ОСНОВЫ ТЕОРИИ ЧИСЕЛ И ИХ ИСПОЛЬЗОВАНИЕ В КРИПТОГРАФИИ

**Вариант 2**

Выполнил: Горощеня Владислав Сергеевич, 3 курс 4 группа 1 подгруппа

2024

**Ход работы**

Теория чисел представляет собой раздел математики, изучающие натуральные числа и иные похожие величины. Натуральные числа есть подмножество целых чисел (набора действительных чисел без дробной части). Каждое натуральное число, большее 1, делится по крайней мере, на 1 и на само себя, причём если нет других делителей, то такое число является ***простым***. Согласно третьему свойству простых чисел, *наименьший простой делитель составного числа n не превышает √n, поэтому для проверки простоты числа достаточно проверить его делимость на 2 и на все простые числа, не превосходящие √n*. Сложность решения задачи разложения больших чисел на простые сомножители, известной как «проблема факторизации», определяет криптостойкость некоторых алгоритмов асимметричной криптографии, в частности алгоритма RSA. В листинге 1 приведена реализация функции, определяющей, является ли число простым.

Листинг 1. Реализация функции определения простого числа.

|  |
| --- |
| bool IsPrime(int n) {  if (n < 2) {  return false;  }  for (int i = 2; i \* i <= n; i++) {  if (n % i == 0) {  return false;  }  }  return true;  } |

Также необходимо было проверить с помощью программы L\_PROST.EXE количество простых чисел от 2 до 553 и от 521 до 553. Результат подсчёта числа простых чисел в промежутке от 2 до 553 приведён на рисунке 1.

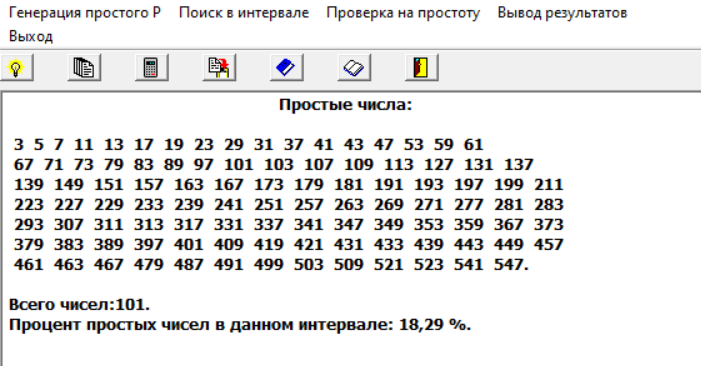


Рисунок 1 – простые числа от 2 до 553

Результат подсчёта количества простых чисел от 521 до 553 приведён на рисунке 2.

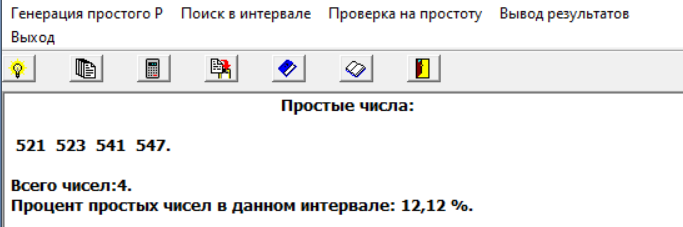


Рисунок 2 - простые числа от 521 до 553

Для проверки результатов были написана перегружаемая функция. Её реализации представлены в листингах 2 и 3.

Листинг 2. Реализация функции получения простых чисел от 2 до 553

|  |
| --- |
| vector<int>GetPrimes(int m, int n) {  if (m > n) {  swap(m, n);  }  vector<int>vec;  for (int i = m; i < n; i++) {  if (IsPrime(i)) {  vec.push\_back(i);  }  }  return vec;  } |

Листинг 3. Реализация функции получения простых чисел от 521 до 553

|  |
| --- |
| vector<int>GetPrimes(int n) {  vector<int>vec;  for (int i = 2; i < n; i++) {  if (IsPrime(i)) {  vec.push\_back(i);  }  }  return vec;  } |

Согласно второму свойству простых чисел, *существует примерно чисел, меньших* . Это также можно и определить с помощью решета Эратосфена. В листинге 4 приведена реализация подсчёта примерного числа простых чисел, меньше чем .

