Chương 2: Chiến lược tìm kiếm mù

Nội dung

- Bài toán
  - Biểu diễn bài toán
  - Tìm kiếm
- Các chiến lược điều khiển
- Các đặc trưng của bài toán
- Vấn đề trong thiết kế chương trình tìm kiếm

Mô hình ứng dụng của TTNT

TTNT = Presentation & Search

Các lại bài toán tìm kiếm

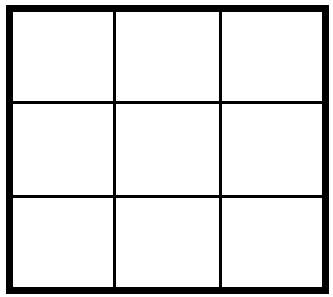
- Fully observable, deterministic
  - single-belief-state problem
- Non-observable
  - sensorless (conformant) problem
- Partially observable/non-deterministic
  - contingency problem
  - interleave search and execution
- Unknown state space
  - exploration problem
  - execution first

Bài toán

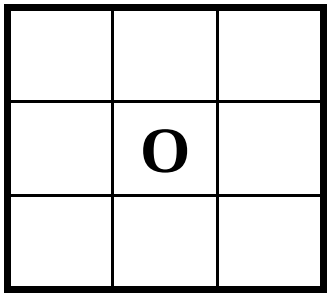
- Giải bài toán bằng cách tìm kiếm, gồm:
  - Cấu trúc bài toán: VD. tìm đường đi trên đồ thị
  - Biểu diễn bài toán bằng không gian trạng thái
  - Giải bài toán = Tìm ra một trạng thái/con đường trong không gian trạng thái (trạng thái đầu -> trạng thái đích)

Bài toán

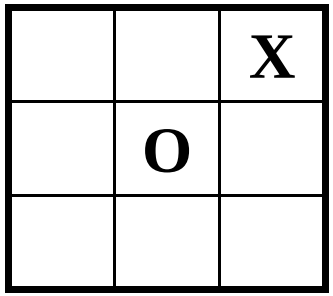
- Trạng thái
  - Biểu diễn một bước nào đó của bài toán
  - Trong trò chơi, như tic-tac-toe, mỗi bàn cờ có thể là trạng thái



Trạng thái



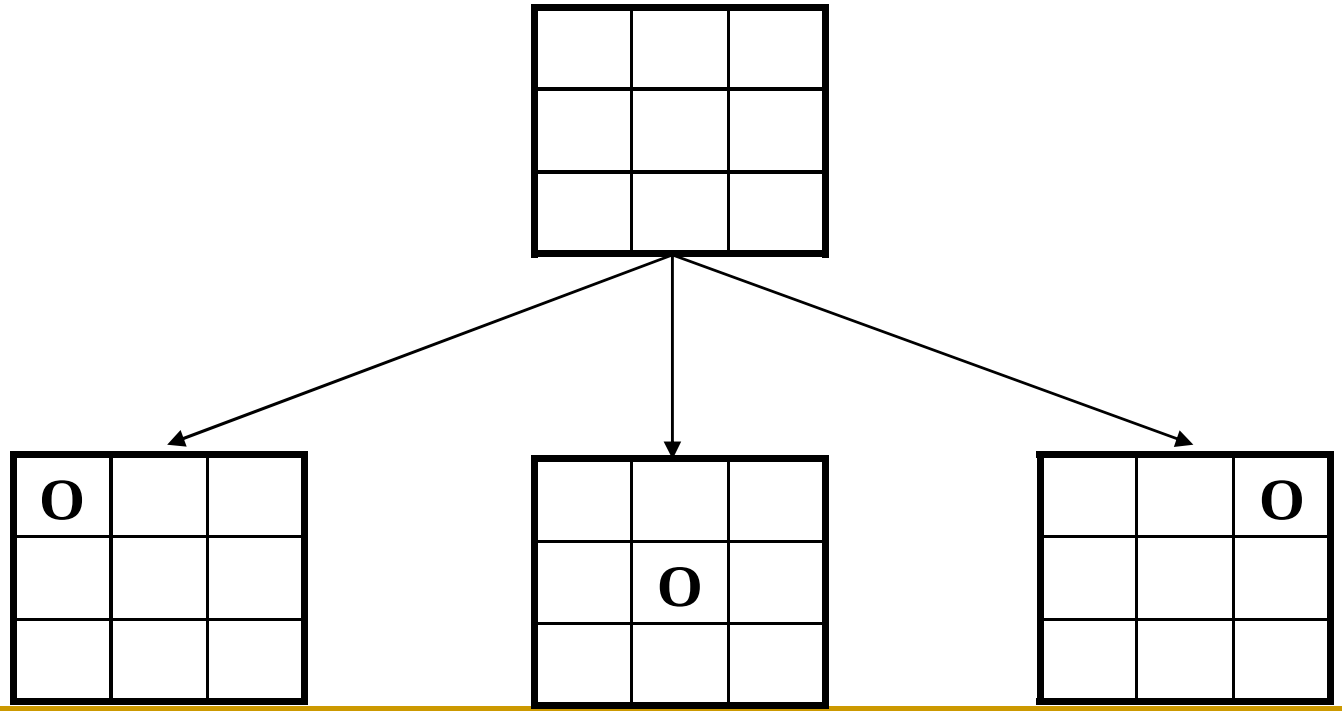
Trạng thái



Trạng thái

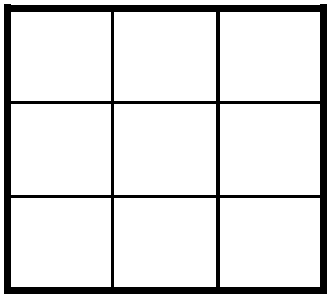
Bài toán (tt)

- Chuyển trạng thái, luật chuyển
  - Biểu diễn cho khả năng của việc chuyển từ trạng thái nào đó đến trạng thái khác.
  - Ví dụ: trong trò chơi, đó là luật chơi của game.

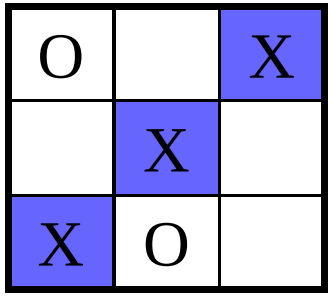


Bài toán (tt)

- Trạng thái đầu
  - Trạng thái xuất phát của bài toán
  - Một bài toán có thể có nhiều trạng thái khởi đầu
  - Ví dụ: game tic-tac-toe, trạng thái rỗng



- Trạng đích
  - Trạng thái mà bài toán đã được giải
  - Một bài toán có thể có nhiều trạng thái đích
  - Ví dụ: game tic-tac-toe, trạng thái đích là:



Bài toán (tt)

- Không gian trạng thái: một hệ thống gồm **4** thành phần [N,A,S,G]
  - N là tập nút của Graph. Mỗi nút là một trạng thái của quá trình giải quyết vấn đề
  - A: Tập các cung nối giữa các nút N. Mỗi cung là một bước trong giải quyết vấn đề. Cung có thể có hướng
  - S: Tập các trạng thái bắt đầu. S khác rỗng.
  - G: Tập các trạng thái đích. G Không rỗng
  - Không gian trạng thái sẽ được xây dựng DẦN khi chương trình chạy
  - Với bài toán lớn, không đủ thời gian, không gian để đặc tả cho từng trạng thái cụ thể, và đường chuyển cụ thể

