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

Сакович А.С.

Проверил:

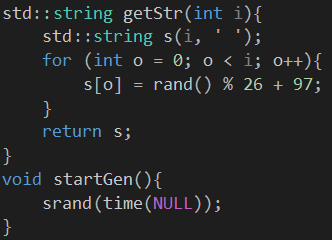
Бракович А.И.

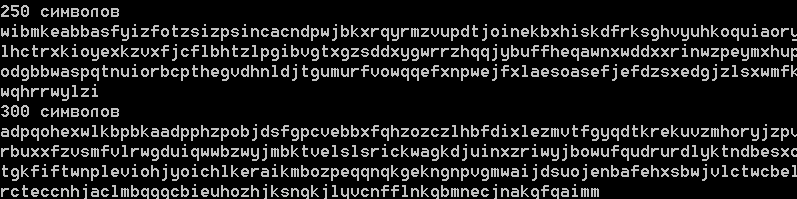
Минск 2017

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

**Ход выполнения работы**

**Задание 1.**

****

****

**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).

// Levenshtein.h

#pragmaonce

intlevenshtein(

intlx, // длина слова x

constcharx[], // слово длиной lx

intly, // длинаслова y

constchary[] // слово y

);

// -- дистанции Левенштeйна (рекурсия)

intlevenshtein\_r(

intlx, // длина строки x

constcharx[], // строка длиной lx

intly, // длина строки y

constchary[] // строка y

);

//Levenshtein.cpp

#include<iomanip>

#include<algorithm>

#include"Levenshtein.h"

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

int min3(intx1, intx2, intx3)

{

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

}

intlevenshtein(intlx, constcharx[], intly, constchary[])

{

int \*d = newint[(lx + 1)\*(ly + 1)];

for (inti = 0; i<= lx; i++) DD(i, 0) = i;

for (int j = 0; j <= ly; j++) DD(0, j) = j;

for (inti = 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));

}

returnDD(lx, ly);

}

intlevenshtein\_r(

intlx, constcharx[],

intly, constchary[]

)

{

intrc = 0;

if (lx == 0) rc = ly;

elseif (ly == 0) rc = lx;

elseif (lx == 1 &&ly == 1 &&x[0] == y[0]) rc = 0;

elseif (lx == 1 &&ly == 1 &&x[0] != y[0]) rc = 1;

elserc = 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)

);

returnrc;

};

// main.cpp

#include<algorithm>

#include<iostream>

#include<ctime>

#include<iomanip>

#include"Levenshtein.h"

#include<time.h>

#defineS1\_SIZE 300

#defineS2\_SIZE 250

intmain()

{

setlocale(LC\_ALL, "rus");

clock\_t t1 = 0, t2 = 0, t3, t4;

srand((unsigned)time(NULL));

char S1[S1\_SIZE],S2[S2\_SIZE];

for (inti = 0; i<S1\_SIZE; i++)

{

S1[i] =(rand() % ('z' - 'a' + 1)) + 'a';

}

for (inti = 0; i<S2\_SIZE; i++)

{

S1[i] = (rand() % ('z' - 'a' + 1)) + 'a';

}

int lS1 = sizeof(S1)-1, lS2 = sizeof(S2)-1;

intk[] = { 25, 20, 15, 10, 5, 2, 1 };

std::cout<<std::endl;

std::cout<<std::endl<<"-- расстояниеЛевенштейна -----"<<std::endl;

std::cout<<std::endl<<"--длина --- рекурсия -- дин.програм. ---"

<<std::endl;

for (inti = 0, j = 0; i< 7; i++, j++)

{

std::cout<<"k = 1/"<< k[j]<<std::endl;

int pref1 = (int)(sizeof(S1) / k[j]), pref2 = (int)(sizeof(S2) / k[j]);

char \*prefS1 = newchar[pref1];

char \*prefS2 = newchar[pref2];

for (int n = 0; n < pref1; n++)

prefS1[n] = S1[n];

for (int n = 0; n < pref2; n++)

prefS2[n] = S2[n];

for (inti = 8; i<std::min(pref1,pref2); i++)

