NHẬP MÔN TRÍ TUỆ NHÂN TẠO

ThS Nguyễn Thị Trang
CNTT1
Học viện Công nghệ Bưu chính Viễn thông
Email: trangnguyen.hust117@gmail.com

GIẢI QUYẾT VẤN ĐỀ BẰNG TÌM KIẾM

Nhập môn trí tuệ nhân tạo

Nguyễn Thị Trang

Nội dung

- Bài toán tìm kiếm trong không gian trạng thái
- Một số ví dụ
- Các thuật toán tìm kiếm cơ bản

Tìm kiếm trong không gian trạng thái

- Tìm kiếm và khoa học trí tuệ nhân tạo
- Phát biểu bài toán tìm kiếm
- Các tiêu chuẩn đánh giá thuật toán tìm kiếm

Tìm kiếm & khoa học trí tuệ nhân tạo

- Nhiều bài toán có thể phát biểu và giải quyết dưới dạng tìm kiếm
  - Game: Tìm kiếm nước đi tối ưu (mang lại lợi thế)
  - Lập lịch: Tìm kiếm phương án sắp xếp thoả mãn yêu cầu đề ra (thoả mãn ràng buộc)
  - Tìm đường: Tìm đường đi tối ưu (chiều dài, thời gian, giá,..)
- Là hướng nghiên cứu quan trọng của trí tuệ nhân tạo
  - Phát triển các thuật toán tìm kiếm hiệu quả
  - Cơ sở cho nhiều nhánh nghiên cứu khác (ML, NLP, ..)

Giải quyết vấn đề bằng tìm kiếm

- Mục tiêu: Tìm các chuỗi hành động cho phép đạt đến các trạng thái mong muốn.
- Các bước chính:
  - Xác định **mục tiêu** cần đạt đến (goal formulation)
    - Là một tập hợp của các trạng thái (đích)
    - Dựa trên: trạng thái hiện tại (của môi trường) và đánh giá hiệu quả hành động (của tác tử)
  - Phát biểu **bài toán** (problem formulation)
    - Với một mục tiêu, xác định các *hành động* và *trạng thái* cần xem xét
  - Quá trình **tìm kiếm** (search process)
    - Xem xét các chuỗi hành động có thể
    - Chọn chuỗi hành động tốt nhất

Giải quyết vấn đề bằng tìm kiếm

- Giải thuật tìm kiếm
  - Đầu vào: Một bài toán cần giải quyết
  - Đầu ra: Một giải pháp, dưới dạng một chuỗi các hành động cần thực hiện

Phát biểu bài toán tìm kiếm

- Một bài toán tìm kiếm được phát biểu qua 5 thành phần sau
  1. Tập hữu hạn các trạng thái có thể Q
  2. Tập các trạng thái xuất phát: $S \subseteq Q$
  3. Hành động hay hàm nối tiếp hay toán tử P(x), là các trạng thái nhận được từ trạng thái x do kết quả thực hiện hành động hay toán tử
  4. Xác định đích:
     1. Tường minh, cho bởi tập đích G $\subseteq Q$
     2. Không tường minh, cho bởi một số điều kiện

Phát biểu bài toán tìm kiếm

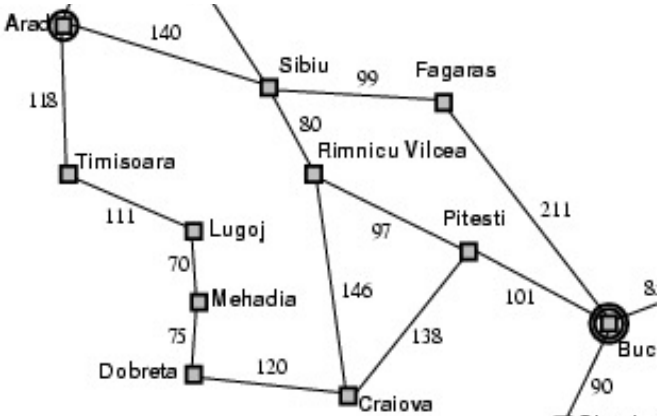
- Chi phí
  - Ví dụ: Tổng khoảng cách, số lượng hành động,…
  - $c(x, a, y) \geq 0$ là giá thành bước : từ trạng thái x, thực hiện hành động a và chuyển sang trạng thái y
- Lời giải là chuỗi hành động cho phép di chuyển từ trạng thái xuất phát tới trạng thái đích.

Các tiêu chuẩn đánh giá thuật toán tìm kiếm

- Độ phức tạp
  - Khối lượng tính toán cần thực hiện để tìm ra phương án
  - Số lượng trạng thái cần xem xét trước khi tìm lời giải
- Yêu cầu bộ nhớ
  - Số lượng trạng thái cần lưu trữ đồng thời trong bộ nhớ khi thực hiện thuật toán
- Tính đầy đủ
  - Nếu bài toán có lời giải thì thuật toán có khả năng tìm ra lời giải đó không?
- Tính tối ưu
  - Nếu bài toán có nhiều lời giải thì thuật toán có cho phép tìm ra lời giải tốt nhất hay không?

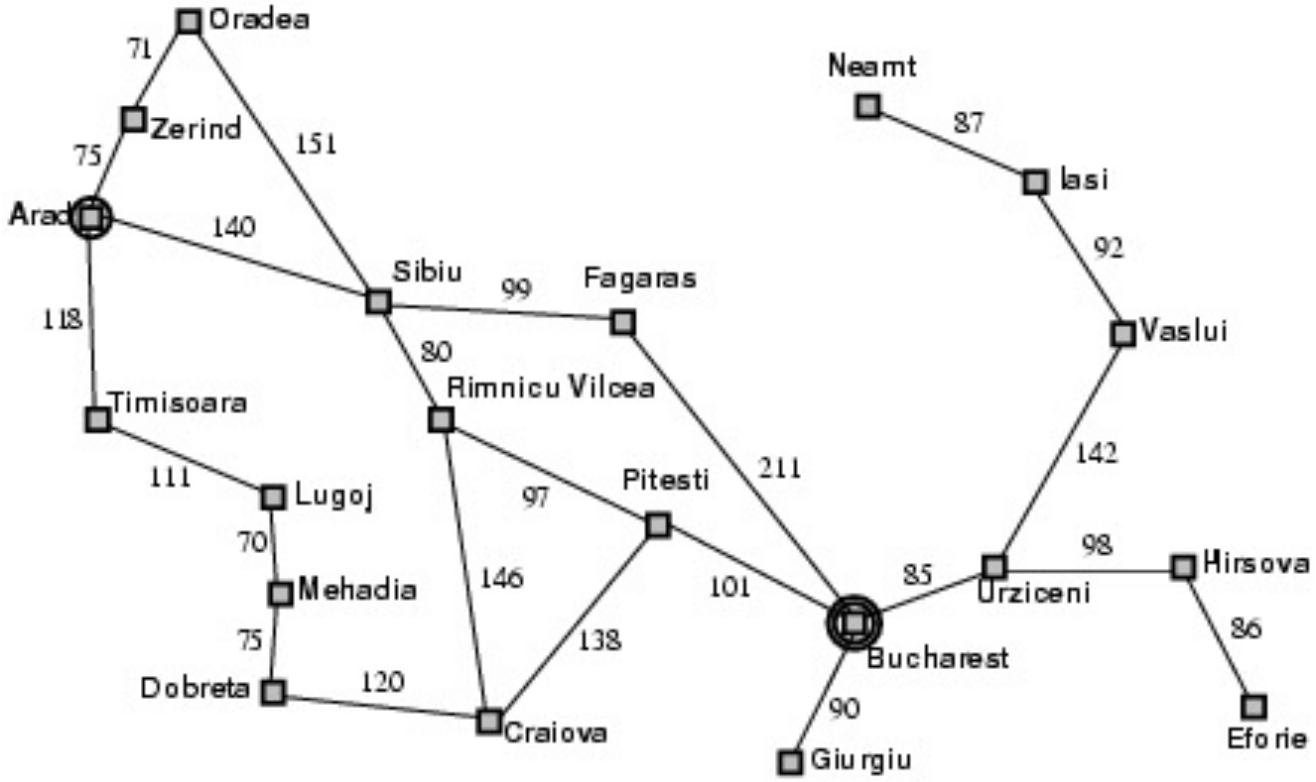
Ví dụ (1) - Người đi du lịch

- Một người du lịch đang trong chuyến đi du lịch ở Rumani
  - Anh ta hiện thời đang ở Arad
  - Ngày mai, anh ta có chuyến bay khởi hành từ Bucharest
  - Bây giờ, anh ta cần di chuyển (lái xe) từ Arad đến Bucharest
- Phát biểu **mục tiêu**:
  - Cần phải có mặt ở Bucharest
- Phát biểu **bài toán**:
  - Các *trạng thái*: các thành phố (đi qua)
  - Các *hành động*: lái xe giữa các thành phố

