Amortiseret analyse

Søren Dahlgaard

Datalogisk Institut, Københavns Universitet

Diskret Matematik og Algoritmer, 2015
Baseret på CLRS kapitel 17

Motivation

Når vi udfører mange operationer kan worst-case af en enkelt operation være misvisende.

Vi ønsker en måde at sige n operationer bruger samlet højest $T(n)$ tid.

Alternativt: Når vi udfører n operationer tager hver højest $T'(n)$ tid.

Simpelt eksempel

At tælle dumt

Forestil os at vi har en sorteret tabel A med n tal mellem 1 og m.

Vi ønsker at køre en funktion $f(A[i], A[i+1])$ for alle $1 \leq i \leq n-1$, hvor $f(x, y)$ tager tid $O(y-x)$.

Hvis $A[i] = 1$ og $A[i+1] = m$ får vi at $f(A[i], A[i+1])$ tager $O(m)$ tid. Altså tager alle operationerne til sammen $O(mn)$.

Det er rigtigt, men vores grænse er alt for høj.

Summen af forskellene kan højest være m, da tallene er i sorteret rækkefølge! Altså bruger vi kun $O(m)$ tid!

Eksempel: Multi-pop

Vi har en stack S med de sædvanlige operationer (pop og push).

Derudover kan vi også $multipop(S, k)$, som fjerner op til k elementer fra vores stack S.

Spørgsmål: Hvad er køretiden af n pop, push, og multipop operationer?

Svar: multipop kan tage $O(n)$ tid, så det hele er $O(n^2)$.

Bedre svar: Hvert element, som er blevet pushed kan højest poppes én gang. Højest n elementer pushes, så de n operationer tager højest $O(n)$ tid!

Vi siger at den amortiserede køretid for push, pop og multipop er $O(1)$.

Eksempel: En binær tæller

Vi ønsker at tælle fra 1 til n og vide hvor mange bits vi flipper i alt.

Svar: Simpelt.. Der er højest $\lceil \log_2 n \rceil$ bits der kan flippes hver gang, så $O(n \log n)$ i alt!

Kan vi give en bedre grænse?

Idé: Nogle gange kan $\lceil \log_2 n \rceil$ bits flippes, men ofte bliver der kun flippet én. Kan vi beskrive hvor ofte i bits flippes for hver i?

Eksempel: En binær tæller

Eksempel fortsat

Lad os kigge på de første tal i binær:

Tal	Binær
0	0
1	1
2	10
3	11
4	100
5	101
6	110
7	111
8	1000
9	1001

Eksempel: En binær tæller

Eksempel fortsat

Observation: Den mindst signifikante bit flippes hver gang. Den næstmindst signifikante flippes hver anden gang, den tredjemindst signifikante bit flippes hver fjerde gang, osv.

Altså har vi i alt $\approx n + n/2 + n/4 + ... + 1 \leq 2n$ flips!

Hvis vi vil tælle fra 1 til n flipper vi altså højest $O(n)$ bits. Dvs. at den amortiserede køretid pr. inkrement er $O(1)$.

Revisorens metode

En anden måde at regne på

Det er ikke altid, at vi nemt kan regne den totale køretid ud som i de forrige eksempler!

Idé: Giv hver operation en amortiseret pris $\hat{c}_i$, så summen af de udførte operationer $\sum_{i=1}^{k} \hat{c}_i$ altid er mindst lige så stor som den reelle pris $\sum_{i=1}^{k} c_i$ for alle k.

Mål: De amortiserede priser er nemmere at regne på, så vi kan give en grænse.

Intuition: Når vi udfører den i'te operation betaler vi $\hat{c}_i$ ind på vores "konto" og hæver c_i. Hvis der altid er ≥ 0 penge på kontoen er vi glade.

Revisorens metode: Eksempel

Multipop igen

Den reelle pris for stakoperationerne fra før er:

Operation	**Pris**
Pop	1
Push	1
Multipop	$\min(k, \lvert S \rvert)$

Vi kan bruge den amortiserede pris:

Operation	**Amortiseret pris**
Pop	0
Push	2
Multipop	0

Revisorens metode: Eksempel

Multipop fortsat

Husk priserne:

Operation	**Amortiseret pris**
Pop	0
Push	2
Multipop	0

Hvorfor virker det? Hver gang vi pusher et element betaler vi 1 for at pushe det samt 1 for at poppe det senere. Derfor kan pop være gratis amortiseret.

Ud fra priserne i tabellen herover er det nemt at se, at alle operationer er $O(1)$ amortiseret. Nemmere end hvis vi skulle analysere den samlede pris!