**Tìm kiếm Đối kháng: Giải quyết Tic-Tac-Toe bằng Tìm kiếm Minimax và Cắt tỉa Alpha-Beta**

**Giới thiệu**

Trò chơi nhiều người chơi có thể được triển khai như sau:

1. Hành động không xác định: Đối thủ được xem như một phần của môi trường với các hành động không xác định. Tính không xác định là kết quả của các nước đi không rõ của đối thủ.
2. **Quyết định Tối ưu:** Tìm kiếm Minimax (tìm kiếm toàn bộ cây trò chơi) và cắt tỉa alpha-beta.
3. Tìm kiếm Cây Alpha-Beta Heuristic: Cắt bớt cây tìm kiếm và sử dụng heuristic để ước tính giá trị trạng thái.
4. Tìm kiếm Cây Monte Carlo: Mô phỏng các lượt chơi để ước tính giá trị trạng thái.

Ở đây, chúng ta sẽ triển khai tìm kiếm cho Tic-Tac-Toe. Trò chơi này là một **trò chơi có tổng bằng không (zero-sum game)**: Thắng bởi x kết quả là +1, thắng bởi o là -1 và hòa có giá trị là 0. Max chơi x và cố gắng tối đa hóa kết quả trong khi Min chơi o và cố gắng tối thiểu hóa kết quả.

Chúng ta sẽ triển khai:

* **Tìm kiếm minimax,**
* **Tìm kiếm minimax với cắt tỉa alpha-beta,** và
* **Sắp xếp nước đi (move ordering).**

Thời gian tìm kiếm (số lượng nút được khám phá) có thể được cải thiện thêm bằng cách:

* **Tìm kiếm cây alpha-beta heuristic** (cắt bớt tìm kiếm và sử dụng hàm đánh giá heuristic để xấp xỉ giá trị tiện ích),
* **Cắt tỉa về phía trước (forward pruning)** (cắt tỉa các nước đi có vẻ kém), và
* **Tra cứu bảng (table lookups)** (cho các nước đi mở đầu và kết thúc ván cờ).

Các thuật toán tìm kiếm trên cây trò chơi và chúng ta có thể trả về một kế hoạch có điều kiện (hoặc kế hoạch một phần nếu sử dụng cắt bớt), nhưng việc triển khai ở đây chỉ xác định và trả về nước đi tối ưu tiếp theo.

Mã cho các hàm cơ bản được sử dụng cho tìm kiếm được triển khai trong tictactoe.py.

**Mã nguồn:**

Python

from tictactoe import empty\_board, actions, result, terminal, utility, other, show\_board

from tictactoe import random\_player, play

**Thuật toán DFS đệ quy cho Tìm kiếm Minimax**

Xem AIMA trang 150.

Lưu ý rằng thuật toán tìm kiếm minimax rất giống với tìm kiếm AND-OR, trong đó max thực hiện OR và min được biểu diễn bằng các nút AND.

**Mã nguồn:**

Python

import math

# global variables

DEBUG = 1 # 1 ... count nodes, 2 ... debug each node

COUNT = 0

def minimax\_search(board, player = 'x'):

"""start the search."""

global DEBUG, COUNT

COUNT = 0

value, move = max\_value(board, player)

if DEBUG >= 1: print(f"Number of nodes searched: {COUNT}")

return { "move": move, "value": value}

def max\_value(state, player):

"""player's best move."""

global DEBUG, COUNT

COUNT += 1

# return utility of state if it is a terminal state

v = utility(state, player)

if DEBUG >= 2: print("max in: " + str(state) + str([v]) )

if v is not None: return v, None

v, move = -math.inf, None

# check all possible actions in the state, return move with the largest value

for a in actions(state):

v2, a2 = min\_value(result(state, player, a), player)

if v2 > v:

v, move = v2, a

if DEBUG >= 2: print("max out: " + str(state) + str([v, move]) )

return v, move

def min\_value(state, player):

"""opponent's best response."""

global DEBUG, COUNT

COUNT += 1

# return utility of state if it is a terminal state

v = utility(state, player)

if DEBUG >= 2: print("min in: " + str(state) + str([v]) )

if v is not None: return v, None

v, move = +math.inf, None

# check all possible actions in the state, return move with the smallest value

for a in actions(state):

v2, a2 = max\_value(result(state, other(player), a), player)

if v2 < v:

v, move = v2, a

if DEBUG >= 2: print("min out: " + str(state) + str([v, move]) )

return v, move

**Một số Thử nghiệm**

**x sắp thắng (chọn 8)**

**Mã nguồn:**

Python

board = empty\_board()

board[0] = 'x'

board[1] = 'o'

board[3] = 'o'

board[4] = 'x'

print("Board:")

show\_board(board)

print()

