A description...

НТУУ «Київський Політехнічний Інститут»

Спеціальні розділи обчислювальної математики

Комп’ютерний практикум №2

**Виконав:**

Студент 3 курсу ФТІ

групи ФИ-83

Паршин О.Ю.

**Прийняла:**

Пекарчук Н.А.

1. **Мета роботи**

Отримання практичних навичок програмної реалізації багаторозрядної арифметики; ознайомлення з прийомами ефективної реалізації критичних по часу ділянок програмного коду та методами оцінки їх ефективності.

**2. Теоретичні відомості**

**2.1. Алгоритм Евкліда**

Алгоритм Евкліда обчислює найбільший спільний дільник двох чисел *d* =gcd( *a*,*b*)

шляхом ітеративної процедури, яка ґрунтується на такому факті: якщо *a* ³ *b* , то

gcd( *a*, *b*) = gcd(*b*, *a* - *b*) . Звідси одразу випливає, що gcd( *a*, *b*) = gcd(*b*, *a* mod *b*) , і процедура

обчислення НСД задається наступним чином.

Нехай *r*0 = *a* , *r*1 = *b* ; обчислюємо послідовність (*ri* ) для *i* ³ 2 шляхом ділення з остачею:

*r*0= *r*1*q*1+ *r*2,

*r*1= *r*2*q*2+ *r*3,

…

*rs* -2= *rs* -1*qs* -1+ *rs* ;

*rs* -1= *rs qs* .

Якщо на відповідному кроці виявилось, що *rs*+1 = 0 , то *d* = *rs* .

Складність алгоритму Евкліда є лінійною по відношенню до бітової довжини *n* аргументів. Дійсно, найгірший випадок для роботи алгоритму – коли всі *qi* = 1. Цей випадок відповідає ситуації, коли *a* та *b* – два послідовні числа Фібоначчі. З формули Біне випливає, що

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| число Фібоначчі асимптотично веде себе як *f* |  | ~ *f* *n* , де *f* = |  | 5 | +1 | | – відношення «золотого |  |
| *n* |  |  |  |  |  |
|  | 2 | | | |  |  |  |
|  |  |  |  |  |



перерізу», а тому алгоритм Евкліда виконає не більше ніж élog*f* *a*ù= *O*(log *a*) = *O*(*n*) операцій.

Розширений алгоритм Евкліда обчислює дві додаткові послідовності (*ui* ) та (*vi* ) такі, що на кожному кроці виконується рівність *ri* = *ui* *a* + *vib* ; зокрема, для найбільшого спільного дільника матимемо *d* = *rs* = *us* *a* + *vsb* . Ці послідовності також можна обчислити рекурентно за

допомогою часток *qi* :

*u*0=1, *u*1=0, *ui*+1= *ui*-1- *qiui* ;

*v*0=0, *v*1=1, *vi*+1= *vi*-1- *qivi* .

Зауважимо, що хоча алгоритм Евкліда є лінійним за кількістю операцій, він використовує операції ділення з остачею, які самі по собі є важкими. Для запобігання цього був розроблений інший алгоритм, що має назву *бінарний алгоритм обчислення НСД* або *алгоритм Стейна*. Цей алгоритм базується на таких спостереженнях:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| æ *a* | |  | *b* | ö |  |
| 1) якщо і *a*, і *b* – парні, то gcd( *a*, *b*) = 2 gcd ç |  | , |  | ÷ ; |  |
| 2 | 2 |  |
| è |  | ø |  |

|  |  |  |  |
| --- | --- | --- | --- |
| æ *a* | | ö |  |
| 2) якщо *a* – парне, а *b* – непарне, то gcd( *a*, *b*) = gcd ç |  | , *b* ÷ ; |  |
| 2 |  |
| è | ø |  |

1. якщо і *a*, і *b* – непарні, то gcd( *a*, *b*) = gcd (min{*a*, *b*}, *a* - *b* ), причому різниця є парним числом.

На відміну від алгоритму Евкліда, бінарний алгоритм використовує лише віднімання та ділення на два, яке у двійкових архітектурах ефективно реалізується як бітовий зсув.

Отже, бінарний алгоритм можна подати у вигляді такої процедури.

d := 1;

while (a – парне) and (b парне) do: // виокремлення загальної парної частини a := a / 2;

b := b / 2;

d := d \* 2;

while (a – парне) do:

a := a / 2;

while (b <> 0) do:

while (b – парне) do:

b := b / 2;

(a, b) := (min{a, b}, abs(a – b))

d := d \* a;

return d;

На кожному кроці одне з двох оброблюваних чисел скорочується на один біт (шляхом ділення на два), тому бінарний алгоритм виконає не більш ніж 2 log *a* кроків. Це більше, ніж в

класичному алгоритмі Евкліда, але використання віднімання та зсувів замість ділення на практиці робить бінарний алгоритм суттєво швидшим.

**2.2. Редукція за Барреттом**

* багатьох асиметричних криптографічних алгоритмах потрібно виконувати обчислення за модулем деякого натурального числа. Прямолінійний підхід полягає в тому, щоб застосовувати ділення з остачею після кожної арифметичної операції; однак, як зазначалось, ділення є дуже складною операцією, тому були розроблені спеціальні алгоритми, які б дозволяли замінити ділення на інші, більш прості операції.

Перевага таких алгоритмів відчутна тоді, коли потрібно виконувати багато обчислень за одним й тим самим модулем. В цьому випадку за рахунок деяких передобчислень вдається

суттєво зекономити на обчисленнях в кожному конкретному випадку, що призводить до значного пришвидшення обчислень в цілому.

