Dynamische Programmierung

und die Verwendung in Programmierwettbewerben

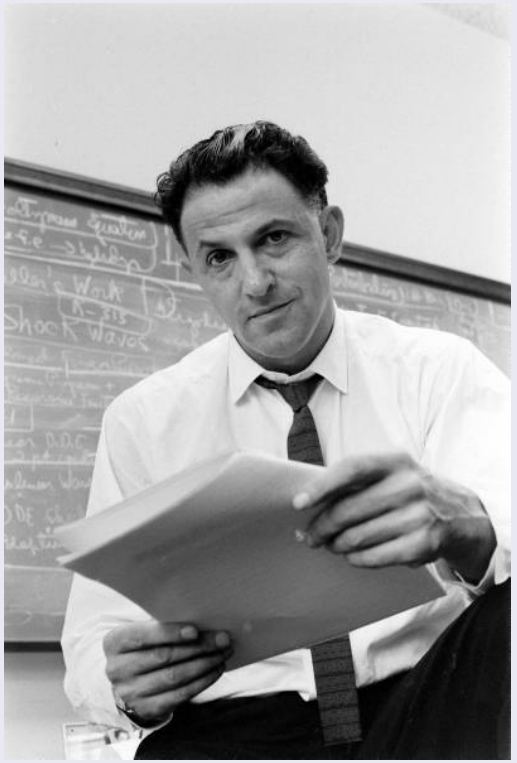
Sandro Feuz

Entstehung der Dynamischen Programmierung

Entstehung

- Erstmal *Bellman* 1940-50 als Forscher bei RAND (**R**esearch **AN**d **D**evelopment).
- *Dynamisch* im Sinne von mehrstufig.
- *Programmierung* im Sinne von Planung (z.B. von militärischen Abläufen).

Richard Bellman



Dynamischen Programmierung ein einführendes Beispiel

$$0, 1, 1, 2, 3, 5, 8, 13, \ldots$$

Dynamischen Programmierung ein einführendes Beispiel

$$0, 1, 1, 2, 3, 5, 8, 13, \ldots$$

$$F_n = F_{n-1} + F_{n-2}, \text{ for } n \geq 2$$

$$F_0 = 0$$

$$F_1 = 1$$

Fibonacci-Zahlen

```
def fib(n):
    if n < 2: return n
    return fib(n-1) + fib(n-2)
```

Laufzeitanalyse

Fibonacci-Zahlen

```
def fib(n):
    if n < 2: return n
    return fib(n-1) + fib(n-2)
```

Laufzeitanalyse

Laufzeit von fib(n): $\Theta(\sim 1.6^n)$.

Für $n = 100$:

Fibonacci-Zahlen

```
def fib(n):
    if n < 2: return n
    return fib(n-1) + fib(n-2)
```

Laufzeitanalyse

Laufzeit von fib(n): $\Theta(\sim 1.6^n)$.

Für $n = 100$:

$1.6^{100} \geq 10000000000000000000$ Operationen.

≥ 10000000000 Sekunden.

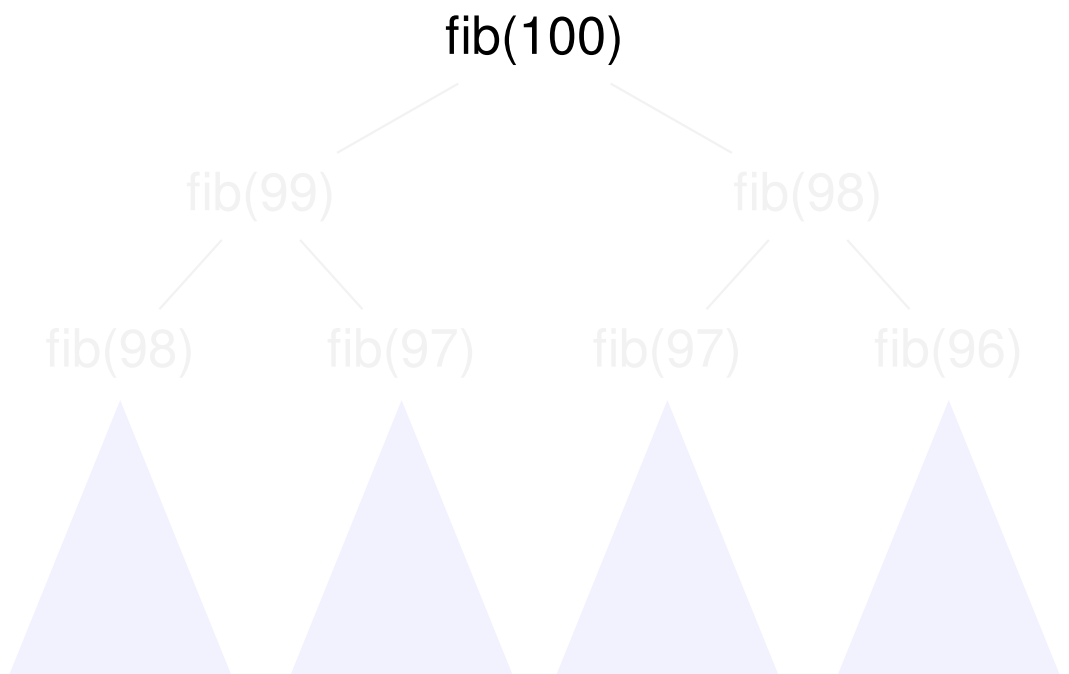
≥ 1000000 Stunden.

≥ 10000 Tage.

≥ 10 Jahre.

