Белорусский государственный технологический университет

Кафедра Программной инженерии

**“Математическое программирование”**

**Отчет по лабораторной работе №4**

**Динамическое программирование**

**Вариант 6**

Выполнил: Карленок Ю.А.

ФИТ 2 курс, 4 группа

Проверила: Павловская К.И.

Минск 2019

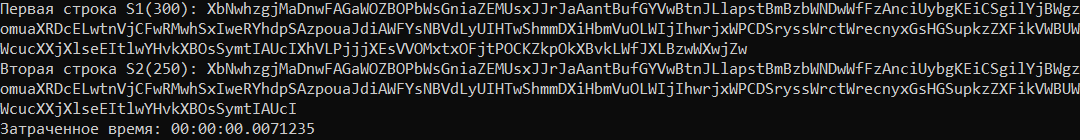
**Цель работы:** освоить общие принципы решения задач методом динамического программирования, сравнить полученные решения задач с рекурсивным методом.

**Ход выполнения**

**Задание 1**

**Сгенерировать две строки размерами 300 и 250 символов:**

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).**

public static int LevensteinRecursion(string first, string second)

{

if (first == second)

{

return 0;

}

if (first == "")

{

return second.Length;

}

if (second == "")

{

return first.Length;

}

else

{

int firstLenght = first.Length, secondLenght = second.Length;

int[] result = new int[3];

result[0] = LevensteinRecursion(first.Substring(0, firstLenght - 1), second) + 1;

result[1] = LevensteinRecursion(first, second.Substring(0, secondLenght - 1)) + 1;

result[2] = LevensteinRecursion(first.Substring(0, firstLenght - 1),

second.Substring(0, secondLenght - 1))

+ ((first[firstLenght - 1] == second[secondLenght - 1]) ? 0 : 1);

return result.Min();

}

}

Пример выполнения:



public static int LevensteinDynamic(string first, string second)

{

int firstLenght = first.Length, secondLenght = second.Length;

int[,] result = new int[(firstLenght + 1), (secondLenght + 1)];

for (int i = 0; i <= firstLenght; i++)

{

result[i, 0] = i;

}

for (int j = 0; j <= secondLenght; j++)

{

result[0, j] = j;

}

for (int i = 1; i <= firstLenght; i++)

{

for (int j = 1; j <= secondLenght; j++)

{

result[i, j] = GetMin(result[i - 1, j] + 1, result[i, j - 1] + 1,

result[i - 1, j - 1] + (first[i - 1] == second[j - 1] ? 0 : 1));

}

}

return result[firstLenght, secondLenght];

}

private static int GetMin(int x, int y, int z)

{

return

(x < y && x < z) ? x :

(y < x && y < z) ? y : z;

}

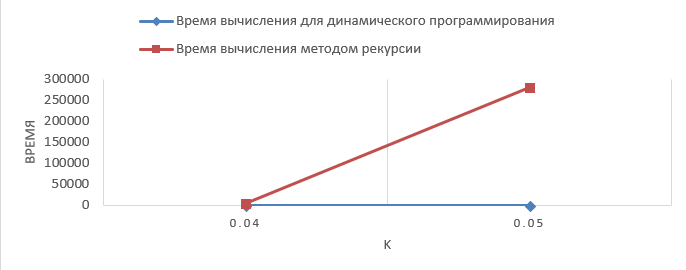
Пример выполнения:



**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).**

Метод динамического программирования значительно эффективнее рекурсивного метода, т.к. выполняется намного быстрее.



Как видно из задания номер 2 и графиков, при больших значениях k, а соответственно, при небольшой длине строк, метод динамического программирования является выигрышным вариантом по сравнению с методом рекурсии. Это происходит по той причине, что в методе ДП мы должны рассмотреть полиноминальное количество вариантов, пока не найдем решение, а в методе рекурсии перебор является экспоненциальным.

**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).**

Найти расстояние Левенштейна между словами «Сом» и «Домик».

