The Basic RL Setup

- Couldn’t be simpler:

  — you are at a state

  — you take an action

  — you observe a reward

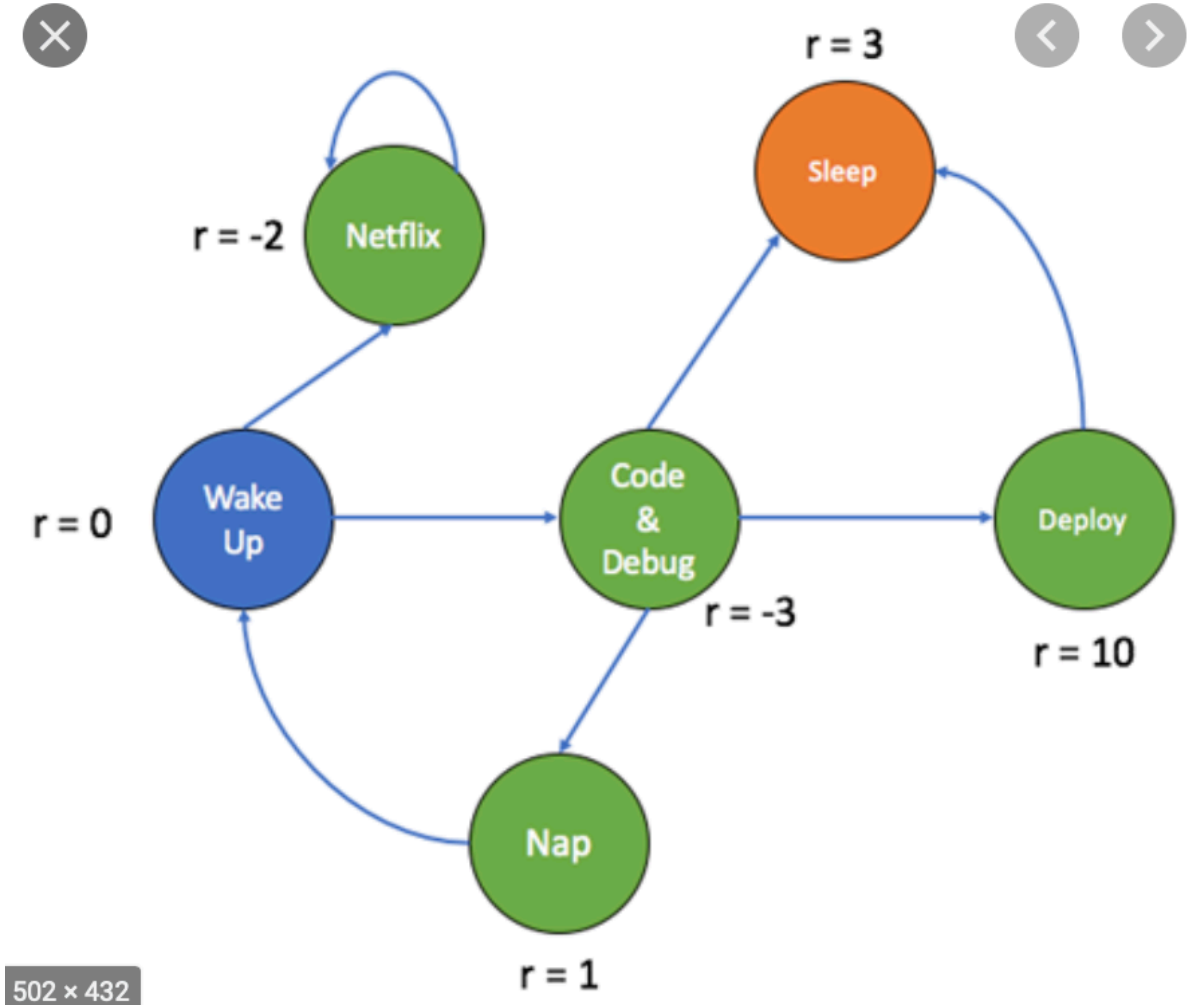
  — you transition to a new state
- Key assumption: given the current state, current action, your next state is independent of the past history.
- The big problem: what are the best actions to take?
- Let’s introduce notation for all of this.

- 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 $s_{k-1}, a_{k-1}, s_{k-2}, a_{k-2}, \ldots$
- 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

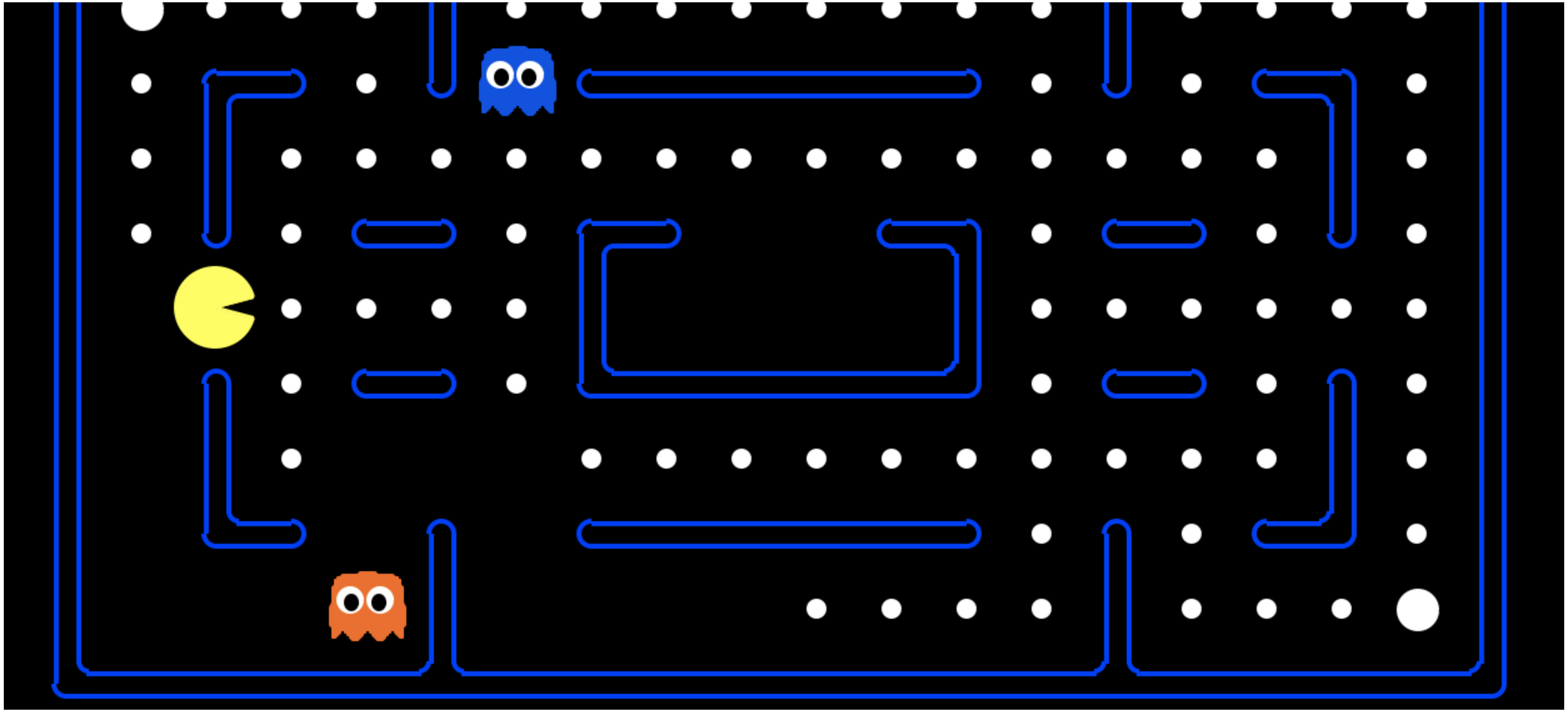
s_4 = nap, r_4 = 1

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:

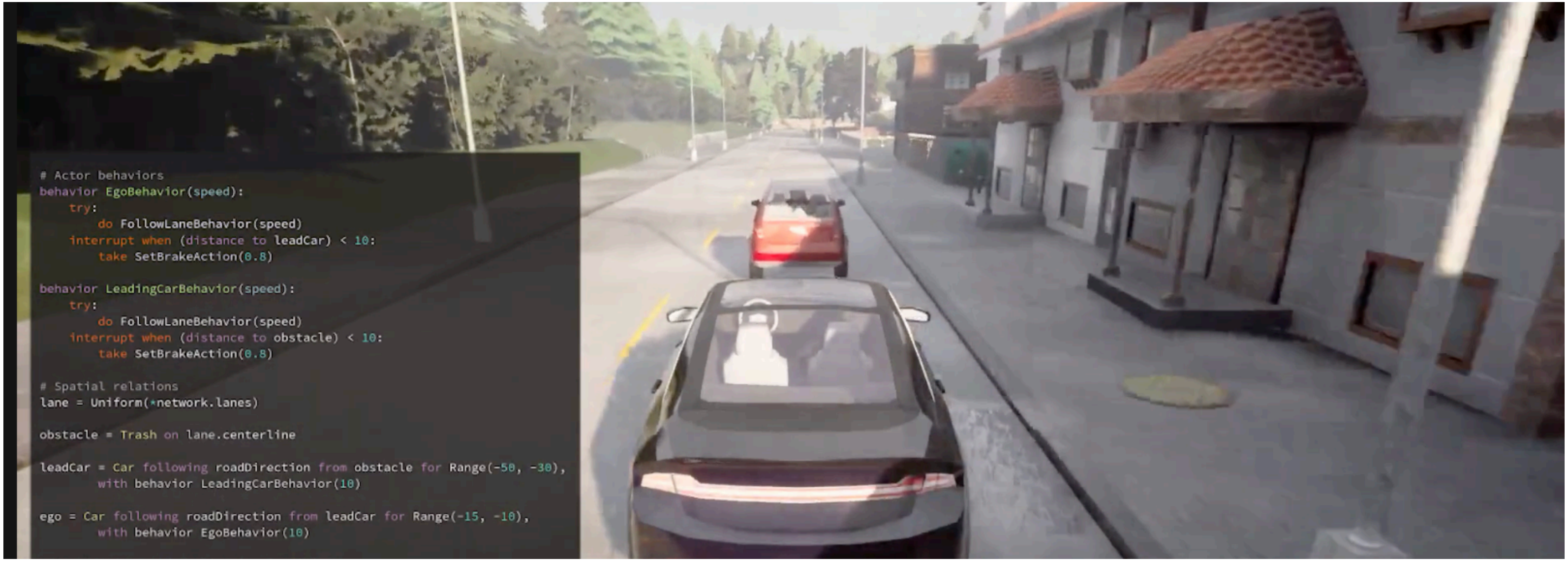
State = the entire board over the past TWO time steps

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.

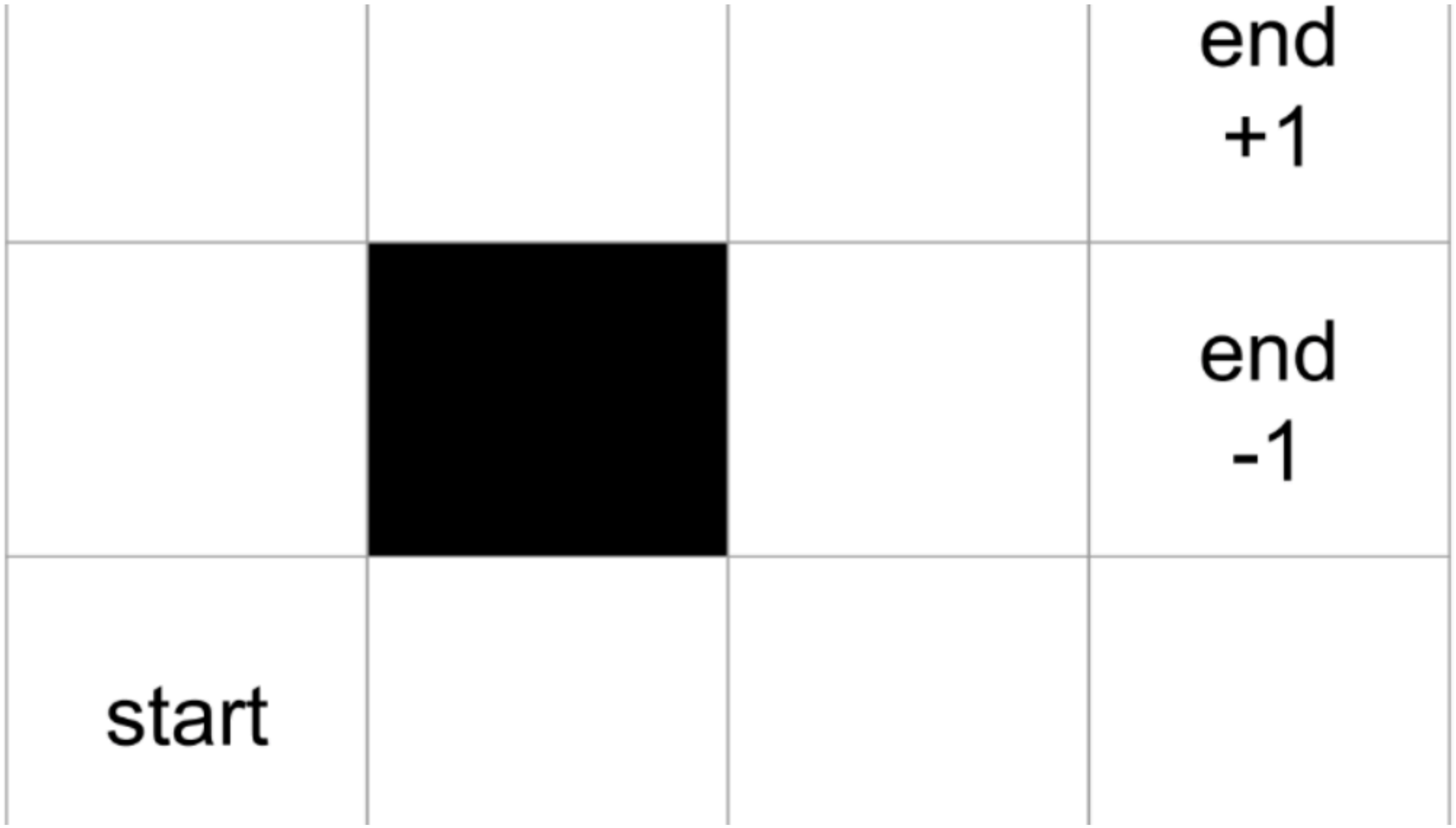
- So we are trying to maximize the reward-to-go form state s:

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

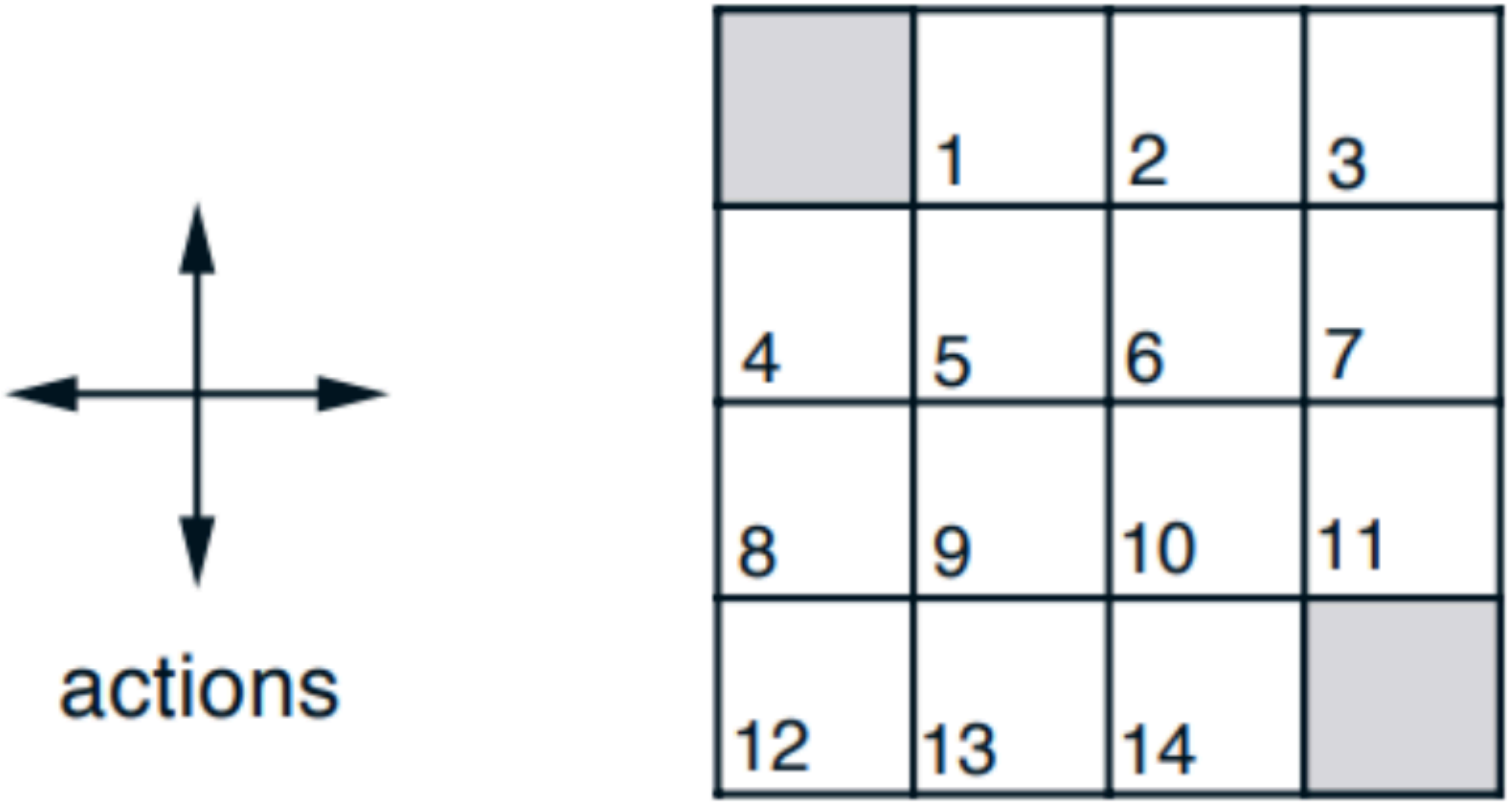
$$J^*(s) = \max_{\pi \in \Pi_{\text{det}}} J_\pi(s).$$

- 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!



$$R_t = -1$$

on all transitions

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 $P(s' \mid s, a)$ 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:

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

- 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 $r(s', a')$

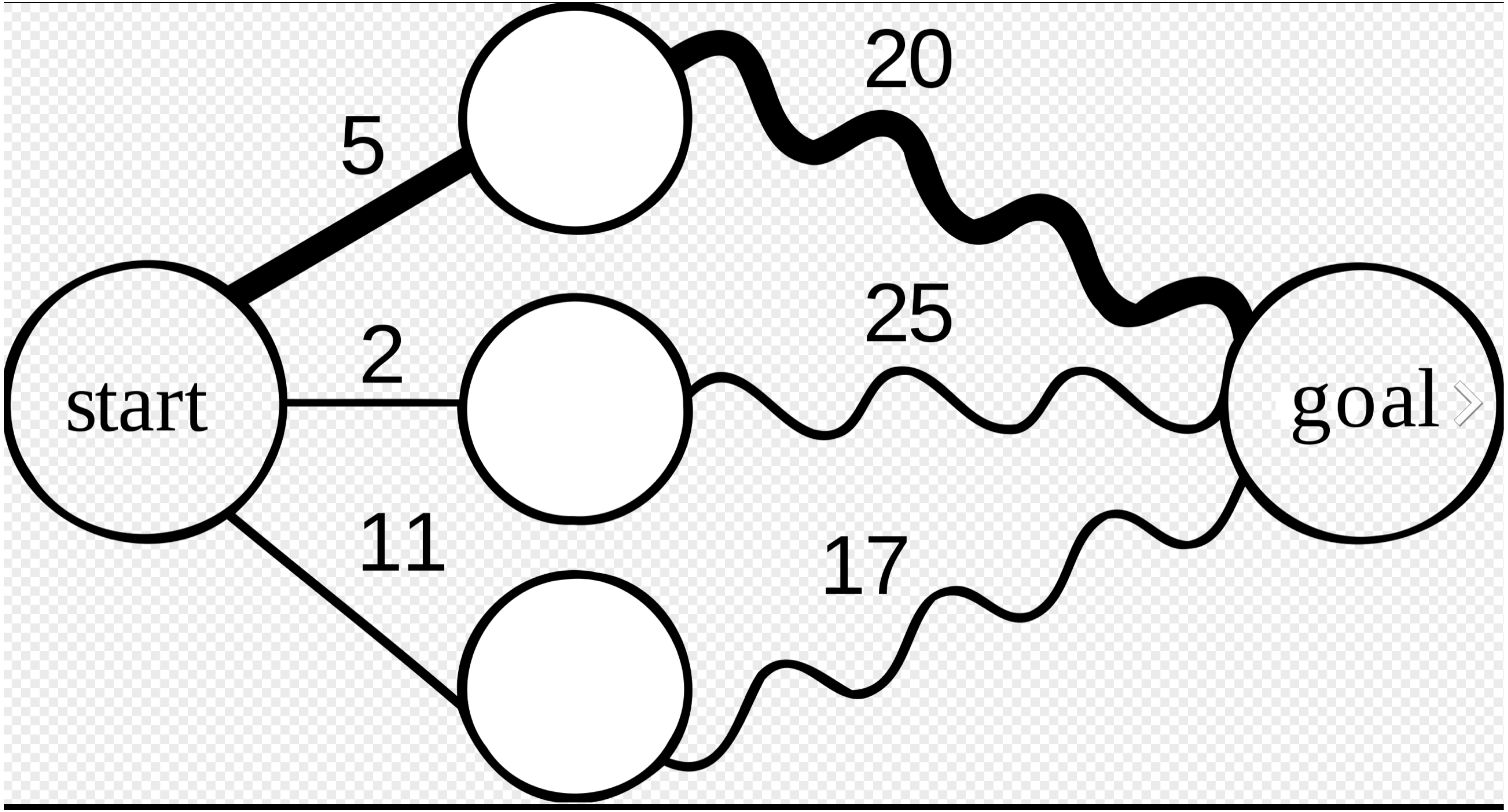
[The distribution of $r_1$depends on s', a', but conditional on these it is independent of s, a]

— 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?
- $a_1^*(s') \in \arg\max_{a \in A} r(s', a) + \gamma \sum_{s''} P(s'' \,|\, s', a) J(s'')$.
- ...and the forward-looking reward you expect to obtain is just $\max_{a \in A} r(s', a) + \gamma \sum_{s''} P(s'' \,|\, s', a) J(s'')$.

  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

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

  and expect to obtain reward

$$J_0^*(s) = \max_a \left[r(s, a) + \gamma \sum_{s'} P(s' \mid s, a) J_1^*(s') \right]$$

- 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:

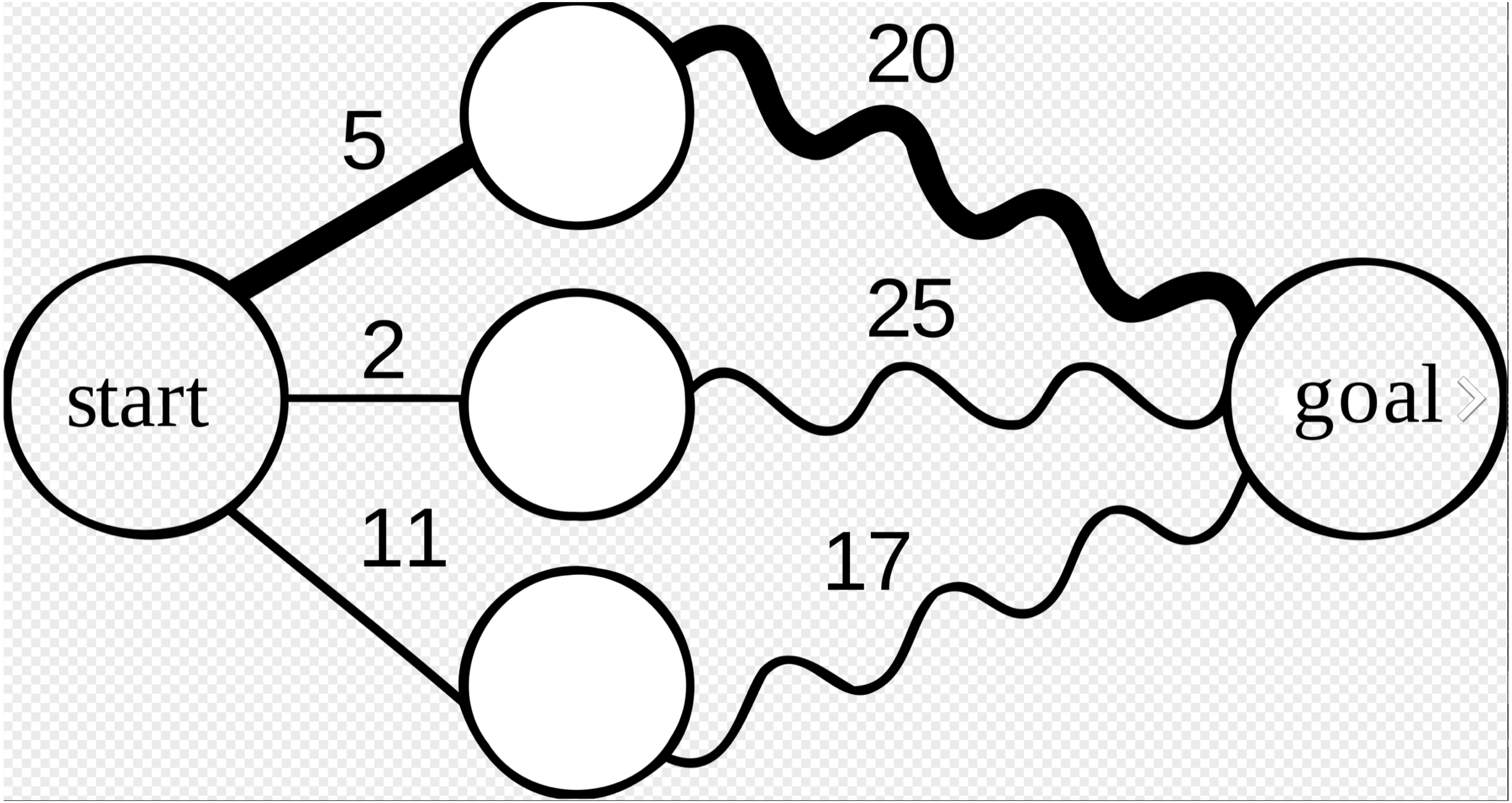
$$a_1^*(s) \in \arg\max_{a \in A} r(s, a) + \gamma \sum_{s'} P(s' | s, a) J(s')$$

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

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

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

- 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

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

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

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

- 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

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

  and define the cost to go is

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

- 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:

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

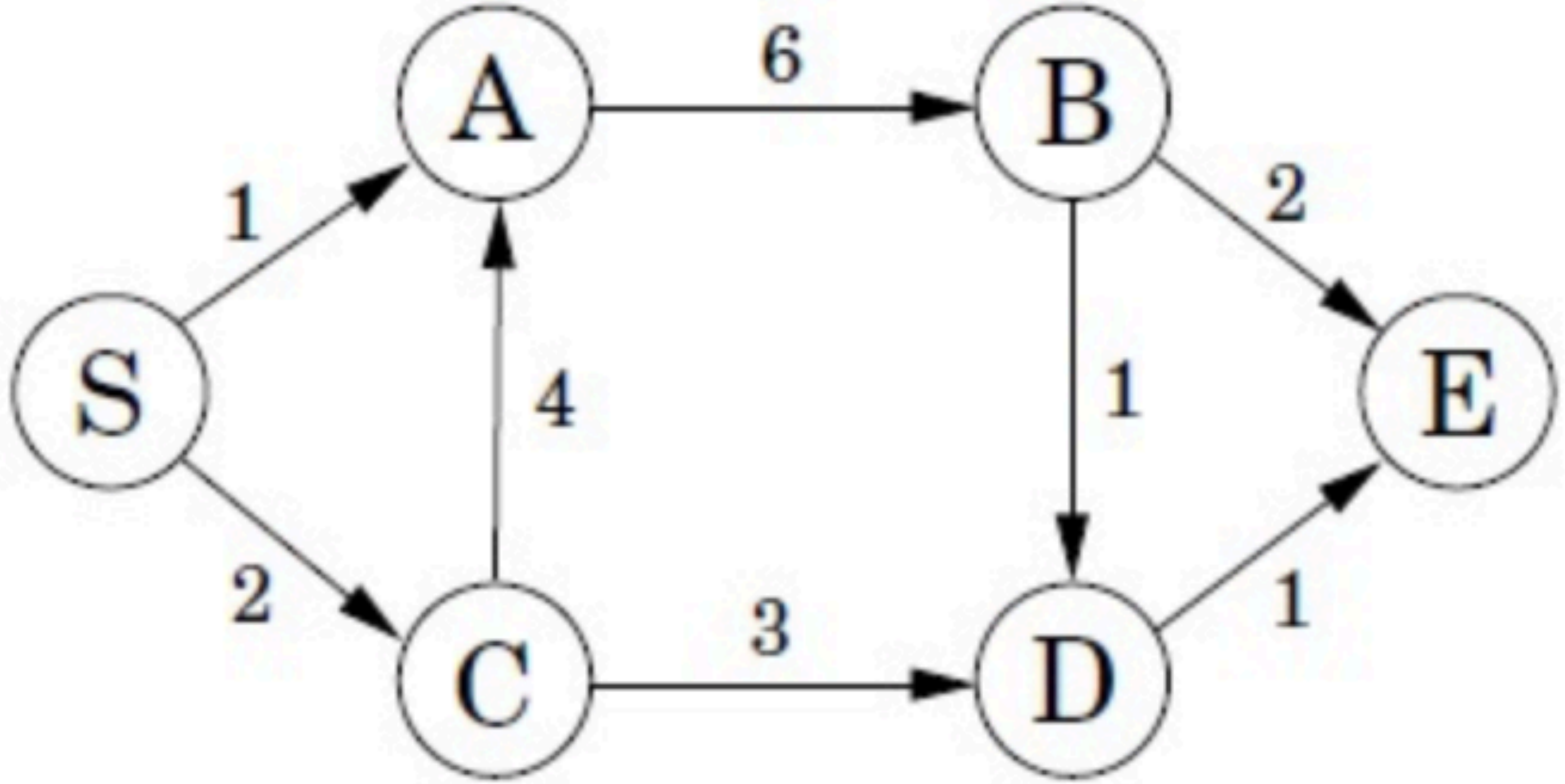
$$a^*_{K-1}(s) = \arg\max_a r(s,a) + \gamma \sum_{s'} P(s' \,|\, s,a) J(s')$$

