МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«МИРЭА – Российский технологический университет»

РТУ МИРЭА

Отчет по выполнению практического задания №8.2

Тема: «ОТДЕЛЬНЫЕ ВОПРОСЫ АЛГОРИТМИЗАЦИИ»

Дисциплина: Структуры и алгоритмы обработки данных

Выполнил студент Кузнецов Л. А.
группа ИКБО-20-23

Москва 2024

СОДЕРЖАНИЕ

ЧАСТЬ 8.2

Задание

1. Разработать алгоритм решения задачи с применением метода, указанного в варианте и реализовать программу.

2. Оценить количество переборов при решении задачи стратегией «в лоб» - грубой силы. Сравнить с числом переборов при применении метода.

3. Оформить отчет в соответствии с требованиями документирования разработки ПО: Постановка задачи, Описание алгоритмов и подхода к решению, Код, результаты тестирования, Вывод.

Вариант

18 вариант - разработать процедуру оптимального способа расстановки скобок в произведении последовательности матриц, размеры которых равны (5,10,3,12,5,50,6), чтобы количество скалярных умножений стало минимальным (максимальным) (“Жадный алгоритм”).

Решение задачи

Для начала стоит ознакомиться с самим жадным алгоритмом: он представляет из себя выбор наилучшего исхода на данный момент без анализа последствий сделанного выбора.

Так как мы работаем с матрицами и их скалярным умножением, то итоговый результат кол-ва умножений можно получить перемножив ряды первой матрицы на столбцы второго, что и будет использоваться в дальнейшей работе как основной метод оценки кол-ва перемножений.

Математическая модель: проходимся необходимое кол-во раз по всем имеющимся матрицам и определяем выше описанным методом кол-во перемножений. После этого проставляем скобки там, где нашли наименьшее/наибольшее кол-во перемножений. Повторяем столько раз, сколько матриц используется в самой задаче.

У моего метода решения задачи фиксированное значение проходов, а именно (формула 1)

$$S = (n * 2 + 1) * n \quad (1)$$

Где n – кол-во матриц.

Стоит отметить, что S – кол-во проходов по всем элементам, а не кол-во повторений цикла, ведь кол-во повторений цикла тоже фиксированное, а именно n раз повторяется цикл.

Также можно аналитически определить кол-во повторений проверки всех возможных исходов «в лоб». В случае такого перебора нам придётся пройти приблизительно $\log_2 n * n^2$, что больше полученных значений перебора, используя “жадный алгоритм”.

Теперь же перейдём к описанию кода программы (рис. 1-4).