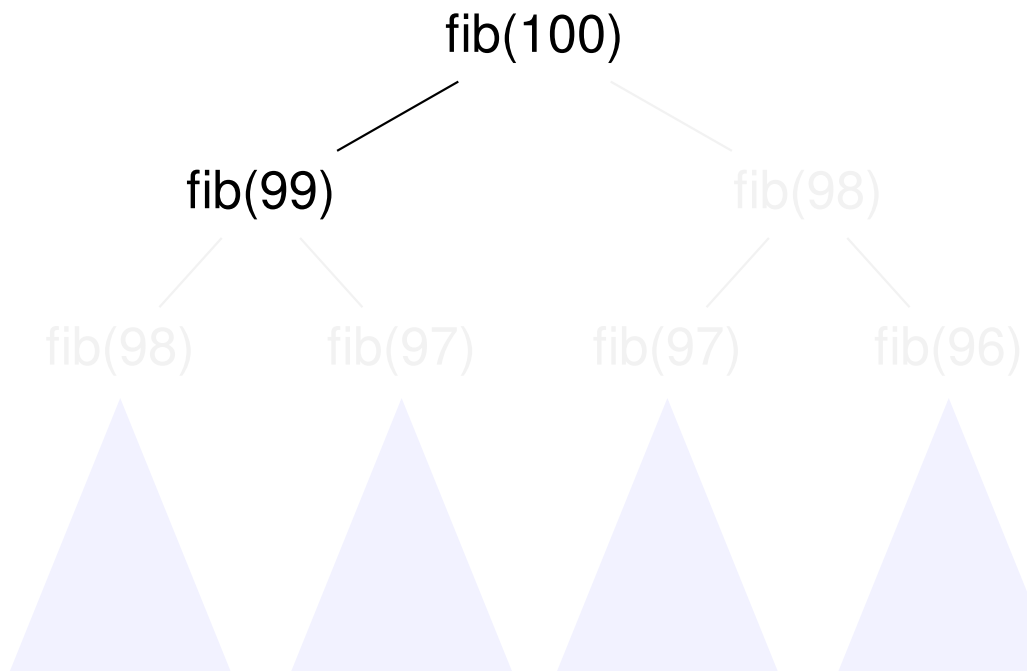
Fibonacci-Zahlen

```
def fib(n):
    if n < 2: return n
    return fib(n-1) + fib(n-2)
```



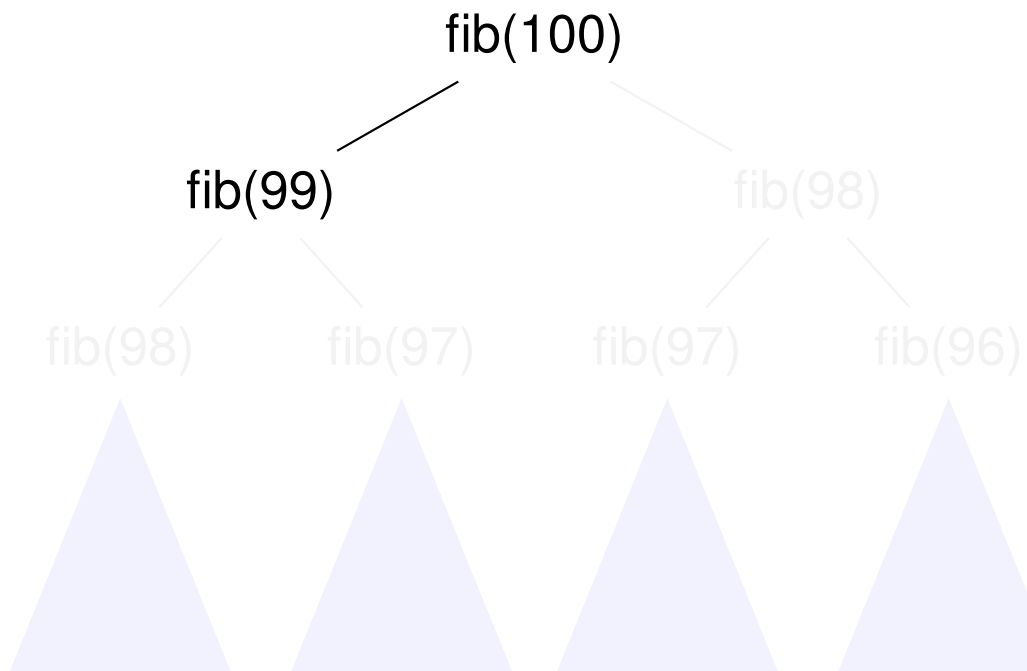
Fibonacci-Zahlen

```
def fib(n):
    if n < 2: return n
    return fib(n-1) + fib(n-2)
```



