Institut for Matematik og Datalogi
Syddansk Universitet, Odense

2. april 2020

DM507 — Algoritmer og Datastrukturer

Eksaminatorie-timer uge 14, Forår 2020

Instruktorerne for DM507

Indhold

1 Opgave 1 - Cormen et al. opgave 4.5-1

Vi kan se, at alle ligningerne har $a = 2$, $b = 4$ og $\alpha = \log_b a = \log_4 2 = 0.5$. Den eneste forskel på ligningerne er $f(n)$, som ender med at bestemme, hvilken case vi er i.

a)

Her er $f(n) = 1$. Vi er derfor i **case 1**, da hvis vi f.eks. vælger $\epsilon = 0.1$, så er

$$f(n) = 1 = O(n^{\alpha-\epsilon}) = O(n^{0.5-0.1}) = O(n^{0.4})$$

Vi har derfor $T(n) = \Theta(n^\alpha) = \Theta(n^{0.5}) = \Theta(\sqrt{n})$

b)

Her er $f(n) = \sqrt{n}$. Vi er derfor i **case 2**, da

$$f(n) = \sqrt{n} = \Theta(n^\alpha) = \Theta(n^{0.5}) = \Theta(\sqrt{n})$$

Vi har derfor $T(n) = \Theta(n^\alpha \cdot \log n) = \Theta(n^{0.5} \cdot \log n) = \Theta(\sqrt{n} \cdot \log n)$

c)

Her er $f(n) = n$. Vi er derfor i **case 3**, da hvis vi f.eks. vælger $\epsilon = 0.1$, så er

$$f(n) = n = \Omega(n^{\alpha+\epsilon}) = \Omega(n^{0.5+0.1}) = \Omega(n^{0.6})$$

I case 3 skal vi også tjekke, at der findes et $c < 1$ og n_0, som opfylder at $a \cdot f\left(\frac{n}{b}\right) \leq c \cdot f(n)$ når $n \geq n_0$. Dvs. vi skal finde c og n_0 således at

$$\begin{aligned} a \cdot f\left(\frac{n}{b}\right) &\leq c \cdot f(n) \Leftrightarrow \\ 2 \cdot f\left(\frac{n}{4}\right) &\leq c \cdot f(n) \Leftrightarrow \\ 2 \cdot \frac{n}{4} &\leq c \cdot n \Leftrightarrow \\ \frac{1}{2} \cdot n &\leq c \cdot n \end{aligned}$$

Som er sandt, hvis vi f.eks. vælger $c = \frac{1}{2}$ og $n_0 = 1$.

Dvs. den ekstra betingelse er opfyldt, og vi har derfor $T(n) = \Theta(f(n)) = \Theta(n)$

d)

Her er $f(n) = n^2$. Vi er derfor i **case 3**, da hvis vi f.eks. vælger $\epsilon = 0.1$, så er

$$f(n) = n^2 = \Omega(n^{\alpha+\epsilon}) = \Omega(n^{0.5+0.1}) = \Omega(n^{0.6})$$

I case 3 skal vi som før også finde c og n_0 således at

$$\begin{aligned} a \cdot f\left(\frac{n}{b}\right) &\leq c \cdot f(n) \Leftrightarrow \\ 2 \cdot f\left(\frac{n}{4}\right) &\leq c \cdot f(n) \Leftrightarrow \\ 2 \cdot \left(\frac{n}{4}\right)^2 &\leq c \cdot n^2 \Leftrightarrow \\ 2 \cdot \frac{n^2}{16} &\leq c \cdot n^2 \Leftrightarrow \\ \frac{1}{8} \cdot n^2 &\leq c \cdot n^2 \end{aligned}$$

Som er sandt, hvis vi f.eks. vælger $c = \frac{1}{8}$ og $n_0 = 1$.

Dvs. den ekstra betingelse er opfyldt, og vi har derfor $T(n) = \Theta(f(n)) = \Theta(n^2)$

2 Opgave 2 - Eksamen juni 2013, opgave 1

Vi skal angive løsningen for følgende rekursionsligninger:

- $T(n) = 8 \cdot T\left(\frac{n}{3}\right) + n^2$
- $T(n) = 9 \cdot T\left(\frac{n}{3}\right) + n^2$
- $T(n) = 10 \cdot T\left(\frac{n}{3}\right) + n^2$

Vi kan løse dette med Master Theorem (side 94 [1]).

i) $T(n) = 8 \cdot T\left(\frac{n}{3}\right) + n^2$

Siden rekursionsligningen er på den korrekte form kan variabler nemt aflæses og $\log_b(a)$ udregnes.

$a = 8$, $b = 3$, $\log_b(a) = \log_3(8) \approx 1.893$, $f(n) = n^2$.

Vi kan sætte $n^2 = \Omega\left(n^{1.893+\epsilon}\right)$ for $\epsilon = 0.1$. Derfor er vi i case 3.

