СОДЕРЖАНИЕ

ОПРЕДЕЛЕНИЯ И СОКРАЩЕНИЯ

В настоящей пояснительной записке применяются следующие определения и сокращения.

Ген – элементарная единица информации, которая может быть изменена и переданы от одного поколения генетического алгоритма к другому.

Кодирование – процесс представления решений задачи в виде последовательности генов.

Хромосома – структура данных, которая представляет генетическую информацию решения задачи в генетическом алгоритме. Хромосома может быть представлена в виде последовательности генов, которые кодируют определенные характеристики решения.

Фитнес-функция – функция, которая используется для оценки качества каждого решения в популяции. Она определяет, насколько хорошо данное решение соответствует заданным критериям оптимизации.

Селекция – процесс выбора родительских хромосом из текущей популяции для скрещивания и создания потомства в следующем поколении. Целю селекции является сохранение лучших решений в популяции и исключение менее подходящих решений.

Скрещивание – процесс комбинации генетического материала родительских хромосом для создания потомства в следующем поколении. Оно является одним из ключевых операторов генетического алгоритма, который позволяет создавать новые решения на основе существующих.

Родитель – хромосома, при скрещивании с другим родителем производит ребенка, также являющегося хромосомой.

Прото-ребенок – хромосома, с частично не дополненным геномом на промежуточном шаге алгоритма.

Ребенок – хромосома, получающаяся в результате скрещивания двух родителей.

Мутация – оператор, который случайным образом изменяет генетический материал в хромосоме с целью внести разнообразие в популяцию и обнаружить новые оптимальные решения. Мутация происходит после скрещивания и позволяет избежать застревания в локальных оптимумах.

Популяция – множество хромосом, которые представляют текущее поколение решений задачи. Она является основным объектом работы генетического алгоритма и обычно состоит из нескольких десятков или сотен хромосом.

Элитизм – техника в генетическом алгоритме, при которой хромосома с наилучшей фитнес-функцией переходит в следующее поколение при любых обстоятельствах.

Composable – главный элемент в декларативном подходе к построению пользовательского интерфейса.

ГА – Генетический алгоритм.

KMM – Kotlin Multiplatform Mobile, технология для кросс-платформенной разработки.

UI – User Interface, или пользовательский интерфейс.

MVVM – Model-View-ViewModel, архитектурный паттерн для разработки android-приложений.

ВВЕДЕНИЕ

В настоящее время существует множество задач, связанных с поиском оптимального маршрута между городами, которые могут быть решены с помощью современных технологий. Одним из наиболее перспективных подходов к решению данной задачи является использование генетического алгоритма [1].

Цель данного дипломного проекта – разработка приложения под платформу Android для поиска оптимального маршрута между городами с использованием генетического алгоритма и его вариаций. Приложение предназначено для использования в различных сферах, таких как путешествия, грузоперевозки, сфера доставок, разного рода инспекции и т.д. Оно решает ровно одну проблему – при наличии достаточно большого количества точек для посещения на автомобиле построить оптимальный маршрут для сокращения расстояния, времени пути и экономии топлива.

Первая глава записки посвящена генетическому алгоритму, который используется для поиска оптимального маршрута между городами. В рамках этой главы рассмотрены альтернативы, основные принципы работы генетического алгоритма, его основные компоненты (кодирование, селекция, скрещивание, мутация) и особенности применения в задаче поиска оптимального маршрута между городами.

Вторая глава содержит обзор технологий, которые были использованы и будут использоваться в дальнейшем при разработке приложения. Её задача – сформировать полный набор основных необходимых инструментов для реализации приложения и последующей монетизации.

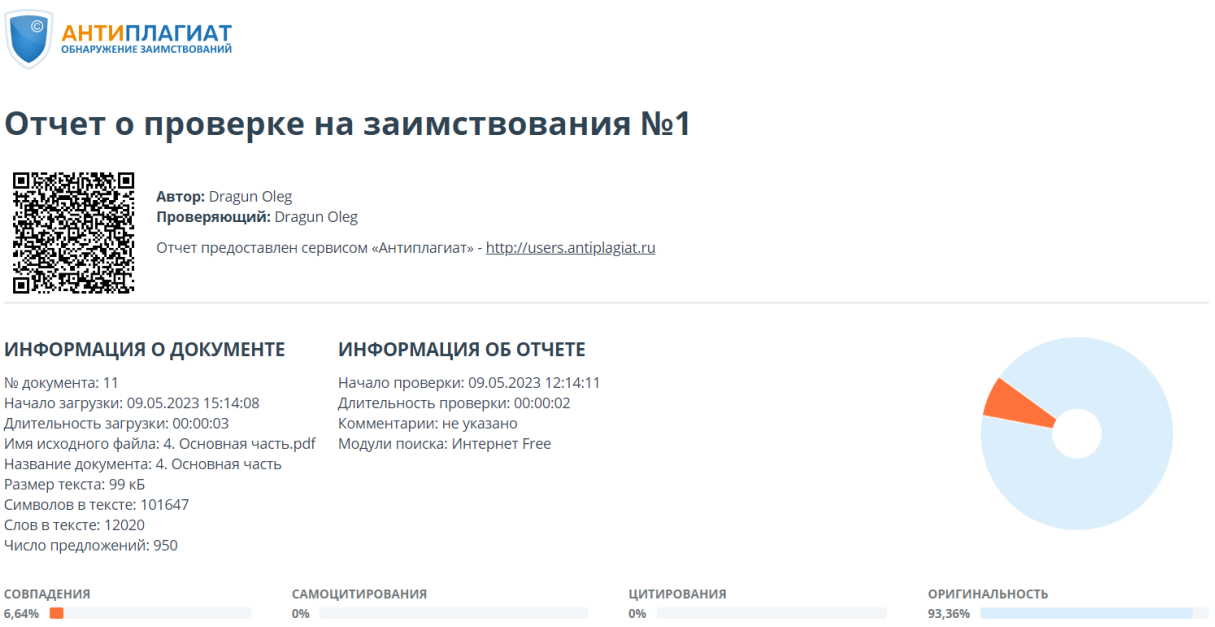
Третья глава описывает общую архитектуру, обзор ключевых частей приложения, особенности реализации каждой из этих частей. Её задача – описать, как именно запаковать абстрактную задачу поиска оптимального маршрута в мобильное приложение, какие структуры данных, api, подходы и принципы в современной мобильной разработки были применены и будут применены для реализации генетического алгоритма, нахождения матрицы расстояний, динамического формирования задачи, построения маршрута и создания привлекательного пользовательского интерфейса.

Четвертая глава посвящена технико-экономическому обоснованию разработки мобильного приложения для поиска оптимального маршрута между городами. Она содержит расчет затрат на первые полтора года разработки и развития приложения, расчет потенциальной прибыли и будет интересна потенциальным инвесторам.

Ограничение поиска оптимального маршрута только белорусскими городами обусловлено ограниченным временем на разработку и необходимостью доказать жизнеспособность концепта. Ограничение платформой android обусловлено желанием как можно быстрее выйти на рынок хотя бы одной мобильной ОС, а также простотой расширения на платформу iOS с помощью

кроссплатформенного кода. Выбор Беларуси позволяет сфокусироваться на ограниченном наборе данных, что упрощает задачу по разработке и тестированию приложения. При этом полученные результаты могут служить доказательством жизнеспособности концепта, а в дальнейшем приложение может быть расширено на другие регионы и страны.

Дипломный проект проверен в системе «Атиплагиат». Процент оригинальности соответствует норме, установленной кафедрой информатики. Цитирования обозначены ссылками на публикации, указанные в «Списке использованных источников».



АНТИПЛАГИАТ
ОБНАРУЖЕНИЕ ЗАИМСТВОВАНИЙ

Отчет о проверке на заимствования №1

Автор: Dragun Oleg
Проверяющий: Dragun Oleg

Отчет предоставлен сервисом «Антиплагиат» - http://users.antiplagiat.ru

ИНФОРМАЦИЯ О ДОКУМЕНТЕ

№ документа: 11
Начало загрузки: 09.05.2023 15:14:08
Длительность загрузки: 00:00:03
Имя исходного файла: 4. Основная часть.pdf
Название документа: 4. Основная часть
Размер текста: 99 кБ
Символов в тексте: 101647
Слов в тексте: 12020
Число предложений: 950

ИНФОРМАЦИЯ ОБ ОТЧЕТЕ

Начало проверки: 09.05.2023 12:14:11
Длительность проверки: 00:00:02
Комментарии: не указано
Модули поиска: Интернет Free

СОВПАДЕНИЯ	САМОЦИТИРОВАНИЯ	ЦИТИРОВАНИЯ	ОРИГИНАЛЬНОСТЬ
6,64%	0%	0%	93,36%

1 ГЕНЕТИЧЕСКИЙ АЛГОРИТМ

1.1 Возможные решения для исходной задачи

Исходная задача нахождения оптимального маршрута является np-сложной задачей, с экспоненциальным ростом возможных решений по мере роста количества участвующих в задаче элементов. Для её решения существует множество возможных алгоритмом, перечислим некоторые из них и коротко опишем каждый.

1.1.1 Жадный алгоритм

Жадный алгоритм для решения оптимизации транспортной задачи работает по принципу нахождения наименьшей стоимости перевозки грузов из каждого источника до каждого потребителя, начиная с наименьших затрат. Этот алгоритм прост в реализации и может дать приближенный результат за короткое время, но не гарантирует нахождение оптимального решения.

1.1.1 2-Opt алгоритм

2-Opt – это алгоритм оптимизации маршрута, использующий принцип локального поиска. Алгоритм пытается улучшить качество текущего решения, переставляя порядок двух произвольных вершин маршрута, и проверяет, улучшится ли стоимость маршрута. Этот алгоритм может работать быстро и дать хороший результат в некоторых случаях, но не гарантирует нахождение глобального оптимума.

1.1.2 Жадный 2-Opt алгоритм

Жадный 2-Opt – это алгоритм, который комбинирует принципы жадного алгоритма и 2-Opt. Он начинает с поиска наименьшей стоимости перевозки грузов из каждого источника до каждого потребителя, а затем применяет 2-Opt для оптимизации маршрута. Этот алгоритм может работать быстро и давать хороший результат, но не гарантирует нахождение глобального оптимума.

1.1.3 3-Opt алгоритм

3-Opt – это алгоритм оптимизации маршрута, который расширяет принципы 2-Opt, переставляя порядок трех вершин маршрута, а не двух, как в 2-Opt. Этот алгоритм может работать лучше, чем 2-Opt, но имеет большую вычислительную сложность.

1.1.4 Жадный 3-Opt алгоритм

Жадный 3-Opt – это алгоритм, который комбинирует принципы жадного алгоритма и 3-Opt. Он начинает с поиска наименьшей стоимости перевозки грузов из каждого источника до каждого потребителя, а затем применяет 3-

Opt для оптимизации маршрута. Этот алгоритм может работать быстро и давать хороший результат, но не гарантирует нахождение глобального оптимума.

1.1.5 Генетический алгоритм

Генетический алгоритм для решения оптимизации транспортной задачи использует принципы эволюционной биологии, чтобы найти оптимальное решение. Алгоритм создает популяцию случайных решений и выполняет операции скрещивания и мутации, чтобы создавать новые решения. Затем оценивается стоимость каждого решения, и те, которые показывают лучшие результаты, используются для создания следующего поколения решений. Процесс повторяется, пока не будет найдено оптимальное решение.

1.1.6 Алгоритм иммитации отжига

Имитации отжига – это алгоритм, который использует техники стохастического моделирования для оптимизации транспортной задачи. Он начинает со случайного решения и выполняет последовательность случайных изменений в решении, которые могут как улучшить, так и ухудшить стоимость решения. Вероятность принятия изменения, которое ухудшает решение, определяется температурой системы, которая постепенно снижается во время выполнения алгоритма. Этот алгоритм может работать хорошо для оптимизации сложных задач, но требует настройки параметров.

1.1.7 Нейронная сеть

Нейронная сеть – это алгоритм, который использует искусственные нейронные сети для оптимизации транспортной задачи. Нейронная сеть обучается на данных о предыдущих решениях транспортной задачи, чтобы предсказывать оптимальный маршрут для новых данных. Она может работать хорошо для оптимизации задач с большим количеством переменных, но требует большого объема данных и вычислительных ресурсов для обучения.

1.1.8 Муравьиная колония

Муравьиная колония – это алгоритм, который использует принципы поведения муравьев для оптимизации транспортной задачи. Алгоритм создает колонию виртуальных муравьев, которые перемещаются по графу между источниками и потребителями, оставляя феромоны на своем пути. Чем больше феромонов оставляют муравьи на определенном маршруте, тем больше вероятность, что другие муравьи выберут этот маршрут. Этот алгоритм может работать хорошо для задач оптимизации.

Ввиду ограничения мощности процессора на множестве мобильных устройств, рекомендации из последних исследований в области решений задач оптимизации, [2], [3], ограничения по времени на реализацию минимально рабочей версии, наиболее перспективным вариантом выглядит Генетический алгоритм (далее по тексту – ГА). Рассмотрим его подробнее.

1.2 Общая схема ГА

Генетический алгоритм (ГА) – это оптимизационный метод, использующий механизмы естественного отбора и генетической рекомбинации для решения различных задач. Он включает в себя 6 главных шагов (рисунок 1.1).

Рисунок 1.1 – Схема ГА

1.2.1 Инициализация

Создание начальной популяции из случайных решений (хромосом). Может содержать параметр обязательной уникальности между хромосомами. Также для задач с большим числом параметром может иметь место неслучайная генерация хромосом, что не является классической инициализацией в ГА.

1.2.2 Оценка

Оценка каждого индивида в популяции на основе фитнесс-функции которая определяет качество решения. Для транспортной задачи это либо

суммарная дистанция между начальной и конечной точкой, со всеми остальными точками между ними, либо суммарное ожидаемое время в пути.

1.2.3 Селекция

После оценки генетических кодов происходит выбор лучших кодов для создания следующего поколения. Это делается на основе ранжирования генетических кодов по их фитнес-функции. Лучшие генетические коды имеют больший шанс быть выбранными для скрещивания и создания следующего поколения. Существуют разные стратегии селекции, которые мы рассмотрим далее.

1.2.4 Скрещивание

Скрещивание генов от двух разных родителей. Выбранные хромосомы скрещиваются, чтобы создать новые хромосомы для следующего поколения. Скрещивание может происходить по различным правилам, таким как одноточечное или многоточечное скрещивание, или смешивание генов

1.2.5 Мутация

Некоторые генетические коды в следующем поколении могут подвергаться мутации, что приводит к изменению их генетического материала. Мутация может случайно изменить гены генетического кода, что иногда помогает избежать застревания в локальном оптимуме.

1.2.6 Повторение

Все эти шаги повторяются, пока не будет достигнуто желаемое решение или пока не будет исчерпан лимит итераций. При каждом повторении популяция становится более оптимизированной, поскольку лучшие генетические коды имеют больший шанс продолжить свое существование.

1.3 Кодирование в ГА

Рассмотрим различные варианты кодирования хромосомы, возможные для ГА. В зависимости от задачи должен выбираться наиболее подходящий вариант.

1.3.1 Двоичное кодирование

Этот метод заключается в представлении каждой переменной решения в двоичной форме, используя битовую строку. Так, каждый бит представляет собой одно из двух возможных значений (0 или 1), которые соответствуют определенному значению переменной. Например, если переменная имеет диапазон значений от 0 до 15, то для ее кодирования потребуется 4 бита.

Пусть имеется 9 генов, каждый из которых может принимать значение 0 или 1. Возможная хромосома изображена на рисунке 1.2.

Рисунок 1.2 – Пример двоичного кодирования

1.3.2 Восьмеричное кодирование

Этот метод использует восьмеричную систему счисления для кодирования каждой переменной. Таким образом, каждый символ в строке соответствует одному значению переменной. Например, если переменная имеет диапазон значений от 0 до 63, то для ее кодирования потребуется 2 символа восьмеричной системы.

Пусть имеется 9 генов, каждый из которых может принимать значение от 0 до 7. Возможная хромосома изображена на рисунке 1.3.

Рисунок 1.3 – Пример восьмеричного кодирования

1.3.3 Шестнадцатеричное кодирование

Этот метод использует шестнадцатеричную систему счисления для кодирования каждой переменной. Таким образом, каждый символ в строке соответствует одному значению переменной. Например, если переменная имеет диапазон значений от 0 до 255, то для ее кодирования потребуется 2 символа шестнадцатеричной системы.

Пусть имеется 9 генов, каждый из которых может принимать значение от 0 до 15. Возможная хромосома изображена на рисунке 1.4.

Рисунок 1.4 – Пример шестнадцатеричного кодирования

1.3.4 Перестановочное кодирование

Этот метод используется для кодирования переменных, которые представляют собой наборы элементов, упорядоченных в определенном порядке. Каждый элемент пронумерован, и кодирование осуществляется путем представления порядка элементов в виде перестановки. Например, для набора из 5 элементов возможно 5! = 120 различных перестановок.

Пусть имеется 9 генов, которые могут принимать любое из 9 возможных значений без повторений. Возможная хромосома изображена на рисунке 1.5.

Рисунок 1.5 – Пример перестановочного кодирования

1.3.5 Кодирование значения

Этот метод используется для кодирования переменных, которые представляют собой дискретные значения, такие как целые числа, булевы или категориальные переменные. Каждое значение переменной назначается уникальный код, который может быть использован для генерации новых вариантов комбинаций. Например, для переменной, которая может принимать одно из четырех возможных значений (например, "красный", "зеленый", "синий", "желтый"), каждому значению может быть назначен уникальный код (например, 00, 01, 10, 11).

Пусть имеется 9 генов, каждый из которых может принимать значение от 1 до 5. Возможная хромосома изображена на рисунке 1.6.

Рисунок 1.6 – Пример кодирования значения

1.3.6 Кодирование деревом

Этот метод используется для кодирования структур данных, таких как деревья или графы. Каждый узел в дереве или графе кодируется отдельно, и кодирование осуществляется путем представления каждого узла в виде строки или последовательности символов. Например, для бинарного дерева каждый узел может быть закодирован с использованием 3 символов, соответствующих

значению узла и его двум потомкам.

Пусть имеется дерево, состоящее из 9 узлов. Каждый узел может быть либо листом, либо внутренним узлом, и иметь метку соответствующую его значению. Тогда хромосома может быть закодирована в виде последовательности меток узлов дерева. Возможная хромосома изображена на рисунке 1.7.

Рисунок 1.7 – Пример кодирования деревом

Задача поиска оптимального маршрута идеально кодируется при помощи перестановочного кодирования. Стоит оценить количество всех возможных перестановок для различного количества городов, чтобы иметь представление о сложности задачи по мере роста количества городов. В таблице 1.1 приведено количество перестановок для разного числа городов.

Таблица 1.1 – Количество перестановок для разного числа городов

Число городов	Количество перестановок
10	3628800
50	30414093201713378043612608166064768844377641568960512000000000000
90	1485715964481761497309522733620825737885569961284688766942216863704985393094065876545992131370884059645617234469978112000000000000000000000000
130	6466855489220473672507304395536485253155359447828049608975952322944781961185526165512707047229268452925683969240398027149120740074042105844737747799459310029635780991774612983803150965145600000000000000000000000000000

Для 130 городов это ~6.5 E+219. При взгляде на это число становится понятно, насколько сложную задачу способен решить генетический алгоритм при корректной имплементации.

