NHẬP MÔN TRÍ TUỆ NHÂN TẠO

GIẢI QUYẾT VẤN ĐỀ BẰNG TÌM KIẾM

ThS. Vũ Hoài Thư

Ngày 31 tháng 1 năm 2024

Nội dung

Bài toán tìm kiếm trong không gian trạng thái

Tìm kiếm và 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:

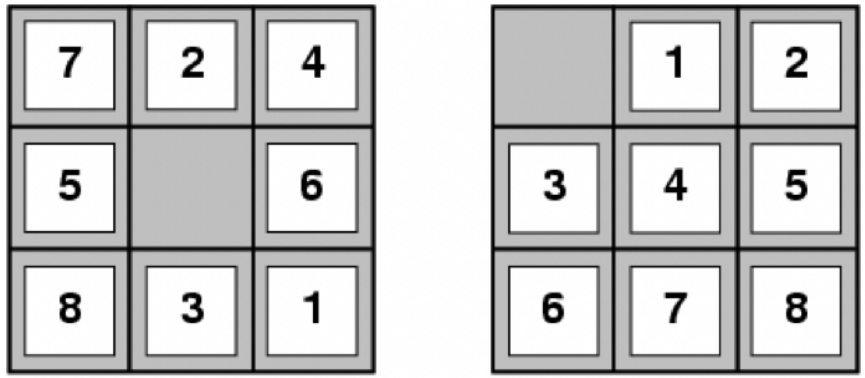
1. Tập hữu hạn các trạng thái có thể: Q
2. Tập các trạng thái xuất phát: $\mathrm{S} \subseteq \mathrm{Q}$
3. Hành động hay hàm nối tiếp hay toán tử $\mathrm{P}(x)$, là tập các trạng thái nhận được từ trạng thái x do kết quả thực hiện hành động hay toán tử
4. Xác định đích:
   - Đích tường minh, cho bởi tập đích $\mathrm{G} \subseteq \mathrm{Q}$
   - Đích không tường minh, cho bởi một số điều kiện.
5. Giá thành đường đi:
   - Ví dụ: tổng khoảng cách, số lượng hành động
   - Gọi $c(x, a, y) > 0$ là giá thành bước, từ trạng thái x, thực hiện hành động a và chuyển sang trạng thái y.

Các tiêu chuẩn đánh giá thuật toán tìm kiếm

1. 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
2. Yêu cầu bộ nhớ:
   - Số lượng trạng thái cần lưu trữ đồng thời trong bộ nhớ khi thực hiện thuật toán
3. .Tính đầy đủ:
   - Nếu bài toán có lời giải thì thuật toán có khả năng tìm ra lời giải đó không?
4. Tính tối ưu:
   - Nếu bài toán có nhiều lời giải thì thuật toán có cho phép tìm ra lời giải tốt nhất không?

Một số ví dụ

Trò chơi 8 ô



- Trạng thái: tổ hợp vị trí các ô
- Trạng thái xuất phát: một trạng thái bất ký
- Hành động: di chuyển các ô lên trên, xuống dưới, sang trái, phải
- Đích: so sánh với trạng thái đích cho trước (khi cả 8 ô được xếp đungs vị trí)
- Giá thành: số lần di chuyển

Bài toán 8 con hậu

Đặt 8 con hậu lên bàn cờ vua kích thước 8×8 sao cho không có đôi hậu nào đe doạ nhau.

- Trạng thái: sắp xếp của 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 doạ nhau

Các thuật toán tìm kiếm cơ bản

Thuật toán tìm kiếm tổng quát

- 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":
  - Một trạng thái là một nút
  - Các nút biên (nút mở) là nút đang chờ mở rộng tiếp
  - Nút đã mở rộng được gọi là nút đóng

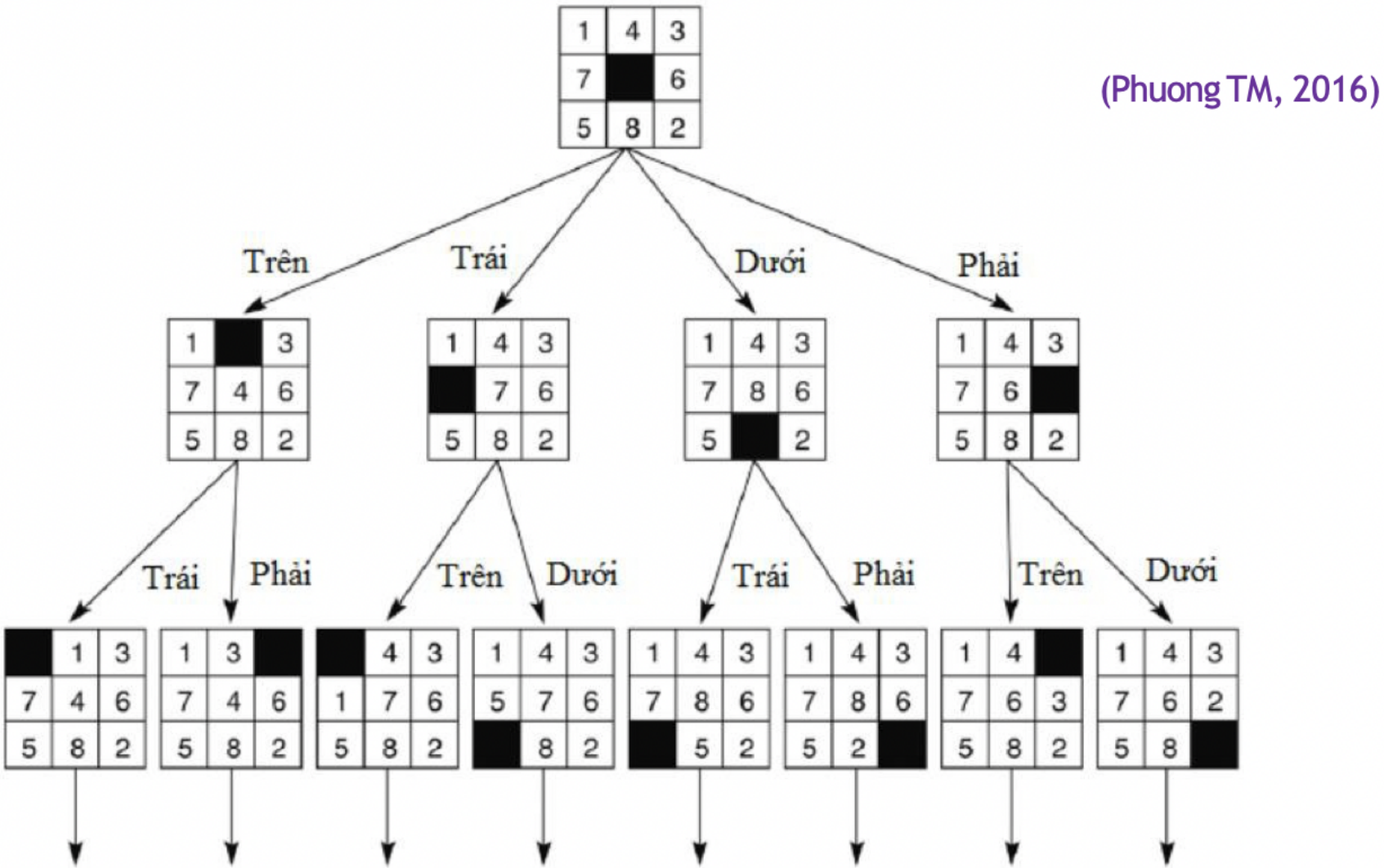
Thuật toán tìm kiếm tổng quát

Search$(\mathrm{Q}, \mathrm{S}, \mathrm{G}, \mathrm{P})$
(Q: không gian trạng thái; S: trạng thái bắt đầu; G: trạng thái đích; P: hành động)
Đầu vào: Bài toán tìm kiếm
Đầu ra: Trạng thái đích
Khởi tạo: $\mathrm{O} \leftarrow \mathrm{S}$ (O: danh sách các nút mở)
while$(\mathrm{O} \neq \emptyset)$ **do**:

1. Chọn nút $n \in \mathrm{O}$ và xoá n khỏi O
2. **if** $n \in \mathrm{G}$, **return** (đường đi tới n)
3. thêm $\mathrm{P}(n)$ 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 hay không (nếu có)
  - Độ phức tạp tính toán: số lượng các 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 ∞)

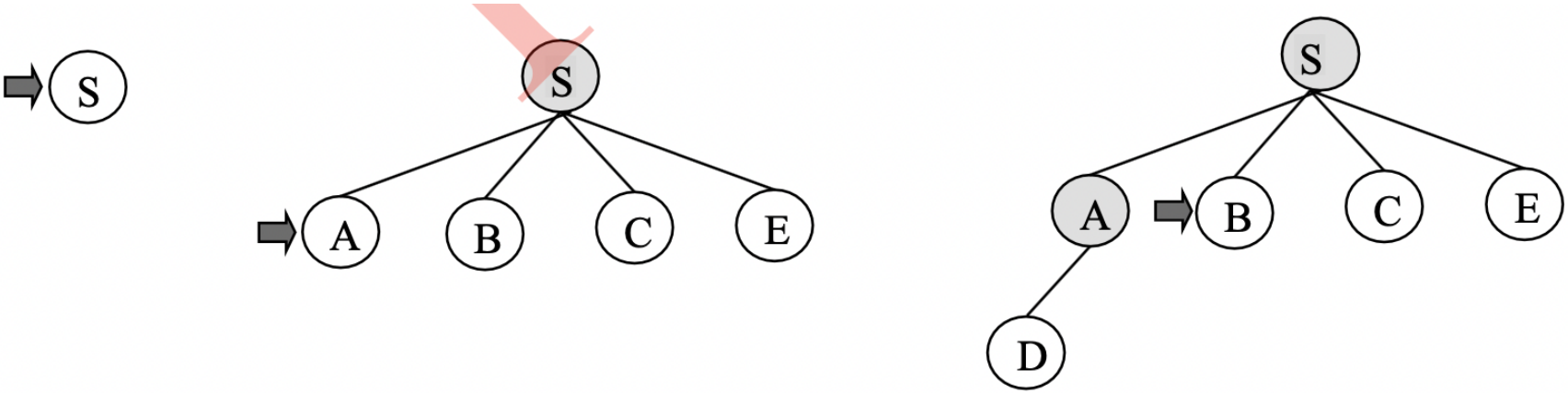
Tìm kiếm mù (Tìm kiếm không có thông tin)

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

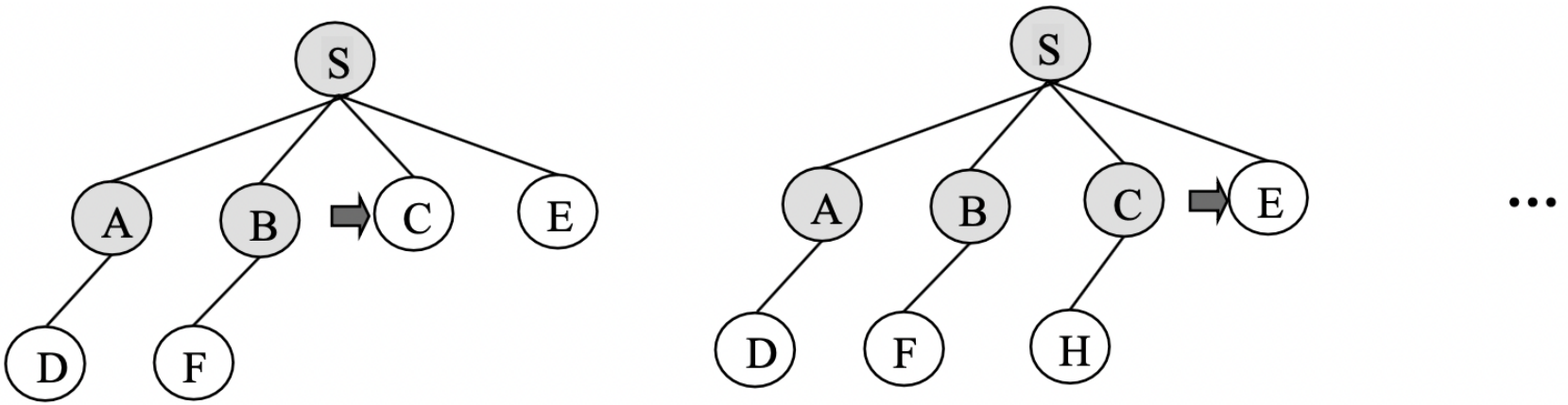
Tìm kiếm theo chiều rộng - BFS

Nguyên tắc: Trong số các nút biên, lựa chọn nút nông nhất (gần gốc nhất) để mở rộng.



