|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | |  |  |  | | 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 № 5** | |
| **по дисциплине** | |
| **«**Структуры и алгоритмы обработки данных**»**  **Тема: «Сбалансированные деревья поиска (СДП) и их применение для поиска данных в файле.»** | |
|  | |
| 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 в разработки и реализации алгоритмов управления бинарным деревом поиска и сбалансированными бинарными деревьями поиска (АВЛ – деревьями);

# получить навыки в применении файловых потоков прямого доступа к данным файла;

# получить навыки в применении сбалансированного дерева поиска для прямого доступа к записям файла.

# Постановка задачи

1. Разработать класс управления файлом. Включить методы: создание двоичного файла записей фиксированной длины из заранее подготовленных данных в текстовом файле; поиск и вывод записи в файле с помощью линейного поиска; вывод записи по известному адресу в файле, добавление записи в файл. Структура записи файла определена индивидуальным вариантом задания.
2. Разработать класс «Бинарное дерево поиска». Структура информационной части узла дерева включает ключ и ссылку на запись в файле. Разработать основные методы: построение дерева по существующему файлу; добавление элемента в дерево; поиск по ключу в дереве; удаление элемента из дерева; вывод дерева в форме дерева.
3. Разработать класс «Сбалансированное дерево поиска». Структура информационной части узла дерева включает ключ и ссылку на запись в файле. Разработать основные методы: построение дерева по существующему файлу; добавление элемента в дерево; поиск по ключу в дереве; удаление элемента из дерева; вывод дерева в форме дерева.
4. Разработать приложение, демонстрирующее выполнение всех методов и поиск записей в файле с помощью БДП, СДП и линейного поиска.
5. Провести анализ времени выполнения поиска записей, находящихся в начале, середине и конце большого файла тремя способами.
6. Определить среднее число выполненных поворотов (число поворотов на общее число вставленных ключей) при включении ключей в СДП при его формировании из двоичного файла.
7. Выполнить тестирование.
8. Составить отчет, отобразив в нем описание выполнения всех этапов разработки, тестирования и код всей программы со скриншотами результатов тестирования.

Таблица 1. Вариант задания

|  |  |  |
| --- | --- | --- |
| № | Тип балансированного  дерева поиска (СДП) | Структура записи (ключ – подчеркнутое поле) |
| 8 | Косое | Книга: ISBN – двенадцатизначное число, Автор, Название |

# Решение

Бинарное дерево поиска ーдвоичное дерево, для которого выполняются следующие дополнительные условия (свойства дерева поиска):

* оба поддерева — левое и правое — являются двоичными деревьями поиска;
* у всех узлов левого поддерева произвольного узла X значения ключей данных меньше либо равны, нежели значение ключа данных самого узла X;
* у всех узлов правого поддерева произвольного узла X значения ключей данных больше, нежели значение ключа данных самого узла X.

Бинарное сбалансирвоанное дерево поиска ー Бинраное дерево поиска, которе с помощью левых и прпавых поворотов приводится к виду у которого количество левых и правых узлов каждого поддерева равны.

Косое дерево ーдвоичное дерево поиска, в котором поддерживается свойство сбалансированности. Это дерево принадлежит классу «саморегулирующихся деревьев», которые поддерживают необходимый баланс ветвления дерева, чтобы обеспечить выполнение операций поиска, добавления и удаления за логарифмическое время от числа хранимых элементов. Это реализуется без использования каких-либо дополнительных полей в узлах дерева (как, например, в Красно-чёрных деревьях или АВЛ-деревьях, где в вершинах хранится, соответственно, цвет вершины и глубина поддерева). Вместо этого «расширяющие операции» (splay operation), частью которых являются вращения, выполняются при каждом обращении к дереву.

Учётная стоимость (англ.) в расчёте на одну операцию с деревом составляет O(log n).

Расширяющееся дерево придумали Роберт Тарьян и Даниель Слейтор в 1983 году.

1. Разработать класс управления файлом. Включить методы: создание двоичного файла записей фиксированной длины из заранее подготовленных данных в текстовом файле; поиск и вывод записи в файле с помощью линейного поиска; вывод записи по известному адресу в файле, добавление записи в файл. Структура записи файла определена индивидуальным вариантом задания.

|  |
| --- |
| #include "BinFile.h"  int BinFile::txt\_to\_bin(string txt\_file, string bin\_file)  {  ifstream inp;  inp.open(txt\_file, ios::binary || ios::app);  ofstream outp;  outp.open(bin\_file, ios::binary || ios::app);  if (inp.good() && outp.good())  {  Book tt;  while (true)  {  inp >> tt.ISBN;  if (!inp.eof())  {  inp >> tt.fam >> tt.name;  outp.write((char\*)&tt, sizeof(Book));  }  else break;  }  inp.close();  outp.close();  return 1;  }  else  {  inp.close();  outp.close();  return 0;  }  }  Book BinFile::get\_class(string filename, int position)  {  Book book;  ifstream bin\_file(filename, ios::binary);  bin\_file.seekg((position) \* sizeof(book));  bin\_file.read((char\*)&book, sizeof(Book));  bin\_file.close();  return book;  }  void BinFile::AddToBin(ofstream& bin\_file, Book book)  {  bin\_file.seekp(0, ios::end);  bin\_file.write((char\*)&book, sizeof(Book));  }  Book BinFile::linear\_search(string bin\_file, long long int key)  {  ifstream inp;  inp.open(bin\_file, ios::binary || ios::app);  Book tt;  tt.ISBN = 0;  while (true)  {  inp.read((char\*)&tt, sizeof(tt));  if (!inp.eof())  {  if (tt.ISBN == key)  {  return tt;  }  }  else  {  break;  }  }  inp.close();  return tt;  }  long long int\* BinFile::getISBN(string txt\_file)  {  ifstream inp;  inp.open(txt\_file, ios::binary|| ios::app);  Book tt;  long long int ISBN[10000] = {0};  int i = 0;  while (true)  {  inp.read((char\*)&tt, sizeof(tt));  if (!inp.eof())  {  ISBN[i] = tt.ISBN;  i++;  }  else  {  break;  }  }  inp.close();  return ISBN;  } |

1. Разработать класс «Бинарное дерево поиска». Структура информационной части узла дерева включает ключ и ссылку на запись в файле. Разработать основные методы: построение дерева по существующему файлу; добавление элемента в дерево; поиск по ключу в дереве; удаление элемента из дерева; вывод дерева в форме дерева.

|  |
| --- |
| #include "BST.h"  bool node::exists(node \* root, long long int a)  {  if (root == nullptr)  {  return false;  }  if (root->key == a)  {  return true;  }  if (a < root->key)  {  return exists(root->left, a);  }  return exists(root->right, a);  }  int node::search(node \* root, long long int a)  {  if (root == nullptr)  {  return -1;  }  if (root->key == a)  {  return root->value;  }  if (a < root->key)  {  return search(root->left, a);  }  return search(root->right, a);  }  node \* node::insert(node \* root, long long int a, int b)  {  if (exists(root, a))  {  return root;  }  if (root == nullptr)  {  return new node(a, b);  }  if (a < root->key)  {  root->left = insert(root->left, a, b);  }  if (a > root->key)  {  root->right = insert(root->right, a, b);  }  return root;  }  node \* node::minValueNode(class node \*node)  {  class node \*current = node;  while (current && current->left != NULL)  {  current = current->left;  }  return current;  }  node \* node::deleteNode(node \* root, long long int key)  {  if (root == NULL)  {  return root;  }  if (key < root->key)  {  root->left = deleteNode(root->left, key);  }  else if (key > root->key)  {  root->right = deleteNode(root->right, key);  }  else  {  if (root->left == NULL)  {  class node \*temp = root->right;  free(root);  return temp;  }  else if (root->right == NULL)  {  class node \*temp = root->left;  free(root);  return temp;  }  class node \*temp = minValueNode(root->right);  root->key = temp->key;  root->value = temp->value;  root->right = deleteNode(root->right, temp->key);  }  return root;  }  void node::printBT(const string& prefix, const node\* node, bool isLeft, bool key)  {  if (key)  {  if (node != nullptr)  {  std::cout << prefix;  std::cout << (isLeft ? "|-----------" : "L-----------");  std::cout << node->key << endl;  printBT(prefix + (isLeft ? "| " : " "), node->right, true,1);  printBT(prefix + (isLeft ? "| " : " "), node->left, false,1);  }  }  else  {  if (node != nullptr)  {  std::cout << prefix;  std::cout << (isLeft ? "|--" : "L--");  std::cout << node->value << endl;  printBT(prefix + (isLeft ? "| " : " "), node->right, true,0);  printBT(prefix + (isLeft ? "| " : " "), node->left, false,0);  }  }  } |

1. Разработать класс «Сбалансированное дерево поиска». Структура информационной части узла дерева включает ключ и ссылку на запись в файле. Разработать основные методы: построение дерева по существующему файлу; добавление элемента в дерево; поиск по ключу в дереве; удаление элемента из дерева; вывод дерева в форме дерева.

|  |
| --- |
| #include "SplayTree.h"  #include "BST.h"  double counter = 0;  nodesplay\* nodesplay::newNode(long long int key)  {  nodesplay\* Node = new nodesplay();  Node->key = key;  Node->left = Node->right = NULL;  return (Node);  }  nodesplay \*nodesplay::rightRotate(nodesplay \*x)  {  nodesplay \*y = x->left;  x->left = y->right;  y->right = x;  counter++;  return y;  }  nodesplay \*nodesplay::leftRotate(nodesplay \*x)  {  nodesplay \*y = x->right;  x->right = y->left;  y->left = x;  counter++;  return y;  }  nodesplay \*nodesplay::splay(nodesplay \*root, long long int key)  {  if (root == NULL || root->key == key)  return root;  if (root->key > key)  {  if (root->left == NULL)  {  return root;  }  // Zig-Zig (Левый-левый)  if (root->left->key > key)  {  // Сначала рекурсивно поднимаем  // ключ как корень left-left  root->left->left = splay(root->left->left, key);  // Первый разворот для root,  // второй разворот выполняется после else  root = rightRotate(root);  }  else if (root->left->key < key) // Zig-Zag (Левый-правый)  {  // Сначала рекурсивно поднимаем  // ключ как корень left-right  root->left->right = splay(root->left->right, key);  // Выполняем первый разворот для root->left  if (root->left->right != NULL)  root->left = leftRotate(root->left);  }  // Выполняем второй разворот для корня  return (root->left == NULL) ? root : rightRotate(root);  }  else // Ключ находится в правом поддереве  {  // Ключа нет в дереве, мы закончили  if (root->right == NULL)  {  return root;  }  // Zag-Zig (Правый-левый)  if (root->right->key > key)  {  // Поднять ключ как корень right-left  root->right->left = splay(root->right->left, key);  // Выполняем первый поворот для root->right  if (root->right->left != NULL)  root->right = rightRotate(root->right);  }  else if (root->right->key < key)// Zag-Zag (Правый-правый)  {  // Поднимаем ключ как корень  // right-right и выполняем первый разворот  root->right->right = splay(root->right->right, key);  root = leftRotate(root);  }  // Выполняем второй разворот для root  return (root->right == NULL) ? root : leftRotate(root);  }  }  nodesplay \*nodesplay::searchsplay(nodesplay \*root, long long int key)  {  return splay(root, key);  }  void nodesplay::preOrder(nodesplay \*root)  {  if (root != NULL)  {  cout << root->key << " ";  preOrder(root->left);  preOrder(root->right);  }  }  nodesplay \*nodesplay::insertsplay(nodesplay \*root, long long int key, int value, bool depth)  {  if (root == nullptr)  {  root = newNode(key);  root->value = value;  }  else if (key < root->key)  {  if (root->left == nullptr)  {  root->left = newNode(key);  root->left->value = value;  if (!depth)  {  root = splay(root, key);  }  }  else  {  root->left = insertsplay(root->left, key, value, 1);  root = splay(root, key);  }  }  else  {  if (root->right == nullptr)  {  root->right = newNode(key);  root->right->value = value;  if (!depth)  {  root = splay(root, key);  }  }  else  {  root->right = insertsplay(root->right, key, value, 1);  root = splay(root, key);  }  }  return root;  }  nodesplay \*nodesplay::tree\_max\_key(nodesplay \*root)  {  nodesplay \*curr = root;  while (curr->right != nullptr)  {  curr = curr->right;  }  return curr;  }  nodesplay \*nodesplay::merge(nodesplay \*tree1, nodesplay \*tree2)  {  tree1 = splay(tree1, tree\_max\_key(tree1)->key);  tree1->right = tree2;  return tree1;  }  nodesplay \*nodesplay::removesplay(nodesplay \*root, long long int key)  {  root = splay(root, key);  class nodesplay \*tree1 = NULL;  class nodesplay \*tree2 = NULL;  tree1 = root->left;  tree2 = root->right;  root = merge(tree1, tree2);  return root;  }  void nodesplay::printsplayBT(const string& prefix, nodesplay\* node, bool isLeft, bool key)  {  if (key)  {  if (node != nullptr)  {  std::cout << prefix;  std::cout << (isLeft ? "|-----------" : "L-----------");  std::cout << node->key << endl;  printsplayBT(prefix + (isLeft ? "| " : " "), node->right, true, 1);  printsplayBT(prefix + (isLeft ? "| " : " "), node->left, false, 1);  }  }  else  {  if (node != nullptr)  {  std::cout << prefix;  std::cout << (isLeft ? "|--" : "L--");  std::cout << node->value << endl;  printsplayBT(prefix + (isLeft ? "| " : " "), node->right, true, 0);  printsplayBT(prefix + (isLeft ? "| " : " "), node->left, false, 0);  }  }  } |

1. Разработать приложение, демонстрирующее выполнение всех методов и поиск записей в файле с помощью БДП, СДП и линейного поиска.

