УДК 510.863(043.2)

А. В. Крупський
О. І. Тютюнник

ОБЧИСЛЕННЯ ДИСКРЕТНОГО ЛОГАРИФМА З ВИКОРИСТАННЯМ АЛГОРИТМУ BABY-STEP GIANT-STEP У PAYTON

Вінницький національний технічний університет

Анотація
У роботі досліджено ефективність алгоритмів brute-force та baby-step giant-step для розв'язання задачі дискретного логарифмування. Реалізацію виконано мовою Python. Експериментально підтверджено істотну перевагу алгоритму Шенкса за часом виконання для великих модулів.
Ключові слова: *дискретний логарифм, алгоритм Шенкса, Python, криптографія, baby-step giant-step.*

Abstract
The paper investigates the effectiveness of the brute-force and baby-step giant-step algorithms for solving the discrete logarithm problem. The implementation is done in Python. The significant advantage of Shanks' algorithm in terms of execution time for large modules is experimentally confirmed.
Keywords: *discrete logarithm, Shanks' algorithm, Python, cryptography, baby-step giant-step.*

Вступ

У сучасному цифровому суспільстві питання захисту інформації набуває особливої актуальності. Передача конфіденційних даних через глобальні мережі, використання електронного підпису, онлайн-банкінгу, хмарних сервісів та мобільних застосунків потребують надійних криптографічних механізмів захисту. Значна частина сучасних асиметричних криптосистем базується на математичних задачах, що є обчислювально складними для розв'язання за класичних умов.

Саме складність знаходження невідомого показника лежить в основі стійкості протоколу обміну ключами Діффі-Геллмана, криптосистеми Ель-Гамаля та ряду алгоритмів на еліптичних кривих [1].

Класичний підхід повного перебору має надзвичайно високу обчислювальну складність, що робить його практично непридатним для великих параметрів. Тому особливого значення набувають оптимізовані алгоритми, серед яких важливе місце займає алгоритм baby-step giant-step (алгоритм Шенкса), що дозволяє суттєво скоротити кількість операцій пошуку.

Сучасні мови програмування високого рівня, зокрема Python, забезпечують ефективні засоби реалізації математичних алгоритмів завдяки підтримці довгої арифметики, роботі з хеш-таблицями та простоті програмної реалізації. Це робить Python зручним інструментом для дослідження криптографічних алгоритмів у навчальних і прикладних задачах [3,4].

Метою роботи є порівняльний аналіз алгоритмів brute-force та baby-step giant-step для розв'язання задачі дискретного логарифмування засобами Python, а також експериментальна оцінка їх продуктивності.

Результати дослідження

Задача дискретного логарифмування належить до класу обчислювально складних задач теорії чисел і широко використовується у сучасній криптографії. Її суть полягає у знаходженні невідомого показника степеня x, який задовольняє співвідношення:

$$g^x \equiv h (\mathrm{mod} p),$$

де g— породжувальний елемент мультиплікативної групи, h— відоме значення, p— просте число, а x— шуканий дискретний логарифм.

Класичним способом розв'язання цієї задачі є метод повного перебору, що передбачає послідовну перевірку всіх можливих значень показника x. Такий підхід є простим у реалізації, однак при великих значеннях параметра pвимагає значних часових витрат. Обчислювальна складність повного перебору становить:

$$O(p),$$

що робить його практично неефективним для сучасних криптографічних застосувань.

Альтернативним підходом є алгоритм *baby-step giant-step* (алгоритм Шенкса), який використовує принцип «зустрічі посередині». Його основна ідея полягає у поданні шуканого показника у вигляді:

$$x = im + j,$$

де $m \approx \sqrt{p}$, i та j— цілі невід'ємні числа.

Такий підхід дозволяє розбити задачу на два етапи: попереднє формування таблиці значень «малих кроків» та подальший пошук відповідностей серед «гігантських кроків». У результаті кількість необхідних перевірок значно зменшується, а часову складність алгоритму можна оцінити як:

$$O(\sqrt{p}).$$

Для кращого розуміння математичної сутності та програмної реалізації алгоритму *baby-step giant-step* доцільно розглянути його структурну схему. Візуальне представлення основних етапів роботи алгоритму дозволяє простежити логіку пошуку дискретного логарифма та принцип скорочення кількості обчислень. На рис. 1 наведено покрокову схему функціонування методу Шенкса для розв'язання задачі дискретного логарифмування.

