"Memoisierung"

Beispiel: Das Playoff-Problem

Teams A und B spielen ein "Playoff" um die Meisterschaft:

Es wird eine Folge von Spielen gespielt. Meister ist, wer zuerst n Spiele gewonnen hat.

Unter der Annahme, dass A jedes einzelne Spiel (unabhängig von vorherigen Spiele mit Wahrscheinlichkeit p gewinnt, was ist die Wahrscheinlichkeit, das A Meister wird?

M(i,j) sei die Wahrscheinlichkeit, A gewinnt i Spiele, bevor es j Spiele verliert

Die Antwort für das Playoff-Problem ist dann M(n,n).

$M(i,j)$ sei die Wahrscheinlichkeit, A gewinnt i Spiele, bevor es j Spiele verliert

Es gilt:

$M(i,0) = 0$ für $i>0$
$M(0,j) = 1$ für $j>0$
$M(0,0)$ nicht definiert
$M(i,j) = p\cdot M(i-1,j) + (1-p)\cdot M(i,j-1)$ sonst

Naive Berechnung von $M(i,j)$:

function $M(i,j)$

if $j=0$ **then** return 1
else if $i=0$ **then** return 0
else return $p\cdot M(i-1,j) + (1-p)\cdot M(i,j-1)$

Miserable Laufzeit!! Berechnung von $M(n,n)$ braucht mehr als $\Theta(4^n/n)$ Zeit, weil $M(\ )$ in den Rekursionen immer wieder für die gleichen Argumentpaare aufgerufen wird.

Verbesserung: schon berechnete Werte merken ("Memoisierung")

Naive Berechnung von $M(i,j)$:

function $M(i,j)$

if $j=0$ **then** return 1
else if $i=0$ **then** return 0
else return $p \cdot M(i-1,j) + (1-p) \cdot M(i,j-1)$

Schlaue Berechnung von $M(i,j)$ mit Hilfe von Memoisierung:

Verwende Feld $S[0..n,0..n]$ überall initialisiert auf undef

function $M(i,j)$

if $S[i,j]$ = undef **then**
if $j=0$ ~~**then**~~ return $S[i,j] = 1$
else if $i=0$ ~~**then**~~ return $S[i,j] = 0$
~~**else**~~ ~~re~~turn $S[i,j] = p \cdot M(i-1,j) + (1-p) \cdot M(i,j-1)$

return $S[i,j]$

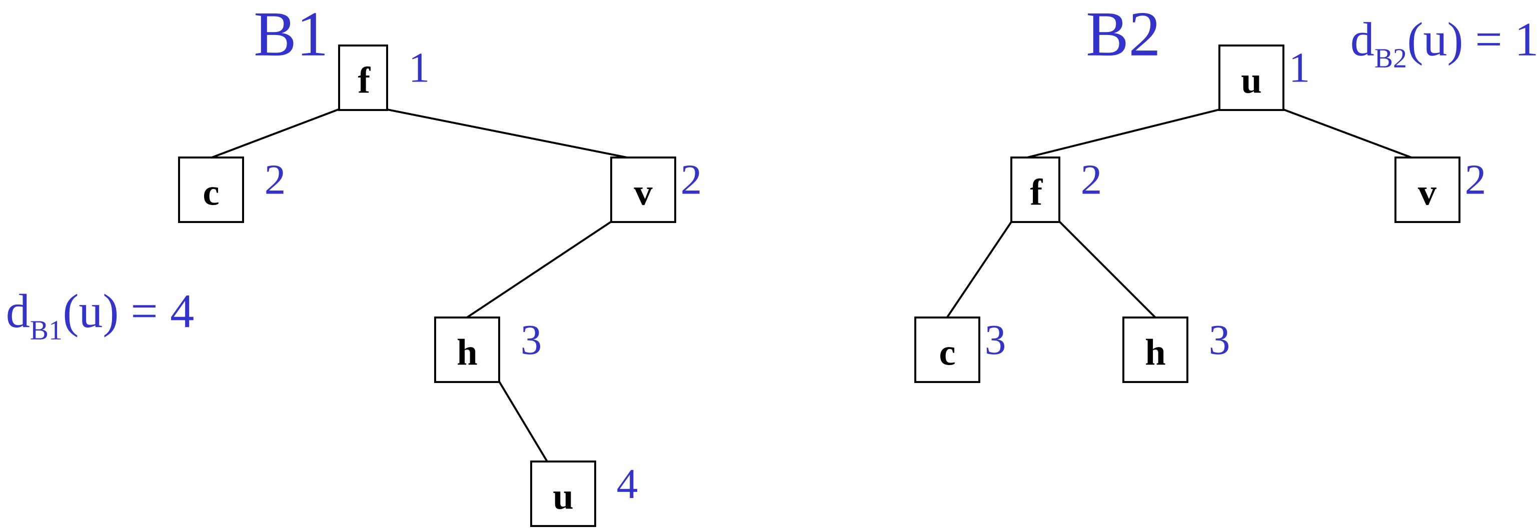
$O(n^2)$ Zeit

Definierende Aufrufe: höchstens $(n+1)^2$, jeder macht 2 Aufrufe und braucht ohne Rekursion $O(1)$ Zeit, also insgesamt $O(n^2)$ Zeit