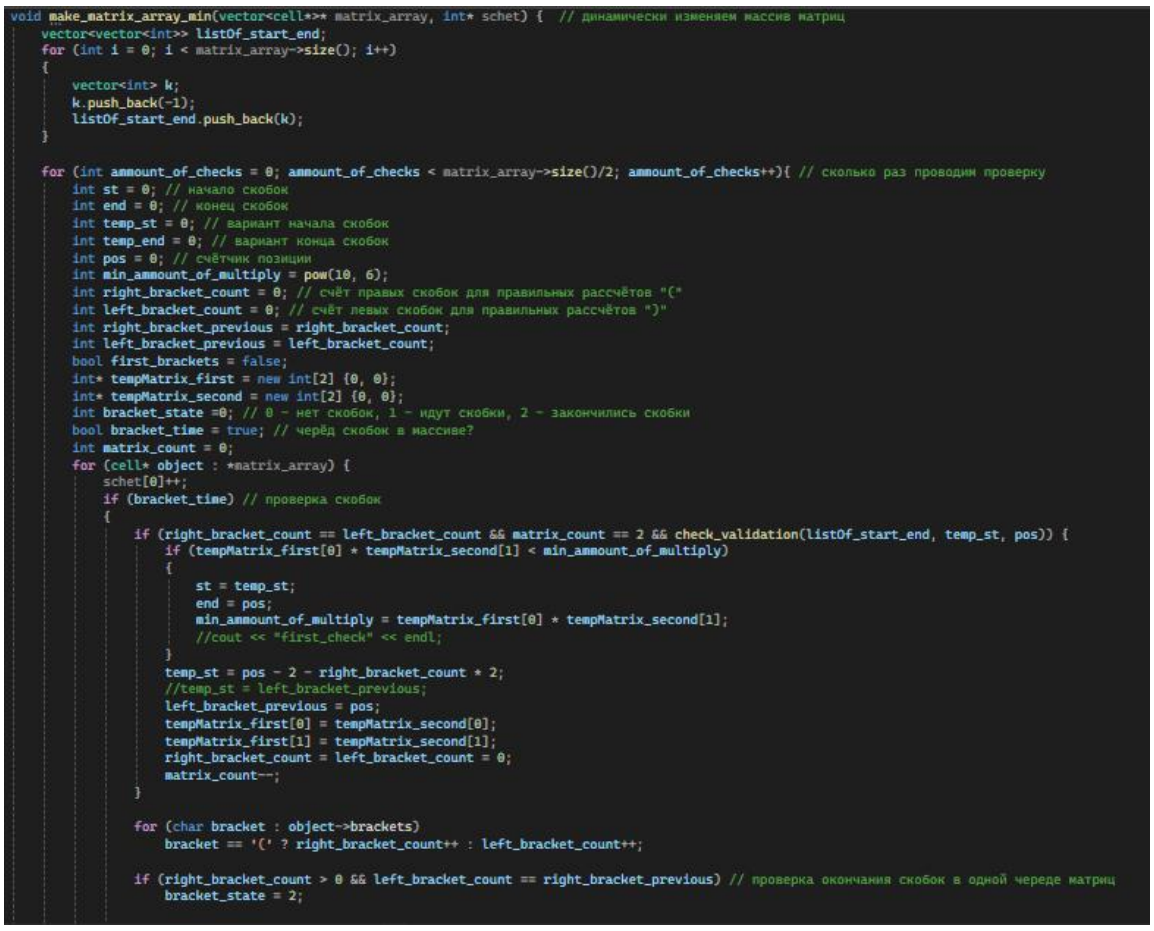
Начнём, пожалуй, с описания всех методов, необходимых для реализации алгоритма (рис. 1).

```
struct cell {
    int* matrix = nullptr;
    string brackets = "";
};

void out(vector<cell*>* matrix_array) {
    bool brackets_time = true;
    for (cell* object : *matrix_array) {
        if (brackets_time){
            cout << ' ' << object->brackets << ' ';
        }
        else{
            cout << object->matrix[0] << "x" << object->matrix[1];
        }
        brackets_time = !brackets_time;
    }
    cout << endl;
}


bool check_validation(vector<vector<int>> listOf_start_end, int st, int end)
{
    for (int end_value : listOf_start_end[st])
        if (end_value == end)
            return false;
    return true;
}
```

