Eksamensdisposition - Dynamisk programmering

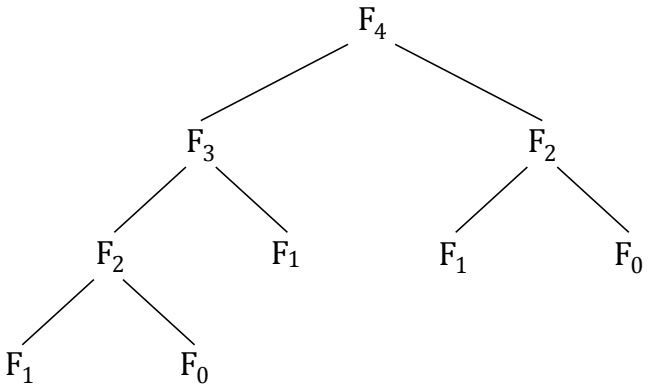
Søren Mulvad, rbn601

3. april 2019

- **Paradigmet**
- **Eksempel med beregning af Fibonacci**
- **Bevis af Theorem 15.1 (*Optimal delstruktur af en LCS*)**
- **Håndkørsel af** `LCS`
  - $X = ABA$ og $Y = BA$

1 Dynamisk Programmering

- **Paradigmet**
  - Minder en anelse om del og hersk-paradigmet: Del problemet ind i mindre delproblemer, løs delproblemerne rekursivt og kombiner herefter løsningerne.
  - Forskellen ligger i, at problemerne har *optimal delstruktur*. Det vil sige delproblemerne på en eller anden måde overlapper. Altså når flere delproblemer har de samme deldelproblemer. Det kunne f.eks. være når vi skal beregne et Fibonacci tal eller i det jeg vil vise med LCS.
- **Eksempel med Fibonacci**



Figur 1: Problemer der skal løses i Fib. Her ser vi, at F_2 optræder flere gange.

Uden DP er køretiden eksponentiel, med DP er køretiden $O(n)$ (såfremt vi kan lægge store tal sammen i konstant tid).

- **Beskrivelse af LCS-problemet**
  Vi vil gerne finde Longest Common Subsequence af f.eks. to DNA-strenge.
  Hvis vi f.eks. har $S_1 = ACCG$ og $S_2 = AG$ ville den længste den længste sekvens være AG, og længden er hermed 2.
- **Theorem 15.1 (*Optimal delstruktur af en LCS*)**
  Lad $X = \langle x_1, x_2, ..., x_m \rangle$ og $Y = \langle y_1, y_2, ..., y_n \rangle$ være sekvenserne (f.eks. DNA-strenge), og lad $Z = \langle z_1, z_2, ..., z_k \rangle$ være en hvilken som helst LCS af X og Y:

  1: Hvis $x_m = y_n$, så er $z_k = x_m = y_n$ og Z_{k-1} er en LCS af X_{m-1} og Y_{n-1}.

  2: Hvis $x_m \neq y_n$ og $z_k \neq x_m$, så medfører det at Z er en LCS af X_{m-1} og Y.

  3: Hvis $x_m \neq y_n$ og $z_k \neq y_n$, så medfører det at Z er en LCS af X og Y_{n-1}.

- **Beviser af Theorem 15.1**
  Alle er modstridsbeviser. Tegn strenge ala. følgende for at understøtte argumentation:

```
X: *****A
Y: ****A
Z: **A
```

  1: Antag $z_k \neq x_m$. Så kunne vi appende $x_m = y_n$ på Z, hvorved vi fik en CS af X og Y med en længde $k + 1$.
  Men vi antog jo netop at Z var en *longest* common subsequence med længden k, og derfor har vi en modstrid. Altså er $z_k = x_m = y_n$.

Antag Z_{k-1} ikke LCS af X_{m-1} og Y_{n-1}. Da Z_{k-1} er CS af X_{m-1} og Y_{n-1}, men ikke den længste pr. vores antagelse, så må der findes en anden delstreng W som er CS af X_{m-1} og Y_{n-1}, og som er længere hvorved $|W| > |Z_{k-1}| = k - 1$.
Men så kunne vi appende $x_m = y_n$ til W, og herved få en CS af X og Y med længden $|W| + 1 > k$. Nu har vi fundet en fælles delstreng af X og Y, som er længere end vores længste delstreng, hvilket er en modstrid.

2: Antag Z ikke LCS af X_{m-1} og Y. Z er stadig CS fordi vi antager $z_k \neq x_m$. Fordi vi antager det ikke er den længste delstreng, så findes der en delstreng W som er CS af X_{m-1} og Y og længere end Z, altså $|W| > |Z|$.
Men W er også CS af X og Y, og dermed længere end vores længste delstreng, hvilket er en modstrid.

3: Symmetrisk med 2.

- **Håndkørsel af `LCS`**
  - Ud fra vores Theorem kan vi se, at vi får følgende formel for hvad der skal være på plads $c[i, j]$, idet det angiver længden af en LCS for X_i og Y_j:

$$c[i,j] = \begin{cases} 0 & \text{hvis } i = 0 \text{ eller } j = 0 \\ c[i-1, j-1] + 1 & \text{hvis } i, j > 0 \text{ og } x_i = y_j \\ \max(c[i, j-1], c[i-1, j]) & \text{hvis } i, j > 0 \text{ og } x_i \neq y_j \end{cases}$$

  Vi ser at det er rekursivt defineret, og løsningerne til visse delproblemer vil optræde flere gange. Derfor er det optimalt at løse med dynamisk programmering. Jeg vil nu give en håndkørsel af en implementering der er bottom-up:
  - Eksempel hvor $X = ABA$ og $Y = BA$:

		j	0	1	2
i				B	A
0			0	0	0
1	A		0	↑ 0	↖ 1
2	B		0	↖ 1	↑ 1
3	A		0	↑ 1	↖ 2

  - Vi får nu en køretid $O(nm)$ og et umiddelbart pladsforbrug $O(nm)$.

    Såfremt vi kun er interesserede i hvor lang den længste delstreng er, men ikke hvilke bogstaver den består af, kan vi forbedre pladsforbruget til $O\Big(\min(n, m)\Big)$. Det gøres ved at vælge den mindste delstreng til at være i toppen, og kun gemme den nuværende række samt rækken lige over.
  - Fordelene ved bottom-up, som vi bruger her, er at vi undgår rekursion og derfor ofte vil have lavere konstantled i forhold til top-down. Fordelen ved top-down er tilgengæld, at vi kun beregner de delproblemer som der faktisk er behov for, for at løse det oprindelige problem hvorimod bottom-up regner alle.

- **Bonusviden om `Cut-Rod` til eventuelle spørgsmål**

1. Der er et optimalt indeks i hvor vi kan cutte stangen op i to mindre delstange (delproblemer) - denne kan være evt. være 0:

$$r_n = \max\{p_n, r_1 + r_{n-1}, r_2 + r_{n-2}, ..., r_{n-1} + r_1\}$$

2. Antag vi får det første cut der skal laves fra venstre side med længde i, så får vi:

$$r_n = p_i + r_{n-i}$$

3. Da får vi følgende formel:

$$r_n = \max_{1 \leq i \leq n} (p_i + r_{n-i})$$

4. Dette kan vi meget nemt implementere som en rekursiv algoritme, hvor vi så får en eksponentiel køretid. Ved at bruge DP og gemme resultaterne når vi har beregnet dem får en køretid på $O(n^2)$.