FIBONACCI, SHORTEST PATH

Dynamic Programming:

general, powerful (DP)
alg. design technique

* DP ≈ "careful brute force"
* DP ≈ subproblems + "reuse"

Fibonacci numbers

$$F_1 = F_2 = 1$$

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

goal: compute F_n

Naive recursive algo:

```
fib(n):
    if n ≤ 2 : f = 1
    else f = fib(n-1) + fib(n-2)
    return f
```

EXPONENTIAL

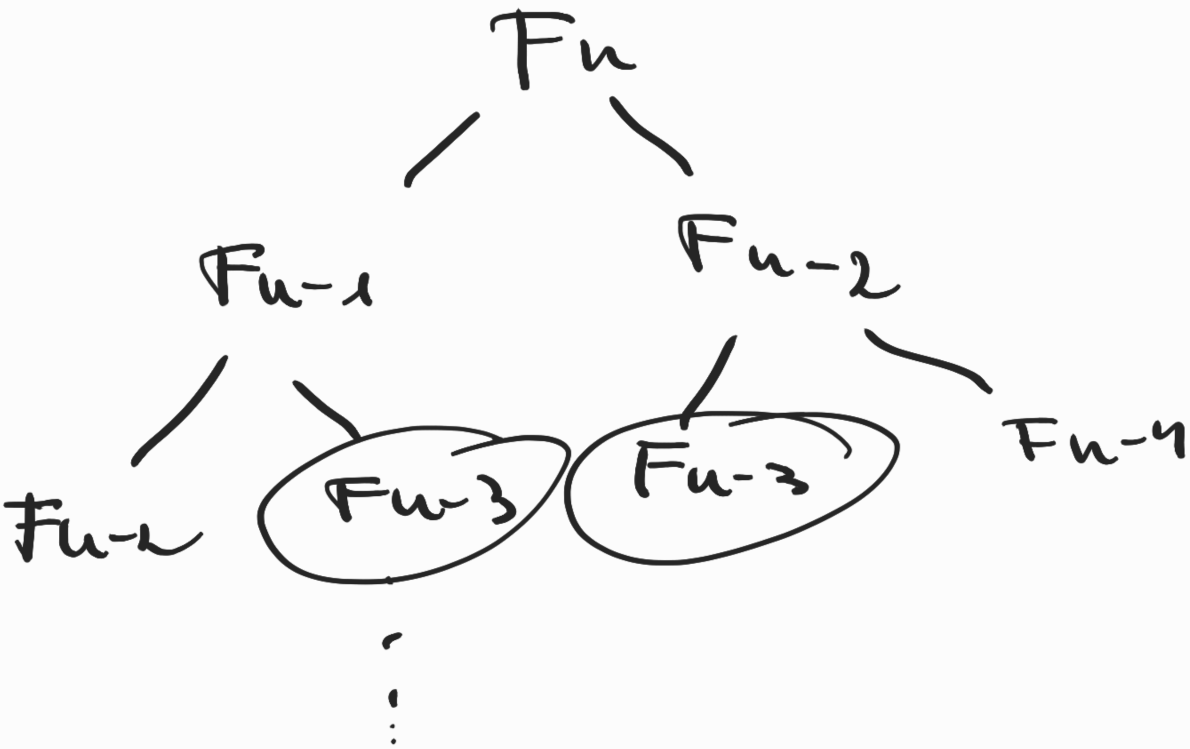
$$T(n) = T(n-1) + T(n-2) + \Theta(1)$$

$\geq F_n \approx \varphi$

$$T(n) \geq 2T(n-2) = \Theta(2^{n/2})$$

<u>memorized DP algo:</u>

```
memo = {}
fib(n):
  if n in memo: return memo[n]
  if n ≤ 2: f = 1
  else: f = fib(n-1) + fib(n-2)
  memo[n] = f
  return f
```



fib(k) only recurses the first time it's called. $\forall k$ memoized calls cost $\Theta(1)$

- memorized calls ...
- # nonmemorized calls is n
fib(1), fib(2) ..., fib(n)

=> time $\Theta(n)$

DP ≈ recursion + memoization
- memoize (remember) & re-use solutions to subproblems that help solve the problem

