

# NHẬP MÔN TRÍ TUỆ NHÂN TẠO

## CHƯƠNG 2

### GIẢI QUYẾT VẤN ĐỀ BẰNG TÌM KIẾM

Trần Nguyễn Minh Thư  
tnmthu@ctu.edu.vn

1

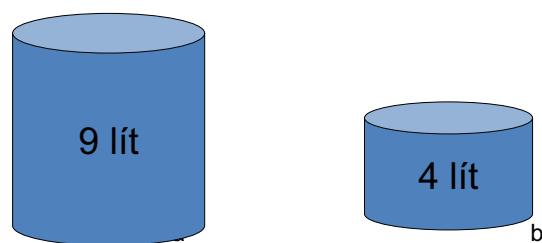
## Nội dung

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

2

1

## Ví dụ: Bài toán đong nước



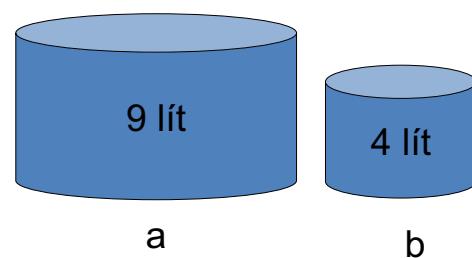
Cho 2 bình 9 lít và 4 lít, không có vạch chia, và 1 vòi bơm.  
Làm cách nào đong **được 6 lít**?

3

3

## 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>
		Goal



4

4

## 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

5

5

## 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

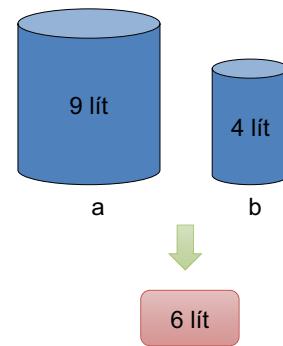
6

3

## 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?*



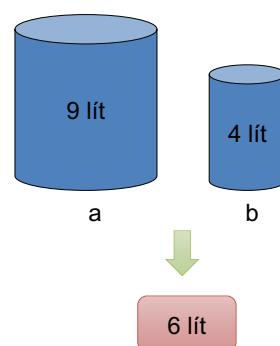
7

7

## 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?*

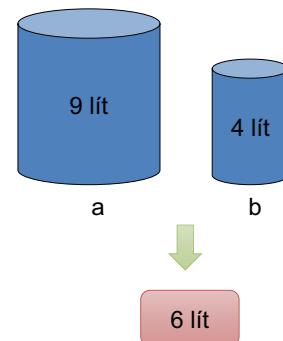


8

8

## 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



9

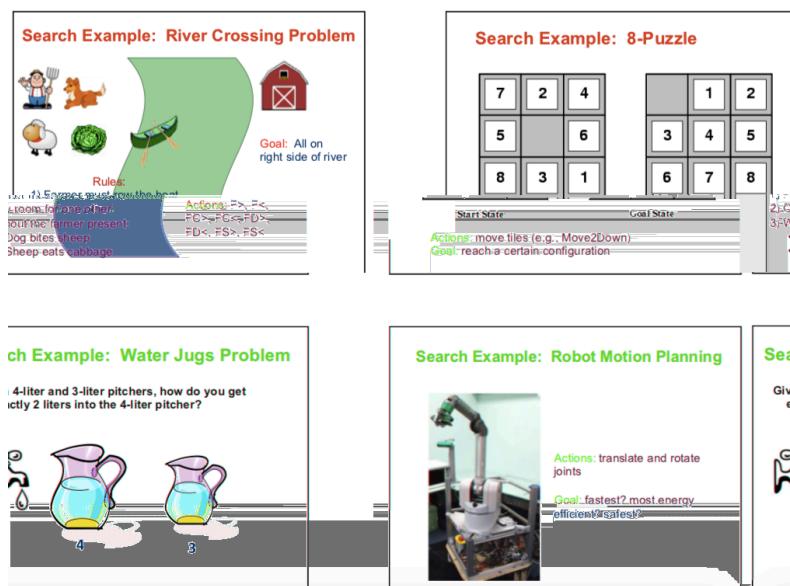
9

## 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
  - Games
  - Navigation
  - Assignment
  - Motion planning
  - Scheduling
  - Routing

