Изображение выглядит как цепь

Автоматически созданное описание

МИНОБРНАУКИ РОССИИ

федеральное государственное бюджетное образовательное учреждение высшего образования

«Санкт-Петербургский государственный технологический институт (технический университет)»

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| УГНС | | | 09.00.00 | Информатика и вычислительная техника | | |
| Направление подготовки | | | 09.03.01 | Информатика и вычислительная техника | | |
| Направленность (профиль) | | |  | Автоматизированные системы обработки информации и управления | | |
| Форма обучения | | |  | очная | | |
|  | | |  |  | | |
| Факультет | | |  | Информационных технологий и управления | | |
| Кафедра | | |  | Систем автоматизированного проектирования и управления | | |
| Учебная дисциплина | |  | Программирование | | |
| Курс | I | | | Группа | 403 |

Отчёт по лабораторной работе № 1

Вариант № 1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Исполнитель: |  |  |  |  |
| обучающийся группы 403 |  |  |  | Старков Силантий Денисович |
|  |  | (дата, подпись) |  |  |
|  |  |  |  |  |
| Проверили: |  |  |  | Корниенко Иван Григорьевич |
|  |  | (дата, подпись) |  | Федин Алексей Константинович |

Оглавление

[1. Постановка задачи 3](#_Toc65781932)

[2. Исходный данные 3](#_Toc65781933)

[3. Особые ситуации 3](#_Toc65781934)

[4. Математические методы и алгоритмы решения задач 3](#_Toc65781935)

[5. Форматы представления данных 4](#_Toc65781936)

[6. Структура программы 6](#_Toc65781937)

[7. Блок-схемы алгоритмов программы 7](#_Toc65781938)

[8. Описание хода выполнения лабораторной работы 8](#_Toc65781939)

[9. Результаты работы программы 8](#_Toc65781940)

[10. Исходный текст программы 10](#_Toc65781941)

[11. Документирование и комментирование исходного текста 16](#_Toc65781942)

## Постановка задачи

Необходимо реализовать структуру данных «Двоичное дерево поиска», продемонстрировать ее характерные особенности, реализовать возможность добавления и удаления элементов, визуализировать дерево. В программе должны быть предусмотрены три варианта заполнения: пользователем с клавиатуры, из файла и случайными числами.

## Исходный данные

В качестве исходных данных программа использует вводимую пользователем последовательность чисел, в программе она парсится и каждое числовое значение, содержащееся в ней, становится частью объекта класса Node и представляет значение узла в дереве. В этом классе также хранится информация о ближайших большем и меньшем узле, а также информация о родителе узла, если таковые имеются.

## Особые ситуации

Необходимо рассмотреть следующие особые ситуации.

* Если пользователь ввёл значение, не соответствующее операбельному типу, вычислений не происходит, программа просит ввести корректное значение, либо выполняет преобразование к нужному типу.
* Если пользователь ввёл несколько одинаковых значений, в дерево попадает только первая итерация данного числа.
* Если при файловом вводе или при удалении или добавлении элемента возникает конфликт построения дерева, программа выводит сообщение об ошибке и переносит пользователя в главное меню.

## Математические методы и алгоритмы решения задач

Согласно постановке задачи для составления программы будет использовано двоичное дерево поиска.

У каждого узла будут члены, содержащие ссылки на правый элемент, левый элемент и элемент-родителя при их наличии.

Добавление, поиск и удаление узлов будут осуществляться с помощью рекурсии и проверки на соответствие числовых значений узлов.

:

## Форматы представления данных

Программа использует следующие переменные:

|  |  |  |
| --- | --- | --- |
| Имя | Тип | Описание |
| Right | Node | Член-данное класса Node, хранящий правый узел от данного |
| Left | Node | Член-данное класса Node, хранящий левый узел от данного |
| Parent | Node | Член-данное класса Node, хранящий ссылку на родителя этого узла |
| Data | int | Член-данное класса Node, хранящий собственное значение |
| tree | BinaryTree | Объект, представляющий методы по добавлению, поиску и удалению элемента классу Node |
| root | Node | Объект – член класса BinaryTree, являющийся корнем всего дерева. |
| pictureBox | PictureBox | Контейнер, необходимый для визуализации дерева |
| NodeDraw | NodeForDraw | Объект, представляющий собой узел с собственным цветом, положением |
| nodeListLines | List  <List  <NodeForDraw  >> | Список линий, соединяющих узлы для отрисовки дерева |

Таблица 1 – Переменные, используемые в программе

Для хранения матрицы и ее характеристик используется класс cMatrix, в котором можно хранить размер матрицы и количество перестановок в ней.

Для реализации разных методов сортировки используется виртуальный класс ISort, от которого наследуются классы, в которых определены методы сортировки

Также в программе используется перечисляемый тип inputChoise, который хранит в себе цифры выборы, которые может сделать пользователь при вводе, и перечисляемый тип binarChoise, который хранит цифры, свидетельствующие о том, выбрал ли пользователь сохранение в файл.

## Структура программы

Программа разделена на следующие модули:

1. main: начальный модуль с функцией main. В нем изменяются параметры языка в консоли и запускаются функции интерфейса.
2. interface: в нем находятся функции sayHi и menu. С их помощью осуществляется основная навигация.
3. keyboardInput, randomInput, fileInput: модули, в которых определены методы для создания объектов класса в соответствии с выбором пользователя.
4. sort: модуль, осуществляющий сортировку матрицы.
5. test: модуль, хранящий тесты основной рассчетных функции.
6. save: модуль, хранящий функции, необходимые для записи в файл.
7. get: модуль, хранящий функции для пользовательского ввода.

