**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

отчет

**по лабораторной работе №1**

**по дисциплине «Построение и анализ алгоритмов»**

**Тема: «Поиск с возвратом»**

Студент гр. 6381 \_\_\_\_\_\_\_\_\_\_\_\_\_ Федянин Н.И.

Преподаватель \_\_\_\_\_\_\_\_\_\_\_\_\_ Филатов А.Ю.

Санкт-Петербург

2018

**Задание.**

У Вовы много квадратных обрезков доски. Их стороны (размер) изменяются от 1 до N−1, и у него есть неограниченное число обрезков любого размера. Но ему очень хочется получить большую столешницу - квадрат размера N. Он может получить ее, собрав из уже имеющихся обрезков(квадратов).

Например, столешница размера 7×7 может быть построена из 9 обрезков.



Внутри столешницы не должно быть пустот, обрезки не должны выходить за пределы столешницы и не должны перекрываться. Кроме того, Вова хочет использовать минимально возможное число обрезков.

**Входные данные:**

Размер столешницы - одно целое число N(2≤N≤40).

**Выходные данные:**

Одно число K, задающее минимальное количество обрезков (квадратов), из которых можно построить столешницу(квадрат) заданного размера N. Далее должны идти K строк, каждая из которых должна содержать три целых числа x,y и w, задающие координаты левого верхнего угла (1≤x,y≤N) и длину стороны соответствующего обрезка(квадрата).

**Пояснение задания.**

Необходимо, чтобы программа считывала длину стороны исходного квадрата, а затем c помощью перебора с возвратом производила его разложение на минимальное количество квадратов.

**Описание алгоритма.**

На вход подается длина стороны квадрата. Далее число проверяется на простоту и делимость на 2, 3 и 5.

Если число четное, кратное 3 или 5, применяется соответствующий алгоритм (если число четное, то квадрат делится на 4 равных квадрата, если число кратно 3 – то квадрат делится на 6 квадратов (Рисунок 1), если число кратно 5 – то квадрат делится на 8 (Рисунок 2)).

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

Рисунок 1(для N = 9)

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

Рисунок 2(для N = 10)

Убедимся, что для квадратов с длиной стороны кратной 2, 3 и 5, можно применить упрощение в целях ускорения времени работы программы. Для этого разработаем программу, выполняющую полный перебор и проверим, совпадают ли её решения с решениями программы, содержащей упрощения.

При тестировании программы c упрощением с числами 4, 5 и 9 получаем:

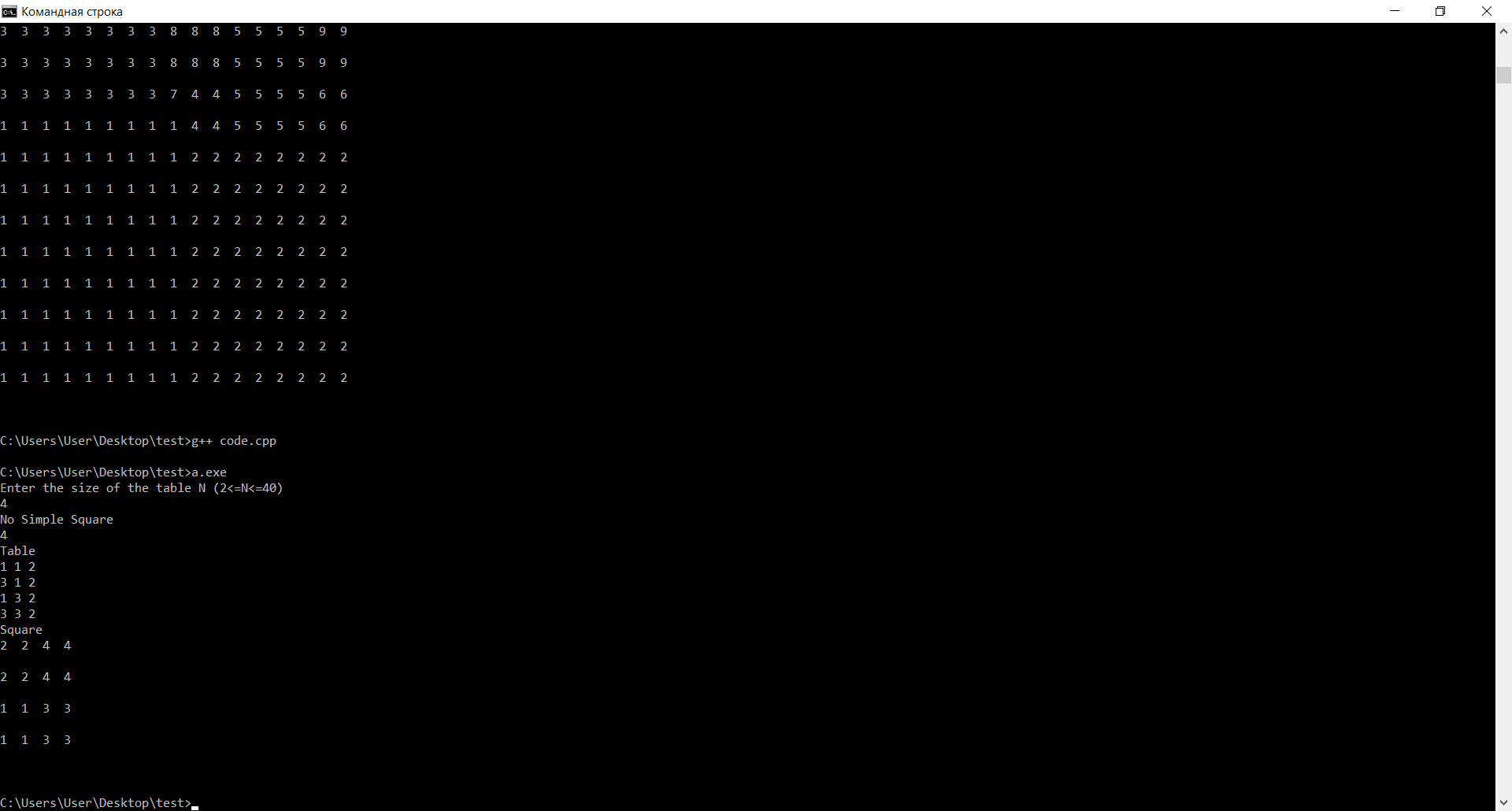


Рисунок 3(для N = 4)

