DTS203TC
Design and Analysis of Algorithms

Lecture 12: Dynamic programming

Dr. Qi Chen and Pascal Lefevre
School of AI and Advanced Computing

Xi'an Jiaotong-Liverpool University
西交利物浦大学

Learning outcomes

- Dynamic programming
  - Basics
  - Problem solving: Climbing Stairs
- Greedy algorithms
  - Basics
  - Problem solving: Jump game

Dynamic programming

What is Dynamic Programming?

It's a computer programming technique *(From R. Bellman, 1950)*

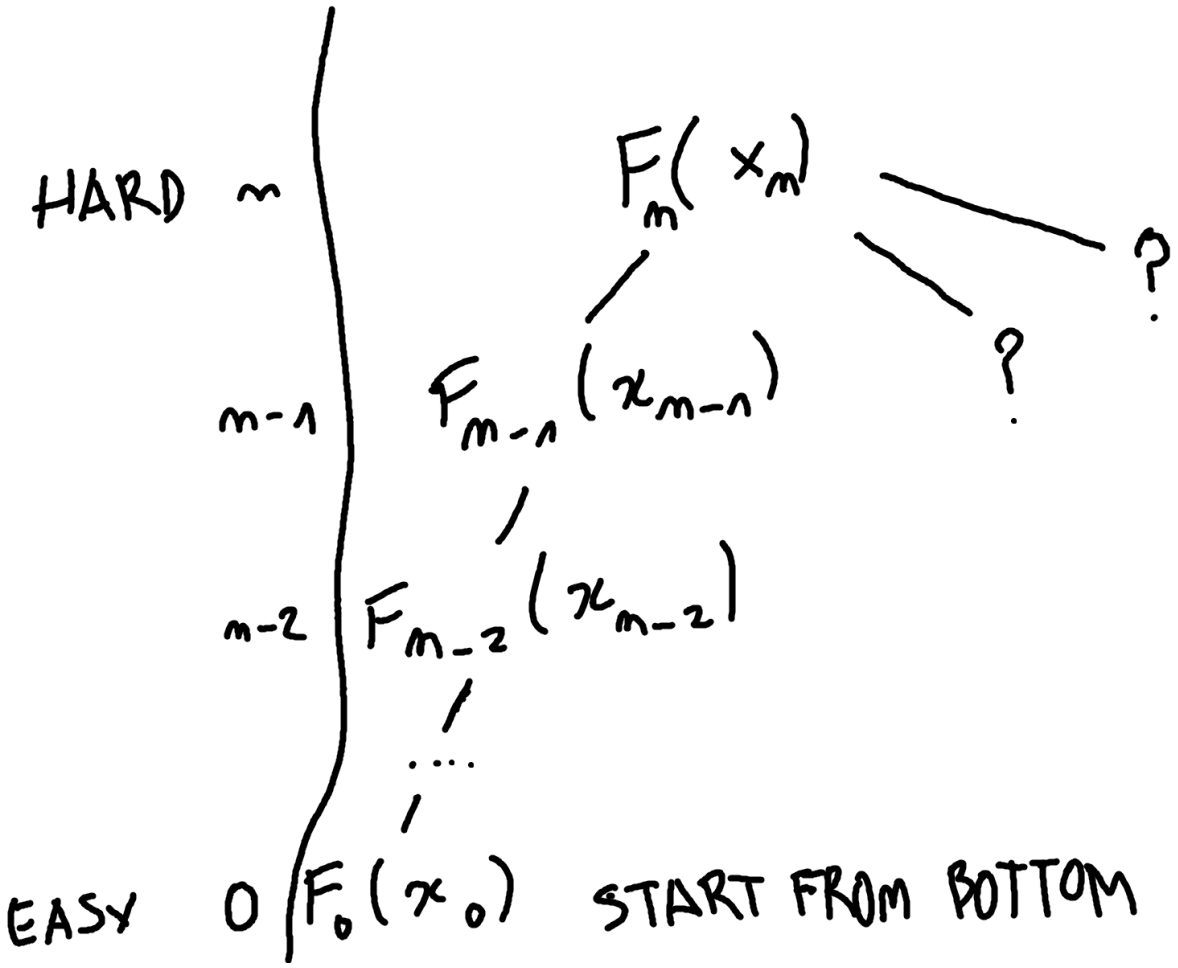
1. An algorithmic problem is divided into sub-problems
   - Bottom-up approach

