COMP.CS.300 Tietorakenteet ja algoritmit 1, kevät 2021

Harjoitustyö 2: Retkikartta

Viimeksi päivitetty 05.05.2021

Muutoshistoria

Alla ohjeeseen tehdyt merkittävät muutokset julkaisun jälkeen:

- 6.4. Muutettu, että ways_from() palauttaa tyhjän vektorin, jos koordinaatti ei ole risteys (siitä ei lähde teitä).
- 6.4. Lisätty unohtunut remove_way-operaation kuvaus.
- 6.4. Lisätty maininta, että route_with_cycle:ssä yksikäsitteisyyden varmistamiseksi pääohjelma kääntää tarvittaessa tulostuksessa löytyneen syklin niin, että syklin aloittaa pienempi wayid. Metodi route_with_cycle() ei enää palauta etäisyyksiä.
- 26.4. Lisätty pääohjelman toteuttama komento random_ways.
- 27.4. Esimerkkiajo käyttää nyt clear_ways-komentoa alussa clear_all:n sijaan.
- 3.5. Lisätty huomautus, että route_any-toiminnon ei tarvitse tuottaa tietoa kuljetuista väylä-ID:stä
- 4.5. Päivitetty esimerkkiajo ajan tasalle.

Sisällys

Harjoitustyön aihe

Harjoitustyön toisessa vaiheessa ensimmäisen vaiheen ohjelmaa laajennetaan käsittelemään myös kulkuväyliä (polut, tiet) ja tekemään niihin liittyviä reittihakuja. Osa harjoitustyön operaatioista on pakollisia, osa vapaaehtoisia (pakollinen = vaaditaan läpipääsyyn, vapaaehtoinen = ei-pakollinen, mutta silti osa arvostelua arvosanan kannalta).

Tässä kakkosvaiheen dokumentissa esitellään vain kakkosvaiheen uudet asiat. Tarkoitus on kopioida ykkösvaiheen toteutus kakkosvaiheen pohjaksi ja jatkaa siitä. Kaikki pääohjelman ykkösvaiheen ominaisuudet ja komennot ovat käytössä myös kakkosvaiheessa, vaikka niitä ei toisteta tässä dokumentissa.

Terminologiaa

Menossa olevan englanninkielisen sisarkurssin vuoksi ohjelman käyttöliittymä ja rajapinta ovat englanniksi. Tässä selitys tärkeimmistä 2. vaiheen termeistä:

- **Way = kulkuväylä.** (Kuvaa polkuja, teitä yms, joita pitkin voi kulkea.) Kulkuväylällä on yksilöivä *merkkijono-ID* (koostuu merkeistä A-Z, a-z ja 0-9) ja lista koordinaatteja, jotka kuvaavat väylän maastossa. Väyliä voi kulkea kumpaan suuntaan tahansa. Väylän *pituus* lasketaan (pyöristysvirheiden minimoimiseksi) niin, että etäisyys väylän koordinaatista seuraavaan ($\sqrt{(x_1-x_2)^2+(y_1-y_2)^2}$) **pyöristetään *ensin* alaspäin kokonaisluvuksi**, ja sitten nämä lasketaan yhteen.
- **Crossroad = risteys.** Väylältä voi siirtyä toiselle väylälle vain risteyskohdissa, ja tässä harjoitustyössä risteyskohdiksi lasketaan *vain väylien päätepisteet.* Eli vaikka sattumalta kaksi väylää risteäisikin ”keskellä”, ei tuossa kohtaa voi vaihtaa väylältä toiselle (tämä rajoitus helpottaa harjoitustyön toteuttamista).
- **Route = reitti.** Reitti koostuu jonosta väyliä, jotka ”jatkuvat” niin että edellinen väylä loppuu aina risteykseen, josta seuraava alkaa. Reitin *pituus* on sen sisältämien väylien pituuksien summa.
- **Cycle = silmukka.** Reitissä on silmukka, jos reittiä kulkemalla päädytään jossain vaiheessa takaisin sellaiseen risteykseen, jonka läpi reitti on jo kulkenut.