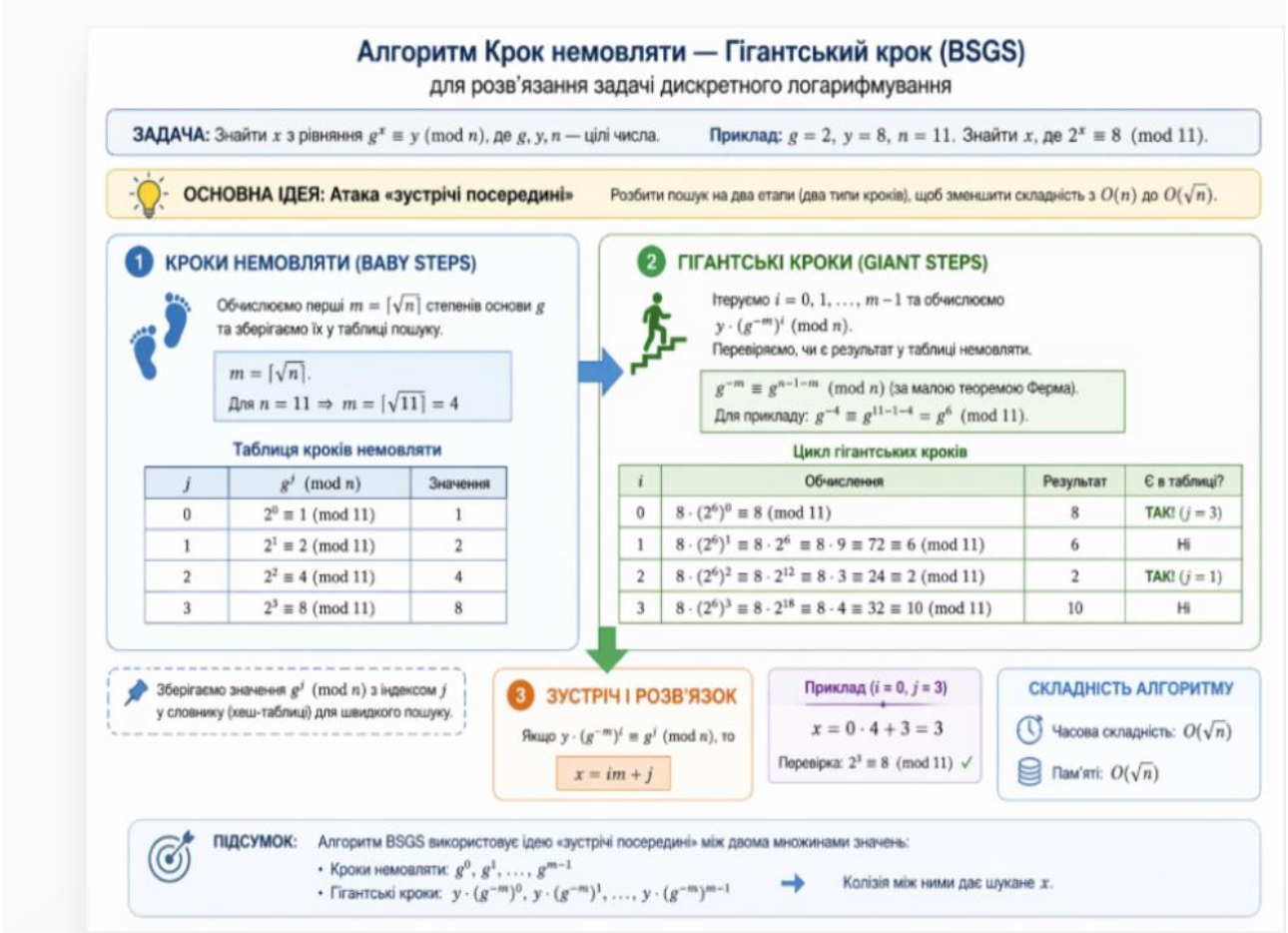

Рис. 1. Схема роботи алгоритму baby-step giant-step для знаходження дискретного логарифма

Як показано на рис. 1, алгоритм складається з двох ключових фаз. На першому етапі формується таблиця так званих «малих кроків», у якій обчислюються значення степенів породжувального елемента за модулем числа p. На другому етапі виконуються «гігантські кроки», під час яких послідовно перевіряється наявність збігів із попередньо сформованою таблицею.

Таким чином, застосування методу «зустрічі посередині» дає змогу знизити часову складність алгоритму повного перебору з $O(n)$ до:

$$O(\sqrt{n}),$$

що робить алгоритм ефективним для задач середньої розмірності та корисним інструментом для навчального моделювання криптоаналітичних процесів.

На основі схеми, наведеної на рис. 1, було реалізовано програмний код мовою Python для експериментального дослідження швидкодії алгоритму.

