게임

개요

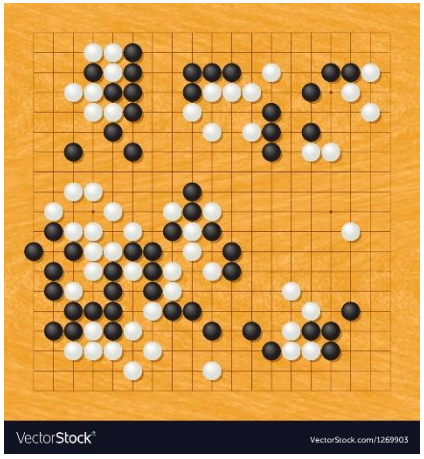
I. 적대적 탐색 알고리즘으로 풀어보는 게임

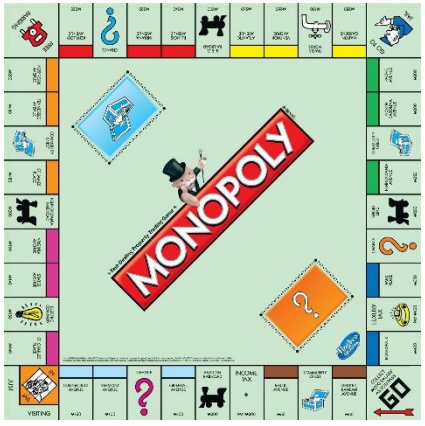
II. 최소최대 알고리즘

I. 게임

경쟁적 환경: 게임 참가자들간 목표 충돌(예: 서로 이기려고 함)

적대적 탐색 문제 (대표적인 예: 게임)



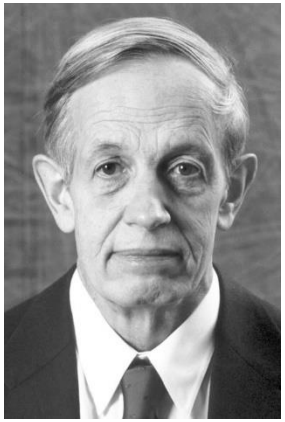


게임을 연구하는 이유

- 다중 에이전트 환경
  - 많은 에이전트가 함께 예측에 참가(예: 가격 상승)
  - 적대적 에이전트에 의한 비결정성(불확실성)
  - 새로운 모델링 기법 도입
- 수학적 게임 이론 - 경제학의 중요한 지류

  비협조적 게임 상황에 대한 나쉬 균형

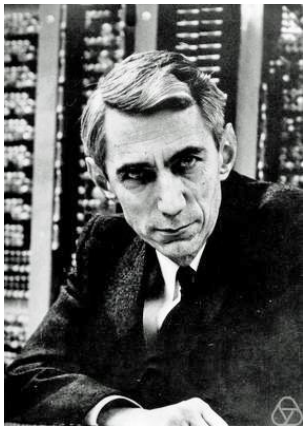
  한 플레이어가 전략을 선택하고, 상대방 플레이어가 자신들의 전략을 바꾸지 않는 한, 자신의 예상 이익을 늘릴 수 없음



존 나쉬 (프린스턴대)
노벨 경제학상(1994)

- 인공지능 분야의 매력적인 연구주제
  - 재미있고 즐거운 주제
  - 어렵지만 지적인 호기심을 자극하는 주제
  - 추상적인 특징 - 액션 종류가 작아서 쉽게 표현 가능

컴퓨터 게임의 역사



클로드 새넌(MIT)
“정보 이론의 아버지”
미국 국가 과학 메달(1966)

