МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по учебной практике

ТЕМА: ГЕНЕТИЧЕСКИЕ АЛГОРИТМЫ И PSA

Студенты Кроткина З.Э.,
Ларукова А.А.,
Романова К..

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург
2024

Цель работы.

Изучение работы генетических алгоритмов на примере решения задачи о назначениях.

Задание.

Задача о назначениях

Пусть имеется N работ и N кандидатов на выполнение этих работ, причем назначение j-й работы i-му кандидату требует затрат $Cij > 0$.

Необходимо назначить каждому кандидату по работе, чтобы минимизировать суммарные затраты. Причем каждый кандидат может быть назначен на одну работу, а каждая работа может выполняться только одним кандидатом.

Выполнение работы.

Графика

Класс MainWindow

Этот класс является основным окном приложения и наследует от QMainWindow.

Публичные методы и поля:

MainWindow(QWidget *parent = nullptr): Конструктор класса, инициализирующий главное окно приложения.

~MainWindow(): Деструктор класса, освобождающий ресурсы.

bool choice: Логическая переменная для выбора состояния.

Приватные методы и поля:

*Ui::MainWindow ui: Указатель на интерфейс пользователя.

QPalette pl: Объект для управления цветовой палитрой приложения.

*QWidget stripe: Виджет для отображения полоски сверху окна.

*QLabel textTitle: Метка для отображения заголовка на полоске.

*QWidget menu: Виджет для бокового меню.

*QLabel textMenu: Метка для текста в боковом меню.

*QPushButton matrixButton: Кнопка для выбора матрицы.

*QPushButton fileButton: Кнопка для выбора файла.

*QPushButton randomButton: Кнопка для случайной генерации данных.

*QPushButton DaleeButtom: Кнопка "Далее" для подтверждения ввода данных.

QString fileName: Переменная для хранения имени файла.

*QLineEdit line: Поле ввода для имени файла.

QString valMatrix: Переменная для хранения значений матрицы.

int sizeMatrix: Переменная для хранения размера матрицы.

*QSpinBox countSpinBox: Поле для ввода количества работников.

*QLabel countLabel: Метка для поля ввода количества работников.

*QTableWidget matrixTable: Таблица для ввода данных матрицы.

bool allFilled: Переменная для проверки заполненности всех ячеек матрицы.

Описание методов:

SideMenu(): Метод для создания бокового меню.

PushButtonMenu(int x, int y, const QString &text): Метод для создания кнопки в боковом меню.

DataEntryButton(): Метод для создания кнопки "Далее".

Matrix(): Метод для инициализации ввода матрицы.

createMatrix(): Метод для создания и отображения таблицы матрицы.

checkAllCellsFilled(): Метод для проверки заполненности всех ячеек матрицы.

handleFilledMatrix(): Метод для обработки данных после заполнения матрицы.

File(): Метод для инициализации ввода данных из файла.

ReadLine(): Метод для чтения имени файла.

Random(): Метод для случайной генерации данных.

getSize(): Метод для получения размера матрицы.

clearMenuExceptButtons(): Метод для очистки меню от виджетов, кроме кнопок "Матрица", "Из файла" и "Случайная генерация".

Файл mainwindow.cpp

Этот файл содержит реализацию методов класса MainWindow, определенных в mainwindow.h.

Конструктор и деструктор

MainWindow(QWidget *parent): Конструктор инициализирует главное окно, устанавливает размеры и перемещает его на экран. Также он создает и настраивает виджеты полоски сверху и бокового меню.

~MainWindow(): Деструктор освобождает ресурсы, используемые главным окном.

Методы:

void SideMenu(): Создает боковое меню и настраивает его внешний вид.

QPushButton* PushButtonMenu(int x, int y, const QString &text): Создает и возвращает кнопку с заданными координатами и текстом.

