Các Thuật Toán Thông Minh Nhân Tạo & Ứng Dụng

Chương 2

Solving Problems by Searching

Giảng viên: Thái Hùng Văn

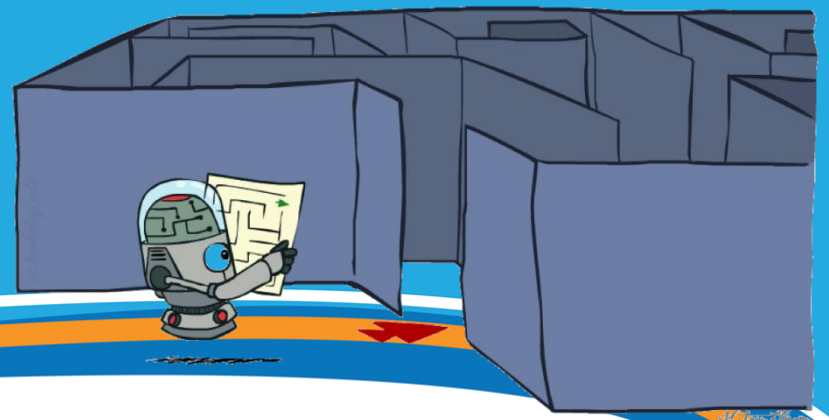
Email: thvan@fit.hcmus.edu.vn

Khoa Công Nghệ Thông Tin
Trường Đại Học Khoa Học Tự Nhiên
ĐHQG-HCM

Các Thuật Toán Thông Minh Nhân Tạo & Ứng Dụng

2.2

CÁC CHIẾN LƯỢC TÌM KIẾM CÓ THÔNG TIN



Khoa Công Nghệ Thông Tin
Trường Đại Học Khoa Học Tự Nhiên
VNU-HCMC

Tổng quan

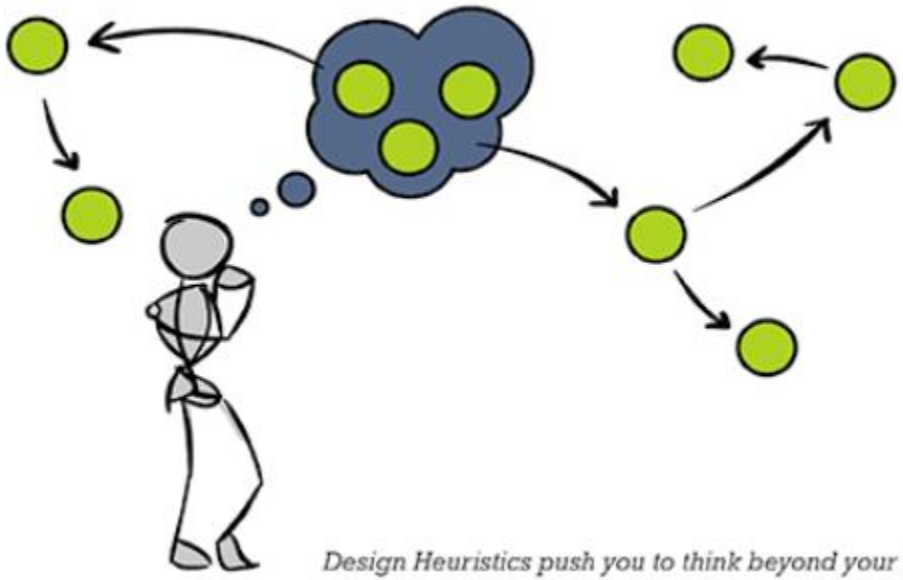
Các thuật toán tìm kiếm mù (DFS, BFS, UCS,…) không sử dụng thông tin TT đích để định hướng – do đó thường khó tối ưu. (*Cho dù UCS đảm bảo tìm được lời giải với chi phí nhỏ nhất nhưng do UCS mở rộng theo mọi hướng chứ không có định hướng về TT đích nên* ***chạy chậm***)

⇒Cần quan tâm tới các thuật toán tìm kiếm có sử dụng thông tin trạng thái đích khi định hướng để gia tăng tốc độ TK.

Có 2 thuật toán điển hình là **Greedy Search** và **A***

Heuristic

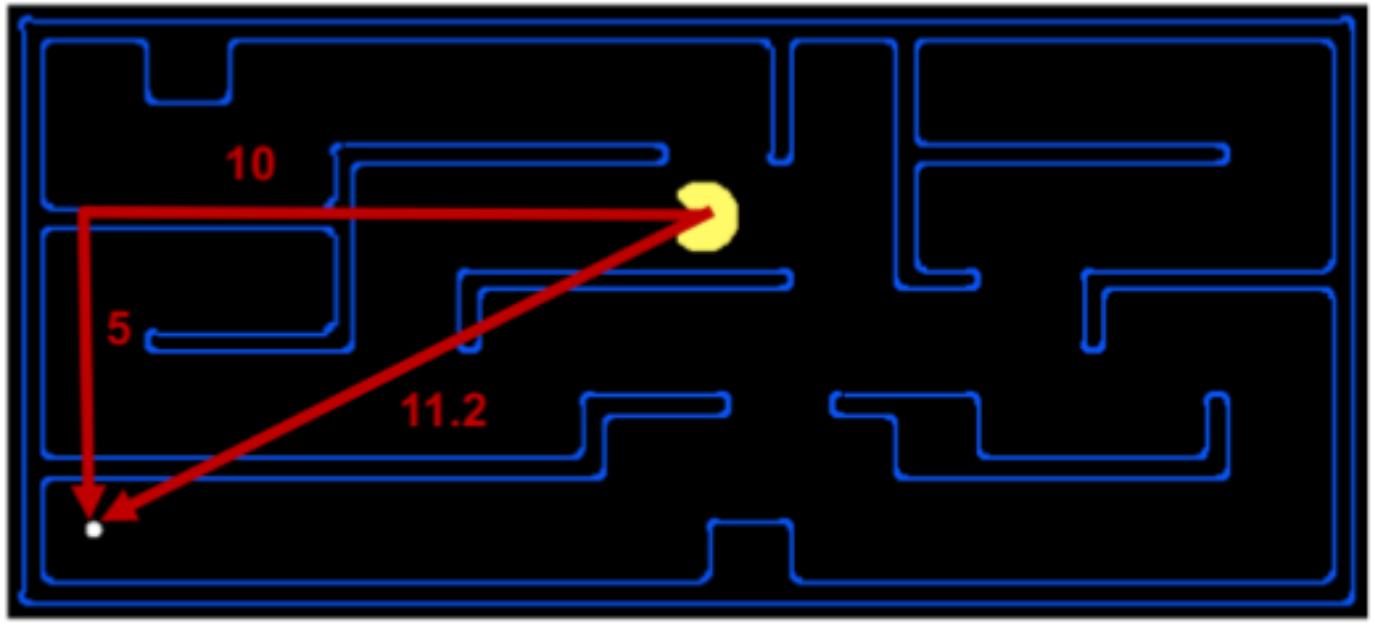
- Một Heuristic là **một hàm** nhận đầu vào là một TT và **trả về mức độ gần đích ước lượng của TT** này
- Mỗi Heuristic được thiết kế riêng cho một bài toán cụ thể, tùy thuộc vào ngữ cảnh đặc trưng của bài toán



