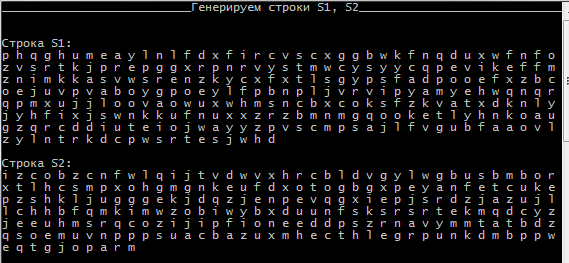
Математическое программирование

Лабораторная работа №4 – ДИНАМИЧЕСКОЕ ПРОГРАММИРОВАНИЕ

Вариант №7

**Задание 1.**

На языке С++ сгенерировать случайным образом строку букв латинского алфавита  длиной  символов и длиной .



**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).

cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Дистанция Левенштейна\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << endl << endl;

int dist = 0;

clock\_t t1 = 0, t2 = 0;

float k[] = { 0.04, 0.05, 0.0625, 0.1, 0.2, 0.5, 1 };

for (int i = 0; i < 300\* k[0]; i++) cout << S1[i] << " ";

cout << endl << endl;

for (int i = 0; i < 250\* k[0]; i++) cout << S2[i] << " ";

t1 = clock();

dist = levenshtein(300 \* k[0], S1, 250 \* k[0], S2);

t2 = clock();

cout << endl << endl;

cout << "Дистанция Левенштейна с помощью динамического программирования: " << dist << endl;

cout << "Время вычисления алгоритма: (сек) " << ((double)(t2 - t1)) / ((double)CLOCKS\_PER\_SEC) << endl;

t1 = clock();

dist = levenshtein\_r(300 \* k[0], S1, 250 \* k[0], S2);

t2 = clock();

cout << "Дистанция Левенштейна с помощью рекурсии: " << dist << endl;

cout << "Время вычисления алгоритма: (сек) " << t2 - t1<< endl;

//system("pause");

cout << endl << endl;

for (int i = 0; i < 300 \* k[1]; i++) cout << S1[i] << " ";

cout << endl << endl;

for (int i = 0; i < 250 \* k[1]; i++) cout << S2[i] << " ";

t1 = clock();

dist = levenshtein(300 \* k[1], S1, 250 \* k[1], S2);

t2 = clock();

cout << endl << endl;

cout << "Дистанция Левенштейна с помощью динамического программирования: " << dist << endl;

cout << "Время вычисления алгоритма: (сек) " << ((double)(t2 - t1)) / ((double)CLOCKS\_PER\_SEC) << endl;

t1 = clock();

dist = levenshtein\_r(300 \* k[1], S1, 250 \* k[1], S2);

t2 = clock();

cout << "Дистанция Левенштейна с помощью рекурсии: " << dist << endl;

cout << "Время вычисления алгоритма: (сек) " << ((double)(t2 - t1)) / ((double)CLOCKS\_PER\_SEC) << endl;

//system("pause");

cout << endl << endl;

for (int i = 0; i < 300 \* k[2]; i++) cout << S1[i] << " ";

cout << endl << endl;

for (int i = 0; i < 250 \* k[2]; i++) cout << S2[i] << " ";

t1 = clock();

dist = levenshtein(300 \* k[2], S1, 250 \* k[2], S2);

t2 = clock();

cout << endl << endl;

cout << "Дистанция Левенштейна с помощью динамического программирования: " << dist << endl;

cout << "Время вычисления алгоритма: (сек) " << ((double)(t2 - t1)) / ((double)CLOCKS\_PER\_SEC) << endl;

t1 = clock();

dist = levenshtein\_r(300 \* k[2], S1, 250 \* k[2], S2);

t2 = clock();

cout << "Дистанция Левенштейна с помощью рекурсии: " << dist << endl;

cout << "Время вычисления алгоритма: (сек) " << ((double)(t2 - t1)) / ((double)CLOCKS\_PER\_SEC) << endl;