10

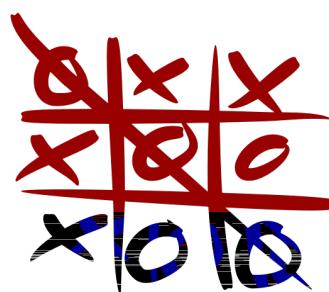
## AI – Giải quyết vấn đề bằng tìm kiếm



11

## 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*



12

12

## Ví dụ: Trò chơi Tic--Đắc-ci-đắc-uu-Tđc-TTui

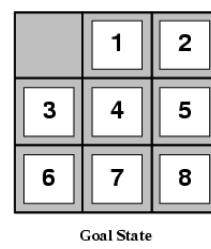
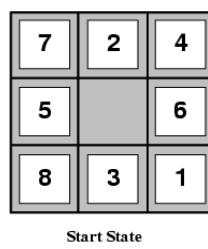
### ■ Phân tích:

- *ác trạng thái*: *mỗi lượt i sẽ tạo n n trạng thái m i*
- *ác hành động*: *i X hoặc O*
- *Mục tiêu*: *dây ký hiệu giống nhau ngang, dọc hoặc chéo*
- *hi phí*: *mỗi lượt i)*

## Ví dụ: Trò chơi 8 ô số (8-Puzzle)

### ■ Cần xác định:

- *ác trạng thái*
- *ác hành động*
- *Mục tiêu*
- *chi phí di chuyển*



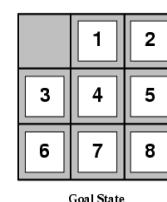
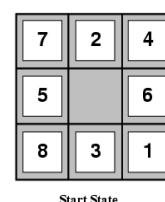
15

15

## 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)

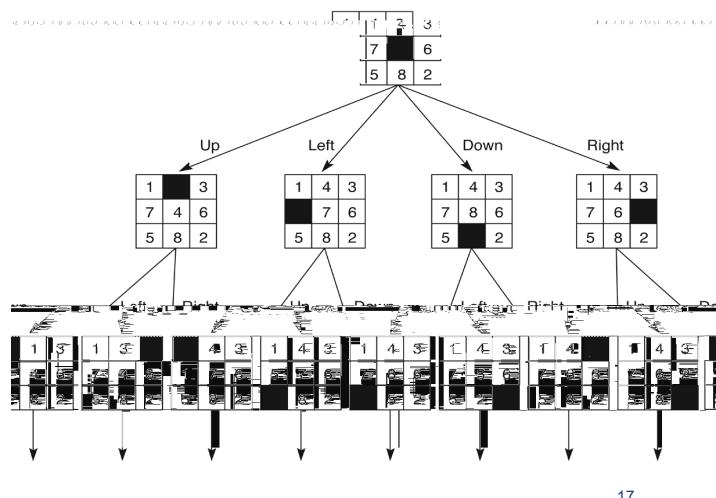


16

16

## Biểu diễn bài toán trên KGTT

### ■ Trò chơi 8-Puzzle

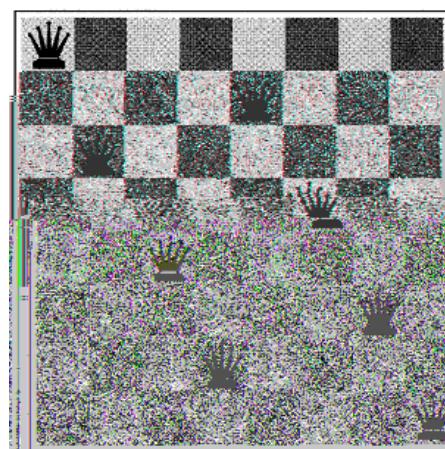


17

17

## 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



18

## 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 không chế qua một ô khác trong cùng một cột.*
- Hàm kiểm tra mục tiêu:
  - *Có quân hậu nào còn bị không chế không*

21

## 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
- **M** **t** **tr** **ng** **thái** **ích** **mong**      **c:** là trạng thái cuối cùng thỏa mãn mục tiêu yêu cầu

22

22

## G c c c b

- Biểu diễn biến trong KGTT theo cách tìm kiếm
- Các bước xem xét 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*
- Táp cung cấp thông tin cần thiết để xác định, vì cách tìm kiếm thông tin ở mức độ cao

23

23

## Giải quyết bài toán bằng tìm kiếm

### Chiến lược tìm kiếm:

- Tầm tìm kiếm ban đầu là trang thái ban đầu, điều kiện sinh ra các trạng thái mới (một khung thời gian trang thái)
- Tầm culling, quản lý trạng thái, chỉ cho những trạng thái chưa khám phá
- Táp cung cấp thông tin để xác định, vì cách tìm kiếm thông tin ở mức độ cao

24

24

*Phương pháp tìm kiếm xác định thử 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 trọng vụ:** 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?