Программа разделена на несколько функций:

|  |  |
| --- | --- |
| Имя | Описание |
| sort | Функция необходимая для первичного отбора элементов и последующей сортировки |
| sortArr | Виртуальная функция, переопределенная в каждом из классов-наследников сортировки ISort |
| gettValue | Функция, вычленяющая числовые параметры окружностей при файловом вводе. |
| testSort | Функция, содержащие модульные тесты функции testSort |
| menu | Меню, где происходит выбор между методами ввода |
| askRewrite | Функция запрашивающая разрешение на перезапись файла |
| check | Функция, проверяющая существование файла |
| saveData | Функция, осуществляющая сохранение исходных данных в файл |
| saveResult | Функция, осуществляющая сохранение результата выполнения в файл |
| menu | Функция, осуществляющая навигацию по программе |
| sayHi | Функция, выводящая приветствие при запуске программы |

## Таблица 3 – Функции, составляющие программу

## Блок-схемы алгоритмов программы

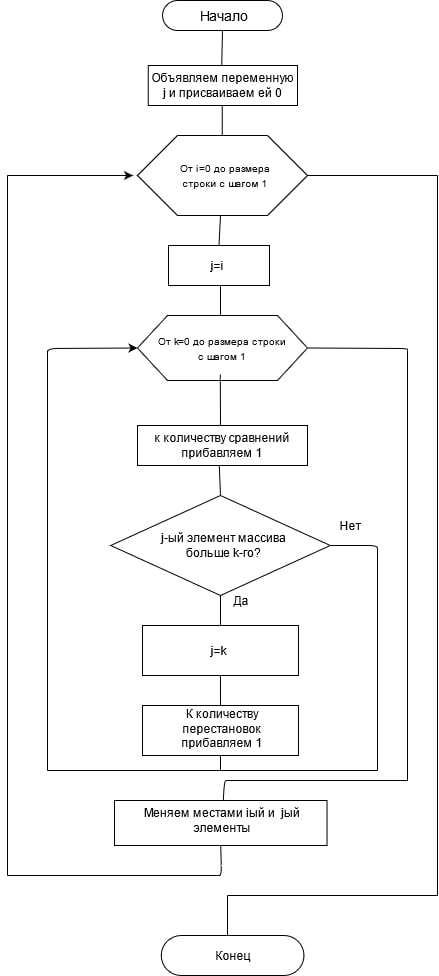


Рисунок 1 – Блок схема алгоритма сортировки пузырьком

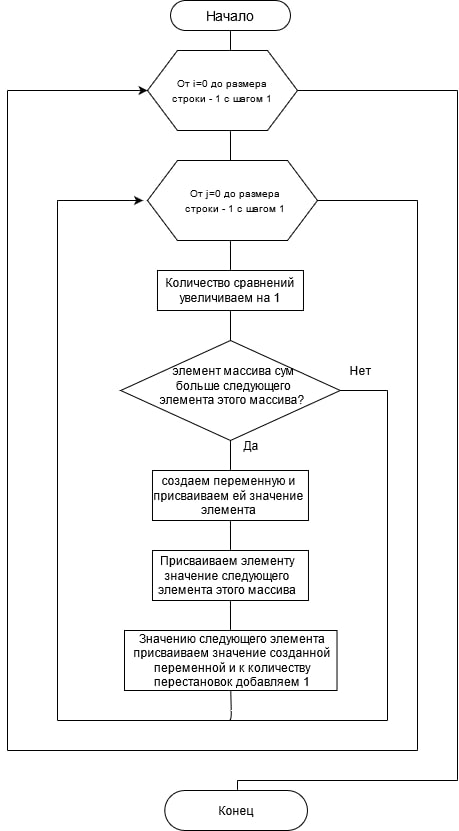


Рисунок 2 – Блок схема алгоритма сортировки выбором

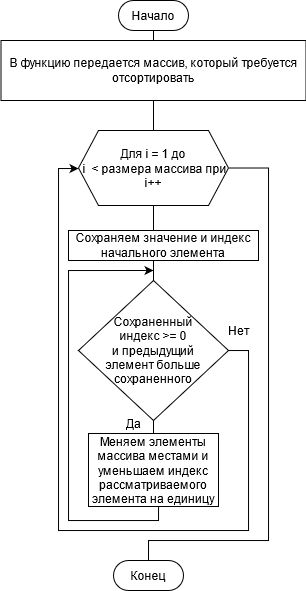


Рисунок 3 – Блок схема алгоритма сортировки вставки

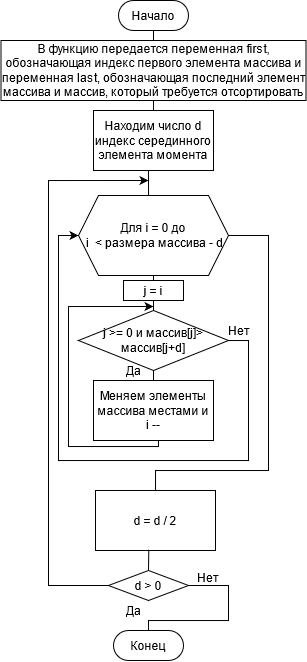


Рисунок 4 – Блок схема алгоритма сортировки Шелла

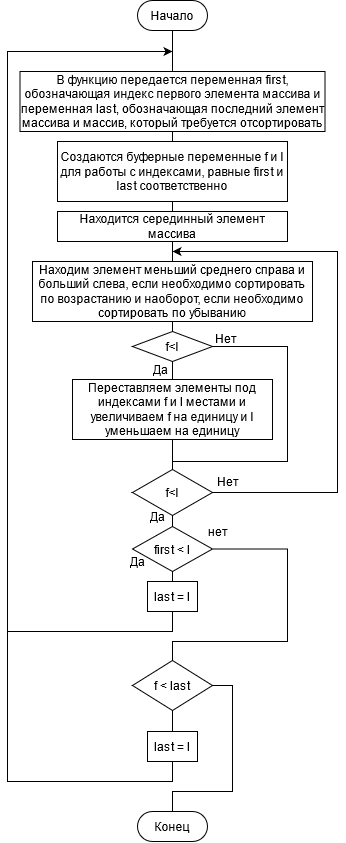


Рисунок 5 – Блок схема алгоритма быстрой сортировки

## Описание хода выполнения лабораторной работы

1. В ходе лабораторной работы было создано решение (Solution) в интегрированной среде разработки Microsoft Visual Studio C++ 2017. В нём был создан проект.
2. После набора текста программы выяснилось, что вывод текста на экран консольного приложения работает неправильно из-за различия кодировок консольного приложения и среды разработки. Для решения этой проблемы были использованы функции SetConsoleCP(1251) и

SetConsoleOutputCP(1251), которые обеспечивает работу приложения с символами кириллицы.

1. Была добавлена проверка ввода регулярных выражений с помощью функции is\_regular\_file().

## Результаты работы программы

В результате работы программы, на экран выводится исходная матрица и ее отсортированные разными алгоритмами версии. В конце выводится сравнительная таблица методов сортировки.

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 6 – Результат работы программы

## Исходный текст программы

//main.cpp

#include <Windows.h>

#include "interface.h"

int main()

{

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

system("color 0F");

sayHi();

menu();

}

// interface.h

#pragma once

#include <iostream>

#include "enums.h"

#include "cMatrix.h"

#include "cInput.h"

#include "ISort.h"

#include "get.h"

#include "save.h"

#include "test.h"

void sayHi();

void menu();

//interface.cpp

#include "interface.h"

#include <memory>

void sayHi() {

std::cout << " Студент СПБГТИ(ТУ)" << std::endl <<

" Старков Силантий Денисович" << std::endl <<

" 403 группа" << std::endl <<

" Контрольная работа №2" << std::endl <<

" Вариант №24" << std::endl <<

" Представить шахматную доску. Упорядочить белые клетки по возрастанию по"

<< std::endl << " строкам, черные фигуры по убыванию по столбцам " << std::endl << std::endl;

