модулем;

4) знаходження найбільшого спільного дільника (НСД).

У роботі [3] були наведені наступні теоретичні оцінки складності операцій, які можуть бути реалізовані за допомогою «машини, що складає» (таблиці). Також дані оцінки можна знайти у роботі [4].

Основною ідеєю алгоритму знаходження лишку від цілочисельного ділення, який був запропонований Р. Флойдом та Д. Кнутом у [3], є використання представлення Фібоначчі замість традиційного бінарного представлення. Відомо, що будь-яке невід'ємне ціле число може бути представлене сумою чисел Фібоначчі. Далі ключовими моментами є те, що, по-перше, за допомогою «машини, що складає» можна легко переходити від пари чисел Фібоначчі $< F_t, F_{t+1} >$ до наступної пари $< F_{t+1}, F_{t+2} >$ використанням лише однієї операції додавання, або до попередньої пари $< F_{t-1}, F_t >$ лише за допомогою однієї операції віднімання. Числа Фібоначчі зростають експоненційно, а саме тому можуть бути використані як аналоги степенів двійки.

Таблиця. Число команд регістрової машини при виконанні арифметичних операцій

Операція	**Час виконання**
Лишок *x mod y*	*O(log(x/y))*
Множення *xy*	*O(log(min(|x|,|y|))*
Ціла частина *y/z*	*O(log(|y/z|))*
Найбільший спільний дільник *НСД(x,y)*	*O(log(max(x,y)/НСД(x,y)))*
Експонента $x^y \bmod z$	*O((log y)(log z)+log(x/z))*

У командах регістрової «машини, що складає» алгоритм знаходження лишку від цілочисельного ділення **x mod y** (*P*1) має наступний вигляд [3]:

```
P1: read x; read y;   { вважається, що x ≥ 0, y > 0 }
if x >= y then
   begin z ← y;
   repeat <y, z> ← <z, y + x> until not x >= z;
   repeat if x >= y then x ← x - y;
     <y, z> ← <z - y, y>;
   until y >= z;
   end;
write x.
```

Операція $<y, z> \leftarrow <z, y + x>$ позначає одночасне присвоювання $y \leftarrow z$ та $z \leftarrow y + x$.

Аналогічно алгоритм обчислення **x[y/z]** (*P*2) у командах регістрової «машини, що складає» є наступним [3]:

```
P2: read x; read y; read z;  { вважається, що y ≥ 0, z > 0 }
w ← w - w;
```

```
if y >= z then
   begin u ← x; v ← z;
   repeat <u, x> ← <x, u + x>; <v, z> ← <z, v + z>;
   until not y >= z;
   repeat if y >= v then <w, y> ← <w + u, y - v>;
     <u, x> ← <x - u, u>; <v, z> ← <z - v, v>;
   until  v >= z;
   end;
write w.
```

Алгоритм обчислення найбільшого спільного дільника **НСД(x,y)** (*P*3) у командах регістрової машини виглядає наступним чином [3]:

```
P3: read x; read y;   { вважається, що x > 0, y ≥ 0 }
z ← y; z ← z + z;
while not y >= z do
   begin while x >= z do <y, z> ← <z, y + z>
   repeat if x >= y then x ← x - y;
     <y, z> ← <z - y, y>
   until y >= z;
   <x, y> ← <y, x>; z ← y; z ← z + z;
   end;
write x.
```

І, нарешті, останній важливий алгоритм обчислення $\mathbf{x^y}$ **mod z** (*P*5) за допомогою «машини, що складає» виконується наступним чином, враховуючи допоміжний алгоритм *P*4 [3]:

```
P4: u ← 1; v ← 1; w ← y;   { u = F_l, v = F_{l+1}, l = 1 }
repeat <u, v> ← <v, u + v> until not w >= v;    { u = F_l, v = F_{l+1}, y ≥ u }
r ← 1; s ← 1; t ← t - t;    { u = F_l, v = F_{l+1}, l = λy }
repeat if w >= u then
   begin w ← w - u; t ← t + s;
   end;
   <u, v> ← <v - u, u>;
   <r, s> ← <s, r + s>   { l ← l -1 }
until u >= v.
```

```
P5: read x; read y; read z;
<r, s, t> ← <F_{λy}, F_{λy+1}, y^R> { x = x_l, w = x_{l+1}, l = 1 }
x ← x mod z; w ← x; u ← 1;
repeat if t >= r then
   begin t ← t - r; u ← (uw) mod z;
   end;
   <r, s> ← <s - r, r>;
   <x, w> ← <w, (xw) mod z>;   { l = l + 1 }
   until r >= s;
write u.
```

