The Basic RL Setup

- Couldn’t be simpler:

  — you are at a state

  — you take an action

  — you observe a reward

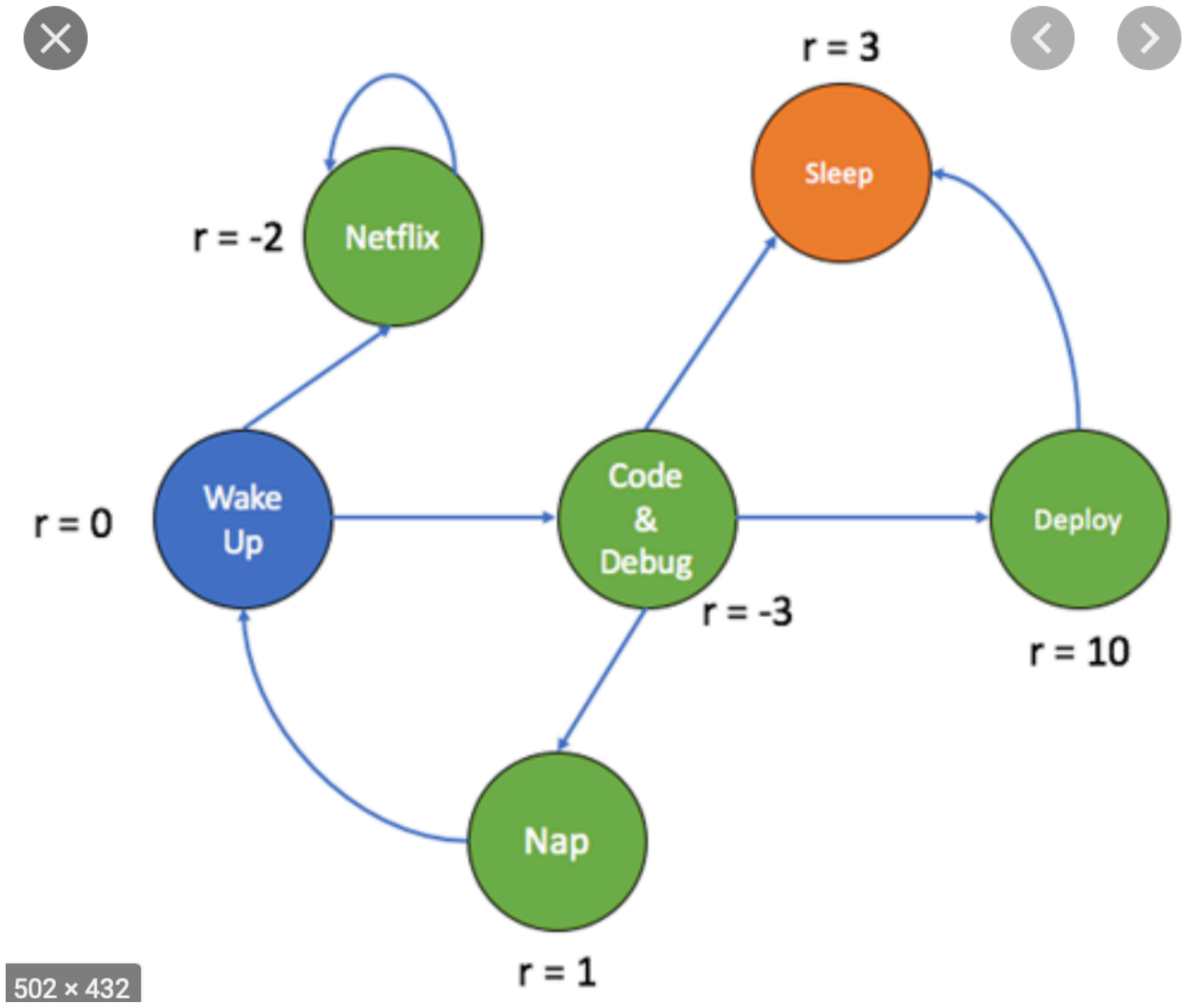
  — you transition to a new state
- Key assumption: given the current state, current action, your next state is independent of the past history.
- The big problem: what are the best actions to take?
- Let’s introduce notation for all of this.

- Let us denote the state at time k by s_k.
- At time k, you take the action a_k.
- You then receive reward r_k.

  This reward could be random.

  It depends on your state and the action you take.
- You then transition to the next state, which is s_{k+1}.
- The next state s_{k+1} can be random.

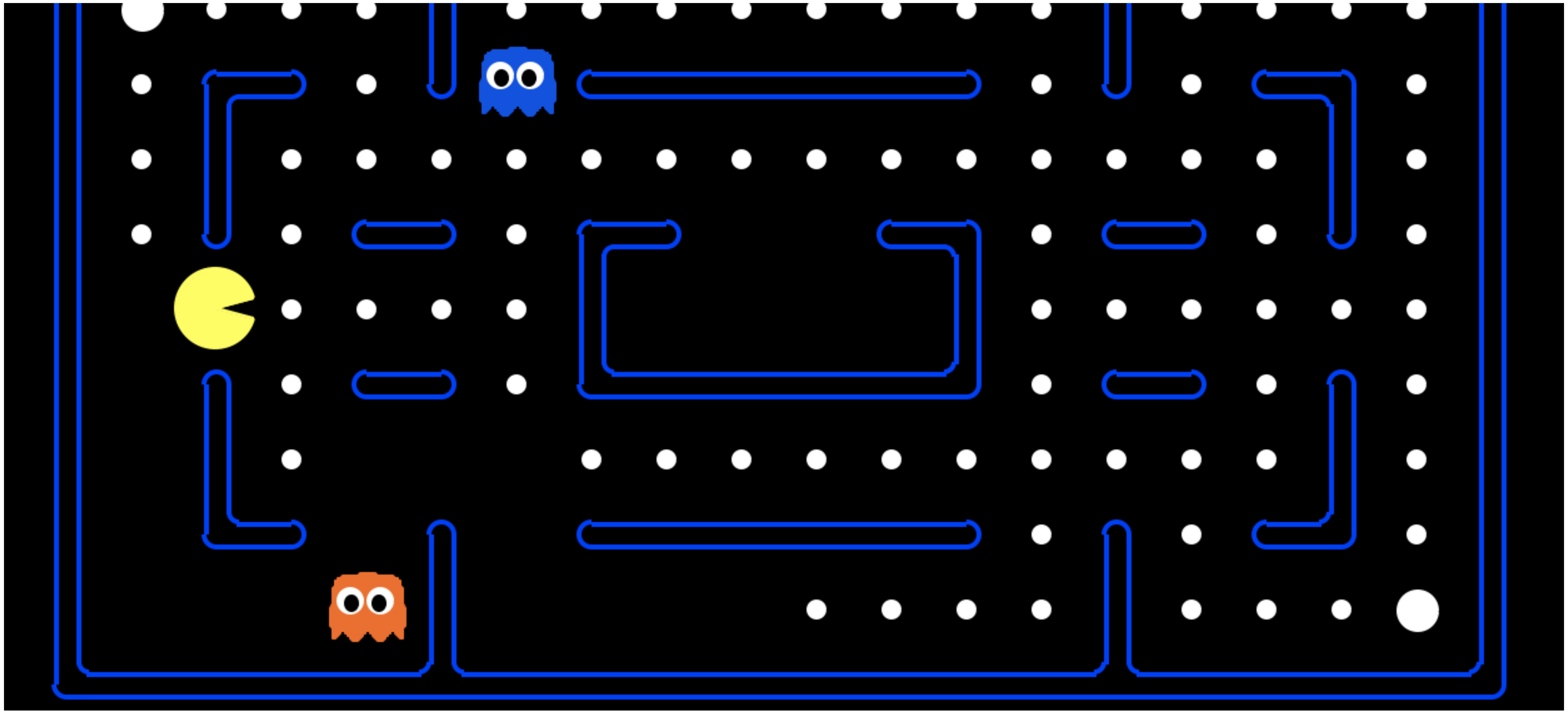
  It can depend on both s_k and a_k.
- **Key assumption:** conditional on s_k, a_k the state s_{k+1} is independent of $s_{k-1}, a_{k-1}, s_{k-2}, a_{k-2}, \dots$
- In particular: suppose you fix a policy (a rule by which you choose a_k as a function of s_k). Then s_k is a Markov chain.
- This setup is referred to as an MDP (Markov Decision Process).



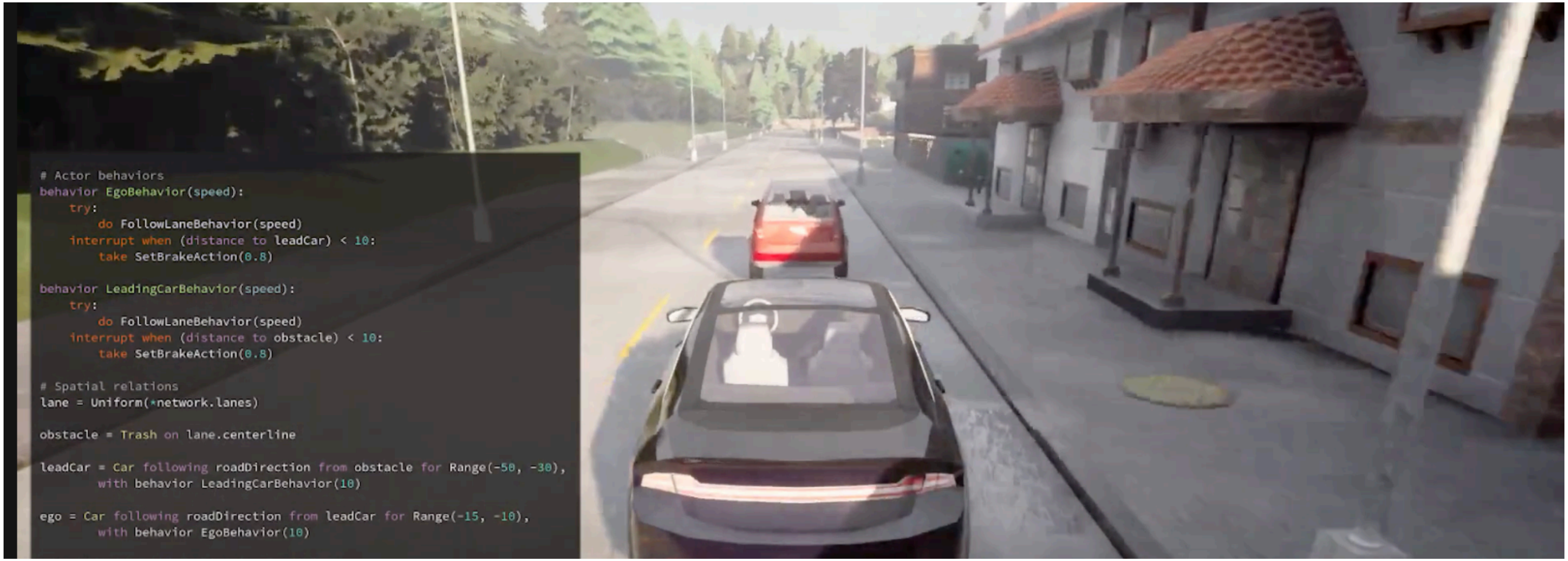
s_0 = nap, r_0 = 1
s_1 = wake up, r_1 = 0
s_3 = code and debug, r_3 = -3
s_4 = nap, r_4 = 1
s_5 = wake up, r_5 = 0
s_6 = netflix, r_6 = -2

- We will typically assume the state space is finite (as in the previous slide).
- Also, the number of actions at every stage is finite.
- Also, the reward can take values in a finite set.
- At some points, we'll talk about relaxing this set of assumptions.

  But for 99% of our discussions, we'll assume everything is finite.
- This ends up avoiding some thorny mathematical issues we want to avoid.



Can think of pacman, but with the ghosts moving randomly.
Can think of this as an MDP:
State = the entire board over the past TWO time steps
Actions: move left, right, up, down
Key assumption is satisfied: current state + action determine the distribution of the next outcome
Past history is irrelevant
Rewards = +10 for every food you collects, -10000000 if eaten by ghosts, -0.01 when moving through empty space



```
# Actor behaviors
behavior EgoBehavior(speed):
    try:
        do FollowLaneBehavior(speed)
    interrupt when (distance to leadCar) < 10:
        take SetBrakeAction(0.8)

behavior LeadingCarBehavior(speed):
    try:
        do FollowLaneBehavior(speed)
    interrupt when (distance to obstacle) < 10:
        take SetBrakeAction(0.8)

# Spatial relations
lane = Uniform(*network.lanes)

obstacle = Trash on lane.centerline

leadCar = Car following roadDirection from obstacle for Range(-50, -30),
        with behavior LeadingCarBehavior(10)

ego = Car following roadDirection from leadCar for Range(-15, -10),
        with behavior EgoBehavior(10)
```

Screenshot from CARLA, a popular simulator for self-driving cars

If state = the last few screenshots, then it makes sense to think of this as an MDP

Suppose you want to go from point A to point B.

Rewards: — +10,000 when you first arrive at point B

— -1 if you are not at point B

— -1000,000,000,000,000 if you collide with something

- Two problem variations: terminal and continuing.
- Terminal means the process terminates once it reaches a certain state.
- Continuing means the process goes on forever.
- We will mostly deal with continuing problems.
- But if the game goes on forever, what does it mean to maximize the reward?
- Consider an MDP with one state, and two actions. Action 1 gives you a reward of 1, action 2 gives you a reward of 2.
- Natural to prefer action 2, but if the game goes on forever then one might argue

  $$1 + 1 + \cdots = \infty = 2 + 2 + \cdots$$

- One solution: optimize the average reward

  $$r_{\text{ave}} = \lim_{t \to +\infty} \frac{r_0 + \cdots + r_{t-1}}{t}.$$

- Another solution: choose a "discount factor" $0 < \gamma < 1$ and optimize

  $$r_{\text{discounted}} = r_0 + \gamma r_1 + \gamma^2 r_2 + \gamma^3 r_3 + \cdots$$

- So you act as if rewards later in time are less valuable.
- Key idea: remember the formula for the sum of a geometric series

  $$1 + \gamma + \gamma^2 + \gamma^3 + \cdots = \frac{1}{1 - \gamma}$$

  when $0 < \gamma < 1$. So as long as rewards are bounded, discounting translates an infinite sum of rewards into a finite number.
- For us in this class, rewards are always bounded because we assumed rewards have to lie in some finite set.

- Let's be a little bit more formal.
- An MDP is defined by:

  — a set of states S.

  — a collection of sets A_k of actions A [to be taken in step k]

  — a distribution for each pair (s, a) with $s \in S, a \in A$ taking value in some set R
  [this is the reward]

  — a distribution for each pair (s, a) with $s \in S, a \in A$ taking value in S.
  [this is the next state].
- This is how your typical textbook defines an MDP.
- Sometimes, there's a particular state in which you have to start. Other times, you can start anywhere.

- A deterministic policy is a map from states to actions: $\pi : S \to A$.
- Interpretation: π tells you which action to take at which state.
- Let's use Π_{det} to denote the set of deterministic policies.
- Note: the number of deterministic policies in our context is finite.
- A randomized policy is the same, except instead of outputting an action, you output a distribution over the state of actions.
- Let's use Π_{rand} to denote the set of randomized policies.

- Given a policy π and state s, the “reward to go” is defined as

$$J_\pi(s) = E\left[\sum_{k=0}^{+\infty} \gamma^k r_k\right].$$

- Our goal: find the best reward-to-go by optimizing over all policies, i.e., compute $J^*(s)$ defined as

$$J^*(s) = \max_{\pi \in \Pi_{\text{det}}} J_\pi(s).$$

- Can also try to find

$$\max_{\pi \in \Pi_{\text{rand}}} J_\pi(s).$$

- Spoiler alert: these turn out to be the same (will discuss this later).
- Of course, we don’t just want the cost-to-go; we want the policy that achieves it!
- Be mindful: in some of the papers, you suffer costs rather than accumulate rewards.
  Then, you have to minimize the total cost suffered.
  Of course, this is the same as what we are discussing here (can simply set reward = negative of cost).

- Let's consider a simple example.
- Our MDP has two states: "Watch TV" and "Be outside."
- If you are in "Watch TV," you have two actions "Stay" and "Switch."
  Stay keeps you in "Watch TV" on the next step.
  "Switch" brings you to "Be outside."
- If you are outside, then you just remain outside.
  Formally, we can say that your next state will remain "Be outside" regardless of which of the two actions you take.
- When you are in "Watch TV," and you choose to stay, you get 1 reward.
  If you choose to switch, you get -1 reward.
- When you are in "Be outside," you get 2 reward on all transitions.
- Suppose discount factor is 1/2. What should you do in the "Watch TV" state?
- Option 1 is stay. Associated reward stream: 1 + (1/2) + (1/4) + … = 2
  Option 2 is switch. Associated reward stream: -1 + 2*(1/2) + 2*(1/4) + … = 1.
  Stay is better!
- What if discount factor is 0.9?
- Option 1: 1 + 0.9 + 0.9^2 + … = 1/(1-0.9) = 10
  Option 2: -1 + 2*0.9 + 2*0.9^2 + … = -1 + 2*0.9*1/(1-0.9)=17. So switch is better.

