## **TЕХНОЛОГИЧНО УЧИЛИЩЕ ЕЛЕКТРОННИ СИСТЕМИ**

**

**към ТЕХНИЧЕСКИ УНИВЕРСИТЕТ - СОФИЯ**

**ДИПЛОМНА РАБОТА**

**по професия код 481020 „Системен програмист“**

**специалност код 4810201**  **„Системно програмиране“**

Тема: Алгоритъм за изчисляване на велосипедни маршрути върху данни от OpenStreetMap карти, приложен в iOS мобилно приложение

Дипломант: Дипломен ръководител:

*Николай Здравков Динков Иван Дилчовски*

СОФИЯ

2 0 2 2



**Отзив на научния ръководител**

**Увод**

Всеки човек с някакви разполагаеми финанси в модерния, цивилизован свят на XXI век притежава мобилен телефон – по-конкретно смартфон. Това е резултат на огромното влияние, което индустрията свързана с телефони има върху цялостния живот на човека в XXI век. Пътят на първите смартфони датира още от края на ХХ век. През 1992г. излиза първият смартфон, който бива управляван посредством жестове върху екрана си. Той съдържа приложения като календар, електронна поща, факс и други.

Бивайки смартфони, тези устройства се нуждаят и от съответните разновидности операционни системи и приложен софтуер за ефективно, качествено и приятно използване от потребителите, както и за избор от страна на производителите на тези телефони. Оттам се заражда и индустрията на разработката на мобилни приложения след направата на първите публични комплекти за разработка на софтуер (SDK) от притежателите на тези операционни системи с цел популяризирането на платформите чрез разнообразни приложения. Първоначално, повечето телефони са имали само прости приложения като калкулатори, хронометри, аларми, настройки (т.нар. системни приложения) разработени от създателите на дадената операционна система, но появата на нови SDK и платформи като Google Play Store и App Store дават възможност за дистрибутиране на такива собствени приложения. Има много големи мобилни операционни системи като една от най-известните е iOS, която е разработена от Apple. Тя се появява на пазара през 2007 година заедно оригиналния iPhone. По-късно тя бива използвана и в мобилните устройства iPod Touch, iPad и Apple TV на Apple.

С появата на много собствени приложения се появяват и първите приложения, чиято цел е да навигират от дадено начално място до дадена дестинация (т.нар. навигации). Навигациите набират популярност с изключително висока скорост като през годините се появяват много и различни реализации от различни компании, което се дължи на факта, че навигациите имат доста широк набор от приложения в различни сфери. Като през XXI век има доста различни реализация като се делят на различен вид спрямо начина на придвижване от началната точка до крайната, а именно – велосипед, кола, градски транспорт и пеш. Към 2022 г. навигациите, които целят да предоставят маршрут за велосипедисти са изключително малко на брой в глобален мащаб, за България са ограничени.

Съответно, дипломният проект има за главни цели да реализира алгоритъм за изчисляване на велосипедни маршрути, приложен в iOS мобилно приложение.

**Първа Глава**

**Методи и технологии за реализация на мобилни приложения**

* 1. **История и основни понятия на мобилните операционни системи – iOS и Android**

В днешно време има голямо разнообразие на мобилни операционни системи, използвани от телефоните. На пазара има представени всякакви разновидности и новости в сферата на мобилните операционни системи, но най-популярните и използвани през годините са iOS и Android.

* + 1. **iOS**

iOS операционната система е издадена и направена от Apple Inc. през 2007 г. Тогава за пръв път е показана заедно с iPhone устройството на широката публика. До 2022 г. е достигнало съдържание на повече от 2,1 милиона различни приложения, като тяхното разпределение се извършва чрез iOS App Store. Главната хардуерна платформа за iOS е ARM архитектура (ARMv7, ARMv8-A, ARMv8.2-A, ARMv8.3-A). Изданията на iOS преди iOS 7 могат да се изпълняват само на устройствата с iOS с 32-битови ARM процесори (архитектура ARMv6 и ARMv7-A). През 2013 г. iOS 7 е пуснат с пълна 64-битова поддръжка, която включва собствено 64-битово ядро, библиотеки, както и всички вградени приложения. Софтуерният стек на iOS се състои от многослойна и йерархична архитектура. От най-високото ниво на абстракция до най-ниското, те са: приложенията, Cocoa Touch слоя (съдържащ платформи за потребителския интерфейс написани в Objective C), медийния слой (графика, аудио, видео, т.н.), слоя на главните сервизи, които са разделени на публични и поверителни приложни интерфейси за програмисти (API) модели и самият слой операционната система.

* + 1. **Android**

Android е създаден и демонстриран за пръв път през 23 септември, 2008 година. Към днешна дата, Android поддържа над 2 милиарда активни потребители с устройства произведени от всякакви различни създатели. Тяхното SDK за създаване на мобилни приложения първоначално излиза октомври, 2009 г. Android операционната система е базирана на силно модифицирано Linux ядро от LTS (дълготрайна подкрепа) бранчовете на проекта и поддържано от AOSP (Android проекта с отворен код). Android архитектурата подобно на iOS е базирана на платформата на ARM, която идва с поддръжка на x86 архитектури с нови версии. Софтуерният стек, подобно на iOS, също е представен като йерархична структура от различни компоненти. От най-високото ниво на абстракция до най-ниското, те са: приложенията, Android SDK, местни библиотеки написани на C, Android Runtime (сега ART, а преди Dalvik), HAL и самото ядро. Android се води от стриктна практика на инкрементална публикация на стабилни версии, които са възможни за актуализация от страна на всеки потребител по всяко време на всяко устройство. Тази практика води до повишена загриженост от страна на програмистите, които трябва да знаят как приложенията им ще функционират на различни версии на операционната система. Най-новата и стабилна версия до момента е Android 12, съпътствана от API ниво 31.

* 1. **Проучване на различни SDK за разработка на мобилни приложения**

За разлика от доминантността на популярните и често използвани от мобилни устройства операционни системи, има сравнително по-голям набор от ефективни и популярни SDK направени за създаване на приложения за тези две платформи. Разбира се, съществуват и основните инструменти предоставени от Google и Apple (съответно Android SDK и iOS SDK), но се открояват и много други платформи. Ще се разгледа най-удостовереното и популярно SDK различно от Android и iOS: Flutter.

* + 1. **Flutter SDK**

Flutter е SDK, чийто код се компилира на iOS, Android и Fuchsia. Flutter е направен от Google и демонстриран за пръв път през 2015 г., достигайки публична версия 1.0 през 2018 г. Силата му идва с междуплатформените способности и сравнително улеснена система за разработка на приятни за окото потребителски интерфейси, използвайки като главен език Dart. От архитектурна гледна точка, Flutter е отличаваща се платформа. На база своя специален двигател за изграждане на потребителски интерфейс, който, заедно с главните Foundation API написани на Dart, успява да създаде три отделни условни “дървета”. Те изграждат едно приложение, чиято реализация се гради на бърза и активна система за създаване на потребителски интерфейс разделена тотално от главното състояние на приложението. Това SDK предоставя функционалността hot reload, която означава бързо, живо презареждане на потребителския интерфейс, докато се запазва цялостното състояние на приложението, ускорявайки и улеснявайки процеса на разработка на дадено приложение. Flutter предоставя и широка гама от инструменти и ресурси. Разбира се, това SDK има и лоши страни, една от тях е това, че има голямо нарастване на зависимостите от библиотеки на цялостното приложение и това, че с нарастването на използваните библиотеки приложението става по-тежко. Flutter, също така, е сравнително нов на сцената, и освен че има някои основни проблеми за оправяне, някои от главните API версии все още са в процес на дълга и обменна реализация. Въпреки това, той има установени стандарти и принципи за разработка на мобилни приложения. Това най-ясно се забелязва в различните методи за запазване на състоянието на приложението. Съществуват и бъдеще планове за стабилизиране и доразвиване на версиите на Flutter Web и Flutter Desktop, които ще разширят хоризонта на платформата, но към сегашният момент няма големи продукти или много добри проекти реализирани с тях.

