Tìm kiếm heuristic
Tìm kiếm A*

Tô Hoài Việt
Khoa Công nghệ Thông tin
Đại học Khoa học Tự nhiên TPHCM
thviet@fit.hcmuns.edu.vn

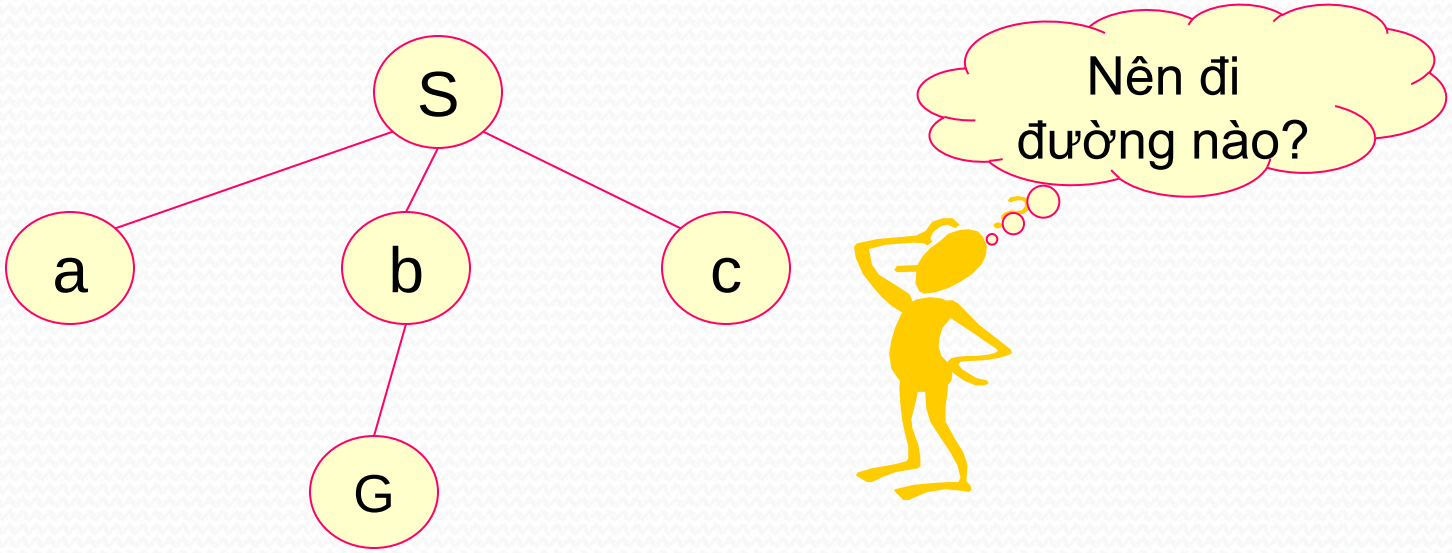
Ref: http://www.cs.cmu.edu/~awm/tutorials

Tổng quan

- Tìm kiếm heuristic Tối ưu kiểu “Tham lam” (“Greedy Best-First Search)
- Những điểm không thích hợp của tìm kiếm heuristic “Tham lam”.
- Mẹo: tính luôn chi phí đi đến trạng thái hiện tại.
- Việc tìm kiếm kết thúc khi nào?
- Heuristic chấp nhận được
- Tìm kiếm A* là đầy đủ
- Tìm kiếm A* luôn dừng
- Khuyết điểm của A*
- Tiết kiệm nhiều bộ nhớ với IDA* (Iterative Deepening A*)

Tìm kiếm Heuristic

- Các phương pháp tìm kiếm mù (blind search): thông tin về trạng thái đích không đóng vai trò trong việc tìm kiếm.



- Có thể sử dụng ước lượng khoảng cách đến đích giữa các trạng thái để tìm đường đi?

Tìm kiếm Heuristic

Giả sử ngoài việc đặc tả tìm kiếm chuẩn ta cũng có một *heuristic*.

Một hàm heuristic ánh xạ một trạng thái thành một ước lượng về chi phí đến đích từ trạng thái đó.

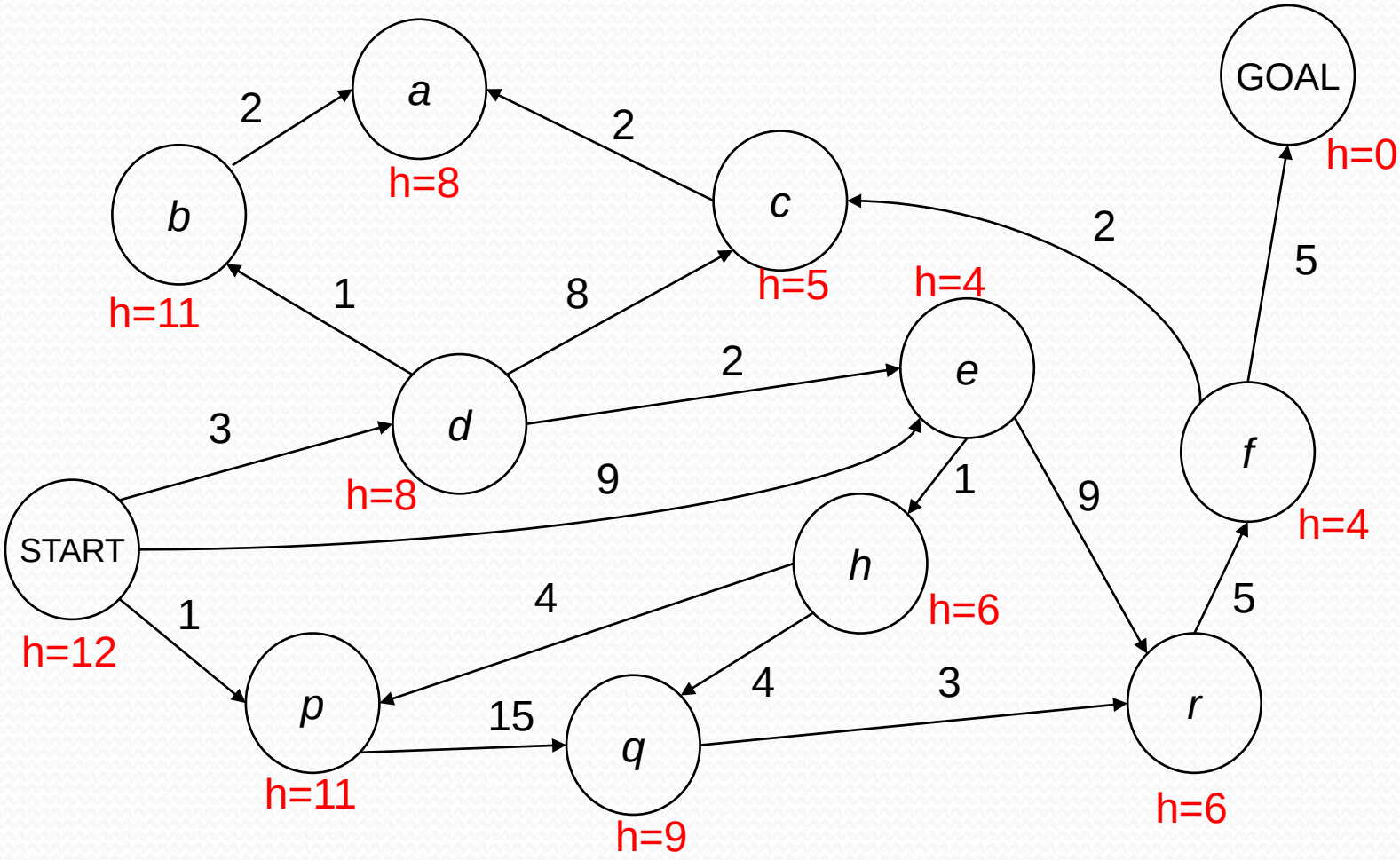
Bạn có thể nghĩ ra ví dụ về heuristics?

VD. đối với bài toán 8-puzzle?

