알파베타 가지치기

개요

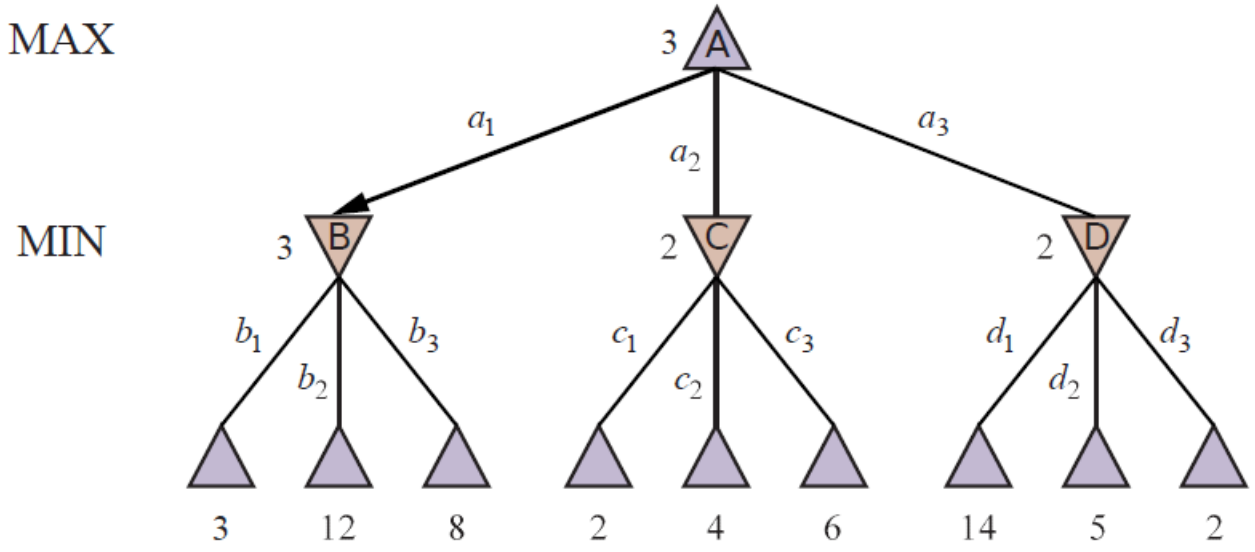
I. 알파베타 가지치기 개념

II. 알파베타 가지치기 알고리즘

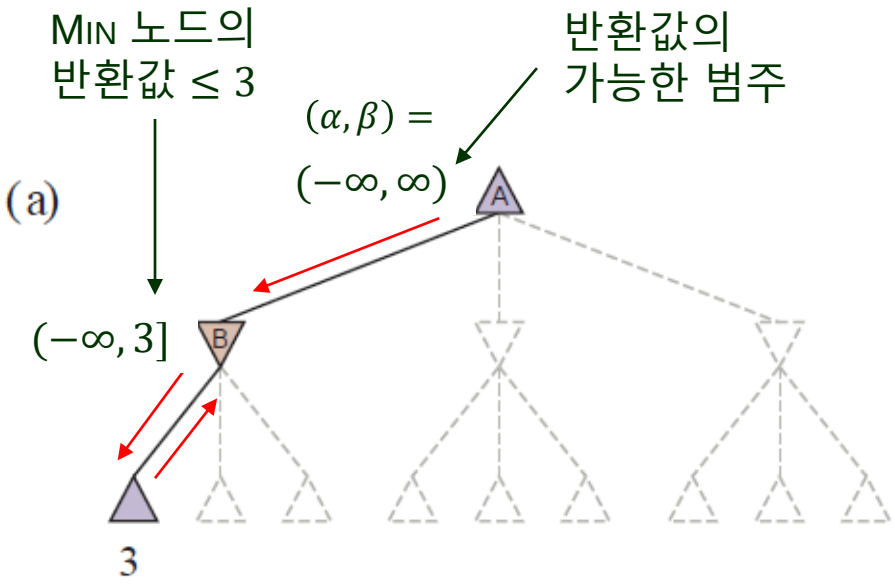
I. 알파베타 가지치기 개념

♣ 상태의 개수는 게임 트리의 깊이가 깊어짐에 따라 지수적으로 증가

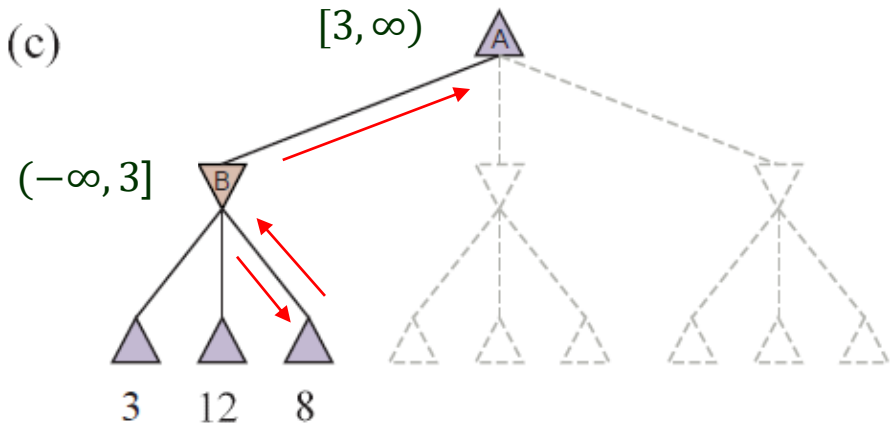
♦ 트리의 일부를 잘라내더라도, 판단 결과에 영향을 주지 않으면서 올바른 최소최대 결정을 계산할 수 있음