Разработанное приложение представлено на рисунке 1.

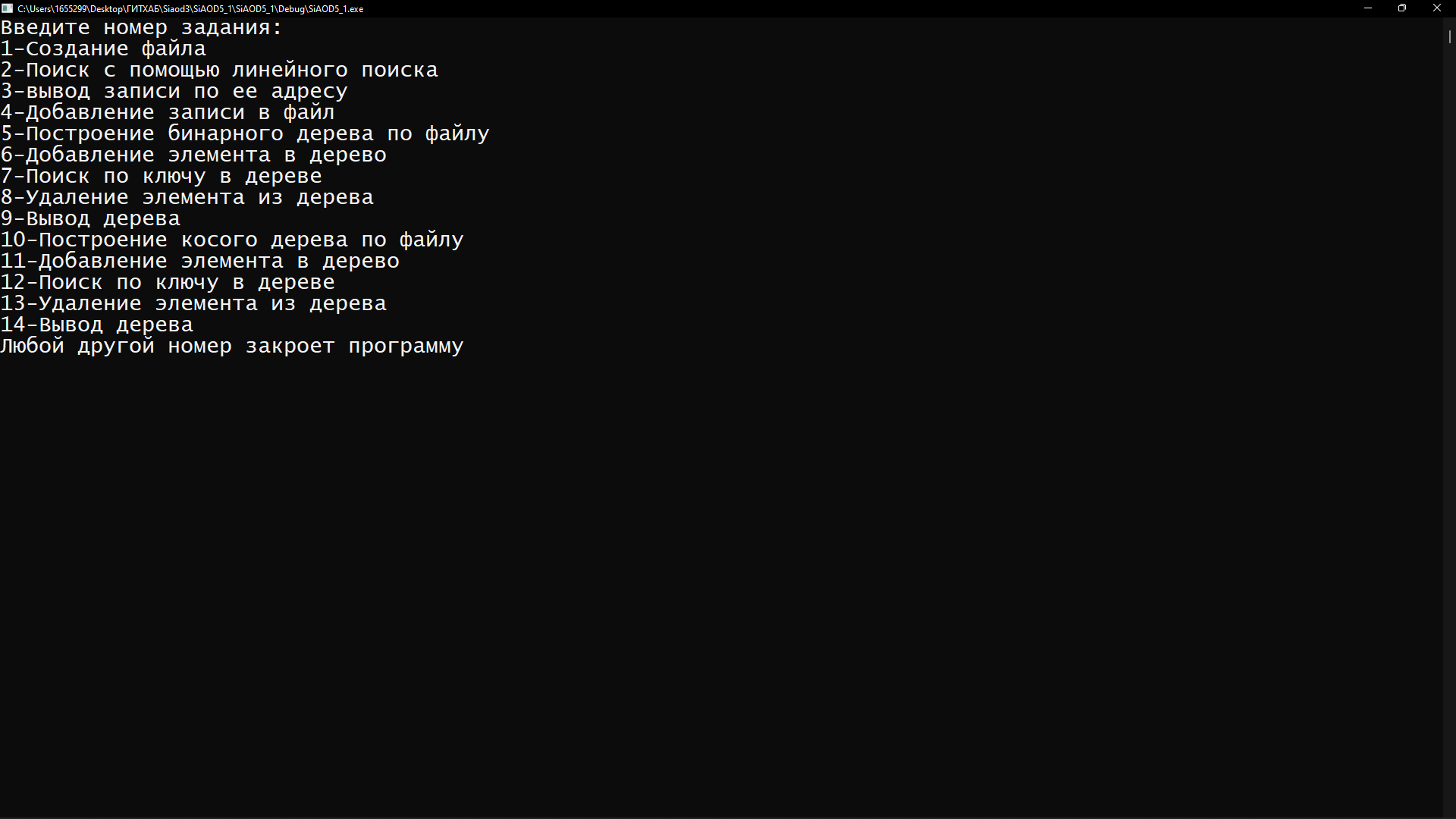


Рисунок 1. Разработанное приложение

1. Провести анализ времени выполнения поиска записей, находящихся в начале, середине и конце большого файла тремя способами.

Анализ проведен. Результат будет представлен в пункте 4 данного отчета.

1. Определить среднее число выполненных поворотов (число поворотов на общее число вставленных ключей) при включении ключей в СДП при его формировании из двоичного файла.

Количесвто поворотов определено. Результат будет представлен в пункте 4 данного отчета.

1. Выполнить тестирование.

Составить отчет, отобразив в нем описание выполнения всех этапов разработки, тестирования и код всей программы со скриншотами результатов тестирования.

Тестирование проведено. Результат представлен в пункте 4 данного отчета.

# Тестирование

Тестирование приложения.

Длинна ключа была уменьшена в связи с тем, что ключ длинной 12 знаков вызывает переполнение памяти.

Создание бинарного файла представлено на рисунке 2.

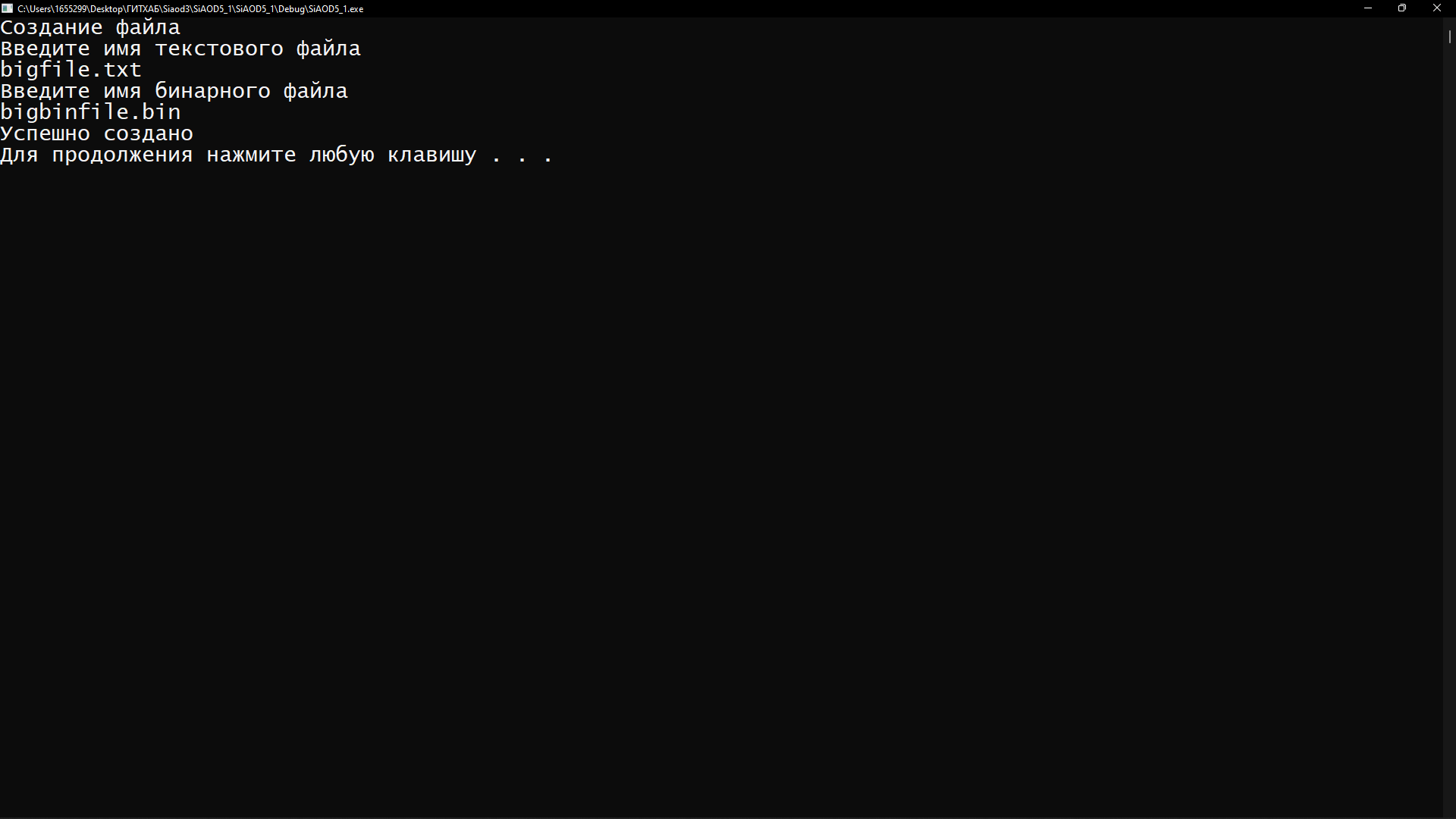


Рисунок 2. Создание бинарного файла

1 строка большого файла 674 hvfzogyjdt inbmmegswd.

500 строка большого файла 368 unkvgfeМdz gnpuwydtiu.

1000 строка большого файла 540 onuoubbslw nvtveoМsye.

Линейный поиск в начале файла представлен на рисунке 3.

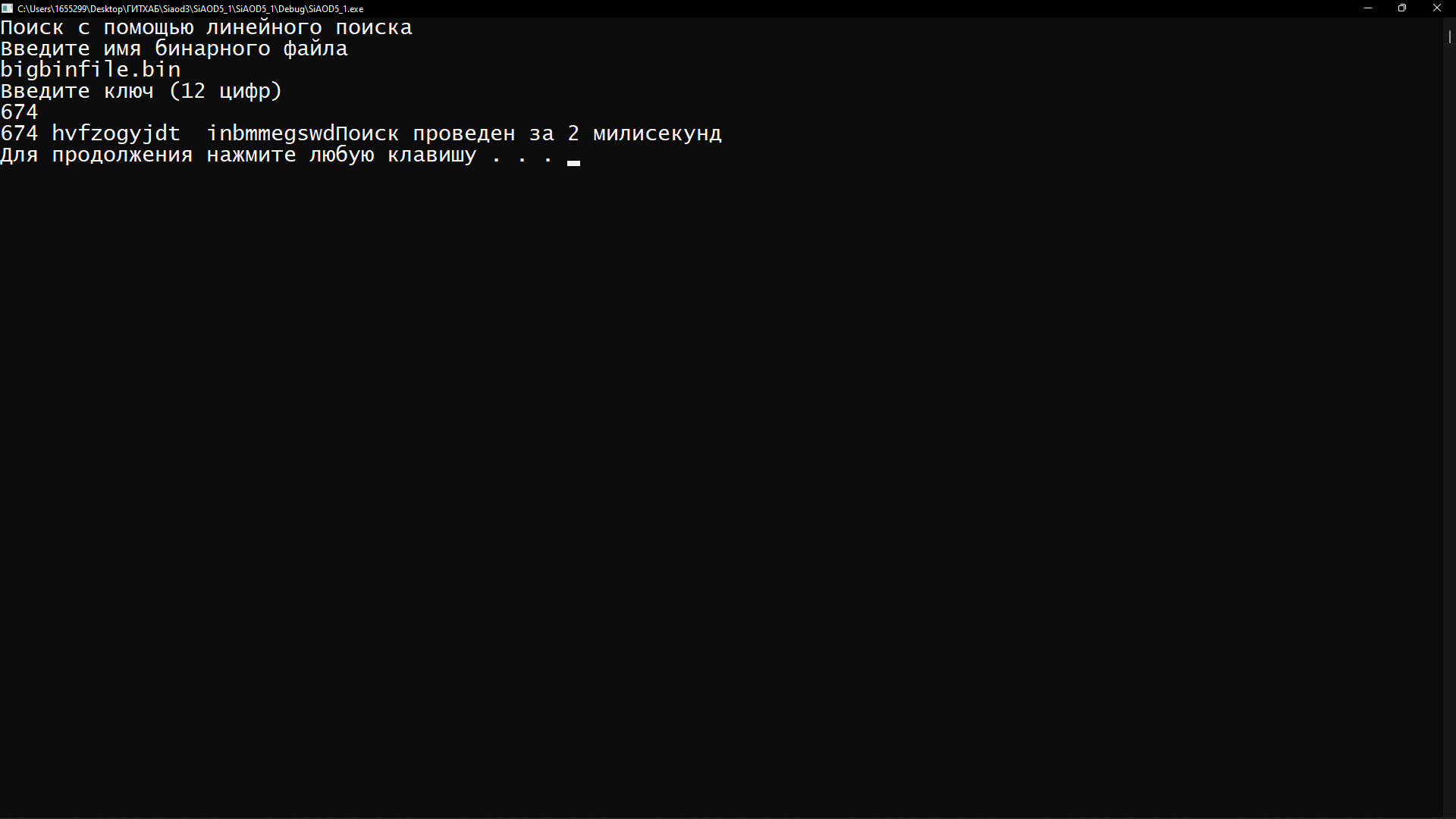


Рисунок 3. Создание бинарного файла

Линейный поиск в середине файла представлен на рисунке 4.