25

25

## C c

- **Tìm kiếm mù (uninformed/blind search):**
  - 
  - **Tìm kiếm dựa trên kinh nghiệm (informed/ heuristic search):**
    -

26

26

# TÌM KIẾM MÙ (UNINFORMED BLIND SEARCH)

27

T

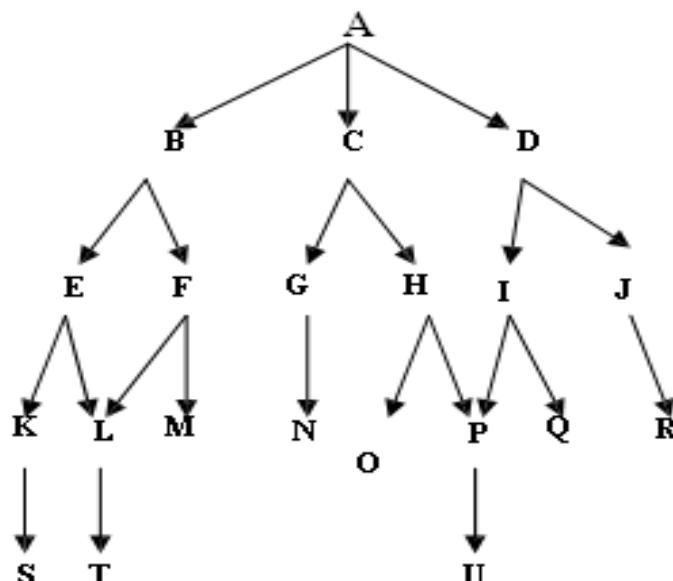
- Tìm kiếm rộng (breadth-first search)
- Tìm kiếm sâu (depth-first search)
- Tìm kiếm theo độ sâu có giới hạn (depth-limited search)
- Tìm kiếm theo chiều sâu lặp (iterative deepening depth-first search)

28

28

T

■ Xét KGTT sau:



29

29

T

(b ea -f ea c - 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;**

30

30

T:  
 ■ T : ; ;  
 ■ C :  
 1. Open = [A]; closed = []  
 2. Open = [B,C,D]; closed = [A]  
 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  
 Đưa các con còn lại của X vào đầu bên phải của open; % hinzufügen  
 end;  
 End;  
 Trả lời kết quả (thất bại); % không còn trạng thái nào  
 End;

30      31

31

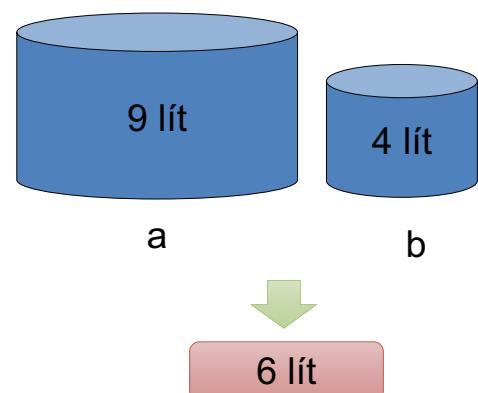
7  
 T:  
 ■ T : ; ;  
 ■ C :  
 1. Open = [A]; closed = []  
 2. Open = [B,C,D]; closed = [A]  
 3. Open = [C,D,E,F];closed = [B,A]  
 4. Open = [D,E,F,G,H];  
 closed = [C,B,A]  
 5. Open = [E,F,G,H,I,J];  
 closed = [D,C,B,A]  
 6. Open = [F,G,H,I,J,K,L];  
 closed = [E,D,C,B,A]  
 7. Open = [G,H,I,J,K,L,M];  
 (vì L đã có trong open);  
 closed = [F,E,D,C,B,A]  
 8. ...