Результати дослідження

Для реалізації алгоритмів були використані мови програмування C++ з бібліотекою арифметики довільної точності GMP [5] та Python з вбудованою підтримкою арифметики довільної точності. Деякі більш глибокі результати порівняння ефективності мов програмування C++ та Python містяться у роботі [6].

Характеристики персонального комп'ютера, на якому було проведене тестування реалізованих алгоритмів, такі: процесор Intel Pentium Duo 2.8 ГГц, 4 Гб оперативної пам’яті, ОС Ubuntu 14.04 LTS x64. Версії програмного забезпечення: GCC 4.8.4, GMP 5.1.3, Python 2.7.6.

На рис. 1 показано результати порівняння швидкодії алгоритму *P*1 *(x mod y)* на персональному комп’ютері при реалізації його за допомогою мов програмування C++ та Python.

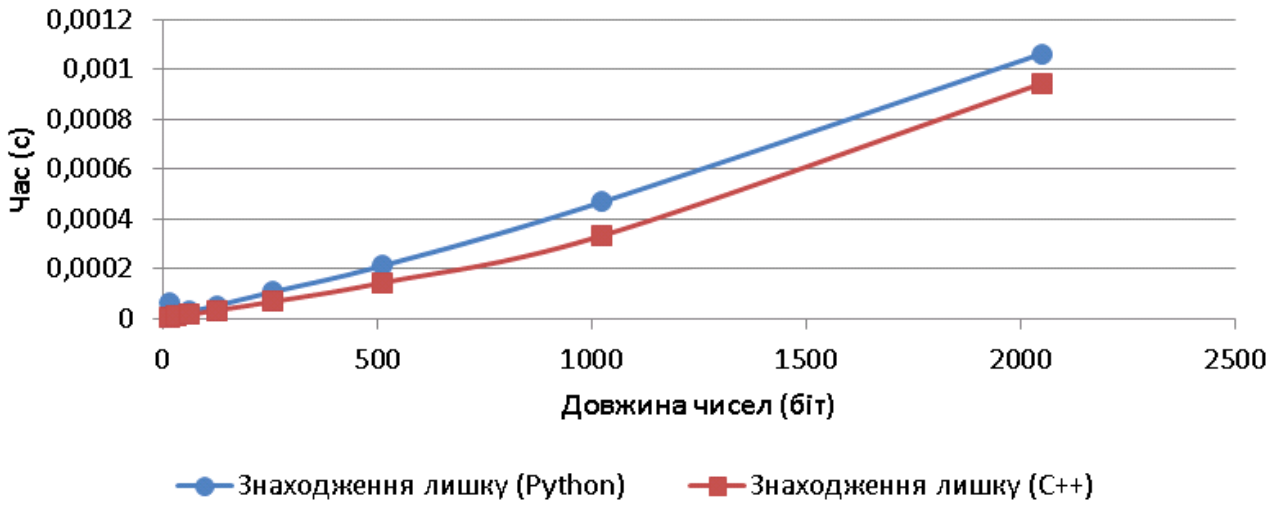

Рис. 1. Порівняння часу виконання операції знаходження лишку від ділення при реалізації на мовах програмування C++ та Python

На рис. 2 показано результати порівняння швидкодії алгоритму *P*2 (*x*[*y*/*z*]) на персональному комп'ютері при реалізації його за допомогою мов програмування C++ та Python.

