TIE-20100 Harjoitustyö dokumentaatio

Yleiset ratkaisut

Pysäkeille (Stop) on käytetty tietorakenteena structia, johon tallennettu seuraavat tiedot

```
StopID id_
Name name_
Coord coord_
Region* region_
```

Alueille (Region) on myös käytetty structia, jossa seuraavat tiedot

```
RegionID id_
Name name_
std::vector<Stop*> childrenStops_
Region* parentRegion_
std::vector<Region*> childrenRegions_
```

Alun perin vain pysäkkeihin tallennettiin tieto mihin alueeseen ne kuuluvat, mutta alueiden rajoja tutkittaessa valinta aiheutti turhaa työtä ja alueeseen lisättiin myös tieto pysäkeistä. Alueisiin lisättiin myös lista sen alialueista samasta syystä. Alueiden alialueille ja pysäkeille ei käytetty hash-taulukkoja, koska pysäkkien määrä realistisesti ei ole suuri. Vaikka hash-taulukko on vakioaikainen alkion etsimisessä (vektori lineaarinen), tämä vakioaika on suurempi kuin vektorilla pienellä määrällä alkioita. Lisäksi ratkaisu ei olisi kovin muistitehokas.

Kaikki pysäkit ja alueet tallennettiin hash-taulukkoihin std::unordered_map<StopID, Stop> ja std::unordered_map<RegionID, Region>. Ohjelman toiminta perustuu hyvin pitkälti pysäkkien ja alueiden etsimiseen käyttämällä niiden tunnistetta StopID ja RegionID. Ensiarvoisen tärkeää oli siis saada näiden haku mahdollisimman nopeaksi ja hash-taulukko tarjoaa nopean (käytännössä) vakioaikaisen haun riippumatta pysäkkien ja alueiden määrästä.

Funktioiden ratkaisut

Tehokkuuksissa std::unordered_map<> tehokkuudeksi on oletettu hauissa, lisäyksissä ja poistoissa Θ(1) vaikka hash-taulukon rakenteen takia on mahdollista, että tehokkuus on Θ(n) jos kaikki alkiot ovat samassa indeksissä. Funktioissa, joissa palautetaan vektori, on varattu vektorille valmiiksi oikea koko, jos se on ollut tiedossa, esimerkiksi **all_stops**() -funktiossa, jossa se on varmasti kaikkien pysäkkien määrä.

Julkiset jäsenfunktiot	Tehokkuus	Selitys
int **stop_count**()	Θ(1)	std::unordered_map<> tallentaa kokonsa ja haku on tällöin vakioaikainen.
void **clear_all**()	Θ(n)	std::unordered_map<>::**clear**(), poistaa jokaisen alkion erikseen(?)
std::vector<StopID> **all_stops**()	Θ(n)	Jokainen pysäkki iteroidaan ja lisätään vektoriin.
bool **add_stop**(StopID id,Name const& name, Coord xy)	Θ(1)	Tutkitaan onko samalla tunnisteella jo pysäkkiä ja lisätään taulukkoon jos ei ole. Molemmat operaatiot ovat vakioaikaisia

Name **get_stop_name**(StopID id)	Θ(1)	Taulukosta etsiminen vakioaikainen operaatio.
Coord **get_stop_coord**(StopID id)	Θ(1)	Taulukosta etsiminen vakioaikainen operaatio.
void **stops_alphabetically**()	Θ(n log n) / Θ(n k)	Järjestäminen on jaettu kahteen eri tapaan riippuen pysäkkien määrästä. Kun määrä on ”suhteellisen” pieni (<10000) on käytetty standardikirjaston std::**sort**(), jolle on kirjoitettu oma nimien vertailufunktio **sort_names**(). Kun määrä ylittää edellä mainitun, siirrytään käyttämään kantalukulajittelun ja std::**sort** () yhdistelmää. Raja on valittu mielivaltaisesti testauksen perusteella. Tehokkuus käyty läpi **sort_alphabetical**() kohdalla.
void **stops_coord_order**()	Θ(n log n)	Ks. **stops_distance_to_coord_order**()
StopID **min_coord**()	Θ(n)	Jokainen pysäkki käydään läpi ja pidetään jatkuvasti kirjaa pienimmästä etäisyydestä.
StopID **max_coord**()	Θ(n)	Jokainen pysäkki käydään läpi ja pidetään jatkuvasti kirjaa suurimmasta etäisyydestä.
std::vector<StopID> **find_stops**(Name const& name)	Θ(n)	Jokainen pysäkki iteroidaan ja tarkistetaan vastaako nimi pysäkin nimeä. Lisähuomio: ohjelmaa testattaessa funktio ei käyttäytynyt täysin lineaarisesti suurilla pysäkkien määrällä ja kokonaisuutena koko funktio oli kummallisen hidas syystä, joka ei selvinnyt.
bool **change_stop_name**(StopID id, Name const& newname)	Θ(1)	Pysäkin löytäminen ja muutos vakioaikaisia.
bool **change_stop_coord**(StopID id, Coord newcoord)	Θ(1)	Pysäkin löytäminen ja muutos vakioaikaisia.
bool **add_region**(RegionID id, Name const& name)	Θ(1)	Tutkitaan onko alueen tunniste jo olemassa ja lisätään jos ei ole.
Name **get_region_name**(RegionID id)	Θ(1)	Haku taulukosta vakioaikainen.
std::vector<RegionID> **all_regions**()	Θ(n)	Jokainen alue lisätään palautettavaan vektoriin.
bool **add_stop_to_region**(StopID id, RegionID parentid)	Θ(1)	Tutkitaan onko pysäkki ja alue olemassa ja lisätään jos ovat.
bool **add_subregion_to_region**(RegionID id, RegionID parentid)	Θ(1)	Tutkitaan ovatko molemmat alueet olemassa ja lisätään jos ovat.
std::vector<RegionID> **stop_regions**(StopID id)	Θ(n)	Tutkitaan onko pysäkki olemassa. Pysäkin alue lisätään vektoriin. Tämän jälkeen alueen ylemmät alueet käydään läpi ja lisätään myös vektoriin. Tehokkuuden voisi arvioida olevan toisaalta Θ(log n), koska alueiden määrän kasvaessa ”rinnakkaisten” alueiden määrä kasvaa myös eikä alueilla ole lineaarisesti enemmän alialueita.
void **creation_finished**()	Θ(1)	

