**Bài 17: Heuristic Search - Lập trình AI bằng Python**

Tìm kiếm heuristic đóng một vai trò quan trọng trong trí tuệ nhân tạo. Trong bài này, ta sẽ tìm hiểu chi tiết về nó.

**1. Khái niệm Heuristic Search trong AI :**

Heuristic là một quy tắc ngón tay cái dẫn chúng ta đến giải pháp có thể xảy ra. Hầu hết các vấn đề trong trí tuệ nhân tạo đều có tính chất cấp số nhân và có nhiều giải pháp khả thi. Bạn không biết chính xác giải pháp nào là đúng và việc kiểm tra tất cả các giải pháp sẽ rất tốn kém.

Do đó, việc sử dụng heuristic thu hẹp việc tìm kiếm giải pháp và loại bỏ các tùy chọn sai. Phương pháp sử dụng heuristic để dẫn đầu tìm kiếm trong không gian tìm kiếm được gọi là Heuristic Search. Các kỹ thuật heuristic rất hữu ích vì có thể tăng cường tìm kiếm khi bạn sử dụng chúng.

**2. Sự khác biệt giữa Uninformed và Informed Search :**

Có hai loại chiến lược kiểm soát hoặc kỹ thuật tìm kiếm: không được thông tin và được thông báo :

**Uninformed Search :**

Nó còn được gọi là chiến lược tìm kiếm mù hoặc kiểm soát mù. Nó được đặt tên như vậy bởi vì chỉ có thông tin về định nghĩa vấn đề và không có thông tin bổ sung nào khác về các trạng thái. Loại kỹ thuật tìm kiếm này sẽ tìm kiếm toàn bộ không gian trạng thái để tìm giải pháp. Tìm kiếm đầu tiên theo chiều rộng (BFS) và Tìm kiếm đầu tiên theo chiều sâu (DFS) .

**Informed Search :**

Nó còn được gọi là chiến lược tìm kiếm theo phương pháp heuristic hoặc chiến lược điều khiển heuristic. Nó được đặt tên như vậy vì có thêm một số thông tin về các trạng thái. Thông tin bổ sung này rất hữu ích để tính toán tùy chọn giữa các nút con để khám phá và mở rộng. Sẽ có một hàm heuristic được liên kết với mỗi nút. Tìm kiếm đầu tiên tốt nhất (BFS), A \*, Trung bình và Phân tích là các ví dụ về tìm kiếm có thông tin.

**Constraint Satisfaction Problems (CSPs)**

Ràng buộc có nghĩa là hạn chế hoặc giới hạn. Trong AI, vấn đề thỏa mãn hạn chế là những vấn đề phải được giải quyết dưới một số ràng buộc. Trọng tâm phải là không vi phạm các ràng buộc trong khi giải quyết các vấn đề đó. Cuối cùng, khi ta đạt được giải pháp cuối cùng, CSP phải tuân theo các hạn chế.

**3. Giải quyết vấn đề thực với Constraint Satisfaction**

Các phần trước đã đề cập đến việc tạo ra các vấn đề về thỏa mãn hạn chế. Bây giờ, chúng ta hãy áp dụng điều này cho các vấn đề trong thế giới thực. Một số ví dụ về các vấn đề trong thế giới thực được giải quyết bằng cách thỏa mãn ràng buộc như sau:

**a. Quan hệ đại số :**

Với sự trợ giúp của bài toán thỏa mãn ràng buộc, chúng ta có thể giải quyết các quan hệ đại số. Trong ví dụ này, chúng ta sẽ cố gắng giải một quan hệ đại số đơn giản a \* 2 = b. Nó sẽ trả về giá trị của a và b trong phạm vi mà chúng ta sẽ xác định.

Sau khi hoàn thành chương trình Python này, bạn sẽ có thể hiểu những điều cơ bản về cách giải quyết vấn đề với sự hài lòng về ràng buộc.

Lưu ý rằng trước khi viết chương trình, chúng ta cần cài đặt gói Python có tên là python-binding

pip **install** python-**constraint**

Các bước sau đây cho bạn thấy một chương trình Python để giải quyết mối quan hệ đại số bằng cách sử dụng thỏa mãn ràng buộc:

import package constraingt :

from constraint **import** \*

Tạo object model với tên là problem() :

problem = Problem()

Bây giờ, hãy xác định các biến. Lưu ý rằng ở đây chúng ta có hai biến a và b, và chúng ta đang xác định 10 là phạm vi của chúng, có nghĩa là chúng ta đã nhận được giải pháp trong vòng 10 số đầu tiên.

problem.addVariable('a', **range**(10))

problem.addVariable('b', **range**(10))

Tiếp theo, xác định ràng buộc cụ thể mà chúng tôi muốn áp dụng cho vấn đề này. Quan sát rằng ở đây chúng ta đang sử dụng ràng buộc a \* 2 = b

problem.addConstraint(lambda a, b: a \* 2 == b)

Tạo object của model **getSolution()**

solutions = problem.getSolutions()

In output :

print (solutions)

Kết quả như sau :

[{'a': 4, 'b': 8}, {'a': 3, 'b': 6}, {'a': 2, 'b': 4}, {'a': 1, 'b': 2}, {'a': 0, 'b': 0}]

**b. Magic Square :**

Hình vuông ma thuật là sự sắp xếp các số riêng biệt, thường là số nguyên, trong một lưới vuông, trong đó các số trong mỗi hàng, trong mỗi cột và các số trong đường chéo, tất cả cộng lại thành một số được gọi là “hằng số ma thuật” .

Sau đây là cách thực thi từng bước mã Python đơn giản để tạo các ô vuông ma thuật:

Xác định một hàm có tên magic\_square, như sau :

**def** **magic\_square**(matrix\_ms):

iSize = len(matrix\_ms[0])

sum\_list = []

Đoạn mã sau đây hiển thị mã cho chiều dọc của hình vuông:

**for** col in **range**(iSize):

sum\_list.append(sum(row[col] **for** row in matrix\_ms))

Chiều ngang của hình vuông:

sum\_list.extend([sum (lines) **for** lines **in** matrix\_ms])

for i in range(0,iSize):

dlResult +=matrix\_ms[i][i]

sum\_list.append(dlResult)

drResult = 0

for i in range(iSize-1,-1,-1):

drResult +=matrix\_ms[i][i]

sum\_list.append(drResult)

if len(set(sum\_list))>1:

return False

return True

Đưa ra giá trị của ma trận và kiểm tra kết quả :

print(magic\_square([[1,2,3], [4,5,6], [7,8,9]]))

Bạn có thể quan sát rằng kết quả đầu ra sẽ là False vì tổng không lên đến cùng một số.

print(magic\_square([[3,9,2], [3,5,7], [9,1,6]]))

Bạn có thể quan sát rằng kết quả đầu ra sẽ là True vì tổng là cùng một số, ở đây là 15.

[**BÀI TIẾP THEO: GAMING - PHẦN 1 >>**](https://vncoder.vn/bai-hoc/gaming-phan-1-414)