Ohjelman yleisrakenne

Työssä toteutetut tietorakenteet ovat binäärihakupuu, AVL-puu ja trie, jotka löytyvät erillisistä pakkauksista. Tietorakenteet käyttävät apunaan DataStructures paketin sisältöä. Nopeustestit paketista löytyvä Nopeustestit.java testaa tietorakenteiden yleisimpien metodien nopeuksia. Test Packages nimisestä paketista löytyy jokaiselle tietorakenteelle tehdyt testit, jotka testaavat metodien toimivuutta. Lisäksi jokaisella tietorakenteella on omassa paketissaan Main.java, jolla voidaan testata tietorakenteen toimivuutta.

Binäärihakupuuhun ja AVL-puuhun tallennetaan DataStructures paketin sisältämiä solmuja(Node) ja ne käyttävät LinkedListiä leveysjärjestyksen toteuttamisessa ja graafisessa tulostuksessa. Trie käyttää ArrayListiä lapsisolmujen tallentamiseen. Triellä on omassa paketissaan oma solmunsa Node.java.

Saavutetut aikavaativuudet

Binäärihakupuu
- Lisäys: aluksi luodaan uusi solmu ajassa O(1). Sitten etsitään sille paikka juuresta alaspäin O(h)

- Delete: tälle operaatiolle annetaan parametriksi poistettava Node, joka saadaan operaatiolla search. Delete veisi muuten aikaa O(1), mutta koska poistettava Node joudutaan etsimään puusta se vie aikaa O(h) eli searchin vaativan ajan.

- Search: käy rekursiivisesti läpi annetusta solmusta alaspäin solmuja, kunnes oikea solmu löydetään. Maksimissaan puun korkeuden h kertaa mennään alaspäin, joten aikavaativuus tälle on O(h)

AVL-puu
- Lisäys: aluksi luodaan uusi solmu ajassa O(1) ja lisätään se insert metodilla puuhun ajassa O(h). Sen jälkeen lähdetään kelaamaan lisätyn solmun parentista ylöspäin puuta ja katsotaan löytyykö epätasapainossa olevia solmuja. Jos solmu on epätasapainossa, tehdään kierrot ajassa O(1) ja jatketaan kohti juurta O(h). Mutta koska AVL-puu on ennen lisäystä aina tasapainossa, niin puun korkeus on log n, joten aikavaativuudeksi saadaan O(log n).

- Delete: samaan tapaan kuin AVL:n lisäyksessä, solmu poistetaan, epätasapainot katsotaan poistetun solmun vanhemmasta ylöspäin kunnes ollaan juuressa. Aikavaativuus on myöskin O(log n).

- Search: metodi liikkuu rekursiivisesti annetusta solmusta alaspäin kunnes annettu avain löytyy tai päädytään puun lehteen. Aikavaativuus on O(log n), koska puu on tasapainossa.

Trie
- Lisäys: metodi toimii niin, että sille parametrinä annetusta sanasta s, otetaan ensimmäinen kirjain ja katsotaan onko sillä sen nimistä lasta, ajassa O(n), jossa n on sen lasten lukumäärä. Sen jälkeen jos lasta ei löytynyt, tehdään sen niminen lapsi (ArrayList) vakioajassa O(1) ja kutsutaan sen lisäys metodia lopuilla sanan kirjaimilla. Tämä tehdään siis sanan s pituuden verran O(s), jossa s on sanan pituus. Aikavaativuudeksi muodostuu siis lasten mahdollinen lukumäärä n + sanan pituus s O(n+s)

- Delete: metodi suorittaa ensin search metodin, johon menee aikaa O(s^n^n) (katso alempi kohta search). Sen jälkeen kutsutaan poistoa rekursiivisesti kirjain kirjaimelta aloittaen sanan ensimmäisestä kirjaimesta johon kuluu aikaa sanan pituuden s verran O(s). Mutta koska search on paljon enempi aikaa vievä, se on lopullinen aikavaativuus.