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

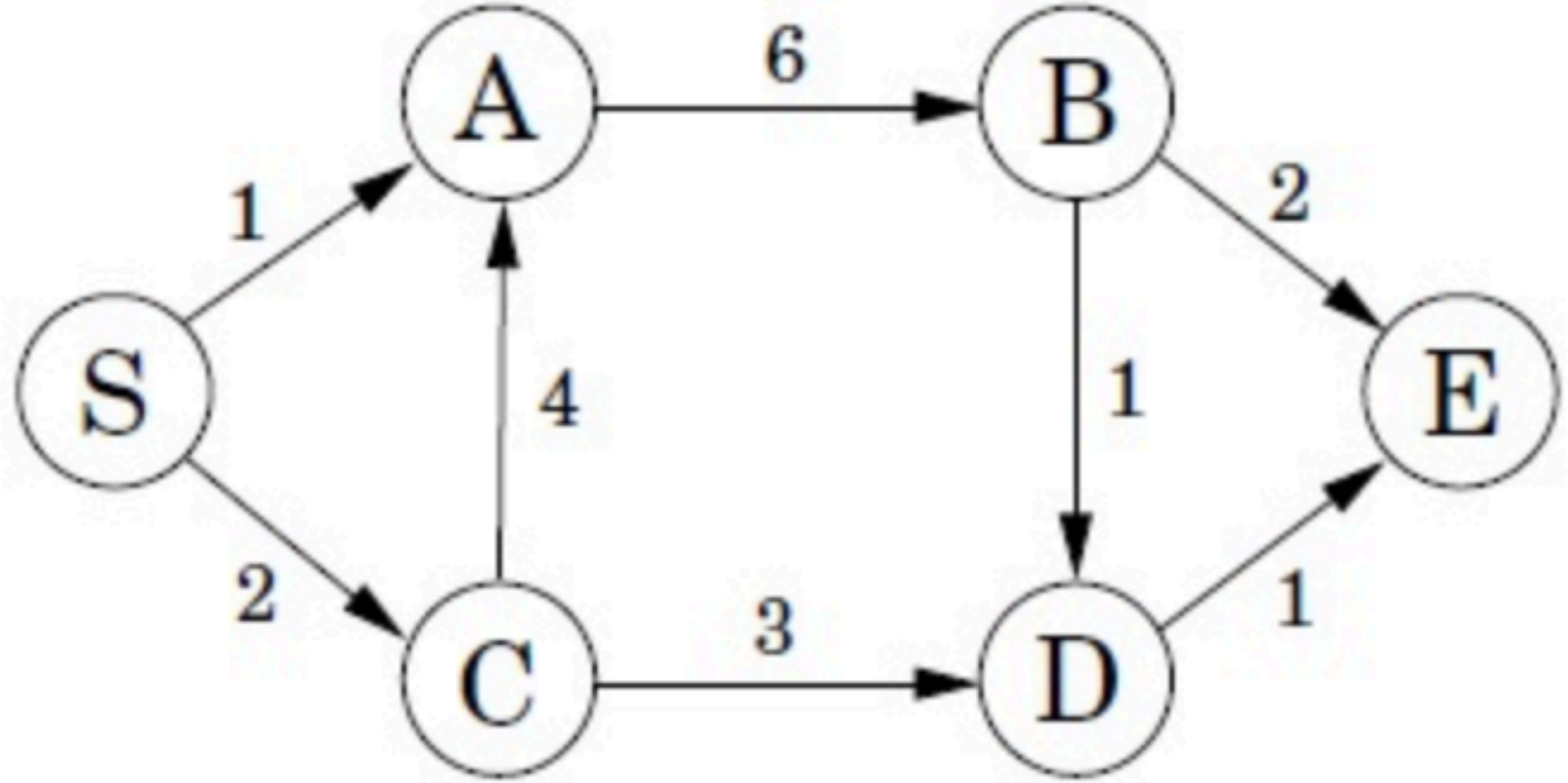
$$J^*_k(s) = \max_a r(s,a) + \gamma \sum_{s'} P(s' \,|\, s,a) J^*_{k+1}(s')$$

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

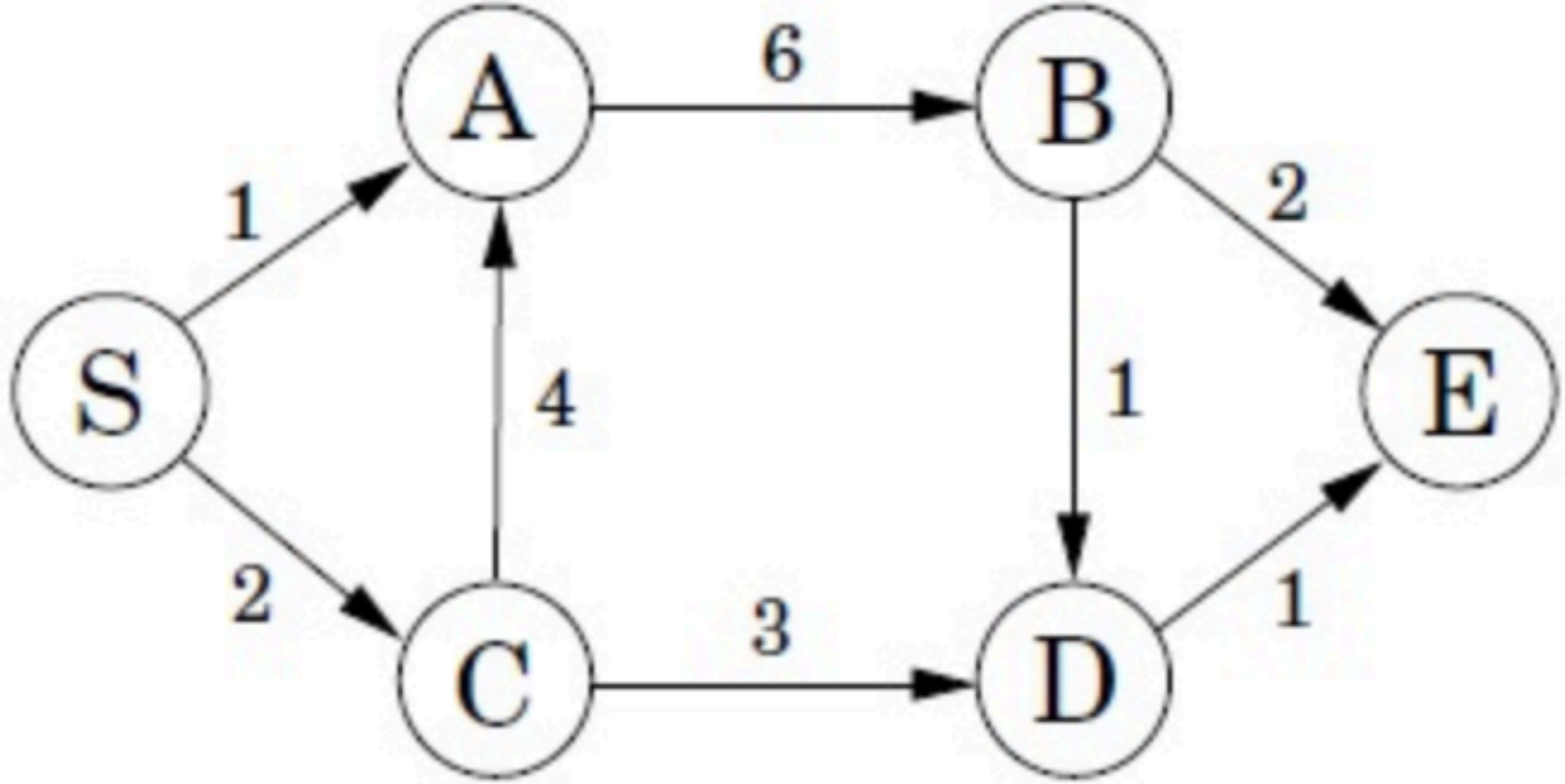
- For obvious reasons, this is called “backwards induction.”