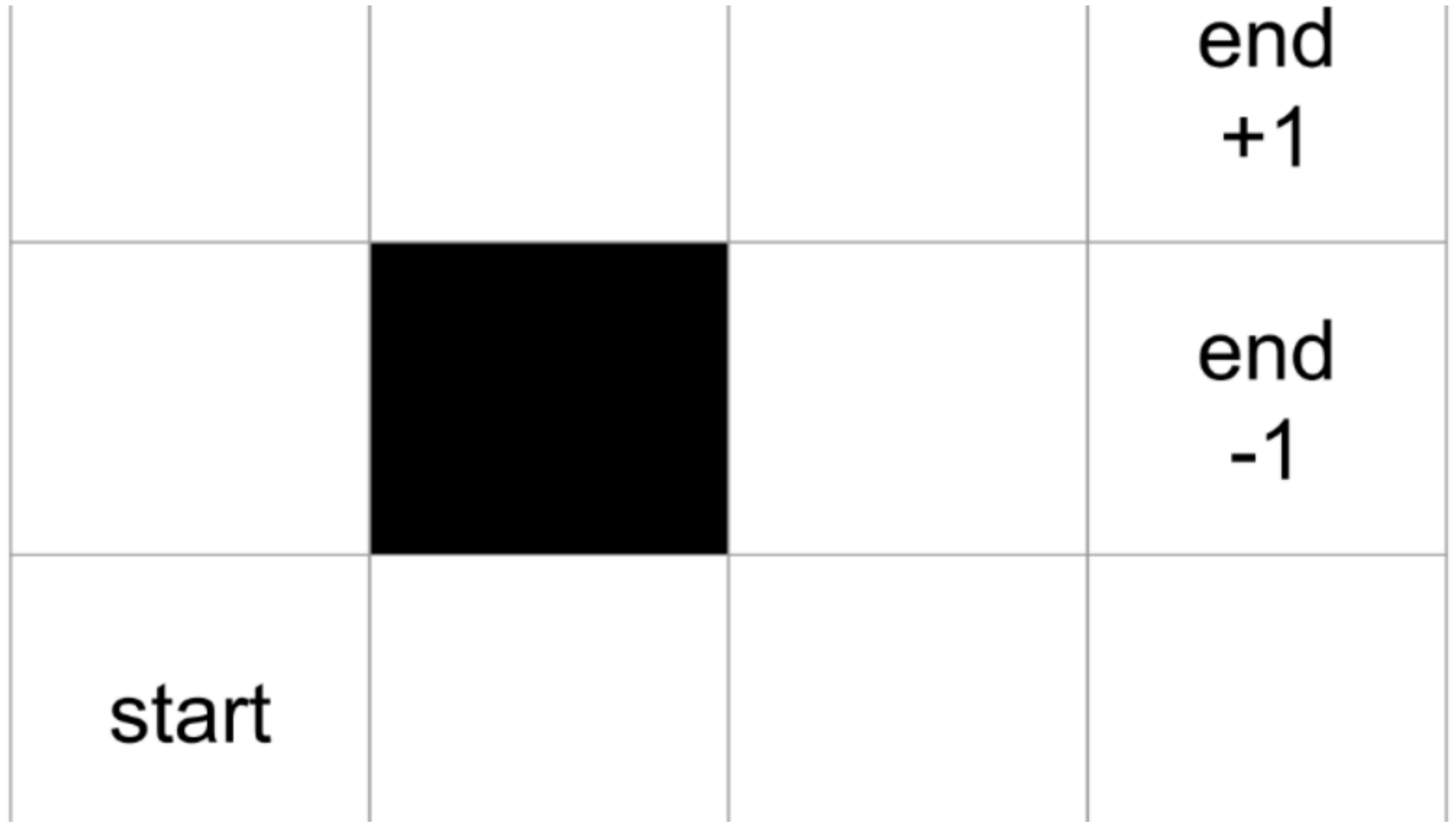
- So we are trying to maximize the reward-to-go form state s:

$$J_\pi(s) = E\left[\sum_{k=0}^{+\infty} \gamma^k r_k\right].$$

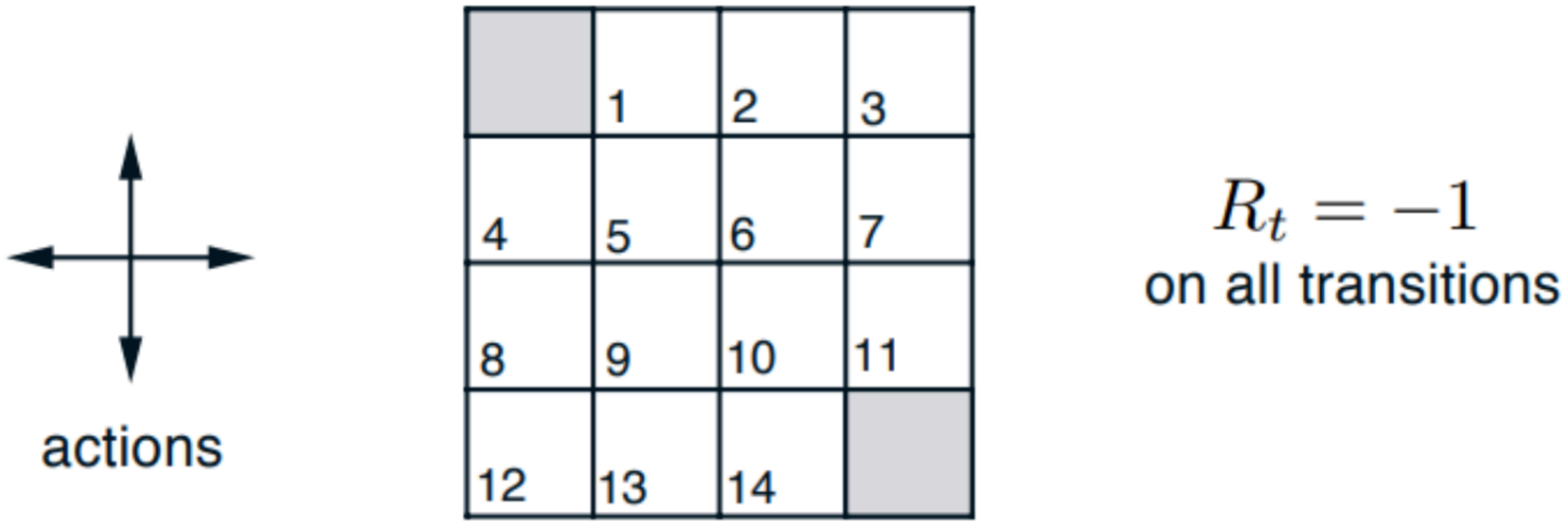
$$J^*(s) = \max_{\pi \in \Pi_{\text{det}}} J_\pi(s).$$

- Consider the optimal policy....the way, we have defined it, it could depend on s.
- Recall from just a few slides ago: a policy π maps **every** state to an action.
- Could the best policy be different for different states?
- As in: the policy π^1 that maximizes the reward-to-go from state s_1 is different than the policy π^2 that maximizes the reward to go from state s_2?
- Spoiler alert: this will not happen
- But: at this stage, we cannot quite talk about "**the** optimal policy."

- An episodic MDP is one that has a set of "terminal states." The game ends when the agent enters a terminal state.



An instance of Gridworld
Typically, you will set per step-rewards to a small negative to incentivize the agent to finish the game
If the rewards on intermediate steps are set to zero, the agent may prefer to wander around when it can't figure out how to end at the +1
You will have to code this in one homework!



Another popular Gridworld setup
You basically need to find the shortest path to a terminal state
Sometimes there is “noise”: with a small probability (e.g., 20%)
a random action is executed regardless of what you chose
The noise makes the learning more difficult
You will have to code for this on a future homework (without noise though)

- An episodic MDP can be viewed as an instance of a continuing MDP. How?
- ...just add a self-loop at the terminal states, and set all rewards to zero.
- In the episodic case, the standard thing to do is to still discount future rewards by a factor of $\gamma \in (0,1)$.
- However, it is also possible to have an objective which is just the sum of the rewards in an episode.
  This often work but can easily break down if there is a way to get $+\infty$ reward.

- OK, so we now have a problem statement!
  Next: solving the problem.
  This will take the remainder of the class.
- Before jumping to a discussion of how the problem might be solved, it helps to simplify it.
- Simplification: the game goes on for a finite number of steps.
- Let's start with the simplest possible scenario: the game goes on for one step.

- So here is the problem: you start in state s.

  You can take any action a in the set A.

  When you take action a, you obtain a random reward which has **expectation** $r(s, a)$.

  Which action should you take?

- Let's assume $r(s, a)$ is known to you.
- Answer is obvious. You take the action that maximizes the reward.

  $$a^*(s) \in \arg\max_{a \in A} r(s, a)$$

- OK, now let's add a "**terminal reward**" to the problem statement.
- New rules: you start in state s.

  You can take any action a in the set A.

  When you take action a, you obtain a random reward r_0 which has expectation $r(s, a)$.

  You transition to state s'. The distribution of s' depends on s and a.

  You get the reward $J(s')$ (a deterministic function of the new state).

  Then the game ends.
- The total reward you get is actually $r_0 + \gamma J(s')$ because you discount future rewards by a factor of γ.
- We assume that $r(s, a)$ is known for all state-action pairs, and likewise $J(s')$ is known for all states s'.
- Let $P(s' | s, a)$ be the probability of transitioning to s' conditional on taking action a in state s. So you take the action that maximizes the expected reward:

$$a^*(s) \in \arg\max_{a \in A} \left[r(s, a) + \gamma \sum_{s' \in S} P(s' | s, a) J(s') \right]$$

- OK, now let's do two steps!
- Rules of the game are:

— you start at state s.

— you take action $a \in A$.

— you obtain a random reward r_0 with expectation $r(s, a)$.

— you transition to a random state s'.

[The distribution of s' depends on s, a]

— you take action a'.

— you receive reward r_1 with expectation $r(s', a')$

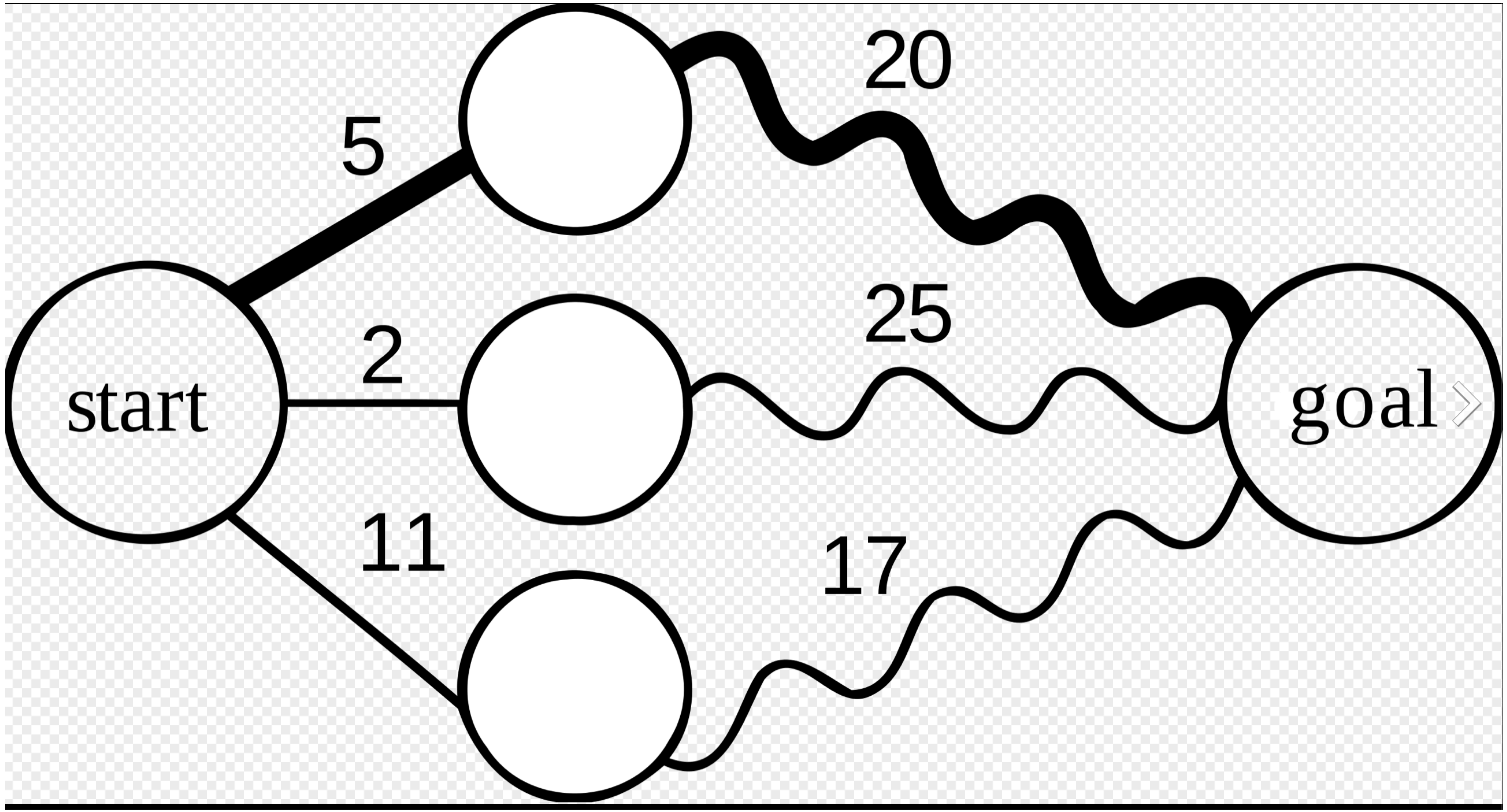
[The distribution of $r_1$depends on s', a', but conditional on these it is independent of s, a]

— you transition to a random state s'' [which likewise has the Markov property]

— you obtain a **terminal reward** $J(s'')$.

— the game ends.

- The total reward you obtain is $r_0 + \gamma r_1 + \gamma^2 J(s'')$.



An instance of the two step game.
Numbers of transitions represent costs.
Terminal cost is zero.

- Key idea: total reward you obtain is $r_0 + \gamma r_1 + \gamma^2 J(s'')$.

  Write this as

  $r_0 + \gamma(r_1 + \gamma J(s''))$.
- Your reward equals reward from first transition + reward from playing the one-step after you transition once.
- AFTER you transition, you are just playing the one-step game.
- So AFTER you transition, we already know how to choose your optimal policy.
- So suppose you have chosen action a, transitioned to some s'.

  Let $a_1^*(s')$ be the optimal action to take now. What is it?
- $a_1^*(s') \in \arg\max_{a \in A} r(s', a) + \gamma \sum_{s''} P(s'' \mid s', a) J(s'')$.
- ...and the forward-looking reward you expect to obtain is just $\max_{a \in A} r(s', a) + \gamma \sum_{s''} P(s'' \mid s', a) J(s'')$.

  Let's call that $J_1^*(s')$, the "reward-to-go."

- OK, so suppose that, after transitioning, you take the action on the previous slide.
- As we discussed the cost you suffer is

  $r_0 + \gamma(r_1 + \gamma J(s''))$.
- Don't forget that r_0 is a function of s, a and r_1 is a function of the next state-action pair s', a'.
- The expected reward you expect to obtain after taking action a in state s is

  $r(s,a) + \gamma \sum_{s'} P(s' \mid s, a) J_1^*(s')$
- ...OK, but this exactly like the one-step problem with $J_1^*(s')$ being the terminal cost!
- So you take action

$$a_0^*(s') \in \arg\max_a r(s,a) + \gamma \sum_{s'} P(s' \mid s, a) J_1^*(s')$$

  and expect to obtain reward $\max_a r(s,a) + \gamma \sum_{s'} P(s' \mid s, a) J_1^*(s')$

- Our solution:

$$a_1^*(s') \in \arg\max_{a \in A} r(s', a) + \gamma \sum_{s''} P(s'' | s', a) J(s'')$$

$$J_1^*(s') = \max_{a \in A} r(s', a) + \gamma \sum_{s''} P(s'' | s', a) J(s'')$$

$$a_0^*(s) \in \arg\max_{a} r(s, a) + \gamma \sum_{s'} P(s' | s, a) J_1^*(s')$$

$$J_0^*(s) = \max_{a} r(s, a) + \gamma \sum_{s'} P(s' | s, a) J_1^*(s')$$

  will usually refer to J_0^* as just J^*.

- Usually written as:

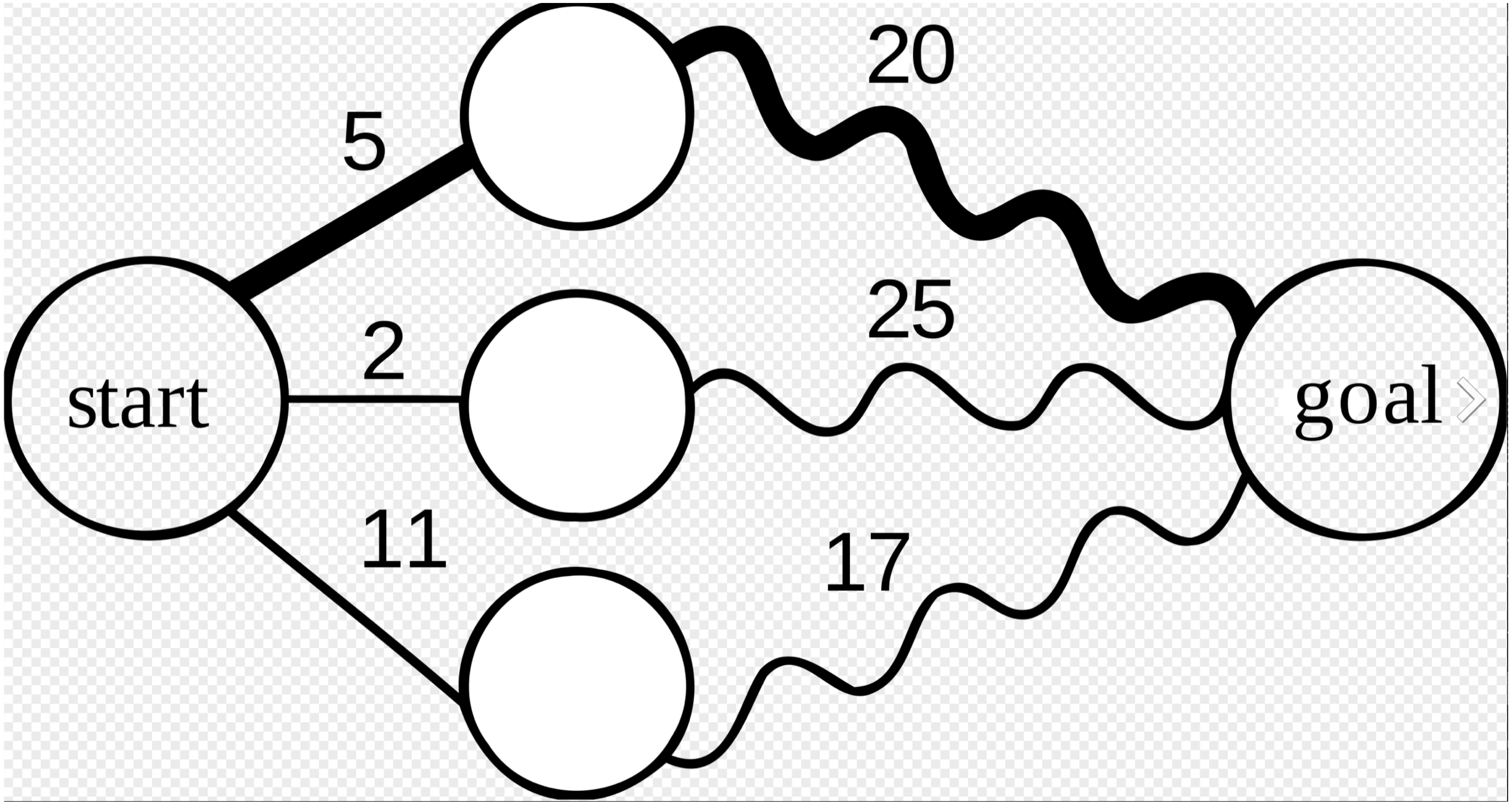
$$a_1^*(s) \in \arg\max_{a \in A} r(s, a) + \gamma \sum_{s'} P(s' | s, a) J(s')$$

$$J_1^*(s) = \max_{a \in A} r(s, a) + \gamma \sum_{s'} P(s' | s, a) J(s')$$

$$a_0^*(s) \in \arg\max_{a} r(s, a) + \gamma \sum_{s'} P(s' | s, a) J_1^*(s')$$

$$J_0^*(s) = \max_{a} r(s, a) + \gamma \sum_{s'} P(s' | s, a) J_1^*(s')$$

- Throughout textbook & papers on RL, there is a little bit of notation abuse between using s' as the next variable, and as something we sum over. You should make sure you are comfortable with this.



