Subway - Optimalizace jízdních řádů pomocí diskrétní simulace

Zadání

Cílem programu *Subway* je na základě vstupních dat, které se skládají z linek dopravy, frekvence jednotlivých stanic, vzdálenosti mezi nimi, počtu cestujících apod., optimalizovat jízdní řád tak, aby byl potenciál dopravního prostředku ideálně využit.

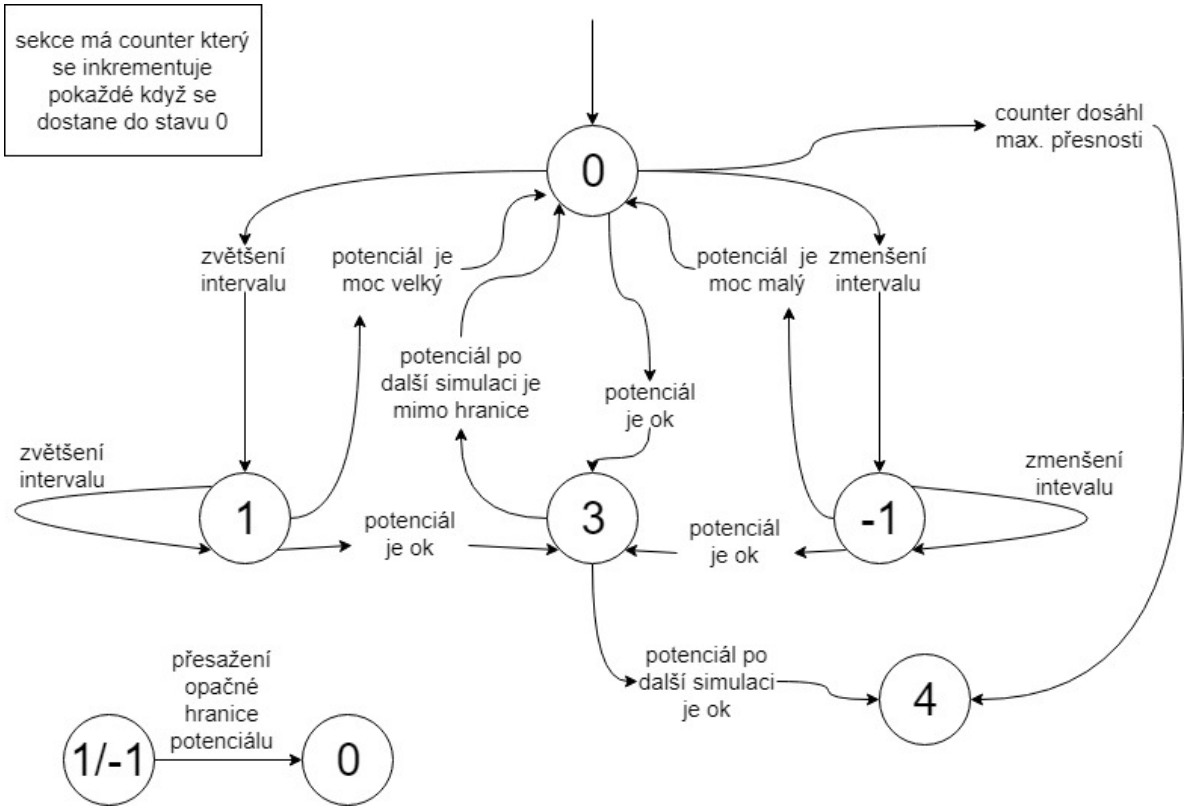
Řešení

Algoritmus výroby ideálního jízdního řádu

Celá doba provozu je v programu rozplánovaná do sekcí. Na začátku je pro každou linku pouze jedna sekce. Sekce jednotlivých linek jsou nezávislé. Každá sekce má uvnitř nastaveno v jakých intervalech se budou v dané sekci vypouštět vlaky na trať. Simulace pak probíhá popořadě podle všech sekcí. Každý vypuštěný vlak si zapamatuje v jaké sekci byl vypuštěn na trať a pak pravidelně aktualizuje potenciál dané sekce, avšak sekce si vždy zapamatuje maximum. Tudíž na začátku simulace každá sekce začíná s potenciálem = 0 a velkým intervalem a v průběhu simulace si zapamatuje maximální potenciál ze všech vlaků vypuštěných v dané sekci.