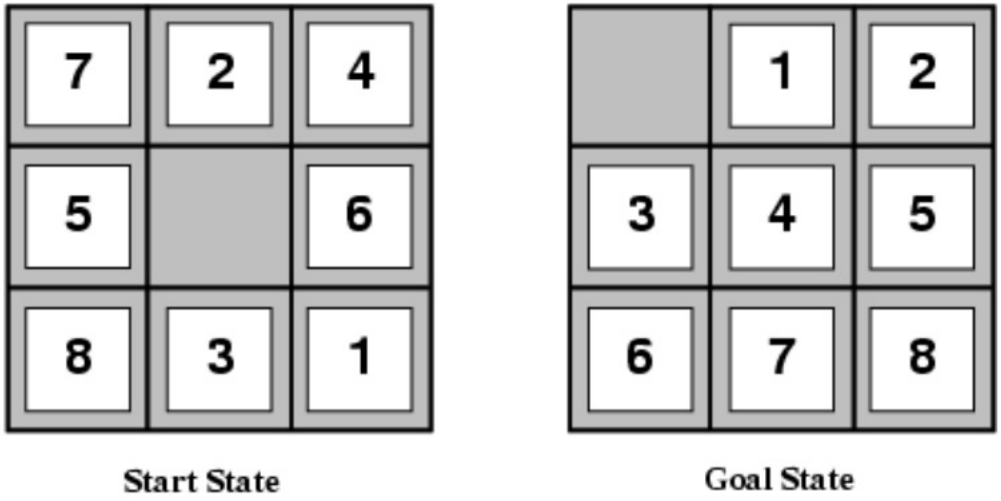


- **Tìm kiếm** giải pháp:
  - Chuỗi các thành phố cần đi qua, ví dụ: Arad, Sibiu, Fagaras, Bucharest

Ví dụ (1) - Người đi du lịch



Ví dụ (2) – Trò chơi 8 ô



(Russell & Norvig, 2010)

Ví dụ (2) – Trò chơi 8 ô

- Trạng thái: Tổ hợp vị trí của các ô
- Trạng thái xuất phát: Một trạng thái bất kỳ
- Hành động: Di chuyển ô trống sang trái, phải, lên, xuống
- Đích: So sánh với trạng thái đích (cho trước)
- Chi phí: Số lần di chuyển

Ví dụ (3) – Bài toán 8 con hậu

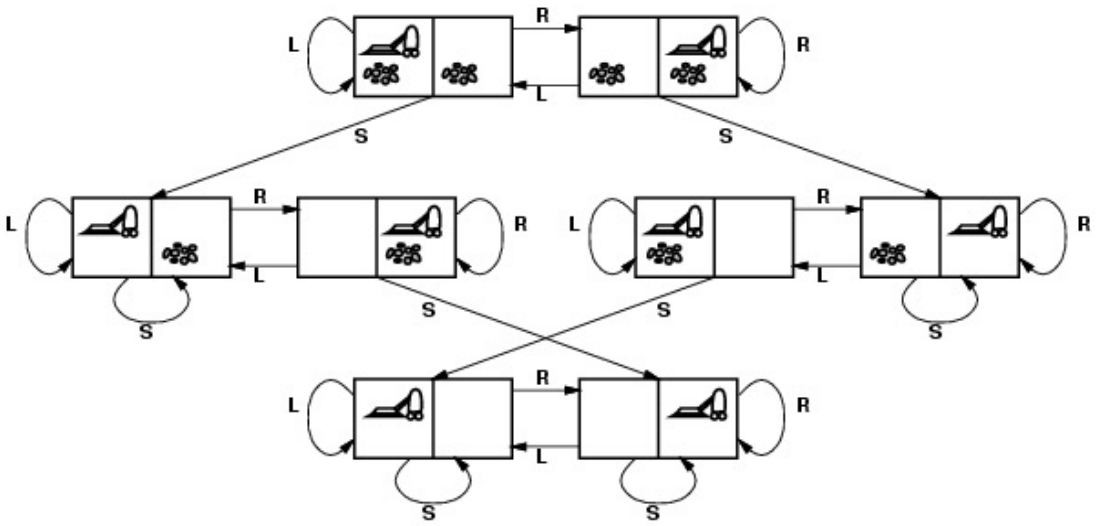
- Đặt 8 con hậu lên bàn cờ vua 8x8 sao cho không có đôi hậu nào đe doạ nhau
- Trạng thái: Tổ hợp vị trí của 0 đến 8 con hậu trên bàn cờ
- Trạng thái xuất phát: Không có con hậu nào trên bàn cờ
- Trạng thái đích: 8 con hậu trên bàn cờ, không có 2 con nào đe doạ nhau.

Xác định không gian trạng thái

- Các bài toán thực tế thường được mô tả phức tạp
  - → Không gian trạng thái cần được khái quát (abstracted) để phục vụ cho việc giải quyết bài toán
- Trạng thái (khái quát) = Một tập các trạng thái thực tế
- Hành động (khái quát) = Một kết hợp phức tạp của các hành động thực tế
  - Ví dụ: Hành động "Arad → Zerind" biểu diễn một tập kết hợp các đường, đường vòng, chỗ nghỉ, …
- Để đảm bảo việc thực hiện (quá trình tìm kiếm), bất kỳ trạng thái thực tế nào cũng phải có thể đạt đến được từ trạng thái thực tế khác
- Giải pháp (khái quát) = Một tập các đường đi giải pháp trong thực tế

Đồ thị không gian trạng thái (1)

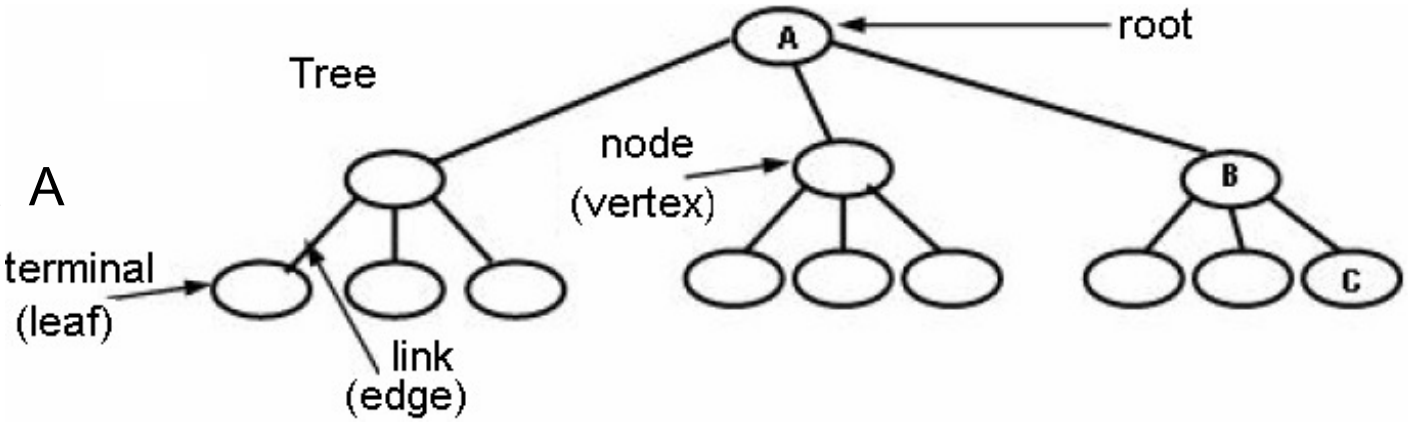
Bài toán máy hút bụi

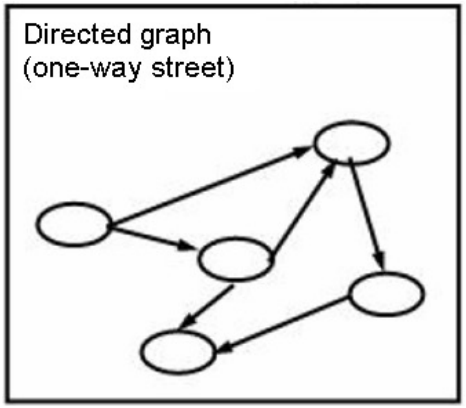


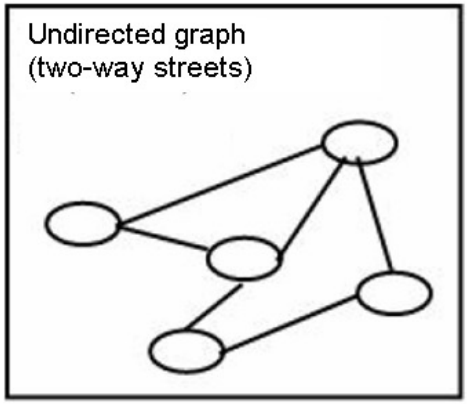
- Các trạng thái? *Chỗ bẩn và vị trí máy hút bụi*
- Các hành động? *Sang trái, sang phải, hút bụi, không làm gì*
- Kiểm tra mục tiêu? *Không còn chỗ (vị trí) nào bẩn*
- Chi phí đường đi? *1 (mỗi hành động), 0 (không làm gì cả)*

