МІНІСТЕРСТВО ОСВІТИ І НАУКИ, МОЛОДІ ТА СПОРТУ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені Ігоря Сікорського»
ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ

Методи реалізації криптографічних механізмів

Лабораторна робота №1

“Бібліотека багаторозрядної арифметики GNU GMP”

Виконали:

Студент ФІ-22ми
Підгрупа 1С

Бондаренко Андрій
Гузей Дмитро
Яценко Артем

Перевірила:

Байденко П.В.

Київ-2023

Завдання

Бібліотека багаторозрядної арифметики GNU GMP для паралельної моделі обчислень – декілька процесорів (можливо багатоядерних) із 64-розрядною архітектурою та обсягом оперативної пам’яті до 32 ГБ. Приклад – сервер обробки транзакцій.

Теорія:

Що таке GMP?

GMP — це безкоштовна бібліотека для арифметики довільної точності, що працює з цілими числами зі знаком, раціональними числами та числами з плаваючою комою. Немає жодних практичних обмежень точності, окрім тих, що передбачають доступну пам’ять у машині, на якій працює GMP. GMP має багатий набір функцій, і функції мають звичайний інтерфейс.

Основними цільовими програмами для GMP є програми та дослідження криптографії, програми безпеки в Інтернеті, системи алгебри, дослідження обчислювальної алгебри тощо.

GMP ретельно розроблено, щоб бути максимально швидким як для малих, так і для великих операндів. Швидкість досягається використанням повних слів як основного арифметичного типу, використанням швидких алгоритмів, високооптимізованим кодом складання для найпоширеніших внутрішніх циклів для багатьох ЦП, а також загальним наголосом на швидкості.

Основними цільовими платформами GMP є системи типу Unix, такі як GNU/Linux, Solaris, HP-UX, Mac OS X/Darwin, BSD, AIX тощо. Також відомо, що він працює як у 32-бітній, так і в 64-розрядній версіях Windows. бітовий режим.

Категорії функцій GMP

У GMP є кілька категорій функцій:

1. Цілочисельні арифметичні функції зі знаком високого рівня (**mpz**). У цій категорії близько 150 арифметичних і логічних функцій.
2. Раціональні арифметичні функції високого рівня (**mpq**). Ця категорія складається з приблизно 35 функцій, але всі функції mpz також можна використовувати, застосовуючи їх до чисельника та знаменника окремо.
3. Високорівневі арифметичні функції з плаваючою комою (**mpf**). Це категорія функції GMP, яка використовується, якщо тип C "double" не дає достатньої точності для програми. У цій категорії близько 70 функцій. Новим проектам слід наполегливо розглянути можливість використання значно повнішої бібліотеки розширення GMP *mpfr* замість *mpf*.
4. Інтерфейс на основі класу C++ для всього вищезазначеного. (Звичайно, функції та типи C також можна використовувати безпосередньо з C++.)
5. Додатні цілі функції низького рівня, складні у використанні та дуже низькі накладні витрати, знаходяться в категорії mpn. Керування пам'яттю не виконується; абонент повинен переконатися, що достатньо місця для результатів. Набір функцій не завжди звичайний, як і інтерфейс виклику. Ці функції приймають вхідні аргументи у формі пар, що складаються з вказівника на найменш значуще слово та розміру інтеграла, який повідомляє, скільки кінцівок (= слів) є в цьому аргументі. Функції в інших категоріях викликають mpn для майже всіх своїх обчислень. З цих функцій близько 60 є державними.

C/C++

Ми завантажили gmp для C/C++ з https://gmplib.org/

Ми будемо демонструвати бібліотеку GMP на мові C++, хоча бібліотека на мові C є надзвичайно схожою у використанні. Бібліотека включає 3 типи класів:

- Класс **mpz_class** для великих цілих чисел (відповідний тип у мові C - *mpz_t*)
- Класс **mpq_class** для відношень (відповідний тип у C - *mpq_t*)

- Класс **mpf_class** для чисел з плаваючою комою (відповідний тип у мові C - *mpf_t*)

Я зупинюсь на класі **mpz_class**, оскільки припускаю, що саме він цікавить більшість. Класи **mpq_class** та **mpf_class** мають однакову структуру.

