

# CHƯƠNG 3: CÁC KỸ THUẬT TÌM KIẾM HEURISTIC

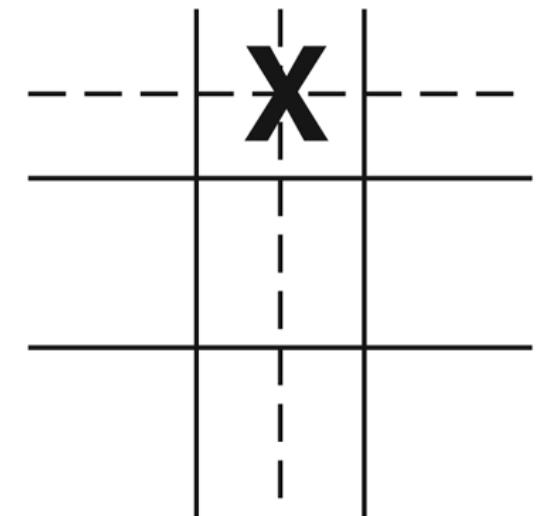
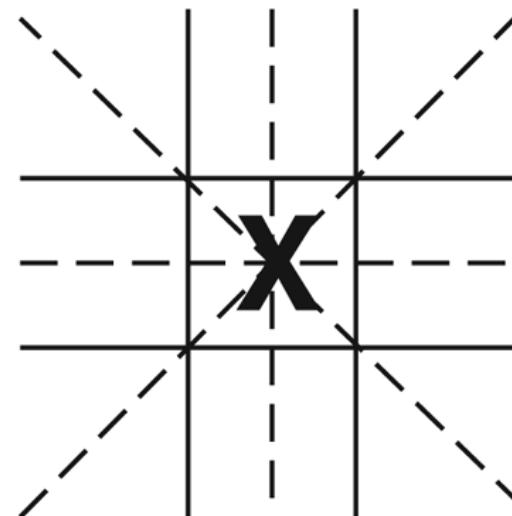
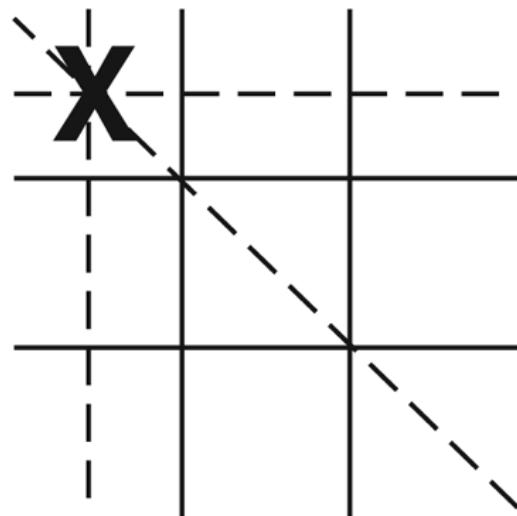
# Ôn tập – Các kiến thức đã học

1. Biểu diễn bài toán trong KGTT
2. Tìm kiếm mù (uninformed search)
  - 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)

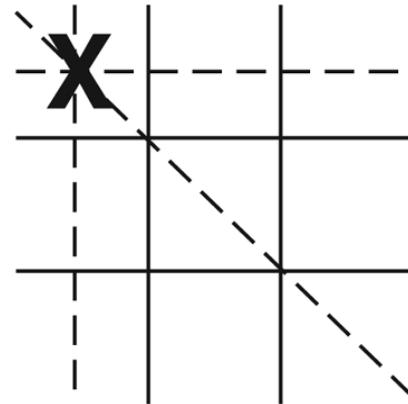
# NỘI DUNG

1. Giới thiệu Heuristic
2. Các kỹ thuật tìm kiếm Heuristic
  - Tìm kiếm tốt nhất đầu tiên (Best-first-search)
    - Tìm kiếm háu ăn (Greedy best-first-search)
    - Giải thuật A\*
  - Leo đồi (Hill climbing)
  - Thỏa mãn ràng buộc (Constraint satisfaction)

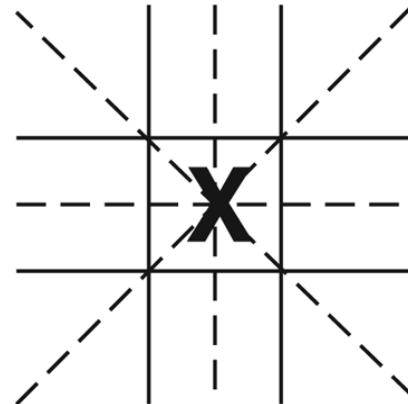
# Giới thiệu Heuristic



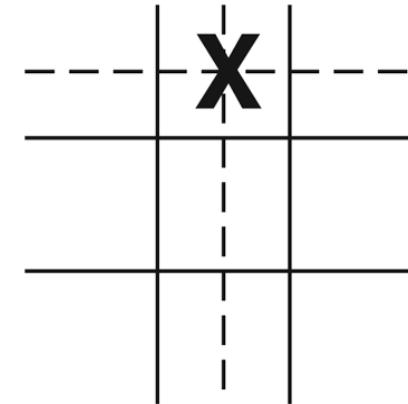
# Giới thiệu Heuristic (††)



Three wins through  
a corner square



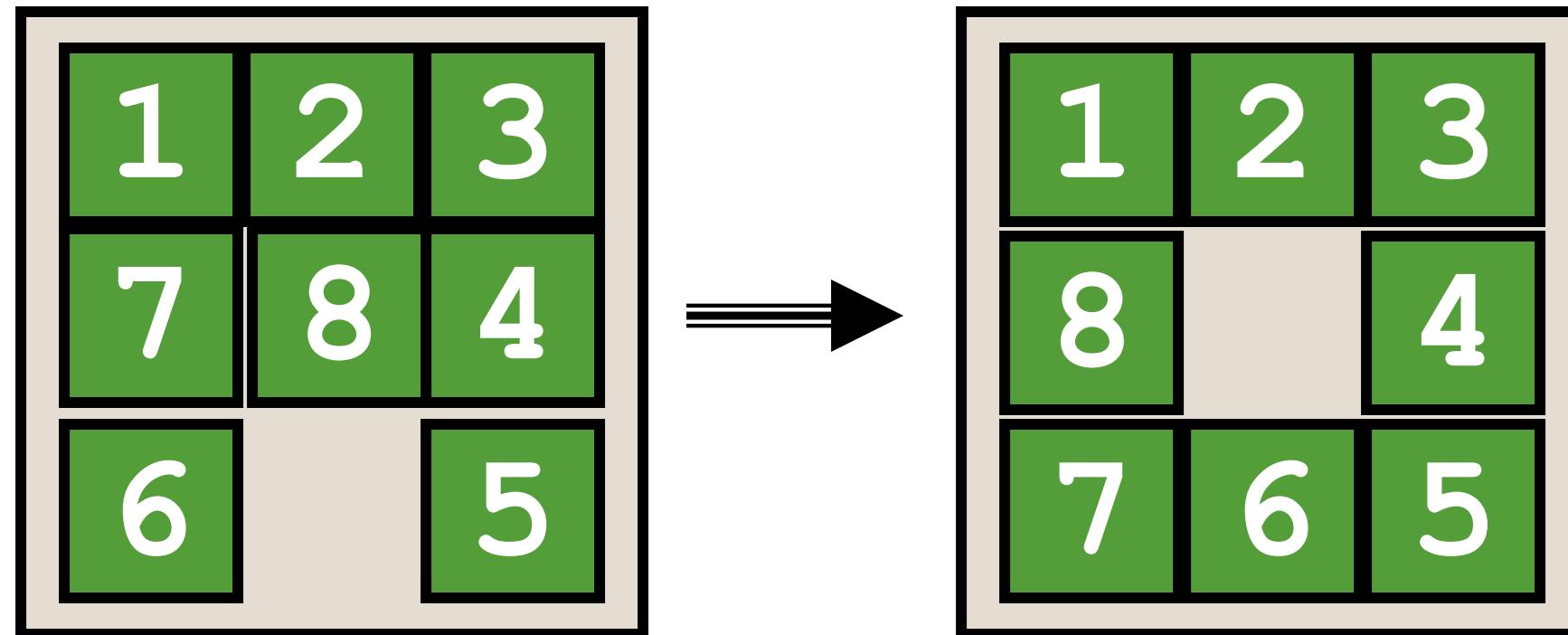
Four wins through  
the center square



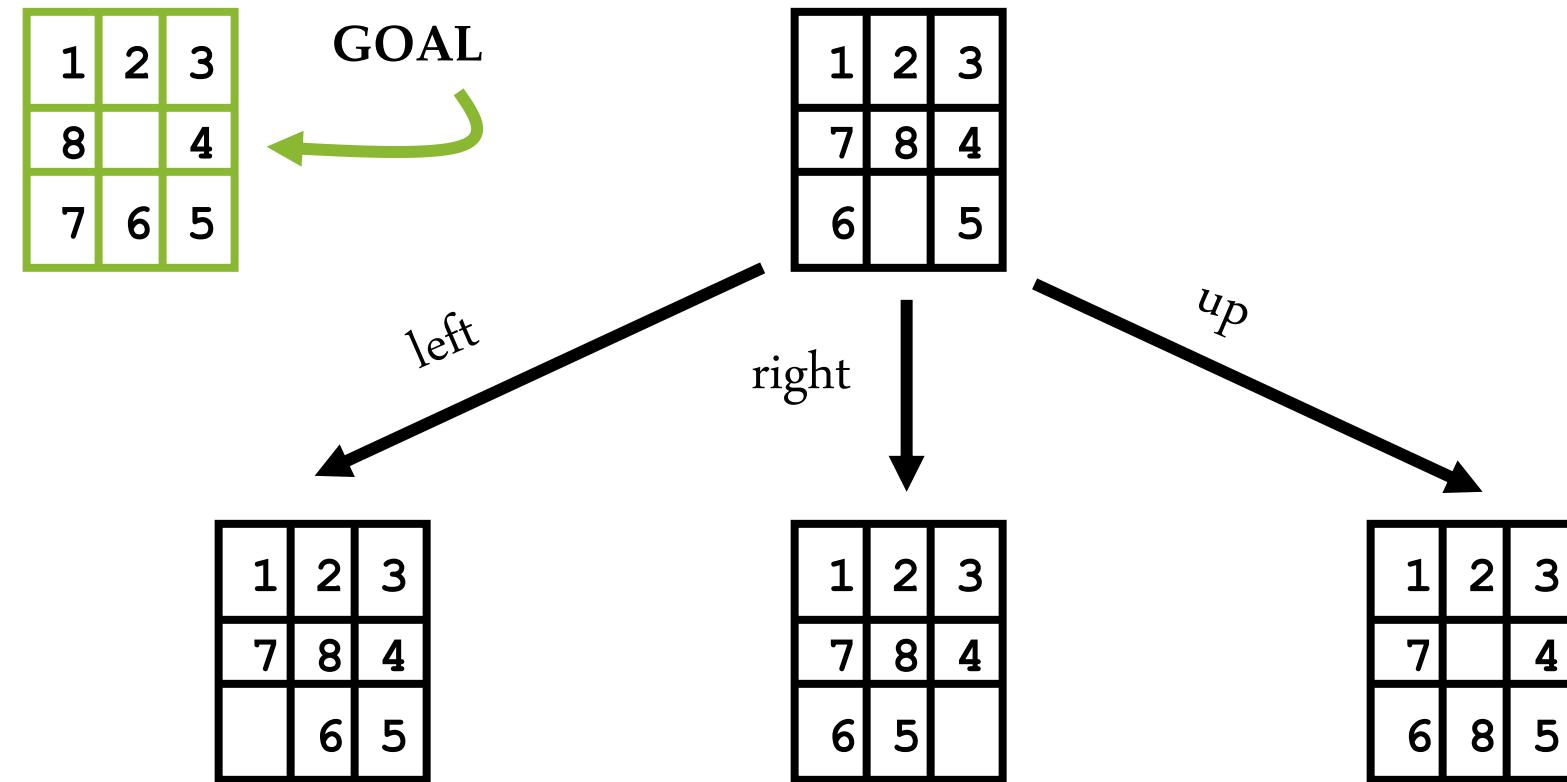
Two wins through  
a side square

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

# Giới thiệu Heuristic (tt)



# Heuristic



Di chuyển nào là **tốt nhất?**

# 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 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 diễn đạt vấn đề hoặc trong các dữ liệu có sẵn. VD: *chẩn đoán y khoa*,...

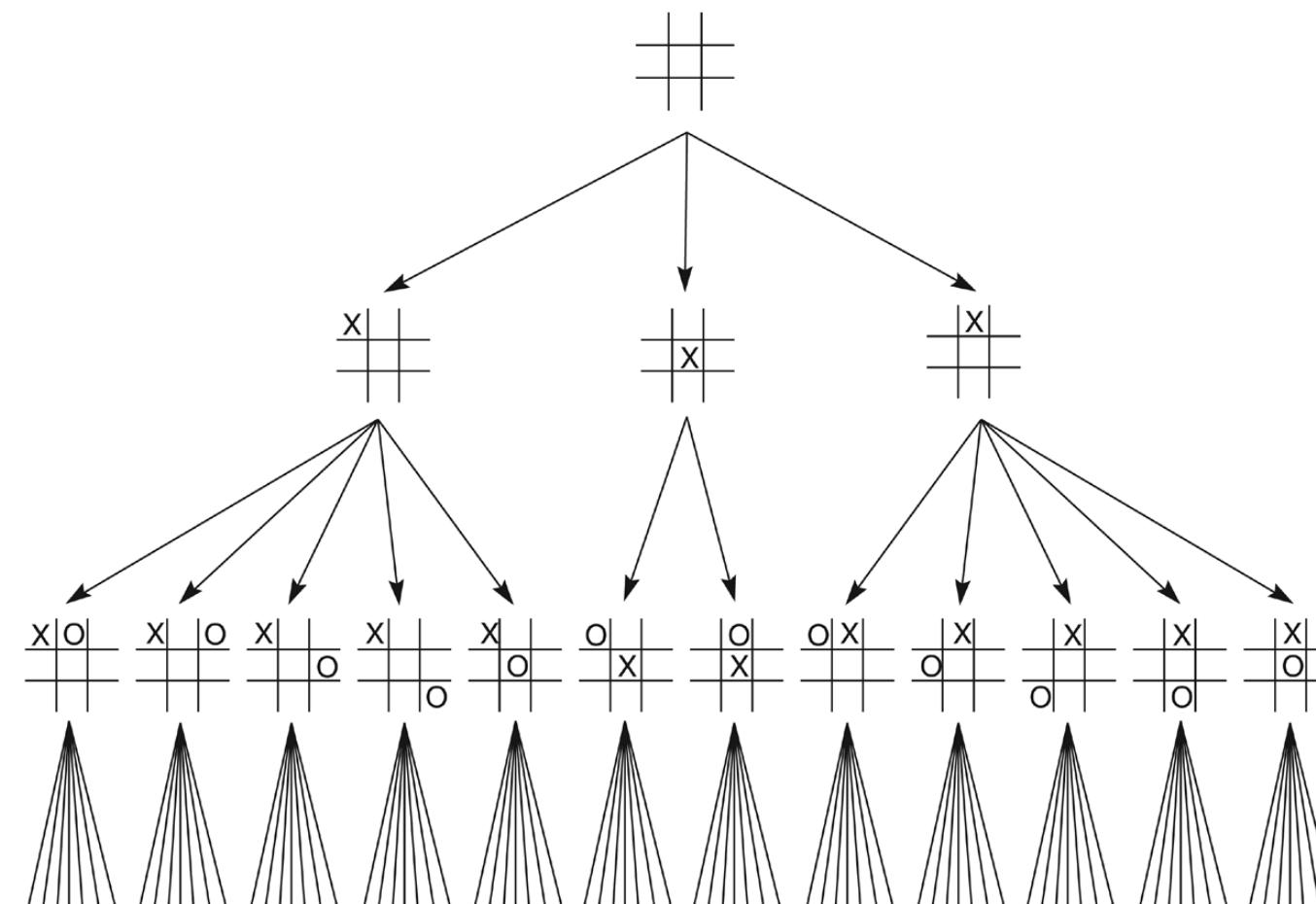
# Heuristic

- 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...
    - ❑ Tìm kiếm leo đồi
    - ❑ ...

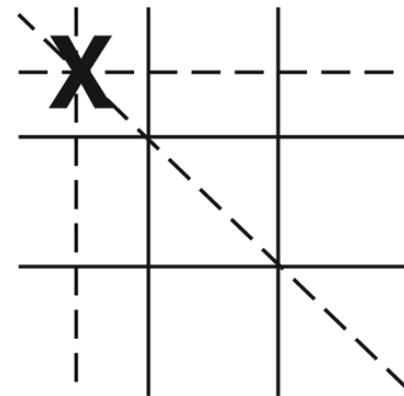
# Heuristic

- 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)

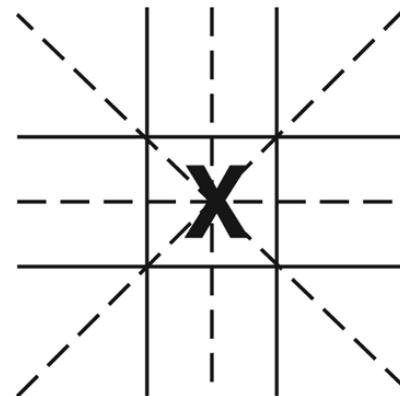
**KGTT của tic-tac-toe được thu nhỏ nhờ tính đối xứng của các trạng thái.**



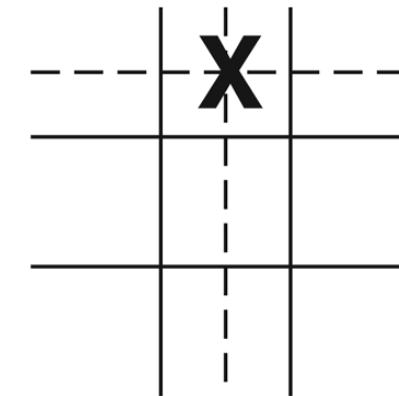
## KGTT của tic-tac-toe được thu nhỏ nhờ tính đối xứng của các trạng thái.



Three wins through  
a corner square



Four wins through  
the center square



Two wins through  
a side square

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

# Cài đặt hàm đánh giá Heuristic

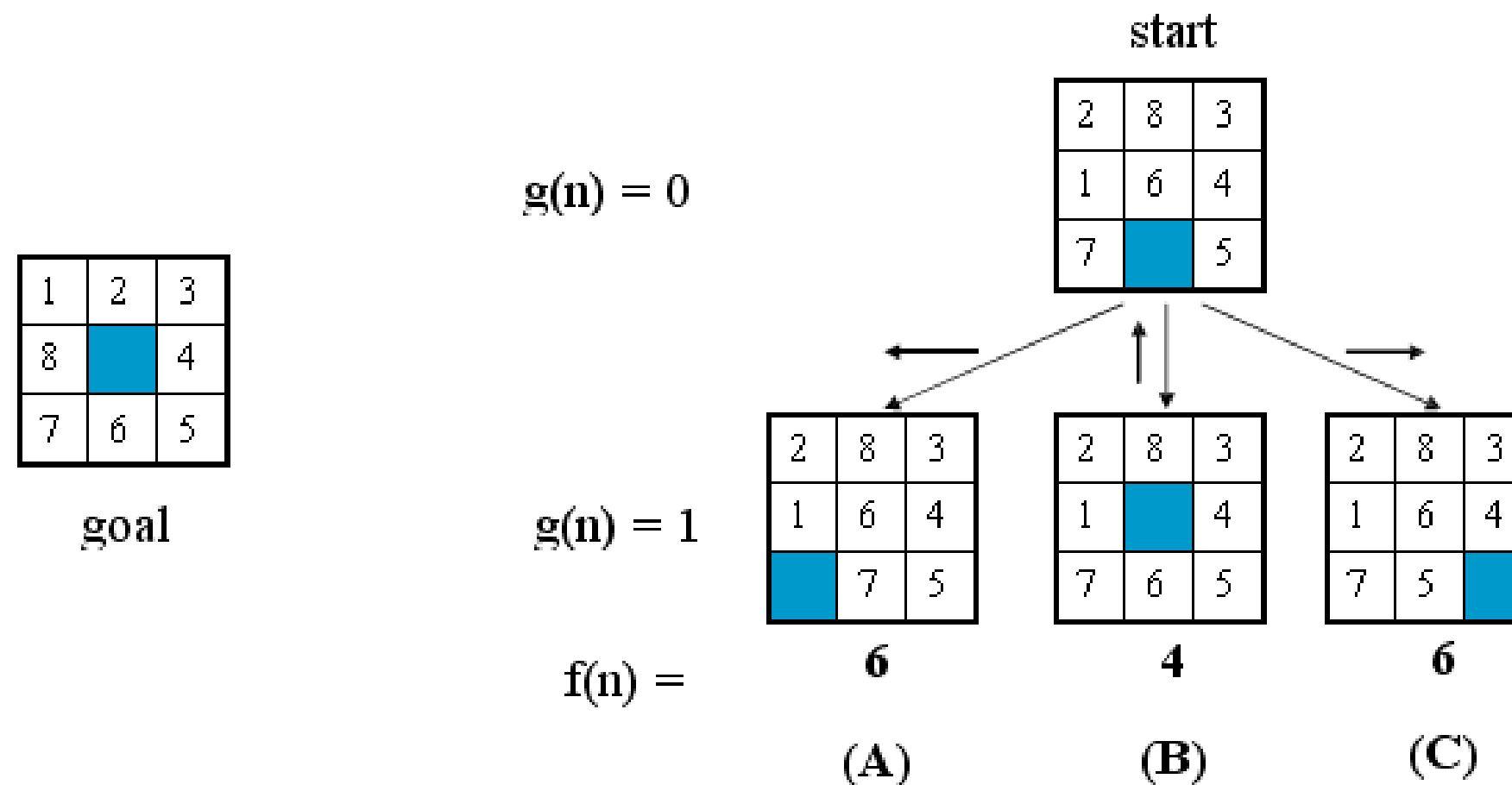
- Xét trò chơi 8-puzzle. Hàm đánh giá Heuristic tại trạng thái n là  $f(n)$ :

