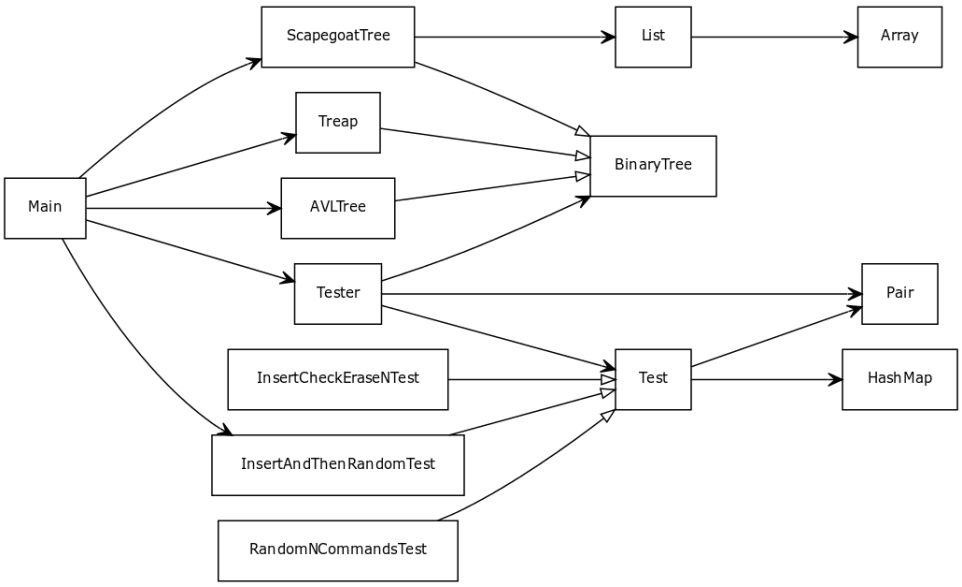
Toteutusdokumentti

Ohjelman rakenne

Ohjelma koostuu geneerisistä luokista `AVLTree`, `Treap` ja `ScapegoatTree`, jotka perivät abstraktin geneerisen luokan `BinaryTree`. Kukin näistä toteuttaa tasapainotetun binääripuun omalla tavallaan.

Lisäksi ohjelmassa on mukana luokka `Tester`, joka ottaa parametreikseen abstraktin luokan `Test` periviä testiluokkia, jotka generoivat syötteen ja antavat sen annetulle puulle testattavaksi. Kunkin testin jälkeen `Tester` tulostaa raportin testin onnistuneisuudesta ja siihen kuluneesta ajasta. Testaaja käyttää hajautustaulua pitääkseen kirjaa siitä, mitä lukuja puussa kuuluisi olla sillä hetkellä.

Lisäksi luokat hyödyntävät `util`-paketissa olevia tietorakenteita, kuten `Pair`, `HashMap` ja `List`.



Kuva 1. Ohjelman luokkakaavio.

Aika- ja tilavaativuudet

AVL-puu

AVL-ehto sanoo, että kunkin solmun lapsien syvyyksien ero saa olla enintään yksi. Todistetaan puun syvyyden olevan $O(\log n)$, mikäli AVL-ehto pätee.

Olkoon A_n pienin määrä solmuja, jotka tarvitaan, että puun syvyys on n

ja AVL-ehto pätee. Selvästi $A_0 = 1$, $A_1 = 2$ ja $A_n = 1 + A_{n-1} + A_{n-2}$, sillä konstruktio tapauksesta A_{n-1} tapaukseen A_n tarkoittaa, että A_{n-1}:n juureen kiinnitetään uusi juuri, ja nyt uusi juuri vaatii AVL-ehdon säilymiseksi toiseksi lapsekseen vähintään $n-2$-syvyisen puun.

Koska A on aidosti kasvava, niin $A_n \geq 2A_{n-2}$. Puun syvyyden on siis oltava logaritminen AVL-ehdon pätiessä.

Voidaan todistaa, että lisäyksen tai poiston yhteydessä rikkoutuneen AVL-ehdon korjaamiseksi riittää tehdä enintään $O(\log n)$ kappaletta $O(1)$-aikaisia kääntöjä.

Treap

Treapin jokainen solmu sisältää prioriteetin ja säilytettävän arvon. Arvoja säilytetään puussa siten, että pienemmät arvot ovat aina vasemmassa alipuussa ja suurin prioriteetti ylimpänä. Tämä määrää puun rakenteen yksiselitteisesti, olettaen että prioriteetit ovat keskenään eriäviä. Koska törmäyksiä tulee vain harvoin, voimme käytännössä olettaa näin olevan.

Perusidea puun odotusarvoisen $O(logn)$ syvyyden todistuksessa on tarkastella solmujen odotusarvoista syvyyttä, mutta todistus on sen verran matematiikkapainotteinen, ettei sitä luultavasti tämän työn puitteissa kannata toteuttaa.

