**Лабораторная работа 4**

**ДИНАМИЧЕСКОЕ ПРОГРАММИРОВАНИЕ**

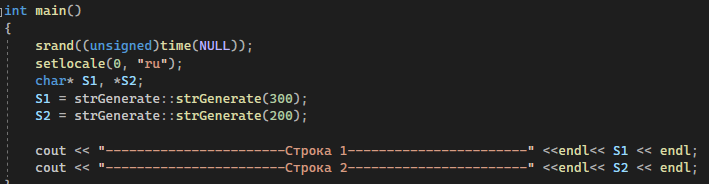
**ЦЕЛЬ РАБОТЫ:** освоить общие принципы решения задач методом динамического программирования, сравнить полученные решения задач с рекурсивным методом.

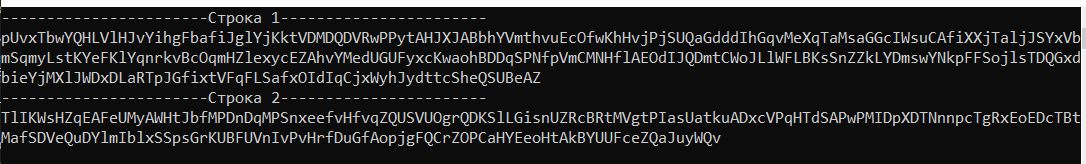
***Задание 1.***

Генератор случайных строк:



Генерирование строк S1(300 символов) и S2(200 символов):

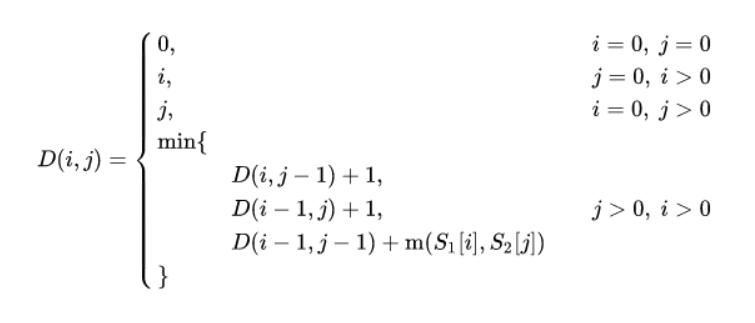




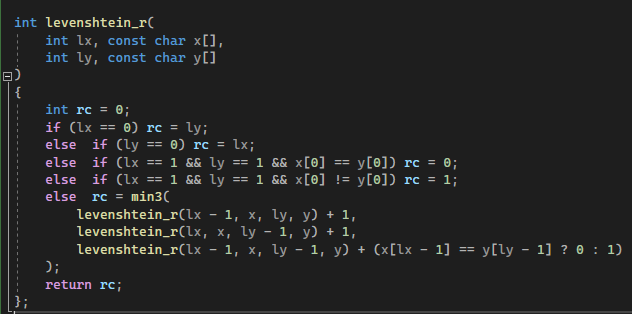
***Задание 2.***

Расстояние Левенштейна (редакционное расстояние, дистанция редактирования) — метрика, измеряющая по модулю разность между двумя последовательностями символов. Это минимальное число односимвольных преобразований (удаления, вставки или замены), необходимых, чтобы превратить одну последовательность в другую.

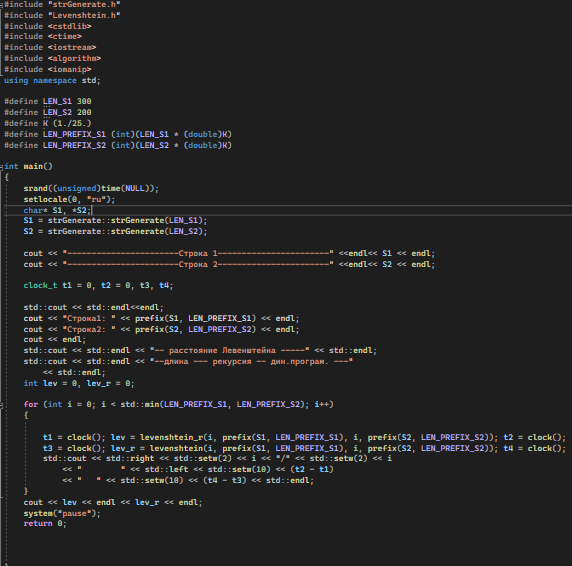
Вычисление каждого преобразования строки вычисляется по формуле Вагнера — Фишера:



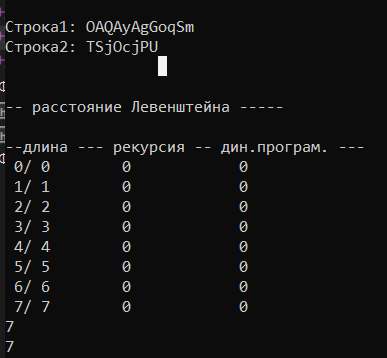
**Вычисление расстояния Левенштейна для строк S1 S2**



