

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

**ОТЧЕТ**  
**по учебной практике**  
**Тема: Задача о назначениях**

Студенты гр. 0304

Руководитель

\_\_\_\_\_

\_\_\_\_\_

Решоткин А.С.  
Докучаев Р.А.  
Крицын Д.Р.  
Козиков А.Е.

Жангиров Т.Р.

Санкт-Петербург

2022

## ЗАДАНИЕ НА УЧЕБНУЮ ПРАКТИКУ

Студенты Решоткин А.С., Докучаев Р.А., Крицын Д.Р., Козиков А.Е.

Группа 0304

Тема практики: Задача о назначениях.

Задание на практику:

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

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.

Сроки прохождения практики: 29.06.2022 – 12.07.2022

Дата сдачи отчета: 2.07.2022

Дата защиты отчета: 2.07.2022

Студенты

\_\_\_\_\_

Решоткин А.С.  
Докучаев Р.А.  
Крицын Д.Р.,  
Козиков А.Е.

Руководитель

\_\_\_\_\_

Жангиров Т.Р.

## **АННОТАЦИЯ**

Целью работы является знакомство и применение на практике генетических алгоритмов, а также их оптимизаций, для решения поставленной задачи о назначениях. Генетические алгоритмы — это адаптивные методы поиска, которые в последнее время используются для решения задач оптимизации. В них используются как аналог механизма генетического наследования, так и аналог естественного отбора. При этом сохраняется биологическая терминология в упрощенном виде и основные понятия линейной алгебры.

## СОДЕРЖАНИЕ

	Введение	5
1.	Итерация 2	6
1.1.	Скетч с GUI, который планируется реализовать.	7
1.2.	Описание сценариев взаимодействия пользователя с программой	7
1.3	Определение и обоснование параметров модификации ГА для решения задачи.	7
2.	Итерация 3	9
2.1	Частичная реализация GUI	9
2.2	Реализовано хранение данных и основные элементы ГА.	11
	Заключение	12
	Список использованных источников	13
	Приложение А. Исходный код программы	14

## **ВВЕДЕНИЕ**

Целью работы является программная реализация решения поставленной оптимизационной задачи на языке C++ с использованием ГА. Основными задачами выполнения работы являются: формирование прототипа GUI и выбор метода решения задачи, частичная реализация программы, в которой присутствует GUI и реализовано хранения данных и основные элементы ГА. Также создана инструкция по сборке и запуску программы. На конечной итерации должна быть выполнена цель работы, а именно программа должна полностью работать вместе с её графической частью, ГА должен гарантированно находить решения.

## 1. Итерация 2

### 1.1. Скетч с GUI, который планируется реализовать.

		-	X
<b>Настройка:</b>			
Ввод значений:			
<input <="" td="" type="button" value="Овыбрать файл..."/> <td colspan="2"><input type="radio"/> Ввести в программе</td>		<input type="radio"/> Ввести в программе	
<b>Параметры ГА:</b>			
Размер начальной популяции:	<input type="text"/>		
Критерий остановки:	<input type="text"/>		
Коэффициент влияния:	<input type="text"/>		
Плотность мутации:	<input type="text"/>		
количество особей при элитарном отборе:	<input type="text"/>		

		-	X
<b>Пошаговая визуализация:</b>			
Начальная популяция:			
Особи	<input type="text"/>	<input type="text"/>	
приспособленность	<input type="text"/>	<input type="text"/>	
Выбор родителей:			
Особи	<input type="text"/>	<input type="text"/>	
Вероятность	<input type="text"/>	<input type="text"/>	
Кроссинговер:			
Родители	<input type="text"/>	<input type="text"/>	<input type="text"/>
Потомки	<input type="text"/>		<input type="text"/>

		-	X
<b>Ввод данных:</b>			
<input type="text" value="Введите значение N"/>			
кандидаты:	1	...	
Работы:			
1			
...			

		-	X
<b>Пошаговая визуализация:</b>			
Мутации:			
До мутации	<input type="text"/>	<input type="text"/>	
После мутации	<input type="text"/>	<input type="text"/>	
Новая популяция:			
особи	<input type="text"/>	<input type="text"/>	
лучшая особь:	<input type="text"/>		
<input type="button" value="К ответу &gt;&gt;"/>		<input type="button" value="Следующий шаг &gt;"/>	

		-	X
<b>Результат:</b>			
Распределение работ:			
Работник	<input type="text"/>	<input type="text"/>	
Работа	<input type="text"/>	<input type="text"/>	
Суммарные затраты:	<input type="text"/>		
<input type="button" value="Задать новые данные"/>			

## **1.2. Описание сценариев взаимодействия пользователя с программой**

Возможный сценарий взаимодействия пользователя с программой представлен ниже.

