CSE 411: Artificial Intelligence (Elective Course #6)

400 Level, Mechatronics Engineering
2^{nd} Term 2016/2017, Lecture #6

Hazem Shehata

Dept. of Computer & Systems Engineering
Zagazig University

April 3^{rd}, 2017

Credits to Dr. Mohamed El Abd for the slides

Adminstrivia

Notes

- Assignment #2:
  - Due on Thursday.

Course Info:

- Website: http://hshehata.github.io/courses/zu/cse411/
- Office hours: Sunday 11:30am - 12:30pm

Adminstrivia

Notes

- Assignment #2:
  - Due on Thursday.
- Assignment #3:
  - Released Today.
  - Due on Sunday, April 16, 2017.

Course Info:

- Website: http://hshehata.github.io/courses/zu/cse411/
- Office hours: Sunday 11:30am - 12:30pm

Outline

Outline

1. Adversarial Search

2. Requirements & Reading Material

Adversarial search

Introduction

- Multi-agent environments:
  - Cooperative.
  - Competitive.

Adversarial search

Introduction

- Multi-agent environments:
  - Cooperative.
  - Competitive.
- In competitive environments, different agents have conflicting goals.

Adversarial search

Introduction

- Multi-agent environments:
  - Cooperative.
  - Competitive.
- In competitive environments, different agents have conflicting goals.
- This gives rise to ***adversarial search***, also known as ***games***.

Adversarial search

Introduction

- In games, other agents:
  - Want our agent to lose.
  - Introduce uncertainty, don't know what will they do.

Adversarial search

Introduction

- In games, other agents:
  - Want our agent to lose.
  - Introduce uncertainty, don't know what will they do.
- The unpredictability of other agents can introduce many possible ***contingencies***.

Adversarial search

Introduction

- In games, other agents:
  - Want our agent to lose.
  - Introduce uncertainty, don't know what will they do.
- The unpredictability of other agents can introduce many possible ***contingencies***.
- We will consider a specific type:
  - Deterministic.
  - Fully observable.
  - Two agents with alternating actions.

Adversarial search

	deterministic	chance
perfect information	chess, checkers, go, othello	backgammon monopoly
imperfect information	battleships, blind tictactoe	bridge, poker, scrabble nuclear war

Adversarial search

Introduction

- The main problem with games is that the search space is too huge:
  - A search tree for chess has an average branching factor of 35.
  - An average chess game lasts for 50 moves per player.
  - The average search tree has 35^{100} nodes.

Adversarial search

Introduction

- The main problem with games is that the search space is too huge:
  - A search tree for chess has an average branching factor of 35.
  - An average chess game lasts for 50 moves per player.
  - The average search tree has 35^{100} nodes.
- There is no enough time to search the whole tree and calculate the exact consequences of every move.

Adversarial search

Introduction

- The main problem with games is that the search space is too huge:
  - A search tree for chess has an average branching factor of 35.
  - An average chess game lasts for 50 moves per player.
  - The average search tree has 35^{100} nodes.
- There is no enough time to search the whole tree and calculate the exact consequences of every move.
- If the optimal decision is not feasible (due to time or memory constraints), some decision has to be taken.

Adversarial search

Introduction

We will investigate:

- What is an optimal move and how to find it.

Adversarial search

Introduction

We will investigate:

- What is an optimal move and how to find it.
- Pruning the search tree.

Adversarial search

Introduction

We will investigate:

- What is an optimal move and how to find it.
- Pruning the search tree.
- How to choose a good move when time is limited.

Adversarial search

Introduction

We will investigate:

- What is an optimal move and how to find it.
- Pruning the search tree.
- How to choose a good move when time is limited.
- How to define a heuristic evaluation function to approximate the utility of a state.

Adversarial search

Minimax Algorithm

- First thing to consider is how to make an optimal move.

Adversarial search

Minimax Algorithm

- First thing to consider is how to make an optimal move.
- Certain assumptions exist:
  - A two player game.
  - The play is sequential (agents taking turns).
  - The opponent is playing perfectly.

Adversarial search

Minimax Algorithm

- First thing to consider is how to make an optimal move.
- Certain assumptions exist:
  - A two player game.
  - The play is sequential (agents taking turns).
  - The opponent is playing perfectly.
- The two players are referred to as **MAX** and **MIN**:
  - **MAX** wants to maximize the utility.
  - **MIN** wants to minimize the utility.

Adversarial search

Minimax Algorithm

- A **game** can be formally defined as a kind of search problem with the following elements :

Adversarial search

Minimax Algorithm

- A **game** can be formally defined as a kind of search problem with the following elements :
  - S_0: **initial state**; how game is set up at start.

Adversarial search

Minimax Algorithm

- A **game** can be formally defined as a kind of search problem with the following elements :
  - S_0: **initial state**; how game is set up at start.
  - $\text{PLAYER}(s)$: which player has the move in a state.

Adversarial search

Minimax Algorithm

- A **game** can be formally defined as a kind of search problem with the following elements :
  - S_0: **initial state**; how game is set up at start.
  - PLAYER(s): which player has the move in a state.
  - ACTIONS(s): set of legal moves in a state.

Adversarial search

Minimax Algorithm

- A **game** can be formally defined as a kind of search problem with the following elements :
  - S_0: **initial state**; how game is set up at start.
  - $\text{PLAYER}(s)$: which player has the move in a state.
  - $\text{ACTIONS}(s)$: set of legal moves in a state.
  - $\text{RESULT}(s, a)$: **transition model**; result of a move.

Adversarial search

Minimax Algorithm

- A **game** can be formally defined as a kind of search problem with the following elements :
  - S_0: **initial state**; how game is set up at start.
  - PLAYER(s): which player has the move in a state.
  - ACTIONS(s): set of legal moves in a state.
  - RESULT(s, a): **transition model**; result of a move.
  - TERMINAL-TEST(s): **terminal test**; true if and only if the game is over. States where game ends are called **terminal states**.

Adversarial search

Minimax Algorithm

- A **game** can be formally defined as a kind of search problem with the following elements :
  - S_0: **initial state**; how game is set up at start.
  - PLAYER(s): which player has the move in a state.
  - ACTIONS(s): set of legal moves in a state.
  - RESULT(s, a): **transition model**; result of a move.
  - TERMINAL-TEST(s): **terminal test**; true if and only if the game is over. States where game ends are called **terminal states**.
  - UTILITY(s): **utility function** (a.k.a., payoff function or objective function); final numeric value for a game that ends in a terminal state s.

Adversarial search

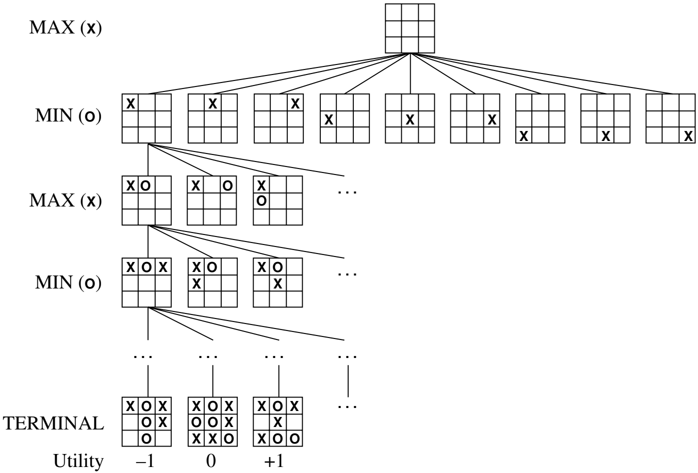
Minimax Algorithm

- A **game** can be formally defined as a kind of search problem with the following elements :
  - S_0: **initial state**; how game is set up at start.
  - PLAYER(s): which player has the move in a state.
  - ACTIONS(s): set of legal moves in a state.
  - RESULT(s, a): **transition model**; result of a move.
  - TERMINAL-TEST(s): **terminal test**; true if and only if the game is over. States where game ends are called **terminal states**.
  - UTILITY(s): **utility function** (a.k.a., payoff function or objective function); final numeric value for a game that ends in a terminal state s.
- S_0, ACTIONS, and RESULT define the ***game tree***.

Adversarial search

Example: game tree for tic-tac-toe

A partial game tree for the game of tic-tac-toe:



Adversarial search

Minimax Algorithm

- **MAX** is the first player to move, starting at the root node.

Adversarial search

Minimax Algorithm

- **MAX** is the first player to move, starting at the root node.

- The utility values at the leaf nodes are from **MAX's** point of view.

Adversarial search

Minimax Algorithm

- **MAX** is the first player to move, starting at the root node.
- The utility values at the leaf nodes are from **MAX's** point of view.
- The optimal strategy is found by examining the ***minimax value*** of each node.

Adversarial search

Minimax Algorithm

- In a normal search tree, a solution is to find a path from initial state to goal state (*i.e.*, a wining state for **MAX**).

Adversarial search

Minimax Algorithm

- In a normal search tree, a solution is to find a path from initial state to goal state (*i.e.*, a wining state for **MAX**).
- However, since **MIN** affects the search, a contingent strategy should be found.

Adversarial search

Minimax Algorithm

- In a normal search tree, a solution is to find a path from initial state to goal state (*i.e.*, a wining state for **MAX**).
- However, since **MIN** affects the search, a contingent strategy should be found.
- The strategy defines:
  - **MAX's** move at the initial state.
  - **MAX's** move at all states generated by all possible moves by **MIN**.
  - ... and so on.

Adversarial search

Minimax Algorithm

- In a normal search tree, a solution is to find a path from initial state to goal state (*i.e.*, a wining state for **MAX**).
- However, since **MIN** affects the search, a contingent strategy should be found.
- The strategy defines:
  - **MAX's** move at the initial state.
  - **MAX's** move at all states generated by all possible moves by **MIN**.
  - ... and so on.
- This means that **MAX** needs a winning strategy independent of what **MIN** does.

Adversarial search

Minimax Algorithm

- **MAX** will choose a move that maximizes its utility value.

Adversarial search

Minimax Algorithm

- **MAX** will choose a move that maximizes its utility value.
- **MIN** will choose a move that minimizes the utility value of **MAX**.

Adversarial search

Minimax Algorithm

- **MAX** will choose a move that maximizes its utility value.
- **MIN** will choose a move that minimizes the utility value of **MAX**.
- This results in **MAX** (at the root) choosing its best move given available information.

