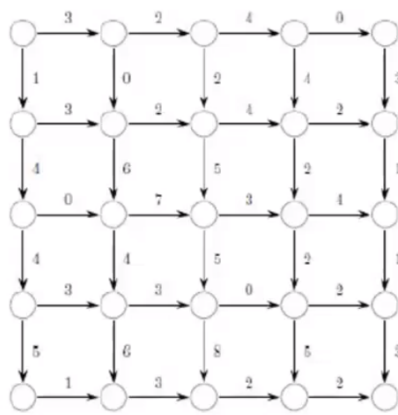


Eigenschaften

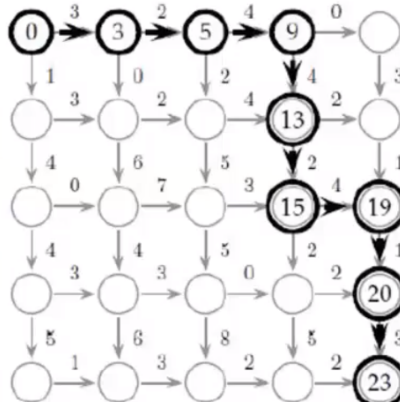
- Gesamtlösung baut auf Teillösungen eines Problems auf
- base cases
 - non-overlapping subproblems
- Teilprobleme müssen nicht voneinander unabhängig sein
 - unlike [[Divide & Conquer]] (only base cases)
- Memorization
 - Teilergebnisse werden wiederverwendet
 - use subproblem's solution if available otherwise compute recursively

Manhattan Tourist Problem

- "Schwerster" Pfad in einem Grid (Manhattan Tourist Problem)

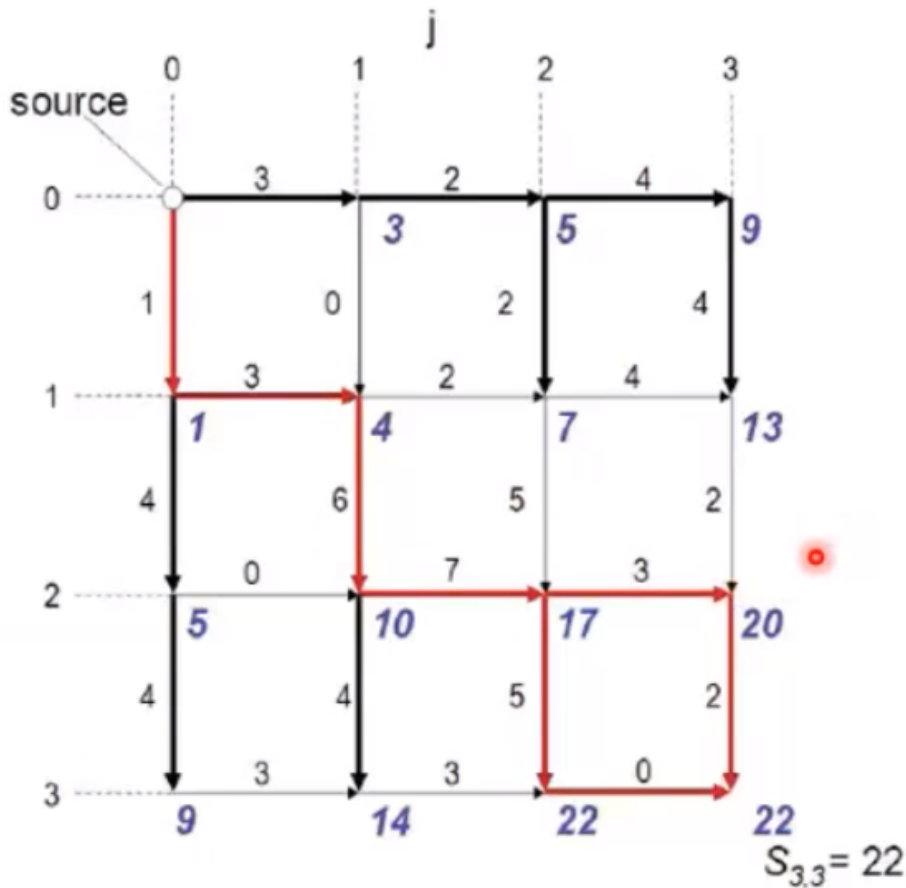


Zahlen=# der Sehenswürdigkeiten



nur Bewegungen nach Osten oder Süden sind erlaubt.





