Testausdokumentti

Kaikki harjoitustyön koodi on testattu Junit-testeillä niin, että koko työn testikattavuus on 100 % Main- ja Demo-luokkia lukuun ottamatta. Jokaiselle luokalle on olemassa oma testiluokkansa, paitsi luokille Main, Demo, Node ja TrieNode. Kaksi ensimmäistä luokkaa muodostavat käyttöliittymän ja niiden on todettu toimivan oikein manuaalisella testauksella – käyttöliittymä toimii sillä tavalla kuin sen on tarkoituskin. Kaksi jälkimmäistä luokkaa sisältävät ainoastaan yksinkertaisia settereitä ja gettereitä, joiden toiminnan oikeellisuus varmistuu, kun toteutetun koodin muut osat on testattu kattavasti.

Junit-testit on tehty niin, että ne testaavat paitsi puiden normaalia toimintaa, myös niiden erikoistapauksia. Lisäksi AVL-puun ja splay-puun rakenne – se, että puut ovat varmasti oikeanmallisia - on testattu kokeilemalla, että niiden alkioiden leveyssuuntainen tulostus on oikeanlainen. Puiden rakennetta ja toimintaa on tarkkailtu myös manuaalisesti koodia kirjoitettaessa ja debugatessa. JUnit-testit voidaan toistaa kehitysympäristössä ajamalla testikansiossa olevat testitiedostot.

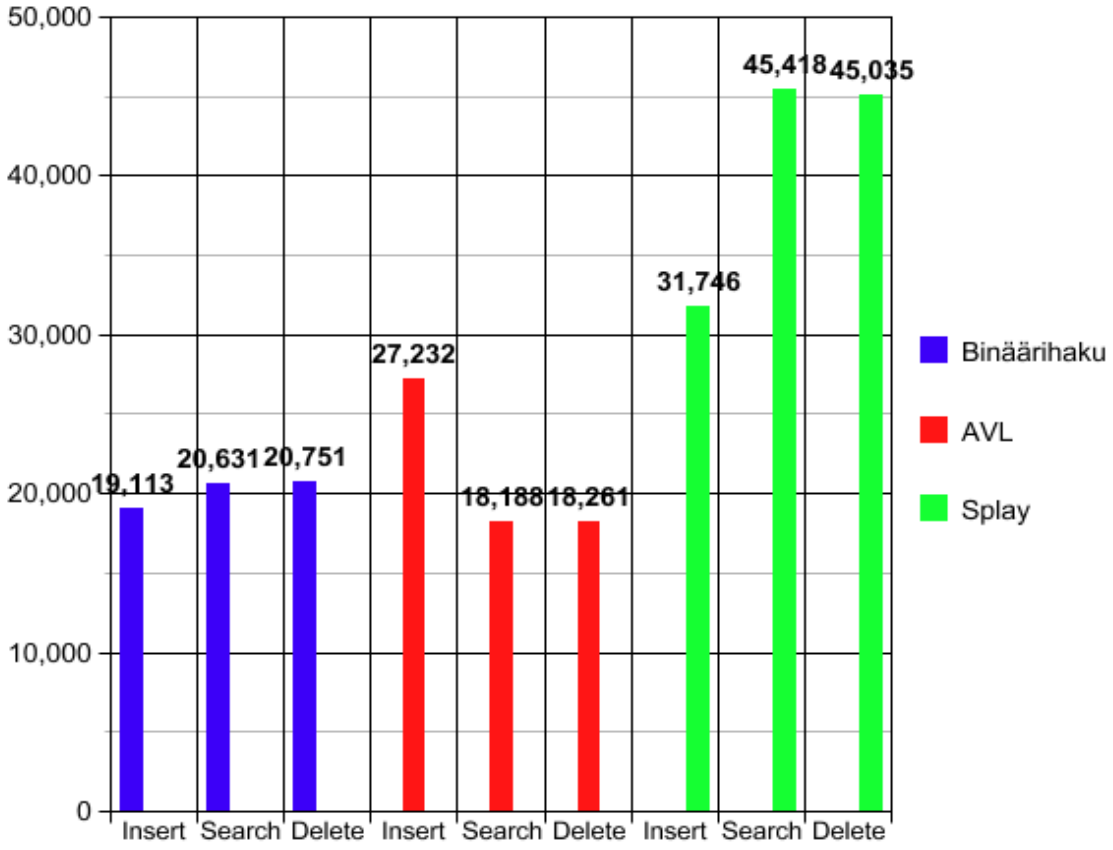
Ajamalla Main.java voidaan vertailla ja testata interaktiivisesti eri puiden lisäys-, haku- ja poistonopeuksia syötteillä, joiden joiden suuruus vaihtelee välillä $0\text{-}2^{32}$ ja toistokertojen määrä välillä 1 – 20 000 000. Binäärihakupuu, AVL-puu ja splay-puu toimivat vakaasti aina 20 000 000 sattumanvaraiseen alkioon asti operaatioiden yhteenlasketun suoritusajan ollessa binäärihakupuun ja AVL:n tapauksessa noin 60 sekuntia ja splayn tapauksessa 120 sekuntia. Trie-puu hajoaa jossain kohtaa 2 000 000 ja 2 500 000 sattumanvaraisen alkion välillä operaatioiden yhteenlasketun suoritusajan ollessa 2 000 000 alkion kanssa noin 40 sekuntia.

