

04강

인공지능

# 게임트리

컴퓨터과학과 이병래교수

# 학습목차

1 최대최소 탐색

2 몬테카를로 트리 탐색

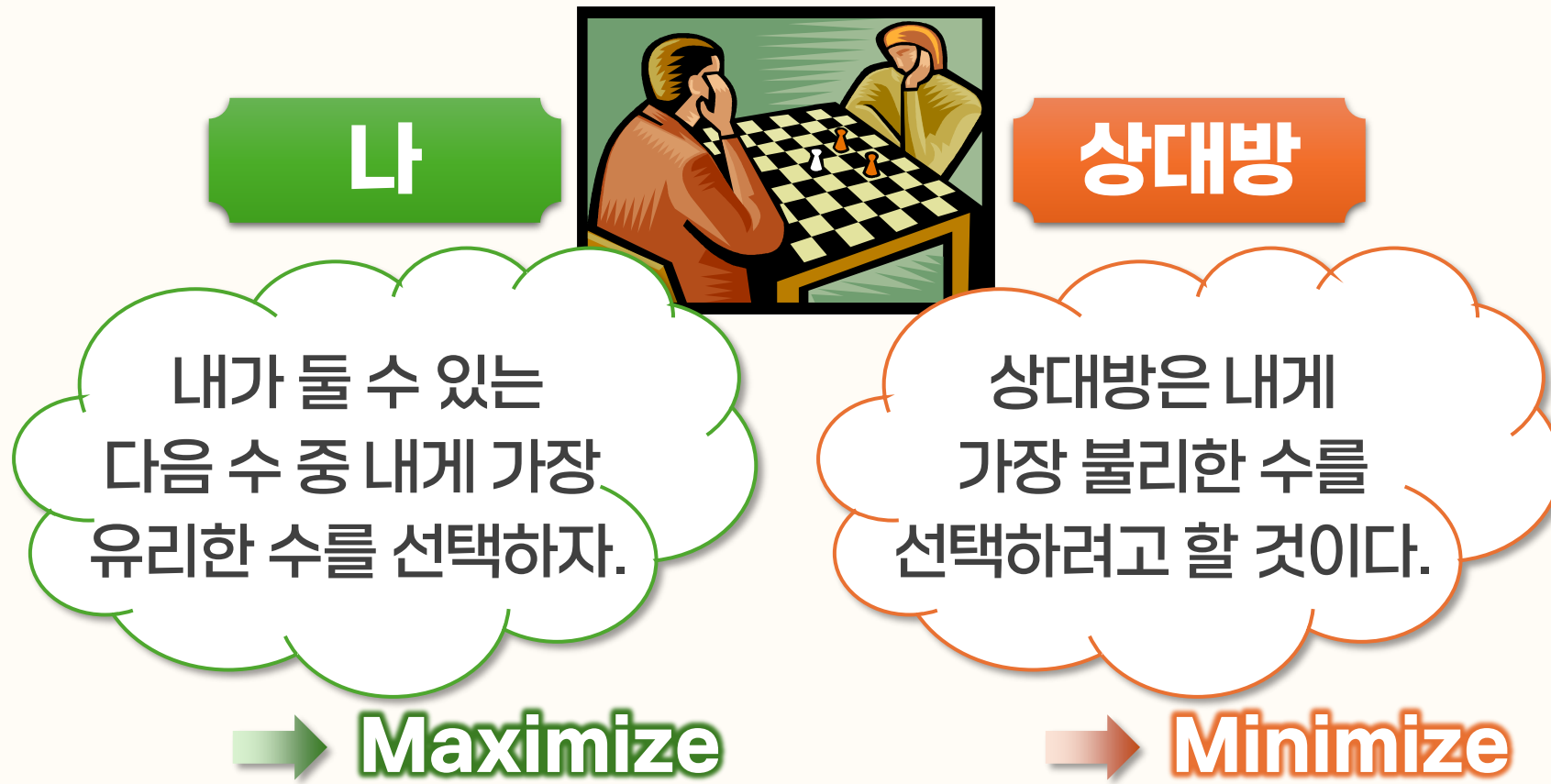




# 최대최소 탐색

# 1. 최대최소 탐색(minimax search)의 개념

- 상대방과의 대결에서 승리하기 위한 게임

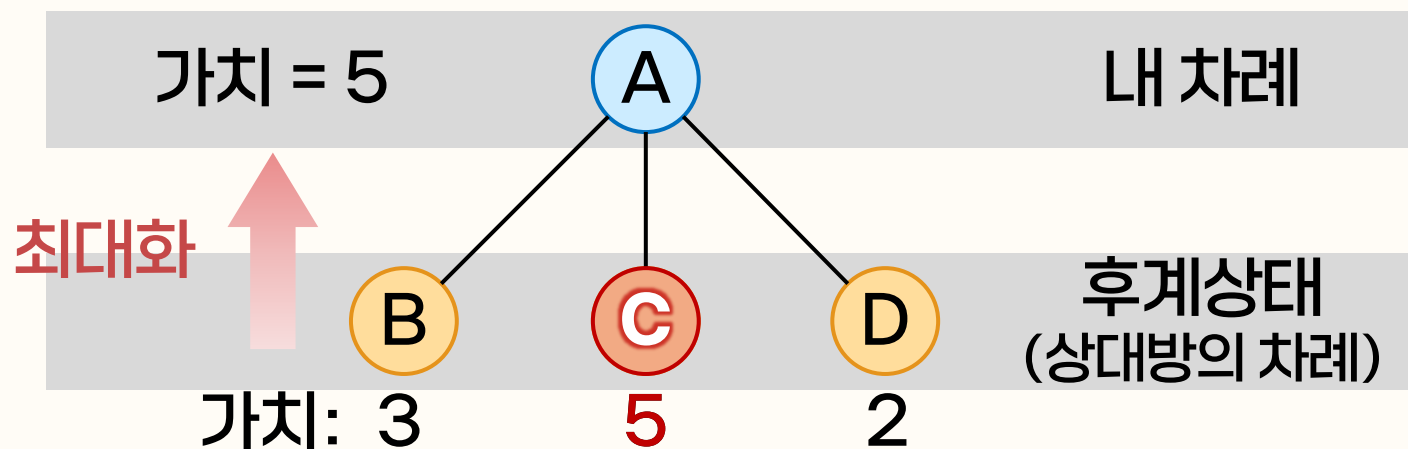


# 1. 최대최소 탐색(minimax search)의 개념

## ❏ 최대화 및 최소화



**최대화**: 내 차례에서는 내가 둘 수 있는 수 중에서 나에게 가장 유리한 수를 선택함

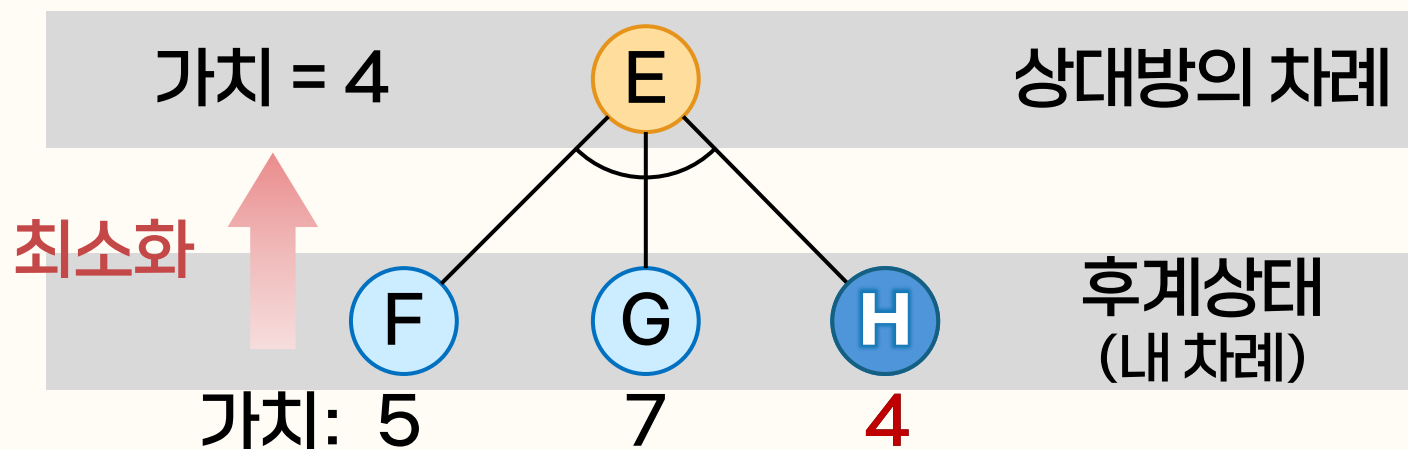


# 1. 최대최소 탐색(minimax search)의 개념

## ❏ 최대화 및 최소화



**최소화**: 상대방은 자신이 둘 수 있는 수 중에서 나에게 가장 불리한 수를 두려고 할 것이라고 가정함



# 1. 최대최소 탐색(minimax search)의 개념

## ■ 최대최소 탐색에 의한 수의 선택

- 현재 상태를 루트노드로 하여 최대화와 최소화 단계를 반복함으로써 현재 상태에서 둘 수 있는 다음 수들의 가치를 평가한 후, 그 중 가장 유리한 수를 선택함

