Toteutusdokumentaatio:

Harjoitustyö: Tietorakenteet ja algoritmit

Ohjelman yleisrakenne

Ohjelma koostuu kolmesta pääkomponentista:

- -käyttöliittymä
- -pakkaaja
- -purkaja

Käyttöliittymä on toteutettu sekä tekstipohjaisena, että graafisena. Graafinen käyttöliittymä ei dokumentin kirjoitushetkellä toimi odotetusti tiedostojen tallentamisen ja operaatioiden keskeytysten osalta.

Tekstipohjaista käyttöliittymää käytettäessä pakkaaja tai purkaja ei ole keskeytettävissä ohjelmallisesti ollenkaan. Pakkaajana käytetään suoraan MultiByteEncoder-luokan ilmentymää, samoin kuin purkajana käytetään suoraan MultiByteDecoder-luokan ilmentymää. Ohjelmaa käytettäessä graafisen käyttöliittymän kautta pakkaaja ja purkaja kapseloidaan erillisiin Compression- ja Decompression -luokkien ilmentymiin. Kumpaa tahansa käyttöliittymää käytetäänkin, on pakkaajan ja purkajan ilmentymät kapseloitu omaan säikeeseensä joka mahdollistaa useampien tiedostojen käsittelyn samanaikaisesti sekä niiden tilojen kyselyn suorituksen aikana.

Tarkempi kuvaus tärkeistä luokista

MultiByteHashedTable:

Hajautustaulu jota pakkaaja käyttää monitavutiedon tallentamiseen, analysointiin ja hakuun. Hajautustaulu on toteutettu kaksiulotteisena taulukkona. Sarakkeiden määrä saattaa kasvaa tiedon lisäämisen yhteydessä, kasvu on aina kaksinkertainen. Rivien määrä on aina 16. Tiedon tallennus tapahtuu seuraavasti:

-Lasketaan monitavun hajautusavaimesta (*ha*) taulukon pituuteen (*l*) ja kierrokseen (c*)* perustuva hajautusavain (kierros merkitsee monennettako kertaa avainta hajautetaan avoimessa hajautuksessa, alkaa nollasta (0)):

$hk = (((3 * ha + 13) \bmod (l + 7)) \bmod l) * (h'k * c)$

h'k = ((5 * ha + 13) mod (l + 37)) mod l

-Jos taulukon kohdassa, johon hajautusavain *hk* osoittaa, ei ole tilaa, katsotaan onko taulukon seuraavalla rivillä tilaa (enintään 16 paikkaa). Jos sarakkeessa ei ole tilaa, haetaan uusi avain kasvattaen kierrosta kunnes taulukosta löytyy avaimen perusteella paikka tiedolle.

-Jos *hk* osoittaa taulukon kohtaan jossa on saman tiedon omaava monitavu kasvatetaan ko monitavun esiintymisfrekvenssiä toisessa kooltaan samankokoisessa taulukossa ko hajautusavaimen osoittamassa paikassa.

-Kun hajautustaulun täyttöaste saavuttaa 80% kasvatetaan hajautustaulun kokoa ja hajautetaan kaikki monitavut uuteen taulukon kokoon. Samoin jos taulukosta ei löydy hajautusavaimen perusteella paikkaa tiedolle kasvatetaan taulun kokoa ja uudelleen hajautetaan tiedot.

Hajautustaulun periaatteiden mukaisesti tiedon lisääminen hajautustauluun on aikavaativuudeltaan keskimäärin O(1)-luokkaa. Hajautustaulun täyttöasteen kasvaessa tiedon lisääminen vie ajallisesti enintään

<0,8 * l * 16 = täyttöaste lähes 80%, *l*16* taulun koko, eli 80% taulun paikoista käytävä läpi (= n)

\+

n = uudelleenhajautuksesta johtuva aika, tietojen lukumäärä

\+

1 = keskimääräinen aikavaativuus yhden avaimen hajautukseen

yhteensä pahimmassa tapauksessa noin O(2n)-luokkaa l. O(n)-luokkaa.

Tiedon sisältymisen hakeminen hajautustaulusta käyttää samaa hajautusavaimen haun funktiota kuin tauluun lisääminen, joten sen aikavaativuus on keskimäärin O(1)-luokkaa. Pahimmassa tapauksessa pätee edellinen lauseke, jolloin tiedon hakeminen voi olla aikavaativuudeltaan O(n)-luokkaa.

Tilavaativuudeltaan hajautustaulu on enintään O(2n), l. O(n)-luokkaa johtuen sen luonteesta (ks edellä).

MultiByteEncoder:
Pakkaaja toimii seuraavasti:

-Luodaan hajautustauluun tiedot kaikesta käsiteltävästä tiedosta.
-Analysoidaan hajautustaulun tietojen esiintymisfrekvenssit ja samalla poistetaan ne joilla se ei ole riittävän korkea suhteessa tarkasteltavan dataikkunan pituuteen. Jos yhtään tietoa ei jää jäljelle, ei edetä.
-Tallennetaan yksiulotteinen taulukko esiintymisfrekvenssin perusteella laskevaan järjestykseen järjestetyistä tiedoista, saadaan hajautustaululta edellisessä kohdassa.
-Luodaan tavukoodit avaimista eli edellisestä kohdasta jäljelle jäävästä tiedosta.
-Pakataan käsiteltävä data.

