|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | |  |  |  | | 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 № 2** | |
| **по дисциплине** | |
| **«**Структуры и алгоритмы обработки данных**»**  **Тема: «Хеширование и организация быстрого поиска данных»** | |
|  | |
| Выполнил студент группы ИКБО-14-20 | Зарожина Я.А**.** |
| Принял преподаватель | Муравьева Е.А. |

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

Москва 2022

# **Цель работы**

Получение навыков разработки хеш-таблиц и их применения.

# **Постановка задачи**

1. Разработать программу, которая использует хеш-таблицу для организации прямого доступа к записям двоичного файла, структура записи которого приведена в варианте.

Разработайте класс хеш-таблицы и реализуйте методы для операций:

1) Хеширование (хеш-функция составляется самостоятельно).

2) Чтение записи из файла и вставка записи в таблицу (запись включает: ключ и номер записи с этим ключом в файле). При записи явно показывать, какой индекс был присвоен записи.

3) Удаление записи из файла.

4) Поиск и вывод записи с заданным ключом.

5) Вывод содержимого файла.

6) Выполнение рехеширования.

1. Провести тестирование программы на небольших объемах данных, введенных вручную. Разработанные тесты должны покрывать случаи возникновения коллизий и рехеширования.
2. Заполнить файл большим объемом данных. Провести исследование времени чтения записей из начала, середины и конца файла
3. Составить отчет, отобразив в нем описание выполнения всех этапов разработки, тестирования и код всей программы со скриншотами результатов тестирования.

Вариант №11. Условие задания:

|  |  |  |
| --- | --- | --- |
| № | Тип хеш-таблицы (способ реализации коллизий) | Структура записи файла. Ключи записей подчеркнуты |
| 11 | Открытый адрес (двойное хеширование) | Железнодорожная справка: номер поезда, пункт отправления, пункт назначения, время отправления |

# **Решение**

Хэш-таблица служит для удобного хранения какого-либо значения, которое может быть возвращено из определённой ячейки таблицы при помощи ключа, который присваивается данной ячейке. Для создания такого ключа используется хэш-функция. В идеальном случае хэш-функция создавать уникальный ключ – “хеш” для каждой ячейки. Но такое случается крайне редко. На практике хэш-функции работают таким образом, что на одну ячейку может создаться сразу несколько хешей. Такие ситуации называются коллизией. Но и для такой ситуации есть методы решения.

Первый метод – цепное хеширование. Смысл его заключается в том, что ключи с одним и тем же хешем хранятся в одном линейном списке, дополняемым в конце в случае возникновения коллизии. Тогда поиск искомой ячейки сводится к определению хеша искомого ключа, а затем к линейному поиску по списку, соответствующему вычисленному хешу.

Второй – рехеширование, заключающееся в том, что хеш вычисляется повторно, либо иной хеш-функцией от ключа, либо функцией, меняющей уже вычисленный хеш. Наиболее распространённым является применение функций, инкрементирующих хеш. Тогда поиск искомой пары/адреса для вставки сводится к определению хеша искомого ключа, а затем к поиску по массиву, с хешем в качестве переменной-индекса.

Был разработан класс хеш-таблицы, которая содержала структуру ячейки таблицы. Сама хеш-таблица:

|  |
| --- |
| class HeshFun  {  public:  int n;  bool b = false;  Reference\* a;  void NewStruct()  {  DeleterA();  a = new Reference[n];  for (int j = 0; j < n; j++)  {  a[j] = { "", "", "", "" };  cout << a[j].numb;  }  b = true;  }  void DeleterA()  {  delete[] a;  }  int toHash(string key)  {  int number, hash;  number = stoi(key);  hash = number / 3;  if (n != 1)  hash = hash % n % (n / 2);//получения номера для хэша в первой части таблицы  else  hash = hash % n;  return hash;  }  int SecondHash(int oldhash)  {  int newhash;  if (n % 2 == 0)  {  newhash = oldhash + n-1 ;  newhash = newhash % n ;  }  else  {  newhash = oldhash + 2;  newhash = newhash % n;  }  return newhash;  }  ~HeshFun()  {  b = false;  DeleterA();  }  }; |

Структура ячейки:

|  |
| --- |
| struct Reference  {  string numb= "";//ключ 1  string op = "";  string ip = "";  string time = "";  }; |

Также в этом классе был создан метод хеширования. Смысл его заключается в том, что если таблица состоит более чем из одного элемента, то первое хеширование возвращает одно из значений в первой половине таблицы. Первая хеш-функция:

|  |
| --- |
| int toHash(string key)  {  int number, hash;  number = stoi(key);  hash = number / 3;  if (n != 1)  hash = hash % n % (n / 2);//получения номера для хэша в первой части таблицы  else  hash = hash % n;  return hash;  } |

Если ячейка с таким хешом уже занята, то свою работу начинает уже метод второго хеширования(рехеширования). Его смысл заключается в том, что если хеш-таблица состоит из четного количества элементов, то метод возвращает значение предыдущей ячейки, если количество – нечетное, то возвращает значение ячейки, находящейся через шаг. Вторая хеш-функция:

|  |
| --- |
| int SecondHash(int oldhash)  {  int newhash;  if (n % 2 == 0)  {  newhash = oldhash + n-1;  newhash = newhash % n ;  }  else  {  newhash = oldhash + 2;  newhash = newhash % n;  }  return newhash;  } |

Метод ввода элемента в файл:

|  |
| --- |
| void WriteFile(int n, Reference a[])//запись в файл элемента  {  fstream f("file.bin", ios::binary | ios::app);  if (!f.is\_open())  {  f.close();  fstream f("file.bin", ios::binary);//если файла не существует, то создаётся  for (int i = 0; i < n; i++)  {  f.write(reinterpret\_cast<char\*>(&a[i]), sizeof(Reference));  }  }  else  {  for (int i = 0; i < n; i++)  {  f.write(reinterpret\_cast<char\*>(&a[i]), sizeof(Reference));  }  }  f.close();  } |

Метод удаления информации из бинарного файла. Для удаления информации из файла, функция сначала создаёт список, далее записывает в список всю информацию из файла, удаляет файл и создаёт новый, но уже без удаляемого поезда. Метод удаления информации:

|  |
| --- |
| bool DeleteinFile(string x)//удаление записи в файле  {  bool b = false;  List\* l;  ifstream f("file.bin", ios::binary);  if (f.is\_open())  {  while (true)  {  l = CreatList();  if (!f.read(reinterpret\_cast<char\*>(&l->numb), sizeof(string)))  break;  f.read(reinterpret\_cast<char\*>(&l->op), sizeof(string));  f.read(reinterpret\_cast<char\*>(&l->ip), sizeof(string));  f.read(reinterpret\_cast<char\*>(&l->time), sizeof(string));  }  l = Head;  Head = l->pnext;  free(l);  }  f.close();  ofstream fo("file.bin", ios::binary | ofstream::trunc | ofstream::out);  for (List\* l = Head; l;)  {  if (l->numb != x)  {  fo.write(reinterpret\_cast<char\*>(&l->numb), sizeof(string));  fo.write(reinterpret\_cast<char\*>(&l->op), sizeof(string));  fo.write(reinterpret\_cast<char\*>(&l->ip), sizeof(string));  fo.write(reinterpret\_cast<char\*>(&l->time), sizeof(string));  }  else  b = true;  Head = l->pnext;  free(l);  l = Head;  }  free(Head);  Head = nullptr;  f.close();  return b;  } |

Метод чтения содержимого файла:

|  |
| --- |
| int ReadFile()//чтение из файла  {  int i = 0;  List\* l;  ifstream f("file.bin", ios::binary);  if (f.is\_open())  {  while (true)  {  i++;  l = CreatList();  if (!f.read(reinterpret\_cast<char\*>(&l->numb), sizeof(string)))  break;  f.read(reinterpret\_cast<char\*>(&l->op), sizeof(string));  f.read(reinterpret\_cast<char\*>(&l->ip), sizeof(string));  f.read(reinterpret\_cast<char\*>(&l->time), sizeof(string));  }  l = Head;  Head = l->pnext;  free(l);  }  f.close();  return i - 1;  } |

Метод заполнения хеш-таблицы:

|  |
| --- |
| int DoHashtable(HeshFun \*H)  {  int numberhash, max;  List\* l;  max = ReadFile();  H->n = max;  H->NewStruct();  l = Head;  while (l != nullptr)  {  numberhash = H->toHash(l->numb);  if (H->a[numberhash].numb != "")  {  while (H->a[numberhash].numb != "")  numberhash = H->SecondHash(numberhash);  }  H->a[numberhash].numb = l->numb;  H->a[numberhash].ip = l->ip;  H->a[numberhash].op = l->op;  H->a[numberhash].time = l->time;  cout << "Для поезда с номером: " << l->numb << " было присвоено значение: " << numberhash << "\n";  l = l->pnext;  }  DeleteList();  return max;  } |

Метод поиска значений по ключу:

|  |
| --- |
| Reference FindRef(string key, HeshFun \*H)  {  int newkey = H->toHash(key);  if (H->a[newkey].numb == key)  return H->a[newkey];  else  {  for (int i = 1; i < H->n; i++)  {  newkey = H->SecondHash(newkey);  if (H->a[newkey].numb == key)  return H->a[newkey];  }  }  return { "", "", "", "" };  } |

При первом заходе в программу, пользователь видит предложение программы ввести значения (опция является необязательной), при согласии пользователь видит поле для ввода поезда (Рис. 1), при отказе пользователь видит основной интерфейс.

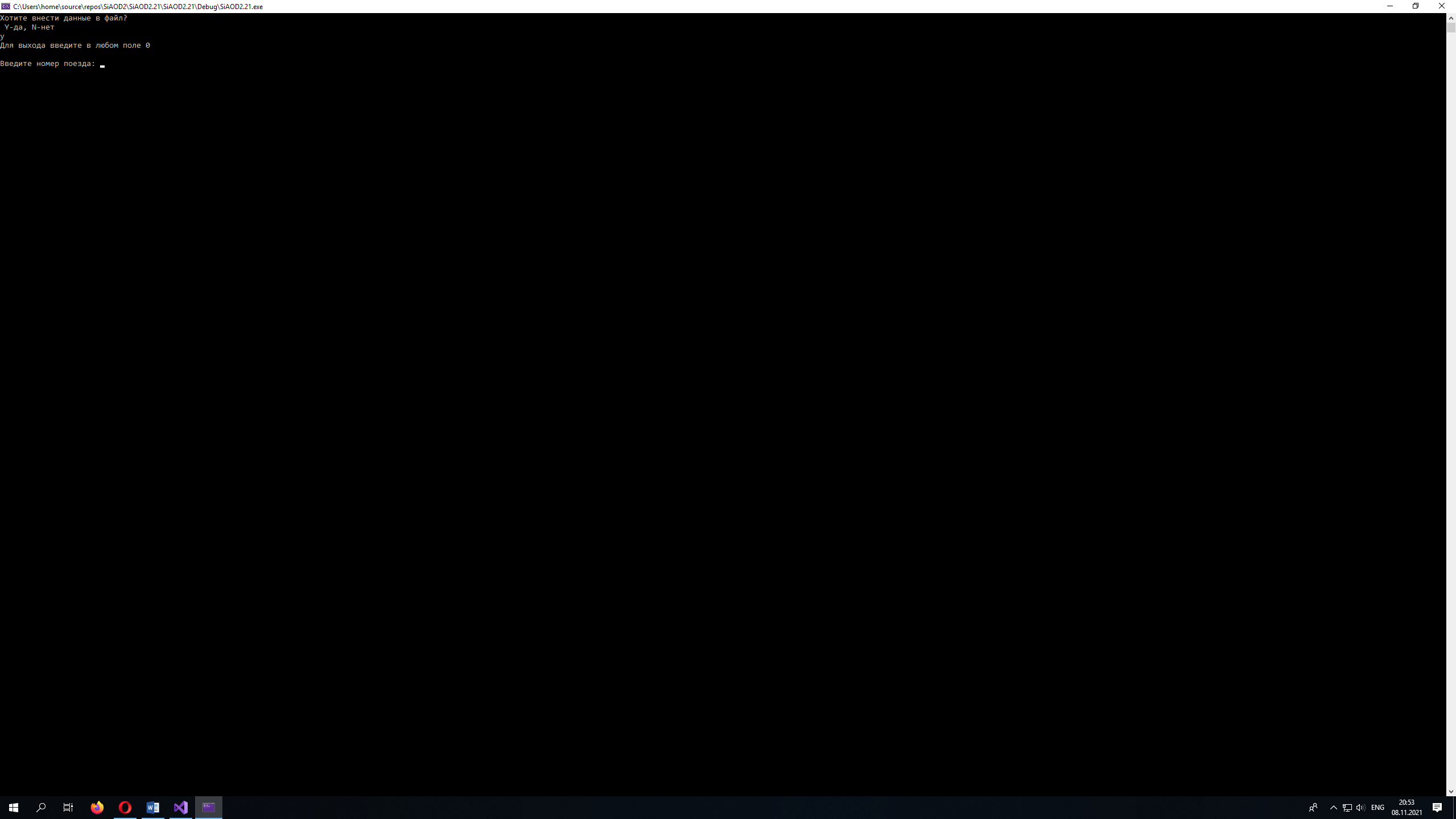


Рис. 1 Первый вход в программу

В основном интерфейсе пользователь видит приглашение для ввода номера задания или завершения работы программы (цифра 0)(Рис. 2).

