Белорусский государственный технологический университет

Кафедра Программной инженерии

**“Математическое программирование”**

**Отчет по лабораторной работе №4**

**Динамическое программирование**

**Вариант 3**

Выполнил: Ермаков К.А.

ФИТ 2 курс, 4 группа

Проверил: Бракович А.И.

Минск 2018

**Цель работы:** освоить общие принципы решения задач методом динамического программирования, сравнить полученные решения задач с рекурсивным методом.

**Ход выполнения**

**Задание 1**

**Сгенерировать две строки размерами 300 и 250 символов:**

|  |  |
| --- | --- |
| 4lab.cpp |  |
| #include "stdafx.h"  #include <iostream>  #include <ctime>  using namespace std;  int main()  {  setlocale(LC\_ALL, "RUS");  cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Генерируем строки S1, S2\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << endl << endl;  char \*S1 = Gen(300);  char \*S2 = Gen(250);  cout << "Строка S1: " << endl;  for (int i = 0; i < 300; i++) cout << S1[i] << " ";  cout << endl << endl;  cout << "Строка S2: " << endl;  for (int i = 0; i < 250; i++) cout << S2[i] << " ";  cout << endl << endl;  return 0;  } | char \* Gen(int size)  {  char \*str = (char\*)malloc(sizeof(char)\*size);  for (int i = 0; i < size; i++) str[i] = rand() % 26 + 'a';  return str;  } |

Результат:



