Лабораторная работа 4

ДИНАМИЧЕСКОЕ ПРОГРАММИРОВАНИЕ

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

ЗАДАНИЕ ДЛЯ ВЫПОЛНЕНИЯ:

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

|  |
| --- |
| #define FIRST\_LEN 300  #define SECOND\_LEN 200  int \_tmain(int argc, \_TCHAR\* argv[])  {  setlocale(LC\_ALL, "rus");  srand(time(0));  char\* s1 = Task1(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 = Task1(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;  } |

Листинг 1.1 – Генерация строк

Результат выполнения программы представлен на рисунке 1.1

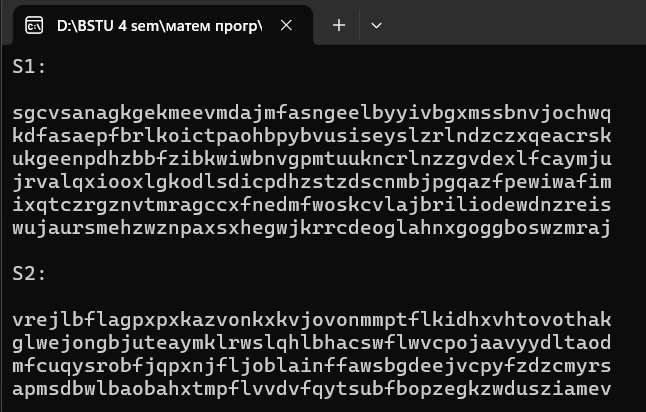


Рисунок 1.1 – Результат работы программы

***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). Код представлен в листинге 1.2. Результат выполнения программы представлен на рисунке 1.2.

|  |
| --- |
| 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\*\* matr;  int w, left, top, left\_top;  matr = new int\* [lx];  for (int i = 0; i < lx; i++)  matr[i] = new int[ly];  matr[0][0] = 0;  for (int i = 1; i < lx; i++)  matr[i][0] = i;  for (int j = 1; j < ly; j++)  matr[0][j] = j;  for (int i = 1; i < lx; i++)  for (int j = 1; j < ly; j++)  {  w = x[i - 1] == y[j - 1] ? 0 : 1;  top = matr[i - 1][j];  left = matr[i][j - 1];  left\_top = matr[i - 1][j - 1];  matr[i][j] = std::min(left\_top + w, std::min(top + 1, left + 1));  }  return matr[lx - 1][ly - 1];  }  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;  }; |

Рисунок 1.2 – Вычисление расстояния Левенштейна

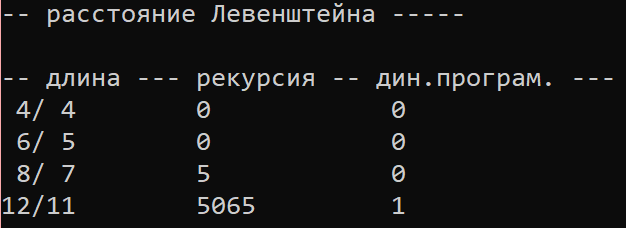
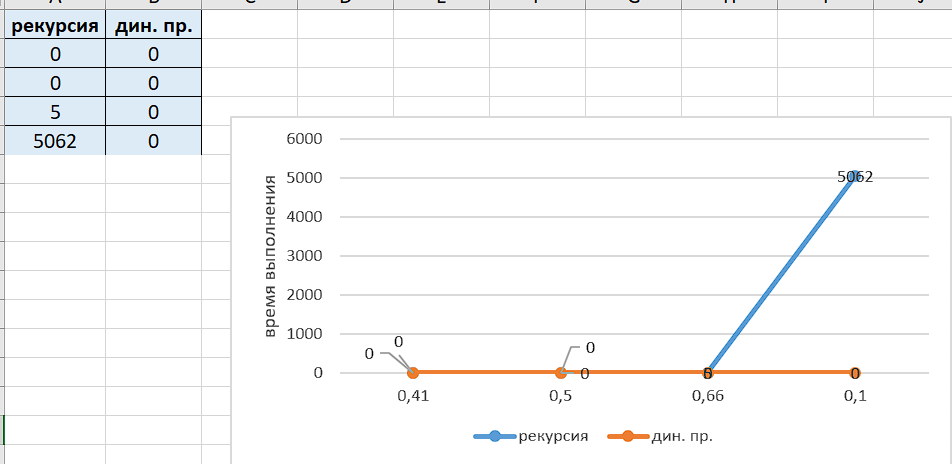