QPushButton* DataEntryButton(): Создает и возвращает кнопку "Далее", если состояние выбора (choice) истинно.

void Matrix(): Инициализирует процесс ввода матрицы, создает соответствующие виджеты.

void createMatrix(): Создает таблицу матрицы на основе введенного пользователем значения.

void checkAllCellsFilled(): Проверяет, заполнены ли все ячейки таблицы матрицы, и вызывает метод обработки заполненной матрицы.

void handleFilledMatrix(): Обрабатывает заполненные данные матрицы.

void File(): Инициализирует процесс ввода данных из файла, создает соответствующие виджеты.

void ReadLine(): Читает и сохраняет введенное пользователем имя файла.

void Random(): Инициализирует процесс случайной генерации данных, создает соответствующие виджеты.

void getSize(): Получает и сохраняет размер матрицы.

void clearMenuExceptButtons(): Очищает меню от всех виджетов, кроме кнопок "Матрица", "Из файла" и "Случайная генерация".

Файл main.cpp

Этот файл содержит функцию main, которая запускает приложение.

Функция main

int main(int argc, char *argv[]): Создает экземпляр приложения QApplication и экземпляр главного окна MainWindow, затем отображает главное окно и запускает основной цикл обработки событий.

Описание функций алгоритма

Класс Chromosome: std::vector<int> _workers - вектор, хранящий назначенную рабочему задачу, int _cost - затраты, int _size - размер.

Класс матрицы стоимостей CostMatrix: std::vector<std::vector<int>> _costArray - матрица стоимостей, int _n - размер.

Класс популяции Population: std::vector<Chromosome> _chromosomes - вектор, хранящий объекты типа Chromosome, long _bestChromosomeCost - затраты на лучшую хромосому, int _bestChromosomeIndex - индекс лучшей хромосомы, Chromosome _bestChromosome - лучшая хромосома, bool _maximise - переменная для минимизации.

void Print(int iteration) - функция вывода в терминал информации.

Chromosome Crossover(Chromosome &chr, std::mt19937 &rnd) - функция скрещивания

void Mutation(std::mt19937 &rnd) - функция мутации

void Copy(Chromosome &chr) - копирование обьекта

void Assign(int worker, int task) - функция назначает каждому рабочему задачу

void SetCost(int agent, int task, int cost) - функция присваивает стоимости

int GetChromosomeCost(Chromosome &chromosome, bool maximise) - функция получения стоимости в хромосоме

void CreateArbitraryPopulation(std::mt19937 &rnd, int populationSize, int taskSize) - создаем произвольную популяцию

void Evaluate(CostMatrix &costMatrix, int iteration) - функция оценки

void ApplyCrossover(std::mt19937 &rnd, int taskSize) - функция применения скрещивания

void Crossover(int parentIndex1, int parentIndex2, std::mt19937 &rnd, int taskSize) - ф-я скрещивания, в кач-ве аргументов принимает 2 родителей, случайное число и размер задачи

void Mutate(std::mt19937 &rnd) - функция мутации, в кач-ве аргумента принимает случайное число

bool IsBetter(long cost1, long cost2) - ф-я для определения лучшей стоимости

void Selection(std::mt19937 &rnd) - функция отбора

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД

Файл mainwindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H
#include <iostream>
#include <QtWidgets>
QT_BEGIN_NAMESPACE
namespace Ui {
class MainWindow;
}
QT_END_NAMESPACE
class MainWindow : public QMainWindow
{
    Q_OBJECT
public:
    MainWindow(QWidget *parent = nullptr);
    ~MainWindow();
    bool choice = false ;
private:
    //общее
    Ui::MainWindow *ui;
    QPalette pl;
    QWidget *stripe;
    QLabel *textTitle;
    QWidget *menu;
    QLabel *textMenu;
    QPushButton *matrixButton;
    QPushButton *fileButton;
    QPushButton *randomButton;
    QPushButton *DaleeButtom;
    //файл
    QString fileName = nullptr;
    QLineEdit*line = nullptr;
    //матрица
    QString valMatrix = nullptr;
    int sizeMatrix;
    QSpinBox* countSpinBox;
```

```
    QLabel *countLabel;
    QTableWidget *matrixTable;
    bool allFilled = false;
    //общее
    void SideMenu();
    QPushButton* PushButtonMenu( int x, int y, const QString &text);
    QPushButton* DataEntryButton();
    void createMatrix();
    void Matrix();
    void checkAllCellsFilled();
    void handleFilledMatrix();
    void File();
    void ReadLine();
    void Random();
    void getSize();
    void clearMenuExceptButtons();
};
#endif // MAINWINDOW_H
```

`Файл mainwindow.cpp