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

- $g(n)$  = 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 **dích**

# Cài đặt hàm đánh giá Heuristic

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



# 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) (trường hợp bỏ qua ô trống)

$\Rightarrow h_1(n) = ???$

START

6	4	2
3		5
0	1	7

Solution Solve Randomize

$\Rightarrow h_2(n) = ???$

GOAL

0	1	2
3	4	5
6	7	

Solution Solve Randomize

# 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) (trường hợp bỏ qua ô trống)

$$\Rightarrow h_1(n) = 5$$

$$\Rightarrow h_2(n) = 8$$

START

6	4	2
3		5
0	1	7
Solution	Solve	Randomize

GOAL

0	1	2
3	4	5
6	7	
Solution	Solve	Randomize

# Cài đặt hàm đánh giá Heuristic (tt)

Bài tập: tính  $h_1(n)$  và  $h_2(n)$

7	2	4
5		6
8	3	1

Trạng thái bắt đầu

	1	2
3	4	5
6	7	8

Trạng thái mục tiêu

# Cài đặt hàm đánh giá Heuristic (tt)

Bài tập: tính  $h_1(n)$  và  $h_2(n)$

7	2	4
5		6
8	3	1

Trạng thái bắt đầu

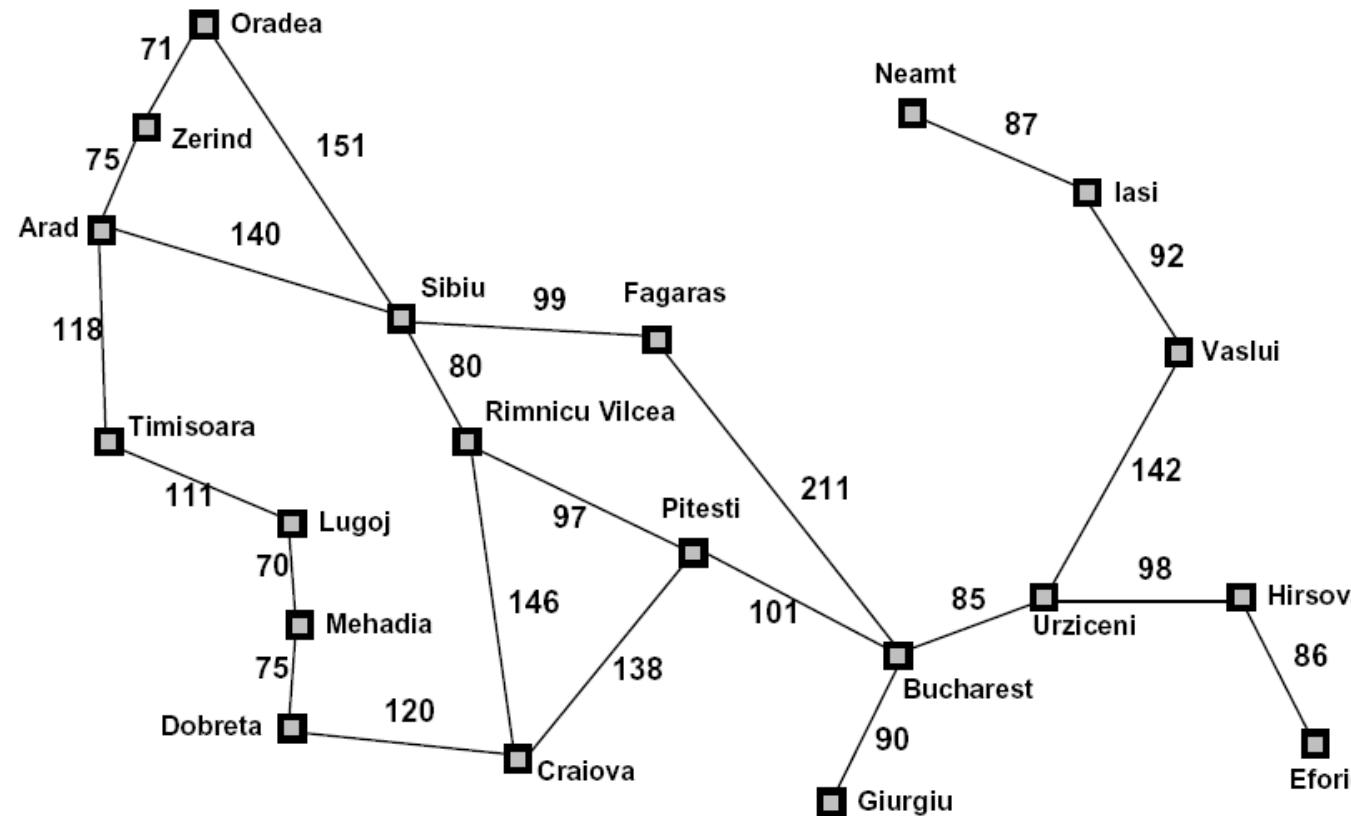
	1	2
3	4	5
6	7	8

Trạng thái mục tiêu

$$\Rightarrow h_1(n) = 8$$

$$\Rightarrow h_2(n) = 3 + 1 + 2 + 2 + \\ 2 + 3 + 3 + 2 = 18$$

# Ví dụ về hàm heuristic ( $h$ )

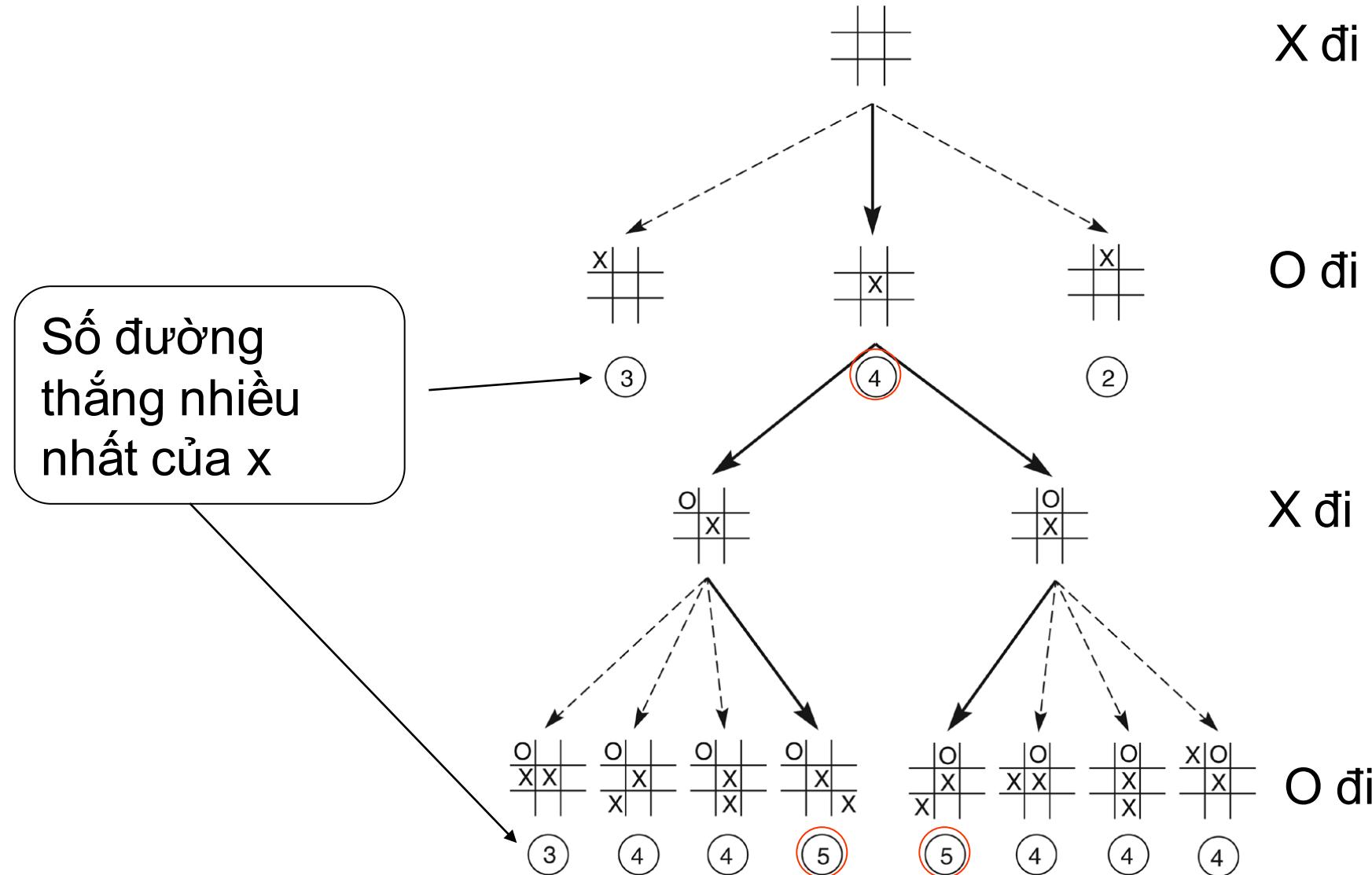


Straight-line distance  
to Bucharest

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

$h(x)$ : khoảng cách chim bay từ  
các thành phố đến Bucharest

# KGTT càng thu nhỏ khi áp dụng heuristic



# Các kỹ thuật tìm kiếm Heuristic

- Một số phương pháp được sử dụng trong kỹ thuật tìm kiếm Heuristic:
  1. Tìm kiếm tốt nhất đầu tiên (Best-first-search)
    - Tìm kiếm háu ăn (greedy best first search)
    - Giải thuật A\*
  2. Leo đồi (Hill climbing)
  3. Thỏa mãn ràng buộc (Constraint satisfaction)

# 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 đánh giá* (evaluation function):  $f(n)$

# 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.**

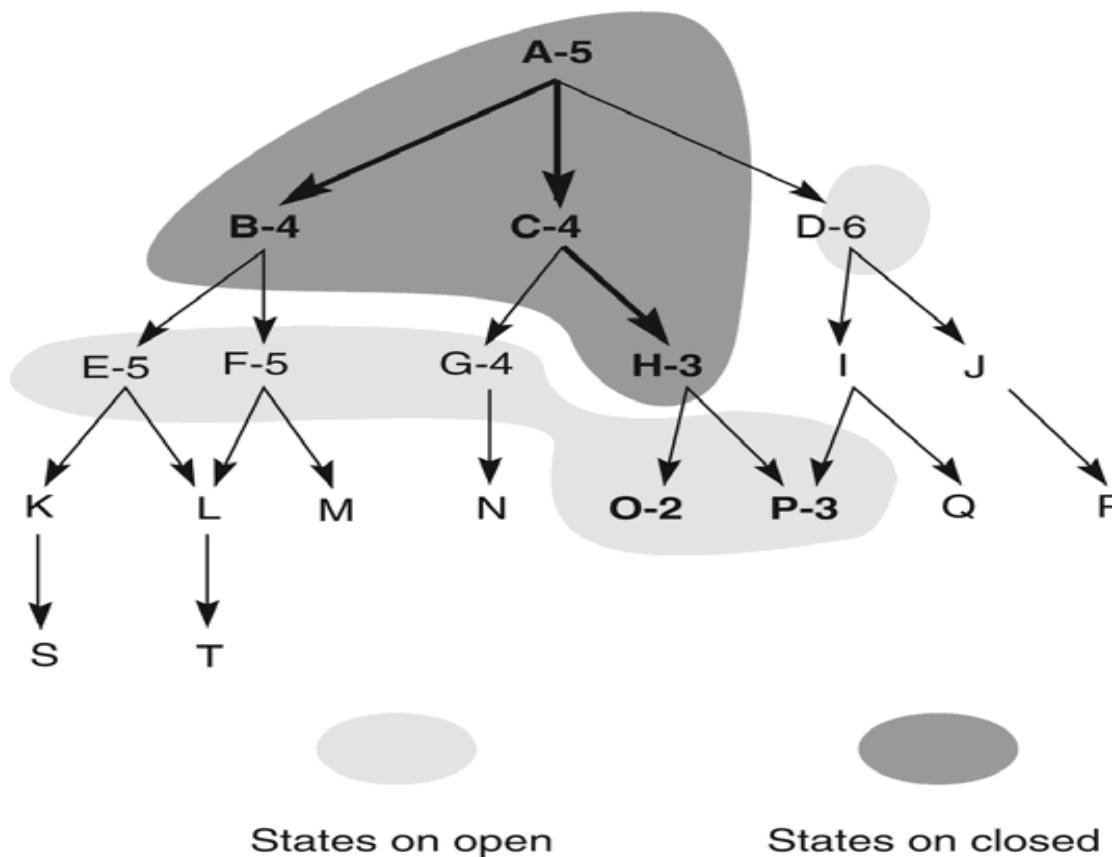
# Tìm kiếm tốt nhất đầu tiên (Best-first-search)

```
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  
}
```

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

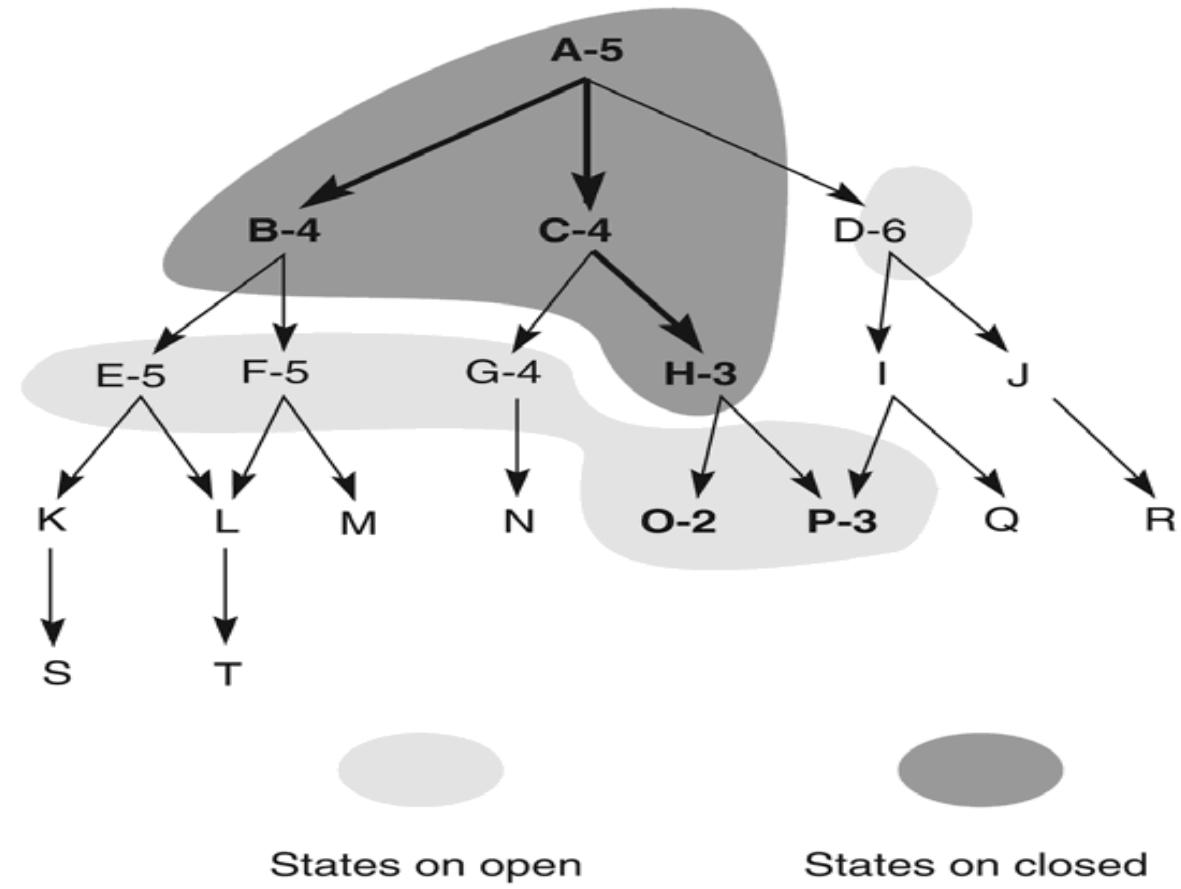
- Xét đồ thị KGTT sau:



- Giá trị cạnh mỗi nút cho biết giá trị ước lượng độ tốt của nút đó trong không gian
- Giá trị thấp nhất là tốt nhất
- Nút đích cần tìm kiếm là P

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

1. open = [A5]; closed = []
2. Đánh giá A5;  
open = [B4,C4,D6];  
closed = [A5]
3. Đánh giá B4;  
open = [C4,E5,F5,D6];  
closed = [B4,A5]
4. Đánh giá C4;  
open = [H3,G4,E5,F5,D6];  
closed = [C4,B4,A5]
5. Đánh giá H3;  
open = [O2,P3,G4,E5,F5,D6];  
closed = [H3,C4,B4,A5]
6. Đánh giá O2;  
open = [P3,G4,E5,F5,D6];  
closed = [O2,H3,C4,B4,A5]
7. Đánh giá P3;  
Tìm được lời giải!

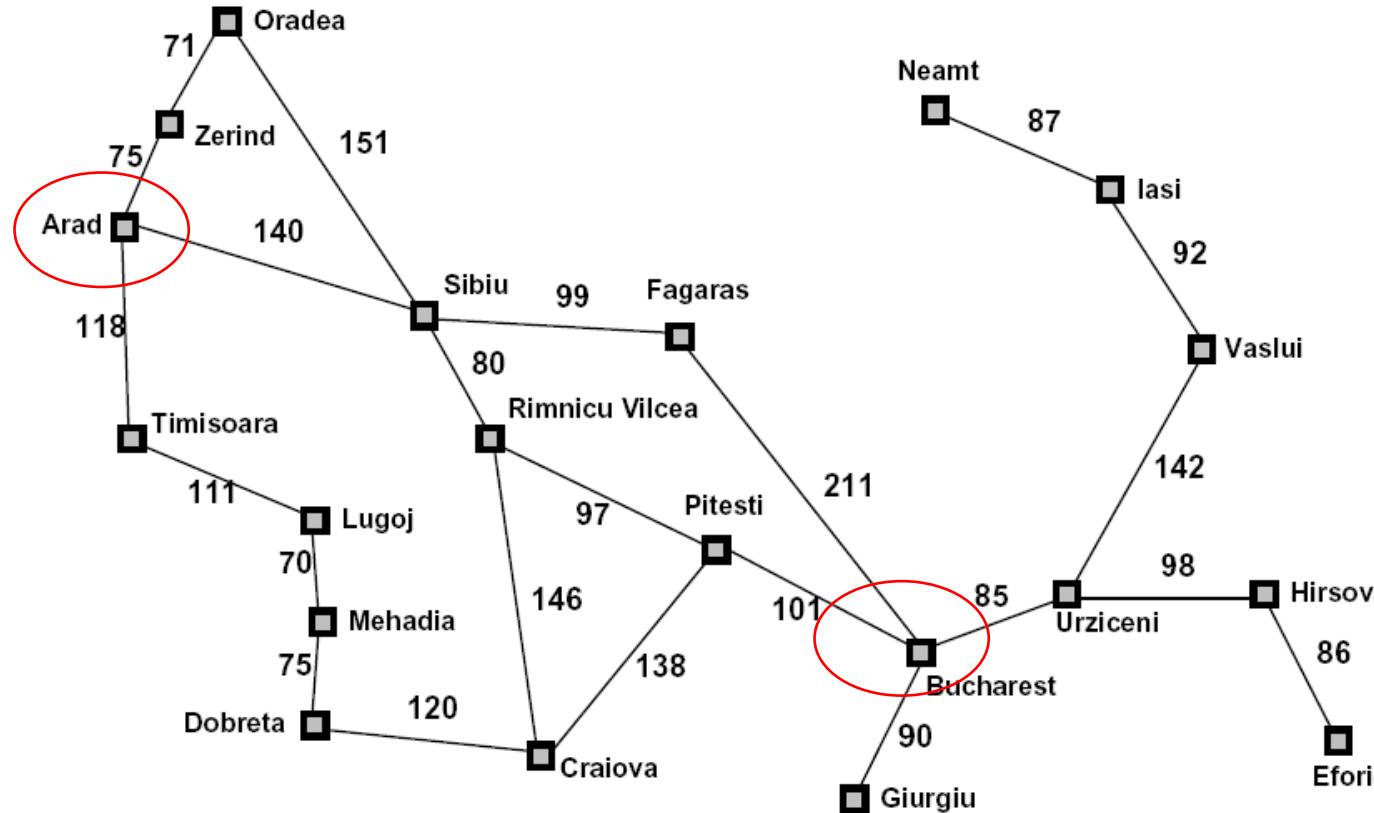


# Tìm kiếm háu ăn (greedy best-first search)

- 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 luợng giá các nút hay:

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

# Ví dụ về tìm kiếm h้าu ăn



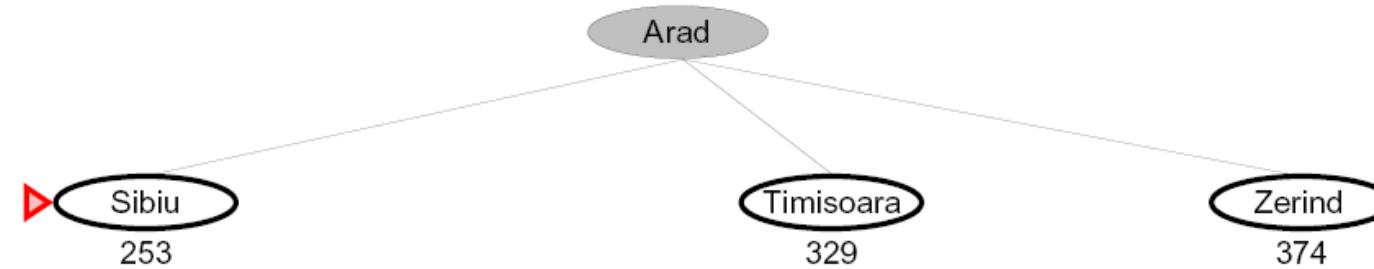
Straight-line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