* + 1. **Android SDK**

Android SDK е основната система предоставена от Google за създаването на приложения за Android. Тясно интегрирана с Android операционната система, Android SDK позволява ефективната комуникация с голям набор от компоненти, комуникация водена като се спазват стриктните правила на операционната система. Android поддържа всичко, което на един разработчик би му трябвало за едно приложение. От известно време официално препоръчания език за писането на Android приложения е Kotlin, като преди него се е препоръчвал Java. Голяма част от NDK (местен комплект за разработка) разработените API са написани на C/C++, които играят като посредници между тези използвани от разработчици на приложения и по-ниските нива на операционната система, предоставят основните от предишно споменатите функционалности. Реализацията на потребителския интерфейс става чрез XML, който XML е разделен на файлове с отделни категории и цели, а те самите се намират в res папката на всеки даден проект. Голяма част от средата за разработка на Android се гради от въвеждането на нови стандарти и препоръки. Това оставя някои по-стари приложения, които не могат да променят целият си сорс код всеки път, да се водят неспазващи модерните принципи.

* + 1. **iOS SDK**

iOS SDK предлага услуги за създаването на приложения за iOS. Въпреки ограничението спрямо устройствата, за които може да се създават приложения, iOS все още е силно установена платформа и предоставя голям набор от инструменти на разработчиците. iOS използва като главен език Swift, а до преди няколко години - Objective C. Създаването на потребителски интерфейс се осъществява чрез UIKit и Storyboard, а заедно с тях през последните години е популяризиран и SwiftUI, който наподобява система близка до тази на Flutter. Те придават на програмисти възможност за избор спрямо тази част. От гледна точка на популярност и наличие на пазара, iOS може да се каже, че е малко по-напред спрямо конкурента си – Android.

* 1. **OSM файлове**

OpenStreetMap използва няколко различни типа файлове, съдържащи различни типове данни и използва различни формати, за да „кодира“ тези данни в битове и байтове.

* + 1. **Типове файлове**

OSM използва три типа файлове за основните си данни:

* Файлове с данни - Това са най-често срещаните файлове, съдържащи OSM данни в определен момент от време. Това може да бъде или планетарен файл, съдържащ всички OSM данни, или някакъв вид извлечение под формата на регион или страна. В този файл се съдържа най-много една версия на всеки обект. Изтритите обекти до дадения момент от време не са в този файл. Обичайният използван суфикс е „.osm“.
* Файлове с история - Тези файлове съдържат не само текущата версия на обект, но и тяхната история. Така че за всеки обект може да има нула или повече версии в този файл. Изтритите обекти също могат да бъдат в този файл. Обичайният използван суфикс е „.osm“ или „.osh“. Тъй като понякога се използва същият суфикс като за нормалните файлове с данни и тъй като в името на файла няма ясен индикатор за това кой тип е дадения файл, не винаги е ясно какъв е типа на файла.
* Промяна на файлове – Понякога тези файлове биват наричани „файлове на различията“, те съдържат промените между едно състояние на OSM базата данни и друго състояние. Файловете за промяна могат да съдържат няколко версии на обект. Обичайният използван суфикс е „.osc“.

В обобщение, общото за всички тези файлове е, че съдържат OSM обекти. Файловете с история и файловете за промяна могат да съдържат няколко версии на един и същ обект, както и изтрити обекти, файловете с данни не могат.

* + 1. **Файлови формати**

Има няколко различни OSM файлови формата, които се използват често. Файловите формати описват начина, по който съдържанието е кодирано в битове и байтове.

Кратък преглед на всички предоставени формати:

* XML - Оригиналният базиран на XML OSM формат. Този формат е доста голям и работата с него е изключително бавна, но все още се използва често и в някои случаи няма алтернатива. Основният API на OSM база данни също връща своите данни в този формат.
* PBF - Двоичен формат, базиран на протокола за буферното кодиране. Това е най-компактният формат.
* O5M / O5C - Този двоичен формат е по-прост от PBF формата, но не се използва толкова широко. O5M е форматът за файлове с данни, O5C версията за файлове за промяна.
* OPL - Прост формат, подобен на CSV-файловете с един OSM обект на ред. Този формат е предназначен за лесна употреба със стандартни инструменти на командния ред на UNIX като „grep“, „cut“, и „awk“.
* Файл служещ за отстраняване на проблеми - Приятно форматиран текстово базиран формат, който е по-лесен за четене за човек от XML или OPL форматите. Както подсказва името му, неговото предназначение е за отстраняване на грешки.
* Черна дупка – Формат, който изхвърля всички данни. Може само да се пише, но не и да се чете.
  + 1. **Подреждане на обекти във файлове**

Всички OSM файлове имат обекти, които се съдържат във файла в произволен ред. Това е независимо от типа или формата на файла. Обикновено обектите се сортират по специфичен начин, но дали обектите са сортирани или не и по какъв начин не е част от самия файлов формат. Имайки това предвид, OSM файловете почти винаги са подредени по специфичен начин. Най-често срещания вариант е - първо възли, на английски „nodes“, след това пътища, на английски „way“, след това отношения, на английски „relation“. Всяка група е подредена по нарастващ идентификатор (и възходяща версия във файловете с история). Файловете с набор от промени обикновено се подреждат по идентификатор на набора от промени.

* + 1. **XML формат**

В проекта OSM се използват няколко различни XML формата. Основните формати са този, използван за файлове за планетата или за региони (суфикс „.osm“), форматът, използван за файлове за промяна (суфикс „.osc“) и форматът на историята (суфикси „.osm“ или „.osh“). Използват се и някои вариации, като например форматът JOSM, който е подобен на нормалния OSM формат, но има някои допълнения.

* + 1. **PBF формат**

Файловият формат PBF се базира на Google Protocol Buffers. PBF файловете са много ефективни от пространството и по-бързи за използване от XML файловете. PBF файловете могат да съдържат нормални OSM данни или данни от историята на OSM.

PBF файловете с OSM данни съдържат съдържа изключително обемна и подробна информация - таблица с низове във всеки блок от данни, както и всички метаданни за обекти като идентификатор, идентификатор на набора за промени, потребителско име, брой на ленти, настилка, координати, сгради маркировка, повърхност и други.

* 1. **Съществуващи решения**

В силно конкурентния пазар за иновативни приложения на 2022 г., идеята за навигиране и изграждане на най-подходящ маршрут от дадена начална точка до крайна е не безизвестна и съществуват вече направени решения. От друга страна в частност изготвяне на велосипедни маршрути е с доста малка популярност и съществуващите приложения биват доста малко.

Едно от тези приложения е Google Maps. То е най-популярното и известно приложения за изготвяне на каквито и да е маршрути в глобален мащаб. Излиза на пазара през 2005 г. като бива развивано и до днешна дата. В Google Maps може да се изготвят различни маршрути в по-голямата част на света в зависимост дали потребителя желае да са пешеходни, велосипедни или предназначени за придвижване с автомобил. Опцията за изготвянето на велосипеден маршрут обаче не е достъпна за България.

Друга реализация, това е приложението – Moovit. То за пръв път се появява на пазара през 2013 г. като неговата цел е да предоставя на потребители маршрут, чрез който да стигнат от едно място до друго без автомобил. Moovit основно цели да предостави маршрут на потребителя, в който да участва придвижване с градския транспорт и ходене пеш, но дава и опцията да се създаде велосипеден маршрут. Към днешна дата приложението е достъпно за изтегляне в над 100 държави и има над 800 милиона ползватели.

OpenStreetMap от своя страна е друго решение, което няма изключително висока популярност, но въпреки това предоставя маршрутизиране за велосипедисти. До известна степен OpenStreetMap наподобява горе описаното приложение Google Maps, но с разликите, че OpenStreetMap предоставя изграждането на велосипедни маршрути и за България заедно с повече информация относно и дава възможност на потребителите си да променят информацията относно дадения регион с цел подобрения на точността на картата или добавяне на детайл, който преди това е липсвал. Както е написано в официалната документацията на OpenStreetMap е възможно в някои конкретни случаи информацията да бъде ненадеждна, поради причината, че информацията относно дадения регион има възможност да бъде неточна.

