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

Ohjelmointiprojekti vaihe 1: Game of Taxes

Viimeksi päivitetty 18.11.2021

Sisällys

Muutoshistoria

- 11.10. Muutettu taxer_path():n ja longest_vassal_path():n paluuarvo, jos annettua kaupunkia ei löydy.
- 18.10. Lisätty tyypin TownID yhteyteen selitys sallituista kaupunki-id:istä.
- 18.10. Poistettu sanaväli kaupunkinimen sallituista merkeistä.
- 2.11. Korjattu pari typoa. Lisätty esimerkkiajoon towns_nearest.

Harjoitustyön aihe

Tässä harjoitustyössä harjoitellaan yksinkertaisten algoritmien toteuttamista ja niiden tehokkuuden arvioimista. Harjoitustyössä ohjelmaan syötetään tietoja kaupungeista (nimi ja koordinaatit ja verokertymä), ja ohjelmalta voi kysyä kaupunkeja halutussa järjestyksessä sekä minimi- ja maksimitietoja. Ei-pakollisena osana kaupunkeja voi myös poistaa. Kysyä voi myös kaupunkien verotussuhteita ja verokertymää.

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. Tällöin arvostelua helpottaa, jos kyseiset kompromissit on dokumentoitu työn osana palautettuun **dokumenttitiedostoon.** (Muistakaa kirjoittaa ja palauttaa myös dokumentti koodin lisäksi!)

Huomaa erityisesti seuraavat asiat:

- Tässä harjoitustyössä harjoitellaan valmiiden tietorakenteiden ja algoritmien (STL) tehokasta käyttöä, joten kannattaa suosia STL:ää omien algoritmien/tietorakenteiden sijaan silloin, kun se on tehokkuuden kannalta järkevää.
- 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. *Lisäys: 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.**
- **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_town()`, `towns_nearest()`, `longest_vassal_path()` ja `total_net_tax()` toteuttaminen ei ole pakollista läpipääsyn kannalta. Ne ovat kuitenkin

osa arvostelua, joten toteuttamatta jättäminen vaikuttaa harjoitustyön arvosanaan! **Vain pakolliset osat toteuttamalla harjoitustyön maksimiarvosana on 3.**

- Harjoitustyön kakkosvaiheen tekeminen ei ole pakollista läpipääsyn kannalta, mutta tällöin kurssin maksimiarvosana on 2 (ja kakkosvaiheen osa-arvosanaksi arvostelussa lasketaan 0). Jos kakkosvaihetta ei aio tehdä, riittää 1 vaiheen palautuksena "raakile", jossa on toteutettu vain operaatiot `town_count()`, `clear_all()`, `add_town()`, `get_town_...()`, `all_towns()`, `add_vassalship()` ja `town_vassals()`. Lopullisen vaiheen 1 palautuksen voi tällöin tehdä vaiheen 2 palautuksen määräaikaan mennessä. *Huomaa, että jos palautat vaihessa 1 vain raakileen, et voi enää jatkaa vaiheeseen 2!*
- Riittävän huonolla toteutuksella työ voidaan hylätä.
- Tehokkuudessa olennaisinta on, miten ohjelman tehokkuus muuttuu datan kasvaessa, eivät pelkät sekuntimäärät. Plussaa tietysti 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 väntely, 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.

Kaupunkien etäisyyksistä

Joissain operaatioissa vertaillaan kaupunkien etäisyyksiä (joko origosta (0,0) tai annetusta koordinaatista). Kahden koordinaatin etäisyys lasketaan (pyöristysvirheiden minimoimiseksi) niin, että "normaali" (euklidinen) etäisyys ($\sqrt{(x_1-x_2)^2+(y_1-y_2)^2}$) **pyöristetään ensin alaspäin kokonaisluvuksi**, ja tätä pyöristettyä lukua käytetään etäisyytenä kaikkialla.

Järjestämisestä

Kaupunkeja järjestettäessä on mahdollista, että etäisyyden mukaan järjestettäessä usealla on sama etäisyys (tai nimijärjestyksessä nimi). Tällaisten tapausten keskinäinen järjestys on mielivaltainen.