Bài toán (tt)

- Các vấn đề khó khăn trong tìm kiếm với các bài toán TTNT
  - Đặc tả vấn đề phức tạp
  - Không gian tìm kiếm lớn
  - Đặc tính của đối tượng cần tìm kiếm thay đổi
  - Đáp ứng thời gian thực
- Khó khăn về kỹ thuật
  - Bộ nhớ và tốc độ truy xuất

Bài toán (tt)

State Space

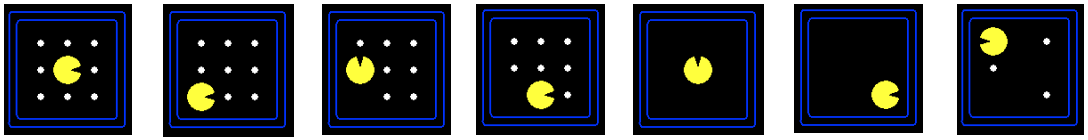
- Không gian tìm kiếm thường là một graph
- Mục tiêu tìm kiếm là một path
- Phải lưu trữ toàn bộ không gian trong quá trình tìm kiếm
- Không gian tìm kiếm biến động liên tục trong quá trình tìm kiếm
- Đặc tính của trạng thái/nút là phức tạp & biến động

Database

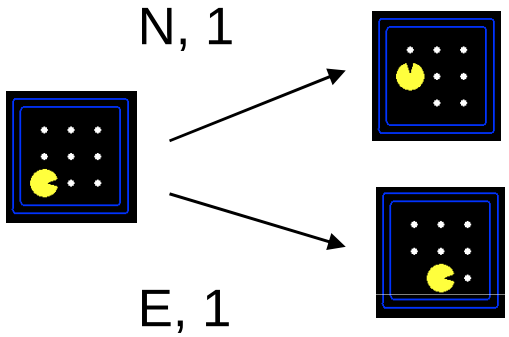
- Không gian tìm kiếm là một danh sách hay cây
- Tìm kiếm một record/nút
- Phần tử đã duyệt qua là không còn dùng tới
- Không gian tìm kiếm là cố định trong quá trình tìm kiếm
- Thuộc tính của một record/nút là cố định

Ví dụ bài toán

- A search problem consists of:
  - A state space



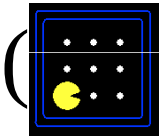
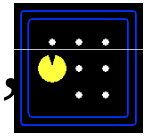
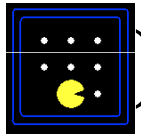
  - A transition model

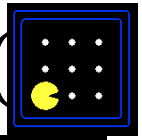
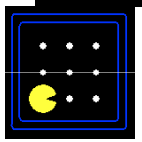
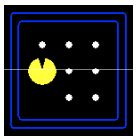
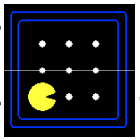
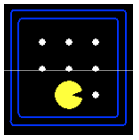
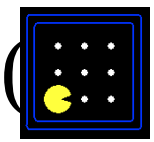
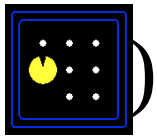
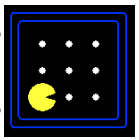
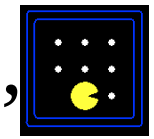


  - A start state, goal test, and path cost function

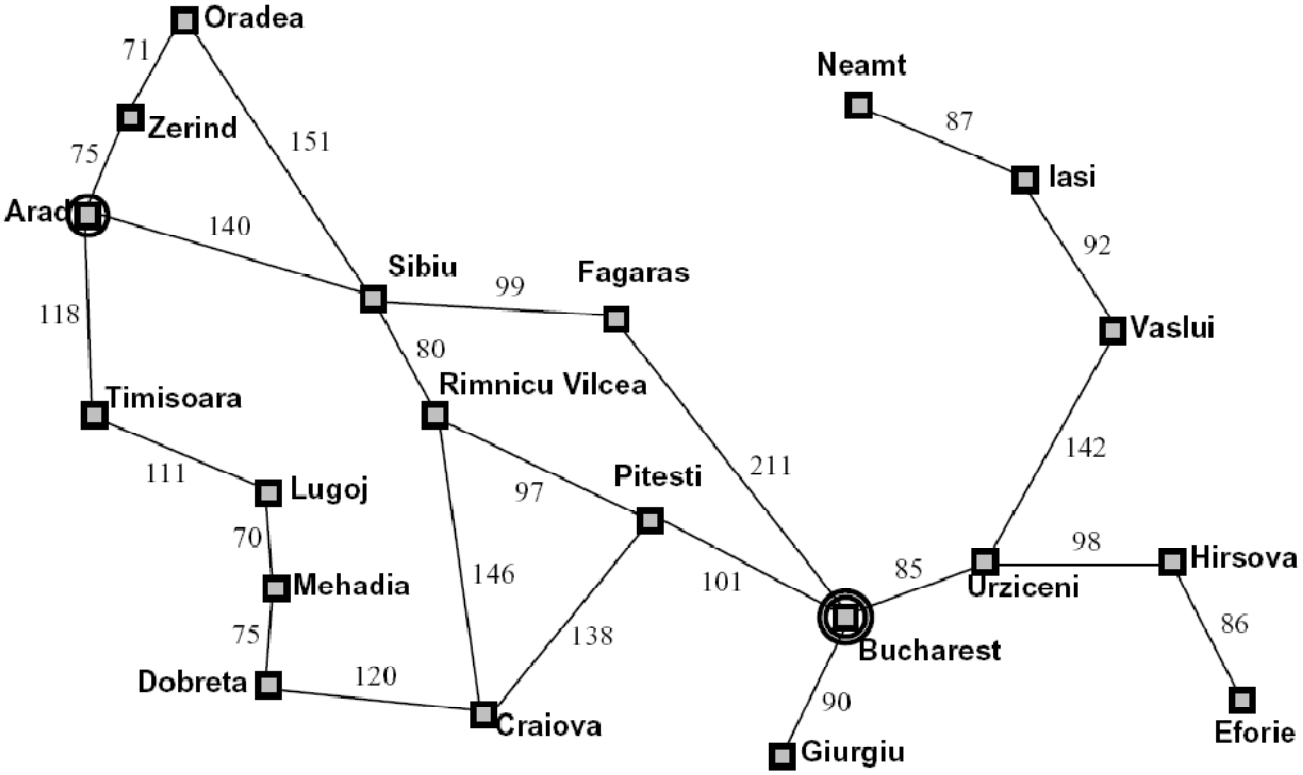
- A solution is a sequence of actions (a plan) which transforms the start state to a goal state

Chuyển trạng thái

- Successor function
  - Successors() = {(N, 1, ), (E, 1, )}

- Actions and Results
  - Actions() = {N, E}
  - Result( , N) =  ; Result( , E) = 
  - Cost( , N, ) = 1; Cost( , E, ) = 1

