

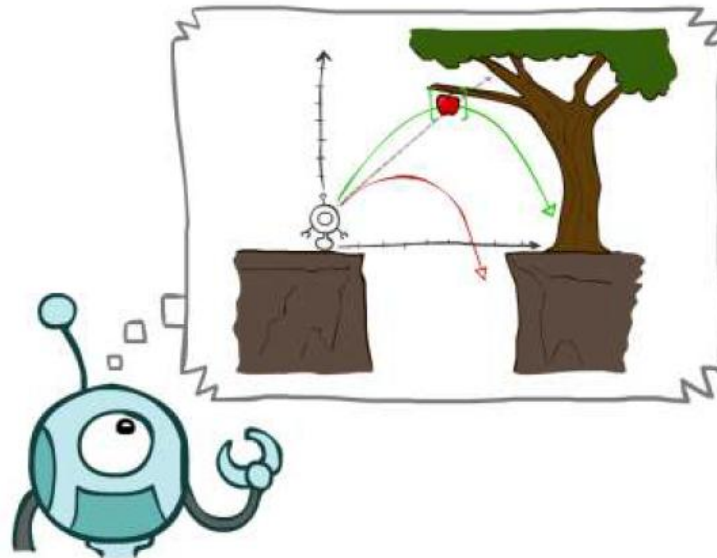
BÀI TOÁN TÌM KIẾM

Nội dung

- Agent
- Bài toán tìm kiếm
- Tìm kiếm mù (uninformed search)
- Tìm kiếm có hiểu biết (informed search)

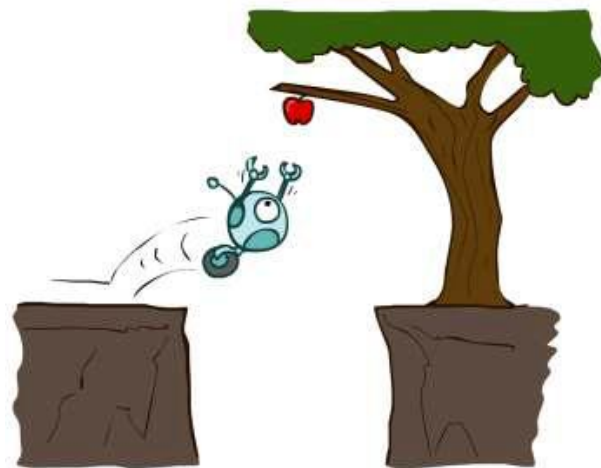
Agent

- Agent là một thực thể độc lập, có khả năng **quan sát** môi trường (environment, world) và có **hành động** tương ứng **nhằm đạt được mục tiêu**.



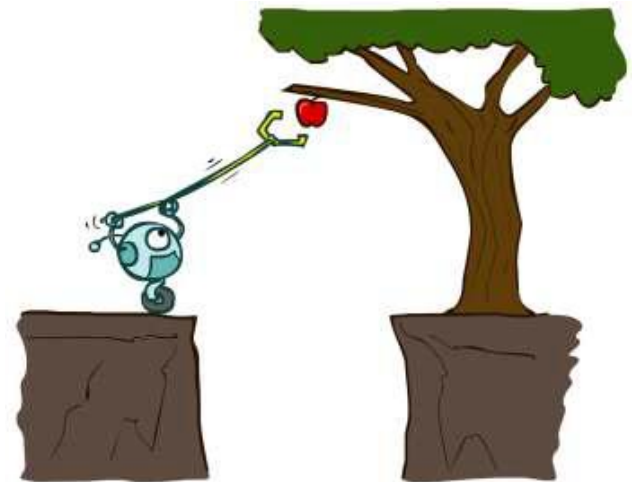
Reflex agent

- Lựa chọn hành động dựa trên tri giác hiện tại (hoặc trí nhớ)
- Biết được hoặc nhớ được trạng thái hiện tại của môi trường.
- Không cân nhắc đến hậu quả về sau, khi lựa chọn hành động



Planning agent

- Luôn tự hỏi “chuyện gì sẽ xảy ra nếu...”
- Ra quyết định dựa trên hậu quả để lại của hành động
- Biết được cách thức/mô hình mà môi trường phản hồi lại với các hành động
- Phải phát biểu có hệ thống đích đến.



Nội dung

- Agent
- **Bài toán tìm kiếm**
- Tìm kiếm mù (uninformed search)
- Tìm kiếm có hiểu biết (informed search)

Bài toán tìm kiếm

- Một bài toán tìm kiếm bao gồm:

- Không gian trạng thái: S



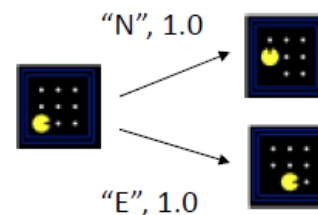
- Tập các hành động: Actions(s)

- Trạng thái bắt đầu

- Hàm kiểm tra trạng thái đích: IsEnd(s)

- Hàm trạng thái kế tiếp (successor function)

