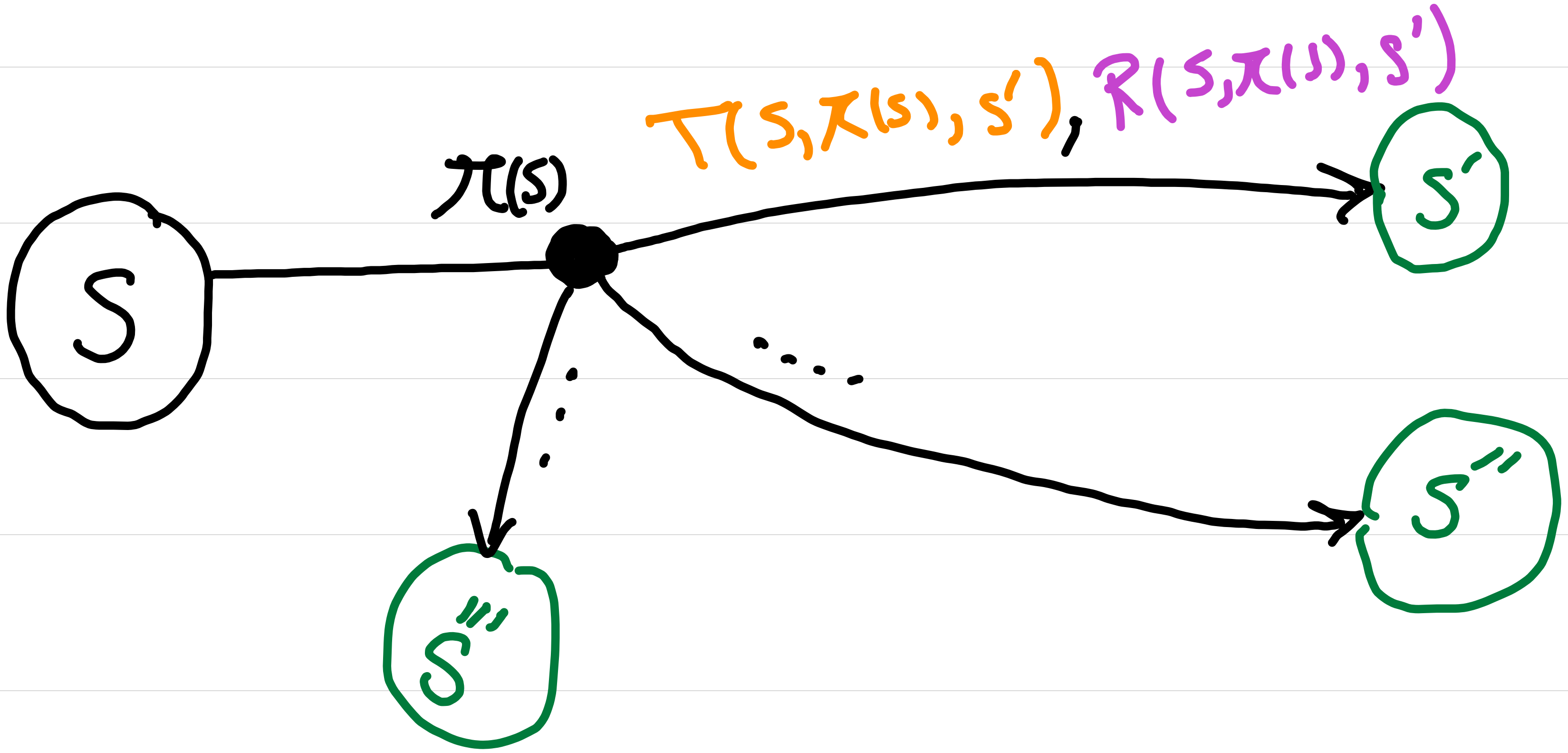
Welcome to Week 12: Solving MDP

- Outline:
  - Review: Computing value of states, given policy π
  - Dynamic Programming: An Iterative Algorithm to find the values given a policy
  - Solving MDP: Computing the best policy
    - Value iteration and Policy Extraction
    - Policy iteration: Next part of the lecture

Review: Markov Property Provides a Nice Structure



with probability $T(s,\pi(s),s')$, you will receive the immediate reward $R(s,\pi(s),s')$ and go to the next state s'. From state s', your expected utility that you would receive is $V_\pi(s')$. This is the reward you receive in the next step. Hence, you should discount it by γ

$$
\begin{aligned}
V_\pi(s) = {} & T(s,\pi(s),s')\left[R(s,\pi(s),s') + \gamma V_\pi(s')\right] \\
& + T(s,\pi(s),s'')\left[R(s,\pi(s),s'') + \gamma V_\pi(s'')\right] \\
& + T(s,\pi(s),s''')\left[R(s,\pi(s),s''') + \gamma V_\pi(s''')\right]
\end{aligned}
$$

