Учреждение образования

«Белорусский государственный технологический университет»

**Лабораторная работа №4**

Динамическое программирование

Выполнила:

Студентка 2 курса 8 группы ИТ

Казакова В.В.

2025 г.

**Лабораторная работа №4**

**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. Листинг кода представлен в листинге 1.1. Результат выполнения программы представлен на рисунке 1.1.

|  |
| --- |
| #include <iostream>  #include <windows.h>  #include <string>  #include <ctime>  using namespace std;  string generate(int countSymb) {  string str;  char symb;  for (int i = 0; i < countSymb; i++) {  //symb = 'a' + rand() % 26; // Генерируем случайную букву от 'a' до 'z'  symb = (rand() % 2 == 0) ? ('A' + rand() % 26) : ('a' + rand() % 26);  str += symb;  }  return str;  }  int main() {  SetConsoleCP(1251);  SetConsoleOutputCP(1251);  srand(time(NULL));  string s1 = generate(300);  string s2 = generate(200);  cout << "S1(300): " << s1 << endl;  cout << "S2(200): " << s2 << endl;  return 0;  } |

Листинг 1.1 – Генерация строк

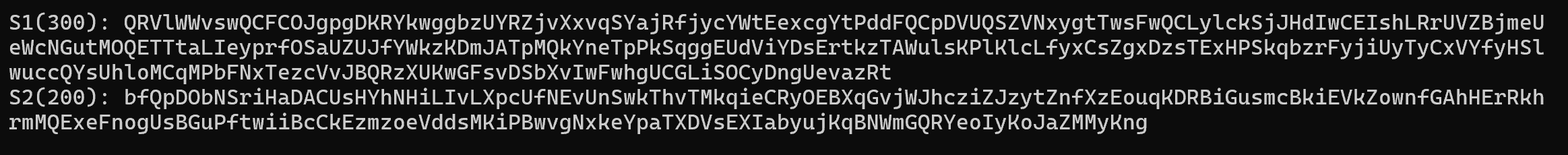


Рисунок 1.1 – Результат работы программы

**Задание 2.**

Вычислить двумя способами (рекурсивно и с помощью динамического программирования)  – Левенштейна для , где - длина строки ,  - строка, состоящая из первых  символов строки . (копии экрана и код вставить в отчет). Листинг кода представлен в листинге 2.1,2.2,2.3. Результат выполнения программы представлен на рисунке 2.1

|  |
| --- |
| // - Levenshtein.cpp  #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 2.1 – файл Levenshtein.cpp

|  |
| --- |
| #include <algorithm>  #include <iostream>  #include <ctime>  #include <iomanip>  #include <cstring>  #include "Levenshtein.h"  #include <Windows.h>  using namespace std;  char\* GenerateRandomString(int size)  {  char\* str = (char\*)malloc(sizeof(char) \* (size + 1));  for (int i = 0; i < size; i++) {  str[i] = rand() % 26 + 'a'; // 26 букв в алфавите  }  str[size] = '\0';  return str;  }  int main()  {  SetConsoleCP(1251);  SetConsoleOutputCP(1251);  const int threeHundred = 300;  const int twoHundred = 200;  char\* s1 = GenerateRandomString(threeHundred);  cout << "S1: " << endl;  for (int i = 0; i < threeHundred; i++) {  if (i % 50 == 0)  {  cout << "\n";  }  cout << s1[i];  }  cout << endl << endl;  srand(time(NULL) + 1);  char\* s2 = GenerateRandomString(twoHundred);  cout << "S2: " << endl;  for (int i = 0; i < twoHundred; i++) {  if (i % 50 == 0)  {  cout << "\n";  }  cout << s2[i];  }  cout << endl << endl;  clock\_t t1 = 0, t2 = 0, t3 = 0, t4 = 0;  int lx = strlen(s1);  int ly = strlen(s2);  int s1\_size[]{ threeHundred / 25, threeHundred / 20, threeHundred / 15, threeHundred / 10, threeHundred / 5, threeHundred / 2, threeHundred };  int s2\_size[]{ twoHundred / 25, twoHundred / 20, twoHundred / 15, twoHundred / 10, twoHundred / 5, twoHundred / 2, twoHundred };  cout << "\n\n-- расстояние Левенштейна -----";  cout << "\n\n--длина --- рекурсия -- дин.програм. ---\n";  for (int i = 0; i < min(sizeof(s1\_size) / sizeof(s1\_size[0]), sizeof(s2\_size) / sizeof(s2\_size[0])); i++)  {  t1 = clock();  levenshtein\_r(s1\_size[i], s1, s2\_size[i], s2);  t2 = clock();  t3 = clock();  levenshtein(s1\_size[i], s1, s2\_size[i], s2);  t4 = clock();  cout << right << setw(2) << s1\_size[i] << "/" << setw(2) << s2\_size[i]  << " " << left << setw(10) << (t2 - t1)  << " " << setw(10) << (t4 - t3) << endl;  }  system("pause");  return 0;  } |

