Học viện
Công nghệ Bưu chính Viễn thông

Khoa Công nghệ thông tin 1

Tìm kiếm mù
(Tìm kiếm không có thông tin)

PGS.TS: Ngô Xuân Bách

Email: bachnx@ptit.edu.vn

Nội dung

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

Nội dung

- Bài toán 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
- Một số ví dụ
- Các thuật toán tìm kiếm cơ bản

Tìm kiếm & khoa học trí tuệ nhân tạo

- Nhiều vấn đề (bài toán) có thể phát biểu và giải quyết dưới dạng tìm kiếm
  - Trò chơi: tìm kiếm nước đi tối ưu (mang lại lợi thế)
  - Lập thời khóa biểu: tìm kiếm phương án sắp xếp thỏa mãn yêu cầu đề ra (thỏa mãn ràng buộc)
  - Tìm đường: tìm đường đi tối ưu (chiều dài, thời gian, giá, …)

- Tìm kiếm là một trong những hướng nghiên cứu quan trọng của trí tuệ nhân tạo
  - Phát triển các thuật toán tìm kiếm hiệu quả (đặc biệt trong những trường hợp không gian tìm kiếm có kích thước lớn)
  - Là cơ sở cho nhiều nhánh nghiên cứu khác của trí tuệ nhân tạo
    - Học máy, xử lý ngôn ngữ tự nhiên, suy diễn

Phát biểu bài toán tìm kiếm

Một bài toán tìm kiếm được phát biểu thông 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 tập các trạng thái nhận được từ trạng thái x do kết quả thực hiện hành động hay toán tử.
4. Xác định đích:
   - Tường minh, cho bởi tập đích $G \subseteq Q$
   - Không tường minh, cho bởi một số điều kiện
5. Giá thành đường đi
   - 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

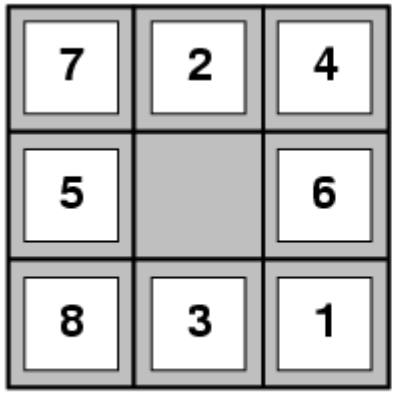
Các tiêu chuẩn đánh giá thuật toán tìm kiếm

- Độ phức tạp tính toán
  - Khối lượng tính toán cần thực hiện để tìm ra lời giải
  - Số lượng trạng thái cần xem xét trước khi tìm ra 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 không?

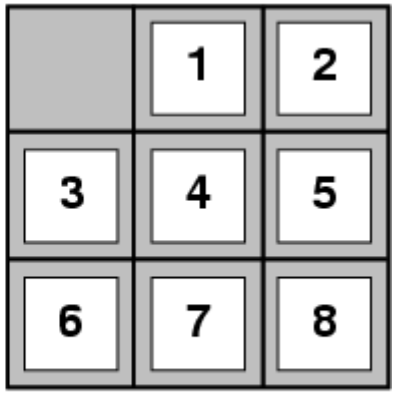
Nội dung

- Bài toán tìm kiếm trong không gian trạng thái
- Một số ví dụ
  - Trò chơi 8 ô
  - Bài toán 8 con hậu
- Các thuật toán tìm kiếm cơ bản

Trò chơi 8 ô (1/2)



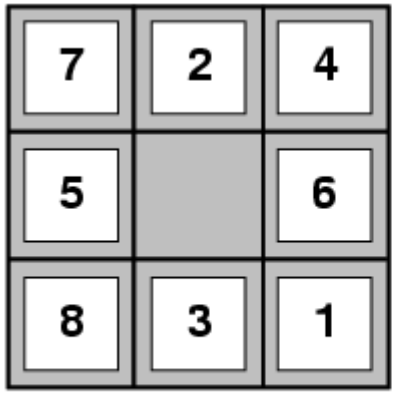
Start State



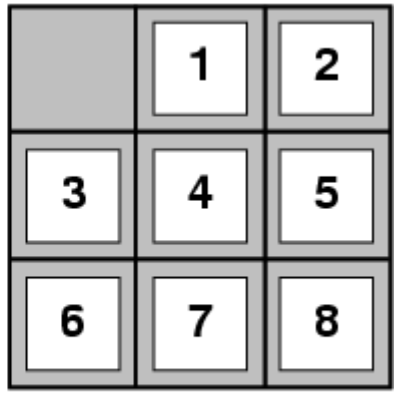
Goal State

(Russell & Norvig, 2010)

Trò chơi 8 ô (2/2)



Start State



Goal State

(Russell & Norvig, 2010)

- Trạng thái: tổ hợp ví trí các ô
- Trạng thái xuất phát: một trạng thái bất kỳ
- Hành động: di chuyển ô trống trái, phải, lên, xuống
- Đích: so sánh với trạng thái đích (cho trước)
- Giá thành: số lần di chuyển

Bài toán 8 con hậu (1/2)

Đặt 8 con hậu lên bàn cờ vua 8x8 sao cho không có đôi hậu nào đe dọa nhau

Bài toán 8 con hậu (2/2)

Đặt 8 con hậu lên bàn cờ vua 8x8 sao cho không có đôi hậu nào đe dọa nhau

- Trạng thái: sắp xếp của 0 đến 8 con hậu trên bàn cờ
- Trạng thái xuất phát: không có con hậu nào trên bàn cờ
- Hành động: đặt một con hậu lên một ô trống trên bàn cờ
- Đích: 8 con hậu trên bàn cờ, không có 2 con nào đe dọa nhau

Nội dung

- Bài toán tìm kiếm trong không gian trạng thái
- Một số ví dụ
- Các thuật toán tìm kiếm cơ bản
  - Thuật toán tìm kiếm tổng quát
  - Tìm kiếm theo chiều rộng (Breadth-first search: BFS)
  - Tìm kiếm theo giá thành thống nhất (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 (1/3)

- **Ý tưởng chung:** 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 tạo ra "cây tìm kiếm"
  - Mỗi trạng thái là một nút
  - 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 (2/3)

$Search(Q, S, G, P)$

(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 xóa 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 (3/3)

$Search(Q, S, G, P)$

(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)