Markov Property Provides a Nice Structure

- Given Policy π, the value of states statisfy

$$V_\pi(s) = \sum_{s' \in S} \underbrace{T(s,\pi(s),s')}_{\text{probability of this transition}} \Big[\underbrace{R(s,\pi(s),s')}_{\text{immediate Reward for this transition}} + \underbrace{\gamma\, V_\pi(s')}_{\text{Discounted Expected Return after this transition}} \Big], \quad \forall s \in S$$

Markov Property Provides a Nice Structure

- For finite state MDP, we can express the previous equations as a matrix equation

Let $\underline{V} = \begin{bmatrix} V_\pi(s) \\ V_\pi(s') \\ \vdots \end{bmatrix}$, $T = \begin{bmatrix} T(s,\pi(s),s) & T(s,\pi(s),s') & \cdots \\ T(s',\pi(s'),s) & T(s',\pi(s'),s') & \cdots \\ \vdots & & \ddots \end{bmatrix}$, $R = \begin{bmatrix} R(s,\pi(s),s) & R(s,\pi(s),s') & \cdots \\ R(s',\pi(s'),s) & R(s',\pi(s'),s') & \cdots \\ \vdots & & \ddots \end{bmatrix}$

$$\underline{V} = (T \odot R)\underline{1} + \gamma T \underline{V}$$

- Thus, $\underline{V} = (I - \gamma T)^{-1} (T \odot R)\underline{1}$
- Solving directly requires taking a matrix inverse $\sim \mathcal{O}(|S|^3)$

Iterative Algorithm for Computing Values, Given a Policy

- Dynamic Programming:

  Initialize $V_{\pi,0}(s) = 0$ for all $s \in \mathcal{S}$

  For $K=1$, until Convergence:

  For all $s \in \mathcal{S}$:

  $$V_{\pi,k}(s) = \sum_{s' \in \mathcal{S}} T(s, \pi(s), s') \left[R(s, \pi(s), s') + \gamma V_{\pi,k-1}(s') \right]$$

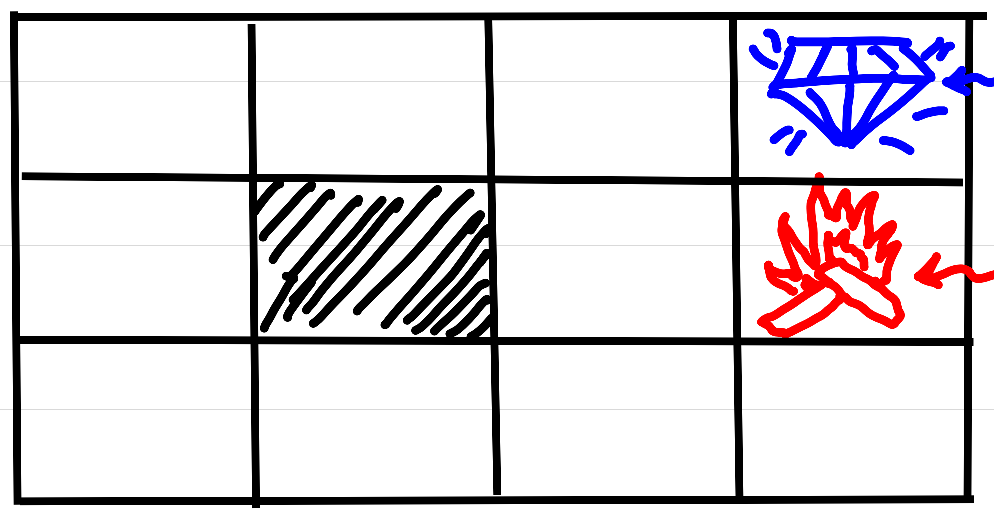
- Computational Complexity For Each Iteration: $O(|S|^2)$

Solving MDP

- An MDP can be specified with a tuple: (S, A, T, R, γ)
- To solve an MDP means: Find the best/optimal policy.
- What is the definition of the "best" policy?
  - Roughly, the best policy gives you a lot of reward.
  - policy π is better than equal to policy π', denoted by $\pi \succeq \pi'$, iff for all $s \in S$, $V_\pi(s) \geq V_{\pi'}(s)$
- Note: There always exists at least one policy which is
- better than equal to all other policies

Example: Solving Mars Rover in 2D-Grid

- Consider the following Mars Rover in 2D Grid.



Mars Rover gets a reward of (+1) if gets to this location. Then, game ends.

Mars Rover gets a reward of (−1) if it gets to this location. Then, game ends.

