ТРАНСПОРТНЫЙ СПРАВОЧНИК

1 Условие задачи

Требуется разработать транспортный справочник – систему

общественного транспорта, к которой будут осуществляться запросы.

Форматом ввода и вывода данных будет являться JSON.

2. Формат ввода

На вход программе подаётся JSON-объект, описывающий запросы на

создание базы данных (по ключу base\_requests), настройки маршрутизации (по

ключу routing\_settings) и запросы к этой базе (по ключу stat\_requests). Ответы

на запросы также выводятся в формате JSON.

По ключу base\_requests находится JSON-массив, элементами которого

являются запросы на построение базы (остановка или автобус).

Описание остановки производится с помощью JSON-объекта и выглядит

следующим образом:

• name — ключ, по которому находится название остановки;

• latitude и longitude — число с плавающей точкой (широта и

долгота);

• road\_distances — объект, в котором вводятся расстояния по дорогам

от этой остановки до соседних. Ключом является название соседней

остановки, а значением – расстояние, которое задается в метрах и

является целым положительным числом, не меньшим чем

географическое расстояние между этими остановками.

Пример:

{

"latitude": 52.427337,

"longitude": 30.905488,

"name": "Predpriyatiye Elektroseti",

"road\_distances": {

"Radiozavod": 1190,

"Zavod Tsentrolit": 585

},

"type": "Stop"

}

Описание автобуса (либо иного вида общественного транспорта)

выглядит следующим образом:

• name – номер автобуса;

• is\_roundtrip – булево значение, которое указывает, является ли этот

автобус кольцевым;

• stops – массив названий остановок, по которым проходит автобус.

Для кольцевого автобуса будут перечислены остановки от первой до

последней, для обычного же – остановки, по которым он проходит

первую половину пути.

Пример:

{

"type": "Bus",

"name": "21",

"stops": [

"Volotova",

"Khram Preobrazheniya Gospodnya",

"Ulitsa Ogorenko",

"Ledovyy dvorets",

"Druzhba"

],

"is\_roundtrip": false

}

Настройки маршрутизации подаются по ключу routing\_settings во

входном JSON-объекте. Они тоже являются объектом, в котором находятся два

поля:

• "bus\_wait\_time" — среднее время ожидания автобуса на остановке

(в минутах). Считается, что, когда бы человек ни пришёл на

остановку и какой бы ни была эта остановка, он будет ждать любой

автобус в точности указанное количество минут. Значение —

положительное целое число.

• "bus\_velocity" — скорость автобуса (в км/ч). Скорость любого

автобуса постоянна и в точности равна указанному числу. Значение

— положительное вещественное число.

Запросы к базе подаются в виде объекта по ключу stat\_requests. Всего

предусмотрено три типа запросов, каждый из которых имеет свой номер:

вывести информацию об автобусе, вывести информацию об остановке и

проложить маршрут между остановками. Каждый из них вводится в виде объекта

с общим ключом id – целое число. Запросы на вывод информации об автобусе

или остановке имеют ключ name – название остановки или автобуса. Запрос на

построение маршрута же имеет два поля – from и to (названия остановок, между

которыми проложить маршрут).

Пример запроса на вывод информации об автобусе:

{

"type": "Bus",

"name": "297",

"id": 1

}

Пример запроса на вывод информации об остановке:

{

"type": " Stop",

"name": "Vokzal",

"id": 2

}

Пример запроса на построение маршрута:

{

"type": "Route",

"from": "Vokzal",

"to": "Ulitsa Lenina",

"id": 3

}

3. Формат вывода

Вывод осуществляется также в формате JSON в виде массива объектовответов на запросы. Каждый ответ содержит поле с ключом request\_id, по

которому и ставится ответ в соответствие запросу.

Если же во время запроса произошла ошибка (не найден автобус или

остановка, либо же невозможно построить маршрут), то вторым и единственным

полем в ответе будет "error\_message": "not found".