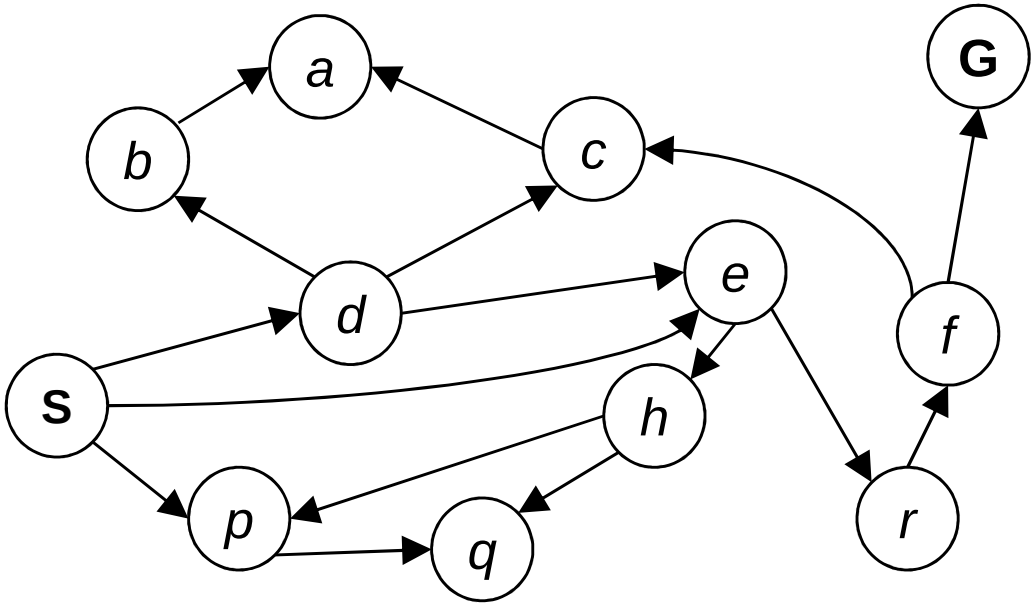
Bài toán Romania



- State space:
  - Cities
- Successor function:
  - Go to adj city with cost = dist
- Start state:
  - Arad
- Goal test:
  - Is state == Bucharest?
- Solution?

Không gian Graphs

- State space graph: A mathematical representation of a search problem
  - For every search problem, there's a corresponding state space graph
  - The successor function is represented by arcs