- Search: haetaan ensin juuresta sanan alkukirjain ajassa O(n), jossa n on lasten lukumärä. Sen jälkeen haetaan sanan pituuden s verran aina kirjain kirjaimelta solmuja lapsisolmuista O(s) ja jokaisen kirjaimen kohdalla katsotaan jos lapsi löytyy O(n) niin asetetaan tarkasteltavaksi kirjaimeksi löytynyt lapsi O(n). Search sisältää siis sisäkkäin O(s):n, O(n):n ja O(n):n joten aikavaativuus on O(s^n^n). Mutta täytyy muistaa ettei sanoilla ole todellakaan maksimimäärää lapsia, joten realistinen käytetty aika on paljon pienempi.

	Binääripuu	AVL-puu	Trie
Insert	O(h)	O(log n)	O(1)
Delete	O(h)	O(log n)	O(s^n^n)
Search	O(h)	O(log n)	O(s^n^n)

Saavutetut tilavaativuudet

Binäärihakupuu
- Lisäys: vie tilaa vakiomäärän O(1), koska metodin viemä tilavaativuus ei riipu lisättävästä solmusta eikä puun koosta.

- Delete: vie tilaa vakiomäärän O(1), koska metodin viemä tilavaativuus ei riipu poistettavasta solmusta eikä puun koosta.

- Search: vie tilaa puun korkeudesta h riippuen määrän O(h), koska joka kerta kun solmusta liikutaan alaspäin, kutsutaan search operaatiota rekursiivisesti.

AVL-puu
- Lisäys: vie tilaa vakion O(1), koska se ei sisällä rekursiota eikä tilavaativuus riipu lisättävästä solmusta eikä puun koosta.

- Delete: sama tilavaativuus kuin lisäyksellä eli O(1).

- Search: vie tilaa puun korkeudesta h riippuen määrän O(h), koska joka kerta kun solmusta liikutaan alaspäin, kutsutaan search operaatiota rekursiivisesti. Mutta koska AVL-puu on tasapainossa niin puun korkeus on log n, joten search vie aikaa O(log n).

Trie
- Lisäys: vie eri määrän tilaa riippuen siitä, kuinka samankaltaisia tallennetut sanat ovat. Maksimissaan siis määrän O(s), jossa s on kaikkien sanojen pituudet laskettuna yhteen. Mitä enempi eri merkkejä on käytössä ja sanat vaihtelevat, sitä enempi se vie tilaa kun uusia lapsia(ArrayList) kirjaimille joudutaan luomaan.

- Delete: sama kuin lisäys eli O(s), jossa s on kaikkien sanojen pituudet laskettuna yhteen (pahin tapaus, jos sanoilla ei ole yhtään samankaltaisuutta)

- Search: sama kuin lisäys eli O(s), jossa s on kaikkien sanojen pituudet laskettuna yhteen (pahin tapaus, jos sanoilla ei ole yhtään samankaltaisuutta)

	Binääripuu	AVL-puu	Trie
Insert	O(1)	O(1)	O(s)
Delete	O(1)	O(1)	O(s)
Search	O(h)	O(log n)	O(s)

Työn mahdolliset puutteet ja parannusehdotukset

Muuttujien nimissä olisi voinut käyttää vain pelkästään suomea tai englantia, olisi ehkä ollut selkeämpää niin.

Trien tavoitteet aikavaativuuksissa oli O(1), jokaisella operaatiolla ja tilavaativuus sai olla O(lasten lukumäärä). Olisin voinut toteuttaa trien niin, että jokaiselle Nodelle olisi luonut laittanut lapsiksi kaikki haluttavat tietorakenteen hyväksyvät kirjaimet. Tein siis trien huomattavasti pienemmillä tilavaativuuksilla mutta suuremmilla aikavaativuuksilla. Toteuttamani trie näyttää hitaammalta ja enemmän tilaa vievältä kuin se todellisuudessa on.

Lähteet

Binärihakupuu ja AVL-puu:
http://www.cs.helsinki.fi/u/floreen/tira2012/tira.pdf

Trie:
http://www.cs.helsinki.fi/u/ejunttil/opetus/tiraharjoitus/trie.html

http://grepcode.com/file/repository.grepcode.com/java/root/jdk/openjdk/6-b14/java/util/ArrayList.java