Пример ответа на запрос об остановке:

{

"request\_id": 1,

"buses": [

"2",

"6a"

]

}

Где buses – это массив номеров автобусов, проходящих через данную

остановку, рассортированных в алфавитном порядке.

Пример ответа на запрос об автобусе:

{

"curvature": 1.2345,

"unique\_stop\_count": 6,

13

"stop\_count": 7,

"request\_id": 2,

"route\_length": 5750

}

Где:

• curvature – извилистость маршрута (отношение расстояния по

дорогам к географическому расстоянию). Число с плавающей

точкой, большее либо равное 1;

• stop\_count – количество остановок, по которым проезжает автобус;

• unique\_stop\_count – количество уникальных остановок;

• route\_length – длина маршрута, который проходит автобус(в

метрах).

Пример ответа на запрос о построении маршрута:

{

"total\_time": 24.21,

"items": [

{

"time": 6,

"type": "Wait",

"stop\_name": "Vokzal"

},

{

"span\_count": 5,

"bus": "1",

"type": "Bus",

"time": 15.555

},

{

"time": 6,

"type": "Wait",

"stop\_name": "Ulitsa Lenina"

},

{

"span\_count": 2,

"bus": "5a",

"type": "Bus",

"time": 7.78

}

],

"request\_id": 5

}

Где:

• total\_time – все время, затраченное на маршрут в минутах;

• items – массив элементов маршрута – выйти из автобуса (если это не

начало маршрута) и подождать на остановке ("type": "Wait") или

проехать на данном автобусе ("bus": "5a");

• time – время, затраченное на проезд на данном автобусе за

span\_count остановок (для элемента маршрута Bus) или же время,

для того чтобы подождать автобус на остановке (равно

bus\_wait\_time из настроек маршрутизации).

4 Описание решения задачи

JSON

Node – класс, хранящий в себе один из типов JSON. Он является public

наследником std::variant(типобезопасный union).

Его методы:

• const std::vector<Node>& AsArray() const – представить данный

объект как вектор из Node;

• const std::map<std::string, Node>& AsMap() – представить данный

объект как словарь из строки в Node;

• int AsInt() const – представить данный объект как целое число;

• double AsDouble() const – представить данный объект как целое

число с плавающей точкой;

• bool AsBool() const – представить данный объект как bool;

• const std::string& AsString() const – представить данный объект как

строку.

Document – класс, являющийся JSON-документом. Имеет поле Node root,

т. е. корневой узел.

Его методы:

• explicit Document(Node root) – конструктор, принимающий

корневой JSON-узел;

• const Node& GetRoot() const – вернуть константную ссылку на

корневой узел.

Document Load(std::istream& input) – функция, которая принимает

входной поток, считывает оттуда JSON-документ и возвращает его.

std::ostream& Print(std::ostream& output, const Document& document) –

функция, которая принимает выходной поток как первый аргумент и записывает

туда JSON-документ, переданный вторым аргументом.

Граф и маршрутизатор

Edge – шаблонная структура ребра графа, где шаблонным параметром

является вес.

Ее поля:

• VertexId from – номер вершины, откуда выходит ребро графа;

• VertexId to – номер вершины, где заканчивается ребро графа;

• Weight weight – вес ребра графа.

DirectedWeightedGraph – шаблонный класс, который реализовывает

структуру данных взвешенный ориентированный граф. Шаблонным

параметром является вес ребра.

Его поля:

• std::vector<Edge<Weight>> edges\_ – вектор ребер;

• std::vector<std::vector<EdgeId>> incidence\_lists\_ – вектор векторов

инцидентов(ребер, которые начинаются или заканчиваются в данной

вершине), т. е. каждый индекс данного поля является номером

вершины, а вектор который находится в данном элементе – это

множество всех номеров граней, инцидентных данной вершине.

Методы:

• DirectedWeightedGraph(size\_t vertex\_count) – конструктор,

который в качестве параметра принимает количество вершин графа;

• EdgeId AddEdge(const Edge<Weight>& edge) – метод добавления

ребра графа. Принимает структуру Edge и возвращает номер грани.