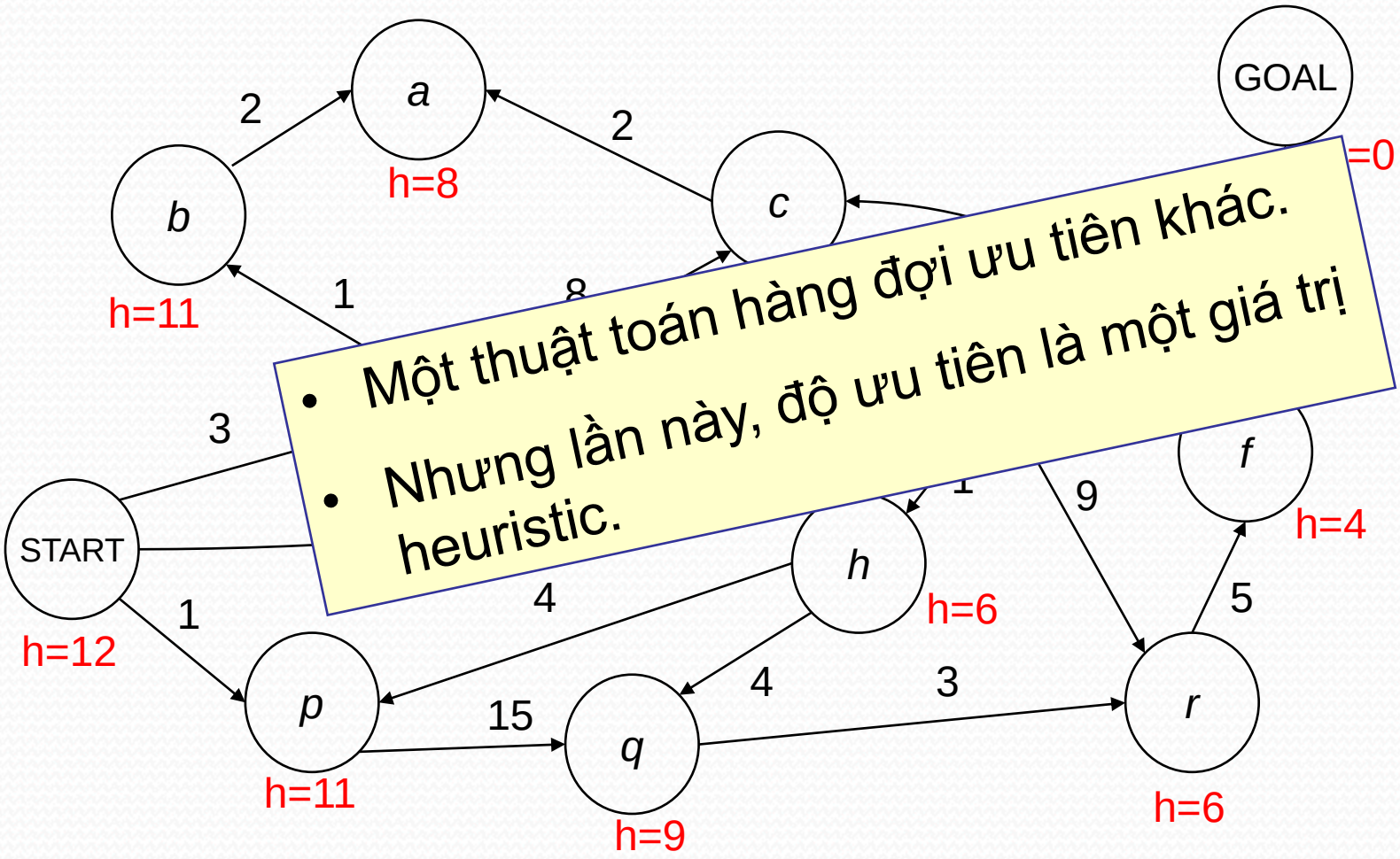
VD. để lập đường đi trong ma trận?

Ký hiệu heuristic bằng một hàm $h(s)$ tính các trạng thái thành giá trị chi phí.