Vi har allerede opfyldt krav 1, nu kigger vi på krav 2:

$$
\begin{aligned}
& a \cdot f\left(\frac{n}{b}\right) \leq c \cdot f(n) \\
& \Updownarrow \\
& 8 \cdot \left(\frac{n}{3}\right)^2 \leq c \cdot n^2 \\
& \Updownarrow \\
& \frac{8n^2}{3^2} \leq c \cdot n^2 \\
& \Updownarrow \\
& \frac{8}{9} \cdot n^2 \leq c \cdot n^2
\end{aligned}
$$

Som det ses ovenfor er krav 2 overholdt for $\frac{8}{9} \leq c < 1$. Dvs. vi kan vælge $c = \frac{8}{9}$ og krav 2 vil være overholdt.

Vi har opfyldt krav 1 og 2 derfor bliver løsningen til rekursionsligningen $T(n) = \Theta\left(n^2\right)$

ii) $T(n) = 9 \cdot T\left(\frac{n}{3}\right) + n^2$

Siden rekursionsligningen er på den korrekte form kan variabler nemt aflæses og $\log_b(a)$ udregnes.

$a = 9$, $b = 3$, $\log_b(a) = \log_3(9) = 2$, $f(n) = n^2$.

Vi kan sætte $n^2 = \Theta\left(n^2\right)$. Derfor er vi i case 2 og løsningen til rekursionsligningen bliver $T(n) = \Theta\left(n^2 \cdot \log(n)\right)$.

iii) $T(n) = 10 \cdot T\left(\frac{n}{3}\right) + n^2$

Siden rekursionsligningen er på den korrekte form kan variabler nemt aflæses og $\log_b(a)$ udregnes.

$a = 10$, $b = 3$, $\log_b(a) = \log_3(10) \approx 2.096$, $f(n) = n^2$.

Vi kan sætte $n^2 = O\left(n^{2.096-\epsilon}\right)$ for $\epsilon = 0.09$. Derfor er vi i case 1 og løsningen til rekursionsligningen bliver $T(n) = O\left(n^{\log_3(10)}\right)$

3 Opgave 3 - Cormen et al. opgave 4.5-3

Brug Master Theorem metoden til at vise rekursionsligningen for binær søgning

$$T(n) = T\left(\frac{n}{2}\right) + \Theta(1) \tag{1}$$

har løsningen $T(n) = \Theta(\lg n)$.

Da (1) er på formen

$$T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n),$$

hvor $a = 1$, $b = 2$ og $f(n) = \Theta(1)$, kan Master Theorem benyttes. Da $\alpha = \log_2 1 = 0$ og

$$f(n) = \Theta(1) = \Theta(n^{\alpha}) = \Theta(n^0) = \Theta(1),$$

så er vi i Case 2 af Master Theorem.

Dvs. løsningen er

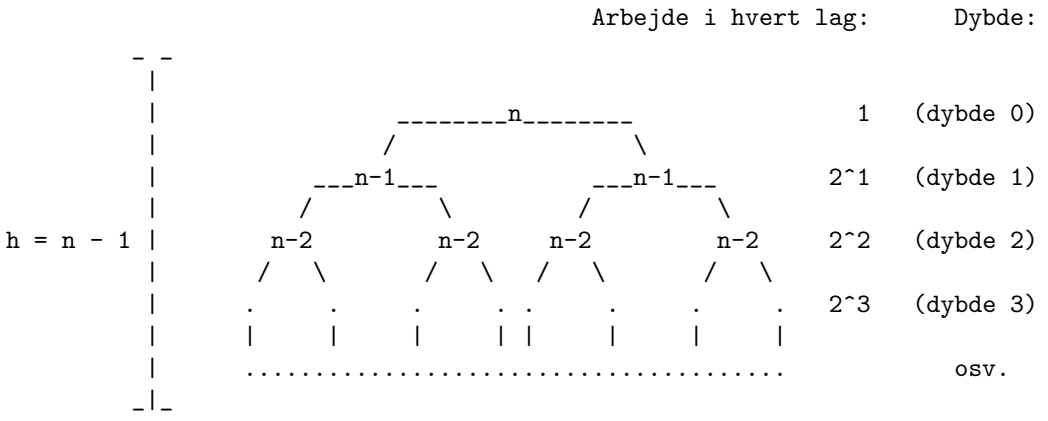
$$T(n) = \Theta(n^{\alpha} \cdot \log n) = \Theta(\log n)$$

4 Opgave 4 - Cormen et al. opgave 4.4-4

I følgende opgave skal vi anvende *rekursionstræsmetoden* til at bestemme en asymptotisk øvre grænse for køretiden på f.eks. en rekursiv algoritme, hvis arbejde kan beskrives vha. den givne rekursion:

$$T(n) = 2 \cdot T(n-1) + 1 \tag{2}$$

Givet denne rekursionsligning kan vi konstatere, at der for et problem af input størrelse n bliver lavet 2 rekursive kald, hvoraf hver af disse rekursive kald arbjeder på delproblemer af størrelse $n - 1$ (dette ses ved at kigge på første led i Eqn. 2). I denne forbindelse ser vi også at det *lokale arbejde* for hvert af de rekursive kald er konstant (dette ses ved at kigge på sidste led i Eqn. 2). Ud fra disse informationer kan vi tegne det korresponderende rekursionstræ:



For at kunne bestemme en asymptotisk øvre grænse for køretiden kigger vi på ovenstående rekursionstræ og observerer:

1. Det totale antal knuder i dybde i er 2^i
2. Delproblemstørrelsen i dybde i er $n - i$

3. Det totale arbejde i dybde i er 2^i (dette er tilfældet, da det lokale arbejde er konstant og ikke en funktion af delproblemstørrelsen. Det totale arbejde i dybde i afhænger derfor kun af antallet af knuder i dybde i)
4. Højden af træet er $n-1$

En asymptotisk tæt grænse for det samlede arbejde er derfor (summen af arbejde i hvert lag):

$$\begin{aligned} T(n) &= 1 + 2^1 + 2^2 + ... + 2^{n-2} + 2^{n-1} && (3)\\ &= \sum_{i=0}^{n-1} 2^i && (4)\\ &= \frac{2^n - 1}{2-1} && (5)\\ &= 2^n - 1 && (6)\\ &= \Theta(2^n) && (7) \end{aligned}$$

Da vi har fundet en tæt grænse, har vi også bestemt en asymptotisk øvre grænse. I ovenstående udledning er resultatet af en *geometrisk* række anvendt (se [1, s. 1147]).

I forbindelse med rekursionstræet og rækken i Eqn. 3, ses det også at arbejdet i lagene vokser eksponentielt. Arbejdet der laves i det sidste lag dominerer, hvilket også afspejles i det sidste led i rækken (det er det største).

Til sidst kan det nævnes at *Master Theorem* ikke kan anvendes, eftersom rekursionsligningen i Eqn. 2 ikke er på den rigtige form. For at kunne anvende Master Theorem skal rekursionsligningen være på formen:

$$T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n), \text{ med } a \geq 1,\ b > 1,\ f(n) \text{ en funktion.} \quad (8)$$

Vi ser i denne sammenhæng at rekursionsligningen i Eqn. 2 ikke arbejder på delproblemer af størrelse $\frac{n}{b}$, men derimod $n-1$.

5 Opgave 5 - Cormen et al. opgave 4.5-4

Kan Master Theorem[3, p. 19] bruges på $T(n) = 4T\left(\frac{n}{2}\right) + n^2 \cdot \lg(n)$? Giv en øvre og nedre asymptotisk grænse.

Der tjekkes om ligning er på den rigtige form:

$$T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n)$$

hvilket den er, hvor $a = 4$, $b = 2$ og $f(n) = n^2 \cdot \lg(n)$.
Vi kan nu udregne α, $\alpha = \log_b a = \log_2 4 = 2$.

Derefter sammenligner vi $n^\alpha = n^2$ og $f(n) = n^2 \cdot \lg(n)$, for at finde ud af om Master Theorem kan bruges, og hvilken case rekursionsligningen falder i. Det asymptotiske forhold mellem to funktioner kan findes ved at finde grænseværdien af deres forhold, når n går mod uendelig.

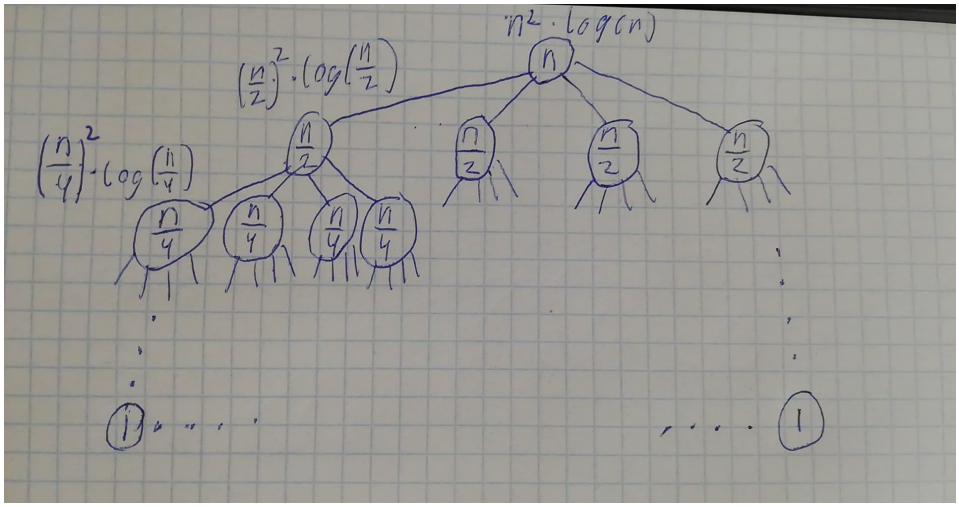
$$\frac{n^\alpha}{f(n)} = \frac{n^2}{n^2 \cdot \lg(n)} = \frac{1}{\lg(n)} \to 0 \text{ for } n \to \infty \text{ [2, p. 17]}$$

Dette betyder, at $n^\alpha = o(f(n)) \Leftrightarrow f(n) = \omega(n^\alpha)$.
Altså kan det ikke være case 2, da $f(n) \neq \Theta(n^\alpha)$.
Det kan heller ikke være case 1, da $f(n) \neq O(n^\alpha) \Rightarrow f(n) \neq O(n^{\alpha-\epsilon})$, fordi $n^{\alpha-\epsilon}$ vokser langsommere end n^α.