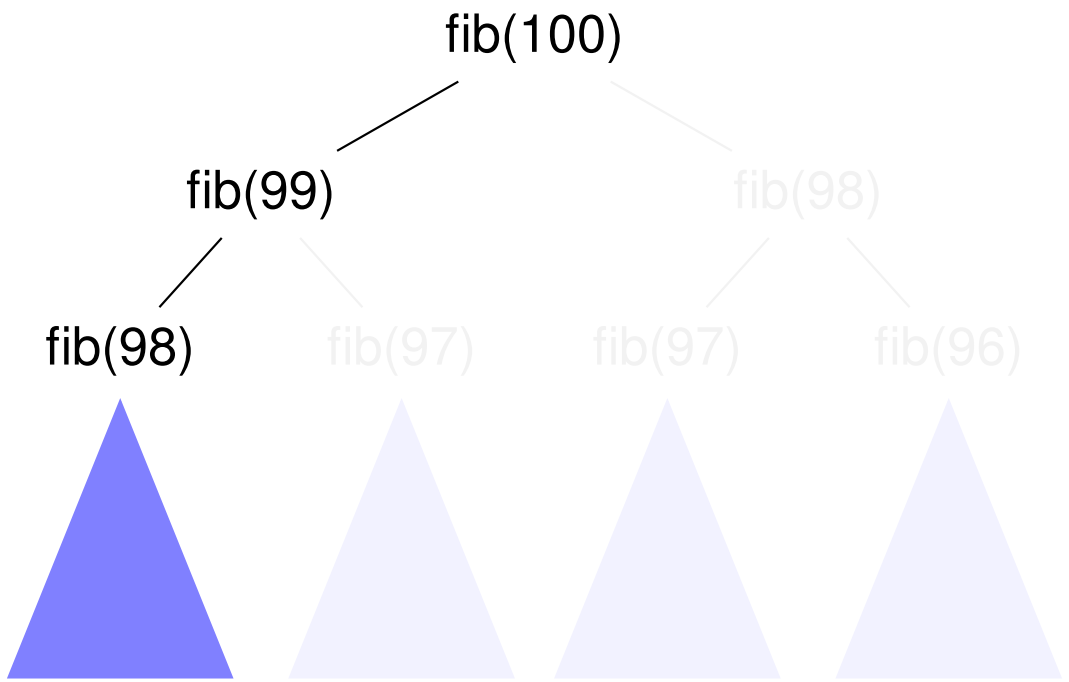
Fibonacci-Zahlen

```
def fib(n):
    if n < 2: return n
    return fib(n-1) + fib(n-2)
```



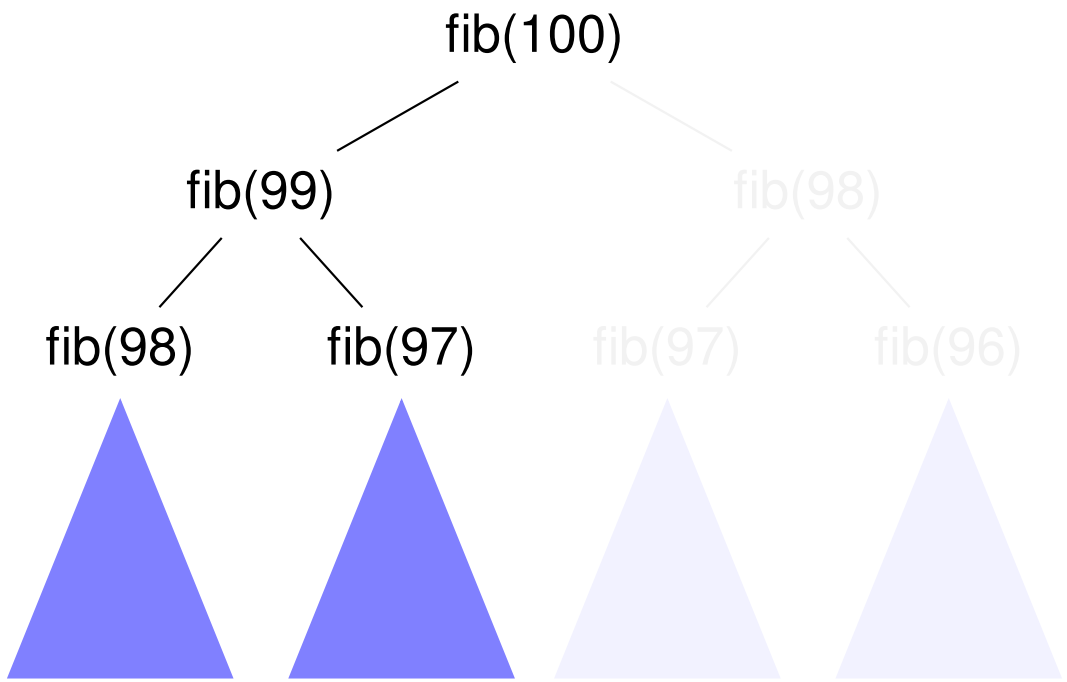
Fibonacci-Zahlen

```
def fib(n):
    if n < 2: return n
    return fib(n-1) + fib(n-2)
```



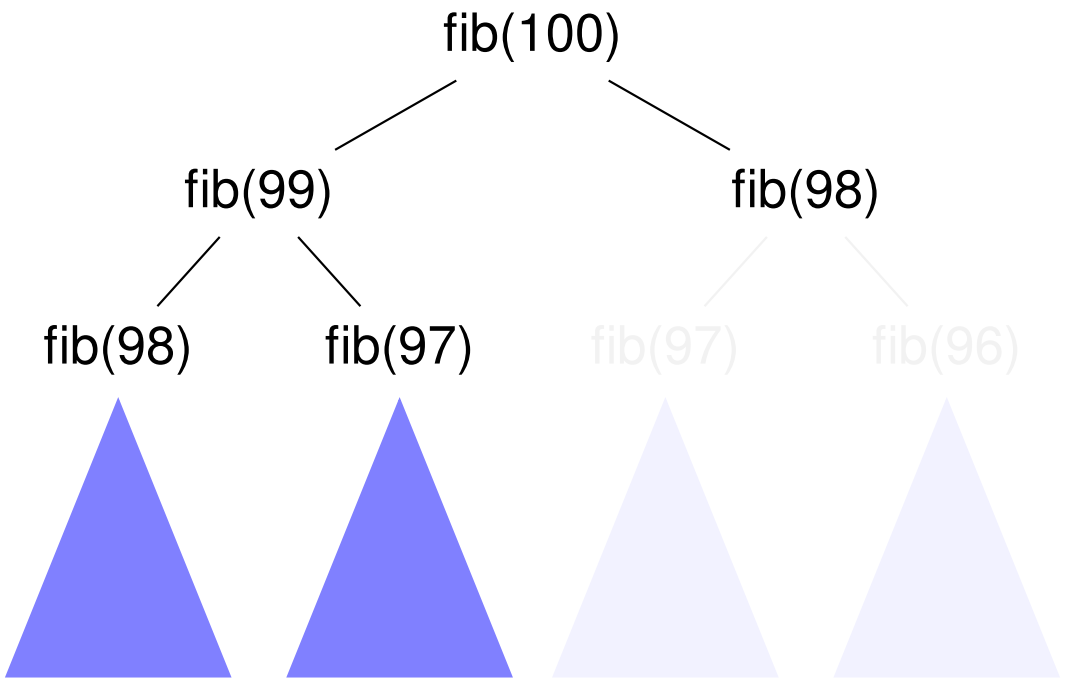
Fibonacci-Zahlen

```
def fib(n):
    if n < 2: return n
    return fib(n-1) + fib(n-2)
```



