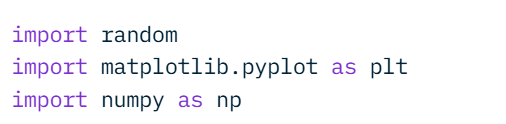
PREVIEW CODE

**Bài toán** : Bài toán 8 con hậu. Ta sẽ quy định mỗi trạng thái là sắp xếp của cả 8 con hậu trên bàn cờ, sao cho mỗi cột chỉ gồm 1 con. Trạng thái xuất phát được khởi tạo ngẫu nhiên. Hàm mục tiêu là hàm heuristic được tính bằng số các đôi hậu đang đe dọa nhau. Như vậy hàm mục tiêu càng nhỏ thì càng gần với trạng thái đích cần tìm. Trạng thái đích là trạng thái có hàm mục tiêu nhỏ nhất (bằng 0)

1. **Leo đồi ngẫu nhiên**
2. Phần import và thiết lập



Ta sẽ cung cấp các thư viện cần thiết cho thuật toán :

* Random sử dụng cho các hàm ngẫu nhiên ( random.sample , random.randint).
* Thư viện matplotlib.pylot và numpy để hỗ trợ việc minh họa bàn cờ

1. Hàm is\_attacking

* Ta định nghĩa hàm is\_attacking nhận board( mảng trạng thái) và chỉ số cột (i,j)
* Lấy hàng của 2 quân hậu : row\_i=board[i] , row\_j=board[j]
* Trả về True nếu hai quân hậu tấn công nhau ( row\_i==row\_j ; abs(i-j)==abs(row\_i – row\_j)
* Chức năng : Nhẳm kiểm tra xem 2 quân hậu ở cột i và j có tấn công nhau hay không . Là hàm con của hàm heuristic.

1. Hàm heuristic

A computer code with text

AI-generated content may be incorrect.

* Ta định nghĩa hàm heuristic nhận mảng board
* Khởi tạo biến attacks đếm số cặp quân hậu tấn công nhau
* Ta sử dụng 2 vòng lặp for lồng nhau kiểm tra tất cả cặp ở cột (i,j) ( i<j) tránh đánh lại
* Gọi hàm is\_attackings để kiểm tra cặp (i,j) , nếu tấn công thì tăng biến attacks lên
* Sau cùng trả về tổng số cặp tấn công đã ghi nhận .
* Tổng độ phức tạp : O^(n^2)

1. Hàm generate\_initial\_board()

A close-up of a text

AI-generated content may be incorrect.

* Định nghĩa hàm không tham số , sau đó tạo hoán vị ngẫu nhiên của [0,1,...,7] bằng random.sample , trả về mảng 8 phần tử
* Chức năng : Tạo trạng thái ban đầu hợp lệ (mỗi cột một quân hậu, không trùng hàng).
* Tổng độ phức tạp : O(n) thời gian , O(n) không gian .

]

1. get\_neighbor

A computer code with text

AI-generated content may be incorrect.

* Ta định nghĩa hàm get\_neighbor nhận mảng board , sau đó sao chép mảng board thành new\_board
* Chọn ngẫu nhiên cột (0-7) , chọn ngẫu ngẫu nhiên hàng mới (0-7)
* 1 vòng lặp while để đảm bảo new\_row khác hàng hiện tại , sau đó gán new\_row vào new\_board[col]
* Sau cùng trả về giá trị new\_board
* Chức năng : Tạo trạng thái hàng xóm bằng cách di chuyển quân hậu trong một cột ngẫu nhiên sang hàng mới ngẫu nhiên (khác hàng hiện tại).
* Tổng độ phức tạp : O(n) thời gian , O(n) không gian .

1. Hàm plot\_board(board, title='Chessboard')

* Đây là hàm có chức năng vẽ bàn cờ minh họa
* Độ phức tạp : O(n^2) thời gian , O(n) không gian

1. Hàm stochastic\_hill\_climbing

A screenshot of a computer code

AI-generated content may be incorrect.

* Đây là hàm quan trọng nhất , chứa thuật toán giải quyết bài toán
* Định nghĩa hàm với tham số max\_attempts (mặc định 10000).
* Gọi generate\_initial\_board để tạo trạng thái ban đầu.
* Tính heuristic ban đầu bằng heuristic(board) . Khởi tạo biến attempts (đếm số lần thử) và steps (đếm số lặp).
* In tiêu đề, trạng thái, và heuristic ban đầu, sau đó vẽ bàn cờ mang trạng thái ban đầu .
* Vòng while chạy nếu current\_heuristic > 0 và attempts < max\_attempts.
* Tạo trạng thái hàng xóm bằng get\_neighbor , sau đó tính heuristic của hàng xóm .
* Nếu heuristic hàng xóm tốt hơn, cập nhật board, current\_heuristic, in bước.
* Mỗi lần lặp, tăng attempts và steps
* In kết quả ( thành công/ thất bại ) , trạng thái cuối , heuristic , số bước sử dụng
* Vẽ bàn cờ kết quả , trả về board và steps

Độ phức tạp : Dòng 2 :O(n) thời gian, O(n) không gian

Dòng 3: O(n^2) thời gian

Dòng 9: O(n^2) thời gian O(n) không gian

...

Tổng : Độ phức tạp thời gian: max\_attempts O(n^2)

Không gian : O(n)

1. Chạy chương trình

A computer screen shot of a computer screen

AI-generated content may be incorrect.

Gọi stochastic\_hill\_climbing với max\_attempts=10000, nhận solution (board) và steps.

1. Độ phức tạp thuật toán giữa 2 giải thuật leo đồi ngẫu nhiên và leo đồi ngẫu nhiên có khởi tạo lại ngẫu nhiên.

* Với giải thuật leo đồi ngẫu nhiên :
* Độ phức tạp thời gian : O(max\_attempts\*n^2)
* Độ phức tạp không gian : O(n)
* Với leo đồi ngẫu nhiên có khởi tạo lại ngẫu nhiên :
* Độ phức tạp thời gian : O(max\_restarts\*max\_attempts\*n^2)
* Độ phức tạp không gian: O(n)

**Nhận xét :**

- Sự khác biệt này xuất phát từ việc thêm vòng lặp khởi động lại để khám phá nhiều vùng không gian tìm kiếm hơn, tránh kẹt ở cực tiểu địa phương .

- Tuy nhiên, điều này đánh đổi bằng thời gian chạy cao hơn, vì mỗi lần khởi động lại là một lần chạy đầy đủ leo đồi (O(max\_attempts \* n²)

-Nếu max\_restarts=1, 2 thuật toán sẽ tương đương về độ phức tạp thời gian lẫn không gian .