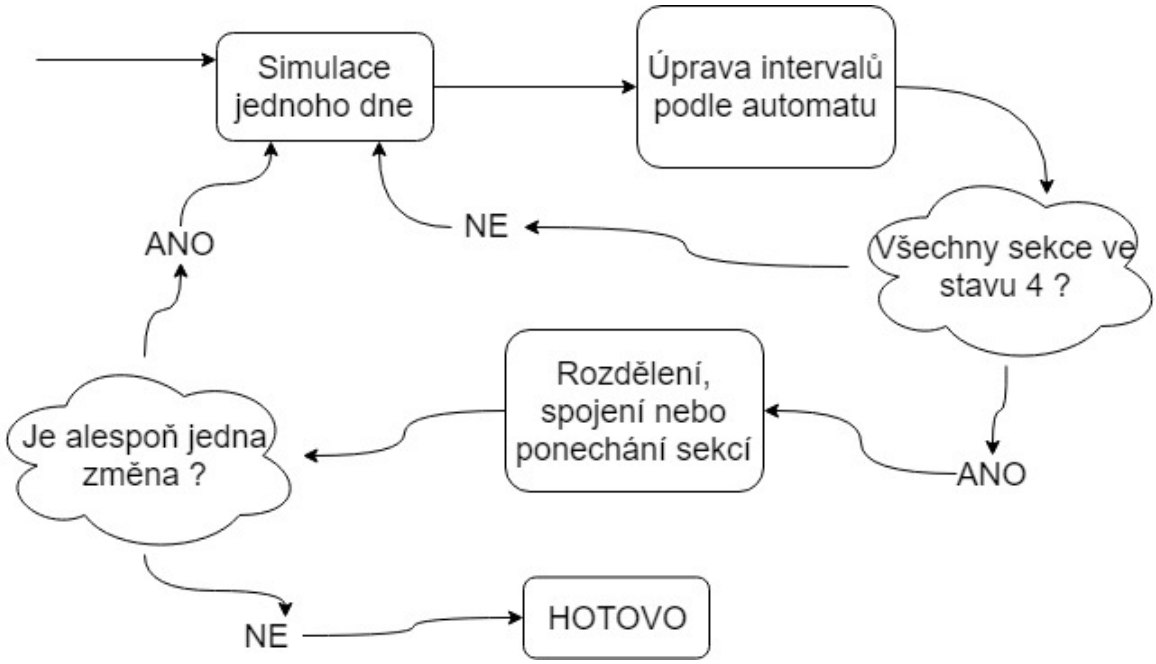
> potenciál = počet cestujících ve vlaku / kapacita vlaku, potenciál ∈ <0,1>

Po doběhnutí simulace jednoho dne se projdou všechny sekce a zkontroluje se, zda-li jsou potenciály sekcí v mezích daných uživatelem. Podle toho zda je nebo není potenciál v rámci mezí se upraví interval vlaků v dané sekci a spustí se další simulace. Každý stav má svůj vnitřní stav, který odpovídá stavu ve stavovém automatu (viz obrázek), který znázorňuje úpravu intervalů.



