Oscar Azrak
2000-09-12
DD1320 Tillämpad datalogi - KTH HT 2020

Sammanfattning

I denna labb undersöktes skillnaden mellan SHA-1 hashfunktion och en självgjord hashfunktion. Undersökningen evaluerades med hjälp av ett program som undersökte tre aspekter, skillnaden i tid och skillnaden i antal kollisioner för tre olika typer av indata samt skillnaden på kollisioner om indatan ändras. De tre olika typer av indata var en textfil med data om en miljon stycken låtar där varje rad lästes in och hashades, den andra typen av indata var en textfil som innehöll cirka 370000 rader med engelska ord och den sista indatan var en textfil som innehöll alla siffror från 1 till 215000. Testerna visade att SHA-1 funktionen var både snabbast och hade minst antal kollisioner vilket var noll för varje test. Testet visade även att antal krockar minskade då indatan ändrades.

1 Uppgiftsbeskrivning

Din uppgift är att göra en jämförelse mellan *två olika hashfunktioner* med avseende på *tre relevanta aspekter*.

2 Metod

De aspekter som undersöks är tiden att använda respektive hashfunktion, antal kollisioner respektive hashfunktion ger samt vad som händer om datan som hashas förändras lite. Jag har valt att undersöka dessa aspekter då de är relevanta för datasäkerhet. Tiden det tar att hasha olika längder av strängar är viktigt att veta då man möjligtvis vill använda en så snabb som möjlig funktion beroende på vad för typ av arbete man gör. Antal kollisioner är nog en av de viktigaste aspekterna, ifall man tar in unika strängar, vad är sannolikheten att dessa ger samma hashning, vilket är viktigt eftersom detta undersöker om hashningen också är unik. Sista aspekten jag valt att undersöka är om strängarna ändras, hur det påverkar tiden samt kollisionerna, det som är viktigt för datasäkerhet är hur nästan likadana strängar påverkar hashningen, ger de unika hashningar eller påverkas hashningen ej?

Modulerna hashlib och timeit importerades för att kunna få tillgång till SHA-1 och för att kunna ta tid på funktionerna.
Både SHA-1 och min hashfunktion fick tre funktioner var i programmet, där funktionerna är i sig lika men kör för SHA-1 respektive min hashfunktion. Dessa funktioner returnerar ett hashvärde, kör alla rader i listan och den sista funktionen räknar antalet kollisioner. Sedan skapades två funktioner för att läsa in textfilerna. Den ena läser in varje rad och lägger in varje rad i en lista och den andra gör detsamma men den tar bort frekventa tecken som kanske påverkade antalet kollisioner.

3 Resultat

Tabell 3.1: Låtdata resultat för 370099, 1 000 000 och 999 999 st indata

För låtdata			
Antal element (st)	370099	1000000	999999
SHA-1 tid (s)	0,47646	1,20884	1,30352
Min Hashfunktions tid (s)	8,65037	23,81896	22,66748
SHA-1 Kollisioner (st)	0	0	0
Min Hashfunktions kollisioner (st)	136089	997456	368227

Tabell 3.2: Nummer resultat

För nummer	
Antal element (st)	215000
SHA-1 tid (s)	0,22897
Min Hashfunktions tid (s)	0,44158
SHA-1 Kollisioner (st)	0
Min Hashfunktions kollisioner (st)	214569

Tabell 3.3: Engelska Ordlista resultat

För engelska ord	
Antal element (st)	370099
SHA-1 tid (s)	0,39343
Min Hashfunktions tid (s)	1,1743
SHA-1 Kollisioner (st)	0
Min Hashfunktions kollisioner (st)	136302

Tabell 3.4: Låtlista med modifierad indata resultat för 370099 respektive 1 000 000 st indata

För låtdata (modifierad)		
Antal element (st)	370099	1000000
SHA-1 tid (s)	0,41616	1,31061
Min Hashfunktions tid (s)	6,23093	16,44707
SHA-1 Kollisioner (st)	0	0
Min Hashfunktions kollisioner (st)	135923	987350

4 Analys

Som vi ser i tabellerna i resultat så är SHA-1 betydligt snabbare än min hashfunktion samtidigt som den ger noll kollisioner vilket var förväntat av SHA-1 då enligt Googles Security Blog kan kollisioner ske om man genererar cirka 2^{80} stycken hashningar.

SHA-1 ska ta mellan 500-900 ms per 1 000 000 strängar beroende på hur lång strängen är. (automationrhapsody.com, 2018) Däremot står det ej hur lång den längsta strängen är, de strängar jag har jämfört har längderna från ett element långt till 439 långt. Den längsta strängens längd ficks fram med hjälp av max() funktionen.

Att min hashfunktion får så många kollisioner var lite märkligt dock. Något som märktes under undersökningen var att vilken siffra man gör modulo med har stor betydelse, som man ser i tabell 3.1 kolumn två så görs modulo med antal elementen och med 1 000 000 st element fås 997 456 hashningar som krockar. Detta var lite märkligt men om man istället ändrar den totala hashningar till 999 999 sänktes antalet krockar till 368 227, vilket fortfarande är många och orsaken till det kan ej förklaras. Jag provade att skriva en ny hashfunktion för att jämföra med min, för denna hashfunktion gavs noll stycken kollisioner trots att de ger samma hashvärde vilket jag också tyckte var märkligt så jag fortsatte med den hashfunktionen jag använt i tidigare laboration och tester. Ett sido-experiment jag gjort var att hitta ett par strängar som krockar, då fick jag reda på att om jag ändrar eller till och med tar bort tecken från början av strängen påverkar det ej hashvärdet, däremot om jag ändrar på de 3-4 tecken i slutet av strängen så fås ett nytt hashvärdet, från detta valdes att undersöka vad som händer om frekventa tecken tas bort från strängarna vilket är resultatet i tabell 3.4.
Skillnaden på resultatet från tabell 3.4 och 3.1 är att jag tagit bort de tecken som jag tyckt är frekventa i listan med låtdata, dessa tecken var: ”TR”, ”SO”, ”\n”, ”12”, ”<SEP>”. Detta gjordes med 5 raders kod:

```
    for row in file:
        row = row.replace("TR", "")
        row = row.replace("SO", "")
        row = row.replace("\n", "")
        row = row.replace("12", "")
        row = row.replace("<SEP>", "")
        listan.append(row)
```

Figur 4.1

Kollisionerna för 1 000 000 element sänktes från 997456 till 987350 stycken vilket medför att en möjlig anledning till mängden kollisioner kommer från att strängarna från min indata kan ha vart lika.