COMP.CS.300 Tietorakenteet ja algoritmit 1, syksy 2022

Harjoitustyö 1: Rautatiet

Viimeksi päivitetty 24.11.2022

Muutoshistoria

Alla ohjeeseen tehdyt merkittävät muutokset julkaisun jälkeen:

- 11.11.: Poistettu vaatimus erillisesti readme-dokumentista.

Sisällys

Harjoitustyön esittely

Tämän syksyn harjoitustyö liittyy rautateihin, eli rautatieasemiin, niiden sijaintiin ja niiden välillä kulkeviin juniin. Ensimmäisessä harjoitustyössä tehdään ohjelma, johon pystyy syöttämään tietoa rautatieasemista, junalähdöistä ja alueista, joihin asemat kuuluvat (kunnat, seutukunnat, maakunnat

jne.). Toisessa harjoitustyössä ohjelmaa laajennetaan käsittelemään asemien välisiä junayhteyksiä ja tekemään niihin liittyviä reittihakuja. Osa harjoitustyön operaatioista on pakollisia, osa vapaaehtoisia (pakollinen = vaaditaan läpipääsyyn, vapaaehtoinen = ei-pakollinen, mutta silti osa arvostelua arvosanan kannalta).

Käytännössä harjoitustyönä koodataan luokka, joka tallettaa tietorakenteisiinsa tarvitut tiedot ja jonka metodit suorittavat tarvitut operaatiot. Kurssin puolesta tulee luokan käyttöön tarvittava pääohjelma ja graafinen Qt-käyttöliittymä (myös pelkkä tekstipohjainen käyttö on mahdollista).

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.

Huomaa erityisesti seuraavat asiat:

- Valmiina annetun pääohjelman voi ajaa joko graafisen käyttöliittymän kanssa QtCreatorilla/qmakella käännettynä, tai tekstipohjaisena pelkällä g++:lla käännettynä. Itse ohjelman toiminnallisuus ja opiskelijan toteuttama osa on täsmälleen sama molemmissa tapauksissa.
- **Vihje** tehokkuudesta: Jos minkään operaation keskimääräinen tehokkuus on huonompi kuin $\Theta(n \log n)$, ratkaisun tehokkuus *ei* ole hyvä. Suurin osa operaatioista on mahdollista toteuttaa paljon nopeamminkin. *Tämä ei tarkoita sitä, että n log n olisi **hyvä** tehokkuus monelle operaatiolle. Erityisesti usein kutsutut operaatiot kannattaa pyrkiä toteuttamaan nopeiksi, niille lineaarinen tehokkuuskin on varsin huono.*
- **Osana ohjelman palautusta tiedostoon datastructures.hh on jokaisen operaation oheen laitettu kommentti, johon lisätään oma arvio kunkin toteutetun operaation asymptoottisesta tehokkuudesta lyhyiden perusteluiden kera.**
- Operaatioiden `all_subregions_in_region`, `stations_closest_to`, `remove_station` ja `common_parent_of_regions` toteuttaminen ei ole pakollista läpipääsyn kannalta. Ne ovat kuitenkin osa arvostelua. **Vain pakolliset osat toteuttamalla harjoitustyön maksimiarvosana on 3.**
- Riittävän huonolla toteutuksella työ voidaan hylätä.

- Tehokkuudessa olennaista on erityisesti, miten ohjelman tehokkuus muuttuu datan kasvaessa, eivät pelkät sekuntimäärät. Plussaa tietysti saa, jos operaatioita saa toteutettua mahdollisimman tehokkaasti.
- Samoin plussaa saa, mitä nopeammaksi operaatiot saa sekunteinakin mitattuna (jos siis kertaluokka on vähintään vaadittu). **Mutta** plussaa saa vain tehokkuudesta, joka syntyy omista algoritmivalinnoista ja suunnittelusta. (Esim. kääntäjän optimointivipujen vääntely, rinnakkaisuuden käyttö, häkkerioptimoinnilla kellojaksojen viilaaminen eivät tuo pisteitä.)
- Operaation tehokkuuteen lasketaan kaikki siihen liittyvä työ, myös mahdollisesti alkioiden lisäyksen yhteydessä tehty operaation hyväksi liittyvä työ.