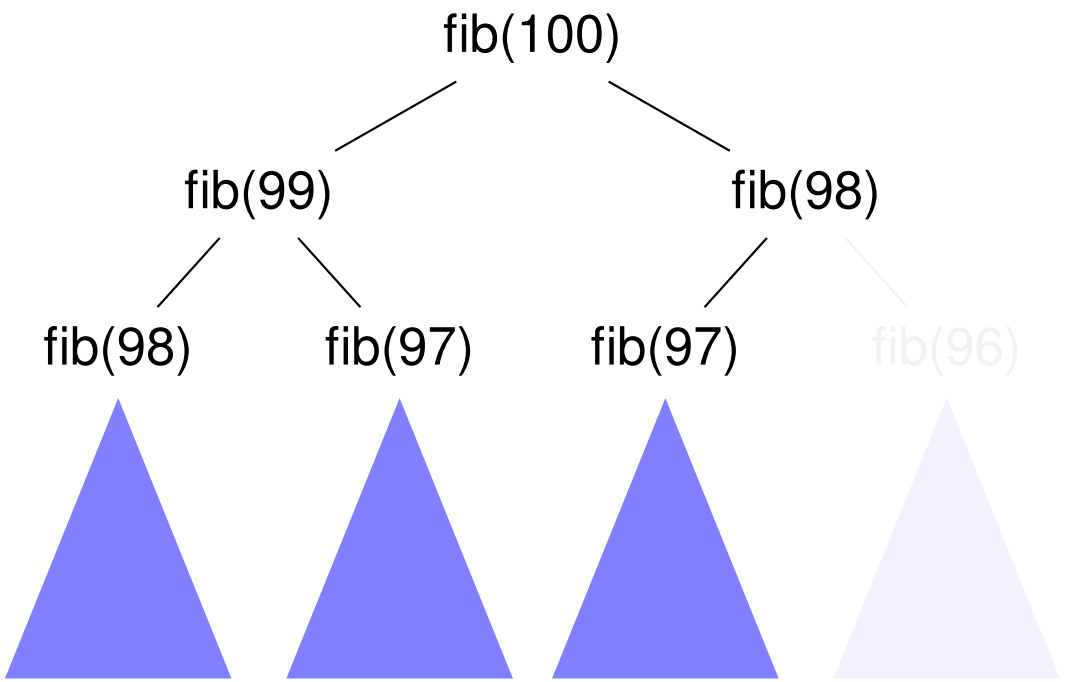
Fibonacci-Zahlen

```
def fib(n):
    if n < 2: return n
    return fib(n-1) + fib(n-2)
```



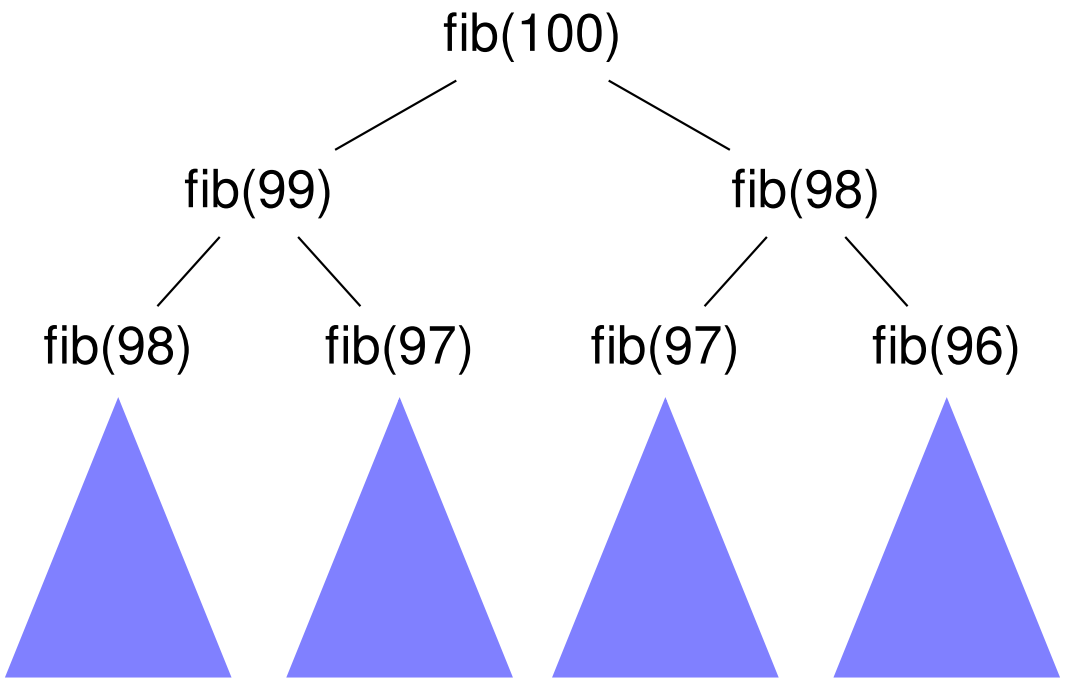
Fibonacci-Zahlen

```
def fib(n):
    if n < 2: return n
    return fib(n-1) + fib(n-2)
```



