

# NHẬP MÔN TRÍ TUỆ NHÂN TẠO

---

## CHƯƠNG 2

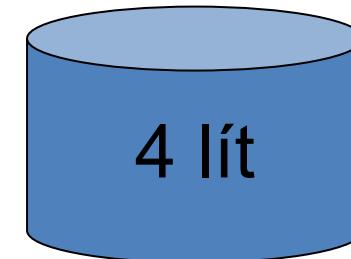
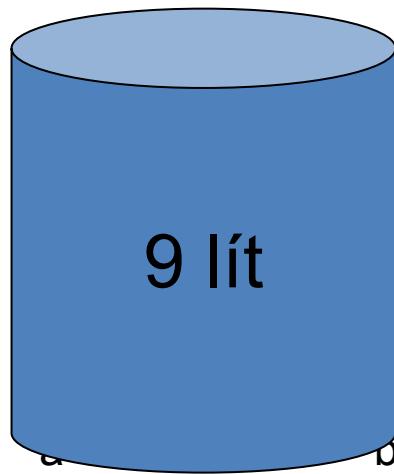
### GIẢI QUYẾT VẤN ĐỀ BẰNG TÌM KIẾM

Trần Nguyễn Minh Thư  
[tnmthu@ctu.edu.vn](mailto:tnmthu@ctu.edu.vn)

# Nội dung

- Biểu diễn bài toán trong KGTT
- Tìm kiếm mù (uninformed search)
- Tìm kiếm heuristic (informed search)
- Cây trò chơi, cắt tỉa alpha –beta
- Bài toán thoả mãn ràng buộc

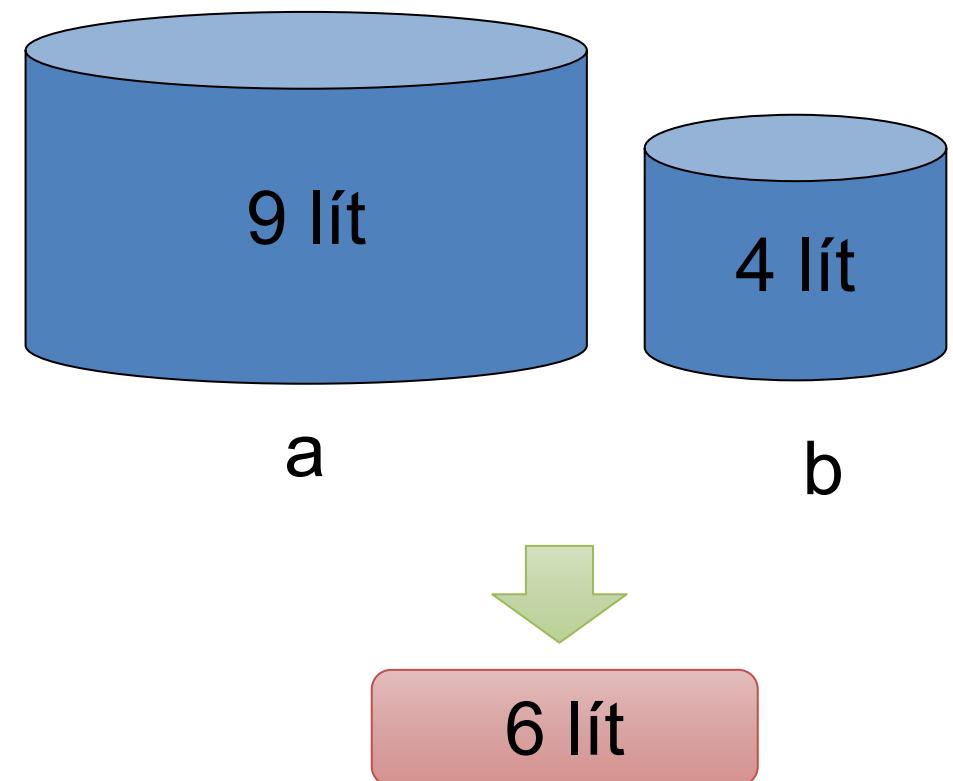
# Ví dụ: Bài toán đong nước



Cho 2 bình 9 lít và 4 lít, không có vạch chia, và 1 vòi bơm.  
Làm cách nào đong **được 6 lít**?

# Ví dụ: Bài toán đong nước

	a	b	
Start	0	0	
1	9	0	
2	5	4	
3	5	0	
4	1	4	
5	1	0	
6	0	1	
7	9	1	
<b>8</b>	<b>6</b>	<b>4</b>	<b>Goal</b>



# Không gian trạng thái (KGTT)

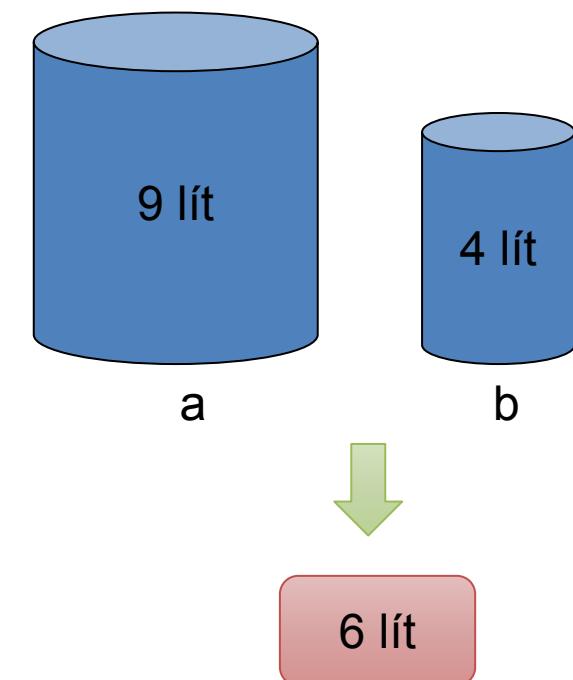
- Một trong những cách để giải quyết các bài toán TTNT là sử dụng Không gian trạng thái (KGTT) để biểu diễn tri thức
- Biểu diễn KGTT là gì?
  - *KGTT bao gồm các trạng thái, kết nối với nhau bằng các hành động*
  - *Từ một trạng thái ban đầu, một hành động có thể chuyển trạng thái đó sang một trạng thái khác*

# Không gian trạng thái (KGTT)

- **Không gian trạng thái** (State space) được mô tả bởi 1 bộ bốn thông tin (**N,S,GD,Op**)
  - **N (node)** : tập các trạng thái
  - **S (start)**: tập không rỗng các trạng thái ban đầu
  - **GD (Goal Description)** - bộ mô tả mục tiêu
    - Tập không rỗng các trạng thái đích
  - **Op (Operator)** tập các toán tử biến đổi trạng thái

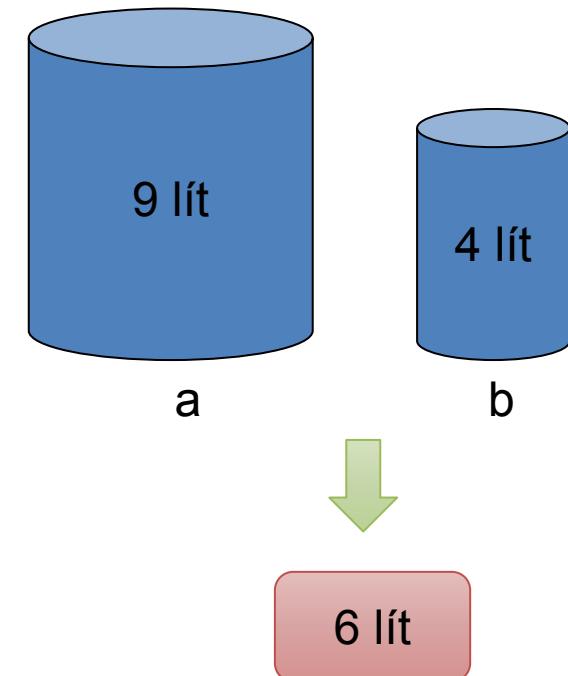
# Ví dụ: Bài toán đong nước

- Xác định các thành phần của bài toán:
  - *Trạng thái đầu:  $(0, 0)$*
  - *Trạng thái đích:  $(6, x)$*
  - *Các thao tác : đong đầy bình (từ vòi, hoặc từ bình còn lại), làm rỗng bình*
  - *Chi phí*
  - *Tìm giải pháp?*



# Ví dụ: Bài toán đong nước

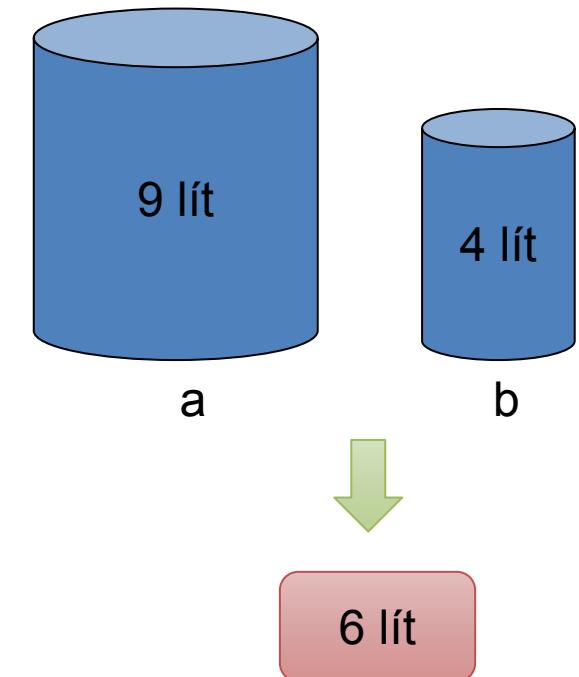
- Xác định các thành phần của bài toán:
  - *Trạng thái đầu:*  $(0, 0)$
  - *Trạng thái đích:*  $(6, x)$
  - Các *thao tác*: *đong đầy bình (từ vòi, hoặc từ bình còn lại), làm rỗng bình*
  - *Chi phí:* 1 cho mỗi thao tác
  - *Tìm giải pháp?*



# Ví dụ: Bài toán đong nước

## ■ Xác định các thành phần của bài toán:

- *Trạng thái đầu:  $(0, 0)$*
- *Trạng thái đích:  $(6, x)$*
- Các thao tác: *đong đầy bình (từ vòi, hoặc từ bình còn lại), làm rỗng bình*
- *Chi phí: 1 cho mỗi thao tác*
- *Tìm giải pháp: tìm dãy các thao tác chuyển từ trạng thái đầu đến trạng thái đích*



# AI – Giải quyết vấn đề bằng tìm kiếm

- Mục tiêu tìm chuỗi các hành động, một số bài toán sử dụng phương pháp tìm kiếm
  - Puzzles – xếp ô số
  - Games – trò chơi
  - Navigation – dẫn đường
  - Motion planning – kế hoạch chuyển động
  - Scheduling – lập lịch biểu
  - Routing – tìm đường

# AI – Giải quyết vấn đề bằng tìm kiếm

**Search Example: River Crossing Problem**

Goal: All on right side of river

Rules:

- 1) Farmer must row the boat
- 2) Only room for one other
- 3) Without the farmer present:
  - Dog bites sheep
  - Sheep eats cabbage

Actions: F>, F<, FC>, FC<, FD>, FD<, FS>, FS<

**Search Example: 8-Puzzle**

Start State

Goal State

Actions: move tiles (e.g., Move2Down)

Goal: reach a certain configuration

**Search Example: Water Jugs Problem**

Given 4-liter and 3-liter pitchers, how do you get exactly 2 liters into the 4-liter pitcher?

4

3

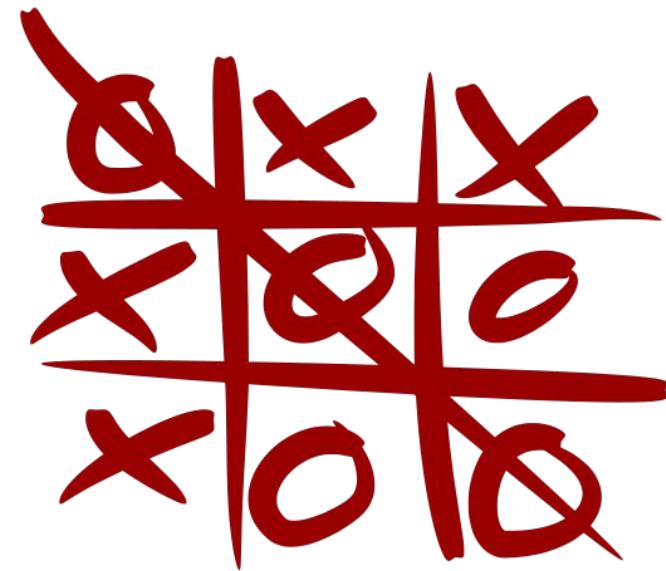
**Search Example: Robot Motion Planning**

Actions: translate and rotate joints

Goal: fastest? most energy efficient? safest?

# Biểu diễn bài toán trên KGTT

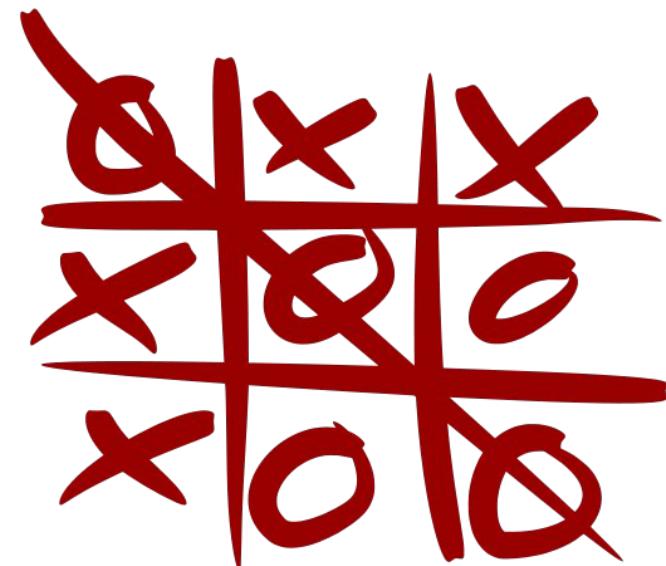
- Trò chơi Tic-tac-toe:
- Xác định các thành phần của bài toán:
  - *Trạng thái của bài toán?*
  - *Trạng thái đầu*
  - *Trạng thái đích*
  - *Các thao tác*



# Ví dụ: Trò chơi Tic-tac-toe

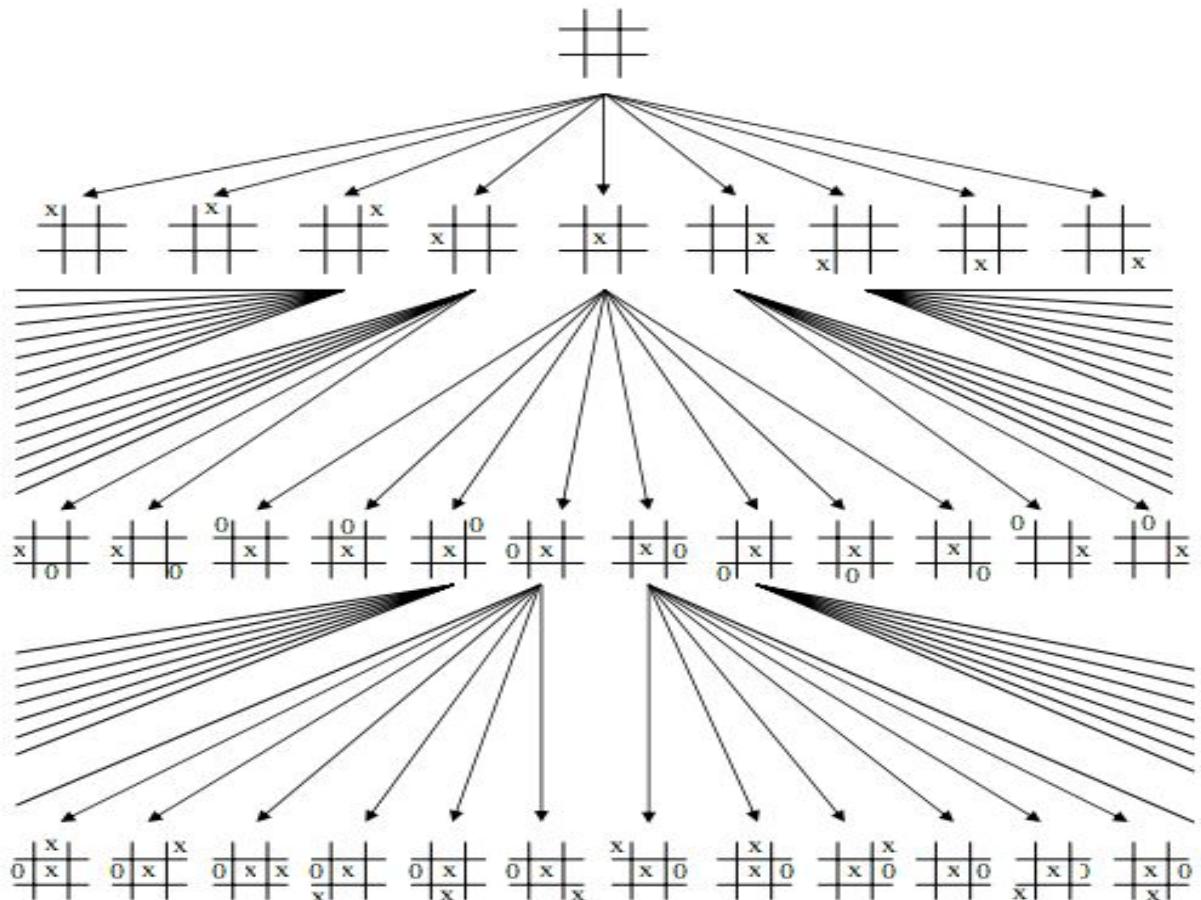
## ■ Phân tích:

- Các trạng thái: mỗi lượt đi sẽ tạo nên 1 trạng thái mới
- Các hành động: đi X hoặc O
- Mục tiêu: dãy 3 ký hiệu giống nhau (ngang, dọc hoặc chéo)
- Chi phí: 1 (mỗi lượt đi)



# Biểu diễn bài toán trên KGTT

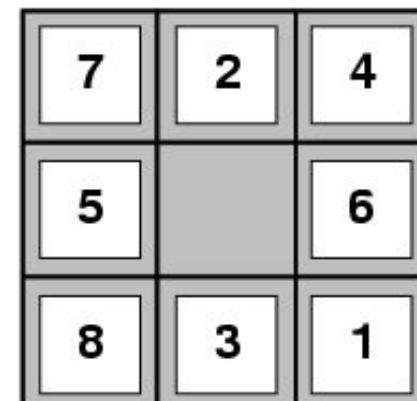
- Trò chơi Tic-tac-toe:



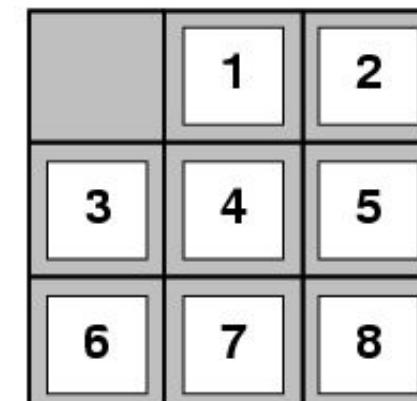
# Ví dụ: Trò chơi 8 ô số (8-Puzzle)

## ■ Cần xác định:

- Các trạng thái?
- Các hành động?
- Mục tiêu?
- Chi phí đường đi?



Start State



Goal State

# Ví dụ: Trò chơi 8 ô số (8-Puzzle)

## ■ Phân tích:

- Các trạng thái: vị trí khác nhau của các ô số
- Các hành động: di chuyển ô trống sang trái phải, lên, xuống
- Mục tiêu: Goal state
- Chi phí đường đi: 1 (mỗi lần di chuyển)

7	2	4
5		6
8	3	1

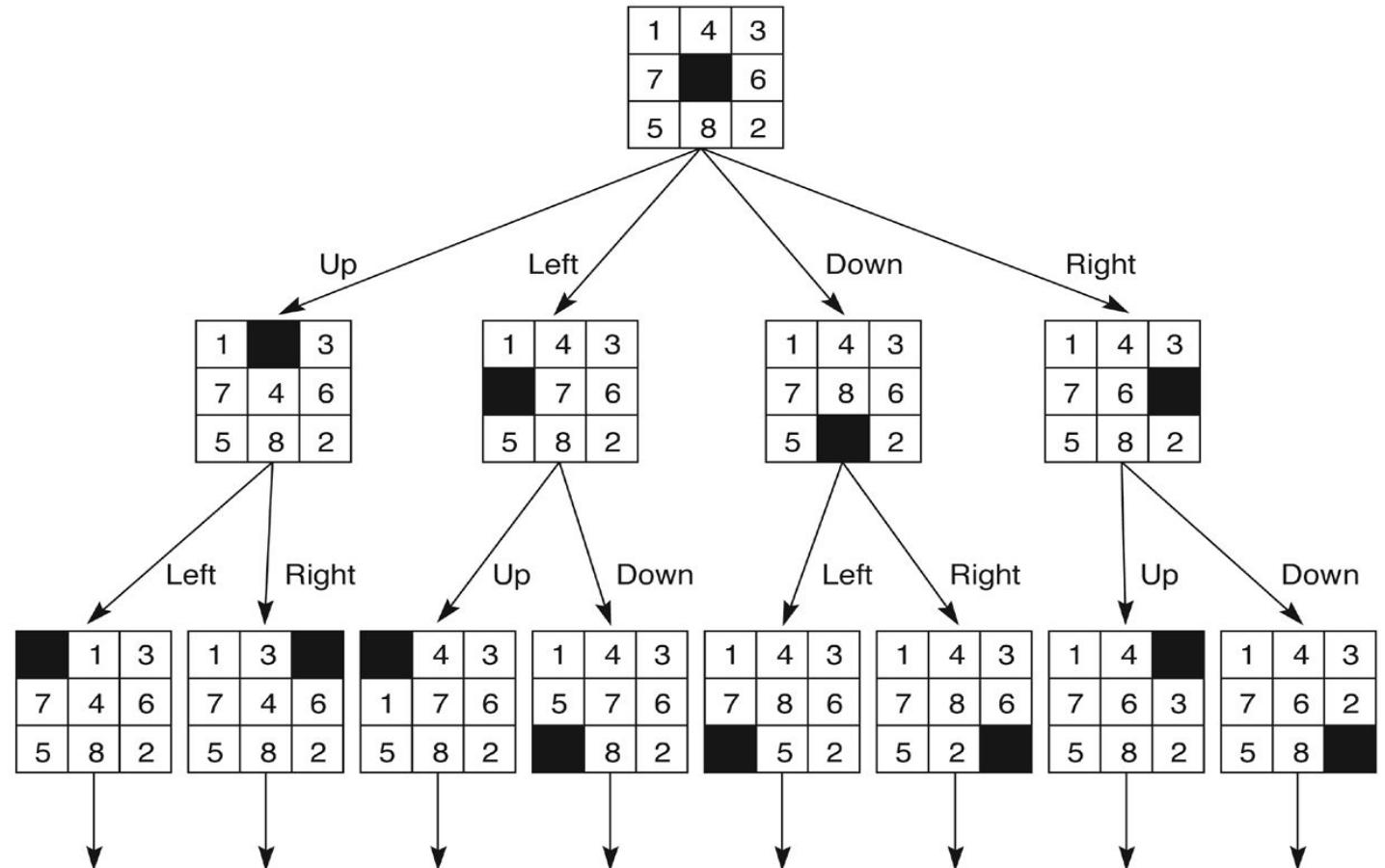
Start State

	1	2
3	4	5
6	7	8

Goal State

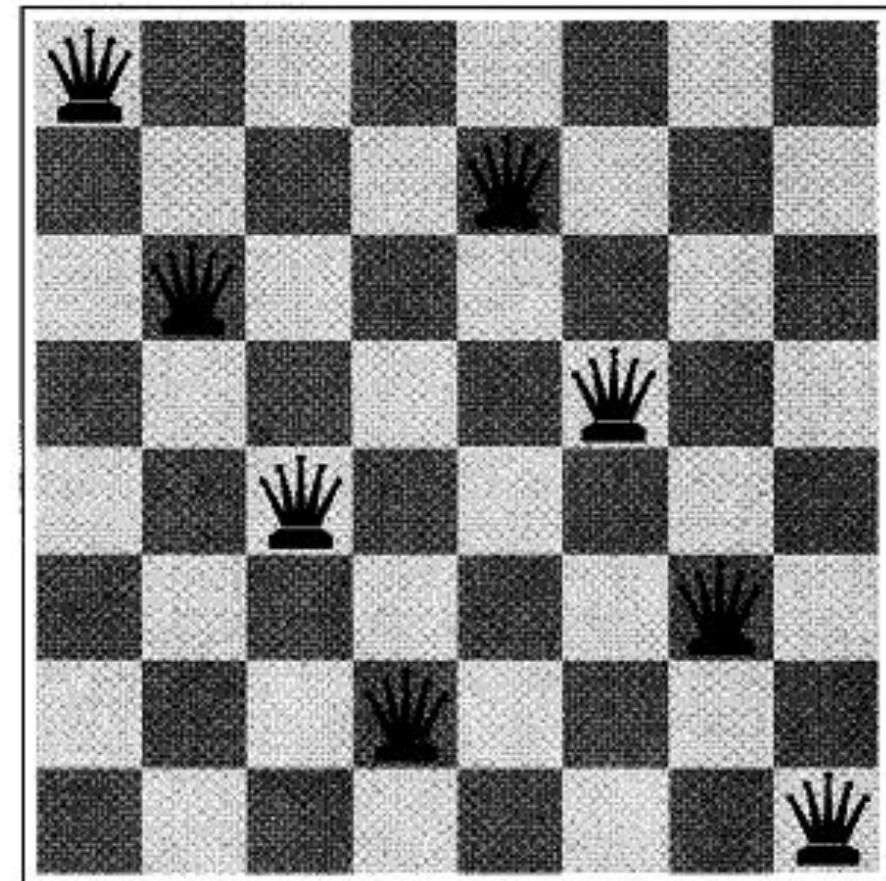
# Biểu diễn bài toán trên KGTT

## ■ Trò chơi 8-Puzzle



# Ví dụ: Bài toán 8 quân hậu

- Đặt 8 quân hậu lên bàn cờ vua sao cho không có quân hậu nào bị khống chế.
- Xác định trạng thái, phép toán biến đổi trạng thái



# Ví dụ: Bài toán 8 quân hậu

## ■ Phương pháp 1:

- Trạng thái:
  - *Bàn cờ và vị trí của các quân hậu (có thể có từ 0 đến 8 quân hậu)*
- Phép toán:
  - *Đặt thêm một quân hậu vào một vị trí nào đó trên bàn cờ*
- Hàm kiểm tra mục tiêu:
  - *Bàn cờ đủ 8 quân hậu và không có quân nào bị khống chế không*
- $C_{64}^8 = 64 * 63 * 62 * 61 * 60 * 59 * 58 * 57 = 1.8 * 10^{14}$  cách đặt các quân hậu

# Ví dụ: Bài toán 8 quân hậu

## Phương pháp 2:

- Trạng thái:
  - *Bàn cờ và vị trí của các quân hậu trên bàn cờ (có thể có từ 0 đến 8 quân hậu trên bàn cờ), không có quân nào bị khống chế.*
- Phép toán:
  - *Đặt thêm một quân hậu vào một vị trí nào đó trên cột trống đầu tiên của bàn cờ sao cho quân hậu vừa mới đặt không bị khống chế*
- Hàm kiểm tra mục tiêu:
  - *Có đủ 8 quân hậu chưa*
- Số lượng đường đi sẽ giảm đi rất đáng kể

# Ví dụ: Bài toán 8 quân hậu

## Phương pháp 3

- Trạng thái:
  - *Bàn cờ và vị trí của cả 8 quân hậu, mỗi quân trên một cột.*
- Phép toán:
  - *Di chuyển 1 quân hậu bị khống chế qua một ô khác trong cùng một cột.*
- Hàm kiểm tra mục tiêu:
  - *Có quân hậu nào còn bị khống chế không*

# Giải quyết bài toán bằng tìm kiếm

- Khi bài toán đã được biểu diễn bằng không gian trạng thái => **Tìm kiếm** một đường đi **lời giải**/ một **trạng thái đích mong ước**
- **Đường đi lời giải** (Solution path)
  - Là đường đi trong đồ thị/cây trạng thái dẫn từ trạng thái bắt đầu đến trạng thái thỏa mãn mục tiêu/trạng thái đích
- **Một trạng thái đích mong ước:** là trạng thái cuối cùng thỏa mãn mục tiêu yêu cầu

# Giải pháp cho các bài toán

- Biểu diễn bài toán trên KGTT để thực hiện tìm kiếm
- Các bước xây dựng KGTT:
  - *Bắt đầu từ một trạng thái đã cho (trạng thái ban đầu)*
  - *Kiểm tra xem có phải là trạng thái đích*
  - *Mở rộng 1 trong các trạng thái*
  - *Nếu có nhiều khả năng, chọn 1 khả năng nào đó*
  - *Quy trình: chọn, kiểm tra và mở rộng đến khi tìm ra giải pháp hoặc không thể tiếp tục mở rộng => quy trình xây dựng cây tìm kiếm*
- Tập các trạng thái được mở rộng sẽ có nhiều lựa chọn, việc chọn trạng thái nào để mở rộng được gọi là chiến lược tìm kiếm

# Giải quyết bài toán bằng tìm kiếm

## Chiến lược tìm kiếm:

- Từ một trạng thái ban đầu hay trạng thái bất kỳ chưa phải là trạng thái đích, dựa vào các hành động để sinh ra các trạng thái mới (mở rộng không gian trạng thái)
- Để tìm được lời giải, quá trình mở rộng KGTT được thực hiện cho đến khi tìm được trạng thái đích hay không thể mở rộng được KGTT
- Tập các trạng thái được mở rộng sẽ có nhiều lựa chọn, việc chọn trạng thái nào để mở rộng được gọi là chiến lược tìm kiếm

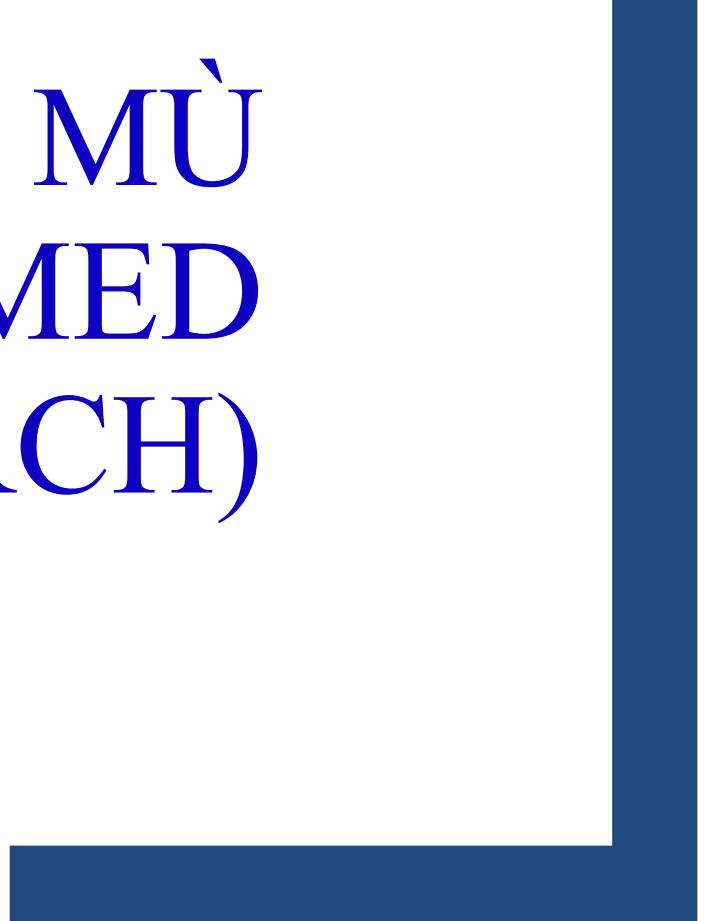
# Các kỹ thuật tìm kiếm

- Tìm kiếm mù (uninformed/blind search):
  - Trạng thái được chọn để phát triển chỉ dựa theo cấu trúc của KGTT mà không dùng thêm thông tin hỗ trợ.
  - Tìm kiếm dựa trên kinh nghiệm (informed/ heuristic search):
    - Dựa vào kinh nghiệm và sự hiểu biết để xây dựng hàm đánh giá hướng dẫn tìm kiếm

# Đánh giá phương pháp tìm kiếm

- *Phương pháp tìm kiếm xác định thứ tự triển khai của các đỉnh trong cây tìm kiếm*
- *Các giải thuật tìm kiếm thường được đánh giá dựa trên 4 tiêu chí sau:*
  - **Tính trọn vẹn:** Liệu nó luôn tìm ra nghiệm không nếu bài toán tồn tại nghiệm.
  - **Độ phức tạp thời gian:** giải thuật có mất nhiều thời gian không?
  - **Độ phức tạp không gian:** yêu cầu bộ nhớ lưu trữ?
  - **Tính tối ưu:** Giải thuật có đảm bảo tìm ra nghiệm với hàm chi phí ít nhất không?

# TÌM KIẾM MÙ (UNINFORMED BLIND SEARCH)



# Tìm kiếm mù

- Tìm kiếm rộng (breadth-first search)
- Tìm kiếm sâu (depth-first search)
- Tìm kiếm theo độ sâu có giới hạn (depth-limited search)
- Tìm kiếm theo chiều sâu lặp (iterative deepening depth-first search)
- Tìm kiếm giá thành đồng nhất

# Tìm kiếm rộng (breath-first search - BFS)

Procedure breadth-first-search;

# Begin % khởi đầu

**Open:= [start];**      **Closed:= [ ];**

*While open ≠ [] do % còn các trạng thái chưa khảo sát*

# *Begin*

Loại bỏ trạng thái ngoài cùng bên trái khỏi open, gọi nó là X;

If X là một đích then trả lời kết quả (thành công) % tìm thấy đích

# else begin

## Phát sinh các con của X;

Đưa X vào closed;

Loại các con của X đã tồn tại trong open hoặc closed; % kiểm tra vòng lặp

Đưa các con còn lại của X vào **đầu bên phải** của open; % hàng đợi

end;

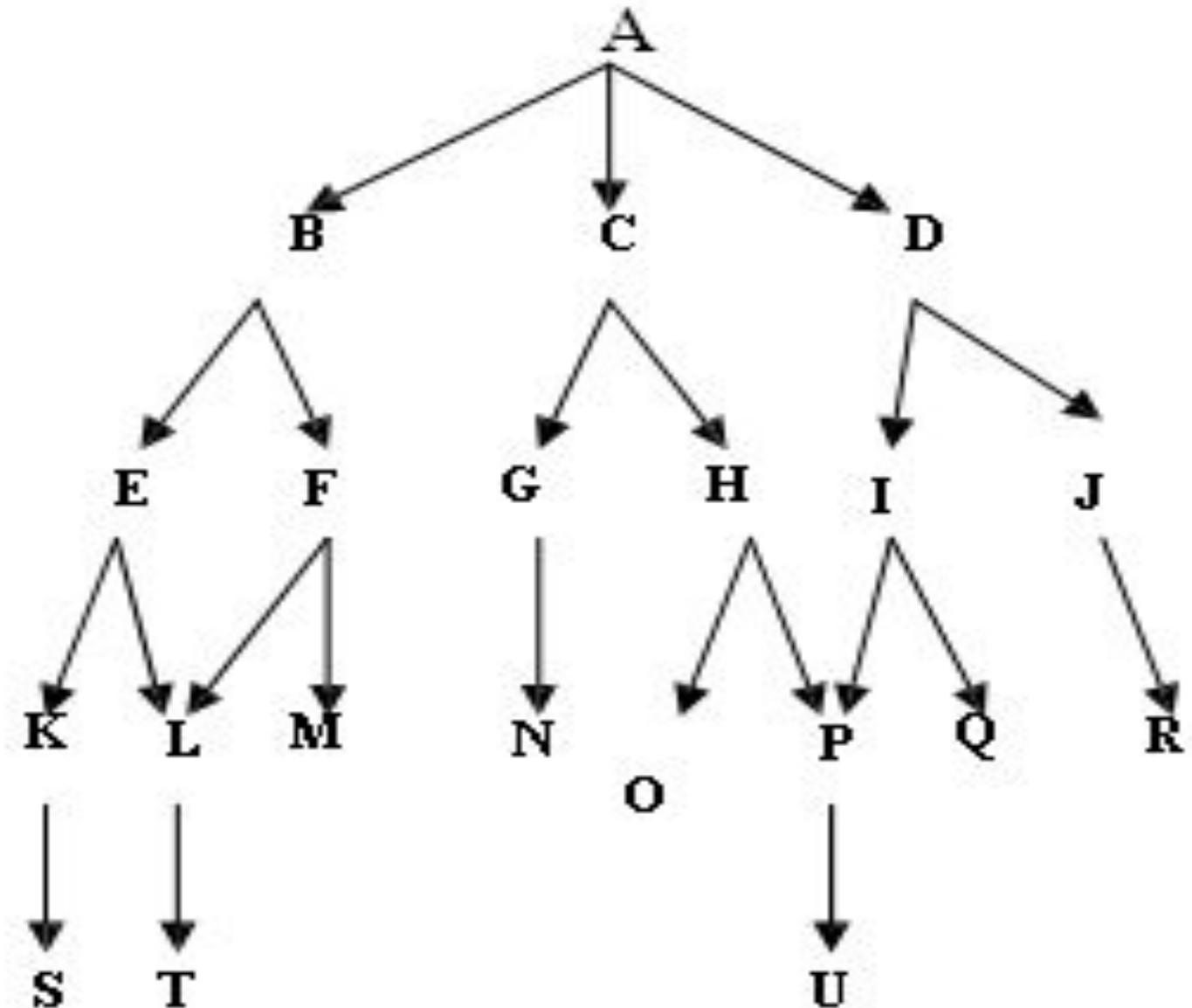
*End;*

*Trả lời kết quả (thất bại); % không còn trạng thái nào*

**End;**

# Tìm kiếm mù

- Xét KGTT sau:



# Tìm kiếm rộng (breadth-first search - BFS)

- Tìm kiếm rộng:
- Các bước thực hiện tìm kiếm rộng:
  1.  $Open = [A]; closed = []$
  2.  $Open = [B,C,D]; closed = [A]$
  - 3.

Procedure breadth-first-search;

Begin % khởi đầu

Open:= [start];

Closed:= [ ];

While open  $\neq []$  do % còn các trạng thái chưa khảo sát

Begin

Loai bỏ trạng thái ngoài cùng bên trái khỏi open, gọi nó là X;

If X là một đích then trả lời kết quả (thành công) % tìm thấy đích

else begin

Phát sinh các con của X;

Đưa X vào closed;

Loai các con của X đã tồn tại trong open hoặc closed; % kiểm tra vòng lặp

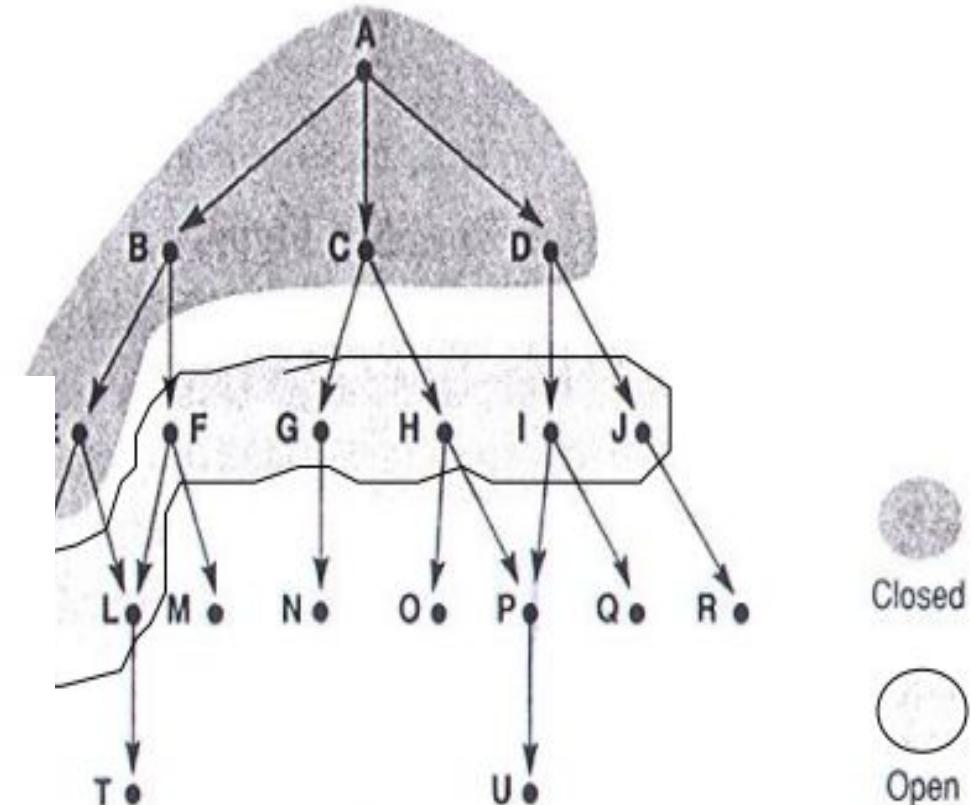
Đưa các con còn lại của X vào đầu bên phải của open; % hàng đợi

end;

End;

Trả lời kết quả (thất bại); % không còn trạng thái nào

End;



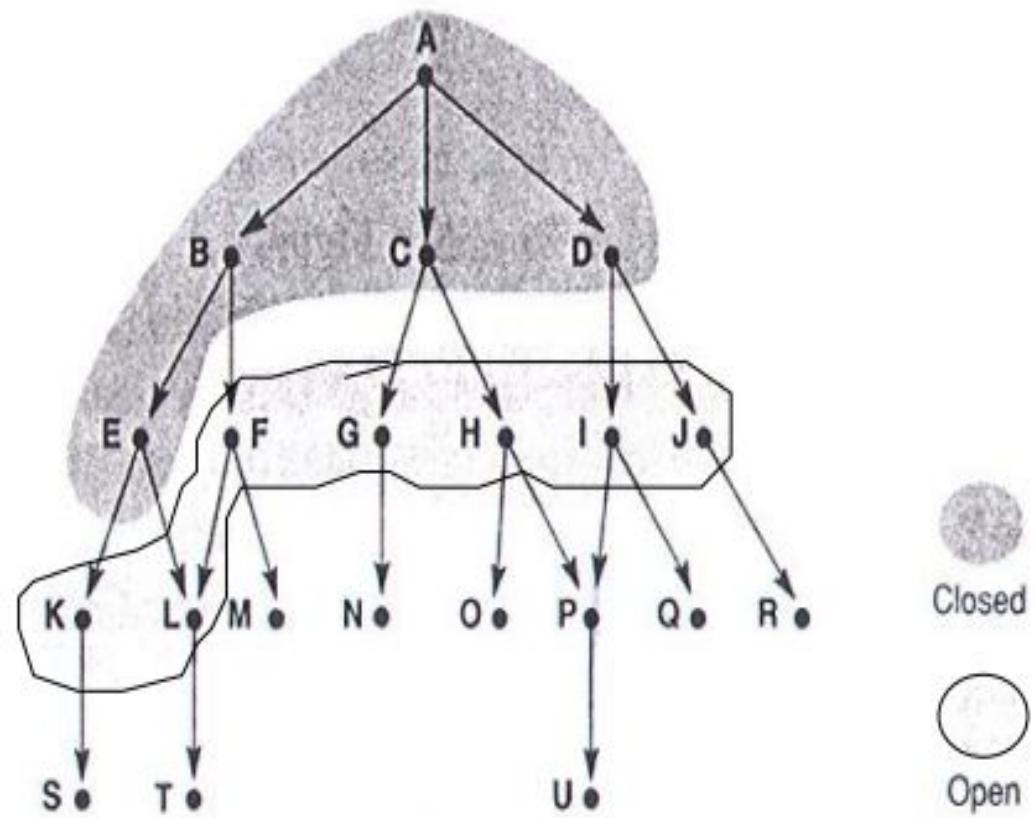
Closed



Open

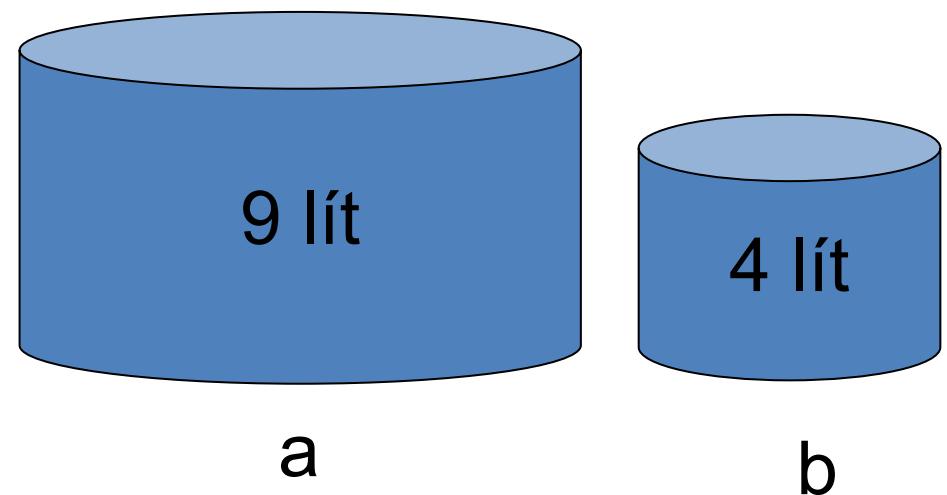
# Tìm kiếm rộng (breath-first search - BFS)

- Tìm kiếm rộng:
- Các bước thực hiện tìm kiếm rộng:
  1.  $\text{Open} = [A]; \text{closed} = []$
  2.  $\text{Open} = [B,C,D]; \text{closed} = [A]$
  3.  $\text{Open} = [C,D,E,F]; \text{closed} = [B,A]$
  4.  $\text{Open} = [D,E,F,G,H];$   
 $\text{closed} = [C,B,A]$
  5.  $\text{Open} = [E,F,G,H,I,J];$   
 $\text{closed} = [D,C,B,A]$
  6.  $\text{Open} = [F,G,H,I,J,K,L];$   
 $\text{closed} = [E,D,C,B,A]$
  7.  $\text{Open} = [G,H,I,J,K,L,M];$   
(vì  $L$  đã có trong open);  
 $\text{closed} = [F,E,D,C,B,A]$
  8. ...