Design Heuristics push you to think beyond your initial ideas

Xây dựng Heuristic

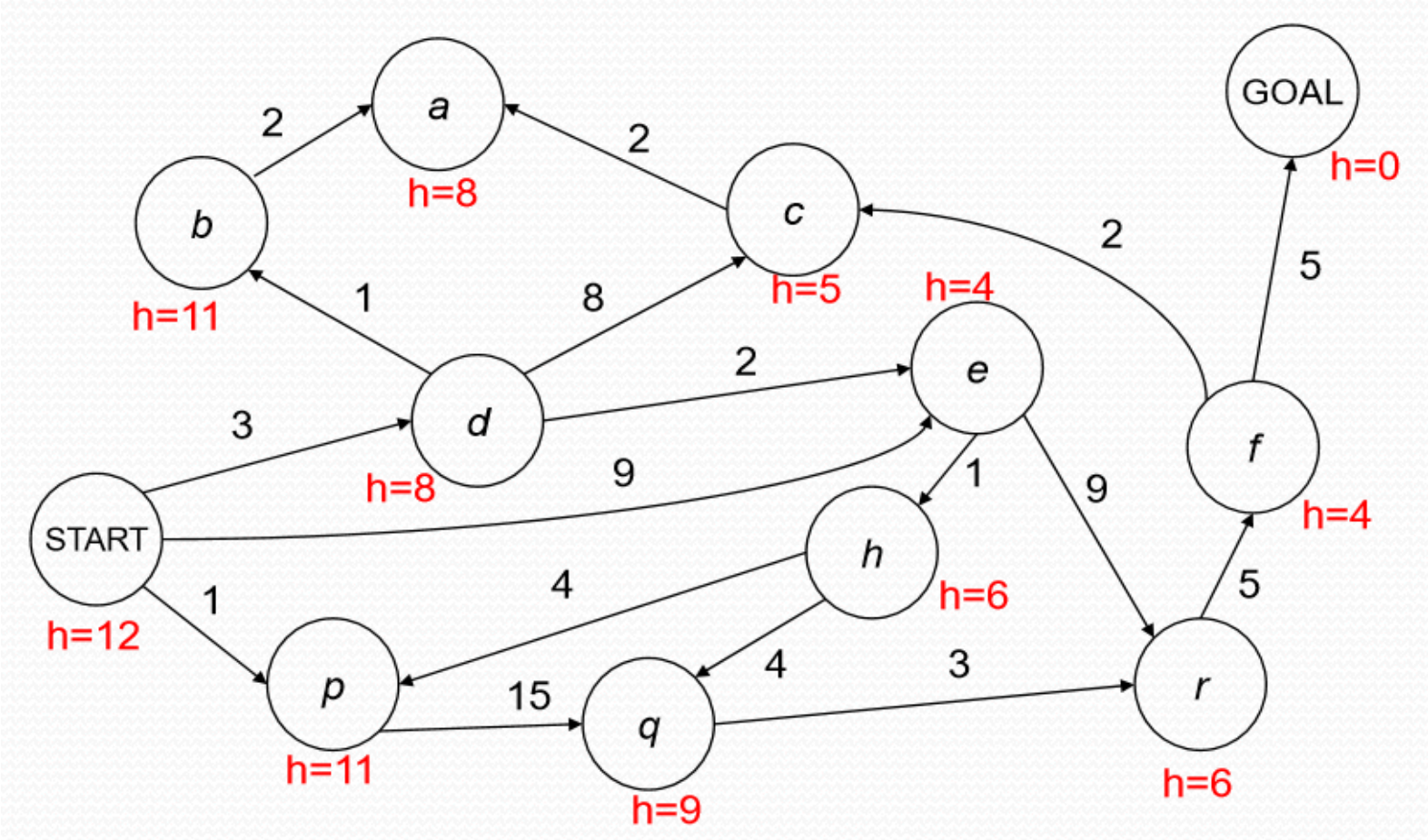
- Tùy vào bài toán, thông thường Heuristic có thể là khoảng cách Manhattan hoặc khoảng cách Euclidean (như trong trò Pacman):