Let's label the middle states 1,2,3. Suppose gamma = 1/2.
J_1(1) = 20, J_1(2) = 25, J_1(3) = 17.
To choose action at first step, must choose the biggest among
5 + (1/2)*20, 2 + (1/2)*25, 11+(1/2)*17,
so action 3 is the best one

- Let's now do the most general case.
- Rules of the game are:

  — you start at state s_0.

  — you take action $a_0 \in A$.

  — you obtain a random reward r_0 with expectation $r(s, a)$.

  — you transition to a random state s_1 which is a function of s_0, a_0.

  — After step k, you are in state s_k.

  — You take action a_k, obtain reward r_k, transition to s_{k+1}.

  — After K steps, you are in state s_K.

  You obtain the terminal reward $J(s_K)$ and the game ends.
- This is called a K-stage **dynamic programming** problem.
- The total reward you obtain is $r_0 + \gamma r_1 + \gamma^2 r_2 + \cdots + \gamma^{K-1} r_{K-1} + \gamma^K J(s_K)$.

- Main idea: work backwards!
- Suppose $K-1$ steps have passed and we are now in state s. What should be do?
- Take action

  $$a^*_{K-1}(s) = \arg\max_a r(s,a) + \gamma \sum_{s'} P(s' \mid s,a) J(s').$$

  and note that, in expectation, the reward-to-go is

  $$J^*_{K-1}(s) = \max_a r(s,a) + \gamma \sum_{s'} P(s' \mid s,a) J(s')$$

- OK, so we now know what we will do after $K-1$ steps.

  What about after $K-2$ steps?
- Total cost is $r_0 + \gamma r_1 + \gamma^2 r_2 + \cdots + \gamma^{K-1} r_{K-1} + \gamma^K J(s_K)$ which can be written as $r_0 + \cdots + \gamma^{K-3} r_{K-3} + \gamma^{K-2}(r_{K-2} + \gamma(r_{K-1} + \gamma J(s_K)))$
- You can't do anything about all the cost you've suffered in the past. But you can look towards the future.
- So you take action

  $$a^*_{K-2}(s) = \arg\max_a r(s,a) + \gamma \sum_{s'} P(s' \mid s,a) J^*_{K-1}(s')$$

  and define the cost to go is

  $$J^*_{K-2}(s) = \max_a r(s,a) + \gamma \sum_{s'} P(s' \mid s,a) J^*_{K-1}(s')$$

- More generally, consider what happens after k steps. Assume that you know what actions to take starting at step $k+1$.
  In particular, $J^*_{k+1}(s) = E\left[r_{k+1} + \gamma r_{k+2} + \cdots + \gamma^{K-k-1} J(s_K)\right]$ is the reward-to-go under the optimal set of choices starting from time $k+1$.
- Total cost is $r_0 + \gamma r_1 + \gamma^2 r_2 + \cdots + \gamma^{K-1} r_{K-1} + \gamma^K J(s_K)$ which can be written as $r_0 + \cdots + \gamma^{k-1} r_{k-1} + \gamma^k (r_k + \gamma(r_{k+1} + \gamma r_{k+2} + \cdots \gamma^{K-k-1} J(s_K)))$
- So you take action
  $$a^*_k(s) = \arg\max_a r(s,a) + \gamma \sum_{s'} P(s' \mid s,a) J^*_{k+1}(s')$$
  and define the cost to go is
  $$J^*_k(s) = \max_a r(s,a) + \gamma \sum_{s'} P(s' \mid s,a) J^*_{k+1}(s')$$

- We have our solution:

$$J^*_{K-1}(s) = \max_a r(s,a) + \gamma \sum_{s'} P(s' \mid s,a) J(s')$$

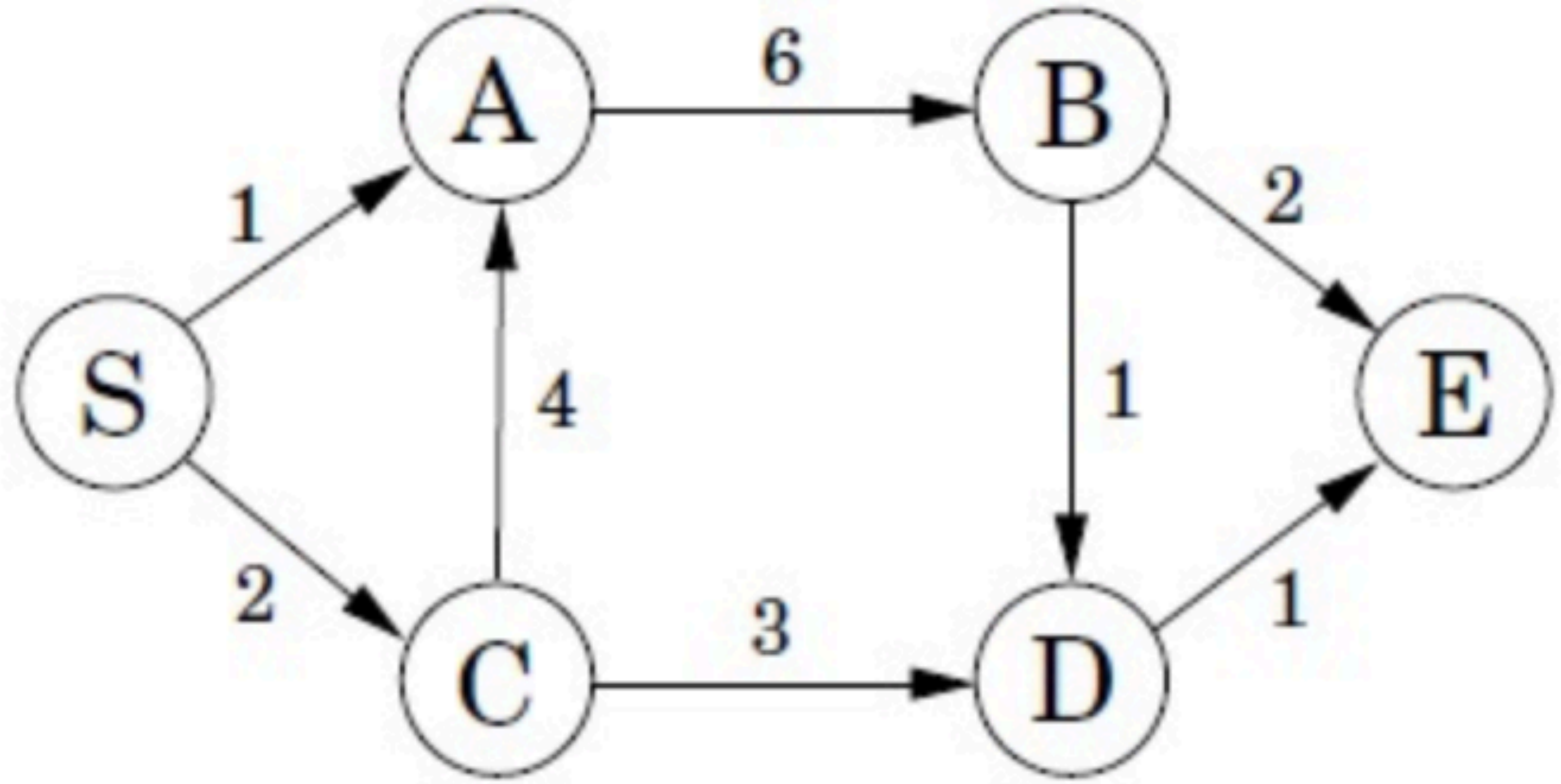
$$a^*_{K-1}(s) = \arg\max_a r(s,a) + \gamma \sum_{s'} P(s' \mid s,a) J(s')$$

  and then for $k = K-2, \ldots, 1$:

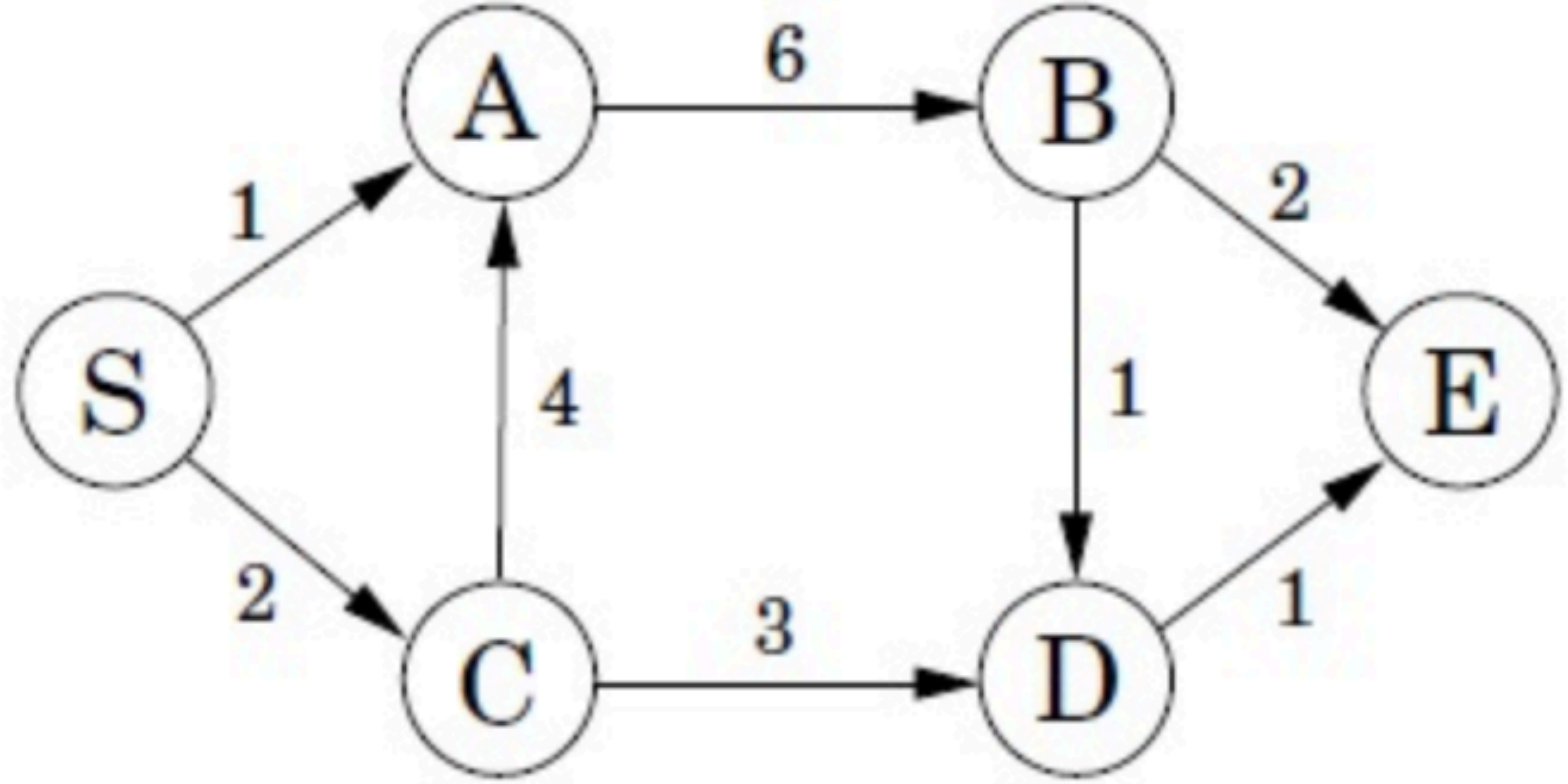
$$J^*_k(s) = \max_a r(s,a) + \gamma \sum_{s'} P(s' \mid s,a) J^*_{k+1}(s')$$

$$a^*_k(s) = \arg\max_a r(s,a) + \gamma \sum_{s'} P(s' \mid s,a) J^*_{k+1}(s')$$

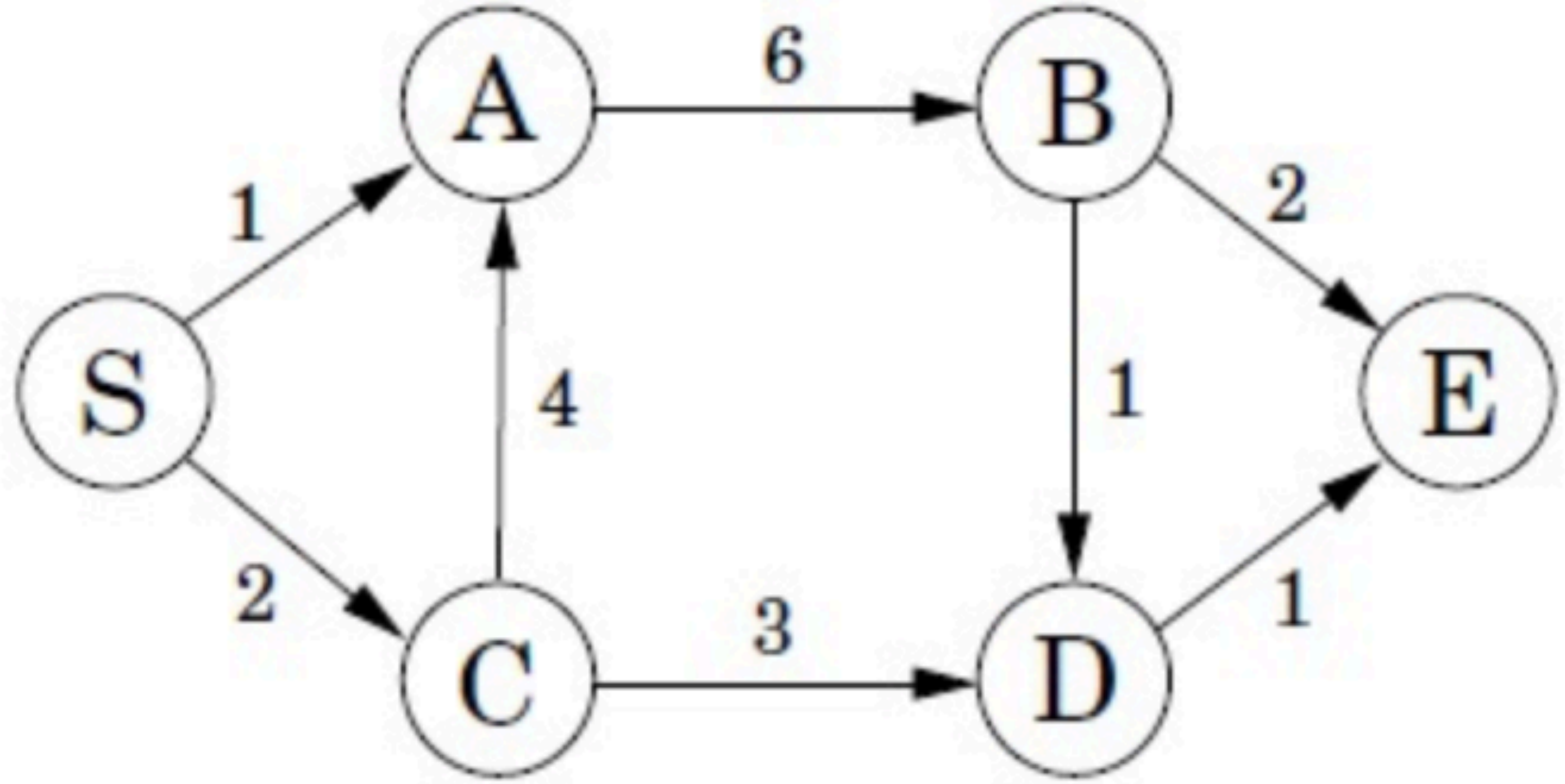
- For obvious reasons, this is called “backwards induction.”



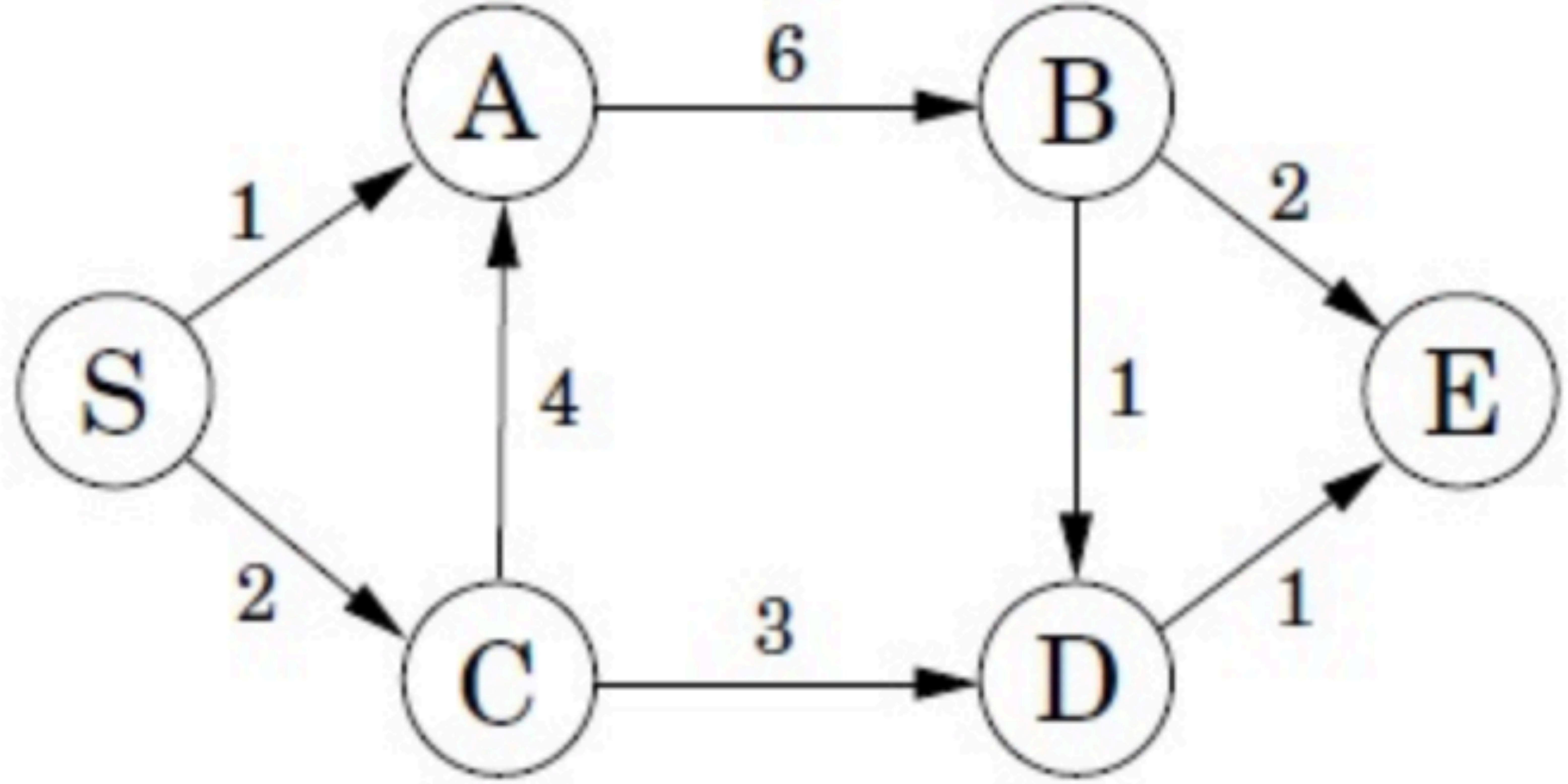
Question: what is the shortest path from S to E of length at most 5?
Here the number on the edge is the cost of traversing that edge.
We set the reward for traversing the edge to be NEGATIVE of that
We also assume there is a self-loop at E of cost ZERO
We will say there is a terminal cost of INFINITY for any path not ending at E
Discount factor equals ONE