**Втора Глава**

**Проектиране на избраните технологии за реализация на iOS мобилно приложение**

* 1. **Основни функционални изисквания**

Основните изисквания към готовия продукт, водещи се по официално одобреното задание, са както следва:

* Изтегляне на OSM карта за страна/регион под формата на бинарен Protocol Buffer файл на устройството
* Извличане на нужните за велосипедно маршрутизиране данни от файла
* Построяване на граф от извлечените данни.
* Алгоритъм за навигиране и намиране на най-добрия път през графа
* Визуализиране на предложения маршрут върху рендерирана OSM карта в iOS приложение
  1. **Операционни системи**

Мобилните приложения в момента са абсолютна необходимост, в дигитален пейзаж приложенията за iOS и Android са привлекли огромната част от световната аудитория. Не е преувеличение да се смята, че потребителите на мобилни приложения по целия свят са разделени между iOS и Android. Никога обаче не трябва да се забравя, че разработването на приложения на двете операционни има своите различия:

* Една от основните разлики между iOS и Android е, че имат различни препоръчвани езици за изграждането на приложения като за Android е Kotlin или Java, а за iOS е Swift. Езикът не оказва голям натиск върху избора на операционна система, тъй като всеки един от изброените езици има огромен размер на активна общност и ресурси в интернет, нужни за научаването на езика и развитието на приложението.
* Когато става въпрос за интегрирана среда за разработка (IDE) на мобилни приложения, iOS и Android използват различни инструменти. Android на Google предоставя многофункционално Android Studio. От друга страна, XCode на Apple е IDE, което е доказало своята последователност през годините, като помага на разработчиците на iOS да извлекат най-доброто от често актуализираните версии на операционната система.
* Приложения за iOS са склонни да генерират повече приходи от инвестициите си. Те получават редовен поток от приходи за разлика от приложенията за Android. Това се дължи главно на това, че потребителите на iOS устройства най-вече принадлежат към групата, чийто жизнен стандарт се определя от силната икономика.

Въпреки че има постоянна интензивна надпревара между приложенията за iOS и Android, изборът за операционна система както е записано и в заданието за дипломната работа ще бъде iOS, с оглед на факта, че към 2022 г. множеството от мобилните приложения се пишат на iOS като те имат само допълнителни приложения, които наподобяват тези на iOS, които са на Android.

* 1. **Езици за програмиране**

Когато става дума за подбор на език, който да бъде използван в дадено приложение за iOS винаги първите езици, които биват разглеждани са Objective-C и Swift. Objective-C е създаден от Брад Кокс и Том Лав през 1984 г. и в продължение на години е основният език за писане на iOS приложения. Objective-C е полезен, обектно ориентиран език, който надгражда езика C. Тъй като той обаче е на 30 години, езикът не отговаря на настоящите нужди и, когато от Apple представят езика Swift, той става предпочитан и основен език за писане на iOS приложения.

Друг език, който набира популярност в изграждането на мобилни приложения е Flutter, разработен от Google. Flutter се използва за изграждане на междуплатформени приложения. Flutter използва езикът Dart и единна кодова база за създаване на приложение за множество платформи, включително Android и iOS. Swift обаче е много по-добър от Flutter в доста отношения:

* Популярността на Swift е значително по-голяма от тази на Flutter
* Swift е препоръчвания език за писане на iOS приложения
* За писането на Swift се изискват Xcode + macOS, а за писане на Flutter се изисква и още една интегрирана среда за разработка освен вече споменатите Xcode + macOS
* Swift е по-бърз, когато става дума времето за изграждане на файла на приложението

Поради тези причини езикът избран за реализацията на дипломната работа е Swift.

* 1. **Технология за изграждане на потребителски интерфейс**

Когато става дума за разработка на iOS потребителски интерфейс има разгорещен дебат между двете основни технологии SwiftUI и UIKit. И двете от тези рамки предлагат различни предимства, като в същото време всяка има свой собствен набор от недостатъци. Изводът както за UIKit, така и за SwiftUI е, че Apple са разработили и двете рамки, за да позволят на разработчиците да създават мощни приложения.

* + 1. **UIKit**

Предшественикът на SwiftUI, UIKit е една от причините за възхода на езика за програмиране на Apple, Swift. С UIKit възможността за изграждане на потребителски интерфейси е възможна дори без силен опит в програмирането. Това се дължи на въвеждането на сторибордове, решение за плъзгане и пускане за изграждане на потребителски интерфейси.

Това обаче идва с различни недостатъци: изграждането на потребителски интерфейс програмно (без сторибордове) в UIKit е значително по-трудно в сравнение с SwiftUI. Не само това, но UIKit е известен като императивна рамка, което означава, че програмиста заявява как да се направи нещо. За разлика от SwiftUI, който е декларативна рамка, което означава, че програмиста декларира нещо, което иска да се случи. Това означава, че начинът, по който е структуриран кодът, различен между SwiftUI и UIKit.

Последната важна подробност за UIKit е, че това е по-стара рамка, което означава, че с течение на времето може да започне да се наблюдава по-малко подкрепа от Apple за UIKit, тъй като SwiftUI продължава да се развива.

* + 1. **SwiftUI**

SwiftUI е доказан като абсолютна промяна на играта в света на разработката на iOS поради факта, че SwiftUI предлага синтаксис, който е лесен за разбиране и е структуриран по начин, който има смисъл. Не само това, но SwiftUI е много по-мощна библиотека. SwiftUI само ще продължи да се развива като мощен език, тъй като Apple продължава да измества фокуса си от UIKit.

SwiftUI е влез всичко, което Apple са научили от UIKit, и предоставя на разработчиците много по-добри функции, които не са налични в UIKit. Една от оптимизациите на SwiftUI е, че се използва по-малко код за създаването нещо сравнение с UIKit. Изграждането на потребителски интерфейси програмно е норма за SwiftUI. С SwiftUI се отделя повече внимание на програмното изграждане на потребителски интерфейси. Въпреки че SwiftUI е превъзходна рамка за изграждане на iOS приложения и продължава да се развива масово в бранша все още се използва UIKit и новите приложения тепърва започват да се пишат на SwiftUI.

С оглед на казаното до тук за целите на дипломната работа е избран SwiftUI за изграждане на потребителския интерфейс на приложението.

* 1. **Избор на интегрирана среда за разработка (IDE)**

Когато става дума за реализация на iOS приложения двете основни интегрирани среди за разработка са: AppCode и Xcode.

* AppCode е IDE, което е разработка на JetBrains като е изградено на платформата на IntelliJ IDEA, което също е собственост на JetBrains. Първата публична версия на AppCode става достъпна през 2011 г. и дава възможност за разработка на приложения посредством Swift, Objective-C, C, C++, JavaScript.
* Xcode е IDE от Apple за macOS и се използва за разработка на софтуерни приложения за macOS, iOS , iPadOS , watchOS и tvOS. Първоначално Xcode е пуснат в края на 2003 г. като най-новата стабилна версия е версия 13.3, пусната на 14 март 2022 г. и е достъпна чрез Mac App Store безплатно за потребителите на macOS.

Xcode е изключително продуктивна среда за създаване на приложения за всички устройства издадени от Apple и основното му предимство спрямо неговия аналог от JetBrains – AppCode е, че няма такси, които са нужни да бъдат заплатени за използването му, както и факта, че Apple препоръчват собственото си IDE за разработка на iOS приложения. Поради тези причини за нуждите на дипломната работа е избрано предложението от Apple – Xcode.

* 1. **Избор на формат на OSM файл**

В много от случаите не може да се избере кой формат да се използва в даденото приложение, защото се получава от някъде файл в определен формат и трябва да се работи с него. В случая тази пречка не съществува и формата на изтегления файл може да бъде свободно избран.

Основните два формата на файл, които биват разглеждани са PBF и XML. За целите на дипломната работа избрания формат е PBF, поради следните причини:

* PBF файловете са много ефективни от гледна точка на пространство
* PBF файловете са по-бързи за четене и писане отколкото XML файловете
* В последните години се наблюдава тенденция да се преминава на колкото се може повече места от XML на PBF файлове, както и в началото на проекти да се предпочита започването на работата с PBF файлове
  1. **Алгоритми за намиране на най-кратък път в граф**

Когато става дума за изготвяне на маршрут от дадена начална до дадена крайна точка се използва някакъв вид алгоритъм, чрез който се намира най-подходящият път. За да се използва някакъв алгоритъм, обаче, трябва да има данни, през които да се търси този маршрут. Данните биват структурирани в структурата от данни - граф. Като графа притежава върхове и ребра като ребрата е възможно да имат някаква тежест – разстояние, натовареност и други, а върховете представляват точките, от които е възможно да се избере начална или крайна дестинация.

