Белорусский государственный технологический университет

Факультет информационных технологий

Кафедра программной инженерии

 Лабораторная работа № 4

По дисциплине «Математическое программирование»

На тему «Динамическое программирование»

Выполнила:

Студентка 2 курса 10 группы

Рублевская Маргарита Владимировна

Преподаватель: асс. Ромыш А.С.

2025, Минск

**Вариант 10**

**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 200. Код приведен в листинге 1.1.

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

void main()

{

srand(time(NULL));

char abc[25];

char s1[300], s2[250];

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

{

abc[n] = (char)i;

}

std::cout << "S1 = ";

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

{

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

if (i % 100 == 0)

std::cout << std::endl;

std::cout << s1[i];

}

std::cout << std::endl << std::endl << "S2 =";

for (int i = 0; i < 251; i++)

{

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

if (i % 100 == 0)

std::cout << std::endl;

std::cout << s2[i];

}

std::cout << std::endl;

}

Листинг 1.1. – Генерация строк

Результат выполнения программы представлен на рисунке 1.1.

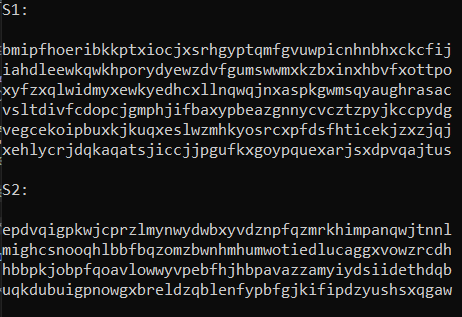
****

Рисунок 1.1. – Результат работы программы

**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). Код приведен в листинге 1.2.

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\* 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[] )

{

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;

};

Листинг 1.2. – Вычисление расстояния Левенштейна

Результат выполнения программы представлен на рисунке 1.2.

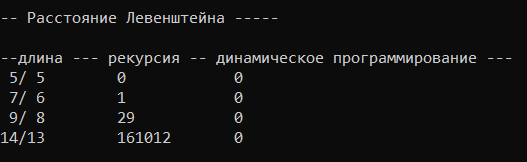


Рисунок 1.2. – Результат работы программы

**Задание № 3**

Выполнить сравнительный анализ времени, затраченного на вычисление дистанции Левенштейна для двух методов решения. Построить графики зависимости времени вычисления от . На рисунке 1.3 представлены графики зависимости времени вычисления от *k*.

****

Рисунок 1.3. – Графики зависимости

**Задание № 4**

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

|  |  |
| --- | --- |
| Задание 4 | |
| Ель | Дрель |

1.  
2.  
3.  
4.  
5.  

 = 5.

 = 4.

1.  

 = 4.

 = 3.

1.  
2.  
3.  

 = 3.

 = 2.

1.  
2.  
3.  

 = 2.

 = 1.

1.  

 = 3.

 = 2.

1.  

 = 2.

 = 1.

1.  

 = 1.

 = 1.

 = 0.

1. 
2. 
3. 
4. 
5. 
6. 
7. 
8. 
9. 
10. 
11. 
12. 
13. 
14. 
15. 

Дистанция Левенштейна для «ель» и «дрель» равно 2.

**Задание № 5**

Выполнить сравнительный анализ времени, затраченного на решение задачи об оптимальной расстановке скобок при умножении нескольких матриц для двух методов решения (рекурсивное решение, динамическое программирование). Код приведен в листинге 1.3.

#include "stdafx.h"

#include <memory.h>

#include "MultyMatrix.h"

// расстановка скобок (рекурсия)

#define INFINITY 0x7fffffff

#define NINFINITY 0x80000000

int OptimalM(int i, int j, int n, const int c[], int\* s)

{

#define OPTIMALM\_S(x1,x2) (s[(x1-1)\*n+x2-1])

int o = INFINITY; int bo = INFINITY;

if (i < j)

{

for (int k = i; k < j; k++)

{

bo = OptimalM(i, k, n, c, s) + OptimalM(k + 1, j, n, c, s) + c[i - 1] \* c[k] \* c[j];

if (bo < o)

{

o = bo; OPTIMALM\_S(i, j) = k;

}

}

}

else o = 0; return o;

#undef OPTIMALM\_S

};

// расстановка скобок (динамическое программирование)

int OptimalMD(int n, const int c[], int\* s)

