Lab 11

[Frivillig] Kompression 2, Huffmankodning

Syftet med denna laboration är dels att träna på pythonprogrammering, dels att öva på datakompression (huffmankodning), och därtill att återkoppla till DALGO (för er som läst denna kurs).

Laborationen innehåller lite mer utmanade (och roligare) algoritmer än tidigare labbar, och laborationsanvisningarna är relativt sparsamma. Detta för att lämna ett större utrymme åt personlig kreativitet.

Laborationen är frivillig men ger labpoäng om den lämnas in före deadline: tisdag vecka 12, 2018-03-20, 23:55.

11.1 Uppgift 1, beräkna medellängden

I en tidigare laboration skulle man bland annat beräkna entropin för de tecken som ingår i en viss exempeltext som ligger i pingpong.

Uppgift

Skriv ett program som skapar en huffmankod baserad på statistik för samma exempeltext. Beräkna också kodordsmedellängden! Om du gör rätt bör medellängden hamna i närheten den tidigare beräknade entropin.

Tips

Istället för en utförlig steg-för-steg beskrivning ger vi här bara några tips. Det är frivilligt att följa dessa tips!

1. Man kan representera ett huffmanträd på många olika sätt i python, här beskriver jag en möjlig metod.

   Man kan representera en nod med hjälp av `Node`-klassen från tidigare laboration. Alla noder lagrar en sannolikhet (`n.prio`). Löven lagrar därtill ett *byte*-värde (i `n.data`). De interna noderna lagrar istället två subträd som en tuple i `n.data`.

   (a) Löv kan representeras med ett `Node(p, byte)`-objekt.

   (b) Interna noder representeras med ett `Node( p, (t1, t2))` objekt, där `t1` och `t2` är två subträd.

   (c) För att avgöra om en nod `n` är ett löv eller en intern nod kan man undersöka datatypen hos `n.data`. Om `type(n.data)==int` är det ett löv, annars en intern nod.

2. Skapa huffmanträdet med hjälp av en prioritetskö som från början fylls med alla de noder som så småningom skall bli löv.

3. När man väl har skapat trädet bör man kunna beräkna den eftersökta medellängden med en relativt enkel rekursion.

   (a) Om man vill kan man dock istället lösa uppgift 2, och sedan återkomma till beräkningen av medelängden

11.2 Uppgift 2, skriv ut alla kodord

Skriv ut en tabell med alla (upp till 256) olika oktetter (eng. *byte*) som förekommer i din kod och deras binära kodord. För varje *byte* skall tabellen visa (se nedanstående exempel):

1. *byte*-värdet (tal mellan 0 och 255)
2. ASCII-tecknet ifall det aktuella *byte*-värdet ligger mellan 32 och 127
3. Det binära kodordet.
4. Antal bitar i detta kodord.
5. Den ideala kodordslängden, dvs $\log_2 \frac{1}{P(x)}$

Nedan syns en liten del av den eftersökta tabellen:

```
byte= 87 (W) 1110110111100       len=13 log(1/p)=12.3
byte= 89 (Y) 1110110110110101    len=15 log(1/p)=14.9
byte= 91 ([) 1110110010 10       len=12 log(1/p)=11.6
byte= 93 (]) 1110110010 11       len=12 log(1/p)=11.6
byte= 97 (a) 1011                len= 4 log(1/p)=3.72
byte= 98 (b) 1110101             len= 7 log(1/p)=6.58
byte= 99 (c) 1010011             len= 7 log(1/p)=6.8
byte=100 (d) 10101               len= 5 log(1/p)=4.69
byte=101 (e) 1111                len= 4 log(1/p)=3.63
byte=102 (f) 010011              len= 6 log(1/p)=6.25
byte=103 (g) 00101               len= 5 log(1/p)=5.39
byte=104 (h) 010100              len= 6 log(1/p)=6.2
byte=105 (i) 0000                len= 4 log(1/p)=4.57
byte=106 (j) 10100100            len= 8 log(1/p)=8.0
byte=107 (k) 01000               len= 5 log(1/p)=5.35
byte=108 (l) 0001                len= 4 log(1/p)=4.52
byte=109 (m) 00100               len= 5 log(1/p)=5.42
byte=110 (n) 1000                len= 4 log(1/p)=3.97
byte=111 (o) 01110               len= 5 log(1/p)=5.04
byte=112 (p) 011111              len= 6 log(1/p)=5.92
byte=113 (q) 10100101001001      len=14 log(1/p)=13.9
```

Notera att de faktiska kodordslängderna är ganska lika de ideala längderna, och att exempelvis kodordet för det vanliga tecknet a är betydligt kortare än längden för ovanliga q.

Om man beräknar medelvärdet, $\sum P(x)L(x)$, för de faktiska kodordslängderna får man det medelvärde som eftersöks i uppgift 1, och om man istället beräknar medelvärdet av de ideala längderna får man entropin.

11.2.1 Tips

Det är frivilligt att följa dessa tips.

1. Skapa ett globalt dict()-objekt med uppgift att för varje i koden förekommande *byte* lagra motsvarande binära kodord.
2. Skriv en rekursiv funktion som klättrar i huffmanträdet och placerar in alla lövs *byte*-värden och deras kodord i detta dict()-object
3. Iterera igenom alla *byte*-värden 0...255. Om motsvarande *byte* finns i dict()-objektet, skriv ut *byte*-värdet, etc.

11.3 Redovisning

Redovisa genom att

1. Demonstrera programmet för labbansvarig, var beredd på att förklara din kod.
2. Ladda upp din källkod (py-filen) till pingpong.