Белорусский государственный технологический университет

Факультет информационных технологий

Кафедра программной инженерии

Лабораторная работа 4

По дисциплине «Математическое программирование»

На тему «Динамическое программирование»

Выполнил:

Студентка 2 курса 9 группы

Волосюк Арина Вадимовна

Преподаватель: Ромыш А.С.

2025, Минск

**ЦЕЛЬ РАБОТЫ:** освоить общие принципы решения задач методом ветвей и границ, решить задачу о коммивояжере данным методом, сравнить полученное решение задачи с комбинаторным методом перестановок.

**Ход работы**

**Задание 1.**На языке С++ сгенерировать случайным образом строку букв латинского алфавита S1 длиной 300 символов и S2длиной 200.

Реализация функции показана в листинге 1.

Листинг 1 – Код функции generate\_random\_string

|  |
| --- |
| std::string generate\_random\_string(int length) {  static const char alphabet[] = "abcdefghijklmnopqrstuvwxyz";  static std::random\_device rd; // Источник случайных чисел  static std::mt19937 gen(rd()); // Генератор случайных чисел  static std::uniform\_int\_distribution<> dis(0, sizeof(alphabet) - 2); // Равномерное распределение  std::string result;  for (int i = 0; i < length; ++i) {  result += alphabet[dis(gen)];  }  return result;  } |

**Задание 2.** Вычислить двумя способами (рекурсивно и с помощью динамического программирования)  – дистанцию Левенштейна для , где - длина строки ,  - строка состоящая из первых  символов строки . (копии экрана и код вставить в отчет).

Реализация функций показана в листинге 2,3,4.

Листинг 2 – Код файла Levenshtein.cpp

|  |
| --- |
| // - Levenshtein.cpp  #include "Levenshtein.h"  #include <iomanip>  #include <algorithm>  #include "Levenshtein.h"  #define DD(i,j) d[(i)\*(ly+1)+(j)]  int min3(int x1, int x2, int x3)  {  return std::min(std::min(x1, x2), x3);  }  int levenshtein(int lx, const char x[], int ly, const char y[])  {  int\* d = new int[(lx + 1) \* (ly + 1)];  for (int i = 0; i <= lx; i++) DD(i, 0) = i;  for (int j = 0; j <= ly; j++) DD(0, j) = j;  for (int i = 1; i <= lx; i++)  for (int j = 1; j <= ly; j++)  {  DD(i, j) = min3(DD(i - 1, j) + 1, DD(i, j - 1) + 1,  DD(i - 1, j - 1) + (x[i - 1] == y[j - 1] ? 0 : 1));  }  return DD(lx, ly);  }  int levenshtein\_r(  int lx, const char x[],  int ly, const char y[]  )  {  int rc = 0;  if (lx == 0) rc = ly;  else if (ly == 0) rc = lx;  else if (lx == 1 && ly == 1 && x[0] == y[0]) rc = 0;  else if (lx == 1 && ly == 1 && x[0] != y[0]) rc = 1;  else rc = min3(  levenshtein\_r(lx - 1, x, ly, y) + 1,  levenshtein\_r(lx, x, ly - 1, y) + 1,  levenshtein\_r(lx - 1, x, ly - 1, y) + (x[lx - 1] == y[ly - 1] ? 0 : 1)  );  return rc;  }; |

Листинг 3 – Прототипы функций Levenshtein.h

|  |
| --- |
| #pragma once  // - Levenshtein.h  // -- дистанции Левенштeйна (динамическое программирование)  int levenshtein(  int lx, // длина слова x  const char x[], // слово длиной lx  int ly, // длина слова y  const char y[] // слово y  );  // -- дистанции Левенштeйна (рекурсия)  int levenshtein\_r(  int lx, // длина строки x  const char x[], // строка длиной lx  int ly, // длина строки y  const char y[] // строка y  ); |

