Національний технічний університет України «Київський Політехнічний Інститут імені Ігоря Сікорського»

Комп’ютерний практикум №2 Застосування алгоритму дискретного логарифмування

Виконав:
студент групи ФІ-94
Маринін Іван Павло Ігорович

Перевірив:
Якимчук Олексій Петрович

Київ – 2022

Зміст

1. Мета комп’ютерного практикуму. 3
2. Постановка задачі та варіант завдання. 4
3. Хід роботи, опис труднощів, що виникали під час виконання завдання, та шляхи їх подолання. 5
3.1 Хід роботи: 5
3.2 Опис труднощів та шляхи їх подолання: 5
4. Результати дослідження та результати продуктивності роботи програмної реалізації алгоритмів. 6
5. Аналіз отриманих результатів. 7
5.1 Пошук дискретного логарифма: 7
5.2 Час роботи: 7
6. Висновки. 8

1. Мета комп’ютерного практикуму.

Ознайомитися з алгоритмом дискретного логарифмування Сільвера-Поліга-Геллмана. Виконати практичну реалізацію цього алгоритму. Знайти переваги, недоліки та особливості застосування даного алгоритму дискретного логарифмування. Оцінити складність роботи алгоритму.

2. Постановка задачі та варіант завдання.

Звичайний логарифм $\log_a(b)$ — це пошук розв'язку x рівняння $a^x = b$ в полі дійсних (або комплексних) чисел. Розглянемо G — довільну скінченну мультиплікативну абелеву групу, а α і β є елементами цієї групи, тоді, аналогічно до звичайного, дискретний логарифм числа β за основою α ($\log_\alpha(\beta)$) — це ціле число x таке, що $\alpha^x = \beta$.

Ефективного методу для обчислення дискретного логарифма в загальному випадку не існує. Проте вони можуть ефективно обчислюватись в деяких особливих випадках. Деякі криптосистеми з відкритим ключем базуються на складності обчислення дискретного логарифма, наприклад, криптосистема Ель-Гамаля.

Задача дискретного логарифмування формулюється так: маючи просте число p, число α — генератор групи $Z * p$, число β — елемент $Z * p$, знайти число x, $0 \leq x \leq p - 2$, таке, що $\alpha^x = \beta \bmod p$. Найпростіший розв'язок цієї задачі — повний перебір всіх елементів $Z * p$. Складність при такому підході: O(n).

3. Хід роботи, опис труднощів, що виникали під час виконання завдання, та шляхи їх подолання.

3.1 Хід роботи:

1. Написано програму, що реалізовує алгоритм Сільвера-Поліга-Геллмана для груп типу Z_p^*.

2. Встановити Docker (https://docs.docker.com/get-docker/). За допомогою команди docker run -it salo1d/nta_cp2_helper запустити допоміжну програму, яка генерує задачу пошуку дискретного логарифма. Вхідним параметром програми є просте число p — модуль кільця лишків.

3. Застосовувати реалізований алгоритм Сільвера-Поліга-Геллмана до задачі дискретного логарифма, яку формує програма з попереднього пункту, почергово зі збільшенням порядку p. Потрібно пам'ятати, що вхідний параметр p повинен бути простим числом. У випадку, якщо допоміжна програма не справляється зі завданням генерації задачі, або ваша реалізація не справляється з розв'язком задачі за відведений час, зупинити збільшення вхідного параметра p.

Приклад збільшення параметру p:

1. p = 3
2. p = 17
3. p = 157
4. . . .

4. Оформлено звіт до комп'ютерного практикуму.