Tìm kiếm theo chiều rộng - BFS

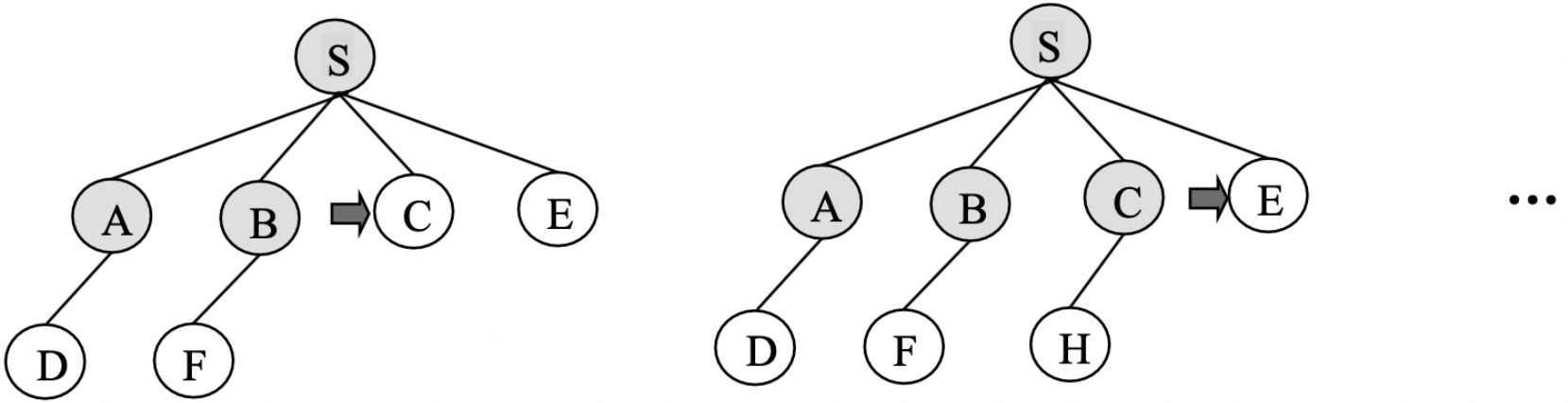
Nguyên tắc: Trong số các nút biên, lựa chọn nút nông nhất (gần gốc nhất) để mở rộng.



Tìm kiếm theo chiều rộng - BFS

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

BFS($\mathrm{Q}, \mathrm{S}, \mathrm{G}, \mathrm{P}$)
(Q: không gian trạng thái; S: trạng thái bắt đầu; G: trạng thái đích; P: hành động)
Đầu vào: Bài toán tìm kiếm
Đầu ra: Trạng thái đích
Khởi tạo: $\mathrm{O} \leftarrow \mathrm{S}$ (O: danh sách các nút mở)
while($\mathrm{O} \neq \emptyset$) **do**:

1. Lấy nút đầu tiên n khỏi O
2. **if** $n \in \mathrm{G}$, **return** (đường đi tới n)
3. thêm $\mathrm{P}(n)$ vào đuôi O (Sử dụng cấu trúc hàng đợi FIFO)

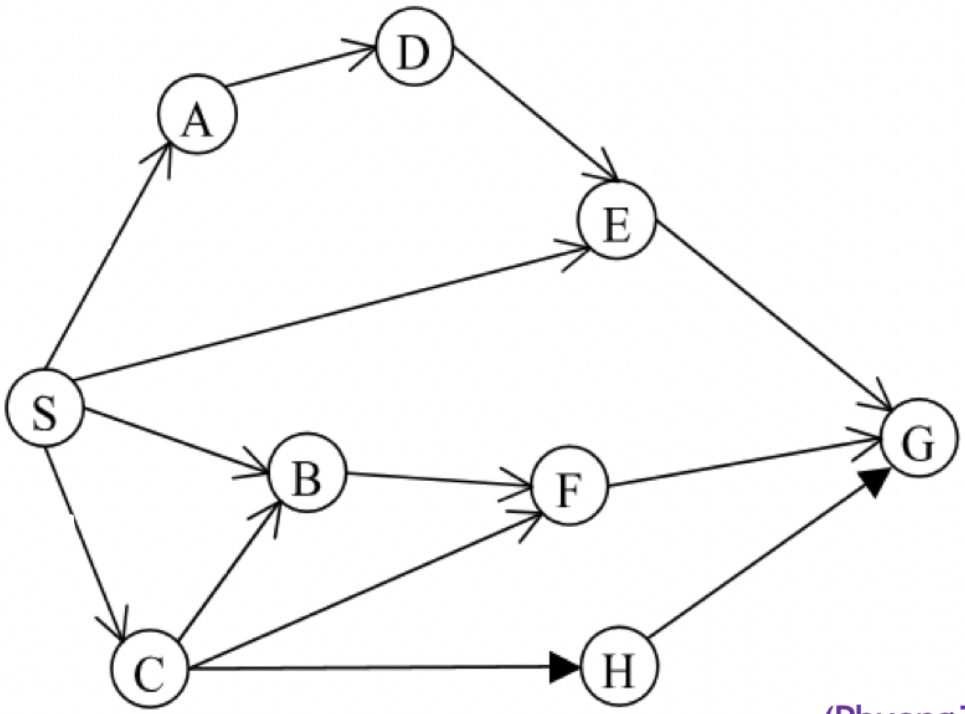
return: Không tìm được đường đi.

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 (tập các nút chờ được duyệt)
  - Cần nhớ ít nút, thời gian kiểm tra nhanh, tránh được vòng lặp vô hạn.

Ví dụ BFS

Tìm đường đi từ nút S tới nút G sử dụng thuật toán BFS



(Phuong TM, 2016)

Ví dụ 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

Kết quả đường đi: $G \leftarrow E \leftarrow S$

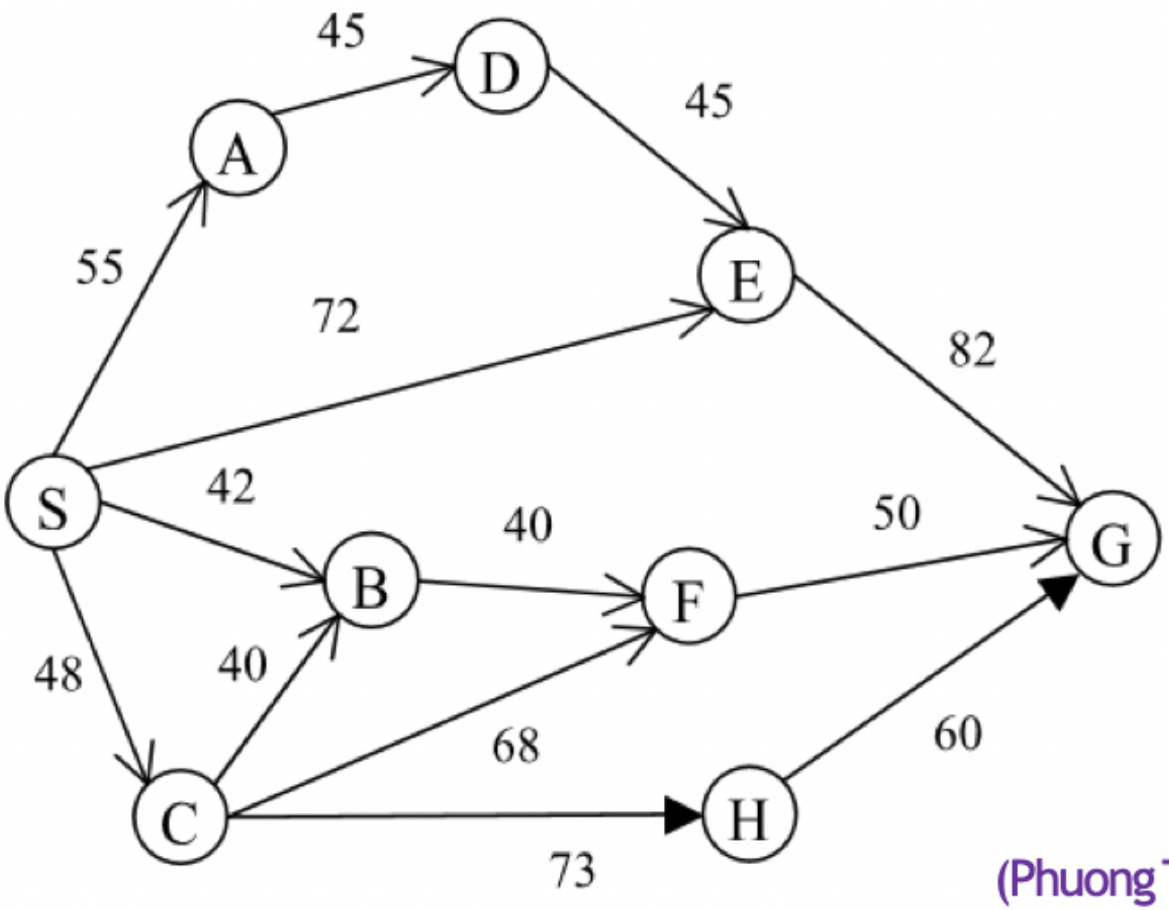
Tính chất của BFS

- Đầy đủ: Có (nếu b là hữu hạn)
- Thời gian: $1 + b + b^2 + b^3 + \ldots + b^d = O(b^d)$
- Bộ nhớ: $O(b^d)$ (lưu tất cả các nút)
- Tối ưu: Có (nếu c=1 với mọi bước)

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 có 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



(Phuong TM, 2016)