Biểu diễn bằng cây và đồ thị

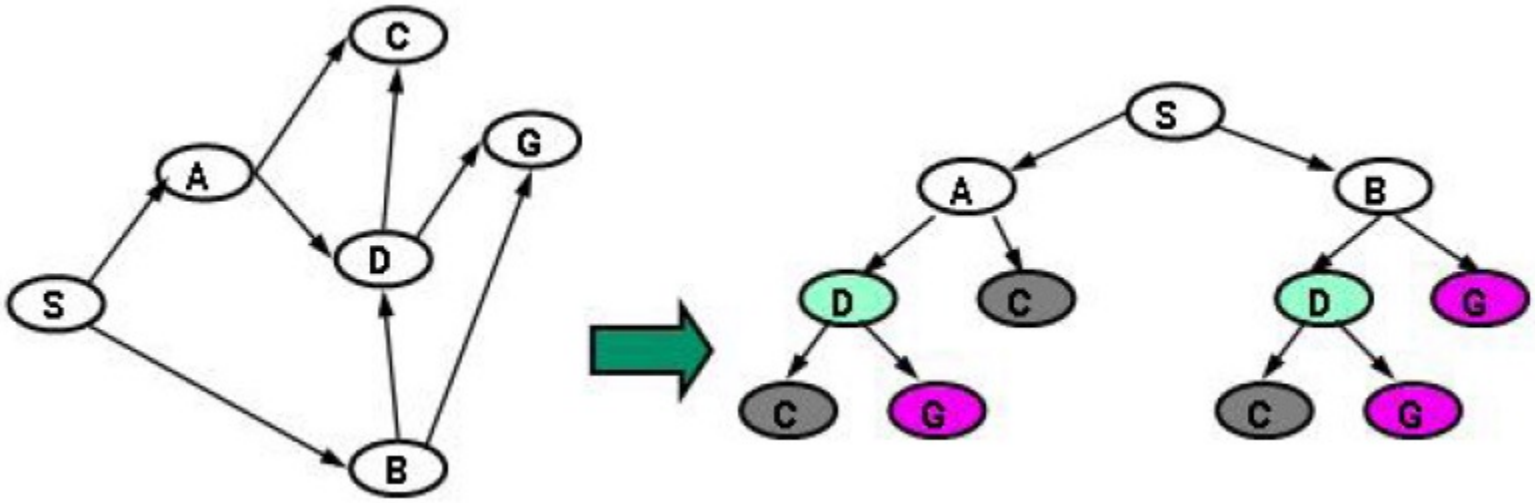
B là cha của C
C là con của B
A là tổ tiên của C
C là con cháu của A







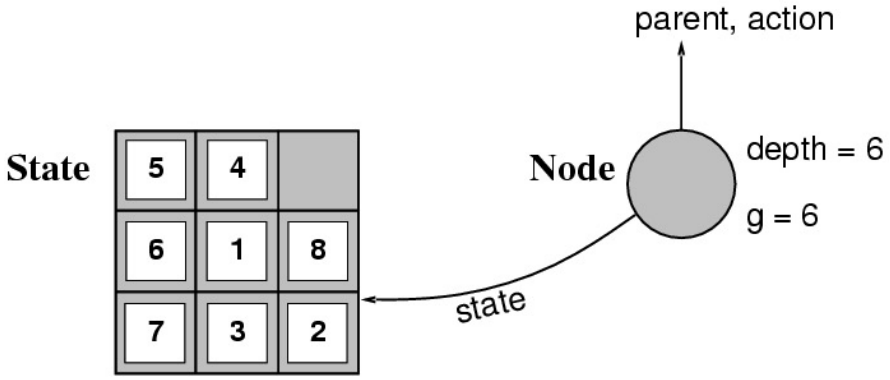
Đồ thị tìm kiếm → Cây tìm kiếm



- Các bài toán tìm kiếm trên đồ thị có thể được chuyển thành các bài toán tìm kiếm trên cây
  - Thay thế mỗi liên kết (cạnh) vô hướng bằng 2 liên kết (cạnh) có hướng
  - Loại bỏ các vòng lặp tồn tại trong đồ thị (để tránh không duyệt 2 lần đối với một nút trong bất kỳ đường đi nào)

Biểu diễn cây tìm kiếm

- Một *trạng thái* là một biểu diễn của một hình trạng (configuration) thực tế
- Một *nút* (của cây) là một phần cấu thành nên cấu trúc dữ liệu của một cây tìm kiếm
  - Một nút chứa các thuộc tính: *trạng thái*, *nút cha*, *nút con*, *hành động*, *độ sâu*, *chi phí đường đi g(x)*



- Hàm `Expand` tạo nên các nút mới,
  - Gán giá trị cho các thuộc tính (của nút mới)
  - Sử dụng hàm `Successor-Fn` để tạo nên các trạng thái tương ứng với các nút mới đó

Các chiến lược/thuật toán tìm kiếm cơ bản

- Thuật toán tìm kiếm tổng quát
- Tìm kiếm theo chiều rộng (Breadth-first search: BFS)
- Tìm kiếm với chi phí cực tiểu (Uniform-cost search: UCS)
- Tìm kiếm theo chiều sâu (Depth-first search: DFS)
- Tìm kiếm sâu dần (Iterative deepening search: IDS)

Thuật toán tìm kiếm tổng quát

- **Ý tưởng**: Xem xét các trạng thái, sử dụng các hàm trạng thái để mở rộng các trạng thái đó cho tới khi đạt đến trạng thái mong muốn.
- Mở rộng các trạng thái sẽ tạo ra ”cây tìm kiếm”
  - Mỗi trạng thái là 1 nút (node)
  - Các nút biên (nút mở) là nút đang chờ mở rộng tiếp
  - Nút đã mở rộng gọi là nút đóng.

Thuật toán tìm kiếm tổng quát

Search (Q, S, G, P)

(Q: không gian trạng thái, S: trạng thái bắt đầu, G: đích, P: hành động)

Đầu vào: Bài toán tìm kiếm

Đầu ra: Trạng thái đích

Khởi tạo: $O \leftarrow S$ (O: danh sách các nút mở)

while $(O \neq \emptyset)$ **do**

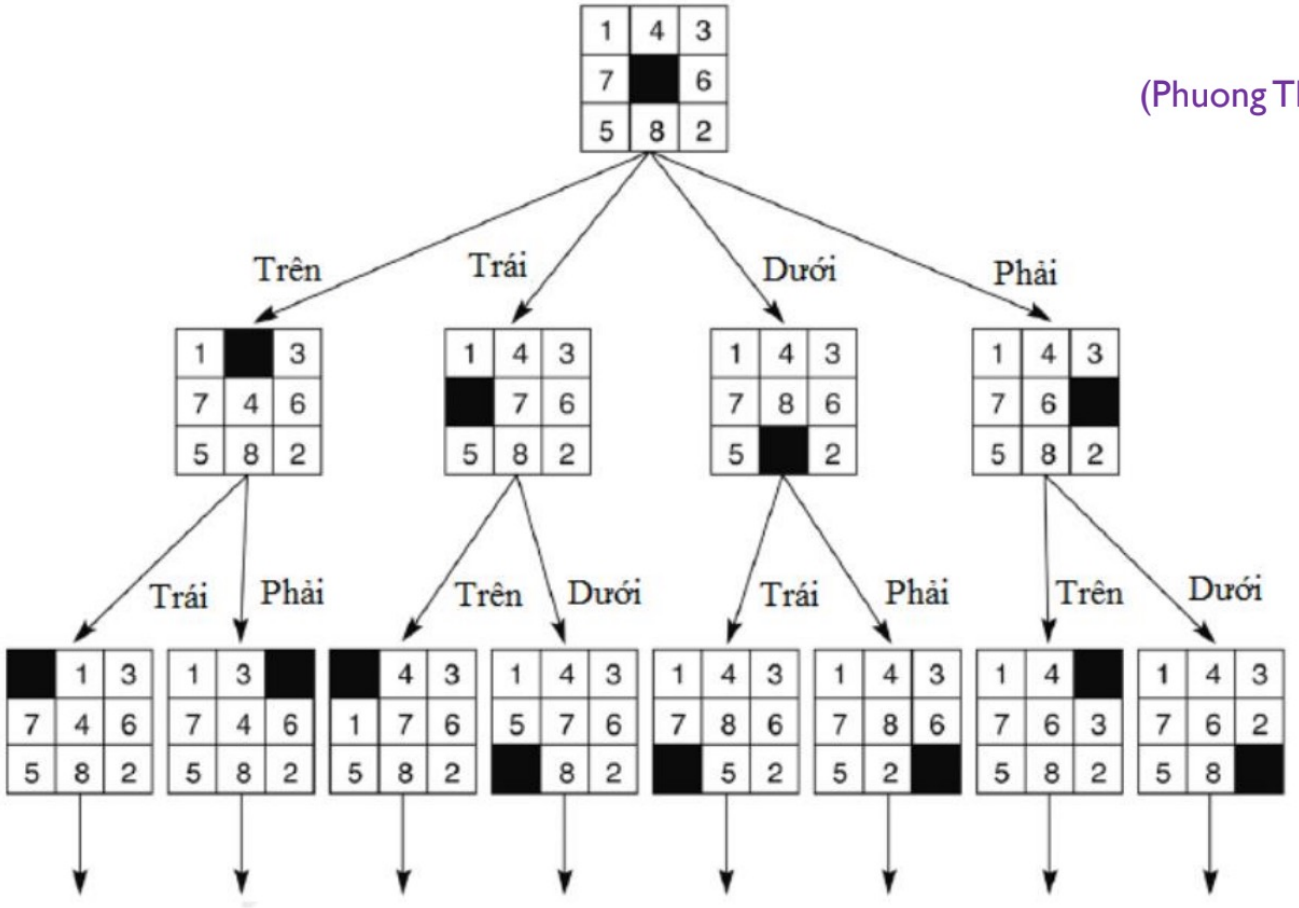
1. Chọn nút $n \in O$ và xoá n khỏi O
2. **if** $n \in G$, **return** (đường đi tới n)
3. Thêm P(n) vào O

return: Không tìm được lời giải

Thuật toán tìm kiếm tổng quát

➔ Chọn nút n thế nào ?

