**Дисциплина «Алгоритмы решения прикладных задач»**
**Рабочая тетрадь 2.**
**Числовые алгоритмы: поиск наибольшего общего делителя, факторизация чисел, поиск простых чисел, псевдослучайные числа**

**Теоретический материал**

**Алгоритмы поиска наибольшего общего делителя**

НОД – наибольший общий делитель, – число, делящее без остатка пару чисел и являющееся наибольшим из возможных делителей. Например: НОД(8, 6)=2; НОД(12, 8)=4. Очень важное понятие, необходимость в вычислении которого возникает достаточно часто. Простой расчетной формулы для НОД не существует. Решение полным перебором возможно, но для больших чисел необходимо большое количество вычислительных операций.

Хорошее решение проблемы поиска наибольшего общего делителя двух чисел нашел еще Евклид. В самом простом случае алгоритм Евклида применяется к паре положительных целых чисел и формирует новую пару, которая состоит из меньшего числа и разницы между большим и меньшим числом. Процесс повторяется, пока числа не станут равными. Найденное число и есть наибольший общий делитель исходной пары. Блок-схема алгоритма выглядит следующим образом:


Для уменьшения количества вычислительных операций новая пара положительных чисел формируется не в результате вычитания из большего числа меньшего, а в результате деления большего числа на меньшее. В результате новая пара положительных чисел формируется из делителя и остатка от деления. Как только остаток от деления становится равным нулю, вычисления прекращаются.

Искомый наибольший общий делитель – это делитель на последнем шаге вычислений. Примеры для обоих подходов (вычитание и деление):

Найти наибольший общий делитель для чисел 128 и 96.

128 – 96 = 32

96 – 32 = 64

64 – 32 = 32

32 – 32 = 0

128 / 96 = 1 (остаток 32)

96 / 32 = 3

Ответ: 32

*Пример 1 демонстрирует поиск наибольшего общего делителя двух чисел методом вычитания*

**Алгоритмы факторизации**

Факторизация – это задача разбиения числа на множители. Известно, что любое число можно представить в следующем виде $A=a_1^{k1}a_2^{k2}\dots a_n^{kn}$, где $a_i$ – простые числа, $k_i$ – натуральные числа. Такое разложение существует всегда и называется оно **каноническим разложением числа**. Для построения канонического разложения достаточно научиться находить только один делитель. Получив делитель и поделив на него число $A$ (до тех пор, пока это возможно, поскольку число может поделиться на найденный простой делитель несколько раз), мы переходим к задаче поиска делителя для меньшего числа. Например, если найден первый простой делитель числа $A$, то $A=B\cdot a_1^{k1}$, где $a_1$ – простое число, а $k1$ – количество раз, на которое число $A$ поделилось. Например, число 60 делится на 2 (первое простое число) 2 раза, то есть $60=15\cdot 2^2$. Далее необходимо осуществлять поиск простого делителя числа 15. Итоговое каноническое разложение числа 60 имеет вид: $60=2^2\cdot 3\cdot 5$. Таким образом, для поиска канонического разложения числа необходимо осуществить поиск именно простых делителей. Для этого можно изначально рассматривать в качестве кандидатов на делитель только простые числа, которые могут быть первоначально найдены с помощью одного из алгоритмов: например, решето Эратосфена или решето Сундарама.

Поиск возможного делителя, если речь не идет о скорости – задача простая. Достаточно перебрать все простые числа от 2 до $\sqrt{A}$ и для каждого из них проверить делимость числа $A$.