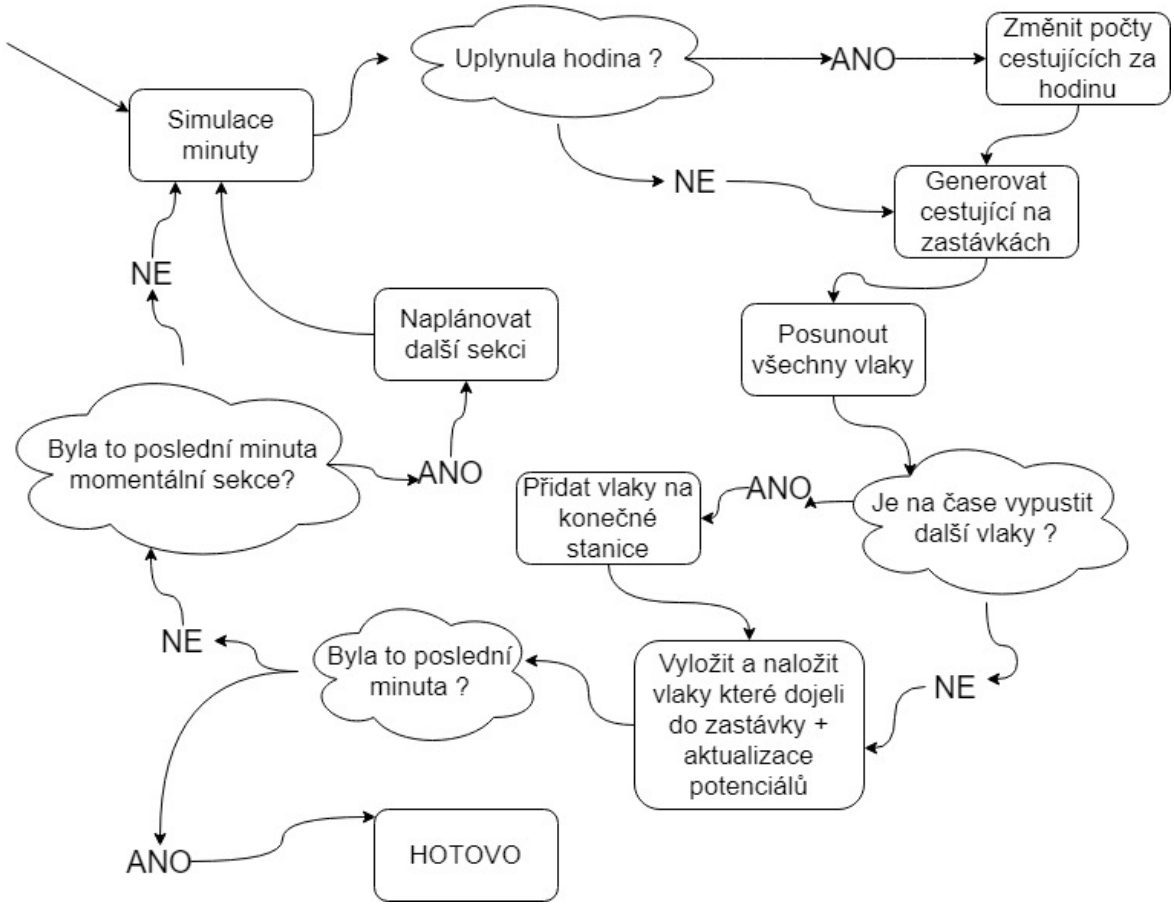
Součástí vstupu je i přirozené číslo určující přesnost. Toto číslo určuje kolikrát každá sekce může navštívit stav 0. Ve chvíli, kdy jsou všechny sekce ve stavu 4, se všechny sekce, které jsou větší než dvojnásobek svého intervalu rozdělí na poloviční. Ze zbylých sekcí ty, které skončily ve stavu 4, s potenciálem nižším než spodní hranice, se spojí s jednou následující aby se v nich mohl zvětšovat interval. Interval totiž nesmí přesáhnout velikost sekce. Zbytek zůstane takový jaký je. Všem se okopíruje původní interval vlaků a vynuluje potenciál na další simulaci.

Tento cyklus se stále opakuje dokud u všech linek se nezmění ani jedna sekce. Pak algoritmus skončí a vydá nový soubor s ideálními intervaly, které odpovídají intervalům z poslední simulace.



Algoritmus simulace dne

Délka jednoho dne odpovídá délce provozu, která je součástí vstupu. Nejmenší jednotkou času branou v potaz jsou celé minuty, tedy i přesnost jízdních řádů je omezená na celé minuty. Zároveň simulace jednoho dne (doby provozu) probíhá po minutách. V každé minutě se odehraje sled událostí popsaných v následujícím obrázku.



Pokud uplynula už celá hodina, aktualizují se na zastávkách počet cestujících kteří přicházejí každou minutu, poněvadž počty cestujících jsou dány 1) frekvencí zastávky, 2)počtem cestujících které linka za danou hodinu přepraví. Frekvence zastávky je přirozené číslo od 1 do 10, kde vyšší číslo znamená frekventovanější zastávka. Každá zastávka si pamatuje kolik cestujících má každou minutu přibýt. Každou minutu se toto množství cestujících vygeneruje na každé zastávce a to sice náhodně, ale statisticky v poměru frekvencí všech zastávek na lince.

Po vygenerování dalších cestujících se pohne všemi vlaky ve směru jejich jízdy. Pokud na dané lince vzhledem k dané sekci je čas vypustit další vlak tak se přidají vlaky na obě koncové stanice linky. Pak se u všech vlaků zkontroluje jestli právě nedorazili do zastávky. Ty které ano, se "vyloží/naloží", respektive vystoupí všichni cestující mířící do dané stanice a nastoupí maximum cestujících z dané zastávky mířící daným směrem.

Tím končí samotná simulace, ale ještě je zde režie sekcí. Pokud momentální sekce dosáhne své poslední minuty, musí se přeplánovat na další sekci. To znamená, že se přenastaví především intervaly vlaků. Pokud má nová sekce kratší interval než předchozí sekce a poslední vlak nejel déle než je interval nové sekce, vyjede okamžitě.

