**Лабораторная работа 4**

**ДИНАМИЧЕСКОЕ ПРОГРАММИРОВАНИЕ**

**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 .

**Решение:**

#define \_rand(min, max) ( rand() % ((max) - (min) + 1) + (min) )

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

{

setlocale(LC\_ALL, "rus");

srand(time(NULL));

char abc[25]; // наш алфавит

char s1[300];

char s2[250];

// заполняем массив

for (int i = 97, n = 0; i <= 122; ++i, ++n)

{

abc[n] = (char)i;

}

for (int i = 0; i < 300; i++)

{

s1[i] = abc[\_rand(0, 25)];

}

for (int i = 0; i < 250; i++)

{

s2[i] = abc[\_rand(0, 25)];

}

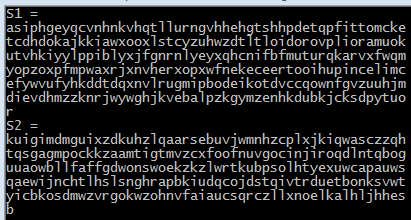


Рис.1 – пример генерации строк

**Задание 2.** Вычислить двумя способами (рекурсивно и с помощью динамического программирования)  – Левенштейна для , где - длина строки ,  - строка, состоящая из первых  символов строки . (копии экрана и код вставить в отчет).

**Решение:**

Ниже приведены варианты реализации нахождения дистанции Левенштейна при помощи динамического программирования и при помощи рекурсивного алгоритма.

Исходный код реализации через динамическое программирование:

int min3(int x1, int x2, int x3)

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

int levenshtein(int lx, const char x[],

int ly, const char y[])

{

int \*\*matr;

int w, left, top, left\_top;

matr = new int\*[lx];

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

matr[i] = new int[ly];

matr[0][0] = 0;

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

matr[i][0] = i;

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

matr[0][j] = j;

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

for (int j = 1; j < ly; j++){

w = x[i - 1] == y[j - 1] ? 0 : 1;

top = matr[i - 1][j];

left = matr[i][j - 1];

left\_top = matr[i - 1][j - 1];

matr[i][j] = std::min(left\_top + w, std::min(top + 1, left + 1));

}

return matr[lx-1][ly-1];

}

Пример реализации рекурсивным методом:

int min3(int x1, int x2, int x3)

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

int levenshtein\_r(int lx, const char x[],

int ly, const char y[])

{

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;

};

На рисунке 2 представлены дистанции Левенштейна вычисленные при помощи метода динамического программирования, а также рекурсивным алгоритмом.

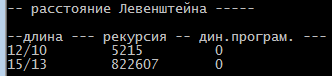


Рис. 2 – проверка работоспособности решений

**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).

**Решение:**

На графике, представленном на рисунке 3, можно заметить, что выполненные с помощью динамического алгоритма, вычисления производятся в разы быстрее, чем с помощью рекурсивного алгоритма.

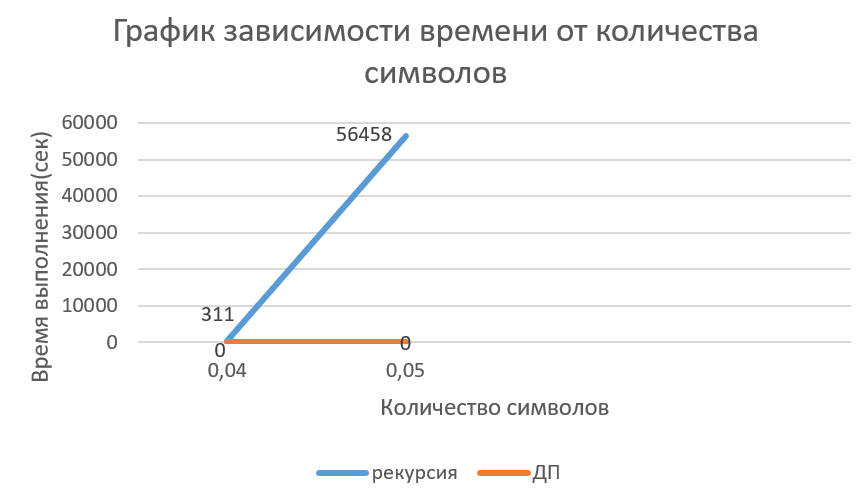


Рис. 3 – график зависимости времени от количества символов

**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 3 | |
| Лом | Гомон |

1. Лом, Гомон = min(Ло, Гомон+1)

( Лом, Гомо+1)

(Ло, Гомо+1)

1. Ло, Гомон = min(Л, Гомон+1)

(Ло, Гомо+1)

(Л, Гомо+1)

1. Лом, Гомо = min(Ло, Гомо+1)

(Лом, Гом+1)