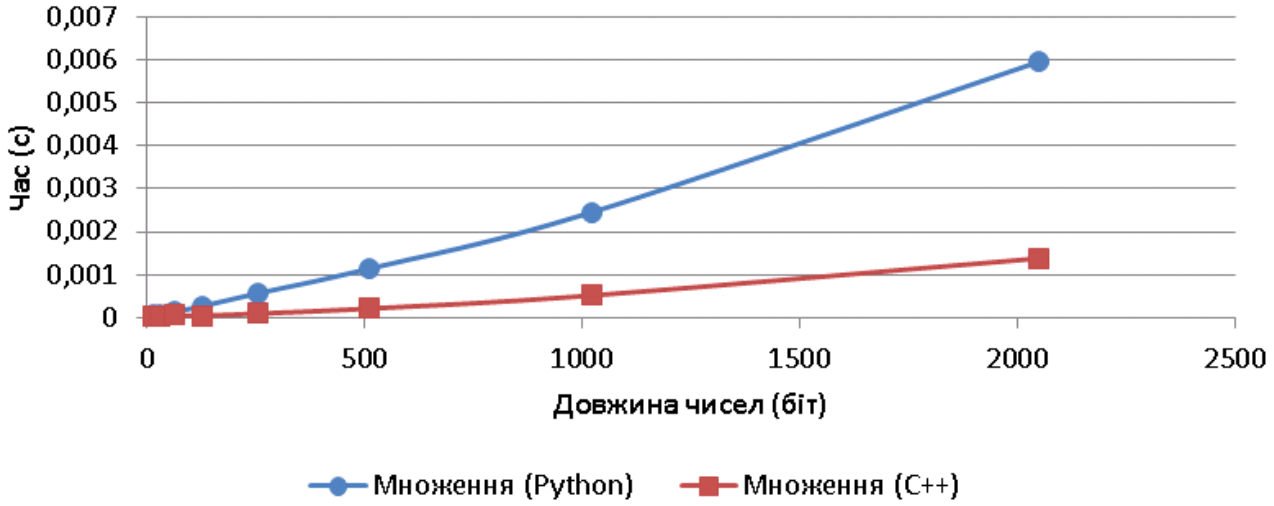

Рис. 2. Порівняння часу виконання операції множення до степеня при реалізації на мовах програмування C++ та Python

На рис. 3 показано результати порівняння швидкодії алгоритму *P*3 *(НСД(x, y))* на персональному комп'ютері при реалізації його за допомогою мов програмування C++ та Python.

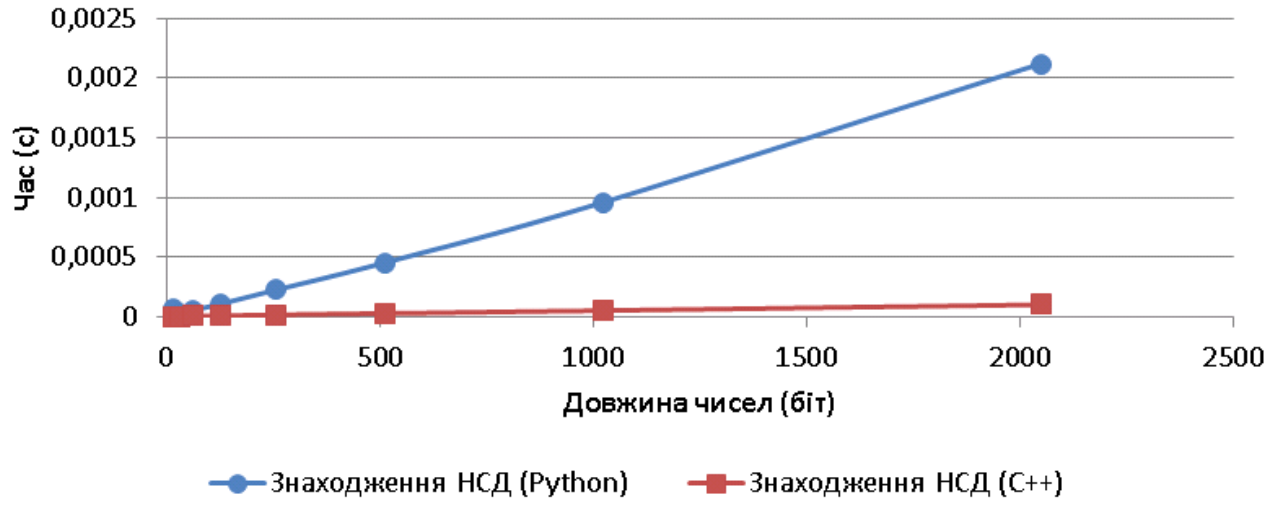

Рис. 3. Порівняння часу виконання операції знаходження найбільшого спільного дільника при реалізації на мовах програмування C++ та Python

