인공지능 실습 Chapter 4
Markov Decision Process

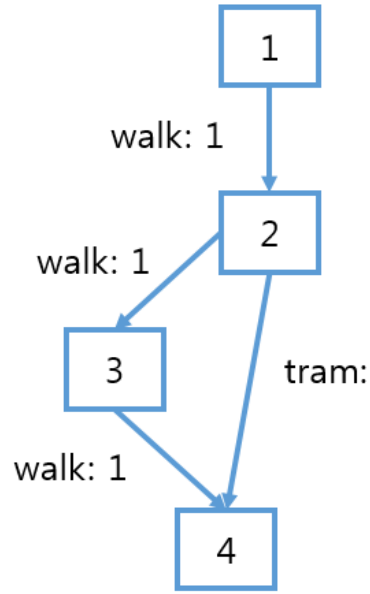
포항공과대학교 컴퓨터공학과

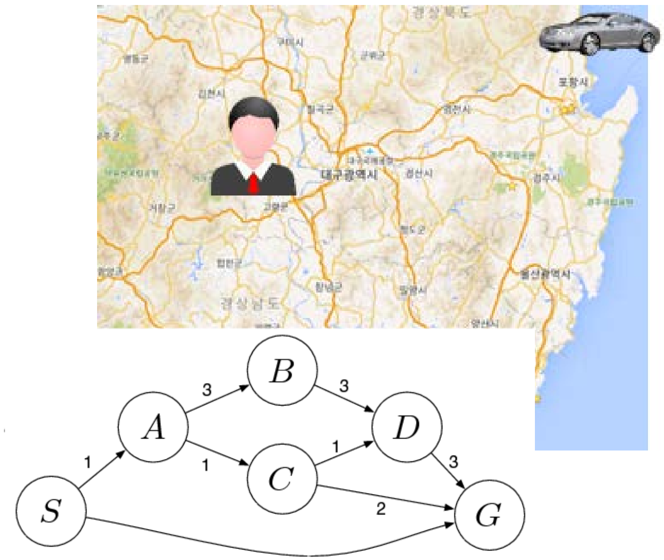
정보통신연구소
POSTECH INFORMATION RESEARCH LABORATORIES