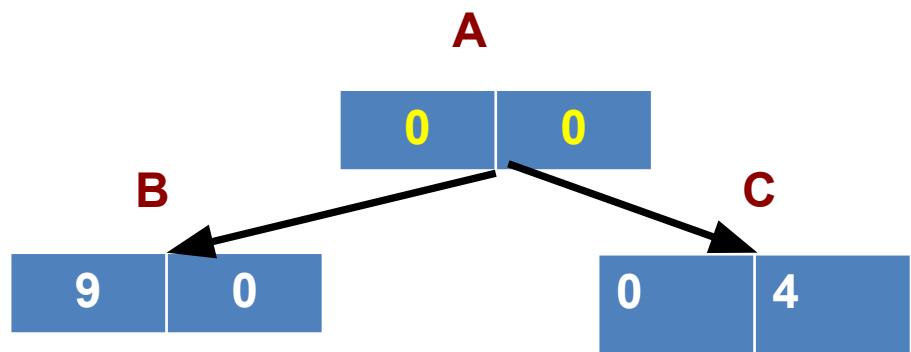
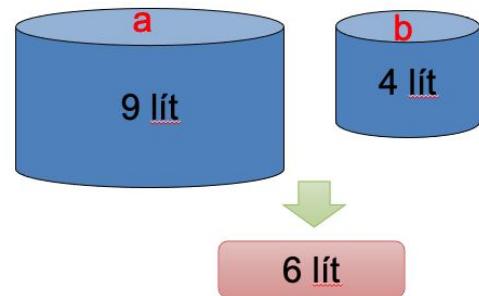
# Ví dụ: Bài toán đong nước

	a	b
<b>Start</b>	0	0
1		



6 lít

# Ví dụ: Bài toán đong nước



```
Procedure breadth-first-search;
Begin % khởi đầu
    Open:= [start];
    Closed:= [ ];
    While open ≠ [ ] do % còn các trạng thái chưa khảo sát
        Begin
            Loai bỏ trạng thái ngoài cùng bên trái khỏi open, gọi nó là X;
            If X là một đích then trả lời kết quả (thành công) % tìm thấy đích
        else begin
            Phát sinh các con của X;
            Đưa X vào closed;
            Loai các con của X đã tồn tại trong open hoặc closed; % kiểm tra vòng lặp
            Đưa các con còn lại của X vào đầu bên phải của open; % hàng đợi
        end;
        End;
        Trả lời kết quả (thất bại); % không còn trạng thái nào
    End;
```

30

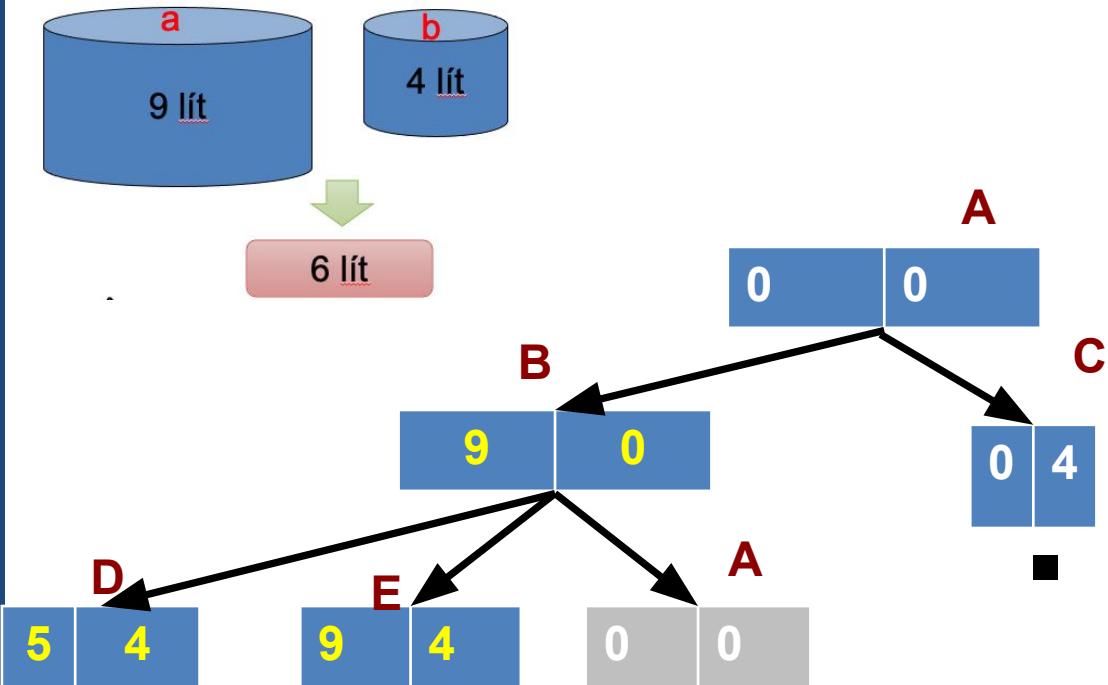
## ■ Các bước thực hiện tìm kiếm rộng:

**Bước 1:** Open = [A: 0-0]; closed = []

- Xét A, A ko phải trạng thái đích
- Đưa A vào closed: closed=[A:0-0]
- Xét các con của A là B:9-0 và C:0-4
- Các con của A ko tồn tại trong open-closed
- Đưa các con của A và bên phải open open [B: 9-0,C: 0-4];

34

# Ví dụ: Bài toán đong nước



Procedure breadth-first-search;

```

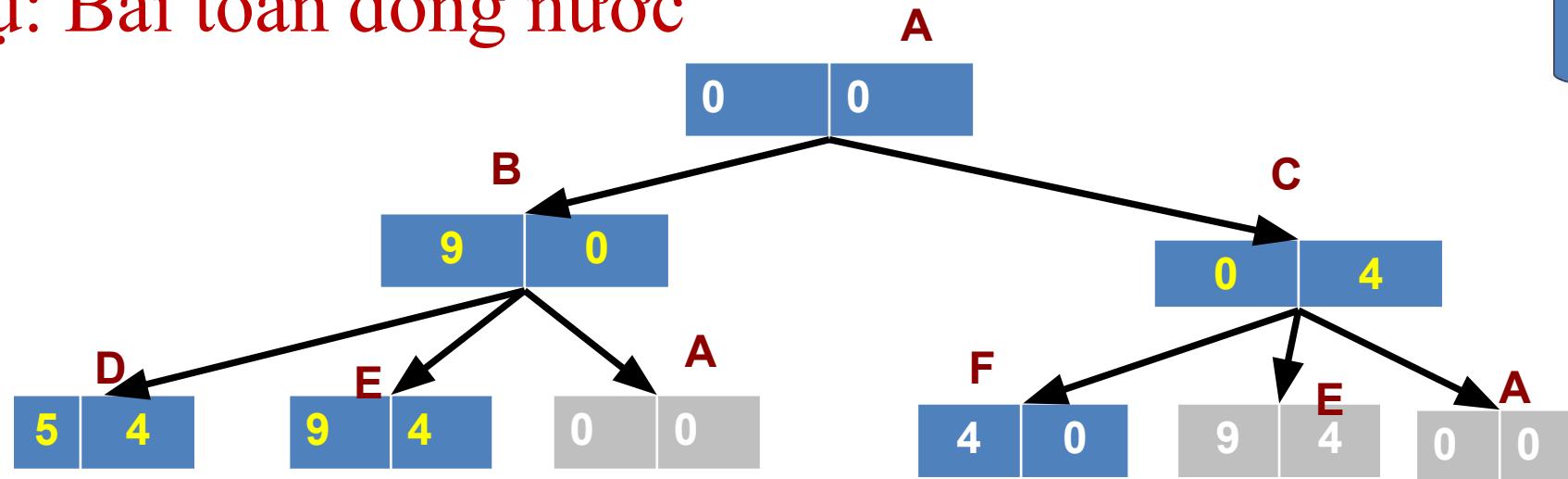
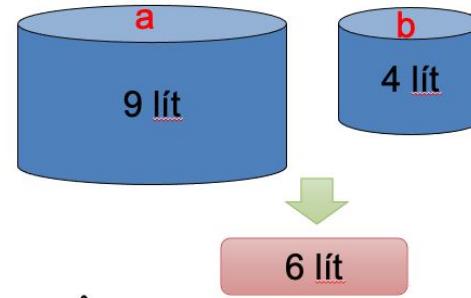
Begin % khởi đầu
  Open:= [start];
  Closed:= [ ];
  While open ≠ [] do % còn các trạng thái chưa khảo sát
    Begin
      Loai bỏ trạng thái ngoài cùng bên trái khỏi open, gọi nó là X;
      If X là một đích then trả lời kết quả (thành công) % tìm thấy đích
    else begin
      Phát sinh các con của X;
      Đưa X vào closed;
      Loai các con của X đã tồn tại trong open hoặc closed; % kiểm tra vòng lặp
      Đưa các con còn lại của X vào đầu bên phải của open; % hàng đợi
    end;
    End;
    Trả lời kết quả (thất bại); % không còn trạng thái nào
End;
  
```

■ Các bước thực hiện tìm kiếm rộng:

**Bước 2:** Open = [B: 9-0, C: 0-4]; closed = [A: 0-0]

- Xét B:9-0, B không là trạng thái đích
  - Đưa B vào Closed: closed[A,B]
  - Xét các con của B: E:9-4, D:5-4, A:0-0
  - Loại các con đã tồn tại trong open và closed – loại A:0-0
  - Thêm các con còn lại vào open
- Open = [C:0-4, D:5-4, E:9-4,]; closed = [A,B]

## Ví dụ: Bài toán đong nước



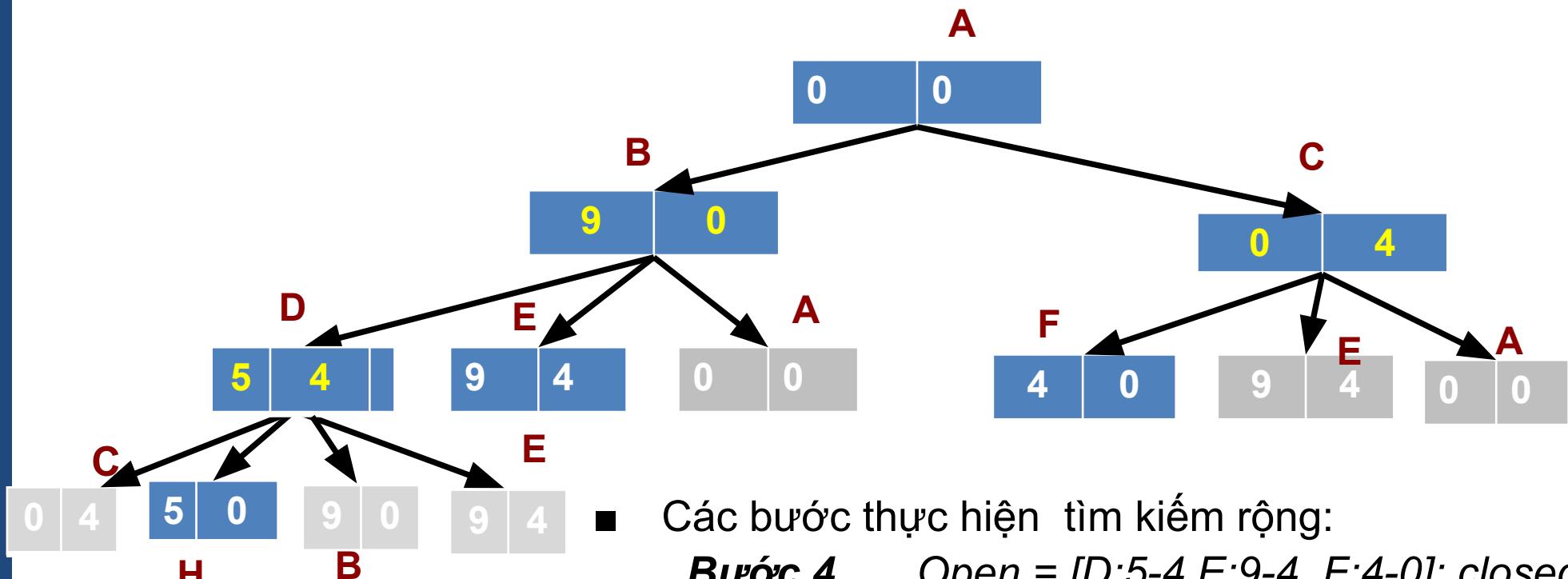
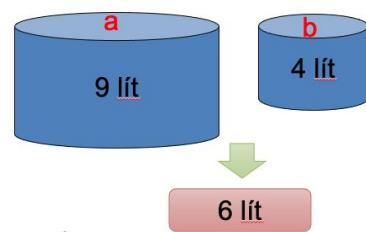
- Các bước thực hiện tìm kiếm rộng:

**Bước 3.** Open = [C:0-4, D:5-4, E:9-4,]; closed = [B,A]

- Xét C, ko phải trạng thái đích
- Đưa C vào Closed
- Các con của C: F: 4-0, E:9-4, A:0-0
- Loại các con đã tồn tại trong open và closed – loại A:0-0 và E:9-4
- Thêm các con còn lại vào open

Open = [D:5-4, E:9-4, F:4-0]; closed = [A,B,C]

# Ví dụ: Bài toán đong nước



■ Các bước thực hiện tìm kiếm rộng:

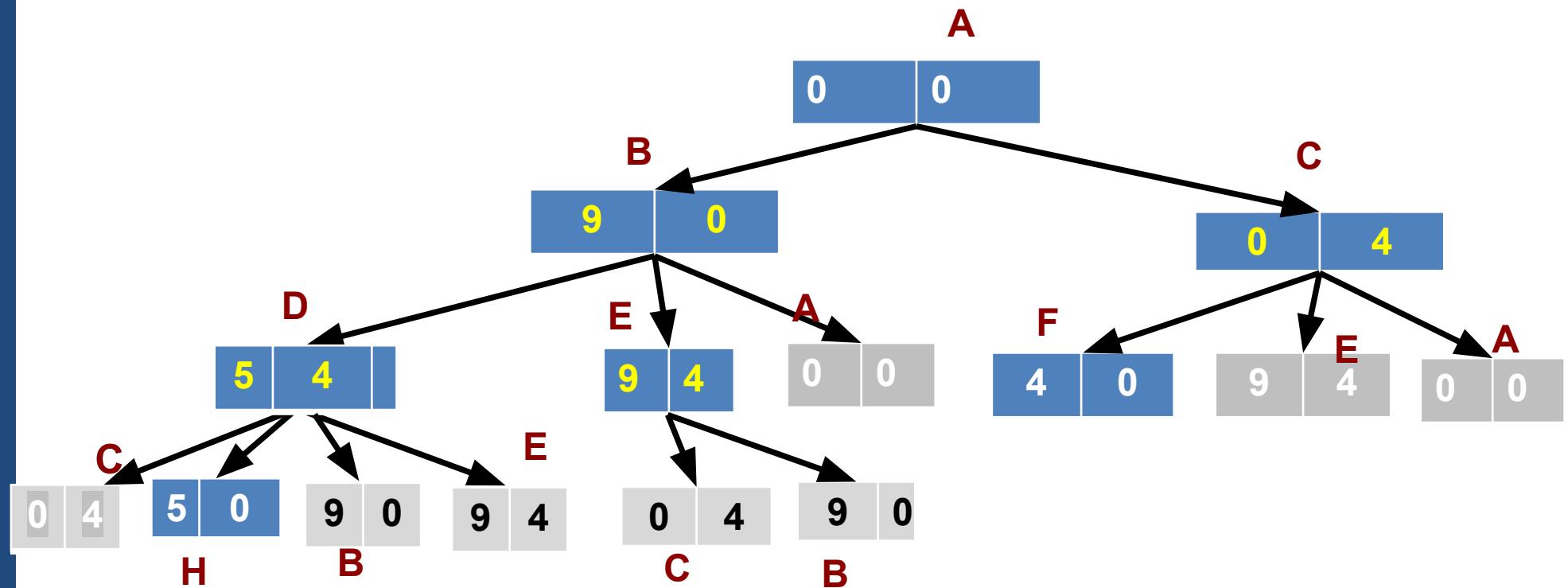
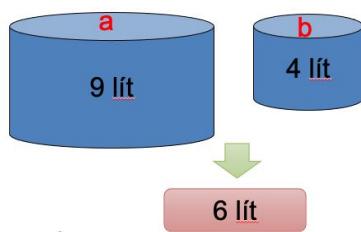
**Bước 4.** Open = [D:5-4, E:9-4, F:4-0]; closed = [A,B,C]

Xét D, ko phải trạng thái đích

- Đưa D vào Closed: closed = [A,B,C,D]
- Các con của D: C: 0-4, H:5-0, B:9-0, E:9-4
- Loại các con đã tồn tại trong open và closed –
- loại C: 0-4, B:9-0, E:9-4
- Thêm các con còn lại vào open

Open = [E:9-4, F:4-0 , H:5-0, ]; closed = [A,B,C,D]

# Ví dụ: Bài toán đong nước

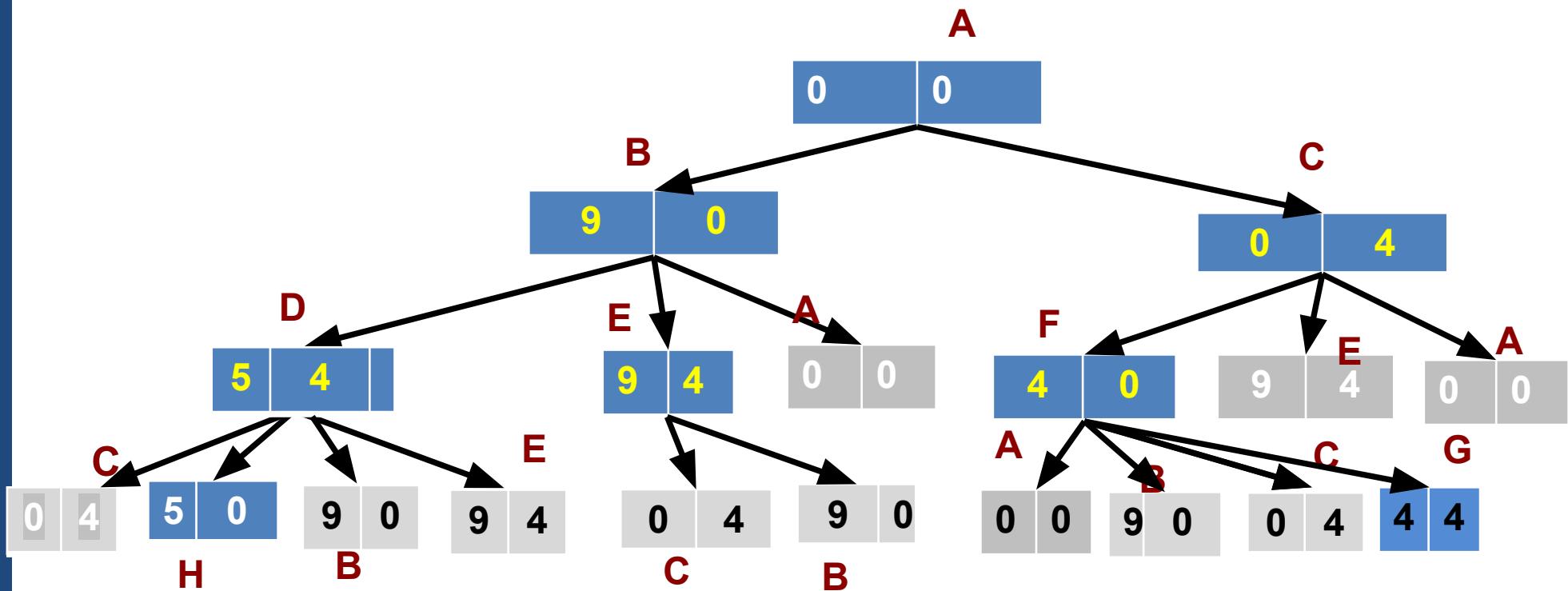
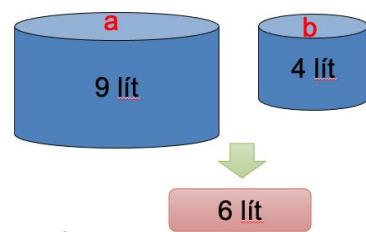


**Bước 5:** Open = [E:9-4, F:4-0 , H:5-0, ]; closed = [A,B,C,D]

Xét E, ko phải trạng thái đích

- Đưa E vào Closed: closed = [A,B,C,D,E]
- Các con của E: C: 0-4, B:9-0
- Loại các con đã tồn tại trong open và closed – loại : C: 0-4, B:9-0
- Open = [F:4-0, H:5-0, ]; closed = [A,B,C,D,E]

# Ví dụ: Bài toán đong nước

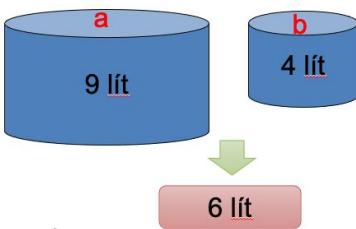
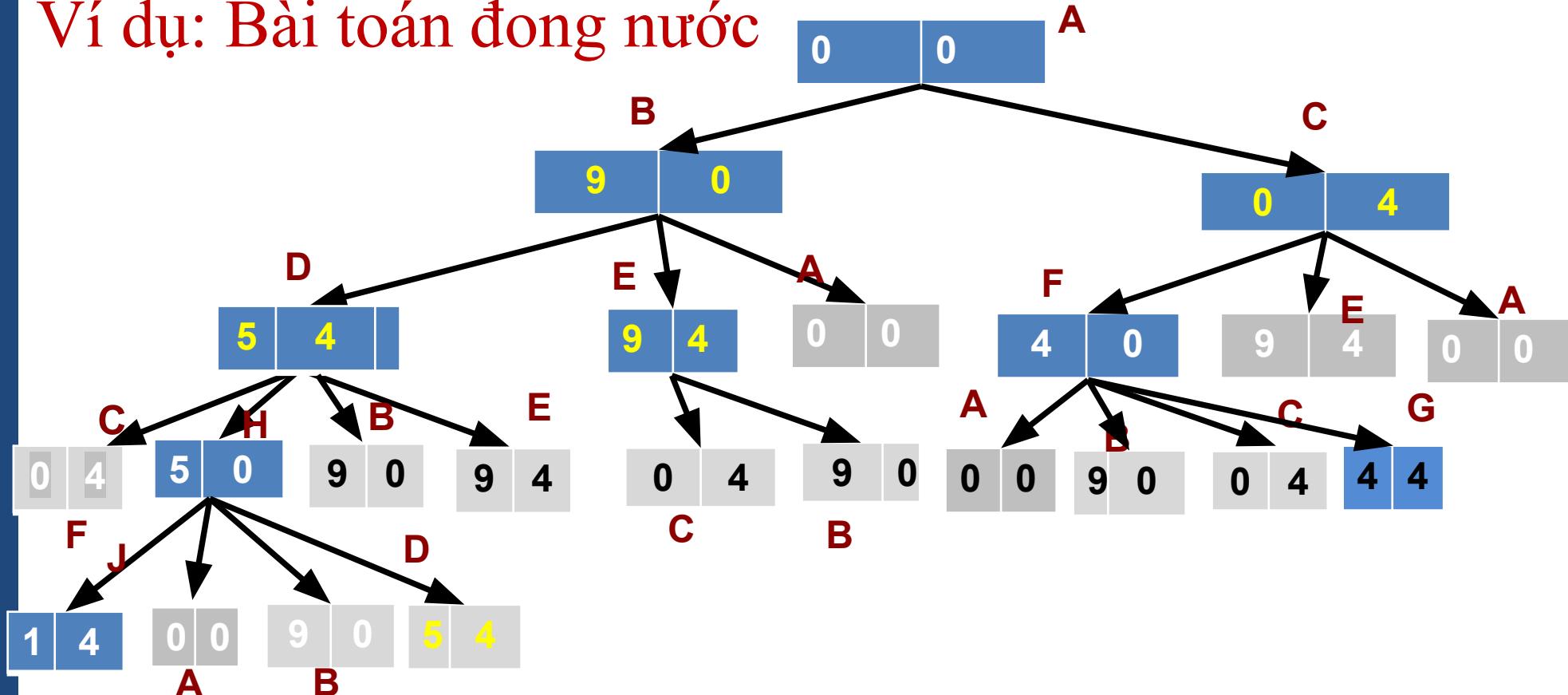


**Bước 6:**  $[F:4-0, H:5-0, ]$ ; closed =  $[A,B,C,D,E]$

Xét F, ko phải trạng thái đích

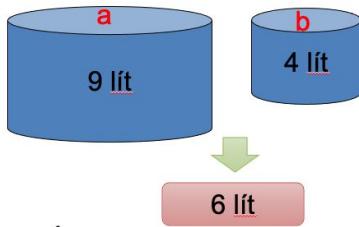
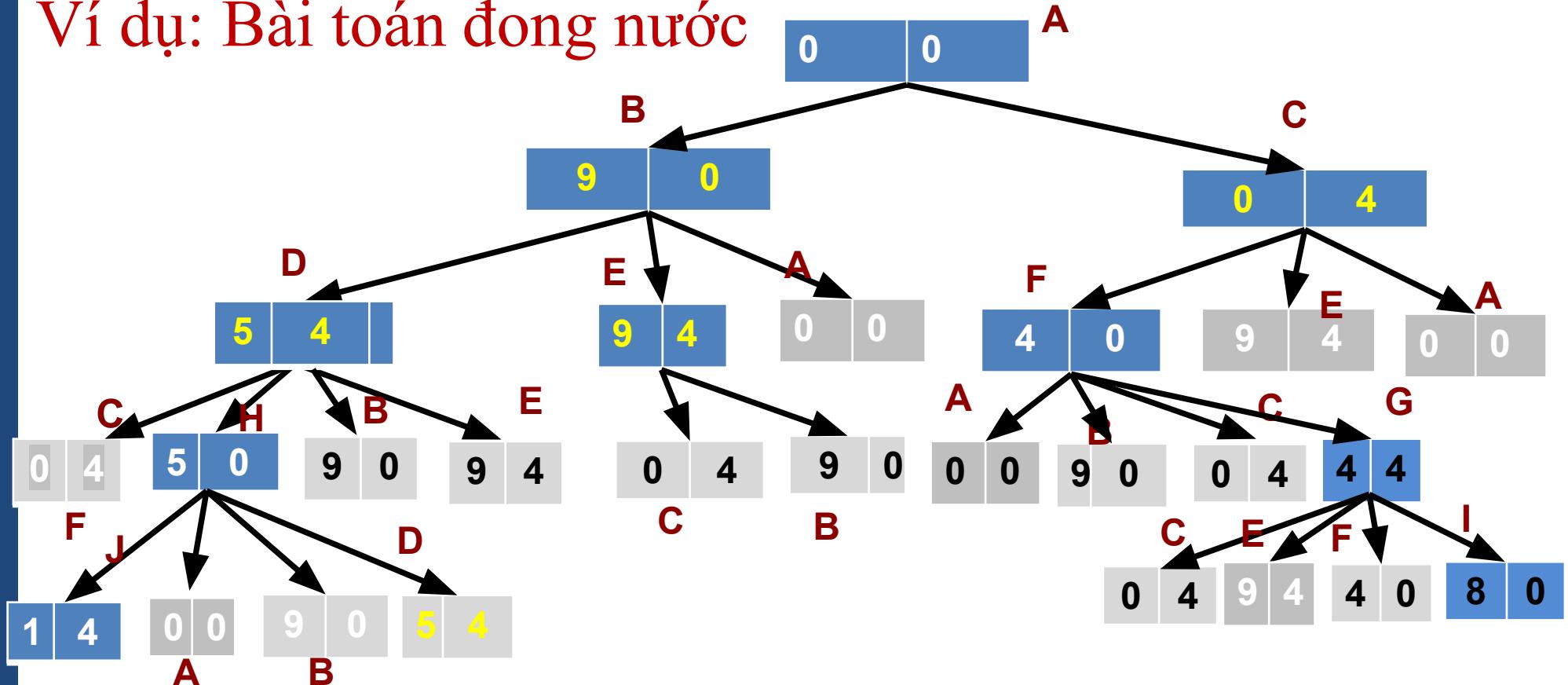
- Đưa F vào Closed: closed =  $[A,B,C,D,E,F]$
- Các con của F: A:0-0, B:9-0, C: 0-4, G:4-4
- Loại các con đã tồn tại trong open và closed – loại : A:0-0, B:9-0, C: 0-4,
- Open =  $[H:5-0, G:4,4 ]$ ; closed =  $[A,B,C,D,E,F]$

# Ví dụ: Bài toán đong nước



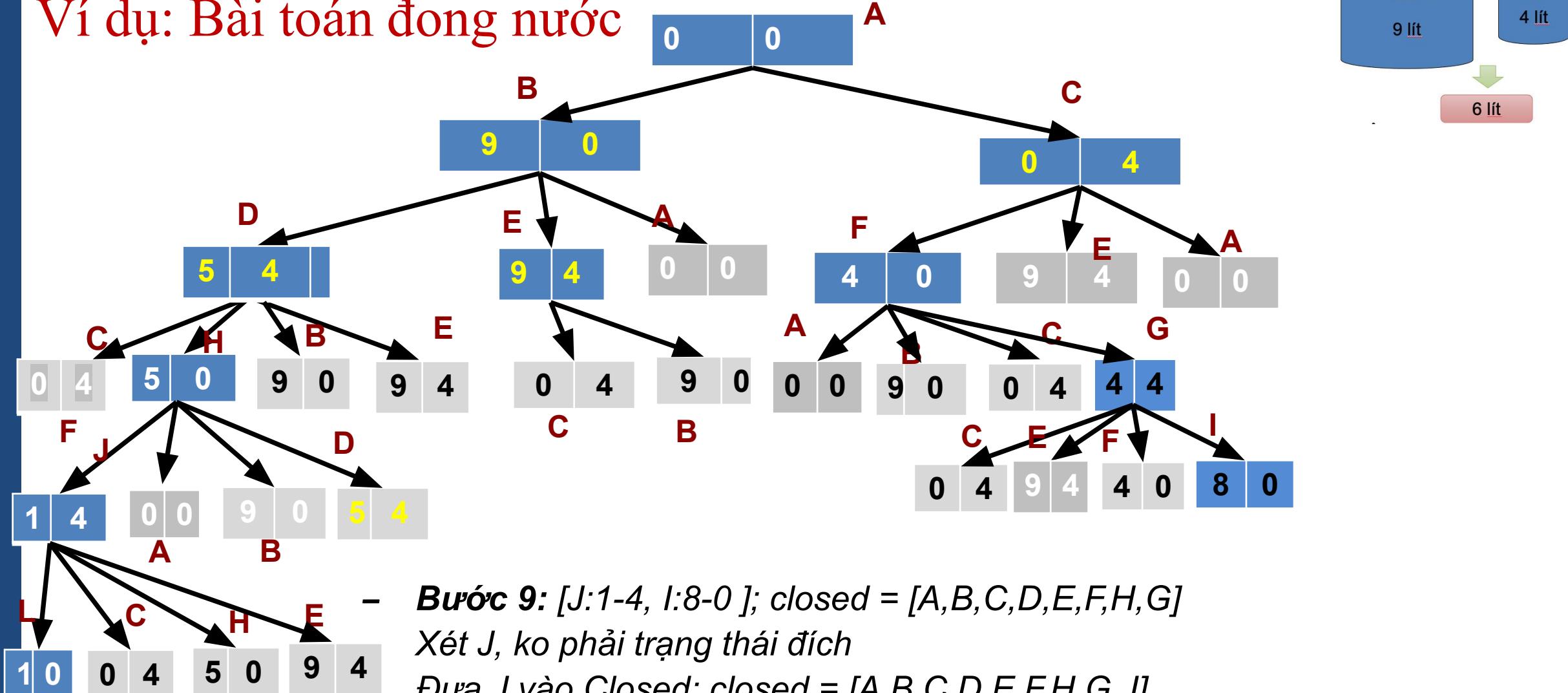
- **Bước 7:** Open = [H:5-0, G:4,4 ]; closed = [A,B,C,D,E,F]
  - Xét H, ko phải trạng thái đích
- Đưa H vào Closed: closed = [A,B,C,D,E,F,H]
- Các con của H: A: 0-0, B:9-0, D:5-4, J:1-4
- Loại các con đã tồn tại trong open và closed – loại : A: 0-0, B:9-0, D:5-4,
- Thêm J:1-4 vào Open = [G:4-4,J:1-4 ]; closed = [A,B,C,D,E,F,H]

# Ví dụ: Bài toán đong nước



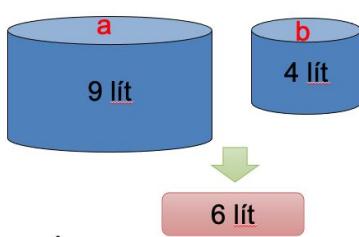
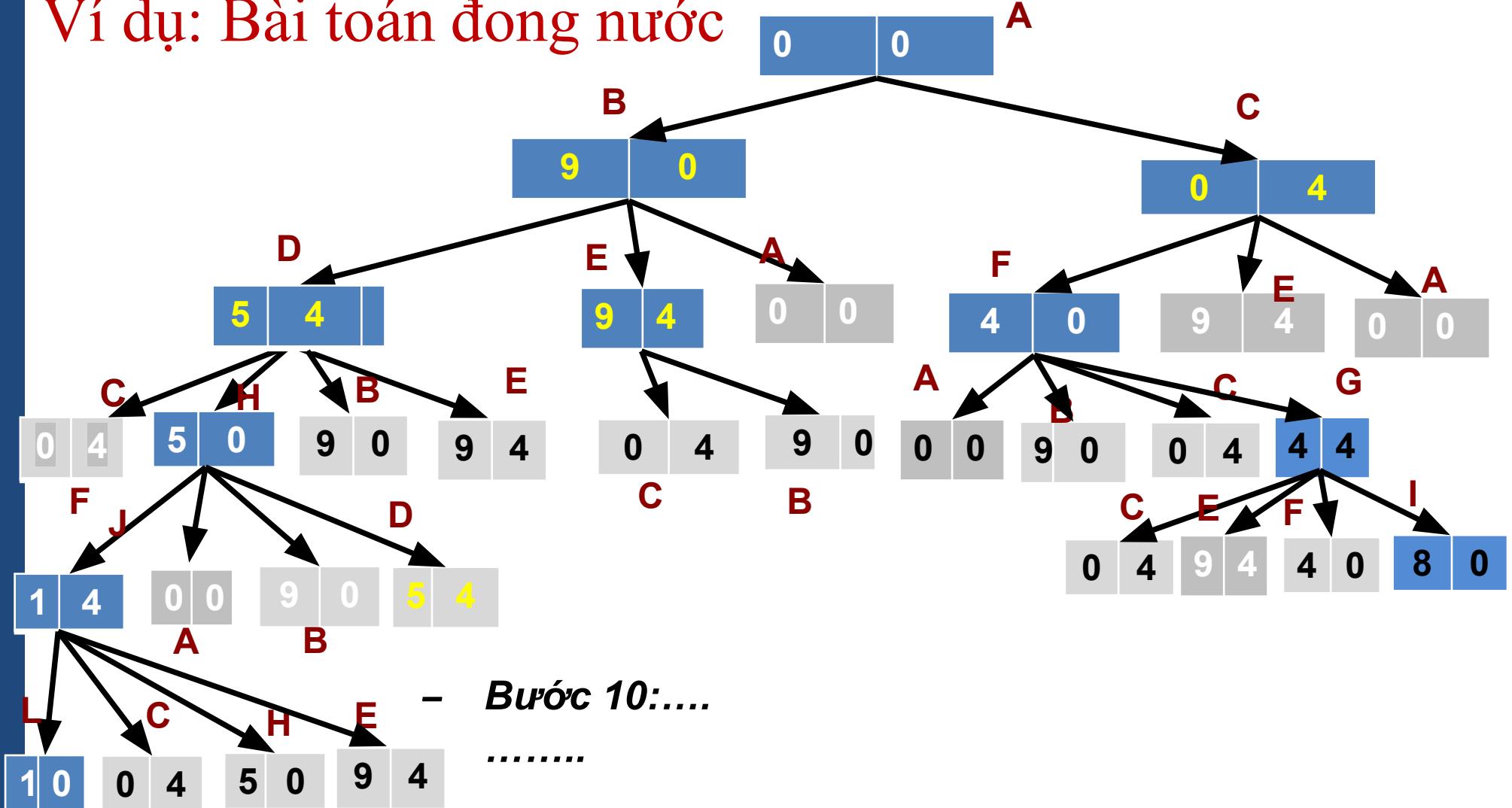
- **Bước 8:** [G:4-4,J:1-4]; closed = [A,B,C,D,E,F,H]
- Xét G, ko phải trạng thái đích
- Đưa G vào Closed: closed = [A,B,C,D,E,F,H,G]
- Các con của G: C: 0-4, E:9-4, F:4-0, I:8-0
- Loại các con đã tồn tại trong open và closed – loại : C: 0-4, E:9-4, F:4-0
- Thêm I:8-0 vào Open = [J:1-4, I:8-0]; closed = [A,B,C,D,E,F,H,G]

# Ví dụ: Bài toán đong nước

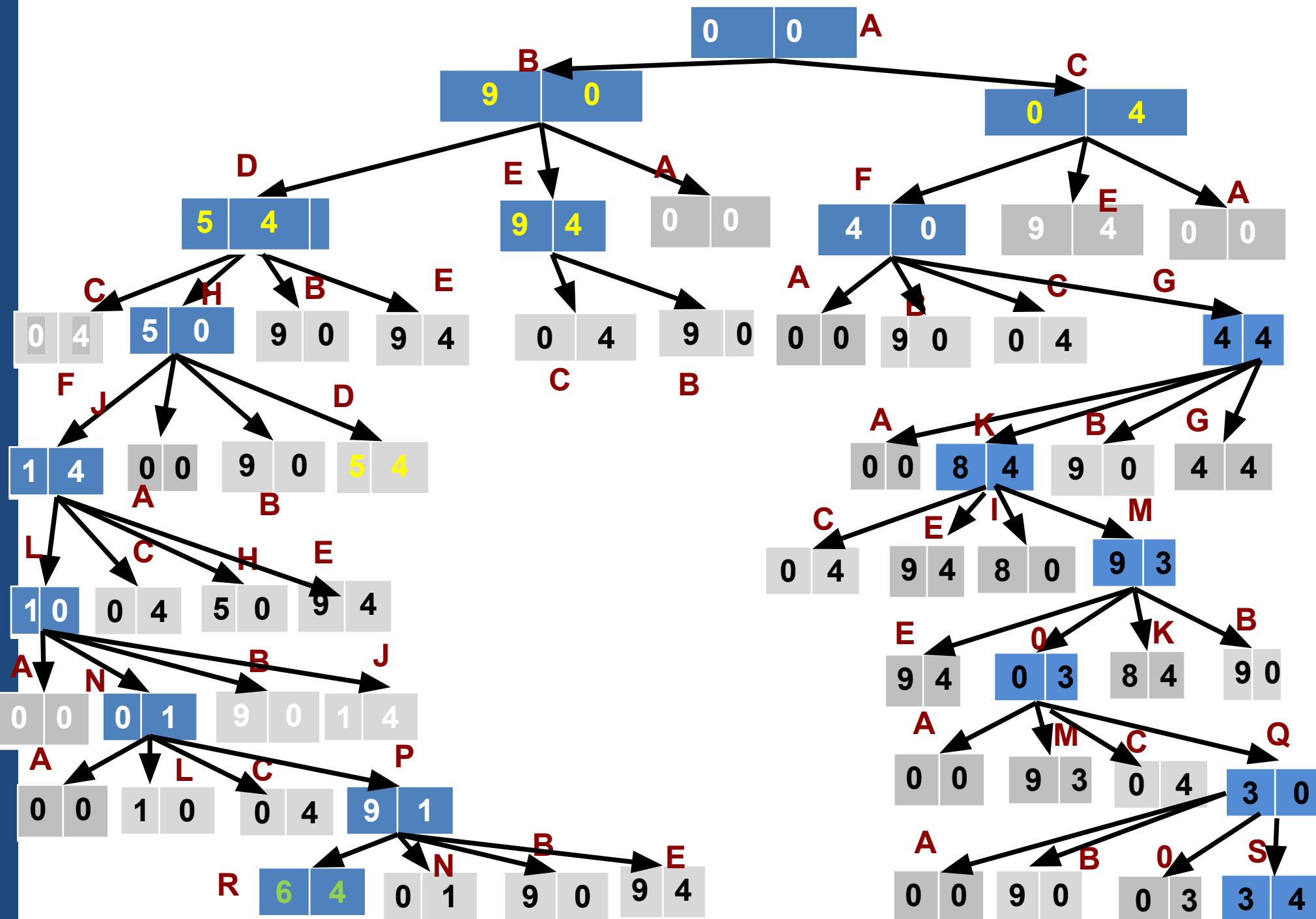


- **Bước 9:**  $[J:1-4, I:8-0]$ ; closed =  $[A,B,C,D,E,F,H,G]$   
Xét J, ko phải trạng thái đích  
Đưa J vào Closed: closed =  $[A,B,C,D,E,F,H,G,J]$
- Các con của J: L: 1-0, C:0-4, H:5-0, E:9-4
- Loại các con đã tồn tại trong open và closed – loại C:0-4, H:5-0, E:9-4
- Thêm L: 1-0, vào Open =  $[I:8-0, L: 1-0, ]$ ; closed =  $[A,B,C,D,E,F,H,G,J]$

# Ví dụ: Bài toán đong nước



- Bước 10:....



## Tìm kiếm sâu (depth-first search)

Procedure depth – first –search;

Begin % khởi đầu

    Open:= [start];      Closed:= [ ];

    While open ≠ [ ] do % còn các trạng thái chưa khảo sát

        Begin

*Loại bỏ trạng thái ngoài cùng bên trái khỏi open, gọi nó là X;*

*If X là một đích then trả lời kết quả (thành công)% tìm thấy đích*  
            *else begin*

*Phát sinh các con của X;*

*Đưa X vào closed;*

*Loại các con của X trong open hoặc closed;*

*các con còn lại vào **đầu bên trái** của open; %**ngăn xếp***  
                *end;*

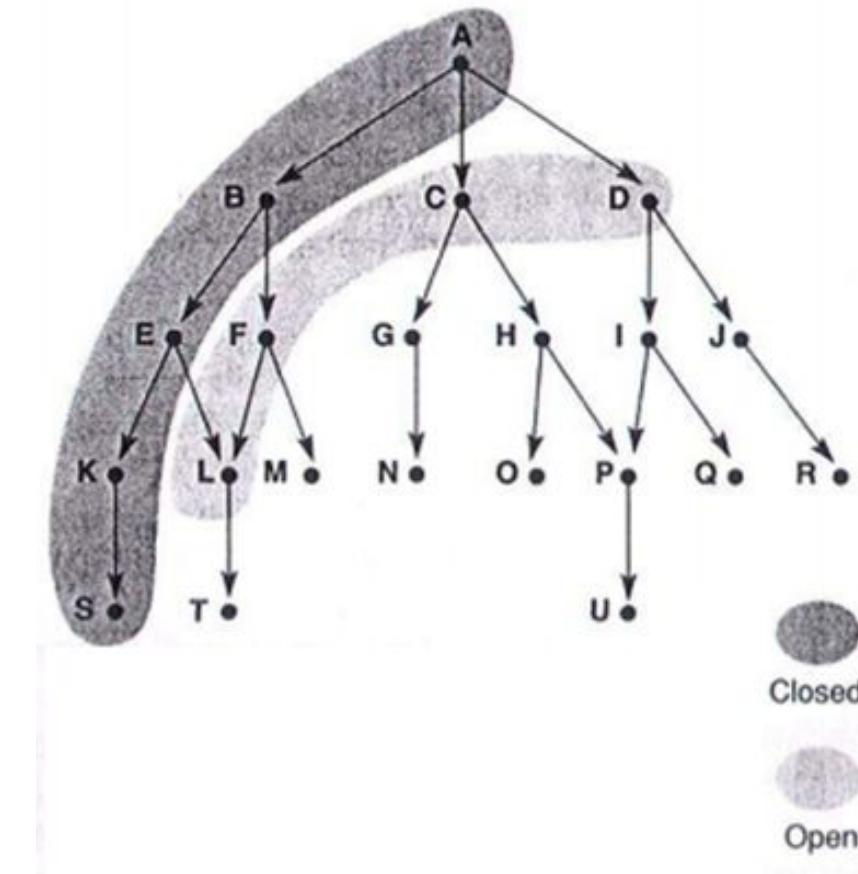
*End;*

*Trả lời kết quả (thất bại); % không còn trạng thái nào*

*End;*

# Tìm kiếm sâu (depth-first search - DFS)

- Tìm kiếm sâu:
- Các bước thực hiện tìm kiếm sâu:
  1. Open = [A]; closed = []
  2. Open = [B,C,D]; closed = [A]
  3. Open = [E,F,C,D];closed = [B,A]
  4. Open = [K,L,F,C,D];  
closed = [E,B,A]
  5. Open = [S,L,F,C,D];  
closed = [K,E,B,A]
  6. Open = [L,F,C,D];  
closed = [S,K,E,B,A]
  7. Open = [T,F,C,D];  
closed = [L,S,K,E,B,A]
  8. Open = [F,C,D];  
closed = [T,L,S,K,E,B,A]
  9. ....