{

t1 = clock(); levenshtein\_r(i, prefS1, i - 2, prefS2);

t2 = clock();

t3 = clock();

levenshtein(i, prefS1, i - 2, prefS2);

t4 = clock();

std::cout<<std::right <<std::setw(2) <<i - 2 <<"/"<<std::setw(2) <<i

<<" "<<std::left <<std::setw(10) << (t2 - t1)

<<" "<<std::setw(10) << (t4 - t3) <<std::endl;

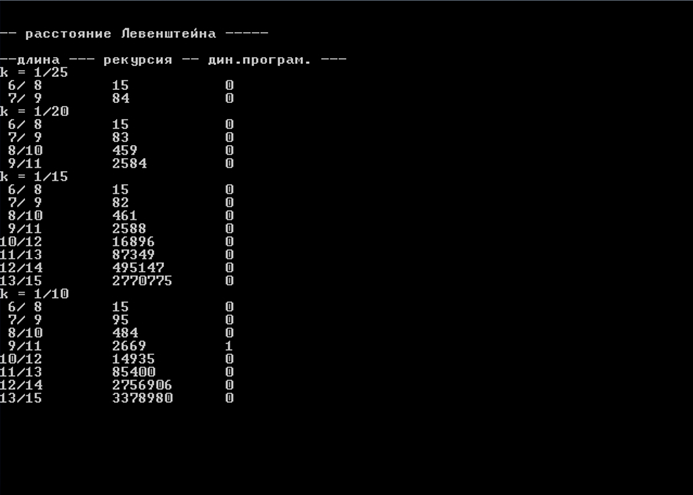
}

}

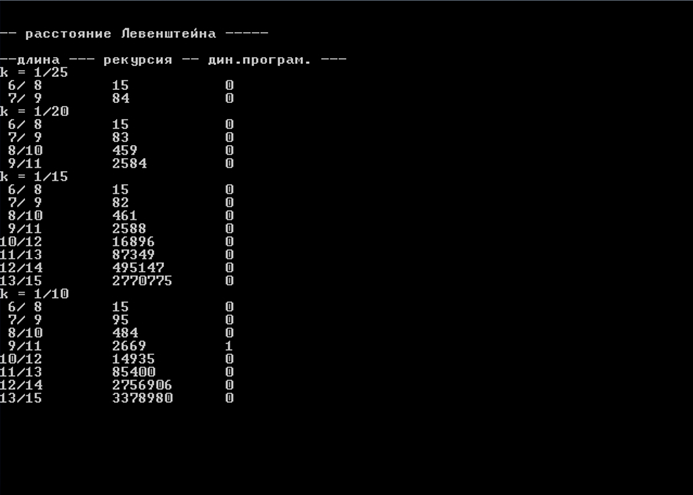
system("pause");

return 0;

}

****

**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).



**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 | |
| 3 | Лом | Гомон |

**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 | |
|  | ABCDFGI | EATUFI |

Наибольшая общая подпоследовательность–LDA.

**Динамический метод**

//LCS.h

int lcsd(

constchar x[], // последовательность X

constchar y[], // последовательность Y

char z[] // наибольшая общая подпоследовательность

);

//LCS.cpp

#include<cstring>

#include"LCS.h"

#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, constchar 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;

}

elseif (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(constchar x[], constchar y[], char z[])

{

int n;

int lenx = strlen(x), leny = strlen(x),

\*C = newint[(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;

}

elseif (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

#include<iostream>

#include"LCS.h"

void main()

{

setlocale(LC\_ALL, "rus");

char z[100] = "";

char x[] = " ABCDFGI ",

y[] = " EATUFI ";

int l = lcsd(x, y, z);

std::cout << std::endl

<<"-- наибольшая общая подпоследовательость - LCS(динамическое"

<<"программирование)"<< std::endl;

std::cout << std::endl <<"последовательость X: "<< x;

std::cout << std::endl <<"последовательость Y: "<< y;