Ví du UCS

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

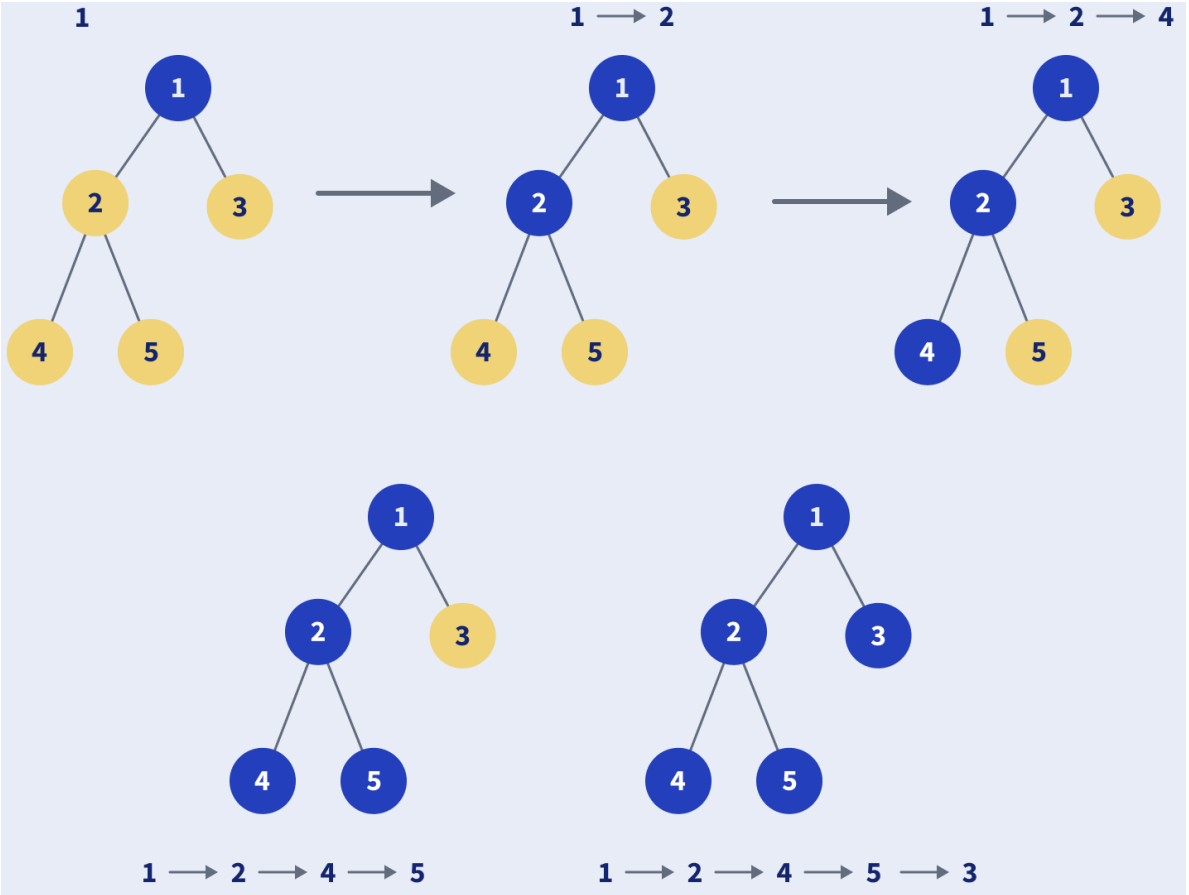
Đường đi: $G \leftarrow F \leftarrow B \leftarrow S$

Xử lý nút lặp

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 chờ)

Tìm kiếm theo chiều sâu - DFS

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



Thuật toán DFS

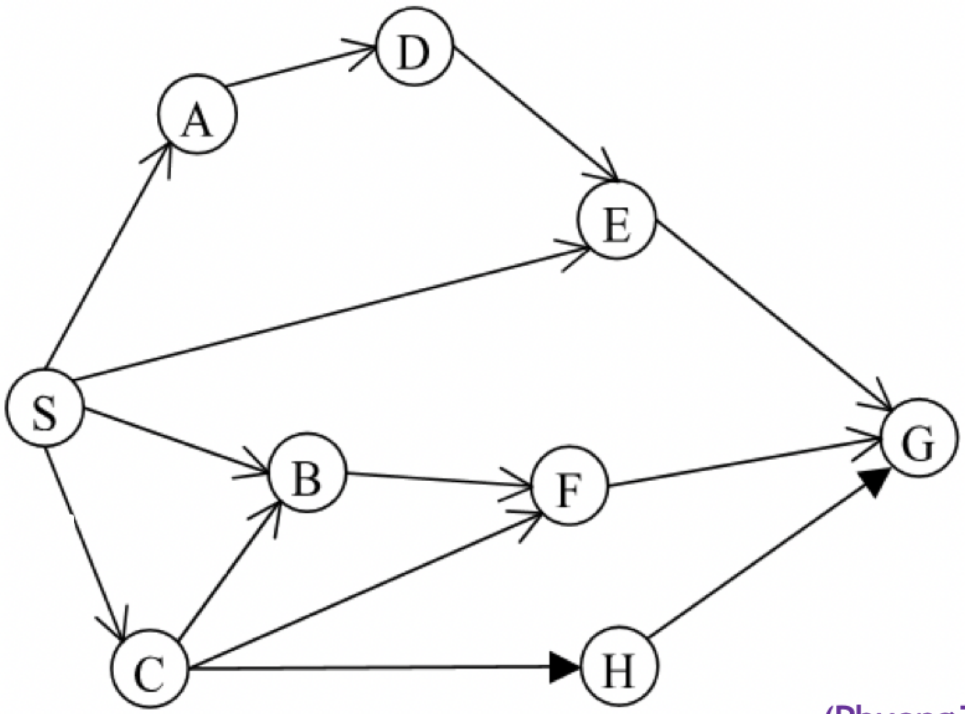
DFS$(\mathrm{Q}, \mathrm{S}, \mathrm{G}, \mathrm{P})$
(Q: không gian trạng thái; S: trạng thái bắt đầu; G: trạng thái đích; P: hành động)
Đầu vào: Bài toán tìm kiếm
Đầu ra: Trạng thái đích
Khởi tạo: $\mathrm{O} \leftarrow \mathrm{S}$ (O: danh sách các nút mở)
while$(\mathrm{O} \neq \emptyset)$ **do**:

1. Lấy nút đầu tiên n khỏi O
2. **if** $n \in \mathrm{G}$, **return** (đường đi tới n)
3. thêm $\mathrm{P}(n)$ vào đầu O (Sử dụng cấu trúc ngăn xếp LIFO)

return: Không tìm được đường đi.

Ví dụ DFS

Tìm đường đi từ nút S tới nút G sử dụng thuật toán DFS



(Phuong TM, 2016)

Ví dụ DFS

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.
- 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.
- Tối ưu: Không

Xử lý nút lặp

- Với một nút đã được duyệt rồi thì sẽ không đưa vào ngăn xếp nữa.
- Với các nút đang trong tập biên, có thể đưa các nút lặp vào ngăn xếp. 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.

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.

Thuật toán IDS

IDS$(\mathrm{Q}, \mathrm{S}, \mathrm{G}, \mathrm{P})$
(Q: không gian trạng thái; S: trạng thái bắt đầu; G: trạng thái đích; P: hành động)
Đầu vào: Bài toán tìm kiếm
Đầu ra: Trạng thái đích
Khởi tạo: $\mathrm{O} \leftarrow \mathrm{S}$ (O: danh sách các nút mở)
$c = 0$ là độ sâu hiện thời
1. **while**$(\mathrm{O} \neq \emptyset)$ **do**:
   a. Lấy nút đầu tiên n khỏi O
   b. **if** $n \in \mathrm{G}$, **return** (đường đi tới n)
   c. **if** $depth(n) \leq c$, **then** thêm $P(n)$ vào đầu O
2. $c++; \mathrm{O} \leftarrow \mathrm{S}$

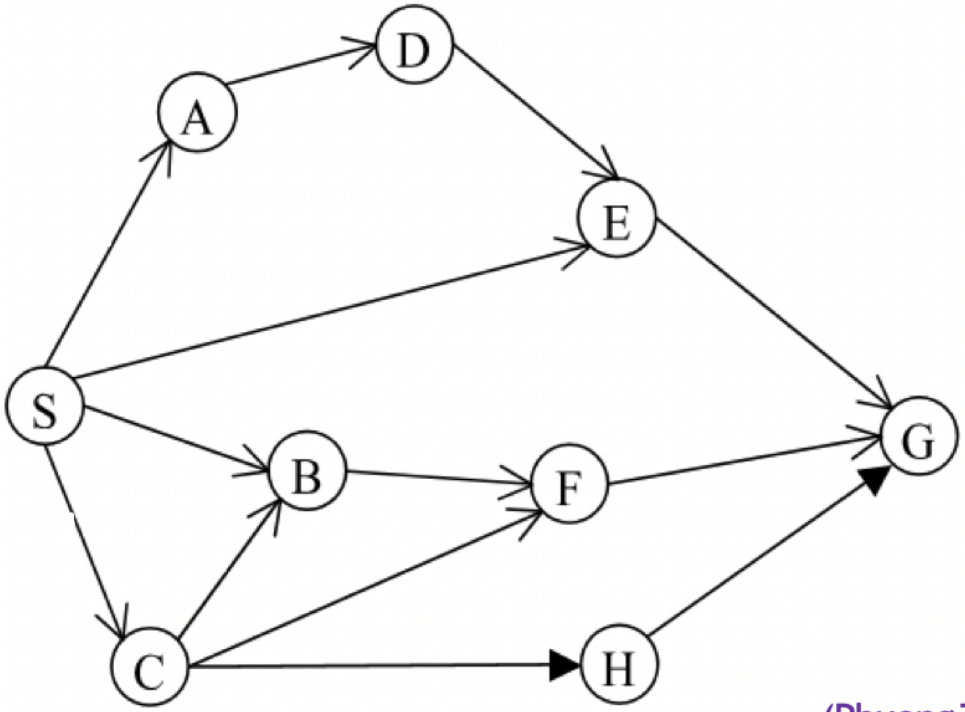
Tính chất của IDS

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

Xử lý nút lặp

- Với các nút đã duyệt hoặc đang chờ được mở rộng, có thể đưa lại các nút lặp vào lại tập biên.

Ví dụ IDS



(Phuong TM, 2016)

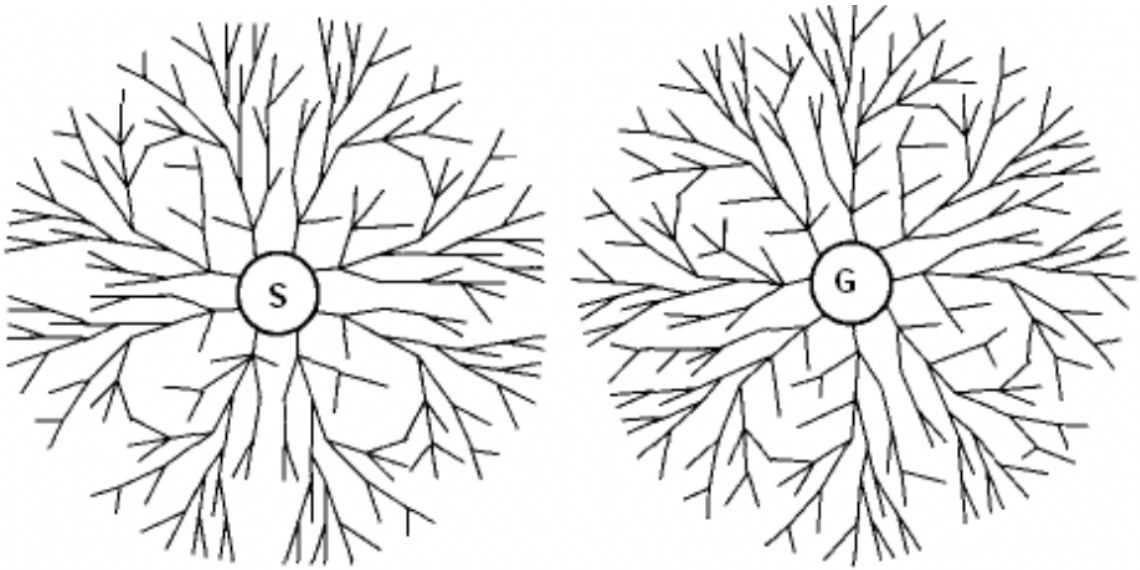
Khi nào đưa nút lặp vào danh sách

- 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

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

Chú ý:

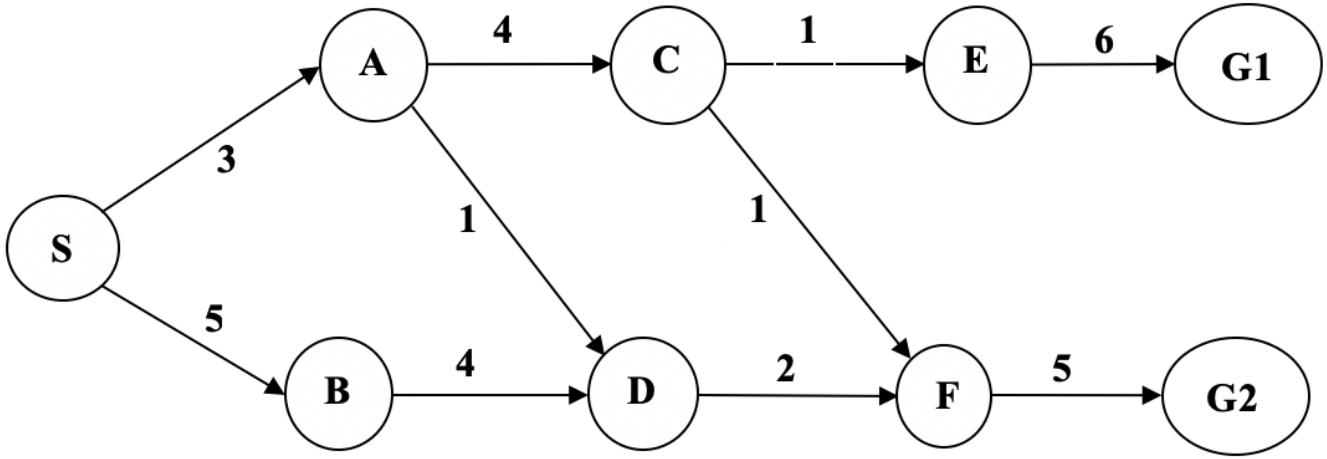
- Cần sử dụng tìm theo chiều rộng. Tìm theo chiều sâu có thể không ra lời giải nếu hai cây tìm kiếm phát triển theo hai nhánh không gặp nhau
- Cần xây dựng các hàm
  - $P(x)$: tập các nút con của x
  - $D(x)$: tập các nút cha của x

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 $(b^{d/2})$. Số lượng nút cần mở rộng của hai cây giảm đáng kể

Bài tập 1

Cho đồ thị như trên hình vẽ, S là nút xuất phát, $G1, G2$ là các nút đích. Các số nằm cạnh cung là giá thành đường đi.



Hãy sử dụng các thuật toán BFS, UCS, DFS, IDS để tìm đường đi từ nút xuất phát tới nút đích.

Tìm kiếm có thông tin

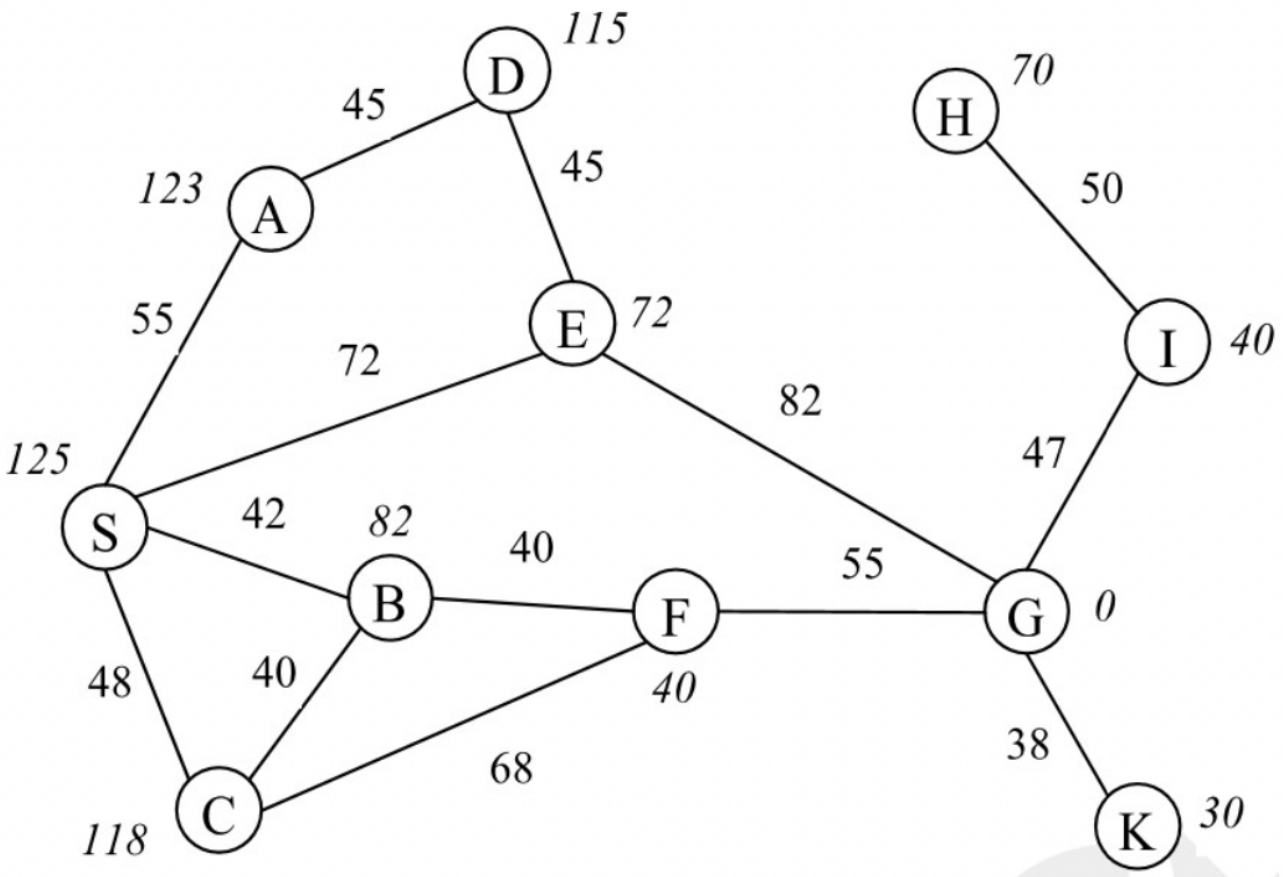
Tìm kiếm mù và tìm kiếm có thông tin

- Tìm kiếm mù
  - Mở rộng các nút tìm kiếm theo một quy luật có trước, không dựa vào thông tin hỗ trợ của bài toán.
  - Di chuyển trong không gian trạng thái không có định hướng, phải xem xét nhiều trạng thái.
  - Không phù hợp trong các bài toán có không gian trạng thái lớn.
- Tìm kiếm có thông tin
  - Sử dụng thông tin phụ từ bài toán để định hướng tìm kiếm
  - Sử dụng một hàm $f(n)$ đánh giá độ "tốt" tiềm năng của nút n, từ đó chọn nút tốt nhất để mở rộng trước:
    - Tìm kiếm tốt nhất đầu tiên (best-first search)
    - Xây dựng hàm $f(n)$ thế nào?

Các thuật toán tìm kiếm có thông tin

- Thuật toán tìm kiếm tham lam (greedy search)
- Thuật toán A*
- Thuật toán A* sâu dần (IDA*)

Ví dụ đồ thị trong bài toán tìm kiếm



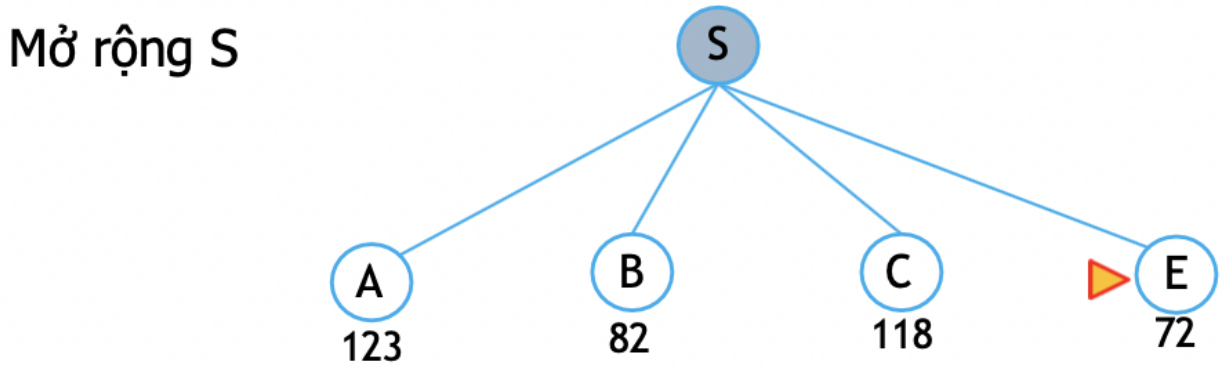
Tìm kiếm tham lam

Phương pháp: mở rộng nút có giá thành đường đi tới đích nhỏ nhất trước:

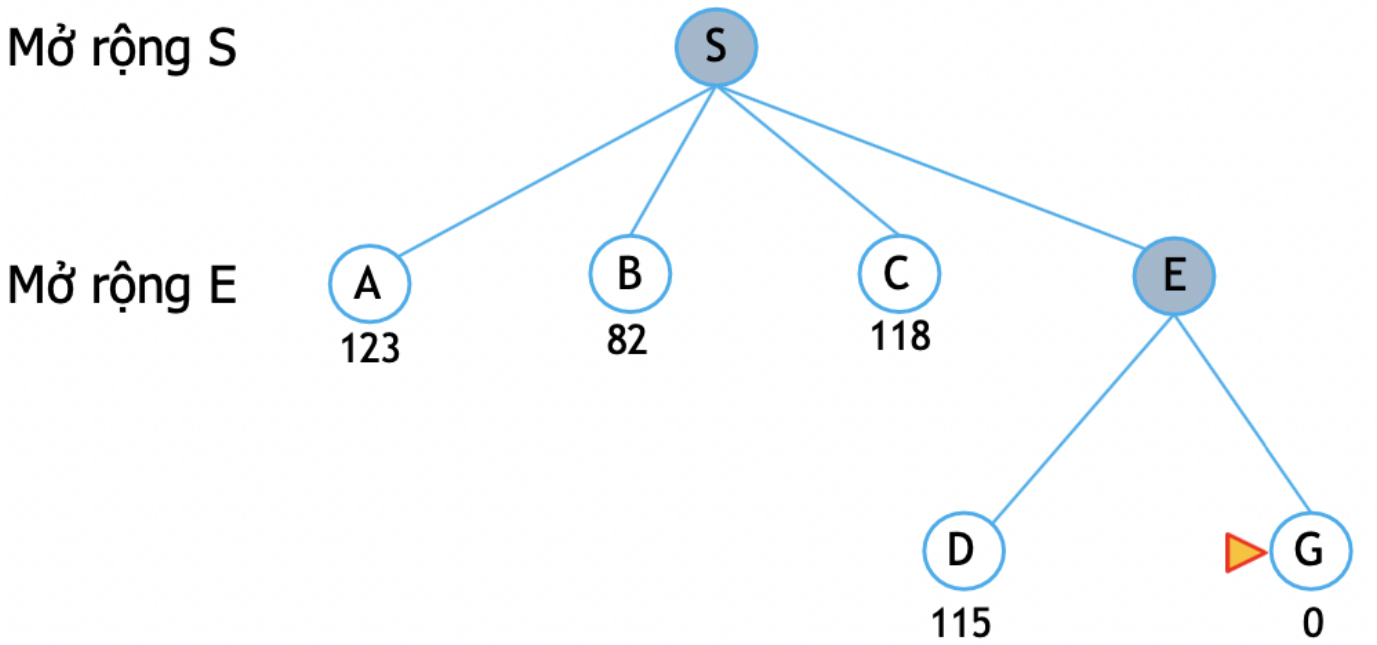
- $f(n) = h(n)$: hàm heuristic ước lượng giá thành đường đi từ n tới đích.
- Vú dụ: $h(n) =$ đường chim bay từ n tới đích.