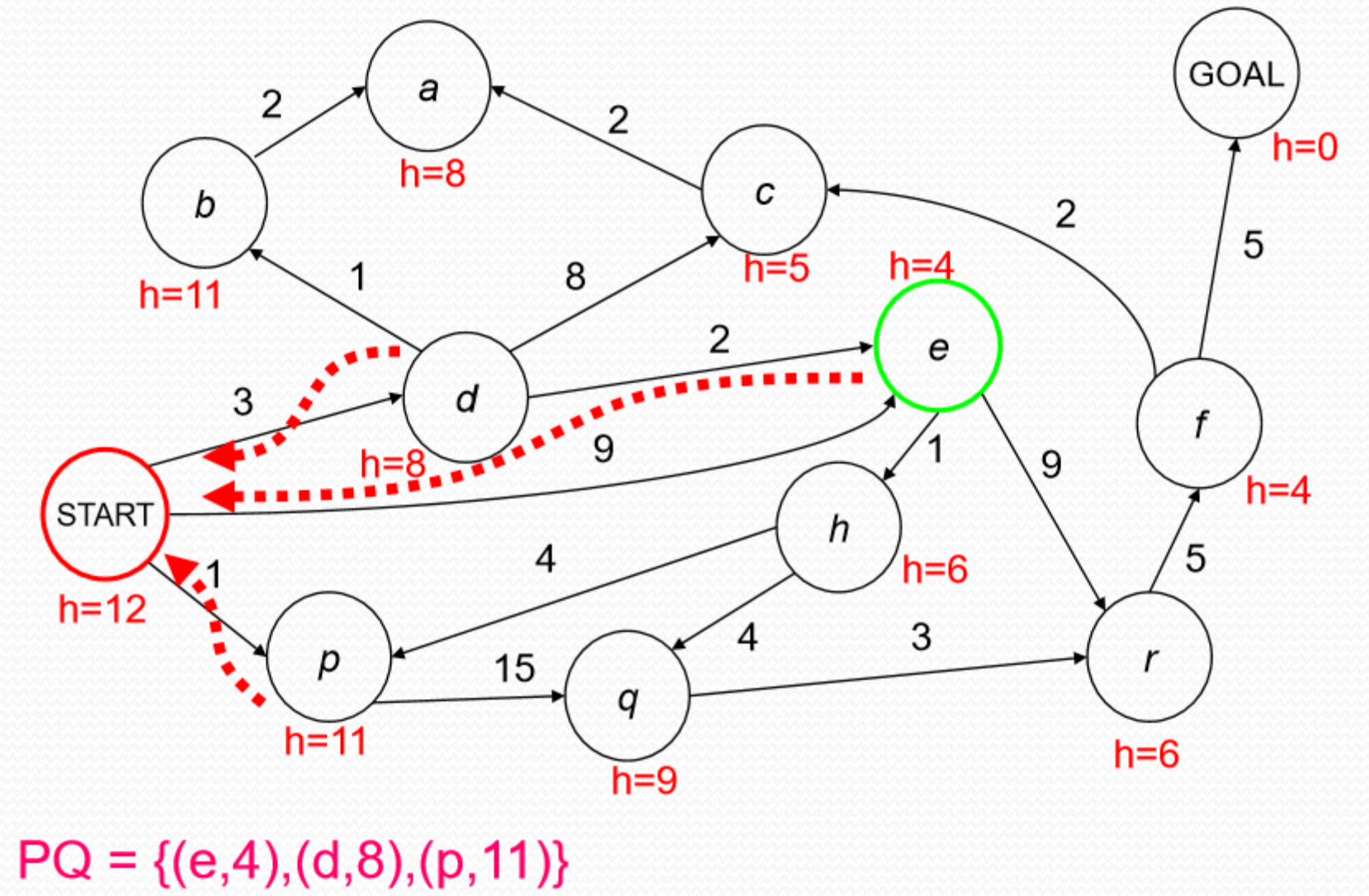
Nguồn ảnh: UC Berkely. CS188 Intro to AI

Heuristic cho bài đường đi "vật lý"

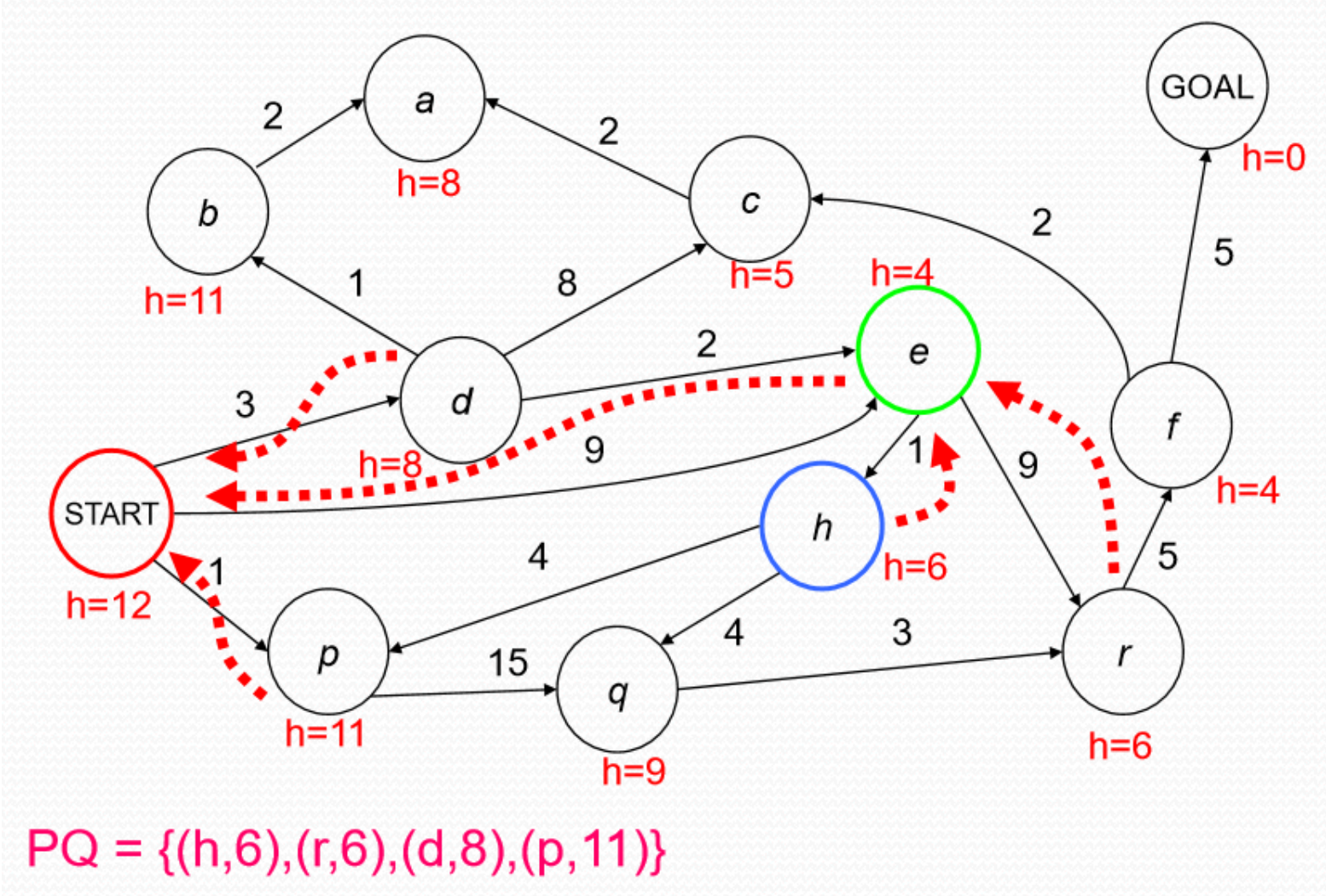
- Các đỉnh có Heuristic "**mức độ gần đích ước lượng**" là khoảng cách Euclide tới đích.