$h(x)$ : khoảng cách chim bay từ các thành phố đến Bucharest

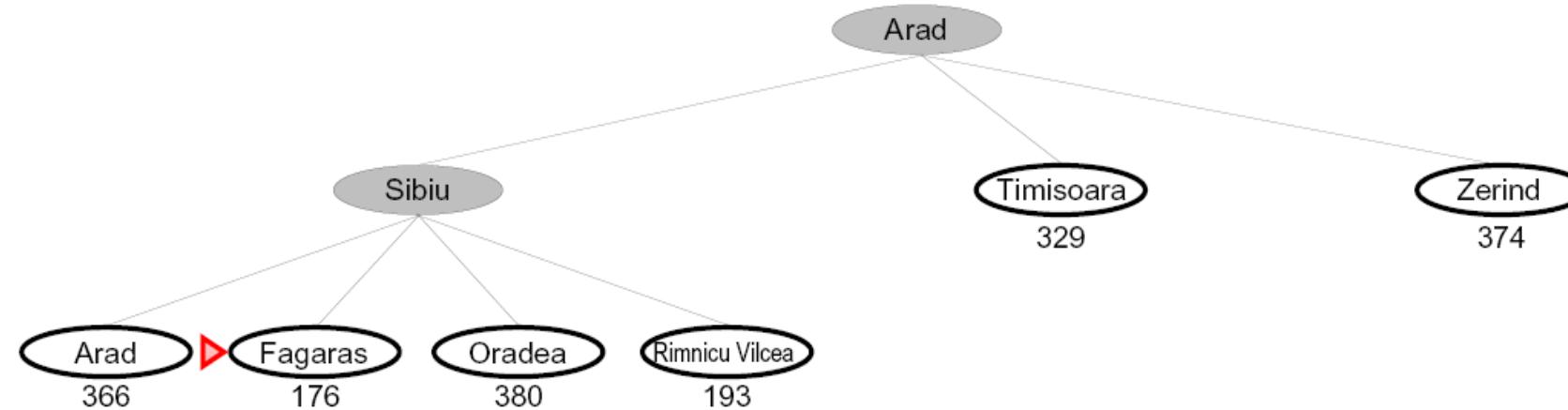
# Ví dụ về tìm kiếm háu ăn

► Arad  
366

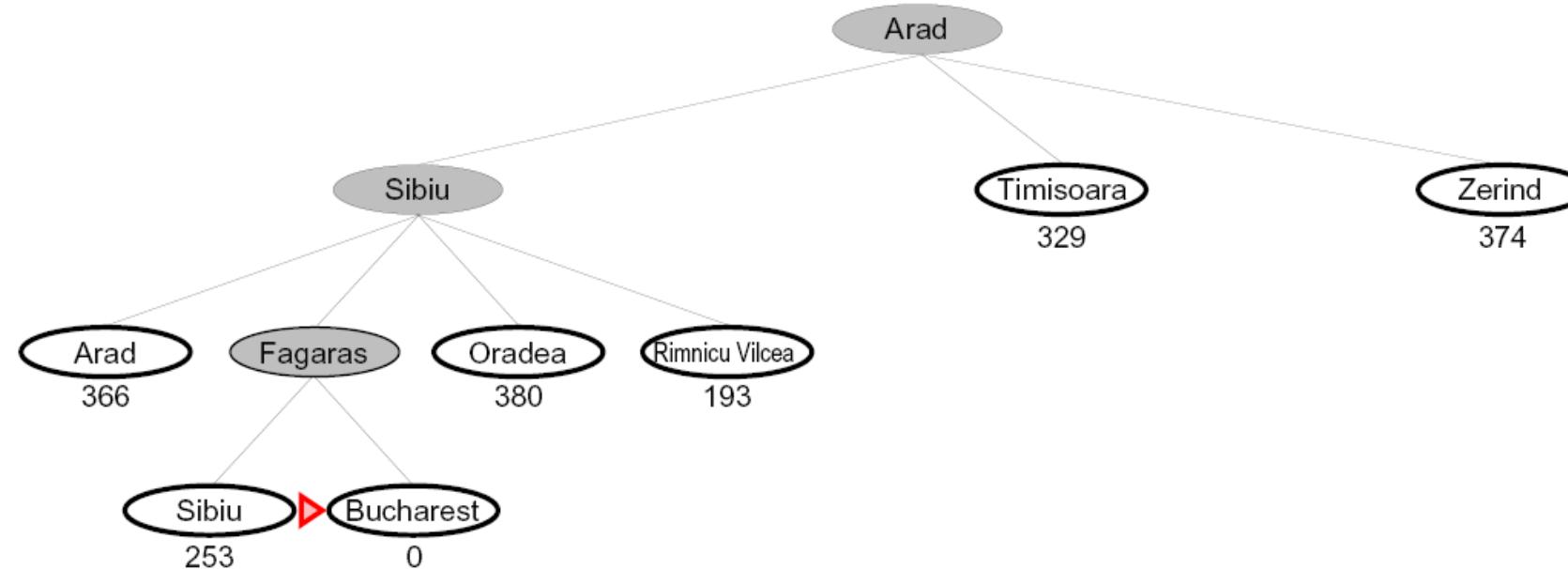
# Ví dụ về tìm kiếm h้าu ăn



# Ví dụ về tìm kiếm h้าu ăn

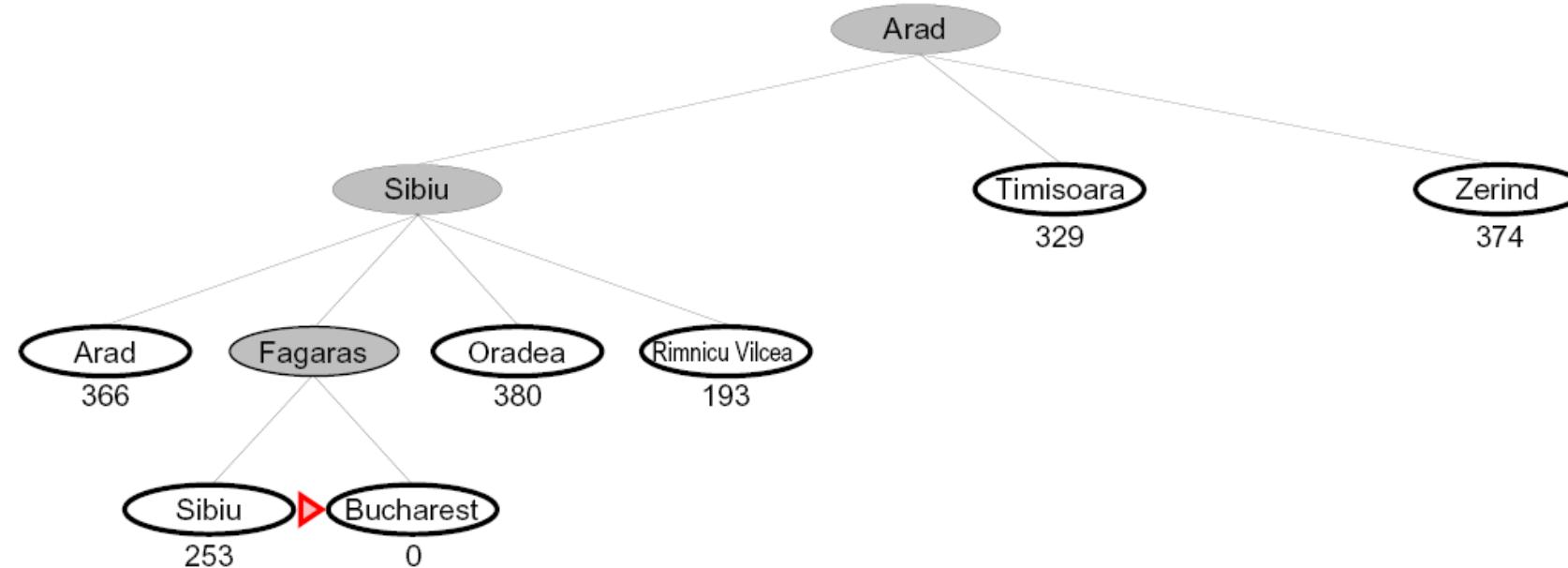


# Ví dụ về tìm kiếm h้าu ăn



Lời giải : A -> S -> ????? (chi phí: ??????)

# Ví dụ về tìm kiếm h้าu ăn



Lời giải : A -> S -> ????? (chi phí: ??????)

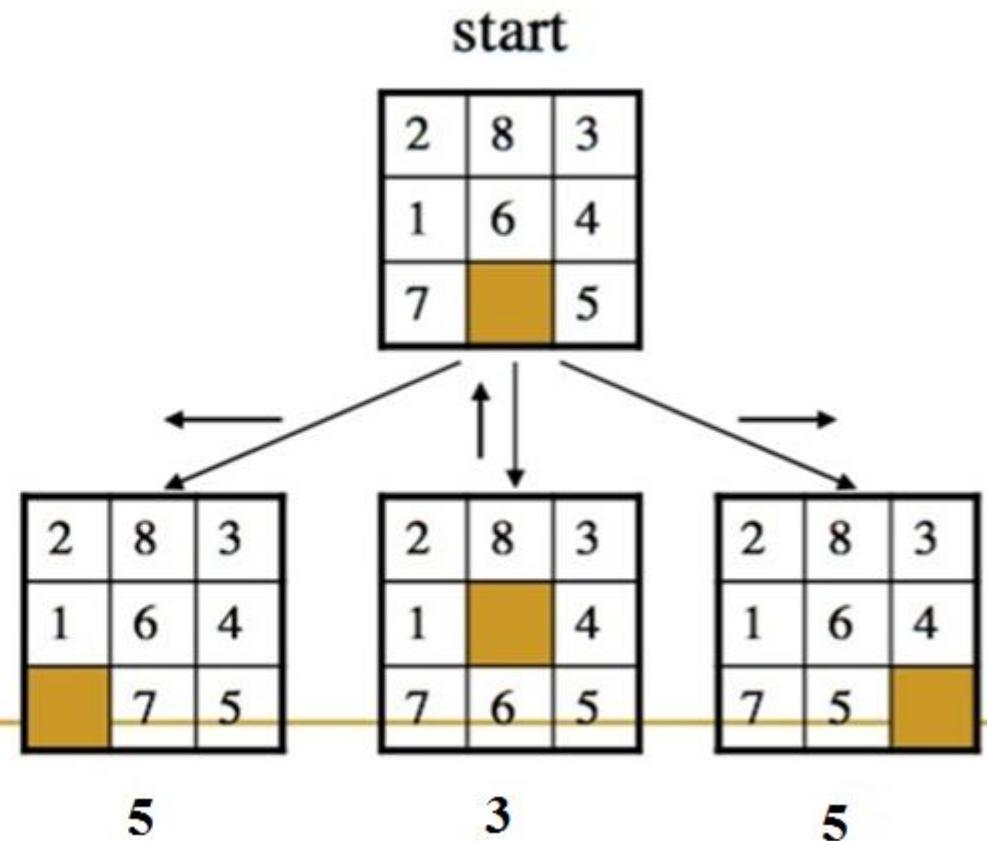
# Sử dụng tìm kiếm háu ăn để tìm trạng thái goal

1	2	3
8		4
7	6	5

goal

$h(n)$ : số lượng các vị trí còn sai;

$$f(n) =$$



# Giải thuật A\*



UCS



Greedy

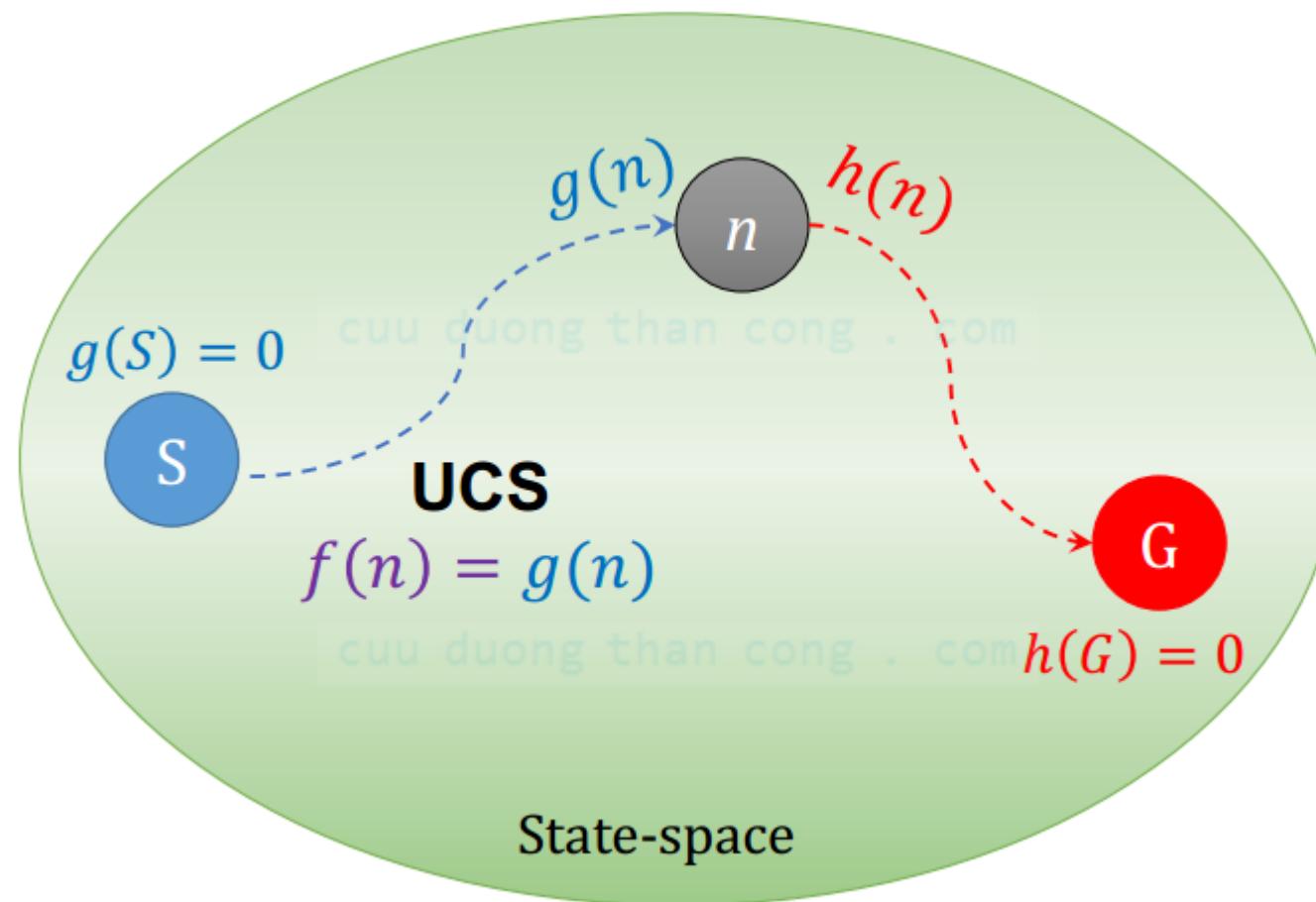


A\*

# Giải thuật A\*

- Giải thuật A\* là trường hợp đặc biệt Best first search (việc cập nhật lại đường đi dựa trên giá trị  $h(n)$  thay vì dựa trên giá trị  $f(n)$  tổng quát)
  - $f(n) = g(n) + h(n)$
  - $g(n)$  khoảng cách thực sự từ  $n$  đến **trạng thái bắt đầu**
  - **$h(n)$  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 => cập nhật lại  $g$  khi đường đi mới tốt hơn)
- Mỗi trạng thái  $n$  tùy ý sẽ gồm 4 yếu tố ( $g(n)$ ,  $h(n)$ ,  $f(n)$ ,  $cha(n)$ ) với  $cha(n)$  là nút cha của nút đang xét  $n$

# Hàm chi phí và hàm ước lượng heuristic



## Mã giả giải thuật A\*

$g(n_o)=0$ ;  $f(n_o)=h(n_o)$ ;

open:=[ $n_o$ ]; closed:=[ ];

**while** open $\neq$ [ ] **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**

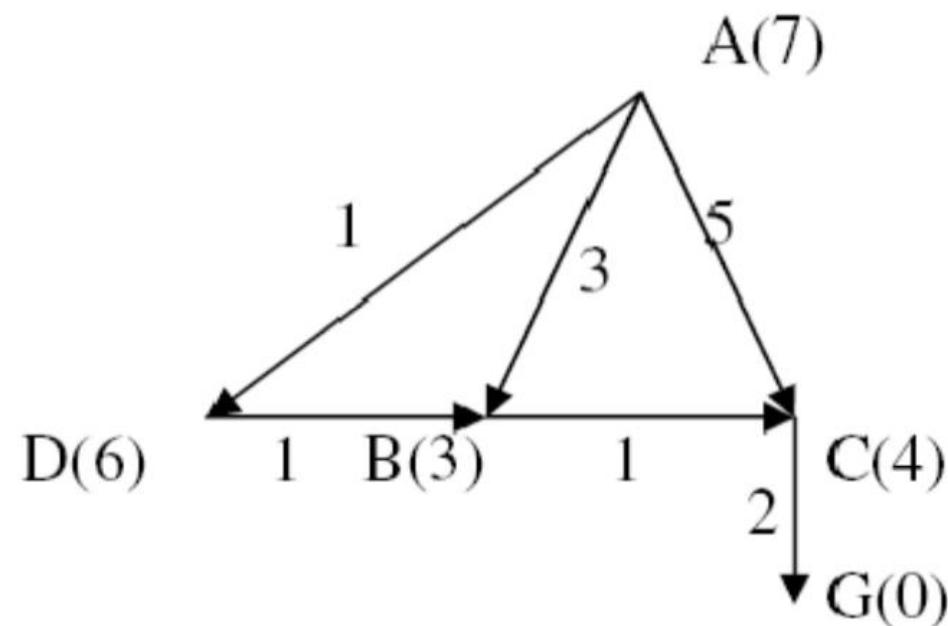
**If**  $g(m) < g(m')$  **then**  $f(m)=g(m)+h(m)$ ; cha(m)=n;

                Đưa m vào open; loại m' khỏi closed;

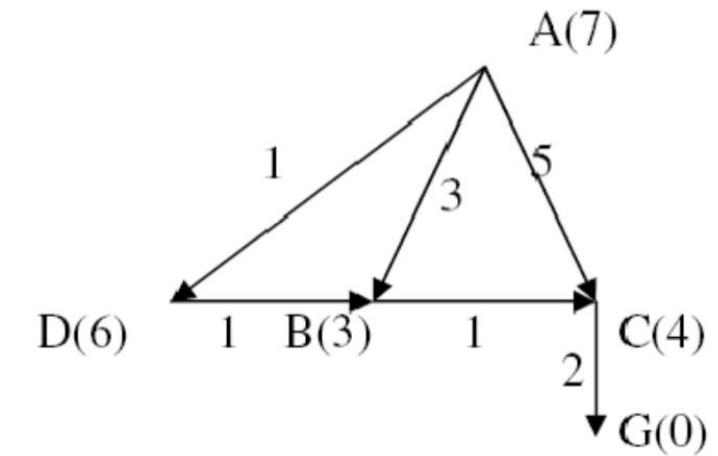
Sắp xếp open để t.thái tốt nhất nằm bên trái;

# 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:**
  - $\text{Open} = \{ A(0,7,7,-) \}$  ;  $\text{close} = \{ \}$
- **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) ( $g(m) = g(n) + c[m,n]$ )

- 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

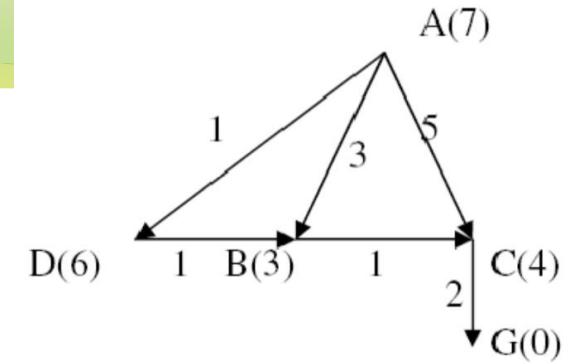
- $g(C) = g(A) + c[A,C] = 0 + 5 = 5$  (do đề bài cung cấp) ( $g(m) = g(n) + c[m,n]$ )

- 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)

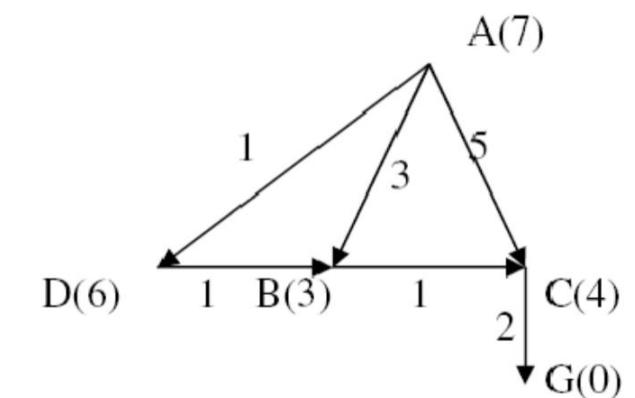
- Sắp xếp các phần tử trong open để trạng thái tốt nhất bên trái

Open{**B(3,3,6,A)**, D(1,6,7,A), C(5,4,9,A)} ; close={A(0,7,7,-)}



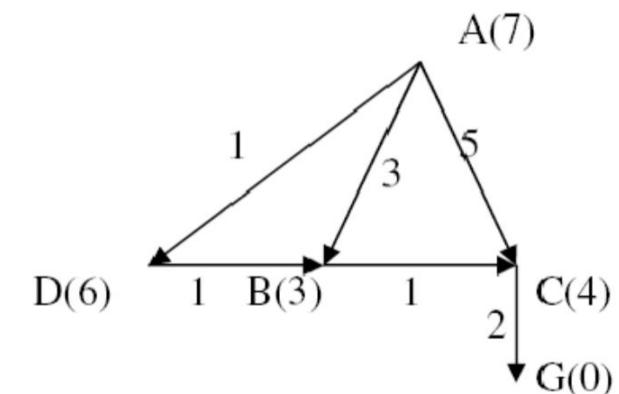
### Bước 3:

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



## Bước 4:

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



## Bước 5:

- Quay lại đầu vòng lặp while
- Lấy phần tử B ra khỏi Open đưa vào Close

Open{C(4,4,8,B) }; close={A(0,7,7,-), D(1,6,7,A), **B(2,3,5,D)** }

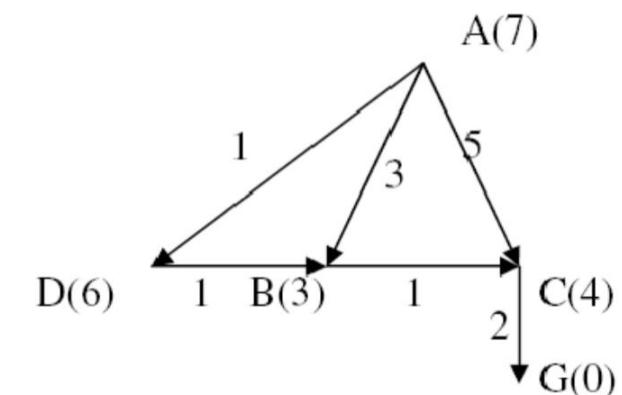
- Các con của B: C

### Xét C

- $g(C) = g(B) + c[B,C] = 1 + 2 = 3$  ( $g(m) = g(n) + c[m,n]$ )
- 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)