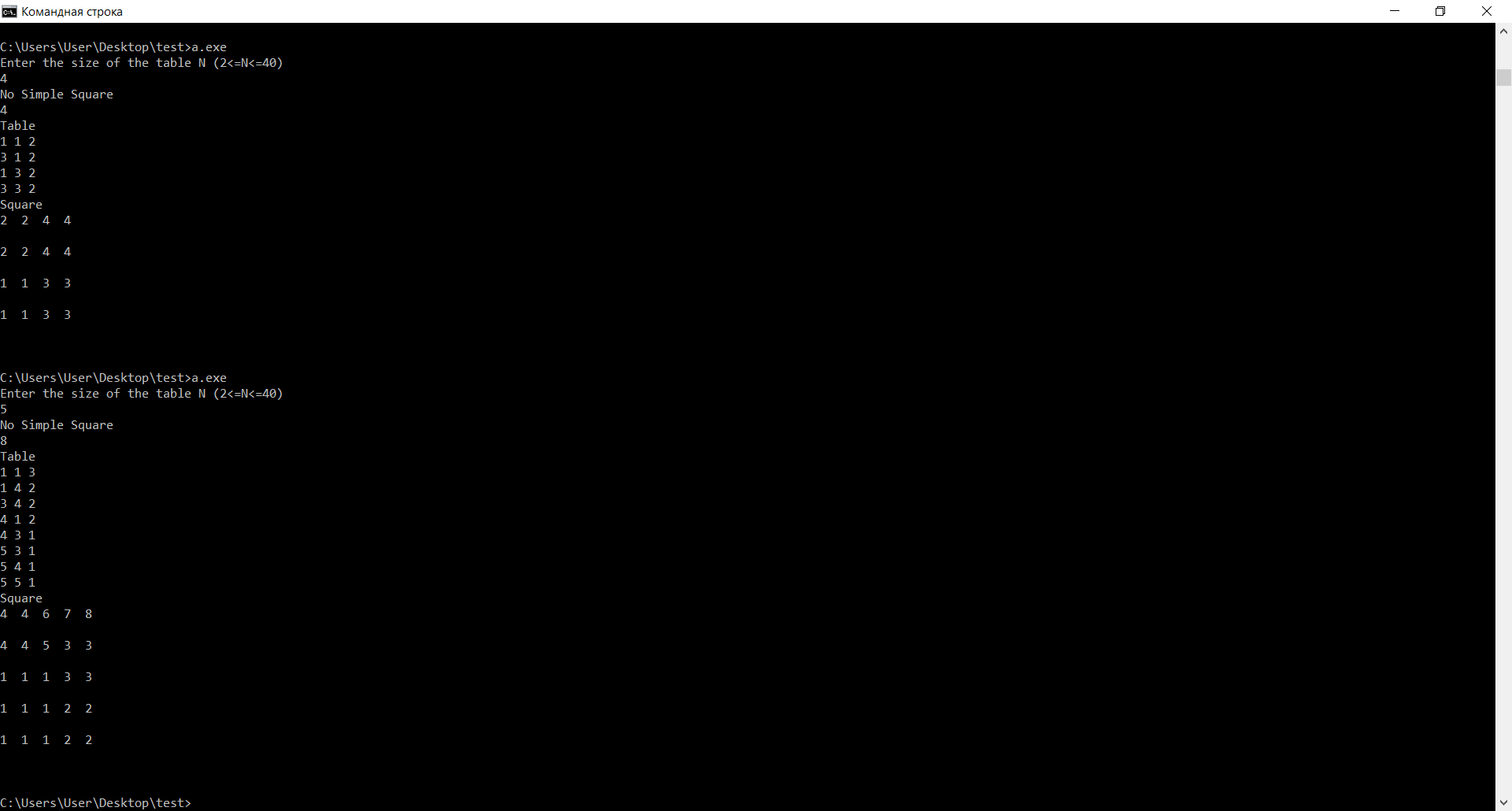


Рисунок 4(для N = 5)