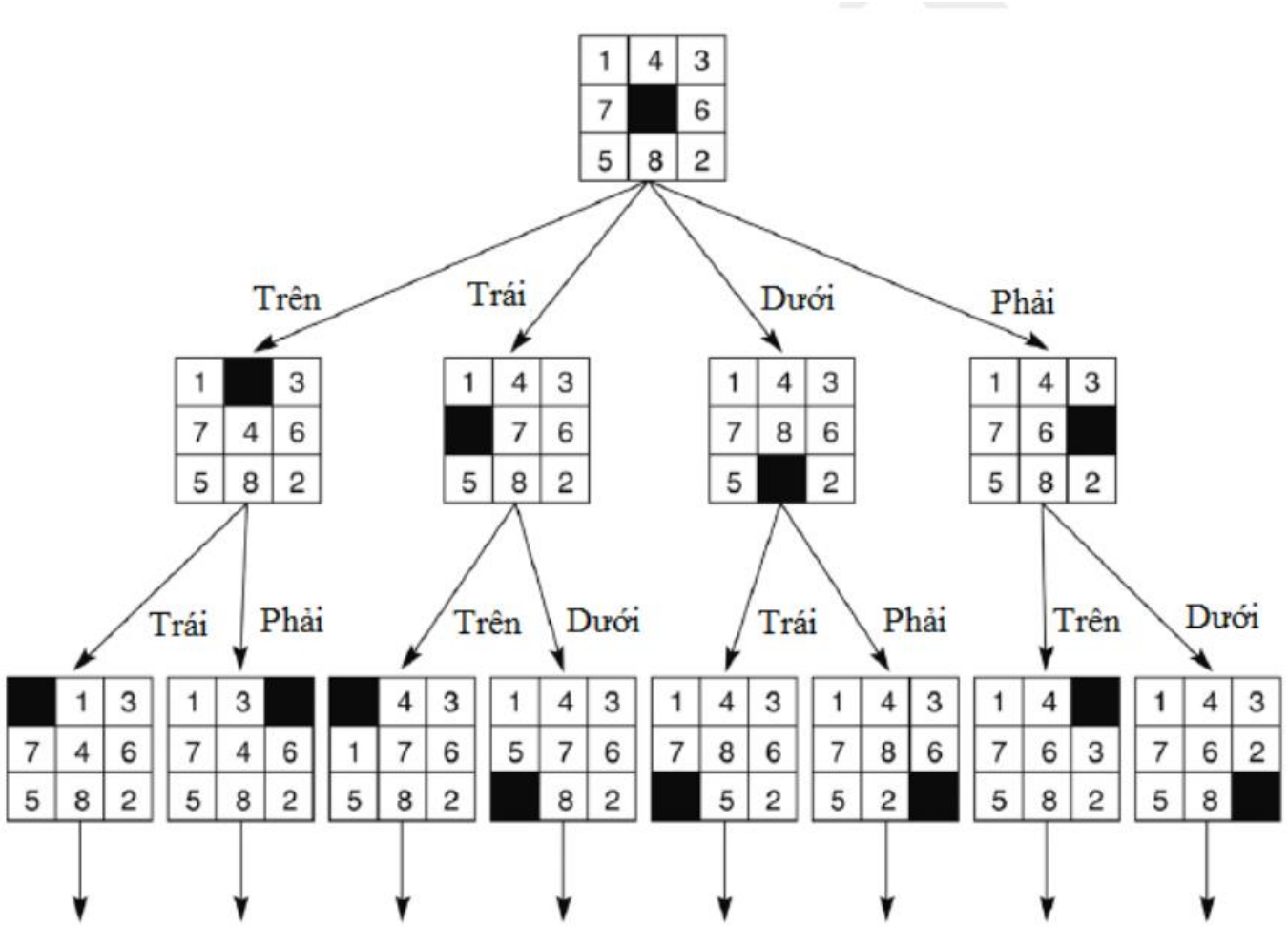
while($O \neq \emptyset$) **do**

Chọn nút n thế nào?

1. chọn nút $n \in O$ và xóa 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

Ví dụ cây tìm kiếm



Các chiến lược tìm kiếm

- Chiến lược tìm kiếm được xác định bởi thứ tự mở rộng các nút trên cây tìm kiếm

- Chiến lược tìm kiếm được đánh giá theo những tiêu chí sau:
  - đầy đủ: có tìm được lời giải không (nếu có)?
  - độ phức tạp tính toán: số lượng nút sinh ra
  - yêu cầu bộ nhớ: số lượng nút tối đa cần lưu trong bộ nhớ
  - tối ưu: có tìm được lời giải có giá thành nhỏ nhất không

- Độ phức tạp được tính theo các tham số 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 ∞)

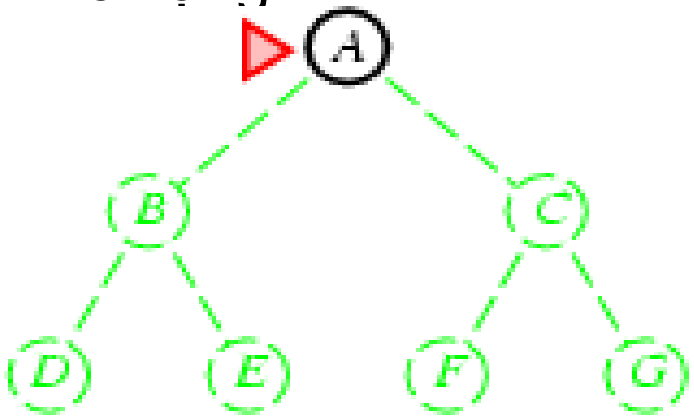
Tìm kiếm mù (Tìm kiếm không có thông tin)

- Tìm kiếm mù (blind, uninformed) chỉ sử dụng các thông tin theo phát biểu của bài toán trong quá trình tím kiếm

- Các phương pháp tìm kiếm mù
  - Tìm kiếm theo chiều rộng (Breadth-first search: BFS)
  - Tìm kiếm theo giá thành thống nhất (Uniform-cost search: UCS)
  - Tìm kiếm theo chiều sâu (Depth-first search: DFS)
  - Tìm kiếm sâu dần (Iterative deepening search: IDS)

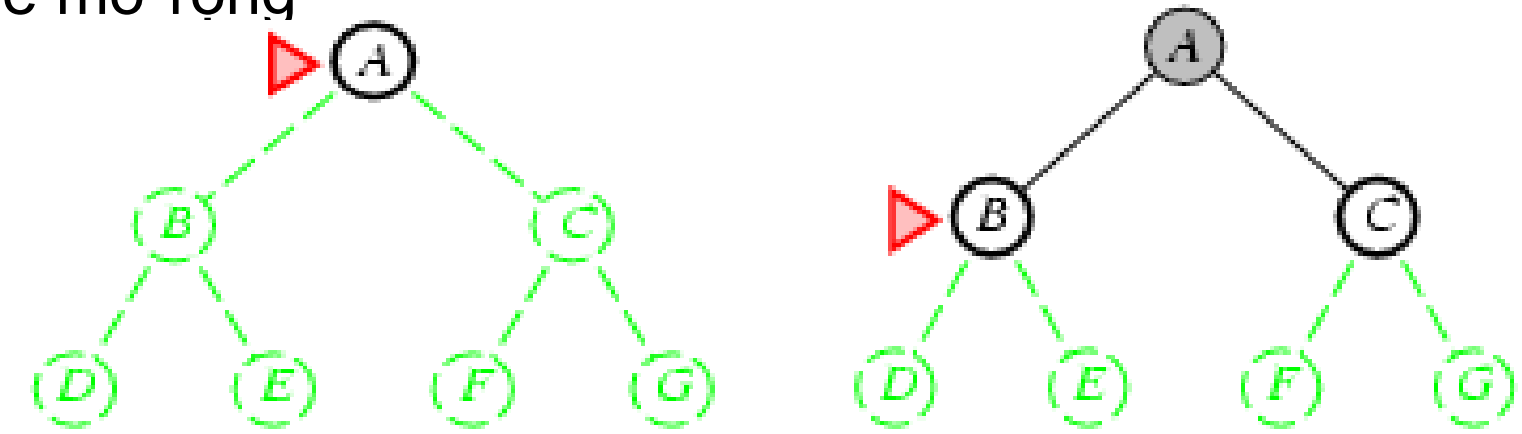
Tìm kiếm theo chiều rộng – BFS (1/4)