Розглянемо перший з таких методів – *алгоритм модулярної редукції Барретта*.

Отже, нехай дано багаторозрядні числа *n* та *x* у системі числення із основою *b*, причому довжина *x* вдвічі більша за *n*: *n* = *k* , *x* = 2*k* . Необхідно знайти частку *q* та остачу *r* від ділення *x* на *n*: *x* = *qn* + *r* , 0 £ *r* < *n* .

*Зауваження:* для коректного обчислення *m* необхідно правильно визначити степінь *b* 2*k* ,

яка має 2*k* +1 цифру в записі (на одну більше, ніж в записі *x*); це може бути критичним при реалізації у моделі із фіксованою довжиною числа.

Алгоритм редукції за Барреттом можна подати у вигляді такої процедури.

**Процедура BarrettReduction (*x*, *n*, *m*, *r*)**

ê*b* 2*k* ú

Вхід: багаторозрядні числа *x*, *n*, передобчислене значення *m* = ê ú .

ë *n* û

Вихід: багаторозрядне число *r* = *x* mod *n* .

q := KillLastDigits(x, k-1); q := q \* m;

// відкидання останніх k-1 цифри

1. := KillLastDigits(q, k+1);
2. := x – q \* n;

while (r >= n) do:

r := r – n;

return r;

// Барретт гарантує, що цикл виконується // не більше двох разів

Бачимо, що редукція за Барретом для обчислення остачі використовує не ділення, а множення (та декілька зсувів та віднімань), що значно пришвидшує обчислення за модулем.

Покажемо застосування редукції за Барреттом на прикладі схеми Горнера. Саме в схемі Горнера під час піднесення до степеня потрібно виконувати багато операцій множення за одним модулем, що є необхідною передумовою для ефективного застосування редукції за Барреттом.

**Процедура LongModPowerBarrett (*A*, *B*, *N*, *C*)**

Вхід: багаторозрядні числа *A*, *B*, *N*; *В* задане двійковим записом *B* = *bm*-1 2*m*-1 + ... + *b*1 2 + *b*0 .

Вихід: багаторозрядне число *C* = *AB* mod *N* .

C:=1;

m := LongShiftDigitsToHigh(1, 2\*k) / n;

for i := 0 to m-1 do:

if b[i] = 1 then:

// єдине ділення!

C := BarrettReduction(C \* A, N,

A := BarrettReduction (A \* A, N, m);

return C

m);

**2.3. Редукція за Монтгомері**

На відміну від метода Барретта, Монтгомері запропонував для обчислення лишків взагалі перейти у іншу арифметичну систему, в якій редукція сама по собі виконується значно швидше.

Нехай *n* – непарне число, а *R* > *n* – число, взаємно просте із *n* (зазвичай для зручності обирають *R* = 2*t* ). *Лишком Монтгомері* числа *x* називають вираз *mont*(*x*) = *xR* mod *n* . *Функцією* *редукції Монтгомері* називають функцію *redc*(*x*)= *x* × *R*-1mod *n* ,де *R*-1–число,обернене до *R* замодулем *n*. Система лишків Монтгомері утворює арифметику із такими операціями:

*mont*(*x* ± *y*)= *mont*(*x*)± *mont*( *y*),

*mont*(*x* × *y*)= *redc*(*mont*(*x*)× *mont*( *y*)),

і треба мати на увазі, що *redc*(*mont*(*x*))= *x* mod *n* . Таким чином, довільний арифметичний

алгоритм, який використовує лише додавання, віднімання та множення, може бути переписаний для системи лишків Монтгомері таким чином:

1. всі вхідні змінні *x* та константи замінюються на лишки Монтгомері *mont*(*x*) ;
2. всі множення *xy* замінюються на *redc*(*xy*) ;
3. всі вихідні змінні *z* замінюються на *redc*(*z*) .

Наприклад, схема Горнера у системі лишків Монтгомері буде виглядати так:

**Процедура LongModPowerMontgomery (*A*, *B*, *N*, *C*)**

Вхід: багаторозрядні числа *A*, *B*, *N*; *В* задане двійковим записом *B* = *bm*-1 2*m*-1 + ... + *b*1 2 + *b*0 .

Вихід: багаторозрядне число *C* = *AB* mod *N* .

A := mont(A);

C := mont(1);

for i := 0 to m-1 do:

if b[i] = 1 then:

C := redc(C \* A);

A := redc(A \* A);

return redc(C)

Головною перевагою методу Монтгомері є обчислення функції *redc*(*x*) , яке може бути виконане суттєво швидше, ніж звичайне ділення з остачею. Операція ділення залишається лише при початкових обчисленнях *mont*(*x*) .

Покажемо схематично, як швидко обчислювати значення *redc*(*x*) .

* 1. За допомогою розширеного алгоритму Евкліда знаходяться такі числа *R*-1 та *n*¢ , що *RR*-1- *nn*¢ =1(зауважимо, що *R*-1– це обернений до *R* за модулем *n*).Цей крок єпередобчисленням.
  2. Обчислити *u* = *x* + (*x* × *n*¢mod *R*) × *n* . Оскільки внутрішнє множення береться за модулем

*R* =2*t* , то для його обчислення достатньо взяти по *t* біт аргументів, що прискорює обчислення;також відмітимо, що операція mod *R* – це просто одержання останніх *t* біт числа.

3) Обчислити *u* = *u* / *R* (тобто у числа *u* відкидаються останні *t* біт). Якщо *u* ³ *n* , то

* = *u* - *n* . Повернути *u* як *redc*(*x*).

