МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»

Кафедра інформаційних систем і мереж

Лабораторна робота №2

з дисципліни: «Екстремальне програмування»

Ключовий обмін Діффі-Хеллмана

|  |
| --- |
| Виконав:  Студент групи ЕКП-ІСМ-301  Гасій В. Ю.  Прийняв:  Щербак С. С. |

Львів – 2020

**Тест Рабіна-Міллера**

Вибрати для перевірки випадкове число *p*. Обчислити як найбільше число ділення *р-1* на 2, тобто *b* - найбільший показник ступеня числа 2, на яку ділиться *p-1* без остачі. Потім обчислити *m* як таке, що *p =1+\*m*

1. Вибрати випадкове число *a*, менше за *р*.

2. Встановити *j = 0* та *z = mod p*.

3. Якщо *z = 1* чи *z = p-1*, то *p* проходить перевірку і може бути простим числом.

4. Якщо *j>0* і *z=1*, то *p* не є простим числом.

5. Встановити *j = j+1*. Якщо *j<b* і *z<p-1*, встановити *z = mod p* і повернутися до пункту 4. Якщо *z=p-1*, то p проходить перевірку і може бути простим числом.

6. Якщо j = b і *z ≠ p-1*, то *p* не є простим числом.

Повторити цю перевірку потрібно *t* раз. Доведено, що в цьому тесті ймовірність проходження перевірки складеним числом зменшується скоріше, ніж в інших. Гарантується, що 75% можливих значень *а* виявляться показниками того, що вибране число *р* – складене. Це означає, що ймовірність прийняти складене число *p* за просте не перевищує величини .

**Завдання**

Реалізувати тест Рабіна-Міллера.

**Розв’язання**

Main.cpp

#include "Include.h"

using boost::multiprecision::cpp\_int;

int main()

{

cpp\_int num = create\_number(80);

std::cout << num << std::endl;

std::cout << Rabin\_Miller(num, 80, 5) << std::endl;

system("pause");

return 0;

}

Include.h

#pragma once

#include <iostream>

#include <boost/multiprecision/cpp\_int.hpp>

#include "Prime\_Number.h"

Prime\_Number.h

#pragma once

#include <iostream>

#include <cmath>

#include <boost/multiprecision/cpp\_int.hpp>

#include <ctime>

#include <vector>

using boost::multiprecision::cpp\_int;

cpp\_int modular\_pow(cpp\_int base, cpp\_int exponent, cpp\_int modulus);

cpp\_int create\_number(int n);

cpp\_int convert\_binary\_to\_decimal(cpp\_int number);

bool Rabin\_Miller(const cpp\_int &p, int n , int t );

Prime\_Number.cpp

#include "Prime\_Number.h"

using boost::multiprecision::cpp\_int;

cpp\_int modular\_pow(cpp\_int base, cpp\_int exponent, cpp\_int modulus)

{

cpp\_int result = 1;

while (exponent > 0)

{

if (exponent % 2 == 1)

{

result = (result \* base) % modulus;

}

exponent = exponent >> 1;

base = (base \* base) % modulus;

}

return result;

}

cpp\_int create\_number(int n)

{

srand(time(NULL));

cpp\_int number;

std::string temp = "\0";

temp += "1";

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

{

temp += std::to\_string(rand() % 2);

}

number = convert\_binary\_to\_decimal(cpp\_int(temp));

return number;

}

cpp\_int convert\_binary\_to\_decimal(cpp\_int number)

{

cpp\_int decimalNumber = 0, remainder;

int i = 0;

while (number != 0)

{

remainder = number % 10;

number /= 10;

decimalNumber += remainder \* cpp\_int(pow(2, i));

++i;

}

return decimalNumber;

}

bool Rabin\_Miller(const cpp\_int &p, int n = 0, int t = 0)

{

int b = 1;

while ((p - 1) % cpp\_int(pow(2, b)) == 0)

{

b++;

}

b -= 1;

cpp\_int m = (p - 1) / cpp\_int(pow(2, b));

cpp\_int a;

cpp\_int z;

srand(time(NULL));

for (int i = 0; i < t; i++)

{

a = create\_number(rand() % (n - 2) + 2);

z = modular\_pow(a, m, p);

if (z == 1 || z == (p - 1))

{

continue;

}

for (int j = 0; j < b - 1; j++)

{

z = modular\_pow(z, 2, p);

if (z == 1)

{

return false;

}

if (z == (p - 1))

{

break;

}

}

if (b == i && z != (p - 1))

{

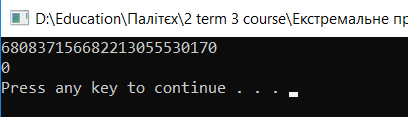
return false;

}

}

return true;

}



**Висновок**: на цій лабораторній роботі було реалізовано тест Рабіна-Міллера.