2013-02-14

# Programkonstruktion och datastrukturer
# Inlämningsuppgift 3

ziv-lempel komprimering

Daniel Eliassen
Philip Åkerfeldt

Inlämning 2
2013-02-28

## Funktioner

Funktioner som används:

**Vector.update** - Uppdaterar en vektor i konstant tid oberoende på givna argument. Worst-case Θ(1).
**Vector.tabulate** – Används i decompress och tar funktionen chr. Detta tar linjär tid mot storleken på vektorn, som alltid är 4049, alltså konstant. Θ(1)
**Vector.sub** - Arbetar i konstant tid oberoende på storlek av indata. Worst-case Θ(1).
**concatinate** - infix operator kör i linjär tid mot summan av längderna på sina två argument. Worstcase Θ(n).
**build** - skapar en hashtabell och lägger in de 255 första tecken i ASCII.
**map** - map tar en funktion och applicerar denna på varje element i listan som anropas. Map används i compress och anropas med funktionen str. Funktionen str är linjär mot längden på sitt argument som alltid är av längd ett, (charlista), därför blir den i vårt fall konstant. Map är linjär mot listans längd. Theta(n).

Funktionerna cocatinate och build tar konstant tid och därför samlar vi dessa under en gemensam konstant K.

## compress'

compress' tillämpar ziv-lempel algoritmen för att komprimerar listor av enskilda tecken till heltalslistor av förhoppningsvis mindre storlek.
n = antalet tecken i listan.
I varje klausul utförs funktioner med konstant tidskomplexitet samt ett rekursivt anrop av compress' där n minskar med ett, (n-1).

**Öppen form:**

$$T_{compress'}(n) = \begin{cases} t_{update} + t_{cons} + t_{substring} + t_{cons\text{-}tomlista} & \text{if } n =< 1 \\ T(n-1) + K & \text{if } n = 1 \end{cases}$$

**Stängd form:**
den stängda formeln fås genom Theorem 1.

$T_{compress'}(n) = \Theta(n)$

## compress
compress är beroende på tidsåtgången hos compress' samt tidsåtgången för map och båda dessa är linjära mot storleken på listan. 2*T(n) = Theta (n)

## decompress'
decompress' - rekursion för att avgöra vilka tecken som skall ersätta heltalselementen i indata genom uppslag i vektorn och ziv-lempel algoritmen. Implementationen av algoritmen förutsätter att det första elementet motsvarar ett enskilt ASCII-tecken. Funktionen hanterar därefter återstående element ett i taget.

Worst-case för decompress' uppstår när funktionen anropas med en heltalslista där heltalen motsvarar kedjor av enbart ett sorts tecken. Det får konsekvensen att längden för teckenkombinationerna som läggs in i ordboken ökar för varje element. Med en maxlängd för varje heltalskod blir körtiden linjär mot längden av antal element och linjär mot antal tecken som motsvaras av varje enskilt element. Om vi antar att listan som skickas in är oändligt lång så kommer alla 4048 koder i ordboken att användas. Detta innebär att koderna i vektorn kommer att öka i storlek/längd och bli större vid varje ny input. Och eftersom vi använder många ”rev” och ”@” så påverkas tidskomplexiteten mycket av detta. Vi antar därför att antalet element i vår ordbok (vektor) kommer bli konstant då vi aldrig kommer överstiga 4048 element. Det leder till att alla operationer som vi kommit fram till tar linjär tid mot sina argument kommer bli konstanta. Detta får som följd att decompress blir linjär mot antalet element i listan.

Bindningarna som skapas i varje anrop av decompress' genomförs i kontant tid samt alla jämförelser. I respektive rekursivt anrop görs en vektor uppdatering där sammanslagning sker med hjälp av append. Append tar linjär tid mot sitt vänstra argument som i worst-case ökar med ett för varje anrop. List.take körs i konstant tid eftersom vi endast hämtar ett element ur listan. Rev av det avkodade elementet tar linjär tid mot längden på koden, append tar också linjär tid mot samma längd.

Vid rekursion utöver basfallet så har vi följande tidsåtgång:
$T_{bindningar}(1) + T_{jämförelser}(1) + T_{append}(n) + T_{list.take}(1) + T_{rev(}n) + T_{append}(n) =$

Vi benämner funktionerna som tar konstant tid som en enda konstant, K.

Bevis för vårt antagande:
Vi kallar antalet element i listan L för n och detta ger:

$T_{decompress'}(n) = T_{decompress'}(n-1) + T(@) + T(rev) + K$
Vårt worst-case är alltså när samtliga koder är av maximal längd (4048) vilket gör att T(@) och T(rev) blir ”konstanta” enligt vårt tidigare antagande.

Detta ger oss:
$T_{decompress'}(n) = T_{decompress'}(n-1) + 2*\Theta(1)$

Vilket enligt theorem 1 ger att $T_{decompress'}(n) = \Theta(n)$

## decompress

Tidskomplexiteten i det värsta fallet är för decompress helt beroende på tidsåtgången hos decompress' och decompress delar därför även dess tidskomplexitet.