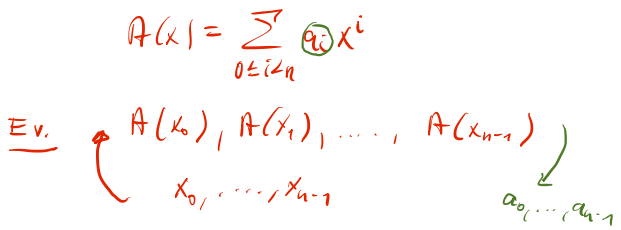
DYNAMIC PROGRAMMING, PART 1: INTRODUCTION AND EXAMPLES

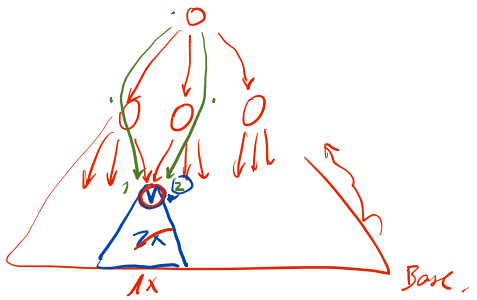
Marcus Ritt

CMP 601 – Algorithms and Theory of Computation —

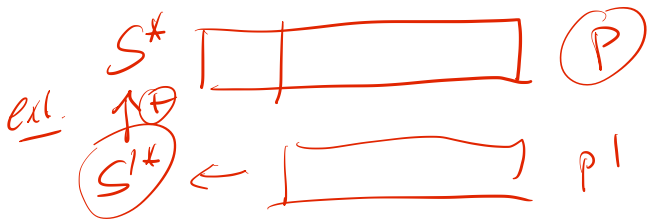
<2020-04-14 ter>

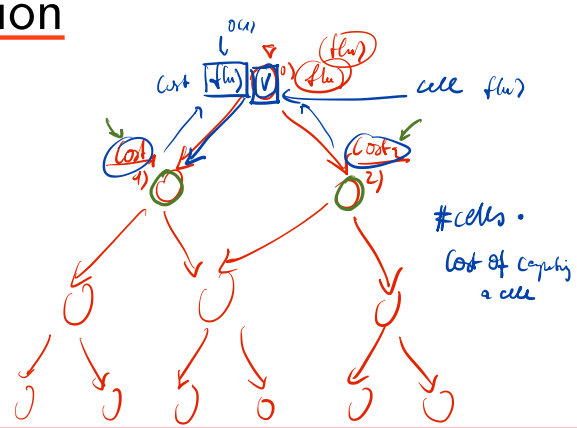
Outline





- A recursive definition
  - So: problem is solved via subproblems
- A large overlap, such that *caching* makes sense
- Complexity then: size of the cache times time to fill each cell
- Optimal substructure:
  - For decision problems: subproblems cover all cases
  - For optimization problems: one of the subproblems can be extended to the optimal solution





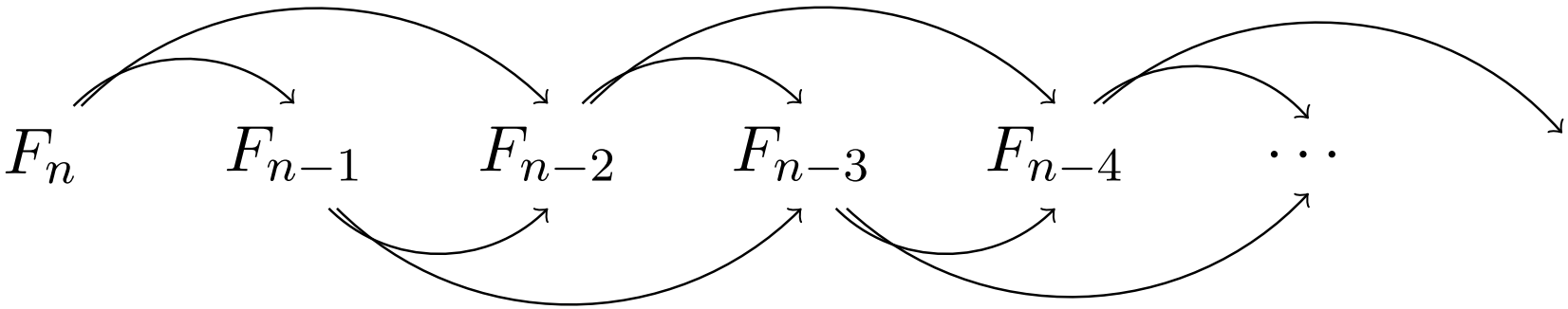
Definition

Correctness

$$F_n = \begin{cases} F_{n-1} + F_{n-2}, & \text{caso } n \geq 2, \\ 1, & \text{caso } n \in \{0, 1\}. \end{cases}$$

