Toteutusdokumentti

Ohjelman Yleisrakenne:

Pääluokat:

Huffman luokka

Main Luokka josta ajetaan muita luokkia, ja jossa toistaiseksi on vajavainen käyttöliittymä.

Tiedostonkasittelija

Luokka joka käsittelee tiedoston characterejen määrän laskemista ja kompressoidun tiedoston luomista. Kutsutaan ja luodaan Main metodissa, ei näe muita luokkia mutta tarvitsee listan uusista biitjonoista chareille Puunkäsittelijältä.

Puunkasittejia

Luokka joka luo puun Tiedostonkäsittelijältä saadulta charejen frekvenssiä kuvaavasta listasta, jonka mukaan määrätään charejen uudet bittijonot. Kutsutaan ja luodaan Main metodissa, ei näe muita pää luokkia.

Tulkki

Luokka joka hoitaa compressoidun tiedoston saattamisen takaisin luettavaan muotoon. On myös Tiedostonkäsittelijän ja Puunkäsittelijän välillä jotta nuo kaksi eivät joudu sotkeutumaan keskenään. Luodaan ja kutsutaan Main metodista.

Tietorakenteet:

MaaraLista:

Lista characthereista ja niihin liittyvistä määristä alkuperäisessä dokumentissa. MaaraListaNode on listan objekti.

MerkkiLista:

Lista characthereista ja niihin liittyvistä uusista bittirepresentaatioista jotka rakennettiin puun avulla. MerkkiListaNode on listan objekti.

Merkki:

Luokka tallentamaan characteri ja niihin liittyvä uusi bittijono.

Solmu:

Luodun puun node.

MinimiKeko:

Solmuista koottu Array puun rakentamisen mahdollistamiseksi.

Saavutetut aika-tilavaativuudet:

Pseudokoodi ohjelmalle menee näin kompressoinnille:

```
LueTiedosto()    //O(N) Jossa N on tiedostona annetun syötteen koko.
LuoPuu()         //O(1) Määrittelydokumentissa esitetyistä syistä.
While(tekstiaJaljella)         //O(N)
        LueTiedostosta()
        KirjoitaTiedostoonKompressoituData()
```

Pseudokoodi purkamiselle:

```
LueJaLuoPuu()         //O(1)
While(tekstiaJaljella) //O(N)
        LueKompTiedostosta()
        KirjoitaLuettavaanTiedostoon()
```

Koodista voi selkeästi siis nähdä että aikavaatimus on O(N).

Tilavaatimus itse algoritmille on O(1), sillä sen tarvitsee tallentaa muistiin ainoastaan kaksi max 256 alkiota sisältävää listaa ja yhden PriorityQueuen, ja max 511 solmua sisältävän puun, ja muutaman kahdeksan kokoisen arrayn ja vakiomäärän muuttujia.
Jos itse tiedosto tai sen kompressoitu muoto lasketaan mukaan niin sitten tulee tilavaatimukseksi O(1+N) = O(N) jossa N on tiedoston koko.

Parannusehdotukset:

Nätimpi ja monimutkaisempi käyttöliittymä voisi tehdä ohjelmasta mukavamman käytettävän.

Tehokkuuden optimointia listojen läpikäynnissä monimutkaisemmilla etsintäalgoritmeilla tai järjestämällä listoja ja stringejen luontia esim. StringBuilderillä tai muulla vastaavalla new String komennon sijaan. Pullonkaulaksi tehokkuuden kannalta muodostuu itse tekstin lukeminen ja kirjoittaminen tiedostoon. Jos osaamista olisi, on varmaan tällä alueella eniten viilattavaa tehokkuuden suhteen.

Hoidan tällä hetkellä ongelmatilanteen, jossa writer printtaa ekstra kirjaimen tiedoston loppuun koska bittibufferi ei täyttynyt viimeisestä kirjaimesta ja se lukee bufferin lopun uutena kirjaimena, liittämällä kompressoituun tiedostoon charejen määrän. Uskon että ongelmaan on siistimpikin ratkaisu joka säästäisi muutaman bytein, mutta en tähän

hätään pysynyt sellaista keksimään.

Mainitsin joitakin erikoistapauksia testausdokumentissani jossa ohjelma tuottaa virheellisiä characterejä purettuun tiedostoon windowsin puolella erikoismerkkeihin liittyen, tai luo tyhjästä ÿ merkin. Näiden erikoisten bugejen poistaminen myös parantaisi ohjelmaa.

Kompressoinnin tehostamiseksi voisi pelkästään yksittäisten merkkejen laskemisen lisäksi ottaa mukaan yleisimpien merkkejen kombinaatioita tai yleisimmin esiintyviä kombinaatioita esim ”aa”, ”kk” suomessa tai ”th” englannissa.