Algorithmen und Datenstrukturen II

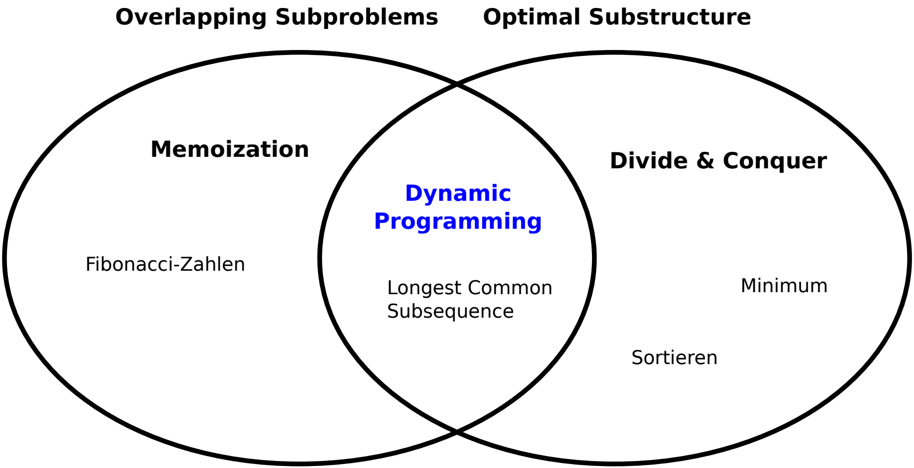
Übung 1

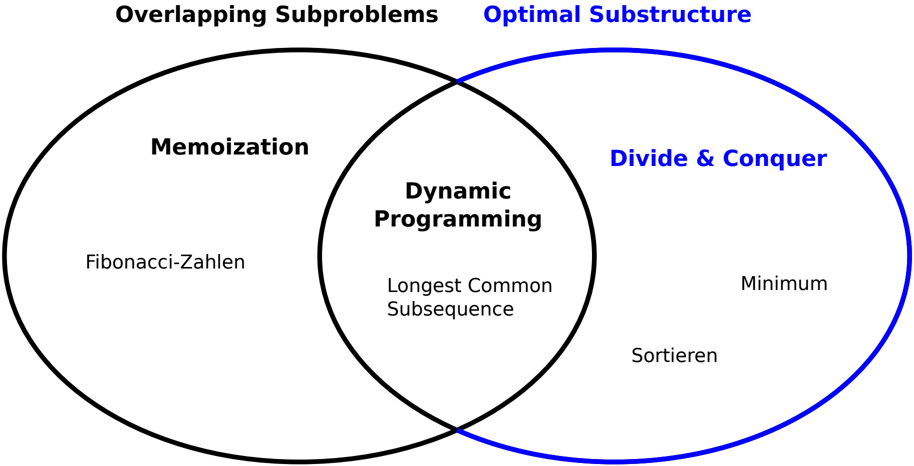
Stephan Friedrichs

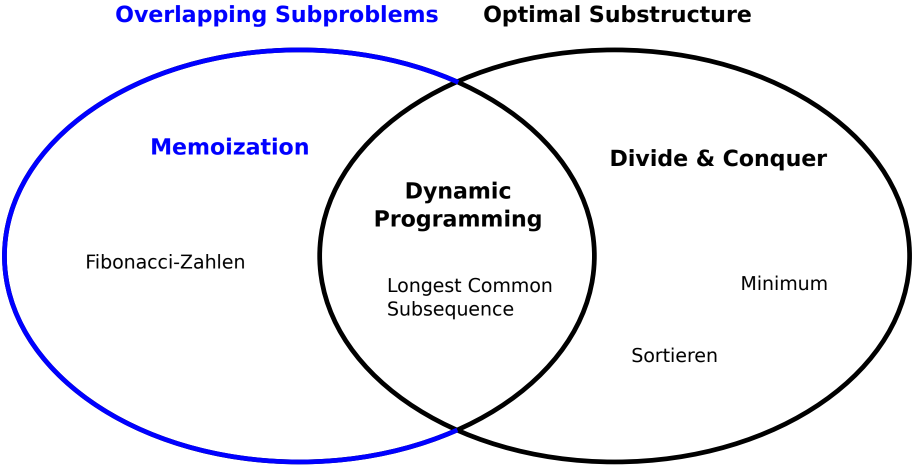
Technische Universität Braunschweig, IBR

24. April 2013

Dynamic Programming


Overlapping Subproblems
Optimal Substructure
Memoization
Dynamic Programming
Divide & Conquer
Fibonacci-Zahlen
Longest Common Subsequence
Minimum
Sortieren


Overlapping Subproblems
Optimal Substructure
Memoization
Fibonacci-Zahlen
Dynamic Programming
Longest Common Subsequence
Divide & Conquer
Minimum
Sortieren


Overlapping Subproblems
Optimal Substructure
Memoization
Dynamic Programming
Divide & Conquer
Fibonacci-Zahlen
Longest Common Subsequence
Minimum
Sortieren