J4(E) = 0, J4(B) = -2, J4(D) = -1 and J4(S) = J4(A)=J4(C)=INFINITY



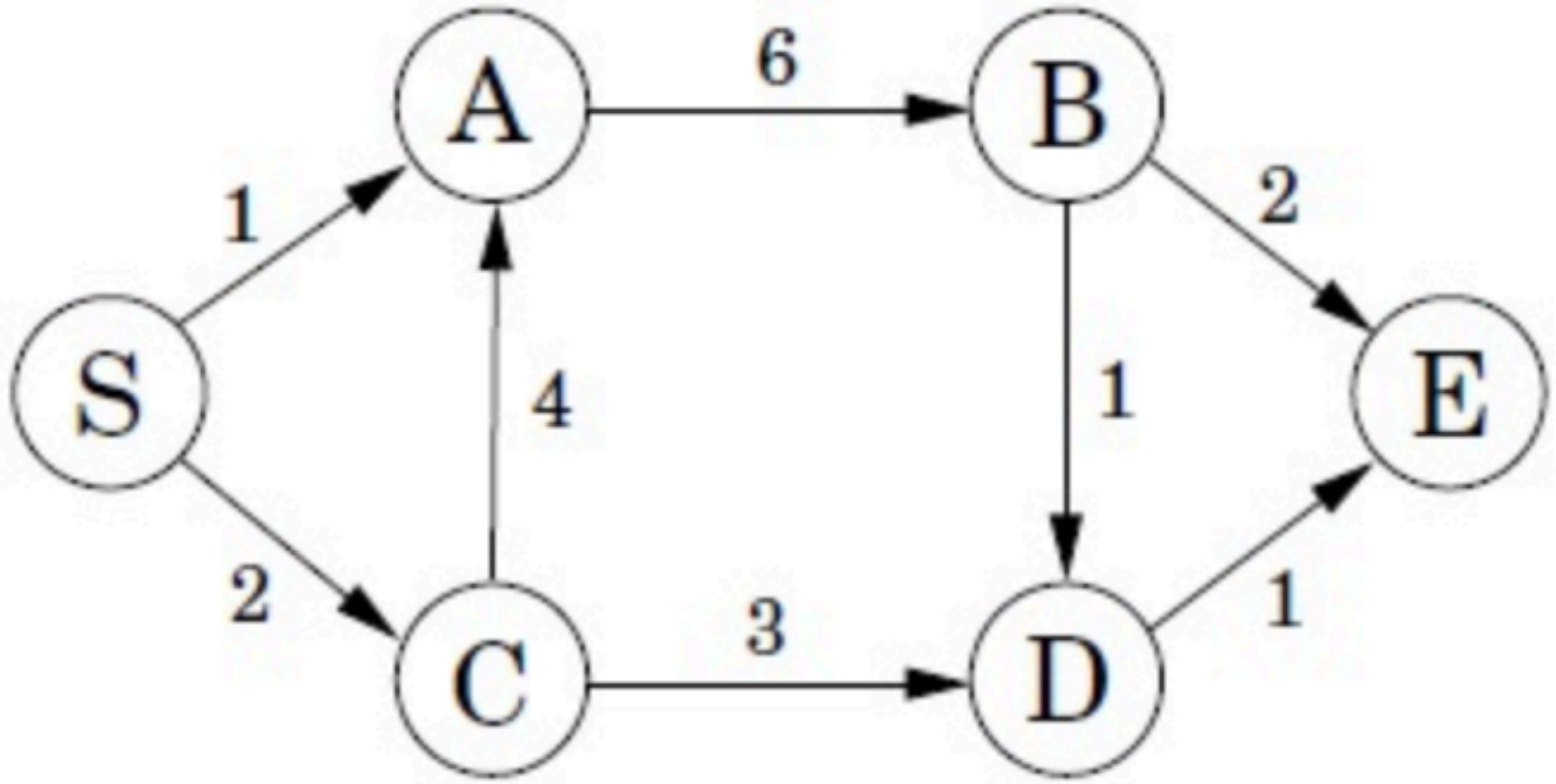
J4(E) = 0, J4(B) = -2, J4(D) = -1 and J4(S) = J4(A)=J4(C)=INFINITY
J3(E)=0, J3(B)=-2,J3(D)=-1, J3(C)=-4, J3(A)=-8 and J3(A)=INFINITY



J4(E) = 0, J4(B) = -2, J4(D) = -1 and J4(S) = J4(A)=J4(C)=INFINITY

J3(E)=0, J3(B)=-2,J3(D)=-1, J3(C)=-4, J3(A)=-8 and J3(S)=INFINITY

J2(E)=0,J2(B)=-2,J2(D)=-1, J2(C)=-4, J2(A)=-8, J2(S)=-6



J4(E) = 0, J4(B) = -2, J4(D) = -1 and J4(S) = J4(A)=J4(C)=INFINITY

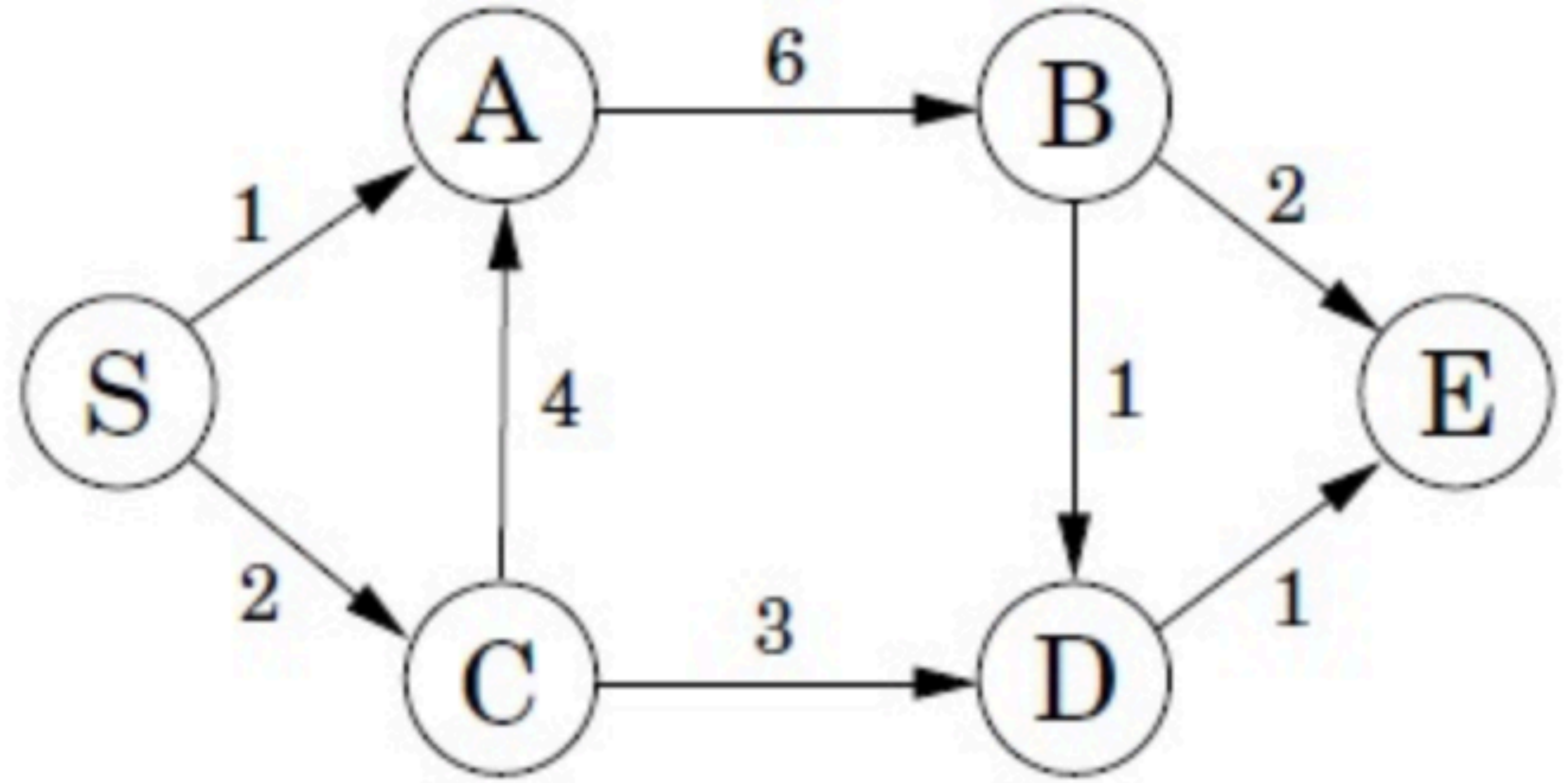
J3(E)=0, J3(B)=-2,J3(D)=-1, J3(C)=-4, J3(A)=-8 and J3(S)=INFINITY

J2(E)=0,J2(B)=-2,J2(D)=-1, J2(C)=-4, J2(A)=-8, J2(S)=-6

J1(E)=0,J1(B)=-2,J1(D)=-1, J1(C)=-4, J1(A)=-8, J1(S)=-6

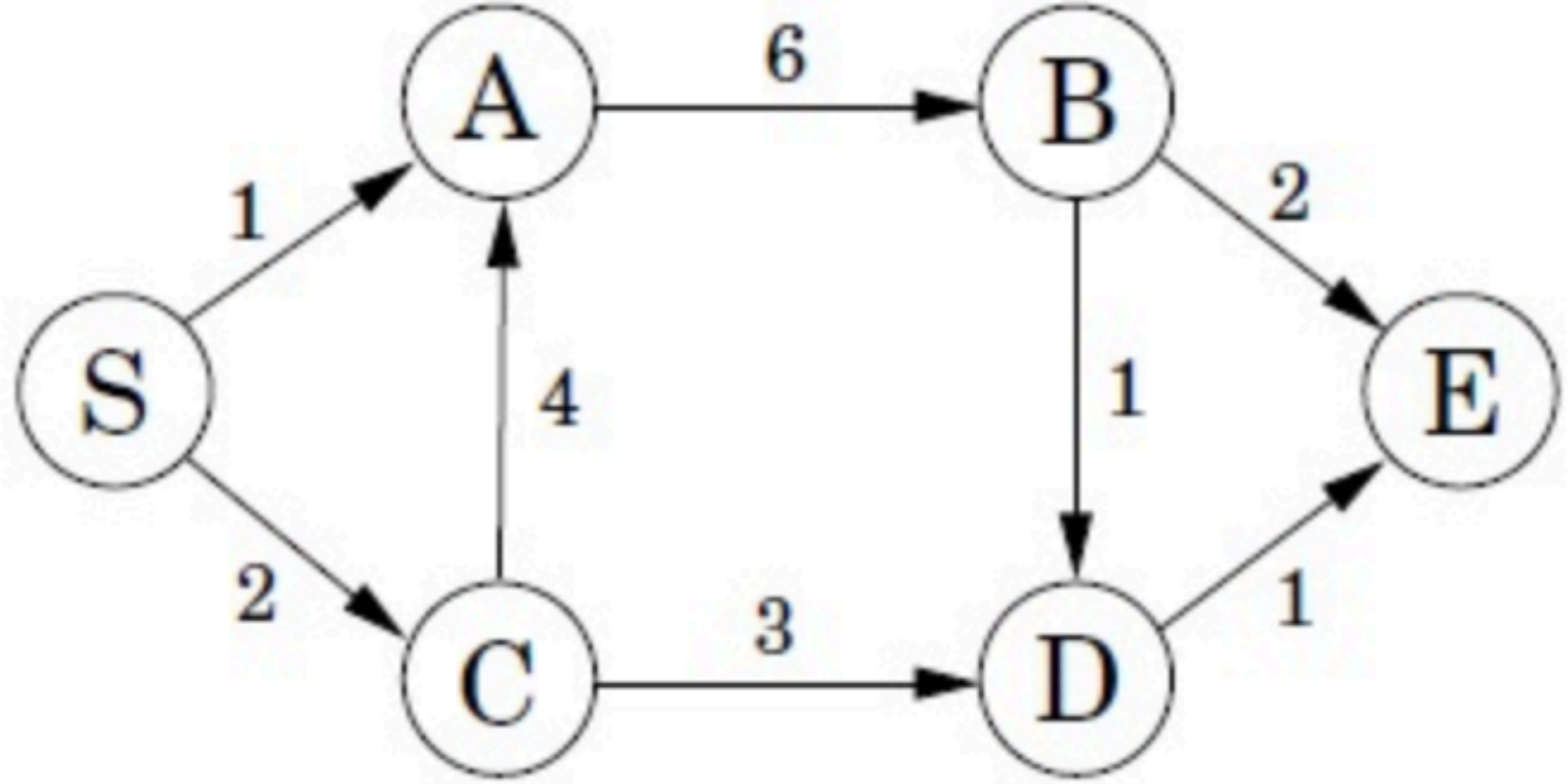
J0(E)=0,J0(B)=-2,J0(D)=-1, J0(C)=-4, J0(A)=-8, J0(S)=-6

IMPLICITLY, this finds the path S—> C —> D — > E



Now let's redo everything with discount factor equal 1/4!

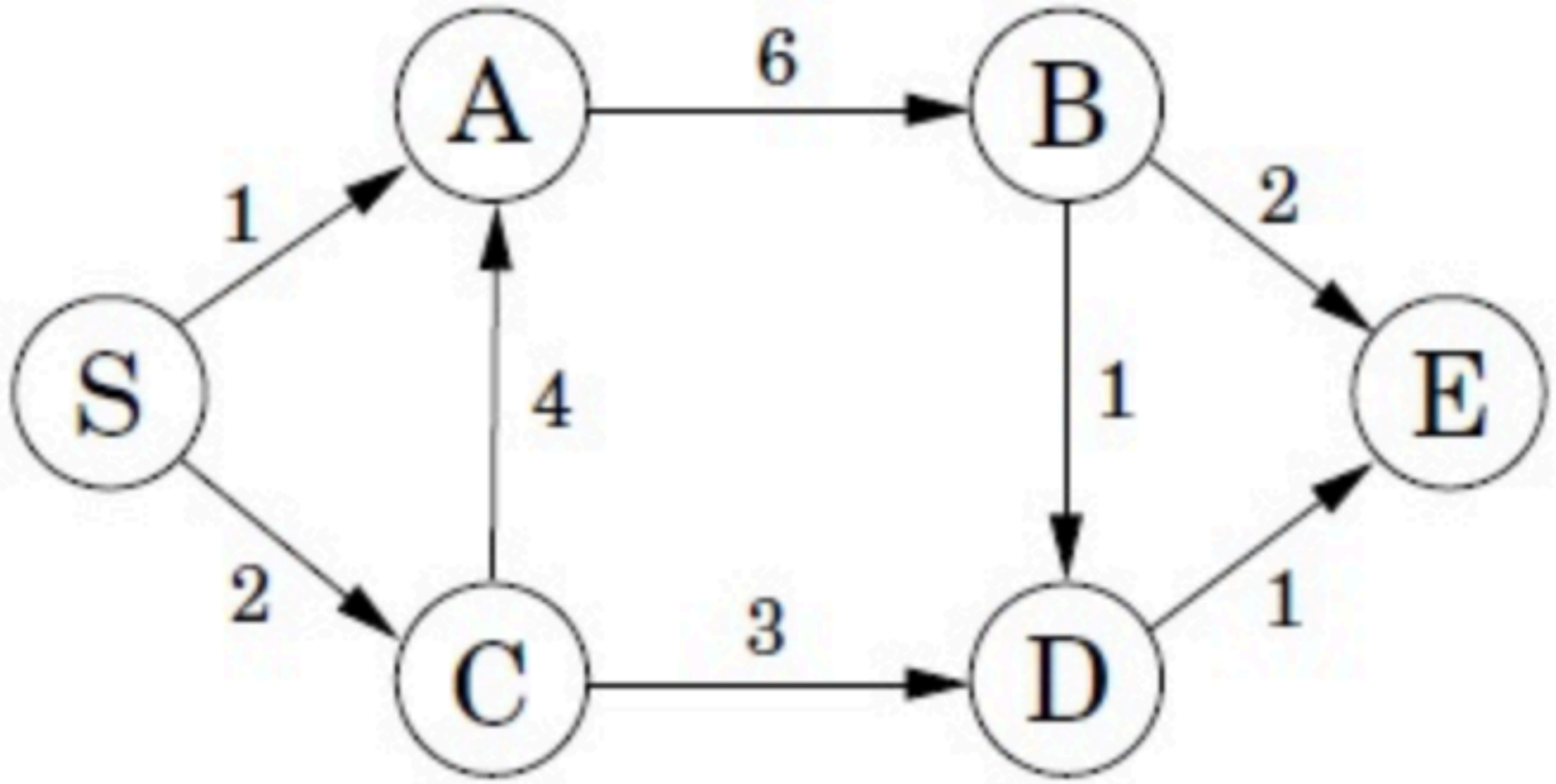
J4(E)=0, J4(B)=-2, J4(D)=-1.



Now let's redo everything with discount factor equal 1/4!

J4(E)=0, J4(B)=-2, J4(D)=-1.

J3(E)=0, J3(B)=-1.25, J3(D)=-1, J3(A)=-6.5, J3(C)=-3.25.

