TIE-20100 Tietorakenteet ja algoritmit, kevät 2020

Harjoitustyö 2: Nykse Meni?

Viimeksi päivitetty 29.04.2020

Muutoshistoria

Alla ohjeeseen tehdyt merkittävät muutokset julkaisun jälkeen:

- 7.4. Tarkennettu, että add_route palautta false myös silloin, jos reitille annetaan vain yksi pysäkki.
- 29.4. Korjattu typo journey_earliest_arrival-operaation paluuarvossa, jos pysäkkejä ei löydy.

Tärkeää vaiheesta 2

Harjoitustyön vaihe 2 rakentuu vaiheen 1 päälle. Tämä dokumentti kuvaa harjoitustyön vaiheen 2 uudet asiat. Ellei tässä dokumentissa toisin sanota, kaikki asiat, mitä vaiheen 1 kuvauksessa sanottiin, pätevät edelleen.

Vaiheen 2 tekeminen ei vaadi minkään vaiheen 1 ei-pakollisen osan toteuttamista, ja vaiheen 1 ei-pakollisten osien toteuttaminen vaiheessa 2 ei vaikuta arvosteluun. Tarkemmin sanottuna vaihe 2 vaatii lähinnä vaiheen 1 pysäkkien lisäämisen ja hakemisen toimimista, suurinta osaa muusta vaiheen 1 toiminnallisuudesta ei tarvita. Tämän vuoksi vaiheen 2 tekemisen pitäisi onnistua ongelmitta riippumatta siitä, kuinka ”laadukkaasti” vaiheen 1 sai toteutettua.

Harjoitustyön aihe

Harjoitustyönä on tänä vuonna ratikkaprojektin loppusuoran kunniaksi joukkoliikenne. Toisessa vaiheessa bussipysäkkien tietoihin lisätään tiedot niiden läpi kulkevista bussireiteistä (ja ei-pakollisena osana reittejä kulkevien bussien aikatauluista). Näiden tietojen perusteella tehdään matkahakuja. Osa harjoitustyön operaatioista on pakollisia, osa vapaaehtoisia (pakollinen = vaaditaan läpipääsyyn, vapaaehtoinen = ei-pakollinen, mutta silti osa arvostelua arvosanan kannalta).

Terminologiaa

Menossa olevan englanninkielisen sisarkurssin vuoksi ohjelman käyttöliittymä ja rajapinta ovat englanniksi. Tässä selitys tärkeimmistä harjoitustyön termeistä:

- **Reitti (”route”).** Bussireitti on nimetty sarja pysäkkejä, joiden läpi reitillä kulkevat bussit ajavat. Jokaisella reitillä on yksilöivä *ID* (koostuu merkeistä A-Z, a-z ja 0-9) sekä lista pysäkki-ID:itä joiden läpi reitti kulkee. *Huom! Reitit ovat yksisuuntaisia, eli bussi kulkee*

lisättyä reittiä pysäkkien mukaisessa järjestyksessä, mutta ei tule samaa reittiä takaisin (ellei paluumatkaa varten ole lisätty omaa reittiä eri id:llä).

- **Matka (”trip”).** Matka on ei-pakollinen osa harjoitustyötä, joka kuvaa yksittäisen bussin aikataulun jollakin bussireitillä. Matkasta kerrotaan ko. reitin ID (jonka on oltava jokin olemassa oleva reitti-ID) sekä lista kellonaikoja, jotka kertovat milloin bussi lähtee kultakin pysäkiltä (viimeisen pysäkin osalta milloin bussi saapuu pysäkille). Aikoja on luonnollisesti oltava sama määrä kuin reitillä pysäkkejä. Tässä harjoitustyössä oletetaan, että bussi jatkaa matkaa välittömästi, ts. sen lähtöaika pysäkiltä on sama kuin saapumisaika ko. pysäkille.
- **Kulkureitti (”journey”).** Kulkureitti kuvaa matkan lähtöpysäkiltä muiden pysäkkien kautta päämääräpysäkille. Kulkureitti koostuu listasta, joissa on kussakin kohdassa pysäkki-id, reitti-id (mitä reittiä pitkin siirrytään seuraavalle pysäkille, viimeisen pysäkin osalta arvo on NO_ROUTE, koska matka ei jatku) ja operaatiosta riippuen joko etäisyys (kulkureitin kokonaismatka ko. pysäkille saakka) tai aika (lähtöaika ko. pysäkiltä tai viimeisen pysäkin tapauksessa saapumisaika päämäärään).
- (Vaiheen 1 termi ”pysäkki (stop)” löytyy vaiheen 1 dokumentista.)