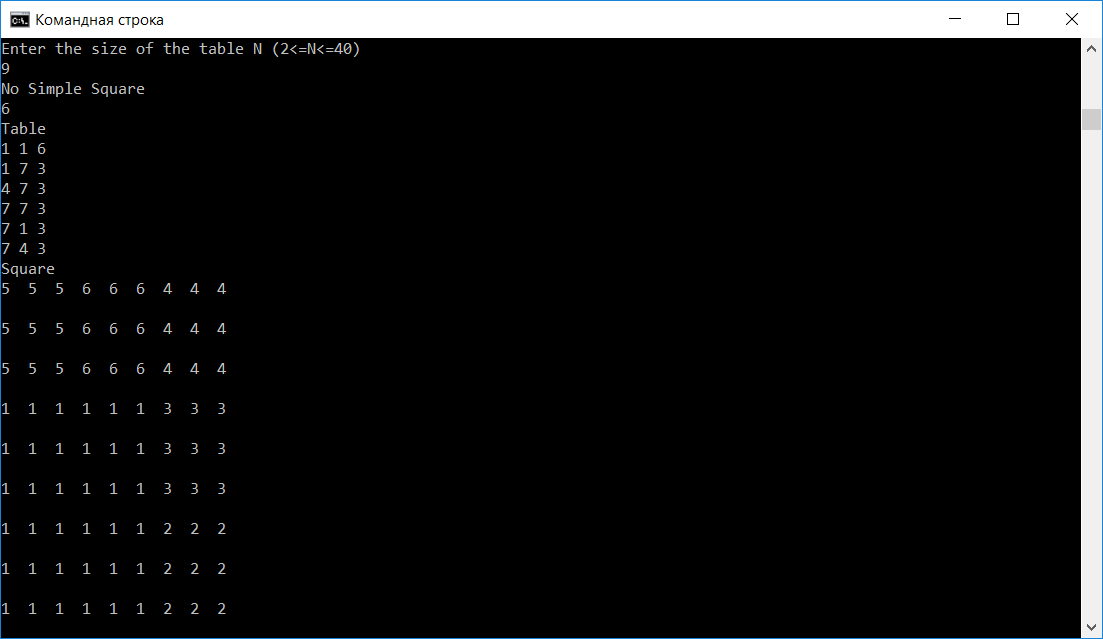


Рисунок 5(для N = 9)

При тестировании программы полным перебором:

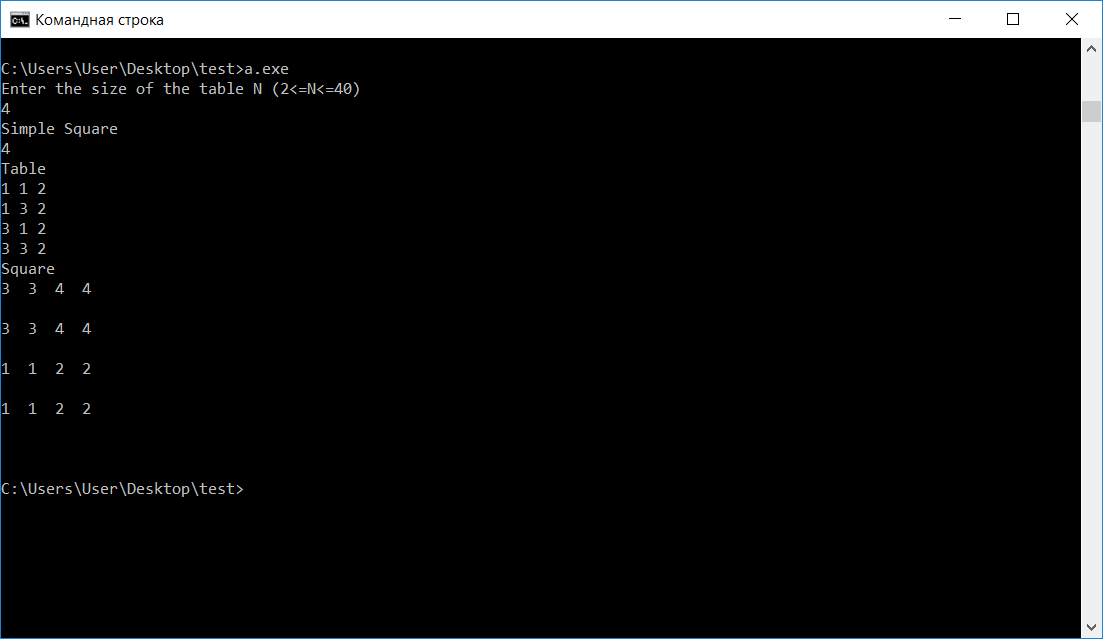


Рисунок 6(для N = 4)

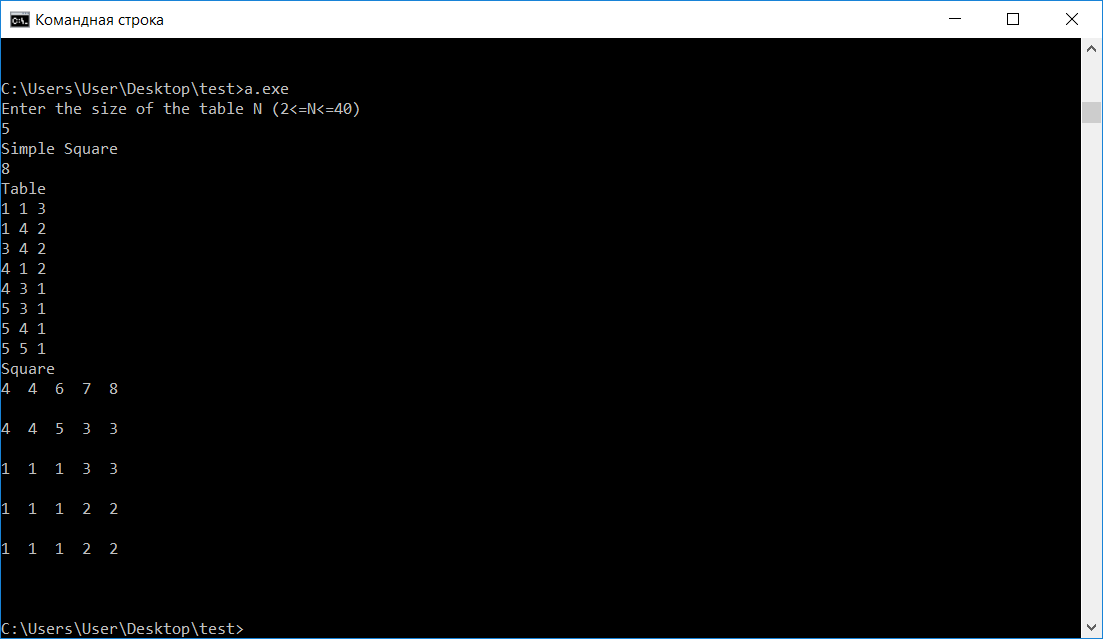


Рисунок 7(для N = 5)