Алгоритмите за намиране на най-добър маршрут, които ще бъдат разгледани са Dijkstra и A\*:

* + 1. **Dijkstra**

Алгоритъмът на Dijkstra е алгоритъм, който намира най-краткия път между връх A и връх B в граф с неотрицателни тегла на ребрата. Накратко, това става като се намират най-кратките пътища от зададения точка в началото – връх А до всички други върхове, които ще включват и търсения - връх B. Това става като се изпълняват следните стъпки:

* Първо се създава масив, който да съдържа всички вече обходени върхове, който служи, за да се следи дали на всеки един вече обходен връх му е присвоен правилният най-кратък път до началния връх А. Този списък е нужен, за да не се минава отново през върхове, чийто най-кратък път е вече зададен.
* Докато се стигне до крайния връх Б се изпълнява следното:
  + Избира се връх, чието текущо разстояние до началния връх А е най-кратко и бива посетен, при което се добавя при масива с посещаваните ъгли.
  + Актуализират се разстоянията за всички съседни върхове на избрания, които все още не са посетени. Това се прави като се сравнява текущото му най-кратко разстояние на съседа със сумата на разстоянието на избрания събрано с разстоянието между избрания и неговия съсед, за който се прави проверката
* Когато се стигне до търсения връх В, алгоритъма е приключил и най-краткия записан маршрут е намерен правилно
  + 1. **A\***

Алгоритъмът A\* се основава на евристика за навигация в търсенето, но за разлика от много подобни алгоритми с тази база, той е едновременно пълен и при определени условия оптимален:

* Пълният алгоритъм е алгоритъм, който гарантира правилен отговор за всеки правилен вход, ако този отговор съществува
* Оптималният алгоритъм е алгоритъм, който връща отговор за възможно най-кратък период от време. Оптималността на този алгоритъм е силно зависима от качеството на неговата функция за оценка.

За алгоритъма A\*, функцията за оценка има следната форма:

 Където g(n) представлява най-краткия път от началния връх до връх n и h(n) представлява евристичното приближение на стойността на върха n.

В зависимост от проблема, който трябва да бъде решен, могат да се използват различни евристични приближения, за да се постигне оптималност. Изборът на качествена евристика е една от най-важните стъпки, когато става въпрос за прилагане на този алгоритъм.

При изпълнение на алгоритъм A\* се изпълняват следните стъпки:

* За да бъдат избегнати безкрайни цикли и да се гарантира пълнота и сигурност се изграждат два списъка:
  + Списък със затворени върхове - списък, в който се съхраняват посетените върхове, чиито съседи също са посетени
  + Списък с отворени върхове - списък, в който съхраняваме посетените върхове, чиито съседи не са непременно посетени

В началото списъкът с отворени върхове съдържа само началния връх, докато списъкът със затворени върхове е празен.

* Избира се връх n с най-добра стойност от f(n). Ако върха n е търсения краен връх, алгоритъма е готов. Ако не, ще се разгледат преките му съседи.
* За всеки съсед m на върха n се проверява дали е в един от двата списъка. Ако не, той бива поставен в списъка с отворени върхове. Върха n се маркира като родител на m. След това се изчисляват g(m) и f(m). Въпреки това, ако съседът е в един от двата списъка, се проверява дали пътят от началното връх до върха m посредством връх n е по-къс от текущия съществуващ път до m. Ако това е вярно се актуализират g(m) и f(m). Ако връх m е бил в затворения списък преди, той се поставя в отворения списък.
* Накрая се поставя текущия връх в списъка със затворени върхове.
* Докато има елементи в списъка с отворени върхове, се повтарят предишните три стъпки.
* Ако не се стигне до крайния търсен връх, но нашият списък с отворени върхове е празен, пътят към крайния връх не съществува

С оглед на описаните два алгоритъма горе, изборът за алгоритъм използван в реализацията на дипломния проект е алгоритъма Dijkstra, поради причината, че алгоритъма A\* изисква имплементация на функция за оценка.

* 1. **Визуализиране на карта и маршрут**

Най-стабилните и гъвкави опции за визуализиране на карта са Mapbox, Google Maps и OpenStreetMap API.

* + 1. **OpenStreetMap API**

OpenStreetMap е проект, задвижван от общност, която предоставя на множество уебсайтове и приложения своите картографски данни. Тъй като е карта с отворен код, тя е напълно безплатна за използване, но поддържа високо ниво на точност и детайлност благодарение на усилията на местните ентусиасти и инженери, които я попълват с данни и я поддържат.

OpenStreetMap API има своите плюсове, но и своите недостатъци. Като най-големия плюс на OpenStreetMap API е, че изцяло безплатен, но е разработен с цел актуализиране и редактиране на карти, което го превръща в доста неудобен вариант за работа. Въпреки това OpenStreetMap API предоставя много добра база, че дори компании като Mapbox използват OSM като гръбнак за своите карти.

* + 1. **Google Maps**

Google Maps притежава най-голямо присъствие между платформи и устройства в сравнение с други решения за картографиране. Google Maps е изключително лесно разпознаваем, което се дължи на факта, че повечето потребители са се сблъсквали и са запознати с интерфейса на Google Maps. Това вероятно ще създаде присъщо доверие при взаимодействие с Google Maps, ако се вгради в даденото приложение. Още един от плюсовете на Google Maps е, че притежава изключително високо качество на данните като през годините то само се подобрява. Един от основните недостатъци е, че цената, която трябва да бъде заплатена за използването на Google Maps и продължава да расте непредвидимо през години. В началото използването на Google Maps е безплатно, но тавана на безплатните заявки, които могат да се извършат е много нисък.

* + 1. **Mapbox**

Основана през 2010 г., за да предостави на различни организации картографски данни и анализи, Mapbox се превръща в пълноправен индустриален гигант. Mapbox предоставя персонализирани инструменти за карти и данни за местоположение на големи компании като Facebook, Snapchat и Foursquare и допринася за множество библиотеки и приложения за картографиране. Mapbox предлага изчерпателен набор от функции и инструменти за интегриране на своите картографски услуги във всякакви различни уебсайтове и мобилни приложение.

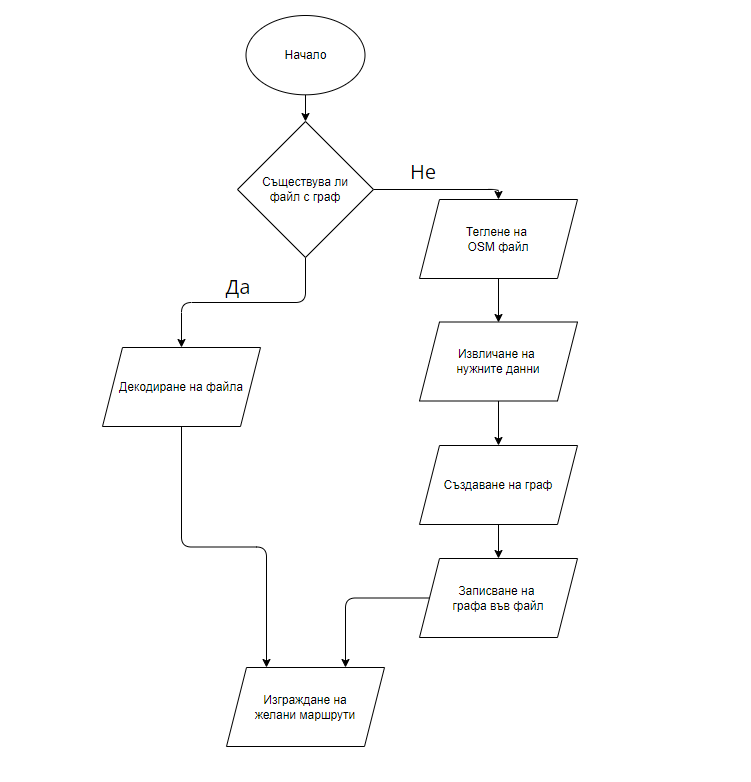
Mapbox притежава следните плюсове:

* Когато става въпрос за дизайн, персонализиране и гъвкавост Mapbox изпреварва всички негови конкуренти като предоставя промяна на цветове и стилове по лесен и бърз начин.
* При наличие на по-големи масиви от данни Mapbox е изключително бърз при зареждането им и притежава страхотна производителност. Картите предоставени от компанията зареждат бързо и се визуализират гладко, което е от голямо значение, когато се работи със сложни набори от данни.
* Mapbox пуска своя код и насърчава общността да го проверява и подобрява. Друга скрита полза се крие в използването на Mapbox библиотеки, които позволяват вмъкване на персонализирани интерактивни карти в родни приложения на iOS и Android.
* Mapbox не изисква плащания, за да бъде използвано до един определен брой завки, който сравнение с Google Maps е доста по-голям и труден за достигане.