Relate Computation

- time complexity to solve SP
- if solutions to other SPs already known
- overall runtime = sum of relate computation of all SPs

Fibonacci subproblems: $F(0), F(1), \dots, F(n)$

Relate computation = adding two numbers

$$F(i) = F(i-1) + F(i-2)$$

Fibonacci numbers are large, use up to $O(n)$ bits:

$$O(n/w)$$

Runtime:

- #subproblems = n

$$\Rightarrow \text{runtime } O\left(\#subproblems \cdot \frac{n}{w}\right) = O\left(\frac{n^2}{w}\right)$$

assumption: wordsize w addition in $O(1)$ time

Bowling

Input:

sequence of n **bowling pins**, numbered with integers $\in [-K, K]$

You can either hit **single pins**, or **two adjacent pins**.

Throw as many balls as you want (you don't have to hit all pins)

Goal: Maximize your points

- hit a single pin with number $x_i \rightarrow x_i$ points
- hit two adjacent pins with numbers x and $y \rightarrow x \cdot y$ points

- Subproblems:** $S[i]$: Optimal number of points that we can get with the first i pins (prefix problem)
- Original problem:** $S[n]$
- Base case:** $S[0] = 0$ points (empty prefix)

Relation: if $i \geq 2$: $S[i] = \max\{S[i-1], S[i-1] + x_i, S[i-2] + x_{i-1} \cdot x_i\}$
if $i = 1$: $S[i] = \max\{S[i-1], S[i-1] + x_i\}$

- Topological order:** $S[0], S[1], S[2], \dots, S[n]$

```
S[0]=0, S[1]=max{0, x1} //base case
FOR i=2 to n
    S[i]=max{S[i-1], S[i-1] + x_i, S[i-2] + x_{i-1} · x_i}
Return S[n]
```

“for

Runtime: $\sum_S \text{relate computation of } S = O(n) \cdot O(1) = O(n)$

subproblem

Rucksack problem

Knapsack

Input:

n items: i -th item has weight w_i and value v_i

W : the weight capacity of the knapsack.

- Question:** What's the maximum value of items you can pack such that their weights' sum does not exceed W ?

Subproblems: $S[i, x]$, $0 \leq i \leq n$, $0 \leq x \leq W$:

best value we can get with first i items with “using” capacity x

Original problem: $S[n, W]$ (all items, full capacity)

Base case: $S[0, x] = 0$ for $0 \leq x \leq W$; $S[i, 0] = 0$ for $0 \leq i \leq n$

Relate: $S[i, x] = \text{if } w_i \leq x: \max\{S[i-1, x], S[i-1, x - w_i] + v_i\} \text{ else: } S[i-1, x]$

- Topological order:** For $i=0$ to n : For $x=0$ to W : $S[i, x]$

For i from 0 to n :

For x from 0 to W :

If $i == 0$ or $x == 0$: $S(i, x) = 0$

else if: $w_{i-1} \leq x$: $S(i, x) = \max(v_{i-1} + S(i-1, x - w_{i-1}), S(i-1, x))$

else: $S(i, x) = S(i-1, x)$

- Return $S(n, W)$

$w_1 = 2, v_1 = 3$

$w_2 = 3, v_2 = 4$

$w_3 = 4, v_3 = 5$

$w_4 = 5, v_4 = 6$

$w_5 = 6, v_5 = 7$

	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	3	3	3	3	3	3	3	3	3
2	0	0	3	4	4	7	7	7	7	7	7
3	0	0	3	4	5	7	8	9	9	12	12
4	0	0	3	4	5	7	8	9	10	11	13
5	0	0	3	4	5	7	8	9	10	11	13

Runtime:

Relate computation: $O(1)$

#subproblems: $O(n \cdot W)$

Is this polynomial time?

No. W can be exponential in the input.

Knapsack is **NP-complete**.