Efficient Recursion, and Dynamic Programming

Instructor: Shahriar Talebi

Efficient Recursion: Memoization and Dynamic programming

Fibonacci Recursive code

```
Counter = 0
def fib(n):
    global counter
    counter += 1
    if n == 1:
        return 1
    elif n == 2:
        return 1
    else:
        return fib(n-1) + fib(n-2)
```

Fibonacci sequence F_n :

$$F_1 = 1,$$
$$F_2 = 1,$$

for $n > 2$:

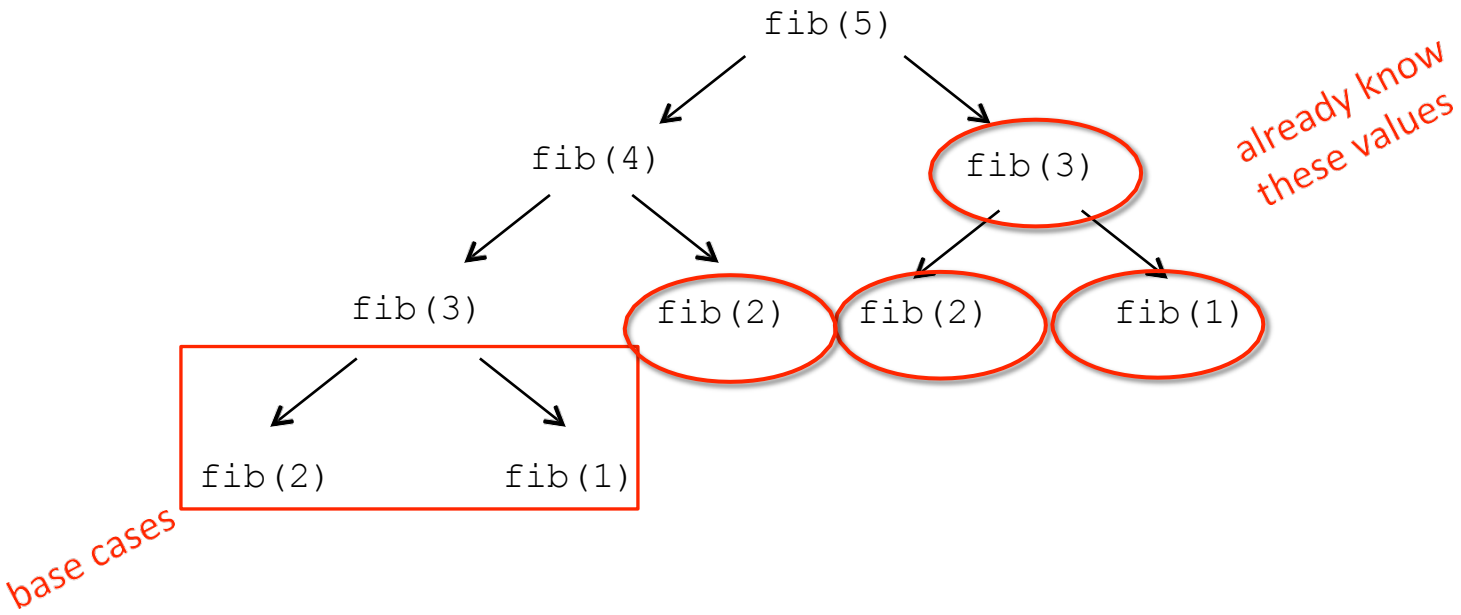
$$F_n = F_{n-1} + F_{n-2}$$

$$1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, \ldots$$

- two base cases
- calls itself "twice" -- this code is very inefficient

Inefficient Fibonacci code

```
fib(n) = fib(n-1) + fib(n-2)
```



- **recalculating** the same values many times! Try `fib(134)` !
- Instead, we could keep **track** of already calculated values!

Fibonacci with a dictionary

```
Counter = 0
def fib_efficient(n, d):
    global counter
    counter += 1
    if n in d:
        return d[n]
    else:
        ans = fib_efficient(n-1, d) + fib_efficient(n-2, d)
        d[n] = ans
        return ans

d = {1:1, 2:1}
print(fib_efficient(134, d))
```

Method sometimes called " memoization"

Initialize dictionary with base cases

- do a **lookup first** in case already calculated the value
- **modify dictionary** as progress through function calls

The efficiency gain

- Calling **`fib(n)`** results in $2F_n - 1 \approx (1.6)^n$ recursive calls to the procedure (why?)
- The first calling of **`fib_efficient(n)`** results in $2n - 3$ recursive calls to the procedure (why?). How about if calling again any smaller n?
- **Memoization** is very efficient
- This only works for procedures without side effects (i.e., the procedure will always produce the same result for a specific argument independent of any other computations between calls)

Dynamic programming (intro)