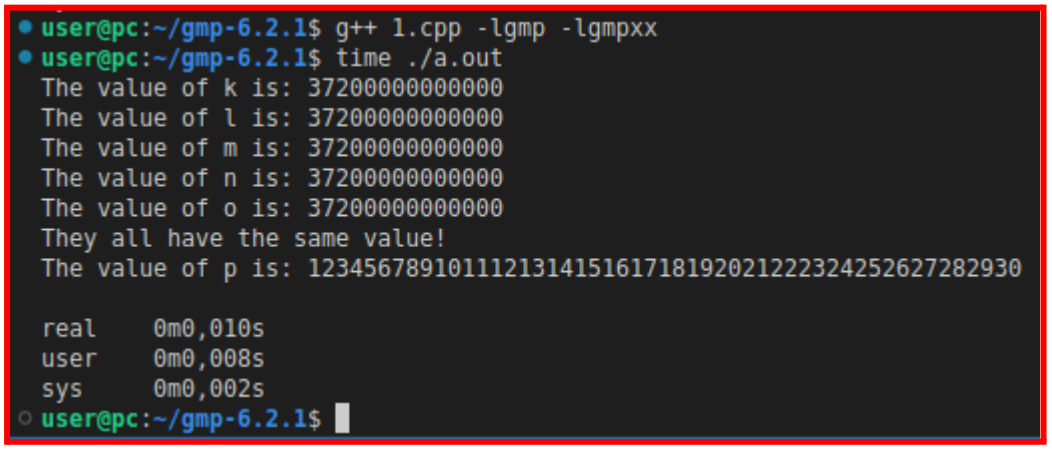
Ініціалізація **mpz_class** така ж проста, як і створення об'єкту в класі, який ви створили самі. При створенні нового цілого числа ви можете надати йому початкове значення або не надавати. Нижче продемонстровано різні способи створення по суті одного і того ж цілого числа.

```cpp
#include <iostream>
#include "gmpxx.h"

int main()
{
    // Different ways to initialize the same integer
    // 37200000000000 is an estimation of the number of cells in the human body
    mpz_class k(37200000000000);
    mpz_class l = 37200000000000;
    mpz_class m{37200000000000};
    mpz_class n;
    n = 37200000000000;
    long o_long = 37200000000000;
    mpz_class o(o_long);

    std::cout << "The value of k is: " << k << '\n';
    std::cout << "The value of l is: " << l << '\n';
    std::cout << "The value of m is: " << m << '\n';
    std::cout << "The value of n is: " << n << '\n';
    std::cout << "The value of o is: " << o << '\n';

    if ((k == l && m == n) && (k == m && o == k)) {
        std::cout << "They all have the same value!" << '\n';
    }

    // You can also initialize using strings!
    mpz_class p("12345678910111213141516171819202122232425262727282930");
    std::cout << "The value of p is: " << p << '\n';

    return 0;
}
```

Тепер нам потрібно скомпілювати наш код. Ми використовую g++. Єдине, що потрібно зробити додатково, це зв'язати бібліотеки GMP libgmp та libgmpxx за допомогою прапорця -l.

```
user@pc:~/gmp-6.2.1$ g++ 1.cpp -lgmp -lgmpxx
user@pc:~/gmp-6.2.1$
```

Після цього запустити виконуваний файл і результат:



Також зверніть увагу, що для передачі цілого числа в потік cout не потрібно було використовувати спеціальну функцію, а достатньо використати стандартний оператор вставки '<<'.
Бібліотека C++ має *перевантажені* оператори, які дозволяють використовувати об'єкти **mpz_class** так само, як і звичайне ціле число.