Ohjelman hyväksymissä nimissä voi olla vain kirjaimia A-Z, a-z, 0-9, sanaväli ja väliviiva -. Järjestämisen voi tehdä joko std::string-luokan vertailuoperaattorin "<" mukaan (jossa isot kirjaimet tulevat ennen pieniä) tai "oikealla tavalla", jossa vastaavat isot ja pienet kirjaimet ovat samanarvoisia.

Kaupunkien ID:iden järjestämisessä tulee käyttää C++:n string-luokan "<"-vertailua.

Verokertymän laskemisesta

Harjoitustyössä kaupungit maksavat toisille kaupungeille veroa. Kaupunkia, joka maksaa veroa toiselle kaupungille, kutsutaan veroja saavan kaupungin *vasallikaupungiksi (vassal town).*Vastaavasti veroja vasallikaupungilta saavaa kaupunkia kutsutaan *isäntäkaupungiksi (master town).* Kaupunki voi olla vain yhden kaupungin vasallikaupunki.

Jokaiselle kaupungille määritellään kaupunkia lisättäessä sen vuotuinen kansalaisilta saatava vero (kokonaislukuna). Tämän lisäksi kaupunki saa 10 % jokaisen vasallikaupunkinsa kokonaisverokertymästä, pyöristettynä alaspäin kokonaisluvuksi (mukaan lukien vasallikaupungin mahdollisesti omilta vasalleiltaan saamat verot). Tästä kertymästä (kansalaisilta saatava määrä + vasalleilta saatava kymmenys) kaupunki maksaa sitten vielä veroa omalle isäntäkaupungilleen (jos sellainen on). Kun saaduista veroista vähennetään eteenpäin maksettu vero, saatavaa lukua kutsutaan tässä *nettoverokertymäksi (total net tax).*

Esimerkki: kaupungilla *x* (vero 20) on kaksi vasallia *v1* (vero 10) ja *v2* (vero 5), joilla ei ole omia vasalleja. Lisäksi kaupungilla *x* on isäntäkaupunki *i*. Tällöin kaupungin *v1* nettoverokertymä on 9 (koska 10 % maksetaan verona *x*:lle), kaupungin *v2* nettoverokertymä on 5 (koska 10 % 5:stä pyöristyy nollaksi). Kaupungin *x* nettoverokertymä on 19 (20 + v*1*:ltä saatava 1 (ja v*2*:lta saatava 0) = 21, josta 10 % eli pyöristettynä 2 maksetaan *i*:lle, tekee yhteensä 19).

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 kaupungit ja niiden vasallisuhteet 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 `TownID`, jota käytetään kaupungit yksilöivänä tunnisteena (samannimisiä kaupunkeja voi olla monta ja vaikka samoissa koordinaateissa, mutta jokaisella on eri id). (Tyyppi on merkkijono, johon pääohjelma hyväksyy merkkejä a-z, A-Z ja 0-9.)
- Tyyppimäärittely `Name`, jota käytetään kaupunkien nimenä. (Tyyppi on merkkijono, johon pääohjelma hyväksyy merkkejä a-z, A-Z, 0-9 ja väliviiva -.)
- Vakiot `NO_TOWNID`, `NO_NAME`, `NO_COORD` ja `NO_VALUE`, joita käytetään paluuarvoina, jos tietoja kysytään kaupungista, jota ei ole olemassa.
- Tyyppimäärittely `Coord`, jota käytetään rajapinnassa (x,y)-koordinaattien esitykseen. (Tälle tyypille on esimerkinomaisesti valmiiksi määritelty joitain operaatioita.)
- Poikkeusluokka `NotImplemented`, jonka jokainen vielä toteuttamaton operaatio aluksi heittää. Näin pääohjelma voi huomauttaa operaatioista, joita ei ole vielä toteutettu (tai jotka on ei-pakollisina jätetty toteuttamatta).

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. Ykkösvaiheessa käyttöliittymässä on ei-aktiivisena (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 kaupungit ja vasallisuhteet graafisesti (*jos olet toteuttanut tarvittavat operaatiot, ks. alla*). Graafista näkymää voi vierittää ja sen skaalausta voi muuttaa. Kaupunkien 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 paikkojen piirto on päällä, käyttöliittymä hakee kaikki kaupungit operaatiolla `all_towns()` ja kysyy kaupunkien tiedot operaatioilla `get_...()`. Jos vasallisuhteiden piirtäminen on päällä, ne kysytään operaatiolla `get_town_vassals()`.

