TOTEUTUSDOKUMENTTI

Aihe: Sanaindeksoija

Sovellus lukee käyttäjän antamista tekstitiedostoista sanat hakupuurakenteeseen, jonka jälkeen käyttäjän on mahdollista kysellä sovellukselta sanojen tai niiden yhdistelmien (haluamiensa merkkijonojen) esiintymisestä kyseisissä tekstitiedostoissa.

Ohjelman yleisrakenne

Sovellus toteutettiin neljää luokkaa käyttäen

- Sanaindeksoija
- Hakupuu
- Node
- Lista

Sanaindeksoija sisältää sovelluksen main-metodin. Ensimmäisen sisäänluku-tiedoston löytämisen jälkeen Sanaindeksoija luo uuden Hakupuu-olion. Sisäänluettevan tiedoston tulee olla ascii-muotoinen tekstitiedosto. Kunkin sisään-luettavan tiedoston kohdalla toteutetaan seuraava alimetodi:

- **public static boolean lueSisään(String)** tarkastaa annetun tekstitiedoston löytymisen ja käynnistää Hakupuu-luokan metodin lueTdsto(tiedosto, tiedos-ton järjestysnro). Metodi saa parametrina käsiteltävän tiedoston nimen ja palauttaa boolean arvon, joka kertoo sisäänluvun onnistumisesta.

Jos sisäänluku on onnistunut, tallentaa main-metodi tiedoston nimen String-taulukkoon ja tulostaa käyttäjän niin valitessa hakupuun standardisyöttövirtaan. Tulostus käyttää Hakupuu-luokan toString()-metodia. Kun sisäänlukutiedostoja ei enää ole tai sovelluksen käynnistyessä annettu sisäänluettavien tiedostojen lukumäärä täyttyy, siirtyy main-metodi toteuttamaan hakutoimintoa.

Haku kohdistuu ensin koko Hakupuu-luokan hakupuuhun sisäänluettuun teksti-massaan, jonka jälkeen saatua hakutulosta voi tarkentaa tekemällä siihen alahakuja. Alahaku tekee haun myös koko materiaaliin, jonka jälkeen saatua hakutulosta verrataan aiempaan hakutulokseen yhteisten osumien löytymiseksi. Kukin haku käyttää seuraavia alimetodeja:

- **private static int teeHaku(String, boolean)** käynnistää Hakupuu-luokan metodin etsiJono(merkkijono). Metodi saa parametreina haettavan merkki-jonon ja tiedon siitä, onko kyse varsinaisesta vai alahausta. Jos metodille palautuu Lista-luokan olio, on käyttäjän antama merkkijono löytynyt hakupuusta. Jos kyseessä on alahaku, toteutetaan aiemman hakutuloksen ja nyt saadun hakutuloksen välillä vertailu metodilla teeVertailu(alahaun hakutulos), jonka palautuksena saadaan näiden tulosten leikkaus. Haun hakutulos tai alahaun lopputulos sekä osumien lukumäärä tulostetaan

käyttäjälle standardisyöttövirtaan. Tulostuksessa käytetään Lista-luokan metodeja getTdsto(), getTdstonrivi() ja getHakurivi() sekä Hakupuu-luokan metodia getRivi(sisäänlukurivin järjestysnumero). Metodi palauttaa int-arvon, joka ilmoittaa haussa löytyneiden osumien lukumäärän.

- **private static Lista teeVertailu(Lista)** saa parametrina tehdyn alahaun hakutuloksena tulleen linkitetyn listan ensimmäisen olion. Metodi käy synkronoidusti läpi aiemmin tehdyn haun lopputuloksen ja parametrina saadun linkitetyn listan ja tuottaa niiden pohjalta kolmannen linkitetyn listan, johon talletetaan syöttötietona käytettyjen listojen leikkaus. Vertailussa käytetään Lista-luokan metodeja getTdsto(), getTdstonrivi(), getHakurivi(), getSeurLista() ja setSeur(new Lista). Metodi palauttaa tuloksena saadun linkitetyn listan ensimmäisenä olevan Lista-olion.