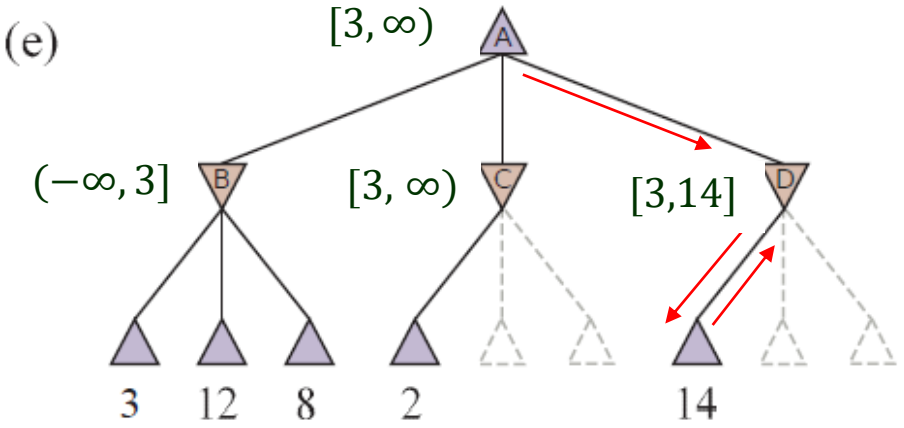
실행 예-1



Cont'd

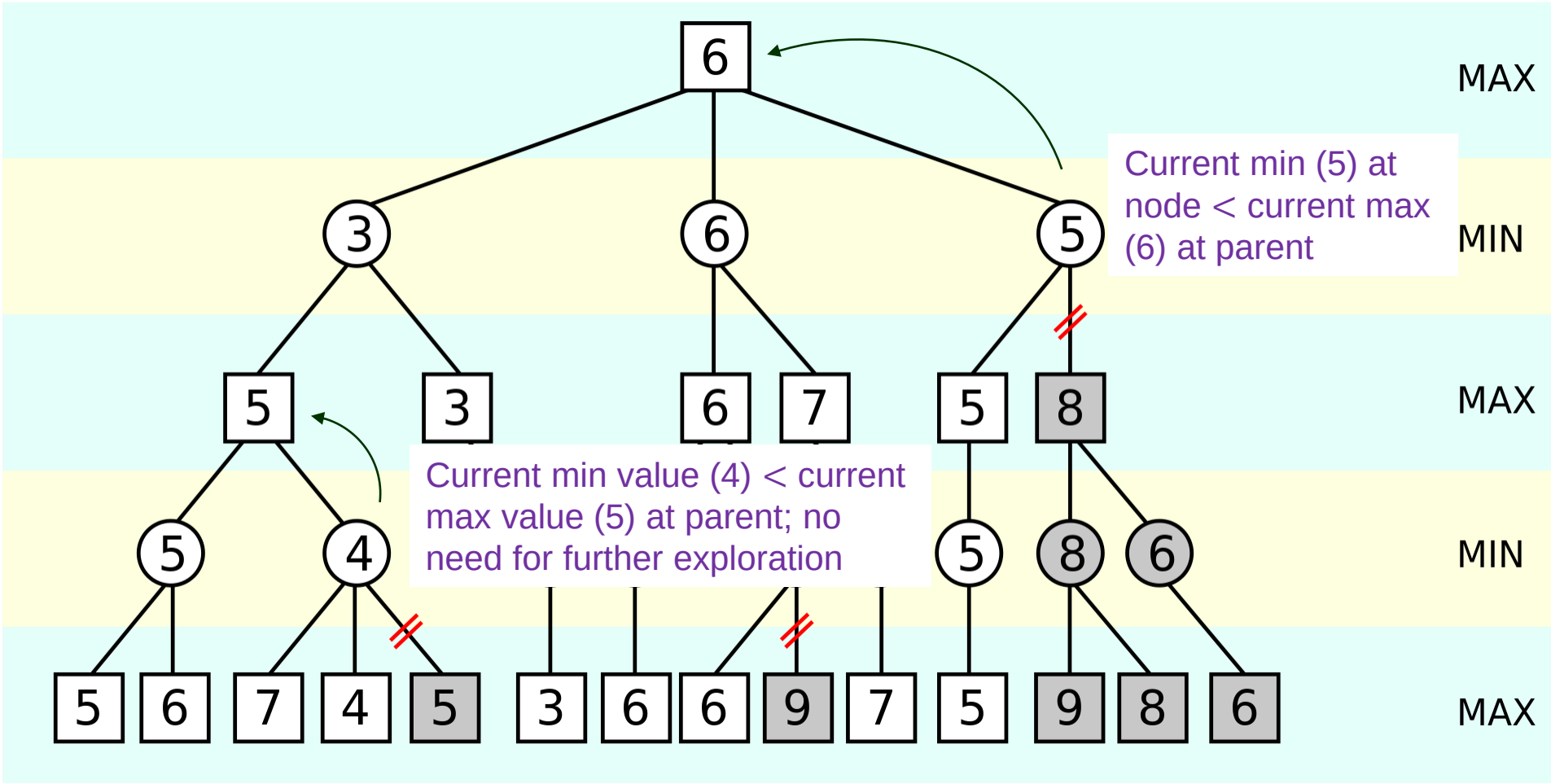


Cont'd

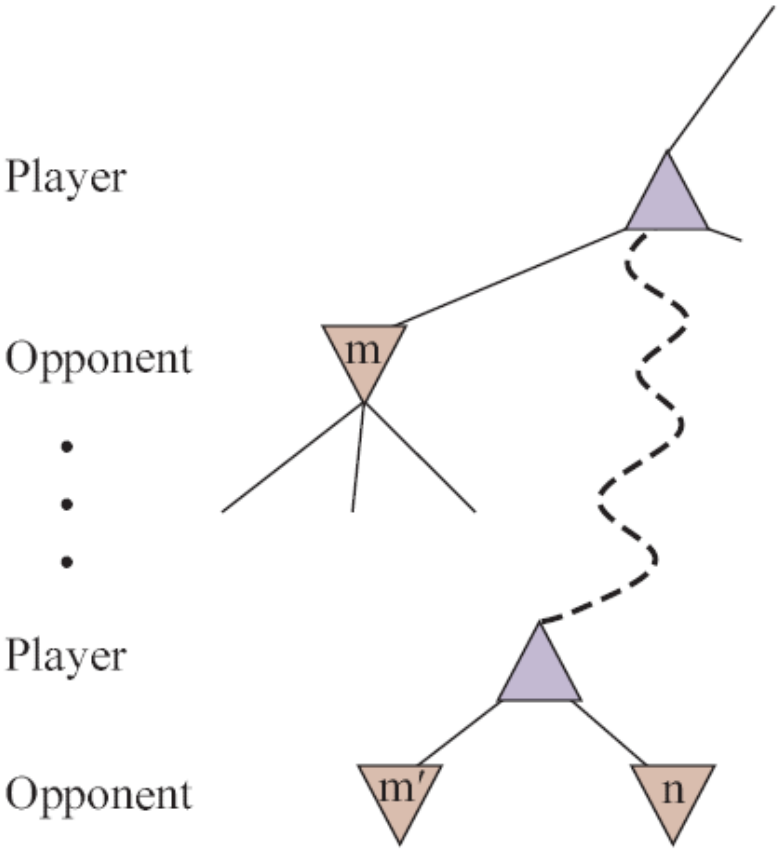


$$\begin{aligned}\text{MINIMAX(root)} &= \max(\min(3,12,8), \min(2, x, y), \min(14, 5, 2)) \\ &= \max(3, \min(2, x, y), 2) \\ &= \max(3, z, 2) \qquad \text{where } z = \min(2, x, y) \leq 2 \\ &= 3.\end{aligned}$$

