**БУ ВО «Сургутский государственный университет»**

Политехнический институт

Кафедра автоматизированных систем обработки информации и управления

**ОТЧЕТ**

ПО ЛАБОРАТОРНОЙ РАБОТЕ №2

ПО ТЕМЕ «Генетический алгоритм для задачи подбора товаров»

ПО ДИСЦИПЛИНЕ «Интеллектуальные системы»

Выполнил: студент группы №606-12,

Демьянцев Виталий Владиславович

Принял: ст. преподаватель кафедры АСОИУ,

Гавриленко Анна Владимировна

Сургут 2025

[1 Введение 3](#_Toc197463976)

[2 Основныефункциипрограммы 4](#_Toc197463977)

[4 Функции в коде 6](#_Toc197463978)

[4.1 InitializeItems() 6](#_Toc197463979)

[4.2 PopulateGrid() 6](#_Toc197463980)

[4.3 UpdatePriceRange() 6](#_Toc197463981)

[4.4 NumberValidation(object sender, TextCompositionEventArgs e) 6](#_Toc197463982)

[4.5 StartButton\_Click(object sender, RoutedEventArgs e) 6](#_Toc197463983)

[4.6 RunGeneticAlgorithm(int targetSum, int maxIterations) 6](#_Toc197463984)

[4.7 CalculateFitness(int[] solution, int targetSum) 6](#_Toc197463985)

[5 Заключение 8](#_Toc197463986)

# 1 Введение

Целью данной работы является разработка программы для решения задачи оптимального подбора товаров с использованием генетического алгоритма. Программа реализована на языке программирования C# с использованием технологии WPF (Windows Presentation Foundation).

Задача заключается в выборе количеств товаров из заданного списка (50 наименований) таким образом, чтобы их общая стоимость была как можно ближе к целевой сумме, указанной пользователем. Каждый товар имеет свою цену и ограничения на минимальное и максимальное количество. Программа визуализирует список товаров, позволяет пользователю задавать целевую сумму и количество итераций алгоритма, а также отображает результаты выполнения.

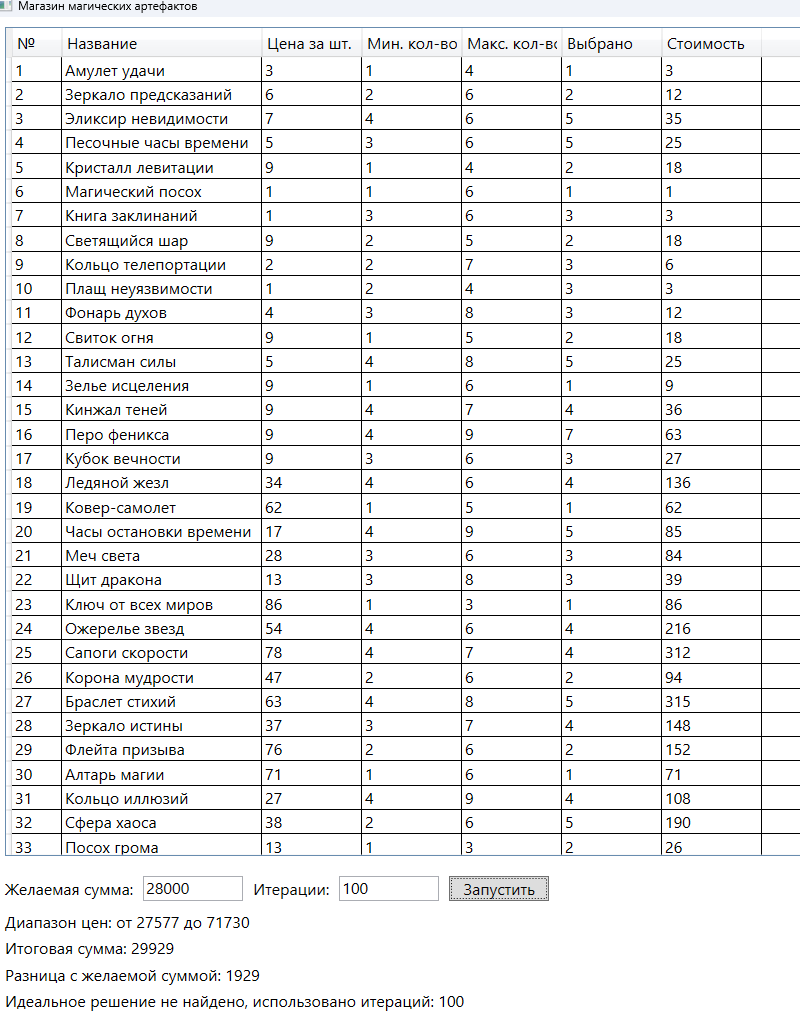
# 2 Основныефункциипрограммы

1. **Инициализацияспискатоваров:** Программа генерирует 50 товаров с уникальными названиями, ценами и ограничениями на количество. Цены варьируются в трёх диапазонах: низком (1–10), среднем (10–100) и высоком

(100–1000).