"Tham lam": Chọn nút trông có vẻ tốt nhất để mở rộng, không quan tâm tới tương lai.

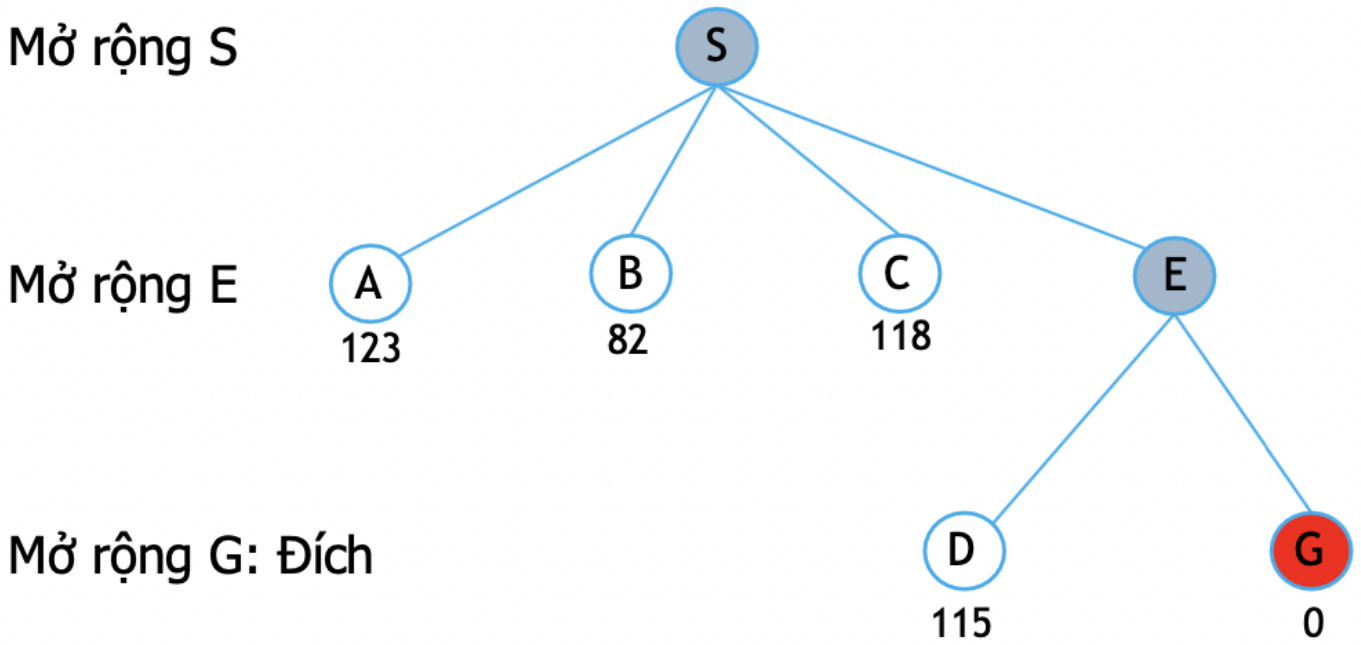
Ví dụ tìm kiếm tham lam



Ví dụ tìm kiếm tham lam



Ví dụ tìm kiếm tham lam



Tính chất của tìm kiếm tham lam

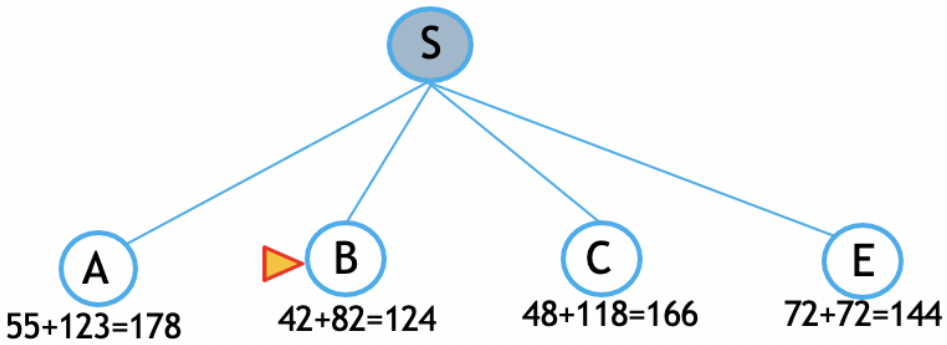
- Đầy đủ: Không (có thể tạo thành vòng lặp, hoặc có nhánh gồm vô hạn nút có giá trị hàm h nhỏ nhưng không dẫn tới đích).
- Thời gian:
  - $O(b^m)$
  - Nếu hàm heuristic tốt, thuật toán có thể sẽ nhanh hơn rất nhiều.
- Bộ nhớ: $O(b^m)$: lưu tất cả các nút trong bộ nhớ. Nếu hàm heuristic tốt, số nút cần lưu có thể giảm đi rất nhiều.
- Tối ưu: Không

Thuật toán A*

- Khắc phục nhược điểm của tìm kiếm tham lam
  - Tham lam: chỉ quan tâm tới đường đi tới đích.
  - A*: quan tâm cả đường đi từ điểm xuất phát tới nút đang xét. Không mở rộng đường đi có giá thành lớn
- Phương pháp: $f(n) = g(n) + h(n)$:
  - $g(n)$: giá thành đường đi từ điểm xuất phát tới nút n
  - $h(n)$: hàm heuristic ước lượng giá thành đường đi từ n tới đích
  - $f(n)$: ước lượng giá thành đường đi từ điểm xuất phát, qua n, tới đích

Ví dụ thuật toán A* (1/4)

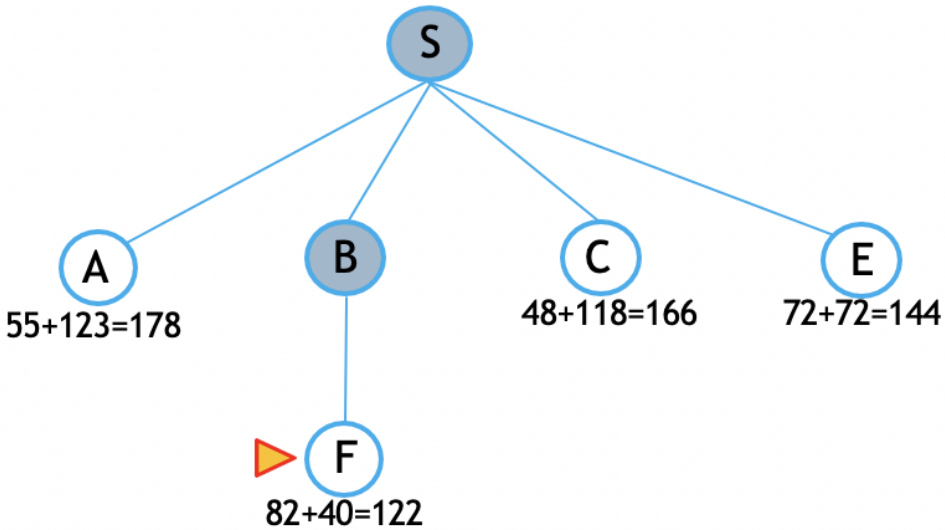
Mở rộng S



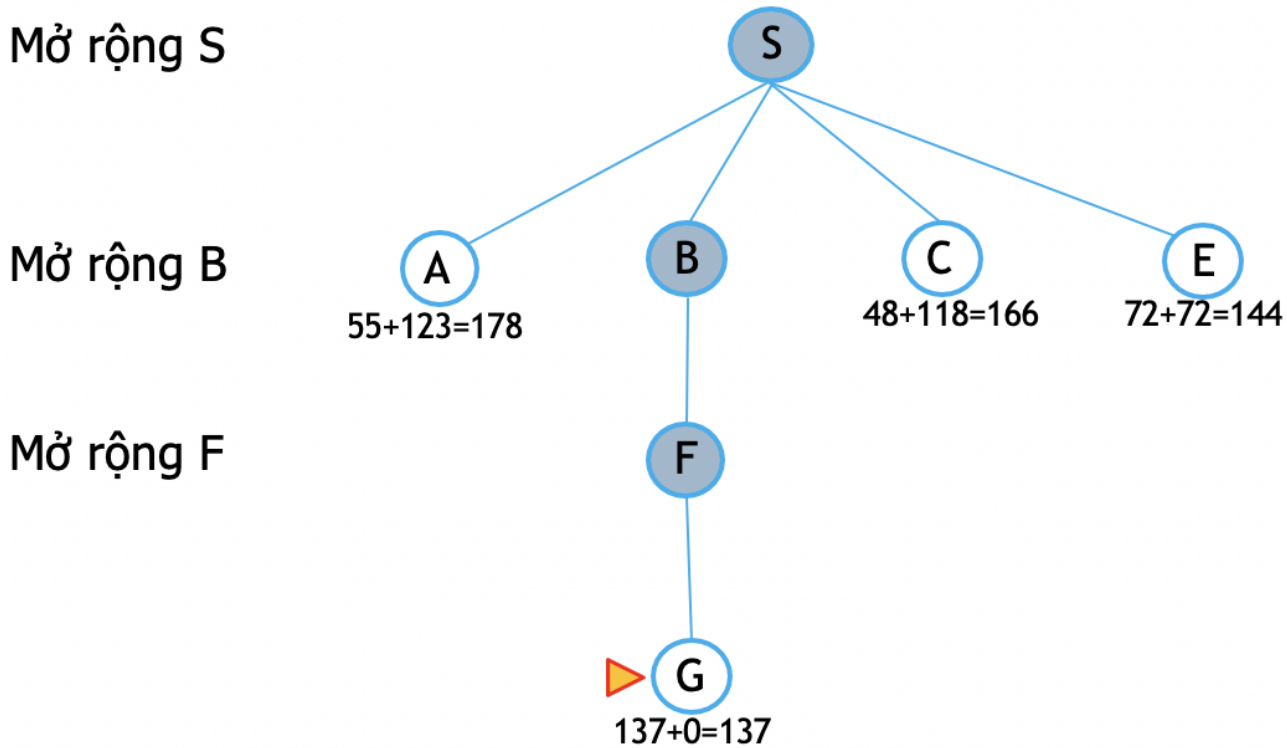
Ví dụ thuật toán A* (2/4)

