Белорусский государственный технологический университет

Кафедра Программной инженерии

**“Математическое программирование”**

**Отчет по лабораторной работе №4**

**Динамическое программирование**

**Вариант 13**

Выполнил: Савин Владислав Александрович

ФИТ 2 курс, 8 группа

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

Динамическое программирование позволяет свести глобальную оптимизацию аддитивной или мультипликативной целевой функции к поэтапной оптимизации промежуточных целевых функций.

К преимуществам метода динамического программирования по сравнению с “классическими” методами оптимизации относятся более высокая скорость расчетов и широкая область применимости. В частности, для него некритично требование линейности и дифференцируемости функций ***fj***(***xj***) и функция выигрыша может быть задана не в аналитическом, а в табличном виде. Другими словами, метод применим и при решении задач нелинейного и дискретного програмирования.

***Дистанция Левенштейна (расстояние Левенштейна, редакционное расстояние, дистанция редактирования)*** определяется между двумя строками и равна минимальному количеству операций вставки одного символа, удаления одного символа и замены одного символа на другой, необходимых для превращения одной строки в другую.

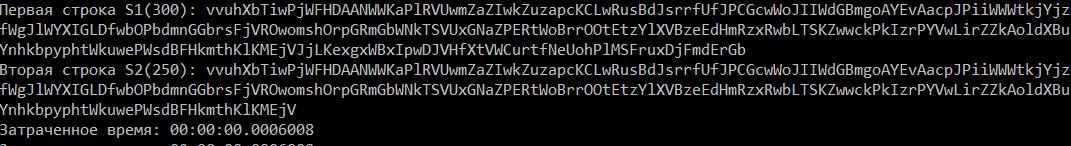
Расстояние Левенштейна активно применяется для исправления ошибок в поисковых системах, в текстовых редакторах, а также в биоинформатике.

**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).**

***Дистанция Левенштейна (расстояние Левенштейна, редакционное расстояние, дистанция редактирования)*** определяется между двумя строками и равна минимальному количеству операций вставки одного символа, удаления одного символа и замены одного символа на другой, необходимых для превращения одной строки в другую.

**Не закончит своё выполнение**

|  |
| --- |
| 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;  } |

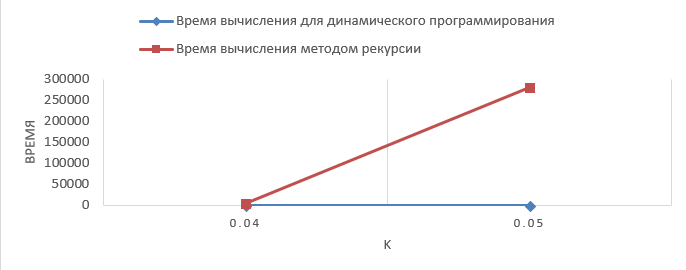
*Пример выполнения:*

****

**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.

Многие оптимизационные алгоритмы основаны на принципе разбиения основной задачи на подзадачи, каждая из которых повторяет основную, но входные их данные таковы, что область допустимых решений становится меньше.

***Рекурсивный алгоритм*** – это алгоритм, решающий задачу путем сведения ее к решению одной или нескольких таких же задач, но в сокращенном их варианте.

Первое определение рекурсивной функции относится к теории вычислимости и является синонимом понятия вычислимой функции, т. е. функции, для вычисления значения которой можно указать алгоритм.

Второе определение, которое и будет использоваться здесь, происходит из области теории программирования.

***Рекурсивная функция*** – это функция, которая вызывает саму себя.

Рекурсивный алгоритм может быть записан в виде рекурсивной функции.

Классическими примерами рекурсивных функций являются функции для вычисления факториала, чисел Фибоначчи и наибольшего общего делителя с помощью алгоритма Эвклида.

Рекурсивную функцию всегда можно преобразовать в цикл, и, наоборот любой цикл можно представить в виде рекурсивной функции.

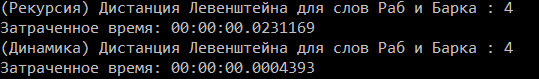
Рекурсивная запись алгоритма, как правило, не дает выигрыша в скорости его работы. Скорее наоборот, так как вызов любой функции связан с сохранением и восстановлением контекста вызывающей функции, что является затратной по времени операцией.