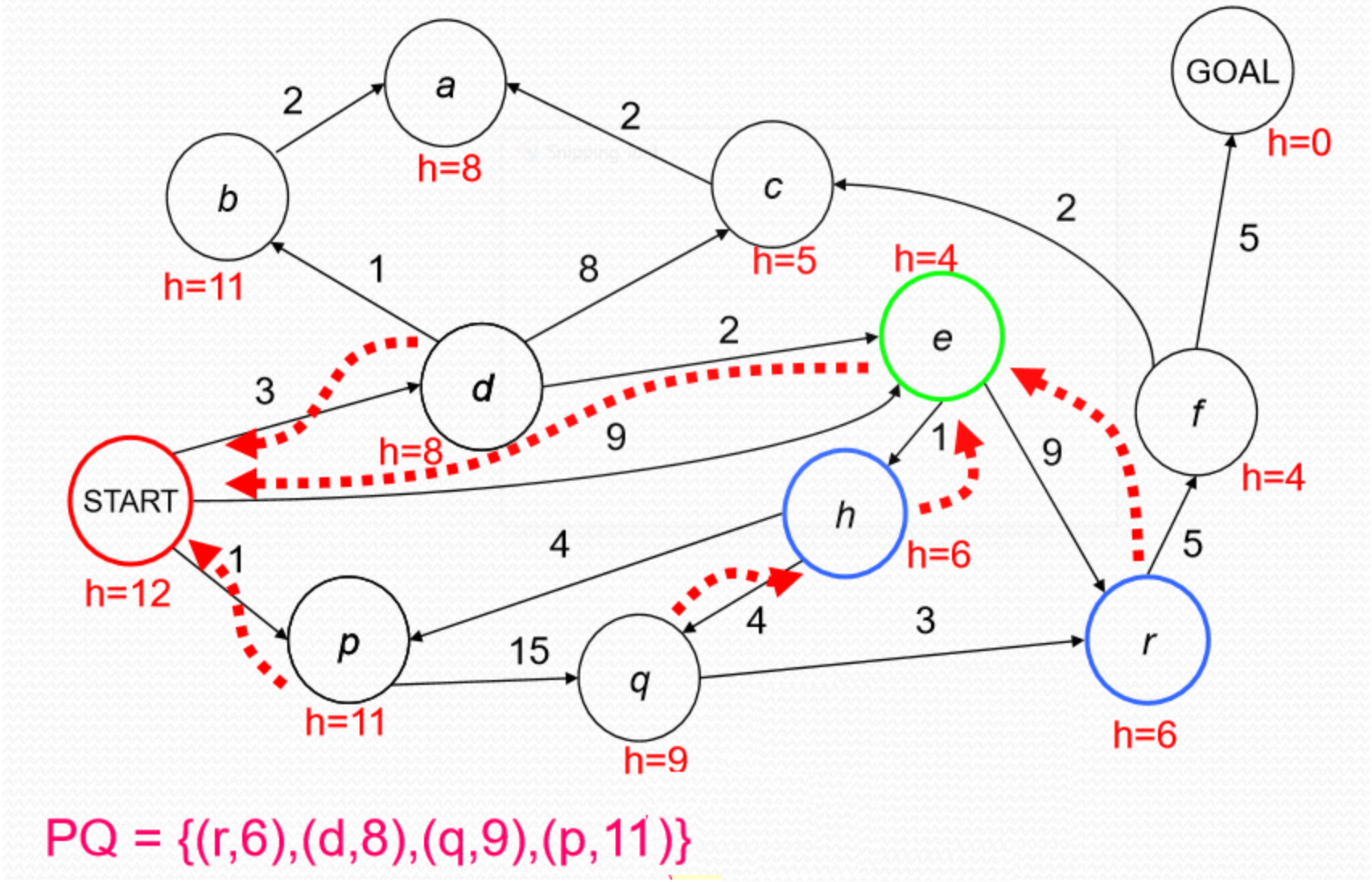
Heuristic cho bài đường đi “vật lý”



Heuristic cho bài đường đi “vật lý”



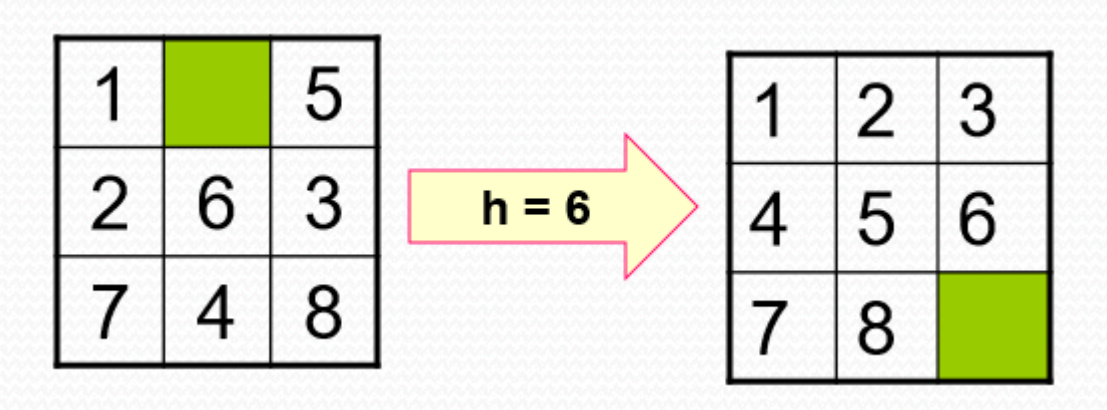
Heuristic cho bài đường đi “vật lý”