# Sử dụng BFS, DFS để tìm trạng thái đích

3	1
2	

Start

1	2
3	

Goal

# Sử dụng BFS, DFS để tìm trạng thái đích

Start

3	1
2	

Goal

1	2
3	

Procedure breadth-first-search;

**Begin** % khởi đầu

Open:= [start];

Closed:= [ ];

While open ≠ [ ] do % còn các trạng thái chưa khảo sát

**Begin**

Loai bỏ trạng thái ngoài cùng bên trái khỏi open, gọi nó là X;

If X là một đích then trả lời kết quả (thành công) % tìm thấy đích

**else begin**

Phát sinh các con của X;

Đưa X vào closed;

Loai các con của X đã tồn tại trong open hoặc closed; % kiểm tra vòng lặp

Đưa các con còn lại của X vào đầu bên phải của open; % hàng đợi

**end;**

**End;**

Trả lời kết quả (thất bại);

% không còn trạng thái nào

**End;**

# Sử dụng BFS, DFS để tìm trạng thái đích

Start	
3	1
2	

Goal

Procedure depth – first –search;

Begin % khởi đầu

Open:= [start]; Closed:= [ ];

While open  $\neq$  [ ] do % còn các trạng thái chưa khảo sát

Begin

Loai bỏ trạng thái ngoài cùng bên trái khỏi open, gọi nó là X;  
If X là một đích then trả lời kết quả (thành công) % tìm thấy đích  
else begin

Phát sinh các con của X;

Đưa X vào closed;

Loai các con của X trong open hoặc closed;  
Đưa các con còn lại vào đầu bên trái của open; % ngăn xếp  
end;

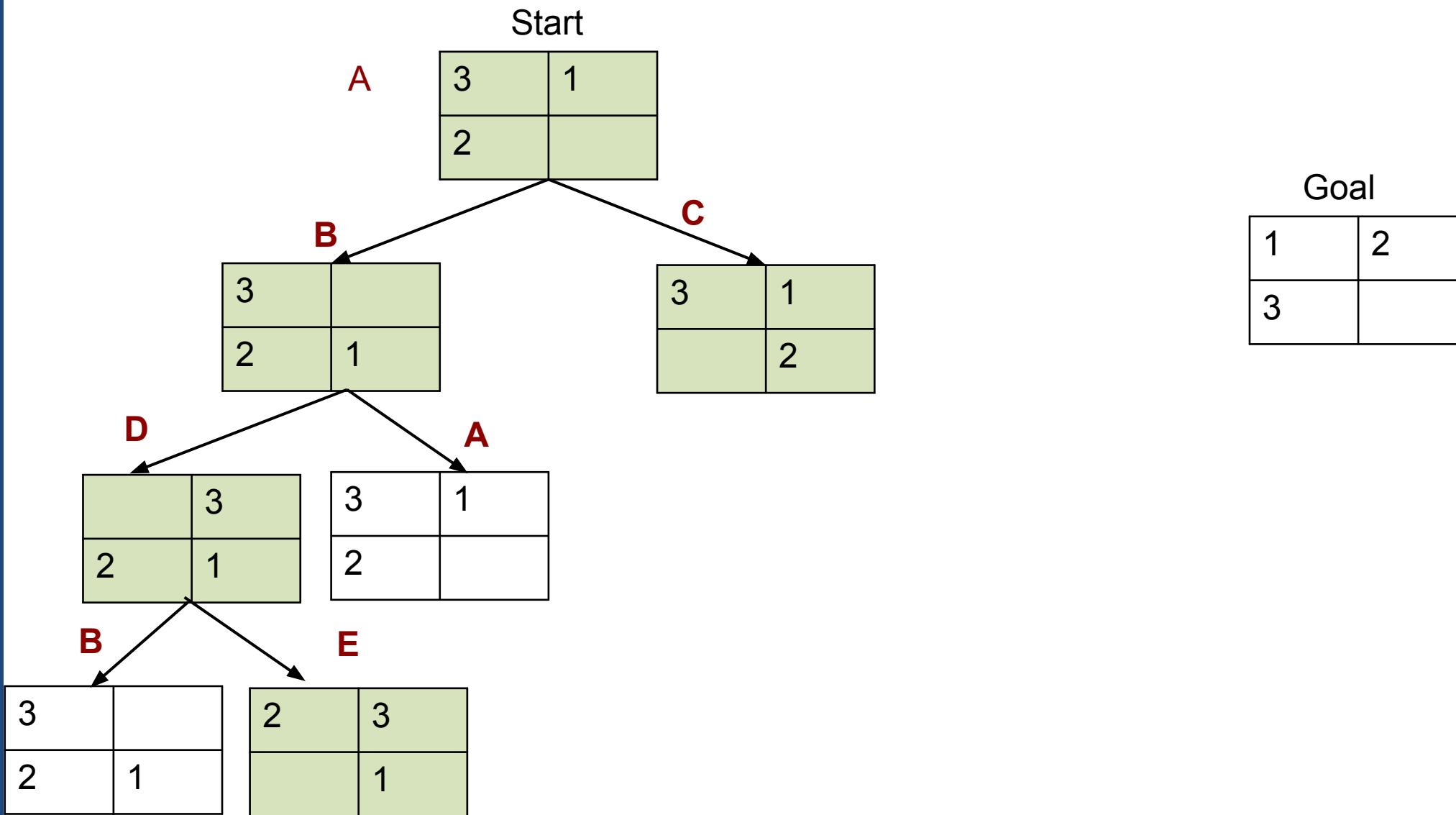
End;

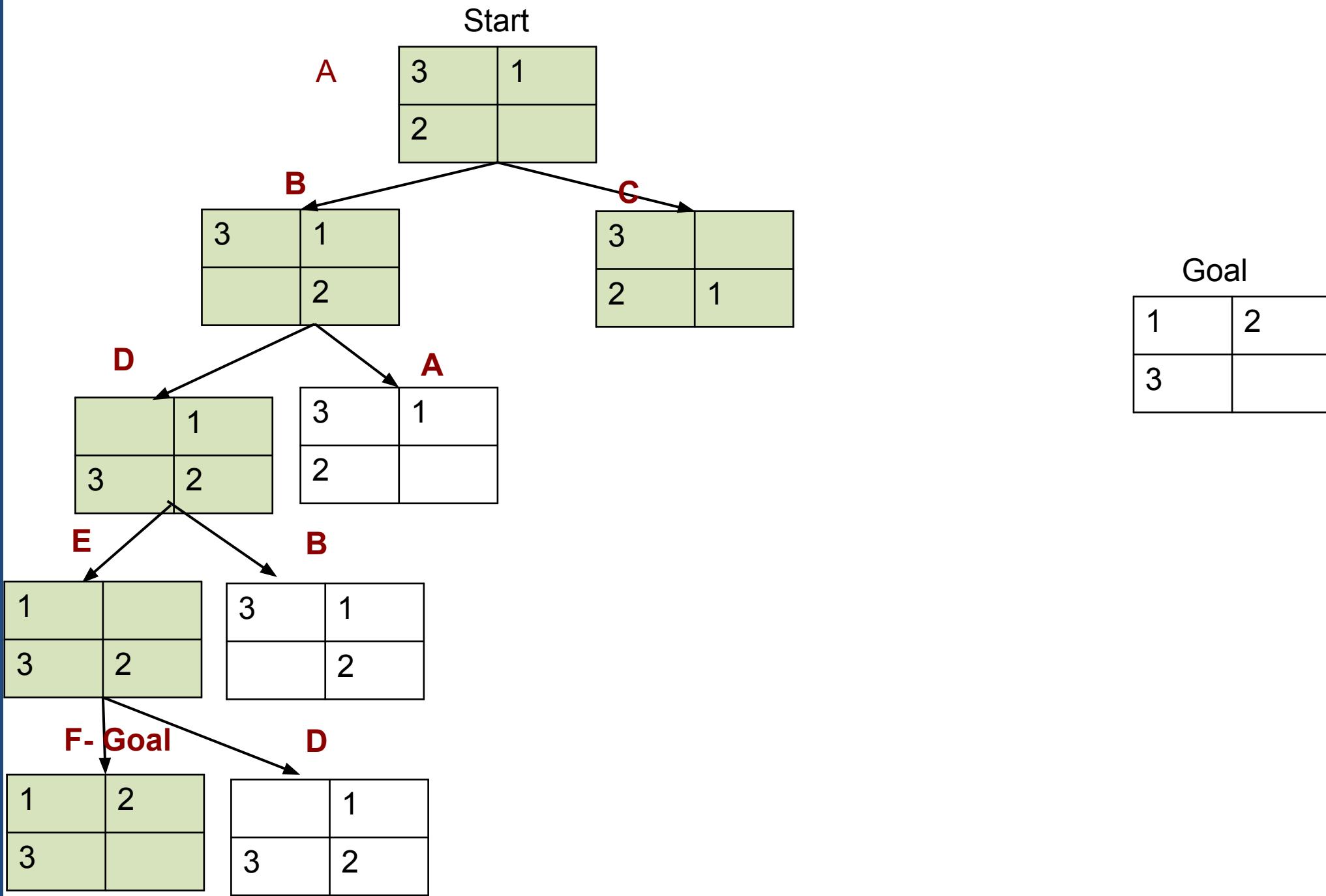
Trả lời kết quả (thất bại); % không còn trạng thái nào

End;

**Tìm kiếm sâu  
(depth-first search)**

# Sử dụng BFS, DFS để tìm trạng thái đích





# Tìm kiếm theo độ sâu có giới hạn (depth-limited search)

- Tương tự như tìm kiếm theo độ sâu, tuy nhiên chỉ tìm đến một độ sâu **d** nhất định nào đó

# Tìm kiếm theo chiều sâu lặp (iterative deepening depth-first search)

- Tương tự như tìm kiếm theo độ sâu, tuy nhiên lặp lại việc tìm kiếm với độ sâu giới hạn được tăng dần cho đến khi tìm được trạng thái đích
- *Vd: lần thứ nhất giới hạn độ sâu = 1, nếu tìm không thấy tăng độ sâu giới hạn lên 2, ...*

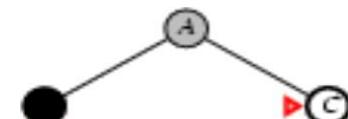
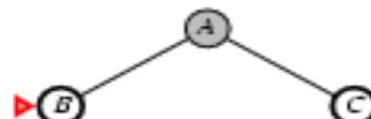
Tìm kiếm sâu lăp,  $l = 0$

Limit = 0



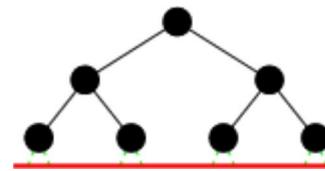
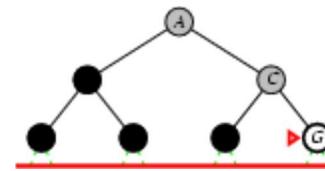
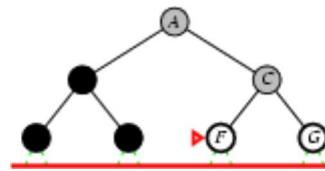
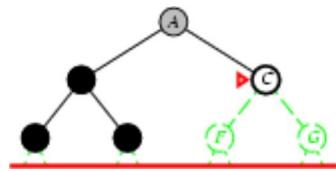
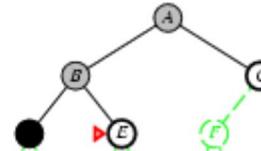
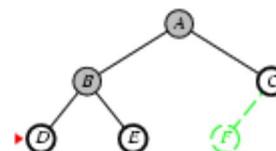
# Tìm kiếm sâu lấp, $l = 1$

Limit = 1

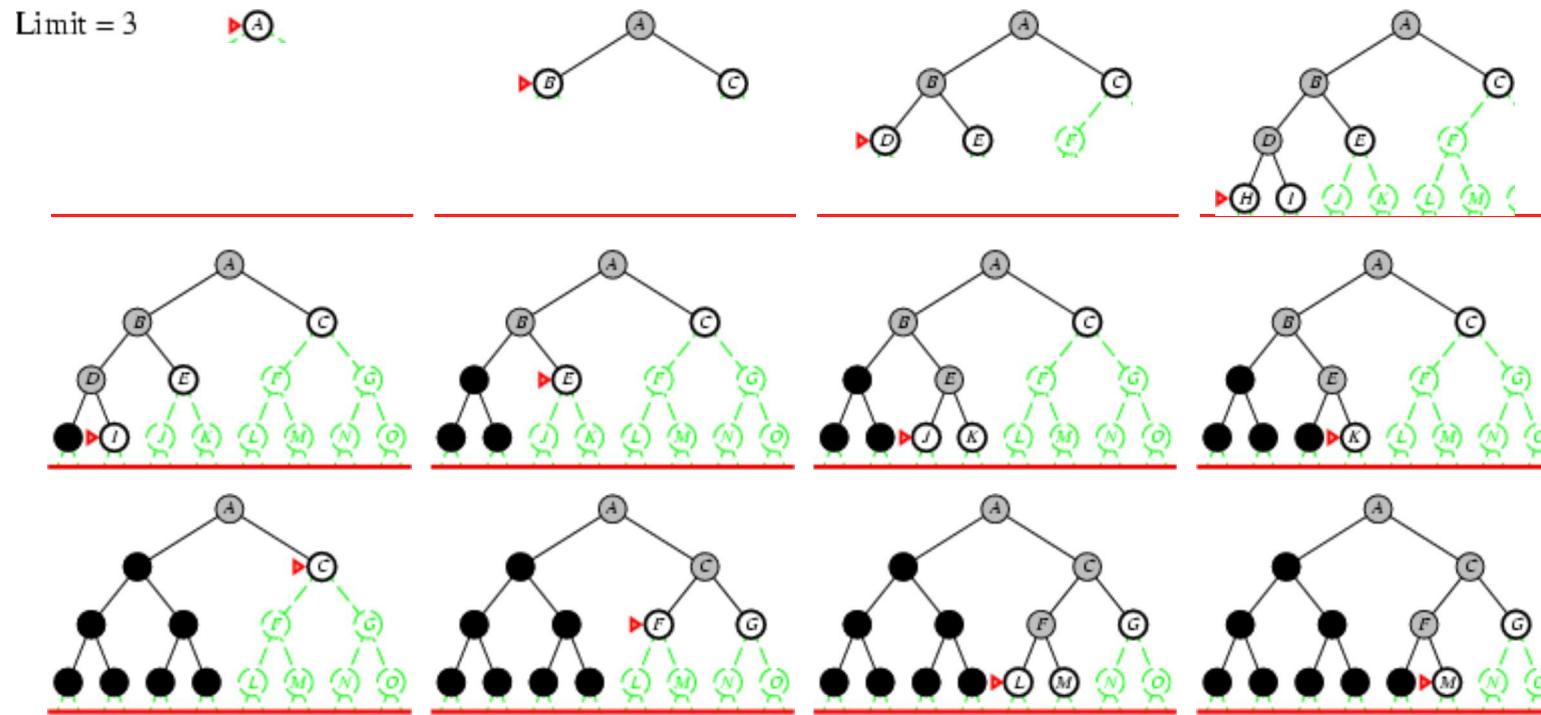


# Tìm kiếm sâu lấp, $l = 2$

- Limit = 2



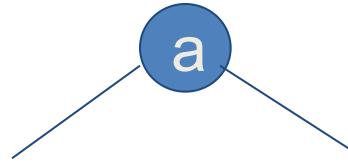
# Tìm kiếm sâu lấp, $l = 3$



# Tìm kiếm giá thành đồng nhất (Uniform-cost search)

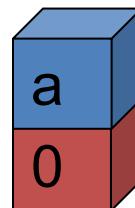
- Tìm kiếm theo chiều rộng
  - Đảm bảo tìm ra giải pháp cho bài toán
  - Không chắc tìm ra **đường đi chi phí thấp nhất**
- Tìm kiếm giá thành đồng nhất (rất giống GT Dijkstra)
  - Chọn nút có **giá thành đường đi** đến nó thấp nhất mà triển khai
  - Đảm bảo tìm được giải pháp với chi phí thấp nhất nếu biết rằng chi phí tăng khi chiều dài đường đi tăng
  - Tối ưu và đầy đủ
  - Nhưng có thể rất chậm

# Ví dụ Uniform Cost Search

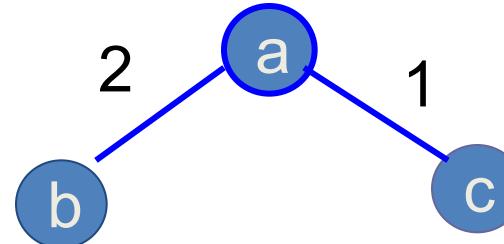


***Closed list:***

***Open list:***



# Ví dụ Uniform Cost Search

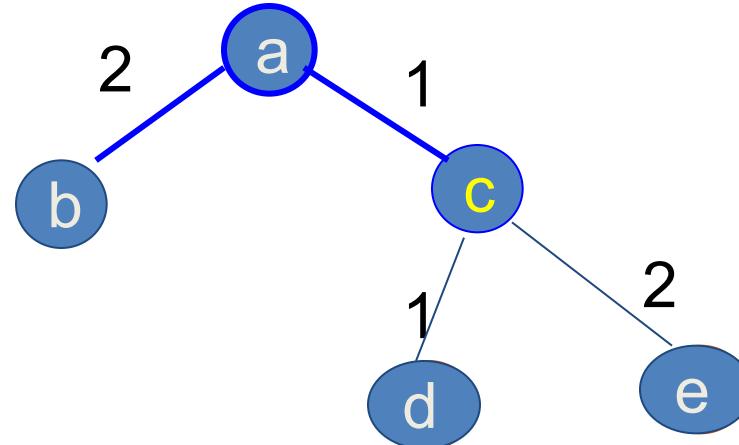


*Closed list:* a

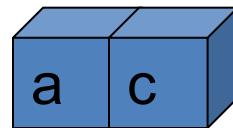
*Open list:*

b	c
2	1

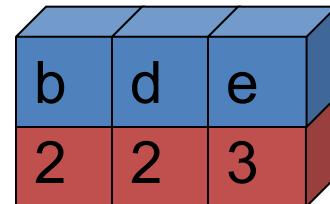
# Ví dụ Uniform Cost Search



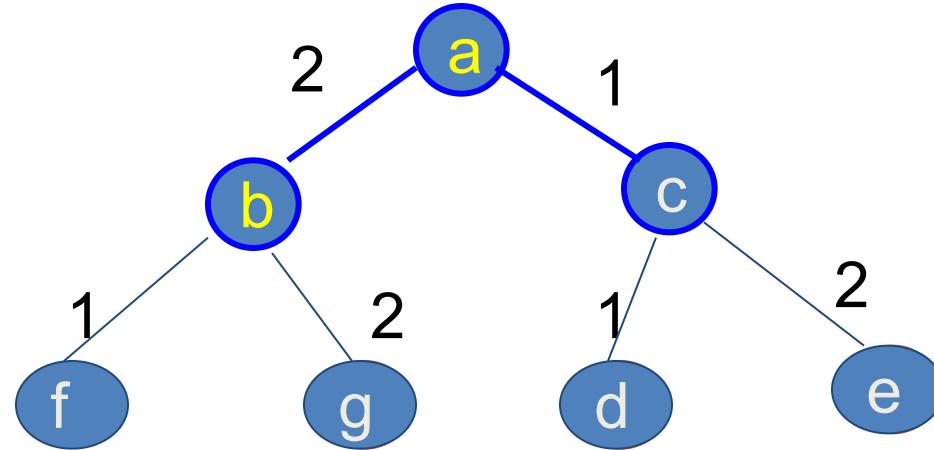
***Closed list:***



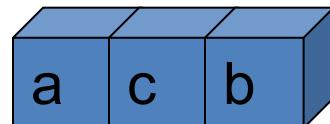
***Open list:***



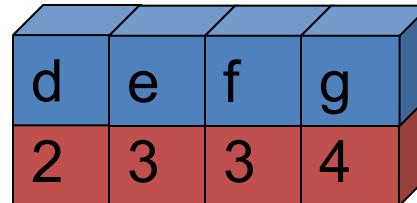
# Ví dụ Uniform Cost Search



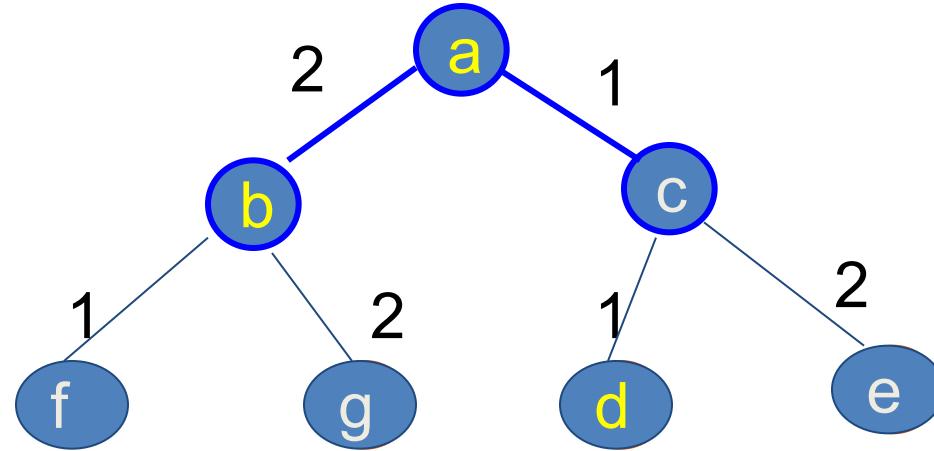
**Closed list:**

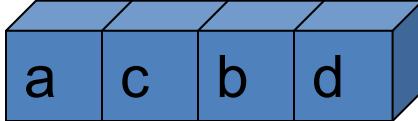


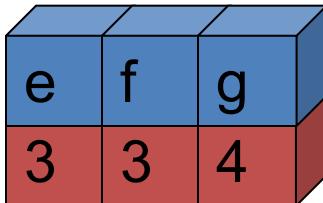
**Open list:**



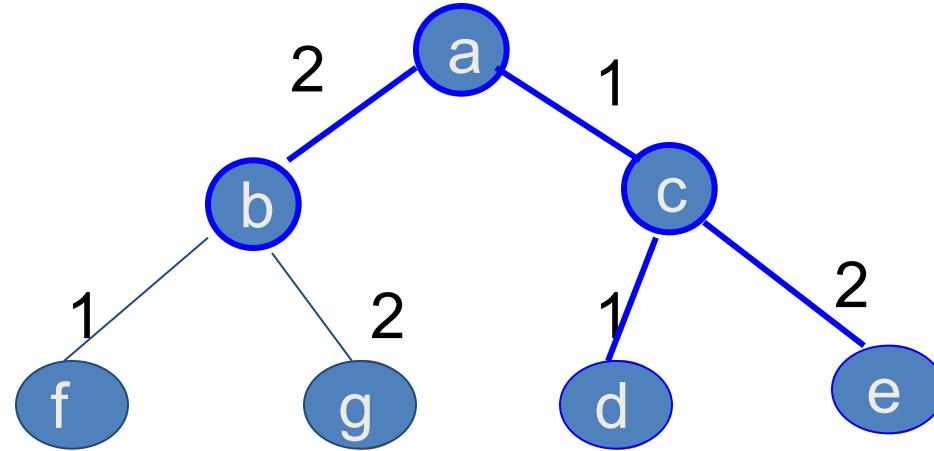
# Ví dụ Uniform Cost Search

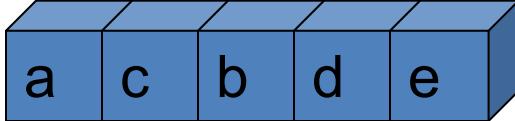


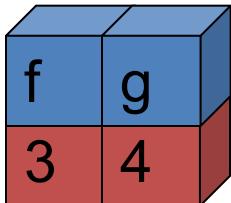
**Closed list:** 

**Open list:** 

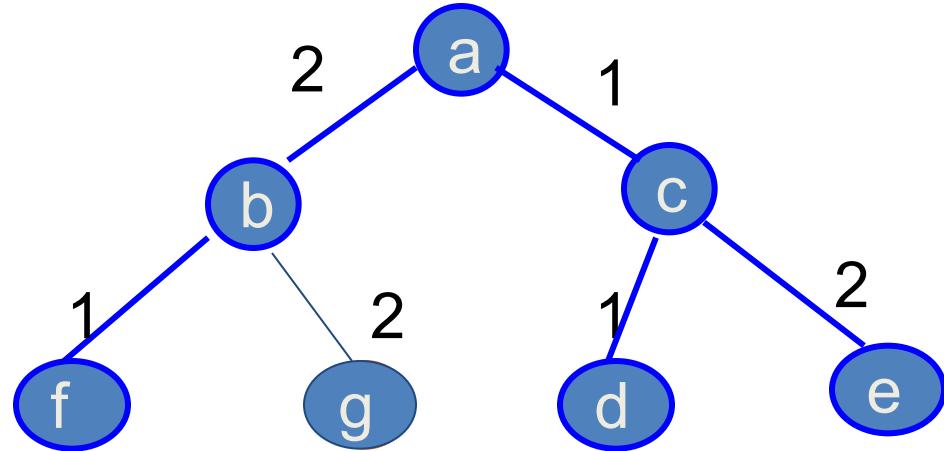
# Ví dụ Uniform Cost Search



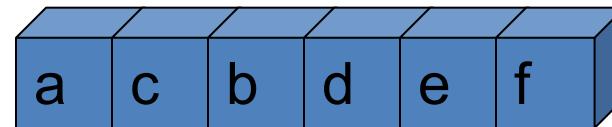
**Closed list:**  a | c | b | d | e

**Open list:**  f | g  
3 | 4

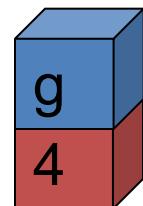
# Ví dụ Uniform Cost Search



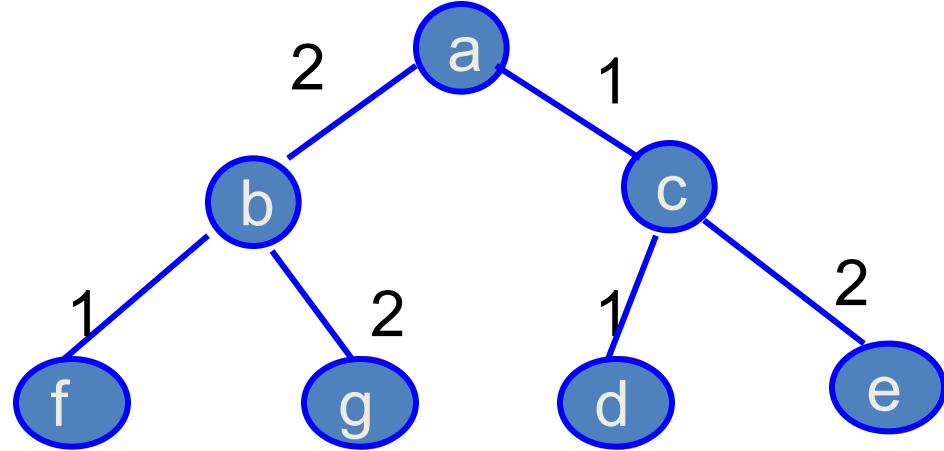
***Closed list:***



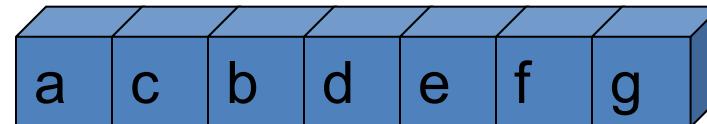
***Open list:***



# Ví dụ Uniform Cost Search



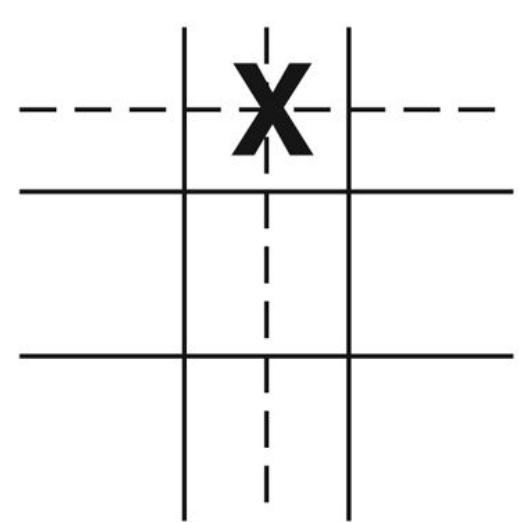
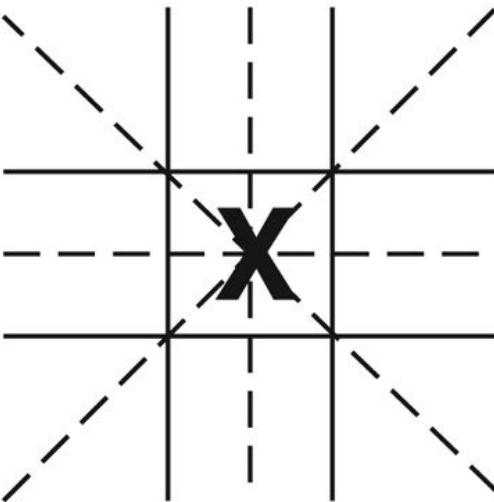
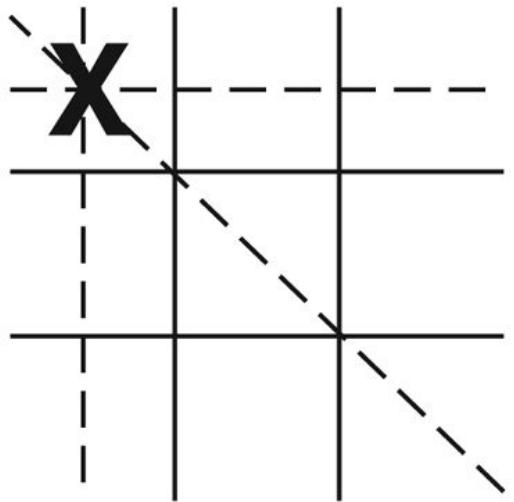
***Closed list:***



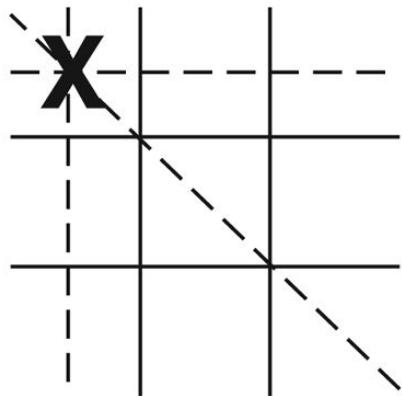
***Open list:***

# TÌM KIẾM DỰA TRÊN KINH NGHIỆM (INFORMED/ HEURISTIC SEARCH)

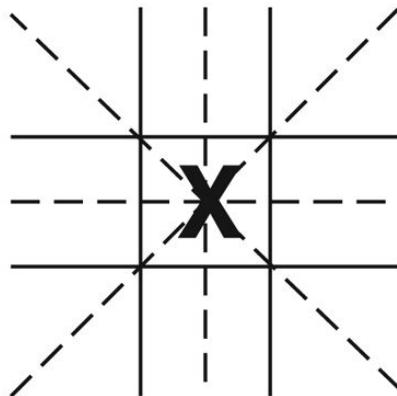




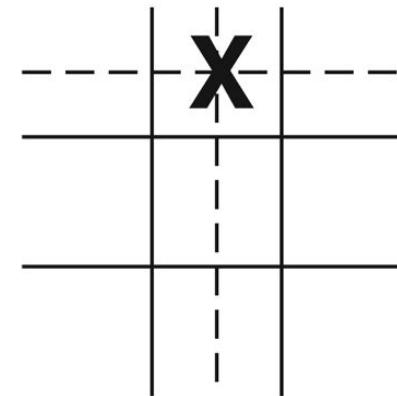
# Phép đo heuristic (2)



Three wins through  
a corner square



Four wins through  
the center square



Two wins through  
a side square

Heuristic “Số đường thắng nhiều nhất” áp dụng cho các nút con đầu tiên trong tic-tac-toe.

# Heuristic

- Để giải quyết các bài toán lớn, phải cung cấp những **kiến thức đặc trưng ở từng lĩnh vực để nâng cao hiệu quả của việc tìm kiếm**
- Trong TK KGTT, heuristic là các luật dùng để chọn những nhánh/ **lựa chọn nào có nhiều khả năng nhất** dẫn đến một giải pháp chấp nhận được

# Heuristic

## ■ Sử dụng Heuristic trong trường hợp nào?

- Vấn đề có thể có giải pháp chính xác, nhưng **chi phí tính toán** để tìm ra nó không cho phép. VD: cờ vua, ...
- Vấn đề có thể **không có giải pháp chính xác** vì sự không rõ ràng trong điều kiện vấn đề hoặc trong các dữ liệu có sẵn. VD: chẩn đoán y khoa, ...

# Heuristic

- Thuật toán Heuristic gồm hai phần:
  1. **Phép đo Heuristic:** thể hiện qua hàm đánh giá heuristic, dùng để đánh giá các đặc điểm của một trạng thái trong KGTT.
  2. **Giải thuật tìm kiếm heuristic:**
    - Tìm kiếm tốt nhất đầu tiên (best-first search)
      - *Tìm kiếm háu ăn (greedy best-first search)*
      - *Giải thuật A\**
    - Tìm kiếm leo đồi

# Heuristic

- **Phép đo Heuristic:**

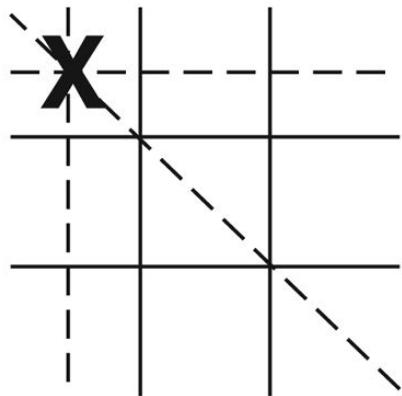
- *Ước lượng chi phí tối ưu giữa hai hoặc nhiều giải pháp*
- *Không quá tốn kém để tính toán*
- **Được xác định cụ thể trong từng bài toán** (Ví dụ: đối với bài toán TSP, đánh giá khoảng cách giữa các thành phố sao cho chi phí là thấp nhất)

- **Hàm đánh giá Heuristic** tại trạng thái n là  $f(n)$ :

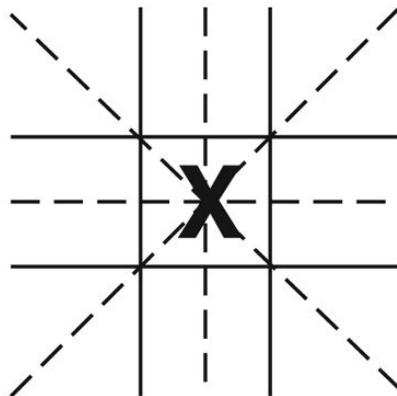
$$f(n) = g(n) + h(n)$$

- *g(n) = khoảng cách thực sự từ n đến trạng thái bắt đầu*
- *h(n) = ước lượng heuristic cho khoảng cách từ trạng thái n đến trạng thái đích*

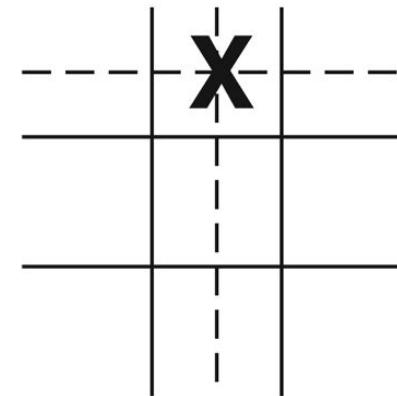
# Phép đo heuristic (2)



Three wins through  
a corner square



Four wins through  
the center square



Two wins through  
a side square

Heuristic “Số đường thắng nhiều nhất” áp dụng cho các nút con đầu tiên trong tic-tac-toe.

# Cài đặt hàm đánh giá Heuristic

- Xét trò chơi 8-puzzle. Hàm đánh giá Heuristic tại trạng thái  $n$  là  $f(n)$ :

$$f(n) = g(n) + h(n)$$

- $g(n)$  = khoảng cách thực sự từ  $n$  đến **trạng thái bắt đầu**
- $h(n)$  = ước lượng heuristic cho khoảng cách từ **trạng thái n đến trạng thái đích**

# Cài đặt hàm đánh giá Heuristic

$$f(n) = g(n) + h(n)$$

1	2	3
8		4
7	6	5

goal

$$g(n) = 0$$

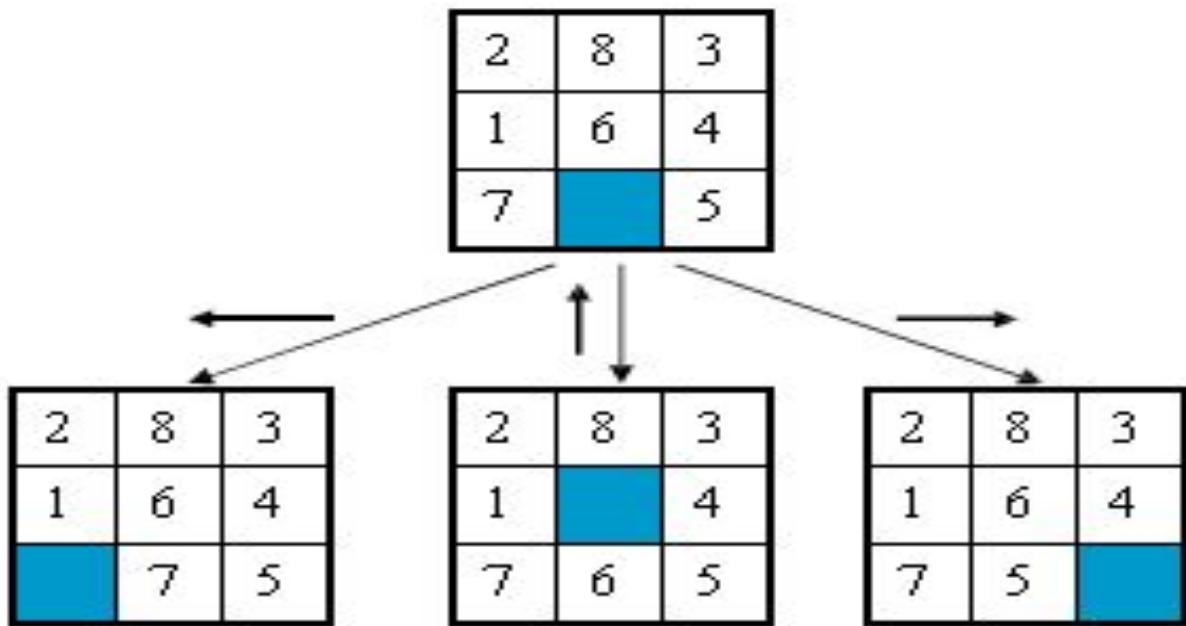
$$g(n) = 1$$

$$f(n) =$$

**6**

(A)

start



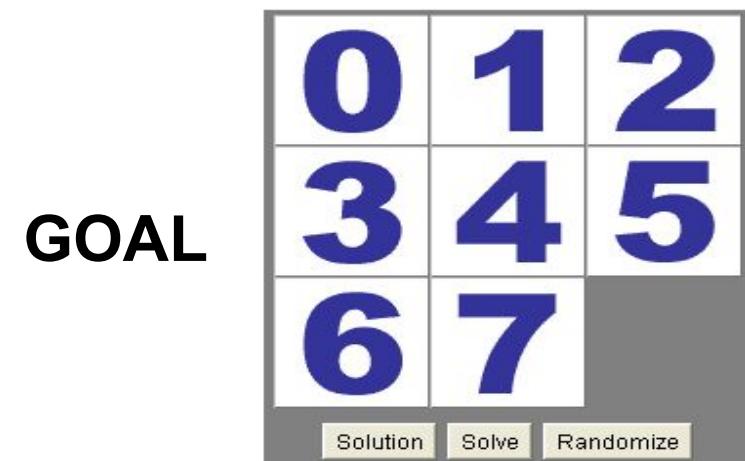
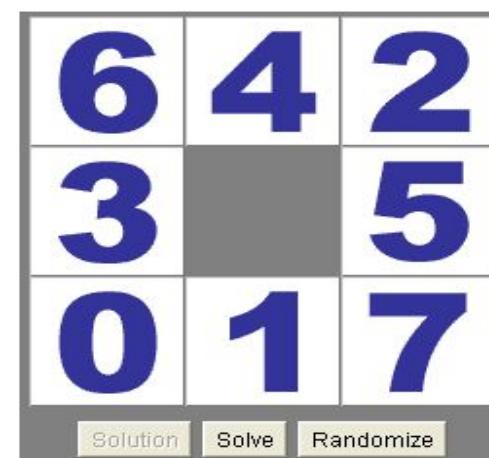
(B)

**6**

(C)

# Cài đặt hàm đánh giá Heuristic

- Việc ước lượng heuristic cho khoảng cách từ trạng thái n đến trạng thái đích như thế nào?
  - Giả sử có các định nghĩa sau:
    - $h_1(n)$  = số vị trí sai khác của trạng thái n so với goal
    - $h_2(n)$  = khoảng cách dịch chuyển ( $\leftarrow, \rightarrow, \uparrow, \downarrow$ ) ngắn nhất để dịch chuyển các ô chữ nằm sai vị trí về vị trí đúng (khoảng cách Manhattan)



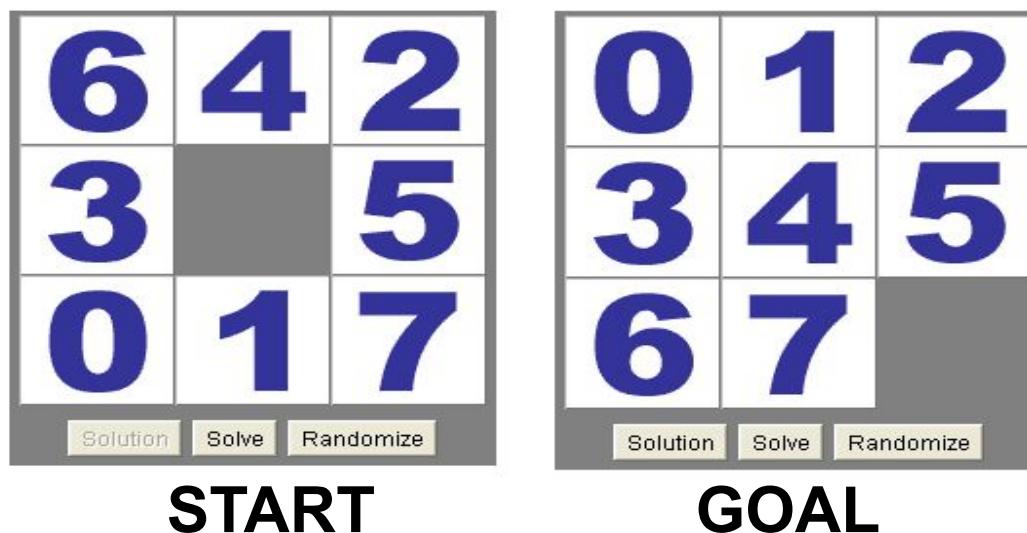
# Cài đặt hàm đánh giá Heuristic

- Việc ước lượng heuristic cho khoảng cách từ trạng thái n đến trạng thái đích như thế nào?
  - Giả sử có các định nghĩa sau:
    - $h_1(n)$  = số vị trí sai khác của trạng thái n so với goal
    - $h_2(n)$  = khoảng cách Manhattan
  - $\Rightarrow h_1(n) = ???$
  - $\Rightarrow h_2(n) = ???$

■  $h_1(n) = 6.$

■  $h_2(n) = 8.$





# Tìm kiếm tốt nhất đầu tiên (Best-first-search)

- **Ý tưởng:** sử dụng hàm đánh giá trong quá trình phát triển cây tìm kiếm, hàm đánh giá ước lượng độ gần của mỗi đỉnh trạng thái với trạng thái đích, chỉ triển khai cây tìm kiếm theo nhánh có triển vọng đi đến đích nhanh nhất
- *Tìm kiếm tốt nhất đầu tiên* (best-first search) là tiếp cận tổng quát của tìm kiếm với thông tin bổ sung.
- Việc chọn nút để triển khai dựa trên một *hàm lượng giá* (evaluation function):  $f(n)$

# Tìm kiếm tốt nhất đầu tiên (Best-first-search)

- *hàm lượng giá* (evaluation function):  $f(n)$  được xây dựng dựa trên việc ước lượng chi phí và nút có giá ước lượng nhỏ nhất được ưu tiên triển khai trước. Sự lựa chọn hàm sẽ quyết định chiến lược tìm kiếm.
- Hàm  $h(n)$  là *chi phí ước lượng* của đường đi tốt nhất từ trạng thái của nút đến trạng thái mục tiêu.
  - $f(n) = g(n) + h(n)$
  - $g(n)$ : *chi phí thực tế đi từ gốc đến n*
  - $h(n)$ : *chi phí ước lượng đi từ n đến nút mục tiêu*
- Hàm heuristic là cách thường dùng nhất để sử dụng kiến thức bổ sung trong quá trình tìm kiếm:  $h(n)$  là một hàm không âm bất kỳ và *phụ thuộc vào vấn đề đang giải quyết*. Nếu là nút mục tiêu thì  $h(n) = 0$ .

# Tìm kiếm tốt nhất đầu tiên (Best-first-search)

- Dùng mảng có sắp xếp theo hàm đánh giá
- Tương tự DFS và BFS, best-first search dùng các danh sách để lưu trữ các trạng thái:
  - *OPEN*: lưu các trạng thái sắp được kiểm tra
  - *CLOSED*: lưu các trạng thái đã duyệt qua
- Các trạng thái trong **OPEN** list sẽ được sắp xếp theo thứ tự dựa trên 1 hàm Heuristic nào đó (**đặt thứ tự ưu tiên cho các giá trị gần trạng thái đích**)
- Do đó, mỗi lần lặp sẽ xem xét trạng thái tiềm năng nhất trong **OPEN** list

## Sử dụng tìm kiếm tốt nhất đầu tiên để tìm trạng thái goal

- Xét trò chơi 8-ô, mỗi trạng thái n, một giá trị  $f(n)$ :  
 **$f(n) = g(n) + h(n)$**

- $g(n)$  = khoảng cách thực sự từ n đến trạng thái bắt đầu
- $h(n)$  = hàm heuristic đánh giá khoảng cách từ trạng thái n đến mục tiêu.