1950 클로드 새넌, 체스 두는 컴퓨터 프로그래밍

1956 존 매카시, 알파-베타 탐색 고안

1982 BELLE, 마스터 등급을 달성한 최초의 체스 프로그램

1984 쥬데아 펄, *휴리스틱*

1997 딥 블루(IBM), 세계 체스 챔피언 게리 카스파로프를 이김

2016 알파고, 이세돌을 이김

- 시각적 패턴 인식
- 강화 학습
- 신경망
- 몬테칼로 트리 탐색

2018 알파제로, 바둑/체스/장기 등 최고 프로그램들 모두 이김

2019 Pluribus (CMU), 6명의 최고 선수와 텍사스 홀덤에서 승리

게임 유형

- 결정론적 / 완전 정보 (체스, 바둑, 체커) 유형의 게임

- 확률론적 게임 (예: 백가몬, 윷놀이)

  플레이어의 의도에 확률이 추가되어 게임 진행(전이 모델)

- 부분적으로 관측가능한 게임 (예: 브릿지, 포커)

II. 2명이 하는 게임

- 완전 정보 – 전체 관측 가능
- 제로 섬 – 한 플레이어에게 유리한 것은 상대방에게는 불리함

말의 움직임 ⇔ 액션
말의 위치 ⇔ 상태

최대와 최소: 두 명의 플레이어를 나타냄

게임의 공식적인 정의

- s_0: 초기 상태 – 게임 셋업

임의의 상태 s에서:

- TO-MOVE(s): 다음 차례 수를 놓을 플레이어
- ACTIONS(s): 특정 상태에서 취할 수 있는 적법한 움직임 집합
- RESULT(s, a): 특정 상태 s에서 액션 a를 취했을 때 다음 상태
- IS-TERMINAL(s): 게임이 끝났는지 검사 (즉, s가 종료 상태인지)
- UTILITY (s, p): 효용성 함수. 게임이 종료 상태 s에서 끝났을 때 플레이어 p가 갖는 값

  예: 체스에서 이기면 1, 지면 0, 비기면 1/2

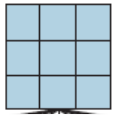
  게임 참가자들에 주어지는 값의 총합은 일정(zero-sum):

  $$1 + 0 = 0 + 1 = \frac{1}{2} + \frac{1}{2} = 1$$

상태 공간 그래프 (틱-택-토)

노드 ↔ 상태, 에지 ↔ 움직임

MAX (**x**)

