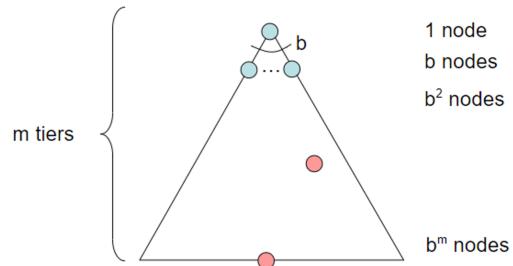


# 02 - Uninformed Search

## Các yếu tố phân tích thuật toán tìm kiếm

- Phân tích thuật toán tìm kiếm:
  - Thuật toán có đầy đủ hay không (complete):  
complete khi nó đảm bảo tìm được lời giải
  - Thuật toán có tối ưu hay không?
  - Độ phức tạp thời gian/không gian
- b: hệ số rẽ nhánh, m: độ sâu tối đa
- Lời giải có thể nằm ở vị trí bất kỳ trên cây
- Số lượng node tối đa:  $O(b^m)$



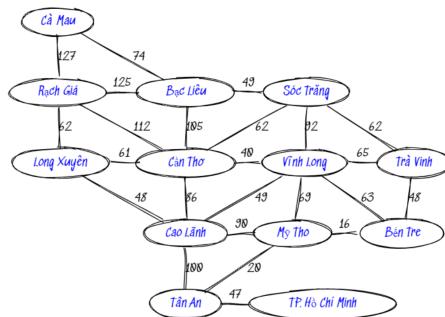
## Tree Search

- Mỗi node trên cây đại diện cho 1 lời giải
  - Những node lá tạo thành đường biên giới (fringe/frontier): node đang được chờ xử lí
- ⇒ Tất cả thuật toán tìm kiếm đều có khung như sau

```
function TREE-SEARCH( problem, strategy) returns a solution, or failure
    initialize the search tree using the initial state of problem
    loop do
        if there are no candidates for expansion then return failure
        choose a leaf node for expansion according to strategy
        if the node contains a goal state then return the corresponding solution
        else expand the node and add the resulting nodes to the search tree
    end
```

- Khởi tạo cây tìm kiếm, node gốc là trạng thái bắt đầu
- Node gốc vào danh sách chờ (chỉ đang có 1 node duy nhất là node gốc)
- Kiểm tra trong hàng đợi có rỗng hay không?
  - Nếu không rỗng thì lấy node ra, nhìn vào trạng thái
    - nếu node chứa trạng thái mục tiêu ⇒ tìm được đường đi, kết thúc
    - nếu không phải target ⇒ xem từ trạng thái đó đi qua bao nhiêu trạng thái kế tiếp (có bao nhiêu hành động có thể làm) ⇒ những trạng thái kế tiếp đó sẽ biến thành các node con mới ⇒ thay vào danh sách chờ ⇒ tiếp tục vòng lặp
  - Nếu rỗng ⇒ đã vét cạn cây tìm kiếm ⇒ bài toán không có lời giải
- Ví dụ:

- Bung cà mau  $\rightarrow$  queue = {rạch giá, bạc liêu}
  - chọn bạc liêu bung  $\rightarrow$  queue = {rạch giá, cà mau, cần thơ, sóc trăng, bạc liêu}
  - nếu chọn cà mau hay bạc liêu  $\rightarrow$  rơi vô loop dài vô tận
- $\Rightarrow$  chiến thuật mở node ành hưởng tới lời giải
- $\Rightarrow$  Graph Search



## Graph Search

```

function GRAPH-SEARCH(problem, fringe) return a solution, or failure
  closed  $\leftarrow$  an empty set
  fringe  $\leftarrow$  INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
  loop do
    if fringe is empty then return failure
    node  $\leftarrow$  REMOVE-FRONT(fringe)
    if GOAL-TEST(problem, STATE[node]) then return node
    if STATE[node] is not in closed then
      add STATE[node] to closed
      for child-node in EXPAND(STATE[node], problem) do
        fringe  $\leftarrow$  INSERT(child-node, fringe)
      end
    end
  end

