**Лабораторная работа 4**

**ДИНАМИЧЕСКОЕ ПРОГРАММИРОВАНИЕ**

**1.**

|  |
| --- |
| random\_device rd;  mt19937 gen(rd());  const char alphabet[] = "abcdefghijklmnopqrstuvwxyz";  uniform\_int\_distribution<> gg(0, 25);  string S1, S2;  for (int i = 0; i < 300; i++) {  S1 += alphabet[gg(gen)];  }  cout << "S1: " << S1 << "\n\n";  for (int i = 0; i < 200; i++) {  S2 += alphabet[gg(gen)];  }  cout << "S2: " << S2 << "\n\n"; |

Листинг 1 – генерация S1 и S2

2.

|  |
| --- |
| #include <algorithm>  #include <iostream>  #include <ctime>  #include <iomanip>  #define DD(i,j) d[(i)\*(ly+1)+(j)]  using namespace std;  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;  };  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 = 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(); cout<<levenshtein\_r(i, x, i - 2, y)<<" "; t2 = clock();  t3 = clock(); cout<<levenshtein(i, x, i - 2, y); t4 = clock();  std::cout << std::right << std::setw(3) << 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 – Вычисление расстояния Левенштейна и замер времени выполнения при каждом k

3.

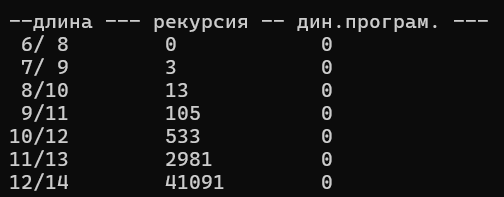


Рис.1 Результат выполнения программы

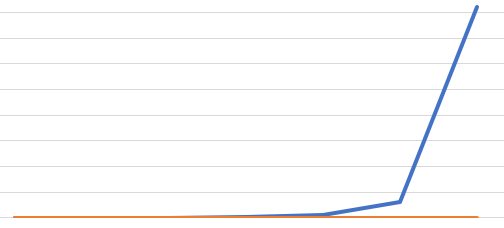


Рис.2 График зависимости

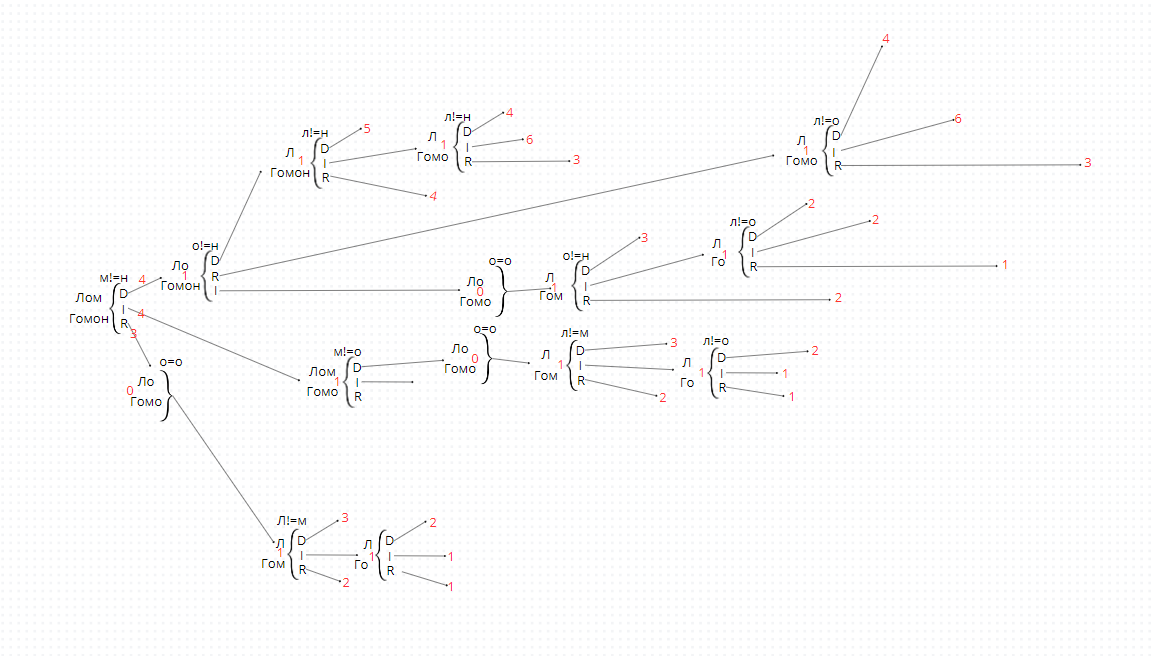
4.