На рис. 4 показано результати порівняння швидкодії алгоритму *P*5 ($x^y \bmod z$) на персональному комп'ютері при реалізації його за допомогою мов програмування C++ та Python. Для зручності тестування як степінь *y* була обрана константа 16.

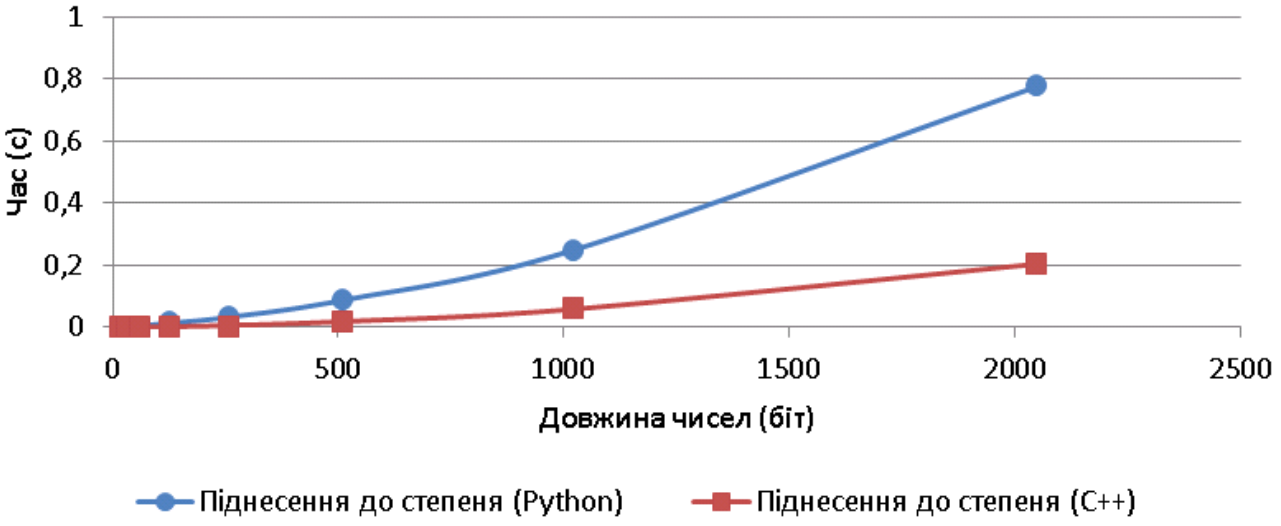

Рис. 4. Порівняння часу виконання операції піднесення до степеня при реалізації на мовах програмування C++ та Python

Висновки

В результаті дослідження були отримані чисельні характеристики ефективності алгоритмів «машини, що складає» та побудовані відповідні графіки, що дозволило на практиці підтвердити та уточнити теоретичні оцінки складності даних алгоритмів, виведені Р. Флойдом та Д. Кнутом. Одним із практичних застосувань розглянутих алгоритмів може бути їх ефективне використання на спеціально побудованому апаратному забезпеченні, наприклад, для схеми RSA [3, 7], основою якої є операція піднесення до степеня за модулем ($x^y \bmod z$), а також застосування у криптографічних протоколах для випадків, коли набір доступних операцій є обмеженим.

Експериментальним шляхом були отримані такі основні результати:

- для C++/GMP середня швидкодія вбудованих складних операцій є у 47 разів більшою, ніж їх аналогів, реалізованих за допомогою «машини, що складає»;
- для Python середня швидкодія вбудованих складних операцій є у 228 разів більшою, ніж їх аналогів, реалізованих за допомогою «машини, що складає»;
- реалізація на C++/GMP арифметичних алгоритмів «машини, що складає» є у 3,8 разу швидшою за аналогічну реалізацію на Python.