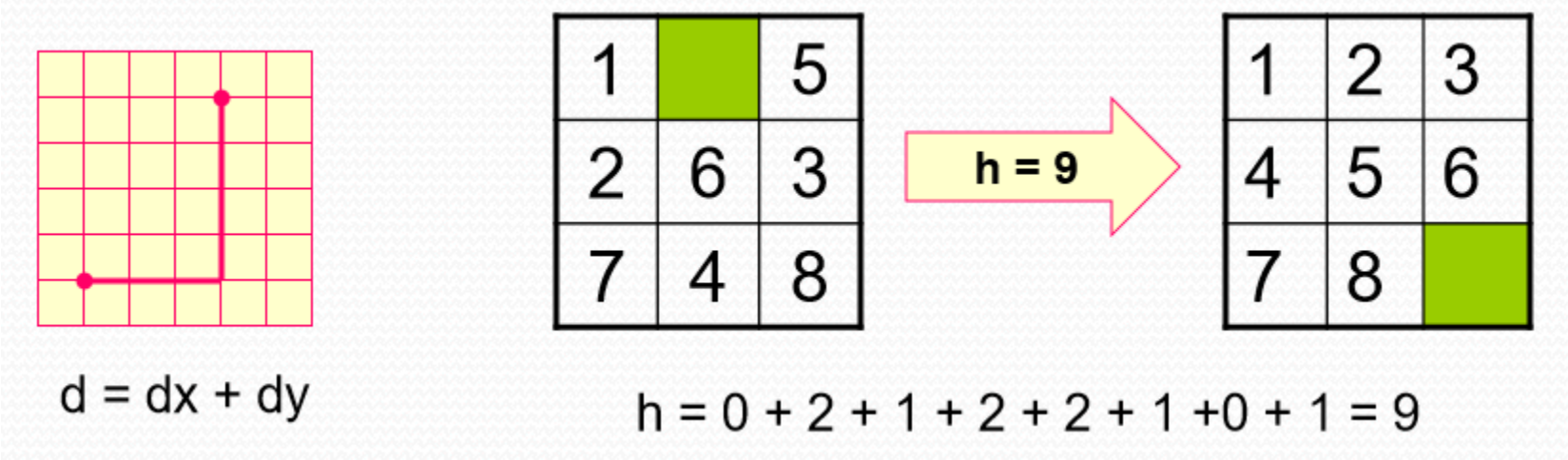
Heuristic cho bài sliding puzzle

- Heuristic có thể là:

1/ Số ô sai vị trí:



2/ Tổng số bước di chuyển các ô sai vị trí đến đúng chỗ (khoảng cách Manhattan):



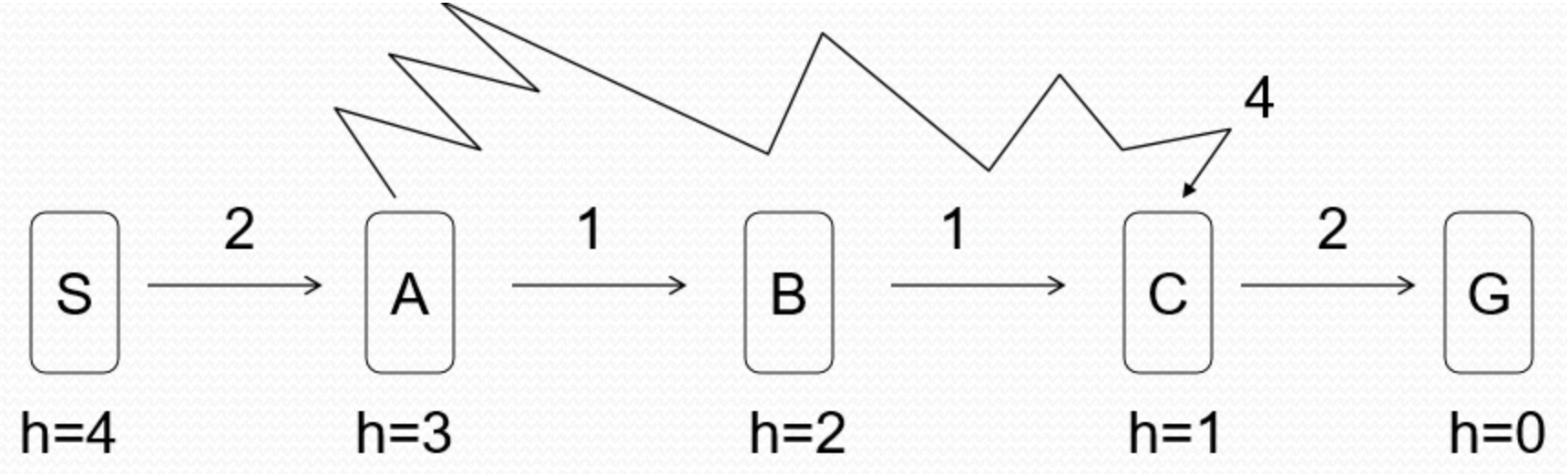
Thuật toán Greedy Search

- Thuật toán Greedy tìm kiếm tối ưu kiểu "Tham lam"