Рисунок 1 – Все необходимые для реализации алгоритма методы

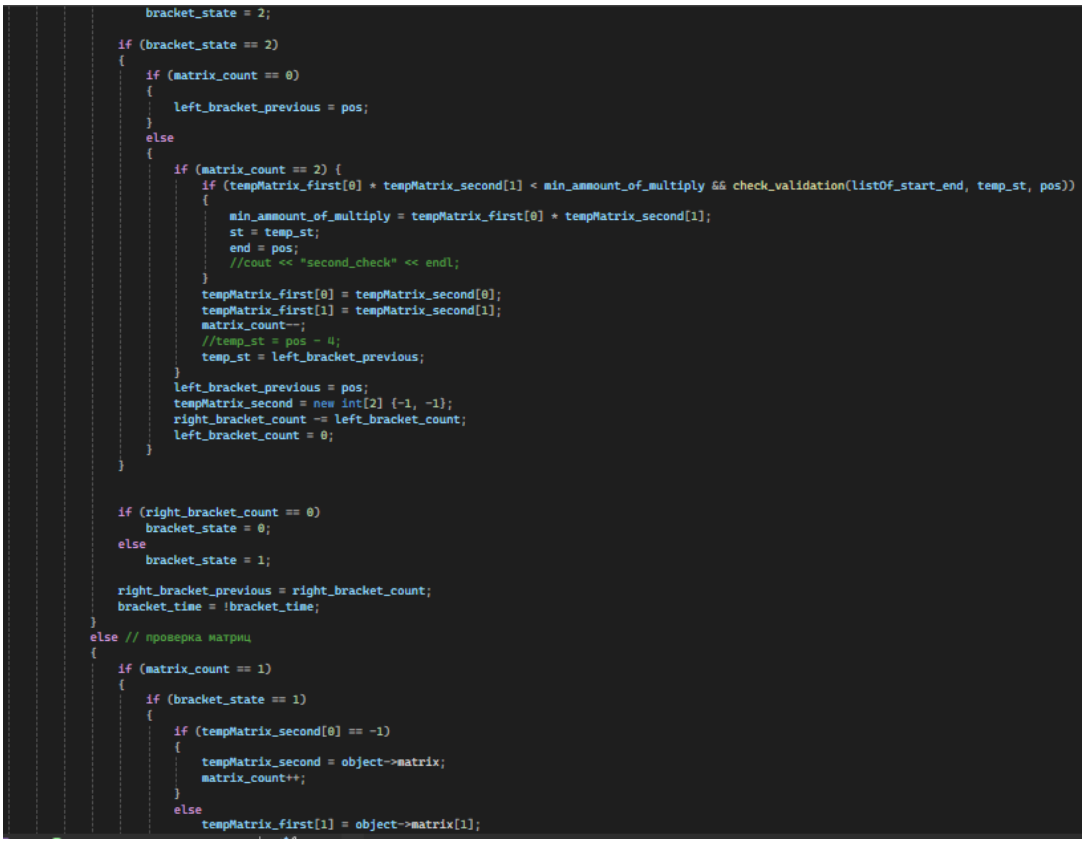
На рис.1 представлена структура cell, хранящая в себе все данные о наших взаимодействиях с матрицами (саму матрицу и наличие скобок). Ещё есть метод out, который реализует вывод на экран всех элементов нашего уравнения. А check_validation служит способом избежания повторной простановки скобок в одном и том же месте.

На рис.2-4 представлен сам код «Жадного алгоритма».



```
void make_matrix_array_min(vector<cell*>* matrix_array, int* schet) {  // динамически изменяем массив матриц
    vector<vector<int>> listOf_start_end;
    for (int i = 0; i < matrix_array->size(); i++)
    {
        vector<int> k;
        k.push_back(-1);
        listOf_start_end.push_back(k);
    }

    for (int ammount_of_checks = 0; ammount_of_checks < matrix_array->size()/2; ammount_of_checks++){ // сколько раз проводим проверку
        int st = 0; // начало скобок
        int end = 0; // конец скобок
        int temp_st = 0; // вариант начала скобок
        int temp_end = 0; // вариант конца скобок
        int pos = 0; // счётчик позиции
        int min_ammount_of_multiply = pow(10, 6);
        int right_bracket_count = 0; // счёт правых скобок для правильных рассчётов "("
        int left_bracket_count = 0; // счёт левых скобок для правильных рассчётов ")"
        int right_bracket_previous = right_bracket_count;
        int left_bracket_previous = left_bracket_count;
        bool first_brackets = false;
        int* tempMatrix_first = new int[2] {0, 0};
        int* tempMatrix_second = new int[2] {0, 0};
        int bracket_state =0; // 0 - нет скобок, 1 - идут скобки, 2 - закончились скобки
        bool bracket_time = true; // черёд скобок в массиве?
        int matrix_count = 0;
        for (cell* object : *matrix_array) {
            schet[0]++;
            if (bracket_time) // проверка скобок
            {
                if (right_bracket_count == left_bracket_count && matrix_count == 2 && check_validation(listOf_start_end, temp_st, pos)) {
                    if (tempMatrix_first[0] * tempMatrix_second[1] < min_ammount_of_multiply)
                    {
                        st = temp_st;
                        end = pos;
                        min_ammount_of_multiply = tempMatrix_first[0] * tempMatrix_second[1];
                        //cout << "first_check" << endl;
                    }
                    temp_st = pos - 2 - right_bracket_count * 2;
                    //temp_st = left_bracket_previous;
                    left_bracket_previous = pos;
                    tempMatrix_first[0] = tempMatrix_second[0];
                    tempMatrix_first[1] = tempMatrix_second[1];
                    right_bracket_count = left_bracket_count = 0;
                    matrix_count--;
                }

                for (char bracket : object->brackets)
                    bracket == '(' ? right_bracket_count++ : left_bracket_count++;

                if (right_bracket_count > 0 && left_bracket_count == right_bracket_previous) // проверка окончания скобок в одной череде матриц
                    bracket_state = 2;
```