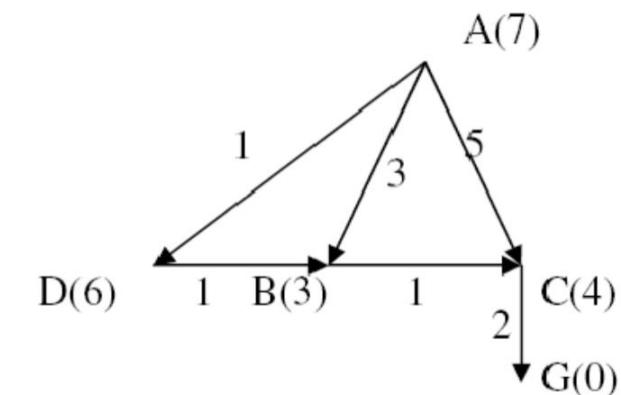
- Sắp xếp các phần tử trong open để trạng thái tốt nhất bên trái

Open{C(3,4,7,B) } ; 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  
 $\text{Open}(); \quad \text{close} = \{\text{A}(0,7,7,-), \text{D}(1,6,7,\text{A}), \text{B}(2,3,5,\text{D}), \text{C}(3,4,7,\text{B})\}$
- 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  
 $\text{Open}\{G(5,0,5,C)\} ; \text{close} = \{\text{A}(0,7,7,-), \text{D}(1,6,7,\text{A}), \text{B}(2,3,5,\text{D})\}$



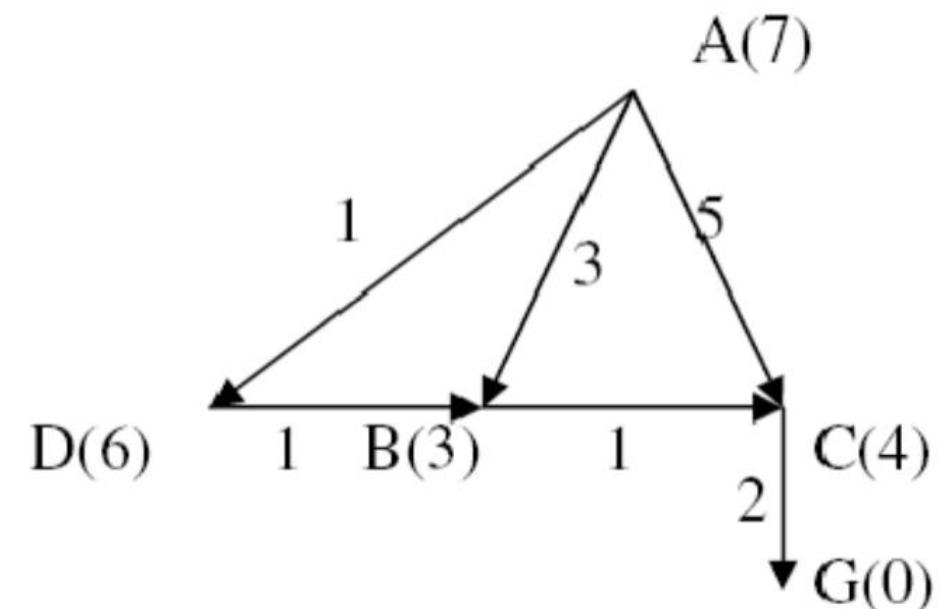
## Bước 6:

- Quay lại đầu vòng lặp while
- Lấy phần tử **G** ra khỏi Open đưa vào Close  
Open{}; 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 => giải thuật dừng lại

Ta có đường đi

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)**}

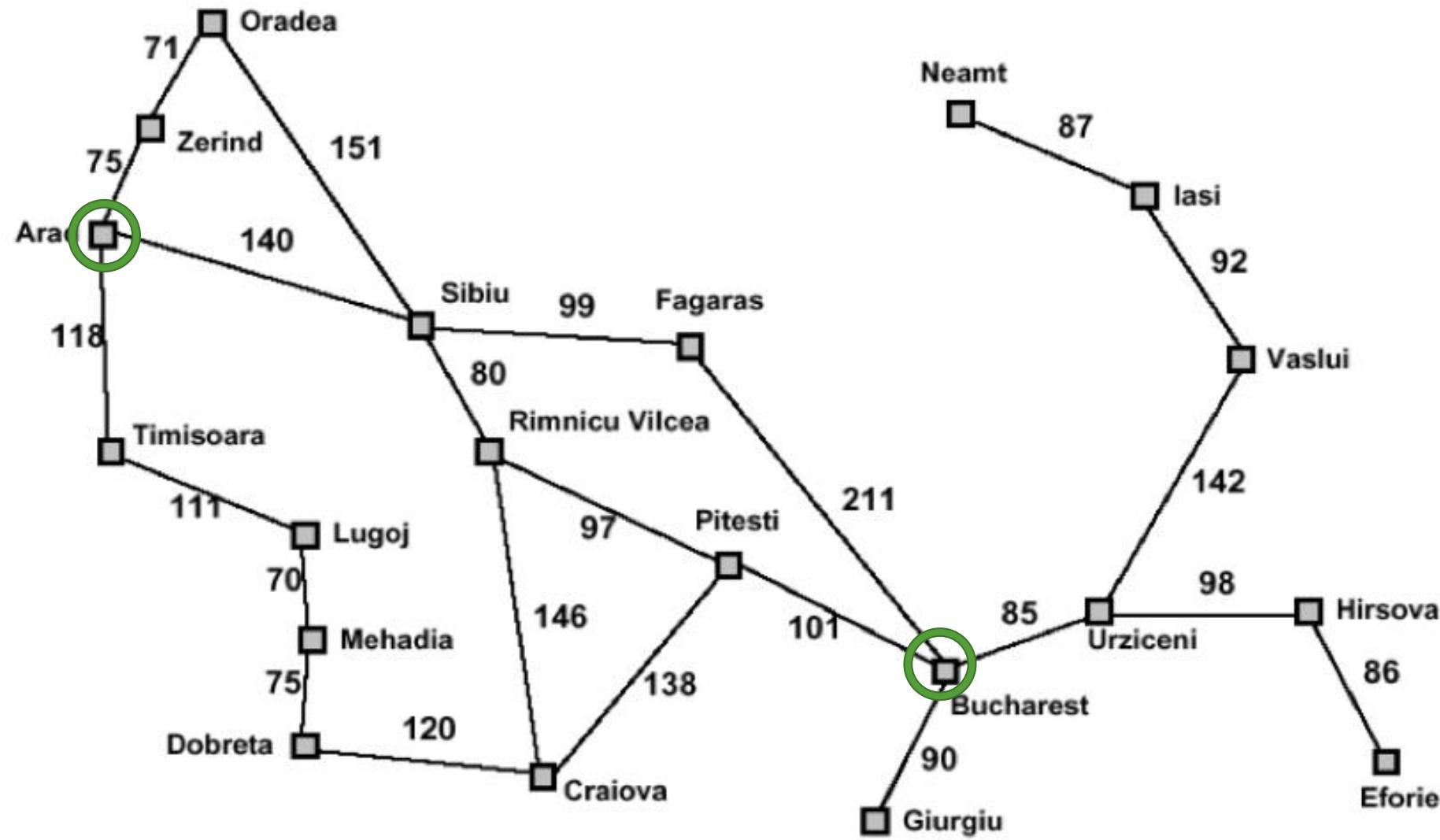
A => D => B =>C =>G



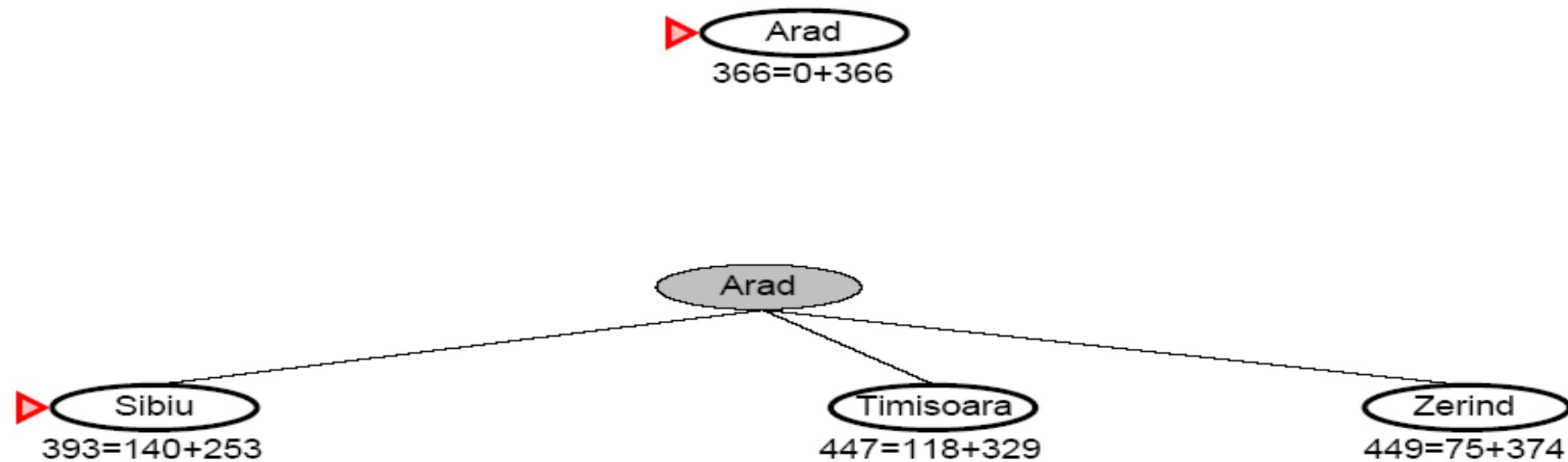
# Giải thuật A\* - Ví dụ

- 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)

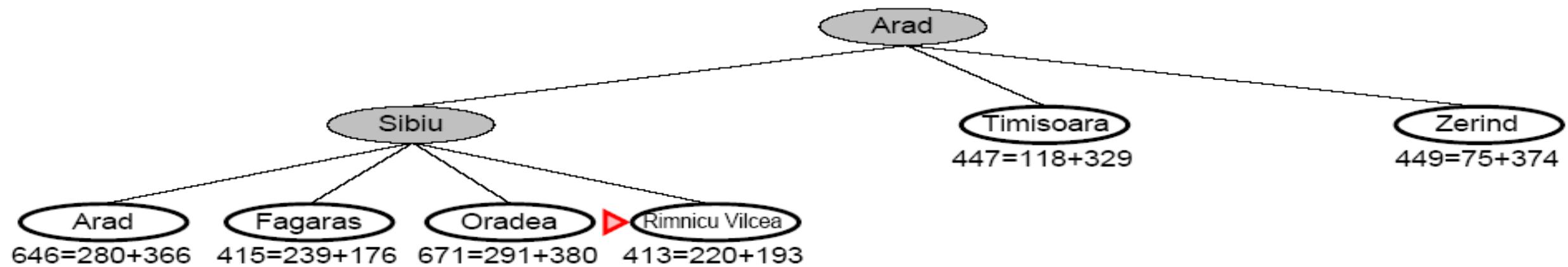
# Giải thuật A\* - Ví dụ



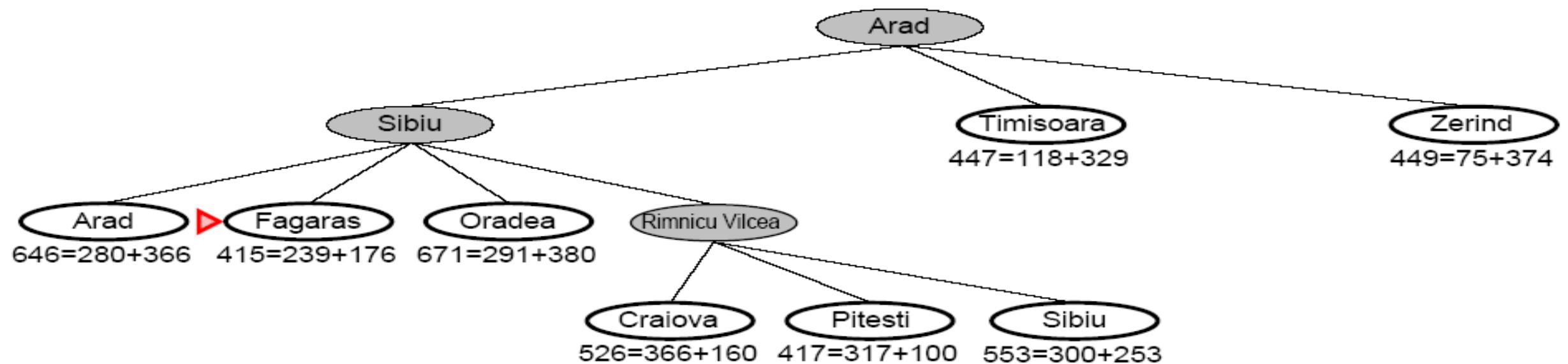
# Giải thuật A\* - Ví dụ



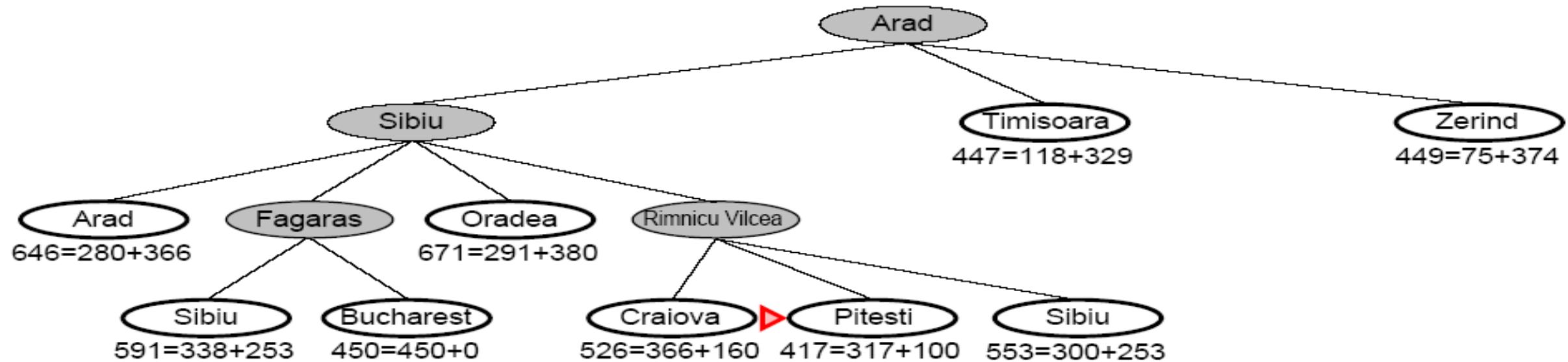
# Giải thuật A\* - Ví dụ



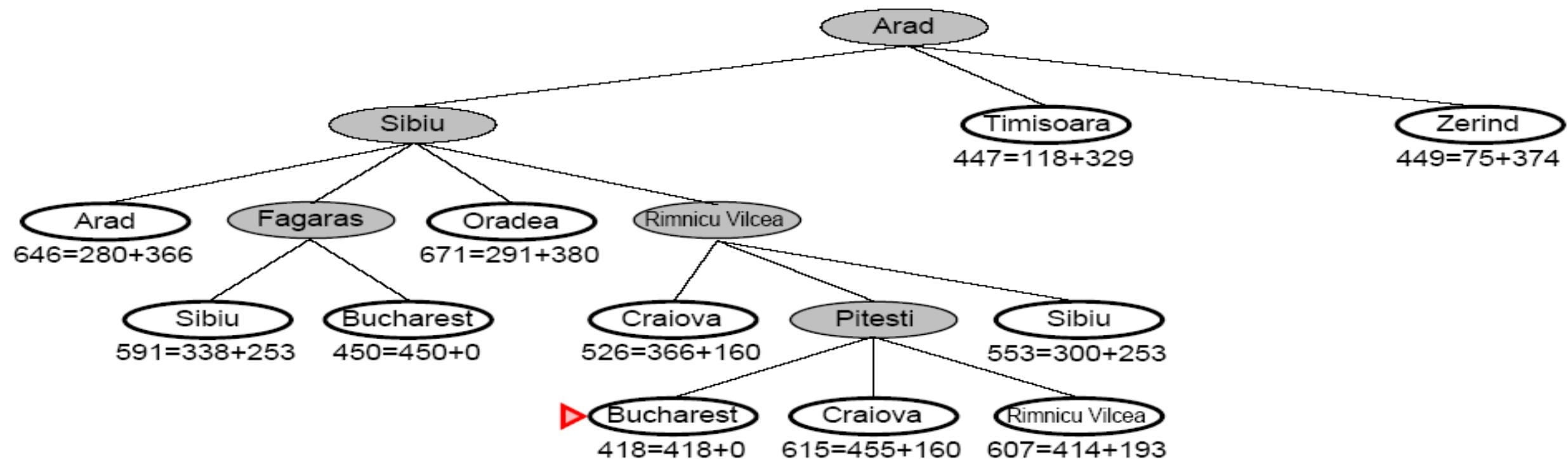
# Giải thuật A\* - Ví dụ



# Giải thuật A\* - Ví dụ



# Giải thuật A\* - Ví dụ



- 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

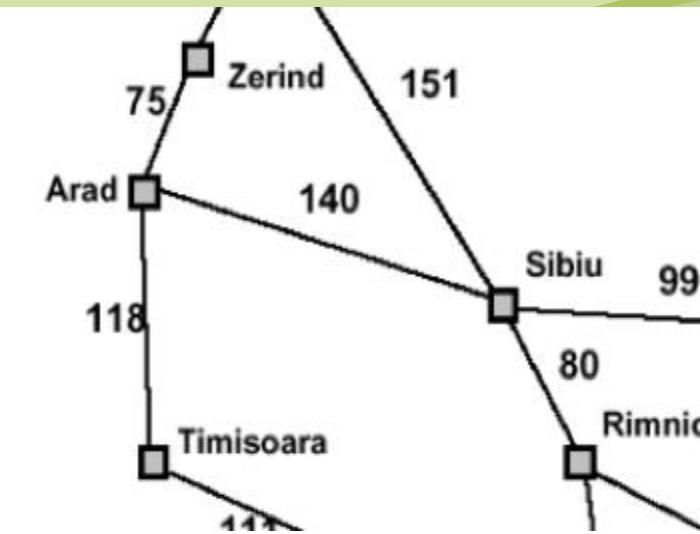
- $g(\text{Timisoara}) = g(\text{Arad}) + c[\text{Arad}, \text{Timisoara}] = 0 + 118 = 118$  (do đề bài  
cung cấp) ( $g(m) = g(n) + c[m,n]$ )

- Timisoara không thuộc Open; Close

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

- 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)

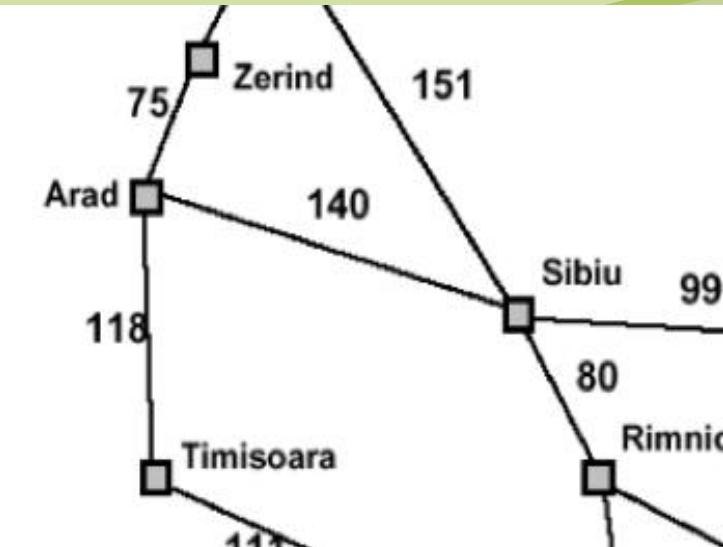
(  $g(m) = g(n) + c[m,n]$  )

- **Sibu 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:**

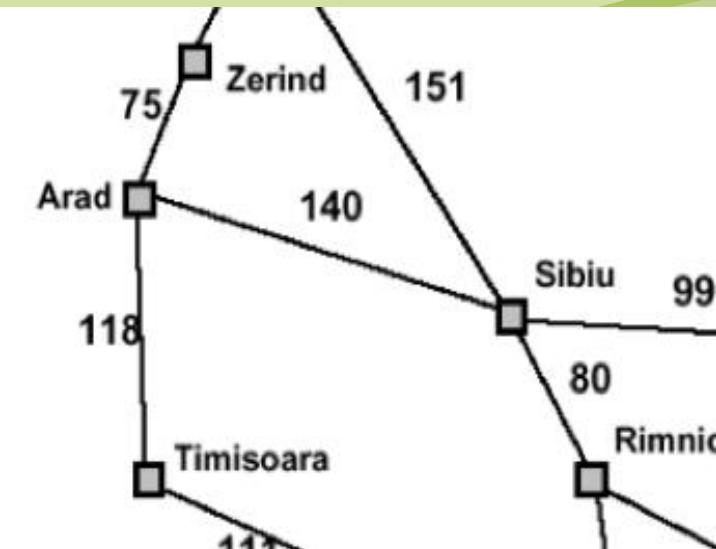
- .....
- Xét Zerind
  - $g(\text{Zerind}) = g(\text{Arad}) + c[\text{Arad}, \text{Zerind}]$   
 $= 0 + 75 = 75$  (do đề bài cung cấp)  
 $( g(m) = g(n) + c[m,n] )$

- 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: Zerind(75,374,449,Arad)**