Цікавим фактом є те, що швидкодія алгоритму *НСД*(*x*,*y*) на C++ є повністю однаковою як для вбудованої операції mpz_gcd() з бібліотеки GMP [5], так і для його реалізації за допомогою «машини, що складає». Для Python-реалізації алгоритму *НСД*(*x*,*y*) ситуація схожа: стандартна функція fractions.gcd() лише у 2 рази швидша за реалізацію за допомогою «машині, що складає».

Отже, результати експериментального порівняння надають можливість зробити висновок про те, що реалізація арифметичних алгоритмів над цілими числами довільної точності на C++/GMP є в середньому у 3,8 раз швидшою за Python-реалізацію, але у той же час процес програмування на C++ є складнішим.

Загалом, інтерпретатор Python показав досить високу швидкодію, що разом з вбудованою арифметикою довільної точності, зручністю програмування та різноманіттям програмних модулів робить мову Python одним з перспективних інструментів для дослідників в області інформатики та математики. У той же час, є клас задач, які можуть бути ефективно вирішені тільки за допомогою мов C/C++ (відповідно, і за допомогою мови Assembler), наприклад, написання драйверів пристроїв, програмування мікроконтролерів. Тому для кожного класу задач необхідно правильно вибирати інструменти з урахуванням пріоритетів та уникати суперечок з приводу «найкращої» або «ідеальної» мови програмування.

1. *Tiobe*.com. Tiobe Index | Tiobe - The Software Quality Company. [online] Available at: http://www.tiobe.com/tiobe_index [Accessed 10 Feb. 2016].
2. *Pypl*.github.io. PYPL PopularitY of Programming Language index. [online] Available at: https://pypl.github.io/PYPL.html [Accessed 10 Feb. 2016].
3. *Floyd R. and Knuth D.* Addition Machines. SIAM J. Comput. – 1990. – 19(2). – P. 329–340.

4. *Анісімов А.В.* Алгоритмічна теорія великих чисел. Модулярна арифметика великих чисел. Київ: Академперіодика. – 2001.
5. *Gmplib*.org. The GNU MP Bignum Library. [online] Available at: https://gmplib.org/ [Accessed 10 Feb. 2016].
6. *Prechelt L.* An empirical comparison of C, C++, Java, Perl, Python, Rexx and Tcl // IEEE Computer. – 2000. – 33(10). – P. 23–29.
7. *Rivest R., Shamir A. and Adleman L.* A method for obtaining digital signatures and public-key cryptosystems. Communications of the ACM. – 1978. – 21(2). – P. 120–126.

References

1. Tiobe.com. Tiobe Index | Tiobe - The Software Quality Company. [online] Available at: http://www.tiobe.com/tiobe_index [Accessed 10 Feb. 2016].
2. Pypl.github.io. PYPL PopularitY of Programming Language index. [online] Available at: https://pypl.github.io/PYPL.html [Accessed 10 Feb. 2016].
3. Floyd R. and Knuth D. Addition Machines. SIAM J. Comput. – 1990. – 19(2). – P. 329–340.
4. Anisimov A.V. Algorithmic Theory of Large Numbers. Modular Arithmetic of Large Numbers [in Ukrainian], Akademperiodika, Kyiv. – 2001.
5. Gmplib.org. The GNU MP Bignum Library. [online] Available at: https://gmplib.org/ [Accessed 10 Feb. 2016].
6. Prechelt L. An empirical comparison of C, C++, Java, Perl, Python, Rexx and Tcl // IEEE Computer. – 2000. – 33(10). – P. 23–29.
7. Rivest R., Shamir A. and Adleman L. A method for obtaining digital signatures and public-key cryptosystems. Communications of the ACM. – 1978. – 21(2). – P. 120–126.

Про автора:

Новокшонов Андрій Костянтинович,
аспірант.
Кількість наукових публікацій в українських виданнях – 2.
http://orcid.org/0000-0001-8556-6158

Місце роботи автора:

Київський національний університет імені Тараса Шевченка, факультет кібернетики.
03680, Україна, м. Київ, проспект Академіка Глушкова, 4 Д.
Тел.: (044) 521 3554.
E-mail: andrey.novokshonov@ukr.net