TiLa II
Analyzing words in a text file

Anni Järvenpää
014338836

20. joulukuuta 2015

1 Johdanto

Tavoitteena on laskea tiedostossa olevien sanojen toistokertojen määrä. Tällaisia niinkutsuttuja frekvenssilistoja käytetään paljon kielitieteessä ja erityisesti korpuslingvistiikassa. Niistä on hyötyä muun muassa tekstiä koskevien hypoteesien muodostamisessa sekä tehtyjen oletusten tarkastamisessa. [2]

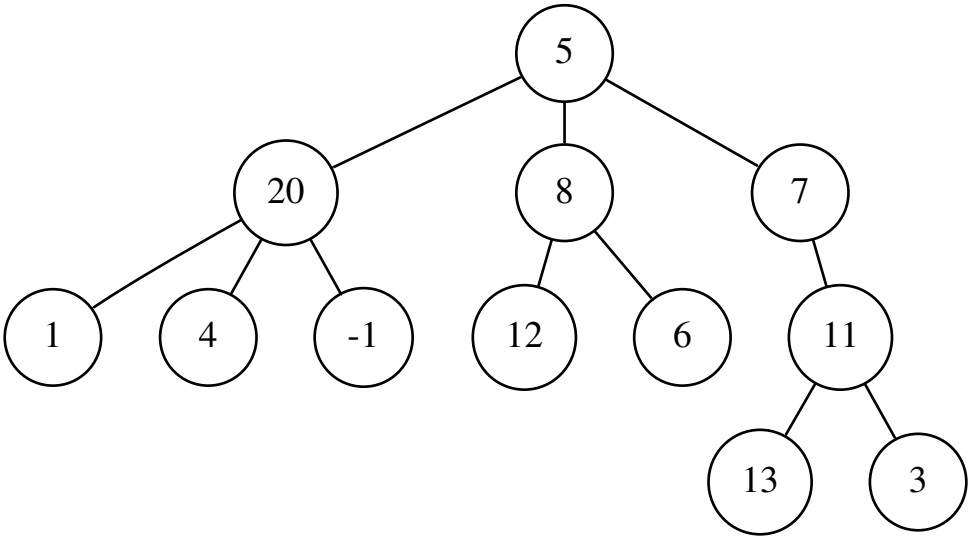
Lisäksi eri teksteistä saatuja frekvenssilistoja voidaan vertailla toisiinsa ja tutkia näin tekstien eroja. Frekvenssilistoihin voidaan myös käyttää erilaisia tilastollisia analyysimenetelmiä. Frekvenssilistojen on muun muassa havaittu usein noudattavan Zipfin lakia, jonka mukaan sanan esiintymismäärä korpuksessa on kääntäen verrannollinen sen järjestyslukuun esiintymismäärän mukaan järjestetyssä frekvenssilistassa. [2, 3]

2 Menetelmät

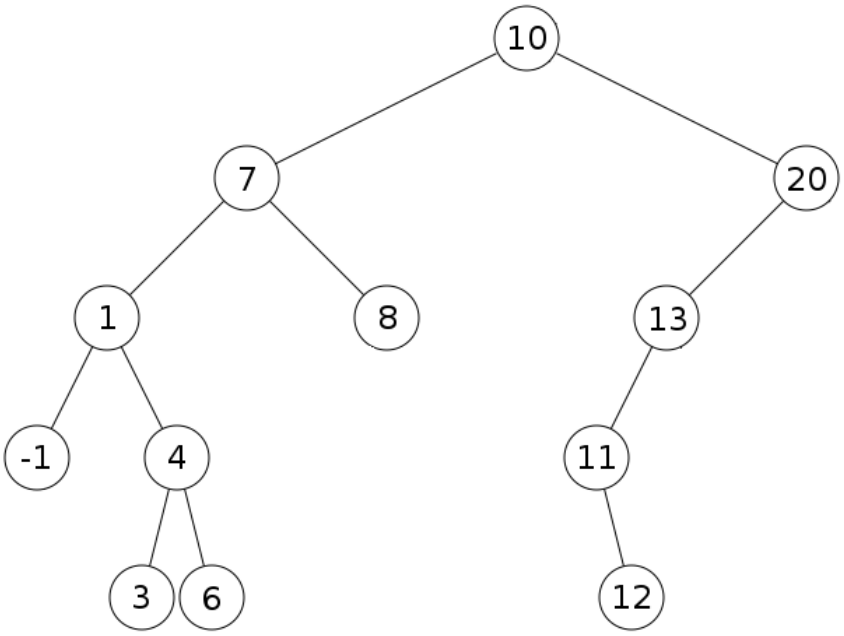
Sanojen laskemisessa hyödynnetään binäärihakupuuta. Puu on solmuista koostuva tietorakenne, jossa jokaiseen solmuun voidaan tallentaa tietoa ja jokaisella solmulla on 1 tai 0 vanhempaa sekä n lasta missä $n \in \mathbb{N}_0$. Puu voidaan esittää suunnattuina yhtenäisinä verkkoina, joissa jokaisesta solmusta on kaari jokaiseen lapseensa. [1]