1.4 Селекция в ГА

Селекция – важный шаг генетического алгоритма, который определяет, будет ли конкретная хромосома участвовать в процессе скрещивания или нет.

Скорость сходимости ГА зависит от давления селекции. Хорошо известными методами селекции являются колесо рулетки, ранг, турнир, Больцман и стохастическая универсальная выборка.

Селекция *колесом рулетки* отображает все возможные хромосомы на колесе с частью колеса, выделенной для них в соответствии с их значением фитнес-функции. Затем это колесо случайным образом вращается для выбора конкретных хромосом, которые будут участвовать в формировании следующего поколения [4]. Однако данная селекция страдает от многих проблем, например ошибками, вызванными его стохастическим характером.

Де Йонг и Бриндл модифицировали метод выбора колеса рулетки, чтобы устранить ошибки, введя концепцию детерминизма в процедуру селекции. *Селекция рангом* – это модифицированная форма селекции колесом рулетки. Она использует ранги вместо значения фитнес-функции. Ранги присваиваются хромосомам в соответствии с их фитнес-функцией, так что каждая хромосома получает шанс быть выбранной в соответствии со своим рангом. Метод ранговой селекции снижает вероятность преждевременной сходимости решения к локальным минимумам.

Техника турнирной селекции была впервые предложена Бриндлом в 1983 году. Хромосомы выбираются парами в соответствии с их значениями фитнес-функции из стохастического колеса рулетки. После селекции особи с более высоким значением фитнесс-функции добавляются в пул следующего поколения. В этом методе селекции каждая особь сравнивается со всеми n-1 другими особями, если она достигает конечной популяции.

Стохастическая универсальная выборка является расширением существующего метода селекции колесом рулетки. Этот метод использует случайную начальную точку в списке особей из поколения и выбирает новую особь через равные промежутки [5]. Это дает равные шансы всем хромосомам быть отобранными для участия в скрещивании следующего поколения. Хотя в случае задачи коммивояжера стохастическая универсальная выборка работает хорошо, но по мере увеличения размера задачи традиционный выбор колеса рулетки работает относительно хорошо [6].

Выбор Больцмана основан на методах энтропии и дискретизации, которые используются в моделировании Монте-Карло. Это помогает в решении проблемы преждевременной сходимости [7]. Вероятность выбора наилучшей хромосомы очень высока, хотя она выполняется за очень короткое время. Однако существует вероятность потери информации. Этого можно избежать при помощи элитизма [8]. Выбор элитизма был предложен К. Д. Джонгом (1975) для улучшения эффективности выбора колеса рулетки. Это гарантирует, что элитарная хромосома в поколении всегда передается следующему поколению. Если хромосома с наивысшим значением фитнес-функции не присутствует в следующем поколении после обычной процедуры селекции, то элитарная особь также автоматически включается в следующее поколение. Сравнение вышеупомянутых методов селекции показано в таблице 1.2.

Таблица 1.2 – Сравнение разных техник селекции в ГА

Техника селекции	Преимущества	Недостатки
Колесо рулетки	Легок в имплементации. Прост. Непредвзят.	Риск преждевременной конвергенции. Зависит от дисперсии, присутствующей в фитнес-функции.
Ранг	Сохраняет разнообразие. Непредвзят.	Медленная сходимость. Требуется сортировка. Вычислительно дорого.
Турнир	Сохраняет разнообразие. Параллельная реализация. Сортировка не требуется.	Потеря разнообразия при большом размере турнира.
Больцман	Достигает глобального оптимума.	Вычислительно дорого.
Стохастическая универсальная выборка	Быстрый метод. Непредвзят.	Преждевременная конвергенция.
Элитизм	Сохраняет лучшую особь в популяции.	Лучшая хромосома может быть потеряна из-за операторов скрещивания и мутации.

1.5 Скрещивание в ГА

Существует множество вариантов скрещивания в зависимости от выбранной системы кодирования для конкретной задачи. Рассмотрим подробнее операции скрещивания, применимые к перестановочному кодированию, поскольку именно это кодирование используется в задаче поиска оптимального маршрута. Будем использовать общепринятые названия на английском, поскольку общепринятых устоявшихся названий в русскоязычной литературе не встречается.

1.5.1 Order crossover

Order crossover характеризуется следующим набором шагов:

1) Выбрать случайную подстроку генов родителя.

2) Сформировать прото-ребенка путем копирования подстроки родителя в соответствующие позиции.

3) Удалить гены, присутствующие в подстроке у второго родителя. Результирующая последовательность генов содержит гены, которых не достает прото-ребенку.

4) Поставить гены на незанятые позиции прото-ребенка слева направо в соответствии с порядком в последовательности, полученной на предыдущем шаге для получения ребенка.

Схематично шаги изображены на рисунке 1.8.

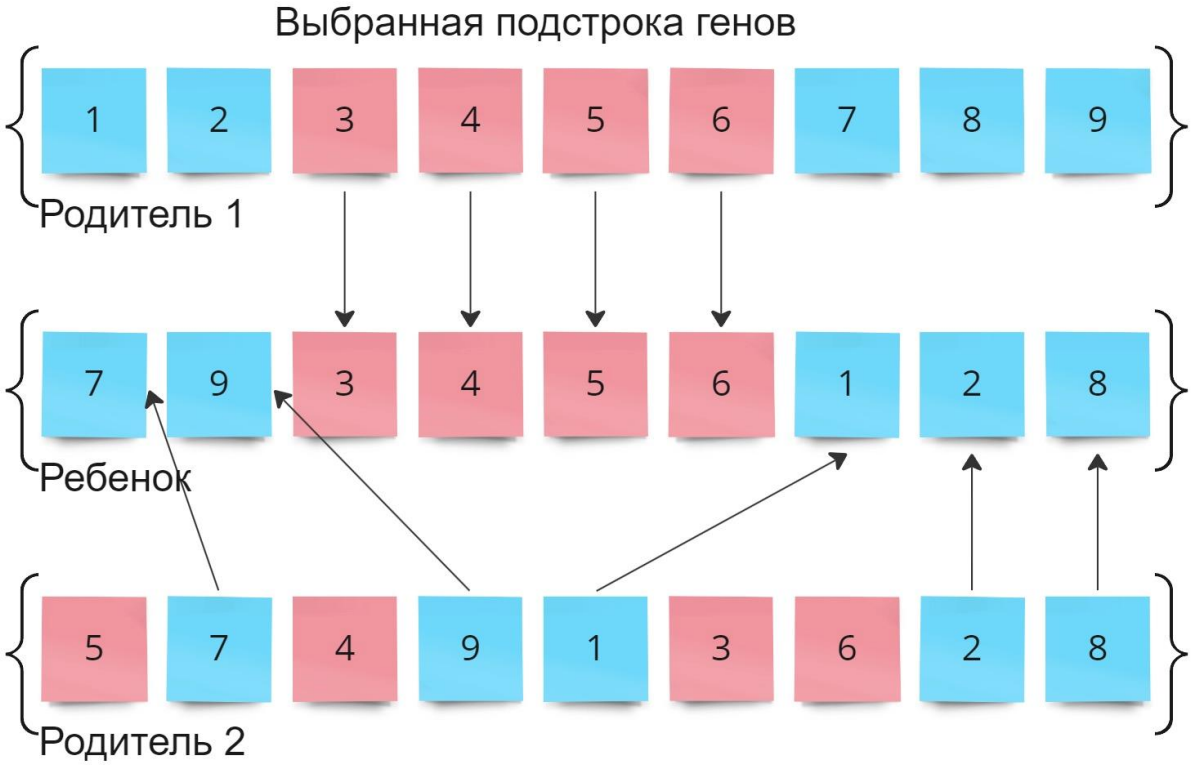


Рисунок 1.8 – Order crossover

1.5.2 Position-based crossover

Position-based crossover характеризуется следующим набором шагов:

1) Выбрать случайный набор генов родителя на случайных позициях.

2) Сформировать прото-ребенка копированием набора в соответствующие позиции.

3) Удалить гены, присутствующие в наборе у второго родителя. Результирующая последовательность генов содержит гены, которых не достает прото-ребенку.

4) Поставить гены на незанятые позиции прото-ребенка слева направо в соответствии с порядком в последовательности, полученной на предыдущем шаге для получения ребенка.

Схематично шаги изображены на рисунке 1.9.

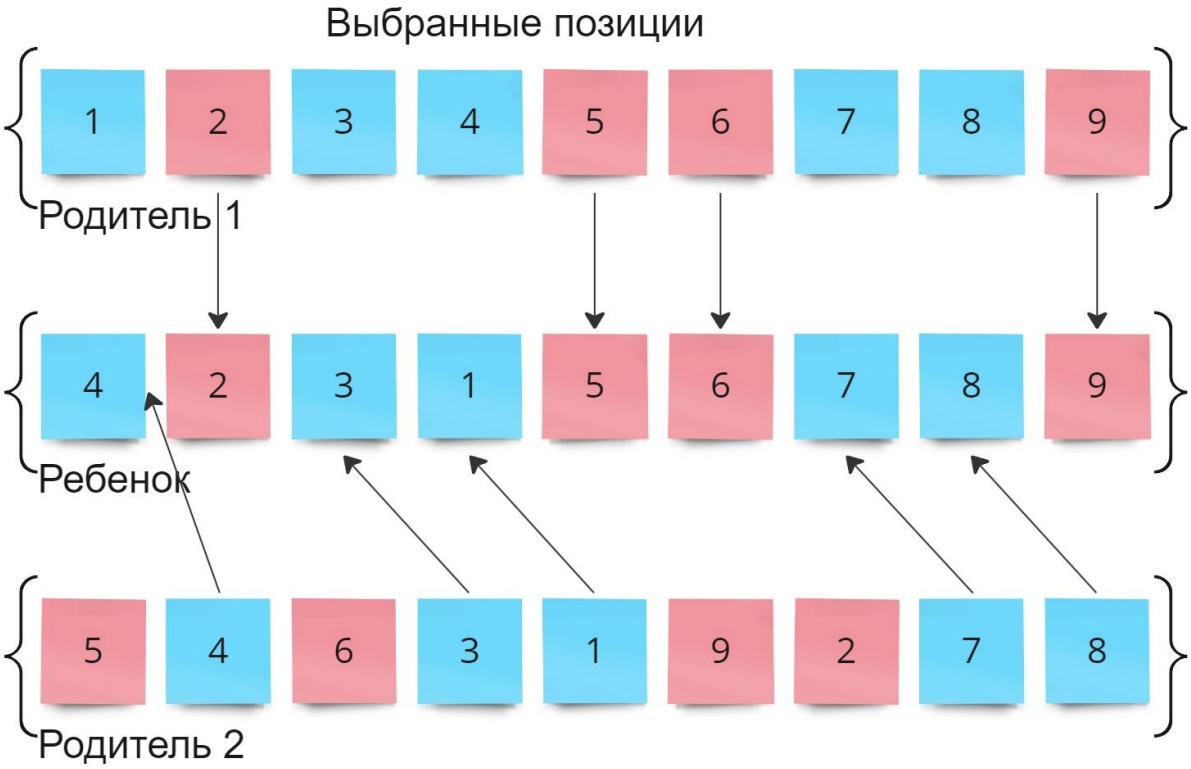


Рисунок 1.9 – Position-based crossover

1.5.3 Order-based crossover

Order crossover характеризуется следующим набором шагов:

1) Выбрать случайный набор генов первого родителя на случайных позициях.

2) Удалить гены, присутствующие в наборе с предыдущего шага у второго родителя.

3) Поставить гены выбранные у первого родителя на позиции удаленных генов второго родителя.

4) Поставить оставшиеся гены второго родителя на незанятые позиции прото-ребенка слева направо в соответствии с порядком в последовательности, полученной на предыдущем шаге для получения ребенка.

Схематично шаги изображены на рисунке 1.10.

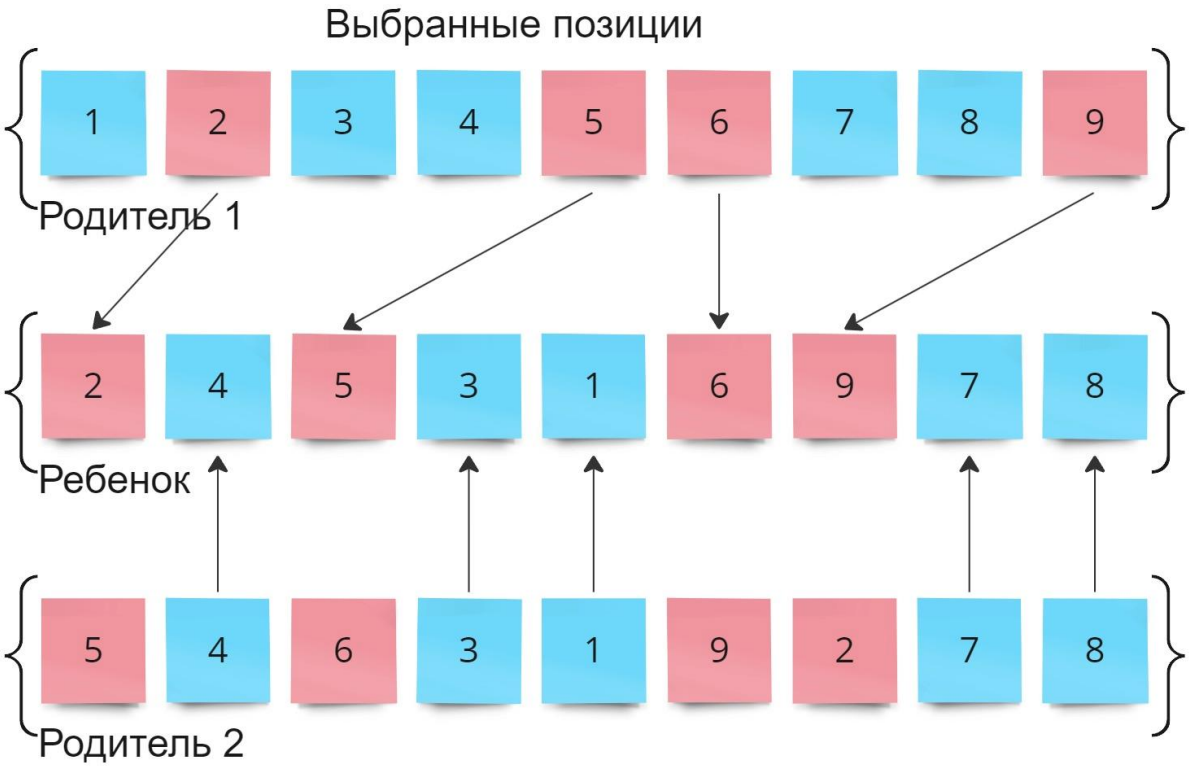


Рисунок 1.10 – Order-based crossover

1.5.4 CX crossover

CX crossover характеризуется следующим набором шагов:

1) Найти цикл, определяющий соответствующие позиции генов между родителями.

2) Скопировать гены из цикла в прото-ребенка на соответствующие первому родителю позиции.

3) Определить оставшиеся города у ребенка, удалив те города, которые уже есть в цикле у другого родителя.

4) Дополнить прото-ребенка оставшимися генами для получения ребенка.

Схематично шаги изображены на рисунке 1.11.

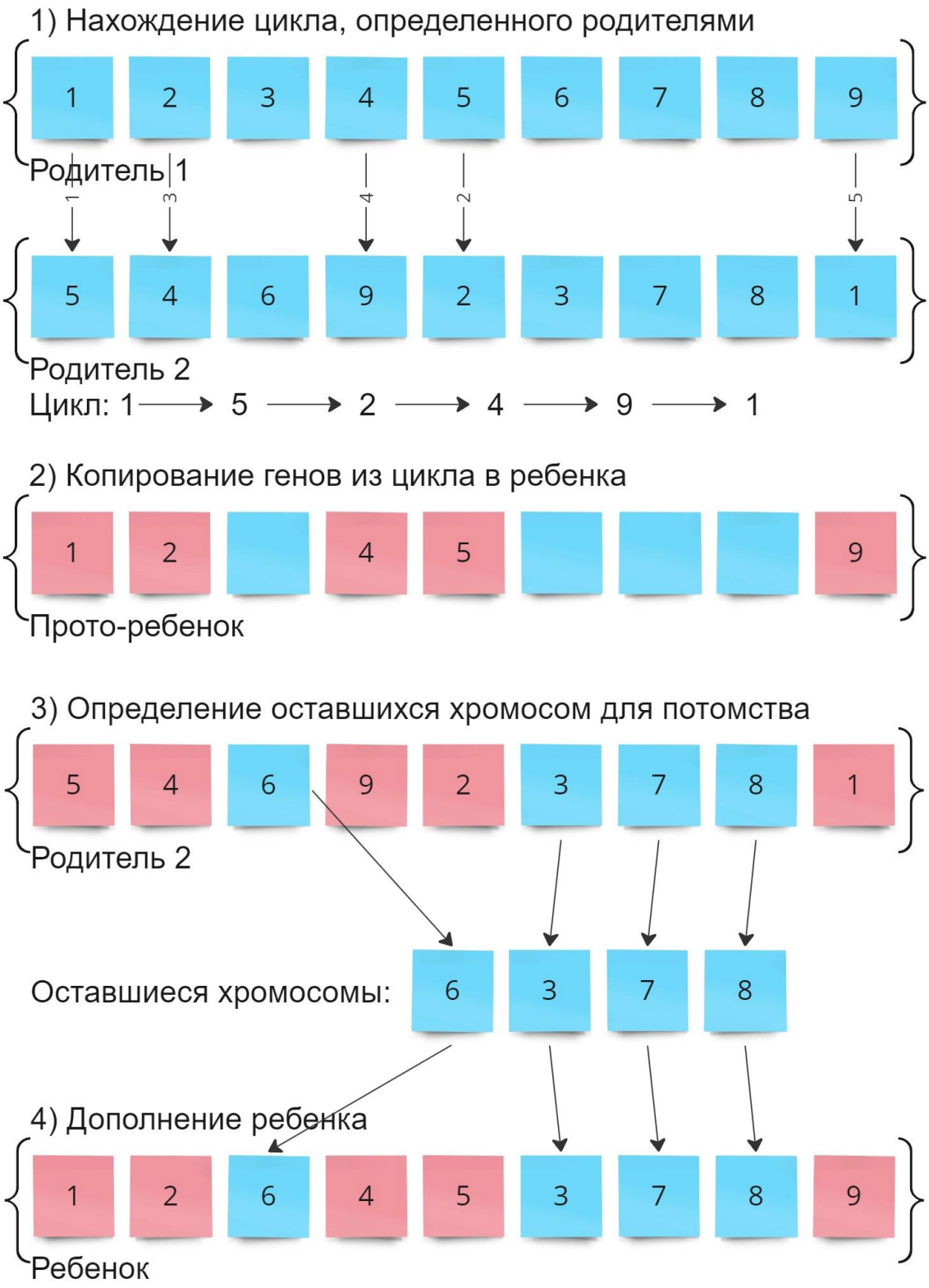


Рисунок 1.11 – CX crossover

1.5.5 Subtour exchange crossover

Subtour exchange crossover характеризуется следующим набором шагов:

1) Найти максимальный подпуть, содержащийся в обоих родителях.