- Đi kèm với hành động và chi phí



- Lời giải của bài toán tìm kiếm là mỗi chuỗi các hành động (một kế hoạch) để chuyển được từ trạng thái bắt đầu sang trạng thái đích

Trạng thái (state)

- **Trạng thái của môi trường** bao gồm mọi chi tiết về môi trường



- **Trạng thái tìm kiếm** chỉ lưu giữ các chi tiết cần thiết cho việc lên kế hoạch hoặc ra quyết định

Bài toán: tìm đường

- Trạng thái: (x, y) , vị trí
- Hành động: N, S, E, W
- Trạng thái kế: cập nhật một thông tin vị trí
- Hàm kiểm tra đích: $IsEnd(x,y)$

Bài toán: ăn tất cả các dấu chấm

- Trạng thái: $\{(x, y); \text{các dot boolean}\}$
- Hành động: N, S, E, W
- Trạng thái kế: cập nhật một thông tin vị trí và (có thể) cập nhật dot
- Hàm kiểm tra đích: Tất cả các dot boolean đều false

Không gian trạng thái

- Tập tất cả các trạng thái tìm kiếm có thể có tạo thành không gian trạng thái
- Kích thước của không gian trạng thái:
 - Mỗi trạng thái gồm N chi tiết $|S| = |D_1| \times |D_2| \times \dots \times |D_N|$
 - Mỗi chi tiết có miền giá trị là D_i
 - Kích thước của không gian trạng thái
- Không gian trạng thái càng lớn thì độ phức tạp bài toán càng cao

Không gian trạng thái

- Thông tin về môi trường

- Số vị trí của agent: 120
- Số dấu chấm: 30
- Số vị trí của một con ma: 12
- Agent có 4 lựa chọn: N, S, E, W

- Trạng thái môi trường

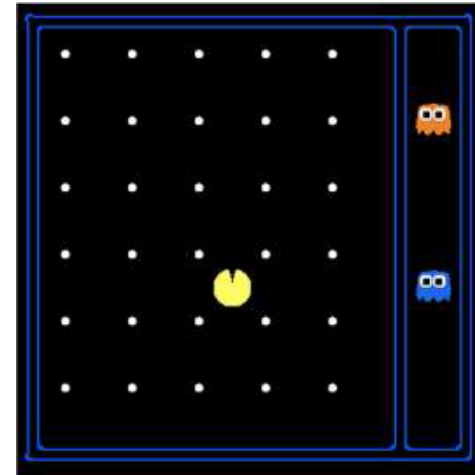
- $120 \times (2^{30}) \times 12 \times 12 \times 4$

- Bài toán tìm đường:

- 120

- Bài toán ăn tất cả dấu chấm

- $120 \times (2^{30})$

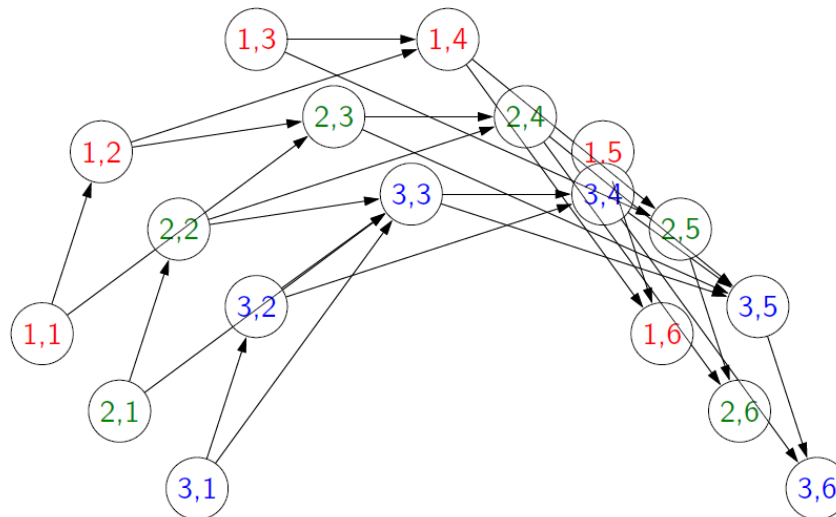


Câu hỏi

- Bài toán: tìm đường đi ngắn nhất từ thành phố 1 đến thành phố n , chỉ di chuyển tới. Chi phí từ thành phố i đến j là c_{ij}
 - Ràng buộc: Không được viếng thăm quá 2 thành phố lẻ.
- **Trạng thái:** (thành phố trước là lẻ?, thành phố hiện tại)

Câu hỏi

- Bài toán: tìm đường đi ngắn nhất từ thành phố 1 đến thành phố n , chỉ di chuyển tới. Chi phí từ thành phố i đến j là c_{ij}
 - Ràng buộc: cần viếng thăm ít nhất 3 thành phố lẻ.
- Trạng thái: ($\min(\text{số thành phố lẻ đi qua}, 3)$, thành phố hiện tại)