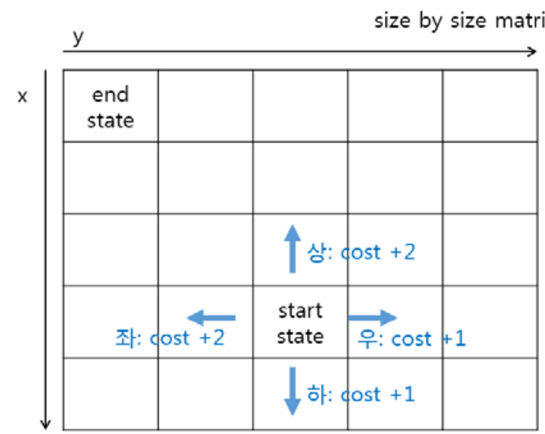
POSTECH
POHANG UNIVERSITY OF SCIENCE AND TECHNOLOGY

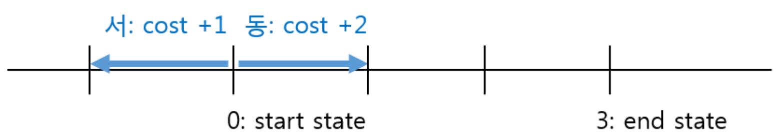
Search Problem

- 대중교통 문제
- '고령'가기 문제
- Line Search 문제
- Grid Search 문제
- Graph Search 문제





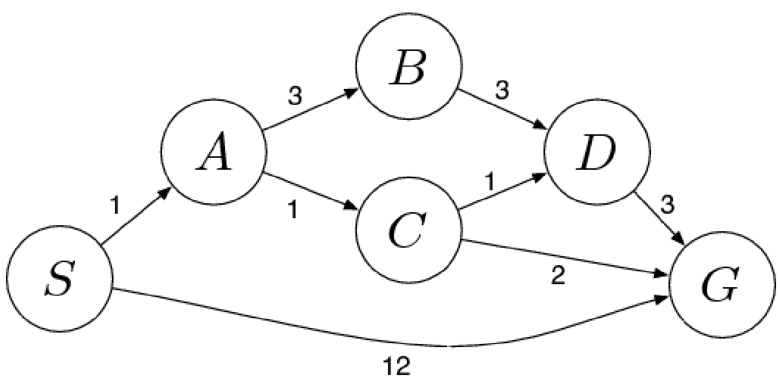




- **Search Problem?**
- Start state에서부터 end state까지의 <u>최소 cost 경로</u>를 찾는 것

Search Problem

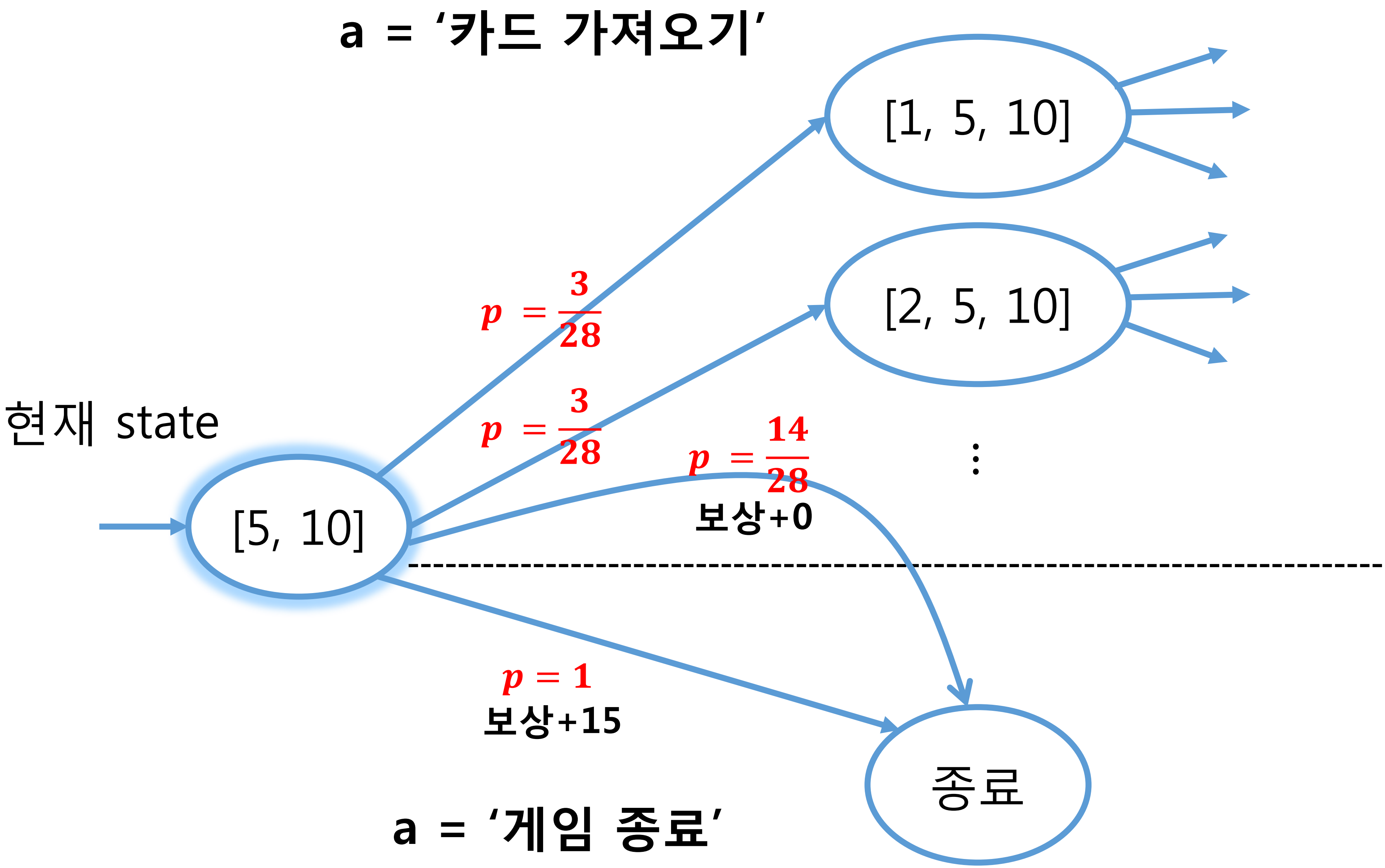
- $States$: state들의 집합
- s_{start} : 시작 state
- $Actions(s)$: state s에서 가능한 action들의 집합
- $Succ(s, a)$: state s에서 action a를 수행하면 도착하는 state
- $Cost(s, a)$: state s에서 action a를 수행하는데 필요한 cost
- $IsEnd(s)$: state s가 end state인지 여부



카드 게임

- **카드 게임 규칙**
- 1부터 10까지 숫자가 적힌 카드가 3장씩 총 30장 존재
- 한 턴에 '카드 가져오기', '게임 끝내기' 중 하나를 수행
- 게임을 끝내면 현재 가지고 있는 카드 숫자의 합만큼 점수를 획득
- 카드를 가져온 후, 현재 가지고 있는 카드 숫자의 합이 20보다 크면 0점을 획득하고 게임 종료

카드 게임


a = '카드 가져오기'
[1, 5, 10]
[2, 5, 10]
$p = \frac{3}{28}$
$p = \frac{3}{28}$
$p = \frac{14}{28}$
보상+0
현재 state
[5, 10]
$p = 1$
보상+15
종료
a = '게임 종료'

Markov Decision Process

- $States$: state들의 집합
- s_{start} : 시작 state
- $Actions(s)$: state s에서 가능한 action들의 집합
- ~~$Succ(s,a)$: state s에서 action a를 수행하면 도착하는 state~~
- ~~$Cost(s,a)$: state s에서 action a를 수행하는데 필요한 cost~~
- $IsEnd(s)$: state s가 end state인지 여부

Markov Decision Process

- $States$: state들의 집합
- s_{start} : 시작 state
- $Actions(s)$: state s에서 가능한 action들의 집합
- ~~$Succ(s, a)$: state s에서 action a를 수행하면 도착하는 state~~
- ~~$Cost(s, a)$: state s에서 action a를 수행하는데 필요한 cost~~
- $IsEnd(s)$: state s가 end state인지 여부

- $T(s, a, s')$: state s에서 action a를 수행하여 s′에 도착할 확률
- $Reward(s, a, s')$: 위의 transition에 따른 보상

Markov Decision Process

- $States$: [[], [1], [2], …, [10, 10], '종료']
- s_{start} : []
- $Actions(s)$: ['카드 가져오기', '게임 종료']
- $IsEnd(s)$: s == '종료'
- $T(s, a, s')$:
  - T([], '게임 종료', '종료') = 1
  - T([], '카드 가져오기', [1]) = 3/30
  - T([], '카드 가져오기', [2]) = 3/30
- $Reward(s, a, s')$:
  - T([1,2,3], '게임 종료', '종료') = 6

MDP를 어떻게 풀까?