Ví dụ cây tìm kiếm (bài toán 8 ô)



Các chiến lược tìm kiếm

- Các chiến lược tìm kiếm xác định bởi **thứ tự mở rộng các nút** trên cây tìm kiếm.
- Đánh giá những tiêu chí:
  - Đầy đủ
  - Độ phức tạp tính toán
  - Yêu cầu bộ nhớ
  - Tối ưu
- Độ phức tạp tính theo các yếu tố sau:
  - b: Độ rẽ nhánh tối đa của cây tìm kiếm
  - d: Độ sâu của lời giải
  - m: Độ sâu tối đa của không gian trạng thái (có thể là vô cùng)

Tìm kiếm không có thông tin

- Tìm kiếm không có thông tin (tìm kiếm mù) chỉ sử dụng thông tin theo phát biểu của bài toán trong quá trình tìm kiếm
- Các phương pháp tìm kiếm mù:
  - Tìm kiếm theo chiều rộng
  - Tìm kiếm theo chi phí thấp nhất
  - Tìm kiếm theo chiều sâu
  - Tìm kiếm sâu dần

Tìm kiếm theo chiều rộng - BFS

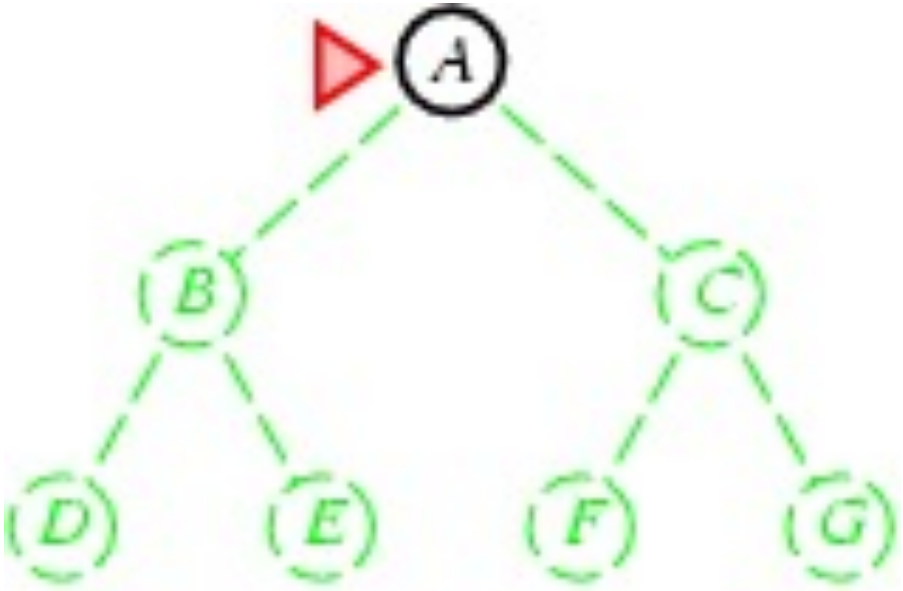
- Phát triển các nút chưa xét theo chiều rộng – Các nút được xét theo thứ tự độ sâu tăng dần
- Cài đặt giải thuật BFS
  - *fringe* là một cấu trúc kiểu hàng đợi FIFO (các nút mới được bổ sung vào cuối của *fringe*)
- Các ký hiệu được sử dụng trong giải thuật BFS
  - *fringe*: Cấu trúc kiểu hàng đợi (queue) lưu giữ các nút (trạng thái) ***sẽ*** được duyệt
  - *closed*: Cấu trúc kiểu hàng đợi (queue) lưu giữ các nút (trạng thái) ***đã*** được duyệt
  - $G=(N,A)$: Cây biểu diễn không gian trạng thái của bài toán
  - n_0: Trạng thái đầu của bài toán (nút gốc của cây)
  - *ĐICH*: Tập các trạng thái đích của bài toán
  - $\Gamma(n)$: Tập các trạng thái (nút) con của trạng thái (nút) đang xét n

BFS – Giải thuật

BFS (N, A, n_0, ĐICH)
{
  fringe ← n_0;
  closed ← ∅;
  while (fringe ≠ ∅) do
  { n ← GET_FIRST(fringe); // lấy phần tử đầu tiên của *fringe*
    closed ← closed ⊕ n;
    if (n ∈ ĐICH) then return SOLUTION(n);
    if (Γ(n) ≠ ∅) then fringe ← fringe ⊕ Γ(n);
  }
  return ("No solution");
}

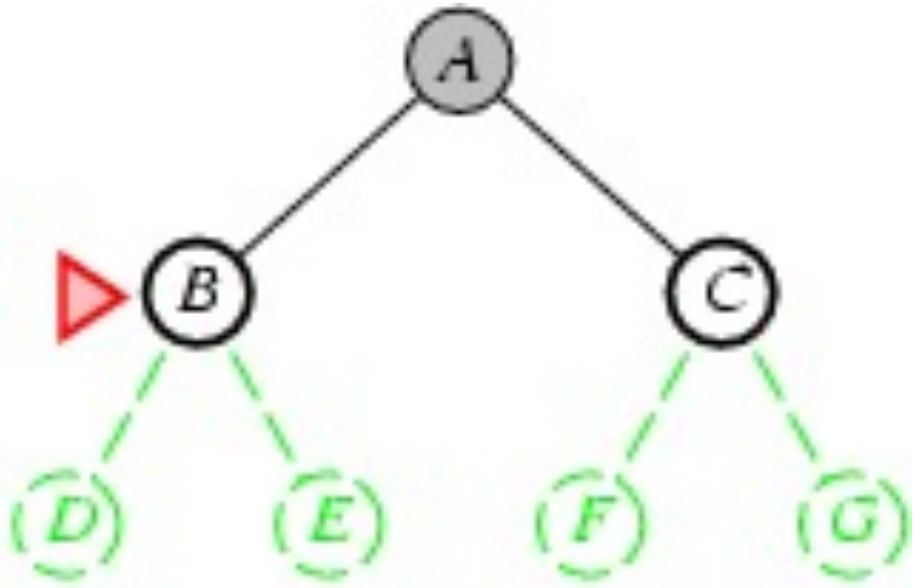
BFS – Ví dụ (1)

- Nguyên tắc: Phát triển các nút chưa xét theo chiều rộng – Các nút được xét theo thứ tự độ sâu tăng dần



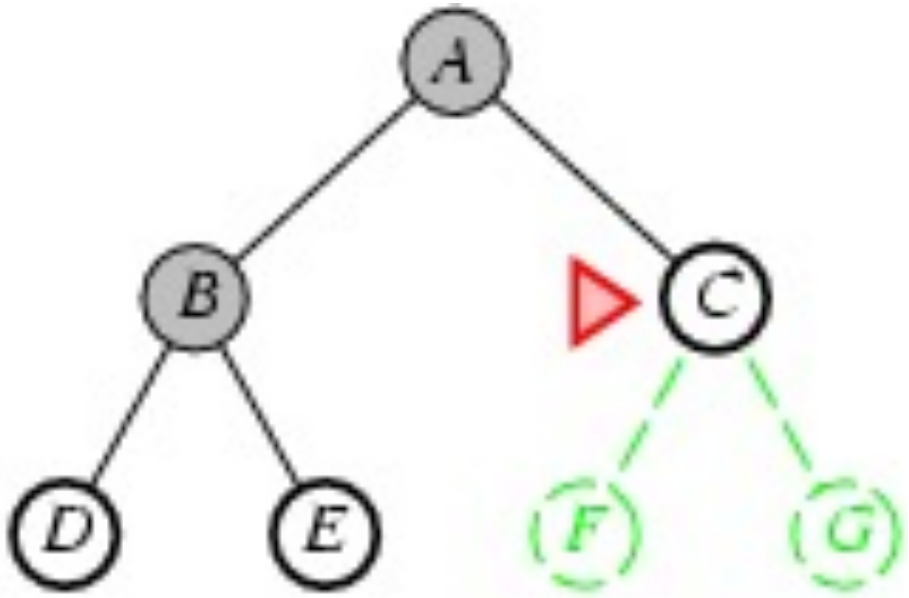
BFS – Ví dụ (2)

- Phát triển các nút chưa xét theo chiều rộng – Các nút được xét theo thứ tự độ sâu tăng dần



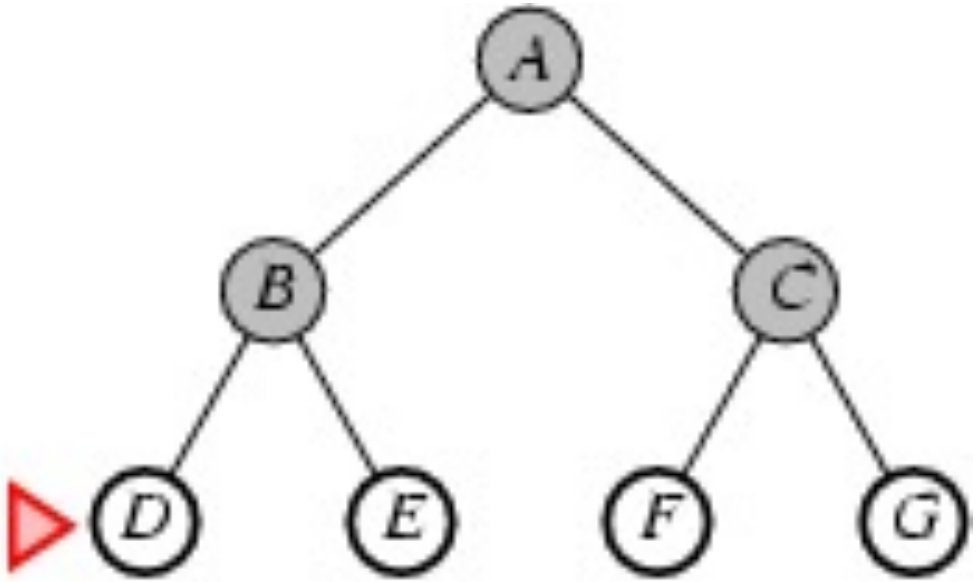
BFS – Ví dụ (3)

- Phát triển các nút chưa xét theo chiều rộng – Các nút được xét theo thứ tự độ sâu tăng dần



BFS – Ví dụ (4)