}

void menu() {

int makeChoise = 0, testFailErrors = 0;

if (testFindMedian()) {

do {

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

"Введите '1', чтобы сгенерировать случайные значения" << std::endl <<

"Введите '2', чтобы ввести числа с клавиатуры" << std::endl <<

"Введите '3', чтобы взять значения из файла" << std::endl <<

"Введите '0', чтобы завершить работу" << std::endl;

makeChoise = getInt();

if (makeChoise == randomInput || makeChoise == keyboardInput || makeChoise == fileInput) {

auto method = getTypeArr(makeChoise);

cMatrix matrix;

method->createMatrix(matrix);

if (matrix.getSizeColumn() != 0) {

print(matrix);

for (int i = bubbleS; i <= quickS; i++) {

auto sortType = getTypeSort(i);

std::cout<<ISort::printMethod[i];

cMatrix matrixSort(matrix);

sortType->sort(matrixSort);

print(matrixSort);

}

printTable();

if (makeChoise != fileInput)

saveData(matrix);

saveResult(matrix);

}

}

} while (makeChoise != ext);

}

}

//cInput.h

#pragma once

#include "cMatrix.h"

#include "get.h"

#include "enums.h"

#include <filesystem>

#include <fstream>

#include <string>

#include <memory>

class IInput {

public:

virtual ~IInput() = default;

virtual void createMatrix(cMatrix&) = 0;

};

std::unique\_ptr<IInput> getTypeArr(int);

class RandomInput final : public IInput {

public:

RandomInput();

~RandomInput();

void setMatrixSize(cMatrix&);

void createMatrix(cMatrix&) override;

};

class KeyboardInput final : public IInput {

public:

KeyboardInput();

~KeyboardInput();

void setMatrixSize(cMatrix&);

void createMatrix(cMatrix&) override;

};

class FileInput final : public IInput {

public:

FileInput();

~FileInput();

void createMatrix(cMatrix&) override;

};

//cInput.cpp

#include"cInput.h"

std::unique\_ptr<IInput> getTypeArr(int inputChoise) {

if (inputChoise == randomInput) {

return std::make\_unique<RandomInput>();

}

if (inputChoise == keyboardInput) {

return std::make\_unique<KeyboardInput>();

}

if (inputChoise == fileInput) {

return std::make\_unique<FileInput>();

}

}

//randomInput.cpp

#include "cInput.h"

void RandomInput::setMatrixSize(cMatrix& matrix) {

std::cout << "Введите количество строк: ";

int arrRSize = getInt();

while (arrRSize<=0) {

std::cout << "Введите количество строк больше 0 : ";

arrRSize = getInt();

}

std::cout << "Введите количество элементов строки: ";

int arrCSize = getInt();

while (arrCSize <= 0) {

std::cout << "Введите количество элементов строки больше 0 : ";

arrCSize = getInt();

}

matrix.setSize(arrRSize,arrCSize );

}

void RandomInput::createMatrix(cMatrix&matrix) {

setMatrixSize(matrix);

srand(static\_cast<unsigned int>(time(0)));

for (int i = 0; i < matrix.getSizeRow(); i++) {

for (int j = 0; j < matrix.getSizeColumn(); j++) {

matrix.setArr(i, j, random());

}

}

std::cout << std::endl

<< "--------------------------------" << std::endl

<< " Исходная матрица:" << std::endl

<< "--------------------------------" << std::endl;

}

//keyboardInput.cpp

#include "cInput.h"

void KeyboardInput::setMatrixSize(cMatrix& matrix) {

std::cout << "Введите количество строк: ";

int arrCSize = getInt();

while (arrCSize <= 0) {

std::cout << "Введите количество строк больше 0 : ";

arrCSize = getInt();

