Luomani tietorakenteet tiedostoon datastructures.hh

```
enum Colour { WHITE, GRAY, BLACK};

struct Town
{
    TownID id;
    Name name;
    Coord coord;
    int tax;
    TownID master;
    std::vector<TownID> vassals;
    Colour colour;
    int d;
};

std::unordered_map<TownID, Town> towns_;
std::vector<TownID> towns_alphabetical_;
std::vector<TownID> towns_distance_;
bool alphabetical_order_;
bool distance_order_;
bool sorting_by_distance;
Coord coord_to_compare_;
```

Kun lähdin suunnittelemaan käytettyjä tietorakenteita, oli heti selvää, että jokainen kaupunki on oma structinsa (struct Town). Oma luokka kaupungeille olisi tietenkin ollut toimiva, paitsi ei tämän harjoitustyön rajoissa. Struct on tietenkin yksinkertaisempi toteuttaa ja muokata kuin kokonainen luokka.

Tutustuin harjoitustyön ohjeeseen ja panin merkille mitkä ovat vaatimukset tietorakenteille ja mitkä tärkeitä ominaisuuksia. Pidin tärkeänä sitä, että kaupungin etsiminen id:llä on mahdollisimman tehokasta, koska sitä tullaan tarvitsemaan paljon. Moni funktio myös vaati sitä, että tarkastetaan onko kyseistä id:tä edes olemassa. Kaupungin nimellä hakeminen oli toissijainen asia. Havaitsin myös, että moni funktio ottaa paluuarvokseen std::vector<TownID>:n.

Tiesin, että std::unsorted_map:in .at() on vakioaikainen ja .find() on keskimäärin vakioaikinen, joten halusin tallettaa structit std::unsorted_map:iin towns_. Päädyin tallettamaan TownID:t myös kahteen std::vector<TownID>:seen, towns_alphabetical_:iin ja towns_distance_:een, joissa säilytän aakkos- ja etäisjärjestystä. Koska niin moni funktio ottaa vektorin paluuarvona, tämä tuntui järkevältä vaihtoehdolta. Nämä vektorit järjestetään, kun on tarve ja muuttuujiin alphabetical_order_ ja distance_order_ on tallennettu tieto siitä, onko vektorit järjestyksessä. mindist ja maxdist hyödyntää myös

etäisyysjärjestystä, koska kun towns_distance_ on järjestyksessä, ensimmäinen alkio on lähin ja viimeisin alkio kauimmaisin.

Viimeiset muuttujat (sorting_by_distance ja coord_to_compare) on sitä varten, että sorttaus voidaan tehdä samalla merge- ja mergesort-funktioilla riippumatta mitä ollaan järjestämässä. Sorting_by_distance on 1, kun järjestetään etäisyyden mukaan ja 0 kun järjestetään nimen mukaan. Coord_to_compare viittaa siihen, mihin koordinaattiin verrataan (pointtina se, että voidaan towns_distance_:ssa verrata origoon ja towns_nearest:ssa annettuun koordinaattiin).

En ole täysin tyytyväinen tähän ratkaisuun, koska tämä vaatii usean eri kopion ylläpitämistä ja näyttää mielestäni vähän kömpelöltä. Olisin ollut tyytyväisempi, jos olisin keksinyt yksinkertaisemman ratkaisun. Toisena vaihtoehtona mietin tietojen tallentamista map:iin, joka ylläpitäisi järjestystä, mutta vakioaikainen etsiminen unsorted_map:sta houkutti liikaa. Toisaalta olen tyytyväinen, että moni funktio voi palauttaa suoraan valmiina olevan vektorin, eikä tarvitse luoda funktion sisällä mitään uutta vektoria. Tämän takia myös Town:in vasallit on eivät ole osoittimia, jotta ne voidaan palauttaa suoraan get_town_vassals:ssa.

Mielestäni kaikki muut funktiot onnistuivat paitsi longest_vassal_path, mutta oli kiva kuitenkin että pääsin soveltamaan BFS:ää siinä ja rekursiosta kertausta total_net_tax:ssa.