**Белорусский государственный технологический университет**

**Факультет информационных технологий**

**Специальность программная инженерия**

Отчёт по лабораторной работе №4

По дисциплине «Математическое программирование»

На тему «Динамическое программирование»

Выполнил:

Студент 2 курса 8 группы

Мамонько Денис Александрович

Преподаватель: асс. Ромыш А.С.

2025, Минск

**Вариант 7**

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

***Задание 1.***

На языке С++ сгенерировать случайным образом строку букв латинского алфавита S1 длиной 300 символов и S2 длиной 200.

|  |
| --- |
| #include <iostream>  #include <ctime>  using namespace std;  char\* generateRandomString(int size) {  char\* str = (char\*)malloc(sizeof(char) \* size);  for (int i = 0; i < size; i++) {  str[i] = rand() % 26 + 'a';  }  return str;  }  int main()  {  setlocale(LC\_ALL, "RU");  char\* str1 = generateRandomString(300);  cout << "Первая строка: ";  for (int i = 0; i < 300; i++) {  if (i % 50 == 0) {  cout << endl;  }  cout << str1[i];  }  cout << endl;  char\* str2 = generateRandomString(200);  cout << "Вторая строка: ";  for (int i = 0; i < 200; i++) {  if (i % 50 == 0) {  cout << endl;  }  cout << str2[i];  }  } |

Листинг 1.1 – Листинг firstTask.cpp

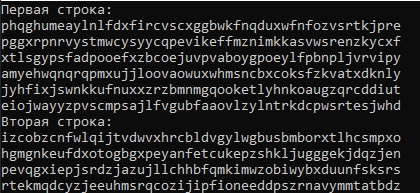
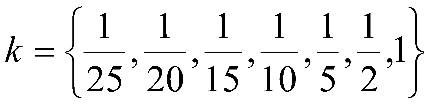


Рисунок 1 – Результат программы

***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).

Сначала разберем пример динамического программирования.

|  |
| --- |
| #pragma once  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 2.1 – Файл 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 2.2 – Файл Levenshtein.cpp

|  |
| --- |
| #include <algorithm>  #include <iostream>  #include <ctime>  #include <iomanip>  #include "Levenshtein.h"  int main()  {  setlocale(LC\_ALL, "rus");  clock\_t t1 = 0, t2 = 0, t3, t4;  char x[] = "abcdefghklmnoxm", y[] = "xyabcdefghomnkm";  int lx = sizeof(x) - 1, ly = sizeof(y) - 1;  std::cout << std::endl;  std::cout << std::endl << "-- расстояние Левенштейна -----" << std::endl;  std::cout << std::endl << "--длина --- рекурсия -- дин.програм. ---"  << std::endl;  for (int i = 8; i < std::min(lx, ly); i++)  {  t1 = clock(); levenshtein\_r(i, x, i - 2, y); t2 = clock();  t3 = clock(); levenshtein(i, x, i - 2, y); t4 = clock();  std::cout << std::right << std::setw(2) << i - 2 << "/" << std::setw(2) << i  << " " << std::left << std::setw(10) << (t2 - t1)  << " " << std::setw(10) << (t4 - t3) << std::endl;  }  system("pause");  return 0;  } |

Листинг 2.3 – Файл secondTask.cpp

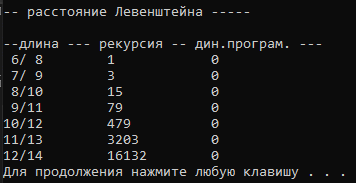


Рисунок 2 – Результат работы программы

Теперь рассмотрим пример исключительно рекурсивного алгоритма, а затем сравним результаты.