**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).**

1. L(“Раб”, “Барка”) = min(L(“Ра”, “ Барка ”) + 1, L(“Раб”, “ Барк”) + 1, L(“Ра”, “ Барк”) + 1) = min(5, 3, 3) = 3;
2. L(“Ра”, “ Барка ”) = min(L(“Р”, “ Барка ”) + 1, L(“Ра”, “ Барк”) + 1, L(“Р“ Барк ”) + 1) = min(6, 4, 5) = 4;
3. L(“Р”, “ Барка ”) = min(L(“”, “ Барка ”) + 1, L(“Р”, “ Барк”) + 1, L(“”, “ Барк”) + 1) = min(6, 5, 5) = 5;
4. L(“”, “ Барка ”) = 5 + 1 = 6;
5. L(“Р”, “ Барка ”) = min(L(“”, “ Барк”) + 1, L(“Р”, “ Бар”) + 1, L(“”, “ Бар”) + 1) = min(5, 4, 4) = 4;
6. L(“Раб”, “ Барка ”) = min(L(“Ра”, “ Барк”) + 1, L(“Раб”, “Бар”) + 1, L(“ра”, “Бар”) + 1) = (4, 2, 3) = 2;
7. L(“Ра”, “барк”) = min(L(“Р”, “Барк”) + 1, L(“Ра”, “Бар”) + 1, L(“Р”, “Бар”) + 1) = (5, 3, 4) = 3;
8. L(“Р”, “Барк”) = min(L(“”, “Барк”) + 1, L(“Р”, “Бар”) + 1, L(“”, “Бар”) + 1) = min(5, 4, 4) = 4;
9. L(“Р”, “Бар”) = min(L(“”, “Бар”) + 1, L(“Р”, “Ба”) + 1, L(“”, “ба”) + 1) = min(4, 3, 3) = 3;
10. L(“Р”, “Ба”) = min(L(“”, “Ба”) + 1, L(“Р”, “Б”) + 1, L(“”, “Б”) + 1) = min(3, 2, 2) = 2;
11. L(“Раб”, “Барка”) = min(L(“Ра”, “Бар”) + 1, L(“Раб”, “Бар”) + 1, L(“Ра”, “Ба”)) = min(3, 3, 1) = 1;
12. L(“Ра”, “Бар”) = min(L(“Р”, “Бар”) + 1, L(“Ра”, “Бар”) + 1, L(“Р”, “Ба”) + 1) = min(4, 2, 3) = 2;
13. L(“Р”, “Бар”) = 3 + 1 = 4;
14. L(“Раб”, “Ба”) = min(L(“Ра”, “Ба”) + 1, L(“Раб”, “Б”) + 1, L(“Ра”, “Б”) + 1) = min(2, 4, 3) = 2;
15. L(“Раб”, “Б”) = min(L(“Ра”, “Б”) + 1, L(“Раб”, “”) + 1, L(“Ра”, “”) + 1) = min(3, 4, 3) = 3;
16. L(“Ра”, “Ба”) = min(L(“Р”, “Ба”) + 1, L(“Ра”, “Б”) + 1, L(“Р”, “Б”)+1) = min(3, 3, 2) = 2;
17. L(“Раб”, “”) = 2 + 1 = 3;
18. L(“Ра”, “”) = 1 + 1 = 2;
19. L(“Б”, “”) = 0 + 1 = 1;
20. L(“”, “Барак”) = 4 + 1 = 5;
21. L(“”, “Бара”) = 3 + 1 = 4;
22. L(“”, “Бар”) = 2 + 1 = 3;
23. L(“”, “Ба”) = 1 + 1 = 2;
24. L(“”, “Р”) = 0 + 1 = 1;
25. L(“Р”, “Ба”) = 1 + 1 = 2;
26. L(“ра”, “Б”) = 1 + 1 = 2;
27. L(“Р”, “Б”) = 0 + 1 = 1;

Дистанция Левенштейна для слов «Раб» и «Барак»: 4.



**Задание 5**

**Выполнить сравнительный анализ времени затраченного на решение задачи о наибольшей общей под последовательности для двух методов решения**