Puussa on aina tasan yksi solmu, jolla ei ole vanhempaa ja tätä solmua kutsutaan juureksi. Solmuja, joihin ei tule kaarta mistään toisesta solmusta kutsutaan lehdiksi. Solmun korkeus on kaarien määrä pisimmällä polulla solmusta lehteen. Usein puhutaan myös puun korkeudesta, jolla tarkoitetaan puun juurisolmun korkeutta. Esimerkiksi kuvassa 1 on esitettynä puu, jonka solmuihin on tallennettu kokonaislukuja. Puun juurisolmun arvo on 5 ja lehtisolmuissa on arvot 1, 4, -1, 12, 6, 13 ja 3 sekä korkeus 3 (kaaret $5 \rightarrow 7$, $7 \rightarrow 11$ ja $11 \rightarrow 13$ tai $11 \rightarrow 3$). [1]

Binäärihakupuussa jokaisella solmulla on 0, 1 tai 2 lasta ja kunkin solmun vasemmasta lapsesta lähtevässä alipuussa on vain arvoltaan solmun arvoa pienempiä arvoja ja oikeassa lapsessa lähtevässä alipuussa vain solmun arvoa suurempia arvoja. Näin etsittäessä tiettyä solmua, voidaan kunkin solmun kohdalla sulkea pois toinen solmun lapsista, jolloin joudutaan tutkimaan korkeintaan puun korkeuden verran solmuja. Eräs binäärihakupuu on esitetty kuvassa 2. [1]



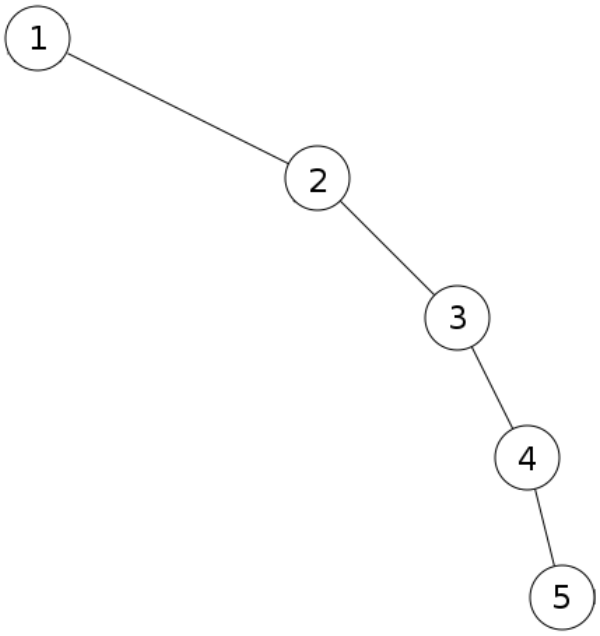
Kuva 1: Esimerkki puusta, johon on tallennettu kokonaislukuja.



Kuva 2: Kuvan 1 solmut järjestettynä eräällä mahdollisella tavalla binäärihakupuuhun.