1	2	3
8		4
7	6	5

goal

$$g(n) = 0$$

$h(n)$ : số lượng các vị trí còn sai;  $g(n) = 1$

$$f(n) =$$

2	8	3
1	6	4
7	5	

6

2	8	3
1		4
7	6	5

4

2	8	3
1	6	4
7	5	

6 21

start

2	8	3
1	6	4
7	5	

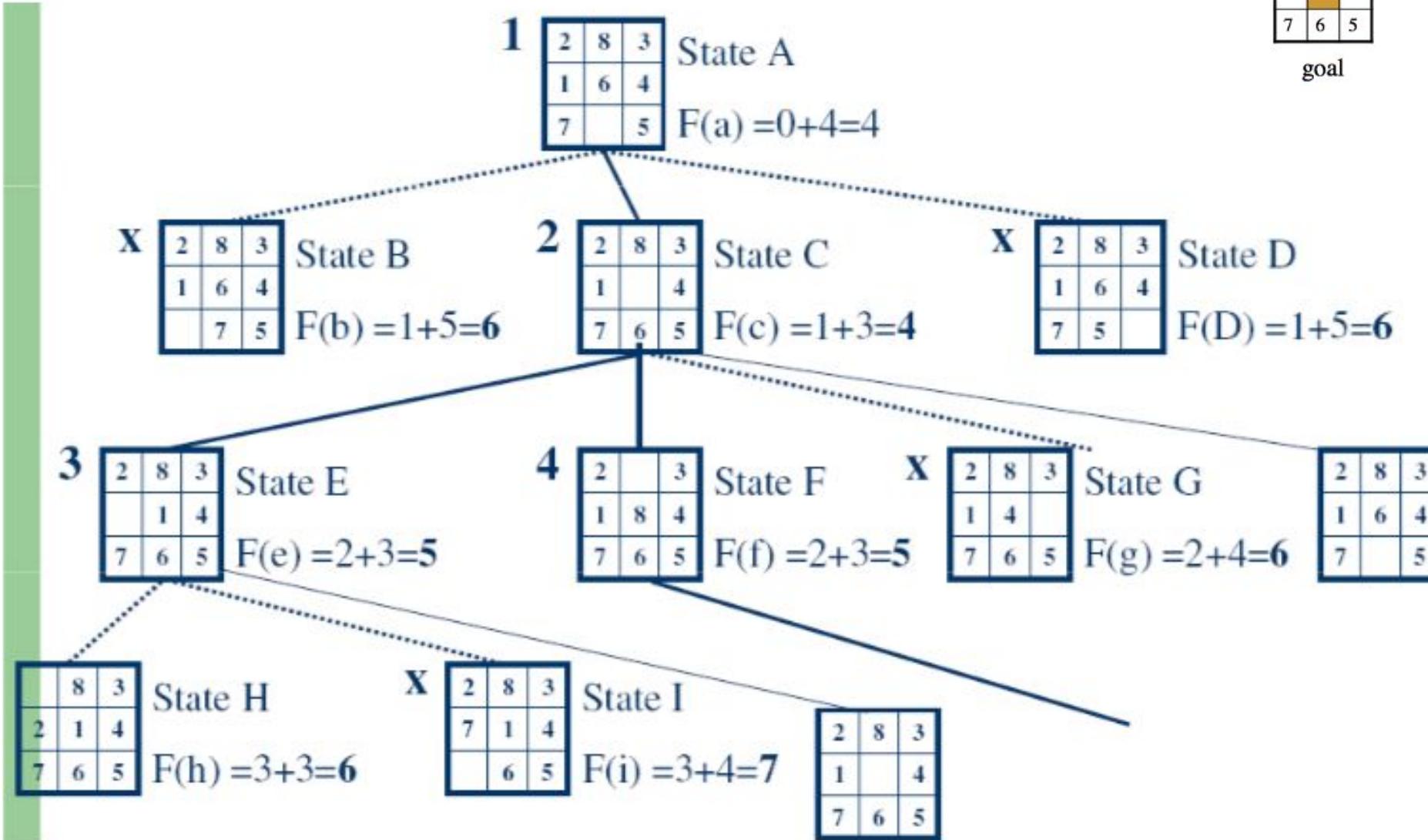
2	8	3
1		4
7	6	5

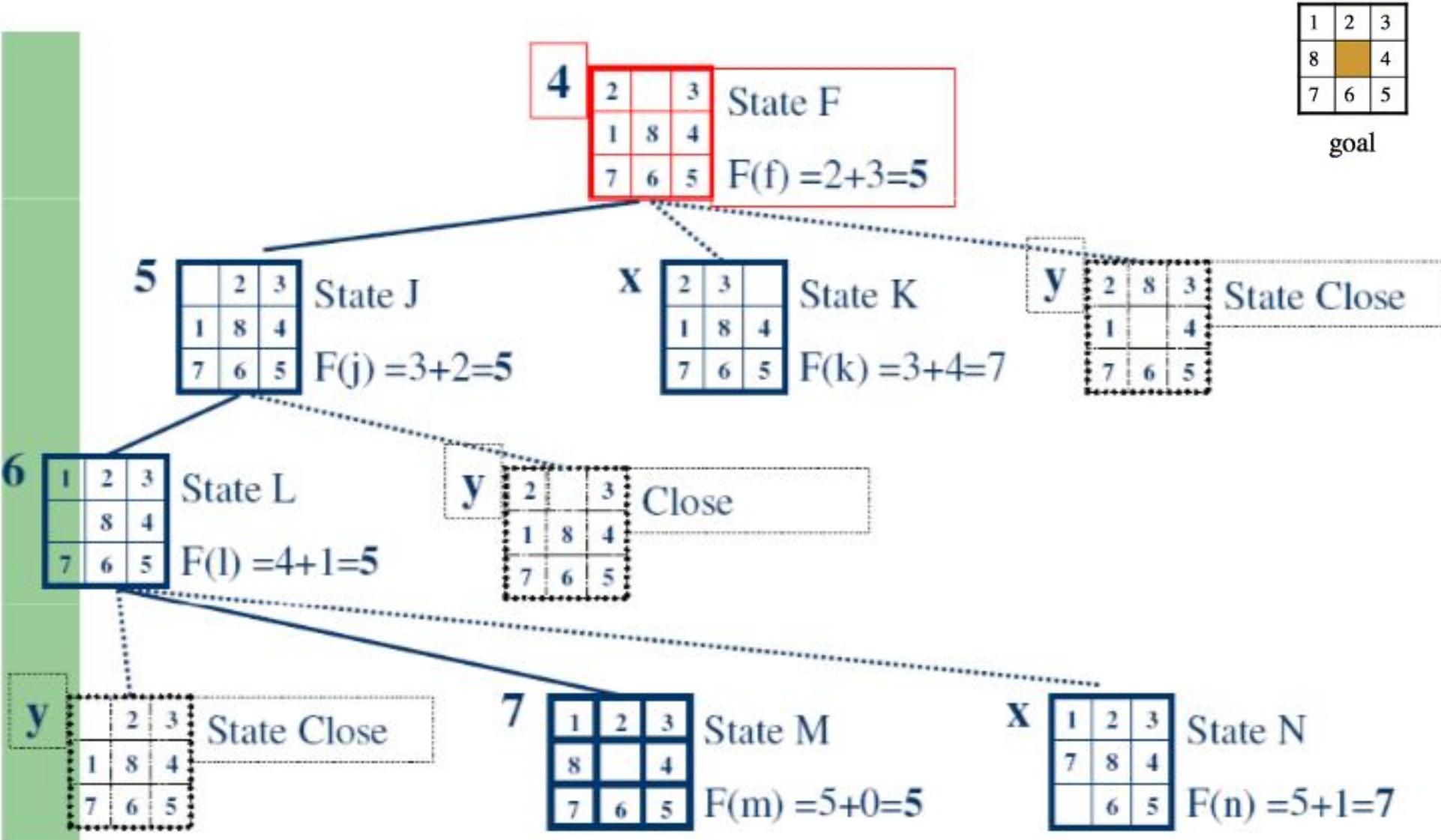
2	8	3
1	6	4
7	5	

# Ví dụ

1	2	3
8		4
7	6	5

goal





# Tìm kiếm háu ăn (greedy best-first search)

- *Tìm kiếm háu ăn* (greedy best-first search) hay còn gọi là *tìm kiếm chỉ sử dụng heuristic* (pure heuristic search) cố gắng triển khai nút “gần” với mục tiêu nhất.
- Vì thế, nó chỉ dùng hàm heuristic  $h(n)$  để lượng giá các nút, tức là

$$f(n) = h(n)$$

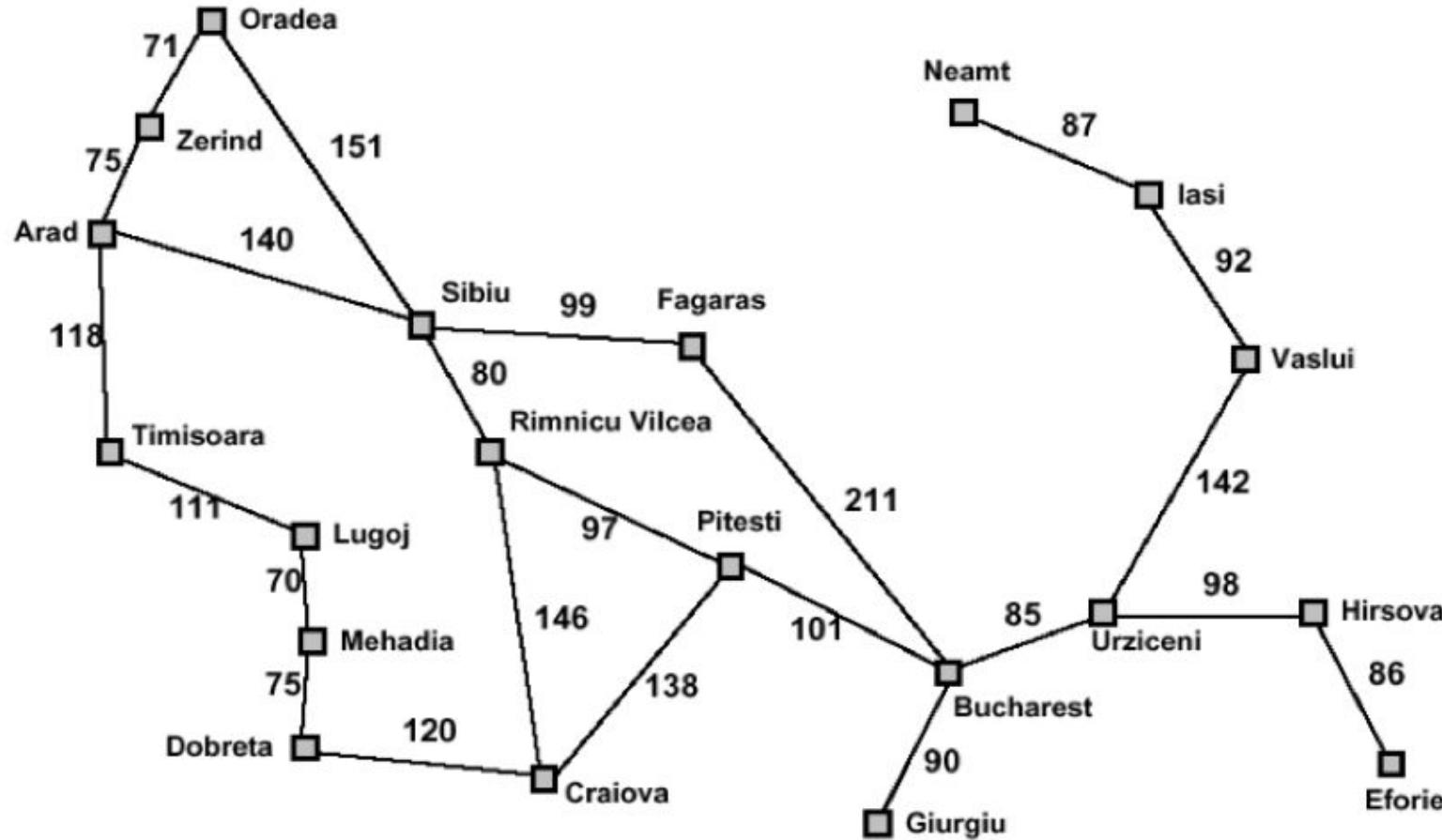
# *Tìm kiếm háu ăn (greedy best-first search)*

Bài toán tìm đường:

- Thành phố xuất phát: Arad
- Thành phố đích: Bucharest
- Các cạnh biểu diễn đường nối trực tiếp giữa hai thành phố, các con số ghi trên các cạnh là chi phí đi giữa hai thành phố.
- Cột bên phải là khoảng cách Euclid từ các thành phố đến thành phố đích Bucharest. ( $h(n)$ )

# Tìm kiếm h้าu ăn (greedy best-first search)

- Initial State = Arad
- Goal State = Bucharest

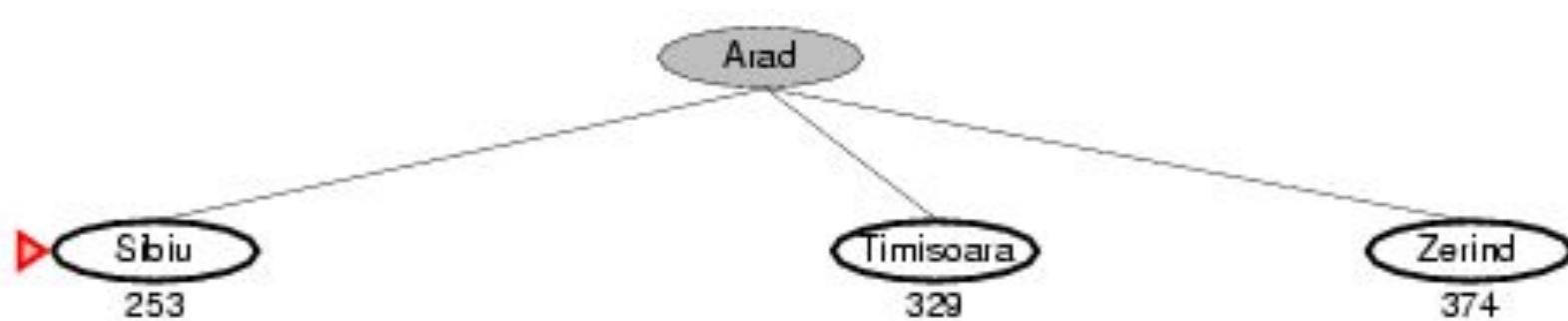


Straight-line distance  
to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

# Tìm kiếm h้าu ăn

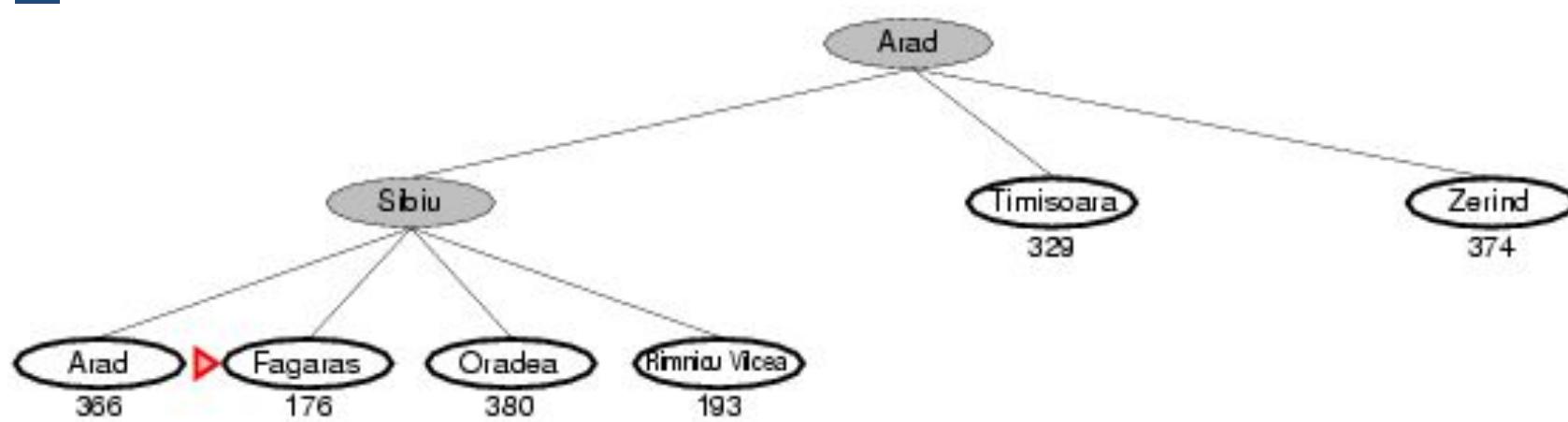
- Initial State = Arad
- Goal State = Bucharest



	Straight-line distance to Bucharest
<b>Arad</b>	366
<b>Bucharest</b>	0
<b>Craiova</b>	160
<b>Dobreta</b>	242
<b>Eforie</b>	161
<b>Fagaras</b>	178
<b>Giurgiu</b>	77
<b>Hirsova</b>	151
<b>Iasi</b>	226
<b>Lugoj</b>	244
<b>Mehadia</b>	241
<b>Neamt</b>	234
<b>Oradea</b>	380
<b>Pitesti</b>	98
<b>Rimnicu Vilcea</b>	193
<b>Sibiu</b>	253
<b>Timisoara</b>	329
<b>Urziceni</b>	80
<b>Vaslui</b>	199
<b>Zerind</b>	374

# Tìm kiếm h้าu ăn

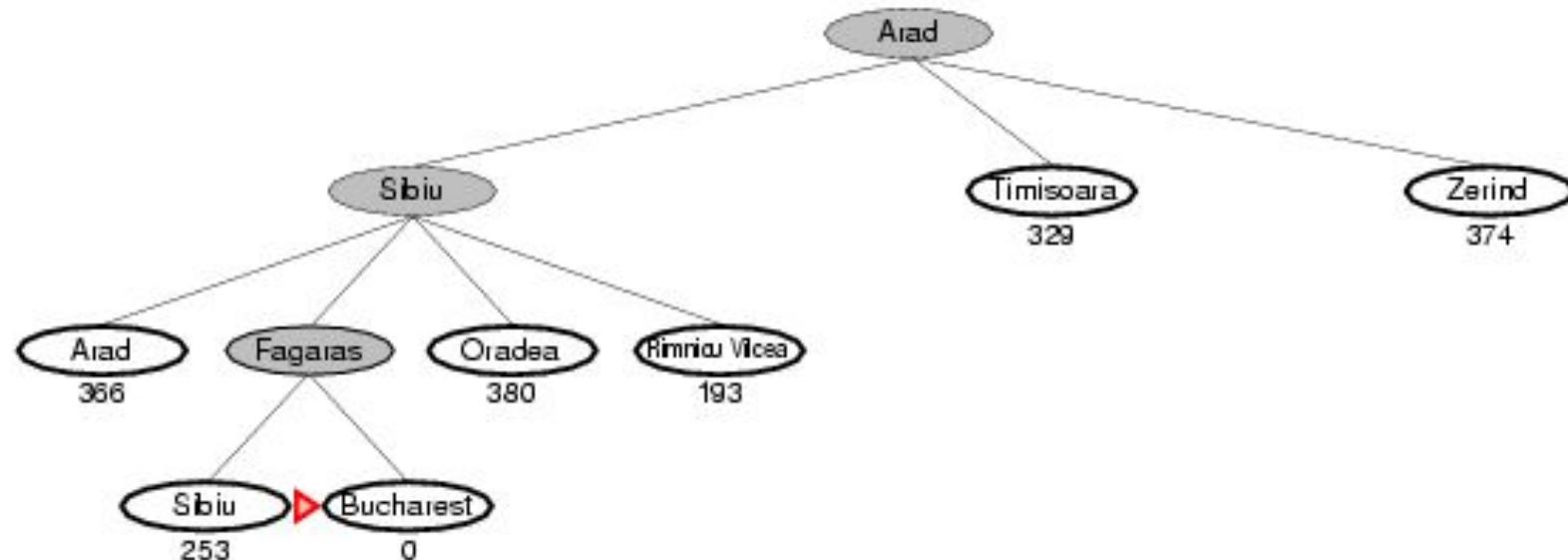
- Initial State = Arad
- Goal State = Bucharest



Straight-line distance to Bucharest	
<b>Arad</b>	366
<b>Bucharest</b>	0
<b>Craiova</b>	160
<b>Dobreta</b>	242
<b>Eforie</b>	161
<b>Fagaras</b>	178
<b>Giurgiu</b>	77
<b>Hirsova</b>	151
<b>Iasi</b>	226
<b>Lugoj</b>	244
<b>Mehadia</b>	241
<b>Neamt</b>	234
<b>Oradea</b>	380
<b>Pitesti</b>	98
<b>Rimnicu Vilcea</b>	193
<b>Sibiu</b>	253
<b>Timisoara</b>	329
<b>Urziceni</b>	80
<b>Vaslui</b>	199
<b>Zerind</b>	374

# Tìm kiếm h้าu ăn

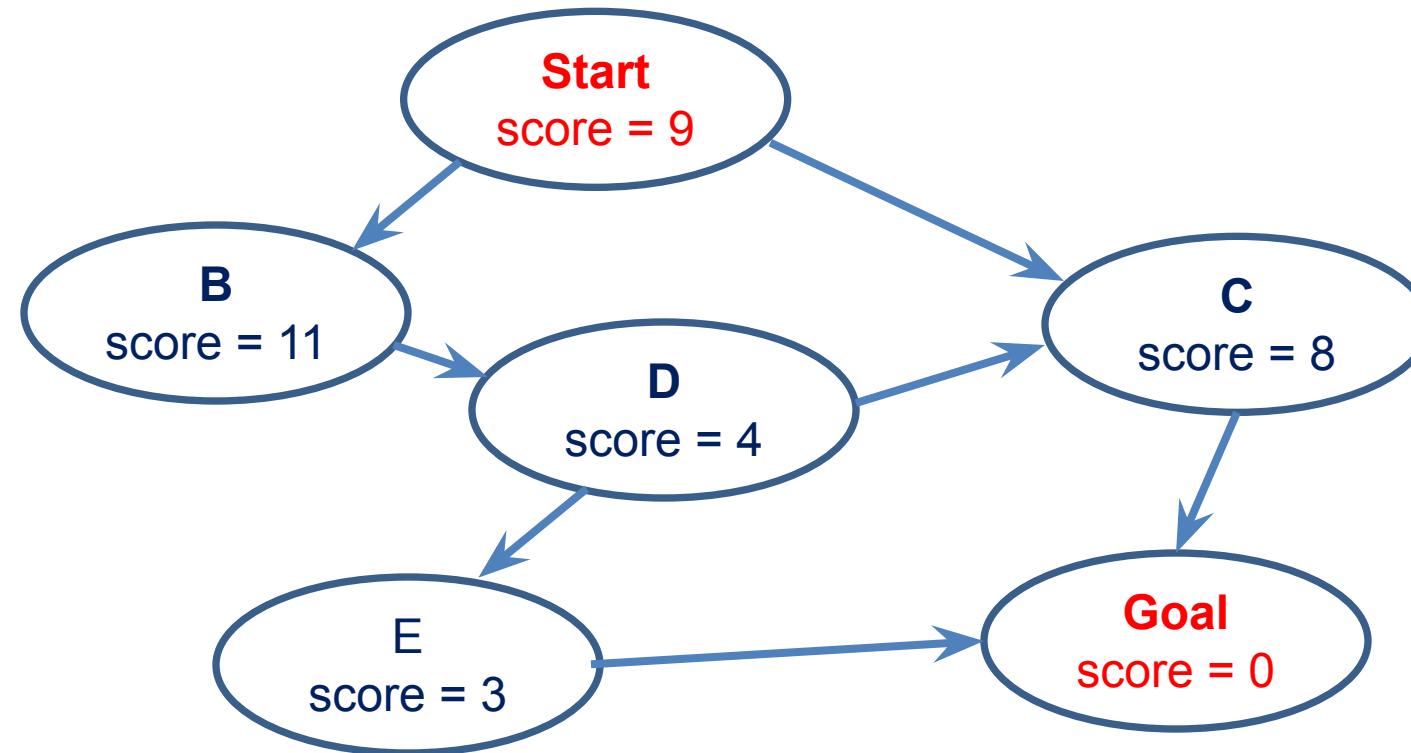
- Initial State = Arad
- Goal State = Bucharest

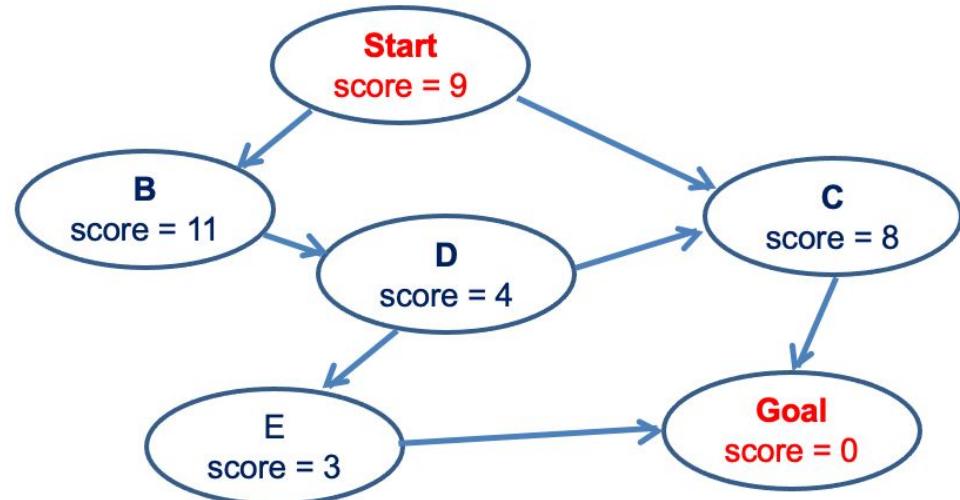


Straight-line distance  
to Bucharest

<b>Arad</b>	366
<b>Bucharest</b>	0
<b>Craiova</b>	160
<b>Dobreta</b>	242
<b>Eforie</b>	161
<b>Fagaras</b>	178
<b>Giurgiu</b>	77
<b>Hirsova</b>	151
<b>Iasi</b>	226
<b>Lugoj</b>	244
<b>Mehadia</b>	241
<b>Neamt</b>	234
<b>Oradea</b>	380
<b>Pitesti</b>	98
<b>Rimnicu Vilcea</b>	193
<b>Sibiu</b>	253
<b>Timisoara</b>	329
<b>Urziceni</b>	80
<b>Vaslui</b>	199
<b>Zerind</b>	374

Tìm kiếm tốt nhất đầu tiên, tìm kiếm rộng, tìm kiếm sâu  
để tìm kiếm trạng thái “Goal” từ trạng thái “Start”

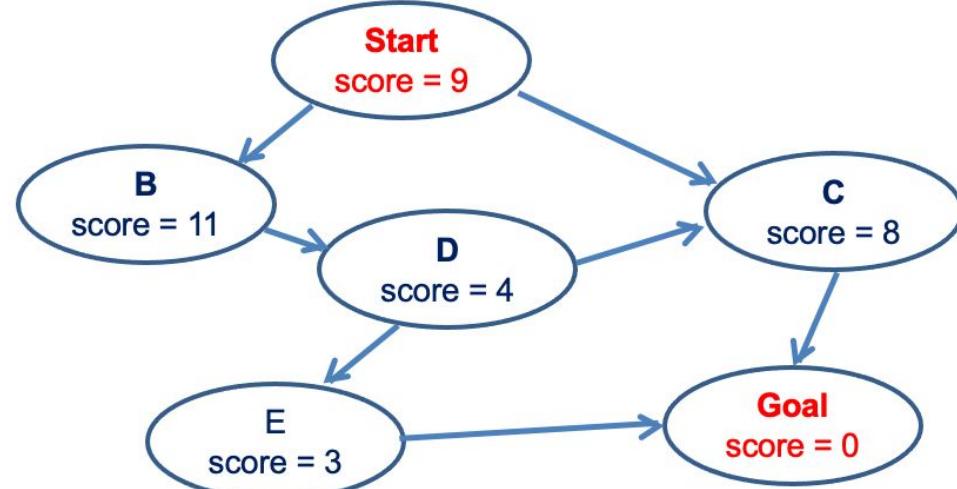




## BFS - Tìm kiếm theo chiều rộng

<u>Step#</u>	<u>OPEN</u>	<u>CLOSED</u>	<u>X CHILDREN</u>	<u>Remaining CHILDREN</u>
1	{ S }	{ }	S	{ S, B, C }
2	{ B, C }	{ S }	B	{ D }
3	{ C, D }	{ S, B }	C	{ G }
4	{ D, G }	{ S, B, C }	D	{ C, E }
5	{ G, E }	{ S, B, C, D }	G	<b>DONE</b>

- note we check for **GOALS** upon removal from OPEN in order to get SHORTEST PATHS in later algo's

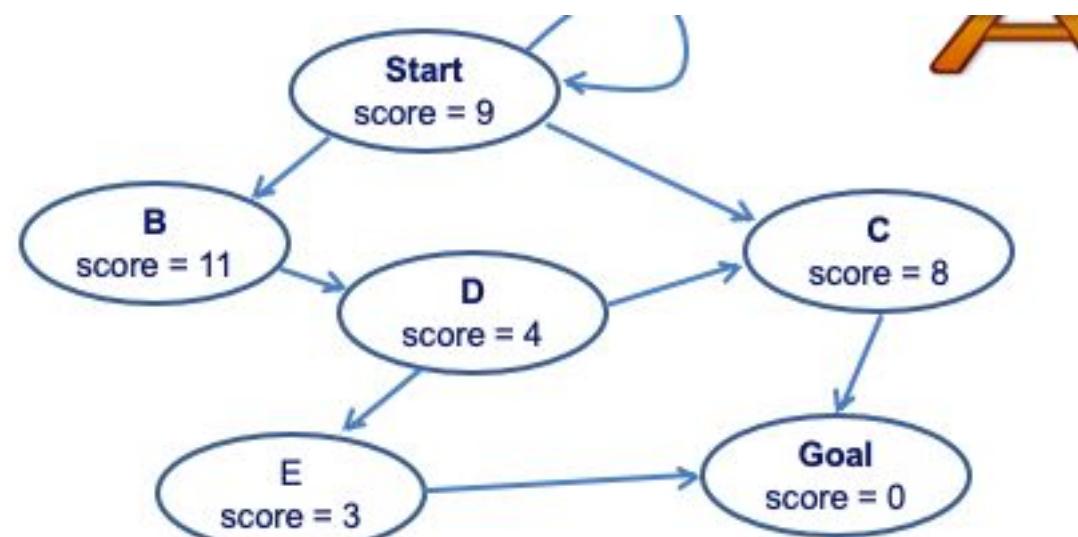


## DFS Tìm kiếm theo chiều sâu

Step#	OPEN	CLOSED	X	CHILDREN	Remaining CHILDREN
1	{ S }	{ }	S	{ S <sup>S</sup> , B <sup>S</sup> , C <sup>S</sup> }	{ B <sup>S</sup> , C <sup>S</sup> }
2	{ B <sup>S</sup> , C <sup>S</sup> }	{ S }	B <sup>S</sup>	{ D <sup>B</sup> }	{ D <sup>B</sup> }
3	{ D <sup>B</sup> , C <sup>S</sup> }	{ S, B <sup>S</sup> }	D <sup>B</sup>	{ C <sup>D</sup> , E <sup>D</sup> }	{ E <sup>D</sup> }
4	{ E <sup>D</sup> , C <sup>S</sup> }	{ S, B <sup>S</sup> , D <sup>B</sup> }	E <sup>D</sup>	{ G <sup>E</sup> }	{ G <sup>E</sup> }
5	{ G <sup>E</sup> , C <sup>S</sup> }	{ S, B <sup>S</sup> , D <sup>B</sup> , E <sup>D</sup> }	G <sup>E</sup>	<b>DONE</b>	

## BEST

<u>Step#</u>	<u>OPEN</u>	<u>CLOSED</u>	<u>X</u>	<u>CHILDREN</u>	<u>Remaining CHILDREN</u>
1	{ S <sub>9</sub> }	{ }	S <sub>9</sub>	{ S <sub>9</sub> , B <sub>11</sub> , C <sub>8</sub> }	{ B <sub>11</sub> , C <sub>8</sub> }
2	{ C <sub>8</sub> , B <sub>11</sub> }	{ S <sub>9</sub> }	C <sub>8</sub>	{ G <sub>0</sub> }	{ G <sub>0</sub> }
3	{ G <sub>0</sub> , B <sub>11</sub> }	{ S <sub>9</sub> , C <sub>8</sub> }	G <sub>0</sub>	DONE	



# Giải thuật A\*

- Giải thuật A\* là một trường hợp đặc biệt Best first search (việc cập nhật lại đường đi dựa trên giá trị  $g(n)$  thay vì dựa trên giá trị  $f(n)$  tổng quát)
  - $f(n) = g(n) + h(n)$
  - $h(n)$  phụ thuộc vào trạng thái  $n$  nên  $f(n)$  chỉ thay đổi khi  $g(n)$  thay đổi hay nói cách khác khi ta tìm được một đường đi mới đến  $n$  tốt hơn đường đi cũ  $\Rightarrow$  cập nhật lại  $g$  khi đường đi mới tốt hơn)
- Mỗi trạng thái  $n$  tùy ý sẽ gồm 4 yếu tố ( $g(n)$ ,  $h(n)$ ,  $f(n)$ ,  $cha(n)$ )

Cha( $n$ ) là nút cha của nút đang xét  $n$

```
PNode AStarSearch(PState init_state) {
    PNode root = new Node();
    root->state = init_state;
    root->parent = NULL;
    root->f = 0;
    frontier.insert(root);
    explored.clear();
    while (!empty(frontier)) {
        //Lấy 1 nút từ đường biên có f(n) nhỏ nhất
        //và loại bỏ nó ra khỏi đường biên
        Node* node = frontier.pop();
        if (node là nút mục tiêu)
            return node;
        insert(node, explored);
```

```
        for (child là nút con của node) {
            child->g = node->g + stepCost(node, child);
            child->h = estimate(child->state);
            child->f = child->g + child->h;
            if (child->state không thuộc frontier và
                child->state không thuộc explored) {
                child->parent = node;
                frontier.insert(child);
            } else if (child->state nằm trong đường biên
                       và có g lớn hơn child->g)
                (*) Thay thế nút nằm trên đường biên
                      bằng child
            }
        }
    return NULL; //Thất bại, không tìm thấy lời giải
}
```

## Mã giả giải thuật A\*

```
g(no)=0; f(no)=h(no);  
open:=[no]; closed:=[];  
while open<>[] do
```

    loại n bên trái của open và đưa n vào closed;

**if** (n là một đích) **then** thành công, thoát

**else**

        Sinh các con m của n;

**For** m thuộc con(n) **do**

            g(m)=g(n)+c[n,m];

**If** m không thuộc open hay closed **then**

                f(m)=g(m)+h(m); cha(m)=n; Bỏ m vào open;

**If** m thuộc open (tồn tại m' thuộc open, sao cho m=m') **then**

**If** g(m)<g(m') **then** g(m')=g(m); f(m')=g(m')+h(m'); Cha(m')=n;

**If** m thuộc closed (tồn tại m' thuộc closed, sao cho m=m') **then**

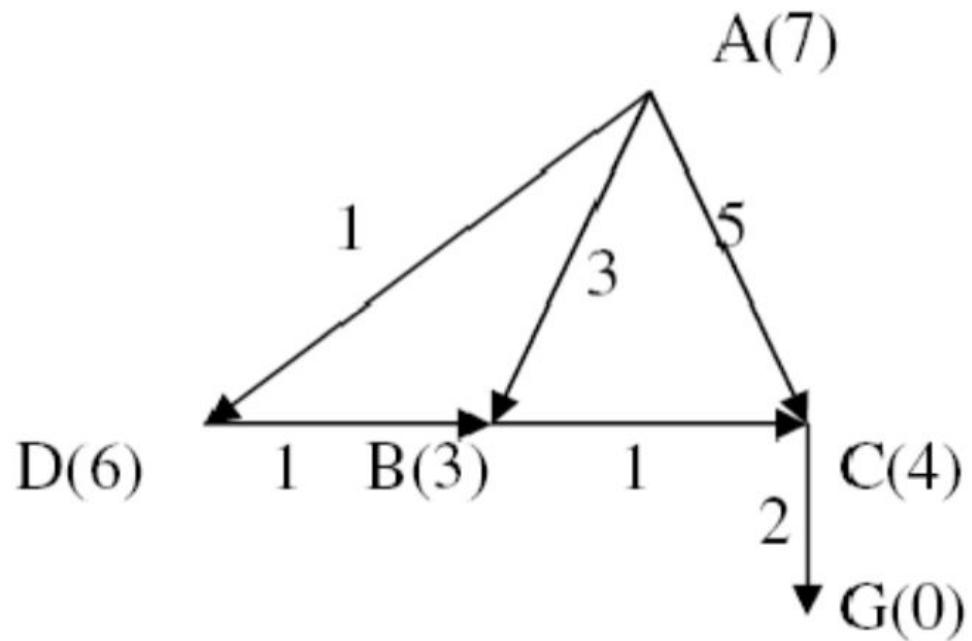
**If** g(m)<g(m') **then** f(m)=g(m)+h(m); cha(m)=n;

                Đưa m vào open; loại m' khỏi closed;

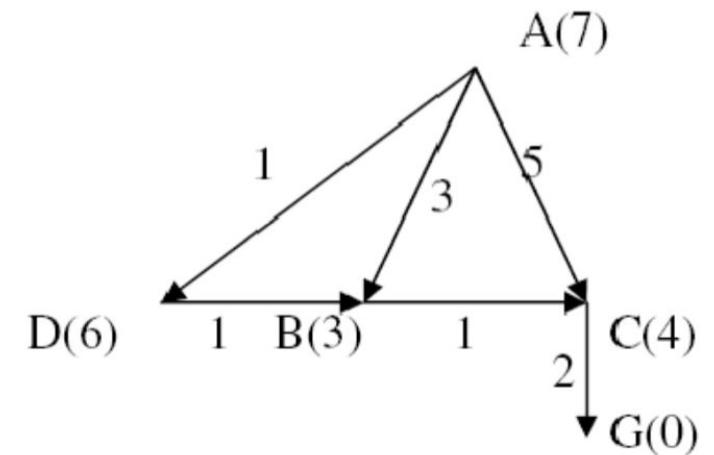
Sắp xếp open để t.thái tốt nhất nằm bên trái;

# Giải thuật A\*

- Sử dụng giải thuật A\* để tìm đường đi từ A đến G với giá trị  $g(n)$  được ghi trên cạnh của đồ thị và  $h(n)$  ghi ngay đỉnh của đồ thị



- Mỗi trạng thái n tùy ý sẽ gồm bốn yếu tố ( $g(n)$ ,  $h(n)$ ,  $f(n)$ ,  $\text{cha}(n)$ )
- **Bước 1:**
  - $\text{Open} = \{ A(0,7,7,-) \} ; \quad \text{close} = \{ \}$
- **Bước 2:**
  - Các con của A: D, B, C
  - **Xét D**
    - $g(D) = g(A) + c[A,D] = 0 + 1 = 1$  (do đề bài cung cấp) ( $g(m) = g(n) + c[m,n]$ )
    - D không thuộc Open; Close
      - Tính giá trị  $f(D) = g(D) + h(D) = 1 + 6 = 7$
      - Cập nhật cha của D: A
      - Đưa D vào open:  $D(1,6,7,A)$
  - **Xét B**



## Bước 2:

- Các con của A: D,B,C

- ....

- Xét B

- $g(B) = g(A) + c[A,B] = 0 + 3 = 3$  (do đề bài cung cấp) ( $g(m) = g(r, \dots)$ )

- B không thuộc Open; Close

- Tính giá trị  $f(B) = g(B) + h(B) = 3 + 3 = 6$

- Cập nhật cha của B: A

- Đưa B vào open: B(3,3,6,A)

- Xét C

- $g(C) = g(A) + c[A,C] = 0 + 5 = 5$  (do đề bài cung cấp) ( $g(m) = g(n) + c[m,n]$ )

- C không thuộc Open; Close

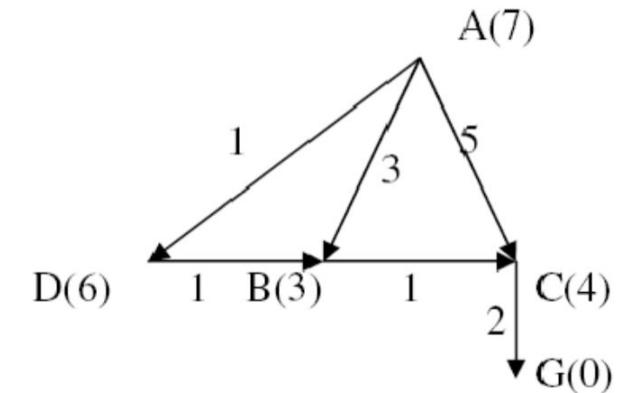
- Tính giá trị  $f(C) = g(C) + h(C) = 5 + 4 = 9$

- Cập nhật cha của C: A

- Đưa C vào open: C(5,4,9,A)

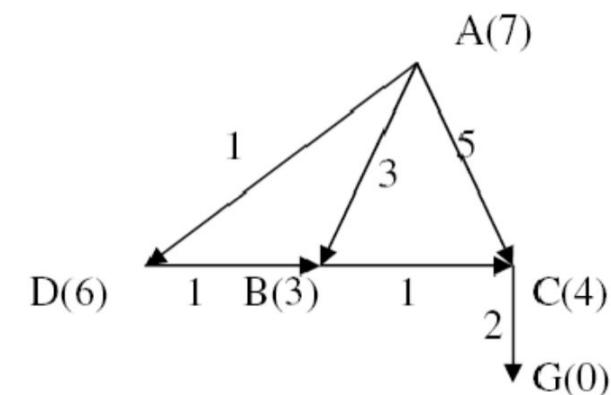
- Sắp xếp các phần tử trong open để trạng thái tốt nhất bên trái

$\text{Open}\{\textcolor{red}{B(3,3,6,A)}, D(1,6,7,A), C(5,4,9,A)\}$  ;  $\text{close}=\{A(0,7,7,-)\}$



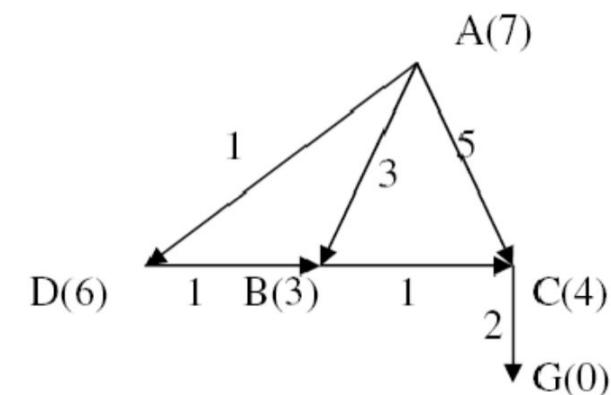
## Bước 3:

- Quay lại đâu vòng lặp while
- Lấy phần tử B ra khỏi Open đưa vào Close  
 $\text{Open}\{D(1,6,7,A), C(5,4,9,A)\}$  ;  $\text{close}=\{A(0,7,7,-), \textcolor{red}{B(3,3,6,A)}\}$
- Các con của B: C
- Xét C
  - $g(C) = g(B) + c[B,C] = 3 + 1 = 4$  ( $g(m) = g(n) + c[m,n]$ )
  - C thuộc Open;
    - So sánh giá trị  $g(C_n)$  hiện tại và  $g(C_o^{(4,4,8,B)})$  đã tồn tại trong Open
    - $G(C_o) = 5 > g(C_n) = 4 \Rightarrow$  cập nhật lại C
    - Tính giá trị  $f(C) = g(C) + h(C) = 4 + 4 = 8$
    - Cập nhật cha của C: B
    - Đưa C vào open:  $C(4,4,8,B)$
  - Sắp xếp các phần tử trong open để trạng thái tốt nhất bên trái  
 $\text{Open}\{D(1,6,7,A), C(4,4,8,B)\}$  ;  $\text{close}=\{A(0,7,7,-), \textcolor{red}{B(3,3,6,A)}\}$



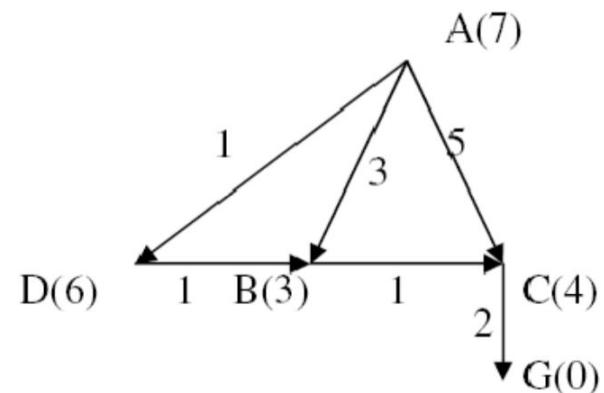
## Bước 4:

- Quay lại đâu vòng lặp while
- Lấy phần tử D ra khỏi Open đưa vào Close  
 $Open\{ C(4,4,8,B) \}; \quad close=\{A(0,7,7,-), B(3,3,6,A)\}, \textcolor{red}{D(1,6,7,A)}\}$
- Các con của D: B
- Xét B
  - $g(B) = g(D) + c[D,B] = 1 + 1 = 2 \quad ( g(m) = g(n) + c[m,n] )$
  - B thuộc Closed;
    - So sánh giá trị  $g(B_n)$  hiện tại và  $g(B_o)$  đã tồn tại trong Open
    - $G(B_o) = 3 > g(B_n) = 2 \Rightarrow$  cập nhật lại B
    - Tính giá trị  $f(B) = g(B) + h(B) = 2 + 3 = 5$
    - Cập nhật cha của B: D
    - Đưa B(2,3,5,D) vào open
    - Loại B(3,3,6,A) ra khỏi Closed
- Sắp xếp các phần tử trong open để trạng thái tốt nhất bên trái  
 $Open\{B(2,3,5,D), C(4,4,8,B)\} \quad ; close=\{A(0,7,7,-), \textcolor{red}{D(1,6,7,A)}\}$



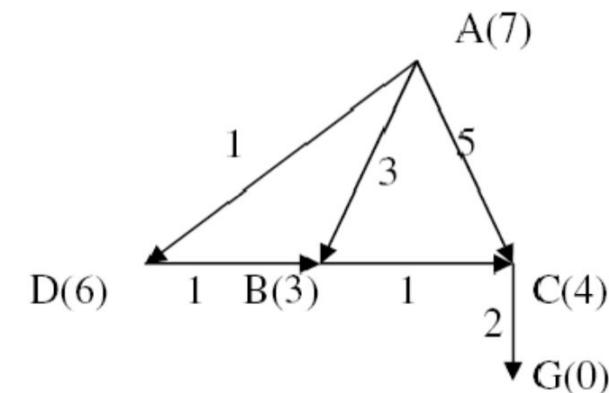
## Bước 5:

- Quay lại đâu vòng lặp while
- Lấy phần tử B ra khỏi Open đưa vào Close  
 $\text{Open}\{\mathbf{C}(4,4,8,B)\}; \quad \text{close}=\{A(0,7,7,-), D(1,6,7,A), \mathbf{B}(2,3,5,D)\}$
- Các con của B: C
- Xét C
  - $g(C) = g(B) + c[B,C] = 1 + 2 = 3 \quad (g(m) = g(n) + c[m,n])$
  - C thuộc Open;
    - So sánh giá trị  $g(C_n)$  hiện tại và  $g(C_o)$  đã tồn tại trong Open
    - $G(C_o) = 4 > g(C_n) = 3 \Rightarrow$  cập nhật lại C
    - Tính giá trị  $f(C) = g(C) + h(C) = 3 + 4 = 7$
    - Cập nhật cha của C: B
    - Đưa C vào open:  $C(3,4,7,B)$
  - Sắp xếp các phần tử trong open để trạng thái tốt nhất bên trái  
 $\text{Open}\{C(3,4,7,B)\} ; \text{close}=\{A(0,7,7,-), D(1,6,7,A), \mathbf{B}(2,3,5,D)\}$