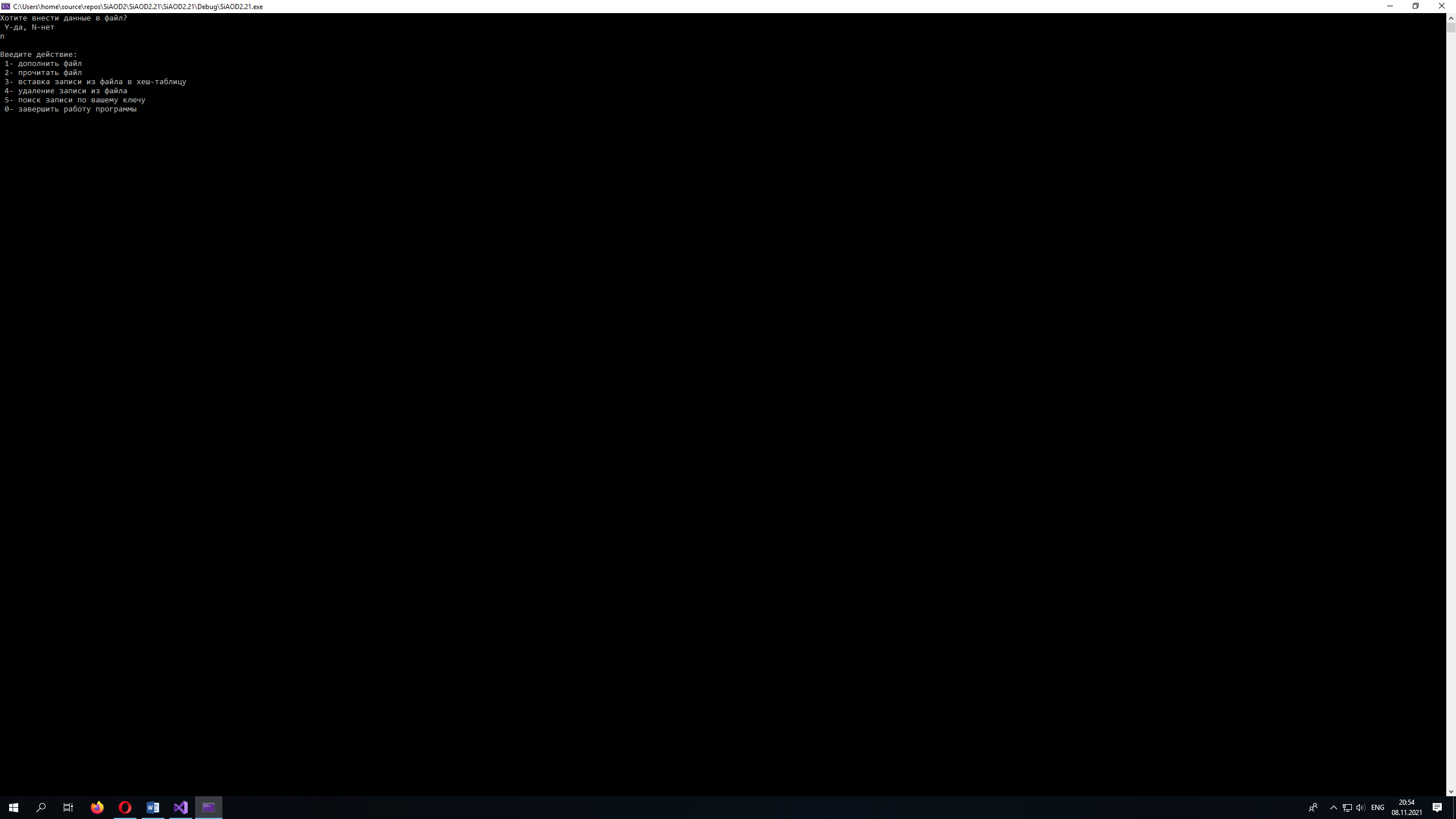


Рис. 2 Интерфейс программы

# **Тестирование**

Тестовые прогоны:

В первом прогоне используется таблица длинной в 5 элементов, сначала создаётся и заполняется файл (Рис. 3).

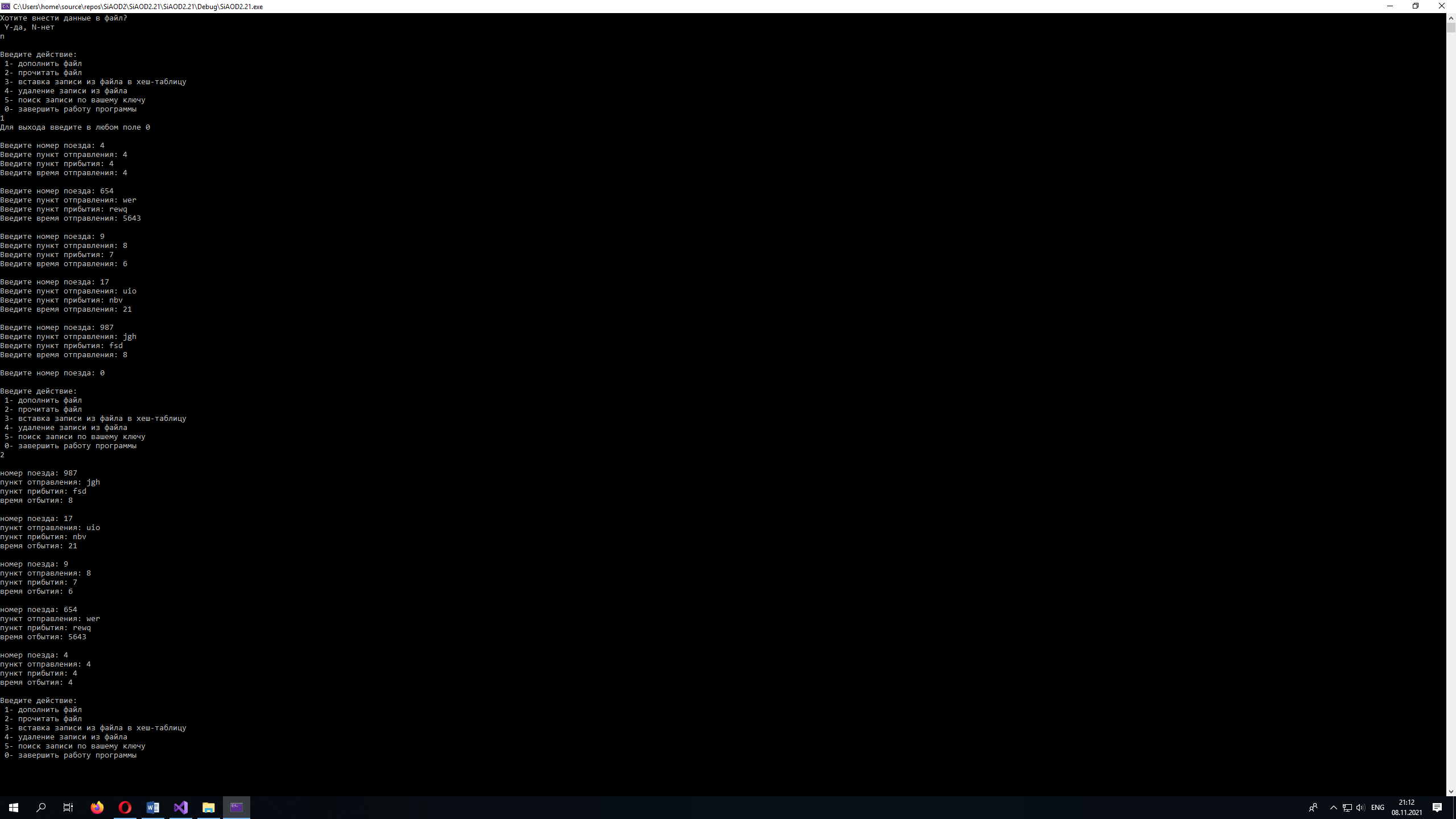


Рис. 3 Заполнение файла первого тестового прогона

Далее проверяются данные в файле (Рис. 4).

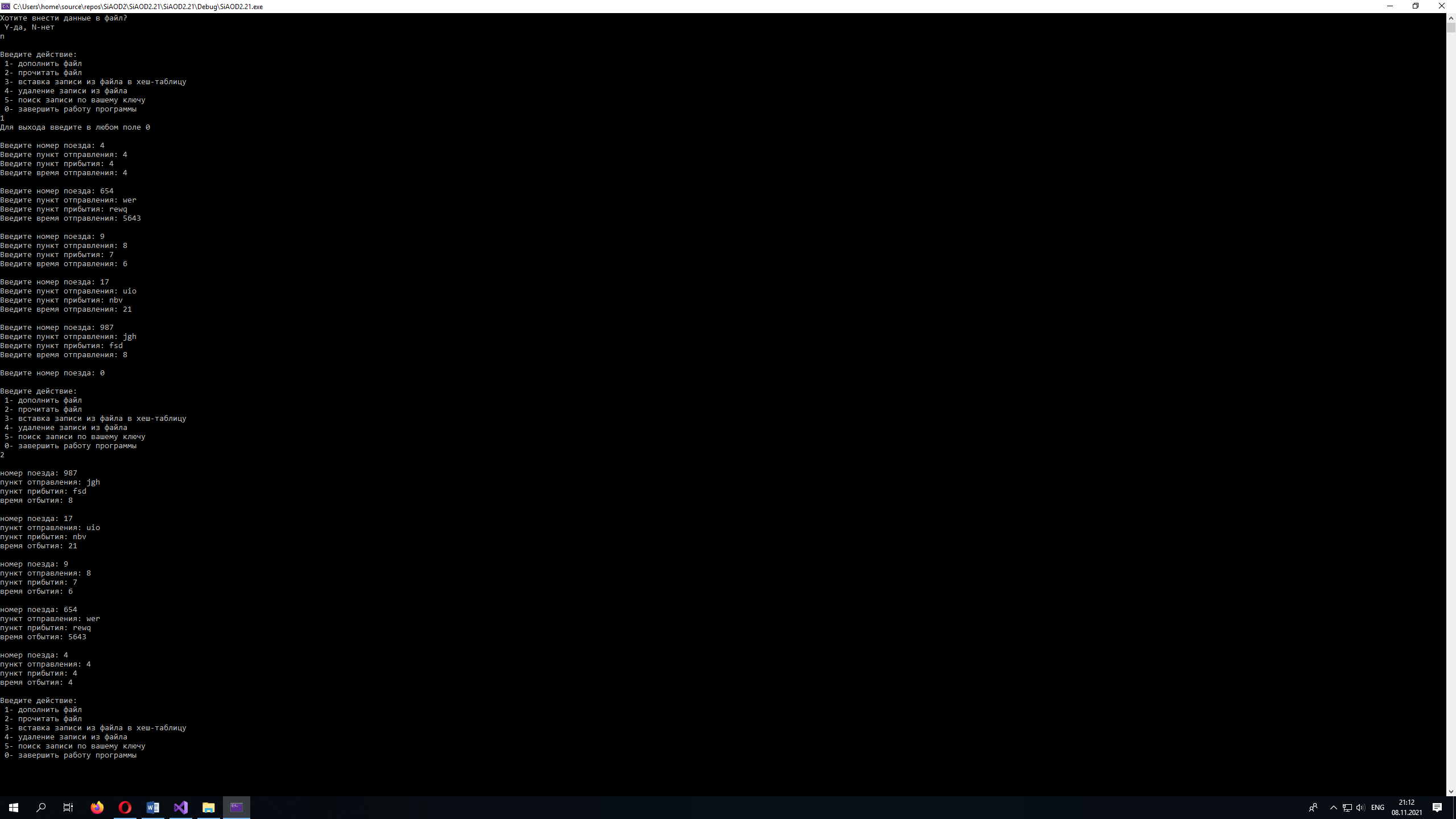


Рис. 4 Вывод данных первого тестового прогона

Заполнение хеш-таблицы текущими данными (Рис. 5).

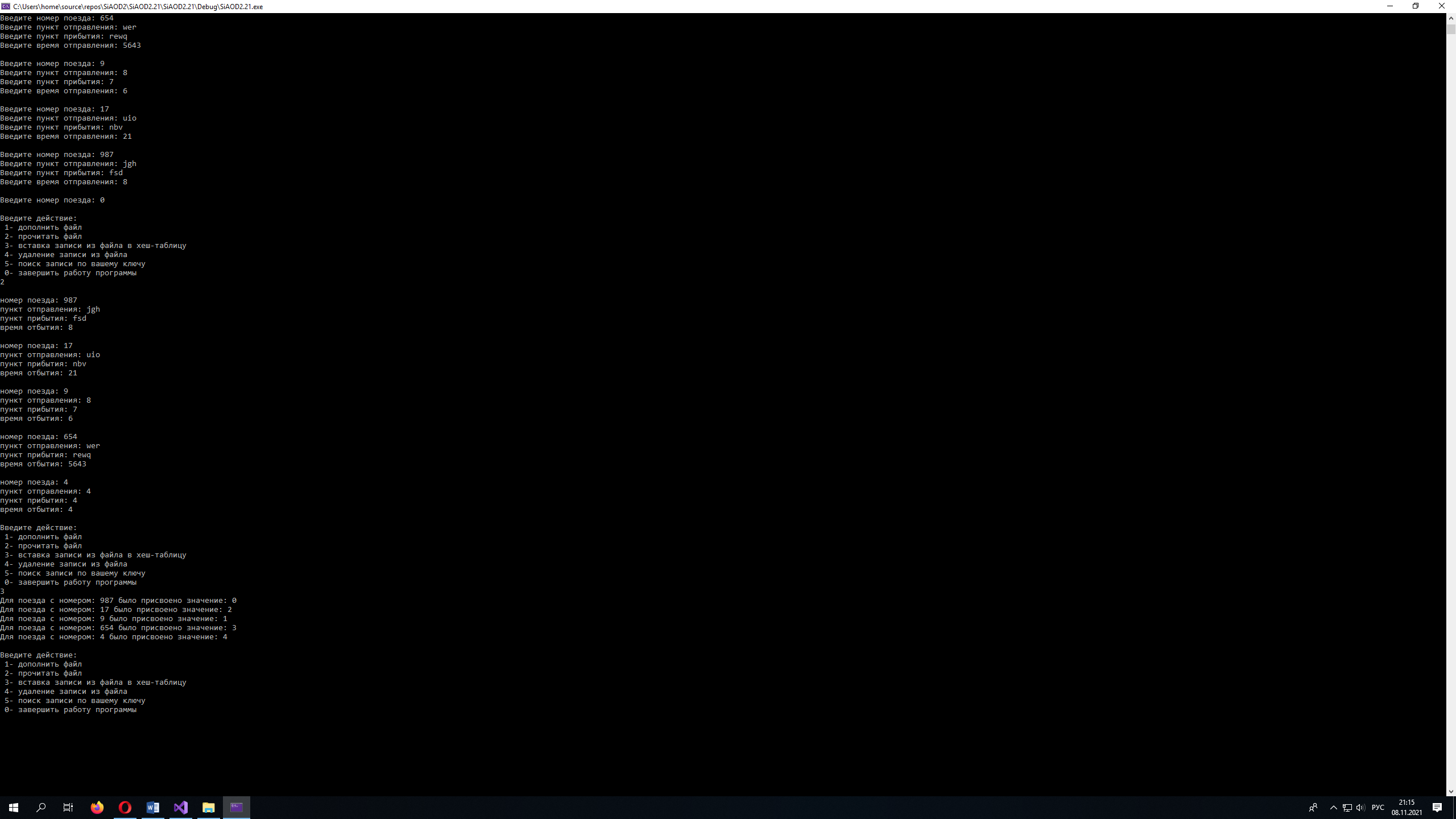


Рис. 5 Заполнение хеш-таблицы первого тестового прогона

Удаление указанного пользователем поезда (Рис. 6)