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). Usein 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 (myös kaikki vaiheen 1 huomioitavat asiat pätevät edelleen):

- *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 `remove_way`, `route_least_crossroads`, `route_shortest_distance`, `route_with_cycle` ja `trim_ways` toteuttaminen ei ole pakollista läpipääsyn kannalta. Ne ovat kuitenkin osa arvostelua. **Vain pakolliset osat toteuttamalla vaiheen maksimiarvosana on 2.**
- Riittävän huonolla toteutuksella työ voidaan hylätä.

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*

- `class Datastructures`: Luokka, johon harjoitustyö kirjoitetaan. Luokasta annetaan valmiina sen julkinen rajapinta (johon **EI SAA TEHDÄ *MITÄÄN* MUUTOKSIA**, luonnollisesti luokkaan saa private-puolelle lisätä omia jäsenmuuttujia ja -funktioita).

- Tyyppimäärittely `Coord`, jota käytetään rajapinnassa (x,y)-koordinaattien esitykseen. Tälle tyypille on esimerkinomaisesti valmiiksi määritelty vertailuoperaatiot ==, != ja < sekä hajautusfunktio.
- Tyyppimäärittely `WayID`, jota käytetään kulkuväylän yksilöivänä tunnisteena.
- Tyyppimäärittely `Distance`, jota käytetään etäisyyksien (= pituuksien) ilmaisemiseen.
- Vakiot `NO_WAY`, `NO_COORD` ja `NO_DISTANCE`, joita käytetään paluuarvoina, jos tietoja kysytään asiasta, jota ei ole olemassa.

Tiedosto *datastructures.cc*

- Tänne luonnollisesti kirjoitetaan luokan operaatioiden toteutukset.
- Funktio `random_in_range`: Arpoo luvun annetulla välillä (alku- ja loppuarvo ovat molemmat välissä mukana). Voit käyttää tätä funktiota, jos tarvitset toteutuksessasi satunnaislukuja.

Graafisen käyttöliittymän käytöstä

QtCreatorilla käännettäessä harjoitustyön valmis koodi tarjoaa graafisen käyttöliittymän, jolla ohjelmaa voi testata ja ajaa valmiita testejä sekä visualisoida ohjelman toimintaa.

Käyttöliittymässä on komentotulkki, jolle voi antaa myöhemmin kuvattuja komentoja, jotka kutsuvat opiskelijan toteuttamia operaatioita. Käyttöliittymä näyttää myös luodut paikat, alueet ja väylät graafisesti (*jos olet toteuttanut tarvittavat operaatiot, ks. alla*). Graafista näkymää voi vierittää ja sen skaalausta voi muuttaa. Paikkojen nimien (ja alueiden ääriviivojen, joihin osuminen voi olla hankalaa) klikkaaminen hiirellä tulostaa sen tiedot ja tuottaa sen ID:n komentoriville (kätevä tapa syöttää komentojen parametreja). Risteyksen klikkaaminen tulostaa sen koordinaatit ja kopioi sen myös komentoriville. Käyttöliittymästä voi myös valita, mitä graafisessa näkymässä näytetään.

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 paikkojen piirto on päällä, käyttöliittymä hakee kaikki paikat operaatiolla `all_places()` ja kysyy paikkojen tiedot operaatioilla `get_...()`. Jos alueiden piirtäminen on päällä, ne kysytään operaatiolla `all_areas()`, ja alueen koordinaatit operaatiolla `get_area_coords()`. Jos väylien piirto on päällä, ne kysytään operaatioilla `all_ways()` ja `get_way_coords()`, ja niiden tuloksia käytetään myös risteyksien piirtämiseen.

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).