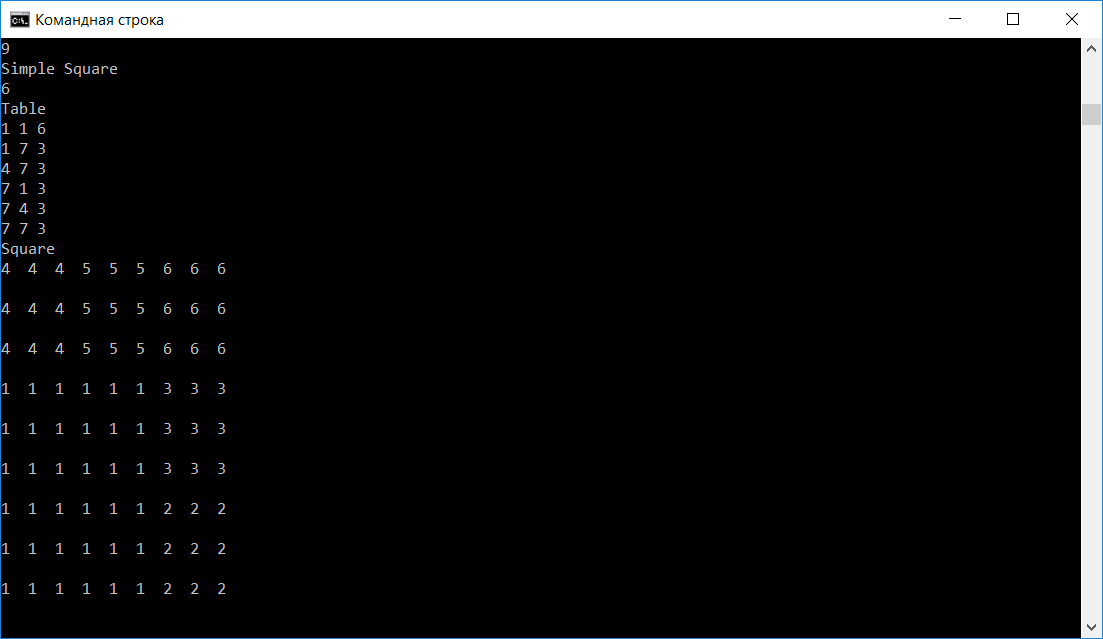


Рисунок 8(для N = 9)

Как можно заметить разложения получились идентичными за исключением положения квадратов, но они не меняют решения.

Если же число простое, происходит заполнение квадратом со стороной (N+1)/2 и двумя квадратами со стороной N/2 в левый верхний, правый верхний и левый нижний углы (Рисунок 3).

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

Рисунок 9(для N = 7)

Для некоторых простых чисел, такое как 11, например, решения программ могут не совпасть, однако содержать они будут одинаковое количество квадратов. Связано это с тем, что полный перебор выбирает один из вариантов решения с наименьшим числом квадратом, а этих вариантов может быть несколько. Если в программе с полным перебором указать вставку квадратов до размера (n+1)/2, то решения совпадут.

Протестируем программы для n = 11 и изобразим решения:

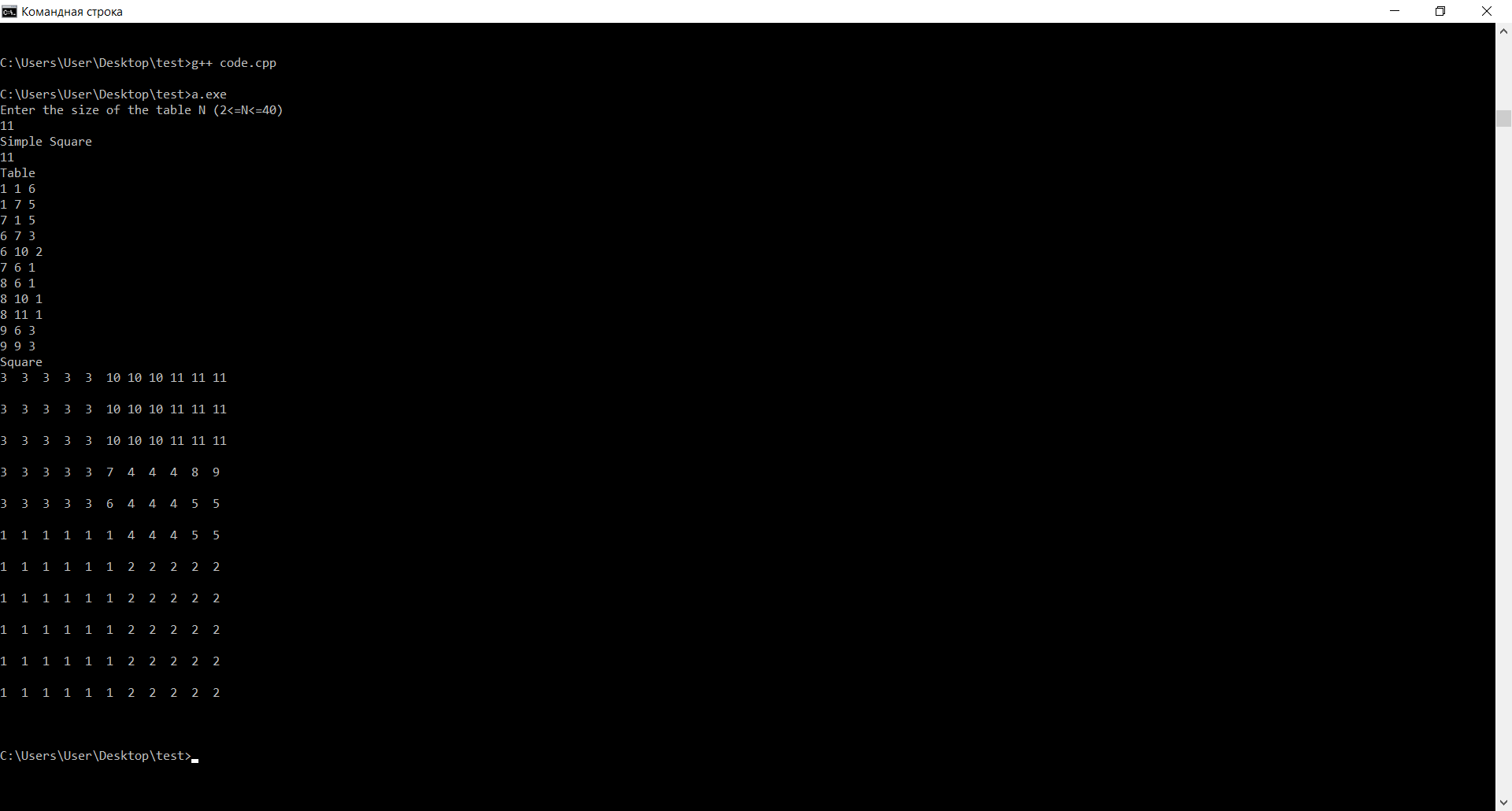


Рисунок 10. Работа программы с упрощенным перебором.

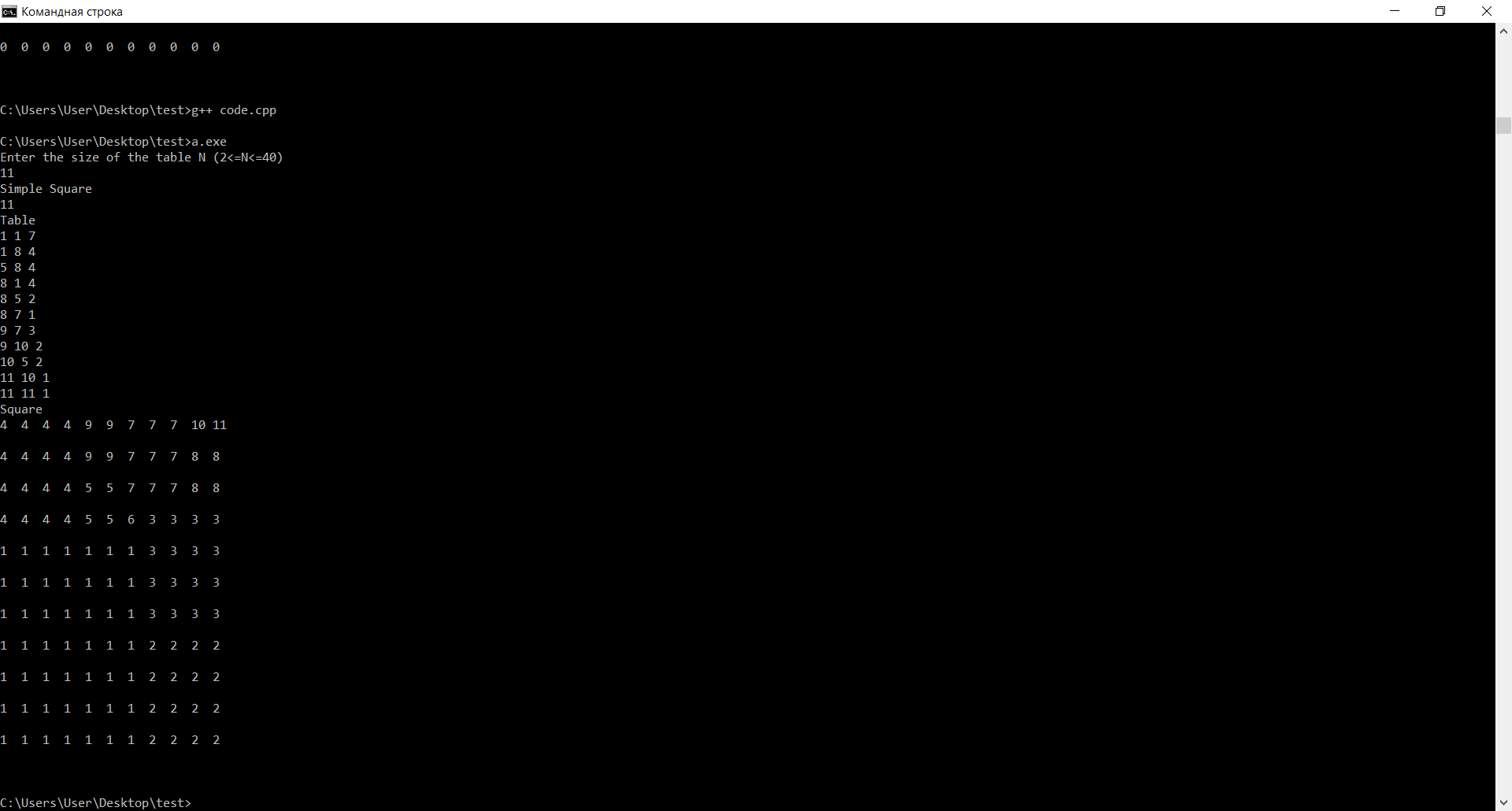


Рисунок 11. Работа программы с полным перебором.

Приходим к выводу о том, что используемые алгоритмы, а также заполнение начальных трех квадратов в простых числах имеют место быть в программе для уменьшения времени её исполнения.

Затем осуществляется вход в рекурсивную функцию для оставшейся незаполненной части.

В рекурсивной функции начинается заполнение пустого места квадратами, начиная с квадратов длиной (N-1) х (N-1) и уменьшая его длину, если квадрат выходит за пределы исходного квадрата со стороной N. Наименьшее число полученных квадратов хранится в отдельной переменной, и как только во время перебора текущее число квадратов становится больше или равно наименьшему, либо заканчиваются пустые ячейки и число квадратов оказывается больше полученного на тот момент наименьшего, решение удаляется и делается шаг назад в поисках более оптимального решения. Если же по окончании пустых ячеек квадратов получилось меньше, чем наименьшее их число, полученное на тот момент, программа перезаписывает лучшее решение. В итоге программа получит разделение на наименьшее количество квадратов.

