DSAL - 11. Globalübung

David Korzeniewski, Tim Quatmann

10. Juli 2018

Agenda

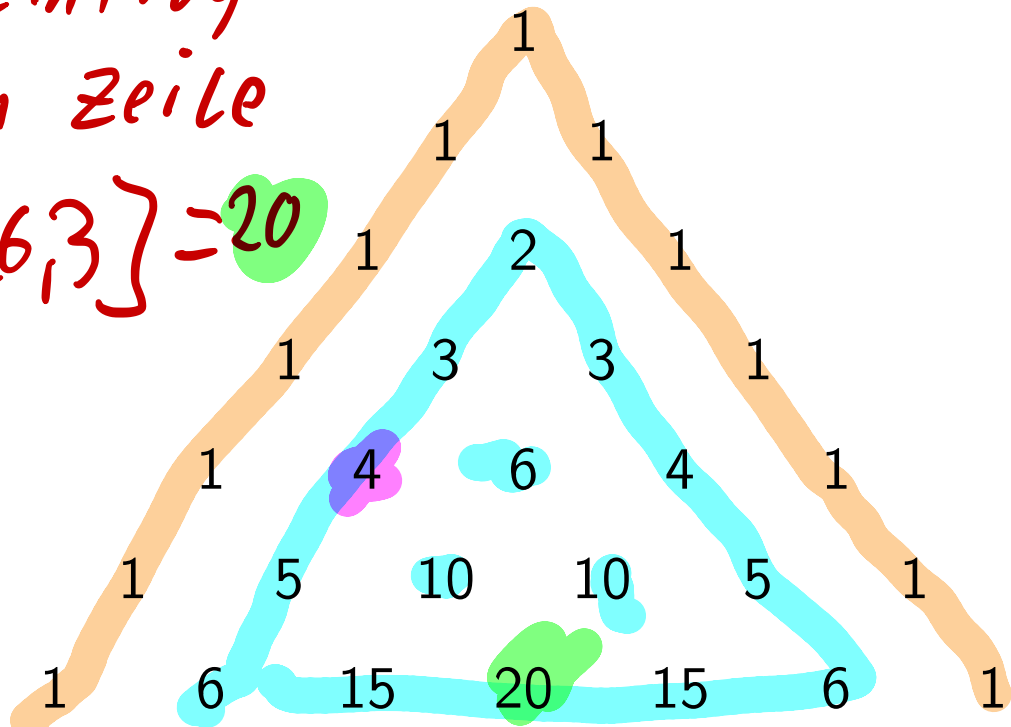
1. Dynamische Programmierung
   - Pascalsches Dreieck
   - Maximal Sum Decreasing Subsequence

Pascalsches Dreieck

$C[i,j]$ = "j-ter Eintrag in der i-ten Zeile

$C[4,1] = 4$ $C[6,3] = 20$

$C[n,k] = \binom{n}{k}$



Rekursionsgleichung:

$$C[i,j] = \begin{cases} C[i-1,j-1] + C[i-1,j] & , \text{falls } 0 < j < i \\ 1 & , \text{falls } j = 0 \vee j = i \end{cases}$$

Rekursionsgleichung

$$C[i,j] = \begin{cases} C[i-1,j-1] + C[i-1,j] & \text{, falls } 0 < j < i \\ 1 & \text{, falls } j = 0 \text{ oder } j = i \end{cases}$$