Harjoitustyössä harjoitellaan valmiiden tietorakenteiden ja algoritmien tehokasta käyttöä (STL), mutta siinä harjoitellaan myös omien algoritmien tehokasta toteuttamista ja niiden tehokkuuden arvioimista (kannattaa tietysti suosia STL:ää omien algoritmien/tietorakenteiden sijaan silloin, kun se on tehokkuuden kannalta järkevää). Toisin sanoen arvostelussa otetaan huomioon valintojen asymptoottinen tehokkuus, mutta sen lisäksi myös ohjelman yleinen tehokkuus (= järkevät ja tehokkaat toteutusratkaisut). ”Mikro-optimoinnista” (tyyliin kirjoitanko ”a = a+b;” vai ”a += b;” tai kääntäjän optimointivipujen säätäminen) ei saa pisteitä.

Tavoitteena on tehdä mahdollisimman tehokas toteutus, kun oletetaan että kaikki ohjelman tuntemat komennot ovat suunnilleen yhtä yleisiä (ellei komentotaulukossa toisin mainita). Monasti tehokkuuden kannalta joutuu tekemään joissain tilanteissa kompromisseja. Tällöin arvostelua helpottaa, jos kyseiset kompromissit on dokumentoitu työn osana palautettuun **dokumenttitiedostoon.** (Muistakaa kirjoittaa ja palauttaa myös dokumentti koodin lisäksi!)

Huomaa erityisesti seuraavat asiat:

- Tämän harjoitustyön uusissa operaatioissa asymptoottiseen tehokkuuteen ei välttämättä pysty hirveästi vaikuttamaan, koska käytetyt algoritmit määräävät sen. Sen vuoksi harjoitustyössä algoritmien toteutukseen ja toiminnallisuuteen kiinnitetään enemmän huomiota kuin vain asymptoottiseen tehokkuuteen.
- **Osana ohjelman palautusta tiedostoon datastructures.hh on jokaisen operaation oheen laitettu kommentti, johon lisätään oma arvio kunkin toteutetun operaation asymptoottisesta tehokkuudesta lyhyiden perusteluiden kera.**
- **Osana ohjelman palautusta palautetaan git:ssä myös dokumentti (samassa hakemistossa/kansiossa kuin lähdekoodi), jossa perustellaan toteutuksessa käytetyt tietorakenteet ja ratkaisut tehokkuuden kannalta. Hyväksyttäviä dokumentin formaatteja ovat puhdas teksti (readme.txt), markdown (readme.md) ja Pdf (readme.pdf).**

- Operaatioiden `journey_least_stops()`, `journey_shortest_distance()`, `journey_with_cycle()`, `add_trip()`, `route_times_from()` ja `journey_earliest_arrival()` toteuttaminen ei ole pakollista läpipääsyn kannalta. Ne ovat kuitenkin osa arvostelua. Osa ei-pakollisista operaatioista saattaa vaatia kurssilla esiteltyjen algoritmien soveltamista.
- Riittävän huonolla toteutuksella työ voidaan hylätä.

Pysäkkien etäisyyksistä

Jotkin operaatiot vaativat tietoa koordinaattien välisistä etäisyyksistä. Tällöin vertaillaan ensisijaisesti koordinaattien ”normaalia” euklidista etäisyyttä $\sqrt{x^2+y^2}$. Reittien tapauksessa kahden pysäkin väliset etäisyydet pyöristetään alaspäin lähimpään kokonaislukuun.

Kulkureittien tulostamisesta

Kun kurssin puolen koodi tulostaa kulkureittejä, se jättää tulostamatta sellaiset arvot, jotka ovat NO_… . Eli jos esim. kulkureitillä olevan pysäkin osalta reitti-id on NO_ROUTE, ko. pysäkin kohdalla ei tulostu seuraavalle pysäkille vievän reitin id:tä (tämä näkyy lähinnä viimeisen pysäkin osalta sekä silloin, jos olet toteuttanut kävely-yhteydet).

Ohjelman toiminta ja rakenne

Osa ohjelmasta tulee valmiina kurssin puolesta, osa toteutetaan itse.

Valmiit osat, jotka tarjotaan kurssin puolesta

Tiedostot *mainprogram.hh, mainprogram.cc, mainwindow.hh, mainwindow.cc, mainwindow.ui* (joihin **EI SAA TEHDÄ *MITÄÄN* MUUTOKSIA)**

Tiedosto *datastructures.hh*

- Kaikki mitä tiedostossa oli vaiheessa 1 (voit kopioida vaiheen 1 toteutuksestasi haluamasi osat vaiheen 2 pohjaksi).
- Tyyppimäärittely `RouteID`, jota käytetään reitin yksilöivänä tunnisteena (jokaisella reitillä on eri id).
- Tyyppimäärittely `Time`, jota käytetään ilmaisemaan kellonaikaa. Tyyppi on kokonaisluku, joka kertoo kellonajan sekunteina keskiyöstä.
- Tyyppimäärittely `Duration`, jota käytetään ilmaisemaan matkan kestoa. Tyyppi on kokonaisluku, joka kertoo keston sekunteina.
- Tyyppimäärittely `Distance`, jota käytetään ilmaisemaan etäisyyttä. Tyyppi on kokonaisluku, joka kertoo etäisyyden metreinä.
- Vakiot `NO_ROUTE`, `NO_TIME`, `NO_DURATION` ja `NO_DISTANCE`, joita käytetään paluuarvoina, jos tietoja kysytään asiasta, jota ei ole olemassa tai jonka arvoa ei tiedetä/tarvita (lisätietoja kunkin operaation kuvauksessa).