Terminologiaa

Menossa olevan englanninkielisen sisarkurssin vuoksi ohjelman käyttöliittymä ja rajapinta ovat englanniksi. Tässä selitys tärkeimmistä harjoitustyön termeistä (lista täydentyy harjoitustyössä 2):

- **Station = asema.** Asemalla on yksilöivä *merkkijono-ID, nimi* sekä *sijainti (x,y),* jossa x ja y ovat kokonaislukuja (mittakaava ja koordinaatiston origo on mielivaltainen, x-koordinaatti kasvaa oikealle ja y-koordinaatti ylös). Harjoitustyössä saa olettaa, että samassa koordinaatissa ei voi olla kahta asemaa.
- **Region = (hallinnollinen) alue.** Alueet ovat mielivaltaisia monikulmion rajoittamia alueita kartalla. Jokaisella alueella on yksilöivä *kokonaisluku-ID, nimi* sekä lista koordinaatteja, jotka kuvaavat alueen muodon. Alueeseen voi kuulua asemia ja myös toisia (ali-)alueita, niin että jokainen alue voi kuulua korkeintaan yhteen ”ylempään” alueeseen. Esimerkkinä voisi olla kunta, joka sisältää kunnan asemat ja joka kuuluu seutukuntaan, joka kuuluu johonkin maakuntaan jne. Alueiden ja asemien suhteet annetaan erikseen, ts. niitä ei tarvitse päätellä monikulmioiden sisäkkäisyyksistä. *Aluesuhteet eivät voi muodostaa syklejä, ts. alue 1 ei voi olla alueen 2 alialue, jos alue 2 on jo alueen 1 suora tai epäsuora alialue. Harjoitustyön ei tarvitse huomioida tällaista mahdollisuutta.*

Järjestämisestä

Nimien järjestämisessä voi käyttää suoraan string-luokan ”<”-vertailua, joka toimii koska nimissä ei sallita kuin kirjaimet a-z, A-Z, 0-9, sanaväli ja väliviiva -. Samannimisten nimien keskinäinen järjestys voi olla mikä tahansa.

Operaatio `stations_distance_increasing`() vaatii koordinaattien vertailua. Tällöin vertaillaan ensisijaisesti koordinaatin ”normaalia” euklidista etäisyyttä origosta $\sqrt{x^2+y^2}$ (lähempänä origoa oleva koordinaatti tulee ensin). Jos etäisyys origosta on sama, tulee ensin koordinaatti, jonka y-koordinaatti on pienempi. Koordinaattien, joiden etäisyys ja y-koordinaatti on sama, keskinäinen järjestys voi olla mikä tahansa.

Vastaavasti vapaaehtoisessa operaatiossa `stations_closest_to()` järjestetään asemat niiden annetusta sijainnista etäisyyden perusteella. Silloin etäisyys on ”normaali” kahden pisteen välinen euklidinen etäisyys, ts. $\sqrt{(x_1-x_2)^2+(y_1-y_2)^2}$, ja jälleen jos etäisyys on sama, tulee ensin koordinaatti, jonka y-koordinaatti on pienempi.

Harjoitustyön toteuttamisesta ja C++:n käytöstä

Harjoitustyön kielenä on C++17. Tämän harjoitustyön tarkoituksena on opetella valmiiden tietorakenteiden ja algoritmien käyttöä, joten C++:n STL:n käyttö on erittäin suotavaa ja osa arvostelua. Mitään erityisiä rajoituksia C++:n standardikirjaston käytössä ei ole. Kielen ulkopuolisten kirjastojen käyttö ei ole sallittua (esim. Windowsin omat kirjastot, Qt:n kirjastot, Boost tms.). *Huomaa kuitenkin, että jotkut operaatiot joudut todennäköisesti toteuttamaan myös kokonaan itse.*

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

- Näihin tiedostoihin **EI SAA TEHDÄ *MITÄÄN* MUUTOKSIA!**
- Pääohjelma, joka hoitaa syötteiden lukemisen, komentojen tulkitsemisen ja tulostusten tulostamisen. Pääohjelmassa on myös valmiina komentoja testaamista varten.
- QtCreatorilla tai qmakella käännettäessä graafinen käyttöliittymä, jonka "komentotulkkiin" voi näppäimistön lisäksi hiirellä lisätä komentoja, tiedostoja yms. Graafinen käyttöliittymä näyttää myös luodut asemat ja alueet graafisesti samoin kuin suoritettujen operaatioiden tulokset.