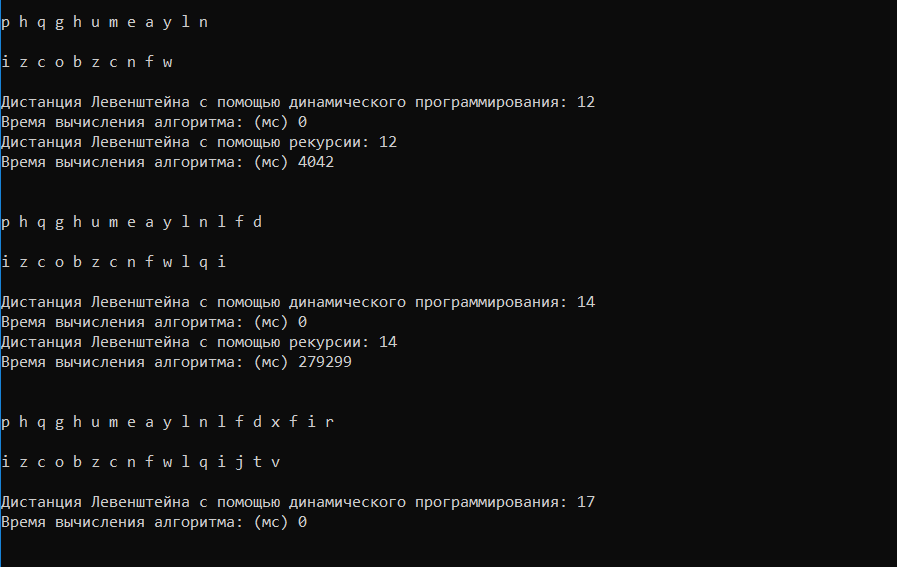
**Задание 2**

**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).**

|  |  |
| --- | --- |
| 4lab\_2.cpp | |
| cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Дистанция Левенштейна\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << endl << endl;  int dist = 0;  clock\_t t1 = 0, t2 = 0;  const char Lom[] = { 'Л', 'о', 'м' };  const char Gomon[] = { 'Г', 'о', 'м', 'о', 'н' };  for (int i = 0; i < 3; i++) cout << Lom[i] << " "; cout << endl;  for (int i = 0; i < 5; i++) cout << Gomon[i] << " "; cout << endl;  dist = levenshtein\_r(3, Lom, 5, Gomon);  cout << "Дистанция Левенштейна для Лом и Гомон: " << dist << endl << endl;  // system("pause");  float k[] = { 0.04, 0.05, 0.0625, 0.1, 0.2, 0.5, 1 };  for (int i = 0; i < 300 \* k[0]; i++) cout << S1[i] << " ";  cout << endl << endl;  for (int i = 0; i < 250 \* k[0]; i++) cout << S2[i] << " ";  t1 = clock();  dist = levenshtein(300 \* k[0], S1, 250 \* k[0], S2);  t2 = clock();  cout << endl << endl;  cout << "Дистанция Левенштейна с помощью динамического программирования: " << dist << endl;  cout << "Время вычисления алгоритма: (сек) " << t2 - t1 << endl;  t1 = clock();  dist = levenshtein\_r(300 \* k[0], S1, 250 \* k[0], S2);  t2 = clock();  cout << "Дистанция Левенштейна с помощью рекурсии: " << dist << endl;  cout << "Время вычисления алгоритма: (сек) " << t2 - t1 << endl;  // system("pause");  cout << endl << endl;  for (int i = 0; i < 300 \* k[3]; i++) cout << S1[i] << " ";  cout << endl << endl;  for (int i = 0; i < 250 \* k[3]; i++) cout << S2[i] << " ";  t1 = clock();  dist = levenshtein(300 \* k[3], S1, 250 \* k[3], S2);  t2 = clock();  cout << endl << endl;  cout << "Дистанция Левенштейна с помощью динамического программирования: " << dist << endl;  cout << "Время вычисления алгоритма: (сек) " << t2 - t1 << endl;  t1 = clock();  dist = levenshtein\_r(300 \* k[3], S1, 250 \* k[3], S2);  t2 = clock();  cout << "Дистанция Левенштейна с помощью рекурсии: " << dist << endl;  cout << "Время вычисления алгоритма: (сек) " << t2 - t1 << endl;  //system("pause");  cout << endl << endl;  for (int i = 0; i < 300 \* k[4]; i++) cout << S1[i] << " ";  cout << endl << endl;  for (int i = 0; i < 250 \* k[4]; i++) cout << S2[i] << " ";  t1 = clock();  dist = levenshtein(300 \* k[4], S1, 250 \* k[4], S2);  t2 = clock();  cout << endl << endl;  cout << "Дистанция Левенштейна с помощью динамического программирования: " << dist << endl;  cout << "Время вычисления алгоритма: (сек) " << t2 - t1 << endl;  t1 = clock();  dist = levenshtein\_r(300 \* k[4], S1, 250 \* k[4], S2);  t2 = clock();  cout << "Дистанция Левенштейна с помощью рекурсии: " << dist << endl;  cout << "Время вычисления алгоритма: (сек) " << t2 - t1 << endl;  //system("pause");  cout << endl << endl;  for (int i = 0; i < 300 \* k[5]; i++) cout << S1[i] << " ";  cout << endl << endl;  for (int i = 0; i < 250 \* k[5]; i++) cout << S2[i] << " ";  t1 = clock();  dist = levenshtein(300 \* k[5], S1, 250 \* k[5], S2);  t2 = clock();  cout << endl << endl;  cout << "Дистанция Левенштейна с помощью динамического программирования: " << dist << endl; | cout  for (int i = 0; i < 300 \* k[1]; i++) cout << S1[i] << " ";  cout << endl << endl;  for (int i = 0; i < 250 \* k[1]; i++) cout << S2[i] << " ";  t1 = clock();  dist = levenshtein(300 \* k[1], S1, 250 \* k[1], S2);  t2 = clock();  cout << endl << endl;  cout << "Дистанция Левенштейна с помощью динамического программирования: " << dist << endl;  cout << "Время вычисления алгоритма: (сек) " << t2 - t1 << endl;  t1 = clock();  dist = levenshtein\_r(300 \* k[1], S1, 250 \* k[1], S2);  t2 = clock();  cout << "Дистанция Левенштейна с помощью рекурсии: " << dist << endl;  cout << "Время вычисления алгоритма: (сек) " << t2 - t1 << endl;  // system("pause");  cout << endl << endl;  for (int i = 0; i < 300 \* k[2]; i++) cout << S1[i] << " ";  cout << endl << endl;  for (int i = 0; i < 250 \* k[2]; i++) cout << S2[i] << " ";  t1 = clock();  dist = levenshtein(300 \* k[2], S1, 250 \* k[2], S2);  t2 = clock();  cout << endl << endl;  cout << "Дистанция Левенштейна с помощью динамического программирования: " << dist << endl;  cout << "Время вычисления алгоритма: (сек) " << t2 - t1 << endl;  t1 = clock();  dist = levenshtein\_r(300 \* k[2], S1, 250 \* k[2], S2);  t2 = clock();  cout << "Дистанция Левенштейна с помощью рекурсии: " << dist << endl;  cout << "Время вычисления алгоритма: (сек) " << t2 - t1 << endl;  // system("pause");  cout << endl << endl;  тр  cout << "Время вычисления алгоритма: (сек) " << t2 - t1 << endl;  t1 = clock();  dist = levenshtein\_r(300 \* k[5], S1, 250 \* k[5], S2);  t2 = clock();  cout << "Дистанция Левенштейна с помощью рекурсии: " << dist << endl;  cout << "Время вычисления алгоритма: (сек) " << t2 - t1 << endl;  //system("pause");  cout << endl << endl;  for (int i = 0; i < 300 \* k[6]; i++) cout << S1[i] << " ";  cout << endl << endl;  for (int i = 0; i < 250 \* k[6]; i++) cout << S2[i] << " ";  t1 = clock();  dist = levenshtein(300 \* k[6], S1, 250 \* k[6], S2);  t2 = clock();  cout << endl << endl;  cout << "Дистанция Левенштейна с помощью динамического программирования: " << dist << endl;  cout << "Время вычисления алгоритма: (сек) " << t2 - t1 << endl;  t1 = clock();  dist = levenshtein\_r(300 \* k[6], S1, 250 \* k[6], S2);  t2 = clock();  cout << "Дистанция Левенштейна с помощью рекурсии: " << dist << endl;  cout << "Время вычисления алгоритма: (сек)" << t2 - t1 << endl;  //system("pause");  return 0;  } |