Komento **`Julkinen jäsenfunktio`** `(Komennoissa ei-pakolliset parametrit hakasuluissa ja vaihtoehdot erotettu |-merkillä.)`	**Selitys**
(Kaikki vaiheen 1 operaatiot ovat myös saatavilla.)	(Ja tekevät saman asian kuin vaiheessa 1.)
`clear_ways` `void clear_ways()`	Poistaa kaikki väylät (ja niiden mukana risteykset), mutta ei koske paikkoihin eikä alueisiin. *Tämä operaatio ei ole mukana tehokkuustesteissä.*
`all_ways` `std::vector<WayID> all_ways()`	Palauttaa kaikki tietorakenteessa olevat väylät mielivaltaisessa järjestyksessä, ts. järjestyksellä ei ole väliä (pääohjelma järjestää ne id:n mukaan). *Tämä operaatio ei ole oletuksena mukana tehokkuustesteissä.*
`add_way ID Coord1 Coord2...` `bool add_way(WayID id, std::vector<Coord> coords)`	Lisää tietorakenteeseen uuden väylän annetulla uniikilla id:llä koordinaateilla. Jos annetulla id:llä on jo väylä, ei tehdä mitään ja palautetaan `false`, muuten palautetaan `true`.

Komento **`Julkinen jäsenfunktio`** `(Komennoissa ei-pakolliset parametrit hakasuluissa ja vaihtoehdot erotettu |-merkillä.)`	**Selitys**

`way_coords ID`
`std::vector<Coord>`
`get_way_coords(WayID id)`
	Palauttaa annetulla ID:llä olevan väylän koordinaattivektorin, tai vektorin, jonka ainoa alkio on NO_COORD, jos id:llä ei löydy väylää. (Pääohjelma kutsuu tätä eri paikoissa.)

`ways_from Coord` `std::vector<std::pair<WayID, Coord>> ways_from(Coord xy)`	Palauttaa luettelon annetusta koordinaatista lähtevistä väylistä ja risteyksistä, joihin ne vievät. Jos koordinaatissa ei ole risteystä, palautetaan tyhjä vektori.
(Allaolevat kannattaa toteuttaa todennäköisesti vasta, kun ylläolevat on toteutettu.)	
`route_any Coord Coord` `std::vector<std::tuple<Coord, WayID, Distance>> route_any(Coord fromxy, Coord toxy)`	Palauttaa jonkin (mielivaltaisen) reitin annettujen koordinaattien välillä. Palautetussa vektorissa on ensimmäisenä alkupiste ja matka 0 (**WayID:n arvolla ei tässä toiminnossa ole väliä, pääohjelma ei välitä siitä**). Sitten tulevat kaikki reitin varrella olevat risteykset, ja kokonaismatka ko. risteykseen saakka. Viimeisenä alkiona on kohdekoordinaatti, NO_WAY (koska ei jatketa) ja reitin kokonaismatka. Jos reittiä ei löydy, palautetaan tyhjä vektori. Jos jompikumpi koordinaatti ei ole risteys (ts. siitä ei lähde yhtään väylää), palautetaan {NO_COORD, NO_WAY, NO_DISTANCE}.
(Seuraavien operaatioiden toteuttaminen ei ole pakollista, mutta ne parantavat arvosanaa.)	
`remove_way ID` `bool remove_way(WayID id)`	Poistaa annetulla id:llä olevan väylän. Jos id ei vastaa mitään väylää, ei tehdä mitään ja palautetaan `false`, muuten palautetaan `true`. (Jos poistetun väylän päätepisteisiin ei enää poiston jälkeen johda väyliä, päätepisteet eivät luonnollisesti enää ole risteyksiä.)

Komento **Julkinen jäsenfunktio** (Komennoissa ei-pakolliset parametrit hakasuluissa ja vaihtoehdot erotettu |-merkillä.)	**Selitys**

`route_least_crossroads Coord Coord`
`std::vector<std::tuple<Coord, WayID, Distance>>`
`route_least_crossroads(Coord fromxy, Coord toxy)`
	Palauttaa sellaisen reitin annettujen koordinaattien välillä, jossa on mahdollisimman vähän risteyksiä. Jos useilla reiteillä on yhtä monta risteystä, voi niistä palauttaa minkä tahansa. Palautetussa vektorissa on ensimmäisenä alkupiste, mitä väylää siitä jatketaan, ja matka 0. Sitten tulevat kaikki reitin varrella olevat risteykset, mitä väylää niistä jatketaan ja kokonaismatka ko. risteykseen saakka. Viimeisenä alkiona on kohdekoordinaatti, NO_WAY (koska ei jatketa) ja reitin kokonaismatka. Jos reittiä ei löydy, palautetaan tyhjä vektori. Jos jompikumpi koordinaatti ei ole risteys (ts. siitä ei lähde yhtään väylää), palautetaan {NO_COORD, NO_WAY, NO_DISTANCE}.