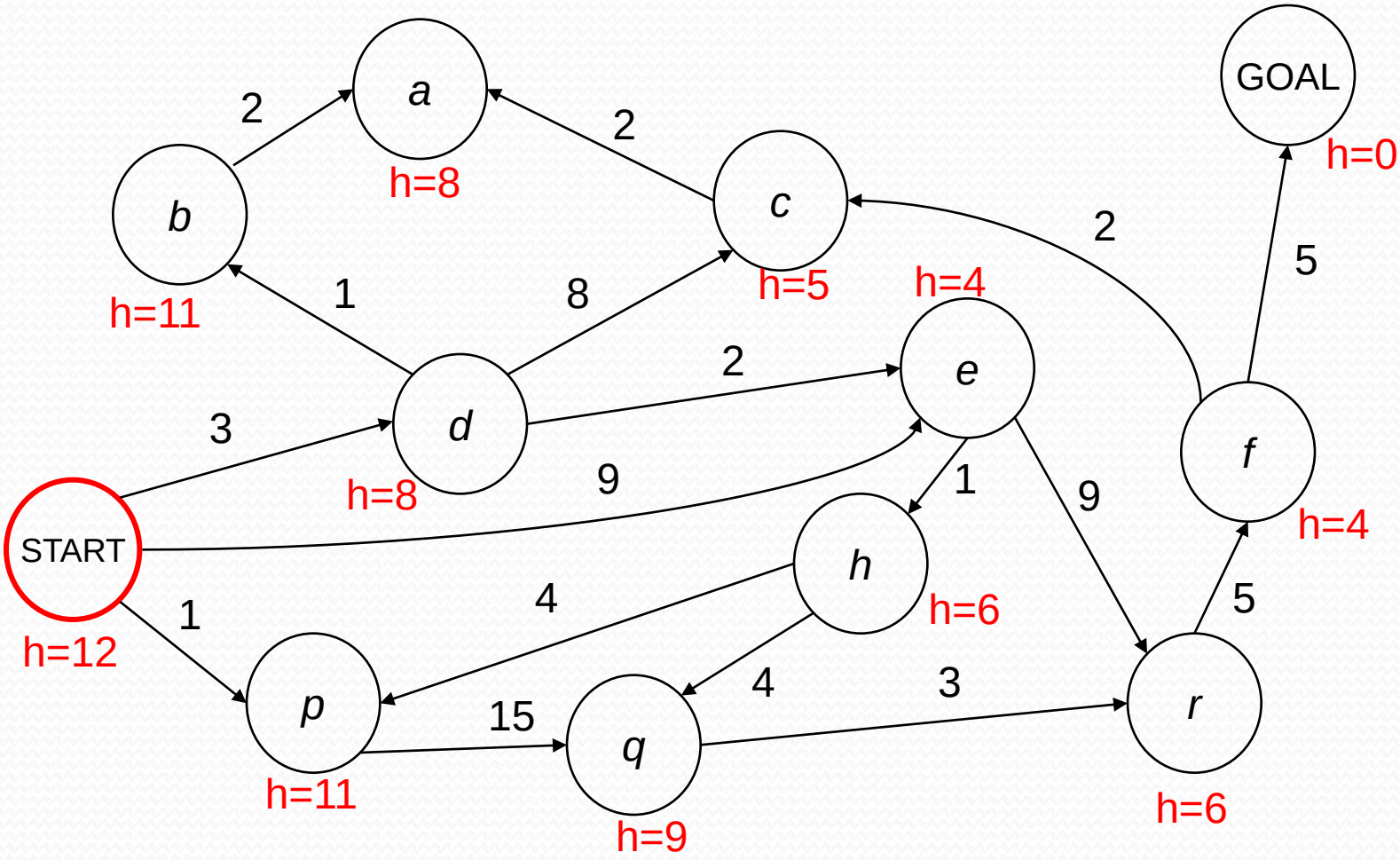
Heuristic theo Khoảng cách Euclide



Heuristic theo Khoảng cách Euclide

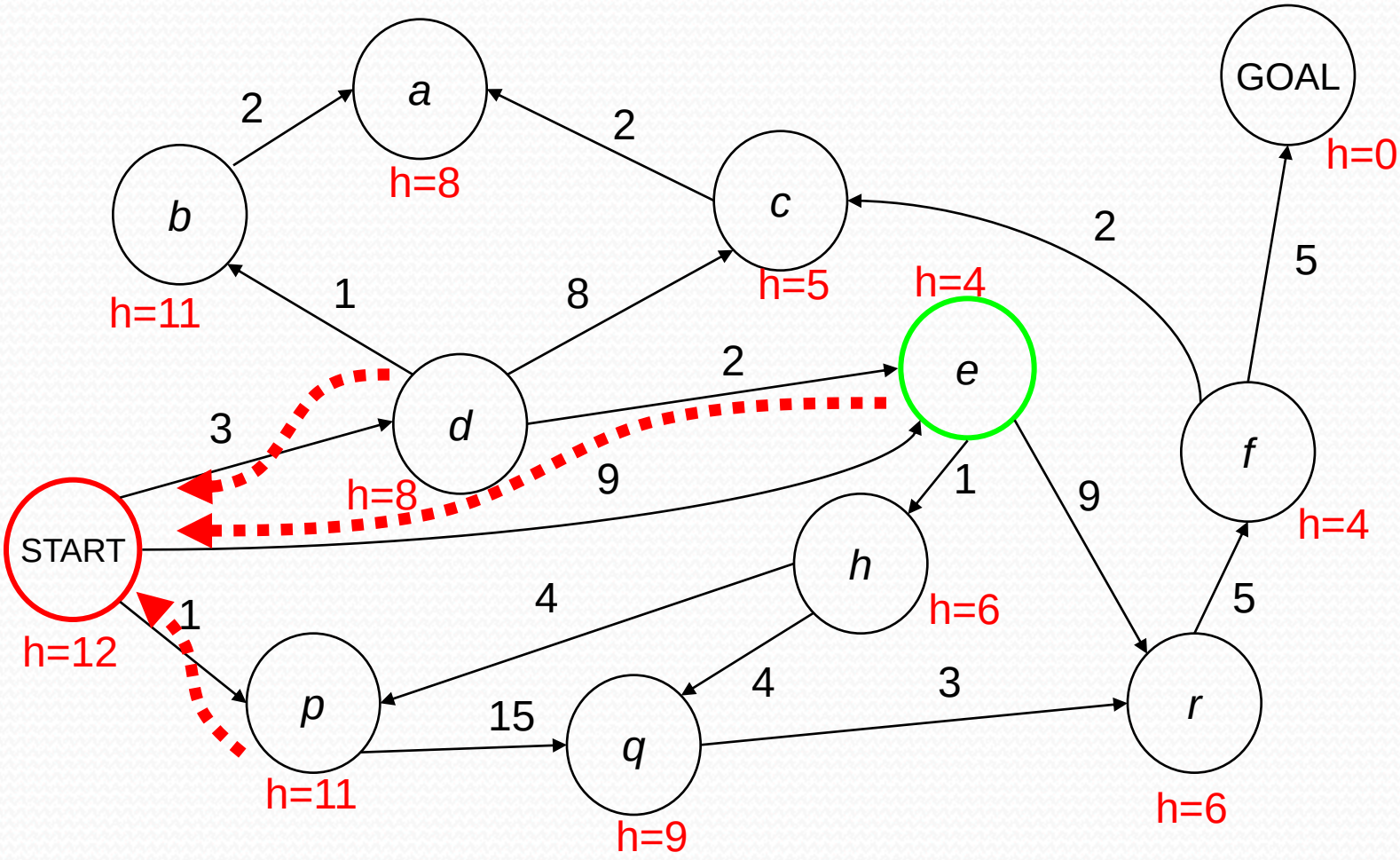


Heuristic theo Khoảng cách Euclide



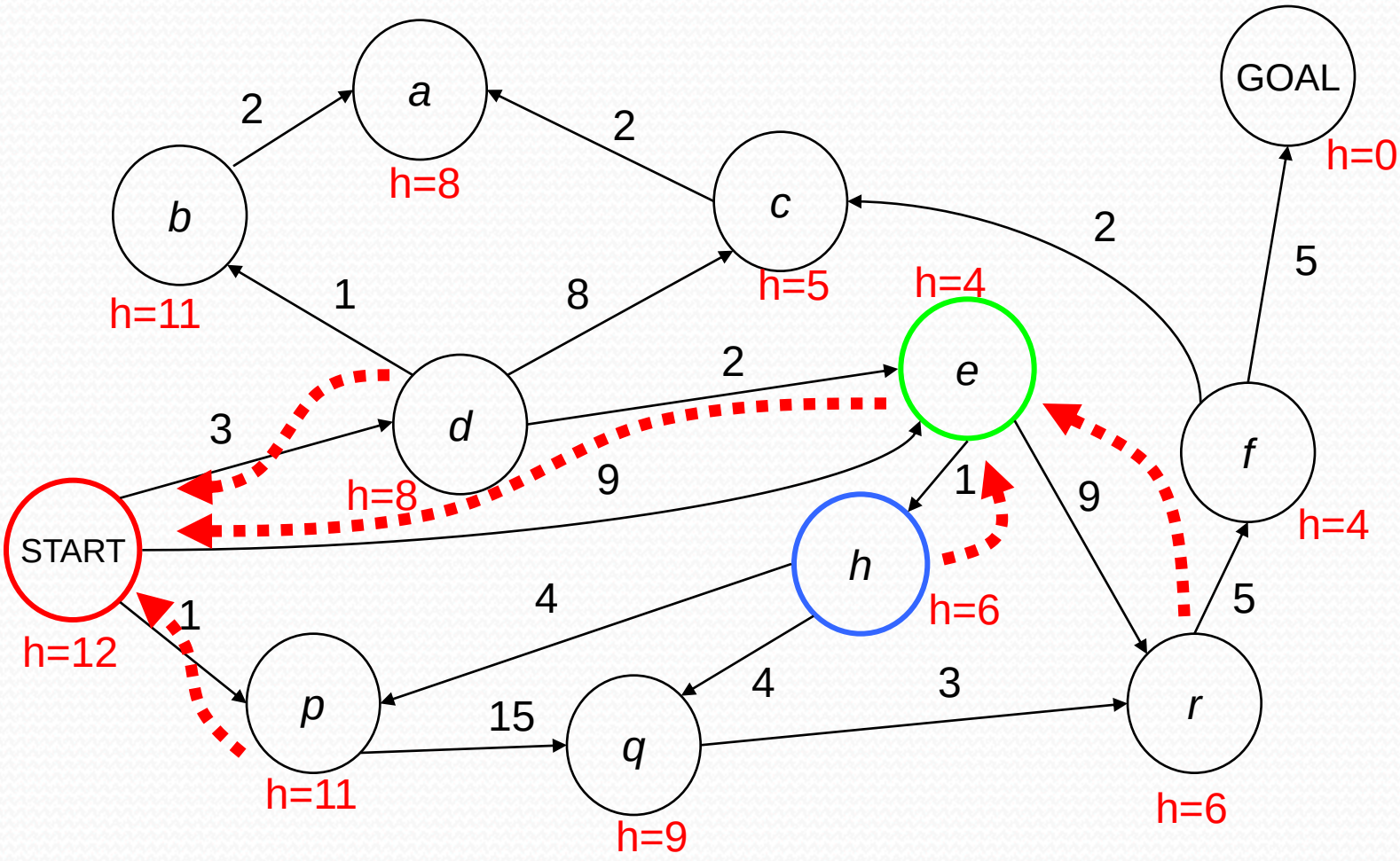
PQ = {(Start,12)}

Heuristic theo Khoảng cách Euclide



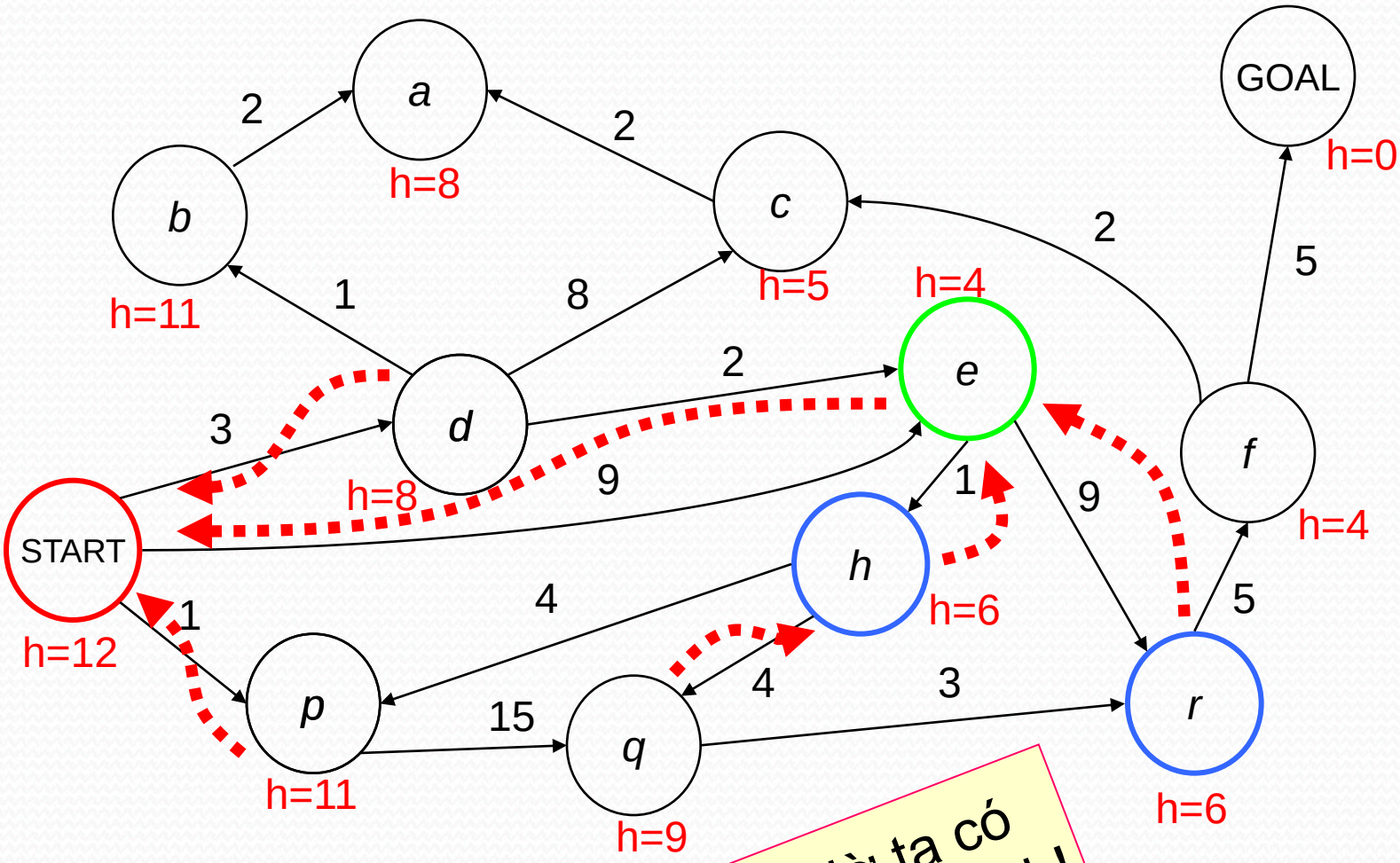
PQ = {(e,4),(d,8),(p,11)}

Heuristic theo Khoảng cách Euclide



PQ = {(h,6),(r,6),(d,8),(p,11)}

Heuristic theo Khoảng cách Euclide

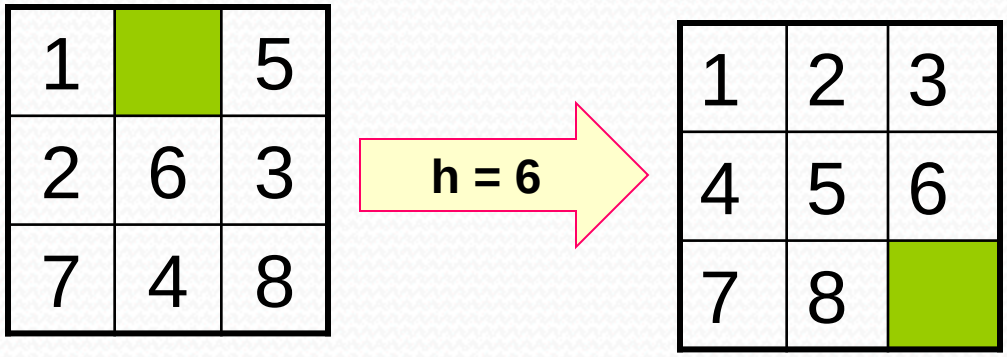


PQ = {(r,6),(d,8),(q,9),(p,11)}

Bây giờ ta có thể tiến về đích!