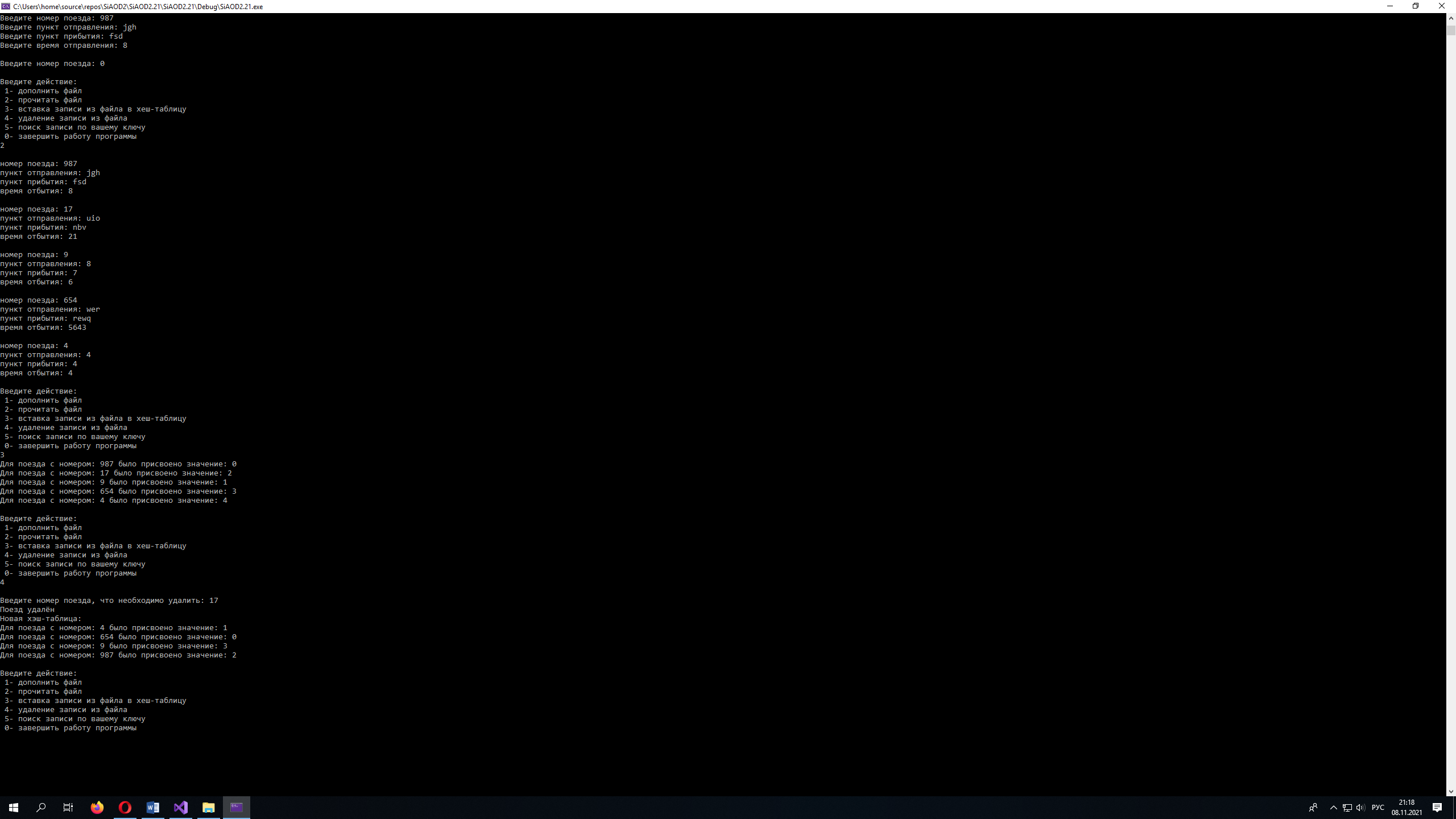


Рис. 6 Удаление элемента первого тестового прогона

Поиск записи с заданным ключом (Рис. 7)

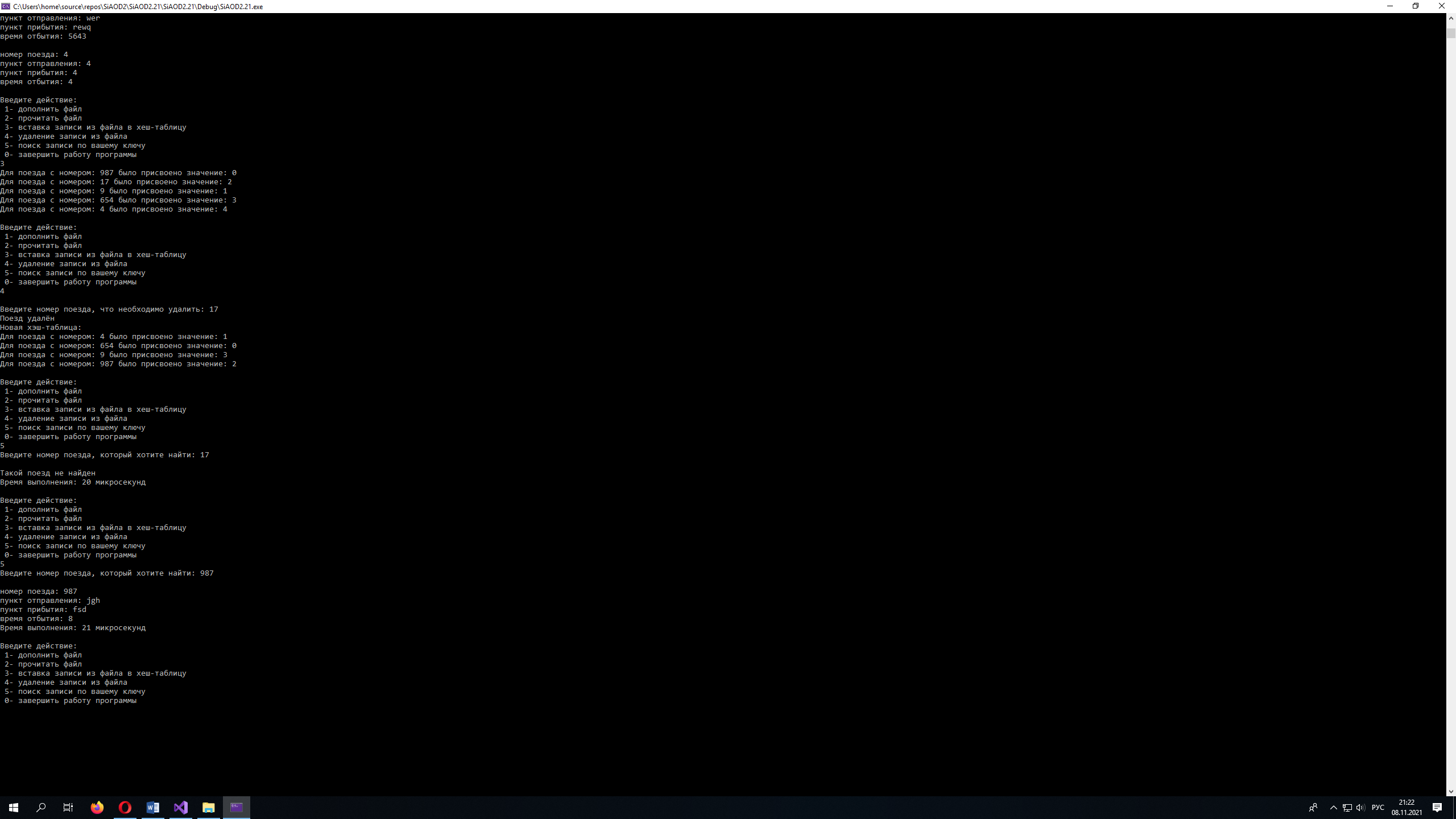


Рис. 7 Поиск по ключу первого тестового прогона

Во втором прогоне используется таблица длинной в 20 элементов, сначала создаётся и заполняется файл (Рис. 8).

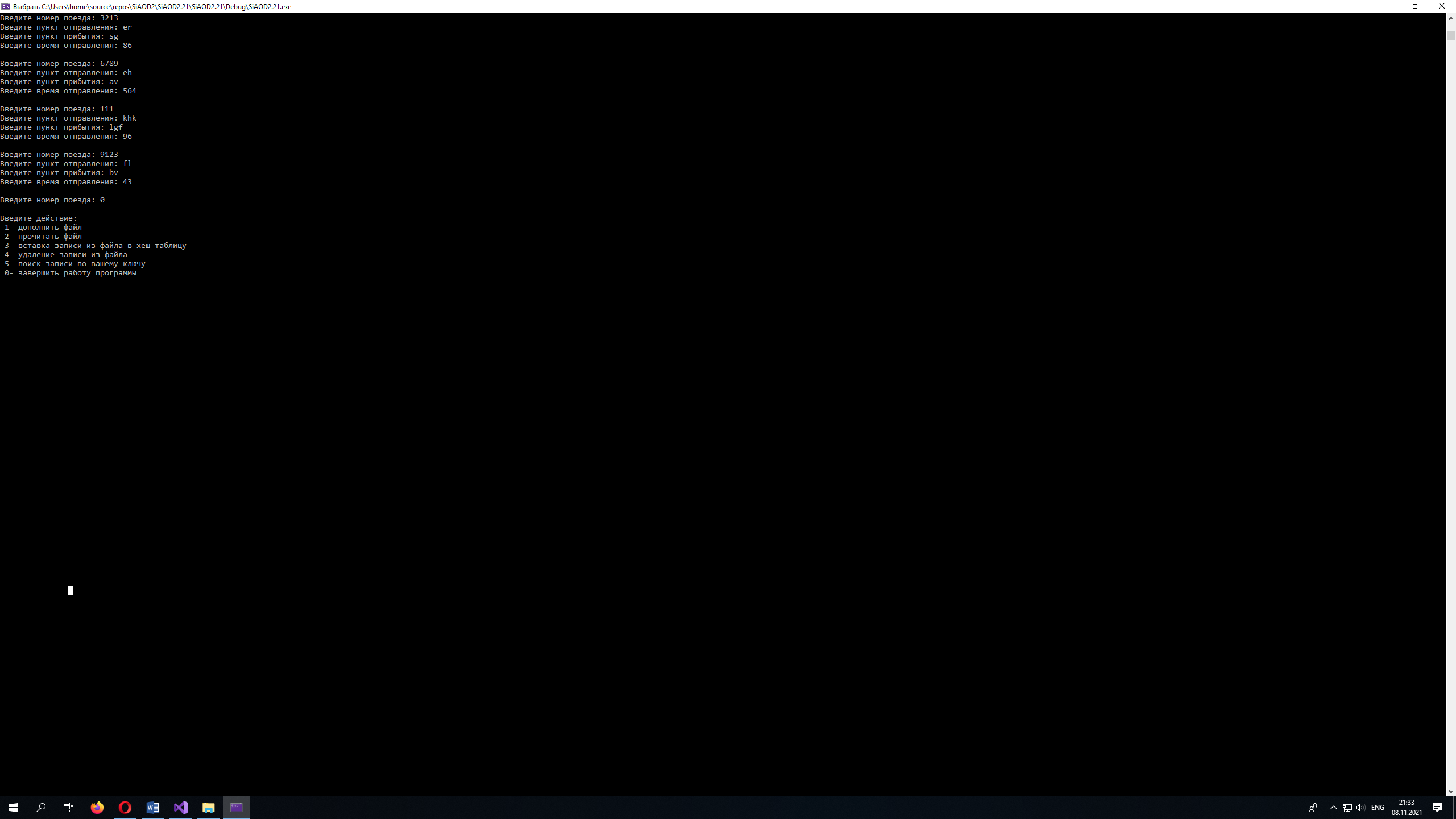
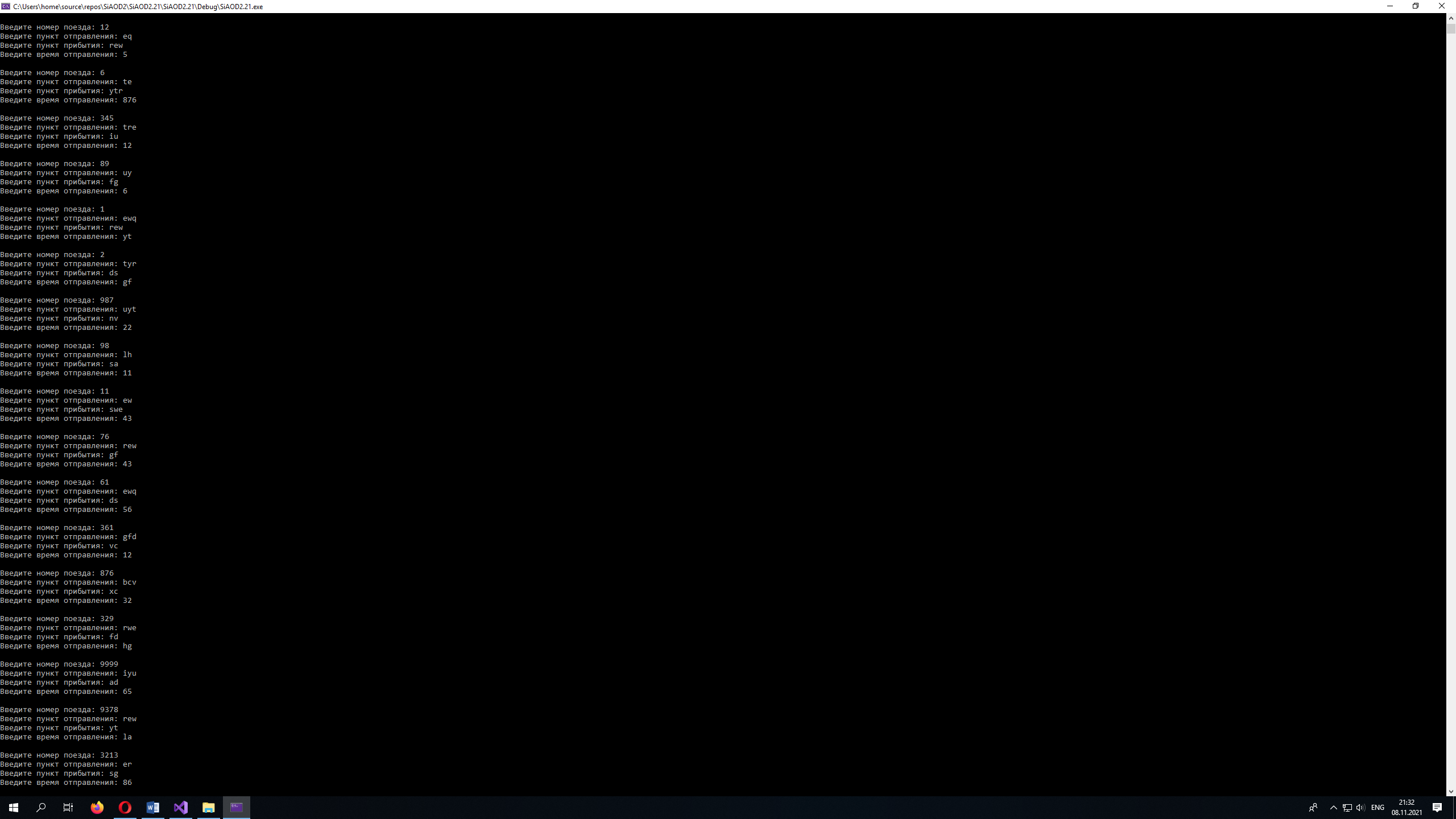


Рис. 8 Заполнение файла второго тестового прогона

Далее проверяются данные в файле (Рис. 9).