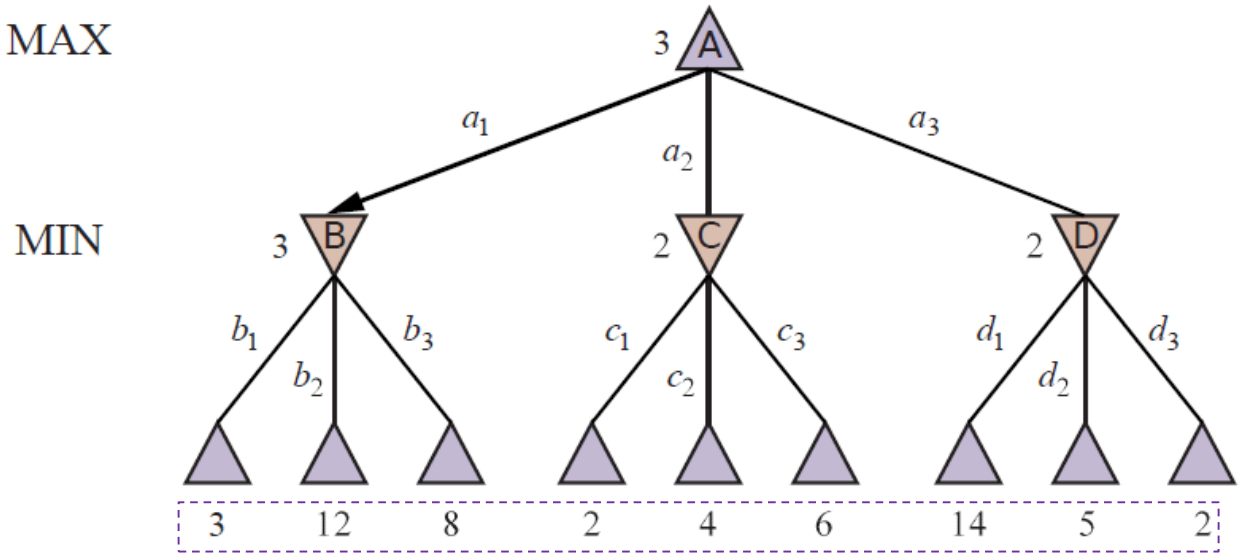


$9! = 362{,}880$ 개 종료 노드
(5,478 구분되는 상태)

10^{40} , 체스의 경우!

두 겹 게임 트리

겹(Ply): 한 플레이어에 의한 한 번의 움직임



효용성 값(Utility values)

최적 전략

트리를 구성하는 모든 상태 s에 대한 각각의 최소 최대값을 계산

$$\text{MINIMAX}(s)$$

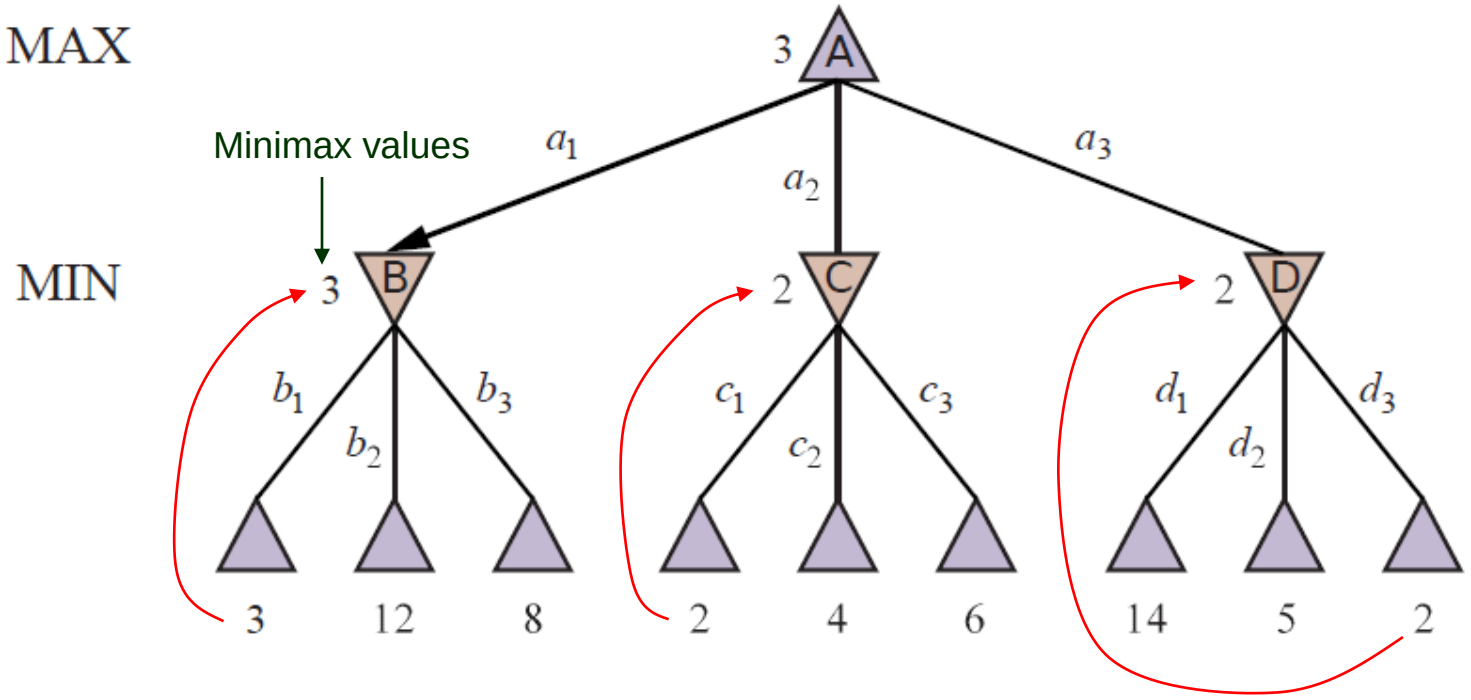
양측 모두 최적의 게임을 한다고 가정:

- MAX 측은 자기 차례가 오면 최대값을 갖는 상태로 이동
- MIN 측은 자기 차례에 최소값을 갖는 상태로 이동

$$\text{MINIMAX}(s) = \begin{cases} \text{UTILITY}(s, \text{MAX}) & \text{if IS-TERMINAL}(s) \\ \max_{a \in Actions(s)} \text{MINIMAX}(\text{RESULT}(s, a)) & \text{if TO-MOVE}(s) = \text{MAX} \\ \min_{a \in Actions(s)} \text{MINIMAX}(\text{RESULT}(s, a)) & \text{if TO-MOVE}(s) = \text{MIN} \end{cases}$$

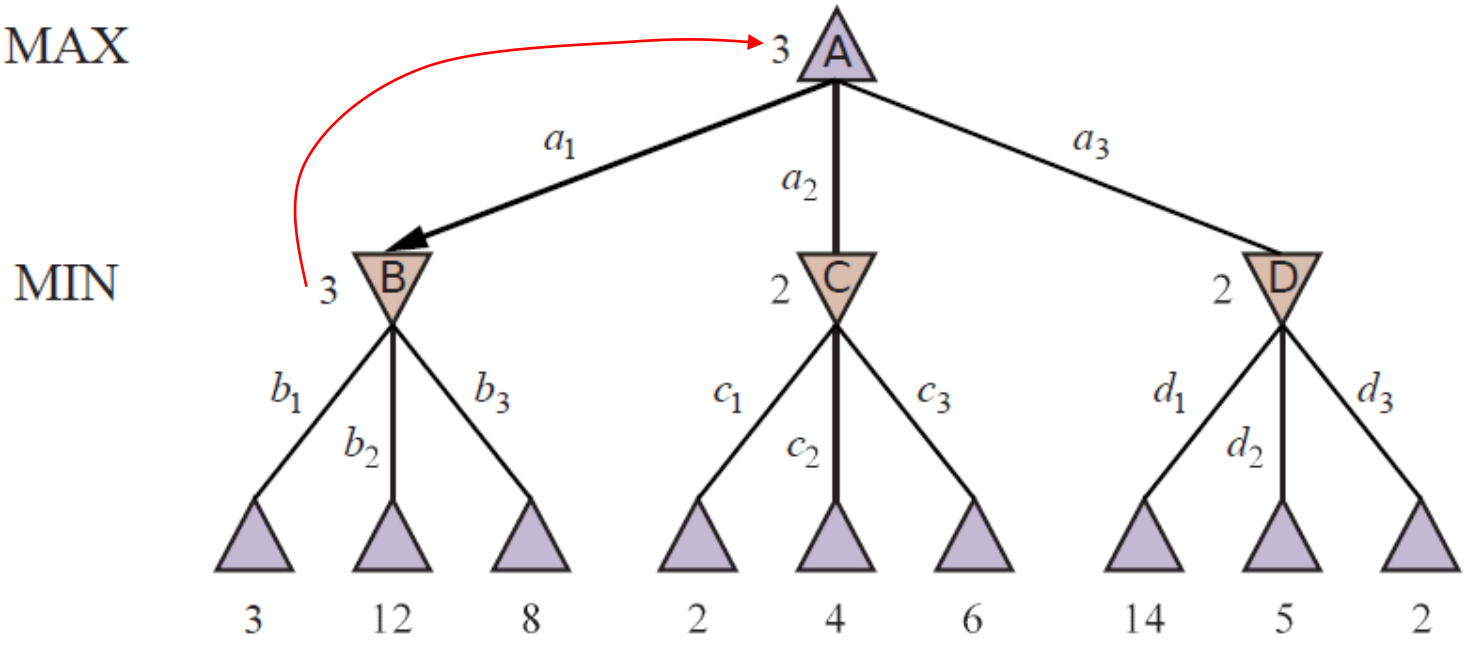
MIN 노드에서의 최소최대 값

MIN: 최저값을 갖는 MAX 노드로 이동을 선택

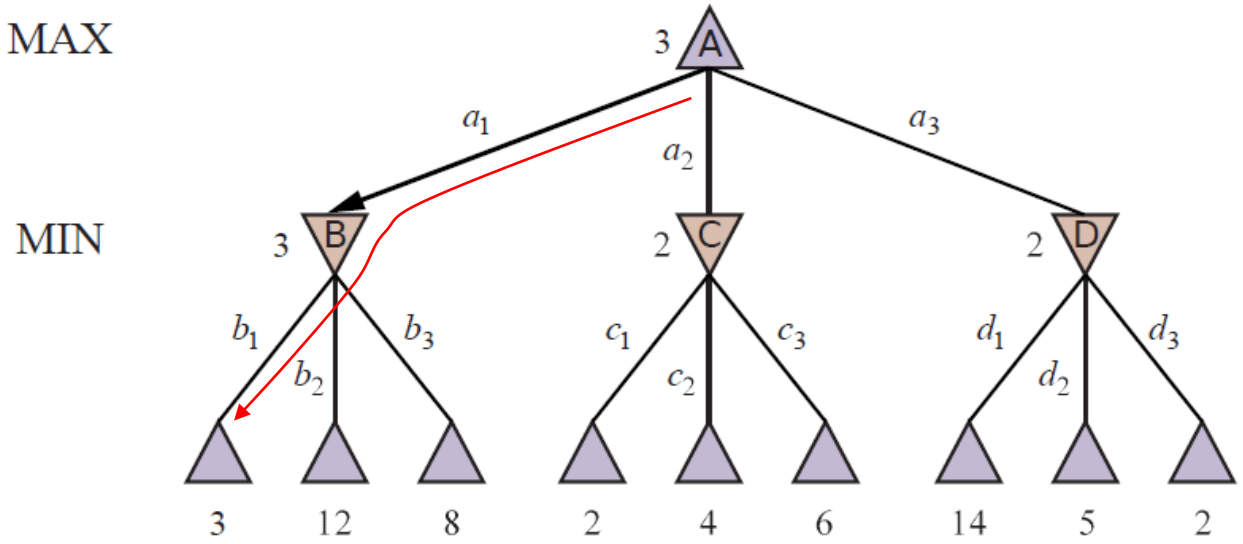


MAX 노드에서의 최소최대 값

MAX: 최고값을 갖는 MIN노드로의 이동을 선택