Листинг 4 –Код файла LevenshteinDistance.h

|  |
| --- |
| #include "Levenshtein.h"  #include <algorithm>  #include <iostream>  #include <iomanip>  #include <chrono>  #include <random>  std::string generate\_random\_string(int length) {  static const char alphabet[] = "abcdefghijklmnopqrstuvwxyz";  static std::random\_device rd; // Источник случайных чисел  static std::mt19937 gen(rd()); // Генератор случайных чисел  static std::uniform\_int\_distribution<> dis(0, sizeof(alphabet) - 2); // Равномерное распределение  std::string result;  for (int i = 0; i < length; ++i) {  result += alphabet[dis(gen)];  }  return result;  }  int main()  {  setlocale(LC\_ALL, "rus");  int lx = 15;  int ly = 15;  std::string x = generate\_random\_string(lx);  std::string y = generate\_random\_string(ly);  std::cout << "Сгенерированная строка x: " << x << std::endl;  std::cout << "Сгенерированная строка y: " << y << std::endl;  std::cout << std::endl << "-- расстояние Левенштейна -----" << std::endl;  std::cout << std::endl << "--длина --- рекурсия -- дин.програм. ---"  << std::endl;  std::cout << std::endl;  std::cout << std::endl << "-- расстояние Левенштейна -----" << std::endl;  std::cout << std::endl << "--длина --- рекурсия(мс) -- дин.програм.(мс) ---"  << std::endl;  for (int i = 8; i < std::min(lx, ly); i++)  {  // Измерение времени для рекурсивного метода  auto start1 = std::chrono::high\_resolution\_clock::now();  levenshtein\_r(i, x, i - 2, y);  auto end1 = std::chrono::high\_resolution\_clock::now();  auto duration1 = std::chrono::duration\_cast<std::chrono::microseconds>(end1 - start1).count();  // Измерение времени для метода динамического программирования  auto start2 = std::chrono::high\_resolution\_clock::now();  levenshtein(i, x, i - 2, y);  auto end2 = std::chrono::high\_resolution\_clock::now();  auto duration2 = std::chrono::duration\_cast<std::chrono::microseconds>(end2 - start2).count();  // Вывод результатов  std::cout << std::right << std::setw(2) << i - 2 << "/" << std::setw(2) << i  << " " << std::left << std::setw(10) << duration1  << " " << std::setw(10) << duration2 << std::endl;  }  system("pause");  return 0;  } |

Результат работы программы представлен на рисунке 1.1.

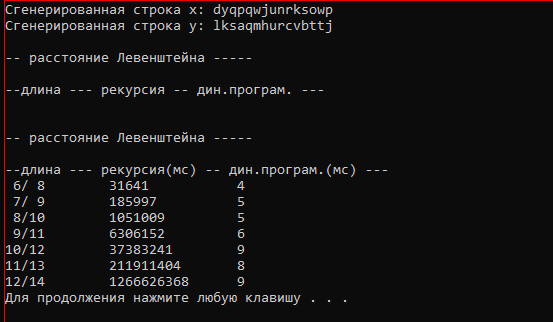


Рисунок 1.1 Результаты работы программы

**Задание 3.**Выполнить сравнительный анализ времени, затраченного на вычисление дистанции Левенштейна для двух методов решения. Построить графики зависимости времени вычисления от . (копии экрана и график вставить в отчет).

Время представлено на рисунке 1.2.

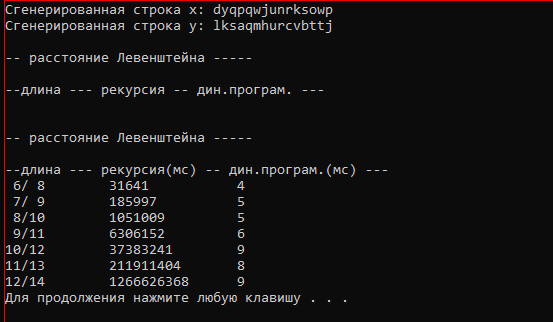


Рисунок 1.2 Результаты работы программы

Измерение скорости выполнения рекурсивного и динамического поиска расстояния Левенштейна предоставлены на рисунке 1.3 и 1,4.

