Testausdokumentti

Mitä testattu

Yksikkötestaus

Ohjelmassa on JUnitilla tehty yksikkötestaus paketeissa `util` ja `datastructure` olevia luokkia varten. Nämä testit testaavat näiden luokkien perustoiminnallisuuksien toiminnan. Paketin `datastructure` testiluokat ovat valitettavasti pitkälti kopioita toisistaan, sillä testien periminen ei vaikuttanut olevan mahdollista Javalla.

Toisaalta nämä luokat oli tärkeää testata erikseen, sillä niissä on kaikissa omat erilaisuutensa lisäyksen ja poiston suhteen. Luokkaa `AVLTree` varten on myös yksikkötesti AVL-ehdon ja solmuihin merkityn korkeuden testaamiseen, ja luokka `Treap` tarkistaa yksikkötesteissään kekoehdon pätevyyden.

Suorituskykytestaus

Yksikkötestien lisäksi ohjelmassa on suorituskykytestaus. Ohjelman suorituskykytestaus koostuu kolmesta eri testiryhmästä, jotka kukin ajetaan kullekin puutyypille.

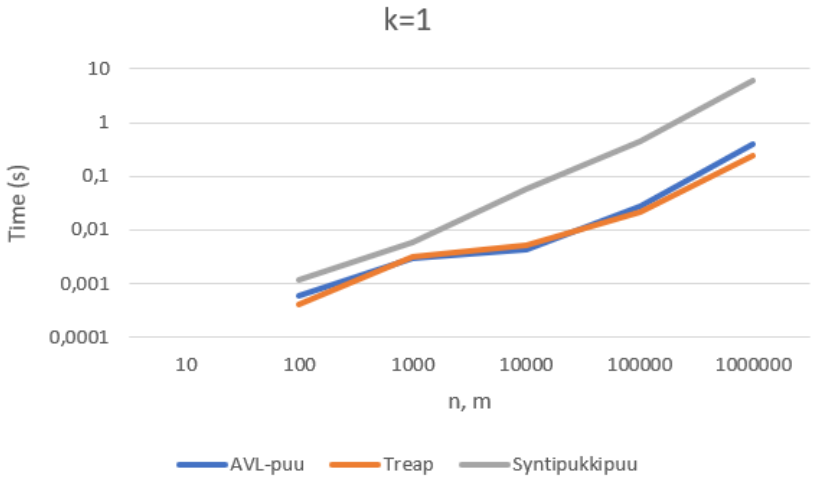
Kukin testi koostuu kolmesta parametrista, eli n:stä, m:stä ja k:sta. Tämä tarkoittaa testiä, jossa lisätään ensin n kokonaislukua väliltä $[0, k[$, jonka jälkeen tulee vielä m operaatiota, jotka ovat sattumanvaraisesti joko lisäyksiä, sisältämiskyselyitä tai poistoja, ja myös näiden käyttämät luvut ovat väliltä $[0, k[$. Näille testeille annetaan kuitenkin sama siemenluku, joten kaikki puut ajavat täsmälleen samat testit.

Testejä valitessa koettiin järkeväksi, että pätisi $n = m$. Tämä siksi, että puun koko halutaan tarpeeksi suureksi ennen kuin sitä kunnolla testataan ja viimeisten m operaation aikana puun koon muutosten odotusarvo on nolla. Kussakin kolmesta testiryhmässä parametrit n ja m käyvät seuraavat arvot läpi: 10, 10^2, 10^3, 10^4, 10^5 ja 10^6.

Kolme testiryhmää toisistaan erottava arvo on k, joka ensimmäisessä ryhmässä on 1, toisessa 100 ja kolmannessa 1000000. Yleisesti ottaen ensimmäinen ryhmä on binääripuiden syvyyksien kannalta pahin tapaus, sillä se sisältää vain samaa lukua. Näin ollen uusi arvo menisi aina vasemmanpuoleisimmaksi lapseksi puuhun, ja puun koko kasvaisi lineaarisesti. Toki näin ei tasapainottuvien puiden kanssa ole, mutta tämä pakottaa puita korjaamaan tasapainoaan silti useammin. Sen sijaan k:n suurin arvo käytännössä takaa sen, että suurin osa puun arvoista ei ole samoja, ja näin ollen niiden sijoittelu on tasaisempaa. Testiryhmät on suoritettu viidesti, ja sitten tuloksista on otettu keskiarvo.

Seuraavista visualisaatioista puuttuvat arvot tapaukselta $n = m = 10$, sillä logaritmiseen asteikkoon ei voinut sijoittaa niiden käyttämää nollan millisekuntin aikaa.