- 1) Пользователь запускает программу.
- 2) Пользователю предоставляется выбор: ввести данные из файла или ввести данные в меню.
- 3) Если пользователь выбирает ввод данных из файла, то программа проверяет корректность введенных данных и запускает алгоритм, выводя промежуточные итерации в виде таблицы.
- 4) Если пользователь выбирает ввод данных в меню, то программа соответственно проверяет наличие всех необходимых данных для решения задачи, а затем решает задачу в соответствии с алгоритмом, выводя промежуточные результаты в виде таблицы.
- 5) Пользователь на выходе получает ответ на задачу в виде таблицы.

## **1.3 Определение и обоснование параметров модификации ГА для решения задачи.**

Пробные решения для генетического алгоритма (хромосомы) представлены в виде перестановок, которые будут переведены в числа в факториальной системе счисления при помощи кода Лемера, которые затем будут переведены в двоичную систему счисления.

Оператор выбора родителей - метод рулетки. Выбор обоснован тем, что при настройке макропараметров можно настроить формулу отбора, изменив влияние приспособленности на выбор родителей: это может быть полезно, так как расположение и количество локальных минимумов зависят от поданной на вход матрицы затрат.

Оператор рекомбинации (кроссинговера) - многоточечный кроссинговер. В отличие от кроссинговера с фиксированным количеством точек, такой метод

позволяет лучше адаптироваться к разным размерностям входной матрицы затрат, в то же время не затрачивая много времени на генерацию случайной двоичной строки, как в методе однородного кроссинговера.

Оператор мутации - мутация с использованием понятия плотности. В отличие от двоичной мутации, работает без особых отличий для двоичных строк разной размерности, что хорошо подходит для данной задачи (размерность матрицы затрат заранее неизвестна).

Оператор отбора в новую популяцию - элитарный отбор в сочетании с отбором усечением. Выбор обоснован тем, что содержимое поданной на вход матрицы заранее неизвестно, и при не сильно отличающихся значениях элементов такой матрицы элитарный отбор не будет терять решения, сошедшиеся к локальным экстремумам, которых может быть довольно много. При этом слишком неперспективные решения рассматривать нет особого смысла - лучше отсеять их и попытаться выйти из локального минимума за счёт выбора случайных хорошо приспособленных особей через отбор усечением и их дальнейшей мутации.



## 2. Итерация 3

### 2.1 Частичная реализация GUI.

Генетический алгоритм

Настройки

Ввод значений

Через файл

Через программу

Настройки генетического алгоритма

Значение N:

1

—

+

Начальный размер популяции:

1

—

+

Критерий остановки:

1

—

+

Коэффициент влияния:

1

—

+

Плотность мутации:

1

—

+

Количество особей элитного отбора:

1

—

+

Следующий шаг

Генетический алгоритм

Введите значения:

	1	2	3	4
1	1	—	+	1
2	1	—	+	1
3	1	—	+	1
4	1	—	+	1
5	1	—	+	1

Следующий шаг

# Генетический алгоритм

## Пошаговая визуализация:

### Особи Приспособленность

34	32
23	23

### Особи Вероятность

34	32
23	23

### Родители Потомки

34	32
23	23

### До мутации После мутации

34	32
23	23

### Новая популяция

34	32
23	23

### Лучшая особь

34	32
23	23

Следующий шаг

К результату

# Генетический алгоритм

## Результат:

### Работник Работа

1	15.3
2	2.4
3	0.05
4	22.3
5	42

Ещё раз

# Генетический алгоритм

Cancel

## Выберите файл

Q

Open

## Настройки

### Ввод значений

☒ Через файл ☐ Через программу

### Настройки генетического алгоритма

Значение N: 1 - +

Начальный размер популяции: 1 - +

Критерий остановки: 1 - +

Коэффициент влияния: 1 - +

Плотность мутации: 1 - +

Количество особей элитного отбора: 1 - +

Следующий шаг

## Недавние

## Домашняя папка

## Видео

## Документы

## Загрузки

## Изображения

## Музыка

## sumProj

## Загрузки

## JNS

## jns

## ubuntu-server

## RAN

## + Другие места

## <

## alex

## Рабочий стол

## sumProj

## >

## Имя

## Размер

## Тип

## Изменён

## main.cpp

249 байт

Текст

Сб

## SpinBox.cpp

318 байт

Текст

Сб

## Makefile

318 байт

Текст

Вс

## InputTable.h

211 байт

Текст

Вчера

## InputTable.cpp

772 байта

Текст

Вчера

## StepTable.cpp

803 байта

Текст

Вчера

## StepTable.h

647 байт

Текст

Вчера

## exp.txt

20 байт

Текст

Вчера

## ResultFrame.h

272 байта

Текст

Вчера

## res.txt

32 байта

Текст

Вчера

## ResultFrame.cpp

331 байт

Текст

Вчера

## SpinBox.h

253 байта

Текст

13:15

## ChooseFileDialog.h

274 байта

Текст

16:04

## ChooseFileDialog.cpp

989 байт

Текст

16:25

## StepFrame.h

455 байт

Текст

16:32

## StepFrame.cpp

1,2 кБ

Текст

16:34

## SettingsFrame.cpp

2,3 кБ

Текст

16:54

## SettingsFrame.h

537 байт

Текст

16:54

## mainWindow.h

476 байт

Текст

17:07

## mainWindow.cpp

1,3 кБ

Текст

17:11

## InputFrame.h

255 байт

Текст

17:11

## InputFrame.cpp

325 байт

Текст

17:11

Text files

Что реализовано в GUI:

Поля для настройки параметров и способ задания данных. Границы макропараметров ГА всегда зависят от заданного значения N, значения для всех пунктов могут быть только больше 0. При выборе через файл открывается меню выбора. При выборе через программу, открывается матрица "работник x работа", в которой можно задать неотрицательные значения для сложности. Далее происходит пошаговый вывод работы ГА, с возможностью перейти к результату. В результате выводится таблица работник и подходящая ему работа. При желании можно запустить программу вновь, не закрывая её.

## 2.2 Реализовано хранения данных и основные элементы ГА

1) Хранение данных реализовано с помощью вектора `current_population` (вектора двоичных строк).

2) Основные элементы ГА реализованы с помощью функций:

a) `fitness_func` - функция приспособленности. Чем больше значение, тем больше приспособленность (знак учитывается).

b) `recombination_op` - оператор рекомбинации. Возвращает одного или несколько потомков двух родителей. Предполагается, что количество генов в обоих родителях совпадает (в большинстве видов такого оператора).

c) `mutation_op` - оператор мутации. Возвращает модифицированную хромосому.

d) `parent_selection_op` - оператор выбора родителей. Выбор производится из поданного на вход списка особей.

e) `survivor_selection_op` - оператор отбора в новую популяцию. Напрямую модифицирует поданный на вход вектор.

Данные функции задаются в конструкторе класса `GARunner`. Функция `do_iteration` производит одну итерацию ГА: выбирает родителей, формирует потомков, создает новую популяцию и возвращает количество потомков. Функция `thread_iterate_count` совершает заданное количество итераций, функция `thread_iterate_until_no_children` совершает итерации пока не будет произведено ни одного потомка. `Thread_terminate` останавливает генетический алгоритм.

## **ЗАКЛЮЧЕНИЕ**

На второй итерации практики удалось создать скетч с графическим интерфейсом, который будет в программе, определить сценарий взаимодействия пользователя с программой, определить и обосновать модификации ГА, которые были выбраны для решения поставленной задачи. На третьей итерации было реализовано GUI и основные элементы ГА.

## **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Панченко Т.В. Учебно-методическое пособие “Генетический алгоритмы”.