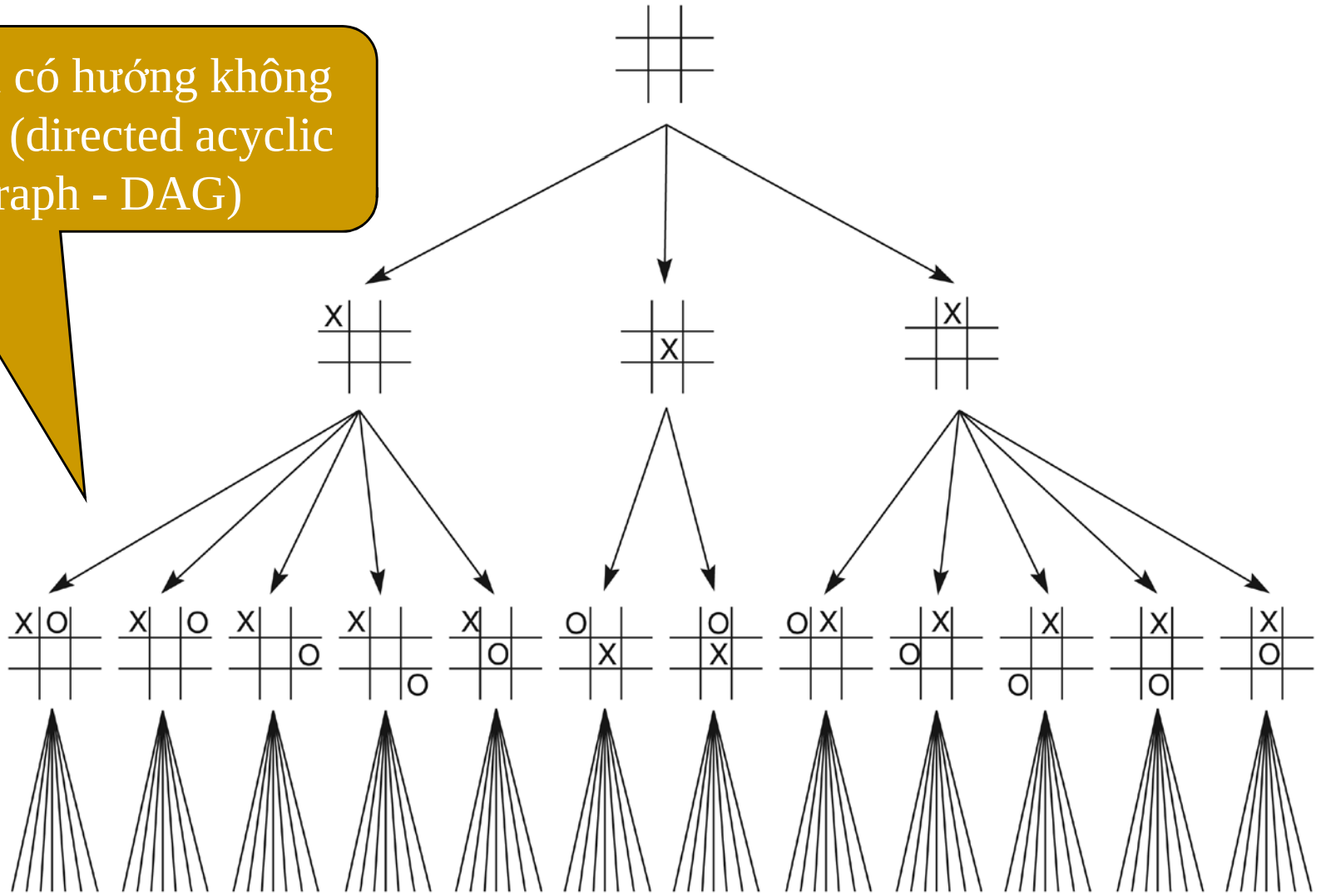


Ridiculously tiny search graph for a tiny search problem

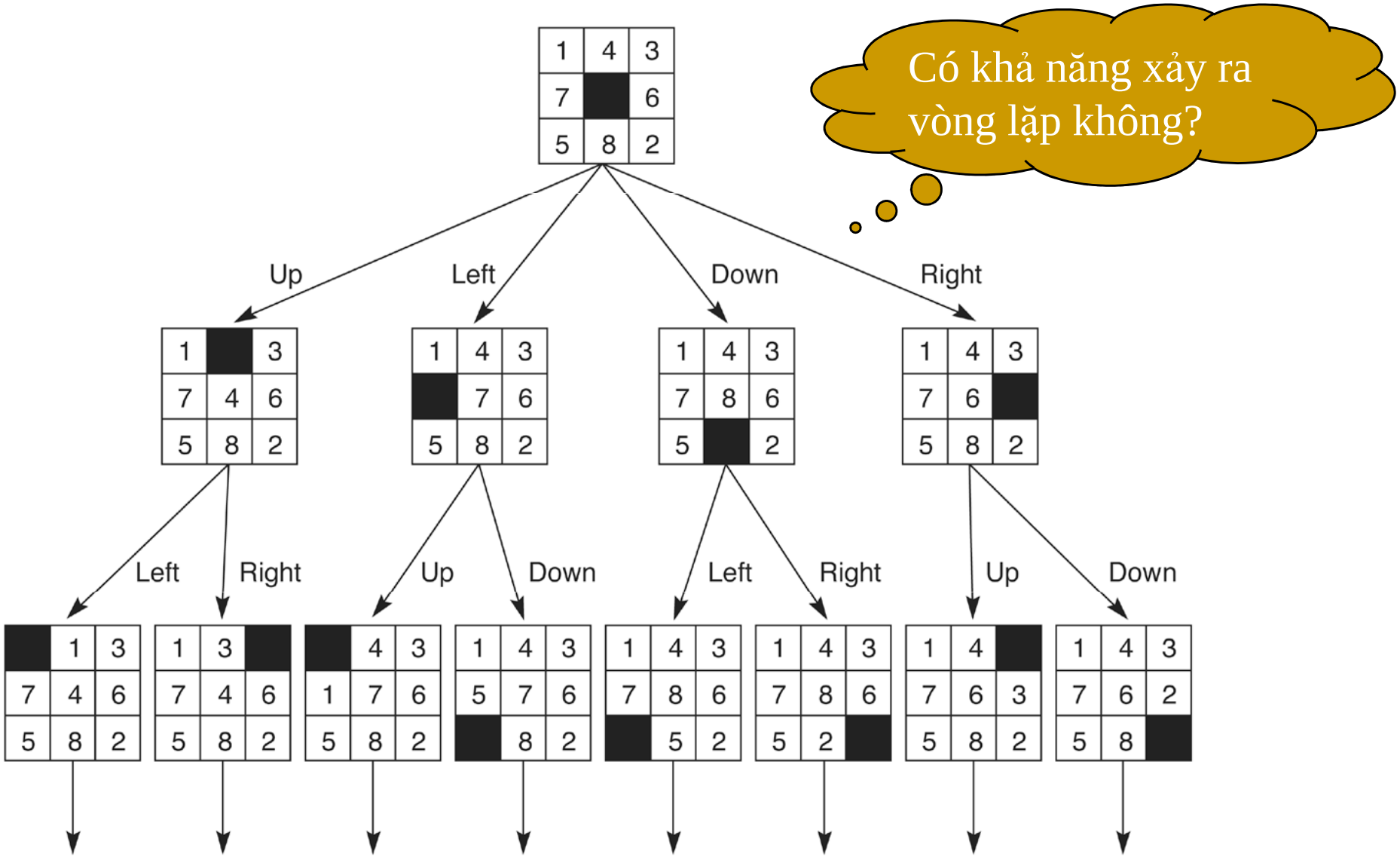
- This can be large or infinite, so we won't create it in memory

Bài toán: Tic tac toe

Đồ thị có hướng không lặp lại (directed acyclic graph - DAG)



Bài toán: 8 puzzle



Chiến lược tìm kiếm?

- Khi tìm kiếm lời giải, từ một trạng thái nào đó chưa phải là trạng thái đích, ta dựa theo toán tử sinh ra tập các trạng thái mới: mở rộng.
- Để được lời giải, ta phải liên tục chọn trạng thái mới, mở rộng, kiểm tra cho đến khi tìm được trạng thái đích hoặc không mở rộng được KGTT.
- Tập các trạng thái được mở rộng sẽ có nhiều phần tử, việc chọn trạng thái nào để tiếp tục mở rộng được gọi là chiến lược tìm kiếm.

Đánh giá một chiến lược?

- Tính đầy đủ: chiến lược phải đảm bảo tìm được lời giải nếu có.
- Độ phức tạp thời gian: mất thời gian bao lâu để tìm được lời giải.
- Độ phức tạp không gian: tốn bao nhiêu đơn vị bộ nhớ để tìm được lời giải.
- Tính tối ưu: tốt hơn so với một số chiến lược khác hay không.

Thông tin mỗi nút?

- Nội dung trạng thái mà nút hiện hành đang biểu diễn.
- Nút cha đã sinh ra nó.
- Toán tử đã được sử dụng để sinh ra nút hiện hành.
- Độ sâu của nút.
- Giá trị đường đi từ nút gốc đến nút hiện hành.

Tìm kiếm mù?

- Trạng thái được chọn để phát triển chỉ đơn thuần dựa theo cấu trúc của KGTT mà không có thông tin hướng dẫn nào khác.
- Nói chung tìm kiếm mù sẽ không hiệu quả.
- Đây là cơ sở để chúng ta cải tiến và thu được những chiến lược hiệu quả hơn.

Breadth-First-Search

- Tạo biến Open.
- Đưa TT bắt đầu vào Open.
- Lặp: (đến khi gặp TT đích) OR (Open trống):
  - E = RemoveFirst(Open).
  - Với mỗi luật so trùng được với E:
    - Áp dụng luật để sinh TT mới.
    - Nếu TT mới là TT đích thì thoát, trả về TT này.
    - Ngược lại: Đưa TT mới vào CUỐI của Open.

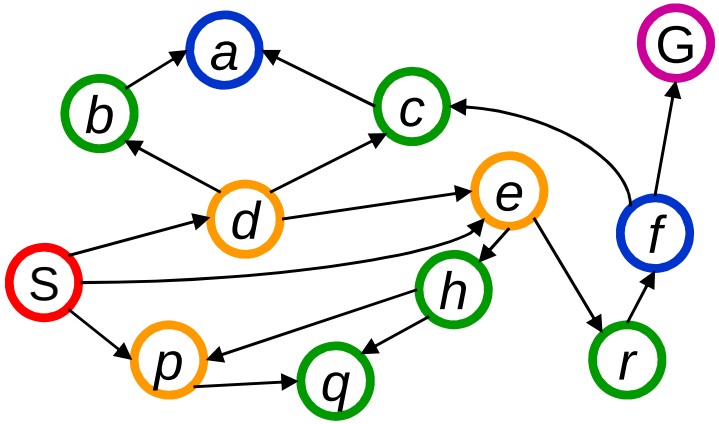
Breadth-First-Search (tt)

