**СОФИЙСКИ УНИВЕРСИТЕТ „СВ.КЛИМЕНТ ОХРИДСКИ”**

**ФАКУЛТЕТ ПО МАТЕМАТИКА И ИНФОРМАТИКА**

**СТРУКТУРИ ОТ ДАННИ И ПРОГРАМИРАНЕ**

**ПРОЕКТ: КАРТА НА ГРАД**

Разработил: **Христо Христов**, ФН 82109

Специалност “Компютърни науки”,

Втори курс

**СЪДЪРЖАНИЕ**

**Глава 1.Увод**

* **Описание и идея на проекта.**

Проекта цели да представи опростена имплементация на карта на град. Информацията за улиците и кръстовищата се прочита от файл и се зарежда в паметта на програмата. Добавени са функции, които да работят върху зададената структура от данни – в случая – граф, който съдържа информацията за картата на града.

* три основни вида обекти:
* EdgeInfo(Ребро)
* File(Файл)
* Graph(Граф)

**1.2.Цел и задачи на разработката**

Целта е, използвайки възможностите на езика C++ да се реализира функции, които да се прилагат върху структурата от данни граф.

**Глава 2.Преглед на предметната област**

**2.1. Основни дефиниции , концепции и алгоритми , които ще бъдат използвани**

Използват се map, list, queue, vector, динамична памет и алгоритми за обхождане на графи(DFS).

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

Сложността на проекта се изразява в правилното имплементиране на Граф в компютърната памет, както и създаването на правилните алгоритми и функции, които по оптимален начин с минимална сложност да позволяват бърза работа със съответната структура от данни.

**2.3. Подходи , методи за разрешаване на проблеми**

Използването на map, list, pair и динамична памет спомага за подобряването на функционалността. Зареждането на файла в паметта на програмата се прави от отделен клас File. Използвани са познати алгоритми за обхождане на графи.

**2.4.Потребителски и качествени изисквания**

Потребителят разполага с множество от команди с които да може да проверява няколко неща: дали има път от едно кръстовище до друго, дали има път от кръстовище към всички останали , най-кратки пътища от точка А до точка Б, проверка за наличие на цикли, както и извеждане на всички задънени улици.

**Глава 3.Проектиране**

**3.1.Обща архитектура – ООП дизайн**

Класовете *File* и *EdgeInfo* са използвани в композиция. Класът *File* използва *EdgeInfo* за да прочете информация от файл и да я зареди в паметта на класа Graph.

**Глава 4.Реализация , тестване**

**4.1.Реализация на класовете**

Класът *EdgeInfo* - член-данни – nodeStart(string), nodeEnd(string), weight(int). Те съдържат информация за това от кое кръстовище до кое кръстовище е улицата и колко е дълга тя. Реализирана е голямата четворка – конструктор, копи-конструктор, оператор = (равно) и деструктор. Реализирани са get и set методи за промяна и достъп до член данните на класа. Използвани са знания от ООП.

**Класът EdgeInfo съдържа:**

* set – методи с които се задава/променя стойността на отделните член данни
* get – методи с които се достъпва стойността на отделните член данни

Класът File – член-данни – file(ifstream), nodes(map<int,string>), edges(list<pair<int,int>>\*). Резилирани са конструктор и деструктор. Конструктора приема string с името на файла който искаме да прочетем.

**Класът File съдържа:**

* Функция int parseToInt(string), която приема string и го превръща в int
* Функция int searchKeyInMap(string), която приема string и намира съответстващата му Int стойност в map
* Функция void loadNodes(string), която приема стринг и го зарежда в nodes, като автоматично го индексира
* Функция void loadEdges(vector<EdgeInfo>), която приема вектор от EdgeInfo обекти и ги добавя в edges
* Функция Graph loadTextToGraph(), която взима данните от File и ги прехвърля в Graph, който ни връща. Така цялата информация от файла е вече в Graph

Класът Graph – член-данни – V(int), nodes(map<int,string>), edges(list<pair<int,int>>\*). Разполага с множество от private и public методи. Реализирана е голямата четворка.

**Класът Graph съдържа:**

* Функция int searchKeyInMap(string), която приема string с името на кръстовище и връща Int стойността му в map
* Функция searchInEdges(int), която приема int и връща list<pair<int,int>> с данните за дадената улица(начална точка, дължина, крайна точка)
* Функция void bubbleSort(vector<pair<int,int>>), която приема вектор от двойки и го сортира използвайки помощната функция swap(pair<int,int>, pair<int,int>)
* Функция void shortestPathAlgorithm (int,int,bool\*,vector<int>,int,int,vector<pair<vector<int>,int>>), която взима начално и крайно кръстовище и намира най-краткия път между тях, като намира всички пътища и избира този с най-малка дължина
* Функция void deleteNodes(vector<int>), която приема вектор от Int и се използва за затварянето на кръстовища. Служи за помощна функция.
* Функция bool isCyclicUtil(int, bool\*, bool\*), която приема int и 2 масива от нули и единици, които представляват посетените до сега кръстовища и опашка, функцията търси цикъл в дълбочина
* Функция int countPaths(), която връща броя на всички пътища
* Функция void DFS(int,bool\*), която реализира алгоритъм за обхождане в дълбочина
* Функции void addNode(int,string) и void addPath(int,int,int), които добавят съответно ново кръстовище или нова улица
* Функция void print(), която принтира картата на града на конзолата
* Функция bool isReachable(string,string), която проверява дали има път от едно кръстовище до друго
* Функция bool isReachableToAllNodes(string), която проверява дали от дадено кръстовище можем да стигнем до всички останали
* Функция map<string,string> findDeadEnds(), която връща всички задънени улици в графа
* Функция void getShortestPaths(string,string), която намира всички пътища от точка до точка, сортира ги и извежда 3-те най-кратки ако такива съществуват
* Функция void getShortestPaths(string,string,list<string>), която също извежда 3-те най-кратки пътя от точка до точка, но при нея можем да дадем списък от кръстовища, които са затворени
* Функция bool isCyclicFromNode(string), която използва функцията isCyclicUtil и проверява дали от дадено кръстовище можем да направим цикъл и да се върнем на същото кръстовище

4.2**.Управление на паметта и реализация на алгоритмите.Оптимизация.**

Управлението на паметта се осъщестявава от конструкторите и деструкторите , които са реализирани във всеки клас.

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

В main.cpp могат да бъдат тествани всички функции, с които програмата разполага. Резултатите от тях могат да бъдат видени на конзолата.

**Глава 5.Заключение**

**5.1.Обобщение на изпълнението на началните цели.**

Имплементирана е карта на град. Създадени са кръстовищата, както и улиците които ги свързват. Има реализирана функционалност.

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

Приложението дава възможност за развитие , чрез добавяне на графичен интерфейс на по-късен етап.