- **Policy**
  - 현재 state에서 어떤 action을 선택할 것인가?
  - $\pi([1,2])$ = '카드 가져오기' (or '게임 종료')
- **Utility**
  - 주어진 policy를 사용했을 때 얻게되는 임의 path에서 Reward의 총합
  - 식넣기
  - [1,2] 카드가져오기 → [1,2,5] 게임종료 : Utility = 8
  - [1,2] 카드가져오기 → [1,2,10] 카드가져오기 → [1,2,9,10] : Utility = 0
- **Value**
  - 주어진 policy를 사용했을 때 현재 state에서의 utility 기대값
  - $V_\pi([1,2]) = 13$

MDP를 어떻게 풀까?

1. Policy iteration
2. Value iteration

Value Iteration

- 목표 : 각 state에서 $V_{opt}(s)$를 계산한다.

- $V_\pi(s)$와 $V_{opt}(s)$의 차이?
    - $V_\pi(s)$: 현재 state에서의 policy를 따르는 action에 대한 value값을 저장
    - $V_{opt}(s)$: 현재 state에서 가능한 모든 action을 고려하여 최대 value값을 저장

Value Iteration

Algorithm: value iteration [Bellman, 1957]

- Initialize $V_{\text{opt}}^{(0)}(s) \leftarrow 0$ for all states s.
- For iteration $t = 1, \ldots, t_{\text{VI}}$:
  - For each state s:

$$V_{\text{opt}}^{(t)}(s) \leftarrow \max_{a \in \text{Actions}(s)} \underbrace{\sum_{s'} T(s,a,s')[\text{Reward}(s,a,s') + \gamma V_{\text{opt}}^{(t-1)}(s')]}_{Q_{opt}^{(t)}(s,a)}$$

- $V_{opt}([10{,}10])$ = max(

$$\underbrace{1[20 + 1 \cdot 0]}_{a=\text{'게임 종료'}}, \quad \underbrace{\frac{3 + 3 + \cdots + 1}{28}[0 + 1 \cdot 0]}_{a=\text{'카드 가져오기'}} \;)$$

= 20

- $\pi_{opt}([10, 10])$ = '게임 종료'

Value & Q-Value

- **Value & Q-Value**
- $V(s)$: 현재 state s로부터 utility 기대값
- $Q(s,a)$: 현재 state s에서 action a를 취했을 때의 utility 기대값

Value & Q-Value

- **Value & Q-Value**
- $V(s)$: 현재 state s로부터 utility 기대값
- $Q(s,a)$: 현재 state s에서 action a를 취했을 때의 utility 기대값

- **Policy Iteration**
- $V_\pi(s)$: 현재 state s로부터 policy π를 따를 경우 utility 기대값

$$V_\pi(s) = \begin{cases} 0 & \text{If IsEnd}(s) \\ Q_\pi(s, \pi(s)) & \text{otherwise.} \end{cases}$$

- $Q_\pi(s,a)$: 현재 state s에서 action a를 취한 후, 이후 state부터 policy π 를 따를 경우 utility 기대값

$$Q_\pi(s,a) = \sum_{s'} T(s,a,s')[\text{Reward}(s,a,s') + \gamma V_\pi(s')]$$

Value & Q-Value

- **Value & Q-Value**
- $V(s)$: 현재 state s로부터 utility 기대값
- $Q(s,a)$: 현재 state s에서 action a를 취했을 때의 utility 기대값

- **Value Iteration**
- $V_{opt}(s)$: 현재 state s로부터, optimal policy를 따를 경우 utility 기대값

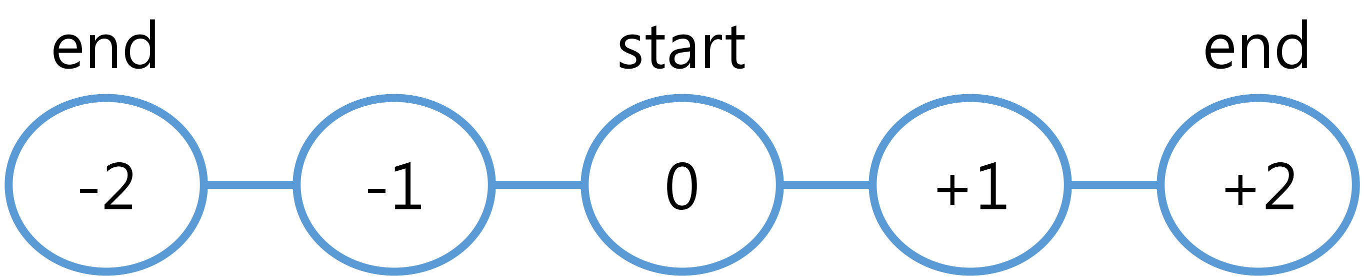
$$V_{\mathrm{opt}}(s) = \begin{cases} 0 & \text{If IsEnd}(s) \\ \max_{a \in \mathrm{Actions}(s)} Q_{\mathrm{opt}}(s,a) & \text{otherwise.} \end{cases}$$

