ЛАБОРАТОРНА РОБОТА No3

з курсу

ТЕОРЕТИКО-ЧИСЛОВІ АЛГОРИТМИ В КРИПТОЛОГІЇ

Реалізація та застосування алгоритму дискретного логарифмування index-calculus

Виконав: Приходько Юрій ФБ-12

1. Мета роботи

Ознайомлення з алгоритмом дискретного логарифмування index-calculus. Програмна реалізація цього алгоритму та визначення його переваг, недоліків та особливостей застосування. Практична оцінка складності роботи та порівняння різних реалізацій цього алгоритму.

2. Хід роботи

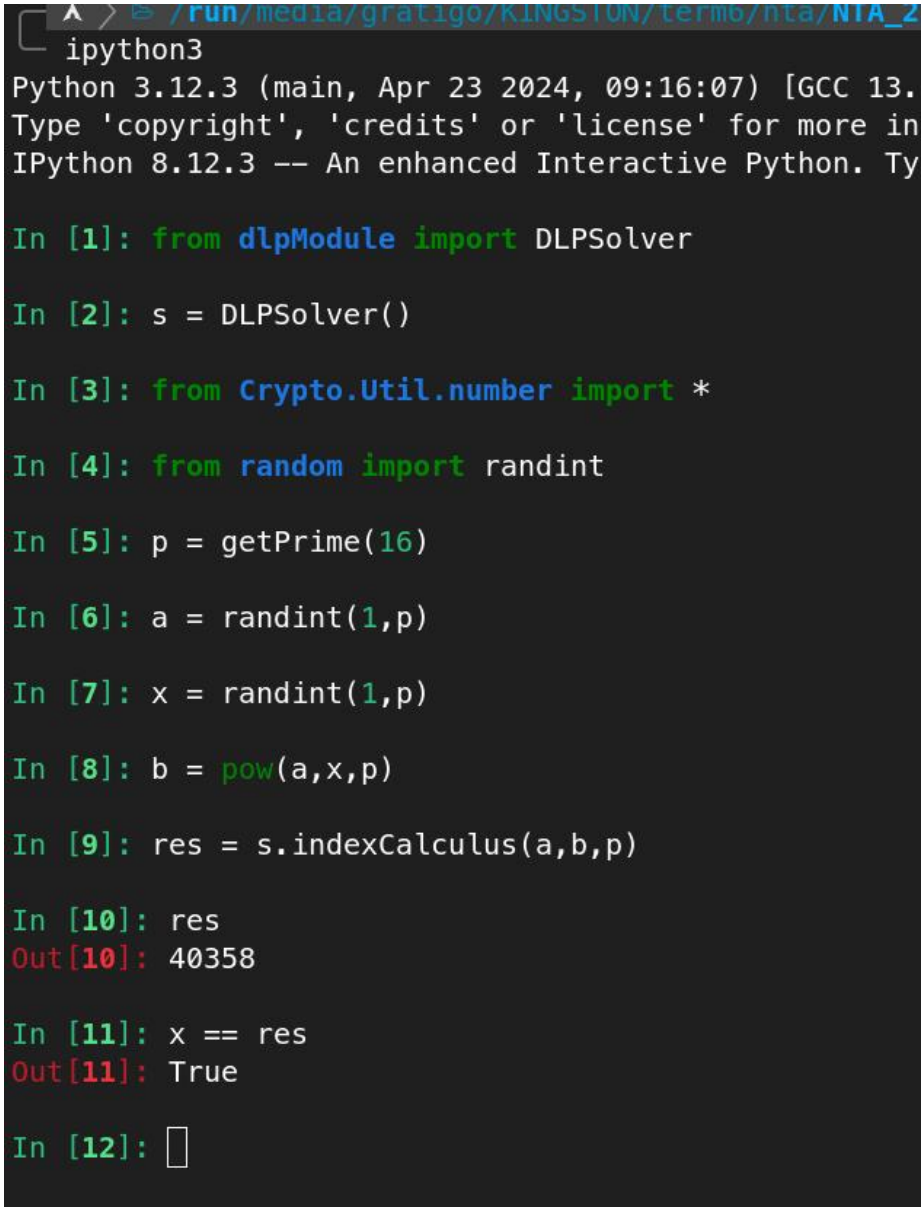
Написати програму, що реалізовує алгоритм index-calculus для груп типу Zp* без розпаралелювання етапу побудови системи лінійних рівнянь.