(Ло, Гом+1)

1. Ло, Гомо = min(Л, Гомо+1)

(Ло, Гом+1)

(Л, Гом)

1. Л, Гомон = min(« », Гомон) = 5

(Л, Гомо+1)

(«», Гомо) = 4

1. Л, Гомо = min («», Гомо) = 4

(Л, Гом+1)

(«», Гом) = 3

1. Лом, Гом = min(Ло, Гом+1)

(Лом, Го+1)

(Ло, Го)

1. Ло, Гом = min(Л, Гом+1)

(Ло, Го+1)

(Л, Го+1)

1. Л, Гом = min(«», Гом) = 3

(Л, Го+1)

(«», Го) = 2

1. Лом, Го = min(Ло, Го+1)

(Лом, Г+1)

(Ло, Г+1)

1. Ло, Го = min(Л, Го+1)

(Ло, Г+1)

(Л, Г)

1. Л, Го = min(«», Го) = 2

(Л, Г+1)

(«», Г) = 1

1. Лом, Г = min(Ло, Г+1)

(Лом, «») = 3

(Ло, «») = 2

1. Ло, Г = min(Л, Г+1)

(Ло, «») = 2

(Л, «») = 1

1. Л, Г = min(«», Г) = 1

(Л, «») = 1

(«», «») =0

1. Л, Г = min (2, 2, 1) = 1
2. Ло, Г = min (2, 3, 2) = 2
3. Лом, Г = min (3, 4, 3) = 3
4. Л, Го = min (3, 2, 2) = 2
5. Ло, Го = min (3, 3, 1) = 1
6. Лом, Го = min (2, 4, 3) = 2
7. Л, Гом = min (4, 3, 3) = 3
8. Ло, Гом = min (4, 2, 3) = 2
9. Лом, Гом = min (3, 3, 1) = 1
10. Л, Гомо = min (5, 4, 4) = 4
11. Л, Гомон = min (6, 5, 5) = 5
12. Ло, Гомо = min (5, 3, 4) = 3
13. Лом, Гомо = min (5, 4, 2) = 2
14. Ло, Гомон = min (6, 3, 4) = 3
15. Лом, Гомон = min (4, 3, 4) = 3

**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 с соответствии с вариантом. Построить графики зависимости времени вычисления от k. Отобразить ход решения в отчете (по примеру лекции) + код + копии экрана.

|  |  |
| --- | --- |
| Вариант 3 | |
| ABCDFGI | EATUFI |

**Решение:**

// - LCS.h

// -- рекурсивное вычисление длины LCS

int lcs ( int lenx, // длина последовательности X

const char x[], // последовательность X

int leny, // длина последовательности Y

const char y[] // последовательность Y

);

// -- динамическое вычисление LCS

int lcsd( const char x[], // последовательность X

const char y[], // последовательность Y

char z[] // наибольшая общая подпоследовательность

);

// - LCS.cpp

// -- рекурсивное вычисление длины LCS

#include "stdafx.h"

#include <algorithm>

#include "LCS.h"

int lcs (int lenx, const char x[],

int leny, const char y[])

{

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[])

{

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

// -- вычисления длины LCS

#include "stdafx.h"

#include <iostream>

#include "LCS.h"

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

{

setlocale(LC\_ALL, "rus");

char z[100]="";

char X[] = "ABCDFGI";

char 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, " BXWAFRE", sizeof(Y)-1, " XCDUFR" );

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

// наибольшая общая подпоследовательность

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: " << x;

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

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

std::cout<<std::endl;

system("pause");

return 0;

}

На рисунке 3 представлена наибольшая общая подпоследовательность последовательностей Х и У, вычисленные при помощи метода динамического программирования, а также рекурсивным алгоритмом.

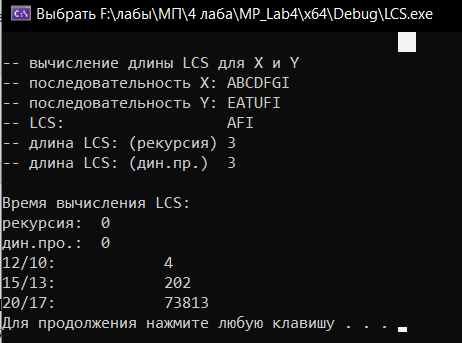


Рис. 3 – проверка работоспособности решений

На графике, представленном на рисунке 4, можно заметить, что выполненные с помощью динамического алгоритма, вычисления производятся в разы быстрее, чем с помощью рекурсивного алгоритма.