*Пример 2 демонстрирует алгоритм нахождения наименьшего положительного целого делителя (большего единицы) для заданного числа, а также определение того, сколько раз число будет делиться на этот делитель без остатка*

**Алгоритм Ферма для разделения числа на 2 множителя**

Пусть $n = p \cdot q$ – известное целое число, являющееся произведением двух неизвестных простых чисел $p$ и $q$, которые требуется найти. Большинство современных методов факторизации основано на идее, предложенной еще Пьером Ферма, заключающейся в поиске пар натуральных чисел A и B таких, что выполняется соотношение:

$$n = A^2 - B^2. \tag{2.20}$$

Алгоритм Ферма может быть описан следующим образом:

1. Вычислим целую часть от квадратного корня из $n$:

$$m = \lceil \sqrt{n} \rceil.$$

2. Для $x = 1, 2, \ldots$ будем вычислять значения

$$q(x) = (m + x)^2 - n, \tag{2.21}$$

до тех пор, пока очередное значение $q(x)$ не окажется равным полному квадрату.

3. Пусть $q(x)$ является полным квадратом, например, числа $B$: $q(x) = B^2$. Определим $A = m + x$, откуда из равенства $A^2 - n = B^2$ найдем $n = A^2 - B^2 = (A + B) \cdot (A - B)$, и искомые делители $p$ и $q$ вычисляются, как $p = A + B$, $q = A - B$.

**Пример.** Пусть факторизуемое число $n = 19\,691$. Вычислим $m = \lfloor \sqrt{n} \rfloor = 140$. Представим процедуру вычисления делителей $n$ в виде таблицы:

| x | q | $\sqrt{q}$ |
|---|---|---|
| 1 | 190 | 13,78 |
| 2 | 473 | 21,75 |
| 3 | 758 | 27,53 |
| 4 | 1045 | 32,33 |
| 5 | 1334 | 36,52 |
| 6 | 1625 | 40,31 |
| 7 | 1918 | 43,79 |
| 8 | 2213 | 47,04 |
| 9 | 2510 | 50,10 |
| 10 | 2809 | 53 |

Из последнего столбца получим: $(140 + 10)^2 - n = 53^2$, откуда $n = 150^2 - 53^2 = 203 \cdot 97$. Итак, $19\,691 = 203 \cdot 97$, и вычисление потребовало 10 итераций, в каждой из которых было выполнено 1 возведение в степень, 1 вычитание и одно вычисление квадратного корня, т.е. константное число операций.

Естественно, алгоритм Ферма может привести к тому, что по крайней мере один из найденных множителей не будет простым числом. В этом случае необходимо для составного множителя применить снова алгоритм Ферма для разбиения его на множители.

Примечание: Алгоритм Ферма не применим к числам вида *4k+2* (где *k – натуральное число), т.е. числам 6, 10, 14, 18, 22, 26…*

*Пример 3 иллюстрирует применение алгоритма Ферма для разбиения числа на 2 множителя*

**Поиск простых чисел с помощью решета Эратосфена**

Простое число – это число, делящееся только на единицу и на себя. Определение дает и несложный метод поиска простых путем полного перебора вариантов. Необходимо записать цикл, проходящий все целые числа от 2 до некоторой границы, и для каждого числа пытаться найти нетривиальный (отличный от 1 и

самого числа) делитель. Если такой делитель найден, то число составное, иначе – простое. Боле эффективный метод – решето Эратосфена.

Алгоритм:

1. Выписать подряд все целые числа от двух до n (2, 3, 4, ..., n).
2. Пусть переменная p=2— первому простому числу.
3. Зачеркнуть в списке числа, кратные p (это будут числа: 2p, 3p, 4p, ...).
4. Найти первое не зачёркнутое число в списке, большее чем p, и присвоить переменной p это значение.
5. Повторять шаги 3 и 4, пока возможно.

Теперь все не зачёркнутые числа в списке — это простые числа от 2 до n.

Пример для n = 24.
a = 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24

1 Шаг. Находим все числа, кратные 2 и удаляем их:

a = 2 3 5 7 9 11 13 15 17 19 21 23

2 Шаг. Находим все числа, кратные 3 и удаляем их:

a = 2 3 5 7 11 13 17 19 23

3 Шаг. Находим все числа, кратные 5 и удаляем их:

Таких чисел нет, поэтому на этом шаге алгоритм останавливается.