Mapbox има и своите недостатъци, които са свързани в това, че има по-ниско покритие на картата в определени региони, което се дължи на факта, че Mapbox разчита на колективно картографиране, като OpenStreetMap, който основният му източник на данни и това носи недостатъците на OSM със себе си. Въпреки това Mapbox е избран за визуализиране на карта поради горе описаните причини и факта, че е лесен и бърз за интеграция в iOS приложения.

* 1. **Случващото се при стартиране на проекта**

При стартиране на приложението е възможно да се случат едно от двете неща показани на фиг. 2.2, според зависи дали това е първия път, в който се пуска приложението на даденото устройство. Това е направено с цел да се пропусне цялото теглене на големия файл с информацията за целия регион или държава и извличането на данни от него при всяко пускане на приложението, поради причината, че това изисква доста по-голямо количество време от колкото само да се декодира файл със нужната информация за създаването на граф, чрез който да бъдат изграждани маршрути.

****

****

Както е показано на фиг. 2.2 се проверява дали съществува вече файл с подходящата информация за създаването на нужния граф или той все още не е бил създаден. Съответно ако този файл не съществува се тегли OSM файл за дадена държава или регион, извлича се нужната информация от него създава се граф и се записва в файл на устройството. Ако пък съществува се декодира файла и се продължава към изграждането на маршрути.

**Трета Глава**

**Реализация на програмния продукт. Описание на начина на реализация**

На всички места в кода на дипломния проект, в които присъстват следните редове код, показани от долу на фиг. 3.1, трябва да се има предвид, че те целят, единствено и само да покажат колко време отнема, за да се изпълни дадена част от кода.

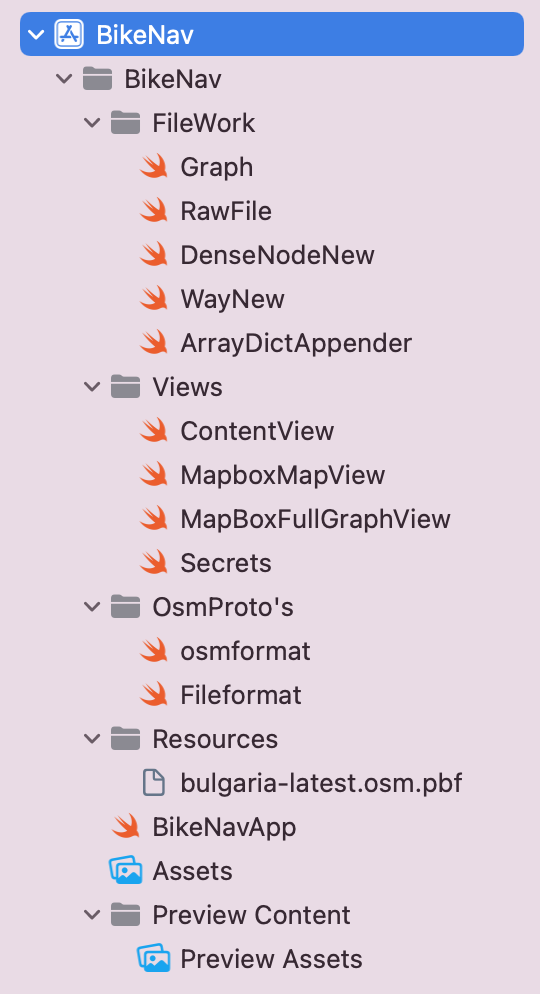




* 1. **Файлова структура на проекта**

Проекта е разделен на няколко групи или папки както е показано на фиг. 3.2. Групите целят различните файлове с код да бъдат подреден по удобен за четене и писане за разработчика начин. В случая са както следва:

* FileWork - групата, в която са разположени всички файлове с код свързани с извършването на какъвто и да е тип работа с файла и с графа, включително и алгоритъма за намиране на най-подходящ маршрут.
* Views - групата, в която са поставени всички файлове, които имат общо с екраните на приложението, тоест с показването на карата и маршрута
* OsmProto’s – групата, в която са разположени генерираните файлове за работа с файлове от OpenStreetMap типа файлове с данни с формат PBF. Те съдържат структури с наименования – „OSMPBF\_BlobHeader“, „OSMPBF\_HeaderBlock“, „OSMPBF\_PrimitiveBlock“, „OSMPBF\_Blob“, които служат единствено и само за отварянето и прочитането на файла с информацията за конкретния регион или държава
* Resources – тази група цели да съдържа всички файлове с данните за България като към този момент този файлът, който е в групата не се използва тъй като неговото теглене става автоматично при нужда и единственото предназначение на файла е ако има съмнение, че файла не е изтеглен правилно или подбраната информация от него не е както трябва, приложението да бъде пуснато и във варианта, при който тегленето не е автоматично и да се види дали наистина има някакъв проблем.

****

****

* 1. **Извършващите се проверки при пускане на приложението и случващото след тях**

При началото на пускане на приложението, както е споменато и във втора глава се извършва проверка дали файла, съдържащ информацията нужна за графа, е запазен. Това се случва във функцията launch, която е част от класа RawFile. Съответно ако файла го има той бива декодиран и върнат като обект от типа - Graph. Ако файла не съществува обаче, започва тегленето на големия OSM файл, след което биват извиквани следващите функции, които трябват да могат да бъдат изпълнени успешно преди да може да се създаде графа и информацията за него да може да бъде записана във файл на съответното устройство под формата на JavaScript Object Notation (JSON) файл. След като това стане графа бива върнат като обект от типа – Graph. Написаното описва случващото се във функцията на фиг. 3.3.







* 1. **Изтегляне на файла**

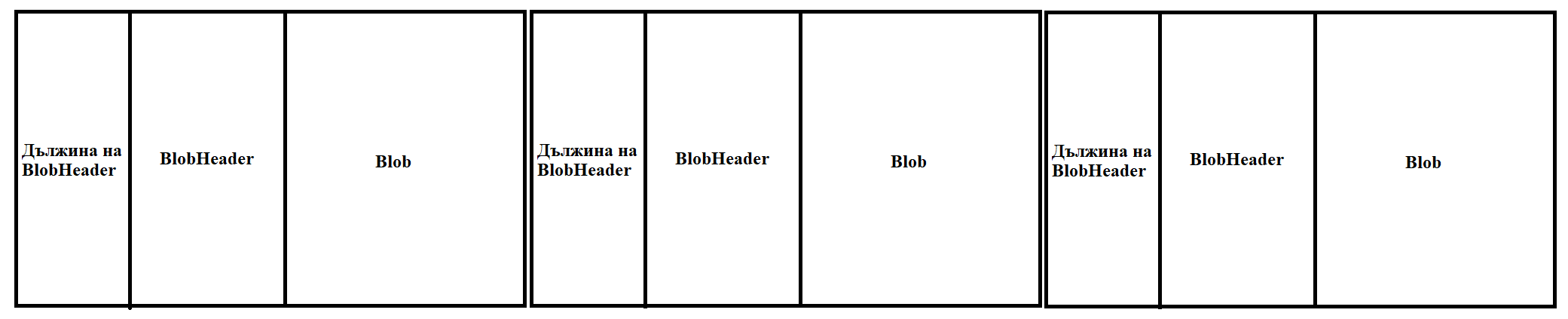
На фиг. 3.4 е показан кода, който служи за изтеглянето на информацията от последния възможен OSM файл за България. Това става така, защото винаги последния качен OSM файл за даден регион или страна завършва на „-latest.osm.pbf“, след което като се изтегли информацията тя бива върната, за да може да се използва при отварянето и четенето и.