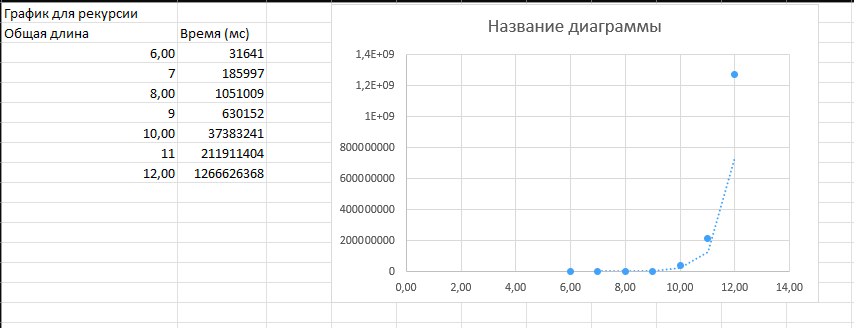


Рисунок 1.3 Результаты измерений и их график

**Вывод:** Скорость выполнения программы возрастает экспоненциально с увеличением итерации. Такая зависимость обусловлена рекурсией.

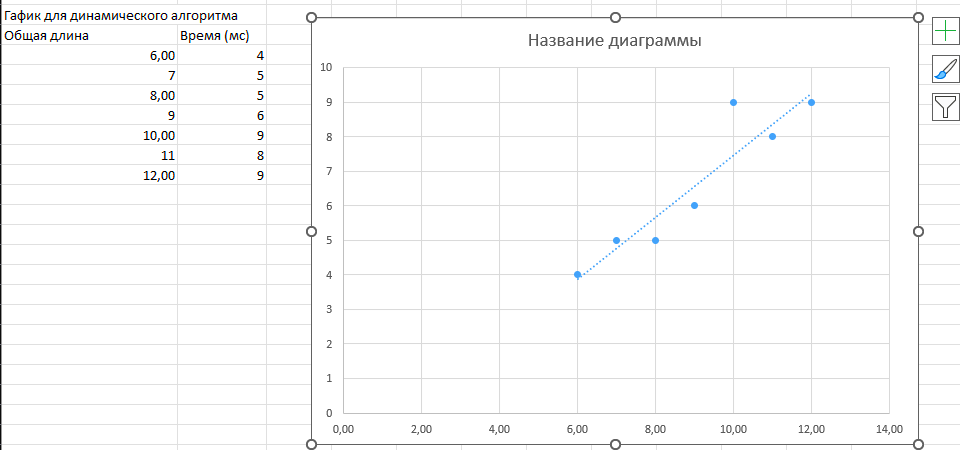


Рисунок 1.2 Результаты измерений и их график

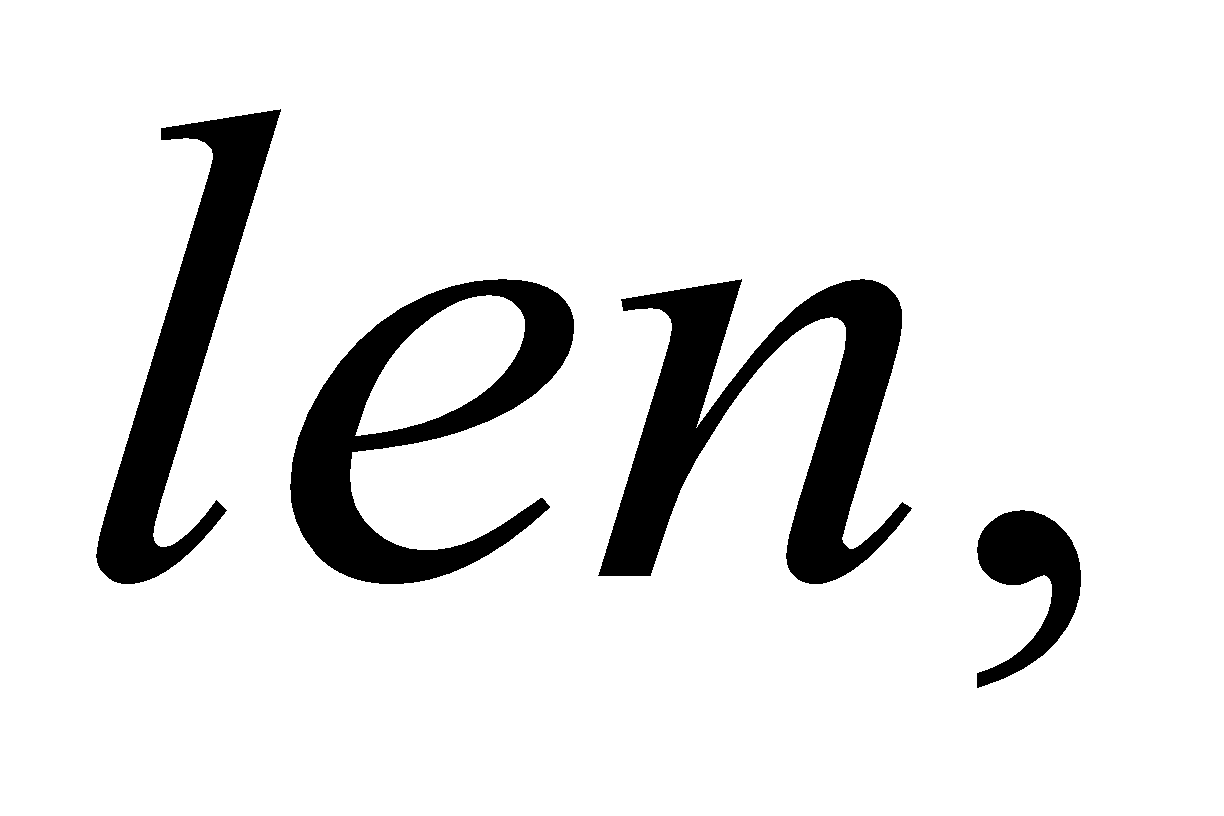