Tiedosto *datastructures.cc*

- Tänne luonnollisesti kirjoitetaan luokan operaatioiden toteutukset.

Graafisen käyttöliittymän käytöstä

QtCreatorilla käännettäessä harjoitustyön valmis koodi tarjoaa graafisen käyttöliittymän, jolla ohjelmaa voi testata ja ajaa valmiita testejä sekä visualisoida ohjelman toimintaa. Käyttöliittymään on nyt otettu mukaan reittien piirto ja siihen liittyvät valinnat.

Huom! Käyttöliittymän graafinen esitys kysyy kaikki tiedot opiskelijoiden koodista! **Se ei siis ole "oikea" lopputulos vaan graafinen esitys siitä, mitä tietoja opiskelijoiden koodi antaa.** Jos pysäkkien piirto on päällä, käyttöliittymä hakee kaikki pysäkit operaatiolla `all_stops()` ja kysyy pysäkkien tiedot operaatioilla `get_...()`. Jos alueiden piirtäminen on päällä, ne kysytään operaatiolla `all_regions()`, ja alueen nimi operaatiolla `get_region_name()`. Alueen koko ja sijainti kysytään operaatiolla `region_bounding_box()`. **HUOM! Operaatio `region_bounding_box()` ei ole pakollinen. Jos sitä ei ole toteutettu, alueiden piirto ei tee mitään.** Jos reittien piirto on päällä, käyttöliittymä hakee pysäkiltä lähtevät reitit operaatiolla `routes_from()`. Jos reittejä kahden pysäkin välillä on useita, niiden id:t näytetään pilkulla erotettuna. Kun käyttöliittymä näyttää operaation tuloksena olevan reitin (punaisella), näytetään ensin tuloksessa olevan reitin id, ja sen jälkeen suluissa kaikki pysäkkien väliset mahdolliset reitit, jos niitä on useita. (*Ei-pakollisena osana olevat kävelyreitit näytetään katkoviivoilla, ja niiden id:n tilalla näytetään viiva. Paluuarvoissa kävelyreitti ilmaistaan reitti-id:llä NO_ROUTE).*

Harjoitustyönä toteutettavat osat

Tiedostot *datastructures.hh* ja *datastructures.cc*

- `class Datastructures`: Luokan julkisen rajapinnan jäsenfunktiot tulee toteuttaa. Luokkaan saa lisätä omia määrittelyitä (jäsenmuuttujat, uudet jäsenfunktiot yms.)
- **Tiedostoon *datastructures.hh* kirjoitetaan jokaisen toteutetun operaation yläpuolelle kommentteihin oma arvio ko. operaation toteutuksen asymptoottisesti tehokkuudesta ja lyhyt perustelu arviolle.**

Lisäksi harjoitustyönä toteutetaan alussa mainittu dokumentti readme.pdf.

Huom! Omassa koodissa ei ole tarpeen tehdä ohjelman varsinaiseen toimintaan liittyviä tulostuksia, koska pääohjelma hoitaa ne. Mahdolliset Debug-tulostukset kannattaa tehdä cerr-virtaan (tai `qDebug`:lla, jos käytät Qt:ta), jotta ne eivät sotke testejä.

Ohjelman tuntemat komennot ja luokan julkinen rajapinta

Kun ohjelma käynnistetään, se jää odottamaan komentoja, jotka on selitetty alla. Komennot, joiden yhteydessä mainitaan jäsenfunktio, kutsuvat ko. Datastructure-luokan operaatioita, jotka siis opiskelijat toteuttavat. Osa komennoista on taas toteutettu kokonaan kurssin puolesta pääohjelmassa.

Jos ohjelmalle antaa komentoriviltä tiedoston parametriksi, se lukee komennot ko. tiedostosta ja lopettaa sen jälkeen.

Alla operaatiot on listattu siinä järjestyksessä, kun ne suositellaan toteutettavaksi (tietysti suunnittelu kannattaa tehdä kaikki operaatiot huomioon ottaen jo alun alkaen).

Alla luetellaan vain tämän harjoitustyön uudet tai toiminnaltaan muuttuneet komennot, myös kaikki harjoitustyön 1 komennot ovat käytettävissä.