Nguồn ảnh: UC Berkely. CS188 Intro to AI

Thuật toán Greedy Search

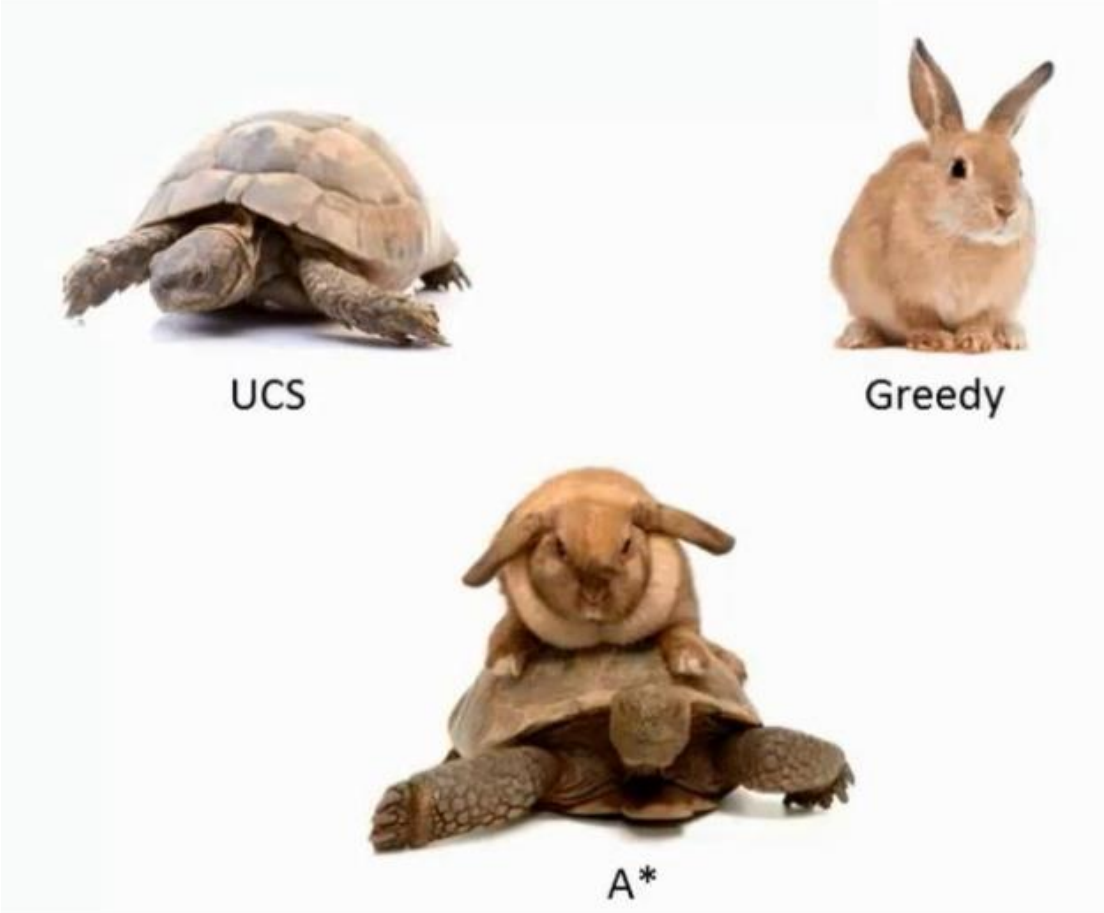
- **Chiến lược** chọn kế hoạch từ frontier để mở rộng: chọn kế hoạch **gần với TT đích nhất** theo heuristic → chạy nhanh (nếu heuristic đủ tốt) nhưng không đảm bảo tìm được kế hoạch có chi phí thấp nhất.