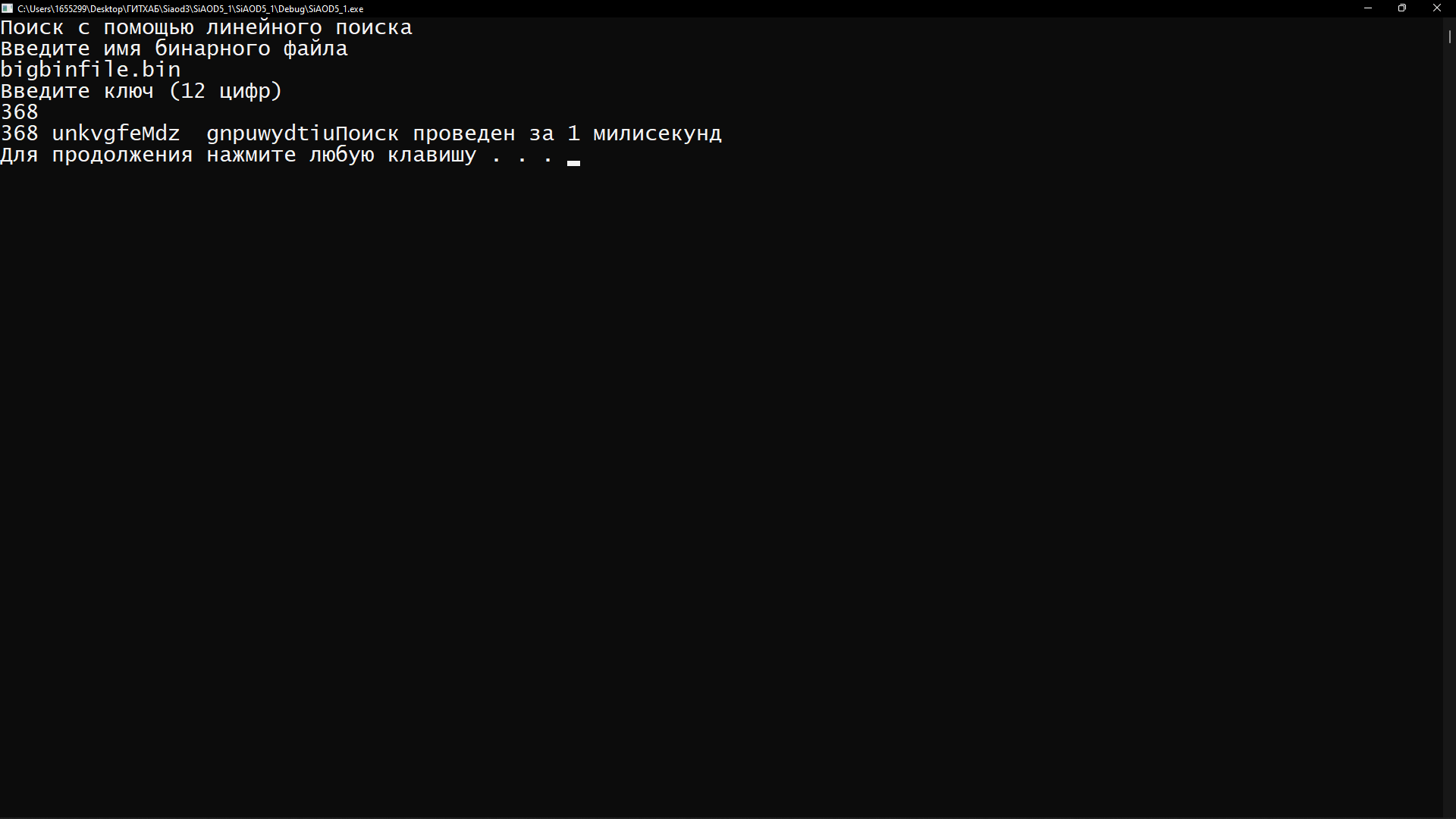


Рисунок 4. Линейный поиск в середине файла

Линейный поиск в конце файла представлен на рисунке 5.

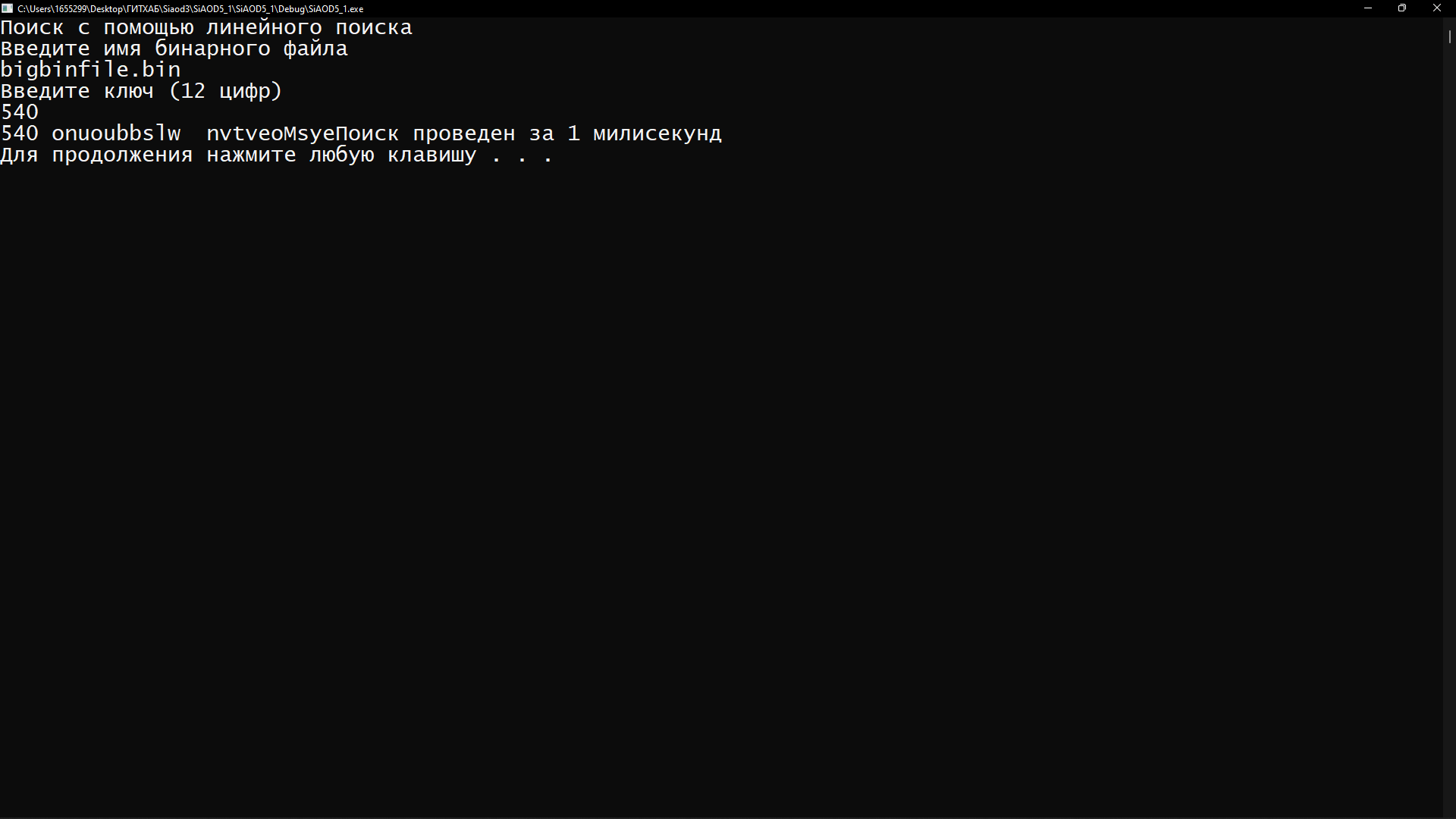


Рисунок 5. Линейный поиск в конце файла

Поиск по позиции в файле представлен ан рисунке 6.

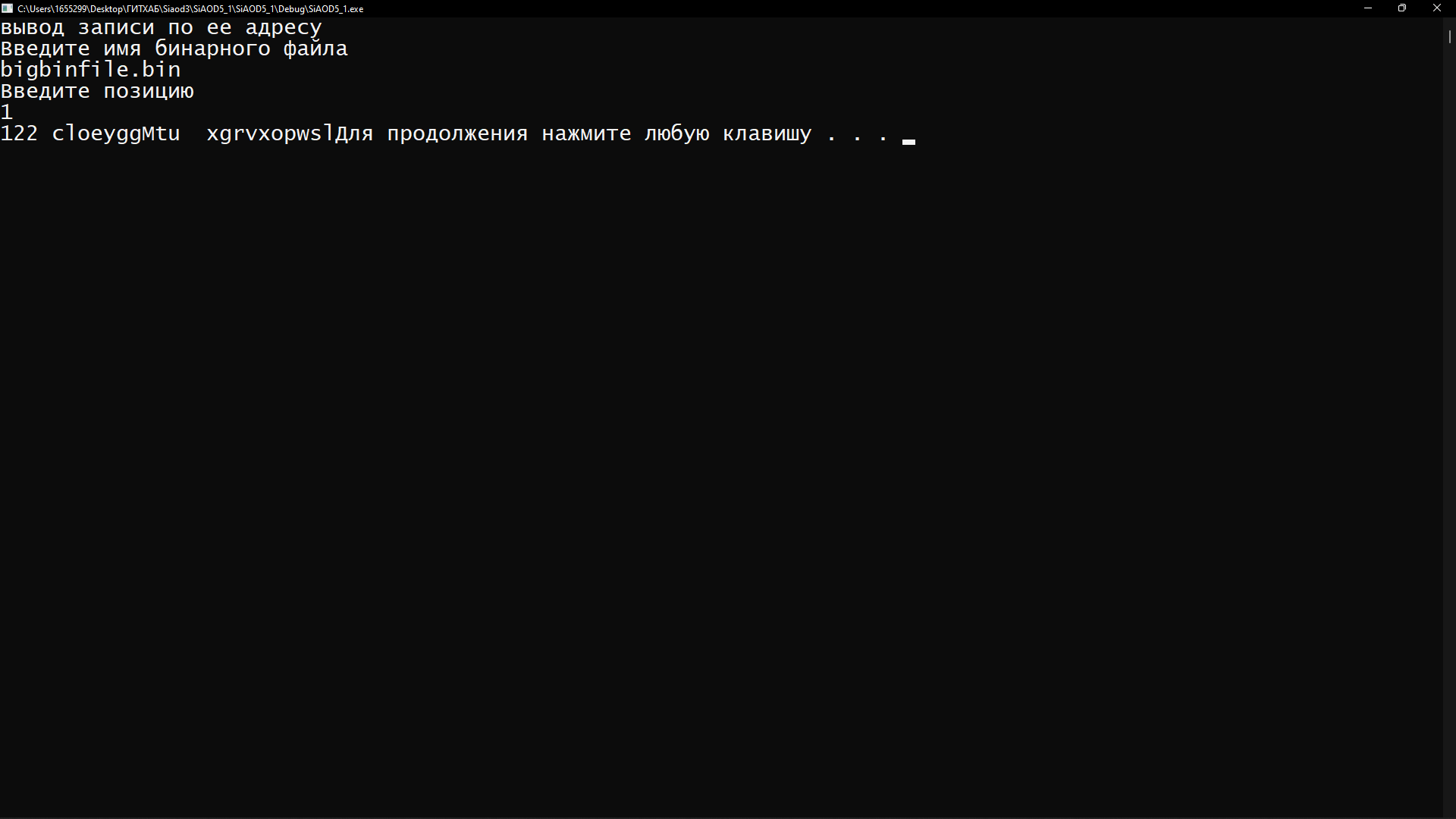


Рисунок 6. Поиск по позиции в файле

Вывод бинарного дерева поиска представлен на рисунке 7.