- Phát triển các nút chưa xét theo chiều rộng – Các nút được xét theo thứ tự độ sâu tăng dần

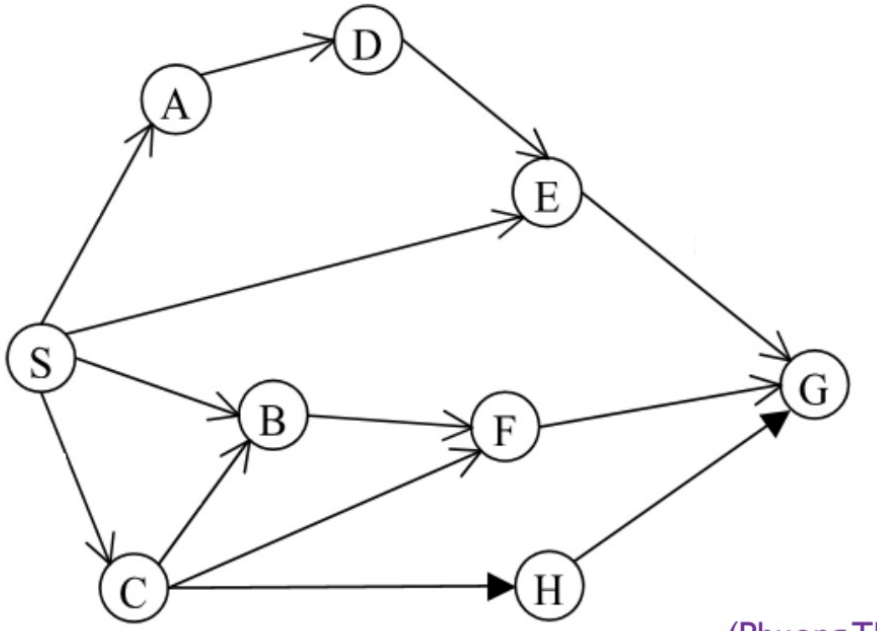


Tránh các nút lặp

- Có thể có nhiều đường đi cùng dẫn tới 1 nút
  - Thuật toán có thể mở rộng một nút nhiều lần
  - Có thể dẫn tới vòng lặp vô hạn
- Giải pháp
  - Không thêm nút vào hàng đợi nếu nút đã được duyệt hoặc đang nằm trong hàng đợi (chờ được duyệt)
    - Cần nhớ ít nút, thời gian kiểm tra nhanh
    - Tránh được vòng lặp vô hạn

Ví dụ 2- BFS

Tìm đường đi từ S tới G duyệt theo chiều rộng



(Phuong TM, 2016)

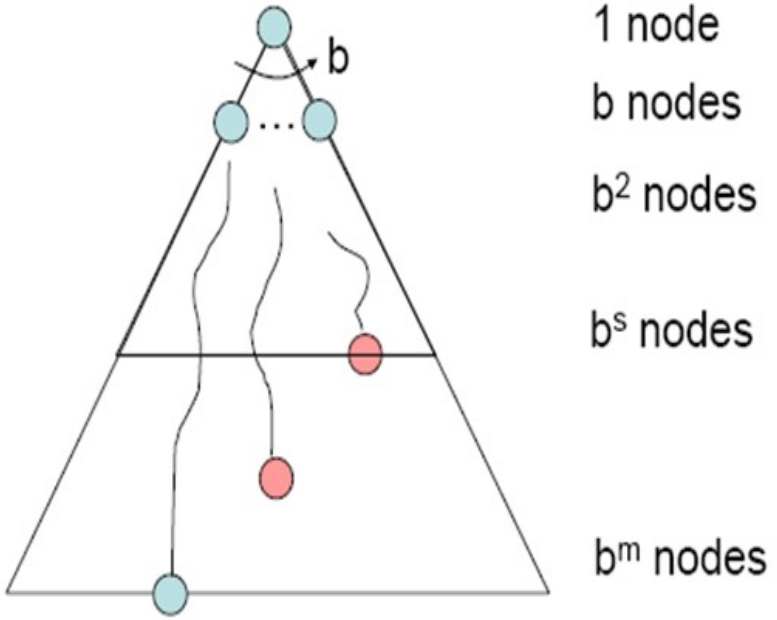
Ví dụ 2 - BFS

STT	Nút được mở rộng	Tập biên O (hàng đợi FIFO)
0		S
1	S	A_S, B_S, C_S, E_S
2	A_S	B_S, C_S, E_S, D_A
3	B_S	C_S, E_S, D_A, F_B
4	C_S	E_S, D_A, F_B, H_C
5	E_S	D_A, F_B, H_C, G_E
6	D_A	F_B, H_C, G_E
7	F_B	H_C, G_E
8	H_C	G_E
9	G_E	Đích

Đường đi: $G \leftarrow E \leftarrow S$

BFS – Tính chất

- Tính hoàn chỉnh?
  - Có (nếu *b* là hữu hạn)
- Độ phức tạp về thời gian?
  - $1+b+b^2+b^3+\ldots+b^d = O(b^{d+1})$
- Độ phức tạp về bộ nhớ?
  - $O(b^{d+1})$ – Lưu tất cả các nút trong bộ nhớ)
- Tính tối ưu?
  - Có (nếu chi phí =1 cho mỗi bước)



Tìm kiếm với chi phí cực tiểu (UCS)

- Trong trường hợp chi phí di chuyển giữa hai nút là không bằng nhau giữa các cặp nút.
  - BFS không cho lời giải tối ưu
  - Cần sử dụng phương pháp tìm kiếm theo chi phí thống nhất (là 1 biến thể của BFS)
- Phương pháp: Chọn nút có chi phí nhỏ nhất để mở rộng trước thay vì chọn nút nông nhất như trong BFS

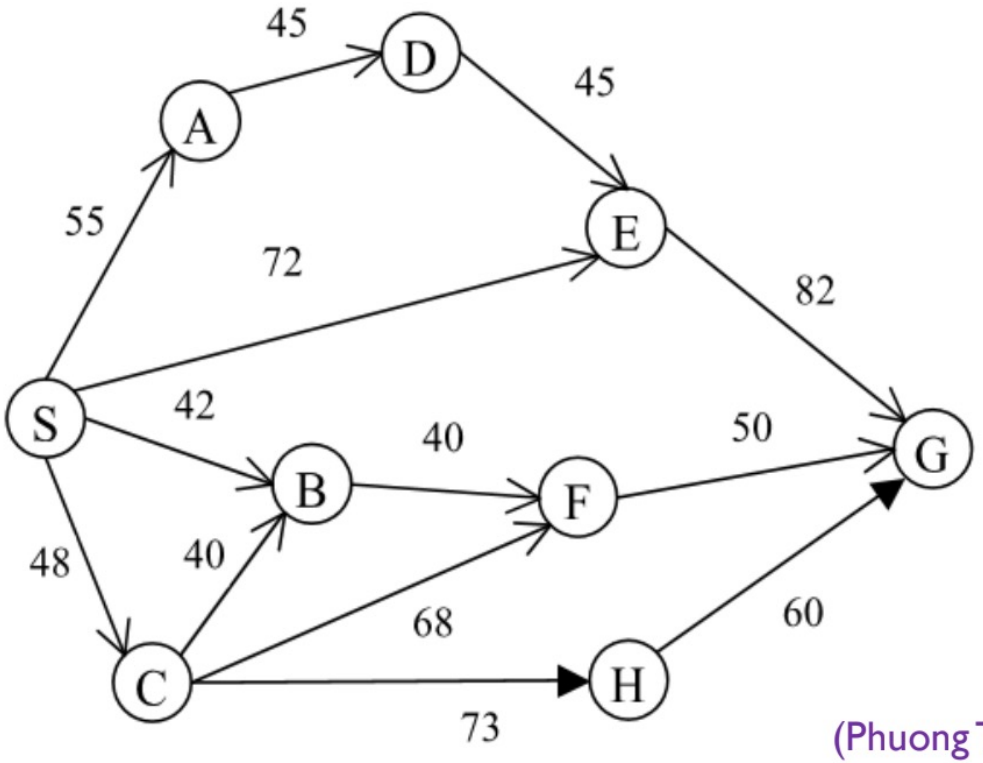
Tìm kiếm với chi phí cực tiểu (UCS)

- Phát triển các nút chưa xét có chi phí thấp nhất – Các nút được xét theo thứ tự chi phí (từ nút gốc đến nút đang xét) tăng dần
- Cài đặt:
  - *fringe* là một cấu trúc hàng đợi, trong đó các phần tử được sắp xếp theo chi phí đường đi
- Trở thành phương pháp tìm kiếm theo chiều rộng, nếu các chi phí ở mỗi bước (mỗi cạnh của cây tìm kiếm) là như nhau

UCS: Giải thuật

```
UCS (N, A, n0, ĐICH, c)
{
    fringe ← n0;
    closed ← ∅;
    while  (fringe ≠ ∅)  do
    {       n ← GET_LOWEST_COST(fringe);          // lấy phần tử có chi phí
                                                  // đường đi c(n) nhỏ nhất
            closed ← closed ⊕ n;
            if  (n ∈ ĐICH)  then  return  SOLUTION(n);
            if  (Γ(n) ≠ ∅) then  fringe ← fringe ⊕ Γ(n);
    }
    return  ("No solution");
}
```

UCS – Ví dụ



(Phuong TM, 2016)