Komento **`Julkinen jäsenfunktio`**	**Selitys**
`all_routes` `std::vector<RouteID> all_routes()`	Palauttaa kaikki tietorakenteessa olevat reitit mielivaltaisessa järjestyksessä, ts. järjestyksellä ei ole väliä (pääohjelma järjestää ne id:n mukaan). *Tämä operaatio ei ole oletuksena mukana tehokkuustesteissä.*
`clear_all` `void clear_all()`	Tyhjentää tietorakenteet (tämän jälkeen `all_stops`(), `all_regions`() ja `all_routes`() palauttavat tyhjän vektorin). *Tämä operaatio ei ole mukana tehokkuustesteissä.*

`add_route`
`bool add_route(RouteID id,`
`std::vector<StopID> stops)`
	Lisää tietorakenteeseen uuden reitin, joka kulkee annettujen pysäkkien kautta. Jos annetulla id:llä on jo reitti, jos jotain pysäkki-id:tä ei löydy tai jos annettuja pysäkkejä on vain yksi, ei tehdä mitään ja palautetaan `false`, muuten palautetaan `true`.

`routes_from`
`std::vector<std::pair<RouteID,`
`StopID>> routes_from(StopID stopid)`
	Palauttaa listan reiteistä, joita pysäkiltä lähtee, ja mikä pysäkki milläkin reitillä on seuraavana. Jos annetulta pysäkiltä ei lähde reittejä, palautetaan tyhjä lista. Jos annetulla id:llä ei löydy pysäkkiä, palautetaan pari {NO_ROUTE, NO_STOP}.

`route_stops`
`std::vector<StopID>`
`route_stops(RouteID id)`
	Palauttaa listan reitin pysäkeistä ajojärjestyksessä (eli samassa järjestyksessä, jossa annettiin reittiä lisättäessä). Jos id:llä ei löydy reittiä, palautetaan yksi alkio NO_STOP. *Tämä operaatio ei ole mukana tehokkuustesteissä.*

`clear_routes` `void clear_routes()`	Tyhjentää reitit ja matkat (tämän jälkeen `all_routes`() palauttaa tyhjän vektorin), mutta *ei* poista pysäkkejä tai alueita. *Tämä operaatio ei ole mukana tehokkuustesteissä.*
(Alla olevat kannattaa toteuttaa todennäköisesti vasta, kun yllä olevat on toteutettu.)	

Komento `Julkinen jäsenfunktio`	Selitys

`journey_any`
`std::vector<std::tuple<StopID, RouteID, Distance>>`
`journey_any(StopID fromstop, StopID tostop)`
	Palauttaa jonkin (mielivaltaisen) kulkureitin annettujen pysäkkien välillä (ks. "Kulkureittien tulostamisesta"). Palautetussa vektorissa on ensimmäisenä alkupiste kokonaismatkalla 0, sitten kaikki reitin varrella olevat pysäkit ja kokonaismatka ko. pysäkkiin saakka, viimeisenä kohdepysäkki ja kulkureitin kokonaismatka. Jos kulkureittiä ei löydy, palautetaan tyhjä vektori. Jos pysäkki-id:llä ei löydy pysäkkiä, palautetaan yksi alkio {NO_STOP, NO_ROUTE, NO_DISTANCE}. *Huom! Tässä operaatiossa paluuarvon RouteID:llä ei ole väliä, ts. se voi olla mikä vain, esim. NO_ROUTE tai käytetyn reitin id.*

(Seuraavien operaatioiden toteuttaminen ei ole pakollista, mutta ne ovat osa arvostelua. Operaatiot on listattu arvioidun vaikeuden mukaisessa järjestyksessä.)	

`journey_least_stops`
`std::vector<std::tuple<StopID, RouteID, Distance>>`
`journey_least_stops(StopID fromstop, StopID tostop)`
	Palauttaa pysäkkien välisen kulkureitin (ks. "Kulkureittien tulostamisesta"), jossa on mahdollisimman vähän pysäkkejä. Palautetussa vektorissa on ensimmäisenä alkupiste matkalla 0 ja reitti, jolla päästään seuraavalle pysäkille. Sitten kaikki reitin varrella olevat pysäkit ja matka ko. pysäkkiin saakka ja aina seuraavalle pysäkille johtavan reitin id. Viimeisenä on kohdepysäkki ja kokonaismatka, ja viimeisen pysäkin kohdalla reitiksi tulee NO_ROUTE (koska ei enää jatketa). Jos kulkureittiä ei löydy, palautetaan tyhjä vektori. Jos pysäkki-id:llä ei löydy pysäkkiä, palautetaan yksi alkio {NO_STOP, NO_ROUTE, NO_DISTANCE}. Jos samalla pysäkkimäärällä on useita mahdollisia kulkureittejä, palautetaan jokin niistä.