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 `StationID:` Käytetään aseman yksilöivänä tunnisteena (koostuu merkeistä A-Z, a-z, 0-9 ja väliviiva -). Samannimisiä asemia voi olla monta, mutta jokaisella on eri id.

- Tyyppimäärittely `Coord:` Käytetään rajapinnassa (x,y)-koordinaattien esitykseen. Tälle tyypille on esimerkinomaisesti valmiiksi määritelty vertailuoperaatiot ==, != ja < sekä hajautusfunktio.
- Tyyppimäärittely `TrainID:` käytetään junan yksilöivänä tunnisteena (koostuu merkeistä A-Z, a-z, 0-9 ja väliviiva -).
- Tyyppimäärittely `RegionID:` Ei-negatiivinen kokonaisluku, jota käytetään alueen yksilöivänä tunnisteena. Samannimisiä alueita voi olla monta, mutta jokaisella on eri id.
- Tyyppimäärittely `Name:` Käytetään asemien ja alueiden nimenä (koostuu merkeistä A-Z, a-z, 0-9, sanaväli ja väliviiva -).
- Tyyppimäärittely `Time:` Kokonaisluku, joka kuvaa kellonaikaa muodossa HHMM. Pääohjelma hoitaa syötettyjen kellonaikojen oikeellisuuden tarkastelun. Huomaa, että kellonajan esitysmuodosta johtuen kellonaikojen vertailu onnistuu yksinkertaisesti operaattorilla <. Harjoitustyössä ei tarvitse huomioida tilanteita, joissa vuorokausi vaihtuu.
- Vakiot `NO_STATION`, `NO_REGION`, `NO_TRAIN`, `NO_NAME`, `NO_COORD`, `NO_TYPE` ja `NO_DISTANCE:` Käytetään virhekoodeina, jos tietoja kysytään asemasta tai alueesta, 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. Ensimmäisen harjoitustyön käyttöliittymässä on ei-aktiivisena (harmaana) mukana myös muutama toisen harjoitustyön tarvitsema asetus.

Käyttöliittymässä on komentotulkki, jolle voi antaa myöhemmin kuvattuja komentoja, jotka kutsuvat opiskelijan toteuttamia operaatioita. Käyttöliittymä näyttää myös luodut asemat ja alueet graafisesti (*jos olet toteuttanut tarvittavat operaatiot, ks. alla*). Graafista näkymää voi vierittää ja sen skaalausta voi muuttaa. Asemien 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). 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 asemien piirto on päällä, käyttöliittymä hakee kaikki asemat operaatiolla `all_stations()` ja kysyy asemien tiedot operaatioilla `get_...()`. Jos alueiden piirtäminen on päällä, ne kysytään operaatiolla `all_regions()`, ja alueen koordinaatit operaatiolla `get_region_coords()`.

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.**

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. QtCreatorillakin käännetyn ohjelman voi käynnistää komentoriviltä tekstimuotoisena antamalla komentoriviparametrin --console.

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**
`station_count` `int station_count()`	Palauttaa tietorakenteessa olevien asemien lukumäärän.

Komento **`Julkinen jäsenfunktio`** `(Komennoissa ei-pakolliset parametrit hakasuluissa ja vaihtoehdot erotettu |-merkillä.)`	**Selitys**
`clear_all` `void clear_all()`	Tyhjentää tietorakenteet (tämän jälkeen `all_stations`() ja `all_regions`() palauttavat tyhjän vektorin). *Tämä operaatio ei ole mukana tehokkuustesteissä.*

`all_stations`
`std::vector<StationID>`
`all_stations()`
	Palauttaa kaikki tietorakenteessa olevat asemat mielivaltaisessa järjestyksessä, ts. järjestyksellä ei ole väliä (pääohjelma järjestää ne id:n mukaan).

`add_station ID "Name" (x,y)` `bool add_station(StationID id, Name const& name, Coord xy)`	Lisää tietorakenteeseen uuden aseman annetulla uniikilla id:llä, nimellä ja sijainnilla. Jos annetulla id:llä on jo asema, ei tehdä mitään ja palautetaan `false`, muuten palautetaan `true`.
`station_info ID` `Name get_station_name(StationID id)`	Palauttaa annetulla ID:llä olevan aseman nimen tai NO_NAME, jos id:llä ei löydy asemaa. (Pääohjelma kutsuu tätä eri paikoissa.) *Tätä operaatiota kutsutaan useammin kuin muita (myös perftest-komennossa parametrilla ”all” tai ”compulsory”).*
`station_info ID` `Coord get_station_coord(StationID id)`	Palauttaa annetulla ID:llä olevan aseman sijainnin tai arvon NO_COORD, jos id:llä ei löydy asemaa. (Pääohjelma kutsuu tätä eri paikoissa.) *Tätä operaatiota kutsutaan useammin kuin muita (myös perftest-komennossa parametrilla ”all” tai ”compulsory”).*
(Allaolevat kannattaa toteuttaa todennäköisesti vasta, kun ylläolevat on toteutettu.)	