Fibonacci-Zahlen

```
def fib(n):
    if n < 2: return n
    return fib(n-1) + fib(n-2)
```



Fibonacci mit Dynamischer Programmierung

```
def fib(n):
    if n < 2: return n
    return fib(n-1) + fib(n-2)
```

Idee - Dont calculate the same thing twice!

Jedes fib(n) abspeichern sobald berechnet.

Fibonacci mit Dynamischer Programmierung

```
def fib(n):
    if n < 2: return n
    return fib(n-1) + fib(n-2)
```

Idee - Dont calculate the same thing twice!

Jedes fib(n) abspeichern sobald berechnet.

```
gesp = {}
def fib(n):
    if n < 2: return n
    if n not in gesp: gesp[n] = fib(n-1) + fib(n-2)
    return gesp[n]
```

Fibonacci mit Dynamischer Programmierung

```
def fib(n):
    if n < 2: return n
    return fib(n-1) + fib(n-2)
```

Idee - Dont calculate the same thing twice!

Jedes fib(n) abspeichern sobald berechnet.

```
gesp = {}
def fib(n):
    if n < 2: return n
    if n not in gesp: gesp[n] = fib(n-1) + fib(n-2)
    return gesp[n]
```

Komplexität

Speicher- und Laufzeitkomplexität: Linear $\Theta(n)$.

Dynamischer Programmierung

Voraussetzungen an das Problem

- (Optimale) Lösung besteht aus (optimalen) Lösungen von Teilproblemen des gleichen Typs.
- Die Anzahl Teilprobleme ist nicht zu gross.

Dynamischer Programmierung

Voraussetzungen an das Problem

- (Optimale) Lösung besteht aus (optimalen) Lösungen von Teilproblemen des gleichen Typs.
- Die Anzahl Teilprobleme ist nicht zu gross.

Die 4 DP-Schritte [Cormen et al.]

1. Die Struktur einer optimalen Lösung beschreiben.
2. Rekursive Formulierung einer optimalen Lösung.
3. Berechnen des Wertes einer optimalen Lösung.
4. Rekonstruieren der optimalen Lösung.

Max Triangle Sum Path Problem

Maximum Triangular Sum Path [IOI 1994]

Gegeben ist eine Zahlenpyramide der Höhe n. Gesucht ist ein maximaler Pfad von der Spitze zur Basis, wobei in jedem Schritt diagonal nach links oder rechts gezogen wird.

Max Triangle Sum Path Problem

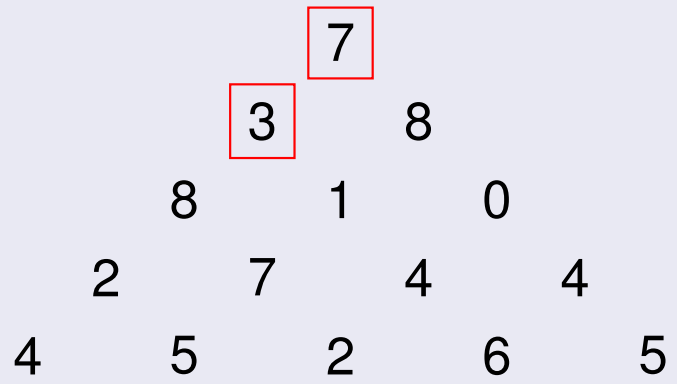
Maximum Triangular Sum Path [IOI 1994]

Gegeben ist eine Zahlenpyramide der Höhe n. Gesucht ist ein maximaler Pfad von der Spitze zur Basis, wobei in jedem Schritt diagonal nach links oder rechts gezogen wird.

Max Triangle Sum Path Problem

Maximum Triangular Sum Path [IOI 1994]

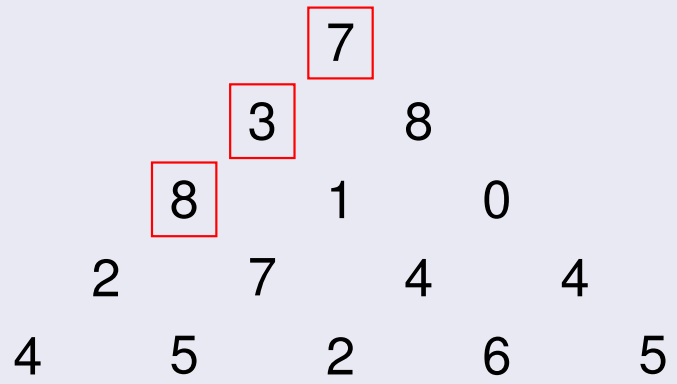
Gegeben ist eine Zahlenpyramide der Höhe n. Gesucht ist ein maximaler Pfad von der Spitze zur Basis, wobei in jedem Schritt diagonal nach links oder rechts gezogen wird.



