Saving redundant work

- We won’t look at dynamic programming in this course
- But we can show a simpler version of the underlying idea, which is still valuable to know and utilize

- Note that in exploring tree, we may compute an answer to the same subproblem several times
  - What is the best answer using items {b,c,d} before deciding to include item {a}; similarly for {c,d} and so on

Saving redundant work

- We might be much more efficient if we could keep track of partial solutions, and then just look up the answer somewhere when we need to use this computation again
- Let’s illustrate this idea with a simpler example – our old friend, Fibonacci

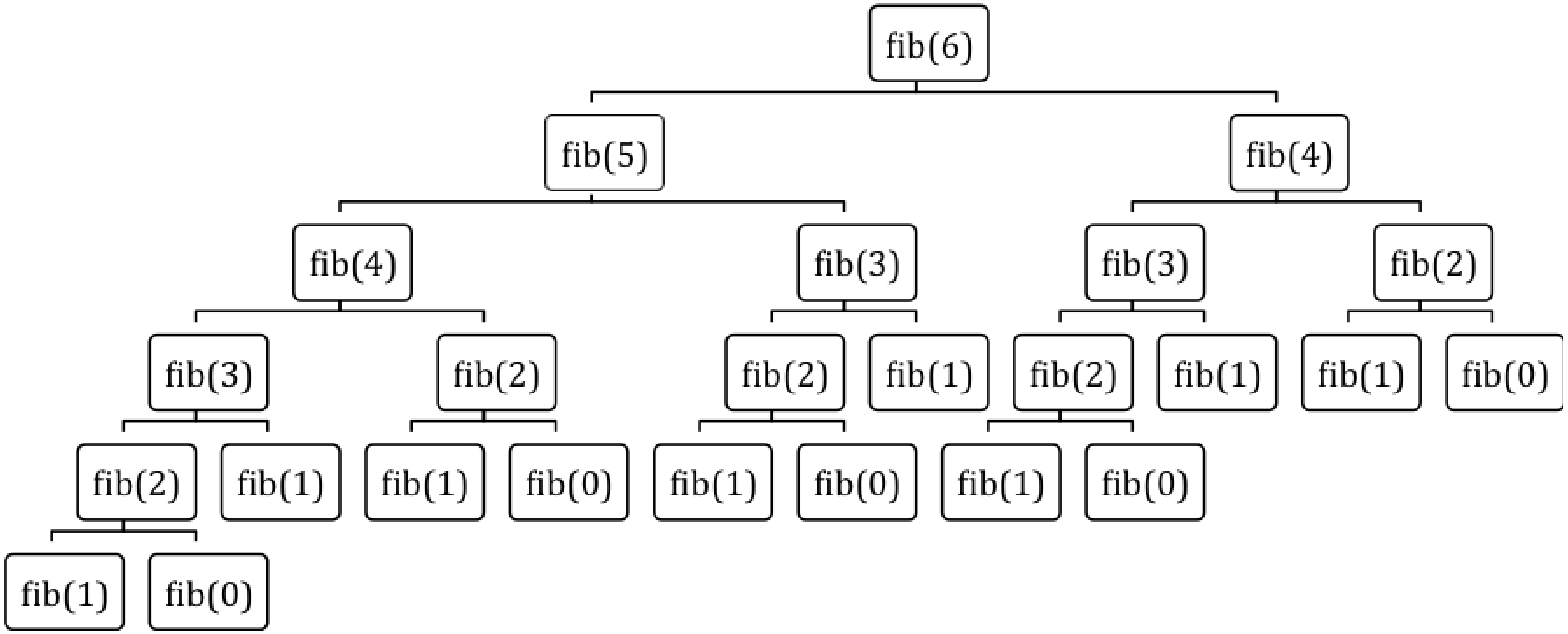
Standard Fibonacci

```
def fib(x):
    assert type(x) == int and x >= 0
    if x == 0 or x == 1:
        return 1
    else:
        return fib(x-1) + fib(x-2)

def testFib(n):
    assert type(n) == int and n >=  0
    for i in range(n):
        print ('fib of', i, '=', fib(i))
```

Improving Fibonacci

- This can be very slow
- It is repeating computations multiple times


fib(6)
fib(5)
fib(4)
fib(4)
fib(3)
fib(3)
fib(2)
fib(3)
fib(2)
fib(2)
fib(1)
fib(2)
fib(1)
fib(1)
fib(0)
fib(2)
fib(1)
fib(1)
fib(0)
fib(1)
fib(0)
fib(1)
fib(0)
fib(1)
fib(0)

Saving redundant work

- Note how many different times we compute fib(3) or fib(2)
- Since we know that fib does not do any side-effects, i.e., it doesn’t mutate an internal variable, the answer is not going to be different
- So can we save this extra effort?
- **Memoization** – the first time we compute a function, keep track of the value; any subsequent time, just look up the value

```
def fastFib(x, memo):
    assert type(x) == int and x >= 0 and type(memo) == dict
    if x == 0 or x == 1:
        return 1
    if x in memo:
        return memo[x]
    result = fastFib(x-1, memo) + fastFib(x-2, memo)
    memo[x] = result
    return result

def testFastFib(n):
    assert type(n) == int and n >=  0
    for i in range(n):
        print ('fib of', i, '=', fastFib(i, {}))
```

Memoization saves time

- Under this method, we only compute fib(n) once, thus saving a great deal of additional effort
- This idea can be used to improve other methods (check out dynamic programming)
- Memoization only applies if there is no mutation of internal variables