Nguyên tắc: trong số những nút biên, lựa chọn nút nông nhất (gần gốc nhất) để mở rộng



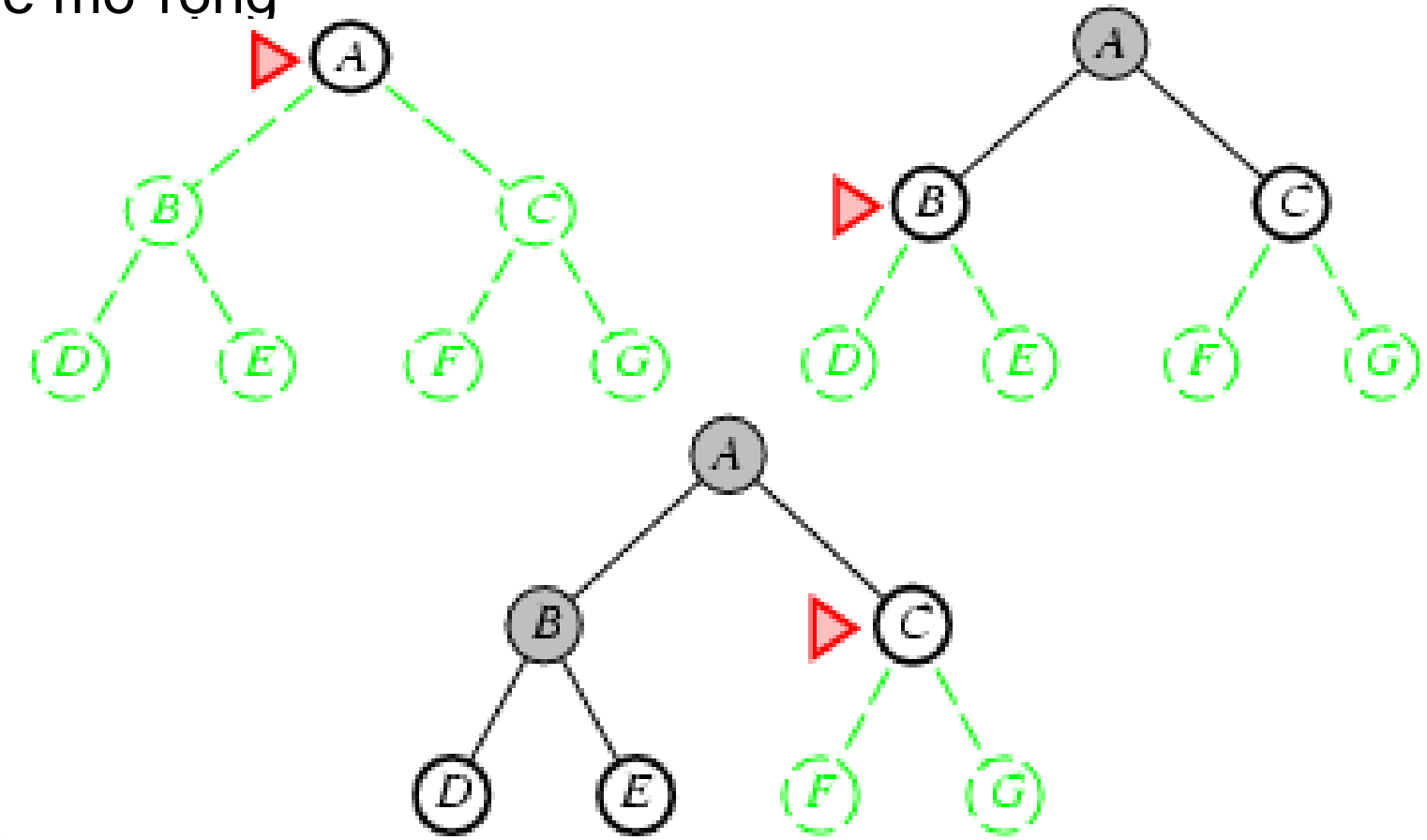
Tìm kiếm theo chiều rộng – BFS (2/4)

Nguyên tắc: trong số những nút biên, lựa chọn nút nông nhất (gần gốc nhất) để mở rộng



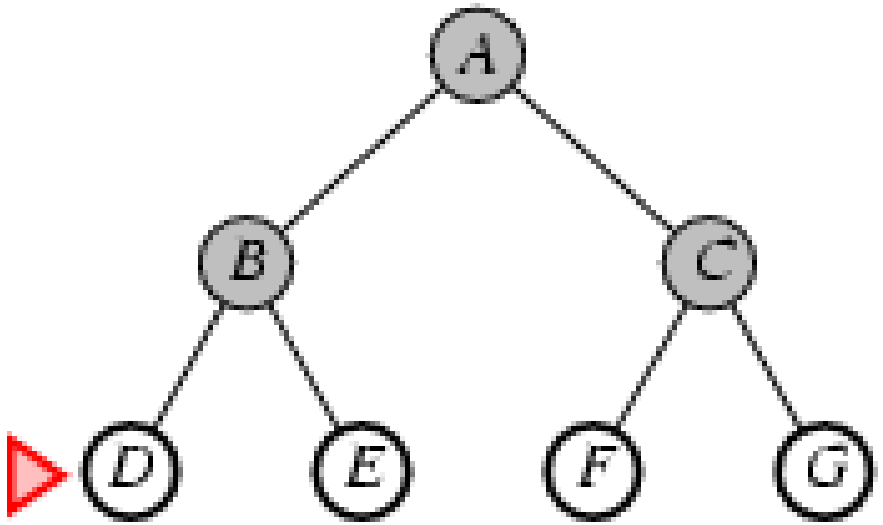
Tìm kiếm theo chiều rộng – BFS (3/4)

Nguyên tắc: trong số những nút biên, lựa chọn nút nông nhất (gần gốc nhất) để mở rộng



Tìm kiếm theo chiều rộng – BFS (4/4)

- Ghi nhớ đường đi
  - Khi chuyển sang một nút cần ghi nhớ nút cha của nút đó bằng cách sử dụng con trỏ ngược
  - Sau khi đạt tới đích, con trỏ ngược được sử dụng tìm đường đi trở lại nút xuất phát



Thuật toán BFS (1/2)

$BFS(Q, S, G, P)$

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

Đầu vào: bài toán tìm kiếm

Đầu ra: đường tới nút đích

Khởi tạo: tập các nút biên (nút mở) $O = S$

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

1. lấy nút đầu tiên n khỏi O
2. **if** $n \in G$, **return** (đường đi tới n)
3. thêm $P(n)$ vào đuôi O

return không tìm được đường đi

Thuật toán BFS (2/2)