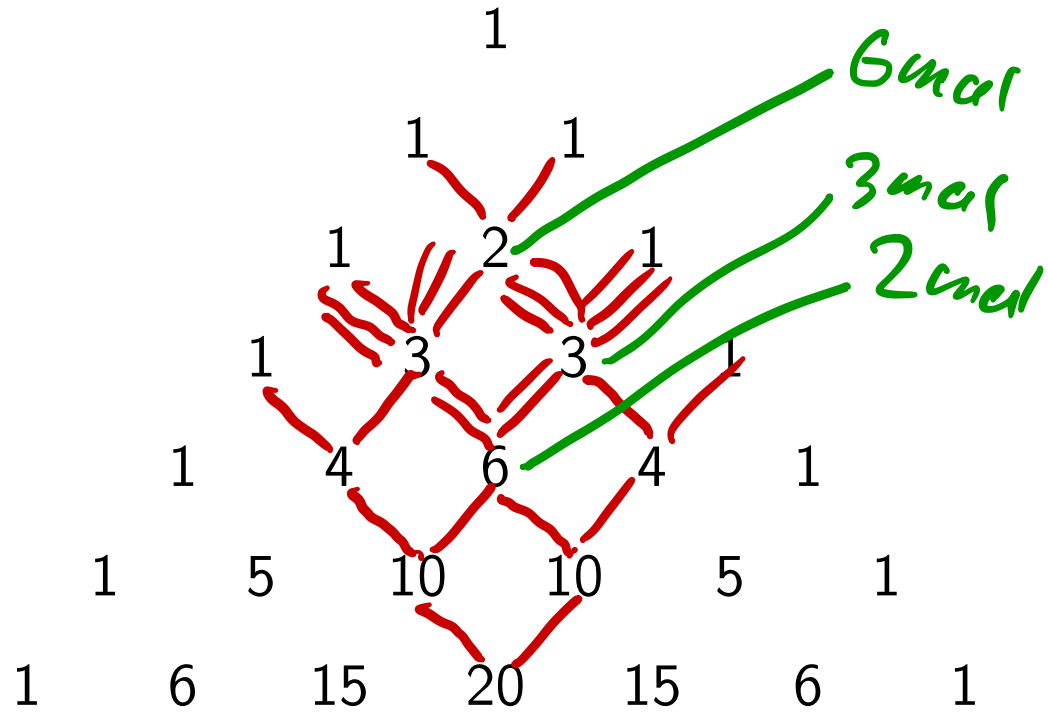
```
// Berechnet C[n,k] unter der Vorbedingung 0<=k<=n.
int f(int n, int k) {
   if (0<k && k<n){
      return f(n-1,k-1) + f(n-1, k);
  }else{
      return 1;
  }

}
```

Dieser Algorithmus benutzt nicht das Prinzip der dynamischen Programmierung

Beispiel $f(6,3)$

+ Speicher in $O(n)$

- Berechnet Werte mehrmals



Rekursionsgleichung

$$C[i,j] = \begin{cases} C[i-1,j-1] + C[i-1,j] & \text{, falls } 0 < j < i \\ 1 & \text{, falls } j = 0 \text{ oder } j = i \end{cases}$$

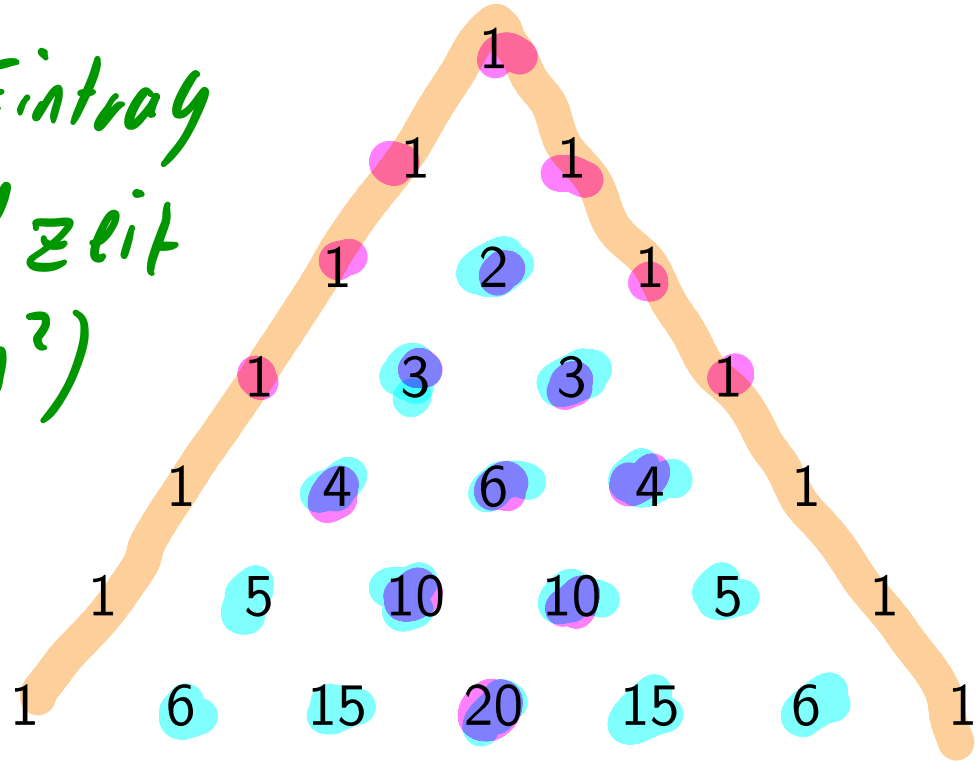
```
// Berechnet C[n,k] unter der Vorbedingung 0<=k<=n.
int g(int n, int k) {
  int C[n+1][n+1] = 1;
  for (int i=2; i<=n; i++){
    for (int j=1; j<i; j++){
      C[i][j] = C[i-1][j-1]+C[i-1][j];
    }
  }
  return C[n][k];
}
```