Цю процедуру також можна пришвидшити, якщо звести відповідні обчислення до окремих цифр результату. Детальніше про це наведено у конспекті лекцій.

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |

**Завдання до комп’ютерного практикуму**

А) Допрацювати бібліотеку для роботи з *m*-бітними цілими числами, створену на комп’ютерному практикумі №1, додавши до неї такі операції:

1. обчислення НСД та НСК двох чисел;
2. додавання чисел за модулем;
3. віднімання чисел за модулем;
4. множення чисел та піднесення чисел до квадрату за модулем;
5. піднесення числа до багаторозрядного степеня *d* по модулю *n* .

Модулярну арифметику рекомендовано реалізовувати на базі редукції Баррета, піднесення до степеня – на базі схеми Горнера. Мова програмування, семантика функцій та спосіб реалізації можуть обиратись довільним чином.

Окрім основного завдання, ви також можете виконати додаткове завдання згідно варіанту.

**Результати**

*Тестові дані:*

*A:* D4D2110984907B5625309D956521BAB4157B8B1ECE04043249A3D379AC112E5B9AF44E721E148D88A942744CF56A06B92D28A0DB950FE4CED2B41A0BD38BCE7D0BE1055CF5DE38F2A588C2C9A79A75011058C320A7B661C6CE1C36C7D870758307E5D2CF07D9B6E8D529779B6B2910DD17B6766A7EFEE215A98CAC300F2827DB

*B:*

3A7EF2554E8940FA9B93B2A5E822CC7BB262F4A14159E4318CAE3ABF5AEB1022EC6D01DEFAB48B528868679D649B445A753684C13F6C3ADBAB059D635A2882090FC166EA9F0AAACD16A062149E4A0952F7FAAB14A0E9D3CB0BE9200DBD3B0342496421826919148E617AF1DB66978B1FCD28F8408506B79979CCBCC7F7E5FDE7

*N:*

269D7722EA018F2AC35C5A3517AA06EAA1949059AE8240428BBFD0A8BE6E2EBF91223991F80D7413D6B2EB213E7122710EDEC617460FA0191F3901604619972018EBEF22D81AED9C56424014CADCC2CCDEE67D36A54BFC500230CA6693ABA057B374746622341ED6D52FE5A79E6860F54F197791B3FEF49FD534CB2C675B6BDB

Gcd(A,B):

1

Середній час виконання: 0.0988379 s

Lcm(A,B):



Середній час виконання: 0.137333 s

A+B mod N:

102c16a6d0ed225693dd8c7a79e56c55cce8d4c49ce26920413599bd1f8f7418f71bd53506aec5052c66e01a4ed59fc3a47baf9ea0ebefaa32aadcd43012ea56d2ee253ac2c64796059644cb9db2ab9f00601b6c38c4f61caafce078bfa165f691ac5868185f397625521e17ce5f547bb2d29af180ce9504ee7dac1338e32c5

Середній час виконання: 0.139564 s

A-B mod N:

267ab94b78028cdb3f87dc513600d9797e5ae56f81235f3919b626c015db91f9fb209fde3b37a5fa9cc14b4ad57b5b0b8b55c9d48374c9a7ca037887a7168713b15bd109ce82c54f8c21a076a8ba23487baaa06816e8990abba0b78660329139a424541a382445d5f41ed4c9295862de5d411774ddfb4c9cb0218de3e12fe663

Середній час виконання: 0.141671 s

A\*B mod N:

b274d0d82e9b2997834fe06626976c55d7bad04a38623214d2c29f08d4bb6d4c3eb31d0bf3b5bb7965f820af3ceb31fd0498d6e94feab692c36df783c400d19a0f1a25e9232945024c249770c34f610d9dd31672b1cfbde78856afe52cb54ca3750fb6d4529c0b4e491c12aebb0f2db2f8449107611cc66cac56b4275c31ee7

Середній час виконання: 0.208653 s

A2 mod N:

24e77f478d875f430d39e7fdc1b054047b6a9651b77dc041616a5fdb3a901199bf6634c79eef04e49d73aba45b2e4415c2320b7f12fc3029a66838d8f29d2368d0cd8cc8b6c6c2504c81e98560bfcb0598735a5318279d82cf623df27a1b5debe73571d3b18a348376e14d99d91fd6707428085a74ea0e836ab851c9496a65a1

Середній час виконання: 0.217559 s

A^B mod N:

11e97d13ac2368a6ddcc2a87b5c71cd0b33951d68ef905f95a7efa2bf49ebeec63bfdd69c6022e93d0963407b8a0942a72c6cafbccfc66c3ed2bbe86751b41e7d69e23a430794e9edd055b5741ad83bda475af10fed1b331cb4287280d51a518a3940c24c336f789cd8035d4156a29e15a88ede96e0ef32cd1be12ea034fece1

Середній час виконання: 357.304 sec

**Результати тестових перевірок**

*Тест Б.1*

*A:* D4D2110984907B5625309D956521BAB4157B8B1ECE04043249A3D379AC112E5B9AF44E721E148D88A942744CF56A06B92D28A0DB950FE4CED2B41A0BD38BCE7D0BE1055CF5DE38F2A588C2C9A79A75011058C320A7B661C6CE1C36C7D870758307E5D2CF07D9B6E8D529779B6B2910DD17B6766A7EFEE215A98CAC300F2827DB

*B:*

3A7EF2554E8940FA9B93B2A5E822CC7BB262F4A14159E4318CAE3ABF5AEB1022EC6D01DEFAB48B528868679D649B445A753684C13F6C3ADBAB059D635A2882090FC166EA9F0AAACD16A062149E4A0952F7FAAB14A0E9D3CB0BE9200DBD3B0342496421826919148E617AF1DB66978B1FCD28F8408506B79979CCBCC7F7E5FDE7

