**Định nghĩa trò chơi: Tic-Tac-Toe**

**Vấn đề Tìm kiếm**

Một bài toán tìm kiếm tổng quát được định nghĩa theo cách sau:

* **Trạng thái ban đầu (Initial State):** Bàn cờ $3 \times 3$ trống. Đến lượt $x$ đi.
* **Hành động (Actions):** Đặt biểu tượng của bạn vào bất kỳ ô trống nào.
* Hàm chuyển tiếp (Transition function): Biểu tượng của bạn được đặt lên bàn cờ theo hành động. Đối thủ

đặt biểu tượng của họ. Từ góc nhìn của chúng ta, điều này làm cho môi trường trở nên không xác định (non-deterministic).

* **Trạng thái mục tiêu (Goal state):** Một chiến thắng (ba biểu tượng liên tiếp trên một hàng, cột hoặc đường chéo).
* **Chi phí đường đi (Path cost):** số lượng nước đi.

Vì Tic-tac-toe là một trò chơi, chúng ta sẽ sử dụng các thành phần chuyên biệt hơn (xem Chương 5):

* Actions(s): Các nước đi hợp lệ trong trạng thái s.
* Result(s, a): Mô hình chuyển tiếp.
* Terminal(s): Kiểm tra trạng thái kết thúc.
* Utility(s): Giá trị tiện ích cho người chơi Max tại các trạng thái kết thúc.

**Ghi chú:**

* Để định nghĩa trạng thái mục tiêu, chúng ta sẽ sử dụng một bài kiểm tra cho **trạng thái kết thúc** (trò chơi kết thúc) và một **hàm tiện ích** (thắng hay thua).
* Chúng ta sẽ sử dụng DFS (Tìm kiếm theo chiều sâu) vốn không phải là tối ưu. Vì vậy, có thể sẽ không tìm thấy giải pháp có chi phí đường đi thấp nhất, tức là, chiến thắng với số lượng nước đi ít nhất. Do đó, chúng ta bỏ qua chi phí đường đi.

**Quy mô bài toán**

**Ước tính không gian trạng thái**

Mỗi trạng thái là một bàn cờ có thể có. Mỗi ô trong 9 ô có thể có 3 giá trị (trống, x và o), nhưng một số bàn cờ là không thể (ví dụ: một người chơi có nhiều chuỗi 3). Số lượng trạng thái trong biểu đồ không gian trạng thái nhỏ hơn $3^9$.

**Mã nguồn:**

Python

3\*\*9

**Kết quả:**

19683

Một giới hạn chặt chẽ hơn cho kích thước không gian tìm kiếm sẽ là xem xét tất cả các cách chúng ta có thể chọn $i = 1,2,3,...,9$ vị trí từ bàn cờ, được cho bởi ${9 \choose i}$. Bây giờ chúng ta cần tìm tất cả các cách làm thế nào chúng ta có thể chọn một nửa số biểu tượng cho người chơi o, được cho bởi ${i \choose \lfloor \frac{i}{2} \rfloor}$.

Kết hợp lại, chúng ta có $\sum\_{i=1}^9{{9 \choose i}{i \choose \lfloor \frac{i}{2} \rfloor}}$.

**Mã nguồn:**

Python

import math

print("level\tboards")

sum = 0

for i in range(1,9):

sum += math.comb(9, i) \* math.comb(i, math.floor(i/2))

print(i, "\t", math.comb(9, i) , " x ", math.comb(i, math.floor(i/2)) , " = ", math.comb(9, i) \* math.comb(i, math.floor(i/2)))

sum

**Kết quả:**

level boards

1 9 x 1 = 9

2 36 x 2 = 72

3 84 x 3 = 252

4 126 x 6 = 756

5 126 x 10 = 1260

6 84 x 20 = 1680

7 36 x 35 = 1260

8 9 x 70 = 630

5919

Đây là một giới hạn trên vì nó chứa một số bàn cờ không khả thi, nơi cả hai người chơi đều có 3 biểu tượng liên tiếp.

**Ước tính độ phức tạp về không gian và thời gian sử dụng cây tìm kiếm**

Để giải quyết vấn đề, chúng ta sử dụng cây tìm kiếm, có thể lớn hơn đáng kể so với không gian trạng thái.

Một cây tìm kiếm là một đồ thị con đã loại bỏ tất cả các chu trình. Việc triển khai DFS tiêu chuẩn không thể loại bỏ các đường đi dư thừa (tức là, đến cùng một bàn cờ bằng cách đặt các biểu tượng theo một thứ tự khác). Do đó, một trạng thái có thể được biểu diễn bởi nhiều nút trong các nhánh khác nhau! Chúng ta quan sát thấy:

* Cây tìm kiếm hoàn chỉnh có độ sâu tối đa $m=9$
* Hệ số nhánh tối đa $b=9$ (cho nước đi đầu tiên).

DFS có:

* độ phức tạp không gian là $O(bm)$ (đường đi hiện tại cộng với biên) và
* độ phức tạp thời gian là $O(b^m)$ (số lượng nút được mở rộng).