`journey_with_cycle`
`std::vector<std::tuple<StopID, RouteID, Distance>>`
`journey_with_cycle(StopID fromstop)`
	Palauttaa pysäkkien välisen kulkureitin (ks. "Kulkureittien tulostamisesta"), jossa on sykli, ts. kulkureitti palaa takaisin johonkin matkan varrella olevaan pysäkkiin. Palautetussa vektorissa on ensimmäisenä alkupiste kokonaismatkalla 0 ja reitti, jolla päästään seuraavalle pysäkille. Sitten kaikki reitin varrella olevat pysäkit ja kokonaismatka ko. pysäkkiin saakka ja aina seuraavalle pysäkille johtavan reitin id. Viimeisenä on syklin aiheuttava toiseen kertaan tuleva pysäkki ja kulkureitin kokonaismatka, ja viimeisen pysäkin kohdalla reitiksi tulee NO_ROUTE (koska ei enää jatketa). Jos syklistä kulkureittiä ei löydy, palautetaan tyhjä vektori. Jos pysäkki-id:llä ei löydy pysäkkiä, palautetaan yksi alkio {NO_STOP, NO_ROUTE, NO_DISTANCE}. Jos syklisiä kulkureittejä on useita, palautetaan jokin niistä.

Komento `Julkinen jäsenfunktio`	Selitys

`journey_shortest_distance`
`std::vector<std::tuple<StopID, RouteID, Distance>>`
`journey_shortest_distance(StopID fromstop, StopID tostop)`
	Palauttaa pysäkkien välisen kulkureitin (ks. "Kulkureittien tulostamisesta"), jonka kokonaismatka on mahdollisimman lyhyt. Palautetussa vektorissa on ensimmäisenä alkupiste kokonaismatkalla 0 ja reitti, jolla päästään seuraavalle pysäkille. Sitten kaikki reitin varrella olevat pysäkit ja kokonaismatka ko. pysäkkiin saakka ja aina seuraavalle pysäkille johtavan reitin id. Viimeisenä on kohdepysäkki ja kulkureitin kokonaismatka, ja viimeisen pysäkin kohdalla reitiksi tulee NO_ROUTE (koska ei enää jatketa). Jos kulkureittiä ei löydy, palautetaan tyhjä vektori. Jos pysäkki-id:llä ei löydy pysäkkiä, palautetaan yksi alkio {NO_STOP, NO_ROUTE, NO_DISTANCE}. Jos samalla kokonaismatkalla on useita mahdollisia kulkureittejä, palautetaan jokin niistä.

`add_trip` `bool add_trip(RouteID routeid, const std::vector<Time> &stop_times)`	Lisää annetulle reitille matkan (ks. ”matkan” selitys dokumentin alussa), joka lähtee kultakin reitin pysäkiltä annettuna aikana (paitsi päätepysäkiltä, jonne se vain saapuu viimeisenä annettuna aikana). Jos annetulla id:llä ei löydy reittiä, ei tehdä mitään ja palautetaan `false`, muuten palautetaan `true`. Kurssin puolen koodi tarkastaa, että aikoja on sama määrä kuin reitin pysäkkejä (kurssin puolen komennon toteutus tarkastaa tämän kutsumalla route_stops():a).

`route_times_from`
`std::vector<std::pair<Time, Duration> >`
`route_times_from(RouteID routeid, StopID stopid)`
	Palauttaa tiedon tietyn reitin tietyltä pysäkiltä lähtevistä busseista. Kustakin bussista ilmoitetaan sen lähtöaika pysäkiltä ja matkan kesto seuraavalle pysäkille. Ajat voi palauttaa mielivaltaisessa järjestyksessä (pääohjelma järjestää ne lähtöajan mukaan). Jos id:illä ei löydy reittiä tai pysäkkiä tai reitti ei kulje pysäkin kautta, palautettaan yksi alkio {NO_TIME, NO_DURATION}. *Tämä operaatio ei ole mukana tehokkuustesteissä.*

Komento **`Julkinen jäsenfunktio`**	**Selitys**

`journey_earliest_arrival`
`std::vector<std::tuple<StopID, RouteID, Time>>`
`journey_earliest_arrival(StopID fromstop, StopID tostop, Time starttime)`
	Palauttaa pysäkkien välisen kulkureitin (ks. "Kulkureittien tulostamisesta"), jonka saapumisaika kohdepysäkille on mahdollisimman aikainen. Operaatiossa otetaan huomioon ainoastaan ne reitit, joille on lisätty matkoja `add_trip`-komennolla (ja kävelyreitit, jos `add_walking_connections` on toteutettu). Palautetussa vektorissa on ensimmäisenä alkupiste ja reitti, jolla päästään seuraavalle pysäkille sekä lähtöaika lähtöpysäkiltä. Sitten kaikki reitin varrella olevat pysäkit, seuraavalle pysäkille johtavan reitin id ja lähtöaika pysäkiltä. Viimeisenä on kohdepysäkki ja saapumisaika perille, ja viimeisen pysäkin kohdalla reitiksi tulee NO_ROUTE (koska ei enää jatketa). Jos kulkureittiä ei löydy, palautetaan tyhjä vektori. Jos pysäkki-id:llä ei löydy pysäkkiä, palautetaan yksi alkio {NO_STOP, NO_ROUTE, NO_TIME}. Jos samalla saapumisajalla on useita mahdollisia kulkureittejä, palautetaan jokin niistä. *Huom! Operaatiossa saa halutessaan olettaa, että perille päästään saman vuorokauden aikana, ts. vuorokauden vaihtumista kesken matkan ei tarvitse (välttämättä) ottaa huomioon. (Huom2: Jos haluat lisähaasteen, yritä toteuttaa operaatio niin, että useiden mahdollisten kulkureittien tapauksessa valitaan se, jossa bussia täytyy vaihtaa mahdollisimman harvoin).*