*N:*

269D7722EA018F2AC35C5A3517AA06EAA1949059AE8240428BBFD0A8BE6E2EBF91223991F80D7413D6B2EB213E7122710EDEC617460FA0191F3901604619972018EBEF22D81AED9C56424014CADCC2CCDEE67D36A54BFC500230CA6693ABA057B374746622341ED6D52FE5A79E6860F54F197791B3FEF49FD534CB2C675B6BDB

*C:*

87D6D58D3991D536544389CEFA72FD0EBED75B2EBDC2C79BC3717793108F0952011E7E2D7040FFFB32F10BEB8ED0A485026B6860020B230128A8222B0525A6888942FB01C537800BF25D6F021D4B99D3CBD6DF9055FA22F91A6CFC4FDFC408AEF78F6418D3CE4E20EC7888B61BAE3D73C27C257CCA905DE0353C3A7CFFD9FE15

(A + B) \* C mod N:

25d22a3888644e804955e9b3381212ab7e0a1a7c0388ddf05cfc15485bf95920e3ac76cc4875f5a789ba94ab5d5ed65c77fe0b79f4bcee24e2a544d75ea990cd008594ecc48cbb7650451015fdc787c16d38850f9e7442bd78f057000f2375fb42af4f19eda259b714d1383db3fb6a35e962ddc894684e607f1d1efa2d2410ea

C \* (A + B) mod N:

25d22a3888644e804955e9b3381212ab7e0a1a7c0388ddf05cfc15485bf95920e3ac76cc4875f5a789ba94ab5d5ed65c77fe0b79f4bcee24e2a544d75ea990cd008594ecc48cbb7650451015fdc787c16d38850f9e7442bd78f057000f2375fb42af4f19eda259b714d1383db3fb6a35e962ddc894684e607f1d1efa2d2410ea

(A \* C + B \* C) mod N:

25d22a3888644e804955e9b3381212ab7e0a1a7c0388ddf05cfc15485bf95920e3ac76cc4875f5a789ba94ab5d5ed65c77fe0b79f4bcee24e2a544d75ea990cd008594ecc48cbb7650451015fdc787c16d38850f9e7442bd78f057000f2375fb42af4f19eda259b714d1383db3fb6a35e962ddc894684e607f1d1efa2d2410ea

*Тест Б.2 (num = 15)*

num \* A:

19ded65fcff75e599a4456bf5883b96580a6ea142c83a9a18d288e1415b6be01975a27ee4ee51ca725957fdc60f95ca2e205f9694aed2012584b15daf0feb10bb69cb7492f653a2411cae327d75675708f5f5368e158e905600660da619d8695fb2a116a810fd6d43416726a89f8ee610d88a391c944e412a4550297c811c9af

Sum A num times:

19ded65fcff75e599a4456bf5883b96580a6ea142c83a9a18d288e1415b6be01975a27ee4ee51ca725957fdc60f95ca2e205f9694aed2012584b15daf0feb10bb69cb7492f653a2411cae327d75675708f5f5368e158e905600660da619d8695fb2a116a810fd6d43416726a89f8ee610d88a391c944e412a4550297c811c9af

# Код програми

#include <iostream>

#include <string>

#include <sstream>

#include <array>

#include <chrono>

using namespace std;

// ----------------------- CONST ZONE -----------------------

const unsigned int len = 1024, max\_len = 2049;

// ----------------------- CONVERTATION ZONE -----------------------

unsigned int\* CreateConstant(unsigned int number, unsigned int w)

{

unsigned int\* A = new unsigned int[max\_len];

fill(&A[0], &A[max\_len], 0);

A[0] = number & ((1i64 << w) - 1);

return A;

}

// Function that gets value of the first non-null bit of a number

int GetFirstNotNullBit(unsigned int A[max\_len])

{

int counter = 0, temp = max\_len;

while (A[max\_len - 1 - counter] == 0)

{

counter++;

}

return max\_len - counter - 1;

}

void CopyArray(unsigned int A[max\_len], unsigned int B[max\_len])

{

for (int i = 0; i < max\_len; i++)

{

B[i] = A[i];

}

}

void PrintArray(unsigned int A[max\_len])

{

cout << endl;

for (int i = GetFirstNotNullBit(A); i >= 0; i--)

{

cout << A[i];

}

cout << endl;

}

// Function that converts hex string into big number

unsigned int\* ConvertHexToBigNumber(string s, int w)

{

int i = 0, counter = 0;

unsigned int \*A = new unsigned int[max\_len];

fill(&A[0], &A[max\_len], 0);

unsigned int num = 0;

string sub = "", temp = s;

if (w >= 4)

{

while (temp.size() % (w / 4) != 0)

{

temp.insert(0, "0");

}

while (temp.size() != 0)

{

sub = temp.substr(temp.size() - (w / 4), w / 4);

temp.erase(temp.size() - (w / 4), w / 4);

sscanf\_s(sub.c\_str(), "%x", &num);

A[i] = num;

i++;

}

return A;

}

if (w == 1)

{

int calc = 0;

for (int i = s.length() - 1; i >= 0; i--)

{

char temp = s[i];

int number = strtol(&temp, NULL, 16);

for (int j = 0; j < 4; j++)

{

A[calc] = ((number >> j) & 1);

calc++;

}

}

return A;

}

return A;

}

// Function that converts int 2^w to hex

string ConvertIntToHex(unsigned int n, int w)