`stations_alphabetically`
`std::vector<StationID>`
`stations_alphabetically()`
	Palauttaa id:t asemien nimen mukaan aakkosjärjestyksessä. Keskenään samannimiset asemat saavat olla missä järjestyksessä tahansa.

`stations_distance_increasing`
`std::vector<StationID>`
`stations_distance_increasing()`
	Palauttaa id:t asemien koordinaattien mukaan järjestettynä (koordinaattien järjestys on määritelty aiemmin tässä dokumentissa).

`find_station_with_coord (x,y)`
`StationID`
`find_station_with_coord(Coord xy)`
	Palauttaa aseman, joka on annetussa koordinaatissa tai NO_STATION, jos sellaista ei ole.

`change_station_coord ID (x,y)`
`bool`
`change_station_coord(StationID id, Coord newcoord)`
	Muuttaa annetulla ID:llä olevan aseman sijainnin. Jos asemaa ei löydy, palauttaa `false`, muuten `true`.

Komento **Julkinen jäsenfunktio** (Komennoissa ei-pakolliset parametrit hakasuluissa ja vaihtoehdot erotettu |-merkillä.)	**Selitys**
`add_departure StationID TrainID Time` `bool add_departure(StationID stationid, TrainID trainid, Time time)`	Lisää tiedon siitä, että annetulta asemalta lähtee annettu juna annetulla kellonajalla. Jos asemaa ei ole, tai jos lähtö on jo lisätty (annettu juna lähtee jo annettuun aikaan annetulta asemalta) palautetaan `false`, muuten `true`.
`remove_departure StationID TrainID Time` `bool remove_departure(StationID stationid, TrainID trainid, Time time)`	Poistaa annetulta asemalta annetun junan lähdön annetulla kellonajalla. Jos asemaa tai junaa annetulla lähtöajalla asemalta ei löydy, palautetaa `false`, muuten `true`.
`station_departures_after StationID Time` `std::vector<std::pair<Time, TrainID>> station_departures_after(StationID stationid, Time time)`	Listaa kaikki lähdöt annetulta asemalta annettuna kellonaikana tai sen jälkeen. Lähdöt listataan kellonaikajärjestyksessä, ja samalla kellonlyömällä lähtevät juna-id:n mukaan. Jos annettua asemaa ei löydy, palautetaan {NO_TIME, NO_TRAIN}.
(Allaolevat kannattaa toteuttaa todennäköisesti vasta, kun ylläolevat on toteutettu.)	
`add_region ID Name Coord1 Coord2...` `bool add_region(RegionID id, Name const& name, std::vector<Coord> coords)`	Lisää tietorakenteeseen uuden alueen annetulla uniikilla id:llä, nimellä ja monikulmiolla (koordinaatit). Aluksi alueeseen ei kuulu toisia alueita tai asemia, eikä alue ole minkään alueen alialue. Jos annetulla id:llä on jo alue, ei tehdä mitään ja palautetaan `false`, muuten palautetaan `true`.
`all_regions` `std::vector<RegionID> all_regions()`	Palauttaa kaikki tietorakenteessa olevat alueet mielivaltaisessa järjestyksessä, ts. järjestyksellä ei ole väliä (pääohjelma järjestää ne id:n mukaan). *Tämä operaatio ei ole oletuksena mukana tehokkuustesteissä.*
`region_info ID` `Name get_region_name(RegionID id)`	Palauttaa annetulla ID:llä olevan alueen nimen, tai NO_NAME, jos id:llä ei löydy aluetta. (Pääohjelma kutsuu tätä eri paikoissa.) *Tätä operaatiota kutsutaan useammin kuin muita (myös perftest-komennossa parametrilla ”all” tai ”compulsory”).*
`region_info ID` `std::vector<Coord> get_region_coords(RegionID id)`	Palauttaa annetulla ID:llä olevan alueen koordinaattivektorin, tai vektorin, jonka ainoa alkio on NO_COORD, jos id:llä ei löydy aluetta. (Pääohjelma kutsuu tätä eri paikoissa.) *Tämä operaatio ei ole mukana tehokkuustesteissä.*

