Учреждение образования

«Белорусский государственный технологический университет»

**Лабораторная работа №4**

**Динамическое программирование**

Выполнил:

Студент 2 курса 7 группы ФИТ

Володькин Никифор Дмитриевич

**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.

Была реализована программа на языке C++ для генерация случайных строк.

#include <iostream>

#include <string>

int main()

{

srand(time(NULL));

std::string S1 = "";

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

{

S1 += ('a' + rand() % 26);

}

std::string S2 = "";

for (int i = 0; i < 200; i++)

{

S2 += ('a' + rand() % 26);

}

std::cout << "S1: " << S1 << std::endl;

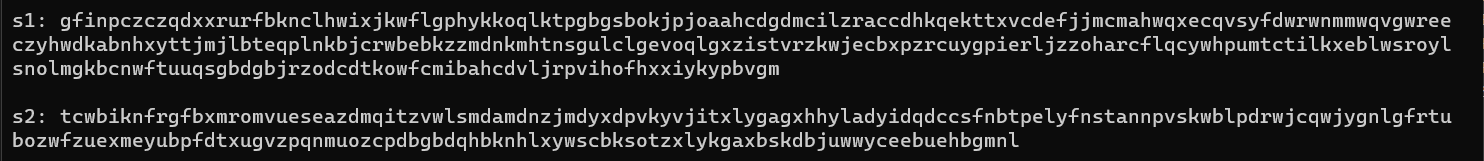
std::cout << std::endl;

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

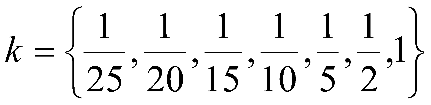
std::cout << std::endl;

}

Генерация строк на языке C++



Результат выполнения программы

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

Были разработаны функции для вычисления дистанции Левенштейна.

#include <iostream>

#include <iomanip>

#include <algorithm>

#include <string>

#include "Levenshtein.h"

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

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

{

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

}

int levenshtein(int lx, std::string x, int ly, std::string 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, std::string x, int ly, std::string 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,

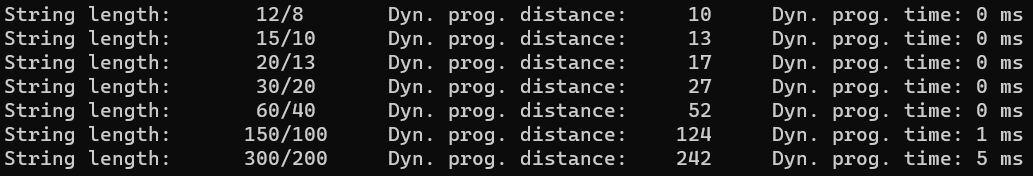
levenshtein\_r(lx, x, ly - 1, y) + 1,

levenshtein\_r(lx - 1, x, ly - 1, y) + (x[lx - 1] == y[ly - 1] ? 0 : 1)

);

return rc;

};

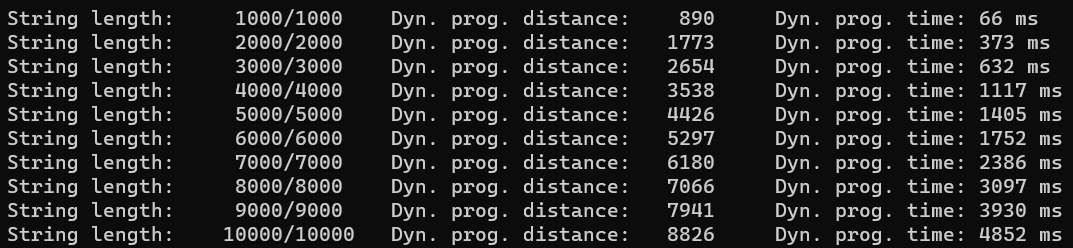


Результат вычисления дистанции Левенштейна

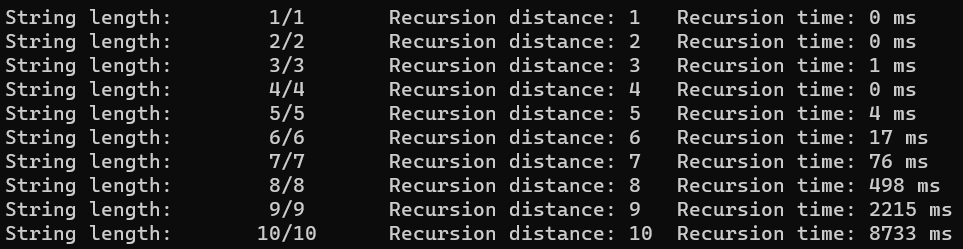
Из-за большой ресурсоемкости не удается вычислить дистанцию Левенштейна для строк такой длины рекурсивным методом.