1. **Отображение данных:** Список товаров представлен в виде таблицы, где пользователь может видеть номер, название, цену, минимальное и максимальноеколичество,а такжевыбранноеколичествои общую стоимость для каждого товара.
2. **Генетический алгоритм:** Реализован алгоритм, который использует мутацию и скрещивание для поиска оптимального набора количеств товаров. Алгоритм минимизирует разницу между текущей и целевой суммой.
3. **Валидацияввода:** Программа проверяет корректность введённых данных (целевая сумма и количество итераций) и уведомляет пользователя в случае ошибок.
4. **Обновлениеинтерфейса:** После выполнения алгоритма табли recoverable errorцаобновляется,отображаявыбранныеколичестваирезультаты,включая итоговую сумму, разницу с целевой суммой и информацию об итерациях.

**3 Скриншотпрограммы**



# 4 Функции в коде

## 4.1 InitializeItems()

Инициализирует массив из 50 товаров, генерируя случайные цены и ограничения на количество. Названия товаров заранее определены в массиве строк.

## 4.2 PopulateGrid()

Обновляет таблицу в интерфейсе, связывая её с массивом товаров для отображения текущих данных.

## 4.3 UpdatePriceRange()

Вычисляет минимальную и максимальную возможную стоимость набора товаров на основе их цен и ограничений, отображая диапазон в интерфейсе.

## 4.4 NumberValidation(object sender, TextCompositionEventArgs e)

Проверяет, что вводимые пользователем данные в поля целевой суммы и количества итераций содержат только цифры.

## 4.5 StartButton\_Click(object sender, RoutedEventArgs e)

Обрабатывает нажатие кнопки «Запустить». Проверяет корректность ввода, запускает генетический алгоритм и обновляет интерфейс с результатами.

## 4.6 RunGeneticAlgorithm(int targetSum, int maxIterations)

Реализует основной цикл генетического алгоритма. Создаёт начальную популяцию,выполняетмутациюискрещивание,оцениваетприспособленностькаждой особи и выбирает лучшую. Возвращает лучшее решение, количество использованных итераций и флаг точного совпадения.

## 4.7 CalculateFitness(int[] solution, int targetSum)

Вычисляет приспособленность решения как абсолютную разницу между текущей суммой и целевой.

# 5 Заключение

Программа успешно решает задачу подбора товаров с использованием генетического алгоритма. Она предоставляет удобный интерфейс для ввода параметров и визуализации результатов, а также демонстрирует эффективность эволюционного подхода для оптимизации. Гибкость программы позволяет использовать её для различных входных данных, что делает её полезной для учебных целей и демонстрации принципов генетических алгоритмов.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Input;

namespace GeneticAlgorithm

{

public partial class MainWindow : Window

{

private readonly Random random = new();

private readonly Item[] items = new Item[50];

private readonly int[] quantities = new int[50];

public MainWindow()

{

InitializeComponent();

InitializeItems();

PopulateGrid();

UpdatePriceRange();

}

// Класс для представления товара

public class Item

{

public int Number { get; set; }

public string Name { get; set; }

public int Price { get; set; }

public int MinQuantity { get; set; }

public int MaxQuantity { get; set; }

public int SelectedQuantity { get; set; }

public int TotalCost => Price \* SelectedQuantity;

}

// Инициализация товаров

private void InitializeItems()

{

string[] names = new string[]

{

"Амулет удачи", "Зеркало предсказаний", "Эликсир невидимости", "Песочные часы времени", "Кристалл левитации",

"Магический посох", "Книга заклинаний", "Светящийся шар", "Кольцо телепортации", "Плащ неуязвимости",

"Фонарь духов", "Свиток огня", "Талисман силы", "Зелье исцеления", "Кинжал теней",

"Перо феникса", "Кубок вечности", "Ледяной жезл", "Ковер-самолет", "Часы остановки времени",

"Меч света", "Щит дракона", "Ключ от всех миров", "Ожерелье звезд", "Сапоги скорости",

"Корона мудрости", "Браслет стихий", "Зеркало истины", "Флейта призыва", "Алтарь магии",

"Кольцо иллюзий", "Сфера хаоса", "Посох грома", "Трон теней", "Кубик судьбы",

"Лампа желаний", "Кольцо времени", "Скрижаль пророчеств", "Пояс гиганта", "Очки ясновидения",

"Крылья ангела", "Свеча вечности", "Чаша жизни", "Молот грома", "Арфа гармонии",

"Сумка без дна", "Кинжал молний", "Палочка превращений", "Кольцо защиты", "Книга тайн"

};

// Генерация цен

int[] prices = new int[50];

for (int i = 0; i < 17; i++) prices[i] = random.Next(1, 10);

for (int i = 17; i < 34; i++) prices[i] = random.Next(10, 100);

for (int i = 34; i < 50; i++) prices[i] = random.Next(100, 1000);

// Генерация мин/макс количеств

for (int i = 0; i < 50; i++)