Til sidst prøves case 3. Hvis ϵ vælges til at være 0.001

$$\frac{f(n)}{n^{\alpha+\epsilon}} = \frac{n^2 \cdot \lg(n)}{n^{2+0.001}} = \frac{\lg(n)}{n^{0.001}} \to 0 \text{ for } n \to \infty \text{ [2, p. 19]}$$

Altså er $f(n) = o(n^{\alpha+\epsilon}) \Leftrightarrow f(n) \neq \Omega(n^{\alpha+\epsilon})$. Da ethvert polynomium vokser hurtigere end enhver logaritme, så må dette gælder for alle positive ϵ og ikke kun 0.001, altså er det ikke case 3. Rekursionsligningen falder mellem case 2 og 3. $f(n)$ vokser hurtigere end n^α, men langsommere end $n^{\alpha+\epsilon}$. Altså kan Master Theorem ikke bruges. Der kigges derfor på rekursiontræet.



Figur 1: Rekursiontræet for T(n)

Ud fra rekursionstræet kan vi observere:

- Antal knuder i dybden i er 4^i
- Hver knudes arbejde, til dybden i, er $f(n/2^i) = (n/2^i)^2 \cdot \lg(n/2^i)$
- Det totale arbejde i dybden i er $4^i \cdot (n/2^i)^2 \cdot \lg(n/2^i)$
- Højden af træet er $\lg(n)$, da n halveres ved hver rekursion.

Formlen for det totale arbejde til dybden i kan forkortes.

$$4^i \cdot \left(\frac{n}{2^i}\right)^2 \cdot \lg\left(\frac{n}{2^i}\right) = 4^i \cdot \frac{n^2}{4^i} \cdot \lg\left(\frac{n}{2^i}\right)$$

$$= n^2 \cdot \lg\left(\frac{n}{2^i}\right) = n^2 \cdot (\lg(n) - \lg(2^i)) = n^2 \cdot (\lg(n) - i)$$

Vi skal nu argumentere for en øvre og nedre grænse.

Da hvert lags arbejde bliver mindre som dybden i stiger, så må arbejdet for det øverste lags, med $i = 0$, være størst. Hvis det øverste lags arbejde blev lavet i hvert lag, så vil dette være mere end arbejdet som $T(n)$ rigtig skaber. Dette vil være en øvre grænse. Da der er $\lg(n)$ lag, så må den øvre grænse kunne udregnes på følgende måde:

$$T(n) = O(\text{øverstes lags arbejde } \cdot \text{ antal lag})$$

$$T(n) = O((n^2 \cdot (lg(n) - 0)) \cdot \lg(n)) = O(n^2 \cdot \lg(n)^2)$$

Der kigges derefter på den øverste halvdel af lagene, for at finde en nedre grænse. Dvs. lagene med dybde

$$i = 0, 1, 2, \ldots, k \text{ hvor } k = \frac{1}{2}\lg(n)$$

Alle disse lag, må lave det samme arbejde som lag k eller mere arbejde end lag k.

Arbejdet i lag k er

$$n^2 \cdot \left(\lg(n) - \frac{1}{2}\lg(n)\right) = n^2 \cdot \frac{1}{2}\lg(n)$$

Arbejdet i alt for disse k lag må derfor mindst være

$$n^2 \cdot \frac{1}{2}\lg(n) \cdot k = n^2 \cdot \frac{1}{2}\lg(n) \cdot \frac{1}{2}\lg(n) = n^2 \cdot \frac{1}{4}\lg(n)^2$$

Arbejdet i alt for $T(n)$ må være større eller lig dette, altså

$$T(n) = \Omega\left(n^2 \cdot \frac{1}{4}\lg(n)^2\right) = \Omega(n^2 \cdot \lg(n)^2)$$

Dermed er en øvre og nedre grænse for $T(n)$ givet. Da både den øvre og nedre grænse er det samme, så kan der også konkluderes, at $T(n) = \Theta(n^2 \cdot \lg(n)^2)$

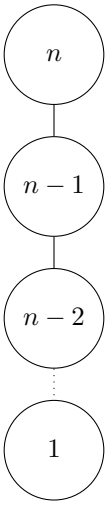
6 Opgave 6 - Cormen et al. opgave 2.3-4

Her får vi givet en rekursiv version af insertionsort, hvor vi ved brug af rekursionstræsmetoden skal finde køretiden og derefter svare på om vi kan finde køretiden ved hjælp af Master Theorem.