Рисунок 2 – 1-ая часть реализации «Жадного алгоритма»



```
                bracket_state = 2;

            if (bracket_state == 2)
            {
                if (matrix_count == 0)
                {
                    left_bracket_previous = pos;
                }
                else
                {
                    if (matrix_count == 2) {
                        if (tempMatrix_first[0] * tempMatrix_second[1] < min_ammount_of_multiply && check_validation(listOf_start_end, temp_st, pos))
                        {
                            min_ammount_of_multiply = tempMatrix_first[0] * tempMatrix_second[1];
                            st = temp_st;
                            end = pos;
                            //cout << "second_check" << endl;
                        }
                        tempMatrix_first[0] = tempMatrix_second[0];
                        tempMatrix_first[1] = tempMatrix_second[1];
                        matrix_count--;
                        //temp_st = pos - 4;
                        temp_st = left_bracket_previous;
                    }
                    left_bracket_previous = pos;
                    tempMatrix_second = new int[2] {-1, -1};
                    right_bracket_count -= left_bracket_count;
                    left_bracket_count = 0;
                }
            }


            if (right_bracket_count == 0)
                bracket_state = 0;
            else
                bracket_state = 1;

            right_bracket_previous = right_bracket_count;
            bracket_time = !bracket_time;
        }
        else // проверка матриц
        {
            if (matrix_count == 1)
            {
                if (bracket_state == 1)
                {
                    if (tempMatrix_second[0] == -1)
                    {
                        tempMatrix_second = object->matrix;
                        matrix_count++;
                    }
                    else
                        tempMatrix_first[1] = object->matrix[1];
```

Рисунок 3 – 2-ая часть реализации «Жадного алгоритма»