## Bước 6:

- Quay lại đâu vòng lặp while
- Lấy phần tử C ra khỏi Open đưa vào Close  
 $Open\{ \}; close=\{A(0,7,7,-), D(1,6,7,A), B(2,3,5,D) , C(3,4,7,B)\}$
- Các con của C: G
- Xét G
  - $g(G) = g(C) + c[C,G] = 3 + 2 = 5$       ( $g(m) = g(n) + c[m,n]$ )
  - G không thuộc Open; Closed
    - Tính giá trị  $f(G) = g(G) + h(G) = 5 + 0 = 5$
    - Cập nhật cha của G: C
    - Đưa G vào open:  $G(5,0,5,C)$
- Sắp xếp các phần tử trong open để trạng thái tốt nhất bên trái  
 $Open\{G(5,0,5,C)\} ; close=\{A(0,7,7,-), D(1,6,7,A), B(2,3,5,D) \}$



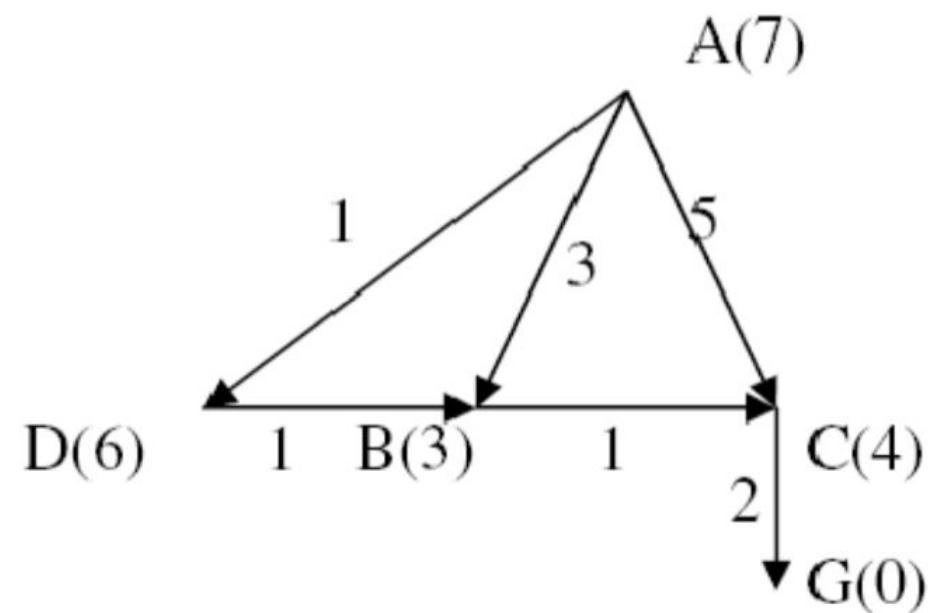
## Bước 6:

- Quay lại đầu vòng lặp while
- Lấy phần tử C ra khỏi Open đưa vào Close  
 $Open\{\}; \quad closed=\{A(0,7,7,-), D(1,6,7,A), B(2,3,5,D) , C(3,4,7,B), G(5,0,5,C)\}$
- Các **G(5,0,5,C)** là trạng thái đích  $\Rightarrow$  giải thuật dừng lại

Ta có đường đi

$closed=\{A(0,7,7,-), D(1,6,7,A), B(2,3,5,D) ,$

$A \Rightarrow D \Rightarrow B \Rightarrow C \Rightarrow G$



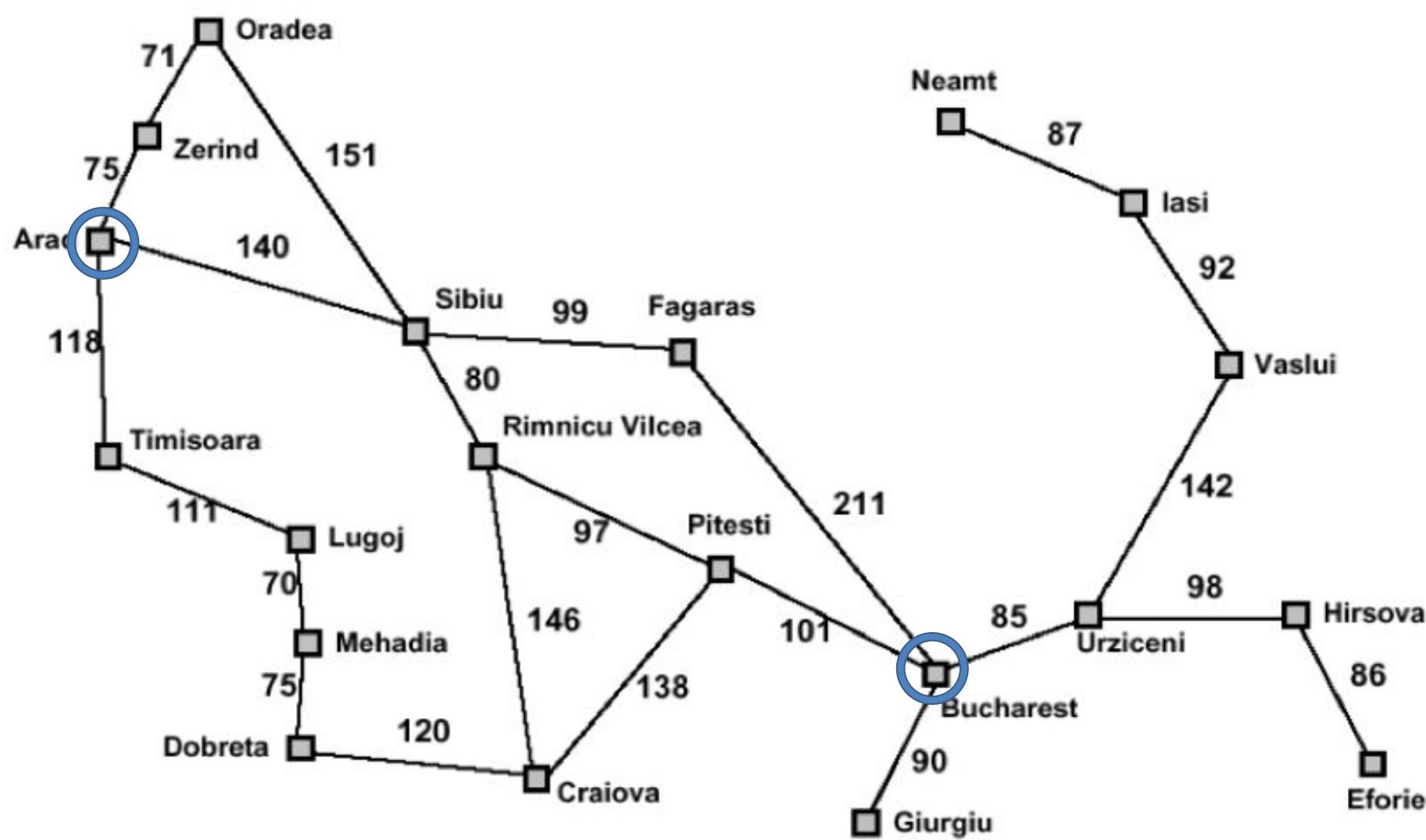
# Tìm kiếm heuristic: giải thuật A\*

Bài toán tìm đường:

- Thành phố xuất phát: Arad
- Thành phố đích: Bucharest
- Các cạnh biểu diễn đường nối trực tiếp giữa hai thành phố, các con số ghi trên các cạnh là chi phí đi giữa hai thành phố.
- Cột bên phải là khoảng cách Euclid từ các thành phố đến thành phố đích Bucharest.

Sử dụng phương pháp tìm kiếm A\* ( hàm ước lượng  $f(n) = g(n) + h(n)$ , với  $g(n)$  là chi phí từ thành phố xuất phát đến  $n$  và  $h(n)$  là khoảng cách Euclid từ  $n$  đến đích)

# Tìm kiếm heuristic



## Mã giả giải thuật A\*

```
g(no)=0; f(no)=h(no);  
open:=[no]; closed:=[];  
while open<>[] do
```

    loại n bên trái của open và đưa n vào closed;

**if** (n là một đích) **then** thành công, thoát

**else**

        Sinh các con m của n;

**For** m thuộc con(n) **do**

            g(m)=g(n)+c[n,m];

**If** m không thuộc open hay closed **then**

                f(m)=g(m)+h(m); cha(m)=n; Bỏ m vào open;

**If** m thuộc open (tồn tại m' thuộc open, sao cho m=m') **then**

**If** g(m)<g(m') **then** g(m')=g(m); f(m')=g(m')+h(m'); Cha(m')=n;

**If** m thuộc closed (tồn tại m' thuộc closed, sao cho m=m') **then**

**If** g(m)<g(m') **then** f(m)=g(m)+h(m); cha(m)=n;

                Đưa m vào open; loại m' khỏi closed;

Sắp xếp open để t.thái tốt nhất nằm bên trái;

- Mỗi trạng thái n tùy ý sẽ gồm: ( $g(n)$ ,  $h(n)$ ,  $f(n)$ , cha( $n$ ))

### Bước 1:

- $Open = \{Arad(0, 366, 366, -)\}; close = \{\}$

### Bước 2:

- Các con của Arad: Timisoara, Sibiu, Zerind
- Xét Timisoara

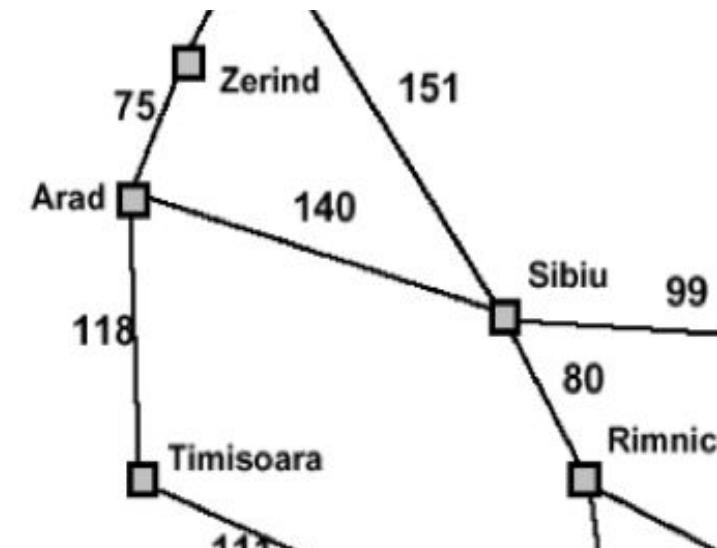
■  $g(Timisoara) = g(Arad) + c[Arad, Timisoara] = 0 + 118 = 118$  (do đề bài  
cung cấp) ( $g(m) = g(n) + c[m, n]$ )

■ Timisoara không thuộc Open; Close

- Tính giá trị  $f(Timisoara) = g(Timisoara) + h(Timisoara)$   
 $= 118 + 329 = 447$

- Cập nhật cha của Timisoara : Arad

- Đưa Timisoara vào open: Timisoara(118, 329, 447, Arad)



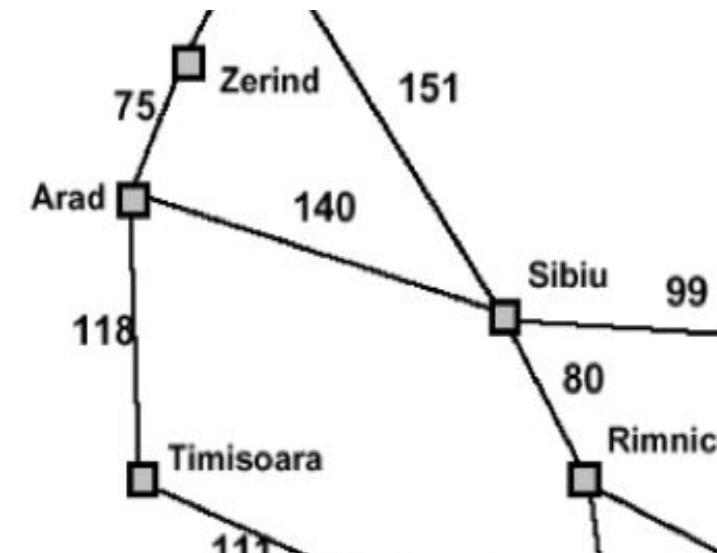
## Bước 2:

- .....
- Xét Sibiu

- $g(\text{Sibiu}) = g(\text{Arad}) + c[\text{Arad}, \text{Sibiu}]$   
 $= 0 + 140 = 140$  (do đề bài cung cấp)  
( $g(m) = g(n) + c[m,n]$ )

- Sibiu không thuộc Open; Close

- Tính giá trị  $f(\text{Sibiu}) = g(\text{Sibiu}) + h(\text{Sibiu}) = 140 + 253 = 393$
  - Cập nhật cha của Sibiu : Arad
  - Đưa Sibiu vào open: Sibiu (140,253,393,Arad)



## ■ Bước 2:

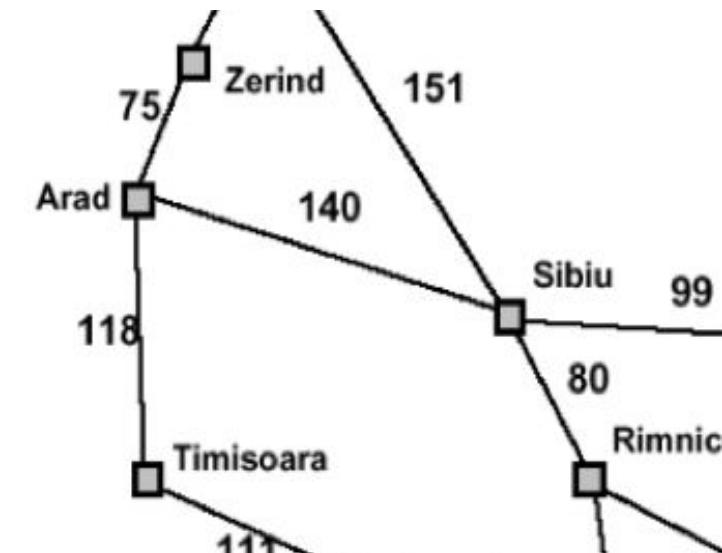
- .....
- Xét Zerind

■  $g(\text{Zerind}) = g(\text{Arad}) + c[\text{Arad}, \text{Zerind}]$   
 $= 0 + 75 = 75$  (do đề bài cung cấp)  
 $(g(m) = g(n) + c[m,n])$

■ Zerind không thuộc Open; Close

- Tính giá trị  $f(\text{Zerind}) = g(\text{Zerind}) + h(\text{Zerind}) = 75 + 374 = 449$
- Cập nhật cha của Zerind : Arad
- Đưa Zerind vào open:  $\text{Zerind}(75, 374, 449, \text{Arad})$
- Sắp xếp các phần tử trong open để trạng thái tốt nhất bên trái

$\text{Open}\{\text{Sibiu } (140, 253, 393, \text{Arad}), \text{Timisoara}(118, 329, 447, \text{Arad}), \text{Zerind}(75, 374, 449, \text{Arad})\};$   
 $\text{Closed} = \{ \text{Arad } (0, 366, 366, -) \}$



### Bước 3

- Quay lại đâu vòng lặp while
- Lấy phần tử Sibiu ra khỏi Open đưa vào Closed

Open{Timisoara(118,329,447,Arad), Zerind(75,374,449,Arad)};

Closed = { Arad (0,366,366,-), Sibiu (140,253,393,Arad) }

- Các con của Sibiu : Rimnicu Vilces, Fagaras, Arad, Oradea
- Xét Rimnicu

■  $g(\text{Rimnicu}) = g(\text{Sibiu}) + c[\text{Sibiu}, \text{Rimnicu}] = 140 + 80 = 220$

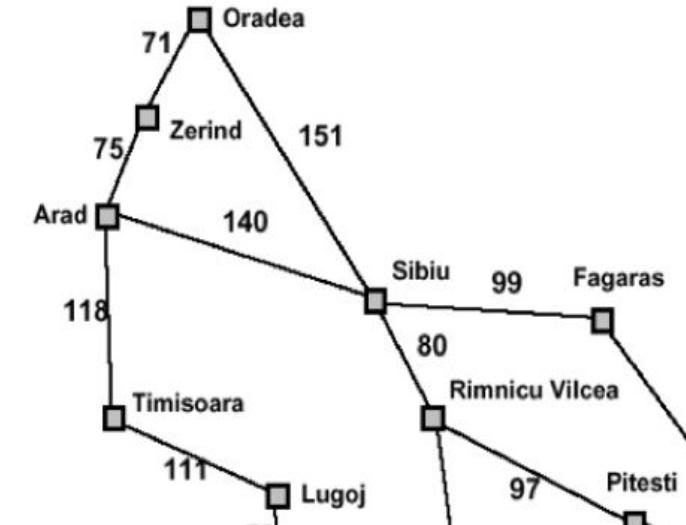
(  $g(m) = g(n) + c[m,n]$  )

■ Rimnicu không thuộc Open; Close

- Tính giá trị  $f(\text{Rimnicu}) = g(\text{Rimnicu}) + h(\text{Rimnicu})$   
 $= 220 + 193 = 413$

- Cập nhật cha của Rimnicu : Sibiu

- Đưa Rimnicu vào open: Rimnicu(220,193,413,Sibiu)



### Bước 3

- .....
- Các con của Sibiu : Rimnicu Vilces, Fagaras, Arad, O
- Xét Fagaras

■  $g(Fagaras) = g(Sibiu) + c[Sibiu, Fagaras] = 140 + 99 = 239$  (  $g(m) = g(n) + c[m,n]$  )

■ Fagaras không thuộc Open; Close

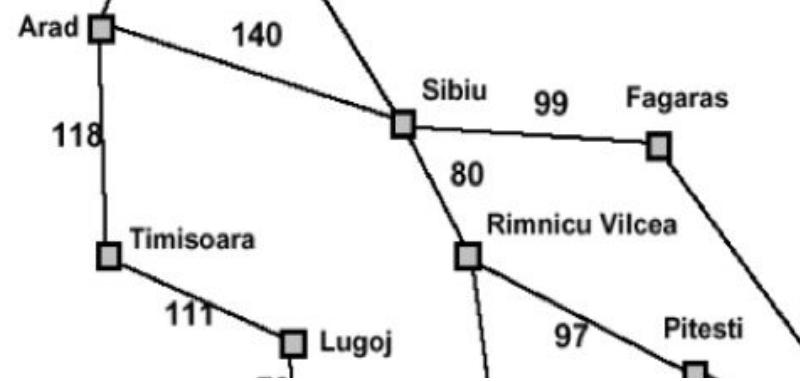
- Tính giá trị  $f(Fagaras) = g(Fagaras) + h(Fagaras) = 239 + 178 = 417$

- Cập nhật cha của Fagaras : Sibiu

- Đưa Fagaras vào open: Fagaras(239,178,417,Sibiu)

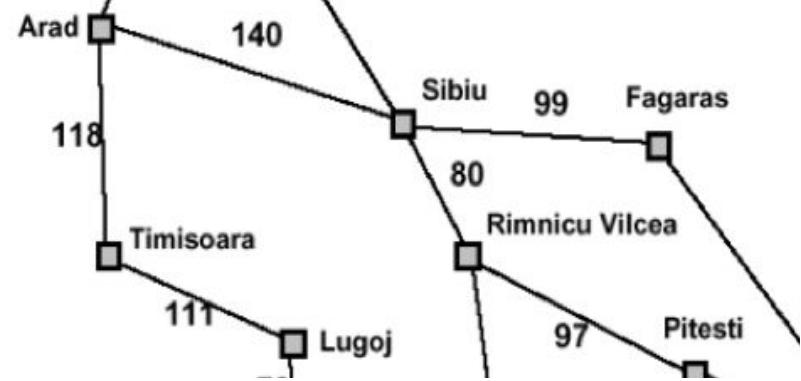
- Sắp xếp các phần tử trong open để trạng thái tốt nhất bên trái  
 $Open\{Rimnicu(220,193,413,Sibiu), Fagaras(239,178,417,Sibiu),$   
 $Timisoara(118,329,447,Arad), Zerind(75,374,449,Arad)\};$

$$Closed = \{ Arad (0,366,366,-), Sibiu (140,253,393,Arad) \}$$



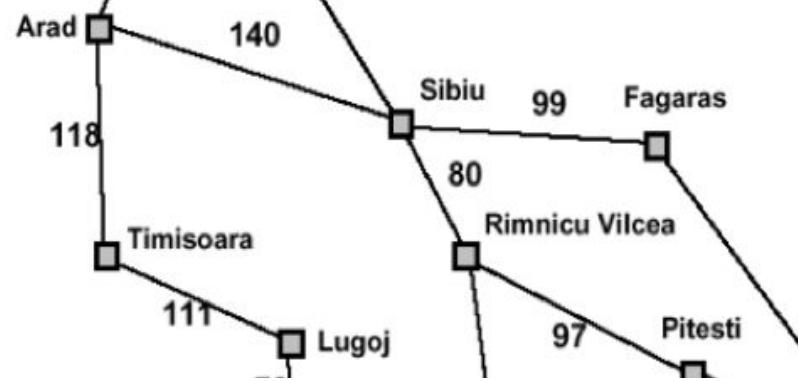
### Bước 3

- .....
- Các con của Sibiu : Rimnicu, Fagaras, Arad, Oradea
- Xét Oradea
  - $g(\text{Oradea}) = g(\text{Sibiu}) + c[\text{Sibiu}, \text{Oradea}] = 140 + 151 = 291$  (  $g(m) = g(n) + c[m,n]$  )
  - Oradea không thuộc Open; Close
    - Tính giá trị  $f(\text{Oradea}) = g(\text{Oradea}) + h(\text{Oradea}) = 291 + 380 = 671$
    - Cập nhật cha của Oradea : Sibiu
    - Đưa Oradea vào open:  $\text{Oradea}(291, 380, 671, \text{Sibiu})$
- Sắp xếp các phần tử trong open để trạng thái tốt nhất bên trái  
 $\text{Open}\{\text{Rimnicu}(220, 193, 413, \text{Sibiu}), \text{Fagaras}(239, 178, 417, \text{Sibiu}), \text{Timisoara}(118, 329, 447, \text{Arad}), \text{Zerind}(75, 374, 449, \text{Arad}), \text{Oradea}(291, 380, 671, \text{Sibiu})\};$   
 $\text{Closed} = \{ \text{Arad } (0, 366, 366, -), \text{Sibiu } (140, 253, 393, \text{Arad}) \}$



### Bước 3

- .....
- Các con của Sibiu : Rimnicu, Fagaras, Arad, Oradea
- Xét Arad
  - $g(\text{Arad}) = g(\text{Sibiu}) + c[\text{Sibiu}, \text{Arad}] = 140 + 140 = 280$  (  $g(m) = g(n) + c[m, n]$  )
  - Arad thuộc Closed
    - $G(\text{Arad}) = 280 > G(\text{Arad}')$   $\Rightarrow$  ko làm gì cả (ko đưa vào open hay closed)
- Sắp xếp các phần tử trong open để trạng thái tốt nhất bên trái  
 $\text{Open}\{\text{Rimnicu}(220, 193, 413, \text{Sibiu}), \text{Fagaras}(239, 178, 417, \text{Sibiu}),$   
 $\text{Timisoara}(118, 329, 447, \text{Arad}), \text{Zerind}(75, 374, 449, \text{Arad}), \text{Oradea}(291, 380, 671, \text{Sibiu})\};$   
 $\text{Closed} = \{ \text{Arad } (0, 366, 366, -), \text{ Sibiu } (140, 253, 393, \text{Arad}) \}$



## Bước 4

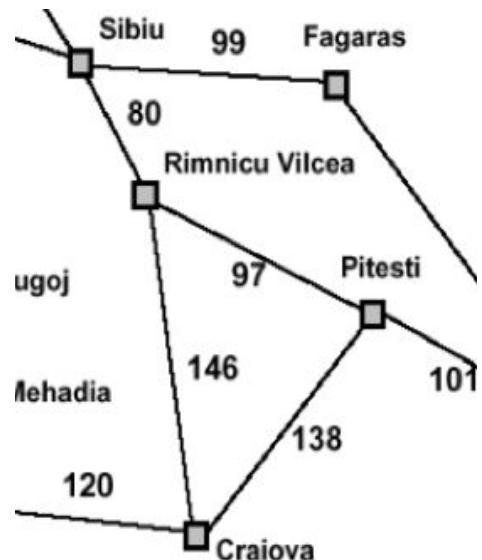
- Quay lại đầu vòng lặp while
- Lấy phần tử Rimnicu ra khỏi Open đưa vào Closed

Open{Fagaras(239,178,417,Sibiu), Timisoara(118,329,447,Arad),  
Zerind(75,374,449,Arad), Oradea(291,380,671,Sibiu)};

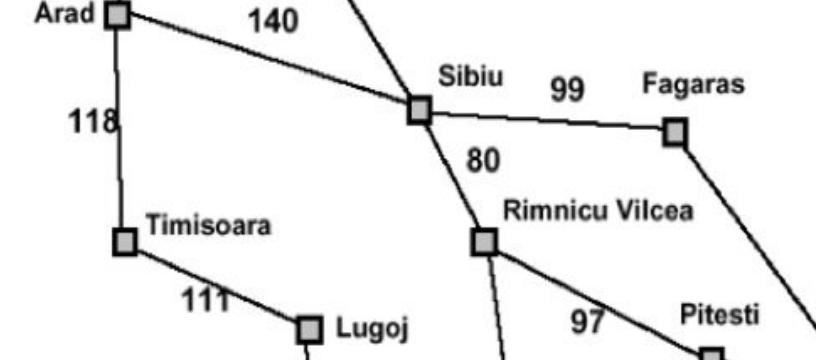
Closed = { Arad (0,366,366,-), Sibiu (140,253,393,Arad), Rimnicu(220,193,413,Sibiu) }

- Các con của Rimnicu : Craiova, Pitesti, Sibiu
- Xét Craiova

- $g(\text{Craiova}) = g(\text{Rimnicu}) + c[\text{Rimnicu}, \text{Craiova}] = 220 + 146 = 366$
- Craiova không thuộc Open; Close
  - Tính giá trị  $f(\text{Craiova}) = g(\text{Craiova}) + h(\text{Craiova}) = 366 + 160 = 526$
  - Cập nhật cha của Craiova là Rimnicu
  - Đưa Craiova vào open: Craiova(366,160,526, Rimnicu )



## Bước 4

- ....
  - Các con của Rimnicu : Craiova, Pitesti, Sibiu
  - Xét Pitesti
    - $g(\text{Pitesti}) = g(\text{Rimnicu}) + c[\text{Rimnicu}, \text{Pitesti}] = 220 + 97 = 317$
    - Pitesti không thuộc Open; Close
      - Tính giá trị  $f(\text{Pitesti}) = g(\text{Pitesti}) + h(\text{Pitesti}) = 317 + 98 = 415$
      - Cập nhật cha của Pitesti là Rimnicu
      - Đưa Pitesti vào open: Pitesti (317, 98, 415, Rimnicu)
    - Sắp xếp các phần tử trong open để trạng thái tốt nhất bên trái
- Open{Pitesti (317, 98, 415, Rimnicu), Fagaras(239, 178, 417, Sibiu), Timisoara(118, 329, 447, Arad), Zerind(75, 374, 449, Arad), Craiova(366, 160, 526, Rimnicu), Oradea(291, 380, 671, Sibiu)};
- Closed = { Arad (0, 366, 366, -), Sibiu (140, 253, 393, Arad), Rimnicu(220, 193, 413, Sibiu) }
- 

## ■ Bước 4

- ....
- Các con của Rimnicu : Craiova, Pitesti, Sibiu
- Xét Sibiu

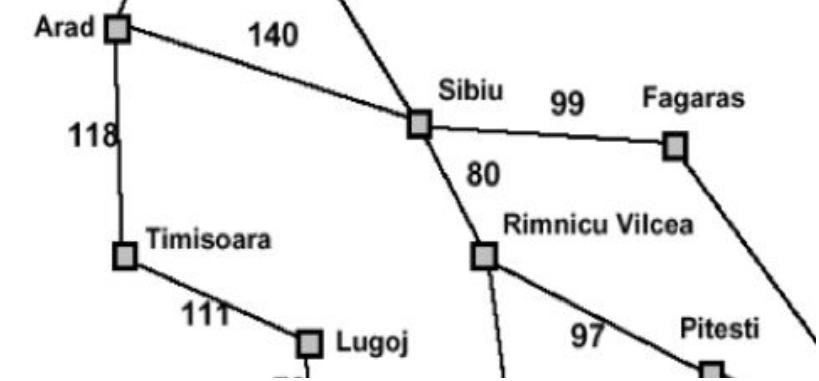
■  $g(\text{Sibiu}) = g(\text{Rimnicu}) + c[\text{Rimnicu}, \text{Sibiu}] = 220 + 80 = 400$

### ■ Sibiu thuộc Close

- Tính giá trị  $g(\text{Sibiu}) = 400 > g(\text{Sibiu}') = 140$
- $\Rightarrow$  không làm gì cả (ko đưa vào open hay closed)
- 
- Sắp xếp các phần tử trong open để trạng thái tốt nhất bên trái

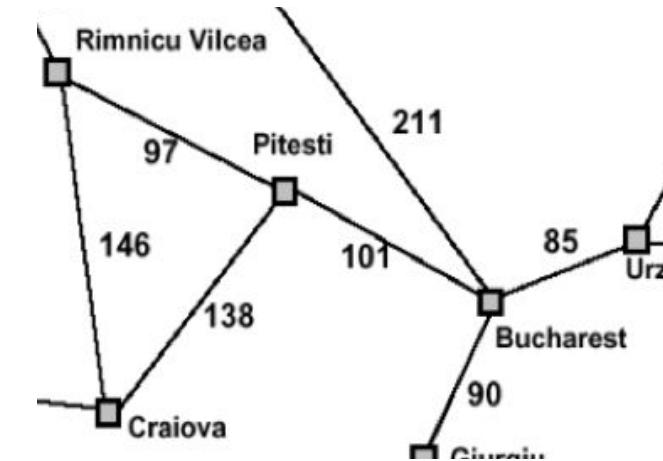
Open{Pitesti (317,98,415, Rimnicu ), Fagaras(239,178,417,Sibiu), Timisoara(118,329,447,Arad), Zerind(75,374,449,Arad), Craiova(366,160,526, Rimnicu ), Oradea(291,380,671,Sibiu)};

Closed = { Arad (0,366,366,-), Sibiu (140,253,393,Arad), Rimnicu(220,193,413,Sibiu )}



## Bước 5

- Quay lại đâu vòng lặp while
- Lấy phần tử Pitesti ra khỏi Open đưa vào Closed  
 $\text{Open}\{\text{Fagaras}(239,178,417,\text{Sibiu}), \text{Timisoara}(118,329,447,\text{Arad}),$   
 $\text{Zerind}(75,374,449,\text{Arad}), \text{Craiova}(366,160,526, \text{Rimnicu}), \text{Oradea}(291,380,671,\text{Sibiu})\};$
- $\text{Closed} = \{ \text{Arad } (0,366,366,-), \text{Sibiu } (140,253,393,\text{Arad}), \text{Rimnicu}(220,193,413,\text{Sibiu}),$   
 $\text{Pitesti } (317,98,415, \text{Rimnicu}) \}$
- Các con của Pitesti : Craiova, Bucharest, Rimnicu
- Xét Craiova
  - $\text{g}(\text{Craiova}) = \text{g}(\text{Pitesti}) + c[\text{Pitesti}, \text{Craiova}] = 317 + 138 = 455$
  - Craiova thuộc Open;
  - $\text{g}(\text{Craiova})=455 > \text{g}(\text{Craiova}')= 368 \Rightarrow$  Không làm gì cả



## Bước 5

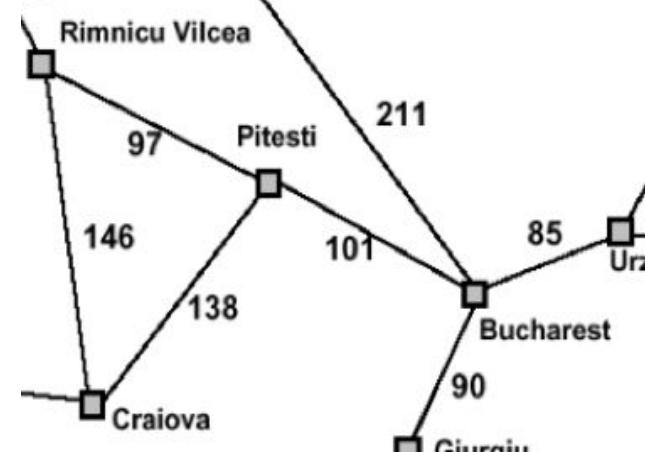
- Quay lại đầu vòng lặp while
- Lấy phần tử Pitesti ra khỏi Open đưa vào Closed

*Open{Fagaras(239,178,417,Sibiu),Timisoara(118,329,447,Arad),  
Zerind(75,374,449,Arad), Craiova(366,160,526, Rimnicu ), Oradea(291,380,671,Sibiu)};*

*Closed = { Arad (0,366,366,-), Sibiu (140,253,393,Arad), Rimnicu(220,193,413,Sibiu),  
Pitesti (317,98,415, Rimnicu ) }*

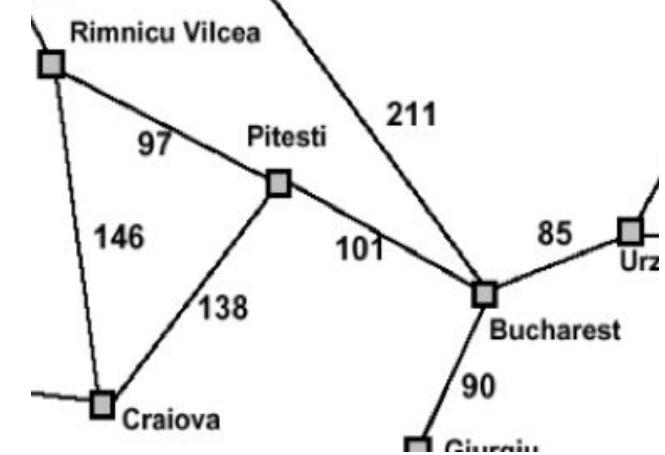
Các con của Pitesti : Craiova, Bucharest, Rimnicu

- Xét Bucharest
  - $g(\text{Bucharest}) = g(\text{Pitesti}) + c[\text{Pitesti}, \text{Bucharest}] = 317 + 101 = 418$
  - Bucharest **không thuộc Open; Closed**
    - Tính giá trị  $f(\text{Bucharest}) = g(\text{Bucharest}) + h(\text{Bucharest}) = 418 + 0 = 418$
    - Cập nhật cha của Bucharest : Pitesti
    - Đưa Bucharest vào open: Bucharest (418,0,418,Pitesti)



## ■ Bước 5

- ...
- Các con của Pitesti : Craiova, Bucharest, Rimnicu
- Xét Rimnicu
  - $g(\text{Rimnicu}) = g(\text{Pitesti}) + c[\text{Pitesti}, \text{Rimnicu}] = 317 + 138 = 455$
  - Rimnicu thuộc Closed
  - Xét  $g(\text{Rimnicu}) = 455 > g(\text{Rimnicu}') = 220$ 
    - $\Rightarrow$  Không làm gì cả



*Open*{Fagaras(239,178,417,Sibiu), Bucharest(418,0,418,Pitesti), Timisoara(118,329,447,Arad), Zerind(75,374,449,Arad), Craiova(368,160,528, Rimnicu), Oradea(291,380,671,Sibiu)};

*Closed* = { Arad (0,366,366,-), Sibiu (140,253,393,Arad), Rimnicu(220,193,413,Sibiu), Pitesti (317,98,415, Rimnicu) }

## ■ Bước 6

- Quay lại đâu vòng lặp while
- Lấy phần tử **Fagaras** ra khỏi Open đưa vào Closed

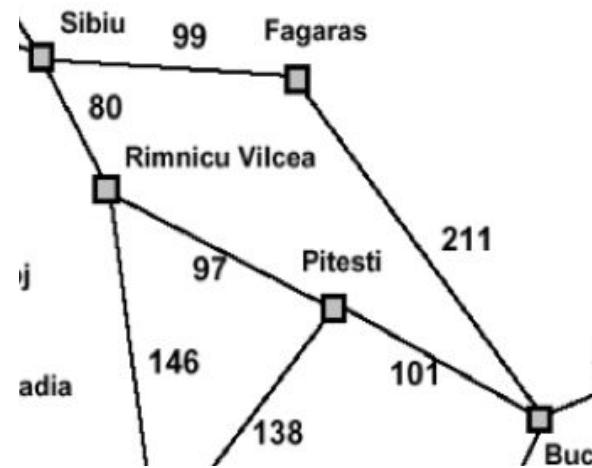
**Open**{Bucharest(418,0,418,Pitesti), Timisoara(118,329,447,Arad),

Zerind(75,374,449,Arad), Craiova(368,160,528, Rimnicu), Oradea(291,380,671, Sibiu)};

**Closed** = { Arad (0,366,366,-), Sibiu (140,253,393,Arad), Rimnicu(220,193,413,Sibiu), Pitesti (317,98,415, Rimnicu ), Fagaras(239,178,417,Sibiu) }

- Các con của **Fagaras** : Bucharest, Sibiu
- Xét **Sibiu**

- $g(\text{Sibiu}) = g(\text{Fagaras}) + c[\text{Fagaras}, \text{Sibiu}] = 239 + 99 = 338$
- Sibiu thuộc **Closed**;
- $g(\text{Sibiu}) = 338 > g(\text{Sibiu}') = 140 \Rightarrow$  Không làm gì cả



## ■ Bước 6

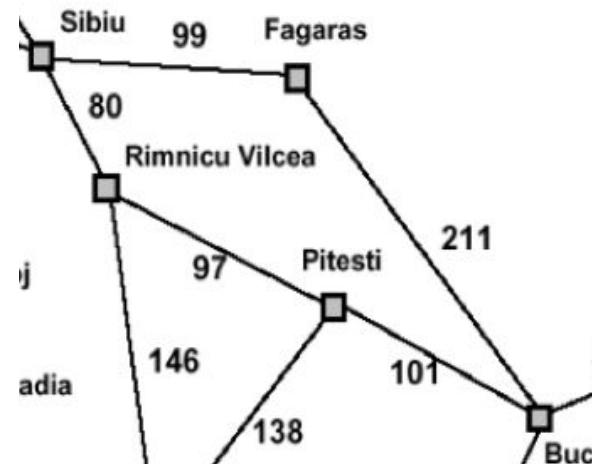
- Quay lại đầu vòng lặp while
- Lấy phần tử *Fagaras* ra khỏi *Open* đưa vào *Closed*

*Open*{Bucharest(418,0,418,Pitesti), Timisoara(118,329,447,Arad),

Zerind(75,374,449,Arad), Craiova(368,160,528, Rimnicu), Oradea(291,380,671, Sibiu)};

*Closed* = { Arad (0,366,366,-), Sibiu (140,253,393,Arad), Rimnicu(220,193,413,Sibiu), Pitesti (317,98,415, Rimnicu ), Fagaras(239,178,417,Sibiu) }

- Các con của *Fagaras* : Bucharest, Sibiu
- Xét Bucharest
  - $g(\text{Bucharest}) = g(\text{Fagaras}) + c[\text{Fagaras}, \text{Bucharest}] = 239 + 211 = 450$
  - Bucharest thuộc Open;
  - $g(\text{Bucharest}) = 450 > g(\text{Bucharest}') = 418 \Rightarrow$  Không làm gì cả



## ■ Bước 7

- Quay lại đâu vòng lặp while
- Lấy phần tử **Bucharest** ra khỏi *Open* đưa vào *Closed*

*Open*{Timisoara(118,329,447,Arad),

Zerind(75,374,449,Arad), Craiova(368,160,528, Rimnicu), Oradea(291,380,671, Sibiu)};

**Closed** = { Arad (0,366,366,-), Sibiu (140,253,393,Arad), Rimnicu(220,193,413,Sibiu), Pitesti (317,98,415, Rimnicu), Fagaras(239,178,417,Sibiu), **Bucharest(418,0,418,Pitesti)** }

- **Xét Bucharest là trạng thái đích, giải thuật dừng lại.**
- **Đường đi được tìm bằng cách truy ngược cha của nút gốc và các nút kế tiếp:**  
**Bucharest(418,0,418,Pitesti) => Pitesti (317,98,415, Rimnicu) =>**  
**Rimnicu(220,193,413,Sibiu) => Sibiu (140,253,393,Arad) => Arad (0,366,366,-)**

**Lưu ý:** Các trạng thái trong closed là trạng thái đã xét qua, tuy nhiên **không phải tất cả các trạng thái trong closed đều góp phần trong đường đi lời giải** (Ví dụ đỉnh Fagaras trong bài toán này không thuộc đường đi lời giải)

# Leo đồi (Hill climbing)

- **Ý tưởng:** Tìm kiếm trạng thái đích bằng cách hướng tới trạng thái tốt hơn trạng thái hiện tại (Leo lên đỉnh của một ngọn đồi)
- **Đặc điểm của giải thuật leo đồi:**
  - Trạng thái con tốt nhất sẽ được chọn cho bước tiếp theo
  - Không lưu giữ bất kỳ thông tin về các nút cha và anh em.
  - Quá trình tìm kiếm sẽ dừng lại khi gặp trạng thái đích hoặc trạng thái kế tiếp "xấu" hơn trạng thái đang xét ( $f_{\text{đang xét}} < f_{\text{trạng thái kế tiếp}}$ )
  - Sử dụng hàm đánh giá để đo tính tốt hơn của một trạng thái so với trạng thái khác

# Leo đồi (Hill climbing)

1. **Đánh giá trạng thái khởi đầu.** Nếu nó là trạng thái đích, thoát, nếu không xét nó như trạng thái hiện hành
2. **Lặp lại đến khi tìm thấy một lời giải hoặc đến khi không tìm thấy « toán tử » mới nào có thể áp dụng lên trạng thái hiện hành:**
  - a) *Chọn một toán tử chưa được áp dụng đối với trạng thái hiện hành, áp dụng nó để sinh ra một trạng thái mới NS*
  - b) *Đánh giá trạng thái mới NS*
    - i. Nếu NS là một trạng thái đích, return NS và thoát
    - ii. Nếu NS không là đích nhưng « tốt hơn » trạng thái hiện hành, lấy NS làm trạng thái hiện hành
    - iii. Nếu NS không tốt hơn trạng thái hiện hành, tiếp tục vòng lặp

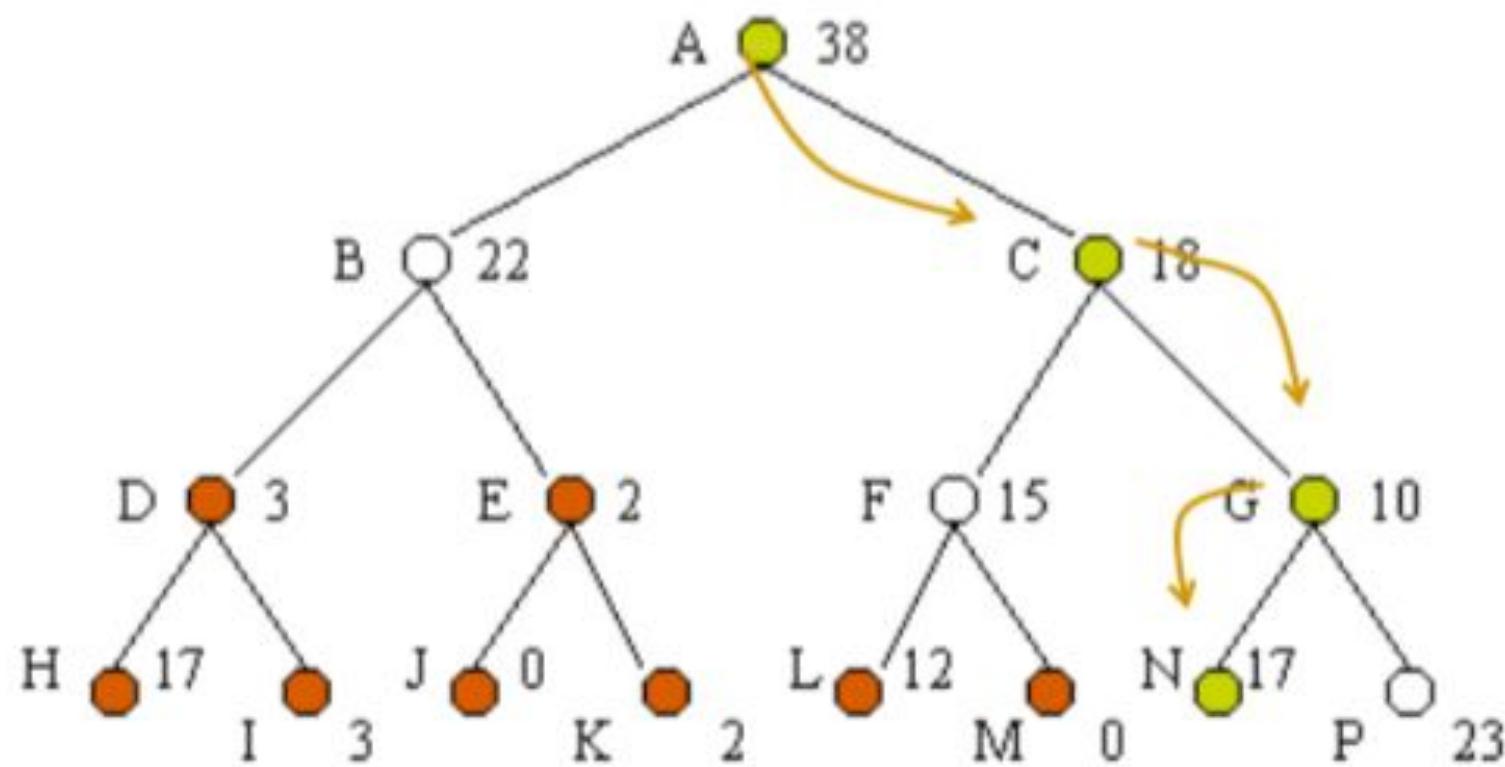
# Leo đồi (Hill climbing)

1. **Đánh giá trạng thái khởi đầu:** nếu nó là trạng thái đích thì thoát; ngược lại, gọi nó là trạng thái hiện hành
2. **Lặp lại:**
  - Áp dụng các toán tử để sinh ra tập các trạng thái con mới NS
  - Nếu tập này rỗng: **trả lời bài toán không có lời giải và thoát**
  - Nếu tập này khác rỗng:
    - Nếu có trạng thái con mới là trạng thái đích: **trả về trạng thái đích và thoát;**
    - Tìm trạng thái con mới tốt nhất;
      - Nếu nó không tốt hơn trạng thái hiện hành: **trả lời bài toán không có lời giải và thoát;** ngược lại: xem nó là trạng thái hiện hành