32

32

## Ví dụ: Bài toán đong nước

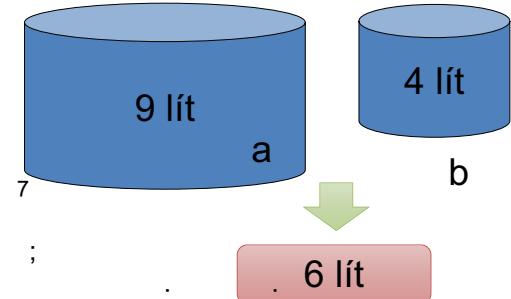
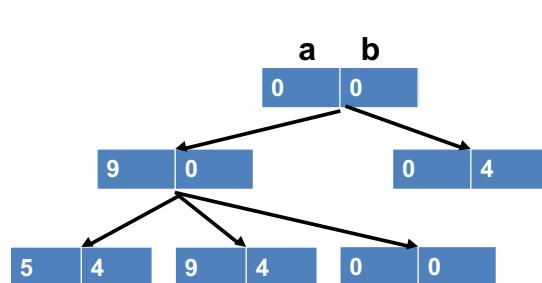
Start      a      b  
 0      0  
 1



33

33

## Ví dụ: Bài toán đong nước



- T
  - C :
1.  $Open = [A: 0-0]; closed = []$
  2.  $Open = [B: 9-0, C: 0-4]; closed = [A: 0-0]$
  3.  $Open = [C: 0-4, D: 5-4, E: 9-4, F: 0-0]; closed = [B, A]$
  4. ....

34

34

## 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;
                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;
    End;