Adversarial search

Minimax Algorithm

- **MAX** will choose a move that maximizes its utility value.
- **MIN** will choose a move that minimizes the utility value of **MAX**.
- This results in **MAX** (at the root) choosing its best move given available information.
- Available information is via look ahead (the search down the tree).

Adversarial search

Minimax Algorithm

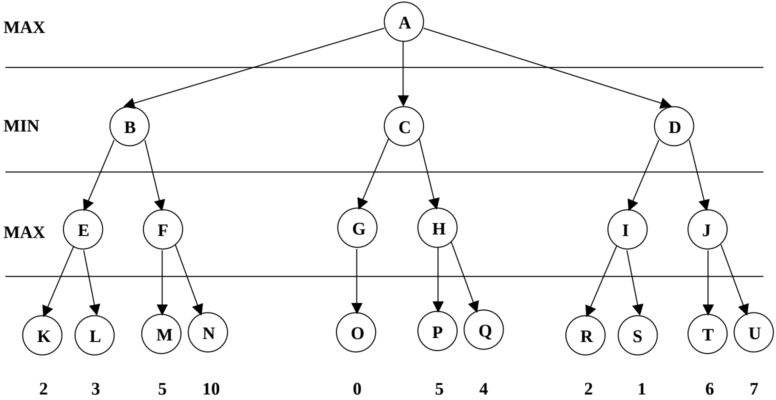
- The minimax value of a node is defined as follows:

$$\text{MINIMAX}(s) = \begin{cases} \text{UTILITY}(s) & \textbf{if } \text{TERMINAL-TEST}(s) \\ max_{a \in \text{ACTIONS}(s)} \text{MINIMAX}(\text{RESULT}(s,a)) & \textbf{if } \text{PLAYER}(s) = \text{MAX} \\ min_{a \in \text{ACTIONS}(s)} \text{MINIMAX}(\text{RESULT}(s,a)) & \textbf{if } \text{PLAYER}(s) = \text{MIN} \end{cases}$$

Adversarial search

Example: applying minimax

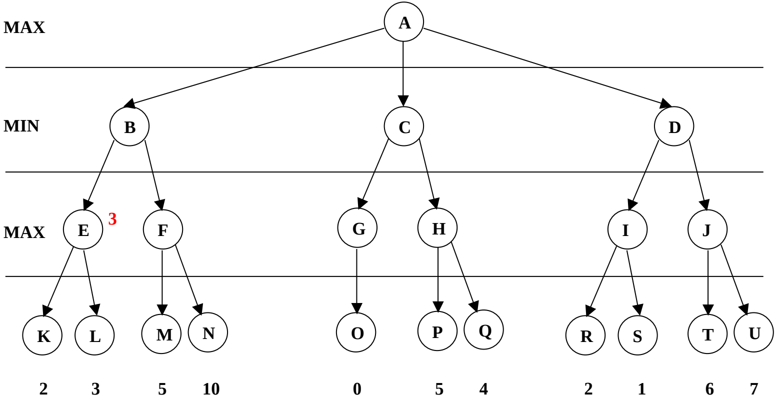
Calculate the minimax value for the node A.



Adversarial search

Example: applying minimax

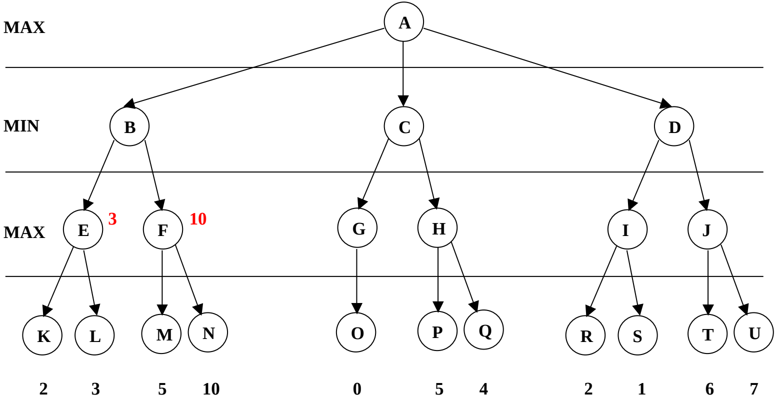
Calculate the minimax value for the node A.



Adversarial search

Example: applying minimax

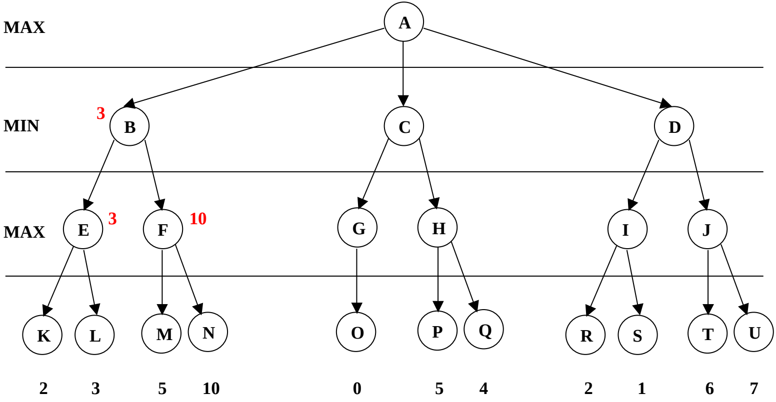
Calculate the minimax value for the node A.



Adversarial search

Example: applying minimax

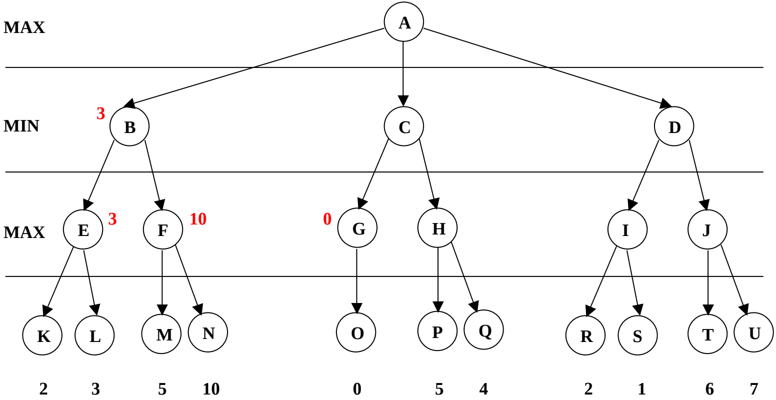
Calculate the minimax value for the node A.



Adversarial search

Example: applying minimax

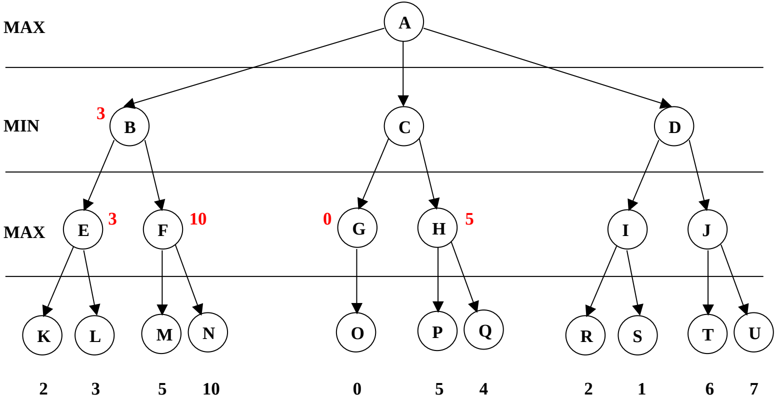
Calculate the minimax value for the node A.



Adversarial search

Example: applying minimax

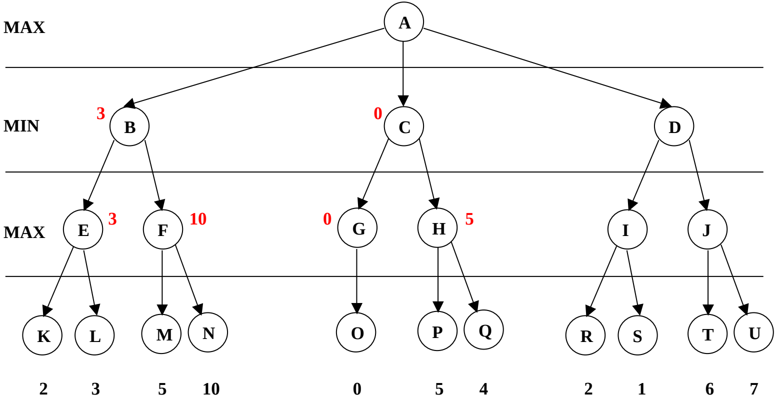
Calculate the minimax value for the node A.



Adversarial search

Example: applying minimax

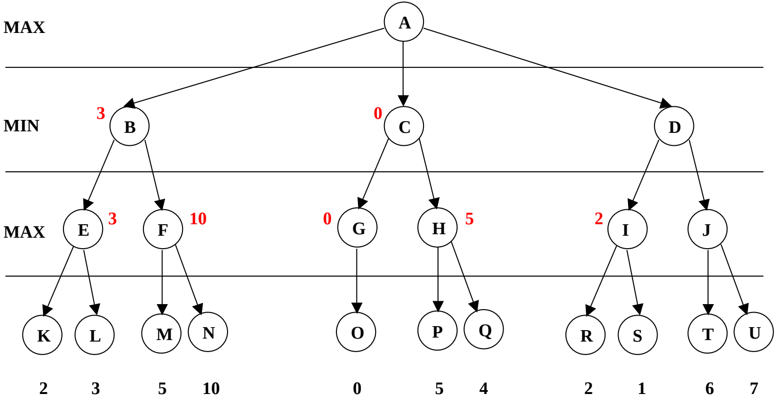
Calculate the minimax value for the node A.