Kun hakutoiminto päättyy käyttäjän annettua tyhjän merkkijonon, tuottaa main-metodi käyttäjälle tilaston tehdyistä hauista. Tässä käytetään seuraavaa metodia:

- **private static void tulostaTilasto()** tulostaa standardisyöttövirtaan tiedon sisäänluettujen tiedostojen lukumäärästä, sisäänluettujen tekstirivien lukumäärästä ja keskiarvosta per tiedostojen lukumäärä, varsinaisten (koko massaan kohdistuneiden) hakujen lukumäärästä, alahakujen lukumäärästä ja keskiarvosta per varsinaiset haut sekä saaduista osumista (lasketaan lopullisista löydettyjen rivien lukumäärästä tehtyjen alahakujen jälkeen) ja keskiarvosta per varsinaiset haut.

Sovelluksen toiminta päättyy tilaston tulostuksen jälkeen. Sanaindeksoija toteuttaa seuraavat tietorakenteet:

- String-taulukko tiedostojen nimille. Taulukon koko määräytyy käyttäjän sovellukselle antaman tiedostolukumäärän mukaan.

Hakupuu on luokka, joka toteuttaa hakupuuna käytetyn Trie-puun. Hakupuun attribuutit ovat Node-luokkaan kuuluva puun juurisolmu, String-taulukko hakupuuhun sisään luettujen rivien tallennusta varten ja sekä int-arvo, joka kertoo seuraavan vapaan solun String-taulukossa. Konstruktori luo uuden Node-luokan olion, jonka avaimena on tyhjä merkki. Hakupuun metodeja kutsutaan joko Sanaindeksoija-luokasta tai ne ovat sisäisiä metodeja. Hakupuu-luokassa on seuraavat metodit:

- **public int lueTdsto(File, int)** saa parametreina sisäänlukutiedoston File-olion sekä sen järjestysnumeron. Metodi lukee sisäänlukutiedoston läpi ja käsittelee kunkin rivin metodeilla tallennaRivi(rivi) ja kasitteleRivi(rivi, tiedostonnro, tiedostonrivin järjestysnro, tallennusrivin järjestysnumero). Metodi palauttaa int-arvon, joka ilmoittaa kyseisestä tiedostosta sisäänluettujen rivien lukumäärän.
- **private void tallennaRivi(String)** saa parametrina sisäänlukutiedostosta luetun rivin. Jos tallennustaulukko on täynnä, kasvattaa metodi sitä dynaamisesti kaksinkertaistamalla sen koon. Parametrina saatu rivi tallennetaan taulukon ensimmäiseen vapaaseen paikkaan.
- **public void kasitteleRivi(String, int, int, int)** saa parametreina sisäänluku-

tiedostosta luetun rivin, tiedoston järjestysnumeron, rivin järjestysnumeron tiedoston sisällä sekä tallennustaulukon solun indeksin, johon rivi tallennaRivi(rivi)-metodilla tallennettiin. Rivi käydään läpi merkki merkiltä etsien seuraavan sanan aloittavaa tallennuskelpoista merkkiä. Sana voi alkaa rivin alusta tai tyhjän merkin jälkeen. Tallennuskelpoisuus tarkistetaan metodilla selvitaKelpo(merkki). Sananalun löydyttyä tallennetaan kaikki loppurivin kelvolliset merkit hakupuuhun. Kunkin merkin lapset eli sitä välittömästi seuraavat merkit tallennetaan maksimitaulukkoon, jonka oikea solu (jokaisella kelvollisella merkillä oma sijoituspaikka taulukossa) löytyy metodin selvitaInd(merkin int-arvo) avulla. Lapsisolmun toteutuksessa käytetään Node-luokan metodeja getLapsi(indeksi), setLapsi(indeksi) ja setRivi(tiedostonro, tiedostonrivinro, hakurivinro). Kun löydetyn sananalun jälkeiset merkit on käsitelty, palaa metodi etsimään samalta riviltä seuraavaa sananalkua ja tallentamaan sen jälkeisen tekstin hakupuuhun. Tätä tehdään rivin loppuun asti.