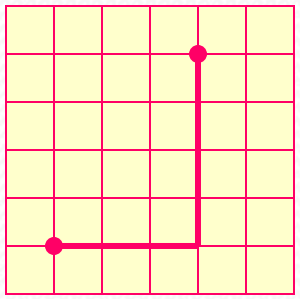
Heuristic trong bài toán 8-puzzle

- Theo số ô nằm sai vị trí



Heuristic trong bài toán 8-puzzle

- Theo tổng khoảng cách Manhattan



d = dx + dy

1		5
2	6	3
7	4	8

h = 9

1	2	3
4	5	6
7	8	

h = 0 + 2 + 1 + 2 + 2 + 1 +0 + 1 = 9

Tìm kiếm "Tham lam"

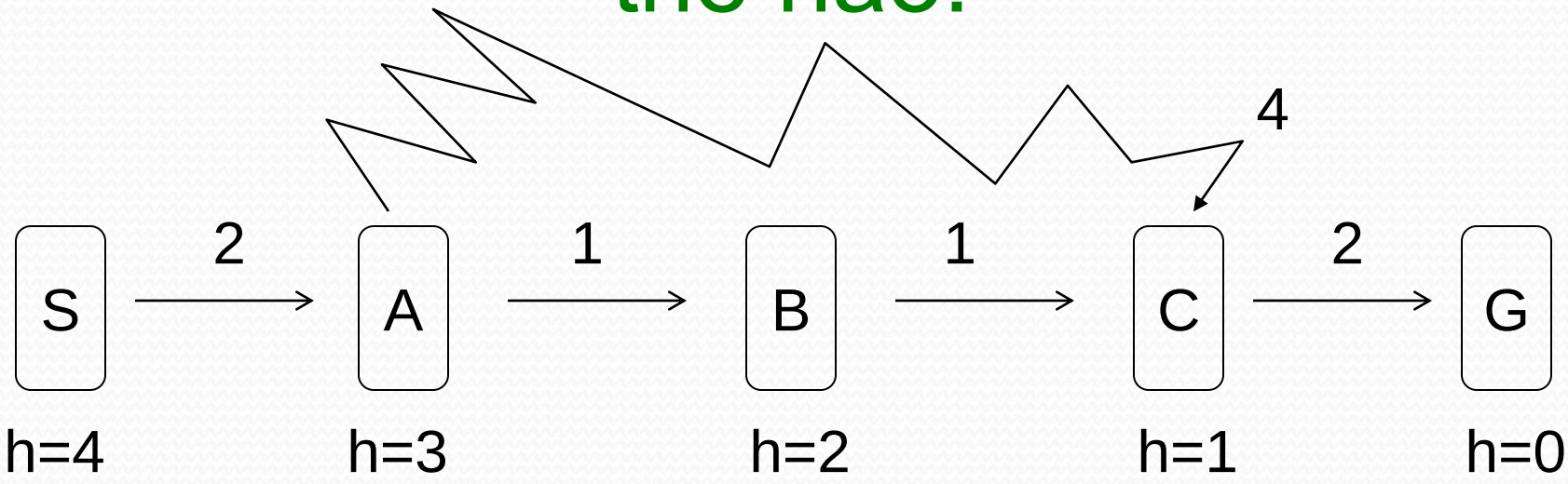
Init-PriQueue(PQ)
Insert-PriQueue(PQ,START,h(START))
while (PQ khác rỗng và PQ không chứa trạng thái đích)
(s , h) := Pop-least(PQ)
Với mỗi s' trong succs(s)
Nếu s' không có trong PQ và s' chưa được viếng trước đó bao giờ
Insert-PriQueue(PQ,s',h(s'))

