Toteutusdokumentti/Suunnittelydokumentti

Yleisrakenne

Ohjelmaan tietorakenteisiin kuuluu neljä puurakennetta ja automaattisesti järjestyvä linkitetty lista. Puurakenteita ovat kaksikolme-hakupuu, treap, threaded-puu ja punamustapuu, ja jokaiselle puulle kuuluu lisäksi oma solmu-luokka.

Käyttäjälle ohjelma tarjoaa mahdollisuuden vertailla puurakenteita. Vertailun voi suorittaa joko rakenteille yksitellen, tai hakea kunkin operaatioin parhaiten ja huonointen suorittavat rakenteet kaikista mahdollisuuksista. Syötteen koon ja alkioiden syöttöjärjestyksen voi myös valita.

Punamustapuiden kohdalla ohjelma muotoilee hyvin läheisesti valmista koodia. Tämä on valittu sen takia, koska nojaavista punamusta puista löytyvä materiaali käyttää hyvin läheisesti valmista koodia puiden toimimisen selittämiseksi.

Aika- ja tilavaatimukset

Listan operaatiot vievät aikaa vaativuudella O(n). Tilavaativuus on samaa luokkaa, koska operaatioita ei ole toteutettu rekursiivisesti. Kaikkien puiden tilavaativuudet koko puulle ovat samoin O(n), jossa n = solmujen lukumäärä. Eri solmuihin kuuluu vaihteleva määrä attribuutteja, mutta koska tallennettujen attribuuttien määrä on syötteestä riippumaton vakiomäärä, tämä ei vaikuta O-analyysiin.

Punamustapuiden tulostusoperaatio vaatii aikaa O(n) (puun läpikäyminen sisäjärjestyksessä) + O(n) (puun alkioiden lisääminen listalle) + O(n) (listan alkioiden läpikäyminen) = O(3n) = O(n) . Solmun etsiminen puusta on vaatimusluokkaa O (log n):

```
etsiSolmu(key)
  //aloitetaan puun juuresta
   if solmu = key tai solmu = null, solmu tai null arvo palautetaan ja metodi päättyy
   else
     solmu = joko oikea tai vasen lapsi
```

Metodin komennot toistuvat korkeintaan puun korkeuden verran, joka punamustapuiden kohdalla voi olla korkeinaan log n, jossa n= solmujen määrä. Metodin yksittäiset operaatiot ovat vakioaikaisia, joten solmun etsiminen vie aikaa O(log n).

Punamustapuiden lisäys ja poisto on vaatimusluokkaa O(log n). Solmulle etsitään uusi paikka lehtisolmuna, ja paikan löytäminen on – kuten etsimisoperaatiossa – korkeintaan puun korkeuden verran solmuja läpikäyvä, O(log n) vaatimusluokaltaan. Puun poisto ja lisäys suorittavat myös tasapainotusoperaatioita, mutta operaatioita ei täydy tehdä sille puolelle puuta, jolle solmua ei ole lisätty. Täten aikavaativuus on siis tarkemmin 2 x O(log n), joka kuitenkin pyöristyy muotoon O(log n).

Tilavaativuudeltaan raskaimmat operaatiot ovat delete, insert ja puun läpikäynti, koska nämä käyttävät hyväkseen rekursiota – O(log n) deletellä ja insertillä, O(n) läpikäynnillä.

Threaded ei ole automaattisesti tasapainottuva puu, joten aikavaativuus pahimmassa tapauksessa insert, delete ja search operaatioilla on O(n) (alkiot lisätty järjestyksessä). Threaded-puilla tulostusoperaatio sujuu nopeimmiten, koska alkiot voidaan tulostaa suuruusjärjestyksessä ilman näiden erillistä järjestämistä: tulostamiseen riittää puun läpikäynti pienimmästä alkiosta tämän seuraaja-pointtereita edeten. Threadedin delete käyttää rekursiota, mutta rekusiokutsuun voidaan päätyä vain kerran per operaatiokerta.

