ĐẠI HỌC NHA TRANG

Chuyên ngành: Công Nghệ Thông Tin



BÁO CÁO HỌC PHẦN TRÍ TUỆ NHÂN TẠO

Sinh Viên Thực Hiện: xxx

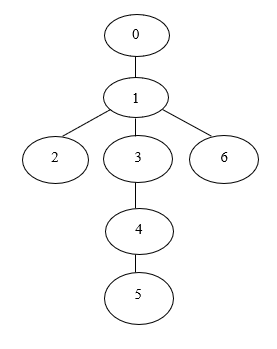
Mã Số Sinh Viên: xxx

Lớp: xxx

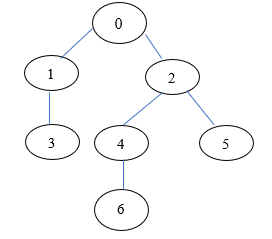
Nha Trang 2020

**1.Bài toán tìm kiếm heuristic**

**Duyệt chiều rộng**

Trong lý thuyết đồ thị, tìm kiếm theo chiều rộng (BFS) là một thuật toán tìm kiếm trong đồ thị trong đó việc tìm kiếm chỉ bao gồm 2 thao tác: (a) cho trước một đỉnh của đồ thị; (b) thêm các đỉnh kề với đỉnh vừa cho vào danh sách có thể hướng tới tiếp theo. Có thể sử dụng thuật toán tìm kiếm theo chiều rộng cho hai mục đích: tìm kiếm đường đi từ một đỉnh gốc cho trước tới một đỉnh đích, và tìm kiếm đường đi từ đỉnh gốc tới tất cả các đỉnh khác. Trong đồ thị không có trọng số, thuật toán tìm kiếm theo chiều rộng luôn tìm ra đường đi ngắn nhất có thể. Thuật toán BFS bắt đầu từ đỉnh gốc và lần lượt nhìn các đỉnh kề với đỉnh gốc. Sau đó, với mỗi đỉnh trong số đó, thuật toán lại lần lượt nhìn trước các đỉnh kề với nó mà chưa được quan sát trước đó và lặp lại. Hình minh họa mảng khi chạy code chương trình dùng tìm kiếm theo chiều rộng như trên.

**Duyệt chiều sâu**

Tìm kiếm ưu tiên chiều sâu hay tìm kiếm theo chiều sâu (tiếng Anh: Depth-first search - DFS) là một thuật toán duyệt hoặc tìm kiếm trên một cây hoặc một đồ thị. Thuật toán khởi đầu tại gốc (hoặc chọn một đỉnh nào đó coi như gốc) và phát triển xa nhất có thể theo mỗi nhánh.

Thông thường, DFS là một dạng tìm kiếm thông tin không đầy đủ mà quá trình tìm kiếm được phát triển tới đỉnh con đầu tiên của nút đang tìm kiếm cho tới khi gặp được đỉnh cần tìm hoặc tới một nút không có con. Khi đó giải thuật quay lui về đỉnh vừa mới tìm kiếm ở bước trước. Trong dạng không đệ quy, tất cả các đỉnh chờ được phát triển được bổ sung vào một ngăn xếp LIFO. Hình minh họa mảng khi chạy code chương trình dùng tìm kiếm theo chiều sâu như trên

**2. Phát triển trò chơi**

**Nim**

Trò chơi truyền thống Nim được chơi với một số đồng xu được sắp xếp thành nhiều đống, cách sắp số lượng đồng xu và đống tùy thuộc vào bạn. Có hai người chơi, khi đến lượt, người chơi có thể lấy một số lượng tùy ý đồng xu từ một đống duy nhất. Họ phải lấy ít nhất 1 đồng xu và họ không được lấy các đồng xu từ đống khác. Người thắng là người lấy đồng xu cuối cùng, nghĩa là không còn đồng xu nào sau nước đi của người đó.

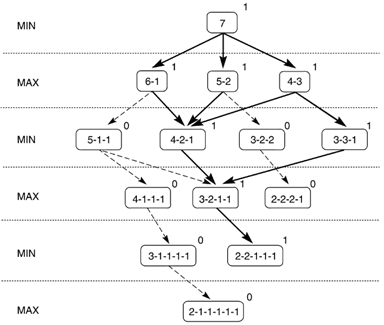
Bí mật để tìm ra chiến lược thắng nằm ở cách viết số kích thước của các đống (số lượng đồng xu của từng đống) theo dạng nhị phân và cộng chúng lại với nhau, nhưng không phải cộng theo cách thông thường, mà là cộng theo một cách khác, gọi là "Phép cộng Nim".

