НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

НАВЧАЛЬНО-НАУКОВИЙ ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ

КОМП'ЮТЕРНИЙ ПРАКТИКУМ № 1

з дисципліни:

«МЕТОДИ РЕАЛІЗАЦІЇ КРИПТОГРАФІЧНИЇ МЕХАНІЗМІВ»

Вибір та реалізація базових фреймворків та бібліотек

варіант 1А

Виконала:
Студентка групи ФІ-22мн
Калитюк Дар'я

КИЇВ 2022

Мета роботи: Вибір базових бібліотек/сервісів для подальшої реалізації криптосистеми.

Постановка задачі: дослідження алгоритмів реалізації арифметичних операцій над великими (багато розрядними) числами над скінченими полями та групами з точки зору їх ефективності за часом та пам'яттю для різних програмно-апаратних середовищ. Бібліотеки багаторозрядної арифметики, вбудовані в програмні платформи C++/C# (BigInteger), Java (BigInt) та Python (обрати одну з них) для процесорів із 32-розрядною архітектурою та обсягом оперативної пам'яті до 8 ГБ (робочі станції).

Хід роботи

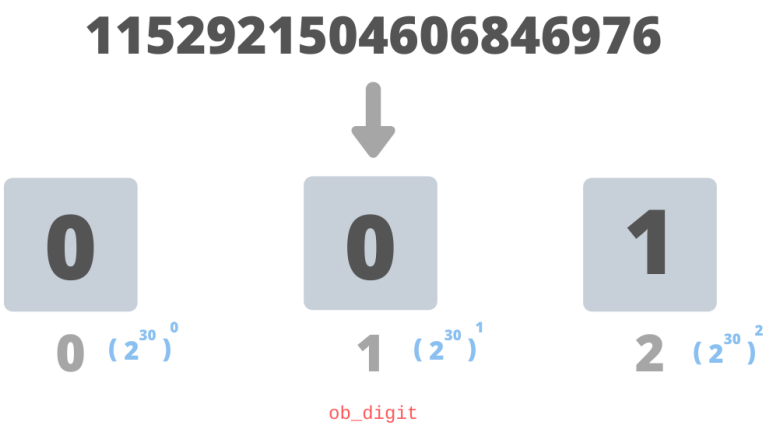
Обираючи для роботи низькорівневу мову програмування, наприклад `C`, програміст прирікає себе на постійні переживання щодо правильності вибору типу даних: чи буде достатньо `int`, чи вже слід переходити до `long` або `long double`. Мова `Python` бере цю задачу на себе, адже вона підтримує цілі числа довільного розміру.

Ціле число в Python представлено структурою C:

```
struct _longobject {
      PyObject ob_base;
      Py_ssize_t ob_size;
      digit ob_digit[1];};
```

`ob_digit` – це масив типу `digit` ,початково ініціалізований вказівником `digit *` , тож він може бути розбитий на будь-яку довжину, що і дозволяє Python представляти та обробляти великі числа.

Як Python зберігає довгі числа: задля ефективного використання усіх 32-х бітів на елемент `ob_digit` Python конвертує число на число з основою 2^{30}.



gmpy2

gmpy2 – це модуль розширення Python, кодований на C, який підтримує арифметику множинної точності. Тип даних gmpy2 mpz підтримує цілі числа довільної точності і є значно швидшим за «довгий» Python: в залежності від платформи та конкретної операції момент зміни продуктивності варіюється від 20 до 50 цифр. Для типу даних mpz модуль налічує 11 методів та 65 функцій, деякі з яких ми далі розглянемо детальніше.

Разом з mpz gmpy2 надає доступ до експерементального цілочисельного змінного типу даних xmpz: операції змінюють вихідних об'єкт і не творюють новий:

```
import gmpy2
from gmpy2 import xmpz
```

```
a = xmpz(22)
b = a
a += 1
a, b
```

```
(xmpz(23), xmpz(23))
```

Можливість змінювати об'єкт xmpz на місці дозволяє ефективно та швидко маніпулювати бітами.

Детальніше про зміст і призначення модулю, а також про особливості його встановлення для різних операційних систем можна дізнатися на сайті https://gmpy2.readthedocs.io/en/latest/intro.html.

Python VS gmpy2

Розглянемо час роботи основних операції на числах довжини 1024 біт:

```
import gmpy2
from gmpy2 import mpz
```

```
A_m = mpz(A)
B_m = mpz(B)
N_m = mpz(N)
```

Для вимірювання часу роботи операцій будемо використовувати так звану чарівну функцію IPython %timeit, що автоматично обчислює необхідну кількість запусків на основі загального вікна виконання 2 секунди.