Technická dokumentace

Třídy

Důležité v celém programu je, že se nepracuje přímo s instancemi jednotlivých tříd, nýbrž s odpovídajícími shared pointery.

Line

```
class Line {
public:
    Line(int i, int num, std::vector<int> passangers);
    int GetID() { return id; }
    int GetnumberOfStations() { return numberOfStations; }
    std::deque<StationPtr> stations;
    std::deque<TrainPtr> onTheWayFront;
    std::deque<TrainPtr> onTheWayBack;
    std::vector<int> amountOfPassangers;
    std::vector<int> probabilityMap;
private:
    int id;
    int numberOfStations;
};
```

Třída *Line* reprezentuje linku metra (nebo jiného dopravního prostředku). Má tedy seřazený seznam všech zastávek. Pak právě *Line* udržuje seznam všech vlaků na lince. To je uloženo ve dvou dvousměrných frontách (stačila by obyčejná fronta), každá pro jeden směr jízdy. Je to tak proto, aby se dalo v konstantním čase přidávat a odebírat vlaky. Dále si drží *vector* počtu cestujících přepravených v daných hodinách provozu. A nakonec *vector* "probabilityMap". Ten má tolik políček kolik je součet všech frekvencí stanic napříč celou linkou. Když se pak generuje směr pro cestujícího, stačí vygenerovat náhodné číslo od 1 do počtu políček "probabilityMap" (odečíst jedničku) a zaindexovat ve vectoru. Tím se náhodně vygeneruje směr cestujícího.

Station

```
class Station {
public:
    Station(std::string s, int freq, bool transf, int numberOfStations);
    void AddPassengers(std::vector<int> & probability, int passengersAmount);
    int GetFrequency() { return frequency; }
    int id;
    std::string GetName(){ return name; }
    bool IsTransferStation() { return transferStation; }
    StationPtr next;
    int nextDistance;
    StationPtr prev;
    int prevDistance;
    std::set<std::pair<LinePtr, StationPtr>> transfers;
    std::set<int> transfersToResolve;
    int passengersPerMinute;
    std::map<int, int> waiting;
private:
    int frequency;
    bool transferStation;
    std::string name;
};
```

Třída *Station* si především drží odkazy na vedlejší stanice a vzdálenosti na další stanici. Potom také seznam přestupových stanic, u těch stanic, které přestupové jsou. Množina *transfersToResolve* je zde jen z důvodu parsování. Seznam přestupových stanic je samozřejmě uchován ve formě odkazů, jenže v době parsování jedné stanice nemusí být nutně vytvořená instance ekvivalentní přestupové stanice. Důležitou vlastností třídy je také mapa čekajících cestujících *waiting*, kde klíč odpovídá id nějaké stanice na lince a hodnota odpovídá počtu cestujících mířící na danou stanici. Pak už jde jen o jednoduché vlastnosti jako např. id, frekvence, boolová hodnota říkající jestli je stanice přestupová atd.

Funkce:

AddPassengers

Funkce *AddPassengers* přidá do stanice *passengersAmount* nových cestujících, kteří jsou náhodně namířeni na nějakou další stanici na lince. Pro každého cestujícího se zvlášť vygeneruje náhodné číslo od 0 do součtu všech frekvencí na lince a podle toho do jakého intervalu náhodné číslo padne, tam se i cestující namíří. Pokud by cestující měl dojet na momentální stanici, proces se opakuje.

Train

```
class Train {
public:
    Train(int maxCapacity, TimeSectionPtr timeSection, bool forward, StationPtr
startingStation, int distance);
    void Move() { remainsToNext--; }
    void PassangersOnOff();
    bool MovingForward() { return forwardDirection; }
    double GetPotential() { return ((double)passengersCount / capacity); }
    int FreeSpace() { return capacity - passengersCount; }
    StationPtr station;
    TimeSectionPtr start;
    int remainsToNext;
private:
    void GetOn();
    void GetOff();
    bool forwardDirection;
    int passengersCount;
    std::map<int, int> passengers;
    int capacity;

};
```

Nejdůležitější a nejneintuitivnější vlastností vlaku je odkaz na instanci třídy *TimeSection*. Ten totiž udává, během které sekce byl vlak vypuštěn a tedy kde se má potenciálně upravit interval. Odkaz na stanici *station* je odkaz na stanici kam vlak právě míří. Ve chvíli kdy se vlak vyloží a naloží, nastaví se následující stanice ve směru nebo se vlak z linky odstraní. To však provádí instanční funkce *ServiceTrains* třídy *Scheduler*. Vlastnost *remainsToNext* počítá kolik minut zbývá, než vlak dojede do další stanice. Mapa pasažérů je ekvivalentem čekajících na zastávce. Zbytek vlastností je v celku intuitivní.