В результаті виконання лабораторної роботи для підвищення ефективності алгоритмів було вирішено написати код програми низько рівневою мовою C, з використанням функцій для пошуку канонічного розкладу числа з попереднього комп’ютерного практикума.

В результаті написання була отримана бібліотека а також інтерфейс виклику її функції мовою python що дозволяють знайти дискретний логарифм за заданими параметрами за допомогою алгоритма index-calculus.

Для роботи з великими числами була використана бібліотека gmp.

Приклад виконання:



3. Написати програму, що реалізовує алгоритм index-calculus для груп типу Zp* з розпаралелюванням етапу побудови системи лінійних рівнянь. Процес розпаралелювання необхідно будувати, зважаючи на технічні особливості обчислювальної техніки, на якій буде запускатися ця програма.

В результаті виконання цього завдання, шляхом розпаралелення на потоки була удосконалена частина алгоритму де відбувається генерація системи лінійних рівнянь, а також остання частина безпосередньо пошуку дискретного алгоритму за умовою, вже після вирішення системи рівнянь.

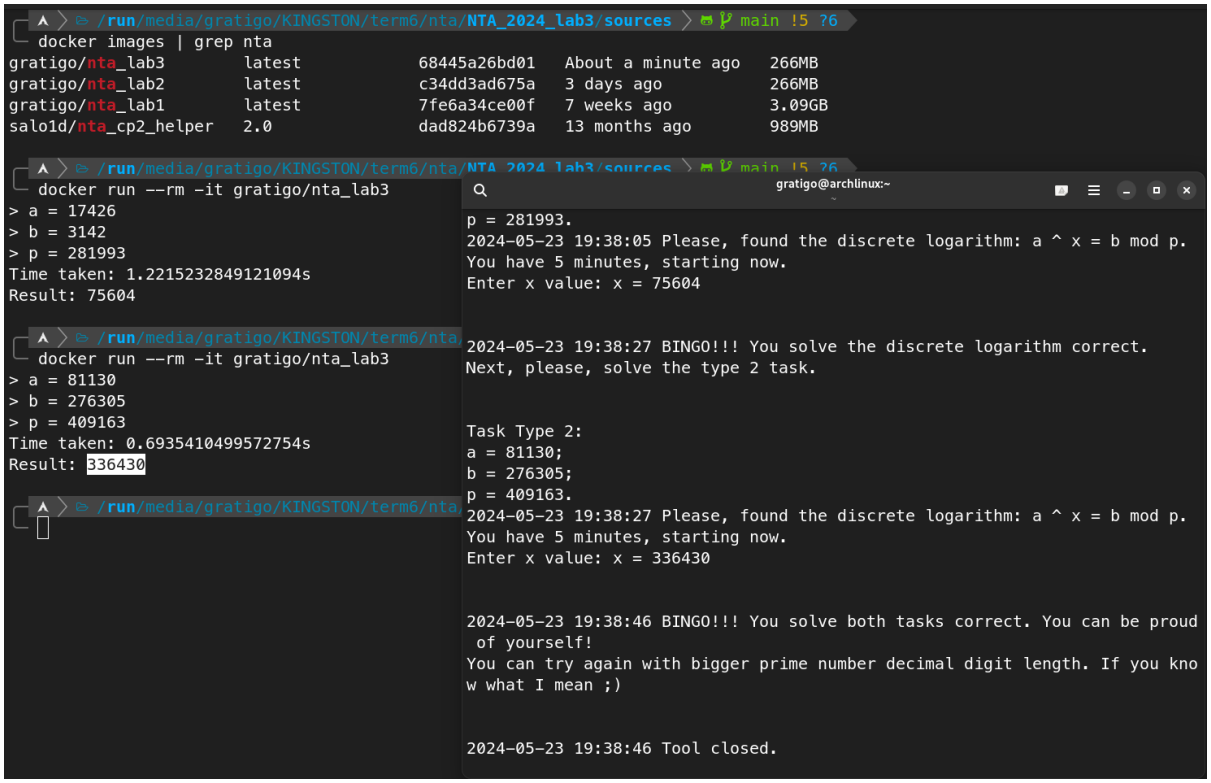
Для створення потоків були зроблені окремі структури передачі даних, використана бібліотека POSIX Threads, контроль race conditions організовано шляхом використання mutex.

