|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | |  |  |  | | МИНОБРНАУКИ РОССИИ | | | | Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** | | |   Институт Информационных технологий | |
|  | |
| Кафедра Математического обеспечения и стандартизации информационных технологий | |
|  | |
|  | |

|  |  |
| --- | --- |
| **ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ № 8** | |
| **по дисциплине** | |
| **«**Структуры и алгоритмы обработки данных**»**  **Тема: «Алгоритмические стратегии или методы разработки алгоритмов. Перебор и методы его сокращения.»** | |
|  | |
| Выполнил студент группы ИКБО-04-21 | Исаев В.В. |
| Принял преподаватель | Филатов А.С. |

|  |  |  |
| --- | --- | --- |
| Лабораторная работа выполнена | «\_\_»\_\_\_\_\_\_\_202\_\_ г. | *(подпись студента)* |
|  |  |  |
| «Зачтено» | «\_\_»\_\_\_\_\_\_\_202\_\_ г. | *(подпись руководителя)* |

Москва 2022

# Цель работы

Получить навыки применения методов, позволяющих сократить число переборов в задачах, которые могут быть решены только методом перебора всех возможных вариантов решения.

# Постановка задачи

1. Разработать алгоритм решения задачи с применением метода, указанного в варианте и реализовать программу.
   1. Оценить количество переборов при решении задачи стратегией «в лоб» - грубой силы.
   2. Привести анализ снижения числа переборов при применении метода.
2. Составить отчет, отобразив в нем описание выполнения всех этапов разработки, тестирования и код всей программы со скриншотами результатов тестирования.

Таблица 1. Варианты заданий

|  |  |  |
| --- | --- | --- |
| №  Варианта | Задача | Метод |
| 8 | Черепашке нужно попасть из пункта А в пункт В. Поле движения разбито на квадраты. Известно время движения вверх и вправо в каждой клетке (улицы). На каждом углу она может поворачивать только на север или только на восток. Найти минимальное время, за которое черепашка может попасть из А в В. | Динамическое программирование |

# Решение

Динамическое программирование в теории управления и теории вычислительных систем — способ решения сложных задач путём разбиения их на более простые подзадачи. Он применим к задачам с оптимальной подструктурой, выглядящим как набор перекрывающихся подзадач, сложность которых чуть меньше исходной. В этом случае время вычислений, по сравнению с «наивными» методами, можно значительно сократить.

Как правило, чтобы решить поставленную задачу, требуется решить отдельные части задачи (подзадачи), после чего объединить решения подзадач в одно общее решение. Часто многие из этих подзадач одинаковы. Подход динамического программирования состоит в том, чтобы решить каждую подзадачу только один раз, сократив тем самым количество вычислений. Это особенно полезно в случаях, когда число повторяющихся подзадач экспоненциально велико.

Метод динамического программирования сверху — это простое запоминание результатов решения тех подзадач, которые могут повторно встретиться в дальнейшем. Динамическое программирование снизу включает в себя переформулирование сложной задачи в виде рекурсивной последовательности более простых подзадач.

1. Разработать алгоритм решения задачи с применением метода, указанного в варианте и реализовать программу.

Задача была подразбита на более малые подзадачи. Производится поиск пути до искомой клетки от предыдущей клетки. Пример работы представлен на рисунках ниже. Заполнение верхней строки и правого столбца представлено на рисунках 1-5.