Treap on keskimäärin aikavaativuuksiltaan sama kuin Punamustapuu – koska keko-ominaisuus on toteutettu satunnaisuudella, on silti mahdollista että harvinaisesti päädytään tapaukseen O(n) delete, search, ja insert operaatiolla, jos prioriteetti sattuu jakautumaan erittäin epäoptimaalisesti. Muuten rakenteen nopeudet ovat kuten punamustapuilla. O(log n) deletellä, searchillä ja insertillä, O(n) alkioiden tulostamissa.

```
deleteTreap(solmu)
  if solmu == null; return                                                    (vakio)
  if solmu == lehtisolmu; poistetaan puusta                                   (vakio)
    korjataan puun keko-ominaisuus poistetun solmun vanhemmasta lähtien; return   (lehdestä juureen → O(log n))
  teelehti(solmu)  //tekee solmusta lehtisolmun kiertoja suorittamalla        (juuresta lehteen → O(log n))
  deleteTreap(solmu) //solmu on nyt lehti, metodi suoritetaan rekusiivisesti vain kerran (vakio)
```

Ennen tätä delete-operaatiota on suoritettu solmun etsimisoperaatio (O(log n) kun keko hyvin toteutunut). Aikavaativuus on täten (vakio + 3(O(log n))) = O log(n).

Kaksikolme-puu on vaativuksiltaan samaa luokka Threaded-puun kanssa, alkioiden tulostamista lukuunottamatta. Alkioiden tulostamiseen käytetään O(n), kuten Punamustapuissa ja Treapeissä.

Ideaalijakaumalla Kaksikolmepuussa search, delete ja insert vievät kuitenkin hieman vähemmän aikaa kuin muiden puiden operaatiot. Koska puun haarautuminen on hienojakoisempaa, aikavaativuuden logaritmisuus olisi kolmikantaista muiden puiden kaksikantaisuuden sijasta.

Vertailija-luokan aikavaativuus on puiden operaatioiden aikavaativuudet lisättynä yhteen, ja tähän vielä lisätään listan rakentaminen ja läpikäynti. Nämä pyöristyvät muotoon O(n). Koska search ja delete operaatiot käydään läpi jokaiselle puun solmulle, nousee aikavaativuus siis muotoon $O(n^2)$. Tilavaativuus on luokkaa O(n). Tarkemmin: n = puun solmujen lukumäärä; kaikkia puita vertailtaessa solmuja on kerralla 4n; lisäksi jokaisen puun solmukokoelma on tallennettu listalle, eli O(4n +4n) = O(n).

Ohjelman puutteita

Ohjelmalle mahdollisilla syötteillä ei ole ylärajaa, mutta 10 000 alkiota suurempia syötteitä ei todellakaan suositella - jo 10 000 alkion vertailu vaatii huomattavaa odotusaikaa. Ohjelman valmiista syötteen koista suurin (5000 alkiota) vie aikaa keskimäärin noin 8 sekuntia, kun kaikkia puita vertaillaan keskenään. Negatiiviset syötteet käsitellään 0:n alkion syötteinä.

Ohjelmaa suositellaan vain keskimääräisten tendenssien laskemiseen. Puihin lisätä satunnainen joukko arvoja Javan valmista Random-luokkaa käyttäen, eri arvojoukot joka puuhun myös kaikkia yhtäaikaa vertailtaessa. Puut eivät tallenna samaa arvoa kahteen kertaan, joten kokoerot puiden välillä voivat huonolla tuurilla olla suuret. Tällöin ajankulutus voi antaa joistain puista tehokkaamman vaikutelman muihin verrattuna, kuin on totta käytännössä. Jos testituloksien keskiarvoja tarkastellaan useamman käyttökerran jälkeen, tällöin voidaan todennäköisemmin saada vertailukelpoisia tuloksia.

Alkioiden järjestäminen ja tallentaminen vertailutarkoituksiin olisi myös mahdollista toteuttaa nopeammin. Ohjelma käyttää nyt automaattisesti järjestyvää listaa, jonka järjestyminen toimii bubble-sortin kaltaisesti (alkiolle etsitään aina oikea paikka sitä lisättäessä). Järjestämättömälle rengaslistalle alkion lisääminen olisi vakioaikaista, ja tehokkaammalla järjestysalgoritmilla ohjelman vertailuosion suorittamista voitaisiin mahdollisesti nopeuttaa.

Lähteet:

http://en.wikipedia.org/wiki/Red%E2%80%93black_tree
http://en.wikipedia.org/wiki/Sorting_algorithm