Градски транспорт

(Документация)

Глава 1. Увод

1.1. Описание и идея на проекта

Проектът Градски транспорт има за цел да реализира информационна система за градска транспортна мрежа. Основната задача е да се изчислява най- къс път (по време) от дадени начална спирка, крайна спирка и време на тръгване. За по-лесното решаване на задачата е прието, че всеки автобус от градската транспортна мрежа се движи абсолютно точно по разписание, като не изчезва по време на пътя.

1.2. Цели и задачи на разработка

Основната цел на проекта е да бъде разработена програма, която да поддържа база данни за автобусни спирки и автобуси, които са част от градската транспортна мрежа, и планира пътувания с градския транспорт. Транспортната мрежа се състои от N спирки, номерирани от 0 до N-1.

За реализацията на този проект е нужно данните за автобусните спирки да бъдат представени по максимално най-добрия начин. За целта информацията за автобусна спирка ще бъде представена в отделен клас, който има член-данни за името на спирката, номера на спирката и списъкът от автобусите, които преминават през нея. Тъй като в заданието на самия проект е упоменато, че за всеки автобус, който преминава през дадена спирка е известен график, в който той пристига на спирката. Това е последователност от числа, описващи минути след 0 часа и 0 минути за деня, в който пристига съответния автобус. Това налага реализирането на отделен клас, който да характеризира график на автобус. Този клас ще съдържа данни за номера на спирката и последователността от числа, определящи времето на пристигане на автобуса.

Информацията за всеки автобус също ще бъде представена в отделен клас с член-данни за маршрута на автобуса- последователност от номера на спирки, през които минава автобуса, номер на автобуса, броя спирки и графиците за всяка спирка от маршрута.

За транспортната мрежа ще бъде реализирана с помощта на сруктура от данни тип граф, тъй като тя е подходяща за реализирането на транспортна мрежа. Всеки връх в този граф ще бъде от сложен тип Node, който ще съдържа информация за автобусна спирка и списък от ребра. Ребрата на графа също ще са от сложен тип Edge, който ще има член-данни за автобусна спирка и номер на автобус, с който се достига тази спирка. Класът, който ще представлява транспортната мрежа ще съдържа информация за това какво е ребро, връх и ще има вектор от върхове, както и броя на върховете и броя на ребрата.

За стартирането на самата програма ще бъде нужен още един клас, който да поддържа една транспортна мрежа и всички функции за работа с нея.

Глава 2. Преглед и предметна област

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

За реализирането на проекта е необходимо да бъдат дефинирани класове, които са основен инструмент на ООП и са средство за дефиниране на абстрактни типове данни. За описанието на това, какво са автобусна спирка, автобус, график и транспортна мрежа ще бъдат използвани класове, които ще бъдат свързани помежду си. За някои от член-данните на тези класове ще бъдат използвани структури от данни тип вектор(динамичен масив), двусвързан списък и граф. Графът ще съдържа информация за върхове и ребра. Като връх в този граф, реализиращ транспортна мрежа, ще представлява автобусна спирка и последователност от други върхове, такива че от автобусната спирка има автобус до всяка спирка от този списък с други спирки.

За планирането на пътуване с градския транспорт и търсене на най-къс път по време от една спирка до друга ще бъде използван алгоритъмът на Дейкстра, тъй като той строи дърво на минималните пътища от едни връх да всички останали. Алгоритъмът ще бъде оптимизиран с помощта на структура от данни тип пирамида и така сложността от O(m+n) ще стане O (mlogn), където m е броят на ребрата, а n броят на върховете.

При добавянето на нов курс към разписа на даден автобус, се налага да се сортират графиците с времената на пристигане на автобус на дадена спирка, за да може програмата да работи коректно. За таза цел ще се използва алгоритъмът за пирамидално сортиране, тъй като той има сложност O(nlogn), като знаем, че това е теоритичната долна граница за сложност по време на сортиращ алгоритъм. Този алгоритъм има константна сложност по памет, универсален е и е частичен. Минус на алгоритъма е че не е стабилен, но в случая не е важно за нас да се запазва последователността на еднаквите времена на пристигане.

2.2. Дефиниране на проблем и сложност на поставената задача

Тъй като за всеки автобус е известен график за всяка спирка, през която минава, няма как графикът да бъде просто последователност от числа. Трябва да се запазва и някаква характерна информация за спирката, за която е този график.