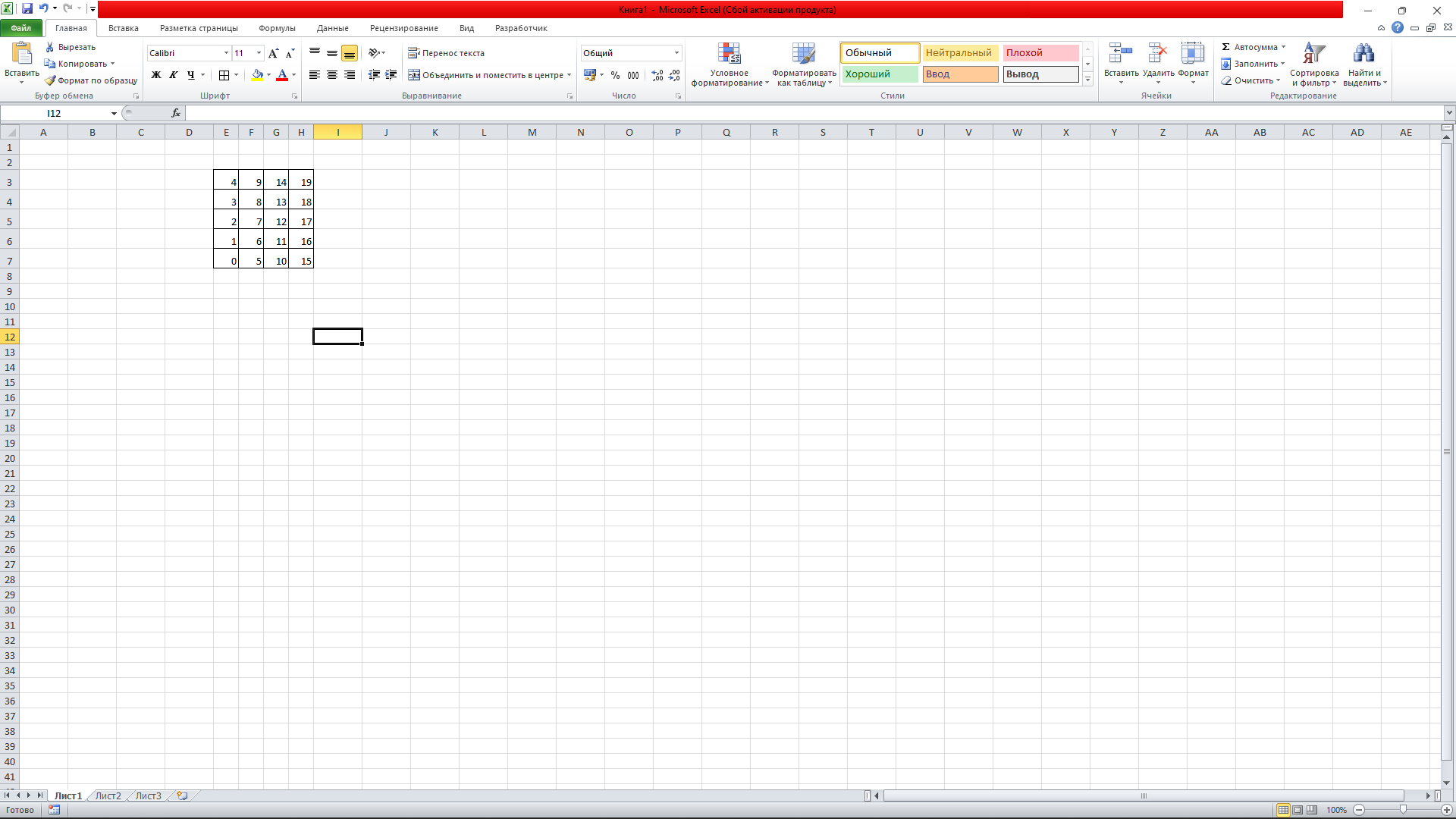


Рисунок 1. 1 шаг заполнения таблицы

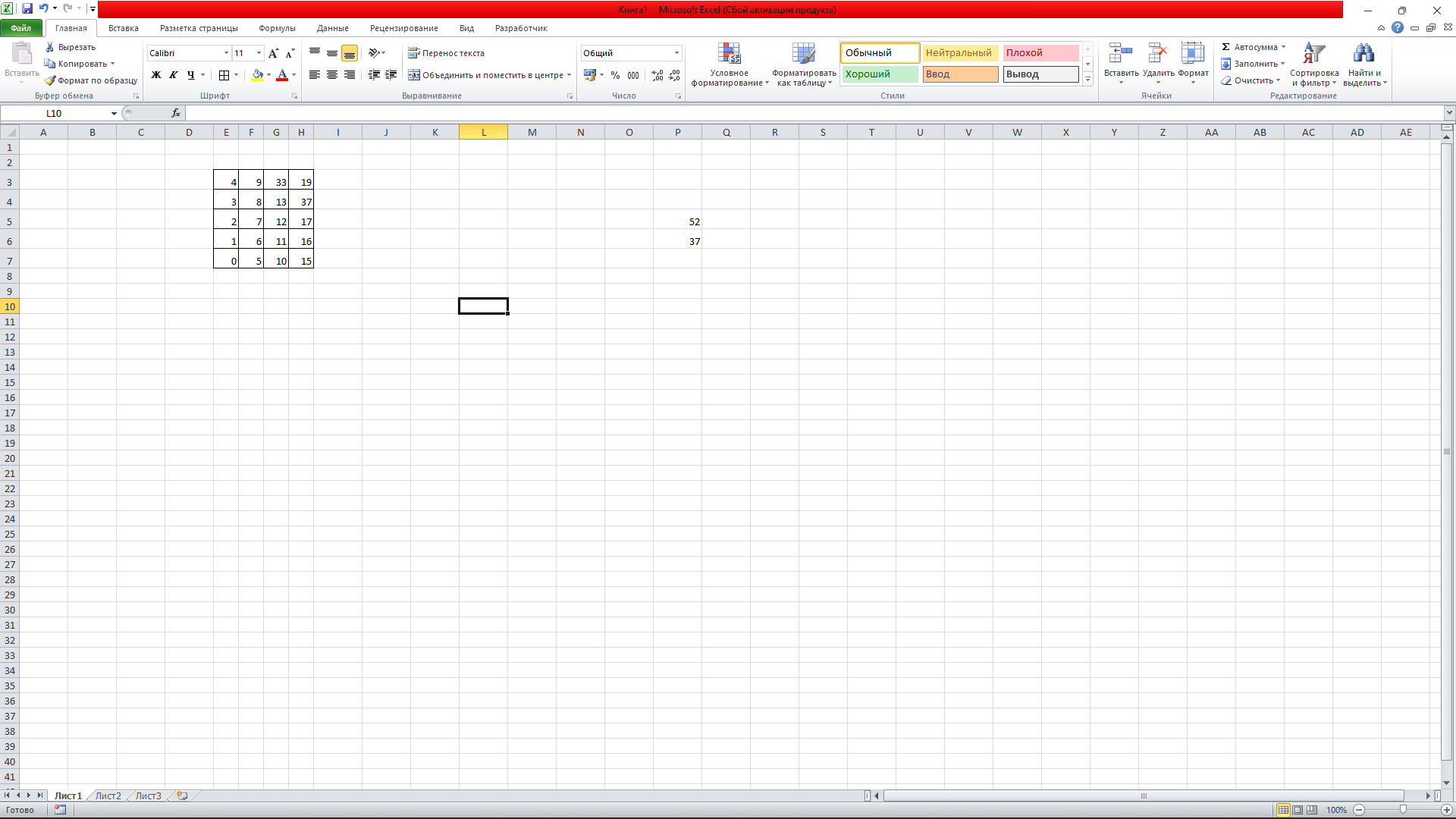


Рисунок 2. 2 шаг заполнения таблицы

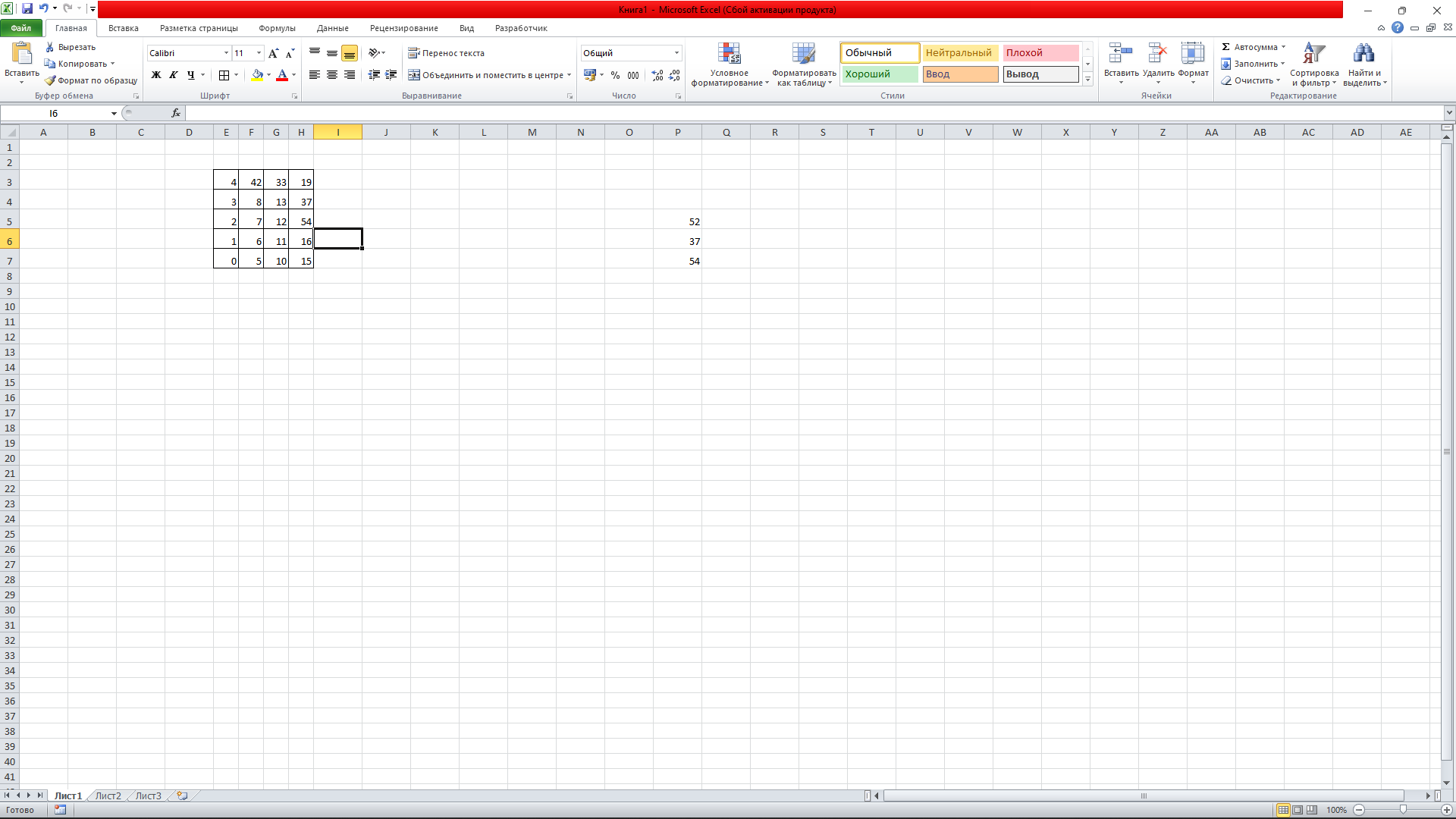


Рисунок 3. 3 шаг заполнения таблицы

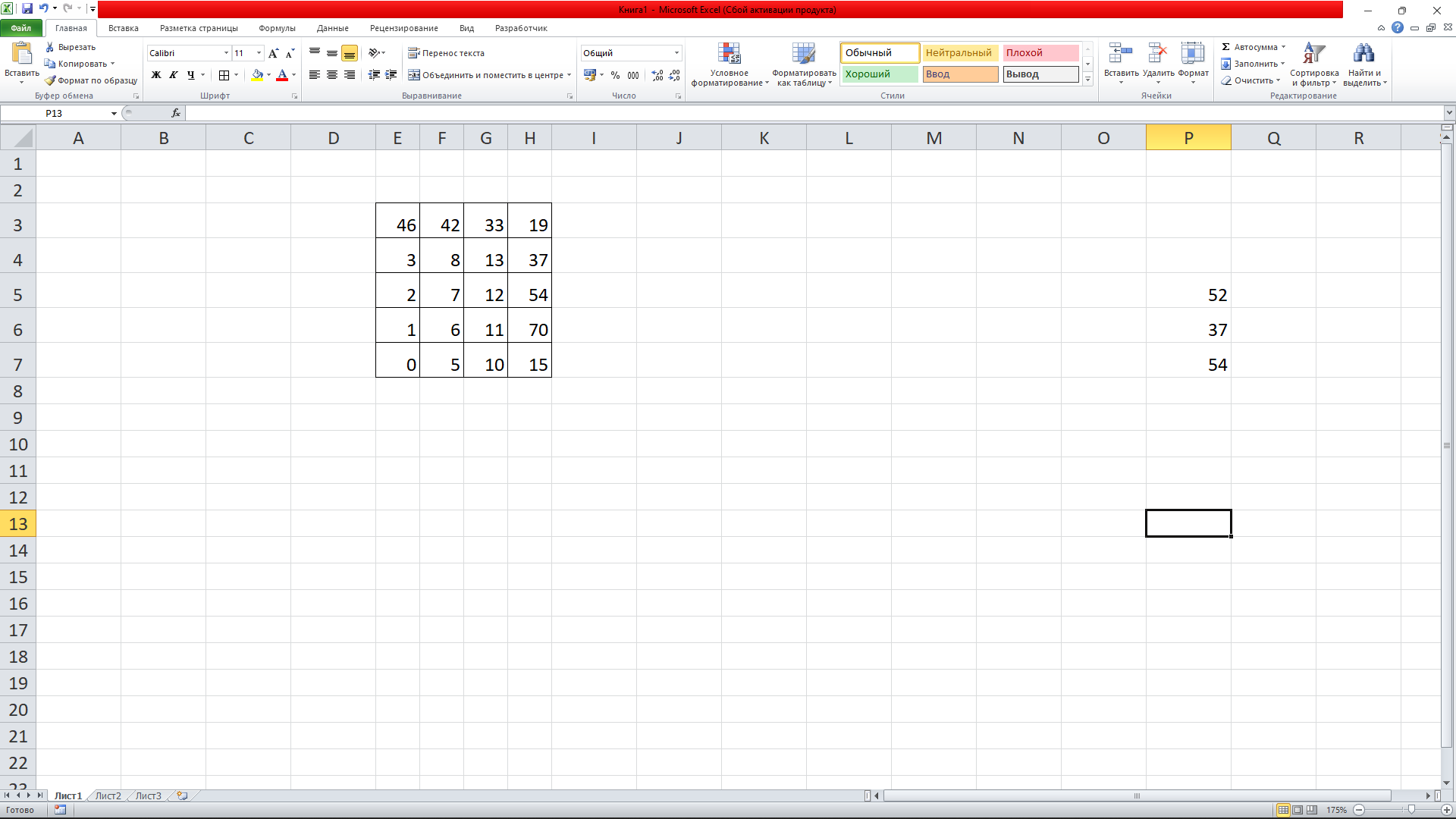


