Toteutus

Ohjelman rakenne

Projekti on jaettu ylätasolla kolmeen hakemistoon:

- doc - dokumentaatio
- perftest - suorituskykytestit
- routefinder - itse ohjelma ja sen yksikkötestit

Hakemistojen `perftest` ja `routefinder` rakenne on Maven-käytännön mukainen, oleellisimpien alihakemistojen ollessa:

- src/main/java - varsinainen ohjelmakoodi
- src/test/java - testikoodi
- src/main/resources - ajonaikaiset resurssit
- src/test/resources - testiresurssit

Maven-projektirakenne on toteutettu siten, että juurihakemistossa sijaitsee POM-tiedosto, joka määrittelee alihakemistot `perftest`ja `routefinder` moduleikseen. Näin juurihakemistossa ajetut Maven-komennot suoritetaan kummallekin modulille, ja modulien hakemistoissa ajettuina komennot koskevat vain kyseistä modulia.

Pakkaukset ja luokat

Sovellus on jaettu useampaan pakkaukseen:

- tira.collections - tietorakennetoteutukset
- tira.commandline - käynnistyskomennot
- tira.domain - sovelluksen omia tietotyyppejä
- tira.input - JSON-syötteen käsittely
- tira.navigation - reitinhakulogiikka ja sen käyttämiä tietotyyppejä
- tira.visualization.starmap - alkeelliselle asteelle jäänyt visualisointi

Testiluokat ovat testattavia luokkia vastaavissa pakkauksissa. Lisäksi `perftest` -modulin paketissa `tira.perftest` on pari luokkaa suorituskykytestaamista varten, sekä yksikkötesti niille.

O-analyysi

Ohjelman suorituksessa on kolme pääasiallista vaihetta:

1. JSON-tähtilistan lukeminen
2. Tähtikartan rakentaminen (eli yhteyksien luominen tähtien välille)
3. Reittihaku

Periaatteessa 1. ja 2. voidaan mieltää kertaluontoiseksi alustamiseksi, kartan muodostamisen jälkeen voidaan tehdä useampia hakuja ilman uutta alustamista.

Tähtilistan lukeminen on mitä ilmeisimmin aikavaativuudeltaan luokkaa O(n), sillä asiaan ei liity erityisiä iteraatioita, rekursioita tai muuta vastaavaa. Levyltä lukeminen on mahdollisesti hitaahkoa, mutta kasvaa "vain" suorassa suhteessa tähtien määrään.

Tähtikartan rakentaminen on selkeästi vaativampaa, mennen jokseenkin näin:

1. Tähtilista kopioidaan (koska JSON-kirjasto tuottaa `java.util.List` -tyyppisen listan):
    - tähdet käydään yksitellen läpi ja kopioidaan uuteen listaan, O(n)
2. Lista käydään läpi, luoden jokaiselle tähdelle sen yhteydet:
    - jokaista tähteä kohden käydään läpi jäljellä olevat tähdet
    - etäisyyden alittaessa kynnyksen, luodaan tähdelle ja löydetylle "naapurille" kummallekin yhteys
    - näin ollen käydään n * n/2 tähteä läpi, eli karkeasti pyöristäen O(n^2)

BFS-haun aikavaativuus riippuu verkon yhteyksien määrästä - harvoilla yhteyksillä käydään vähemmän solmuja, kuin tiheillä. Pahimmassa tapauksessa lähes kaikki solmut ovat yhteydessä toisiinsa ja määränpää löytyy viimeisimpänä, jolloin solmuja käydään läpi noin n, ja jokaista kohti noin n yhteyttä. Näin ollen aikavaativuus huonoimmillaan luokkaa O(n^2) - tosin huonoin tapaus ei ole kovin tyypillinen.