• size\_t GetVertexCount() const / size\_t GetEdgeCount() const – метод

получения количества вершин / ребер графа;

• const Edge<Weight>& GetEdge(EdgeId edge\_id) – метод получения

грани по ее номеру;

• IncidentEdgesRange GetIncidentEdges(VertexId vertex) – метод

получения списка инцидентов по вершины по ее номеру.

Router – шаблонный класс маршрутизатора, шаблонным параметром

которого является вес грани. Он строит самый оптимальный маршрут, т. е.

минимального общего веса.

Его поля:

• const DirectedWeightedGraph<Weight>& graph\_ - ссылка на

взвешенный ориентированный граф;

• RouteId next\_route\_id\_ – счетчик номера маршрута;

• std::unordered\_map<RouteId, std::vector<EdgeId>>

expanded\_routes\_cache\_ – словарь, ключом которого является номер

маршрута, а значением – вектор граней, по которым проходит

маршрут;

Его методы:

• std::optional<RouteInfo> BuildRoute(VertexId from, VertexId to) –

метод построения маршрута. Принимает номера вершин, между

которыми построить маршрут;

• EdgeId GetRouteEdge(RouteId route\_id, size\_t edge\_idx) const –

получить номер ребра в графе по текущему номеру ребра edge\_idx в

маршруте route\_id;

• void ReleaseRoute(RouteId route\_id) – освободить маршрут из

памяти по его номеру.

Транспортный справочник

GeoCoordinates – структура, которая хранит в себе географические

координаты (долготу и широту). Координаты вводятся в градусах.

double ComputeDistanceForCoords(const GeoCoordinates& lhs, const

GeoCoordinates& rhs) – функция, которая принимает две структуры

GeoCoordinates и вычисляет координаты между двумя точками на сфере.

Stop – структура, которая хранит информацию об остановке.

Поля данной структуры:

• GeoCoordinates coords\_ – поле, которое хранит географические

координаты остановки;

• std::set<std::string\_view> buses\_ – поле, хранящее номера

автобусов, которые ездят по данной остановке, в алфавитном

порядке.

Bus – структура, хранящая информацию об автобусе.

Поля:

• Type type\_ – тип автобуса;

• std::vector<ConstStopRawPtr> stops\_ – вектор указателей на пару из

названия остановки, по которой едет данный автобус, и структуры

Stop.

TransportManager – класс, который и является транспортным

справочником. Он хранит в себе информацию об остановках, автобусах, а также

ориентированный нагруженный граф, который помогает при определении

наиболее эффективного по времени пути между двумя остановками.

Его поля:

• RoutingSettings settings – настройки маршрутизации (время

ожидания автобуса на остановке, скорость движения автобусов)

• std::unordered\_map<std::string, Stop> stops\_ – словарь, в котором

ключ – это название остановки, а значение – структура Stop;

• std::unordered\_map<std::string, Bus> buses\_ – словарь, в котором

ключ – это номер автобуса, а значение – структура Bus;

• std::unordered\_map<ConstStopRawPtr,

std::unordered\_map<ConstStopRawPtr, double>> – поле, которое

хранит в себе данные о расстоянии между остановками по дорогам;

• std::unordered\_map<Graph::VertexId, ConstStopRawPtr>

vertex\_id\_to\_stop – словарь, где ключом является номер вершины

графа, а значением – указатель на пару из названия остановки и

информации о ней;

• std::unordered\_map<ConstStopRawPtr, Graph::VertexId>

stop\_to\_vertex\_id, wait\_stop\_to\_vertex\_id – словари, где ключом

является указатель на пару из названия остановки и информации о

ней, а значением – номер вершины;

• std::unordered\_map<Graph::EdgeId, EdgeInfo> edge\_id\_to\_info –

словарь, где ключом является номер ребра графа, а значением –

информация об этой ребра;

• std::unique\_ptr<BusGraph> graph – взвешенный ориентированный

граф;

• std::unique\_ptr<Graph::Router<EdgeWeight>> router – класс

маршрутизатора, который строит самый короткий по времени путь

во взвешенном ориентированном графе.