{

#define OPTIMALM\_S(x1,x2) (s[(x1-1)\*n+x2-1])

#define OPTIMALM\_M(x1,x2) (M[(x1-1)\*n+x2-1])

int\* M = new int[n \* n], j = 0, q = 0;

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

OPTIMALM\_M(i, i) = 0;

for (int l = 2; l <= n; l++)

{

for (int i = 1; i <= n - l + 1; i++)

{

j = i + l - 1;

OPTIMALM\_M(i, j) = INFINITY;

for (int k = i; k <= j - 1; k++)

{

q = OPTIMALM\_M(i, k) + OPTIMALM\_M(k + 1, j) + c[i - 1] \* c[k] \* c[j];

if (q < OPTIMALM\_M(i, j))

{

OPTIMALM\_M(i, j) = q; OPTIMALM\_S(i, j) = k;

}

}

}

}

return OPTIMALM\_M(1, n);

#undef OPTIMALM\_M

#undef OPTIMALM\_S

};

Листинг 1.3. – Решение задачи о расстановке скобок

Результат выполнения программы представлен на рисунке 1.4.

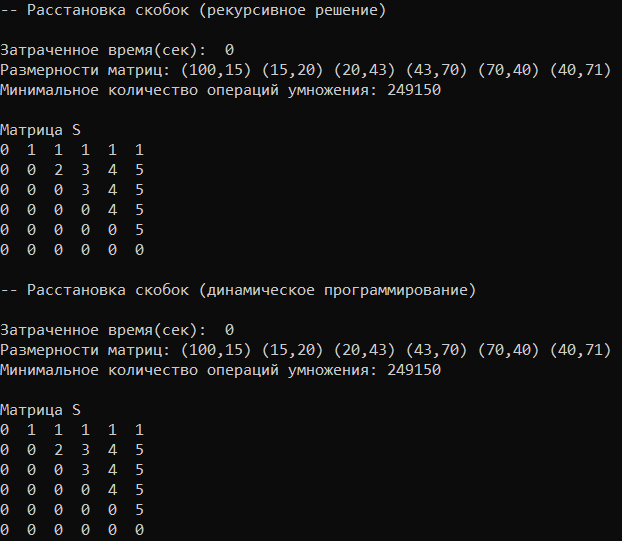
****

Рисунок 1.4. – Результат работы программы

Принцип расстановки скобок по итоговой матрице:

Скобки расставляются по принципу «сначала внешние – затем внутренние». Имеется 6 матриц, вот их размерность:

А1=100\*15,

А2=15\*20,

А3=20\*43,

А4 =43\*70,

А5 =70\*40,

А6 =40\*71.

Матрица S:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **1** | **2** | **3** | **4** | **5** | **6** |
| **1** | 0 | 1 | 1 | 1 | 1 | 1 |
| **2** | 0 | 0 | 2 | 3 | 4 | 5 |
| **3** | 0 | 0 | 0 | 3 | 4 | 5 |
| **4** | 0 | 0 | 0 | 0 | 4 | 5 |
| **5** | 0 | 0 | 0 | 0 | 0 | 5 |
| **6** | 0 | 0 | 0 | 0 | 0 | 0 |

Найдем элемент (1,6) в матрице S, он равен 1. Это означает, что точка разрыва между 1-ой и 6-ой матрицей находится после 1-ой матрицы. Что позволяет расставить скобки следующим образом:

A1\*(A2\*A3\*A4\*A5\*A6)

Точку разрыва между второй и шестой матрицей определяет элемент (2,6). Он равен 5. Следовательно разрыв будет после 5-ой матрицы.

A1\*((A2\*A3\*A4\*A5) \*A6)

Далее берем элемент (2,5) и получаем, что он равен 4. Следовательно получаем:

A1\*(((A2\*A3\*A4) \*A5) \*A6)

И на последнем шаге мы возьмем элемент (2,4) и он равен 3:

A1\*((((A2\*A3) \*A4) \*A5) \*A6)

Это выражение и есть конечное.

Полученная расстановка скобок позволяет получить минимальное количество операций умножения, равное 249150.

**Вывод:** в результате выполнения лабораторной работы были освоены общие принципы решения задач методом динамического программирования. Были изучены его основные этапы и принципы работы алгоритмов. Были рассмотрены примеры решения задач методом динамического программирования и сравнены с рекурсивным методом.