{

stringstream ss;

string result;

ss << hex << n;

result = ss.str();

while (result.length() != (w / 4))

{

result.insert(0, "0");

}

return result;

}

// Function that converts binary number to hex

string ConvertBinNumberToHex(unsigned int A[max\_len])

{

stringstream ss;

string result, temp;

int A\_len = GetFirstNotNullBit(A) + 1;

int value = 0;

while (A\_len % 4 != 0)

{

A\_len++;

}

for (int i = 0; i < A\_len; i = i + 4)

{

value = 0;

for (int j = 0; j <= 3; j++)

{

value = value + (A[i+j] \* (1 << j));

}

ss << hex << value;

temp = ss.str();

result.insert(0, temp);

ss.str("");

}

while (result[0] == '0')

{

result.erase(0, 1);

}

return result;

}

// Function that coonverts decimal 2^w number to hex

string ConvertBigNumberToHex(unsigned int A[max\_len], int w)

{

if (w >= 4)

{

int counter = 0;

string temp = "", result = "";

int A\_len = GetFirstNotNullBit(A);

while (counter != A\_len + 1)

{

temp = ConvertIntToHex(A[counter], w);

result.insert(0, temp);

counter++;

}

while (result[0] == '0')

{

result.erase(0, 1);

}

if (result == "") result = "0";

return result;

}

if (w == 1)

{

return ConvertBinNumberToHex(A);

}

return "0";

}

// ----------------------- OPERATIONS ZONE -----------------------

// Function calculating A + B

unsigned int\* LongAdd(unsigned int A[max\_len], unsigned int B[max\_len], unsigned int w, bool SelfCulc)

{

unsigned int carry = 0;

const unsigned long long base = 1i64 << w;

unsigned long long temp = 0;

unsigned int\* C = new unsigned int[max\_len];

fill(&C[0], &C[max\_len], 0);

for (int i = 0; i < max\_len; i++)

{

temp = (unsigned long long)A[i] + B[i] + carry;

C[i] = temp & (base - 1);

carry = temp >> w;

}

if (SelfCulc)

{

delete[]A;

A = NULL;

return C;

}

return C;

}

// Function comparing A and B

int LongCmp(unsigned int A[max\_len], unsigned int B[max\_len])

{

int i = max\_len - 1;

while (A[i] == B[i] && i >= 0)

{

i--;

}

if (i == -1) return 0;

else

{

if (A[i] > B[i]) return 1;

else return -1;

}

}

// Function calculating A - B

unsigned int\* LongSub(unsigned int A[max\_len], unsigned int B[max\_len], unsigned int w, bool SelfCulc)

{

int borrow = 0, temp = 0;

unsigned int\* C = new unsigned int[max\_len];

fill(&C[0], &C[max\_len], 0);

int check = LongCmp(A, B);

if (check == -1)

{

if (SelfCulc)

{

delete[]A;

A = NULL;

return C;

}

return C;

}

else

{

int A\_len = GetFirstNotNullBit(A);

for (int i = 0; i < A\_len + 1; i++)

{

temp = A[i] - B[i] - borrow;

if (temp >= 0)

{

C[i] = temp;

borrow = 0;

}

else

{

C[i] = (1i64 << w) + temp;

borrow = 1;

}

}

if (SelfCulc)

{

delete[]A;

A = NULL;

return C;

}

return C;

}

}

// ----------------------- Section of A \* B -----------------------

// Function calculating A \* b, where is b - is one digit

unsigned int\* LongMulOneDigit(unsigned int A[max\_len], unsigned int b, unsigned int w, bool SelfCulc)

{

unsigned int carry = 0;

unsigned int \*C = new unsigned int[max\_len];

fill(&C[0], &C[max\_len], 0);

int A\_len = GetFirstNotNullBit(A);

for (int i = 0; i < A\_len + 1; i++)

{

unsigned long long temp = (unsigned long long)A[i] \* b + carry;

C[i] = temp & ((1i64 << w) - 1);

carry = temp >> w;

}

if (A\_len == max\_len)

{

if (SelfCulc)

{

delete[]A;

A = NULL;

return C;

}

return C;

}

else

{

C[A\_len + 1] = carry;

if (SelfCulc)

{

delete[]A;

A = NULL;

return C;

}

return C;

}

}

// Function that moves \*index\* bits to higher places

void LongShiftBitsToHigh(unsigned int A[max\_len], int index)

{

for (int i = max\_len - 1 - index; i >= 0 ; i--)

{

A[i + index] = A[i];

}

for (int i = 0; i <= index - 1; i++)

{

A[i] = 0;

}

}

// Function that moves \*index\* bits to lower places

void LongShiftBitsToLow(unsigned int A[max\_len], int index)

{

for (int i = index; i <= max\_len - 1; i++)

{

A[i - index] = A[i];

}

for (int i = max\_len - index; i <= max\_len - 1; i++)

{

A[i] = 0;

}

}

// Function calculating A \* B

unsigned int\* LongMul(unsigned int A[max\_len], unsigned int B[max\_len], unsigned int w, bool SelfCulc)

{

unsigned int\* C = new unsigned int[max\_len];

fill(&C[0], &C[max\_len], 0);

for (int i = 0; i < max\_len; i++)

{

unsigned int\* temp = LongMulOneDigit(A, B[i], w, false);

LongShiftBitsToHigh(temp, i);

C = LongAdd(C, temp, w, true);

delete[]temp;

temp = NULL;

}

if (SelfCulc)

{

delete[]A;

A = NULL;

return C;

}

return C;

}

// ----------------------- Section of A / B -----------------------

unsigned int\*\* LongDivMod(string s\_A, string s\_B)