Funkce:

PassangersOnOff

Jedinou funkcí je, že nejdříve zavolá *GetOff* a pak *GetOn*.

GetOff

Díky odkazu na stanici na kterou vlak směřuje, respektive do ní už dojel, si jen podle id stanice podívá do mapy cestujících, která má na pozici klíče právě id stanice a vyloží z vlaku všechny cestující na daném id.

GetOn

Funkce nechá nasednout ze stanice maximální možné množství cestujících směřujících stejným směrem jako vlak. Pak tomu přizpůsobí daní vlastnosti vlaku.

Parser

```
class Parser {
public:
    Parser(std::string file) : fileName(file) { ifs = std::ifstream(file); }
    std::pair<int, std::shared_ptr<std::map<int, LinePtr>>>  ParseInputFile();
private:
    std::shared_ptr<Line> ParseSubwayLine(int hours);
    void ReadLine();
    std::string ReadWord();
    StationPtr ParseStation(int numberOfStations);
    void ResolveTransfers(std::map<int, LinePtr> & subway);
    std::ifstream ifs;
    std::stringstream ss;
    std::string fileName;
};
```

Parser v konstruktoru dostane jméno výchozího souboru a má pouze za úkol ho přečíst a naparsovat do připravené datové struktury. Formát vstupních dat je striktně daný a pokud se nedodrží, může program kdekoliv spadnout nebo doběhnout s nesmyslným výstupem.

Funkce:

ParseInputFile

Funkce vrací seznam dvojic <id linky, ukazatel na linku>. Postupně si naparsuje režijní data ze vstupního souboru, např. počet linek apod. Pak podle nich rekurzivně volá *ParseSubwayLine*. Když jsou všechny linky naparsované, tak ještě vytvoří provázání mezi stanicemi, vyplní u všech linek "probabilityMap" a nakonec zavolá *ResolveTransfers*, aby se vytvořilo propojení mezi přestupními stanicemi.

ParseSubwayLine

Stejně jako *ParseInputFile* si nejdříve přečte a naparsuje nějaké řídící hodnoty a pak podle nich parsuje linku metra. Po tom co vytvoří instance všech stanic na lince pomocí *ParseStation*, ještě nastaví vzdálenosti mezi nimi.

ParseStation

Funkce naparsuje stanici a stejně jako v celém parsování i tato funkce má zaručené správné fungování pouze v případě, že jsou striktně dodržena pravidla pro vstup.

ReadLine

Funkce pouze přečte řádek a vloží ho do string streamu *Parseru*.

ReadWord

Vrací slova oddělená z přečtené řádky. Pokud dojde nakonec, vrací "\n".

ResolveTransfers

Funkce má za úkol pouze projít všechny stanice a vytvořit propojení mezi přestupovými stanicemi. Aby toto fungovalo, musí mít ekvivalentní přestupové stanice stejná jména. Také je tato funkce díky mnoha do sebe vnořeným cyklům velmi neefektivní, ale to nás netrápí, protože se to stane pouze jednou v běhu celého programu a předpokládá se že ne na nijak velkých datech.

TimeSection

```
class TimeSection {
public:
    TimeSection(int length, int startingInterval) : sectionLength(length),
currentInterval(startingInterval) , potential(0), state(0), cycle(0),
merged(false) {}
    int GetSectionLength() { return sectionLength; }
    double potential;
    int currentInterval;
    int state;
    int cycle;
    bool merged;
private:
    int sectionLength;
};
```

Fungování a účel třídy *TimeSection* je v podstatě popsána v algoritmu programu. Většina jejích vlastností je hlavně řídících vzhledem ke korekci intervalů :

- *state* - stav ve stavovém automatu popsaném výše
- *cycle* - kontroluje kolikrát si sekce už prošla stavem 0
- *merged* - pokud se nějaká sekce spojila s následující za účelem snížení intervalů již jí nedovolím se znovu rozdělit