Noise = 0.2, Discount = 0.9, Living reward = 0

- What would the solution of this MDP look like?

2D-Grid Mars Rover: Optimal Policy

- Optimal Policy π^*:

→	→	→	
↑		↑	
↑	←	↑	←

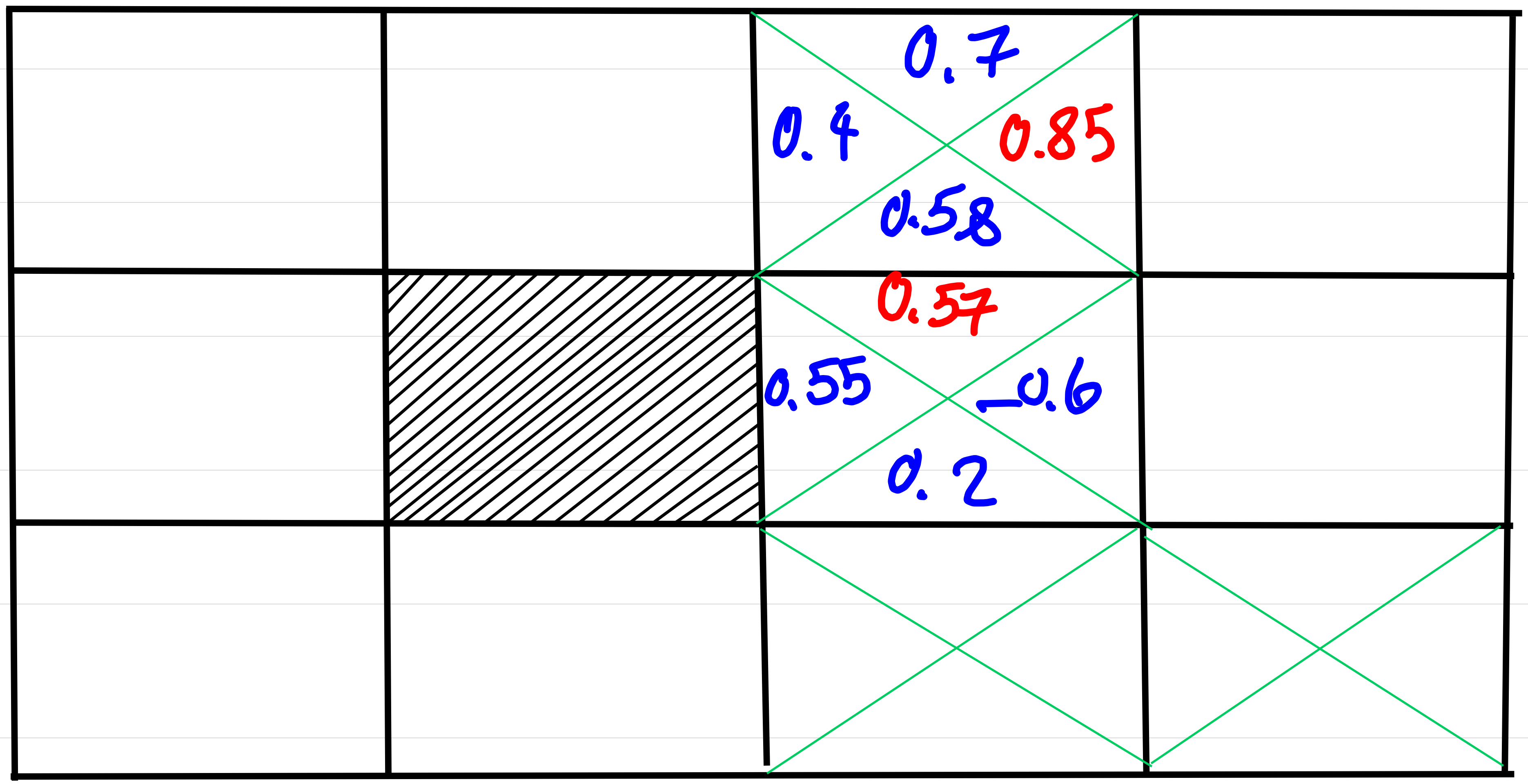
- V values of the optimal policy:

		0.85	+1
		0.57	-1
		0.48	0.28

- Q-value of state-action pair:

  The expected utility of taking action "a" from state "s", and then following the policy

$$V_\pi(s) = Q_\pi(s, \pi(s))$$



Values of States: Bellman Equation

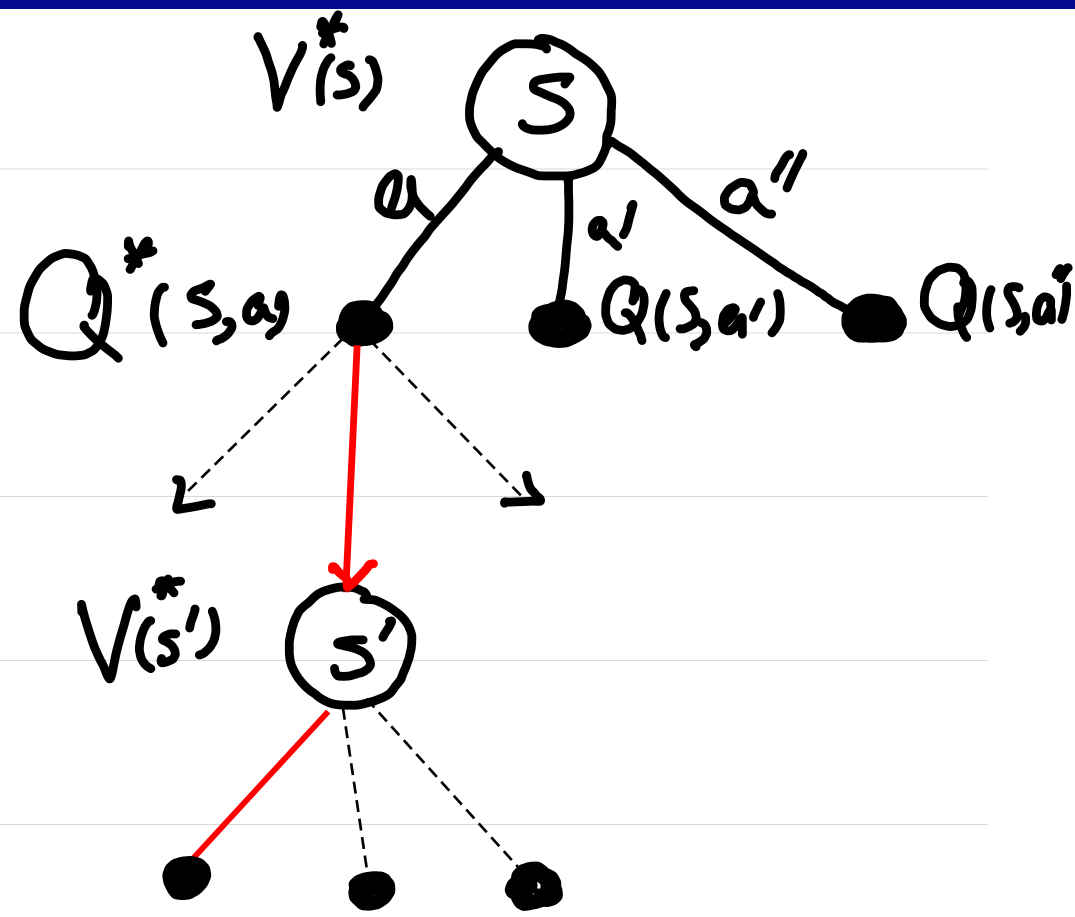
- Recursive definition of values:

$$V^*(s) = \max_a Q^*(s,a)$$

$$Q^*(s,a) = \sum_{s'} T(s,a,s')\left[R(s,a,s') + \gamma V^*(s')\right]$$

$$\Rightarrow V^*(s) = \max_{a \in A} \sum_{s' \in S} T(s,a,s')\left[R(s,a,s') + \gamma V^*(s')\right]$$

Bellman equation



- But how do we solve these equations?

Value Iteration Algorithm

- Value iteration:

  Initialize $V_0(s) = 0$ for all $s \in S$

  For $k=1$, until Convergence:

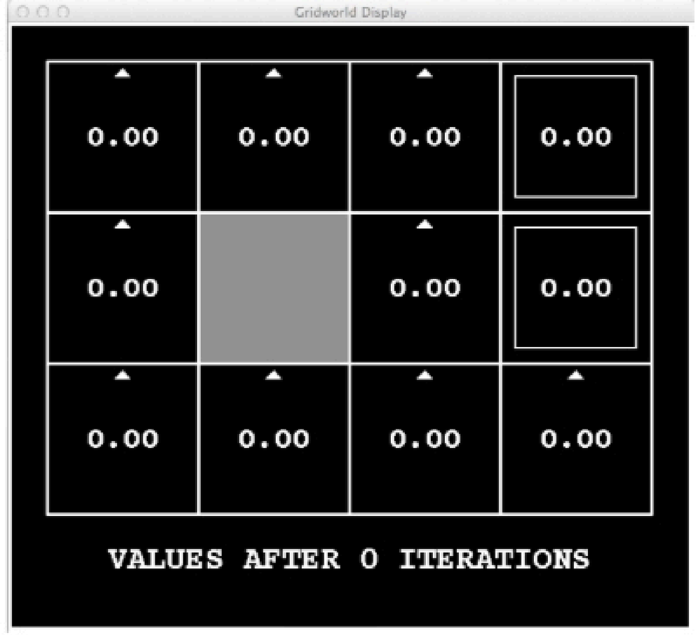
  For $s \in S$:

  $$V_{k+1}(s) \leftarrow \max_{a \in A} \sum_{s' \in S} T(s,a,s') \left[R(s,a,s') + \gamma V_k(s') \right]$$