Adversarial search

Example: applying minimax

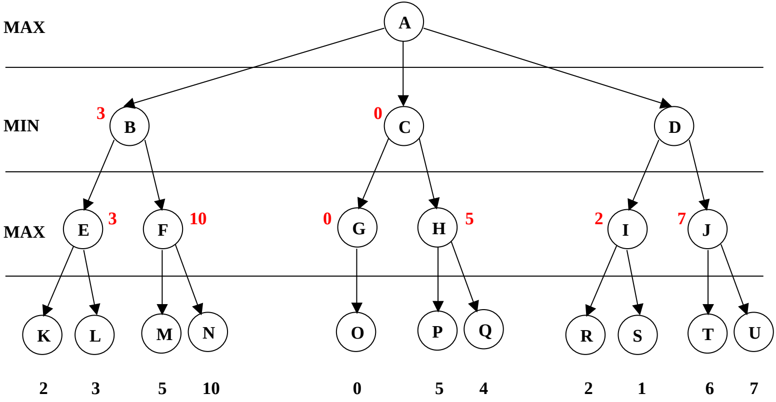
Calculate the minimax value for the node A.



Adversarial search

Example: applying minimax

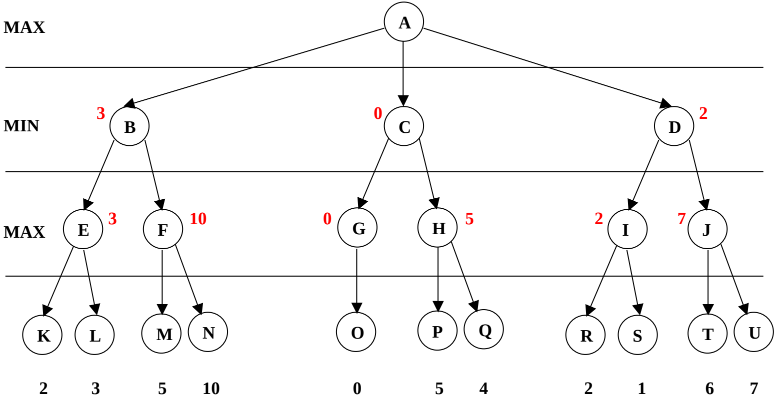
Calculate the minimax value for the node A.



Adversarial search

Example: applying minimax

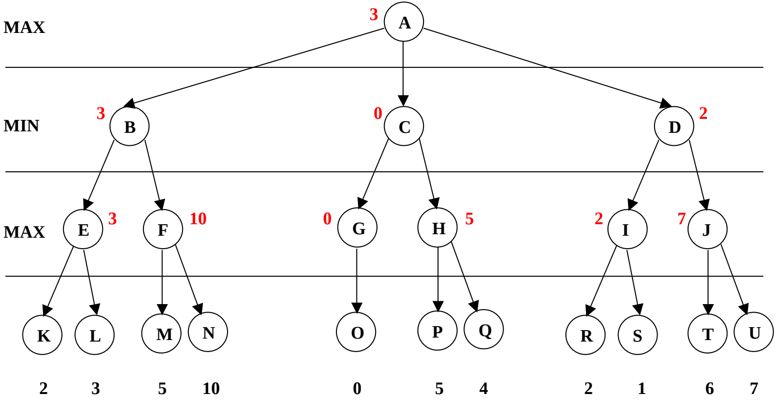
Calculate the minimax value for the node A.



Adversarial search

Example: applying minimax

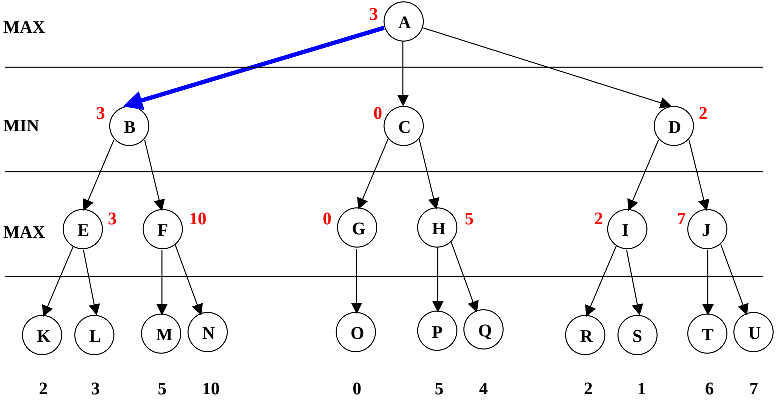
Calculate the minimax value for the node A.



Adversarial search

Example: applying minimax

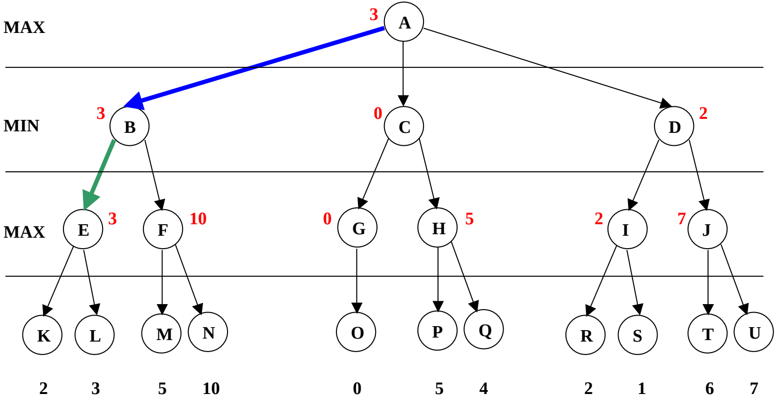
Calculate the minimax value for the node A.



Adversarial search

Example: applying minimax

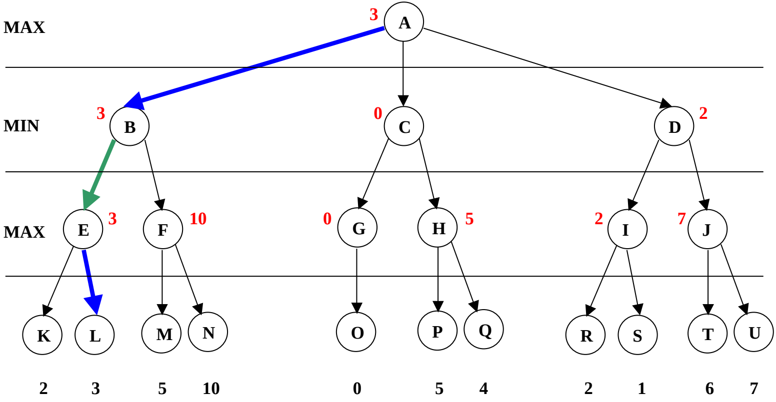
Calculate the minimax value for the node A.



Adversarial search

Example: applying minimax

Calculate the minimax value for the node A.



Adversarial search

Example: applying minimax to NIM

- Consider the game of NIM:
  - 7 matches placed in a pile.
  - Each player divides a pile of matches into 2 non-empty piles with a different number of matches.
  - The player who cannot make a move is the loser.

Adversarial search

Example: applying minimax to NIM

- Consider the game of NIM:
  - 7 matches placed in a pile.
  - Each player divides a pile of matches into 2 non-empty piles with a different number of matches.
  - The player who cannot make a move is the loser.
- The utility is +1 (**MAX** wins) or -1 (**MAX** loses).

Adversarial search

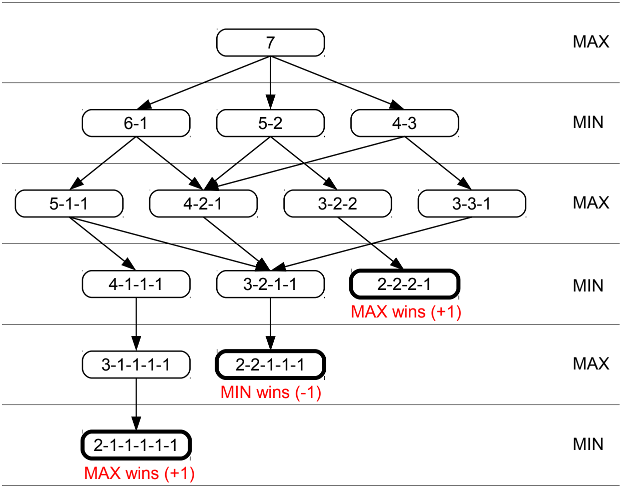
Example: applying minimax to NIM

- Consider the game of NIM:
  - 7 matches placed in a pile.
  - Each player divides a pile of matches into 2 non-empty piles with a different number of matches.
  - The player who cannot make a move is the loser.
- The utility is +1 (**MAX** wins) or -1 (**MAX** loses).
- The value at each node represents the best value of the best terminal state the current player can hope to achieve.

Adversarial search

Example: applying minimax to NIM

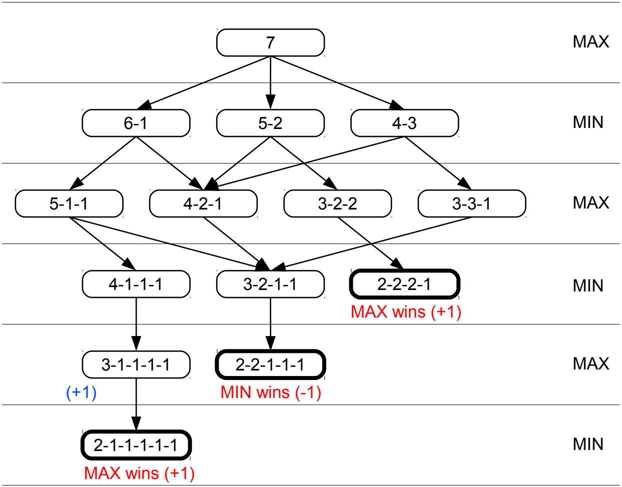
Calculate the minimax value for the root node.



Adversarial search

Example: applying minimax to NIM

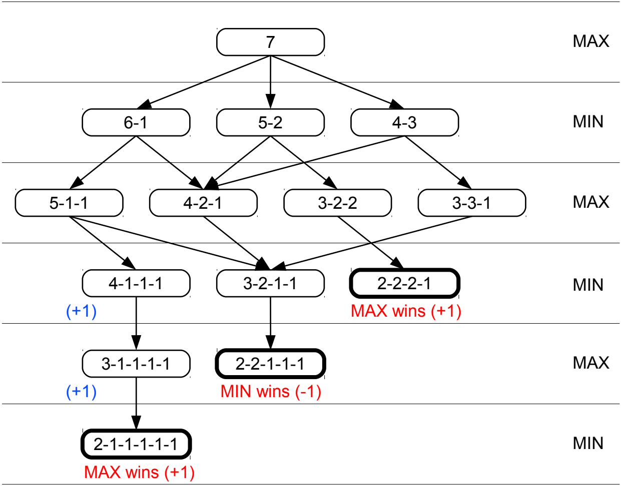
Calculate the minimax value for the root node.