Реализация рекурсивного метода вычисления расстояния Левенштейна

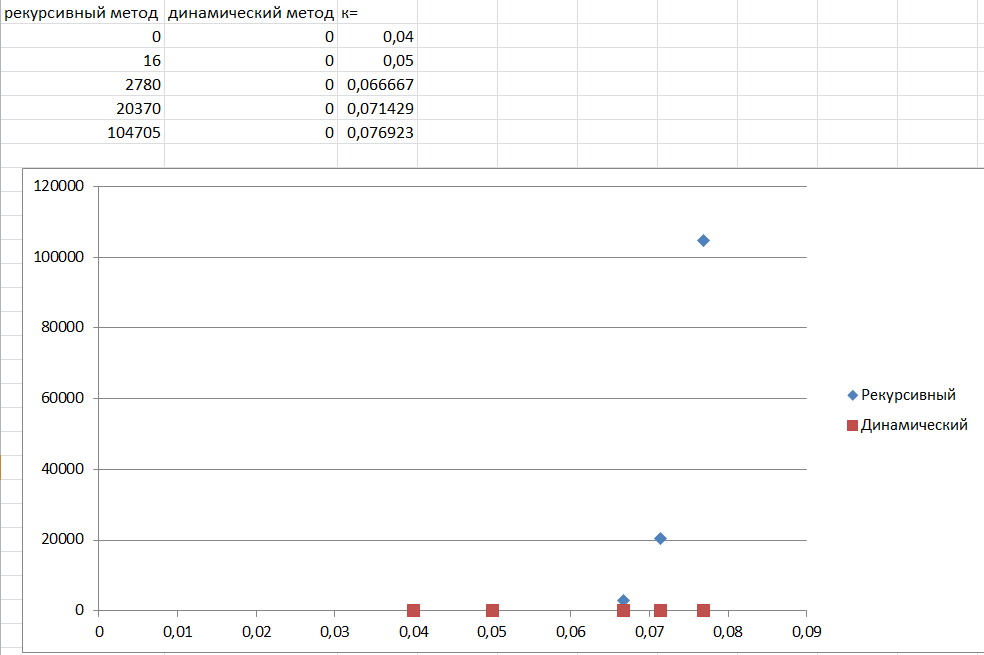


Вычисление



Результат для k = 1/25

***Задание 3.***



Из графиков видно, что рекурсивный метод крайне неэффективен, по сравнению с методом динамического программирования.