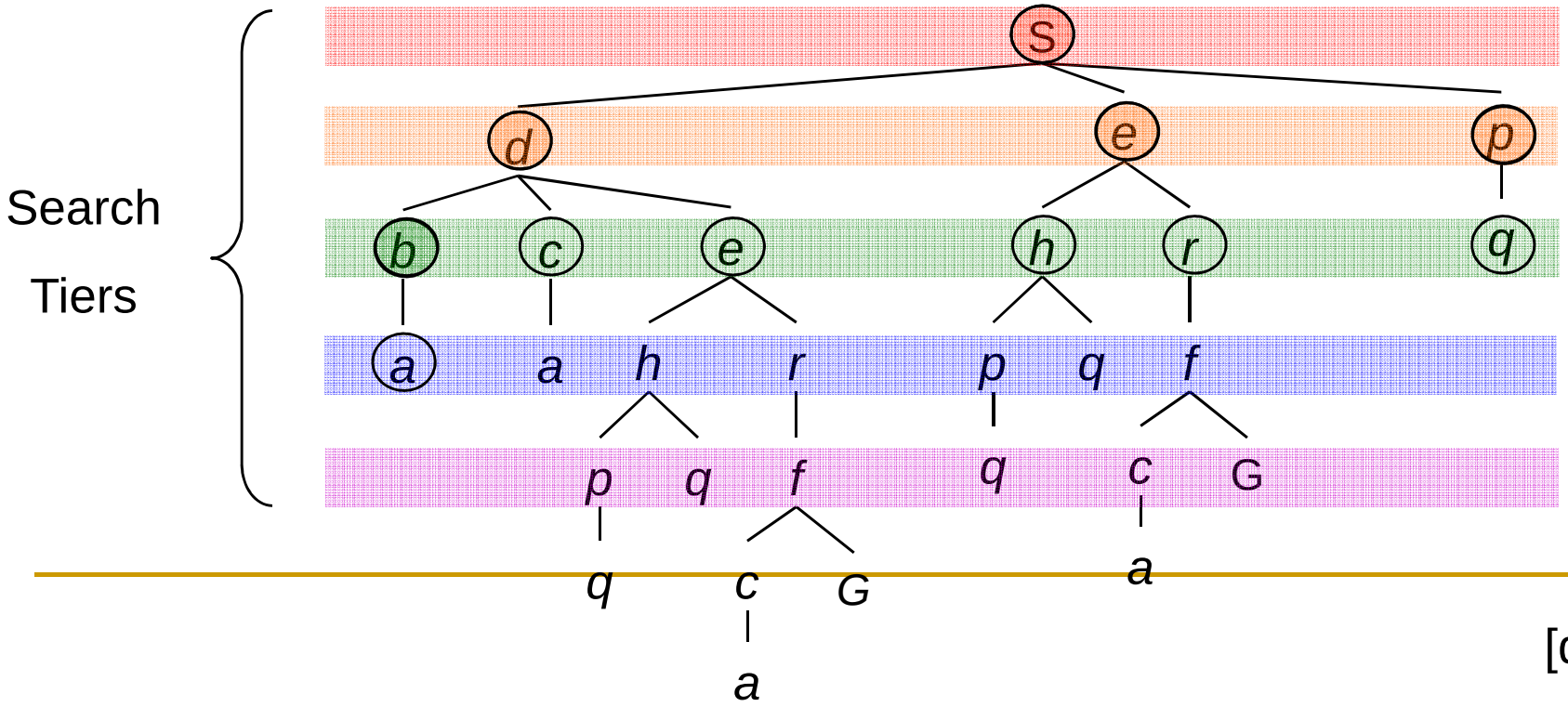
```
Procedure Breath_first_search;
BEGIN
     Open :=[start]; Close:=[ ];
    WHILE (Open <>[ ]) do
    BEGIN
            remove X which is the leftmost of Open;
            IF (X=goal) THEN return (Success)
            ELSE BEGIN
                generate children of X; Put X to Close;
                remove children of X which is in Open or Close;
                Put remain children on RIGHT end of Open;
            END;
    END;
    Return (FAIL);
END;
```

Breadth First Search

Strategy: expand shallowest node first

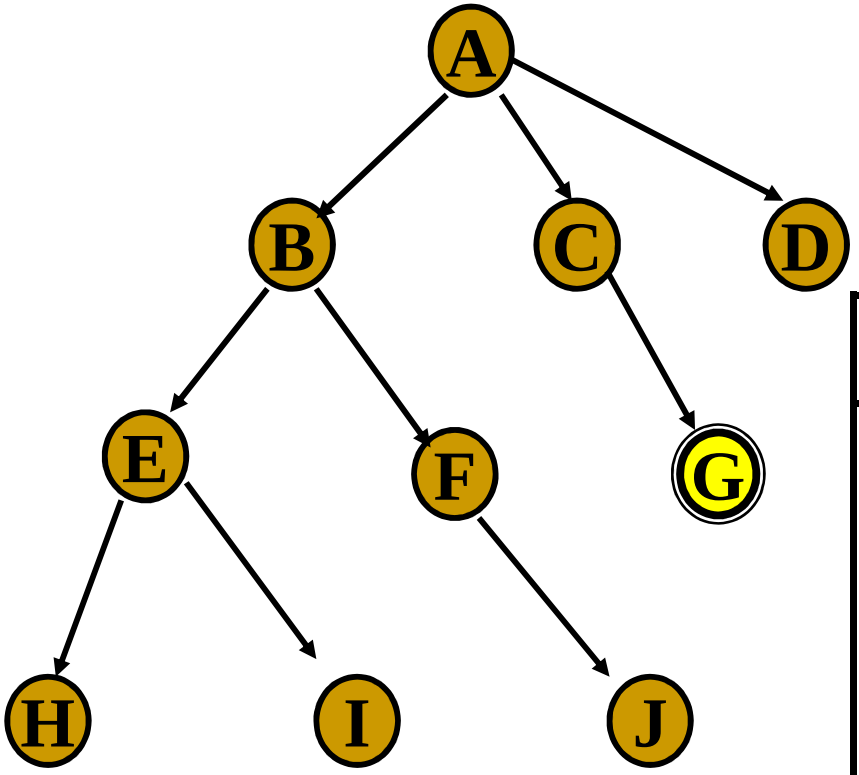
Implementation: Fringe is a FIFO queue





[demo: bfs]

Breadth-First-Search (tt)



Lần lặp	X	Open	Close
0		[A]	[ ]
1	A	[B C D]	[A]
2	B	[C D E F]	[A B]
3	C	[D E F G]	[A B C]
4	D	[E F G]	[A B C D]
5	E	[F G H I]	[A B C D E]
6	F	[G H I J]	[A B C D E F]
7	**G**	[H I J]	[A B C D E F]

Depth-First-Search

- Nếu TT đầu là đích $\Rightarrow$ dừng, trả về success
- Ngược lại: Lặp đến khi gặp succes hay gặp error
  - Phát sinh TT con của TT bắt đầu, gọi là E. Nếu không có con nào thì báo error
  - Gọi DFS với E như TT bắt đầu
  - Nếu success được trả về thì trả về success. Ngược lại: tiếp tục lặp

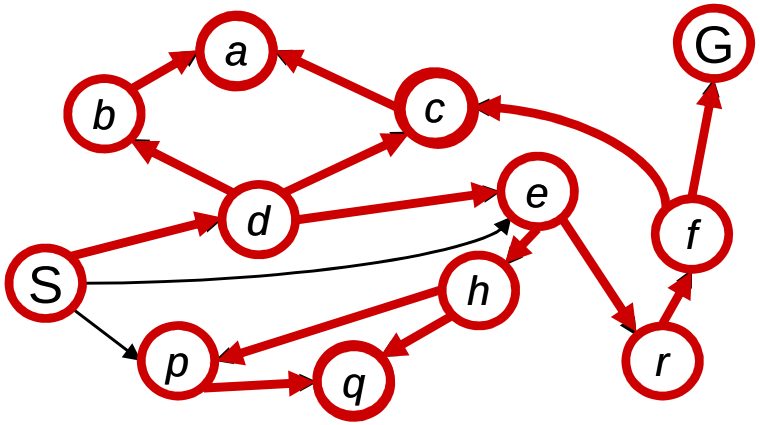
Depth-First-Search (tt)