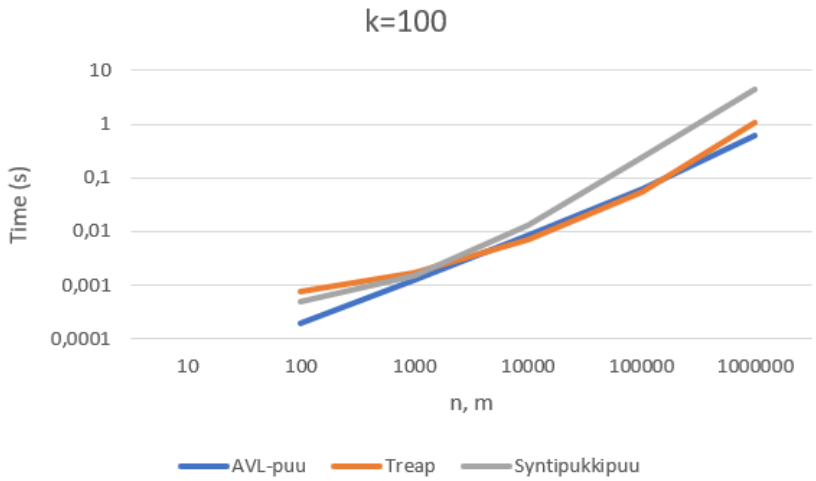
Testiryhmä k=1



Tässä ryhmässä syntipukkipuu on ylivoimaisesti hitain. Tämä selittyy pitkälti sillä, että syntipukkipuun tasapainotusoperaatio on suhteellisen hidas. Satunnaisemmalla datalla tasapainotusta tapahtuu jonkin verran harvinaisemmin, mutta tapauksessa $k = 1$ tavara pakkautuu vasempaan alipuuhun pakottaen tasapainotuksia useammin.

AVL-puu ja treap ovat lähellä toisiaan, mutta treap lienee aavistuksen nopeampi sen AVL-puuta kevyemmän alipuupyöritysoperaation vuoksi.

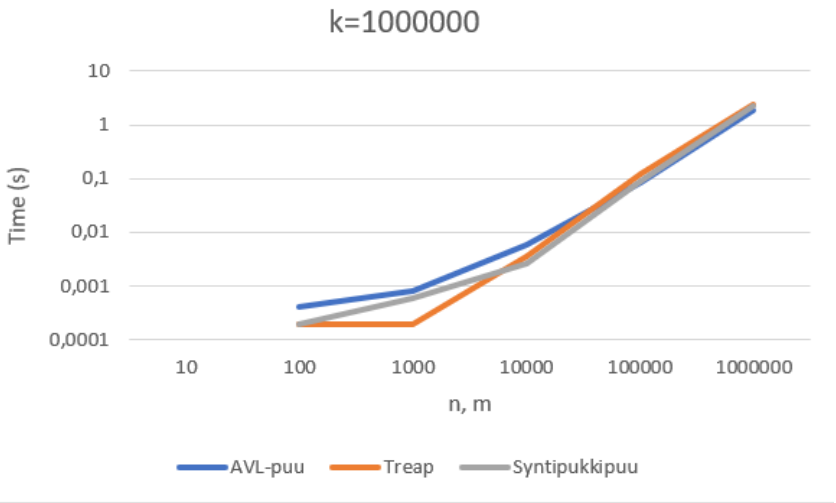
Testiryhmä k=100



Tässä ryhmässä syntipukkipuu on edelleen hitain samoista syistä kuin aiemminkin, mutta arvojen monipuolistumisen ansiosta se on jo aavistuksen nopeampi.

AVL-puu ohittaa tässä ryhmässä treapin nopeuedessa oletettavasti sen takia, että tässä testiryhmässä sen syvyydestä tulee odotusarvoisesti pienempi kuin tapauksessa $k = 1$.

Testiryhmä k=1000000



Viimeisessä testiryhmässä suuremmilla arvoilla kaikkien puiden nopeudet lähestyvät toisiaan. Syntipukkipuun tarvitsee tasapainottaa itseään entistä harvemmin, ja tämän vuoksi se nopeutuu merkittävästi ohittaen treapin nopeudessaan.

Treap kärsinee odotusarvoisesti suurimmasta syvyydestään, ja näin ollen häviää AVL-puulle, jonka tasapainotusoperaatio on kevyt ja jonka syvyys pysyy erittäin pienenä.

Testien toisto

Yksikkötestit voi ajaa helpoiten NetBeansin omalla testauskomennolla. Suorituskykytestauksen voi suorittaa ajamalla ohjelman käyttöohjeen mukaisella `java`-komennolla ilman lippuja.