Рисунок 4. 4 шаг заполнения таблицы

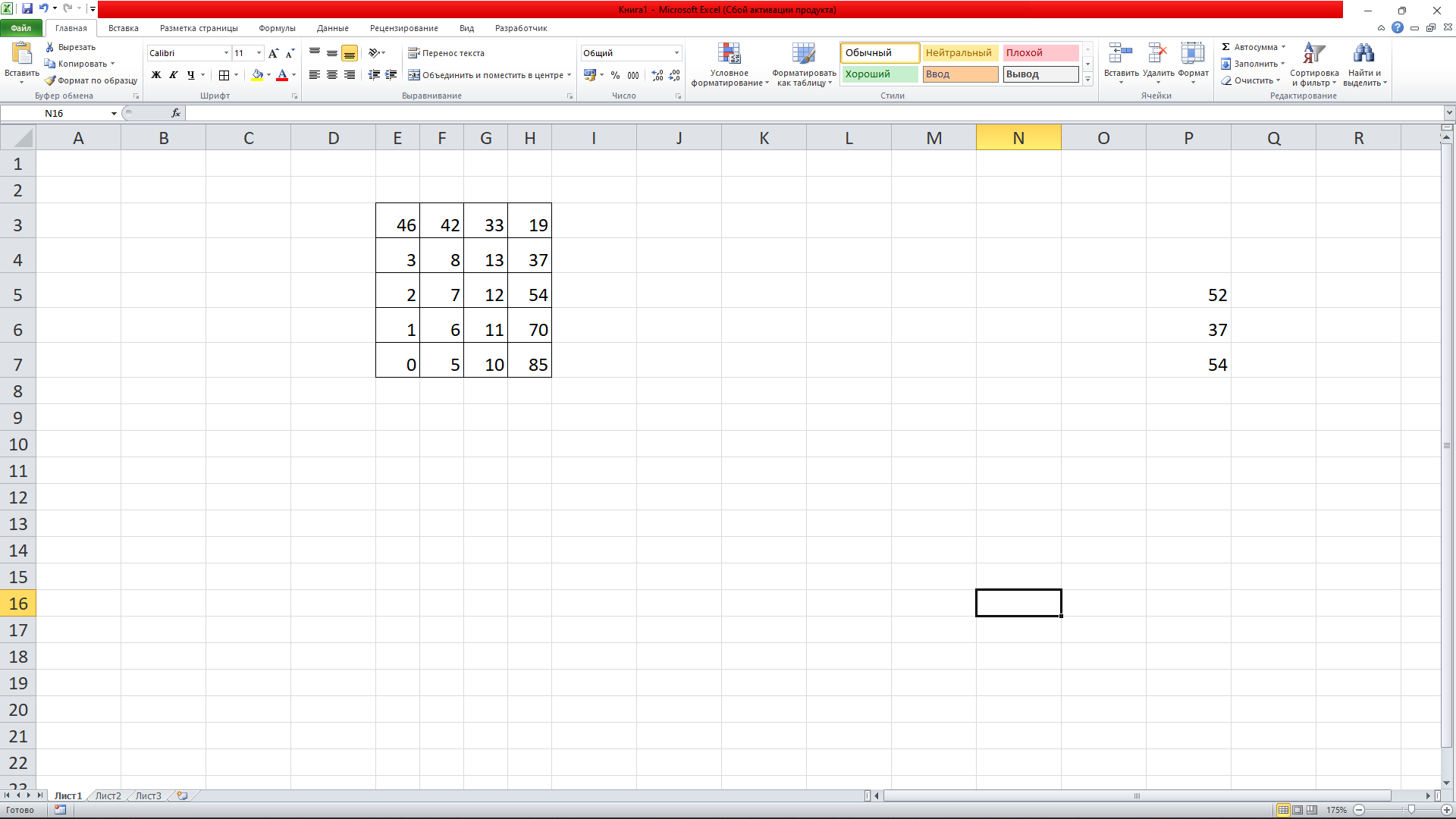


Рисунок 5. 5 шаг заполнения таблицы

После заполнения верхней строки и правого столбца, остальные клетки заполняются по принципу поиска минимального значения между суммой текущего значеничя с значением сверху и справа

|  |
| --- |
| int getWay(int x, int y)  {  if (x < b\_x&&y < b\_y)  {  return min(V[x + 1][y].back() + V[x][y].back(), V[x][y + 1].back() + V[x][y].back());  }  else if (x >= b\_x&&y < b\_y)  {  return V[x][y + 1].back() + V[x][y].back();  }  else if (x < b\_x&&y >= b\_y)  {  return V[x + 1][y].back() + V[x][y].back();  }  else  {  return V[x][y].back();  }  } |

13+37=50, 13+33=46. 46 меньше, значит в клетку 2;3 записывается 46. Дозаполнение таблицы представлено на рисунках 6-8.