Продемонструвати це можна у наступній програмі:

```cpp
#include <iostream>
#include <gmpxx.h>

int main()
{
    mpz_class a ("150000000000000000000");
    mpz_class b {"150000000000000000000"};
    std::cout << "a is: "<< a << " while b is: " << b << '\n';

    // All the usual operations can be used!
    std::cout << "The value of a + b is: " << a+b << '\n';
    std::cout << "The value of a - b is: " << a-b << '\n';
    std::cout << "The value of a * b is: " << a*b << '\n';
    std::cout << "The value of a / b is: " << a/b << '\n';
    std::cout << "The value of a % b is: " << a%b << '\n';
    std::cout << "The value of -a is: " << -a << '\n';

    // The big integers also can be intermixed with regular C/C++ types!
    mpz_class d = 10;
    int k = 34;
    std::cout << "d is a mpz_class, e is an int; d + e = " << d + k << '\n';
    double l = 3.14;
    std::cout << "d is a mpz_class, l is a double; d + l = " << d + l << '\n';
    long long m = 2.71;
    std::cout << "d is a mpz_class, m is a long long, so they can't be directly added;"
    " using conversion d + m = " << d + int(m) << '\n';

    // Comparisons also work!
    mpz_class e = 9;
    if (d > e) std::cout << "> works as intended" << '\n';
    if (b-1 >= e) std::cout << ">= works too" << '\n';

    // Intermixing with standard types works as well!
    if (d >= 8 && d < 11) std::cout << "Comparisons with standard types work!" << '\n';

    return 0;
}
```

```
user@pc:~/gmp-6.2.1$ g++ 2.cpp -lgmp -lgmpxx
user@pc:~/gmp-6.2.1$ time ./a.out
a is: 150000000000000000000 while b is: 150000000000000000000
The value of a + b is: 165000000000000000000
The value of a - b is: 135000000000000000000
The value of a * b is: 225000000000000000000000000000000000000000
The value of a / b is: 10
The value of a % b is: 0
The value of -a is: -150000000000000000000
d is a mpz_class, e is an int; d + e = 44
d is a mpz_class, l is a double; d + l = 13
d is a mpz_class, m is a long long, so they can't be directly added; using conversion d + m = 12
> works as intended
>= works too
Comparisons with standard types work!

real    0m0,009s
user    0m0,007s
sys     0m0,003s
user@pc:~/gmp-6.2.1$
```

Як бачимо, змішування з більшістю стандартних типів працює чудово.

GMP включає в себе безліч дуже корисних функцій. Найважливіші з них мають еквіваленти для класів C++ (наприклад, абсолютне значення, найбільший спільний знаменник або факторіал), тоді як інші працюють тільки для типів C GMP. Ви можете легко використовувати їх з типами C++, передаючи посилання на базовий об'єкт C разом з функціями:

Для mpz:

mpz_t mpz_class::get_mpz_t()

Для mpq:

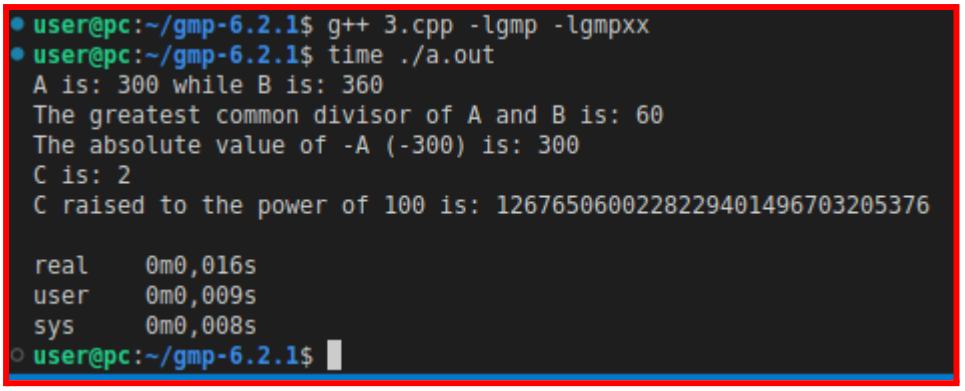
mpq_t mpq_class::get_mpq_t()