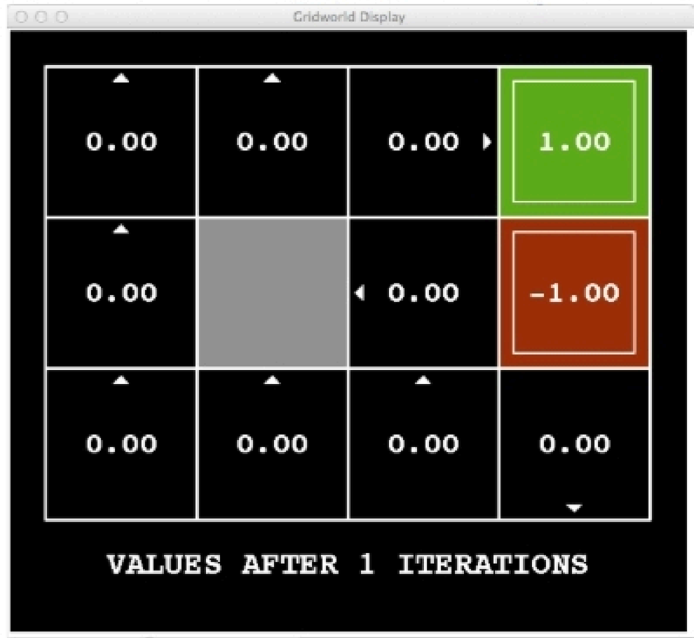
- Applies the Bellman Equation to update the value of state s based on the state values derived in the previous iteration
- Repeat until convergence, which yields V^*
- Complexity of each iteration: $O(|A| |S|^2)$