Теперь a состоит только из простых чисел, которые меньше 24.

*Пример 4 иллюстрирует работу алгоритма «Решето Эратосфена».*

**Алгоритмы поиска простых чисел. Числа Мерсенна. Тест Люка-Лемера**

Число Мерсе́нна — число вида $\mathbf{2^n-1}$, где n — натуральное число; такие числа примечательны тем, что некоторые из них являются простыми при больших значениях n. Названы в честь французского математика Маре́на Мерсенна, исследовавшего их свойства в XVII веке.

1, 3, 7, 15, 31, 63, 127, 255, 511, 1023, 2047, 4095, 8191, 16 383, 32 767, 65 535, 131 071, …

Не все числа Мерсенна являются простыми, однако простых чисел среди всех чисел Мерсенна очень много. По этой причине числа Мерсенна удобно использовать для получения большого простого числа.

Для проверки числа Мерсенна на простоту существует простой тест **Люка-Лемера.** Это алгоритм, который предполагает на первом этапе построение последовательности по следующему правилу:

Пусть $p$ — простое нечётное. Число Мерсенна $M_p = 2^p - 1$ простое тогда и только тогда, когда оно делит нацело $(p-1)$-й член последовательности

$$4,\ 14,\ 194,\ 37634,\ \ldots$$ [2],

задаваемой рекуррентно: $S_k = \begin{cases} 4 & k = 1, \\ S_{k-1}^2 - 2 & k > 1. \end{cases}$

То есть последовательность всё время одна и та же.

Например, третье число Мерсенна $2^3$-1=7. Для проверки его на простоту с помощью теста Люка-Лемера необходимо построить последовательность (она приведена выше). Третье число Мерсенна (7) должно делить без остатка второй член последовательности (14). Действительно, 14%7=0. Математическим языком операция «%» записывается словом mod. То есть можно записать 14(mod 7)=0. Поскольку числа в последовательности очень быстро становятся большими, то при программной реализации каждый получаемый член последовательности заменяется на остаток от деления на рассматриваемое число Мерсенна. При этом для получения следующего члена последовательности используется именно остаток от деления с предыдущего шага.

Алгоритм Люка-Лемера (на псевдокоде) выглядит следующим образом:

```
►Вход: простое нечётное число p
S = 4
k = 1
M = 2^p - 1
До тех пока k != p - 1 выполнять
    S = ((S × S) - 2) mod M
    k += 1
Конец цикла
Если S = 0 выполнять
    Возвратить ПРОСТОЕ
иначе
    Возвратить СОСТАВНОЕ
Конец условия
```

Примеры применения теста Люка-Лемера к простому и составному числам Мерсенна:

Число $M_{13} = 2^{13} - 1 = 8191$ — простое[13]. Действительно,

$$S_1 = 4$$
$$S_2 \equiv 4^2 - 2 \pmod{8191} = 14$$
$$S_3 \equiv 14^2 - 2 \pmod{8191} = 194$$
$$S_4 \equiv 194^2 - 2 \pmod{8191} = 4870$$
$$S_5 \equiv 4870^2 - 2 \pmod{8191} = 3953$$
$$S_6 \equiv 3953^2 - 2 \pmod{8191} = 5970$$
$$S_7 \equiv 5970^2 - 2 \pmod{8191} = 1857$$
$$S_8 \equiv 1857^2 - 2 \pmod{8191} = 36$$
$$S_9 \equiv 36^2 - 2 \pmod{8191} = 1294$$
$$S_{10} \equiv 1294^2 - 2 \pmod{8191} = 3470$$
$$S_{11} \equiv 3470^2 - 2 \pmod{8191} = 128$$
$$S_{12} \equiv 128^2 - 2 \pmod{8191} = 0$$

