Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Математическое программирование

Лабораторная работа № 4

Динамическое программирование

Студент: Жамойдо А.И.

ФИТ 2 курс 10 группа

Преподаватель: Ромыш А.С.

Минск 2025

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

**Ход выполнения работы**

**Задание 1.** На языке С++ сгенерировать случайным образом строку букв латинского алфавита  длиной  символов и длиной .

**Решение:**

Исходный код программы приведен в листинге 3.1, а результат показан на рисунке 3.1.

|  |
| --- |
| #include <iostream>  #include <ctime>  #include <Windows.h>  using namespace std;  #define FIRST\_LEN 300  #define SECOND\_LEN 250  char\* GenerateRandomString(int size)  {  char\* str = (char\*)malloc(sizeof(char) \* size);  for (int i = 0; i < size; i++)  {  str[i] = rand() % 26 + 'a'; // 26 букв в алфавите  }  return str;  }  int main()  {  SetConsoleCP(1251);  SetConsoleOutputCP(1251);  srand(time(NULL));  char\* s1 = GenerateRandomString(FIRST\_LEN);  cout << "S1: " << endl;  for (int i = 0; i < FIRST\_LEN; i++)  {  if (i % 50 == 0)  {  cout << "\n";  }  cout << s1[i];  }  cout << endl << endl;  char\* s2 = GenerateRandomString(SECOND\_LEN);  cout << "S2: " << endl;  for (int i = 0; i < SECOND\_LEN; i++)  {  if (i % 50 == 0)  {  cout << "\n";  }  cout << s2[i];  }  cout << endl << endl;  return 0;  } |

Листинг 3.1 — файл Lab\_04\_1.cpp

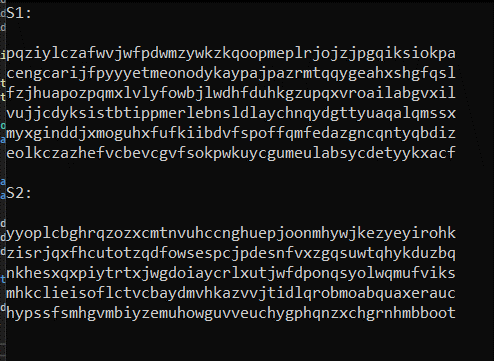


Рисунок 3.1 — результат

**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).

**Решение:**

Ниже приведены варианты реализации нахождения дистанции Левенштейна при помощи динамического программирования и при помощи рекурсивного алгоритма.

|  |
| --- |
| #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  );  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 3.4 — 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 3.5 — файл Levenshtein.cpp

|  |
| --- |
| #include <algorithm>  #include <iostream>  #include <ctime>  #include <iomanip>  #include "Levenshtein.h"  #include <Windows.h>  #define FIRST\_LEN 300  #define SECOND\_LEN 250  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'; // 26 букв в алфавите  }  return str;  }  int main()  {  SetConsoleCP(1251);  SetConsoleOutputCP(1251);  char\* s1 = GenerateRandomString(FIRST\_LEN);  cout << "S1: " << endl;  for (int i = 0; i < FIRST\_LEN; i++)  {  if (i % 50 == 0)  {  cout << "\n";  }  cout << s1[i];  }  cout << endl << endl;  srand(time(NULL) + 1);  char\* s2 = GenerateRandomString(SECOND\_LEN);  cout << "S2: " << endl;  for (int i = 0; i < SECOND\_LEN; 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 = sizeof(s1);  int ly = sizeof(s2);  int s1\_size[]{ FIRST\_LEN / 25, FIRST\_LEN / 20};  int s2\_size[]{ SECOND\_LEN / 25, SECOND\_LEN / 20};  cout << "\n\n-- расстояние Левенштейна -----";  cout << "\n\n--длина --- рекурсия -- дин.програм. ---\n";  for (int i = 0; i < 2; 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 3.6 — файл Lab\_04\_2.cpp

На рисунке 3.2 представлены дистанции Левенштейна вычисленные при помощи метода динамического программирования, а также рекурсивным алгоритмом.

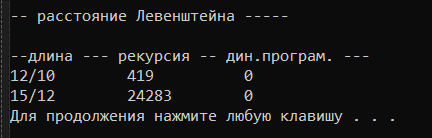


Рисунок 3.2 — результат программы

**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).

**Решение:**

На графике, представленном на рисунке 3.3, можно заметить, что выполненные с помощью динамического алгоритма, вычисления производятся в разы быстрее, чем с помощью рекурсивного алгоритма.

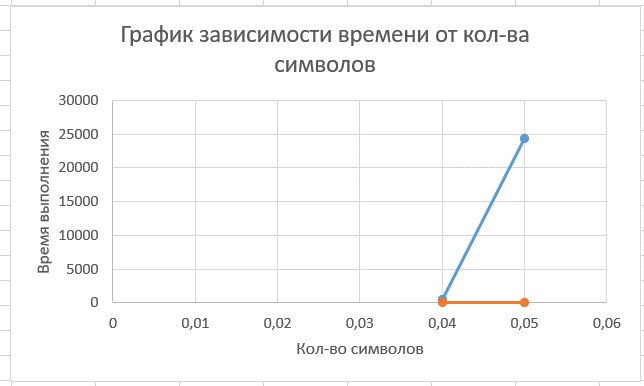


Рисунок 3.3 — сравнительный анализ времени выполнения

**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).

|  |  |
| --- | --- |
| Задание 4 (Вариант 3) | |
| Лом | Гомон |

**Решение:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  | Г | О | М | О | Н |
|  | 0 | 1 | 2 | 3 | 4 | 5 |
| Л | 1 | 1 | 2 | 3 | 4 | 5 |
| О | 2 | 2 | 1 | 3 | 3 | 4 |
| М | 3 | 3 | 2 | 1 | 2 | 3 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  | Г | О | М | О | Н |
|  | 0 | 1 | 2 | 3 | 4 | 5 |
| Л | 1 | 1 | 2 | 3 | 4 | 5 |
| О | 2 | 1 | ⬉1 | 3 | 3 | 4 |
| М | 3 | 3 | 2 | ⬉1 | ⬅2 | ⬅3 |

