Lab 9

[Obligatorisk] Kompression 1

Denna laboration behandlar datakompression av text, inklusive entropibegreppet och minneskällor. (Laborationen berör också skillnaden mellan 8 bitars utvidgad ASCII och UTF-8.)

9.1 Teckenkodsuppgifter

Nedanstående uppgifter behandlar ASCII, LATIN-1, och UTF-8.

1. Som bekant kan man göra om en text till en *byte*-array. Knappa in nedanstående i din interaktiva pythonterminal och tolka resultatet!

   ```
   >>txt = "ABC abc"
   >>b = bytearray(txt, "ASCII")
   >>len( txt )
   >>len( b )
   >>b[0]
   >>b[1]
   >>b[2]
   >>b[3]
   >>b[4]
   >>b[5]
   >>b[6]
   ```

   Vilka *byte*-värden motsvarar de 7 olika tecknen (ABCabc och mellanslag)?

2. Inrymmer ASCII-koden svenska tecken? *Försök* att omvandla textsträngen *ÅÄÖ* till en ASCII-*byte*-array, och undersök om du får en felutskrift eller inte.

3. Eftersom ASCII är en 7-bitars kod har den endast plats för 128 olika tecken. Det finns flera utvidgningar av ASCII som använder alla 8 bitar i en *byte*, och som därmed har plats för 256 olika tecken. *LATIN-1* är ett exempel på en sådan utvidgning. Prova att koda *ÅÄÖ* till en *byte*-array med teckenkoden LATIN-1". Hur kodas *Å, Ä och Ö?*

4. UTF-8 är en annan slags utvidgning. UTF-8 är en variabel-längd-kod. Vissa tecken är en *byte*, men andra tecken kan vara två eller tre *byte* långa. Koda *ÅÄÖ* till en *byte*-array med "UTF-8". Hur kodas *Å, Ä och Ö?*

9.2 Kompressionsuppgifter

I pingpong ligger det en fil som heter *exempeltext.txt (alt exempeltextMac.txt)*. Kopiera den till din egen labkatalog. Titta på filens innehåll, så att du ser vilket språk den är skriven på, etc. Skriv sedan ett pythonprogram som skall göra följande:

1. Läsa och undersöka filen...
   (a) Öppna filen och läs in dess innehåll till en variabel `txt`
   (b) Konvertera texten till en *byte*-array med
   `byteArr = bytearray(txt, "utf-8")`
   (c) Hur många tecken innehåller texten? Och hur lång är *byte*-arrayen? Förklara eventuella skillnader! Att redovisa!
2. Beräkna statistik och entropi...
   (a) Skriv en funktion `makeHisto( byteArr )` som returnerar ett histogram (array av längd 256) som anger hur många gånger varje tal (0-255) förekommer i `byteArr`.
   (b) Skriv en funktion `makeProb( histo )` som givet ett histogram returnerar en sannolikhetsfunktion. Sannolikhetsfunktionen är en array som är lika lång som histo och den innehåller ett normerat histogram som summerar till 1.
   (c) Skriv en funktion `entropi( prob )` som returnerar sannolikhetsfunktionens entropi, $H = \sum_i P(i) \log \frac{1}{P(i)}$, där log avser logaritmen i bas 2. . Det gäller att undvika division med 0, och det är därför viktigt att känna till att $p \log \frac{1}{p} \to 0$ då $p \to 0$.
   (d) Ned till *hur många bit/byte* skulle man kunna komprimera *byte*-arrayen om den behandlades som en minnesfri källa och kodades optimalt? Att redovisa!

3. Kopierara texten och blanda kopian...

   (a) Importera pythonmodulen `random`.

   (b) Gör en kopia (`theCopy`) av `byteArr` och blanda den slumpmässigt med hjälp av `random.shuffle( theCopy )`. Kopian kommer nu att innehålla samma tal som orginalarrayen, men i en slumpmässig ordning –som om den vore genererad av en minnessfri källa!

   (c) Verifiera att du inte av misstag också har råkat blanda `byteArr`!

4. Nedan skall vi zip-komprimera `theCopy`. Syftet är att undersöka hur mycket zip-algoritmen klarar av att komprimera denna *byte*-array. Enligt teorin skall det inte gå att komprimera minnesfritt data under entropigränsen...

   (a) Importera pythonmodulen `zlib`.

   (b) Anropet `kod = zlib.compress(theCopy)` komprimerar kopian och returnerar dess kod som en ny *byte*-array. Gör detta!

   (c) Tag reda på hur lång zip-koden blir *mätt i byte*. En *byte* är ju 8 bit, så hur lång blir koden *mätt i bit?* Hur många *källsymboler* (dvs *byte*) innehåller `theCopy`? Och ned till hur många bit/källsymbol har alltså zip-algoritmen lyckats komprimera den minnesfria kopian? Att redovisa!

   (d) Upprepa ovanstående experiment på den oblandade arrayen `byteArr`. Ned till hur många bit/källsymbol kan zip-algoritmen komprimera denna array?

   (e) Du har nu tre olika mätetal på antal bit/källsymbol. Dels (A) entropin. Dels (B) zlib-kodning av randomiserad källa. Dels (C) zlib-kodning av orginalkälla. I vilket av de tre fallen fick vi... Att redovisa!

      i. lägst antal bit/källsymbol
      ii. och i vilket fall fick vi högst antal bit/källsymbol
      iii. förklara detta!

5. Zippa repeterad text...

   (a) Lägg en godtycklig liten text i en variabel, och lägg samma text fast repeterad 10 gånger i en annan variabel, t.ex

```
t1   = """Hoppas att denna laboration aldrig tar slut
     för nu börjar ett mycket intressant experiment! """
t10 = 10*t1
```

(b) Testa! Till hur många *byte* zippas den första respektive den andra texten? Att redovisa!

(c) Strängen `t10` innehåller tio gånger så många tecken som `t1`, men blir zip-koden för `t10` tio gånger så lång? Förklara varför/varför inte! Att redovisa!

9.3 Redovisning

Ladda upp följande till pingpong

1. Källkoden (py-filen)
2. Ett pdf-dokument (eller en vanlig pingpongkommentar) som besvarar de frågor som är märkta med *att redovisa* i marginalen i detta pm.