- $Q_{opt}(s,a)$: 현재 state s에서 action a를 취한 후, 이후 state부터 optimal policy를 따를 경우 utility 기대값

$$Q_{\mathrm{opt}}(s,a) = \sum_{s'} T(s,a,s')[\mathrm{Reward}(s,a,s') + \gamma V_{\mathrm{opt}}(s')]$$

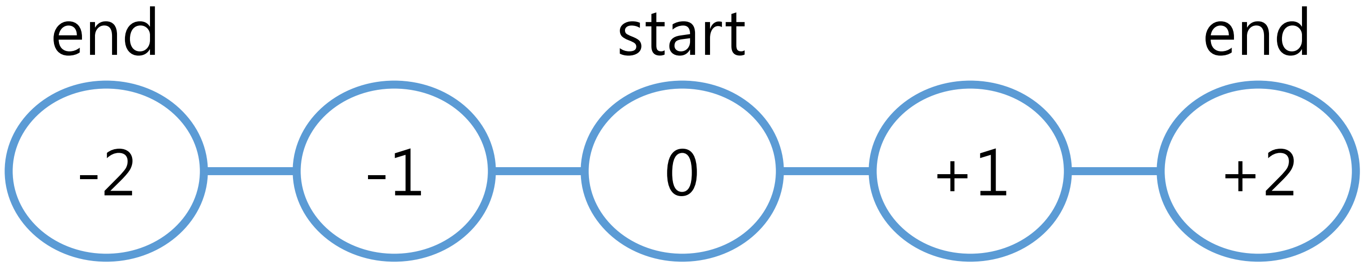
A. Value Iteration 예제

- state = $\{-2, -1, 0, +1, +2\}$
- action = {‘Left’, ‘Right’}
  - ‘Left’ 선택시, 80% 확률로 왼쪽, 20% 확률로 오른쪽 이동
  - ‘Right’ 선택시, 70% 확률로 왼쪽, 30% 확률로 오른쪽 이동
- +2로 이동하는 action을 수행하는 경우 Reward = 100
- −2로 이동하는 action을 수행하는 경우 Reward = 20
- 그 외의 경우, Reward = -5



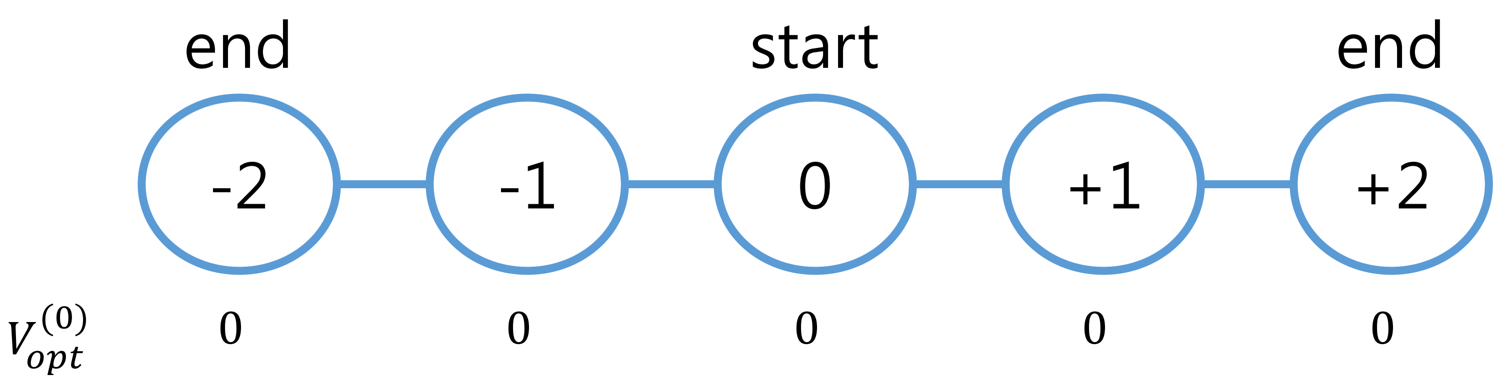
A. Value Iteration 예제

- 각각의 state s에 대해 value iteration 알고리즘을 적용하여 0, 1, 2번 iteration 수행한 이후의 $V_{opt}(s)$ 값을 계산해보자.
- Terminal states에는 optimal policy가 존재하지 않으며, value 값은 0이다.



