Учреждение образования

«Белорусский государственный технологический университет»

**Лабораторная работа №4**

Динамическое программирование

Выполнил:

Студент 2 курса 5 группы ФИТ

Гайков Дмитрий Викторович

2023 г.

**Задание 1.**

#include <iostream>

#include <string>

using namespace std;

string generateString(uint16\_t len);

void main() {

setlocale(LC\_ALL, "ru");

srand(time(NULL));

string str300 = generateString(300);

string str200 = generateString(200);

cout << "Строка на 300 элементов: " << str300 << "\n\n";

cout << "Строка на 200 элементов: " << str200 << "\n\n";

}

string generateString(uint16\_t len) {

string res = ""s;

static const auto calc = [](\_\_int8 start, \_\_int8 end) {

return rand() % (end - start + 1) + start;

};

const char fstart = 'a';

const char fend = 'z';

const char sstart = 'A';

const char send = 'Z';

char symb;

\_\_int8 rnum;

for (uint16\_t i = 0; i < len; i++) {

rnum = rand() % 100 + 1;

symb = rnum > 50 ?

calc(fstart, fend) :

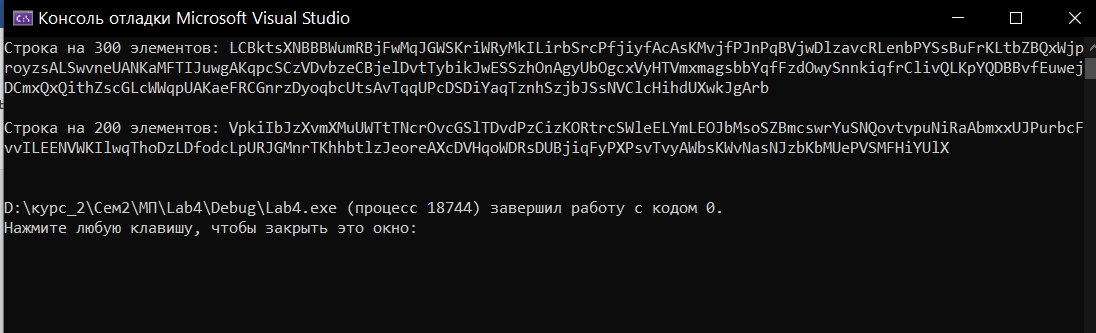
calc(sstart, send);

res += symb;

}

return res;

}



**Задание 2.**

#include <iostream>

#include <string>

#include <conio.h>

#include <iomanip>

#include "Ex2/Levenshtein.h"

using namespace std;

string generateString(uint16\_t len);

void main() {

setlocale(LC\_ALL, "ru");

srand(time(NULL));

string str300 = generateString(300);

string str200 = generateString(200);

cout << "Строка на 300 элементов: " << str300 << "\n\n";

cout << "Строка на 200 элементов: " << str200 << "\n\n";

clock\_t t1 = 0, t2 = 0, t3 = 0, t4 = 0;

int lx = sizeof(str300);

int ly = sizeof(str200);

int s1\_size[]{ 300 / 25, 300 / 20, 300 / 15, 300 / 10, 300 / 5, 300 / 2, 300 };

int s2\_size[]{ 200 / 25, 200 / 20, 200 / 15, 200 / 10, 200 / 5, 200 / 2, 200 };

std::cout << std::endl;

std::cout << std::endl << "-- расстояние Левенштейна -----" << std::endl;

std::cout << std::endl << "--длина --- рекурсия -- дин.програм. ---"

<< std::endl;

for (int i = 0; i < min(lx, ly); i++)

{

t1 = clock();

levenshtein\_r(s1\_size[i], str300.c\_str(), s2\_size[i], str200.c\_str());

t2 = clock();

t3 = clock();

levenshtein(s1\_size[i], str300.c\_str(), s2\_size[i], str200.c\_str());

t4 = clock();

cout << right << setw(2) << s1\_size[i] << "/" << setw(2) << s2\_size[i]

<< " " << left << setw(10) << (t2 - t1)