Để cộng một số số nhị phân cho trước bằng "phép cộng Nim", đầu tiên bạn viết chúng theo dạng cột (viết số sau đứng dưới số trước), như cách thực hiện phép cộng thông thường. Sau đó bạn nhìn từng cột theo thứ tự. Nếu số lượng các con số 1 là lẻ, bạn viết 1 ở dưới chúng, nếu số lượng chẵn, viết số 0 dưới chúng. Làm như vậy với từng cột cho ra số nhị phân mới, và đó là kết quả của "phép cộng Nim”

Khi Charles Bouton (người phát hiện cách giải bài toán này) phân tích trò chơi Nim, ông phát hiện ra 2 điều nắm giữ chìa khóa dẫn đến chiến lược thắng.

Điều 1: Giả sử đến lượt bạn và tổng Nim các đồng xu trong các đống bằng 0. Cho dù bạn làm gì đi nữa, tổng Nim các đồng xu sau khi bạn thực hiện nước đi đều khác 0.

Điều 2: Giả sử đến lượt bạn và tổng Nim các đồng xu trong các đống khác 0. Lúc đó, luôn có một nước đi đảm bảo rằng sau khi bạn thực hiện nước đi, tổng Nim các đồng xu trong các đống bằng 0.



Khi chạy chương trình code bạn luôn là người thua cuộc dù bạn có đi thế nào cũng như vậy cả

**Puzzle**

Xét ví dụ là bài toán 8 puzzle với công thức sau:

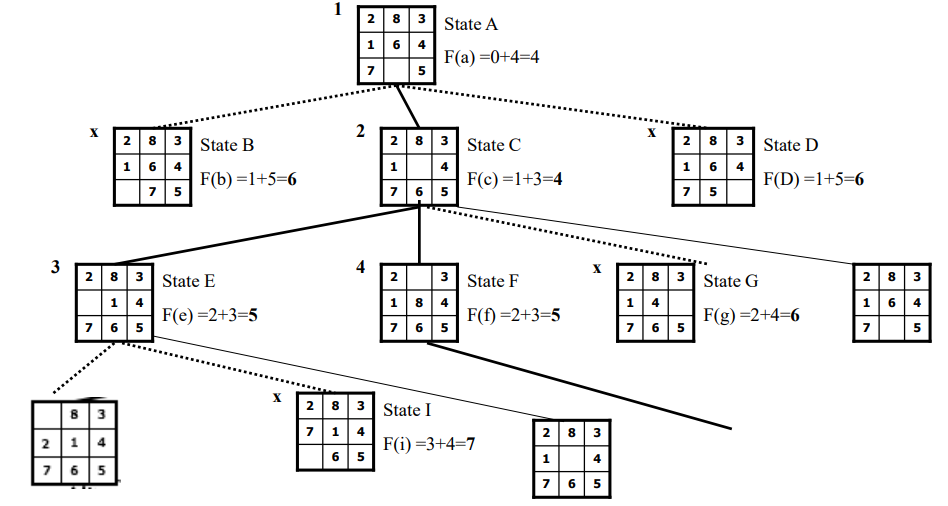
Hàm lượng giá: F(n) = G(n) + H(n)

Với G(n): số lần chuyển vị trí đã thực hiện

H(n): Số miếng nằm sai vị trí

Nút X có giá trị heuristic tốt hơn nút Y nếu F(x) < F(y).

Ta có hoạt động của giải thuật Best First search trên như hình sau:



Khi chạy code phần ma trận ban đầu và kết thúc thì bạn phải nhập dữ liệu và khi xuất là quá trình thực hiện ma trận bằng thuật toán minmax theo chiều ký hiệu “->” từ trên xuống

**Tic-tac-toe**

Chìa khóa của thuật toán Minimax chính là lượt chơi qua lại giữa hai người chơi, khi mà cả hai đều mong muốn nước đi có điểm cao nhất. Điểm số của từng nước đi của ta được quyết định bởi đối thủ, người quyết định nước nào sẽ mang lại điểm số nhỏ nhất cho chúng ta. Ngược lại, điểm số của đối thủ lại được quyết định bởi chúng ta, người tìm cách tối ưu hóa điểm số. Nói ngắn gọn thì thuật toán Minimax trong trò chơi Tic Tac Toe chính là thử hết các nước đi có thể cho đến khi trò chơi kết thúc.

Giả dụ đang tới lượt X, thì trò chơi sẽ có quy trình như sau:

Nếu trò chơi kết thúc, trả về điểm số dưới góc nhìn của X.