{

unsigned int\* A\_bin = new unsigned int[max\_len];

unsigned int\* B\_bin = new unsigned int[max\_len];

unsigned int\* C\_bin = new unsigned int[max\_len];

unsigned int\* R = new unsigned int[max\_len];

unsigned int\* Q = new unsigned int[max\_len];

unsigned int\*\* result = new unsigned int\*[2];

fill(&Q[0], &Q[max\_len], 0);

A\_bin = ConvertHexToBigNumber(s\_A, 1);

B\_bin = ConvertHexToBigNumber(s\_B, 1);

int k = GetFirstNotNullBit(B\_bin);

CopyArray(A\_bin, R);

while (LongCmp(R, B\_bin) == 1 || LongCmp(R, B\_bin) == 0)

{

int t = GetFirstNotNullBit(R);

CopyArray(B\_bin, C\_bin);

LongShiftBitsToHigh(C\_bin, t - k);

if (LongCmp(R, C\_bin) == -1)

{

t = t - 1;

CopyArray(B\_bin, C\_bin);

LongShiftBitsToHigh(C\_bin, t - k);

}

R = LongSub(R, C\_bin, 1, true);

Q[t - k] = 1;

}

result[0] = Q;

result[1] = R;

delete[]A\_bin, delete[]B\_bin, delete[]C\_bin;

A\_bin = NULL, B\_bin = NULL, C\_bin = NULL;

return result;

}

unsigned int\* LongSquare(unsigned int A[max\_len], unsigned int w, bool SelfCulc)

{

//unsigned int\* temp = new unsigned int[max\_len];

unsigned int\* result = new unsigned int[max\_len];

//CopyArray(A, temp);

//CopyArray(LongMul(A, temp, w), result);

result = LongMul(A, A, w, false);

//delete[]temp;

if (SelfCulc)

{

delete[]A;

A = NULL;

return result;

}

//temp = NULL;

return result;

}

unsigned int\* LongPower(unsigned int A[max\_len], unsigned int B[max\_len], unsigned int w, bool SelfCulc)

{

unsigned int\* C = CreateConstant(1, w);

int B\_len = GetFirstNotNullBit(B);

for (int i = 0; i < B\_len + 1; i++)

{

if (B[i] == 1)

{

//CopyArray(LongMul(C, A, w), C);

C = LongMul(C, A, w, true);

}

A = LongSquare(A, w, true);

//CopyArray(LongSquare(A, w), A);

}

if (SelfCulc)

{

delete[]A;

A = NULL;

return C;

}

return C;

}

// ------------------------- LAB 2 -------------------------

string LongGcd(string s\_A, string s\_B, unsigned int w)

{

unsigned int\* A\_bin = ConvertHexToBigNumber(s\_A, 1);

unsigned int\* B\_bin = ConvertHexToBigNumber(s\_B, 1);

unsigned int\* zero = new unsigned int[max\_len];

fill(&zero[0], &zero[max\_len], 0);

if (LongCmp(A\_bin, B\_bin) == 0)

{

delete[]A\_bin, delete[]B\_bin, delete[]zero;

A\_bin = NULL, B\_bin = NULL, zero = NULL;

return s\_A;

}

if (LongCmp(A\_bin, zero) == 0)

{

delete[]A\_bin, delete[]B\_bin, delete[]zero;

A\_bin = NULL, B\_bin = NULL, zero = NULL;

return s\_B;

}

if (LongCmp(B\_bin, zero) == 0)

{

delete[]A\_bin, delete[]B\_bin, delete[]zero;

A\_bin = NULL, B\_bin = NULL, zero = NULL;

return s\_A;

}

unsigned int\* d = CreateConstant(1, 1);

unsigned int\* temp = new unsigned int[max\_len];

string result = "";

while ((LongCmp(A\_bin, zero) != 0) && (LongCmp(B\_bin, zero) != 0))

{

while ((A\_bin[0] == 0) && (B\_bin[0] == 0))

{

LongShiftBitsToLow(A\_bin, 1);

LongShiftBitsToLow(B\_bin, 1);

LongShiftBitsToHigh(d, 1);

}

while ((A\_bin[0] == 0) && (LongCmp(A\_bin, zero) != 0))

{

LongShiftBitsToLow(A\_bin, 1);

}

while ((B\_bin[0] == 0) && (LongCmp(B\_bin, zero) != 0))

{

LongShiftBitsToLow(B\_bin, 1);

}

if (LongCmp(A\_bin, B\_bin) == 1)

{

A\_bin = LongSub(A\_bin, B\_bin, 1, true);

}

else

{

B\_bin = LongSub(B\_bin, A\_bin, 1, true);

}

}

d = LongMul(d, A\_bin, 1, true);

result = ConvertBinNumberToHex(d);

delete[]A\_bin, delete[]B\_bin, delete[]temp, delete[]zero, delete[]d;

A\_bin = NULL, B\_bin = NULL, temp = NULL, zero = NULL, d = NULL;

return result;

}

string LongLcm(string s\_A, string s\_B, int w)

{

string g = LongGcd(s\_A, s\_B, w);

unsigned int\* A\_decimal = ConvertHexToBigNumber(s\_A, w);

unsigned int\* B\_decimal = ConvertHexToBigNumber(s\_B, w);

unsigned int\* gcd = ConvertHexToBigNumber(g, w);

if (g == "1")

{

A\_decimal = LongMul(A\_decimal, B\_decimal, w, true);

string res = ConvertBigNumberToHex(A\_decimal, w);

delete[]A\_decimal, delete[]B\_decimal, delete[]gcd;

