Algoritmer

Emil Wiklund*

Luleå tekniska universitet
971 87 Luleå, Sverige

7 oktober 2020

Sammanfattning

Delen från september 1983 beskriver den vardagliga verkan som design av algoritmer kan ha på programmerare. Från en algoritmisk synvinkel på problemet kan man se att dessa kan göra ett program enklare att förstå och skriva. I denna del har vi studerat ett bidrag inom ämnet sofistikerade algotim metoder, dessa kan ibland leda till en stor ökning prestandamässigt. Denna delen är framtagen kring ett mindre problem med betoning på de algoritmer som ska lösa dessa problem samt teknikerna på hur man designar dom. Några av algoritmerna är lite mer komplicerade men dessa motverkas.

Den första algoritmen som vi kommer att gå igenom tar 39 dagar att lösa när problemet har storleken 10 000, den sista algoritmen löser samma problem men på gör detta på en tredjedel av en sekund.

1 Introduktion

Här kan man skriva massor om uppgiften, dess bakgrund, varför det är viktiga att lösa det osv men för oss räcker det med att introducera den. Ett bra sätt att skriva på är att sen behandla varje problem i egna avsnitt, där man inleder med att beskriva problemet noga innan man visar hur man kan lösa det.

Faktum är att det blir lite krystat att skriva en liten rapport om hur man löser en uppgift som denna men vi gör det som en övning i LaTeX.

2 Problemet och ett simpelt program

Problemet uppstod i en endimensionellt mönsterigenkänning. Historien beskrivs senare. Invärdet är en vektor X av N naturliga tal. Dess utvärde är den maximala summan

*email: emiwik-9@student.ltu.se

inom en delad sub vektor av invärdet. Exempelvis om invärdet är 3 och 4 ur listan med nummer:

$$[31, -41, 59, 26, -53, 58, 97, -93, -23, 84]$$

kommer programmet att returnera summan av $X[3...7]$, dvs 187. Problemet är enkelt när alla nummer är positiva, den största sub vektorn är input vektorn. Problemetuppstår när siffror är negativa. Skulle vi inkludera ett negativt nummer skulle man kunna hoppas på att ett positivt nummer skulle kunna ta ut det negativa eller med andra ord kompensera. Om alla inputs skulle vara negativa nummer skulle summan av sub vektorn vara noll vilket också ger den totala summan 0.

Det programmet man vill använda för detta är ett simpelt program som kör for each på paren av heltalen L och U där $1 \leq L \leq U \leq N$. Detta beräknar summan av $x[L...U]$ och gör en kontroll om summan är större än den summan som är störst inuläget. Koden som visas i Algorithm 1 är kort och enkel att förstå. Däremot så är den långsam. Koden tar kring 1 timme att köra om N är 1000 och 39 dagar om N är 10000. Detaljerna angående tider går vi igenom senare.

Vi får en annan känsla för algoritmer och hur effektiva man skulle kunna göra dom. Detta kan vi göra genom att använda oss utav $\backslash big - oh$", beteckningen[1].

Uttrycken i den yttersta snurran exekveras exakt N-gånger och det i mittersta snurran exekveras som mest N-gånger i varje exekvering av den yttersta snurran.Multiplicera dessa två faktorer inuti den mittersta snurran visar att dessa fyrarader exekveras $O(N^2)$ antal gånger. Snurran i dessa fyra rader exekveras aldrig mer än N-gånger. Detta ger kostnad på algoritmen lika med $O(N)$. Om kostnadenmultipliceras per antalet gånger som den innersta snurran körs får vi kostnadenför hela programmet och som även är proportionerligt till N^2. Så detta kan vi kalla för en kvadratisk algoritm.

Dessa enkla steg illustrerar tekniken av $\backslash big - oh$", analys av tiden som det taratt köra och andra fördelar men även nackdelar. Den största nackdelen med denna är att vi fortfarande inte vet exakt hur lång tid det tar programmet för en särskild input. Vi vet bara att antalet steg det tar att exekveras är $O(N^3)$. Två stycken fördelar med denna metod är att den ofta kompenserar för sina nackdelar. $\backslash Big-oh$" analyser är ofta användbara när man utföra sådant som vibeskrev tidigare. Sedan är den ungefärliga tiden ofta tillräcklig för att utföra den uträckning som används för att bestämma om ett program är tillräckligt effektivt för den givna uppgiften.
De kommande delar använder sig av ungefärliga exekveringstiden som ett sätt att mäta hur effektivt ett program är.

[1] $O(N^2)$ kan ses som proportionerligt till N^2; både $15N^2 + 100N$ och $N^2/2 - 10$ är $O(N^2)$. Sen så $f(N) = O(g(N))$ betyder att $f(N) < cg(N)$ för en konstant c och tillräckligt stora värden av N. En formell definition av beteckningen kan man hitta i de flesta böcker om design av algoritmer eller diskret matematik.

```
61    MaxSoFar:= 0.0
62    sum L := 1 to N do
63    for U := 0.0
64          sum := sum + X[I]
65          /* Sum now contains the
66                   sum   of x[L..U] */
67          MaxSoFar := max(MaxSoFar, sum)
```

68 3 Nästa problem

69 4 Och ännu nästa problem...

70 5 Diskussion [och slutsatser]

71 Sammanfatta vad som avhandlats i dokumentet och sätt det i sitt sammanhang.

72 Referenser

73 [1] Michel Goossens, Frank Mittelbach, and Alexander Samarin. *The LaTeX Companion*.
74 Addison-Wesley, Reading, Massachusetts, 1993.

75 [2] Albert Einstein. *Zur Elektrodynamik bewegter Körper*. (German) [*On the electrody-*
76 *namics of moving bodies*]. Annalen der Physik, 322(10):891–921, 1905.