Toteutusdokumentti

Ohjelma Puuvertailu vertailee erilaisten tietorakenteiden - tässä tapauksessa puiden - tehokkuutta. Vertailtavana ovat binäärihakupuu, punamusta puu, splay-puu ja treap, joka on puun ja keon yhdistelmä. Syötteinä käytetään positiivisia kokonaislukuja, jotka toimivat samalla puun solmujen avaimina.

Ohjelman päävalikosta voi valita mitä operaatioista haluaa testata.

Ohjelman yleisrakenne

Ohjelmassa on Main-class, jossa on ohjelman käynnistävä main-metodi ja tekstipohjainen käyttöliittymä. Puu on rajapinta, jota kaikki eri puut toteuttavat. Binäärihakupuun luokka on yliluokka ja muut puut sen aliluokkia. Solmu on binäärihakupuun solmu ja muiden tarvittavien solmujen yliluokka. Vertailu-luokan avulla tehdään itse vertailut.

Aikavaativuus

Alla olevassa taulukossa on tavoiteltavat aikavaativuudet.

Tietorakenne	Pahimman tapauksen aikavaativuus
Binäärihakupuu	O(n)
Punamusta puu	O(log *n*)
Splay-puu	(tasoitettu aikavaativuus) O(log *n*)
Treap	O(*n*)

Tasoitettu aikavaativuus liittyy aina operaatiojonoon ja on oleellinen silloin kun hitain yksittäinen operaatio on hyvin hidas mutta tapahtuu vain harvoin. Kyseessä on pahin mahdollinen operaatiojono ja tarkoituksena on tasoittaa kalliiden operaatioiden viemä aika jakamalla se halvemmille operaatioille.[1]

Vaikka treapin pahimman tapauksen aikavaativuus on O(n), satunnaisuuden käyttö tekee siitä todennäköisesti korkeudeltaan logaritmisen ja odotusarvoisesti tasapainoisen.

Operaatioiden aikavaativuus riippuu puun korkeudesta. Koska tässä työssä on käytetty satunnaisia lukuja puun täyttämiseen, tulos eroaa paljon siitä, jos testejä olisi tehty lisäämällä numeroja puihin numerojärjestyksessä. Lähemmäs tällaista tulosta päästiin testissä, jossa lisättiin puuhun lukuja vain väliltä 1-1000, missä testissä binäärihakupuu näytti hitautensa, vaikka muuten pärjäsikin yllättävän hyvin muille puille. Tarkemmin miettien ymmärtää, että kyse on nimenomaan satunnaisuudesta, joka tekee binäärihakupuustakin varsin tasapainoisen.

Punamusta puu pärjäsi hyvin kaikissa testeissä. Vaikuttaa siltä, että splay-operaatio ja keon tasapainottaminen veivät paljon aikaa. Vaikka arvattavasti pienemmältä väliltä lukujen etsiminen oli

[1] https://noppa.aalto.fi/noppa/kurssi/t-106.4100/luennot/T-106_4100_kalvot_3.pdf

nopeampaa kaikille tietorakenteille kuin suuremmalta väliltä, splaypuu suoriutui yllättävän huonosti pienemmältä väliltä etsimisestä, vaikka oletin sen olevan tässä vahvoilla.

Parannettavaa

Ensinnäkin on sanottava, että ohjelmassa saavutetut ajat eivät mielestäni olleet kovin hyviä. Testauksessa käytetyt syötteetkin olisi voinut harkita tarkemmin, samoin suorituskyky- ja O-analyysivertailut jäivät puutteellisiksi (jätin dokumenttien teon liian viime tippaan). Tilavaativuudet jätin pois kokonaan. Ohjelma kuitenkin toimii.

Itse koodiin jäi joitakin pitkiä metodeja ja Vertailu-luokkaa olisi voinut tiivistää. Tosin olin tietyissä kokeiluissa huomaavani, että nopeus saattoi kärsiä, jos panosti kovasti selkeyteen ja metodien pilkkomiseen, esim. jos erillisillä metodeilla alkoi etsiä isovanhempi- ja sisarus-solmuja.

Olisi hauskaa, mutta liian suuritöistä tähän hätään tehdä käyttöliittymä, jossa voisi syöttää tarkemmin millaisilla syötteillä haluaa puita testata. Ja tietenkin itse tietorakenteilla ei tällaisenaan ole mitään käyttöä, vaan sisällöksi pitäisi voida tallentaa jotain muutakin kuin kokonaislukuja.

Lähteet

Tietorakenteet ja algoritmit -kurssin opintomoniste

T.H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein: Introduction to Algorithms. 3rd ed. MIT Press, 2009

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

http://en.wikipedia.org/wiki/Splay_tree

http://en.wikipedia.org/wiki/Treap

https://noppa.aalto.fi/noppa/kurssi/t-106.4100/luennot/T-106_4100_kalvot_3.pdf

http://opendatastructures.org/ods-java/7_2_Treap_Randomized_Binary.html