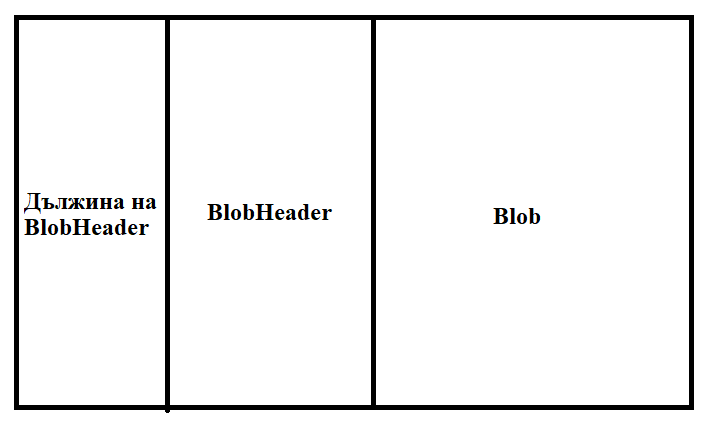
********

* 1. **Извличане на нужната информация**

След като цялата информация е изтеглена от файла по някакъв начин следва от нея да се извади нужната информация, за да се направи това обаче, трябва да се разбере по какъв начин е структуриран файла. Той съдържа образно казано блокове от информация, които са много на брой. Това е показано на фиг. 3.5. Съдържанието на един блок, както е показано на фиг. 3.6 е разделено на три части, които са:

* Дължина на BlobHeader – число с големина 4 байта
* Blob Header – съдържа типа данни на следващия Blob и сериализирания размер на следващия Blob. Типа данни е възможен да бъде само „OSMHeader“ или „OSMData“. В случай, че типа е „OSMData“, данните могат да бъдат компресирани или не. В повечето случаи данните са компресирани в формат „zlib“, но е възможно и да бъдат в друг формат. След като данните са разкомпресирани се появяват обектите от типа PrimitiveBlock, които съдържат съществените картографски данни. В случай, че типа е „OSMHeader“ следващия Blob съдържа обект от типа HeaderBlock, който съдържа информация за границите на координатите на всички данни в следващите обектите от типа „PrimitiveBlock“.
* Blob – съдържа данните от зададения тип в Blob Header.

Съдържанието на файла е такова, че винаги започва с дължината на първия „Blob Header“ и задължително приключва с „Blob“. 





Целия този процес по извличане на обектите от типа PrimitiveBlock се извършва от следния фрагмент код показан на фиг. 3.7. Взима се дължината на BlobHeader, след което се пресмята големината на BlobHeader, от който се взима дължината на Blob и съответно се пресмята и неговата големина. След което като вече е наличен даден Blob, ако в предходния BlobHeader типа на данните е бил „OSMData“, от него се декомпресира информацията за да може да се достигне до обектите от тип PrimitiveBlock, които да бъдат поставени в един общ масив, с който да се работи след това. Това е направено така поради причината, че файла предоставен от OSM е подреден по неподходящ начин за текущите цели.

Специфичното при декомпресирането на информацията на данни в „zlib“ формат е, че трябва да се премахнат първите два байта, за може да бъдат декомпресирани успешно от предоставената от Apple функция - decompress(using: ).







Това, което се забелязва във функцията readFile , която е част от класа RawFile и представена на фиг. 3.8 е, че се използва група за изпращане (DispatchGroup), която позволява група от задачи да се наблюдават като едно цяло и прикачвайки множество работни елементи към една група, те биват планирани за асинхронно изпълнение на една и съща опашка или на различни опашки. В текущия случай има две опашки, които са инициализирани по следния начин показан на фиг. 3.8.





Опашката serialSyncQueue служи, за да синхронизира асинхронната работа върху общо манипулирани данни, докато опашката parallelProcessingQueue служи за асинхронното извършване на задачи.

Начина, по който всеки път се взима дължината на следващия BlobHeader е чрез структурата Parser показан на фиг. 3.9. Структурата притежава методи, които връщат число от целочислен тип с дадена големина варираща между 8, 16, 32 и 64 бита като единственото, от което има нужда са данните, от които да предоставя числото, което представлява член променливата data както и къде в данните да пробва да вземе числото, което представлява брояч, който е член променливата с име offset.







Всеки един обект от тип „PrimitiveBlock“ съдържа в себе по една таблица с низове, масив с обекти от тип „PrimitiveGroup“ заедно с granularity, lat\_offset, lon\_offset, които отговарят за координати на всеки един елемент в текущия обект от тип „PrimitiveBlock“ и се използват във формулата показана на фигура 3.10.





Променливите latitude и longitude са реалните координати на дадения елемент, докато променливите lat и lon са стойностите записани във всеки един елемент, с които трябва да се замести в уравненията на фиг 3.10, за да се получат реалните координати за дадения елемент.

Това, което се съдържа във всеки един обект от тип PrimitiveGroup е задължително едно от следните няколко варианта – обект от тип DenseNodes, масив с тип Node, масив от тип Way, масив от тип Relation, масив от тип ChangeSet, като за целите на дипломната работа единствените два варианта, които имат значение са следните - този с един обект от тип DenseNodes и този с масив от тип Way.

След като масива с обектите от типа PrimitiveBlock е запълнен с информацията от целия файл, от тези обекти трябва да се извлече само нужната информация. За да се случи това обаче трябва масива с обектите от типа PrimitiveBlock да бъде преструктуриран на други два масива, които съдържат само тези обекти от тип PrimitiveBlock, чийто обект от масив от тип PrimitiveGroup съдържа само обект от тип DenseNodes или само масив от тип Way. Това преструктуриране се случва във фрагмента от код показа на фиг. 3.11. Целите на това преструктуриране са да може да бъде взета първо информацията за данните от тип Way. Подредбата на цялата информация във файла е неудобна като първо са сложените данните от тип DenseNodes, а след тях от тип Way, което не позволява да бъдат взети само нужните данни от тип DenseNodes, които участват в данните от тип Way. Ако бъдат взети всички данни от всички обекти DenseNodes тяхната бройка става прекалено голяма като отнема часове за едно обхождане на всички данни.





Това, което се забелязва във функцията handlePrimitiveBlocks, която е част от класа RawFile е, че един обект от тип PrimitiveBlock е възможно да бъде поставен и в двата нови масива. Това е така, защото в официалната документация на OSM за PBF файловете не е споменато дали може различните обекти от PrimitiveGroup в един обект от PrimitiveBlock могат да съдържат различни типове данни. След като новите масиви са напълнени, стария бива изчистен.

След като вече са налични двата масива трябва да се извлече и запише информацията за обектите от тип Way, за целта е създадена нова структура WayNew, показана на фиг 3.12. Структура е направена като в нея се съдържат само нужните за целите на дипломния проект неща от тип Way като е добавена и дължина, която се пресмята на по-късен етап.





Начина, по който се извлича информацията от масива с обектите от типа Way е демонстриран на фрагмента с код на фиг. 3.13.







Функцията parseWaysFromFile, която е част от класа RawFile показана на фиг. 3.13 работи на всички възможни ядра на даденото устройства. Това се прави по този начин, поради причината, че информацията е в изключително голямо количество. След като се определи по каква част от масива с всички обекти от тип Way ще работи всяко едно ядро се обхожда масива като всеки един елемент от тип PrimitiveGroup се предава на функцията handleWays показана на фиг. 3.14, която връща обект от тип ArrayDictAppender показан на фиг. 3.15. След като новия масив - ways и речник - references са запълнени, предходния масив, който съдържа всички обекти с тип на данни Way е освободен. Масива ways служи, за да съхранява всички пътища, докато речника – references служи, за да съхранява общите възлови точки на различните пътища и съответно кои са тези пътища.









Интересното във функцията handleWays, която е част от класа RawFile и е показана на фиг. 3.15 е, че, за да може да бъдат взети идентификаторите на възловите точки на даден път те трябват да бъдат разкодирани, за което помага простата структура на фиг. 3.17. Начина, по който те са кодирани е наречен „delta“, което означава, че стойността на дадената променлива е вярна само за първия елемент, а за всеки следващ се добавя записаното число от файла с реалната стойност на предишния елемент.





Друго важно нещо за функцията handleWays, показана на фиг. 3.15 е, че тя взима само и единствено обекти от тип Way, които реално са пътища като това се разбира ако за съответния обект е написано, че е „highway“ всички други обекти от тип Way могат да описват сгради, граници, квартали и други. Елементите от тип Way, които не притежават име или референция към текущия момент се препускат, както и тези, които все още са в строеж.

След като е налична извлечената информация за пътищата, трябва да се извлече и информацията за възловите точки, които са ключове на речника references. Това се изпълнява от функцията parseNodes, която е част от класа RawFile и е показана на фиг. 3.19.