Không thì thu thập các nước đi có thể ở lượt mới.

Tạo một mảng (một cấu trúc dữ liệu bao gồm một nhóm các giá trị phần tử hoặc biến) chứa các điểm số tương ứng với các nước nêu trên.

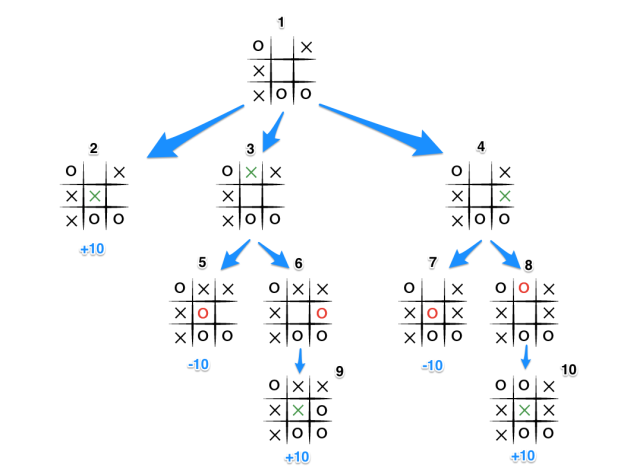
Đối với từng tình huống, cập nhật điểm số tính bởi thuật Minimax vào mảng nêu trên.

Nếu lượt đi là của X, trả về điểm số lớn nhất trong mảng.

Nếu lượt đi là của O, trả về điểm số nhỏ nhất trong mảng.

Bạn sẽ để ý rằng thuật toán Minimax là một thuật toán đệ quy – sự qua lại giữa lượt đi của hai người chơi cho đến khi số điểm cuối cùng được tìm thấy.

Hãy cũng xem qua nguyên lý của thuật toán đối với một cây nước đi đầy đủ và chỉ ra làm sao mà nước đi dẫn tới chiến thắng được lựa chọn:



Tình huống 1 là lượt đi của X. X tạo ra 3 tình huống 2, 3, và 4 và áp dụng thuật toán Minimax trên từng tình huống.

Tình huống 2 trả về điểm +10 cho mảng điểm của tình huống 1, bởi vì trò chơi đã đến hồi kết.

Tình huống 3 và 4 chưa tới hồi kết nên lần lượt tạo ra các tình huống 5, 6 và 7, 8 đồng thời áp dụng thuật toán Minimax.

Tình huống 5 trả về điểm -10 cho mảng điểm của tình huống 3, tương tự với tình huống 7 trả về điểm -10 cho mảng 4.

Tình huống 6 và tình huống 8 chỉ tạo ra một tình huống, đồng thời là tình huống cuối cùng dẫn tới trò chơi kết thúc với X là người thắng, nên cả hai sẽ lần lượt trả về điểm +10 cho mảng 3 và 4.

Bởi vì sau tình huống thứ 3 và thứ 4 là lượt đi của O, nên O sẽ đi nước đi dẫn tới điểm số thấp nhất. Bởi vì cả hai tình huống đều có mảng chứa +10 và -10, nên tình huống 3 và 4 sẽ trả về -10 cho mảng điểm của tình huống 1 (điểm thấp nhất).

Trong mảng điểm của tình huống 1 sẽ chứa các điểm lần lượt là +10, -10, -10. Vì tình huống 1 là lượt đi của X nên X sẽ lựa nước đi có điểm cao nhất, nghĩa là chọn nước đi dẫn tới số điểm +10 – nước đi tương tự tình huống thứ 2.

Khi chạy code chương trình thì nếu bạn hoặc máy tính ai chơi đầu thì luôn khả năng thắng rất cao , tệ thì hòa, chơi sau thì hòa hoặc thua sẽ rất cao

**Bao búa kéo**

Với code chương trình thì dùng thư viện random (ngẩu nhiên) cùng với các lệnh if, else để viết chương trình cùng các điều kiện xảy ra và cách giải quyết tương ứng, khi chơi người dùng sẽ thấy khả năng hòa rất cao, bạn cũng có thể tự tin nếu thắng trò này

**3. Bài toán hệ chuyên gia**

**Suy diễn tiến**

Từ đề bài ta có các luật như sau

R1: Nếu nóng và có khói thì có lửa

R2: Nếu alert\_beeps (báo động cháy) thì có khói

R3: Nếu cháy thì switch sprinklers on (bật vòi phun nước)

Thì sprinklers on (vòi phun nước sẽ hoạt động)

R4: Nếu trời khô thì switch\_on\_humidifier (bật vòi chống cháy có độ ẩm)