Thuật toán		Đủ	Tối ưu	Thời gian	Không gian
BestFS	Best First Search	C	**K**	$O(min(N,B^{LMAX}))$	$O(min(N,B^{LMAX}))$

Một vài cải tiến của thuật toán này có thể làm cho mọi việc tốt đẹp hơn. Nó là thứ mà chúng ta gọi là: A*….

* Việc sử dụng heuristic làm thay đổi B

Hãy Xem "Tham lam" Ngớ Ngẩn thế nào!



- Tìm kiếm tham lam rõ ràng không đảm bảo tìm thấy lời giải mong muốn
- Câu hỏi: Chúng ta có thể làm gì để tránh lỗi ngớ ngẩn này?

A* - Ý tưởng Cơ bản

- Best-first greedy: Khi bạn mở một node *n*, lấy node con *n'* và đặt nó vào PriQueue với độ ưu tiên $h(n')$
- A*: Khi bạn mở một node *n*, lấy node con *n'* và đặt nó vào PriQueue với độ ưu tiên

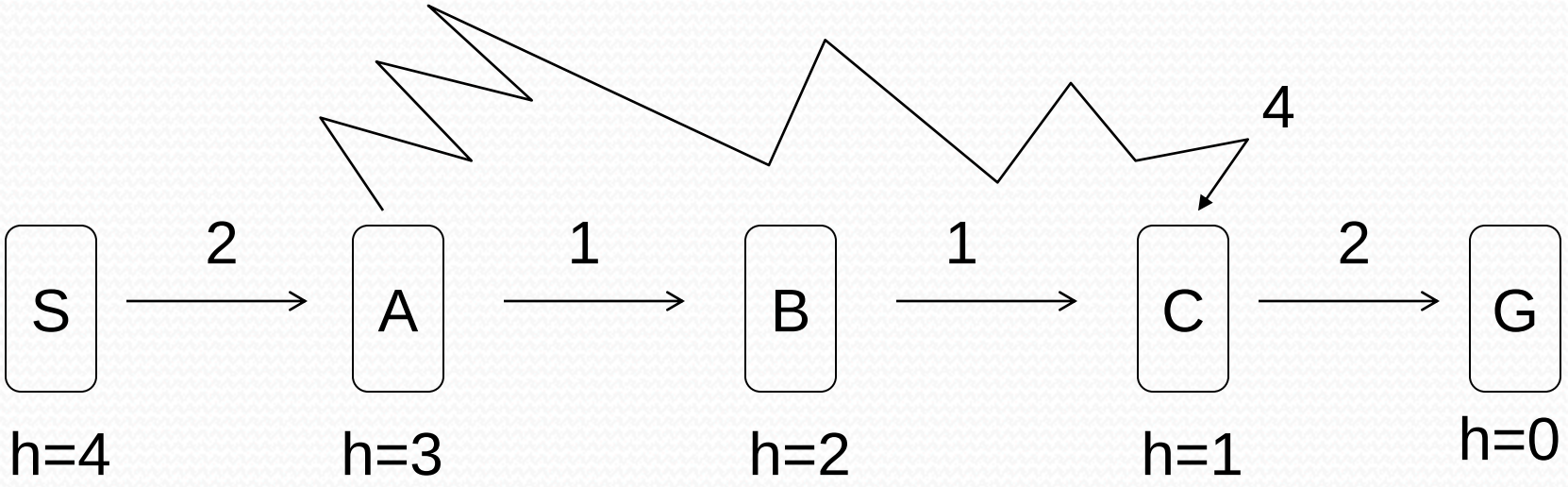
(Chi phí đi đến n') + $h(n')$ (1)

Đặt $g(n)$ = Chi phí đi đến n (2)

và định nghĩa…

$$f(n) = g(n) + h(n) \quad (3)$$

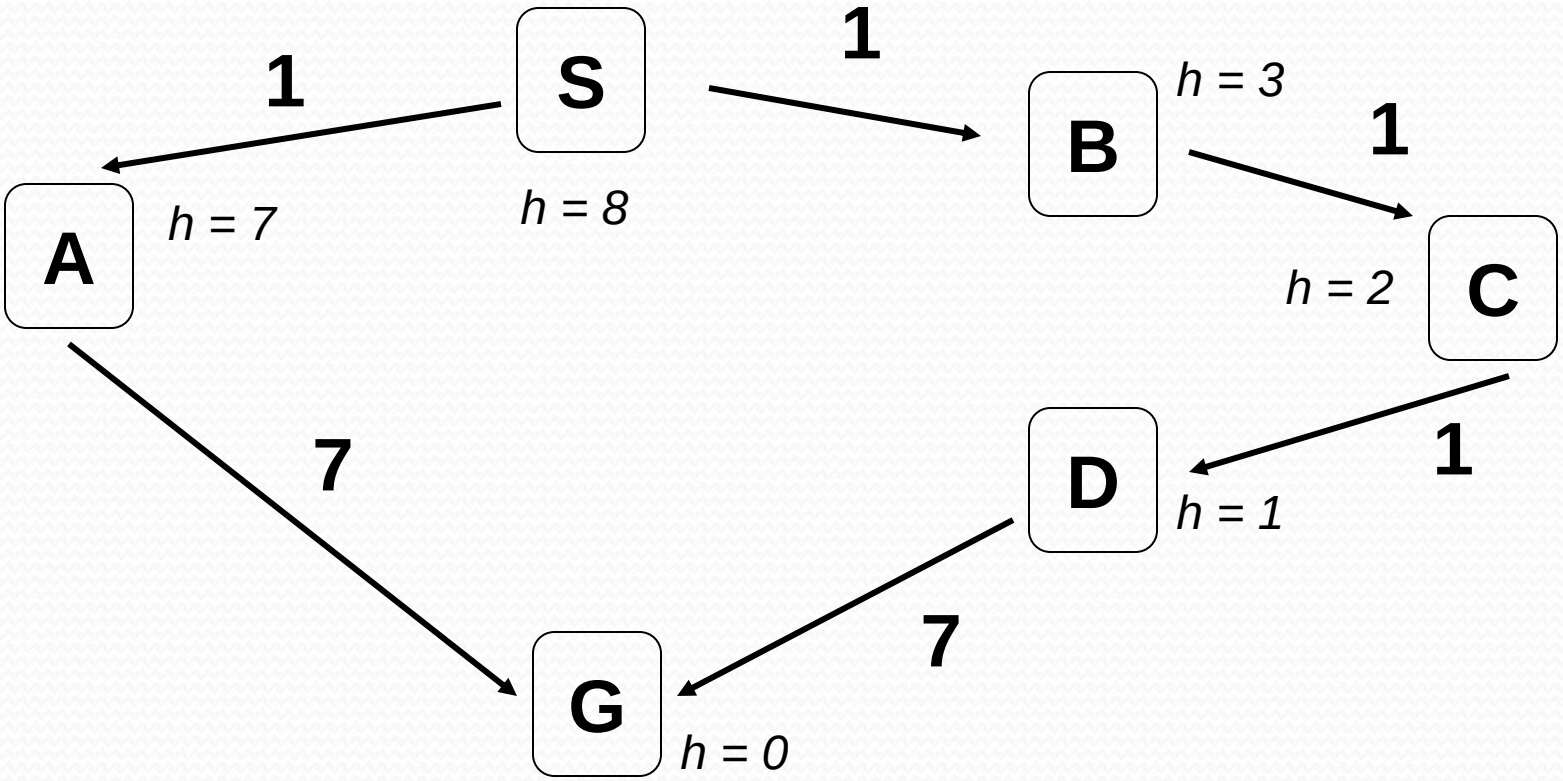
A* Không Ngớ Ngẩn!