Komento **`Julkinen jäsenfunktio`** `(Komennoissa ei-pakolliset parametrit hakasuluissa ja vaihtoehdot erotettu |-merkillä.)`	**Selitys**
`add_subregion_to_region RegionID RegionID` `bool add_subregion_to_region(RegionID id, RegionID parentid)`	Lisää ensimmäisen alueen alialueeksi toiseen alueeseen. Jos annetuilla id:illä ei löydy alueita, tai jos annettu alialue kuuluu jo johonkin alueeseen, ei tehdä mitään ja palautetaan `false`, muuten palautetaan `true`.
`add_station_to_region StationID RegionID` `bool add_station_to_region(StationID id, RegionID parentid)`	Lisää aseman annettuun alueeseen. Jos annetuilla id:illä ei löydy asemaa tai aluetta, tai jos annettu asema kuuluu jo johonkin alueeseen, ei tehdä mitään ja palautetaan `false`, muuten palautetaan `true`.
`station_in_regions StationID` `std::vector<RegionID> station_in_regions(StationID id)`	Palauttaa kaikki alueet, joihin annettu asema kuuluu suoraan tai epäsuorasti. Paluuarvossa on ensin alue, johon annettu asema kuuluu suoraan, sitten alue, johon tämä alue kuuluu jne. Jos asema ei kuulu mihinkään alueeseen, palautetaan tyhjä vektori. Jos id:llä ei ole asemaa, palautetaan vektori, jonka ainoa alkio on NO_REGION.
(Seuraavien operaatioiden toteuttaminen ei ole pakollista, mutta ne parantavat arvosanaa.)	
`all_subregions_of_region RegionID` `std::vector<RegionID> all_subregions_of_region(RegionID id)`	Palauttaa kaikki alueet, jotka kuuluvat annettuun alueeseen suoraan tai epäsuorasti alialueina. Alueiden järjestys paluuarvossa saa olla mikä tahansa (pääohjelma järjestää ne ID:n mukaan). Jos annettuun alueeseen ei kuulu muita alueita, palautetaan tyhjä vektori. Jos id:llä ei ole aluetta, palautetaan vektori, jonka ainoa alkio on NO_REGION.
`stations_closest_to Coord` `std::vector<StationID> stations_closest_to(Coord xy)`	Palauttaa etäisyysjärjestyksessä kolme annettua koordinaattia lähinnä olevaa asemaa. Jos sopivia asemia ei ole kolmea, palautetaan luonnollisesti vähemmän asemia. *Tämän komennon toteutus ei ole pakollinen (mutta se vaikuttaa arvosteluun).*
`remove_station ID` `bool remove_station(StationID id)`	Poistaa annetulla id:llä olevan aseman. Jos id ei vastaa mitään asemaa, ei tehdä mitään ja palautetaan `false`, muuten palautetaan `true`. *Tämän operaation toteuttaminen ei ole pakollista (mutta otetaan huomioon arvostelussa).*

Komento **`Julkinen jäsenfunktio`** `(Komennoissa ei-pakolliset parametrit hakasuluissa ja vaihtoehdot erotettu |-merkillä.)`	**Selitys**
`common_parent_of_regions RegionID RegionID [RegionID ...]` `common_parent_of_regions(RegionID id1, RegionID id2)`	Palauttaa ”lähimmän” alueen aluehierarkiassa, jonka alialueita molemmat annetut alueet ovat suoraan tai epäsuorasti. "Lähimmän" tarkoittaa, että alueet eivät enää kuulu yhdessä mihinkään palautetun alueen alialueeseen. Jos jompikumpi id ei vastaa mitään aluetta, tai yhteistä ylempää aluetta ei löydy, palautetaan NO_REGION.
(Seuraavat komennot on toteutettu valmiiksi pääohjelmassa.)	
random_add *n* (pääohjelman toteuttama)	Lisää tietorakenteeseen (testausta varten) *n* kpl asemia ja *n/10* kpl alueita, joilla on satunnainen id, nimi ja sijainti. Lisätyt alueet lisätään alialueina satunnaisiin alueisiin ja asemat 50 % todennäköisyydellä johonkin alueeseen. Huom! Arvot ovat tosiaan satunnaisia, eli saattavat olla kerrasta toiseen eri. Myös koordinaatit ovat satunnaisia, eli alialueet ja asemat sijaitsevat todennäköisesti "isäntäalueensa" ulkopuolella yms.
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*" [silent] (pääohjelman toteuttama)	Lukee lisää komentoja annetusta tiedostosta. Jos parametri ’silent’ annetaan, luettujen komentojen tulostusta ei näytetä. (Tällä voi esim. lukea tiedostossa olevilla komennoilla asioita 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).