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 asymptoottisesta tehokkuudesta ja lyhyt perustelu arviolle.**

Lisäksi harjoitustyönä toteutetaan alussa mainittu readme-dokumentti.

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. **Muista poistaa debug-tulostukset lopullisesta palautuksestasi!**

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**
town_count `unsigned int town_count();`	Palauttaa tietorakenteessa olevien kaupunkien lukumäärän.
clear_all `void clear_all();`	Tyhjentää tietorakenteet eli poistaa kaikki kaupungit (tämän jälkeen `town_count` palauttaa 0).
add_town id nimi (x,y) vero `bool add_town(TownID id, std::string const& name, Coord coord, int tax);`	Lisää tietorakenteeseen uuden kaupungin annetulla uniikilla id:llä, nimellä, sijainnilla ja verotulolla. Aluksi kaupungit eivät ole minkään toisen kaupungin vasalleja. Jos annetulla id:llä on jo kaupunki, ei tehdä mitään ja palautetaan `false`, muuten palautetaan `true`.
(print_town id) `Name get_town_name(TownID id);`	Palauttaa annetulla ID:llä olevan kaupungin nimen, tai NO_NAME, jos id:llä ei löydy kaupunkia. (Pääohjelma kutsuu tätä eri paikoissa.) *Tätä operaatiota kutsutaan useammin kuin muita.* (Komento print_town kutsuu kaikkia kolmea get_-operaatiota.)

Komento `Julkinen jäsenfunktio`	Selitys
`(print_town id)` `Coord get_town_coordinates(TownID id);`	Palauttaa annetulla ID:llä olevan kaupungin sijainnin, tai NO_COORD, jos id:llä ei löydy kaupunkia. (Pääohjelma kutsuu tätä eri paikoissa.) *Tätä operaatiota kutsutaan useammin kuin muita.* (Komento print_town kutsuu kaikkia kolmea get_-operaatiota.)
`(print_town id)` `int get_town_tax(TownID id);`	Palauttaa annetulla ID:llä olevan kaupungin verotiedon, tai NO_VALUE, jos id:llä ei löydy kaupunkia. (Pääohjelma kutsuu tätä eri paikoissa.) *Tätä operaatiota kutsutaan useammin kuin muita.* (Komento print_town kutsuu kaikkia kolmea get_-operaatiota.)
all_towns `std::vector<TownID> all_towns();`	Palauttaa kaikki tietorakenteessa olevat kaupungit mielivaltaisessa järjestyksessä. *Tämä operaatio ei ole oletuksena mukana tehokkuustesteissä.*
`find_towns nimi` `std::vector<TownID> find_towns(Name const& name);`	Palauttaa kaupungit, joilla on annettu nimi tai tyhjän vektorin, jos sellaisia ei ole. Paluuarvon kaupunkien järjestys voi olla mielivaltainen. *Tätä operaatiota kutsutaan harvoin, eikä se ole oletuksena mukana tehokkuustestissä.*
`change_town_name id newname` `bool change_town_name(TownID id, Name const& newname);`	Muuttaa annetulla ID:llä olevan kaupungin nimen. Jos kaupunkia ei löydy, palauttaa `false`, muuten `true`.
towns_alphabetically `std::vector<TownID> towns_alphabetically();`	Palauttaa kaupungit nimen mukaan aakkosjärjestyksessä.
towns_distance_increasing `std::vector<TownID> towns_distance_increasing();`	Palauttaa kaupungit etäisyysjärjestyksessä origosta (0,0). *Huomaa etäisyyden määritelmä aiempana.*
mindist `TownID min_distance();`	Palauttaa kaupungin, joka on lähinnä origoa (0,0). Jos tällaisia on useita, palauttaa jonkin niistä. Jos kaupunkeja ei ole, palautetaan NO_TOWNID. *Huomaa etäisyyden määritelmä aiempana.*
maxdist `TownID max_distance();`	Palauttaa kaupungin, joka on kauimpana origosta (0,0). Jos tällaisia on useita, palauttaa jonkin niistä. Jos kaupunkeja ei ole, palautetaan NO_TOWNID. *Huomaa etäisyyden määritelmä aiempana.*
add_vassalship vassalid masterid `bool add_vassalship(TownID vassalid, TownID masterid);`	Lisää kaupunkien välille vasallisuhteen. Kaupunki voi olla vain yhden kaupungin vasallikaupunki. Työssä saa olettaa, että vasallisuhteet eivät voi muodostaa silmukoita (ts. kaupunki ei voi olla suoraan tai epäsuorasti itsensä vasalli). Jos jompaakumpaa kaupunkia ei löydy tai vasallilla on jo isäntä, ei tehdä mitään ja palautetaan `false.` Muuten palautetaan `true`.