**Описание функций.**

1. **bool Primes(int n) -** проверка числа на простоту и делимость на 2, 3, 5**;**
2. **void NoSimpleSquares(vector<vector<int>>& TempTable, int size) -**функция для квадратов с не простой стороной, делящейся на 2, 3, 5;
3. **void SimpleSquares(vector<vector<int>>& TempTable, int size, vector<vector<int>>& BestTable) -** функция для квадратов с простой стороной, не делящейся на 2, 3, 5;
4. **void Print\_x\_y\_size\_of\_a\_square(int x, int y, int size) -** вывестии х, у, size квадрата;
5. **bool ArrangementSquares(vector<vector<int>>& Table, int N, int x, int y, int scale, int number) –** расстановка квадратов;
6. **bool go\_beyond\_the\_table(int x, int y, int scale, int N) –** проверка на выход за пределы массива;
7. **void RemoveSquare(vector<vector<int>>& Table, int N, int x, int y, int number)** – удаление квадратов;
8. **bool FindFreeCell(vector<vector<int>>& Table, int N, int& x, int& y)** – поиск свободных клеток;
9. **int Alignment(vector<vector<int>>& Table, int N, int x, int y, vector<vector<int>>& bestTable) –** подсчёт минимального числа квадратов;
10. **int FindSquare(vector<vector<int>>& Table, int N, int& x, int& y, int K) –** возвращает размер квадрата;
11. **void Print\_x\_y\_size\_of\_a\_table(vector<vector<int>>& Table, int N, int partNmbr) –** вывести таблицу с координатами всех квадратов;
12. **void PrintSquare(vector<vector<int>>& TempTable) -** вывести исходный квадрат размера N;

**Тестирование.**

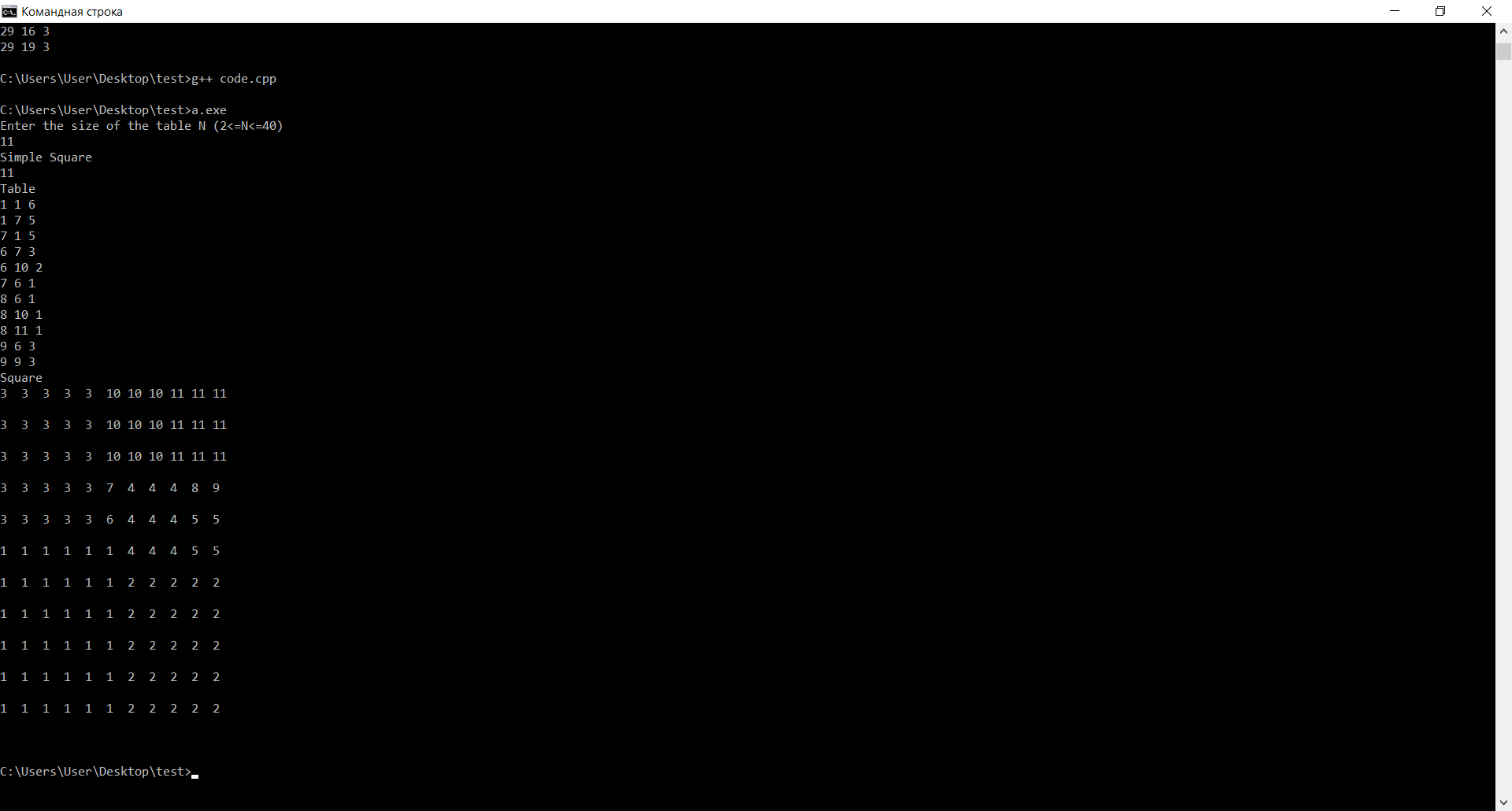
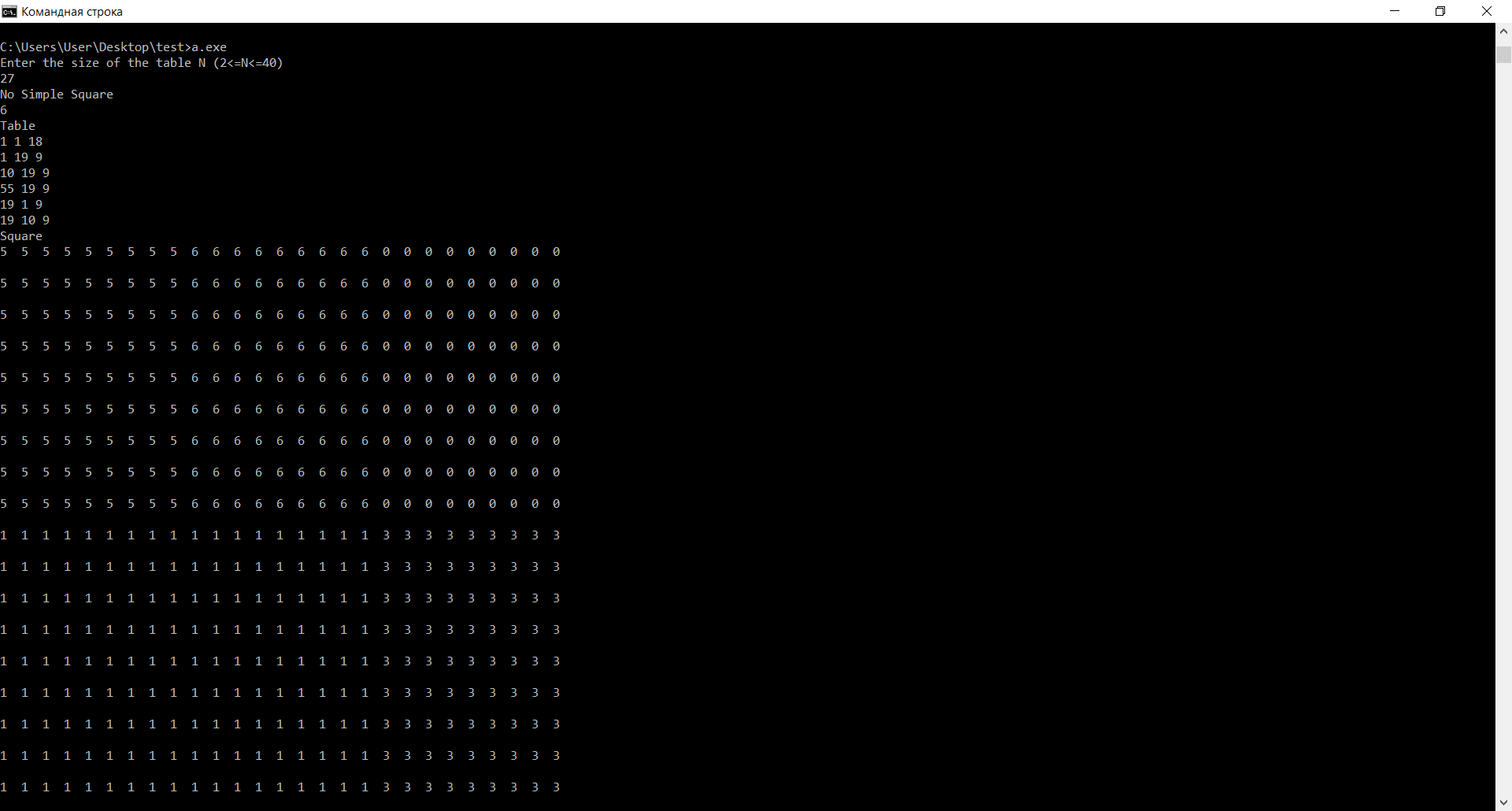