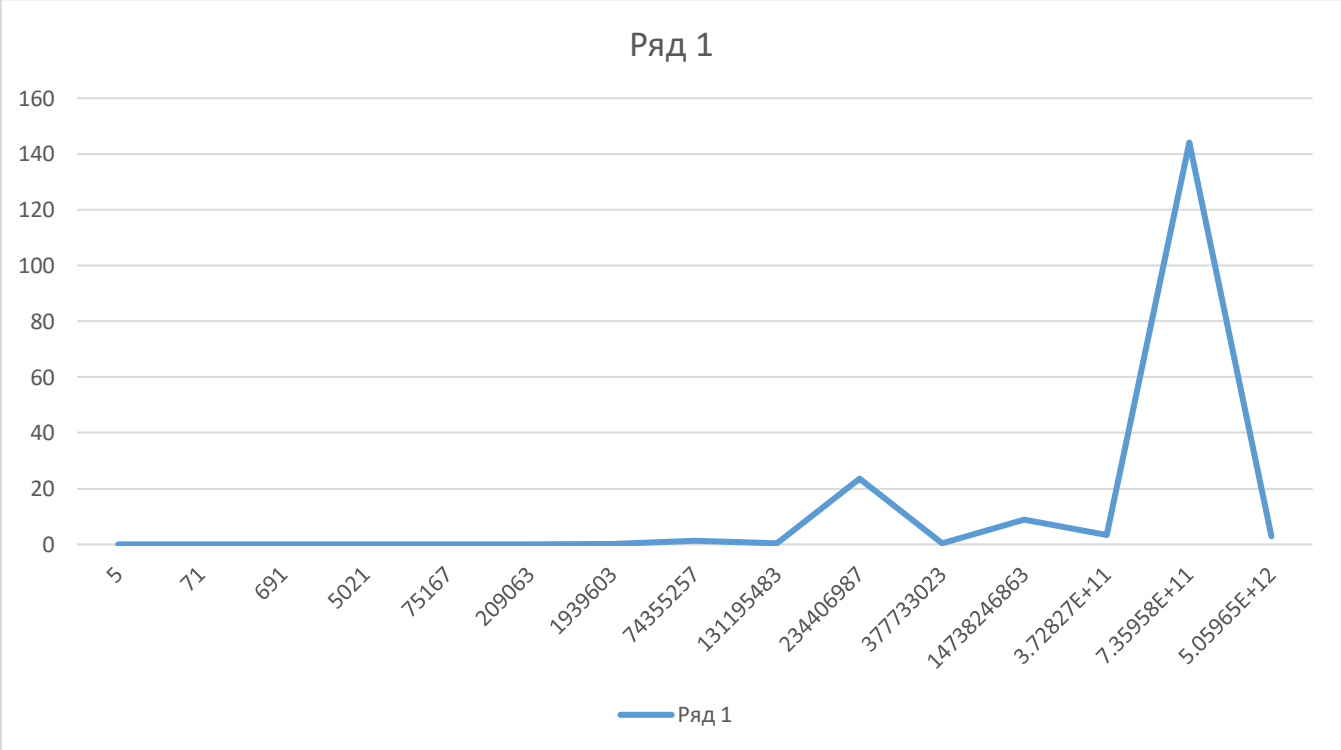
3.2 Опис труднощів та шляхи їх подолання:

Алгоритм не викликав жодних труднощів у програмній реалізації. Проте виникли наступні проблеми з платформою Docker: на моєму власному ноутбуці не вийшло встановити 'докер' тому що у мене версія Windows, яка не підтримується програмою. Далі я використав інший ПК, версія Windows якого вже була підходяща, але 'докер' все одно не працював коректно. Було проведено ряд дій, що могли би усунити потенційні проблеми, як от надання дозволу для віртуалізації, умікнення та оновлення для WSL 2 та інші, які не дали жодного результату. 'докер' стартував на 2-3 сенкунди, а потім зупинявся. Аналогічні дії були проведені на третьому ноутбуці, але результат був тим же (на другому і третьому встановлені ліцензійовані ОС Windows). Зрештою, я звернувся за допогою до одногрупників, які знаходили необхідні мені α та β за простим модулем p. Вони надсилали мені ці дані, я вносив їх у власну програму, знаходив логарифм та надсилав їм для перевірки 'докером'.

4. Результати дослідження та результати продуктивності роботи програмної реалізації алгоритмів.