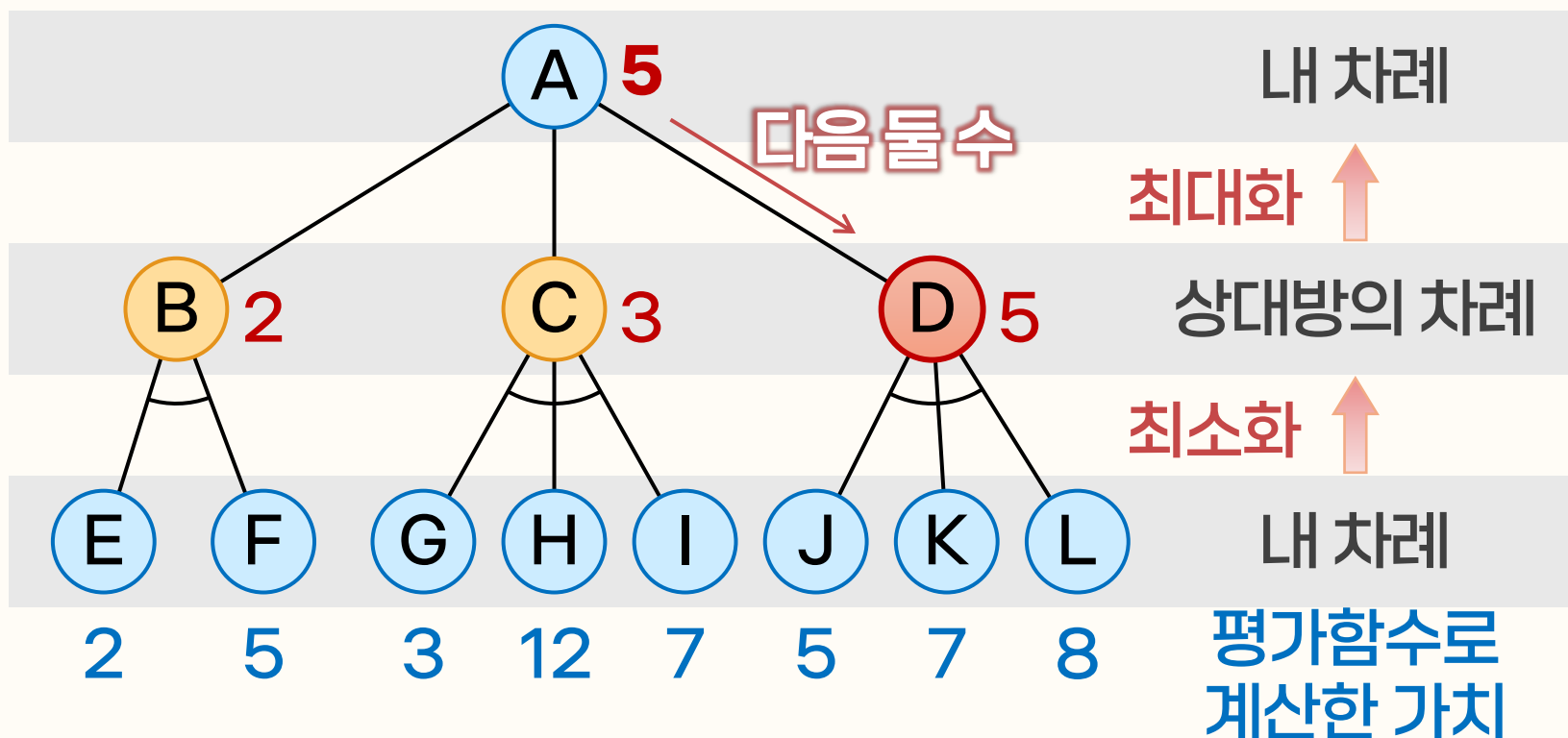


나와 상대방이 **각자에게 최적의 선택**을 한다는 가정하에 **나에게 최악인 선택(최소가치)**을 하는 **상대방**을 대상으로 **나의 결정의 가치가 최대가 되는 결정**을 내림

# 1. 최대최소 탐색(minimax search)의 개념

## ❑ 최대최소 탐색에 의한 수의 선택

- 최대최소 탐색트리





## 2. 최대최소 탐색 알고리즘

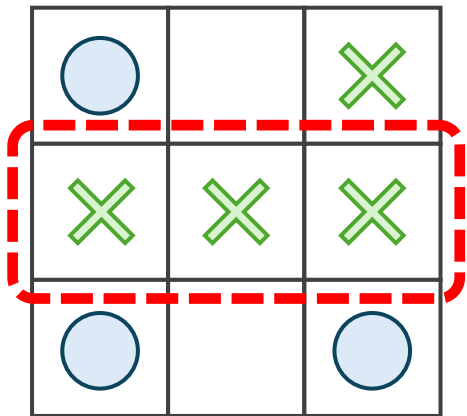
### ■ 탐색의 진행

- 트리의 규모가 매우 클 경우 모든 경우의 수를 탐색하여 종단상태까지 도달할 수 없음
  - 종단상태: 더 이상 게임을 진행할 수 없는 상태
    - 승리 / 패배 / 무승부가 결정되는 상태
- ⇒ 시스템의 가용자원에 따라 얼마나 깊이 탐색 과정을 반복할 것인지를 정함
  - 정해진 깊이에도달하면 경험적 지식을 반영하여 설계된 평가함수를 통해 그 노드의 가치를 추정

### 3. 최대최소 탐색의 예: 삼목게임

#### 삼목게임(Tic-tac-toe)이란?

- 두 사람이 가로/세로 3×3 크기의 판에 교대로 수를 두어 한 행, 열, 또는 대각선을 모두 점유하면 이기는 게임



→ X가 승리함

### 3. 최대최소 탐색의 예: 삼목게임

#### ■ 평가함수 F의 정의



승리한 상태:  $F = \infty$



패배한 상태:  $F = -\infty$

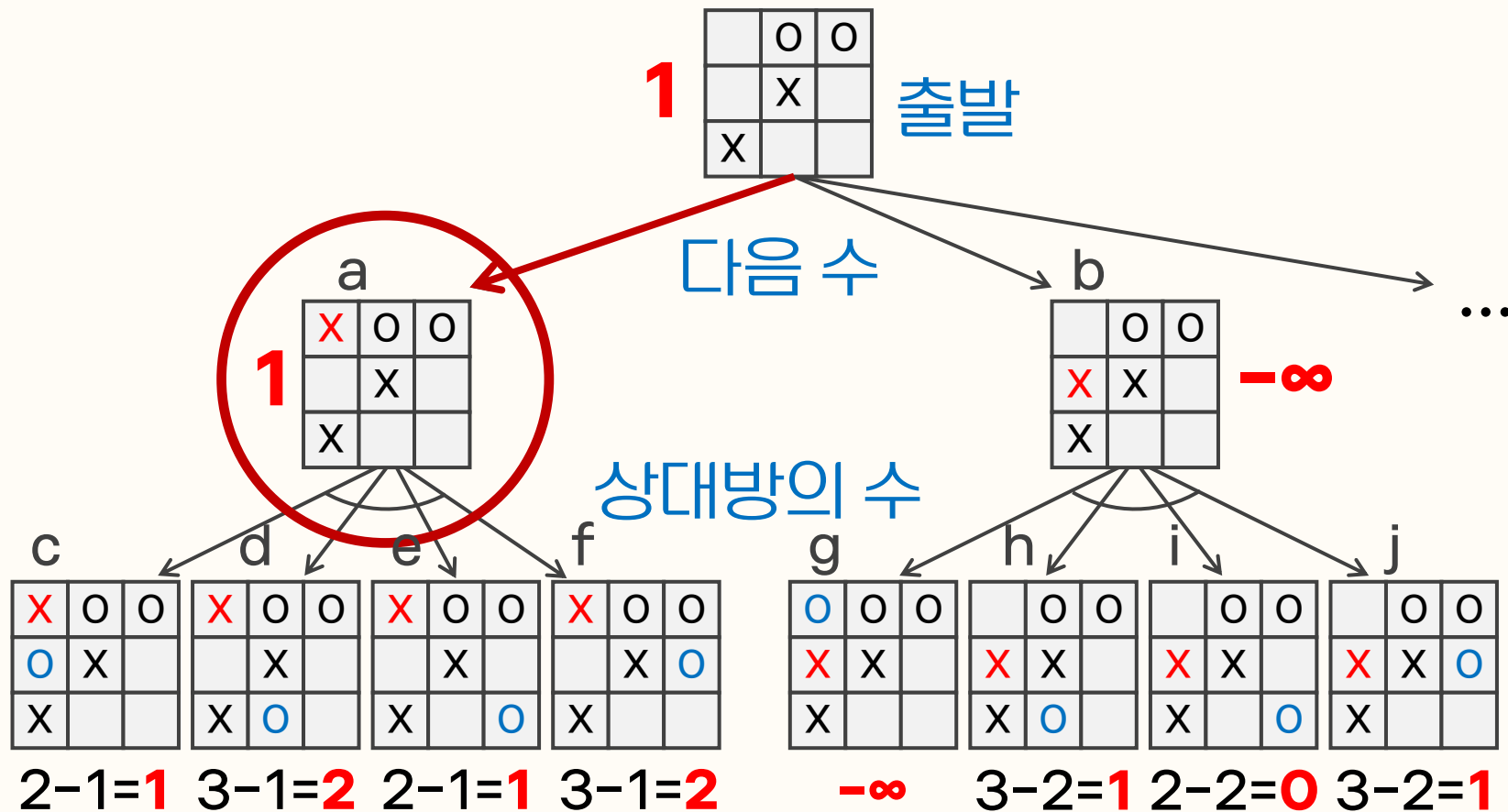


그 외의 경우:  $F = W - L$

- W: 이길 가능성이 있는 행, 열, 대각선의 수
- L: 질 가능성이 있는 행, 열, 대각선의 수

### 3. 최대최소 탐색의 예: 삼목게임

#### ■ 최대최소 탐색에 의한 수의 결정



## 4. $\alpha$ - $\beta$ 가지치기

### ■ $\alpha$ - $\beta$ 가지치기란?

- 최대최소 탐색트리에서 탐색이 불필요한 가지를 잘라내서 탐색의 성능을 높이기 위한 알고리즘



$\alpha$ : 어떠한 최대화 노드의 최대화 과정에서 지금까지 구한 가장 큰 가치

- ➡ 앞으로 가치를 구하려는 후계노드(최소화노드임)는 보다 더 큰 가치를 가져야만 그 최대화 노드의 가치가 될 수 있음
- ➡ 최소화 노드에서 어느 후계노드의 가치가  $v$ 일 때  $\alpha \geq v$ 라면 그 최소화 노드의 나머지 후계노드들은 가지치기함

## 4. $\alpha$ - $\beta$ 가지치기

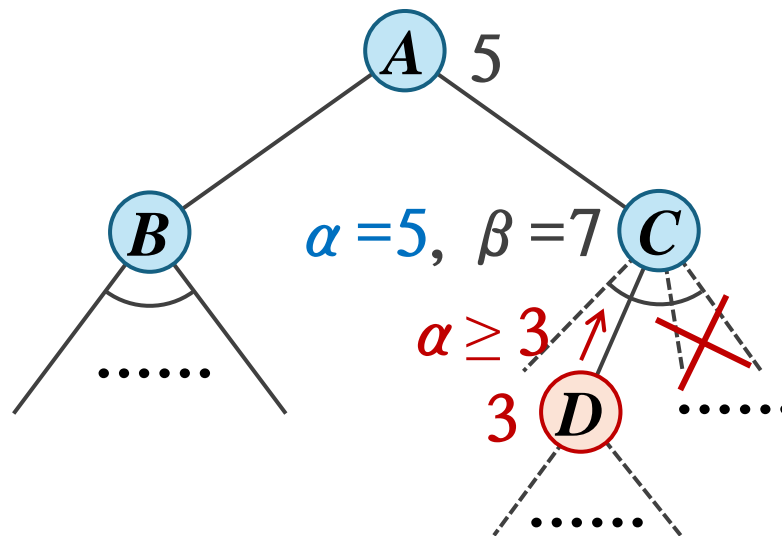
### ■ $\alpha$ - $\beta$ 가지치기란?

