Laboration 2
Programspråk
5DV086
VT-14

Emil Palm (c11epm@cs.umu.se)

Handledare
Jan-Erik Moström
Petter Ericson

6 mars 2014

Innehåll

1 Inledning

Uppgiften var att implementera en bakvändT9-funktion. Utifrån ett givet meddelande skall den kortaste knapptyckningskombinationen för att skapa meddelandet räknas ut. Laborationsspecifikationen finns att tillgå här: `www8.cs.umu.se/kurser/5dv086/VT14/xlab2.html`

2 Åtkomst användarhandledning

Källkoden återfinns på datavetenskaps datorer under katalogen
`/home/c11/c11epm/5DV086/lab2/`

2.1 Användning

Filen laddas in genom kommandot `:l T9.hs` i ghci. Programmet har två funktioner som användaren kan använda sig av. calculate och calcMult

2.1.1 calculate

Funktionen `calculate` används sedan för att räkna ut knapptryckningskombinationen för ett meddelande.

Användning: `calculate <dictionary> <message>` Där `dicitionary` är vilken ordlista man vill använda, `dictionary` för att använda den givna ordlistan, och `message` skall innehålla meddelandet man vill beräkna.

2.1.2 calcMult

Funktionen `calcMult` kan användas för att översätta flera meddelanden vid ett anrop.

Användning: `calcMult <dictionary> <messages>` I detta anrop fungerar `<dictionary>` och `<messages>` likt anropet för calculate. Skillnaden är bara att `messages` skall innehålla en lista av meddelanden.

3 Algoritmbeskrivning

3.1 Hitta rätt knapptryckningskombination

Målet med Algoritmen är att hitta den kortaste knapptryckningskombinationen för ett meddelande.

1. Tag meddelandet och dela upp det i en lista av ord
2. För varje ord i listan, räkna ut förjande:
   (a) Ordets korstaste knappkombination, se 3.1.1
   (b) Finns ordet med i ordlistan, om inte, retunera ett fel
3. Sätt ihop alla delsträngar med ett mellanrumstecken, i detta fall knapptryckning ’0’

3.1.1 Ett ords knappkombination

Ett ord kan skrivas på flera sätt. I den givna ordlistan kan tillexempel ordet 'THOSE' ha kombinationerna 8^^^, 84^^, 842^ eller 8427 som kortaste kombination.

Denna lösning kommer att räkna hur många bokstäver som skrivs, samt hur många ^ som används. Detta görs rekursivt då först en knapptryckning av en bokstav görs för att sedan beräkna hur många ^ som behövs för att hitta rätt ord. Båda dessa antal läggs sedan ihop för att få det totala antalet knapptryckningar.

I nästa steg jämförs detta tal mot då två knapptryckningar och sedan beräknar antalet ^ för att hitta rätt ord. Detta fortsätter sedan nedåt i rekursionen och det minsta antalet totala knapptryckningar väljs.

Kombinationen som väljs retuneras sedan tillbaka som en tupel uppbyggd på följande sätt:
`(<knapptryckningar för bokstäver>, <knapptryckningar för att hitta rätt ord i förslagslista>, <lista av vilka knapptryckningar>`

Exempel: För ordet 'THOSE' kommer denna lösning retunera tupeln (4,0,[8,4,2,7]). Dvs ordet 'THOSE' har korstaste kombinationen fyra siffror samt noll ^ och siffrorna som bygger upp ordet är [8,4,2,7].

Användningen av ^ finns förklarad i laborationsspecifikationen som finns länkad i 2.

4 Lösningens begränsningar

Lösningen är inte implementerad med någon tanke på effektivitet eller hårdvarukrav såsom exempelvis lite RAM-minne, då lösningen använder sin av långa och djupa rekursioner kan det bli resurskrävande för längre meddelanden och större ordlistor. Om användaren vill använda funktionen `calcMult` som tar en lista av meddelanden så märks det att det tar lite tid att lösa uppgiften.

5 Testkörningar

5.1 calculate

I testkörningen testas implementationen mot referensmeddelandet och dess översättning. Som kan ses nedan så ges inte exakt samma lösning, men längden på lösningen är lika lång som referensen. Dvs, de ger tillbaka samma antal knapptryckningar.

Meddelande tolv plockas ut ur meddelandelistan och skrivs ut för att visa att det är referensmeddelandet.
Funktionen calculate används sedan för att räknaut knapptryckningskombinationen.
Längden för kombinationen blir 45 knapptryckningar.
Referensmeddelandets knapptryckningskombination skrivs ut och längden på denna beräknas.
De båda längderna stämmer överens!

```
*Main> let message = messages !! 11
*Main> message
THOSE WHO DO NOT HAVE GOALS ARE DOOMED TO WORK FOR THOSE WHO DO
*Main> calculate dictionary message
"8467094603060420460270366080967 03^08467094603"
*Main> length $ calculate dictionary message
45
*Main> facit
"846^094^030604^04602^036608096703^0846^094^03"
*Main> length facit
45
```

5.2 calcMult

OBS, extra radbrytningar är insatta i exemplet för att öka läsligheten
De tre första meddelandena tas ut från meddelandelistan och skrivs sedan ut.
calcMult används sedan tillsammans med den givna ordlistan med de tre första meddelandena.
Resultatet retuneras som tupler på förlande sätt:
`(<knapptryckningskombination>, <meddelande>)`

```
*T9> let multMess = [(messages!!0)]++[(messages!!1)]++[(messages!!2)]

*T9> multMess
[THERE IS NO SUBSTITUTE FOR HARD WORK,
THE SHORTEST WAY TO DO MANY THINGS IS TO DO A SINGLE THING AT A TIME,
UNDERTAKE NOT WHAT YOU CANNOT PERFORM BUT BE CAREFUL TO KEEP YOUR PROMISE]

*T9> calcMult dictionary multMess
[(843704066^^^0782703^042730967,
THERE IS NO SUBSTITUTE FOR HARD WORK),
(840740920803062690844040803020746408 44^028020846,
THE SHORTEST WAY TO DO MANY THINGS IS TO DO A SINGLE THING AT A TIME),
(8630609409022607370288023022730805309^0776,
```

```
UNDERTAKE NOT WHAT YOU CANNOT PERFORM BUT BE CAREFUL TO KEEP YOUR PROMISE)]
```

5.3 Ord som inte finns med i ordlistan

Ord som inte finns med i en ordlista skall skrivas ut som ett ? istället.

```
*T9> calculate dictionary SOME OF THESE WORDS ARE NOT IN THE DICTIONARY"
?0630?0?02706046^^0840?
```

6 Diskussion

Min implemantation av denna T9-funktion skall möta vissa grundkrav och restriktioner. Dessa finns att se i laborationsspecifikationen länkad i 2. Min implementation stöjder och följer alla dessa krav, detta kan ses i Testkörningar 5.

Som jag skriver i Lösningens begränsningar 4 så har jag inte implementerat min lösning utifrån några hårdvaru- eller tidskomplexitetskrav. Detta gör att programmet kan jobba väldig långsamt vid stora mängder data. Jag valde att implementera programmet med hjälp av rekursiva funktioner iställer för exempelvis foldl som skulle ha kunnat används, för att jag känner att jag vill ha en hundra procentig koll på vad som händer i varje steg. Och det känner jag att fick när jag löste delstegen med rekusion