%time display(minimax\_search(board))

**Kết quả:**

Board:

[Hình ảnh: Bàn cờ Tic-Tac-Toe với x ở vị trí 0, 4 và o ở vị trí 1, 3]

Number of nodes searched: 190

{'move': 2, 'value': 1}

CPU times: user 14.1 ms, sys: 1.99 ms, total: 16.1 ms

Wall time: 5.34 ms

**Lưu ý:** DFS không chọn đường đi ngắn nhất để thắng! Việc chiết khấu (discounting) giá trị tiện ích có thể hữu ích.

**x có thể hòa nếu chọn 7**

**Mã nguồn:**

Python

board = empty\_board()

board[0] = 'x'

board[1] = 'o'

board[2] = 'x'

board[4] = 'o'

print("Board:")

show\_board(board)

print()

%time display(minimax\_search(board))

**Kết quả:**

Board:

[Hình ảnh: Bàn cờ Tic-Tac-Toe với x ở vị trí 0, 2 và o ở vị trí 1, 4]

Number of nodes searched: 206

{'move': 7, 'value': 0}

CPU times: user 13.3 ms, sys: 5.27 ms, total: 18.6 ms

Wall time: 6.14 ms

**o sắp thắng dù x làm gì**

**Mã nguồn:**

Python

board = empty\_board()

board[0] = 'o'

board[1] = 'o'

board[3] = 'o'

board[4] = 'x'

board[8] = 'x'

print("Board:")

show\_board(board)

print()

%time display(minimax\_search(board))

**Kết quả:**

Board:

[Hình ảnh: Bàn cờ Tic-Tac-Toe với o ở vị trí 0, 1, 3 và x ở vị trí 4, 8]

Number of nodes searched: 33

{'move': 2, 'value': -1}

CPU times: user 1.88 ms, sys: 6.02 ms, total: 7.9 ms

Wall time: 2.57 ms

**Bàn cờ trống**

**Mã nguồn:**

Python

board = empty\_board()

print("Board:")

show\_board(board)

print()

%time display(minimax\_search(board))

**Kết quả:**

Board:

[Hình ảnh: Bàn cờ Tic-Tac-Toe trống]

Number of nodes searched: 549946

{'move': 0, 'value': 0}

CPU times: user 13.1 s, sys: 165 ms, total: 13.3 s

Wall time: 13 s

**Lưu ý:** Bắt đầu với một bàn cờ trống sẽ tìm kiếm toàn bộ cây trò chơi và mất một lúc. Số lượng nút ở trên là kích thước thực tế của cây trò chơi hoàn chỉnh. Một bảng với các 'nước đi mở đầu' (khai cuộc) đã biết (ví dụ: đặt x ở góc, o chọn trung tâm, v.v.) sẽ hữu ích để tăng tốc.

**Thuật toán DFS đệ quy cho Tìm kiếm Minimax với Cắt tỉa Alpha-Beta**

Xem AIMA trang 154.

Thêm cắt tỉa alpha-beta vào tìm kiếm minimax. Alpha và beta được sử dụng để duy trì các giới hạn về giá trị minimax của một nút dưới dạng [alpha, beta]. Alpha có nghĩa là giá trị ít nhất là cao bằng đó và beta có nghĩa là giá trị thực tế nhiều nhất là cao bằng đó. Các cây con bên dưới một nút mà tệ hơn giới hạn hiện đã biết không cần phải khám phá thêm và có thể được cắt tỉa. Max sử dụng alpha và Min sử dụng beta để cắt tỉa.

Việc triển khai dành cho người chơi 'x' trả về nước đi tối ưu tiếp theo.

**Mã nguồn:**

Python

# global variables

DEBUG = 1 # 1 ... count nodes, 2 ... debug each node

COUNT = 0

def alpha\_beta\_search(board, player = 'x'):

"""start the search."""

global DEBUG, COUNT

COUNT = 0

value, move = max\_value\_ab(board, player, -math.inf, +math.inf)

if DEBUG >= 1: print(f"Number of nodes searched: {COUNT}")

return { "move": move, "value": value }

def max\_value\_ab(state, player, alpha, beta):

"""player's best move."""

global DEBUG, COUNT

COUNT += 1

# return utility of state is a terminal state

v = utility(state, player)

if DEBUG >= 2: print(f"max: {state} [alpha,beta]=[{alpha},{beta}] v={v}")

if v is not None:

if DEBUG >= 2: print(f" found terminal state. backtracking.")