- 최대최소 탐색트리에서 탐색이 불필요한 가지를 잘라내서 탐색의 성능을 높이기 위한 알고리즘



$\alpha$ : 어떠한 최대화 노드의 최대값과 그 자식 노드들 중 가장 큰 값

- 앞으로 가치를 구하려는 노드에서 가장 큰 값을 가져야만 그 노드의 부모 노드의  $\alpha$  값을 갱신할 수 있다
- 최소화 노드에서 어느 한 자식 노드의 값이 부모 노드의  $\beta$  값을 갱신할 수 없게 되면 그 이후의 자식 노드들은 탐색하지 않는다



## 4. $\alpha$ - $\beta$ 가지치기

### ■ $\alpha$ - $\beta$ 가지치기란?

- 최대최소 탐색트리에서 탐색이 불필요한 가지를 잘라내서 탐색의 성능을 높이기 위한 알고리즘



$\beta$ : 어떠한 최소화 노드의 최소화 과정에서 지금까지 구한 가장 작은 가치

- ➡ 앞으로 가치를 구하려는 후계노드(최대화노드임)는 보다 더 작은 가치를 가져야만 그 최소화 노드의 가치가 될 수 있음
- ➡ 최대화 노드에서 어느 후계노드의 가치가  $v$ 일 때  $\beta \leq v$ 라면 그 최대화 노드의 나머지 후계노드들은 가지치기함

## 4. $\alpha$ - $\beta$ 가지치기

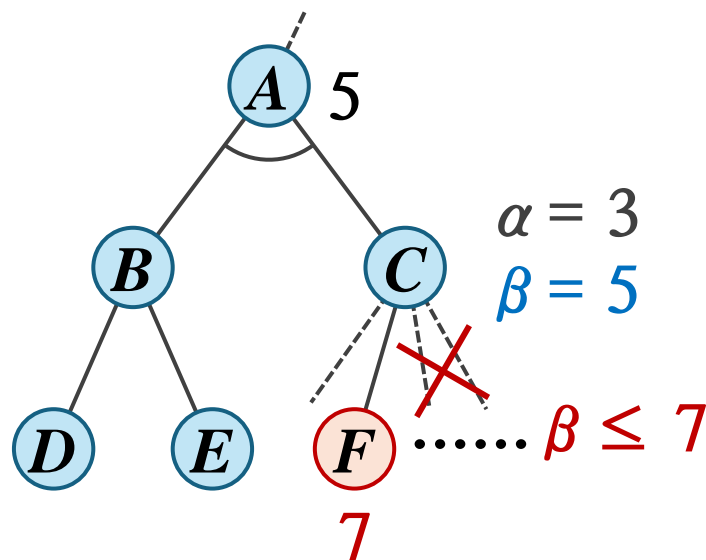
### ■ $\alpha$ - $\beta$ 가지치기란?

- 최대최소 탐색트리에서 탐색이 불필요한 가지를 잘라내서 탐색의 성능을 높이기 위한 알고리즘



$\beta$ : 어떠한 최소화 노드의 최소한 그자에서 나타나는 그자  
가장 작은 가치

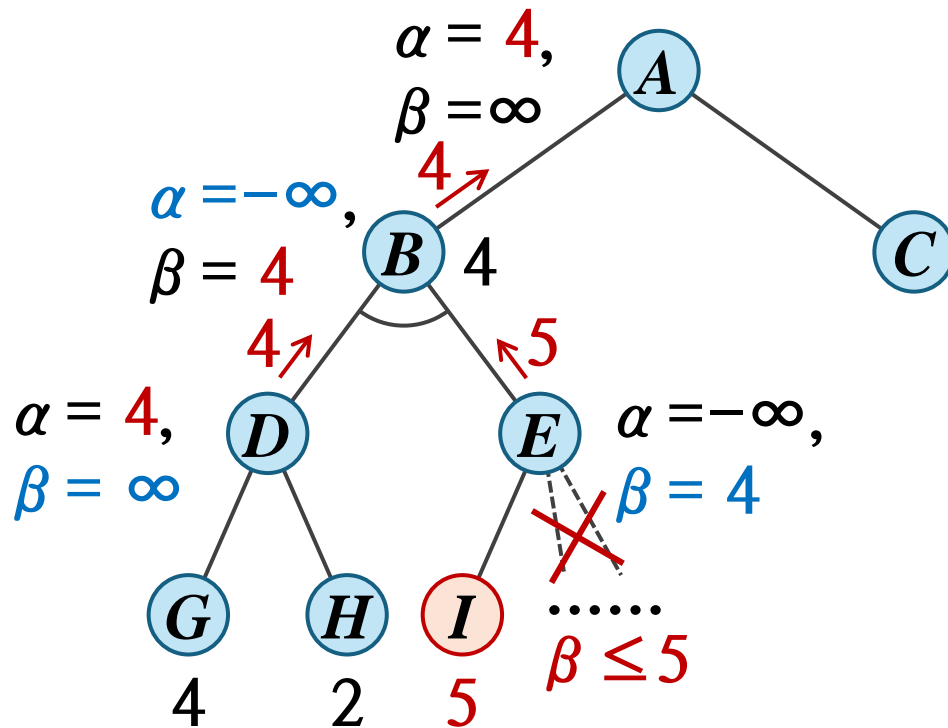
- 앞으로 가치를 구하려는 작은 가치를 가져야만 그
- 최대화 노드에서 어느 후 그 최대화 노드의 나머지





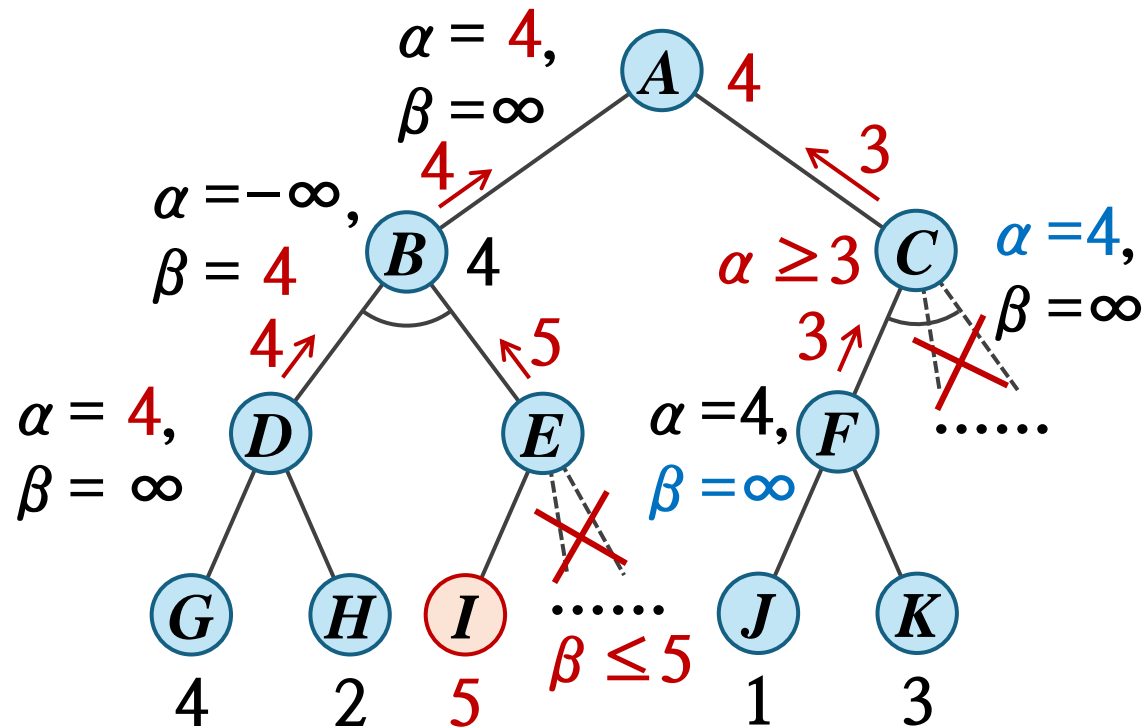
## 4. $\alpha$ - $\beta$ 가지치기

### ■ $\alpha$ - $\beta$ 가지치기 적용 예



## 4. $\alpha$ - $\beta$ 가지치기

### ■ $\alpha$ - $\beta$ 가지치기 적용 예



## 4. $\alpha$ - $\beta$ 가지치기

### $\alpha$ - $\beta$ 가지치기 알고리즘

```
1 function MiniMaxAB(n) //n은 루트노드
2    $\alpha \leftarrow -\infty$ ,  $\beta \leftarrow \infty$ ;
3   bestAction  $\leftarrow$  NULL;
4   for each a in n.possibleAction do
5     value  $\leftarrow$  MinimizeAB(n.doAction(a),  $\alpha$ ,  $\beta$ , 1);
6     if value >  $\alpha$  then
7        $\alpha \leftarrow$  value, bestAction  $\leftarrow$  a;
8     end-if;
9   end-for;
10  return bestAction;
11
12 function MinimizeAB(n,  $\alpha$ ,  $\beta$ , depth) //n:현재 노드, depth:n의 깊이
13   if n.CheckTerminal() or depth >= DEPTH_BOUND then
14     .....
```