A. Value Iteration 예제

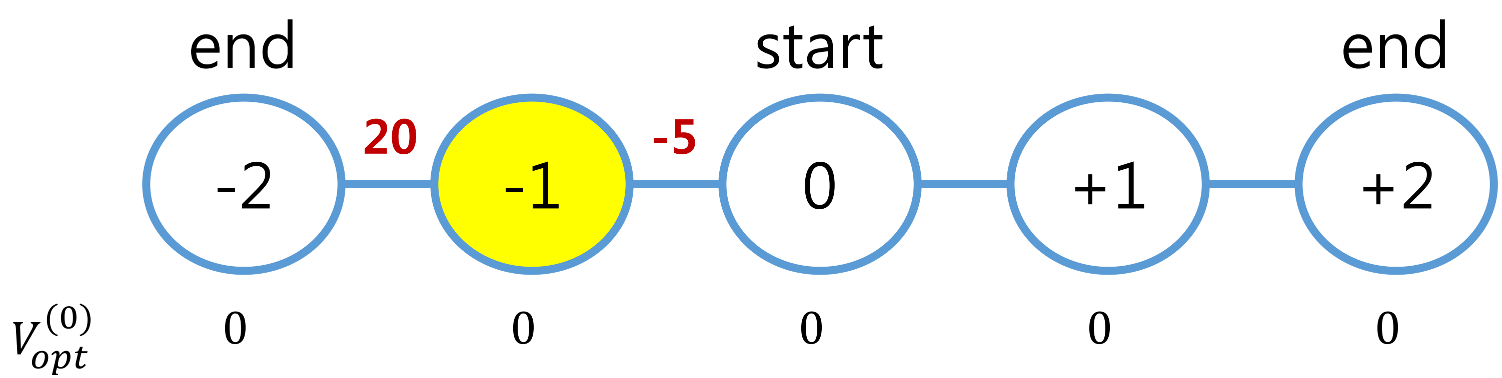
- 첫 번째 iteration
- $V_{opt}^{(1)}(-2) = 0$
- $V_{opt}^{(1)}(-1) = \max\big(0.8(20+0) + 0.2(-5+0), 0.7(20+0) + 0.3(-5+0)\big) = 15$
- $V_{opt}^{(1)}(0) = \max\big(0.8(-5+0) + 0.2(-5+0), 0.7(-5+0) + 0.3(-5+0)\big) = -5$
- $V_{opt}^{(1)}(+1) = \max\big(0.8(-5+0) + 0.2(100+0), 0.7(-5+0) + 0.3(100+0)\big) = 26.5$
- $V_{opt}^{(1)}(+2) = 0$



A. Value Iteration 예제

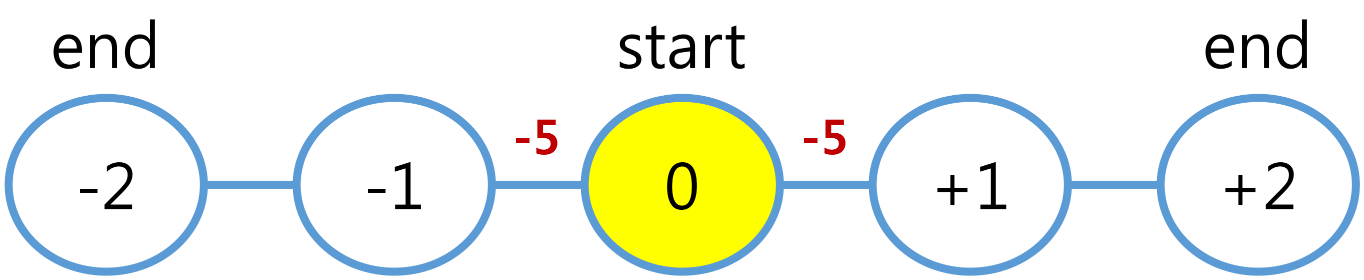
- 첫 번째 iteration

- $V_{opt}^{(1)}(-2) = 0$
- $V_{opt}^{(1)}(-1) = \max\Big(\underbrace{0.8(20+0)+0.2(-5+0)}_{a=\text{'Left'}}, \underbrace{0.7(20+0)+0.3(-5+0)}_{a=\text{'Right'}}\Big) = 15$
- $V_{opt}^{(1)}(0) = \max\big(0.8(-5+0)+0.2(-5+0), 0.7(-5+0)+0.3(-5+0)\big) = -5$
- $V_{opt}^{(1)}(+1) = \max\big(0.8(-5+0)+0.2(100+0), 0.7(-5+0)+0.3(100+0)\big) = 26.5$
- $V_{opt}^{(1)}(+2) = 0$



A. Value Iteration 예제

- 첫 번째 iteration
- $V_{opt}^{(1)}(-2) = 0$
- $V_{opt}^{(1)}(-1) = \max\big(0.8(20+0) + 0.2(-5+0), 0.7(20+0) + 0.3(-5+0)\big) = 15$
- $V_{opt}^{(1)}(0) = \max\big(\underbrace{0.8(-5+0) + 0.2(-5+0)}_{\text{a='Left'}}, \underbrace{0.7(-5+0) + 0.3(-5+0)}_{\text{a='Right'}}\big) = -5$
- $V_{opt}^{(1)}(+1) = \max\big(0.8(-5+0) + 0.2(100+0), 0.7(-5+0) + 0.3(100+0)\big) = 26.5$
- $V_{opt}^{(1)}(+2) = 0$