<< " " << setw(10) << (t4 - t3) << endl;

}

system("pause");

}

string generateString(uint16\_t len) {

string res = ""s;

static const auto calc = [](\_\_int8 start, \_\_int8 end) {

return rand() % (end - start + 1) + start;

};

const char fstart = 'a';

const char fend = 'z';

const char sstart = 'A';

const char send = 'Z';

char symb;

\_\_int8 rnum;

for (uint16\_t i = 0; i < len; i++) {

rnum = rand() % 100 + 1;

symb = rnum > 50 ?

calc(fstart, fend) :

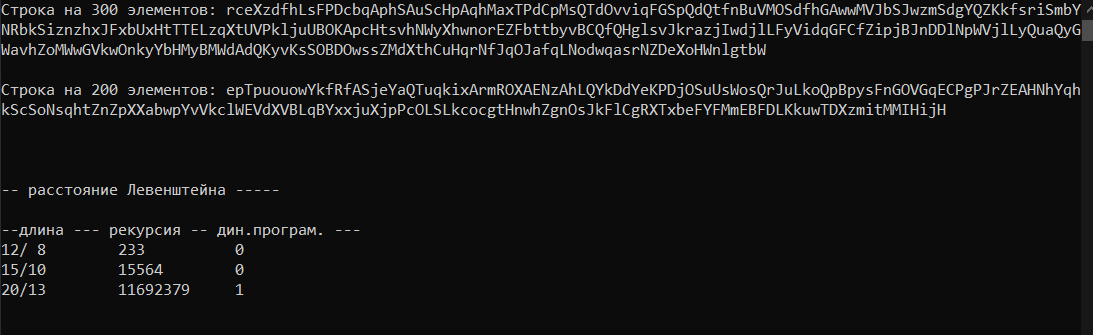
calc(sstart, send);

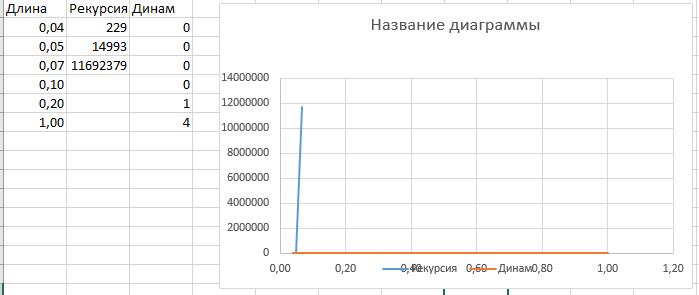
res += symb;

}

return res;

}





**Задание 4.**

1)

Акр Якорь

ø Я Як Яко Якор Якорь

ø 0 1 2 3 4 5

А 1 1 2 3 4 5

Ак 2 2 1 2 3 4

Акр 3 3 2 2 2 3

1. Акр, Якорь = min(Ак, Якорь + 1)(Акр, Якор + 1)(Ак, Якор + 1)

2. Ак, Якорь = min(А, Якорь + 1)(Ак, Якор + 1)(А, Якор + 1)

3. Акр, Якор = min(Ак, Якор + 1)(Акр, Яко + 1)(Ак, Яко + 1)

4. Ак, Якор = min(А, Якор + 1)

(Ак, Яко + 1)

(А, Яко + 1)

5. А, Якорь = min(«», Якорь) = 5

(А, Якор)

(«», Якор) = 4

6. А, Якор = min (“”, Якор) = 4

(А, Яко + 1)

(“”, Яко ) = 3

7. Акр, Яко = min (Ак, Яко) (Акр, Як) (Ак, Як)

8. Ак, Яко = min(А, Яко) (Ак, Як) (А, Як)

9. Акр, Як = min(Ак, Як) (Акр, Я) (Ак, Я)

10. Акр, Я = min (Ак, Я)

(Акр, «») = 3

(Ак, «» ) = 2

11. А, Яко = min (“” , Яко) = 3

(А, Як)

(«», Як) = 2

12. Ак, Як = min (А, Як) (Ак, Я) (А, Я)