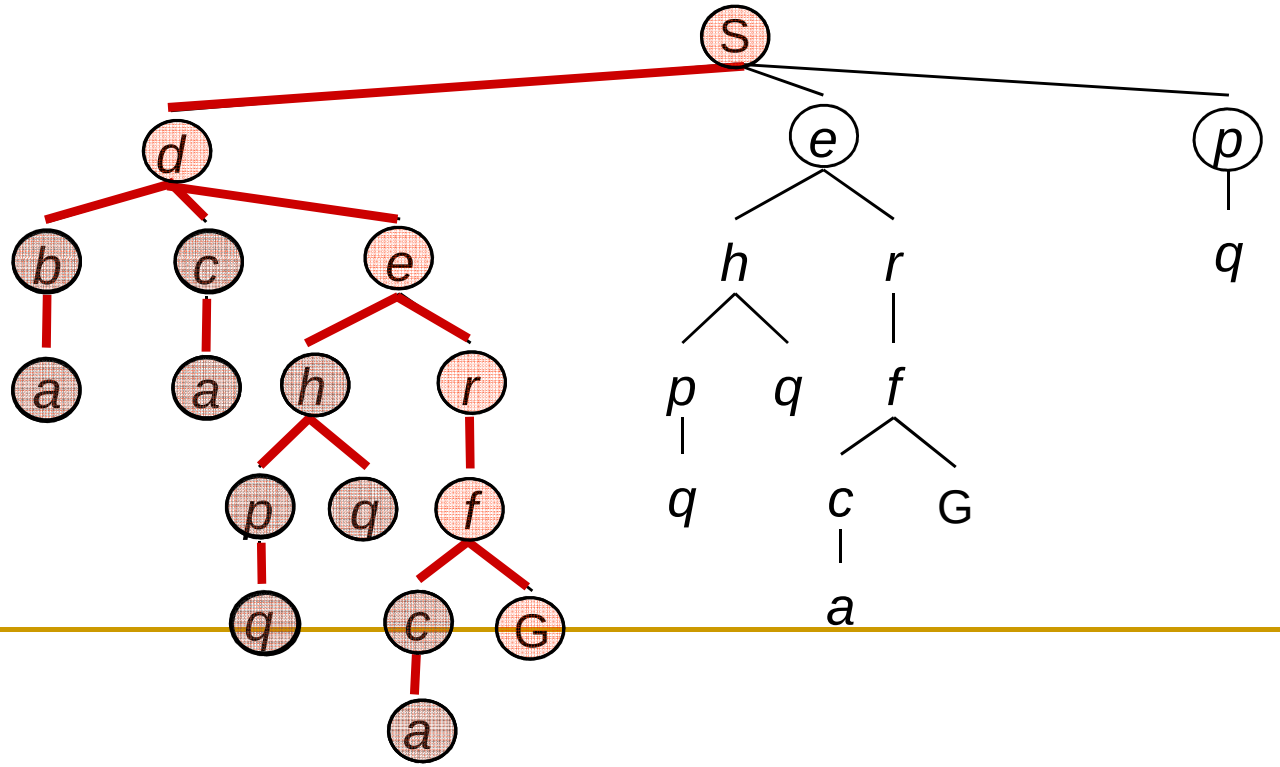
```
Procedure depth_first_search;
Begin
    Open :=[start]; Close:=[ ];
    While (Open <>[ ]) do
    begin
            remove X which is the leftmost of Open;
            If (X=goal) the return (Success)
            else begin
                generate children of X; Put X to Close;
                remove children of X which is in Open or Close;
                Put remain children on LEFT end of Open;
            End;
    End;
    Return (FAIL);
End;
```

Depth First Search

Strategy: expand deepest node first

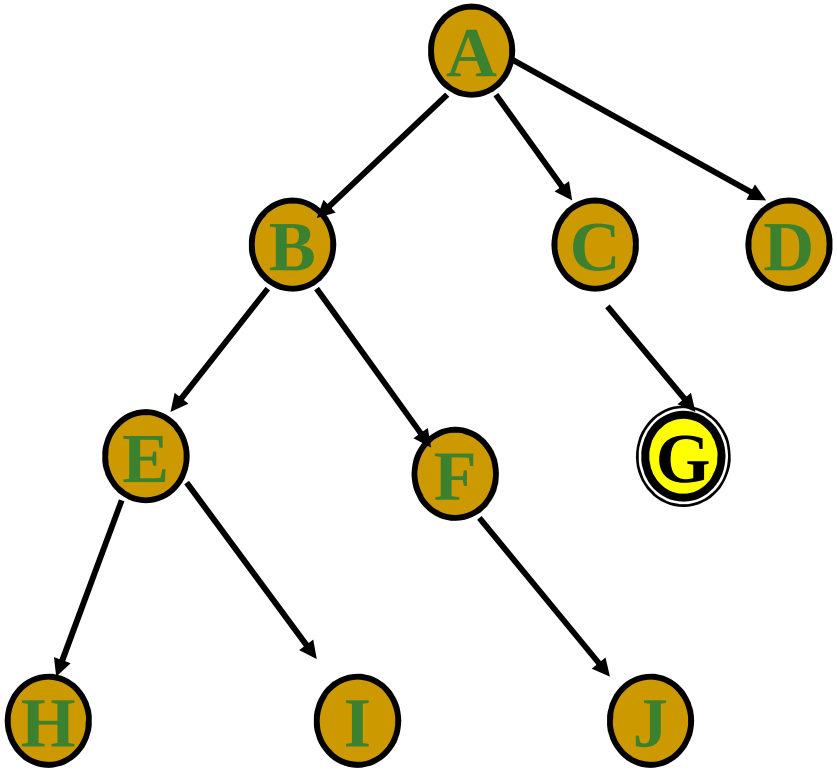
Implementation: Frontier is a LIFO stack





[demo: dfs]

Depth-First-Search (tt)



Lần lặp	X	Open	Close
0		[A]	[ ]
1	A	[B C D]	[A]
2	B	[E F C D]	[A B]
3	E	[H I F C D]	[A B E]
4	H	[I F C D]	[A B E H]
5	I	[F C D]	[A B E H I]
6	F	[J C D]	[A B E H I F]
7	J	[C D]	[A B E H I F J]
8	C	[G D]	[A B E H I F J C]
9	**G**		