A\_decimal = NULL, B\_decimal = NULL, gcd = NULL;

return res;

}

else

{

unsigned int\* temp = new unsigned int[max\_len];

temp = LongMul(A\_decimal, B\_decimal, w, false);

unsigned int\* lcm = LongDivMod(ConvertBigNumberToHex(temp, w), g)[0];

string res = ConvertBigNumberToHex(lcm, 1);

delete[]A\_decimal, delete[]B\_decimal, delete[]gcd, delete[]temp, delete[]lcm;

A\_decimal = NULL, B\_decimal = NULL, gcd = NULL, temp = NULL, lcm = NULL;

return res;

}

}

string BarrettReduction(string s\_x, string s\_n, unsigned int w)

{

unsigned int \*x = ConvertHexToBigNumber(s\_x, 1);

unsigned int \*n = ConvertHexToBigNumber(s\_n, 1);

int k = GetFirstNotNullBit(n);

unsigned int\* beta = CreateConstant(1, 1);

LongShiftBitsToHigh(beta, 2 \* k);

string s\_beta = ConvertBinNumberToHex(beta);

delete[]beta;

beta = NULL;

unsigned int \*\*temp = LongDivMod(s\_beta, s\_n);

unsigned int \*mu = temp[0];

unsigned int\* q = new unsigned int[max\_len];

CopyArray(x, q);

LongShiftBitsToLow(q, k - 1);

q = LongMul(q, mu, 1, true);

LongShiftBitsToLow(q, k + 1);

unsigned int \*help = LongMul(q, n, 1, false);

unsigned int \*r = LongSub(x, help, 1, false);

while (LongCmp(r, n) >= 0)

{

r = LongSub(r, n, 1, true);

}

delete[]x, delete[]n, delete[]help, delete[]q, delete[]temp[0], delete[]temp[1], delete[]temp;

help = NULL, x = NULL, n = NULL, q = NULL, temp = NULL;

string answer = ConvertBinNumberToHex(r);

delete[]r;

return answer;

}

unsigned int\* LongAddMod(unsigned int A[max\_len], unsigned int B[max\_len], unsigned int N[max\_len], int w, bool SelfCulc)

{

unsigned int \*ad = LongAdd(A, B, w, false);

string s\_result = BarrettReduction(ConvertBigNumberToHex(ad, w), ConvertBigNumberToHex(N, w), w);

delete[]ad;

ad = NULL;

if (SelfCulc)

{

delete[]A;

A = NULL;

return ConvertHexToBigNumber(s\_result, w);

}

return ConvertHexToBigNumber(s\_result, w);

}

unsigned int\* LongSubMod(unsigned int A[max\_len], unsigned int B[max\_len], unsigned int N[max\_len], int w, bool SelfCulc)

{

unsigned int \*sub = LongSub(A, B, w, false);

string s\_result = BarrettReduction(ConvertBigNumberToHex(sub, w), ConvertBigNumberToHex(N, w), w);

delete[]sub;

sub = NULL;

if (SelfCulc)

{

delete[]A;

A = NULL;

return ConvertHexToBigNumber(s\_result, w);

}

return ConvertHexToBigNumber(s\_result, w);

}

unsigned int\* LongMulMod(unsigned int A[max\_len], unsigned int B[max\_len], unsigned int N[max\_len], int w, bool SelfCulc)

{

unsigned int \*mul = LongMul(A, B, w, false);

string s\_result = BarrettReduction(ConvertBigNumberToHex(mul, w), ConvertBigNumberToHex(N, w), w);

delete[]mul;

mul = NULL;

if (SelfCulc)

{

delete[]A;

A = NULL;

return ConvertHexToBigNumber(s\_result, w);

}

return ConvertHexToBigNumber(s\_result, w);

}

unsigned int\* LongSquareMod(unsigned int A[max\_len], unsigned int N[max\_len], int w, bool SelfCulc)

{

unsigned int \*sq = LongSquare(A, w, false);

string s\_result = BarrettReduction(ConvertBigNumberToHex(sq, w), ConvertBigNumberToHex(N, w), w);

delete[]sq;

sq = NULL;

if (SelfCulc)

{

delete[]A;

A = NULL;

return ConvertHexToBigNumber(s\_result, w);

}

return ConvertHexToBigNumber(s\_result, w);

}

unsigned int\* LongPowerMod(unsigned int A[max\_len], unsigned int B[max\_len], unsigned int N[max\_len], int w, bool SelfCulc)

{

unsigned int\* C = CreateConstant(1, w);

int B\_len = GetFirstNotNullBit(B);

for (int i = 0; i < B\_len + 1; i++)

{

if (B[i] == 1)

{

//CopyArray(LongMulMod(C, A, N, w), C);

C = LongMulMod(C, A, N, w, true);

}

//CopyArray(LongSquareMod(A, N, w), A);

A = LongSquareMod(A, N, w, true);

}

if (SelfCulc)

{

delete[]A;

A = NULL;

return C;

}

return C;

}

// ----------------------- MAIN ZONE -----------------------

int main()

{

// LAB-2 Section --------------------------------------------------------------

int i = 0;

const unsigned int base = 32, bin\_base = 1;

unsigned int \*A\_decimal, \*B\_decimal, \*N\_decimal;

string s\_A, s\_B, s\_N;

cout << "Enter number A: " << endl;

cin >> s\_A;

A\_decimal = ConvertHexToBigNumber(s\_A, base);

cout << "Enter number B: " << endl;

cin >> s\_B;

B\_decimal = ConvertHexToBigNumber(s\_B, base);

cout << "Enter number N: " << endl;

cin >> s\_N;