Mở rộng S

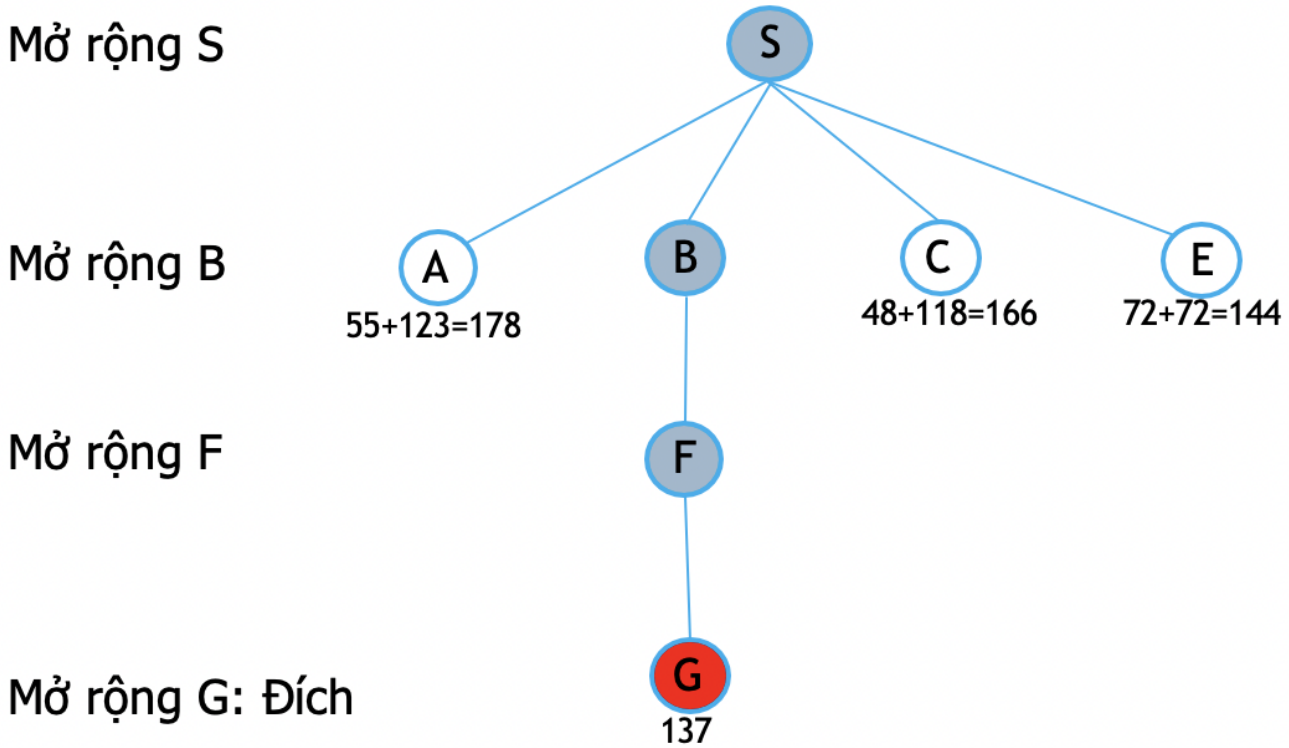
Mở rộng B



Ví dụ thuật toán A* (3/4)



Ví dụ thuật toán A* (4/4)



Thuật toán A*

*A**$(\mathrm{Q}, \mathrm{S}, \mathrm{G}, \mathrm{P}, \mathrm{c}, \mathrm{h})$
(Q: không gian trạng thái; S: trạng thái bắt đầu; G: trạng thái đích; P: hành động); c: giá; h: giá trị hàm heuristic
Đầu vào: Bài toán tìm kiếm, hàm heuristic h
Đầu ra: Trạng thái đích
Khởi tạo: $\mathrm{O} \leftarrow \mathrm{S}$ (O: danh sách các nút mở)
while$(\mathrm{O} \neq \emptyset)$ **do**:

1. Lấy nút n có $f(n)$ là nhỏ nhất khỏi O
2. **if** $n \in \mathrm{G}$, **return** (đường đi tới n)
3. Với mọi $m \in \mathrm{P}(n)$:
   a) $g(m) = g(n) + c(n, m)$
   b) $f(m) = g(m) + h(m)$
   c) thêm m vào O cùng giá trị $f(m)$

return: Không tìm được đường đi.

Tính chất của thuật toán A*

- Đầy đủ: Có (trừ khi có vô số nút n với $(n) \neq f(G)$
- Thời gian:
  - $O(b^m)$
  - Nếu hàm heuristic tốt, thuật toán có thể sẽ nhanh hơn rất nhiều.
- Bộ nhớ: $O(b^m)$: lưu tất cả các nút trong bộ nhớ. Nếu hàm heuristic tốt, số nút cần lưu có thể giảm đi rất nhiều.
- Tối ưu: Có (nếu hàm heuristic là chấp nhận được)

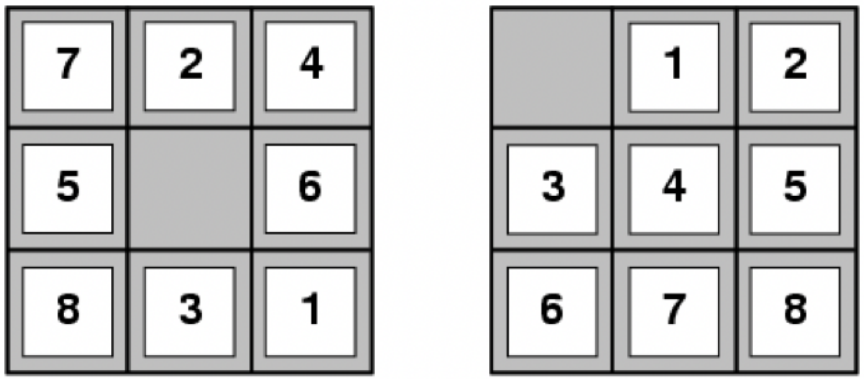
Tính tối ưu của A*

- Hàm heuristic chấp nhận được:
  - Mọi nút n thì $h(n) \leq h*(n)$, với $h*(n)$ là giá thành thực để đi từ n tới đích
  - Ví dụ: hàm heuristic đo khoảng cách đường chim bay là chấp nhận được.
- **Định lý**: Thuật toán A* sẽ cho kết quả tối ưu nếu $h(n)$ là hàm chấp nhận được

Các hàm heuristic

- Các hàm heuristic được xây dựng tùy vào từng bài toán cụ thể
  - Một bài toán có thể có nhiều hàm heuristic
  - Chất lượng hàm heuristic ảnh hưởng rất nhiều tới quá trình tìm kiếm
- Hàm heuristic trội
  - Nếu $h_1(n)$ và $h_2(n)$ là 2 hàm heuristic chấp nhận được thỏa mãn $h_1(n) \leq h_2(n)$ với mọi nút n, thì h_2 trội hơn (tốt hơn) h_1

Ví dụ hàm heuristic



- $h_1(n)$: số ô đặt sai chỗ
  - $h_1(S) = 8$
- $h_2(n)$: tổng khoảng cách Manhattan
  - $h_2(S) = 3 + 1 + 2 + 2 + 2 + 3 + 3 + 2 = 18$

Thuật toán A* sâu dần

- **Mục tiêu:** giải quyết vấn đề bộ nhớ trong tìm kiếm A*
- **Phương pháp:** lặp lại việc tìm kiếm theo chiều sâu trên các cây tìm kiếm con có giá trị hàm $f(n)$ không lớn hơn một ngưỡng
  - Giá trị ngưỡng được tăng dần sau mỗi vòng lặp, để mỗi vòng lặp có thể xét thêm các nút mới

Thuật toán IDA*

*IDA**$(\mathrm{Q}, \mathrm{S}, \mathrm{G}, \mathrm{P}, \mathrm{c}, \mathrm{h})$
Đầu vào: Bài toán tìm kiếm, hàm heuristic h
Đầu ra: Trạng thái đích
Khởi tạo: $\mathrm{O} \leftarrow \mathrm{S}$ (O: danh sách các nút mở);
giá trị $i \leftarrow 0$ là ngưỡng cho hàm f
while(1) **do**
1. while$(\mathrm{O} \neq \emptyset)$ **do**:

a) Lấy nút n từ đầu O

b) **if** $n \in \mathrm{G}$, **return** (đường đi tới n)

c) Với mọi $m \in \mathrm{P}(n)$:

i) $g(m) = g(n) + c(n, m)$

ii) $f(m) = g(m) + h(m)$

iii) **if** $f(m) \leq i$ **then** thêm m vào đầu O

2. $i \leftarrow i + \beta, O \leftarrow S$

Tính chất của IDA*

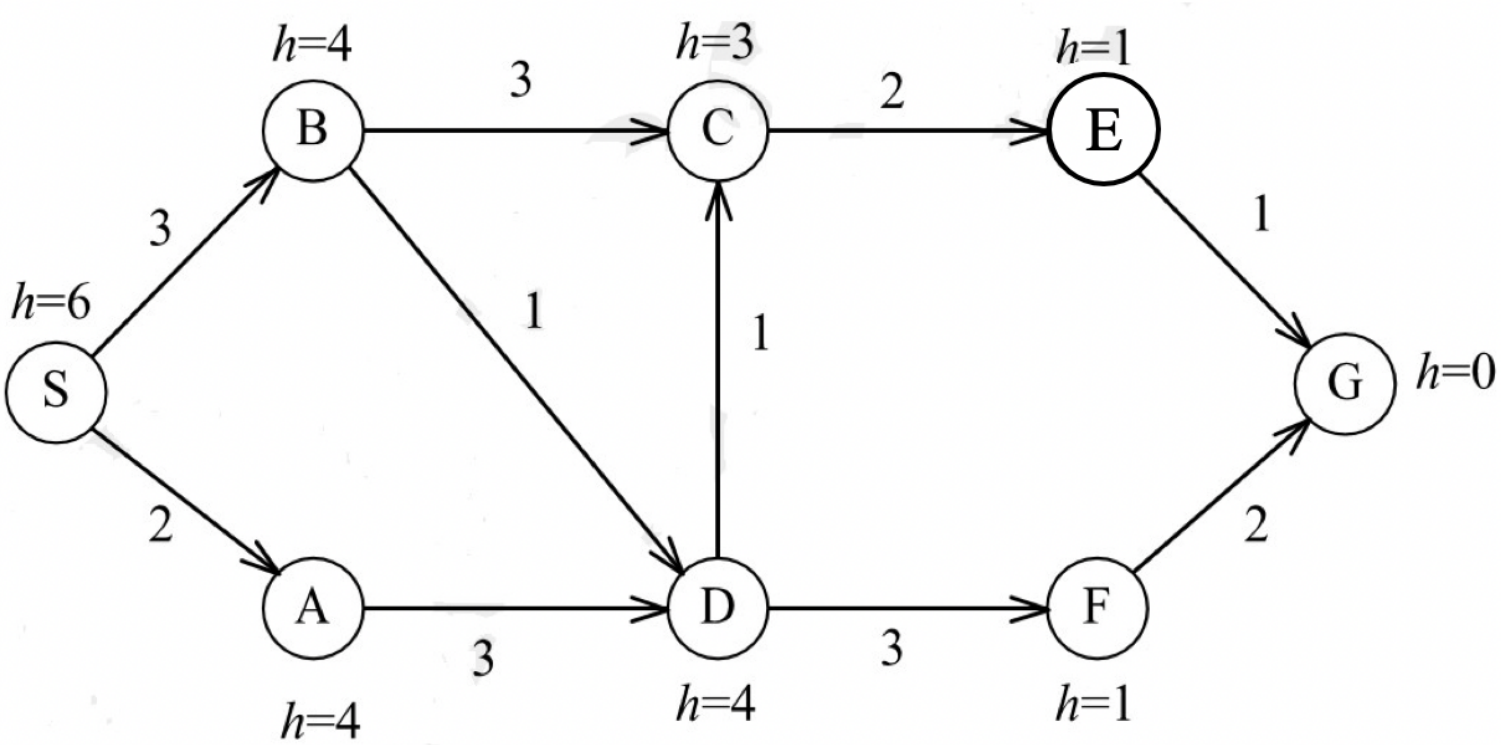
- Đầy đủ: Có
- Thời gian: Độ phức tạp tính toán lớn hơn thuật toán A*
- Bộ nhớ: Yêu cầu bộ nhớ tuyến tính
- Tối ưu: β-tối ưu (giá thành của lời giải tìm được không vượt quá β so với giá thành của lời giải tối ưu)

Khi nào đưa nút lặp vào danh sách?