Thì humidifier\_on (vòi chống cháy có độ ẩm sẽ được bật)

R5: NẾU sprinkler\_on (bật vòi phun nước) thì không có trời khô

F1: báo động ; F2: nóng; F3: trời khô

Các luật dùng được thêm

Sử dụng R2:

- Thêm khói: bây giờ R1 và R4 phù hợp

• Sử dụng R1:

- Thêm lửa: bây giờ R3 và R4 phù hợp

• Sử dụng R3:

- Thêm sprinkler\_on: đấu R4 và R5

• Sử dụng R5:

- Xóa khô: bây giờ R4 không khớp

• Lưu ý rằng R4 không bao giờ được sử dụng trong trình tự này; vì vậy sự lựa chọn có thể ảnh hưởng đến kết quả

**Suy diễn lùi**

Giả thuyết có các biến như sapa và hanoi

Ta có luật như sau:

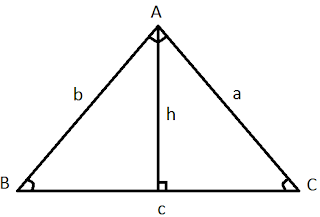
Nơi nào (X) là chỗ có tuyết nếu chỗ đó (X) có mưa (rain) và lạnh (cold)

Và khi chạy thì kết quả là sapa luôn cho là true (đúng)

Với câu lệnh logic là: snowy(sapa).

**Mạng ngữ nghĩa (bài toán tam giác)**

Sơ lược về bài toán tam giác bằng mạng ngữ nghĩa.



Có 22 yếu tố liên quan đến cạnh và góc của tam giác. Để xác định một tam giác thì ta phải có 3 yếu tố. Trong đó phải có yếu tố cạnh. Như vậy có khoảng 33C2-1 (khoảng vài ngàn) cách để xây dựng hay xác định một tam giác. Theo thống kê thì có khoảng trên 200 công thức liên quan đến cạnh và góc của tam giác.

Để giải bài toán này bằng mạng ngữ nghĩa, ta phải sử dụng khoảng 200 đỉnh để chứa công thức và khoảng 22 đỉnh để chứa các yếu tố của tam giác. Mạng ngữ nghĩa cho bài toán này có cấu trúc như sau:

Đỉnh của đồ thị bao gồm 2 loại:

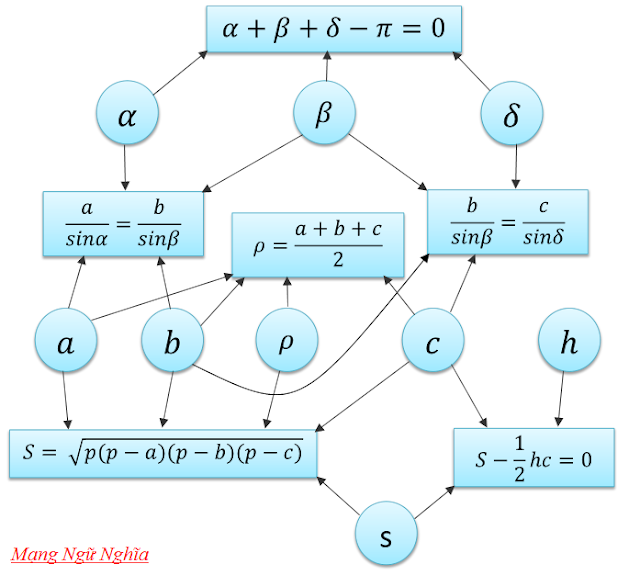
- Đỉnh chứ công thức (ký hiệu bằng hình chữ nhật).

- Đỉnh chứa các yếu tố tam giác (ký hiệu bằng hình tròn).

Cung: chỉ nối từ đỉnh hình tròn đến đỉnh hình chữ nhật cho biết yếu tố tam giác xuất hiện trong công thức nào(không có trường hợp cung nối giữa hai đỉnh hình tròn hay nối giữa 2 đỉnh hình chữ nhật).

\* Lưu ý : trong một công thức liên hệ giữa n yếu tố của tam giác, ta giả định rằng nếu đã biết giá trị của n-1 yếu tố thì sẽ tính được giá trị của yếu tố còn lại. Chẳng hạn như trong công thức tổng 3 góc của tam giác bằng 1800 thì khi biết được hai góc, ta sẽ tính được góc còn lại.

