Tietorakenteet-kurssin harjoitustyöraportti aiheesta: Union-Find

Cihan Bebek

Cihan Bebek
96337
cihan.bebek@uta.fi
tietojenkäsittelyoppi

1. Irtonaiset ryhmät -tietorakenne (Disjoint-set data structure)

Harjoitustyössä keskeistä oli luoda tietorakenne, joka voi pitää sisällään ryhmän (peräkkäisiä) numeroita siten, että numeroita pystytään yhdistämään toisiinsa eräänlaisiksi ryhmiksi. Tätä varten työnannossa kerrottiin käyttämään nk. union-find, eli disjoint-set -tietorakennetta.

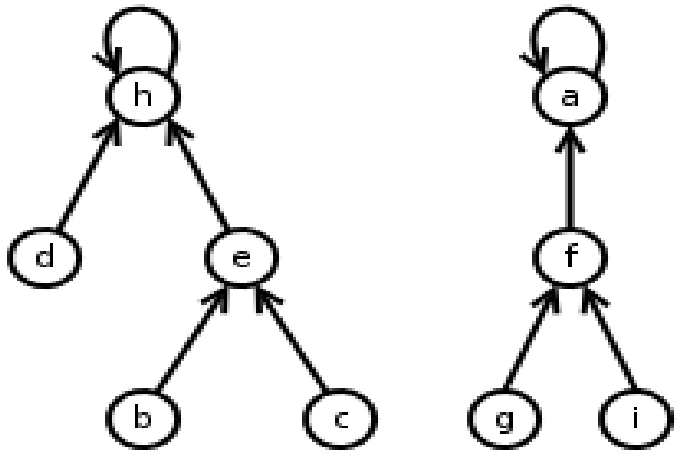
Disjoint-setissä on alkioita, tässä tilanteessa numeroita, joista jokaisella on vanhempi. Mikäli alkiota ei ole vielä yhdistetty mihinkään ryhmään (tällöin alkio itsessään on ”ryhmä”), on sen vanhempi se itse, eikä se ole minkään alkion vanhempi (eli juuri on puun ainoa alkio).

Jotta alkioita voidaan yhdistää toisiinsa, tarvitaan Union-toiminta, joka yksinkertaisesti yhdistää kaksi sille annettua ryhmää.

Jokaisella ryhmällä on juuri. Juurea käytetään periaatteessa ryhmänsä ”edustajana”, jota tarvitaan union- ja find-toiminnoissa. Juuren vanhempi on juuri itse. Mikäli jonkin vanhempi on tämä juurialkio, ne kaksi numeroa ovat samassa ryhmässä, ja mikäli jokin alkio pitää tätä numeroa vanhempanaan, ovat kaikki nämä alkiot samassa ryhmässä.

Mikään alkio ei voi esiintyä kahdessa eri alkioita sisältävässä ryhmässä. Tietorakenteen voi luoda joko siten että ryhmistä voi olla usea duplikaatio (vanhoja ei poisteta niitä yhdistäessä) tai siten että jokaisesta ryhmästä on vain yksi ilmentymä. Tietysti järkevämpää, tehokkaampaa ja selvempää on tehdä siten, että jokaisesta ryhmästä (ja siten alkiosta) on vain yksi ilmentymä.

Tietorakenteen luomiseen on luonnollista käyttää linkitettyä listaa.



Kuva 1: Kuvassa esitetään tietorakenteessa sijaitsevia alkioita, ja niiden ”vanhempi”-viitteiden (nuolien) kautta muodostuneita ryhmiä (h, d, e, b, c) ja (a, f, g, i). Ryhmien juurina ovat alkiot h & a.

1.1 Tietorakenteen operaatiot

Tietorakenteen toimintaan tarvitaan kolme operaatiota/toimintoa:

1. Make-Set

Make-Set(x) luo uuden alkion tietorakenteeseen. Kun uusi alkio luodaan, asetetaan sen vanhemmaksi se itse (x). Tällöin alkio on ”yksinään” tietorakenteessa, eikä ole linkitetty mihinkään muuhun alkioon. Make-Set ei tee mitään, mikäli alkio on jo tietorakenteessa (oli se ryhmässä tai yksin).

