Dynamische Programmierung und "Memoisierung"

Beispiel: Kürzester gemeinsamer Teilstring

String $X=x_1x_2x_3 \ldots x_n$ **Teilstring** $x_{j_1}x_{j_2}\ldots x_{j_k}$ mit $j_1<j_2<\ldots<j_k$

Beispiel: bara ist Teilstring von abrakadabra

Finde einen längsten gemeinsamen Teilstring von $X=x_1x_2x_3 \ldots x_m$ und $Y=y_1y_2y_3 \ldots y_n$

Bsp: arakba ist längster gem.Teilstring von abrakadabra und barackobama

LCS($x_1x_2 \ldots x_m$, $y_1y_2 \ldots y_n$) =

if $m=0$ or $n=0$ then leerer String
else if $x_m=y_n$ then LCS($x_1x_2 \ldots x_{m-1}$, $y_1y_2 \ldots y_{n-1}$) y_n
else der längere von LCS($x_1 \ldots x_{m-1}$, $y_1 \ldots y_n$) und LCS($x_1 \ldots x_m$, $y_1 \ldots y_{n-1}$)

LCS … „longest common substring“

Vereinfachung: Finde die **Länge** des längsten gemeinsamen Teilstring von X und Y

$\mathrm{LLCS}(x_1x_2\ldots x_m, y_1y_2\ldots y_n) =$

if $m=0$ or $n=0$ then 0
else if $x_m=y_n$ then $\mathrm{LLCS}(x_1x_2\ldots x_{m-1}, y_1y_2\ldots y_{n-1}) + 1$
else max{ $\mathrm{LLCS}(x_1\ldots x_{m-1}, y_1\ldots y_n)$, $\mathrm{LLCS}(x_1\ldots x_m, y_1\ldots y_{n-1})$ }

Einfache rekursive Berechnung, aber naiv, wil Laufzeit groß!!

Laufzeitabschätzung: $T(m,n) \geq 1$ wenn $m=0$ oder $n=0$
$\geq 1 + \max\{ T(m-1,n-1), T(m-1,n)+T(m,n-1) \}$ sonst

$\varphi(m,n) = 1$ wenn $m=0$ oder $n=0$
$\varphi(m-1,n)+\varphi(m,n-1)$ sonst

$T(m,n) \geq \varphi(m,n) = \binom{m+n}{m}$

$T(n,n) \geq \varphi(n,n) = \binom{2n}{n} > 2^{2n}/(2n+1)$

exponentiell!!

Wo liegt das Problem? LLCS() wird in der Rekursion für die gleichen Parameter immer wieder aufgerufen und redundant neu berechnet.

Lösung: schon berechnete Ergebnisse merken („Memoisierung“)

Naive Berechnung von LLCS:

LLCS($x_1 x_2 \ldots x_m$, $y_1 y_2 \ldots y_n$) =

if $m=0$ or $n=0$ then 0
else if $x_m = y_n$ then LLCS($x_1 x_2 \ldots x_{m-1}$, $y_1 y_2 \ldots y_{n-1}$) $+1$
else max{ LLCS($x_1 \ldots x_{m-1}$, $y_1 \ldots y_n$) , LLCS($x_1 \ldots x_m$, $y_1 \ldots y_{n-1}$) }

Schlaue Berechnung von LLCS mit Memoisierung.

Verwendet Ergebnisfeld `S[0..m,0..n]` überall initialisiert auf `undef`

LLCS($x_1 x_2 \ldots x_m$, $y_1 y_2 \ldots y_n$) =
if `S[m,n]` = `undef` then
`S[m,n]` := if $m=0$ or $n=0$ then 0
else if $x_m = y_n$ then LLCS($x_1 x_2 \ldots x_{m-1}$, $y_1 y_2 \ldots y_{n-1}$) $+1$
else max{ LLCS($x_1 \ldots x_{m-1}$, $y_1 \ldots y_n$) , LLCS($x_1 \ldots x_m$, $y_1 \ldots y_{n-1}$) }

return `S[m,n]`

Laufzeitanalyse:

Verwendet Ergebnisfeld `S[0..m,0..n]` überall initialisiert auf `undef`.

LLCS($x_1x_2 \ldots x_m$, $y_1y_2 \ldots y_n$) =
    if `S[m,n]` = `undef` then
        `S[m,n]` := if $m=0$ or $n=0$ then 0
            else if $x_m=y_n$ then LLCS($x_1x_2 \ldots x_{m-1}$, $y_1y_2 \ldots y_{n-1}$) $+1$
            else max{ LLCS($x_1 \ldots x_{m-1}$, $y_1 \ldots y_n$) , LLCS($x_1 \ldots x_m$, $y_1 \ldots y_{n-1}$) }

    return `S[m,n]`

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

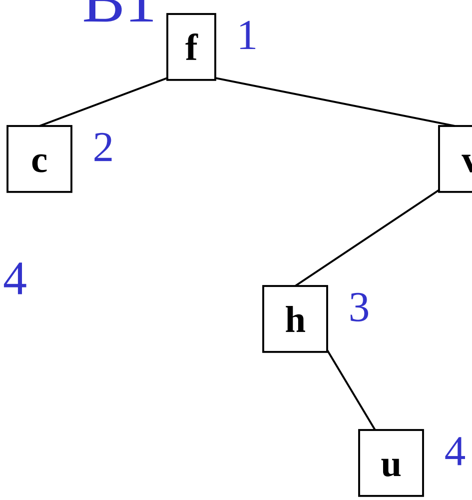
Nicht definierende Aufrufe: höchstens $2(m+1)(n+1)$, (weil höchstens 2 pro definierenden Aufruf) und jeder braucht $O(1)$ Zeit, also insgesamt $O(mn)$ Zeit.

Gesamtzeit: $O(mn)$ Speicher $O(mn)$

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

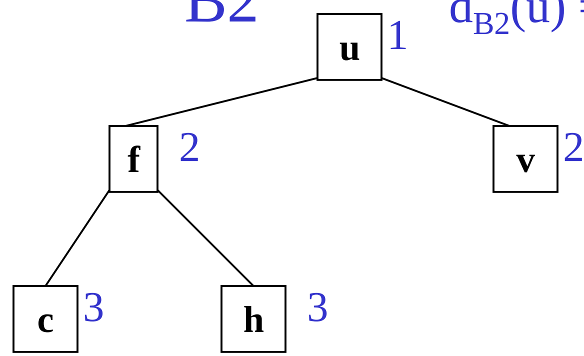
B1



$d_{B1}(u) = 4$

B2

$d_{B2}(u) = 1$



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

$$cost_B = \sum_{1 \le i \le n} f_i \cdot d_B(x_i)$$

minimal ist.

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

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

$$C_{i,j} = 0 \quad \text{für } i>j$$
$$C_{i,j} = f_i \quad \text{für } i=j$$
$$C_{i,j} = \min\{ f_{ij} + C_{i,k-1} + C_{k+1,j} \mid i \le k \le j\} \quad \text{für } i<j$$

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 $O(n^2)$