2. The sub-problems are optimized to find the solution

3. The results of sub-problems are stored inside a table (results are reused to save memory and time)

What is Dynamic Programming?

1. An algorithmic problem is divided into sub-problems
   - Bottom-up approach
2. The sub-problems are optimized to find the solution
3. The results of sub-problems are stored inside a table (results are reused to save memory and time)



Climbing stairs

- Problem statement:
  - You are climbing a staircase.
  - It takes n steps to reach the top.
  - Each time you can either climb 1 or 2 steps.

- Goal: in how many ways can you climb to the top?

Climbing stairs

- Example:
  - Input N = 2 stairs to climb
  - Solution: 2

- Explanation (brute force, we list all the possible solutions)
  - 1 + 1
  - 2

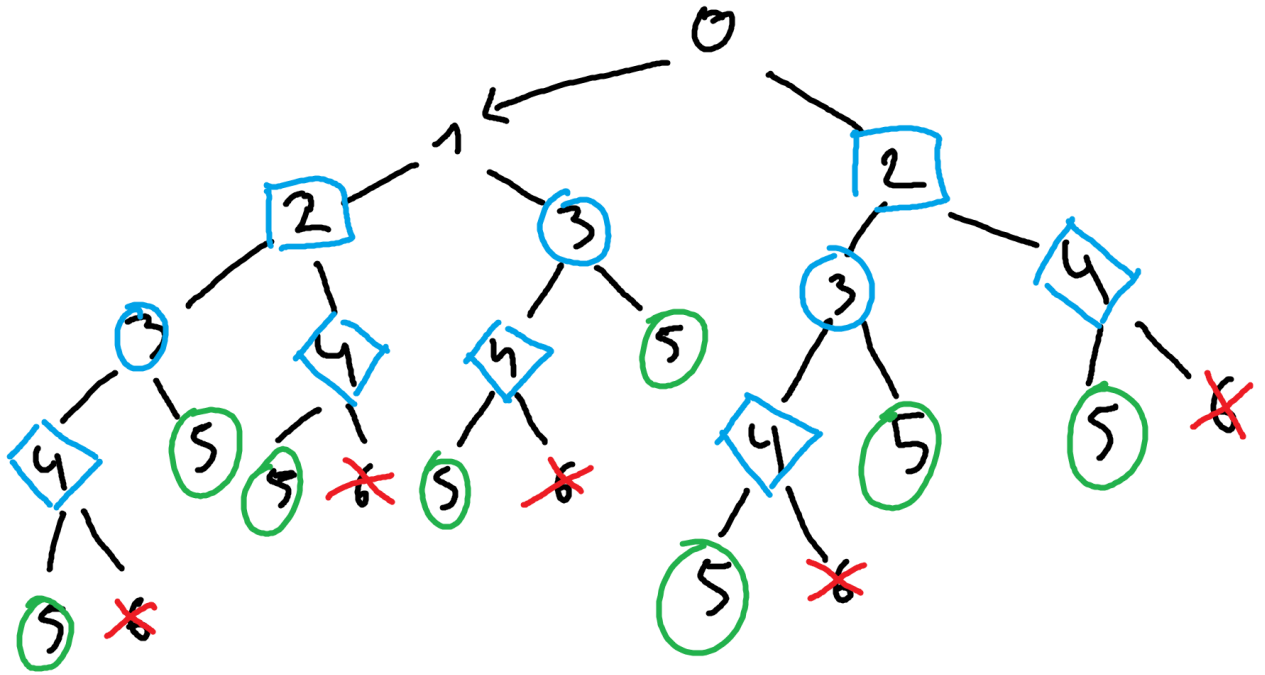
Climbing stairs

- Example:
  - Input N = 3 stairs to climb
  - Solution: 3

- Explanation (brute force)
  - 1 + 1 + 1