2) Первого ребенка сформировать скопировав родителя 1 с заменой подпути на подпуть от родителя 2.

3) Второго ребенка сформировать скопировав родителя 2 с заменой подпути на подпуть от родителя 1.

Схематично шаги изображены на рисунке 1.12.

Рисунок 1.12 – Subtour exchange crossover

1.6 Мутация в ГА

Мутация является последним шагом в ГА. Она позволяет избежать застревания алгоритма в локальном оптимуме и вырождения генома. Существует множество вариантов мутации в зависимости от выбранной системы кодирования для конкретной задачи. Рассмотрим подробнее операции мутации, применимые к перестановочному кодированию, поскольку именно это кодирование используется в задаче поиска оптимального маршрута. Будем использовать общепринятые названия на английском, поскольку общепринятых устоявшихся названий в русскоязычной литературе не встречается.

1.6.1 Inversion mutation

Inversion mutation характеризуется следующим набором шагов:

1) Выбрать случайную подстроку генов.

2) Мутировать хромосому путем перестановки генов в ней.

Схематично шаги изображены на рисунке 1.13.

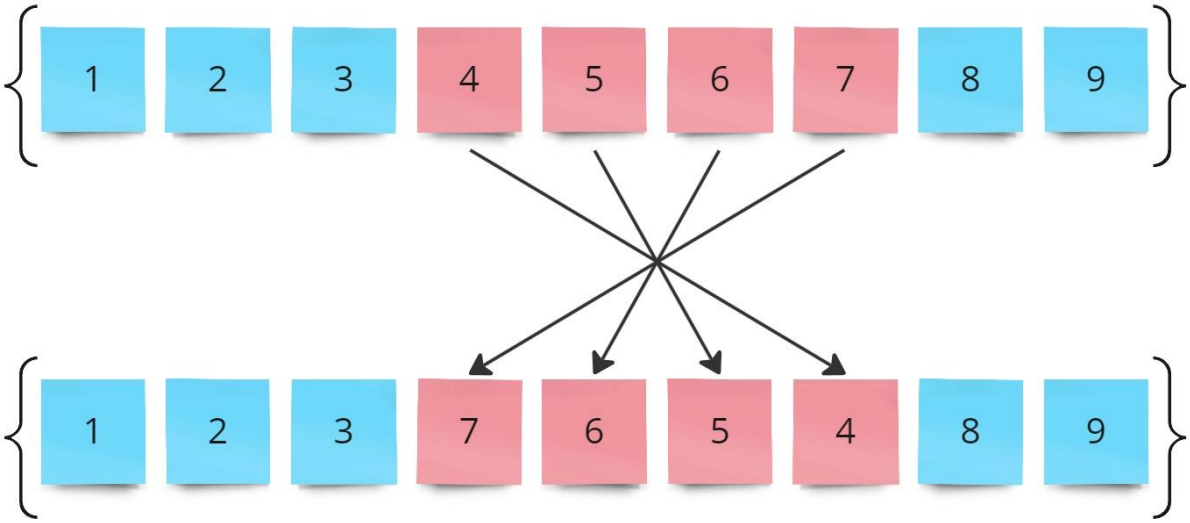


Рисунок 1.13 – Inversion mutation

1.6.2 Insertion mutation

Insertion mutation характеризуется следующим набором шагов:

1) Выбрать случайный ген хромосомы.

2) Мутировать хромосому путем перемещения выбранного гена на случайную позицию, сдвинув остальные гены соответствующе.

Схематично шаги изображены на рисунке 1.14.

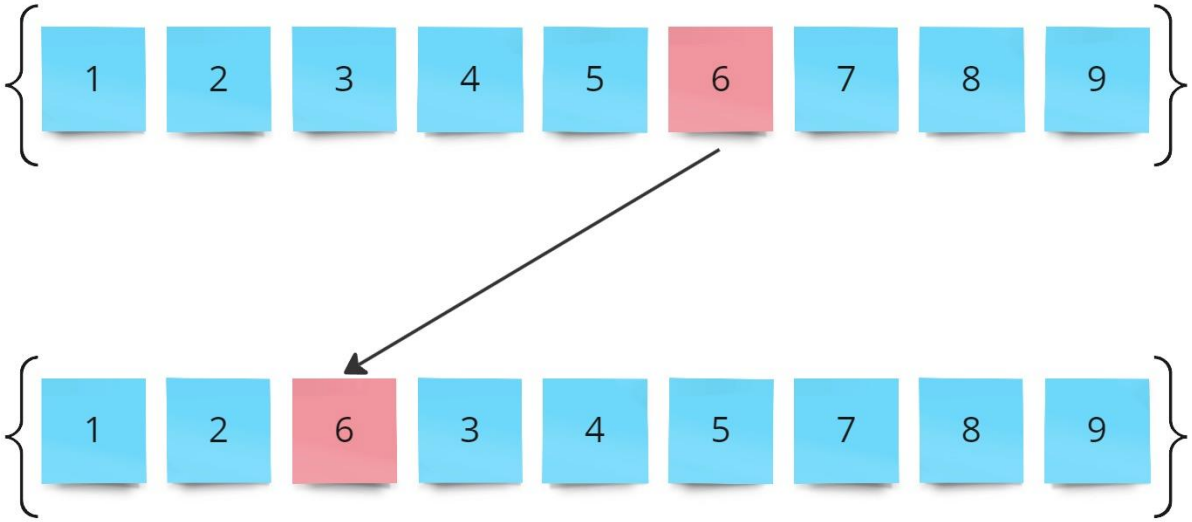


Рисунок 1.14 – Insertion mutation

1.6.3 Displacement mutation

Displacement mutation характеризуется следующим набором шагов:

1) Выбрать случайную подстроку генов.

2) Переместить гены на случайную позицию, сдвинув все остальные гены.

Схематично шаги изображены на рисунке 1.15.

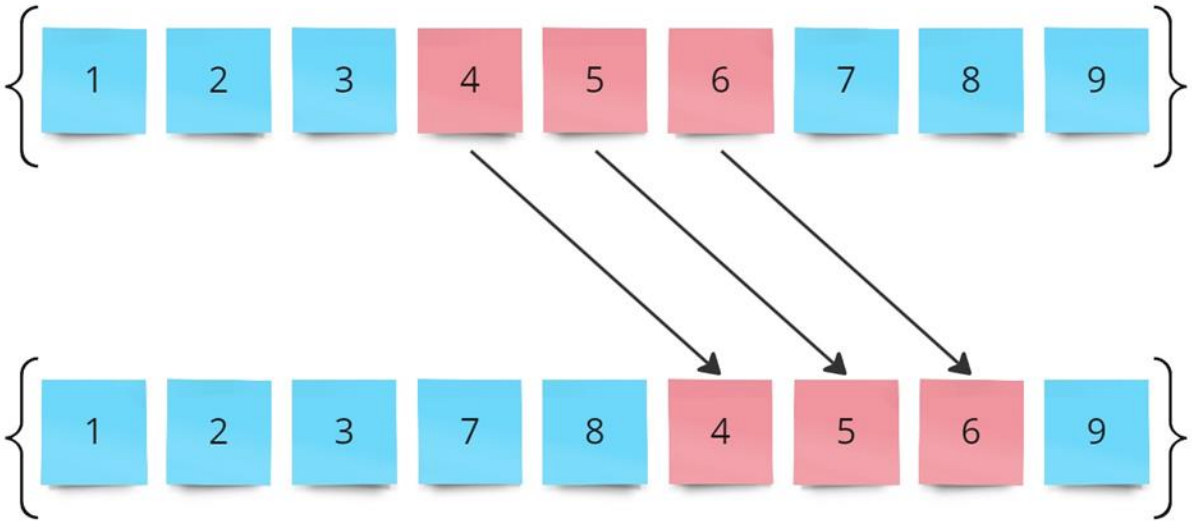


Рисунок 1.15 – Displacement mutation

1.6.4 Exchange mutation

Exchange mutation характеризуется следующим набором шагов:

1) Выбрать два случайных гена.

2) Мутировать хромосому, обменяв выбранные гены местами.

Схематично шаги изображены на рисунке 1.16.

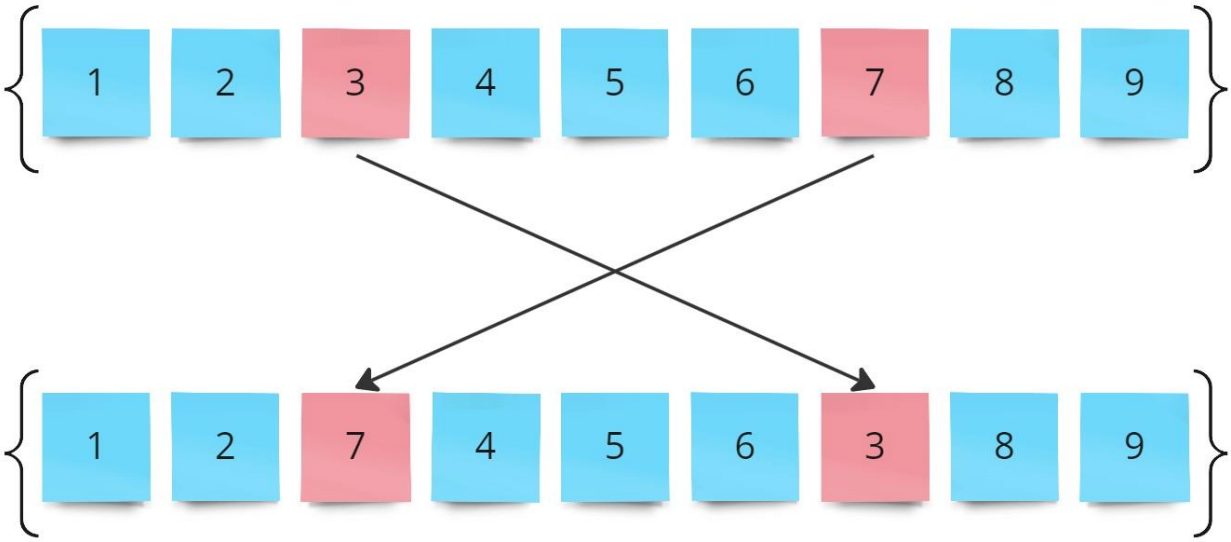


Рисунок 1.16 – Exchange mutation

1.6.5 Heuristic mutation

Heuristic mutation характеризуется следующим набором шагов:

1) Выбрать λ генов на случайных позициях.

2) Сгенерировать соседей путем всех возможных перестановок выбранных генов.

3) Сравнить фитнес-функции всех соседей и выбрать лучшего в качестве мутировавшей хромосомы.

Схематично шаги изображены на рисунке 1.17.

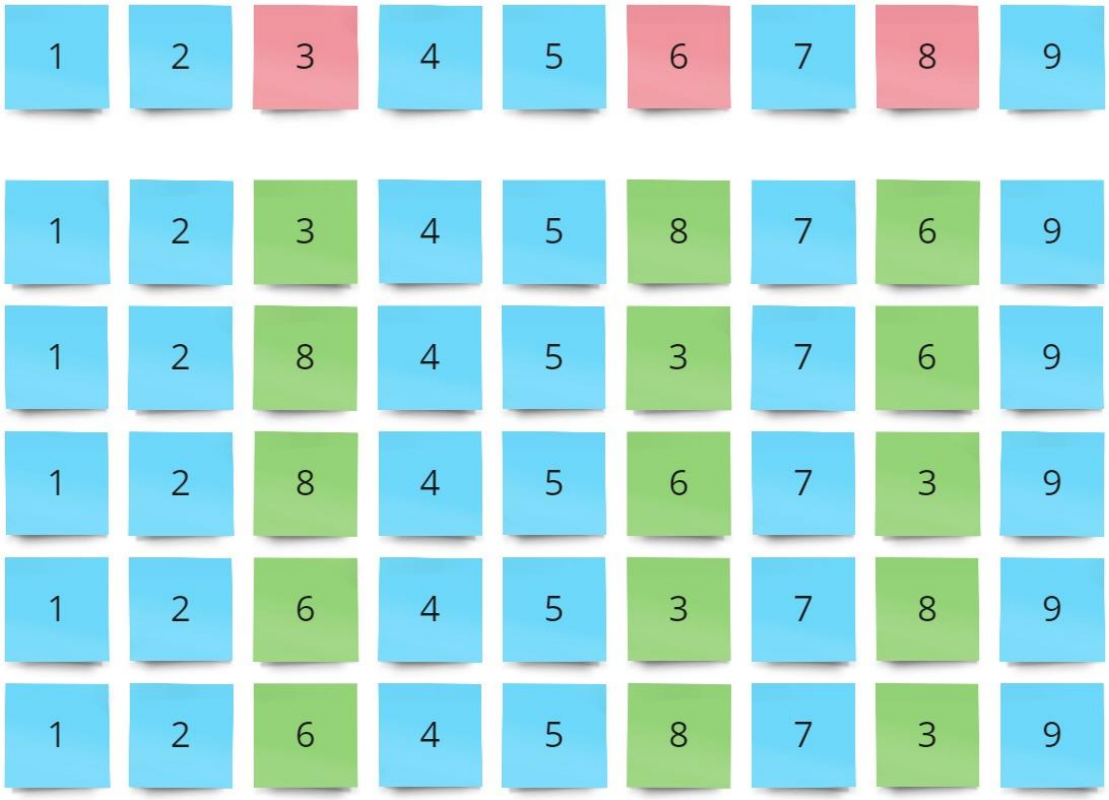


Рисунок 1.17 – Heuristic mutation

2 ОБЗОР ТЕХНОЛОГИЙ

2.1 Операционная система Android

Android – это операционная система, разработанная компанией Google для мобильных устройств, таких как смартфоны, планшеты, умные часы и телевизоры. Она была выпущена в 2008 году и быстро стала одной из самых популярных мобильных операционных систем в мире. ОС основана на ядре Linux и использует открытые стандарты и протоколы [9].

Одним из главных преимуществ Android является его открытость. Это означает, что разработчики могут создавать приложения и настраивать операционную систему под свои нужды. Это способствует более широкому выбору приложений для пользователей и повышает конкуренцию на рынке. Кроме того, Android имеет богатый набор инструментов для разработчиков, включая среду разработки Android Studio и открытые API для создания интеграций с другими приложениями и сервисами.

Вторым преимуществом Android является его экосистема приложений. Google Play, официальный магазин приложений для большинства смартфонов Android. Это означает, что пользователи могут легко найти и загрузить приложения для различных целей, от игр до продуктивности и социальных сетей. Что дает доступ к пользователям по всему земному шару. Это важно, если мобильное приложение подразумевает его развитие и окупаемость.

Третьим преимуществом Android является его адаптивность и масштабируемость. Android поддерживает широкий диапазон устройств, от недорогих смартфонов до мощных планшетов и флагманских смартфонов. Это означает, что разработчики могут создавать приложения для разных устройств, и пользователи могут выбирать устройство, соответствующее их потребностям и бюджету. Кроме того, Android имеет богатый набор функций и возможностей, которые могут быть настроены и масштабированы в соответствии с потребностями пользователя.

Одним из главных недостатков Android является фрагментация. Это означает, что разные устройства могут использовать разные версии операционной системы и иметь разные функции и возможности. Это может создавать эффект "разброса" среди пользователей, что затрудняет установку одних и тех же приложений и создание универсальных решений для всех устройств. Кроме того, фрагментация может привести к более медленным обновлениям системы и безопасности, поскольку различные производители могут задерживать или не выпускать обновления для своих устройств.

Вторым недостатком Android является проблема безопасности. Android как операционная система стала целью для многих кибератак, поскольку многие производители устройств не выпускают регулярные обновления безопасности. Кроме того, многие пользователи не осознают важность защиты своих устройств и могут устанавливать приложения из неизвестных источников, что

может привести к уязвимостям и нарушению конфиденциальности.

Третья достаточно существенная проблема – количество потенциально платежеспособных пользователей значительно ниже, по сравнению с ОС iOS.

2.2 Android studio

Android Studio – это интегрированная среда разработки (IDE) для создания приложений для мобильной операционной системы Android [10]. Это официальная IDE, поддерживаемая Google, и предоставляет разработчикам инструменты, необходимые для создания высококачественных приложений для мобильных устройств на базе Android. Главное окно интегрированной среды разработки представлено на рисунке 2.1.

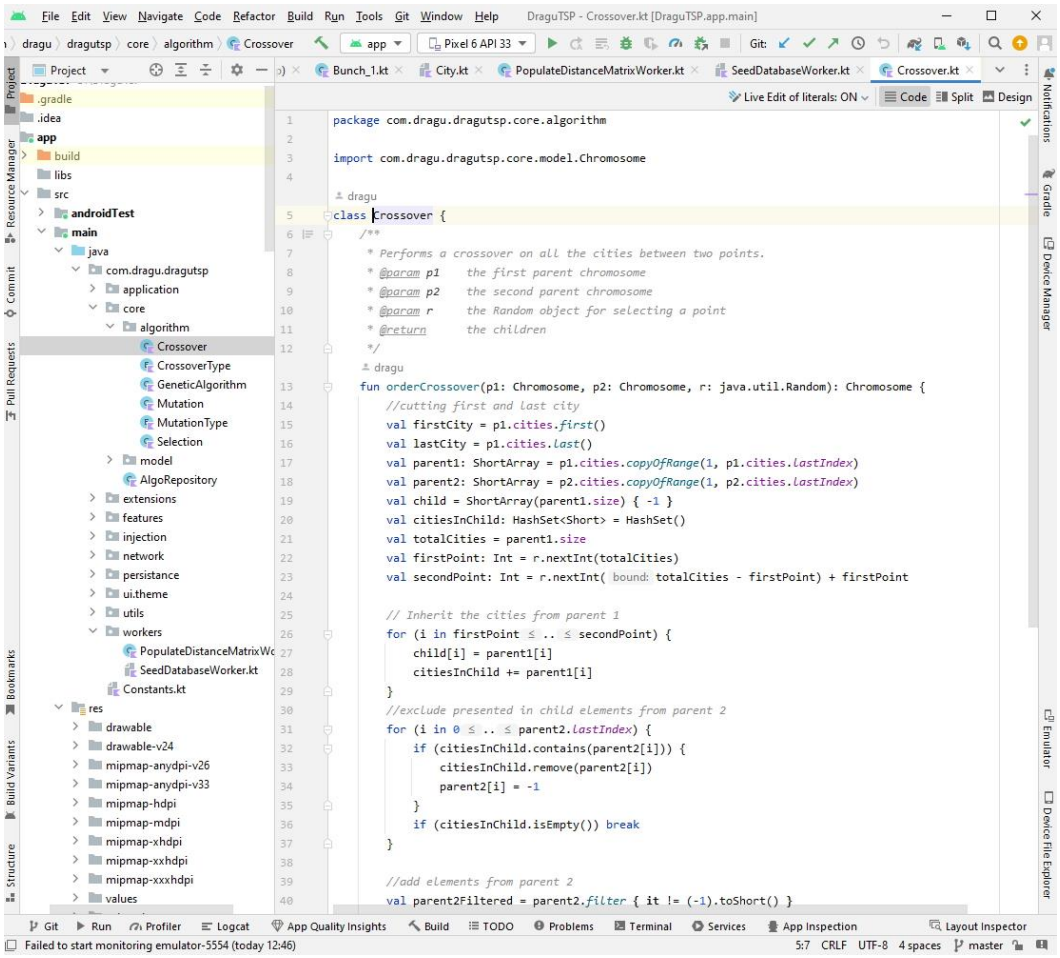


Рисунок 2.1– Главное окно IDE

Android Studio основана на IntelliJ IDEA, известной IDE для разработки на языках программирования Java и Kotlin. Это означает, что Android Studio предоставляет множество инструментов, которые существенно упрощают процесс разработки для разработчиков, знакомых с IntelliJ IDEA.