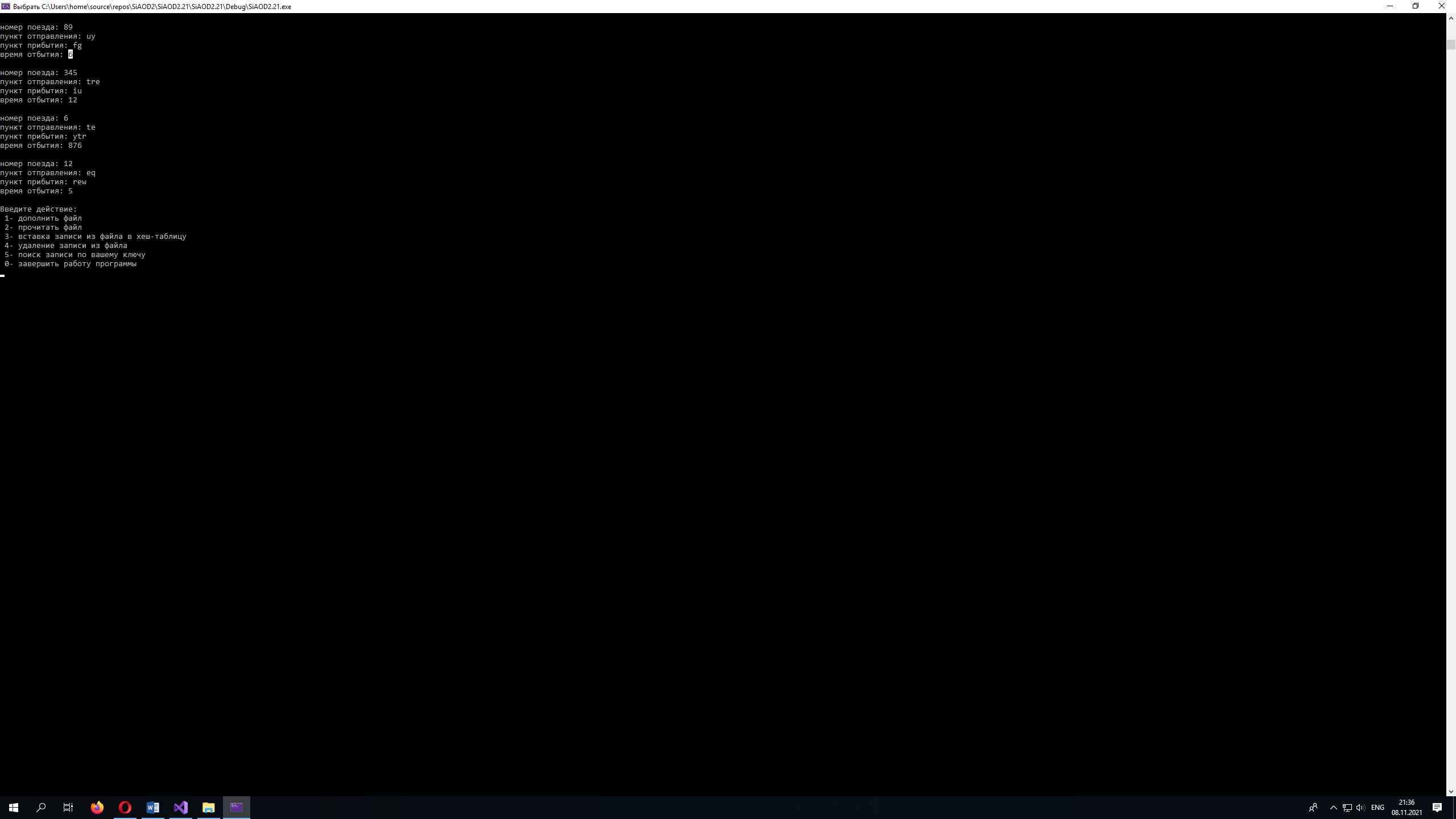
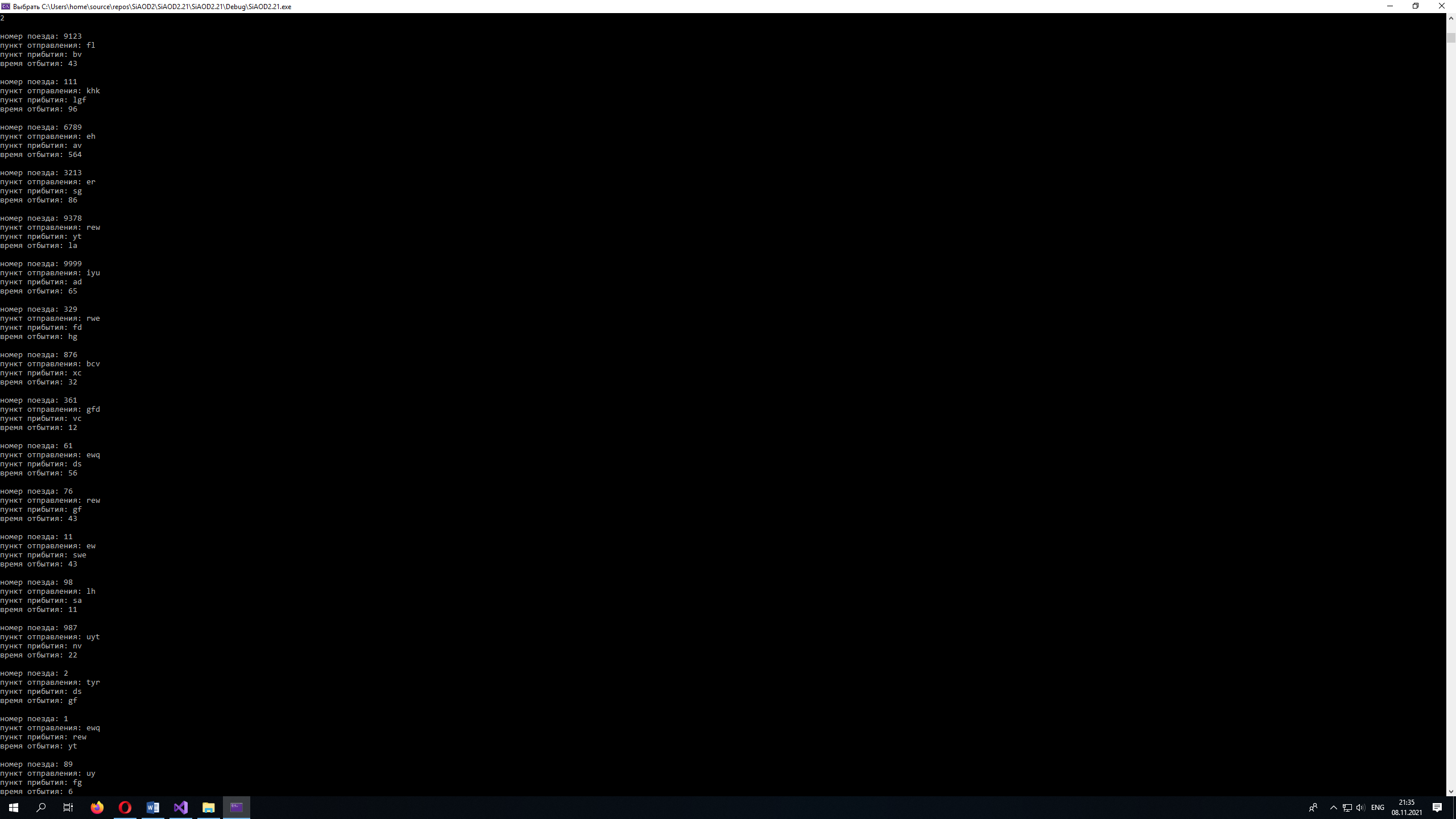


Рис. 9 Вывод данных второго тестового прогона

Заполнение хеш-таблицы текущими данными (Рис. 10).

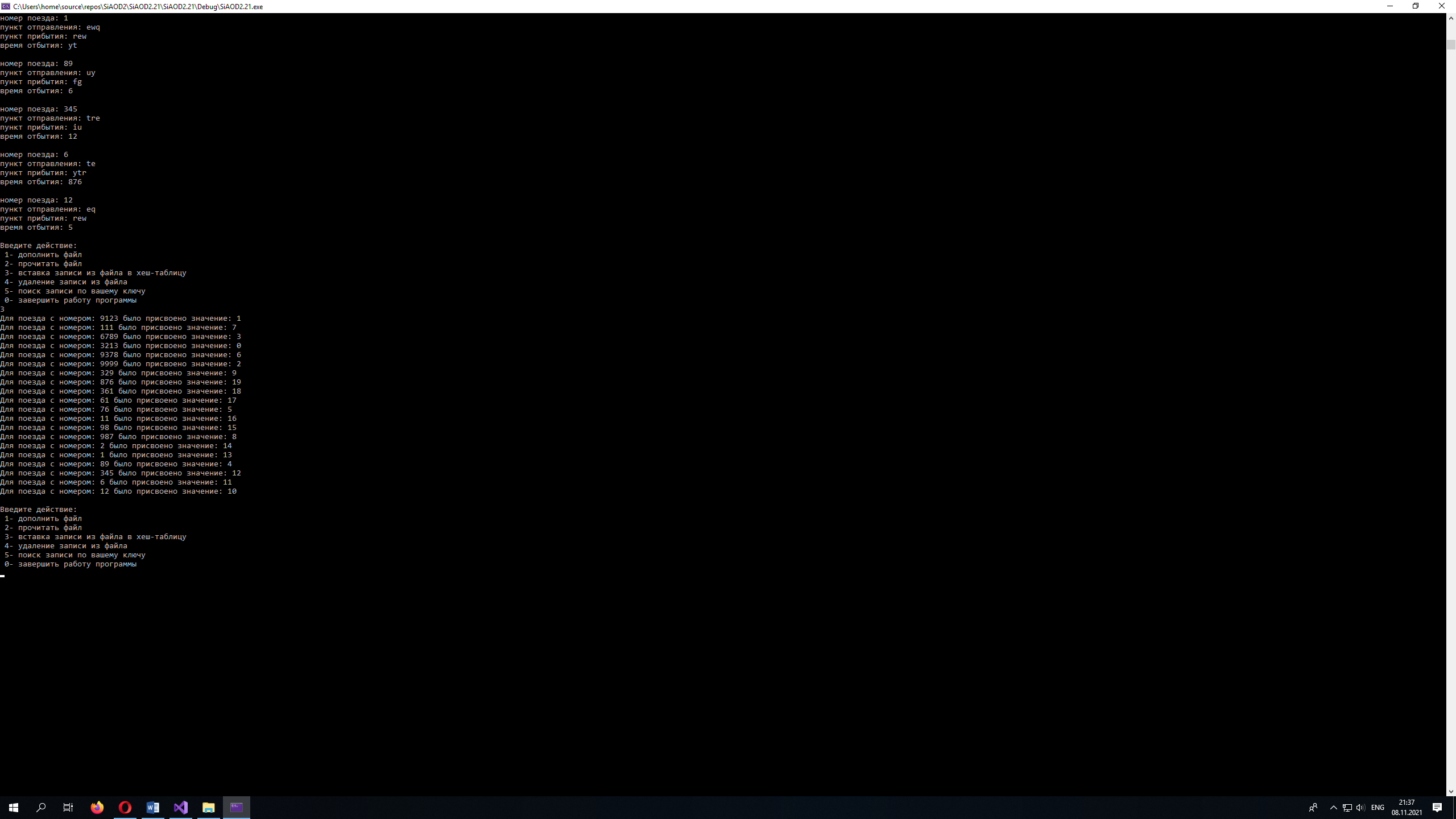


Рис. 10 Заполнение хеш-таблицы второго тестового прогона

Удаление указанного пользователем поезда (Рис. 11)