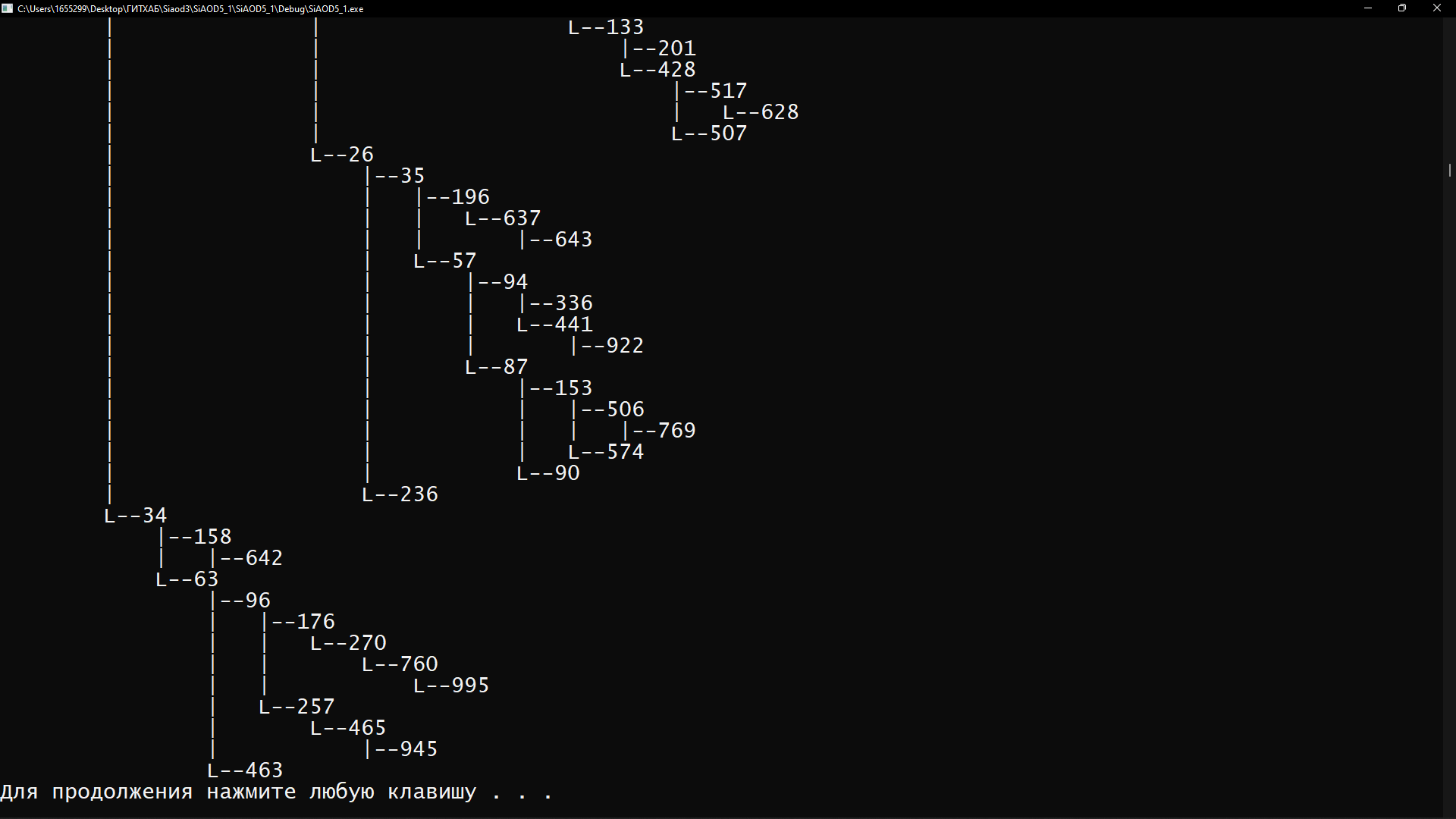


Рисунок 7. Вывод бинарного дерева поиска

Поиск в бинарном дереве поиска в начале файла представлен на рисунке 8.

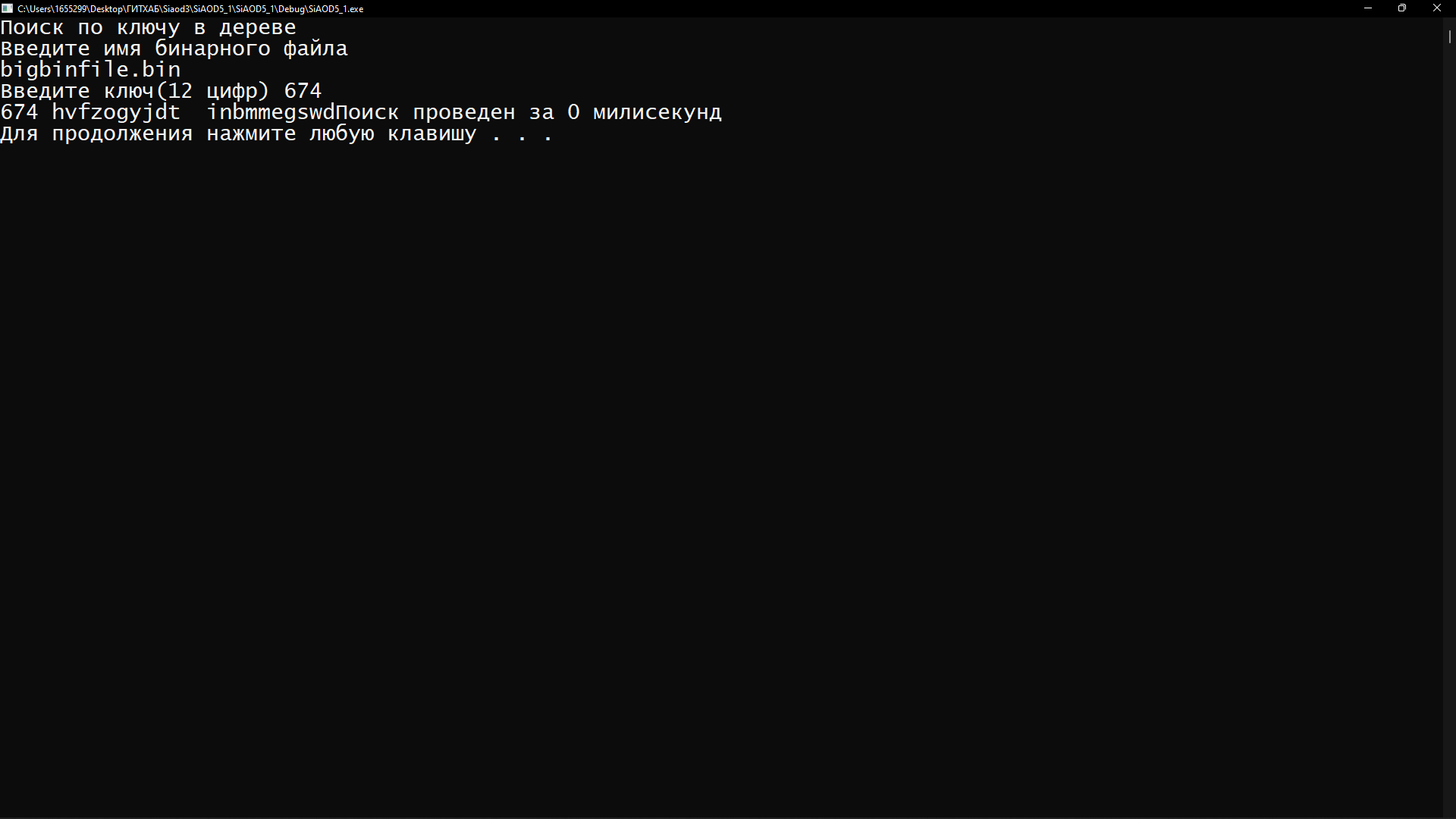


Рисунок 8. Поиск в бинарном дереве поиска в начале файла

Поиск в бинарном дереве поиска в середине файла представлен на рисунке 9.

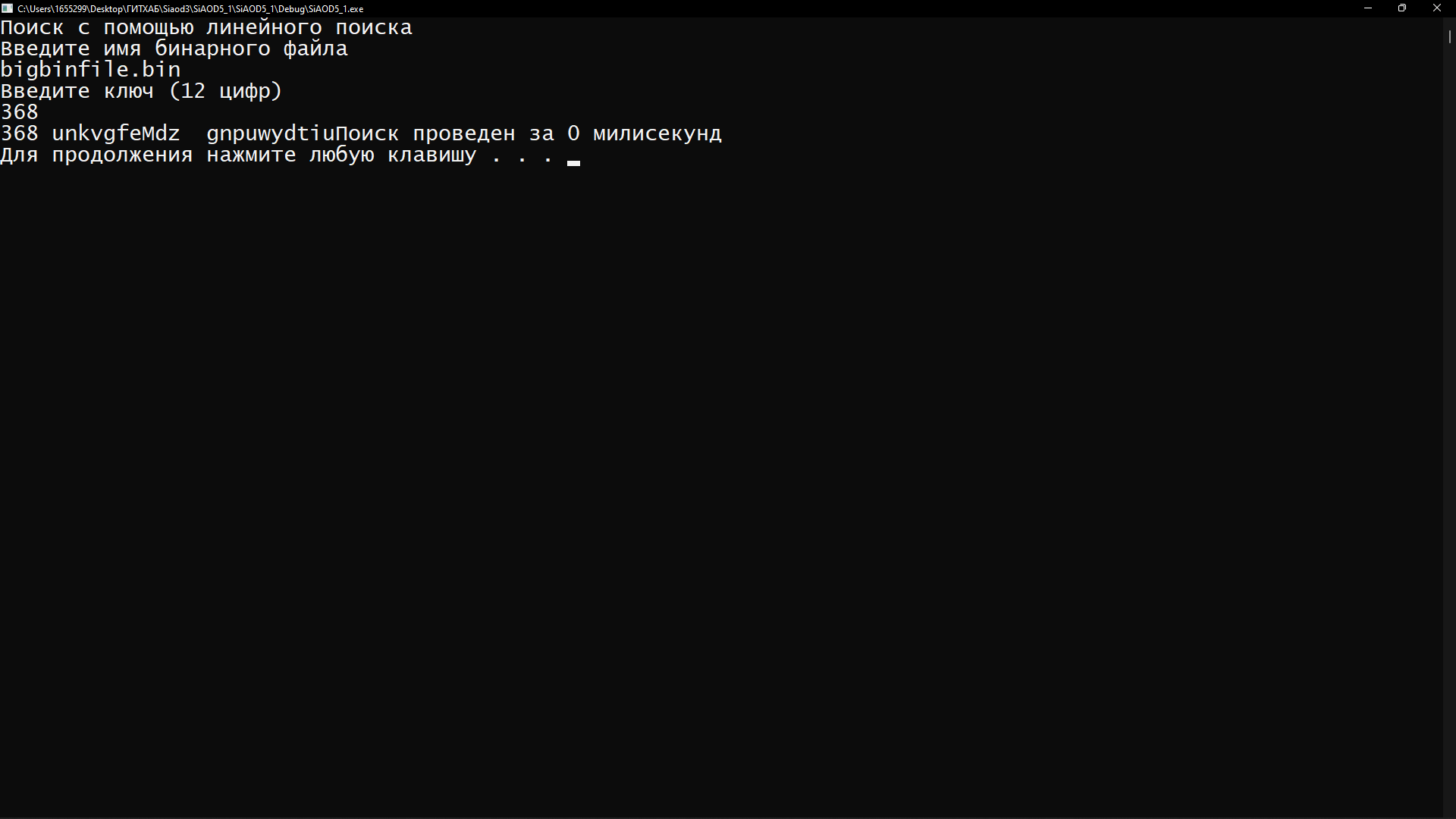


Рисунок 9. Поиск в бинарном дереве поиска в середине файла

Поиск в бинарном дереве поиска в конце файла представлен на рисунке 10.