α	β	p	$\log_\alpha(\beta) \pmod p$	Час роботи
2	1	5	0	0:00:00
67	23	71	45	0:00:00
300	295	691	196	0:00:00
1645	2780	5021	2285	0:00:00
13634	30224	75167	66113	0:00:00
181331	71436	209063	173638	0:00:00
1583099	914798	1939603	722661	0:00:00.234399
61655742	67277301	74355257	23294097	0:00:01.254940
101409556	12244691	131195483	127874934	0:00:00.390667
76384303	152159776	234406987	56596411	0:00:23.623820
259912333	329337424	377733023	238293124	0:00:00.296908
12373849077	13885045634	14738246863	14670690264	0:00:08.842629
343639941376	14044004075	372826589809	173118122672	0:00:03.406558
526008406447	44734954454	735958035331	249487526383	0:02:24.108465
1495189482107	1394538985924	5059646485291	1795108196176	0:00:03.015945
12953794210505	16073085221671	44769087922303	-	> 5 хв
32922756611615	39984248570848	73537252713889	-	> 5 хв
70764157997	25955962093	73898126689	-	> 5 хв
2689132274736	782066633920	3133394330377	-	> 5 хв

Таблиця 1



Графік 1

5. Аналіз отриманих результатів.

5.1 Пошук дискретного логарифма:

Судячи з отриманих результатів можна підтвердити, що зі збільшенням значення модуля за яким шукається дискретний логарифм, збільшується і сам час роботи алгоритму (ніби до цього були й сумніви☺). Проте, іноді буває, що й числа однакової розрядності мають доволі відчутну різницю в часі. Наприклад, p = 234406987 та p = 377733023: 23.623820 та 0.296908 секунд відповідно. Також має місце наступне: логарфм за модулем числа одної розрядності знаходить за декілька секунд, а за модулем числа меншої розрядності результат на видає впродовж 5 хвилин. Це означає, що не лише розрядність(довжина) числа має значення, а й його сутність (детальніше у розділі 5.2).

Якщо критичною точкою для оцінки максимального порядку вхідного параметра p, при якому процес побудови задачі і її розв'язання відбувався, вважати роботу алгоритму у 5 хвилин, то у моєму випадку параметр p може бути довжию до 14 знаків.

5.2 Час роботи:

Оцінка складності: найбільш затратними виходять таблиці(там, де для кожного дільника шукати r). Я реалізував ці таблиці у вигляді списку, елементами якого є списки. Відповідно, якщо число розкладається на небагато, зате великих дільників, то кількість 'ерок' для усіх дільників буде значно більшою, ніж якщо би це ж число розкладалося на приблизно однакові значення. Наприклад, візьмемо числа 2*79=158 та 2^6*3=192. Хоча 158 < 192, але кількість 'ерок' для 158 відносно 192 більша у 16 разів. А якщо взяти 2^{100}? Кількість 'ерок' буде лише 2, а 158 для 2^{100} можна вважати за матеріальну точку. Тому на швидкість роботи даного алгоритму дуже сильно впливає те, який канонічний розклад має число: чим менше дільників і чим більша степінь кожного з них, тим краще.

Іноді буває таке, що для одного числа видає різні результати в часі. Вони не є критично різними, тобто відхиляються менш, ніж на третину від середнього значення декількох замірів для одного числа. Чому так, я не маю гадки. Можливо, розгадка в можливостях комп'ютера, і, що при різних запусках, він видає різну к-сть ресурсів.

Алгоритм Сільвера-Поліга-Геллмана найбільш ефективний, коли:

• канонічний розклад n завчасно відомий;

• всі прості дільники числа n невеликі: $p_i < (\log_2 n)^{c_1}$, в такому випадку складність алгоритму поліноміальна $O((\log_2 n)^{c_2})$, де c1, c2 – додатні константи. В інших випадках (навіть якщо хоча б один дільник числа n великий) алгоритм має експоненційну складність.

!Disclaimer: для побудови змістовніших висновків та припущень необхідно провести більше дослідів та вдосконалити, де можливо, програмну реалізацію алгоритму.

6. Висновки.

У рамках комп’ютерного практикуму було розглянуто алгоритм Сільвера-Поліга-Геллмана для задачі дискретного логарифмування. Реалізовано програму, що виконує роботу методу, за допомогою мови програмування Python у середовищі розробки PyCharm. Для побудови задачі дискреного логарфму використовувалось стороннє програмне забезпечення, доступ до якого отримано за допомогою платформи Docker. Проведено серію дослідів для простих чисел довжиною від 1 до 14 символів разом із замірами часу роботи алгоритму. Наведено аналіз отриманих даних.