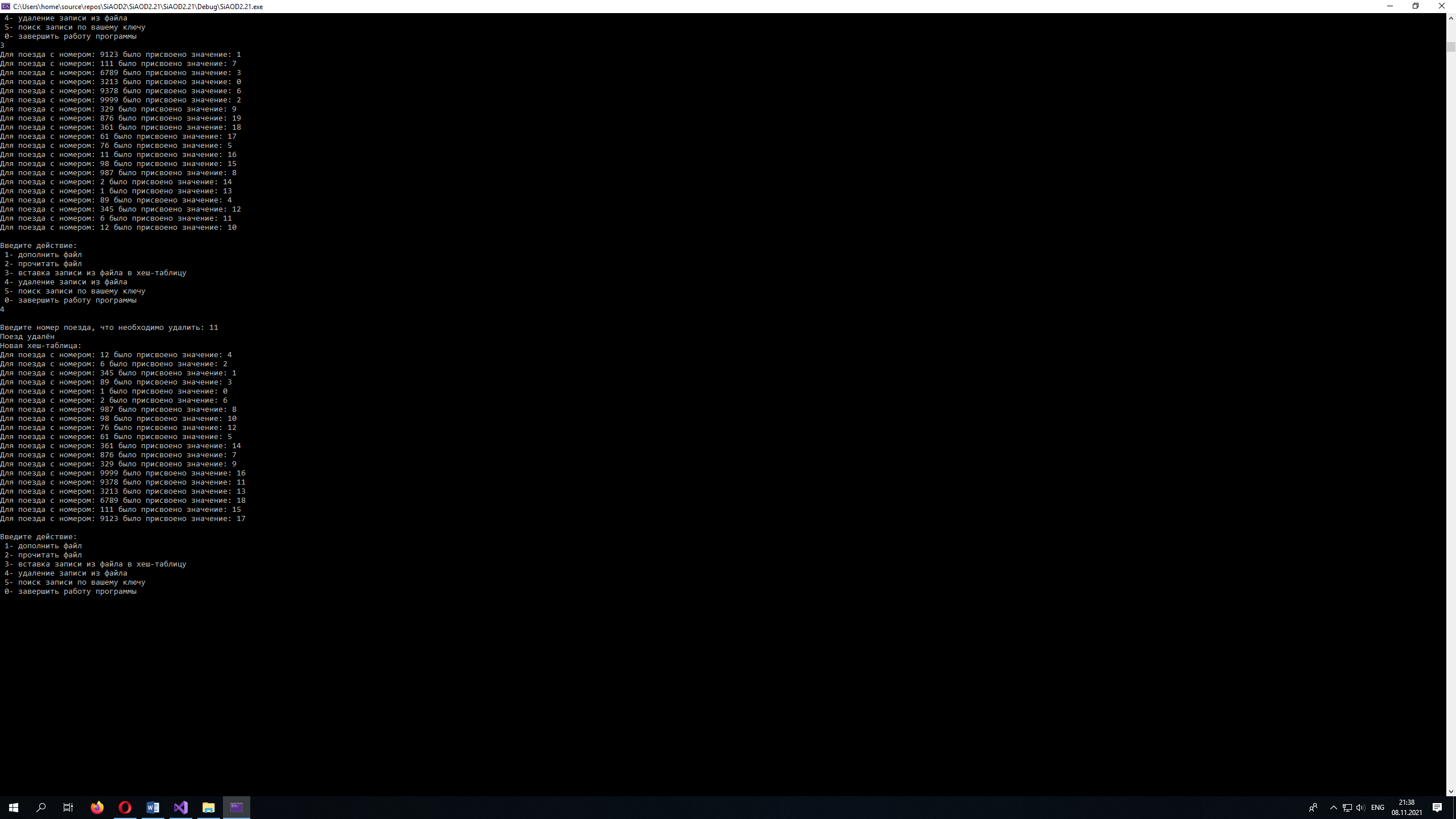


Рис. 11 Удаление элемента второго тестового прогона

Поиск записи с заданным ключом (Рис. 12)

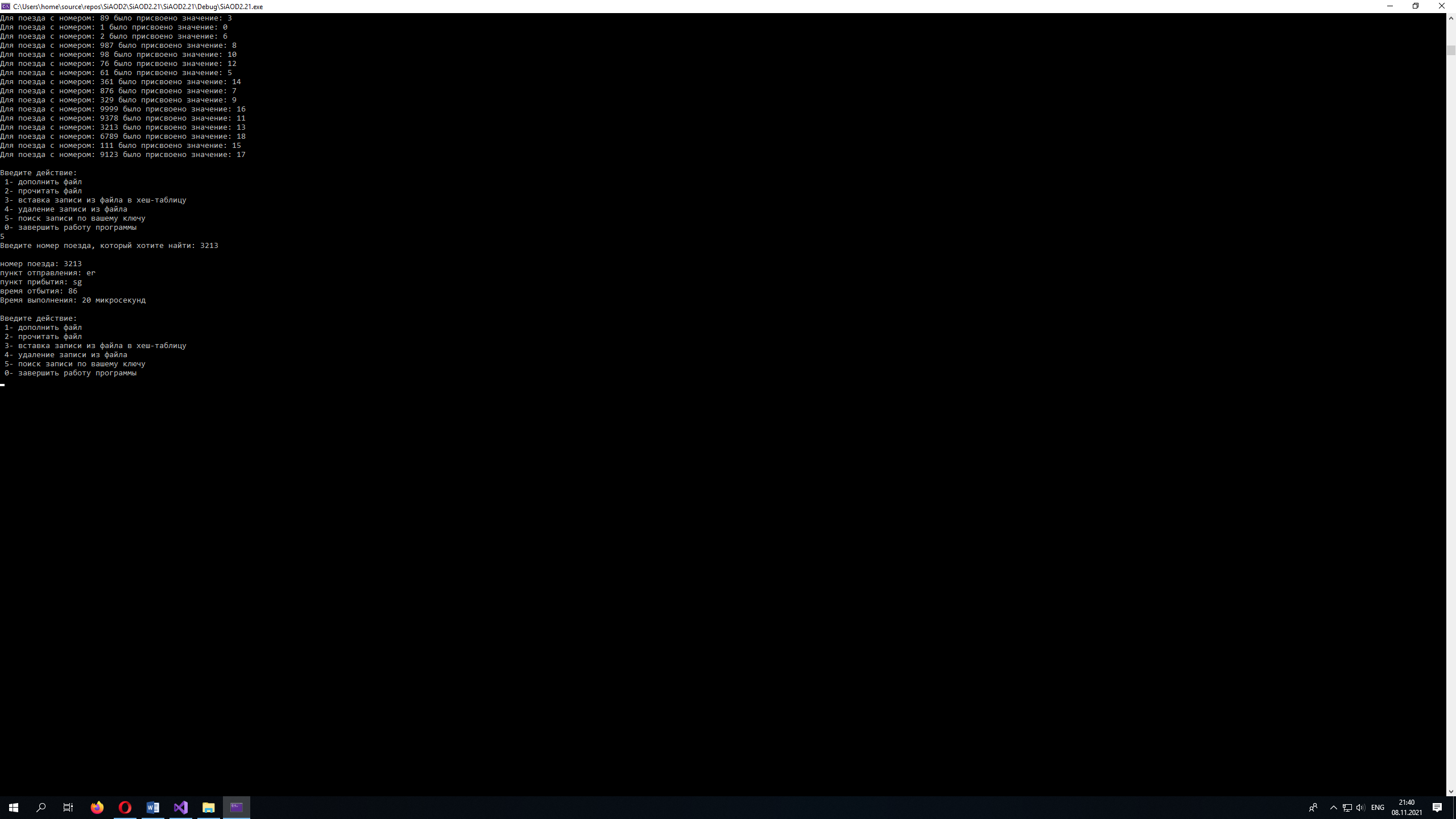


Рис. 12 Поиск по ключу второго тестового прогона

Рабочие прогоны:

Время поиска первого элемента первого рабочего прогона (Рис. 13)

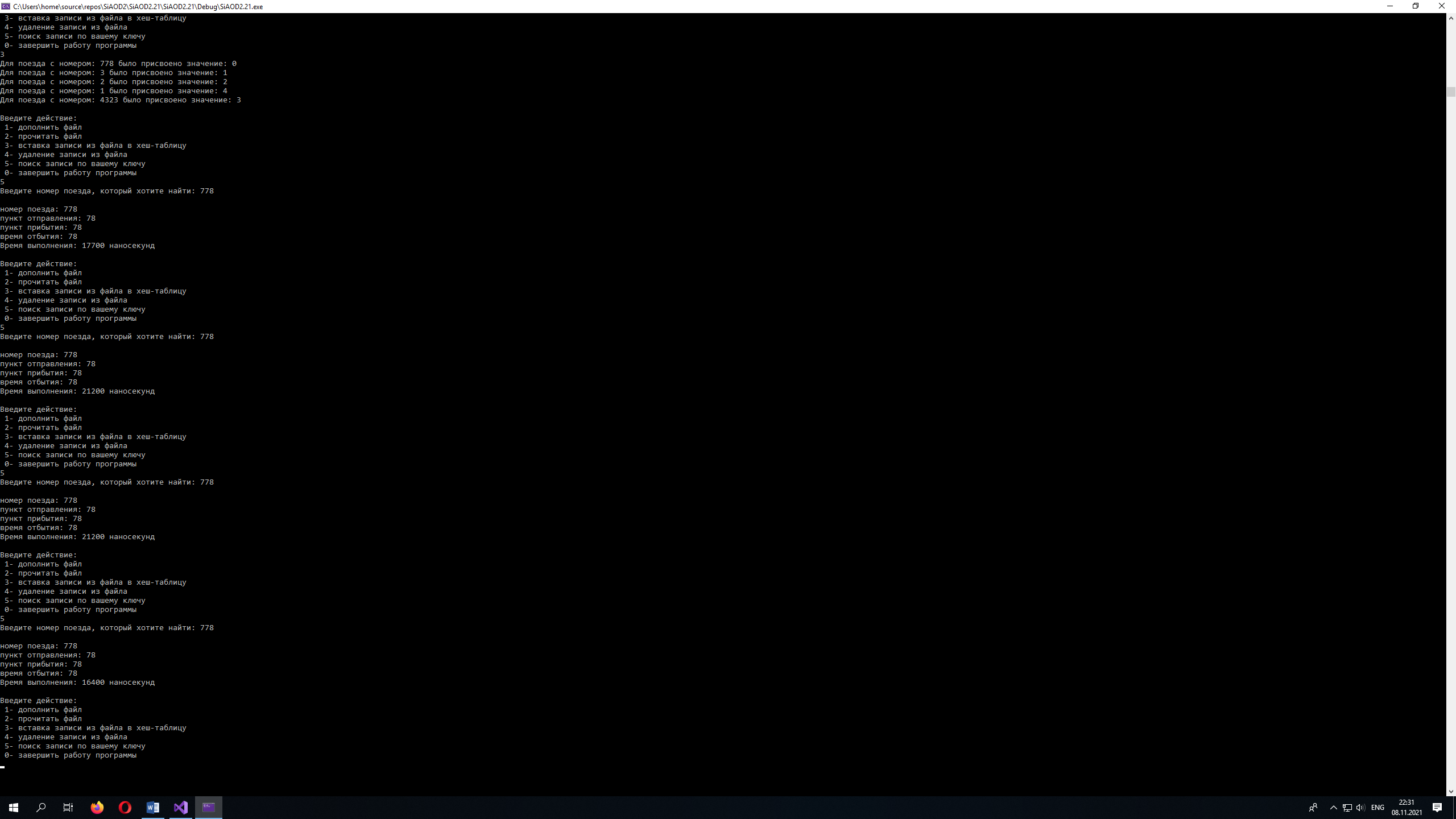


Рис. 13 Время поиска первого элемента первого рабочего прогона

Время поиска среднего элемента первого рабочего прогона (Рис. 14)

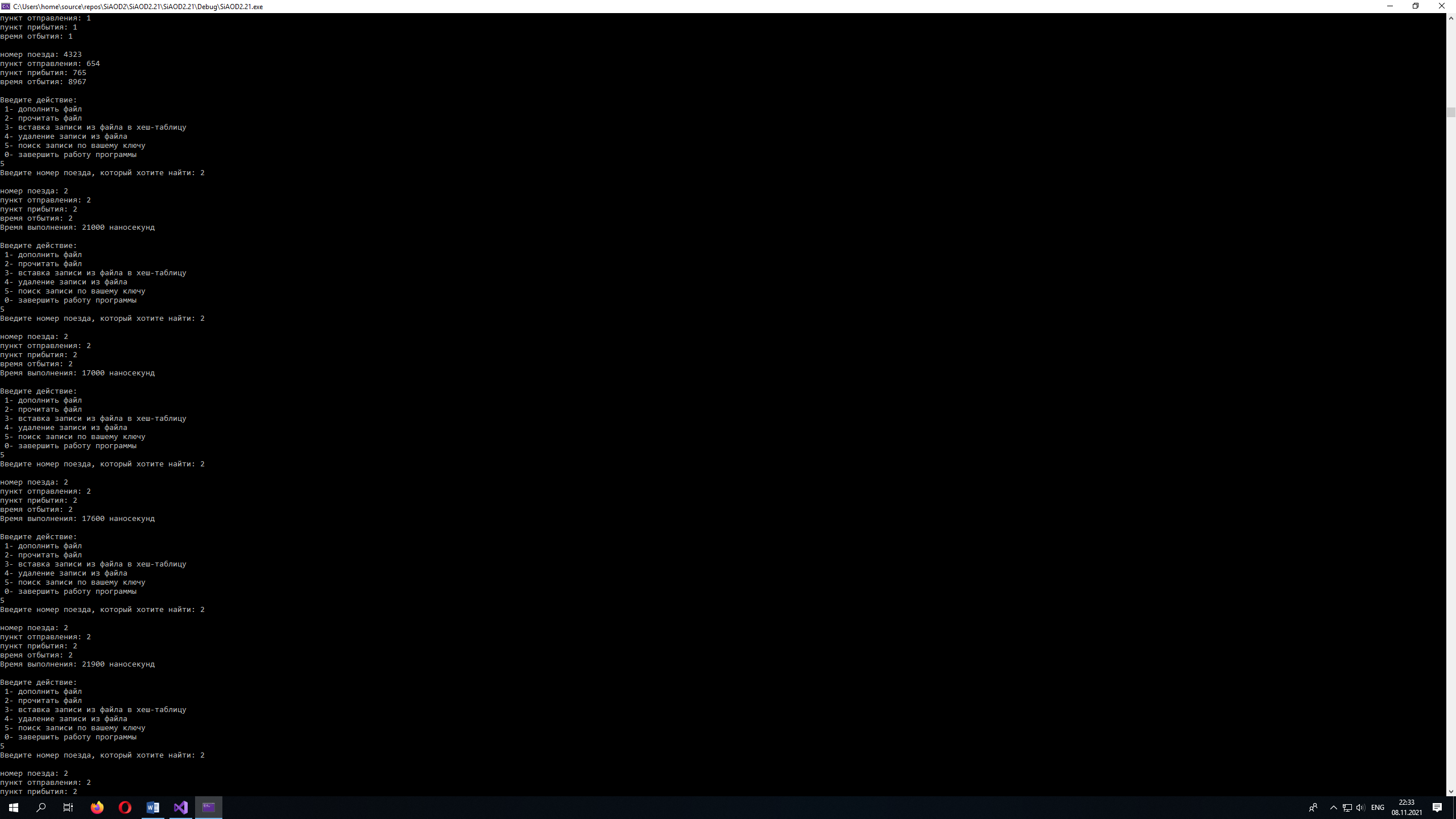


Рис. 14 Время поиска среднего элемента первого рабочего прогона

Время поиска последнего элемента первого рабочего прогона (Рис. 15)

