TIE-20100 Tietorakenteet ja algoritmit, kevät 2020

Harjoitustyö 1: Misse Meni?

Viimeksi päivitetty 23.02.2020

Muutoshistoria

Alla ohjeeseen tehdyt merkittävät muutokset julkaisun jälkeen:

- 19.2. Muutettu operaatioiden pakollisuutta niin, että `stop_regions()` on nyt pakollinen ja `region_bounding_box()` ei.

Harjoitustyön aihe

Harjoitustyönä on tänä vuonna ratikkaprojektin loppusuoran kunniaksi joukkoliikenne. Ensimmäisessä vaiheessa tehdään ohjelma, joka osaa kertoa tietoja bussipysäkeistä ja niihin liittyvistä alueista (esim. kaupunginosat, kunnat, maksuvyöhykkeet). Toisessa vaiheessa ohjelmaa laajennetaan käsittelemään myös bussireittejä ja tekemään niihin liittyviä matkahakuja. 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).

Terminologiaa

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

- **Stop = bussipysäkki.** Bussipysäkillä on yksilöivä *kokonaisluku-ID*, *nimi* (koostuu merkeistä A-Z, a-z, 0-9, sanaväli ja väliviiva -) sekä *sijainti (x,y)*, jossa x ja y ovat kokonaislukuja (mittakaava on suunnilleen metreissä, koordinaatiston origo on mielivaltainen).
- **Region = alue.** Jokainen bussipysäkki voi kuulua korkeintaan yhteen alueeseen. Pysäkkien lisäksi alueeseen voi kuulua myös toisia (ali-)alueita, niin että jokainen alue voi kuulua korkeintaan yhteen ”ylempään” alueeseen. Esimerkkinä voisi olla kaupunginosa, joka kuuluu johonkin kuntaan. Jokaisella alueella on yksilöivä *ID* (koostuu merkeistä A-Z, a-z ja 0-9) sekä *nimi* (koostuu merkeistä A-Z, a-z, 0-9, sanaväli ja väliviiva -).

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:

- 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.
- **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 `stops_closest_to()`, `remove_stop()`, `region_bounding_box()` ja `stops_common_region()` toteuttaminen ei ole pakollista läpipääsyn kannalta. Ne ovat kuitenkin osa arvostelua. **Vain pakolliset osat toteuttamalla vaiheen 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.

Järjestämisestä

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

Operaatiot `min_coord`(), `max_coord`() ja `stops_coord_order`() vaativat 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 `stops_closest_to()` pysäkit järjestetään 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. Yhtä kaukana olevien pysäkkien keskinäinen järjestys voi olla mikä tahansa.

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. Luonnollisesti kielen ulkopuolisten kirjastojen käyttö ei ole sallittua (esim. Windowsin omat kirjastot 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* (joihin **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 pysäkit 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 `StopID`, jota käytetään pysäkin yksilöivänä tunnisteena (samannimisiä pysäkkejä voi olla monta ja vaikka samoissa koordinaateissa, mutta jokaisella on eri id).

- Tyyppimäärittely `Coord`, jota käytetään rajapinnassa (x,y)-koordinaattien esitykseen. Tälle tyypille on esimerkinomaisesti valmiiksi määritelty vertailuoperaatiot ==, != ja <.
- Tyyppimäärittely `RegionID`, jota käytetään alueen yksilöivänä tunnisteena (samannimisiä alueita voi olla monta, mutta jokaisella on eri id).
- Tyyppimäärittely `Name`, jota käytetään pysäkkien ja alueiden nimenä. (Tyyppi on merkkijono, johon pääohjelma hyväksyy merkkejä a-z, A-Z, 0-9, sanaväli ja väliviiva -.)
- Vakiot `NO_STOP`, `NO_REGION`, `NO_NAME` ja `NO_COORD`, joita käytetään paluuarvoina, jos tietoja kysytään pysäkistä 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. Käyttöliittymässä on harmaana mukana myös muutama vaiheen 2 tarvitsema asetus.