Max Triangle Sum Path Problem

Maximum Triangular Sum Path [IOI 1994]

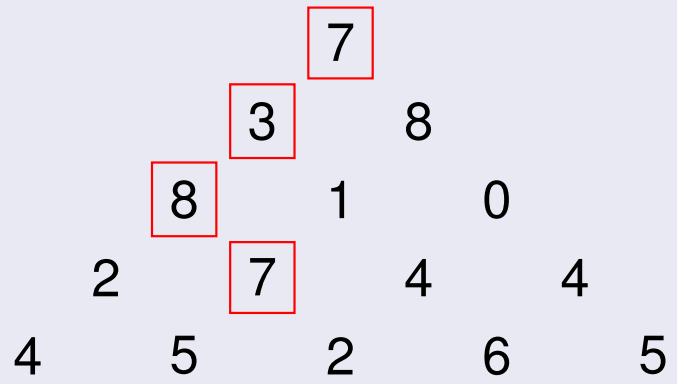
Gegeben ist eine Zahlenpyramide der Höhe n. Gesucht ist ein maximaler Pfad von der Spitze zur Basis, wobei in jedem Schritt diagonal nach links oder rechts gezogen wird.



Max Triangle Sum Path Problem

Maximum Triangular Sum Path [IOI 1994]

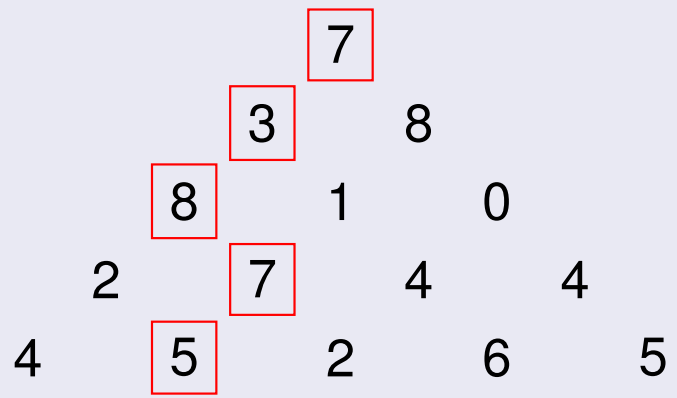
Gegeben ist eine Zahlenpyramide der Höhe n. Gesucht ist ein maximaler Pfad von der Spitze zur Basis, wobei in jedem Schritt diagonal nach links oder rechts gezogen wird.



Max Triangle Sum Path Problem

Maximum Triangular Sum Path [IOI 1994]

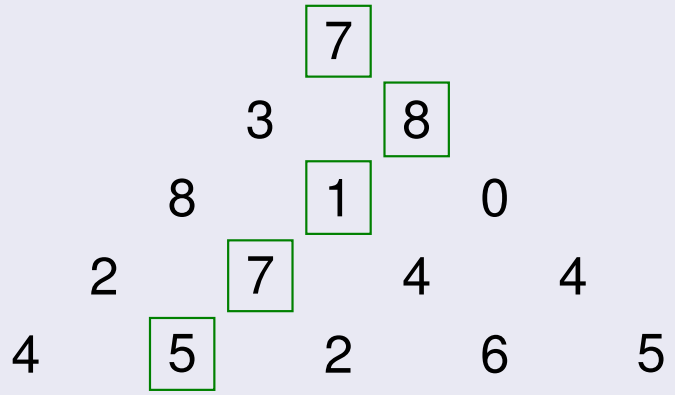
Gegeben ist eine Zahlenpyramide der Höhe n. Gesucht ist ein maximaler Pfad von der Spitze zur Basis, wobei in jedem Schritt diagonal nach links oder rechts gezogen wird.



Max Triangle Sum Path Problem

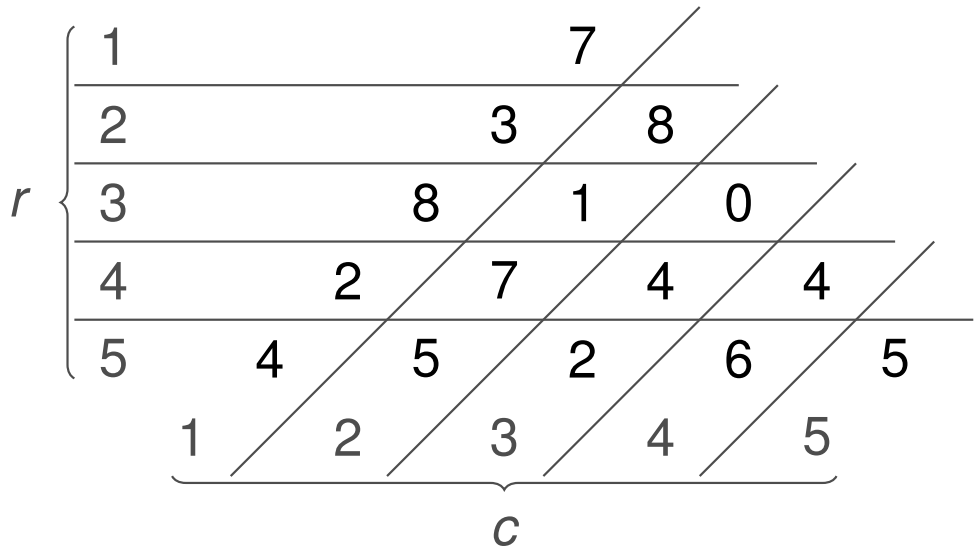
Maximum Triangular Sum Path [IOI 1994]

Gegeben ist eine Zahlenpyramide der Höhe n. Gesucht ist ein maximaler Pfad von der Spitze zur Basis, wobei in jedem Schritt diagonal nach links oder rechts gezogen wird.