return v, None

v, move = -math.inf, None

# check all possible actions in the state, update alpha and return move with the largest value

for a in actions(state):

v2, a2 = min\_value\_ab(result(state, player, a), player, alpha, beta)

if DEBUG >= 2: print(f"max: {state} (backtracked) [alpha,beta]=[{alpha},{beta}] v={v2}")

if v2 > v:

v, move = v2, a

alpha = max(alpha, v)

if v >= beta:

if DEBUG >= 2: print(f" v>=beta ({v}>={beta}): pruning remaining subtree (actions). backtracking.")

return v, move

return v, move

def min\_value\_ab(state, player, alpha, beta):

"""opponent's best response."""

global DEBUG, COUNT

COUNT += 1

# return utility of state is a terminal state

v = utility(state, player)

if DEBUG >= 2: print(f"min: {state} [alpha,beta]=[{alpha},{beta}] v={v}")

if v is not None:

if DEBUG >= 2: print(f" found terminal state. backtacking.")

return v, None

v, move = +math.inf, None

# check all possible actions in the state, update beta and return move with the smallest value

for a in actions(state):

v2, a2 = max\_value\_ab(result(state, other(player), a), player, alpha, beta)

if DEBUG >= 2: print(f"min: {state} (backtracked) [alpha,beta]=[{alpha},{beta}] v={v2}")

if v2 < v:

v, move = v2, a

beta = min(beta, v)

if v <= alpha:

if DEBUG >= 2: print(f" v<=alpha ({v}<={alpha}): pruning remaining subtree (actions). backtracking.")

return v, move

return v, move

**Một số Thử nghiệm**

**x sắp thắng (chơi 8)**

**Mã nguồn:**

Python

DEBUG = 2 # show more debugging info

board = empty\_board()

board[0] = 'x'

board[1] = 'o'

board[3] = 'o'

board[4] = 'x'

print("Board:")

show\_board(board)

print()

%time display(alpha\_beta\_search(board))

**Kết quả:**

Board:

[Hình ảnh: Bàn cờ Tic-Tac-Toe với x ở vị trí 0, 4 và o ở vị trí 1, 3]

max: ['x', 'o', ' ', 'o', 'x', ' ', ' ', ' ', ' '] [alpha,beta]=[-inf,inf] v=None

min: ['x', 'o', 'x', 'o', 'x', ' ', ' ', ' ', ' '] [alpha,beta]=[-inf,inf] v=None

max: ['x', 'o', 'x', 'o', 'x', 'o', ' ', ' ', ' '] [alpha,beta]=[-inf,inf] v=None

min: ['x', 'o', 'x', 'o', 'x', 'o', 'x', ' ', ' '] [alpha,beta]=[-inf,inf] v=1

found terminal state. backtacking.

max: ['x', 'o', 'x', 'o', 'x', 'o', ' ', ' ', ' '] (backtracked) [alpha,beta]=[-inf,inf] v=1

min: ['x', 'o', 'x', 'o', 'x', 'o', ' ', 'x', ' '] [alpha,beta]=[1,inf] v=None

max: ['x', 'o', 'x', 'o', 'x', 'o', 'o', 'x', ' '] [alpha,beta]=[1,inf] v=None

min: ['x', 'o', 'x', 'o', 'x', 'o', 'o', 'x', 'x'] [alpha,beta]=[1,inf] v=1

found terminal state. backtacking.

max: ['x', 'o', 'x', 'o', 'x', 'o', 'o', 'x', ' '] (backtracked) [alpha,beta]=[1,inf] v=1

min: ['x', 'o', 'x', 'o', 'x', 'o', ' ', 'x', ' '] (backtracked) [alpha,beta]=[1,inf] v=1

v<=alpha (1<=1): pruning remaining subtree (actions). backtracking.

max: ['x', 'o', 'x', 'o', 'x', 'o', ' ', ' ', ' '] (backtracked) [alpha,beta]=[1,inf] v=1

min: ['x', 'o', 'x', 'o', 'x', 'o', ' ', ' ', 'x'] [alpha,beta]=[1,inf] v=1

found terminal state. backtacking.