Основными функциями Android Studio являются редактор кода,

отладчик, графический редактор макетов и средства для сборки и тестирования приложений. Android Studio также поддерживает использование многих других инструментов и технологий, таких как Git для контроля версий, Gradle для сборки проектов и Android Virtual Device Manager (AVD Manager) для эмуляции устройств Android.

Одна из главных причин, по которой разработчики предпочитают использовать Android Studio, это потому, что оно обеспечивает полную интеграцию с платформой Android и предоставляет разработчикам множество средств и ресурсов для создания мобильных приложений, таких как мощные средства отладки, симуляторы и тестовые инструменты.

2.3 Язык программирования Kotlin

Kotlin – это статически типизированный язык программирования, который был разработан JetBrains и выпущен в 2011 году [11]. Kotlin является языком программирования, совместимым с Java, и может быть использован для написания приложений для платформы Java Virtual Machine (JVM), а также для Android и других платформ.

Kotlin был создан для улучшения производительности и эффективности программистов, сделав программирование более простым, чистым и безопасным. Он предлагает множество удобных функций, таких как нулевые ссылки, расширения функций, анонимные объекты и многое другое.

Одно из главных преимуществ Kotlin – это то, что он является языком программирования, который легко читается и понимается, даже для тех, кто никогда не работал с Java или другими языками программирования. Благодаря его интуитивно понятному синтаксису Kotlin позволяет разработчикам писать код более быстро и эффективно, что особенно важно в мобильной разработке.

Кроме того, Kotlin также предлагает множество инструментов и библиотек, которые делают разработку мобильных приложений еще более простой и удобной. Kotlin имеет мощную систему типов, которая обеспечивает безопасность и надежность кода, что особенно важно для мобильных приложений, работающих с чувствительными данными.

В целом, Kotlin является отличным выбором для разработки мобильных приложений из-за его легкочитаемости, безопасности и возможности использования для мультиплатформенной разработки. Благодаря растущей популярности Kotlin Multiplatform, его использование в мобильной разработке будет только расти, что позволит разработчикам создавать приложения для различных платформ быстрее и проще, используя один и тот же язык программирования.

2.4 Kotlin multiplatform mobile

Kotlin Multiplatform Mobile (KMM) – это технология, которая позволяет разработчикам создавать приложения для различных мобильных платформ,

используя Kotlin [12]. Она позволяет сократить время разработки и облегчить поддержку кода, поскольку разработчики могут использовать один и тот же код на нескольких платформах, включая iOS и Android.

KMM позволяет разработчикам писать общий код для бизнес-логики, моделей данных и других функций приложения, которые не зависят от конкретной платформы. Код, который зависит от платформы, такой как пользовательский интерфейс, может быть написан в специфических для каждой платформы модулях. Более того, в марте 2023 года Compose Multiplatform для системы iOS перешел в стадию Alpha, что означает возможность использовать один код и для UI. Такой подход позволяет разработчикам сократить время на написание кода, упростить его поддержку и сократить затраты на разработку и тестирование.

Одно из главных преимуществ KMM – это то, что он предлагает эффективное взаимодействие между общим кодом и специфическими для платформ модулями. Разработчики могут использовать общий код на всех платформах, однако при необходимости модифицировать его под специфические для каждой платформы требования, например, при вызове специфичных API, таких как Bluetooth или камера.

Кроме того, KMM обеспечивает высокую производительность и масштабируемость приложений, что является важным фактором при разработке мобильных приложений. Он также поддерживает интеграцию с различными инструментами и библиотеками, что позволяет разработчикам использовать свои любимые инструменты при разработке мобильных приложений.

Наконец, KMM является открытым исходным кодом, что позволяет разработчикам получать доступ к исходному коду и вносить свои вклады в развитие технологии. Это также означает, что KMM является бесплатным для использования и не требует дополнительных затрат на покупку или лицензирование.

В целом, Kotlin Multiplatform Mobile представляет собой мощную и гибкую технологию, которая позволяет разработчикам создавать мобильные приложения для различных платформ, используя Kotlin.

2.5 Пользовательский интерфейс Compose

Compose UI – это новый декларативный подход к созданию пользовательского интерфейса в Android, основанный на языке программирования Kotlin. Он предоставляет простой и интуитивно понятный способ создания пользовательского интерфейса, используя компоненты, называемые Composables.

Composables – это функции Kotlin, которые описывают пользовательский интерфейс как дерево компонентов, каждый из которых может содержать другие компоненты. Компоненты Compose можно реализовать как существующие элементы пользовательского интерфейса, такие как кнопки, поля ввода и т.д., или создать свои собственные компоненты, которые могут быть

повторно использованы в других частях приложения.

Compose UI упрощает процесс создания пользовательского интерфейса и позволяет разработчикам более быстро и легко создавать красивые и функциональные пользовательские интерфейсы. Он также предоставляет функциональность для управления состоянием приложения и реактивного программирования.

Compose Multiplatform UI – это расширение Compose UI, которое позволяет разработчикам создавать многоплатформенные приложения с использованием Compose. Это означает, что разработчики могут использовать Compose для создания пользовательского интерфейса на различных платформах, включая Android, iOS [13].

Compose Multiplatform UI использует ту же модель программирования, что и Compose UI, что позволяет разработчикам переиспользовать код для создания пользовательского интерфейса на различных платформах. Он также предоставляет поддержку для различных элементов пользовательского интерфейса, включая текст, изображения, списки, формы и т.д.

В целом, Compose UI и Compose Multiplatform UI представляют собой мощные инструменты для разработки пользовательского интерфейса в Android и многоплатформенных приложений, которые упрощают процесс создания красивых и функциональных приложений.

2.6 Сетевые запросы

Retrofit и OkHTTP – это две библиотеки для выполнения сетевых запросов в Android-приложениях, написанных на языке Java или Kotlin. Retrofit предоставляет удобный API для описания REST-сервисов и их методов, которые могут быть вызваны из приложения для получения данных с сервера. OkHTTP, в свою очередь, предоставляет мощный и гибкий механизм для выполнения HTTP-запросов с различными параметрами и настройками.

В первой версии нашего мобильного приложения мы будем использовать Retrofit и OkHTTP для выполнения сетевых запросов. С их помощью мы сможем легко и эффективно получать данные с сервера, включая текстовые, бинарные и JSON-данные. Мы сможем использовать Retrofit для создания и настройки интерфейсов API, а затем вызывать методы API с помощью OkHTTP для получения данных. Retrofit и OkHTTP предоставляют обширную документацию и множество примеров использования, что делает их очень популярными инструментами для выполнения сетевых запросов в Android-приложениях.

Во второй стадии, когда мы расширим наше приложение на платформу IOS и Kotlin Multiplatform Mobile, мы будем использовать Ktor вместо Retrofit и OkHTTP. Ktor – это легковесный и многофункциональный веб-фреймворк, написанный на языке Kotlin, который предоставляет инструменты для выполнения сетевых запросов, разработки REST API и обработки HTTP-запросов. Он обладает высокой производительностью и хорошей масштабируемостью,

мощными средствами для обработки ошибок, кеширования данных и поддержки множественных протоколов. Кроме того, использование Ktor в Kotlin Multiplatform Mobile позволит нам создавать общий код для выполнения сетевых запросов на различных платформах, что значительно упростит и ускорит процесс разработки.

Таким образом, в первой версии нашего мобильного приложения мы будем использовать Retrofit и OkHTTP для выполнения сетевых запросов, а во второй стадии, при расширении на платформу IOS и Kotlin Multiplatform Mobile, мы перейдем на Ktor, который обеспечит высокую производительность и надежность выполнения сетевых запросов на всех целевых платформах, что будет способствовать более быстрой и эффективной работе нашего мобильного приложения.

2.7 Локальная база данных

В первой версии нашего мобильного приложения для работы с локальной базой данных мы будем использовать библиотеку ROOM. ROOM – это библиотека, предоставляемая Google, которая позволяет использовать абстракцию базы данных SQLite для более простой и эффективной работы с базой данных в приложении. ROOM предоставляет аннотации для определения сущностей базы данных, DAO-интерфейсов для выполнения запросов к базе данных и других элементов, таких как миграции, которые позволяют изменять схему базы данных без потери данных.

Во второй версии нашего мобильного приложения, при расширении на платформу IOS и Kotlin Multiplatform Mobile, мы будем использовать библиотеку sqldelight. SqlDelight – это библиотека для работы с базой данных, которая предоставляет общую схему базы данных, которую можно использовать на разных платформах, таких как Android, iOS и серверные приложения. SqlDelight генерирует типобезопасный API для доступа к базе данных, основанный на SQL-запросах и позволяет писать SQL-запросы в статически типизированном Kotlin-коде, что уменьшает вероятность ошибок при работе с базой данных.

Использование библиотек ROOM и SqlDelight обеспечивает эффективную и удобную работу с локальной базой данных в нашем мобильном приложении, как на Android, так и на iOS в будущем.

2.8 Платные подписки

Финальная стадия первого шага подразумевает начало получения прибыли. Для этого мы будем использовать систему подписок, предоставляемую Play Market на Android-платформе. Для простоты разработки она будет содержать всего один план, который разблокирует возможность импорта пользовательнского набора данных, ограниченным 150 точками. Это обусловлено тем, что для подсчета матриц дистанции и времени каждая очередная точка требует

платных сетевых запросов, соответственно оставлять этот функционал бесплатным при запуске нельзя.

Система подписок от Play Market – это механизм, который позволяет разработчикам мобильных приложений предлагать подписки для своих пользователей на определенные услуги или функциональность в приложении. Подписки могут быть как ежемесячными, так и годовыми, и предоставлять пользователю доступ к дополнительным функциям, контенту и ресурсам, которые недоступны для пользователей, не подписавшихся на эти услуги.

Основные преимущества системы подписок от Play Market включают:

1) Удобство для пользователей – подписки могут быть оформлены прямо в приложении и не требуют от пользователя дополнительных действий, например перехода на веб-сайт разработчика.

2) Гибкость и настраиваемость – разработчики могут настраивать различные уровни подписок с различными ценами и продолжительностью подписки.

3) Расширенная аналитика – система от Play Market предоставляет разработчикам много данных и аналитики, связанных с поведением пользователей и процессом подписки, что позволяет улучшать продукт и повышать доходность.

Вторая версия нашего мобильного приложения будет включать аналогичный функционал, предоставляемый App Store на iOS-платформе. Мы также будем использовать систему подписок.

Предоставление платных подписок нашим пользователям позволит нам зарабатывать деньги на нашем приложении, что будет способствовать развитию проекта и его поддержке в долгосрочной перспективе. Мы будем собирать отзывы и мнения пользователей о нашем приложении и постоянно работать над его улучшением, чтобы удовлетворить потребности наших пользователей и привлечь новых.

3 АРХИТЕКТУРА ПРИЛОЖЕНИЯ

3.1 Общая архитектура

MVVM (Model-View-ViewModel) – это популярная архитектура для разработки мобильных приложений на платформе Android. (рисунок 3.1)

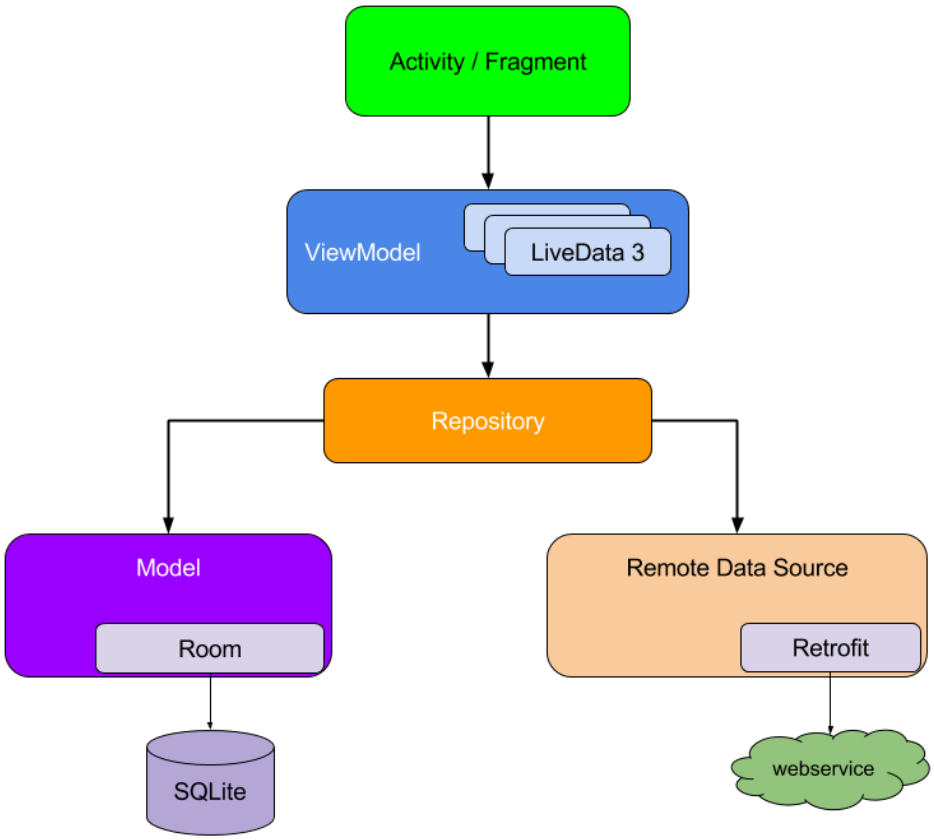


Рисунок 3.1 – Архитектура MVVM

Основные компоненты MVVM:

1) Модель (Model) – отвечает за обработку данных и бизнес-логику приложения. Это может быть класс, который выполняет запросы к базе данных, API или любому другому источнику данных.

2) Представление (View) – это то, что пользователь видит на экране. Оно может быть представлено как Activity, Fragment или View.

3) ViewModel – это прослойка между Представлением и Моделью. Она принимает запросы на данные от Представления, обрабатывает их и предоставляет необходимые данные в Представление. ViewModel также может быть ответственным за управление состоянием Представления.

Главными преимуществами такой архитектуры можно назвать:

1) Улучшенная отдельность компонентов – каждый компонент имеет

свою роль и не знает о реализации других компонентов.

2) Улучшенное тестирование – MVVM позволяет легко тестировать каждый компонент приложения отдельно.

3) Улучшенная поддержка переиспользования кода – благодаря тому, что каждый компонент имеет свою роль и не знает о реализации других компонентов, можно легко использовать компоненты в других приложениях.

4) Улучшенная поддержка многопоточности – ViewModel может использовать асинхронные операции для получения данных и обновления состояния Представления.

Как и у любой архитектуры, у MVVM также имеются недостатки:

1) Дополнительная сложность – MVVM может быть сложнее, чем другие архитектуры, из-за необходимости создания ViewModel для каждого Представления.

2) Раздутое количество классов – использование MVVM может привести к большому количеству классов, что может усложнить код и процесс разработки.

3) Высокий уровень абстракции – MVVM может иметь высокий уровень абстракции, что может усложнить понимание происходящего в приложении для начинающих разработчиков.

3.2 Матрицы расстояний и времени

Для того, чтобы приложение работало, необходимо собрать матрицу расстояний и времени между каждой из 130 точек, чтобы приложение могло оптимизировать маршрут на основе этих данных. Это 17000 пар значений, вручную задача достаточно трудоемка. Стоит заметить, что географическое расстояние по прямой линии нам не подходит, так как фактическое расстояние по дороге может отличаться в разы. Кроме того, в дальнейшем планируется добавление возможности пользовательского импорта со своим набором данных, тогда каждому пользователю необходимо будет хранить свою уникальную матрицу.

Имеются различные платные API для решения этой задачи, после краткого анализа самым оптимальным выглядит The Bing Maps Distance Matrix API [14]. Однако стоит перечислить альтернативы:

Google Maps Distance Matrix API: это, вероятно, самое популярное решение для определения расстояний и времени между точками на карте. Оба API предоставляют данные о маршруте и времени в пути между несколькими точками. Но Google API является более дорогим и имеет более ограниченные бесплатные возможности, в то время как Bing API предоставляет более выгодные тарифные планы, которые позволяют использовать API более широко.

OpenStreetMap Distance Matrix API: это бесплатное решение с открытым исходным кодом, которое также предоставляет данные о расстоянии и времени между точками на карте. Но он имеет ограниченные возможности и не так точен как Bing API. Кроме того, для использования OpenStreetMap Distance

Matrix API необходимо быть знакомым с программированием на языке Python.

MapQuest Distance Matrix API: это еще одно решение для определения расстояний и времени между точками на карте. Он имеет функциональность, аналогичную Bing API, но чуть более дорогой и менее популярный среди разработчиков.

В целом, выбор между различными API зависит от целей и бюджета. Если стоимость играет важную роль, то Bing API является хорошим выбором, если же важнее точность и функциональность, то можно рассмотреть Google Maps Distance Matrix API или другие альтернативы. Мы остановимся на Bing API, из-за большого годового бесплатного лимита до выхода в продакшн. Это 125 000 транзакций, или 600 000 значений отправление-прибытие.

3.3 Модель данных городов

После того, как сетевые запросы написаны и API освоено, нужно эти данные как-то сохранить, дабы не делать 4 000 транзакций при каждом старте приложения. На помощь приходит локальная база данных ROOM.

