# Tekstin Tiivistys - Huffman algoritmi
# Määrittelydokumentti

v1.09

Tulen toteuttamaan Huffmannin algoritmin tekstitiedoston kompressointiin. Kompressoitu tiedosto tulee myös olla muotoa jonka pystyy ilman ongelmia saattamaan taas luettavaan muotoon eri istunnosta. Tavoite on että syötteenä annettun tekstitiedoston saisi n.40-60% pienemmäksi kooltaan alkuperäiseen verrattuna.

## Tiedossa olevat käyttöön tulevat algoritmit:

Huffmannin algoritmi:

Selkeä ja toimiva algoritmi teksitiedoston kompressointiin.

Puun esijärjestyksessä läpikäynti:

Meidän tarkoitukseen sopiva binäärisen hakupuun läpikäynti algoritmi.

Integer.toBinaryString(int i)

Helpoin tapa saattaa integer binääriksi.

## Tiedossa olevat käyttöön tulevat tietorakenteet:

Binäärinen Hakupuu:

Helpoin tapa jakaa uudet binääriarvot kullekkin merkille riippuen niiden toistojen määrästä syötteessä.

PriorityQueue:

Tehokas tietorakenne puun rakentamiseen, sillä pienimpiä solmuja poistetaan ja uusia lisätään solmuja tietorakenteeseen useasti puun rakennuksen aikana.

LinkedList:

Kaksi LinkedList muotoista listaa, yksi säilömään merkkejen frekvenssiä alkuperäisessä tiedostossa, ja toinen säilömään jokaista löydettyä merkkiä vastaavan uuden bitti representaation. Koska maksimikoko ei kummassakaan ylety yli 256, ei näinkään alkeellisen listan läpikäynti monesti tuota pullonkaulaa.

Integer Array

Määritellyn (8) kokoinen lista tavujen kirjoittamisen helpottamiseksi. Koska lista on tietyn kokoinen, ja sen sisältö vaihtuu usein, on Arrayn käyttö tässä tapauksessa tehokkain ja aika ja tilavaatimuksiltaan.

## Syöte:
Ohjelma saa syötteekseen joko normaalin tekstitiedoston josta se tuottaa kompressoidun version tiedostosta huffman-algoritmia käyttäen, tai sille annetaan huffman-algoritmia käyttäen jo kompressoitu tiedosto jonka se saattaa takaisin luettavaan muotoon.
Valinta siitä että kummasta tilanteesta on kyse tehdään käyttäjältä kysytyn syötteen avulla (joko c tai u), ja ohjelma myös kysyy lähdetiedoston nimeä/sijaintia käyttäjältä.

## Tilavaatimus:
Itse algoritmi tarvitsee tilaa vain tietylle määrälle ascii merkkejä ja niihin liittyviin numeroihin maksimissaan O(1), muttta tietenkin jos kompressoitu tiedosto lasketaan mukkaan niin tilavaatimus nousee O(0.4n-0.6n) eli O(n).

## Aikavaatimus:
Kompressoinnissa, koska erilaisia merkkejä on vain teitty määrä, hakupuun ja priorityqueuen läpikäyntiin kuluu maksimissaan vain lineaarinen määrä aikaa O(1), eli meidän pitää välittää ainoastaan tekstitiedoston lukemisesta ja merkin tiedon lisäämisestä Arrayhin. Vaikka tämä joudutaan tekemään kahdesti, niin aikavaatimus ei ole O(2n) suurempi, eli riippuvainen syötteen (tiedoston) koosta. Koska kertoimet eivät tähän aikavaatimuksen merkintä tapaan vaikuta, on kompressoinnilla arvo O(n).
Takaisin luettavaan muotoon saatettaessa taas käydään jokainen merkki kompressoidussa tiedostossa läpi, jokaista merkkiä kohden pitää käydä binäärihakupuuta läpi ja kompressoidussa muodossa oleva bittisetti vaihdetaan takaisin alkuperäiseen. Hakupuulla on maksimissaan 255 lehteä, eli emme ole puun läpikäynnistä tässä tilanteessa kiinostuneita. Eli aikavaatimus on yhä täysin riippuvainen syöteen koosta O(n).

## Lähteet:
http://en.wikipedia.org/wiki/Huffman_coding
https://www.youtube.com/watch?v=ZdooBTdW5bM