max: ['x', 'o', 'x', 'o', 'x', 'o', ' ', ' ', ' '] (backtracked) [alpha,beta]=[1,inf] v=1

min: ['x', 'o', 'x', 'o', 'x', ' ', ' ', ' ', ' '] (backtracked) [alpha,beta]=[-inf,inf] v=1

max: ['x', 'o', 'x', 'o', 'x', ' ', 'o', ' ', ' '] [alpha,beta]=[-inf,1] v=None

min: ['x', 'o', 'x', 'o', 'x', 'x', 'o', ' ', ' '] [alpha,beta]=[-inf,1] v=None

max: ['x', 'o', 'x', 'o', 'x', 'x', 'o', 'o', ' '] [alpha,beta]=[-inf,1] v=None

min: ['x', 'o', 'x', 'o', 'x', 'x', 'o', 'o', 'x'] [alpha,beta]=[-inf,1] v=1

found terminal state. backtacking.

max: ['x', 'o', 'x', 'o', 'x', 'x', 'o', 'o', ' '] (backtracked) [alpha,beta]=[-inf,1] v=1

v>=beta (1>=1): pruning remaining subtree (actions). backtracking.

min: ['x', 'o', 'x', 'o', 'x', 'x', 'o', ' ', ' '] (backtracked) [alpha,beta]=[-inf,1] v=1

max: ['x', 'o', 'x', 'o', 'x', 'x', 'o', ' ', 'o'] [alpha,beta]=[-inf,1] v=None

min: ['x', 'o', 'x', 'o', 'x', 'x', 'o', 'x', 'o'] [alpha,beta]=[-inf,1] v=0

found terminal state. backtacking.

max: ['x', 'o', 'x', 'o', 'x', 'x', 'o', ' ', 'o'] (backtracked) [alpha,beta]=[-inf,1] v=0

min: ['x', 'o', 'x', 'o', 'x', 'x', 'o', ' ', ' '] (backtracked) [alpha,beta]=[-inf,1] v=0

max: ['x', 'o', 'x', 'o', 'x', ' ', 'o', ' ', ' '] (backtracked) [alpha,beta]=[-inf,1] v=0

min: ['x', 'o', 'x', 'o', 'x', ' ', 'o', 'x', ' '] [alpha,beta]=[0,1] v=None

max: ['x', 'o', 'x', 'o', 'x', 'o', 'o', 'x', ' '] [alpha,beta]=[0,1] v=None

min: ['x', 'o', 'x', 'o', 'x', 'o', 'o', 'x', 'x'] [alpha,beta]=[0,1] v=1

found terminal state. backtacking.

max: ['x', 'o', 'x', 'o', 'x', 'o', 'o', 'x', ' '] (backtracked) [alpha,beta]=[0,1] v=1

v>=beta (1>=1): pruning remaining subtree (actions). backtracking.

min: ['x', 'o', 'x', 'o', 'x', ' ', 'o', 'x', ' '] (backtracked) [alpha,beta]=[0,1] v=1

max: ['x', 'o', 'x', 'o', 'x', ' ', 'o', 'x', 'o'] [alpha,beta]=[0,1] v=None

min: ['x', 'o', 'x', 'o', 'x', 'x', 'o', 'x', 'o'] [alpha,beta]=[0,1] v=0

found terminal state. backtacking.

max: ['x', 'o', 'x', 'o', 'x', ' ', 'o', 'x', 'o'] (backtracked) [alpha,beta]=[0,1] v=0

min: ['x', 'o', 'x', 'o', 'x', ' ', 'o', 'x', ' '] (backtracked) [alpha,beta]=[0,1] v=0

v<=alpha (0<=0): pruning remaining subtree (actions). backtracking.

max: ['x', 'o', 'x', 'o', 'x', ' ', 'o', ' ', ' '] (backtracked) [alpha,beta]=[0,1] v=0

min: ['x', 'o', 'x', 'o', 'x', ' ', 'o', ' ', 'x'] [alpha,beta]=[0,1] v=1

found terminal state. backtacking.

max: ['x', 'o', 'x', 'o', 'x', ' ', 'o', ' ', ' '] (backtracked) [alpha,beta]=[0,1] v=1

v>=beta (1>=1): pruning remaining subtree (actions). backtracking.

min: ['x', 'o', 'x', 'o', 'x', ' ', ' ', ' ', ' '] (backtracked) [alpha,beta]=[-inf,1] v=1

max: ['x', 'o', 'x', 'o', 'x', ' ', ' ', 'o', ' '] [alpha,beta]=[-inf,1] v=None

min: ['x', 'o', 'x', 'o', 'x', 'x', ' ', 'o', ' '] [alpha,beta]=[-inf,1] v=None

max: ['x', 'o', 'x', 'o', 'x', 'x', 'o', 'o', ' '] [alpha,beta]=[-inf,1] v=None

min: ['x', 'o', 'x', 'o', 'x', 'x', 'o', 'o', 'x'] [alpha,beta]=[-inf,1] v=1

found terminal state. backtacking.