***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).

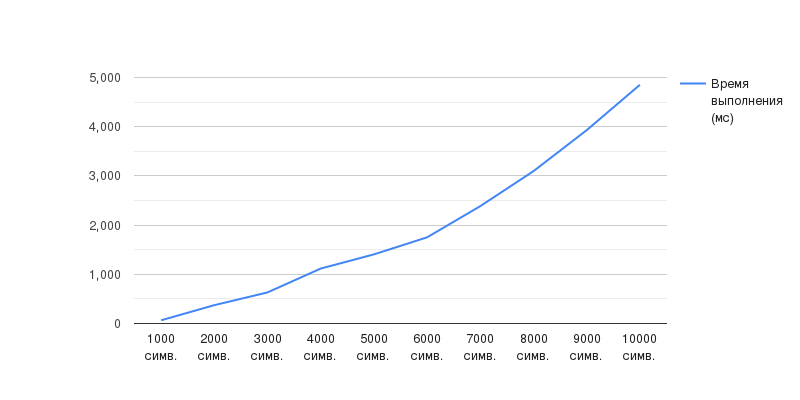
Из-за большой разности в производительности были использованы различные длины строк для демонстрации разницы производительности двух алгоритмов.



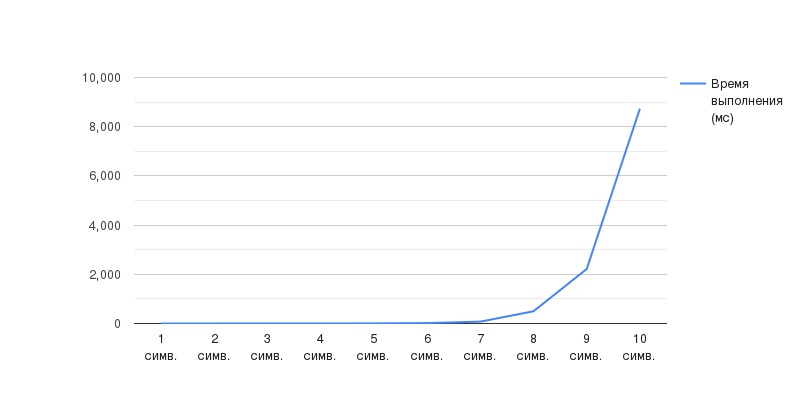
Результат вычисления с помощью динамического программирования



Результат вычисления с помощью рекурсии



Зависимость времени выполнения алгоритма динамического программирования от количества символов



Зависимость времени выполнения алгоритма динамического программирования 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).

1. L(“акр”, “якорь”) = min  
  
2. L("ак", "якорь") = min  
3. L(“акр”, “якор”) = min  
  
4. L(“ак”, “якор”) = min

5. L(“а”, “якорь”) = min  
  
L(“”, “”) = 5, L(“”, “ ”) = 4  
  
6. L(“а”, “якор”) = min  
  
L(“”, “якор”) = 4, L(“”, “яко”) = 3  
  
7. L(“акр”, “яко”) = min  
  
8. L(“ак”, “яко”) = min  
  
9. L(“акр”, “як”) = min  
  
10. L(“акр”, “я”) = min  
  
L(“акр”, “”) = 3, L(“ак”, “”) = 2  
  
11. L(“а”, “яко”) = min  
  
L(“”, “яко”) = 3, L(“”, “як”) = 2  
  
12. L(“ак”, “як”) = min  
  
L(“а”, “я”) = 1  
  
13. L(“а”, “як”) = min  
  
L(“”, “як”) = 2, L(“а”, “я”) = 1, L(“”, “я”) = 1  
  
14. L(“ак”, “я”) = min  
  
L(“а”, “я”) = 1, L(“ак”, “”) = 2, L(“а”, “”) = 1  
  
15. L(“ак”, “я”) = min(2, 3, 2) = 2  
  
16. L(“а”, “як”) = min(3, 2, 2) = 2  
  
17. L(“ак”, “як”) = min(3, 3, 1) = 1  
  
18. L(“а”, “яко”) = min(4, 3, 3) = 3  
  
19. L(“акр”, “я”) = min(3, 4, 3) = 3  
  
20. L(“акр”, “як”) = min(2, 4, 3) = 2  
  
21. L(“ак”, “яко”) = min(4, 2, 3) = 2  
  
22. L(“акр”, “яко”) = min(3, 3, 1) = 1  
  
23. L(“а”, “якор”) = min(5, 4, 4) = 4  
  
24. L(“а”, “якорь”) = min(6, 5, 5) = 5  
  
25. L(“ак”, “якор”) = min(5, 3, 4) = 3  
  
26. L(“акр”, “якор”) = min(4, 3, 3) = 3  
  
27. L(“ак”, “якорь”) = min(6, 4, 5) = 4  
  
28. L(“акр”, “якорь”) = min(5, 3, 4) = 3

Ответ: 3.

***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). Размерность матриц взять в соответствии с вариантом. Объяснить в отчете принцип расставления скобок по итоговой матрице + код + копии экрана.