Nguồn ảnh: UC Berkely. CS188 Intro to AI

Thuật toán A*

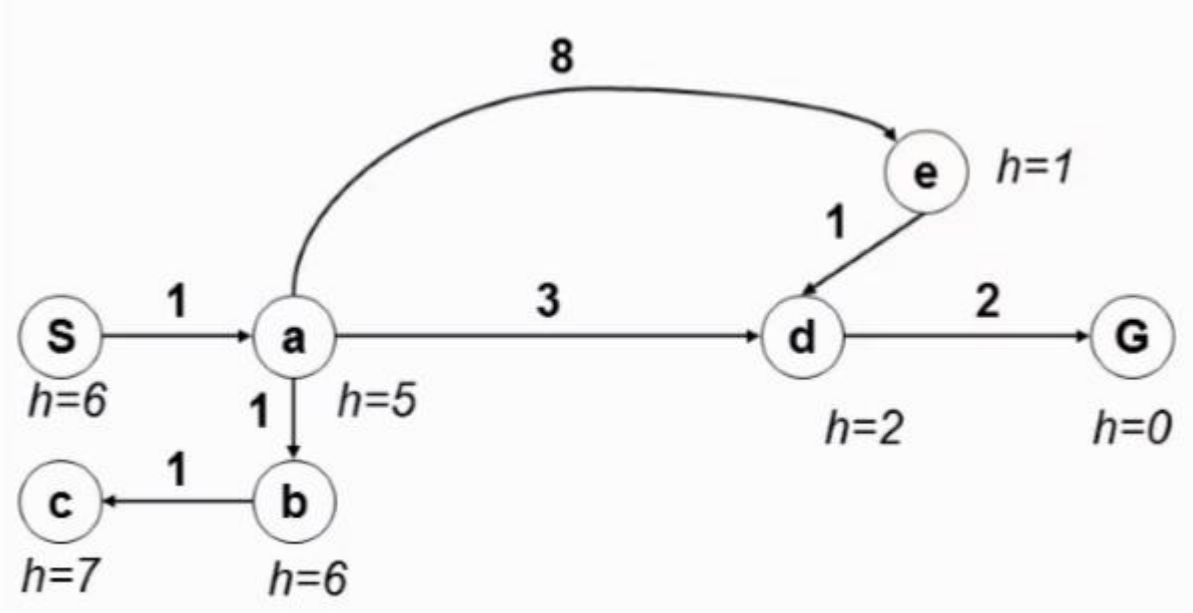
A* = UCS + Greedy



Nguồn ảnh: UC Berkely. CS188 Intro to AI

Thuật toán A* - lý do đề xuất

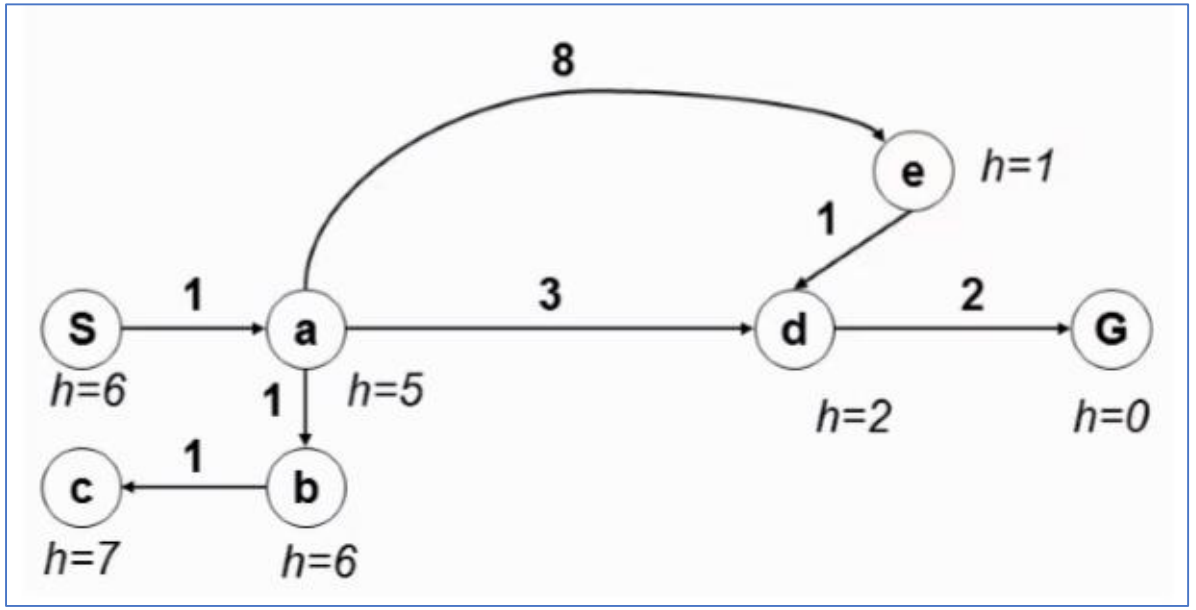
- Thuật toán **UCS** chọn kế hoạch có **chi phí lùi g nhỏ nhất** để mở rộng
- Thuật toán **Greedy Search** chọn kế hoạch có **chi phí tiến h nhỏ nhất**
- Cả 2 chiến lược trên đều có vấn đề! Ví dụ:



Nguồn ảnh: UC Berkely. CS188 Intro to AI

Thuật toán A* - cách chọn kế hoạch

- Khi mở kế hoạch S → a, ta được 3 KH:
  1. S → a → b (g = 2, h = 6)
  2. S → a → d (g = 4, h = 2)
  3. S → a → e (g = 9, h = 1)



- UCS sẽ mở tiếp KH1 dù không về đích
- Greedy sẽ chọn KH3 để mở tiếp dù kế hoạch này có chi phí (lùi) lớn

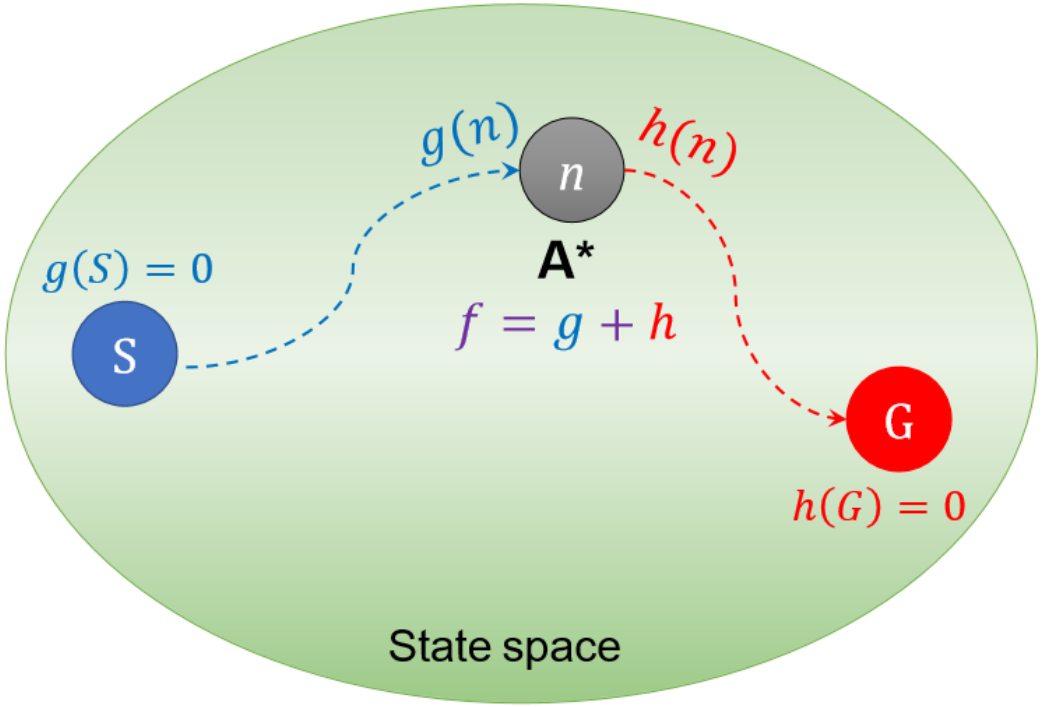
➔A* kết hợp UCS & Greedy sẽ chọn kế hoạch có f = g + h nhỏ nhất để mở rộng

Trong ví dụ trên, A* sẽ chọn KH2 (cả g lẫn h đều không quá lớn)