Применение теста к числу $M_{11} = 2^{11} - 1 = 2047$

$$S_1 = 4$$
$$S_2 \equiv 4^2 - 2 \pmod{2047} = 14$$
$$S_3 \equiv 14^2 - 2 \pmod{2047} = 194$$
$$S_4 \equiv 194^2 - 2 \pmod{2047} = 788$$
$$S_5 \equiv 788^2 - 2 \pmod{2047} = 701$$
$$S_6 \equiv 701^2 - 2 \pmod{2047} = 119$$
$$S_7 \equiv 119^2 - 2 \pmod{2047} = 1877$$
$$S_8 \equiv 1877^2 - 2 \pmod{2047} = 240$$
$$S_9 \equiv 240^2 - 2 \pmod{2047} = 282$$
$$S_{10} \equiv 282^2 - 2 \pmod{2047} = 1736 \neq 0$$

Действительно, $2047 = 23 \cdot 89$.

*Пример 5 демонстрирует построение последовательности для теста Люка-Лемера с учетом нахождения остатков от деления на число Мерсенна*

**Псевдослучайные числа. Числа фон Неймана**

В различных технических и математических приложениях часто необходим ряд случайных чисел. Числа называются случайными в силу того, что в их ряду нет никакой закономерности, зная некоторое количество последовательных чисел ряда, нельзя вычислить следующее. Устройство или программа, получающая такие числа, называется генератором случайных чисел. О принципиальной возможности существования случайных последовательностей можно спорить. Нетрудно встретить мнение, что все процессы, существующие в природе или созданные человеком, строго закономерны. Но, во всяком случае, если это и так, то некоторые из закономерностей настолько сложны и управляются настолько большим количеством параметров, что фактически их невозможно отследить за разумное время с использованием разумного количества ресурсов.

**Алгоритм фон Неймана** для генерации псевдослучайных чисел состоит в следующем. Предположим, что некоторое случайное, достаточно большое число уже известно. Определить такое число совершенно не представляет проблему: так как оно первое, то оно может быть просто любым. Пусть, например, нас интересуют 5-значные случайные числа. Возьмем в качестве первого число 14 563. Возведем его в квадрат, получим 212 080 969. Возьмем из середины пять цифр 20 809. Это и будет следующее случайное число. Псевдослучайность получаемой последовательности очевидна. Каждое следующее число совершенно однозначно определяется предыдущим. Но нас это смущать не должно. Если неизвестно, как получены числа, то они выглядят вполне случайно. Но у метода есть более серьезный недостаток. Если последовательность продолжить, то можетпроявиться так называемый период. Периодом называется строго повторяющаяся последовательность чисел. Впрочем, для алгоритма фон Неймана можно подобрать исходное число, дающее период только после очень большого количества шагов.

*Пример 6 иллюстрирует получение второй, третьей и четвертой цифр пятизначного числа*

**Пример 1**

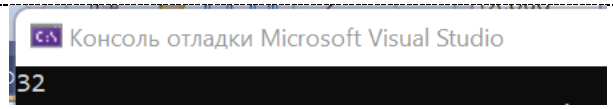
***Задача:***

Написать на языке C++ программу для поиска наибольшего общего делителя двух чисел методом вычитания

***Решение:***

```cpp
#include <iostream>

int nod_min(int first_number, int second_number)
{
    int raznost = 0;
    bool flag = false; //Индикатор окончания цикла
    while (flag == false)
    {
        if (first_number == second_number)
        {
            flag = true;
        }
        else if (first_number < second_number)
        {
            raznost = second_number - first_number;
            second_number = first_number;
            first_number = raznost;
        }
        else //first_number > second_number
        {
            raznost = first_number - second_number;
            first_number = second_number;
            second_number = raznost;
        }
    }

    return first_number;
}

int main()
{
    std::cout << nod_min(128, 96);
}
```