Трябва да се избере подходящ формат за съхранение на данните за транспортната мрежа във файл, за който да се имплементира лесен начин за редактиране.

Налага се да се направят инструменти за промяна на транспортната мрежа- например да се изтрива или добавя нова спирка, автобус и т.н. Тъй като автобус, автобусна спирка, график са свързани, ще трябва да бъдат направени инструменти за промяна на данните във всеки един клас от изброените, за да може една такава промяна да бъде отразена правилно.

2.3. Подходи, методи за решаването на поставени проблеми

За да може да бъде реализирана функция, която показва графика на даден автобус по подаден номер на автобус, то ще бъде реализиран отделен клас, който ще характеризира график на автобус. Така всеки автобус ще има списък (динамичен масив) от обекти на този клас. Избран е динамичен масив, защото функцията за вземане на елемент на дадена позиция е константна.

Това ще помогне и за функция, която показва графика на всички автобуси преминаващи през дадена спирка.

За да може лесно да се редактира транспортната мрежа ще има функции, които да позволяват да се добавя или премахва нова спирка от градския транспорт, да се добавя нов автобус, да се проверява дали има директен автобус от една спирка до друга, да се проверява дали изобщо съществува дадена спирка или автобус. Информацията за транспортната мрежа ще се съхранява в двоичен файл, тъй като това ни позволява пряк достъп до данните във файла. При стартиране на програмата ще се изисква да има отворен файл, от който да бъде прочетена информацията за транспортната мрежа, също така ще има функции, които да записват направени промени в градската транспортна мрежа във същия файл, от който е прочетена, или в изцяло нов файл.

2.4. Потребителски и качествени изисквания

За да бъдат изпълнени всички функции за дадена транспортна мрежа е необходимо винаги да има успешно отворен и прочетен файл за транспортна мрежа. След това програмата очаква да се въведе команда, като след въвеждането командата се изпълнява според въведените правила. Списъкът с позволените команди може да се види при въвеждане на команда “help“. Това действие ще продължава до въвеждането на команда “exit”.

Глава 3. Проектиране

3.1. Обща архитектура- ООП дизайн

Първоначално трябва да бъде реализиран класът, който представлява график на автобус. Той притежава член-данни за номер на спирка, който е от тип unsigned, защото номерата на спирките са от 0 до N-1, и последователност от времето на пристигане на автобус на спирката (std::vector<unsigned>). Времето на пристигане е от тип unsigned, защото няма как да бъде отрицателно. За класа е реализирана голяма четворка, която да се погрижи за правилното създаване на обекти от този клас. Като помощни функции са реализирани методи за копиране (void copyArrivalTime(const std::vector<unsigned>& other)) на член-данни, за добавяне на ново време (void addNewArrivalTime(unsigned time)), за вземане на време на пристигане на даден индекс, за вземане на време на пристигане след друго време (unsigned getNextArrivalTimeAfter(unsigned timeBefore)const), за вземане на номера на станцията (unsigned getNumberOfStation()const). Тъй като информацията трябва да бъде записана във файл, класът има реализирани методи за записване и четене от бинарен файл.

Информацията за автобус е представена в отделен клас Bus. Той има член-данни за маршрута на автобуса, номера на автобуса, броя на станциите и списък от график за всяка една от станциите (std::vector<Schedule> m\_sheduleForEachBusStation), през които минава. Класът има реализирана голяма четворка, селектори, мутатори, както и функции за записване и четене в бинарен файл, извеждане на маршрута, добавяне на нова спирка към маршрута, извеждане на графика за дадена станция или за всички станции, през които минава, проверка дали посещава дадена станция, премахване на дадена станция, копиране на данни от друг автобус и изтриване на данните.

Реализиран е и клас за автобусна спирка. Той има член-данни за името на спирката, номера й и списък с автобусите, които минават през нея. Аналогично на останалите два класа и за този е реализирана голяма четворка, селектори и мутатори, помощни функции за четене и писане в бинарен файл, за добавяне на нов автобус, за премахване на автобус, за проверка дали автобус с даден номер преминава през спирката (bool passedABusWithANumber(unsigned numberBus)) и показване на графика на автобусите.

За транспортната мрежа също е реализиран клас (class Transport). Той реализира граф по следния начин:

class Transport {

private:

struct Edge {

BusStation m\_to;

unsigned m\_numberOfBus;// Ребро ще има само, когато ще има автобус, който изпълнява курс от една сприка до другата- to.