AVL- ja splay-puu toimivat hyvin 20 000 000 alkioon asti myös järjestyksessä olevilla luvuilla (AVL 70 s. ja splay 6 s.), mutta binäärinen hakupuu jämähtää paikoilleen järjestyksessä olevan syötemäärän kanssa. Trie-puu rikkoutuu jossain kohtaa 8 500 000 ja 9 000 000 järjestyksessä olevan alkion kohdalla, suoritusajan ollessa 8 500 000 alkion kohdalla noin 380 sekuntia.

Mainin toteutus on tahallaan jätetty sellaiseksi, että myös kaikkiin puihin voidaan yrittää lisätä 20 000 000 sattumanvaraista tai ei-sattumanvaraista alkiota, vaikka puut hajoavatkin joillain syötemäärillä, sillä ohjelman tarkoitus on nimenomaan erisuuruisten puiden testaus.

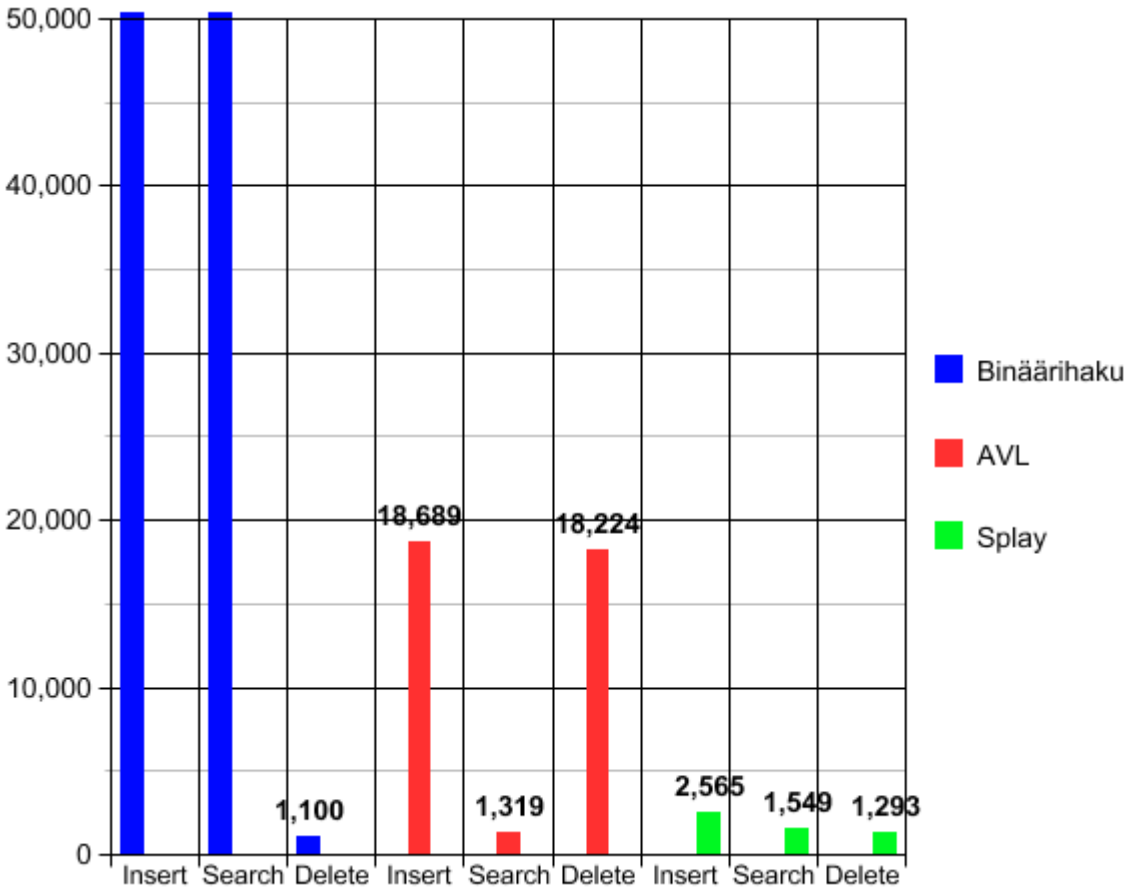
Ohjelman demokomento testaa puita syötteillä, joiden lukumäärä on 10 000, 100 000 ja 1 000 000 sattumanvaraista alkiota ja joiden suuruus vaihtelee välillä -10 000 000 – 10 000 000. Lopuksi demo tulostaa kaikkien puiden - paitsi trie-puun – solmut esi-, sisä-, jälki- ja leveyssuuntaisessa järjestyksessä.