## ПРИЛОЖЕНИЕ А

### ИСХОДНЫЙ КОД ПРОГРАММЫ

#### Файл bool\_string.cpp

```
#include "bool_string.h"

std::ostream& operator<<(std::ostream& os, BoolString bs)
{
    for(size_t i = 0; i < bs.size(); ++i)
        os << bs[i];
    return os;
}
```

#### Файл bool\_string.h

```
#ifndef BOOL_STRING_H
#define BOOL_STRING_H

#include <ostream>
#include <vector>

class BoolString : public std::vector<bool>
{
};

std::ostream& operator<<(std::ostream& os, BoolString bs);

#endif
```

#### Файл ga\_runner.h

```
#ifndef GA_RUNNER_H
#define GA_RUNNER_H

#include <iostream>
#include <vector>
#include <random>
#include <pthread.h>

#include "bool_string.h"

template<typename data>
class GARunner
{
private:
    size_t population_size;
    std::vector<data> current_population;

    double mutation_chance;
    double (*fitness_func)(data);

    std::vector<data> (*recombination_op)(data, data);
    data (*mutation_op)(data);
    std::pair<data, data> (*parent_selection_op)(std::vector<data>, double (*fitness_func_arg)(data));
    std::vector<data> (*survivor_selection_op)(std::vector<data>, size_t amount, double (*fitness_func_arg)(data));
};
```

```

std::random_device dev_urandom;

void print_data(std::vector<data> wh)
{
    for(size_t i = 0; i < wh.size(); ++i){
        std::cout << wh[i] << ' ';
    }
    std::cout << '\n';
}

pthread_t work_thread;

pthread_mutex_t thread_mode_mutex;
enum {
    GATerminated,
    GADoNIterations,
    GADoUntilNoChildren
} thread_mode;
size_t iter_counter;

friend void* thread_iteration(void* _args);
static void* thread_iteration(void* _args)
{
    GARunner* self = (GARunner*)_args;
    size_t prev_children = 1;
    while(1){
        pthread_mutex_lock(&self->thread_mode_mutex);
        switch(self->thread_mode){
            case GATerminated: goto terminate;
            case GADoNIterations: if(self->iter_counter == 0)
goto terminate; else --self->iter_counter; break;
            case GADoUntilNoChildren: if(prev_children == 0)
goto terminate; break;
        }
        pthread_mutex_unlock(&self->thread_mode_mutex);
        prev_children = self->do_iteration();
    }
    terminate:
    self->thread_mode = GATerminated;
    pthread_mutex_unlock(&self->thread_mode_mutex);
    return NULL;
}

public:
    std::vector<data>& get_current_population() { return cur-
rent_population; }

    GARunner(size_t population_size, std::vector<data> ini-
tial_population,
             double mutation_chance,
             double (*fitness_func)(data),
             std::vector<data> (*recombination_op)(data, data),
             data (*mutation_op)(data),
             std::pair<data, data>
(*parent_selection_op)(std::vector<data>, double (*fitness_func_arg)(data)),
             std::vector<data> survivor_selection_op(std::vector<data>,
size_t amount, double (*fitness_func_arg)(data)))
        : population_size(population_size), cur-
rent_population(initial_population), mutation_chance(mutation_chance),
        fitness_func(fitness_func), recombination_op(recombination_op), mu-
tation_op(mutation_op), parent_selection_op(parent_selection_op), survi-
vor_selection_op(survivor_selection_op)

```

```

{
    thread_mode = GATerminated;
    iter_counter = 0;
    pthread_mutex_init(&thread_mode_mutex, NULL);
}

size_t do_iteration()
{
    std::vector<data> children;
    for(size_t i = 0; i <= population_size; i += 2){
        // отбираем двух родителей
        std::pair<data, data> parents = parent_selection_op(current_population, fitness_func);
        std::cout << "parents: "; print_data({parents.first,
parents.second});

        if(parents.first != parents.second){
            std::vector<data> more_children = recombination_op(parents.first, parents.second); // проводим скрещивание
            std::copy(more_children.begin(),
more_children.end(), std::back_inserter(children));
        }
        std::cout << "children: "; print_data(children);

        std::mt19937 rng(dev_urandom());
        std::uniform_real_distribution<double> mutation_dist(0, 1);
        for(size_t i = 0; i < children.size(); ++i)
            if(mutation_dist(rng) < mutation_chance)
                children[i] = mutation_op(children[i]); //
производим мутацию с заданным шансом
        std::cout << "mutated children: "; print_data(children);

        // заносим детей и родителей в один список и отбираем оттуда
особей для следующей популяции
        std::copy(children.begin(), children.end(),
std::back_inserter(current_population));
        current_population = survivor_selection_op(current_population,
population_size, fitness_func);

        std::cout << "current population: ";
print_data(current_population);

        return children.size();
    }

void thread_terminate()
{
    pthread_mutex_lock(&thread_mode_mutex);
    thread_mode = GATerminated;
    pthread_mutex_unlock(&thread_mode_mutex);
    pthread_join(work_thread, NULL);
}

void thread_iterate_count(size_t count)
{
    bool create_thread = (thread_mode == GATerminated);
    pthread_mutex_lock(&thread_mode_mutex);
    iter_counter = count;
    thread_mode = GADoNIterations;
    pthread_mutex_unlock(&thread_mode_mutex);
    if(create_thread)
        pthread_create(&work_thread, NULL, GArunner::thread_iteration, (void*)this);
}