}

std::cout << "Введите количество элементов строки: ";

int arrRSize = getInt();

while (arrRSize <= 0) {

std::cout << "Введите количество элементов строки больше 0 : ";

arrRSize = getInt();

}

matrix.setSize(arrCSize, arrRSize);

}

void KeyboardInput::createMatrix(cMatrix& matrix) {

setMatrixSize(matrix);

for (int i = 0; i < matrix.getSizeRow(); i++) {

for (int j = 0; j < matrix.getSizeColumn(); j++) {

std::cout << "matrix[" << i+1 << "][" << j+1<<"] = ";

matrix.setArr(i, j, getDouble());

}

}

std::cout << std::endl

<< "--------------------------------" << std::endl

<< " Исходная матрица:" << std::endl

<< "--------------------------------" << std::endl;

}

//fileInput.cpp

#include "cInput.h"

#include <vector>

bool checkDir(std::string dir) {

try {

static\_cast<void>(std::filesystem::is\_regular\_file(dir));

}

catch (...) {

std::cout << "Чтение невозможно! Использовано зарезервированное имя." << std::endl;

return true;

}

return false;

}

void getValue(std::vector<double>& mtr, std::string arrayLine) {

int sizeArrayLine = arrayLine.length();

int i = 0;

std::string numStr = "";

while (arrayLine[i] != '|' && (i + 1) < sizeArrayLine) i++;

i++;

while ((i + 1) < sizeArrayLine && arrayLine[i] != '|') {

while (arrayLine[i] != ' ' && arrayLine[i] != '|') {

if (arrayLine[i] == '.' || arrayLine[i] == ',') {

numStr += '.';

}

else if ('0'<=arrayLine[i] && arrayLine[i]<='9' || arrayLine[i] == '-') {

numStr += arrayLine[i];

}

i++;

}

if (numStr!="") {

mtr.push\_back(stod(numStr));

}

i++;

numStr = "";

}

std::cout << std::endl;

}

void FileInput::createMatrix(cMatrix& matrix) {

bool err = false;

std::string dir;

std::cout << "Введите путь тестового файла: ";

std::cin >> dir;

while (checkDir(dir)) {

std::cout << "Введите путь тестового файла: ";

std::cin >> dir;

}

std::ifstream inputTest;

inputTest.open(dir);

while (!inputTest.is\_open()) {

std::cout << "Не удалось считать значения" << std::endl;

std::cout << "Введите путь тестового файла: ";

std::cin >> dir;

inputTest.close();

inputTest.open(dir);

}

std::string stringBuff;

getline(inputTest, stringBuff, '\n');

int i = 0;

std::vector<std::vector<double>> mtr;

while (stringBuff.size() > 1 && stringBuff != "") {

mtr.resize(i + 1);

getValue(mtr[i], stringBuff);

if (i > 0 && mtr[i - 1].size() != mtr[i].size()) {

std::cout << "Некорректная длинна строки матрицы!" << std::endl;

err = true;

break;

}

getline(inputTest, stringBuff, '\n');

if (mtr[i].size() > 0)

i++;

}

if (err == false && mtr.size() != 0) {

matrix.setSize(mtr.size(), mtr[0].size());

for (int j = 0; j < matrix.getSizeRow(); j++) {

for (int k = 0; k < matrix.getSizeColumn(); k++) {

matrix.setArr(j, k, mtr[j][k]);

}

}

std::cout << std::endl

<< "--------------------------------" << std::endl

<< " Исходная матрица:" << std::endl

<< "--------------------------------" << std::endl;

}

else

matrix.setSize(0, 0);

inputTest.close();

}

//cMatric.h

#pragma once

#include <windows.h>

#include <iostream>

#include "enums.h"

class cMatrix{

public:

cMatrix();

cMatrix(const cMatrix&);

~cMatrix();

int getSizeRow();

int getSizeColumn();

void setSize(int, int);

double getArr(int, int);

double\* getArrPtr(int, int);

void setArr(int, int, double);

double& operator[](int);

static int\* comparisons;

static int\* permutations;

double\*\* arr;

private:

int rowSize;

int columnSize;

};

void print(cMatrix&);

void printTable();

//cMatrix.cpp

#include "cMatrix.h"

int\* cMatrix::comparisons = new int[quickS + 1]{ 0 };

int\* cMatrix::permutations = new int[quickS + 1]{ 0 };

cMatrix::cMatrix() {

rowSize = columnSize = 0;

arr = nullptr;

}

cMatrix::cMatrix(const cMatrix& cpo):rowSize(cpo.rowSize), columnSize(cpo.columnSize)

{

arr = new double\* [cpo.rowSize];

for (int i = 0; i < cpo.rowSize; i++) {

arr[i] = new double[cpo.columnSize];

for (int j = 0; j < columnSize; j++) {

this->arr[i][j] = cpo.arr[i][j];

}

}

}

cMatrix::~cMatrix()

{

for (int i = 0; i < this->getSizeRow(); i++) {

delete[] arr[i];

}

delete[] arr;

arr = nullptr;

}

double& cMatrix::operator[](int i) {

return this->arr[i][0];

}

int cMatrix::getSizeRow() {

return this->rowSize;

}

int cMatrix::getSizeColumn() {

return this->columnSize;

}

void cMatrix::setSize(int vSize, int hSize) {

for (int i = 0; i < this->getSizeRow(); i++) {

delete[] arr[i];

}

delete[] arr;

arr = nullptr;

this->rowSize = vSize;

this->columnSize = hSize;

this->arr = new double\* [vSize];

for (int i = 0; i < vSize; i++) {

arr[i] = new double[hSize];

for (int j = 0; j < columnSize; j++) {

this->arr[i][j] = 0;

