МИНОБРНАУКИ РОССИИ

САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ

ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)

Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №1

по дисциплине «Алгоритмы и системы данных»

ТЕМА: РЕКУРСИЯ

Студент гр. 9381 _____ Любимов В.А.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург

2021

Цель работы.

Изучить понятие рекурсии и некоторые алгоритмы на базе рекурсии. Разработать программу для решения поставленной задачи с использованием рекурсивных алгоритмов.

Задание.

Вариант 27

Реализовать рекурсивную функцию вычисления детерминанта квадратной матрицы по формуле:

$\det A = \sum_{i=1}^{n} (-1)^{i+1} a_{i1} \overline{M}_j^1$, где $\overline{M}_j^1$ — дополнительный минор к элементу a_{i1} матрицы A.

Дополнительный минор элемента a_{ij} матрицы A n-го порядка есть детерминант порядка n−1, соответствующий той матрице, которая получается из матрицы A путём вычёркивания i-й строки и j-го столбца.

Описание алгоритма.

По заданию необходимо вычислить определить заданной квадратной матрицы. Будем считать, что её элементы целочисленные.

Входные данные представляют из себя строку, содержащую одно число – размер матрицы, а дальше идут строки матрицы, количество которых соответствует размеру матрицы. Количество чисел в каждой строке матрицы тоже соответствует указанному размеру матрицы.

Пользователь указывает файлы с входными данными и для записи результата и промежуточных данных.

Входные данные считываются построчно в форме строк при помощи функции std::getline(). Дальше полученные данные проверяются на корректность при помощи регулярных выражений. Если данные

некорректны программа информирует об этом пользователя и корректно завершается. Если данные корректны, то они преобразуются в целые числа при помощи функций strtok(), atoi() и atol(). В итоге получается двумерный массив.

Далее выполняется вычисление детерминанта. Для этого вызывается функция вычисления минора calcMinor(), получающая на вход матрицу, её размер и глубину рекурсии и работающая следующим образом:

1. Если получена матрица размера 1 на 1, то вернуть её единственный элемент.
2. Если получена матрица 2 на 2, то вычислить её определитель по известной не рекурсивной формуле: $a_{11}*a_{22} - a_{12}*a_{21}$.
3. Если матрица 3 на 3 или больше, то для каждого элемента первого столбца получать матрицу на порядок меньше, вычеркнув первый столбец и текущую строку из исходной матрицы. После этого запустить функцию calcMinor() от уменьшенной матрицы и увеличенной на 1 глубинной рекурсии.

В итоге получается искомый определитель, который записывается в выходной файл и выводится на экран.

Описание функций.

1. long int secondOrderDet(long int** matrix) – получает на вход двумерный массив – матрицу два на два и вычисляет её определитель по формуле: $a_{11}*a_{22} - a_{12}*a_{21}$.
2. long int** makeMinor(long int** matrix, int line, int size) – получает на вход двумерный массив, номер строки, которую надо убрать и размер массива. Динамически выделяет память под двумерный массив, в котором на строку и столбец меньше. Затем копирует

данные из исходного массива в новый, пропуская данные из «вырезанных» строки и столбца. Возвращает полученный двумерный массив.

3. void writeLog(int step, ofstream& fout, string message) – получает на вход глубину рекурсии, файл для вывода и выводимую строку. Выводит полученную строку в консоль и полученный файл, добавляя передней по табу за каждый уровень рекурсии.
4. long int calcMinor(long int** matrix, int size, int step, ofstream& fout) – получает двумерный массив – матрицу, размер матрицы, глубину рекурсии и ссылку на файл вывода. Для матриц размера меньше 3 на 3 вычисляет их определитель без использования рекурсии. Для всех остальных матриц находит дополнительный минор для каждого элемента нулевого столбца. Для этого находится матрица для вычисления текущего минора функцией makeMinor(). После этого вычисляется сам минор рекурсивным вызовом функции calcMinor() от полученной уменьшенной матрицы и увеличенной на единицу глубины рекурсии. После чего добавляем вычисленный минор к ответу исходя из формулы в задании. Наконец очищаем память выделенную в функции makeMinor() и переходим к следующей строке в полученной функцией матрице. Возвращаемым значением является определитель полученной функцией матрицы. Вычисления сопровождаются поясняющими выводами в консоль и файл выходных данных.
5. string makePattern(int size) – создаёт шаблон для регулярного выражения содержащий size десятичных целых чисел. Возвращает полученный шаблон.
6. bool checkData(string& data, const char* pattern) – получает ссылку на строку данных и шаблон. При помощи регулярных выражений

сравнивает шаблон с полученной строкой. Если строка удовлетворяет шаблону, то функция возвращает True, иначе возвращает False.