## 4. $\alpha$ - $\beta$ 가지치기

### $\alpha$ - $\beta$ 가지치기 알고리즘

```
12 function MinimizeAB(n,  $\alpha$ ,  $\beta$ , depth) //n:현재 노드, depth:n의 깊이
13     if n.CheckTerminal() or depth >= DEPTH_BOUND then
14         return n.Evaluate();
15     end-if;
16     minValue  $\leftarrow$   $\infty$ ;
17     for each a in n.possibleAction do
18         value  $\leftarrow$  MaximizeAB(n.doAction(a),  $\alpha$ ,  $\beta$ , depth+1);
19         minValue  $\leftarrow$  min(value, minValue);
20         if  $\alpha$  >= minValue then
21             return minValue;
22         end-if;
23          $\beta$   $\leftarrow$  min(minValue,  $\beta$ );
24     end-for;
25     return minValue;
```

## 4. $\alpha$ - $\beta$ 가지치기

### $\alpha$ - $\beta$ 가지치기 알고리즘

```
27 function MaximizeAB( $n, \alpha, \beta, \text{depth}$ ) //  $n$ : 현재 노드,  $\text{depth}$ :  $n$ 의 깊이
28   if  $n.\text{CheckTerminal}()$  or  $\text{depth} \geq \text{DEPTH\_BOUND}$  then
29     return  $n.\text{Evaluate}()$ ;
30   end-if;
31    $\text{maxValue} \leftarrow -\infty$ ;
32   for each  $a$  in  $n.\text{possibleAction}$  do
33      $\text{value} \leftarrow \text{MinimizeAB}(n.\text{doAction}(a), \alpha, \beta, \text{depth}+1)$ ;
34      $\text{maxValue} \leftarrow \max(\text{value}, \text{maxValue})$ ;
35     if  $\beta \leq \text{maxValue}$  then
36       return  $\text{maxValue}$ ;
37     end-if;
38      $\alpha \leftarrow \max(\text{maxValue}, \alpha)$ ;
39   end-for;
40   return  $\text{maxValue}$ ;
```



# 몬테카를로 트리 탐색

# 1. 몬테카를로 트리 탐색의 개요

## ■ 몬테카를로 트리 탐색(Monte Carlo tree search: MCTS)

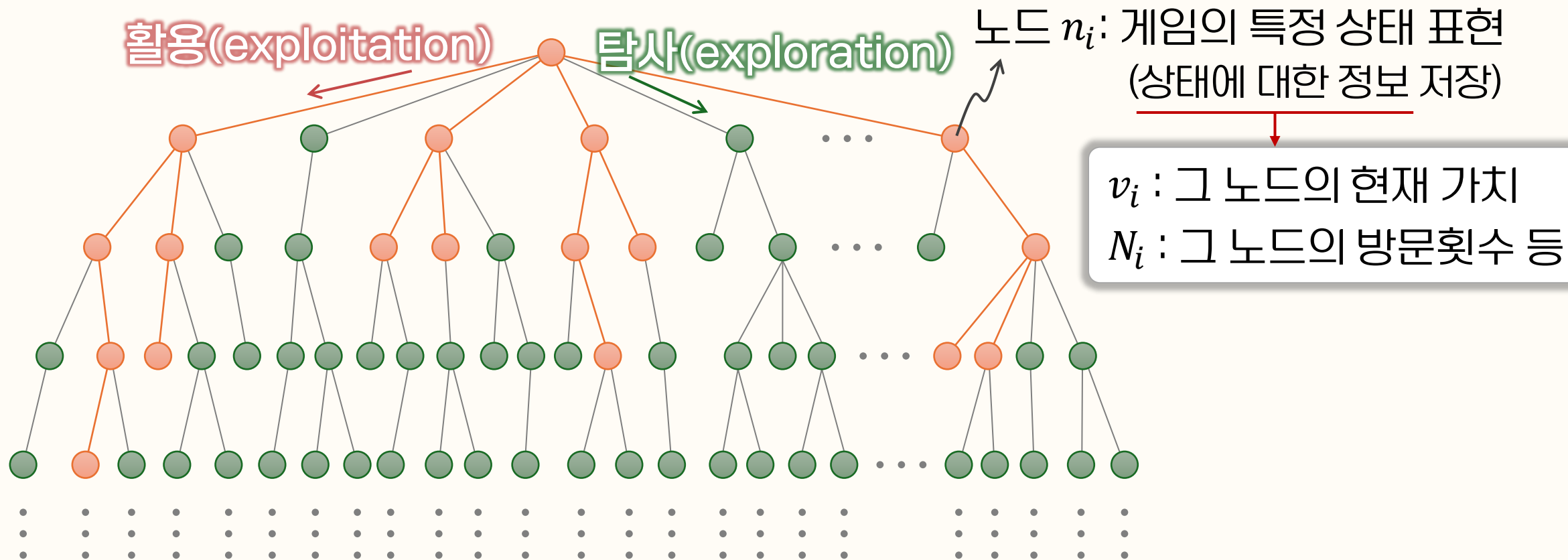
- 게임과 같은 의사결정 문제에 활용되는 경험적 탐색 알고리즘
- 탐색공간의 무작위 표본화를 바탕으로 탐색트리를 구성함



모나코의 몬테카를로(출처: wikipedia)

# 1. 몬테카를로 트리 탐색의 개요

## ■ 몬테카를로 트리 탐색(Monte Carlo tree search: MCTS)



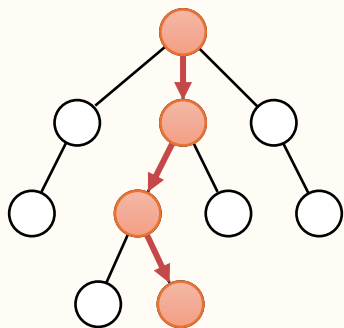
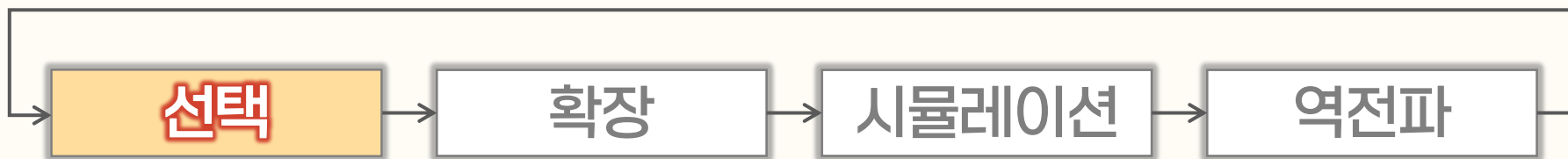
● 검토를 시도해 본 경로의 노드

● 검토해보지 않은 경로의 노드



## 2. 몬테카를로 트리 탐색 알고리즘

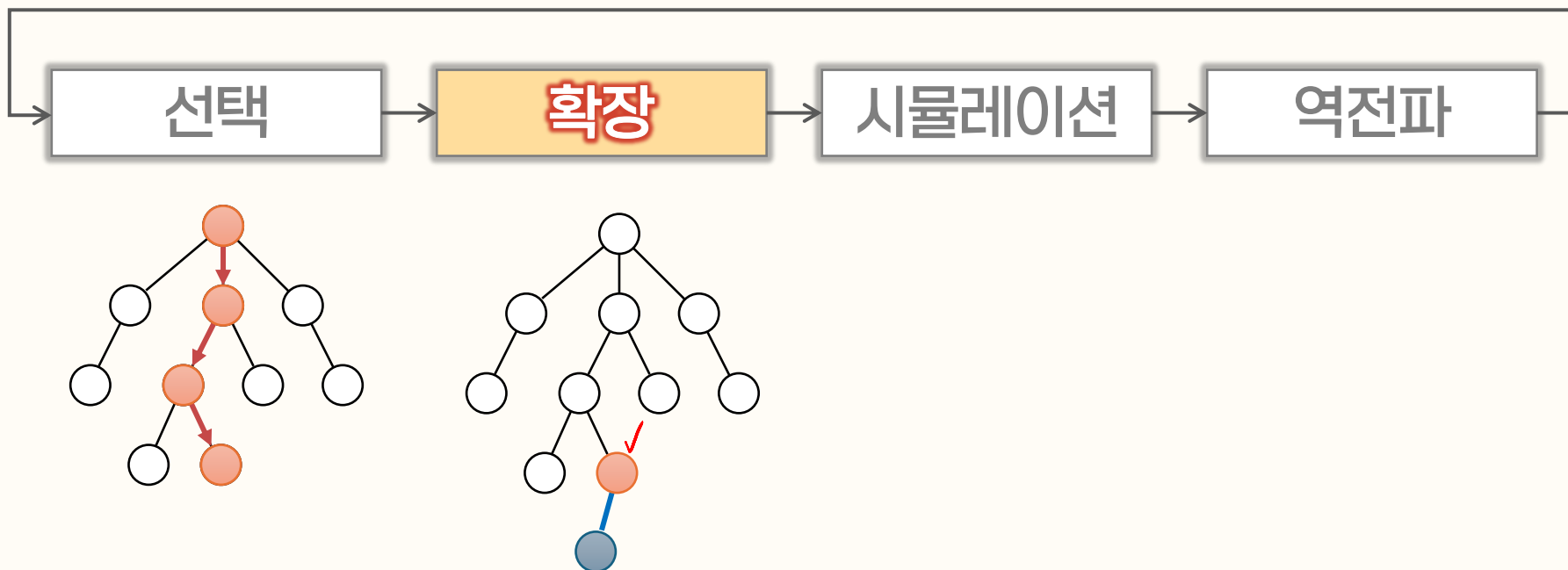
### ■ MCTS 알고리즘을 구성하는 네 단계



- 루트노드에서 시작하여 선택전략에 따라 자식노드를 선택하는 과정을 깊이방향으로 반복
- 아직 시도해 보지 않은 행동이 남아 있는 노드에 도달할 때까지 반복