Рисунок 12(для N = 11)



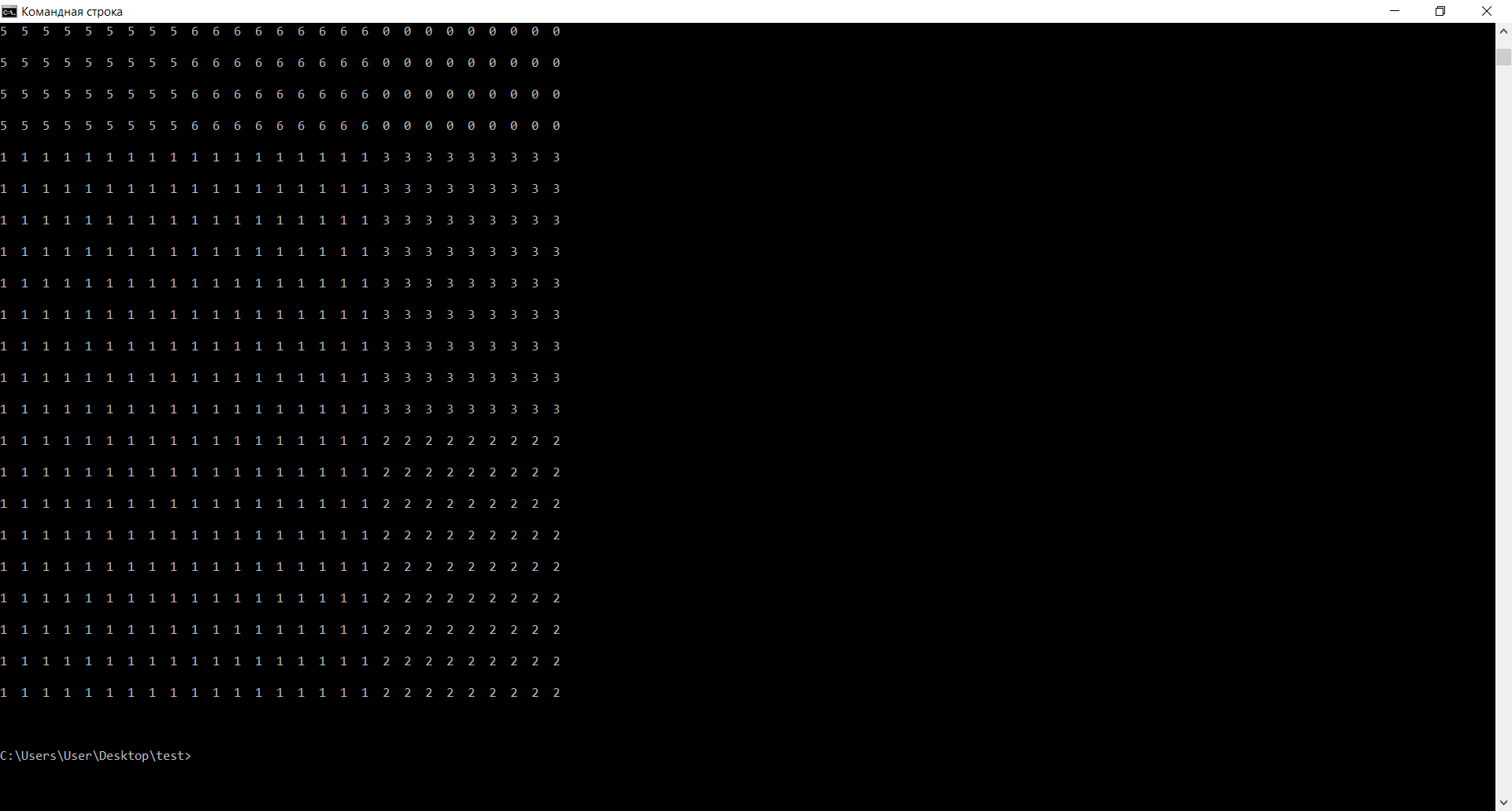


Рисунок 13(для N = 27)

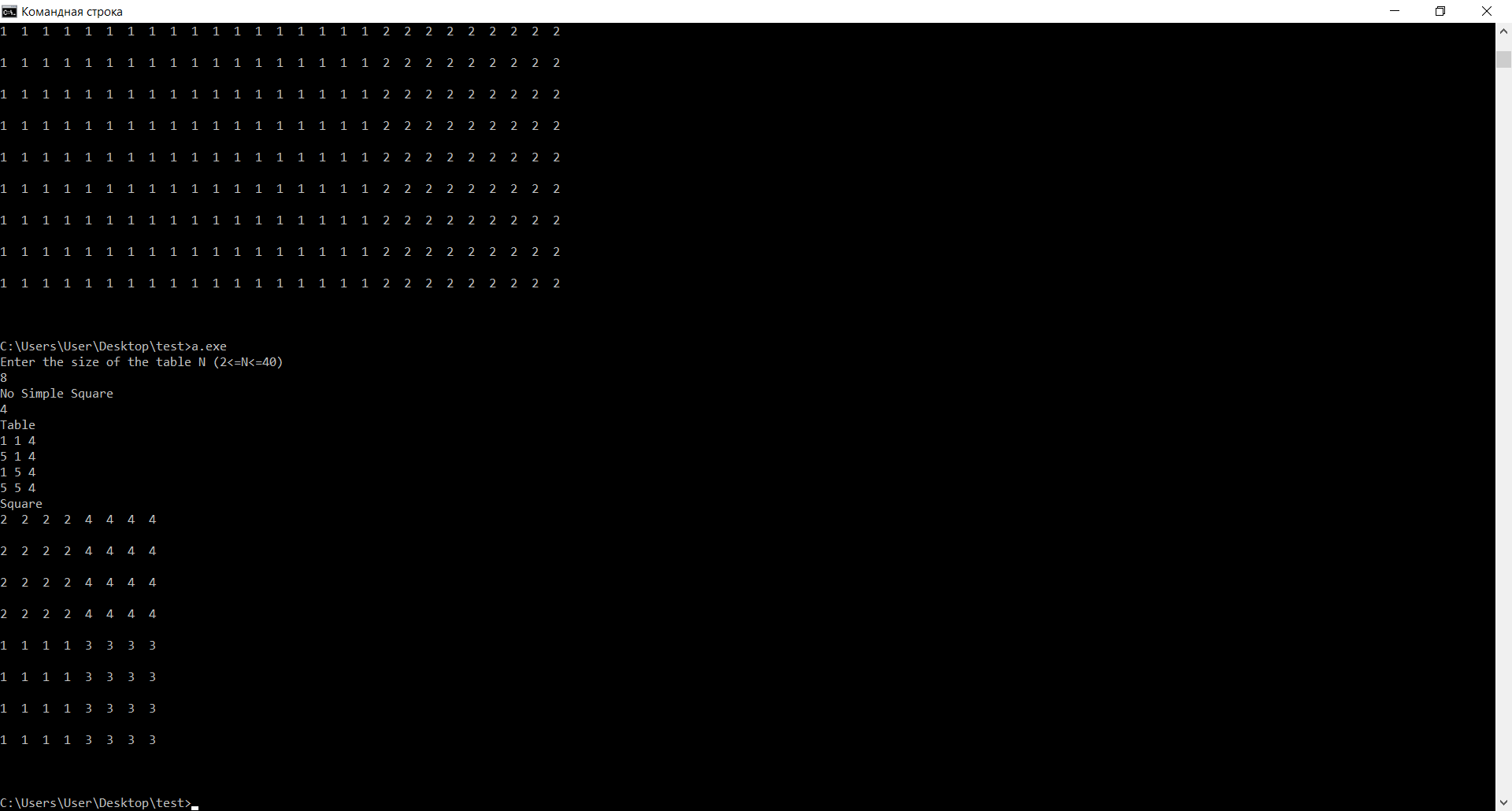


Рисунок 14(для N = 8)

**Выводы.**

В ходе лабораторной работы мы подробно ознакомились с принципом поиска с возвратом. В результате выполнения лабораторной работы была разработана программа, которая вычисляет наименьшее количество квадратов, на которые можно разделить квадрат длиной от 2 до 40.