A* dừng khi nào?

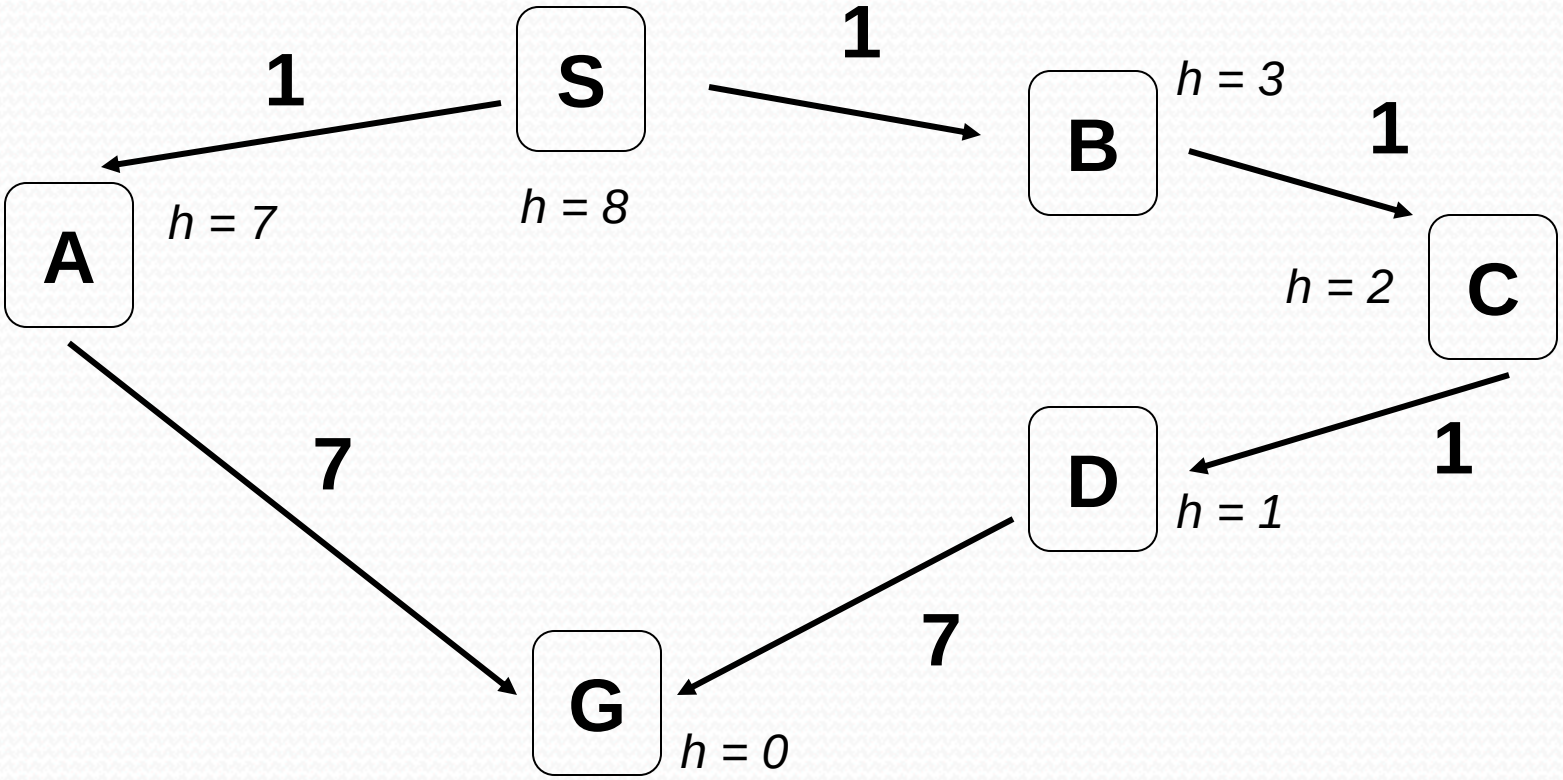
Ý tưởng: Ngay khi nó phát sinh được trạng thái đích?

Xem ví dụ sau:



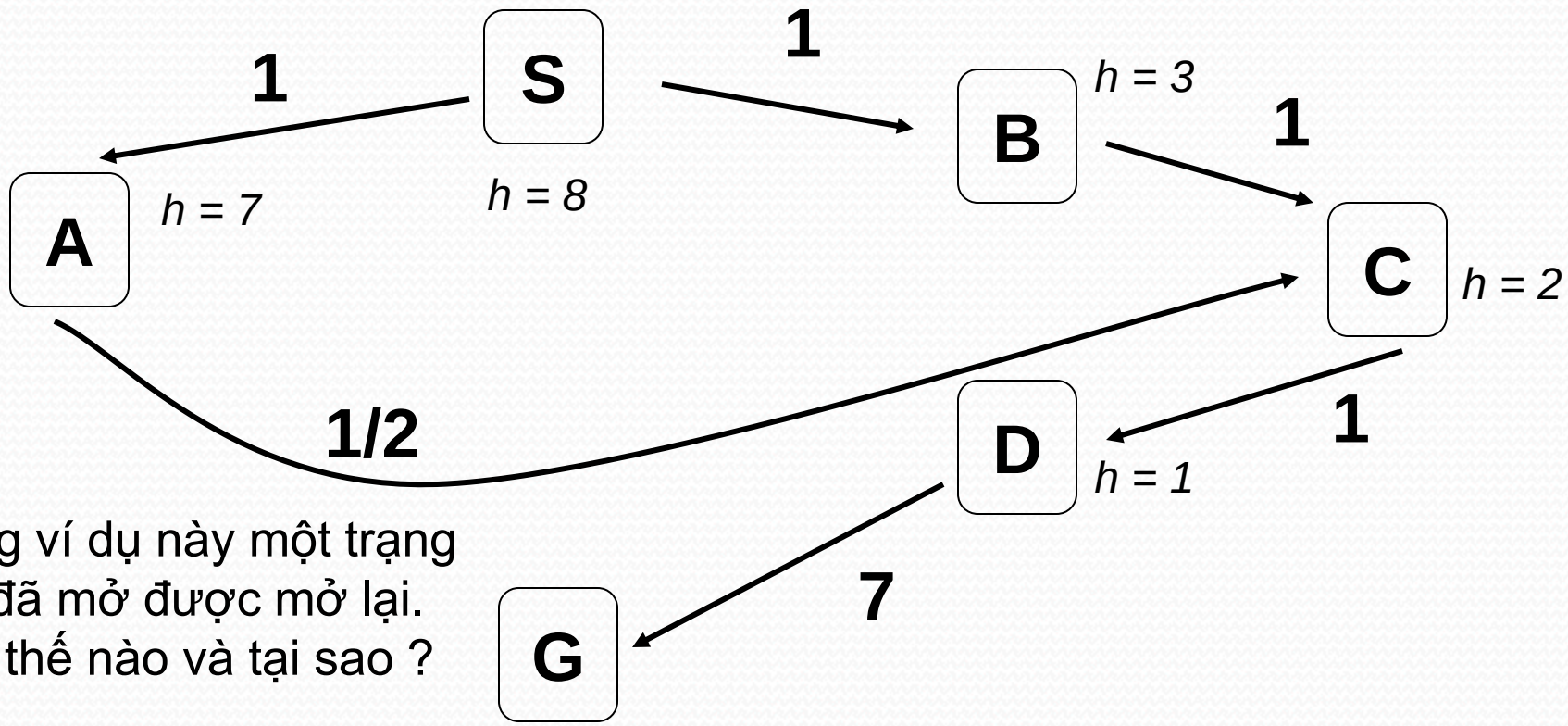
Quy tắc Dừng A* Đúng Đắn:

A* Dừng Chỉ Khi một Trạng Thái Đích Được Lấy ra khỏi Priority Queue



Các trạng thái quay lại A*

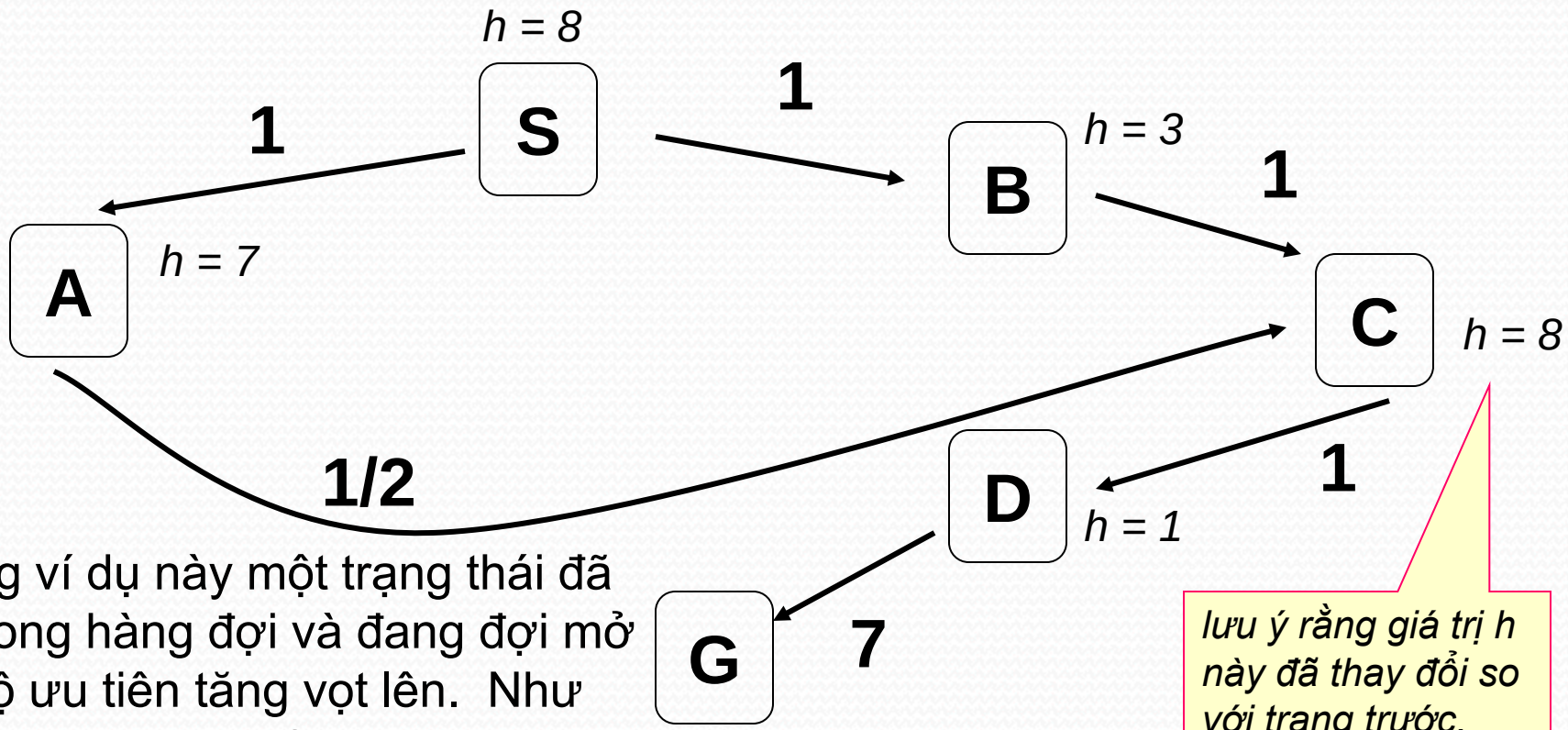
Một câu hỏi khác: Điều gì xảy ra nếu A* quay lại một trạng thái đã mở, và tìm được một đường ngắn hơn?



Trong ví dụ này một trạng thái đã mở được mở lại. Như thế nào và tại sao ?

Các trạng thái quay lại A*

Điều gì nếu A* thăm một trạng thái đã có trong hàng đợi?



Trong ví dụ này một trạng thái đã có trong hàng đợi và đang đợi mở có độ ưu tiên tăng vọt lên. Như thế nào và tại sao?

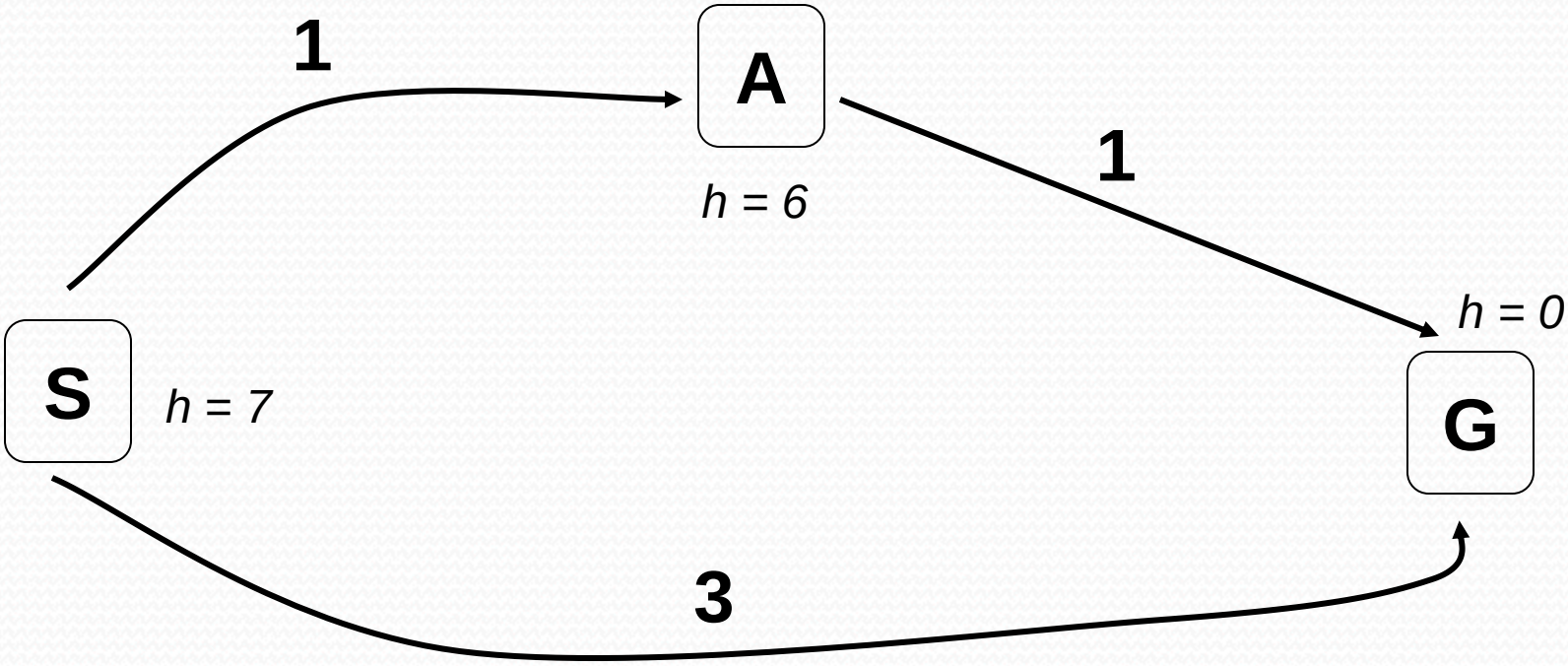
lưu ý rằng giá trị h này đã thay đổi so với trang trước.