Стоит заметить, что SQLite, поверх которого написан ROOM, не позволяет сохранять массивы в одну ячейку. Чтобы обойти это ограничение, воспользуемся конвертером Moshi, который при чтении будет конвертировать строку вида [0.0, 12.5, 18.6] в читаемый список чисел с плавающей точкой, а при записи производить обратную конвертацию. Каждое значение представляет расстояние в километрах от этого города до всех остальных городов, представленных в базе. Аналогичным образом поступим и с загруженными значениями времени. Они будут отображать время в минутах от этого города до всех остальных городов в базе. Модель данных городов представлена на рисунке 3.2

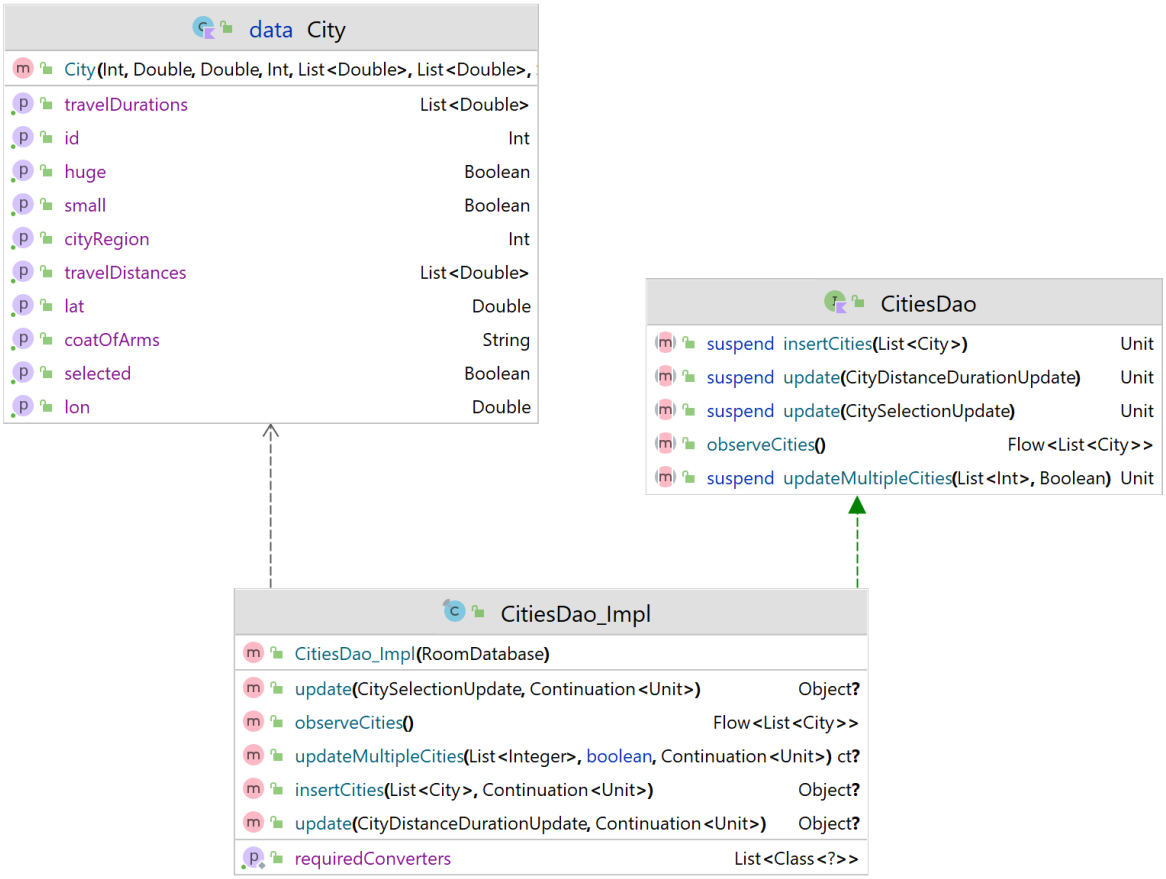


Рисунок 3.2 – Модель данных городов

Ключевые поля в данной модели:
1) id – идентификатор города;
2) lat – широта города;
3) lon – долгота города;
4) travelDistances – список дистанций до других городов в базе;
5) travelDurations – список времени до других городов в базе;
6) selected – выбран ли город как часть маршрута;
7) coatOfArms – ссылка на герб города;
8) cityRegion – регион, к которому относится данный город.

Пользовательский интерфейс выбора городов представлен на рисунке 3.3.

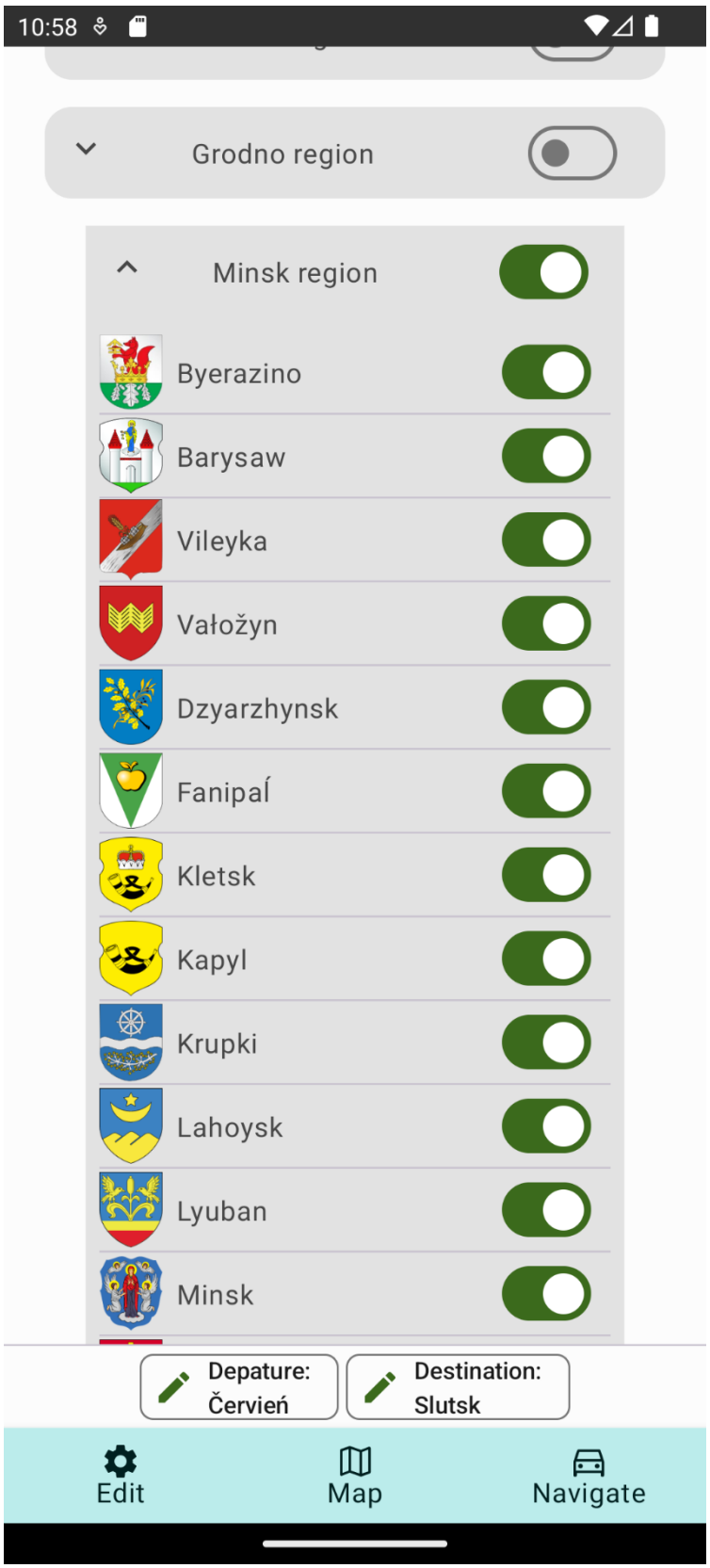


Рисунок 3.3 – Выбор городов

3.4 Загрузка гербов

Хранение изображений в android – не самая тривиальная задача в разрезе нашего приложения. Теоретически можно их загрузить в ресурсы и поставлять вместе с приложением. Но тогда добавление пользовательских точек со своими изображениями становится проблемой. Импорт и экспорт также из тривиальной задачи становятся проблемой. Поскольку нужно разрабатывать собственный формат, в который запаковывать/распаковывать все изображения каждый раз. Вес файла также может быть достаточно большим.

Однако если хранить ссылку на изображения, данных проблем можно будет избежать в будущем. Также можно будет сделать возможность импорта из Excel, что является наиболее простым вариантом для огромного количества пользователей по сравнению с gpkg, shp форматами, в которые можно встроить изображения, и с json форматом, который гипотетически подходит, но пользователя будет отпугивать.

Непосредственную загрузку изображений по ссылке делегируем библиотеке Coil. Так как она создает очень простую абстракцию, а также берет кеширование на себя. Для загрузки гербов на карте будем асинхронно использовать механизмы, поставляемые вместе с google maps API.

3.5 Карта

Google Maps SDK для Android и iOS предоставляет программный интерфейс приложения (API), который позволяет интегрировать картографические данные Google Maps непосредственно в свое мобильное приложение. Этот SDK обеспечивает разработчикам доступ к функциональности карт Google, такой как отображение карты, маршрутизация и поиск местоположений [15].

Google Maps SDK позволяет использовать множество функций, таких как отображение карты, маршрутизация, поиск местоположений, интеграция со службами Google. Поскольку гугл известен своими достаточно дорогими тарифами при большом числе пользователей, стоит с большой опаской очень сильно рассчитывать на его инфраструктуру в своих приложениях. По этой причине воспользуемся только самой необходимой вещью – отображением карты.

Чтобы начать использовать Google Maps SDK, нужно зарегистрировать свое приложение в Google Console, получить API-ключ и настроить SDK в своем проекте. После этого мы можем использовать API для отображения карт Google в своем приложении. Пользовательский интерфейс карты представлен на рисунке 3.4.

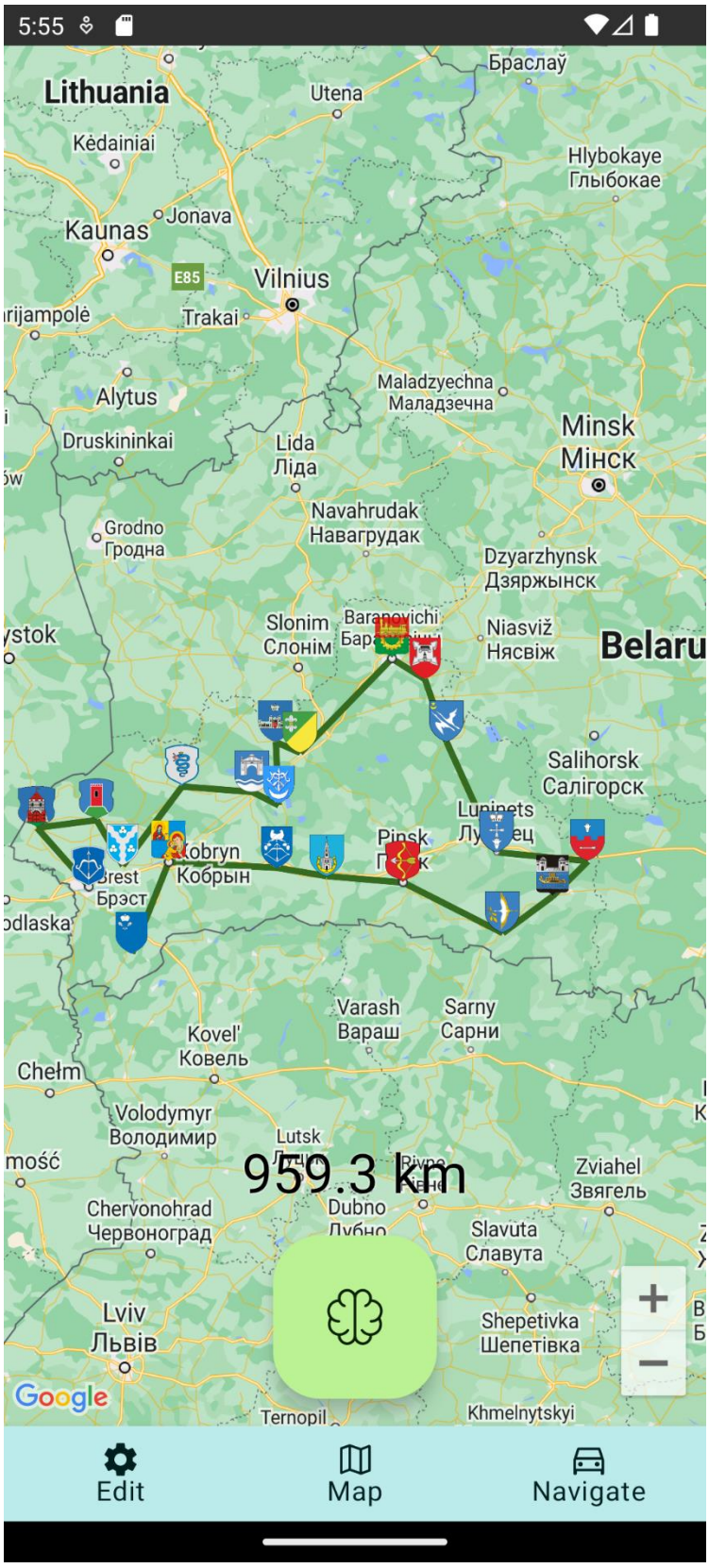


Рисунок 3.4 – Карта

3.6 Реализация генетического алгоритма

Для реализации генетического алгоритма будем использовать список ранее описанных моделей городов при подсчете фитнесс-функции. Чтобы алгоритм работал достаточно быстро и не требовал много оперативной памяти, используем структуру ShortArray для каждой хромосомы. Ограничение в kotlin для short – 32767. Учитывая стоимость API, понимаем, что этого числа более чем достаточно и пользовательская матрица расстояний не будет содержать больше элементов.

Диаграмма классов представлена на рисунке 3.5.

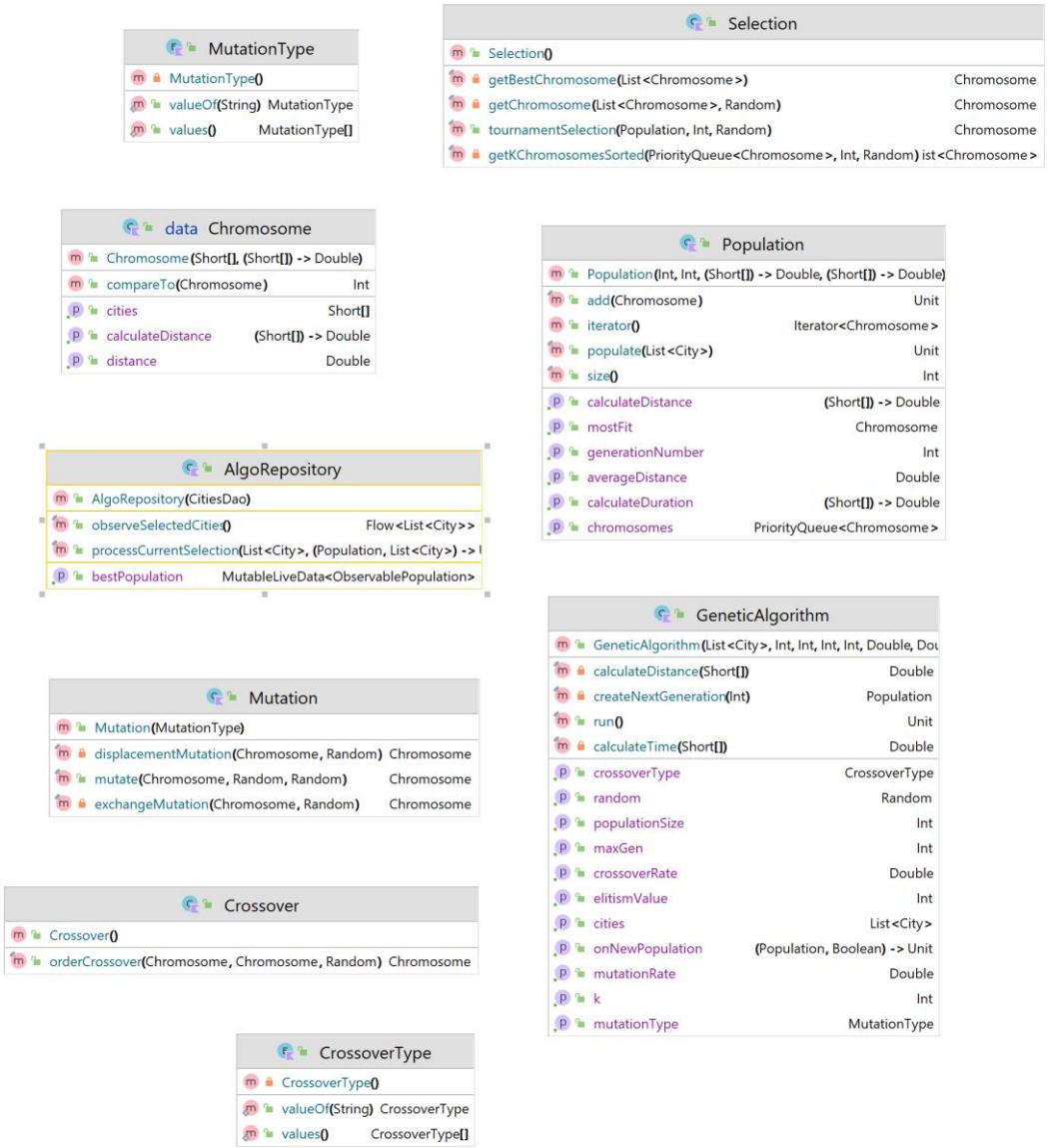


Рисунок 3.5 – Диаграмма классов алгоритма

3.7 Модель данных маршрутов

После того, как оптимальный маршрут найдет, его необходимо сохранить, чтобы направить пользователя в навигатор. Также его стоит сохранить, чтобы в дальнейшем можно было экспортировать и импортировать маршруты. Это добавит привлекательности для приложения, а также сэкономит какую-то часть бюджета, экономя нам платные вызовы стороннего апи. Модель данных маршрутов представлена на рисунке 3.6.

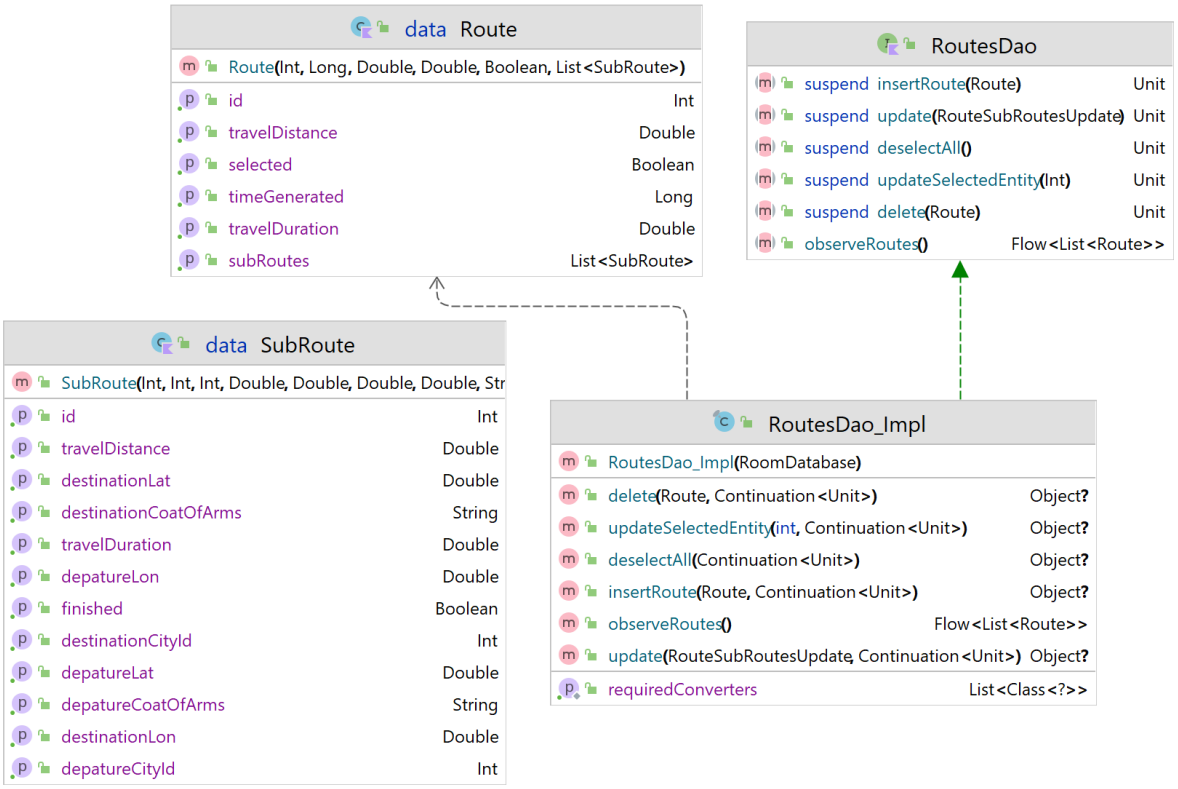


Рисунок 3.6 – Модель данных маршрута

Как было замечено ранее, SQLight не позволяет сохранять массивы в одну ячейку. Поступаем аналогично модели данных городов. При помощи Moshi будем конвертировать каждый подмаршрут в json-строку и хранить её в базе. А при чтении будем производить обратную конвертацию.