1. L(“Сом”, “Домик”) = min(L(“Со”, “Домик”) + 1, L(“Сом”, “Доми”) + 1, L(“Со”, “Доми”) + 1) = min(5, 3, 3) = 3;
2. L(“Со”, “Домик”) = min(L(“С”, “Домик”) + 1, L(“Со”, “Доми”) + 1, L(“С”, “Доми”) + 1) = min(6, 4, 5) = 4;
3. L(“С”, “Домик”) = min(L(“”, “Домик”) + 1, L(“С”, “Доми”) + 1, L(“”, “Доми”) + 1) = min(6, 5, 5) = 5;
4. L(“”, “Домик”) = 4 + 1 = 5;
5. L(“С”, “Доми”) = min(L(“”, “Доми”) + 1, L(“С”, “Дом”) + 1, L(“”, “Дом”) + 1) = min(5, 4, 4) = 4;
6. L(“Сом”, “Доми”) = min(L(“Со”, “Доми”) + 1, L(“Сом”, “Дом”) + 1, L(“Со”, “Дом”) + 1) = (4, 2, 3) = 2;
7. L(“Со”, “Доми”) = min(L(“С”, “Доми”) + 1, L(“Со”, “Дом”) + 1, L(“С”, “Дом”) + 1) = (5, 3, 4) = 3;
8. L(“С”, “Доми”) = min(L(“”, “Доми”) + 1, L(“С”, “Дом”) + 1, L(“”, “Дом”) + 1) = min(5, 4, 4) = 4;
9. L(“С”, “Дом”) = min(L(“”, “Дом”) + 1, L(“С”, “До”) + 1, L(“”, “До”) + 1) = min(4, 3, 3) = 3;
10. L(“С”, “До”) = min(L(“”, “До”) + 1, L(“С”, “Д”) + 1, L(“”, “Д”) + 1) = min(3, 2, 2) = 2;
11. L(“Сом”, “Дом”) = min(L(“Со”, “Дом”) + 1, L(“Сом”, “До”) + 1, L(“Со”, “До”)) = min(3, 3, 1) = 1;
12. L(“Со”, “Дом”) = min(L(“С”, “Дом”) + 1, L(“Со”, “До”) + 1, L(“С”, “До”) + 1) = min(4, 2, 3) = 2;
13. L(“С”, “Дом”) = 2 + 1 = 3;
14. L(“Сом”, “До”) = min(L(“Со”, “До”) + 1, L(“Сом”, “Д”) + 1, L(“Со”, “Д”) + 1) = min(2, 4, 3) = 2;
15. L(“Сом”, “Д”) = min(L(“Со”, “Д”) + 1, L(“Сом”, “”) + 1, L(“Со”, “”) + 1) = min(3, 4, 3) = 3;
16. L(“Со”, “До”) = min(L(“С”, “До”) + 1, L(“Со”, “Д”) + 1, L(“С”, “Д”)) = min(3, 3, 1) = 1;
17. L(“Сом”, “”) = 2 + 1 = 3;
18. L(“Со”, “”) = 1 + 1 = 2;
19. L(“С”, “”) = 0 + 1 = 1;
20. L(“”, “Домик”) = 4 + 1 = 5;
21. L(“”, “Доми”) = 3 + 1 = 4;
22. L(“”, “Дом”) = 2 + 1 = 3;
23. L(“”, “До”) = 1 + 1 = 2;
24. L(“”, “Д”) = 0 + 1 = 1;
25. L(“С”, “До”) = 1 + 1 = 2;
26. L(“Со”, “Д”) = 1 + 1 = 2;
27. L(“С”, “Д”) = 0 + 1 = 1;

Дистанция Левенштейна для слов «Сом» и «Домик»: 3.





**Задание 5**

**Выполнить сравнительный анализ времени затраченного на решение задачи об оптимальной расстановке скобок при умножении нескольких матриц для двух методов решения (рекурсивное решение, динамическое программирование). Размерность матриц взять в соответствии с вариантом. Объяснить в отчете принцип расставления скобок по итоговой матрице + код + копии экрана.**

Последовательность матриц: 9\*12, 12\*20, 20\*23, 23\*30, 30\*40, 40\*51.

public static void MatrixRecursion()

{

int[] Mc = new int[N + 1] { 9, 12, 20, 23, 30, 40, 51 };

int[,] Ms = new int[N, N];

int result = 0;

result = MatrixRecursionRecursion(1, N, N, Mc, Ms);