Додавання:

```
%timeit A + B
```

```
205 ns ± 33.7 ns per loop (mean ± std. dev. of 7 runs, 10,000,000 loops each)
```

```
%timeit gmpy2.add(A_m, B_m)
```

```
306 ns ± 9.69 ns per loop (mean ± std. dev. of 7 runs, 1,000,000 loops each)
```

Віднімання:

```
%timeit A - B
```

```
200 ns ± 3.83 ns per loop (mean ± std. dev. of 7 runs, 1,000,000 loops each)
```

```
%timeit gmpy2.sub(A_m, B_m)
```

```
366 ns ± 30.7 ns per loop (mean ± std. dev. of 7 runs, 1,000,000 loops each)
```

Як можна побачити, для найелементарніших операцій додавання і віднімання використання модулю `gmpy2` не є доцільним. Але вже на множенні ми бачимо суттєву перевагу `gmpy2`: Python для довгого множення використовує алгоритм Карацуби, але це його не рятує.

Множення:

```
%timeit A*B
```

```
3.82 µs ± 43.4 ns per loop (mean ± std. dev. of 7 runs, 100,000 loops each)
```

```
%timeit gmpy2.mul(A_m, B_m)
```

```
741 ns ± 37.7 ns per loop (mean ± std. dev. of 7 runs, 1,000,000 loops each)
```

Модулярне множення:

```
%timeit A*B%N
```

```
9.01 µs ± 45.4 ns per loop (mean ± std. dev. of 7 runs, 100,000 loops each)
```

```
%timeit gmpy2.f_mod(gmpy2.mul(A_m, B_m), N_m)
```

```
2.01 µs ± 54.2 ns per loop (mean ± std. dev. of 7 runs, 1,000,000 loops each)
```

З піднесенням великого числа до степеню великого числа `Python` не справляється зовсім.

Піднесення до квадрату за модулем:

```
%timeit A**2%N
```

```
8.2 µs ± 76.1 ns per loop (mean ± std. dev. of 7 runs, 100,000 loops each)
```

```
%timeit gmpy2.powmod(A_m, 2, N_m)
```

```
3.77 µs ± 236 ns per loop (mean ± std. dev. of 7 runs, 100,000 loops each)
```

Піднесення до великого степеню за модулем:

```
#%timeit A**B%N
```

```
%timeit gmpy2.powmod(A_m, B_m, N_m)
```

```
894 µs ± 12.7 µs per loop (mean ± std. dev. of 7 runs, 1,000 loops each)
```

Пошук частки:

```
%timeit A//B
```

```
622 ns ± 4.44 ns per loop (mean ± std. dev. of 7 runs, 1,000,000 loops each)
```

```
%timeit gmpy2.f_div(A_m, B_m)
```

```
547 ns ± 15.8 ns per loop (mean ± std. dev. of 7 runs, 1,000,000 loops each)
```

Пошук остачі від ділення:

```
%timeit A%B
```

```
622 ns ± 3.41 ns per loop (mean ± std. dev. of 7 runs, 1,000,000 loops each)
```

```
%timeit gmpy2.f_mod(A_m, B_m)
```

```
522 ns ± 5.43 ns per loop (mean ± std. dev. of 7 runs, 1,000,000 loops each)
```

Для двох останніх операцій модуль `gmpy2` має одну функцію, яка їх об’єднує:

```
%timeit gmpy2.f_divmod(A_m, B_m)
```

```
599 ns ± 11.4 ns per loop (mean ± std. dev. of 7 runs, 1,000,000 loops each)
```

Модуль gmpy2 також містить вичерпну кількість інших функцій, що стануть у нагоді при реалізації різноманітних криптографічних механізмів. Наведемо деякі з них:

- `digits(x[,base=10])` – повертає string представлення `x` у системі з основою `base`;
- `divm(a, b, m)` – повертає `x` таке, що `b*x == a mod m`.
- `f_div_2exp(x, n)` – повертає частку від x поділену на `2**n`.
- `f_mod_2exp(x, n)` – повертає остачу від x поділену на `2**n`.
- `f_divmod_2exp(x, n)` – об'єднує дві попередні функції.
- `gcd(a, b)` – повертає найбільше спільне кратне для `a` та `b`.
  Порівняння часу виконання функцій `gcd` з модулей `gmpy2` та `math` на числах довжиною 1024 біта:

```
%timeit gmpy2.gcd(A_m, N_m)
```

```
4.83 µs ± 172 ns per loop (mean ± std. dev. of 7 runs, 100,000 loops each)
```

```
%timeit math.gcd(A, N)
```

```
12 µs ± 14.5 ns per loop (mean ± std. dev. of 7 runs, 100,000 loops each)
```