***Задание 4.***

Вариант 11 лот полет

***Задание 5.***

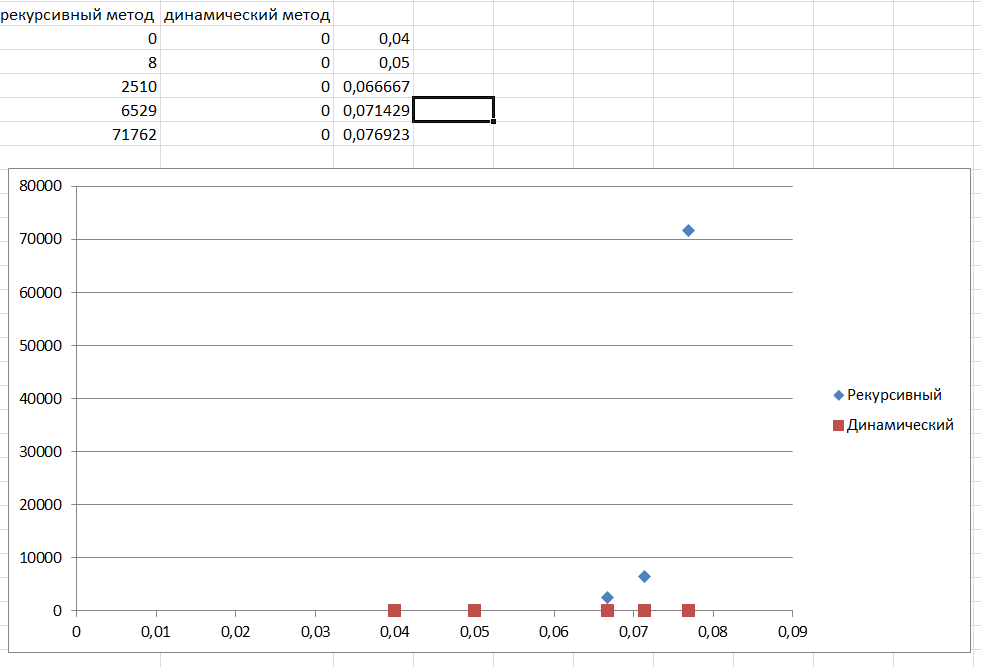
Рекурсия

|  |
| --- |
| #include "strGenerate.h"  #include "Levenshtein.h"  #include <cstdlib>  #include <ctime>  #include <iostream>  #include <algorithm>  #include <iomanip>  using namespace std;  //  #define LEN\_S1 300  #define LEN\_S2 200  #define K (1./20.)  #define LEN\_PREFIX\_S1 (int)(LEN\_S1 \* (double)K)  #define LEN\_PREFIX\_S2 (int)(LEN\_S2 \* (double)K)  #include <iostream>  #include "LCS.h"  int main()  {  srand((unsigned)time(NULL));  setlocale(0, "ru");  char\* S1, \*S2;  S1 = strGenerate::strGenerate(LEN\_S1);  S2 = strGenerate::strGenerate(LEN\_S2);    cout << "-----------------------Строка 1-----------------------" <<endl<< S1 << endl;  cout << "-----------------------Строка 2-----------------------" <<endl<< S2 << endl;    std::cout << std::endl << "-- вычисление длины LCS для X и Y(рекурсия)\n";  cout << "Строка1: " << prefix(S1, LEN\_PREFIX\_S1) << endl;  cout << "Строка2: " << prefix(S2, LEN\_PREFIX\_S2) << endl;  clock\_t t1 = 0, t2 = 0;  t1 = clock();  int s = lcs(  LEN\_PREFIX\_S1, // длина последовательности X  prefix(S1, LEN\_PREFIX\_S1), // последовательность X  LEN\_PREFIX\_S2 - 1, // длина последовательности Y  prefix(S2, LEN\_PREFIX\_S2) // последовательность Y  );  t2 = clock();  std::cout << std::endl << "-- длина LCS: " << s << std::endl;  cout << "Время: " << t2 - t1 << endl;  system("pause");  return 0;  } |

Динамическое программирование

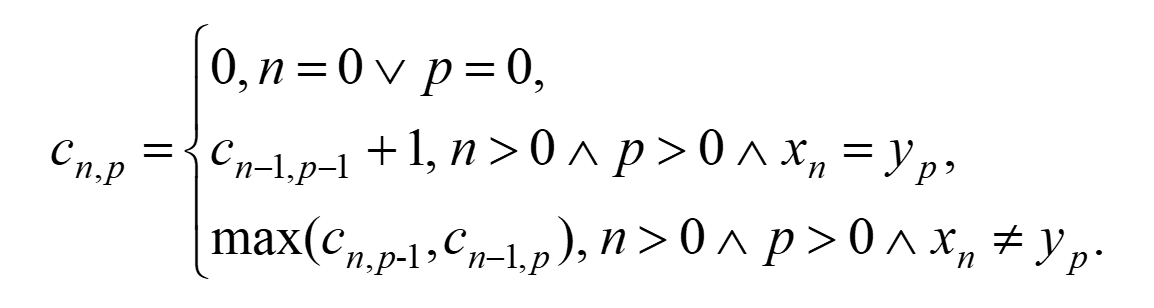
|  |
| --- |
| #include "strGenerate.h"  #include "Levenshtein.h"  #include <cstdlib>  #include <ctime>  #include <iostream>  #include <algorithm>  #include <iomanip>  using namespace std;  //  #define LEN\_S1 300  #define LEN\_S2 200  #define K (1./20.)  #define LEN\_PREFIX\_S1 (int)(LEN\_S1 \* (double)K)  #define LEN\_PREFIX\_S2 (int)(LEN\_S2 \* (double)K)  #include <iostream>  #include "LCH.h"  int main()  {  srand((unsigned)time(NULL));  setlocale(0, "ru");  char\* S1, \*S2;  S1 = strGenerate::strGenerate(LEN\_S1);  S2 = strGenerate::strGenerate(LEN\_S2);    cout << "-----------------------Строка 1-----------------------" <<endl<< S1 << endl;  cout << "-----------------------Строка 2-----------------------" <<endl<< S2 << endl;    std::cout << std::endl << "-- вычисление длины LCS для X и Y(динамичаское программирование )\n";  cout << "Строка1: " << prefix(S1, LEN\_PREFIX\_S1) << endl;  cout << "Строка2: " << prefix(S2, LEN\_PREFIX\_S2) << endl;  clock\_t t1 = 0, t2 = 0;  t1 = clock();  char z[100];  int s = lcsd(  prefix(S1, LEN\_PREFIX\_S1),  prefix(S2, LEN\_PREFIX\_S2),  z  );  t2 = clock();  std::cout << std::endl << "-- длина LCS: " << s << std::endl;  cout << "Время: " << t2 - t1 << endl;  system("pause");  return 0;  } |

График



Из графиков видно, что рекурсивный метод крайне неэффективен, по сравнению с методом динамического программирования.

Работа по вариантам



Вариант 11 TOUEXAZ HIEHXZ