Функцията parseNodes, която е показана на фиг. 3.19 работи на всички възможни ядра на даденото устройства. Това се прави по този начин, поради причината, че информацията за всички обекти от тип DenseNodes е в изключително голямо количество. След като се определи по каква част от масива с всички обекти от тип DenseNodes ще работи всяко едно ядро се обхожда масива като всеки един елемент от тип PrimitiveGroup се предава на функцията handleNodes показана на фиг. 3.20, която връща масив от тип DenseNodeNew, показан на фиг. 3.21. След като новия масив – nodes е запълнен, предходния масив, който съдържа всички обекти с тип на данни DenseNodes е освободен. Масива nodes служи, за да съхранява всички възлови точки.







Функцията handleNodes показана на фиг. 3.20 отново използва структурата DeltaDecoder, показана на фиг. 3.18, за да разкодира „delta“ кодираните данни id, lat и lon от OSM типа DenseNodes и добавя само тези възлови точки в масива, които са ключове в речника references. Закоментираната част от функцията служи, за да извлече и информацията от обекта с име keysVals на OSM типа DenseNodes, но поради причината, че изпълнението на този фрагмент код изисква доста време, за да бъде изпълнен, а информацията от keysVals не се използва, фрагментът е закоментиран.

* 1. **Преработване на извлечената информация и създаването на граф**

След като вече има извлечена цялата нужна информация, тя трябва да бъде преработени така, че да е подходяща да може да се създаде граф. Преработването на информацията се дължи на следния проблем. Информацията за пътищата не е записана по удобен начин във файла предоставен от OSM, във файла един път не представлява от самото му начала до края му, а само една откъсната част от него и тези части като се съберат се получава целия път. За да станат данните подходящи за създаването на граф трябва да се мине през няколко стъпки като първата от тях е да се намерят всички кръстовища, защото върховете на графа ще представляват кръстовища, а ребрата пътищата между кръстовищата. Това извличане на кръстовищата става на фрагмента от код показан на фиг. 3.21.







Фрагментът код показан на фиг. 3.21 е част от функцията reduceMap, която е част от класа RawFile и работи на всички ядра на даденото устройство като намира всяко едно кръстовище, след което намалява значително количеството на ключове в речника references. Начина, по който намира кръстовищата е посредством обект от тип Set.

След като се знае точната бройка на кръстовищата и кои са те трябва да се намери на всяко едно кръстовище неговите съседни и дължината до тях. Това се прави от фрагмента код показа на фиг. 3.22





Фрагментът показан на фиг. 3.22 е част функцията reduceMap, която е част от класа RawFile и работи на всички ядра на даденото устройство, както този фрагмент от същата функция описан преди него. Фрагментът използва структура показана на фиг. 3.23 и следните два псевдонима показани на фиг. 3.24.









Фрагментът от фиг. 3.22 създава нов речник, който накрая бива предаден за създаването на графа, тоест обект от тип Graph, който съдържа абсолютно същия речник като този създаден във функцията.

Функцията обхожда всяко едно кръстовище като го предава на следващата функция nextCrossroads показана на фиг. 3.25, която връща масив съдържащ началното кръстовище с неговото съседно и дължината между тях.









Функцията nextCrossroads, която е част класа RawFile и е показана на фиг. 3.25 използва други три функция, за да вземе дължината до следващото кръстовище и самото съседно кръстовище, като има три варианта ако сегашното кръстовище е в началото на някой откъснат път взет от OSM данните, в края или в средата. Във функцията nextCrossroads се забелязва, че е имплементирано бинарно търсене, което е необходимо поради причината, че се работи с голямо количество от данни и, за да бъдат обходени тези данни се изисква много голямо количество време. Бинарното търсене е имплементирано в фрагмента код показан на фиг. 3.26 като, за да бъде използвано то, колекциите, за които се използва биват сортирани.





Функциите inBeginning, inEnd, inMiddle са доста идентични като са част от класа RawFile, поради което е показана само една от тези функции, а именно inBeginning на фиг. 3.27. Разликата е, че трите покриват различни случаи и функцията inMiddle връща два резултата, защото обхожда в две посоки, докато другите връщат по един резултат, защото обхождат в една посока.







Функцията от фиг. 3.27 пресмята дължината на разстоянието между две съседни кръстовище и намира второто, като това го прави даже ако се наложи да мине на следващата част от даден път посредством рекурсивната функция firstCrossroad, показана на фиг. 3.28. Функцията firstCrossroad се извиква само тогава, когато следващото кръстовище не е на същия път.









Функцията firstCrossroad продължава да смята дистанцията, докато намери следващо кръстовище или не стигне до положение, в което следващо кръстовище няма и в този случай връща последната възлова точка на последния път, до който е стигнато.

* 1. **Взимане на началната и крайна точка за изготвяне на маршрут**

След като има готов граф, от който да се правят маршрути, и заредена карта чрез MapBox началната и крайната дестинация се взимат посредством натискания върху екрана. Това се извършва във функцията показана на фиг. 3.30.





След като са взети точките, които са натиснати трябва да се намери до кои кръстовища са най-близко като става във фрагмента код на фиг. 3.31.





Фрагментът от код на фиг. 3.31 е част от функцията findNprintRoute, която е част от класа Coordinator.

* 1. **Намиране на най-подходящ път**

След като има избрани начална и крайна дестинация от наличните кръстовища в графа те биват предавани на функцията findRoad, която е част от структурата Graph, и е показана на фиг. 3.32.







Функцията findRoad имплементира алгоритъма Dijkstra като има добавена приоритетна опашка, която служи всеки път да се взима най-късия маршрут, до който е стигнато, докато не се стигне до търсената дестинация или не бъдат обходени всички възможни кръстовища, тоест да са направени всички маршрути и желаната дестинация да не бъде включена в нито от маршрутите.

Функцията използва класа PathSegment, показан на фиг. 3.33, за да връща намерените маршрути.





* 1. **Изобразяване на намерения маршрут**

Накрая след като има наличен маршрут за желаната начална и крайна точка, той бива изобразен посредством фрагмента от код показан на фиг. 3.34, където се използват червени точки, които репрезентират кръстовищата и зелени линии, които репрезентират пътищата между кръстовищата.







Фрагментът от код, показан на фиг. 3.34 е част от функцията findNprintRoute, която е част от класа Coordinator и взима всички кръстовища от маршрута посредством върнатия обект от тип PathSegment, след което на всяко едно кръстовища слага червена точка и чертае зелена линия между съседните кръстовища.

**Четвърта Глава**

**Ръководство на потребителя**

* 1. **Инсталация**

Вариантите за инсталация на приложението са два посредством TestFlight, предоставен от Apple и посредством GitHub releases. Приложението е налично за сваляне през TestFlight, докато не бъде одобрено от Apple за публикуване в App Store. Когато приложението се цели да бъде изтеглено от GitHub releases, трябва да бъдат изпълнени следните стъпки:

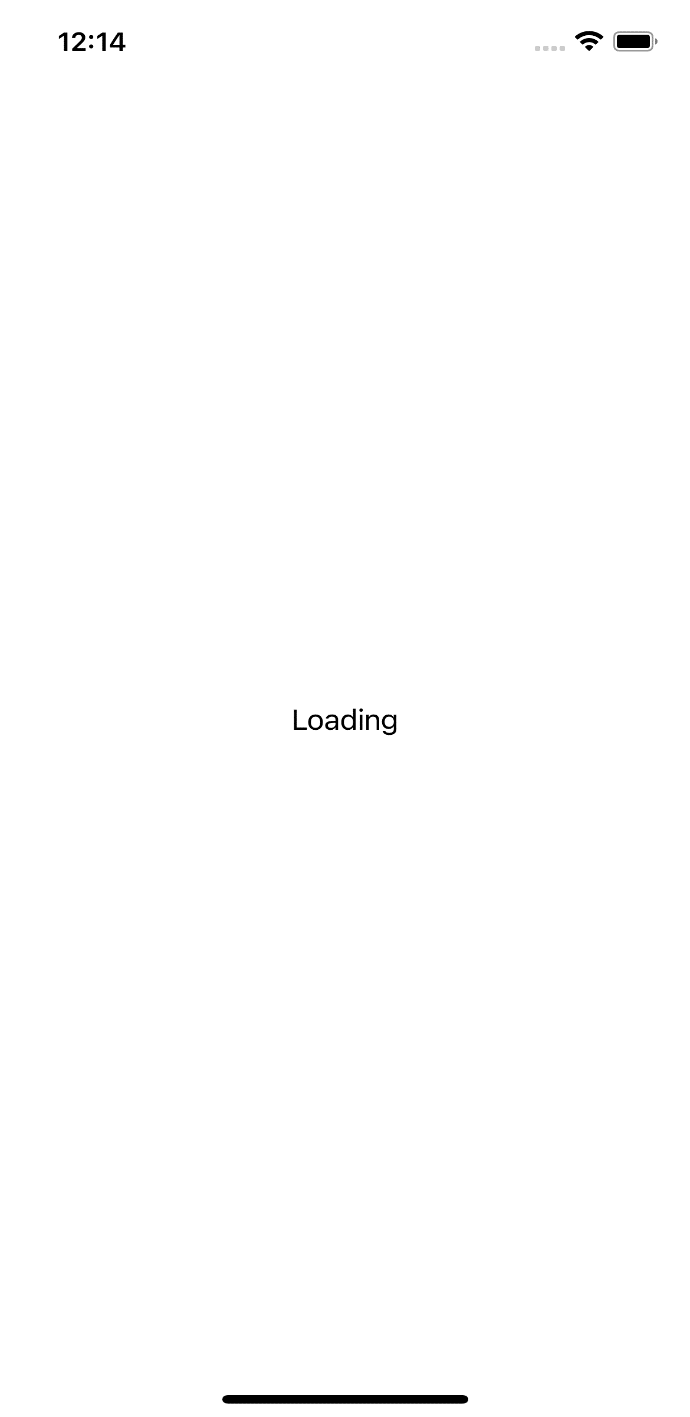
* Отива се на секцията „releases” на хранилището на приложението в GitHub:

https://github.com/NikolayDinkov/BikeNav/releases

* Изтегля се „.ipa“ файлът на мобилно устройство.
* След, което изтегления файл се поставя в друг сайт или програма, който да позволи изтеглянето на телефона като примерен сайт е следния - https://www.installonair.com/
* Изтегля се файла от сайта, след което се пуска на телефона
  1. **Резюме на главния екран**

Приложението притежава един екран, на който е изобразена картата, показаното на фиг. 4.2, и има възможност да се натиска като това избира съответно начална точка и крайна точка на желана дестинация, след което като се намери маршрут той бива изобразен. Това е показано на фиг. 4.3.

Във варианта, когато приложението е пуснато за първи път и все още няма наличен файл с граф, който да се отвори бързо се изчаква като се показва следния екран показан на фиг. 4.1.

 Картина, която съдържа карта

Описанието е генерирано автоматично



Картина, която съдържа карта

Описанието е генерирано автоматично



* 1. **Ръководство за отваряне и редактиране на проекта**

За да се позволи редакция на iOS приложението е нужно инсталирането на Xcode, за което е необходимо притежанието на компютър с операционна система – macOS. След като Xcode бъде успешно инсталиран, трябва да се клонира хранилището на дипломния проект или да се изтегли неговия „.zip” файл. Това става от следния линк:

* <https://github.com/NikolayDinkov/BikeNav>

Като на фиг. 4.4 е показано точното място от къде да бъде свален „.zip” файла или да бъде клонирано.

Картина, която съдържа текст, екранна снимка, компютър, закрито

Описанието е генерирано автоматично



**Заключение**

Всяко едно от зададените изисквания на дипломната работа е изпълнено. Въпреки това, приложението има места, в които може да се подобри съизмеримо. Но все пак спазването на правилен процес за разработка на мобилно приложение за iOS, притежаващо възможността да работи на всички ядра на устройството, на което е пуснато, както и реализацията на достатъчно бърз и ефективен алгоритъм за намиране на най-кратък път са само част от реализираните елементи, които демонстрират, че голяма част от нужните изисквания за дълготрайната реализация на едно успешно приложение се откриват и в този проект. Използват се идеи, имплементации, библиотеки и методи за създаването на приложение водено от практиките на модерната разработка на iOS приложения.

Проекта има елементи, които могат да се подобрят, за да се доусъвършенства приложението като това да се замени JSON файла, който се записва на устройството с PBF файл, това да се подобри потребителския интерфейс, когато потребителя трябва да бъде навигиран до желаната от него крайна дестинация като може да се добави текущата ми локации, това да се разшири обхвата от държави и региони на приложението от България на целия свят или да се предостави избор на регион или държава. Но освен това, конкретните цели и някои други малки функционалности, които не са изрично посочени в изискванията за дипломния проект са изпълнени, с цел направата на едно стабилно чисто, функционално и сравнително добро iOS приложение за изготвяне на велосипедни маршрути. То може да бъде определено като прототип или бета версия на едно същинско конкурентно маршрутизиращо приложение, което, с някои малки архитектурни и други условни редакции, може да бъде използвано от голям брой хора като запълни дупката, в която липсват приложение занимаващи се с изграждането на велосипедни маршрути.

**Използвана литература**

1. Алгоритмите Dijkstra‘s и A\* algorithm - <https://stackabuse.com/dijkstras-algorithm-vs-a-algorithm/>
2. Mapbox, Google Maps и OpenStreetMap API сравнение - <https://relevant.software/blog/choosing-a-map-amapbox-google-maps-openstreetmap/>
3. Xcode и AppCode сравнение - <https://comparisons.financesonline.com/xcode-ide-vs-appcode>
4. OpenStreetMap документация - <https://wiki.openstreetmap.org/wiki/PBF_Format>
5. Подробна информация за всеки един формат - <https://osmcode.org/file-formats-manual/#format-comparison>
6. Apple developer документация - <https://developer.apple.com/>
7. Flutter документация - <https://docs.flutter.dev/resources/inside-flutter>
8. Актуалният „.proto“ файл – <https://github.com/openstreetmap/OSM-binary/blob/master/src/osmformat.proto>
9. Swift protobuf support - <https://github.com/apple/swift-protobuf>
10. Инструмент за ровене по файлове - <https://osmcode.org/osmium-tool/>

**Речник на често използвани абревиатури**

1. SDK - Комплект за разработка на софтуер
2. API - Приложен интерфейс за програмисти
3. NDK - Местен комплект за разработка
4. IDE - Интегрирана среда за разработка
5. JSON - JavaScript Object Notation
6. OSM - OpenStreetMap

**Съдържание**

**Увод1**

**Първа Глава -** **Методи и технологии за реализация на мобилни приложения1**

1.1 История и основни понятия на мобилните операционни системи – iOS и Android2

1.1.1 iOS2

1.1.2 Andorid2

1.2 Проучване на различни SDK за разработка на мобилни приложения2

1.2.1 Flutter SDK2

1.2.2 Andorid SDK2

1.2.3 iOS SDK2

1.3 OSM файлове2

1.3.1 Типове файлове2

1.3.2 Файлови формати2

1.3.3 Подреждане на обекти във файлове2

1.3.4 XML формат2

1.3.5 PBF формат2

1.4 Съществуващи решения2

**Втора Глава - Проектиране на избраните технологии за реализация на iOS мобилно приложение4**

2.1 Основни функционални изисквания2

2.2 Операционни системи2

2.3 Езици за програмиране2

2.4 Технологии за изграждане на потребителски интерфейс2

2.4.1 UIKit2

2.4.2 SwiftUI2

2.5 Избор на интегрална среда за разработка (IDE)2

2.6 Избор на формат на OSM файл2

2.7 Алгоритми за намиране на най-кратък път в граф2

2.7.1 Dijkstra2

2.7.2 A\*2

2.8 Визуализиране на карта и маршрут2

2.8.1 OpenStreetMap API2

2.8.2 Google Maps2

2.8.3 Mapbox2

2.9 Случващото се при стартиране на проекта2

**Трета Глава - Реализация на програмния продукт. Описание на начина на реализация4**

3.1 Файлова структура на проекта2

3.2 Извършващите се проверки при пускане на приложението и случващото след тях2

3.3 Изтегляне на файла2

3.4 Извличане на нужната информация2

3.5 Преработване на извлечената информация и създаването на граф2

3.6 Взимане на началната и крайна точка за изготвяне на маршрут2

3.7 Намиране на най-подходящ път2

3.8 Изобразяване на намерения маршрут2

**Четвърта Глава - Ръководство на потребителя4**

4.1 Инсталация2

4.2 Резюме на главния екран2

4.3 Ръководство за отваряне и редактиране на проекта2

**Заключение1**

**Използвана литература1**

**Речник на често използвани абревиатури1**

**Съдържание1**

**Приложение**