`route_with_cycle Coord` `std::vector<std::tuple<Coord, WayID>> route_with_cycle(Coord fromxy)`	Palauttaa annetusta risteyksestä lähtevän reitin, jossa on sykli, ts. reitti päätyy uudelleen johonkin reitillä jo olevaan risteykseen. Palautetussa vektorissa on ensimmäisenä alkupiste, mitä väylää siitä jatketaan. Sitten tulevat kaikki reitin varrella olevat risteykset ja mitä väylää niistä jatketaan. Viimeisenä on syklin aiheuttava toiseen kertaan tuleva pysäkki ja NO_WAY (koska ei jatketa). Jos syklistä reittiä ei löydy, palautetaan tyhjä vektori. Jos koordinaatti ei ole risteys (ts. siitä ei lähde yhtään väylää), palautetaan {NO_COORD, NO_WAY}. Jos syklisiä reittejä on useita, palautetaan mikä tahansa niistä. Huom 1! Sykliksi ei lasketa sitä, että palataan päinvastaiseen suuntaan takaisin väylää, jota ollaan juuri kuljettu. Huom 2! Tulostuksen pitämiseksi yksikäsitteisenä *pääohjelma* tarvittaessa kääntää löytyneen syklin niin, että se alkaa pienemmällä way id:llä.

Komento **Julkinen jäsenfunktio** (Komennoissa ei-pakolliset parametrit hakasuluissa ja vaihtoehdot erotettu |-merkillä.)	**Selitys**

`route_shortest_distance Coord Coord`
`std::vector<std::tuple<Coord, WayID, Distance>>`
`route_shortest_distance(Coord fromxy, Coord toxy)`
	Palauttaa annettujen koordinaattien välillä kokonaismatkaltaan mahdollisimman lyhyen reitin. Jos useilla reiteillä on sama kokonaismatka, voi niistä palauttaa minkä tahansa. Palautetussa vektorissa on ensimmäisenä alkupiste, mitä väylää siitä jatketaan, ja matka 0. Sitten tulevat kaikki reitin varrella olevat risteykset, mitä väylää niistä jatketaan ja kokonaismatka ko. risteykseen saakka. Viimeisenä alkiona on kohdekoordinaatti, NO_WAY (koska ei jatketa) ja reitin kokonaismatka. Jos reittiä ei löydy, palautetaan tyhjä vektori. Jos jompikumpi koordinaatti ei ole risteys (ts. siitä ei lähde yhtään väylää), palautetaan {NO_COORD, NO_WAY, NO_DISTANCE}.

`trim_ways` `Distance trim_ways()`	Karsii kulkuväylät (=poistaa niitä) niin, että jäljelle jätetään väylät, joiden yhteenlaskettu pituus on mahdollisimman pieni, mutta joilla edelleen löytyy reitti kaikkien sellaisten risteysten välillä, joiden välillä oli ennenkin reitti. Muut väylät poistetaan. Paluuarvona palautetaan tuloksena olevan väyläverkoston kokonaispituus. Jos olemassa useita kokonaispituudeltaan yhtä lyhyitä väyläverkostoja, mikä tahansa niistä kelpaa.
(Seuraavat komennot on toteutettu valmiiksi pääohjelmassa.)	**(Tässä mainitaan vain muutokset vaiheen 1 toiminnallisuuteen)**
random_ways *n* (pääohjelman toteuttama)	Lisää tietorakenteeseen (testausta varten) noin *n* kpl väyliä satunnaisiin koordinaatteihin. Huom! Arvot ovat tosiaan satunnaisia, eli saattavat olla kerrasta toiseen eri, eivätkä ne graafisesti muodosta järkevää ”karttaa”.

Komento **`Julkinen jäsenfunktio`** `(Komennoissa ei-pakolliset parametrit hakasuluissa ja vaihtoehdot erotettu |-merkillä.)`	**Selitys**
perftest all|compulsory|*cmd1*[;*cmd2*...] ***timeout repeat n1*[;*n2*...]** (pääohjelman toteuttama)	Ajaa ohjelmalle tehokkuustestit. Tyhjentää tietorakenteen ja lisää sinne *n1* kpl satunnaisia paikkoja, alueita ja väyliä. Sen jälkeen arpoo *repeat* 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).