13. А, Як = min («», Як) = 2 + 1 = 3

(А, Я) = 1 + 1 = 2 = 2

(«», Я ) = 1 + 1 = 2

14. Ак, Я = min (А, Я) = 1 + 1 = 2

(Ак, «» ) = 2 + 1 = 3 = 2

(А, «» ) = 1 + 1 = 2

15. Ак, Я = min (А, Я) = 1 + 1

(Ак, «») = 2 + 1 = 2

(А, «») = 1 + 1

16. А, Як = min («», Як) + 1 = 3

(А, Я) + 1 = 2 = 2

(«», Я) + 1 = 2

17. Ак, Як = min (А, Як) = 2 + 1 = 3

(Ак, Я) = 2 + 1 = 3 = 2

(А, Я) = 1 + 1 = 2

18. А, Яко = min («», Яко) + 1 = 4

(А, Як) = 2 + 1 = 3 = 3

(«», Як) + 1 = 3

19. Акр, Я = min (Ак, Я) = 2 + 1 = 3

(Акр, «») + 1 = 4 = 3

(Ак, «») + 1 = 3

20. Акр, Як = min (Ак, Як) = 2 + 1 = 3

(Акр, Я) = 3 + 1 = 4 = 3

(Ак, Я) = 2 + 1 = 3

21. Ак, Яко = min (А, Яко) = 3 + 1 = 4

(Ак, Як) = 2 + 1 = 3 = 3

(А, Як) = 2 + 1 = 3

22. Акр, Яко = min (Ак, Яко) = 4 + 1 = 5

(Акр, Як) = 3 + 1 = 4 = 3

(Ак, Як) = 2 + 1 = 3

23. А, Якор = min («», Якор) + 1 = 5

(А, Яко) = 3 + 1 = 4 = 4

(«», Яко) + 1 = 4

24. А, Якорь = min («», Якорь) + 1 = 6

(А, Якор) = 4 + 1 = 5 = 5

(«», Якор) + 1 = 5

25. Ак, Якор = min (А, Якор) = 4 + 1 = 5

(Ак, Яко) = 3 + 1 = 4 = 4

(А, Яко) = 3 + 1 = 4

26. Акр, Якор = min (Ак, Якор) = 3 + 1 = 4

(Акр, Яко) = 3 + 1 = 4

(Ак, Яко) = 2 + 1 = 3

27. Ак, Якорь = min (А, Якорь) = 5 + 1 = 6

(Ак, Якор) = 3 + 1= 4

(А, Якор) = 4 + 1= 4

28. Акр, Якорь = min(5, 3, 4) = 3

**Задание 5.**

**//main**

#include <cmath>

#include <memory.h>

#include <iostream>

#include "MultyMatrix.h" // умножение матриц

#define N 6

int main()

{

clock\_t tstart, tend,

timeR, timeD;

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);

tstart = clock();

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

tend = clock();

timeR = tend - tstart;

setlocale(LC\_ALL, "rus");

std::cout << std::endl;

std::cout << std::endl << "-- расстановка скобок (рекурсивное решение) "

<< std::endl;

std::cout << std::endl << "размерности матриц: ";

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

std::cout << std::endl << "минимальное количество операций умножения: " << r;

std::cout << std::endl << std::endl << "матрица S" << std::endl;

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

{

std::cout << std::endl;

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

}

std::cout << std::endl;

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

tstart = clock();

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

tend = clock();

timeD = tend - tstart;

std::cout << std::endl

<< "-- расстановка скобок (динамическое программирование) " << std::endl;

std::cout << std::endl << "размерности матриц: ";

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

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

std::cout << std::endl << "минимальное количество операций умножения: "

<< rd;

std::cout << std::endl << std::endl << "матрица S" << std::endl;

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

{

std::cout << std::endl;

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

}

std::cout << std::endl << std::endl;

std::cout << "Время затраченное на рекурсию: " << timeR << std::endl

<< "Время затраченное дин. программированием: " << timeD << "\n\n";

system("pause");

return 0;

}

#pragma once

// MultyMAtrix.h

// 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.cpp

// расстановка скобок (рекурсия)