Først kan vi opstille en rekursionsligning for algoritmen. Vi ved at den kun laver et rekursivt kald, hvor input bliver en mindre. Derudover er $f(n) = n$ i worst-case, da man ligesom ved den normale insertionsort kan blive nødt til at rykke alle de sorteret elementer en plads til højre for at gøre plads til elementet efter den sorteret mængde forrest i arrayet. Det vil sige, at rekursionligningen ser således ud:

$$T(n) = T(n-1) + n$$

Herefter kan vi tegne rekursionstræet som kommer til at se ud som følgende figur:



Figur 2: Rekursiontræet

For at finde køretiden skal vi finde summen af lokalt arbejde i algoritmens rekusionstræ, i forhold til rekursionstræsmetoden er denne opsummeret på slide 12 i [3]. Højden må være n, da vi kun reducerer n med 1 fra lag til lag. Da der kun er en knude i hvert lag må dennes lokale arbejde være lagets arbejde. Til sidst skal vi summe hver lags sum sammen, hvilket her bliver $n + (n-1) + (n-2) + .. + 1 = \frac{n \cdot (n+1)}{2} = O(n^2)$. Vi kan derfor konkludere at den rekursive version af insertionsort også har en worst case køretid på $O(n^2)$.

Man kan hurtigt se at rekursionligningen ikke har formen:

$$T(n) = a \cdot T(n/b) + n$$

Da det rekursive kald ikke har formen $T(n/b)$, men derimod $T(n-1)$. Hvilket vil sige at man ikke kan bruge master teoremet til at løse rekursionligningen.

7 Opgave 7 - Cormen et al. opgave 4.2-1

Vi skal bruge Strassens algoritme til at finde matrixproduktet $A \cdot B = C$, hvor

$$A = \begin{pmatrix} 1 & 3 \\ 7 & 5 \end{pmatrix} \qquad B = \begin{pmatrix} 6 & 8 \\ 4 & 2 \end{pmatrix}$$

Vi udregner nu $S_1, ..., S_{10}$, $P_1, ..., P_7$ og $C_{11}, C_{12}, C_{21}, C_{22}$ som beskrevet på slides. Vi er desuden i et specialtilfælde, hvor delmatricerne A_{ij}, B_{ij} og C_{ij} blot er tal, dvs. at gange to delmatricer sammen svarer til en almindelig multiplikation af to tal.

Vi udregner $S_1, ..., S_{10}$:

$$
\begin{aligned}
S_1 &= B_{12} - B_{22} = 8 - 2 = 6 \\
S_2 &= A_{11} + A_{12} = 1 + 3 = 4 \\
S_3 &= A_{21} + A_{22} = 7 + 5 = 12 \\
S_4 &= B_{21} - B_{11} = 4 - 6 = -2 \\
S_5 &= A_{11} + A_{22} = 1 + 5 = 6 \\
S_6 &= B_{11} + B_{22} = 6 + 2 = 8 \\
S_7 &= A_{12} - A_{22} = 3 - 5 = -2 \\
S_8 &= B_{21} + B_{22} = 4 + 2 = 6 \\
S_9 &= A_{11} - A_{21} = 1 - 7 = -6 \\
S_{10} &= B_{11} + B_{12} = 6 + 8 = 14
\end{aligned}
$$

Vi kan nu udregne $P_1, ..., P_7$:

$$
\begin{aligned}
P_1 &= A_{11} \cdot S_1 = 1 \cdot 6 = 6 \\
P_2 &= S_2 \cdot B_{22} = 4 \cdot 2 = 8 \\
P_3 &= S_3 \cdot B_{11} = 12 \cdot 6 = 72 \\
P_4 &= A_{22} \cdot S_4 = 5 \cdot (-2) = -10 \\
P_5 &= S_5 \cdot S_6 = 6 \cdot 8 = 48 \\
P_6 &= S_7 \cdot S_8 = (-2) \cdot 6 = -12 \\
P_7 &= S_9 \cdot S_{10} = (-6) \cdot 14 = -84
\end{aligned}
$$

Og til slut $C_{11}, C_{12}, C_{21}, C_{22}$:

$$
\begin{aligned}
C_{11} &= P_5 + P_4 - P_2 + P_6 = 48 + (-10) - 8 + (-12) = 18 \\
C_{12} &= P_1 + P_2 = 6 + 8 = 14 \\
C_{21} &= P_3 + P_4 = 72 + (-10) = 62 \\
C_{22} &= P_5 + P_1 - P_3 - P_7 = 48 + 6 - 72 - (-84) = 66
\end{aligned}
$$

Dvs. svaret er:

$$
C = \begin{pmatrix} 18 & 14 \\ 62 & 66 \end{pmatrix}
$$

8 Opgave 8 - Cormen et al. opgave 4.2-5

Vi får at vide vi kan udregne matrix, matrix multiplikation på tre måder:

- 68×68 matricer med 132,464 multiplikationer.
- 70×70 matricer med 143,640 multiplikationer.
- 72×72 matricer med 155,424 multiplikationer.