`add_walking_connections` `void add_walking_connections()`	Lisää jokaisesta pysäkistä kävelymahdollisuudet 5 lähimpään pysäkkiin (kävely-yhteyksien reitti-id on NO_ROUTE). Jos toteutat operaation journey_earliest_arrival(), kävely-yhteyksillä ei ole lähtöaikaa (ne ovat aina heti mahdollisia) ja niiden keston voit itse valita kävelynopeutesi ja etäisyyden mukaan (esim. 4 km/h). *Huom! Tämä operaatio ei ole mukana julkaistuissa oikeellisuustesteissä, eikä se ole oletuksena mukana tehokkuustesteissä. Se on tarkoitettu lisähaasteeksi erityisesti journey_earliest_arrival()-operaatioon.*
(Seuraavat komennot on toteutettu valmiiksi pääohjelmassa.)	
random_add n (pääohjelman toteuttama)	Lisää tietorakenteeseen (testausta varten) *n* kpl pysäkkejä, joilla on satunnainen id, nimi ja sijainti. 80 % todennäköisyydellä pysäkki lisätään myös arvottuun alueeseen. Huom! Arvot ovat tosiaan satunnaisia, eli saattavat olla kerrasta toiseen eri.
random_route_trips (pääohjelman toteuttama)	Lisää satunnaisen reitin pysäkkien välille. Lisätyllä reitillä on 5 pysäkkiä ja siihen liittyy 2 matkaa (jos et toteuta add_trip()-operaatiota, matkojen lisäämisestä ei tarvitse välittää).

Komento **`Julkinen jäsenfunktio`**	**Selitys**
random_seed n (pääohjelman toteuttama)	Asettaa pääohjelman satunnaislukugeneraattorille uuden siemenarvon. Oletuksena generaattori alustetaan joka kerta eri arvoon, eli satunnainen data on eri ajokerroilla erilaista. Siemenarvon asettamalla arvotun datan saa toistumaan samanlaisena kerrasta toiseen (voi olla hyödyllistä debuggaamisessa).
read "tiedostonimi" (pääohjelman toteuttama)	Lukee lisää komentoja annetusta tiedostosta. (Tällä voi esim. lukea listan tiedostossa olevia pysäkkejä tietorakenteeseen, ajaa valmiita testejä yms.)
stopwatch on / off / next (pääohjelman toteuttama)	Aloittaa tai lopettaa komentojen ajanmittauksen. Ohjelman alussa mittaus on pois päältä ("off"). Kun mittaus on päällä ("on"), tulostetaan jokaisen komennon jälkeen siihen kulunut aika. Vaihtoehto "next" kytkee mittauksen päälle vain seuraavan komennon ajaksi (kätevää read-komennon kanssa, kun halutaan mitata vain komentotiedoston kokonaisaika).
perftest all/compulsory/cmd1;cmd2... **timeout n n1;n2;n3...** (pääohjelman toteuttama)	Ajaa ohjelmalle tehokkuustestit. Tyhjentää tietorakenteen ja lisää sinne *n1* kpl satunnaisia pysäkkejä ja niille alueita (ks. random_add) sekä reittejä ja matkoja. Sen jälkeen arpoo *n* kertaa satunnaisen komennon. Mittaa ja tulostaa sekä lisäämiseen että komentoihin menneen ajan. Sen jälkeen sama toistetaan *n2*:lle jne. Jos jonkin testikierroksen suoritusaika ylittää *timeout* sekuntia, keskeytetään testien ajaminen (tämä ei välttämättä ole mikään ongelma, vaan mielivaltainen aikaraja). Jos ensimmäinen parametri on *all*, arvotaan lisäyksen jälkeen kaikista komennoista, joita on ilmoitettu kutsuttavan usein. Jos se on *compulsory*, testataan vain komentoja, jotka on pakko toteuttaa. Jos parametri on lista komentoja, arvotaan komento näiden joukosta (tällöin kannattaa mukaan ottaa myös random_add, jotta lisäyksiä tulee myös testikierroksen aikana). Jos ohjelmaa ajaa graafisella käyttöliittymällä, "stop test" nappia painamalla testi keskeytetään (nappiin reagointi voi kestää hetken).
testread "in-tiedostonimi" "out-tiedostonimi" (pääohjelman toteuttama)	Ajaa toiminnallisuustestin ja vertailee tulostuksia. Lukee komennot tiedostosta in-tiedostonimi ja näyttää ohjelman tulostuksen rinnakkain tiedoston out-tiedostonimi sisällön kanssa. Jokainen eroava rivi merkitään kysymysmerkillä, ja lopuksi tulostetaan vielä tieto, oliko eroavia rivejä.
help (pääohjelman toteuttama)	Tulostaa listan tunnetuista komennoista.
quit (pääohjelman toteuttama)	Lopettaa ohjelman. (Tiedostosta luettaessa lopettaa vain ko. tiedoston lukemisen.)