//system("pause");

cout << endl << endl;

for (int i = 0; i < 300 \* k[3]; i++) cout << S1[i] << " ";

cout << endl << endl;

for (int i = 0; i < 250 \* k[3]; i++) cout << S2[i] << " ";

t1 = clock();

dist = levenshtein(300 \* k[3], S1, 250 \* k[3], S2);

t2 = clock();

cout << endl << endl;

cout << "Дистанция Левенштейна с помощью динамического программирования: " << dist << endl;

cout << "Время вычисления алгоритма: (сек) " << ((double)(t2 - t1)) / ((double)CLOCKS\_PER\_SEC) << endl;

t1 = clock();

dist = levenshtein\_r(300 \* k[3], S1, 250 \* k[3], S2);

t2 = clock();

cout << "Дистанция Левенштейна с помощью рекурсии: " << dist << endl;

cout << "Время вычисления алгоритма: (сек) " << ((double)(t2 - t1)) / ((double)CLOCKS\_PER\_SEC) << endl;

//system("pause");

cout << endl << endl;

for (int i = 0; i < 300 \* k[4]; i++) cout << S1[i] << " ";

cout << endl << endl;

for (int i = 0; i < 250 \* k[4]; i++) cout << S2[i] << " ";

t1 = clock();

dist = levenshtein(300 \* k[4], S1, 250 \* k[4], S2);

t2 = clock();

cout << endl << endl;

cout << "Дистанция Левенштейна с помощью динамического программирования: " << dist << endl;

cout << "Время вычисления алгоритма: (сек) " << ((double)(t2 - t1)) / ((double)CLOCKS\_PER\_SEC) << endl;

t1 = clock();

dist = levenshtein\_r(300 \* k[4], S1, 250 \* k[4], S2);

t2 = clock();

cout << "Дистанция Левенштейна с помощью рекурсии: " << dist << endl;

cout << "Время вычисления алгоритма: (сек) " << ((double)(t2 - t1)) / ((double)CLOCKS\_PER\_SEC) << endl;

//system("pause");

cout << endl << endl;

for (int i = 0; i < 300 \* k[5]; i++) cout << S1[i] << " ";

cout << endl << endl;

for (int i = 0; i < 250 \* k[5]; i++) cout << S2[i] << " ";

t1 = clock();

dist = levenshtein(300 \* k[5], S1, 250 \* k[5], S2);

t2 = clock();

cout << endl << endl;

cout << "Дистанция Левенштейна с помощью динамического программирования: " << dist << endl;

cout << "Время вычисления алгоритма: (сек) " << ((double)(t2 - t1)) / ((double)CLOCKS\_PER\_SEC) << endl;

t1 = clock();

dist = levenshtein\_r(300 \* k[5], S1, 250 \* k[5], S2);

t2 = clock();

cout << "Дистанция Левенштейна с помощью рекурсии: " << dist << endl;

cout << "Время вычисления алгоритма: (сек) " << ((double)(t2 - t1)) / ((double)CLOCKS\_PER\_SEC) << endl;

//system("pause");

cout << endl << endl;

for (int i = 0; i < 300 \* k[6]; i++) cout << S1[i] << " ";

cout << endl << endl;

for (int i = 0; i < 250 \* k[6]; i++) cout << S2[i] << " ";

t1 = clock();

dist = levenshtein(300 \* k[6], S1, 250 \* k[6], S2);

t2 = clock();

cout << endl << endl;

cout << "Дистанция Левенштейна с помощью динамического программирования: " << dist << endl;

cout << "Время вычисления алгоритма: (сек) " << ((double)(t2 - t1)) / ((double)CLOCKS\_PER\_SEC) << endl;

t1 = clock();

dist = levenshtein\_r(300 \* k[6], S1, 250 \* k[6], S2);

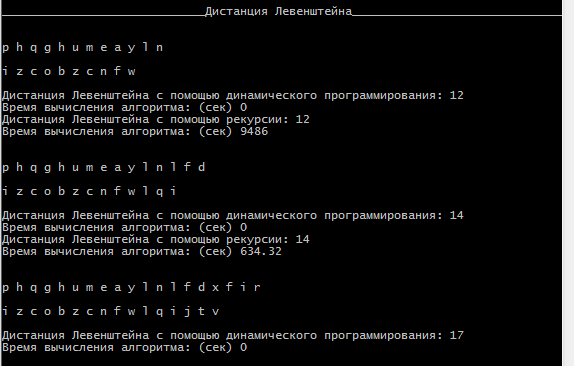
t2 = clock();

cout << "Дистанция Левенштейна с помощью рекурсии: " << dist << endl;

cout << "Время вычисления алгоритма: (сек)" << ((double)(t2 - t1)) / ((double)CLOCKS\_PER\_SEC) << endl;

system("pause");

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.

**Задание 4.**

Найти расстояние Левенштейна между словами «Кол» и «Столб».