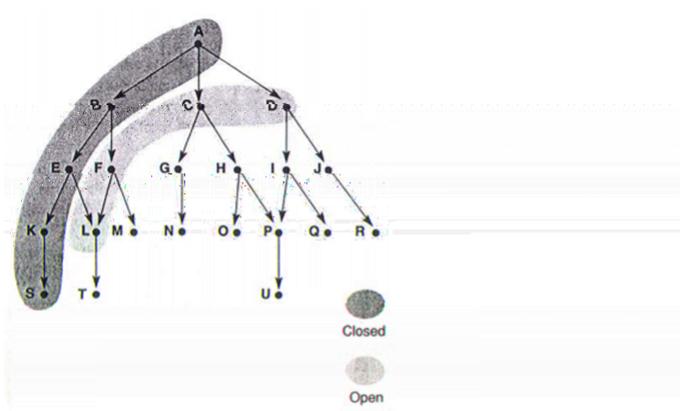
```

35

35

T 7 (de -f ea c - DFS)

- T ;
- C : ; ; :
- 1. O = [A]; = [] = [A]
- 2. O = [B,C,D]; = [A]
- 3. O = [E,F,C,D]; = [B,A]
- 4. O = [K,L,F,C,D]; = [E,B,A]
- 5. O = [S,L,F,C,D]; = [K,E,B,A]
- 6. O = [L,F,C,D]; = [S,K,E,B,A]
- 7. O = [T,F,C,D]; = [L,S,K,E,B,A]
- 8. O = [F,C,D]; = [T,L,S,K,E,B,A]



36

36

## Sử dụng BFS, DFS để tìm trạng thái đích

3	1
2	

Start

1	2
3	

Goal

## 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

40

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

- 
- *Vd: lần thứ nhất giới hạn độ sâu = 1, nếu tìm không thấy tăng độ sâu giới hạn lên 2, ...*

41

41

Tìm kiếm sâu lặp,  $l = 0$

Limit = 0

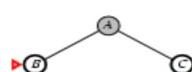


42

42

Tìm kiếm sâu lặp,  $l = 1$

Limit = 1

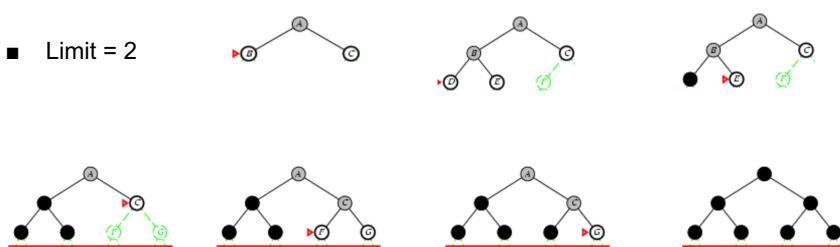


43

43

## Tìm kiếm sâu lấp, $l = 2$

■ Limit = 2

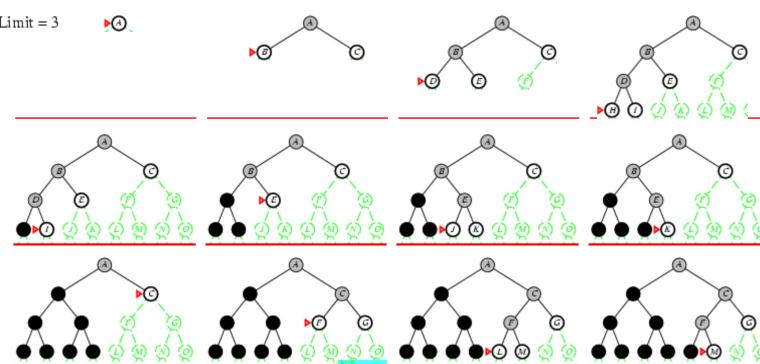


44

44

## Tìm kiếm sâu lấp, $l = 3$

Limit = 3



45

45

## Sử dụng BFS, DFS để tìm trạng thái đích

3	1
2	

Start

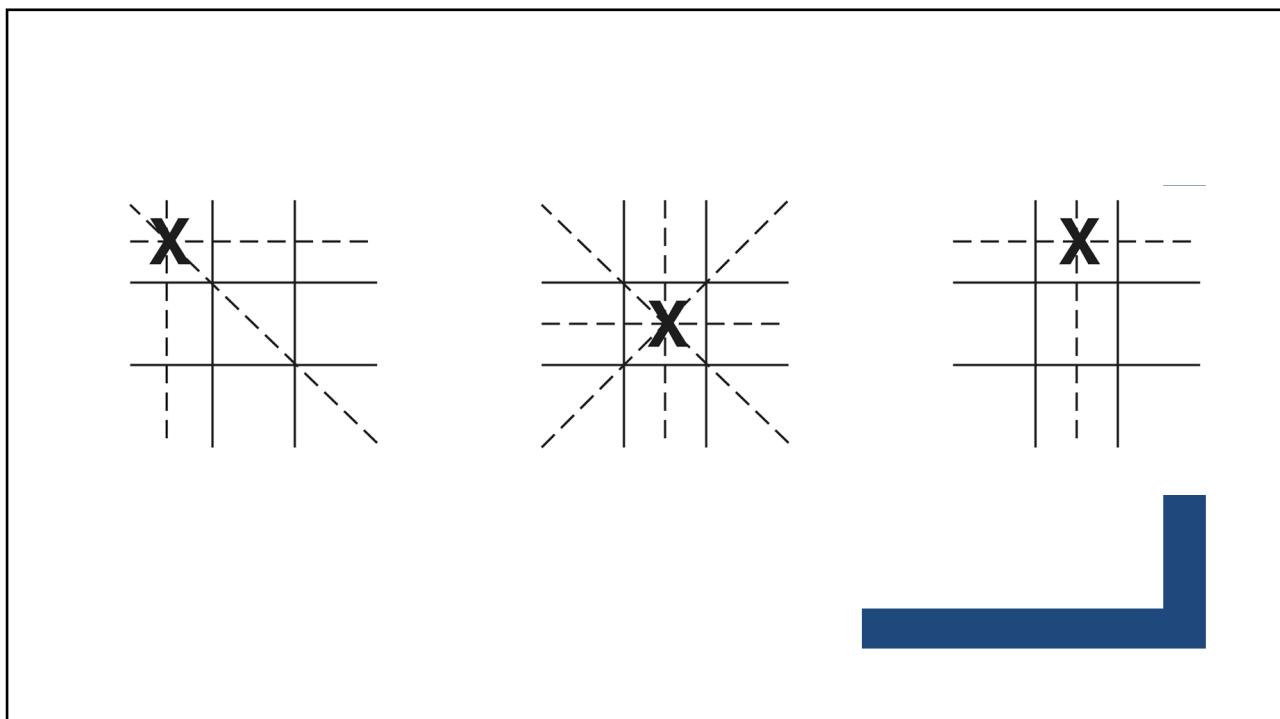
1	2
3	

Goal

46

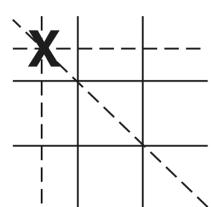
TÌM KIẾM DỰA TRÊN KINH NGHIỆM  
(INFORMED/ HEURISTIC SEARCH)

47

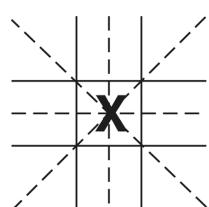


48

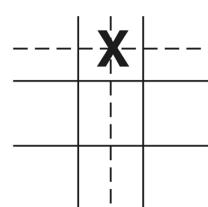
## Phép đo heuristic (2)



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.

49

## 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

50

50

## Heuristic

- **Sử dụng Heuristic trong trường hợp nào?**
  - Vấn đề có thể có giải pháp chính xác, nhưng **chỉ 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,...

51

51

## Heuristic

- Thuật toán Heuristic gồm hai phần:

1. **Phép đọ 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 hau ăn (greedy best-first search)