"Datatiedostot"

Kätevin tapa testata ohjelmaa on luoda "datatiedostoja", jotka add-komennolla lisäävät joukon pysäkkejä ohjelmaan. Pysäkit voi sitten kätevästi lukea sisään tiedostosta read-komennolla ja sitten kokeilla muita komentoja ilman, että pysäkit täytyisi joka kerta syöttää sisään käsin. Alla on esimerkit datatiedostoista, joista toinen lisää pysäkkejä, toinen alueita:

- *example-stops.txt*

```
# Add stops
add_stop 1 One (1,1)
add_stop 2 Two (6,2)
add_stop 3 Three (0,6)
add_stop 4 Four (7,7)
add_stop 5 Five (4,4)
add_stop 6 Six (2,9)
```

- *example-routes.txt*

```
# Add routes
add_route A 1 3 4
add_route B 4 6 3
add_route C 1 2 5
```

- *example-trips.txt*

```
# Add trips
add_trip A 08:30:00 08:40:00 09:00:00
add_trip B 08:45:00 09:00:00 10:00:00
add_trip C 08:00:00 08:10:00 08:20:00
add_trip C 09:00:00 09:15:00 09:30:00
```

Esimerkki ohjelman toiminnasta

Alla on esimerkki ohjelman toiminnasta. Esimerkin syötteet löytyvät tiedostoista *example-compulsory-in.txt* ja *example-all-in.txt*, tulostukset tiedostoista *example-compulsory-out.txt* ja *example-all-out.txt.* Eli esimerkkiä voi käyttää pienenä testinä pakollisten toimintojen toimimisesta antamalla käyttöliittymästä komennon
`testread "example-all-in.txt" "example-all-out.txt"`