Public методы:

• ConstStopRawPtr SetStop(std::string stop\_id, Stop stop) – метод

добавления остановки. Возвращает константный указатель на пару

из названия остановки и структуры Stop;

• StopRawPtr GetStop(const std::string& stop\_id) / ConstStopRawPtr

GetStop(const std::string& stop\_id) const – методы получения

неконстантного / константного указателя на пару из названия

остановки и структуры Stop;

• BusRawPtr SetBus(std::string bus\_id, Bus info) – метод добавления

автобуса. Возвращает константный указатель на пару из номера

автобуса и структуры Bus;

• ConstBusRawPtr GetBus(const std::string& bus\_id) const – метод

получения константного указателя на пару из номера автобуса и

структуры Bus;

• ConstStopRawPtr GetStopByVertexId(Graph::VertexId id) const –

метод получения константного указателя на пару из названия

остановки и структуры Stop по номеру вершины графа;

• Graph::VertexId GetVertexIdByStop(const std::string& stop\_id)

const / Graph::VertexId GetVertexIdByWaitStop(const std::string&

stop\_id) const – метод получения номера вершины графа по названию

остановки;

• const EdgeInfo& GetEdgeInfoByEdgeId(Graph::EdgeId id) const –

метод получения информации о ребре графа по номеру вершины;

• const BusGraph& GetGraph() const – метод получения графа по

ссылке;

• const Graph::Edge<EdgeWeight>& GetGraphEdge(Graph::EdgeId

edge\_id) const – метод получения ребра графа по его номеру;

• const Graph::Router<EdgeWeight>& GetRouter() const – метод

получения маршрутизатора;

• Distance ComputeDistanceForBus(const std::string& bus\_id) const –

метод вычисления дистанции, которую проезжает автобус за один

рейс;

• int GetBusWaitTime() const – метод получения времени ожидания

автобуса на остановке;

• double GetBusVelocity() const – метод получения скорости автобуса

в километрах в минуту.

Private методы:

• TransportManager() – конструктор по умолчанию. Так как

транспортный справочник – это сложный объект, то его построение

должно произвестись правильно. С этой целью создан класс

TransportManagerBuilder, который и создает в себе транспортный

справочник и возвращает его.

TransportManagerBuilder – класс, задачей которого является принятие

запросов на построение транспортного каталога, а после, при вызове

соответствующего метода, – его создание. TransportManagerBuilder является

friend-классом для TransportManager, а значит имеет доступ к его полям и

методам, которые имеют модификатор доступа private.

Его поля:

• int wait\_time\_ – время ожидания автобуса на остановке;

• double velocity\_in\_kmph\_ – скорость автобуса (в километрах в час);

• std::vector<Json::Node> stop\_requests\_, bus\_requests\_, distances\_ –

вектора запросов на построение транспортного каталога.

Public методы:

• void SetBusSettings(int wait\_time, double velocity\_in\_kmph) –

установления настроек маршрутизации(времени ожидания автобуса

на остановке и скорости движения автобуса);

• void AddQuery(const Json::Node& node) – добавления запроса на

построение базы данных в формате JSON;

• TransportManager Build() const – метод построения транспортного

каталога.

Private методы:

• void BuildStops(TransportManager& manager) const – метод

добавления остановок в транспортный каталог. Вызывается из

Build();

• void BuildDistances(TransportManager& manager) const – метод

добавления расстояний между остановками в транспортный каталог.

Вызывается из Build();

• void BuildEdgesForBus(TransportManager& manager, const

std::vector<TransportManager::ConstStopRawPtr>& stops,

TransportManager::ConstBusRawPtr bus\_ptr) const – добавление

ребер, соответствующих времени между остановками, по которым

едет автобус, в граф;

• void BuildBuses(TransportManager& manager) const – метод

добавления автобуса в транспортный каталог. Вызывается из Build().