  - 1 + 2
  - 2 + 1

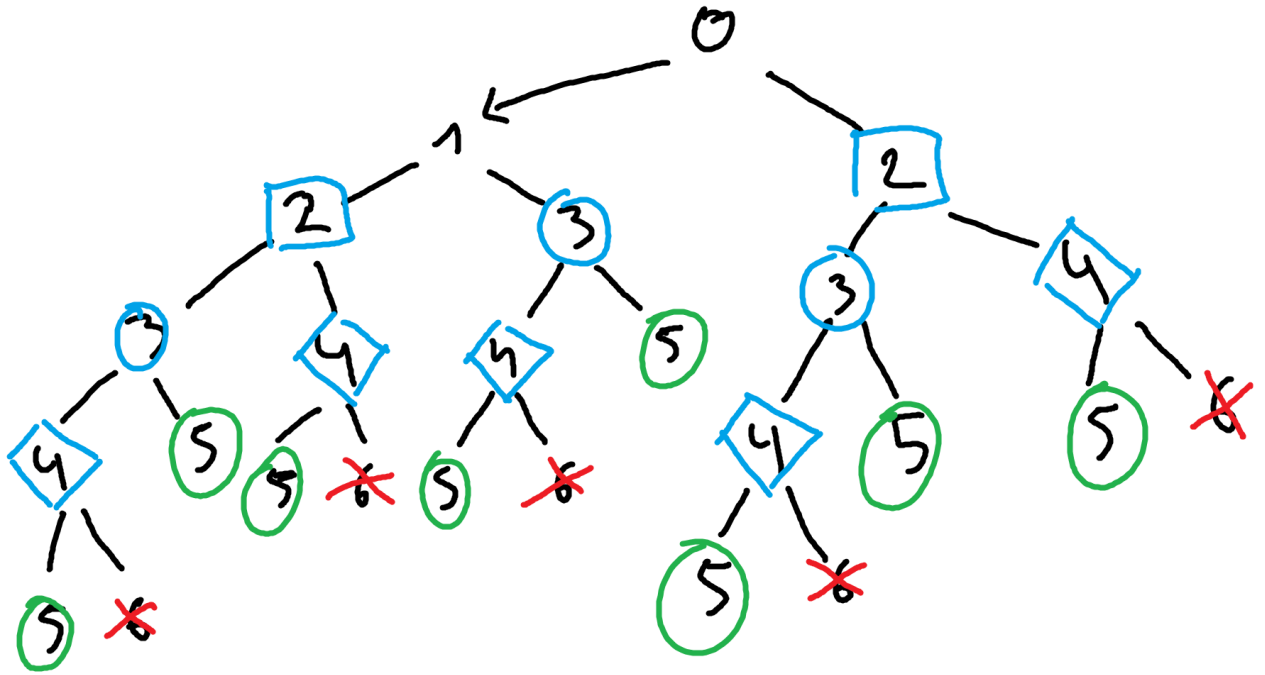
Climbing stairs

- Example with decision tree:
  - Input N = 5 stairs to climb



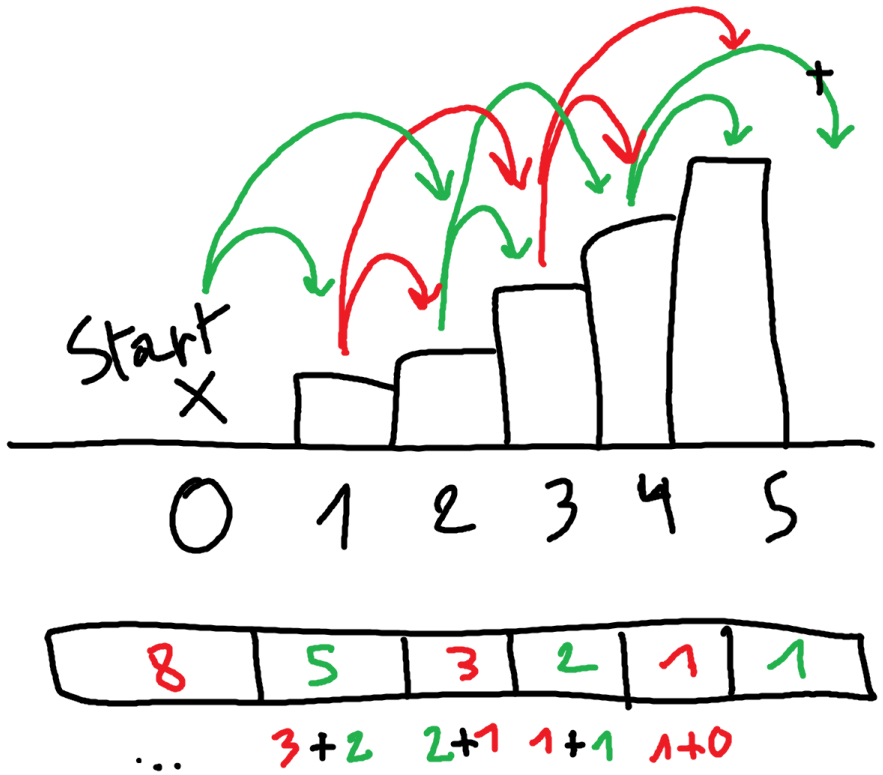
Climbing stairs

- Example with decision tree: lecture exercise
  - How about N = 6? Easy or hard to solve?



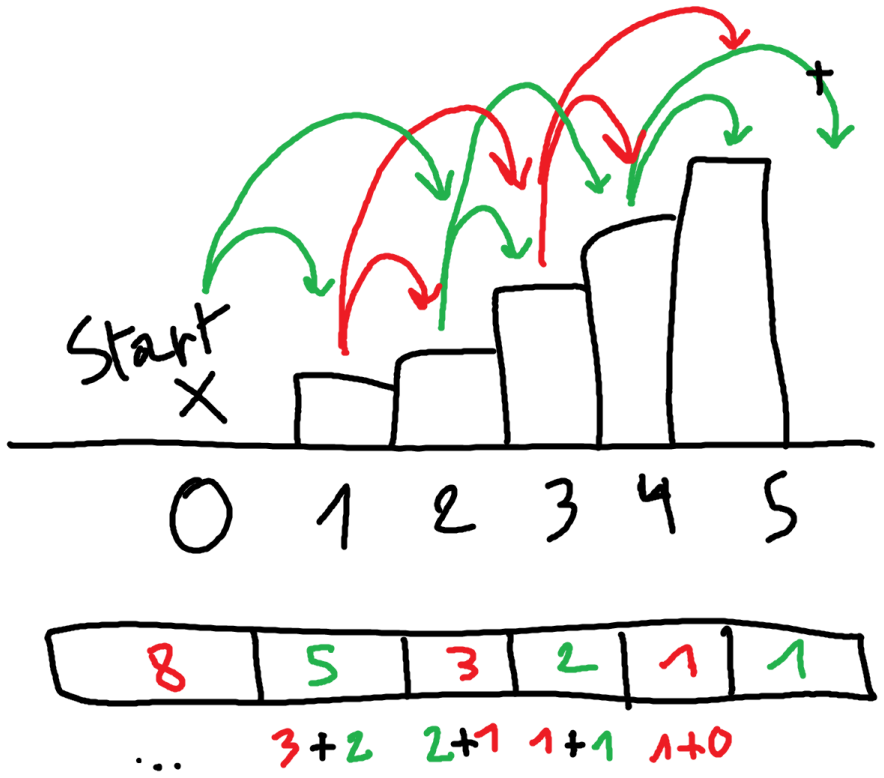
Climbing stairs

- Dynamic Programming solution



Climbing stairs

- Dynamic Programming solution: we find the Fibonacci sequence!