$BFS(Q, S, G, P)$

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

Đầu vào: bài toán tìm kiếm

Đầu ra: đường tới nút đích

Khởi tạo: tập các nút biên (nút mở) $O = S$

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

1. lấy nút đầu tiên n khỏi O
2. **if** $n \in G$, **return** (đường đi tới n)
3. thêm $P(n)$ vào đuôi O

return không tìm được đường đi

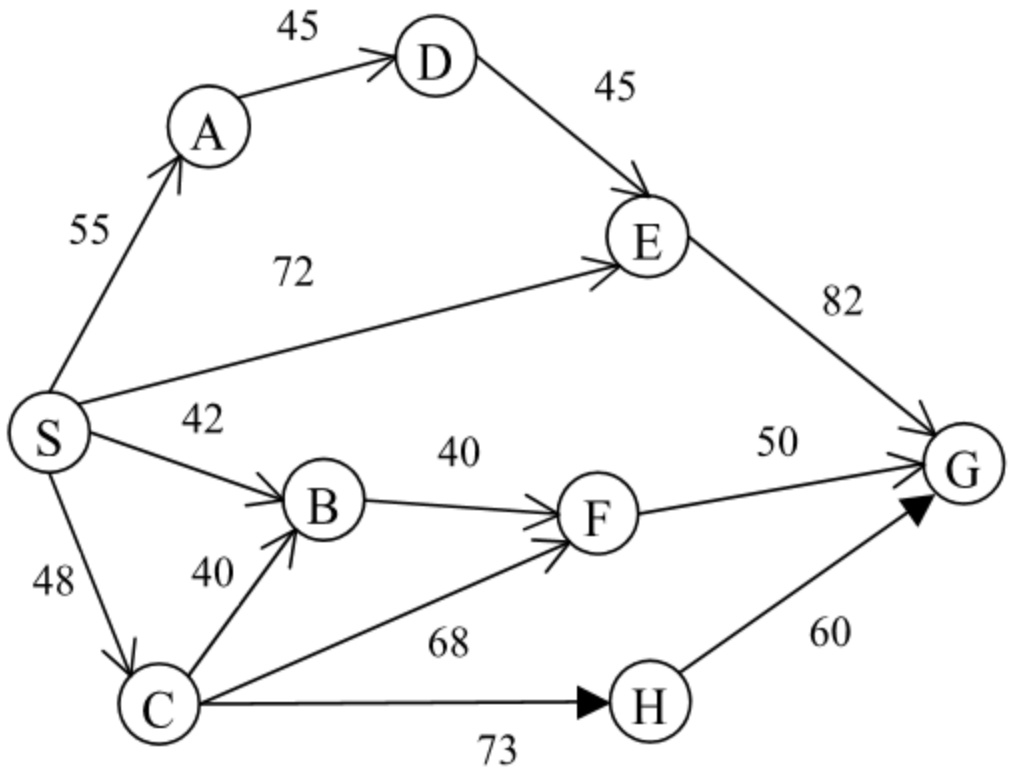
Sử dụng cấu trúc hàng đợi FIFO

Tránh các nút lặp

- Có thể có nhiều đường đi cùng dẫn tới một 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 BFS (1/2)



Ví dụ BFS (2/2)

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: $G \leftarrow E \leftarrow S$

Tính chất của BFS

- Đầy đủ?
  - Có (nếu b là hữu hạn)
- Thời gian?
  - $1 + b + b^2 + b^3 + \cdots + b^d = O(b^d)$
- Bộ nhớ?
  - $O(b^d)$ (lưu tất cả các nút)
- Tối ưu?
  - Có (nếu cost = 1 với mọi bước)
  - Do luôn tìm kiếm tất cả các nút ở mức trên trước khi tìm sang nút ở mức dưới

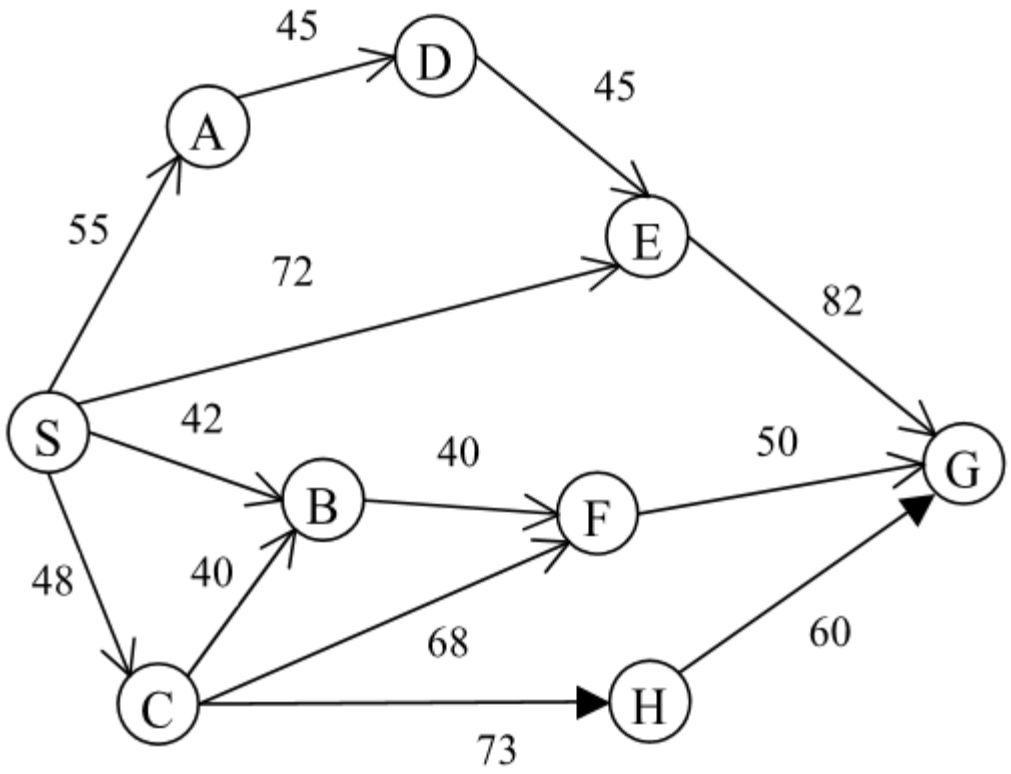
Bộ nhớ là vấn đề quan trọng hơn thời gian

Tìm kiếm theo giá thành thống nhất

- Trong trường hợp giá thành 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 giá thành thống nhất (là một biến thể của BFS)

- **Phương pháp**: chọn nút có giá thành nhỏ nhất để mở rộng trước thay vì chọn nút nông nhất như trong BFS

Ví dụ UCS (1/2)



Ví dụ UCS (2/2)