```
#include "mainwindow.h"
#include "./ui_mainwindow.h"
#include <iostream>

MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
    , ui(new Ui::MainWindow)

{
    ui->setupUi(this);
    setFixedSize(1400, 750);
    move(100, 20);
    pl.setColor(QPalette::Window, QColor(244,244,244));
    setPalette(pl);

    //полоска сверху
```

```
        stripe = new QWidget(this);
        stripe->setGeometry(0,0,1400,150);
        pl.setColor(QPalette::Window, QColor(44,103,115));
        stripe->setPalette(pl);
        stripe->setAutoFillBackground(true);
        stripe->show();

        //текст на полоске
        textTitle = new QLabel("Назначения", stripe);
        QFont font("Cascadia Code", 74);
        textTitle->setFont(font);
        textTitle->setStyleSheet("color: rgb(244, 244, 244);");

        textTitle->setAlignment(Qt::AlignVCenter);
        QHBoxLayout *stripeLayout = new QHBoxLayout(stripe);
        stripeLayout->addWidget(textTitle);
        stripeLayout->setAlignment(Qt::AlignCenter);



        //боковое меню
        SideMenu();



        matrixButton = PushButtonMenu( 35, 120, "Матрица");
        fileButton = PushButtonMenu( 35, 200, "Из файла");
        randomButton = PushButtonMenu( 35, 280, "Случайная \nгенерация");




        connect(matrixButton, &QPushButton::clicked, this,
&MainWindow::Matrix);
        connect(fileButton, &QPushButton::clicked, this,
&MainWindow::File);
        connect(randomButton, &QPushButton::clicked, this,
&MainWindow::Random);
```

```
}

MainWindow::~MainWindow()
{
    delete ui;
}

void MainWindow::SideMenu(){
    menu = new QWidget(this);
    pl.setColor(QPalette::Window, QColor(44,154,176));
    menu->setPalette(pl);
    menu->setAutoFillBackground(true);
    menu->setGeometry(0, 150, 300,600);



    textMenu = new QLabel("Способ ввода \nданных:", menu);
    QFont font("Cascadia Code", 25 );
    textMenu->setFont(font);

    textMenu->setStyleSheet("color: rgb(244, 244, 244);");
    textMenu->setAlignment(Qt::AlignLeft | Qt::AlignTop);
    QHBoxLayout *stripeLayout = new QHBoxLayout(menu);
    stripeLayout->addWidget(textMenu);
    stripeLayout->setAlignment(textMenu, Qt::AlignLeft |
Qt::AlignTop);



}
```

```
    QPushButton* MainWindow::PushButtonMenu( int x, int y, const QString
&text)
    {
        QPushButton *button = new QPushButton(text, menu);
        QFont font("Cascadia Code", 20);
        button->setFont(font);

        button->setStyleSheet("color: rgb(244, 244, 244);");
        pl.setColor(QPalette::Button, QColor(44,154,176));
        button->setPalette(pl);
        button->setGeometry(x, y, 200, 70); // Задайте размеры кнопки
(ширину и высоту) по вашему усмотрению
        return button;
    }



    void MainWindow::Matrix(){




        clearMenuExceptButtons();

        countLabel = new QLabel("Количество \nработников:", menu);
        countSpinBox = new QSpinBox(menu);
        countSpinBox->setMinimum(2);  // Минимальное значение
        countSpinBox->setMaximum(10);

        QFont font("Cascadia Code", 15);
        countLabel->setFont(font);
        countLabel->setStyleSheet("color: rgb(244, 244, 244);");
        countLabel->setGeometry(35, 350, 150, 80);
        countLabel->show();
```

```
        countSpinBox->setGeometry(200, 370, 50,40);
        countSpinBox->show();



        choice = true;
        DaleeButtom = this->DataEntryButton();
        connect(DaleeButtom, &QPushButton::clicked, this,
&MainWindow::createMatrix);