## 2. 몬테카를로 트리 탐색 알고리즘

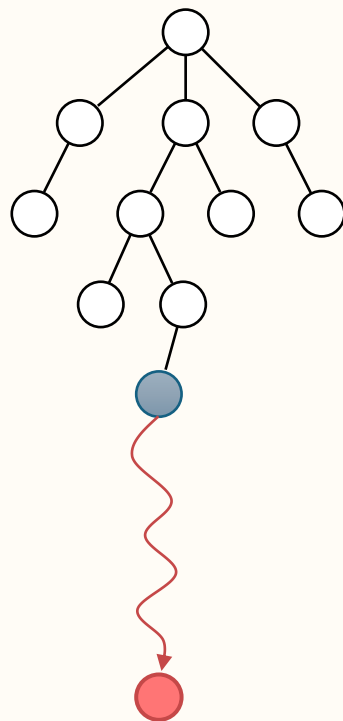
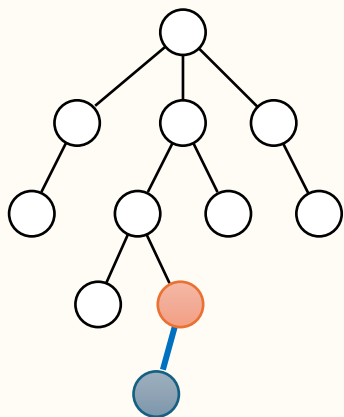
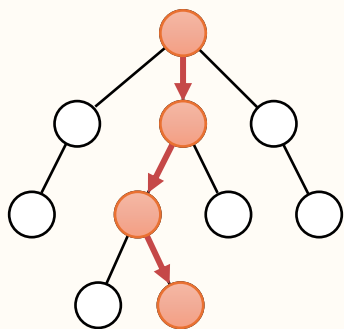
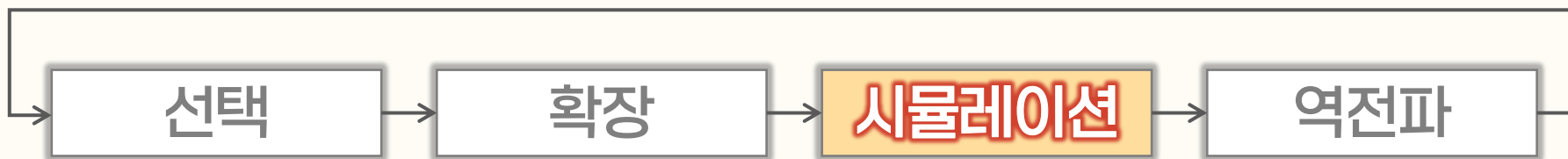
### ■ MCTS 알고리즘을 구성하는 네 단계



- 선택된 노드에 새로운 행동을 함으로써 자식노드를 생성하고 트리에 추가하여 트리를 확장

## 2. 몬테카를로 트리 탐색 알고리즘

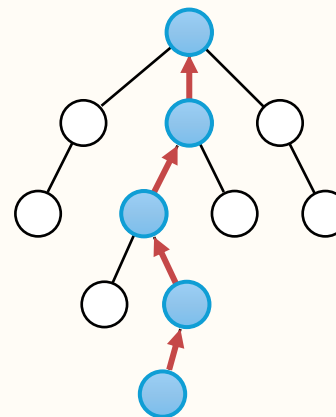
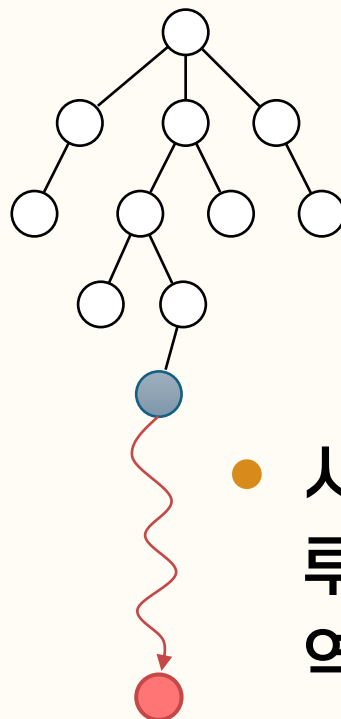
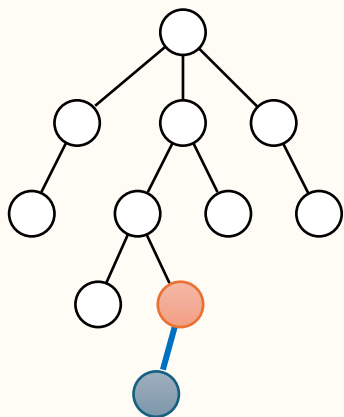
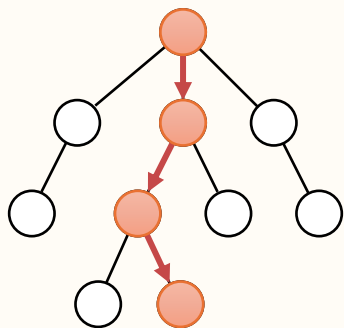
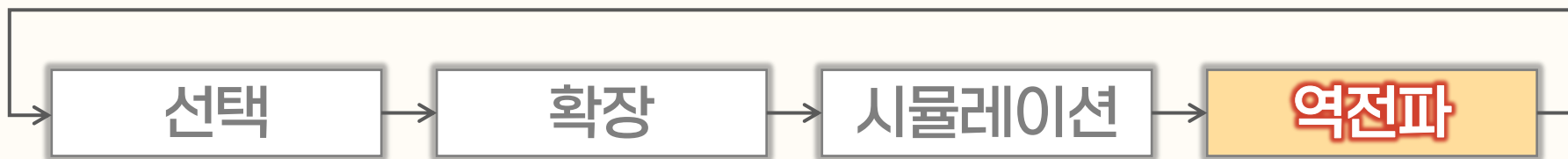
### ■ MCTS 알고리즘을 구성하는 네 단계



- 확장된 노드로부터 시작하여 게임이 끝날 때까지 스스로 게임을 진행(롤아웃, 플레이아웃)
- 모의게임의 결과: 게임에 따라 정의된 점수  
⇒ 승리(+1), 무승부(0), 패배(-1) 등

## 2. 몬테카를로 트리 탐색 알고리즘

### ■ MCTS 알고리즘을 구성하는 네 단계



- 시뮬레이션 결과를 확장된 노드로부터 루트노드까지 선택경로를 따라 역전파하여 통계를 업데이트함

## 2. 몬테카를로 트리 탐색 알고리즘

### 몬테카를로 트리 탐색 알고리즘

```
1 function MCTS(s) // s는 현재 상태
2   nRoot ← CreateNode(s) // 루트노드 하나만 있는 트리 시작
3   nRoot.vSum ← 0, nRoot.NVisit ← 0 // 가치와 방문횟수 초기화
4   while 시간 예산 이내 do
5     n ← nRoot
6     while not Terminal(n) do
7       a ← NewAction(n) // 시도하지 않은 행동 선택
8       if a = NULL then // 시도하지 않은 행동이 없다면
9         n ← BestChild(n) // 선택 정책에 따라 자식노드 선택
10      else
11        n ← Expand(n, a) // 선택된 행동으로 자식노드 확장
12        exit-loop
13      end-if
14    end-while
```

## 2. 몬테카를로 트리 탐색 알고리즘

### 몬테카를로 트리 탐색 알고리즘

```
1 function MCTS(s) // s는 현재 상태
2   nRoot ← CreateNode(s) // 루트노드 하나만 있는 트리 시작
3   nRoot.vSum ← 0, nRoot.NVisit ← 0 // 가치와 방문횟수 초기화
4   while 시간 예산 이내 do
5     n ← nRoot
6     while not Terminal(n) do
7       a ← NewAction(n) // 시도하지 않은 행동 선택
8       .....
9     end-while
10    value ← Rollout(n) // 시뮬레이션을 진행한 결과를 얻는다.
11    // 리프로부터 루트에 이르는 경로에 존재하는 노드의 정보 업데이트
12    Backpropagate(n, value)
13  end-while
14  return BestAction(nRoot)
```

## 2. 몬테카를로 트리 탐색 알고리즘


### 몬테카를로 트리 탐색 알고리즘


```
1 function MCTS(s) // s는 현재 상태
2   nRoot ← CreateNode(s) // 루트노드 하나만 있는 트리 시작
3   nRoot.vSum ← 0, nRoot.NVisit ← 0 // 가치와 방문횟수 초기화
4   while 시간 예산 이내 do
...     .....
18   end-while
19   return BestAction(nRoot)
20
21 function Backpropagate(n, v)
22   while n ≠ NULL do
23     n.vSum ← n.vSum + v // 노드의 가치 업데이트
24     n.NVisit ← n.NVisit + 1 // 노드의 방문횟수 업데이트
25     n ← n.parent
26   end-while
```

### 3. 몬테카를로 트리탐색을 위한 전략의 결정

#### ■ 선택 전략

- 주어진 노드의 자식노드 중 하나를 선택하기 위한 전략
- 탐사와 활용 사이의 균형을 이룰 수 있도록 설계

 **탐사(exploration)**: 평가의 불확실성으로 인해 아직은 덜 유망한 것으로 보이지만 향후 우수한 것으로 드러날 수 있는 수들을 선택할 수 있도록 하는 것

 **활용(exploitation)**: 지금까지의 결과 중 가장 우수한 결과를 이끌어내는 수를 선택하는 것

➡ Multi-armed bandit(MAB) 문제



### 3. 몬테카를로 트리탐색을 위한 전략의 결정

#### ■ 선택 전략

- UCT(upper confidence bound applied to trees) 알고리즘
  - UCB1: 잘 알려진 신뢰도 상한(upper confidence bound: UCB)
    - 노드  $n_p$ 에서 자식노드들 중 하나를 선택할 때 자식노드  $n_i$ 의 UCB1의 계산:

$$\text{UCB1}(n_i) = \bar{v}_i + C \times \sqrt{\frac{\ln N_p}{N_i}},$$

활용

탐사

$\bar{v}_i$ : 노드  $n_i$ 의 평균 가치

$N_i$ : 노드  $n_i$ 의 방문횟수

$N_p$ :  $n_i$ 의 부모노드의 방문횟수

### 3. 몬테카를로 트리탐색을 위한 전략의 결정

#### ■ 선택 전략

- UCT(upper confidence bound applied to trees) 알고리즘
  - UCB1: 잘 알려진 신뢰도 상한(upper confidence bound: UCB)
    - 노드  $n_p$ 에서 자식노드들 중 하나를 선택할 때 자식노드  $n_i$ 의 UCB1의 계산:

$$\text{UCB1}(n_i) = \bar{v}_i + C \times \sqrt{\frac{\ln N_p}{N_i}},$$

$\bar{v}_i$ : 노드  $n_i$ 의 평균 가치  
 $N_i$ : 노드  $n_i$ 의 방문횟수  
 $N_p$ :  $n_i$ 의 부모노드의 방문횟수