Adversarial search

Example: applying minimax to NIM

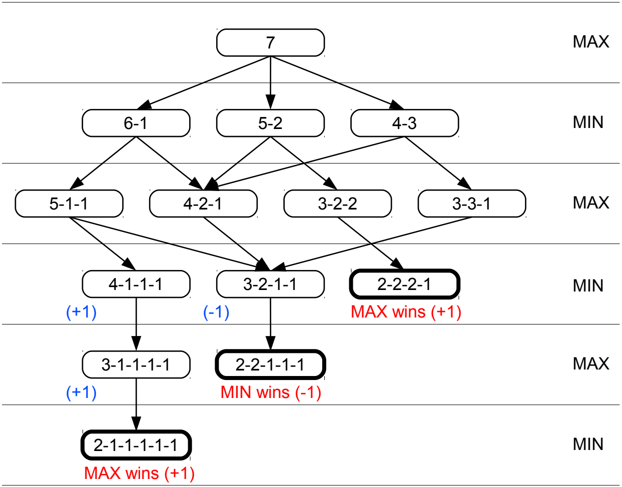
Calculate the minimax value for the root node.



Adversarial search

Example: applying minimax to NIM

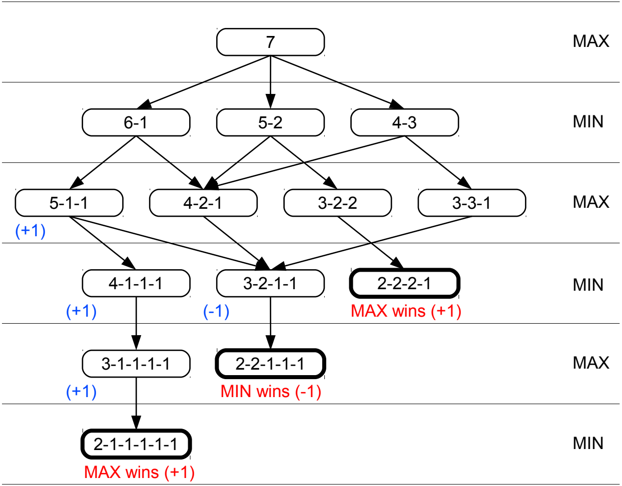
Calculate the minimax value for the root node.



Adversarial search

Example: applying minimax to NIM

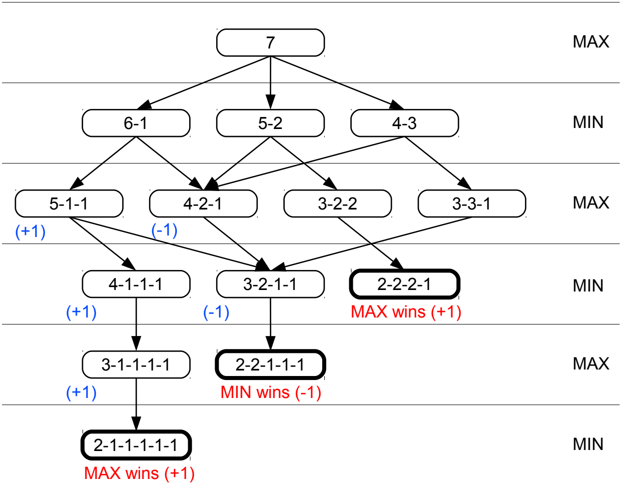
Calculate the minimax value for the root node.



Adversarial search

Example: applying minimax to NIM

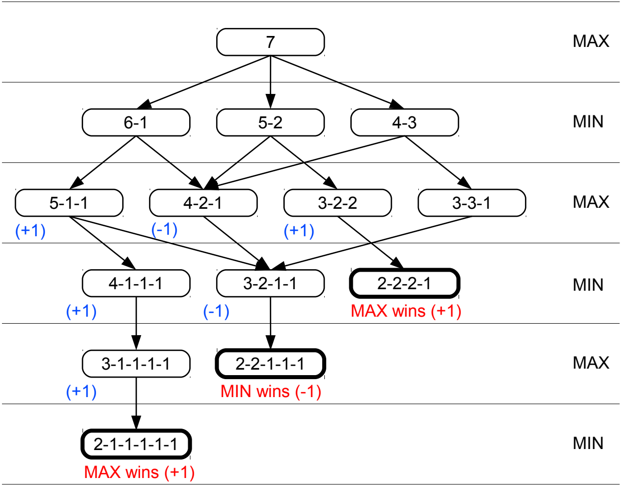
Calculate the minimax value for the root node.



Adversarial search

Example: applying minimax to NIM

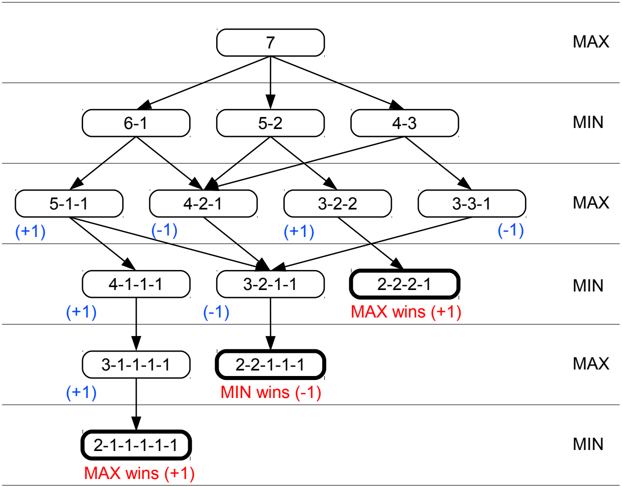
Calculate the minimax value for the root node.



Adversarial search

Example: applying minimax to NIM

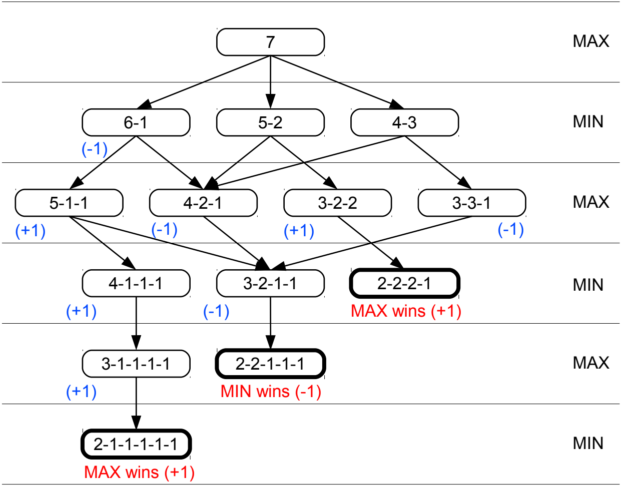
Calculate the minimax value for the root node.



Adversarial search

Example: applying minimax to NIM

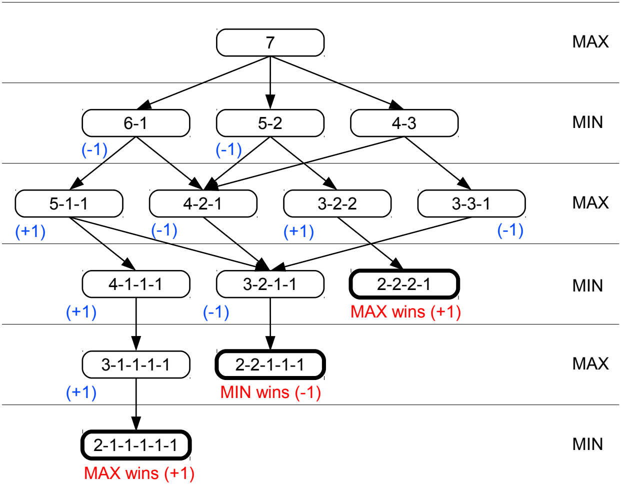
Calculate the minimax value for the root node.



Adversarial search

Example: applying minimax to NIM

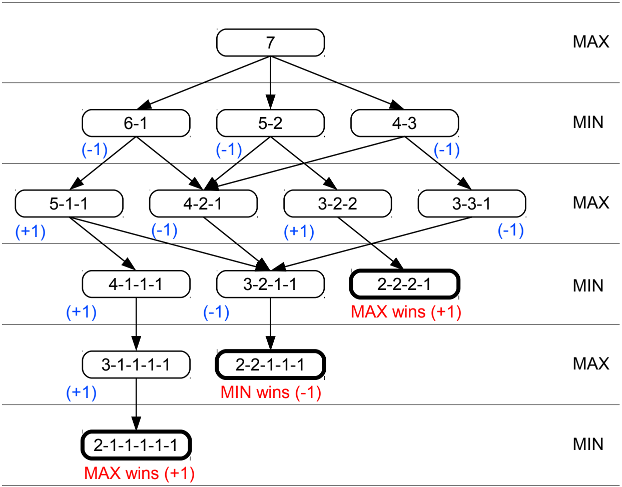
Calculate the minimax value for the root node.



Adversarial search

Example: applying minimax to NIM

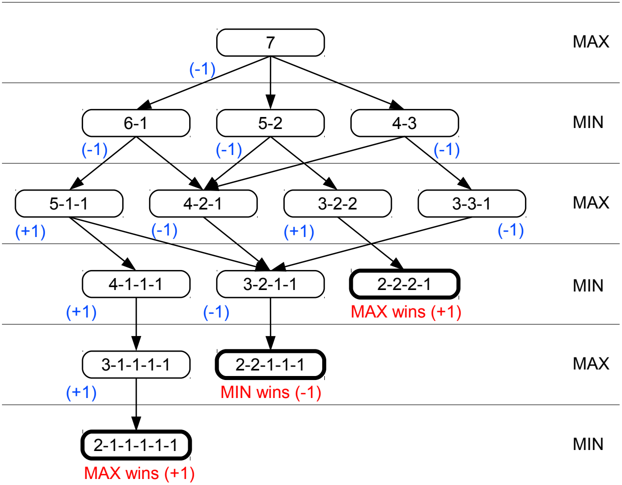
Calculate the minimax value for the root node.



Adversarial search

Example: applying minimax to NIM

Calculate the minimax value for the root node.