Програмний код

1. Підключення бібліотек

```
import math
import time
```

math - надає доступ до базових математичних функцій. У даному коді використовується для функцій math.sqrt() (квадратний корінь) та math.ceil() (округлення до більшого цілого).

time - стандартний модуль Python для роботи з часом. Використовується для фіксації початку та завершення виконання алгоритмів (time.time()), що дозволяє обчислити час роботи програми та провести порівняльний аналіз швидкодії методу Шенкса і повного перебору.

2. Підготовка інструментів для обчислення (Алгоритм Шенкса)

```
def baby_step_giant_step(g, h, p):
 m = math.ceil(math.sqrt(p))
```

Визначення параметра m: Алгоритм починається з обчислення оптимального розміру кроку m, який дорівнює $\lceil\sqrt{p}\rceil$. Це забезпечує оптимальний баланс між кількістю ітерацій та обсягом необхідної оперативної пам'яті.

```
# 1. Baby steps
baby_steps = {}
 for j in range(m):
val = pow(g, j, p)
 baby_steps[val] = j
```

Фаза "Малих кроків" (Baby steps): Створюється порожній словник baby_steps, який виконує роль хеш-таблиці. У циклі обчислюються значення $g^j \ (mod\ p)$ для *j* від 0 до *m-1* . Отримане значення стає ключем у словнику, а показник *j* — його значенням. Завдяки внутрішній архітектурі словників у Python, пошук елементів у такій структурі в подальшому відбуватиметься за константний час $O(1)$.

```
c = pow(g, (p - 2) * m, p)
```

Обчислення множника для великих кроків: Знаходиться обернене значення $g^{-m} (mod\ p)$. Для уникнення складних алгоритмів пошуку оберненого елемента застосовується Мала теорема Ферма: оскільки *p* — просте число, $g^{-1} \equiv g^{p-2}\ (mod\ p)$. Відповідно, $g^{-m} \equiv (g^{p-2})^m\ (mod\ p)$. Вбудована функція pow(base, exp, mod) у Python дозволяє виконати це модульне піднесення до степеня високоефективно.

```
# 2. Giant steps та пошук колізій
 for i in range(m):
  # Обчислення h * (g^-m)^i mod p
 gamma = (h * pow(c, i, p)) % p
  # Перевірка наявності колізії у хеш-таблиці
 if gamma in baby_steps:
  return i * m + baby_steps[gamma]
 return None
```

Фаза "Великих кроків" (Giant steps) та пошук колізій: Програма ітерує змінну *i* від 0 до *m-1* , щоразу обчислюючи нове значення gamma за формулою $h \cdot (g^{-m})^i\ (mod\ p)$. Далі відбувається миттєва перевірка: якщо обчислене значення вже існує як ключ у словнику baby_steps (тобто знайдено колізію), алгоритм успішно завершується. Знайдений дискретний логарифм повертається за формулою $x = i \cdot m + j$.

3. Алгоритм повного перебору (для порівняльного аналізу)

```
def brute_force(g, h, p)
  Знаходить x методом повного перебору (brute-force).
 for x in range(p):
  if pow(g, x, p) == h:
  return x
  return None
```

Ця допоміжна функція реалізує тривіальний лінійний пошук (brute-force). Вона інтегрована в програму виключно для емпіричного підтвердження теоретичної складності $O_{(p)}$ і наочної демонстрації переваг оптимізованого методу Шенкса.

4. Функція main (Ініціалізація та тестування)

```
def main():
# 1. Ініціалізація параметрів
 p = xxx  # Велике просте число
   g = xxx  # Основа (відоме ціле число)
 x_secret = xxx  # Секретний показник степеня
    # Обчислення результату h (відоме ціле число)
 h = pow(g, x_secret, p)
```

Ініціалізація параметрів: Задаються константи для рівняння $g^x \equiv h \ (mod\ p)$. Модуль p обрано достатньо великим (одне з чисел Мерсенна), щоб різниця у швидкодії алгоритмів була відчутною при тестуванні. Значення h генерується на основі наперед заданого секретного x, щоб мати еталон для перевірки правильності роботи алгоритмів.

```
print(f'Пошук x у рівнянні: {g}^x ≡ {h} (mod {p})")
# 2. Виконання алгоритму Шенкса та аналіз часу
start_time = time.time()
result_bsgs = baby_step_giant_step(g, h, p)
end_time = time.time()
```

