Toteutusdokumentti

Ohjelman rakenne:

Ohjelman keskeisin komponentti on PuunTutkija luokka, joka vastaa hakupuiden suorituskyvyn testauksesta. Luokalle annetaan luomisen yhteydessä tutkittavat hakupuut, joihin suoritettavat mittaukset sitten kohdistetaan. Suorituskyvyn tutkimiseen luokka käyttää Sekuntikello luokkaa, jolla voidaan mitata suoritukseen kuluvaa aikaa nanosekunteina.

Mitatut ajat talletetaan Mittaustulos olioon, joka kirjaa tulokset ja laskee näistä tilastoja. Mittaustulos olioon on tarkoitus tallentaa useita saman operaation suoritusaikoja jolloin mahdollisia virheitä mitatuissa ajoissa voidaan arvioida tai poistaa. Ohjelman käyttöliittymä saa parametrina Puuntutkija olion. Käyttöliittymän vastuulla on antaa PuunTutkijalle tietojoukot suoritettaviin mittauksiin.

Saavutetut tilavaativuudet:

Kukin puu sisältää suoraan vain yhden viitemuuttujan, joka on puun juuri. O(1)

Jokaista puuhun lisättävää alkiota kohti on luotava uusi solmu, joista kukin sisältää 3 viitemuuttujaa 3*O(1) ja avaimen, joka on jokin vertailukelpoinen olio. Tässä toteutuksessa integer jolle pätee O(1).

Tällöin koko solmun tilavaativuus on 4 * O(1) = O(1).

Punamustan puun solmut sisältävät edellä mainitun lisäksi nimetyn luokan Vari arvon ja AVL puun solmut vastaavasti kokonaislukuna talletettavan korkeus arvon. Kummankin näistä tilavaativuus on vakio O(1), joten ne eivät vaikuta ratkaisevasti solmun kokoluokkaan. O(1) + O(1) = O(1)

Näin ollen kaikki puut ovat tilavaativuudeltaan O(n), missä n on puussa esiintyvien solmujen määrä.

Saavutetut aikavaativuudet:

Pseudokoodeissa jätetään merkitsemättä vakioaikaisia suorituksia, kuten muuttujan arvon sijoittamisia yms. mikäli niillä ei ole merkittävää vaikutusta kyseisen metodin toimintaan.

Selityksissä h on puun korkeus ja n puun solmujen määrä.

Binääripuiden korkeuden tiedetään aina olevan välillä log2(n) <= h <= n.

Hakeminen:

```
hae(arvo){
        solmu = juuri  //O (1)
        while(solmu != null ja solmu.arvo != arvo){  //  O (h)
                if(solmu.arvo > arvo) solmu = solmu.vasen  // O (1)
                else solmu = solmu.oikea       // O (1)
        }
        return solmu
}
```

Pseudokoodista nähdään nopeasti että haku toiminto sisältää yhden silmukan. Jokaisella kierroksella solmu siirtyy puussa yhden kerroksen alaspäin. Tällöin hakemisen aikavaativuus on O(h).

Lisääminen:

```
Lisää(arvo){
        solmu = juuri  //O (1)
        uusi //tehdään uusi solmu jonka avain on parametrina saatu O(1)
        if(solmu == null) uusi = juuri ; return
        while(solmu != null){  //   O (h)
                if(solmu.arvo == arvo) return
                else if(solmu.arvo > arvo) {
                        if(solmu.vasen == null) solmu.vasen = uusi ; break
                        else solmu = solmu.vasen
                } else {
```

```
                    if(solmu.oikea == null) solmu.oikea = uusi ; break
                    else solmu = solmu.oikea
                    }
        }
        uusi.vanhempi = solmu
}
```

Metodi sisältää yhden silmukan joka taas etenee kerroksen kerrallaan täten O(h).

Edeltäjän etsintä:

```
edeltäjä(solmu){
        solmu = solmu.oikea ()
        while(solmu.vasen != null) solmu = solmu.vasen //O(h), missä h on solmusta alkavan alipuun korkeus
        return solmu
}
```

Edeltäjän hakeminen siis O(h).

Solmun korvaaminen

```
korvaa(vanha, uusi) {
if( vanha.vanhempi == null ) juuri = uusi
else {
        if(vanha = vanha.vanhempi.vasen) vanhempi.vasen = uusi
        else vanhempi.oikea = uusi
}
if(uusi != null) uusi.vanhempi = vanha.vanhempi
}
```

Korvaamisen kaikki operaatiot vakioaikaisia täten O(1).

Solmun poistaminen:

poista(arvo){

solmu = hae(avain) // O(h)

if(solmu == null) return // O(1)

else if(solmu.vasen != null ja solmu.oikea != null) solmu = edeltäjä(solmu) // O(h)

if(solmu.vasen == null) lapsi = solmu.oikea // O(1)

else lapsi = solmu.vasen // O(1)

korvaa(solmu, lapsi) // O(1)

}

Merkitsevät aikaluokat ovatO(h) ja O(h)+O(h) = O(h). Täten poisto-operaatio on aikaluokkaa O(h).