Рис.3 Ручное вычисление расстояния Левенштейна

5.

|  |
| --- |
| #include <iostream>  #include <vector>  #include <algorithm>  #include <chrono>  using namespace std;  // Рекурсивный метод  int lcs\_recursive(int lenx, const string& x, int leny, const string& y) {  if (lenx == 0 || leny == 0)  return 0;  if (x[lenx - 1] == y[leny - 1])  return 1 + lcs\_recursive(lenx - 1, x, leny - 1, y);  else  return max(lcs\_recursive(lenx, x, leny - 1, y), lcs\_recursive(lenx - 1, x, leny, y));  }  // Динамическое программирование  int lcs\_dp(const string& x, const string& y) {  int lenx = x.size(), leny = y.size();  vector<vector<int>> dp(lenx + 1, vector<int>(leny + 1, 0));  for (int i = 1; i <= lenx; i++) {  for (int j = 1; j <= leny; j++) {  if (x[i - 1] == y[j - 1])  dp[i][j] = dp[i - 1][j - 1] + 1;  else  dp[i][j] = max(dp[i - 1][j], dp[i][j - 1]);  }  }  return dp[lenx][leny];  }  int main() {  string S1 = "ABCDFGI";  string S2 = "EATUFI";  int len1 = S1.size();  int len2 = S2.size();  vector<double> k\_values = { 1.0 / 25, 1.0 / 20, 1.0 / 15, 1.0 / 10, 1.0 / 5, 1.0 / 2, 1.0 };  for (double k : k\_values) {  int len\_prefix1 = max(1, static\_cast<int>(k \* len1));  int len\_prefix2 = max(1, static\_cast<int>(k \* len2));  string prefix1 = S1.substr(0, len\_prefix1);  string prefix2 = S2.substr(0, len\_prefix2);  // Динамическое программирование  auto start = chrono::high\_resolution\_clock::now();  int dp\_result = lcs\_dp(prefix1, prefix2);  auto end = chrono::high\_resolution\_clock::now();  chrono::duration<double> time\_dp = end - start;  cout << "DP (" << prefix1 << ", " << prefix2 << "): " << dp\_result << " | Time: " << time\_dp.count() << "s" << endl;  // Рекурсивный метод  start = chrono::high\_resolution\_clock::now();  int rec\_result = lcs\_recursive(prefix1.size(), prefix1, prefix2.size(), prefix2);  end = chrono::high\_resolution\_clock::now();  chrono::duration<double> time\_rec = end - start;  cout << "Recursive (" << prefix1 << ", " << prefix2 << "): " << rec\_result << " | Time: " << time\_rec.count() << "s" << endl;  cout << "------------------------------------------------------" << endl;  }  return 0;  } |

Листинг 3 – Код для нахождения длины наибольшей общей последовательности

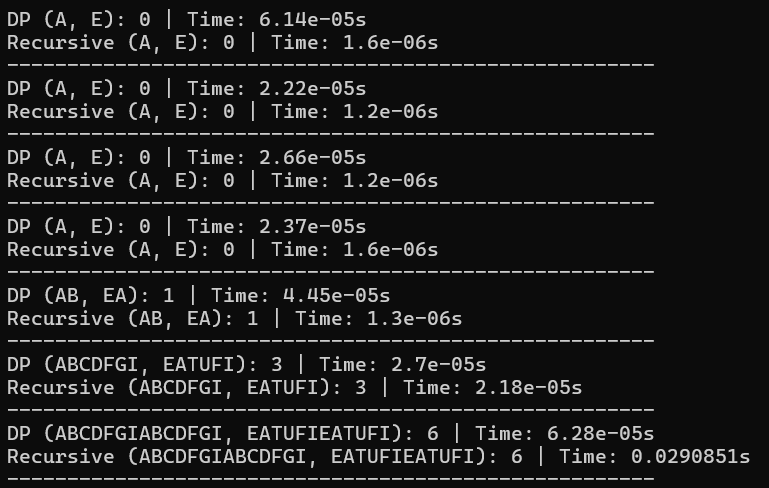


Рис.4 результат работы программы