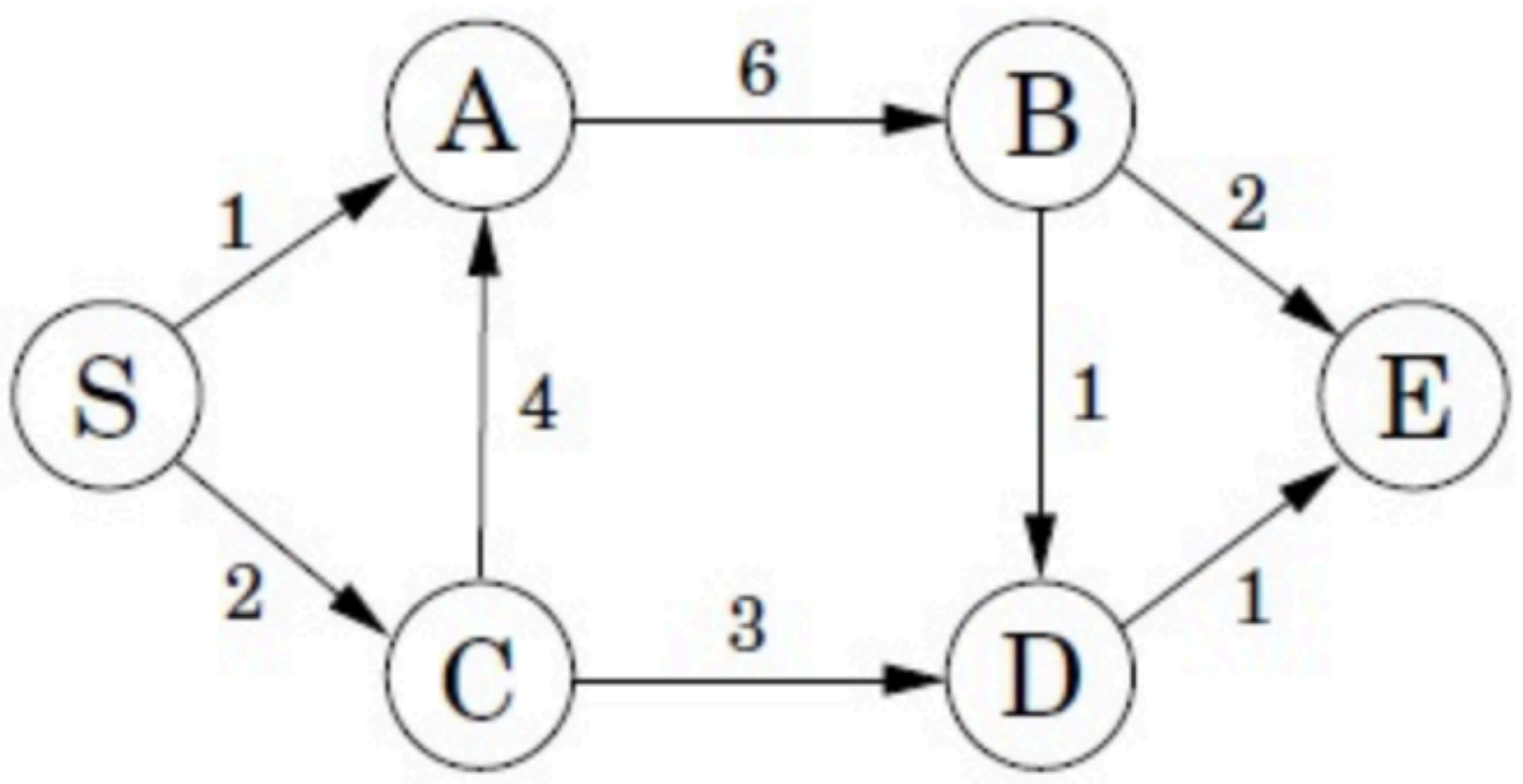


Now let’s redo everything with discount factor equal 1/4!

J4(E)=0, J4(B)=-2, J4(D)=-1.

J3(E)=0, J3(B)=-1.25, J3(D)=-1, J3(A)=-6.5, J3(C)=-3.25.

J2(E)=0, J2(B)=-1.25,J2(D)=-1,J2(A)=-6.3125,J2(C)=-3.25, J2(S)=-2.625



Now let's redo everything with discount factor equal 1/4!

J4(E)=0, J4(B)=2, J4(D)=-1.

J3(E)=0, J3(B)=-1.25, J3(D)=-1, J3(A)=-6.5, J3(C)=-3.25.

J2(E)=0, J2(B)=-1.25,J2(D)=-1,J2(A)=-6.3125,J2(C)=-3.25,J2(A)=-6.625, J2(S)=-2.625

J1(E)=0, J1(B)=-1.25,J1(D)=-1,J1(A)=-6.3125,J1(C)=-3.25,J1(A)=-6.625, J1(S)=-2.578

J0(E)=0, J0(B)=-1.25,J0(D)=-1,J0(A)=-6.3125,J0(C)=-3.25,J0(A)=-6.625, J0(S)=-2.578

This finds the path S—>A—>B—>D—>E

A small discount factor has heavy preference towards costs incurred LATER

In practice, you want a discount factor close to one.

- Let's go back to our equations $J^*_{K-1}(s) = \max_a r(s,a) + \gamma \sum_{s'} P(s' \mid s,a) J(s')$

  $a^*_{K-1}(s) = \arg\max_a r(s,a) + \gamma \sum_{s'} P(s' \mid s,a) J(s')$

  $J^*_k(s) = \max_a r(s,a) + \gamma \sum_{s'} P(s' \mid s,a) J^*_{k+1}(s')$

  $a^*_k(s) = \arg\max_a r(s,a) + \gamma \sum_{s'} P(s' \mid s,a) J^*_{k+1}(s')$

- Let's write this in a more compact way that will be fundamental for us.

  Intuition: we will stack up all the $J^*_k(s)$ into a single vector J^*_k.

- For example, on the previous page we had

  J4*(E)=0, J4*(B)=-2, J4*(D)=-1, all others are +infinity

  J3*(E)=0, J3*(B)=-1.25, J3*(D)=-1, J3*(A)=-6.5, J3*(C)=-3.25.

  We want two write that as

  $$J^*_4 = -\begin{pmatrix} \infty \\ \infty \\ -2 \\ +\infty \\ -1 \\ 0 \end{pmatrix} \text{ and } J^*_3 = -\begin{pmatrix} +\infty \\ -6.5 \\ -1.25 \\ -3.25 \\ -1 \\ 0 \end{pmatrix}$$

- Let's think about what the above equations do on vectors.

- Suppose we have a vector V with as many entries as the number of states. We will refer the s'th entry of V as $V(s)$.
- Let us define the vector TV whose s'th entry is
  $$(TV)(s) = \max_a r(s, a) + \gamma \sum_{s'} P(s' \mid s, a)V(s')$$
- So, using n for the total number of states, T maps vectors in $\mathbb{R}^n$ to vectors in $\mathbb{R}^n$. The bullet above tells us what it does on each coordinate.
- The map T used the expected reward $r(s, a)$ in the definition. If we were very formal, we should write something like T_r to emphasize the dependence on the quantities $r(s, a)$. But we will just write T and the expectation is that this will be understood from context.
- Interpretation: TV is the cost-to-go in a **one step** dynamical programming problem with terminal reward V (and expected rewards $r(s, a)$).

- Let's go back to:

$$a_1^*(s) \in \arg\max_{a \in A} r(s,a) + \gamma \sum_{s'} P(s' | s,a) J(s')$$

$$J_1^*(s) = \max_{a \in A} r(s,a) + \gamma \sum_{s'} P(s' | s,a) J(s')$$

$$a_0^*(s) \in \arg\max_{a} r(s,a) + \gamma \sum_{s'} P(s' | s,a) J_1^*(s')$$

$$J_0^*(s) = \max_{a} r(s,a) + \gamma \sum_{s'} P(s' | s,a) J_1^*(s')$$

- Can write the 2nd & 4th equations more compactly as:

$$J_1^* = TJ$$

$$J_0^* = TJ_1^*.$$

- So $J^* = T^2 J$.
- Interpretation: $T^2 J$ is the optimal reward in a two stage dynamic programming problem with terminal reward J.

- Let's go back to:

$$J_{K-1}(s) = \max_a r(s,a) + \gamma \sum_{s'} P(s' | s, a) J(s')$$

$$a^*_{K-1}(s) = \arg\max_a r(s,a) + \gamma \sum_{s'} P(s' | s, a) J(s')$$

$$J^*_k(s) = \max_a r(s,a) + \gamma \sum_{s'} P(s' | s, a) J^*_{k+1}(s')$$

$$a^*_k(s) = \arg\max_a r(s,a) + \gamma \sum_{s'} P(s' | s, a) J^*_{k+1}(s')$$

- Let's write this as

$$J^*_{K-1} = TJ$$

$$J^*_k = TJ^*_{k+1}.$$

- In particular, $J^* = T^K J$.
- Interpretation: $T^K J$ is the optimal reward in a K stage dynamical programming problem with terminal reward J.
- In other words,

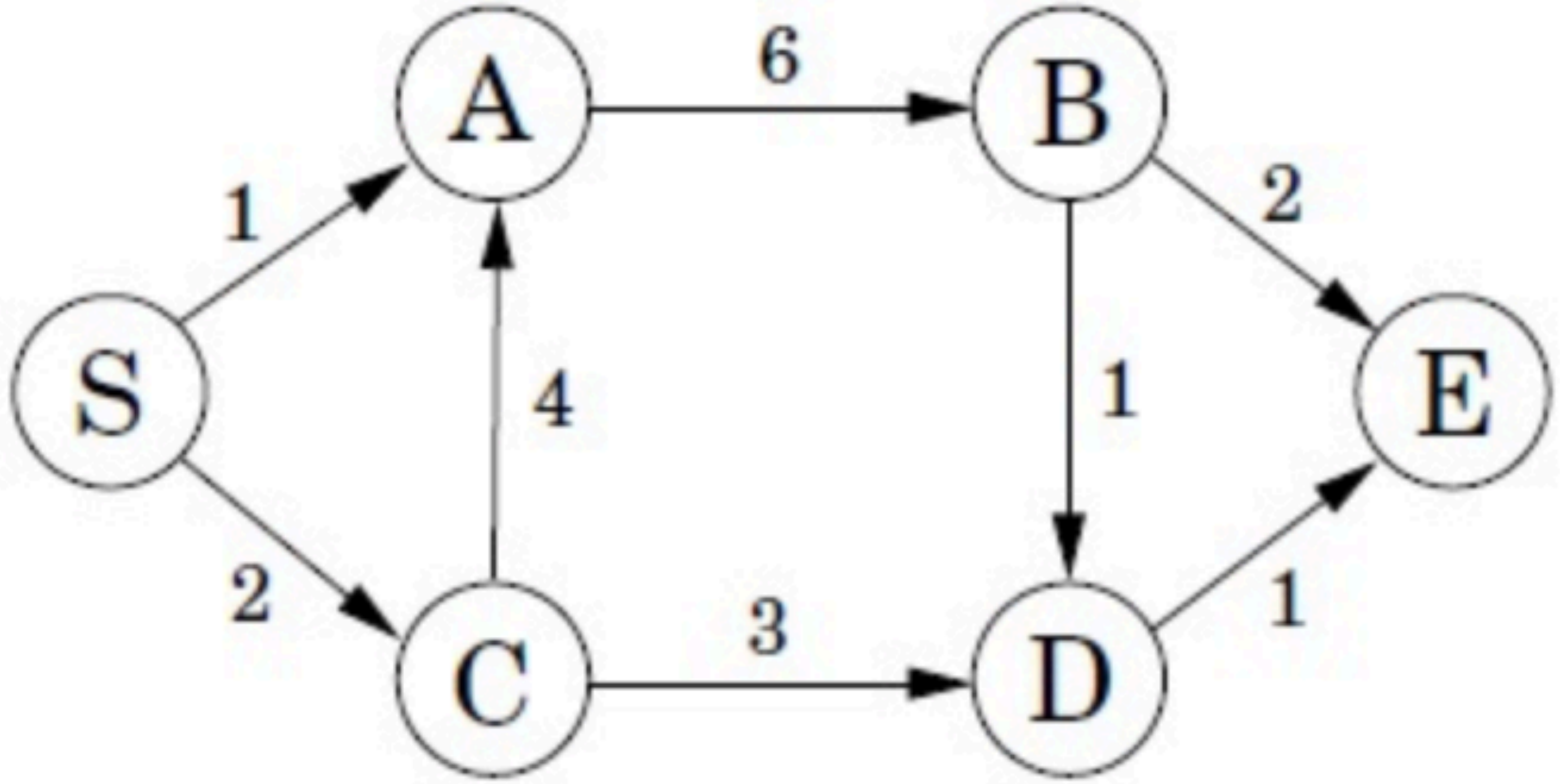
$$T^K J(s) = \max_{\pi \in \Pi_{\text{det}}} E\left[r_0 + \gamma r_1 + \cdots + \gamma^{K-1} r_{K-1} + \gamma^K J(s_K)\right]$$

where the maximum is taken over all deterministic policies which choose action based on the state.

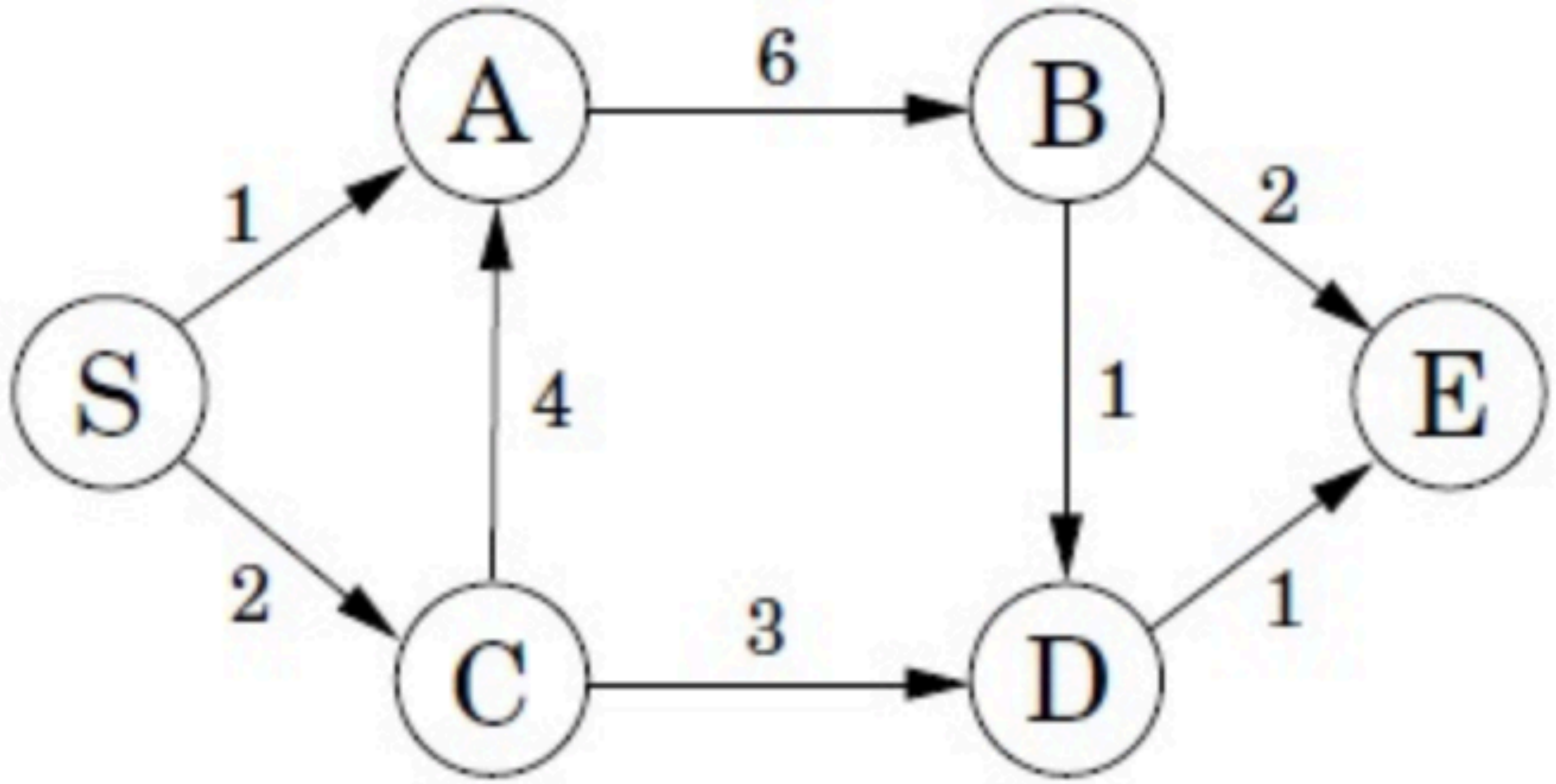
- Punchline: dynamic programing is all about applying the operator T.
- By the way, T is called the Bellman operator. Essentially, this entire course can be thought of as the study of its properties.
- Let's ask the following question: does randomization help?
  That is, suppose you didn't have to choose an action deterministically; suppose instead you could choose the action from a distribution.
- So we want to solve

$$\min_{\pi \in \Pi_{\text{rand}}} E\left[r_0 + \gamma r_1 + \cdots + \gamma^{K-1} r_{K-1} + \gamma^K J(s_K)\right]$$

  Is this better than if Π_{rand} was replaced by Π_{det}?



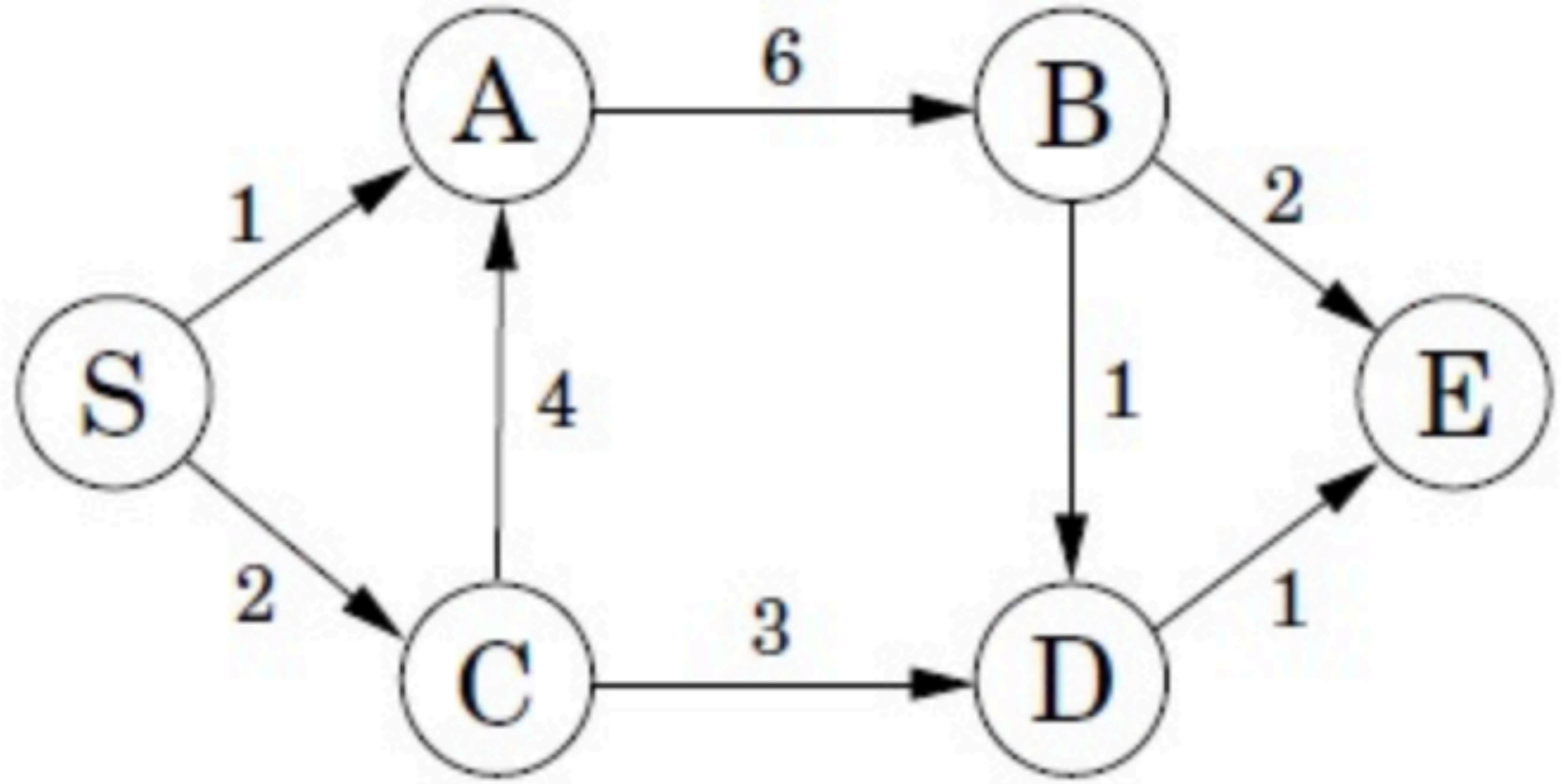
Let's go back here.
Suppose you are at B.
Does it help to be able to toss a coin, and go to E if it comes up on heads and D if it comes up tails?