**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.

|  |  |
| --- | --- |
| Задание 5 (вариант 3) | |
| ABCDFGI | EATUFI |

**Решение:**

|  |
| --- |
| #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 5.1 — 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;  }  #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[] = "ABCDFGI";  char Y[] = "EATUFI";  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, "ABCDFGI", sizeof(Y) - 1, "EATUFI");  std::cout << std::endl << "-- длина LCS: " << s << std::endl;  t2 = clock();    t3 = clock();  char x[] = "ABCDFGI";  char y[] = "EATUFI";  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 Lab\_04\_5.cpp

На рисунке 5.1 представлена наибольшая общая подпоследовательность последовательностей Х и У, вычисленные при помощи метода динамического программирования, а также рекурсивным алгоритмом.

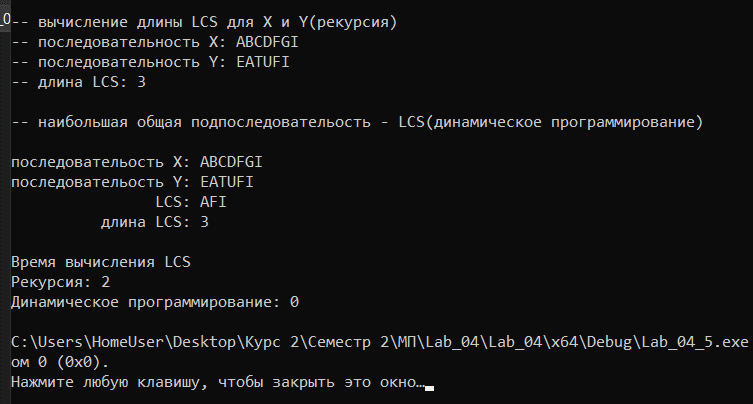


Рисунок 5.1 — результат выполнения программы

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

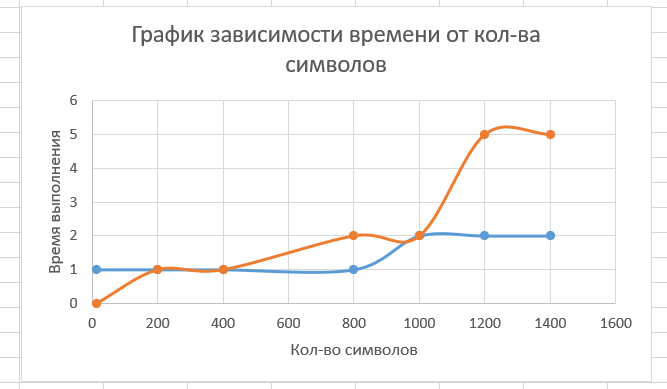


Рисунок 5.2 – График зависимости

**Алгоритм прохождения:**

1. Создаем двумерный массив размером (len(ABCDFGI) + 1) на (len(EATUFI) + 1), где каждый элемент равен 0.

2. Проходим по каждой строке и столбцу массива, начиная с первого и до последнего.

3. Если значение строки или столбца равно 0, то элемент массива на пересечении этой строки и столбца остается равным 0.

4. Если символ строки и столбца равны, то элемент массива на пересечении этой строки и столбца равен значению элемента на предыдущей диагонали плюс 1, т.е. с\_(𝑛,𝑝) = c\_(𝑛−1,𝑝−1) + 1.

5. Если символ строки и столбца не равны, то элемент массива на пересечении этой строки и столбца равен максимуму между значением элемента на предыдущей строке и значением элемента на предыдущем столбце, т.е. c\_(𝑛,𝑝) = max⁡( с\_(𝑛,𝑝"−1" ), c\_(𝑛−1,𝑝)).

6. После прохода по всем элементам массива, наибольшая общая подпоследовательность будет равна значению элемента на пересечении последней строки и последнего столбца.

Алгоритм прохода будет выглядеть следующим образом:

1)

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

2)

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

3)

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

4)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | A | B | C | D | F | G | I |
| E | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| A | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| T | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| U | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| F | 1 | 1 | 1 | 1 | 2 | 2 | 2 |
| I | 2 | 2 | 2 | 2 | 2 | 2 | 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. Так же, я сделал следующие выводы:

1. Динамическое программирование - это мощный метод решения задач, позволяющий эффективно решать широкий спектр задач, которые не могут быть решены простыми алгоритмами.

2. Метод динамического программирования заключается в разбиении сложной задачи на более простые подзадачи, решение которых затем комбинируется в общее решение задачи.

3. Решение задач методом динамического программирования отличается высокой скоростью выполнения благодаря использованию кэширования вычислений и быстрой обработке данных.

4. В ходе выполнения лабораторной работы был исследован один из наиболее популярных методов динамического программирования - расстояние Левенштейна. Результаты экспериментов показали, что данная техника может быть очень эффективна для решения задач, связанных с обработкой текстовых данных.

5. Решение задач методом динамического программирования может быть осуществлено не только с помощью программирования на языке C++, но и на других языках, таких как Python или Java.