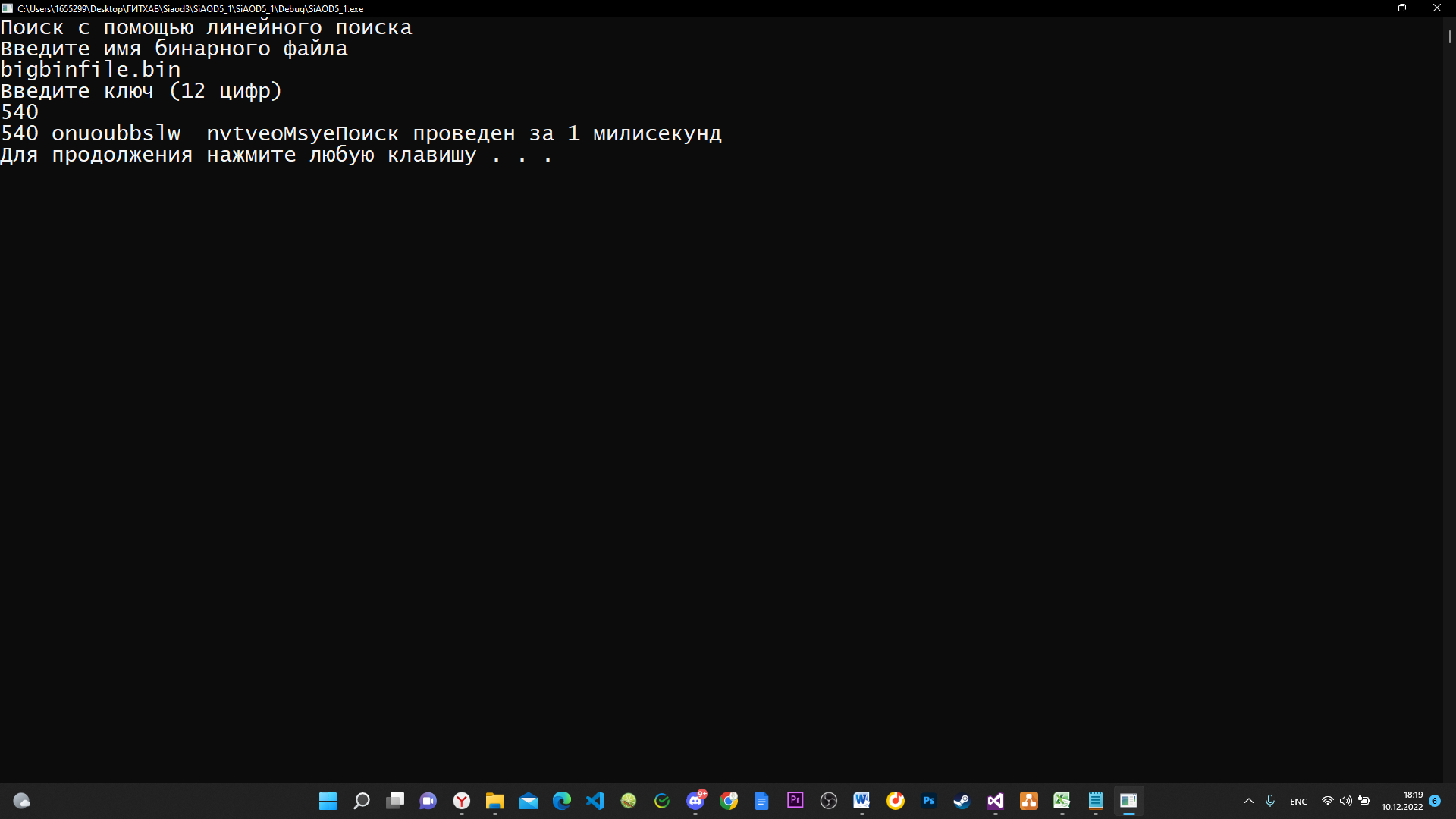


Рисунок 10. Поиск в бинарном дереве поиска в конце файла

Удаление элемента из бинраного дерева поиска представлено на рисунке 11.

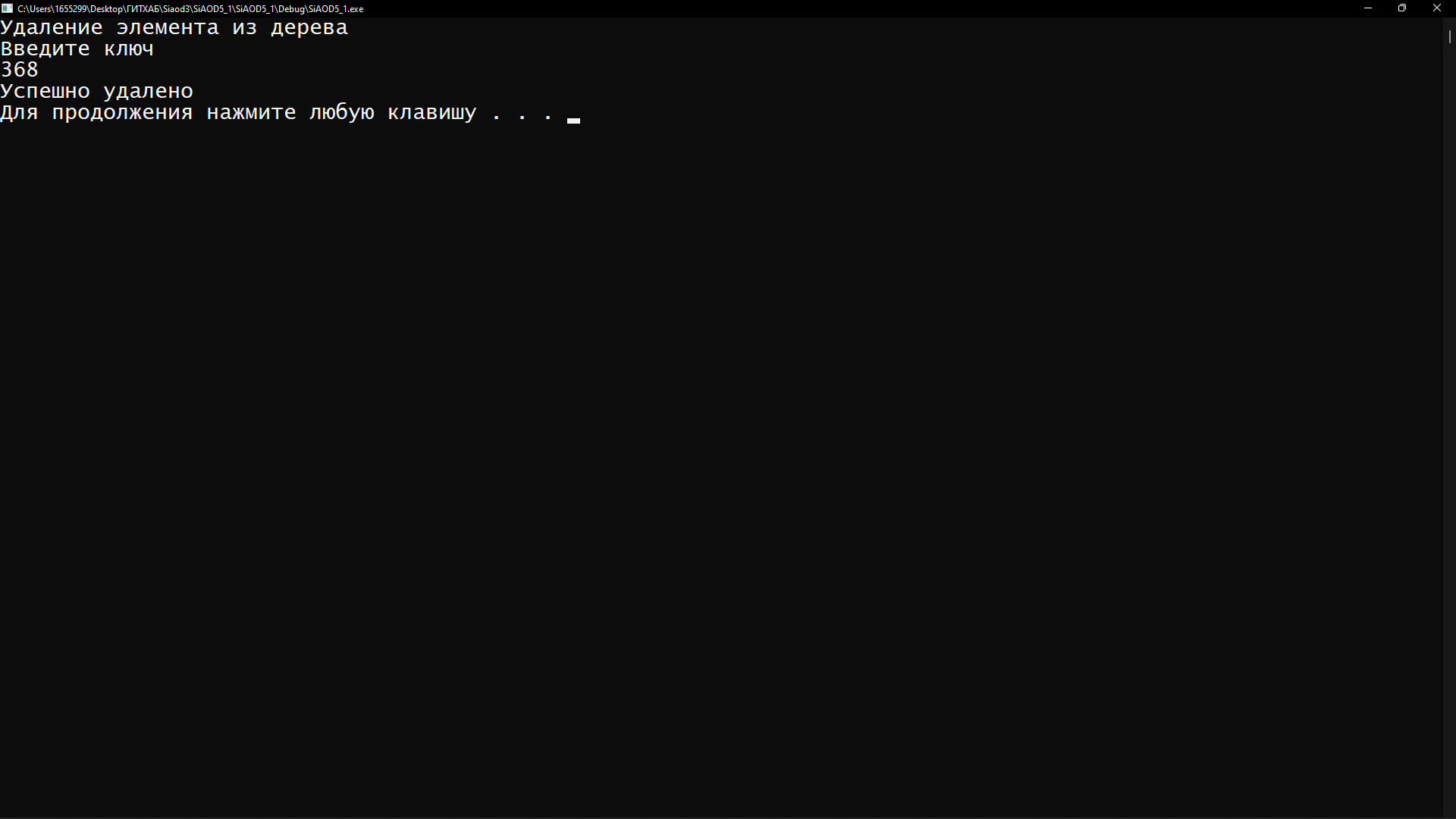


Рисунок 11. Удаление элемента из бинраного дерева поиска представлено

Поиск в бинарном дереве поиска в начале файла представлен на рисунке 12.

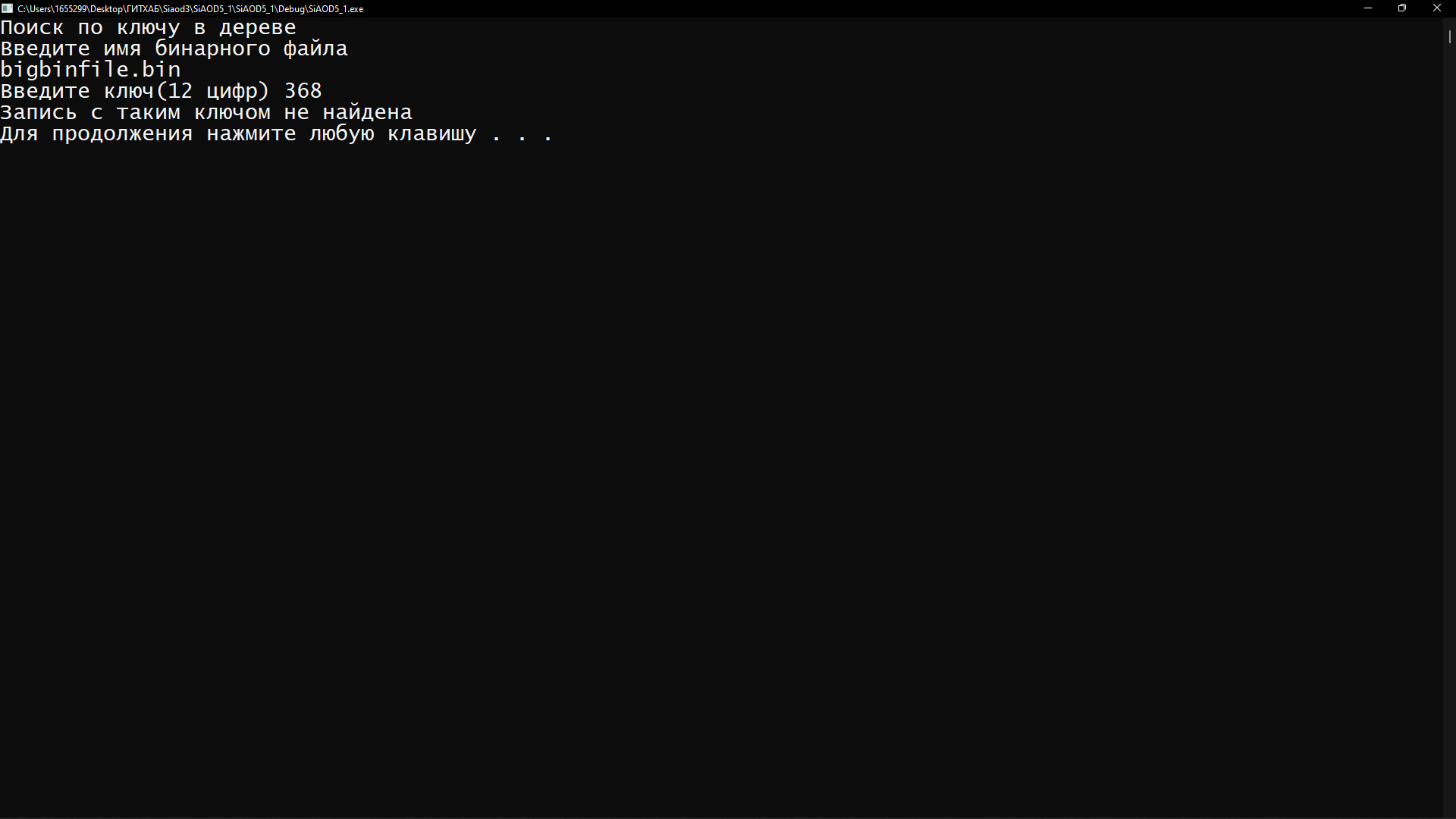


Рисунок 12. Поиск в бинарном дереве поиска в начале файла

Построение косого дерева представлено на рисунке 13.



Рисунок 13. Построение косого дерева представлено

Вывод косого дерева представлен на рисунке 14.

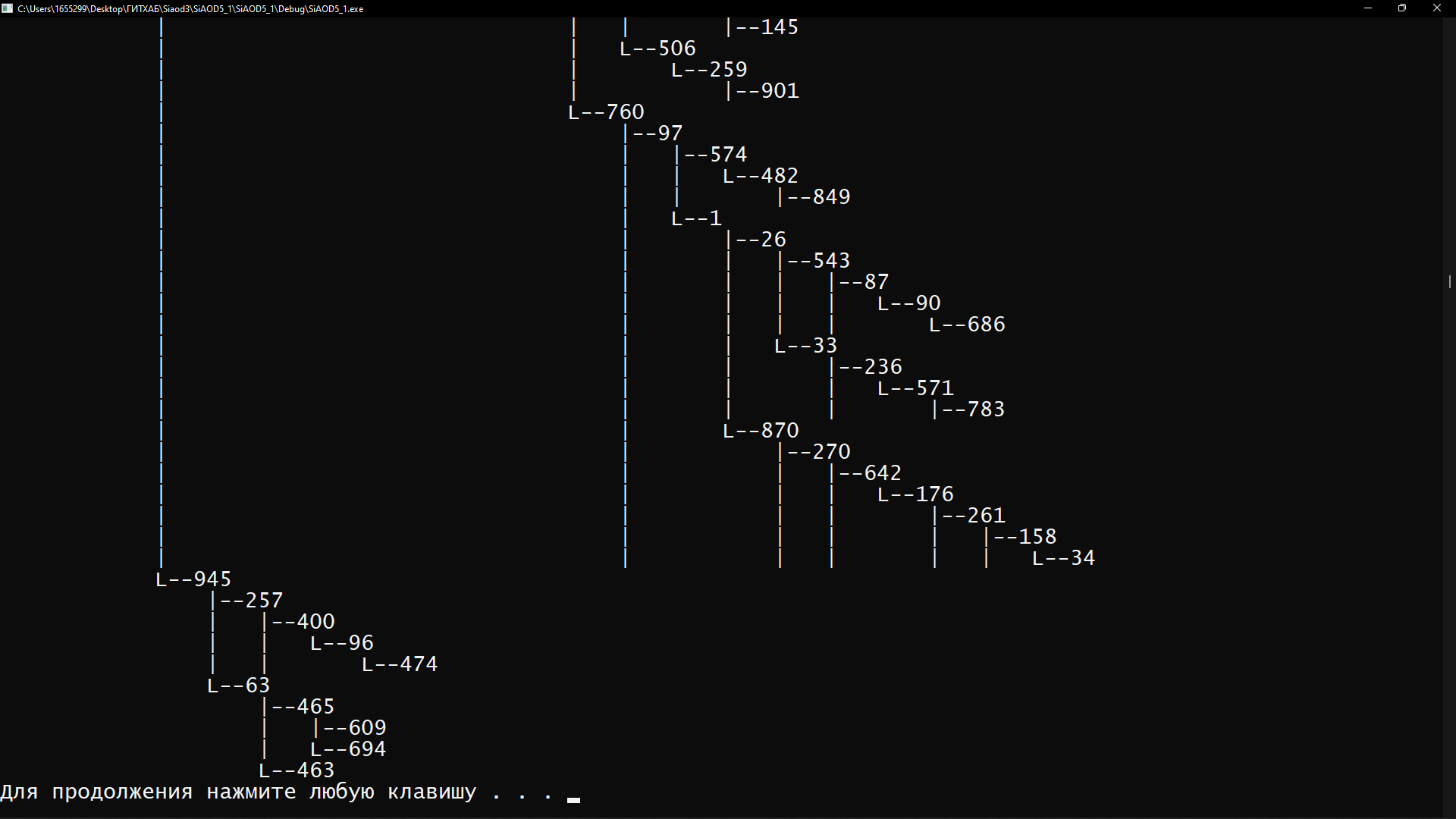


Рисунок 14. Вывод косого дерева

Поиск в косом дереве поиска в начале файла представлен на рисунке 15.