Now suppose the discount factor equals 1/2.

Suppose you are at B.

Does it help to be able to toss a coin, and go to E if it comes up on heads and D if it comes up tails?



Now suppose you are at C, under either of the two discount factors. Does tossing a coin help?

- Conclusion: randomization doesn't help.
- So

$$T^K J = \min_{\pi \in \Pi_{\text{det}}} E\left[r_0 + \gamma r_1 + \cdots + \gamma^{K-1} r_{K-1} + \gamma^K J(s_K)\right]$$

- But also $T^K J = \min_{\pi \in \Pi_{\text{rand}}} E\left[r_0 + \gamma r_1 + \cdots + \gamma^{K-1} r_{K-1} + \gamma^K J(s_K)\right]$
- Let's look again at the definition of T:

$$(TV)(s) = \max_a r(s, a) + \gamma \sum_{s'} P(s' \mid s, a) V(s')$$

- Recall that a linear map is of the form $V \to MV$ for some matrix M.
- An affine map is of the form $V \to V_0 + MV$ for some matrix M and some vector V_0.
- However, the operator T is neither linear nor affine, because of the max in the definition.

- It might have occurred to you that “Dynamic Programming” is a weird name for all this.
- The origins of all the material in this class date back to work done by Richard Bellman in the 1950s at the RAND corporation.
- “An interesting question is, ‘Where did the name, dynamic programming, come from?’
  The 1950s were not good years for mathematical research. We had a very interesting gentleman in Washington named Wilson. He was Secretary of Defense, and he actually had a pathological fear and hatred of the word, research. I’m not using the term lightly; I’m using it precisely. His face would suffuse, he would turn red, and he would get violent if people used the term, research, in his presence. You can imagine how he felt, then, about the term, mathematical.
  The RAND Corporation was employed by the Air Force, and the Air Force had Wilson as its boss, essentially. Hence, I felt I had to do something to shield Wilson and the Air Force from the fact that I was really doing mathematics inside the RAND Corporation. What title, what name, could I choose? In the first place I was interested in planning, in decision making, in thinking. But planning, is not a good word for various reasons. I decided therefore to use the word, ‘programming.’ I wanted to get across the idea that this was dynamic, this was multistage, this was time-varying—I thought, let’s kill two birds with one stone. Let’s take a word that has an absolutely precise meaning, namely dynamic, in the classical physical sense. It also has a very interesting property as an adjective, and that is it’s impossible to use the word, dynamic, in a pejorative sense. Try thinking of some combination that will possibly give it a pejorative meaning. It’s impossible.
  Thus, I thought dynamic programming was a good name. It was something not even a Congressman could object to. So I used it as an umbrella for my activities.”
- https://mathshistory.st-andrews.ac.uk/Biographies/Bellman/

- Next, let's define a closely related operator.
- Given any deterministic policy π, we define
$$(T_\pi V)(s) = r(s, \pi(s)) + \gamma \sum_{s'} P(s' \mid s, \pi(s)))V(s')$$
- Interpretation: $(T_\pi V)(s)$ is the expected cost in a one-stage dynamic programming with terminal reward V problem **when you follow policy π starting from state s.**
- Given a randomized policy which choose action a in state s with probability $\pi(a \mid s)$, define
$$(T_\pi V)(s) = E\left[r(s, \pi(s)) + \gamma \sum_{s'} P(s' \mid s, \pi(s)))V(s')\right]$$
where the expectation is taken with respect to the randomness of the policy π.
- Alternatively, can write this out explicitly as
$$(T_\pi V)(s) = \sum_a \pi(a \mid s) r(s, a) + \gamma \sum_a \pi(a \mid s) \sum_{s'} P(s' \mid s, a)V(s')$$
- Same interpretation: expected cost in a one-stage dynamic programming problem with terminal reward V when you follow randomized policy π starting from state s.
- Compare to
$$(TV)(s) = \max_a r(s, a) + \gamma \sum_{s'} P(s' \mid s, a)V(s')$$
- Somewhat confusingly, **both** T and T_π are called the Bellman operators.

- Of course, I could take the operator T_π and apply it to a vector multiple times.
- $(T_\pi^2 V)(s)$ is the cost in a 2-stage dynamic programming problem with terminal reward V when you follow policy π starting from state s.
- $T_\pi^n V$ is the cost in an n-stage dynamic programming problem with terminal reward V:
$$T_\pi^K J = E\left[r_0 + \gamma r_1 + \cdots + \gamma^{K-1} r_{K-1} + \gamma^K J(s_K)\right],$$
where the expectation is with respect to the rewards/states obtained by following π starting from state s.
- The arguments go analogously to what we have seen before.

- Let's pause and discuss where we are.
- We want to solve the fundamental RL problem: we want to figure out the optimal policy in a continuing MDP.
- This is hard. So we simplify the problem by assuming there are K stages.
- Now the problem has a solution, and we figured it out!
- You just take the operator T and apply it K times to the vector of terminal costs.
  If there's no terminal cost, this is the same as the terminal cost vector being a vector of zeros.
- OK, so might guess what's coming!
- To solve the RL problem, approximate it with an K stage problem, and then let K go to $+\infty$.
- This means we need to apply T to the zero vector K times, and then let $K \to \infty$ to obtain the optimal rewards. But does this process even make sense? Does it converge?
- Next: we study some fundamental properties of the operator T and develop some aspects of a general theories of operators like T. Once this is done, we'll revisit this argument.
- Punchline: the next bit feels like its a foray into math. But we need it.

- OK, let’s discuss this more deeply.
- We need to make an assumption which we’ll make throughout this course.

  Assumption: there exists some $M > 0$ such that all rewards lie in $[-M, M]$ with probability one.
- Referred to as the bounded rewards assumption.
- Usually holds, unless we break things by introducing $+\infty$ as a reward.
- Now let’s discuss the consequences of this assumption.

- Consider the RL problem (i.e., infinite time horizon).
- Fix a policy π.
- Quite reasonably, let us define

  $J_\pi(s) = E\left[r_0 + \gamma r_1 + \gamma^2 r_2 + \cdots\right]$

  Expectation here is taken with respect to the distribution of the rewards given that you start at node s and follow policy π.

  We refer to this as "the value of node s under π."
- Sometimes, people will write $J_\pi(s) = E_{s,\pi}\left[r_0 + \gamma r_1 + \gamma^2 r_2 + \cdots\right]$.

  Other times, the subscripts should be inferred from contexts.
- Clearly, $J_\pi(s) \leq M + \gamma M + \gamma^2 M + \cdots = \dfrac{M}{1-\gamma}$.
- Similarly, $J_\pi(s) \geq -\dfrac{M}{1-\gamma}$.

- Now let's consider the difference between the RL and dynamic programming problems under the bounded reward assumption.
- [BTW, we make the bounded rewards assumption by default from now on...won't even mention it.]
- Same MDP. Only difference is that at one point we stop the game after K steps. At the other point, we let it go on forever.
- So $J_\pi^{\mathrm{RL}}(s) = E\left[r_0 + \gamma r_1 + \gamma^2 r_2 + \cdots\right]$ while $J_\pi^{\mathrm{DP(K)}}(s) = E\left[r_0 + \gamma r_1 + \gamma^2 r_2 + \cdots + \gamma^K r_K\right]$
- On any sample run, the difference between them is just the discounted reward from step $K+1$ onward.
- So $|J_\pi^{RL} - J_\pi^{DP(n)}| = E\left|\gamma^{K+1} r_{K+1} + \gamma^{K+2} r_{K+2} + \cdots\right| \le \gamma^{K+1}\frac{M}{1-\gamma}$.
- Conclusion: $\lim_{K\to+\infty} |J_\pi^{RL} - J_\pi^{\mathrm{DP(K)}}| = 0$ for all policies π and integer n.
  Looks good!

- Now let's talk about the optimal value functions.
- Define $J^{\mathrm{RL}}(s) = \max_{\pi \in \Pi_{\det}} E\left[r_0 + \gamma r_1 + \gamma^2 r_2 + \cdots\right]$ while

  $J^{\mathrm{DP(K)}}(s) = \max_{\pi \in \Pi_{\det}} E\left[r_0 + \gamma r_1 + \gamma^2 r_2 + \cdots + \gamma^K r_K\right]$
- For any fixed policy π, the difference between the value functions corresponding to π is at most $\gamma^{K+1}\dfrac{M}{1-\gamma}$.
- It follows that $|J^{\mathrm{RL}}(s) - J^{\mathrm{DP(K)}}(s)| \leq \gamma^{K+1}\dfrac{M}{1-\gamma}$
- Conclusion: $\lim_{K \to \infty} |J^{\mathrm{RL}}(s) - J^{\mathrm{DP(K)}}(s)| = 0.$
- Punchline: as expected, dynamic programming approximates RL as $K \to +\infty$.

- Where we are: we want to solve RL.
- We know how to solve dynamic programming.
- But dynamic programming approximates RL as the time horizon goes to infinity!
- So suppose you have a K stage problem with zero terminal cost. We know what the solution is:

  $J^{*,\mathrm{DP(k)}} = T^K\mathbf{0}$,

  where $\mathbf{0}$ refers to the all-zero vector.
- Conclusion: the optimal rewards in the RL problem can be obtained as

  $J^* = \lim_{K \to \infty} T^K\mathbf{0}$
- Also, if you have a concrete policy π and you want to evaluate it, you can then do so by computing $J_\pi = \lim_{t \to \infty} T_\pi^K\mathbf{0}$.
- This is nice, but we should be aware of the things:

  — do you really have to apply the operators an infinite number of times?

  — arguably, what we want is the optimal policy and not the optimal rewards. But clearly the two are connected?

- Let us consider the equation $J_\pi = \lim_{K \to \infty} T_\pi^K \mathbf{0}$.
- It would be nice to have a set of equations that are satisfied by J_π. Any guesses?
- Very natural to guess that J_π is a solution to $J = T_\pi J$!
- Question 1: is this true?
- Question 2: are there any other solutions?
- Similarly, since the optimal rewards in the RL problem satisfy $J^* = \lim_{K \to \infty} T^K \mathbf{0}$ it is natural to guess that J^* is a solution of the equation $J = TJ$.
- To argue all this convincingly, we need to do a detour and discuss the properties of the operators T, T_π

- If V_1, V_2 are vectors, will say that $V_1 \leq V_2$ if the inequality holds elementwise. For example, $\begin{pmatrix} 1 \\ 3 \end{pmatrix} \leq \begin{pmatrix} 2 \\ 4 \end{pmatrix}$
- Note that for any two numbers x, y we have that either $x \leq y$ or $y \leq x$ (or both). But the same is not true for vectors.
- For example, $\begin{pmatrix} 1 \\ 2 \end{pmatrix}$ and $\begin{pmatrix} 0 \\ 3 \end{pmatrix}$ are incomparable.
- **Claim:** If $V_1 \leq V_2$, then $TV_1 \leq TV_2$.
- **Claim:** If $V_1 \leq V_2$, then $T_{\underset{\pi}{\mu}} V_1 \leq T_{\underset{\pi}{\mu}} V_2$ for any policy $\overset{\pi}{\cancel{\mu}}$.
- Why is this true?
- This is called the **monotonicity lemma.**

- Let us denote by $\mathbf{e}_n$ the all-ones vector in $\mathbb{R}^n$, e.g., $\mathbf{e}_3 = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$.
- What is $T(V + \mathbf{e})$?
- Let us go back to the definition:

$$(TV)(s) = \max_a r(s, a) + \gamma \sum_{s'} P(s' \,|\, s, a) V(s')$$

- So

$$\begin{aligned}(T(V + \mathbf{e}))(s) &= \max_a r(s, a) + \gamma \sum_{s'} P(s' \,|\, s, a)(V(s') + 1) \\ &= \max_a r(s, a) + \gamma \sum_{s'} P(s' \,|\, s, a) V(s') + \gamma\end{aligned}$$

- Or: $T(V + \mathbf{e}) = Tv + \gamma \mathbf{e}$.
- Similarly, we have that $T(V + \alpha \mathbf{e}) = TV + \gamma \alpha \mathbf{e}$
- This is a fundamental property of the Bellman operator.

- Let's play with this some more.
- What is $T^2(V + \mathbf{e})$?
- Well, it is $T(T(V + \mathbf{e}))$
- But using the argument we just made, this is $T(TV + \gamma\mathbf{e})$.
- OK, but this is $T(TV) + \gamma^2\mathbf{e}$.
- Conclusion: $T^2(V + \mathbf{e}) = T^2V + \gamma^2\mathbf{e}$.
- Let's go deeper:

  $T^3(V + e) = T(T^2V + \gamma^2\mathbf{e}) = T^3V + \gamma^3\mathbf{e}$.
- More generally, $T^n(V + \mathbf{e}) = T^nV + \gamma^n\mathbf{e}$.

- The same properties hold for the operator T_π.
- Indeed, from the definition:

$$(T_\pi V)(s) = \sum_a \pi(a \mid s) r(s, a) + \gamma \sum_a \pi(a \mid s) \sum_{s'} P(s' \mid s, a) V(s')$$

- So

$$\begin{aligned}(T_\pi(V + \mathbf{e}))(s) &= \sum_a \pi(a \mid s) r(s, a) + \gamma \sum_a \pi(a \mid s) \sum_{s'} P(s' \mid s, a)(V(s') + 1) \\ &= \sum_a \pi(a \mid s) r(s, a) + \gamma \sum_a \pi(a \mid s) \left(\sum_{s'} P(s' \mid s, a) V(s') + 1 \right) \\ &= \sum_a \pi(a \mid s) r(s, a) + \gamma + \gamma \sum_a \pi(a \mid s) \sum_{s'} P(s' \mid s, a) V(s')\end{aligned}$$

- Thus $T_\pi(V + \mathbf{e}) = T_\pi V + \gamma\mathbf{e}$.
- From this it follows that $T_\pi(V + \alpha\mathbf{e}) = T_\pi V + \gamma\alpha\mathbf{e}$ as before.
- Similarly, $T_\pi^k(V + \mathbf{e}) = T_\pi^k V + \gamma^k\mathbf{e}$.
- This property is called sub-homogeneity.
- Maybe you are thinking: this is all very nice, but where is this going?
- Well, we've established two properties of the operators T, T_π: monotonicity and subhomogeneity.
- ...not done yet. Next, we discuss a further property of these maps: **contraction**.

- The infinity norm of a vector, denoted by $||x||_\infty$, is the absolute value of its largest entry.
- Example: $\left|\left|\begin{pmatrix}1\\2\\3\end{pmatrix}\right|\right|_\infty = 3$ and $\left|\left|\begin{pmatrix}-4\\2\\3\end{pmatrix}\right|\right|_\infty = 4.$
- We have $||x||_\infty = 0$ if and only if $x = 0.$
- We have that if α is a scalar, then $||\alpha x||_\infty = |\alpha|\,||x||_\infty.$
- We have that $||x + y||_\infty \leq ||x||_\infty + ||y||_\infty.$
- These properties are usually taken to be the definition of being a norm.

- Definition: the mapping T is a strict contraction in the infinity norm if for any two vectors x, y we have that $||Tx - Ty||_\infty < ||x - y||_\infty$
- Interpretation: T brings vectors closer together.
- Definition: if $\alpha \in (0,1)$, then the map T is an α-contraction in the infinity norm if $||Tx - Ty||_\infty \leq \alpha ||x - y||_\infty$.
- Note: we can talk about contractions in any norm, of course. But for RL, we will mostly focus on the infinity norm.

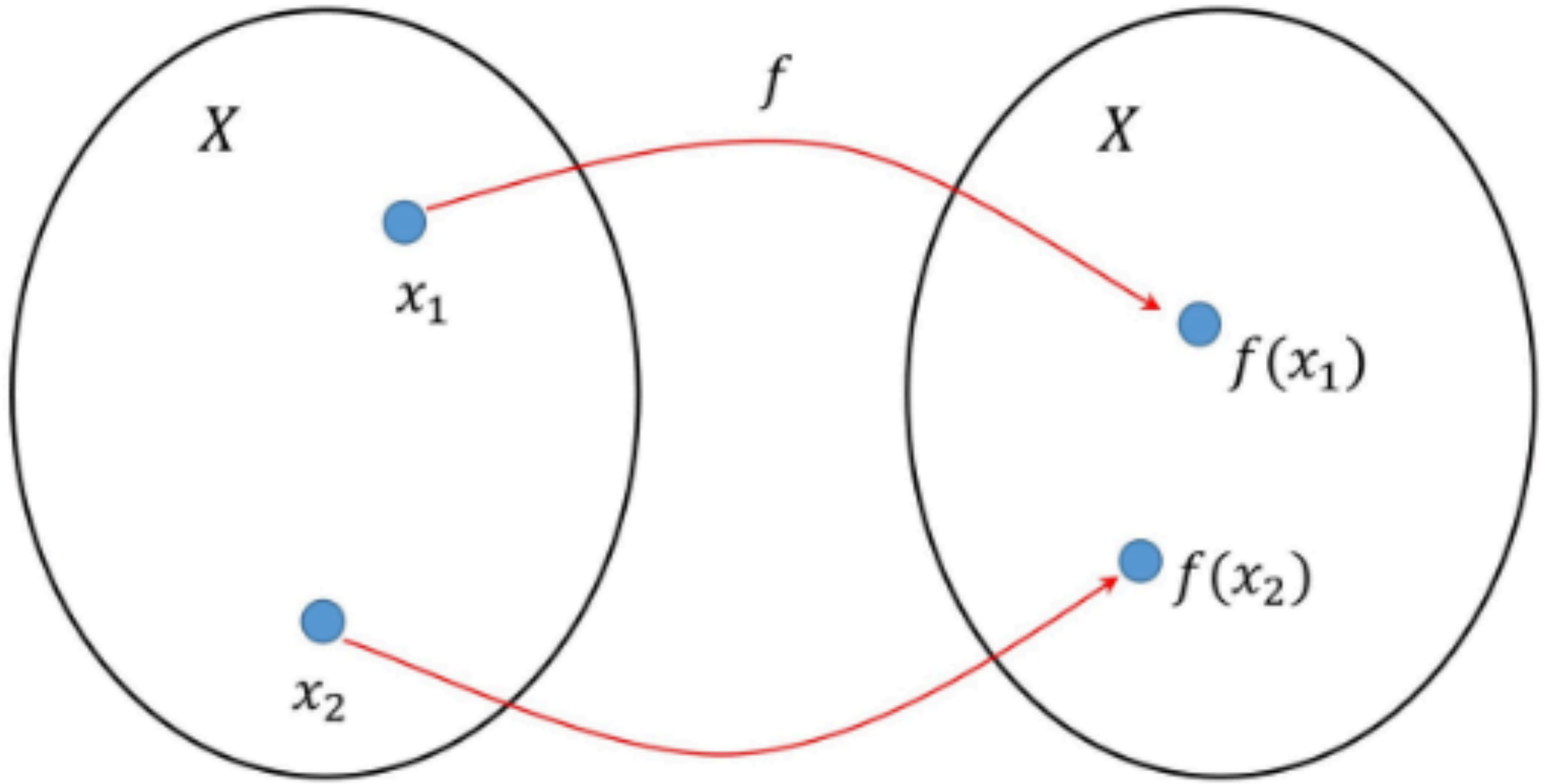


Figure 5.2.1: Contraction mapping $f : X \to X$ shrinks distances between arbitrary two points in a normed space X.

