**­­­­Документация**

**на проект “Карта на град”**

Изготвена от Георги Йорданов Георгиев 81958

1. **Увод**

Гитхъб репо на проекта: <https://github.com/Gosheto012/City-Map>

**Описание и идея на проекта**

В днешно време почти няма хора, които не са чували или използвали Google Maps. Те са особено полезни при пътувания в непознати места. Основната идея на този проект е да се пресъздаде базово едно от най-честите приложения на този софтуер, а именно спомагането за ориентацията на потребителите, намиращи се на кръстовища в непознат град. Информацията за градската карта се съдържа в текстов файл, като промени могат да бъдат осъществявани многократно.

* 1. **Цел и задачи на разработката**

Програмата се фокусира върху основните нужди на хората, пътуващи в непознат град, като някои от нейните функции са: проверка дали има път между две кръстовища, маршрутите и дължината на трите най-кратки пътища между две кръстовища, проверка дали можем да се върнем обратно в същото кръстовище, проверка дали е възможно да се осъществи туристическа обиколка на града, проверка дали е възможно да се стигне до всяко кръстовище, извеждане на всички задънени улици. В програмата се съдържа и интерактивен режим, който при включване дава достъп на много други допълнителни функции. Основните цели и задачи на проекта са изпълнението на тези гореспоменати команди.

* 1. **Структура на документацията**

В документацията има пет основни глави: 1. Увод, 2. Преглед на предметната област, 3. Проектиране, 4. Реализация и тестване и 5. Заключение. Всяка глава си има различни аспекти за разглеждане на проекта.

1. **Преглед на предметната област**
2. **Основни дефиниции, концепции и алгоритми**

.

За да можем да запишем информацията за дадено кръстовище от файла, то трябва да се избере подходяща структура от данни, съхраняваща града. Той от своя страна представлява ориентиран граф с тегла по ребрата. Избрана е вградената в C++ хеш структура unordered\_map, като на всяко кръстовище съответства списък от имената на други кръстовища и съответната дължина към тях ( тоест списък с ребрата и теглата, които притежават върховете на графа ). За реализацията на проекта е необходимо да се изпълнят редица основни алгоритми от теорията на графите, включващи БФС, Дийкстра, Хиерхолцер, Йен и други. Освен това, за да може правилно да вземем всичката информация от даден файл, трябва да се конструира формат, който да чете от него.

1. **Дефиниране на проблеми и сложност на поставената задача**

Има няколко основни проблема в задачата и те са:

1. **Създаването на графа**

Поради факта, че графът е доста сложна структура от данни, особено с върхове низове и тегла по ребрата, то създаването му представлява разрешим проблем.

1. **Четенето от текстов файл**

Поради факта, че по условие се изисква програмата винаги да вкарва информацията за града от файл, то се налага дефинирането на формат и функции, които позволяват четенето от файл в създадената структура.

1. **Алгоритмите**

Определено това е най-големият проблем в задачата. Изискват се сериозни познания от теорията на графите, както и лесното запознаване с механизмите на не толкова известни алгоритми. Прилагането им чрез създадената структура също представлява частичен проблем.

1. **Интерактивният режим**
2. **Подходи и методи за решаването на поставените проблеми**

За да разрешим основните проблеми, то е необходимо да използваме добре основните структури от данни, съдържащи се в библиотеките “utility”, “unordered\_map”, “list”, “stack”, “queue” и други, както и алгоритми от дискретната математика. Добрата работа с файлове налага познаването на библиотеката “fstream”. Изчистеният код и успешното прилагане на ООП-практиките също улесняват форматирането на кода.

1. **Потребителски и качествени изисквания**

Има няколко потребителски и качествени изисквания задачата и те са:

1. **Създаванетофайла**

Поради факта, че програмата изисква четенето на информация от файл, то потребителят трябва предварително да си въведе картата на града. Тъй като форматът за разграничаването на отделните кръстовища и дистанция е чрез интервал, то имената на кръстовищата е необходимо да бъдат изписани слято.

1. **Валидация на града**

По условие няма две еднакви еднопосочни улици между две кръстовища. Затова трябва да се внимава да не се въвеждат едноименни кръстовища, създаващи улица с друго кръстовище. Това би объркало проверките за най-кратки пътища. За върховете няма проблем, тъй като хеш структурата не позволява два еднакви ключа.

1. **Типовете елементи на класовете**

С цел концентриране върху основните функции програмата се прецаква при въвеждане на тип, различен от исканата функция ( например ако се изисква число от тип int, а потребителят въведе низ, то програмата няма да проработи ).