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

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



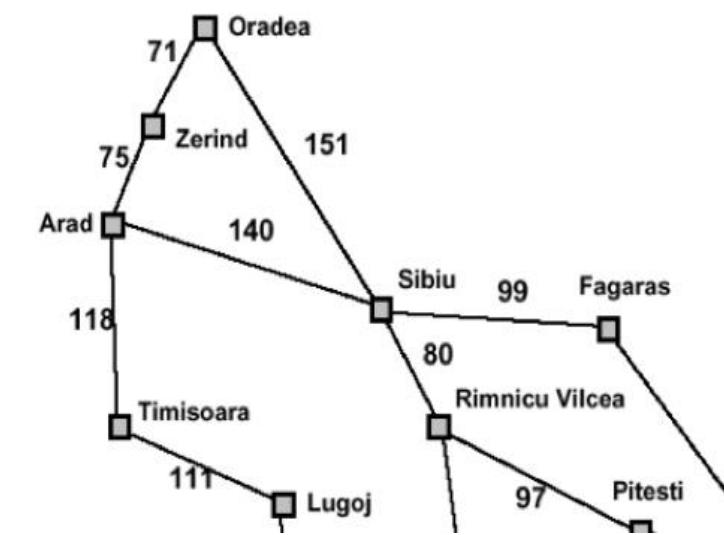
### Bước 3

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

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

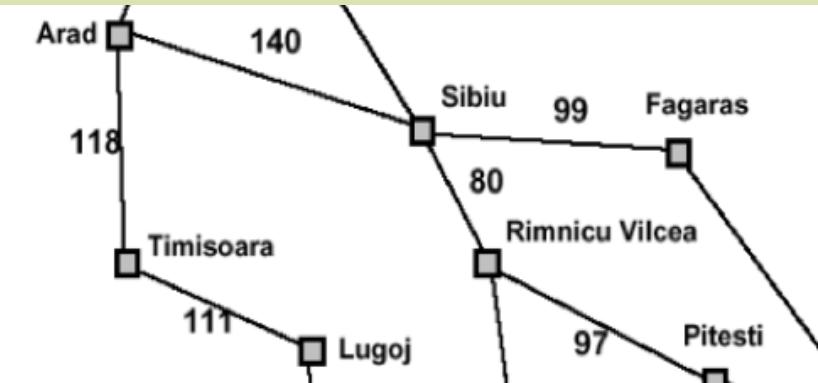
Closed = { Arad (0,366,366,-), **Sibiu (140,253,393,Arad)** }

- 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)



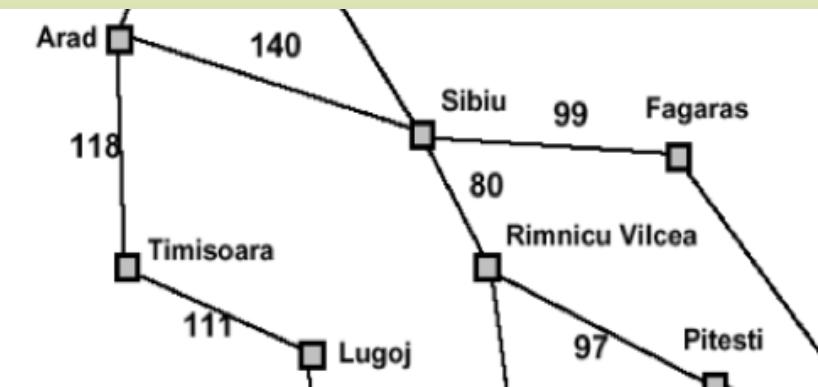
### Bước 3

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



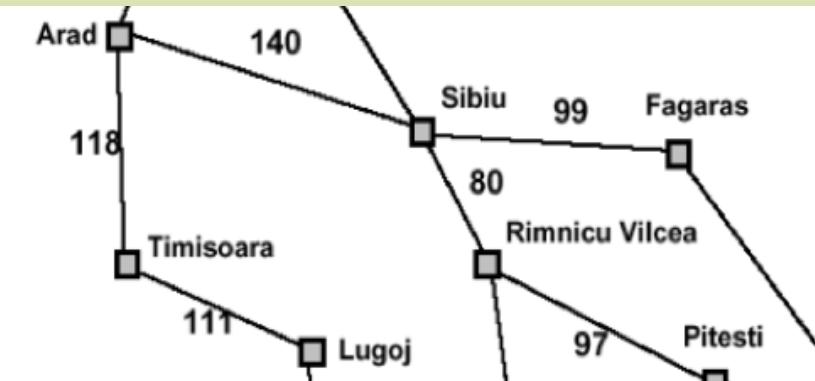
- $g(\text{Oradea}) = g(\text{Sibiu}) + c[\text{Sibiu}, \text{Oradea}] = 140 + 151 = 291$  ( $g(m) = g(n) + c[m,n]$ )
- Oradea không thuộc Open; Close
  - Tính giá trị  $f(\text{Oradea}) = g(\text{Oradea}) + h(\text{Oradea}) = 291 + 380 = 671$
  - Cập nhật cha của Oradea : Sibiu
  - Đưa Oradea vào open: Oradea(291,380,671,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), Oradea(291,381,671,Sibiu)};

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

### Bước 3

- .....



- Các con của Sibiu : Rimnicu Vilces, 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
 

Open{Rimnicu(220,193,413,Sibiu), Fagaras(239,178,417,Sibiu),  
 Timisoara(118,329,447,Arad), Zerind(75,374,449,Arad), Oradea(291,381,671,Sibiu)};

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

## Bước 4

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

Open{Pitesti(317,98,415,Rimnicu), Fagaras(239,178,417,Sibiu),

Timisoara(118,329,447,Arad), Zerind(75,374,449,Arad), Craiova(368,160,5

Closed = { Arad (0,366,366,-), Sibiu (140,253,393,Arad), Rimnicu(220,120,120,Arad) }

- Các con của Rimnicu : Craiova, Pitesti, Sibiu

### Xét Craiova

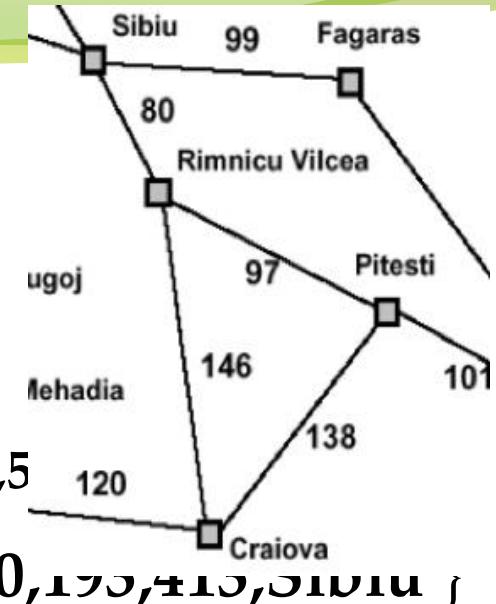
- $g(\text{Craiova}) = g(\text{Rimnicu}) + c[\text{Rimnicu}, \text{Craiova}] = 220 + 146 = 366$

### 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 Pitesti 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

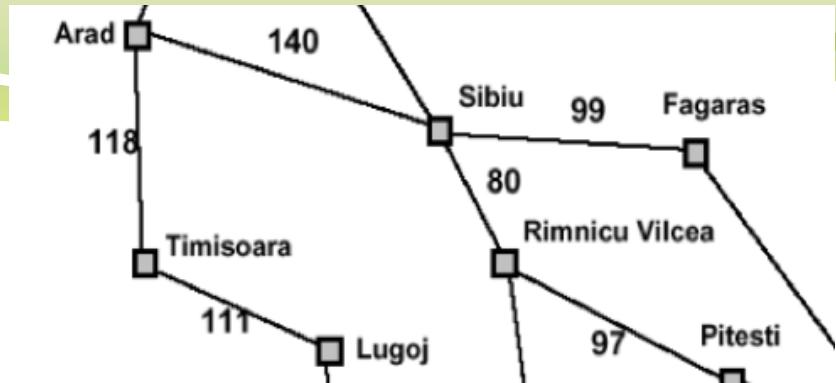
- Đưa Pitesti vào open: Pitesti (317,98,415, Rimnicu )

- 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 )};

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

- $g(Sibiu) = g(Rimnicu) + c[Rimnicu, Sibiu] = 220 + 80 = 400$

- Sibiu thuộc Close

- Tính giá trị  $g(Sibiu) = 400 > g(Sibiu') = 140$

- $\Rightarrow$  không làm gì cả (ko đưa vào open hay closed)

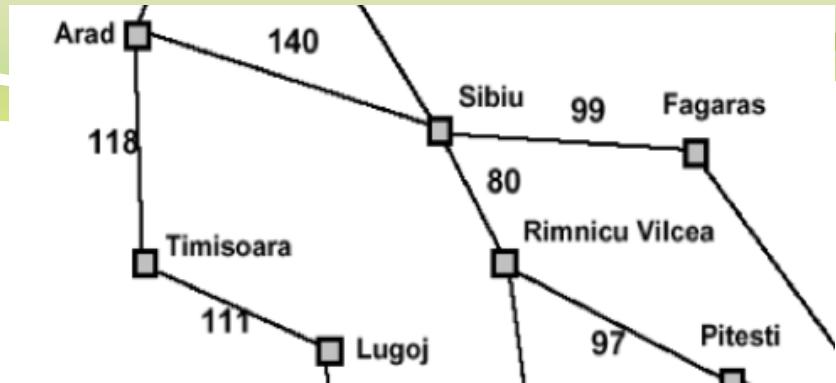
- 

- 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(368,160,528,Rimnicu)};

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



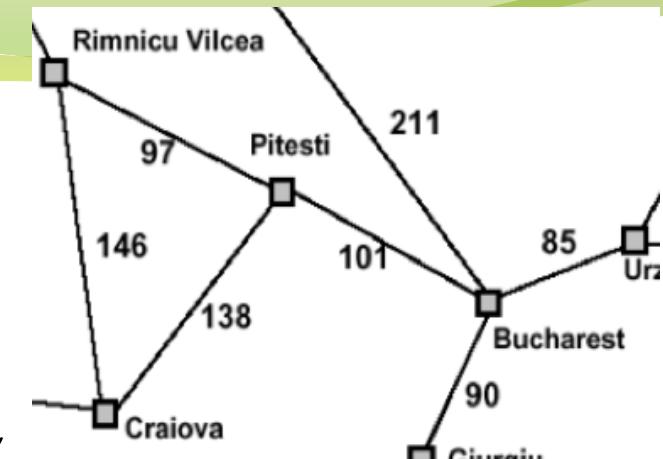
- **Bước 5**

- 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(368,160,528, Rimnicu )**};

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 Craiova**
  - $g(\text{Craiova}) = g(\text{Pitesti}) + c[\text{Pitesti}, \text{Craiova}] = 317 + 138 = 455$
  - **Craiova thuộc Open;**
  - $g(\text{Craiova})=455 > g(\text{Craiova}')= 368 \Rightarrow$  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(118,329,411,118,118,Arad), Zerind(75,374,449,Arad), Craiova(368,160,528, Rimnicu )};

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

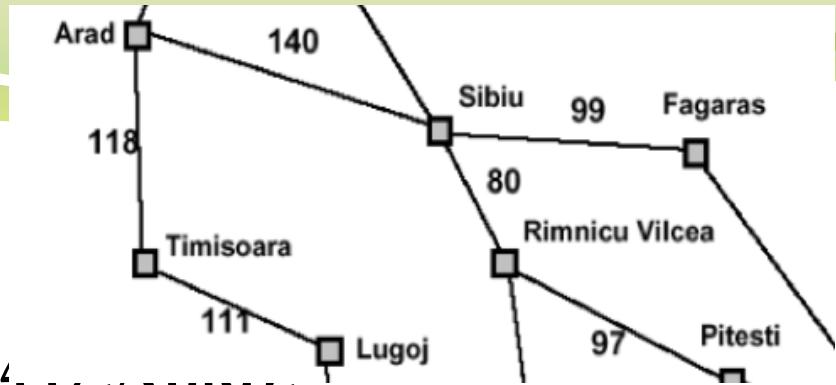
- $g(\text{Bucharest}) = g(\text{Pitesti}) + c[\text{Pitesti}, \text{Bucharest}] = 317 + 101 = 418$

- Bucharest **không thuộc Open; Closed**

- $$\begin{aligned} \text{Tính giá trị } f(\text{Bucharest}) &= g(\text{Bucharest}) + h(\text{Bucharest}) \\ &= 418 + 0 = 418 \end{aligned}$$

- Cập nhật cha của Bucharest : Pitesti**

- Đưa Bucharest vào open:** Bucharest (418,0,418,Pitesti)



## Bước 5

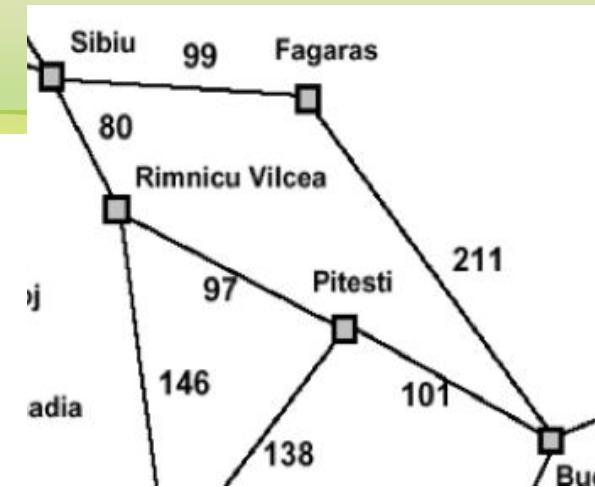
- ...

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





## Bước 6

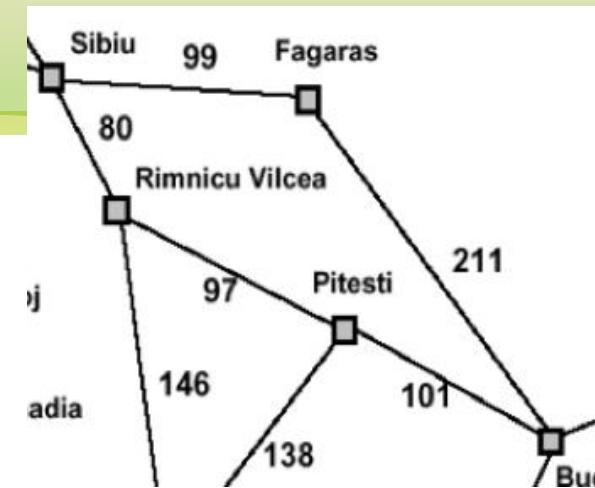
- Quay lại đầu vòng lặp while
- Lấy phần tử **Fagaras** ra khỏi Open đưa vào Closed

Open{ Bucharest (418,0,418,Pitesti), Timisoara(118,329,447,Arad) , Zerind(75,374,449,Arad), Craiova(368,160,528, Rimnicu )};

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 )};

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

# Sử dụng tìm kiếm $A^*$ 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.

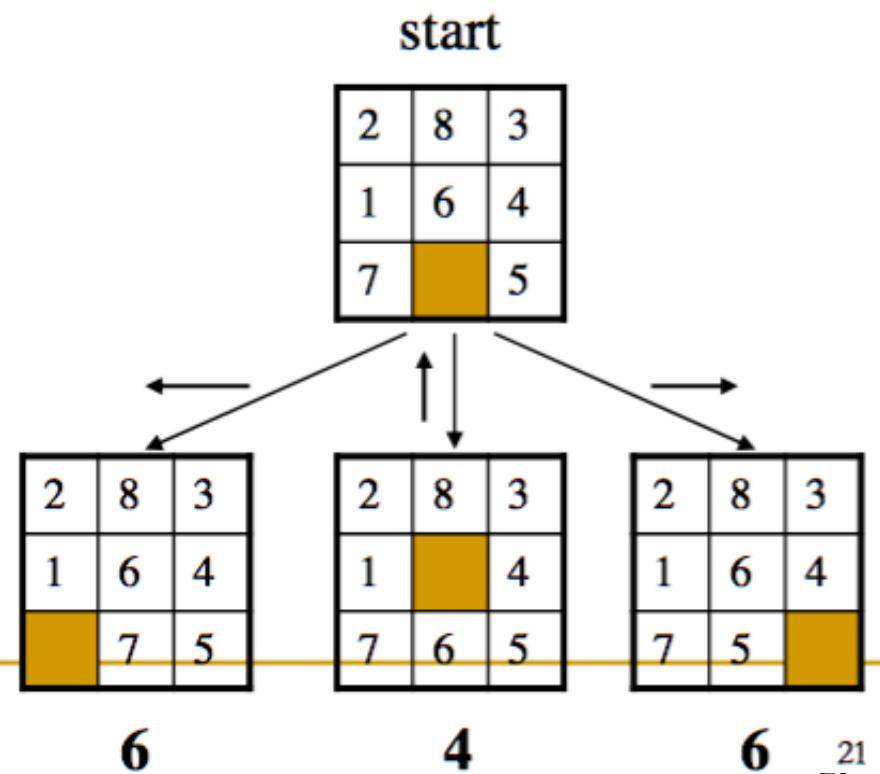
1	2	3
8		4
7	6	5

goal

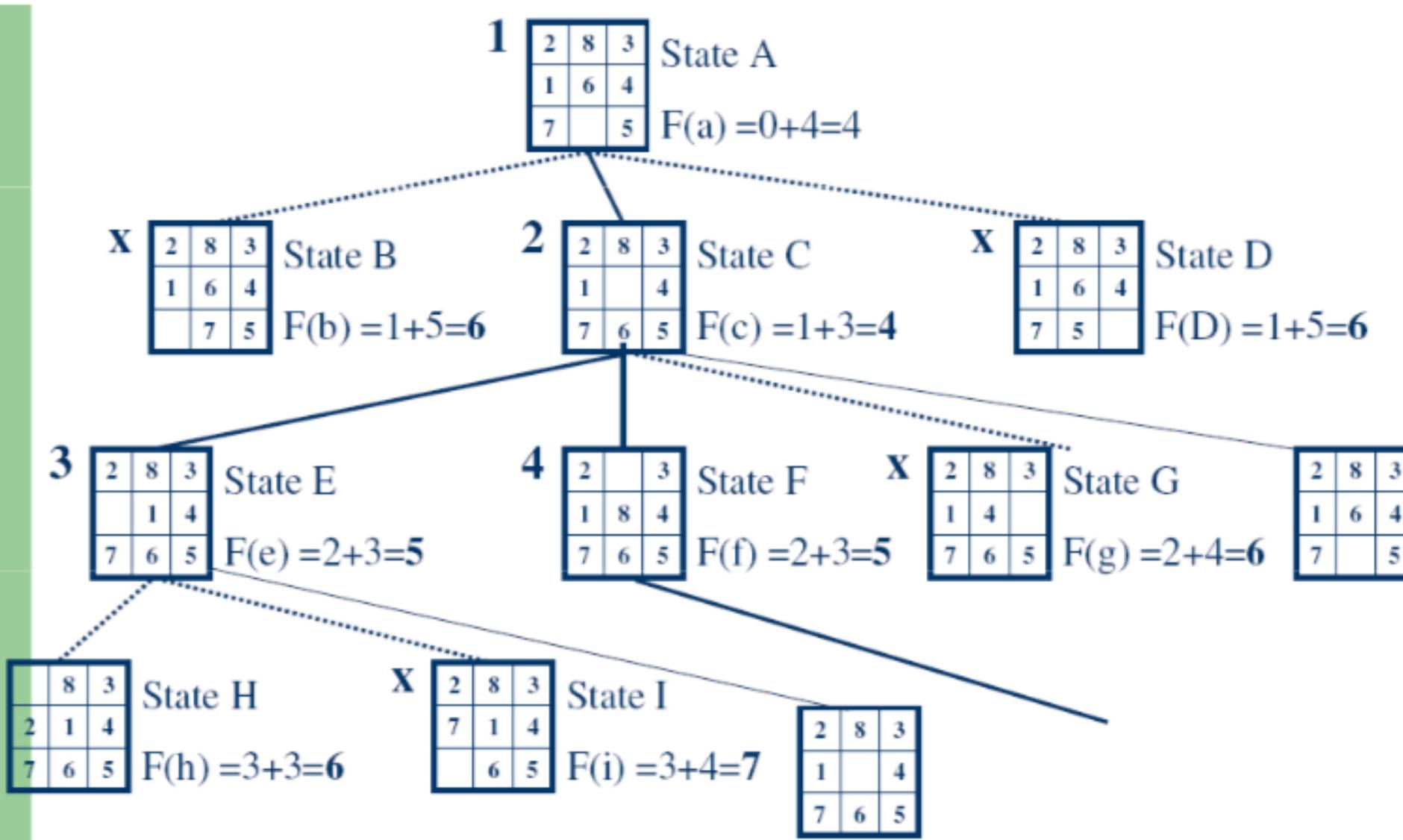
$$g(n) = 0$$

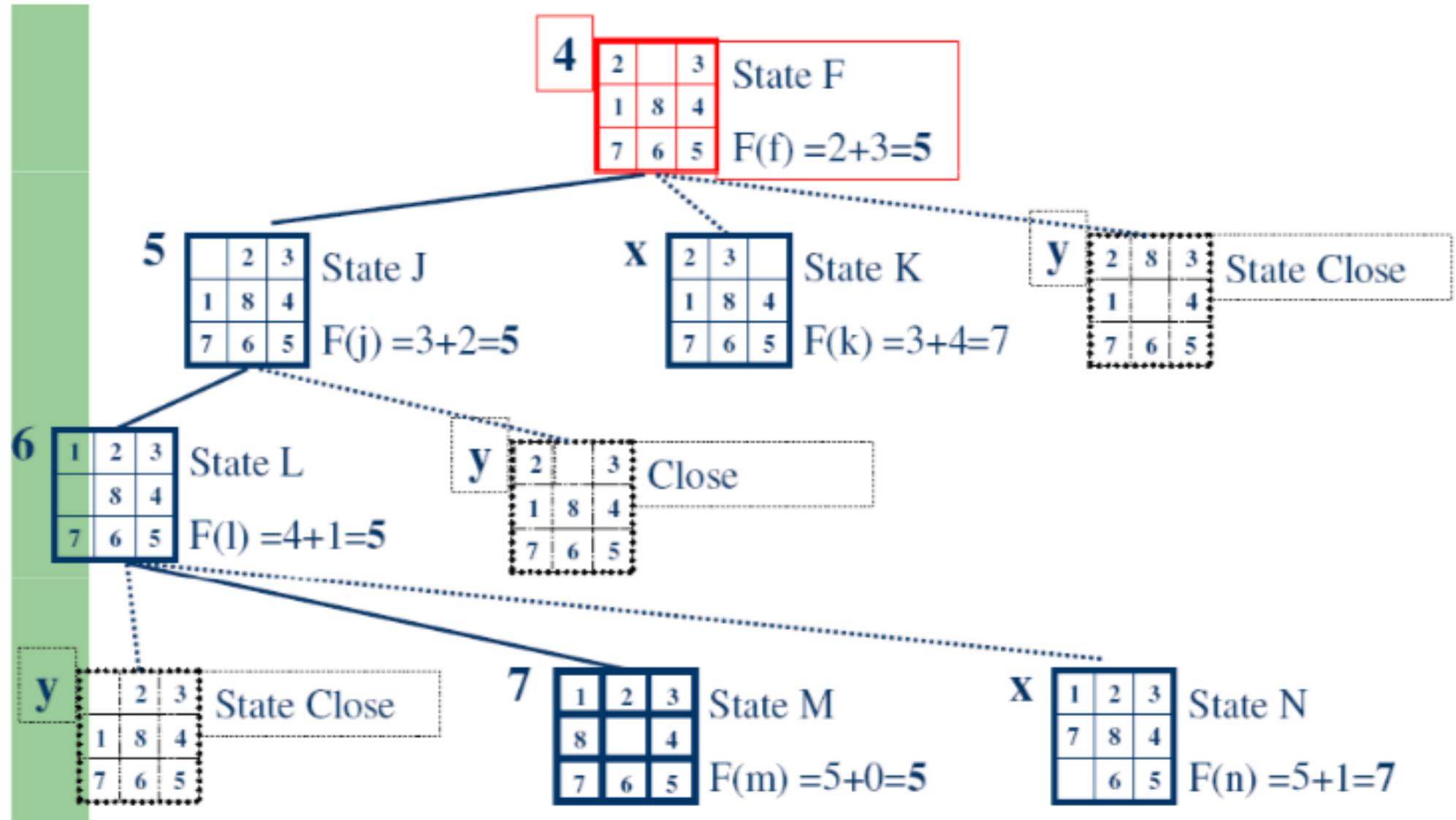
$h(n)$ : số lượng các vị trí còn sai;  $g(n) = 1$

$$f(n) =$$



# Ví dụ





# NỘI DUNG

1. Giới thiệu Heuristic
2. Các kỹ thuật tìm kiếm Heuristic
  - Tìm kiếm tốt nhất đầu tiên (Best-first-search)
    - Tìm kiếm háu ăn (Greedy best-first-search)
    - Giải thuật A\*
  - Leo đồi (Hill climbing)
  - Thỏa mãn ràng buộc (Constraint satisfaction)