게임의 솔루션



MAX가 취할 수 있는 최적의 이동: a_1

그에 대응하여 MIN이 취할 수 있는 최적의 이동: b_1

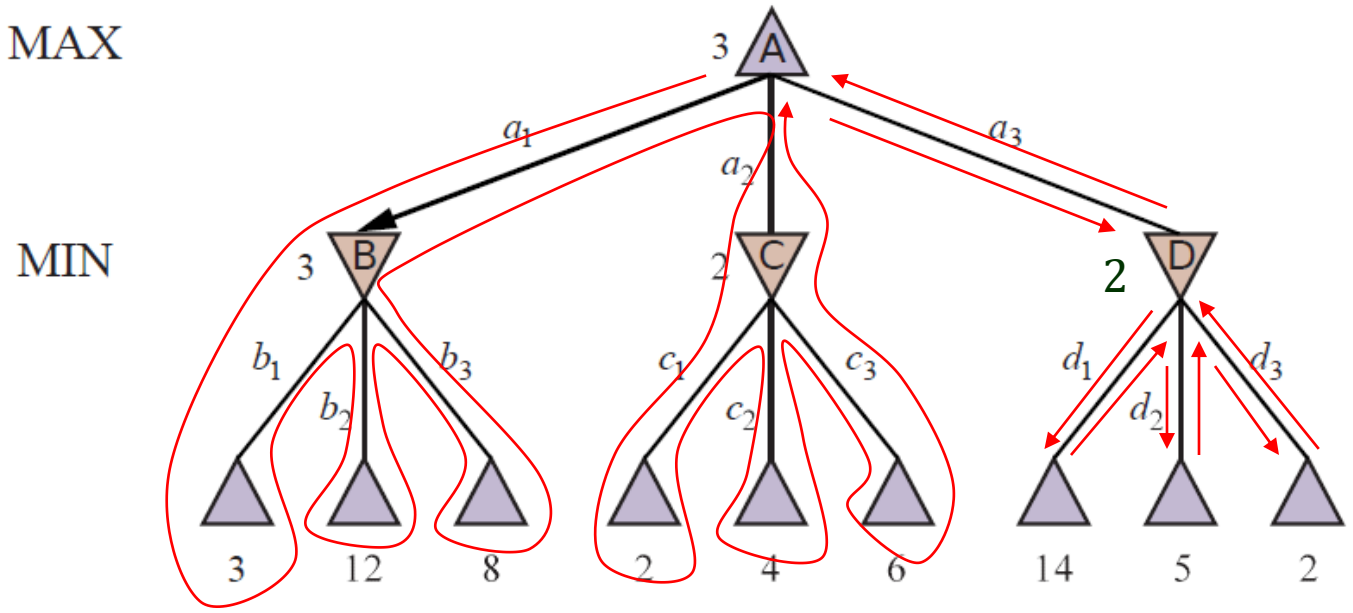
최소최대 탐색 알고리즘

```
function MINIMAX-SEARCH(game, state) returns an action
  player ← game.TO-MOVE(state)   // 다음 차례 알려주는 함수
  if player = MAX then value, move ← MAX-VALUE( game, state )
                  else  value, move ← MIN-VALUE( game, state )
  return move
function MAX-VALUE(game, state) returns a (utility, move) pair
  if game.IS-TERMINAL(state) then return game.UTILITY(state, player), null
  v ← −∞
  for each a in game.ACTIONS(state) do
    v2, a2 ← MIN-VALUE(game, game.RESULT(state, a))
    if v2 > v then
      v, move ← v2, a
  return v, move

function MIN-VALUE(game, state) returns a (utility, move) pair
  if game.IS-TERMINAL(state) then return game.UTILITY(state, player), null
  v ← +∞
  for each a in game.ACTIONS(state) do
    v2, a2 ← MAX-VALUE(game, game.RESULT(state, a))
    if v2 < v then
      v, move ← v2, a
  return v, move
```

알고리즘 실행

Leaf 노드 방문 후 돌아올 때 Utility 값을 갖고 오는 깊이 우선 탐색



최소최대 알고리즘 요약

// 양측 선수를 위한 최적해를 반드시 찾을 수 있음. 전체 게임트리를 탐험할 수 있는 충분한 시간이 주어진다면.