```



```

        void thread_iterate_until_no_children()
        {
            bool create_thread = (thread_mode == GATerminated);
            pthread_mutex_lock(&thread_mode_mutex);
            thread_mode = GADoUntilNoChildren;
            pthread_mutex_unlock(&thread_mode_mutex);
            if(create_thread)
                pthread_create(&work_thread, NULL, GARunner::thread_iteration, (void*)this);
        }

        void print_current_population() { print_data(current_population); }
};

#endif

```

## Файл mutation\_ops.cpp

```

#include "mutation_ops.h"
#include <random>

static std::random_device dev_urandom;
static std::mt19937 rng(dev_urandom());

double density_mutation_chance = 0.01;
BoolString density_mutation_op(BoolString bs)
{
    BoolString _out = bs;
    std::uniform_real_distribution<double> mut_dist(0, 1);
    for(size_t i = 0; i < bs.size(); ++i){
        if(mut_dist(rng) <= density_mutation_chance)
            _out[i] = !_out[i];
    }
    return _out;
}

```

## Файл mutation\_ops.h

```

#ifndef MUTATION_OPS_H
#define MUTATION_OPS_H

#include "ga_runner.h"

/* Мутация бинарных строк */

// Мутация с использованием понятия плотности
extern double density_mutation_chance; // шанс мутации (инверсии) каждого гена: [0; 1], по умолчанию 0.01
BoolString density_mutation_op(BoolString bs);

#endif

```

Файл parent\_selection\_ops.cpp

```

#include "parent_selection_ops.h"
#include <random>

static std::random_device dev_urandom;
static std::mt19937 rng(dev_urandom());

double roulette_fitness_influence = 0.8;

```

```

std::pair<BoolString, BoolString> roulette_bs_selection_op(std::vector<BoolString> individuals,

double (*fitness_func)(BoolString))
{
    double fsum = 0; // сумма всех значений функций приспособленности
    for(size_t i = 0; i < individuals.size(); ++i)
        fsum += fitness_func(individuals[i]);

    BoolString p1, p2; unsigned pcnt = 0;
    std::uniform_real_distribution<double> roulette_dist(0, 1);
    double roll1 = roulette_dist(rng), roll2 = roulette_dist(rng);

    double fcur = 0;
    for(size_t i = 0; i < individuals.size(); ++i){
        double finc = (fitness_func(individuals[i]) / fsum) * roulette_fitness_influence + (1. / individuals.size()) * (1 - roulette_fitness_influence);
        if(fcur < roll1 && fcur + finc >= roll1) { p1 = individuals[i]; ++pcnt; }
        if(fcur < roll2 && fcur + finc >= roll2) { p2 = individuals[i]; ++pcnt; }
        if(pcnt == 2) break;
        fcur += finc;
    }
    return {p1, p2};
}

```

## Файл parent\_selection\_ops.h

```

#ifndef PARENT_SELECTION_OPS_H
#define PARENT_SELECTION_OPS_H

#include "ga_runner.h"

/* Выбор родителей для бинарных строк */

// Метод рулетки
extern double roulette_fitness_influence; // влияние значения функции приспособленности на шанс хромосомы быть выбранной в методе рулетки: [0, 1]
std::pair<BoolString, BoolString> roulette_bs_selection_op(std::vector<BoolString> individuals,

double (*fitness_func)(BoolString));

#endif

```

## Файл recombination\_ops.cpp

```

#include "recombination_ops.h"
#include <random>
#include <cmath>

static std::random_device dev_urandom;
static std::mt19937 point_rng(dev_urandom());
std::vector<BoolString> multi_point_crossingover(BoolString bs1, BoolString bs2)
{
    std::vector<size_t> points;
    // Генерируем ceil(ln(N)) уникальных точек разреза
    std::uniform_int_distribution<size_t> point_dist(0, bs1.size() - 1);
    for(size_t i = 0; i < ceil(log(bs1.size())); ++i){