***Ответ:***




**Пример 2**

***Задача:***

*Написать на языке C++ программу для нахождения наименьшего положительного целого делителя (большего единицы) для заданного числа A, а также определения того, сколько раз число A будет делиться на этот делитель без остатка (кратность остатка)*

***Решение:***

```cpp
#include <iostream>

using namespace std;

int main()
{
    setlocale(LC_ALL, "Russian");
    unsigned int A; //Заданное число
    int delitel = 0;
    int count = 0; //сколько раз найденный делитель
                   //будет делить без остатка (кратность)
    cout << "Введите целое положительное число: ";
    cin >> A;
    for (int i = 2; i <= sqrt(A); i++)
    {
        if (A % i == 0)
        {
            delitel = i;
            while (A % i == 0)
            {
                A = A / i;
                count++;
            }
            break;
        }
    }
    cout << "Наименьший делитель: " << delitel<<endl;
    cout << "Кратность делителя: " << count << endl;
}
```

***Ответ:***

```
Консоль отладки Microsoft Visual Studio
Введите целое положительное число: 60
Наименьший делитель: 2
Кратность делителя: 2
```

**Пример 3**

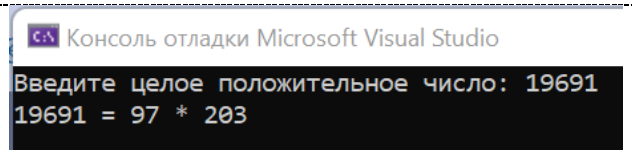
***Задача:***

*Написать на языке C++ программу для* разбиения заданного числа на 2 множителя с помощью алгоритма Ферма

***Решение:***

```cpp
#include <iostream>

using namespace std;

int main()
{
    setlocale(LC_ALL, "Russian");
    unsigned int n; //Заданное число
    int delitel1 = 0;
    int delitel2 = 0;
    cout << "Введите целое положительное число: ";
    cin >> n;

    int m = sqrt(n);
    int x = 1;
    int q = 0;
    int koren_iz_q = 0;
    while(true)
    {
        q = (m + x) * (m + x) - n;
        koren_iz_q = sqrt(q);
        if (koren_iz_q* koren_iz_q == q)
        {
            break;
        }
        else
        {
            x++;
        }
    }
    delitel1 = (m + x) - koren_iz_q;
    delitel2 = (m + x) + koren_iz_q;

    cout << n << " = " << delitel1 << " * " << delitel2 << endl;
}
```

***Ответ:***




**Пример 4**

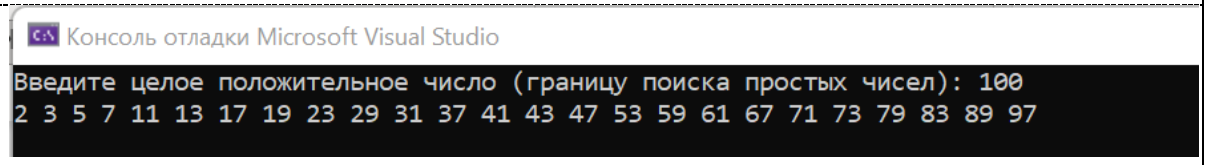
***Задача:***

*Написать на языке C++ программу для* поиска простых чисел, не превосходящих заданную границу, с помощью алгоритма «Решето Эратосфена»

***Решение:***

```cpp
#include <iostream>

using namespace std;

int main()
{
    setlocale(LC_ALL, "Russian");
    unsigned int n = 0; //Заданное число
    int delitel1 = 0;
    int delitel2 = 0;
    cout << "Введите целое положительное число (границу поиска простых чисел): ";
    cin >> n;

    int *numbers = new int[n - 1]; //массив для чисел от 2 до n
    for (int i = 2; i <= n; i++)
    {
        numbers[i - 2] = i;
    }

    int current_index = 0;
    int current_num = 0;

    while (current_index < n)
    {
        if (numbers[current_index] != 0)
        {
            current_num = numbers[current_index];
            for (int i = current_index + 1; i < n; i++)
            {
                if (numbers[i] % current_num == 0)
                {
                    numbers[i] = 0;
                }
            }
        }
        current_index++;
    }

    for (int i = 0; i < n; i++)
    {
        if (numbers[i] != 0)
        {
            cout << numbers[i] << " ";
        }
    }
    cout << endl;

    delete[] numbers;
}
```