Приклад роботи:

```
In [1]: from dlpModule import DLPSolver

In [2]: s = DLPSolver()

In [3]: from Crypto.Util.number import *

In [4]: from random import randint

In [5]: p = getPrime(16)

In [6]: a = randint(1,p)

In [7]: x = randint(1,p)

In [8]: b = pow(a,x,p)

In [9]: res = s.indexCalculus(a,b,p)
Generating equations
equations generated
equations solved
starting solution search
Thread 0 started with random number: 32875
Thread 2 started with random number: 26509
Thread 4 started with random number: 30693
Thread 1 started with random number: 6108
Thread 5 started with random number: 46851
Thread 3 started with random number: 7142
Thread 2 found solution 7473
solution found

In [10]: pow(a,res,p) == b
Out[10]: True
```

4. Створити Docker image, що містить програму, яка містить імплементацію алгоритму index-calculus з кроку 2 (без розпаралелювання), та інструкцію щодо запуску та користування програмою в цьому Docker image.

Створимо docker-image з режимом роботи алгоритму без потоків. Перевіримо його роботу, додамо на докерхаб.



5. Застосувати дві реалізації алгоритму index-calculus (з пунктів 2 та 3) до самостійно сформованих задач дискретного логарифма, поступово збільшуючи порядок числа p. Рекомендується починати з порядку 3. Для кожної задачі заміряти час роботи обох реалізацій. Зупинити збільшення порядку числа p, коли час роботи хоча б однієї з реалізації перевищує певне значення, наприклад 5 хвилин. Мінімальне обмеження часу роботи – 3 хвилини. Для формування задач дискретного логарифма можна використовувати допоміжну програму з КП #2.

Почнемо збільшувати розмір числа, роблячи по 4 заміри на кожен для обох варіантів алгоритму. Для генерації використаємо допоміжну програму з КП2. Кількість потоків у другому варіанті - 10.

Отримуємо наступну таблицю

length	Без розпаралелювання	З розпаралелюванням

3	0.00165 0.00207 0.00238 0.00341	0.00253 0.00215 0.00276 0.00314
4	0.00473 0.00185 0.00557 0.00347	0.00343 0.00282 0.00352 0.00247
5	0.01433 0.00705 0.00447 0.00574	0.00288 0.00439 0.00414 0.00356
6	0.01435 0.02027 0.00933 0.01871	0.00507 0.00475 0.00595 0.00460
7	0.03279 0.03307 0.04073 0.03388	0.01078 0.01139 0.01336 0.01108
8	0.07179 0.08876 0.04747 0.06550	0.01848 0.01487 0.01986 0.01882
9	0.09186 0.15232 0.06550 0.13683	0.02067 0.03517 0.07246 0.04414
10	0.36153 0.29587 0.23915 0.35280	0.09956 0.09638 0.08625 0.10446
11	1.01220 0.99793 1.14724 0.73797	0.20643 0.16282 0.11565 0.09066
12	1.87988 1.71761 2.38277 1.80439	0.36182 0.40554 0.30368 0.44133
13	4.60801 4.89918	0.97221 1.01574

	4.48260 5.57267	0.76716 1.06465
14	11.07779 6.48839 9.18086 9.05675	2.23604 1.92885 1.94133 2.41999
15	17.94169 24.75508 27.36494 27.31568	3.88390 4.52734 7.02415 3.23438
16	41.95163 45.65852 70.80786 70.95171	11.72778 12.39318 7.12883 12.06618

Досліди були зупинені на довжині 16 адже генерація завдань з числами більшого розміру займає багато часу.