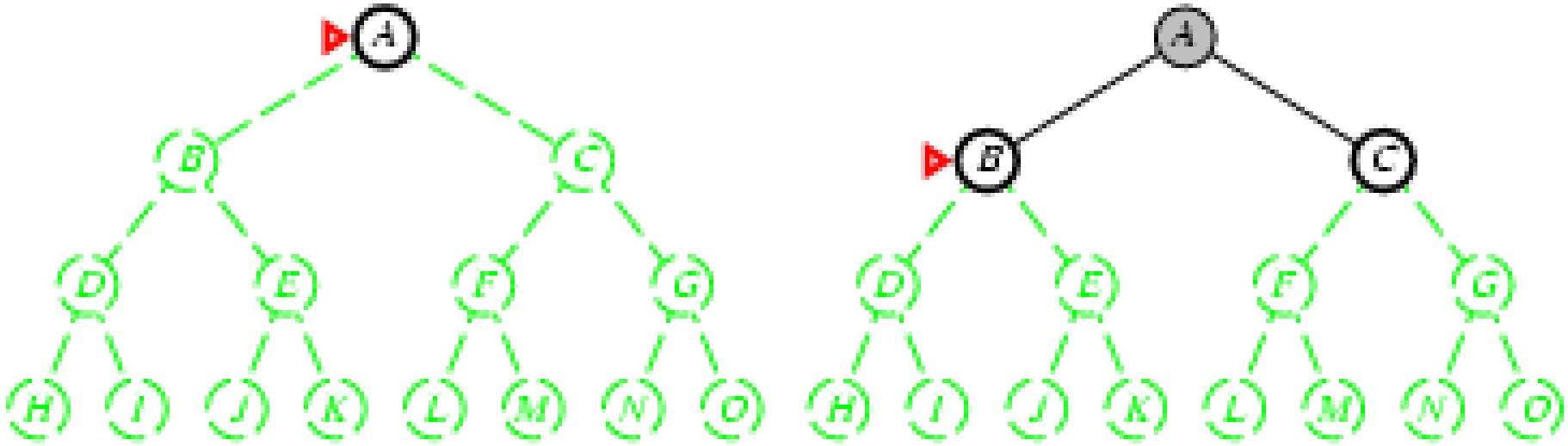
STT	Nút được mở rộng	Tập biên O
0		$S(0)$
1	S	$A_S(55), B_S(42), C_S(48), E_S(72)$
2	B_S	$A_S(55), C_S(48), E_S(72), F_B(82)$
3	C_S	$A_S(55), E_S(72), F_B(82), H_C(121)$
4	A_S	$E_S(72), F_B(82), H_C(121), D_A(100)$
5	E_S	$F_B(82), H_C(121), D_A(100), G_E(154)$
6	F_B	$H_C(121), D_A(100), G_F(132)$
7	D_A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

Cập nhật đường đi tới G

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

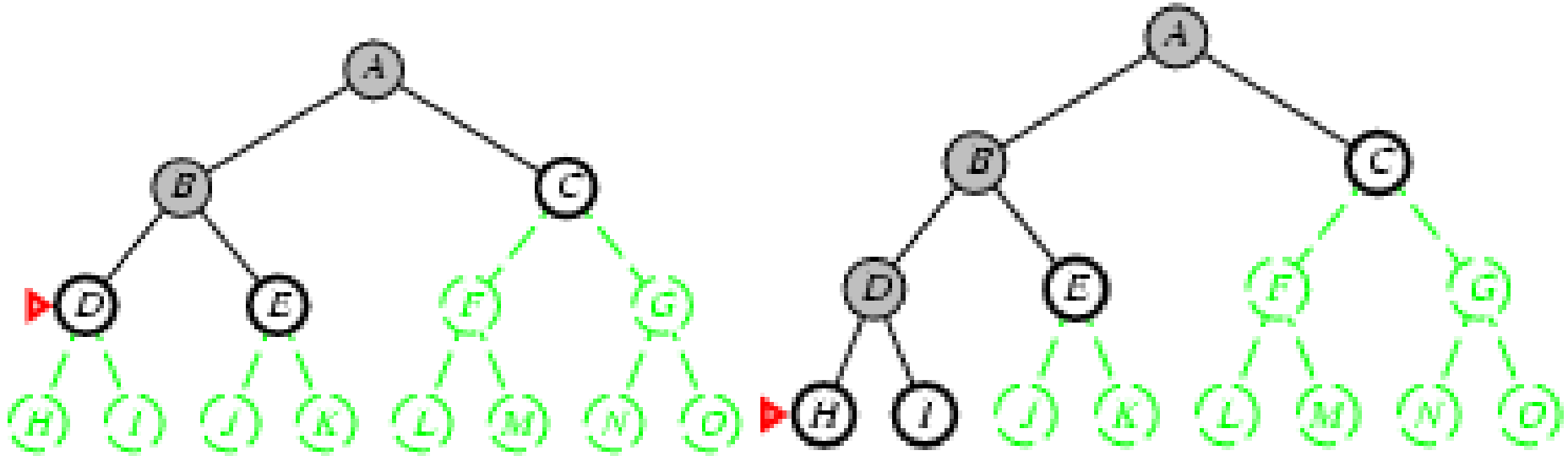
Tìm kiếm theo chiều sâu – DFS (1/4)

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



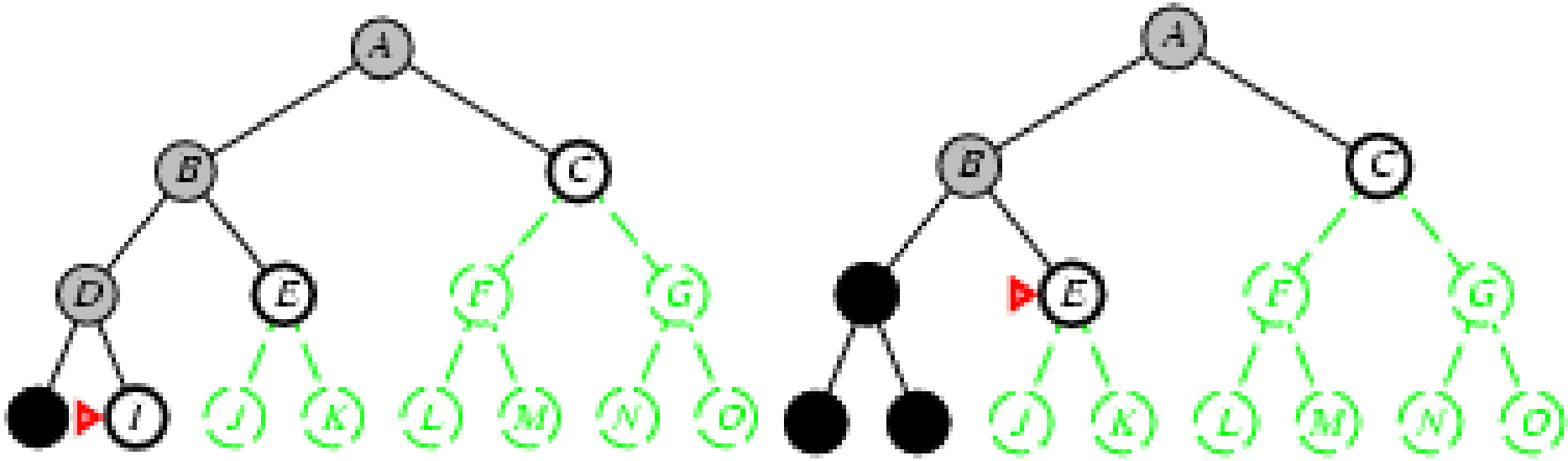
Tìm kiếm theo chiều sâu – DFS (2/4)