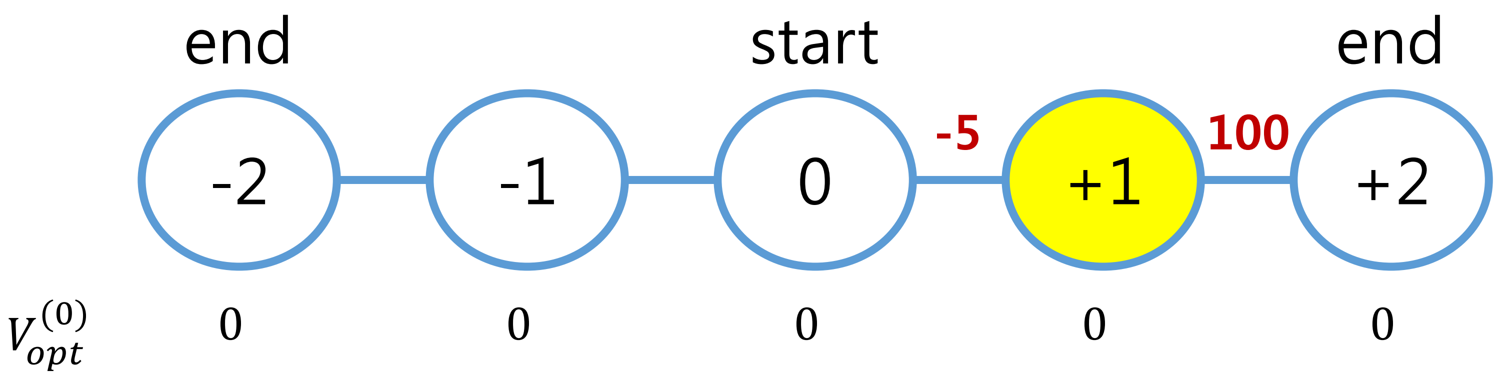


$V_{opt}^{(0)}$	0	0	0	0	0

A. Value Iteration 예제

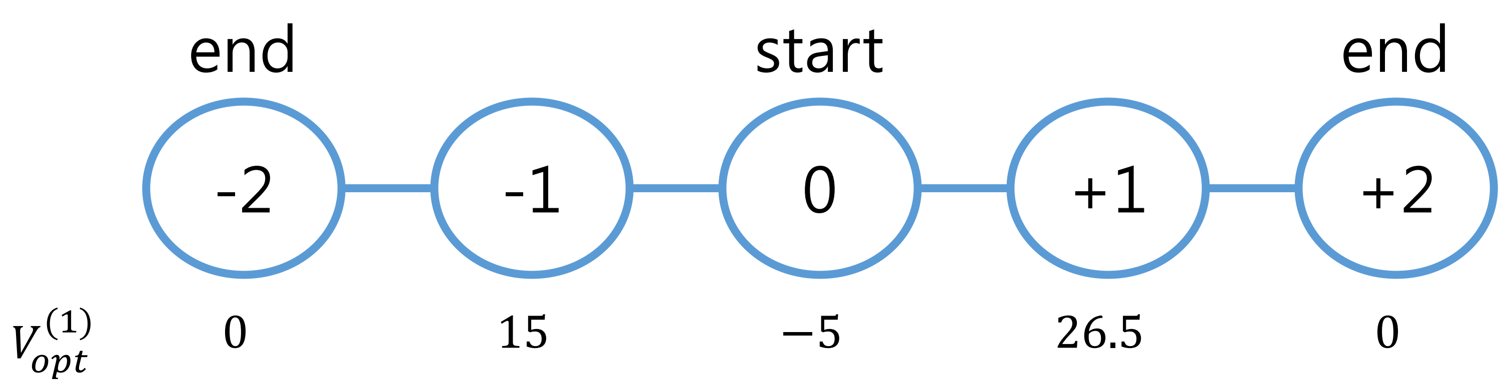
- 첫 번째 iteration

- $V_{opt}^{(1)}(-2) = 0$
- $V_{opt}^{(1)}(-1) = \max\big(0.8(20+0)+0.2(-5+0), 0.7(20+0)+0.3(-5+0)\big) = 15$
- $V_{opt}^{(1)}(0) = \max\big(0.8(-5+0)+0.2(-5+0), 0.7(-5+0)+0.3(-5+0)\big) = -5$
- $V_{opt}^{(1)}(+1) = \max\big(\underbrace{0.8(-5+0)+0.2(100+0)}_{a=\text{'Left'}}, \underbrace{0.7(-5+0)+0.3(100+0)}_{a=\text{'Right'}}\big) = 26.5$
- $V_{opt}^{(1)}(+2) = 0$



A. Value Iteration 예제

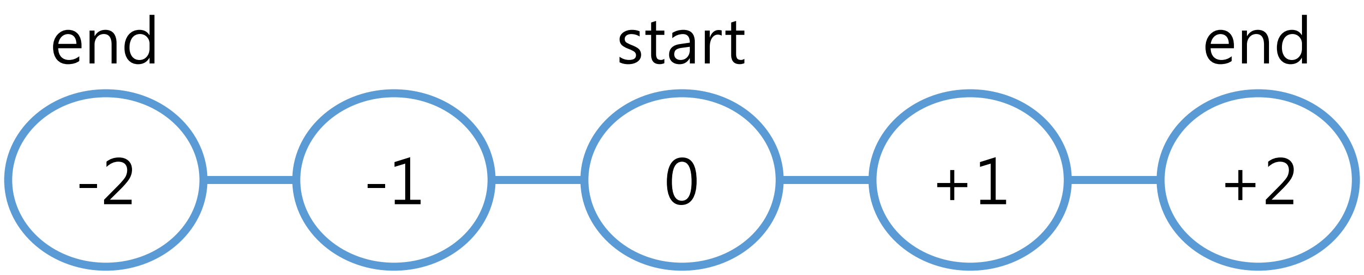
- 두 번째 iteration
- $V_{opt}^{(2)}(-2) = 0$
- $V_{opt}^{(2)}(-1) = \max\big(0.8(20+0) + 0.2(-5-5), 0.7(20+0) + 0.3(-5-5)\big) = 14$
- $V_{opt}^{(2)}(0) = \max\big(0.8(-5+15) + 0.2(-5+26.5), 0.7(-5+15) + 0.3(-5+26.5)\big) = 13.45$
- $V_{opt}^{(2)}(+1) = \max\big(0.8(-5-5) + 0.2(100+0), 0.7(-5-5) + 0.3(100+0)\big) = 23$
- $V_{opt}^{(2)}(+2) = 0$