Листинг 2.2 – файл Main.cpp

|  |
| --- |
| #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 2.3 – файл Levenshtein.h

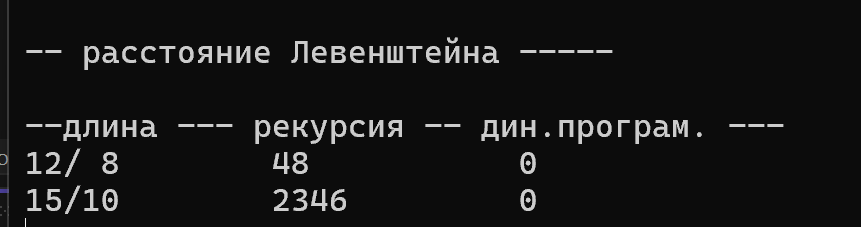


Рисунок 2.1 – Результат работы программы

**Задание 3.**

Выполнить сравнительный анализ времени затраченного на вычисление дистанции Левенштейна для двух методов решения. Построить графики зависимости времени вычисления от . На рисунке 3.1 представлены графики зависимости времени вычисления от *k*.

На графике, который изображён на рисунке 3.1, можно заметить, что использование динамического алгоритма во много раз эффективнее по затраченному времени, нежели рекурсивное выполнение.

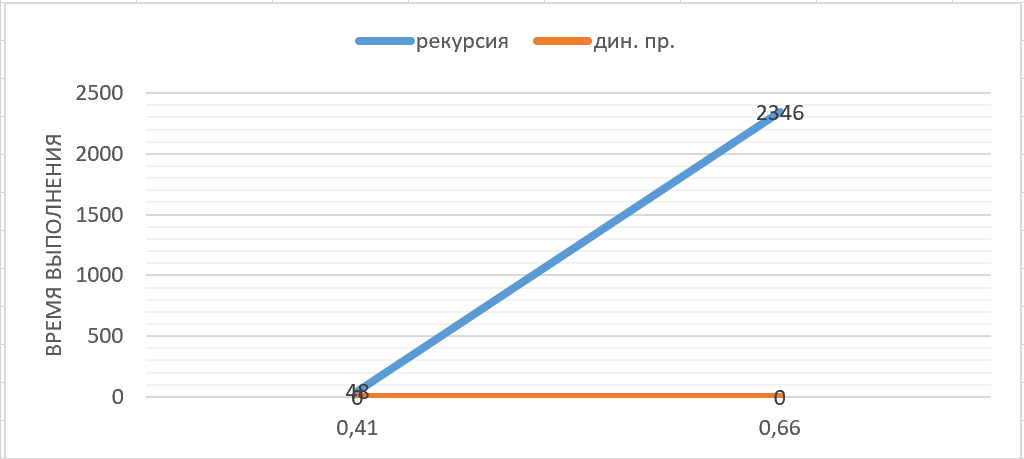


Рисунок 3.1 – График зависимости выполнения

**Задание 4.**

Реализовать вручную пример вычисления дистанции Левенштейна при помощи рекурсивного алгоритма (в соответствии с вариантом).





= 5.

= 4.



= 4.

= 3.



= 3.

= 2.



= 2.

= 1.



= 3.

= 2.



= 2.

= 1.

L(“Л”, “Г”) = 0

1. L(“Ло”, “Г”) = min(2,3,2) =2
2. L(“Лом”, “Г”) = min(3,4,3) = 3
3. L(“Л”, “Го”) = min(3,2,2) = 2
4. L(“Ло”, “Го”) = min(3,3,1) = 1
5. L(“Лом”, “Го”) = min(2,4,3) = 2
6. L(“Л”, “Гом”) = min(4,3,3) = 3
7. L(“Ло”, “Гом”) = min(4,2,3) = 2
8. L(“Лом”, “Гом”) = min(3,3,1) = 1
9. L(“Л”, “Гомо”) = min(5,4,4) = 4
10. L(“Л”, “Гомон”) = min(6,5,5) = 5
11. L(“Ло”, “Гомо”) = min(5,3,3) = 3
12. L(“Лом”, “Гомо”) = min(5,3,3) = 3
13. L(“Ло”, “Гомон”) = min(6,4,5) = 4
14. L(“Лом”, “Гомон”) = min(5,3,4) = 3

Таким образом самая короткая дистанция Левенштейна составляет 3 шага. Результат программы представлен на рисунке 4.1.