|  |
| --- |
| #include <iostream>  #include <ctime>  #include <iomanip>  #include <algorithm>  int min3(int x1, int x2, int x3) {  return std::min(std::min(x1, x2), x3);  }  int levenshtein\_recursive(int lx, const char x[], int ly, const char y[]) {  if (lx == 0) return ly;  if (ly == 0) return lx;  int cost = (x[lx - 1] == y[ly - 1]) ? 0 : 1;  return min3(  levenshtein\_recursive(lx - 1, x, ly, y) + 1, // удаление  levenshtein\_recursive(lx, x, ly - 1, y) + 1, // вставка  levenshtein\_recursive(lx - 1, x, ly - 1, y) + cost // замена  );  }  int main() {  setlocale(LC\_ALL, "rus");  clock\_t t1, t2;  char x[] = "abcdefghklmnoxm", y[] = "xyabcdefghomnkm";  int lx = sizeof(x) - 1, ly = sizeof(y) - 1;  std::cout << "\n-- расстояние Левенштейна (рекурсия) --\n";  for (int i = 8; i < std::min(lx, ly); i++) {  t1 = clock();  int distance = levenshtein\_recursive(i, x, i - 2, y);  t2 = clock();  std::cout << "Длина " << i - 2 << "/" << i << " Время: " << (t2 - t1) << " мс" << std::endl;  }  system("pause");  return 0;  } |

Листинг 2.4 – Файл secondTask.cpp

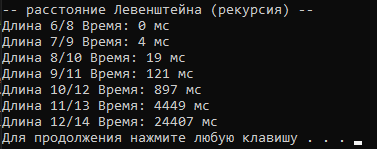


Рисунок 3 – Результат работы программы

***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).

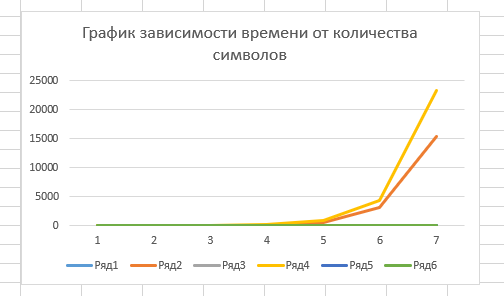


Рисунок 4 – График зависимости времени от количества символов

На графике, который представлен выше, можно сдедать вывод, что алгоритм, вычисленный способом динамического программирования, является более оптимальным и вычисления проходят в разы быстрее, чем рекурсивным алгоритмом.

***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).



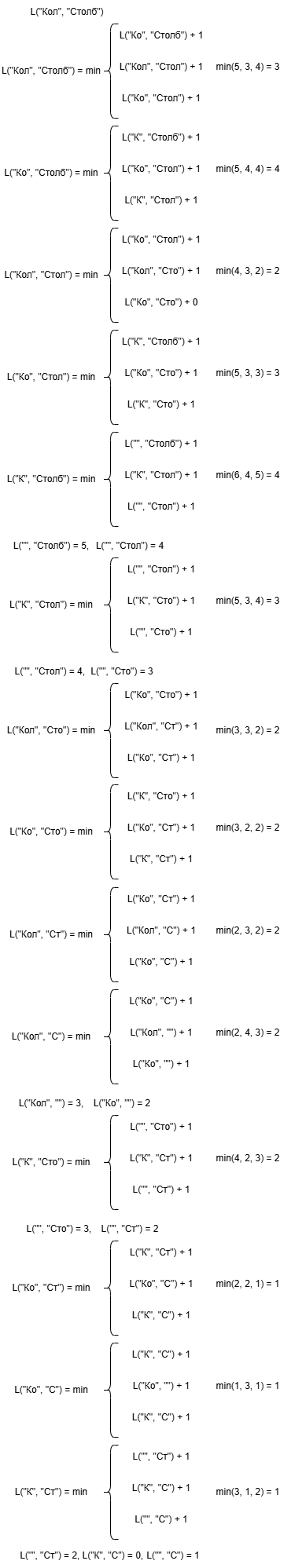


Рисунок 5 – Ручной пример вычисления дистанции Левенштейна

***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.



|  |
| --- |
| #pragma once  int lcs(int lenx, // длина последовательности X  const char x[], // последовательность X  int leny, // длина последовательности Y  const char y[] // последовательность Y  );  // -- динамическое вычисление LCS  int lcsd(const char x[], // последовательность X  const char y[], // последовательность Y  char z[] // наибольшая общая подпоследовательность  ); |