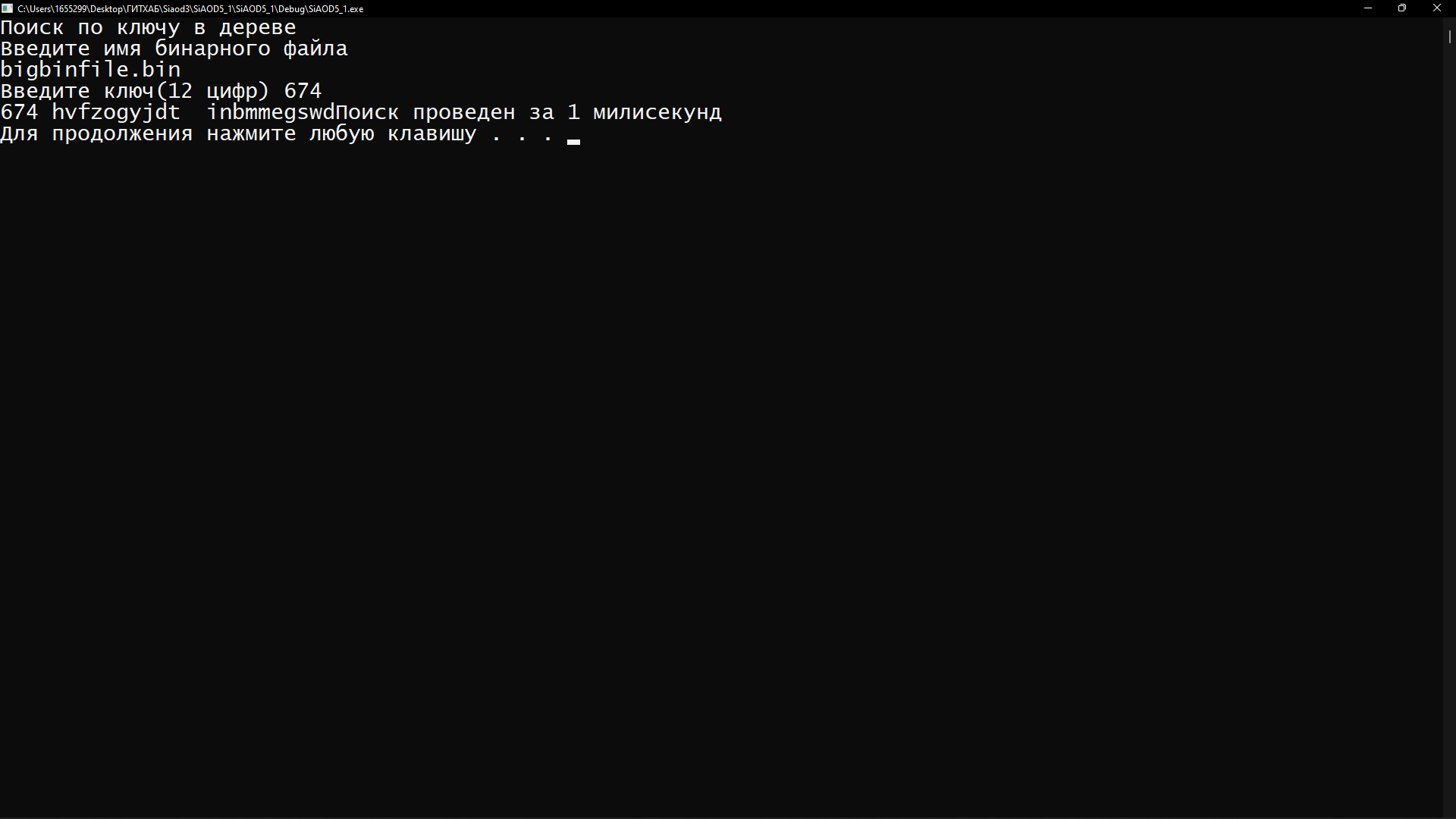


Рисунок 15. Поиск в косом дереве поиска в начале файла

Поиск в косом дереве поиска в середине файла представлен на рисунке 16.

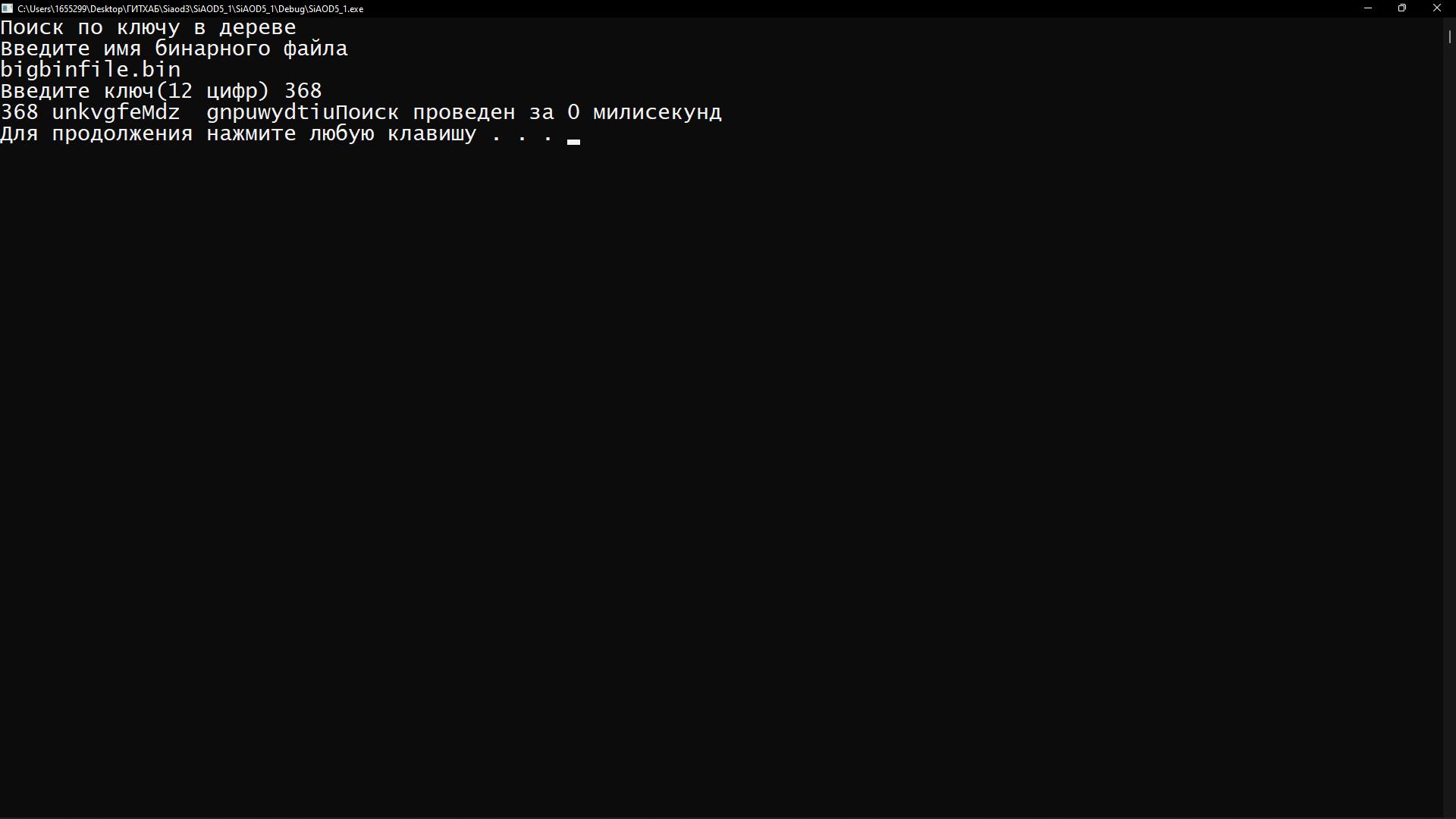


Рисунок 16. Поиск в косом дереве поиска в середине файла

Поиск в косом дереве поиска в конце файла представлен на рисунке 17.

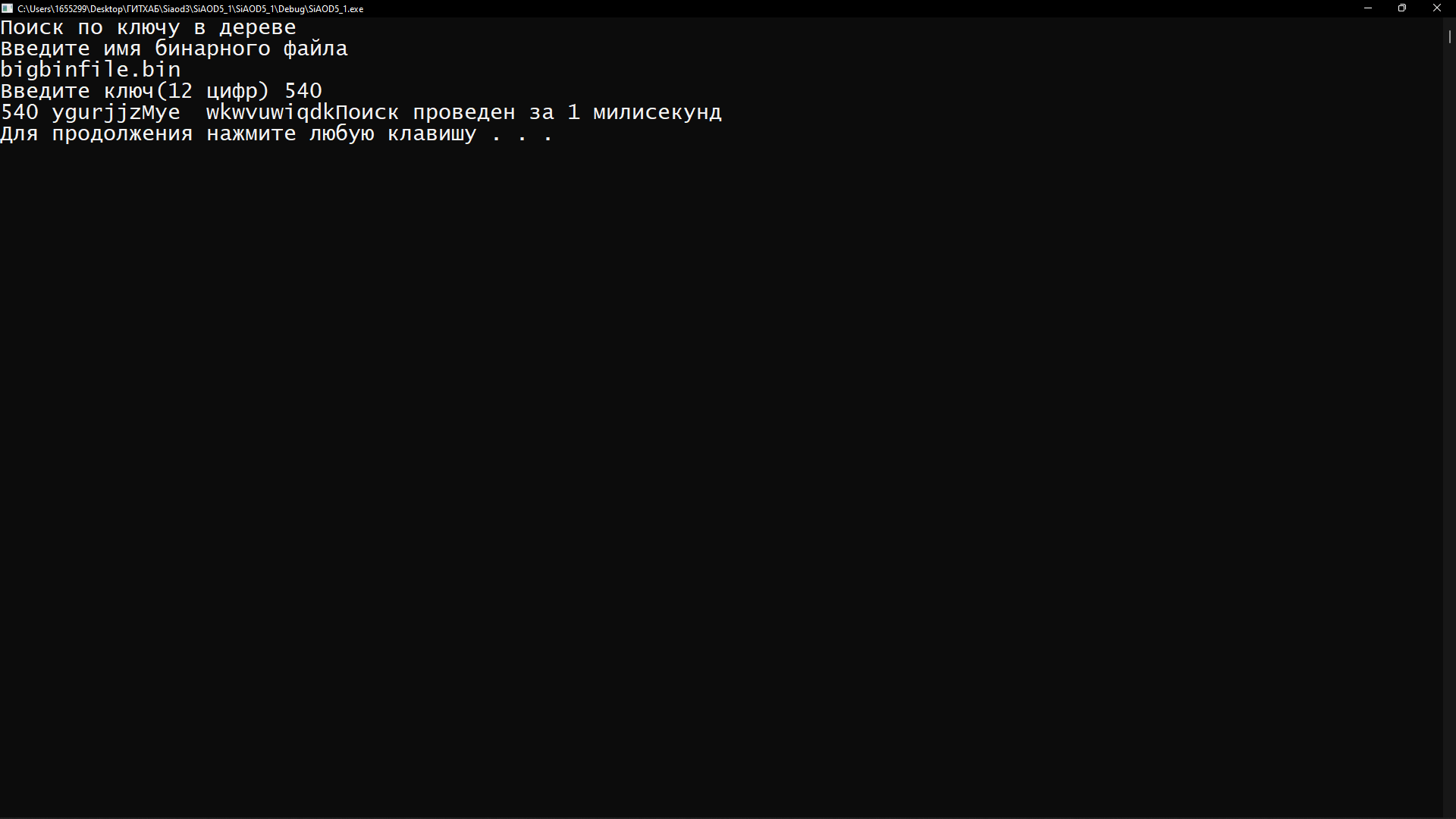


Рисунок 17. Поиск в косом дереве поиска в конце файла

Удаление в косом дереве поиска представлено на рисунке 18.