***Ответ:***




**Пример 5**

***Задача:***

*Написать на языке С++ программу для построение последовательности для теста Люка-Лемера с учетом нахождения остатков от деления на заданное число Мерсенна*

***Решение:***

```cpp
#include <iostream>

using namespace std;

int main()
{
    int M = 8191; //Число Мерсенна номер 13

    int S = 4; //Первый член последовательности
    int count = 12; //Количество членов последовательности
    for (int i = 0; i < count; i++)
    {
        cout << S << " ";
        S = (S * S - 2) % M;
    }
    cout << endl;
}
```

***Ответ:***

```
Консоль отладки Microsoft Visual Studio
4 14 194 4870 3953 5970 1857 36 1294 3470 128 0
```

**Пример 6**

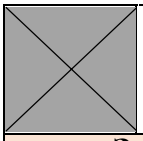
***Задача:***

*Написать на языке C++ программу для получение второй, третьей и четвертой цифр пятизначного числа*

***Решение:***

```cpp
#include <iostream>

using namespace std;

int main()
{
    int A = 14764;
    int dig2 = 0;
    int dig3 = 0;
    int dig4 = 0;

    int d = A % 10; // Получаем последнюю цифру
    A /= 10; // Избавляемся от неё

    dig4 = A % 10; // Получаем четвертую цифру
    A /= 10; // Избавляемся от неё

    dig3 = A % 10; // Получаем третью цифру
    A /= 10; // Избавляемся от неё

    dig2 = A % 10; // Получаем вторую цифру

    cout << dig2 << dig3 << dig4 << endl;
}
```

***Ответ:***

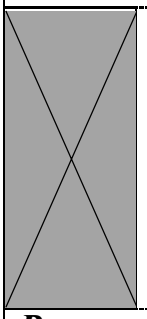


Консоль отладки Microsoft Visual Studio

```
476
```

**Задание 1**

***Задача:***



Написать программу для поиска наибольшего общего делителя (числа вводятся с клавиатуры после запуска программы):

**Нечетные варианты**: наибольший делитель трех чисел методом деления (для поиска остатка отделения в языке C++ используется операция %)

**Четные варианты**: наибольший общий делитель четырех чисел метолом вычитания

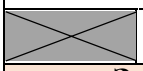
***Решение:***

```python
from sys import stdin


def nod_del_thee(*args):
    args = list(args)
    while 0 in args:
        args[args.index(0)] = max(args)
    args = list(map(lambda x: abs(x), args))
    nums = sorted(args)
    vrem_nod = nums[-1]
    for i in range(len(nums)-1, -1, -1):
        ost1 = nod_del_two(max(nums[i], vrem_nod), min(nums[i], vrem_nod))
        vrem_nod = ost1
    return vrem_nod


def nod_del_two(num1, num2):
    ost = num2
    while num1 % num2 != 0:
        ost = num1 % num2
        num1 = num2
        num2 = ost
    return ost


def nod_subt_four(*args):
    args = list(args)
    while 0 in args:
        args[args.index(0)] = max(args)
    args = list(map(lambda x: abs(x), args))
    nums = sorted(args)

    vrem_nod = nums[-1]
    for i in range(len(nums)-1, -1, -1):
        ost1 = nod_subt_two(max(nums[i], vrem_nod), min(nums[i], vrem_nod))
        vrem_nod = ost1
    return vrem_nod
```

```
def nod_subt_two(num1,num2):
    ost = num2
    while num1 != num2:
        ost = num1 - num2
        num1 = max(num2, ost)
        num2 = min(num2, ost)
    return num1


lines = []
for line in stdin:
    lines.append(int(line))

if lines == [0, 0, 0, 0] or lines == [0, 0, 0]: print("NOD не существует")
else:
    if len(lines) == 3:
        print("NOD методом деления: " + str(nod_del_thee(*lines)))
    elif len(lines) == 4:
        print("NOD методом вычетания: " + str(nod_subt_four(*lines)))
```