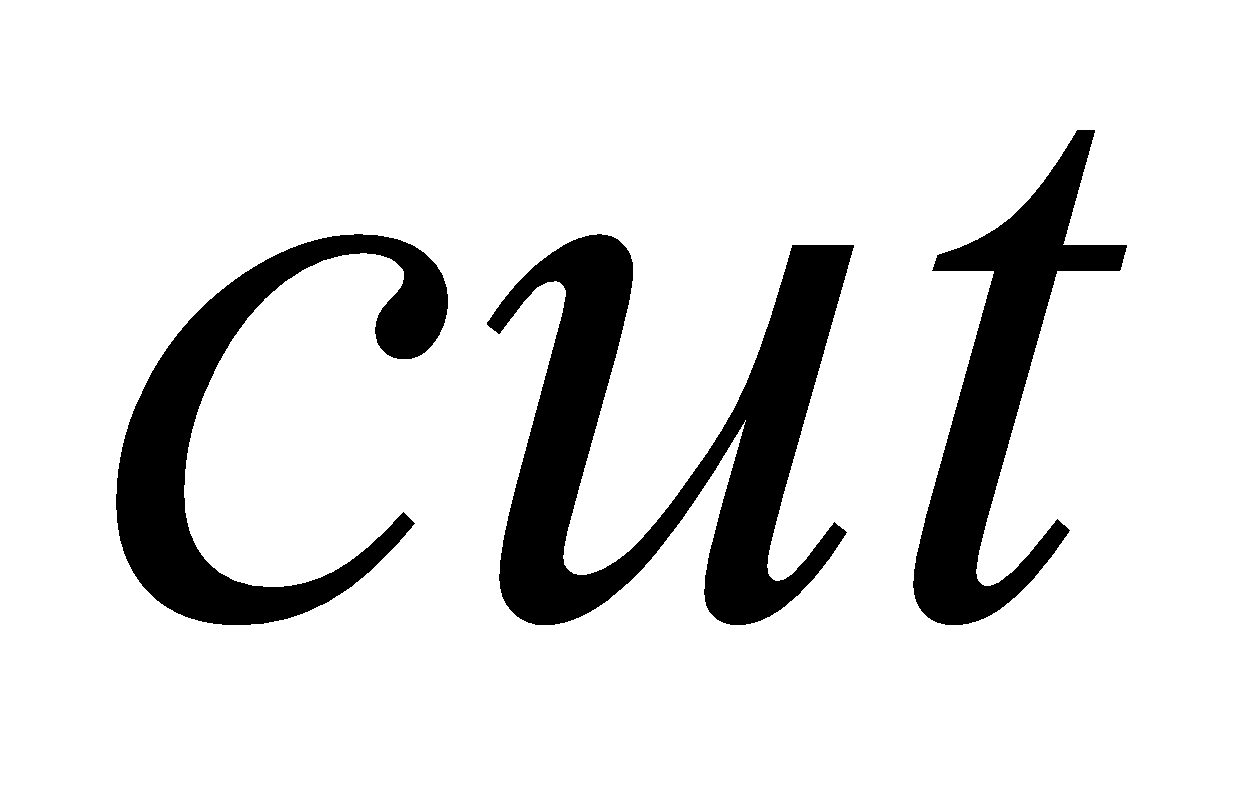
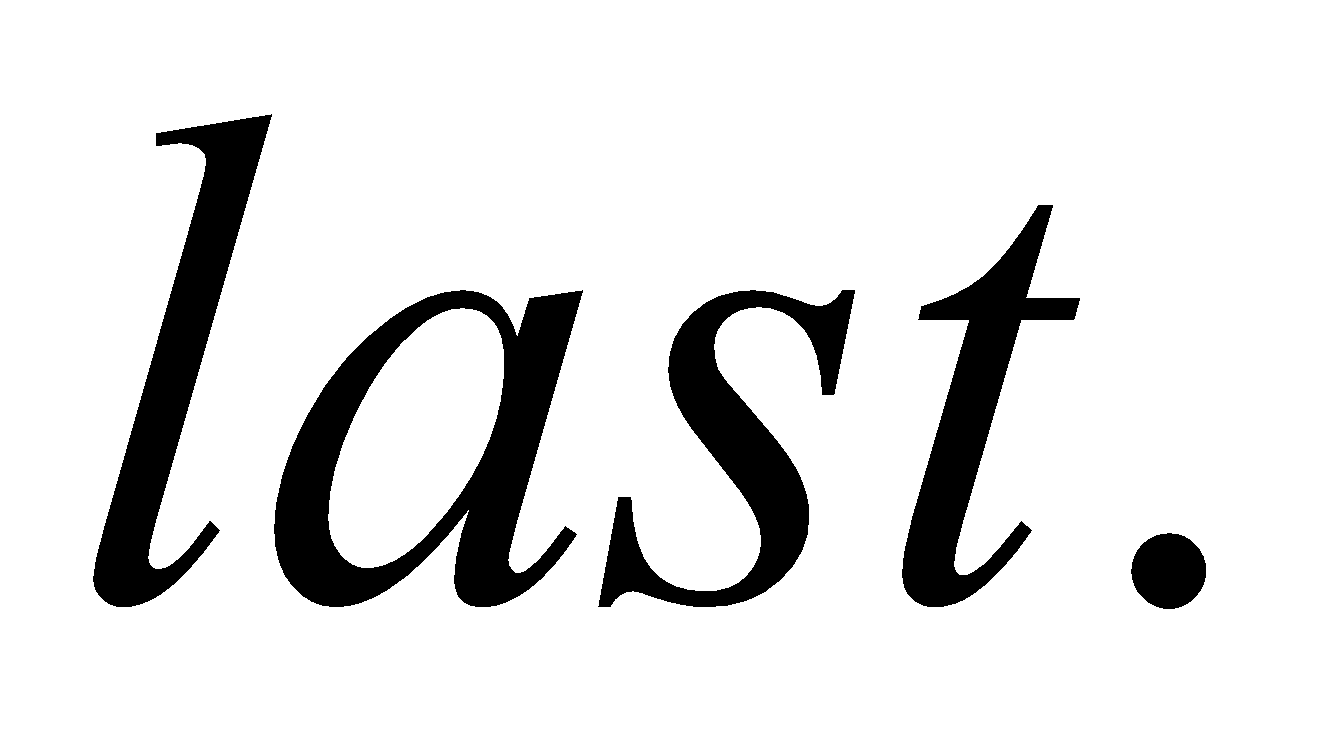
**Вывод**: скорость выполнения программы линейно зависит от количества итераций цикла.