Good image to have in mind.
For us, the space X will always be R^n
And we will measure distances using the infinity norm

- **Claim**: the map T_μ is a γ-contraction.
- Proof: consider two vectors V_1, V_2.
- Suppose $c = ||V_1 - V_2||_\infty$. Want to argue that $||TV_1 - TV_2||_\infty \leq \gamma c$.
- We have that

  $V_1 - c\mathbf{e} \leq V_2 \leq V_1 + c\mathbf{e}$
- Apply monotonicity: $T(V_1 - c\mathbf{e}) \leq T(V_2) \leq T(V_1 + c\mathbf{e})$
- Apply subhomogeneity: $T(V_1) - c\gamma\mathbf{e} \leq T(V_2) \leq T(V_1) + \gamma c\mathbf{e}$
- Conclude: $||T(V_2) - T(V_1)||_\infty \leq \gamma c$. Done!
- **This also implies T is a γ-contraction**. Why?

- OK, but this gives us a new perspective on the claims that $J_\pi = \lim_{K\to\infty} T_\pi^K \mathbf{0}$ and $J^* = \lim_{K\to\infty} T^K \mathbf{0}$.
- For any two vectors $V_1, V_2,$ we have that $||TV_1 - TV_2|| \le \gamma ||V_1 - V_2||_\infty$.
- OK, but this means $||T^2V_1 - T^2V_2||_\infty = ||T(TV_1) - T(TV_2)||_\infty \le \gamma ||TV_1 - TV_2||_\infty \le \gamma^2 ||V_1 - V_2||_\infty$
- Likewise, $||T^KV_1 - T^KV_2||_\infty \le \gamma^k ||V_1 - V_2||_\infty$.
- In particular, $\lim_{K\to\infty} ||T^KV_1 - T^KV_2||_\infty = 0.$
- Conclusion: $J^* = \lim_{K\to\infty} T^K J$ for any vector J.
- Similarly: $J_\pi = \lim_{K\to\infty} T^K J$ for any vector J.

- We can get more mileage out of the contraction property. **Claim:** $T_\pi J_\pi = J_\pi$.
- Proof: let $J^{(k)} = T_\pi^k J$ for some vector J. Let $\epsilon_k = || J^{(k)} - J_\pi ||_\infty$. We know that $\epsilon_k \to 0$ and that

$$J^{(k)} - \epsilon_k \mathbf{e} \leq J_\pi \leq J^{(k)} + \epsilon_k \mathbf{e}.$$

- Use monotonicity: $T_\pi(J^{(k)} - \epsilon_k \mathbf{e}) \leq T_\pi J_\pi \leq T_\pi(J^{(k)} + \epsilon_k \mathbf{e})$
- Use subhomogeneity: $T_\pi J^{(k)} - \gamma\epsilon_k \mathbf{e} \leq T_\pi J_\pi \leq T_\pi J^{(k)} + \gamma\epsilon_k \mathbf{e}$.
- OK, but this is the same as $J^{(k+1)} - \gamma\epsilon_k \mathbf{e} \leq T_\pi J_\pi \leq J^{(k+1)} + \gamma\epsilon_k \mathbf{e}$.
- Take limit as $k \to \infty$ to obtain that $\lim_k J^{(k+1)} = T_\pi J_\pi$.
- But since $\lim_k J^{(k} = J_\pi$, we get $J_\pi = T_\pi J_\pi$.
- By exactly the same argument, $J^* = TJ^*$.

- The equations $J_\pi = T_\pi J_\pi$ and $J^* = TJ^*$ are called the Bellman equations.
- This is good! We now have equations satisfied by the things we are looking for.
- Let's look deeper into this.
- Consider the equation $J = T_\pi J$. Could it have more than one solution?
- More broadly, consider the equation $x = Tx$ where T is an α-contraction where $\alpha \in (0,1)$. Could this equation have more than one solution?
- **Claim:** no.

- Suppose $x_1 = Tx_1$ and $x_2 = Tx_2$.
- $Tx_1 - Tx_2 = x_1 - x_2$
- So $||Tx_1 - Tx_2||_\infty = ||x_1 - x_2||_\infty$.
- But also $||Tx_1 - Tx_2||_\infty \leq \alpha ||x_1 - x_2||_\infty$ with $\alpha \in (0,1)$.
- So $||x_1 - x_2||_\infty \leq \alpha ||x_1 - x_2||_\infty$.
- Conclusion: $||x_1 - x_2||_\infty = 0$ or $x_1 = x_2$.
- Consequences:

— J_π is the **unique** solution of $T_\pi J = J$.

— J^* is the unique solution of $TJ = J$

- Let's look at the Bellman equations, starting with the Bellman equation for a policy: $T_\pi J = J$.
- OK, but let's go back to the definition of T_π:

  $$(T_\pi V)(s) = \sum_a \pi(a \mid s) r(s,a) + \gamma \sum_a \pi(a \mid s) \sum_{s'} P(s' \mid s,a) V(s')$$
- Let's write this in matrix form! Let $r_\pi(s) = \sum_a \pi(a \mid s) r(s,a)$ be the expected reward in the next transition when you follow policy π.
- So $(T_\pi V)(s) = r_\pi(s) + \gamma \sum_{s'} \sum_a \pi(a \mid s) P(s' \mid s,a) V(s')$
- Further, let us set $P_\pi(s' \mid s) = \sum_a \pi(a \mid s) P(s' \mid s,a)$ to be the probability of transitioning from s to s' if you follow policy π.
- So $(T_\pi V)(s) = r_\pi(s) + \gamma \sum_{s'} P(s' \mid s) V(s')$. What does this look like using linear algebra?

- We write the definition

  $$(T_\pi V)(s) = r_\pi(s) + \gamma \sum_{s'} P(s' \mid s) V(s')$$

  in vector form as

  $$T_\pi V = r + \gamma P_\pi V$$

  where P_π is the matrix whose entry in row s and column s' is $P_\pi(s' \mid s)$.
- OK so the Bellman equation $T_\pi J = J$ can be written as

  $$r + \gamma P_\pi J = J.$$
- This is a linear system of equations! We can solve it:

  $$r = (I - \gamma P_\pi) J$$

  and therefore

  $$J_\pi = (I - \gamma P_\pi)^{-1} r.$$
- Punchline: we can find the value of any fixed policy by solving a linear system of equations.

- Let's consider a few simple equations.
- Consider an MDP with one state.

  You have a single action to take, get a reward of 1, and transition to the same/only state.

  Discount factor equals 1/2
- What is the value of this policy? First, let's compute it directly...
- 1 + (1/2) + (1/2)^2 + (1/2)^3 + ... = 2.
- Using the method on the previous slide, the value of the single state is the solution of

  $$V(1) = r(1) + \gamma \cdot V(1)$$

  [here the matrix P_π is just the scalar 1]

  or

  $$V(1) = 1 + \frac{1}{2}V(1)$$
- This does indeed reduce to $V(1) = 2$ as we expect.

- How about an MDP with two states. Given the policy we pursue:
  — at state 1, you get a reward of 2 on each transition, and transition to state 2 with probability 1/2
  — at state 2, you get a reward of 1 on each transition, and transition to state 2 with probability 1.
- The matrix P_π is $P_\pi = \begin{pmatrix} 1/2 & 1/2 \\ 0 & 1 \end{pmatrix}$.
- $V(1) = 2 + \gamma \left(\frac{1}{2} V(1) + \frac{1}{2} V(2) \right)$

  $V(2) = 1 + \gamma V(2)$
- Supposing $\gamma = 1/2$, we can get $V(1) = 10/3, V(2) = 2$

- How about an MDP with two states. Given the policy we pursue:
  — at state 1, you get a reward of 2 on each transition, and transition to state 2 with probability 1/2
  — at state 2, you get a reward of 1 on each transition, and transition to state 2 with probability 3/4, and to state 1 with probability 1/4.
- The matrix P_π is $P_\pi = \begin{pmatrix} 1/2 & 1/2 \\ 1/4 & 3/4 \end{pmatrix}$.
- $V(1) = 2 + \gamma \left(\frac{1}{2} V(1) + \frac{1}{2} V(2) \right)$

  $V(2) = 1 + \gamma \left(\frac{1}{4} V(1) + \frac{3}{4} V(2) \right)$
- Supposing $\gamma = 1/2$, we have

  $V = \left(I - \frac{1}{2} \begin{pmatrix} 1/2 & 1/2 \\ 1/4 & 3/4 \end{pmatrix} \right)^{-1} \begin{pmatrix} 2 \\ 1 \end{pmatrix} \approx \begin{pmatrix} 3.43 \\ 2.26 \end{pmatrix}$

- OK, but that is just the value of following a policy π.

  Would be amazing if we use a similar method to solve for $J*$.
- Unfortunately, there the situation is more complicated. As we have already discussed, $J*$ is the unique solution of the equation $TJ = J$.

  But recall the definition of the operator T:

  $$(TV)(s) = \max_a r(s,a) + \gamma \sum_{s'} P(s' \mid s,a) V(s')$$
- It is not linear: it has a max in it. Trying to solve $TJ = J$ directly means solving a nonlinear system of equations, and it is not entirely clear how to do that.
- In fact, a good method is to just pick any J (the zero vector works fine), and construct the sequence $TJ, T^2J, T^3J, \ldots$
- This is called **value iteration**. As we discussed, the limit is $J*$

- All right, suppose we construct a sequence according to

  $J_{k+1} = TJ_k$.

  We have that $J_k \to J^*$.

  So we can compute J^*. But what we really want is the optimal policy....

- **Claim:** a policy π is optimal if and only if $TJ^* = T_\pi J^*$ (*).

- Proof: indeed, if π is optimal, then $J_\pi = J^*$.

  So $TJ^* = J^* = J_\pi = TJ_\pi$

- On the other hand, if Eq. (*) holds, then $J^* = T_\pi J^*$. But since J_π is the unique solution of these equations, we have $J^* = J_\pi$

- So our strategy is:

  (i) Compute J^* by iterating $J_{k+1} = TJ_k$

  (ii) Solve for the optimal policy π from the equation $TJ^* = T_\pi J^*$

  How hard is it to solve (ii)?

- Not hard.

  $$TJ^*(s) = \max_a r(s,a) + \gamma \sum_{s'} P(s'|s,a)J^*(s') \qquad \text{(!!)}$$

  $$T_\pi J^*(s) r(s,\pi(s)) + \gamma \sum_{s'} P(s'|s,\pi(s))V(s')$$

  (for a deterministic policy π).

  So how to choose π?

- Easy: for each state s, choose $\pi(s)$ to be the maximizing action in Eq. (!!).
- You now know the **value iteration** method.
- Implication: no benefit from having randomized π.

- Let's work this out on the simplest example.
- The MDP has a single state, and two actions.

  Action 1 gives a reward of 1.

  Action 2 gives a reward of 2.

  Discount factor is $1/2$
- The value vector is just a scalar.

  $$T(J) = \max\left(1 + \frac{1}{2}J, 2 + \frac{1}{2}J\right)$$
- Easy: $T(J) = 2 + J/2$.
- Suppose we initialize $J_0 = 0$.

  $J_1 = 2.$

  $J_2 = 3$

  $J_3 = 3.5$

  $J_4 = 3.75$

  $J_5 = 3.875$

  $J_6 = 3.9375$
- Will approach $J = 4$, which is indeed the optimal value.
- To find optimal policy choose the action that achieves the maximum in $T(J) = \max\left(1 + \frac{1}{2}J, 2 + \frac{1}{2}J\right)$ when $J = 4$.

- Let's try another simple example.
  Two states.
  In state 1, you can take two actions.
  Action 1 brings you to state 1 with a reward of 2.
  Action 2 brings you to state 2 with a reward of 4.
  Once in state 2, any action you take gives reward of zero and keeps you in state 2.
  Discount factor is 0.9.
- $$TV(1) = \max\left(2 + \frac{9}{10}V(1), 4 + \frac{9}{10}V(2)\right)$$
  $$TV(2) = \max\left(0 + \frac{9}{10}V(2), 0 + \frac{9}{10}V(2)\right)$$

- $TV(1) = \max\left(2 + \frac{9}{10}V(1), 4 + \frac{9}{10}V(2)\right)$

  $TV(2) = \max\left(0 + \frac{9}{10}V(2), 0 + \frac{9}{10}V(2)\right)$

- Support you start this at $J_0 = \begin{pmatrix} 10 \\ 10 \end{pmatrix}$.

- $J_1 = \begin{pmatrix} 13 \\ 9 \end{pmatrix}$

- $J_2 = \begin{pmatrix} 13.7 \\ 8.1 \end{pmatrix}$

- $J_3 = \begin{pmatrix} 14.33 \\ 7.29 \end{pmatrix}$, $J_4 = \begin{pmatrix} 14.87 \\ 6.56 \end{pmatrix}$, ...

- Will approach $\begin{pmatrix} 20 \\ 0 \end{pmatrix}$.

- You then choose the actions that achieve the maximum in the definition of TV when $V = \begin{pmatrix} 20 \\ 0 \end{pmatrix}$.

- One last example! Two states.
  In state 1, you can take two actions.
  Action 1 brings you to state 1 with a reward of 2 with probability 3/4.
  With probability 1/4, the action brings you to state 2 with a reward. of -1
  Action 2 brings you to state 2 with a reward of 4.
  Once in state 2, any action you take gives reward of zero and keeps you in state 2.
  Discount factor is 0.9.
- Your expected reward from choosing action 1 in state 1 is $\frac{3}{4} \cdot 2 + \frac{1}{4} \cdot -1 = \frac{5}{4}$
- $TV(1) = \max\left(\frac{5}{4} + \frac{9}{10}\left(\frac{3}{4} \cdot V(1) + \frac{1}{4}V(2)\right), 4 + \frac{9}{10}V(2)\right)$
- $TV(2) = \max\left(0 + \frac{9}{10}V(2), 0 + \frac{9}{10}V(2)\right)$
- Will not solve it, but you do the same — just keep iterating $J_{t+1} = TJ_t$. Once you have the limit, you know it equals J^*. Optimal policy attains the maximum in definition of $T(V)$ when you plug in $V = J^*$.

- Where we are: we have a way of solving the reinforcement learning problem!

  We just iterative $J_{t+1} = TJ_t$ where T is the Bellman operator.
- Next: how long does this take?
- This is an important thing to know!

  Consider the sequence from a couple of slides ago:

  $$J_0 = \begin{pmatrix} 10 \\ 10 \end{pmatrix}, J_1 = \begin{pmatrix} 13 \\ 9 \end{pmatrix}, J_2 = \begin{pmatrix} 13.7 \\ 8.1 \end{pmatrix}, J_3 = \begin{pmatrix} 14.33 \\ 7.29 \end{pmatrix} \ldots$$

  which actually approaches $J^* = \begin{pmatrix} 20 \\ 0 \end{pmatrix}$.
- Want to know: when should you stop?
- What we want: $J_t - J^*$. What can we observe?
- Well, we can observe $J_t - J_{t+1}$, i.e., we can see when the iterates start changing slower and slower.
- So, goal: bound $J_t - J^*$ in terms of $J_t - J_{t-1}$.

- The trick, part one: write

$$J_t - J^* = J_t - \lim_{k \to +\infty} J_{t+k}$$

- Next, write this as

$$J_t - J^* = \lim_{k \to \infty} J_t - J_{t+1} + J_{t+1} - J_{t+2} + J_{t+2} \cdots + J_{t+k-1} - J_{t+k}$$

- Rewrite as:

$$J_t - J^* = \lim_{k \to +\infty} (J_t - J_{t+1}) + (TJ_t - TJ_{t+1}) + (T^2 J_t - T^2 J_{t+1}) + \cdots + (T^{k-1} J_t - T^{k-1} J_{t+1})$$

- Take infinity norm of both sides and use the triangle inequality:

$$||J_t - J^*||_\infty \leq \lim_{k \to \infty} ||J_t - J_{t+1}||_\infty + ||T(J_t - J_{t+1})||_\infty + ||T^2(J_t - J_{t+1})||_\infty + ||T^{k-1}(J_t - J_{t+1})||_\infty$$

- What's next?

- The trick, part two:

$$\begin{aligned} ||J_t - J^*||_\infty &\le \lim_{k\to\infty} ||J_t - J_{t+1}||_\infty + \gamma ||J_t - J_{t+1}||_\infty + \cdots + \gamma^{k-1} ||J_t - J_{t+1}||_\infty \\ &\le \lim_{k\to\infty} ||J_t - J_{t+1}||_\infty (1 + \gamma + \cdots + \gamma^k) \\ &\le ||J_t - J_{t+1}||_\infty \frac{1}{1-\gamma} \end{aligned}$$

- Conclusion: $||J_t - J^*||_\infty \le \frac{||J_t - J_{t+1}||_\infty}{1-\gamma}$. Exactly what we need!
- Example: a couple of slides ago we had $J_3 = \begin{pmatrix} 14.33 \\ 7.29 \end{pmatrix}$ and$J_4 = \begin{pmatrix} 14.87 \\ 6.56 \end{pmatrix}$ with discount $\gamma = 0.9$
- Might not seem like we are changing all that much: $||J_3 - J_4||_\infty \approx 0.73$. But the bound above says that $||J_3 - J^*||_\infty \le 7.3$, which is correct and basically tight (recall that limit was $\begin{pmatrix} 20 \\ 0 \end{pmatrix}$).
- Remark: for the update rule $J_{t+1} = T_\pi J_t$ which converges to J_π, we have that $||J_t - J_\pi||_\infty \le \frac{||J_t - J_{t+1}||_\infty}{1-\gamma}$. Why?

- Let’s recap.

  We now of a single method to solve the RL problem, namely $V_{t+1} = TV_{t'}$

  which is called value iteration.
- But one might argue this is not the most natural thing to do.

  How do human beings learn?
- We try something.

  It works or doesn’t.

  We do something to improve it.

  The result works or doesn’t.

  Improve again.

  Repeat.
- Can we do something like this in RL?

- So how about this: first, start with a policy π_0.
- If you don't have any good ideas for a policy to begin with, π_0 can always be "choose a random action."
- Or maybe: enumerate all the actions, choose the first one always.
- Next, compute J_{π_0}.

