Jämförelse mellan kvadratiska och kubiska algoritmer för att beräkna maximalt delfältssumma

Författare: Ditt Namn

1 Introduktion

I denna rapport ska vi jämföra tre olika algoritmer för att beräkna maximalt delfältssumma i en array. Den första algoritmen har en tidskomplexitet på $O(n^3)$, medan de två andra är kvadratiska algoritmer med tidskomplexiteten $O(n^2)$. Vi kommer att undersöka hur varje algoritm fungerar och hur de skiljer sig åt.

2 Den kubiska algoritmen

Den kubiska algoritmen fungerar genom att iterera över alla möjliga subfält (delar av arrayen) och beräkna deras summor. Detta görs med tre nästlade loopar, vilket resulterar i en tidskomplexitet på $O(n^3)$. Nedan följer algoritmen:

Algorithm 1 Kubisk algoritm

```
1:  MaxSoFar := 0.0
2:  for L := 1 till N do
3:      for U := L till N do
4:          Sum := 0.0
5:          for I := L till U do
6:              Sum := Sum + X[I]
7:          end for
8:          MaxSoFar := max(MaxSoFar, Sum)
9:      end for
10: end for
```

Som vi ser, beräknas summan för varje subfält genom att uppdatera en lokal variabel **Sum** för varje ny subarray.

3 Den första kvadratiska algoritmen

Den första kvadratiska algoritmen förbättrar den kubiska algoritmen genom att ta bort den innersta loopen. Istället beräknas summan för varje subfält genom att successivt addera elementen i arrayen. Detta gör att vi kan minska antalet operationer och sänka tidskomplexiteten till $O(n^2)$.

Algorithm 2 Första kvadratiska algoritmen

```
1: MaxSoFar := 0.0
2: for L := 1 till N do
3:     Sum := 0.0
4:     for U := L till N do
5:         Sum := Sum + X[U]
6:         MaxSoFar := max(MaxSoFar, Sum)
7:     end for
8: end for
```

Här behöver vi bara två nästlade loopar. I den inre loopen adderas elementen successivt till **Sum**, vilket gör att vi kan beräkna subfältssummorna effektivare.

4 Den andra kvadratiska algoritmen med kumulativ summa

Den andra kvadratiska algoritmen använder en optimeringsteknik med hjälp av en kumulativ summa-array (**CumArray**). Genom att först beräkna den kumulativa summan för varje element kan vi därefter beräkna summan för ett godtyckligt subfält i konstant tid, vilket minskar beräkningskostnaden.

Algorithm 3 Andra kvadratiska algoritmen med kumulativ summa

```
1: CumArray[0] := 0.0
2: for I := 1 till N do
3:     CumArray[I] := CumArray[I-1] + X[I]
4: end for
5: MaxSoFar := 0.0
6: for L := 1 till N do
7:     for U := L till N do
8:         Sum := CumArray[U] - CumArray[L-1]
9:         MaxSoFar := max(MaxSoFar, Sum)
10:    end for
11: end for
```

Här använder vi **CumArray** för att snabbt kunna beräkna summan för ett subfält, vilket gör algoritmen mer effektiv än den kubiska algoritmen.

5 Jämförelse

- **Kubisk algoritm:** Tidskomplexiteten är $O(n^3)$ eftersom vi har tre nästlade loopar. För varje par av index L och U beräknas summan för subfältet genom att iterera över alla element i det.
- **Första kvadratiska algoritmen:** Tidskomplexiteten är $O(n^2)$. Här minskas antalet loopar till två, vilket ger en förbättring jämfört med den kubiska algoritmen.
- **Andra kvadratiska algoritmen:** Tidskomplexiteten är också $O(n^2)$, men den använder en kumulativ summa-array för att snabbare beräkna summorna, vilket kan ge ytterligare prestandaförbättringar i praktiken.

6 Slutsats

Den största skillnaden mellan algoritmerna är deras tidskomplexitet. Medan den kubiska algoritmen är enkel att förstå och implementera, blir den snabbt ineffektiv för stora datasätt. De kvadratiska algoritmerna erbjuder betydande prestandaförbättringar och är mer praktiska för större array-storlekar. Speciellt den andra kvadratiska algoritmen, som använder en kumulativ summa, kan vara fördelaktig i vissa scenarier där vi snabbt behöver beräkna subfältssummor.