# Leo đồi (Hill climbing)

- Ý tưởng: Tìm kiếm trạng thái đích tức là 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 tiếp cận trạng thái tốt hơn so với mọi trạng thái con của nó hoặc trạng thái đích
  - Sử dụng hàm đánh giá để đo tính tốt hơn của một trạng thái so với trạng thái khác

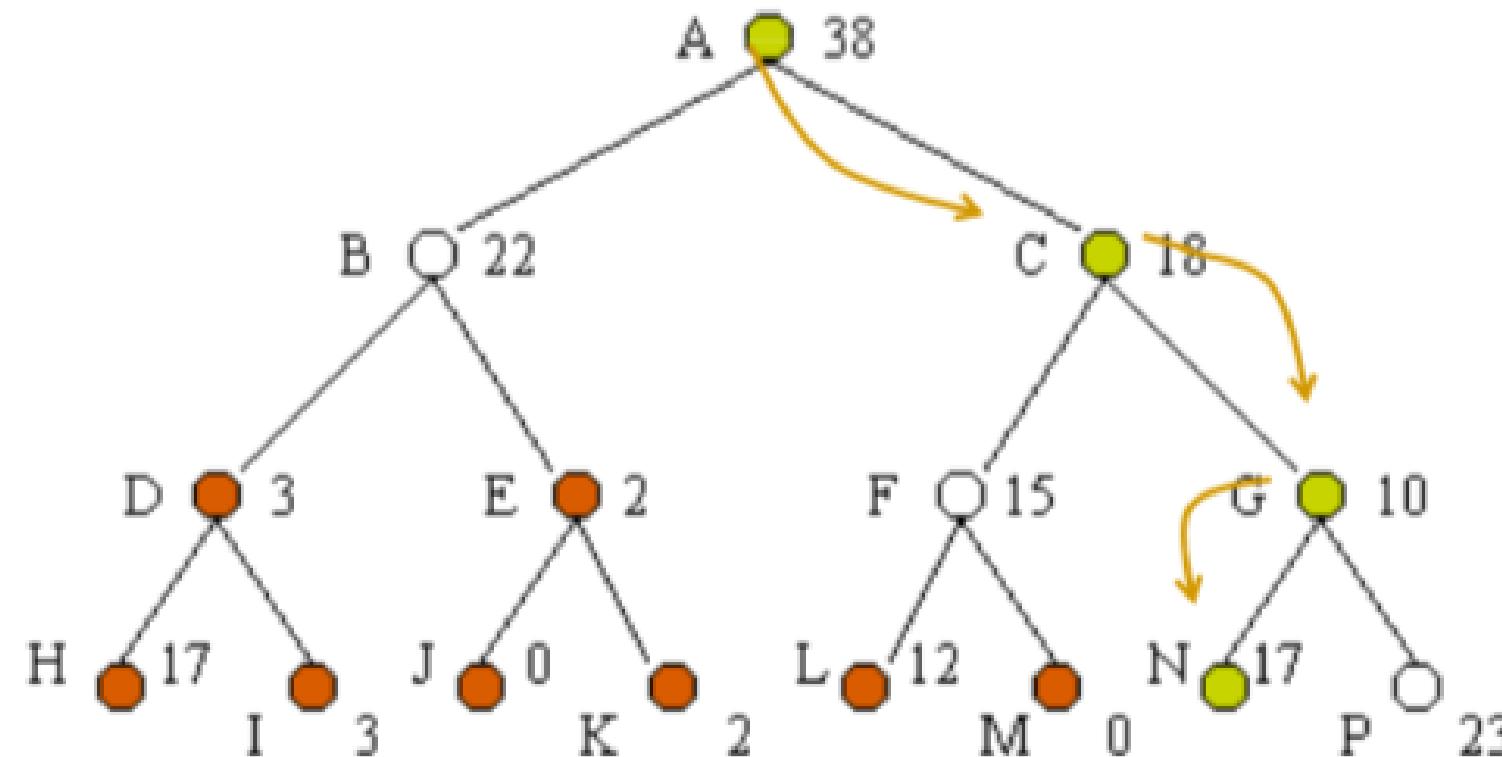
# Leo đồi (Hill climbing)