Breath First vs Depth First

- Breath First: open được tổ chức dạng FIFO (Queue)
- Depth First: open được tổ chức dạng LIFO (Stack)
- Đặc tính
  - Breath First search hiệu quả khi lời giải nằm gần gốc của cây tìm kiếm, tìm nhiều lời giải, luôn tìm ra nghiệm có số cung nhỏ nhất
  - Depth First search hiệu quả khi lời giải nằm sâu trong cây tìm kiếm và có một phương án chọn hướng đi chính xác
- Kết quả
  - Breath First search chắc chắn tìm ra kết quả nếu có
  - Depth First có thể sa lầy vào đường quá dài
- Bùng nổ tổ hợp là khó khăn lớn nhất cho các giải thuật này

Depth first search có giới hạn

- Depth first search có khả năng lặp vô tận do các trạng thái con sinh ra liên tục. Độ sâu tăng vô tận
- Khắc phục bằng cách giới hạn độ sâu của giải thuật
- Sâu bao nhiêu thì vừa?
- Chiến lược giới hạn:
  - Cố định một độ sâu MAX, như các danh thủ chơi cờ tính trước được số nước nhất định
  - Theo cấu hình resource của máy tính
  - Meta knowledge trong việc định giới hạn độ sâu
- Giới hạn độ sâu $\Rightarrow$ co hẹp không gian trạng thái $\Rightarrow$ có thể mất nghiệm

Các đặc trưng của bài toán

- Một số yếu tố cần phân tích khi chọn kỹ thuật giải BT:
  - Khả năng phân rã bài toán
  - Khả năng lờ đi và quay lui
  - Khả năng dự đoán toàn cục
  - Đích là một trạng thái hay con đường (tập các TT)
  - Lượng tri thức cần để giải bài toán
  - Có cần sự can thiệp của con người trong quá trình giải không?

Các đặc trưng của bài toán (tt)

- Khả năng phân rã bài toán
  - Phân rã được: như BT tính tích phân ký hiệu
    - Giải bằng cách
      - Chia nhỏ BT lớn thành các BT con độc lập
      - Giải từng BT nhỏ
      - Kết hợp thành BT lớn
  - Không phân rã được: BT thế giới các khối (??)

Các đặc trưng của bài toán (tt)

- Các bước giải có thể lờ đi hay quay lui
  - Có thể lờ đi : như BT chứng minh định lý
    - Vì: định lý vẫn đúng sau một vài bước áp dụng các luật
  - Có thể quay lui: như BT 8-puzzle
    - Vì: có thể di chuyển theo hướng ngược lại để về TT trước
  - Không thể quay lui: như BT chơi cờ
    - Vì: game over!

Các đặc trưng của bài toán (tt)

- Các bước giải có thể lờ đi hay quay lui:
  - Có thể lờ đi :
    - Có thể áp dụng chiến lược điều khiển đơn giản không cần quay lui
    - Dể dàng hiện thực.
  - Có thể quay lui:
    - Chiến lược phức tạp hơn để quay lui được tại những bước lỗi.
    - Có thể dùng Push-Down Stack.
  - Không thể quay lui:
    - Dùng các chiến lược phức tạp hơn vì mỗi khi ra quyết định thì đó là quyết định cuối cùng.
    - Có thể dùng giải pháp Planning.
    - Sẽ được xem xét trong các chương sau.

Các đặc trưng của bài toán (tt)

- Khả năng dự đoán của bài toán:
  - Có thể dự đoán được: như BT 8 puzzle

    $\Rightarrow$ có thể đề ra 1 chuỗi các nước đi và tự tin vào kết qua sẽ xãy ra

    $\Rightarrow$ Có thể quay lui được
  - Không thể dự đoán được: như các game có đối kháng
    - Cần theo đuổi nhiều kế hoạch
    - Có chiến lược/đánh giá để chọn kế hoạch tốt

Các đặc trưng của bài toán (tt)

- Lời giải là tuyệt đối hay tương đối
  - Tuyệt đối (best-path): như bài toán TSP
    - Tính toán khó hơn (tổng quát)
    - Cần giải thuật tìm kiếm toàn diện hơn
  - Tương đối (any-path): như bài toán suy luận đời thường (xem sau)
    - Có thể dùng heuristic để giải trong thời gian hợp lý

Các đặc trưng của bài toán (tt)

- Lời giải là trạng thái hay con đường (tập các TT)
  - Trạng thái: như bài toán tìm ra cách hiểu phù hợp cho câu. Ví dụ:
    - "The bank president ate a dish of pasta salad with the fork."
    - Từng từ như: bank, president, … có thể được hiểu theo nhiều cách
    - Một kiểu tìm kiếm nào đó được thực hiện để tìm ra cách hiểu toàn bộ cho câu
  - Con đường
  - Song, điều này cũng tương đối. Vì có thể biểu diễn trạng thái để nó có thể bao gồm thông tin về một phần hay toàn bộ con đường

Các đặc trưng của bài toán (tt)

- Vai trò của tri thức là gì?
  - Cần ít tri thức:
    - Như bài toán: “chơi cờ”
    - Tri thức ~ luật để di chuyển hợp lệ, cơ chế điều khiển, chiến lược điều khiển để tăng tốc tìm kiếm
  - Cần nhiều tri thức
    - Như bài toán: Hiểu câu chuyện trên tạp chí
    - Tri thức: nhiều, cả những cái đã ghi tường minh và cả những cái
    - không được ghi trong chính câu chuyện