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

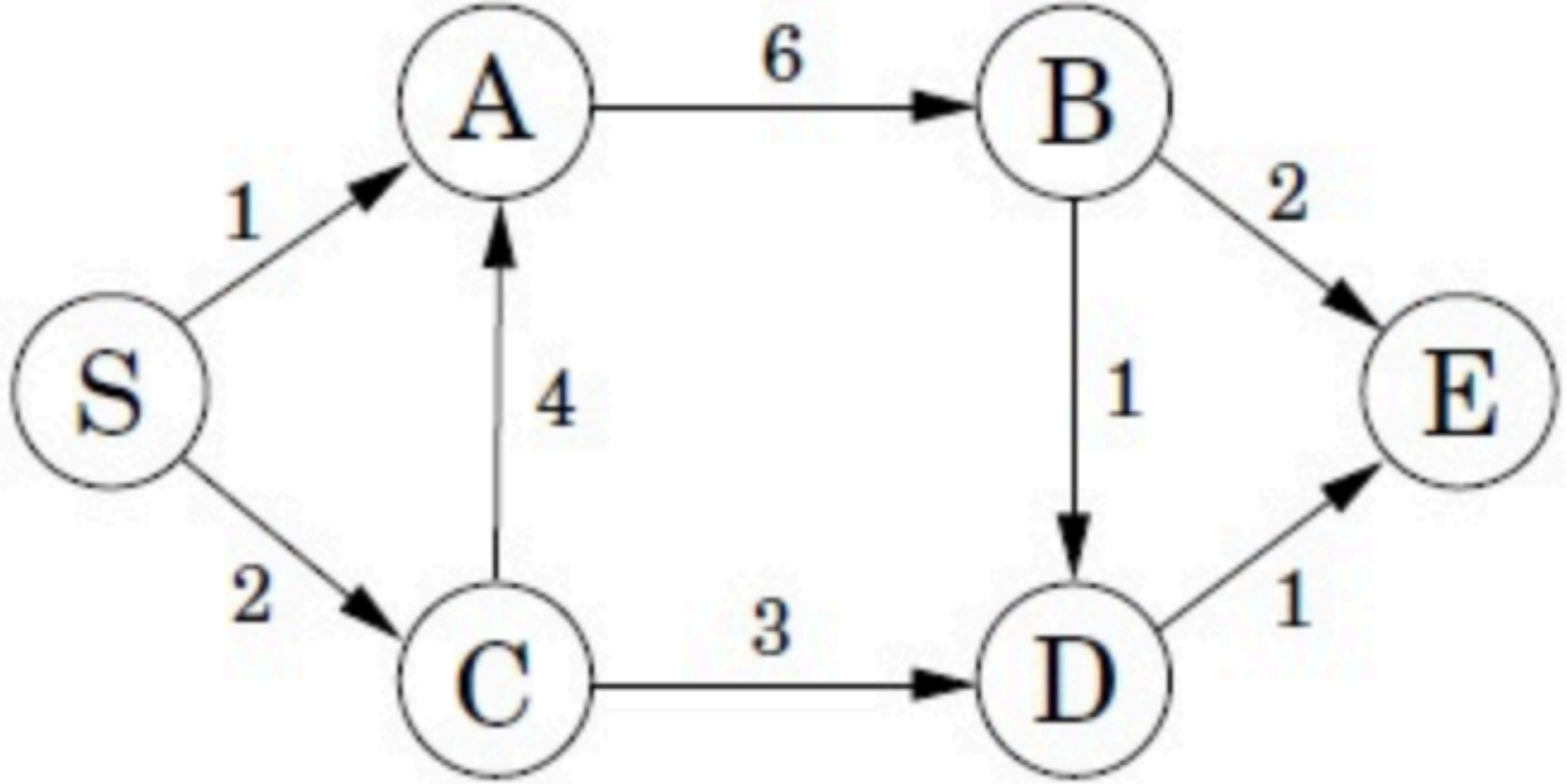


J4(E) = 0, J4(B) = -2, J4(D) = -1 and J4(S) = J4(A)=J4(C)=-INFINITY



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

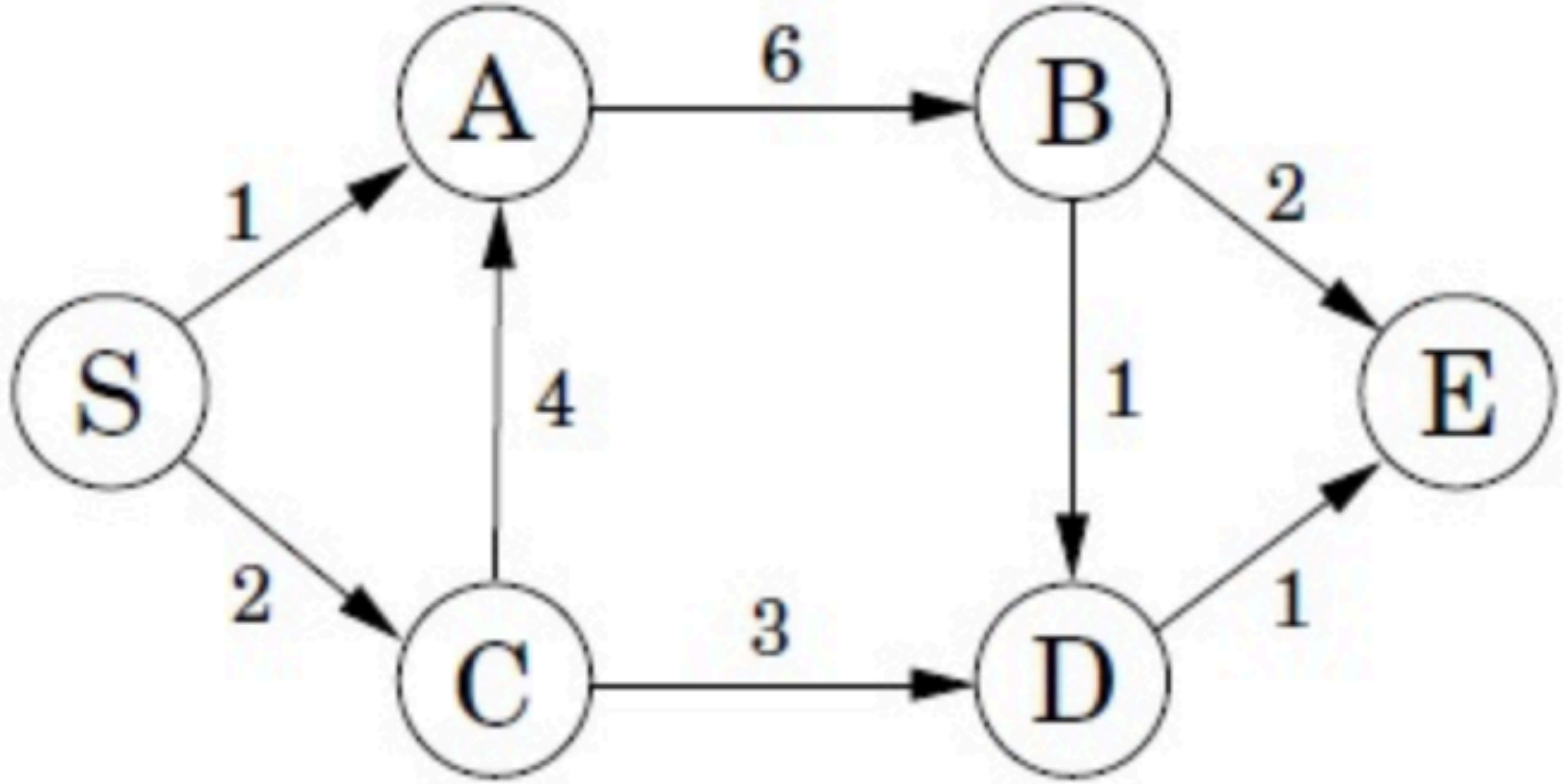
J3(E)=0, J3(B)=-2,J3(D)=-1, J3(C)=-4, J3(A)=-8 and J3(S)=-INFINITY



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

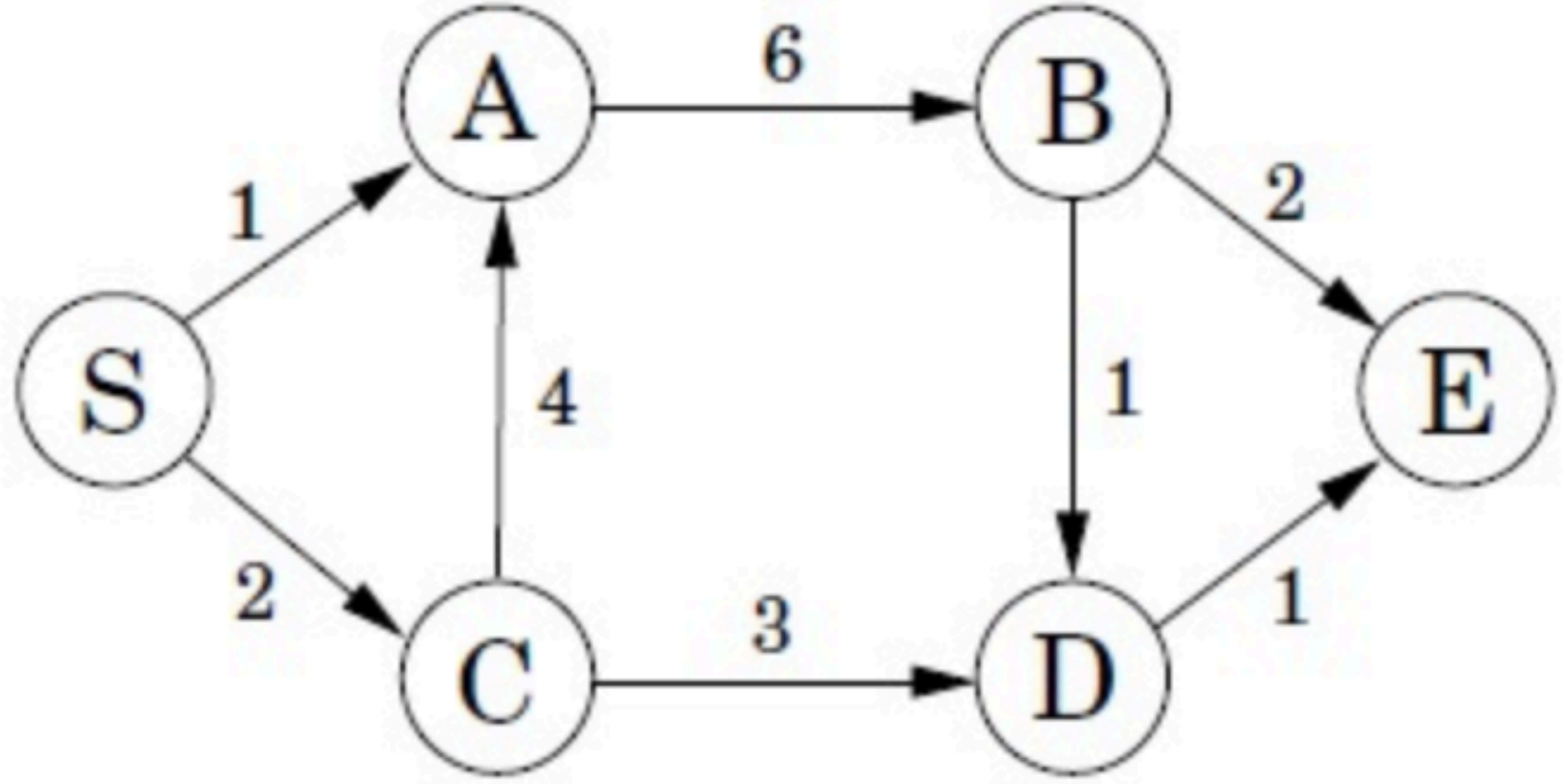
J3(E)=0, J3(B)=-2,J3(D)=-1, J3(C)=-4, J3(A)=-8 and J3(S)=INFINITY