Adversarial search

The minimax search algorithm

function MINIMAX-SEARCH(s) **returns** an action
  $v \leftarrow$ MAX-VALUE(s)
  return the **action** in ACTIONS(s) with value v

Adversarial search

The minimax search algorithm

function MINIMAX-SEARCH(s) **returns** an action
    $v \leftarrow$ MAX-VALUE(s)
    return the **action** in ACTIONS(s) with value v

---

function MAX-VALUE(s) **returns** a utility value
    if TERMINAL-TEST(s) **then return** UTILITY(s)
    $v \leftarrow -\infty$
    for each a **in** ACTIONS(s) **do**
        $v \leftarrow$ MAX(v,MIN-VALUE(RESULT(s,a)))
    return v

Adversarial search

The minimax search algorithm

function MINIMAX-SEARCH(*s*) **returns** an action
  $v \leftarrow$ MAX-VALUE(*s*)
  return the **action** in ACTIONS(*s*) with value *v*

---

function MAX-VALUE(*s*) **returns** a utility value
  if TERMINAL-TEST(*s*) **then return** UTILITY(*s*)
  $v \leftarrow -\infty$
  for each *a* **in** ACTIONS(*s*) **do**
    $v \leftarrow$ MAX(*v*,MIN-VALUE(RESULT(*s*,*a*)))
  return *v*

---

function MIN-VALUE(*s*) **returns** a utility value
  if TERMINAL-TEST(*s*) **then return** UTILITY(*s*)
  $v \leftarrow +\infty$
  for each *a* **in** ACTIONS(*s*) **do**
    $v \leftarrow$ MIN(*v*,MAX-VALUE(RESULT(*s*,*a*)))
  return *v*

Adversarial search

Properties of the minimax search algorithm

- Complete if the tree is finite.

Adversarial search

Properties of the minimax search algorithm

- Complete if the tree is finite.
- Optimal against an optimal opponent.

Adversarial search

Properties of the minimax search algorithm

- Complete if the tree is finite.
- Optimal against an optimal opponent.
  - No other strategy could do any better in this case.

Adversarial search

Properties of the minimax search algorithm

- Complete if the tree is finite.
- Optimal against an optimal opponent.
  - No other strategy could do any better in this case.
  - For non-optimal opponents, outcome cannot be worse but there could be another strategy with better outcome!

Adversarial search

Properties of the minimax search algorithm

- Complete if the tree is finite.
- Optimal against an optimal opponent.
  - No other strategy could do any better in this case.
  - For non-optimal opponents, outcome cannot be worse but there could be another strategy with better outcome!
- Time complexity = $O(b^m)$.

Adversarial search

Properties of the minimax search algorithm

- Complete if the tree is finite.
- Optimal against an optimal opponent.
  - No other strategy could do any better in this case.
  - For non-optimal opponents, outcome cannot be worse but there could be another strategy with better outcome!
- Time complexity = $O(b^m)$.
- Space complexity = $O(bm)$.

Adversarial search

α-β Pruning

- Game trees are too huge to be solved to optimality.

Adversarial search

α-β Pruning

- Game trees are too huge to be solved to optimality.
- Minimax has to generate the entire tree, compute the utility values at the leaf nodes, and then propagate these values up the tree.

Adversarial search

α-β Pruning

- Game trees are too huge to be solved to optimality.
- Minimax has to generate the entire tree, compute the utility values at the leaf nodes, and then propagate these values up the tree.
- We want a way to do less work by removing unnecessary branches thus exploring less nodes.

Adversarial search

α-β Pruning

- Game trees are too huge to be solved to optimality.
- Minimax has to generate the entire tree, compute the utility values at the leaf nodes, and then propagate these values up the tree.
- We want a way to do less work by removing unnecessary branches thus exploring less nodes.
- However, we want to guarantee the same decision as Minimax.

Adversarial search

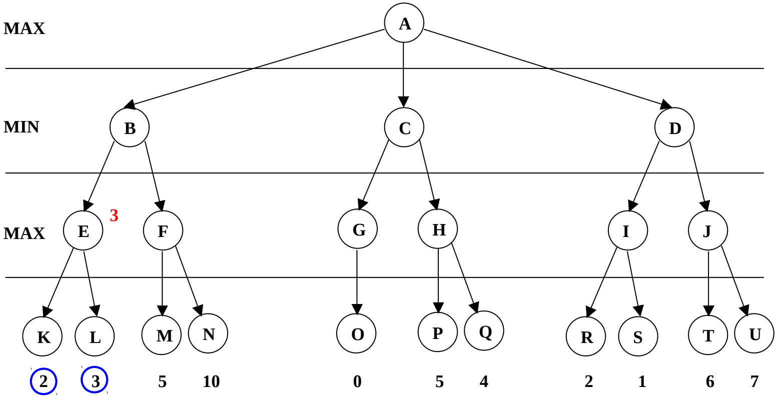
α-β Pruning

- Game trees are too huge to be solved to optimality.
- Minimax has to generate the entire tree, compute the utility values at the leaf nodes, and then propagate these values up the tree.
- We want a way to do less work by removing unnecessary branches thus exploring less nodes.
- However, we want to guarantee the same decision as Minimax.
- We use the **alpha-beta (α-β) pruning** technique.

Adversarial search

Example: illustrating α-β pruning

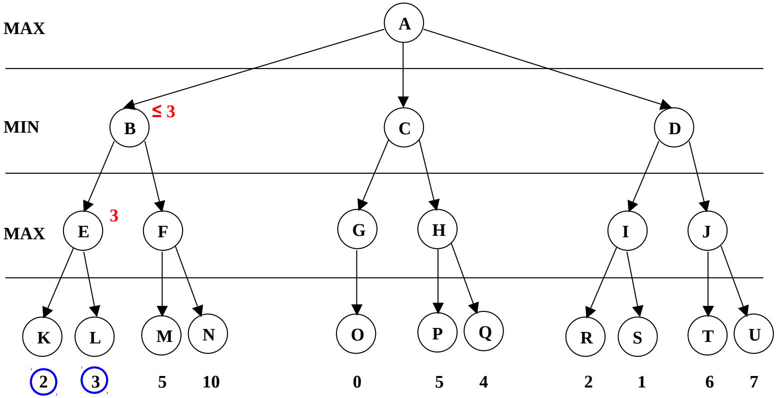
Show a scenario where α-β pruning becomes applicable.



Adversarial search

Example: illustrating α-β pruning

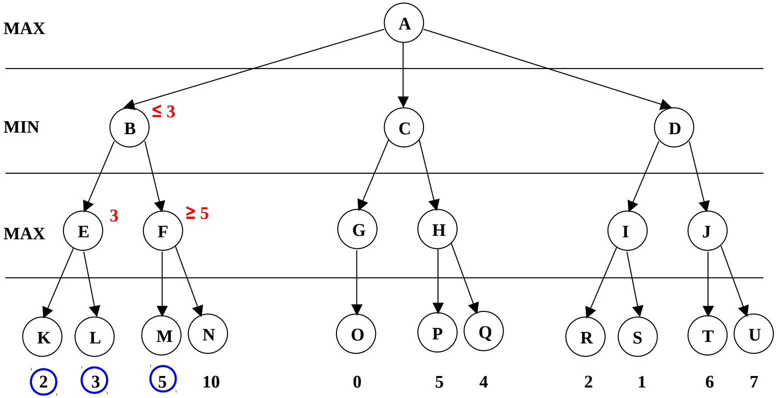
Show a scenario where α-β pruning becomes applicable.



Adversarial search

Example: illustrating α-β pruning

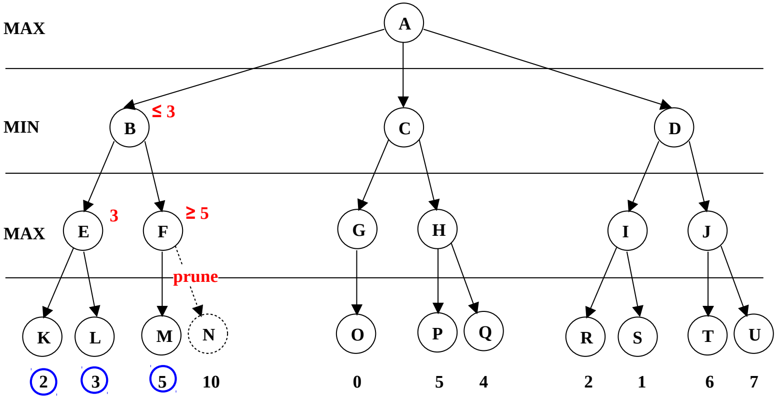
Show a scenario where α-β pruning becomes applicable.



Adversarial search

Example: illustrating α-β pruning

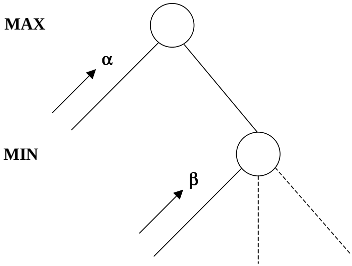
Show a scenario where α-β pruning becomes applicable.



Adversarial search

α-cutoff

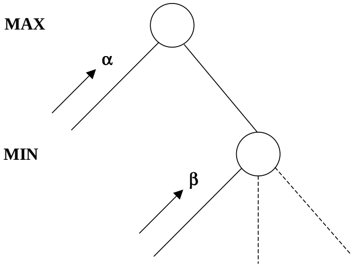
- α is the best (maximum) value **MAX** is assured so far off the current path.



Adversarial search

α-cutoff

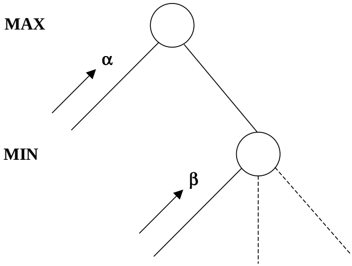
- α is the best (maximum) value **MAX** is assured so far off the current path.
- If β is worse than α (from **MAX's** point of view), **MAX** will avoid it, dashed branches will be pruned.



Adversarial search

α-cutoff

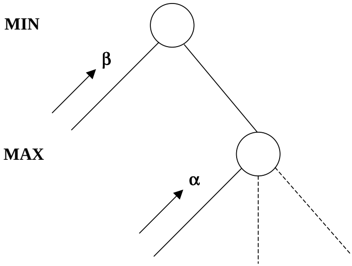
- α is the best (maximum) value **MAX** is assured so far off the current path.
- If β is worse than α (from **MAX's** point of view), **MAX** will avoid it, dashed branches will be pruned.
- Happens when $\alpha \geq \beta$, referred to as α-cutoff.



