НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ім. Ігоря СІКОРСЬКОГО»
ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ

Звіт за темою:
«Застосування алгоритму дискретного логарифмування»

Виконав студент
групи ФІ-94
Кріпака Ілля

Київ — 2022

1 Мета практикуму

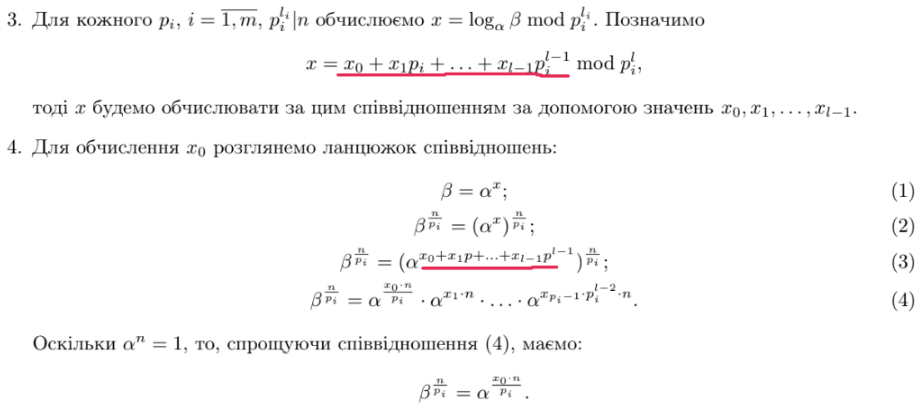
Практично ознайомитися із алгоримом дискретного логарифмування Сільвера-Поліга-Геллмана, реалізувати зазначений метод. Практично оцінити складність роботи алгоритму.

1.1 Постановка задачі та варіант

Треба реалізувати	**Зроблено**
Алгоритм Сільвера-Поліга-Геллмана	Алгоритм Сільвера-Поліга-Геллмана ✓

2 Хід роботи/Опис труднощів

На початку реалізації практикуму трохи виникли незначні проблеми із розумінням того, як треба зберігати степені для обчислення x_i (от саме пункти 3., 4. у самому алгоритмі), але все обійшлося, як тільки почав їх реалізовувати. :)



3. Для кожного p_i, $i = \overline{1,m}$, $p_i^{l_i} | n$ обчислюємо $x = \log_\alpha \beta \bmod p_i^{l_i}$. Позначимо

$$x = x_0 + x_1 p_i + \ldots + x_{l-1} p_i^{l-1} \bmod p_i^l,$$

тоді x будемо обчислювати за цим співвідношенням за допомогою значень $x_0, x_1, \ldots, x_{l-1}$.

4. Для обчислення x_0 розглянемо ланцюжок співвідношень:

$$\beta = \alpha^x; \quad (1)$$

$$\beta^{\frac{n}{p_i}} = (\alpha^x)^{\frac{n}{p_i}}; \quad (2)$$

$$\beta^{\frac{n}{p_i}} = (\alpha^{x_0 + x_1 p + \ldots + x_{l-1} p^{l-1}})^{\frac{n}{p_i}}; \quad (3)$$

$$\beta^{\frac{n}{p_i}} = \alpha^{\frac{x_0 \cdot n}{p_i}} \cdot \alpha^{x_1 \cdot n} \cdot \ldots \cdot \alpha^{x_{p_i - 1} \cdot p_i^{l-2} \cdot n}. \quad (4)$$

Оскільки $\alpha^n = 1$, то, спрощуючи співвідношення (4), маємо:

$$\beta^{\frac{n}{p_i}} = \alpha^{\frac{x_0 \cdot n}{p_i}}.$$

Рис. 1: Пункти 3., 4. у алгортимі

Основна проблема у мене виникла із тим, як треба розв'язувати рівняння. Так, треба було застосувати *Китайську Теорему про Лишки*, але від початку неправильно реалізував обрахунок M_i елементів, тому шукав, чому при обчисленні оберненого елементу на вхід приходить число 0, адже для 0 не існує його.

Також були труднощі не стільки у самому алгоритмі, а скільки у його реалізації із типами чисел у алгоритмах факторизації. Саме там, ще не знав про реалізацію великих чисел, тому там усі числа упираються у тип $u128$ (тип, що містить беззнакові числа довжини 128 біт). Тому, у цьому практикумі прийшлося підлаштовуватися під цей недолік.

Але, не зважаючи на деякі проблеми, на мою думку, цей алгоритм дискретного логарифмування виявився набагато легшим, ніж Брілхарта Морісона у попередньому практикумі. Навчений попереднім досвідом, одразу використовував бібліотеку великих чисел у rust — *num-bigint*.