7. void stringToIntArray(string& data, long int* &line, int size) – получает ссылку на строку данных, ссылку на целочисленный одномерный массив – строка матрицы, длину строки. Входная строка содержит строку матрицы в виде символов. Сначала получаем из строки типа std::string массив символов. При помощи функций strtok() и atol() «нарезаем» строку символов по пробелам и полученные токены преобразуем к целым числам, а затем записываем в полученную функцией строку матрицы.
8. void clearSpace(long int** &matrix, int size) – удаляет выделенную под матрицу память.
9. int main() – запрашивает у пользователя имена файлов ввода-вывода данных и открывает их. Проверяем корректность открытия. В случае успеха считываем данные из файла с входными данными, проверяя их на корректность функцией checkData(). Если данные корректны, то создаём целочисленную матрицу при помощи функции stringToIntArray(). Затем вычисляем определитель полученной матрицы и записываем его в файл выходных данных. Очищаем выделенную память.

Тестирование.

Тестирование программы представлено в таблице 1.

Таблица 1.

№	Входные данные:	Выходные данные:
1	1 8	8

2	2 1 1 7 8	1
3	4 1 2 3 4 645 7234 293 8 1 3 6 4 465 8 6 564	-28779948
4	6 16 23 28 26 26 22 0 21 22 1 6 17 3 20 26 17 12 23 2 26 8 28 25 2 28 22 20 23 2 8 7 18 23 28 7 11	33205417
5	3 12 11 4 17 8 15 28 25 23 5 6 9 6 29 15 10 15 4 23 20 13 5 24 2 26 14 7 6 26 15 10 0 26 6 17 26 13 8 13 6 5 12 16 4 11 23 14 18 27 29 9 2 4 3 4 22 9 4 29 27 11 1 28	53614923708

Тестирование программы на некорректных данных представлено в таблице 2.

Таблица 2.

№	Входные данные:	Выходные данные:
1		Invalid input data!
2	4 1 2 3 4 645 7234 293 8 1 3 6 4 465 8 6 i	The size of matrix is 4x4. 1 2 3 4 645 7234 293 8 1 3 6 4 465 8 6 i Invalid input data!
3	6 16 23 28 26 26 22 0 21 22 1 6 17 3 20 26 17 12 23 2 26 8 28 25 2 28 22 20 23 2 8 7 18 23 28 7 11 9	The size of matrix is 6x6. 16 23 28 26 26 22 0 21 22 1 6 17 3 20 26 17 12 23 2 26 8 28 25 2 28 22 20 23 2 8

		7 18 23 28 7 11 9 Invalid input data!

Файлы с этими входными данными лежат в папке test.

Выводы.