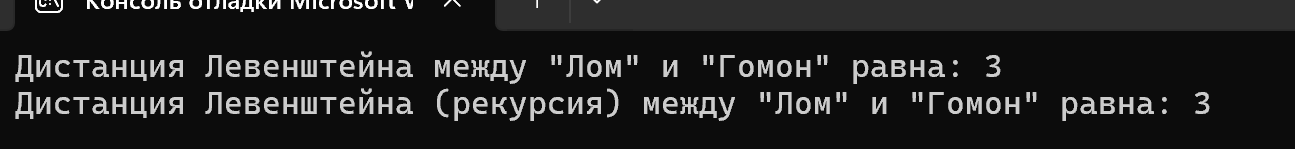


Рисунок 4.1 – результат выполнения программы

**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 .

Листинги кода представлены ниже:

|  |
| --- |
| #include <algorithm>  #include "LCS.h"  #include <cstring>  #include "LCS.h"  #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 };  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  }  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 5.1 - LCS.cpp

|  |
| --- |
| #include <iostream>  #include "LCS.h"  int main()  {  setlocale(LC\_ALL, "rus");  clock\_t t1 = 0, t2 = 0, t3 = 0, t4 = 0;  t1 = clock();  char X[] = "PVTWNHO", Y[] = "RPTWYK";  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, // длина последовательности X  "PVTWNHO", // последовательность X  sizeof(Y) - 1, // длина последовательности Y  "RPTWYK" // последовательность Y  );  std::cout << std::endl << "-- длина LCS: " << s << std::endl;  t2 = clock();  /// <summary>    /// </summary>  t3 = clock();  char z[100] = "";  char x[] = "PVTWNHO",  y[] = "RPTWYK";  int l = lcsd(x, y, z);  std::cout << std::endl  << "-- наибольшая общая подпоследовательость - LCS(динамическое "  << "программирование)" << std::endl;  t4 = clock();  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 << '\n' << "Время выполнения рекурсивно: " << t2 - t1;  std::cout << '\n' << "Время выполнения динамичически: " << t4 - t3 << '\n' << std::endl;  system("pause");  return 0;  } |

Листинг 5.2 – main.cpp

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

Листинг 5.3 – LCS.h

Выполнив сравнительный анализ времени затраченного на решение задачи о наибольшей общей подпоследовательности для двух методов решения, можно заметить, что динамический алгоритм работает быстрее, однако, в данном ситуации оба метода затрачивают для выполнения минимально времени. Результаты представлены на рисунке 5.1:

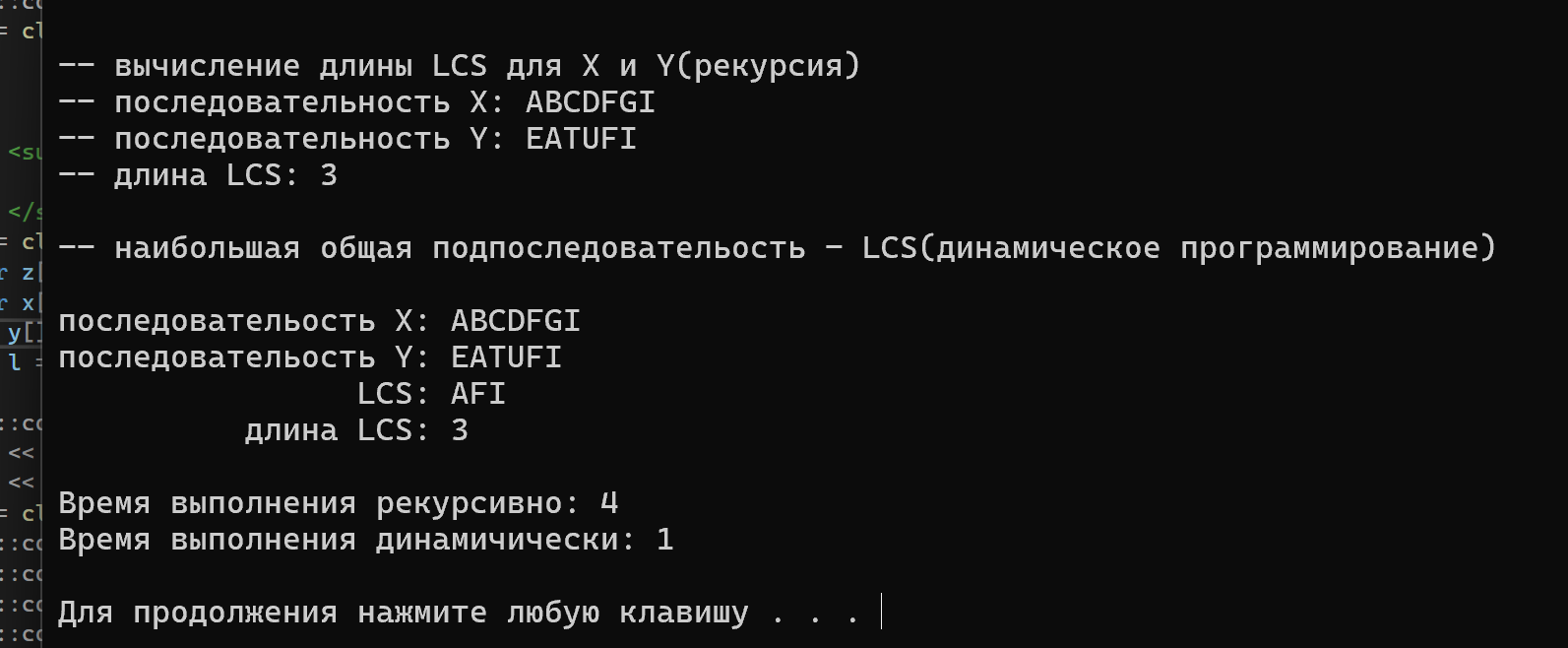


Рисунок 5.1 – выполнение программы

На рисунке 5.2 представлен график зависимости от суммарной длинны двух подстрок и необходимом времени выполнения при помощи динамического алгоритма:

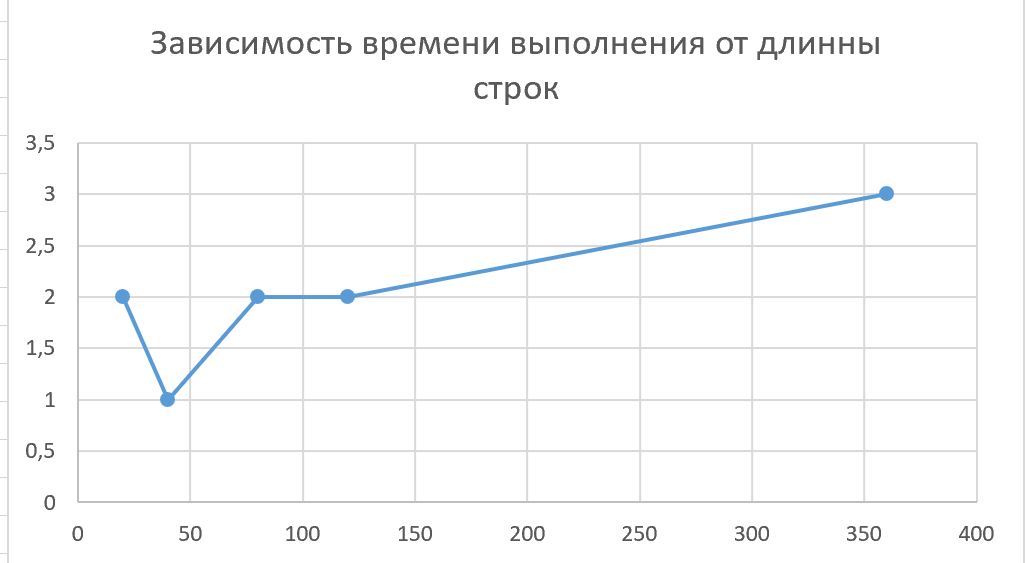


Рисунок 5.2 – график

Видно, что с увеличением длины входных последовательностей время выполнения имеет тенденцию к росту. На небольших длинах (до 50 символов) наблюдаются колебания, что может быть связано с особенностями работы процессора, кеширования или флуктуациями в системе. После 100 символов рост времени выполнения становится более плавным и линейным, что соответствует предсказуемому поведению метода динамического программирования.

**Алгоритм прохождения:**

* Первая строка и первый столбец всегда 0 (базовый случай).
* Если символы совпадают, то берем диагональное значение +1.
* Если не совпадают, берем максимум из верхней или левой ячейки.

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

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | E | A | T | U | F | I |
|  |  |  |  |  |  |  |  |
| A |  |  |  |  |  |  |  |
| B |  |  |  |  |  |  |  |
| C |  |  |  |  |  |  |  |
| D |  |  |  |  |  |  |  |
| F |  |  |  |  |  |  |  |
| G |  |  |  |  |  |  |  |
| I |  |  |  |  |  |  |  |

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