}

}

}

double cMatrix::getArr(int i, int j) {

return this->arr[i][j];

}

double\* cMatrix::getArrPtr(int i, int j) {

return &this->arr[i][j];

}

void cMatrix::setArr(int i, int j, double newSlot) {

this->arr[i][j] = newSlot;

}

void print(cMatrix& arrForPrint) {

HANDLE h = GetStdHandle(STD\_OUTPUT\_HANDLE);

std::cout << std::endl;

for (int i = 0; i < arrForPrint.getSizeRow(); i++) {

for (int j = 0; j < arrForPrint.getSizeColumn(); j++) {

if (!(j % 2 ^ i % 2))

SetConsoleTextAttribute(h, (WORD)((15 << 4) | 0));

else

SetConsoleTextAttribute(h, (WORD)(0 | 15));

std::cout << "\t" << arrForPrint.getArr(i, j);

}

SetConsoleTextAttribute(h, (WORD)(0 | 15));

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

}

std::cout << std::endl;

SetConsoleTextAttribute(h, (WORD)((0 << 1) | 2));

SetConsoleTextAttribute(h, (WORD)(0 | 15));

}

void printTable() {

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

<< "| Метод\t| Сравнений\t| Перестановок\t|" << std::endl

<< "|---------------|---------------|---------------|" << std::endl

<< "|Bubble sort\t|\t" << cMatrix::comparisons[bubbleS] << "\t|\t" << cMatrix::permutations[bubbleS] << "\t|\t" << std::endl

<< "|Selection sort\t|\t" << cMatrix::comparisons[selectionS] << "\t|\t" << cMatrix::permutations[selectionS] << "\t|\t" << std::endl

<< "|Insert sort\t|\t" << cMatrix::comparisons[insertS] << "\t|\t" << cMatrix::permutations[insertS] << "\t|\t" << std::endl

<< "|Shell sort\t|\t" << cMatrix::comparisons[shellS] << "\t|\t" << cMatrix::permutations[shellS] << "\t|\t" << std::endl

<< "|Quick sort\t|\t" << cMatrix::comparisons[quickS] << "\t|\t" << cMatrix::permutations[quickS] << "\t|\t" << std::endl;

}

//ISort.h

#pragma once

#include "cMatrix.h"

#include "enums.h"

#include <string>

#include <vector>

class ISort {

public:

ISort();

virtual ~ISort() = default;

void sort(cMatrix& copyMatrix);

static std::string printMethod[quickS+1];

protected:

virtual void sortArr(std::vector<double\*>&, bool) = 0;

};

std::unique\_ptr <ISort> getTypeSort(int);

class bubbleSort final : public ISort {

public:

bubbleSort();

~bubbleSort()=default;

void sortArr(std::vector<double\*>&, bool) override;

};

class selectionSort final : public ISort {

public:

selectionSort();

~selectionSort() = default;

void sortArr(std::vector<double\*>&, bool) override;

};

class insertionSort final : public ISort {

public:

insertionSort();

~insertionSort() = default;

void sortArr(std::vector<double\*>&, bool) override;

};

class shellSort final : public ISort {

public:

shellSort();

~shellSort() = default;

void sortArr(std::vector<double\*>&, bool) override;

};

class quickSort final : public ISort {

public:

quickSort();

~quickSort() = default;

void sortArr(std::vector<double\*>&, bool) ;

};

#include "ISort.h"

std::string ISort::printMethod[quickS + 1] = {

"\n\n--------------------------------\n Метод сортировки пузырьком:\n--------------------------------\n",

"\n\n--------------------------------\n Метод сортировки выбором:\n--------------------------------\n",

"\n\n--------------------------------\n Метод сортировки вставкой:\n--------------------------------\n",

"\n\n--------------------------------\n Метод сортировки Шелла:\n--------------------------------\n",

"\n\n--------------------------------\n Метод быстрой сортировки:\n--------------------------------\n"

};

std::unique\_ptr <ISort> getTypeSort(int inputChoise) {

if (inputChoise == bubbleS) {

return std::make\_unique<bubbleSort>();

}

else if (inputChoise == selectionS) {

return std::make\_unique<selectionSort>();

}

else if (inputChoise == insertS) {

return std::make\_unique<insertionSort>();

}

else if (inputChoise == shellS) {

return std::make\_unique<shellSort>();

}

else if (inputChoise == quickS) {

return std::make\_unique<quickSort>();

}

}

ISort::ISort() {

}

void ISort::sort(cMatrix& copyMatrix) {

std::vector <double\*> arrForSort;

for (int k = 0; k < copyMatrix.getSizeRow(); k++) {

for (int j = 0; j < copyMatrix.getSizeColumn(); j++) {

if (k % 2 == 0 && j % 2 == 0) {

arrForSort.push\_back(copyMatrix.getArrPtr(k, j));

}

if (k % 2 == 1 && j % 2 == 1) {

arrForSort.push\_back(copyMatrix.getArrPtr(k, j));

}

}

if (arrForSort.size() > 1)

sortArr(arrForSort, true);

arrForSort.clear();

}

for (int k = 0; k < copyMatrix.getSizeColumn(); k++) {

for (int j = 0; j < copyMatrix.getSizeRow(); j++) {

if (k % 2 == 0 && j % 2 == 1) {

arrForSort.push\_back(copyMatrix.getArrPtr(j, k));

}

if (k % 2 == 1 && j % 2 == 0) {

arrForSort.push\_back(copyMatrix.getArrPtr(j, k));

}

}

if (arrForSort.size() > 1)

sortArr(arrForSort, false);

arrForSort.clear();

