Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ

УНИВЕРСИТЕТ»

Отчёт по лабораторной работе №4

«**Динамическое программирование**»

Студентка: Змитревич Д.А.

ФИТ 2 курс 9 группа

Преподаватель: Барковский Е.В.

Минск 2022

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

**Задание 1**. На языке С++ сгенерировать случайным образом строку букв латинского алфавита S1 длиной 300 символов и S2 длиной 250 символов.

#include "Task\_1.h"

#include <stdlib.h>

char\* getString(int size)

{

srand(time(NULL));

char\* str = new char[size] {};

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

{

str[i] = 65 + rand() % 26;

}

return str;

}

Изображение выглядит как текст, бутылка, табличка

Автоматически созданное описание

Рисунок 4.1 – Результат выполнения первого задания

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

Дистанция Левенштейна – метрика, измеряющая разность между двумя последовательностями символов. Она определяется как минимальное количество односимвольных операций, необходимых для превращения одной последовательности символов в другую.

Рекурсивно данное вычисление производится с помощью функции

levenshtein\_r, которая принимает как параметры две строки х и у, а также их длины lx и ly соответственно.

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;

};

Динамически данное вычисление производится с помощью функции

levenshtein, которая принимает как параметры две строки х и у, а также их длины lx и ly соответственно.

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

{

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);

}

Обе функции возвращают количество правок, которые нужно сделать, для эквивалентности строк.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 4.2 – Результат выполнения второго задания

**Задание 3**. Выполнить сравнительный анализ времени затраченного на вычисление дистанции Левенштейна для двух методов решения. Построить графики зависимости времени вычисления от k.

Рисунок 4.3 – График зависимости времени вычисления от k

Метод динамического программирования значительно эффективнее рекурсивного метода, т.к. выполняется намного быстрее.

Это происходит по той причине, что в методе динамического программирования мы должны рассмотреть полиноминальное количество вариантов, пока не найдем верное решение, а в методе рекурсии перебор является экспоненциальным.

**Задание 4**. Реализовать вручную пример вычисления дистанции Левенштейна при помощи рекурсивного алгоритма (в соответствии с вариантом).

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

12. L("К", "Ст") = min

L("", "Ст") = 2

L("", "С") = 1

13. L("Кол", "С") = min

L("Кол", "") = 3

L("Ко", "") = 2

14. L("Ко", "С") = min

L("Ко", "") = 2

L("К", "") = 1

15. L("К", "С") = min

L("", "С") = 1

L("К", "") =1

L("", "") = 0

16. L("К", "С") = min(2,2,1) = 1

17. L("Ко", "С") = min(2, 3, 2) = 2

18. L("Кол", "С") = min(3, 4, 3) = 3

19. L("К", "Ст") = min(3, 2, 2) = 2

20. L("Ко", "Ст") = min(3, 3, 1) = 1

21. L("Кол", "Ст") = min(2, 4, 3) = 2

22. L("К", "Сто") = min(4, 3, 3) = 3

23. L("Ко", "Сто") = min(4, 2, 3) = 2

24. L("Кол", "Сто") = min(3, 3, 1) = 1

25. L("К", "Стол") = min(5, 4, 4) = 4

26. L("К", "Столб") = min(6, 5, 5) = 5

27. L("Кол", "Сто") = min(4, 2, 3) = 2

28. L("Ко", "Стол") = min(3, 3, 4) = 3

29. L("Кол", "Стол") = min(4, 2, 3) = 2

30. L("Ко", "Столб") = min(4, 4, 3) = 3

31. L ("Кол", "Столб") = min(3, 4, 3) = 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 в соответствии с вариантом. Построить графики зависимости времени вычисления от k.

|  |  |
| --- | --- |
| QVTWNHO | RQTWYK |

// LCS.cpp

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 = lcs(lenx - 1, x, leny - 1, y) + 1;

else

rc = max(lcs(lenx, x, leny - 1, y), lcs(lenx - 1, x, leny, y));

}

return rc; //длина LCS

}

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);

}

// LB4.cpp

void Task5()

{

clock\_t t1 = 0;

clock\_t t2 = 0;

clock\_t t3 = 0;

clock\_t t4 = 0;

char Z[100] = "";

char X[] = "QVTWNHO";

char Y[] = "RQTWYK";

t1 = clock();

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

t2 = clock();

t3 = clock();

int l = lcsd(X, Y, Z);

t4 = clock();

cout << "\n-- вычисление длины LCS для X и Y";

cout << "\n-- последовательность X: " << X;

cout << "\n-- последовательность Y: " << Y;

cout << "\n-- LCS: " << Z;

cout << "\n-- длина LCS: (рекурсия) " << s;

cout << "\n-- длина LCS: (дин.пр.) " << l << "\n";

cout << "\nВремя вычисления LCS:\n";

cout << "рекурсия: " << (t2 - t1);

cout << "\nдин.про.: " << (t4 - t3) << "\n";

}

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 4.4 – Результат выполнения пятого задания

Рисунок 4.5 – График зависимости времени вычисления различными методами от количества символов

На графике выше можно заметить, что вычисления выполненные с помощью динамического алгоритма производятся в разы быстрее, чем с помощью рекурсивного алгоритма.

**Вывод**: в ходе данной работы были освоены общие принципы решения задач методом динамического программирования и сравнены полученные решения задач с рекурсивным методом.