Карта на град

Линк към хранилището в *Github*: https://github.com/KrisCvetanov/CityMap

Документацията за проекта е разделена на 3 части:

1. Увод
2. Проектиране
3. Реализация и уточнения

1. Увод

Проекта реализира карта на град. Информацията за картата се съдържа в текстов файл. Самия град е представен чрез ориентиран тегловен граф. Изискваните функционалности са реализирани чрез някои стандартни алгоритми за графи като: **BFS, DFS, Dijkstra, Hierholzer** и други.

За реализирането на програмата трябва да бъдат постигнати следните цели и задачи:

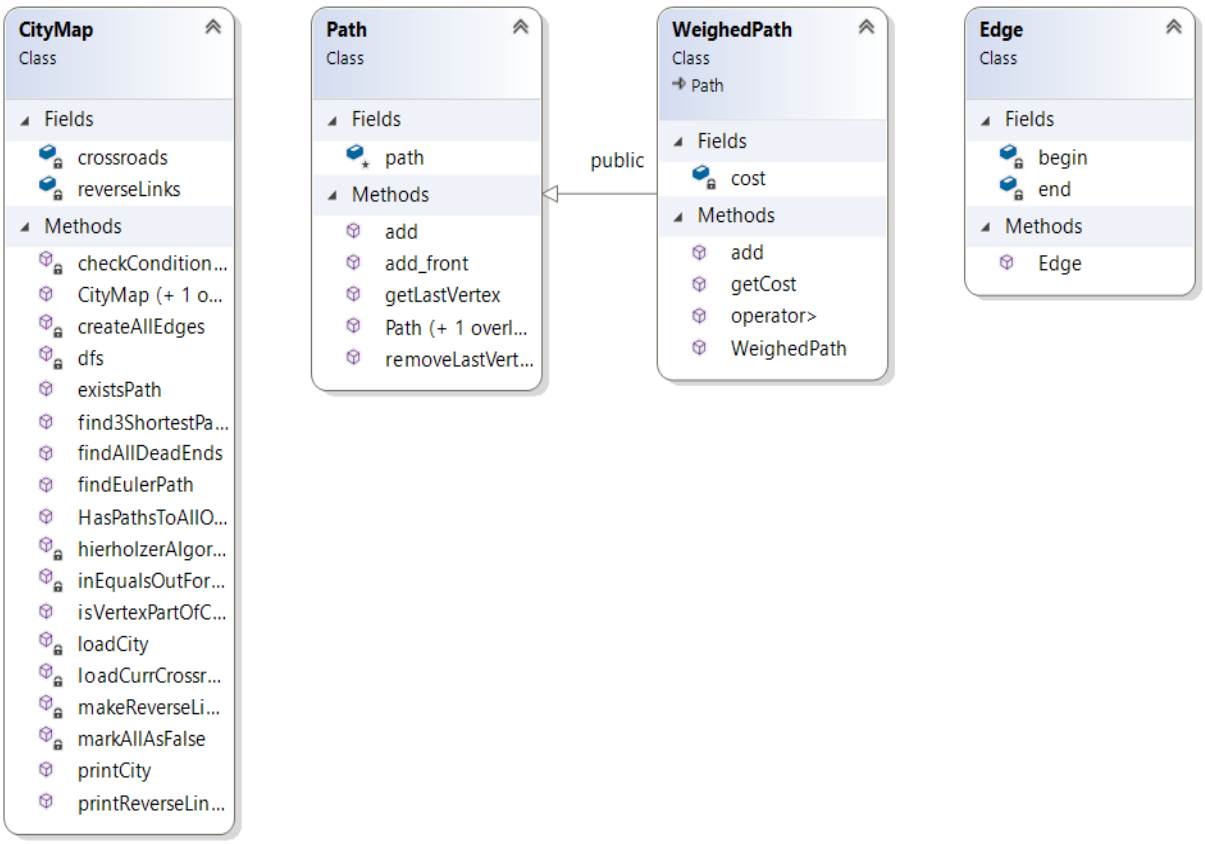
- Подходящо представяне на графа, за да може сложностите по операциите да бъдат оптимални
- Зареждане на информацията за графа от текстов файл
  - Подходящо представяне на ребро за по – чиста реализация
  - Подходящо представяне на път за по – чиста реализация

2. Проектиране

Реализирани са следните 4 класа:

- CityMap
- Edge
- Path
- WeighedPath

Те имат следните характеристики и йерархия:



Диаграма 2.1

1. **Клас Edge:** Представя еднопосочна улица между две кръстовища. Член данни:

- string begin – кръстовището, от което излиза улицата
- string end – кръстовището, в което влиза улицата

Методи:

public:

- Edge(const std::string&, const std::string&) – конструктор, приемащ 2 низа(начално и крайно кръстовище)
- friend std::ostream& operator<<(std::ostream&, const Edge&) – оператор за изход

2. **Клас Path:** Представя поредица от кръстовища, които образуват път.
Член данни:
   - std::list<std::string> path – списък от кръстовищата, образуващи пътя

   Методи:

   - Path() – конструктор по подразбиране
   - Path(const std::list<std::string>&) – конструктор, приемащ списък от кръстовища
   - void add(const std::string&) – добавя кръстовище към края на пътя
   - void add_front(const std::string&) – добавя кръстовище в началото на пътя
   - const std::string& getLastVertex() const – извлича последното кръстовище от пътя
   - void removeLastVertex() – премахва последното кръстовище от пътя. Това е нужно в един от алгоритмите, в случай че самия път е цикъл
   - friend std::ostream& operator<<(std::ostream&, const Path&) – оператор за изход

3. **Клас WeighedPath: public Path :** Път, на който се отчита дължината(теглото). Наследява класа Path. Тъй като графа, с който работим, е тегловен, има алгоритми, в които ни е нужно и теглото на пътя.