Example: Mars Rover in the 2D-Grid

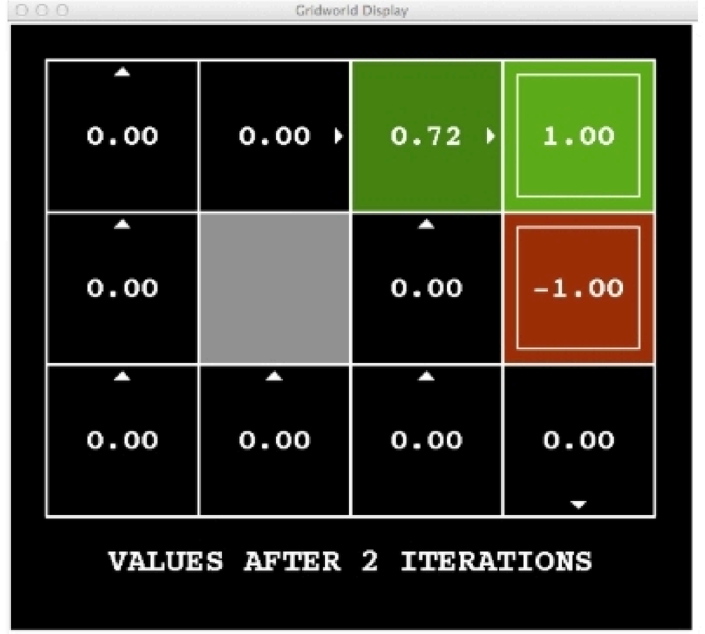
K=0



K=1

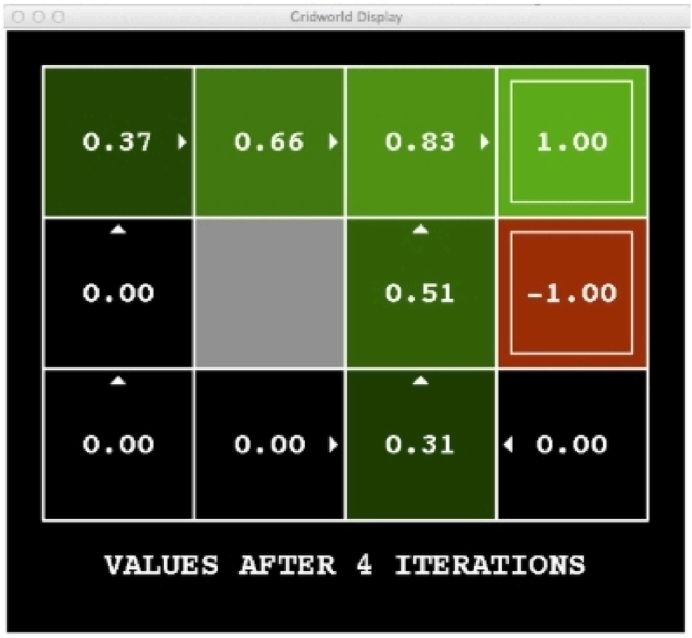


K=2

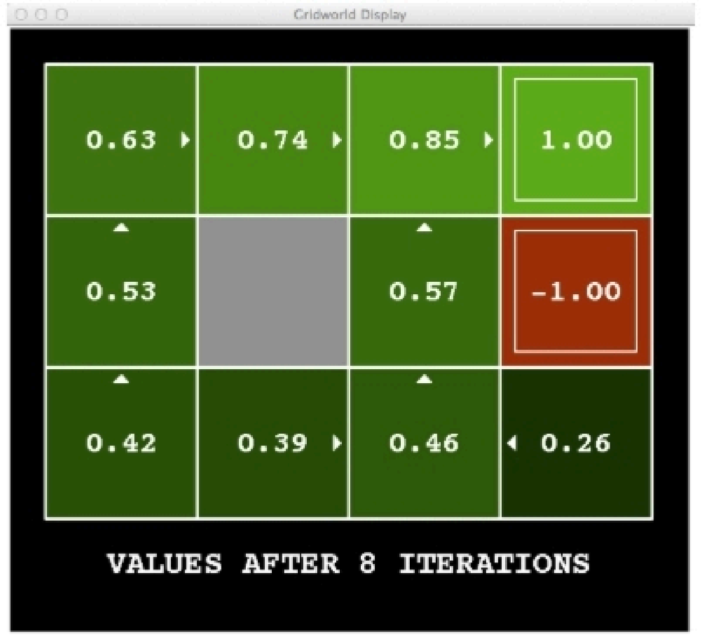


K=3

K=4

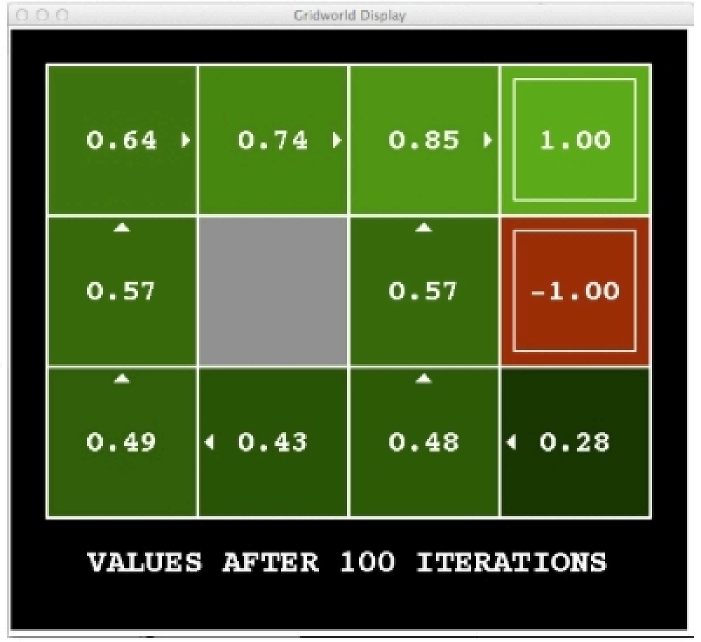


K=8



K=12

K=100



Homework: Run Value Iteration on "Erfan the Undergrad" Example

- After how many iterations of "Erfan the Undergrad" you will reach the optimal values?