max: ['x', 'o', 'x', 'o', 'x', 'x', 'o', 'o', ' '] (backtracked) [alpha,beta]=[-inf,1] v=1

v>=beta (1>=1): pruning remaining subtree (actions). backtracking.

min: ['x', 'o', 'x', 'o', 'x', 'x', ' ', 'o', ' '] (backtracked) [alpha,beta]=[-inf,1] v=1

max: ['x', 'o', 'x', 'o', 'x', 'x', ' ', 'o', 'o'] [alpha,beta]=[-inf,1] v=None

min: ['x', 'o', 'x', 'o', 'x', 'x', 'x', 'o', 'o'] [alpha,beta]=[-inf,1] v=1

found terminal state. backtacking.

max: ['x', 'o', 'x', 'o', 'x', 'x', ' ', 'o', 'o'] (backtracked) [alpha,beta]=[-inf,1] v=1

v>=beta (1>=1): pruning remaining subtree (actions). backtracking.

min: ['x', 'o', 'x', 'o', 'x', 'x', ' ', 'o', ' '] (backtracked) [alpha,beta]=[-inf,1] v=1

max: ['x', 'o', 'x', 'o', 'x', ' ', ' ', 'o', ' '] (backtracked) [alpha,beta]=[-inf,1] v=1

v>=beta (1>=1): pruning remaining subtree (actions). backtracking.

min: ['x', 'o', 'x', 'o', 'x', ' ', ' ', ' ', ' '] (backtracked) [alpha,beta]=[-inf,1] v=1

max: ['x', 'o', 'x', 'o', 'x', ' ', ' ', ' ', 'o'] [alpha,beta]=[-inf,1] v=None

min: ['x', 'o', 'x', 'o', 'x', 'x', ' ', ' ', 'o'] [alpha,beta]=[-inf,1] v=None

max: ['x', 'o', 'x', 'o', 'x', 'x', 'o', ' ', 'o'] [alpha,beta]=[-inf,1] v=None

min: ['x', 'o', 'x', 'o', 'x', 'x', 'o', 'x', 'o'] [alpha,beta]=[-inf,1] v=0

found terminal state. backtacking.

max: ['x', 'o', 'x', 'o', 'x', 'x', 'o', ' ', 'o'] (backtracked) [alpha,beta]=[-inf,1] v=0

min: ['x', 'o', 'x', 'o', 'x', 'x', ' ', ' ', 'o'] (backtracked) [alpha,beta]=[-inf,1] v=0

max: ['x', 'o', 'x', 'o', 'x', 'x', ' ', 'o', 'o'] [alpha,beta]=[-inf,0] v=None

min: ['x', 'o', 'x', 'o', 'x', 'x', 'x', 'o', 'o'] [alpha,beta]=[-inf,0] v=1

found terminal state. backtacking.

max: ['x', 'o', 'x', 'o', 'x', 'x', ' ', 'o', 'o'] (backtracked) [alpha,beta]=[-inf,0] v=1

v>=beta (1>=0): pruning remaining subtree (actions). backtracking.

min: ['x', 'o', 'x', 'o', 'x', 'x', ' ', ' ', 'o'] (backtracked) [alpha,beta]=[-inf,0] v=1

max: ['x', 'o', 'x', 'o', 'x', ' ', ' ', ' ', 'o'] (backtracked) [alpha,beta]=[-inf,1] v=0

min: ['x', 'o', 'x', 'o', 'x', ' ', 'x', ' ', 'o'] [alpha,beta]=[0,1] v=1

found terminal state. backtacking.

max: ['x', 'o', 'x', 'o', 'x', ' ', ' ', ' ', 'o'] (backtracked) [alpha,beta]=[0,1] v=1

v>=beta (1>=1): pruning remaining subtree (actions). backtracking.

min: ['x', 'o', 'x', 'o', 'x', ' ', ' ', ' ', ' '] (backtracked) [alpha,beta]=[-inf,1] v=1

max: ['x', 'o', ' ', 'o', 'x', ' ', ' ', ' ', ' '] (backtracked) [alpha,beta]=[-inf,inf] v=1

min: ['x', 'o', ' ', 'o', 'x', 'x', ' ', ' ', ' '] [alpha,beta]=[1,inf] v=None

max: ['x', 'o', 'o', 'o', 'x', 'x', ' ', ' ', ' '] [alpha,beta]=[1,inf] v=None

min: ['x', 'o', 'o', 'o', 'x', 'x', 'x', ' ', ' '] [alpha,beta]=[1,inf] v=None

max: ['x', 'o', 'o', 'o', 'x', 'x', 'x', 'o', ' '] [alpha,beta]=[1,inf] v=None

min: ['x', 'o', 'o', 'o', 'x', 'x', 'x', 'o', 'x'] [alpha,beta]=[1,inf] v=1

found terminal state. backtacking.