Komento `Julkinen jäsenfunktio`	Selitys
town_vassals id `std::vector<TownID> get_town_vassals(TownID id);`	Palauttaa annetulla ID:llä olevan kaupungin välittömien vasallien id:t (ts. mukaan *ei lasketa* vasallien vasalleja), tai vektorin jonka *ainoa alkio* on NO_TOWNID, jos id:llä ei löydy kaupunkia. Paluuarvon kaupunkien järjestys on mielivaltainen. (Pääohjelma kutsuu tätä eri paikoissa.)
taxer_path id `std::vector<TownID> taxer_path(TownID id);`	Palauttaa listan kaupungeista, joille kaupunki maksaa veroa suoraan tai epäsuorasti. Paluuvektoriin talletetaan ensin kaupunki itse, sitten sen isäntä, isännän isäntä jne. niin kauan kuin isäntiä riittää. Jos id:llä ei ole kaupunkia, palautetaan vektori, jonka *ainoa alkio* on NO_TOWNID.
remove_town id `bool remove_town(TownID id);`	Poistaa annetulla id:llä olevan kaupungin. Jos id ei vastaa mitään kaupunkia, ei tehdä mitään ja palautetaan `false`, muuten palautetaan `true`. Jos poistettavalla kaupungilla on vasallikaupunkeja ja isäntäkaupunki, poistettavan kaupungin vasallit siirtyvät poistettavan kaupungin isännän vasalleiksi. Jos poistettavalla ei ole isäntää, poistettavan mahdolliset vasallitkin jäävät ilman isäntää poiston jälkeen. *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).*
towns_nearest (x,y) `std::vector<TownID> towns_nearest(Coord coord);`	Palauttaa kaupungit etäisyysjärjestyksessä annetusta koordinaatista (x,y), lähin ensin. *Huomaa etäisyyden määritelmä aiempana. Tämän operaation toteuttaminen ei ole pakollista (mutta otetaan huomioon arvostelussa).*
longest_vassal_path id `std::vector<TownID> longest_vassal_path(TownID id);`	Palauttaa pisimmän mahdollisen ketjun vasalleja kaupungista lähtien. Paluuvektoriin talletetaan ensin kaupunki, sitten kaupungin vasalli, vasallin vasalli jne. niin, että ketjussa on mahdollisimman monta kaupunkia (jos yhtä pitkiä ketjuja on useita, palautetaan jokin niistä). Jos id:llä ei ole kaupunkia, palautetaan vektori, jonka *ainoa alkio* on NO_TOWNID. *Tämän komennon toteutus ei ole pakollinen (mutta se vaikuttaa arvosteluun).*
total_net_tax id `int total_net_tax(TownID id);`	Palauttaa kaupungin nettoverokertymän (joka on määritelty tässä ohjeessa aiemmin). Jos id:llä ei löydy kaupunkia, palautetaan NO_VALUE. *Tämän komennon toteutus ei ole pakollinen (mutta se vaikuttaa arvosteluun).*

Komento **`Julkinen jäsenfunktio`**	**Selitys**
random_add n (pääohjelman toteuttama)	Lisää tietorakenteeseen (testausta varten) *n* kpl kaupunkeja, joilla on satunnainen id, nimi, sijainti ja vero. 80 % todennäköisyydellä kaupunki lisätään myös toisen vasalliksi. 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' [silent] (pääohjelman toteuttama)	Lukee lisää komentoja annetusta tiedostosta. Jos parametri ’silent’ annetaan, luettujen komentojen tulostusta ei näytetä. (Tällä voi esim. lukea listan tiedostossa olevia 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).
perftest all/compulsory/cmd1;cmd2... **timeout n n1;n2;n3...** (pääohjelman toteuttama)	Ajaa ohjelmalle tehokkuustestit. Tyhjentää tietorakenteen ja lisää sinne *n1* kpl satunnaisia kaupunkeja (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).