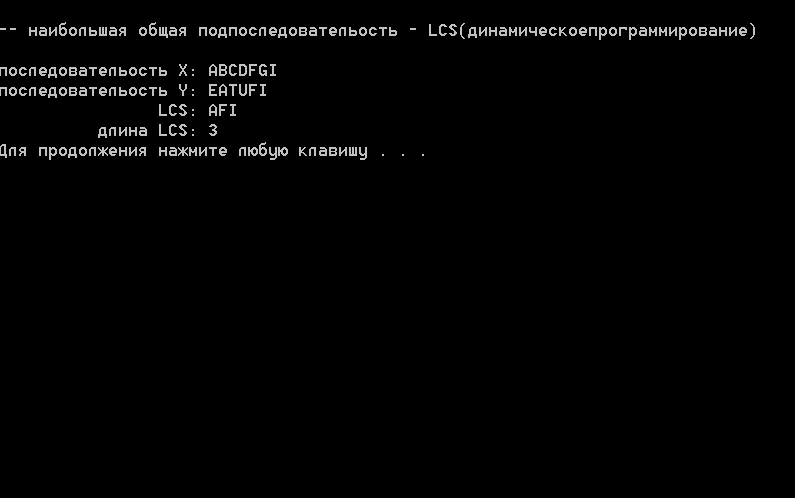
std::cout << std::endl <<" LCS: "<< z;

std::cout << std::endl <<" длина LCS: "<< l;

std::cout << std::endl;

system("pause");

}



**Рекурсивный метод**

// - LCS.h

// -- рекурсивное вычисление длины LCS

int lcs(

int lenx, // длина последовательности X

constchar x[], // последовательность X

int leny, // длина последовательности Y

constchar y[] // последовательность Y

);

//LCS.cpp

#include<algorithm>

#include"LCS.h"

int lcs(intlenx, constcharx[],

intleny, constchary[])

{

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

}

//main.cpp

#include<iostream>

#include"LCS.h"

void main()

{

setlocale(LC\_ALL, "rus");

char X[] = " ABCDFGI ", Y[] = " EATUFI";

std::cout << std::endl <<"-- вычисление длины LCS для X и Y(рекурсия)";

std::cout << std::endl <<"-- последовательность X: "<< X;

std::cout << std::endl <<"-- последовательность Y: "<< Y;

int s = lcs(

sizeof(X)-1, // длина последовательности X

X, // последовательность X

sizeof(Y)-1, // длина последовательности Y

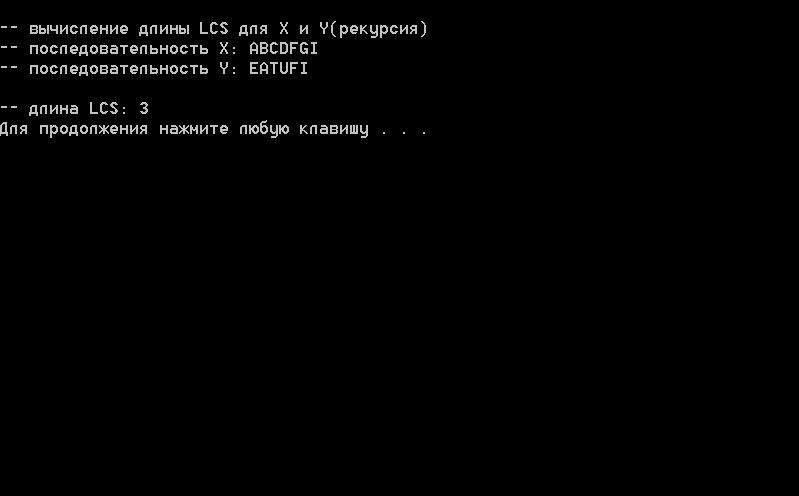
Y// последовательность Y

);

std::cout << std::endl <<"-- длина LCS: "<< s << std::endl;

system("pause");

}

****

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | A | B | C | D | F | G | I |
|  | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| E | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| A | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| T | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| U | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| F | 0 | 1 | 1 | 1 | 1 | 2 | 2 | 2 |
| I | 0 | 1 | 1 | 1 | 1 | 2 | 2 | 3 |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | A | B | C | D | F | G | I |
|  |  |  |  |  |  |  |  |  |
| E |  |  |  |  |  |  |  |  |
| A |  |  |  |  |  |  |  |  |
| T |  |  |  |  |  |  |  |  |
| U |  |  |  |  |  |  |  |  |
| F |  | 1 |  |  |  |  |  |  |
| I |  |  |  |  |  |  |  |  |

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: AFI, 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.