Käyttöliittymässä on komentotulkki, jolle voi antaa myöhemmin kuvattuja komentoja, jotka kutsuvat opiskelijan toteuttamia operaatioita. Käyttöliittymä näyttää myös luodut pysäkit ja alueet graafisesti (*jos olet toteuttanut tarvittavat operaatiot, ks. alla*). Graafista näkymää voi vierittää ja sen skaalausta voi muuttaa. Pysäkkien ja alueiden nimien 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 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.**

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`**	**Selitys**
`stop_count` `int stop_count()`	Palauttaa tietorakenteessa olevien pysäkkien lukumäärän.
`clear_all` `void clear_all()`	Tyhjentää tietorakenteet (tämän jälkeen `stop_count()` palauttaa 0). *Tämä operaatio ei ole mukana tehokkuustesteissä.*
`all_stops` `std::vector<StopID> all_stops()`	Palauttaa kaikki tietorakenteessa olevat pysäkit mielivaltaisessa järjestyksessä, ts. järjestyksellä ei ole väliä (pääohjelma järjestää ne id:n mukaan). *Tämä operaatio ei ole oletuksena mukana tehokkuustesteissä.*
`add_stop ID Name (x,y)` `bool add_stop(StopID id, Name const& name, Coord xy)`	Lisää tietorakenteeseen uuden pysäkin annetulla uniikilla id:llä, nimellä ja sijainnilla. Aluksi pysäkki ei kuulu mihinkään alueeseen. Jos annetulla id:llä on jo pysäkki, ei tehdä mitään ja palautetaan `false`, muuten palautetaan `true`.
`stop_name ID` `Name get_stop_name(StopID id)`	Palauttaa annetulla ID:llä olevan pysäkin nimen, tai NO_NAME, jos id:llä ei löydy pysäkkiä. (Pääohjelma kutsuu tätä eri paikoissa.) *Tätä operaatiota kutsutaan useammin kuin muita.*
`stop_coord ID` `Coord get_stop_coord(StopID id)`	Palauttaa annetulla ID:llä olevan pysäkin sijainnin, tai arvon NO_COORD, jos id:llä ei löydy pysäkkiä. (Pääohjelma kutsuu tätä eri paikoissa.) *Tätä operaatiota kutsutaan useammin kuin muita.*

Komento `Julkinen jäsenfunktio`	Selitys
(Allaolevat kannattaa toteuttaa todennäköisesti vasta, kun ylläolevat on toteutettu.)	

`stops_alphabetically`
`std::vector<StopID>`
`stops_alphabetically()`
	Palauttaa pysäkkien id:t pysäkkien nimen mukaan aakkosjärjestyksessä. Keskenään samannimiset pysäkit saavat olla missä järjestyksessä tahansa.

`stops_coord_order`
`std::vector<StopID>`
`stops_coord_order()`
	Palauttaa pysäkkien id:t pysäkkien koordinaattien mukaan järjestettynä (koordinaattien järjestys on määritelty aiemmin tässä dokumentissa). Keskenään samassa paikassa olevat pysäkit saavat olla missä järjestyksessä tahansa.

`min_coord` `StopID min_coord()`	Palauttaa pysäkin, jonka koordinaatti on pienin (koordinaattien järjestys on määritelty aiemmin tässä dokumentissa). Jos tällaisia on useita, palauttaa jonkin niistä. Jos pysäkkejä ei ole, palautetaan NO_STOP.
`max_coord` `StopID max_coord()`	Palauttaa pysäkin, jonka koordinaatti on suurin (koordinaattien järjestys on määritelty aiemmin tässä dokumentissa). Jos tällaisia on useita, palauttaa jonkin niistä. Jos pysäkkejä ei ole, palautetaan NO_STOP.