Komento `Julkinen jäsenfunktio`	Selitys
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 kaupunkeja ohjelmaan. Kaupungit voi sitten kätevästi lukea sisään tiedostosta read-komennolla ja sitten kokeilla muita komentoja ilman, että kaupungit täytyisi joka kerta syöttää sisään käsin. Alla on esimerkki datatiedostosta, joka löytyy nimellä *example-data.txt*:

```
# Adding towns
add_town Hki Helsinki (3,0) 1000
add_town Tpe Tampere (2,2) 500
add_town Ol Oulu (3,5) 400
add_town Kuo Kuopio (6,3) 300
add_town Tku Turku (1,1) 30
# Adding vassalships
add_vassalship Ol Kuo
add_vassalship Kuo Hki
add_vassalship Tpe Hki
```

Esimerkki ohjelman toiminnasta

Alla on esimerkki ohjelman toiminnasta. Esimerkin syöte löytyy tiedostoista *example-...-in.txt* ja tulostukset tiedostoista *example-...-out.txt*.

```
> clear_all
Cleared all towns
> town_count
Number of towns: 0
> read "example-data.txt"
** Commands from 'example-data.txt'
> # Adding towns
> add_town Hki Helsinki (3,0) 1000
Helsinki: tax=1000, pos=(3,0), id=Hki
> add_town Tpe Tampere (2,2) 500
Tampere: tax=500, pos=(2,2), id=Tpe
```

```
> add_town Ol Oulu (3,5) 400
Oulu: tax=400, pos=(3,5), id=Ol
> add_town Kuo Kuopio (6,3) 300
Kuopio: tax=300, pos=(6,3), id=Kuo
> add_town Tku Turku (1,1) 30
Turku: tax=30, pos=(1,1), id=Tku
> # Adding vassalships
> add_vassalship Ol Kuo
Added vassalship: Oulu -> Kuopio
> add_vassalship Kuo Hki
Added vassalship: Kuopio -> Helsinki
> add_vassalship Tpe Hki
Added vassalship: Tampere -> Helsinki
>
** End of commands from 'example-data.txt'
> town_count
Number of towns: 5
> towns_alphabetically
1. Helsinki: tax=1000, pos=(3,0), id=Hki
2. Kuopio: tax=300, pos=(6,3), id=Kuo
3. Oulu: tax=400, pos=(3,5), id=Ol
4. Tampere: tax=500, pos=(2,2), id=Tpe
5. Turku: tax=30, pos=(1,1), id=Tku
> mindist
Turku: tax=30, pos=(1,1), id=Tku
> maxdist
Kuopio: tax=300, pos=(6,3), id=Kuo
> towns_distance_increasing
1. Turku: tax=30, pos=(1,1), id=Tku
2. Tampere: tax=500, pos=(2,2), id=Tpe
3. Helsinki: tax=1000, pos=(3,0), id=Hki
4. Oulu: tax=400, pos=(3,5), id=Ol
5. Kuopio: tax=300, pos=(6,3), id=Kuo
> find_towns Kuopio
Kuopio: tax=300, pos=(6,3), id=Kuo
> change_town_name Tpe Manse
Manse: tax=500, pos=(2,2), id=Tpe
> town_vassals Hki
1. Kuopio: tax=300, pos=(6,3), id=Kuo
2. Manse: tax=500, pos=(2,2), id=Tpe
> taxer_path Tpe
1. Manse
2. Helsinki
> towns_nearest (7,5)
1. Kuopio: tax=300, pos=(6,3), id=Kuo
2. Oulu: tax=400, pos=(3,5), id=Ol
3. Manse: tax=500, pos=(2,2), id=Tpe
4. Helsinki: tax=1000, pos=(3,0), id=Hki
5. Turku: tax=30, pos=(1,1), id=Tku
> longest_vassal_path Hki
1. Helsinki
```

```
2. Kuopio
3. Oulu
> total_net_tax Tku
Total net tax of Turku: 30
> total_net_tax Ol
Total net tax of Oulu: 360
> total_net_tax Kuo
Total net tax of Kuopio: 306
> total_net_tax Hki
Total net tax of Helsinki: 1084
> remove_town Kuo
Kuopio removed.
> town_vassals Hki
1. Oulu: tax=400, pos=(3,5), id=Ol
2. Manse: tax=500, pos=(2,2), id=Tpe
```