Adversarial search

β-cutoff

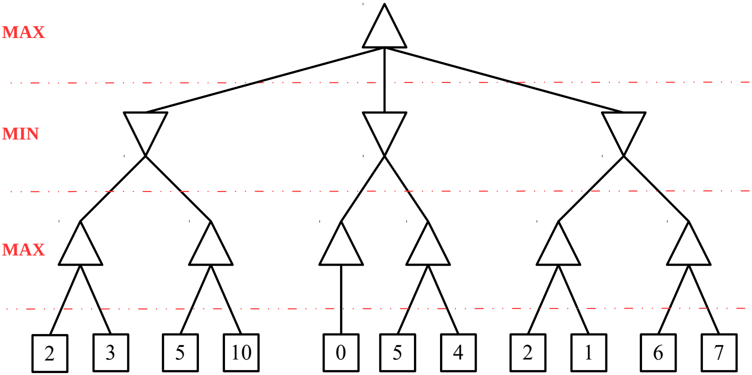
- β is the best (minimum) value **MIN** is assured so far off the current path.
- If α is worse than β (from **MIN's** point of view), **MIN** will avoid it, dashed branches will be pruned.
- Happens when $\alpha \geq \beta$, referred to as β-cutoff.



Adversarial search

Example: applying α-β pruning

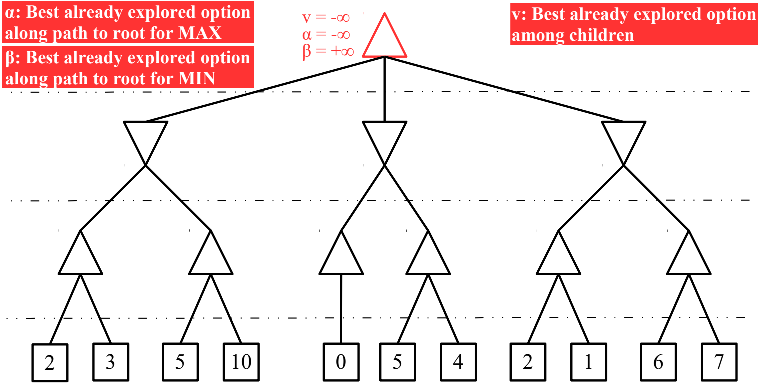
Apply α-β pruning to calculate minimax value for node A.



Adversarial search

Example: applying α-β pruning

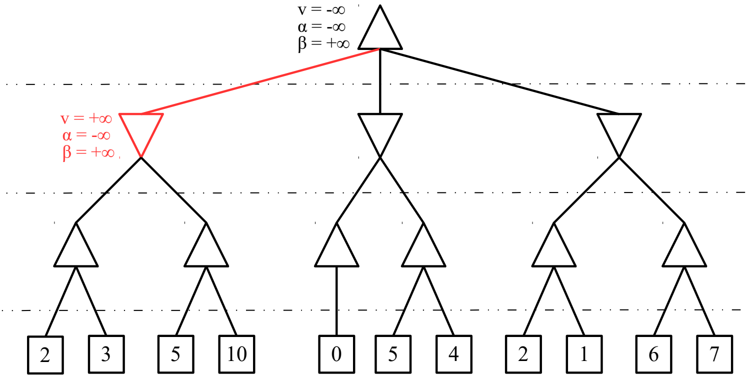
Apply α-β pruning to calculate minimax value for node A.



Adversarial search

Example: applying α-β pruning

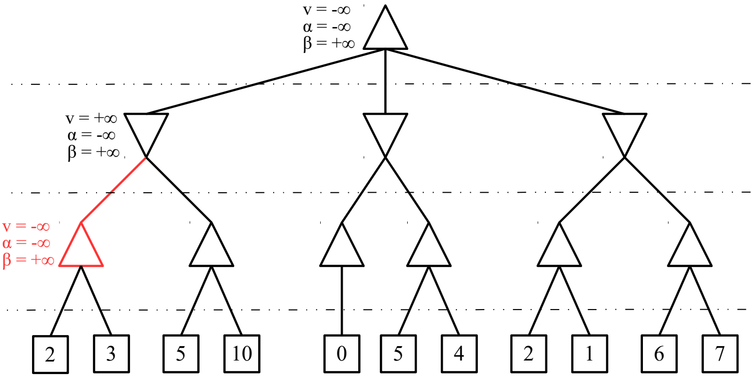
Apply α-β pruning to calculate minimax value for node A.



Adversarial search

Example: applying α-β pruning

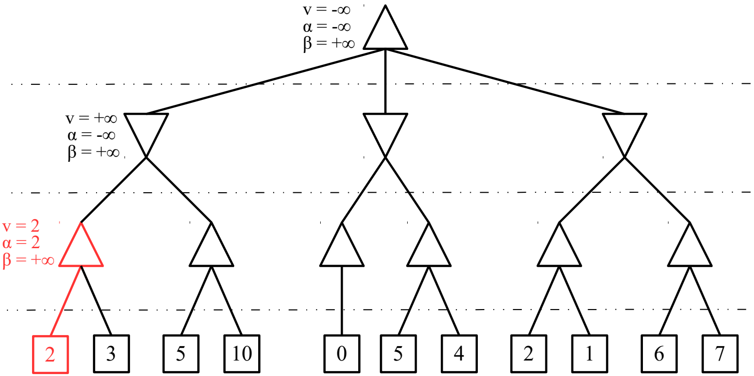
Apply α-β pruning to calculate minimax value for node A.



Adversarial search

Example: applying α-β pruning

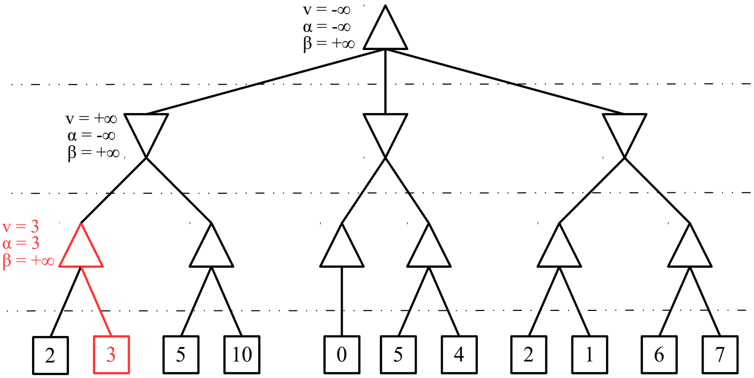
Apply α-β pruning to calculate minimax value for node A.



Adversarial search

Example: applying α-β pruning

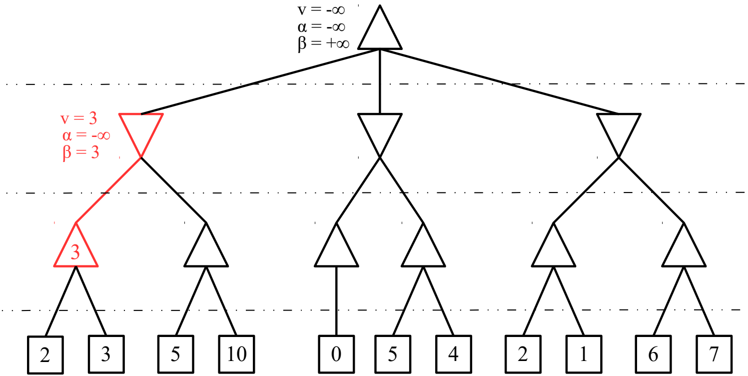
Apply α-β pruning to calculate minimax value for node A.



Adversarial search

Example: applying α-β pruning

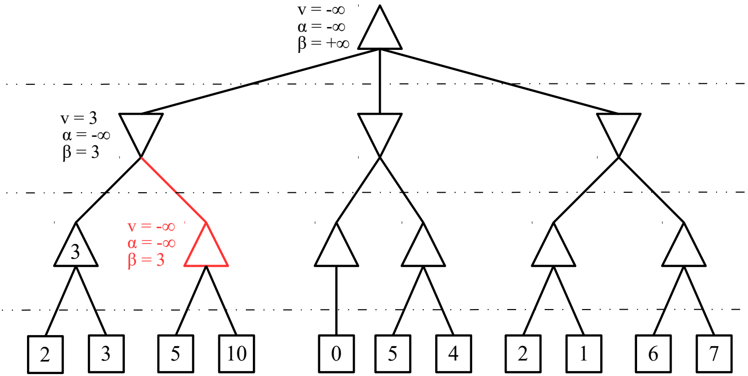
Apply α-β pruning to calculate minimax value for node A.



Adversarial search

Example: applying α-β pruning

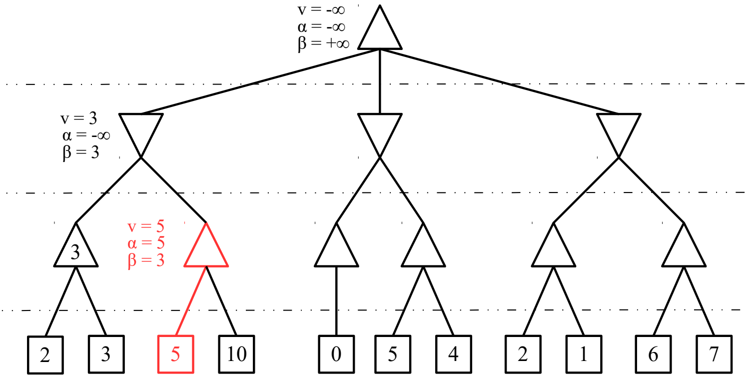
Apply α-β pruning to calculate minimax value for node A.