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



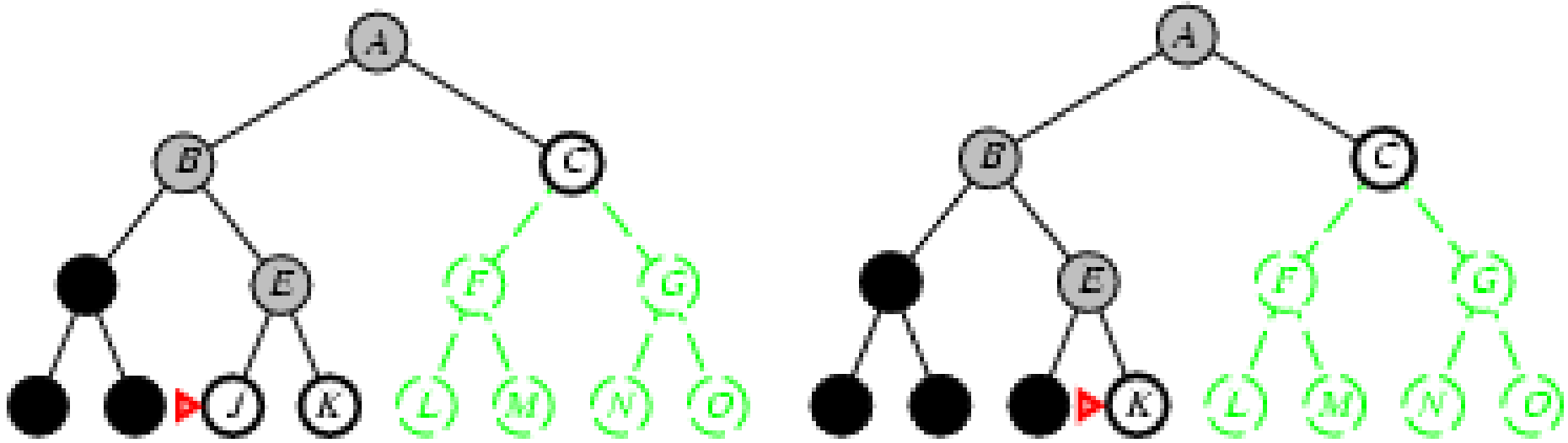
Tìm kiếm theo chiều sâu – DFS (3/4)

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



Tìm kiếm theo chiều sâu – DFS (4/4)

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



Thuật toán DFS (1/2)

$DFS(Q, S, G, P)$

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

Đầu vào: bài toán tìm kiếm

Đầu ra: đường tới nút đích

Khởi tạo: tập các nút biên (nút mở) $O = S$

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

1. lấy nút đầu tiên n khỏi O
2. **if** $n \in G$, **return** (đường đi tới n)
3. thêm $P(n)$ vào đầu O

return không tìm được đường đi

Thuật toán DFS (2/2)

$DFS(Q, S, G, P)$

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

Đầu vào: bài toán tìm kiếm

Đầu ra: đường tới nút đích

Khởi tạo: tập các nút biên (nút mở) $O = S$

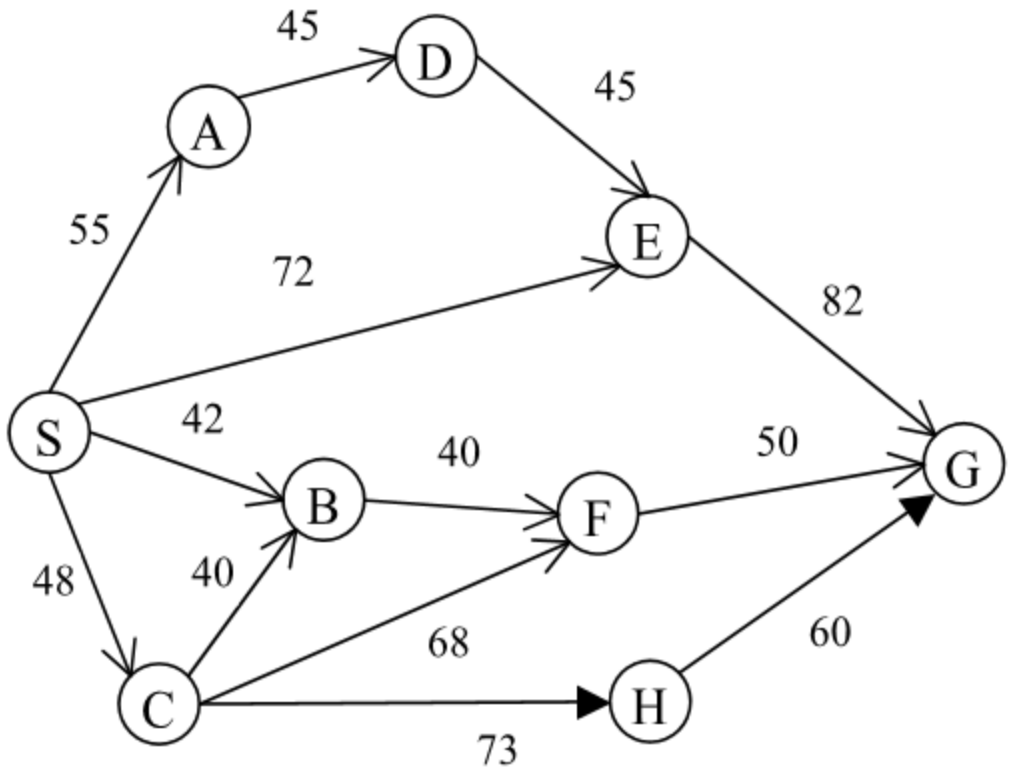
while($O \neq \emptyset$) **do**

1. lấy nút đầu tiên n khỏi O
2. **if** $n \in G$, **return** (đường đi tới n)
3. thêm $P(n)$ vào đầu O

return không tìm được đường đi

Sử dụng cấu trúc ngăn xếp LIFO

Ví dụ DFS (1/2)



Ví dụ DFS (2/2)

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

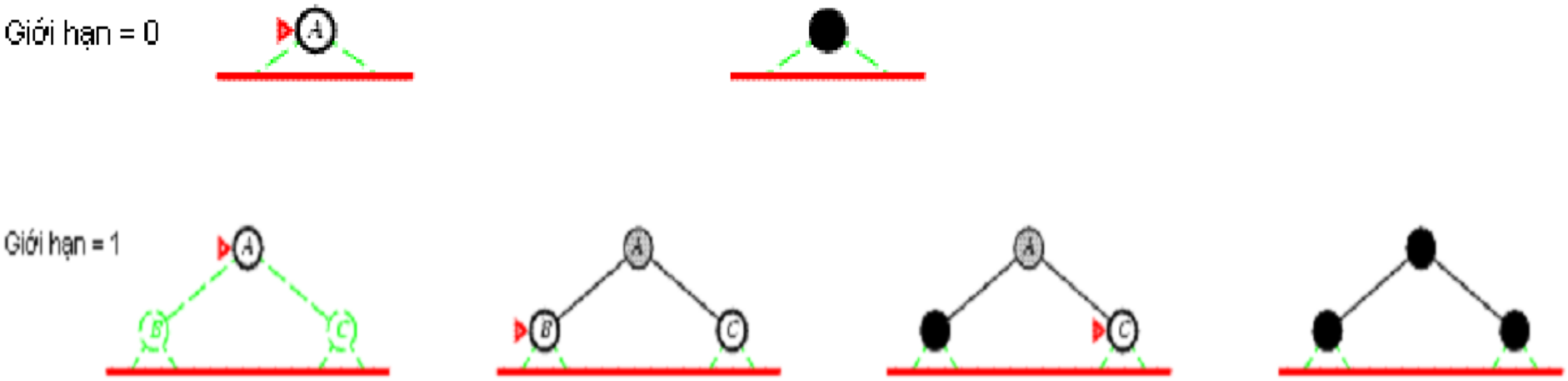
Đường đi: $G \leftarrow E \leftarrow D \leftarrow A \leftarrow S$
Độ sâu: 4