Збір статистики: Програма фіксує точний час системи до та після виклику функції baby_step_giant_step, що дозволяє розрахувати реальні витрати процесорного часу. Аналогічний блок коду застосовується і для виклику функції brute_force, після чого отримані результати виводяться у консоль для проведення порівняльного аналізу.

```
print(f"\n[Baby-step Giant-step]")
  print(f"Знайдений x: {result_bsgs}")
 print(f"Час виконання: {end_time - start_time:.6f} секунд")
```

5. Виконання алгоритму повного перебору (опціонально для порівняння)

```
print(f"\n[Brute-force (Повний перебір)]")
start_time_bf = time.time()
  result_bf = brute_force(g, h, p)
end_time_bf = time.time()
    print(f"Знайдений x: {result_bf}")
 print(f"Час виконання: {end_time_bf - start_time_bf:.6f} секунд"
if __name__ == "__main__":
 main()
```

Нижче наведено порівняльний аналіз алгоритмів brute-force та baby-step giant-step для задачі дискретного логарифмування (табл.1). Реалізацію виконано засобами Python.

Табл. 1. Результати експериментальних вимірювань

№	Baby-step Giant-step (c)	Brute-force (c)
1.0	0.079509	46.784104
2.0	1.9e-05	2.4e-05
3.0	3.258091	72.603004
4.0	0.255769	2.535251
5.0	0.073093	403.586611

Аналіз результатів показує суттєву перевагу алгоритму Шенкса над повним перебором при збільшенні розмірності параметрів. Використання хеш-таблиць у Python забезпечує швидкий пошук колізій та скорочення часу обчислень.

Висновки

У роботі виконано всебічний аналіз задачі дискретного логарифмування, яка становить основу криптографічної стійкості сучасних асиметричних систем, зокрема протоколу Діффі–Геллмана, криптосистеми Ель-Гамаля та методів на основі еліптичних кривих. Показано, що класичний підхід

повного перебору (brute-force) є надзвичайно неефективним і практично непридатним для великих значень модуля. Натомість використання алгоритму baby-step giant-step (алгоритму Шенкса) значно підвищує ефективність обчислень завдяки принципу зустрічного руху, що суттєво скорочує простір пошуку. Алгоритм baby-step giant-step є значно ефективнішим за brute-force та доцільним для навчального моделювання криптоаналітичних задач.

Порівняльний аналіз також підтвердив перевагу алгоритму baby-step giant-step над методом повного перебору. Використання Python є доцільним завдяки простоті реалізації та високій швидкодії вбудованих засобів роботи з великими числами. Перспективою подальших досліджень є аналіз Pollard rho та паралельних реалізацій.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Кормен Т. Г., Лейзерсон Ч. Е., Рівест Р. Л., Стайн К. Вступ до алгоритмів / пер. з англ. — Київ : К.І.С., 2019. — 1288 с. — ISBN 978-617-684-239-2.
2. Крак Ю. В. «Алгоритми комп'ютерної алгебри та їх застосування»: навчальний посібник. — Київ: КНУ ім. Т. Шевченка, 2022.
3. Трофименко О. Г., Прокоп Ю. В., Логінова Н. І. Програмування мовою Python: навчальний посібник. — Одеса: Фенікс, 2021. — 272 с.
4. Python Software Foundation. time — Time access and conversions // The Python Standard Library. — URL: https://docs.python.org/3/library/time.html
5. Statista Research Department. Volume of data/information created, captured, copied, and consumed worldwide from 2010 to 2025. — Statista, 2023. — URL: https://www.statista.com/statistics/871513/worldwide-data-created/ (Офіційне джерело для підтвердження обсягу цифрових даних у 150 Зетабайтів).
6. Villanueva J. C. How Many Atoms Are There in the Universe? // Universe Today. — URL: https://www.universetoday.com/36302/atoms-in-the-universe/ (Науково-популярне пояснення числа Еддінгтона про 10^{80} атомів).

Крупський Андрій Вікторович – студент групи 2БС–25б, факультет інформаційних технологій та комп'ютерної інженерії, Вінницький національний технічний університет, м. Вінниця, e-mail: 04-25-074@vntu.edu.ua

Тютюнник Оксана Іванівна – доцент кафедри вищої математики, Вінницький національний технічний університет, м. Вінниця, e-mail: tyutyunnik@vntu.edu.ua

Krupskyi Andrii V. – student of group 2BS – 25b, Faculty of Information Technologies and Computer Engineering, Vinnytsia National Technical University, Vinnytsia, e-mail: 04-25-074@vntu.edu.ua

Tiutiunnyk Oksana I. – Associate Professor, Department of Higher Mathematics, Vinnytsia National Technical University, Vinnytsia, e-mail: tyutyunnik@vntu.edu.ua