Mạng ngữ nghĩa được xây dựng:



Cơ chế suy diễn thực hiện theo thuật toán "loang" đơn giản sau :

B1 : Kích hoạt những đỉnh hình tròn đã cho ban đầu (những yếu tố đã có giá trị)

B2 : Lặp lại bước sau cho đến khi kích hoạt được tất cả những đỉnh ứng với những yếu tố cần tính hoặc không thể kích hoạt được bất kỳ đỉnh nào nữa.

Nếu một đỉnh hình chữ nhật có cung nối với n đỉnh hình tròn mà n-1 đỉnh hình tròn đãđược kích hoạt thì kích hoạt đỉnh hình tròn còn lại (và tính giá trị đỉnh còn lại này thông qua công thức ở đỉnh hình chữ nhật).

Ví dụ : "Cho hai góc a, b và chiều dài cạnh a của tam giác. Tính chiều dài đường cao hC". Với mạng ngữ nghĩa đã xây dựng. Các bước thực hiện của thuật toán như sau :

Bắt đầu : đỉnh a, b, a của đồ thị được kích hoạt.

Công thức (1) được kích hoạt (vìa, b,a được kích hoạt).

- Từ công thức (1) tính được cạnh b. Đỉnh b được kích hoạt.Công thức (4) được kích hoạt (vì a, b).

- Từ công thức (4) tính được góc d. Công thức (2) được kích hoạt (vì 3 đỉnh b,d,b được kích hoạt).

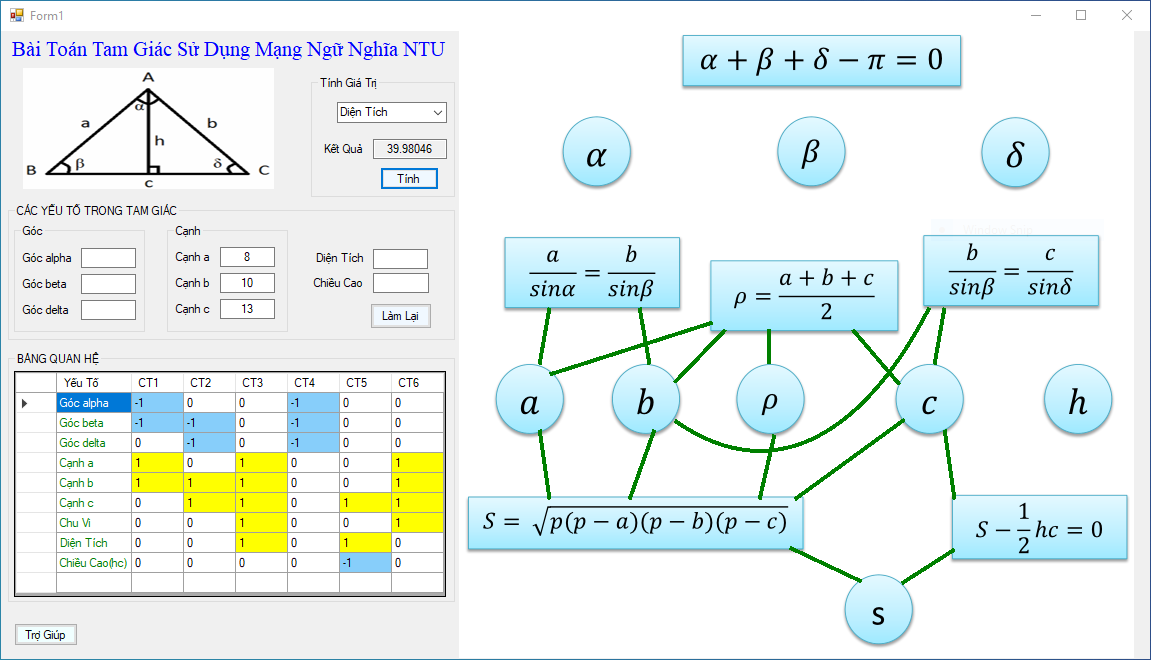
- Từ công thức (2) tính được cạnh c. Đỉnh c được kích hoạt. Công thức (6) được kích hoạt (vì 3 đỉnh a, b, c được kích hoạt) .

- Từ công thức (6) tính được chu vi P. Đỉnh P được kích hoạt. Công thức (3) được kích hoạt (vì 4 đỉnh a, b, c, P được kích hoạt).

- Từ công thức (3) tính được diện tích S. Đỉnh S được kích hoạt(vì 2 đỉnh S,c được kích hoạt) .

- Từ công thức (5) tính được chiều cao hC. Đỉnh hC được kích hoạt. Giá trị hC đã được tính.

Thuật toán kết thúc.



Chương trình được xây dựng trên môi trường visual studio 2019, ngôn ngữ C#

Các yếu tố tam giác: phần này để nhập các yếu tố tam giác.