`find_stops name`
`std::vector<StopID>`
`find_stops(Name const& name)`
	Palauttaa pysäkit, joilla on annettu nimi tai tyhjän vektorin, jos sellaisia ei ole. Palautettujen pysäkkien järjestyksellä ei ole väliä (pääohjelma järjestää ne id:n mukaan). *Tätä operaatiota kutsutaan harvoin, eikä se ole oletuksena mukana tehokkuustestissä.*

`change_stop_name ID newname` `bool change_stop_name(StopID id, Name const& newname)`	Muuttaa annetulla ID:llä olevan pysäkin nimen. Jos pysäkkiä ei löydy, palauttaa `false`, muuten `true`.
`change_stop_coord ID (x,y)` `bool change_stop_coord(StopID id, Coord newcoord)`	Muuttaa annetulla ID:llä olevan pysäkin sijainnin. Jos pysäkkiä ei löydy, palauttaa `false`, muuten `true`.
(Allaolevat kannattaa toteuttaa todennäköisesti vasta, kun ylläolevat on toteutettu.)	
`add_region ID Name` `bool add_region(RegionID id, Name const& name)`	Lisää tietorakenteeseen uuden alueen annetulla uniikilla id:llä ja nimellä. Aluksi alueeseen ei kuulu pysäkkejä eikä toisia alueita, eikä alue ole minkään alueen alialue. Jos annetulla id:llä on jo alue, ei tehdä mitään ja palautetaan `false`, muuten palautetaan `true`.
`region_name 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.*

Komento `Julkinen jäsenfunktio`	Selitys

`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.*

`add_stop_to_region StopID RegionID` `bool add_stop_to_region(StopID id, RegionID parentid)`	Lisää annetun pysäkin annettuun alueeseen. Jos annetuilla id:illä ei ole pysäkkiä ja aluetta, tai jos annettu pysäkki kuuluu jo johonkin alueeseen, ei tehdä mitään ja palautetaan `false`, muuten palautetaan `true`.
`add_subregion_to_region RegionID RegionID` `bool add_subregion_to_region(RegionID id, RegionID parentid)`	Lisää annetun alueen alialueeksi annettuun 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`.

`stop_regions ID`
`std::vector<RegionID>`
`stop_regions(StopID id)`
	Palauttaa kaikki alueet, joihin pysäkki kuuluu suoraan tai epäsuorasti. Paluuarvossa on ensin alue, johon pysäkki kuuluu, sitten alue, johon pysäkin alue kuuluu jne. Jos pysäkki ei kuulu mihinkään alueeseen, palautetaan tyhjä vektori. Jos id:llä ei ole pysäkkiä, palautetaan vektori, jonka ainoa alkio on NO_REGION.

(Seuraavien operaatioiden toteuttaminen ei ole pakollista, mutta ne parantavat arvosanaa.)	
`creation_finished` `void creation_finished()`	Tehokkuustestit kutsuvat tätä operaatiota, kun *suurin osa* pysäkeistä ja alueista on luotu (ts. tämän jälkeen lisäyksiä tapahtuu harvemmin). Tämän operaation ei välttämättä tarvitse tehdä mitään, mutta sitä voi halutessaan käyttää algoritmisiin optimointeihin.

`stops_closest_to ID`
`std::vector<StopID>`
`stops_closest_to(StopID id)`
	Palauttaa etäisyysjärjestyksessä viisi pysäkkiä lähinnä olevaa pysäkkiä (koordinaattien järjestys on määritelty aiemmin tässä dokumentissa). Jos pysäkkejä ei ole kuutta, palautetaan luonnollisesti vähemmän pysäkkejä. Jos id ei vastaa mitään pysäkkiä, palautetaan vektori, jonka ainoa alkio on NO_STOP. *Tämän komennon toteutus ei ole pakollinen (mutta se vaikuttaa arvosteluun).*

`remove_stop ID` `bool remove_stop(StopID id)`	Poistaa annetulla id:llä olevan pysäkin. Jos id ei vastaa mitään pysäkkiä, ei tehdä mitään ja palautetaan `false`, muuten palautetaan `true`. Jos poistettava pysäkki kuuluu johonkin alueeseen, se luonnollisesti poistetaan myös sieltä. *Tämän operaation tehokkuus ei ole kriittisen tärkeää (sitä ei oleteta kutsuttavan usein), joten se ei ole oletuksena mukana tehokkuustesteissä. Tämän operaation toteuttaminen ei ole pakollista (mutta otetaan huomioon arvostelussa).*