Kuva 1



Binäärihakupuun, AVL-puun ja Splay-puun perusoperaatioiden keskiarvoiset (kolmen suorituksen keskiarvo) suoritusajat millisekunteina, kun jokaisen operaation toistomäärä on 20 000 000 sattumanvaraisilla syötteillä, joiden suuruus on $0\text{-}2^{32}$.

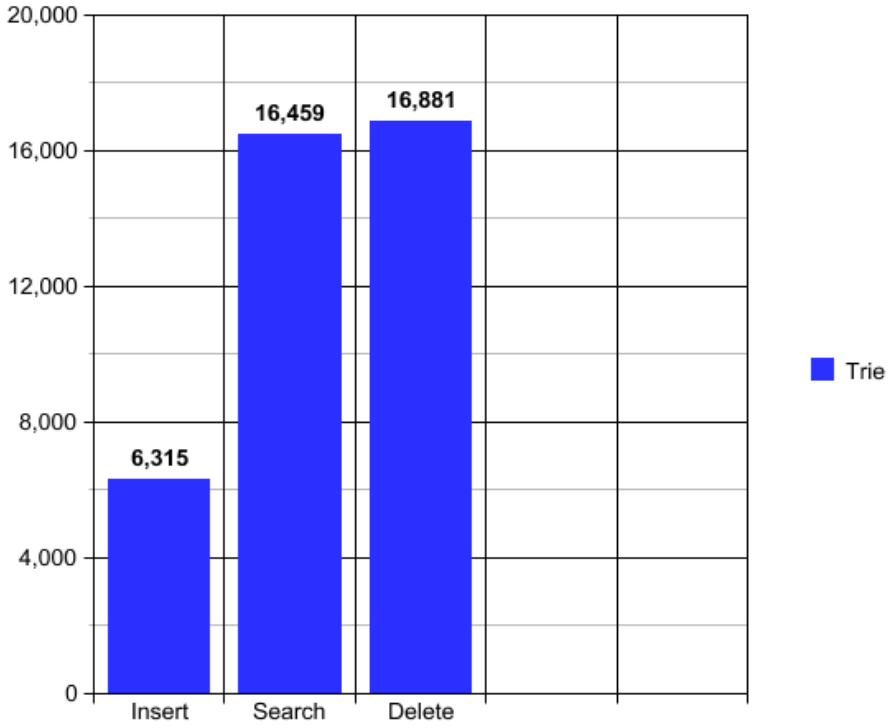
Kuva 2



Binäärihakupuun, AVL-puun ja Splay-puun perusoperaatioiden keskiarvoiset (kolmen suorituksen keskiarvo) suoritusajat millisekunteina, kun jokainen operaatio toistetaan luvuilla 0 – 19 999 999 niin, että luvut ovat suuruusjärjestyksessä.

Binäärihakupuun ajat ovat ***arvioita***, sillä ohjelma ei monen minuutin odottelunkaan jälkeen ollut saanut lisättyä lukuja lineaariseksi muodostuneeseen puuhun. Search vastaa toimintatavaltaan niin paljon insertiä, että sen suoritusnopeus liikkuisi todennäköisesti samoissa lukemissa. Delete sen sijaan olisi nopea, sillä se poistaisi joka kerta puun juuren, eikä sen siten tarvitsisi matkustaa puusta koskaan juurisolmua kauemmas.

Kuva 3



Trie-puun perusoperaatioiden keskiarvoiset (kolmen suorituksen keskiarvo) suoritusajat millisekunteina, kun jokaisen operaation toistomäärä on 2 000 000 sattumanvaraisilla syötteillä, joiden suuruus on $0\text{-}2^{32}$.