Учреждение образования «Белорусский Государственный Технологический Университет»

Факультет Информационных Технологий

Кафедра Информационных систем и Технологий

Отчет по лабораторной работе №4  
**Динамическое программирование**

Выполнил: студент 2 курса 8 группы

Куницкий Н.Д.

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.**

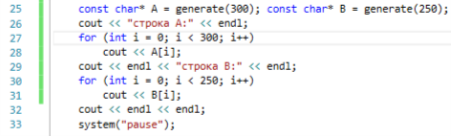


Рисунок 1. Код генерации рандномной строки

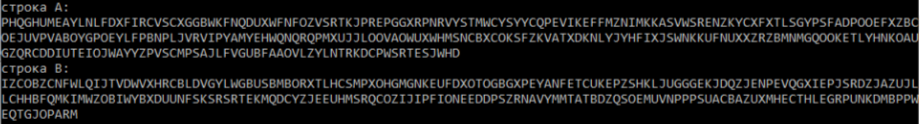


Рисунок 2. Результат генерации рандномной строки

**Отчет:**

**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).

int levenshtein( // ДИНАМИЧЕСКОЕ ПРОГРАММИРОВАНИЕ

int lx, // длина слова x

const char x[], // слово длиной lx

int ly, // длина слова y

const char y[] // слово y

);

int levenshtein\_r( // РЕКУРСИЯ

int lx, // длина строки x

const char x[], // строка длиной lx

int ly, // длина строки y

const char y[] // строка y

);

#include "stdafx.h"

#include <iomanip>

#include <algorithm>

#include "Levenstein.h"

#define DD(i,j) d[(i)\*(ly+1)+(j)]

int min3(int x1, int x2, int x3) // возвращает минимальное значение из трех

{

return std::min(std::min(x1, x2), x3);

}

int levenshtein(int lx, const char x[], int ly, const char y[]) // ДИНАМИЧЕСКОЕ ПРОГРАММИРОВАНИЕ

{

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[]) // РЕКУРСИЯ

{

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;

};

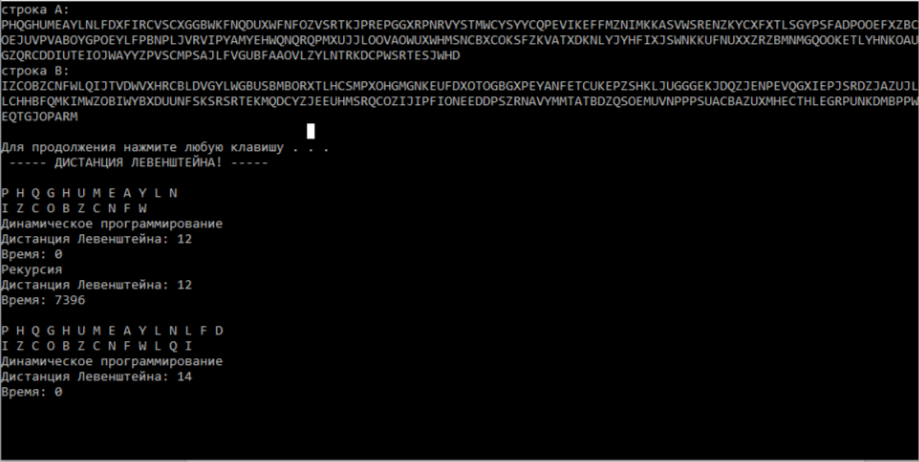


Рисунок 3. Рассчет дистанции Левенштейна.

**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.

Рисунок 4. график времени выполнения

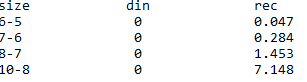


Рисунок 5. сравнение времени работы алгоритмов

**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.**

**Нечетные варианты**. Выполнить сравнительный анализ времени затраченного на решение задачи о наибольшей общей под последовательности для двух методов. Две последовательности взять в соответствии с вариантом. Построить графики зависимости времени вычисления от .

|  |  |  |
| --- | --- | --- |
| Вариант | Задание 5 | |
| 13 | BXWAFRE | XCDUFR |

Наибольшая общая подпоследовательность–LDA.

// ЗАДАЧА О ВЫЧИСЛЕНИИ ДЛИНЫ МАКСИМАЛЬНОЙ ОБЩЕЙ ПОДПОСЛЕДОВАТЕЛЬНОСТИ

LCS.h

int lcs( // РЕКУРСИВНЫЙ МЕТОД

int lenx, // длина последовательности X

const char x[], // последовательность X

int leny, // длина последовательности Y

const char y[] // последовательность Y

);

int lcsd( // ДИНАМИЧЕСКОЕ ПРОГРАММИРОВАНИЕ

const char x[], // последовательность X

const char y[], // последовательность Y

char z[] // наибольшая общая подпоследовательность

);

LCS.cpp

int lcs(int lenx, const char x[], int leny, const char y[]) // РЕКУРСИВНЫЙ МЕТОД

{

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; //длина LCS

}

#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[]) // ДИНАМИЧЕСКОЕ ПРОГРАММИРОВАНИЕ

{

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

Main.cpp

int \_tmain(int argc, \_TCHAR\* argv[])

{

clock\_t t1 = 0, t2 = 0;

setlocale(LC\_ALL, "RUS");

cout << endl << endl << " ----- НАИБОЛЬШАЯ ОБЩАЯ ПОДПОСЛЕДОВАТЕЛЬНОСТЬ ----- " << endl;

char X[] = "BXWAFRE", Y[] = "XCDUFR";

cout << endl << "Последовательность X: " << X;

cout << endl << "Последовательность Y: " << Y;

cout << endl << endl << "Рекурсивный метод: ";

t1 = clock(); int s = lcs(sizeof(X)-1, X, sizeof(Y)-1, Y); t2 = clock();

cout << endl << "Длина LCS: " << s << endl << "Время: " << t2 - t1 << endl << endl;

cout << "Динамическое программирование:" << endl;

char zz[100] = ""; t1 = clock(); int l = lcsd(X, Y, zz); t2 = clock();

cout << "LCS: " << zz << endl << "Длина LCS: " << l << endl << "Время: " << t2 - t1 << endl << endl;

return 0;

}

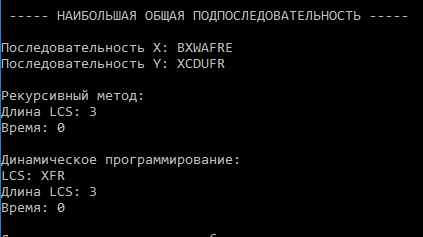


Рисунок 6. Результат вычисления длины LCS.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | 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,jсовпадают, 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 подпоследовательностьможно вычислить проходя по стрелкам от нижнего правого угла в верхнему левому.

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

**Вывод:**

В данной работе были освоены общие принципы решения задач методом динамического программирования, а также с помощью рекурсивных алгоритмов. В ходе сравнения скорости работы данных методов, было замечено, что выполнение с помощью динамического программирования намного быстрее, чем с помощью рекурсивного. Однако стоит заметить, что данный выигрыш чаще всего достигается, только при больших значения входных данных.