Листинг 4. Функция подсчёта примерного числа простых чисел до n.

|  |
| --- |
| double GetApproximativeCountOfPrimes(int n) {  return n / log(n);  } |

Результат проверки данных приведён на рисунке 3.

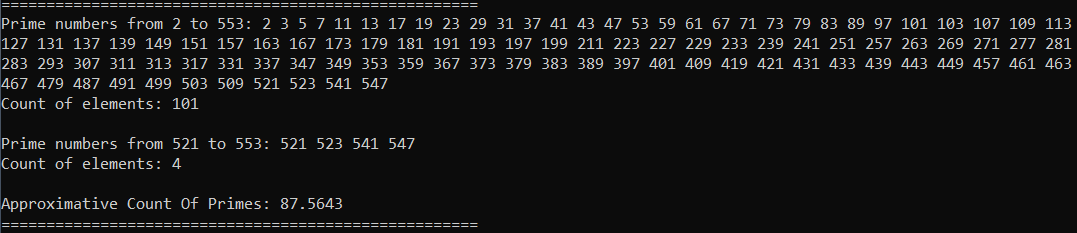


Рисунок 3 – проверка данных

Основная теорема арифметики заключается в том, что *всякое натуральное число, кроме 1, можно представить в виде произведения простых множителей*. Такая форма представления называется ***канонической***.

В листинге 5 и 6 приведены функции определения составляющих множителей и формирования канонической формы соответственно.

Листинг 5. Функция определения составляющих множителей

|  |
| --- |
| vector<int> CanonicalForm(int par) {  vector<int>canon;  int div = 2;  while (par > 1) {  while (par % div == 0) {  canon.push\_back(div);  par /= div;  }  div++;  }  return canon;  } |

Листинг 6. Функция формирования канонической формы

|  |
| --- |
| string PrintCanonicalForm(vector<int>a) {  string str = "";  for (int i = 0; i < a.size(); i++) {  if (i == a.size() - 1) {  str += to\_string(a[i]);  return str;  }  str += (to\_string(a[i])+"\*");  }  return str;  } |

Результат преобразования в каноническую форму приведён на рисунке 4.

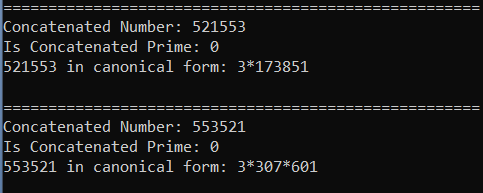


Рисунок 4 – результат преобразования в каноническую форму

Здесь для составления канонической формы использовалась конкатенация чисел 521 и 553. Реализация функции конкатенации приведён в листинге 6.

Листинг 6. Функция конкатенации.

|  |
| --- |
| int GetConcatenatedNumber(int n, int m) {  string nStr = to\_string(n);  string mStr = to\_string(m);  return stoi(nStr + mStr);  } |

НОД (наибольший общий делитель) – наибольшее целое число, которое может делиться как на , так и . Для реализации подсчёта НОД использовался алгоритм Евклида, в основе которого лежит определение 5. В соответствии с этим осуществляется цепочка измерений . Реализация алгоритма Евклида приведена в листинге 7.

Листинг 7. Алгоритм Евклида

|  |
| --- |
| int GCD(int a, int b) {  if (a == 0) {  return b;  }  else if (b == 0) {  return a;  }  else if (a == 1 || b == 1) {  return 1;  }  while (a != 0 && b != 0) {  if (a > b) {  a -= b;  }  else {  b -= a;  }  }  return (a > b) ? a : b;  } |

Результат НОД(521,553) представлен на рисунке 5.



Рисунок 5 – НОД(521, 553)