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

# Giải thuật leo đồi

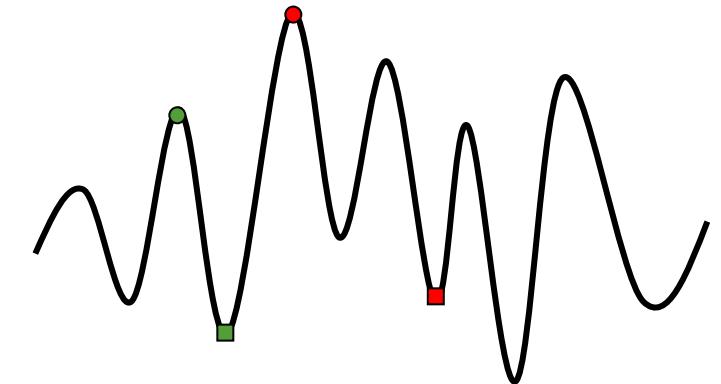
```
PState hillClimbing(PState init_state) {  
    PNode current = new Node ;  
    current->state = init_state;  
    current->h = estimated_cost(current->state);  
    while (true) {  
        PNode neighbor = successor có giá trị cao nhất  
                               của current  
        if (neighbor.h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

# Tìm kiếm leo đồi (tt)

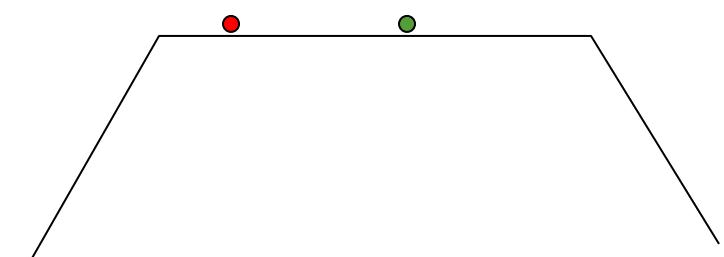


# Leo đồi (Hill climbing)

- 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 »



Cực trị địa phương



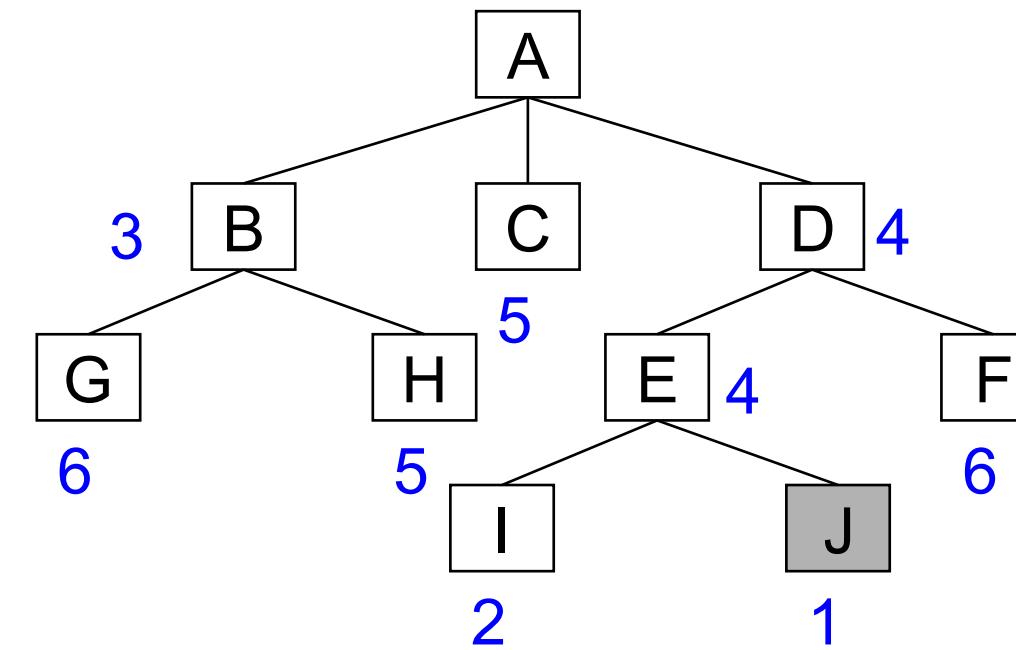
Cao nguyên

# Thuật toán 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 phép toán* để nhận được một trạng thái sau đó mới kiểm thử. Phù hợp để xử lý tình huống « Ridge »

# Bài tập

- Cho biết sự khác biệt giữa Tìm kiếm leo đồi và Tìm kiếm tốt nhất đầu tiên của cây tìm kiếm sau? Biết nút cần tìm là J (hàm heuristic nhỏ nhất là tốt nhất)



# NỘI DUNG

1. Giới thiệu Heuristic
2. Các kỹ thuật tìm kiếm Heuristic
  - Tìm kiếm tốt nhất đầu tiên (Best-first-search)
    - Tìm kiếm háu ăn (Greedy best-first-search)
    - Giải thuật A\*
  - Leo đồi (Hill climbing)
  - Thỏa mãn ràng buộc (Constraint satisfaction)

# Ràng buộc (constraint)

- Một ràng buộc (constraint) là một quan hệ trên một tập các biến
  - Mỗi biến có (gắn với) một tập các giá trị có thể nhận – gọi là miền giá trị (domain)
  - Trong môn học này, chúng ta chỉ xét các miền hữu hạn các giá trị rời rạc
- Một ràng buộc có thể được biểu diễn bằng
  - Một biểu thức (toán học / logic)
  - Một bảng liệt kê các phép gán giá trị phù hợp cho các biến
- Ví dụ về ràng buộc
  - Tổng các góc trong một tam giác là  $180^\circ$
  - Độ dài của từ W là 10 ký tự
  - X nhỏ hơn Y

# Vấn đề 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

# Vấn đề thỏa mãn ràng buộc (Constraint satisfaction problem)

- 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 tượng phép toán)
- Bài toán CSP được giải quyết khi **tất cả các biến đều được gán trị hợp lệ**

# Các kiểu ràng buộc

- **Ràng buộc đơn** (unary constraint) chỉ liên quan đến 1 biến
  - Ví dụ: SA  $\neq$  green
- **Ràng buộc nhị phân** (binary constraint) liên quan đến 2 biến
  - Ví dụ: SA  $\neq$  WA
- **Ràng buộc bậc cao** (higher-order constraint) liên quan đến nhiều hơn 2 biến
  - Ví dụ: Các ràng buộc trong bài toán mật mã số học (trình bày ở slide tiếp theo)

# Vấn đề thỏa mãn ràng buộc (Constraint satisfaction problem)

- 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

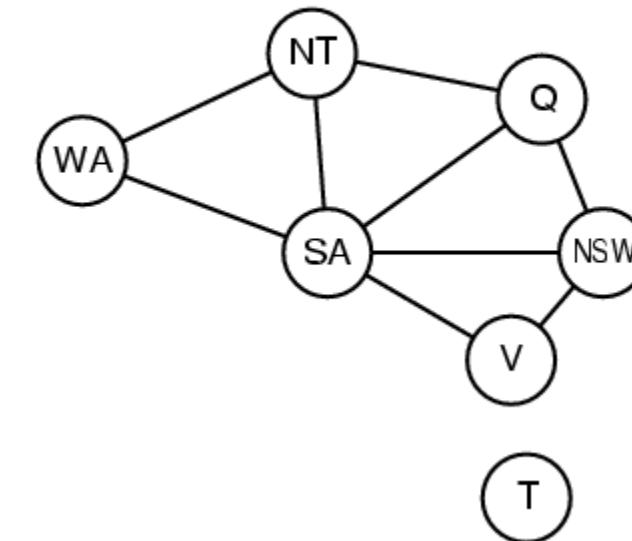
# Ví dụ: Bài toán tô màu bản đồ

- 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
- 



# Các bài toán CSP trong 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

# Các phương pháp giải bài toán CSP

- Tìm kiếm bằng kiểm thử (Generate and Test)
- 
- Tìm kiếm quay lui (Backtracking)
- Lan truyền các ràng buộc (Constraint propagation)

# Tìm kiếm bằng kiểm thử (Generate and Test) (1)

- Là phương pháp giải quyết vấn đề tổng quát nhất
- Ý tưởng:
  - Sinh ra một khả năng (candidate) của lời giải
  - Kiểm tra xem khả năng này có thực sự là một lời giải
- Áp dụng phương pháp kiểm thử đối với bài toán CSP
  - Bước 1. Gán các giá trị cho tất cả các biến
  - Bước 2. Kiểm tra xem tất cả các ràng buộc được thỏa mãn hay không
  - Lặp lại 2 bước này cho đến khi tìm được một phép gán thỏa mãn

# Tìm kiếm bằng kiểm thử (Generate and Test) (2)

- Điểm yếu nghiêm trọng của phương pháp tìm kiếm bằng kiểm thử là việc phải **xét quá nhiều các khả năng gán** (hiển nhiên) không thỏa mãn các ràng buộc
- Ví dụ
  - Các biến X,Y,Z lấy các giá trị {1,2}
  - Các ràng buộc:  $X=Y$ ,  $X \neq Z$ ,  $Y>Z$
  - Các phép (khả năng) gán: (1,1,1); (1,1,2); (1,2,1); (1,2,2); (2,1,1); (2,1,2); (2,2,1)

# Tìm kiếm quay lui (backtracking)

- Tìm kiếm quay lui (backtracking) là giải thuật tìm kiếm được sử dụng phổ biến nhất trong CSP
  - Dựa trên giải thuật tìm kiếm theo chiều sâu (depth-first search)
  - Mỗi lần gán, chỉ làm việc (gán giá trị) cho một biến
    - (Tìm kiếm bằng kiểm thử: mỗi lần gán xác định các giá trị cho tất cả các biến)
- Phương pháp tìm kiếm quay lui đối với bài toán CSP
  - Gán giá trị lần lượt cho các biến – Việc gán giá trị của biến này chỉ được làm sau khi đã hoàn thành việc gán giá trị của biến khác
  - Sau mỗi phép gán giá trị cho một biến nào đó, kiểm tra các ràng buộc có được thỏa mãn bởi tất cả các biến đã được gán giá trị cho đến thời điểm hiện tại – Quay lui (backtrack) nếu có lỗi (không thỏa mãn các ràng buộc)

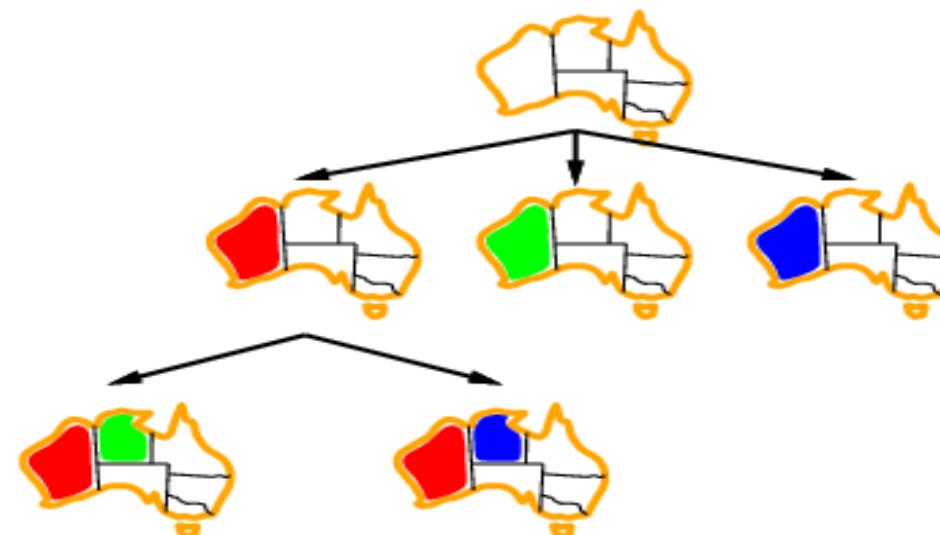
# Tìm kiếm quay lui: ví dụ



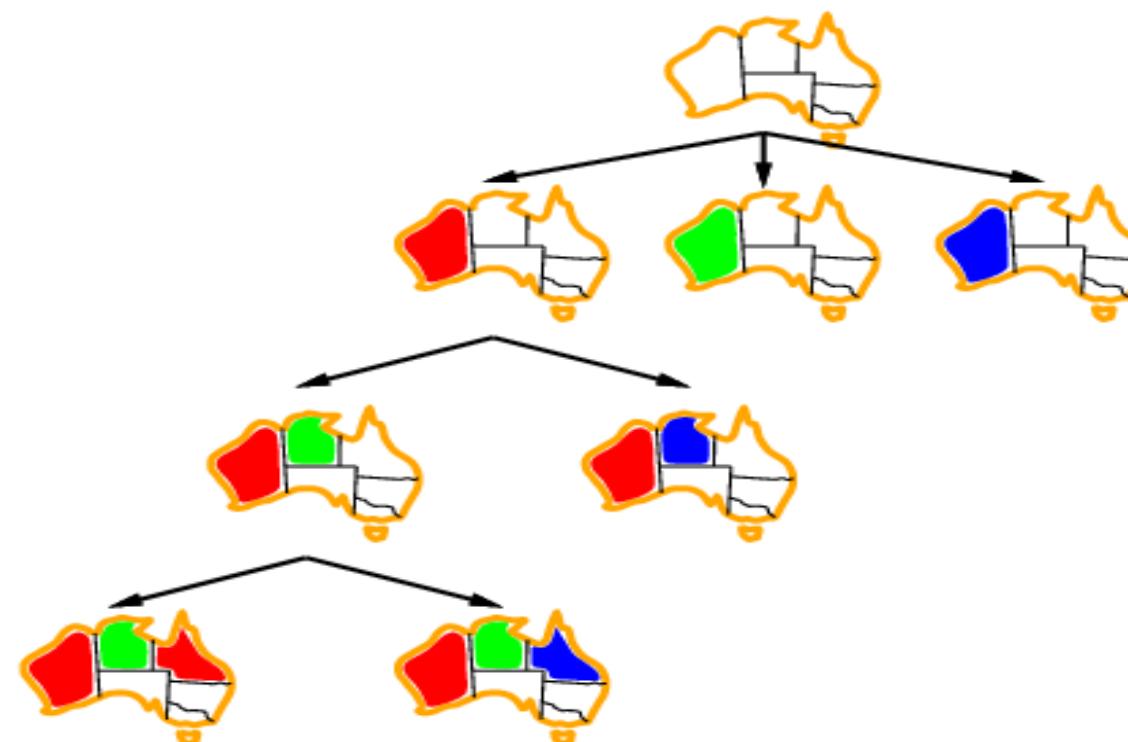
# Tìm kiếm quay lui: ví dụ



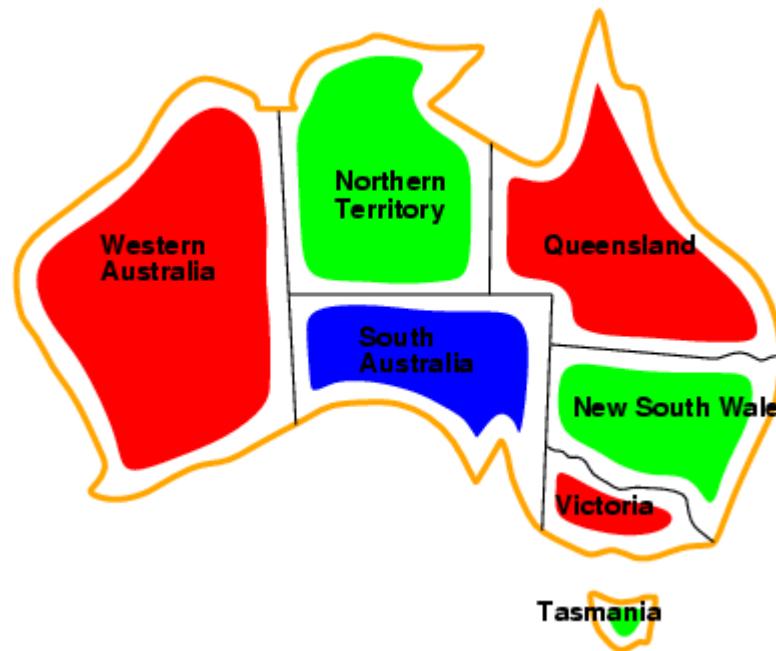
# Tìm kiếm quay lui: ví dụ



# Tìm kiếm quay lui: ví dụ



# Tìm kiếm quay lui: ví dụ



- 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 chứa 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 đố « 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:  
*Đó là phát hiện trạng thái thỏa mãn một tập các ràng buộc đã cho.*  
VD: Số học mật mã (CriptArithmetic)

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

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

- 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)

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

- Các biến:

- ✓ S, E, N, D, M, O, R, Y và

- ✓  $C_i$  ( $i = 1, 2, 3, 4$ ) (các nhớ của phép cộng)