{

int min = random.Next(1, 5);

int max = random.Next(min + 2, min + 6);

items[i] = new Item

{

Number = i + 1,

Name = names[i],

Price = prices[i],

MinQuantity = min,

MaxQuantity = max,

SelectedQuantity = random.Next(min, max + 1)

};

quantities[i] = items[i].SelectedQuantity;

}

}

// Заполнение таблицы

private void PopulateGrid()

{

ItemsGrid.ItemsSource = null;

ItemsGrid.ItemsSource = items;

}

// Обновление диапазона цен

private void UpdatePriceRange()

{

int minPrice = items.Sum(item => item.Price \* item.MinQuantity);

int maxPrice = items.Sum(item => item.Price \* item.MaxQuantity);

PriceRangeLabel.Text = $"Диапазон цен: от {minPrice} до {maxPrice}";

}

// Валидация ввода чисел

private void NumberValidation(object sender, TextCompositionEventArgs e)

{

e.Handled = !char.IsDigit(e.Text, 0);

}

// Обработчик кнопки "Запустить"

private void StartButton\_Click(object sender, RoutedEventArgs e)

{

if (string.IsNullOrEmpty(TargetSum.Text) || string.IsNullOrEmpty(Iterations.Text))

{

MessageBox.Show("Введите сумму и количество итераций!", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);

return;

}

int targetSum = int.Parse(TargetSum.Text);

int iterations = int.Parse(Iterations.Text);

if (targetSum == 0)

{

MessageBox.Show("Сумма не может быть нулевой!", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);

return;

}

// Генетический алгоритм

(int[] bestSolution, int usedIterations, bool isPerfect) = RunGeneticAlgorithm(targetSum, iterations);

// Обновление выбранных количеств

for (int i = 0; i < 50; i++)

{

items[i].SelectedQuantity = bestSolution[i];

}

// Обновление таблицы

PopulateGrid();

// Вывод результатов

int finalSum = items.Sum(item => item.TotalCost);

int difference = Math.Abs(finalSum - targetSum);

ResultSumLabel.Text = $"Итоговая сумма: {finalSum}";

DifferenceLabel.Text = $"Разница с желаемой суммой: {difference}";

IterationResultLabel.Text = isPerfect

? $"Найдено идеальное решение на итерации {usedIterations}"

: $"Идеальное решение не найдено, использовано итераций: {usedIterations}";

}

// Генетический алгоритм

private (int[] bestSolution, int usedIterations, bool isPerfect) RunGeneticAlgorithm(int targetSum, int maxIterations)

{

// Начальная популяция (одна особь)

int[] original = new int[50];

for (int i = 0; i < 50; i++)

{

original[i] = random.Next(items[i].MinQuantity, items[i].MaxQuantity + 1);

}

int[] bestSolution = (int[])original.Clone();

int bestFitness = CalculateFitness(original, targetSum);

int usedIterations = 0;

bool isPerfect = false;

for (int iter = 0; iter < maxIterations; iter++)

{

usedIterations = iter + 1;

// Мутация: создаём две особи с одной мутацией

int[] mutated1 = (int[])original.Clone();

int[] mutated2 = (int[])original.Clone();

// Мутация в первой половине для первой особи

int index1 = random.Next(0, 25);

mutated1[index1] = random.Next(items[index1].MinQuantity, items[index1].MaxQuantity + 1);

// Мутация во второй половине для второй особи

int index2 = random.Next(25, 50);

mutated2[index2] = random.Next(items[index2].MinQuantity, items[index2].MaxQuantity + 1);

// Скрещивание двух мутировавших особей

int[] cross1 = new int[50];

int[] cross2 = new int[50];

for (int i = 0; i < 25; i++)

{

cross1[i] = mutated1[i];

cross2[i] = mutated2[i];

}

for (int i = 25; i < 50; i++)

{

cross1[i] = mutated2[i];

cross2[i] = mutated1[i];

}

// Оценка всех популяций

var populations = new[] { original, mutated1, mutated2, cross1, cross2 };

int bestPopIndex = 0;

int minFitness = CalculateFitness(original, targetSum);

for (int i = 1; i < populations.Length; i++)

{

int fitness = CalculateFitness(populations[i], targetSum);

if (fitness < minFitness)

{

minFitness = fitness;

bestPopIndex = i;

}

}

// Обновление лучшего решения

if (minFitness < bestFitness)

{

bestFitness = minFitness;

bestSolution = (int[])populations[bestPopIndex].Clone();

}

// Проверка на точное совпадение

if (minFitness == 0)

{

isPerfect = true;

break;

}

// Новая оригинальная популяция

original = (int[])populations[bestPopIndex].Clone();

}

return (bestSolution, usedIterations, isPerfect);

}

// Вычисление приспособленности

private int CalculateFitness(int[] solution, int targetSum)

{

int total = 0;

for (int i = 0; i < 50; i++)

{

total += solution[i] \* items[i].Price;

}

return Math.Abs(total - targetSum);

}

}

}