1. **Проектиране**
   1. **Обща архитектура – ООП дизайн**

За да може да се запази цялат информция за графа, то е необходимо да се създаде един базов клас Graph. Негова член-данна е структурата unordered\_|map, като името на всяко кръстовище (всеки връх) представлява ключ, като съответната хеш стойност е списък от двойки име на кръстовище и дистанция, тоест списък с насочени ребра към други кръстовища със съответната дължина. Тази структура е по-ефективна от базовите модели за граф (представени чрез вектор от вектори или списък от списъци), тъй като при търсенето на кръстовище не е необходимо обхождане на контейнера, защото разполагаме с пряк достъп до съседните върхове поименно. Graph притижава и полезни функции като премахване на ребра и върхове, проверка дали има директен път от един връх към друг и т.н. Класът Data служи за вкарването на данните от текстовия файл в графа, като притежава методи за намиране на дума от съответния ред по индекс, създаване на списък и други. Най-важен е класът Algorithms, по чието се разбира, че съдържа реализация на основните алгоритми, изброени по-горе. Те се прилагат в класа UserInput, който притежава пряко функции за потребителския интерфейс. Там се съдържат и допълнителни член-данни location и вектор от затворени кръстовища, които реализират работата на потребителя при интерактивен режим.

* 1. **Диаграми**

**Diagram, schematic

Description automatically generated**

Фиг. 3.2.1 клас Graph

Клас Data

**Graphical user interface, application

Description automatically generated**

Фиг. 3.2.2 клас Algorithms

Graphical user interface, application

Description automatically generated

Фиг. 3.2.3 клас UserInput

1. **Реализация и тестване**
   1. **Реализация на класове**

Както е споменато в предишната глава, реализирани са 4 класа: Graph, Data, Algorithms и UserInput. Всеки от тях съдържа съответни гетъри, позволяващи достъпа до член-данните на класовете. Реализирани са конструктори, където е необходимо оператор “=”. Не се съдържат деструктори, тъй като контейнерите от STL съдържат собствени. Важно е да се подчертае, че в методите на някои класове трябва да се използва псевдоним като аргумент на функцията, защото се извършва промяна върху самия обект.

* 1. **Управление на паметта и алгоритми. Оптимизации.**

С паметта не се намират никакви проблеми. Трудната част е осъществяването на алгоритмите. За BFS се използва популярната структура queue, която реално пази следващото кръстовище, което трябва да бъде обходено. В много от случаите при различните алгоритми е наложително да се създаде и допълнителна структура, която пази двойка от име на кръстовище и булева стойност – посетена и непосетена. При проблеми за добавяне и премахване на елементи се изполва list, но някога за директното индексиране се налага употребата на vector. Приложение намира и stack в определени ситуации. Всеки алгоритъм си има своята специфичност, която ще бъде обяснена по време на защитата. Използването на вече създадени функции от предишния клас и операторът switch правят кода прегледен.

* 1. **Планиране, описание и създаване на тестови сценарии**

Основната идея е потребителят да разполага с главно меню, като при натискане на конкретно число се изпълняват съответните команди. От там се стартира и интерактивният режим, който си има отделно меню. Цялата информация за градската карта се чете от файл, като непосредствено след компилацията на програмата е необходимо да се напише името му, за да се продължи напред. Съществуват команди и за излизане от програмата или ново показване на главното меню. Форматът на картата на града при написването във файла е следният:

На всеки ред във файла на първа позиция стои името на дадено кръстовище, а след него двойки от имена на други кръстовища и дължината на улицата от първото кръстовище до второто. Може да има кръстовища, от които не излизат улици, но името на всяко съществуващо кръстовище в града да бъде въведено във файла, дори това да е задънена улица.

**общ формат:** **<Name><Space><Name><Space><Distance><Space><Name><Space><Distance> …..**

**Пример:**

**Gurko Demokraciq 43 Ivan\_Ivanov 3 NDK 32 Speedy 17**

**Speedy Ivan\_Ivanov 33**

**Demokraciq**

**NDK Gurko 28 Demokraciq 16**

За тестови сценарии са изпробвани всякякави варианти за градска карта. Примери: Всяко кръстовище е свързано със всяко, никое с никое, еднакви дължини на най-кратки пътища между две кръстовища, да няма цикли и т.н.

1. **Заключение**
   1. **Обобщение на изпълнението на началните цели**

Изпълнени са основните цели на проекта, както и са разрешени основните проблеми при изпълняването на задачите, които цели да изпълни тази програма. Създаден е и интерактивен режим, който е от голямо удобство за потребителите.

* 1. **Насоки за бъдещо развитие и усъвършенстване**

Като бъдещо развитие може да се счита намалянето на сложността на някои функции, добавяне на константен списък от затворени кръстовища. Вмъкването на графика също може да направи ползването на програмата от потребителите по-интерактивно и интересно.