=> time = # subproblems · $\frac{\text{time}}{\text{subproblems}}$

$\Theta(1)$

don't count recursions

bottom-up DP algo

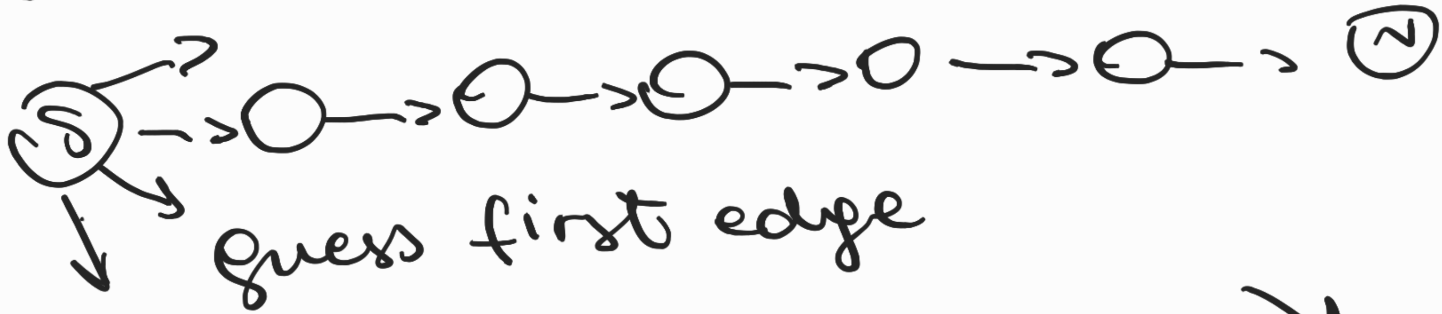
```
fib = {}
for k in range(1, n+1):
    if k ≤ 2: f = 1
    else: f = fib[k-1] + fib[k-2]
    fib[k] = f
return fib[n]
```

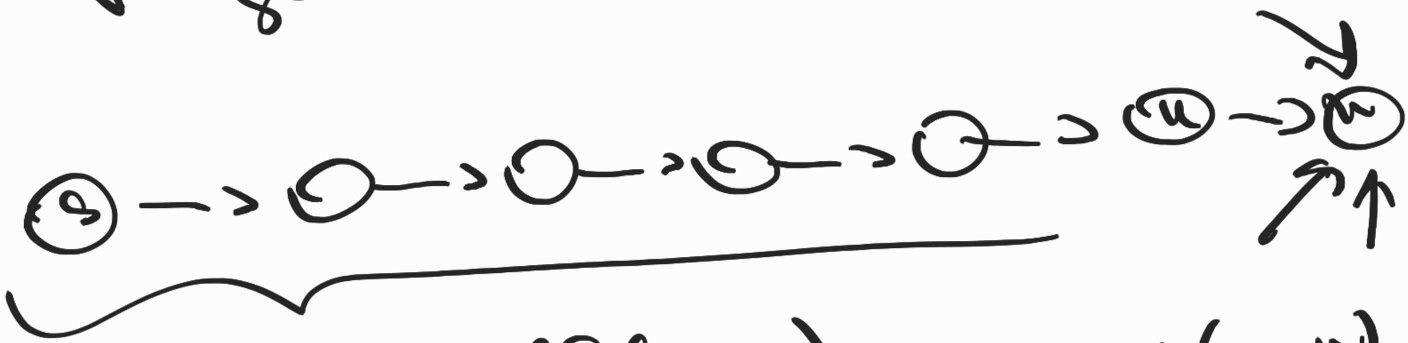
- exactly same computation

- topological sort of subproblem dependency DAG

Shortests path:

$\delta(s,v)\ \forall v$





$\delta(s,v) = \min(\delta(s,u)$ $\qquad w(u,v)$

DAGs: $\Theta(V+E)$

time for subprob. $\delta(s,v)$
$= \text{indegree}(v) + 1$

$\rightarrow$ total time $= \sum_{v \in V} \text{indeg}(v) = \Theta(E+V)$

* subproblem dependencies should be acyclic