Lecture 23: Intro to RL

- Dynamic Programming
- Markov Decision Processes
- RL: MDPs + learning from data
  + function approx.

Dynamic Programming

- Makes recursion more efficient by re-using answer to same function call

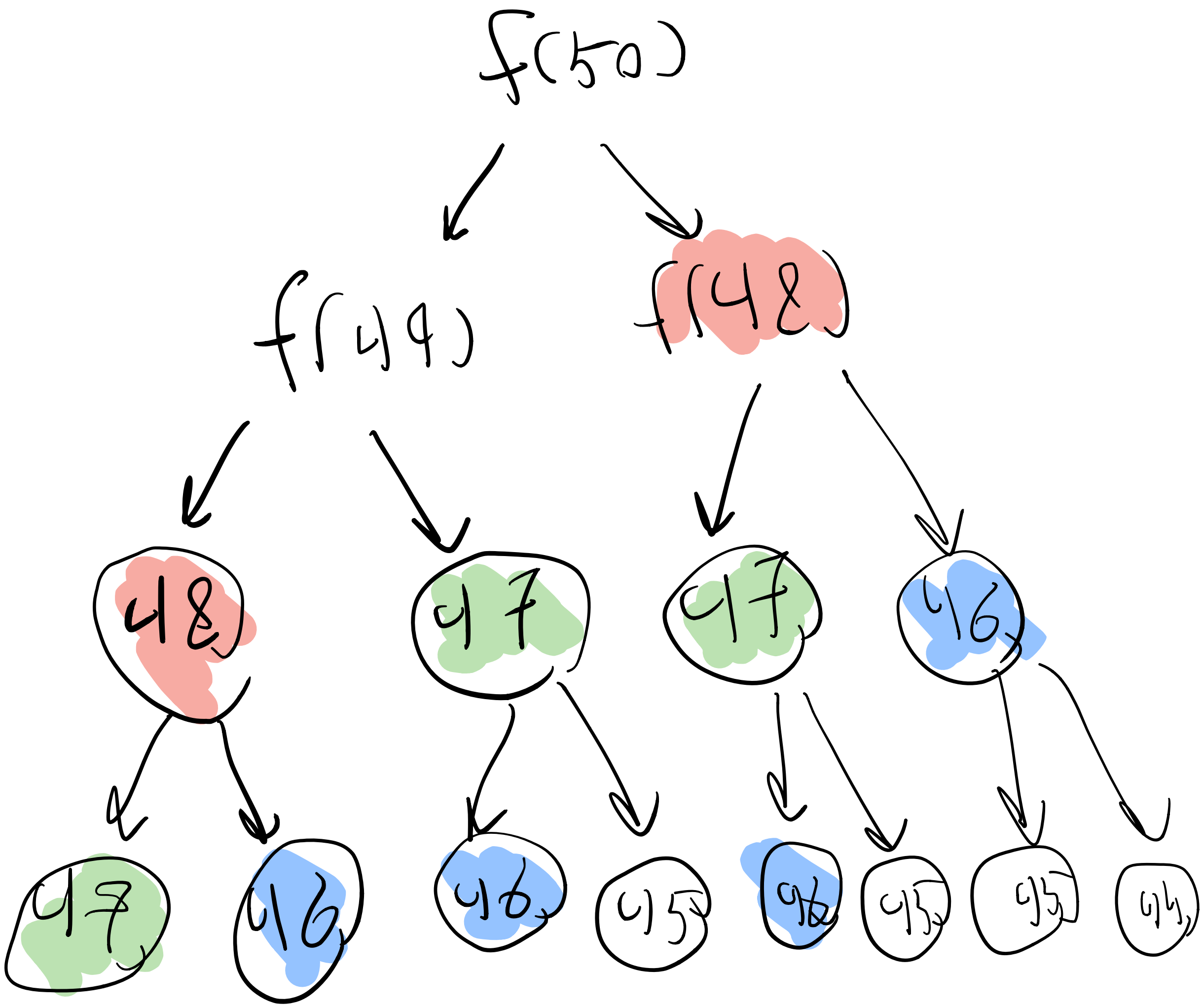
Fibonacci $f(0) = f(1) = 1$

$f(n) = f(n-1) + f(n-2)$

```
def fib(n):
    [illegible] 1:
```

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

f(50)