N\_decimal = ConvertHexToBigNumber(s\_N, base);

unsigned int \*B\_bin = ConvertHexToBigNumber(s\_B, bin\_base);

auto start = std::chrono::system\_clock::now();

string s\_d = LongGcd(s\_A, s\_B, base);

auto end = std::chrono::system\_clock::now();

std::chrono::duration<double> diff = end - start;

cout << "gcd(A, B): " << endl << s\_d << endl;

cout << "Time: " << diff.count() << " sec" << endl;

start = std::chrono::system\_clock::now();

string lcm = LongLcm(s\_A, s\_B, base);

end = std::chrono::system\_clock::now();

diff = end - start;

cout << "lcm(A, B):" << endl << lcm << endl;

cout << "Time: " << diff.count() << " sec" << endl;

unsigned int \*abAdd, \*abSub, \*abMul, \*aSq, \*abPower;

string sAbAdd, sAbSub, sAbMul, sASq, sAbPower;

start = std::chrono::system\_clock::now();

abAdd = LongAddMod(A\_decimal, B\_decimal, N\_decimal, base, false);

end = std::chrono::system\_clock::now();

diff = end - start;

sAbAdd = ConvertBigNumberToHex(abAdd, base);

cout << "A + B mod N: " << endl << sAbAdd << endl;

cout << "Time: " << diff.count() << " sec" << endl;

start = std::chrono::system\_clock::now();

abSub = LongSubMod(A\_decimal, B\_decimal, N\_decimal, base, false);

end = std::chrono::system\_clock::now();

diff = end - start;

sAbSub = ConvertBigNumberToHex(abSub, base);

cout << "A - B mod N: " << endl << sAbSub << endl;

cout << "Time: " << diff.count() << " sec" << endl;

start = std::chrono::system\_clock::now();

abMul = LongMulMod(A\_decimal, B\_decimal, N\_decimal, base, false);

end = std::chrono::system\_clock::now();

diff = end - start;

sAbMul = ConvertBigNumberToHex(abMul, base);

cout << "A \* B mod N: " << endl << sAbMul << endl;

cout << "Time: " << diff.count() << " sec" << endl;

start = std::chrono::system\_clock::now();

aSq = LongSquareMod(A\_decimal, N\_decimal, base, false);

end = std::chrono::system\_clock::now();

diff = end - start;

sASq = ConvertBigNumberToHex(aSq, base);

cout << "A ^ 2 mod N: " << endl << sASq << endl;

cout << "Time: " << diff.count() << " sec" << endl;

start = std::chrono::system\_clock::now();

abPower = LongPowerMod(A\_decimal, B\_bin, N\_decimal, base, false);

end = std::chrono::system\_clock::now();

diff = end - start;

sAbPower = ConvertBigNumberToHex(abPower, base);

cout << "A ^ B mod N: " << endl << sAbPower << endl;

cout << "Time: " << diff.count() << " sec" << endl;

delete[]abAdd, delete[]abSub, delete[]abMul, delete[]aSq, delete[]abPower;;

abAdd = NULL, abSub = NULL, abMul = NULL, aSq = NULL, abPower = NULL;

// TEST-1: (A+B)\*C = C\*(A+B) = A\*C + B\*C

/\*string s\_C;

unsigned int \*sumAB, \*C\_decimal;

cout << "Enter number C: " << endl;

cin >> s\_C;

C\_decimal = ConvertHexToBigNumber(s\_C, base);

sumAB = LongAdd(A\_decimal, B\_decimal, base, false);

cout << "(A + B) \* C mod N: " << endl << ConvertBigNumberToHex(LongMulMod(C\_decimal, sumAB, N\_decimal, base, false), base) << endl;

cout << "C \* (A + B) mod N: " << endl << ConvertBigNumberToHex(LongMulMod(sumAB, C\_decimal, N\_decimal, base, false), base) << endl;

cout << "(A \* C + B \* C) mod N: " << endl << ConvertBigNumberToHex(LongAddMod(LongMul(A\_decimal, C\_decimal, base, false), LongMul(B\_decimal, C\_decimal, base, false), N\_decimal, base, false), base) << endl;

\*/

// TEST-2: A \* N = A + A + ... + A mod m

/\*int num = 15;

unsigned int \*test = CreateConstant(num, base);

unsigned int \*result1 = LongMulMod(A\_decimal, test, N\_decimal, base, false);

unsigned int \*result2 = new unsigned int[max\_len];

CopyArray(A\_decimal, result2);

for (int i = 1; i < num; i++)

{

result2 = LongAddMod(result2, A\_decimal, N\_decimal, base, true);

}

cout << "num \* A: " << endl << ConvertBigNumberToHex(result1, base) << endl;

cout << "Sum A num times: " << endl << ConvertBigNumberToHex(result2, base) << endl;\*/

// TEST-3: A^PHI(N) = 1 mod N

/\*unsigned int\* test2 = LongPower(A\_decimal, N\_decimal, base, false);

unsigned int\* one = CreateConstant(1, base);

unsigned int\* phi = LongSub(N\_decimal, one, base, false);

string phi\_str = ConvertBigNumberToHex(phi, base);

unsigned int\* bin\_phi = ConvertHexToBigNumber(phi\_str, bin\_base);

unsigned int\* test2 = LongPowerMod(A\_decimal, bin\_phi, N\_decimal, base, false);

string test2str = ConvertBigNumberToHex(test2, base);

string result3 = BarrettReduction(test2str, s\_N, base);

cout << "A^PHI(N) mod N: " << endl << result3 << endl;\*/

}