UCS – Ví dụ

STT	Nút được mở rộng	Tập biên O
0		$S(0)$
1	S	$A_S(55), B_S(42), C_S(48), E_S(72)$
2	B_S	$A_S(55), C_S(48), E_S(72), F_B(82)$
3	C_S	$A_S(55), E_S(72), F_B(82), H_C(121)$
4	A_S	$E_S(72), F_B(82), H_C(121), D_A(100)$
5	E_S	$F_B(82), H_C(121), D_A(100), G_E(154)$
6	F_B	$H_C(121), D_A(100), G_F(132)$
7	D_A	$H_C(121), G_F(132)$
8	H_C	$G_F(132)$
9	G_F	Đích

Cập nhật đường đi tới G

Đường đi: $G \leftarrow F \leftarrow B \leftarrow S$

UCS – Đặc điểm

- Tính hoàn chỉnh?
  - Có (nếu chi phí ở mỗi bước $\geq \varepsilon > 0$)
- Độ phức tạp về thời gian?
  - Phụ thuộc vào tổng số các nút có chi phí ≤ chi phí của lời giải tối ưu: $O(b^{\lceil C^*/\varepsilon \rceil})$, trong đó C^* là chi phí của lời giải tối ưu
- Độ phức tạp về bộ nhớ?
  - Phụ thuộc vào tổng số các nút có chi phí ≤ chi phí của lời giải tối ưu: $O(b^{\lceil C^*/\varepsilon \rceil})$
- Tính tối ưu?
  - Có (nếu các nút được xét theo thứ tự tăng dần về chi phí $g(n)$)

Tìm kiếm theo chiều sâu - DFS

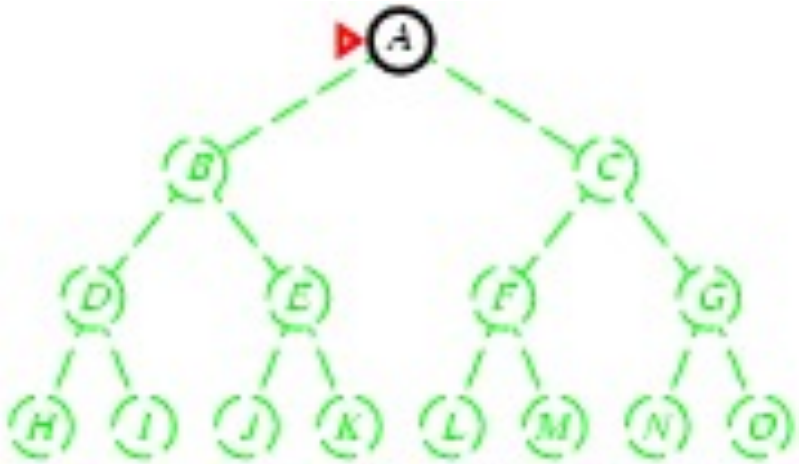
- Phát triển các nút chưa xét theo chiều sâu – Các nút được xét theo thứ tự độ sâu giảm dần
- Cài đặt:
  - *fringe* là một cấu trúc kiểu ngăn xếp LIFO (Các nút mới được bổ sung vào đầu của *fringe*)

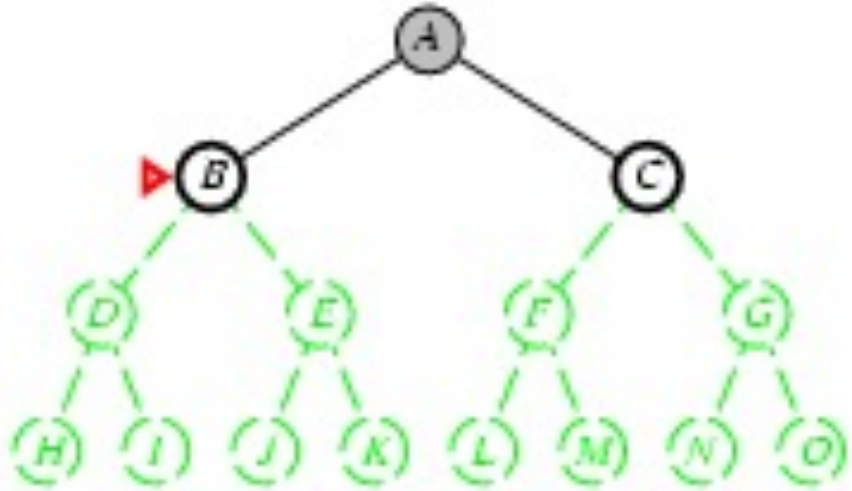
DFS – Giải thuật

```
DFS (N, A, n0, ĐICH)
{
    fringe ← n0;
    closed ← ∅;
    while  (fringe ≠ ∅)  do
    {       n ← GET_FIRST(fringe); // lấy phần tử đầu tiên của fringe
            closed ← closed ⊕ n;
            if  (n ∈ ĐICH)  then  return  SOLUTION(n);
            if  (Γ(n) ≠ ∅) then  fringe ← Γ(n) ⊕ fringe;
    }
    return  (“No solution”);
}
```

DFS – Ví dụ 1

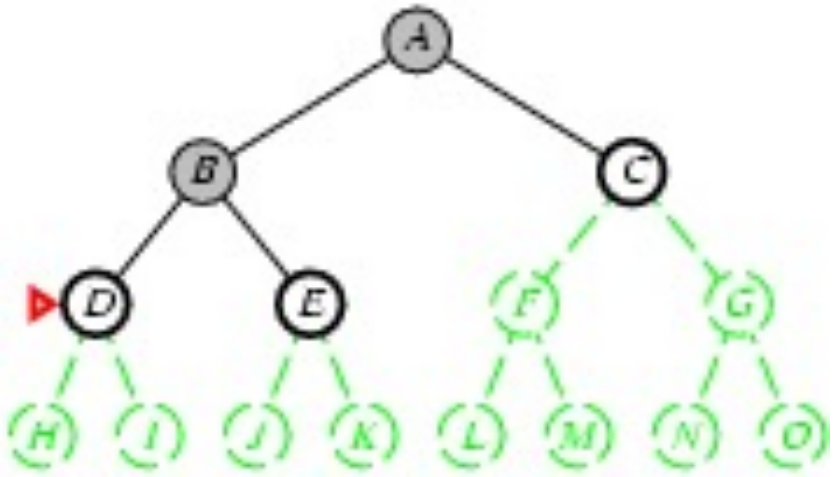
- Nguyên tắc: Trong số những nút biên, lựa chọn nút sâu nhất (xa gốc nhất) để mở rộng

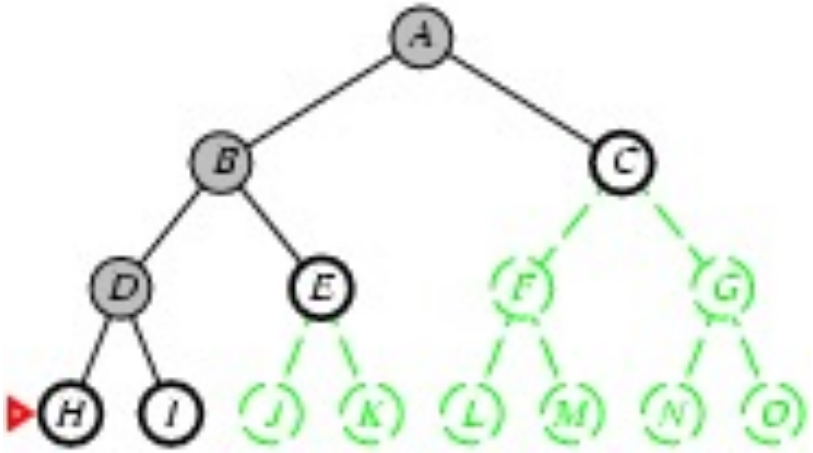




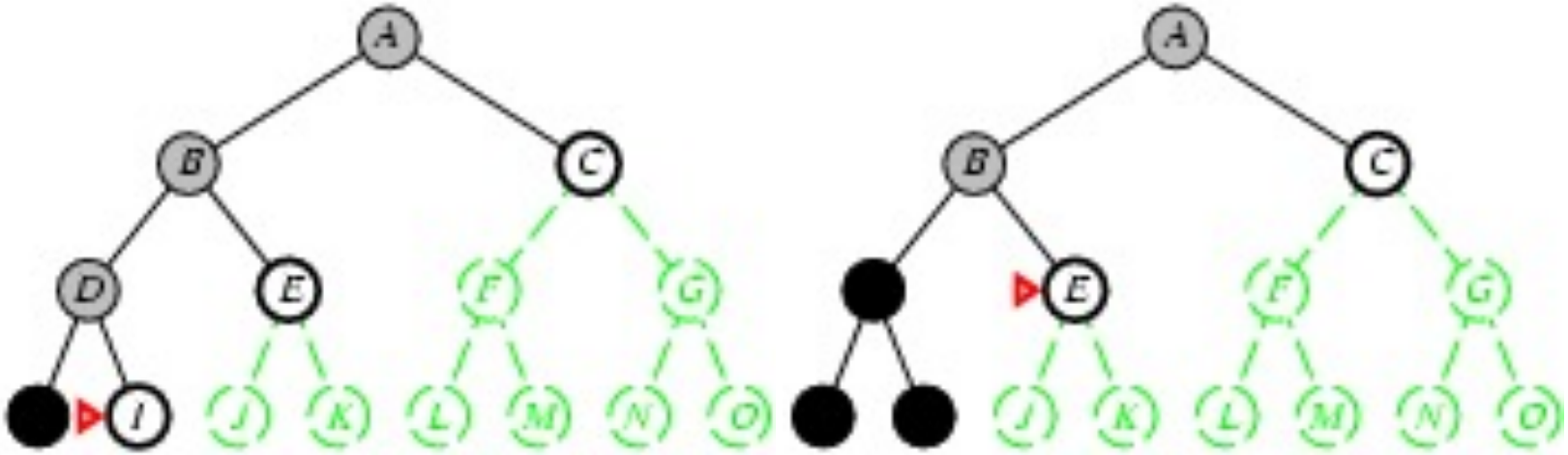
DFS – Ví dụ 1

- Nguyên tắc: Trong số những nút biên, lựa chọn nút sâu nhất (xa gốc nhất) để mở rộng