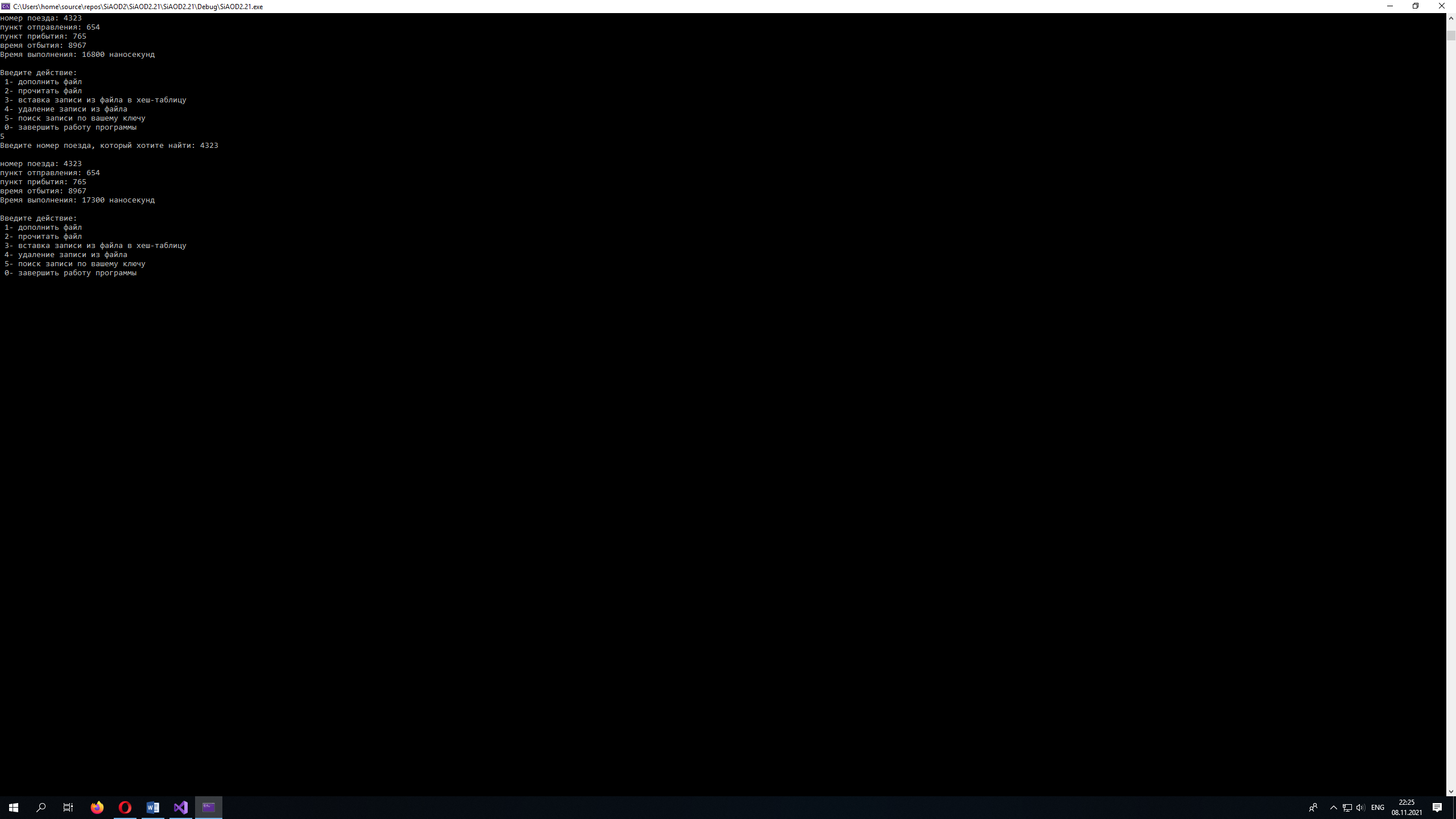


Рис. 15 Время поиска последнего элемента первого рабочего прогона

Время поиска несуществующего элемента первого рабочего прогона (Рис. 16)

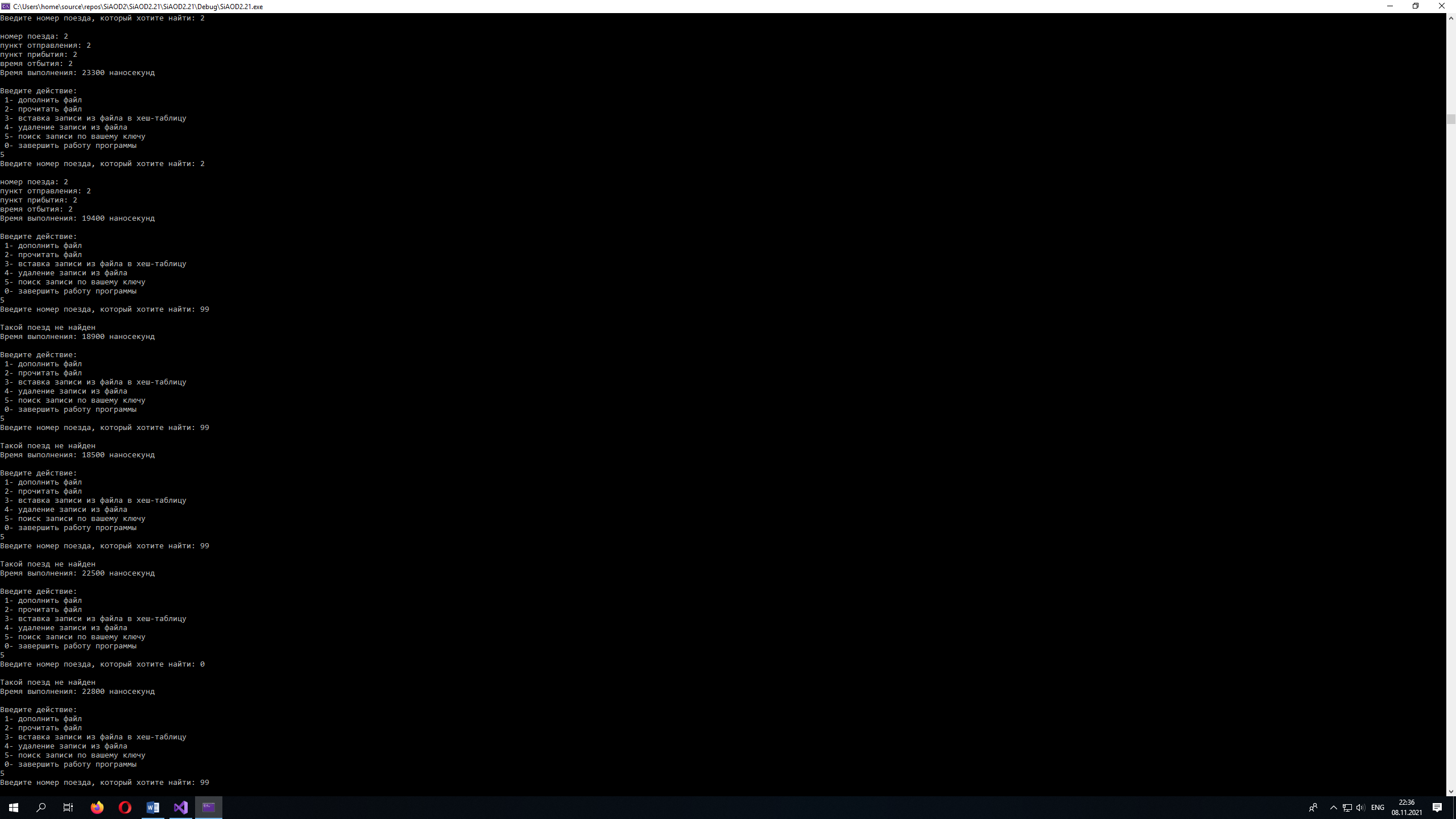


Рис. 16 Время поиска несуществующего элемента первого рабочего прогона

Хеш-таблица первого рабочего прогона (Рис. 17)

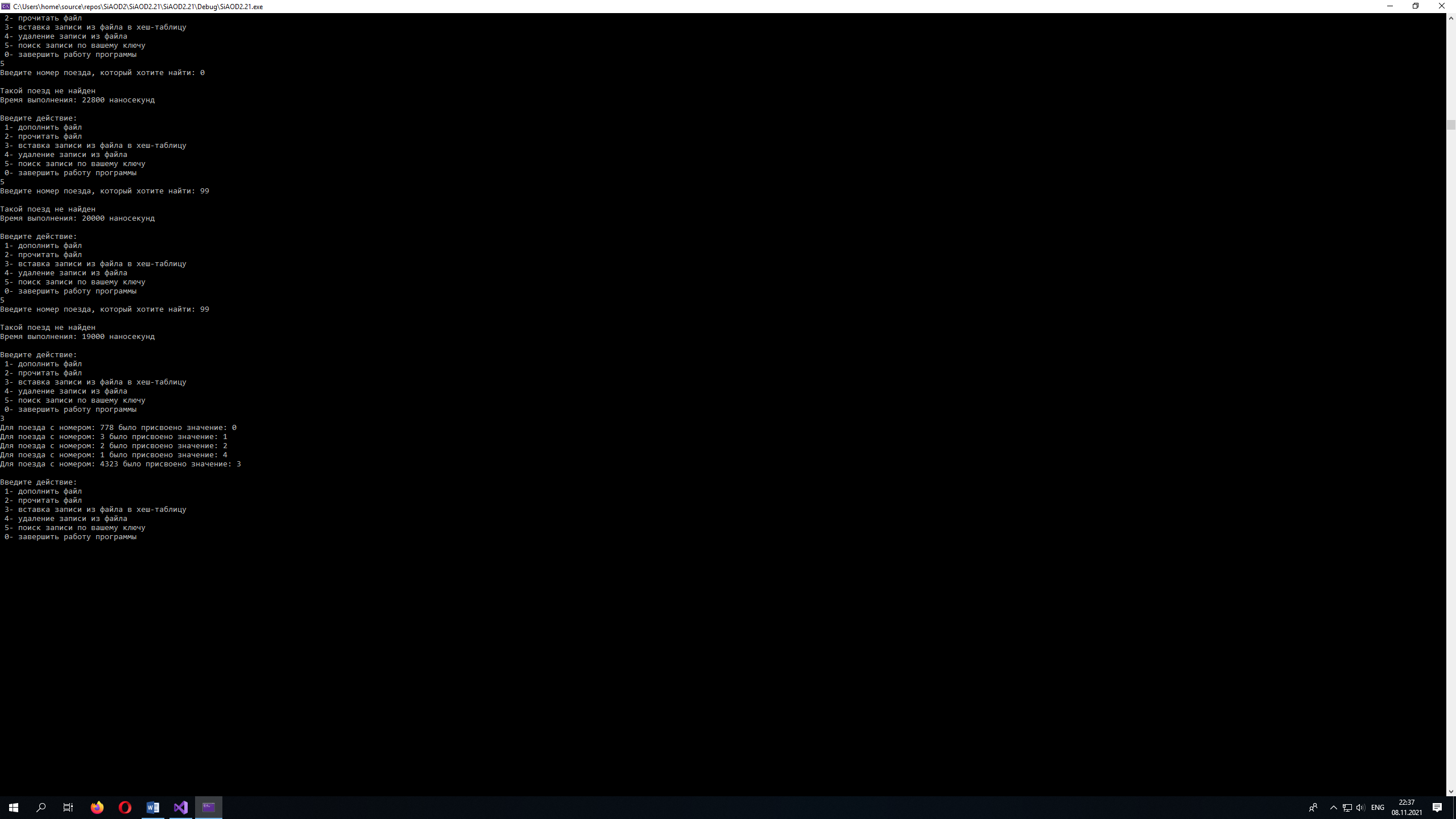


Рис. 17 Хеш-таблица первого рабочего прогона

Время поиска первого элемента второго рабочего прогона (Рис. 18)