Dieser Algorithmus benutzt das Prinzip der dynamischen Programmierung

Beispiel $g(6,3)$:

+ Berechnet jeden Eintrag nur einmal $\rightsquigarrow$ Laufzeit in $O(n^2)$

− Speicher in $\Omega(n^2)$



Rekursionsgleichung

$$C[i,j] = \begin{cases} C[i-1,j-1] + C[i-1,j] & \text{, falls } 0 < j < i \\ 1 & \text{, falls } j = 0 \text{ oder } j = i \end{cases}$$

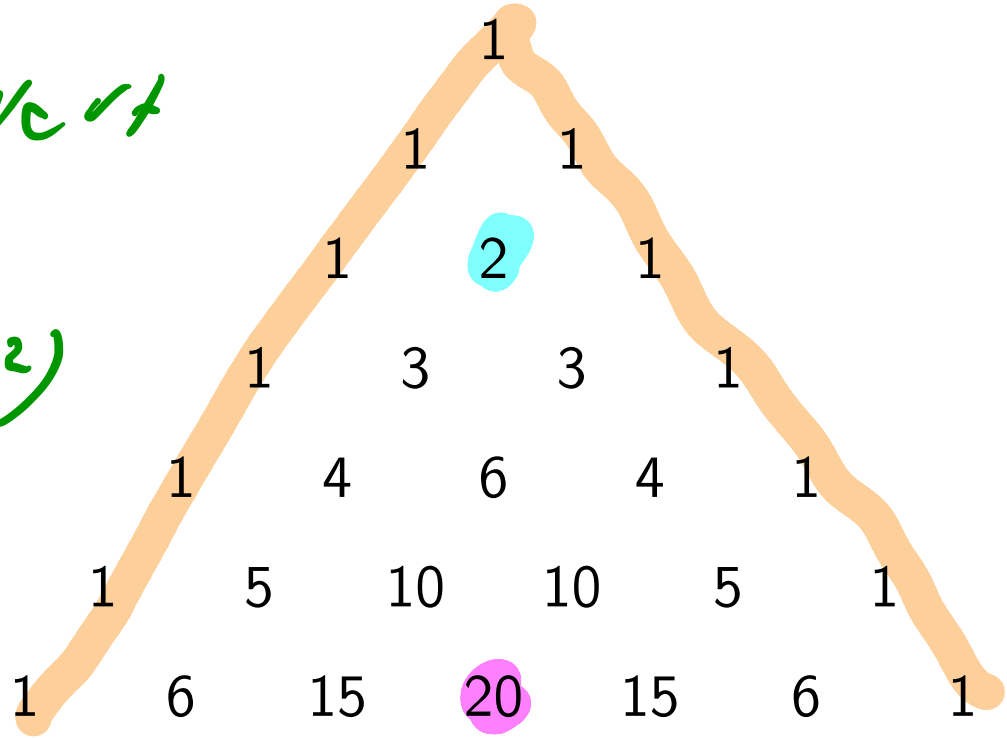
```
// Berechnet C[n,k] unter der Vorbedingung 0<=k<=n.
int h(int n, int k) {
    int C[n+1] = 1;
    for (i=2; i <= n; i++) {
        for (j=i-1; j>0; j--) {
            C[j] = C[j-1] + C[j];
        }
    }
  return C[k];
}
```