2. Union

Union(x, y) yhdistää ne kaksi ryhmää, joissa alkiot x ja y sijaitsee, yhdeksi ryhmäksi. Unionissa asetetaan jomman kumman ryhmän juuren vanhemmaksi jokin toisen ryhmän alkioista. Tällöin voidaan ajatella, että molemmat ryhmät pyyhkitään, ja luodaan uusi ryhmä joka sisältää kaikki alkiot niistä ryhmistä joissa x ja y oli, ja asettaa sen juureksi jomman kumman ryhmän juuren. Mikäli x ja y ovat jo samassa ryhmässä, ei toiminto tee mitään.

3. Find

Find(x) yksinkertaisesti etsii ja palauttaa juurialkion siitä ryhmästä, jossa x on.

1.2 Union

Union siis yhdistää kaksi ryhmää toisiinsa. Yksinkertainen implementaatio Unionista etsii x:n ryhmästä ”peräalkion”, joka asetetaan y:n juuren vanhemmaksi. Kun y on linkitetty x:ään, voidaan ryhmä y poistaa, jolloin jäljelle jää x, joka on vanhan x:n ja y:n unioni. Yksinkertaisessa toteuksessa union vaatii Θ(n) aikaa.

1.2.1 Painotettu-unioni tehostusmenetelmänä

Yksinkertaisessa toteutuksessa voi tulla tilanteita jolloin yhdistetään pienempi ryhmä suurempaan ryhmään. Tämä ei kuitenkaan ole tehokasta, sillä täytyy päivittää useampi määrä vanhempi-viittauksia kuin jos yhdistettäisiin pienempi isompaan. Tämä ongelma voidaan ratkaista nk. **union by rankilla** eli painotuksella. Painotus ei kuitenkaan tehosta unionia, mikäli molemmissa ryhmissä on sama määrä alkioita (jolloinka toteutusaika on jokatapauksessa O(n)).

Painotus voidaan toteuttaa siten, että annetaan alkiolle vanhemman lisäksi toinen arvo, esim. ”rank”, arvo, joka näyttää sen kuinka monta kertaa kyseistä alkiota on unionoitu. Tällöin Unionin pseudokoodi näyttäisi tältä:

```
Algorithm Union(x, y){
    Input: alkiot x ja y, joiden ryhmät alutaan yhdistää
    Output: yhdistetty ryhmä (mikäli x ja y eivät ole samassa ryhmässä)

    rootX = find(x);
    rootY = find(y);

    if(rootX.rank > nodeY.rank)
        nodeY.parent = x;
    else
        nodeX.parent = y;
        if(nodeX.rank == nodeY.rank)
            nodeY.rank++;
```

1.3 Find

Find etsii sen ryhmän juurialkion, jossa x on.
Find on hyvin yksinkertainen algorithmi, ja se tapahtuu yksinkertaisuudessaan näin:

```
Algorithm Find(x)
    Input: alkio x
    Output: sen ryhmän juurialkio, jossa x sijaitsee

    if(x.parent == x)
        return x;
    else
        return Find(x.parent);
```

Tätä voidaan kuitenkin helposti tehostaa huomattava määrä. Tehostusmenetelmänä toimii **path compression** eli polun tiivistyksenä. Path compression toimii siten, että aina kun findia käytetään, se samalla päivittää ryhmän alkioiden vanhemmat ryhmän juureksi. Tällöin polku ”litistyy”, ja tulevien findien suorittaminen nopeutuu.

Path Compressionin kanssa pseudukoodi näyttää jokseenkin tältä:

```
Algorithm Find(x)
    Input: alkio x
    Output: sen ryhmän juurialkio, jossa x sijaitsee. Lisäksi algoritmi kyseisen
    ryhmän läpikäytyjen alkioiden vanhemmaksi ryhmän juuren.

    If(x.parent != x)
        x.parent == find(x.parent);
    return x.parent;
```

Harjoitustyössäni on käytetty molempia tehostusmenetelmiä.

Lähteet:

- Cormen, Leiserson, Rivest, Stein: *Introduction to Algorithms, Third Edition.* The MIT Press, 2009
- Tarjan, Robert E., van Leeuwen, Jan: *Journal of the ACM.* 1948
- Hopcroft, Ullman: *SIAM Journal on Computing.* 1973