Vi vil gerne finde frem til køretid for alle tre metoder ved brug af en "divide and conquer"metode. Vi kan følge samme fremgangsmåde som i kapitlet "A simple divide and conquer algorithm"fra [1, p. 76].

Hver gang vi går et rekursivt step ned vil vi dele matricerne op i mindre dele. Alt efter hvilken metode vi bruger vil vi gøre input til hvert af de næste rekursive kald mindre ved enten $\frac{n}{68}$, $\frac{n}{70}$ eller $\frac{n}{72}$. Formen for vores rekursionsligning bliver så eksempelvis med $\frac{n}{68}$: $T(n) = a \cdot T(\frac{n}{68}) + f(n)$. Den mængde multiplikationer der hører til metoden skal så ganges på hvert af de rekursive kald, det bliver altså sat ind på a's plads. For hvert step ned skal vi lægge matricerne sammen fra kaldet under os, $f(n) = \Theta(n^2)$. Nu kan vi udregne de individuelle rekursionsligninger for de tre metoder.

68×68 metode

Vi indsætter i formen på en rekursionsligning.

$$T(n) = 132464 \cdot T\left(\frac{n}{68}\right) + n^2$$

$\log_b(a) = \log_{68}(132464) \approx 2.79513$. Vi er i case 1 af Master Theorem da $n^2 = O(n^{2.79513-\epsilon})$ for $\epsilon = 0.5$ derfor bliver $T(n) = \Theta(n^{\log_b(a)}) \approx \Theta(n^{2.79513})$

70×70 metode

Vi indsætter i formen på en rekursionsligning.

$$T(n) = 143640 \cdot T\left(\frac{n}{70}\right) + n^2$$

$\log_b(a) = \log_{70}(143640) \approx 2.79512$. Vi er i case 1 af Master Theorem da $n^2 = O(n^{2.79512-\epsilon})$ for $\epsilon = 0.5$ derfor bliver $T(n) = \Theta(n^{\log_b(a)}) \approx \Theta(n^{2.79512})$

72×72 metode

Vi indsætter i formen på en rekursionsligning.

$$T(n) = 155424 \cdot T\left(\frac{n}{72}\right) + n^2$$

$\log_b(a) = \log_{72}(155424) \approx 2.79515$. Vi er i case 1 af Master Theorem da $n^2 = O(n^{2.79515-\epsilon})$ for $\epsilon = 0.5$ derfor bliver $T(n) = \Theta(n^{\log_b(a)}) \approx \Theta(n^{2.79515})$

Sammenligning med Strassen's:

Den bedste af de tre versioner er den med 70×70, sammenlignet med Strassen's algoritme som bruger $\Theta(n^{\log_2(7)}) \approx \Theta(n^{2.80735})$ ([1, p. 80]), så vil 70×70 metoden være bedre. Faktisk vil alle tre metoder være bedre.

9 Opgave 9 - Cormen et al. opgave 4.5-2

Professor Caesar har fundet en matrixmultiplikations algoritme med rekursionsligningen

$$T(n) = a \cdot T\left(\frac{n}{4}\right) + \Theta(n^2), \tag{9}$$

men han undrer sig over hvor stor a (fanout) kan være før algoritmen ikke er så smart mere sammenlignet med Strassen's algoritme. Altså, hvad er den største a værdi for hvilket køretiden af algoritmen beskrevet ved (9) er asymptotisk hurtigere end Strassens algoritme.

Ved brug af Master Theorem på (9) er $\alpha = \log_4 a$. For Strassen's algorime er $\alpha = \log_2 7$. Observer

$$\log_2 7 = \frac{\ln 7}{\ln 2} = \frac{2 \cdot \ln 7}{2 \cdot \ln 2} = \frac{\ln 49}{\ln 4} = \log_4 49$$

Altså, når $a = 49$ vokser de asymptotisk ens. Dermed kan a højst være 48.

Benyttes Master Theorem på (9) med $a = 48$, så følger det af Case 1, at løsningen er $T(n) = \Theta(n^{\log_4 48}) = \Theta(n^{2.792\cdots})$, hvilket er asymptotisk hurtigere end Strassen's algoritme.

10 Opgave 10 - Cormen et al. opgave 4.2-3

I følgende opgave skal vi modificere Strassen's algoritme for matrixmultiplikation, således at algoritmen kan håndtere input matricer af størrelse $n \times n$, hvor sidelængden n ikke er en potens af 2, dvs. når $n \neq 2^k,\ k \in \mathbb{N}$.