3 Результати дослідження

У результаті маємо, що:

1. Алгоритм Сільвера-Поліга-Геллмана — дуже ефективниий для вирішення *dlp*, саме для чисел, які розкладаються на багато малих дільників, доприкладу:

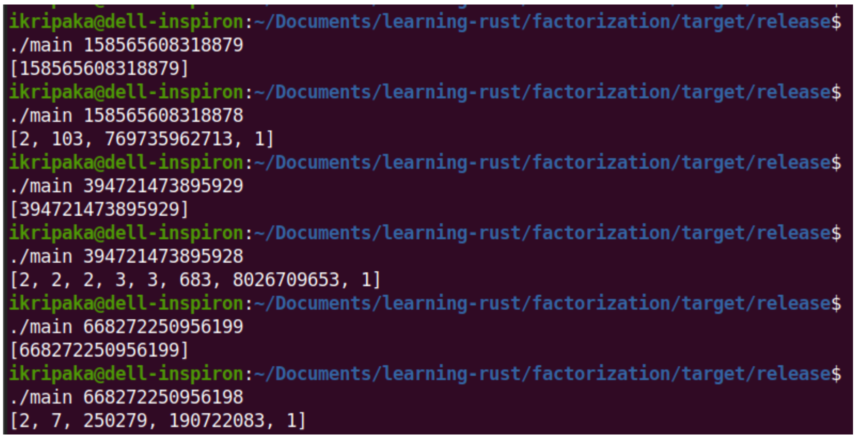


Рис. 2: Приклад поганих чисел для Сільвера-Поліга-Геллмана

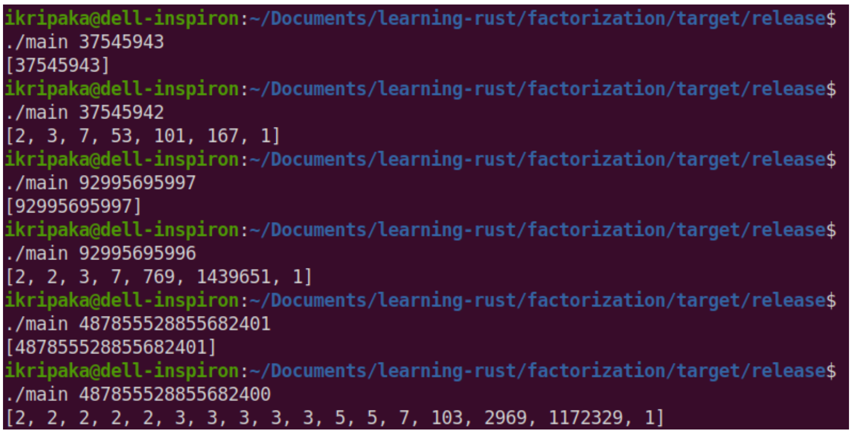


Рис. 3: Приклад добрих чисел для Сільвера-Поліга-Геллмана

2. Проблема полягає у тому, що для формування таблички передобчислених значень треба перебрати числа, до прикладу, для $6682722509956199 = 2 * 7 * 250279 * 190722083$ — максимальна кількість ітерацій буде 190722083, що для однопоточної програми може викликати часові проблеми. На мою думку, це можна вирішити за допомогою розпаралелювання перебору цих усіх значень.

4 Продуктивність

Після обрахунків вийшло наступне:

Числа(a, b, n)	**dlp алгоритм**	**Розклад числа** $(n-1)$
(44, 1229, 6277)	391ns	[2, 2, 3, 523]
(70425, 4498, 98929)	82.973186ms	[2, 2, 2, 2, 3, 3, 3, 229]
(79791, 7727, 106621)	264.272183ms	[2, 2, 3, 5, 1777]
(5347363, 1393557, 5794511)	76.10528318s	[2, 5, 579451]
(32012782, 4740726, 37545943)	250.110651ms	[2, 3, 7, 53, 101, 167]
(431663093, 527715071, 633337597)	16.348184705s	[2, 2, 3, 3, 3, 47, 124771]
(5710238076, 5213445017, 6390644171)	250.787527ms	[2, 5, 41, 79, 1033, 191]
(62169854910, 86077798599, 92995695997)	185.391740656s	[2, 2, 3, 7, 769, 1439651]
(71428636448, 180199541342, 584842224173)	$\geqslant 13min$	[2, 2, 41, 3566111123]
(4313558325450, 6380632412530, 9577259708671)	$\geqslant 11min$	[2, 3, 3, 5, 31, 401, 8560373]
(13637999129366, 2111433979175, 15710549366693)	$\geqslant 20min$	[2, 2, 3670967, 1069919]
(187165199375552, 22166621073359, 336305574949727)	$\geqslant 4min$	[2, 383, 193, 7717, 294781]
(2629622656603408, 3806920734785279, 3821293645373207)	69.077615454s	[2, 36493, 127711, 409961]
(31359535267603010, 3969189589541717, 91983358119398483)	404.263866988s	[2, 11, 59, 4637, 4547, 3361031]
(107477375094958706, 438527732551443546, 487855528855682401)	139.449256575s	[2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 5, 5, 7, 103, 2969, 1172329]

Оцінка продуктивності:

1. Моя реалізація алгоритму має обмеження на вхід чисел $< 340282366920938463463374607431768211455$ (39 значні числа).

2. Практично межа проходить на 18, 19, ... значних числах, адже там уже іде розклад на числа > 1000000, що уже важко перебрати.

5 Висновки

За допомогою практикуму "Застосування алгоритму дискретного логарифмування"дізнався як вирішуються dlp задачі на вище зазначеному алгоритмі.

У результаті одержав, що Сільвер-Поліг-Геллман повинен використовуватися у парі із іншим алгоритмом, до прикладу, із BabyStep-GiantStep, або із Pollard's Kangaroo Algorithm, який активно модифікується за домогою додавання різної кількості "wild kangaroo "tame kangaroo та інших модифікацій різних відомих алгоритмів для вирішення дискретного логарифма, або спробувати реалізувати розпаралелювання передобчислень, які займають більшу кількість роботи алгоритму.