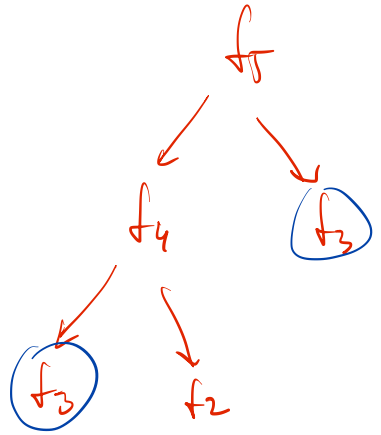
F_n	1	1	2	3	5	8	13
n	0	1	2	3	4	5	6

Dependency graph



Simple recursive algorithm


```
fib(n) :=
   if n ≤ 1 then
      return 1
   else
      return fib(n − 1) + fib(n − 2)
   end if
end
```



$$\begin{cases} f_0 := 1 \\ f_1 := 1 \\ f_i := \bot \end{cases} \texttt{para}\ i \geq 2$$

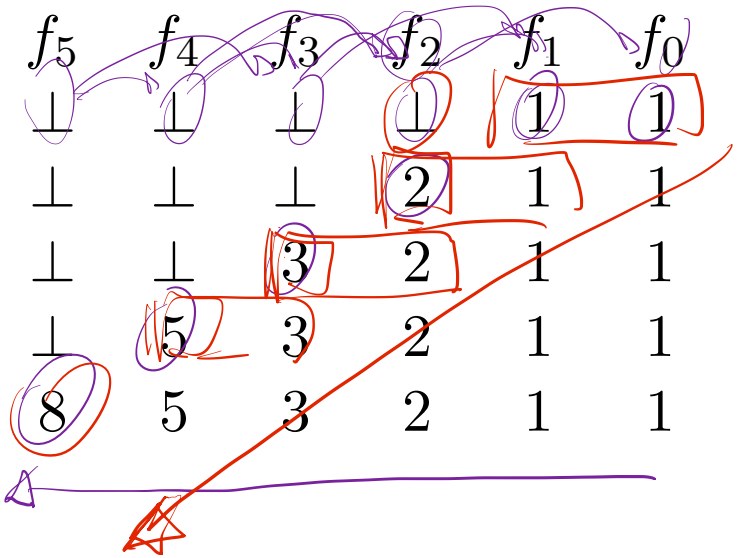
Memoization

```
fib(n) :=
    if f_n ≠ ⊥ then
        f_n := fib(n − 1) + fib(n − 2)
    end if
    return f_n
end
```

cost per call · # calls = n
⇒ O(n).
time / space

Cache evolution

	f_5	f_4	f_3	f_2	f_1	f_0
Initial	$\perp$	$\perp$	$\perp$	$\perp$	1	1
1	$\perp$	$\perp$	$\perp$	2	1	1
2	$\perp$	$\perp$	3	2	1	1
3	$\perp$	5	3	2	1	1
4	8	5	3	2	1	1



Sequential implementation

```
fib(n) :=
    f_0 := 1
    f_1 := 1
    for i ∈ [2, n] do
        f_i := f_{i-1} + f_{i-2}
    end for
    return f_n
end
```