- `gcdext(a, b)` – повертає список з 3х елементів `(g,s,t)` таких, що `g == a*s + b*t`.
- `hamdist(x, y)` – повертає відстань Гемміна між цілими `x` та `y`.
- `invert(x, m)` – повертає y такий, що `x*y == 1 mod m`.
- `is_euler_prp(n, a)` – повертає `True` якщо `n` псевдопросте Ейлера-Якобі за основою a.
- `is_fermat_prp(n, a)` – повертає `True` якщо `n` псевдопросте Ферма за основою a.
- `is_prime(x[,n=25])` – повертає `True`, якщо `x` імовірно є простим числом. `False`, коли `x` є складеним. Спочатку `x` перевіряється для маленьких дільників, потім запускається `n` тестів Міллера-Рабіна.
- `legendre(x, y)` – повертає символ Лежандра.
- `jacobi(x, y)` – повертає символ Якобі.
- `next_prime(x)` – повертає наступне імовірно просте число > `x`.
- `remove(x, f)` – повертає пару `(y, m)` де `y=x//(f**m)`.

Зауваження: усі функції вигляду `f_…(…)` означають, що результат виконання операції округлюється в бік $-\infty$ (`floor rounding`). Для округлення в бік $+\infty$ (`ceil rounding`) треба використовувати функції вигляду `c_…(…)`, а для округлення до 0: `t_…(…)`.

Таблиця з часом виконання операцій для чисел довжини 512, 1024 та 2048 біт

512	A+B	A-B	A*B	A*BmodN	A**2modN	A**BmodN	A//B	A mod B
Python	80.6 ns	87.8 ns	640 ns	1.46 µs	1.52 µs	-	59.6 ns	61.8 ns
gmpy2	143 ns	159 ns	212 ns	547 ns	849 ns	62.4 µs	193 ns	
1024	A+B	A-B	A*B	A*BmodN	A**2modN	A**BmodN	A//B	A mod B
Python	89.2 ns	99.2 ns	1.78 µs	4.21 µs	3.85 µs	-	293 ns	302 ns
gmpy2	147 ns	158 ns	347 ns	921 ns	1.71 µs	417 µs	310 ns	
2048	A+B	A-B	A*B	A*BmodN	A**2modN	A**BmodN	A//B	A mod B
Python	109 ns	127 ns	5.778 µs	13.2 µs	11.2 µs	-	454 ns	464 ns
gmpy2	154ns	157 ns	804 ns	2 µs	4.84 µs	2.92 ms	313 ns	

Висновки:

У даній роботи спочатку було коротко розглянуто спосіб представлення великих чисел в мові Python. Незважаючи на те, що у Python є вбудовані інструменти роботи з довгими числами, час виконання майже всіх операцій (окрім додавання і віднімання) з числами довжини більше 64 біт не є задовільним у порівнянні з часом виконання функцій, реалізованими в запропонованому нами до розгляду модулі gmpy2: ми переконались, що з ростом розміру чисел різниця в часі збільшується в десятки разів.

Більш того, розглянутий модуль gmpy2 містить у собі вичерпну кількість різноманітних функцій, що реалізовують алгоритми з теорії чисел, такі як перевірка на псевдопростоту, перевірка на простоту, пошук оберненого за модулем, обчислення символів Лежандра, Якобі тощо, які завжди стануть у нагоді як при реалізації різних криптомеханізмів, так і при їхньому аналізі.

Додаток А
Числа, що були використані для формування таблиці.