Окремо згенеровано та вирішено для 48bit – 3.81138 секунд

```
└─ ./t.py
Bit Length of p: 48
Input: a = 102339477859829, b = 159211252402714, p = 246661899590363
Solution: x = 97998815258559
Starting algorithm
Generating equations
Thread 2 started with seed 3433141272000
Thread 5 started with seed 8582853180000
Thread 0 started with seed 0
Thread 1 started with seed 1716570636000
Thread 3 started with seed 5149711908000
Thread 4 started with seed 6866282544000
Thread 6 started with seed 10299423816000
Thread 9 started with seed 15449135724000
Thread 7 started with seed 12015994452000
Thread 8 started with seed 13732565088000
equations generated
equations solved
starting solution search
Thread 4 started with random number: 21900534622317
Thread 0 started with random number: 159534088683627
Thread 3 started with random number: 59272247638376
Thread 2 started with random number: 232476110272983
Thread 7 started with random number: 52875382313332
Thread 5 started with random number: 117650482582937
Thread 8 started with random number: 124318553346557
Thread 1 started with random number: 147741411433088
Thread 9 started with random number: 8085700649998
Thread 6 started with random number: 214362228631882
Thread 5 found solution 97998815258559
solution found
Time passed: 3.81138014793396
Obtained result: res = 97998815258559; Validated: True
```

52bit – 13.52067 секунд

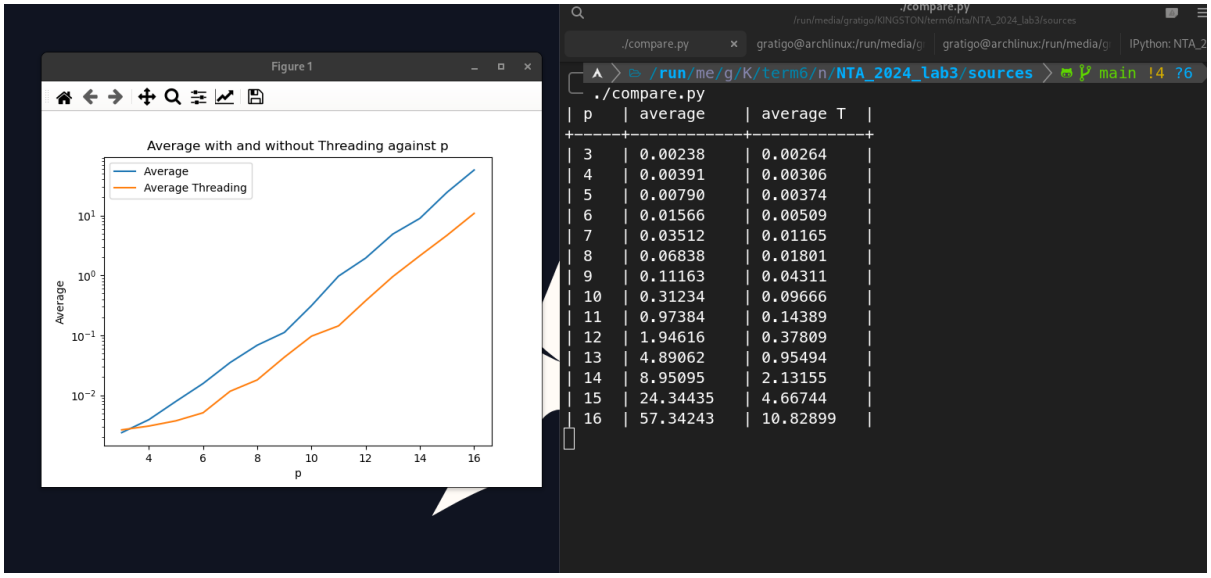
```
└ ./t.py
Bit Length of p: 52
Input: a = 367661998774979, b = 233076651532203, p = 3175507084053289
Solution: x = 1648681688300932
Starting algorithm
Generating equations
Thread 1 started with seed 1716570738000
Thread 3 started with seed 5149712214000
Thread 5 started with seed 8582853690000
Thread 8 started with seed 13732565904000
Thread 2 started with seed 3433141476000
Thread 0 started with seed 0
Thread 4 started with seed 6866282952000
Thread 6 started with seed 10299424428000
Thread 9 started with seed 15449136642000
Thread 7 started with seed 12015995166000
equations generated
equations solved
starting solution search
Thread 4 started with random number: 244490818871144
Thread 0 started with random number: 1561801328781121
Thread 7 started with random number: 2872400824022764
Thread 6 started with random number: 1323160084942032
Thread 2 started with random number: 2428746424127301
Thread 8 started with random number: 1970387800192706
Thread 3 started with random number: 2346628015360387
Thread 5 started with random number: 1558703541208028
Thread 1 started with random number: 216554958974318
Thread 9 started with random number: 1214408218875463
Thread 6 found solution 1648681688300932
solution found
Time passed: 13.520674467086792
Obtained result: res = 1648681688300932; Validated: True
```