Space: $O(1)$
Time: $O(n)$

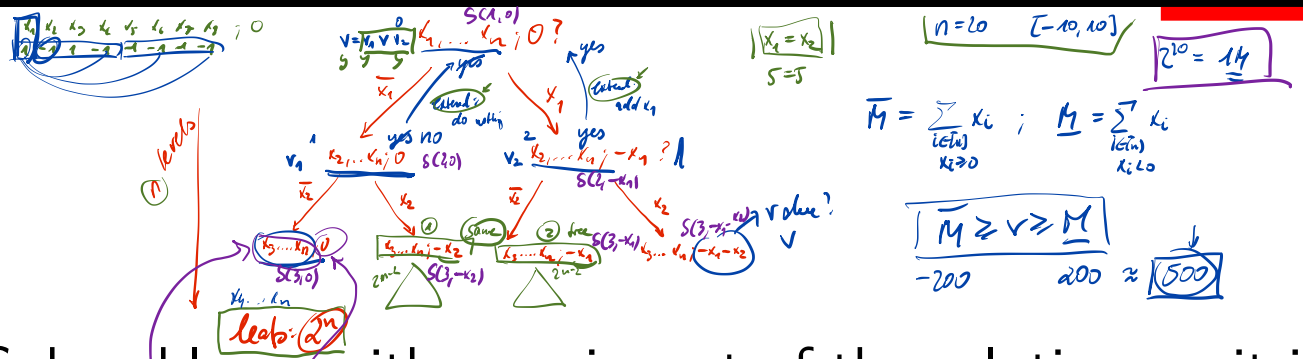
```
fib(n) :=
  f := 1
  g := 1
  for i ∈ [2, n] do
    { invariante: f = F_{i-2} ∧ g = F_{i-1} }
    g := f + g
    f := g − f
    { invariante: f = F_{i-1} ∧ g = F_i }
  end for
```

$X = \sum_{i \in [n]} x_i / 2$

$x_i \in \mathbb{Z}$

- Given numbers $x_1, x_2, \ldots x_n$ is there a non-empty subset whose sum is 0?
- Problem is NP-hard.
- Brute-force algorithm: test all subsets in $O(2^n n)$.