}

}

bubbleSort::bubbleSort() {

cMatrix::comparisons[bubbleS] = 0;

cMatrix::permutations[bubbleS] = 0;

}

void bubbleSort::sortArr(std::vector<double\*>& arrForSort, bool methodAscending) {

double tmp = 0;

for (int i = 0; i < arrForSort.size() - 1; i++) {

for (int j = 0; j < arrForSort.size() - i - 1; j++) {

cMatrix::comparisons[bubbleS]++;

if (\*arrForSort[j] > \*arrForSort[j + 1] && methodAscending || \*arrForSort[j] < \*arrForSort[j + 1] && !methodAscending) {

tmp = \*arrForSort[j];

\*arrForSort[j] = \*arrForSort[j + 1];

\*arrForSort[j + 1] = tmp;

cMatrix::permutations[bubbleS]++;

}

}

}

}

selectionSort::selectionSort() {

cMatrix::comparisons[selectionS] = 0;

cMatrix::permutations[selectionS] = 0;

}

void selectionSort::sortArr(std::vector<double\*>& arrForSort, bool methodAscending) {

for (int startIndex = 0; startIndex < arrForSort.size(); startIndex++) {

int smallestIndex = startIndex;

for (int currentIndex = startIndex + 1; currentIndex < arrForSort.size(); currentIndex++) {

cMatrix::comparisons[selectionS]++;

if (\*arrForSort[currentIndex] < \*arrForSort[smallestIndex] && methodAscending || \*arrForSort[currentIndex] > \*arrForSort[smallestIndex] && !methodAscending)

smallestIndex = currentIndex;

}

if (\*arrForSort[startIndex] > \*arrForSort[smallestIndex] && methodAscending || \*arrForSort[startIndex] < \*arrForSort[smallestIndex] && !methodAscending) {

std::swap(\*arrForSort[startIndex], \*arrForSort[smallestIndex]);

cMatrix::permutations[selectionS]++;

}

}

}

insertionSort::insertionSort() {

cMatrix::comparisons[insertS] = 0;

cMatrix::permutations[insertS] = 0;

}

void insertionSort::sortArr(std::vector<double\*>& arrForSort, bool methodAscending) {

double temp;

int item;

for (int counter = 1; counter < arrForSort.size(); counter++) {

temp = \*arrForSort[counter];

item = counter - 1;

cMatrix::comparisons[insertS]++;

while (item >= 0 && (\*arrForSort[item] > temp && methodAscending || \*arrForSort[item] < temp && !methodAscending)){

cMatrix::comparisons[insertS]++;

\*arrForSort[item + 1] = \*arrForSort[item];

\*arrForSort[item] = temp;

item--;

cMatrix::permutations[insertS]++;

}

}

}

shellSort::shellSort() {

cMatrix::comparisons[shellS] = 0;

cMatrix::permutations[shellS] = 0;

}

void shellSort::sortArr(std::vector<double\*>& arrForSort, bool methodAscending) {

double count;

int d = arrForSort.size() / 2;

while (d > 0) {

for (int i = 0; i < arrForSort.size() - d; i++) {

int j = i;

cMatrix::comparisons[shellS]++;

while (j >= 0 && (\*arrForSort[j] > \*arrForSort[j + d] && methodAscending || \*arrForSort[j] < \*arrForSort[j + d] && !methodAscending)) {

count = \*arrForSort[j];

\*arrForSort[j] = \*arrForSort[j + d];

\*arrForSort[j + d] = count;

j--;

cMatrix::permutations[shellS]++;

}

}

d = d / 2;

}

}

void quicksort(std::vector<double\*>& arrForSort, int first, int last, bool methodAscending)

{

double mid, count;

int f = first, l = last;

mid = \*arrForSort[(f + l) / 2];

do

{

if (methodAscending) {

cMatrix::comparisons[quickS]++;

while (\*arrForSort[f] < mid) {

f++;

cMatrix::comparisons[quickS]++;

}

cMatrix::comparisons[quickS]++;

while (\*arrForSort[l] > mid) {

l--;

cMatrix::comparisons[quickS]++;

}

}

else {

cMatrix::comparisons[quickS]++;

while (\*arrForSort[f] > mid) {

f++;

cMatrix::comparisons[quickS]++;

}

cMatrix::comparisons[quickS]++;

while (\*arrForSort[l] < mid) {

l--;

cMatrix::comparisons[quickS]++;

}

}

if (f <= l)

{

count = \*arrForSort[f];

\*arrForSort[f] = \*arrForSort[l];

\*arrForSort[l] = count;

f++;

l--;

cMatrix::permutations[quickS]++;

}

} while (f < l);

if (first < l) quicksort(arrForSort, first, l, methodAscending);

if (f < last) quicksort(arrForSort, f, last, methodAscending);

}

quickSort::quickSort() {

cMatrix::comparisons[quickS] = 0;

cMatrix::permutations[quickS] = 0;

}

void quickSort::sortArr(std::vector<double\*>& arrForSort, bool methodAscending) {

quicksort(arrForSort, 0, arrForSort.size() - 1, methodAscending);

}

//get.h

#pragma once

#include <iostream>

#include <cstdlib>

#include <ctime>

int getInt();

double getDouble();

double random();

//get.cpp

#include "get.h"

double random() {

return rand() % 10000 / 100.0 - 50;

}

int getInt() {

int input = 0;

while (!( std::cin >> input)) {

std::cin.clear();

std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');

std::cout << "Error" << std::endl << "Введите число: ";

}

std::cin.clear();

std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');

return(input);

}

double getDouble() {

double input = 0;

while (!( std::cin >> input)) {

std::cin.clear();

std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');

std::cout << "Error" << std::endl << "Введите число: ";

}

std::cin.clear();

std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');

return(input);