- Miền giá trị:  $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$  đối với 8 biến S, E, N, D, M, O, R, Y, và  $\{0, 1\}$  đối với 4 biến  $C_1, C_2, C_3, C_4$ .

- Ràng buộc:

- ✓ Alldiff(S, E, N, D, M, O, R, Y)

- ✓  $D + E = Y + 10 * C_1$

- ✓  $C_1 + N + R = E + 10 * C_2$

- ✓  $C_2 + E + O = N + 10 * C_3$

- ✓  $C_3 + S + M = O + 10 * C_4$

- ✓  $C_4 = M$ .  $S \neq 0$ ;  $M \neq 0$ .

# Thỏa mãn ràng buộc (Constraint satisfaction)

+

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

M = (0, 1, ...., 9) ?

# Thỏa mãn ràng buộc (Constraint satisfaction)

- Các quy tắc để lan truyền ràng buộc sinh ra các ràng buộc sau:
  - $M = 1$  ( $S + M + C_3 \leq 19$ )
  - $S = ?$

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

$$\begin{array}{r} S \ E \ N \ D \\ \underline{+} \ 1 \ O \ R \ E \\ \hline 1 \ O \ N \ E \ Y \end{array}$$

# 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  $9$  ( $S + M + C_3 > 9, C_3 \leq 1, M=1$ )
  - $O = ?$

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

$$\begin{array}{r} S \ E \ N \ D \\ \quad \quad \quad \quad \quad \\ 1 \quad O \ R \ E \\ \hline 1 \quad O \ N \ E \ Y \end{array}$$

# 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  $9$  ( $S + M + C_3 > 9, C_3 \leq 1, M=1$ )
- $O = 0$  ( $S + M + C_3 = 10 + O$ , với  $M=1, S \leq 9, C_3 \leq 1$  nên  $O \leq 1$ )
- $S = ?$

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

$$\begin{array}{r} S \quad E \quad N \quad D \\ \quad \quad \quad \quad \\ 1 \quad 0 \quad R \quad E \\ \hline 1 \quad 0 \quad N \quad E \quad Y \end{array}$$

# 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  $O = 0$ )
- $N = ?$

$$\begin{array}{r} \text{S E N D} \\ \text{M O R E} \\ \hline \text{M O N E Y} \end{array}$$
  
$$\begin{array}{r} \text{9 E N D} \\ \text{1 0 R E} \\ \hline \text{1 0 N E Y} \end{array}$$

# 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  $9$  ( $S + M + C_3 > 9, C_3 \leq 1, M=1$ )
  - $O = 0$  ( $S + M + C_3 = 10 + O$ , 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  $N = E + 1$ )
  - $C_2 = 1$
  - $R = ?$

$$\begin{array}{r} \text{S} \quad \text{E} \quad \text{N} \quad \text{D} \\ \text{M} \quad \text{O} \quad \text{R} \quad \text{E} \\ \hline \text{M} \quad \text{O} \quad \text{N} \quad \text{E} \quad \text{Y} \end{array}$$
$$\begin{array}{r} \textcolor{red}{9} \quad \text{E} \quad \text{N} \quad \text{D} \\ \textcolor{red}{1} \quad \textcolor{red}{0} \quad \text{R} \quad \text{E} \\ \hline \textcolor{red}{1} \quad \textcolor{red}{0} \quad \text{N} \quad \text{E} \quad \text{Y} \end{array}$$

# 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  $N = E + 1$ )
  - $C_2 = 1$
  - $C_1 = 1$  và  $R = 8$  (nếu  $C_1 = 0 \Rightarrow R = 9$ )
  - D từ 7 và E từ 5 (vì  $D + E \geq 12$ )

Không có thêm ràng buộc !

$$\begin{array}{r} \text{S} \quad \text{E} \quad \text{N} \quad \text{D} \\ \text{M} \quad \text{O} \quad \text{R} \quad \text{E} \\ \hline \text{M} \quad \text{O} \quad \text{N} \quad \text{E} \quad \text{Y} \end{array}$$
$$\begin{array}{r} \text{9} \quad \text{E} \quad \text{N} \quad \text{D} \\ \text{1} \quad \text{0} \quad \text{8} \quad \text{E} \\ \hline \text{1} \quad \text{0} \quad \text{N} \quad \text{E} \quad \text{Y} \end{array}$$

# Thỏa mãn ràng buộc (Constraint satisfaction)

- Các quy tắc đê 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  $N = E + 1$ )
  - $C_2 = 1$
  - $C_1 = 1$  và  $R = 8$  (nếu  $C_1 = 0 \Rightarrow R = 9$ )
  - $D$  từ 7 và  $E$  từ 5 (vì  $D + E \geq 12$ )

Không có thêm ràng buộc !

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

# Thỏa mãn ràng buộc (Constraint satisfaction)

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

$E = 5;$

.....

- Kết quả:

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

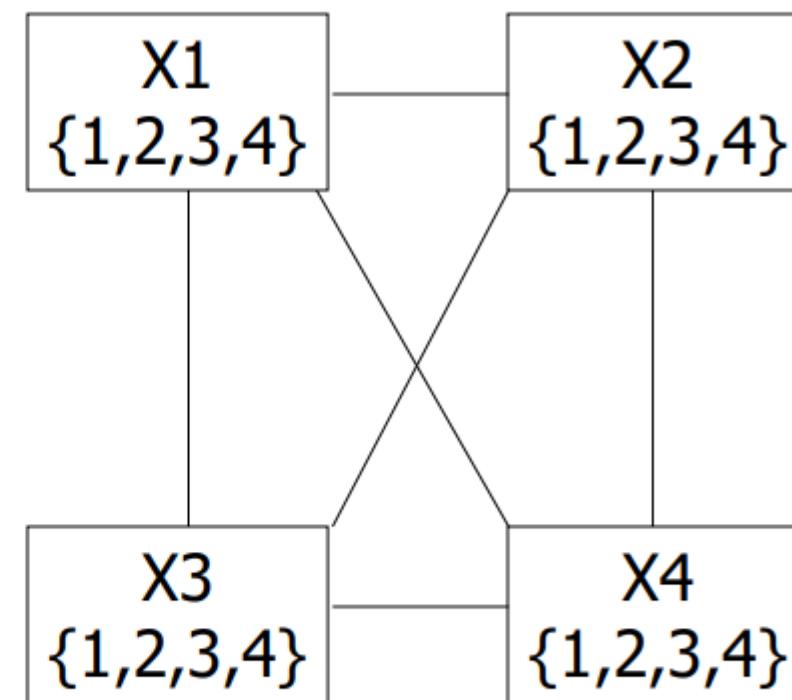
$$\begin{array}{r} 9567 \\ + 1085 \\ \hline 10652 \end{array}$$

# Hạn chế của tìm kiếm Heuristic

- Dựa vào kinh nghiệm hoặc trực giác => phỏng đoán các thông tin về bước tiếp theo sẽ được chọn dùng trong việc giải quyết một vấn đề.
- Heuristic sử dụng những thông tin hạn chế => ít khi dự đoán chính xác cách hành xử của không gian trạng thái ở những giai đoạn xa hơn.
- Chỉ đạt được giải pháp gần tối ưu hoặc hoàn toàn không tìm được bất kỳ giải pháp nào.

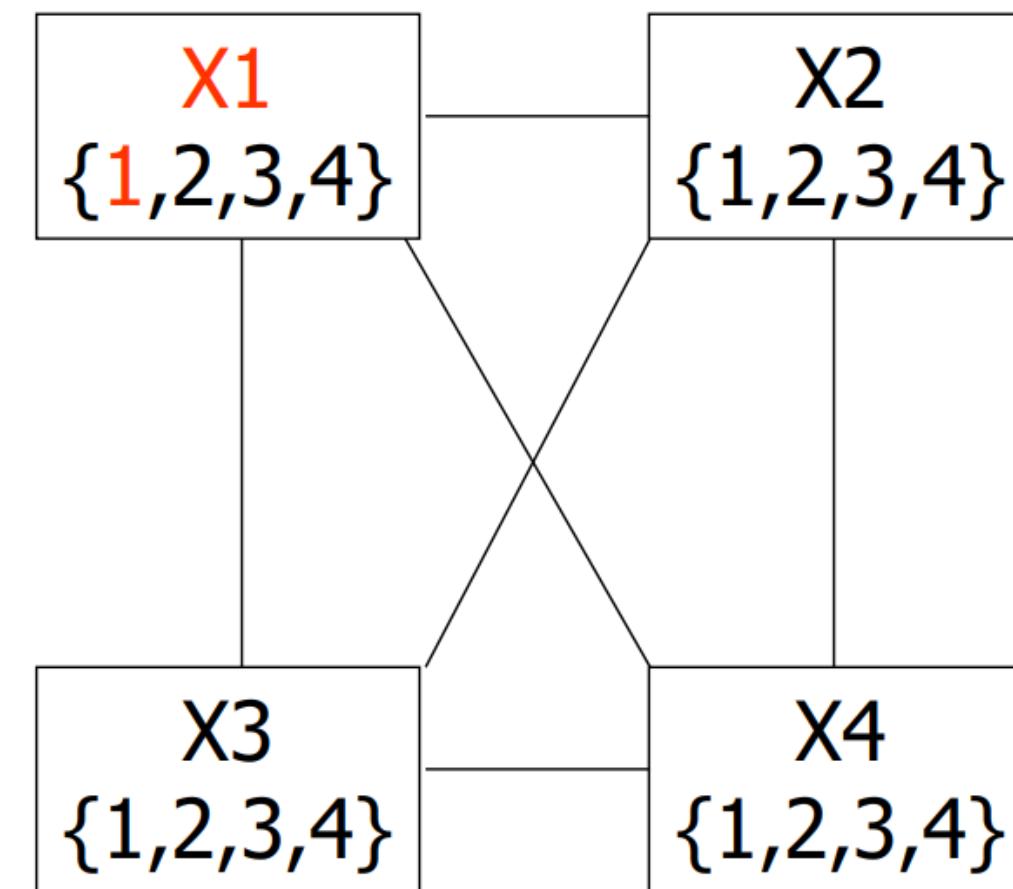
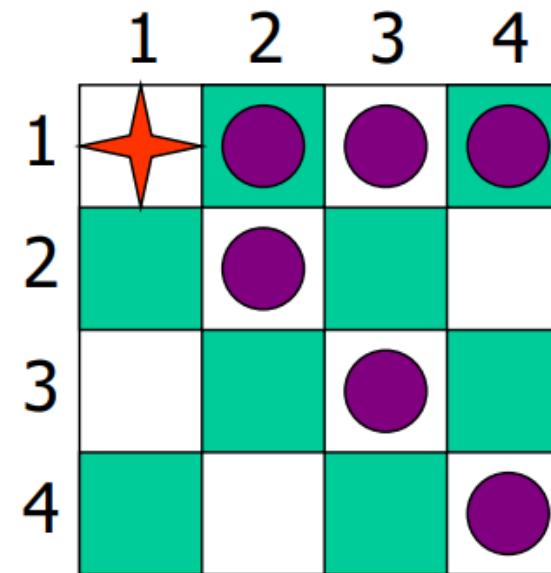
# Ví dụ tìm kiếm tiền

	$x_1$	$x_2$	$x_3$	$x_4$
1				
2				
3				
4				

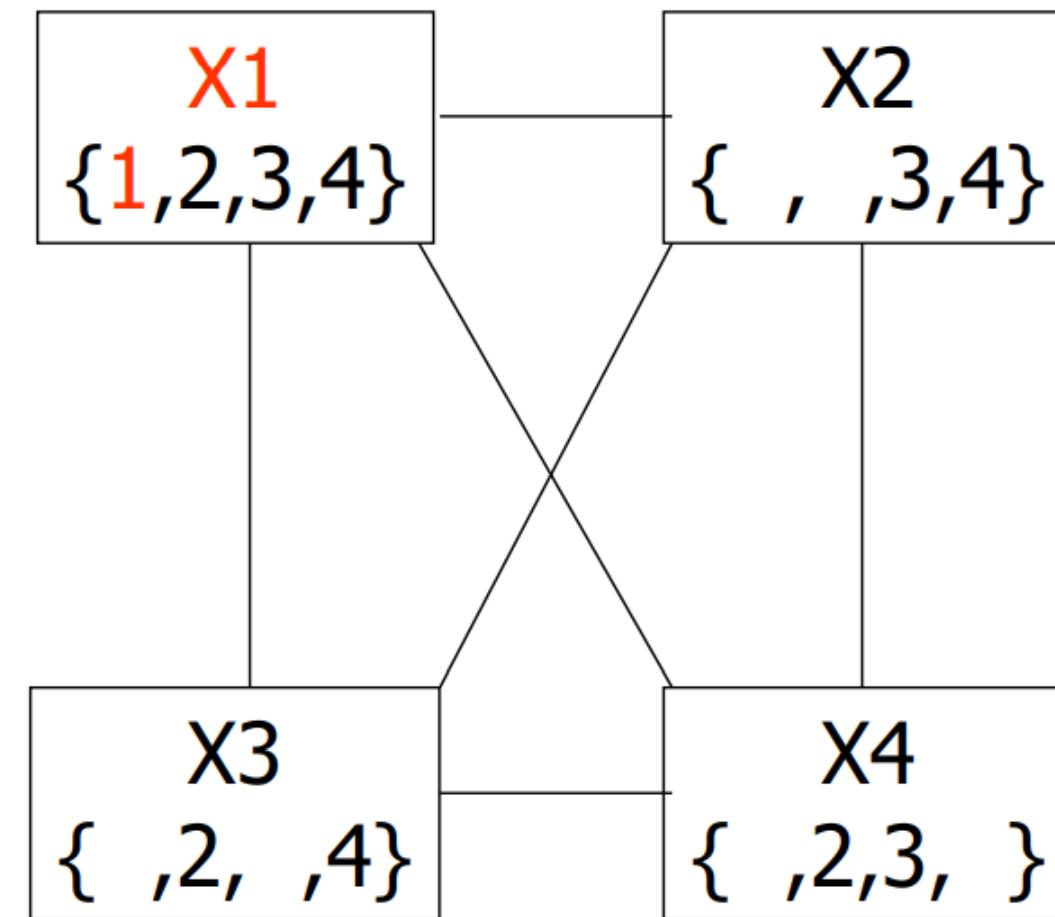
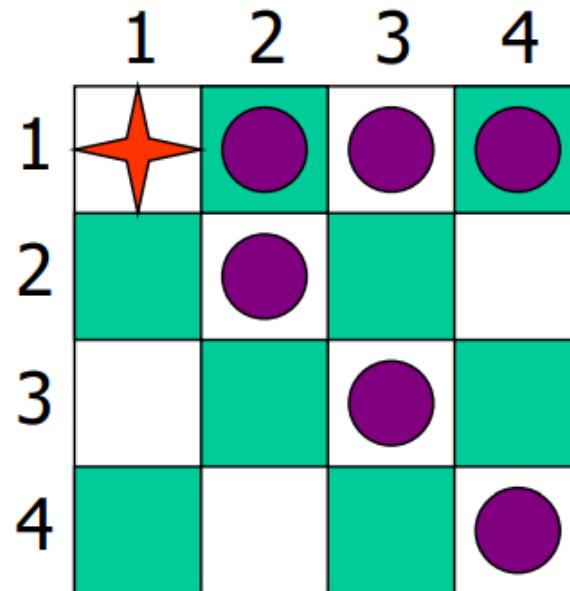


(From Bonnie Dorr, U of Md, CMSC 421)

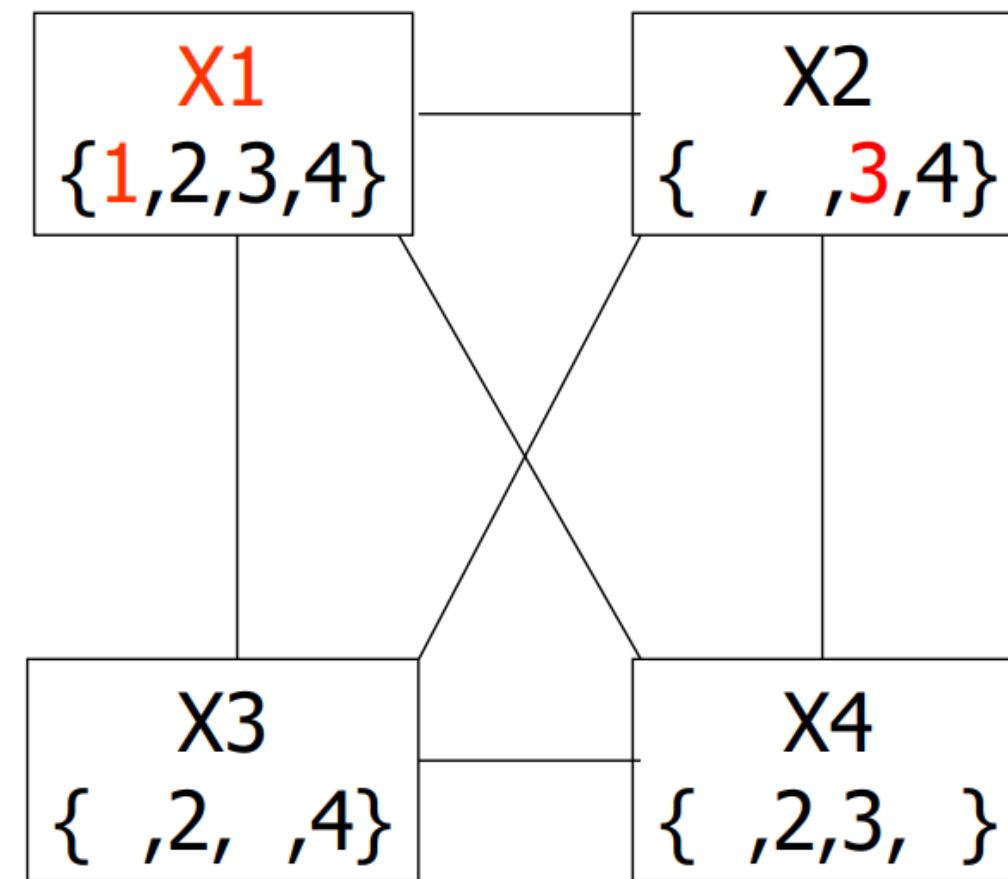
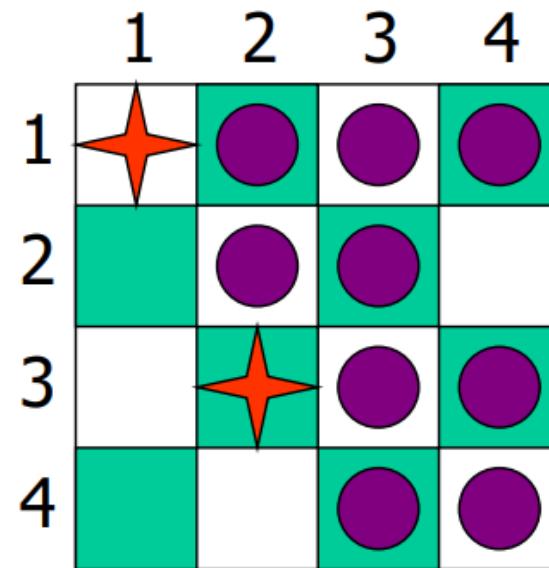
# Ví dụ tìm kiếm tiến



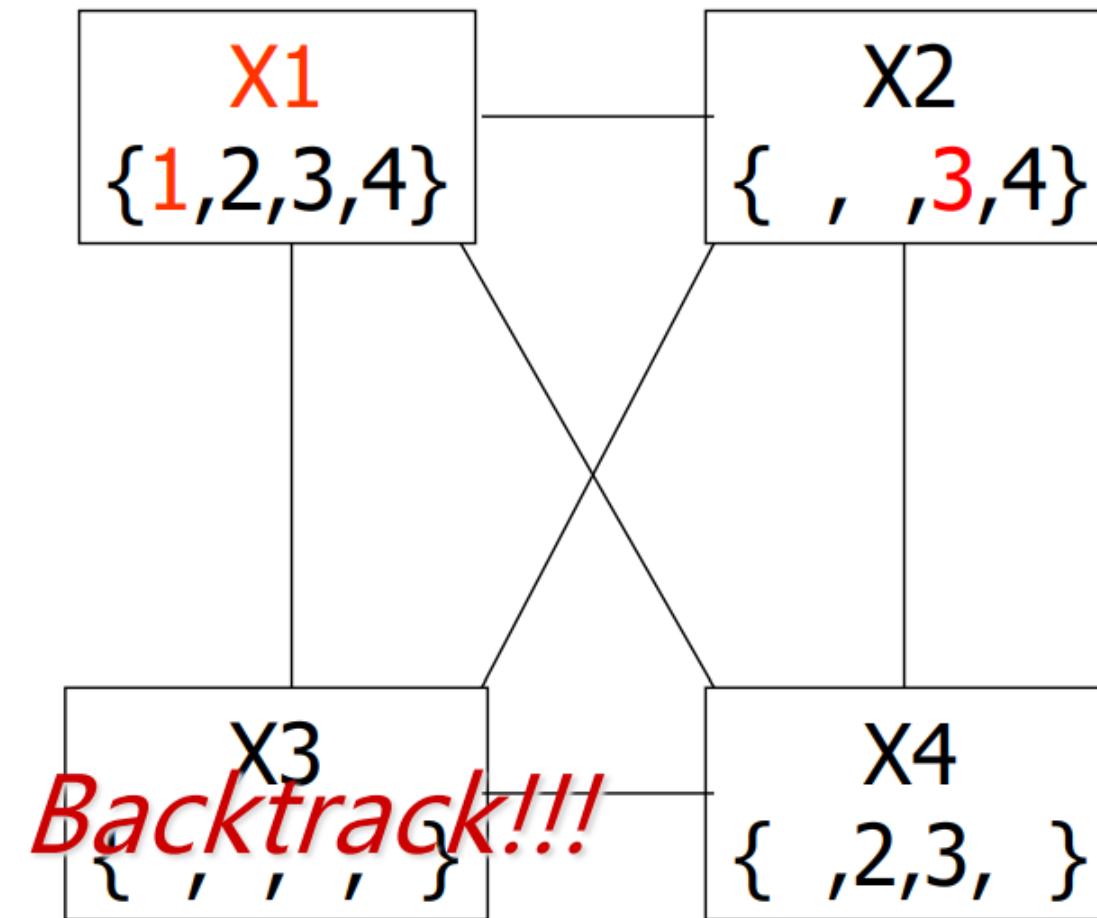
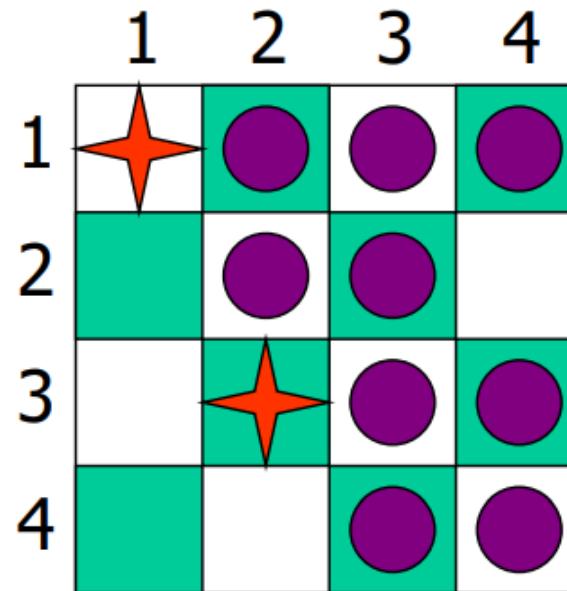
# Ví dụ tìm kiếm tiến



# Ví dụ tìm kiếm tiến

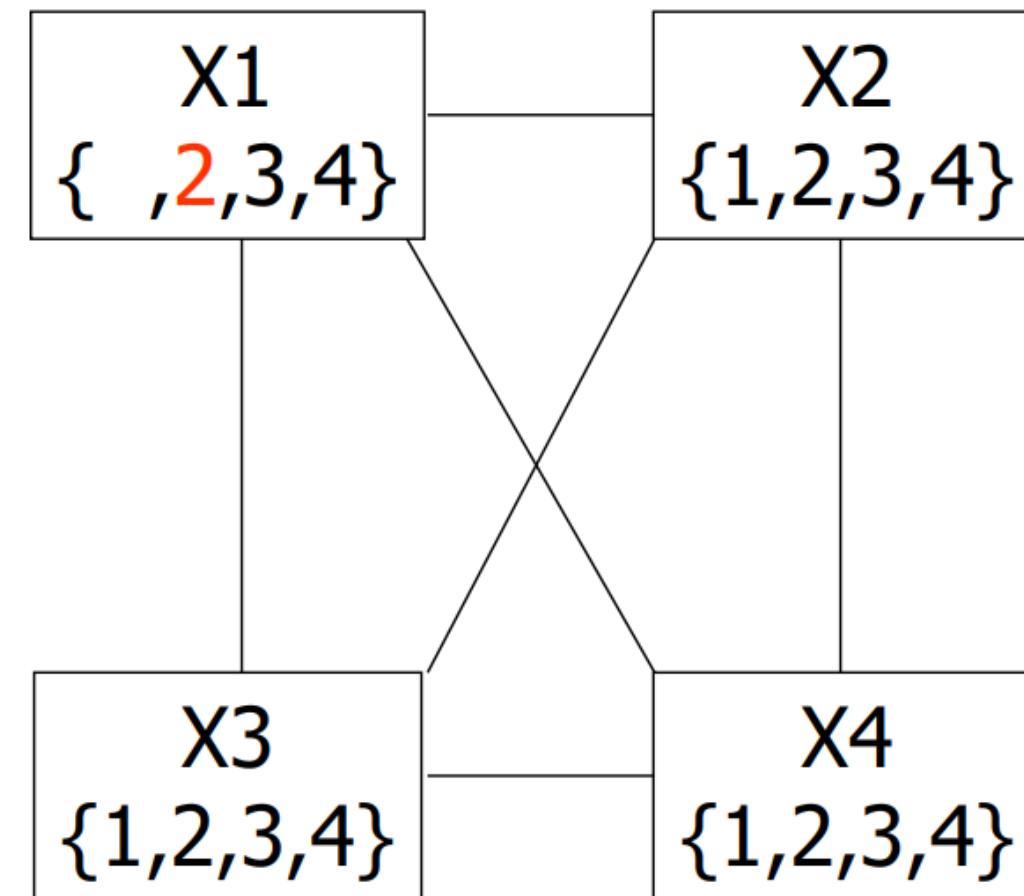


# Ví dụ tìm kiếm tiến

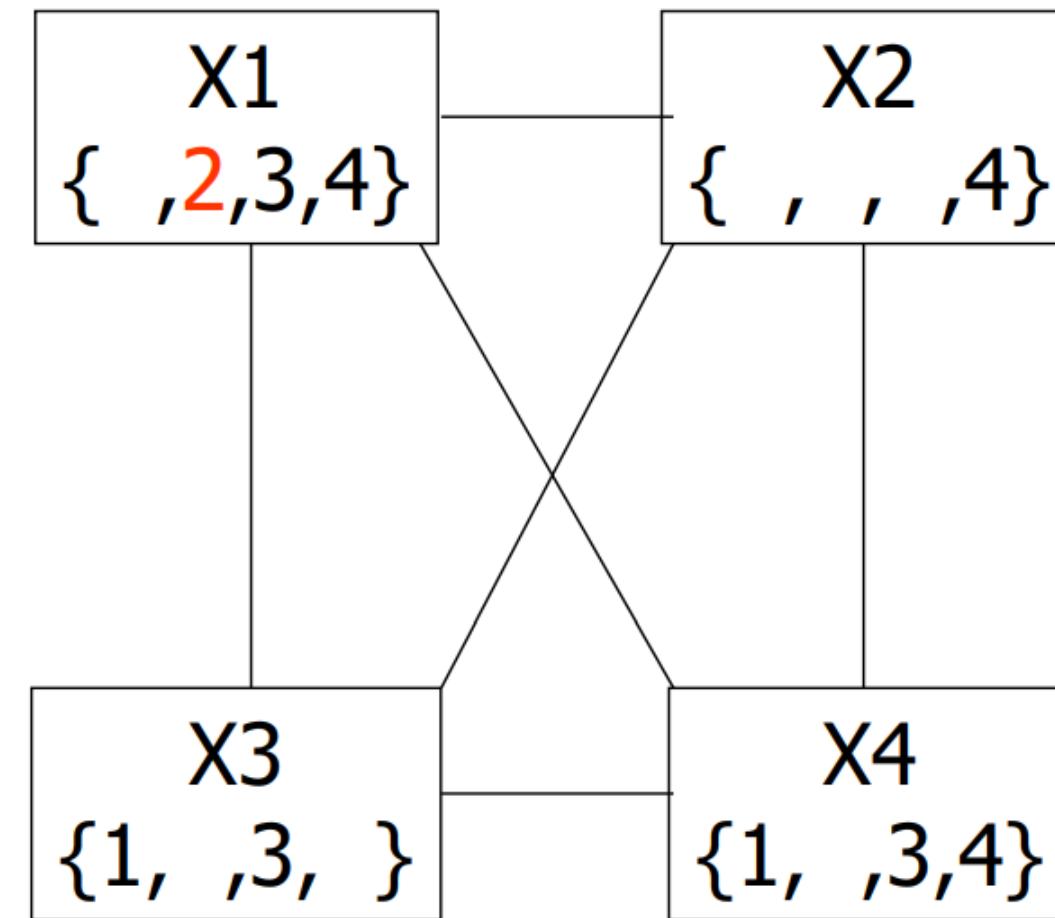
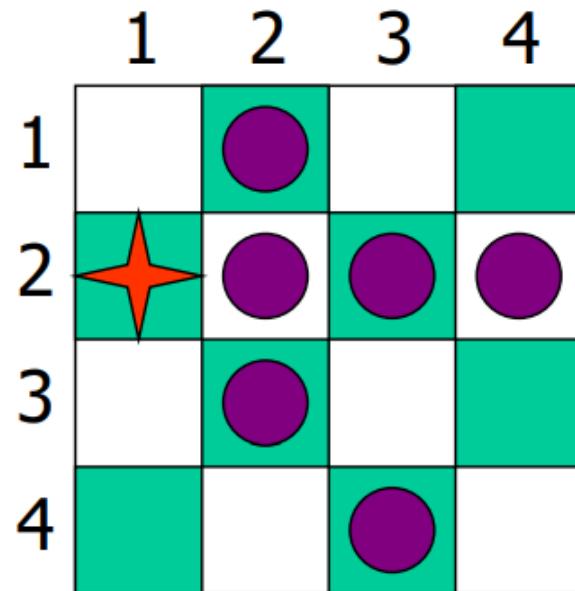


# Ví dụ tìm kiếm tiến

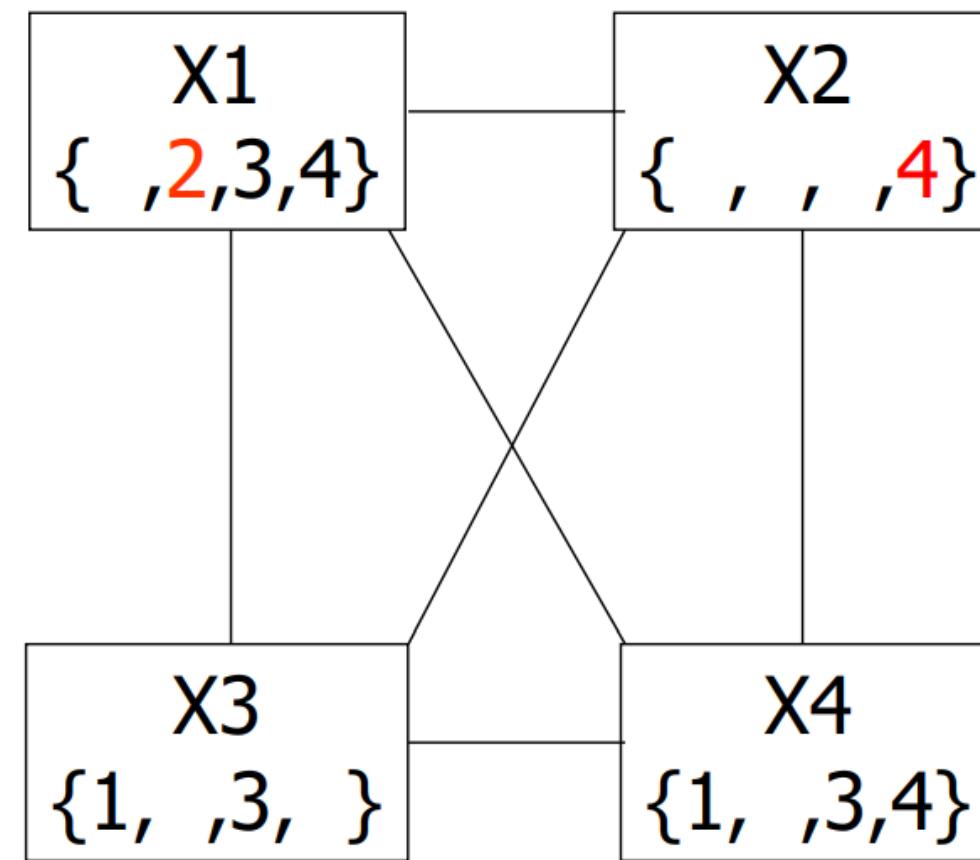
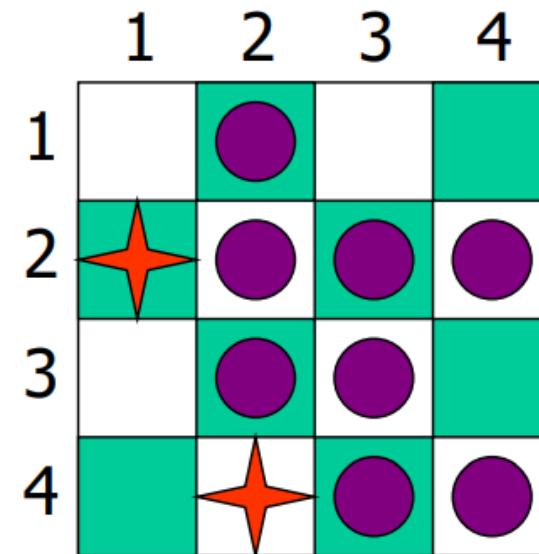
	1	2	3	4
1				
2		★		
3				
4			★	



# Ví dụ tìm kiếm tiến

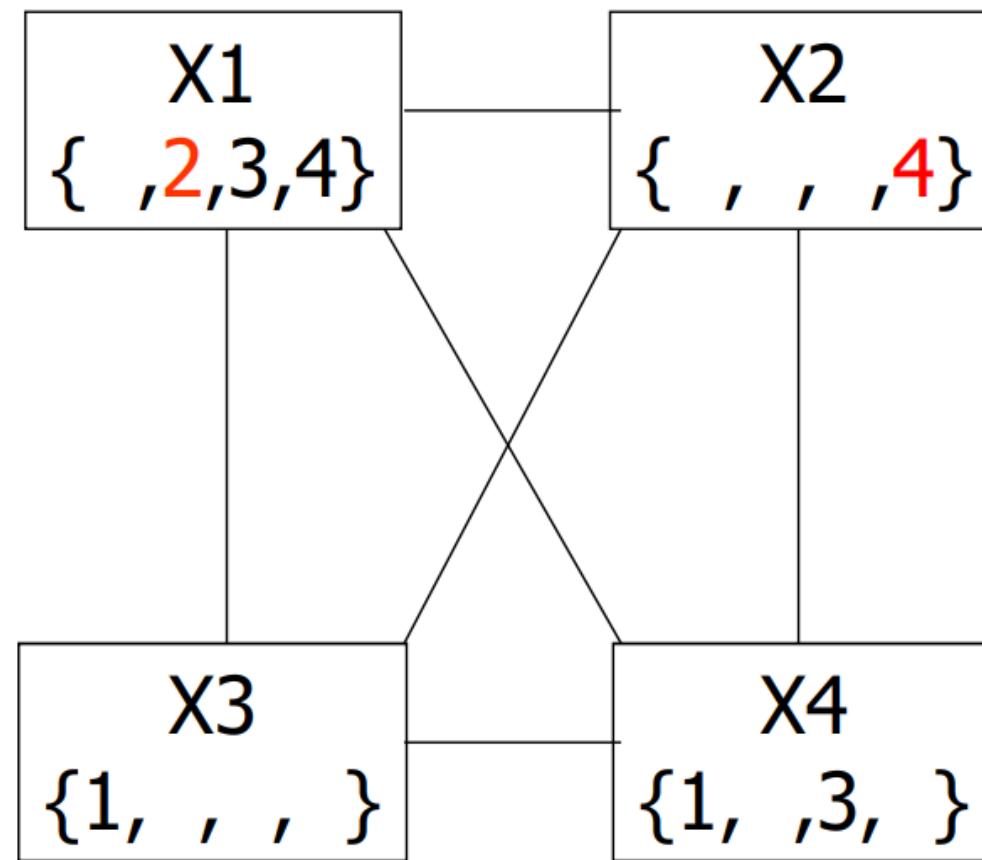


# Ví dụ tìm kiếm tiến

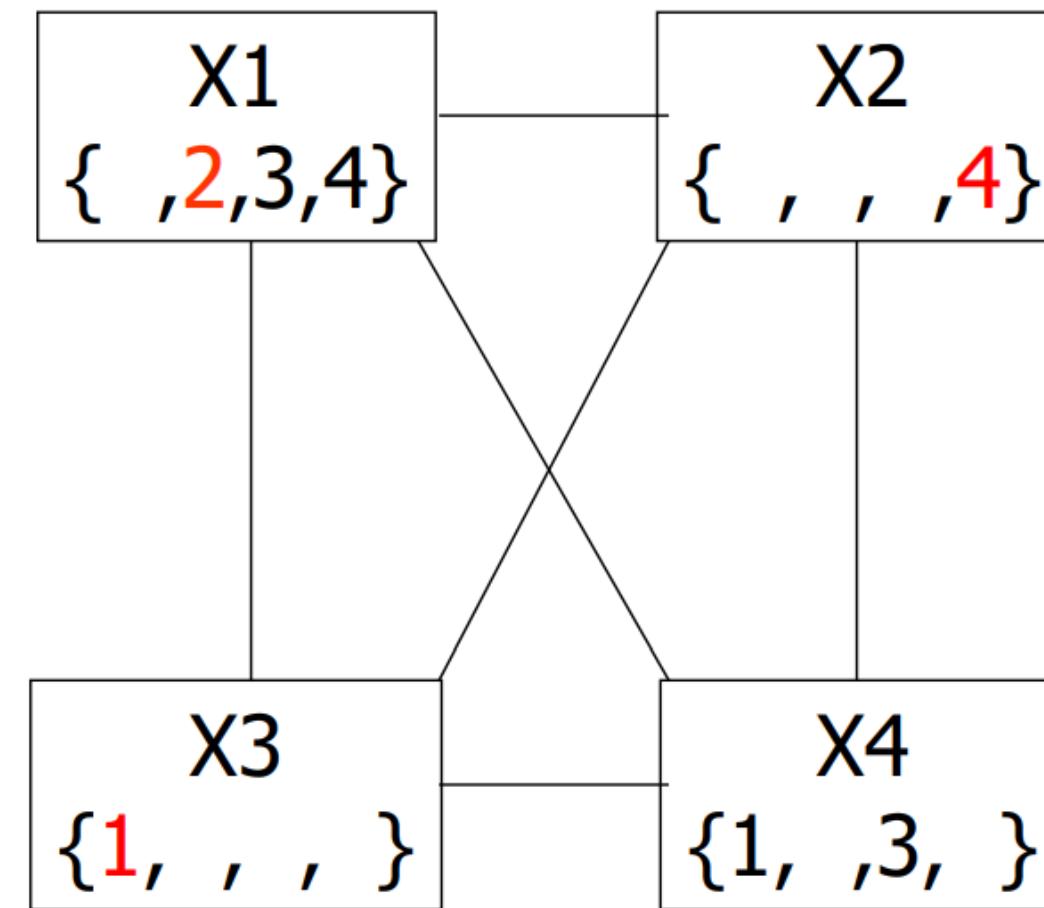
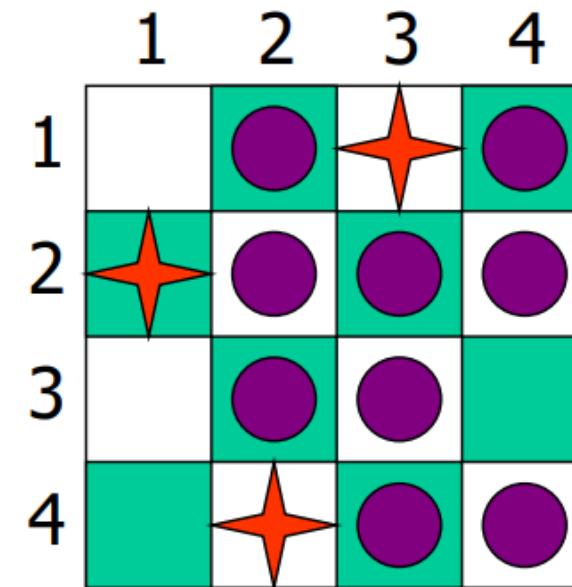


# Ví dụ tìm kiếm tiến

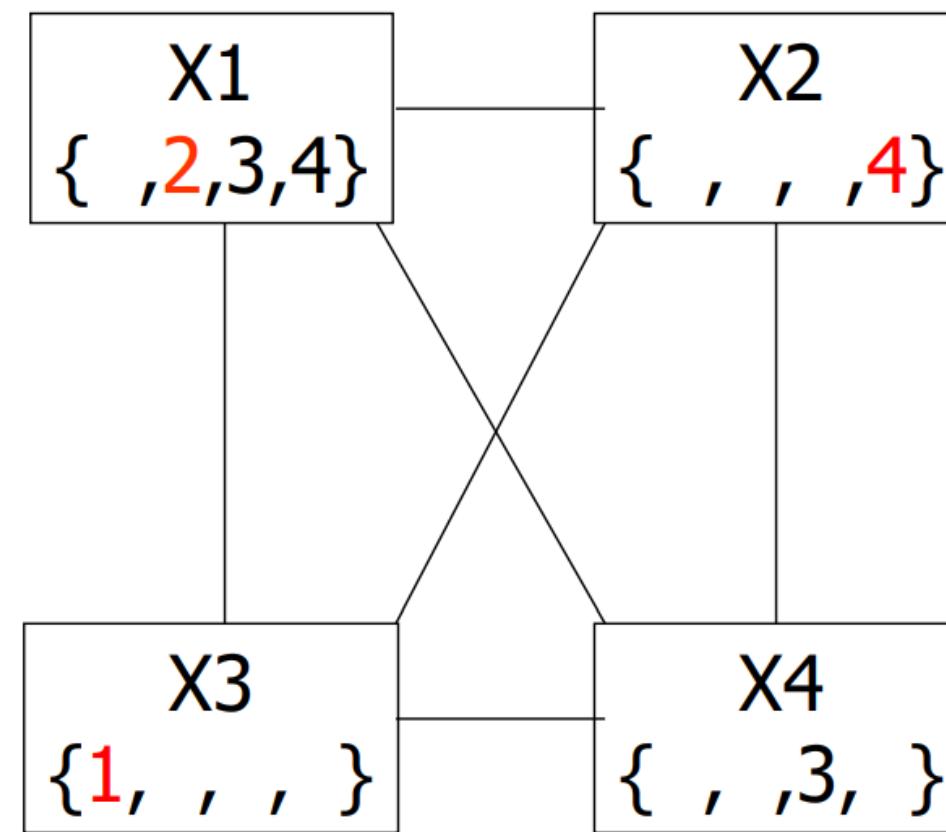
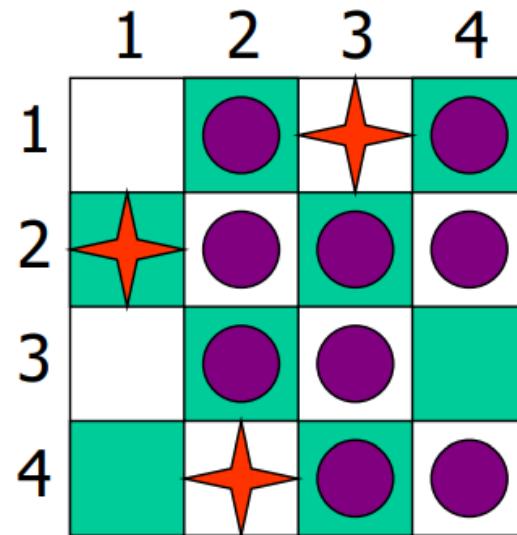
	1	2	3	4
1				
2				
3				
4				



# Ví dụ tìm kiếm tiến



# Ví dụ tìm kiếm tiến



# Ví dụ tìm kiếm tiến