Bảng Quan Hệ: Mô tả các yếu tố chưa tính(-1) và các yếu tố đã tính(1).

Mạng Ngữ Nghĩa: Thể hiện trực quan mối liên hệ giữa các yếu tố và công thức.

Tính Giá Trị: Tính các giá trị còn lại

Về mặt chương trình, ta có thể cài đặt mạng ngữ nghĩa giải bài toán tam giác bằng một mảng hai chiều A trong đó:

Cột : ứng với công thức. Mỗi cột ứng với một công thức tam giác khác nhau (đỉnh hình chữ nhật).

Dòng : ứng với yếu tố tam giác. Mỗi dòng ứng với một yếu tố tam giác khác nhau (đỉnh hình tròn).

Phần tử A[i, j] = -1 nghĩa là trong công thức ứng với cột j có yếu tố tam giác ứng với cột i. Ngược lại A[i,j] = 0.

Để thực hiện thao tác "kích hoạt" một đỉnh hình tròn, ta đặt giá trị của toàn dòng ứng với yếu tố tam giác bằng 1.

Để kiểm tra xem một công thức đã có đủ n-1 yếu tố hay chưa (nghĩa là kiểm tra điều kiện "đỉnh hình chữ nhật có cung nối với n đỉnh hình tròn mà n-1 đỉnh hình tròn đã được kích hoạt"), ta chỉ việc lấy hiệu giữa tổng số ô có giá trị bằng 1 và tổng số ô có giá trị -1 trên cột ứng với công thức cần kiểm tra. Nếu kết quả bằng n, thì công thức đã có đủ n-1 yếu tố.

Cách sử dụng chương trình demo:

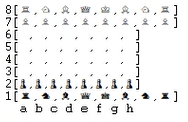
- Nhập vào các giá trị lúc đầu của tam giác vào các textbox trong chương trình.

- Chọn giá trị cần tính trong combobox .

- Nhấn nút Tính để thực hiện tính yếu tố tam giác theo mạng ngữ nghĩa đã được cài đặt.

- Để bắt đầu với bài toán mới, nhấn vào nút Làm Lại.

**Sử dụng phần mềm prolog phát triển ứng dụng (game chơi cờ vua)**



Project có 4 phần tính năng cho người chơi lựa chọn: là

Máy tính với người

Máy tính với máy tính

Con người với con người

Người với máy tính

Mỗi bên thay phiên nhau (Con người và / hoặc Máy tính) Sau mỗi lượt / nước đi

Và được phân loại theo các tính năng như sau:

Hậu- chỉ có thể di chuyển theo chiều ngang, dọc, chéo

Hiệp sĩ - di chuyển theo hình chữ L

Xe- di chuyển theo chiều ngang và chiều dọc

Tượng - di chuyển theo đường chéo

Vua- có thể di chuyển trong bất kỳ ô lân cận nào

Tốt - chỉ có thể di chuyển về phía trước 1 khoảng trống hoặc 2 khoảng trống cho lần di chuyển đầu tiên trong trò chơi

Tất cả các lần thay đổi quy định phải nằm trong không gian bàn cờ

Tất cả các nước đi phải có một con đường mở không có quân cờ giữa vị trí bắt đầu và vị trí kết thúc (ngoại trừ các hiệp sĩ)

Quyết định nước đi tốt nhất dựa trên điểm số

Kiểm tra trạng thái bàn cờ

Điều kiện chiến thắng - trạng thái rõ ràng

Điều kiện thua - trạng thái hiển nhiên

Điều kiện ràng buộc

King không ở trong tù và không có động thái hợp pháp nào đối với người chơi hiện tại

Nếu 3 nước đi cuối cùng của mỗi người chơi là “giống nhau” thì hòa

Đây là các điều kiện của các thành phần trên

**Thuật toán Vương Hạo**

Ta có đề như sau : cho {-p v q, -q v r} kết luận {-p, r }

Bài làm

B1: Dạng chuẩn: -p v q, -q v r -> -p, r

B2: Tách thành 2 vế

1. {-p, -q v r} -> -p, r (được chứng minh)
2. {q, -q v r} -> -p , r

B3: Từ (2) ta có:

(2.1){q, -q} => -p, r

(2.2){q, r} => -p, r (được chứng minh)

B4: Từ (2.1) ta được:

{q} => -p, r, q (được chứng minh)