#include <memory.h>

#include "MultyMatrix.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;

}

}

}

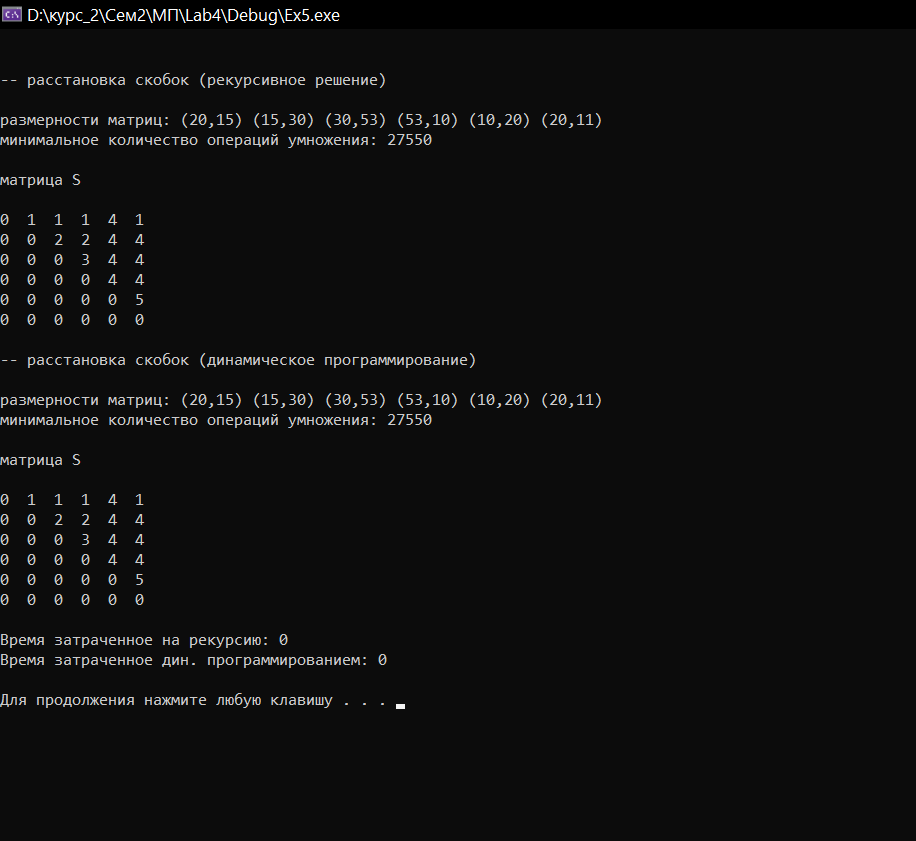
}

return OPTIMALM\_M(1, n);

#undef OPTIMALM\_M

#undef OPTIMALM\_S

};



**Вывод:** Для решения задачи о перемножении матриц нужно умножить матрицы последовательно в определенном порядке. В данном случае имеются 6 матриц, и возможны различные порядки их перемножения.

Для определения оптимального порядка перемножения матриц используется алгоритм динамического программирования. Он заключается в том, что матрицы умножаются в определенном порядке в зависимости от их размера. Таким образом, определяется минимальное количество операций умножения, необходимое для перемножения всех матриц.

При рекурсивном методе для вычисления оптимального порядка перемножения матриц используется метод повторного вызова, который имеет экспоненциальную сложность, что означает, что он неэффективен для больших данных. В данном случае, для нахождения оптимального порядка, необходимо рассмотреть все возможные порядки перемножения матриц, что может быть очень трудоемким процессом.

При использовании алгоритма динамического программирования время выполнения сокращается до квадратичной сложности, что дает значительный выигрыш в производительности. В данном случае, оптимальный порядок перемножения матриц следующий:

(((((20x15)x30)x53)x10)x20)x11)

При таком порядке перемножения матриц необходимое количество операций умножения будет минимальным и равным 27550, что является наилучшим ответом.

Таким образом, использование алгоритма динамического программирования более эффективно для решения задачи о перемножении матриц, чем рекурсивный метод.