Climbing stairs

- Implementation

```
def climb_stairs(N):
    one, two = 1, 1

    for i in range(N - 1):
        temp = one         # store the variable one to a temporary variable
        before we update the variable one
        one = one + two    # add the 2 previous values to get the new result
        two = temp         # update the variable two with the temporary variable

    return one  # return the content of the variable one
```

Climbing stairs

- More information about the Fibonacci sequence
  - https://mathisfun.com/numbers/Fibonacci-sequence.html

Greedy algorithms

What is a greedy algorithm?

- Concept
  - Considering 1 step (local): optimize for the **best** choice, and **repeat**
  - Example: inside a loop, repeat the same step

Open questions:

- Definition of “Best”: open for interpretation…
  - Are there any specific conditions?
  - Algorithm design?
- Which algorithm is the “best”?
  - Make experiments, explore, compare
  - …

What is greedy algorithm?

- Concept
  - Considering 1 step (local): optimize for the **best** choice, and **repeat**
  - Example: inside a loop, repeat the same step
- Pros
  - Simple algorithm, step by step
  - Fast, in general
  - The solution is usually acceptable

What is greedy algorithm?

- Concept
  - Considering 1 step (local): optimize for the **best** choice, and **repeat**
  - Example: inside a loop, repeat the same step
- Pros
  - Simple algorithm, step by step
  - Fast
  - The solution is usually acceptable

What is greedy algorithm?

- Concept
  - Considering 1 step (local): optimize for the **best** choice, and **repeat**
  - Example: inside a loop, repeat the same step
- Pros
  - Simple algorithm, step by step
  - Fast
  - The solution is usually acceptable
- Cons
  - Sometimes, the solution is not good enough

Jump game

- Problem statement:
  - array a, a[i] positive integers of length n
  - First index: 0
  - a[i] is the maximum jump length at index I

- Goal: can we reach index n?

Jump game

- Problem statement:
  - array A, A[i] positive integers of length n
  - First index: 0
  - A[i] is the maximum jump length at index I

- Goal: can we reach index n?
  - Yes or No: True or False question

- Input examples:
- A = [1,1,1,1,1]

Jump game

- Input examples:
- A = [1,1,1,1,1]
- A = [0, 12, 23, 3454]
- A = [5,0,0,0,0]
- A = [2, 0, 0, 4]