  - Giải thuật A\*
- Tìm kiếm leo đồi

52

52

## 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):

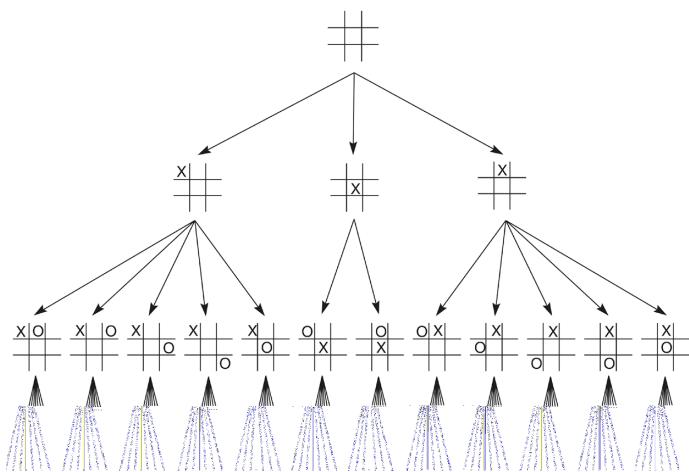
$$f(n) = g(n) + h(n)$$

- $g(n)$  = 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**

53

53

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

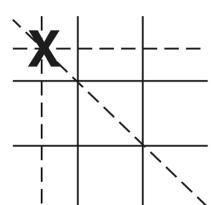


C 4 – Tìm kiếm Heuristic

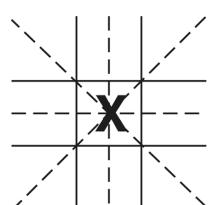
TTNT. p.54

54

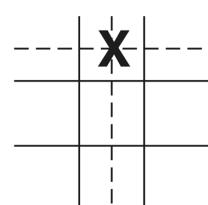
## Phép đo heuristic (2)



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.

C 4 – Tìm kiếm Heuristic

TTNT. p.55

55

## Cài đặt hàm đánh giá Heuristic

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

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

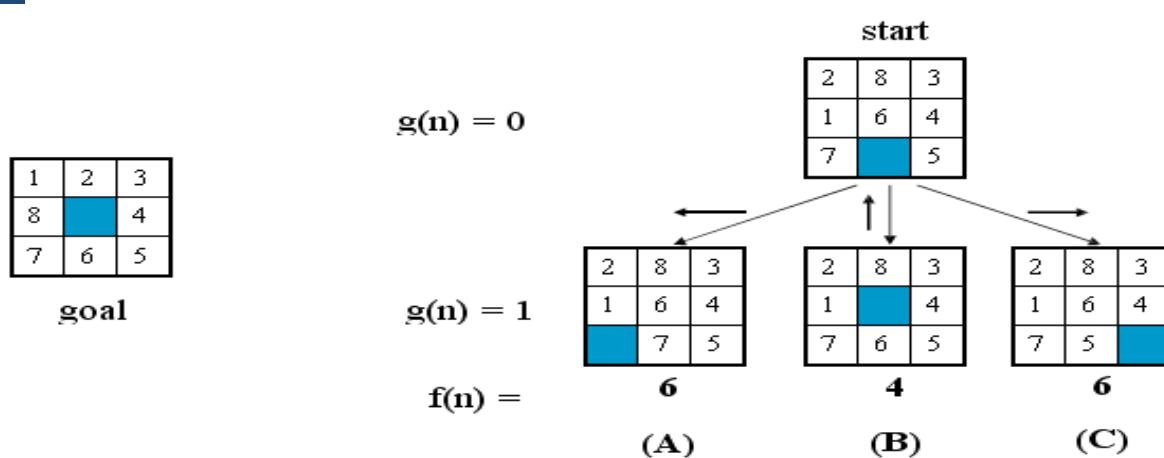
- $g(n) = \text{khoảng cách thực sự từ } n \text{ đến trạng thái bắt đầu}$
- $h(n) = \text{ước lượng heuristic cho khoảng cách từ trạng thái } n \text{ đến trạng thái đích}$

56

56

## Cài đặt hàm đánh giá Heuristic

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



57

## 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)
- =>  $h_1(n) = ???$
- =>  $h_2(n) = ???$

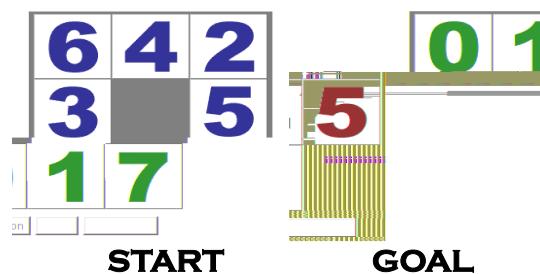


58

58

## 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 sai khác của n so với goal
    - $h_2(n)$  = khoảng cách Manhattan
- =>  $h_1(n) = ???$
- =>  $h_2(n) = ???$
- $h_1(n) = 6.$
  - $h_2(n) = 8.$



59



## 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

62

62

## 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
        //và loại bỏ nó ra khỏi đường
        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 chia child->state
                ra khỏi explored
                frontier.insert(child);
        }
    }
    return NULL; //Thất bại
}

