Белорусский государственный технологический университет

Кафедра Информационных Систем и Технологий

**Курс «Математическое программирование»**

**Отчёт по лабораторной работе №4**

**динамическое программирование**

**Вариант 13**

Выполнила: Сазонова Е.С.

ФИТ 2 курс 4 группа

Проверил: Бракович А.И.

Минск 2017

**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 .

Решением данной задачи будет являться исходный код на языке С++, представленный на рисунке 1.

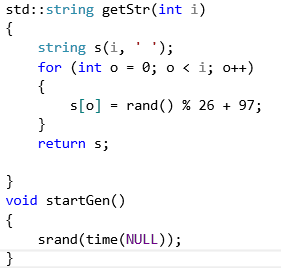


Рис. 1 – код генерации строк

Строки, сгенерированные программой, представлены на рисунке 2.

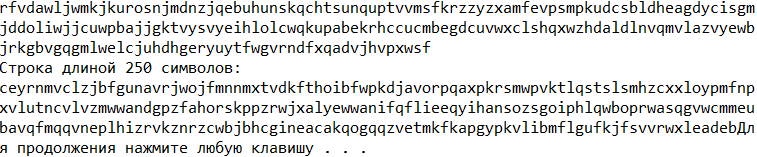


Рис.2 – пример генерации строк

**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).

Ниже приведены варианты реализации нахождения дистанции Левенштейна при помощи динамического программирования - рисунок 3 и при помощи рекурсивного алгоритма – рисунок 4.

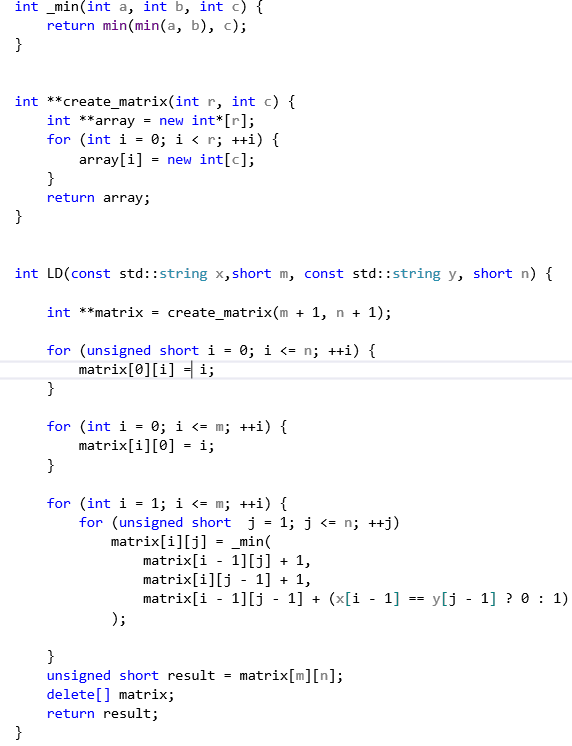


Рис. 3 – исходный код реализации через динамическое программирование

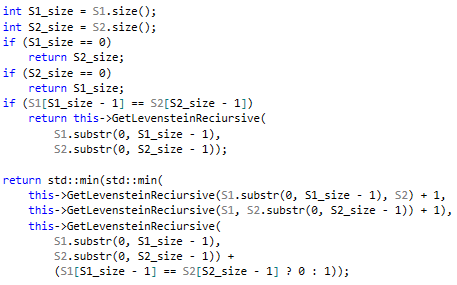


Рис. 4 – Пример реализации рекурсивным методом

На рисунке 5 представлены дистанции Левенштейна, вычисленные при помощи метода динамического программирования, а также рекурсивным алгоритмом.

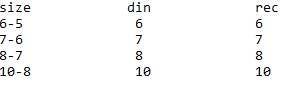


Рис. 5 – проверка работоспособности решений

**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).

На графике, представленном на рисунке 6, а также на результате работы программы - рисунок 7, можно заметить, что вычисление дистанции с помощью динамического алгоритма производятся в разы быстрее, чем с помощью рекурсивного алгоритма. Время на графике представлено в миллисекундах.

Рис. 6 – график времени выполнения

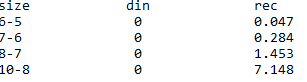


Рис. 7 – сравнение времени работы алгоритмов

**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).

|  |  |  |
| --- | --- | --- |
| Вариант | Задание 4 | |
| 13 | Раб | Барка |

**Решение:**

1.  
2.  
3.  
4.  
5.  
6.  = 5.
7.  = 4.
8.  
9.  = 3.
10.  
11.  
12.  
13.  = 2.
14.  
15.  
16.  
17.  = 1.
18.  
19.  = 3.
20.  = 2.
21.  
22.  = 1.
23. 
24. 
25. 
26. 
27. 
28. 
29. 
30. 
31. 
32. 
33. 
34. 
35. 
36. 

**Задание 5.**

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 k. Отобразить ход решения в отчете (по примеру из лекции) + код и копии экрана.