**Задание 4.**Реализовать вручную пример вычисления дистанции Левенштейна при помощи рекурсивного алгоритма (в соответствии с вариантом) (каждый шаг алгоритма по примеру из лекции вставить в отчет).

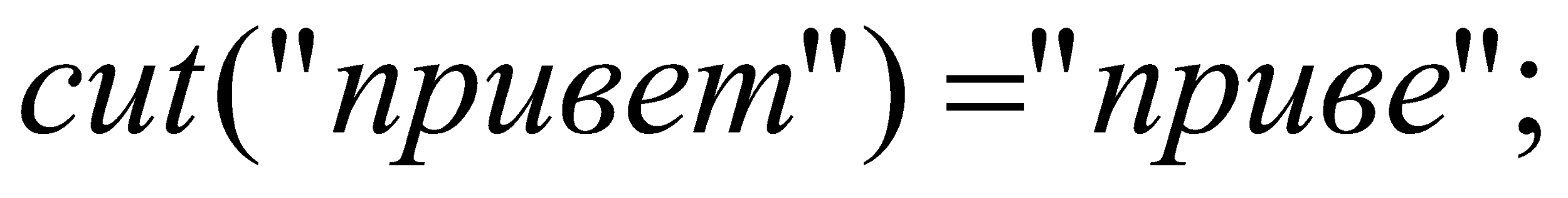
Слова для сравнения: сон, донор

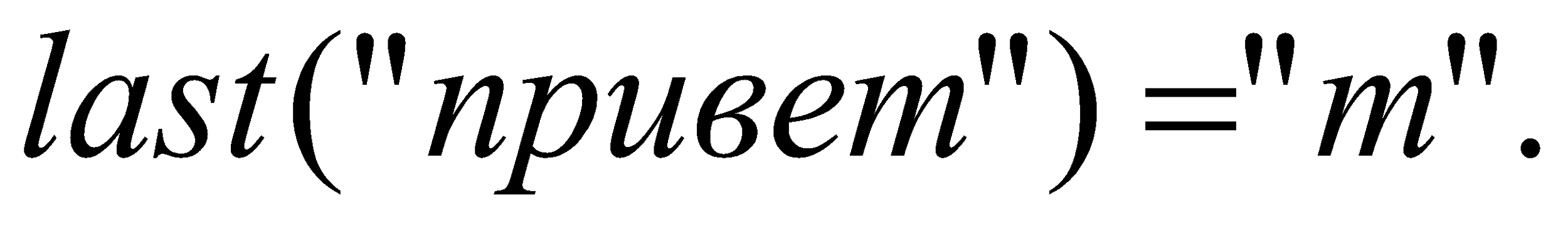
Формула для вычисления дистанции Левенштейна:



В предыдущем выражении используются символы   и  Разъясним их смысл:

 – количество символов в заданной строке. Например, 

 – заданная строка без последнего символа. Например, 

 – последний символ заданной строки. Например, 

1. L(“сон”, “донор”) = min
2. L(“со”, “донор”) = min
3. L(“сон”, “доно”) = min
4. L(“со”, “доно”) = min
5. L(“с”, “донор”) = min , L(“”, “донор”) =5, L(“”, “доно”) =4
6. L(“с”, “доно”) = min , L(“”, “доно”) =4, L(“”, “дон”) =3
7. L(“сон”, “дон”) = min
8. L(“со”, “дон”) = min
9. L(“сон”, “до”) = min
10. L(“сон”, “д”) = min , L(“сон”, “ ”) = 3, L(“со”, “”) = 2
11. L(“с”, “дон”) = min , L(“”, “дон ”) = 3, L(“”, “до”) = 2
12. L(“со”, “до”) = min , L(“с”, “д”) = 1
13. L(“с”, “до”) = min , L(“”, “до”) = 2, L(“”, “д”) = 1, L(“с”, “д”) = 1
14. L(“со”, “д”) = min , L(“с”, “д”) = 1, L(“со”, “”) = 2, L(“с”, “”) = 1
15. L(“со”, “д”) = min(2, 3, 2) = 2
16. L(“с”, “до”) = min(3, 2, 2) = 2
17. L(“со”, “до”) = min(3, 3, 1) = 1
18. L(“с”, “дон”) = min(4, 3, 3) = 3
19. L(“сон”, “д”) = min(3, 4, 3) = 3
20. L(“сон”, “до”) = min(2, 4, 3) = 2
21. L(“со”, “дон”) = min(4, 2, 3) = 2
22. L(“сон”, “дон”) = min(3, 3, 1) = 1
23. L(“с”, “доно”) = min(5, 4, 4) = 4
24. L(“с”, “донор”) = min(6, 5, 5) = 5
25. L(“со”, “доно”) = min(5, 3, 3) = 3
26. L(“сон”, “доно”) = min(4, 2, 3) = 2
27. L(“со”, “донор”) = min(6, 4, 5) = 4
28. L(“сон”, “донор”) = min(5, 3, 4) = 3

**Задание 5.**Нечетные варианты. Выполнить сравнительный анализ времени затраченного на решение задачи о наибольшей общей подпоследовательности для двух методов решения (рекурсивное решение, динамическое программирование). Две последовательности взять в соответствии с вариантом. Построить графики зависимости времени вычисления от . Отобразить ход решения в отчете(по примеру из лекции) + код и копии экрана.

Последовательность: HRWCYUJ, WLPCAU

Измерение скорости выполнения рекурсивного и динамического алгоритма предоставлены на рисунке 1.5 и 1,6.