Листинг 5.1 – Файл LCS.h

|  |
| --- |
| #include <algorithm>  #include "LCS.h"  int lcs(int lenx, const char x[],  int leny, const char y[])  {  int rc = 0;  if (lenx > 0 && leny > 0)  {  if (x[lenx - 1] == y[leny - 1]) rc = 1 + lcs(lenx - 1, x, leny - 1, y);  else rc = std::max(lcs(lenx, x, leny - 1, y), lcs(lenx - 1, x, leny, y));  }  return rc; //длина LCS  }  #define LCS\_C(x1,x2) (C[(x1)\*(leny+1)+(x2)])  #define LCS\_B(x1,x2) (B[(x1)\*(leny+1)+(x2)])  #define LCS\_X(i) (x[(i)-1])  #define LCS\_Y(i) (y[(i)-1])  #define LCS\_Z(i) (z[(i)-1])  enum Dart { TOP, LEFT, LEFTTOP };  void getLCScontent(int lenx, int leny, const char x[],  const Dart\* B,  int n, int i, int j, char z[])  {  if ((i > 0 && j > 0 && n > 0))  {  if (LCS\_B(i, j) == LEFTTOP)  {  getLCScontent(lenx, leny, x, B, n - 1, i - 1, j - 1, z);  LCS\_Z(n) = LCS\_X(i);  LCS\_Z(n + 1) = 0;  }  else if (LCS\_B(i, j) == TOP)  getLCScontent(lenx, leny, x, B, n, i - 1, j, z);  else getLCScontent(lenx, leny, x, B, n, i, j - 1, z);  }  };  int lcsd(const char x[], const char y[], char z[])  {  int n;  int lenx = strlen(x), leny = strlen(x),  \* C = new int[(lenx + 1) \* (leny + 1)];  Dart\* B = new Dart[(lenx + 1) \* (leny + 1)];  memset(C, 0, sizeof(int) \* (lenx + 1) \* (leny + 1));  for (int i = 1; i <= lenx; i++)  for (int j = 1; j <= leny; j++)  if (LCS\_X(i) == LCS\_Y(j))  {  LCS\_C(i, j) = LCS\_C(i - 1, j - 1) + 1;  LCS\_B(i, j) = LEFTTOP;  }  else if (LCS\_C(i - 1, j) >= LCS\_C(i, j - 1))  {  LCS\_C(i, j) = LCS\_C(i - 1, j);  LCS\_B(i, j) = TOP;  }  else  {  LCS\_C(i, j) = LCS\_C(i, j - 1);  LCS\_B(i, j) = LEFT;  }  getLCScontent(lenx, leny, x, B, LCS\_C(lenx, leny), lenx, leny, z);  return LCS\_C(lenx, leny);  }  #undef LCS\_Z  #undef LCS\_C  #undef LCS\_B  #undef LCS\_X  #undef LCS\_Y |