For at kunne anvende Strassen's algoritme med input matricer hvis sidelængde n ikke er en potens af 2, kan man tilføje nuller således at input matricernes sidelængde n bliver en potens af 2. For eksempel, lad os sige vi er givet matricerne:

$$A = \begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{bmatrix} \quad \text{og} \quad B = \begin{bmatrix} B_{11} & B_{12} & B_{13} \\ B_{21} & B_{22} & B_{23} \\ B_{31} & B_{32} & B_{33} \end{bmatrix} \tag{10}$$

I dette tilfælde kan vi ikke anvende Strassen's algoritme, dog hvis vi nu tilføjer nuller, så vil vi have følgende og Strassen's algoritme vil kunne anvendes:

$$A = \begin{bmatrix} A_{11} & A_{12} & A_{13} & 0 \\ A_{21} & A_{22} & A_{23} & 0 \\ A_{31} & A_{32} & A_{33} & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \quad \text{og} \quad B = \begin{bmatrix} B_{11} & B_{12} & B_{13} & 0 \\ B_{21} & B_{22} & B_{23} & 0 \\ B_{31} & B_{32} & B_{33} & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \tag{11}$$

således at det korresponderende produkt af disse vil være:

$$AB = C = \begin{bmatrix} C_{11} & C_{12} & C_{13} & 0 \\ C_{21} & C_{22} & C_{23} & 0 \\ C_{31} & C_{32} & C_{33} & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}. \tag{12}$$

Vi kan se at dette er tilfældet ved f.eks. at kigge på hvad første element i øverste venstre hjørne af matrix C er lig med:

$$A_{11} \cdot B_{11} + A_{12} \cdot B_{21} + A_{13} \cdot B_{31} + 0 \cdot 0 = A_{11} \cdot B_{11} + A_{12} \cdot B_{21} + A_{13} \cdot B_{31} \tag{13}$$

$$= C_{11}. \tag{14}$$

I denne sammenhæng ser vi at de tilføjede nuller ikke har nogen effekt på værdien af C_{11}. Dette er også tilfældet for de resterende elementer i den øverste 3×3 venstre delmatrix af C.

Vi kan ligeledes kigge på hvad elementet i øverste højre hjørne i matrix C er lig med:

$$A_{11} \cdot 0 + A_{12} \cdot 0 + A_{13} \cdot 0 + 0 \cdot 0 = 0. \tag{15}$$

Her ser vi at de tilføjede nuller gør at resultatet i sidste ende reducerer til 0.

I forhold til køretiden af denne modificerede algoritme, så vil køretiden være den samme, da vi i værste tilfælde kun tilføjer et stykke arbejde der er *proportionalt* med n (sidelængden på input matricerne). Mere præcist har vi, at hvis en matrix med sidelængde n ikke er en potens af 2, så tilføjer vi nuller indtil dette er tilfældet. Sidelængden n af matricen fordobles højst i denne sammenhæng. Køretiden for Strassen's algoritme vil derfor forblive $\Theta(n^{\log_2(7)})$.

Det kan dog nævnes at tilføjelsen og fjernelsen af nullerne ialt må tage $O(n^2)$ tid, da vi i *værste tilfælde* først skal indsætte $n \cdot n$ ekstra nuller i matricerne A og B. Vi vil herefter i *værste tilfælde* skulle fjerne $n \cdot n$ nuller i matricen C.

11 Opgave 11 - Eksamen juni 2010, opgave 3

Vi ønsker for et vilkårligt positivt heltal n at finde cifrene $b_k b_{k-1} ... b_1 b_0$ i n's repræsentation i 2-talssystemet. For eksempel, for $n = 43$ er $b_5 b_4 ... b_1 b_0$ lig 101011.

Betragt følgende algoritme til at generere k og b_i'er:

```
1   BinaryDigits(n)
2       i = 0
3       d = n
4       while d > 0
5           b_i = d mod 2
6           d = d div 2
7           i = i + 1
8       k = i - 1
```

Spørgsmål a

Angiv resultatet (dvs. k og b_i'erne) af ovenstående algoritme når input er $n = 55$.

Algoritmen følges:

1. $i = 0$, $d = 55$, $d > 0$ (Ved første test i while (efter 0 gennemløb))
2. $b_0 = 1$, $d = 27$, $i = 1$, $d > 0$ (Ved anden test i while (efter 1 gennemløb))
3. $b_1 = 1$, $d = 13$, $i = 2$, $d > 0$ (Ved tredje test i while (efter 2 gennemløb))
4. $b_2 = 1$, $d = 6$, $i = 3$, $d > 0$ (Ved fjerde test i while (efter 3 gennemløb))
5. $b_3 = 0$, $d = 3$, $i = 4$, $d > 0$ (Ved femte test i while (efter 4 gennemløb))
6. $b_4 = 1$, $d = 1$, $i = 5$, $d > 0$ (Ved sjette test i while (efter 5 gennemløb))
7. $b_5 = 1$, $d = 0$, $i = 6$, $d == 0$ (Ved sidste test i while (efter 6 gennemløb))
8. $k = 5$ (Efter while-løkken)

55 kan nu skrives i binær form via $b_5 b_4 ... b_1 b_0$, som er lig 110111.