```

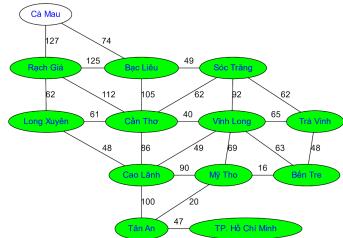
- Có tập đóng: kiểm tra trạng thái đã xét chưa
- Kiểm tra trong hàng đợi có rỗng hay không?
  - Nếu không rỗng thì lấy node ra, nhìn vào trạng thái
    - nếu node chứa trạng thái mục tiêu  $\Rightarrow$  tìm được đường đi, kết thúc
    - nếu không phải target  $\Rightarrow$  **kiểm tra trạng thái có nằm trong tập đóng hay không?**
      - nếu không  $\rightarrow$  xử lí bằng cách bung node  $\rightarrow$  xem từ trạng thái đó đi qua bao nhiêu trạng thái kế tiếp  $\Rightarrow$  thảy trạng thái tiếp theo vào danh sách chờ  $\Rightarrow$  **đánh dấu trạng thái đã xử lí (đưa vào tập đóng)**
      - nếu có  $\rightarrow$  không bung node  $\rightarrow$  di chuyển sang node khác
  - Nếu rỗng  $\Rightarrow$  đã vét cạn cây tìm kiếm  $\Rightarrow$  bài toán không có lời giải

## DFS

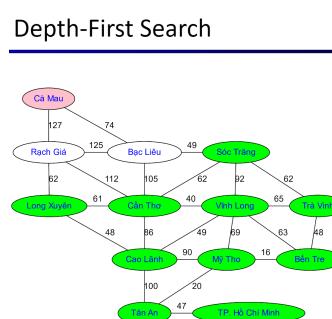
- Chiến lược: Ở mỗi lần chọn node trong queue  $\rightarrow$  chọn node ở sâu nhất (tầng sâu nhất) để xử lí
- Ví dụ: cây bên dưới hàng đợi không chứa những **node đã xử lí** (thực tế là có node đã xử lí, sẽ được loại bỏ khi xét tới)
  - Nếu các node nằm cùng độ sâu, thường sẽ chọn node trái nhất
  - Node trắng: node đang chờ được xử lí
  - Node đỏ: node đã xử lí

- Node xanh: chưa khám phá tới

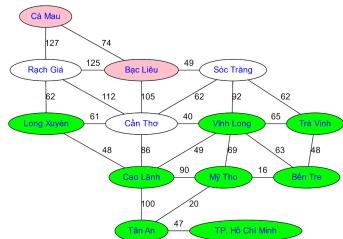
### Depth-First Search



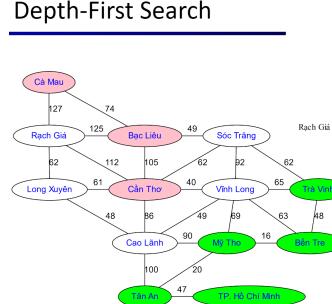
### Depth-First Search



### Depth-First Search



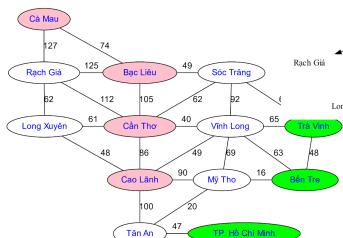
### Depth-First Search



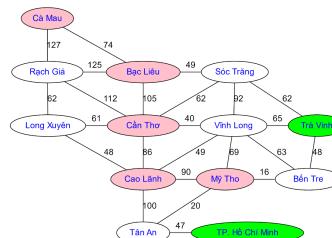
bung bạc liêu → {cần thơ, rạch giá, cà mau, sóc trăng}.  
Đáng lẽ có node **cà mau** trên cây nhưng do node này đã xử lí rồi, khi chạm tới sẽ bị bỏ qua và slide nhô nên trên cây trên không có