#pragma once

// расстановка скобок при умножении матриц

// функции возвращают минимальное количество операций умножения

#define OPTIMALM\_PARM(x) ((int\*)x) // для представления 2мерного массива

int OptimalM( // рекурсия

int i, // [in] номер первой матрицы

int j, // [in] номер последней матрицы

int n, // [in] количество матриц

const int c[], // [in] массив размерностей

int\* s // [out] результат: позиции скобок

);

int OptimalMD( // динамическое программирование

int n, // [in] количество матриц

const int c[], // [in] массив размерностей

int\* s // [out] результат: позиции скобок

);

MultiMatrix.h

// расстановка скобок (рекурсия)

#include <memory.h>

#include "MultiMatrix.h"

#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, 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

};

// --- MultyMatrix.cpp (продолжение)

// 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

};

MultiMatrix.cpp

#include <algorithm>

#include <iostream>

#include <ctime>

#include <iomanip>

#include <cmath>

#include <memory.h>

#include <string>

#include "MultiMatrix.h"

#define N 6

using namespace std;

int main()

{

int Mc[N + 1] = { 20,15,30,53,10,20,11 }, Ms[N][N], r = 0, rd = 0;

memset(Ms, 0, sizeof(int) \* N \* N);

clock\_t t1, t2;

t1 = clock();

r = OptimalM(1, N, N, Mc, OPTIMALM\_PARM(Ms));

t2 = clock();

cout << endl;

cout << endl << "-- brackets (recursion) " << endl;

cout << endl << "matrix size: ";

for (int i = 1; i <= N; i++) cout << "(" << Mc[i - 1] << "," << Mc[i] << ") ";

cout << endl << "minimal amount of operations: " << r;

cout << endl << endl << "matrix S" << endl;

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

{

cout << endl;

for (int j = 0; j < N; j++) cout << Ms[i][j] << " ";

}

cout << endl;

cout << "time: " << (t2 - t1) << " ms" << endl;

memset(Ms, 0, sizeof(int) \* N \* N);

clock\_t t3, t4;

t3 = clock();

rd = OptimalMD(N, Mc, OPTIMALM\_PARM(Ms));

t4 = clock();

cout << endl

<< "-- brackets (dyn. prog.) " << endl;

cout << endl << "matrix size: ";

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

cout << "(" << Mc[i - 1] << "," << Mc[i] << ") ";

cout << endl << "minimal amount of operations: "

<< rd;

cout << endl << endl << "matrix S" << endl;

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

{

cout << endl;

for (int j = 0; j < N; j++) cout << Ms[i][j] << " ";

}

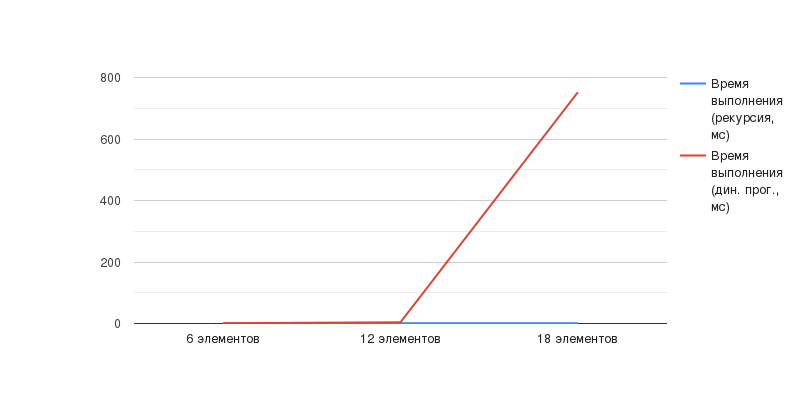
cout << endl;

cout << "time: " << (t4 - t3) << " ms" << endl;

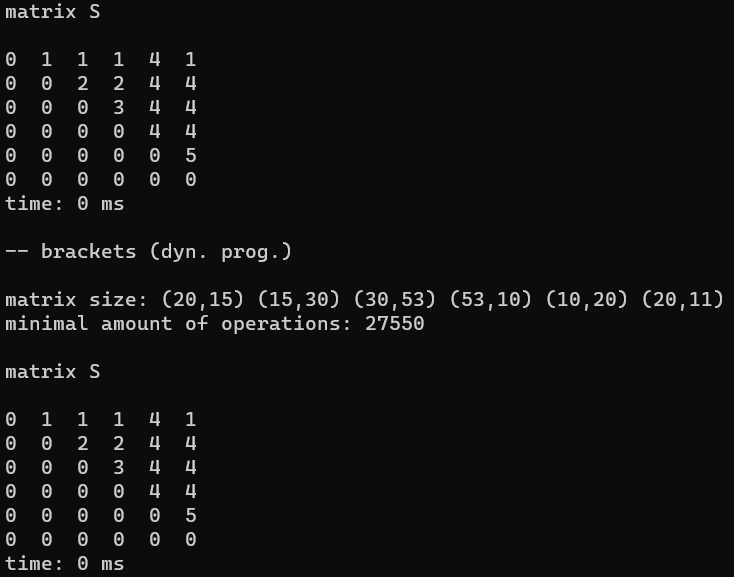
return 0;

}

Main.cpp



Сравнительный анализ времени выполнения двух алгоритмов



Результат выполнения программы

Проверка:

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

20\*15\*11 + 15\*10\*11 + 15\*30\*10 + 30\*53\*10 + 10\*20\*11 = 27550

**ВЫВОД: Освоены общие принципы решения задач методом динамического программирования, полученные решения задач сравнены с рекурсивным методом.**