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 asemia ja alueita (ks. random_add). 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).
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 asemia, alueita ja junia 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 esimerkki datatiedostoista:

example-stations.txt

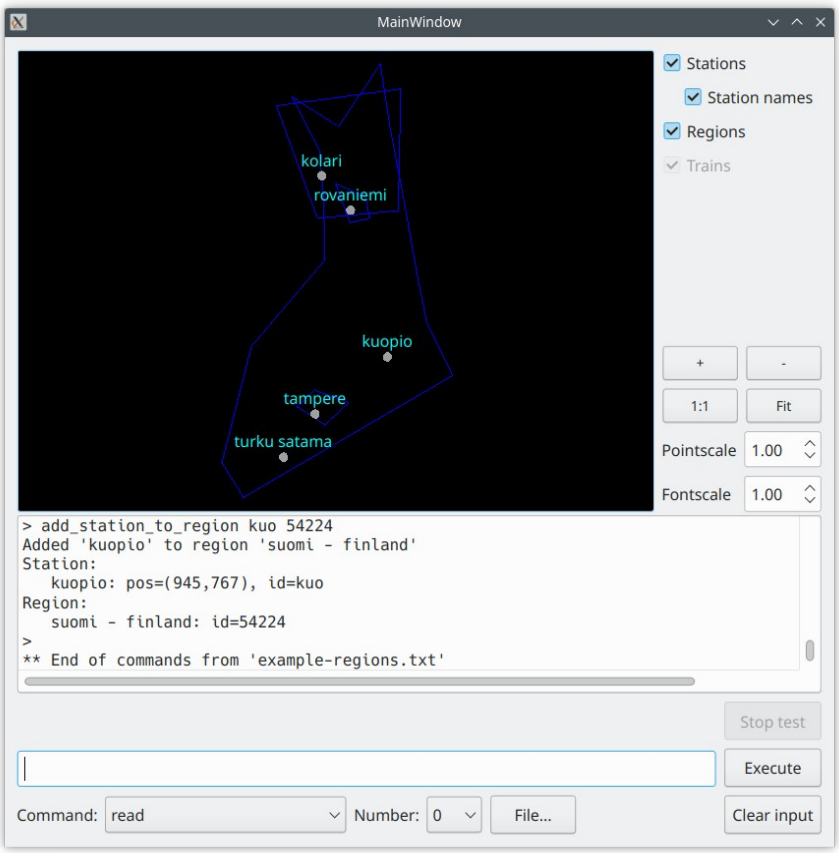
```
add_station kuo "kuopio" (945,767)
add_station tpe "tampere" (542,455)
add_station kli "kolari" (579,1758)
add_station tus "turku satama" (366,219)
add_station roi "rovaniemi" (740,1569)
```

example-regions.txt

```
add_region 6440429 "tampereen seutukunta"  (442,495) (535,586) (729,518)
(597,396) (442,495)
add_station_to_region tpe 6440429
add_region 2528474 "rovaniemi"  (656,1714) (737,1500) (848,1525)
(823,1641) (656,1714)
add_region 1724359 "lappi"  (327,2139) (1020,2232) (1006,1566) (556,1525)
(327,2139)
add_station_to_region roi 2528474
add_subregion_to_region 2528474 1724359
add_station_to_region kli 1724359
add_region 54224 "suomi - finland"  (23,189) (188,827) (598,1300)
(590,1641) (567,1894) (415,2187) (672,2026) (906,2369) (960,2023)
(1030,1664) (1111,1219) (1164,958) (1308,666) (143,0) (23,189)
add_subregion_to_region 6440429 54224
add_subregion_to_region 1724359 54224
add_station_to_region kuo 54224
```