- **private int selvitaKelpo(char)** saa parametrina riviltä luetun merkin. Jos merkki ei ole kelvollinen eli ei kuulu suomalaiseen aakkostoon tai numeroihin tai ole tyhjä merkki, palauttaa metodi kutsuneelle metodille negatiivisen kokonaisluvun.
- **private int selvitaInd(int)** saa parametrina kelvollisuustarkastuksen läpäisseen merkin int-arvon. Metodi palauttaa int-arvoa vastaavan indeksin, joka osoittaa merkin mukaisen lapsisolmun tallennussolun.
- **public Lista etsiJono(String)** saa parametrina käyttäjän antaman haettavan merkkijonon. Haku käynnistyy hakupuun juurisolmusta. Merkkijonon kunkin merkin kelpoisuus tarkastetaan selvitaKelpo(merkki)-metodin avulla, ja mikäli se on kelvollinen, selvitetään sen oikea indeksiarvo selvitaInd(merkin int-arvo) avulla. Merkkijonon etsimisessä käytetään Node-luokan metodia getLapsi(indeksi), jonka palauttaessa null-arvon tiedetään, ettei haettua merkkijonoa löydy hakupuusta. Muussa tapauksessa hakua jatketaan seuraavasta merkkijonon merkistä eteenpäin. Jos merkkijono saadaan käsiteltyä kokonaan läpi, haetaan viimeiseltä löydetyltä Node-oliolta sen linkitetyn listan ensimmäinen Lista-olio Node-luokan metodilla getLista() ja palautetaan se kutsuneelle metodille.
- **public String getRivi(int)** palauttaa parametrina saadun indeksin mukaisen tekstirivin sisäänluetut tiedostonrivit sisältävästä String-taulukosta.
- **public Node getJuuri()** palauttaa Trie-hakupuun juurisolmun.
- **public String toString()** palauttaa Trie-hakupuun sisällön merkkijonona kutsuneelle metodille. Metodi käyttää tässä toStringHelp(solmu, sisennys)-apumetodia.
- **private String toStringHelp(Node, String)** on rekursiivinen metodi, joka tuottaa hakupuun sisällön tulostettavaan muotoon.

Hakupuu-luokalla on seuraavia rajoituksia puuhun tallennettavien tietojen ja niiden löytymisen suhteen:

- Tallennettavat merkkijonot aloitetaan aina sanojen alusta, joten haku ei löydä merkkijonoja sanan keskeltä.
- Tekstitiedostoja käsitellään rivi kerrallaan, joten haettavan merkkijonon tulee sijaita yhdellä rivillä, jotta se löytyy.

- Tallennettava sana alkaa aina rivin alusta tai tyhjän merkin jälkeen.
- Sana päättyy aina rivin loppuun tai tyhjään merkkiin.
- Puuhun tallennetaan vain suomalaisen aakkoston kirjaimia ja numeroita sekä rivillä sanojen välissä olevia tyhjiä merkkejä, joten kelvollisia merkkejä on 40 kpl.

Hakupuu toteuttaa seuraavat tietorakenteet:

- String-taulukko sisäänluetuille tiedostonriveille. Taulukon aloituskoko on 1000 ja sitä kasvatetaan dynaamisesti kaksinkertaistamalla koko silloin, kun vapaa-muuttuja osoittaa sen tulleen täyteen.
- Trie-puu, johon luetaan sisäänlukutiedostojen sisältö rivi ja merkki kerrallaan. Trie-puu on toteutettu Node-luokan olioilla.

Node-luokkaa käytetään toteuttamaan Trie-puun tarvitsemat solmut. Noden attribuutit ovat char-muotoinen merkki, joka toimii solmun avaimena, Lista-luokan ekaEsitys, johon tallennetaan kyseiseen solmuun päättyvien tekstiriviesiintymien linkitetyn listan ensimmäinen listasolmu, Lista-luokan vikaEsitys, johon tallennetaan saman linkitetyn listan viimeisin tallennettu listasolmu, ja Node-taulukko lapset, johon tallennetaan kyseistä solmua seuraavien merkkien Node-muotoiset solmut. Konstruktori luo uuden Node-taulukon. Noden metodeja kutsutaan Hakupuu-luokasta tai ne ovat sisäisiä metodeja. Node-luokassa on seuraavat metodit:

- **public Node getLapsi(int)** palauttaa kutsuneelle metodille parametrina saadun indeksin mukaisen lapset-taulukon Node-olion.
- **public void setLapsi(int, Node)** tallentaa parametrina saadun Node-olion parametrina saadun indeksin osoittamaan lapset-taulukon soluun.
- **public void setRivi(int, int, int)** luo uuden Lista-olion, jonka tallentaa käsittelyssä olevan Node-olion tekstiriviesiintymät sisältävään linkitettyyn listaan. Jos linkitetty lista on jo olemassa, käyttää metodi tässä hyväkseen Lista-luokan metodia setSeur(uusi olio).
- **public char getMerkki()** palauttaa käsiteltävän Node-olion avaimen eli merkki-attribuutin arvon.
- **public Lista getLista()** palauttaa käsiteltävän Node-olion sisältämän linkitetyn listan ensimmäisen Lista-olion.

Node toteuttaa seuraavat tietorakenteet:

- Node-taulukko solmun lapsisolmujen tallentamista varten. Taulukon koko on 40 ja se on ns. maksimitaulukko eli jokaisella siihen tallennettavalla tiedolla on oma paikkansa.
- Yhteen suuntaan linkitetty lista, johon tallennetaan tieto niistä sisäänlukutiedostojen riveistä, joilla käsittelyssä oleva Node-olio esiintyy. Rivitiedot tallentuvat linkitettyyn listaan järjestyksessä, joten lista on myös järjestetty lista. Linkitetty lista on toteutettu Lista-luokan olioilla.

Lista-luokka toteuttaa solmut linkitettyyn listaan, johon tallennetaan kunkin Node-olion esiintymät sisäänluetuilla riveillä. Listan attribuutit ovat int-arvo tdsto, joka kertoo sisäänlukutiedoston, josta esiintymä on peräisin, int-arvo tdstonrivi, joka

kertoo esiintymän rivinumeron sisäänlukutiedostossa ja int-arvo hakurivi, joka ilmoittaa esiintymärivin indeksin Hakupuu-luokan String-taulukossa, sekä Lista-olio seurLista, joka viittaa linkitetyn listan järjestyksessä seuraavaan solmuun. Listan metodeja kutsutaan pääsääntöisesti Node-luokasta, mutta myös Sanaindeksoija-luokka käyttää niitä hakutuloksen tulostamisessa. Lista-luokassa on seuraavat metodit:

- **public void setSeur(Lista)** tallentaa käsiteltävän solmun seurLista-attribuuttiin viittauksen sitä seuraavaan Lista-olioon.
- **public int getTdsto()** palauttaa solmun esiintymän tiedoston järjestysnumeron.
- **public int getTdstonrivi()** palauttaa solmun esiintymän tiedoston rivinumeron.
- **public int getHakurivi()** palauttaa solmun esiintymän sijainnin Hakupuu-olion String-taulukossa.
- **public Lista getSeurLista()** palauttaa viittauksen solmua seuraavaan linkitetyn listan solmuun.

Saavutetut aika- ja tilavaativuudet

Node- ja Lista-luokkien sisältämien metodien aika- ja tilavaativuus on kaikissa tapauksissa O(1), joten niitä ei alla olevassa tekstissä käydä läpi. Samoin tilaston tulostamisen aika- ja tilavaativuus on O(1).

Yhden tekstitiedoston sisäänluvun aikavaativuus muodostui seuraavanlaiseksi:

- Sanaindeksoija.main-metodin runko O(1), koska komennot ovat vakioaikaisia
- Sanaindeksoija.lueSisaan()-metodin runko O(1), koska komennot ovat vakioaikaisia
- Hakupuu.lueTdsto()-metodin runko O(1) ja rungon sisältämä toistolohko O(n), jossa n = rivien lukumäärä: jokainen rivi luetaan kerran
- Hakupuu.tallennaRivi()-metodi O(1), kun dynaamista taulukkoa ei tarvitse kasvattaa, ja O(n), kun sitä kasvatetaan
- Hakupuu.kasitteleRivi()-metodi sisältää kaksi sisäkkäistä for-lohkoa, joissa molemmissa rivin merkit luetaan löydetystä kohdasta loppuun asti, joten sen aikavaativuus on $O(n^2)$ -> koska metodi käynnistetään O(n)-aikavaativuutta olevan metodin toistolohkon sisältä, on näiden metodien yhteenlaskettu aikavaativuus $O(n^3)$; for-lohkojen sisällä olevat käskyt ovat aikavaativuudeltaan O(1)

