Tietorakenteiden Harjoitustyö

Antti Tupamäki

Helsinki 30.04.10

HELSINGIN YLIOPISTO
Tietojenkäsittelytieteen laitos

HELSINGIN YLIOPISTO – HELSINGFORS UNIVERSITET – UNIVERSITY OF HELSINKI

Tiedekunta – Fakultet –Faculty	Laitos – Institution – Department
Matemaattis-luonnontieteellinen tiedekunta	Tietojenkäsittelytieteen laitos

Tekijä – Författare – Author

Antti Tupamäki

Työn nimi – Arbetets titel – Title

Tietorakenteiden Harjoitustyö Toteustusdokumentti

Oppiaine – Läroämne – Subject

Tietojenkäsittelytiede

Työn laji – Arbetets art – Level	Aika – Datum – Month and year	Sivumäärä – Sidoantal – Number of pages
	30.04.10	2

Tiivistelmä – Referat – Abstract

Toteutus dokumentti

Avainsanat – Nyckelord – Keywords

Säilytyspaikka – Förvaringställe – Where deposited

Muita tietoja – Övriga uppgifter – Additional information

Table of Contents

1 Johdanto

Työn aiheena oli tiedoston pakkaaminen Huffman koodilla. Tälle ohjelmalle annetaan tiedosto joka pakataan Huffman puuta käyttäen. Ja puretaan tiedoston alkuun asetettavalla Huffman puulla

2 Ohjelman rakenne

Ohjelma koostuu päätiedostosta joka hoitaa tiedoston lukemisen, Huffman puun generoinnin koodin muodostamisen Huffman puusta ja tiedoston enkoodauksen sekä dekoodaamisen.

Päätiedostoa tukee Heap-luokka joka käsittelee kekoa, ja toteuttaa tähän heap-insert

heap-pop, heap-heapifyn

Sekä Tree moduuli joka koostuu kolmesta luokasta Tree joka on yläluokka, Leaf ja node jotka perivät treen. Tämän luokan avulla muodostetaan itse Huffman puu. Leaf sisältää merkin arvon ja tämän freqvenssin. Node taas sisältää lapsiensa frekvenssit

Ohjelma luo Main luokassa pakattaessa tiedoston_nimi.endc nimisen tiedoston joka voidaan purkaa eri päätteisten tiedostojen purkaminen ei onnistu.

2.1 Tietorakenteet

Ohjelma luo ajonaikana Keon, ja Huffman puun. Kekoon lisääminen tapahtuu ajassa log(n) koska tämä sisältää heapifyn ja keosta poistaminen myös ajassa log(n) koska tämäkin sisältää heapifyn. Periaatteessa nämä voisi suorittaa vakioajassa mutta tällöin keko ehto voi rikkooontua. Heapify toimii ajassa log(n). Huffman puun generointi onnistuu ajassa 0(n) ja sieltä haku onnistuu ajassa 0(n) missä n on korkeus

itse algorytmi suoritettiin seuraavalla pseudokoodilla

1. Create a leaf node for each symbol and add it to the priority queue.
2. While there is more than one node in the queue:
    1. Remove the two nodes of highest priority (lowest probability) from the queue
    2. Create a new internal node with these two nodes as children and with probability equal to the sum of the two nodes' probabilities.

3. Add the new node to the queue.
3. The remaining node is the root node and the tree is complete.

(lähde)[1]

tämän aikavaatimus on Olog(n)

heapissa käytettiin periaatteessa heapsort algoritmia **function** heapSort(a, count) **is**

```
     input:  an unordered array a of length count

     (first place a in max-heap order)
     heapify(a, count)

     end := count-1 //in languages with zero-based arrays the children are
2*i+1 and 2*i+2
     while end > 0 do
          (swap the root(maximum value) of the heap with the last element of
the heap)
          swap(a[end], a[0])
          (put the heap back in max-heap order)
          siftDown(a, 0, end-1)
          (decrease the size of the heap by one so that the previous max
value will
          stay in its proper placement)
          end := end - 1

 function heapify(a,count) is
     (start is assigned the index in a of the last parent node)
     start := (count - 2) / 2

     while start ≥ 0 do
          (sift down the node at index start to the proper place such that
all nodes below
           the start index are in heap order)
          siftDown(a, start, count-1)
          start := start - 1
     (after sifting down the root all nodes/elements are in heap order)

 function siftDown(a, start, end) is
     input:  end represents the limit of how far down the heap
                    to sift.
     root := start

     while root * 2 + 1 ≤ end do          (While the root has at least one
child)
          child := root * 2 + 1           (root*2+1 points to the left
child)
          (If the child has a sibling and the child's value is less than its
sibling's...)
          if child < end and a[child] < a[child + 1] then
```

```
            child := child + 1           (... then point to the right
child instead)
        if a[root] < a[child] then       (out of max-heap order)
            swap(a[root], a[child])
            root := child                (repeat to continue sifting down
the child now)
        else
            return
```

[2]

Rubyssä on myös erittäin monikäyttöinen taulukko ja tästä syystä sitä voidaan käyttää myös jonona ,pinona ja firts-in-first-out-jonona näitä ei välttämättä ole mainittu koodissa erikseen. Näihin lisääminen ja poistaminen tapahtuu vakioajassa.

2.2 Ohjelman kääntäminen

ohjelma ajetaan komentorivillä ruby Main.rb ”haluttu_tiedosto” ”dekoodaus” vai ”enkoodaus”. Voit purkaa ainoastaan .encd päätteellä pakattuja tiedostoja, mutta tiedosta pakatessa tyypillä ei ole väliä. Tosin esim kuvia ”pakatessa en ole tiedostojen oikeellisuudesta varma”

2.3 Syötteet ja tulosteet

Ohjelmalle syötetään haluttu tiedoston nimi ja parametri halutaanko enkoodata vai dekoodata. Ohjelma tämän jälkeen suorittaa enkoodauksen tai dekoodauksen. Virhetilanneessa tulee teksti jossa kerrotaan virheen tyypistä esim. jos haluttua tiedosta ei löydy kyseisestä kansiosta tulee teksti tiedostoa ei löytynyt

Testit

Ohjelma voidaan testata ajamalla irbissä main.rb testi scripti test_script tai ajamalla rspec testit tämä tapahtuu rspec gemin avulla. Tosin Rspec testi eivät ole täysin kattavia

Lähteet:[1]http://en.wikipedia.org/wiki/Heap_(data_structure)

[2] http://en.wikipedia.org/wiki/Heapsort