Adversarial search

Example: applying α-β pruning

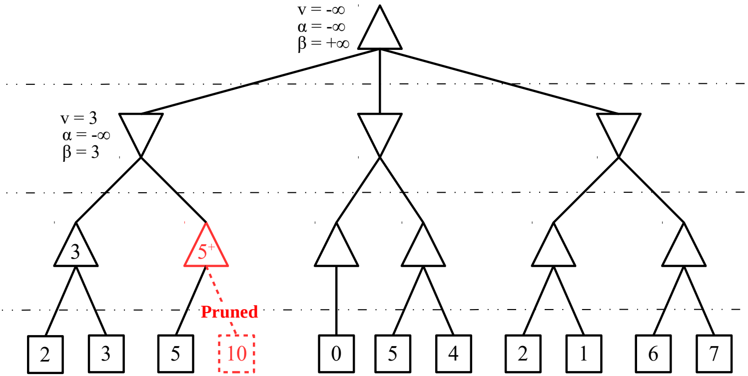
Apply α-β pruning to calculate minimax value for node A.



Adversarial search

Example: applying α-β pruning

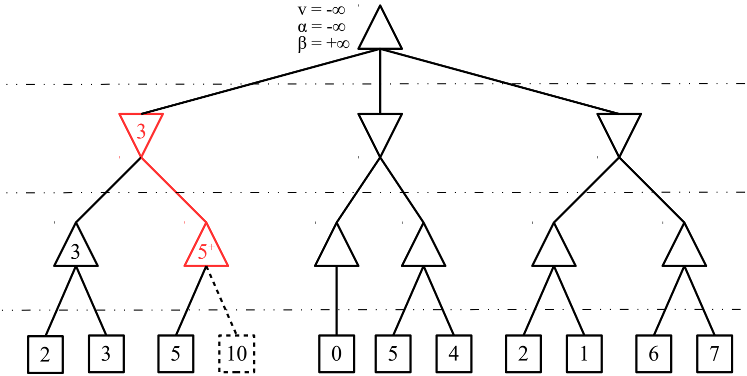
Apply α-β pruning to calculate minimax value for node A.



Adversarial search

Example: applying α-β pruning

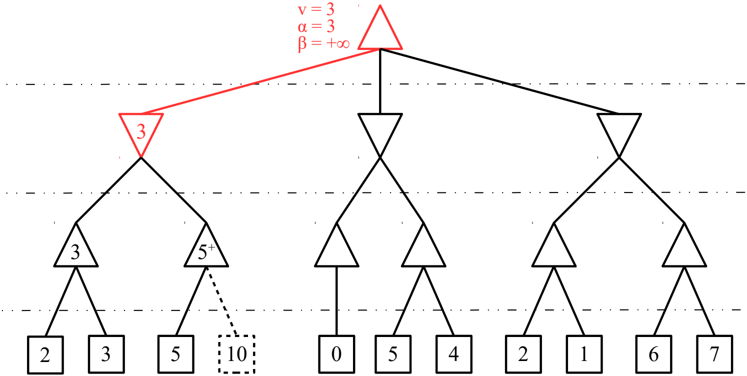
Apply α-β pruning to calculate minimax value for node A.



Adversarial search

Example: applying α-β pruning

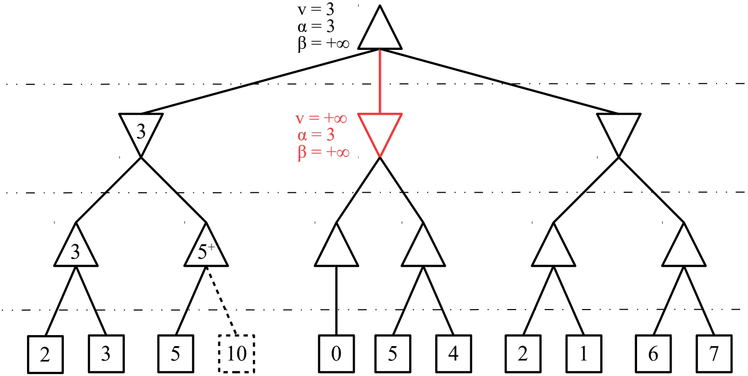
Apply α-β pruning to calculate minimax value for node A.



Adversarial search

Example: applying α-β pruning

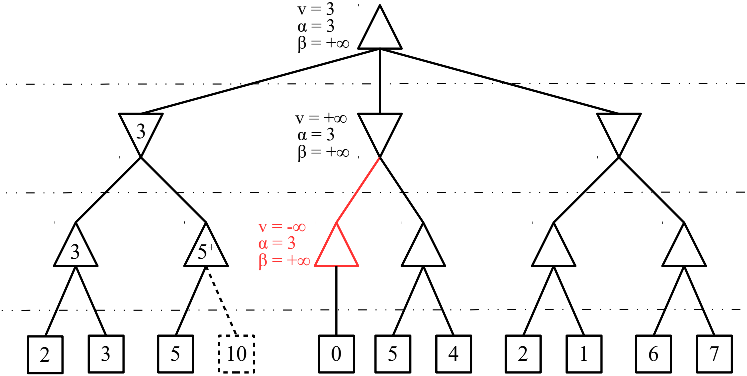
Apply α-β pruning to calculate minimax value for node A.



Adversarial search

Example: applying α-β pruning

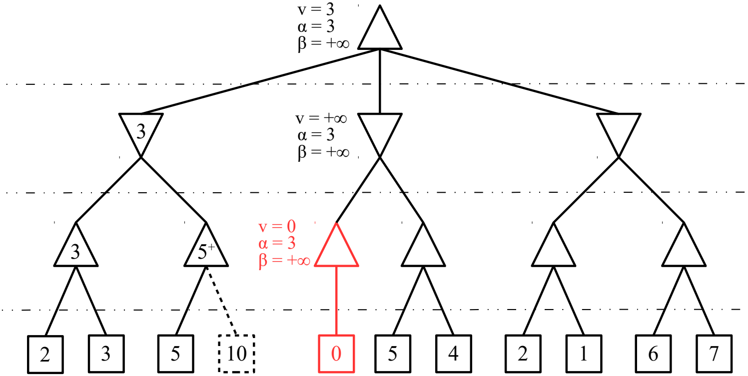
Apply α-β pruning to calculate minimax value for node A.



Adversarial search

Example: applying α-β pruning

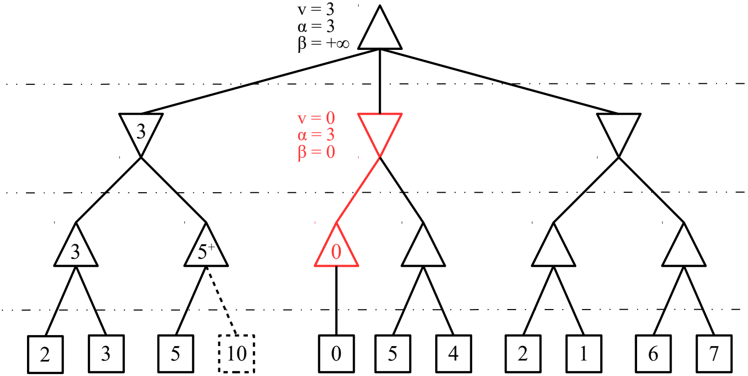
Apply α-β pruning to calculate minimax value for node A.



Adversarial search

Example: applying α-β pruning

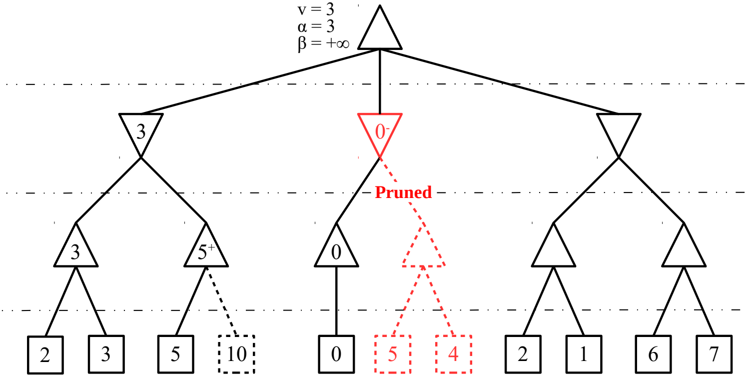
Apply α-β pruning to calculate minimax value for node A.



Adversarial search

Example: applying α-β pruning

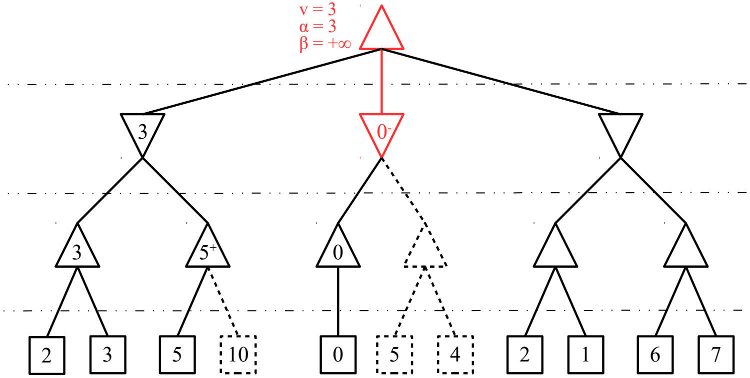
Apply α-β pruning to calculate minimax value for node A.



Adversarial search

Example: applying α-β pruning

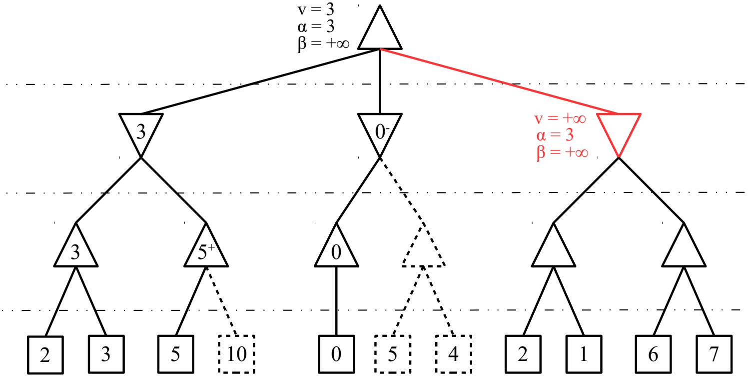
Apply α-β pruning to calculate minimax value for node A.