Рисунок 1.2 – Результат работы программы

Задание 3:



Экспоненциальная зависимость для рекурсивного алгоритма и линейная зависимость для динамического программирования

Задание 4:

Для вычисления расстояния Левенштейна между двумя строками методом рекурсии, мы можем использовать следующий подход.

1. L(«Том», «Исток») = min

2. L(«То», «Исток») = min

3. L(«Том», «Исто») = min

4. L(«То», «Исто») = min

5. L(«Т», «Исток») = min

L(«», «Исток») = 5,

L(«», «Исто») = 4

6. L(«Т», «Исто») = min

L(«», «Исто») = 4,

L(«», «Ист») = 3

7. L(«Том», «Ист») = min

7. L(«Том», «Ис») = min

7. L(«Том», «И») = min

L(«Том», «») = 3

8. L(«То», «Ист») = min

9. L(«Т», «Ист») = min

L(«», «Ис») = 2

10. L(«То», «Ис») = min

11. L(«Т», «Ис») = min

L(«», «И») = 1

12. L(«То», «И») = min

L(«То», «») = 2,

L(«Т», «») = 1,

13. L(«Т», «И») = min

L(«Т», «») = 1

15. L(«Т», «И») = min(2,2,1) = 1

16. L(«То», «И») = min (2, 3, 2) = 2

17. L(«Т», «Ис») = min (3, 2, 2) = 2

18. L(«То», «Ис») = min (3, 3, 2) = 2

19. L(«Т», «Ист») = min (4, 3, 2) = 2

20. L(«Том», «И») = min (3, 4, 3) = 3

21. L(«Том», «Ис») = min (3, 4, 3) = 3

22. L(«То», «Ист») = min (3, 3, 3) = 3

23. L(«Том», «Ист») = min (4, 4, 3) = 3

24. L(«Т», «Исто») = min (5, 3, 4) = 4

25. L(«Т», «Исток») = min (6, 5, 5) = 5

26. L(«То», «Исто») = min (5, 4, 2) = 2

27. L(«Том», «Исто») = min (3, 4, 4) = 3

28. L(«То», «Исток») = min (6, 3, 5) = 3

29. L(«Том», «Исток») = min (4, 4, 3) = 3

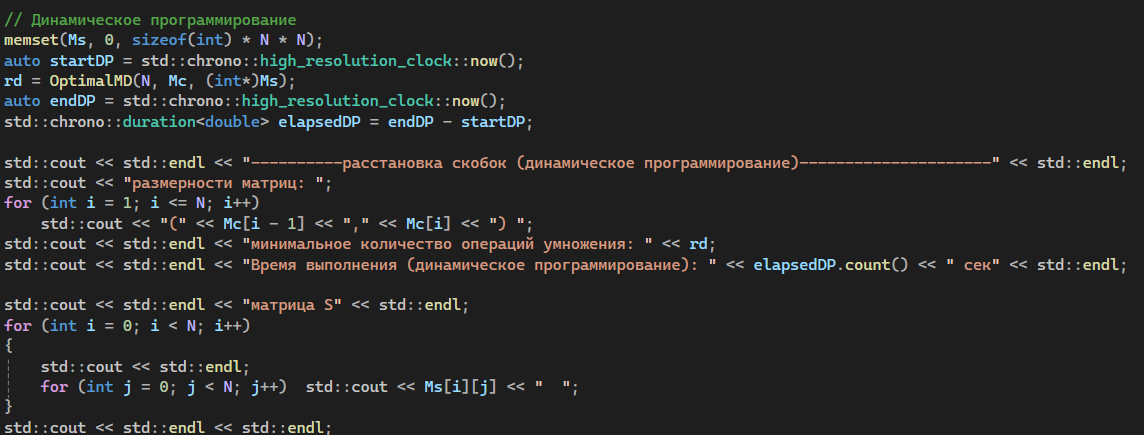
Задание 5:

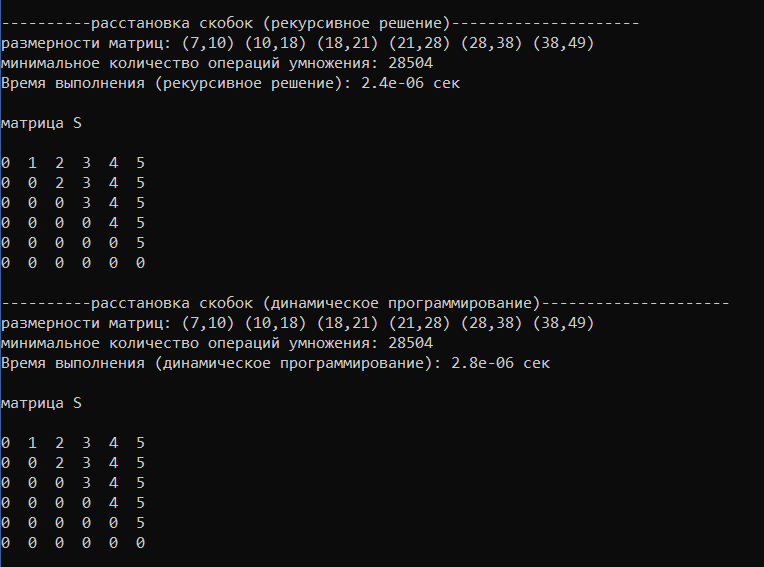
Рекурсивный алгоритм

Изображение выглядит как текст, снимок экрана, программное обеспечение, Шрифт

Автоматически созданное описание

Динамическое программирование





А1=7\*10, А2=10\*18, А3=18\*21, А4 =21\*28, А5 =28\*38, А6 =38\*49.

Скобки расставляются по принципу «сначала внешние – затем внутренние». Найдем элемент (1,6) в матрице s. Он равен 5. Это означает, что точка разрыва находится между первой и шестой матрицей после пятой матрицы, что позволяет расставить внешние скобки следующим образом: (А1\*А2\*А3\*А4\*А5\*)А6).

Точку разрыва между первой и пятой матрицей определяет элемент (1,5). А между четвертой и шестой - (4,6). После расстановки скобок выражение будет выглядеть следующим образом

((((((А1)\*А2)\*А3)\*А4)\*А5)\*А6)

Матрица :

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **1** | **2** | **3** | **4** | **5** | **6** |
| **1** | 0 | 1 | 2 | 3 | 4 | 5 |
| **2** | 0 | 0 | 2 | 3 | 4 | 5 |
| **3** | 0 | 0 | 0 | 3 | 4 | 5 |
| **4** | 0 | 0 | 0 | 0 | 4 | 5 |
| **5** | 0 | 0 | 0 | 0 | 0 | 5 |
| **6** | 0 | 0 | 0 | 0 | 0 | 0 |

Полученная расстановка скобок позволяет получить минимальное количество операций умножения, равное 28504.

**Вывод**: в генерации строк были созданы две строки случайным образом с использованием латинского алфавита, что предоставило данные для вычисления расстояния Левенштейна. Для вычисления расстояния Левенштейна применены два метода: рекурсивный и динамическое программирование. Первый имеет экспоненциальную сложность, в то время как второй позволяет снизить сложность до квадратичной, что делает его более эффективным. Был проведен сравнительный анализ времени выполнения двух методов на различных значениях длин строк. Метод динамического программирования показал значительное улучшение производительности по сравнению с рекурсивным методом. Оба метода решения о расстановке скобок при умножении матриц были применены к задаче, и их производительность сравнена. Метод динамического программирования оказался более эффективным. Исследование подтвердило эффективность метода динамического программирования для решения задач с определенными структурами и подзадачами.