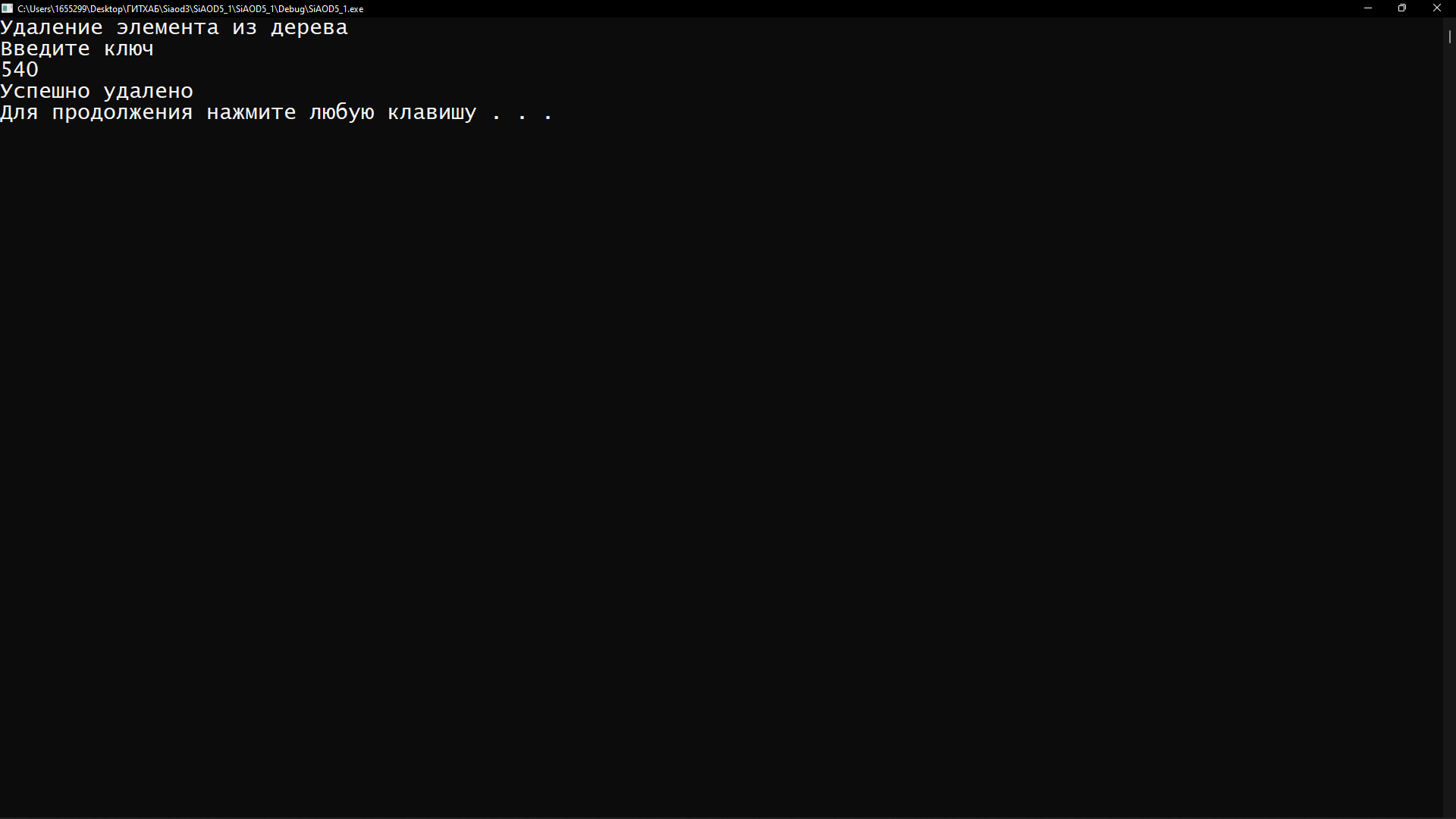


Рисунок 18. Удаление в косом дереве поиска

Таким образом, после проведения тестирования можно подтвердить, что программа работает правильно. Время доступа к записям в деревьях больше/, чем при прямом доступе.

# Вывод

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

1. Улучшены знания о принципе работы бинарных деревьев;
2. Улучшены знания о работе с бинарными файлами;
3. Получены знания о косых деревьях.

# Исходный код программы

Source.cpp

|  |
| --- |
| #include"BST.h"  #include "BinFile.h"  #include "SplayTree.h"  #include <ctime>  using namespace std;  extern double counter;  char getchar(int k)  {  char let[] = { 'a','b','c','d','e','f','g','h','i','j','k','l','m','n','o','p','q','r','s','t','u','v','w','x','y','z' };  return let[k];  }  char word()  {  char word;  int k = rand() % 26 + 1;  word = getchar(k);  return word;  }  void generate(int size)  {  ofstream l;  l.open("bigfile.txt", ios::app);  Book k;  for (long long int i = 1; i <= size; i++)  {  k.fam[10] = '\0';  k.name[10] = '\0';  int k10 = rand() % 9+1;  int k11 = rand() % 10;  int k12 = rand() % 10;  k.ISBN = k10 \* 100 + k11 \* 10 + k12;  for (int i = 0; i < 10; i++)  {  k.fam[i] = word();  k.name[i] = word();  }  l << k.ISBN << " " << k.fam << " " << k.name << endl;  }  l.close();  }  int main()  {  setlocale(LC\_ALL, "rus");  class node \*root = NULL;  class nodesplay \*root\_splay = NULL;  string txt\_file, bin\_file;  long long int key;  Book temp;  int adress;  long long int \*ISBN = new long long int[10000];  int i = 0;  while (true)  {  system("cls");  cout << "Введите номер задания:\n1-Создание файла\n2-Поиск с помощью линейного поиска\n3-вывод записи по ее адресу\n4-Добавление записи в файл\n5-Построение бинарного дерева по файлу\n6-Добавление элемента в дерево\n7-Поиск по ключу в дереве\n8-Удаление элемента из дерева\n9-Вывод дерева\n10-Построение косого дерева по файлу\n11-Добавление элемента в дерево\n12-Поиск по ключу в дереве\n13-Удаление элемента из дерева\n14-Вывод дерева\nЛюбой другой номер закроет программу" << endl;  int k = 0;  cin >> k;  if (k == 1)  {  system("cls");  cout << "Создание файла" << endl;  cout << "Введите имя текстового файла" << endl;  cin >> txt\_file;  cout << "Введите имя бинарного файла" << endl;  cin >> bin\_file;  if (BinFile::txt\_to\_bin(txt\_file, bin\_file) == 1)  {  cout << "Успешно создано" << endl;  }  else  {  cout << "Файл не существует" << endl;  }  system("pause");  }  if (k == 2)  {  system("cls");  cout << "Поиск с помощью линейного поиска" << endl;  cout << "Введите имя бинарного файла" << endl;  cin >> bin\_file;  cout << "Введите ключ (12 цифр)" << endl;  cin >> key;  unsigned int start\_time = clock();  temp=BinFile::linear\_search(bin\_file, key);  unsigned int finish\_time = clock();  if (temp.ISBN != 0)  {  cout << temp.ISBN << " " << temp.fam << " " << temp.name;  unsigned int rez\_time = finish\_time - start\_time;  cout << "Поиск проведен за " <<rez\_time <<" милисекунд"<< endl;  }  else  {  cout <<"Запись с таким ключом не найдена" << endl;;  }  system("pause");  }  if (k == 3)  {  system("cls");  cout << "вывод записи по ее адресу" << endl;  cout << "Введите имя бинарного файла" << endl;  cin >> bin\_file;  cout << "Введите позицию" << endl;  cin >> adress;  temp=BinFile::get\_class(bin\_file, adress);  if (temp.ISBN != 0)  {  cout << temp.ISBN << " " << temp.fam << " " << temp.name;  }  else  {  cout << "Запись с таким ключом не найдена" << endl;;  }  system("pause");  }  if (k == 4)  {  system("cls");  cout << "Добавление записи в файл" << endl;  cout << "Введите имя бинарного файла" << endl;  cin >> bin\_file;  ofstream outp;  outp.open(bin\_file, ios::binary || ios::app);  cout << "Введите ISBN(12 цифр) ";  cin >> temp.ISBN;  cout << "Введите фамилию автора(английский язык) ";  cin >> temp.fam;  cout << "Введите название книги(английский язык) ";  cin >> temp.name;  if (outp.good())  {  BinFile::AddToBin(outp, temp);  }  else  {  cout << "Такого файла не существует" << endl;  }  outp.close();  system("pause");  }  if (k == 5)  {  system("cls");  cout << "Построение бинарного дерева по файлу" << endl;  cout << "Введите имя бинарного файла" << endl;  cin >> bin\_file;  ifstream inp;  inp.open(bin\_file, ios::binary);  if (inp.good())  {  inp.close();  ISBN = BinFile::getISBN(bin\_file);  while (ISBN[i] != 0)  {  root = node::insert(root, ISBN[i], i);  i++;  }  cout << "Успешно" << endl;  }  else  {  cout << "Файл не существует" << endl;  }  system("pause");  }  if (k == 6)  {  system("cls");  cout << "Добавление элемента в дерево" << endl;  cout << "Введите имя бинарного файла" << endl;  cin >> bin\_file;  ofstream outp;  outp.open(bin\_file, ios::binary || ios::app);  cout << "Введите ISBN(12 цифр) ";  cin >> temp.ISBN;  cout << "Введите фамилию автора(английский язык) ";  cin >> temp.fam;  cout << "Введите название книги(английский язык) ";  cin >> temp.name;  if (outp.good())  {  BinFile::AddToBin(outp, temp);  root = node::insert(root, temp.ISBN, i);  i++;  }  else  {  cout << "Такого файла не существует" << endl;  }  outp.close();  system("pause");  }  if (k == 7)  {  system("cls");  cout << "Поиск по ключу в дереве" << endl;  cout << "Введите имя бинарного файла" << endl;  cin >> bin\_file;  ofstream outp;  outp.open(bin\_file, ios::binary || ios::app);  if (outp.good())  {  outp.close();  cout << "Введите ключ(12 цифр) ";  cin >> key;  unsigned int start\_time = clock();  adress=node::search(root, key);  if (adress != -1)  {  temp = BinFile::get\_class(bin\_file, adress);  unsigned finish\_time = clock();  unsigned rez\_time = finish\_time - start\_time;  cout << temp.ISBN << " " << temp.fam << " " << temp.name;  cout << "Поиск проведен за " << rez\_time<< " милисекунд" << endl;  }  else  {  cout << "Запись с таким ключом не найдена" << endl;  }  }  else  {  cout << "такого файла не существует" << endl;  }  system("pause");  }  if (k == 8)  {  system("cls");  cout << "Удаление элемента из дерева" << endl;  cout << "Введите ключ" << endl;  cin >> key;  if (node::search(root, key) != -1)  {  root = node::deleteNode(root, key);  cout << "Успешно удалено" << endl;  }  else  {  cout << "Запись с таким ключом не найдена" << endl;  }  system("pause");  }  if (k == 9)  {  system("cls");  cout << "Вывод дерева" << endl;  cout << "Вывод ключей дерева" << endl;  node::printBT("", root, false, 1);  cout << "Вывод значений дерева" << endl;  node::printBT("", root, false, 0);  system("pause");  }  if (k == 10)  {  system("cls");  cout << "Построение косого дерева по файлу" << endl;  cout << "Введите имя бинарного файла" << endl;  cin >> bin\_file;  ifstream inp;  i = 0;  inp.open(bin\_file, ios::binary || ios::app);  if (inp.good())  {  inp.close();  ISBN = BinFile::getISBN(bin\_file);  while (ISBN[i] != 0)  {  root\_splay = nodesplay::insertsplay(root\_splay, ISBN[i], i, 0);  i++;  }  cout << "Успешно" << endl;  }  else  {  cout << "Файл не существует" << endl;  }  system("pause");  }  if (k == 11)  {  system("cls");  cout << "Добавление элемента в дерево" << endl;  cout << "Введите имя бинарного файла" << endl;  cin >> bin\_file;  ofstream outp;  outp.open(bin\_file, ios::binary || ios::app);  cout << "Введите ISBN(12 цифр) ";  cin >> temp.ISBN;  cout << "Введите фамилию автора(английский язык) ";  cin >> temp.fam;  cout << "Введите название книги(английский язык) ";  cin >> temp.name;  if (outp.good())  {  BinFile::AddToBin(outp, temp);  root\_splay = nodesplay::insertsplay(root\_splay, temp.ISBN, i, 0);  i++;  }  else  {  cout << "Такого файла не существует" << endl;  }  outp.close();  system("pause");  }  if (k == 12)  {  system("cls");  cout << "Поиск по ключу в дереве" << endl;  cout << "Введите имя бинарного файла" << endl;  cin >> bin\_file;  ofstream outp;  outp.open(bin\_file, ios::binary || ios::app);  if (outp.good())  {  cout << "Введите ключ(12 цифр) ";  cin >> key;  unsigned int start\_time = clock();  root\_splay=nodesplay::searchsplay(root\_splay, key);  if (root\_splay->key==key)  {  temp = BinFile::get\_class(bin\_file, root\_splay->value);  unsigned int finish\_time = clock();  unsigned rez\_time = finish\_time - start\_time;  cout << temp.ISBN << " " << temp.fam << " " << temp.name;  cout << "Поиск проведен за " << rez\_time<< " милисекунд" << endl;  }  else  {  cout << "Запись с таким ключом не найдена" << endl;  }  }  else  {  cout << "такого файла не существует" << endl;  }  outp.close();  system("pause");  }  if (k == 13)  {  system("cls");  cout << "Удаление элемента из дерева" << endl;  cout << "Введите ключ" << endl;  cin >> key;  root\_splay = nodesplay::searchsplay(root\_splay, key);  if (root\_splay->key == key)  {  root\_splay = nodesplay::removesplay(root\_splay, key);  cout << "Успешно удалено" << endl;  }  else  {  cout << "Запись с таким ключом не найдена" << endl;  }  system("pause");  }  if (k == 14)  {  system("cls");  cout << "Вывод дерева" << endl;  cout << "Вывод ключей дерева" << endl;  nodesplay::printsplayBT("", root\_splay, false, 1);  cout << "Вывод значений дерева" << endl;  nodesplay::printsplayBT("", root\_splay, false, 0);  system("pause");  }  if (k == 15)  {  ofstream l;  l.open("bigfile.txt");  l << "";  l.close();  generate(1000);  system("pause");  }  if (k < 1 || k>15)  {  cout << "Вы уверены, что хотите выйти? Y/N" << endl;  char j;  cin >> j;  if (j == 'Y')  {  cout << "Выход... ";  system("pause");  exit(0);  }  }  }  } |