→ {bạc liêu, rạch giá, cà mau, sóc trăng, vĩnh long, cao lanh, long xuyên}

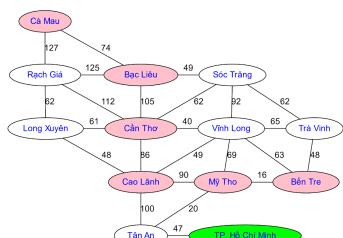
### Depth-First Search



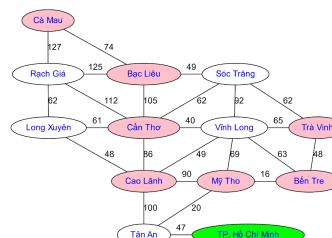
### Depth-First Search



### Depth-First Search

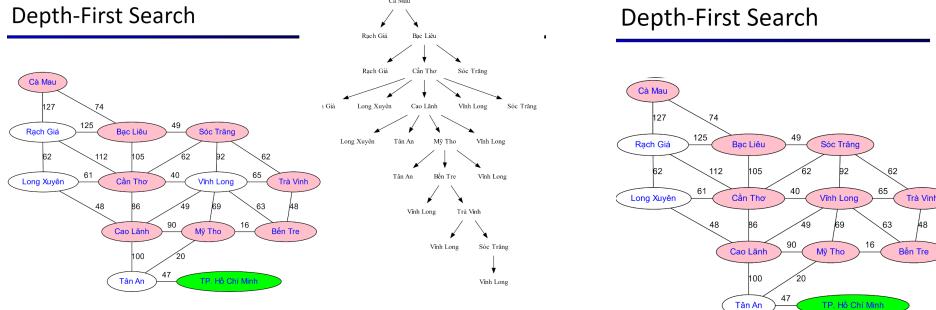


### Depth-First Search

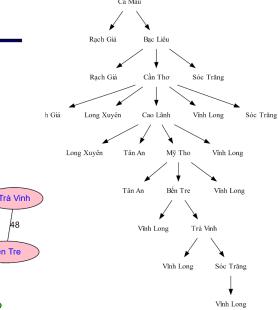


bung trà vinh → + {sóc trăng, vĩnh long, bến tre}

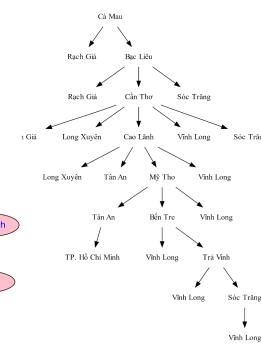
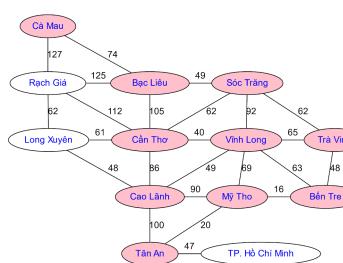
### Depth-First Search



### Depth-First Search



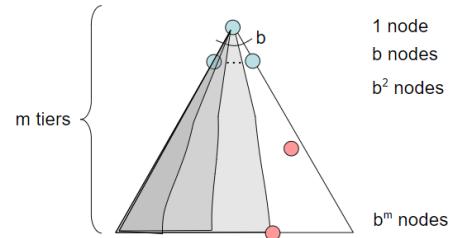
### Depth-First Search



bung Tân An → xuất hiện TPHCM → **CHƯA DỪNG** : phải lấy node ra hàng đợi mới dừng

## Phân tích DFS

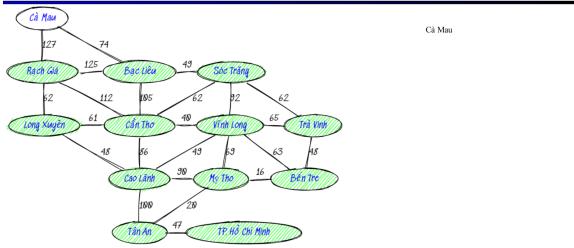
- Nếu DFS tìm thấy lời giải nằm ở trái nhất, và sâu nhất - **TH XẤU NHẤT** (do ta chọn node trái nhất để bung khi cùng độ sâu) mà lời giải tối ưu nằm ở vị trí khác bên phải và nông hơn  $\Rightarrow$  Độ phức tạp thời gian:  $O(b^m)$
- Độ phức tạp không gian: liên quan đến kích thước hàng đợi có thể lớn đến mức nào  $\Rightarrow$  Độ phức tạp không gian:  $O(bm)$
- Thuật toán có đầy đủ không? **KHÔNG**
  - Nếu không xử lý trạng thái trùng lặp  $\rightarrow$  rơi vào vòng lặp vô tận
  - Nếu tầng cây có giới hạn**  $\rightarrow$  đảm bảo tìm ra lời giải  $\rightarrow$  **CÓ**
- Đây có phải đường đi ngắn nhất? **KHÔNG** vì không complete