```

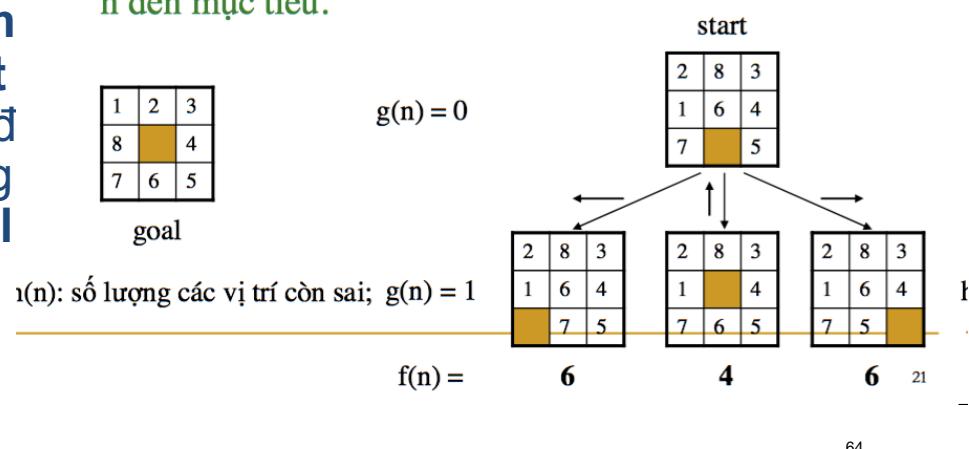
63

## 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  $f(n)$ :

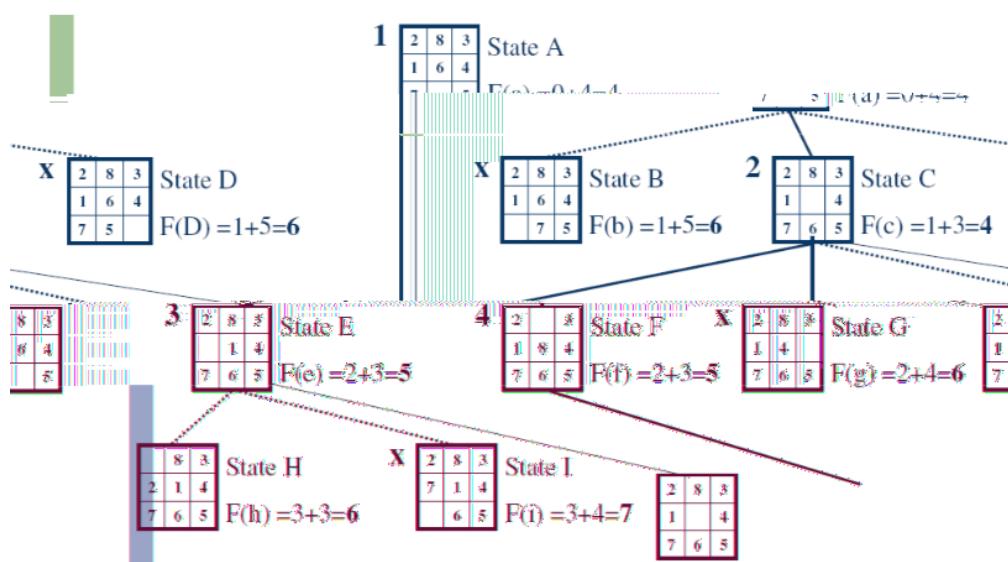
$f(n) = g(n) + h(n)$

- $g(n)$  = 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.