|  |  |
| --- | --- |
| Levenshtein.cpp | Levenshtein.h |
| #include "stdafx.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); //минимальное из трех  }  int levenshtein(int lx, const char x[], int ly, const char y[]) //длина слова х, слово длиной lx  //длина слова y, слово длиной ly  {  int \*d = new int[(lx + 1)\*(ly + 1)];  for (int i = 0; i <= lx; i++) DD(i, 0) = i; //длина строки х=i (кол-во символов в слове x), а DD возвращ. саму строку  for (int j = 0; j <= ly; j++) DD(0, j) = j; //длина строки y=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); //возвращает длину слова и строки  }  int levenshtein\_r( //рекурсия  int lx, const char x[],  int ly, const char y[]  )  {  int rc = 0; //результат, дистанция  if (lx == 0) rc = ly; //елси длина x=0, то дистанция = длине слова y  else if (ly == 0) rc = lx;  else if (lx == 1 && ly == 1 && x[0] == y[0]) rc = 0; //если длина x = длина y = 1 и первые буквы совапали, дистанция = 0  else if (lx == 1 && ly == 1 && x[0] != y[0]) rc = 1; //если длина x = длина y = 1 и первые буквы не совапали, дистанция = 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;  }; | #pragma once  int levenshtein( // ДИНАМИЧЕСКОЕ ПРОГРАММИРОВАНИЕ  int lx, // длина слова x  const char x[], // слово длиной lx  int ly, // длина слова y  const char y[] // слово y  );  int levenshtein\_r( // РЕКУРСИЯ  int lx, // длина строки x  const char x[], // строка длиной lx  int ly, // длина строки y  const char y[] // строка y  ); |