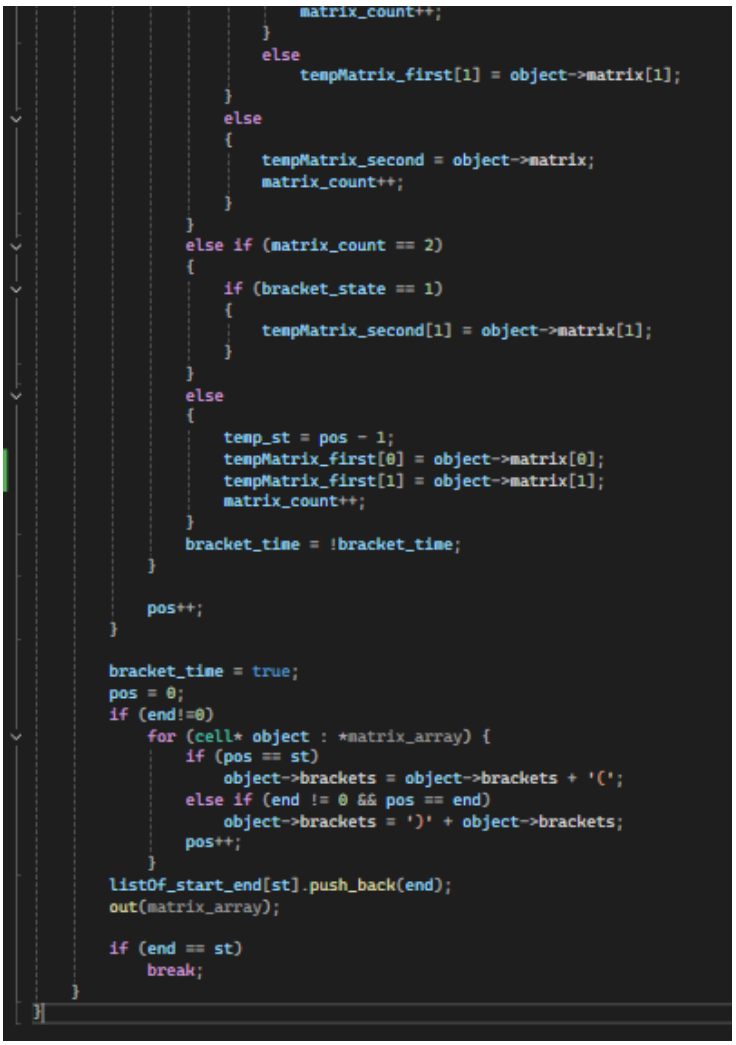


Рисунок 4 – 3-я часть реализации «Жадного алгоритма»

Алгоритм, представленный на рис.2-4 был описан ранее, однако это не отменяет того факта, что, несмотря на всю свою простоту трактовки, реализации представила из себя довольно трудоёмкий процесс, так пришлось обрабатывать множество ситуаций и дать программе такие указания, чтобы она была способна определить необходимый результат, опираясь исключительно на текущие значения. Это была самая большая и трудная часть работы.

Далее проиллюстрируем на рис.5 метод создания списка матриц с заданными ограничениями.

```
int create_matrix(int rows, vector<cell*>* matrix_array, int* allowed_numbers = new int[6] {5, 10, 3, 12, 50, 6}) {
    int tempColumns = allowed_numbers[rand() % 6];
    cell* tempCell = new cell;
    tempCell->matrix = new int[2] {rows, tempColumns};
    matrix_array->push_back(tempCell);
    matrix_array->push_back(new cell);
    return tempColumns;
}
```

Рисунок 5 – Метод создания списка матриц с заданными ограничениями

В данном метода мы банально создаём случайным образом следующую матрицу в зависимости от кол-ва столбцов предыдущей (переменная названа rows из-за того, что в текущей матрице они примет смысл как раз-таки рядов, а не столбцов).

И последняя иллюстрация в текущем разделе – рис.6 реализация алгоритма работы программы.

```
int main() {
    setlocale(LC_ALL, "ru");
    srand(time(nullptr));

    int* schet = new int[1] {0};

    vector<cell*>* matrix_array = new vector<cell*>;
    vector<cell> false_matrix_array;
    matrix_array->push_back(new cell);


    int choice;
    cout << "Ввод автоматом?" << endl
        << "1 - да |" << "2 - нет" << endl;
    cin >> choice;
    if (choice == 1)
    {
        int tempRows = create_matrix(5, matrix_array);
        for (int i = 0; i < 4; i++)
            tempRows = create_matrix(tempRows, matrix_array);
    }
    else {
        cell* tempCell;
        int tempNum = 0;
        bool allow_cycle = true;
        while (allow_cycle) {
            tempCell = new cell;
            tempCell->matrix = new int[2] {0, 0};
            cout << "Введите матрицу через пробел (000... 000...) (для прекращения введите -1)" << endl;
            for (int i=0; i<2; i++){
                cin >> tempNum;
                if (tempNum == -1){
                    allow_cycle = false;
                    break;
                }
                tempCell->matrix[i] = tempNum;
            }
            if (allow_cycle){
                matrix_array->push_back(tempCell);
                matrix_array->push_back(new cell);
            }
        }
    }


    for (cell* cell : *matrix_array)
        false_matrix_array.push_back(*cell);

    make_matrix_array_min(matrix_array, schet);

    cout << "Кол-во проходов по \"жадному\" алгоритму :" << schet[0] << endl;

    return 0;
```

