```
struct Town
{
    TownID id;
    Name name;
    Coord coord;
    int tax;
    TownID master;
    std::vector<TownID> vassals;

    Colour colour;
    int d;
    Town* pi;
    int de;

    std::unordered_map<std::shared_ptr<Town>, Distance> roads;
};

std::multimap<Distance,std::pair<TownID,TownID>> roads_;

std::unordered_map<TownID, std::shared_ptr<Town>> towns_;
```

Teiden tallentamiseksi lisäsin joka Towniin unordered_mapin, jossa on aina avaimena toinen kaupunki ja arvona tien pituus, koska unordered_mapin muokkaus on mutkatonta. Towniin on myös lisätty muuttuja int de ja osoitin pi, joita käytin graafialgoritmeissa (colour ja d oli jo ennestään, koska käytin niitä 1 vaiheessa). Lisäksi tiet on tallennettu multimappiin, jota piti käyttää trim_road_networkissa, jota en saanut toteutettua. Tässä olisi ollut ideana että multimap järjestää tiet pituuden mukaan ja voisin kruskalilla karsia turhat tiet.

Towns_:ia muutin sen verran, että niissä on osoittimet, koska totesin jo 1 vaiheessa, että osoittimien käyttö on tietyissä tilanteissa helpompaa ja selkeämpää etenkin graafialgoritmeissa. Alunperin en jaksanut käyttää osoittimia koska towns_.at(townid):llä Townin saa kuitenkin aika helposti ja nopeasti.

Least_towns_routessa tiesin joko painottamattomien graafialgoritmien BFS:n tai DFS:n olevan tehokkain toteutustapa ja valitsin BFS:n. Shortest_routessa taas A* oli järkevin, koska silloin ei tarvitse käydä kaikkia nodeja läpi.

Road_cycle_routen ja trim_road_networkin toteutus kaatui silmukoiden etsimiseen.
Trim_road_networkissa olisin käyttänyt Kruskalin algoritmia ja Road_cycle_routessa DFS:ää.