Kierrot:

vasenKierto(solmu){

oikea = solmu.oikea

korvaa(solmu, oikea)

solmu.oikea = oikea.vasen

if(oikea.vasen != null) oikea.vasen.vanhempi = solmu

oikea.vasen = solmu

solmu.vanhempi = oikea

}

Kaikki operaatiot ovat vakioaikaisia ja ei sisällä silmukoita tai rekursiota. Oikea kierto on identtinen mutta lapsien käsittelyn suhteen peilikuva. Täten kaikki kierto-operaatiot luokkaa O(1).

SPLAY:

```
Splay(solmu){
while(solmu.vanhempi != null){ // O(h)
        vanhempi = solmu.vanhempi
        isovanhempi = solmu.vanhempi.vanhempi
        if(isovanhempi == null)  {
                if(vanhempi.vasen = =solmu) oikeaKierto(vanhempi) // kierrot O(1)
                else vasenKierto(vanhempi)
        } else {
        if(vanhempi.vasen = = solmu){
        if(vanhempi.vanhempi.vasen = vanhempi) oikeaKierto(isovanhempi) ; vasenKierto(vanhempi)
        else oikeaKierto(vanhempi) ; oikeaKierto(vanhempi)
        }
        else {
        if(vanhempi.vanhempi.oikea = vanhempi) vasenKierto(isovanhempi) ; oikeaKierto(vanhempi)
        else vasenKierto(vanhempi) ; vasenKierto(vanhempi)

        }
        }
}
```

Splayn idea on nostaa parametrin solmu puun huipulle. Jokainen silmukan kierros nostaa solmua vähintään yhden tason ja kunkin kierroksen aikavaativuus on O(1) niin aikaluokka on O(h).

AVL puun tasapainotus:

```
AVLtasapainota(solmu){
solmu = solmu.vanhempi
while(solmu != null){ //O(h)
        if(solmu.vasen.korkeus > solmu.oikea.korkeus){
        if(solmu.vasen.vasen.korkeus > solmu.vasen.oikea.korkeus) oikeaKierto(solmu)
        else{ vasenKierto(solmu.oikea)  ; oikeaKierto(solmu)}
        return;
        } else{
        if(solmu.oikea.oikea.korkeus > solmu.oikea.vasen.korkeus) vasenKiero(solmu)
        else {oikeaKierto(solmu.vasen) ; vasenKiero(solmu)}
        return;
        }
        }
}
```

Tasapainotus sisältää yhden silmukan minkä sisällä suoritetaan kierto-operaatioita, jotka siis ovat O(1). Silmukka etenee enintään puun lehdestä juureen, joten aikavaativuus O(h).

Suorituskykyvertailu:

Selityksissä h on puun korkeus ja n puun solmujen määrä.

Binääripuiden korkeuden tiedetään aina olevan välillä log2(n) <= h <= n.

Haku tapahtuu kaikissa puissa lähes identtisellä tavalla ja se on suhteessa puun korkeuteen. Tällöin hakuoperaatioiden suorituskyky riippuu täysin siitä miten puu on rakentunut.

Haussa etsittävää arvoa verrataan aina solmun arvoon alkaen puun juuresta. Mikäli etsittävä avain on pienempi, edetään solmun vasempaan lapseen. Jos etsittävä avain puolestaan on suurempi, edetään oikeaan lapseen. Avaimien arvojen vertaaminen on vakioaikaista (O(1)) ja vertailuja suoritetaan enintään puun korkeuden verran.

Binäärinen hakupuu on korkeudeltaan pahimmillaan n esimerkiksi silloin kun on lisätty arvoja kasvavassa järjestyksessä. Tällöin hakuaika on myös vastaavasti O(n). Kuitenkin sattumanvaraisia lisäys ja poisto-operaatioita tehtäessä puun korkeus lähenee arvoa log(n), joten hakuaika on myös vastaavasti keskimäärin O(log(n)).

Punamusta puu ja AVL - puu ovat tasapainotettuja, joten niiden lisäys ja poisto-operaatiot takaavat että puun korkeus on aina logaritminen siinä olevien solmujen määrään nähden ja tällöin hakuaika on myös ilman muuta logaritminen eli O(log(n)).