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

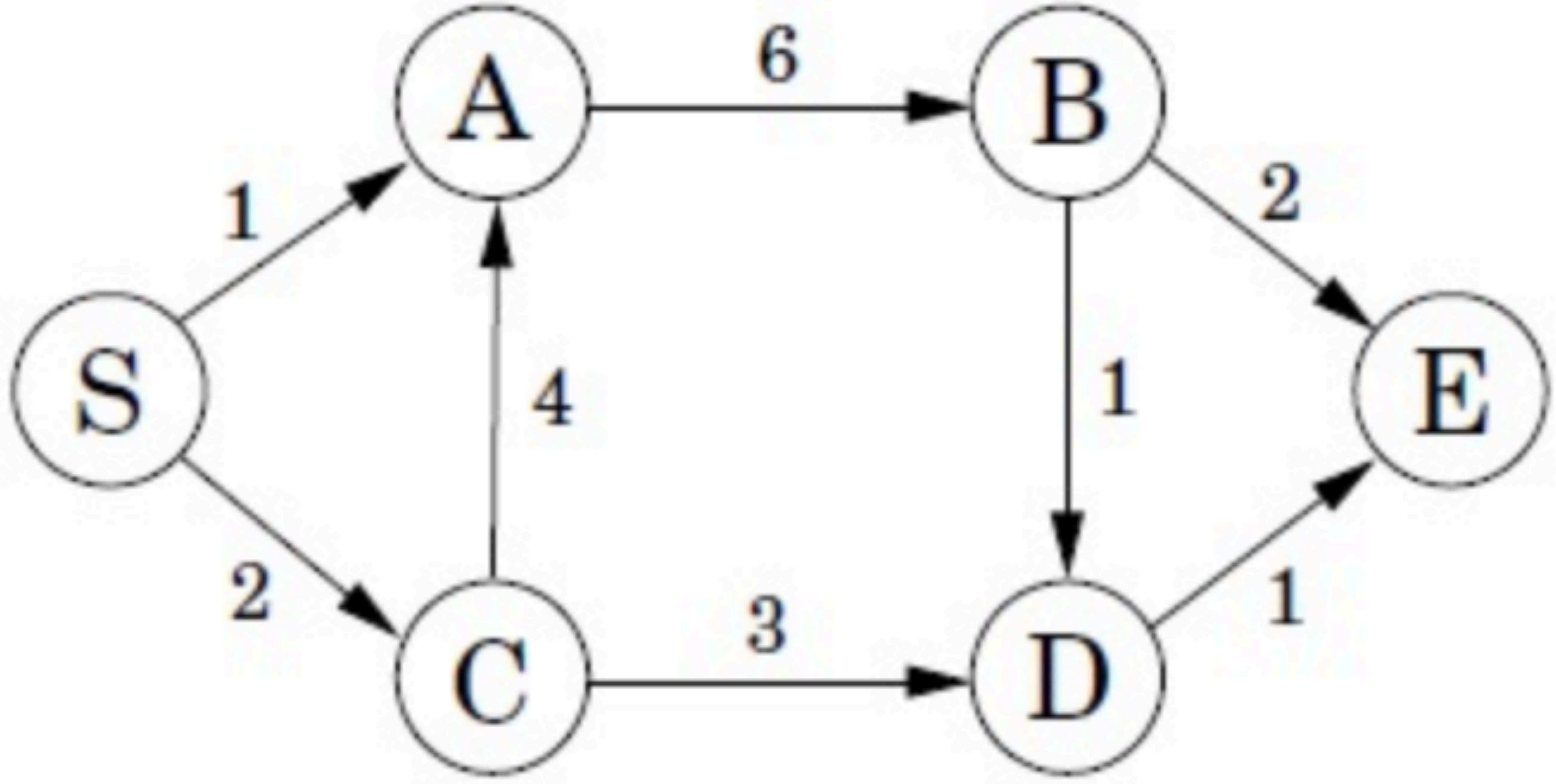
J0(E)=0,J0(B)=-2,J0(D)=-1, J0(C)=-4, J0(A)=-8, J0(S)=-6

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



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

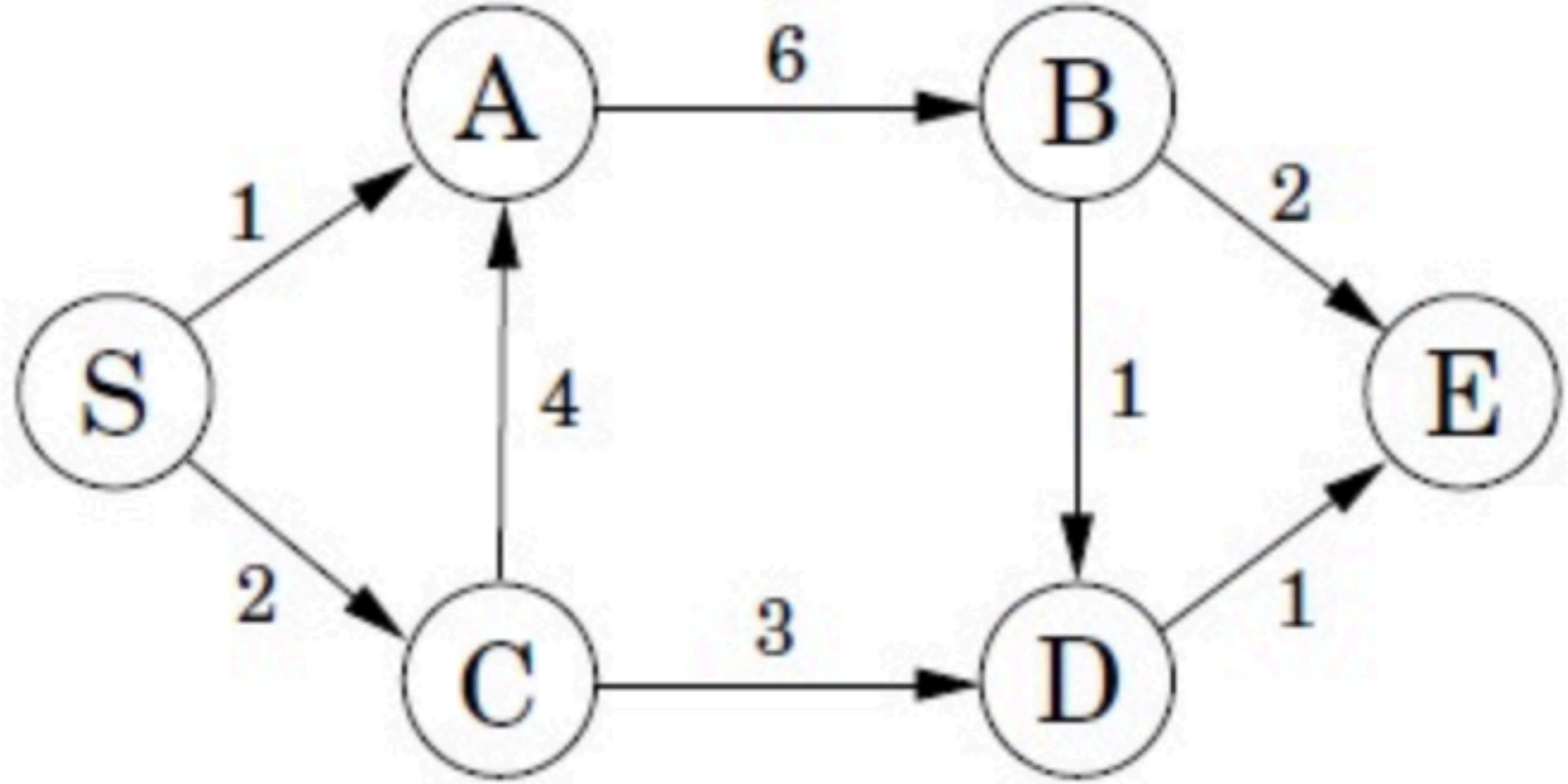
J4(E)=0, J4(B)=-2, J4(D)=-1.



