Карта на град

Проекта може да се раздели на две части:

1. Структури и алгоритми свързани с граф.
    a. Graph
    b. WGraph
    c. {Graph Algorithms}
2. Специфични за проекта структури
    a. CityMap
    b. InteractiveCityMap

Обектите и функциите свързани с първата част от проекта съм отделил в namespace **graph**.

Функции

bool #has_path(const Graph<T>& g, const T& a, const T& b): ture ако в граф *g* има път между връх *a* и връх *b*, в противен случай **false**.

list<T> #shortest_path(const WGraph<T> g, const T& a, const T& b): най-краткия път между връх *a* и *b* в граф *g*.

bool #has_cycle_from(const Graph<T>& g, const T& a): true ако от връх a участва в цикълв, противен случай **false**.

bool #all_reachable_from(const Graph<T>& g, const T& a): true ако от връх *a* може да се достъпи всеки друг връх в графа *g*, противен случай **false**.

list<pair<T, T>> dead_ends(const Graph<T>& g): списък от всички ребра водещи до връх без наследници в граф *g*.

list<T> eulerian_path(Graph<T> g): намира ойлеров път или цикъл в граф *g*, ако не съществува връща празен списък.

#load_WGraph(const char* filename, WGraph<T>& g): от файл *filename* зарежда тегловен граф *g*.

Обекти

Graph<Vertex>: stl рализация на абстрактната структура граф, чрез списъци на съседство. (Vertex е типа на върховете)

unordered_map #graph: всички върхове и техните съседи в графа.
#for_each_vertex(Function fn): за всеки връх се изпълнява функция fn(Vertex)
#for_each_neighbor_of(const Vertex& x , Function fn): за всеки съсед на x се изпълнява функция fn(Vertex).

#vertex_count: боря на върховете в графа.
#add_vertex(const Vertex& x): добавя връх x към графа.
#remove_vertex(const Vertex& x): изтрива връх x от графа.
bool #has_vertex(const Vertex& x): ако връх x е в графа **true**, ако не **false**.
bool #adjacent(const Vertex& x, const Vertex& y): ако съществува ребро (x, y) **true**, ако не **false**.
int #out_degree(const Vertex& x): броя на изходните ребра от върха x.
int #int_degree(const Vertex& x): броя на входните ребра от върха x.
int #degree(const Vertex& x): броя на всички ребра в които върха x участва(**in_degree + out_degree**)
unordered_set #neighbors(const Vertex& x): множество от всички съседи на връх x.
#add_edge(const Vertex& x, const Vertex& y): създава ребро (x, y)
#remove_edge(const Vertex& x, const Vertex& y): изтрива ребро (x, y)
#edge_count: броя на всички ребра в графа.
#clear: изтрива всички ребра и върхове.

WGraph<Vertex>: тегловен граф, наследник на Graph<Vertex>. Използва unordered_map като тегловна функция. (w[(A, B)] = cost)

double #weight(const Vertex& x, const Vertex& y): теглото на ребро (x, y).
#weight(const Vertex& x, const Vertex& y, double w): задава тегло на реброто (x, y).
#add_edge(const Vertex& x, const Vertex& y, double w): създава ребро (x, y) с тегло w.

от Graph<Vertex> наследява:

unordered_map #graph: всички върхове и техните съседи в графа.

#for_each_vertex(Function fn): за всеки връх се изпълнява функция fn(Vertex)

#for_each_neighbor_of(const Vertex& x , Function fn): за всеки съсед на x се изпълнява функция fn(Vertex).

#vertex_count: боря на върховете в графа.

#add_vertex(const Vertex& x): добавя връх x към графа.

#remove_vertex(const Vertex& x): изтрива връх x от графа.

bool #has_vertex(const Vertex& x): ако връх x е в графа **true**, ако не **false**.

bool #adjacent(const Vertex& x, const Vertex& y): ако съществува ребро (x, y) **true**, ако не **false**.

int #out_degree(const Vertex& x): броя на изходните ребра от върха x.

int #int_degree(const Vertex& x): броя на входните ребра от върха x.

int #degree(const Vertex& x): броя на всички ребра в които върха x участва(**in_degree + out_degree**)

unordered_set #neighbors(const Vertex& x): множество от всички съседи на връх x.

#remove_edge(const Vertex& x, const Vertex& y): изтрива ребро (x, y)

#edge_count: броя на всички ребра в графа.

#clear: изтрива всички ребра и върхове.

CityMap: карта на град, реализирана с граф, където кръстовищата и улиците съответстват на върхове и ребра.

```
typedef string Crossroad;
typedef graph::WGraph<Crossroad> StreetMap;
typedef list<Crossroad> path;
typedef pair<Crossroad, Crossroad> street;
```

bool #has_path(const Crossroad&, const Crossroad&): Проверка дали има път между две зададени кръстовища.

path #shortest_path(const Crossroad&, const Crossroad&): Намира най-кратък път между две зададени кръстовища кръстовища.

path #shortest_path(const Crossroad&, const Crossroad&, const path&): Намира най-кратък път между две кръстовищя при подаден списък от затворени кръстовища.

bool #short_tour(const Crossroad&): Проверява дали от зададено кръстовище може да се обиколи част от града и пак да се върнем в началната точка.

path #full_tour: Проверка дали може да се обиколи целия град без да се мине по една и съща улица два пъти, ако може връща пътя.

bool #is_connected(const Crossroad&): Дали от зададено кръстовище може да се стигне до всички останали в града.

list<street> #dead_streets: всички задънени улици в града.

#load_map(const StreetMap&): зарежда карта на града от граф.

InteractiveCityMap: интерактивна карта на град,наследник на CityMap.

#loaction: кръстовището на което се намираме в момента.

#change(const Crossroad&): променя кръстовището на което се намираме.

#neighbours: оптечатва всички съседни кръстовища.

#move(const Crossroad&): мести текщото кръсовище и показва пътя по който минава.

#close(const Crossroad&): затваря кръстовище.

#open(const Crossroad&): отваря кръстовище.

#closed: изпежда всички затворени кръстовища.

#tour: извежда маршрут на туристическа обиколко на града.

Подобрения

Може да се подобри сложността на някои от функциите свързани с граф.