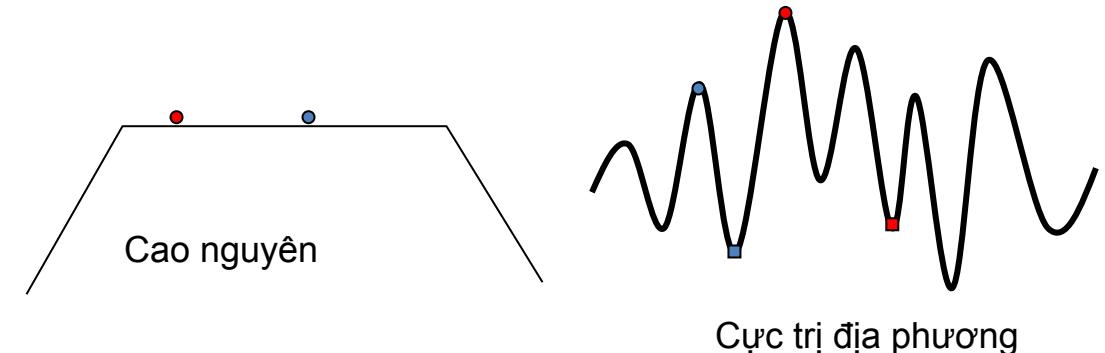
# Giải thuật leo đồi

```
PState hillClimbing(PState init_state) {  
    PNode current = new Node ;  
    current->state = init_state;  
    current->h = estimated_cost(current->state);  
    while (true) {  
        PNode neighbor = successor có giá trị cao nhất  
                               của current  
        if (neighbor.h ≤ current.h)  
            return current.state;  
        current = neighbor;  
    }  
}
```

# Tìm kiếm leo đồi (tt)



# Leo đồi (Hill climbing)



## ■ Hạn chế của tìm kiếm leo đồi:

- Không thể phục hồi lại từ những thất bại trong chiến lược của nó
- Hiệu quả hoạt động chỉ có thể được cải thiện trong một phạm vi giới hạn nào đó
- Lời giải tìm được không tối ưu hoặc không tìm được lời giải mặc dù có tồn tại lời giải do:
  - Có khuynh hướng sa lầy ở cực đại cục bộ
  - Cao nguyên
  - Chỗm chỉ với một phép toán, không cho ra trạng thái « tốt hơn », nhưng với một vài phép toán có thể chuyển đến trạng thái « tốt hơn »

# Leo đồi (Hill climbing)

## ■ Một vài giải pháp xử lý các vấn đề này:

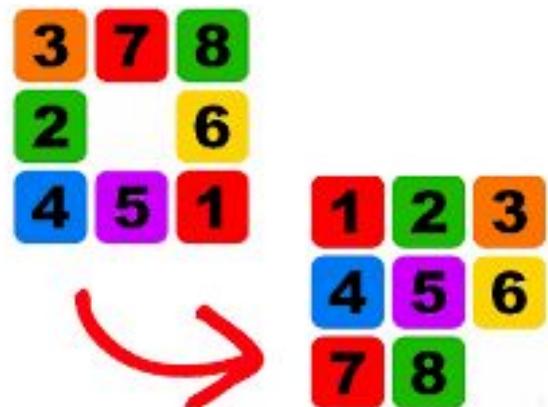
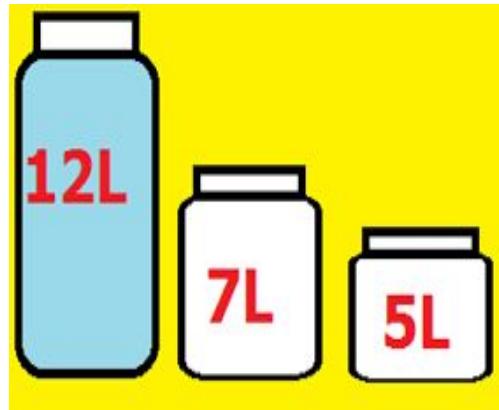
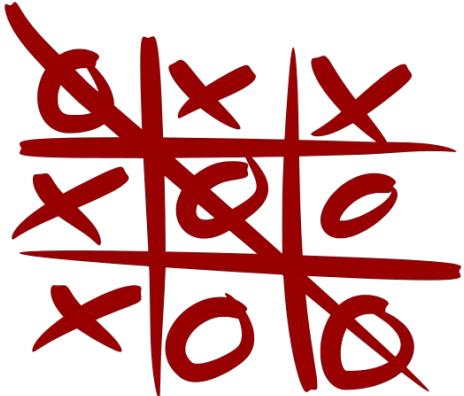
- **Quay lui** « một vài bước » trước đó và thử đi theo một hướng khác. Để thực thi chiến lược này, duy trì một danh sách các bước đã trải qua. Giải pháp này đặc biệt phù hợp để xử lý tình huống « Local Optima »
- **Tạo ra một « bước nhảy đột phá » theo một hướng:** để chuyển sang một « vùng » mới trong không gian tìm kiếm. Phù hợp để xử lý tình huống « Plateau »
- **Áp dụng nhiều hơn một toán tử** để nhận được một trạng thái sau đó mới kiểm thử. Phù hợp để xử lý tình huống « Ridge »



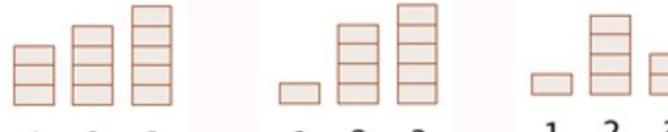
# Nội dung

- Biểu diễn bài toán trong KGTT
- Tìm kiếm mù (uninformed search)
- Tìm kiếm heuristic (informed search)
- Cây trò chơi, cắt tỉa alpha -beta

# Sự khác biệt giữa các bài toán?



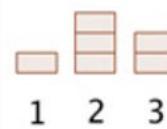
Giả sử có 3 đồng, mỗi đồng lần lượt có 3,4,5 đồng xu



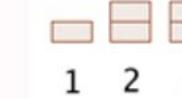
A lấy 2 đồng từ đồng 1

B lấy 3 đồng từ đồng 3

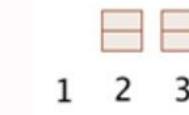
A lấy 1 đồng từ đồng 2



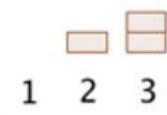
B lấy 1 đồng từ đồng 2



A lấy 1 đồng từ đồng 1



B lấy 1 đồng từ đồng 2



A lấy 1 đồng từ đồng 3



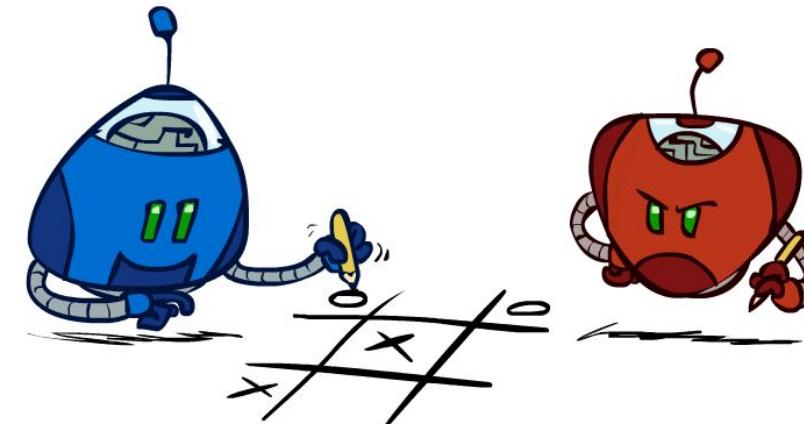
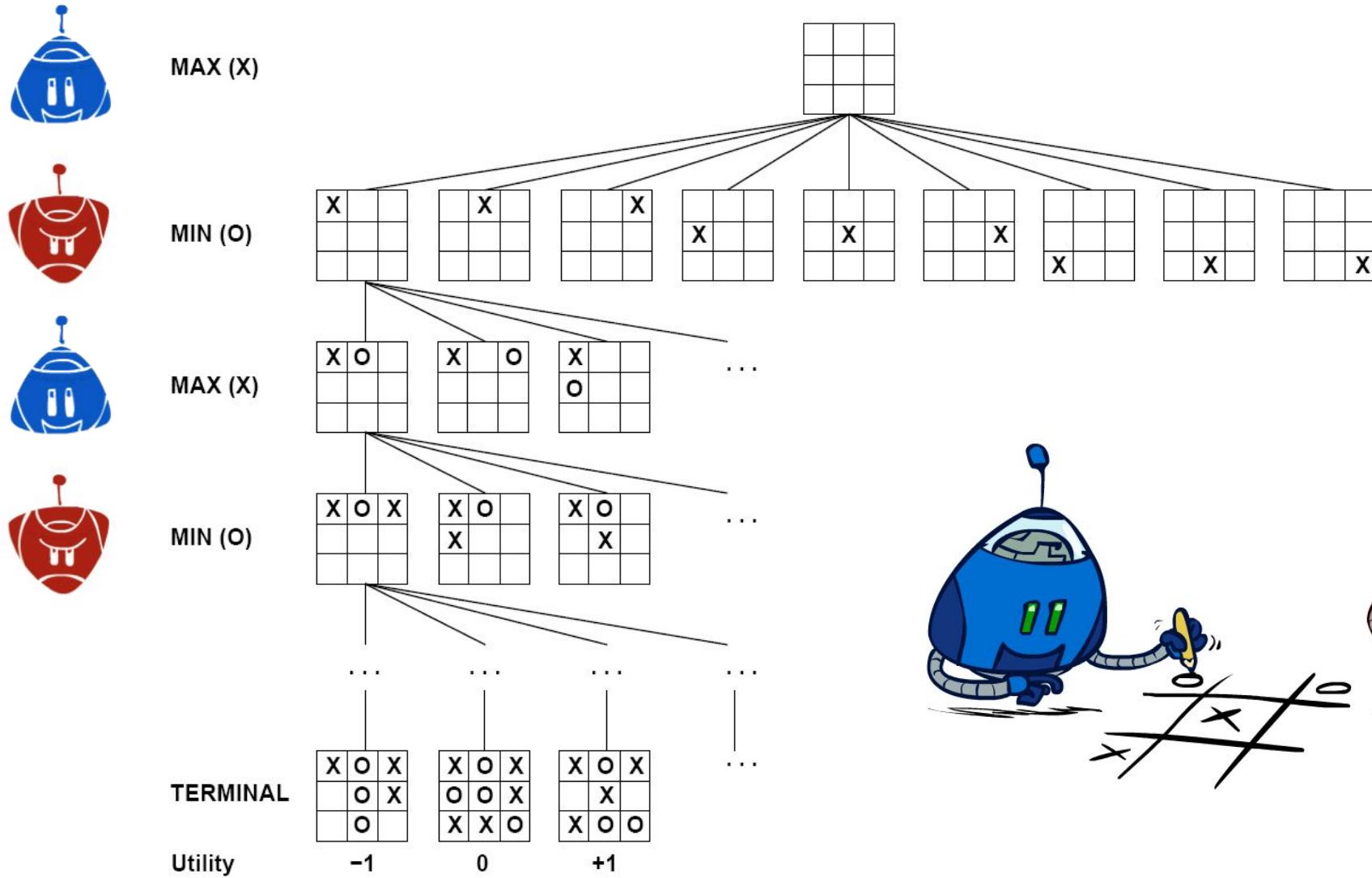
B lấy 1 đồng từ đồng 2



A lấy đồng cuối cùng và thắng

- Có hai người chơi, khi đến lượt, người chơi có thể lấy một số lượng tùy ý đồng xu từ một đồng duy nhất.
- Họ phải lấy ít nhất 1 đồng xu và họ không được lấy các đồng xu từ đồng khác.
- Người thắng là người lấy đồng xu cuối cùng, nghĩa là không còn đồng xu nào sau nước đi của người đó

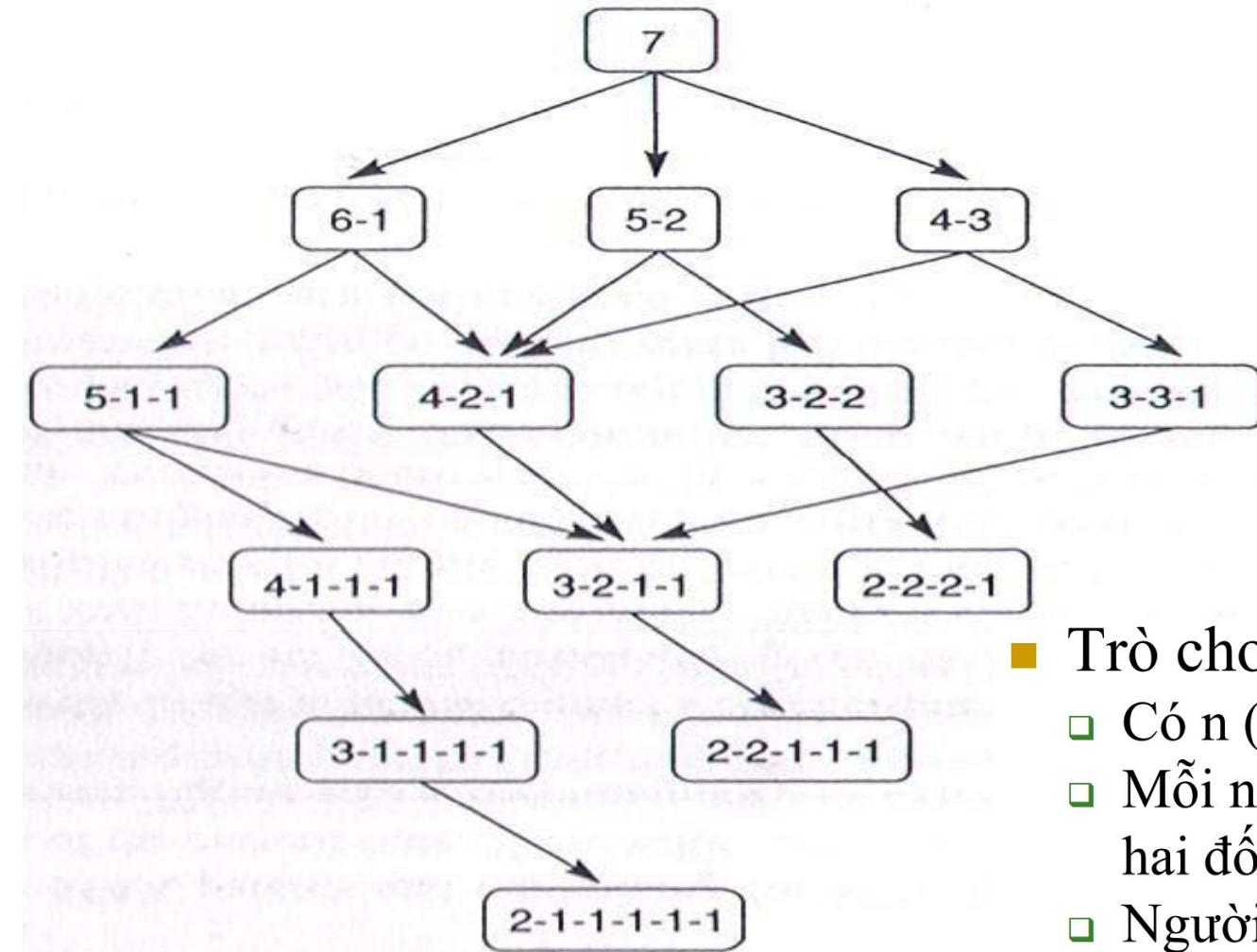
# Cây trò chơi Tic-Tac-Toe



# Cây trò chơi

- Trò chơi một trong những đặc tính được xem là “thông minh” của con người
- Trò chơi là phiên bản “F1” của AI
- Đã đạt được những thành tựu đáng kể
- Ở đây ta chỉ xem xét các dạng trò chơi trí tuệ, đối kháng (board game)

# Không gian trạng thái trò chơi Nim



## ■ Trò chơi Nim

- ❑ Có  $n$  ( $n > 2$ ) đồng xu
- ❑ Mỗi nước đi, người chơi chia các đồng xu này thành hai đồng nhỏ có số lượng mỗi đồng khác nhau
- ❑ Người thua sẽ là người cuối cùng không chia được theo yêu cầu của bài toán

# Ứng dụng Heuristic trong các trò chơi

- Sử dụng không gian trạng thái để giải quyết bài toán
  - *Tìm kiếm mù*
  - *Tìm kiếm heuristic – có thông tin bổ sung*
  - *Sử dụng heuristic cho trò chơi*
  
- *Có 2 người tham gia vào quá trình sinh trạng thái*
- *Bạn tạo ra trạng thái này, đối thủ của bạn sẽ tạo ra trạng thái kế tiếp với mong muốn đánh bại bạn*

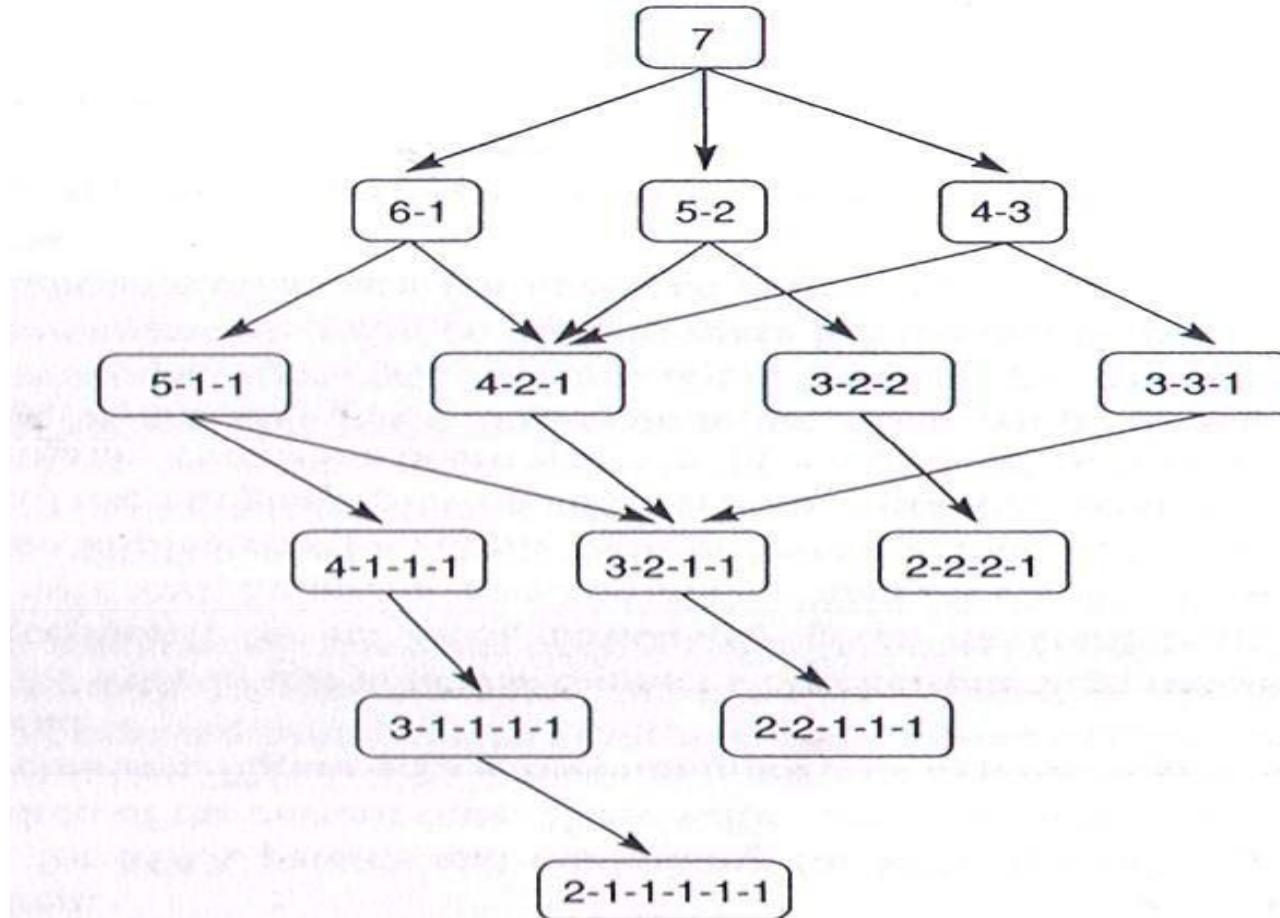
# Ứng dụng Heuristic trong các trò chơi

## ■ Trò chơi Nim:

- Một số token (đồng xu, mảnh gỗ...) được đặt giữa 2 đối thủ.
- Ở mỗi lượt đi, người chơi phải chia các token **thành 2 phần (không rỗng) với số lượng khác nhau**. VD: 6 token có thể được chia thành 5 - 1, 4 - 2 (trường hợp 3 - 3 là không hợp lệ)
- Khi các token không thể được chia một cách hợp lệ ở lượt chơi kế tiếp, người chơi thuộc về lượt đi đó sẽ thua cuộc.

# Giải thuật minimax

## ■ Không gian trạng thái trò chơi Nim



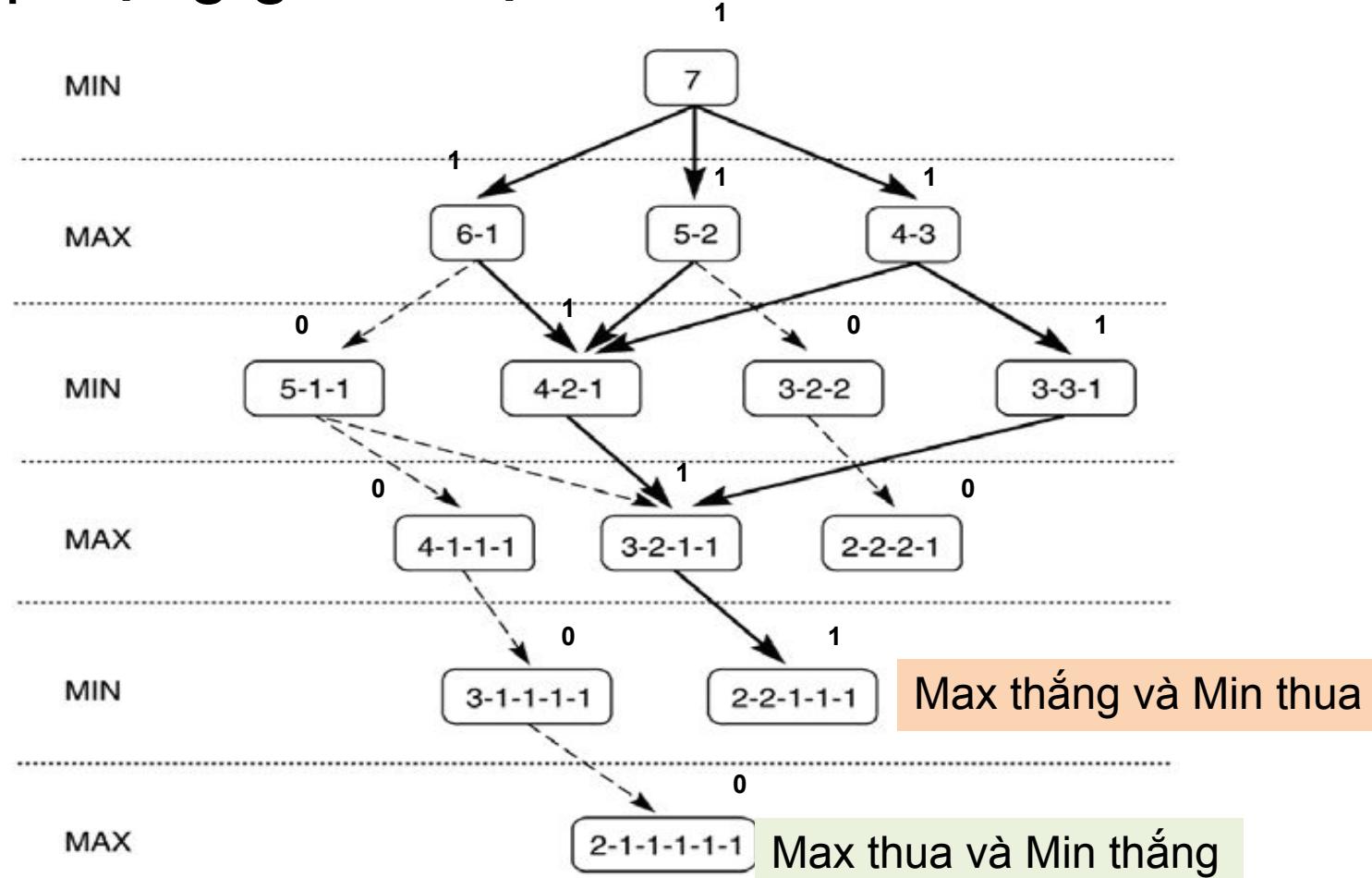
# Giải thuật minimax

## ■ Giải thuật minimax:

- Một đấu thủ trong trò chơi được gọi là **MIN** và đấu thủ còn lại là **MAX**. Max đại diện cho người chơi luôn muốn chiến thắng. Min đại diện cho người chơi cố gắng cho người max giành số điểm càng thấp càng tốt
- Giá trị của nút lá:
  - 1 nếu là MAX thắng,
  - 0 nếu là MIN thắng (MAX thua)
- Minimax sẽ truyền các giá trị này lên cao dần trên đồ thị, qua các nút cha kế tiếp theo các luật sau:
  - Nếu trạng thái cha là **MAX**, gán cho nó giá trị **lớn nhất** có trong các trạng thái con.
  - Nếu trạng thái cha là **MIN**, gán cho nó giá trị **nhỏ nhất** có trong các trạng thái con.

# Giải thuật minimax

## ■ Áp dụng giải thuật minimax vào trò chơi Nim



Chọn các  
nước đi –  
trạng thái  
có mũi tên  
liên tục

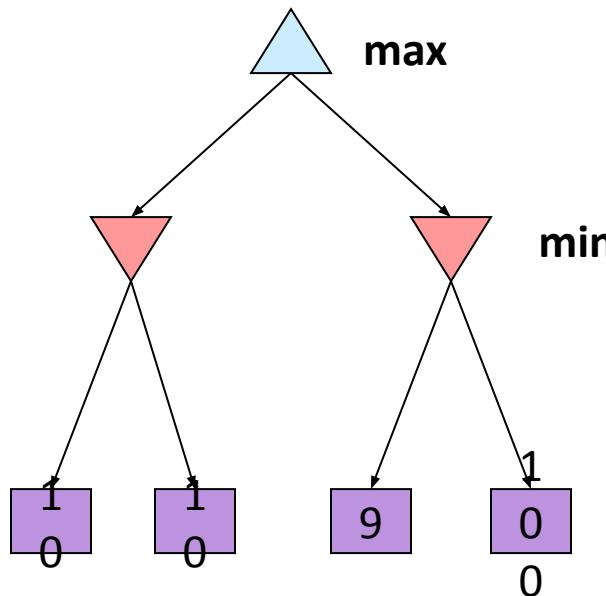
# Giải thuật minimax

## ■ Minimax đến độ sâu lớp cố định

- Đối với các trò chơi phức tạp, đồ thị KGTT có khả năng không được triển khai đến các nút lá
- KGTT chỉ có thể được triển khai đến một số mức xác định (tùy vào tiềm năng về thời gian và bộ nhớ) => tính trước n nước đi
- Vì các nút lá của đồ thị con này không phải là trạng thái kết thúc của trò chơi => không xác định được các giá trị thắng - thua (1 hoặc 0)
- Cần sử dụng một hàm đánh giá Heuristic nào đó
- Giá trị nút lá là các giá trị Heuristic đạt được sau n nước đi kể từ nút xuất phát (tùy vào hàm đánh giá Heuristic cụ thể)
- Các giá trị này sẽ được truyền ngược về nút gốc tương tự như trong trò chơi Nim, và chỉ lá giá trị của trạng thái tốt nhất có thể.

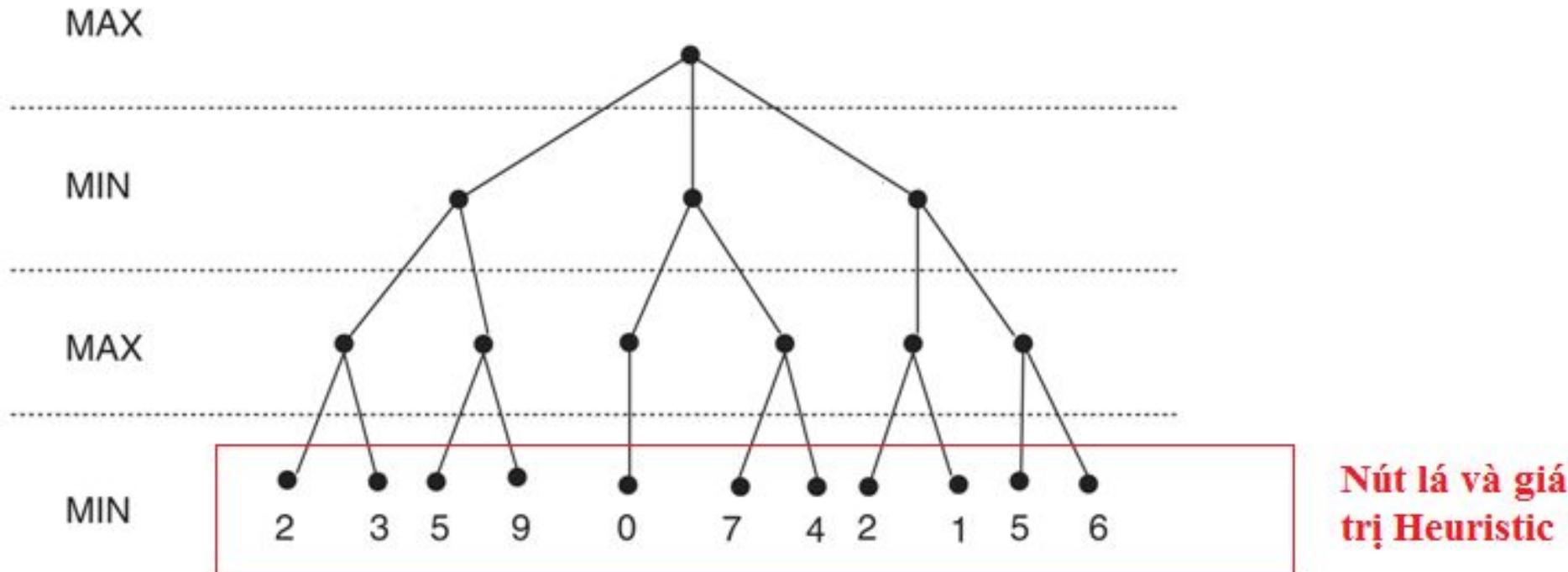
# Minimax với độ sâu lớp cố định

- Các nút lá được gán các giá trị *heuristic*
- Còn giá trị tại các nút trong là các giá trị nhận được dựa trên giải thuật Minimax



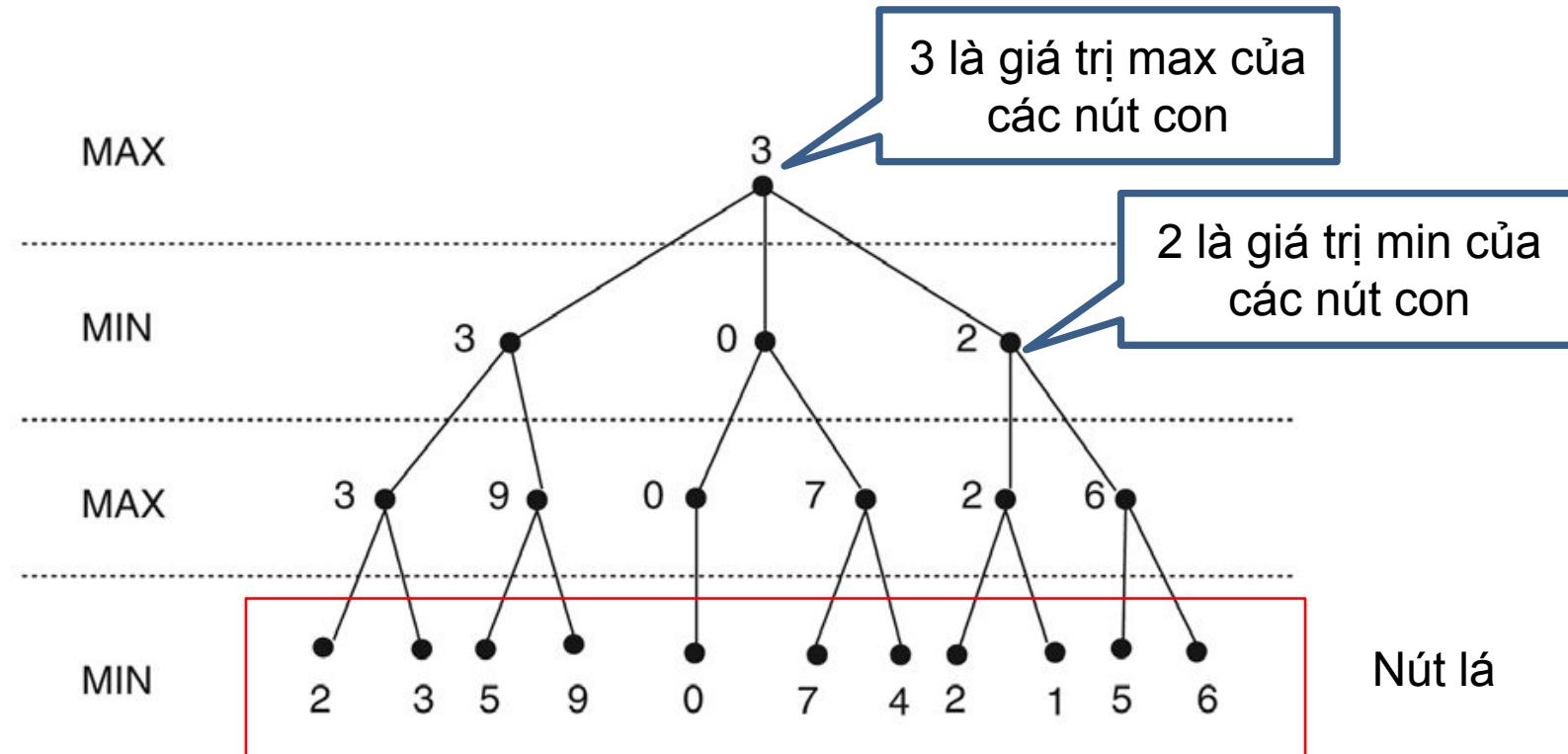
# Minimax với độ sâu lớp cố định

- Ví dụ cho KGTT giả định và giá trị Heuristic của các nút lá. Hãy tính giá trị của các nút còn lại



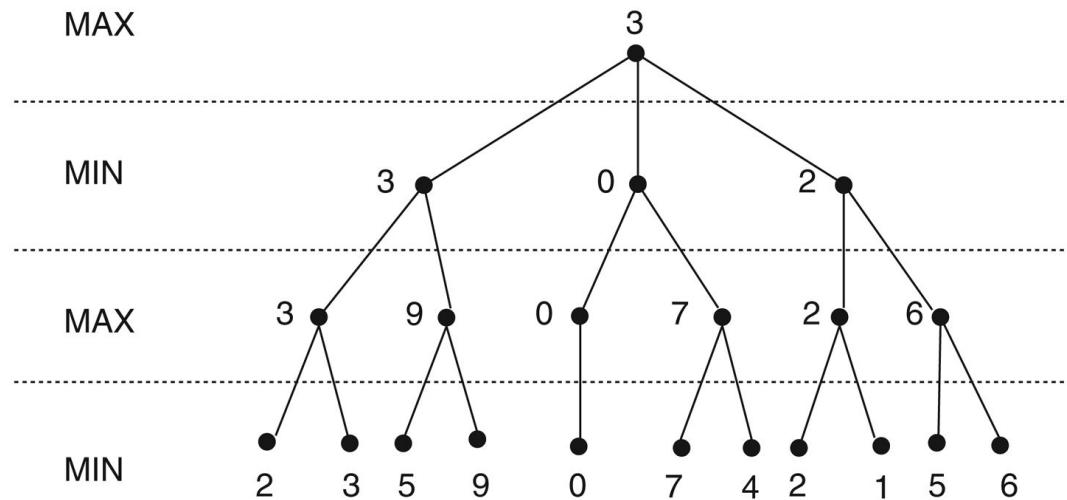
# Giải thuật minimax

## ■ Ví dụ:



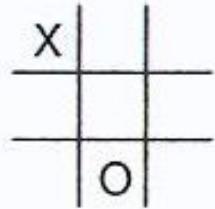
# Minimax với độ sâu lớp cố định

- Minimax đối với một KGTT giả định.

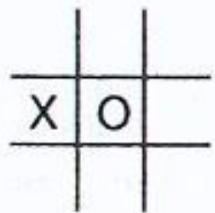
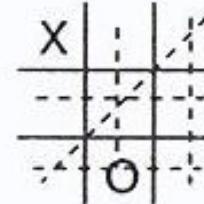
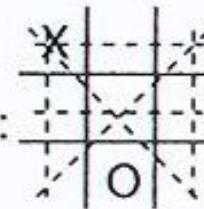


- Các nút lá được gán các giá trị ***heuristic***
- Còn giá trị tại các nút trong là các giá trị nhận được dựa trên giải thuật Minimax

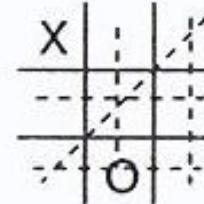
# Heuristic trong trò chơi tic-tac-toe



X has 6 possible win paths:  
O has 5 possible wins:  
 $E(n) = 6 - 5 = 1$



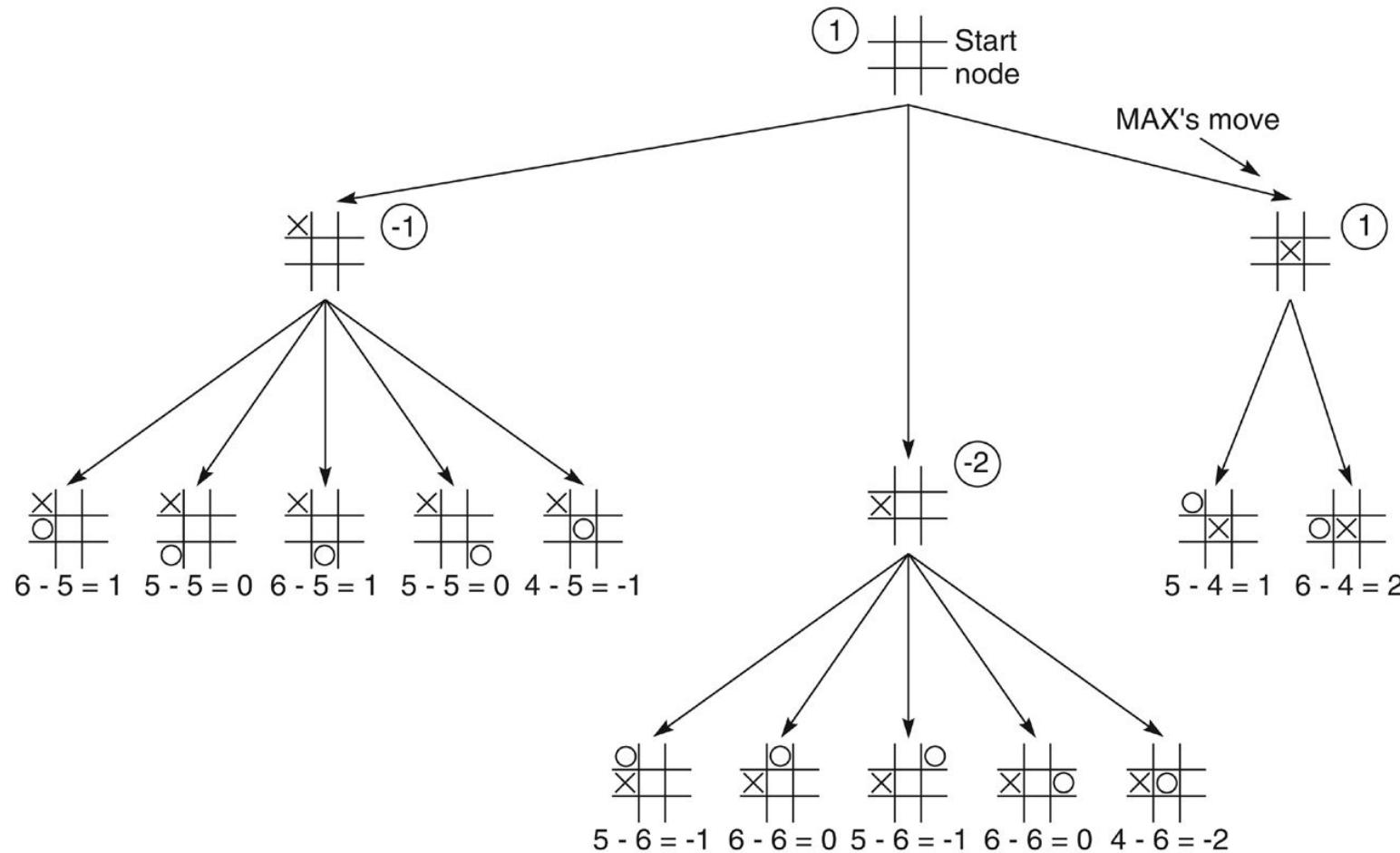
X has 4 possible win paths;  
O has 6 possible wins  
 $E(n) = 4 - 6 = -2$



**Hàm Heuristic:**  $E(n) = M(n) - O(n)$

Trong đó:  $M(n)$  là tổng số đường thắng có thể của tôi  
 $O(n)$  là tổng số đường thắng có thể của đối thủ  
 $E(n)$  là trị số đánh giá tổng cộng cho trạng thái  $n$

# Minimax 2 lớp được áp dụng vào nước đi mở đầu trong tic-tac-toe



Trích từ Nilsson (1971).