Рисунок 6 – Реализация алгоритма работы программы

Алгоритм на рис.6 реализует простейший CLI и направляет пользователя на создание уравнения в соответствии с задачей.

Тесты

Теперь же приступим к тестам нашей программы (рис.7-10).

```
Ввод автоматом?
1 - да |2 - нет
2
Введите матрицу через пробел (000... 000...) (для прекращения введите -1)
5 12
Введите матрицу через пробел (000... 000...) (для прекращения введите -1)
12 6
Введите матрицу через пробел (000... 000...) (для прекращения введите -1)
6 5
Введите матрицу через пробел (000... 000...) (для прекращения введите -1)
5 3
Введите матрицу через пробел (000... 000...) (для прекращения введите -1)
3 10
Введите матрицу через пробел (000... 000...) (для прекращения введите -1)
-1
  5x12  12x6 ( 6x5  5x3 ) 3x10
 ( 5x12  12x6 )( 6x3  5x3 ) 3x10
 (( 5x12  12x6 )( 6x3  5x3 )) 3x10
 ((( 5x12  12x6 )( 6x3  5x3 )) 3x10 )
 ((( 5x12  12x6 )( 6x3  5x3 )) 3x10 )
Кол-во проходов по "жадному" алгоритму :55
```

Рисунок 7 – 1-ый тест работы программы

```
Ввод автоматом?
1 - да |2 - нет
2
Введите матрицу через пробел (000... 000...) (для прекращения введите -1)
6 3
Введите матрицу через пробел (000... 000...) (для прекращения введите -1)
3 5
Введите матрицу через пробел (000... 000...) (для прекращения введите -1)
5 10
Введите матрицу через пробел (000... 000...) (для прекращения введите -1)
10 6
Введите матрицу через пробел (000... 000...) (для прекращения введите -1)
6 6
Введите матрицу через пробел (000... 000...) (для прекращения введите -1)
6 12
Введите матрицу через пробел (000... 000...) (для прекращения введите -1)
-1
 ( 6x3  3x5 ) 5x10  10x6  6x6  6x12
 ( 6x3  3x5 )( 5x10  10x6 ) 6x6  6x12
 ( 6x3  3x5 )(( 5x6  10x6 ) 6x6 ) 6x12
 (( 6x3  3x5 )(( 5x6  10x6 ) 6x6 )) 6x12
 ((( 6x3  3x5 )(( 5x6  10x6 ) 6x6 )) 6x12 )
 ((( 6x3  3x5 )(( 5x6  10x6 ) 6x6 )) 6x12 )
Кол-во проходов по "жадному" алгоритму :78
```

