Jämförelse av algoritmer

Zacharias Brohn*

Luleå tekniska universitet
971 87 Luleå, Sverige

9 oktober 2024

Sammanfattning

Denna rapport kommer att förklara och jämföra olika algoritmer för att undersöka hur de skiljer sig i både utförning och prestanda. Algoritmerna i fråga används för att är en kubisk algoritm som har en komplexitet av $O(n^3)$ med två kvadratiska algoritmer som har en komplexitet av $O(n^2)$.

1 Introduktion

Nu undrar ni säkert vad som menas med $O(n^x)$ och för att förstå det så börjar vi med namnet på uttrycket 'Big-Oh', det används för att beskriva hur många steg en algoritm tar för att nå ett svar i det **värsta fallet**, även kallat komplexitet. Låt oss illustrera detta med hjälp av ett välkänt exempel som också gör det lättare att förstå problemet vi ska undersöka i rapporten:

Telefonboken

Om vi beskriver n som antalet namn i boken och vi börjar leta överst på listan och går sedan stegvis nedför listan tills vi hittar namnet vi vill ha. Det betyder då att i det **värsta fallet** är namnet vi söker längst ner på listan, så antalet steg som behövs för att hitta namnet blir då n, lika många namn som finns i listan, vilket betyder att våran 'algoritm', som hittar namnet vi söker, kan beskrivas med $O(n)$.

*email: zacbro-8@student.ltu.se

2 Problembeskrivning: Maximal delfältssumma

Givet en sekvens av reella tal som representeras av en array X där $X = [x_1, x_2, \ldots, x_N]$, där N är antalet element i arrayen, är målet att hitta den maximala summan av ett sammanhängande delfält. Detta innebär att vi vill identifiera en del av arrayen, $X[L..U]$, där $1 \leq L \leq U \leq N$, för vilken summan

$$S = \sum_{i=L}^{U} x_i$$

är maximal. Problemet är enkelt att lösa när alla tal x_i är positiva, eftersom då är den maximala delfältssumman helt enkelt summan av hela sekvensen:

$$\max S = \sum_{i=1}^{N} x_i.$$

Men utmaningen uppstår när det finns negativa tal i sekvensen. I dessa fall måste vi överväga om vi ska inkludera ett negativt tal x_k i summan, med hopp om att positiva tal på sidorna, x_{k-1} och x_{k+1}, kan kompensera för dess negativa bidrag. För att ytterligare definiera problemet kan vi säga att om alla tal i sekvensen X är negativa, så är den maximala delfältssumman summan av den tomma delfältet, vilket är noll:

$$\max S = 0 \quad (\text{för alla } x_i < 0).$$

En grundläggande metod för att lösa detta problem är att iterera över alla möjliga delfält. För varje par av heltal L och U, där $1 \leq L \leq U \leq N$, beräknar vi summan av delfältet $X[L..U]$ och jämför den med den största summan vi hittills har funnit. Denna metod är dock ineffektiv, med en tidskomplexitet på $O(N^2)$ eller mer, vilket gör den ohållbar för stora datamängder.

Därför behövs mer effektiva algoritmer för att lösa problemet inom rimlig tid. I följande avsnitt kommer vi att utforska dessa algoritmer och deras fördelar.

3 Behovet av effektiva algoritmer

Ett enkelt och uppenbart sätt att lösa problemet är att gå igenom alla möjliga start- och slutpunkter för delfältet. För varje par av heltal L och U, där $1 \leq L \leq U \leq N$, kan vi beräkna summan av $X[L..U]$ och sedan jämföra denna summa med den största summan som vi hittills har funnit. Detta är en enkel och lättförståelig lösning, och pseudokoden för algoritmen är kort och rakt på sak.

Tyvärr har denna metod en allvarlig nackdel: den är långsam. Den beräknar summan för alla möjliga delfält genom att gå igenom alla kombinationer av start- och slutpunkter, vilket ger en tidskomplexitet på $O(n^3)$. För stora datamängder blir denna metod ohållbar, eftersom tiden för beräkningarna växer kraftigt när storleken på arrayn N ökar. Därför behövs mer effektiva algoritmer för att lösa problemet inom rimlig tid.

I kommande avsnitt kommer vi att presentera förbättrade algoritmer som drastiskt minskar antalet nödvändiga beräkningar, och därmed snabbar upp processen. Genom att använda bättre tekniker kan vi reducera tidskomplexiteten till $O(n^2)$ eller till och med $O(n)$, vilket gör det möjligt att lösa problemet även för stora N.

4 Den kubiska algoritmen

Den kubiska algoritmen fungerar genom att iterera över alla möjliga subfält och beräkna summan av elementen i varje subfält. Detta resulterar i en tidskomplexitet på $O(n^3)$.

```
MaxSoFar := 0.0
for L := 1 to N do
    for U := L to N do
        Sum := 0.0
        for I := L to U do
            Sum := Sum + X[I]
        MaxSoFar := max(MaxSoFar, Sum)
```

Denna algoritm är enkel men ineffektiv för stora värden på N, eftersom tre nästlade loopar gör beräkningarna dyra.

5 Första kvadratiska algoritmen

För att förbättra prestanda kan vi reducera en av de tre nästlade looparna. Den första kvadratiska algoritmen beräknar summan för varje subfält genom att successivt addera elementen i arrayen. Detta reducerar tidskomplexiteten till $O(n^2)$.

```
MaxSoFar := 0.0
for L := 1 to N do
    Sum := 0.0
    for U := L to N do
        Sum := Sum + X[U]
        MaxSoFar := max(MaxSoFar, Sum)
```

Här uppdateras summan för varje subfält i den inre loopen utan att behöva använda en tredje loop.

6 Andra kvadratiska algoritmen med kumulativ summa

Den andra kvadratiska algoritmen använder en kumulativ summa-array för att ytterligare optimera beräkningarna. Genom att först beräkna en kumulativ summa för arrayen kan vi beräkna summan för vilket subfält som helst i konstant tid.

```
CumArray[0] := 0.0
for I := 1 to N do
    CumArray[I] := CumArray[I-1] + X[I]
MaxSoFar := 0.0
for L := 1 to N do
    for U := L to N do
        Sum := CumArray[U] - CumArray[L-1]
        MaxSoFar := max(MaxSoFar, Sum)
```

Denna algoritm har fortfarande tidskomplexiteten $O(n^2)$, men genom att använda kumulativ summa kan subfältssummor beräknas mer effektivt.

7 Diskussion och slutsatser

Vi har presenterat tre olika algoritmer för att lösa problemet med maximalt delfältssumma. Den kubiska algoritmen är enkel men ineffektiv för stora datamängder på grund av dess $O(n^3)$ komplexitet. De två kvadratiska algoritmerna förbättrar prestandan markant. Speciellt den andra kvadratiska algoritmen, som använder en kumulativ summa, kan vara användbar i scenarier där subfältssummor behöver beräknas ofta och snabbt.

Referenser

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

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