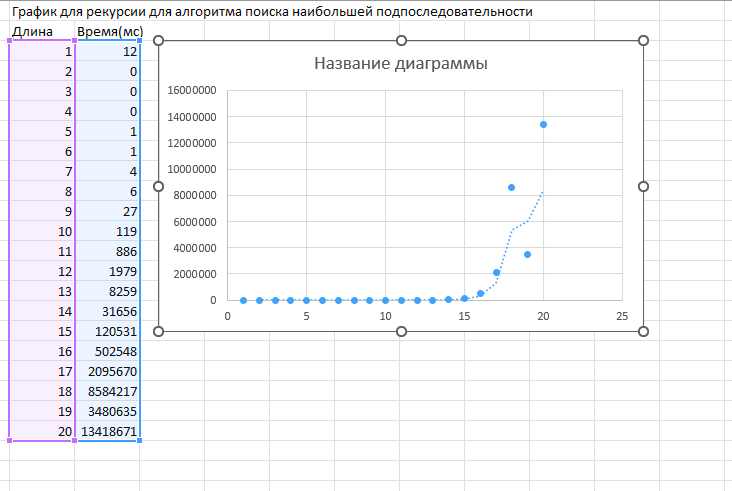


Рисунок 1.5 Результаты измерений и их график

**Вывод**: скорость выполнения программы факториально зависит от увеличения длинны последовательности.

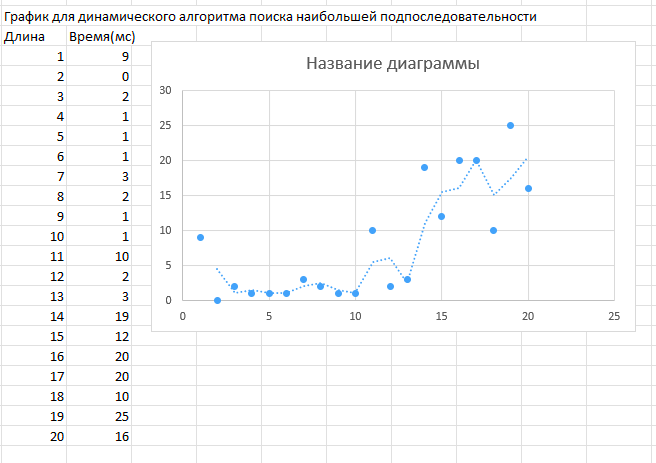


Рисунок 1.6 Результаты измерений и их график

**Вывод**: скорость выполнения программы зависит от увеличения длинны последовательности как n^2.

Результат вычисления представлен на рисунке 1.7.

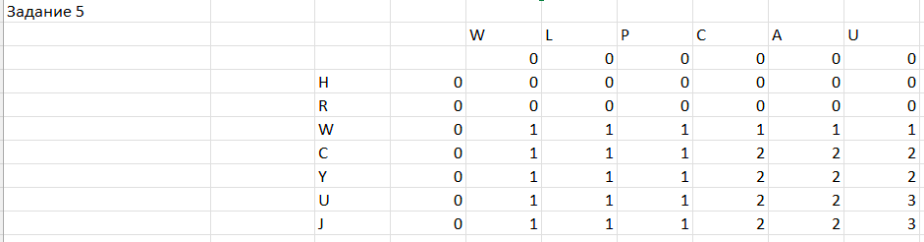


Рисунок 1.7 Результаты измерений и их график

Результат вычисления кода предоставлен на рисунке 1.8.

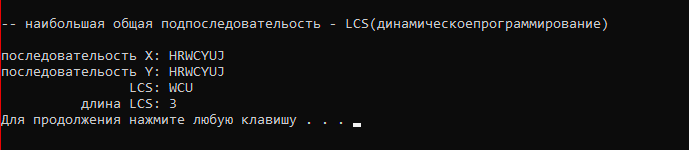


Рисунок 1.7 Результаты измерений и их график

Вывод:

В ходе выполнения лабораторной работы были освоены общие принципы решения задач методом динамического программирования. На примере задачи нахождения наибольшей общей подпоследовательности (LCS) были реализованы и сравнены два подхода: рекурсивный метод и метод динамического программирования.

Основные результаты работы:

Рекурсивный метод показал свою простоту в реализации, однако его время выполнения значительно увеличивается с ростом длины входных данных из-за экспоненциальной сложности. Это делает его неэффективным для задач с большими размерами входных данных.

Метод динамического программирования продемонстрировал высокую эффективность за счет использования таблицы для хранения промежуточных результатов, что позволило избежать повторных вычислений. Время выполнения этого метода оказалось значительно меньше, чем у рекурсивного, особенно для длинных последовательностей.

В ходе эксперимента было проведено сравнение времени выполнения обоих методов для различных длин входных последовательностей. Результаты подтвердили, что динамическое программирование является более предпочтительным подходом для решения задач с высокой вычислительной сложностью.

Таким образом, в процессе работы были успешно освоены принципы динамического программирования, а также проведен анализ и сравнение эффективности рекурсивного и динамического подходов. Полученные результаты подтвердили преимущество динамического программирования для задач, требующих оптимизации времени выполнения.