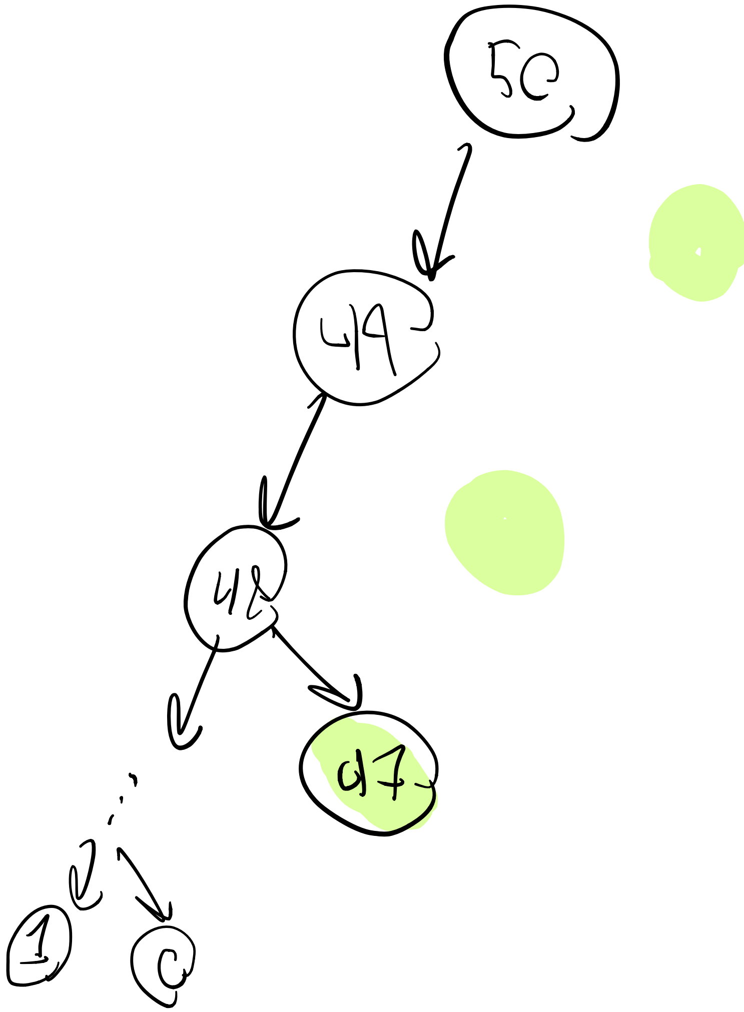
Memoization:

```
memo = dict()
def fib(n):
    if n in memo.keys():
        return memo[n]
    else:
        if n <= 1:
            memo[n] = 1
        else:
            meme[n] = fib(n-1) + fib(n-2)
    return memo[n]
```

memoizing

Python decorators

@memoize



Dynamic programming

"unroll" recursion

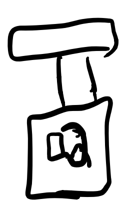
```
import numpy as np
fib = np.zeros(shape=(n+1,))
```

```
fib[0] = 1   # e!
fib[1] = 1
for i in range(2, n+1):
    fib[i] = fib[i-1] + fib[i-2]
```