Algorithmus 1 – Rekursiv

$F_0 = 0$
$F_1 = 1$
$F_i = F_{i-1} + F_{i-2}$ für $i \geq 2$

```
function FIB(i)
    if i ≤ 1 then
        return i
    else
        return FIB(i − 1) + FIB(i − 2)
    end if
end function
```

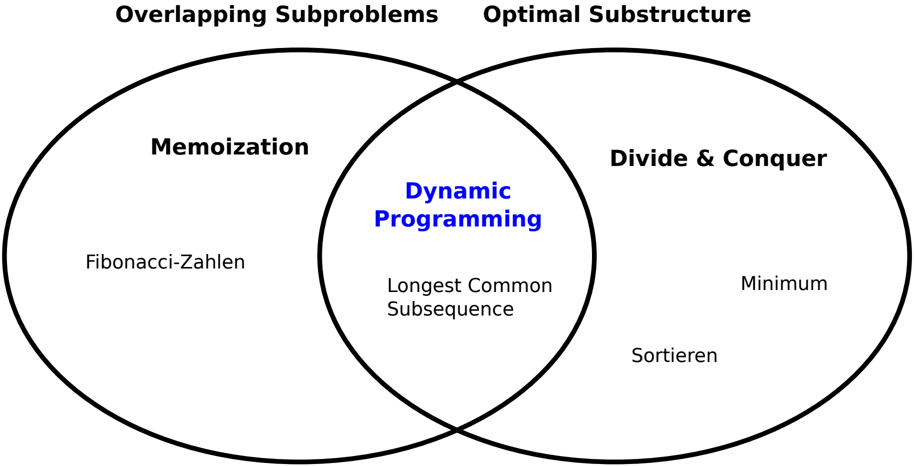
Algorithmus 2 – Top-Down

$F_0 = 0$, $F_1 = 1$, $F_i = F_{i-1} + F_{i-2}$ für $i \geq 2$

```
function FIB(i)
    F ← array[0 ... i]
    F[0] ← 0
    F[1] ← 1
    F[2] ← ··· ← F[i] ← ∞
    return FIB_HELP(i, F)
end function
function FIB_HELP(i, F)
    if F[i] = ∞ then
        F[i] ← FIB_HELP(i − 1, F) + FIB_HELP(i − 2, F)
    end if
    return F[i]
end function
```

Algorithmus 3 – Bottom-Up

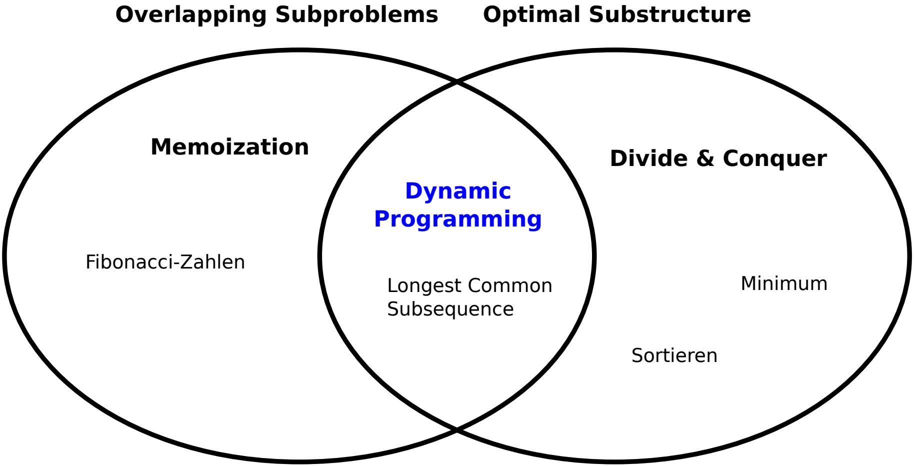
```
function FIB(i)
    if i ≤ 1 then
        return i
    else
        f_prev ← 0
        f ← 1
        for k ← 2, . . . , i do
            f_next ← f + f_prev
            f_prev ← f
            f ← f_next
        end for
    end if
    return f
end function
```


Overlapping Subproblems
Optimal Substructure
Memoization
Dynamic Programming
Divide & Conquer
Fibonacci-Zahlen
Longest Common Subsequence
Minimum
Sortieren

Algorithmus 4

```
function LCS(a_1 ... a_n, b_1 ... b_m)
    C ← array[0 ... n][0 ... m]
    C[0,0] ← ··· ← C[n,0] ← 0
    C[0,1] ← ··· ← C[0,m] ← 0

    for i ← 1 ... n do
        for j ← 1 ... m do
            if a_i = b_j then
                C[i,j] ← C[i−1, j−1] + 1
            else
                C[i,j] ← max{C[i−1, j], C[i, j−1]}
            end if
        end for
    end for
    return C[n,m]
end function
```


Overlapping Subproblems
Optimal Substructure
Memoization
Dynamic Programming
Divide & Conquer
Fibonacci-Zahlen
Longest Common Subsequence
Minimum
Sortieren