A. Value Iteration 예제

- V_{opt} 값으로부터 π_{opt} 값을 구해보자

- $\pi_{opt}(-1)\ =\ ?$
- $\pi_{opt}(0)\ =\ ?$
- $\pi_{opt}(+1) = ?$

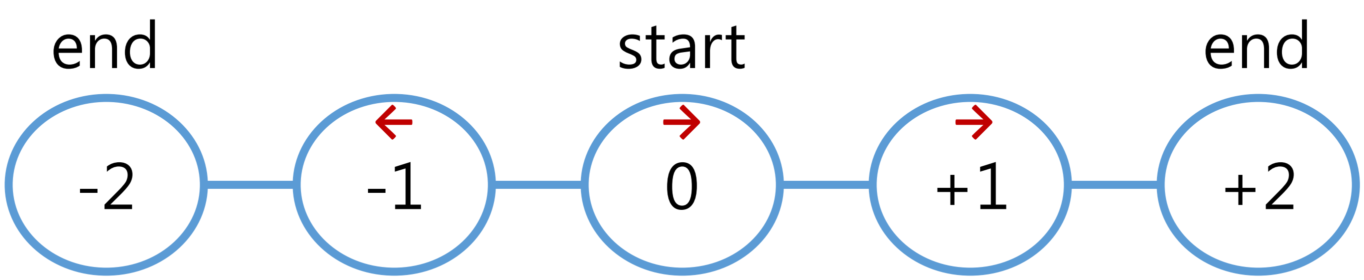


$V_{opt}^{(2)}$	0	14	13.45	23	0

A. Value Iteration 예제

- V_{opt} 값으로부터 π_{opt} 값을 구해보자

- $\pi_{opt}(-1)$ = 'Left'
- $\pi_{opt}(0)$ = 'Right'
- $\pi_{opt}(+1)$ = 'Right'



$V_{opt}^{(2)}$	0	14	13.45	23	0

B. MDP 변형

- Discount factor γ 가 1인 MDP만 풀 수 있는 MDP solver가 존재한다고 가정하자. 이 MDP solver를 이용하여 γ 가 1보다 작은 MDP를 풀기 위해서는 어떻게 해야할까?

- Hint: $\gamma = 1$이지만 $\gamma < 1$인 MDP와 관계식이 같아지도록 transition probability와 reward를 변경하여 새로운 MDP를 정의해보자

B. MDP 변형

$$V_{\text{opt}}(s) = \max_{a \in \text{Actions}(s)} \sum_{s' \in \text{States}} T(s,a,s') \left[\text{Reward}(s,a,s') + \gamma V_{\text{opt}}(s')\right]$$

$$V'_{\text{opt}}(s) = \max_{a \in \text{Actions}(s)} \sum_{s' \in \text{States}'} T'(s,a,s') \left[\text{Reward}'(s,a,s') + V'_{\text{opt}}(s')\right]$$

- Terminal state o를 추가하여 새로운 MDP를 정의하자

  - $V'_{\text{opt}}(o) = 0$ (Let o be the terminal state)
  - $T'(s,a,s') = \gamma T(s,a,s')$
  - $T'(s,a,o) = 1 - \gamma$
  - $\text{Reward}'(s,a,s') = \frac{1}{\gamma}\text{Reward}(s,a,s')$
  - $\text{Reward}'(s,a,o) = 0$

B. MDP 변형

- 증명

$$
\begin{aligned}
V'_{\text{opt}}(s) &= \max_{a \in \text{Actions}(s)} \sum_{s' \in \text{States}'} T'(s,a,s') \left[\text{Reward}'(s,a,s') + V'_{\text{opt}}(s')\right] \\
&= \max_{a \in \text{Actions}(s)} \left[\sum_{s' \in \text{States}} T'(s,a,s') \left[\text{Reward}'(s,a,s') + V'_{\text{opt}}(s')\right] \right. \\
&\qquad\qquad \left. + T'(s,a,o) \left[\text{Reward}'(s,a,o) + V'_{\text{opt}}(o)\right] \right] \\
&= \max_{a \in \text{Actions}(s)} \left[\sum_{s' \in \text{States}} \gamma T(s,a,s') \left[\frac{1}{\gamma}\text{Reward}(s,a,s') + V'_{\text{opt}}(s')\right] + (1-\gamma)\cdot 0 \right] \\
&= \max_{a \in \text{Actions}(s)} \sum_{s' \in \text{States}} T(s,a,s') \left[\text{Reward}(s,a,s') + \gamma V'_{\text{opt}}(s')\right]
\end{aligned}
$$

C. Peeking Blackjack 구현

- blackjack.pdf 파일을 읽고 Blackjack 게임을 위한 MDP를 구현해보자
- BlackjackMDP 클래스 내의 succAndProbReward() 함수를 구현