Рисунок 8 – 2-ый тест работы программы

```
Ввод автоматом?
1 - да |2 - нет
2
Введите матрицу через пробел (000... 000...) (для прекращения введите -1)
5 5
Введите матрицу через пробел (000... 000...) (для прекращения введите -1)
5 3
Введите матрицу через пробел (000... 000...) (для прекращения введите -1)
3 3
Введите матрицу через пробел (000... 000...) (для прекращения введите -1)
5 3
Введите матрицу через пробел (000... 000...) (для прекращения введите -1)
3 10
Введите матрицу через пробел (000... 000...) (для прекращения введите -1)
10 6
Введите матрицу через пробел (000... 000...) (для прекращения введите -1)
6 6
Введите матрицу через пробел (000... 000...) (для прекращения введите -1)
-1
  5x5  5x3 ( 3x3  5x3 ) 3x10  10x6  6x6
 ( 5x5  5x3 )( 3x3  5x3 ) 3x10  10x6  6x6
 (( 5x5  5x3 )( 3x3  5x3 )) 3x10  10x6  6x6
 (( 5x5  5x3 )( 3x3  5x3 ))( 3x10  10x6 ) 6x6
 (( 5x5  5x3 )( 3x3  5x3 ))(( 3x6  10x6 ) 6x6 )
 ((( 5x5  5x3 )( 3x3  5x3 ))(( 3x6  10x6 ) 6x6 ))
 ((( 5x5  5x3 )( 3x3  5x3 ))(( 3x6  10x6 ) 6x6 ))
Кол-во проходов по "жадному" алгоритму :105
```

Рисунок 9 – 3-ый тест работы программы

```
  5x5  5x6  6x5  5x50  50x6  6x50  50x12  12x6 ( 6x5  5x3 )
  5x5 ( 5x6  6x5 ) 5x50  50x6  6x50  50x12  12x6 ( 6x3  5x3 )
 ( 5x5 ( 5x5  6x5 )) 5x50  50x6  6x50  50x12  12x6 ( 6x3  5x3 )
 ( 5x5 ( 5x5  6x5 ))( 5x50  50x6 ) 6x50  50x12  12x6 ( 6x3  5x3 )
 (( 5x5 ( 5x5  6x5 ))( 5x6  50x6 )) 6x50  50x12  12x6 ( 6x3  5x3 )
 (( 5x5 ( 5x5  6x5 ))( 5x6  50x6 )) 6x50  50x12 ( 12x6 ( 6x3  5x3 ))
 (( 5x5 ( 5x5  6x5 ))( 5x6  50x6 ))( 6x50  50x12 )( 12x3 ( 6x3  5x3 ))
 (( 5x5 ( 5x5  6x5 ))( 5x6  50x6 ))(( 6x12  50x12 )( 12x3 ( 6x3  5x3 )))
 ((( 5x5 ( 5x5  6x5 ))( 5x6  50x6 ))(( 6x3  50x12 )( 12x3 ( 6x3  5x3 ))))
 ((( 5x5 ( 5x5  6x5 ))( 5x6  50x6 ))(( 6x3  50x12 )( 12x3 ( 6x3  5x3 ))))
Кол-во проходов по "жадному" алгоритму :210
```

Рисунок 10 – 4-ый тест работы программы

Данные тесты были выбраны именно из-за того, что каждый из них образует неповторимую ситуацию, проверяющую на точность и стабильность нашу программу, ведь каждый раз, когда алгоритм работал во всех случаях, находился один, который выставлял условия, не рассматриваемые плохо реализованным алгоритмом, что нельзя сказать об итоговом результате.

Если же сравнивать с перебором то он получает следующие значения (рис. 11).

Рисунок 11 – Кол-во проходов при прямом переборе

ВЫВОД

В конечном итоге мы получили работающую программу, выполняющую все необходимые требования за фиксированное кол-во итерация. А также получили и развили навыки работы с различными алгоритмами.

СПИСОК ИНФОРМАЦИОННЫХ ИСТОЧНИКОВ

1. Страуструп Б. Программирование. Принципы и практика с использованием C++. 2-е изд., 2016.

2. Документация по языку C++ [Электронный ресурс]. URL: https://docs.microsoft.com/ruru/cpp/cpp/ (дата обращения 01.09.2021).

3. Курс: Структуры и алгоритмы обработки данных. Часть 2 [Электронный ресурс]. URL: https://online-edu.mirea.ru/course/view.php?id=4020 (дата обращения 01.09.2021).