Thuật toán A*

- Priority queue *PQ* ban đầu rộng.
- *V* (= tập các (bộ ba (*state,f,backpointer*)) đã thăm trước đó bắt đầu là rỗng.
- Đặt *S* vào *PQ* và *V* với độ ưu tiên $f(s) = g(s) + h(s)$
- *PQ* rỗng?
  - Có? Không có lời giải.
  - Không? Loại bỏ node với $f(n)$ thấp nhất khỏi queue. Gọi nó là *n*.
  - Nếu *n* là một đích, dừng và báo thành công.
  - "Mở" *n* : Với mỗi n' trong **succs**(n)
    - Đặt $f' = g(n') + h(n') = g(n) + cost(n,n') + h(n')$
    - **Nếu** *n'* chưa thấy trước đó, hay *n'* đã mở với $f(n')>f'$, hay *n'* hiện trong *PQ* với $f(n')>f'$
    - **Thì** Đặt/Cập nhật *n'* trong PQ với độ ưu tiên *f'* và cập nhật *V* để bao gồm (*state=n'*, *f '*, *BackPtr=n*).
    - **Ngược lại** bỏ qua *n'*

dùng những mẹo để tính $g(n)$

A* Có Bảo đảm Tìm thấy Đường đi Tối ưu?



Không. Và ví dụ sau cho thấy tại sao.

Heuristic chấp nhận được

- Đặt $h^*(n)$ = chi phí tối thiểu thấp nhất từ n đến đích.
- Một heuristic h là chấp nhận được nếu
  $h(n) <= h^*(n)$ với mọi trạng thái n.
- Một heuristic chấp nhận được đảm bảo không bao giờ ước tính quá chi phí đến đích.
- Một heuristic chấp nhận được là tối ưu.

Ví dụ 8-Puzzle

Trạng thái ví dụ

1		5
2	6	3
7	4	8

Trạng thái đích

1	2	3
4	5	6
7	8	

Heuristics nào sau đây là chấp nhận được?

- $h(n)$ = Số ô nằm sai vị trí trong trạng thái n
- $h(n) = 0$
- $h(n)$ = Tổng khoảng cách Manhattan giữa mỗi ô so với vị trí đích
- $h(n) = 1$
- $h(n) = \min(2, h^*(n))$
- $h(n) = h^*(n)$
- $h(n) = \max(2, h^*(n))$

A* với Heuristic Chấp nhận được Bảo đảm Đường đi Tối ưu

- Chứng minh đơn giản
- (Bạn có thể tự chứng minh…?)

So sánh Lặp Sâu dần với A*

Trong sách của Russell and Norvig, trang 107, Hình4.8

	Với 8-puzzle, số trạng thái được mở trung bình trong 100 bài toán được chọn ngẫu nhiên trong đó đường đi tối ưu dài…		
	…4 bước	…8 bước	…12 bước
Lặp Sâu dần	112	6,300	3.6×10^6
Tìm kiếm A* dùng "số ô sai vị trí" làm heuristic	13	39	227
A* dùng "Tổng khoảng cách Manhattan" làm heuristic	12	25	73

Ghi chú

1. Xem sơ qua có thể thấy ngay cả "số ô sai vị trí" cũng là một heuristic tốt. Nhưng có lẽ h(state)=0 cũng thực hiện tốt hơn ID, vì thế khác biệt chủ yếu do vấn đề lớn của ID mở cùng một trạng thái nhiều lần, không phải do dùng heuristic.
2. Đánh giá chỉ dựa trên "số trạng thái đã mở" không tính đến việc trả giá quá mức để duy trì bảng băm và hàng đợi ưu tiên cho A*, dù nó khá rõ ở đây rằng nó không thay đổi kết quả quá mức.

Thực sự chỉ có một vài trăm ngàn trạng thái cho toàn bộ bài toán 8-puzzle

		…8 bước	…12 bước
Lặp Sâu dần	112	6,300	3.6×10^6
Tìm kiếm A* dùng "số ô sai vị trí" làm heuristic	13	39	227
A* dùng "Tổng khoảng cách Manhattan" làm heuristic	12	25	73

A* : Khuyết điểm

- A* có thể dùng nhiều bộ nhớ. Trên lý thuyết:

  O(số trạng thái)

- Với không gian tìm kiếm thực sự lớn, A* sẽ dùng hết bộ nhớ.

IDA* : Tìm kiếm Với Bộ nhớ Giới hạn

- A* lặp với độ sâu tăng dần. Thật sự, rất khác so với A*. Giả sử chi phí là số nguyên.
  1. Thực hiện lặp-không dùng DFS, không mở rộng node nào có $f(n) > 0$. Có tìm thấy đích? Nếu có, dừng.
  2. Thực hiện lặp-không dùng DFS, không mở rộng node nào có $f(n) > 1$. Có tìm thấy đích? Nếu có, dừng.
  3. Thực hiện lặp-không dùng DFS, không mở rộng node nào có $f(n) > 2$. Có tìm thấy đích? Nếu có, dừng.
  4. Thực hiện lặp-không dùng DFS, không mở rộng node nào có $f(n) > 3$. Có tìm thấy đích? Nếu có, dừng.

  …lặp lại điều này, tăng ngưỡng $f(n)$ lên một 1 mỗi lần, cho đến khi dừng.
- Cái này
  - Đầy đủ
  - Bảo đảm tìm được lời giải tối ưu
  - Nói chung tốn chi phí nhiều hơn A*.

Điều cần nắm

- Hiểu thấu đáo A*.
- Có thể chạy tay các ví dụ thực thi A* đơn giản.
- Hiểu được "tính chấp nhận được" của heuristics. Chứng minh tính đầy đủ, bảo đảm tính tối ưu của đường đi.
- Có thể nhận xét về các đánh giá.

Chứng minh: A* Heuristic Chấp nhận được Bảo đảm Tối ưu

- Giả sử nó tìm thấy đường đi không tối ưu, kết thúc tại trạng thái đích G_1 trong đó $f(G_1) > f^*$ với $f^* = h^*(start)$ = chi phí đường đi tối ưu.
- Phải tồn tại một node n
  - Chưa mở
  - Đường đi từ điểm đầu đến n (lưu trong các giá trị BackPointers(n)) là bắt đầu của đường đi tối ưu thật sự

- $f(n) >= f(G_1)$ (ngược lại tìm kiếm đã không kết thúc)
- Cũng thế $f(n) = g(n) + h(n)$

  $= g^*(n) + h(n)$

  $<= g^*(n) + h^*(n)$

  $= f^*$

Do đó $f^* >= f(n) >= f(G_1)$

Chứng minh: A* Heuristic Chấp nhận được Bảo đảm Tối ưu

- Giả sử nó tìm thấy đường đi không tối ưu, kết thúc tại trạng thái đích G_1 trong đó $f(G_1) > f^*$ với $f^* = h^*(start)$ = chi phí đường đi tối ưu.
- Phải tồn tại một node *n*
  - Chưa mở
  - Đường đi từ điểm đầu đến *n* (lưu trong các giá trị BackPointers(*n*)) là bắt đầu của đường đi tối ưu thật sự

- $f(n) >= f(G_1)$ (ngược lại tìm k… ết th…
- Cũng thế $f(n) = g(n) + h(n)$

  $= g^*(n) + h(n)$

  $<= g^*(n) + h^*(n)$

  $= f^*$

vì nó nằm trên đường đi tối ưu

Do giả thiết chấp nhận được

Vì *n* nằm trên đường đi tối ưu

mâu thuẫn

Tại sao một node như thế phải tồn tại? Xem xét bất kỳ đường đi tối ưu *s*,*n1*,*n2*…goal. Nếu các node dọc theo nó được mở, đích sẽ được đi đến theo đường đi ngắn nhất.

Do đó $f^* >= f(n) >= f(G_1)$