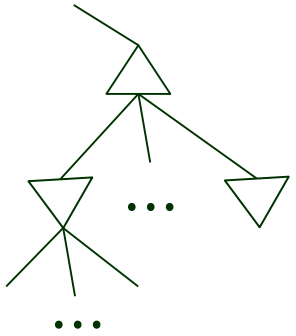
- 게임 트리가 유한하다면 <u>완전함</u>
- 최적 전략을 취하는 상대에 대해 최적

  만약 MIN 측이 최적의 게임을 하지 않는다면,
  1) MAX는 최소한 최적의 플레이어 만큼의 게임을 실행
  2) 하지만 최적이 아닌 MIN 측을 상대하기 위한 더 나은 전략이 있을 수 있음

- 복잡도:

가지 수 → b, 최대 깊이 → m

Time: $O(b^m)$ Space: $O(bm)$



일반적으로 체스 게임 평균: $b \approx 35$, $m \approx 100$
정확한 최적해를 구하는 것은 비현실적임!

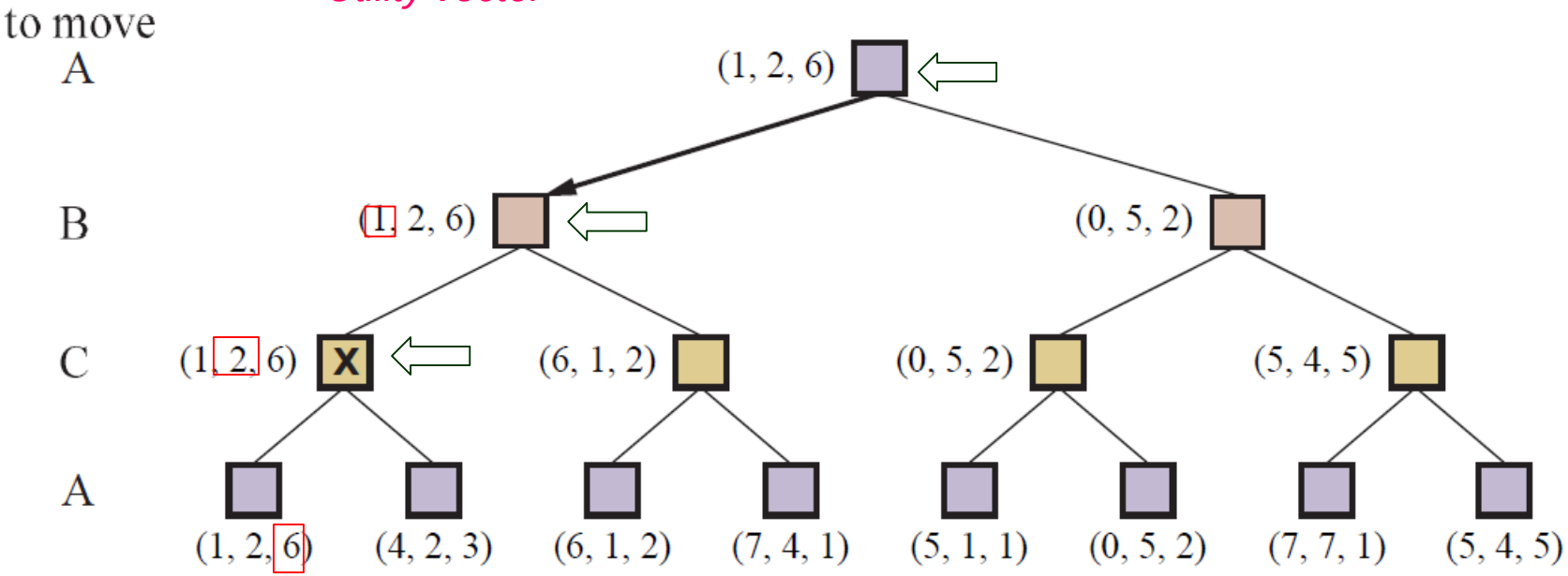
다중 플레이어 게임

최소최대 알고리즘 확장:

- 노드는 단일값 대신 *벡터*값을 가짐

$\langle v_A, v_B, v_C \rangle$ for three players A, B, C

Utility vector



노드 n의 백업 값 = n에서 선택할 수 있는 다음 상태 중 가장 높은 값을 갖는 상태에 대한 유틸리티 벡터