Edge(){}

Edge(BusStation to,unsigned numbBus)

:m\_to(to)

,m\_numberOfBus(numbBus)

{}

unsigned getNumberOfBusOfEdge()const {

return m\_numberOfBus;

}

};

struct Node {

std::list<Edge> m\_E;

BusStation m\_station;

Node() {}

Node(BusStation station)

:m\_station(station)

{}

};

std::vector<Node> m\_V;//Множество от върхове- върховете ще са автобусните спирки

unsigned m\_countV;

unsigned m\_countE;

private:

unsigned numberOfBus;//помощна променлива

//Помощна променлива за printScheduleForStation()

unsigned maxCountV;

void swap(unsigned& a, unsigned& b);

public:

Transport():m\_countV(0),m\_countE(0){}

Transport(unsigned v, unsigned e)

: m\_countV(v)

, m\_countE(e)

{

}

static const int DEFAULT\_VALUE\_FOR\_POS = -1;

void printBusesOfStation(unsigned num)const;

unsigned getCountV()const;

unsigned getCountE()const;

BusStation getStation(unsigned numberOfStation);

void setCountV(unsigned newCountV);

void setCountE(unsigned newCountE);

void addStation(const BusStation& station);//add const

void removeStation(unsigned numberOfStation);//to be tested

bool hasStationWithNumber(unsigned numberStation)const;

unsigned countEdges(unsigned numbStation) const;

void addEdge(BusStation& from, BusStation& to, Bus& busFromTo);

bool hasEdgeWithBus(BusStation& from, BusStation& to, unsigned busNumber);

bool hasEdge(BusStation& from, BusStation& to);

BusStation getStation(unsigned numberStation) const;

Bus getBusFromTransport(unsigned numberOfBus);

//Връща минималното време за достигане от станция from до станция to, след пристигане на станция from в час timeToReachBusStation1

int minTimeToReachBusStation(BusStation& from, BusStation& to, unsigned timeToReachBusStation1);

//Връща номера на автобуса, с който за най-бързо време ще достигнем от станция from до станция to

unsigned numberBusWithMinTimeToReachBusStation(BusStation& from, BusStation& to, unsigned timeToReachBusStation1);

//Извеждане на най-къс път по време от зададена начална, крайна спирка и време на тръгване

void printTheShortestPath(BusStation& startBusStation, BusStation& finalBusStation, unsigned timeToStart);

//Показване на разписа на автобусите минаващи през дадена спирка

void printScheduleForStation(unsigned numberOfStation)const;

//Извеждане на номерата на спирките

void printStation()const;

//Проверка дали в транспортната мрежа има автобус с даден номер

bool hasBusInNetwork(unsigned numberBus) const;

//Добавяне на нов разпис за нова станция на автобус от транспортната мрежа

void addScheduleForBusInNetwork(unsigned numberBus, Schedule& sh);

//Работа с файлове

void serializeTransport(std::ofstream& ofs)

void deserializeTransport(std::ifstream& ifs);

//Връща най-големият номер на станция

unsigned biggestNumOfStation()const;

//Изчистване на данните

void clearMemory();

//Добавяне на нов автобус

void addBus(Bus& newBus);

};

Тъй като една от основните задачи на този проект е планирането на пътуване за най-малко време от една станция до друга, се налага дефинирането на няколко помощни функции, които да бъдат приложени в алгоритъма на Дейкстра, който намира най- къс път. Една такава функция е функцията minTimeToReachBusStation(BusStation& from, BusStation& to, unsigned timeToReachBusStation1). Тя изчислява най-малкото време за достигане от начална станция from до крайна станция to при зададено начално време timeToReachBusStation1, което представлява времето на пристигане на пътника на конкретната начална спирка. За да запазим с кой автобус е достигната станцията to е въведена една помощна променлива numberOfBus, която пази този номер. Чрез функцията numberBusWithMinTimeToReachBusStation(BusStation& from, BusStation& to, unsigned timeToReachBusStation1) се взима стойността на променливата numberOfBus.