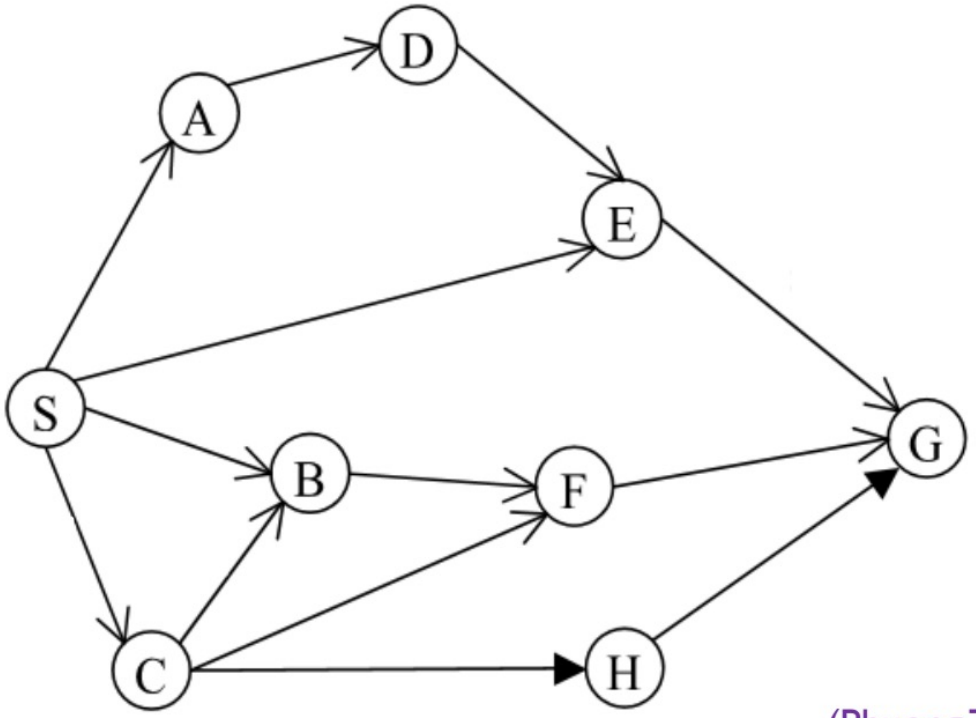




DFS – Ví dụ 1



DFS – Ví dụ 2



(Phuong TM, 2016)

DFS – Ví dụ 2

STT	Nút được mở rộng	Tập biên O (ngăn xếp LIFO)
0		S
1	S	A_S, B_S, C_S, E_S
2	A_S	D_A, B_S, C_S, E_S
3	D_A	E_D, B_S, C_S, E_S
4	E_D	G_E, B_S, C_S, E_S
5	G_E	Đích

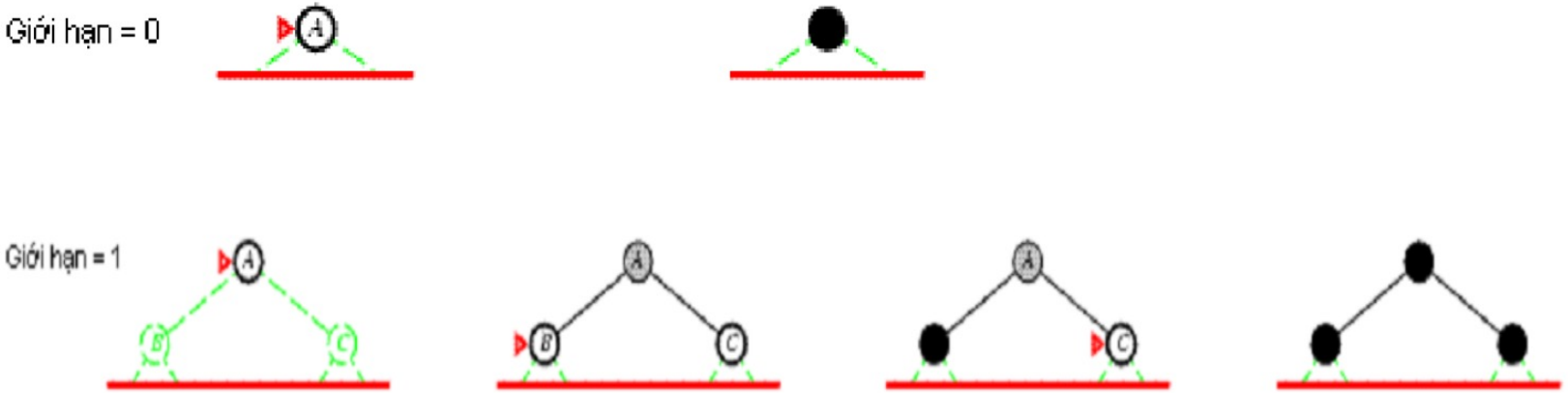
Đường đi: $G \leftarrow E \leftarrow D \leftarrow A \leftarrow S$
Độ sâu: 4

DFS – Tính chất

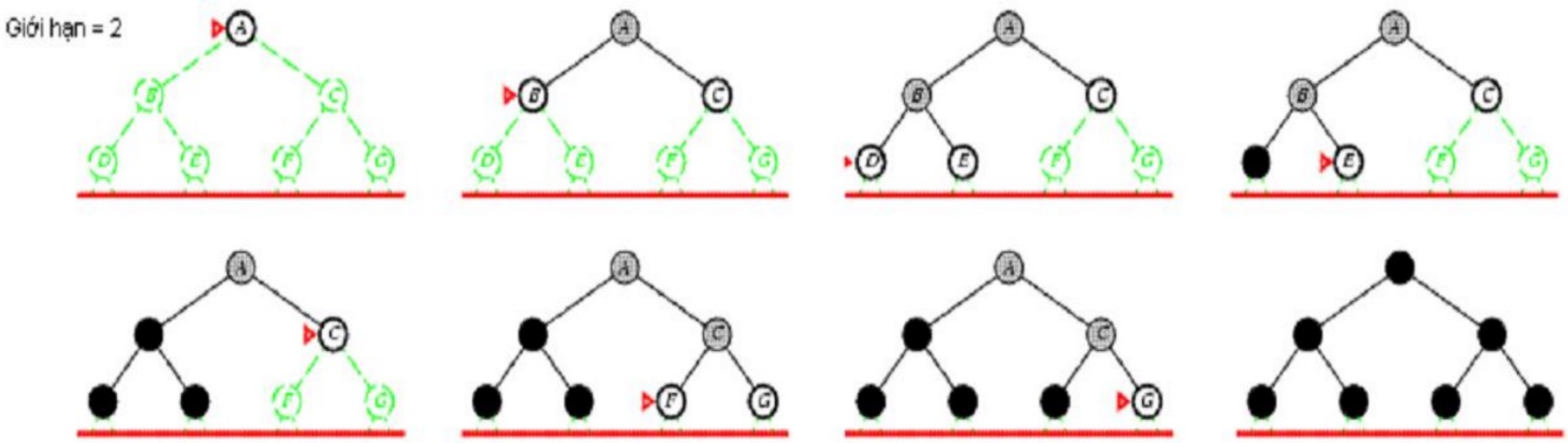
- Tính hoàn chỉnh?
  - Không – Thất bại (không tìm được lời giải) nếu không gian trạng thái có độ sâu vô hạn, hoặc nếu không gian trạng thái chứa các vòng lặp giữa các trạng thái
    - Đề cử: Sửa đổi để tránh việc một trạng thái nào đó bị lặp lại (bị xét lại) theo một đường đi tìm kiếm
    - → Đạt tính hoàn chỉnh đối với không gian trạng thái hữu hạn
- Độ phức tạp về thời gian?
  - $O(b^m)$: rất lớn, nếu m lớn hơn nhiều so với d
- Độ phức tạp về bộ nhớ?
  - $O(bm)$ - độ phức tạp tuyến tính
- Tính tối ưu?
  - Không

