

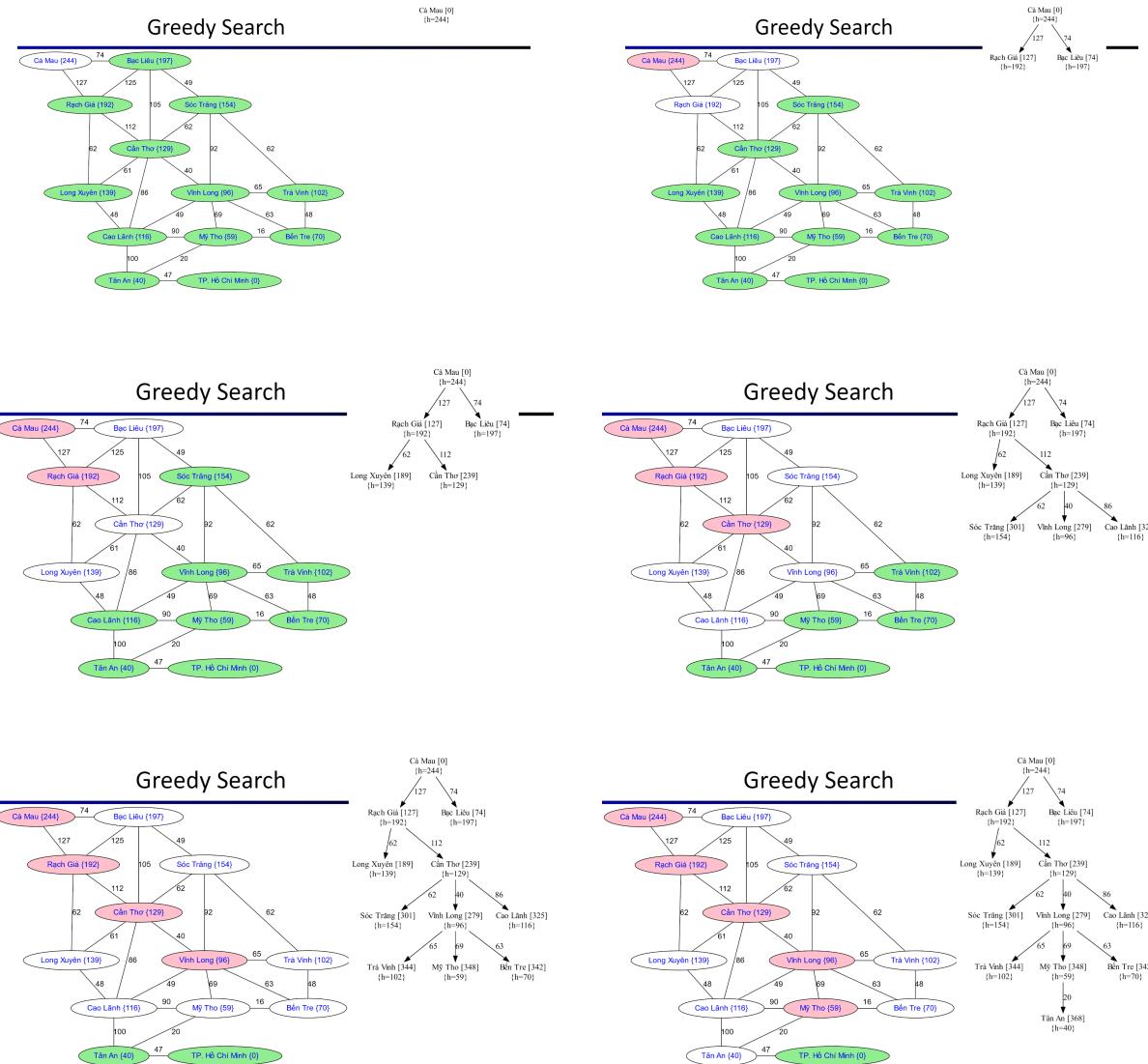
## 03 - Informed Search

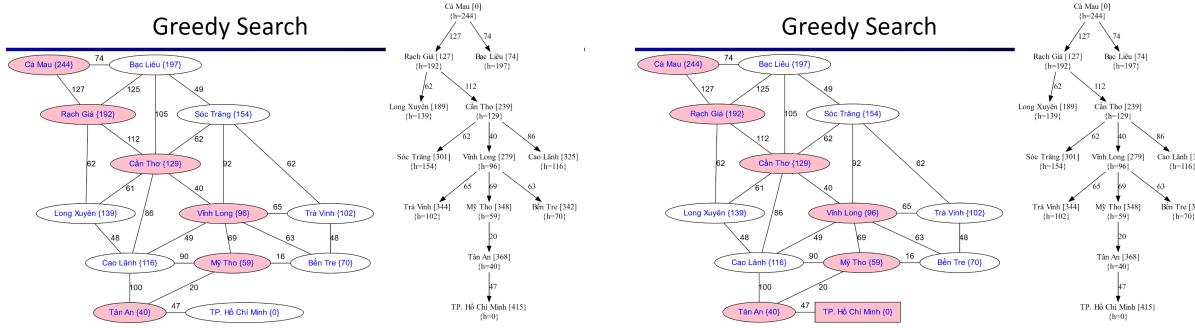
## Heuristic

- Là 1 hàm số, truyền vào trạng thái  $\rightarrow$  ra ước lượng chi phí từ trạng thái đó đến trạng thái mục tiêu
  - Vì sao phải ước tính chi phí thay vì trả chi phí thực sự? Vì để tìm được chi phí thực sự ta cần tìm ra đường đi trước mới tính chi phí. Do đó, khi chưa giải được bài toán thì ta cần phải ước tính
  - Nếu gọi hàm  $Heu$  trên trạng thái mục tiêu  $\rightarrow$  trả về 0

# Greedy Search

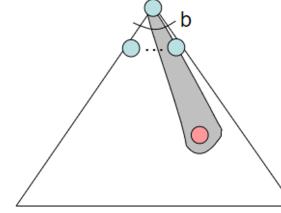
- Chiến lược: chọn node có giá trị heuristic nhỏ nhất (có vẻ gần với mục tiêu nhất) → Heuristic: ước tính chi phí từ node hiện tại tới node mục tiêu



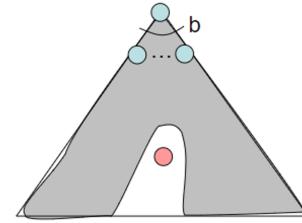


## Phân tích Greedy Search

- KHÔNG** tìm được lời giải tối ưu nhưng nhanh vì không vét cạn toàn bộ cây trạng thái
- Chạy tốt hay không phụ thuộc vào Heuristic