Scheduler

```
class Scheduler {
public:
    Scheduler(int hours, LinePtr currLine, std::vector<TimeSectionPtr>
sections);
    void SimulateMinute();
    void AnulateScheduler();
    std::vector<TimeSectionPtr> timeSections;
    int timeSectionsIndex;
    int GetLineId() { return lineId; }
    bool IsEnd() { return (currentTime + 1 == dayLength); }
private:
    void DistributePassengers(int passengers);
    void GeneratePassengers();
    void MoveTrains();
```

```
    void AddTrains();
    void ServiceTrains();
    void ScheduleNextTimeSection();
    int lastTrain;
    int endOfCurrentTimeSection;
    int dayLength;
    int currentTime;
    int frequencySum;
    LinePtr line;
    int lineId;
};
```

Třída *Scheduler* je nosná třída celého programu co se týče simulace. V *Scheduleru* se odehrává veškerá logika simulace. Pro každou linku metra je právě jedna instance *Scheduleru* a drží si k lince veškeré potřebné informace. Například jak je to dlouho co jel poslední vlak, kdy končí momentálně naplánovaná sekce, jak dlouhá je doba provozu, momentální čas a momentální naplánovanou *TimeSection*.

Zároveň je to nakonec i datonosná třída, poněvadž právě *Schedulery* si drží odkazy na linku metra a na všechny *TimeSection*s .

Co se funkcí třídy *Scheduler* týče, tak všechny jsou to funkce rozbíhající simulaci minuty, respektive všechny jsou volané právě z *SimulateMinute*, krom jí samotné a krom *AnulateScheduler*.

Funkce:

SimulateMinute

Funkce přesně podle výše popsaného schématu postupně volá funkce popsané níže. Zároveň jsou v ní implementovány všechny podmínky také popsané ve schématu. Například jestli je již na čase vypustit další vlaky na trať.

AnulateScheduler

AnulateScheduler je pouze pomocná funkce, která je volána pokaždé před simulací dalšího dne. Jejím úkolem je pouze vynulovat všechny atributy (vlastnosti) *Scheduleru*, tak aby byl znova připravený na simulaci. Dalo by se to nahradit vytvářením nových *Schedulerů*, ale to by patrně bylo časově i prostorově složitější.

DistributePassengers

Funkce je volaná na začátku každé hodiny, aby jednotlivým stanicím nastavila počet cestujících, kteří mají během dané hodiny přijít na zastávku. Pouze se vezme počet cestujících, které linka během dané hodiny přepraví a vydělí se *frequencySum*, což je součet frekvencí všech stanic linky. Pak se na každou stanici během hodiny přibyde *n*-násobek vypočtené hodnoty, kde *n* je frekvence dané stanice.

GeneratePassengers

Funkce pouze projde celou linku a na každé stanici zavolá její instanční metodu *AddPassengers* a nechá přidat tolik cestujících, kolik je ve stanici uloženo jako *passengersPerMinute*. Volitelný počet přidaných cestujících je zde, protože funkce je využita krom generování cestujících každou minutu, ještě při přestupu cestujících na přestupových stanicích.

MoveTrains

Postupně projde všechny vlaky jedoucí v jednom směru a pak v opačném směru a zavolá na nich instanční metodu *Move*, která je posune o jedna.

AddTrains

Vytvoří dvě nové instance *Train* v navzájem opačných směrech a vloží je do *onTheWayFront* respektive do *onTheWayBack*.

ServiceTrains

ServiceTrains je metoda, ve které probíhá většina řídících mechanismů při nástup a výstupu cestujících, respektive při plánovaní příští stanice u vlaku.

Funkce postupně projde všechny vlaky v obou směrech a ty vlaky které mají *remainsToNext* == 0, nechá v stanici uložené v instanci vlaku ve vlastnosti *station*, vystoupit/nastoupit cestující. K tomu slouží *PassangersOnOff*. Po nástupu/výstupu cestujících aktualizuje potenciál sekce, ke které je daný vlak vztažený. Potom ještě nastaví příští stanici. To výrazně ulehčují odkazy stanic na obě vedlejší stanice. Nakonec pokud příští stanice == *NULL*, tak vlak vymaže, protože to znamená, že dojel na konečnou. To se zvládá v konstantním čase díky použití fronty (respektive dvousměrné fronty).