If we have

- Overlapping Subproblems

and

- *Principle of Optimality* holds

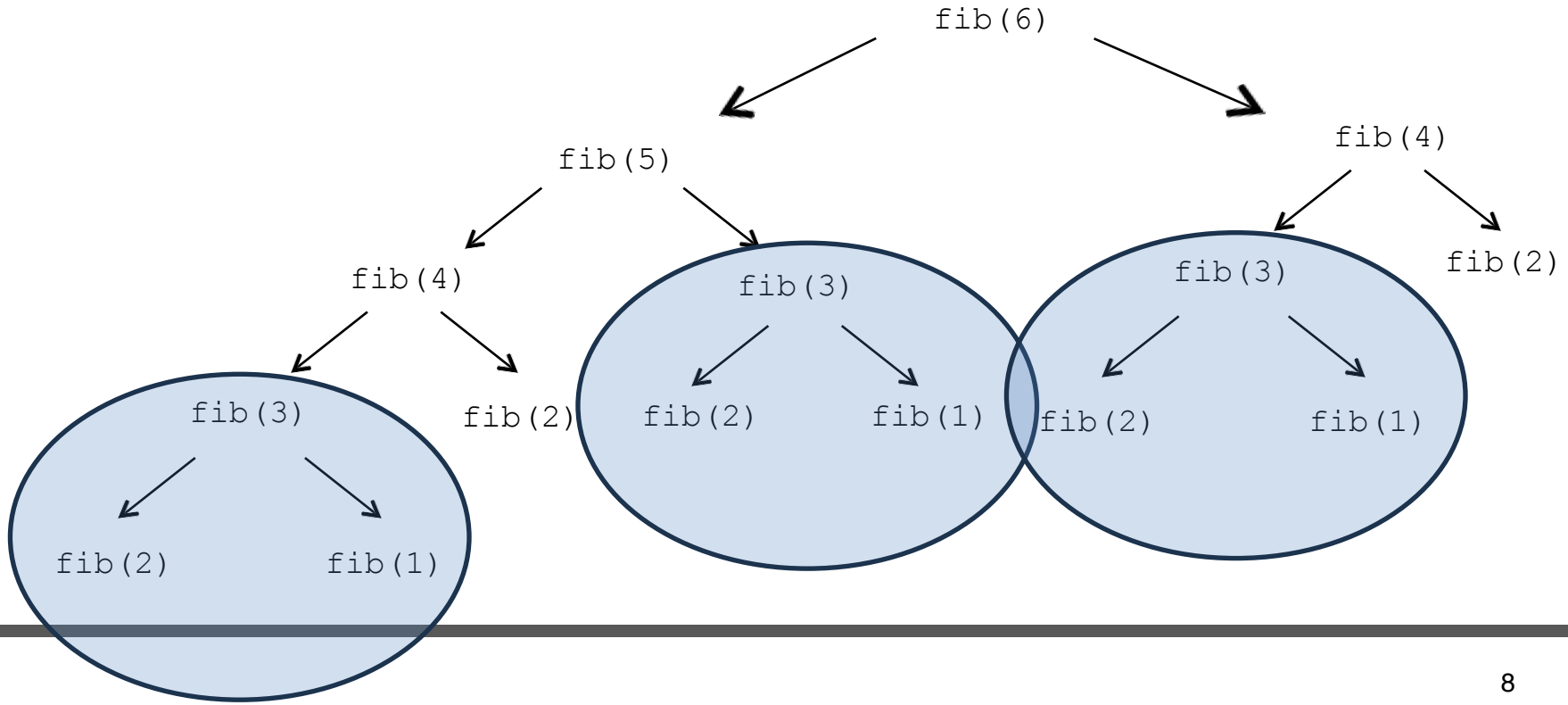
and

- Dependencies of subproblems are acyclic
  (Directed Acyclic Graph)

Then the DP is the optimal efficient solution!

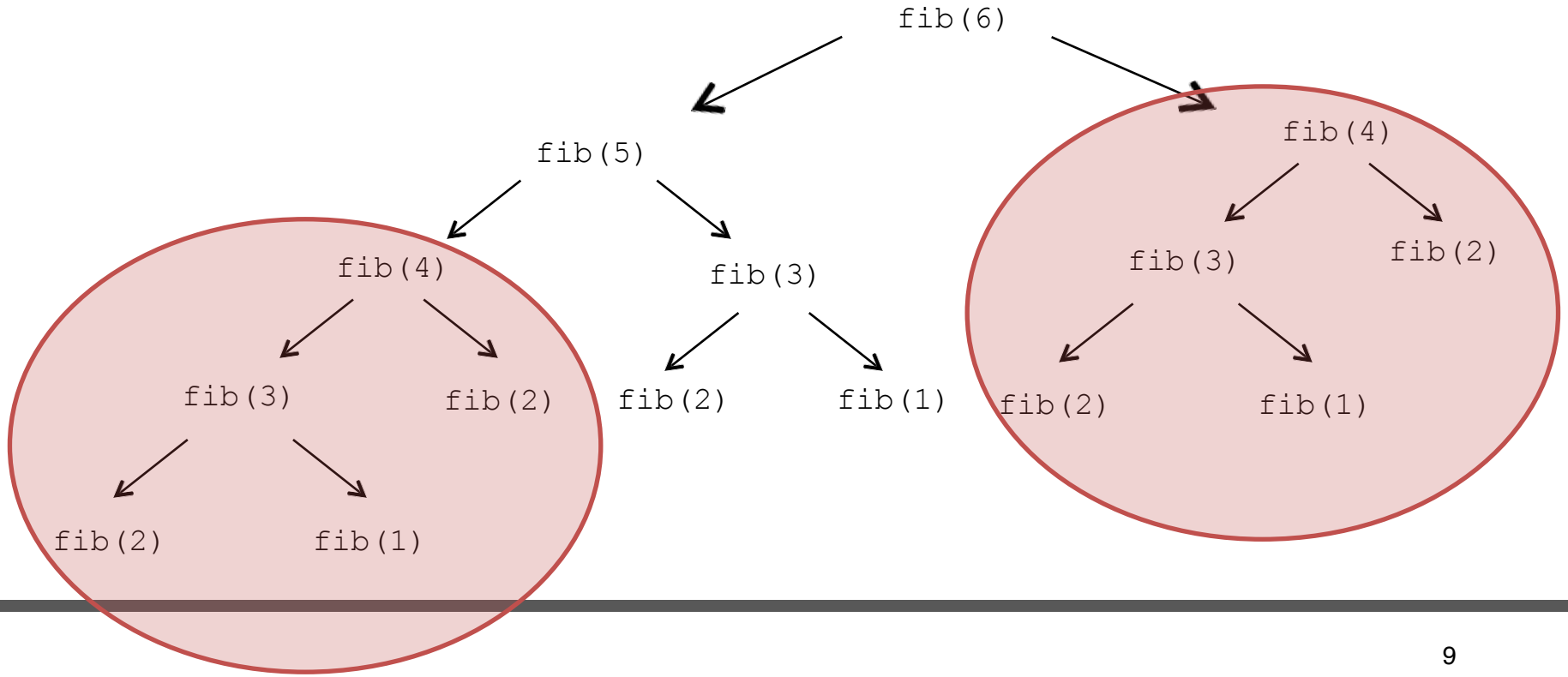
Dynamic programming (intro)
Overlapping Subproblems

```
fib(n) = fib(n-1) + fib(n-2)
```



Dynamic programming (intro)
Overlapping Subproblems

```
fib(n) = fib(n-1) + fib(n-2)
```



Dynamic programming (intro)