실행 예-2


6
3
6
5
5
3
6
7
5
8
5
4
5
8
6
5
6
7
4
5
3
6
6
9
7
5
9
8
6
MAX
MIN
MAX
MIN
MAX
Current min (5) at node < current max (6) at parent
Current min value (4) < current max value (5) at parent; no need for further exploration

II. 노드 가지치기



플레이어는 더 나은 선택지가 있을 경우에는 특정 노드 n으로 이동하지 않음

- 동일 레벨에 있거나 (e.g., m')
- 더 위쪽에 있거나 (e.g., m)

위의 결론을 내릴 수 있는 충분한 정보를 찾아낸 경우 즉시 노드 n을 가지치기

알파값과 베타값

알파 베타 가지치기라는 이름은 두개의 파라미터 α, β 에서 따옴

α = MAX 측 플레이어의 게임 경로상에서 *최고값*(즉, 최선의 선택)

eventual value $\geq \alpha$ "at least"

β = MIN 측 플레이어의 게임 경로상에서 *최저값*(즉, 최선의 선택)

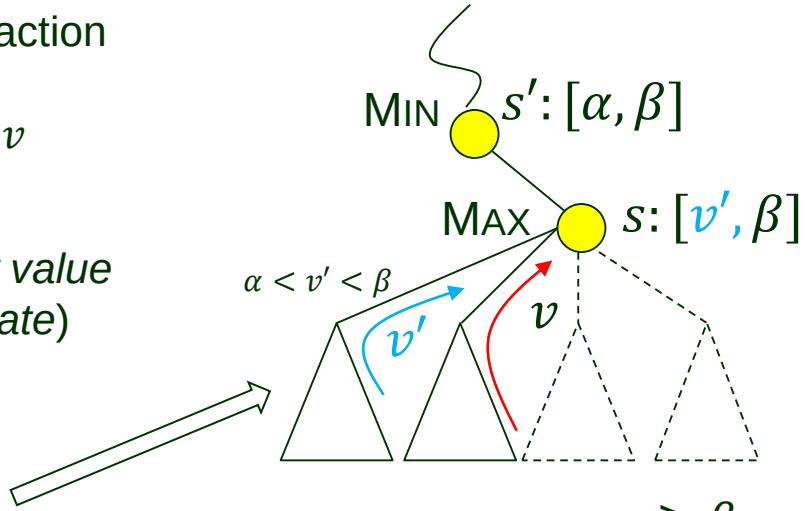
eventual value $\leq \beta$ "at most"