## BFS

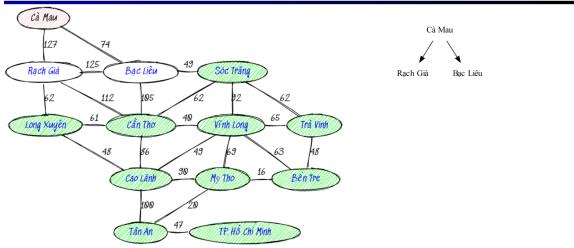
- Chiến lược: Ở mỗi lần chọn node trong queue  $\rightarrow$  chọn node ở nông nhất để xử lí
- Ví dụ:

### Breadth-First Search



bung bạc liêu

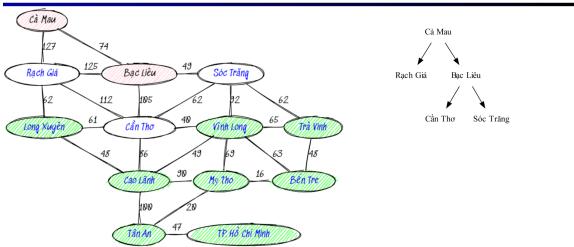
### Breadth-First Search



→ {cà mau, rạch giá, cần thơ, sóc trăng}

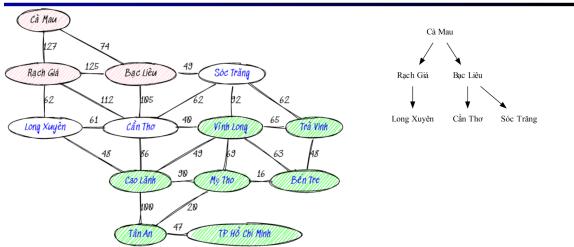
- cà mau: đã xử lí → bỏ qua
- rạch giá: trước đó đã dụng rạch giá tầng trên → đưa RG vào tập đóng → đã xử lí

### Breadth-First Search



bung rạch giá (vì đang ở nông nhất)

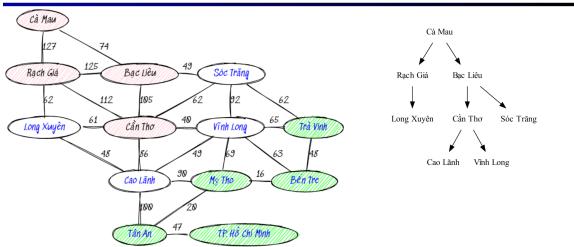
### Breadth-First Search



→ {cà mau, bạc liêu, cần thơ, long xuyên}

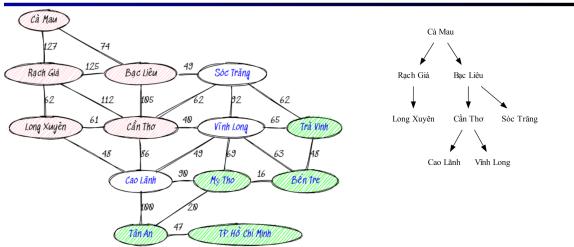
- cần thơ: vì có node cần thơ khác cùng cấp