Adversarial search

Example: applying α-β pruning

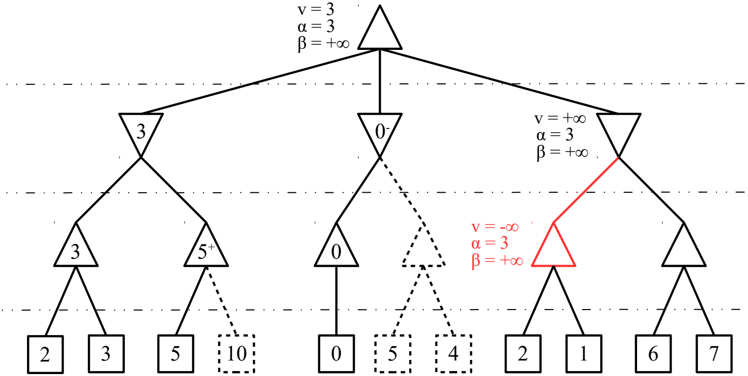
Apply α-β pruning to calculate minimax value for node A.



Adversarial search

Example: applying α-β pruning

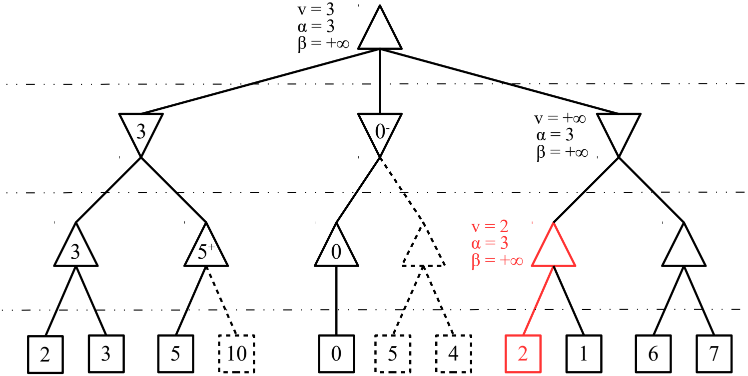
Apply α-β pruning to calculate minimax value for node A.



Adversarial search

Example: applying α-β pruning

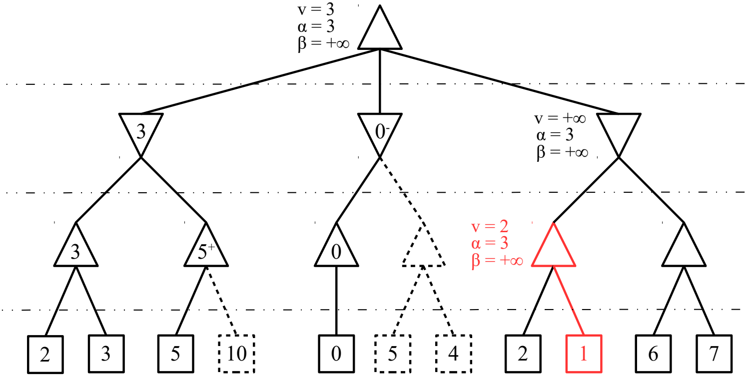
Apply α-β pruning to calculate minimax value for node A.



Adversarial search

Example: applying α-β pruning

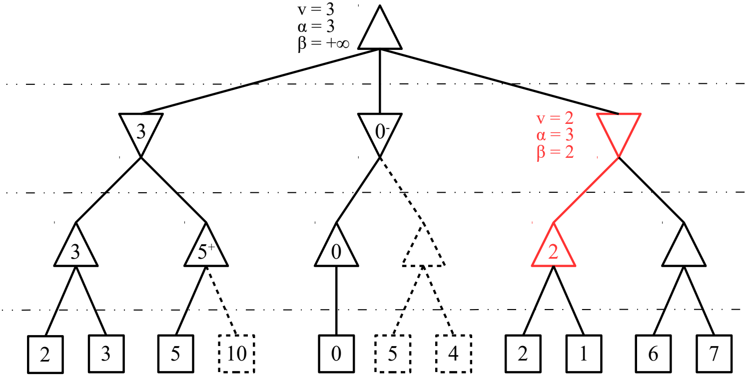
Apply α-β pruning to calculate minimax value for node A.



Adversarial search

Example: applying α-β pruning

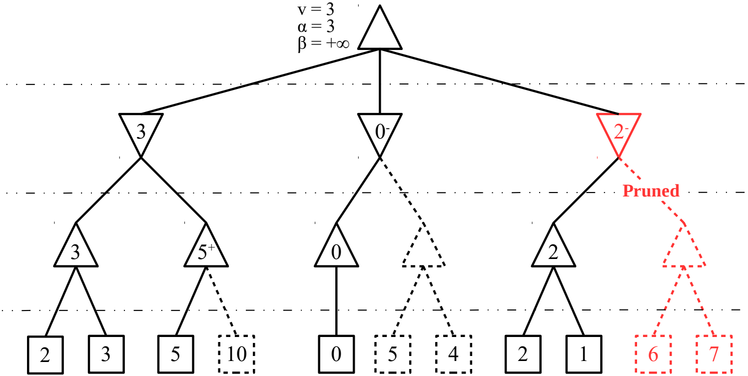
Apply α-β pruning to calculate minimax value for node A.



Adversarial search

Example: applying α-β pruning

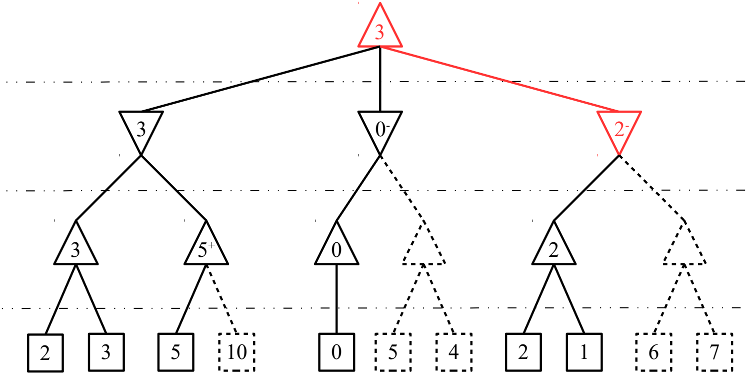
Apply α-β pruning to calculate minimax value for node A.



Adversarial search

Example: applying α-β pruning

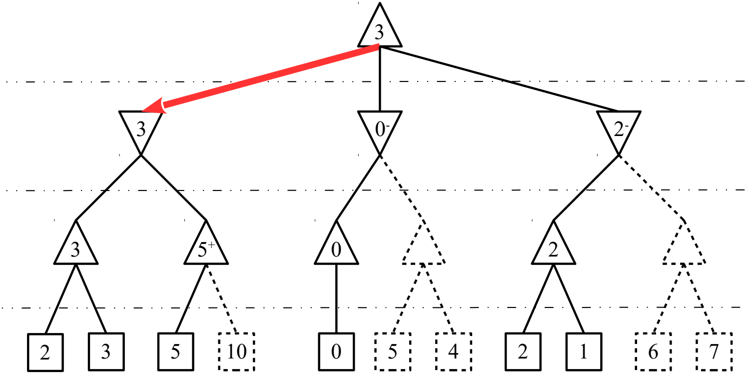
Apply α-β pruning to calculate minimax value for node A.



Adversarial search

Example: applying α-β pruning

Apply α-β pruning to calculate minimax value for node A.



Adversarial search

α-β Pruning

- In the previous example, α-β pruning explored 14 nodes instead of 21 nodes explored by Minimax.

Adversarial search

α-β Pruning

- In the previous example, α-β pruning explored 14 nodes instead of 21 nodes explored by Minimax.
- α-β pruning also reached the same decision reached by Minimax.

Adversarial search

The α-β search Algorithm

function ALPHA-BETA-SEARCH(*s*) **returns** an action
  v ← MAX-VALUE(*s*, -∞, +∞)
  return the **action** in ACTIONS(*s*) with value *v*

Adversarial search

The α-β search Algorithm

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

---

function MAX-VALUE(s, α, β) **returns** a utility value
  if TERMINAL-TEST(s) **then return** UTILITY(s)
  $v \leftarrow -\infty$
  for each a **in** ACTIONS(s) **do**
    $v \leftarrow$ MAX(v,MIN-VALUE(RESULT(s,a), α, β))
    $\alpha \leftarrow$ MAX(α,v)
    if $\alpha \geq \beta$ **then return** α /* β-cutoff */
  return v

Adversarial search

The α-β search Algorithm

function ALPHA-BETA-SEARCH(*s*) **returns** an action
  v ← MAX-VALUE(*s*, -∞, +∞)
  return the **action** in ACTIONS(*s*) with value *v*

---

function MAX-VALUE(*s*, α, β) **returns** a utility value
  if TERMINAL-TEST(*s*) **then return** UTILITY(*s*)
  v ← $-\infty$
  for each *a* **in** ACTIONS(*s*) **do**
    v ← MAX(*v*,MIN-VALUE(RESULT(*s*,*a*), α, β))
    α ← MAX(α,*v*)
    if $\alpha \geq \beta$ **then return** α /* β-cutoff */
  return *v*

---

function MIN-VALUE(*s*, α, β) **returns** a utility value
  if TERMINAL-TEST(*s*) **then return** UTILITY(*s*)
  v ← $+\infty$
  for each *a* **in** ACTIONS(*s*) **do**
    v ← MIN(*v*,MAX-VALUE(RESULT(*s*,*a*), α, β))
    β ← MIN(β,*v*)
    if $\alpha \geq \beta$ **then return** β /* α-cutoff */
  return *v*

Outline

Requirements

What do I need from you

- When given a certain problem you should be able to:
  - Formulate the game problem.

Requirements

What do I need from you

- When given a certain problem you should be able to:
    - Formulate the game problem.
    - Build the game tree up to a given depth.

Requirements

What do I need from you

- When given a certain problem you should be able to:
  - Formulate the game problem.
  - Build the game tree up to a given depth.
  - Apply the minimax search algorithm.

Requirements

What do I need from you

- When given a certain problem you should be able to:
  - Formulate the game problem.
  - Build the game tree up to a given depth.
  - Apply the minimax search algorithm.
  - Apply the $\alpha - \beta$ pruning algorithm.

Requirements

What do I need from you

- When given a certain problem you should be able to:
  - Formulate the game problem.
  - Build the game tree up to a given depth.
  - Apply the minimax search algorithm.
  - Apply the $\alpha - \beta$ pruning algorithm.
- Answer descriptive questions.

Reading Material

Which parts of the textbook are covered

- Russell-Norvig, Chapters 5:
  - Pages 161 - 170.