Поля для маршрутов:

1) id – идентификатор маршрута;
2) travelDistance – расстояние незаконченных подмаршрутов;
3) travelDuration – время незаконченных подмаршрутов;
4) timeGenerated – время создания маршрута;
5) selected – единственный выбранный маршрут;
6) subRoutes – список подмаршрутов.

Поля для подмаршрутов:
1) id – идентификатор подмаршрута
2) travelDistance – расстояние этого подмаршрута;
3) travelDuration – время этого подмаршрута;
4) departureLat – широта начальной точки;
5) departuteLon – долгота начальной точки;
6) destinationLat – широта конечной точки;
7) destinationLon – долгота конечной точки;
8) departureCoatOfArms – ссылка на герб начальной точки;
9) destinationCoatOfArms – ссылка на герб конечной точки;
10) departureCityId – ссылка на модель города отправления;
11) destinationCityId – ссылка на модель города назначения;
12) finished – был ли нажат данный подмаршрут.

Как видим, маршруты и подмаршруты содержат всю необходимую информацию и не опираются на матрицу расстояний. Исходя из этого можно сделать вывод, что в смысле затрат импорт и экспорт маршрутов нисколько для нас не стоят (в отличии от генерации уникальной пользовательской матрицы). Таким образом можно реализовать две версии подписки, «дешевой» плюс версии будет доступен ограниченный кастомный импорт, скажем, до 150 городов и открыта возможность импорта маршрутов. Для «дорогой» про версии кастомный импорт можно расширить до 250 городов, а также добавить возможность экспорта маршрутов. Тогда один «дорогой» для нас пользователь сможет генерировать маршруты по своей большой и дорогой на стадии составления матрицы, и сбрасывать маршруты пользователям с «дешевой» версией.

Это позволит лучше монетезировать приложения, поскольку самая дорогая для нас его часть – составить кастомную матрицу. Независимость модели маршрутов от матрицы – ключ к лучшей монетизации.

Пользовательский интерфейс маршрутов представлен на рисунке 3.7.

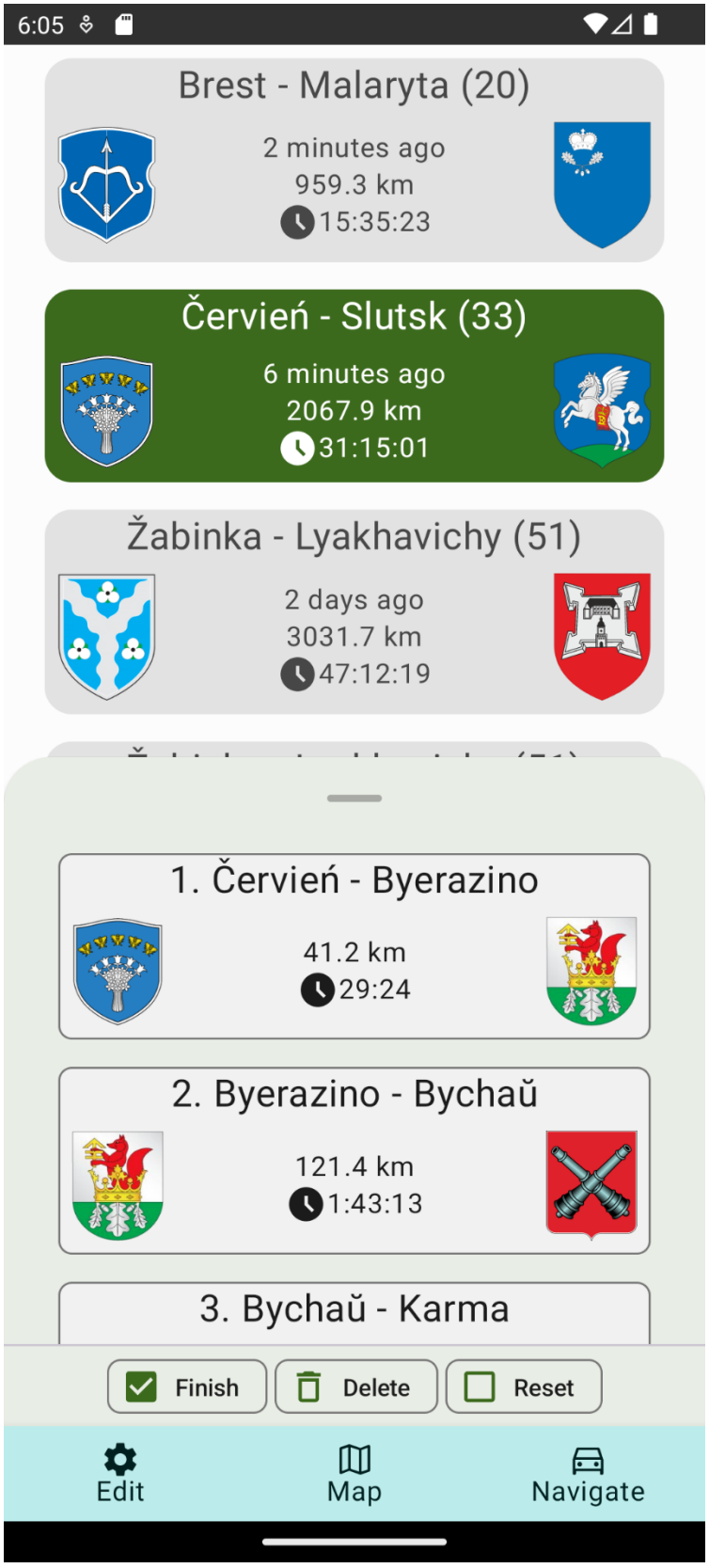


Рисунок 3.7 – Выбор маршрутов

3.8 Навигатор

Последний шаг, который привносит ценность пользователю – открытие навигатора с каждым построенным подмаршрутом. Без данного шага можно задать логичный вопрос – не проще ли сделать десктопную/веб версию приложения? Этот шаг снимает подобный вопрос.

На начальном этапе реализуем открытие Яндекс карт. В случае неоткрытия – google карт. В случае и их неоткрытия – будем открывать браузерную ссылку на google карты.

С развитием приложения эта логика должна быть вынесена в настройки с добавлением большего числа навигаторов, которые можно открыть с двумя точками на карте, например Waze или любые другие.

3.9 Дальнейшее развитие

В случае успешного запуска на платформе android архитектура будет в значительной мере переработана с помощью технологии kotlin multiplatform mobile, где как минимум логику запросов, алгоритмическую логику и всю базу данных можно вывести в общий код для android и iOS. Как максимум находящаяся в стадии alpha compose multiplatform позволит добиться качественного рендеринга на платформе iOS. Именно рендеринг и UI является краеугольным камнем других популярных мультиплатформенных фреймворков – Flutter и React Native. Значительное преимущество kotlin multiplatform перед ними в том, что даже в случае раздельного UI для двух этих платформ разработчику не нужно учить третий язык программирования, что существенно снижает затраты на разработку и поддержку. Переход же с текущей версии проекта на kotlin multiplatform должен быть достаточно легким, так как весь код уже написан на kotlin.

4 ТЕХНИКО–ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ РАЗРАБОТКИ МОБИЛЬНОГО ПРИЛОЖЕНИЯ ДЛЯ РЕШЕНИЯ ЗАДАЧИ ПОИСКА ОПТИМАЛЬНОГО МАРШРУТА МЕЖДУ ГОРОДАМИ

4.1 Краткая характеристика проекта

Цель дипломного проекта – проектирование и реализация мобильного приложения на базе платформы android, подтверждающее гипотетическую возможность решения задачи поиска оптимального маршрута для достаточно большого количества реальных городов. Планируется последующее привлечение инвестиций с последующим расширением на платформу iOS, расширением функционала с локального рынка РБ на мировой рынок.

Данное приложение будет полезно курьерским доставкам, грузоперевозчикам, путешественникам, и любым другим людям, регулярно посещающим достаточно большое количество точек на карте при помощи своего автомобиля.

Внутренними алгоритмами приложение может находить оптимальную очередность посещения любых городов, что будет экономить топливо и время.

Кроме того, побочным эффектом использования приложения достаточно большим количеством людей станет уменьшение выбросов CO2 в атмосферу, к чему должен стремиться современный мир.

Расчеты выполнены на основе методического пособия [16].

В этом разделе рассмотрим вопросы, связанные с технико–экономическим обоснованием проекта. Разработка приложения относится к первой группе сложности. Имеет простой интеллектуальный языковой интерфейс с пользователем, использует внутренние механизмы платформ android и iOS. Обладает дополнительными характеристиками, влияющими на сложность. По степени новизны программный модуль относится к группе «В» с поправочным коэффициентом 0,7. Значение поправочного коэффициента, учитывающего использование стандартных модулей типовых программ 0,8.

Положительный экономический эффект планируется получать за счет продажи ежемесячной подписки, которая снимает все ограничения с приложения. Продажа, равно как и распространение приложения, будет осуществляться внутри магазинов приложений Google Play и AppStore.

4.2 Расчет затрат на разработку программного обеспечения

Стоимостная оценка ПО предполагает составление сметы затрат, которая в денежном выражении включает следующие статьи расходов:

− заработную плату: основную ($З_{oi}$) и дополнительную ($З_{дi}$);

− отчисления в фонд социальной защиты населения ($З_{сз}$);

− отчисления в Белгосстрах от несчастных случаев на производстве ($Н_{з}$);

− материалы и комплектующие ($Р_м$);
− машинное время ($Р_{мв}$);
− прочие затраты ($Р_{пз}$);
− накладные расходы ($Р_н$).

Расчет основной заработной платы исполнителей, занятых разработкой, проведем на основе исходных данных, представленных в таблице 4.1.

Таблица 4.1 – Исходные данные

Наименование статей	Условные обозначения	Единицы	Норматив
Коэффициент новизны	$К_н$	ед.	0,7
Группа сложности		ед.	1
Дополнительный коэффициент сложности	$К_{сл}$	ед.	0,19
Коэффициент, учитывающий степень использования при разработке ПО стандартных модулей	$К_т$	ед.	0,8
Годовой эффективный фонд времени	$Ф_{эф}$	дней	231
Коэффициент премирования	$К_п$	ед.	1,1
Дополнительная заработная плата исполнителей	$Н_д$	%	10
Ставка отчислений в фонд социальной защиты населения	$Н_{сз}$	%	34
Прочие затраты	$Н_{пз}$	%	1,39
Накладные расходы	$Н_{рн}$	%	13
Ставка НДС (при отсутствии льгот)	$Н_{дс}$	%	20
Налог на прибыль при отсутствии льгот	$Н_п$	%	18
Норма расходов материалов	$Н_м$	руб.	0,38
Норма расхода машинного времени	$Н_{мв}$	маш./ч	2
Цена одного машино–часа	$Ц_{мв}$	руб.	10

Программное обеспечение создавалось и продолжит развиваться на языке программирования Kotlin. Для определения строк исходного кода воспользуемся плагином Statistic для интегрированной среды разработки Android Studio. Перечень и объем функций ПО определим на основе нормативных данных (таблица 4.2).

Таблица 4.2 – Перечень и объем функций

Номер функции	Наименование (содержание)	Объем функции, LOC	
		По каталогу (V_i)	Уточненный (V_{yi})
101	Организация ввода информации	150	150
102	Контроль, предварительная обработка и ввод информации	450	450
103	Преобразование операторов входного языка и команды другого языка	660	520
107	Синтаксический и семантический анализ входного языка и генерация кодов команд	5 400	3 000
109	Организация ввода/вывода информации в интерактивном режиме	3 200	4 000
201	Генерация структуры базы данных	4 300	400
204	Обработка наборов и записей базы данных	2 670	700
209	Организация поиска и поиск в базе данных	5 480	1 600
301	Формирование последовательного файла	290	200
305	Обработка файлов	720	900
402	Генерация программ по описанию пользователей	9 880	6 400
506	Обработка ошибочных и сбойных ситуаций	410	500
507	Обеспечение интерфейса между компонентами	970	400
707	Графический вывод результатов	480	980
	Итого	35 060	20 200

Общий объём ПО (V_0) определяется исходя из количества и объёма реализуемых функций и рассчитываем по формуле:

$$V_0 = \sum_{i=1}^{n} V_i, \tag{4.1}$$

Уточненный объем функций равен:

$$V_{\text{оу}} = \sum_{i=1}^{n} V_{iy}, \qquad (4.2)$$

где $V_{\text{о}}$– общий объем функций, строк исходного кода (LOC);
$V_{\text{оу}}$ – уточненный объем функций, строк исходного кода (LOC);
V_i – объём отдельной функций ПО;
$V_{i\text{у}}$ – уточненный объём отдельной функций ПО;
n – общее число функций.

В формулах (4.1) и (4.2) для расчёта общего объёма ПО используем данные, приведенные в таблице 4.2

$V_{\text{о}} = 150 + 450 + 660 + 5\,400 + 3\,200 + 4\,300 + 2\,670 + 5\,480 + +290 + 720 + 9\,880 + 410 + 970 + 480 = 35\,060$ (LOC),

$V_{\text{оу}} = 150 + 450 + 520 + 3\,000 + 4\,000 + 400 + 700 + 1\,600 + 200 + +900 + 6\,400 + 500 + 400 + 980 = 20\,200$ (LOC).

На основании принятого к расчету объема $V_{\text{оу}}$ и категории сложности определяем нормативную трудоемкость ПО:

$$\text{Т}_{\text{н}} = 556 \left(\frac{\text{чел}}{\text{дн}}\right).$$

ПО принято подразделять на три категории сложности. Категорию сложности ранее определил с руководителем экспертным путем. Нормативная трудоемкость служит основой для определения общей трудоемкости. Ее рассчитываем по формуле:

$$\text{Т}_{\text{о}} = \text{Т}_{\text{н}} \times \text{К}_{\text{с}} \times \text{К}_{\text{т}} \times \text{К}_{\text{н}}, \qquad (4.3)$$

где $\text{Т}_{\text{о}}$ – общая трудоемкость ПО, чел./дн.;
$\text{Т}_{\text{н}}$ – нормативная трудоемкость ПО, чел./дн.;
$\text{К}_{\text{с}}$ – коэффициент, учитывающий повышение сложности ПО;
$\text{К}_{\text{т}}$ – поправочный коэффициент, учитывающий степень использования при разработке стандартных модулей;
$\text{К}_{\text{н}}$ – коэффициент, учитывающий степень новизны ПО.

Коэффициент сложности рассчитаем по формуле:

$$\text{К}_{\text{с}} = 1 + \sum_{i=1}^{n} \text{К}_i, \qquad (4.4)$$

где $К_i$ – коэффициент, соответствующий повышению сложности ПО за счет конкретной характеристики;

n – количество учитываемых характеристик.

Вычисляем коэффициент $К_с$, учитывающий повышение сложности ПО.

$К_1 = 0{,}08$, так как ПО характеризуется функционированием в расширенной операционной среде (имеет связь с другим ПО).

$К_2 = 0{,}06$, так как ПО характеризуется наличием интерактивного доступа.

$К_3 = 0{,}07$, так как ПО характеризуется обеспечением хранения, ведения и поиском данных в сложных структурах.

$К_4 = 0{,}12$, так как ПО имеет 2 характеристики по таблице характеристики категорий сложности ПО.

Подставим значения $К_1$, $К_2$, $К_3$ и $К_4$ в формулу (4.4) и получим:

$$К_с = 1 + 0{,}08 + 0{,}06 + 0{,}07 + 0{,}12 = 1{,}33.$$

Поправочный коэффициент, учитывающий степень использования при разработке стандартных модулей $К_т$ равен $0{,}8$, так как реализуемые функции разработанной программы от 20 до 40 % охватываются стандартными модулями.

Коэффициент новизны $К_н = 0{,}7$, так как создаваемое программное обеспечение относится к ПО, являющимся развитием определенного параметрического ряда ПО, разрабатываемого для ранее освоенных типов конфигурации ПК и ОС (категория новизны В).

На основе вычисленных данных по формуле (4.3) рассчитаем общую трудоемкость:

$$Т_о = 556 \times 1{,}33 \times 0{,}8 \times 0{,}7 = 414 \left(\frac{\text{чел}}{\text{дн}}\right).$$

Основной статьей расходов на создание ПО является заработная плата разработчиков (исполнителей) проекта. На основе общей трудоемкости определим плановое число разработчиков ($Ч_р$) и плановые сроки, необходимые для реализации проекта в целом ($T_р$). При этом возможно решить следующие задачи:

– расчет числа исполнителей при заданных сроках разработки проекта;

– определение сроков разработки проекта при заданной численности исполнителей.

Численность исполнителей проекта рассчитаем по формуле:

$$Ч_р = \frac{Т_о}{T_р \times Ф_{эф}}, \quad (4.5)$$

где $Т_о$ – общая трудоемкость проекта, чел./дней;
$T_р$ – срок разработки проекта, лет;
$Ф_{эф}$ – эффективный фонд времени одного исполнителя, дней.

Эффективный фонд рабочего времени рассчитаем по формуле:

$$Ф_{эф} = Д_г - Д_п - Д_в - Д_о, \quad (4.6)$$

где $Д_г$ – количество дней в году (365 дней);
$Д_п$ – количество праздничных дней в году (9 дней);
$Д_в$ – количество выходных дней в году (103 день);
$Д_о$ – количество дней отпуска (24 дня).

$$Ф_{эф} = 365 - 9 - 103 - 24 = 229 \text{ (дн)}.$$

$$Ч_р = \frac{414}{0.5 \times 229} = 3 \text{ (чел)}.$$

Таким образом, в расчете на 6 месяцев в проекте в среднем будет занято 3 разработчика. Согласно данным [17] о средней заработной плате по стране, в феврале 2023 средняя заработная плата в области научных исследований и разработки в области технических наук составила 2574,6 руб. Примем её за месячную тарифную ставку каждого разработчика.

Дневную тарифную ставку рассчитаем, разделив месячную тарифную ставку на установленную при 5–дневной недельной норме рабочего времени расчетную среднемесячную норму рабочего времени в днях – 22 дня:

$$Т_д = \frac{Т_м}{22}, \quad (4.7)$$

где $Т_д$ – дневная тарифная ставка, руб.;
$Т_м$ – месячная тарифная ставка, руб.

