Tema 8: Algoritmdesigntekniker

Daniel Andersson `daan3440`

10 mars 2019

1 Huffman

1.1 Inledning

Huffmans algoritm är en tämligen enkel algoritm att förstå sig på. En av anledningarna är att den är *greedy*. Det gör att en kan blicka på nästa steg och inte behöver bekymra sig över vad som har gjorts. Det är teorin iallafall.
Huffman är en *prefix code*, icke destruktiv och använder sig av variabel längd i komprimeringen. Komprimering skall i sammanhanget förstås som en ompacketering och innebär inte alltid att storleken på datan blir mindre. Under testning så blev t.e.x 200kb-filer över en 1mb "komprimerade".

1.2 Huffman.java

Huffman.java är huvudklassen. Där finns main-funktionen och de metoder som hanterar *I/O*, *encoding*, *decoding*, *statistik* och skapandet av de träd som används vid packningen och expanderingen av datat. Klassen Tree är en enkel klass för träd och för noder. Det enda vi bekymrar oss över vad gäller träden är deras *root*.
Nodeklassen är också enkel. För att kunna användas i PriorityQueue i metoden createTree() så implementerar den Comparable. Den har

också getters och setters för parent, left och right node samt booleans för om de är *root* och *leafs*. Se figur 1.

1.3 readFile(String path, Charset encoding))

Läser in en fil, lägger hela innehållet i en byte array och returnerar en String med teckenkodningen.

1.4 stats(char[] charArray)

Metoden stats(char[] charArray) tar den returnerade strängen som char-array från readFile() som argument. Metoden ograniserar sedan datan i en Map<Character, Integer> med *char* som *key* och antal av sagda *char* som (int) värde. Mapen returneras sedan.

1.5 createTree(Map<Character, Integer> stats, List<Node> leafs)

createTree() tar en *Map* och en *Lista* med *Nodes* som argument och returnerar ett *Tree*. Det första vi gör är att gå igenom mapens keyset och för varje *key* i den skapar vi en *Node* och sätter dess frekvens. Sen lägger vi noden i en *PriorityQueue*. Vi adderar och noden i den tomma listan vi skickat som argument.
Vi går sedan igenom *PriorityQueue* genom att ta de två högst prioriterade (eller endast den första om kön bara har en nod) och skapa en ny nod med summan av de båda nodernas frekvenser.
Notera att algoritmiskt så ska en addera *trees* genom att lägga ihop träd och summera dem i en ny root. Här gör vi detta och vi hanterar endast varje subträds rootnod. De lägger vi ihop och summerar i en ny rootnod. Sen lägger lägger vi till denna nya nod i *PriorityQueue*.

Här är ett exempel på va som sker: De två första i Prioritetskön {a,b} läggs ihop i en nod. Nytt värde är 3. Sedan läggs nästa till {c} och läggs ihop. Nytt värde blir 6.

```
Stats: {a=1, b=2, c=3}
Node chars: ab
Node Freq: 3
Node chars: cab
Node Freq: 6
```

När alla *keys* är bearbetade så skapar vi trädet vi returnerar genom att sätta den första noden i kön till det nya trädets root.

1.6 buildEncodingInfo(List<Node> leafNodes)

Tar en lista som argument och skapar en väg som går att följa via kodningen 0 för leftChild och 1 för rightChild. Informationen sparas i en 'Map<Character, String>' och returneras där värdet är den sträng av koder som representerar varje char.
Exempel på körning av strängen {abbccc}:

```
Chars: a = 10
ReturnBuffer: 10
Chars: b = 11
ReturnBuffer: 1011
Chars: b = 11
ReturnBuffer: 101111
Chars: c = 0
ReturnBuffer: 1011110
Chars: c = 0
ReturnBuffer: 10111100
Chars: c = 0
ReturnBuffer: 101111000
Encoded: 101111000
```

1.7 encode(String originalString, Map<Character, Integer> stats)

Metoden kallar på createTree och buildiEncodingInfo() och använder sedan mapen som returneras från den sistnämnda för att skriva ut

en sträng av alla tecken som har fått koder.
Från exemplet i 1.6:

```
1 String: 'abbccc'
2 Encoded: 101111000
```

1.8 decode(String binaryStr,Map<Character, Integer> stats)

Metoden decode(String binaryStr,Map<Character, Integer> stats) tar en *encoded* sträng i formatet i exemplet från 1.6 och packar upp dem. Strängen som är komprimerad läggs in i en byte array och placeras sedan i en länkad lista. Sedan skapas ett träd av informationen som finns i Mapen 'stats' som skickades som argument . Sedan packas trädet upp genom att matcha noderna och vandra i trädet via de barn som existerar och bygga en sträng av de charvärden som noderna har för att till sist returnera den byggda strängen.

Figur 1: Huffman