"Datatiedostot"

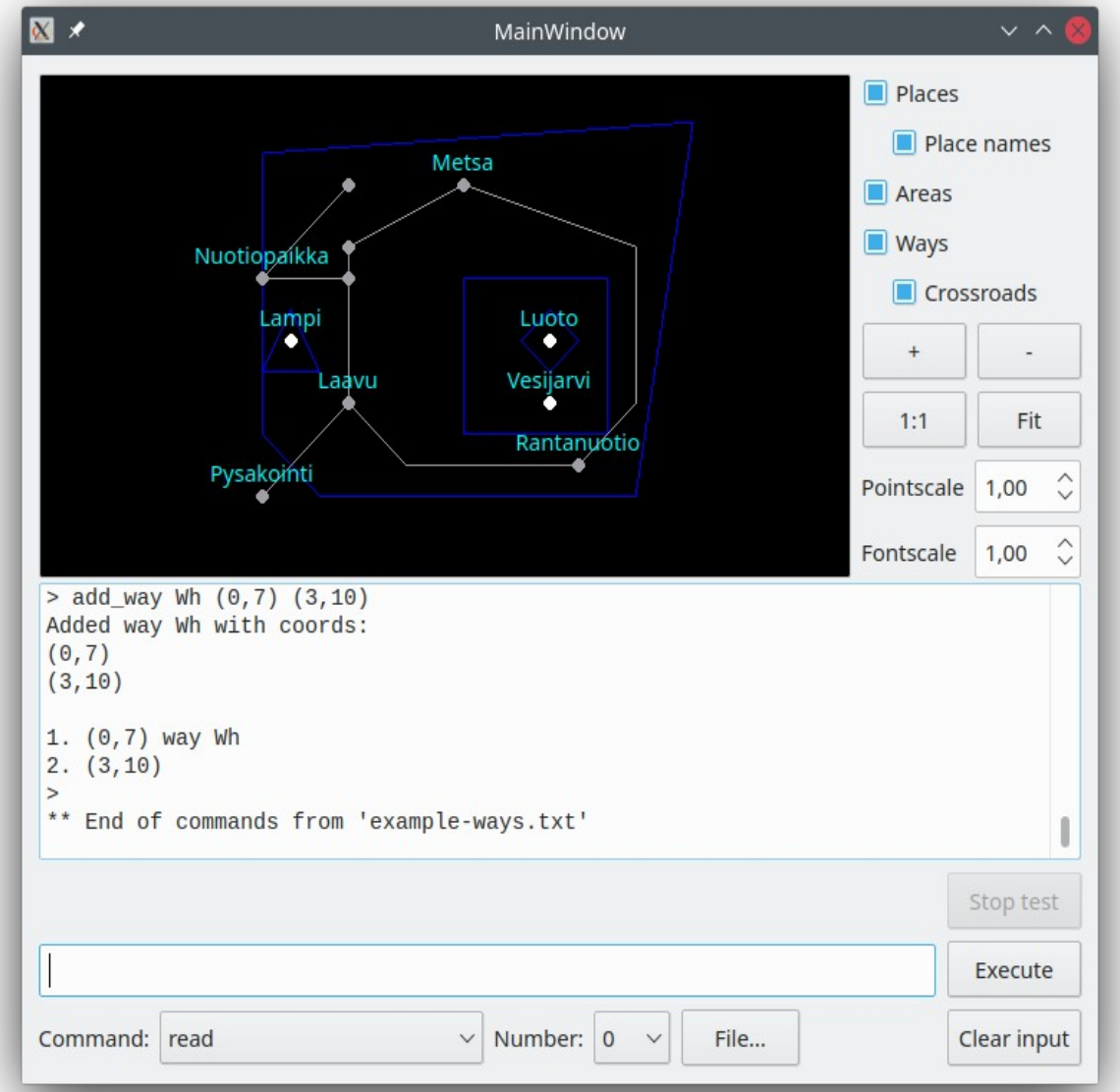
Kätevin tapa testata ohjelmaa on luoda "datatiedostoja", jotka add-komennolla lisäävät joukon paikkoja, alueita ja väyliä ohjelmaan. Tiedot voi sitten kätevästi lukea sisään tiedostosta read-komennolla ja sitten kokeilla muita komentoja ilman, että tiedot täytyisi joka kerta syöttää sisään käsin. Alla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