⇨ Tekstitiedoston sisäänluvun aikavaativuus on pahimmillaan $O(n^3)$ eli tekstirivien lukumäärä * rivin merkkien lukumäärä^2. Kunkin solmun kohdalla sen lapsen lisäämisen aikavaativuus on O(1).

⇨ Tekstitiedoston sisäänluvun tilavaativuus on O(1). Tämän voi perustella sillä, että kunkin Node-solmun ja Lista-solmun tilavaativuus on O(1) kuten myös Hakupuu-olion, koska sisäänluetut tekstirivit sisältävä taulukko on attribuutti, ei kertautuva aputaulukko. Rekursiota ei tallennuksessa käytetä, joten kerrallaan muistissa on vain yksi olio per luokka.

Yhden haun (varsinainen haku tai alahaku) aikavaativuus muodostui seuraavanlaiseksi:

- Sanaindeksoija.main-metodin runko O(1), koska tarkastellaan yhtä yksittäistä hakua, on se sitten varsinainen tai alahaku
- Sanaindeksoija.teeHaku()-metodin runko O(1), koska komennot ovat vakioaikaisia, ja rungon sisältämä hakutuloksen tulostuslohko O(n), jossa n = hakutuloksen rivien lukumäärä
- Hakupuu.etsiJono()-metodin for-lohkon aikavaativuus on O(n), jossa n = haettavan merkkijonon merkkien lukumäärä
- Sanaindeksoija.teeVertailu()-metodi sisältää toistolohkon, jonka aikavaativuus on O(n), jossa n = rivien lukumäärä pienemmässä vertailussa olevassa hakutuloksessa

⇨ Merkkijonon haun aikavaativuus on O(n) eli haettavan merkkijonon merkkien lukumäärä

⇨ Maksimipituisen merkkijonon haun aikavaativuus on myös O(n), jossa n = hakupuun korkeus. Kunkin solmun kohdalla sen lapsen haun aikavaativuus on O(1).

⇨ Merkkijonon haun tilavaativuus on O(lapsisolmujen lukumäärä) = O(1), koska lapsisolmujen lukumäärä on vakio.

Puutteet ja parannusehdotukset

Sovellus toteuttaa ne toiminnallisuudet, jotka sen on tarve toteuttaa. Kehitettävää löytyy seuraavissa alueissa:

- hakupuuhun hyväksyttävien merkkien laajentaminen käsittämään myös muita aakkostoja kuin suomalaiset aakkoset
- Trie-puun lapsisolmujen tallennus merkkien hyväksymistä rajoittavan maksimitaulukon sijaan kekoratkaisuna, jolloin tilan käyttö järkevöityisi ylimääräisen tilavarauksen puuttuessa (kekoratkaisussa käytettävän taulukon aloituskokoa voisi muuttaa sen mukaan, kuinka syvällä Trie-puussa ollaan) ja ratkaisu mahdollistaisi rajoituksettoman määrän Trie-puuhun tallennettavaksi hyväksyttäviä merkkejä (nyt tehdyn kelpoisuustarkistuksen voisi muuttaa hylkäystarkistukseksi, jolla poistetaan käytöstä tunnetut erikoismerkit)
- tekstitiedostojen sisäänluvun toteuttaminen puskuroimalla, jonka ansiosta käyttäjä pääsisi etenemään nopeammin sisäänluvun toimiessa taustalla.

Liitteet

- Sanaindeksoija luokkakaavio
- Sanaindeksoija sekvenssikaavio

Lähteet

- kurssisivulla annetut materiaalit
- Tira-kurssin (kevät 2012) materiaalit
- Java 7 API Specification