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

`region_bounding_box ID`
`std::pair<Coord, Coord>`
`region_bounding_box(RegionID id)`
	Palauttaa alueen *bounding boxin*, eli pienimmän suorakulmion, jonka sisään alueen kaikki pysäkit (mukaan lukien alialueiden pysäkit, niiden alialueiden pysäkit jne.) mahtuvat. Toisin sanoen laskee alueen pysäkkien minimi- ja maksimikoordinaatit, paluuarvon ensimmäinen koordinaatti on vasen alakulma, toinen oikea yläkulma. Jos alueella ja sen alialueilla (ja niiden alialueilla jne.) ei ole yhtään pysäkkiä, palautetaan molempina koordinaatteina NO_COORD. *Tämän operaation toteuttaminen ei ole pakollista (mutta otetaan huomioon arvostelussa).*

`stops_common_region ID ID`
`RegionID`
`stops_common_region(StopID id1,`
`StopID id2)`
	Palauttaa ”alimman” alueen aluehierarkiassa, johon molemmat annetut pysäkit suoraan tai epäsuorasti kuuluvat. Toisin sanoen alueen, johon molemmat pysäkit kuuluvat suoraan tai epäsuorasti, mutta molemmat pysäkit eivät kuulu mihinkään tuo alueen alialueista. Jos jompikumpi id ei vastaa mitään pysäkkiä, tai yhteistä aluetta ei löydy, palautetaan NO_REGION.

(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_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 työntekijöitä 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`	Selitys
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). 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-regions.txt*

```
# Add regions and stops to regions
```

```
add_region S Small
add_stop_to_region 1 S
add_stop_to_region 2 S
add_stop_to_region 3 S
add_region L Large
add_subregion_to_region S L
add_stop_to_region 4 L
add_stop_to_region 5 L
add_stop_to_region 6 L
```

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_all
Cleared everything.
> stop_count
Number of stops: 0
> 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'
> stop_count
Number of stops: 6
> stop_name 1
Stop ID 1 has name 'One'
One: pos=(1,1), id=1
> stop_coord 5
Stop ID 5 is in position (4,4)
Five: pos=(4,4), id=5
> stops_alphabetically
1. Five: pos=(4,4), id=5
2. Four: pos=(7,7), id=4
3. One: pos=(1,1), id=1
4. Six: pos=(2,9), id=6
5. Three: pos=(0,6), id=3
6. Two: pos=(6,2), id=2
> min_coord
One: pos=(1,1), id=1
> max_coord
```

```
Four: pos=(7,7), id=4
> stops_coord_order
1. One: pos=(1,1), id=1
2. Five: pos=(4,4), id=5
3. Three: pos=(0,6), id=3
4. Two: pos=(6,2), id=2
5. Six: pos=(2,9), id=6
6. Four: pos=(7,7), id=4
> change_stop_name 5 Two
Two: pos=(4,4), id=5
> find_stops Two
1. Two: pos=(6,2), id=2
2. Two: pos=(4,4), id=5
> read "example-regions.txt"
** Commands from 'example-regions.txt'
> # Add regions and stops to regions
> add_region S Small
Region: Small: id=S
> add_stop_to_region 1 S
Added stop One to region Small
Region: Small: id=S
One: pos=(1,1), id=1
> add_stop_to_region 2 S
Added stop Two to region Small
Region: Small: id=S
Two: pos=(6,2), id=2
> add_stop_to_region 3 S
Added stop Three to region Small
Region: Small: id=S
Three: pos=(0,6), id=3
> add_region L Large
Region: Large: id=L
> add_subregion_to_region S L
Added subregion Small to region Large
> add_stop_to_region 4 L
Added stop Four to region Large
Region: Large: id=L
Four: pos=(7,7), id=4
> add_stop_to_region 5 L
Added stop Two to region Large
Region: Large: id=L
Two: pos=(4,4), id=5
> add_stop_to_region 6 L
Added stop Six to region Large
Region: Large: id=L
Six: pos=(2,9), id=6
>
** End of commands from 'example-regions.txt'
> all_regions
1. Large: id=L
2. Small: id=S
> region_name S
Region ID S has name 'Small'
Small: id=S
> region_bounding_box L
Stops in region Large (and subregions) are within (0,1)-(7,9)
Large: id=L
> quit
```

Kuvakaappaus käyttöliittymästä

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