Principle of Optimality

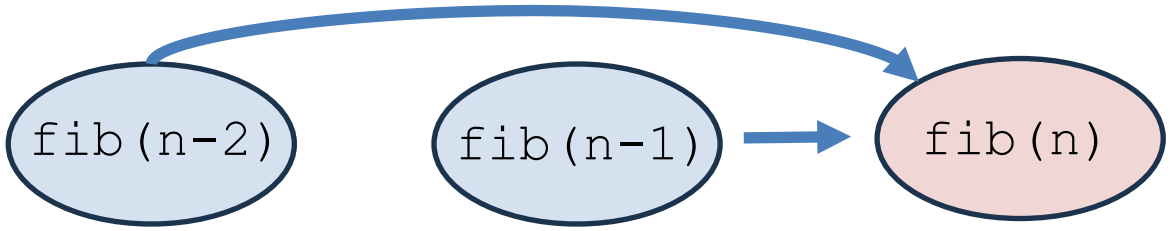
The problem has **optimal substructures**:

- A problem has optimal substructure if the optimal solution can be built from optimal solutions to its subproblems

In other words:

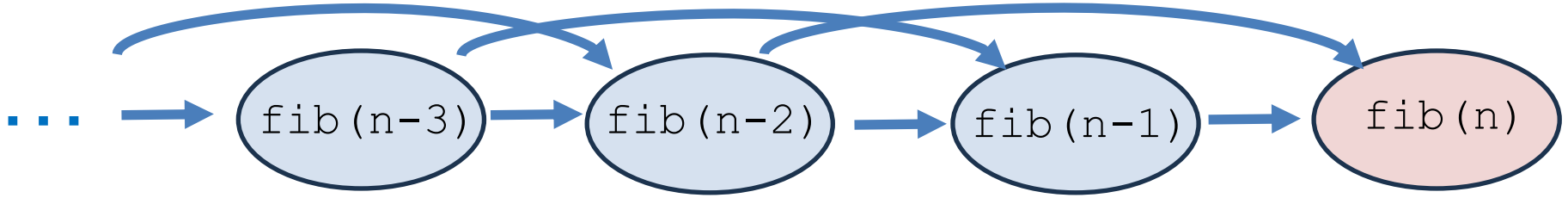
- We can solve main problem using **smaller instance solutions of the same problem**!

Example: Fibonacci



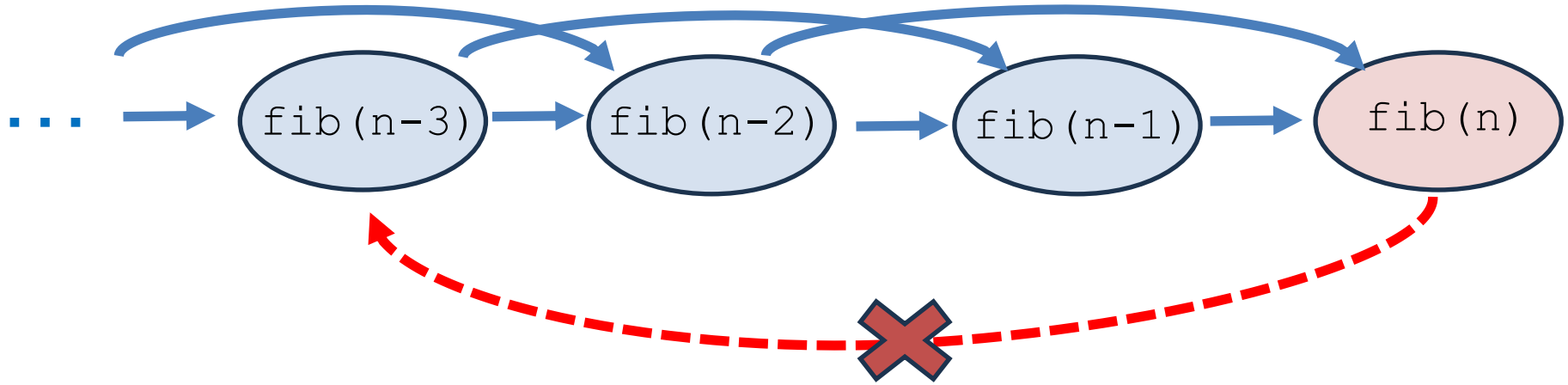
Dynamic programming (intro)
Acyclic Dependencies

The dependencies of the subproblems must be **acyclic**
→ **Directed Acyclic Graph**



Dynamic programming (intro)
Acyclic Dependencies

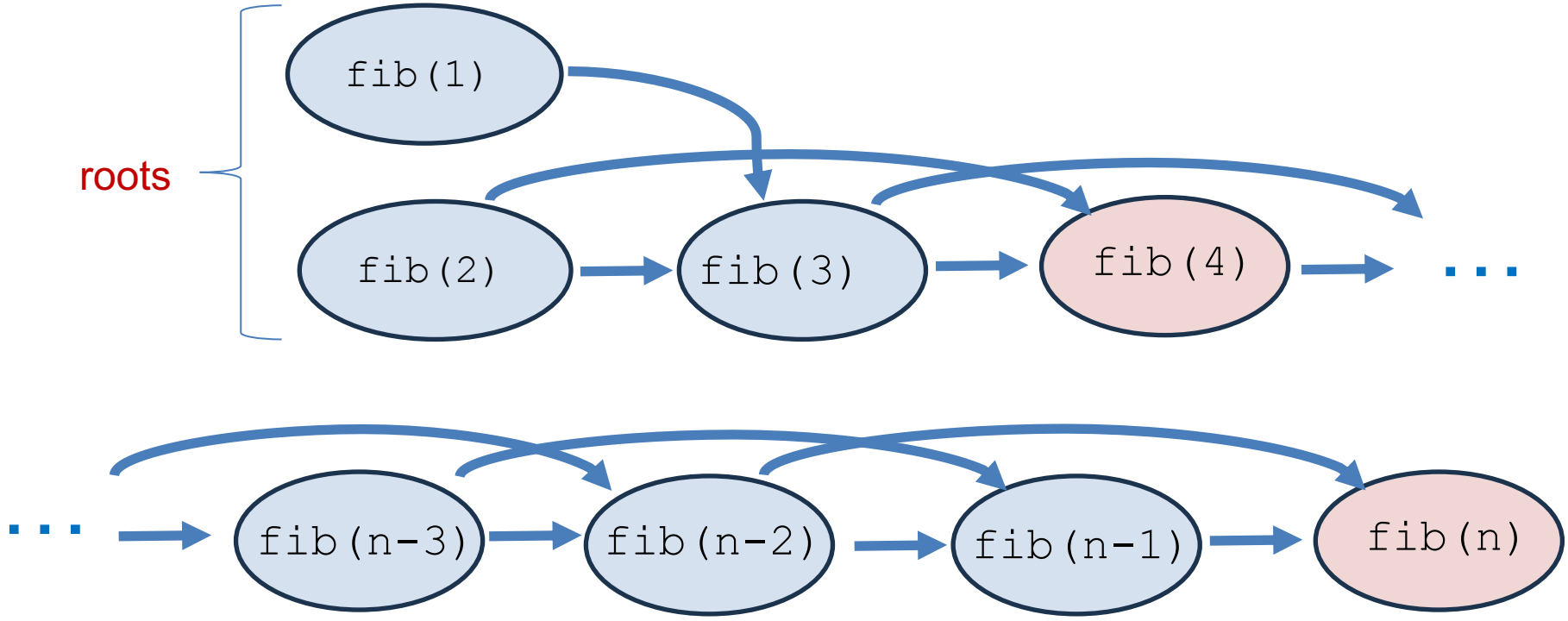
The dependencies of the subproblems must be **acyclic → Directed Acyclic Graph**



