DP vs memoization

Fiboracci #s

$f(0) = 1 \quad , \quad f(1) = 1$

$$f(n) = f(n-1) + f(n-2) \qquad \forall \quad n \geq 2$$

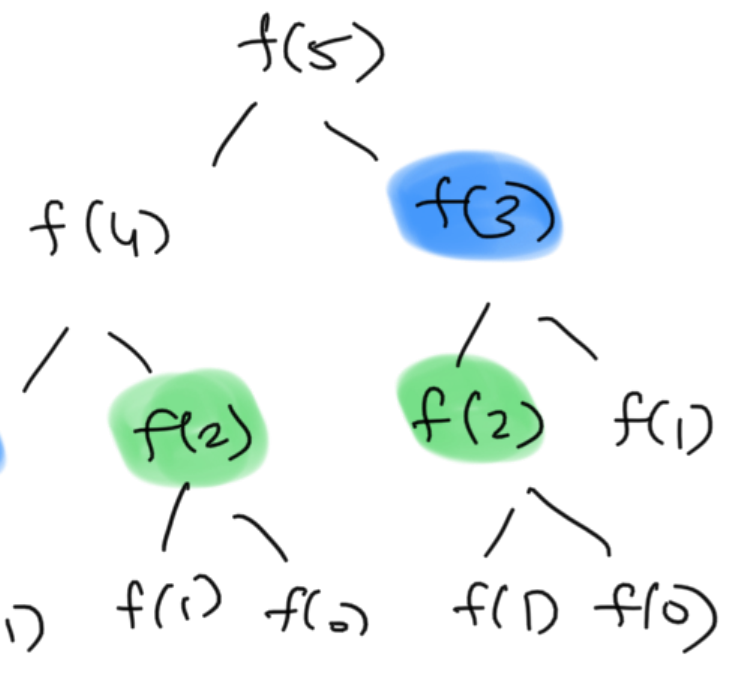
Recursive way of computing f(5)

$f(5) = f(4) + f(3)$

$f(4) = f(3) + f(2)$

$f(3) = f(2) + f(1)$

$f(2) = f(1) + f(0)$



↑

too many needless recomputations

Memoization

→ Store f(i) on the way if you compute it

→ Before computing f(k), look up if you already have it

memodict = { 0 : 1 , 1 : 1 }

f(0) = 1 f(1) = 1

```
def fib (n) :
    if n in memodict,
        return memodict [n]
memodict [n] = fib(n-1) + fib(n-2)
    return memodict [n]
```

memodict

0 : 1

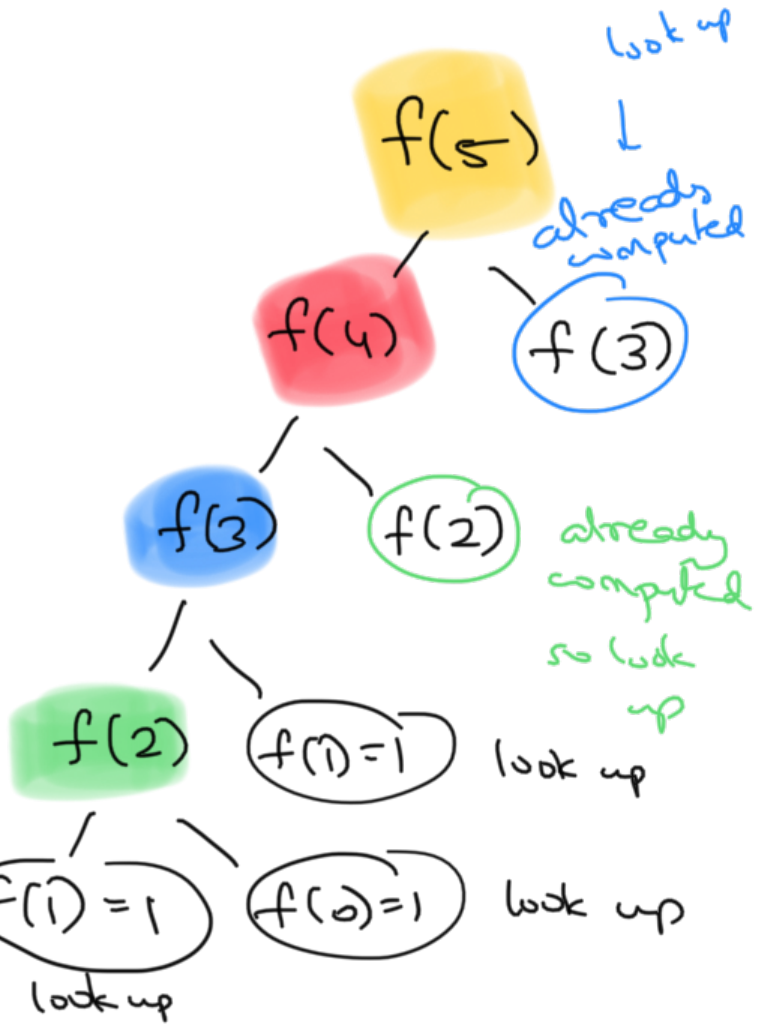
1 : 1

2 : 2

3 : 3

4 : 5

5 : 8



Dynamic programming

- Anticipate how memodict builds up
- Eliminate recursion
- Dependencies form dag
- Solve subproblems in topological order

DAG

Dependency chart

DP

5 , 4 , 3 , 2 , 1 , 0

Compute Fib(i) for i in this order

```
def fib_dp(n):
    memodict = {0:1, 1:1}
    for i in [2,3,...n]:
        memodict[i] = memodict[i-1] + memodict[i-2]
    return memodict[n]
```

Don't need to store values actually

```
def fib_betterdp(n):
    a = 1
    b = 1
    for i in [2,3 ... n]:
        answer = a+b
        a = b
        b = answer
    return answer
```

fib_betterdp(5):

0 : 1

1 : 1

2 : 2

3 : 3

4 : 5

5 : 8

a = 1
b = 1

i = 2

ans = 2

a = 1

b = 2

i = 3

ans = 3

a = 2

b = 3

i = 4

ans = 5

a = 3

b = 5

i = 5

ans = 8

a = 5

b = 8