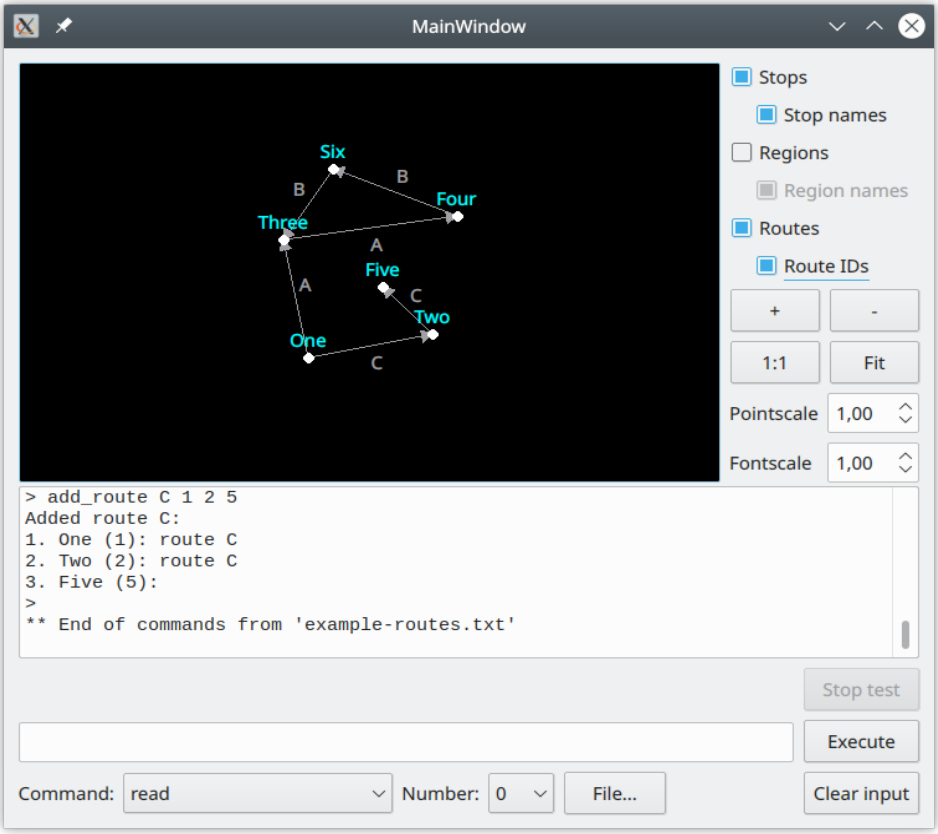
```
> read "example-compulsory-in.txt"
** Commands from 'example-compulsory-in.txt'
> read "example-stops.txt"
** Commands from 'example-stops.txt'
> # Add stops
> add_stop 1 One (1,1)
One: pos=(1,1), id=1
> add_stop 2 Two (6,2)
Two: pos=(6,2), id=2
> add_stop 3 Three (0,6)
Three: pos=(0,6), id=3
> add_stop 4 Four (7,7)
Four: pos=(7,7), id=4
> add_stop 5 Five (4,4)
Five: pos=(4,4), id=5
> add_stop 6 Six (2,9)
Six: pos=(2,9), id=6
>
** End of commands from 'example-stops.txt'
```

```
> read "example-routes.txt"
** Commands from 'example-routes.txt'
> # Add routes
> add_route A 1 3 4
Added route A:
1. One (1): route A
2. Three (3): route A
3. Four (4):
> add_route B 4 6 3
Added route B:
1. Four (4): route B
2. Six (6): route B
3. Three (3):
> add_route C 1 2 5
Added route C:
1. One (1): route C
2. Two (2): route C
3. Five (5):
>
** End of commands from 'example-routes.txt'
> all_routes
1. A
2. B
3. C
> routes_from 1
1. Three (3): route A
2. Two (2): route C
> route_stops A
1. One (1): route A
2. Three (3): route A
3. Four (4):
> journey_any 1 5
1. One (1): distance 0
2. Two (2): distance 5
3. Five (5): distance 7
>
** End of commands from 'example-compulsory-in.txt'
> journey_with_cycle 1
1. One (1): route A distance 0
2. Three (3): route A distance 5
3. Four (4): route B distance 12
4. Six (6): route B distance 17
5. Three (3): distance 20
> add_route D 5 4
Added route D:
1. Five (5): route D
2. Four (4):
> journey_least_stops 1 4
1. One (1): route A distance 0
2. Three (3): route A distance 5
3. Four (4): distance 12
> journey_shortest_distance 1 4
1. One (1): route C distance 0
2. Two (2): route C distance 5
3. Five (5): route D distance 7
4. Four (4): distance 11
> read "example-trips.txt"
** Commands from 'example-trips.txt'
```

```
> # Add trips
> add_trip A 08:30:00 08:40:00 09:00:00
Added trip to route A
1. One (1): route A at 08:30:00
2. Three (3): route A at 08:40:00
3. Four (4): at 09:00:00
> add_trip B 08:45:00 09:00:00 10:00:00
Added trip to route B
1. Four (4): route B at 08:45:00
2. Six (6): route B at 09:00:00
3. Three (3): at 10:00:00
> add_trip C 08:00:00 08:10:00 08:20:00
Added trip to route C
1. One (1): route C at 08:00:00
2. Two (2): route C at 08:10:00
3. Five (5): at 08:20:00
> add_trip C 09:00:00 09:15:00 09:30:00
Added trip to route C
1. One (1): route C at 09:00:00
2. Two (2): route C at 09:15:00
3. Five (5): at 09:30:00
>
** End of commands from 'example-trips.txt'
> route_times_from C 2
Route C leaves from stop Two: pos=(6,2), id=2
 at following times:
08:10:00 ( duration 00:10:00)
09:15:00 ( duration 00:15:00)
> journey_earliest_arrival 1 5 08:45:00
1. One (1): route C at 09:00:00
2. Two (2): route C at 09:15:00
3. Five (5): at 09:30:00
> journey_earliest_arrival 1 6 08:00:00
No journey found!
> add_trip D 08:25:00 08:30:00
Added trip to route D
1. Five (5): route D at 08:25:00
2. Four (4): at 08:30:00
> route_times_from A 1
Route A leaves from stop One: pos=(1,1), id=1
 at following times:
08:30:00 ( duration 00:10:00)
> journey_earliest_arrival 1 6 08:00:00
1. One (1): route C at 08:00:00
2. Two (2): route C at 08:10:00
3. Five (5): route D at 08:25:00
4. Four (4): route B at 08:45:00
5. Six (6): at 09:00:00
>
```

Kuvakaappaus käyttöliittymästä

Alla vielä kuvakaappaus käyttöliittymästä sen jälkeen, kun *example-stops.txt* ja *example-routes.txt* on luettu sisään.


MainWindow
Six
B
B
Four
Three
A
Five
A
C
Two
One
C
Stops
Stop names
Regions
Region names
Routes
Route IDs
+
-
1:1
Fit
Pointscale
1,00
Fontscale
1,00
> add_route C 1 2 5
Added route C:
1. One (1): route C
2. Two (2): route C
3. Five (5):
>
** End of commands from 'example-routes.txt'
Stop test
Execute
Command:
read
Number:
0
File...
Clear input