⇒ 노드  $n_p$ 에서 다음의 자식  $n_k$ 를 선택

$$k = \underset{i}{\operatorname{argmax}} \text{UCB1}(n_i)$$

### 3. 몬테카를로 트리탐색을 위한 전략의 결정

#### ■ 시뮬레이션 전략

- 선택된 노드로부터 게임이 끝날 때까지 스스로 수를 선택하여 게임을 진행함
- 수의 선택 방법
  - 순수한 무작위 방법
  - 적절한 시뮬레이션 전략에 따른 유사 무작위 수

#### ■ 역전파 전략

- 가치의 누적값과 방문횟수를 각각의 노드에 저장하고, 이의 평균을 사용하는 방법이 많이 쓰임

### 3. 몬테카를로 트리탐색을 위한 전략의 결정

#### ■ 최종적인 최적 행동 선택

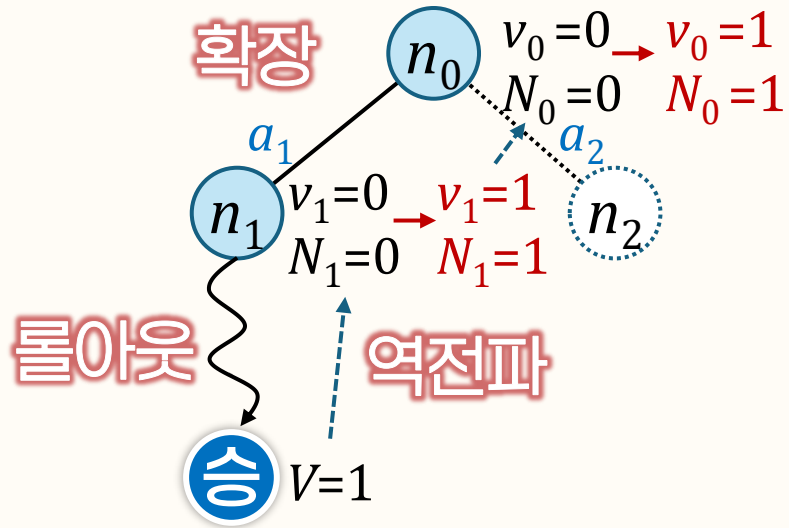
- 적절히 정한 계산 한계에 도달하여 시뮬레이션을 마치고, 최종적으로 루트에서 자식노드들 중 하나를 선택하여 다음 수를 결정하는 전략

- ① 최대 자식(max child): 가장 큰 보상을 갖는 자식을 선택
- ② 강인한 자식(robust child): 가장 많이 방문한 자식을 선택
- ③ 최대-강인 자식(max-robust child): 방문횟수가 가장 많고 가장 큰 보상을 갖는 루트 자식을 선택
- ④ 안전한 자식(secure child): 신뢰도 하한(lower confidence bound)이 최대인 자식을 선택

$$v + \frac{A}{\sqrt{n}}, \quad v \text{는 노드의 가치, } n \text{은 방문횟수, } A \text{는 적절한 상수}$$