Nguồn ảnh: UC Berkely. CS188 Intro to AI

Thuật toán A* - cách chọn kế hoạch

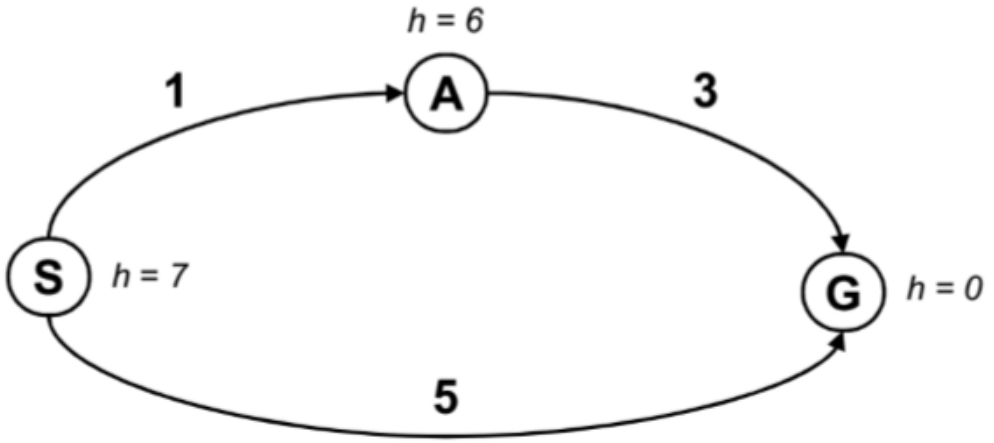
- A* (A-star) là thuật toán cải tiến từ A1 & A2, được đưa ra từ năm 1968.
- Với h(n)=0, thì A* chính là thuật toán Dijkstra



Nguồn ảnh: UC Berkely. CS188 Intro to AI

Thuật toán A* - đánh giá

- A* có tối ưu không?
  - Hình bên cho thấy A*không tối ưu
  - Lý do là bởi Heuristic không chính xác



➔Để **A*** tìm được kế hoạch **tối ưu** thì **Heuristic h phải hợp lệ**

- Đặt **h*(n)** = chi phí tối thiểu từ trạng thái <n> đến đích, Heuristic **h** là hợp lệ khi với mọi đỉnh <n> ta đều có **h(n) <= h*(n)**
- Trong A*, việc quan trọng nhất là thiết kế ra Heuristic hợp lệ và chính xác!

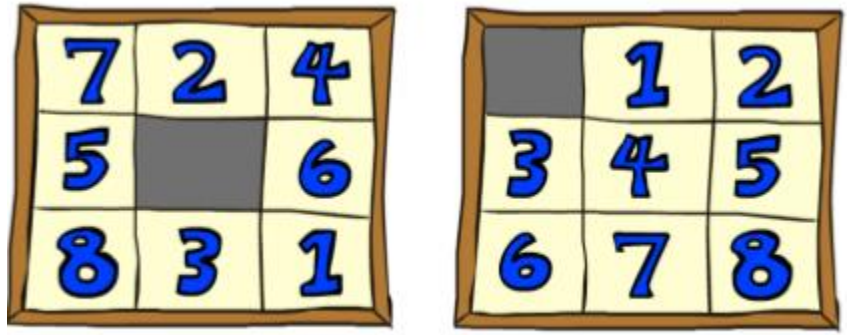
Nguồn ảnh: UC Berkely. CS188 Intro to AI

Thuật toán A* - cách tạo Heuristic hợp lệ

- Tính h(n) bằng cách chạy thuật toán tìm kiếm để tìm chi phí nhỏ nhất từ đỉnh n đến đích? Cho h(n) = 0?
- Heuristic hợp lệ thường được tạo ra bằng cách nới lỏng bài toán (làm cho bài toán đơn giản hơn), để dễ dàng tính chi phí nhỏ nhất từ n đến đích

Ví dụ: bài toán Sliding puzzle có thể nới lỏng thành luôn được di chuyển sang ô kế dù tại đó không trống

→ với hình bên, h(start) = 3+1+2+… = 18



Start State Goal State

Nguồn ảnh: UC Berkely. CS188 Intro to AI

Tiêu chuẩn lựa chọn Heuristic

- Heuristic càng chính xác thì càng giúp định hướng tốt, giảm số trạng thái phải mở
- Tuy nhiên, Heuristic quá chính xác có thể dẫn đến thời gian tính lâu và có thể gây ảnh hưởng đến tốc độ của thuật toán

➔ Heuristic tốt có thể chỉ cần chính xác vừa phải để tính nhanh mà vẫn định hướng tương đối tốt.

Cải tiến các thuật toán tìm kiếm

- Ý tưởng: không mở lại các trạng thái đã mở
- Thực hiện: Dùng 1 biến ClosedSet để lưu lại các trạng thái đã mở, và mỗi khi muốn mở một trạng thái mới thì kiểm xem nó đã có trong ClosedSet chưa – nếu đã có thì bỏ qua.
- Thuật toán tìm kiếm dùng ClosedSet còn được gọi là Graph Search (không dùng ClosedSet được gọi là Tree Search)

Thuật toán tìm kiếm Graph Search

- Khởi tạo frontier là kế hoạch ứng với trạng thái bắt đầu và ClosedSet rỗng
- Trong khi frontier chưa rỗng:
  - Lấy một kế hoạch ra khỏi frontier theo **một chiến lược nào đó**
  - Nếu kế hoạch này đi tới đích: Xong – Kết thúc tìm kiếm!
  - Nếu trạng thái cuối của kế hoạch này chưa có trong ClosedSet thì đưa trạng thái vào ClosedSet và dùng hàm successor xác định + đưa các kế hoạch kế vào frontier
- Frontier rỗng mà chưa tìm ra thì nghĩa là không thể tìm thấy lời giải

Bài tập

1. Chạy trên giấy các thuật toán Greedy và A* với đồ thị các nút giao thông quanh trường (tự chỉ định đỉnh Start & Goal)
2. Chạy trên giấy các thuật toán tìm kiếm đã học với đồ thị các nút giao thông quanh trường – có dùng ClosedSet