### Breadth-First Search



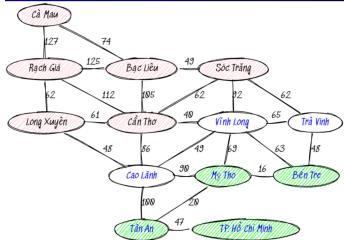
bung cần thơ

### Breadth-First Search



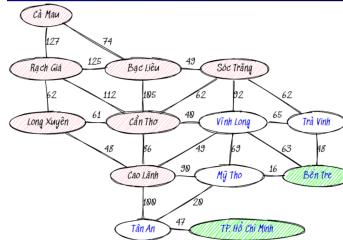
bung long xuyên

### Breadth-First Search



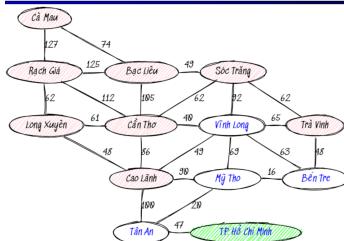
bung sóc trăng

### Breadth-First Search



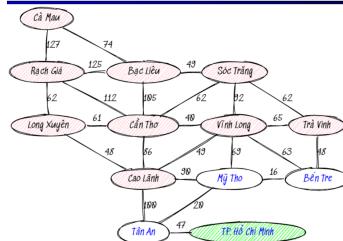
bung cao lanh

### Breadth-First Search



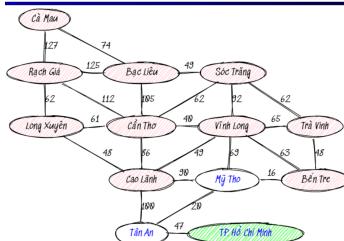
bung trà vinh

### Breadth-First Search



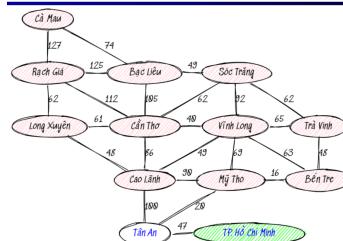
bung vĩnh long

### Breadth-First Search



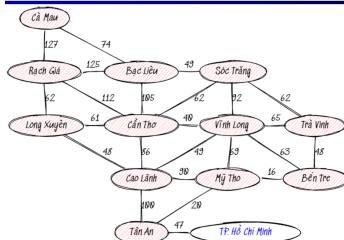
bung bến tre

### Breadth-First Search



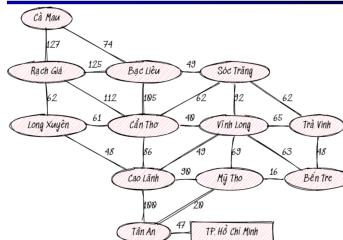
bung mỹ tho

### Breadth-First Search



bung tân an

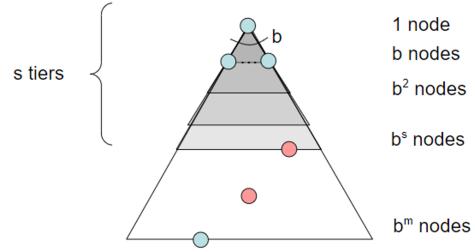
### Breadth-First Search



bung tp.hcm → **CHUA** dừng lại → phải lấy node ra mới  
dừng lại

## Phân tích BFS

- Trong trường hợp xấu nhất: lời giải nằm tầng đáy  
⇒ Độ phức tạp thời gian:  $O(b^m)$
- Độ phức tạp không gian: ⇒ Độ phức tạp không gian:  $O(b^m)$
- Thuật toán có đầy đủ không? **CÓ** cho dù cây có những nhánh vô tận vì lời giải nằm ở tầng hữu hạn nào đó



- Thuật toán có tối ưu không? **KHÔNG** nhưng sẽ tìm được lời giải có **số bước đi ít nhất**

Vì sao người ta không dùng thuật toán này để giải quyết cái bài toán không cần chi phí bước đi?

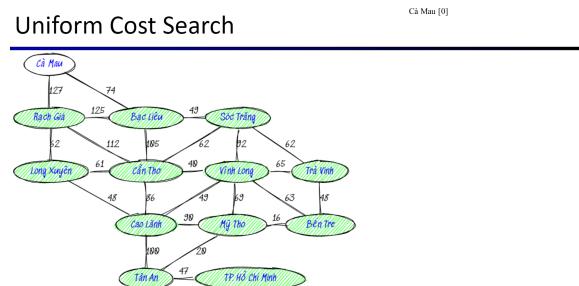
⇒ Vì độ phức tạp thời gian và không gian quá lớn

