Карта на град

Обща структура на проекта: Приложението „Карта на град” позволява намиране на маршрути и отговор на определени заявки за градска мрежа от кръстовища, свързани с улици. То се състои от три основни компонента, вдъхновени от MVC модела: TownMapUI клас, който реализира интерактивния елемент, т.е. взаимодействие с потребителя (в текущия вариант на проекта интерфейсът е конзолен); TownMap клас, който отговаря на заявките и осъществява връзка между интерфейса и вътрешното представяне; Graph клас, който за целите на задачата представлява граф, реализиран като списък от върхове, а всеки връх държи в себе си списъци с входящи и изходящи ребра. Това е подходящо, защото желаните операции лесно се превеждат до операции с графи, където върховете са кръстовищата, а ребрата – улиците между тях. Следва описание на начините, по които е осъществено това, заедно с преоделените проблеми и използваните класове.

Имплементация – класове, алгоритми, особености (избегнати проблеми):

Class Edge – представя ребро в графа (отговаря на улица). Член данните му са начален и завършващ връх – start & end, дължина/тегло – length, и булева променлива inRepairs, която е въведена, за да може да се поддържа операцията извеждане на алтернативен маршрут при ремонт на дадено ребро. За целта при такава ситуация тази променлива се слага на истина и останалата логика е реализирана така, че да игнорира реброто в този случай, т.е. все едно не е част от графа. Така всички функционалности продължават да работят коректно и е достатъчно да маркираме път като „в ремонт” или „ремонт приключен” за смяна между състоянията.

Class Vertex – представя връх в графа (отговаря на кръстовище) чрез динамичен масив (vector) от върхове, към които има ребра – outEdges, и vector от върхове, от които има ребра към него - inEdges, т.е. един вид списъци на наследниците и предшествениците. За удобство и коректност той предоставя и функции за определяне на степента на входа и изхода, които вземат предвид това дали реброто е в ремонт и винаги връщат коректен резултат: getDegreeIn() и getDegreeOut().

Class Graph – имплементира основната алгоритмична логика на програмата. За целта държи в себе си хештаблица от върхове. Освен селектори и мутатори за добавяне на ребро между два върха с определена дължина – **addEdge** (CrossPoint,CrossPoint,int), намиране на ребро по два върха – **findEdge** (CrossPoint, CrossPoint), връщащ указател към това ребро, size(), връщаш размера (броя върхове на графа), той предоставя и следните функции: **getShortestPath** (CrossPoint,CrossPoint,int&), която приема два върха и параметър по псевдоним, в който ще съхрани общата дължина на пътя (сума от дължините на ребрата, а не просто брой ребра). За целта е реализиран алгоритъма на Дийкстра за намиране на най-късите пътища от връх до всички останали. Той пази в таблица предшественици и най-къси разстояния до върховете и на всяка стъпка взима нов връх, разстоянието до което до момента е минимално, добавя го към вече обработените и проверява дали през него не се получава по-къс път до някой от съседите му, а таблицата от предшественици се използва за възстановяване на самия път; **hasPath** (CrossPoint,CrossPoint) - връща дали има път между два върха в графа. За реализирането му е използвано стандартно обхождане в дълбочина: чрез него се обхождат всички достижими от дадения връх и накрая просто се проверява дали този връх е сред тях. За помощна множество от посетени върхове, предпочетено заради бързите операции за включване и търсене; **hasCycle** (CrossPoint) – връща дали в графа има цикъл с участието на дадения връх; за целта се използва отново обхождане в дълбочина и имаме цикъл тогава и само тогава, когато даденият връх се появи за втори път при обхождането. Проблем тук е, когато имаме двупосочна улица – това може да се регистрира като цикъл при наивна имплементация, затова се пази при обхождане освен върха и предшественика му в pair<CrossPoint,CrossPoint> и ако върхът, от който е дошло второто срещане, е такъв, към който има ребро от първоначалния, това ребро временно се слага в „ремонт” и се прави проверка дали при премахването му все още има път между началния връх и този, тъй като само в такъв случай наистина ще имаме цикъл; **countDeadends** () - брои задънените улици, т.е. върхове без изходящи ребра. Тя се позовава на съответните

функции за степен на върховете и така работи коректно дори при ремонтиращи се ребра; **hasEulerCycle** () – проверява дали съществува Ойлеров цикъл чрез популярното НДУ, а именно степента на входа и изхода на всеки връх да са равни. Отново се използват техните член функции за коректна работа и при ремонтиращи се ребра; **eulerCycle** (CrossPoint) - ако има ойлеров цикъл в графа, го принтира с начален връх подадения аргумент, т.е. той се явява начална точка на обиколката. Ако няма, връща празен списък. Използва се известния алгоритъм за намиране на ойлеровия цикъл с помощта на 2 стека – един за пътя досега и един за текущия път. Те се съчетават с обхождането в дълбочина, за да реализират базовата идея: намираме един цикъл и после докато има необходени ребра, откриваме миницикли и ги навързваме към големия. Не на последно място Graph класът има и предефиниран оператор за стандартен изход, който го извежда във формата, дефиниран за файловете, от които трябва да се десериализира, т.е. позволява се и сериализация.

Class TownMap – осъществява взаимодействие между интерфейса и класа за граф, като „превежда" задачите на приложението до задачи за графа. Това разделение е направено предимно с чел четимост и лесни бъдещи модификации и разширения. Функциите за намиране на път, цикъл, проверка за съществуване на път, намиране на задънени улици и обиколка на всички улици (ойлеров цикъл) се превеждат до съответните алгоритми за графа и се извеждат резултатите. Освен това като типична карта този слой поддържа текущото местоположение, което се променя от функцията за намиране на най-кратък път до друго или „ръчно", с функцията **goTo** (CrossPoint). Освен това той предоставя функции за обозначаване на дадено кръстовище като ремонтирано в момента, десериализация и сериализация по описаните в изискванията правила – на всеки ред информация за връх, на първо място е номерът на върха, следват връх, до които има път, дължина на пътя, друг връх и тн.

Class TownMapUI – осъществява връзка с потребителя. Предоставена е help команда, принтираща меню с поддържаните операции, отчита се потребителския избор и се извиква необходимата функция от класа TownMap. Поддържат се дефинираните в изискванията операции, както и команда за зареждане на текуща карта от файл. Идеята на присъствието му като отделен клас е да се улесни бъдеще подобрение на проекта, в който конзолния интерфейс да се замени с графичен или дори уеб базиран.

<u>Бъдещи подобрения:</u> Възможни са подобрения в следните насоки:

- Вместо един най-къс път да се предлагат няколко възможни, подредени в нарастващ ред на дължината, и да се даде право на избор
- Интерфейсът да бъде заменен с графичен интерфейс
- Да се намира маршрут, минаващ през определени кръстовища (например за разглеждане на забележителности по време на екскурзия) – например чрез конкатениране на пътища. Добра опция би било да може да бъде намерен минимален такъв
- Да се въведе допълнителна информация за обекти в близост до кръстовища и да могат да се намират най-близки обекти от даден тип, напр. заведения, магазини, забележителности etc
- Да се предлагат множество алтернативни маршрути