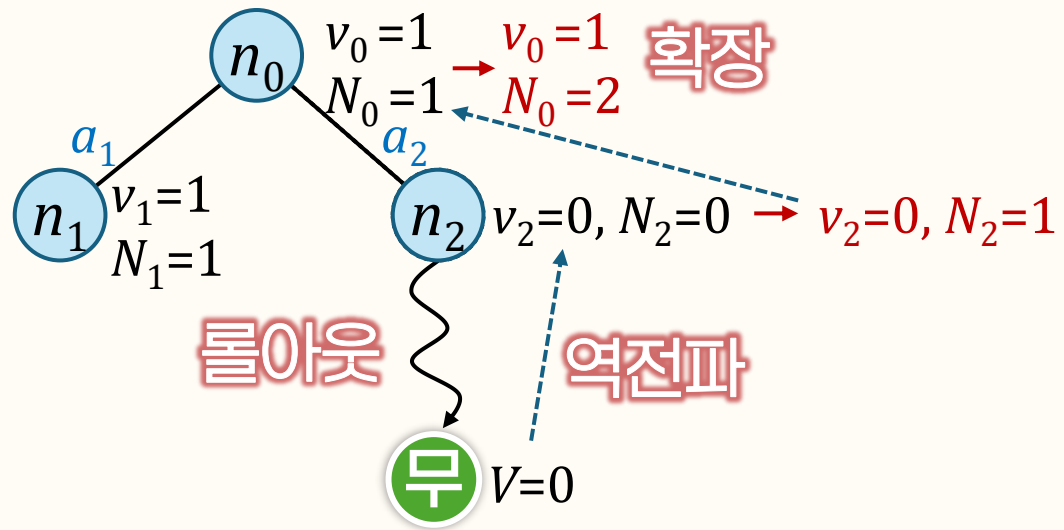
## 4. 몬테카를로 트리 탐색 적용 예

### ■ 1회 반복



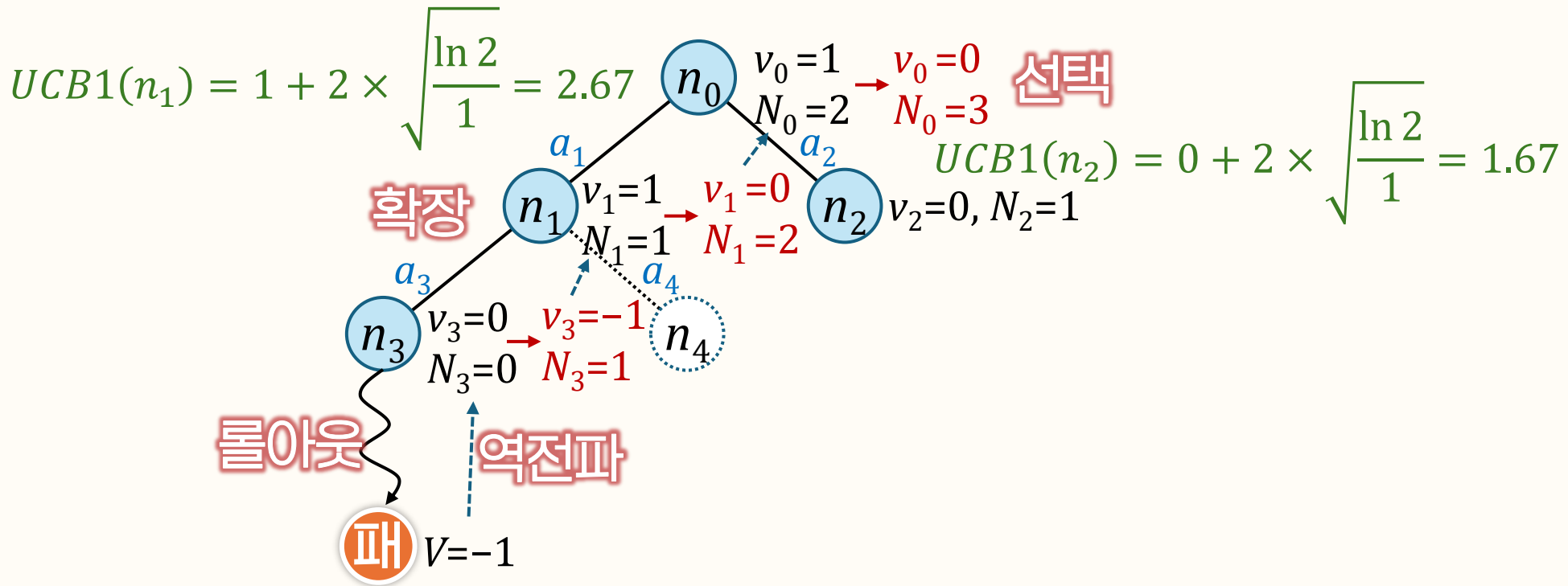
## 4. 몬테카를로 트리 탐색 적용 예

### ■ 2회 반복



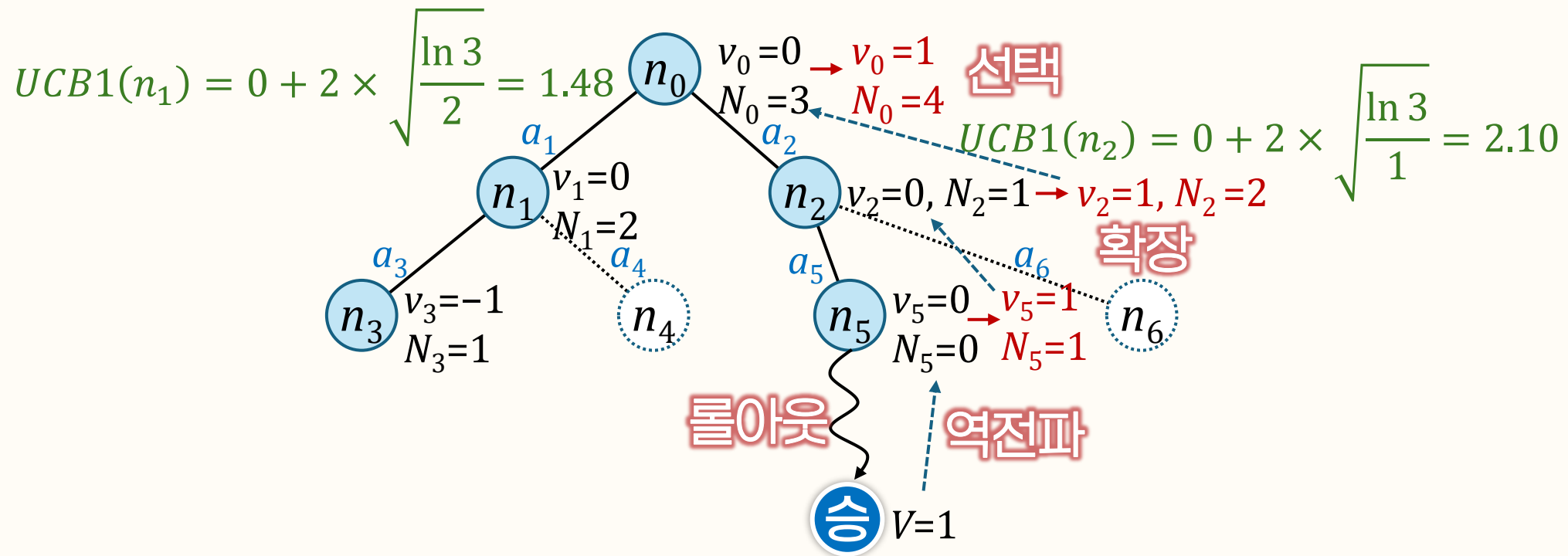
## 4. 몬테카를로 트리 탐색 적용 예

### 3회 반복



## 4. 몬테카를로 트리 탐색 적용 예

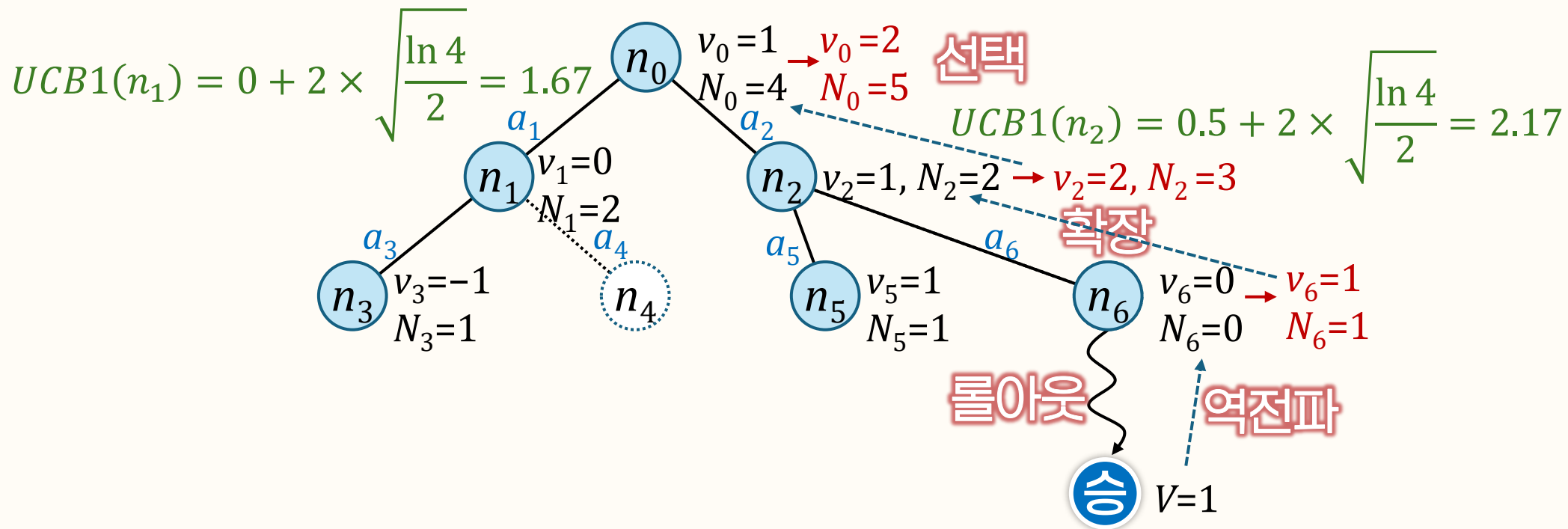
### 4회 반복





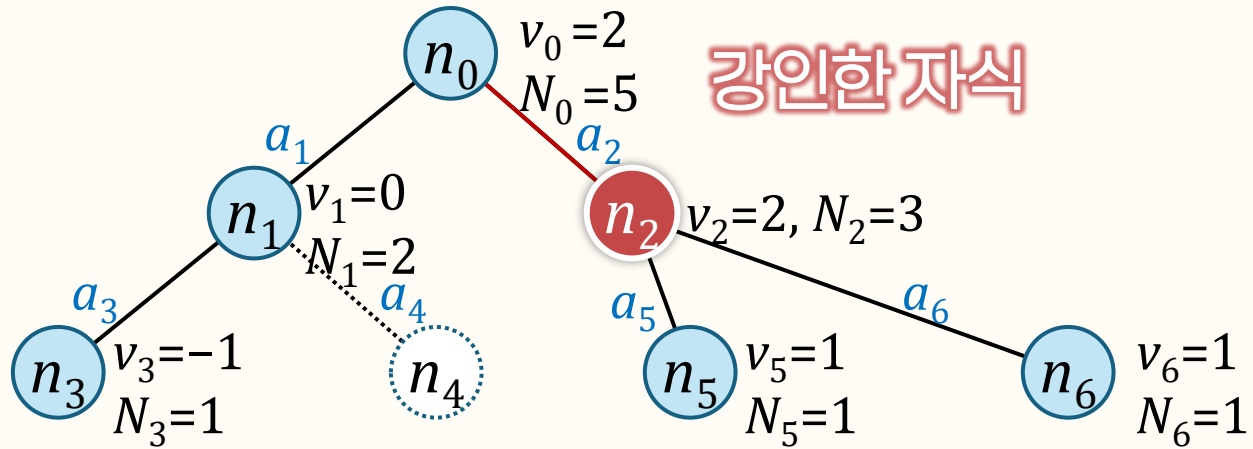
## 4. 몬테카를로 트리 탐색 적용 예

### ■ 5회 반복



## 4. 몬테카를로 트리 탐색 적용 예

### ■ 5회 반복 후 실행할 수의 선택



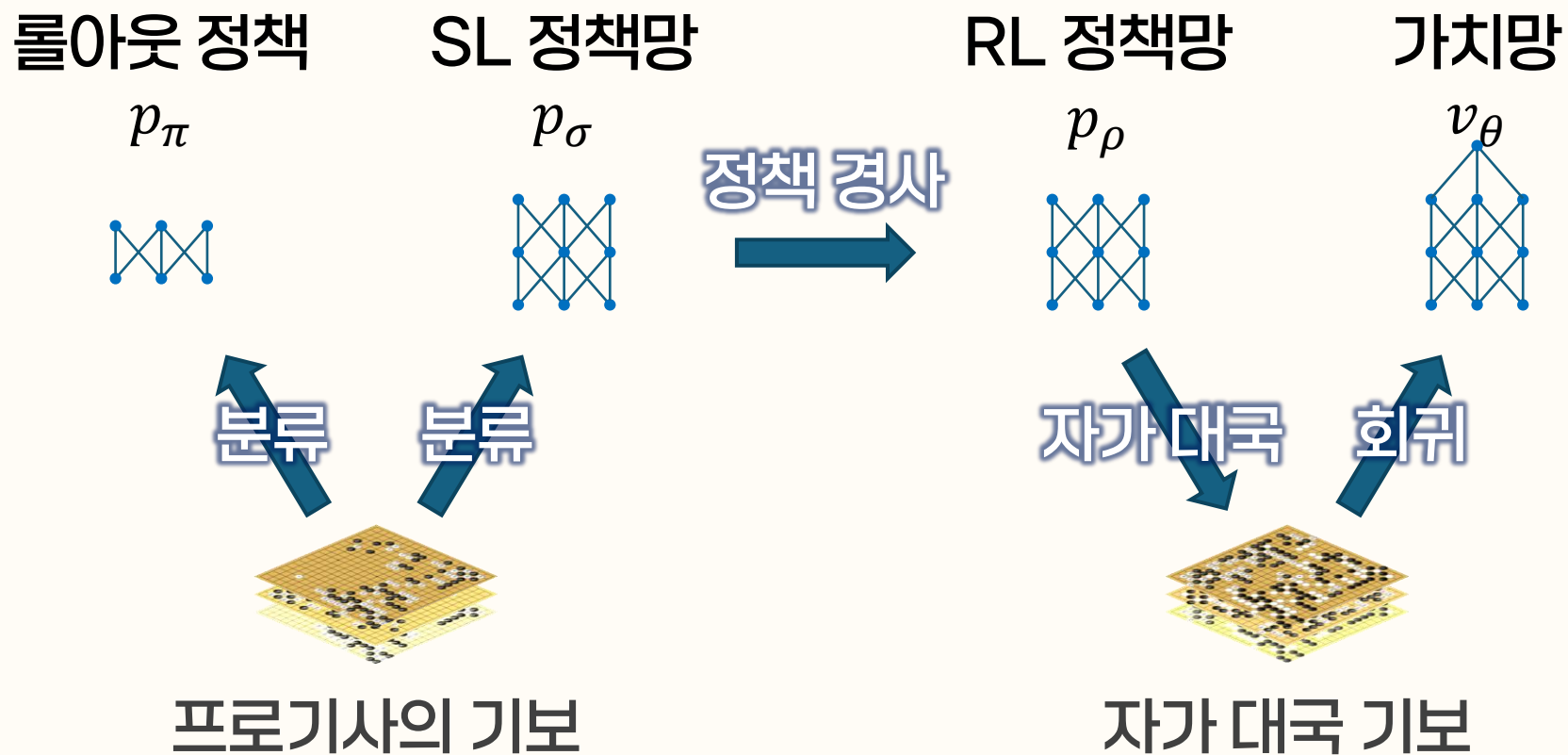
## 5. 사례 : 알파고와 몬테카를로 트리 탐색

### ■ AlphaGo Fan 개요

- 몬테카를로 트리 탐색을 수행함
- 탐색의 각 단계에 필요한 결정을 하기 위해 프로 기사 대국의 기보와 자가대결을 통해 학습된 신경망을 활용
  - 바둑판의 돌의 배치를  $19 \times 19$  영상의 형태로 전달하고, 이를 CNN으로 학습, 분류, 회귀 등의 처리를 함
  - 가치망을 이용하여 착점의 가치를 평가하고, 정책망을 이용하여 착점을 샘플링하거나 탐색 진행 방향 결정
- AlphaGo Lee, AlphaGo Master, AlphaGo Zero, Alpha Zero 등으로 발전