56bit - 24.62912 секунд

```
└ ./t.py
Bit Length of p: 56
Input: a = 29917682293103417, b = 4170291551382813, p = 38519018747782747
Solution: x = 203196890187823
Starting algorithm
Generating equations
Thread 0 started with seed 0
Thread 2 started with seed 3433141692000
Thread 1 started with seed 1716570846000
Thread 3 started with seed 5149712538000
Thread 4 started with seed 6866283384000
Thread 5 started with seed 8582854230000
Thread 6 started with seed 10299425076000
Thread 8 started with seed 13732566768000
Thread 7 started with seed 12015995922000
Thread 9 started with seed 15449137614000
equations generated
equations solved
starting solution search
Thread 0 started with random number: 8040833436581995
Thread 5 started with random number: 33634426688218177
Thread 2 started with random number: 12640805905208285
Thread 1 started with random number: 33799699672181936
Thread 7 started with random number: 36128702033015814
Thread 3 started with random number: 8186451189714591
Thread 9 started with random number: 37874630813373206
Thread 4 started with random number: 35933239757752897
Thread 6 started with random number: 32751403694092908
Thread 8 started with random number: 26023796048677376
Thread 6 found solution 203196890187823
solution found
Time passed: 24.629119396209717
Obtained result: res = 203196890187823; Validated: True
```

6. Порівняти результати часу роботи обох реалізацій алгоритму з пунктів 2 та 3. Створити візуалізації залежності часу роботи реалізації від порядку простого числа p. Пояснити результати.

Напишемо програмний код для обрахування середнього значення часу на кожній довжині вхідних чисел, виведемо графік і таблицю.



Як ми можемо спостерігати, середній час краще майже у всіх випадках у алгоритму з розпаралелюванням. Єдиний випадок коли алгоритм працює швидше без потоків це випадок на найменшій довжині числа коли нам порахувати дискретний логарифм швидше ніж створити 10 потоків)

Швидкість алгоритму який використовує потоки для генерації матриці зростає порівняно з алгоритмом без них коли збільшується довжина числа.

Якщо на довжині 3-8 алгоритм з потоками працює в 2-3 рази швидше то на великій довжині числа спостерігається різниця в швидкості в 5-6 разів.

Висновки:

Вході виконання лабораторної роботи я ознайомився з алгоритмом дискретного логарифмування index-calculus, та створив його програмну реалізацію за допомогою мови C. Також створена програма була модифікована для виконання процесу генерації СЛР на декількох незалежних потоках що в результаті привело до значного пришвидшення роботи алгоритму.

Максимальний розмір числа для якого було вирішено DLP - 56біт, далі алгоритм працює довше 3хв.

Загалом можна зробити висновок про надзвичайну ефективність алгоритму index-calculus для розв’язку DLP, так його можна побудувати з урахуванням розпаралелення і при достатньому об’ємі пам’яті та потужності процесора він здатний швидко факторизувати достатньо великі числа.