Spørgsmål b

Vis følgende invariant for **while**-løkken:

Når testen ved indgangen til **while**-løkken udføres, gælder

i) $n = d \cdot 2^i + \sum_{j=0}^{i-1} b_j \cdot 2^j$
ii) $d \geq 0$

Det vises ved induktion på antal gange testen i **while**-løkken er blevet foretaget.

Basis: Testen er blevet foretaget nul gange. Her følger invarianten del i) af initialiseringen $i = 0$ og $d = n$, da

$$n \cdot 2^0 + \sum_{j=0}^{-1} b_j \cdot 2^j = n \cdot 1 + 0 = n$$

Del ii) følger af at $d = n$, og n er et positivt tal.

Induktionsskridt: For del i), antag udsagnet er sandt før et gennemløb af løkken. Vi skal så vise, at det gælder efter gennemløbet. Kald de nye værdier af i og d efter gennemløbet for i' og d'. Vi har $i' = i + 1$, $d' = d$ div 2, samt $b_i = d$ mod 2.
Der gælder altid at

$$x = y \cdot (x \text{ div } y) + (x \bmod y)$$

Af induktionsantagelsen (at invarianten gælder før gennemløbet), samt overstående formel,

med $x = d$ og $y = 2$, fås:

$$
\begin{aligned}
n &= d \cdot 2^i + \sum_{j=0}^{i-1} b_j \cdot 2^j \\
&= (2 \cdot (d \text{ div } 2) + (d \bmod 2)) \cdot 2^i + \sum_{j=0}^{i-1} b_j \cdot 2^j \\
&= (2 \cdot d' + b_i) \cdot 2^i + \sum_{j=0}^{i-1} b_j \cdot 2^j \\
&= d' \cdot 2^{i+1} + b_i \cdot 2^i + \sum_{j=0}^{i-1} b_j \cdot 2^j \ (b_i \cdot 2^i \text{ sammenlægges med summen, da det er summens næste led}) \\
&= d' \cdot 2^{i+1} + \sum_{j=0}^{i} b_j \cdot 2^j \\
&= d' \cdot 2^{i'} + \sum_{j=0}^{i'-1} b_j \cdot 2^j
\end{aligned}
$$

hvilket var, hvad vi skulle vise.

For del ii) bruger vi, at division (også heltalsdivision) med to ikke-negative tal altid giver ikke-negative tal, så af induktionsantagelsen $d \geq 0$, fås $d' = d \text{ div } 2 \geq 0$.

Spørgsmål c

Vis at algoritmen er korrekt, dvs. at den for alle positive heltal n beregner n's repræsentation $b_k b_{k-1} ... b_1 b_0$ i 2-talssystemet.

Vi viser først, at algoritmen terminerer. I starten er d = n, og n er et heltal. Alle senere værdier af d er resultatet af en heltalsdivision, og derfor heltal, så d er altid et heltal. Hvis $d > 0$ gælder derfor $d \geq 1$, så for ethvert løkkegennemløb har vi $d' = \lfloor \frac{d}{2} \rfloor \leq d - 1$. Da løkken stopper når $d \leq 0$, kan løkken ikke køre uendeligt.

Derefter viser vi, via invarianten, at output er korrekt når den stopper. Når løkken stopper ved vi at $d \leq 0$. Af invarianten del ii) gælder $d \geq 0$. Altså må $d = 0$ når løkken stopper. Indsættes dette sammen med $k = i - 1$ i invarianten del i) fås

$$n = d \cdot 2^i + \sum_{j=0}^{i-1} b_j \cdot 2^j = \sum_{j=0}^{k} b_j \cdot 2^j$$

hvilket vi skulle vise.

Spørgsmål d

Giv en analyse af køretiden for algoritmen som funktion af n.

Vi har for ethvert løkkegennemløb $d' = \lfloor d/2 \rfloor \leq d/2$. Da $d = n$ før første gennemløb, gælder efter t gennemløb at $d \leq n/2^t$.

Før sidste gennemløb af løkken har vi $d \geq 1$ (da $d > 0$ og da d som argumenteret ovenfor altid er et heltal), så før sidste gennemløb har vi $1 \leq d \leq n/2^t$, hvoraf følger $t \leq \lg(n)$. Da hvis t var større end $\lg(n)$, så ville $1 > n/2^t$

Så det samlede antal gennemløb er højst $1 + \lg(n)$. Hvert gennemløb tager $O(1)$ tid, og derudover udføres $O(1)$ yderligere arbejde før og efter løkken. Alt i alt er køretiden $O(\lg(n))$.

Litteratur

[1] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, third edition, 2009.

[2] Rolf Fagerberg. Asymptotisk analyse af algoritmers køretider. URL `https://imada.sdu.dk/~rolf/Edu/DM507/F20/asymptotiskAnalyseAfAlg.pdf`, 2020.

[3] Rolf Fagerberg. Divide-and-Conquer algoritmer. URL `https://imada.sdu.dk/~rolf/Edu/DM507/F20/divideAndConquerSlides.pdf`, 2020.