512 біт:

```
A =
   0x7723A9B9AFFCFCACFEF4537D2C587BA1E07EC2A27B9C907261F646
   1044DE2FEC26DA9BACDAC77063244D575EE788BB1145D7D09A94EC4B
   59859669CF8D86F454
B =
   0xCE22515AE64F0FB5E6E33EF5D0D74AFE107BE2A91150D38A3AFCBC
   7608774631B56F41CDDD8A7D6A6C4170A4D17185E5C7578A27C2BD58
   16A62CD6D3E1E9266E
N =
   0x38CDD88155E5E68A2B66FC28861FB57657E27A1D41D3E61730FAB7
   12FB0E55728443D1A18C27DE41A5C3CAAFE43DE9484F48D282F29F85
   05F4BDF734D492B484
```

1024 біт:

```
A =
   0xD4D2110984907B5625309D956521BAB4157B8B1ECE04043249A3D3
   79AC112E5B9AF44E721E148D88A942744CF56A06B92D28A0DB950FE4
   CED2B41A0BD38BCE7D0BE1055CF5DE38F2A588C2C9A79A75011058C3
   20A7B661C6CE1C36C7D870758307E5D2CF07D9B6E8D529779B6B2910
   DD17B6766A7EFEE215A98CAC300F2827DB
B =
   0x3A7EF2554E8940FA9B93B2A5E822CC7BB262F4A14159E4318CAE3A
   BF5AEB1022EC6D01DEFAB48B528868679D649B445A753684C13F6C3A
   DBAB059D635A2882090FC166EA9F0AAACD16A062149E4A0952F7FAAB
   14A0E9D3CB0BE9200DBD3B034249642182691914 8E617AF1DB66978B
   1FCD28F8408506B79979CCBCC7F7E5FDE7
N =
   0x269D7722EA018F2AC35C5A3517AA06EAA1949059AE8240428BBFD0
   A8BE6E2EBF91223991F80D7413D6B2EB213E7122710EDEC617460FA0
   191F39016046199720 18EBEF22D81AED9C56424014CADCC2CCDEE67D
   36A54BFC500230CA6693ABA057B374746622341ED6D52FE5A79E6860
   F54F197791B3FEF49FD534CB2C675B6BDB
```

2048 біт:

```
A =
   0x87D6D58D3991D536544389CEFA72FD0EBED75B2EBDC2C79BC37177
   93108F0952011E7E2D7040FFFB32F10BEB8ED0A485026B6860020B23
   0128A8222B0525A6888942FB01C537800BF25D6F021D4B99D3CBD6DF
```

9055FA22F91A6CFC4FDFC408AEF78F6418D3CE4E20EC7888B61BAE3D
73C27C257CCA905DE0353C3A7CFFD9FE15170076B15F9575D21DE39D
5C429799BBCDDB867016DE2248E3CFDE73A4D70C8636A9E41ABE671E
7B9FB4739A5FF64DF9D0D3A64E0C9B20BFE58F1C62B28477EE9FD202
010BAC440ADF3CA016A32DB844F23DEC2AB93AE869A6262FC23C5CE4
19807CDBA930A5433884E3B34B22477289BD3A7712CDD4B4110BD988
7E7428FDF7

B =
0x791EDB102DA183759979CEF70E1405AF14B98CD44357EADF6A8E35
E49F99BB56CBD3F68897D6E05502ED1DE14EC46D04F96992C2D12973
7987E84E62371648B37633794016852A8CBFFCFDE06B17EC216AE891
4D59E677A15A90361A594F0D1524A41AE63C59D343D4E522646722B0
292DD7C85571AC9A84FDA6CD2D8DE307F69D1C2D6E1591932F73C2F4
99C4E0A2E252DE828CDA7842CE0972C4101FE772B56C45C475EDDEDA
EC2DBD13E375E02D2C149B69AB51FF3F94533CA34A815484EC86DACE
936BDC62B5F3F9EB6F5BE6BD253E256181D35D7D63EE24459824D462
C53676E3DFF98700415ADA65FDA7CBD3B3F359C817F52BEDA70C9DD8
5F68473C6

N =
0xFAAE2DBD9EECEE161154B081A68CB675BFC633DF8811446F22C2AB
317B4F76CFFC36AF3078C795EBB23EEB59DEA12EA2E2E7F05426B9FA
209A9EF21DFBB4111A49F75684193CF705FE0B1E4A96E88733981ECE
3AABEC42506A92B199681392882EF5180A5EF518373DA17D712E9BF3
936FBCAB1AF13BB215DA73B29B80D36DCAA4A28E711996BE9ED7B976
871E170770E6B4D48DBCE916AC3D978D2146873AF4DA92CF8D023404
EA5D78087C31933D8F2AF0BBE7AB619A5A07AEDBD2E3D582C05826B1
1F143BD80B64007F8E407DF0F1B9A7678846E0CDD1912B7DFBFBFCB1
B9FFF0E54E73FA4248B24771789D2F11885F456545B9ED8C90F925F3
8B620677DC