ScheduleNextTimeSection

Pokud nastane během simulace čas, kdy končí momentální naplánovaná *TimeSection*, musí se naplánovat další. To se zařídí pouze

1. inkrementací vlastnosti *timeSectionsIndex*, která určuje kolikátá *TimeSection* je na řadě
2. výpočet a nastavení *endOfCurrentTimeSection*, což říká kdy nová *TimeSection* končí

Implementace algoritmu pro vytvoření jízdního řádu

Implementaci algoritmu pro vytvoření ideálního jízdního řádu zařizují **ne**instanční metody v souboru *main.cpp*, to sice:

- CreateTimeTable
- IntervalCorrectionAutomat
- SplitTimeSections
- DecreaseInterval
- IncreaseInterval
- SimulateDay
- WriteOutput

Vše samozřejmě začíná v metodě main. Tam se nejprve zpracují vstupní argumenty, pokud nastane při jejich zpracování výjimka nebo pokud je špatný počet argumentů, program skončí. Poté main nechá naparsovat vstupní soubor. Naparsovaná data potom ještě vloží do instancí *Schedulerů*, které se v main vytvoří. Potom se spustí celý algoritmus funkcí *CreateTimeTable* a nakonec pokud je zapnutý výpis do souboru, což defaultně je, zapíše ideální jízdní řád do souboru pomocí funkce *WriteOutput*.

CreateTimeTable

Funkce pracuje na dvou cyklech. Vnější cyklus se opakuje dokud nedoběhne algoritmus tvorby ideálního jízdního řádu, respektive dokud se alespoň jedna *TimeSection* změní. Vnitřní cyklus se opakuje dokud všechny *TimeSection*s nejsou ve stavu 4. Uvnitř cyklů dochází k opakované simulaci jednoho dne provozu metra. Před každou další simulací se ještě vynulují *Schedulery* jejich instanční funkcí *AnulateScheduler*. Naopak po každé simulaci se projdou všechny *TimeSection*s a pokud již nejsou ve stavu 4, zavolá se na ně funkce *IntervalCorrectionAutomat*, která implementuje

stavový automat. Po dokončení vnitřního cyklu se zavolá funkce *SplitTimeSections*, která napůlí, spojí nebo nechá *TimeSection*s. Pokud metoda mimo jiné vrátí false, znamená to, že něco změnila, tudíž se vnější cyklus opakuje. Po skončení obou cyklů se ještě provede jedna konečná simulace, která jen spočítá konečné potenciály (je tam už jen pro případnou kontrolu).

IntervalCorrectionAutomat

Fuknce je pouhou implementací stavového automatu který je výše důkladně popsán. Implementovaná je pomocí switche. Rekurzivně pak volá metody *DecreaseInterval* a *IncreaseInterval*, které zvětšují nebo zmenšují intervaly. Zároveň obsahuje ochranu proti zacyklení, spočívající v tom, že pokud je interval už minimální, tj. 1 a chtěl by se stále snižovat nebo pokud je dlouhý jak jeho *TimeSection*, tak se skece rovnou nastaví do stavu 4.

SplitTimeSections

Metoda je znovu velmi jednoduchá. Pouze dělá nové instance všech sekcí. Pokud to lze tak je napůlí. Pokud potřebují kvůli nevyužitému potenciálu dvě sekce spojit tak je spojí a jinak jsou nechány sekce v původním stavu. Kritérium pro rozdělení sekce je podmínka, že v každé sekci musí vyjet vlak, respektive že délka steré sekce musí být alespoň dvojnásobná vůči starému intervalu. Nové dvě sekce podědí interval po staré a jinak jsou úplně nové v tom smyslu, že se nacházejí poprvé ve stavu 0. Pro spojování sekcí to je stejné, jen se nastaví příznak *merged*, který sekci již zakazuje se dělit.

DecreaseInterval

Pokud interval sekce je == 1, rovnou se její stav nastaví na 3 a vrátí se tak jak je, poněvadž menší interval nelze. Jinak interval vždy napůlí. Pokud je původní interval lichý, vezme se horní celá část.

IncreaseInterval

Zvedá délku intervalu tak, aby stále platilo pravidlo, že v každé sekci musí vyjet alespoň jeden vlak. Interval se buď zdvojnásobuje nebo se inkrementuje pouze o jedna.

SimulateDay

Jedinou úlohou funkce je simulace celého dne provozu na všech linkách. Prochází tedy popořadě stále dokola všechny *Schedulery* a na každém zavolá *SimulateMinute*. Cyklus cyklí, dokud neskončí provozní den.

WriteOutput

Pouze zapíše do souboru s názvem podle vstupního souboru s příponou ".out". Pro každou linku předpokládá počáteční čas 00:00 a k tomu přičítá intervaly.