}

//enums.h

#pragma once

enum inputChoise{

ext=0,

randomInput,

keyboardInput,

fileInput

};

enum binarChoise {

yes = 1,

no

};

enum sort {

bubbleS,

selectionS,

insertS,

shellS,

quickS

};

//save.h

#pragma once

#include <iostream>

#include <string>

#include "get.h"

#include "enums.h"

#include <filesystem>

#include <fstream>

#include "cMatrix.h"

bool askRewrite();

bool check(std::string);

void saveData(cMatrix&);

void saveResult(cMatrix&);

//save.cpp

#include "save.h"

bool askRewrite() {

std::cout << "Перезаписать существующий файл?" << std::endl <<

"1 - Да" << std::endl << "2 - Нет" << std::endl;

int saveAnswer = getInt();

if (saveAnswer != yes)

saveAnswer = false;

return saveAnswer;

}

bool check(std::string dir) {

std::ifstream checkTest;

try {

static\_cast<void>(std::filesystem::is\_regular\_file(dir));

}

catch (...) {

std::cout << "Запись невозможна! Использовано регулярное выражение" << std::endl;

return false;

}

checkTest.open(dir);

if (!checkTest.is\_open()) {

checkTest.close();

std::cout << "Создается новый фаил\n";

return true;

}

else {

std::cout << "Файл Уже существует!\n";

checkTest.close();

return askRewrite();

}

return true;

}

void saveData(cMatrix& inputMatrixObj) {

std::cout <<

std::endl << "Введите '1', чтобы сохранить исходные данные" <<

std::endl << "Введите '2', для отмены сохранения" << std::endl << "Сохранить: ";

int saveChoise = getInt();

while (saveChoise != yes && saveChoise != no)

{

std::cout <<

std::endl << "Введите '1', чтобы сохранить исходные данные" <<

std::endl << "Введите '2', для отмены сохранения" << std::endl << "Сохранить: ";

saveChoise = getInt();

}

if (saveChoise == yes) {

std::string dir;

std::cout << "Введите путь сохранения файла: ";

int ch = getchar();

while (ch != '\n') {

dir += char(ch);

ch = getchar();

}

while (!check(dir)) {

std::cout << "Введите путь сохранения файла: ";

dir = "";

ch = getchar();

while (ch != '\n') {

dir += char(ch);

ch = getchar();

}

}

std::ofstream outputTest;

outputTest.open(dir, std::ios\_base::out);

if (!outputTest.is\_open()) {

std::cout << "Запись не удалась" << std::endl;

saveData(inputMatrixObj);

}

else {

for (int i = 0; i < inputMatrixObj.getSizeRow(); i++) {

outputTest << "|";

for (int j = 0; j < inputMatrixObj.getSizeColumn(); j++) {

if (inputMatrixObj.getArr(i, j) > 0)

outputTest << " ";

outputTest << inputMatrixObj.getArr(i, j) << "\t ";

}

outputTest << "|" << std::endl;

}

}

outputTest.close();

}

}

void sortArr(std::vector<double\*>& arrForSort, bool methodAscending) {

double count;

int d = arrForSort.size() / 2;

while (d > 0) {

for (int i = 0; i < arrForSort.size() - d; i++) {

int j = i;

cMatrix::comparisons[shellS]++;

while (j >= 0 && (\*arrForSort[j] > \*arrForSort[j + d] && methodAscending || \*arrForSort[j] < \*arrForSort[j + d] && !methodAscending)) {

count = \*arrForSort[j];

\*arrForSort[j] = \*arrForSort[j + d];

\*arrForSort[j + d] = count;

j--;

cMatrix::permutations[shellS];

}

}

d = d / 2;

}

}

void sort(cMatrix& copyMatrix) {

std::vector <double\*> arrForSort;

for (int k = 0; k < copyMatrix.getSizeRow(); k++) {

for (int j = 0; j < copyMatrix.getSizeColumn(); j++) {

if (k % 2 == 0 && j % 2 == 0) {

arrForSort.push\_back(copyMatrix.getArrPtr(k, j));

}

if (k % 2 == 1 && j % 2 == 1) {

arrForSort.push\_back(copyMatrix.getArrPtr(k, j));

}

}

sortArr(arrForSort, true);

arrForSort.clear();

}

for (int k = 0; k < copyMatrix.getSizeColumn(); k++) {

for (int j = 0; j < copyMatrix.getSizeRow(); j++) {

if (k % 2 == 0 && j % 2 == 1) {

arrForSort.push\_back(copyMatrix.getArrPtr(j, k));

}

if (k % 2 == 1 && j % 2 == 0) {

arrForSort.push\_back(copyMatrix.getArrPtr(j, k));

}

}

sortArr(arrForSort, false);

arrForSort.clear();

}

}

void saveResult(cMatrix& inputMatrixObj) {

std::cout <<

std::endl << "Введите '1', чтобы сохранить резульат выполнения программы" <<

std::endl << "Введите '2', для отмены сохранения" << std::endl << "Сохранить: ";

int saveChoise = getInt();

while (saveChoise != yes && saveChoise != no)

{

std::cout <<

std::endl << "Введите '1', чтобы сохранить резульата выполнения программы" <<

std::endl << "Введите '2', для отмены сохранения" << std::endl << "Сохранить: ";

saveChoise = getInt();

}

if (saveChoise == yes) {

std::string dir;

std::cout << "Введите путь сохранения файла: ";

int ch = getchar();

while (ch != '\n') {

dir += char(ch);

ch = getchar();

}

while (!check(dir)) {

std::cout << "Введите путь сохранения файла: ";

dir = "";

ch = getchar();

while (ch != '\n') {

dir += char(ch);

ch = getchar();