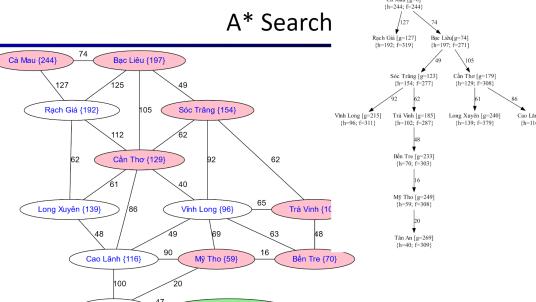
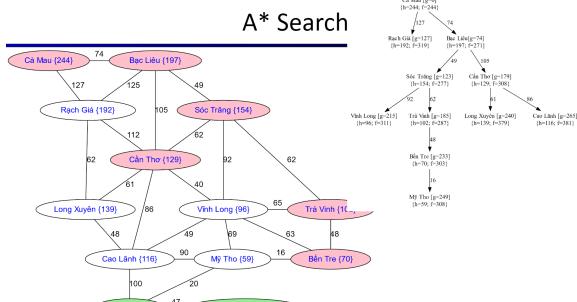
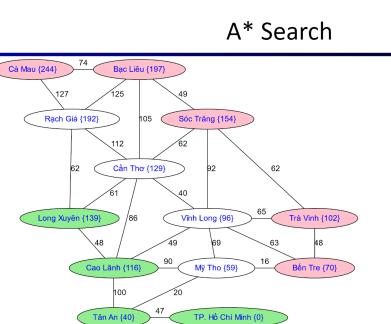
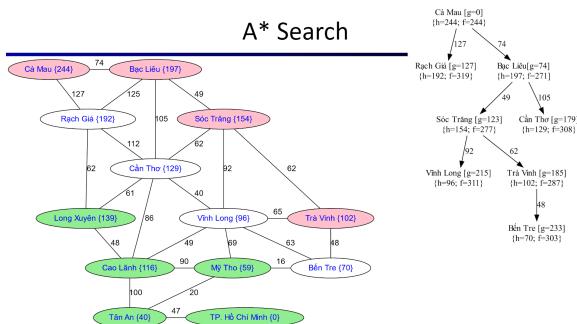
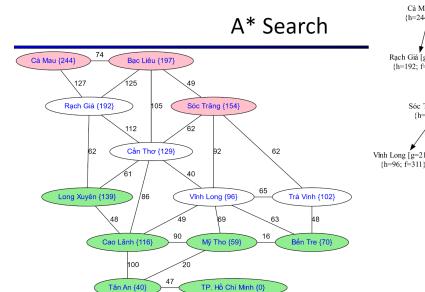
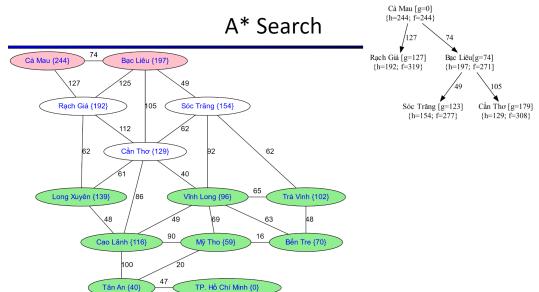
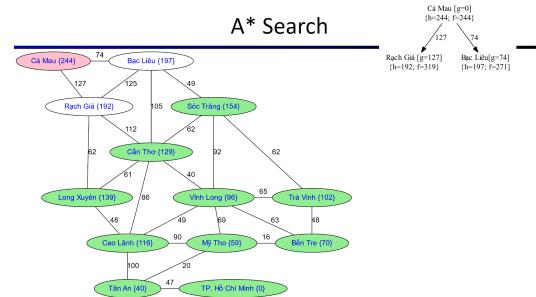
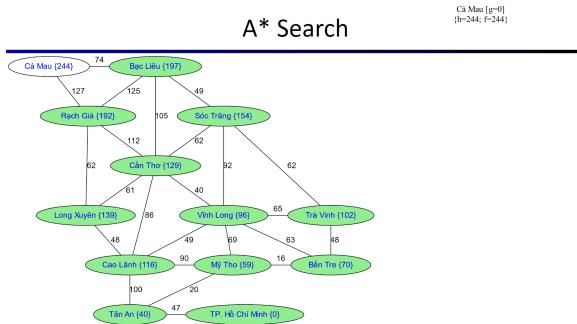
- Nếu hàm Heuristic thì thuật toán sẽ chạy lung tung nhưng không tìm thấy lời giải

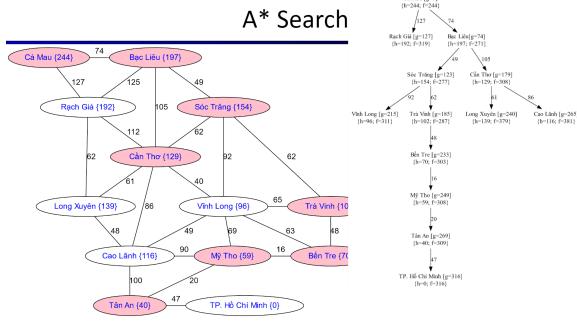


- Thuật toán Gradient Descent là 1 dạng greedy search với
  - Hàm Heuristic là ngược chiều vector gradient (vector trỏ về hướng hàm loss tăng nhanh nhất) do cực trị là hướng ngược chiều với vector gradient.
  - Dù cực trị không phải là tối ưu nhưng xài được, hướng đi tốt
- Khuyết điểm: dễ bị hụt mất lời giải tối ưu