- Tham lam: Không: Việc đưa vào không làm thay đổi thuật toán (có thể dẫn đến vòng lặp)
- A*: 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)
- IDA*: Có

Bài tập 1

Cho đồ thị như hình vẽ với S là nút xuất phát, G là nút đích.

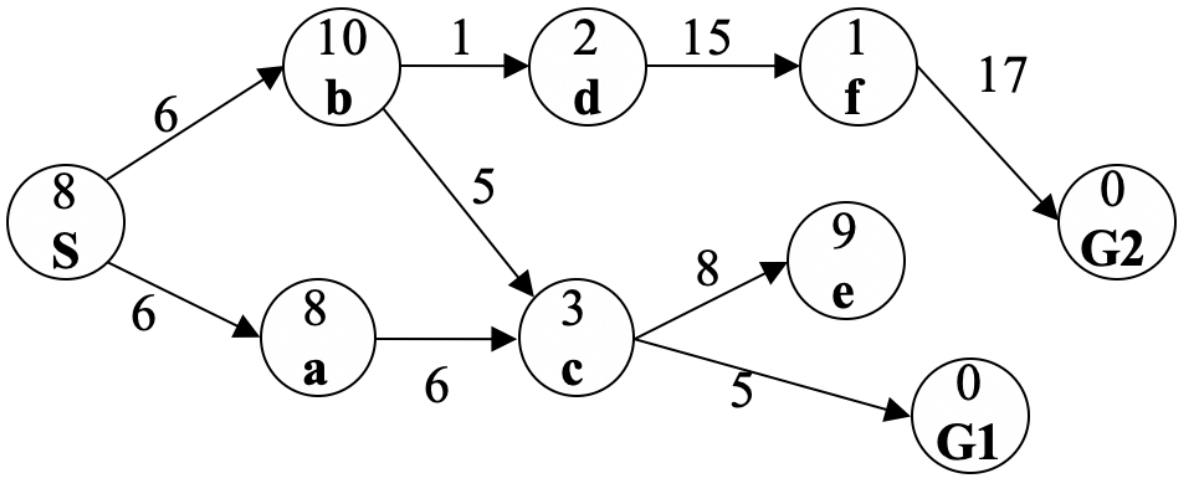


(Phuong TM, 2016)

Tìm đường đi từ S tới G sử dụng thuật toán tham lam, A* và IDA* với giá trị bước nhảy $\beta = 2$

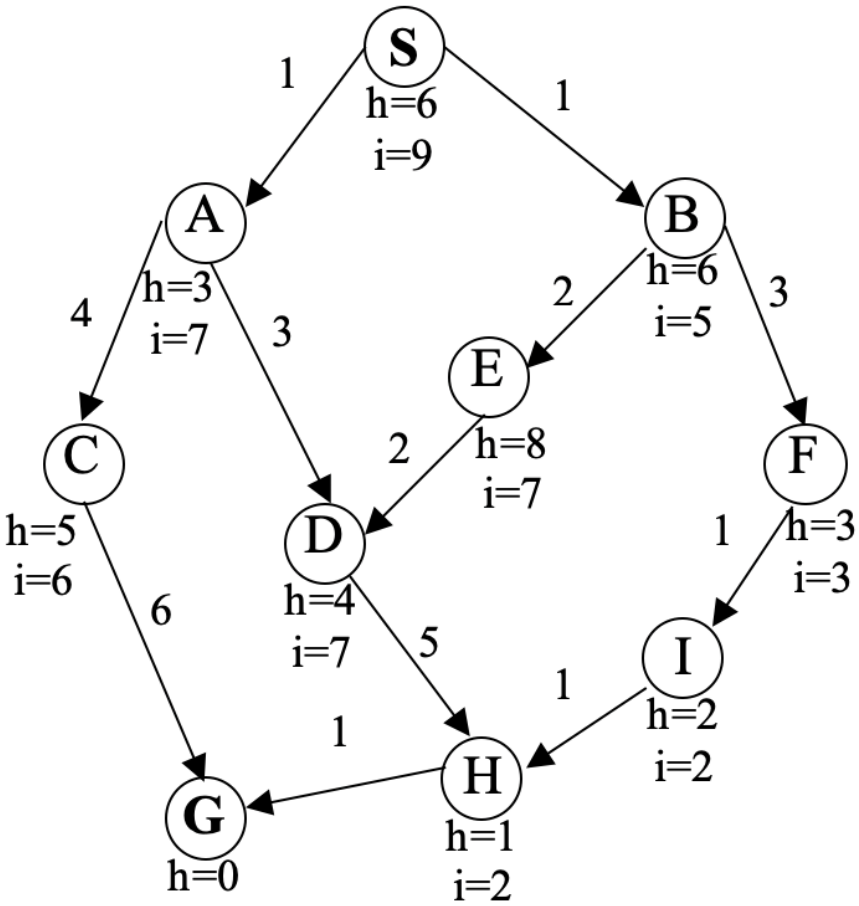
Bài tập 2

Cho đồ thị như hình vẽ với S là nút xuất phát, $G1, G2$ là các nút đích, số nằm trong vòng tròn là giá trị hàm heuristic. Sử dụng thuật toán tìm kiếm A* tìm đường đi từ nút xuất phát tới nút đích?



Bài tập 3

Thuật toán A* được sử dụng tìm đường đi từ S tới G trên đồ thị sau. Giá thành đường đi cho bởi các số bên cạnh mũi tên. Giả sử có hai hàm heuristic h và i được sử dụng (hình vẽ).



Bài tập 3 (Tiếp)

a) Hàm heuristic nào (h,i hoặc cả hai) là hàm chấp nhận được? Giải thích tại sao?
b) Sử dụng hàm heuristic h và thuật toán tìm kiếm A*, tìm đường đi từ S đến G.

Tìm kiếm cục bộ

Nội dung

1. Giới thiệu tìm kiếm cục bộ
2. Thuật toán leo đồi (Hill climbing)
3. Thuật toán tôi thép (Simulated Annealing)

Tìm kiếm cục bộ

- Các thuật toán tìm kiếm đã học (mù hoặc có thông tin) khảo sát không gian tìm kiếm một cách hệ thống theo một số quy tắc nhất định
  - Cần lưu lại thông tin về trạng thái và đường đi đã khảo sát
  - Không thích hợp cho bài toán có không gian trạng thái lớn
- Tìm kiếm cục bộ tại một thời điểm chỉ xem xét trạng thái hiện thời và các trạng thái lân cận
  - Không lưu thông tin về trạng thái và đường đi đã khảo sát
  - Tiết kiệm thời gian và bộ nhớ
  - Có thể áp dụng cho các bài toán có không gian trạng thái lớn
  - Không cho lời giải tối ưu

Bài toán tối ưu hoá tổ hợp (rời rạc)

- Tìm trạng thái tối ưu hoặc tổ hợp tối ưu trong không gian rời rạc các trạng thái
  - Không quan tâm tới đường đi
- Không gian trạng thái rất lớn
  - Không thể sử dụng các phương pháp tìm kiếm đã học để duyệt tất cả các trạng thái
- Không tồn tại thuật toán cho phép tìm lời giải tốt nhất với độ phức tạp tính toán nhỏ
  - Có thể chấp nhận lời giải tương đối tốt
- Ví dụ: Bài toán lập kế hoạch, lập thời khóa biểu, bài toán một triệu con hậu, ...

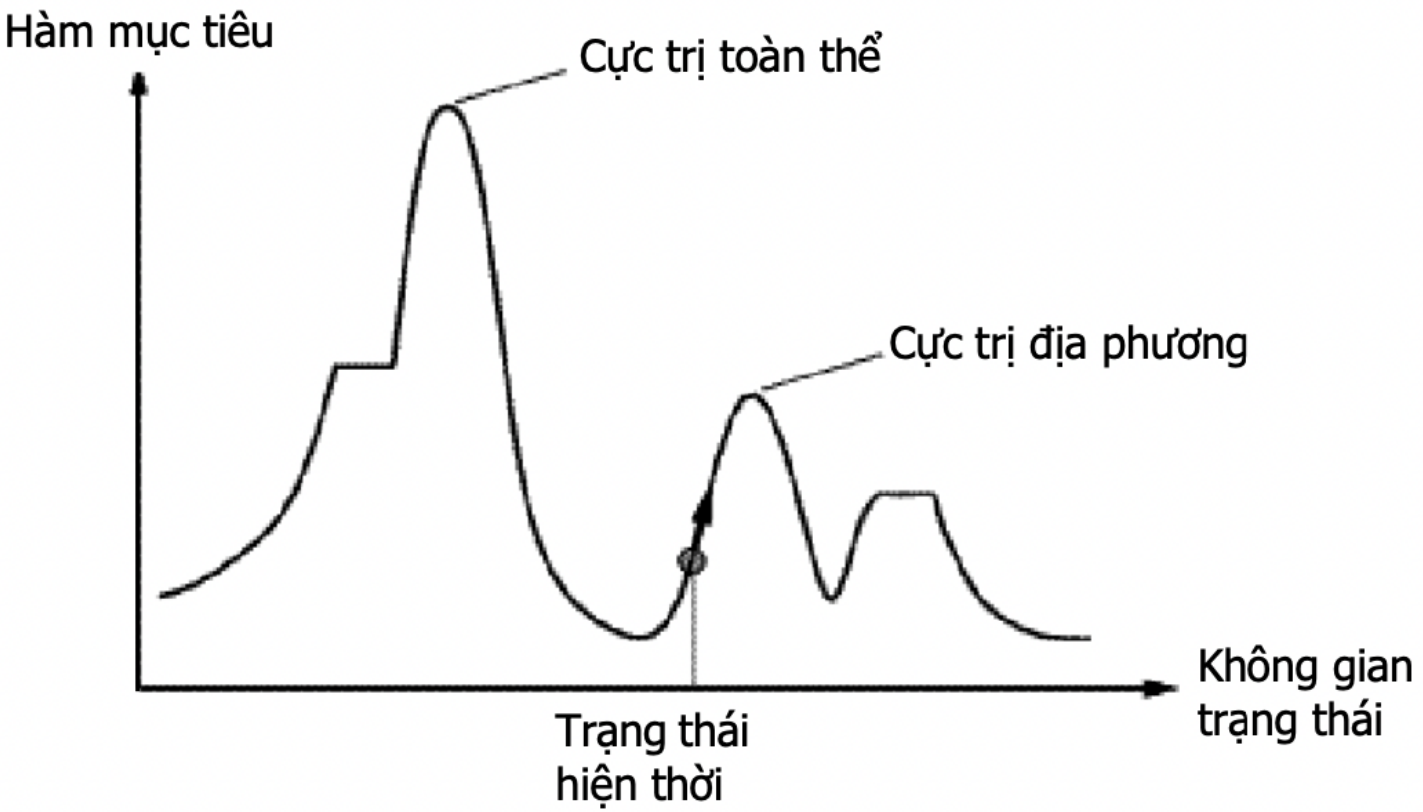
Tìm kiếm cục bộ: tư tưởng

- Khác với bài toán tìm kiếm thông thường, tìm kiếm cục bộ chỉ quan trọng trạng thái đích (trạng thái tốt nhất), không quan trọng đường đi
  - Mỗi trạng thái tương ứng với một lời giải (chưa tối ưu)
- Cải thiện dần (iterative improvement) lời giải bằng cách xuất phát từ một trạng thái, sau đó thay đổi để chuyển sang trạng thái có hàm mục tiêu tốt hơn
- Thay đổi trạng thái bằng cách thực hiện các chuyển động
  - Trạng thái nhận được từ một trạng thái n bằng cách thực hiện các chuyển động gọi là hàng xóm của n