За стартирането на програмата е реализиран отделен клас Manager. Той има само една член-дана от тип Transport. Класът притежава една публична член-дана, която да пази името на последния отворен файл, така че командата save() да се изпълни за последно отворения файл. Дефинирани са функции за отваряне на файл, затваряне на файл, запазване на текущия файл, запазване на информацията за транспортната мрежа в нов файл, за изход от програмата, за добавяне на нова станция, за премахване на станция, за проверка дали съществува станция с даден номер, за проверка дали има директен път от една станция до друга, като се подават само номерата на станциите, извеждане на най-късият път от една спирка до друга и подадено начално време на тръгване, за извеждане на списъка с автобуси минаващи през дадена спирка, за показване на разписа на автобуси, минаващи през дадена спирка, за добавяне на нов автобус, за проверка дали има автобус с даден номер и функция, която стартира самата програма. Декларациите на функциите са:

//Помощни средства за работа с файл

void open(char\* fileName);

void closeFile();

void save();

void saveAs(char\* newFileName);

//Помощна спирка, която извежда командите

void help()const;

//Добавяне на нова спирка

void addStation(const BusStation& station);

//Премахване на спирка

void removeStation(unsigned numberOfStation);

//Има ли станция с даден номер

bool hasStationWithNumber(unsigned numberOfStation);

//Има ли автобус от една спирка да друга- директно без прикачване

bool hasEdge(unsigned numberOfFirstStation, unsigned numberOfSecondStation);

//Минималния по време път от една спирка до друга

void printShortestPath(unsigned numberFromStation, unsigned numberToStation, unsigned startTime);

//Показва списъка с автобуси, минаващи през дадена спирка

void printListOfBuses(unsigned numberOfStation)const;

//Показва разписа на автобусите минаващи през дадена спирка

void printScheduleForStation(unsigned numberOfStation) const;

//Добавяне на нов автобус

void addNewBus(Bus& newBus);

//Добавяне на нов автобус

void addNewCourse(unsigned numBus);

//Проверка дали има автобус в мрежата

const bool hasBus(unsigned numBus) const;

void runProgram();

Глава 4. Реализация, тестване

4.1. Реализация на класове

За да бъдат правени промени в транспортната мрежа са дефинирани функции за добавяне на нова станция(void addStation(BusStation& station)), за изтриване на станция (void removeStation(unsigned numberOfStation)), добавяне на нов курс (void addNewCourse(unsigned numBus)), добавяне на нов автобус (void addNewBus(Bus& newBus)). Под добавяне на нов курс за даден автобус се налага промяна на списъка с графици за всяка спирка от маршрута на автобуса. Добавят се нови времена на пристигане на всяка една от станциите и след това се сортират, за да не се наруши подредбата на пристигане относно времето. При добавянето на нов курс е направена валидация новите времена да бъдат в рамките на един ден, но се разчита и на това да се въвеждат смислени данни от потребителя. За да бъдат правилно отразени тези промени се налага и дефиниране на функции за промяна на данните в класовете BusStation и Schedule. Подробност за всички тези функции може да се види в кода на проекта, като в коментари са добавени пояснения.

4.2. Управление на паметта и алгоритми. Оптимизации

За да се подобри сложността на алгоритъма на Дейкстра се налага да се дефинира структура от данни тип пирамида. Използвайки пирамида в реализацията на алгоритъма ще подобри сложността от O(m+n) на O(mlogn), където m е броят на ребрата, а n броят на върховете. За изчислението на най-къс път в алгоритъма ще се използва помощна функция minTimeToReachBusStation(BusStation& from, BusStation& to, unsigned timeToReachBusStation1), която е част от класа Transport. Тази функция изчислява минималното време за пътуване от станция from до станция to при зададено време на пристигане на станция from, но задължително между двете станции трябва да има поне един автобус. Под минимално време за достигане на станция to се разбира най-малкото време за път, което включва и времето за изчакване, възможно е времето за изчакване да е 0. Знаем, че времето за престой на автобусите е 0. Също така времето за слизане и качване на пътник е 0, т.е ако два автобуса пристигнат в едно и също време, то без проблем пътникът може да се прикачи от един автобус на друг.

При създаването на класовете Schedule, Bus, Transport, BusStation, Manager бяха

необходими константи за дължини на имена, команди, номера на позиции, капацитет, начални стойности и др. Ако бяха дефинирани като глобални константи за цялата програма, то е възможно тези константи да бъдат променени и да се наруши смисъла и реализацията на програмата. За това, когато са необходими константи за създаването на обект от даден клас, то в този клас константите са статични и по този начин те са “глобални„ за класа, но няма външен достъп до тях и не могат да бъдат променени.

4.3. Планиране, описание и създаване на тестови сценарии

По време на създаване на класовете в програмата бяха създадени тестове, които да проверяват дали правилно се създават обекти от различните класове, дали методите им работят правилно, дали е направена пълна валидация на данните, които постъпват преди да се създаде обект от даден клас. Всички тестове се намират във файл с име „Tests.hpp“.