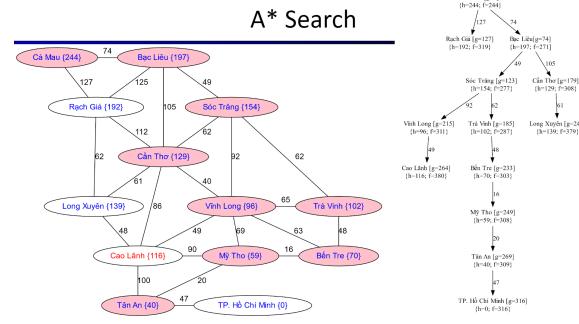
## A\* Search

- Chiến lược: mỗi node tính tổng giữa chi phí thực sự từ node gốc tới node đang xét+ chi phí ước tính từ node đang xét tới node mục tiêu → chọn node có tổng nhỏ nhất
- Ví dụ:
  - g**: chi phí thực sự từ state bắt đầu đến state đang xét
  - h**: chi phí ước tính từ state đang xét tới state mục tiêu
  - f**: tổng của **g** và **h**





chưa dừng do  $f$  HCM >  $f$  vĩnh long  $\rightarrow$  bung node vĩnh long



$f$  cao lãnh >  $f$  HCM  $\rightarrow$  bung node HCM  $\Rightarrow$  node mục tiêu  
 $\rightarrow$  lúc này mới dừng

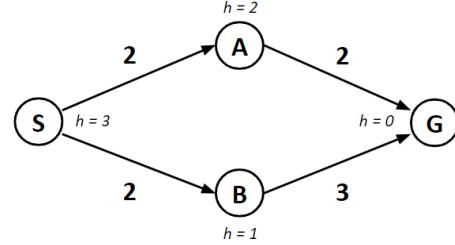
## Phân tích A\*

- Thuật toán tìm được lời giải tối ưu mà vẫn còn những node chưa xử lí tới  $\Rightarrow$  chi phí xử lí của A\* < chi phí xử lí của UCS
- A\* có đảm bảo được lời giải tối ưu không? **KHÔNG** vì có sự ảnh hưởng từ heuristic do H tính giá trị ước tính

**⇒ làm sao để đảm bảo được?**

- Vấn đề 1: khi nào nên dừng A\*?

- Xét vấn đề khi thêm các node vào hàng đợi và trong hàng đợi có node target, vì sao ta không dừng thuật toán mà lại phải tiếp tục?
  - $f(A) = 2 + 2 = 4$
  - $f(B) = 2 + 1 = 3$

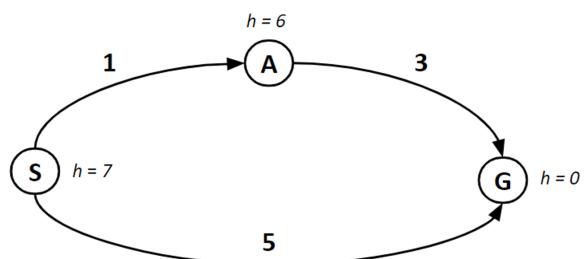


$\rightarrow$  nếu stop sau khi tìm được  $f(B)$  thì  $f(G)$  lúc này = 5 trong khi  $f(A)$  khi đi qua A  $f(G)=4$  dù  $f(A) > f(B)$   
 $\rightarrow$  vẫn phải chạy tiếp khi có node target trong hàng đợi

- Vấn đề 2: khi nào A\* tối ưu?

- $f(A) = 1 + 6 = 7$
- $f(G) = 5 + 0 = 5$

$\rightarrow$  mở  $f(G)$   $\rightarrow$  sai vì đây không là optimal



$\Rightarrow$  nguyên nhân: do hàm Heuristic dở (khoảng cách thực sự từ A đến G là 3 nhưng ước lượng = 6)

$\Rightarrow$  overestimate (bi quan)