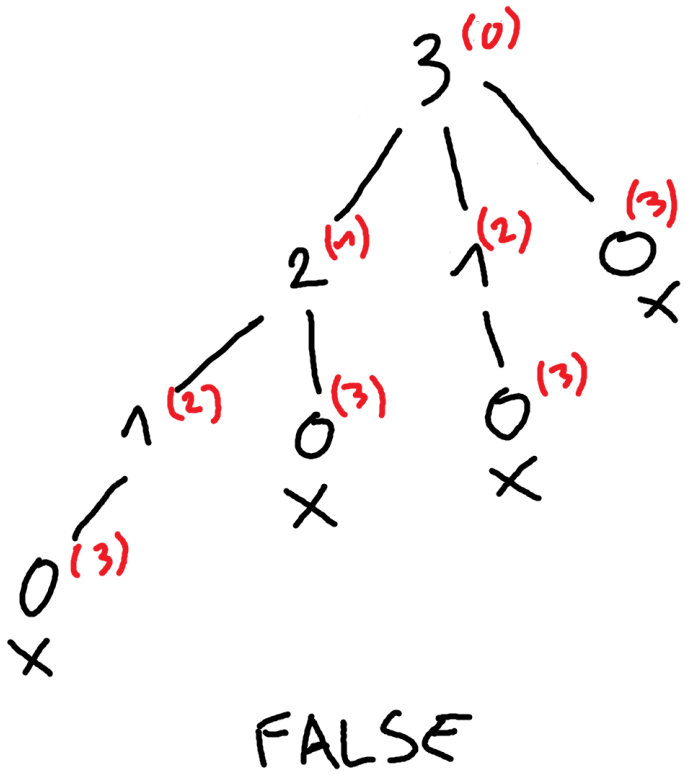
Jump game

- Input examples:
- A = [1,1,1,1,1]: True
- A = [0, 12, 23, 3454]: True?
- A = [5,0,0,0,0]: True
- A = [2, 0, 0, 4]: False

- Can you see the patterns?

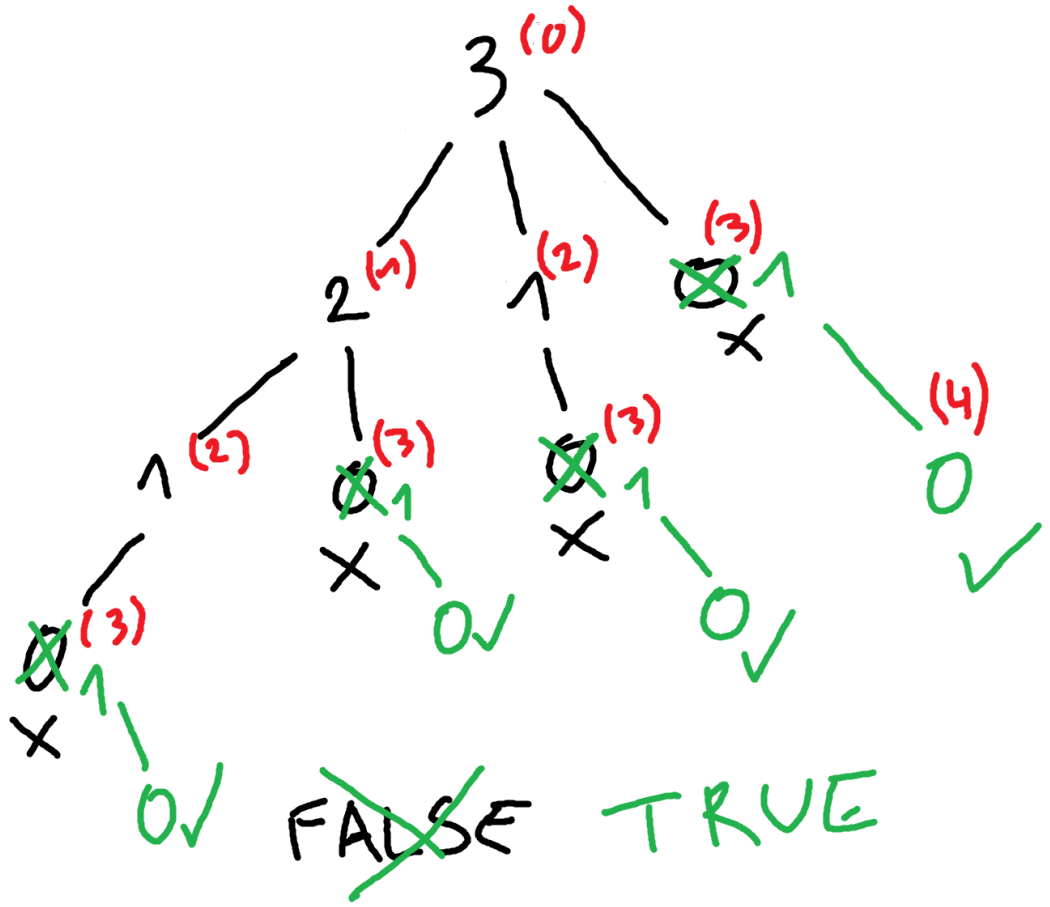
Jump game

- Advice 1: try some examples: Done
- Advice 2: **Brute force** method
- Input: A = [3,2,1,0,0]