max: ['x', 'o', 'o', 'o', 'x', 'x', 'x', 'o', ' '] (backtracked) [alpha,beta]=[1,inf] v=1

min: ['x', 'o', 'o', 'o', 'x', 'x', 'x', ' ', ' '] (backtracked) [alpha,beta]=[1,inf] v=1

v<=alpha (1<=1): pruning remaining subtree (actions). backtracking.

max: ['x', 'o', 'o', 'o', 'x', 'x', ' ', ' ', ' '] (backtracked) [alpha,beta]=[1,inf] v=1

min: ['x', 'o', 'o', 'o', 'x', 'x', ' ', 'x', ' '] [alpha,beta]=[1,inf] v=None

max: ['x', 'o', 'o', 'o', 'x', 'x', 'o', 'x', ' '] [alpha,beta]=[1,inf] v=None

min: ['x', 'o', 'o', 'o', 'x', 'x', 'o', 'x', 'x'] [alpha,beta]=[1,inf] v=1

found terminal state. backtacking.

max: ['x', 'o', 'o', 'o', 'x', 'x', 'o', 'x', ' '] (backtracked) [alpha,beta]=[1,inf] v=1

min: ['x', 'o', 'o', 'o', 'x', 'x', ' ', 'x', ' '] (backtracked) [alpha,beta]=[1,inf] v=1

v<=alpha (1<=1): pruning remaining subtree (actions). backtracking.

max: ['x', 'o', 'o', 'o', 'x', 'x', ' ', ' ', ' '] (backtracked) [alpha,beta]=[1,inf] v=1

min: ['x', 'o', 'o', 'o', 'x', 'x', ' ', ' ', 'x'] [alpha,beta]=[1,inf] v=1

found terminal state. backtacking.

max: ['x', 'o', 'o', 'o', 'x', 'x', ' ', ' ', ' '] (backtracked) [alpha,beta]=[1,inf] v=1

min: ['x', 'o', ' ', 'o', 'x', 'x', ' ', ' ', ' '] (backtracked) [alpha,beta]=[1,inf] v=1

v<=alpha (1<=1): pruning remaining subtree (actions). backtracking.

max: ['x', 'o', ' ', 'o', 'x', ' ', ' ', ' ', ' '] (backtracked) [alpha,beta]=[1,inf] v=1

min: ['x', 'o', ' ', 'o', 'x', ' ', 'x', ' ', ' '] [alpha,beta]=[1,inf] v=None

max: ['x', 'o', 'o', 'o', 'x', ' ', 'x', ' ', ' '] [alpha,beta]=[1,inf] v=None

min: ['x', 'o', 'o', 'o', 'x', 'x', 'x', ' ', ' '] [alpha,beta]=[1,inf] v=None

max: ['x', 'o', 'o', 'o', 'x', 'x', 'x', 'o', ' '] [alpha,beta]=[1,inf] v=None

min: ['x', 'o', 'o', 'o', 'x', 'x', 'x', 'o', 'x'] [alpha,beta]=[1,inf] v=1

found terminal state. backtacking.

max: ['x', 'o', 'o', 'o', 'x', 'x', 'x', 'o', ' '] (backtracked) [alpha,beta]=[1,inf] v=1

min: ['x', 'o', 'o', 'o', 'x', 'x', 'x', ' ', ' '] (backtracked) [alpha,beta]=[1,inf] v=1

v<=alpha (1<=1): pruning remaining subtree (actions). backtracking.

max: ['x', 'o', 'o', 'o', 'x', ' ', 'x', ' ', ' '] (backtracked) [alpha,beta]=[1,inf] v=1

min: ['x', 'o', 'o', 'o', 'x', ' ', 'x', 'x', ' '] [alpha,beta]=[1,inf] v=None

max: ['x', 'o', 'o', 'o', 'x', 'o', 'x', 'x', ' '] [alpha,beta]=[1,inf] v=None

min: ['x', 'o', 'o', 'o', 'x', 'o', 'x', 'x', 'x'] [alpha,beta]=[1,inf] v=1

found terminal state. backtacking.

max: ['x', 'o', 'o', 'o', 'x', 'o', 'x', 'x', ' '] (backtracked) [alpha,beta]=[1,inf] v=1

min: ['x', 'o', 'o', 'o', 'x', ' ', 'x', 'x', ' '] (backtracked) [alpha,beta]=[1,inf] v=1

v<=alpha (1<=1): pruning remaining subtree (actions). backtracking.