  Remember we discussed how to do this: $J \leftarrow T_\pi J$ converges to J_{π_0}.
- Next step: how do you improve the policy? Any ideas?

- Let's consider what happens at state s.

  You can choose from actions in the set A.

  Choosing action a gives you an expected reward $r(s, a)$and brings you to set s' with probability $P(s' | s, a)$.
- If you knew the optimal values of all the states, i.e., the quantities $J^*(s)$, then as we discussed you would choose the action

  $$\arg\max_a r(s, a) + \sum_{s'} P(s' | s, a) J^*(s')$$

  ...except you don't know the quantities $J^*(s)$.
- But you do know the quantities $J_{\pi_0}(s)$.
- Solution: you take the action

  $$\pi_1(s) \in \arg\max r(s, a) + \sum_{s'} P(s' | s, a) J_{\pi_0}(s').$$
- Idea: your best guess about the value of state s right now is $J_{\pi_0}(s)$, so you use that.
- Note: π_1 is deterministic.

- The human analogy should make sense here:

  — you try π_0.

  — you notice certain states are better than others [under this policy!]

  — so you take actions that steer you towards better states.

- OK, now you have π_1. What do you do next?
- You repeat!
- Compute J_{π_1} and then at state s choose the action

$$\pi_2(s) \in \arg\max_a r(s,a) + \sum_{s'} P(s' \mid s,a) J_{\pi_1}(s')$$

- Note: π_2 is deterministic.

- This algorithm is known as **policy iteration**.
- Initialize with some policy π_0.
- At step k:

  (1) Compute J_{π_k} using the update rule $J \leftarrow T_{\pi_k} J$.

  (2) update the policy by setting

  $$\pi_{k+1}(s) \in \arg\min r(s,a) + \gamma \sum_{s'} P(s' \mid s, a) J_{\pi_k}(s')$$

- Then there is **generalized policy iteration**:

  in step (1) only do a few update steps (i.e., don't compute J_{π_k} exactly).

- What is so attractive about policy iteration?
- Imagine you are about to go through the intersection when you notice the light is red (you missed this before). What do you do?
- You press on the brake!
- You don’t need to explore every single variation of what could happen, which is what value iteration does.
- Similarly, if you start with something stupid, generalized policy iteration will give you something reasonable very quickly.

- So policy iteration makes sense on an intuitive level. But lots of things that don't work make sense. We need a mathematical result here.
- We will show that π_k, the policy from the k'the round of policy iteration, approaches the optimal policy.
- First step: let's use the shorthand $J_k = J_{\pi_k}$, i.e., the value vector corresponding to the policy π_k

  So J_k is a vector with as many entries as the number of states.
- Claim: $J_{k+1} \geq J_k$.

  That is, for every state s, we have that

  $J_{k+1}(s) \geq J_k(s)$.
- Interpretation: policy iteration improves the policy from step to step.

- Why isn't this obvious?
- Because J_{k+1} is built on J_k:

  $$\pi_{k+1}(s) \in \arg\min r(s,a) + \gamma \sum_{s'} P(s' \mid s,a) J_k(s')$$

  $$J_{k+1} = J_{\pi_{k+1}}.$$

- Maybe if J_k is really wrong, J_{k+1} will be even more wrong, and the cycle perpetuates.
- Imagine a self-driving car that thinks hitting pedestrians is good. It will develop policies that hit pedestrians more and more.
- Aside: actually, this discussion is a good way to frame what RL is *really* about…
  …whether processes that start with random beliefs/policies and use those policies to *both* interpret the world and refine themselves will yield good outcomes.

- **Claim:** $J_{k+1} \geq J_k$.
- Let's prove this claim in slightly more general form.
- Given two policy $\pi, \hat{\pi}$, we will say that $\hat{\pi}$ is greedy with respect to π if

  $$\hat{\pi}(s) \in \arg\min_a r(s,a) + \gamma \sum_{s'} P(s' \mid s,a) J_\pi(s')$$

- So the policy iteration method can be restated as:

  "choose π_{k+1} to be greedy with respect to π_k"

- **Claim:** If $\hat{\pi}$ is greedy with respect to π, then

  $J_{\hat{\pi}} \geq J_\pi$.
- Will prove the latter claim.

- How do we prove something like this?
- One idea: let's look at the Bellman equation. For all states s, we have that

  $$J_\pi(s) = E\left[r(s,\pi(s)) + \gamma \sum_{s'} P(s' \mid s, \pi(s)) J_\pi(s')\right]$$

  and the expectation here is over the randomness of π.
- The right-hand side takes a convex combination of the quantities

  $$r(s,a) + \gamma \sum_{s'} P(s' \mid s, a) J_\pi(s')$$

  More specifically,

  $$J_\pi(s) = \sum_a \pi(a \mid s)\left(r(s,a) + \gamma \sum_{s'} P(s' \mid s, a) J_\pi(s')\right)$$
- OK, but then

  $$J_\pi(s) \le \max_a r(s,a) + \gamma \sum_{s'} P(s' \mid s, a) J_\pi(s').$$

  Why?

- Starting from the last equation$J_\pi(s) \le \max_a r(s,a) + \gamma \sum_{s'} P(s' \mid s,a) J_\pi(s').$

  ...how is $\hat{\pi}$ defined?
- We obtain:

  $$J_\pi(s) = r(s, \hat{\pi}(s)) + \gamma \sum_{s'} P(s' \mid s, \hat{\pi}(s)) J_\pi(s')$$
- What is a compact way to write the RHS?
- Recall that for a deterministic policy: $(T_{\hat{\pi}} V)(s) = r(s, \hat{\pi}(s)) + \gamma \sum_{s'} P(s' \mid s, \hat{\pi}(s)) V(s)$
- So we have $J_\pi(s) \le (T_{\hat{\pi}} J_\pi)(s)$
- In vector form: $J_\pi \le T_{\hat{\pi}} J_\pi$
- Not quite what we want but seems like progress.

- OK, so we established that $J_\pi \leq T_{\hat{\pi}} J_\pi$
- What's a natural step to take from here?
- Apply $T_{\hat{\pi}}$ to both sides and use monotonicity:

  $T_{\hat{\pi}} J_\pi \leq T_{\hat{\pi}}^2 J_\pi.$
- Combine: $J_\pi \leq T_{\hat{\pi}} J_\pi \leq T_{\hat{\pi}}^2 J_\pi.$
- Do you see where we are going with this?
- Take $T_{\hat{\pi}} J_\pi \leq T_{\hat{\pi}}^2 J_\pi$, apply $T_{\hat{\pi}}$ and use monotonicity:

  $T_{\hat{\pi}}^2 J_\pi \leq T_{\hat{\pi}}^3 J_\pi.$
- Put together: $J_\pi \leq T_{\hat{\pi}} J_\pi \leq T_{\hat{\pi}}^2 J_\pi \leq T_{\hat{\pi}}^3 J_\pi.$
- Can repeat as many times as we want to get

  $J_\pi \leq T_{\hat{\pi}} J_\pi \leq T_{\hat{\pi}}^2 J_\pi \leq \cdots \leq T_{\hat{\pi}}^k J_\pi$
- What is the next step?

- Take $J_\pi \le T_{\hat{\pi}} J_\pi \le T_{\hat{\pi}}^2 J_\pi \le \cdots \le T_{\hat{\pi}}^k J_\pi$ and let $k \to \infty$.
- Conclusion: $J_\pi \le J_{\hat{\pi}}$.
- We have proved the claim.
- In particular, $J_k \le J_{k+1}$: policy iteration does not make things at every step.
- We are making progress!
- Next step: what happens if policy iteration stops?

  i.e., what happens if $J_k = J_{k+1}$?
- Ideally would like: this means you are the optimal solution.
- Let's go back: what would it mean if $J_\pi \le J_{\hat{\pi}}$?

- Suppose $\hat{\pi}$ is greedy with respect to π.

  We showed that $J_\pi \leq T_{\hat{\pi}} J_\pi \leq T_{\hat{\pi}}^2 J_\pi \leq \cdots \leq T_{\hat{\pi}}^k J_\pi \to J_{\hat{\pi}}$
- In particular, $T_{\hat{\pi}}^k J_\pi$ approaches $J_{\hat{\pi}}$ from below, meaning that for all states s, we have

  $T_{\hat{\pi}}^k (J_\pi)(s) \leq J_{\hat{\pi}}(s)$
- So $J_\pi \leq T_{\hat{\pi}} J_\pi \leq T_{\hat{\pi}}^2 J_\pi \leq \cdots \leq T_{\hat{\pi}}^k J_\pi \leq J_{\hat{\pi}}$
- Now suppose $J_\pi = J_{\hat{\pi}}$. Then

  $J_\pi = T_{\hat{\pi}} J_\pi = T_{\hat{\pi}}^2 J_\pi = \cdots = T_{\hat{\pi}}^k J_\pi = J_{\hat{\pi}}$
- Let's focus on the first equation.
- We have that $T_{\hat{\pi}} J_\pi = T J_\pi$. Why?
- So $J_\pi = T J_\pi$!
- Conclusion: if $J_\pi = J_{\hat{\pi}}$, then π is optimal.

- Summary: at every step, policy iteration does not make things work at any state.
- It may make things better.
- If there's no improvement, we are at the optimal policy.
- These are all good things to know. Now let's combine to prove what we really want: policy iteration converges to the optimal policy.
- What we will actually show: there exists some index k_0 such that π_{k_0} is the optimal policy.

- Let's consider two cases.
- Case 1: there exists some index k' such that there is no improvement at step k', i.e., $J_{k'} = J_{k'+1}$.
- ...in that case, we have already shown that $\pi_{k'}$ is optimal. Take $k_0 = k'$ and we are done.
- Case 2: case 1 does not hold.
- This means that, for **every** index k, we have $J_k \neq J_{k+1}$.
- Because we have showed that $J_{k+1} \geq J_k$, this means that there exists at least one state s such that
  $J_{k+1}(s) > J_k(s)$.
- We will argue: case 2 cannot occur. This will complete the proof.

- OK, so we know that $J_{k+1} \geq J_k$. Suppose further that for every k, there exists a state s such that $J_{k+1}(s) > J_k(s)$.
- How can we argue this can't happen?
- Observation: for every k, we have that $\sum_s J_{k+1}(s) > \sum_s J_k(s)$. The inequality here is strict.
- So we want to argue $\sum_s J_k(s)$ cannot increase infinitely many times.
- Key insight: because

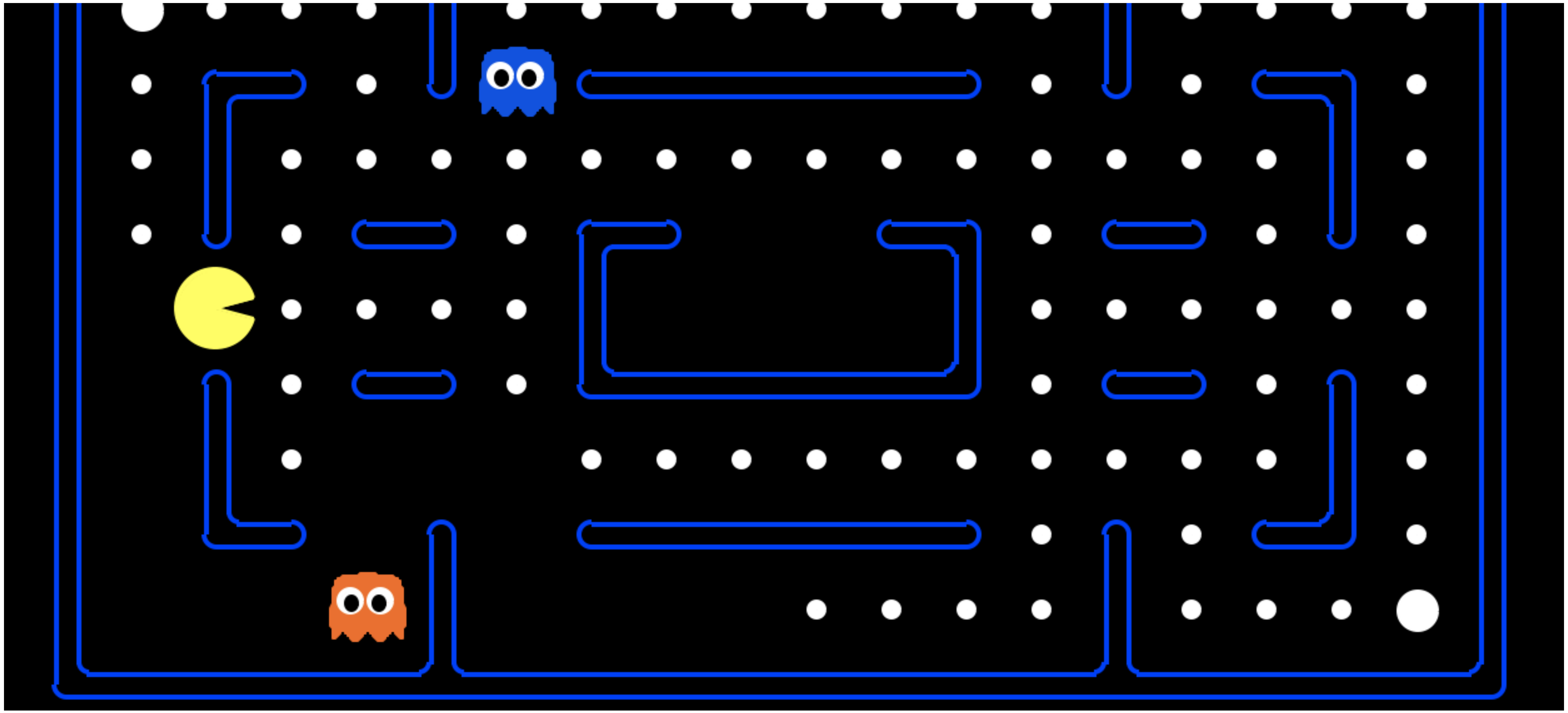
  — there are finitely many states

  — there are finitely many actions

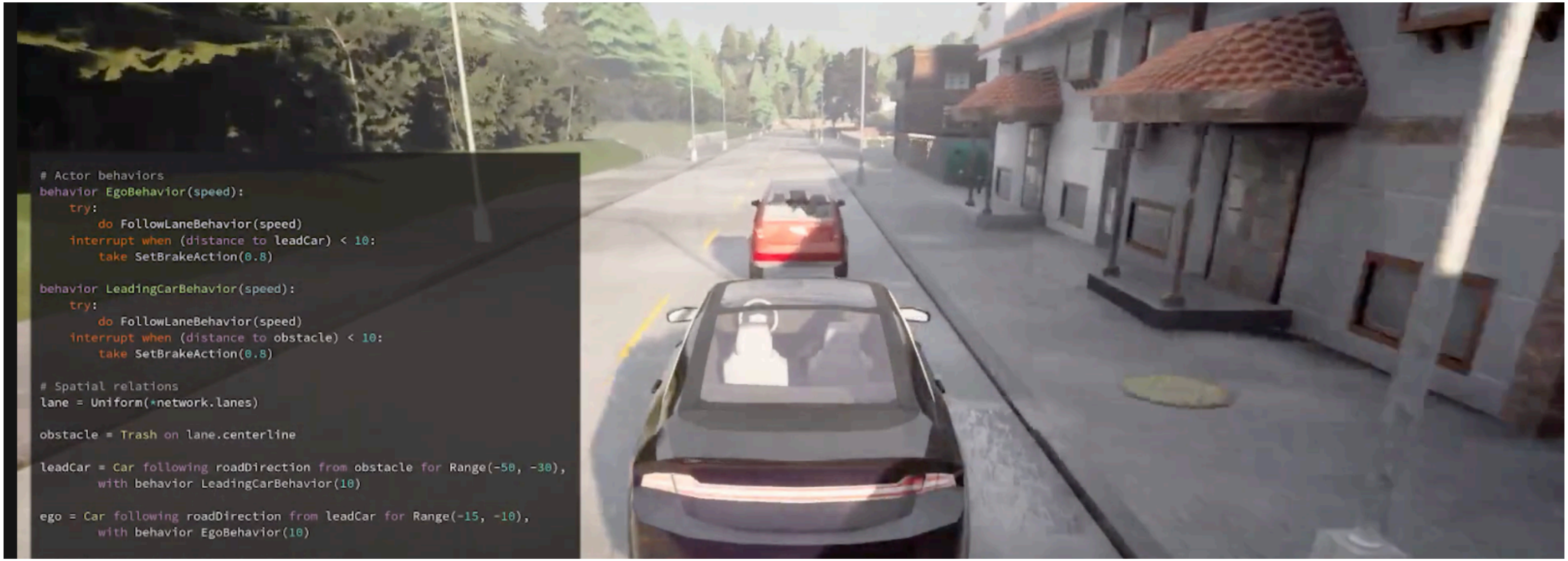
  … it follows there are finitely many deterministic policies!
- $\sum_s J_k(s)$ is the sum of the entries of the value vector of a deterministic policy.
- …so there's only finitely many values it can take!
- Therefore it cannot increase infinitely may times. We are done!

- This concludes our discussion of the basics of reinforcement learning.
- The two methods we have discussed — value iteration and policy iteration — form the foundations of the field. Every method used, in practice or in theory, is some variation on the ideas we have been discussing.
- Nevertheless, value iteration and policy iteration are not practical.
- There are two main reasons why this is so. What are they?

- Reason number 1: computing J_k or updating $J \leftarrow TJ$ means maintaining a vector with one entry per every state.
- Methods that do this, i.e., maintain a single entry per state, are called *tabular methods*.
  Value iteration and policy iteration are tabular methods.
- But in the real world, the state space is huge!



A state of Pacman is a configuration of:
— location of player
— location of ghosts
— direction of ghosts
— which food is eaten
How many states does Pacman have?
I don’t even know how to begin counting but a lot.



```
# Actor behaviors
behavior EgoBehavior(speed):
    try:
        do FollowLaneBehavior(speed)
    interrupt when (distance to leadCar) < 10:
        take SetBrakeAction(0.8)

behavior LeadingCarBehavior(speed):
    try:
        do FollowLaneBehavior(speed)
    interrupt when (distance to obstacle) < 10:
        take SetBrakeAction(0.8)

# Spatial relations
lane = Uniform(*network.lanes)

obstacle = Trash on lane.centerline

leadCar = Car following roadDirection from obstacle for Range(-50, -30),
        with behavior LeadingCarBehavior(10)

ego = Car following roadDirection from leadCar for Range(-15, -10),
        with behavior EgoBehavior(10)
```

Number of states: number of valid input images.

- So the first problem is that we can't use tabular methods. We need methods based on approximation.
- That is, instead of updating

  $J_0, TJ_0, T^2J_0, \ldots$

  we need to deal with **low-dimensional approximations** of these quantities.
- But there's another problem: in the real world, it's a problem to even write down the MDP.
- Think of the self-driving car example: there are so many things to model... (other cars, pedestrians, birds, streets, etc).
- Instead, we need methods that learn the MDP as they go along.
- Next, we will discuss how to solve the *second* problem. After we do that, we'll go back and talk about the first problem.