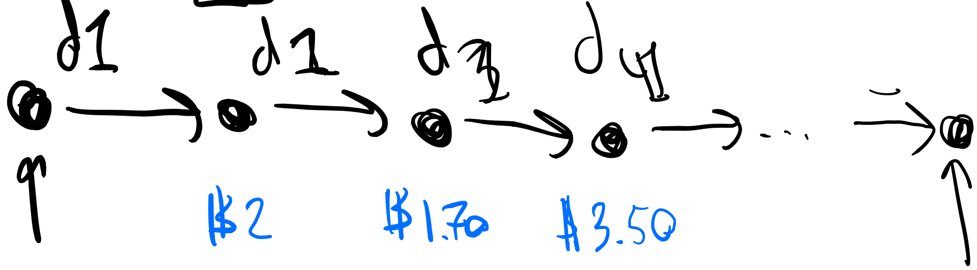
Memoization
- slower (cache)
- easy

DP:
- fast (loops)
- requires laying out computation

Gas stations



loc i:
- g_i units of gas
- c_i dollars per unit



0 $\quad$ 6 $\quad$ 3 $\quad$ 10 $\quad$ N

- 1 unit of gas to move 1 unit to right

How much gas should we buy at each location to minimize total cost?

State:

- location
- how much gas is in the tank?

$f(\text{loc}, \text{gas})$: ← minimum cost to get to end given current state

"cost-to-go"

↓

- buy 1 unit of gas,

stay where we are

- go forward

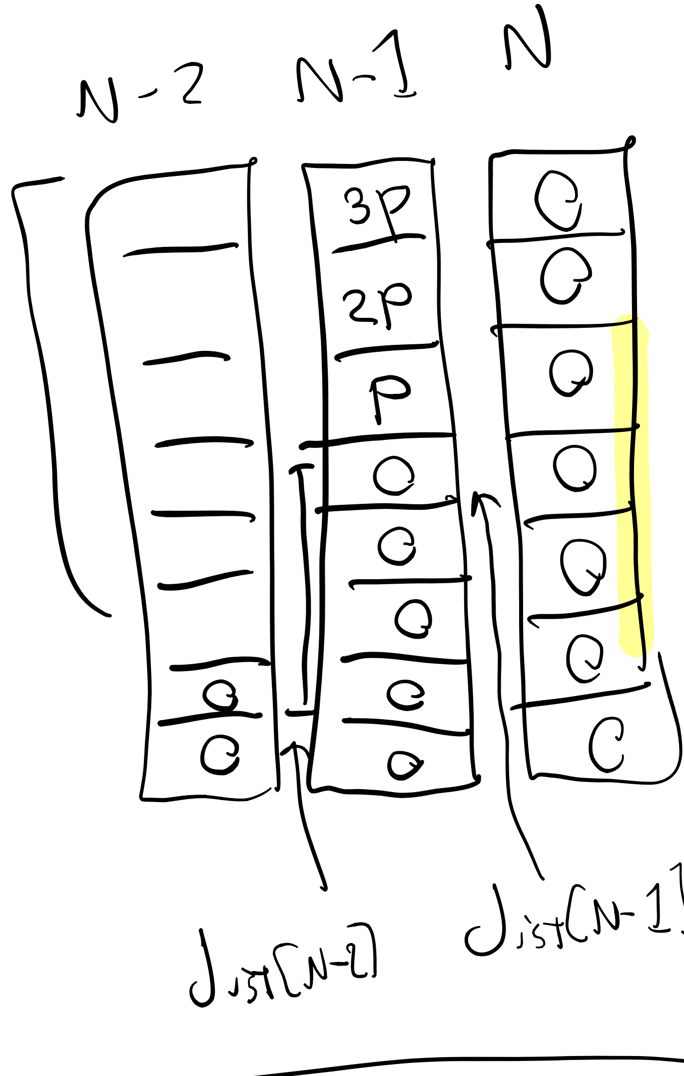
(b) memoize

```
def f(loc, gas):
    cost1 = f(loc, gas+1) + price[loc]
    cost2 = f(loc+1, gas - dist[loc])
    return min(cost1, cost2)

    if loc==N:
        return 0
    if gas < 0:
        return -math.inf

f = np.zeros(shape=(N+1, N+1))
for loc in range(N, 0, -1):
```

```
for year in range(N, 0, -1):
    f[loc, year] =
        min( .... )
```