Tìm kiếm sâu dần - IDS

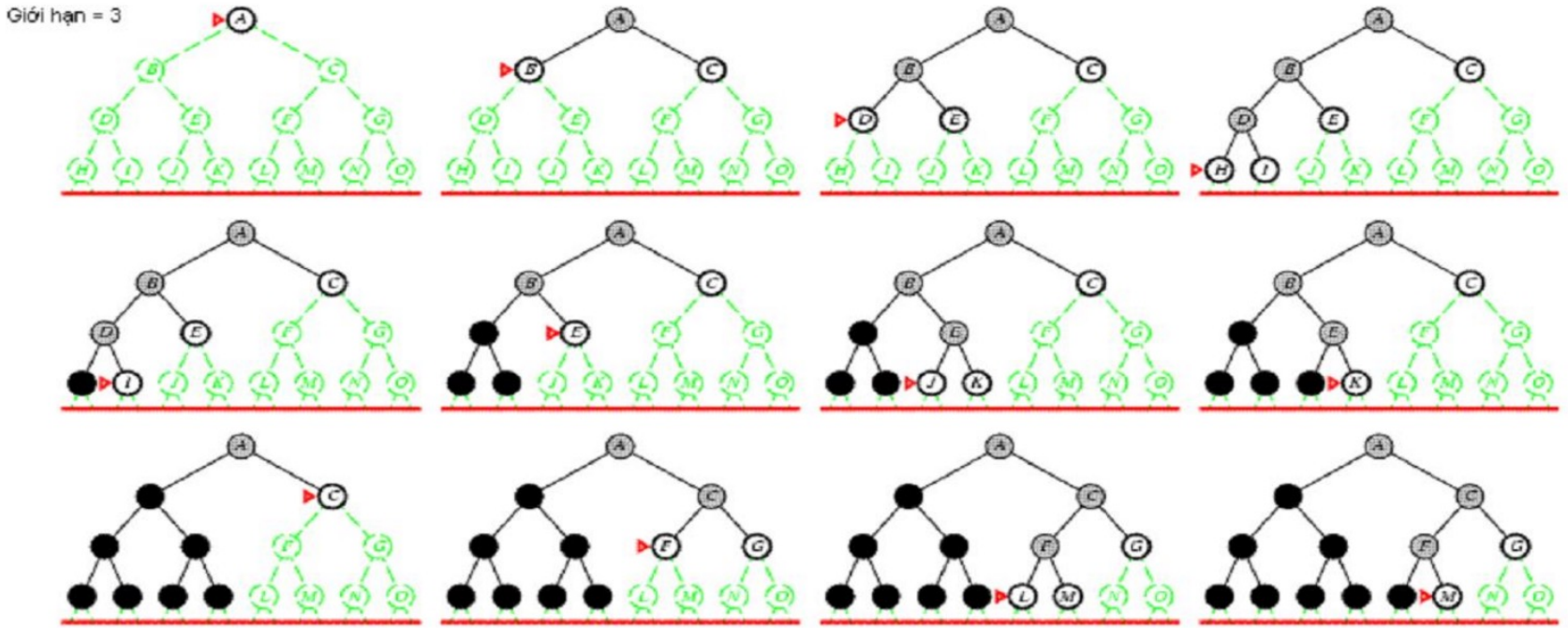
- Phương pháp: tìm theo DFS nhưng không bao giờ mở rộng các nút có độ sâu quá một giới hạn nào đó. Giới hạn độ sâu sẽ được tăng dần cho đến khi tìm được lời giải



Tìm kiếm sâu dần – IDS



Tìm kiếm sâu dần – IDS



IDS – Giải thuật

```
IDS (N, A, n0, ĐICH, l)                          // l: giới hạn độ sâu
{
    fringe ← n0;          closed ← ∅;  depth ← l;
    while  (fringe ≠ ∅)  do
    {   n ← GET_FIRST(fringe);          // lấy phần tử đầu tiên của fringe
        closed ← closed ⊕ n;
        if  (n ∈ ĐICH)  then  return  SOLUTION(n);
        if  (Γ(n) ≠ ∅) then
        {       case  d(n)  do                  // d(n): độ sâu của nút n
                [0..(depth-1)]:  fringe ← Γ(n) ⊕ fringe;
                depth:             fringe ← fringe ⊕ Γ(n);
                (depth+1):      {    depth ← depth + l;
                                     if  (l=1)  then  fringe ← fringe ⊕ Γ(n)
                                                 else  fringe ← Γ(n) ⊕ fringe;
                                }
        }
    }
    return  (“No solution”);
}
```

IDS – Tính chất

- Tính hoàn chỉnh?
  - Có
- Độ phức tạp về thời gian?
  - $(d+1)b^0 + d\,b^1 + (d-1)b^2 + \ldots + b^d = O(b^{d+1})$
- Độ phức tạp về bộ nhớ?
  - $O(bd)$
- Tính tối ưu?
  - Có, nếu chi phí cho mỗi bước (mỗi cạnh của cây tìm kiếm) = 1

So sánh giữa các giải thuật tìm kiếm cơ bản

	BFS	UCS	DFS	IDS
Complete?	Yes	Yes	No	Yes
Optimal?	Yes	Yes	No	Yes
Time	$O(b^d)$	$O(b^{\lceil C^*/\epsilon \rceil})$	$O(b^m)$	$O(b^d)$
Space	$O(b^d)$	$O(b^{\lceil C^*/\epsilon \rceil})$	$O(bm)$	$O(bd)$

So sánh giữa các giải thuật tìm kiếm cơ bản

- Nên chọn BFS nếu độ phân nhánh nhỏ
- Nên chọn DFS nếu biết trước độ sâu tối đa và nhiều trạng thái đích
- Nên chọn IDS nếu cây tìm kiếm có độ sâu m lớn

Khi nào đưa nút lặp vào danh sách

- BFS
  - Không: Việc đưa nút lặp vào hàng đợi không làm thay đổi thứ tự duyệt các nút duyệt trong hàng đợi. Không làm thay đổi nghiệm bài toán. Ngoài ra còn bị rơi vào vòng lặp
- UCS
  - Trong trường hợp nút lặp có chi phí tốt hơn, nó sẽ được đưa lại danh sách (nếu đã phát triển rồi) hoặc cập nhật nút cũ có giá thành kém hơn (nếu đang trong danh sách)

Khi nào đưa nút lặp vào danh sách

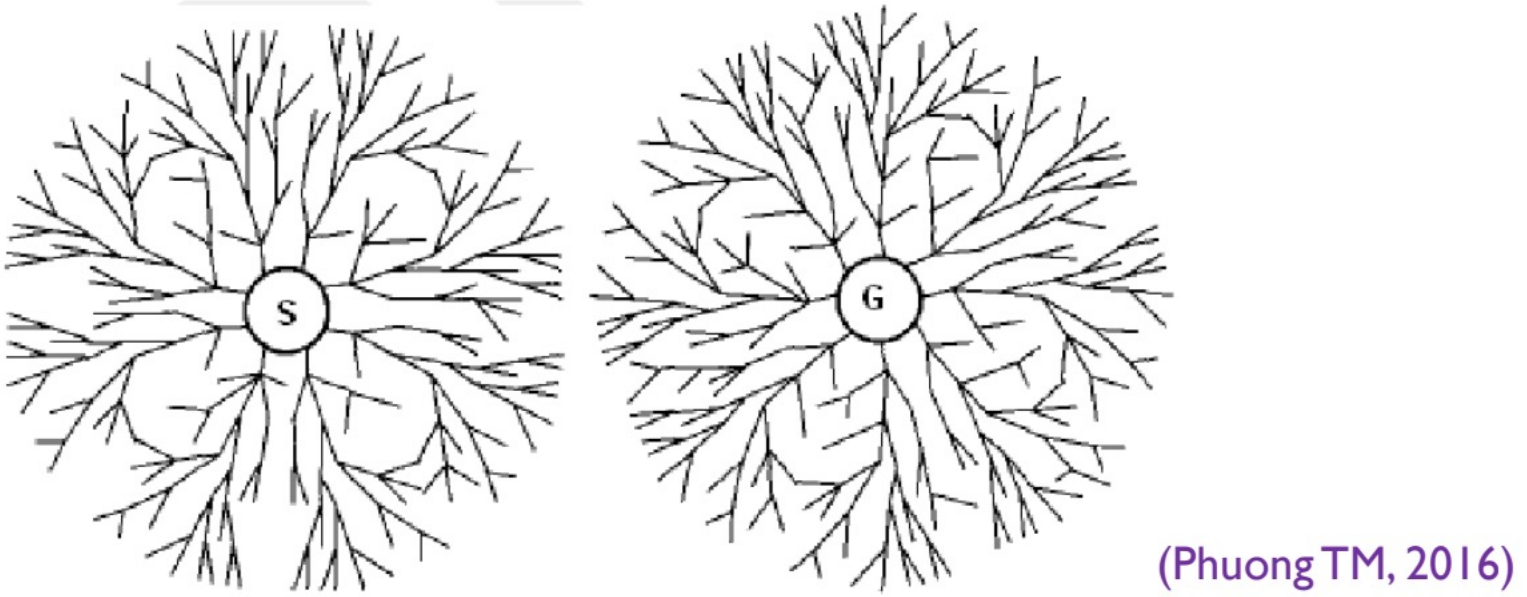
- DFS
  - Có: Việc đưa nút lặp vào ngăn xếp làm thay đổi thứ tự duyệt các nút trong ngăn xếp (thay đổi nhánh tìm kiếm), và thay đổi nghiệm của bài toán.
  - Tuy nhiên nếu đây là một nút đã được duyệt rồi thì sẽ không đưa vào ngăn xếp nữa.
- IDS:
  - Có: Để đảm bảo tính tối ưu của thuật toán

Tìm theo hai hướng

- Phương pháp: Tìm kiếm đồng thời bắt nguồn từ nút xuất phát và nút đích.
  - Tồn tại hai cây tìm kiếm, một cây có gốc là nút xuất phát, một cây có gốc là nút đích.
  - Tìm kiếm kết thúc khi có lá của cây này trùng với lá của cây kia

Tìm theo hai hướng

Minh hoạ cây tìm kiếm



(Phuong TM, 2016)

Tìm theo hai hướng

- Chú ý
  - Cần sử dụng tìm theo chiều rộng
    - Tìm theo chiều sâu có thể không ra lời giải nếu hai cây tìm kiếm phát triển theo hai nhánh không gặp nhau.
  - Cần xây dựng các hàm
    - P(x): Tập các nút con của x
    - D(x): Tập các nút con cha của x
- Tính chất
  - Việc kiểm tra nút lá này có trùng với nút lá kia đòi hỏi tương đối nhiều thời gian (b^d nút cây này với b^d nút của cây kia.
  - Độ phức tạp tính toán là $O(\mathrm{b}^{\mathrm{d}/2})$: Số lượng nút của hai cây giảm đáng kể.