На рисунке 8 представлен пример реализации рекурсивного метода нахождения наибольшей подпоследовательности.

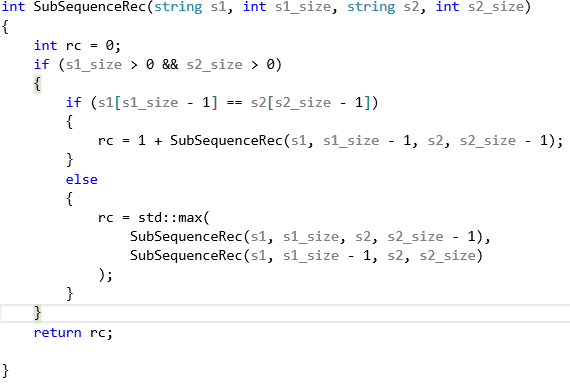


Рис. 8 – пример реализации рекурсивным методом

На рисунках 9 и 10 представлена реализация нахождения наибольшей подпоследовательности методом динамического программирования.

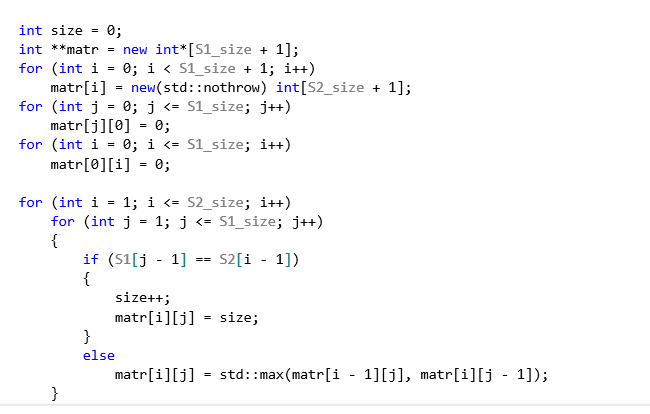


Рис. 9 – матрица значений для динамического программирования

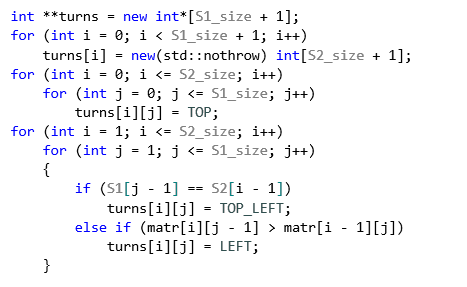


Рис. 10 – матрица направлений для динамического программирования

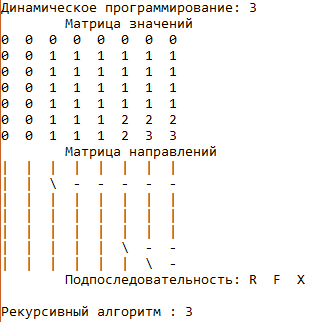


Рис. 11 – итог выполнения программы

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | B | X | W | A | F | R | E |
|  | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| X | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 |
| C | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 |
| D | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 |
| U | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 |
| F | 0 | 0 | 1 | 1 | 1 | 2 | 2 | 2 |
| R | 0 | 0 | 1 | 1 | 1 | 2 | 3 | 3 |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | B | X | W | A | F | R | E |
|  |  |  |  |  |  |  |  |  |
| X |  |  |  |  |  |  |  |  |
| C |  |  |  |  |  |  |  |  |
| D |  |  |  |  |  |  |  |  |
| U |  |  |  |  |  |  |  |  |
| F |  | 1 |  |  |  |  |  |  |
| R |  |  |  |  |  |  |  |  |

Рассмотрим алгоритмы заполнения матриц. Алгоритм заполнения первой матрицы:

1. Заполняем первую строку и столбец нулями
2. Каждый элемент последовательности заполняется по формуле для С: если для позиции ij совпадают, то в нее записывается значение С.1+1, иначе вычисляется максимум от «соседей» слева и сверху.
3. Элемент в правом нижнем углу показывает длину наибольшей общей подпоследовательности.

Алгоритм заполнения второй матрицы:

1. Все ячейки заполняются стрелками вверх (кроме первой строки и первого столбца).
2. Если символы для ячейки i, j совпадают, то стрелка вверх меняется на стрелку вверх-влево.
3. Если числовое значение соседа слева больше, чем значение соседа сверху, то стрелка вверх меняется на стрелку влево.
4. Наибольшую подпоследовательность можно вычислить, проходя по стрелкам от нижнего правого угла в верхнему левому.

Выполняя алгоритм получаем, что искомой подпоследовательностью является: AFI, с длиной равной 3.

**Вывод:**

В данной работе были освоены общие принципы решения задач методом динамического программирования, а также с помощью рекурсивных алгоритмов. Можно заметить, что вычисление с помощью динамического программирования производятся в разы быстрее, чем с помощью рекурсивных алгоритмов. Однако, данный выигрыш чаще всего достигается, только при больших значения входных данных.