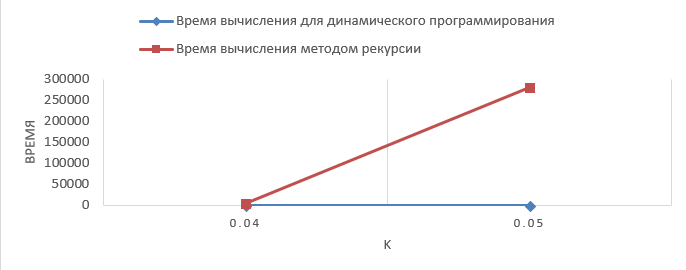
Пример выполнения:



**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).**

Метод динамического программирования значительно эффективнее рекурсивного метода, т.к. выполняется намного быстрее.



Как видно из задания номер 2 и графиков, при больших значениях k, а соответственно, при небольшой длине строк, метод динамического программирования является выигрышным вариантом по сравнению с методом рекурсии. Это происходит по той причине, что в методе ДП мы должны рассмотреть полиноминальное количество вариантов, пока не найдем решение, а в методе рекурсии перебор является экспоненциальным. При кол-ве букв 60 и 50 рекурсивный метод проработал всю ночь, но так и не выдал решения.

**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).

Найти расстояние Левенштейна между словами «ЛОМ» и «ГОМОН».

1.  
2. 
3. 
4. 
5. 
6. 

1. 
2. 
3. 
4. 

1. 

1. 



1. 

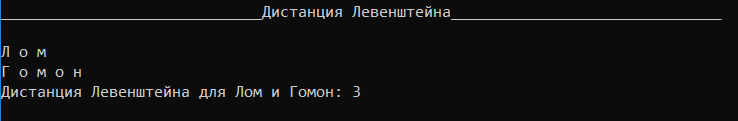
1. 

1. 
2. 
3. 
4. 
5. 
6. 
7. 
8. 
9. 
10. 
11. 
12. 
13. 
14. 

Дистанция Левенштейна для слов «ЛОМ» и «ГОМОН»: 3.

|  |
| --- |
| 4lab\_2.cpp |
| const char Lom[] = { 'Л', 'о', 'м' };  const char Gomon[] = { 'Г', 'о', 'м', 'о', 'н' };  for (int i = 0; i < 3; i++) cout << Lom[i] << " "; cout << endl;  for (int i = 0; i < 5; i++) cout << Gomon[i] << " "; cout << endl;  dist = levenshtein\_r(3, Lom, 5, Gomon);  cout << "Дистанция Левенштейна для Лом и Гомон: " << dist << endl << endl; |



**Задание 5**

**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.**

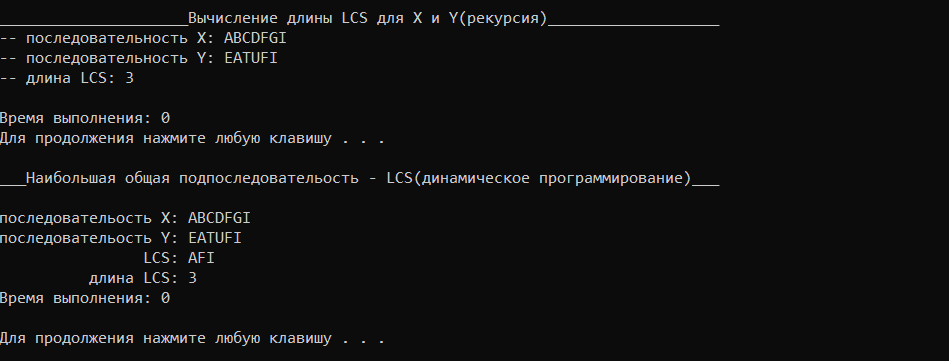
Даны две последовательности: ABCDFGI и EATUFI.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | | **A** | | **B** | | **C** | | **D** | | **F** | | **G** | | **I** | |
|  | 0 | | 0 | | 0 | | 0 | | 0 | | 0 | | 0 | | 0 | |
| **E** | 0 | | 0 | | 0 | | 0 | | 0 | | 0 | | 0 | | 0 | |
| **A** | 0 | | 1 | | 1 | | 1 | | 1 | | 1 | | 1 | | 1 | |
| **T** | 0 | | 1 | | 1 | | 1 | | 1 | | 1 | | 1 | | 1 | |
| **U** | 0 | | 1 | | 1 | | 1 | | 1 | | 1 | | 1 | | 1 | |
| **F** | 0 | | 1 | | 1 | | 1 | | 1 | | 2 | | 2 | | 2 | |
| **I** | 0 | | 1 | | 1 | | 1 | | 1 | | 2 | | 2 | | 3 | |
|  | |  | | **A** | | **B** | | **C** | | **D** | | **F** | | **G** | | **I** | |
|  | |  | |  | |  | |  | |  | |  | |  | |  | |
| **E** | |  | | ↑ | | ↑ | | ↑ | | ↑ | | ↑ | | ↑ | | ↑ | |
| **A** | |  | | ↖ | | ← | | ← | | ← | | ← | | ← | | ← | |
| **T** | |  | | ↑ | | ↖ | | ← | | ← | | ← | | ← | | ← | |
| **U** | |  | | ↑ | | ↑ | | ↑ | | ← | | ← | | ← | | ← | |
| **F** | |  | | ↑ | | ↑ | | ↑ | | ↑ | | ↖ | | ← | | ← | |
| **I** | |  | | ↑ | | ↑ | | ↑ | | ↑ | | ↑ | | ↑ | | ↖ | |