1. L(«Кол», «Столб») = min

2. L(«Ко», «Столб») = min

3. L(«Кол», «Стол») = min

4. L(«Ко», «Стол») = min

5. L(«К», «Столб») = min

L(« », «Столб») = 5, L(« », «Стол») = 4

6. L(«К», «Стол») = min

L(« », «Стол») = 4, L(« », Сто») = 3

7. L(«Кол», «Сто») = min

8. L(«Ко», «Сто») = min

9. L(«К», «Сто») = min

L(« », «Сто») = 3, L(« », «Ст») = 2

10. L(«Кол», «Ст») = min

11. L(«Ко», «Ст») = min

L(«К», «С») = 1

12. L(«К», «Ст») = min

L(« », «Ст») = 2, L(«К», «С») = 1, L(« », «С») = 1

13. L(«Кол», «С») = min

L(«Кол», « ») = 3, L(«Ко», « ») = 2

14. L(«Ко», «С») = min

L(«К», «С») = 1, L(«Ко», « ») = 2, L(«К», « ») = 1

15. L(«Ко», «С») = min (2, 3, 2) = 2

16. L(«Кол», «С») = min (3, 4, 3) = 3

17. L(«К», «Ст») = min (3, 2, 2) = 2

18. L(«Ко», «Ст») = min (3, 3, 2) =2

19. L(«Кол», «Ст») = min (3, 4, 3) = 3

20. L(«К», «Сто») = min (4, 3, 3) = 3

21. L(«Ко», «Сто») = min (4, 3, 2) = 2

22. L(«Кол», «Сто») = min (3, 4,3) = 3

23. L(«К», «Стол») = min (5, 4 , 4) = 4

24. L(«К», «Столб») = min (6, 5, 5) = 5

25. L(«Ко», «Стол») = min (5, 3, 4) = 3

26. L(«Кол», «Стол») = min (4, 4, 2) = 2

27. L(«Ко», «Столб») = min (6, 3, 4) = 3

28. L(«Кол», «Столб») = min (4, 3, 4) = 3

ДИСТАНЦИЯ ЛЕВЕНШТЕЙНА ДЛЯ СЛОВ «КОЛ» И «СТОЛБ» = 3.

Проверка рекурсивным алгоритмом на C++:

const char Kol[] = { 'К', 'о', 'л' };

const char Stolb[] = { 'С', 'т', 'о', 'л', 'б' };

for (int i = 0; i < 3; i++) cout << Kol[i] << " "; cout << endl;

for (int i = 0; i < 5; i++) cout << Stolb[i] << " "; cout << endl;

dist = levenshtein\_r(3, Kol, 5, Stolb);

cout << "Дистанция Левенштейна для Kол и Столб: " << dist << endl << endl;

system("pause");



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

Даны две последовательности: QVTWNHO и RQTWYK.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | **Q** | **V** | **T** | **W** | **N** | **H** | **O** |
|  | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **R** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **Q** | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| **T** | 0 | 1 | 1 | 2 | 2 | 2 | 2 | 2 |
| **W** | 0 | 1 | 1 | 2 | 3 | 3 | 3 | 3 |
| **Y** | 0 | 1 | 1 | 2 | 3 | 3 | 3 | 3 |
| **K** | 0 | 1 | 1 | 2 | 3 | 3 | 3 | 3 |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | **Q** | **V** | **T** | **W** | **N** | **H** | **O** |
|  |  |  |  |  |  |  |  |  |
| **R** |  | ↑ | ↑ | ↑ | ↑ | ↑ | ↑ | ↑ |
| **Q** |  | ↖ | ← | ← | ← | ← | ← | ← |
| **T** |  | ↑ | ↑ | ↖ | ← | ← | ← | ← |
| **W** |  | ↑ | ↑ | ↑ | ↖ | ← | ← | ← |
| **Y** |  | ↑ | ↑ | ↑ | ↑ | ↑ | ↑ | ↑ |
| **K** |  | ↑ | ↑ | ↑ | ↑ | ↑ | ↑ | ↑ |

НАИБОЛЬШАЯ ОБЩАЯ ПОДПОСЛЕДОВАТЕЛЬНОСТЬ: **Q T W**

Проверка рекурсивным и динамичным алгоритмами на C++:

cout << endl << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Вычисление длины LCS для X и Y(рекурсия)\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_";

char X[] = "QVTWNHO", Y[] = "RQTWYK";

