Toteutusdokumentti

1 Ohjelman yleisrakenne

Ohjelman rakenne on hyvin yksinkertainen. Pakkauksia on neljä:

1. `main` sisältää pääohjelman, joka lukee asetukset ja käynnistää pakkauksen/purkamisen. Ainoa luokka on `Main`.
2. `util` sisältää joitain itse toteutettuja yleishyödyllisiä luokkia:
   - `BitInputStream`, `BitOutputStream`: mahdollistavat biteittäin tapahtuvan streamien lukemisen ja kirjoittamisen
   - `List`, `Set`: itse toteutetut `ArrayList` ja `HashSet`
   - `Pair`: kahden objektin pari
   - `Math`: muutama simppeli matematiikkafunktio
   - `Options`, `Option`: komentoriviltä annettavien parametrien lukeminen
3. `huffman` sisältää Huffman-koodausalgoritmit. Luokat:
   a) `Huffman`: varsinaset pakkaus/purkufunktiot sekä tarvittavat apufunktiot. Kaikki staattisia metodeja.
   b) `HuffmanTree`, `HuffmanTreeNode`: Huffman-puu (tavallinen binääripuu paitsi että lehtiin pääsee suoraan käsiksi taulukon kautta).
   c) `HuffmanHeap`: itse toteutettu erikoistapaus `PriorityQueuesta` (minimikeko). Käytetään Huffman-puun konstruoinnissa.
4. `lzw` sisältää LZW-koodausalgoritmit: Luokat:
   a) `LZW`: varsinaiset pakkaus/purkufunktiot sekä tarvittavat apufunktiot. Kaikki staattisia metodeja.
   b) `LZWDictionary`, `LZWDictionaryEntry`: pakkausvaiheessa käytettävä sanasto (prefiksipuu).

2 Saavutetut aika- ja tilavaativuudet

Luokkien List, Set ja HuffmanHeap vaativuudet ovat samat kuin Javan valmiilla ArrayList, HashSet ja PriorityQueue -luokilla.

2.1 Huffman

2.1.1 Pakkaus

Pakkausalgoritmi:

```
compress():
  frequencies = calculateFrequencies()
  tree = buildTree(frequencies)
  while !EOF:
    b = read()
    write(findCode(tree, b))
```

Funktio `calculateFrequencies()` on yksinkertainen:

```
calculateFrequencies():
  result = int[256]
  while !EOF:
    b = read()
    result[b]++
  return result
```

Kyseessä on $O(n)$-aikainen ja $O(1)$-tilainen funktio.

Funktio `buildTree()` muodostaa merkkitaajuuksista Huffman-puun:

```
buildTree(frequencies):
  queue = new MinHeap()
  for b = 0 ... 255:
    queue.push(TreeNode(frequencies[b]))

  while queue.size >= 2:
    a = queue.pop()
    b = queue.pop()
    queue.push(new Node(a.frequency + b.frequency))

  return queue.pop()
```

Ensimmäinen simukka käydään aina 256 kertaa läpi. Toisessa silmukassa jonon alkioiden määrä vähenee aina yhdellä, joten sekin käydään (noin) 256 kertaa läpi. Lopputuloksena `buildTree()` on vakioaikainen ja -tilainen.

Sitten vielä `findCode()`:

```
findCode(node):
  result = new List()
  while node.parent:
    if node == node.parent.right:
      res.add(true)
    else:
      res.add(false)
    node = node.parent
  return res.reverse()
```

Kuten `buildTree()`, Huffman-puun korkeus on korkeintaan 256. Tästä seuraa, että `findCode()`:n silmukka ajetaan korkeintaan 256 kertaa ja palautettava lista on korkeintaan näin pitkä. Siis kyseessä on vakioaikainen- ja tilainen funktio. Lopputuloksena `compress()` on $O(n)$-aikainen ja $O(1)$-tilainen.

3 Työn mahdolliset puutteet ja parannusehdotukset

4 Lähteet