Для mpf:

mpf_t mpf_class::get_mpf_t()

Наступна програма демонструє деякі з функцій:

```cpp
#include <iostream>
#include <gmpxx.h>

int main()
{
    mpz_class A = 300;
    mpz_class B = 360;
    std::cout << "A is: " << A << " while B is: " << B << '\n';
    // Many functions are included in the C++ interface
    std::cout << "The greatest common divisor of A and B is: " << gcd(A, B) << '\n';
    std::cout << "The absolute value of -A (-300) is: " << abs(A) << '\n';

    // With functions of the C library a simple conversion works
    mpz_class C = 2;
    std::cout << "C is: " << C << '\n';

    // Raise to the 65th power
    mpz_pow_ui(C.get_mpz_t(), C.get_mpz_t(), 100);
    std::cout << "C raised to the power of 100 is: " << C << '\n';

    return 0;
}

```



В останньому прикладі алгоритм сортування вставкою зі створеними мною шаблонами для сортування масиву mpz_class. Ми будемо зчитувати та вставляти значення в масив з допомогою рядків.

```cpp
#include <iostream>
#include <gmpxx.h>

template <typename T>
void insertion_sort(T arr[], int N)
{
    for (int i = 1; i < N; i++) {
        for (int j = i; j > 0; j--) {
            if (arr[j] < arr[j-1]) {
                auto temp = arr[j-1];
                arr[j-1] = arr[j];
                arr[j] = temp;
            }
            else break;
        }
    }
    return;
}

int main()
{
    int size;
    std::cin >> size;
    std::cin.ignore(100, '\n'); // tell the input to ignore the first newline character
    mpz_class arr[size];
    std::string in;

    // read the numbers line by line
    for (int i = 0; i < size; i++) {
        std::getline(std::cin, in);
        arr[i] = in;
    }

    std::cout << "Array before sorting: " << '\n';
    for (int i = 0; i < size; i++) {
        std::cout << arr[i] << '\n';
    }

    insertion_sort(arr, size);
    std::cout << "Array after sorting: " << '\n';
    for (int i = 0; i < size; i++) {
        std::cout << arr[i] << '\n';
    }

    return 0;
}
```


user@pc:~/gmp-6.2.1$ g++ 4.cpp -lgmp -lgmpxx
user@pc:~/gmp-6.2.1$ time ./a.out
10
9999999999999999999999999999999999999999999999999999
8888888888888888888888888888888888888888888888888888
7777777777777777777777777777777777777777777777777777
6666666666666666666666666666666666666666666666666666
5555555555555555555555555555555555555555555555555555
4444444444444444444444444444444444444444444444444444
3333333333333333333333333333333333333333333333333333
2222222222222222222222222222222222222222222222222222
1111111111111111111111111111111111111111111111111111
0000000000000000000000000000000000000000000000000000
Array before sorting:
9999999999999999999999999999999999999999999999999999
8888888888888888888888888888888888888888888888888888
7777777777777777777777777777777777777777777777777777
6666666666666666666666666666666666666666666666666666
5555555555555555555555555555555555555555555555555555
4444444444444444444444444444444444444444444444444444
3333333333333333333333333333333333333333333333333333
2222222222222222222222222222222222222222222222222222
1111111111111111111111111111111111111111111111111111
0
Array after sorting:
0
1111111111111111111111111111111111111111111111111111
2222222222222222222222222222222222222222222222222222
3333333333333333333333333333333333333333333333333333
4444444444444444444444444444444444444444444444444444
5555555555555555555555555555555555555555555555555555
6666666666666666666666666666666666666666666666666666
7777777777777777777777777777777777777777777777777777
8888888888888888888888888888888888888888888888888888
9999999999999999999999999999999999999999999999999999
real 0m48,868s
user 0m0,012s
sys 0m0,004s
user@pc:~/gmp-6.2.1$

