Algorithm Design Techniques – Rapport

Viktor Jarl Moe*
Luleå tekniska universitet

9 oktober 2025

Sammanfattning

Denna rapport sammanfattar huvudidéerna i Jon Bentleys artikel *Programming Pearls: Algorithm Design Techniques* (1984). Artikeln handlar om hur man kan lösa ett klassiskt problem inom algoritmdesign – att hitta den del av en lista med tal som har störst summa. Tre olika lösningar presenteras och jämförs utifrån hur effektiva de är. Rapporten förklarar även några centrala begrepp som behövs för att förstå resonemanget.

1 Inledning

Syftet med rapporten är att förklara Bentleys exempel på hur man stegvis kan förbättra en algoritm genom bättre design. Rapporten är skriven för en student på kursen D0015E som inte läst artikeln och därför innehåller förklaringar av viktiga begrepp och idéer.

2 Grundproblemet

Problemet som tas upp i artikeln kallas *maximum subvector problem*. Det handlar om att, givet en lista med tal, hitta den sammanhängande del som ger störst summa. Om alla tal är negativa säger man att den bästa summan är noll (den tomma delsekvensen).

Exempel

Anta listan:

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

Här är den bästa delsekvensen $59, 26, -53, 58, 97$ eftersom deras summa, 187, är den största möjliga.

*email: vikmoe-4@student.ltu.se

3 Begrepp och Big-O i korthet

För att jämföra algoritmer används **Big-O-notationen**. Den visar ungefär hur snabbt tiden växer när listan blir längre. Man bryr sig inte om exakta siffror utan om tillväxten i stort:

$$O(n) < O(n^2) < O(n^3)$$

En linjär algoritm växer långsammare än en kvadratisk, som i sin tur är snabbare än en kubisk.

Några begrepp:

- **Kontiguös delsekvens:** En följd av tal som ligger direkt efter varandra.
- **Prefixsumma:** En lista där varje element är summan av alla tidigare tal.
- **Invariant:** Ett påstående som är sant i varje steg i en algoritm.

4 Den naiva (kubiska) algoritmen

Den första metoden testar alla möjliga start- och slutpunkter och räknar ut summan för varje delsekvens. Sedan väljer den den största.

Den fungerar alltid eftersom alla möjligheter prövas, men den gör mycket onödigt arbete. Därför tar den lång tid när listan blir stor. Man säger att den har tidskomplexitet $O(n^3)$.

5 Första kvadratiska algoritmen

Den andra algoritmen förbättrar den första genom att spara tidigare resultat. I stället för att räkna om hela summan varje gång lägger man bara till nästa tal. På så sätt slipper man upprepa samma beräkningar.

Detta gör algoritmen mycket snabbare. Den går på ungefär $O(n^2)$ tid.

6 Andra kvadratiska algoritmen

Den tredje algoritmen använder prefixsummor. Genom att först räkna ut en hjälplista med summor kan man sedan få ut varje delsumma genom att ta skillnaden mellan två värden i prefixlistan. Detta gör varje beräkning mycket snabb.

Även denna metod går i $O(n^2)$ tid, men kräver lite extra minne.

7 Diskussion och sammanfattning

Alla tre algoritmer löser samma problem men på olika sätt. Den kubiska är lätt att förstå men långsam. De kvadratiska visar hur man genom att återanvända tidigare resultat eller förbereda data kan spara mycket tid.

Algoritm	Idé	Komplexitet
Kubisk	Testar alla möjligheter	$O(n^3)$
Första kvadratiska	Återanvänder tidigare summa	$O(n^2)$
Andra kvadratiska	Använder prefixsummor	$O(n^2)$

Sammanfattning: Små förändringar i hur man tänker kan ge stor skillnad i effektivitet. Bentley visar tydligt att förståelse för algoritmdesign kan göra program både snabbare och enklare.

Referenser

Jon Bentley (1984). "Programming Pearls: Algorithm Design Techniques". *Communications of the ACM*, 27(9). `https://doi.org/10.1145/358234.381162`