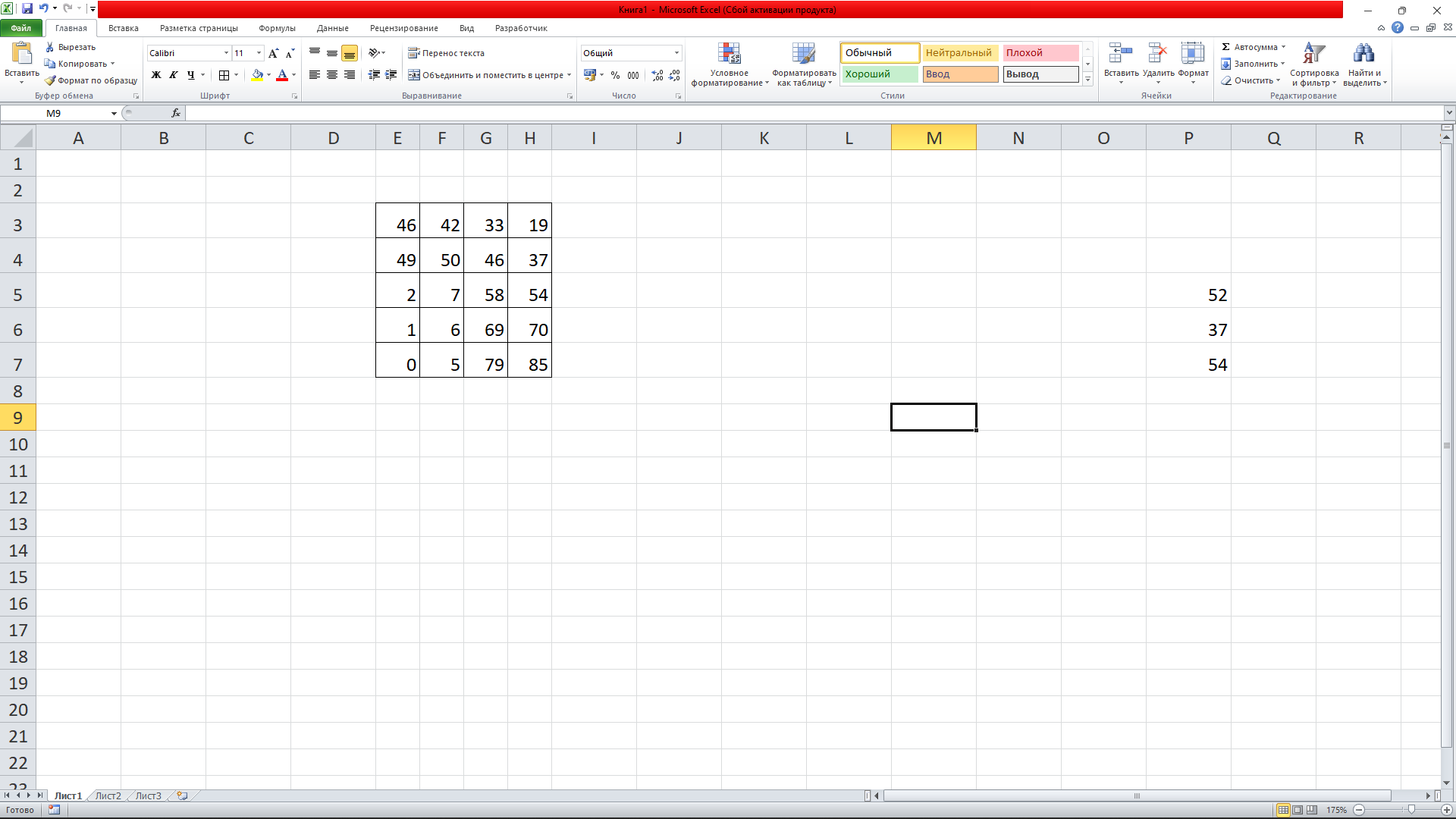


Рисунок 6. 6 шаг заполнения таблицы

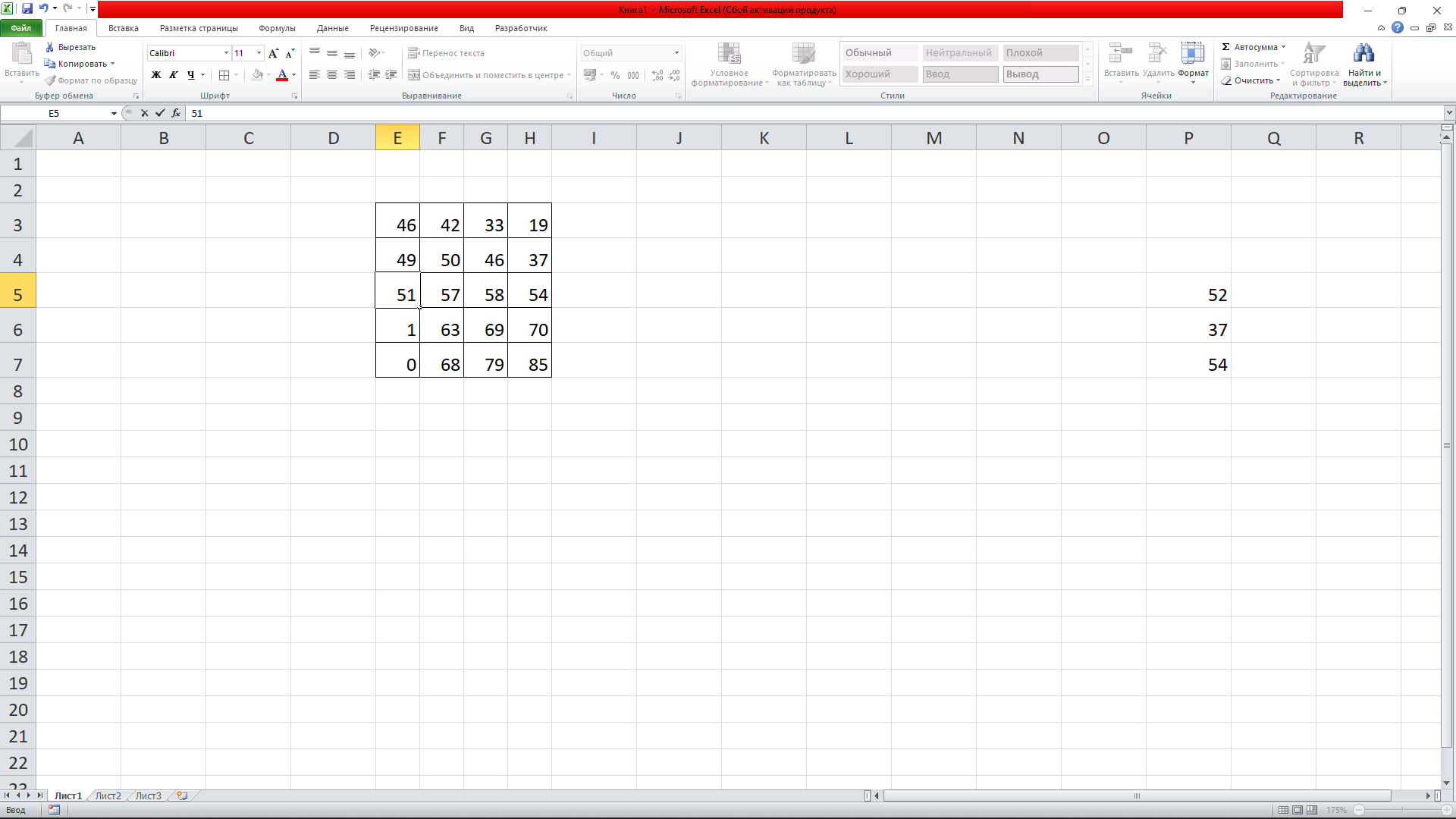


Рисунок 7. 7 шаг заполнения таблицы

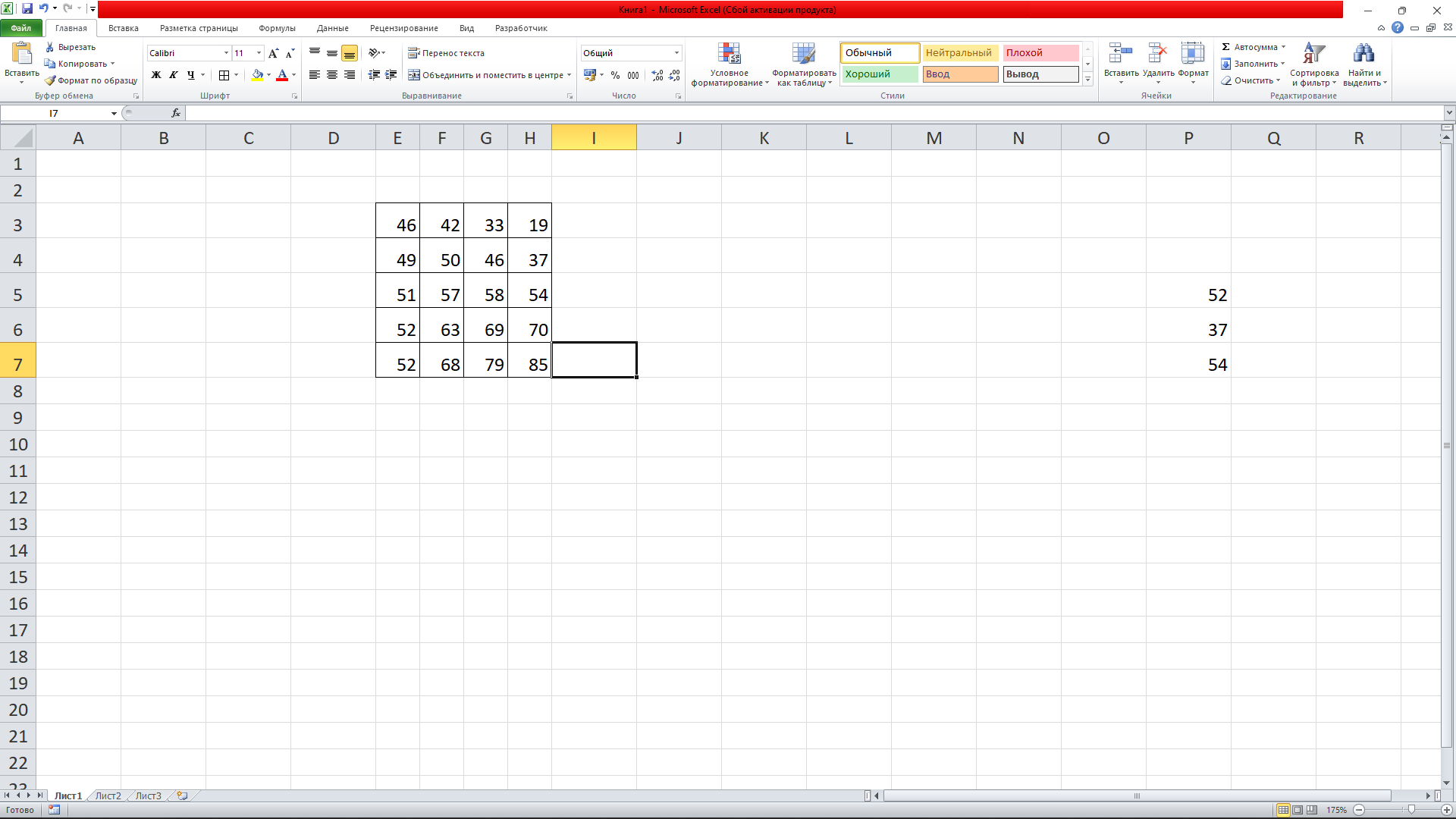


Рисунок 8. 8 шаг заполнения таблицы

Таким образом, кратчайший путь из 0;0 в 3;4 занимает 52.

Тестирование функции представлено в пункте 4 данного отчета.

# Тестирование

Тестирование приложения.

При вводе значений, как в примере, программа выдает правильный ответ. Работа программы представлена на рисунке 9.



Рисунок 9. Работа программы

Таким образом, после проведения тестирования можно подтвердить, что программа работает правильно.

# Вывод

В ходе выполнения практической работы были выполнены все задания, а также:

1. Изучен принцип работы динамического программирования;
2. Улучшены знания о векторах.

# Исходный код программы

Source.cpp

|  |
| --- |