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

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

t1 = clock();

int s = lcs(

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

"QVTWNHO", // последовательность X

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

"RQTWYK" // последовательность Y

);

t2 = clock();

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

cout << endl << "Время выполнения: " << t2 - t1 << endl;

system("pause");

char x[] = "QVTWNHO",

y[] = "RQTWYK";

t1 = clock();

l = lcsd(x, y, z);

t2 = clock();

cout << endl

<< "\_\_\_Наибольшая общая подпоследовательость - LCS(динамическое программирование)\_\_\_" << endl;

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

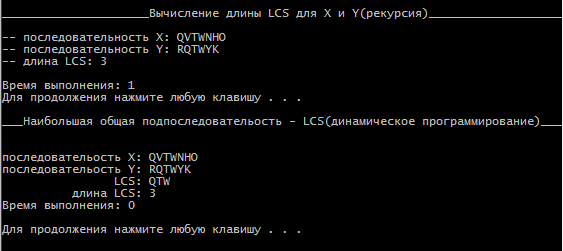
cout << endl << "последовательость Y: " << y;

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

cout << endl << " длина LCS: " << l;

cout << endl << "Время выполнения: " << t2 - t1 << endl;

cout << endl;



Исследуем зависимость времени работы алгоритма от количества символов в строках:

cout << endl << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_НАИБОЛЬШАЯ ОБЩАЯ ПОДПОСЛЕДОВАТЕЛЬНОСТЬ(LCS)\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_";

int res = 0, k;

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

for (k = 5; k < 15; k++)

{

for (int i = 0; i < k; i++) cout << S1[i] << " ";

cout << endl;

for (int i = 0; i < k; i++) cout << S2[i] << " ";

cout << endl<< endl;

t1 = clock();

res = lcs(k, S1, k, S2);

t2 = clock();

cout << "Длина строк: " << k << endl << "Длина LCS: " << res << endl << "Время: " << t2 - t1 << endl << endl;

}

cout << "Динамическое программирование" << endl;

char z[100] = "";

t1 = clock();

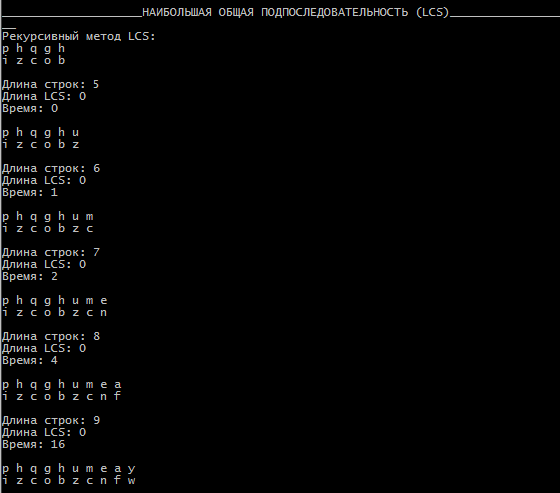
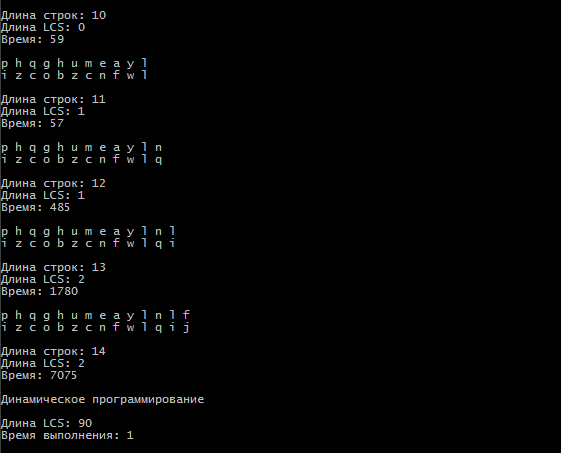
int l = lcsd(S1, S2, z);

t2 = clock();

cout << endl << "Длина LCS: " << l;

cout << endl << "Время выполнения: " << t2 - t1 << endl;

Вывод на консоли:

Метод динамического программирования справляется со строками длиной 300 и 250 символов за 3 единицы тактового времени, в то время как рекурсивный метод уже с длиной строк 14 считает почти 7 секунд (7075 единиц тактового времени). Таким образом, можно сделать вывод, что для решения задачи о нахождении наибольшей обшей подподследовательности динамическое программирование будет более эффективным методом, чем рекурсивный метод.