Splay-puun haku toimii muuten vastaavalla tavalla kuin binäärisen hakupuun, mutta etsitylle solmulle tehdään Splay operaatio. Splay operaatio siirtää solmun puun juureksi käyttäen kierto-operaatioita. Yksi kierto-operaatio on suoritusajaltaan vakio ja niitä suoritetaan splay-operaatioissa enintään puun korkeuden verran. Koska yksi kierto nostaa solmua yhden asteen ylöspäin ja Puun korkeus voi splay puussa olla pahimmassa tapauksessa n, suoritetaan pahimmillaan n-1 kiertoa. Tämä antaisi ymmärtää että pahimmassa tapauksessa suoritusaika olisi O(n). Kuitenkin koska Splay-operaatio muokkaa puuta ja operaation jälkeen etsitty arvo on siirtynyt puun huipulle, tästä saadaan monimutkaisen matematiikan kautta selvitettyä että keskimääräinen suoritusaika kaikille splay-puun operaatioille on pahimmassakin tapauksessa keskimäärin O(log(n)). Tällöin myös haun täyty tapahtua pahimmassakin tapauksessa keskimäärin ajassa O(log(n)).

Lisäys binäärisessä hakupuussa tapahtuu hyvin vastaavalla tavalla kuin haku. Puuta edetään kerros kerrokselta alaspäin siirtyen joko oikealle tai vasemmalle kunnes ei voida edetä pitemmälle. Tämän jälkeen asetetaan osoittimille oikeat arvot. Tällöin lisäyksen aikavaativuus on suhteessa puun korkeuteen. Binäärisen hakupuun korkeus käsiteltiin jo aiemmin joten lisäysaika on keskimäärin O(log(n)) ja pahimmillaan O(n).

Lisäys AVL-puussa alkaa aivan kuten binäärisessäkin hakupuussa. Kun solmu on lisätty puuhun, tehdään kuitenkin vielä tasapainotusoperaatio, jonka idea on taata että puun korkeus on logaritminen. Tasapainotus koostuu 0 tai enintään (log(n) - 1):stä kierto-operaatiosta. Kierto-operaation aikavaativuuden todettiin jo

aiemmin olevan vakio, joten tasapainotuksen aikavaativuus on vastaavasti pahimmillaan luokkaa O(log(n)). Tällöin koko lisäysoperaatio on myös O(log(n)).

Punamustan puun lisäys toimii vastaavalla idealla kuin AVL-puun, mutta tasapainotus poikkeaa toteutukseltaan huomattavasti. Tasapainotuksessa on 5 eri tilannetta. Näistä kaikki ovat suoritusajaltaan vakioaikaisia, mutta tilanteessa jossa lisätyn solmun vanhemman sisarsolmu on punainen, joudutaan nousemaan puuta ylöspäin rekursiivisesti. Nousu tapahtuu 2 kerrosta kerrallaan joten pahimmassakin tapauksessa toistoja tulee log(n) / 2 ja kukin tekee vakioaikaisia operaatioita. Tästä seuraa että suoritus pysyy aina aikaluokassa O(log(n)).

Splay puu toimii binäärisen hakupuun tavoin mutta lisäyksen jälkeen lisätty solmu tuodaan puun juureksi splay operaation avulla. Splay puun molempien operaatioiden aikavaativuus on jo selvillä joten lisäys on pahimmassakin tapauksessa keskimääräisesti aikaluokassa O(log(n)).

Poisto Binäärisessä hakupuussa alkaa haku-toiminnolla jolla etsitään poistettava solmu puusta. Tämän jälkeen solmu poistetaan yhdellä kolmesta tavasta, riippuen siitä montako lasta kyseisellä solmulla on. Kaikki 3 tapausta ovat kuitenkin vakioaikaisia joten poisto-operaation tehokkuus riippuu täysin haun tehokkuudesta joka on jo käsitelty aiemmin. Näin ollen poisto on pahimmillaan aikaluokkaa O(n) ja keskimäärin O(log(n)).

AVL-puun poisto alkaa vastaavalla tavalla kuin binäärisen hakupuun. Solmun poistamisen jälkeen tehdään vielä jo aiemmin käsitelty tasapainotusoperaatio. Näistä saadaan pahimman tapauksen aikavaativuudeksi O(log(n)).

Punamusta puun poisto koostuu lisäyksen tavoin vakioaikaisista tapauksista, joista vain yksi sisältää mahdollisen rekursion. Pahimmassa tapauksessa edetään taas ylöspäin puuta ja puun ollessa pahimmillaankin logaritmisen korkuinen voidaan rekursio suorittaa enintään log(n) kertaa. Täten poisto-operaatiokin pysyy aikaluokassa O(log(n)).

Splay puun poistossa suoritetaan ensin poistettavalle solmulle splay operaatio, jolloin se tulee puun juureksi. Tämän jälkeen poistetaan solmu ja korvataan se edeltäjällään (vasemman alipuun suurin alkio). Splayn aikavaativuus on jo käsitelty joten luokaksi saadaan taas pahimmassakin tapauksessa keskimäärin O(log(n)).

Puutteet ja parannusehdotukset:

- Graafinen käyttöliittymä ja/tai puiden tulostus.
- Teet koko työ uudestaan.
- Tieteelliset matemaattiset todistukset aikavaativuuksille ja korkeuksille.

Lähteet:

- www.wikipedia.com/*
- www.wikipedia.fi/*
- Tira kurssimateriaali