Solving MDPs with dynamic programming

$$V(s) = \min_{\underline{a}} \sum_{s'} P(s' \mid a, s) \cdot \left(R(s,a,s') + \gamma \cdot V(s') \right)$$

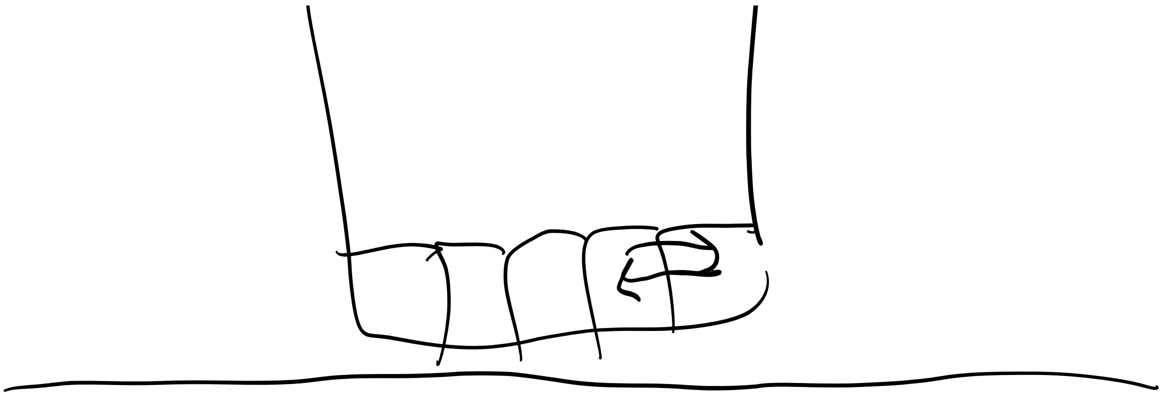
def V(s):

cost = math.inf

for a in actions:

cost = min(cost, $\sum_{s'} P(s', a, s) \cdot \left(R(s,a,s') + \gamma \cdot V(s') \right)$)

return cost



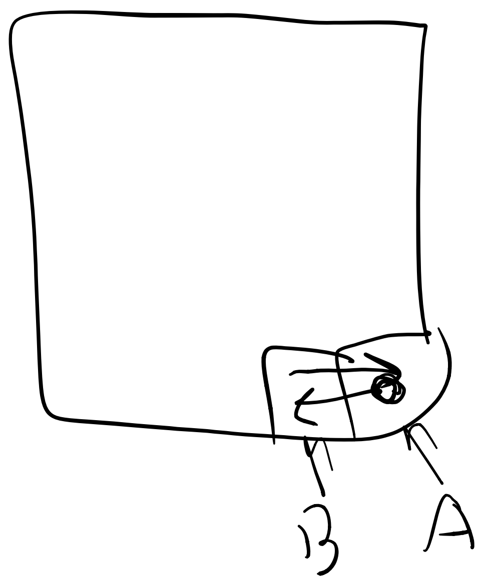
Time horizon:

- Final time T

State: s replaced with (s, t) $t \in \{0, \dots, T\}$

```
def V(s, t):
    if t == T:
        return 0
    else:
```

$$\min_{a} \sum_{s'} P(s'|a,s) \cdot \left(R(s,a,s') + \gamma \cdot V(s', t+1) \right)$$



$A \to B \to A$

$(A, 0)$

$\downarrow$

$(B, 1)$

$\downarrow$

$(A, 2)$

```
V = np.zeros(shape=(S, T+1))
for t in range(T-1, 0, -1):
```

for s in states:

$$V[s, t] = \min_a \sum_{s'} P(\ldots) \cdot \left(R(\ldots) + \gamma \cdot V[s, t+1]\right)$$

Value iteration