- 탐색 수행 중 α 값과 β 값 지속 갱신
- 가지치기가 수행되는 부분
  - MIN 노드에서 현재 값이 $\leq \alpha$ 인 부분
  - MAX 노드에서 현재 값이 $\geq \beta$ 인 부분

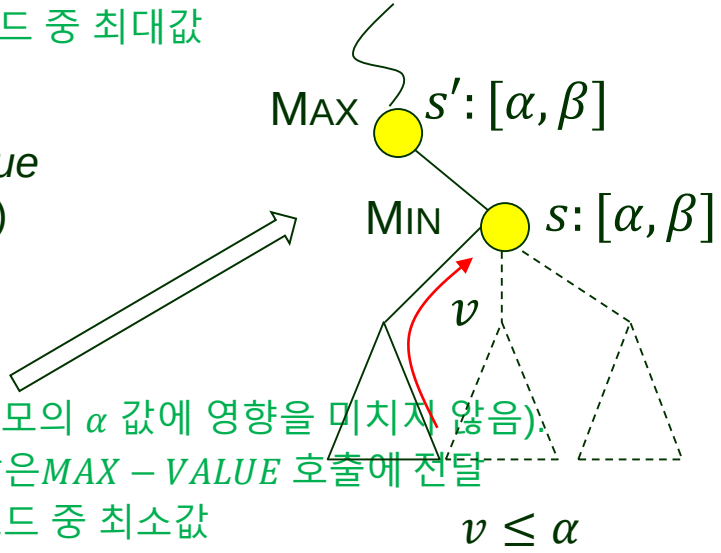
II. 알파 베타 탐색 알고리즘

function ALPHA-BETA-SEARCH(*state*) **returns** an action
  $v \leftarrow$MAX-VALUE(*state*, $-\infty, +\infty$)
  return the action in ACTIONS(*state*) with value v

// MAX-VALUE() 내에서는 β 값 고정
function MAX-VALUE (*state*,α, β) **returns** *a utility value*
  if TERMINAL-TEST(*state*) **then return** UTILITY(*state*)
  $v \leftarrow -\infty$
  for each a in ACTIONS(*state*) **do**
    $v \leftarrow$MAX(v, MIN-VALUE(RESULT(s, a),α, β))
    if $v \geq \beta$ **then return** v // 가지치기(반환된 v가 부모의 β 값에 영향을 미치지 않음)
    $\alpha \leftarrow$MAX(α, v) // $v < \beta$: 새로운 α는 for 루프 내의 남은 $MIN-VALUE$ 호출에 전달
  return v // 아무 것도 제거되지 않은 경우 모든 자식 노드 중 최대값



// MIN-VALUE() 내에서는 α값 고정
function MIN-VALUE (*state*,α, β) **returns** *a utility value*
  if TERMINAL-TEST(*state*) **then return** UTILITY(*state*)
  $v \leftarrow +\infty$
  for each a in ACTIONS(*state*) **do**
    $v \leftarrow$MIN(v, MAX-VALUE(RESULT(s, a),α, β))
    if $v \leq \alpha$ **then return** v // 가지치기(반환된 v가 부모의 α 값에 영향을 미치지 않음)
    $\beta \leftarrow$MIN(β, v)// $v > \alpha$: 새로은β 는for 루프 내의 남은$MAX-VALUE$ 호출에 전달
  return v // 아무 것도 제거되지 않은 경우 모든 자식 노드 중 최소값