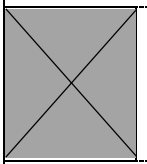
```
PS D:\MireaFulstekProblemSolving\Алгоритмы и стр
lving/Алгоритмы и структуры данных/РТ 2/алг_рт2_
0
0
15
25
^Z
NOD методом вычетания: 5
```

```
PS D:\MireaFulstekProblemSolving\Алгоритмы и структуры д
lving/Алгоритмы и структуры данных/РТ 2/алг_рт2_з1.py"
-3
-15
3
^Z
NOD методом деления: 3
```



**Задание 2**

***Задача:***



Написать программу для факторизации заданного с клавиатуры числа методом простого перебора (указать простые множители и их кратность). Для анализа числа на простоту использовать решето Эратосфена

***Решение:***

```python
def eretosfen(n):
    arr = [i for i in range(2, n+1)]
    p = 2
    while arr != list(filter(lambda x: x % p != 0, arr)):
        i = 1
        while i < len(arr):
            if arr[i] != p and arr[i] % p == 0:
                arr.pop(i)
            i += 1
        if arr.index(p) < len(arr) - 1:
            p = arr[arr.index(p) + 1]
        else: break
    return arr


def razlozh(a):
    itog = []
    b = 18
    if a in eretosfen(a):
        return f"{a} ^ 1"
    else:
        for i in range(2, int(a ** 0.5) + 1):
            if a % i == 0 and (i in eretosfen(i)):
                c = 0
                b = a
                while b % i == 0:
                    c += 1
                    b = b // i
                itog.append(f"{i} ^ {c}")

                i = a // i
                if a % i == 0 and (i in eretosfen(i)):
                    c = 0
                    b = a
                    while b % i == 0:
                        c += 1
                        b = b // i
                    itog.append(f"{i} ^ {c}")

    return " * ".join(itog)


print(razlozh(17))
print(razlozh(18))
```

***Ответ:***

```
Ivlng/Алгоритмы и структуры данных/РТ 2/алг_рт2_32.py
17 ^ 1
2 ^ 1 * 3 ^ 2
```

**Задание 3**

***Задача:***

Написать программу для факторизации заданного с клавиатуры нечетного числа методом Ферма (указать простые множители и их кратность). Для анализа числа на простоту использовать решето Эратосфена

***Решение:***

```
def eretosfen(n):
    arr = [i for i in range(2, n+1)]
    p = 2
    while arr != list(filter(lambda x: x % p != 0, arr)):
        i = 1
        while i < len(arr):
            if arr[i] != p and arr[i] % p == 0:
                arr.pop(i)
            i += 1
        if arr.index(p) < len(arr) - 1:
            p = arr[arr.index(p) + 1]
        else: break
    return arr


def ferma(n):
    n = n
    m = int(n**0.5)
    x = 1
    q = (m + x)**2 - n
    if n ** 0.5 == int(n ** 0.5) and int(n ** 0.5) in eretosfen(int(n ** 0.5)):
        return [int(n ** 0.5), int(n ** 0.5)]

    while q ** 0.5 != int(q ** 0.5):
        x += 1
        q = (m + x)**2 - n
    p1 = int(q**0.5 + (x + m))
    p2 = int((m + x) - q**0.5)
    itog = [p1,p2]
    if p1 not in eretosfen(p1) and p1 != 1:
        for i in list(ferma(p1)):
            itog.append(i)
        itog.remove(p1)
    if p2 not in list(eretosfen(p2))and p2 != 1:
        for i in ferma(p2):
            itog.append(i)
        itog.remove(p2)
    return itog


def format_ferma(n):
    if n % 2 == 0:
        return "нет"
```

```
    itog = ferma(n)

    while 1 in itog:
        itog.remove(1)

    set_itog = sorted(set(itog))
    res = []
    for i in set_itog:
        res.append(f"{i} ** {itog.count(i)}")
    return " * ".join(res)


print(format_ferma(1275))
print(format_ferma(405))
print(format_ferma(17))
print(format_ferma(10))
```