- *example-way.txt*

```
# Ways
add_way Wa (0,0) (3,3)
add_way Wb (3,3) (5,1) (11,1)
add_way Wc (3,3) (3,7)
add_way Wd (0,7) (3,7)
add_way We (7,10) (3,8)
add_way Wf (3,7) (3,8)
add_way Wg (11,1) (13,3) (13,8) (7,10)
add_way Wh (0,7) (3,10)
```

Kuvakaappaus käyttöliittymästä

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



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-compulsory-in.txt" "example-compulsory-out.txt"
```

```
> clear_ways
All routes removed.
> all_ways
No ways!
> read "example-places.txt" silent
** Commands from 'example-places.txt'
...(output discarded in silent mode)...
** End of commands from 'example-places.txt'
> read "example-areas.txt" silent
** Commands from 'example-areas.txt'
...(output discarded in silent mode)...
** End of commands from 'example-areas.txt'
> read "example-ways.txt"
** Commands from 'example-ways.txt'
> # Ways
> add_way Wa (0,0) (3,3)
Added way Wa with coords: (0,0) (3,3)
1. (0,0) way Wa
2. (3,3)
> add_way Wb (3,3) (5,1) (11,1)
Added way Wb with coords: (3,3) (5,1) (11,1)
1. (3,3) way Wb
2. (11,1)
> add_way Wc (3,3) (3,7)
Added way Wc with coords: (3,3) (3,7)
1. (3,3) way Wc
2. (3,7)
> add_way Wd (0,7) (3,7)
Added way Wd with coords: (0,7) (3,7)
1. (0,7) way Wd
2. (3,7)
> add_way We (7,10) (3,8)
Added way We with coords: (7,10) (3,8)
1. (7,10) way We
2. (3,8)
> add_way Wf (3,7) (3,8)
Added way Wf with coords: (3,7) (3,8)
1. (3,7) way Wf
2. (3,8)
> add_way Wg (11,1) (13,3) (13,8) (7,10)
Added way Wg with coords: (11,1) (13,3) (13,8) (7,10)
1. (11,1) way Wg
2. (7,10)
> add_way Wh (0,7) (3,10)
Added way Wh with coords: (0,7) (3,10)
1. (0,7) way Wh
2. (3,10)
>
** End of commands from 'example-ways.txt'
> all_ways
1. Wa
2. Wb
3. Wc
4. Wd
```

```
5. We
6. Wf
7. Wg
8. Wh
> way_coords Wb
Way Way id Wb has coords:
(3,3)
(5,1)
(11,1)

> ways_from (3,3)
1. (0,0) way Wa
2. (11,1) way Wb
3. (3,7) way Wc
> route_any (3,7) (3,10)
1. (3,7) distance 0
2. (0,7) distance 3
3. (3,10) distance 7>
(… loput tulostukset ovat tiedostosta example-all-out.txt)
> route_least_crossroads (0,0) (7,10)
1. (0,0) way Wa distance 0
2. (3,3) way Wb distance 4
3. (11,1) way Wg distance 12
4. (7,10) distance 25
> route_with_cycle (0,0)
1. (0,0) way Wa
2. (3,3) way Wb
3. (11,1) way Wg
4. (7,10) way We
5. (3,8) way Wf
6. (3,7) way Wc
7. (3,3)
> route_shortest_distance (0,0) (7,10)
1. (0,0) way Wa distance 0
2. (3,3) way Wc distance 4
3. (3,7) way Wf distance 8
4. (3,8) way We distance 9
5. (7,10) distance 13
> trim_ways
The remaining ways have a total length of 28
> all_ways
1. Wa
2. Wb
3. Wc
4. Wd
5. We
6. Wf
7. Wh
>
```