max: ['x', 'o', 'o', 'o', 'x', ' ', 'x', ' ', ' '] (backtracked) [alpha,beta]=[1,inf] v=1

min: ['x', 'o', 'o', 'o', 'x', ' ', 'x', ' ', 'x'] [alpha,beta]=[1,inf] v=1

found terminal state. backtacking.

max: ['x', 'o', 'o', 'o', 'x', ' ', 'x', ' ', ' '] (backtracked) [alpha,beta]=[1,inf] v=1

min: ['x', 'o', ' ', 'o', 'x', ' ', 'x', ' ', ' '] (backtracked) [alpha,beta]=[1,inf] v=1

v<=alpha (1<=1): pruning remaining subtree (actions). backtracking.

max: ['x', 'o', ' ', 'o', 'x', ' ', ' ', ' ', ' '] (backtracked) [alpha,beta]=[1,inf] v=1

min: ['x', 'o', ' ', 'o', 'x', ' ', ' ', 'x', ' '] [alpha,beta]=[1,inf] v=None

max: ['x', 'o', 'o', 'o', 'x', ' ', ' ', 'x', ' '] [alpha,beta]=[1,inf] v=None

min: ['x', 'o', 'o', 'o', 'x', 'x', ' ', 'x', ' '] [alpha,beta]=[1,inf] v=None

max: ['x', 'o', 'o', 'o', 'x', 'x', 'o', 'x', ' '] [alpha,beta]=[1,inf] v=None

min: ['x', 'o', 'o', 'o', 'x', 'x', 'o', 'x', 'x'] [alpha,beta]=[1,inf] v=1

found terminal state. backtacking.

max: ['x', 'o', 'o', 'o', 'x', 'x', 'o', 'x', ' '] (backtracked) [alpha,beta]=[1,inf] v=1

min: ['x', 'o', 'o', 'o', 'x', 'x', ' ', 'x', ' '] (backtracked) [alpha,beta]=[1,inf] v=1

v<=alpha (1<=1): pruning remaining subtree (actions). backtracking.

max: ['x', 'o', 'o', 'o', 'x', ' ', ' ', 'x', ' '] (backtracked) [alpha,beta]=[1,inf] v=1

min: ['x', 'o', 'o', 'o', 'x', ' ', 'x', 'x', ' '] [alpha,beta]=[1,inf] v=None

max: ['x', 'o', 'o', 'o', 'x', 'o', 'x', 'x', ' '] [alpha,beta]=[1,inf] v=None

min: ['x', 'o', 'o', 'o', 'x', 'o', 'x', 'x', 'x'] [alpha,beta]=[1,inf] v=1

found terminal state. backtacking.

max: ['x', 'o', 'o', 'o', 'x', 'o', 'x', 'x', ' '] (backtracked) [alpha,beta]=[1,inf] v=1

min: ['x', 'o', 'o', 'o', 'x', ' ', 'x', 'x', ' '] (backtracked) [alpha,beta]=[1,inf] v=1

v<=alpha (1<=1): pruning remaining subtree (actions). backtracking.

max: ['x', 'o', 'o', 'o', 'x', ' ', ' ', 'x', ' '] (backtracked) [alpha,beta]=[1,inf] v=1

min: ['x', 'o', 'o', 'o', 'x', ' ', ' ', 'x', 'x'] [alpha,beta]=[1,inf] v=1

found terminal state. backtacking.

max: ['x', 'o', 'o', 'o', 'x', ' ', ' ', 'x', ' '] (backtracked) [alpha,beta]=[1,inf] v=1

min: ['x', 'o', ' ', 'o', 'x', ' ', ' ', 'x', ' '] (backtracked) [alpha,beta]=[1,inf] v=1

v<=alpha (1<=1): pruning remaining subtree (actions). backtracking.

max: ['x', 'o', ' ', 'o', 'x', ' ', ' ', ' ', ' '] (backtracked) [alpha,beta]=[1,inf] v=1

min: ['x', 'o', ' ', 'o', 'x', ' ', ' ', ' ', 'x'] [alpha,beta]=[1,inf] v=1

found terminal state. backtacking.

max: ['x', 'o', ' ', 'o', 'x', ' ', ' ', ' ', ' '] (backtracked) [alpha,beta]=[1,inf] v=1

Number of nodes searched: 61

{'move': 2, 'value': 1}

CPU times: user 7.28 ms, sys: 8.41 ms, total: 15.7 ms

Wall time: 5.2 ms

Mã không chọn đường đi ngắn nhất để thắng! Chiết khấu có thể hữu ích.