## Iterative Deepening

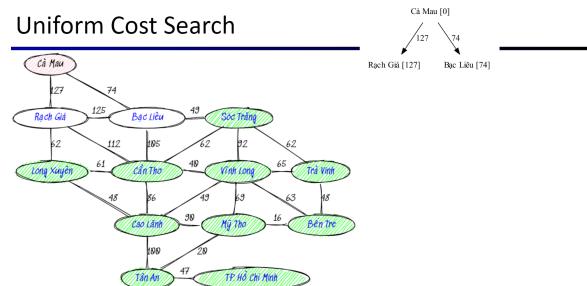
- Idea: lấy độ phức tạp không gian của DFS và khả năng tìm được lời giải nồng của BFS
  - Chạy DFS → đào xuống 1 tầng → chạy BFS để tìm lời giải trong tầng đó → không tìm được → ngừng → chạy lại DFS → đào xuống 2 tầng → tìm được lời giải bằng BFS tầng đó → trả về lời giải
- ⇒ chạy đi chạy lại nhưng sự dư thừa này chấp nhận được

## UCS

- Có tốn chi phí tính toán
- Chiến lược: cứ mỗi node cho vào hàng đợi sẽ tính chi phí đi từ node gốc tới node cho vào hàng đợi → mỗi lần chọn node để bung thì sẽ chọn node có tổng chi phí nhỏ nhất
- Ví dụ:

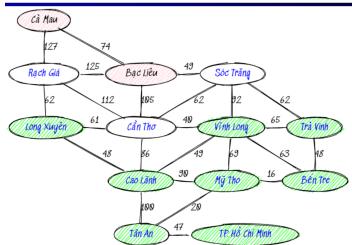


→ {cà mau:0}



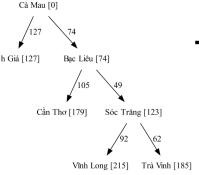
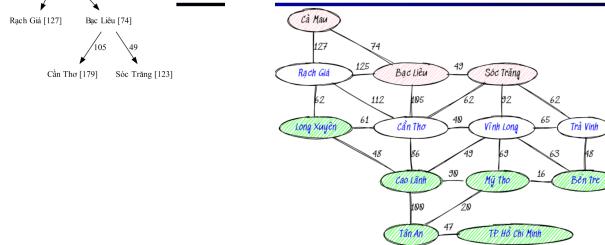
bung cà mau → {rạch giá-127, bạc liêu-74}

### Uniform Cost Search



### Uniform Cost Search

### Uniform Cost Search

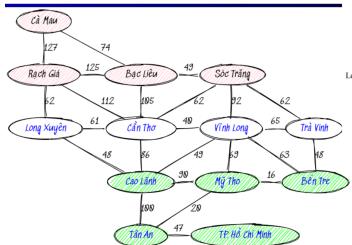


bung bạc liêu min → + {rạch giá-127, rạch giá-74+125, cần thơ-74+105, sóc trăng-74+49}

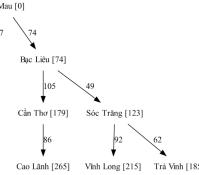
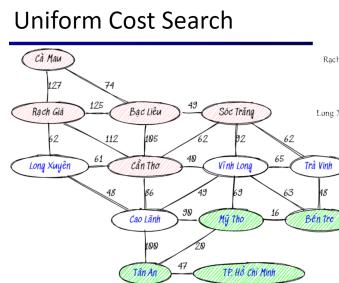
- cà mau vẫn nằm trong queue nhưng vì trong tập đóng nên sẽ skip
- rạch giá 127 sẽ được xử lí trước rạch giá 199 → sẽ không bung rạch giá 199 nhưng trên cây vẫn có

bung sóc trăng → + {vịnh long-123+92, trà vinh-123+62}

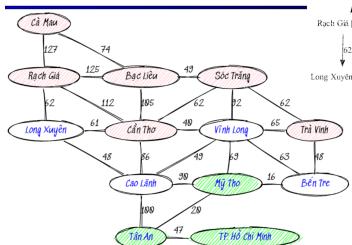
### Uniform Cost Search



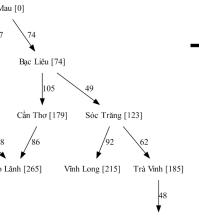
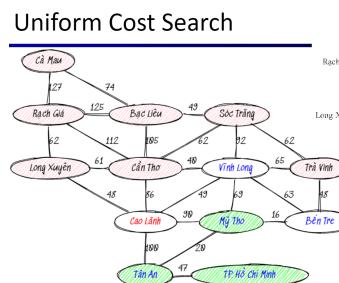
### Uniform Cost Search