Uživatelská příručka

Používání

Program se otevře s příkazové řádky s 5 argumenty:

1. cesta ke vstupnímu souboru
2. spodní hranice potenciálu, kde číslo je reálně číslo od 0 do 1
3. horní hranice potenciálu, kde číslo je reálně číslo od 0 do 1
4. přesnost výpočtu, kde vstupem je přirozené číslo od 1; doporučená hodnota je 5 - 10
5. kapacita jedné vlakové soupravy, přirozené číslo; pro představu souprava pražského metra má udávanou kapacitu cca1500 cestujících

Program posléze nějakou dobu poběží. Pro představu na dvou pražských linkách s relativně reálnými hodnotami, pouze dvouhodinovým provozem a přesností 10 mi program doběhl zhruba po 45 sekundách.

Je nutné přené dodržení formátu vstupních dat, viz níže. Pokud by formát nebyl dodržen není zajištěn správný běh programu ! Na konci běhu se pak výstup zapíše do souboru pojmenovaným stejně jako vstupní soubor, pouze s přidanou koncovkou ".out". Jedná se pouze o textová data.

Vstupní soubor

Vstupní soubor je pouze textový soubor popisující linky metra a počty cestujících. Všechny "slova" jsou jednoduše oddělena mezerou. Na první řádce souboru se udávají pouze dvě čísla:

1. počet linek na vstupu
2. počet hodin provozu

př.:

```
2 3
```

Toto odpovídá 2 linkám a 3-hodinovému provozu.

Po těchto dvou číslech už se na dalších řádkách opakují linky, kterých je tolik, kolik bylo výše napsáno. Popis jedné linky zabere 4 řádky.

Linka - formát

Každá linka je popsaná 4-mi řádky, kde vždy je vše odděleno mezerou:

1.řádek

Na první řádce se nachází dvě čísla:

1. jedinečné **id linky** ve formátu přirozeného čísla
2. počet zastávek dané linky

př.:

```
1 5
```

Toto odpovídá lince s identifikátorem 1 a se 5 zastávkami.

2.řádek

Na druhé řádce se nachází **n** čísel, kde n je doba provozu. Respektive pro každou dobu provozu jedno číslo. Číslo určuje kolik cestujících daná linka v danou hodinu provozu celkem přepraví.

př.:

> 25400 50100 55000

Takto vypadá vstup pro 3-hodinový provoz. Pokud by byla doba provozu např 24 hodin, muselo by zde být 24 oddělených přirozených čísel.

3.řádek

Na 3. řádku se nachází popis všech stanic linky. Každá stanice se skládá ze tří komponent :

1. jméno stanice bez mezer
2. přirozené číslo od 1 do 10 udávající frekvenci zastávky, kdy vyšší číslo znamená vyšší frekvenci
3. přestupy:

-> Pokud je stanice nepřestupová, tak se tam napíší pouze prázdné závorky bez mezery: "()"

-> Pokud je stanice přestupová, tak se dovnitř závorek napíší identifikátory linek, na které zde lze přestoupit. Vše je oddělené mezerou i číslo a závorky : "(2 3)"

Důležité je, že pokud je nějaká stanice přestupová, musí na lince kam lze přestoupit, být stejně se jmenující zastávka.

Po takto definované jedné stanici, následuje další stejně definovaná, oddělená mezerou. Počet definovaných stanic na dané lince musí odpovídat 2. číslu na 1. řádce popisu linky.

př.:

> Malostranska 6 () Staromestska 6 () Mustek 9 (2) Muzeum 9 () NamestiMiru 7 ()

4.řádek

Na 4. řádku se nachází vzdálenosti mezi jednotlivými stanicemi. Vzdálenost je zde vyjádřena v čase jízdy mezi stanicemi v celých kladných minutách. To znamená že je zde *(n-1)* přirozených čísel, kde *n* je počet stanic dané linky. První číslo odpovídá době jízdy mezi první a druhou zastávkou, tedy obecně i-té číslo odpovídá době jízdy mezi i-tou a (i+1)-ní zastávkou.

př.:

> 2 3 3 3

Těmito 4-mi řádky je definovaná právě jedna linka. Linek může být libovolně mnoho, ale jejich množství musí odpovídat číslu ze začátku souboru.

Zde ještě celistvý vstupní soubor jako příklad:

> 2 3 1 5 50000 4500 21500 Malostranska 6 () Staromestska 6 () Mustek 9 (2) Muzeum 9 () NamestiMiru 7 () 2 3 3 3 2 5 9000 35000 26000 KarlovoNamesti 7 () NarodniTrida 7 () Mustek 8 (1) NamestiRepubliky 6 () Florenc 9 3 2 1 3