Syntipukkipuu

Syntipukkipuun uudelleenrakennus tarkoittaa, että uudelleenrakennattavan solmu ja kaikki sen jälkeläiset laitetaan järjestyksessä listaan sisäjärjestyksessä lineaarisessa ajassa niiden määrään nähden. Tämän jälkeen alipuun juureksi valitaan näistä mediaanialkio, jonka jälkeen tätä pienemmistä alkioista rakennetaan rekursiivisesti vasen alipuu ja suuremmista oikea alipuu samalla järjestelmällä. Näin ollen siis vasemman ja oikean alipuun koot eroavat enintään yhdellä jälleenrakennuksen jälkeen.

Alipuu rakennetaan uusiksi, kun sen oikean tai vasemman alipuun koko on yli $\alpha \cdot k$, missä k on nykyisen alipuun koko. Tämä tarkoittaa sitä, että kummassakin alipuussa on nyt enintään $\frac{k}{2}$ solmua. Jos alipuuhun lisätään m alkiota, niin tasapainoehto voi rikkoutua seuraavan kerran aikaisintaan kun epäyhtälö $\frac{k}{2} + m > \alpha \cdot (k+m)$ pätee. Tämä tarkoittaa siis, että $m > \frac{k}{2-2\alpha} - k$, mikä selvästi tarkoittaa lisäyksien määrän on oltava lineaarinen alipuun kokoon nähden ennen kuin uudelleenrakennusta tarvitaan.

Solmuihin talletetaan myös tieto siitä, mikä on suurin alipuun koko joka sillä on ollut enimmillään viimeisimmän uudelleenrakennuksen jälkeen. Mikäli

tämän arvon ja α:n tulo on suurempi kuin alipuun nykyinen koko, niin alipuu uudelleenrakennetaan. Tämä voi tapahtua vain poiston seurauksena.

Oletetaan alipuun koon olevan nykyisellään k ja juuri äsken uudelleenrakennettu. Jos alipuusta poistetaan m alkiota, niin uudelleenrakennus tapahtuu aikaisintaan kun $k - m \leq k\alpha$, eli $m \geq (1 - \alpha)k$. Näin ollen m on jälleen lineaarinen suhteessa nykyiseen kokoon, eli lineaaristen uudelleenrakennusten välissä voidaan tehdä lineaarinen määrä lisäyksiä.

Näiden ansiosta puun syvyys säilyy logaritmisena, eli lisäys ja poisto ovat keskimäärin logaritmisia.

Hakeminen

Sekä AVL-puu että syntipukkipuu ovat syvyydeltään $O(\log n)$, joten tietyn solmun hakeminen puusta vie vain $O(\log n)$ aikaa. Treap on odotusarvoisesti ja lähes varmasti myös syvyydeltään logaritminen suhteessa puun kokoon, joten myös sille voidaan olettaa haun vievän $O(\log n)$ aikaa.

Muistivaativuus

Kukin solmu vaatii vain vakiomäärän tilaa, joten kaikkien puiden tilavaativuus on $\Theta(n)$.

Suorituskykyvertailu

Nykyisien tuloksien perusteella treap on vakiokertoimiensa suhteen optimaalisempi kuin AVL-puu tai syntipukkipuu. Tämä johtunee osittain siitä, ettei treapin tarvitse päivittää pituuttaan tai kokoaan tai uudelleenrakentaa itseään. AVL-puu on aavistuksen huonompi vakiokertoimiltaan, mutta päihittää syntipukkipuun yli kaksinkertaisella nopeudella suuremmissa tapauksissa. Ilmeisesti syntipukkipuuta hidastaa erityisesti puun uudelleenrakennus, vaikka sen operaatiot ovatkin asymptoottisesti $O(\log n)$.

On kuitenkin tärkeää muistaa, että tulokset eivät ole aivan vertailukelpoisia, sillä implementaatiota voisi varmasti optimoida kaikille luokille.

Parannusehdotukset

Binääripuun abstraktia luokkaa olisi mahdollista täydentää vielä monin erilaisin funktioin, kuten pienimmän arvon etsiminen joka ei ole annettua arvoa pienempi (engl. *lower bound*) tai puun arvojen läpi iteroimisen mahdollisuus.

Lisäksi, kuten suorituskykyvertailussa todettiin, niin kaikkien puiden implementaatioita voisi todennäköisesti optimoida lisää.

Lähteet

• Wikipedia. (n.d.). AVL Tree. Viitattu 28.9.2018
https://en.wikipedia.org/wiki/AVL_tree
• Wikipedia. (n.d.). Treap. Viitattu 28.9.2018
https://en.wikipedia.org/wiki/Treap
• Wikipedia. (n.d.). AVL Tree. Viitattu 28.9.2018
https://en.wikipedia.org/wiki/Scapegoat_tree
• Jeff Erickson. (n.d.). Randomized Binary Search Trees. Viitattu 12.10.2018
http://jeffe.cs.illinois.edu/teaching/algorithms/notes/10-treaps.pdf