### Uniform Cost Search

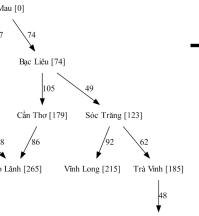
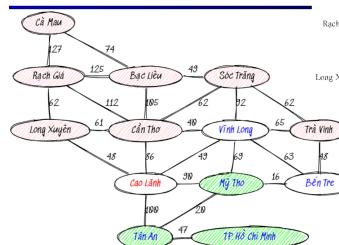


### Uniform Cost Search



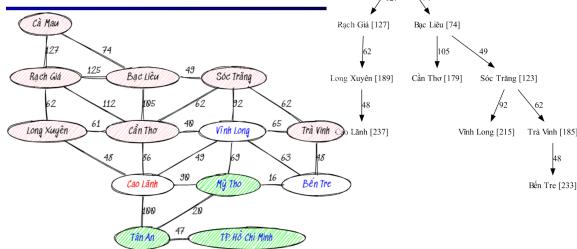
bung trà vinh → + {bến tre-185+48}

### Uniform Cost Search

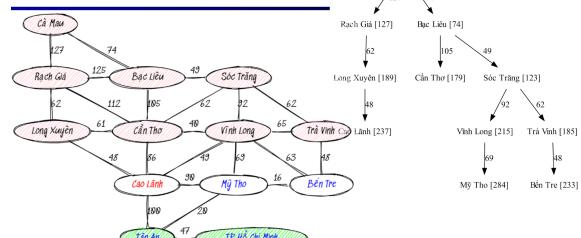


bung long xuyên → đang có node cao lãnh 265 (từ cần thơ) mà long xuyên đến cao lãnh bé hơn - 189 + 48 = 237 → đã chạm tới cao lãnh 237 trước 265 nên skip 265  
(trên cây search vẫn có 265)