- 다음 카드를 엿볼 수 있는 '**Peek**' action이 추가됨 (Reward = -peekCost)
- 각 State를 아래와 같은 triple로 정의할 것

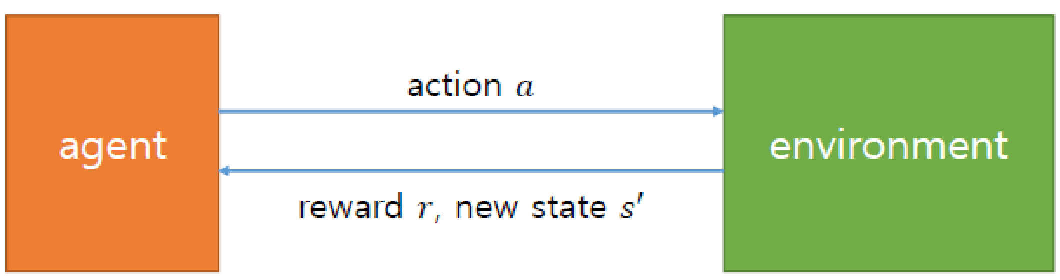
<u>(현재 가지고 있는 카드의 총합, 다음 카드, 남아있는 덱의 카드 수)</u>

- State 예시
  - (1, None, (0, 1, 1)), (2, None, (1, 0, 1)), (3, None, (1, 1, 0))
  - (0, 0, (1, 1, 1)), (0, 1, (1, 1, 1)), (0, 2, (1, 1, 1))
  - (0, None, (1, 1, 1)) $\xrightarrow{\text{Peek}}$ (0, 0, (1, 1, 1)) $\xrightarrow{\text{Take}}$ (1, None, (0, 1, 1))

Reinforcement Learning

- Transition probabilities와 rewards를 모르는 경우에는..?
- Markov Decision Process → **<u>Reinforcement Learning</u>**

Reinforcement learning framework



Q Learning

- MDP에서의 Q-value 정의

$$Q_{\text{opt}}(s,a) = \sum_{s'} T(s,a,s')[\text{Reward}(s,a,s') + \gamma V_{\text{opt}}(s')]$$

- Q-Learning 알고리즘

Algorithm: Q-learning [Watkins/Dayan, 1992]

- On each (s,a,r,s'):

$$\hat{Q}_{\text{opt}}(s,a) \leftarrow (1-\eta)\underbrace{\hat{Q}_{\text{opt}}(s,a)}_{\text{prediction}} + \eta[\underbrace{r + \gamma\hat{V}_{\text{opt}}(s')}_{\text{target}}]$$

- Recall:

$$\hat{V}_{\text{opt}}(s') = \max_{a' \in \text{Actions}(s')} \hat{Q}_{\text{opt}}(s',a')$$

Q Learning

- 어떻게 Q-value를 더욱 정확하게 예측할 수 있을까?
- **Key idea: Linear regression model**

$$\hat{Q}_{\text{opt}}(s,a;\mathbf{w}) = \mathbf{w}\cdot\phi(s,a)$$

- 예시: Blackjack
  - 현재 가지고 있는 카드의 총합
  - 남아 있는 덱의 상태
  - 현재 'Peek'을 수행했는지 여부

Algorithm: Q-learning with function approximation

- On each (s,a,r,s'):

$$\mathbf{w} \leftarrow \mathbf{w} - \eta[\underbrace{\hat{Q}_{\text{opt}}(s,a;\mathbf{w})}_{\text{prediction}} - \underbrace{(r+\gamma\hat{V}_{\text{opt}}(s'))}_{\text{target}}]\phi(s,a)$$

D. Blackjack 학습하기

- Q-Learning algorithm을 구현해보자
- QLearningAlgorithm 클래스의 incorporateFeedback() 함수를 구현

- 가장 먼저, util.py 내의 simulate() 함수를 보고 Reinforcement Learning이 어떤 방식으로 동작하는지 파악
- 아래의 weight 업데이트 식을 incorporateFeedback() 함수 내에 구현

$$\mathbf{w} \leftarrow \mathbf{w} - \eta[\underbrace{\hat{Q}_{\text{opt}}(s, a; \mathbf{w})}_{\text{prediction}} - \underbrace{(r + \gamma \hat{V}_{\text{opt}}(s'))}_{\text{target}}]\phi(s, a)$$

E. Q-Learning의 성능 평가

- 같은 MDP를 Value Iteration과 Q-Learning algorithm을 이용하여 각각 학습한 후, optimal policy가 다른 state의 비율은 얼마인지 계산해보자

- 미리 정의된 두 MDP(smallMDP, largeMDP)에 대해 각각 계산한 후, 결과를 비교
- largeMDP에서 Q-Learning이 더 안좋은 이유는 무엇인가?

F. Blackjack을 위한 FeatureExtractor 구현

- 학습 성능을 높이기 위해서 더 나은 feature extractor를 구현해보자

- blackjackFeatureExtractor() 함수를 구현
- Blackjack에 대한 domain knowledge를 바탕으로 설계된 feature를 추출
  - 각 feature에 대한 설명은 submission.py 주석을 참고
- 구현된 identityFeatureExtractor() 함수를 참고

- **input**: 타겟 (state, action) pair
- **output**: input (state, action) pair에 대한 feature