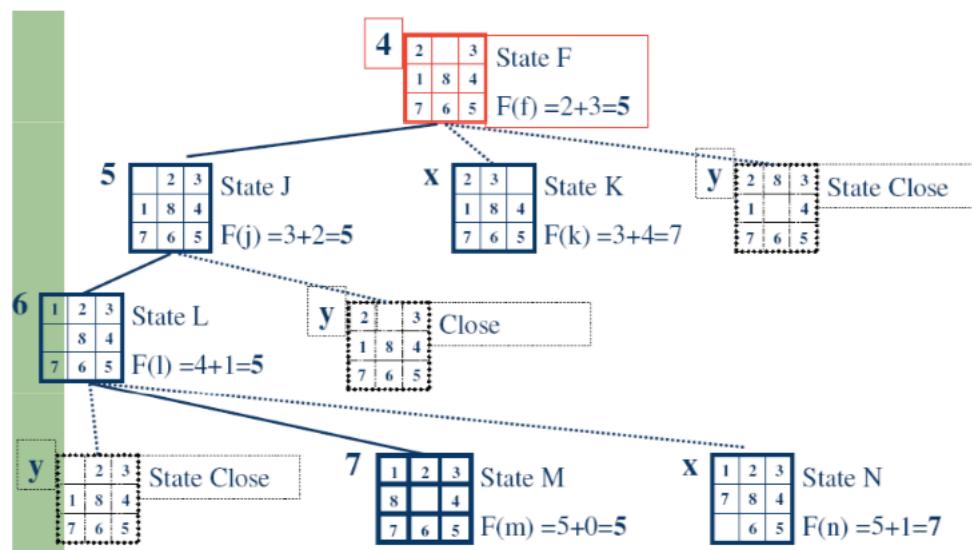


64

## Ví dụ



65



66

## *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

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

67

67

## Tìm kiếm heuristic

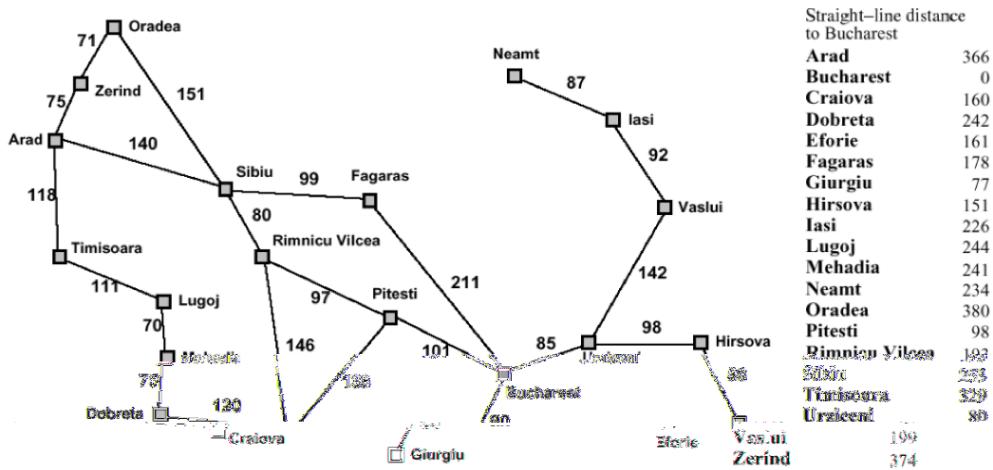
Bài toán tìm đường:

- 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. ( $h(n)$ )

68

## Tìm kiếm heuristic

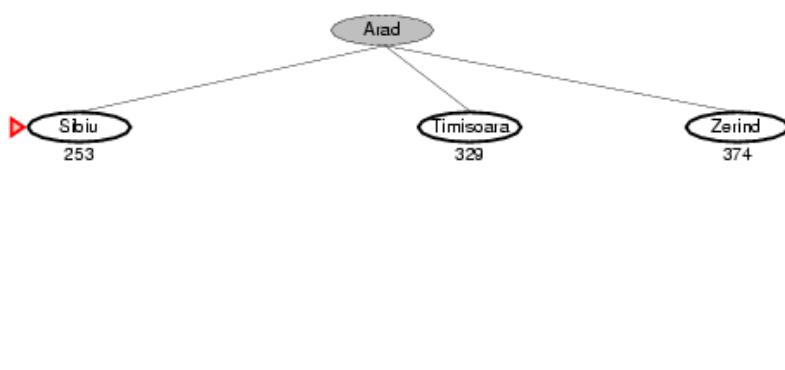
- Initial State =
- Goal State =



69

## Tìm kiếm heuristic

- 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
Urzicent	88
i	199
d	374