FÖRELÄSNING

ALGORITMKONSTRUKTION 3

DYNAMISK PROGRAMMERING

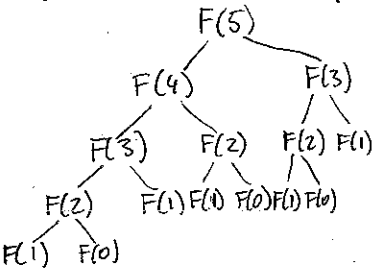
1. BESTÄM STRUCTUREN HOS OPTIMAL LÖSNING
2. STÄLL UPP REKURSION FÖR OPTIMALVÄRDET
3. BERÄKNA DELPROBLEMENS OPTIMALVÄRDEN FRÅN SMÅ TILL STORA
4. KONSTRUERA OPTIMALLÖSNINGEN (EV. MED EXTRA INFORMATION SOM LAGRAS I STEG 3)

EXEMPEL 1: FIBONACCITALEN

$F(0)=1,\ F(1)=1,\ F(n)=F(n-1)+F(n-2)$

1 1 2 3 5 8 13 21 34 55 89...

REKURSIV IMPLEMENTATION – EXPONENTIELL TID



F(40) TAR 235s MED PYTHON
45 MED C/JAVA

F(70) TAR 14 ÅR MED PYTHON
3 MÅNADER MED C/JAVA

DYNAMISK PROGRAMMERING – LINJÄR TID

```
F[0]←1; F[1]←1
FOR i←2 TO n DO
  F[i]←F[i-1]+F[i-2]
RETURN F[n]
```

F(40) TAR 80ms MED PYTHON
180ms MED JAVA
10ms MED C

OBSERVATION: VI BEHÖVER INTE SPARA HELA HISTORIEN, BARA SENASTE TVÅ VÄRDENA!

```
f0←1; f1←1
FOR i←2 TO n DO
  f2←f1+f0
  f0←f1
  f1←f2
RETURN f1
```

EXEMPEL 2: LÄNGSTA VÄXANDE DELFÖLJD

$a_1, a_2, \ldots, a_n$ FÖLJD TAL

HUR LÅNG ÄR DEN LÄNGSTA STRIKT VÄXANDE DELFÖLJDEN SOM INTE BEHÖVER VARA SAMMANHÄNGANDE?

8, 3, 9, 5, 6, 0, 17, 10, 2, 15

LÅT $q[k]$ VARA LÄNGDEN PÅ LÄNGSTA VÄXANDE DELFÖLJDEN SOM SLUTAR MED a_k

REKURSIV DEFINITION AV q:

$$q[k] = \max_{j:\, j<k,\, a_j<a_k} q[j]+1$$

SVARET ÄR $\max_{1\le k\le n} q[k]$

BERÄKNINGSORDNING FÖR $q[k]$: $k=1,2,3,\ldots,n$

```
totmax←0
FOR k←1 TO n DO
  qmax←0
  FOR j←1 TO k-1 DO
    IF a_j<a_k AND q[j]>qmax THEN
      qmax←q[j]
  q[k]←qmax+1
  IF q[k]>totmax THEN totmax←q[k]
RETURN totmax
```

OBS: HÄR MÅSTE HELA HISTORIEN SPARAS!

TIDSKOMPLEXITET: $O(n^2)$ (GÅR ATT GÖRA SNABBARE)

Exempel 3: Matriskedjemultiplikation

Matriskedjemultiplikation är problemet att hitta bästa sättet att multiplicera ihop en kedja av matriser av varierande storlek. Genom att multiplicera ihop matriserna med varandra i en listig ordning kan antalet operationer som krävs minimeras. Att multiplicera en $a \times b$-matris och en $b \times c$-matris med vanlig matrismultiplikation tar abc talmultiplikationer.

Anta att man vill beräkna matrisprodukten $A_1 A_2 \cdots A_n$ där matris A_i har dimension $r_{i-1} \times r_i$. Låt $M[i,j]$ stå för det minimala antalet multiplikationer som behövs för att räkna ut matrisprodukten $A_i A_{i+1} \cdots A_j$. Rekursionsekvationen för $M[i,j]$ är:

$$M[i,j] = \begin{cases} 0 & \text{om } i = j \\ \min_{i \le k < j}(M[i,k] + M[k+1,j] + r_{i-1} r_k r_j) & \text{om } i < j \end{cases}$$

Här behövs alltså en tvådimensionell struktur (matris) för att lagra värdena i dynamiskprogrammeringsdatastrukturen. Basfallen är $M[i,i]$, det vill säga diagonalen i (den triangulära) matrisen M.

Beräkningsordning: Beräkna elementen i efter stigande avstånd från diagonalen, alltså så att indexen i och j skiljer sig mer och mer.

```
OptimalMatmul(M[1..n, 1..n], r) =
    for i ← 1  to n do M[i, i] ← 0
    for len ← 1  to n − 1
        for i ← 1  to n − len
            j ← i + len
            M[i, j] ← ∞
            for k ← i+  to j − 1
                t ← M[i, k] + M[k + 1, j] + r_{i−1} · r_k · r_j
                if t < M[i, j] then M[i, j] ← t
```

Det minimala antalet multiplikationer beräknas av `OptimalMatmul(M, r)`. Svaret finns då i $M[1,n]$. Algoritmen har komplexiteten $O(n^3)$.

Man vill inte bara veta det minimala antalet multiplikationer utan också i vilken ordning man ska multiplicera ihop matriserna för att uppnå det minimala antalet. Låt oss bygga ut `OptimalMatmul` så att man kan få reda på ordningen. Lägg till en hjälpmatris $bryt[i,j]$ som innehåller var gränsen går mellan dom två matriserna som bildar produkten $A_i A_{i+1} \cdots A_j$. Exempel: $bryt[7,9] = 7$ betyder att man ska multiplicera $A_7(A_8 A_9)$ och $bryt[7,9] = 8$ betyder att man ska multiplicera $(A_7 A_8)A_9$.

Det enda vi behöver ändra i algoritmen är att göra tilldelningen $bryt[i,j] \leftarrow k$ sist på sista raden, dvs efter att värdet på $M[i,j]$ uppdaterats.

Skriv allra sist i algoritmen ut hur multiplikationen ska ske i form av ett parentesuttryck där matriserna symboliseras med A, till exempel $(((AA)A)(AA))$, genom att anropa nedanstående metod med SkrivParentetiserat$(1, n, bryt)$.

```
SkrivParentetiserat(a, b, bryt[1..n, 1..n]) =
    if a = b then Write('A')
    else
        Write('(')
        SkrivParentetiserat(a, bryt[a, b], bryt)
        SkrivParentetiserat(bryt[a, b] + 1, b, bryt)
        Write(')')
```