BST.h

|  |
| --- |
| #pragma once  #include <iostream>  #include <string>  using namespace std;  class node  {  public:  int value;  long long int key;  node \*left, \*right;  node(long long int a, int b)  {  key = a;  value = b;  left = nullptr;  right = nullptr;  }  static bool exists(node \* root, long long int a);  static int search(node \* root, long long int a);  static node \* insert(node \* root, long long int a, int b);  static class node \*minValueNode(class node \*node);  static class node \*deleteNode(class node \*root, long long int key);  static void printBT(const string& prefix, const node\* node, bool isLeft, bool key);  }; |

BST.cpp

|  |
| --- |
| #include "BST.h"  bool node::exists(node \* root, long long int a)  {  if (root == nullptr)  {  return false;  }  if (root->key == a)  {  return true;  }  if (a < root->key)  {  return exists(root->left, a);  }  return exists(root->right, a);  }  int node::search(node \* root, long long int a)  {  if (root == nullptr)  {  return -1;  }  if (root->key == a)  {  return root->value;  }  if (a < root->key)  {  return search(root->left, a);  }  return search(root->right, a);  }  node \* node::insert(node \* root, long long int a, int b)  {  if (exists(root, a))  {  return root;  }  if (root == nullptr)  {  return new node(a, b);  }  if (a < root->key)  {  root->left = insert(root->left, a, b);  }  if (a > root->key)  {  root->right = insert(root->right, a, b);  }  return root;  }  node \* node::minValueNode(class node \*node)  {  class node \*current = node;  while (current && current->left != NULL)  {  current = current->left;  }  return current;  }  node \* node::deleteNode(node \* root, long long int key)  {  if (root == NULL)  {  return root;  }  if (key < root->key)  {  root->left = deleteNode(root->left, key);  }  else if (key > root->key)  {  root->right = deleteNode(root->right, key);  }  else  {  if (root->left == NULL)  {  class node \*temp = root->right;  free(root);  return temp;  }  else if (root->right == NULL)  {  class node \*temp = root->left;  free(root);  return temp;  }  class node \*temp = minValueNode(root->right);  root->key = temp->key;  root->value = temp->value;  root->right = deleteNode(root->right, temp->key);  }  return root;  }  void node::printBT(const string& prefix, const node\* node, bool isLeft, bool key)  {  if (key)  {  if (node != nullptr)  {  std::cout << prefix;  std::cout << (isLeft ? "|-----------" : "L-----------");  std::cout << node->key << endl;  printBT(prefix + (isLeft ? "| " : " "), node->right, true,1);  printBT(prefix + (isLeft ? "| " : " "), node->left, false,1);  }  }  else  {  if (node != nullptr)  {  std::cout << prefix;  std::cout << (isLeft ? "|--" : "L--");  std::cout << node->value << endl;  printBT(prefix + (isLeft ? "| " : " "), node->right, true,0);  printBT(prefix + (isLeft ? "| " : " "), node->left, false,0);  }  }  } |

BinFile.h

|  |
| --- |
| #pragma once  #include <iostream>  #include <fstream>  #include <string>  using namespace std;  struct Book  {  long long int ISBN;  char fam[50];  char name[50];  };  class BinFile  {  public:  static int txt\_to\_bin(string txt\_file, string bin\_file);  static Book get\_class(string filename, int position);  static void AddToBin(ofstream& bin\_file, Book book);  static Book linear\_search(string bin\_file, long long int key);  static long long int\* getISBN(string txt\_file);  }; |

BinFile.cpp

|  |
| --- |
| #include "BinFile.h"  int BinFile::txt\_to\_bin(string txt\_file, string bin\_file)  {  ifstream inp;  inp.open(txt\_file,ios::in);  ofstream outp;  outp.open(bin\_file, ios::binary || ios::app);  if (inp.good() && outp.good())  {  Book tt;  while (true)  {  inp >> tt.ISBN;  if (!inp.eof())  {  inp >> tt.fam >> tt.name;  outp.write((char\*)&tt, sizeof(Book));  }  else break;  }  inp.close();  outp.close();  return 1;  }  else  {  inp.close();  outp.close();  return 0;  }  }  Book BinFile::get\_class(string filename, int position)  {  Book book;  ifstream bin\_file(filename, ios::binary);  bin\_file.seekg((position) \* sizeof(book));  bin\_file.read((char\*)&book, sizeof(Book));  bin\_file.close();  return book;  }  void BinFile::AddToBin(ofstream& bin\_file, Book book)  {  bin\_file.seekp(0, ios::end);  bin\_file.write((char\*)&book, sizeof(Book));  }  Book BinFile::linear\_search(string bin\_file, long long int key)  {  ifstream inp;  inp.open(bin\_file, ios::binary );  Book tt;  Book m;  m.ISBN = 0;  while (true)  {  inp.read((char\*)&tt, sizeof(Book));  if (!inp.eof())  {  if (tt.ISBN == key)  {  m.ISBN = tt.ISBN;  for (int i = 0; i < 50; i++)  {  m.fam[i] = tt.fam[i];  m.name[i] = tt.name[i];  }  }  }  else  {  break;  }  }  inp.close();  return m;  }  long long int\* BinFile::getISBN(string txt\_file)  {  ifstream inp;  inp.open(txt\_file, ios::binary);  Book tt;  long long int ISBN[10000] = {0};  int i = 0;  while (true)  {  inp.read((char\*)&tt, sizeof(Book));  if (!inp.eof())  {  ISBN[i] = tt.ISBN;  i++;  }  else  {  break;  }  }  inp.close();  return ISBN;  } |

SplayTree.h

|  |
| --- |
| #pragma once  #include <string>  using namespace std;  extern double counter;  class nodesplay  {  public:  long long int key;  int value;  nodesplay \*left, \*right;  static void printsplayBT(const string& prefix, nodesplay\* node, bool isLeft, bool key);  static nodesplay\* newNode(long long int key);  static nodesplay \*rightRotate(nodesplay \*x);  static nodesplay \*leftRotate(nodesplay \*x);  static nodesplay \*splay(nodesplay \*root, long long int key);  static nodesplay \*searchsplay(nodesplay \*root, long long int key);  static void preOrder(nodesplay \*root);  static nodesplay \*insertsplay(nodesplay \*root, long long int key, int value, bool depth);  static nodesplay \*tree\_max\_key(nodesplay \*root);  static nodesplay \*merge(nodesplay \*tree1, nodesplay \*tree2);  static nodesplay \*removesplay(nodesplay \*root, long long int key);  }; |

SplayTree.cpp

|  |
| --- |
| #include "SplayTree.h"  #include "BST.h"  double counter = 0;  nodesplay\* nodesplay::newNode(long long int key)  {  nodesplay\* Node = new nodesplay();  Node->key = key;  Node->left = Node->right = NULL;  return (Node);  }  nodesplay \*nodesplay::rightRotate(nodesplay \*x)  {  nodesplay \*y = x->left;  x->left = y->right;  y->right = x;  counter++;  return y;  }  nodesplay \*nodesplay::leftRotate(nodesplay \*x)  {  nodesplay \*y = x->right;  x->right = y->left;  y->left = x;  counter++;  return y;  }  nodesplay \*nodesplay::splay(nodesplay \*root, long long int key)  {  if (root == NULL || root->key == key)  return root;  if (root->key > key)  {  if (root->left == NULL)  {  return root;  }  // Zig-Zig (Левый-левый)  if (root->left->key > key)  {  // Сначала рекурсивно поднимаем  // ключ как корень left-left  root->left->left = splay(root->left->left, key);  // Первый разворот для root,  // второй разворот выполняется после else  root = rightRotate(root);  }  else if (root->left->key < key) // Zig-Zag (Левый-правый)  {  // Сначала рекурсивно поднимаем  // ключ как корень left-right  root->left->right = splay(root->left->right, key);  // Выполняем первый разворот для root->left  if (root->left->right != NULL)  root->left = leftRotate(root->left);  }  // Выполняем второй разворот для корня  return (root->left == NULL) ? root : rightRotate(root);  }  else // Ключ находится в правом поддереве  {  // Ключа нет в дереве, мы закончили  if (root->right == NULL)  {  return root;  }  // Zag-Zig (Правый-левый)  if (root->right->key > key)  {  // Поднять ключ как корень right-left  root->right->left = splay(root->right->left, key);  // Выполняем первый поворот для root->right  if (root->right->left != NULL)  root->right = rightRotate(root->right);  }  else if (root->right->key < key)// Zag-Zag (Правый-правый)  {  // Поднимаем ключ как корень  // right-right и выполняем первый разворот  root->right->right = splay(root->right->right, key);  root = leftRotate(root);  }  // Выполняем второй разворот для root  return (root->right == NULL) ? root : leftRotate(root);  }  }  nodesplay \*nodesplay::searchsplay(nodesplay \*root, long long int key)  {  return splay(root, key);  }  void nodesplay::preOrder(nodesplay \*root)  {  if (root != NULL)  {  cout << root->key << " ";  preOrder(root->left);  preOrder(root->right);  }  }  nodesplay \*nodesplay::insertsplay(nodesplay \*root, long long int key, int value, bool depth)  {  if (root == nullptr)  {  root = newNode(key);  root->value = value;  }  else if (key < root->key)  {  if (root->left == nullptr)  {  root->left = newNode(key);  root->left->value = value;  if (!depth)  {  root = splay(root, key);  }  }  else  {  root->left = insertsplay(root->left, key, value, 1);  root = splay(root, key);  }  }  else  {  if (root->right == nullptr)  {  root->right = newNode(key);  root->right->value = value;  if (!depth)  {  root = splay(root, key);  }  }  else  {  root->right = insertsplay(root->right, key, value, 1);  root = splay(root, key);  }  }  return root;  }  nodesplay \*nodesplay::tree\_max\_key(nodesplay \*root)  {  nodesplay \*curr = root;  while (curr->right != nullptr)  {  curr = curr->right;  }  return curr;  }  nodesplay \*nodesplay::merge(nodesplay \*tree1, nodesplay \*tree2)  {  tree1 = splay(tree1, tree\_max\_key(tree1)->key);  tree1->right = tree2;  return tree1;  }  nodesplay \*nodesplay::removesplay(nodesplay \*root, long long int key)  {  root = splay(root, key);  class nodesplay \*tree1 = NULL;  class nodesplay \*tree2 = NULL;  tree1 = root->left;  tree2 = root->right;  root = merge(tree1, tree2);  return root;  }  void nodesplay::printsplayBT(const string& prefix, nodesplay\* node, bool isLeft, bool key)  {  if (key)  {  if (node != nullptr)  {  std::cout << prefix;  std::cout << (isLeft ? "|-----------" : "L-----------");  std::cout << node->key << endl;  printsplayBT(prefix + (isLeft ? "| " : " "), node->right, true, 1);  printsplayBT(prefix + (isLeft ? "| " : " "), node->left, false, 1);  }  }  else  {  if (node != nullptr)  {  std::cout << prefix;  std::cout << (isLeft ? "|--" : "L--");  std::cout << node->value << endl;  printsplayBT(prefix + (isLeft ? "| " : " "), node->right, true, 0);  printsplayBT(prefix + (isLeft ? "| " : " "), node->left, false, 0);  }  }  } |