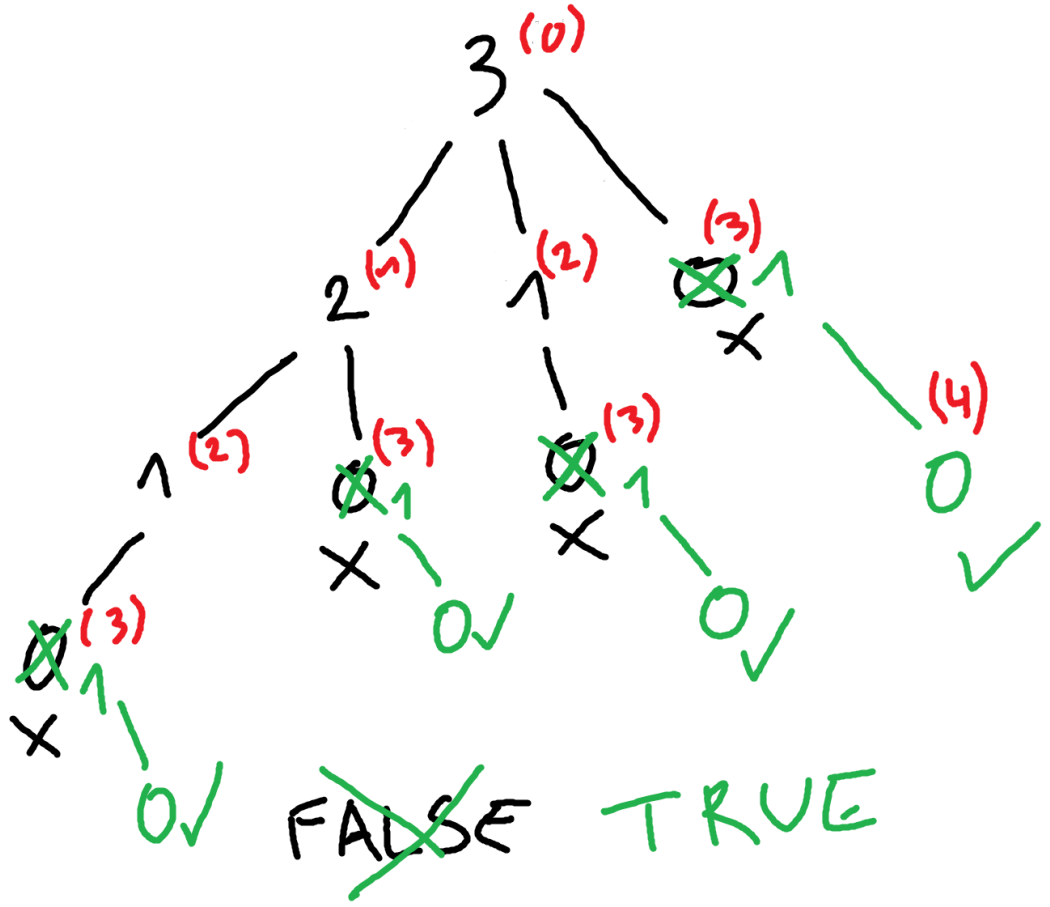
Jump game

- Input: A = [3,2,1,**1**,0]
- Complexity: O(n^n)