Месячные тарифные ставки возьмем по последней доступной актуальной средней зарплате по стране в схожей области. Месячные и дневные тарифные ставки специалистов: руководителя проекта ($Т_{м1}$, $Т_{д1}$), инженера–программиста 1–й категории ($Т_{м2}$, $Т_{д2}$), инженера–программиста 2–й категории ($Т_{м3}$, $Т_{д3}$) определим ниже:

$$Т_{м1} = Т_{м2} = Т_{м3} = 2574.6 \text{ руб.},$$

$$Т_{д1} = Т_{д2} = Т_{д3} = \frac{1\ 264{,}8}{22} = 117.02 \text{ руб.},$$

На основе результатов и исходных данных таблицы 4.1 рассчитаем сумму основной заработной платы ($З_{оi}$) всех исполнителей по формуле:

$$З_{оi} = \sum_{i=1}^{n} Т_{дi} \times Ф_{эi} \times К_{п}, \quad (4.8)$$

где n – количество исполнителей, занятых разработкой ПО;
$Т_{дi}$ – дневная тарифная ставка i–го исполнителя, руб.;
$Ф_{эi}$ – эффективный фонд рабочего времени i–го исполнителя, дней;
$К_{п}$ – коэффициент премирования.

$$З_{оi} = (117{,}02 \times 115 \times 1{,}1) + (117{,}02 \times 115 \times 1{,}1) + \\ + (117{,}02 \times 115 \times 1{,}1) = 44\ 409{,}09 \text{ руб.}$$

Дополнительную заработную плату ($З_{дi}$) посчитаем по формуле:

$$З_{дi} = \frac{З_{оi} \times Н_{д}}{100}, \quad (4.9)$$

где $Н_{д}$ – норматив дополнительной заработной платы.

$$З_{дi} = \frac{44\ 409{,}09 \times 10}{100} = 4440{,}909 \text{ руб.}$$

Отчисления в фонд социальной защиты населения ($З_{сз}$) рассчитываем в соответствии с действующими законодательными актами по нормативу в процентном отношении к фонду основной и дополнительной зарплаты исполнителей, определенной по нормативу, в целом по организации:

$$З_{сз} = \frac{(З_{оi} + З_{дi}) \times Н_{сз}}{100}, \quad (4.10)$$

где $Н_{сз}$– норматив отчислений в фонд социальной защиты населения, %.

$$З_{сз} = \frac{(44\ 409{,}09 + 4\ 440{,}9) \times 34}{100} = 16\ 608{,}9 \text{ руб.}$$

Отчисления в Белгосстрах от несчастных случаев на производстве ($Н_з$) рассчитываем по формуле:

$$Н_з = \frac{(З_{оi} + З_{дi}) \times Н_е}{100}, \quad (4.11)$$

где $Н_е$– норматив отчислений в Белгосстрах от несчастных случаев на производстве, %.

$$Н_з = \frac{(44\,409{,}09 + 4\,440{,}9) \times 0{,}6}{100} = 293{,}1 \text{ руб.}$$

Расходы по статье «Материалы» ($Р_м$) определяем на основании сметы затрат, разрабатываемой на ПО с учетом действующих нормативов. По статье «Материалы» отражаем расходы на магнитные носители, бумагу, красящие ленты и другие материалы, необходимые для разработки ПО. Нормы расходов материалов определяем в расчете на 100 строк исходного кода. Сумму затрат на расходные материалы рассчитываем по формуле:

$$Р_м = Н_м \times \frac{V_{оу}}{100}, \quad (4.12)$$

где $Н_м$ – норма расхода материалов в расчёте на 100 строк кода ПО;
$V_{оу}$ – общий объем ПО (строк кода).

$$Р_м = 0{,}38 \times \frac{20\,200}{100} = 76 \text{ руб.}$$

Во время разработки данного ПО технические и программные средства специального назначения использоваться не будут. Таким образом расходы по статье «Спецоборудование» ($Р_{со}$) не рассчитываем $Р_{со} = 0$.

Расходы по статье “Машинное время” ($Р_{мв}$) включают оплату машинного времени, необходимого для разработки и отладки ПО:

$$Р_{мв} = Н_{мв} \times \frac{V_o}{100} \times Ц_{мв}, \quad (4.13)$$

где $Ц_{мв}$ – цена одного машино–часа;
$Н_{мв}$ – норматив расхода машинного времени на отладку 100 строк исходного кода, машино–часов.

$$Р_{мв} = 2 \times \frac{20\,200}{100} \times 0.12 = 48{,}48 \text{ руб.}$$

Расходы по статье «Прочие затраты» ($Р_{пз}$) включают затраты на приобретение и подготовку специальной научно–технической информации и специальной литературы. Рассчитываем расходы по статье «Прочие затраты» по формуле, в соответствии с нормативом, разрабатываемым в целом по научной организации, в процентах к основной заработной плате:

$$Р_{пз} = З_{оi} \times \frac{Н_{пз}}{100}, \quad (4.14)$$

где $Н_{пз}$ – норматив прочих затрат в целом по организации, %.

$$Р_{пз} = 44\,409{,}09 \times \frac{1{,}39}{100} = 617{,}3 \text{ руб.}$$

Затраты по статье “Накладные расходы” ($Р_н$) рассчитываем по формуле:

$$Р_н = З_{оi} \times \frac{Н_{рн}}{100}, \quad (4.15)$$

где $Н_{рн}$ – норматив накладных расходов в целом по организации.

$$Р_н = 44\,409{,}09 \times \frac{13}{100} = 5\,773{,}2 \text{ руб.}$$

Общую сумму расходов по вышеизложенным статьям сметы рассчитываем по формуле:

$$С_р = З_{оi} + З_{дi} + З_{сз} + Н_з + Р_м + Р_{мв} + Р_{пз} + Р_н. \quad (4.16)$$

$$С_р = 44\,409{,}09 + 4\,440{,}9 + 16\,608{,}9 + 293{,}1 + 76 + 48 + 617{,}3 + 5\,773{,}2 = 72\,267.$$

Полная себестоимость и её компоненты приведены в таблице 4.3.

Таблица 4.3 – Расчет себестоимости ПО

№	Наименование статей	Условные обозначения	Руб.
1	Основная заработная плата исполнителей	$З_{оi}$	44 409
2	Дополнительная заработная плата исполнителей	$З_{дi}$	4 441
3	Отчисления в фонд социальной защиты населения	$З_{сз}$	16 609
4	Отчисления в Белгосстрах от несчастных случаев на производстве	$Н_{з}$	293
5	Материалы и комплектующие	$Р_{м}$	76
6	Машинное время	$Р_{мв}$	48,5
7	Прочие затраты	$Р_{пз}$	617
8	Накладные расходы	$Р_{н}$	5 773
9	Полная себестоимость	$С_{р}$	72 267

4.3 Расчет стоимостной оценки результата

Результатом реализации проекта будет являться приложение, готовое к выходу на мировой рынок. Прибыль предполагается извлекать ограничением некоторый функциональности приложения, и единой ежемесячной подпиской, снимающей ограничения.

Стоимостная оценка результата предполагает составление прогнозных данных реализации проекта ПО:

1) Определение расчётного периода и расчётных шагов проекта.

2) Обоснование цены ПО.

3) Определение денежных потоков с включением всех денежных поступлений по проекту в ходе его осуществления.

4) Оценку затрат и результатов по проекту в соответствии с принципом «без проекта» и «с проектом».

5) Учёт налогов, сборов, отчислений и льгот, предусмотренных законодательными нормами, действующими в расчётном периоде.

За расчетные периоды возьмем три полугодовых отрезка, начиная со второго полугодия 2023 года.

Рекомендованную цену ПО определил с руководителем экспертным путем на основе анализа конкурентов, примем её за 10 руб./мес.

Приток денежных средств идет исключительно из реализации подписок в Play Market и App Store. Прогнозируемое число месячных платных пользователей по трем расчетным периодам 0, 2000, 6000 соответственно. Следовательно число оплат за каждый расчетный период будет 0, 12000, 36000.

Входной денежный поток $ДП_п$ рассчитывается по формуле

$$ДП_п = \sum_{t_0}^{t_m} Д_{pt} + \sum_{t_0}^{t_m} A_{0t}, \quad (4.19)$$

где $Д_{pt}$ – выручка от продаж;
A_{0t} – амортизационные отчисления.

Входной денежный поток $ДП_п$ рассчитываем по формуле 4.19:

$$ДП_п = 0 + 120\,000 + 360\,000 + 0 + 0 + 0 = 480\,000 \text{ руб.}$$

Выходной денежный поток $ДП_0$ рассчитывается по формуле

$$ДП_0 = И_{t_0} + \sum_{t_0}^{t_m} P_{пt} + \sum_{t_0}^{t_m} P_{yt} + \sum_{t_0}^{t_m} P_{нt}, \quad (4.20)$$

где $И_{t_0}$ – первоначальные инвестиции;
$P_{пt}$ – переменные затраты на производство и реализацию;
P_{yt} – расходы постоянные на управление и обслуживание;
$P_{нt}$ – налоги, уплачиваемые из прибыли.

Выходной денежный поток $ДП_0$ рассчитываем по формуле 4.20:

$$ДП_0 = 72\,267 + (0) + (0 + 72\,267 + 72\,267) + (24\,000 + 72\,000) = 312\,801$$

Чистый денежный поток (ЧД) определяется по формуле

$$ЧД = ДП_п - ДП_0, \quad (4.21)$$

где $ДП_п$ – входной денежный поток;
$ДП_0$ – выходной денежный поток.

Чистый денежный поток (ЧД) определяем по формуле 4.21:

$$ЧД = 480\,000 - 312\,801 = 167\,199 \text{ руб.}$$

Постоянные затраты за второй и третий период примем равными первоначальным инвестициям. По подсчетам выше можем спрогнозировать Прогнозные данные затрат и доходов за три полугодия (таблица 4.4).

Таблица 4.4 – Прогнозные данные реализации проекта ПО

№	Показатели	Единицы измерения	Значение показателей по шагам		
			2023(2) t_0	2024(1) t_1	2024(2) t_2
1	Суммарный объем продаж подписок	экземпляров		12 000	36 000
2	Прогнозная рыночная цена	Руб.		10	10
3	Выручка от продаж (без налогов, включаемых в цену)	Руб.		120 000	360 000
4	Постоянные затраты	Руб.		72 267	72 267
5	Налогооблагаемая прибыль	Руб.		120 000	360 000
6	Налог на прибыль	Руб.		24 000	72 000
7	Чистая прибыль	Руб.		96 000	288 000
8	Капитальные вложения	Руб.	72 267		
9	Входной денежный поток	Руб.		120 000	360 000
10	Выходной денежный поток	Руб.	72 267	96 267	144 267
11	Сумма ЧДД с нарастающим итогом	Руб.	-72 267	-48 534	167 199

В итоге видим, что чистый дисконтированный доход (ЧДД) спустя полтора года развития проекта равен 167 199. Следовательно, рассматриваемый проект по показателю ЧДД соответствует нормативным требованиям: ЧДД > 0. В проект можно инвестировать.

4.4 Выводы по технико–экономическому обоснованию

В технико–экономическом обосновании составлена смета затрат и выполнен расчет суммы расходов, связанных с разработкой компонентов. Также рассчитана сумма доходов. Расчет показывает, что данная разработка является экономически целесообразной и выходит в прибыль спустя полтора года от первоначальных инвестиций.

Чистый дисконтированный доход за 1.5 года поддержки и развития составит 167 199 руб., что представляет собой положительный экономический эффект от создания нового программного средства.

Затраты на разработку программного продукта окупятся на второй год его использования.

Таким образом, разработка и запуск данного продукта является эффективной мерой, инвестиции целесообразно осуществлять. Положительный экономический эффект достигается за счет продажи подписки на полноценную версию ПО конечным пользователям по всему земному шару.

ЗАКЛЮЧЕНИЕ

В результате проделанной работы был полностью изучен, исследован и успешно реализован генетический алгоритм для решения задачи оптимизации на реальных данных. Были изучены альтернативы и экономические детали для составления динамической матрицы расстояний по сторонним апи, а также успешно составлена такая матрица для всех районных и областных центров Республики Беларусь.

Разработано приложение под платформу Android, решающее исходную задачу для городов Республики Беларусь. Показаны преодоленные трудности, особенности реализации и примененные передовые знания и технологии. Разработка решает поставленную задачу в полной мере и легко может быть расширена до конкурентноспособного продукта на мировом рынке.

Сформулирована полная концепция приложения, решающую поставленную проблему, заключающуюся в построении оптимальный маршрут для сокращения расстояния, времени и экономии топлива при наличии достаточно большого количества точек для посещения на автомобиле. Сформулирован концепт развития и дальнейшей монетизации приложения для расширения до глобального рынка.

Цели и задачи работы достигнуты. В дальнейшем предполагается доработка приложения для добавления пользовательского набора данных, добавления платных подписок, добавления импорта и экспорта как для набора городов, так и составленных маршрутов, добавление версии для iOS.

Экономическая целесообразность разработанного приложения и дальнейшего развития была описана в разделе технико-экономического обоснования.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Katoch, S., Chauhan, S.S. & Kumar, V. A review on genetic algorithm: past, present, and future. Multimed Tools Appl 80, 8091–8126 (2021).

[2] Sabry, Ahmed & Benhra, Jamal & El Hassani, Hicham. (2015). Article: A Performance Comparison of GA and ACO Applied to TSP. International Journal of Computer Applications. 117. 28-35. 10.5120/20674-3466.

[3] Kim, Byung-In, Jaechan Shim and Min Zhang. "Comparison of TSP Algorithms Project for Models in Facilities Planning and Materials Handling December 1998." (2001).

[4] Jebari, K. and Madiafi, M., 2013. Selection methods for genetic algorithms. International Journal of Emerging Sciences, 3(4), pp.333-344.

[5] Abdulal, W. and Ramachandram, S., 2011, June. Reliability-aware genetic scheduling algorithm in grid environment. In 2011 International Conference on Communication Systems and Network Technologies (pp. 673-677). IEEE.

[6] Sharma S, Gupta K (2011) Solving the traveling salesman problem through genetic algorithm with new variation order crossover. International Conference on Emerging Trends in Networks and Computer Communications (ETNCC), Udaipur, pp. 274–276

[7] Chang-Yong Lee (2003) Entropy-Boltzmann selection in the genetic algorithms. IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics), vol. 33, no. 1, pp. 138–149, Feb. 2003.

[8] Saini, N., 2017. Review of selection methods in genetic algorithms. International Journal of Engineering and Computer Science, 6(12), pp.22261-22263.

[9] ОС Android [Электронный ресурс]. – Режим доступа : https://android.com

[10] Android Studio [Электронный ресурс]. – Режим доступа : https://developer.android.com/studio

[11] Kotlin [Электронный ресурс]. – Режим доступа : https://kotlinlang.org/

[12] Kotlin Multiplatform Mobile [Электронный ресурс]. – Режим доступа : https://kotlinlang.org/docs/multiplatform-mobile-getting-started.html

[13] Compose Multiplatform UI [Электронный ресурс]. – Режим доступа : https://www.jetbrains.com/lp/compose-multiplatform/

[14] The Bing Maps Distance Matrix API [Электронный ресурс]. – Режим доступа: https://www.microsoft.com/en-us/maps/distance-matrix

[15] Google Maps SDK [Электронный ресурс]. – Режим доступа: https://developers.google.com/maps/documentation/android-sdk

[16] Палицын В.А. Технико–экономическое обоснование дипломных проектов: Методическое пособие для студентов всех специальностей БГУИР. Часть 4: Проекты программного обеспечения. Мн.: БГУИР, 2006. – 76с.

[17] Средняя заработная плате по стране [Электронный ресурс]. – Режим доступа : https://www.belstat.gov.by/ofitsialnaya-statistika/realny-sector-

ekonomiki/stoimost-rabochey-sily/operativnye-dannye/o-nachislennoy-sredney-zarabotnoy-plate-rabotnikov/

ПРИЛОЖЕНИЕ А
(обязательное)

Листинг исходного кода классов генетического алгоритма

GeneticAlgorithm

```
package com.dragu.dragutsp.core.algorithm

import android.util.Log
import com.dragu.dragutsp.core.model.Chromosome
import com.dragu.dragutsp.core.model.Population
import com.dragu.dragutsp.extensions.distanceTo
import com.dragu.dragutsp.extensions.timeTo
import com.dragu.dragutsp.persistance.room.cities.model.City
import kotlin.random.asKotlinRandom

/**
 * The Genetic Algorithm system. This class brings together the entire process
 * of the Genetic Algorithm.
 */
class GeneticAlgorithm(
    val cities: List<City>,
    val populationSize: Int,
    val k: Int,
    val maxGen: Int,
    val elitismValue: Int,
    val crossoverRate: Double,
    val mutationRate: Double,
    val random: java.util.Random,
    val crossoverType: CrossoverType,
    val mutationType: MutationType,
    val onNewPopulation: (population: Population, isFinal: Boolean) -> Unit
) {
    companion object {
        private val TAG = GeneticAlgorithm::class.java.simpleName
    }

    //
    private var population: Population
    private val crossover = Crossover()
    private val mutation = Mutation(mutationType)
    private val selection = Selection()
    private val ktRandom = random.asKotlinRandom()

    private val averageDistanceOfEachGeneration: MutableList<Double> = mutableLis-
tOf()
    private val bestDistanceOfEachGeneration: MutableList<Double> = mutableListOf()
    private val hashMapCitiesByShId: HashMap<Short, City> = hashMapOf()

    init {
        Log.d(TAG, "init genetic Algo")
        cities.forEach {
            hashMapCitiesByShId[it.id.toShort()] = it
        }
        population = Population(populationSize, generationNumber = 1, calculateDis-
tance = {
            calculateDistance(it)
```

```
        }, calculateDuration =
        { calculateTime(it) }
        ).apply {
            populate(cities)
        }
    }

    @Synchronized
    private fun calculateDistance(array: ShortArray): Double {
        var distanceTravelled = 0.0
        array.forEachIndexed { index, sh ->
            distanceTravelled += if (index != array.lastIndex) {
                hashMapCitiesByShId[sh]?.distanceTo(
                    hashMapCitiesByShId[array[index + 1]]!!
                ) ?: 0.0
            } else 0.0
        }
        return distanceTravelled
    }

    @Synchronized
    private fun calculateTime(array: ShortArray): Double {
        var time = 0.0
        array.forEachIndexed { index, sh ->
            time += if (index != array.lastIndex) {
                hashMapCitiesByShId[sh]?.timeTo(
                    hashMapCitiesByShId[array[index + 1]]!!
                ) ?: 0.0
            } else 0.0
        }
        return time
    }

    fun run() {
        averageDistanceOfEachGeneration.add(population.getAverageDistance())
        bestDistanceOfEachGeneration.add(population.getMostFit().getDistance())
        val repeatTimes = maxGen
        repeat(repeatTimes) {
            Log.d(TAG, "running generation ${it + 1}")
            population = createNextGeneration(it + 1)
            onNewPopulation(population, it + 1 == repeatTimes)
            averageDistanceOfEachGeneration.add(population.getAverageDistance())
            bestDistanceOfEachGeneration.add(population.getMostFit().getDistance())
        }
    }

    private fun createNextGeneration(generationNumber: Int): Population {
        val nextGeneration =
            Population(maxSize = populationSize, generationNumber, calculateDistance
= {
                calculateDistance(it)
            },calculateDuration =
            { calculateTime(it) })
        val sortedChromosomes = population.getChromosomes().sortedBy { it.getDis-
tance() }
        //save elits to the next generation
        if (elitismValue > 0) {
            sortedChromosomes.subList(0, elitismValue).also { eliteChromosomes ->
                eliteChromosomes.forEach {
                    nextGeneration.add(it)
```

```
                }
            }
        }

        while (nextGeneration.size() < population.size()) {
            val doCrossover = random.nextDouble() <= crossoverRate
            val doMutation = random.nextDouble() <= mutationRate
            val p1 = selection.tournamentSelection(population, k, ktRandom)

            var child: Chromosome
            child = if (doCrossover) {
                val p2 = selection.tournamentSelection(population, k, ktRandom)
                crossover.orderCrossover(p1, p2, random)
            } else {
                p1
            }
            if (doMutation) {
                child = mutation.mutate(child, random, ktRandom)
            }
            nextGeneration.add(child)
        }
        return nextGeneration
    }
}
```