НАИБОЛЬШАЯ ОБЩАЯ ПОДПОСЛЕДОВАТЕЛЬНОСТЬ: **AFI**

Проверка рекурсивным и динамичным алгоритмами на C++:

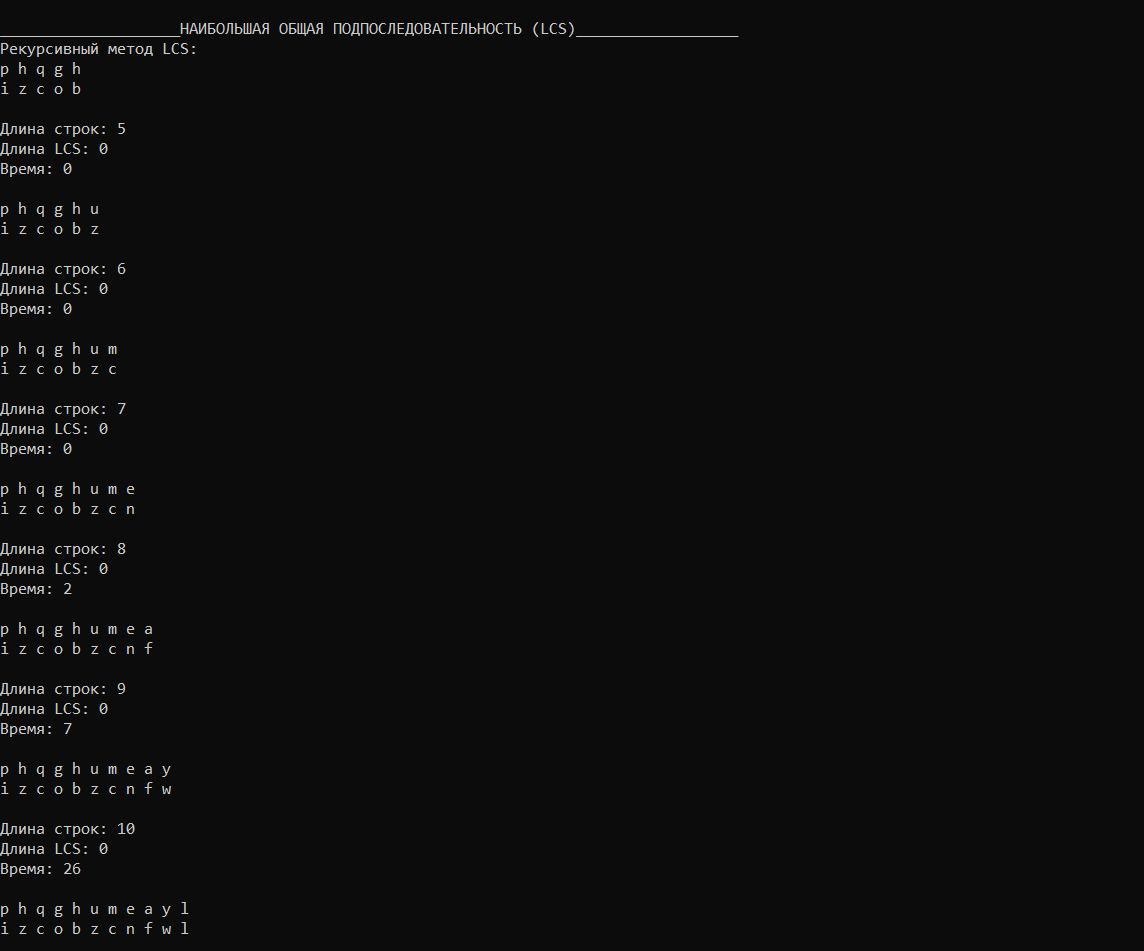
|  |
| --- |
| lab4\_55.cpp |
| cout << endl << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Вычисление длины LCS для X и Y(рекурсия)\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_";  char X[] = "ABCDFGI", Y[] = "EATUFI";  cout << endl << "-- последовательность X: " << X;  cout << endl << "-- последовательность Y: " << Y;  t1 = clock();  int s = lcs(  sizeof(X) - 1, // длина последовательности X  "ABCDFGI", // последовательность X  sizeof(Y) - 1, // длина последовательности Y  "EATUFI" // последовательность Y  );  t2 = clock();  cout << endl << "-- длина LCS: " << s << endl;  cout << endl << "Время выполнения: " << t2 - t1 << endl;  system("pause");  char x[] = "ABCDFGI",  y[] = "EATUFI";  t1 = clock();  l = lcsd(x, y, z);  t2 = clock();  cout << endl  << "\_\_\_Наибольшая общая подпоследовательость - LCS(динамическое программирование)\_\_\_" << endl;  cout << endl << "последовательость X: " << x;  cout << endl << "последовательость Y: " << y;  cout << endl << " LCS: " << z;  cout << endl << " длина LCS: " << l;  cout << endl << "Время выполнения: " << t2 - t1 << endl;  cout << endl;  return 0; |



Исследуем зависимость времени работы алгоритма от количества символов в строках:

|  |
| --- |
| lab4\_55.cpp |
| cout << endl << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_НАИБОЛЬШАЯ ОБЩАЯ ПОДПОСЛЕДОВАТЕЛЬНОСТЬ (LCS)\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_";  int res = 0, m;  cout << endl << "Рекурсивный метод LCS: " << endl;  for (m = 5; m < 15; m++)  {  for (int i = 0; i < m; i++) cout << S1[i] << " ";  cout << endl;  for (int i = 0; i < m; i++) cout << S2[i] << " ";  cout << endl << endl;  t1 = clock();  res = lcs(m, S1, m, S2);  t2 = clock();  cout << "Длина строк: " << k << endl << "Длина LCS: " << res << endl << "Время: " << t2 - t1 << endl << endl;  }  cout << "Динамическое программирование" << endl;  char z[100] = "";  t1 = clock();  int l = lcsd(S1, S2, z);  t2 = clock();  cout << endl << "Длина LCS: " << l;  cout << endl << "Время выполнения: " << t2 - t1 << endl; |

Вывод на консоли:







**Вывод:** Метод динамического программирования справляется со строками длиной 300 и 250 символов за 97 единиц тактового времени, в то время как рекурсивный метод уже на 14 символах считает примерно 3 секунды (3367 единиц тактового времени). Таким образом для решения задачи о нахождении наибольшей общей подпоследовательности динамическое программирование будет более эффективным методом, чем рекурсивный метод.