| #include "Source.h"  using namespace std;  int main()  {  cout << "enter start x position" << endl;  cin >> a\_x;  cout << "enter start y position" << endl;  cin >> a\_y;  cout << "enter finish x position" << endl;  cin >> b\_x;  cout << "enter finish y position" << endl;  cin >> b\_y;  task();    cout << "the shortest way between " << a\_x << ";" << a\_y << " and " << b\_x << ";" << b\_y << " is " << V[a\_x][a\_y].back() << endl;  system("pause");  } |

Source.h

|  |
| --- |
| #pragma once  #include <iostream>  #include <vector>  using namespace std;  vector <int> V[100][100];  int a\_x = 0, a\_y = 0, b\_x = 2, b\_y = 3, v\_speed = 3, h\_speed = 2;  int min(int a, int b)  {  if (a < b) return a;  else if (b < a) return b;  else return a;  }  int getWay(int x, int y)  {  if (x < b\_x&&y < b\_y)  {  return min(V[x + 1][y].back() + V[x][y].back(), V[x][y + 1].back() + V[x][y].back());  }  else if (x >= b\_x&&y < b\_y)  {  return V[x][y + 1].back() + V[x][y].back();  }  else if (x < b\_x&&y >= b\_y)  {  return V[x + 1][y].back() + V[x][y].back();  }  else  {  return V[x][y].back();  }  }  void task()  {  if (a\_x >= b\_x || a\_y >= b\_y)  {  cout << "no way" << endl;  system("pause");  exit(1);  }  V[a\_x][a\_y].push\_back(0);  V[a\_x][a\_y].push\_back(0);  for (int i = a\_x + 1; i <= b\_x; i++)  {  V[i][a\_y].push\_back(i);  V[i][a\_y].push\_back(0);  }  for (int i = a\_y + 1; i <= b\_y; i++)  {  V[a\_x][i].push\_back(0);  V[a\_x][i].push\_back(i);  }  for (int x = a\_x + 1; x <= b\_x; x++)  {  for (int y = a\_y + 1; y <= b\_y; y++)  {  V[x][y].push\_back(x);  V[x][y].push\_back(y);  }  }  int k;  for (int i = a\_x; i <= b\_x; i++)  {  for (int v = a\_y; v <= b\_y; v++)  {  cout << "enter weight of " << i << ";" << v << endl;  cin >> k;  V[i][v].push\_back(k);  }  }  for (int x = b\_x; x >= a\_x; x--)  {  for (int y = b\_y; y >= a\_y; y--)  {  int u = getWay(x, y);  V[x][y].pop\_back();  V[x][y].push\_back(u);  }  }  } |