```

```

        while(1){
            size_t pt = point_dist(point_rng);
            int repeat = 0;
            for(size_t j = 0; j < i; ++j)
                if(points[j] == pt)
                    { repeat = 1; break; }
            if(!repeat){
                points.push_back(pt);
                break;
            }
        }
        std::sort(points.begin(), points.end());
        // Генерируем первого потомка: берём первую булеву строку и вставляем в
        неё фрагменты из второй согласно точкам разреза
        BoolString ch1 = bs1;
        for(size_t i = 0; i < points.size(); i += 2){
            if(i != points.size() - 1){
                std::copy(bs2.begin() + points[i], bs2.begin() + points[i +
1], ch1.begin() + points[i]);
            }
            else{ // обмен генами с закольцовыванием от конца в начало
                std::copy(bs2.begin() + points[i], bs2.end(), ch1.begin() +
points[i]);
                std::copy(bs2.begin(), bs2.begin() + points[0], ch1.begin());
            }
        }
        // Генерируем второго потомка: берём вторую булеву строку и вставляем в
        неё фрагменты из первой согласно точкам разреза
        BoolString ch2 = bs2;
        for(size_t i = 0; i < points.size(); i += 2){
            if(i != points.size() - 1){
                std::copy(bs1.begin() + points[i], bs1.begin() + points[i +
1], ch2.begin() + points[i]);
            }
            else{ // обмен генами с закольцовыванием от конца в начало
                std::copy(bs1.begin() + points[i], bs1.end(), ch2.begin() +
points[i]);
                std::copy(bs1.begin(), bs1.begin() + points[0], ch2.begin());
            }
        }
        return {ch1, ch2};
    }
}

```

## Файл recombination\_ops.h

```

#ifndef RECOMBINATION_OPS
#define RECOMBINATION_OPS

#include "ga_runner.h"

/* Бинарная рекомбинация (кроссинговер) */

// Количество точек равно ceil(ln(N)), где N - количество бит в двоичных строках
std::vector<BoolString> multi_point_crossingover(BoolString bs1, BoolString
bs2);

#endif

```

## Файл survivor\_selection\_ops.cpp

```

#include "survivor_selection_ops.h"

```

```

#include <random>

static std::random_device dev_urandom;
static std::mt19937 rng(dev_urandom());

struct FitnessBsComp
{
    double (*fitness_func)(BoolString);
    FitnessBsComp(double (*fitness_func)(BoolString)) { this->fitness_func =
fitness_func; }
    bool operator()(BoolString a, BoolString b) { return fitness_func(a) >
fitness_func(b); }
};

double survivor_selection_elite_fraction = 0.2;
double survivor_selection_truncate_threshold = 0.4;
std::vector<BoolString>
elite_truncation_survivor_selection_op(std::vector<BoolString> individuals,
size_t amount,
double (*fitness_func)(BoolString))
{
    std::sort(individuals.begin(), individuals.end(), Fitness-
BsComp(fitness_func));

    size_t elite_amount = ceil(amount * survivor_selection_elite_fraction);
    std::vector<BoolString> new_individuals;
    std::copy(individuals.begin(), individuals.begin() + elite_amount,
std::back_inserter(new_individuals));

    size_t trunc_amount = floor(amount * (1 - survi-
vor_selection_elite_fraction));
    size_t trunc_fraction = ceil(amount * survi-
vor_selection_truncate_threshold);

    std::uniform_int_distribution<size_t> trunc_dist(elite_amount,
elite_amount + trunc_fraction);
    for(size_t i = 0; i < trunc_amount; ++i)
        new_individuals.push_back(individuals[trunc_dist(rng)]);

    return new_individuals;
}

```

## Файл survivor\_selection\_ops.h

```

#ifndef SURVIVOR_SELECTION_OPS_H
#define SURVIVOR_SELECTION_OPS_H

#include "ga_runner.h"

/* Отбор особей в новую популяцию для бинарных строк */

// Метод рулетки
extern double survivor_selection_elite_fraction;
// доля особей, выбираемая элитарным отбором: [0, 1]
extern double survivor_selection_truncate_threshold; //
// доля лучших особей, участвующая в отборе усечением: (0, 1]
std::vector<BoolString>
elite_truncation_survivor_selection_op(std::vector<BoolString> individuals, //
комбинация элитарного отбора и отбора усечением
size_t amount,
double (*fitness_func)(BoolString));

```

```
#endif
```