**x có thể hòa nếu chọn 7**

**Mã nguồn:**

Python

DEBUG = 1

board = empty\_board()

board[0] = 'x'

board[1] = 'o'

board[2] = 'x'

board[4] = 'o'

print("Board:")

show\_board(board)

print()

%time display(alpha\_beta\_search(board))

**Kết quả:**

Board:

[Hình ảnh: Bàn cờ Tic-Tac-Toe với x ở vị trí 0, 2 và o ở vị trí 1, 4]

Number of nodes searched: 101

{'move': 7, 'value': 0}

CPU times: user 18.5 ms, sys: 0 ns, total: 18.5 ms

Wall time: 6.11 ms

**o sắp thắng dù x làm gì**

**Mã nguồn:**

Python

board = empty\_board()

board[0] = 'o'

board[1] = 'o'

board[3] = 'o'

board[4] = 'x'

board[8] = 'x'

print("Board:")

show\_board(board)

print()

%time display(alpha\_beta\_search(board))

**Kết quả:**

Board:

[Hình ảnh: Bàn cờ Tic-Tac-Toe với o ở vị trí 0, 1, 3 và x ở vị trí 4, 8]

Number of nodes searched: 15

{'move': 2, 'value': -1}

CPU times: user 8.66 ms, sys: 1.28 ms, total: 9.94 ms

Wall time: 3.27 ms

**Bàn cờ trống: Chỉ có thể đảm bảo hòa**

**Mã nguồn:**

Python

board = empty\_board()

print("Board:")

show\_board(board)

print()

%time display(alpha\_beta\_search(board))

**Kết quả:**

Board:

[Hình ảnh: Bàn cờ Tic-Tac-Toe trống]

Number of nodes searched: 18297

{'move': 0, 'value': 0}

CPU times: user 720 ms, sys: 188 ms, total: 908 ms

Wall time: 660 ms

**Lưu ý:** Tìm kiếm Alpha-Beta mở rộng ít nút hơn và **nhanh hơn nhiều** so với tìm kiếm minimax.

**Sắp xếp nước đi (Move ordering)**

Sắp xếp lại nước đi một cách thông minh có thể làm cho việc cắt tỉa alpha-beta hiệu quả hơn. Tôi nghĩ trung tâm [4] và các góc [0, 2, 6, 8] là tốt. Tôi triển khai sắp xếp lại nước đi trong hàm actions().

**Mã nguồn:**

Python

import numpy as np

def actions(board):

"""return possible actions as a vector ot indices"""

actions = np.where(np.array(board) == ' ')[0].tolist()

priority = [1,0,1,

0,2,0,

1,0,1]

priority = [priority[i] for i in actions]

actions =[a for \_,a in sorted(zip(priority,actions), reverse=True)]

return actions

board = empty\_board()

show\_board(board)

actions(board)

Kết quả:

[Hình ảnh: Bàn cờ Tic-Tac-Toe trống]

[4, 8, 6, 2, 0, 7, 5, 3, 1]

**Bàn cờ trống: Chỉ có thể đảm bảo hòa**

**Mã nguồn:**

Python

board = empty\_board()

print("Board:")

show\_board(board)

print()

%time display(alpha\_beta\_search(board))

**Kết quả:**

Board:

[Hình ảnh: Bàn cờ Tic-Tac-Toe trống]

Number of nodes searched: 7275

{'move': 4, 'value': 0}

CPU times: user 259 ms, sys: 196 ms, total: 455 ms

Wall time: 202 ms

**Lưu ý:** So sánh với số lượng nút được tìm kiếm mà không sắp xếp lại nước đi (ngay bên trên).

**Thử nghiệm**

**Minimax với Cắt tỉa Alpha-Beta vs. Ngẫu nhiên**

**Mã nguồn:**

Python

DEBUG = 0

def alpha\_beta\_player(board, player = 'x'):

return alpha\_beta\_search(board, player)["move"]

print("alpha-beta vs. random:")

%time display(play(alpha\_beta\_player, random\_player))

print()

print("random vs. alpha-beta")

%time display(play(random\_player, alpha\_beta\_player))

**Kết quả:**

alpha-beta vs. random:

{'x': 100, 'o': 0, 'd': 0}

CPU times: user 15.5 s, sys: 1.96 ms, total: 15.5 s

Wall time: 15.5 s

random vs. alpha-beta

{'x': 0, 'o': 84, 'd': 16}

CPU times: user 5.16 s, sys: 7.74 ms, total: 5.16 s

Wall time: 5.17 s