    }

    void MainWindow::createMatrix()
    {
        clearMenuExceptButtons();
        matrixTable = new QTableWidget(countSpinBox->value(),
countSpinBox->value(), menu);
        for (int i = 0; i < countSpinBox->value(); ++i) {
            matrixTable->setColumnWidth(i, 200/countSpinBox->value()); //
Adjust the width as needed
        }
        matrixTable->setGeometry(10, 350, 280, 200);
        matrixTable->show();
        DaleeButtom = this->DataEntryButton();
        connect(DaleeButtom, &QPushButton::clicked, this,
&MainWindow::checkAllCellsFilled);



    }

    void MainWindow::checkAllCellsFilled()
```

```
{

    bool allFilled = true;

    for (int i = 0; i < matrixTable->rowCount(); ++i) {
        for (int j = 0; j < matrixTable->columnCount(); ++j) {
            QTableWidgetItem *item = matrixTable->item(i, j);
            if (!item ||item->text().isEmpty()) {
                allFilled = false;
                break;
            }
        }
        if (!allFilled) {
            break;
        }
    }

    if (allFilled){

        handleFilledMatrix();
    }



}

void MainWindow::handleFilledMatrix()
{
    sizeMatrix = matrixTable->rowCount();

    for (int i = 0; i < matrixTable->rowCount(); ++i) {
        for (int j = 0; j < matrixTable->rowCount(); ++j) {
            QTableWidgetItem *item = matrixTable->item(i, j);
            if (item) {
                valMatrix.append(item->text()) ;
            }
        }
```

```
        }


    }

    void MainWindow::File(){

        clearMenuExceptButtons();



        QLabel *text = new QLabel("Название файла:", menu);
        text->setFont(QFont("Cascadia Code", 15));
        text->setStyleSheet("color: rgb(244, 244, 244);");
        text->move(35, 370);
        text->show();

        line = new QLineEdit(menu);
        line->setGeometry(35, 400, 200, 30);
        line->show();
        this->choice = true;
        DaleeButtom = this->DataEntryButton();
        connect(DaleeButtom, &QPushButton::clicked, this,
&MainWindow::ReadLine);



    }
    void MainWindow::ReadLine(){
        fileName = line->text();
    }



    void MainWindow::Random(){
        clearMenuExceptButtons();
        countLabel = new QLabel("Количество \nработников:", menu);
        countSpinBox = new QSpinBox(menu);
```

```
        countSpinBox->setMinimum(2);  // Минимальное значение
        countSpinBox->setMaximum(10);

        QFont font("Cascadia Code", 15);
        countLabel->setFont(font);
        countLabel->setStyleSheet("color: rgb(244, 244, 244);");
        countLabel->setGeometry(35, 350, 150, 80);
        countLabel->show();
        countSpinBox->setGeometry(200, 370, 50,40);
        countSpinBox->show();



        choice = true;
        DaleeButtom = this->DataEntryButton();
        connect(DaleeButtom, &QPushButton::clicked, this,
&MainWindow::getSize);
    }

    void MainWindow::getSize(){
        sizeMatrix = countSpinBox->value();

    }
    QPushButton* MainWindow::DataEntryButton(){
        if (choice){
            QPushButton *button = new QPushButton("Далее", menu);
            QFont font("Cascadia Code", 17);
            button->setFont(font);
            button->setStyleSheet("color: rgb(244, 244, 244);");
            button->setGeometry(200, 550, 90, 40);
            pl.setColor(QPalette::Button, QColor(44,103,115));
            button->setPalette(pl);
            button->show();
            return button;

        }
        return nullptr;
    }
```

```
void MainWindow::clearMenuExceptButtons()
{

    QList<QWidget*> children = this->findChildren<QWidget*>();

    for (QWidget *child : children) {

        if (child != matrixButton && child != fileButton && child != randomButton && child != textMenu && child != DaleeButtom && child != menu && child != stripe && child != textTitle) {
            child->hide(); /
            child->deleteLater();
        }
    }
}
```

Файл main.cpp

```
#include "mainwindow.h"
#include <QApplication>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    MainWindow w;
    w.show();

    return a.exec();
}
```