Kuvakaappaus käyttöliittymästä

Alla vielä kuvakaappaus käyttöliittymästä sen jälkeen, kun *example-stations.txt~example-regions.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"`

example-compulsory

```
> clear_all
Cleared all stations
> # Stations
> station_count
Number of stations: 0
> read "example-stations.txt" silent
** Commands from 'example-stations.txt'
...(output discarded in silent mode)...
** End of commands from 'example-stations.txt'
> station_count
Number of stations: 5
> station_info tpe
Station:
   tampere: pos=(542,455), id=tpe
> station_info kli
Station:
   kolari: pos=(579,1758), id=kli
> stations_alphabetically
Stations:
1. kolari: pos=(579,1758), id=kli
2. kuopio: pos=(945,767), id=kuo
3. rovaniemi: pos=(740,1569), id=roi
4. tampere: pos=(542,455), id=tpe
5. turku satama: pos=(366,219), id=tus
> stations_distance_increasing
Stations:
1. turku satama: pos=(366,219), id=tus
2. tampere: pos=(542,455), id=tpe
3. kuopio: pos=(945,767), id=kuo
4. rovaniemi: pos=(740,1569), id=roi
5. kolari: pos=(579,1758), id=kli
> change_station_coord tpe (600,500)
Station:
   tampere: pos=(600,500), id=tpe
> find_station_with_coord (600,500)
Station:
   tampere: pos=(600,500), id=tpe
> add_departure tpe ic20 1000
Train ic20 leaves from station tampere (tpe) at 1000
> add_departure tpe ic22 1200
Train ic22 leaves from station tampere (tpe) at 1200
> add_departure kli pyo276 1942
```

```
Train pyo276 leaves from station kolari (kli) at 1942
> station_departures_after tpe 1100
Departures from station tampere (tpe) after 1100:
 ic22 at 1200
> station_departures_after kli 1900
Departures from station kolari (kli) after 1900:
 pyo276 at 1942
> remove_departure kli pyo276 1942
Removed departure of train pyo276 from station kolari (kli) at 1942
> station_departures_after kli 1900
No departures from station kolari (kli) after 1900
> # Regions
> read "example-regions.txt" silent
** Commands from 'example-regions.txt'
...(output discarded in silent mode)...
** End of commands from 'example-regions.txt'
> all_regions
Regions:
1. suomi - finland: id=54224
2. lappi: id=1724359
3. rovaniemi: id=2528474
4. tampereen seutukunta: id=6440429
> region_info 6440429
Region:
   tampereen seutukunta: id=6440429
> station_in_regions kuo
Station:
   kuopio: pos=(945,767), id=kuo
Region:
   suomi - finland: id=54224
> station_in_regions kli
Station:
   kolari: pos=(579,1758), id=kli
Regions:
1. lappi: id=1724359
2. suomi - finland: id=54224
```

example-all

```
> # First read in compulsory example
> read "example-compulsory-in.txt"
** Commands from 'example-compulsory-in.txt'
...
** End of commands from 'example-compulsory-in.txt'
> all_subregions_of_region 54224
Regions:
1. suomi - finland: id=54224
2. lappi: id=1724359
3. rovaniemi: id=2528474
4. tampereen seutukunta: id=6440429
> stations_closest_to (500,400)
Stations:
1. tampere: pos=(600,500), id=tpe
2. turku satama: pos=(366,219), id=tus
```

```
3. kuopio: pos=(945,767), id=kuo
> common_parent_of_regions 2528474 6440429
Regions:
1. rovaniemi: id=2528474
2. tampereen seutukunta: id=6440429
3. suomi - finland: id=54224
> remove_station tpe
tampere removed.
> stations_alphabetically
Stations:
1. kolari: pos=(579,1758), id=kli
2. kuopio: pos=(945,767), id=kuo
3. rovaniemi: pos=(740,1569), id=roi
4. turku satama: pos=(366,219), id=tus
> stations_distance_increasing
Stations:
1. turku satama: pos=(366,219), id=tus
2. kuopio: pos=(945,767), id=kuo
3. rovaniemi: pos=(740,1569), id=roi
4. kolari: pos=(579,1758), id=kli
> find_station_with_coord (600,600)
Failed (NO_STATION returned)!
> stations_closest_to (500,400)
Stations:
1. turku satama: pos=(366,219), id=tus
2. kuopio: pos=(945,767), id=kuo
3. rovaniemi: pos=(740,1569), id=roi
```