**Mã nguồn:**

Python

# Độ phức tạp không gian O(bm):

9\*9

**Kết quả:**

81

**Mã nguồn:**

Python

# Độ phức tạp thời gian O(b^m):

9\*\*9

**Kết quả:**

387420489

Hệ số nhánh giảm sau mỗi nước đi. Cấp độ đầu tiên có hệ số nhánh là 9, cấp độ thứ hai là 8, v.v. Tổng số nút là:

| **Cấp độ (Level)** | **Số nút (# of nodes)** |
| --- | --- |
| Gốc (root) | $1$ |
| Cấp 1 | $9 = 9!/8!$ |
| Cấp 2 | $9 \times 8 = 9!/7!$ |
| Cấp 3 | $9 \times 8 \times 7 = 9!/6!$ |
| ... | ... |
| Cấp 9 | $9 \times 8 \times \dots \times 2 \times 1 = 9!$ |

Tổng số nút của cây trò chơi nhỏ hơn (một số trò chơi kết thúc sớm) so với tổng các nút ở trên. Giới hạn trên cho số lượng nút là:

$\sum\_{i = 0}^9 \frac{9!}{i!}$

**Mã nguồn:**

Python

sum = 1

partial\_fac = 1

print("level\t# nodes")

print("root\t 1")

for i in range(9, 0, -1):

partial\_fac \*= i

sum += partial\_fac

print(10-i, "\t", partial\_fac)

sum

**Kết quả:**

level # nodes

root 1

1 9

2 72

3 504

4 3024

5 15120

6 60480

7 181440

8 362880

9 362880

986410

Vì một số chuỗi bị cắt ngắn do chiến thắng, chúng ta mong đợi có ít nút hơn trong cây trò chơi hoàn chỉnh.

* **Đường đi dư thừa (Redundant paths):** Ước tính kích thước cây chứa các đường đi dư thừa.
* **Chu trình (Cycles):** Tic-tac-toe không có chu trình (bạn không thể xóa một biểu tượng đã chơi).

Lưu ý: Kích thước này làm cho tic-tac-toe trở thành một bài toán rất nhỏ có thể dễ dàng giải quyết bằng tìm kiếm trên cây. Hầu hết các trò chơi và bài toán thực tế đều quá lớn và không thể giải quyết theo cách này. Ví dụ, cờ vua có kích thước không gian trạng thái ước tính là $>10^{30}$.

Chúng ta sẽ học một số phương pháp giải quyết vấn đề này sau.

**Triển khai**

Chúng ta cần triển khai các hàm sau:

* Actions(s): Các nước đi hợp lệ trong trạng thái s.
* Result(s, a): Mô hình chuyển tiếp.
* Terminal(s): Kiểm tra trạng thái kết thúc.
* Utility(s): Giá trị tiện ích cho người chơi Max tại các trạng thái kết thúc.

Trạng thái (một vị trí trên bàn cờ) sẽ được triển khai dưới dạng một vector (trong Python là một danh sách) có độ dài 9. Các giá trị là ' ', 'x', 'o'. Các hành động chỉ là chỉ số trong danh sách (0, 1, 2, ..., 8).

Việc triển khai các hàm này và một số hàm trợ giúp khác có thể được tìm thấy trong tictactoe.py và được nhập từ tệp đó.

**Mã nguồn:**

Python

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

from tictactoe import show\_board, show\_board\_text

**Ví dụ**

**Mã nguồn:**

Python

board = empty\_board()

board[0] = 'x'; board[3] = 'x'; board[6] = 'x'; board[1] = 'o'; board[4] = 'o'

display(board)

**Kết quả:**

['x', 'o', ' ', 'x', 'o', ' ', 'x', ' ', ' ']

**Mã nguồn:**

Python

show\_board\_text(board)

**Kết quả:**

[['x' 'o' ' ']

['x' 'o' ' ']

['x' ' ' ' ']]

**Mã nguồn:**

Python

show\_board(board)

**Kết quả:**

[Hình ảnh hiển thị bàn cờ Tic-Tac-Toe với 3 'x' ở cột đầu tiên và 2 'o' ở cột thứ hai]

**Mã nguồn:**

Python

actions(board)

**Kết quả:**

[2, 5, 7, 8]

**Mã nguồn:**

Python

board2 = empty\_board()

board2 = result(board2, 'x', 4)

show\_board(board2)

**Kết quả:**

[Hình ảnh hiển thị bàn cờ Tic-Tac-Toe với một chữ 'x' ở ô chính giữa]

**Mã nguồn:**

Python

print(terminal(board))

print(terminal(board2))

**Kết quả:**

True

False

**Thử nghiệm**

**Đường cơ sở: Người chơi ngẫu nhiên**

Một tác nhân người chơi hoàn toàn ngẫu nhiên có thể được sử dụng làm đường cơ sở yếu. Một tác nhân ngẫu nhiên đơn giản và môi trường trò chơi được triển khai trong tictactoe.py dưới dạng random\_player() và play().

**Mã nguồn:**

Python

from tictactoe import random\_player, play

**Ngẫu nhiên vs. Ngẫu nhiên**

Hãy để một người chơi ngẫu nhiên đấu với một người chơi ngẫu nhiên khác trong 100 ván và xem số lần thắng.

**Mã nguồn:**

Python

%time display(play(random\_player, random\_player, N = 100))

**Kết quả:**

{'x': 68, 'o': 24, 'd': 8}

CPU times: user 80.9 ms, sys: 76.7 ms, total: 158 ms

Wall time: 51.8 ms