If dependencies has cycles, then DP might never converge

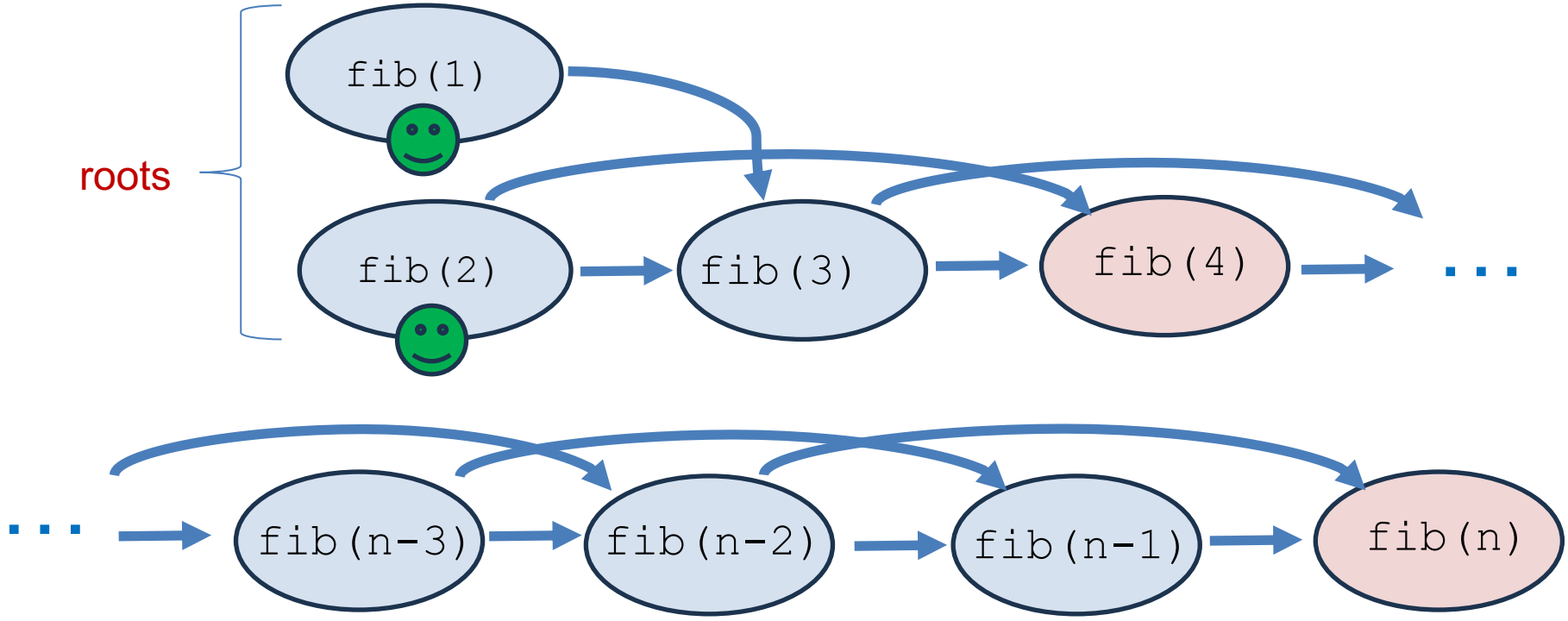
Dynamic programming (intro)
Solution

Find the root(s) and start solving recursively from there:



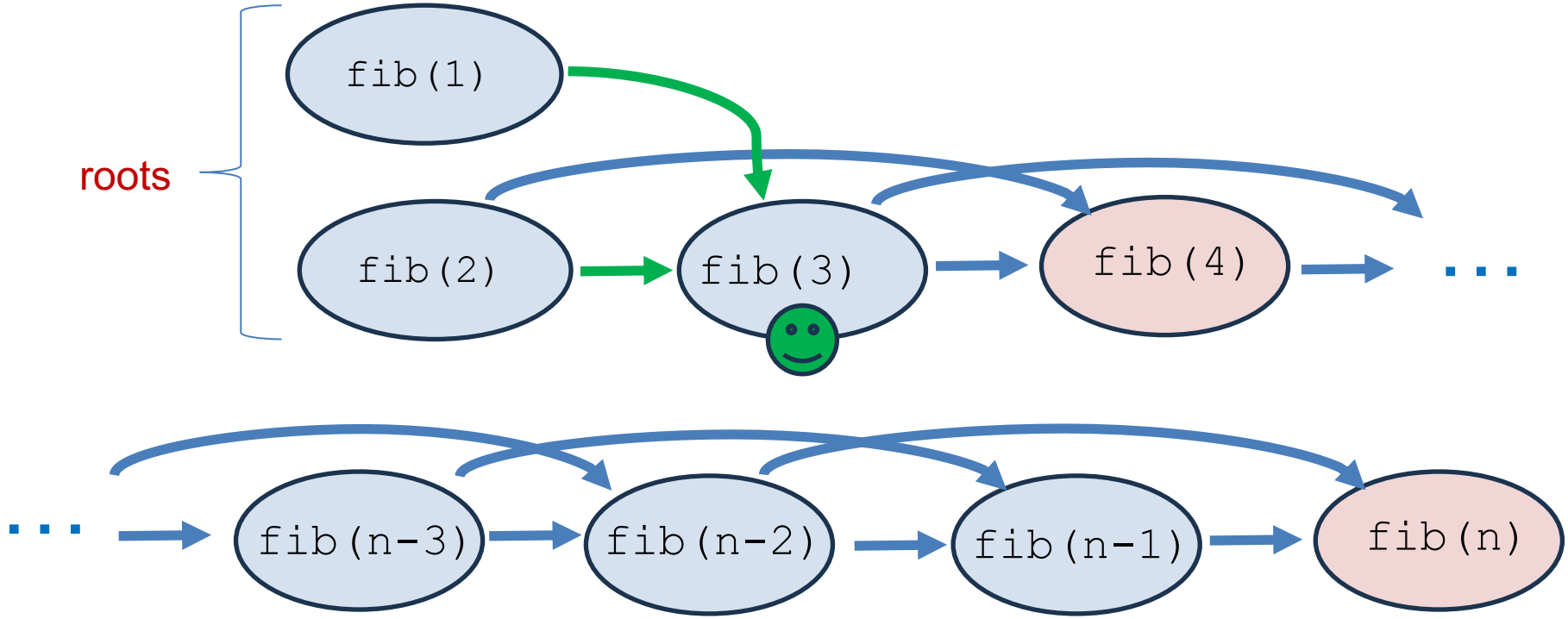
Dynamic programming (intro)
Solution

Find the root(s) and start solving recursively from there:



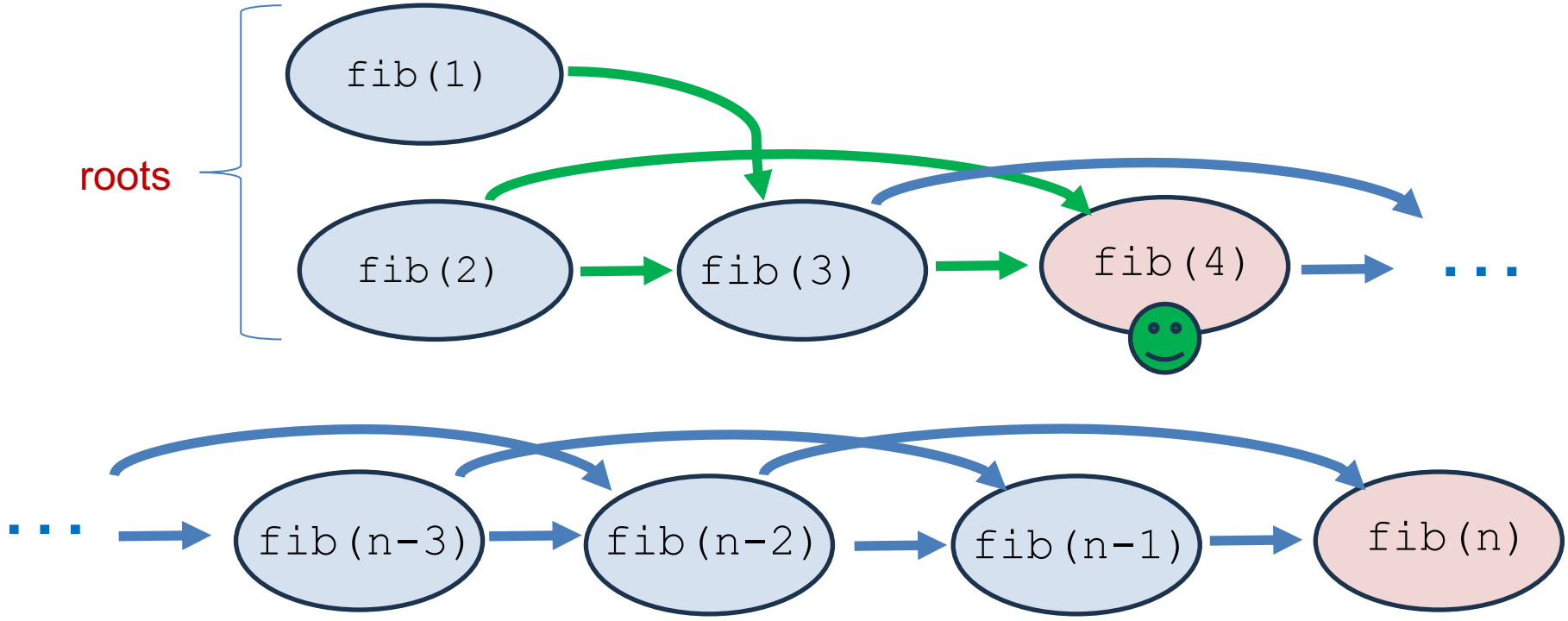
Dynamic programming (intro)
Solution

Find the root(s) and start solving recursively from there:



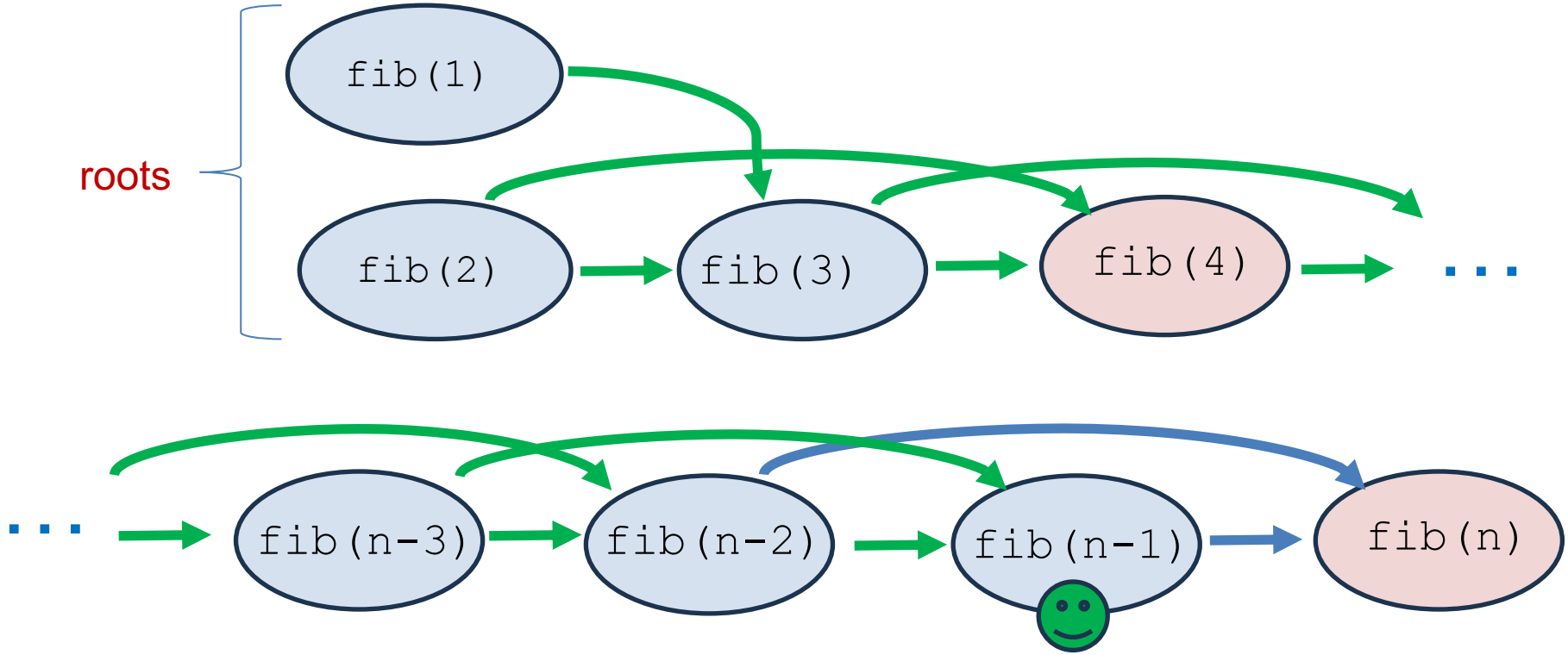
Dynamic programming (intro)
Solution

Find the root(s) and start solving recursively from there:



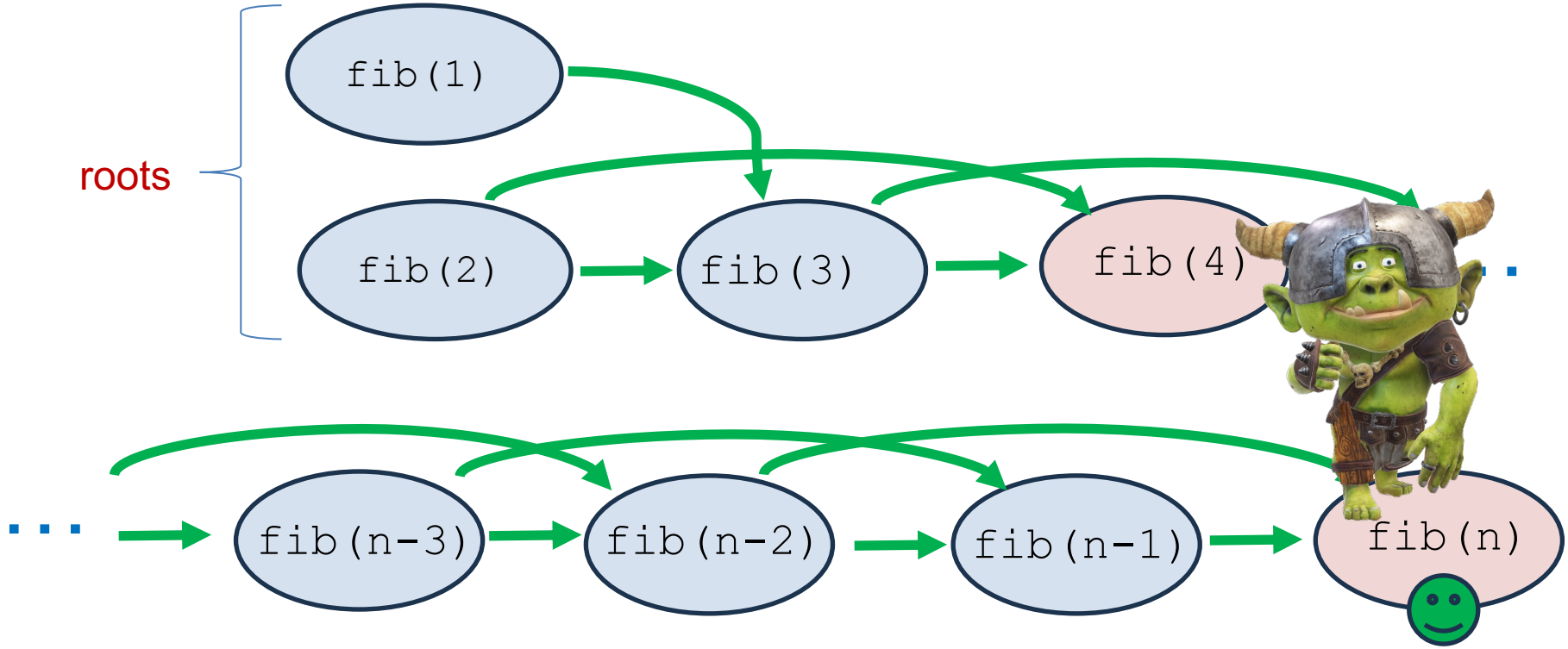
Dynamic programming (intro)
Solution

Find the root(s) and start solving recursively from there:



Dynamic programming (intro)
Solution

Find the root(s) and start solving recursively from there:



Dynamic programming

Reflecting back

If we have

- Overlapping Subproblems ✅

and

- *Principle of Optimality* holds ✅

and

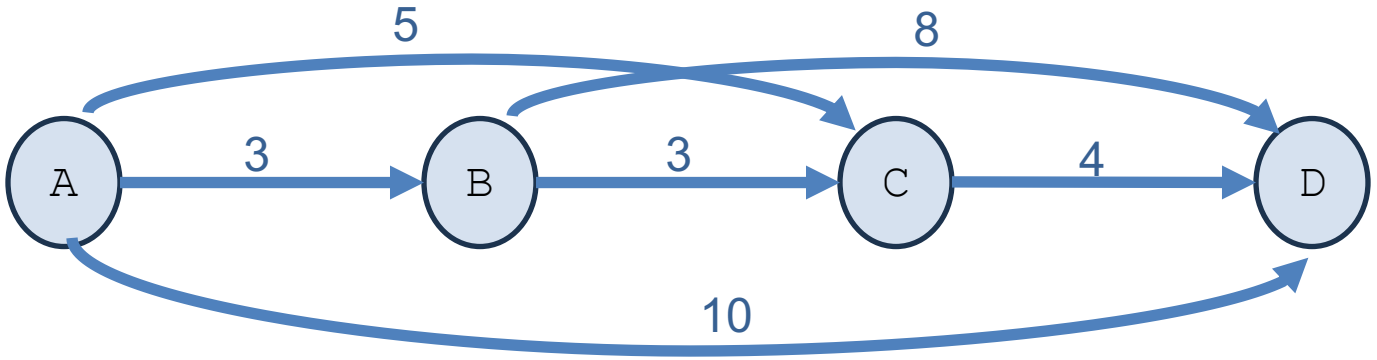
- Dependencies of subproblems are acyclic
  (Directed Acyclic Graph) ✅

Then the DP is the optimal efficient solution!

Application of DP:

Shortest path on Acyclic Directed Graph

Find the shortest path from A to D:

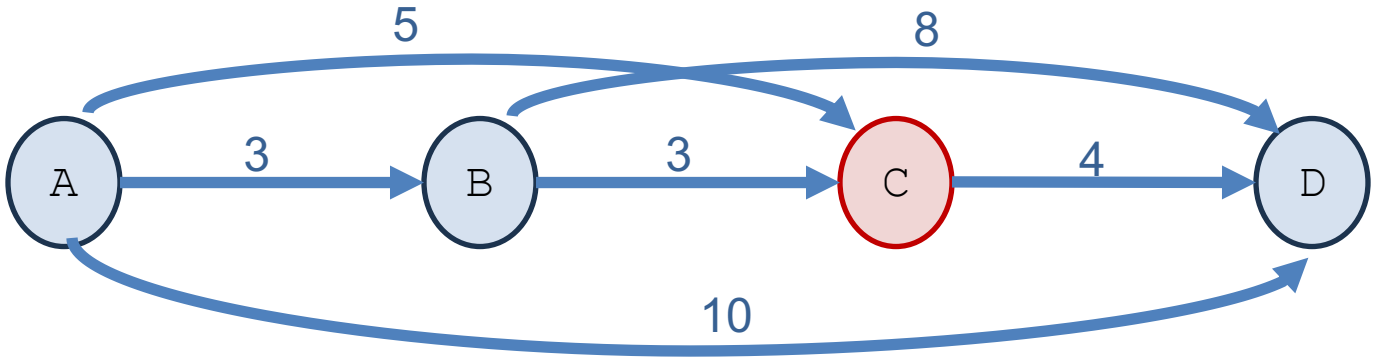


- What are the overlapping subproblems?

Application of DP:

Shortest path on Acyclic Directed Graph

Find the shortest path from A to D:

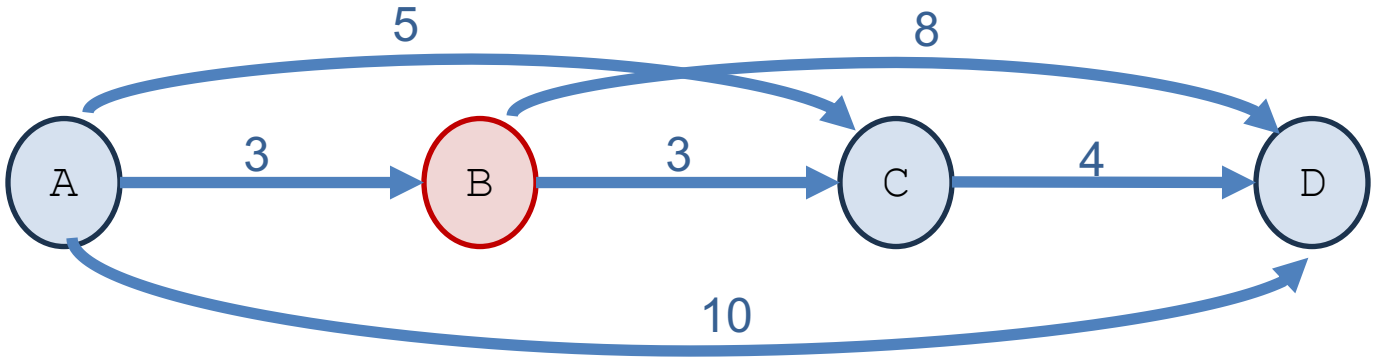


- What are the overlapping subproblems?
  - Optimal path from **first-hop** neighbors of D to D?