알고리즘 간단 요약

MAX 노드와 MIN 노드

- 부모로부터 α 값과 β 값을 받음
- 실행된 액션의 최상의 값(결과)을 부모에게 전달

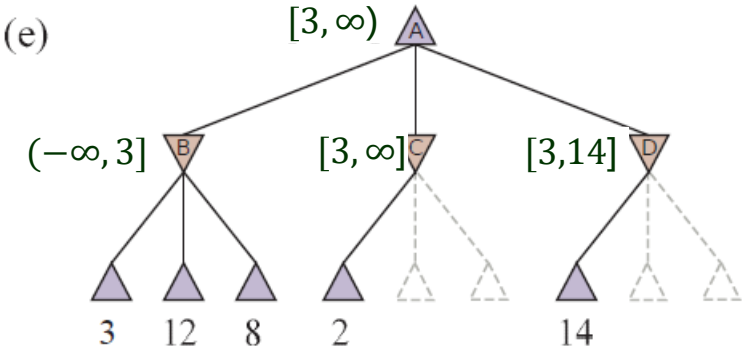
MAX 노드

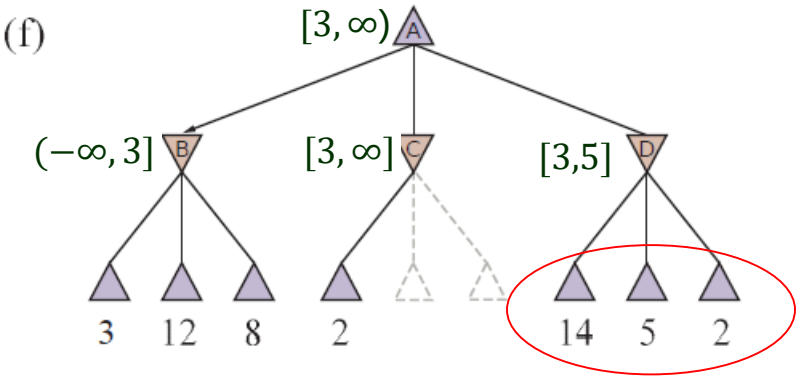
- 어떤 액션이 더 나은 값을 산출하면 α 값 갱신
- β 값은 갱신하지 않음
- 결과에 영향을 미치지 않는 액션은 β 값을 기준으로 스킵

MIN 노드

- 어떤 액션이 더 나은 값을 산출하면 β 값 갱신
- α 값은 갱신하지 않음
- 결과에 영향을 미치지 않는 액션은 α 값을 기준으로 스킵

액션 순서





2가 먼저 생성되었다면 14와 5는 가지치기 될 수 있었음

- 가지 치기의 효과는 후계자가 생성되는 순서에 매우 의존적

- "완벽 순서(Perfect Order)": 실효성 있는 가지치기 인수 $\sqrt{b}$ 채택. 노드 검사 복잡도 $O(b^{m/2})$ 로 축소. 원본 미니맥스 알고리즘은 $O(b^m)$

- 무작위로 액션 순서를 취할 경우 $O(b^{3m/4})$

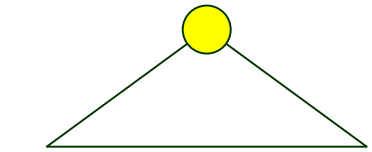
두가지 전략

♦ 체스의 경우, 간단한 정렬 함수(캡처, 위협, 전진, 후진의 순서 유지)를 사용하면 최적화에 근접한 결과를 얻을 수 있음

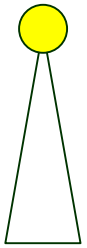
♠ 알파-베타 가지치기와 우수한 정렬 방법을 사용해도, 체스와 바둑과 같은 게임은 상태 공간이 매우 넓기 때문에 미니맥스만으로는 충분히 잘 작동하지 않음

클로드 새넌(1950) 두가지 전략을 제안:

- 타입 A (휴리스틱 알파베타 트리 탐색) – 체스
  - ♣ 특정 깊이까지 모든 가능한 움직임을 고려
  - ♣ 해당 깊이에서 각 상태의 효용값을 추정하기 위해 휴리스틱 함수 사용
- 타입 B (몬테칼로 트리 탐색) – 바둑
  - ♣ 나빠 보이는 수는 무시
  - ♣ 유망한 경로를 '가능한 한 멀리까지' 따라감



트리를 넓고 얕게 탐험



트리를 앏고
깊게 탐험