1. Harjoitustyö Karttaretki

Työn tarkoituksena oli tehdä omat toteutukset kurssin tarjoamaan datastructures hh:hen/cc:hen projektiin karttaretki. Grafiikat, sekä logiikka oli toteutettu valmiiksi, niin opiskelijan vastuuksi jäi toteuttaa vain tietorakenteet, sekä metodit niiden tietojen käsittelyyn ja järjestelyyn. datastructures pohjaan oli valmiiksi useita metodeja jotka opiskelijan tuli toteuttaa itse. Työssä oli erikseen pakollinen osuus ja vapaaehtoinen osuus. Toteutin molemmat osuudet. Dokumentissa käydään läpi valitut tietorakenteet, sekä hieman avataan syitä siihen mitä on valittu ja miksi.

2. Valitut tietorakenteet:

Valitsin kolme eri rakennetta harjoitustyötä varten, jotta voisin helposti tallentaa relevanttia dataa kuhunkin rakenteeseen. Valitsin kaikkiin rakenteisiin unordered mapin, koska sen datan lisäämisen, poistamisen ja löytämisen tehokkuus oli huonoimmillaan O(n), mutta keskimäärin O(1). Yleiseksi rakenteeksi unordered map osoittautui hyväksi valinnaksi sen tehokkuuden vuoksi.

2.1 Alueiden tallennus

Ensimmäisenä on alueille tarkoitettu rakenne, minne tallennettaan kaikki alueisiin liittyvä data paitsi alialue data. Rakenteen avaimeksi valitsin alue ID:n, koska ID:t ovat uniikkeja arvoja eivätkä esiinny useamman kerran. Mapin toisena arvona on std::pair, mihin tallennetaan alueen nimi ja koordinaatit vectoriin.

```
std::unordered_map<AreaID, std::pair<Name, std::vector<Coord>>> areaMap
```

2.2 Paikkojen tallennus

Toisena on paikoille tarkoitettu rakenne, minne tallennettaan kaikki rakenteeseen. Rakenteen avaimena on jälleen alueen ID, jotta arvojen etsiminen ja käsittely olisi mahdollisimman helppoa ja

järkevää, sekä koska ne ovat uniikkeja arvoja. Toisena arvona rakenteeseen on tallennettu tuple, missä on tietona paikan nimi, koordinaatit ja paikan tyyppi, koska ne olivat paikkaan liittyviä tietoja.

```
std::unordered_map<PlaceID, std::tuple<Name, Coord, PlaceType>> placeMap
```

2.2 Aluehierarkia tallennus

Kolmantena rakenteena toimii alueiden hierarkiaa ylläpitävä rakenne. Rakenteen avaimena toimii Alueen ID, ja sinne sisälle on tallennettu sen alueen suora Ylialua(parent node) ja sen alialueet. Tämän rakenteen oli voinut sisällyttää suoraan ensimmäiseen rakenteeseen, mutta valitsin tehdä kolmannen rakenteen, jotta tietorakenteet olisivat hieman selkeämpiä. Ylialueen tallentaminen oli tietoinen valinta, jotta subarea_in_area:an toteuttaminen olisi mahdollisimman tehokasta.

```
std::unordered_map<AreaID, std::pair<AreaID, std::unordered_set<AreaID>>> subArea
```

3. Harjoitustyön osuudet

Työ oli jaettu pakolliseen osuuteen ja valinnaiseen osuuteen. Suunnittelin rakenteet niin, että pakollisen osuuden metodit olisivat mahdollisimman tehokkaita.

3.1 Pakollinen osuus

Suurin osa pakollisen osuuden metodeista oli yksinkertaista datan etsimistä, lisäämistä tai muokkaamista. Nämä osuudet sain suoritettua tehokkuudella O(n).

Pakollisten joukossa oli myös järjestely metodeja, missä piti syöte järjestää aakkosten tai koordinaattien mukaan, jotka sain suoritettua nopeudella O(nlog(n)). Tämä nopeus tulee siitä, että rakenteessa lisätään for-loop:in sisällä set-rakenteeseen arvoja, jotta set-rakenne tekisi raskaan järjestely työn suoraan. Set rakenne on red-black tree tietorakenne minne arvojen sijoittaminen on tehokkuudeltaan log(n). Valitsin set-rakenteen tähän järjestämiseen, koska se oli valmis rakenne mikä on aina järjestyksessä ja sinne lisääminen on hyvin tehokasta.

Viimeisenä toteutin **subarea_in_areas** metodin**.** Tämä metodi päätyi tehokkuuteen O(n), kun sen toteutti linkittämällä alialueeseen aina suoraan sen ylemmän alueen ja käytiin nämä rekurssiivisesti läpi ja palautettiin vektorissa eteenpäin.

3.2 Valinnainen osuus:

Valinnaiseen osuuteen metodien tehokkuudet jäivät melko heikoiksi. **all_subareas_in_area** ja **common_area_of_subareas** ovat molemmat tehokkuudeltaan O(n^2). **all_subareas_in_area** jäi hitaaksi, koska en keksinyt parempaa tapaa käydä läpi rakennetta kuin käydä läpi subarea rakenteen alialueet läpi ja rekursiivisesti käydä niiden alialueet läpi, mikä osoittautui hitaaksi, mutta sai homman suoritettua.

common_area_of_subareas taas jäi hitaaksi, koska en keksinyt tapaa miten tehdä kahdelle vectorille leikkaus tehokkaasti kun ne eivät ole järjestyksessä. Päädyin ratkaisuun, että tallennan hash map rakenteeseen toisen alueen ylialueet ja etsin sieltä toisen ylialueita eli rakenteessa olisi for loopissa find metodi. Tämä olisi pahimmillaan O(n^2), mutta valitsin tämän, koska keskiarvoltaan se on vakio aikainen. Näinpä keskiarvoltaan metodi olisi lineaarinen.

places_closest_to jäi tehokkuudeltaa nlog(n) tehokkuuteen samasta syystä kuin aiemmatkin järjestely metodit. **remove_place:n** sain lineaariseksi, alkuperäisen rakenteen tehokkuuden vuoksi.

Monia metodeja olisi varmasti saanut tehokkaammaksi mm. kirjoittamalla itse algoritmit ja käyttämällä fiksumpia rakenteita.