Phát biểu bài toán tìm kiếm cục bộ

- Không gian trạng thái X
- Hàm mục tiêu $Obj : X \to R$
- Tập chuyển động để sinh ra hàng xóm: $N(x)$ là tập các hàm xóm của x
- Yêu cầu: Tìm trạng thái x^* sao cho $Obj(x^*)$ là lớn nhất hoặc nhỏ nhất

Minh hoạ tìm kiếm cục bộ



Thuật toán leo đồi: tư tưởng

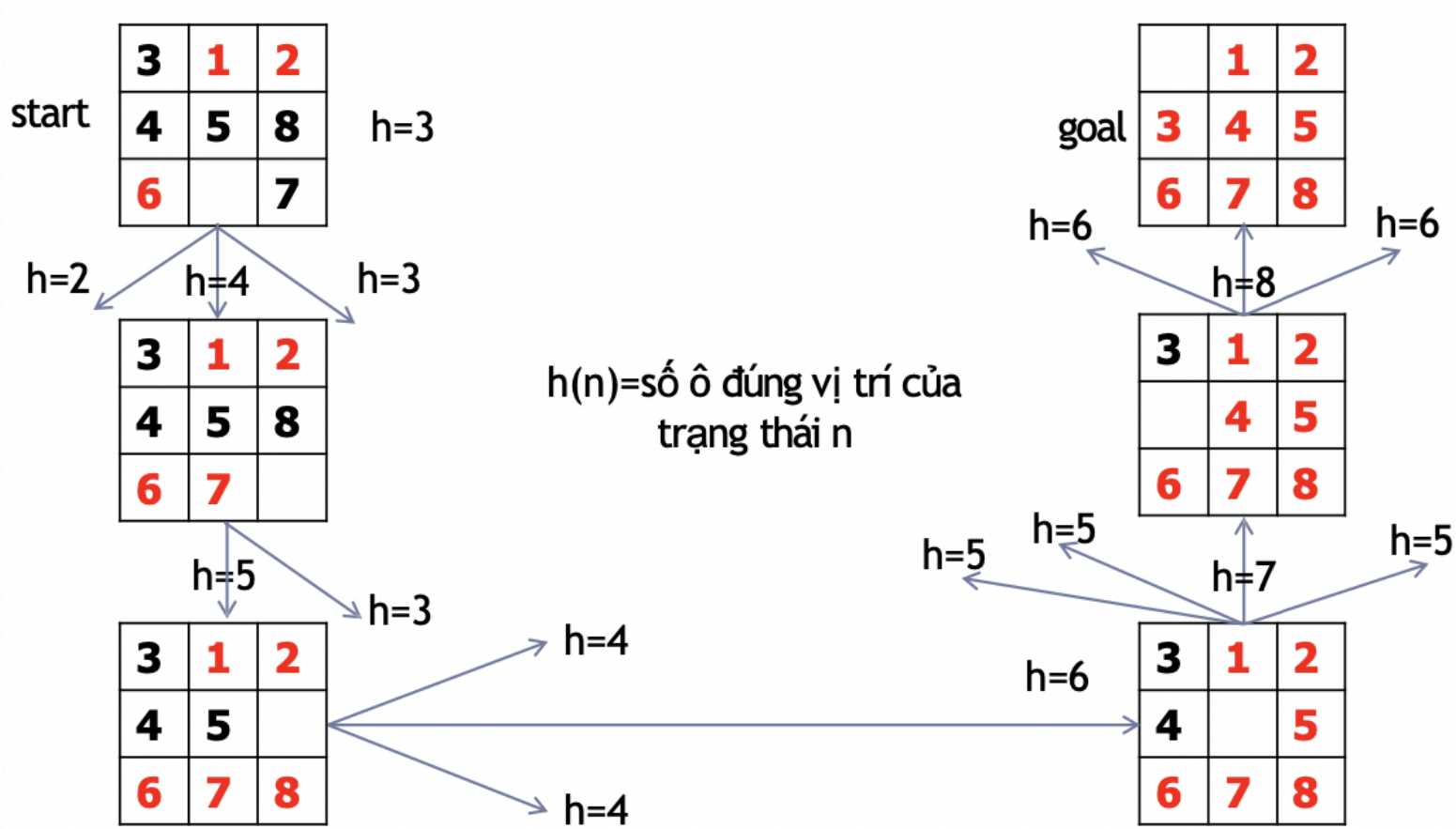
- **Leo đồi**: là tên chung của một họ thuật toán cùng nguyên lý
- **Cách thức**: Từ trạng thái hiện tại, xem xét tập hàng xóm, di chuyển sang trạng thái tốt hơn
  - Chọn trạng thái hàng xóm để di chuyển thế nào?
- **Trạng thái đích**: Thuật toán dừng lại khi không có trạng thái hàng xóm nào tốt hơn
  - Thuật toán có thể tìm được cực trị hoặc cực trị địa phương

Di chuyển sang trạng thái tốt nhất

Đầu vào: Bài toán tối ưu tổ hợp
Đầu ra: Trạng thái với hàm mục tiêu lớn nhất (hoặc cực đại địa phương)

1. Chọn ngẫu nhiên trạng thái x
2. Gọi Y là tập các trạng thái hàng xóm của x
3. **if** $\forall y_i \in Y : Obj(y_i) < Obj(x)$
   return x
4. $x \leftarrow y_i$ trong đó $i = argmax_i(Obj(y_i))$
5. **Go to** 2

Ví dụ leo đồi


start
h=3
h=2
h=4
h=3
h=5
h=3
h=4
h=4
h(n)=số ô đúng vị trí của trạng thái n
goal
h=6
h=6
h=8
h=5
h=5
h=7
h=5
h=6

Tính chất thuật toán leo đồi

- Đơn giản, dễ lập trình
- Không tốn bộ nhớ (không phải ghi nhớ các trạng thái)
- Dễ bị lời giải tối ưu cục bộ (cực trị địa phương)
- Việc lựa chọn chuyển động rất quan trọng, không có quy tắc chung
  - Nếu có quá nhiều chuyển động: Sinh ra quá nhiều hàng xóm, Mất nhiều thời gian lựa chọn phương án tốt nhất
  - Nếu quá ít chuyển động: Rất dễ bị cực trị địa phương

Leo đồi ngẫu nhiên: tư tưởng

- Là một phiên bản khác của thuật toán leo đồi
- Lựa chọn ngẫu nhiên một trạng thái hàng xóm
  - Chuyển sang trạng thái hàng xóm nếu trạng thái này tốt hơn
  - Nếu không tốt hơn lại chọn ngẫu nhiên một hàng xóm khác
- Kết thúc khi nào hết kiên nhẫn
  - Số hàng xóm mà thuật toán xem xét trong mỗi bước lặp hoặc trong toàn bộ thuật toán

Thuật toán leo đồi ngẫu nhiên

1. Chọn ngẫu nhiên trạng thái x
2. Gọi Y là tập các trạng thái hàng xóm của x
3. Chọn ngẫu nhiên $y_i \in Y$
4. **if** $Obj(y_i) > Obj(x)$ **then** $x \leftarrow y_i$
5. **Go to** 2 nếu chưa hết kiên nhẫn

Vấn đề: Chọn tiêu chuẩn kết thúc thế nào?

Một số tính chất

- Trường hợp mỗi trạng thái có nhiều láng giềng
  - Leo đồi ngẫu nhiên thường cho kết quả nhanh hơn, và ít gặp cực trị địa phương hơn
- Với những không gian trạng thái có ít cực trị địa phương
  - Các thuật toán leo đồi thường tìm được lời giải khá nhanh
- Với những không gian phức tạp
  - Các thuật toán leo đồi thường chỉ tìm được cực trị địa phương
  - Bằng cách thực hiện nhiều lần với trạng thái xuất phát ngẫu nhiên, leo đồi thường tìm được cực trị địa phương khá tốt

Thuật toán tôi thép

- Là phiên bản khái quát của thuật toán leo đồi
- Mục tiêu: Giải quyết phần nào vấn đề cực trị địa phương trong các thuật toán leo đồi
- Nguyên tắc chung: chấp nhận những trạng thái kém hơn trạng thái hiện thời với một xác suất p
  - Chọn xác suất p thế nào?

Lựa chọn p

Nguyên tắc: không chọn p cố định, giá trị p phụ thuộc hai yếu tố

- Nếu trạng thái mới kém hơn nhiều so với trạng thái hiện thời, thì p phải giảm đi: Xác suất chấp nhận trạng thái tỉ lệ nghịch với độ kém của trạng thái.
- Theo thời gian, giá trị của p phải giảm dần
  - Khi mới bắt đầu, thuật toán chưa ở vùng trạng tốt, do vậy chấp nhận thay đổi lớn
  - Theo thời gian, thuật toán chuyển sang vùng trạng thái tốt, do vậy cần hạn chế thay đổi

Thuật toán tôi thép

SA (X, Obj, N, m, x, C) // Obj càng nhỏ càng tốt
Đầu vào: Số bước lặp m; Trạng thái bắt đầu x; Sơ đồ làm lạnh C
Đầu ra: Trạng thái tốt nhất $x*$
Khởi tạo: $x^* = x$
For $i = 1$ **to** m:

1. Chọn ngẫu nhiên $y \in N(x)$
   a) Tính $\Delta(x, y) = Obj(y) - Obj(x)$
   b) **if** $\Delta(x, y) < 0$ **then** $p = 1$
   c) **else** $p = e - \frac{\Delta(x,y)}{T}$
   d) **if** $rand[0, 1] < p$ **then** $x \leftarrow y$ // Gán y cho x với xác suất p.
   if $Obj(x) < Obj(x*)$ **then** $x^* \leftarrow x$
2. Giảm T theo sơ đồ C

return x^* // x^* là tốt nhất trong những trạng thái đã xem xét

Sơ đồ làm lạnh C

- $T_t = T_0 * \alpha^{tk}$. Trong đó:
  - $T_0 > 0$
  - $\alpha \in (0, 1)$
  - $1 \leq t \leq m$
  - $1 < k < m$
- Ý nghĩa
  - t càng tăng thì T càng nhỏ, p càng nhỏ
  - T lớn: chấp nhận bất cứ trạng thái nào
  - T nhỏ: không chấp nhận trạng thái kém hơn

Tính chất thuật toán tôi thép

- Không có cơ sở lý thuyết rõ ràng
- Thường cho kết quả tốt hơn leo đồi: Do ít bị cực trị địa phương hơn
- Việc lựa chọn tham số phụ thuộc vào bài toán cụ thể