std::vector<StopID> **stops_closest_to**(StopID id)	Θ(1)	Ks. **stops_distance_to_coord_order**(). Tämän jälkeen vektoria lyhennetään sisältämään vain 5 lähintä pysäkkiä.
bool **remove_stop**(StopID id)	Θ(n)	Tutkitaan onko pysäkki olemassa. Sen jälkeen se poistetaan alueen vektorista. Pysäkin löytäminen alueesta vaatii lineaarisen haun. Operaatio on todennäköisesti kuitenkin nopeampi vektorilla kuin käyttämällä hash-taulukkoa myös alueiden pysäkkien tallentamiseen.
std::pair<Coord,Coord> **region_bounding_box**(RegionID id)	Θ(n)	Tutkitaan alueen olemassaolo. Käyttämällä funktiota **find_stops_in_region**() löydetään alueen kaikki pysäkit (myös alialueilla olevat). Kaikki pysäkit käydään läpi ja tallennetaan niistä pienimmät ja suurimmat koordinaatit, jotka vastaavat vasenta yläkulmaa ja oikeata alakulmaa.
RegionID **stops_common_region**(StopID id1, StopID id2)	Θ(n^2)	Tutkitaan ovatko molemmat pysäkit olemassa. Jokaiselle pysäkin id1 aluetta kohti yritetään löytää sama alue pysäkin id2 alueista. Funktiota pystyisi nopeuttamaan järjestämällä ensin alialueet jotenkin, jolloin jokaista aluetta ei tarvitsisi vertailla jokaiseen toiseen alueeseen. Käytännössä järjestäminen olisi hankalaa toteuttaa eikä sellaista välttämättä edes olisi.
Yksityiset jäsenfunktiot		
float **distance_between_coords**(Coord crd1, Coord crd2)	Θ(1)	Kahden pysäkin euklidinen etäisyys.
static bool **sort_distance**(std::pair<float, Stop*> s1, std::pair<float, Stop*> s2)	Θ(1)	Vertailufunktio pysäkkien järjestämiseen etäisyyden mukaan.
static bool **sort_names**(Stop* s1, Stop* s2)	Θ(1)	Vertailufunktio pysäkkien järjestämiseen nimen mukaan.
std::vector<Stop*> **sort_alphabetical**(std::vector<Stop*>& stops, int index=0)	Θ(n log n) / Θ(n k) k merkitsee järjestämisen ämpärien määrää (27)	Funktio järjestää pysäkit nimen mukaan käyttämällä kantalukulajittelua sekä standardikirjaston std::**sort**(). Käytetyt rajat ovat testauksen perusteella valittuja. Funktio luo 27 ämpäriä alkioille, joista ensimmäinen sisältää numerot ja erikoismerkit (0-9, ” ”, ”-”) ja loput 26 aakkoset A-Z ja a-z kirjainkoosta välittämättä (tarkalleen ottaen kaikki isot kirjaimet tulevat järjestyksessä ennen kaikkia pieniä kirjaimia, mutta ohjelman luonteen vuoksi sillä ei ole merkitystä). Ämpäreihin jako tehdään kolmelle ensimmäiselle kirjaimelle rekursiivisesti, jonka jälkeen ämpärit lajitellaan käyttämällä

		std::**sort**(). Tehokkuus on lineaarinen logaritminen kuten vain std::**sort**(), mutta todellinen suoritusaika on nopeampi.
std::vector<StopID> **stops_distance_to_coord_order**(Coord crd)	Θ(n log n)	Funktio laskee jokaiselle pysäkille ensin sen etäisyyden annetusta koordinaatista ja järjestää pysäkit etäisyyksien mukaan. Pysäkki ja etäisyys tallennetaan parina, jolla vältetään etäisyyden laskeminen useammin kuin kertaalleen. Pareista pysäkit luetaan palautettavaan vektoriin.
void **find_subregions**(Region* region, std::unordered_set<Region*>& regions)	Θ(n)	Funktio tallentaa alueen kaikki alialueet ja itsensä sille välitettyyn vektoriin ja kutsuu itseään rekursiivisesti. Lineaarinen alueiden määrässä, jokainen käsitellään varmasti vain kerran.
void **find_stops_in_region**(Region* region, std::vector<Stop*>& stops)	Θ(n)	Funktio tallentaa alueen kaikki pysäkit sille välitettyyn vektoriin ja rekursiivisesti kutsuu itseään kaikille alialueillensa. Lineaarinen pysäkkien ja alueiden määrässä, koska jokainen käsitellään varmasti vain kerran.