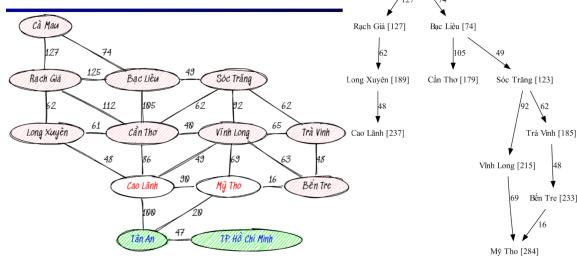
## Uniform Cost Search



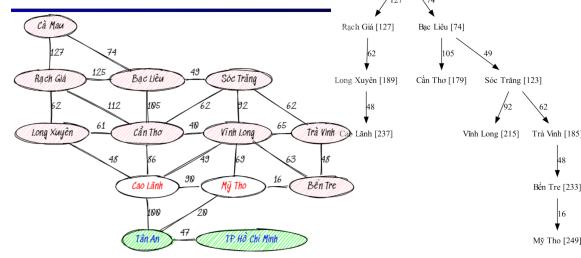
## Uniform Cost Search



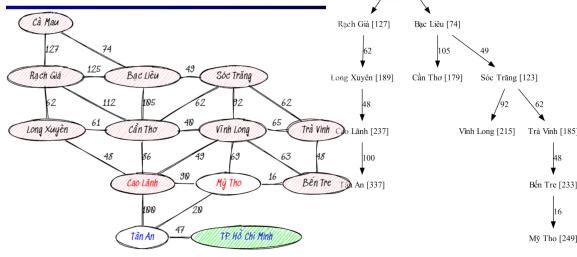
## Uniform Cost Search



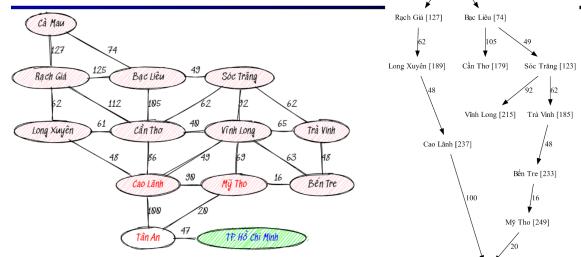
## Uniform Cost Search



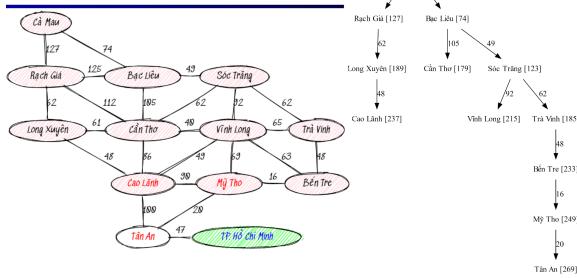
## Uniform Cost Search



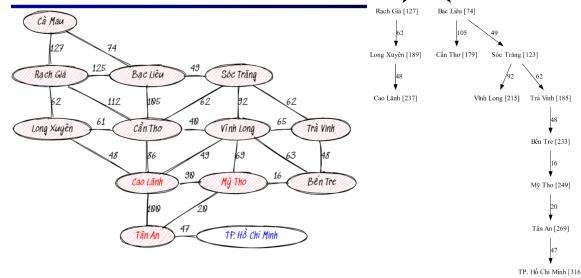
## Uniform Cost Search

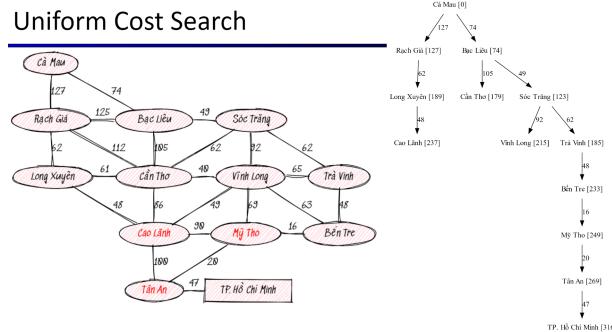


## Uniform Cost Search



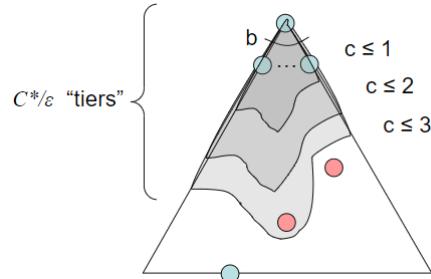
## Uniform Cost Search





## Phân tích UCS

- Mỗi 1 node có chứa thông tin tổng chi phí từ bắt đầu tới node đang xét, khi chọn sẽ chọn node tốn chi phí thấp nhất  $\Rightarrow$  cuối cùng sẽ đưa ra lời giải có tổng chi phí ít nhất  $\Rightarrow$  **lời giải tối ưu**
- Giống Dijkstra
- Nếu chi phí tối ưu của lời giải tối ưu là  $C^*$ , chi phí trung bình mỗi cạnh là  $\epsilon$   $\Rightarrow$  lời giải tìm được nằm ở  $\frac{C^*}{\epsilon}$
- Độ phức tạp thời gian và không gian  $O(b \frac{C^*}{\epsilon})$
- Thuật toán có đầy đủ không? **Có**
  - Thuật toán đảm bảo có lời giải nếu chi phí mỗi hành động là **số dương**
- Thuật toán có tối ưu không? **Có**
- Khuyết điểm: chi phí về không gian và thời gian lớn vì nó phải xét toàn bộ cung đường có thể xảy ra dù **1 vài đường chắc chắn sẽ không có lời giải**  $\Rightarrow$  tìm kiếm không có thông tin, tức không xét độ quan trọng của node hiện tại với node mục tiêu



## Nhận xét

- DFS: chọn node tầng sâu nhất: thêm vào sau cùng nhưng lấy ra đầu tiên  $\Rightarrow$  hàng đợi sẽ là **stack**
- BFS: chọn node tầng nông nhất  $\Rightarrow$  hàng đợi sẽ là **queue**
- UCS: chọn node theo chi phí  $\Rightarrow$  hàng đợi sẽ là **priority queue**