## Hàm Heuristic:

$$E(n) = M(n) - O(n)$$

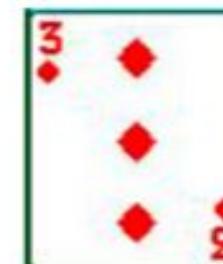
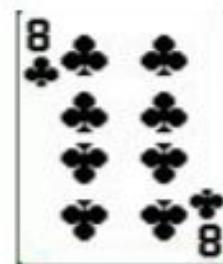
Trong đó:  $M(n)$  là *tổng số đường thắng có thể của tôi*

$O(n)$  là *tổng số đường thắng có thể của đối thủ*

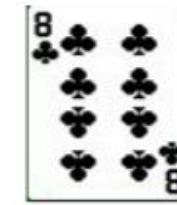
$E(n)$  là *tri số đánh giá tổng cộng cho trạng thái n*

# Bài toán bắt lá bài

- Mỗi người chơi chọn 1 lá bài trong 4 lá bài cho trước
- Với 4 lá bài được cho sẵn, mỗi người sẽ chọn 2 lá bài
- Cộng điểm 2 lá bài đã chọn, nếu người chơi nào có tổng điểm là chẵn và cao nhất sẽ thắng

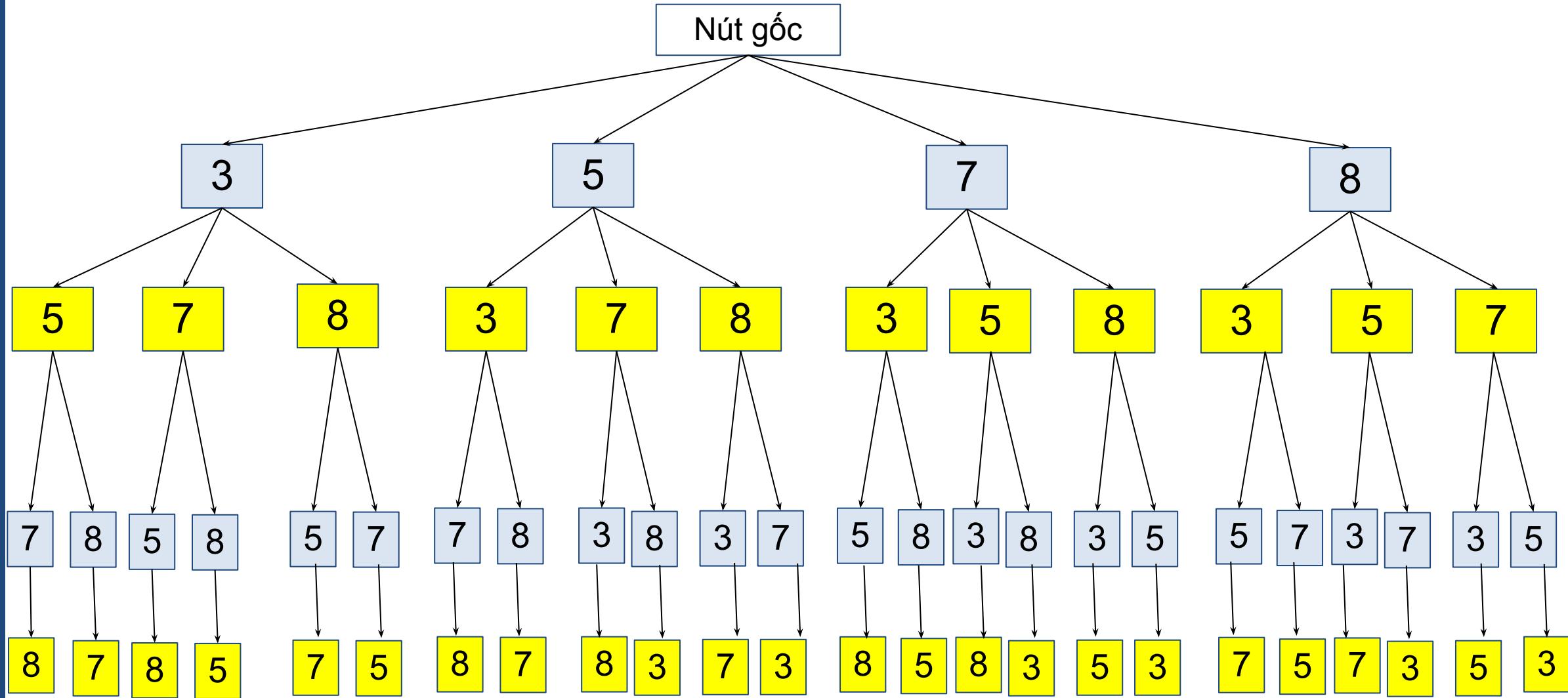
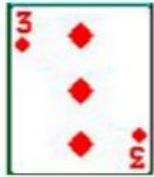
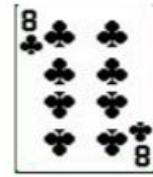


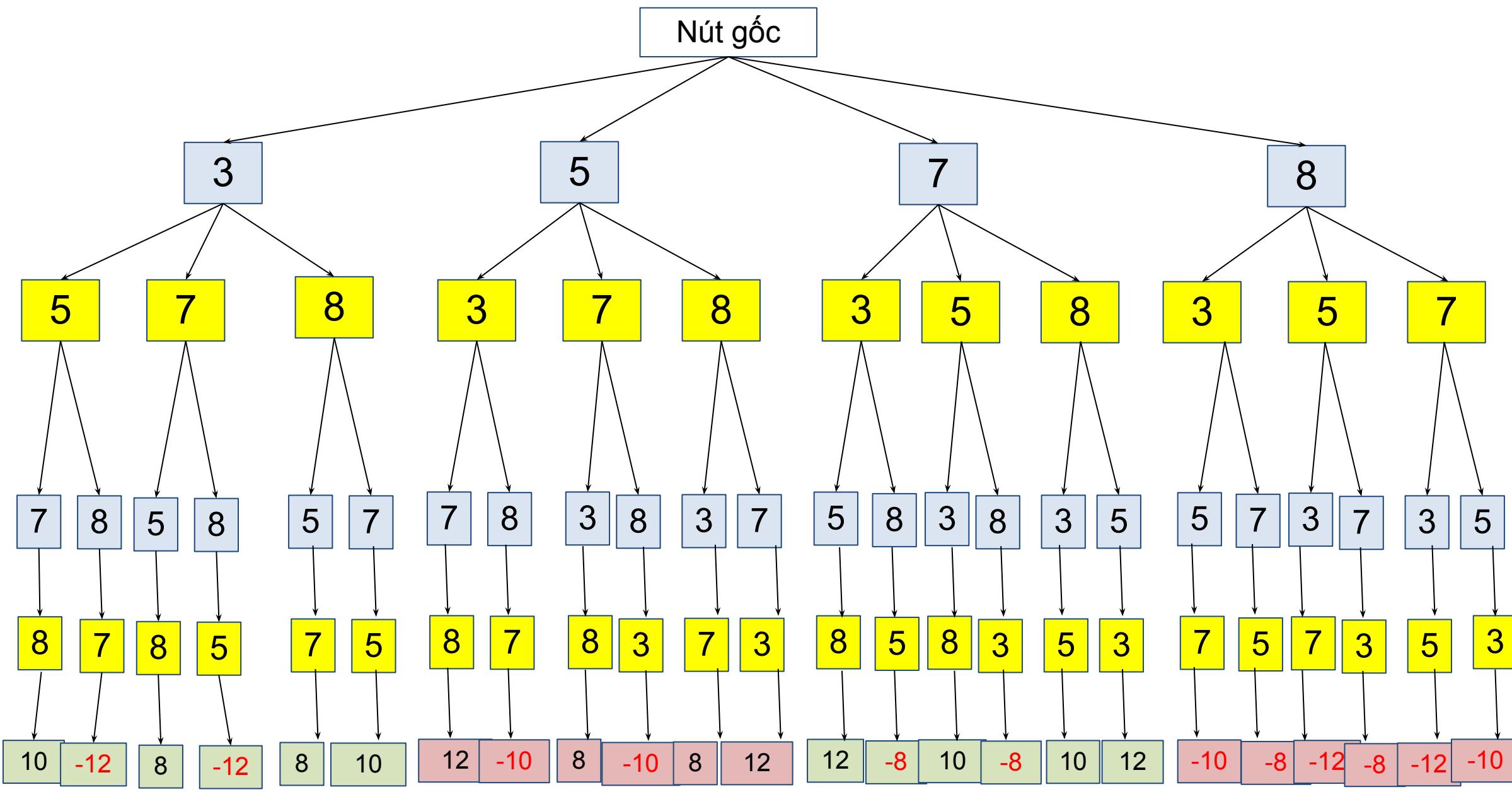
# Bài toán bắt lá bài



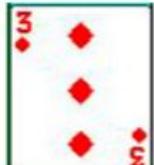
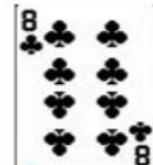
- Để có thể thắng trong trò chơi trên
  - Vẽ toàn bộ không gian trạng thái
  - Gắn điểm số vào mỗi trạng thái kết thúc
  - Sử dụng giải thuật Minimax để cập nhật điểm số từ nút lá đến nút gốc
  - Chọn đường đi có điểm số lớn nhất

# Bài toán bắt lá bài





# Bài toán bắt lá bài



Player 1

Max

Player 2

Min

Player 1

Max

Player 2

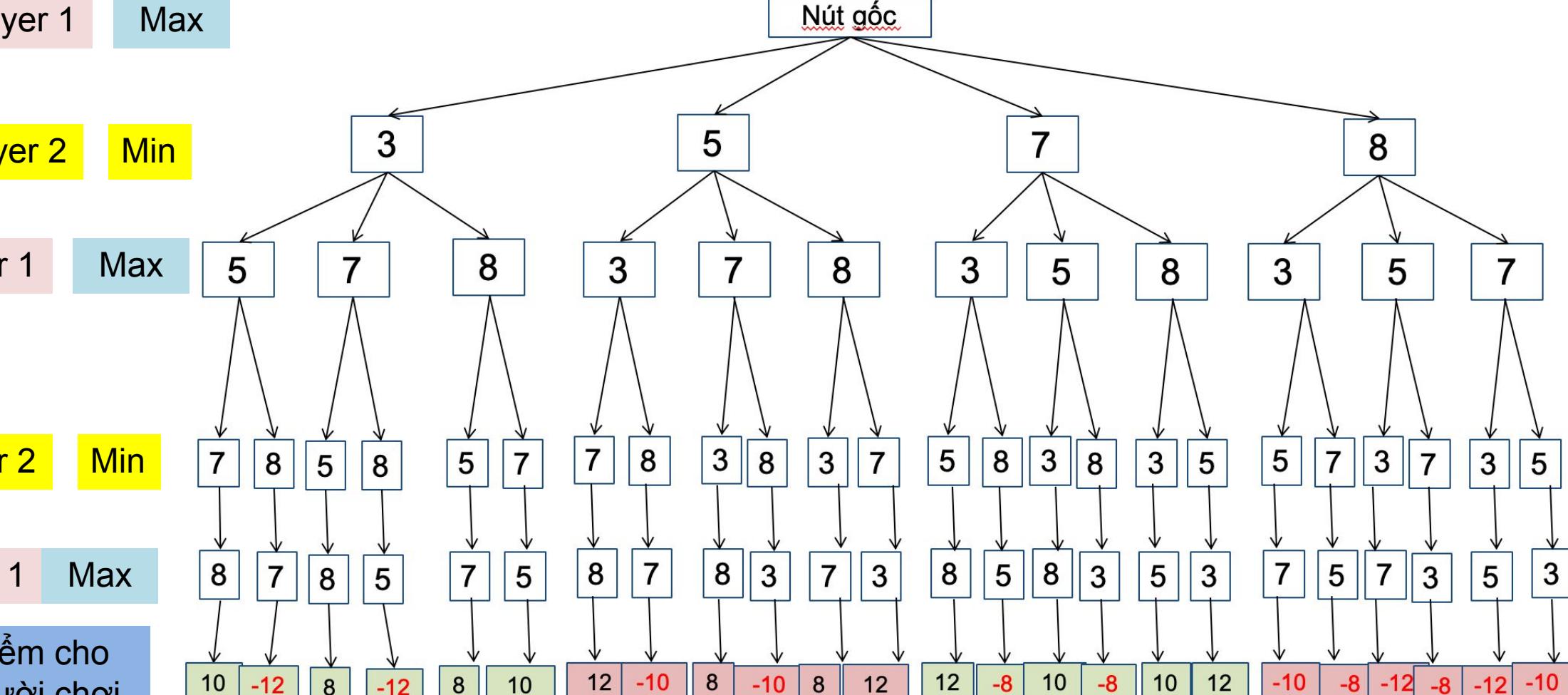
Min

Player 1

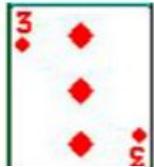
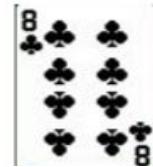
Max

Điểm cho  
người chơi  
Player 1

Nút gốc



# Bài toán bắt lá bài



Player 1 Max

Nút gốc

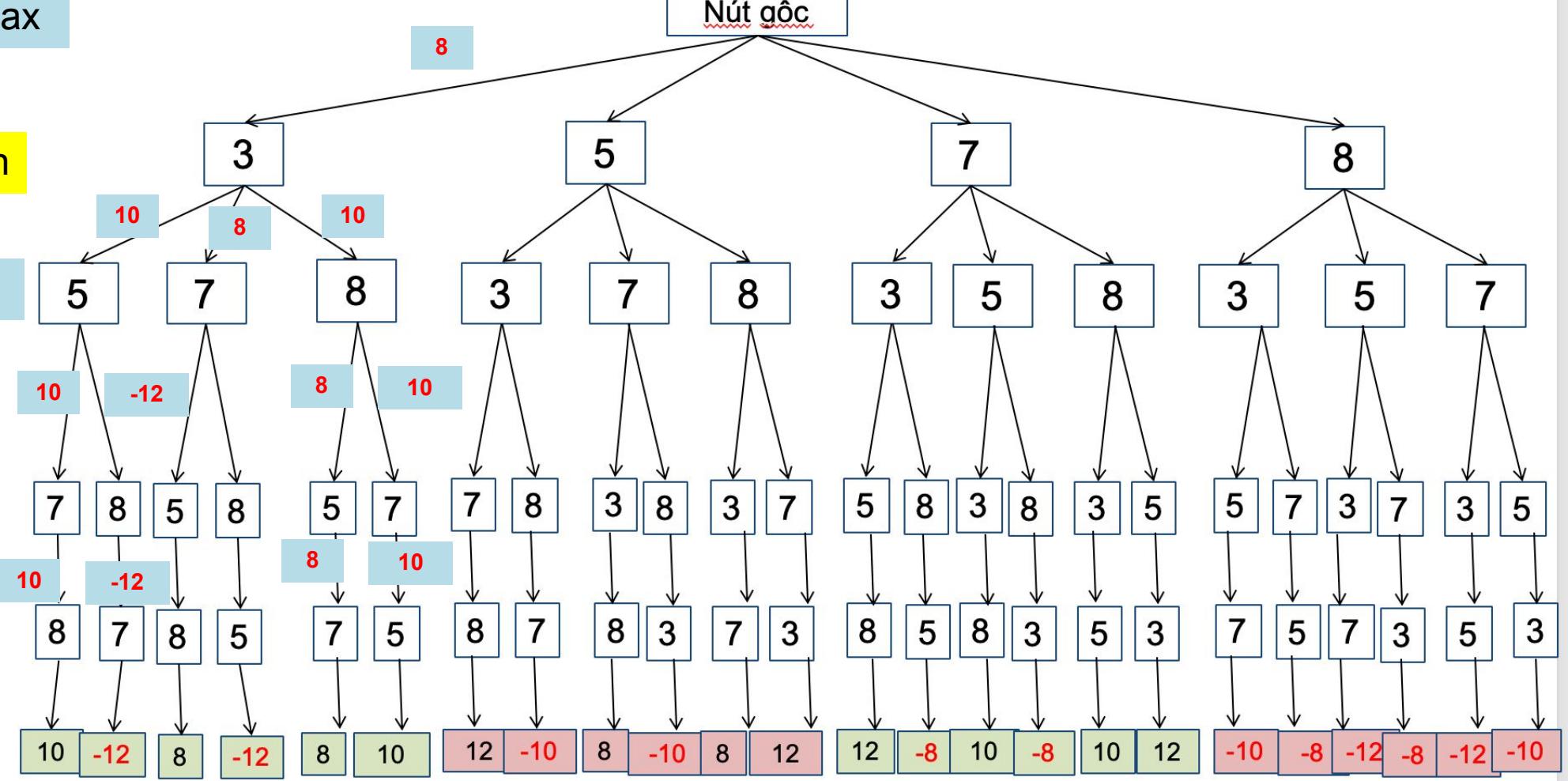
Player 2 Min

Player 1 Max

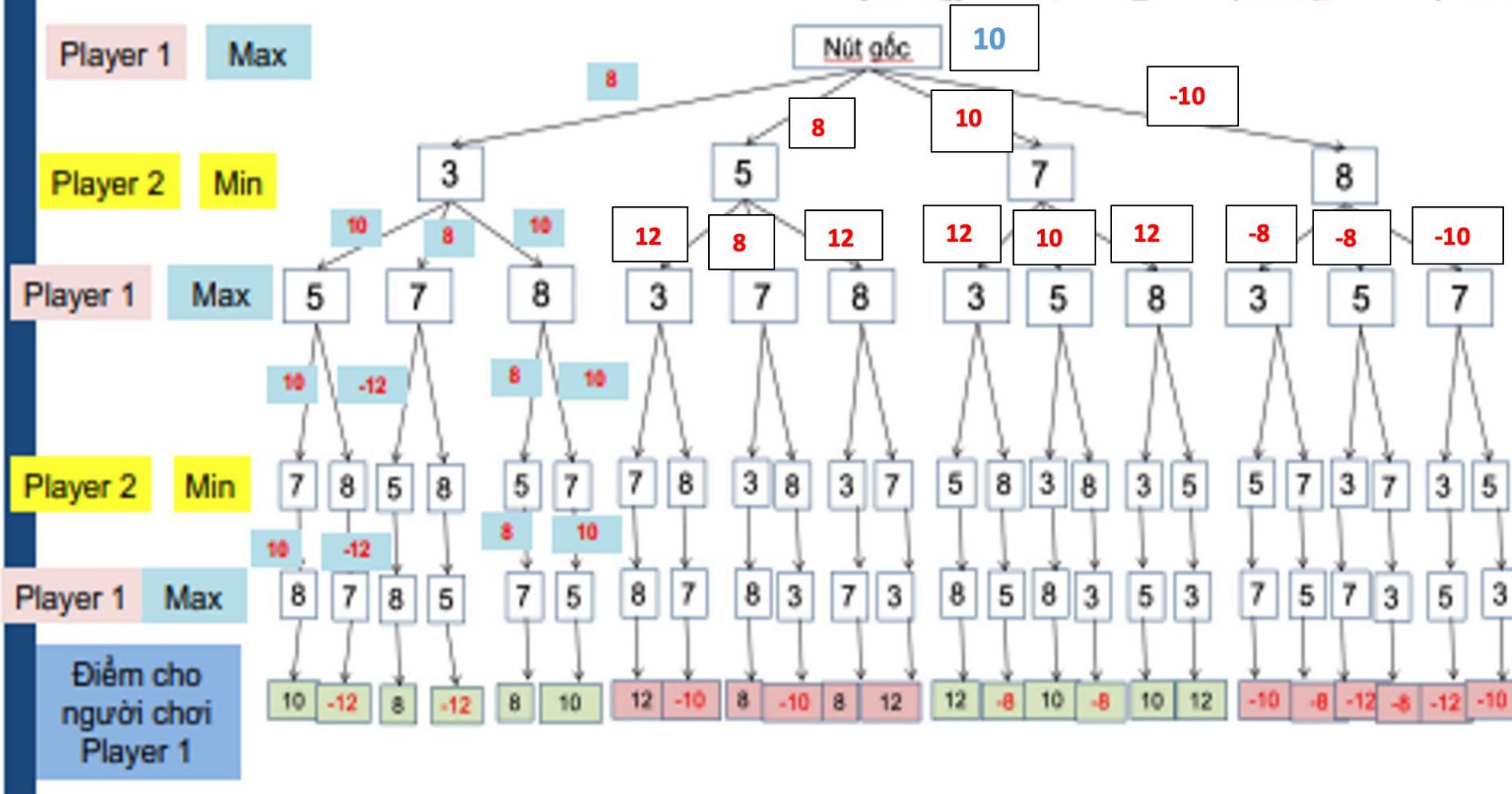
Player 2 Min

Player 1 Max

Điểm cho  
người chơi  
Player 1



# Bài toán bắt lá bài

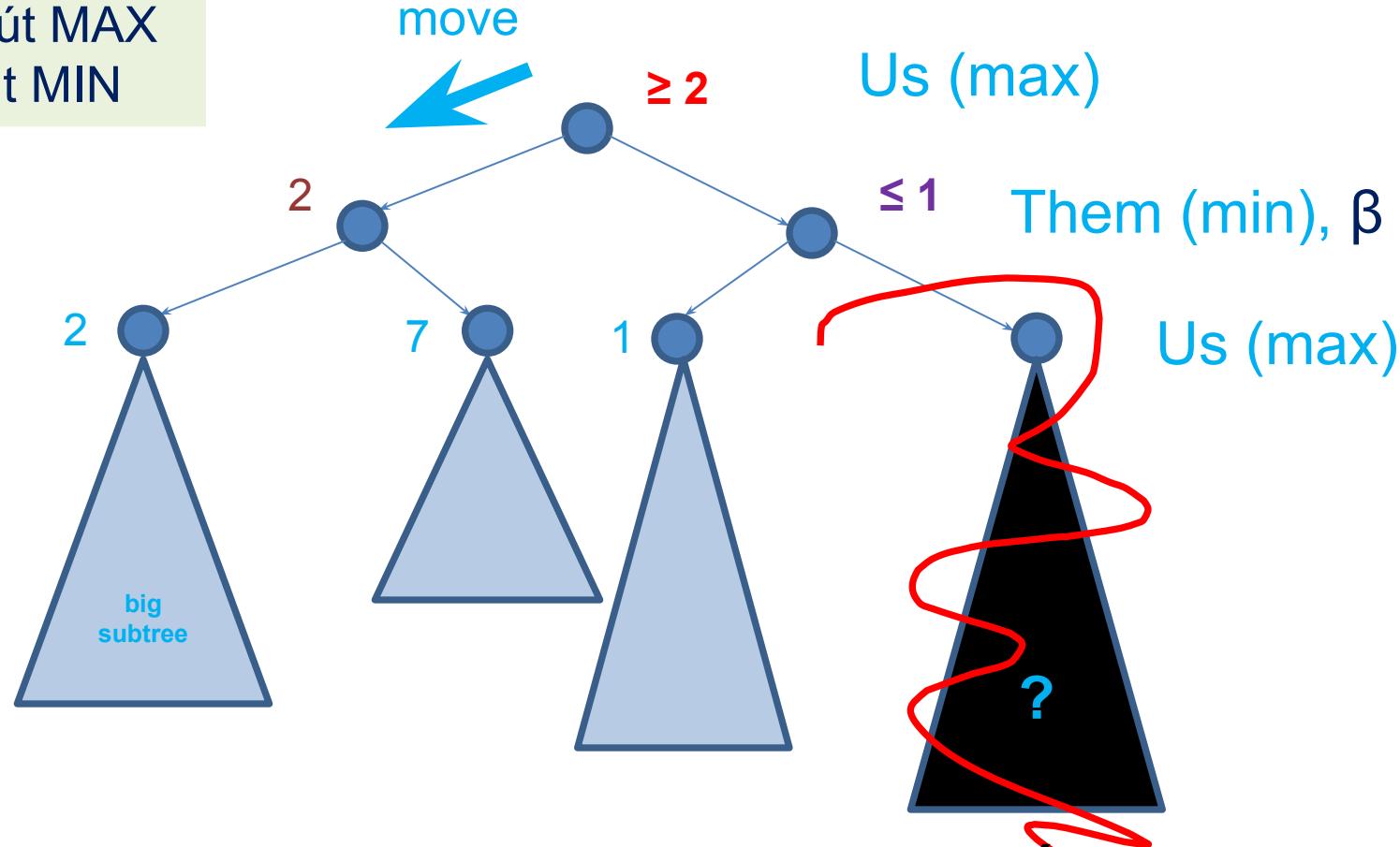


# Thủ tục cắt tỉa alpha – beta

- Tìm kiếm alpha – beta thực hiện theo kiểu tìm kiếm sâu.
- Hai giá trị  $\alpha$ ,  $\beta$  được tạo ra trong quá trình tìm kiếm.
  - $\alpha$  gắn với các nút MAX, không giảm.
  - $\beta$  gắn với các nút MIN, không tăng.
- Luật cắt tỉa alpha – beta: Quá trình tìm kiếm có thể kết thúc bên dưới:
  - Nút MIN có  $\beta \leq \alpha$  của nút cha MAX bất kỳ ( $\alpha$ -cut)
  - Nút MAX có  $\alpha \geq \beta$  của nút cha MIN bất kỳ ( $\beta$ -cut)

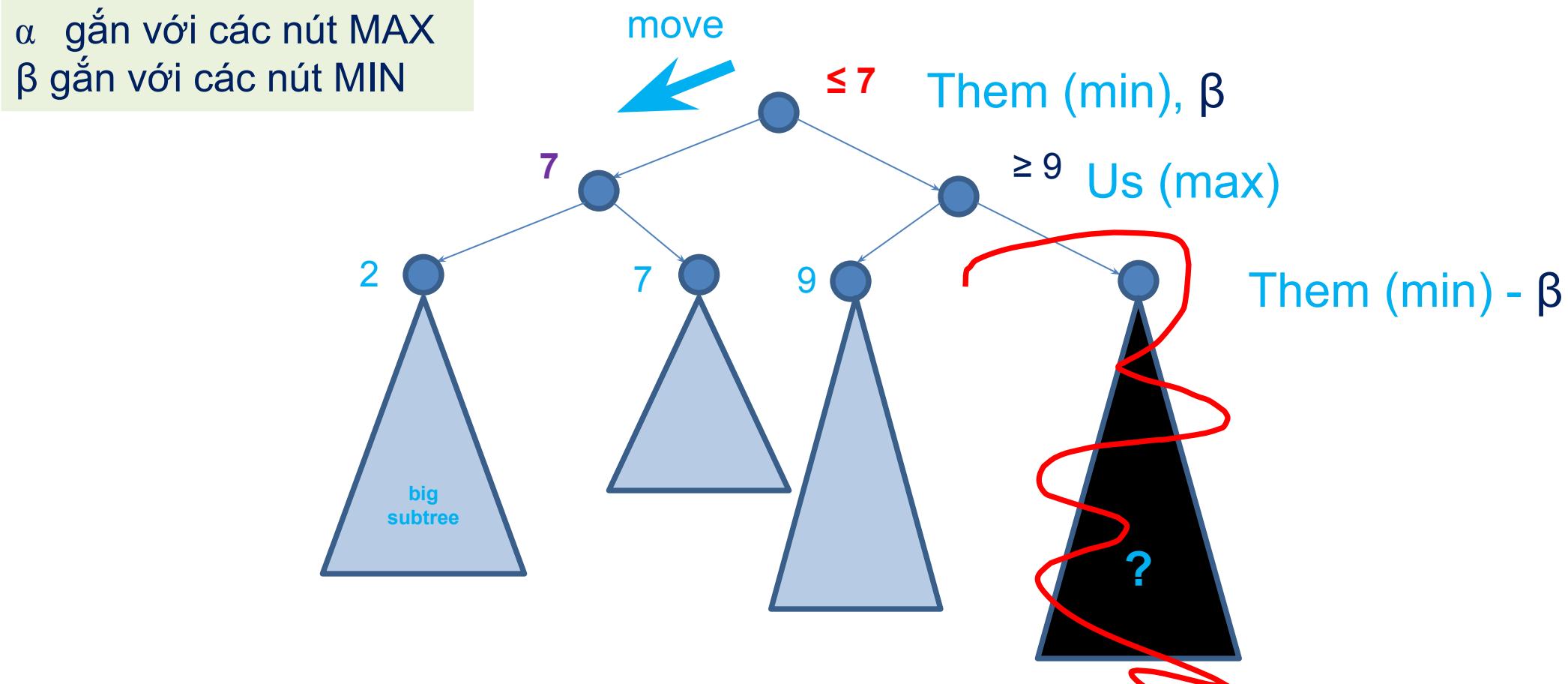
# $\alpha$ -cut

$\alpha$  gắn với các nút MAX  
 $\beta$  gắn với các nút MIN



Nút MIN có  $\beta \leq \alpha$  của nút cha MAX bất kỳ ( $\alpha$ -cut)  
Nút MAX có  $\alpha \geq \beta$  của nút cha MIN bất kỳ ( $\beta$ -cut)

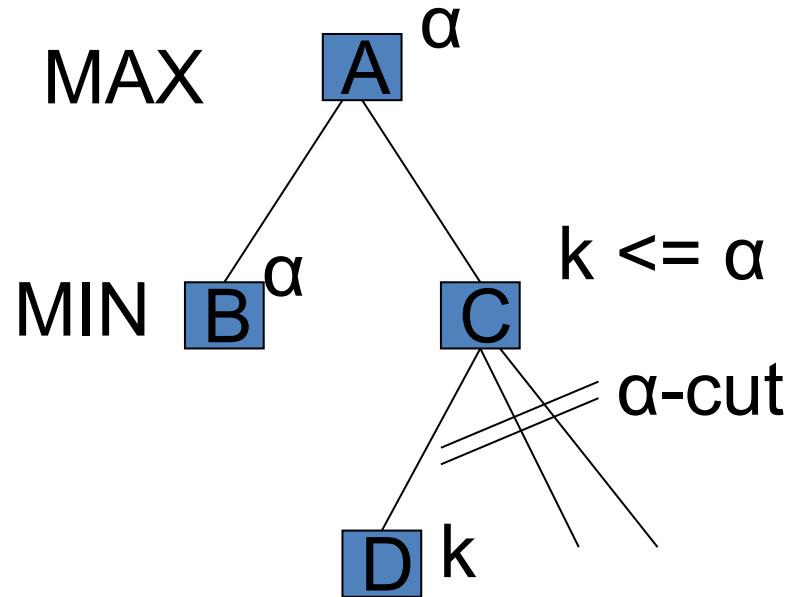
# $\beta$ -cut



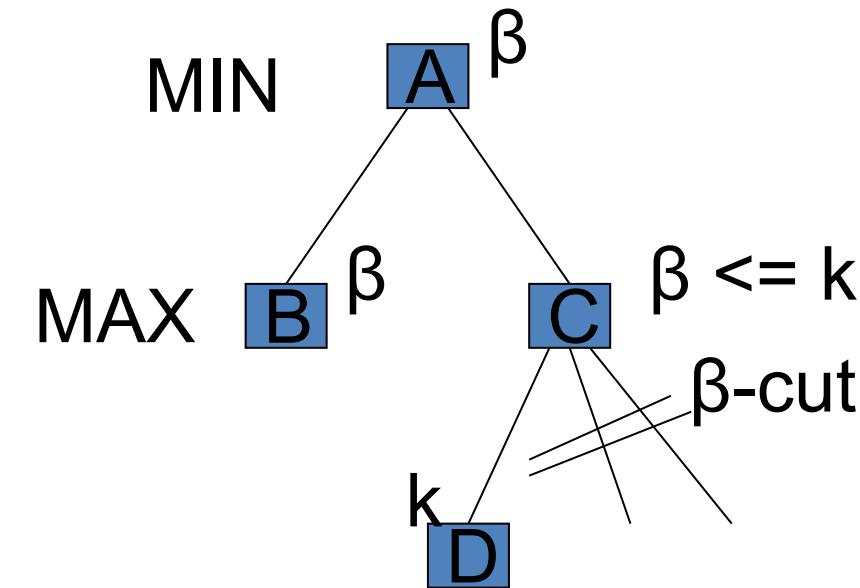
Nút MIN có  $\beta \leq \alpha$  của nút cha MAX bất kỳ ( $\alpha$ -cut)  
Nút MAX có  $\alpha \geq \beta$  của nút cha MIN bất kỳ ( $\beta$ -cut)

# Thủ tục cắt tỉa alpha – beta

## ■ α-cut

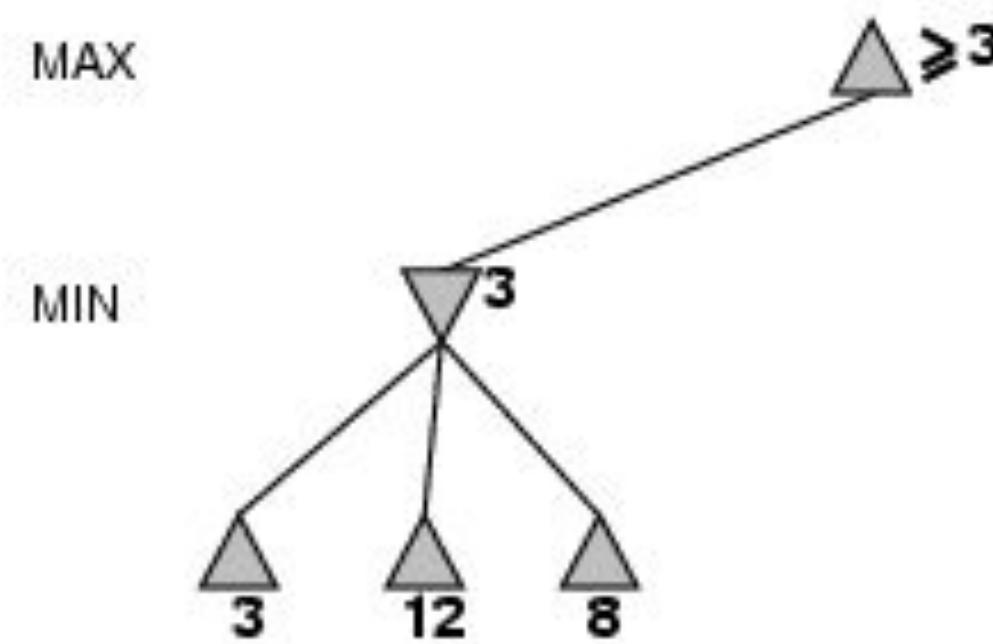


## β-cut

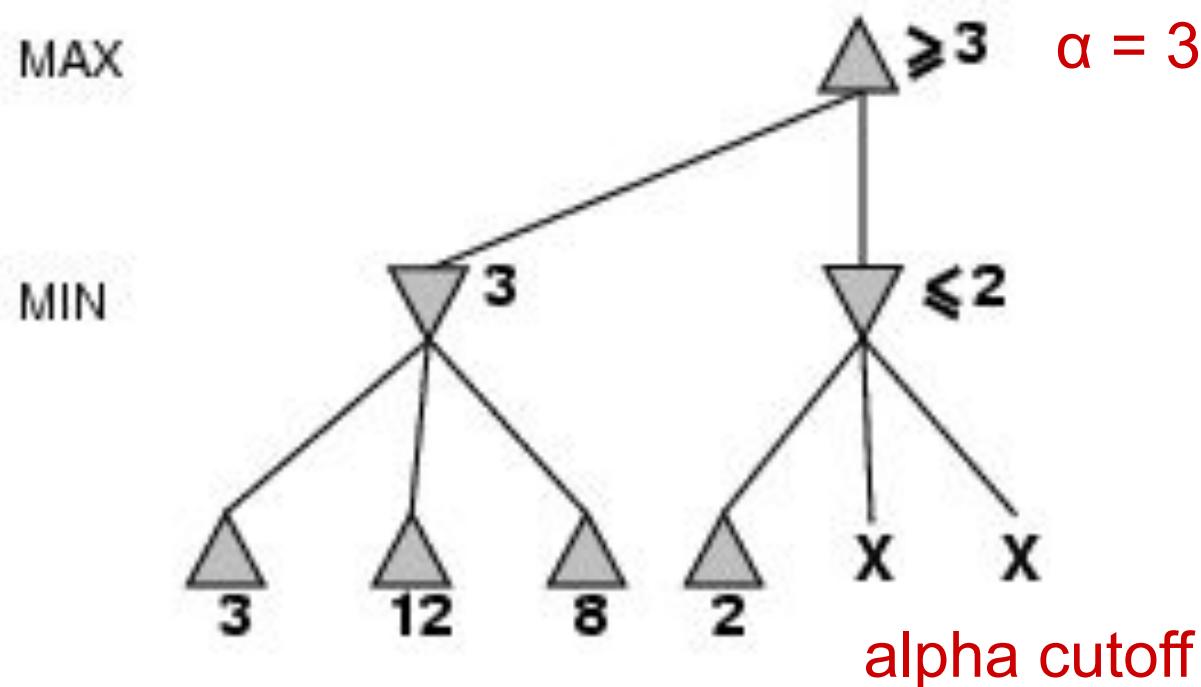


Nút MIN có  $\beta \leq \alpha$  của nút cha MAX bất kỳ ( $\alpha$ -cut)  
Nút MAX có  $\alpha \geq \beta$  của nút cha MIN bất kỳ ( $\beta$ -cut)

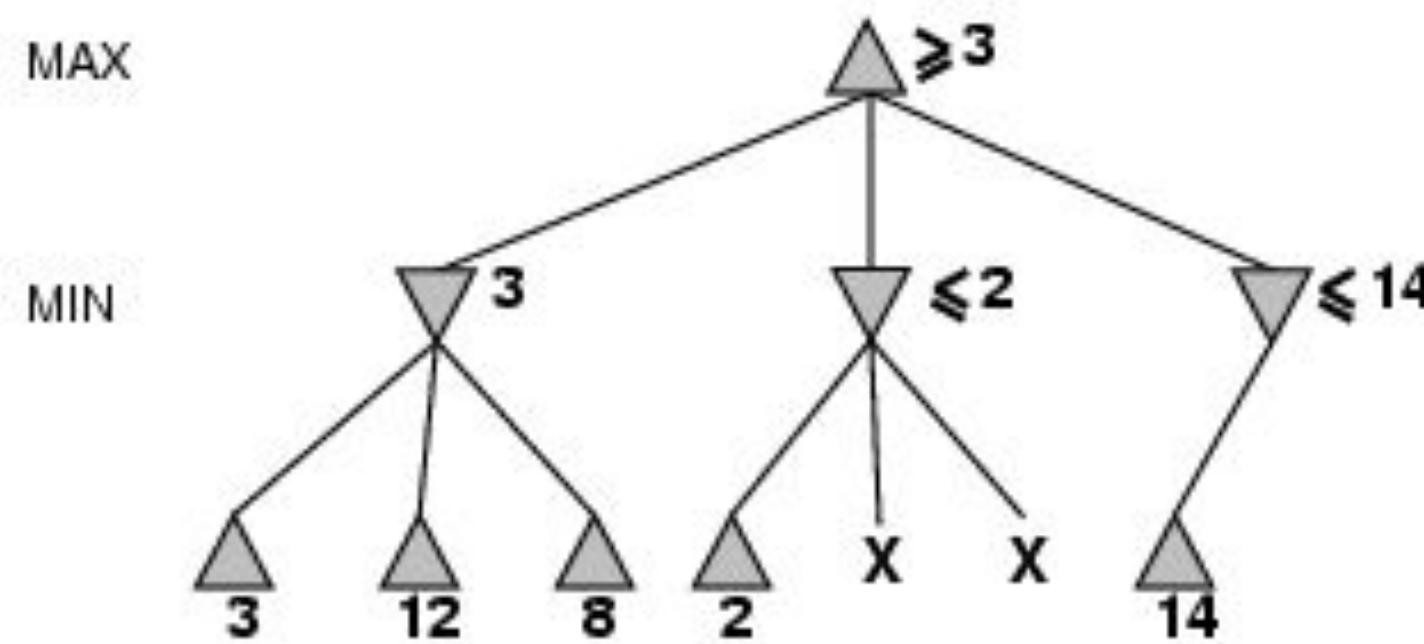
# $\alpha$ - $\beta$ pruning example



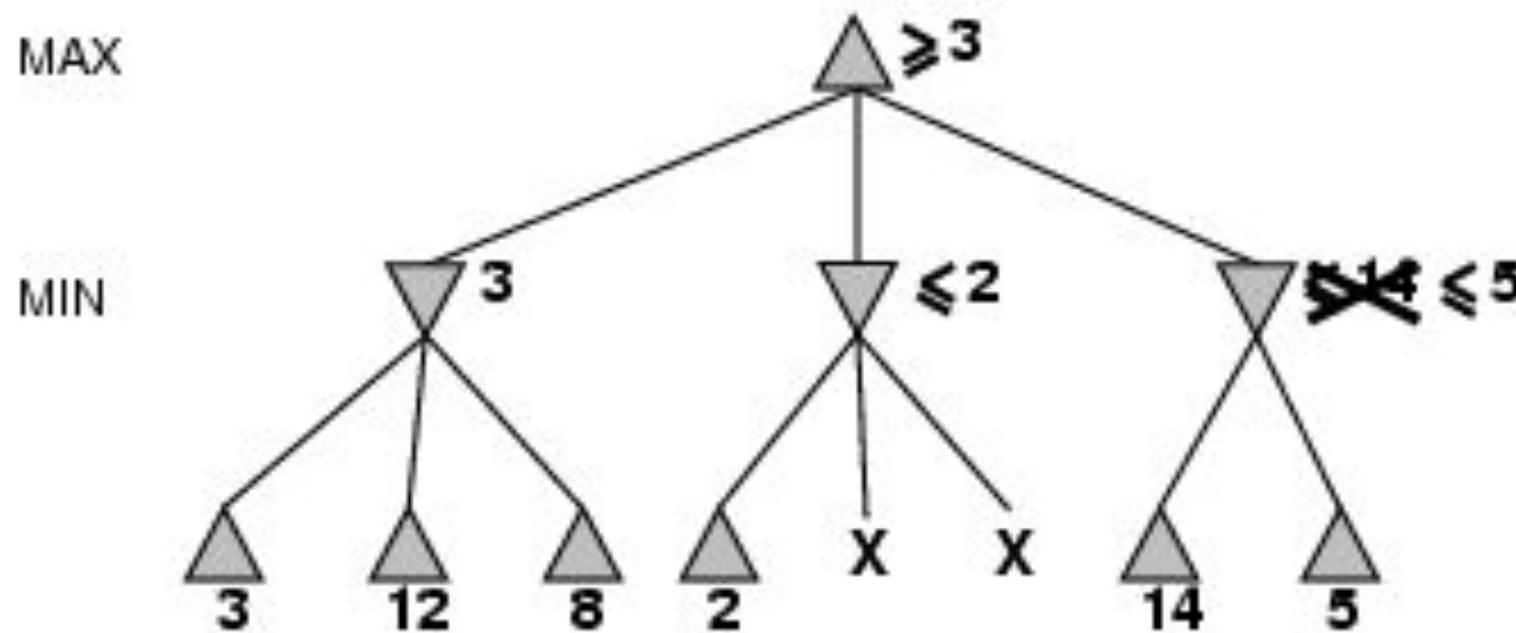
# $\alpha$ - $\beta$ pruning example



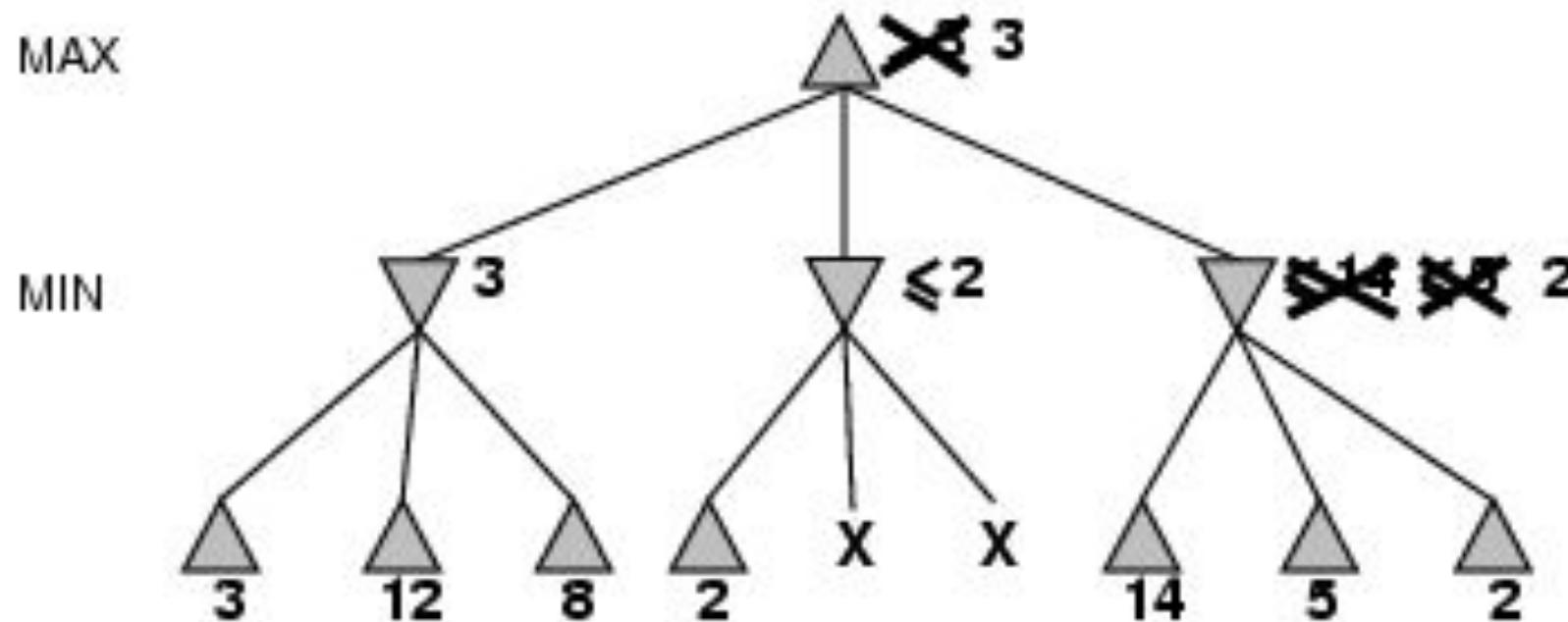
# $\alpha$ - $\beta$ pruning example



# $\alpha$ - $\beta$ pruning example

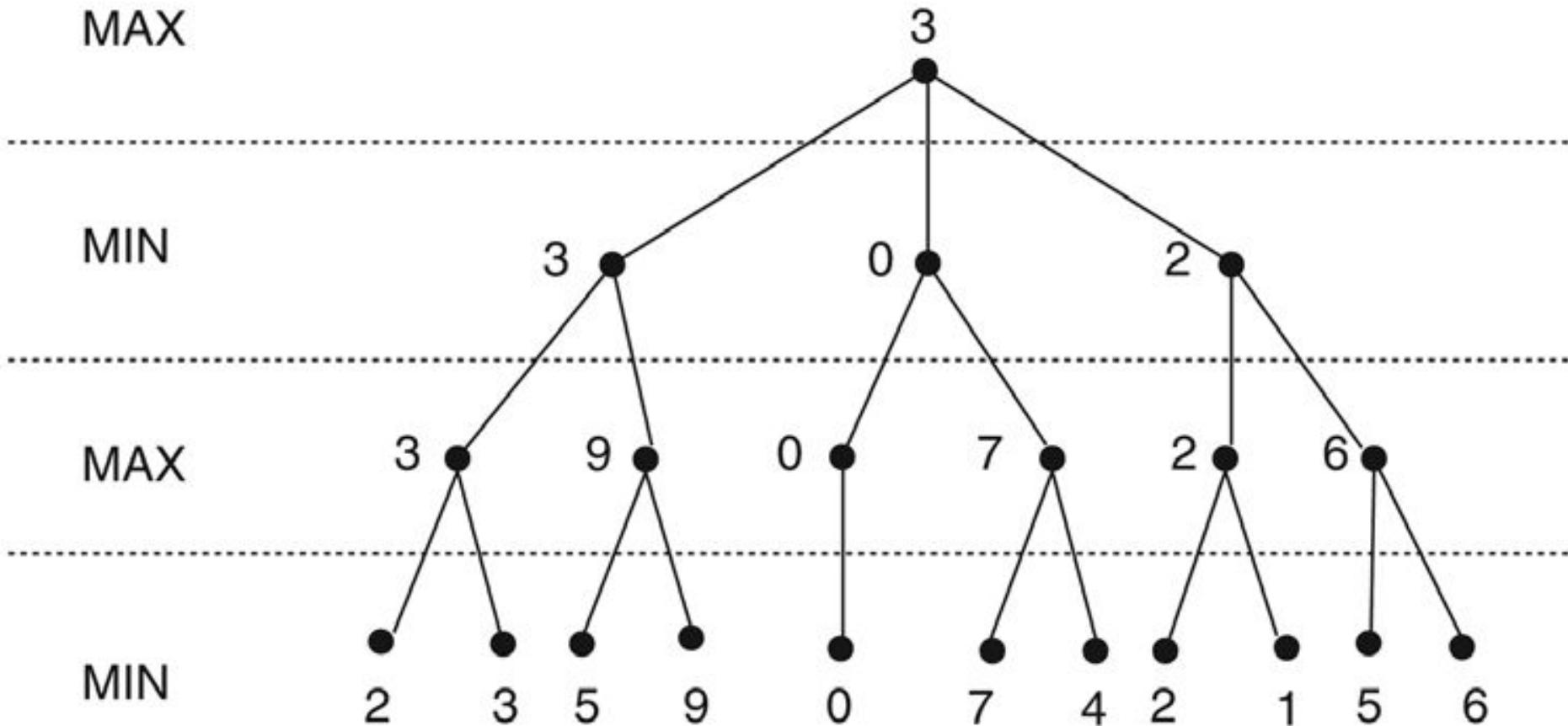


# $\alpha$ - $\beta$ pruning example

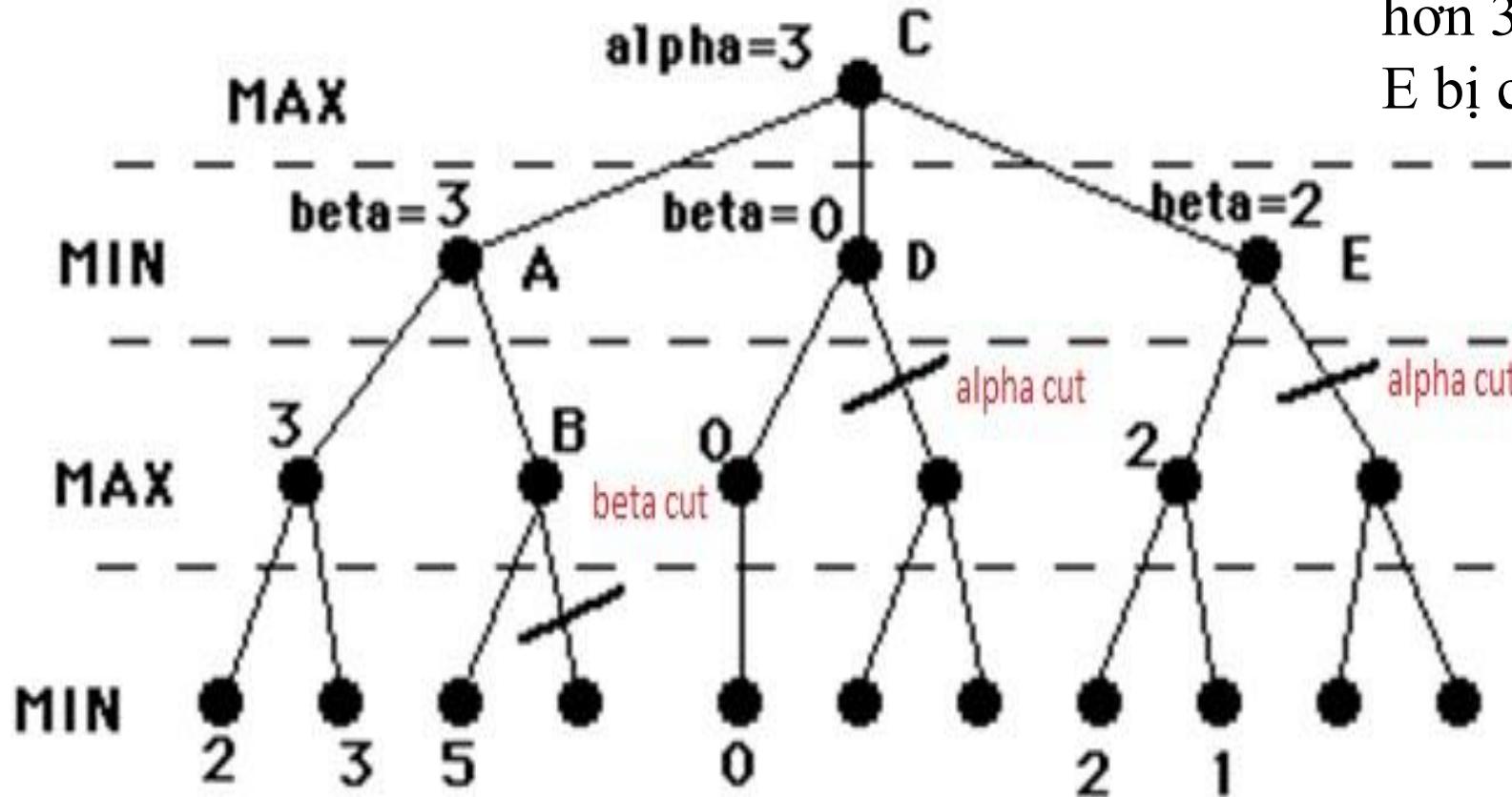


Nút MIN có  $\beta \leq \alpha$  của nút cha MAX bất kỳ ( $\alpha$ -cut)  
Nút MAX có  $\alpha \geq \beta$  của nút cha MIN bất kỳ ( $\beta$ -cut)