Листинг 5.2 – Файл LCS.cpp

|  |
| --- |
| #include <iostream>  #include "LCS.h"  #include <Windows.h>  #include <ctime>  int main()  {  SetConsoleCP(1251);  SetConsoleOutputCP(1251);  clock\_t t1 = 0, t2 = 0, t3 = 0, t4 = 0;  char z[100] = "";  t1 = clock();  char X[] = "QVTWNHO";  char Y[] = "RQTWYK";  std::cout << std::endl << "-- вычисление длины LCS для X и Y(рекурсия)";  std::cout << std::endl << "-- последовательность X: " << X;  std::cout << std::endl << "-- последовательность Y: " << Y;  int s = lcs(sizeof(X) - 1, "QVTWNHO", sizeof(Y) - 1, "RQTWYK");  std::cout << std::endl << "-- длина LCS: " << s << std::endl;  t2 = clock();  // наибольшая общая подпоследовательность  t3 = clock();  char x[] = "TOUEXAZ";  char y[] = "HIEHXZ";  int l = lcsd(x, y, z);  t4 = clock();  std::cout << std::endl  << "-- наибольшая общая подпоследовательость - LCS(динамическое"  << " программирование)" << std::endl;  std::cout << std::endl << "последовательость X: " << x;  std::cout << std::endl << "последовательость Y: " << y;  std::cout << std::endl << " LCS: " << z;  std::cout << std::endl << " длина LCS: " << l;  std::cout << std::endl;  std::cout << std::endl << "Время вычисления LCS";  std::cout << std::endl << "Рекурсия: " << (t2 - t1);  std::cout << std::endl << "Динамическое программирование: " << (t4 - t3) << std::endl;  return 0;  } |

Листинг 5.3 – Файл fifthTask.cpp

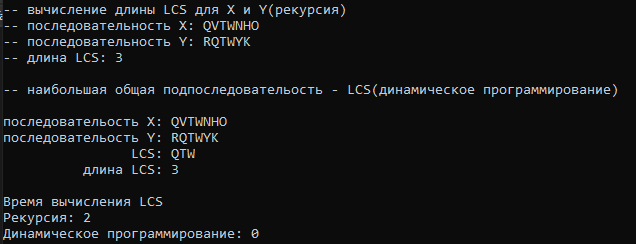


Рисунок 6 – Результат работы программы

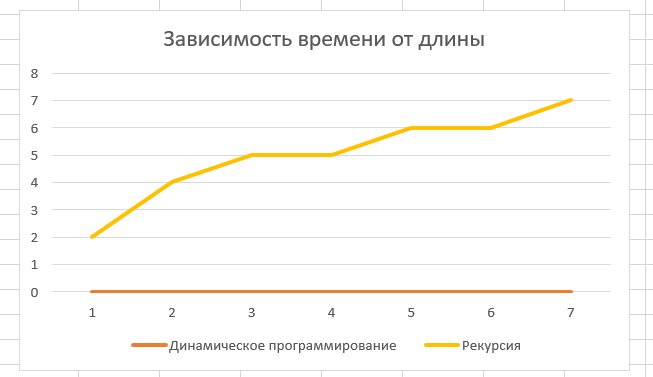


Рисунок 7 – График зависимости времени от длины

На рисунке 7 представлен график зависимости времени от длины последовательности. Сам рисунок представлен двумя графиками, где первый график это поиск общей подпоследовательности рекурсивным алгоритмом за определенное время, а второй график – это результат работы алгоритма, который работал по принципу динамического программирования. При построении графика, мы руководствовались принципом, когда посимвольно сравнивали строки, то есть каждый раз мы добавляли по символу и смотрели за какое время выполняется алгоритм. Данную операцию мы проделывали, пока не дошли до строки в исходном виде. Исходя из данного графика, можно сделать вывод, что поиск общей подпоследовательности лучше реализовывать с помощью динамического программирования, данный подход является более эффективным.

**Матричный способ:**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | Q | V | T | W | N | H | O |
|  | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| R | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Q | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| T | 0 | 1 | 1 | 2 | 2 | 2 | 2 | 2 |
| W | 0 | 1 | 1 | 2 | 3 | 3 | 3 | 3 |
| Y | 0 | 1 | 1 | 2 | 3 | 3 | 3 | 3 |
| K | 0 | 1 | 1 | 2 | 3 | 3 | 3 | 3 |

**Вывод:** динамический подход к решению задач позволяет выполнять их значительно быстрее, чем рекурсивный, особенно это будет заметно при решении задач с большим объёмом информации.