Kết luận: Vậy tất cả nhánh con của bài toán đã được chứng minh

Với phần code thì so với công thức của đề là

V là tuyển

^ là hội, phủ

- là phủ định

Thì so với nội dùng trong code python thì các ký hiệu đã được đổi thành

V là tuyển

^ là hội, phủ

! là phủ định

Và ta có mẫu quy ước tương tự với đề bài toán trong code python

menhde1 = tuyen(phu(val('p')),val('q'))

menhde2 = tuyen(phu(val('q')),val('r'))

menhde3 = tuyen(phu(val('p')),val('r'))

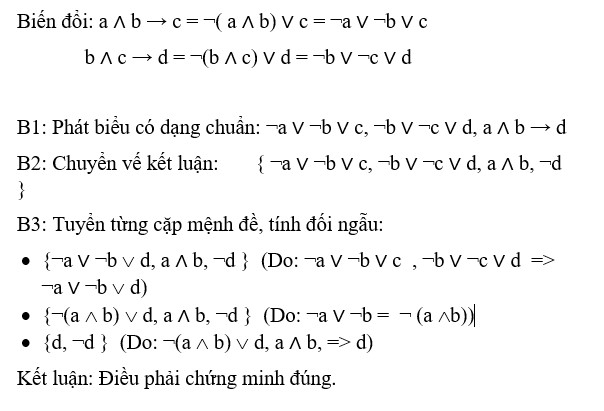
mybieuthuc = bieuthuc([menhde1,menhde2],[menhde3])

dòng code chạy cũng là quá trình thực hiện so với lời giải tự luận lý thuyết

**Thuật toán Robinson**

Cho đề bài toán như sau

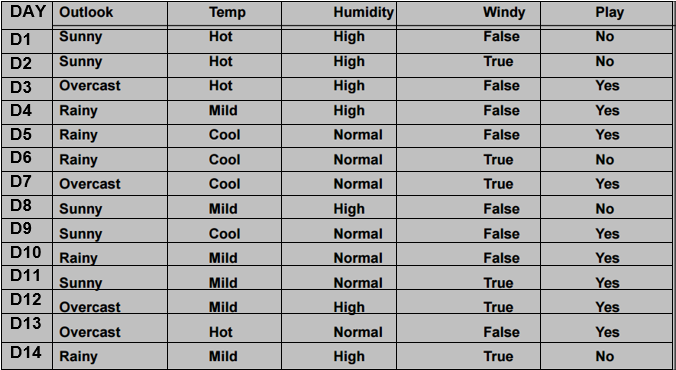
Cho {a ^b ->c, b^c ->d, a^b}. Hỏi d ?



**4. Bài toán máy học – Machine learning**

**Thuật giải Quinland**

Ta có đề bài ví dụ sau



Vector đặc trưng cho các thuộc tính:

**Thuộc tính: Outlook**

V(outlook = sunny)**=**(T(sunny,No),T(sunny,Yes))=(3/5,2/5)

V(outlook = overcast) =(T(overcast,No),T(overcast,Yes))  =(0/4,4/4) = **(0,1)**

V(outlook =rain) =(T(rain,No),T(rain,Yes))=(2/5,3/5)

**Thuộc tính: Temp**

V(Temp = hot) =(T(hot,No),T(hot,Yes)) = (2 /4 ,2/4)

V (Temp = mild) =(T(mild,No),T(mild,Yes)) = ( 2/6 ,4/6)

V(Temp = cool) =(T(cool,No),T(cool,Yes)) = ( 1/4 ,3/4)

**Thuộc tính: Humidity**

V(Humidity = high) =(T(high,No),T(high,Yes)) = ( 4/7 ,3/7)

V(Humidity = normal) =(T(normal,No),T(normal,Yes)) = ( 1/7 ,6/7)

**Thuộc tính: Wind**

V(wind= false) =(T(false,No),T(false,Yes)) = (2 /8 ,6/8)

V(wind=true) =(T(true No),T(true,Yes)) = (3 /6 ,3/6)

Như vậy, thuộc tính Outlook có số vector đơn vị nhiều nhất nên sẽ được phân hoạch.