Binäärihakupuussa solmun etsimisen aikavaativuus on $\mathcal{O}(h)$, missä h on puun korkeus. Esimerkiksi kuvassa 3 esitetyssä binäärihakupuussa solmuun 5 kulkeminen edellyttää kaikkien solmujen läpikäyntiä, eli tässä tapauksessa binääripuu ei ole linkitettyä listaa parempi tietorakenne. Mikäli tälläistä tasapainottamatonta binääripuuta käyttäisi tekstiin, jonka sanat ovat aakkosjärjestyksessä (esimerkiksi sanakirjan hakemisto), on puuhun lisäämisen ja sieltä hakemisen aikavaativuus $\mathcal{O}(n)$ missä n on puun solmujen määrä. [1]

Siksi binäärihakupuu kannattaakin pyrkiä pitämään jollain tavalla tasapainossa siten,



Kuva 3: Hyvin epätasapainoinen binäärihakupuu.

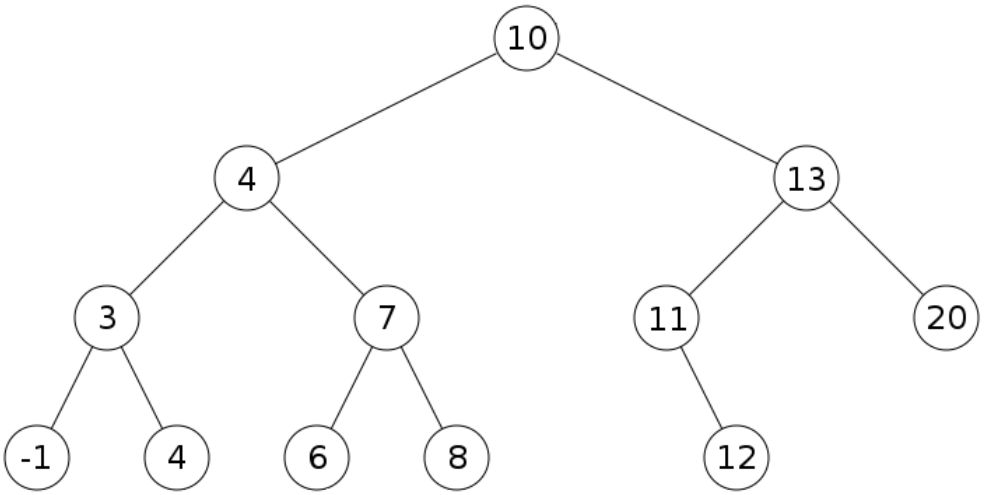
että puun tasapainottamiseen kuluva aika on pienempi kuin siitä saatava hyöty muiden operaatioiden aikavaativuuksissa. Vaihtoehtoisia toteutuksia on useita, muun muassa splay-puut, punamustat puut sekä AVL-puut. Tässä työssä käytetään AVL-puuta. [1]

AVL-puun katsotaan olevan tasapainossa jos minkään solmun oikean ja vasemman lapsen korkeuksien ero ei ole suurempi kuin yksi. Mikäli solmun lisääminen tai poistaminen johtaa epätasapainon syntymiseen, se korjataan erilaisilla kierroilla. Kuvassa 4 on nähtävillä eräs mahdollinen tapa järjestää kuvan 2 solmut siten, että puu täyttää AVL-puun tasapainoehdon. [1]

3 Toteutus

Sanalaskuria varten toteutin AVL-puun ja tiedostonluvun omina moduuleinaan sekä näitä käyttävän pääohjelman. AVL-puu koostuu solmuista, joista kukin tallentaa korkeintaan 50 merkkiä pitän sanan sekä sanojen esiintymismäärän. Lisäksi solmuun on tallennettu pointteri sen vanhempaan sekä oikeaan ja vasempaan lapseen. Mikäli lasta tai vanhempaa ei ole, on pointteri NULL.

Lisäksi toteutin funktiot tietyn sanan sisältävän solmun etsimiseksi puusta, solmun korkeuden laskemiseksi sekä solmujen lisäämiseksi puuhun. Näitä varten tarvitsin myös puun tasapainottavan funktion sekä sen käyttämät kierrot. Sanalaskurissa solmujen poistaminen on tarpeetonta, joten en toteuttanut sitä puuhun.