Population

```
package com.dragu.dragutsp.core.model

import com.dragu.dragutsp.persistance.room.cities.model.City
import java.nio.BufferOverflowException
import java.util.PriorityQueue

/**
 * Represents a Population of chromosomes.
 * @param maxSize the maximum size of the Population
 */
class Population(
    private val maxSize: Int,
    val generationNumber: Int,
    val calculateDistance: (arrayOfIds: ShortArray) -> Double,
    val calculateDuration: (arrayOfIds: ShortArray) -> Double
    ) : Iterable<Chromosome> {

    private val chromosomes: PriorityQueue<Chromosome> = PriorityQueue()

    override fun iterator(): Iterator<Chromosome> = chromosomes.iterator()



    /**
     * Adds a Chromosome to the Population.
     * @param chromosome the chromosome to add
     */
    fun add(chromosome: Chromosome) {
        if (chromosomes.size == maxSize) {
            throw BufferOverflowException()
        }
        chromosomes.add(chromosome)
    }
```

```
    fun populate(cities: List<City>) {
        if (chromosomes.size == maxSize) {
            throw BufferOverflowException()
        }

        while (chromosomes.size < maxSize) {
            add(Chromosome.constructAndShuffle(
                cities = cities.map { it.id.toShort() }.toShortArray(),
                calculateDistance = { calculateDistance(it) }
            ))
        }
    }

    /**
     * Get a list of all the Chromosomes.
     * @return  the list of the Chromosomes
     */
    fun getChromosomes(): PriorityQueue<Chromosome> = chromosomes


    /**
     * Get the size of the Population.
     * @return  the number of all the Chromosomes.
     */
    fun size(): Int {
        return chromosomes.size
    }

    /**
     * Gets the average distance of all the chromosomes.
     * @return  the mean distance travelled
     */
    fun getAverageDistance(): Double {
        var averageDistance = 0.0
        for (chromosome in chromosomes) {
            averageDistance += chromosome.getDistance()
        }
        return averageDistance / chromosomes.size
    }

    /**
     * Get the Chromosome that has the path with the least distance.
     * @return  the most fit Chromosome
     */
    @Throws(NullPointerException::class)
    fun getMostFit(): Chromosome {
        return chromosomes.peek()!!
    }
}
```

Chromosome

```
package com.dragu.dragutsp.core.model

import android.util.Log
import java.util.concurrent.ThreadLocalRandom

/**
 * @param cities short max value is 32767
```

```
 */
data class Chromosome(
    val cities: ShortArray,
    val calculateDistance: (arrayOfIds: ShortArray) -> Double
) : Comparable<Chromosome> {
    private var distance: Double = -1.0

    fun getDistance(): Double {
        //If this was already calculated, don't calc again
        if (distance != -1.0) {
            return distance
        }
        distance = calculateDistance(cities)
        return distance
    }

    override fun compareTo(other: Chromosome): Int {
        return (getDistance() - other.getDistance()).toInt()
    }

    companion object {
        val TAG = Chromosome::class.java.simpleName

        fun constructAndShuffle(
            cities: ShortArray,
            calculateDistance: (arrayOfIds: ShortArray) -> Double
        ): Chromosome {
            //first and last cities are immutable. So for 3 cities Chromosome would
be the same
            if (cities.size < 4) {
                Log.d(TAG, "too small root size, returning initial cities size ${cit-
ies.size}")
                return Chromosome(cities, calculateDistance)
            } else {
                /**
                 * Helper method. Swaps tho cities
                 * @param i index of the first city
                 * @param j index of the second city
                 * @param cities mutable list, shuffling would be done inside this
list
                 */
                fun swap(i: Int, j: Int, cities: ShortArray) {
                    val temp = cities[i]
                    cities[i] = cities[j]
                    cities[j] = temp
                }

                for (i in 1 until cities.lastIndex) {
                    swap(i, ThreadLocalRandom.current().nextInt(1, cities.lastIndex),
cities)
                }
                return Chromosome(cities, calculateDistance)
            }
        }
    }
}
```

Mutation

```
package com.dragu.dragutsp.core.algorithm
```

```
import com.dragu.dragutsp.core.model.Chromosome

class Mutation(private val type: MutationType) {

    fun mutate(p1: Chromosome, r: java.util.Random, rkt: kotlin.random.Random): Chro-
mosome {
        return when (type) {
            MutationType.DISPLACEMENT -> displacementMutation(p1, r)
            MutationType.EXCHANGE -> exchangeMutation(p1, rkt)
            MutationType.DRAGU -> {
                if (r.nextBoolean()) {
                    displacementMutation(p1, r)
                } else {
                    exchangeMutation(p1, rkt)
                }
            }
        }
    }

    /**
     * Exchange 2 random cities
     */
    private fun exchangeMutation(p1: Chromosome, r: kotlin.random.Random): Chromosome
{
        val parent: ShortArray = p1.cities.clone()
        val i = r.nextInt(1, parent.lastIndex)
        val j = r.nextInt(1, parent.lastIndex)
        val temp = parent[i]
        parent[i] = parent[j]
        parent[j] = temp
        return Chromosome(parent, p1.calculateDistance)
    }


    /**
     * Select the random sublist of cities, then shift it to the random place
     * @param p1        chromosome to be mutated
     * @param r         Random object
     * @return          New mutated Chromosome
     */
    private fun displacementMutation(p1: Chromosome, r: java.util.Random): Chromosome
{
        val firstCity = p1.cities.first()
        val lastCity = p1.cities.last()
        val parent: ShortArray = p1.cities.copyOfRange(1, p1.cities.lastIndex)
        val totalCities = parent.size
        val firstPoint: Int = r.nextInt(totalCities)
        val secondPoint: Int = r.nextInt(totalCities - firstPoint) + firstPoint
        val insertionSublist = parent.copyOfRange(firstPoint, secondPoint + 1)
        val cuttedSubParent: ArrayList<Short> =
            ArrayList(parent.subtract(insertionSublist.asIterable().toSet()))
        val indexToInsert = cuttedSubParent.indices.randomOrNull() ?: 0
        cuttedSubParent.addAll(index = indexToInsert, insertionSublist.asItera-
ble().toList())
        val result = ShortArray(p1.cities.size)
        result[0] = firstCity
        cuttedSubParent.toShortArray().copyInto(result, destinationOffset = 1)
        result[result.lastIndex] = lastCity
        if (result.size != p1.cities.size) {
            throw AssertionError("Check the implementation! Probably your cutting is
```

```
incorrect")
        }
        return Chromosome(result, p1.calculateDistance)
    }
}
```

Crossover

```
package com.dragu.dragutsp.core.algorithm

import com.dragu.dragutsp.core.model.Chromosome

class Crossover {
    /**
     * Performs a crossover on all the cities between two points.
     * @param p1    the first parent chromosome
     * @param p2    the second parent chromosome
     * @param r     the Random object for selecting a point
     * @return      the children
     */
    fun orderCrossover(p1: Chromosome, p2: Chromosome, r: java.util.Random): Chromo-
some {
        //cutting first and last city
        val firstCity = p1.cities.first()
        val lastCity = p1.cities.last()
        val parent1: ShortArray = p1.cities.copyOfRange(1, p1.cities.lastIndex)
        val parent2: ShortArray = p2.cities.copyOfRange(1, p2.cities.lastIndex)
        val child = ShortArray(parent1.size) { -1 }
        val citiesInChild: HashSet<Short> = HashSet()
        val totalCities = parent1.size
        val firstPoint: Int = r.nextInt(totalCities)
        val secondPoint: Int = r.nextInt(totalCities - firstPoint) + firstPoint

        // Inherit the cities from parent 1
        for (i in firstPoint..secondPoint) {
            child[i] = parent1[i]
            citiesInChild += parent1[i]
        }
        //exclude presented in child elements from parent 2
        for (i in 0..parent2.lastIndex) {
            if (citiesInChild.contains(parent2[i])) {
                citiesInChild.remove(parent2[i])
                parent2[i] = -1
            }
            if (citiesInChild.isEmpty()) break
        }

        //add elements from parent 2
        val parent2Filtered = parent2.filter { it != (-1).toShort() }
        var parent2InheritanceIndex = 0
        for (i in 0 .. child.lastIndex) {
            if (child[i] == (-1).toShort()){
                child[i] = parent2Filtered[parent2InheritanceIndex]
                parent2InheritanceIndex++
            }
        }

        val result = ShortArray(child.size + 2)
        result[0]= firstCity
        child.copyInto(result, destinationOffset = 1)
```

```
        result[result.lastIndex] = lastCity
        if (result.size != p1.cities.size || result.size != p2.cities.size) {
            throw AssertionError("Check the implementation! Probably your cutting is
incorrect")
        }
        return Chromosome(result, p1.calculateDistance)
    }
}
```

Selection

```
package com.dragu.dragutsp.core.algorithm

import com.dragu.dragutsp.core.model.Chromosome
import com.dragu.dragutsp.core.model.Population
import java.util.PriorityQueue


class Selection {
    // There is a 1 in 5 chance that fittest individual is not selected.
    private val ODDS_OF_NOT_PICKING_FITTEST = 5

    /**
     * Picks k Chromosomes at at random and then return the best one.
     * There is a small chance that the best one will not be selected.
     * @param population    the population to selected from
     * @param k             the number of chromosomes to select
     * @param random        the Random object for randomly selecting
     * @return              usually the fittest Chromosome from k randomly selected
chromosomes
     */
    fun tournamentSelection(population: Population, k: Int, random: kotlin.ran-
dom.Random): Chromosome {
        require(k >= 1) { "K must be greater than 0." }
        val chromosomes = population.getChromosomes()
        val kChromosomes: List<Chromosome> = getKChromosomesSorted(chromosomes, k,
random)
        return getChromosome(kChromosomes, random)
    }

    /**
     * Returns k randomly selected Chromosomes.
     * @param pop       a list of Chromosomes (a population)
     * @param k         the number of Chromosomes to randomly select
     * @param random    the Random object used for picking a random chromosomes
     * @return          k randomly selected chromosomes
     */
    private fun getKChromosomesSorted(
        pop: PriorityQueue<Chromosome>,
        k: Int,
        random: kotlin.random.Random
    ): List<Chromosome> {
        val kChromosomes= mutableListOf<Chromosome>()
        for (j in 0 until k) {
            kChromosomes.add(pop.random(random))
        }
        return kChromosomes.sorted()
    }

    /**
```

```
     * Get the best Chromosome in a list of Chromosomes. There is a small chance
     * that a randomly selected Chromosome is picked instead of the best one.
     * @param list     the list of Chromosomes
     * @param random        the Random object used for selecting a random Chromosome
if needed
     * @return              usually the best Chromosome
     */
    private fun getChromosome(list: List<Chromosome>, random: kotlin.random.Random):
Chromosome {
        val bestChromosome: Chromosome = getBestChromosome(list)

        // 1 in 5 chance to return a chromosome that is not the best.
        if (random.nextInt(ODDS_OF_NOT_PICKING_FITTEST) == 0) {
            return list.random(random)
        }
        return bestChromosome
    }

    /**
     * Get the best Chromosome in a list of Chromosomes.
     * @param list  the list to search
     * @return      the best chromosome
     */
    private fun getBestChromosome(list: List<Chromosome>): Chromosome =
        list.first()

}
```

ПРИЛОЖЕНИЕ Б
(обязательное)

Листинг исходного кода моделей данных

City.kt

```
package com.dragu.dragutsp.persistance.room.cities.model

import androidx.room.Entity
import androidx.room.PrimaryKey
import com.dragu.dragutsp.CITIES_TABLE_NAME
import com.dragu.dragutsp.IS_SELECTED_COLUMN_NAME
import com.squareup.moshi.Json
import com.squareup.moshi.JsonClass

@Entity(tableName = CITIES_TABLE_NAME)
@JsonClass(generateAdapter = true)
data class City(
    //make id-city strong connection
    @PrimaryKey
    val id: Int,
    val lat: Double,
    val lon: Double,
    val cityRegion: Int,
    val travelDistances: List<Double> = listOf(),
    val travelDurations: List<Double> = listOf(),
    val coatOfArms: String = "",
    @Json(name = IS_SELECTED_COLUMN_NAME)
    val isSelected: Boolean = false,
    val isHuge: Boolean = false, //10 biggest cities
    val isSmall: Boolean = false //cities smaller then administrative district center
)

@Entity
@JsonClass(generateAdapter = true)
data class CityDistanceDurationUpdate(
    val id: Int,
    val travelDistances: List<Double> = listOf(),
    val travelDurations: List<Double> = listOf()
)

@Entity
@JsonClass(generateAdapter = true)
data class CitySelectionUpdate(
    val id: Int,
    val isSelected: Boolean
)

const val CITY_REGION_BREST = 1
const val CITY_REGION_VITEBSK = 2
const val CITY_REGION_GOMEL = 3
const val CITY_REGION_GRODNO = 4
const val CITY_REGION_MINSK = 5
const val CITY_REGION_MOGILEV = 6
//if sometimes we extend to add custom points to the graph
const val CITY_REGION_OTHER = 7
```

CitiesDao

```
package com.dragu.dragutsp.persistance.room.cities

import androidx.room.Dao
import androidx.room.Insert
import androidx.room.OnConflictStrategy
import androidx.room.Query
import androidx.room.Update
import com.dragu.dragutsp.CITIES_TABLE_NAME
import com.dragu.dragutsp.IS_SELECTED_COLUMN_NAME
import com.dragu.dragutsp.persistance.room.cities.model.City
import com.dragu.dragutsp.persistance.room.cities.model.CityDistanceDurationUpdate
import com.dragu.dragutsp.persistance.room.cities.model.CitySelectionUpdate
import kotlinx.coroutines.flow.Flow

@Dao
interface CitiesDao {

    @Insert(onConflict = OnConflictStrategy.REPLACE)
    suspend fun insertCities(cities: List<City>)

    /**
     * Update distance and duration matrix values
     */
    @Update(entity = City::class)
    suspend fun update(obj: CityDistanceDurationUpdate)

    /**
     * Update isSelected flag for city id
     */
    @Update(entity = City::class)
    suspend fun update(obj: CitySelectionUpdate)

    @Query("UPDATE $CITIES_TABLE_NAME SET $IS_SELECTED_COLUMN_NAME = :newIsSelected
WHERE id IN (:ids)")
    suspend fun updateMultipleCities(ids:List<Int>, newIsSelected:Boolean)

    @Query("SELECT * FROM $CITIES_TABLE_NAME ORDER BY id")
    fun observeCities(): Flow<List<City>>

}
```

Route.kt

```
package com.dragu.dragutsp.persistance.room.route.model

import androidx.room.Entity
import androidx.room.PrimaryKey
import com.dragu.dragutsp.IS_SELECTED_COLUMN_NAME
import com.dragu.dragutsp.ROUTES_TABLE_NAME
import com.squareup.moshi.Json
import com.squareup.moshi.JsonClass

@Entity(tableName = ROUTES_TABLE_NAME)
@JsonClass(generateAdapter = true)
data class Route(
    @PrimaryKey(autoGenerate = true)
    val id: Int = 0,
    val timeGenerated: Long = System.currentTimeMillis(),
    val travelDistance: Double,
```

```
    val travelDuration: Double,
    @Json(name = IS_SELECTED_COLUMN_NAME)
    val isSelected: Boolean = false,
    val subRoutes: List<SubRoute> = listOf()
)

@JsonClass(generateAdapter = true)
data class SubRoute(
    val id: Int,
    val depatureCityId: Int,
    val destinationCityId: Int,
    val travelDistance: Double,
    val travelDuration: Double,
    val depatureLat: Double,
    val depatureLon: Double,
    val depatureCoatOfArms: String,
    val destinationLat: Double,
    val destinationLon: Double,
    val destinationCoatOfArms: String,
    val isFinished: Boolean
)

@Entity
@JsonClass(generateAdapter = true)
data class RouteSubRoutesUpdate(
    val id: Int,
    val subRoutes: List<SubRoute> = listOf()
)
```

RoutesDao

```
package com.dragu.dragutsp.persistance.room.route

import androidx.room.Dao
import androidx.room.Delete
import androidx.room.Insert
import androidx.room.OnConflictStrategy
import androidx.room.Query
import androidx.room.Update
import com.dragu.dragutsp.IS_SELECTED_COLUMN_NAME
import com.dragu.dragutsp.ROUTES_TABLE_NAME
import com.dragu.dragutsp.persistance.room.route.model.Route
import com.dragu.dragutsp.persistance.room.route.model.RouteSubRoutesUpdate
import kotlinx.coroutines.flow.Flow

@Dao
interface RoutesDao {
    @Insert(onConflict = OnConflictStrategy.REPLACE)
    suspend fun insertRoute(route: Route)

    @Query("SELECT * FROM $ROUTES_TABLE_NAME ORDER BY id")
    fun observeRoutes(): Flow<List<Route>>

    /**
     * Update subroutes
     */
    @Update(entity = Route::class)
    suspend fun update(obj: RouteSubRoutesUpdate)

    @Query("UPDATE $ROUTES_TABLE_NAME SET $IS_SELECTED_COLUMN_NAME = 0")
```

```
    suspend fun deselectAll()

    /**
     * So the query will update the isSelected field to 1 for the row where the id
column matches
     * the id parameter, and set isSelected to 0 for all other rows where the id col-
umn is not equal
     * to the id parameter. This ensures that only one row has isSelected set to true
at any given time.
     */
    @Query("UPDATE $ROUTES_TABLE_NAME SET $IS_SELECTED_COLUMN_NAME = CASE WHEN id =
:id THEN 1 ELSE 0 END")
    suspend fun updateSelectedEntity(id: Int)

    @Delete
    suspend fun delete(route: Route)
}
```

CityExtensions.kt

```
package com.dragu.dragutsp.extensions

import com.dragu.dragutsp.R
import com.dragu.dragutsp.persistance.room.cities.model.City

/**
 * return double value, km
 */
fun City.distanceTo(city: City): Double {
    //Id starts from 1, and matrix indices starts from 0
    return this.travelDistances[city.id - 1]
}

/**
 * return double value, minutes
 */
fun City.timeTo(city: City): Double {
    //Id starts from 1, and matrix indices starts from 0
    return this.travelDurations[city.id - 1]
}
```