***Ответ:***

```
3 ** 1 * 5 ** 2 * 17 ** 1
3 ** 4 * 5 ** 1
17 ** 1
нет
```

**Задание 4**

***Задача:***

Написать программу для проверки на простоту числа Мерсенна с использование теста Люка-Лемера. С клавиатуры вводится номер числа Мерсенна

***Решение:***

```
from sys import stdin


def marsen(p):
    return  (2 ** p) - 1


def luka_lemar(p):
    k = p - 1
    s = 4
    for _ in range(k-1):
        s = s**2 - 2
    if s % marsen(p) == 0:
        return "Простое", marsen(p)
    else: return "Не простое", marsen(p)
```

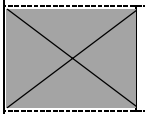
```
for num in stdin:
    print(luka_lemar(int(num)))
```

***Ответ:***

```
IVing/Алгоритмы и структуры данных/РТ 2/алг_рт2_34.py
3
('Простое', 7)
4
('Не простое', 15)
5
('Простое', 31)
6
('Не простое', 63)
```

**Задание 5**

***Задача:***



Написать программу для генерации последовательности из 10 пятизначных чисел фон Неймана

***Решение:***

```
def neiman(n):
    return str(int(n)**2)[2:7]


n = input()
for i in range(10):
    n = neiman(n)
    print(n)
```

***Ответ:***

```
IVing/Алгоритмы и структуры данных/РТ 2/алг_рт2_35.py
12345
23990
55201
47150
23122
46268
40727
58688
44281
60806
97369
```

**Задание 6***

***Задача:***

Написать программу для поиска простых чисел по алгоритму «**Решето Сундарама**»

Решето Сундарама — второй по известности алгоритм поиска нечетных простых чисел, придуманный индийским студентом Сундарамом в 1934 году. Его суть состоит в вычеркивании из списка чисел от 1 до $(N-1)/2$ всех чисел вида

$$i + j + 2ij$$
$$1 <= i <= j$$

После этого, для оставшихся в списке чисел $n$, последовательность чисел $2n+1$ будет представлять все нечетные простые числа до N.

```
вычеркиваем 1+j+2*1*j: [1, 2, 3,~~4~~, 5, 6,~~7~~, 8, 9,~~10~~, ...]
вычеркиваем 2+j+2*2*j: [1, 2, 3, 5, 6, 8, 9, 11,~~12~~, 14, 15, ...]
вычеркиваем 3+j+2*3*j: [1, 2, 3, 5, 6, 8, 9, 10, 11, 14, ...,~~24~~, ...]
и так далее...
последовательность 2n+1: [3, 5, 7, 11, 13, 17, 19, ...]
```

***Решение:***

```python
def sundaram(n):
    new_n = (n-1)//2
    arr = [i for i in range(1, (n-1)//2+1)]
    for i in range(1, new_n+1):
        for j in range(1, new_n+1):
            if i + j + 2 * i * j > len(arr):
                break
            elif i + j + 2 * i * j in arr:
                arr[(i + j + 2 * i * j)-1] = 0
    arr = list(filter(lambda x: x != 0, arr))
    arr = list(map(lambda x: x*2 + 1, arr))
    return arr


print(sundaram(int(input())))
```

***Ответ:***

```
123
[3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97, 101, 103, 107, 109, 113]
```