Было изучено понятие рекурсии и некоторые рекурсивные алгоритмы. На языке программирования C++была реализована программа, рекурсивно вычисляющая определитель заданной матрицы.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файлов: main.cpp

```
#include <regex.h>
#include <string.h>
#include <iostream>
#include <string>
#include <fstream>
#include <cstdlib>
#include <cmath>

using namespace std;

long int secondOrderDet(long int** matrix){//вычисление определителя матрицы второго порядка
    return matrix[0][0]*matrix[1][1] - matrix[0][1]*matrix[1][0];
}

long int** makeMinor(long int** matrix, int line, int size){//создание матрицы меньшего порядка для вычисления минора
    long int** minor = new long int*[size-1];
    for(int i = 0; i < size-1; i++){
        minor[i] = new long int[size-1];
    }

    int i = 0;
    int k = 0;
    while(k < size){
        if(k != line){
            for(int j = 0; j < size-1; j++){
                minor[i][j] = matrix[k][j+1];
            }
            k++;
            i++;
        }
        else{
            k++;
        }
    }
    return minor;
}

void writeLog(int step, ofstream& fout, string message){//логирование промежуточных и итоговых данных
    for(int i = 0; i < step; i++){
```

```
            fout << "\t";
            cout << "\t";
        }
        fout << message;
        cout << message;
}

long int calcMinor(long int** matrix, int size, int step, ofstream&
fout){//рекурсивная функция вычисления определителя
        long int res = 0;
        if(size == 1){//матрица 1 на 1
            writeLog(step, fout, "It's is the simplest matrix content only one
element.\n");
            return matrix[0][0];
        }

        if(size == 2){//матрица 2 на 2
            int value = secondOrderDet(matrix);
            writeLog(step, fout, "The 2x2 matrix have been found. Determinant can
be found without recursion. Its value is " + to_string(value) + ".\n");
            return value;
        }

        for(int i = 0; i < size; i++){//матрица 3 на 3 и больше
            long int** minor = makeMinor(matrix, i, size);
            writeLog(step, fout, "To find this " + to_string(size) + "x" +
to_string(size) + " matrix determinant the additional minor of elemet " +
to_string(i+1) + ";1 should be found.\n");
            long int minor_value = calcMinor(minor, size-1, step + 1, fout);
            res += (pow(-1, i+2) * matrix[i][0] * minor_value);//получение
итогвого определителя
            writeLog(step, fout, "The value of this minor is " +
to_string(minor_value) + ". Current value of determinant of full matrix for
this minor is " + to_string(res) + ".\n");
            for(int j = 0; j < size-1; j++){
                delete[] minor[j];
            }
            delete[] minor;
        }

        return res;
}

string makePattern(int size){//создание шаблона для использования регулярок
        string pattern = "^";
        for(int i = 0; i < size-1; i++){
            pattern += "[0-9]+\\s";
        }
```

```
    pattern += "[0-9]+$";
    return pattern;
}

bool checkData(string& data, const char* pattern){//проверка входных данных
регулярками
    regex_t rexp;
    regmatch_t pm;
    regcomp(&rexp, pattern, REG_EXTENDED);
    if(!regexec(&rexp, data.data(), 0, &pm, 0)){
        regfree(&rexp);
        return true;
    }

    regfree(&rexp);
    return false;
}

void stringToIntArray(string& data, long int* &line, int
size){//преобразование строки входных данных к строке целых чисел
    char* c_st = new char[data.size() + 1];
    strcpy(c_st, data.data());
    int i = 0;
    char* pch = strtok(c_st, " ");
    line[i] = atol(pch);
    i += 1;
    while(i < size){
        pch = strtok(NULL, " ");
        line[i] = atol(pch);
        i += 1;
    }
    delete[] c_st;
}

void clearSpace(long int** &matrix, int size){//очистка выделенной под матрицу
памяти
    for(int j = 0; j < size-1; j++){
            delete[] matrix[j];
        }
        delete[] matrix;
}

int main(){
    //открытие файлов ввода-вывода и проверка этого
    cout << "Input the path to data file:\n";
    string fname;
    cin >> fname;
    ifstream fin(fname);
```

```
    if(!fin.is_open()){
        cout << "Opening file with test data failed!\n";
        return 0;
    }
    cout << "Input the path to result file:\n";
    cin >> fname;
    ofstream fout(fname);
    if(!fout.is_open()){
        cout << "Opening file for writing result data failed!\n";
        return 0;
    }

    //чтение и проверка входных данных
    int size;
    string data;
    getline(fin, data);
    if(!checkData(data, "^[0-9]+$")){
        writeLog(0, fout, "Invalid input data!\n");
        fin.close();
        fout.close();
        return 0;
    }
    size = atoi(data.data());
    writeLog(0, fout, "The size of matrix is " + to_string(size) + "x" +
to_string(size) + ".\n");

    long int** matrix = new long int*[size];
    for(int i = 0; i < size; i++){
        matrix[i] = new long int[size];
    }

    string pattern = makePattern(size);
    for(int i = 0; i < size; i++){
        getline(fin, data);
        writeLog(0, fout, data + "\n");
        if(!checkData(data, pattern.data())){
            writeLog(0, fout, "Invalid input data!\n");
            clearSpace(matrix, size);
            fin.close();
            fout.close();
            return 0;
        }
        stringToIntArray(data, matrix[i], size);
    }

    //вычисление определителя
    long int det = calcMinor(matrix, size, 0, fout);
```

```
    writeLog(0, fout, "The target determinant value is " + to_string(det) +
".\n");

    //очистка памяти и закрытие файлов
    clearSpace(matrix, size);

    fin.close();
    fout.close();

    return 0;
}
```