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.

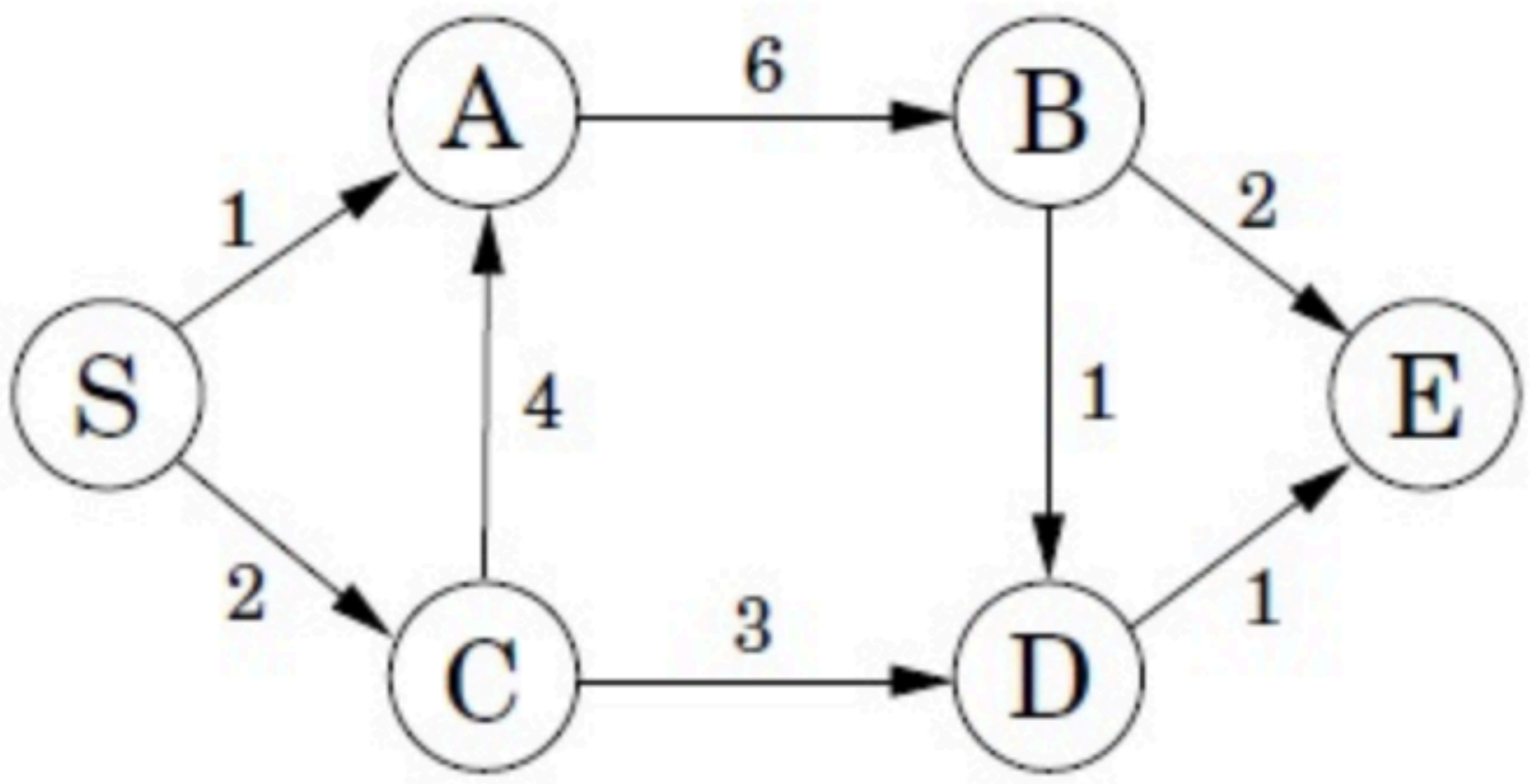


Now let's redo everything with discount factor equal 1/4!

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 $J_0^* = T^2 J$.
- Interpretation: $T^2 J$ is the optimal reward in a two stage dynamic programming problem with terminal reward J.

- Let's go back to:

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

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

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

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

- Let's write this as

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

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

- In particular, $J^*_0 = T^K J$.
- Interpretation: $T^K J$ is the optimal reward in a K stage dynamical programming problem with terminal reward J.
- In other words,

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

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

- 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, $J^*_0 = T^K J$.
- 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}?

- Summary:

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

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

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

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

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

  $J^*_k = TJ^*_{k+1}$.
- In particular, $J^*_0 = T^K J$.
- 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.

- Consider a two-state MDP with states {“watch tv”, “go outside”} (states 1 and 2, respectively).
- When you are outside, your only action is “stay outside” and you get a reward of 2.
- When you are watching tv, you have two actions: “keep watching”, reward of 1, and “switch”, reward of -4
- Let’s set the discount factor to 0.9.
- Game ends after five rounds, with a terminal cost of zero.
- We solve this by first thinking about what you would do at round four.

  If you are watching TV:

  $$J_4^*(1) = \max\{1 + \gamma 0, -4 + \gamma 0\} = 1$$

  If you are outside:

  $$J_4^*(2) = 2 + \gamma 0 = 2$$

- $(TV)(s) = \max_a r(s,a) + \gamma \sum_{s'} P(s' \mid s,a) V(s')$
- $T\begin{pmatrix} V_1 \\ V_2 \end{pmatrix} = \begin{pmatrix} \max\{1 + \gamma V_1, \ -4 + \gamma V_2\} \\ 2 + \gamma V_2 \end{pmatrix}$
- $J_4^* = \begin{pmatrix} J_4^*(1) \\ J_4^*(2) \end{pmatrix} = T\begin{pmatrix} 0 \\ 0 \end{pmatrix}$

- Now time step 3. If you are watching TV:

  $J_3^*(1) = \max\{1 + \gamma J_4^*(1), -4 + \gamma J_4^*(2)\} = \max\{1 + \gamma 1, -4 + \gamma 2\} = 1 + \gamma$

  If you are outside:

  $J_3^*(2) = 2 + \gamma J_4^*(2) = 2 + \gamma 2 = 2(1 + \gamma)$

- $T\begin{pmatrix} V_1 \\ V_2 \end{pmatrix} = \begin{pmatrix} \max\{1 + \gamma V_1, \ -4 + \gamma V_2\} \\ 2 + \gamma V_2 \end{pmatrix}$

- $J_3^* = \begin{pmatrix} J_3^*(1) \\ J_3^*(2) \end{pmatrix} = T\begin{pmatrix} J_4^*(1) \\ J_4^*(2) \end{pmatrix} = TJ_4^*$

- Likewise, $J_2^* = TJ_3^*$