Приклад 5:

```cpp
#include <gmp.h>
#include <stdio.h>

const int N = 20000;
const int M = 1000;
mpz_t A[N], B[N];

unsigned int max(unsigned int x, unsigned int y) {
  return x > y ? x : y;
}

void polynomial_multiply(mpz_t A[], int n) {
  for (int i = N - 1; i >= n; --i) {
    mpz_sub(A[i], A[i], A[i - n]);
  }
}

int main() {
    printf("Output in series.out file\n");
  freopen("series.out", "w", stdout);
  mpz_array_init(A[0], N, M);
  mpz_array_init(B[0], N, M);
  mpz_set_si(A[0], 1);
  for (int i = 1; i < N; ++i) {
    polynomial_multiply(A, i);
    polynomial_multiply(A, i);
    polynomial_multiply(A, i);
  }
  for (int i = 3; i < N; i += 3) {
    polynomial_multiply(A, i);
  }
  mpz_set_si(B[0], 1);
  for (int i = 1; i < N; ++i) {
    mpz_set_si(B[i], 0);
    for (int j = 1; j <= i; ++j) {
      mpz_submul(B[i], A[j], B[i - j]);
    }
  }
  for (int i = 0; i < N; ++i) {
    gmp_printf("%5d: %Zd\n", i, B[i]);
  }

  unsigned int max_size = 0;
  for (int i = 0; i < N; ++i) {
    max_size = max(max_size, mpz_size(A[i]));
    max_size = max(max_size, mpz_size(B[i]));
  }
  printf("%u\n", max_size);
  return 0;
}
```

Результат:

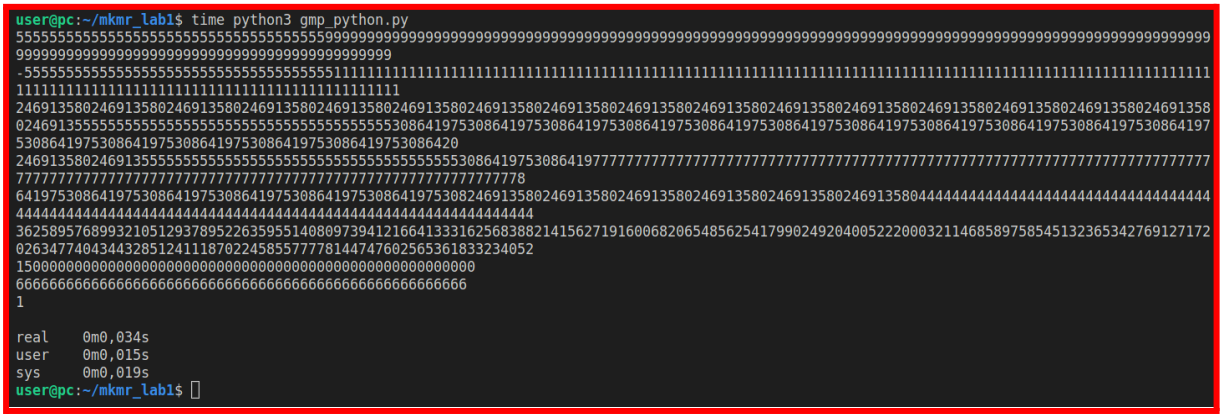
```
user@pc:~/gmp-6.2.1$ g++ 5.cpp -lgmp -lgmpxx
user@pc:~/gmp-6.2.1$ ./a.out
Output in series.out file
user@pc:~/gmp-6.2.1$ time ./a.out
Output in series.out file

real    0m14,221s
user    0m14,157s
sys     0m0,026s
user@pc:~/gmp-6.2.1$
```