## 5. 사례 : 알파고와 몬테카를로 트리 탐색

### ■ 정책 결정 및 가치 계산을 위한 신경망 학습 구조



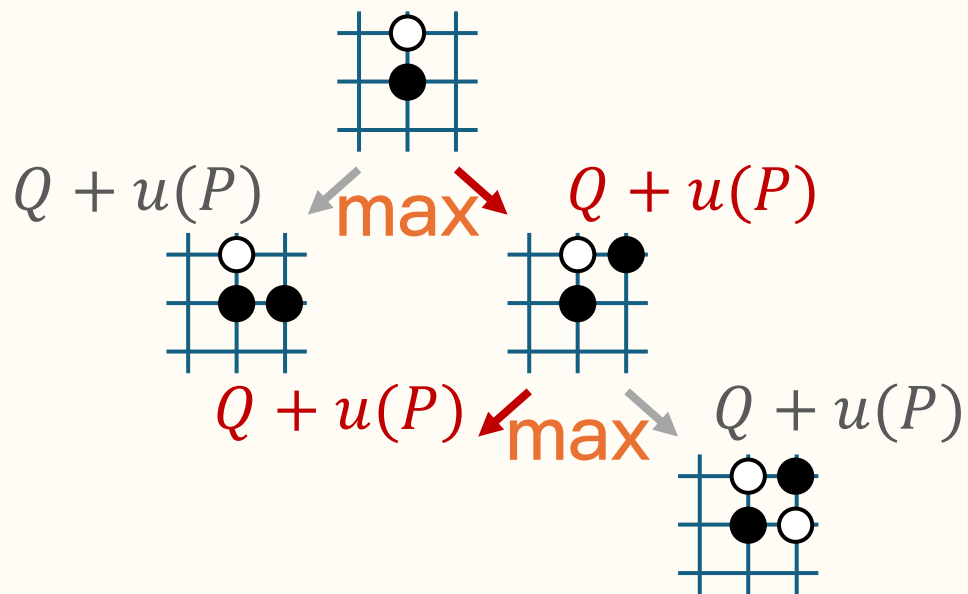
- *D. Silver, et al., "Mastering the game of Go with deep neural networks and tree search," Nature vol.529, pp.484-489, Jan. 2016.*

## 5. 사례 : 알파고와 몬테카를로 트리 탐색

### 알파고의 몬테카를로 트리 탐색



선택



$Q$ : 수의 가치

$u$ : 보너스(탐사를 장려하기 위한 항 포함)

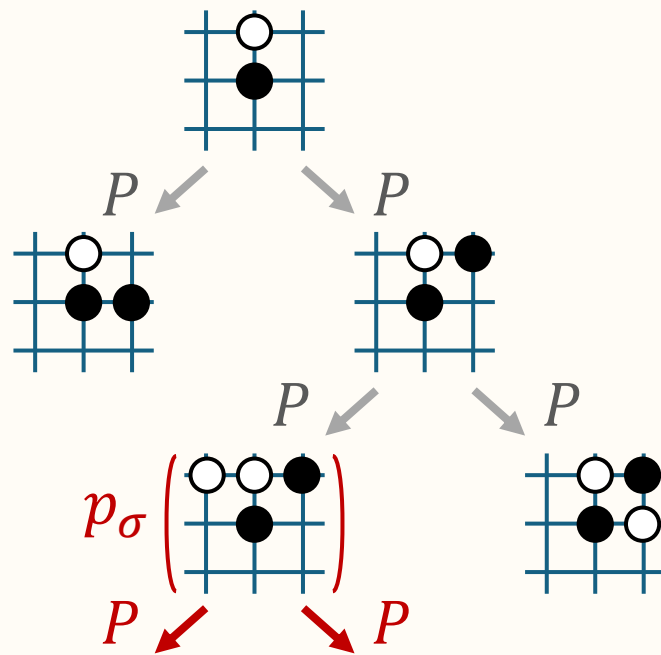
- *D. Silver, et al., "Mastering the game of Go with deep neural networks and tree search," Nature vol.529, pp.484-489, Jan. 2016.*

## 5. 사례 : 알파고와 몬테카를로 트리 탐색

### 알파고의 몬테카를로 트리 탐색



확장



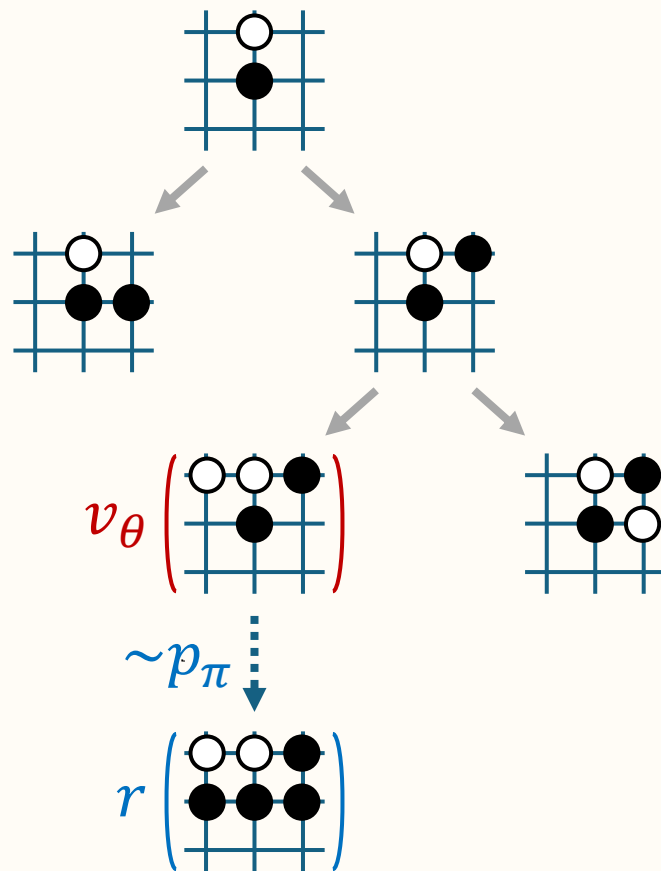
- *D. Silver, et al., “Mastering the game of Go with deep neural networks and tree search,” Nature vol.529, pp.484-489, Jan. 2016.*

## 5. 사례 : 알파고와 몬테카를로 트리 탐색

### 알파고의 몬테카를로 트리 탐색



평가

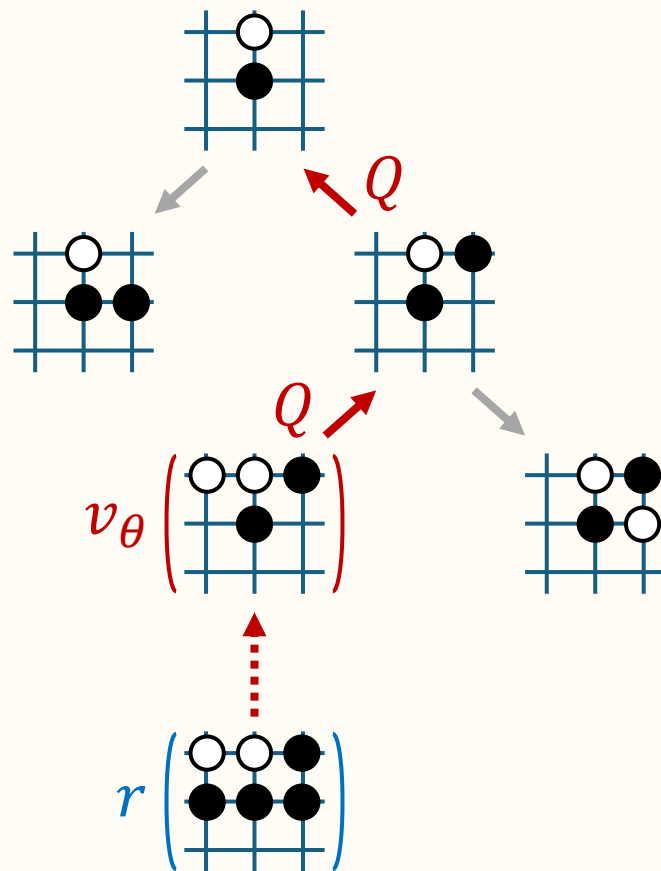


- *D. Silver, et al., “Mastering the game of Go with deep neural networks and tree search,” Nature vol.529, pp.484-489, Jan. 2016.*

## 알파고의 몬테카를로 트리 탐색



## 역전파



- *D. Silver, et al., “Mastering the game of Go with deep neural networks and tree search,” Nature vol.529, pp.484-489, Jan. 2016.*



## ✓ 최대최소 탐색

- 나와 상대방이 최적의 선택을 한다는 가정하에 나에게 최악인 선택을 하는 상대방을 대상으로 나의 결정의 가치가 최대가 되는 결정을 내리는 방식으로 게임을 진행한다.
- 최대화와 최소화를 번갈아 반복하여 가장 우수한 후계노드를 선택한다.
- 계산시간이나 메모리 등의 자원 한계를 고려하여 적절한 깊이 이상이 되면 경험적 지식을 반영하여 정의한 평가함수를 이용해서 그 상태의 가치를 추정한다.
- 탐색이 불필요한 가지를 잘라 내어 탐색의 성능을 높이기 위해  $\alpha$ - $\beta$  가지치기 알고리즘을 활용할 수 있다.

## ✓ 몬테카를로 트리 탐색

- 의사결정 문제에 활용되는 경험적 탐색 알고리즘으로, 탐색공간의 무작위 표본화를 바탕으로 탐색트리를 구성한다.
- 선택, 확장, 시뮬레이션, 역전파 단계를 반복한다.
- 선택 단계에서는 탐사와 활용의 균형을 이룰 수 있는 전략을 활용하며, UCT 알고리즘은 잘 알려진 선택전략 중 하나이다.
- 시뮬레이션 단계에서는 순수한 무작위 방법이나 적절한 전략에 따른 유사 무작위 방법으로 수를 선택하는 방법을 사용할 수 있다.
- 최종적인 최적 행동 선택을 위해서는 최대 자식, 강인한 자식, 최대-강인 자식, 안전한 자식 등 적절한 전략을 선택할 수 있다.

05강

다음시간안내 ▶▶▶

# 지식과 인공지능