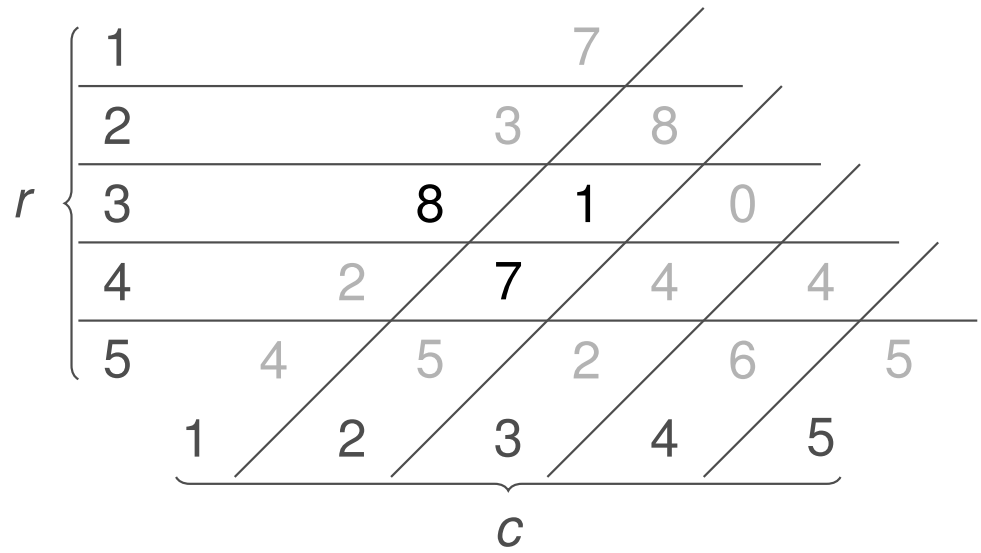
Greedy nicht optimal.

Max Triangle Sum Path Problem - Lösung



$tri(r,c)$ beschreibt den Wert in Zeile r und Spalte c.
Bsp: $tri(1,1) = 7$, $tri(4,3) = 4$.

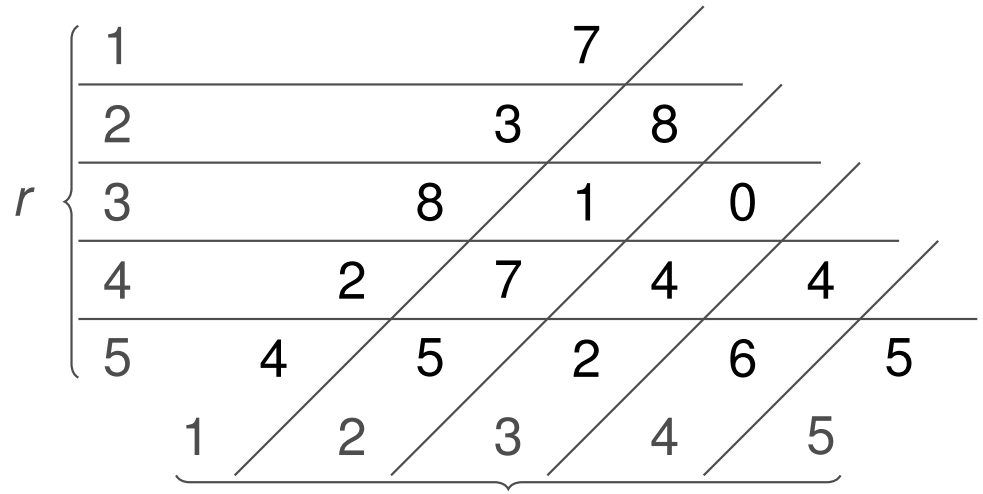
Max Triangle Sum Path Problem - Lösung



1. Die Struktur einer optimalen Lösung beschreiben.

Der optimale Weg bis (r, c) geht durch $(r-1, c-1)$ oder $(r-1, c)$.

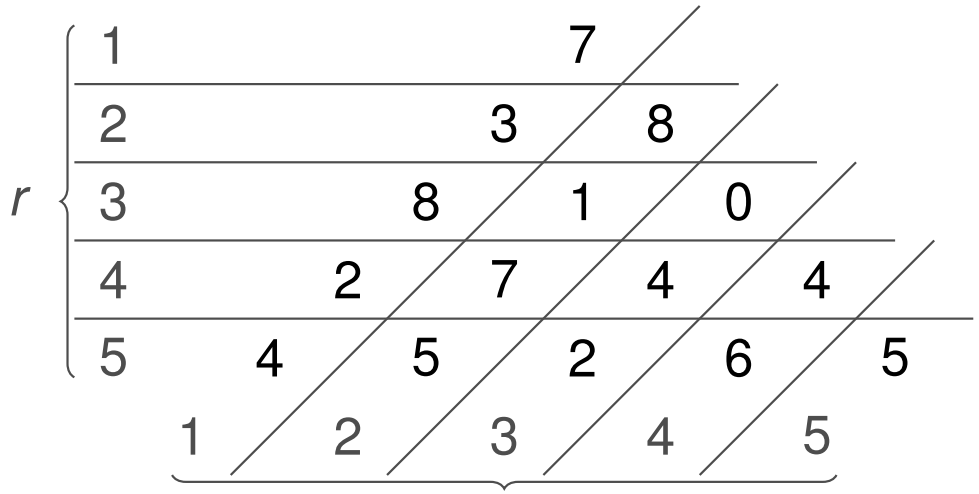
Max Triangle Sum Path Problem - Lösung



2. Rekursive Formulierung einer optimalen Lösung.

$$opt(r,c) = \text{Gewicht des optimalen Weges bis } (r,c)$$
$$= tri(r,c) + \max(opt(r-1,c-1), opt(r-1,c))$$
$$opt(1,1) = tri(1,1)$$
$$opt(r,1) = tri(r,1) + opt(r-1,1)$$
$$opt(r,r) = tri(r,r) + opt(r-1,r-1)$$