Python

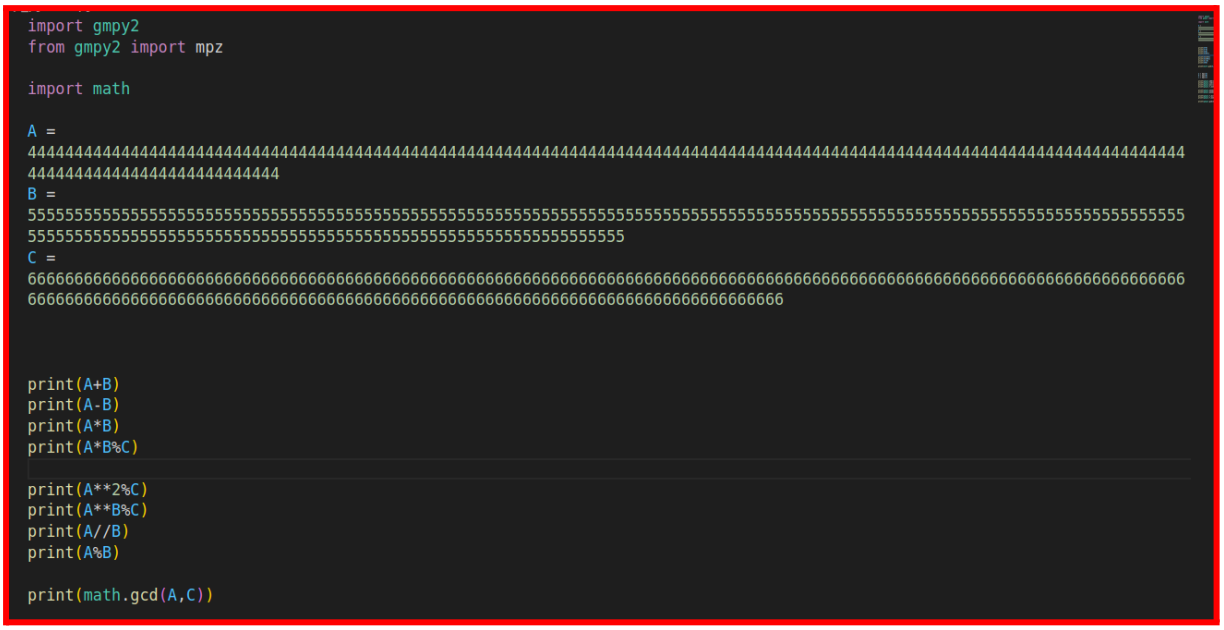
Тип gmpy2 mpz підтримує цілі числа довільної точності. Це має бути додаткова заміна long типу Python. Залежно від платформи та конкретної операції mpz буде швидшим, ніж long у Python, коли точність перевищить 20–50 цифр. Підтримуються всі спеціальні цілочисельні функції в GMP.

Gmpy2:

```
import gmpy2
from gmpy2 import mpz

# import math

A =
44444444444444444444444444444444444444444444444444444444444444444444444444444444444444444444444444444444444444444444444444444
444444444444444444444444444
B =
55555555555555555555555555555555555555555555555555555555555555555555555555555555555555555555555555555555555555555555555555555
5555555555555555555555555555555555555555555555555555555555555555
C =
66666666666666666666666666666666666666666666666666666666666666666666666666666666666666666666666666666666666666666666666666666
666666666666666666666666666666666666666666666666666666666666666666666666666666666

a = mpz(A)
b = mpz(B)
c = mpz(C)

print(gmpy2.add(a,b))
print(gmpy2.sub(a,b))
print(gmpy2.mul(a,b))
print(gmpy2.f_mod(gmpy2.mul(a,b),c))

print(gmpy2.powmod(a,2,c))
print(gmpy2.powmod(a,b,c))

print(gmpy2.f_div(c,a))
print(gmpy2.f_mod(c,a))

print(gmpy2.gcd(b,c))
```



Звичайний Пайтон:



Працювало більше 10 хвилин…

Висновок

Ми навчились встановлювати бібліотеку gmp для мов платформ (C/C++, Python). Розглянули базові операції з класом mpz. Для python порівняли швидкодію вбудованих арифметичних операцій і gmp операцій.