Console.WriteLine("(Рекурсия) Задача о расстановке скобок при перемножении матриц\nРазмерности матриц: ");

for (int i = 1; i <= N; i++)

{

Console.Write("(" + Mc[i - 1] + ", " + Mc[i] + ") ");

}

Console.WriteLine("\nМинимальное количество операций умножения: " + result);

Console.WriteLine("Матрица S: ");

for (int i = 0; i < N; i++)

{

for (int j = 0; j < N; j++)

{

Console.Write(Ms[i, j] + " ");

}

Console.WriteLine();

}

}

public static int MatrixRecursionRecursion(int i, int j, int n, int[] c, int[,] s)

{

int result = Int32.MaxValue, tempResult = Int32.MaxValue;

if (i < j)

{

for (int k = i; k < j; k++)

{

tempResult = MatrixRecursionRecursion(i, k, n, c, s)

+ MatrixRecursionRecursion(k + 1, j, n, c, s)

+ c[i - 1] \* c[k] \* c[j];

if (tempResult < result)

{

result = tempResult;

s[i- 1, j - 1] = k;

}

}

}

else

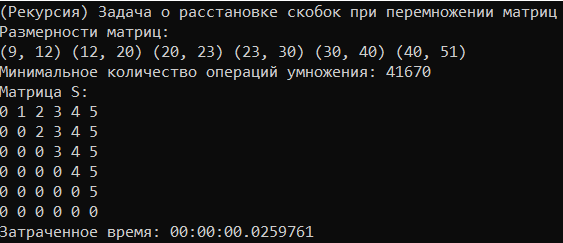
{

result = 0;

}

return result;

}



public static void MatrixDynamic()

{

int[] Mc = new int[N + 1] { 9, 12, 20, 23, 30, 40, 51 };

int[,] Ms = new int[N, N];

int result = 0;

result = MatrixDynamicCicle(N, Mc, Ms);

Console.WriteLine("(Динамика) Задача о расстановке скобок при перемножении матриц\nРазмерности матриц: ");

for (int i = 1; i <= N; i++)

{

Console.Write("(" + Mc[i - 1] + ", " + Mc[i] + ") ");

}

Console.WriteLine("\nМинимальное количество операций умножения: " + result);

Console.WriteLine("Матрица S: ");

for (int i = 0; i < N; i++)

{

for (int j = 0; j < N; j++)

{

Console.Write(Ms[i, j] + " ");

}

Console.WriteLine();

}

}

public static int MatrixDynamicCicle(int n, int[] c, int[,] s)

{

int[,] M = new int[n, n];

int j = 0, q = 0;

for (int i = 1; i <= n; i++)

{

s[i - 1, i - 1] = 0;

}

for (int l = 2; l <= n; l++)

{

for (int i = 1; i <= n - l + 1; i++)

{

j = i + l - 1;

M[i - 1, j - 1] = Int32.MaxValue;

for (int k = i; k <= j - 1; k++)

{

q = M[i - 1, k - 1] + M[k, j - 1] + c[i - 1] \* c[k] \* c[j];

if (q < M[i - 1, j - 1])

{

M[i - 1, j - 1] = q;

s[i - 1, j - 1] = k;

}

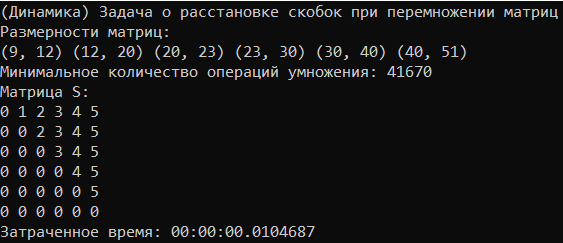
}

}

}

return M[0, n - 1];

}



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 скобки следующим образом: (A1 \* A2 \* A3 \* A4 \* A5) \* (A6).

Точку разрыва между первой и третьей матрицей определяет элемент (1, 3), а между четвертой и шестой – (4, 6). После расстановки скобок выражение будет выглядеть следующим образом ((((A1 \* A2) \* A3) \* A4) \* A5) \* A6. Полученная расстановка скобок позволяет получить минимальное количество операций умножения, равное 41670.