Nicht definierende Aufrufe: höchstens $2(n+1)^2$, jeder braucht $O(1)$ Zeit, also insgesamt $O(n^2)$ Zeit

Problem: Baue einen binären Suchbaum B für die Schlüsselmenge {c,f,h,u,v}, sodass folgende Anfragefolge Q möglichst schnell beantwortet werden kann: c,u,v,u,u,c,f,u,v

Kosten für einzelne Anfrage nach Schlüssel x:
Anzahl der besuchten Baumknoten, also die Tiefe $d_B(x)$ in Baum B



Kosten von Q in B1:
2+4+2+4+4+2+1+4+2 = 25

Kosten von Q in B2:
3+1+2+1+1+3+2+1+2 = 16

Allgemeineres Problem:
Gegeben ist eine Menge $x_1 < x_2 < \cdots < x_n$ von n Schlüsseln und für jeden Schlüssel x_i eine Zugriffsfrequenz f_i.

Konstruiere einen binären Suchbaum B, so dass Anfragefolgen Q, die diesen Zugriffsfrequenzen genügen (also jedes x_i kommt genau f_i oft in Q vor) mit möglichst geringen Kosten abgearbeitet weden können.

Gesucht ist der Suchbaum B, für den

$$\text{cost}_B = \sum_{1 \leq i \leq n} f_i \cdot d_B(x_i)$$

minimal ist.

Gesucht ist der Suchbaum B, für den $cost_B = \sum_{1\leq i\leq n} f_i \cdot d_B(x_i)$ minimal ist.

Seien $C_{i,j}$ die minimalen Suchbaumkosten für die

Schlüsselmenge $X_{i,j} = \{x_i, x_{i+1}, \cdots, x_j\}$ (mit Zugriffsfrequenzen $f_i, \cdots, f_j$)

Der optimale Baum für $X_{i,j}$ hat irgendein x_k als Wurzel, mit $i \leq k \leq j$,

und der linke Teilbaum muss ein optimaler Baum für $X_{i,k-1}$ sein
und der rechte Teilbaum muss ein optimaler Baum für $X_{k+1,j}$ sein.

Die Kosten dieses Baums sind dann $f_{ij} + C_{i,k-1} + C_{k+1,j}$

mit $f_{ij} = f_i + f_{i+1} + \cdots + f_j$. (der Term f_{ij} zählt alle Besuche der Wurzel)

Es gilt

$C_{i,j} = 0$ für $i>j$

$C_{i,j} = f_i$ für $i=j$

$C_{i,j} = \min\{ f_{ij} + C_{i,k-1} + C_{k+1,j} \mid i \leq k \leq j\}$ für $i<j$

Naive Berechnung von $C_{i,j}$: (mit Zugriffsfrequenzarray f[1..n])

function C(i,j)

$F[k] = \sum_{0 \leq h \leq k} f[h]$

$f_{ij} = F[j]-F[i-1]$

```
    if i>j then return 0
    else if i=j then return f[i]
    else m := ∞
        for k = i to j do m:= min( m , F[j]-F[i-1] + C(i,k-1) + C(k+1,j) )
        return m
```

Schlaue Berechnung von C(i,j) mit Hilfe von Memoisierung:

Verwende Feld S[1..n,1..n] überall initialisiert auf undef

function C(i,j)

"Dynamisches Programmieren"

```
    if S[i,j] = undef then
            if i>j ~~then~~ return S[i,j] := 0
            else if i=j ~~then~~ return S[i,j]:= f[i]
            else m := ∞
                for k = i to j do m:= min( m , F[j]-F[i-1] + C(i,k-1) + C(k+1,j) )
            ~~return~~ S[i,j] := m

    return S[i,j]
```

Schlaue Berechnung von $C(i,j)$ mit Hilfe von Memoisierung:

Verwende Feld $S[1..n,1..n]$ überall initialisiert auf undef

function $C(i,j)$

```
if S[i,j] = undef then
        if i>j ~~then~~ return S[i,j] := 0
        else if i=j ~~then~~ return S[i,j]:= f[i]
        else m := ∞
            for k = i to j do m:= min( m , F[j]-F[i-1] + C(i,k-1) + C(k+1,j) )
        ~~return~~ S[i,j] := m

return S[i,j]
```

Laufzeitanalyse: definierender Aufruf $O(n)$ Zeit (ohne Rekursion)
nicht definierender Aufruf $O(1)$ Zeit

Anzahl der definierenden Aufrufe $\leq n^2$
Anzahl der nicht definierenden Aufrufe $\leq 1 + n^3$ ($\leq n$ pro definierenden Aufruf)

Gesamtzeit: $O(n^3)$ Speicher $O(n^2)$