Application of DP:

Shortest path on Acyclic Directed Graph

Find the shortest path from A to D:



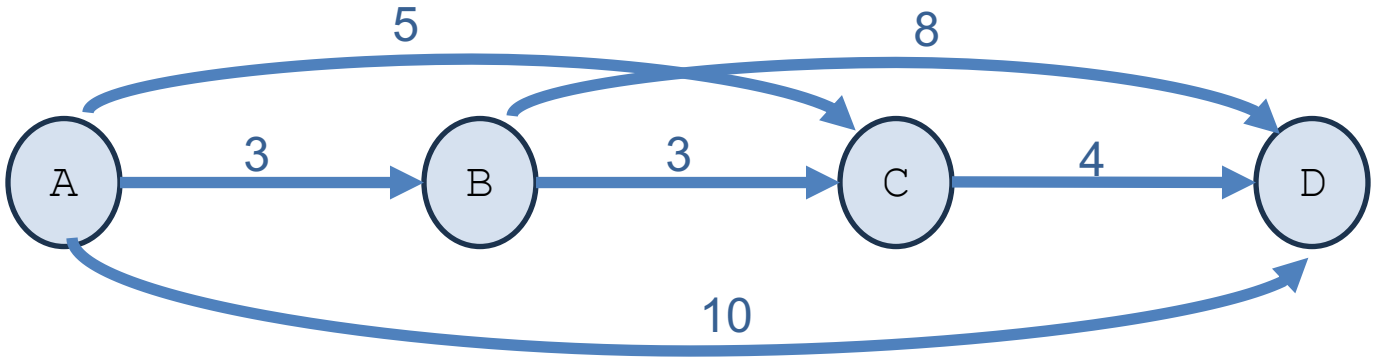
- What are the overlapping subproblems?

  - Optimal path from **first-hop** neighbors of D to D?
  - Optimal path from **second-hop** neighbors of D to D?
  - …

Application of DP:

Shortest path on Acyclic Directed Graph

Find the shortest path from A to D:

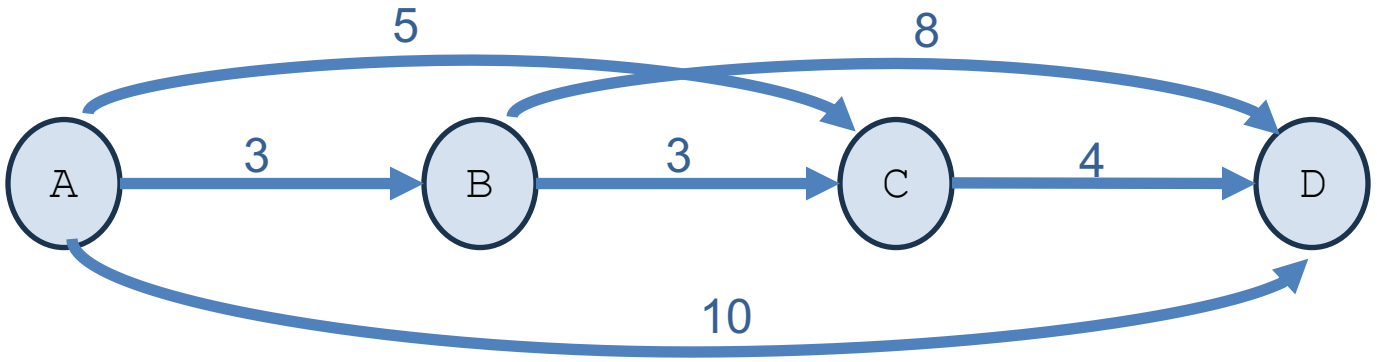


- Does Principle of Optimality hold?

if x is a node on the shortest path A….x….D from A to D
then x….D is also shortest path from x to D

Application of DP:

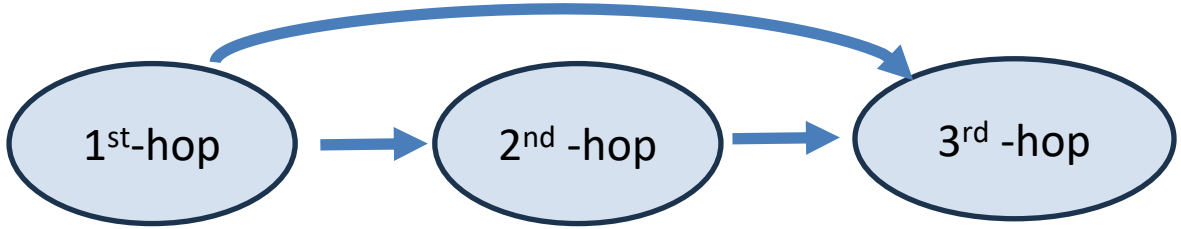
Shortest path on Acyclic Directed Graph



- Are the dependencies acyclic?

Because the graph is acyclic,

the n^{th}-hop neighbor subproblems form an acyclic directed graph



A→D : 10
B→D : 8
C→D : 4

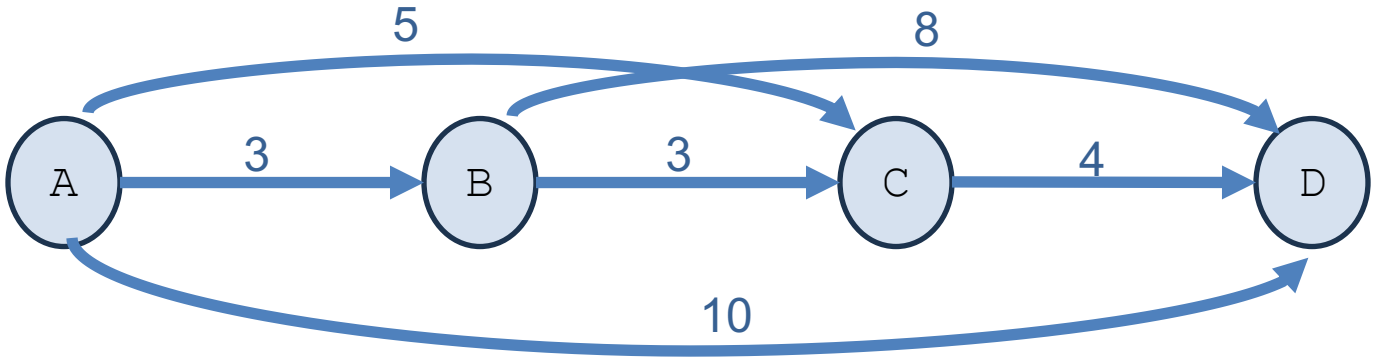
A→B→D: 3+(8)
A→C→D: 5+(4)
B→C→D: 3+(4)

A→B→C→D : 3+(3+(4))

Application of DP:

Shortest path on Acyclic Directed Graph

Find the shortest path from A to D:



- What are the overlapping subproblems?
- Does Principle of Optimality hold?
- Are the dependencies Acyclic?

What if there are also Cycles? **Dijkstra's algorithm**