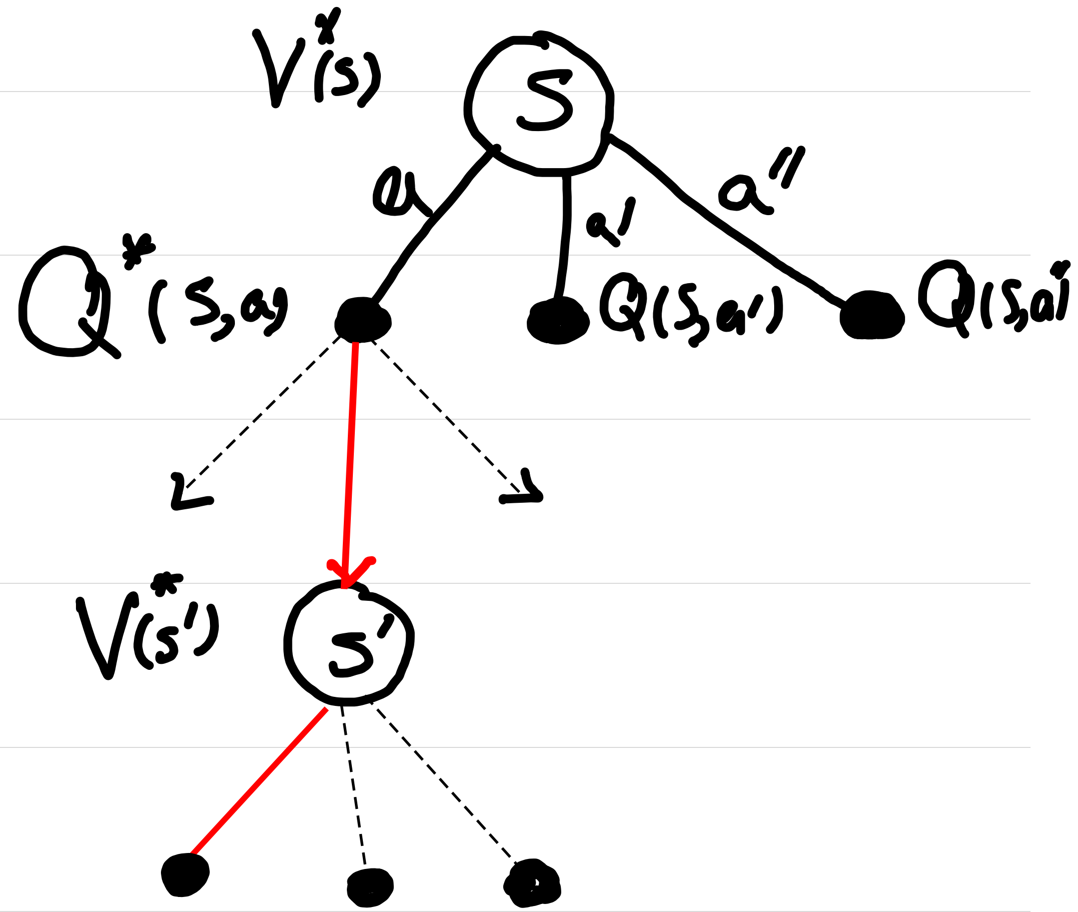
Bellman Equation for Q-Values

- We saw the Bellman Equation for optimal $V^*(s)$

$$V^*(s) = \max_{a \in A} \sum_{s' \in S} T(s,a,s') \left[R(s,a,s') + \gamma V^*(s') \right]$$

- Can you write down Bellman Equation for $Q^*(s,a)$?

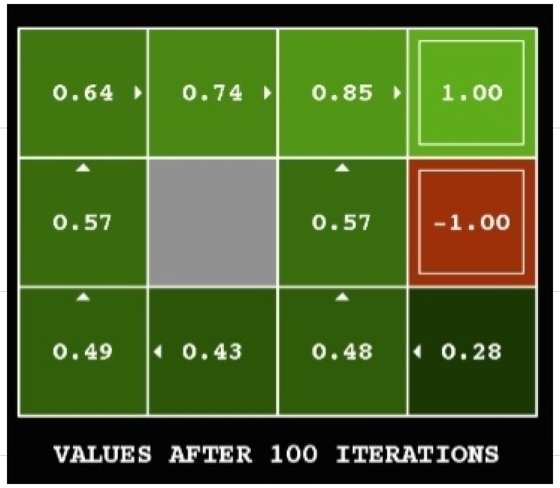
$$Q^*(s,a) = \sum_{s' \in S} T(s,a,s') \left[R(s,a,s') + \gamma V^*(s') \right]$$



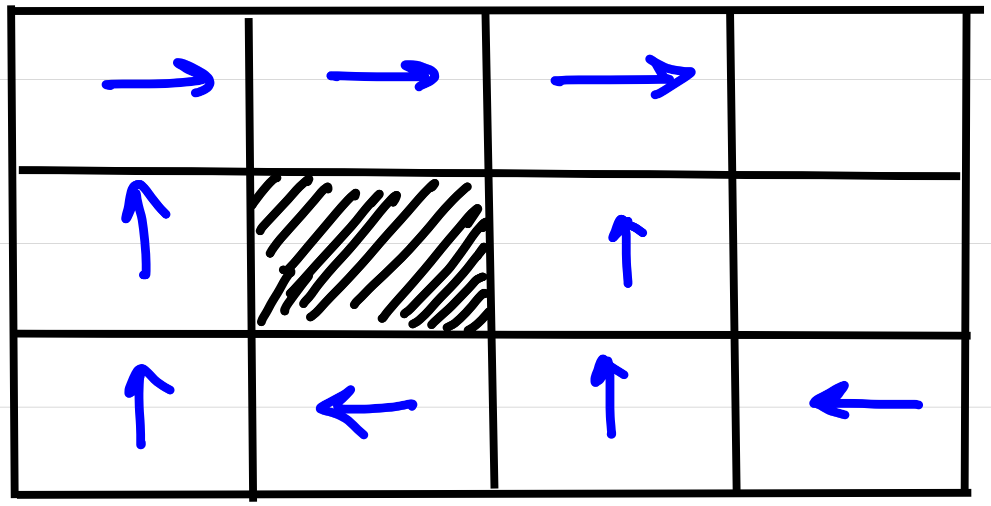
- Leads to Q-value iteration we will see later