TransportManagerBuilder ParseBaseRequests(const

std::vector<Json::Node>& requests, const std::map<std::string, Json::Node>&

routing\_settings) – функция, которая принимает запросы на построение базы

данных и возвращает объект класса TransportManagerBuilder.

Запросы к транспортному справочнику

Request – базовая абстрактная структура запросов, которая содержит

виртуальные методы запросов к транспортному справочнику. Имеет

единственное поле – номера запроса.

Ее методы:

• Request(int32\_t id) – конструктор, принимающий номер запроса;

• virtual Json::Node Process(const TransportManager& manager)

const = 0 – абстрактный метод выполнения запроса к транспортному

справочнику. Возвращает JSON-сообщение с ответом;

• static std::unique\_ptr<Request> Create (Type type, int32\_t id) –

статический метод создания умного указателя на запрос из его типа

и id;

• virtual void ParseFrom(const std::map<std::string, Json::Node>&

request\_map) = 0 – абстрактный метод инициализации запроса из

JSON;

• virtual ~Request() = default – виртуальный деструктор.

BusInfo – структура запроса к транспортному каталогу, которая при

обращении возвращает информацию об определенном автобусе. Является public

наследником Request и имеет единственное поле std::string name – номер

автобуса.

Методы:

• void ParseFrom(const std::map<std::string, Json::Node>&

request\_map) override – метод, который переопределяет абстрактный

метод базового класса Request и инициализирует свое поле name

номером автобуса из переданного JSON-запроса;

• Json::Node Process(const TransportManager& manager) const

override – метод, который переопределяет абстрактный метод

базового класса Request и выполняет запрос к транспортному

справочнику для возврата информации об автобусе в формате JSON.

StopInfo – структура запроса к транспортному каталогу, которая при

обращении возвращает информацию об определенной остановке. Является

public наследником Request и имеет единственное поле std::string name –

название остановки.

Методы:

• void ParseFrom(const std::map<std::string, Json::Node>&

request\_map) override – метод, который переопределяет абстрактный

метод базового класса Request и инициализирует свое поле name

названием остановки из переданного JSON-запроса;

• Json::Node Process(const TransportManager& manager) const

override – метод, который переопределяет абстрактный метод

базового класса Request и выполняет запрос к транспортному

справочнику для возврата информации об остановке в формате

JSON.

RouteInfo – структура запроса к транспортному каталогу, которая при

обращении строит самый эффективный по времени маршрут между двумя

остановками. Является public наследником Request и имеет два поля std::string

from, to – название остановки между которыми проложить маршрут.

Методы:

• void ParseFrom(const std::map<std::string, Json::Node>&

request\_map) override – метод, который переопределяет абстрактный

метод базового класса Request и инициализирует свои поля

названиями остановок из переданного JSON-запроса;

• Json::Node Process(const TransportManager& manager) const

override – метод, который переопределяет абстрактный метод

базового класса Request и выполняет запрос к транспортному

справочнику для возврата построенного маршрута в формате JSON.

std::unique\_ptr<Request> ParseRequest(const std::map<std::string,

Json::Node>& request\_map) – функция, принимающая JSON-запрос и

возвращающая умный указатель на Request.

std::vector<std::unique\_ptr<Request>> ReadStatRequests(const

std::vector<Json::Node>& stat\_requests) – функция, которая принимает вектор

JSON-запросов и возвращает вектор умных указателей на Request.

Json::Document ProcessRequests(TransportManager& manager, const

std::vector<RequestHolder>& requests) – функция, которая принимает

транспортный справочник и запросы к нему и возвращает ответы на эти запросы.

void PrintResponses(const Json::Document& responses, std::ostream& os)

– функция, которая печатает ответы на запросы в выходной поток