Pakkauksen toiminta:

-Koodataan pakattuun dataan ikkunan leveys ja avainten tavujen lukumäärä
-Koodataan pakattuun dataan avaimet
-Koodataan pakattuun tietoon jäännös alkuperäisestä datasta.
@Luetaan syötteestä dataikkunan verran dataa
-Luodaan monitavu, ja kysytään hajautustaululta onko taulussa saman tiedon omaavaa monitavua
-Jos oli, lasketaan sille sen indeksistä sisäisessä taulukossa avain.
    -Jos avaimen koko täsmää edellisen avaimen kanssa, lisätään jakson pituutta koodauksessa
    -Jos avaimen koko ei täsmää, luodaan edelliselle avaimen koolle uusi prefiksi.
-Jos ei ollut, ko data koodataan raakadatana, avaimen koko on tällöin nolla (0).
    -Jos avaimen koko täsmää edellisen kanssa, lisätään jakson pituutta.
    -Jos ei, luodaan edelliselle avaimen koolle uusi prefiksi.
-Koodataan pakattuun dataan joko avain tai raakadata ja toistetaan @.

Aikavaativuudeltaan pakkaaja vaatii seuraavien operaatioiden ajat:

-Syötteen lukeminen hajautustauluun: keskimäärin O(1)
-Hajautustaulun tiedon analysointi, vapautus ja järjestäminen:
    Analysointi ja vapautus tehdään samassa, jokainen monitavu käydään läpi: O(n)
    Järjestäminen toteutettu lomitusjärjestämällä, pahimmassa tapauksessa (yhtään monitavua ei vapautettu): O(n log n)
-Syötteen uudelleen lukeminen: O(n)
-Lukemisen aikana tehtävä tarkistus (onko ko tieto hajautustaulussa): keskimäärin O(1)
-Lukemisen aikana tehtävä haku (mikä avain ko tiedolle): keskimäärin O(n/2) => O(n)

Yhteensä pahimmassa tapauksessa: O(1) + O(n) + O(n log n) + O(n) + O(1) + O(n) = O(n log n)

Tilavaativuudeltaan pakkaaja vie enintään kolme kertaan niin paljon kuin pakattava tieto: pakattava tieto luetaan muistiin O(n), siitä muodostetaan hajautustaulu joka vie enintään O(n), hajautustaulun analysoinnin jälkeen tallennetaan vielä yksiulotteinen taulu joka sisältää tiedot O(n). Yhteensä siis enintään O(3n) => O(n).

MultiByteDecoder:
Purkaja toimii käänteisesti pakkaajaan nähden:

-Luetaan syötteestä avaintaulukon prefiksi, joka kertoo dataikkunan leveyden ja avainten tavujen määrän.
-Puretaan syötteestä avaimet monitavuiksi
-Puretaan syötteestä jäännös
@Luetaan syötteestä prefiksi
    -Jos on avain luetaan seuraavat tavut prefiksin osoittaman leveyden mukaan ja puretaan avaimen perusteella, toistetaan prefiksin osoittaman jakson pituuden ajan
    -Jos ei ollut avain, puretaan tavuleveyden verran raakadataa, toistetaan prefiksin osoittaman jakson pituuden ajan.
-Toistetaan @.

Aikavaativuudeltaan purkajan toimenpiteet ovat luokkaa O(n), jossa *n* on pakatun tiedon koko.
Tilavaativuudeltaan purkaja on luokkaa O(2n) => O(n).

Puutteita ja parannuksia

Dokumentin kirjoitushetkellä pakkaus ei kaikissa tapauksissa ilmeisesti toimi oikein. Tämä tuli ilmi vasta demottaessa eikä sen jälkeen ole ollut aikaa virheen löytämiseen saatikka korjaukseen. Ennen demoa suoritetuissa pakkauksissa kaikennäköisellä tiedolla pakkaus toimi oikein, joten on varsin outoa, ettei se enää toiminut pari päivää myöhemmin. Pakkaamiseen on kirjoitettu testejä jotka tarkistavat pakkaamisen tuloksena olevasta tiedosta sen oikeellisuuden, ja nämä testit menevät läpi ongelmitta. Joskus testejä ajettaessa pakkaamisen tai purkamisen testit jumittuvat tuntemattomasta syystä ja ne on käynnistettävä uudestaan, mutta menevät kuitenkin joka kerta läpi.

Kirjoitushetkellä graafinen käyttöliittymä ei toimi oikein: tieto pystytään pakkaamaan mutta pakatun tiedon tiedostoon tallentaminen ei tuntemattomasta syystä onnistu, luultavasti javan tapa käsitellä tiedostopolkuja tässä muodossa ei ole oikea. Lisäksi graafiseen käyttöliittymään rakennettu ominaisuus keskeyttää operaatio (pakkaus tai purku) ei toimi, eli keskeyttäminen ei onnistu. Luultavasti kyse on tavasta, jolla Java käsittelee säikeitä graafisten käyttöliittymien yhteydessä.

Graafiseen käyttöliittymään ei ole koodattu rekursiivisia tai kansioiden pakkaukseen liittyviä menetelmiä. Tekstipohjaisessa käyttöliittymässä pakattava tai purettava tiedosto tulee olla samassa kansiossa kuin ohjelma, tässäkään absoluuttisten polkujen käyttö ei toimi suunnitellusti.

Ohjelma on rakennettu käyttämään säikeitä, joten rekursiivisten menetelmien koodaaminen pitäisi olla kohtalaisen helppoa.

Pakkaamisen päämetodi tulisi analysoida ja refaktoroida jotta pakkaaminen toimisi oikein ja että se olisi siistimpi.

Lähteet:

en.wikipedia.org:
- Lempel-Ziv-Welch
- Run-length encoding
- Lempel-Ziv
- Huffman coding
- perfect hash function
- hash table

7-zip.org/sdk.html

A STATISTICAL LEMPEL-ZIV COMPRESSION ALGORITHM FOR PERSONAL DIGITAL ASSISTANT (PDA), IEEE Transactions on Consumer Electronics, Vol. 47, No. 1, FEBRUARY 2001