Vấn đề trong thiết kế CT tìm kiếm

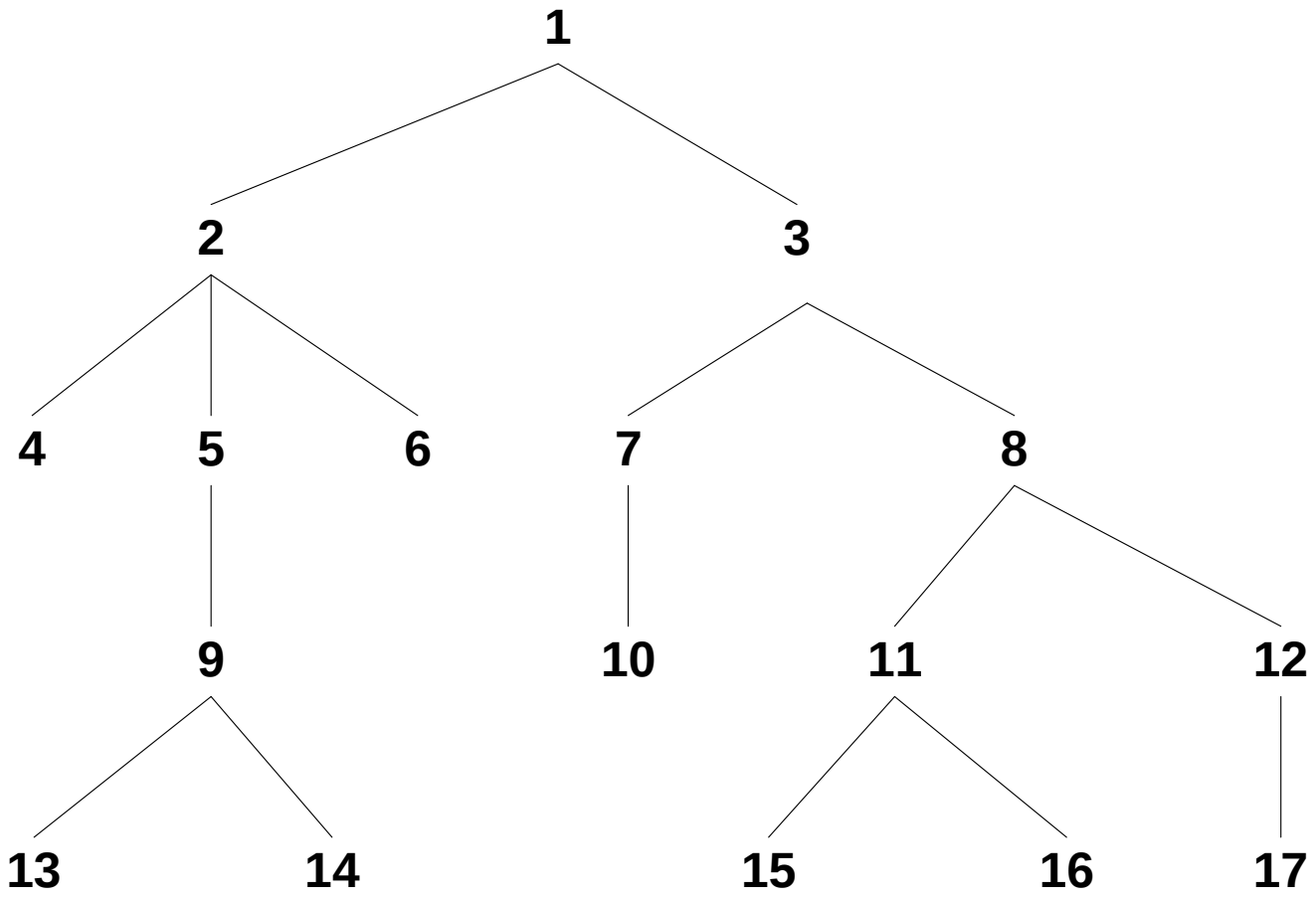
- Sự tìm kiếm
  - Tìm kiếm ~ duyệt cây, từ TT bắt đầu -> TT đích
  - Cả cây tìm kiếm thường không được xây dựng sẵn
  - Cấu trúc đồ thị thường thay thế cho cây trong biểu diễn KGTT
- Các vấn đề
  - Xác định hướng tìm (forward hay backward reasoning).
  - Cách lựa chọn luật để áp dụng (matching)
  - Cách biểu diễn nút (NODE) của quá trình tìm kiếm
  - Các NODE trong đồ thị có thể được phát sinh nhiều lần, và có thể đã được xem xét trước đó trong quá trình duyệt $\Rightarrow$ cần loại bỏ những NODE lặp lại $\Rightarrow$ Cần lưu lại các NODE đã xét.

Vấn đề trong thiết kế CT …

- Giải thuật kiểm tra NODE lặp lại (DFS):
  - Xem xét tập NODE đã tạo ra, để xem NODE mới đã có chưa
  - Nếu chưa thì thêm NODE mới vào đồ thị
  - Nếu đã có:
    - Thiết lập điểm mở rộng kế tiếp là con của NODE đang tồn tại , NODE có thể bỏ đi
    - Nếu GT có lưu giữ con đường tốt nhất hiện có thì cần xem xét xem nó đạt đến NODE mới trên con đường tốt hơn không, nếu vậy thì cập nhật lại con đường tốt nhất

BÀI TẬP 1

Xét đồ thị trạng thái sau đây, với mỗi chiến lược tìm kiếm bên dưới hãy liệt kê với danh sách thứ tự các nút được duyệt qua:



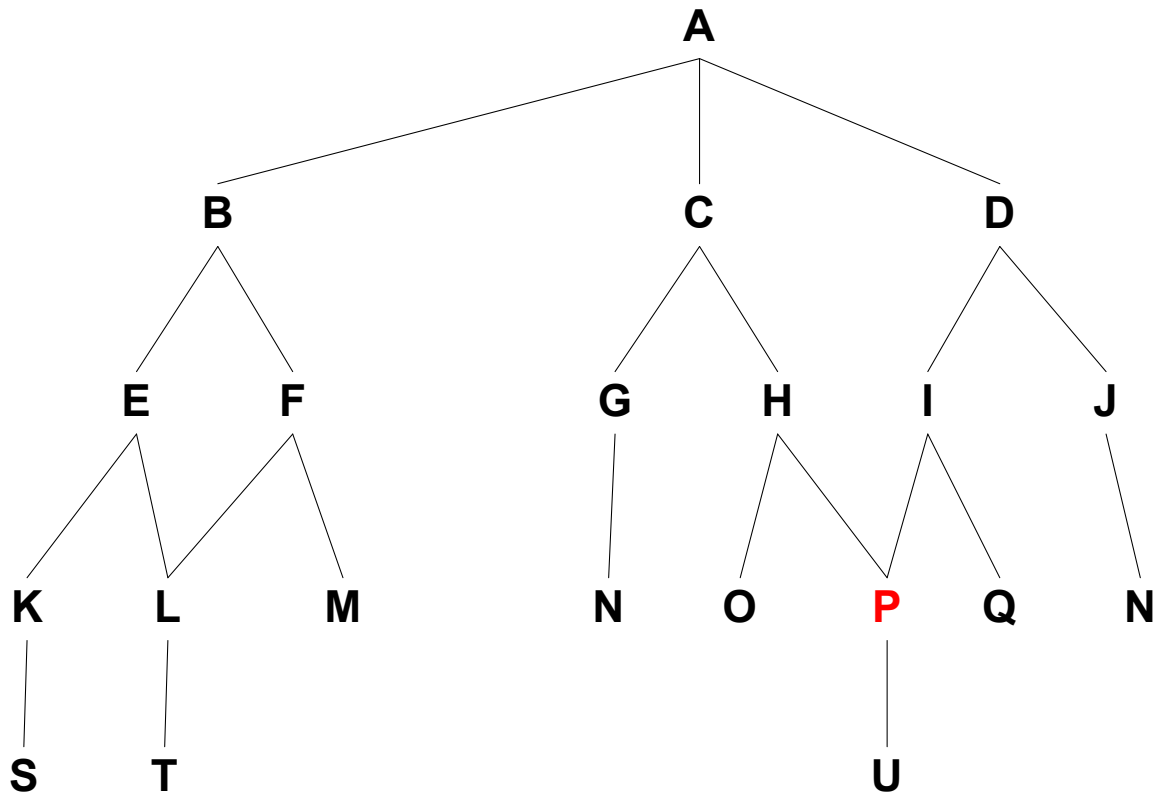
1/ Tìm kiếm rộng (BFS)

2/ Tìm kiếm sâu (DFS)

3/ Tìm kiếm sâu với độ sâu là 3

BÀI TẬP 2

Giả sử P là nút mục tiêu của đồ thị bên dưới. Hãy liệt kê danh sách thứ tự các nút duyệt qua ứng với từng chiến lược tìm kiếm.



1/ Tìm kiếm rộng (BFS)

2/ Tìm kiếm sâu (DFS)

3/ Tìm kiếm sâu với độ sâu là 3