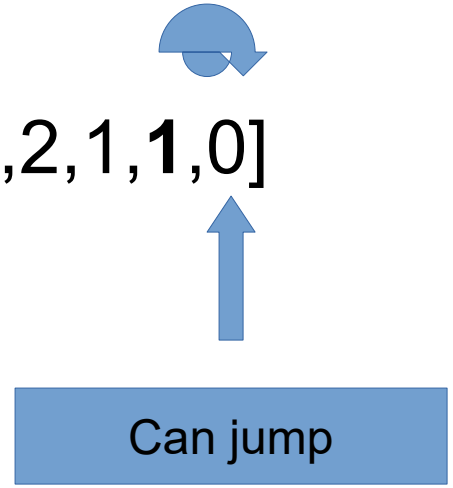
Jump game

- Input: A = [3,2,1,**1**,0]
- Complexity: O(n^n) (very slow)
- Can be O(n^2) with DP:
- DP[index] = True/False
  - DP[0] = ?
  - DP[1] = ?
  - DP[2] = ?
  - DP[3] = ?



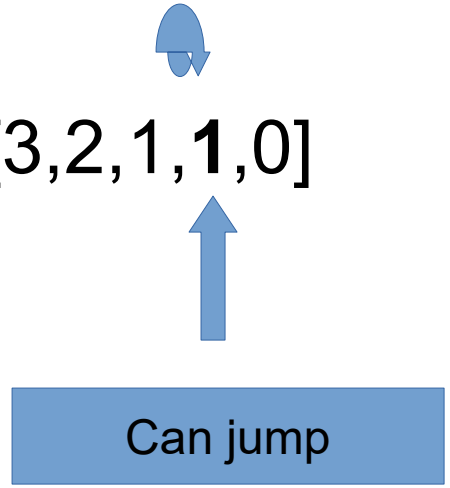
Jump game

- Greedy solution in O(n) (this is fast)
  - Start from the end! (bottom of the tree)



Jump game

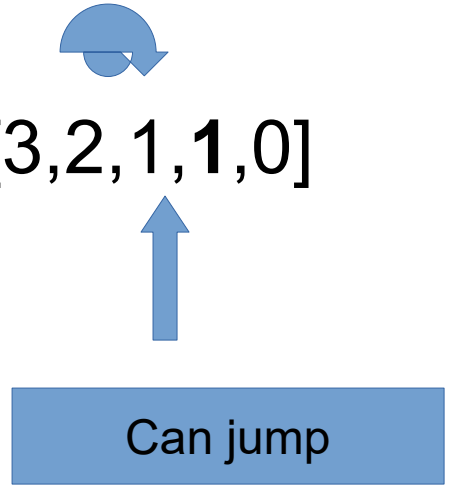
- Greedy solution in O(n)



- Input: A = [3,2,1,**1**,0]

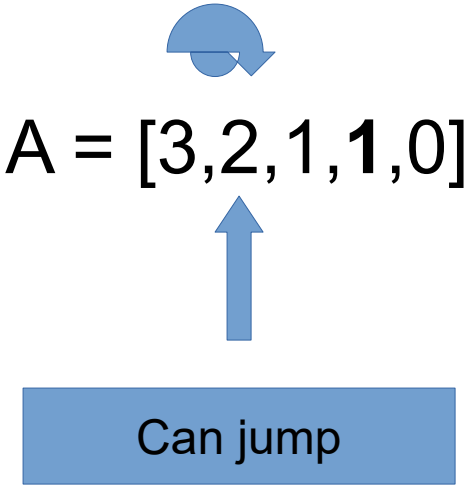
Jump game

- Greedy solution in O(n)



Jump game

- Greedy solution in O(n)



Jump game

- Implementation

```
def can_jump(A):
    goal = len(A) - 1  # the last index to reach
    n = len(A)  # length of array A

    for i in range(n - 1, -1 -1):  # start the greedy algorithm from the end of
    the array (at n - 1)

        if i + A[i] >= goal:  # A[i] is the maximum jump from i
            # if the jump is larger than goal, it means we can reach goal from
            i by adding A[i]
            goal = i

    return goal == 0  # if goal contains index 0, the statement is True and it
    is possible to go from index 0 to index n - 1
```