■ Minimax đối với một KGTT giả định:



# Thủ tục cắt tỉa alpha – beta

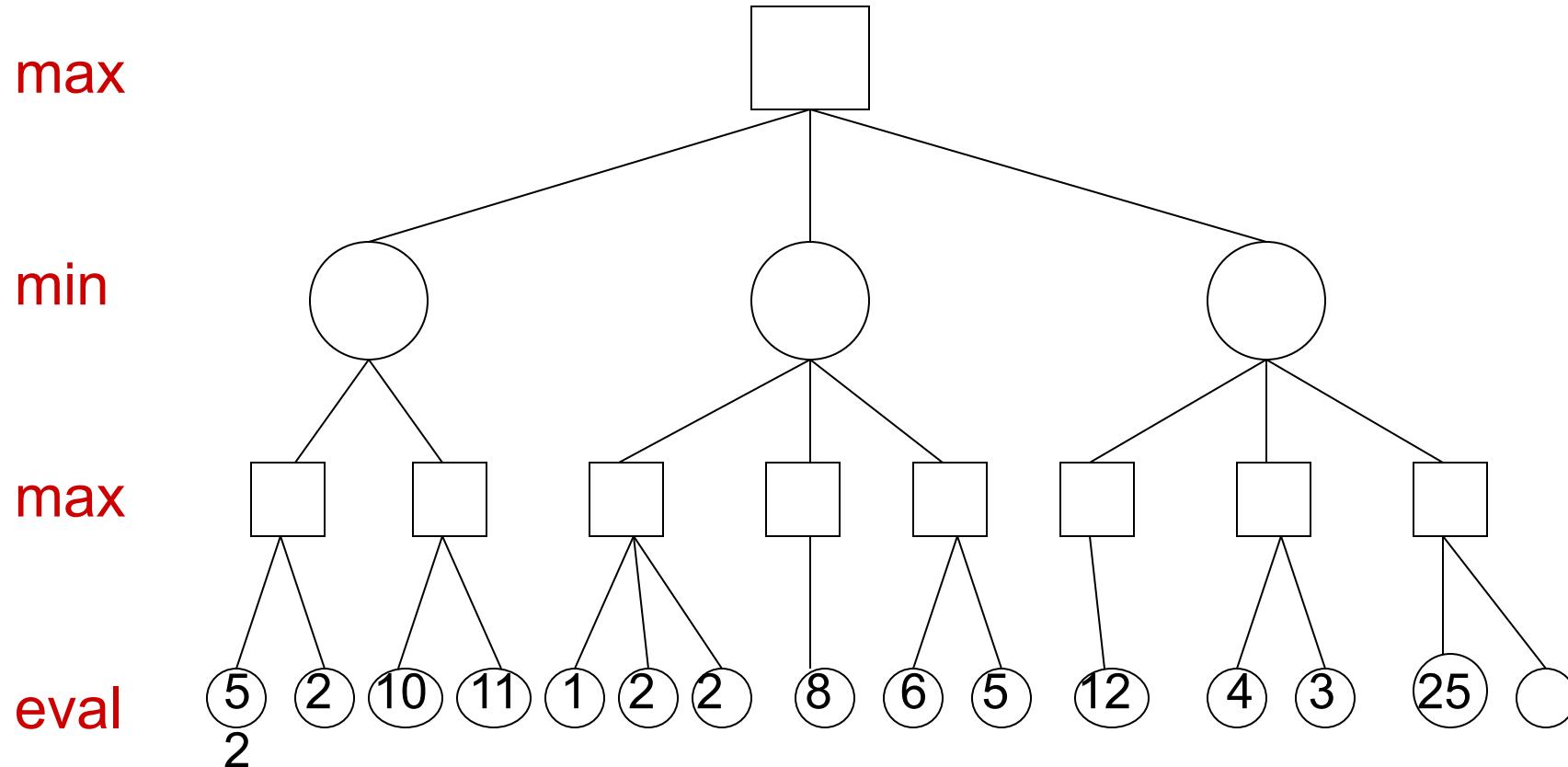


A có  $\beta = 3$  (Trị nút A sẽ không lớn hơn 3) B bị cắt tỉa  $\beta$ , vì  $5 > 3$

C có  $\alpha = 3$  (Trị nút C sẽ không nhỏ hơn 3) D bị cắt tỉa  $\alpha$ , vì  $0 < 3$

E bị cắt tỉa  $\alpha$ , vì  $2 < 3$  Trị nút C là 3

# Alpha-Beta Pruning



Nút MIN có  $\beta \leq \alpha$  của nút cha MAX bất kỳ ( $\alpha$ -cut)  
Nút MAX có  $\alpha \geq \beta$  của nút cha MIN bất kỳ ( $\beta$ -cut)

# Bài toán thỏa mãn ràng buộc

# Một số bài toán thoả mãn ràng buộc thực tế

- Xét bản đồ các bang của nước Úc như hình bên. Cần tô màu các bang với ba màu red, green, blue sao cho hai bang cạnh nhau được tô màu khác nhau



				8				4
	8	4		1	6			
			5			1		
1		3	8			9		
6		8				4	3	
		2			9	5	1	
	7				2			
			7	8		2	6	
2		3						

$$\begin{array}{r} SEND \\ + MORE \\ \hline MONEY \end{array}$$

# Một số bài toán thoả thực tế

- Các bài toán giao nhiệm vụ
  - Ví dụ: Giáo viên nào dạy lớp nào?
- Các bài toán lập thời khóa (gian) biểu
  - Ví dụ: Lớp học nào được dạy vào thời gian nào và ở đâu?
- Các bài toán lập lịch vận tải (giao hàng) của các công ty
- Các bài toán lập lịch sản xuất của các nhà máy

# Bài toán thỏa mãn ràng buộc (Constraint satisfaction problem)

- Bài toán thỏa mãn ràng buộc (CSP): sử dụng phương pháp biểu diễn có cấu trúc để biểu diễn các trạng thái, mỗi trạng thái là một tập biến, mỗi biến có một giá trị
- Bài toán được giải quyết khi mỗi biến đều được gán trị thỏa mãn tất cả ràng buộc
- Một số bài toán được giải quyết nhanh chóng khi mô hình hóa về CSP trong khi không giải quyết được bằng phương pháp tìm kiếm trong không gian trạng thái

# Bài toán thỏa mãn ràng buộc

- Một bài toán thỏa mãn ràng buộc gồm ba thành phần:  $X$ ,  $D$ , và  $C$  trong đó:
  - $X$ : *tập hợp các biến*  $\{X_1, X_2, \dots, X_n\}$
  - $D$ : *tập hợp các miền giá trị*  $\{D_1, D_2, \dots, D_n\}$ ,  $D_i = \{v_1, v_2, \dots, v_k\}$  gán vào biến  $X_i$  tương ứng
  - $C$ : *tập hợp các ràng buộc*,  $C_i$  là một cặp  $\langle \text{scope}, \text{rel} \rangle$  với scope là một bộ biến tham gia vào quan hệ rel (biểu diễn như một danh sách các bộ giá trị tương ứng minh thỏa mãn ràng buộc, hoặc dưới dạng trùu tương ứng phép toán)
- Bài toán CSP được giải quyết khi tất cả các biến đều được gán trị hợp lệ

# Bài toán thỏa mãn ràng buộc

- Giải quyết bài toán CSP bằng cách sử dụng lan truyền ràng buộc
  - *Dùng các ràng buộc để giảm số giá trị hợp lệ cho một biến*
  - *Kết quả các giá trị này lại có thể làm giảm các giá trị hợp lệ cho biến khác...*
- Ý tưởng chính của lan truyền ràng buộc là tính nhất quán cục bộ:
  - *Mỗi biến là một nút trong đồ thị*
  - *Mỗi ràng buộc nhị phân (trên 2 biến) như một cung thì quá trình ép buộc tính nhất quán cục bộ trong mỗi phần của đồ thị tạo ra các giá trị bất hợp lệ cần phải loại bỏ*

# Ví dụ: Bài toán tô màu bản đồ

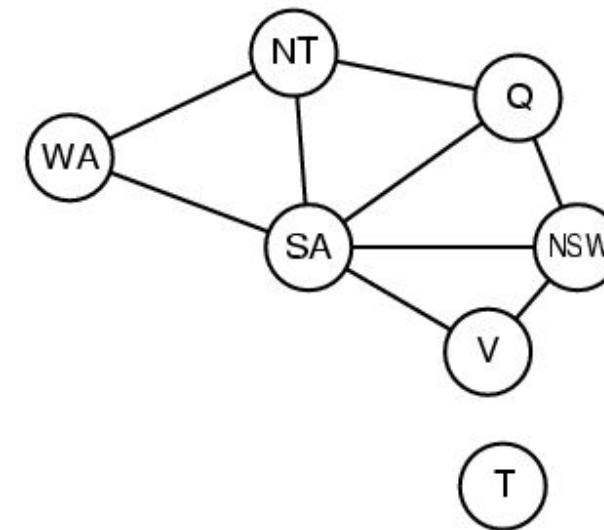
- Xét bản đồ các bang của nước Úc như hình bên. Cần tô màu các bang với ba màu red, green, blue sao cho hai bang cạnh nhau được tô màu khác nhau

- Variables  $WA, NT, Q, NSW, V, SA, T$
- Domains  $D_i = \{\text{red}, \text{green}, \text{blue}\}$
- Ràng buộc: các vùng lân cận có màu khác nhau  
e.g.,  $WA \neq NT$ , or  $(WA, NT) \in \{(red, green), (red, blue), (green, red), (green, blue), (blue, red), (blue, green)\}$

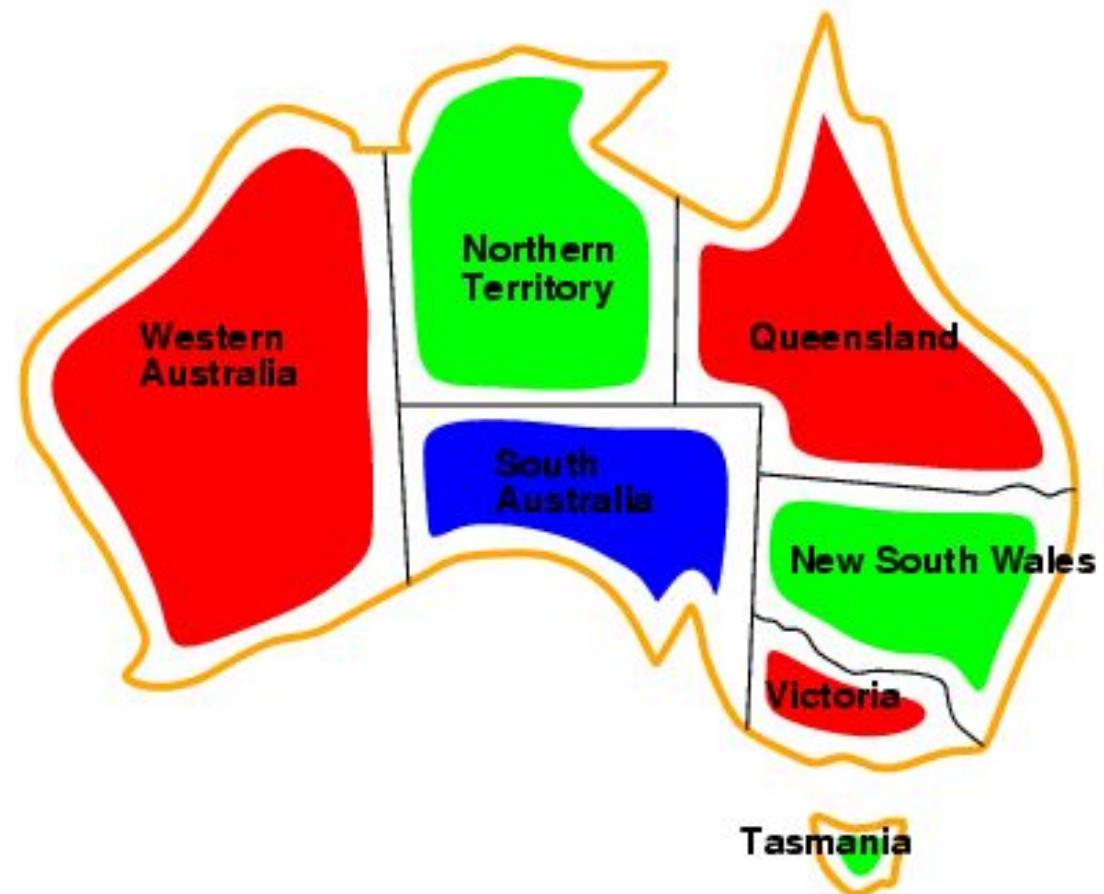


# Ví dụ: Bài toán tô màu bản đồ

- **Đồ thị ràng buộc:** các nút của đồ thị tương ứng với các biển, các cung là các ràng buộc



# Ví dụ: Bài toán tô màu bản đồ



- Giải pháp: WA = red, NT = green, Q = red, NSW = green, V = red, SA = blue, T = green

# Thỏa mãn ràng buộc (Constraint satisfaction)

- *Trạng thái khởi đầu chưa các ràng buộc được cho trong mô tả vấn đề*
- *Một trạng thái đích là trạng thái đã bị ràng buộc « đủ », « đủ » được định nghĩa tùy thuộc vấn đề. Ví dụ, trong câu đó « mật mã số học » « đủ » có nghĩa là mỗi chữ được gán một chữ số duy nhất.*
- Thỏa mãn ràng buộc là quá trình hai bước:
  - *Đầu tiên, các ràng buộc được phát hiện và được lan truyền xa như có thể.*
  - *Sau đó, nếu vẫn chưa có lời giải, bắt đầu tìm kiếm. Đoán một sự kiện, thêm vào ràng buộc mới, lan truyền ràng buộc ...*

# Thỏa mãn ràng buộc (Constraint satisfaction)

1. Lan truyền các ràng buộc sẵn có: Đặt OPEN = tập chứa các đối tượng cần phải gán trị (trong lời giải). Tiến hành các bước sau đến tận khi gặp mâu thuẫn hoặc OPEN rỗng:
  - a) Chọn một đối tượng  $X$  trong OPEN. Tăng cường tập các ràng buộc trên  $X$
  - b) Nếu tập này khác với tập đã được gán cho  $X$  trong lần kiểm tra trước hoặc  $X$  lần đầu tiên được kiểm tra, thêm vào OPEN tất cả các đối tượng chia sẻ các ràng buộc với  $X$
  - c) Xóa  $X$  khỏi OPEN
2. Nếu hợp các ràng buộc được phát hiện ở trên xác định lời giải, thông báo lời giải và thoát
3. Nếu hợp các ràng buộc được phát hiện ở trên xác định một mâu thuẫn, thông báo thất bại

# Thỏa mãn ràng buộc (Constraint satisfaction)

4. Nếu 2. và 3. không xảy ra, Lặp lại đến tận khi tìm thấy một lời giải hoặc tất cả các lời giải có thể bị loại bỏ:
  - a) *Chọn một đối tượng chưa được xác định giá trị, chọn một phương pháp tăng cường ràng buộc trên đối tượng*
  - b) *Gọi đệ quy thỏa mãn ràng buộc với tập hiện hành các ràng buộc được tăng cường thêm qua bước a)*

# Thỏa mãn ràng buộc (Constraint satisfaction)

- Nhiều vấn đề AI có thể được xem như vấn đề thỏa mãn ràng buộc:  
**Đích là phát hiện trạng thái thỏa mãn một tập các ràng buộc đã cho.**

VD: Số học mật mã (CriptArithmetic)

$$\begin{array}{r} \text{S E N D} \\ + \text{M O R E} \\ \hline \text{M O N E Y} \end{array} \quad \begin{array}{l} \text{Ràng buộc:} \\ \bullet \text{ Mỗi chữ tương ứng với một chữ số } 0 .. 9 \\ \bullet \text{ Các chữ khác nhau tương ứng với các chữ số khác nhau} \\ \bullet \text{ Các chữ số làm thỏa mãn phép cộng} \end{array}$$

- Sự thỏa mãn ràng buộc thường làm giảm khối lượng tìm kiếm
- Thỏa mãn ràng buộc là một thủ tục tìm kiếm hoạt động trong không gian các tập ràng buộc:

# Thỏa mãn ràng buộc (Constraint satisfaction)

$$\begin{array}{r} + \quad S \quad E \quad N \quad D \\ M \quad O \quad R \quad E \\ \hline M \quad O \quad N \quad E \quad Y \end{array}$$

Số học mật mã (CriptArithmetic)

Ràng buộc:

- Mỗi chữ tương ứng với một chữ số 0 .. 9
- Các chữ khác nhau tương ứng với các chữ số khác nhau
- Các chữ số làm thỏa mãn phép cộng

Ký hiệu các số mang của cột i là  $C_i$  (  $i = 1, 2, 3, 4$  )

$C_i = 0$  hoặc  $1$

# Thỏa mãn ràng buộc (Constraint satisfaction)

- Các quy tắc để lan truyền ràng buộc sinh ra các ràng buộc sau:
  - *M chỉ có thể là 1 or 0, nhưng M đứng trước nên chỉ có thể là 1*
  - $M = 1$
  - *Phát sinh thêm ràng buộc :  $S + M + C_3 \leq 19$*

*Hai số bất kỳ + thêm số dư từ hàng trăm mang sang  $\leq 19$*

*ví dụ như: 9+8+1*

$$\begin{array}{r} \text{S E N D} \\ \text{M O R E} \\ \hline \text{M O N E Y} \end{array}$$

# Thỏa mãn ràng buộc (Constraint satisfaction)

- Các quy tắc để lan truyền ràng buộc sinh ra các ràng buộc sau:

- $M = 1$  ( $S + M + C_3 \leq 19$ )
- Vì  $S + M$  có số nhớ (dư) để mang sang hàng chục nghìn nên  $S + M + C_3 > 9$
- Ta có  $M=1 \Rightarrow S+ C_3 > 8$
- Mà  $C_3 = 0$  hoặc  $1$
- $C_3 = 0$  thì  $S+ 0 > 8 \Rightarrow S= 9$
- $C_3 = 1$  thì  $S+ 1 > 8 \Rightarrow S > 7 \Rightarrow S= 8$  hoặc  $S=9$

$$\begin{array}{r} \text{S E N D} \\ \text{M O R E} \\ \hline \text{M O N E Y} \end{array}$$

# Thỏa mãn ràng buộc (Constraint satisfaction)

- Các quy tắc để lan truyền ràng buộc sinh ra các ràng buộc sau:

- $M = 1$  ( $S + M + C_3 \leq 19$ )

- $S = 8$  hoặc  $9$  ( $S + M + C_3 > 9$ ,  $C_3 \leq 1$ ,  $M=1$ )

- $S + M + C_3 = 10 + O$

với  $M = 1$ :  $S + C_3 = 9 + O$

Và  $S \leq 9$ ,  $C_3 \leq 1$  nên  $O \leq 1$

vì  $M$  đã bằng 1 nên  $O = 0$

( ~~$9+1=9+O$~~ ;  $9=9+O$ ;  $8+1=9+O$ ;  ~~$8+0=9+O$~~ )

$$\begin{array}{r} \boxed{S} \quad E \quad N \quad D \\ M \quad O \quad R \quad E \\ \hline \boxed{M} \quad O \quad N \quad E \quad Y \end{array}$$

# Thỏa mãn ràng buộc (Constraint satisfaction)

- Các quy tắc để lan truyền ràng buộc sinh ra các ràng buộc sau:
  - $M = 1$  ( $S + M + C_3 \leq 19$ )
  - $S = 8$  hoặc  $9$  ( $S + M + C_3 > 9$ ,  $C_3 \leq 1$ ,  $M=1$ )
  - $O = 0$  ( $S + M + C_3 = 10 + O$ , với  $M=1$ ,  $S \leq 9$ ,  
 $C_3 \leq 1$  nên  $O \leq 1$ )
  - $S = 9$ ,  $C_3 = 0$  (vì nếu  $C_3 = 1$  thì  $E = 9$  và  $N = 0$   
mà  $O = 0$ )
  - $N = E+1$  (vì  $N = E + O + C_2 = E + C_2$  nên  $N = E$   
 $\frac{S \quad E \quad N \quad D}{M \quad O \quad R \quad E}$   
 $\hline$  $M \quad O \quad N \quad E \quad Y$ )
  - $C_2 = 1$

# Thỏa mãn ràng buộc (Constraint satisfaction)

- Để tiến triển thực hiện « phương pháp đoán »:

$E = 5;$

.....

- Kết quả:

$S = 9, E = 5, N = 6, D = 7, M = 1, O = 0, R = 8, Y = 2$

$$\begin{array}{r} 9567 \\ + 1085 \\ \hline 10652 \end{array}$$

$$\begin{array}{r} SEND \\ MORE \\ \hline MONEY \end{array}$$

Cần xếp lịch dạy của 3 giảng viên cho 5 lớp học có lịch gần nhau, bao gồm:

- Lớp 1: Phân tích thiết kế thuật toán, lịch học 8:00 - 09:00AM
- Lớp 2: Nhập môn AI, lịch học 8:30 - 09:30AM
- Lớp 3: Cơ sở dữ liệu, lịch học 9:00 - 10:00AM
- Lớp 4: Hệ điều hành, lịch học 9:00 - 10:00AM
- Lớp 5: Nguyên lý máy học, lịch học 9:30 - 10:30AM

Biết rằng giảng viên chỉ có thể dạy 1 lớp tại một thời điểm và thông tin về giảng viên lần lượt là:

- GV A có thể dạy lớp 3 và 4
- GV B có thể dạy lớp 2, 3, 4, và 5
- GV C có thể dạy lớp 1, 2, 3, 4, 5

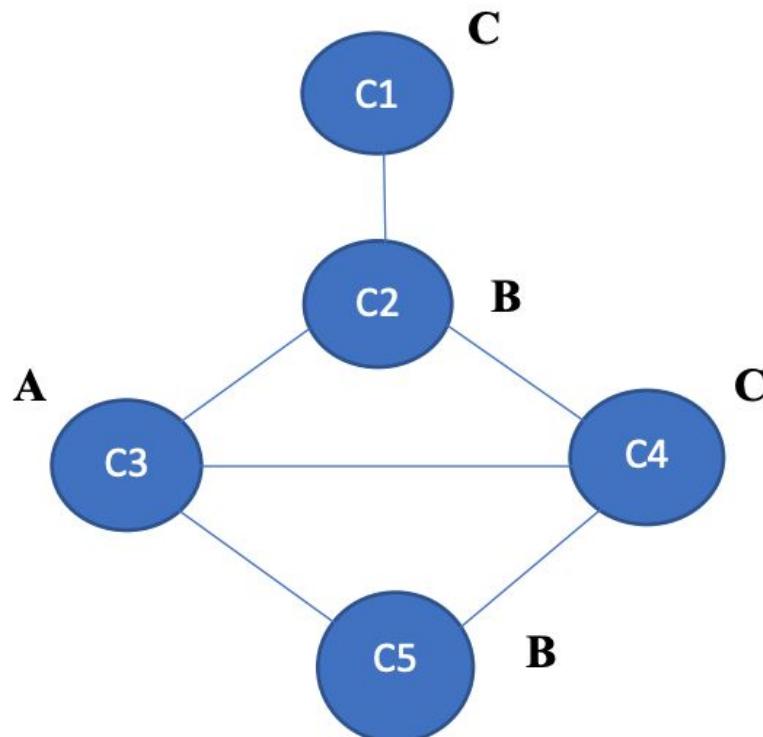
Hãy giải quyết vấn đề trên sử dụng giải thuật thỏa mãn ràng buộc với gợi ý các biến lần lượt là các lớp học, xác định miền giá trị cho tập biến và các ràng buộc cần thiết

+ **Variables** (Biến): C1, C2, C3, C4, C5.

+ **Domains** (Giá trị): C1={C}; C2={B,C}; C3={A,B,C}; C4={A,B,C}; C5={B,C}.

+ **Ràng buộc**: Giảng viên chỉ dạy 1 lớp tại một thời điểm

+ e.g: C1 ≠ C2, C2 ≠ C3, C3 ≠ C4, C4 ≠ C5, C2 ≠ C4, C3 ≠ C5.



Lớp 1: Phân tích thiết kế thuật toán, lịch học 8:00 - 09:00AM  
Lớp 2: Nhập môn AI, lịch học 8:30 - 09:30AM  
Lớp 3: Cơ sở dữ liệu, lịch học 9:00 - 10:00AM  
Lớp 4: Hệ điều hành, lịch học 9:00 - 10:00AM  
Lớp 5: Nguyên lý máy học, lịch học 9:30 - 10:30AM

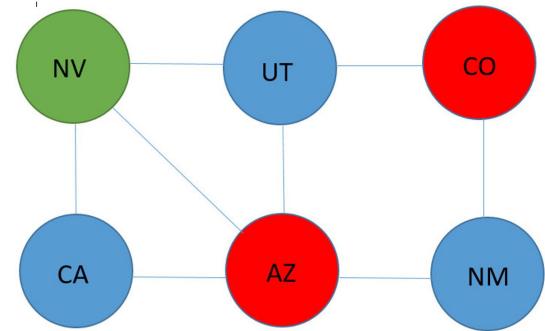
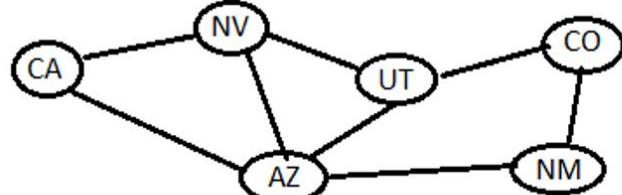
GV A có thể dạy lớp 3 và 4  
GV B có thể dạy lớp 2, 3, 4, và 5  
GV C có thể dạy lớp 1, 2, 3, 4, 5

Sử dụng 3 màu: R (red), G(green), B(blue) để tô màu bản đồ bên dưới với yêu cầu các vùng lân cận phải sử dụng các màu khác nhau



1. Hãy xây dựng đồ thị ràng buộc tương ứng với bản đồ đã cho
2. Giả sử gán AZ là R, xác định 1 giải pháp tô màu có thể cho các đỉnh còn lại

Đồ thị ràng buộc

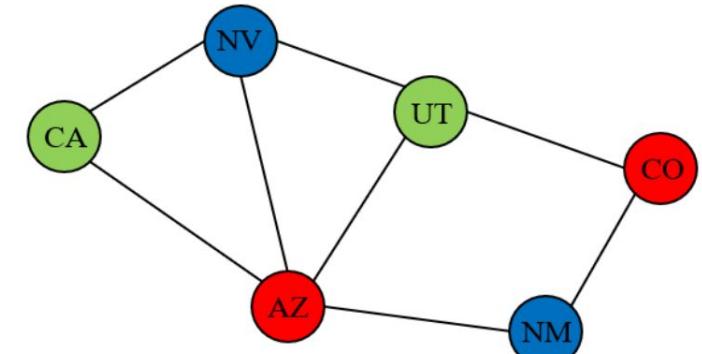
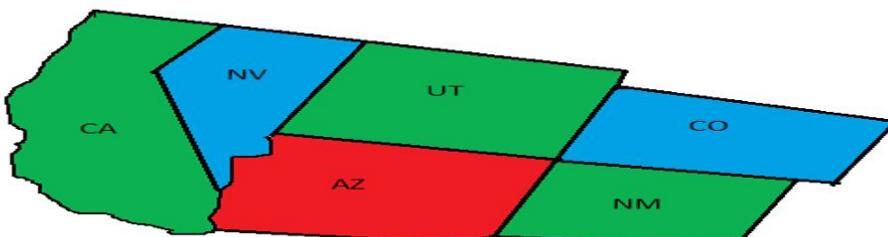


Variables: CA,NV,AZ,UT,NM,CO.

Variables' domain: $D_{CA} = D_{NV} = D_{AZ} = D_{UT} = D_{CO} = D_{NM} = \{R,G,B\}$ .

Constraints:  $AZ \neq CA$ ;  $AZ \neq NV$ ;  $AZ \neq UT$ ;  $AZ \neq NM$ ;  $CA \neq NV$ ;  $NV \neq UT$ ;  $UT \neq CO$ ;  $CO \neq NM$

Giải pháp:  $AZ=R$ ,  $CA=G$ ,  $NV=B$ ,  $UT=G$ ,  $CO=B$ ,  $MN=G$



*The End*

- Variables: GV A, GV B, GV C
- Domains: Lớp 1, Lớp 2, Lớp 3, Lớp 4, Lớp 5
- Ràng buộc:
  1. GV A có thể dạy lớp 3 và 4
  2. GV B có thể dạy lớp 2, 3, 4 và 5
  3. GV C có thể dạy lớp 1, 2, 3, 4 và 5
  4. GV chỉ có thể dạy 1 lớp tại một thời điểm về GV lần lượt là

Giải

- **Lớp 1:** GV C (vì chỉ có GV C có thể dạy lớp 1) **8:00 – 9:00**
- **Lớp 2:** GV B (chỉ có GV B và GV C mới có thể dạy lớp 2, mà 8:00 – 9:00 GV C đang dạy lớp 1) **8:30 – 9:30**
- **Lớp 3:** GV C (chỉ có GV B và GV C mới có thể dạy lớp 3, mà 8:30 – 9:30 GV B đang dạy lớp 2) **9:00 – 10:00**
- **Lớp 4:** GV A (vì 9:00-10:00 GV C đang dạy lớp 3, 8:30-9:30 GV B đang dạy lớp 2) **9:00 – 10:00**
- **Lớp 5:** GV B (vì 9:00-10:00 GV A đang dạy lớp 4 và GV C đang dạy lớp 3) **9:30 – 10:30**

Tập hợp các biến X = {C1, C2, C3, C4, C5} với Ci là Lópi

Tập hợp các miền giá trị D = {D1, D2, D3, D4, D5} với

$$D1 = \{C\}$$

$$D2 = \{B, C\}$$

$$D3 = \{A, B, C\}$$

$$D4 = \{A, B, C\}$$

$$D5 = \{B, C\}$$

Tập hợp các ràng buộc

$$C: \{C1 \neq C2, C2 \neq C3, C2 \neq C4, C3 \neq C4, C3 \neq C5, C4 \neq C5\}$$

- C1 chỉ có giá trị C

- C2 khác C1 => C2 có giá trị B

- C3 và C4 đều khác C2 nên C3 và C4 chỉ nhận giá trị A hoặc C  
=> C3 có giá trị A

- C4 khác C2 và C3 => C4 có giá trị C

- C5 khác C3 và C4 => C4 có giá trị B

GV A có thể dạy lớp 3 và 4

GV B có thể dạy lớp 2, 3, 4, và 5

GV C có thể dạy lớp 1, 2, 3, 4, 5

Vậy Lớp 1 giáo viên C, Lớp 2 giáo viên B, Lớp 3 giáo viên A, Lớp 4 giáo viên C,  
Lớp 5 giáo viên B

Lớp 1: Phân tích thiết kế thuật toán, lịch học 8:00 - 09:00AM

Lớp 2: Nhập môn AI, lịch học 8:30 - 09:30AM

Lớp 3: Cơ sở dữ liệu, lịch học 9:00 - 10:00AM

Lớp 4: Hệ điều hành, lịch học 9:00 - 10:00AM

Lớp 5: Nguyên lý máy học, lịch học 9:30 - 10:30AM

Sử dụng 3 màu: R (red), G(green), B(blue) để tô màu bản đồ bên dưới với yêu cầu các vùng lân cận phải sử dụng các màu khác nhau



1. Hãy xây dựng đồ thị ràng buộc tương ứng với bản đồ đã cho
2. Giả sử gán AZ là R, xác định 1 giải pháp tô màu có thể cho các đỉnh còn lại

### **Giải pháp cho câu 1:**

$$CA = R, NV = G, UT = R, AZ = B, CO = G, NM = R$$

### **Giải pháp cho câu 2:**

$$CA = B, NV = G, UT = B, AZ = R, CO = G, NM = B$$

Sử dụng 3 màu: R (red), G(green), B(blue) để tô màu bản đồ bên dưới với yêu cầu các vùng lân cận phải sử dụng các màu khác nhau



1. Hãy xây dựng đồ thị ràng buộc tương ứng với bản đồ đã cho
2. Giả sử gán AZ là R, xác định 1 giải pháp tô màu có thể cho các đỉnh còn lại

+ **Variables** (Biên ): CA, NV, AZ, UT, CO, NM.

+ **Domains** (Giá trị):  $D_i = \{\text{red}, \text{green}, \text{blue}\}$

+ **Ràng buộc**: Các vùng lân cận có màu khác nhau.

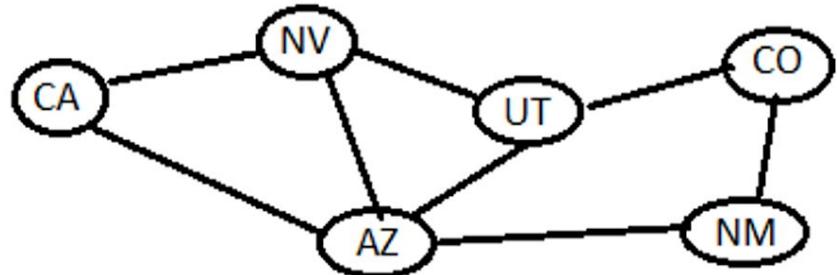
+ **e.g**:  $CA \neq NV$ ,  $CA \neq AZ$ ,  $NV \neq UT$ ,  $NV \neq AZ$ ,  $AZ \neq UT$ ,  $AZ \neq NM$ ,  $UT \neq CO$ ,  $CO \neq NM$ .

+ **Giải Pháp**: { **CA = red**, **NT = green**, **AZ=blue**, **UT = red**, **CO= blue** , **NM = green** }

- Miền giá trị:
  - R – Red
  - B – Blue
  - G – Green
- Đặt biến:
  - C1: AZ
  - C2: NV
  - C3: CA
  - C4: UT
  - C5: CO
  - C6: NM
- Ràng buộc:
  - C1 – **R**
  - $C2 \neq C1 \text{ & } C3, C4 \Rightarrow C2 - \textcolor{blue}{B}$
  - $C3 \neq C1 \text{ & } C2 \Rightarrow C3 - \textcolor{green}{G}$
  - $C4 \neq C1 \text{ & } C2 \Rightarrow C4 - \textcolor{green}{G}$
  - $C5 \neq C4 \text{ & } C6 \Rightarrow C5 - \textcolor{red}{R}$
  - $C6 \neq C5 \text{ & } C1 \Rightarrow C6 - \textcolor{blue}{B}$
- Kết luận : AZ = R ; NV = B ; CA = G; UT = G; CO = R ; NM = B



Đồ thị ràng buộc



Variables: CA,NV,AZ,UT,NM,CO.

Variables' domain:  $D_{CA} = D_{NV} = D_{AZ} = D_{UT} = D_{CO} = D_{NM} = \{R,G,B\}$ .

Constraints:  $AZ \neq CA$ ;  $AZ \neq NV$ ;  $AZ \neq UT$ ;  $AZ \neq NM$ ;  $CA \neq NV$ ;  $NV \neq UT$ ;  $UT \neq CO$ ;  $CO \neq NM$

# Uniform-Cost (UCS)

- Let  $g(n)$  = cost of the path from the start node to an open node  $n$
- Algorithm outline:
  - *Always select from the OPEN the node with the **least  $g(.)$  value** for expansion, and put all newly generated nodes into OPEN*
  - *Nodes in OPEN are sorted by their  $g(.)$  values (in ascending order)*
  - *Terminate if a node selected for expansion is a goal*
- Called “*Dijkstra's Algorithm*” in the algorithms literature and similar to “*Branch and Bound Algorithm*” in operations research literature

# Uniform-Cost Search

GENERAL-SEARCH(problem, ENQUEUE-BY-PATH-COST)

**exp. node**   **nodes list**

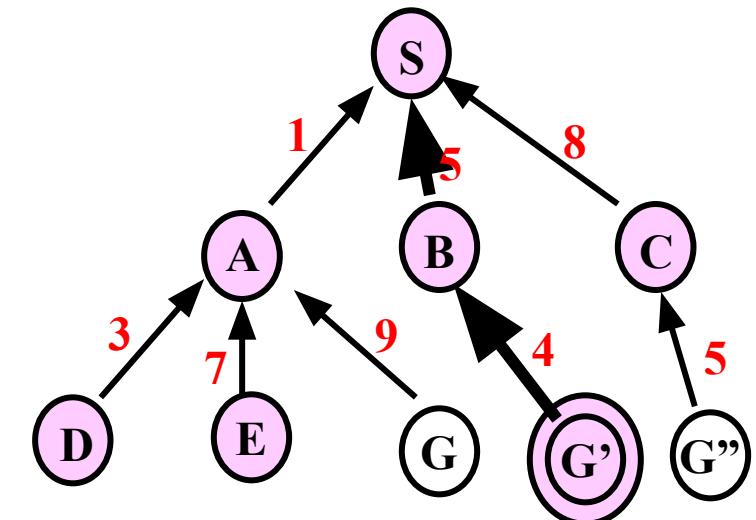
**CLOSED list**

{S(0)}

S    {A(1) B(5) C(8)}

Solution path found is S B G <-- this G has cost 9, not 10

Number of nodes expanded (including goal node) = 7



# Uniform-Cost Search

GENERAL-SEARCH(problem, ENQUEUE-BY-PATH-COST)

exp. node nodes list

CLOSED list

{S(0)}

S {A(1) B(5) C(8)}

A {D(4) B(5) C(8) E(8) G(10)}

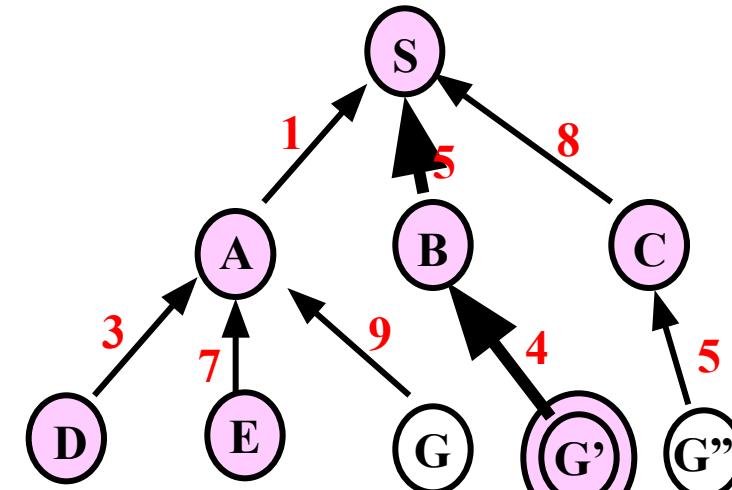
D {B(5) C(8) E(8) G(10)}

B {C(8) E(8) G'(9) G(10)}

C {E(8) G'(9) G(10) G"(13)}

E {G'(9) G(10) G"(13) }

G' {G(10) G"(13) }



Solution path found is S B G <-- this G has cost 9, not 10

Number of nodes expanded (including goal node) = 7

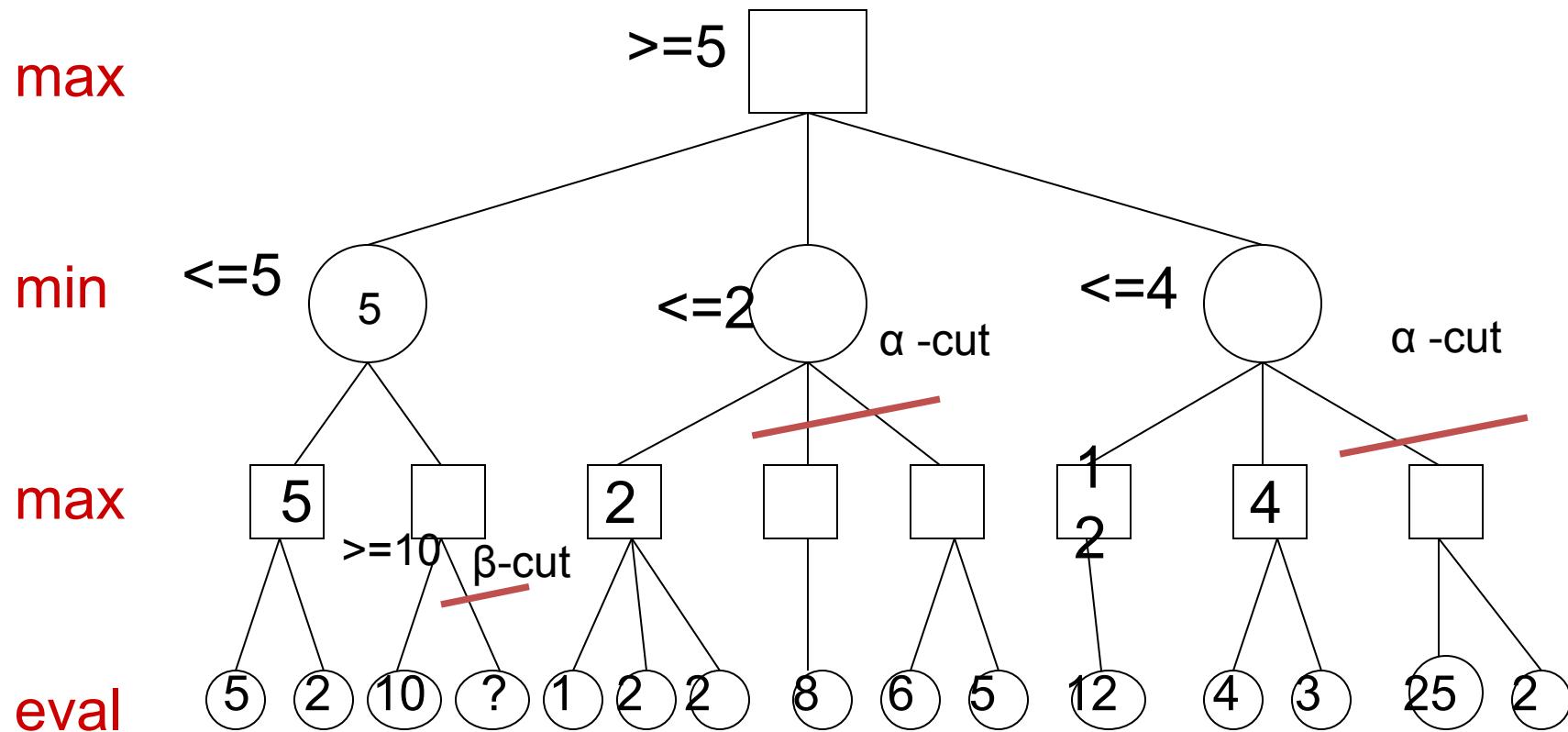
# Tìm kiếm tốt nhất đầu tiên (Best-first-search)

Frontier – Open; Explored=Closed

```
PNode bestFirstSearch(PState init_state) {  
    PNode root = new Node();  
    root->state = init_state;  
    root->parent = NULL;  
    root->f = 0;  
    frontier.insert(root);  
    explored.clear();  
    while (!empty(frontier)) {  
        //Lấy 1 nút từ đường biên có f(n) nhỏ nhất  
        //và loại bỏ nó ra khỏi đường biên  
        Node* node = frontier.pop();  
        if (node là nút mục tiêu)  
            return node;  
        insert(node, explored);
```

```
        for (child là nút con của node) {  
            Tính child->f;  
            if (child->state không thuộc frontier và  
                child->state không thuộc explored) {  
                child->parent = node;  
                frontier.insert(child);  
                (*) else if (child->state nằm trong đường biên  
                    và có f lớn hơn child->f)  
                    Thay thế nút nằm trên đường biên bằng child  
                (***) else if (child->state nằm trong explored  
                    và có f lớn hơn child->f) {  
                    Loại bỏ nút chứa child->state  
                    ra khỏi explored  
                    frontier.insert(child);  
                }  
            }  
        }  
        return NULL; //Thất bại, không tìm thấy lời giải  
    }
```

# Alpha-Beta Pruning



Nút MIN có  $\beta \leq \alpha$  của nút cha MAX bất kỳ ( $\alpha$ -cut)  
Nút MAX có  $\alpha \geq \beta$  của nút cha MIN bất kỳ ( $\beta$ -cut)