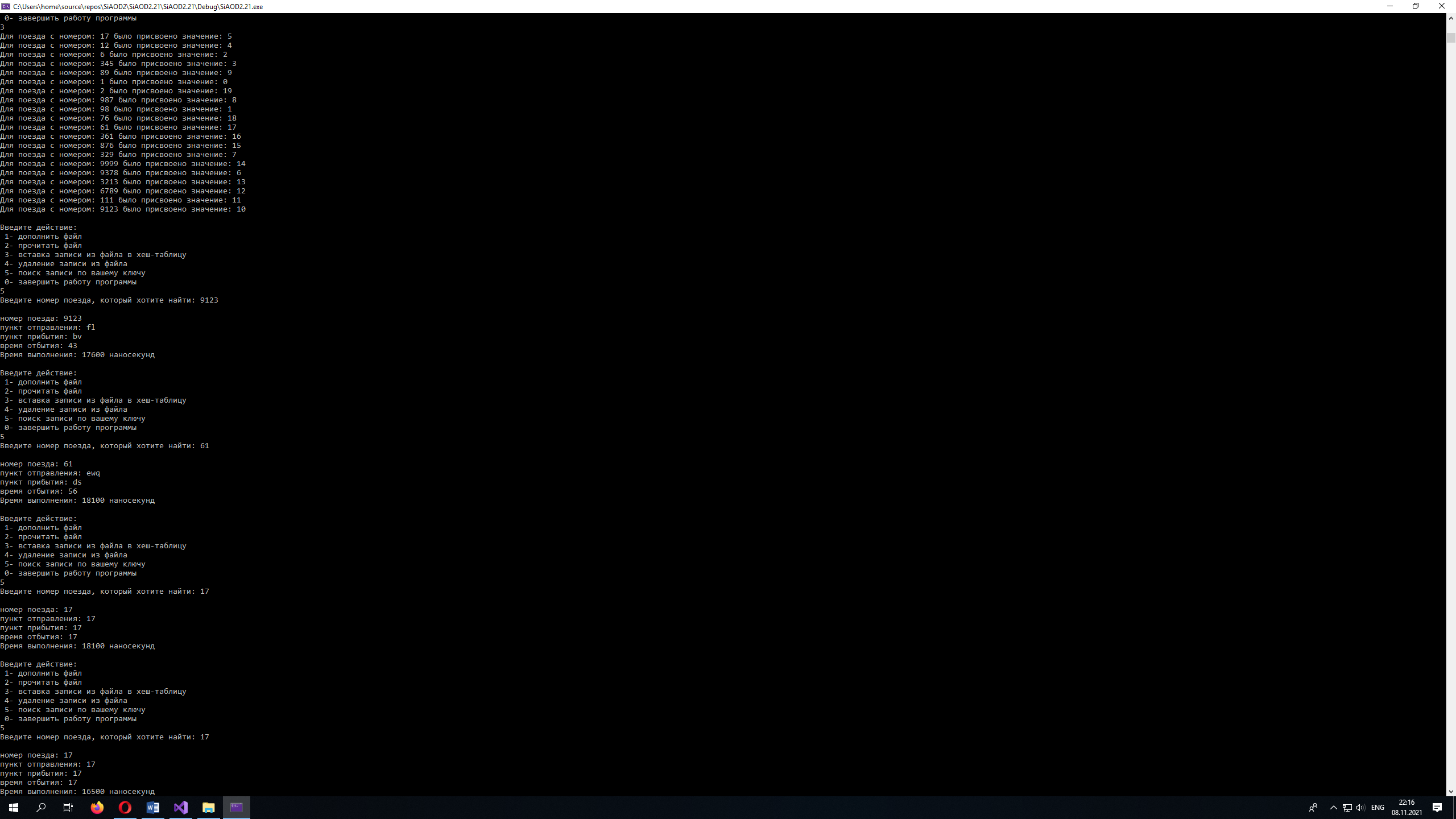


Рис. 18 Время поиска первого элемента второго рабочего прогона

Время поиска среднего элемента второго рабочего прогона (Рис. 19)

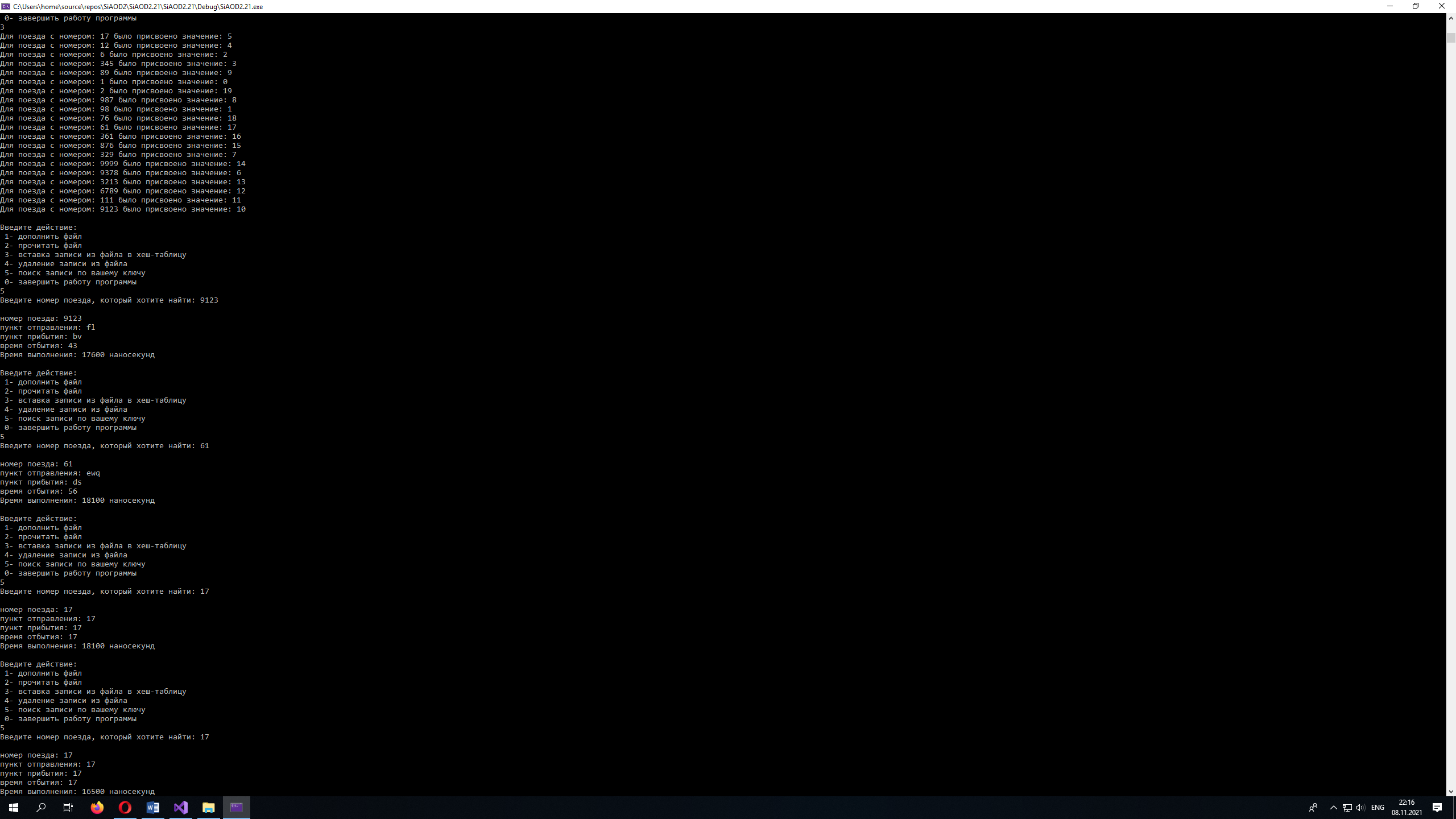


Рис. 19 Время поиска среднего элемента второго рабочего прогона

Время поиска последнего элемента второго рабочего прогона (Рис. 20)

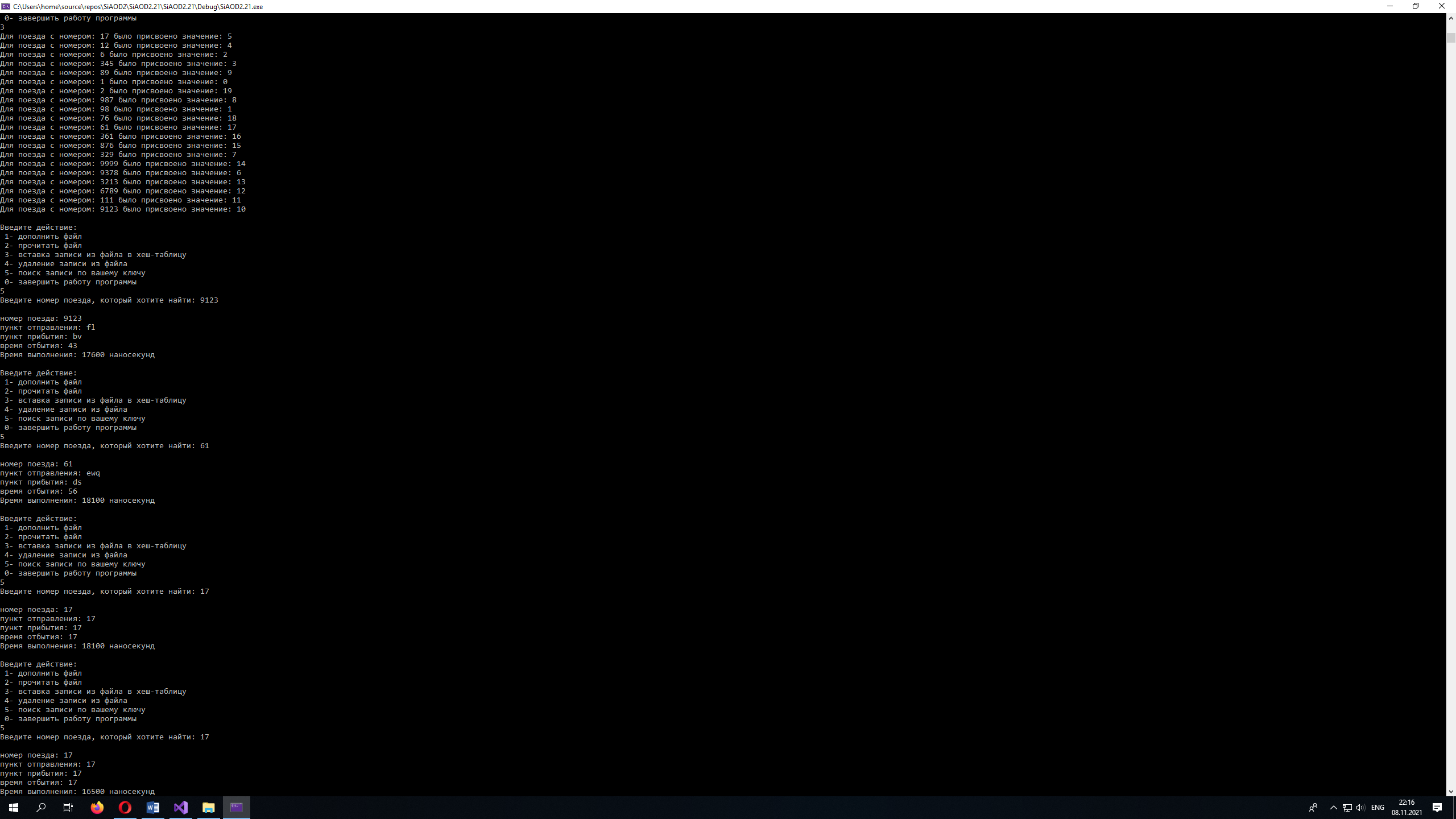


Рис. 20 Время поиска последнего элемента второго рабочего прогона

Время поиска несуществующего элемента второго рабочего прогона (Рис. 21)

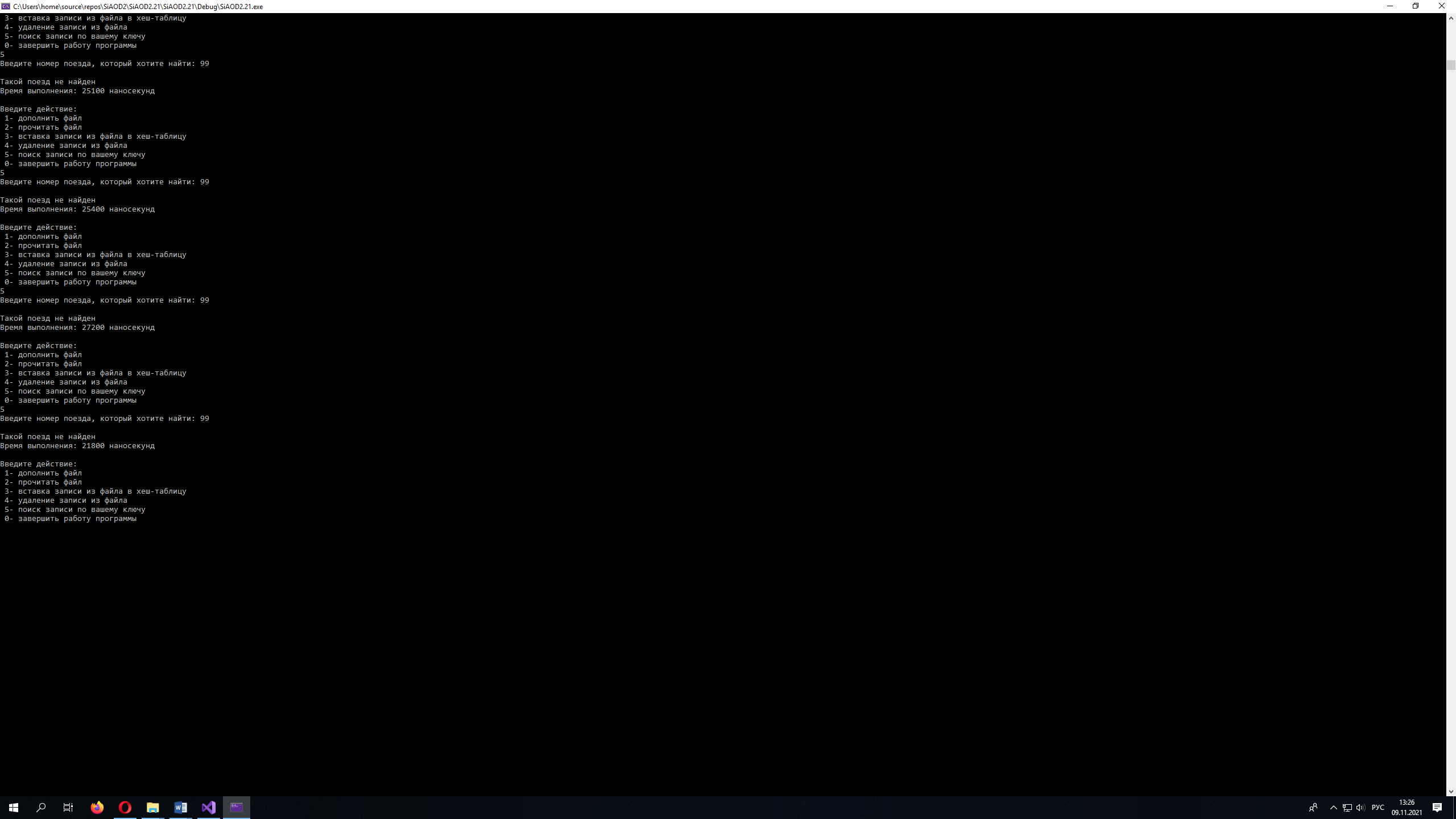


Рис. 21 Время поиска несуществующего элемента второго рабочего прогона

Хеш-таблица второго рабочего прогона (Рис. 22)