Kuva 4: AVL-puu

Tulosten näyttämiseksi solmulla on myös aliohjelma, joka tulostaa solmuun tallennetun sanan sekä sen lukumäärät. Koko puun tulostamiseksi kirjoitin lisäksi aliohjelman, joka kutsuu edellistä aliohjelmaa kaikille solmusta lähtevän alipuun solmuille, jolloin kaikki puun sanat saadaan tulostettua. Kun solmut tulostetaan sisäjärjestyksessä (ensin vasen lapsi rekursiivisesti, sitten solmu itse, viimeiseksi oikea lapsi rekursiivisesti), tulostuvat solmut aakkosjärjestyksessä.

Tiedostonlukumoduuli lukee avattua tiedostoa rivi (maksimipituus 10 000 merkkiä) kerrallaan ja palauttaa aina funktiota `lue_sana()` kutsuttaessa seuraavan sanan tiedostossa. Mikäli rivi loppuu, katsotaan myös sanan loppuvan ja lukeminen aloitetaan seuraavan sanan kohdalla seuraavalta riviltä. Sanoiksi katsotaan yhtenäiset korkeintaan 50 merkin mittaiset merkkijonot, jotka voivat sisältää isoja tai pieniä kirjaimia a-z, numeroita 0-9 ja väliviivoja (-). Kaikkien muiden merkkien katsotaan olevan sanaerottimia eikä niitä käsitellä mitenkään. Ennen sanan palauttamista sen isot kirjaimet muutetaan pieniksi. Mikäli tiedosto on tyhjä tai luettu loppuun, `lue_sana()` palauttaa tyhjän merkkijonon.

Luettavan tiedoston voi asettaa tai sitä voi vaihtaa `avaa(pituus, tiedostonimi)`-funktiolla, joka sulkee vanhan tiedoston mikäli sellainen on auki ja avaa sen jälkeen uuden tiedoston käyttäen polkuna annettua tiedostonimeä. Funktio palauttaa true mikäli tiedoston avaaminen onnistuu ja false mikäli ei onnistu. Jälkimmäisessä tapauksessa ohjelma tulostaa myös virheviestin, joka kertoo, mikä iostatin arvo oli. Samalla funktiolla voi myös siirtää lukukohdan takaisin tiedoston alkuun avaamalla jo auki olevan tiedoston uudelleen.

Pääohjelma sanalaskuri ottaa komentoriviargumenttina polun tutkittavaan tiedostoon ja yrittää avata kyseisen tiedoston. Mikäli argumentteja ei ole, ohjelma tulostaa virheviestin ja suoritus lopetataan. Myös tiedoston avaamisen epäonnistuessa ohjelman suoritus pysäytetään. Onnistuneen tiedoston avaamisen jälkeen luetaan tiedoston ensimmäinen sana AVL-puun juureksi. Mikäli tämä sana on tyhjä, tiedetään tiedoston olevan tyhjä. Jos taas sana ei ole tyhjä, luetaan uusia sanoja ja lisätään niitä puuhun, kunnes `lue_sana()`-funktio palauttaa tyhjän sanan, minkä jälkeen kutsutaan AVL-puun aliohjelmaa, joka tulostaa puun.

Ohjelman kääntämiseksi kirjoitin makefilen, jolloin ohjelma voidaan kääntää yksinkertaisesti komennolla `make` ja tämän jälkeen ajaa muotoa `./bin/sanalaskuri path/to/file.txt` olevalla komennolla, missä `path/to/file.txt` on korvattu halutulla tiedostolla.

4 Tulokset

5 Johtopäätökset

Viitteet

[1] Thomas H. Cormen. *Introduction to algorithms*. MIT Press, Cambridge (MA), 2001.

[2] Hanna Tuomisto. Xterm-korpuskyselykielen kehittäminen ja korpuskyselykielten vertailu. `https://tampub.uta.fi/bitstream/handle/10024/83713/gradu06022.pdf?sequence=1`. Luettu 20.12.2015.

[3] Wikipedia. Zipf's law — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/wiki/Zipf's_law`, 2015. Luettu 20.12.2015.