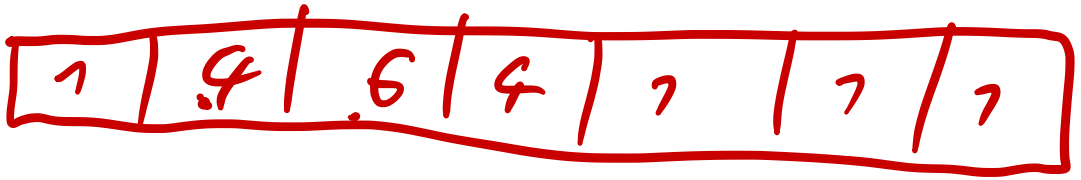
Dieser Algorithmus benutzt das Prinzip der dynamischen Programmierung

Beispiel: $n(6,3)$:

+ Berechnet jeden Wert nur einmal
  ⇝ Laufzeit in $O(n^2)$

+ Speichert nur eine Zeile des Dreiecks
  ⇝ Speicher in $O(n)$





Maximal Sum Decreasing Subsequence

Gegeben ein Array mit Integern

Teilmenge der Elemente, die absteigend im Array stehen

↳ Wähle Teilmenge so, dass die Summe der Elemente maximal ist

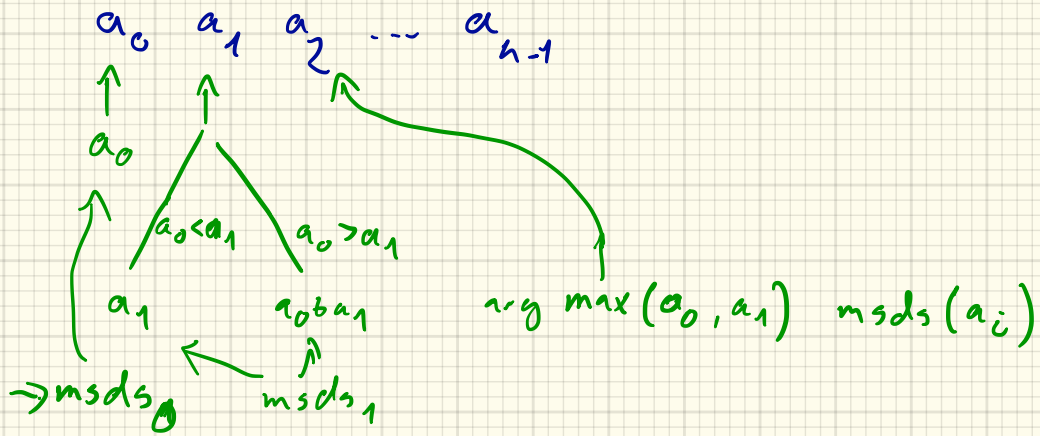
Output: die maximale Summe

Bsp: 10 5 4 7 6

10 + 7 + 6 = 23

5 4 100 3 2 101 1

106



maximale summe decreasing subsequence, sodass die sequenz in 0 endet

$msds_2$ ——//—— in 1 endet

$$MSDS_i := \begin{cases} a_i + \max MSDS_j & | \; 0 < j < i \wedge a_i < a_j \\ a_i & | \; \text{wenn kein solches } j \text{ existiert} \end{cases}$$

Bsp:

3 2 1 4

$MSDS_0 = 3$

$MSDS_1 = 3 + 2 = 5$

$MSDS_2 = 5 + 1 = 6$

$MSDS_3 = 4$

$$MSDS(a) = \max_{0 \le i < n} (MSDS_i)$$

4 5 1 100 99

$msds_2 = 6$

input: array a länge n

Speicher $O(n+1)$
$= O(n)$

```
for i = 0 ... n-1:
    msds[i] = a[i]
```

} msds = copy(a) $O(n)$

```
for i = 0 ... n-1:
  for j = 0 ... i-1:
    if (a[i] < a[j] &&
        msds[i] < msds[j] + a[i])
      msds[i] = msds[j] + a[i]
```

} $O(j)$ $(j < n-1)$ $O(n)$ → $O(n^2)$

```
max = 0
for i = 0 ... n-1:
    if (max < msds[i])
        max = msds[i]
return max
```

$O(n)$

Insgesamt $O(n^2)$

Nächster Termin

Nächste Vorlesung

null

Nächste Globalübung

Dienstag, 17. Juli, 14:15 (Aula 1).
Thema: Klausur vorrechnen