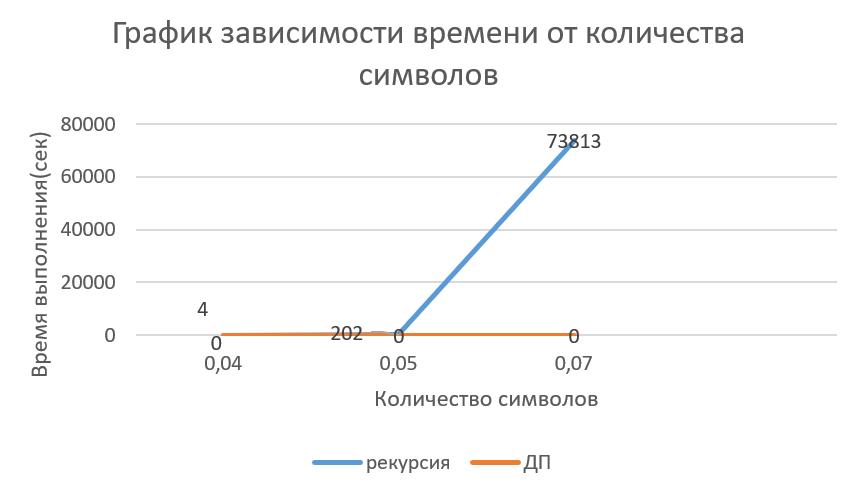


Рис. 4 – сравнительный анализ времени выполнения

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | A -1 | B -2 | C - 3 | D - 4 | F - 5 | G - 6 | I - 7 |
|  | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| E -1 | 0 | ↑0 | ↑0 | ↑0 | ↑0 | ↑0 | ↑0 | ↑0 |
| A - 2 | 0 | ↖1 | ←1 | ←1 | ←1 | ←1 | ←1 | ←1 |
| T - 3 | 0 | ↑1 | ←1 | ←1 | ←1 | ←1 | ←1 | ↑1 |
| U - 4 | 0 | ↑1 | ←1 | ←1 | ←1 | ←1 | ←1 | ↑1 |
| F - 5 | 0 | ↑1 | ←1 | ←1 | ←1 | ↖2 | ←2 | ←2 |
| I - 6 | 0 | ↑1 | ←1 | ←1 | ←1 | ↑2 | ↑2 | ↖3 |

A F I

1. С 7,6 = c 6,5 +1
2. C 6,5 = max (c 5,5, c 6,4)
3. C 5,5 = c 4,4 +1
4. C 6,4 = max (c 6,3, c 5,4)
5. C 4,4 = max (c 3,4, c 4,3)
6. C 6,3 = max (c 6,2, c 5,3)
7. C 5,4 = max (c 5,3, c 4,4)
8. C 3,4 = max (c 3,3, c 2,4)
9. C 4,3 = max (c 4,2, c 3,3)
10. C 6,2 = max (c 6,1, c 5,2)
11. C 5,3 = max (c 5,2, c 4,3)
12. C 3,3 = max (c 3,2, c 2,3)
13. C 2,4 = max (c 2,3, c 1,4)
14. C 4,2 = max (c 4,1, c 3,2)
15. C 6,1 = max (c 6,0, c 5,1) = c 5,1
16. C 5,2 = max (5,1 4,2)
17. C 3,2 = max (3,1 2,2)
18. C 2,3 = max (2,2 1,3)
19. C 1,4 = max (1,3 0,4) = c 1,3
20. C 5,1 = max (5,0 4,1) = c 4,1
21. C 4,1 = max (4,0 3,1) = 3,1
22. C 3,1 = max (3,0 2,1) = c 2,1
23. C 2,2 = max (2,1 1,2)
24. C 1,3 = max (1,2 0,3) = c 1,2
25. C 2,1 = max (2,0 1,1) = c 1,1
26. C 1,2 = c 0,1 +1 = 1
27. C 1,1 = max (0,1 1,0) = 0
28. C 2,1 = 0
29. C 1,3 = 1
30. C 2,2 = max (1,0) = 1
31. C 3,1 = 0
32. C 4,1 = 0
33. C 5,1 = 0
34. C 1,4 = 1
35. C 2,3 = max (1,1) =1
36. C 3,2 = max (1, 0) = 1
37. C 5,2 = max (1, 0) = 1
38. C 6,1 = 0
39. C 4,2 = max (1, 0) = 1
40. C 2,4 = max (1, 1) = 1
41. C 33 = max (1, 1) = 1
42. C 53 = max (1, 1) = 1
43. C 62 = max (1, 0) = 1
44. C 43 = max (1, 1) = 1
45. C 34 = max (1, 1) = 1
46. C 54 = max (1, 1) = 1
47. C 63 = max (1, 1) = 1
48. C 44 = max (1, 1) = 1
49. C 64 = max (1, 1) = 1
50. C 55 = 1+1 = 2
51. C 65 = max (2, 1) = 2
52. C 76 = 2+1 = 3