Tính chất của DFS

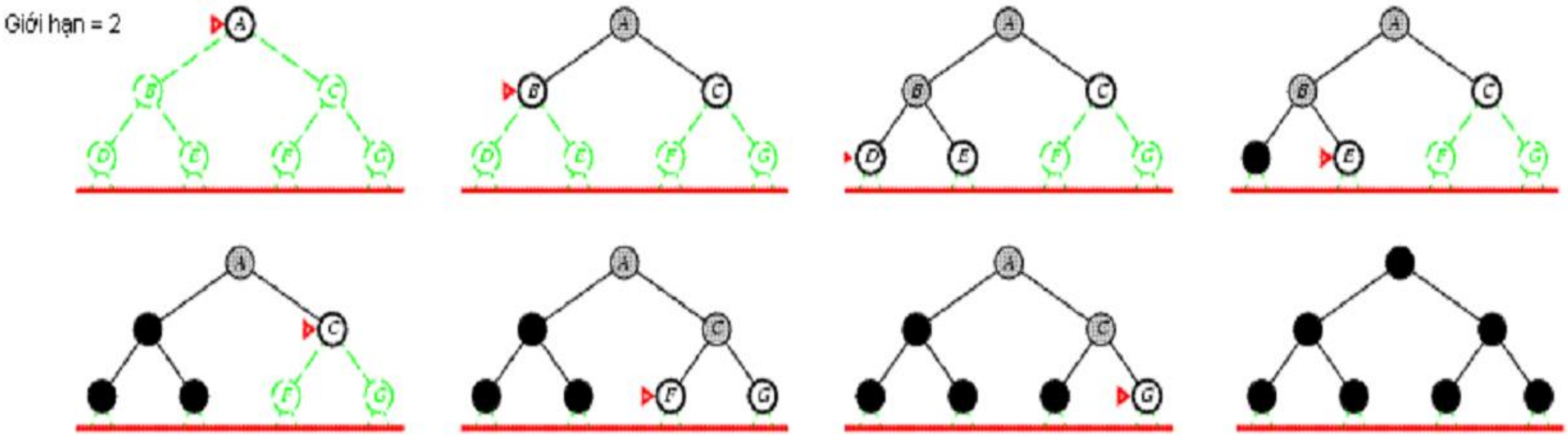
- Đầy đủ?
  - Không : trong trường hợp không gian trạng thái có độ sâu vô hạn
- Tối ưu?
  - Không
- Thời gian?
  - $O(b^m)$: rất lớn nếu m lớn hơn d
  - Nếu có nhiều lời giải thì có thể nhanh hơn tìm kiếm theo chiều rộng nhiều
- Bộ nhớ?
  - $O(bm)$: tốt hơn nhiều so với tìm kiếm theo chiều rộng

Tìm kiếm sâu dần – IDS (1/3)

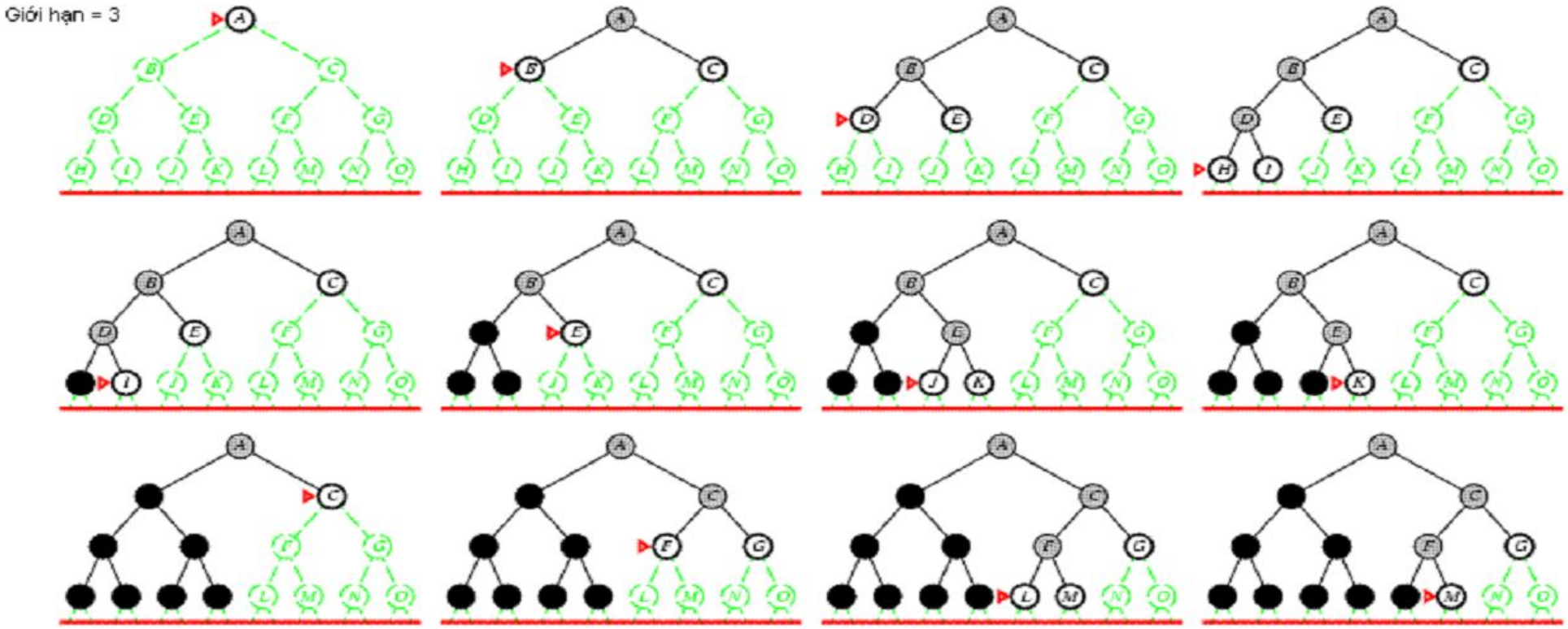
Phương pháp: tìm theo DFS nhưng không bao giờ mở rộng các nút có độ sâu quá một giới hạn nào đó. Giới hạn độ sâu sẽ được tăng dần cho đến khi tìm được lời giải.