Policy Extraction

- So far, with value iteration, we could find the optimal values



- How can we use the optimal values to find the optimal policy?

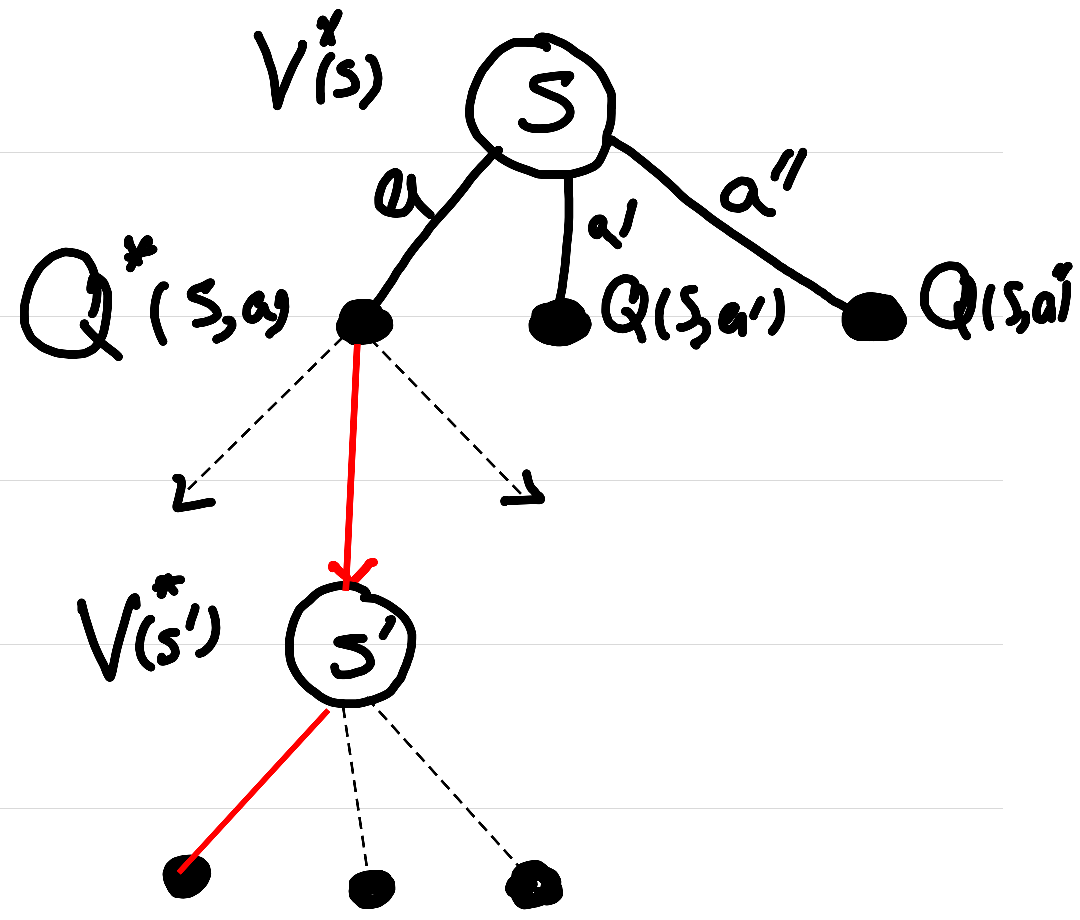


Computing Actions from Values

- The action a that maximizes $Q^*(s,a)$ is the optimal action

$$\pi^*(s) = \arg\max_{a} Q^*(s,a)$$

$$= \arg\max_{a\in} \sum_{s'} T(s,a,s')\left[R(s,a,s') + \gamma V^*(s')\right]$$



- This process is called policy extraction.
- If we had access to q-values, we could simply find $\pi^*(s)$ as
- Important lesson: Actions are easier to select from q-values than values!