Член данни:

- double cost – теглото на пътя

Методи:

public:

- WeighedPath() – конструктор по подразбиране
- void add(const std::string&, double) – предефинира метода от родителския клас. Добавя ребро към края на пътя, като се добавя и разстоянието между последното кръстовище от пътя и текущо добавеното.
- double getCost() const – връща теглото на пътя
- bool operator>(const WeighedPath& other) – проверява дали текущия път е по – дълъг от подадения като параметър. Метода е нужен при ползването на priorityQueue, съхраняваща пътища с тегло.

1. **Клас CityMap:** Главният клас в проекта**.** Осъществява представянето на града от кръстовища като ориентиран тегловен граф. Член данни:
   - std::unordered_map<std::string, std::unoredered_map<std::string, double> > crossroads – графът, представящ кръстовищата и техните връзки (улиците). Избрано е представяне чрез **map** без наредба. Операциите на структурата от данни са константни по време(макар и не гарантирано във всички случаи), като добре пасва на логиката от задачата : всяко кръстовище(ключ) има множество от кръстовища, към които има улица(стойност).
   - std::unordered_map<std::string, std::unoredered_set<std::string> > reverseLinks – същият граф, но с ребрата са с обърната посока. Това представяне е нужно за някои от алгоритмите.

Методи:

private:

- void loadCity(const std::string&) – зарежда информацията за града от файл, чието име е подадено като параметър
- void loadCurrCity(const std::string&) – зарежда информацията от текущ ред на файла
- void makeRevereseLinks() – инициализира графа с обратни връзки reverseLinks, след като е заредена информацията от файла
- markAllAsFalse(std::unordered_map<std::string, bool>&) const – приема като параметър **map** с ключове имената на кръстовищата, като инициализира стойностите за всеки ключ с **false.** Методът маркира всички върхове като необходени
- createAllEdges(std::unorederd_map<std::string, unordered_set<std::string > >&) const – инициализра подадения като параметър граф, който представя ребрата на оригиналния. Този метод се ползва само веднъж в алгоритъма за намиране на Ойлеров път
- void dfs(const std::string&, const std::string&, std::unoredered_map<std::string, bool>&, int, bool&) const – Обхождане в дълбочина. Приема като параметри начален и краен връх, **map** следящ обходените върхове, цяло число, ограничаващо броя извиквания на рекурсията, и булева променлива, върната като резултат в existsPath()
- Path hierholzerAlgorithm(const std::string&, std::unordered_map<std::string, std::unordered_set<string> >&) const – реализира алгоритъма на Hierholzer за намиране на Ойлеров цикъл. Връща като резултат самия цикъл
- bool checkConditionsForEulerPath(std::string&, std::string&) const – проверява конкретни условия за съществуване на Ойлеров път, които са :

- точно един от върховете на графа е с полустепен на входа с едно повече от полустепента на изхода

- точно един от върховете на графа е с полустепен на изхода с едно повече от полустепента на входа

- всички останали върхове са с равни полустепени на входа и изхода

- bool inEqualsOutForAllVerticies(std::string&, std::string&) const – проверява дали всички върхове са с равни полустепени на входа и изхода

public:

- CityMap() – конструктор по подразбиране, зарежда информацията от файл с име “citymap.txt”
- CityMap(const std::string&) – конструктор, приемащ име на файл, от който да зареди информацията за града
- bool existsPath(const std::string&, const std::string&) const – приема като параметри два върха от графа и проверява дали има път между тях. В условието на проекта това е задача 1.
- std::list<WeighedPath> find3ShortestPaths(const std::string&, const std::string&, const std::unordered_set<std::string>&) const – намира 3-те най – кратки пътя между два върха, подадени като параметри, като функцията приема и множество от затворени кръстовища. Изпълнява от условието задачи 2. и 3. Задача 2. е подмножество на задача 3., тъй като може да подадем празно множество от затворени кръстовища.
- bool isVertexPathOfCycle(const std::string&) const – Проверява дали връх е част от цикъл. Задача 4. от условието
- std::optional<Path> findEulerPath() const – връща Ойлеров път, ако такъв съществува. В противен случай връща nullopt_t (или само {}). **Хубаво е да се отбележи, че библиотеката optional е достъпна след C++17. Т.е. ако програмата се стартира на по – стара версия, няма да се компилира!**

- bool hasPathsToAllOthers(const std::string&) const – проверява дали от даден връх има път до всички останали върхове. Задача 6. от условието
- std::list<Edge> findAllDeadEnds() const – Намира всички задънени улици. По – точно, всички ребра, които свършват във върхове с полустепен на изхода 0. Задача 7. от условието

3. Реализация и уточнения

- В класовете, реализирани в проекта, не се ползва динамично заделяне на памет. За мое щастие, този семестър може да се ползва stl.
- Относно избора на структурите от данни за реализиране на алгоритмите, първо е гледано те да са с оптимална сложност и след това да са удобни и чисти за ползване.
- Четенето от файл е с указания в условието формат.
- В main има подготвени тестове. Файловете, откъдето се зарежда информацията, са в хранилището в *Github*.
- Източници, от които са гледани реализации на някои от алгоритмите:
  - https://en.m.wikipedia.org/wiki/K_shortest_path_routing?fbclid=IwAR0PtRqL1BzfeoeLjhpFpHwIv-5TRQUZEK388Owp9OkZDLllUkqOGtadd1w
  - https://math.stackexchange.com/questions/1871065/euler-path-for-directed-graph