Tìm kiếm sâu dần – IDS (2/3)



Tìm kiếm sâu dần – IDS (3/3)



Thuật toán IDS

$IDS(Q, S, G, P)$

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

Đầu vào: bài toán tìm kiếm

Đầu ra: đường tới nút đích

Khởi tạo: tập các nút biên (nút mở) $O = S$

$c = 0$ là độ sâu hiện thời

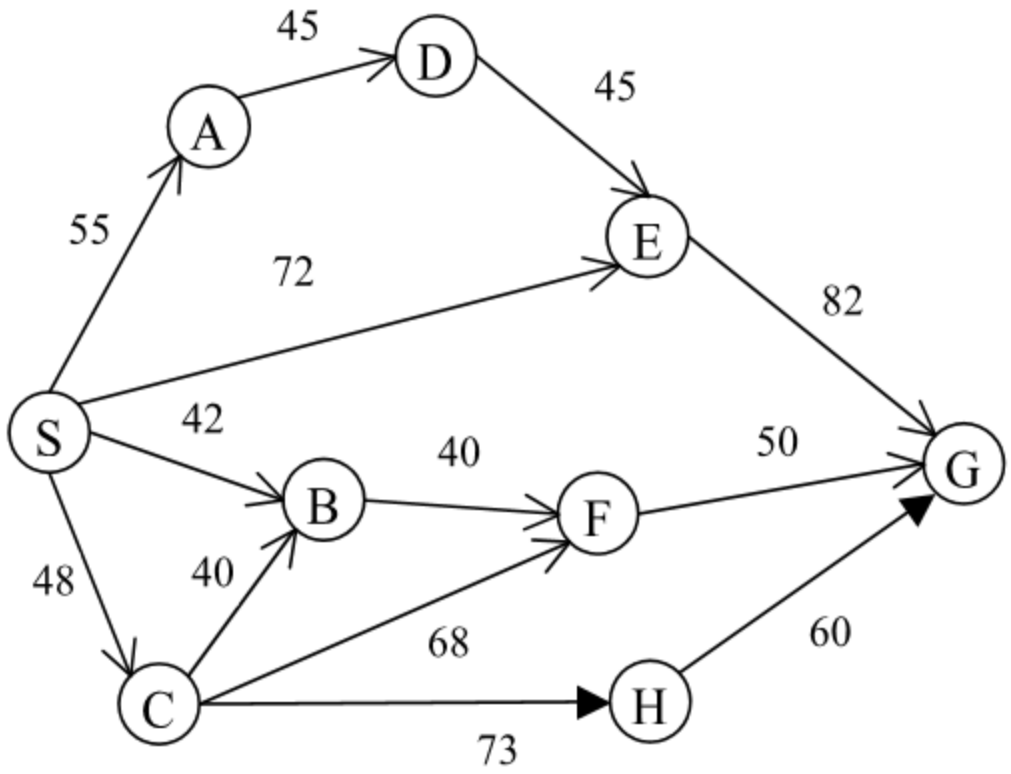
while (1) **do**

1. **while** ($O \neq \emptyset$) **do**
   a. lấy nút đầu tiên n khỏi O
   b. **if** $n \in G$, **return** (đường đi tới n)
   c. **if** $depth(n) < c$ **then**
      thêm $P(n)$ vào đầu O
2. $c++;\ O = S$

Tính chất của IDS

- Đầy đủ?
  - Có
- Tối ưu?
  - Có: nếu có nhiều lời giải, có thể tìm ra lời giải gần gốc nhất
- Bộ nhớ?
  - $O(bd)$: nhỏ
- Thời gian?
  - $(d+1)1 + db + (d-1)b^2 + \ldots + 2b^{d-1} + 1b^d = O(b^d)$

Ví dụ IDS



Tóm tắt

	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	$O(b^d)$	$O(b^{\lceil C^*/\epsilon \rceil})$	$O(bm)$	$O(bd)$

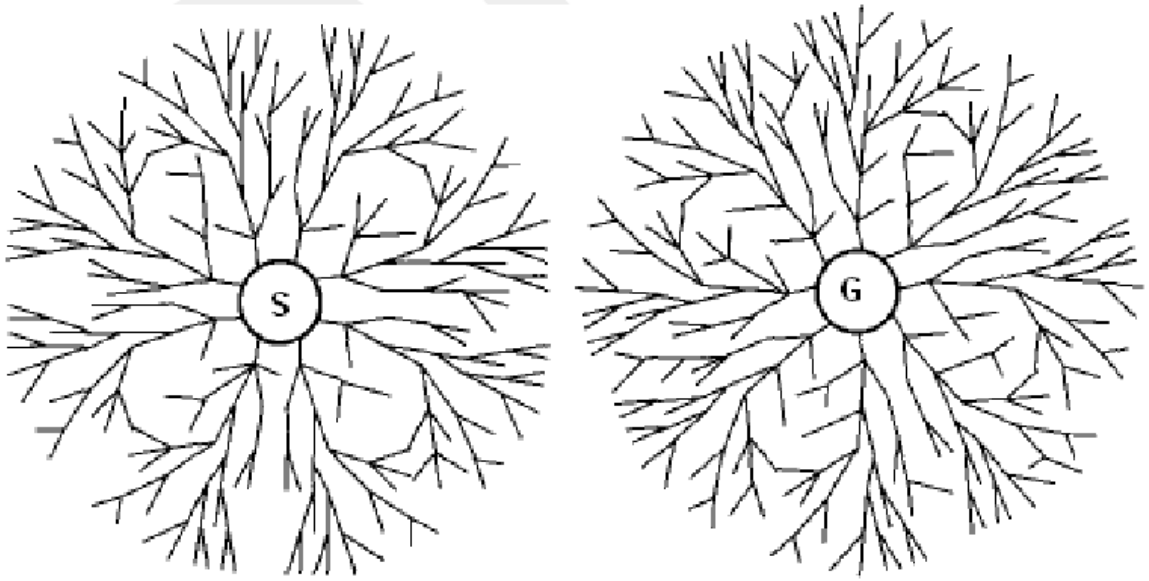
- Nên chọn BFS nếu độ phân nhánh b nhỏ
- Nên chọn DFS nếu biết trước độ sâu tối đa và có nhiều trạng thái đích
- Nên chọn IDS nếu cây tìm kiếm có độ sâu m lớn

Khi nào đưa nút lặp vào danh sách?

- BFS
  - **Không**: Việc đưa nút lặp vào hàng đợi không làm thay đổi thứ tự 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 giá thành (chi phí) tốt hơn, nó sẽ được đưa lại danh sách (nếu đã phát triển rồi) hoặc cập nhật thay nút cũ có giá thành kém hơn (nếu đang trong danh sách)
- DFS
  - **Có**: Việc đưa nút lặp vào ngăn xếp sẽ 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 (1/2)

- **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
- Minh họa cây tìm kiếm



Tìm theo hai hướng (2/2)

- 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 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 của cây này với b^d nút của cây kia)
  - Độ phức tạp tính toán là $O(b^{d/2})$
    - Số lượng nút cần mở rộng của hai cây giảm đáng kể