Dynamic programming

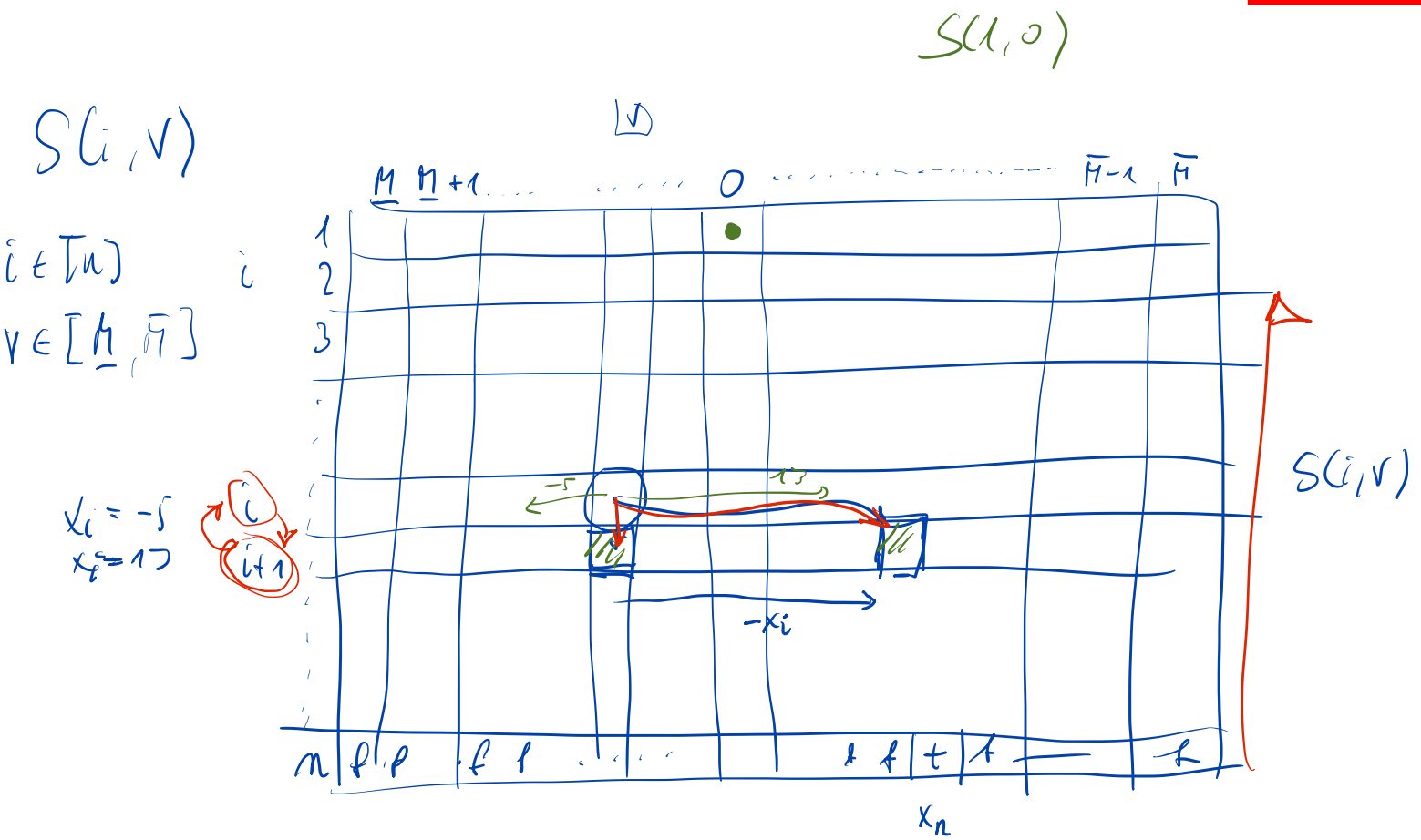


- Subproblems: either x_1 is part of the solution, or it is not
- If it isn't: find a subset of $x_2, \ldots, x_n$ that sums to 0
- If it is: find a subset of $x_2, \ldots, x_n$ that sums to $-x_1$.
- We can see: a subproblem consists of
  - a suffix of the elements $x_i, x_{i+1}, \ldots, x_n$
  - and a target value v

- So we define: $S(i, v)$: there is a non-empty subset of $x_i, x_{i+1}, \ldots, x_n$ summing to v
- And we want to know: $S(1, 0)$.
- Recursive definition:

$$S(i,v) = \begin{cases} S(i+1,v) \vee S(i+1, v-x_i), & 1 \le i < n, \\ x_n = v, & i = n. \end{cases}$$

- Now we can apply all the same steps as for the Fibonacci example
- Complexity?


S(1,0)
S(i,v)
i ∈ [n]
v ∈ [M, M̄]
v
M
M+1
0
M̄-1
M̄
1
2
3
S(i,v)
-5
13
-xi
xi = -5
xi = 13
i
i+1
xn

(Empty)

(add init) $S(n, r) = [x_n = v]$, $\forall v$

$S(i, r) = \bot$ $\forall 1 \le i < n$, $\forall v$

for $i = n-1, n-2, \ldots, 1$

  for $v \in [\underline{M}, \overline{M}]$ | $\underline{M}, \underline{M}+1, \ldots, \overline{M}-1, \overline{M}$ / $\overline{M}, \overline{M}-1, \ldots, \underline{M}$

    $S(i, r) = S(i+1, v) \vee S(i+1, r - x_i)$

return $S(1, 0)$

- Given n items with values $v_1, \ldots, v_n \geq 0$ and weights $w_1, \ldots, w_n \geq 0$ and a maximum weight W
- Find the subset $S \subseteq [n]$ of the largest value $v(S) = \sum_{i \in S} v_i$ that fits into the Knapsack, i.e. $w(S) = \sum_{i \in S} w_i \leq W$

- Same dichotomy as for subset sum.
- If item 1 is not part of the optimal solution: find the optimal solution for $2, \ldots, n$ with maximum weight W
- If item 1 is part of the optimal solution: find the optimal solution for $2, \ldots, n$ with maximum weight $W - w_1$ and add v_1
- Thus: a subproblem is defined by
  - a suffix of the item $i, i+1, \ldots, n$
  - and a maximum weight w

- So we define: $K(i, w)$: the optimal value using item $i, \ldots, n$ with maximum weight w.
- And we want to know: $K(1, W)$.
- Recursive definition:

$$K(i,w) = \begin{cases} \max\{K(i+1,w), K(i+1,w-w_i)+v_i\}, & 1 \leq i \leq n, w_i \leq \\ K(i+1,w), & 1 \leq i \leq n, w_i > \\ 0, & i > n. \end{cases}$$

- Now we can apply all the same steps as for the Fibonacci example, again
- Complexity?

(Empty)

(Empty)