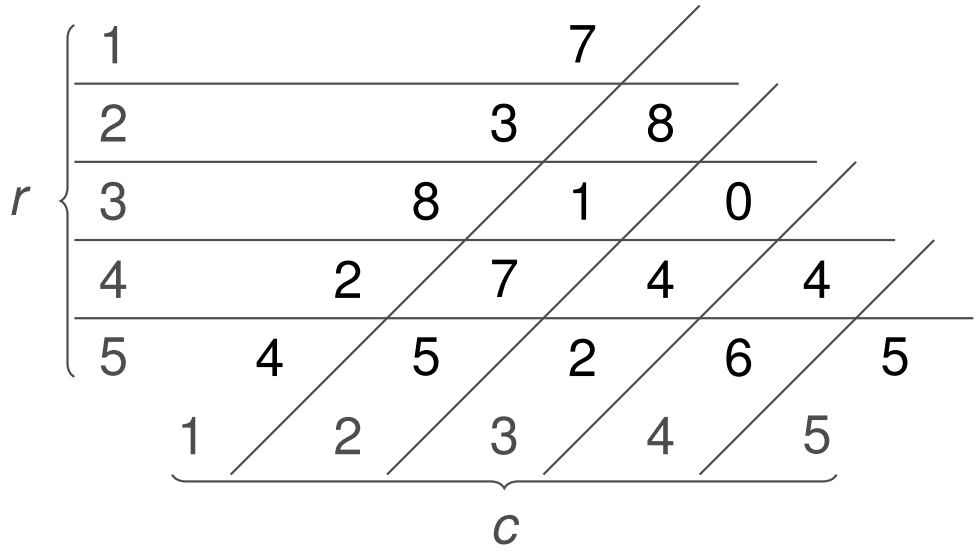
Max Triangle Sum Path Problem - Lösung



3. Berechnen des Wertes einer optimalen Lösung.

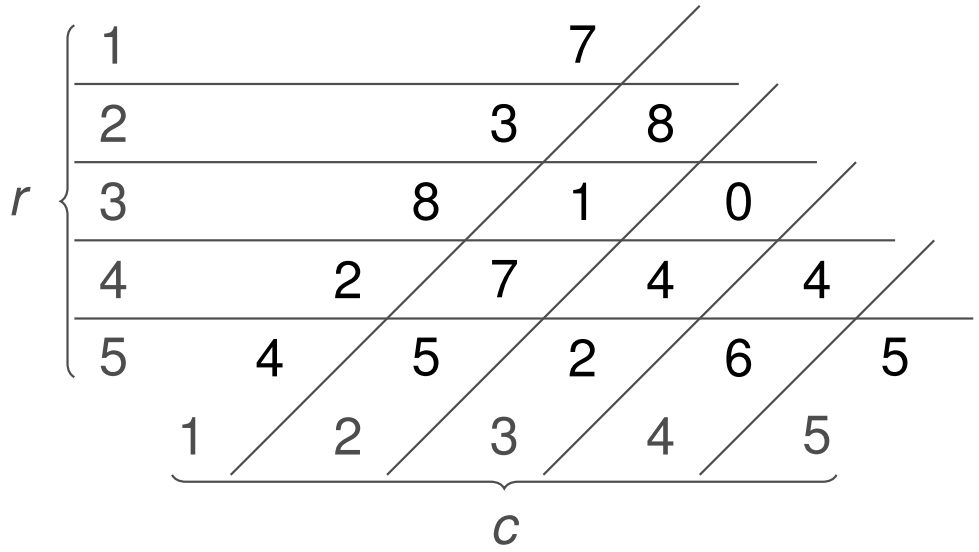
```
opt[1][1] = tri[1][1];
for (int r=2; r<=n; r++) for (int c=1; c<=r; c++)
 if (c==1) opt[r][c] = tri[r][c] + opt[r-1][c];
 else if (c==r) opt[r][c] = tri[r][c] + opt[r-1][c-1];
 else opt[r][c] = tri[r][c] + Math.max(opt[r-1][c-1], opt[r-1][c]);
for (int c=1;c<=n;c++) if (opt[n][c] > longest) longest = opt[n][c];
```

Max Triangle Sum Path Problem - Lösung



4. Rekonstruieren der optimalen Lösung.
   - Wir haben den Wert eines längsten Pfades aber nicht den Pfad selbst.
   - Speichere in jedem Schritt ob von links oder rechts gekommen und rekonstruiere Pfad rückwärts.

Max Triangle Sum Path Problem - Lösung



Komplexität

Speicher- und Laufzeitkomplexität linear in der Eingabegrösse: $\Theta(n^2)$.

Zusammenfassung

Dynamische Programmierung ist ...

- algorithmische Technik zum Lösen etlicher Probleme.
- der Fokus von vielen Olympiadenaufgaben.

Dynamische Programmierung funktioniert durch ...

- Aufteilen des Problems in Teilprobleme des gleichen Typs.
- Lösen aller möglichen Teilprobleme.
- Zusammensetzen der optimalen Lösung durch optimale Lösungen zu den Teilproblemen.