}

}

std::ofstream outputTest;

outputTest.open(dir, std::ios\_base::out);

if (!outputTest.is\_open()) {

std::cout << "Запись не удалась" << std::endl;

saveResult(inputMatrixObj);

}

else {

outputTest

<< " --------------------------------" << std::endl

<< " Исходная матрица:" << std::endl

<< " --------------------------------" << std::endl;

for (int i = 0; i < inputMatrixObj.getSizeRow(); i++) {

outputTest << "|";

for (int j = 0; j < inputMatrixObj.getSizeColumn(); j++) {

outputTest << inputMatrixObj.getArr(i, j)<< "\t ";

}

outputTest << "|" << std::endl;

}

outputTest << std::endl << std::endl;

sort(inputMatrixObj);

outputTest

<< " --------------------------------" << std::endl

<< " Отсортированная матрица:" << std::endl

<< " --------------------------------" << std::endl;

for (int i = 0; i < inputMatrixObj.getSizeRow(); i++) {

outputTest << "|";

for (int j = 0; j < inputMatrixObj.getSizeColumn(); j++) {

if (inputMatrixObj.getArr(i, j) > 0)

outputTest << " ";

outputTest << inputMatrixObj.getArr(i, j) << "\t ";

}

outputTest << "|" << std::endl;

}

outputTest << std::endl <<std::endl

<< "| Метод\t| Сравнений\t| Перестановок\t|" << std::endl

<< "|---------------|---------------|---------------|" << std::endl

<< "|Bubble sort\t|\t" << cMatrix::comparisons[bubbleS] << "\t|\t" << cMatrix::permutations[bubbleS] << "\t|\t" << std::endl

<< "|Selection sort\t|\t" << cMatrix::comparisons[selectionS] << "\t|\t" << cMatrix::permutations[selectionS] << "\t|\t" << std::endl

<< "|Insert sort\t|\t" << cMatrix::comparisons[insertS] << "\t|\t" << cMatrix::permutations[insertS] << "\t|\t" << std::endl

<< "|Shell sort\t|\t" << cMatrix::comparisons[shellS] << "\t|\t" << cMatrix::permutations[shellS] << "\t|\t" << std::endl

<< "|Quick sort\t|\t" << cMatrix::comparisons[quickS] << "\t|\t" << cMatrix::permutations[quickS] << "\t|\t" << std::endl;

outputTest.close();

}

}

}

//test.h

#pragma once

#include "cMatrix.h";

#include <iostream>

bool testFindMedian();

//test.cpp

#include "test.h"

#include "ISort.h"

bool testFindMedian() {

cMatrix testmatrix;

{

for (int i = bubbleS; i <= quickS; i++) {

auto sortType = getTypeSort(i);

sortType->sort(testmatrix);

}

for (int i = 0; i < quickS; i++) {

if (cMatrix::comparisons[i] != 0 || cMatrix::permutations[i] != 0) {

std::cout << "Тест №1 не пройден" << std::endl <<

"Не пройден тест, где матрица не создается." << std::endl;

return false;

}

}

}

{

testmatrix.setSize(2, 4);

for (int i = bubbleS; i <= quickS; i++) {

auto sortType = getTypeSort(i);

sortType->sort(testmatrix);

}

int c[5] = { 2,2,2,2,2 };

int p[5] = { 0,0,0,0,0 };

for (int i = 0; i < quickS; i++) {

if (cMatrix::comparisons[i] != c[i] || cMatrix::permutations[i] != p[i]) {

std::cout << "Тест №2 не пройден" << std::endl <<

"Не пройден тест, где все значения матрицы нули." << std::endl;

return false;

}

}

}

{

for (int i = 0; i < testmatrix.getSizeRow(); i++) {

for (int j = 0; j < testmatrix.getSizeColumn(); j++) {

testmatrix.setArr(i, j, j);

}

}

int c[5] = { 2,2,2,2,2 };

int p[5] = { 0,0,0,0,0 };

for (int i = bubbleS; i <= quickS; i++) {

auto sortType = getTypeSort(i);

sortType->sort(testmatrix);

}

for (int i = 0; i < quickS; i++) {

if (cMatrix::comparisons[i] != c[i] || cMatrix::permutations[i] != p[i]) {

std::cout << "Тест №3 не пройден" << std::endl <<

"Не пройден тест, где все значения массива положительны." << std::endl;

return false;

}

}

}

{

for (int i = 0; i < testmatrix.getSizeRow(); i++) {

for (int j = 0; j < testmatrix.getSizeColumn(); j++) {

testmatrix.setArr(i, j, -j);

}

}

int c[5] = { 2,2,4,2,2 };

int p[5] = { 2,2,2,2,2 };

for (int i = bubbleS; i <= quickS; i++) {

auto sortType = getTypeSort(i);

cMatrix sortMatrix(testmatrix);

sortType->sort(sortMatrix);

}

for (int i = 0; i < quickS; i++) {

if (cMatrix::comparisons[i] != c[i] || cMatrix::permutations[i] != p[i]) {

std::cout << "Тест №4 не пройден" << std::endl <<

"Не пройден тест, где все значения массива отрицательны." << std::endl;

return false;

}

}

}

{

testmatrix.setSize(1, 1);

for (int i = bubbleS; i <= quickS; i++) {

auto sortType = getTypeSort(i);

sortType->sort(testmatrix);

}

for (int i = 0; i < quickS; i++) {

if (cMatrix::comparisons[i] != 0 || cMatrix::permutations[i] != 0) {

std::cout << "Тест №5 не пройден" << std::endl <<

"Не пройден тест, где в массиве сохранен единственный элемент." << std::endl;

return false;

}

}

}

std::cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << std::endl

<< "Тесты пройдены." << std::endl

<< "Программа готова к работе!" << std::endl

<< "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << std::endl;

return true;

}

F v