За класа Bus е важно да бъдат тествани дали правилно работят конструкторите, мутаторите, селекторите, функцията за добавяне на нова спирка към маршрута, за показване на маршрута, за показване на разписа за дадена станция, за показване на разписа за всички станции, за премахване на станция, за записване на автобус във бинарен файл и за четене от бинарен файл.

Аналогично за Schedule да бъдат направени същите тестове, като от основно значение са методите са сериализация и десериализация.

void testSchedule(){

Schedule sh;

std::cout << "Scheduele for station: " << sh.getNumberOfStation() << std::endl;

sh.addNewArrivalTime(12);

sh.addNewArrivalTime(15);

sh.addNewArrivalTime(30);

sh.addNewArrivalTime(60);

sh.addNewArrivalTime(70);

sh.addNewArrivalTime(79);

sh.addNewArrivalTime(87);

sh.addNewArrivalTime(99);

sh.printSchedule();

std::cout << "Time in index 7: " << sh.getArrivalTimeOfIndex(7) << std::endl;

std::cout << "Time in index 1: " << sh.getArrivalTimeOfIndex(1) << std::endl;

std::cout << "Time in index 5: " << sh.getArrivalTimeOfIndex(5) << std::endl;

Schedule sh2 = sh;

std::cout << sh2.getNumberOfStation()<< std::endl;

std::cout << sh2.getArrivalTimeOfIndex(7) << std::endl;

sh2.printSchedule();

sh2.setNumberOfStation(2);

std::cout << "number st of sh2: " << sh2.getNumberOfStation()<<std::endl;

sh2.addNewArrivalTime(101);

sh2.printSchedule();

Schedule sh3;

sh3 = sh2;

std::cout << "Print info of sh3" << std::endl;

sh3.printSchedule();

Schedule sh2 = sh;

std::cout << sh2.getNumberOfStation()<< std::endl;

std::cout << sh2.getArrivalTimeOfIndex(7) << std::endl;

sh2.printSchedule();

sh2.setNumberOfStation(2);

std::cout << "number st of sh2: " << sh2.getNumberOfStation()<<std::endl;

sh2.addNewArrivalTime(101);

sh2.printSchedule();

Schedule sh3;

sh3 = sh2;

std::cout << "Print info of sh3" << std::endl;

sh3.printSchedule();

std::cout << "Before write to file: "<<sh.getNumberOfStation() << std::endl;

sh.printSchedule();

std::ofstream ofs("Schedule.bin", std::ios::out | std::ios::binary);

sh.serializeSchedule(ofs);

ofs.close();

std::ifstream ifs("Schedule.bin", std::ios::in | std::ios::binary);

Schedule schdeuleRead;

schdeuleRead.deserializeSchedule(ifs);

ifs.close();

std::cout << "After read file: "<<schdeuleRead.getNumberOfStation() << std::endl;

schdeuleRead.printSchedule();

}

Аналогично за всички останали класове са тествани конструкторите, мутаторите, селекторите, методите за четене и писане в бинарен файл и всички останали помощни функции.

Класът, който стартира програмата също е тестван и по време на изпълнение с подадени от потребителя данни.

Глава 5. Заключение

5.1. Обобщение на изпълнението на началните цели

Началните цели на проекта са изпълнени, използвайки знанията от курсовете по УП, ООП и СДП.

5.2. Насоки за бъдещо развитие и усъвършенстване

В бъдеще проектът може да бъде усъвършенстван. Възможно е да се добавят видове потребители и определени функции да се изпълняват само при наличието на потребител отговарящ за транспортната мрежа. Например функциите за добавяне и премахване на автобуси, спирки, графици да не могат да се изпълняват, когато в системата е влязъл обикновен потребител. Обикновените потребители ще имат ограничени права, те ще могат само да следят разписите, движенията на автобусите и всички функции, които не променят изградената транспортна мрежа да са валидни за тях. Възможно е също да се добавят нови инструменти за премахване на курс от вече съществуващи автобуси и спирки. Също така да се поправи проблемът с извеждането на графиците на автобуси за дадена спирка след изтриване на дадена станция от градската мрежа, тъй като в последната версия на проекта след изтриване на дадена спирка от транспортната мрежа и след извеждане на графика се извежда и графика на автобус от вече изтритата станция.