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

# Звіт за темою
# «Пошук канонічного розкладу великого числа, використовуючи відомі методи факторизації»

Виконав студент
групи ФІ-94
Кріпака Ілля

Київ — 2022

# 1 Мета практикуму

Практично ознайомитися із різними методами факторизації чисел, реалізувати методи та їх порівняння. Застосувати комбінації алгоритмів факторизації для пошуку канонічного розкладу заданого числа.

## 1.1 Постановка задачі та варіант

| **Треба реалізувати** | **Зроблено** |
|---|---|
| Алгоритм перевірки числа на простоту | Імовірнісний тест Міллера-Рабіна ✓ |
| Метод пробних ділень | Метод пробних ділень✓ |
| $\rho$-метод Полларда | $\rho$-метод Полларда ✓ |
| Метод Брілхарта-Морісона/Метод Померанця | Метод Брілхарта-Морісона ✓ |

# 2 Хід роботи/Опис труднощів

На початку реалізації практикуму, зробив алгоритм перевірки числа на простоту, методу пробних ділень, $\rho$-методу Полларда, адже, на мою думку вони були найпростіші. Але, на жаль, відтягував реалізацію до кінця практикуму, Брілхарт-Морісон вселяв жах у мене. Зараз його таки зробив :)

Основні труднощі виникали у піднесенні числа до певного степеня, адже числа великі та ще й не поміщалися у стандартні типи тому спробував вирішити проблему за допомогою схеми Горнера - допомогло, але для Брілхарта-Морісона уже було переповнення. Тому прийшлося шукати сторонню бібліотеку для rust. Допомогла саме ця бібліотека(якщо так можна називати) *num-bigint*. Саме там знайшов відповідне піднесення до степеня за модулем та без нього.

Також виникли труднощі із роботою самого Брілхарта-Морісона, бо там реалізований пошук розв'язків системи за допомогою перебору, і це давало відчутний приріст у часі роботи. До прикладу, для числа $9172639163 = 91753 * 99971$ факторна база уже буде із 37 чисел, що погано піддається перебору, адже перебираю усі можливі бітові комбінації від 1 до $(2^{37+1} - 1)$, тобто від 0000..001 до 1111..111, де 1 означає, що треба взяти відповідний розклад числа і додати його до загального вектора, де уже буде видно по заверешенню обходу бітового числа чи є розв'язком певна комбінація веторів за $mod 2$. Саме на цьому числі алгоритм буде працювати $> 27$ хвилин.

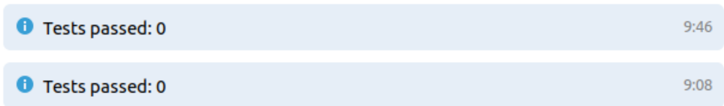


Рис. 1: Приклад орієнтовоного часу роботи Брілхарта-Морісона для великих чисел.

# 3 Результати дослідження

У результаті маємо, що:

1. Імовірнісний тест Міллера-Рабіна — дуже ефективниий алгоритм перевірки на простоту за допомогою означення *псевдопростоти за Ойлером*, що дає нам ймовірність помилки

$\dfrac{1}{4^k}$(краще чим у Соловея-Штрассена — $\dfrac{1}{2^k}$, де $k-$кількість повторень алгоритму), але треба звернути увагу на ефективну реалізацію піднесення до степеня за модулем.

2. Алгоритм пробних ділень — є ефективним для дуже малих чисел, до прикладу, для $\leqslant 47$. Адже він працює за допомогою перебору чисел до $\sqrt{n}, n-$ вхідне число, що не є ефективним для набагато більших чисел.

3. $\rho$-метод Полларда — є дуже ефективним для розкладу чисел, де прості дільники знаходяться не далеко від одного, наприклад, як $8633 = 97 * 87$. Адже цей алгоритм побудований за допомогою обчислення орбіти, яка у свою чергу обчислюється псевдорандомними функціями типу $x^2 + 1$..

4. Метод Брілхарта-Морісона — належить зовсім іншому класу алгоритмів, адже побудований на факторній базі, яка дозволяє розкладати дуже великі числа (але у мене не вийшло ефективно його реалізувати :( ). Саме тут критичним є перебір усіх можливих варіантів розв'язку, дуже впливає на сам час роботи.

```
Factorizing number: 44729396957
Input number in trial division: 44729396957
Trial division divider: 7
Input number in rho pollard: 6389913851
Rho pollard dividers: 1307
Input number in brillhart morrison: 4888993
Brillhart morrison dividers: 10039, 487
Result: [7, 1307, 10039, 487]
```

Рис. 2: Приклад факторизації усіма алгоритмами числа 44729396957.

```
Factorizing number: 37007068943
Input number in trial division: 37007068943
Trial division divider: 13
Input number in rho pollard: 2846697611
Rho pollard dividers: 1051
Input number in brillhart morrison: 2708561
Brillhart morrison dividers: 269, 10069
Result: [13, 1051, 269, 10069]
```

Рис. 3: Приклад факторизації усіма алгоритмами числа 37007068943.

## 4 Продуктивність

Після обрахунків вийшло наступне:

| **Числа** | **$\rho$-метод Полларда** | **Брілхарт Морісон** |
|---|---|---|
| 9172639163 | 549.089µs | $\geqslant$ 27min |
| 9172639163862795 | 1.862472ms | — |
| 8627969789 | 514.899µs | — |
| 8937716743 | 875.838µs | — |
| 278874899 | 350.56µs | — |
| 99400891 | 180.081µs | — |
| 116381389 | 195.414µs | — |
| 4252083239 | 477.629µs | — |
| 6633776623 | 272.11µs | — |
| 227349247 | 138.315µs | — |
| 3568572617 | 327.826µs | — |
| 25511 | 48.185µs | 10.984851ms |
| 3973573 | 90.012µs | 28.210255ms |
| 10528813 | 202.815µs | 42.002997ms |
| 4248311 | 146.742µs | 28.151853ms |
| 10252159 | 93.764µs | 66.936453ms |
| 5746331 | 107.961µs | 35.10992ms |
| 2708561 | 27.814µs | 62.165807ms |
| 10169711 | 152.216µs | 32.830382ms |
| 4888993 | 85.48µs | 27.730359ms |
| 12404297 | 107.104µs | 35.126146ms |
| 11587117 | 86.53µs | 79.249916737s |

**Оцінка продуктивності**:

1. $\rho$-метод Полларда — працює для $\leqslant 16$ значних чисел, для 17 уже треба чекати.

2. Метод Брілхарта-Морісона — на маленьких числах застосовується, що із двома великими дільниками добре, а от із 4 - уже виникають проблеми. Погано працює уже для $> 11587117$.

## 5 Висновки

За допомогою реалізації практикуму "Пошук канонічного розкладу великого числа, використовуючи відомі методи факторизації", дізнався на практиці, як працюють алгоритми факторизації. $\rho$-метод Полларда вийшов досить ефективним, працює навіть краще чим Брілхарта Морісона. Рекомендую при реалізації цих алгоритмів самотужки заздалегідь визначитися із тим, як: реалізувати обчислення елементарних операцій, взяття по модулю, піднесення до степеня, зберігання великих чисел. Бо, як переконався на власному досвіді, все вищезазначене дуже сильно впливає на ефективність реалізації. А от криптографічних цілях, на мою думку, краще реалізувати алгоритм Померанця та інші ефективніші алгоритми базовані на факторній базі.