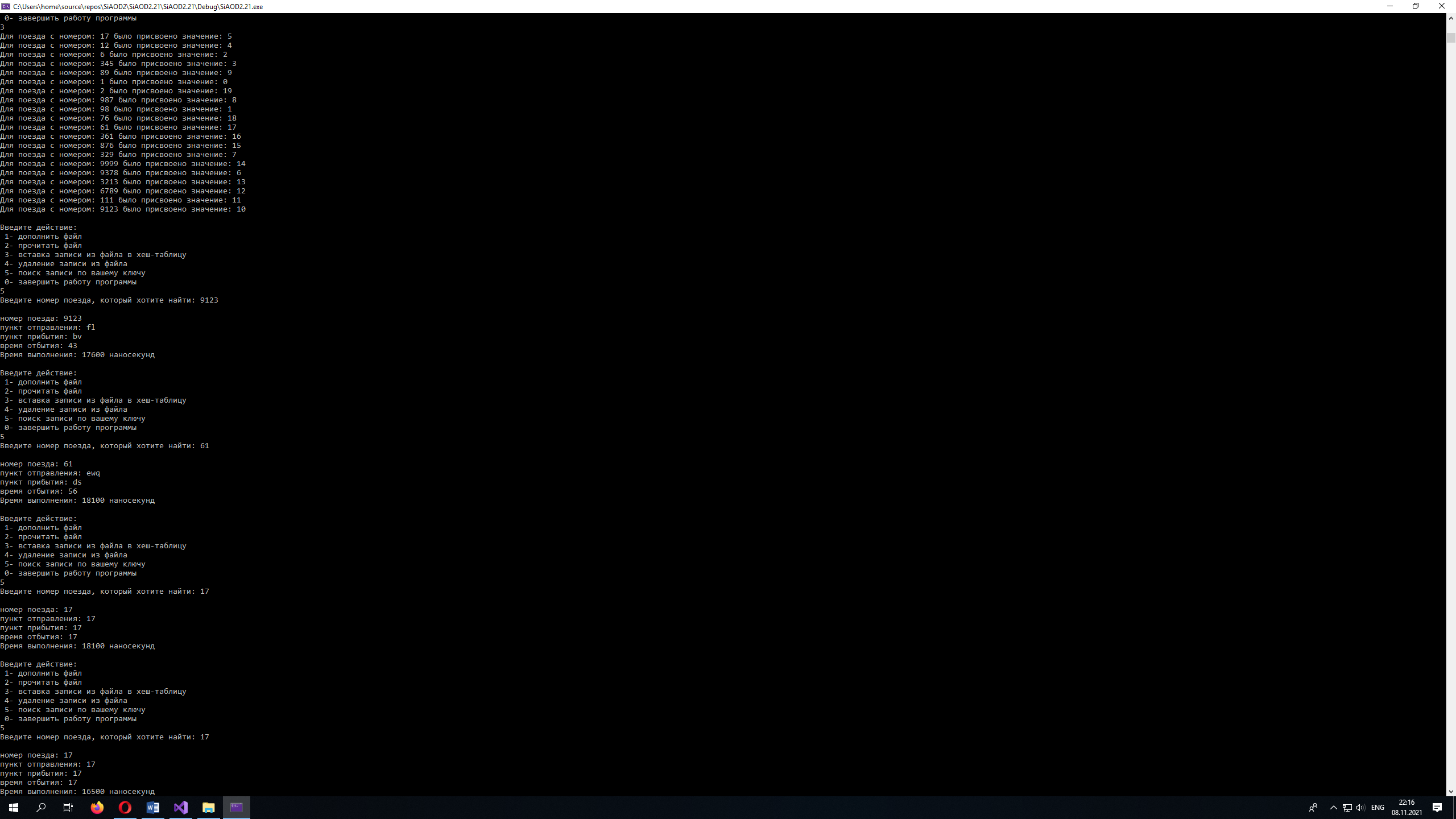


Рис. 22 Хеш-таблица второго рабочего прогона

Таблица 1. Сводная таблица результатов рабочих прогонов

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Количество элементов** | **Первый элемент, нс** | **Средний элемент, нс** | **Последний элемент, нс** | **Несуществующий элемент, нс** |
| 5 | 16400 | 17000 | 17300 | 18500 |
| 20 | 16500 | 18100 | 17600 | 25400 |

Из результатов выполнения программы видно:

* Зависимость от положения в файле имеется, т.к. чем ближе находится элемент к началу файла, тем ближе к зоне поиска он потом окажется в хеш-таблице.
* Но эта зависимость не является существенной, т.к. как мы видим из прогонов, во втором прогоне последний элемент (Рис. 20) находится быстрее, чем средний (Рис.19), это происходит из-за того, что средний элемент занял 17 позицию, а последний занял 10 позицию (Рис. 22), поэтому в большей степени на время поиска влияет позиция элемента в хеш-таблице.

Вопросы:

1. Что такое «первичный кластер» в таблице с открытым адресом?

Ответ: Это способ решения коллизии, в котором при возникновении коллизии используется постоянный шаг в 1 ячейку, таким образом, если первоначальный хеш имел значение x, то последующие будут увеличины на 1: x+1, x+2, x+3… Такой подход снижает производительность.

1. Как реализуется двойное хеширование?

Ответ: Двойное хеширование реализуется путём создания второй хеш-функции. При возникновении коллизии хеш, полученный в результате работы первой хеш-функции, проходит обработку второй хеш-функции. Если коллизия не была устранена, получившийся хеш проходит повторную обработку второй хеш-функцией и так до тех пор, пока коллизия не будет устранена.

# **Вывод**

В результате выполнения работы я:

1. Получила навыки разработки хеш-таблиц и их применения.
2. Получила навыки организации быстрого поиска данных.

# **Исходный код программы**

|  |
| --- |
| #include <iostream>  #include <cstdio>  #include <fstream>  #include <string>  #include <chrono>  using namespace std;  struct Reference  {  string numb= "";//ключ 1  string op = "";  string ip = "";  string time = "";  };  class HeshFun  {  public:  int n;  bool b = false;  Reference\* a;  void NewStruct()  {  DeleterA();  a = new Reference[n];  for (int j = 0; j < n; j++)  {  a[j] = { "", "", "", "" };  cout << a[j].numb;  }  b = true;  }  void DeleterA()  {  delete[] a;  }  int toHash(string key)  {  int number, hash;  number = stoi(key);  hash = number / 3;  if (n != 1)  hash = hash % n % (n / 2);//получения номера для хэша в первой части таблицы  else  hash = hash % n;  return hash;  }  int SecondHash(int oldhash)  {  int newhash;  if (n % 2 == 0)  {  newhash = oldhash + n-1 ;  newhash = newhash % n ;  }  else  {  newhash = oldhash + 2;  newhash = newhash % n;  }  return newhash;  }  ~HeshFun()  {  b = false;  DeleterA();  }  };  struct List//список  {  string numb="";//ключ 1  string op="";//ключ 2  string ip="";  string time="";  List\* pnext = NULL;  };  List\* Head = NULL;  List\* CreatList()//создание списка  {  List\* l = new List;  if (!Head)  {  Head = l;  l->pnext = nullptr;  }  else  {  l->pnext = Head;  Head = l;  }  return l;  }  void DeleteList()//очищаем список  {  if (Head != 0)  {  for (List\* l = Head; l->pnext;)  {  Head = l->pnext;  free(l);  l = Head;  }  free(Head);  Head = nullptr;  }  }  bool DeleteinFile(string x)//удаление записи в файле  {  bool b = false;  List\* l;  ifstream f("file.bin", ios::binary);  if (f.is\_open())  {  while (true)  {  l = CreatList();  if (!f.read(reinterpret\_cast<char\*>(&l->numb), sizeof(string)))  break;  f.read(reinterpret\_cast<char\*>(&l->op), sizeof(string));  f.read(reinterpret\_cast<char\*>(&l->ip), sizeof(string));  f.read(reinterpret\_cast<char\*>(&l->time), sizeof(string));  }  l = Head;  Head = l->pnext;  free(l);  }  f.close();  ofstream fo("file.bin", ios::binary | ofstream::trunc | ofstream::out);  for (List\* l = Head; l;)  {  if (l->numb != x)  {  fo.write(reinterpret\_cast<char\*>(&l->numb), sizeof(string));  fo.write(reinterpret\_cast<char\*>(&l->op), sizeof(string));  fo.write(reinterpret\_cast<char\*>(&l->ip), sizeof(string));  fo.write(reinterpret\_cast<char\*>(&l->time), sizeof(string));  }  else  b = true;  Head = l->pnext;  free(l);  l = Head;  }  free(Head);  Head = nullptr;  f.close();  return b;  }  void WriteFile(int n, Reference a[])//запись в файл элемента  {  fstream f("file.bin", ios::binary | ios::app);  if (!f.is\_open())  {  f.close();  fstream f("file.bin", ios::binary);//если файла не существует, то создаётся  for (int i = 0; i < n; i++)  {  f.write(reinterpret\_cast<char\*>(&a[i]), sizeof(Reference));  }  }  else  {  for (int i = 0; i < n; i++)  {  f.write(reinterpret\_cast<char\*>(&a[i]), sizeof(Reference));  }  }  f.close();  }  int ReadFile()//чтение из файла  {  int i = 0;  List\* l;  ifstream f("file.bin", ios::binary);  if (f.is\_open())  {  while (true)  {  i++;  l = CreatList();  if (!f.read(reinterpret\_cast<char\*>(&l->numb), sizeof(string)))  break;  f.read(reinterpret\_cast<char\*>(&l->op), sizeof(string));  f.read(reinterpret\_cast<char\*>(&l->ip), sizeof(string));  f.read(reinterpret\_cast<char\*>(&l->time), sizeof(string));  }  l = Head;  Head = l->pnext;  free(l);  }  f.close();  return i - 1;  }  Reference FindRef(string key, HeshFun \*H)  {  int newkey = H->toHash(key);  if (H->a[newkey].numb == key)  return H->a[newkey];  else  {  for (int i = 1; i < H->n; i++)  {  newkey = H->SecondHash(newkey);  if (H->a[newkey].numb == key)  return H->a[newkey];  }  }  return { "", "", "", "" };  }  int DoHashtable(HeshFun \*H)  {  int numberhash, max;  List\* l;  max = ReadFile();  H->n = max;  H->NewStruct();  l = Head;  while (l != nullptr)  {  numberhash = H->toHash(l->numb);  if (H->a[numberhash].numb != "")  {  while (H->a[numberhash].numb != "")  numberhash = H->SecondHash(numberhash);  }  H->a[numberhash].numb = l->numb;  H->a[numberhash].ip = l->ip;  H->a[numberhash].op = l->op;  H->a[numberhash].time = l->time;  cout << "Для поезда с номером: " << l->numb << " было присвоено значение: " << numberhash << "\n";  l = l->pnext;  }  DeleteList();  return max;  }  int main()  {  setlocale(LC\_ALL, "RUSSIAN");  char sw;  int a, max = 0, i = 0;  List\* l;  HeshFun H;  Reference m[100];  cout << "Хотите внести данные в файл?\n Y-да, N-нет\n";  cin >> sw;  if (sw == 'Y' || sw == 'y')  {  cout << "Для выхода введите в любом поле 0\n";  while (true)  {  cout << "\nВведите номер поезда: ";  cin >> m[i].numb;  if (m[i].numb == "0")  break;  cout << "Введите пункт отправления: ";  cin >> m[i].op;  if (m[i].op == "0")  break;  cout << "Введите пункт прибытия: ";  cin >> m[i].ip;  if (m[i].numb == "0")  break;  cout << "Введите время отправления: ";  cin >> m[i].time;  if (m[i].numb == "0")  break;  i++;  if (i >= 100)  {  cout << "\nдостигнут лимит записей, введите снова\n";  break;  }  }  WriteFile(i, m);  max = i;  for (int j = 0; j < i; j++)  m[j] = { "", "", "", "" };  i = 0;  }  res: //продолжение выбора задания  cout << "\nВведите действие:\n 1- дополнить файл\n 2- прочитать файл\n 3- вставка записи из файла в хеш-таблицу\n 4- удаление записи из файла\n 5- поиск записи по вашему ключу\n 0- завершить работу программы\n";  cin >> a;  switch (a)  {  case 1:  {  cout << "Для выхода введите в любом поле 0\n";  while (true)  {  cout << "\nВведите номер поезда: ";  cin >> m[i].numb;  if (m[i].numb == "0")  break;  cout << "Введите пункт отправления: ";  cin >> m[i].op;  if (m[i].op == "0")  break;  cout << "Введите пункт прибытия: ";  cin >> m[i].ip;  if (m[i].numb == "0")  break;  cout << "Введите время отправления: ";  cin >> m[i].time;  if (m[i].numb == "0")  break;  i++;  if (i >= 100)  {  cout << "\nдостигнут лимит записей, введите снова\n";  break;  }  }  WriteFile(i, m);  max += i;  for (int j = 0; j < i; j++)  m[j] = { "", "", "", "" };  i = 0;  goto res;  }  case 2:  {  max = ReadFile();  l = Head;  while (l != nullptr)  {  cout << "\nномер поезда: " << l->numb;  cout << "\nпункт отправления: " << l->op;  cout << "\nпункт прибытия: " << l->ip;  cout << "\nвремя отбытия: " << l->time<<"\n";  l = l->pnext;  }  DeleteList();  goto res;  }  case 3:  {  HasTable:  max = DoHashtable(&H);  goto res;  }  case 4:  {  bool f;  string deleteNumber;  cout << "\nВведите номер поезда, что необходимо удалить: ";  cin >> deleteNumber;  f=DeleteinFile(deleteNumber);  if (f)  {  cout << "Поезд удалён\n";  if (H.a != NULL && H.n!=1)  {  cout << "Новая хеш-таблица:\n";  goto HasTable;  }  else  {  H.DeleterA();  goto res;  }  }  else  {  cout << "Запись не найдена!\n";  goto res;  }  }  case 5:  {  string key="";  Reference ref = {"", "", "", ""};  cout << "Введите номер поезда, который хотите найти: ";  cin >> key;  if (H.b == false)  {  cout << "Вы не заполнили хеш-таблицу!\n";  goto res;  }  auto begin = chrono::steady\_clock::now();  ref = FindRef(key, &H);  auto end = chrono::steady\_clock::now();  auto time = chrono::duration\_cast<chrono::nanoseconds> (end - begin);  if (ref.numb != "")  {  cout << "\nномер поезда: " << ref.numb;  cout << "\nпункт отправления: " << ref.op;  cout << "\nпункт прибытия: " << ref.ip;  cout << "\nвремя отбытия: " << ref.time << "\n";  }  else  {  cout << "\nТакой поезд не найден\n";  }  cout << "Время выполнения: " << time.count() << " наносекунд\n";  goto res;  }  default:  break;  }  } |