Sau khi phân hoạch theo Outlook xong, chỉ có phân hoạch theo Outlook (Psunny) và (Prain)  là còn chứa kết quả là Yes và No nên ta sẽ tiếp tục phân hoạch tập này. Ta sẽ thực hiện thao tác tính vector đặc trưng tương tự đối với các thuộc tính còn lại (Temp,Humidity,Wind). Trong phân hoạch (Psunny), (Prain)  , tập dữ liệu của chúng ta còn lại là :

TH1: Trong phân hoạch (Psunny) , tập dữ liệu của chúng ta còn lại là :

**Thuộc tính: Temp**

V(Temp = hot) =(T(hot,No),T(hot,Yes)) = (2 /2 ,0/2) = (1,0)

V (Temp = mild) =(T(mild,No),T(mild,Yes)) = ( 1/2 ,1/2)

V(Temp = cool) =(T(cool,No),T(cool,Yes)) = ( 0/1 ,2/1) = (0,1)

**Thuộc tính: Humidity**

V(Humidity = high) =(T(high,No),T(high,Yes)) = ( 3/3 ,0/3) = (1,0)

V(Humidity = normal) =(T(normal,No),T(normal,Yes)) = ( 0/2,2/2) = (0,1)

**Thuộc tính: Wind**

V(wind=false) =(T(false,No),T(false,Yes)) = (2/3,1/3)

V(wind= true) =(T(true,No),T(true,Yes)) = (1/2,1/2)

Như vậy, thuộc tính **Humidity** có số vector đơn vị nhiều nhất nên sẽ được phân hoạch.

TH2: Trong phân hoạch (Prain) , tập dữ liệu của chúng ta còn lại là :

**Thuộc tính: Temp**

V(Temp = hot) =(T(hot,No),T(hot,Yes)) = (0,0)

V (Temp = mild) =(T(mild,No),T(mild,Yes)) = ( 1/3,2/3)

V(Temp = cool) =(T(cool,No),T(cool,Yes)) = ( 1/2 ,1/2)

**Thuộc tính: Humidity**

V(Humidity = high) =(T(high,No),T(high,Yes)) = ( 1/2 ,1/2)

V(Humidity = normal) =(T(normal,No),T(normal,Yes)) = ( 1/3,2/3)

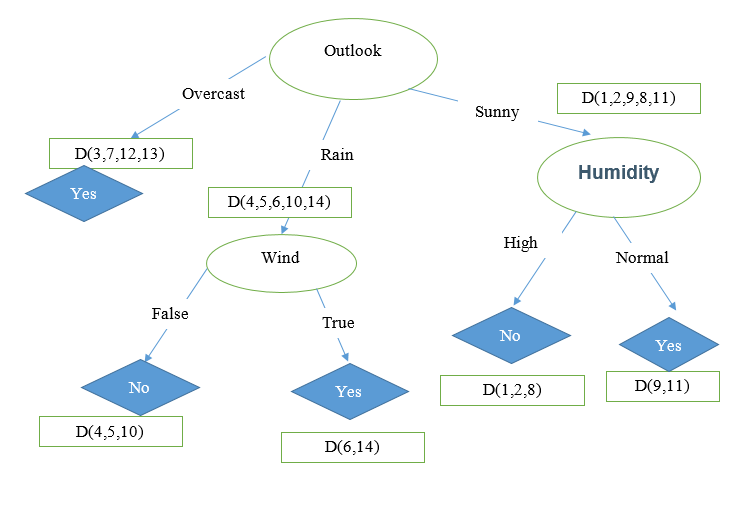
**Thuộc tính: Wind**

V(wind=false) =(T(false,No),T(false,Yes)) = (0/3,3/3) = (0,1)

V(wind= true) =(T(true,No),T(true,Yes)) = (2/2,0/2) = (1,0)

Như vậy, thuộc tính **Wind** có số vector đơn vị nhiều nhất nên sẽ được phân hoạch.

**Kết luận: Kết quả Cây định danh cuối cùng :**



Vậy ta có tập luật:

IF Outlook = Overcast THEN Play = Yes;

IF Outlook = Sunny  AND Humidity= Normal THEN Play = Yes;

IF Outlook = Sunny AND Humidity=High THEN Play = No;

IF Outlook = Rain  AND Wind = False THEN Play = No;

IF Outlook = Rain  AND Wind = True THEN Play = Yes;

Từ tập luật ta có thể suy ra:

Ngày có Overlook là sunny,Temperature là cool, Humidity là high, wind là true thì Play sẽ là: No

Về phần code thì dùng thư viện chefboost của ngôn ngữ python, thư viện này cho phép ta dùng thuật toán của quinland có sẵn trong chỉ cần khai báo file dữ liệu và vài dòng code cơ bản, ở đây em dùng jupyter để code mục đích để hiện cột file dữ liệu như trong video nhằm người xem có thể hình dùng được dữ liệu. Để tăng phần chuẩn xác thì em có dùng phần mềm weka để so sánh kết quả một cách chính xác nhất, kết quả chương trình là các luật nhưng được trình bày theo dạng code logic gồm các thành phần if, else,.. và khi đối chiếu luôn cho kết quả chính xác.