$\Rightarrow$  Admissible Heuristic

## Admissible Heuristics - tính lạc quan

- Heuristic phải đảm bảo **chi phí ước lượng luôn ≤ chi phí thực sự**

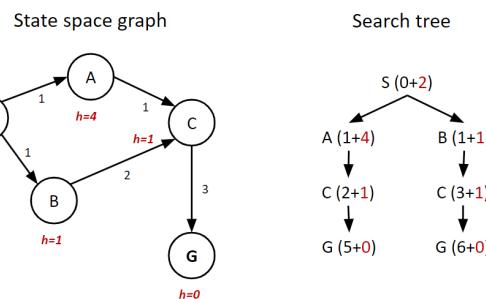
$$0 \leq h(n) \leq h^*(n)$$

- Nếu H bi quan  $\rightarrow h >> f >>$  (quá lớn)  $\rightarrow$  khiến những node nằm trên đường đi tối ưu quá lớn  $\rightarrow$  thuật toán sẽ bỏ qua nó
- Nếu H lạc quan  $\rightarrow$  chi phí ước lượng luôn nằm dưới chi phí thực sự  $\rightarrow f$  của trạng thái nằm trên đường đi tốt sẽ ko lớn hơn  $f$  nằm trên đường đi xấu  $\rightarrow$  sẽ chạm được tới đường đi tốt

## Tree Search vs Graph Search trong A\*

- Xét trường hợp:

- $S \rightarrow B (f(B) < f(A)) \rightarrow C \rightarrow G \rightarrow A \Rightarrow$   
không xét tiếp C được vì C đã từng  
được bung  $\rightarrow$  C đã nằm trong tập đóng
- Dù đã đảm bảo H lạc quan nhưng với  
tập đóng graph search vẫn có thể lạc  
mất lời giải tối ưu



- Đối với Tree Search, một Heuristic lạc quan đã đủ. Nhưng đối với Graph Search, Heuristic lạc quan chưa đủ để đảm bảo lời giải tối ưu
  - Tuy nhiên lại không thể dùng Tree Search vì có rất nhiều trạng thái lặp đi lặp lại
  - *Tree Search đang đầy đủ, nếu dùng Graph Search thì có khiến mất tính đầy đủ không do khả năng chặt nhánh của nó? KHÔNG*  
vì vẫn còn 1 nhánh khác đi qua trạng thái đó  $\rightarrow$  nhưng lại có khả năng phá đi **tính tối ưu**
- $\Rightarrow$  Cần 1 tính chất mạnh mẽ hơn: tính nhất quán - Consistency

## Consistency of Heuristics - tính nhất quán

- Ý tưởng: **chi phí ước lượng ≤ chi phí thực sự**
  - Tính lạc quan: giá trị heuristic ≤ chi phí thực sự tới đích

$$h(A) \leq \text{chi phí thực sự từ A đến G}$$

- Tính nhất quán: giá trị **cạnh** heuristic ≤ chi phí thật sự mỗi **cạnh**

$$h(A) - h(C) \leq \text{chi phí A đến C}$$

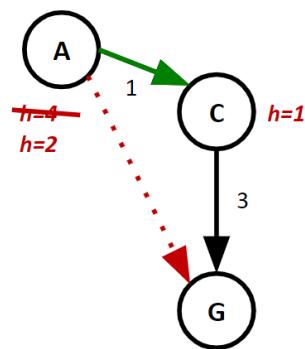
- Ví dụ:

Với  $h(A) = 4$

Xét tính lạc quan:  $h(A) = 4 \leq cost(A - G) = 4 \rightarrow$  thỏa

Xét tính nhất quán:  $h(A) - h(C) = 4 - 1 = 3 \leq cost(A - C) = 1 \rightarrow$  không thỏa

⇒ Heuristic lạc quan nhưng không nhất quán



Với  $h(A) = 2$ :

Xét tính lạc quan:  $h(A) = 2 \leq cost(A - G) = 4 \rightarrow$  thỏa

Xét tính nhất quán:  $h(A) - h(C) = 2 - 1 = 1 \leq cost(A - C) = 1 \rightarrow$  thỏa

- **Hệ quả:** giá trị  $f$  đọc đường đi sẽ không giờ giảm

$$h(A) \leq cost(A - C) + h(C)$$

- Tính tối ưu:

- Tree Search:
  - A\* sẽ tối ưu khi Heuristic có tính lạc quan
  - UCS là trường hợp đặc biệt với  $h=0$
- Graph Search:
  - A\* sẽ tối ưu khi Heuristic có tính nhất quán
  - UCS tối ưu (do  $h=0$  thì sẽ nhất quán)

- Trong đa số trường hợp, những heuristic **lạc quan** có xu hướng **nhất quán**

## A\* vs UCS

- UCS mở mọi hướng
- A\* mở về nhiều hướng nhưng có 1 hướng tập trung sức lực tìm kiếm nhiều hơn

