Algoritm designtekniker

Hamid Ehsani*

Luleå tekniska universitet
971 87 Luleå, Sverige

11 oktober 2019

Sammanfattning

Nuförtiden är vi beroende av algoritm t.ex. När du öppnar en app i din mobil eller öppnar du din dator algoritmen bestämmer vad som ska visas på skärmen, Men vad är algoritm och hur den fungerar? I denna uppgiften kommer jag att berätta vad algoritm är och hur den fungerar T.ex. hur du beräknar antal personer i ett rum och även hur en programmerare blir påverkat av algoritm på "vardagliga".

1 introduktion:

Kortfattar kan man beskriva algoritmen så här, en instruktion som visar hur man löser ett problem steg för steg med hjälp av datorteknik eller med huvudräkning. Med hjälp av algoritmisk vy kan vi lösa konkreta problemet på ett enkelt sätt. Vi tar ett exempel och den här exempel tar 39 dagar för den första algoritmen att lösa ett problem i storlek 10000 och den slutligen algoritmen med samma problem löste det problemet på en tredjedel av sekund.

31	-41	59	26	-53	58	97	-93	-23	84

Alltså algoritmen handlar om att hur snabbt proceduren går att genomföra och ibland finns det många lösning till ett problem, men den som går snabbaste vill man ha.

2 Uppgiften och lösningen

2.1 Problem:

Till exempel om man skulle räkna summan av index $3-7$ på detta exempel [1] från programming pearls av Jon Bentley Vi antar att X är ingångs vektor och N är reella tal

*email: `hamqur-9@student.ltu.se`

som står för, utgången är den maximala summan som finns i någon sammanhängande subvektor av ingång. Man kan skriva så här X [3..7] eller kan man skriva som 187 som är summan av dem. Problemet löser sig enkelt när alla värdena är positiva, men när vi har från båda positiva och negativa då hoppas vi på att de positiva siffror tar ut den negativa siffror. Problemet uppstår när alla värdena blir negativa då den maximalla summan är den tomma vektor som har summan noll. Den är tydliga programmet för den här typen av problem är enkelt t.ex. för varje par heltal L och U där $1 \leq L \leq U \leq N$. Vi kan beräkna summan av $N\left[L...U\right]$ där n är antal gånger, X och U är värden som man räknar summan av. För att kunna veta om svaret är rätt så kontrollerar man om summan av den är större ön den maximala summan hittills. Eftersom den jämför varje delmängd därför svaret är rätt. Den metoden är väldigt enkelt, lätt att förstå och men tyvärr den har en nackdel som är långsam och tar ganska mycket tid.

2.2 Lösning:

2.2.1 Stora o notationen

För att kunna lösa det problemet så behöver vi veta vad stora o notationen *(big o notation)*. Stora O notationen sätter en övre gräns för en funktion för att hur dess tidsåtgång skalas i takt med att datamängden ökar. Till exempel om man läser indata från en fil. Denna funktioner behöver läsa varje rad en gång och det spelar ingen roll hur stor datamängd är. Vi antar att denna funktion är $O\left(n\right)$ Om man läser varje rad så tar det inte jättelång tid. Men tänk om man vill även sortera denna filen då tar det väldigt lång tid för att då varje inläst dataelement måste relateras i andra element. Men om vi använder oss av en naiv sorteringsalgorim och jämför alltså n stycken element med vardera $n-1$ andra element då får vi den här ekvationen $O\left(n^2\right)$. Den ekvationen betyder att algoritmens tidsåtgång oavsett datamängd så ska de ge resultaten på samma tidslängd. Den är en bättre sorteringsalgorim som ger mycket bättre resultat.

2.2.2 Första och andra kvadratiska algoritmen

Den första påstående i den yttersta slingan exekveras exakt N gånger och de i den mittersta slingan är körs högst N gånger i varje utförande av yttre slinga. Att multiplicera dessa två faktorer i N visar att de fyra linjerna i mellerslingan är utförs $O\left(N2\right)$ gånger. Slingan i de fyra linjerna är utfördes aldrig mer än N gånger, så kostnaden är $O\left(N\right)$. Att multiplicera kostnaden per inre slinga gånger antalet avrättningar visar att kostnaden för hela programmet är proportionell mot N-kuben, så vi kommer att referera till detta som kubik algoritm.

Precis som den första kvadratiska algoritmen så använder sig den andra kvadratiska algoritmen utav O(N^2) steg på med N i inmatning. Den kan även räknad ut en exakta tiden det tar att slutföra algoritmen med hjälp av att räkna ut summan med ett redan uträknat nummer av steg. Men de två algoritmerna tar helt olika vägar när dom ska räkna ut summan i konstant tid. Den första kvadratiska algoritmen beräknar summan så snabbt som den gör genom att den lägger märke till att X[$L..U$] har en summa som har en koppling till den summan som beräknades steget innan eller med andra ord X[$L..U-1$]. Den andra kvadratiska algoritmen tar detta sätt men använder den på ett väldigt smart sätt då den har en total eksekverings tid på O(N^2). Så hur gör den detta? Alla påståenden i första loopen utförs exakt N gånger och alla påståenden i den inre loopen körs som mest N gånger varje gång den ytre loopen körs. Denna algoritm är den snabbaste av alla de tre olika som jag har skrivigt om samt den mest pålitliga.

Referenser

[1] Jon Bently *Algorithm design techniques,* Programming Pearls, sidan 865, 1984-09