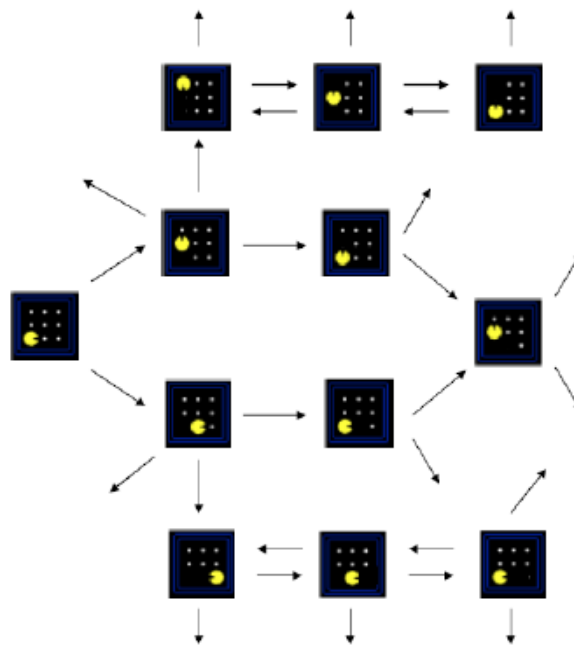
Câu hỏi

- Giả sử phải đi từ thành phố 1 đến thành phố n và cần phải viếng thăm **số thành phố lẻ nhiều hơn thành phố chẵn**. Tìm tập trạng thái nhỏ nhất?

Đồ thị trạng thái

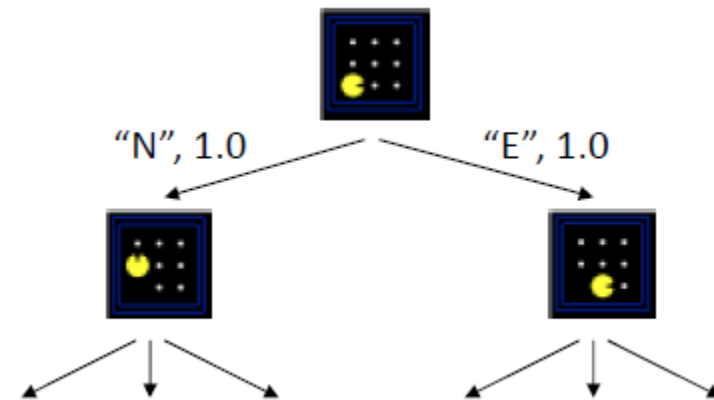
- Đồ thị trạng thái (state graph): một cách biểu diễn hình học cho bài toán tìm kiếm
 - Mỗi node tương ứng với một cấu hình của môi trường hay trạng thái
 - Các cạnh nối cho biết trạng thái kế tiếp, mỗi cạnh tương ứng với một hành động.
 - Hàm kiểm tra trạng thái đích là tập các node đích (thường chỉ có một)
- Trong đồ thị trạng thái, mỗi trạng thái chỉ xuất hiện duy nhất một lần.
- Thường chúng ta không xây dựng đầy đủ đồ thị trạng thái trên bộ nhớ vì rất lớn.

Đồ thị trạng thái



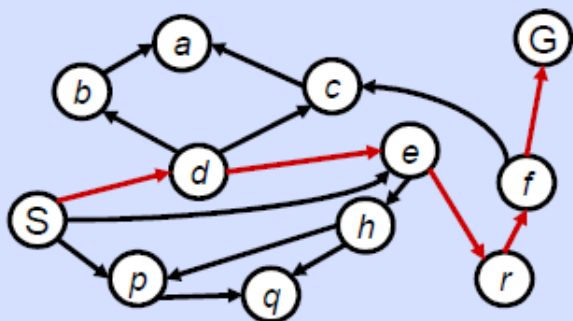
Cây tìm kiếm

- **Cây tìm kiếm (search tree):** là một cây thể hiện các hành động và đầu ra/hậu quả tương ứng
 - Node gốc tương ứng với trạng thái bắt đầu
 - Các node con tương ứng với trạng thái kế tiếp của node cha.
 - Node lá tương ứng với trạng thái đích
 - Mỗi lời giải tương ứng với một đường đi từ gốc đến node lá
- **Mỗi trạng thái có thể xuất hiện nhiều hơn 1 lần.**
- Thường không xây dựng đầy đủ cây tìm kiếm trong bộ nhớ vì rất lớn

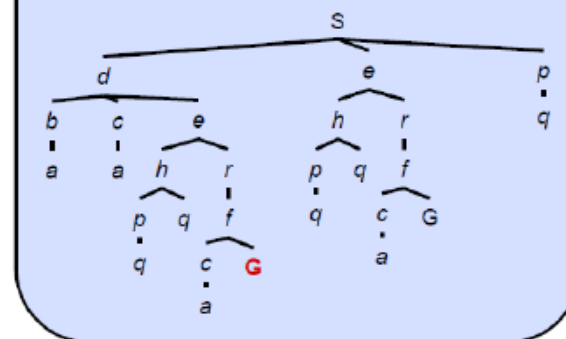


Đồ thị trạng thái vs cây tìm kiếm

Đồ thị trạng thái

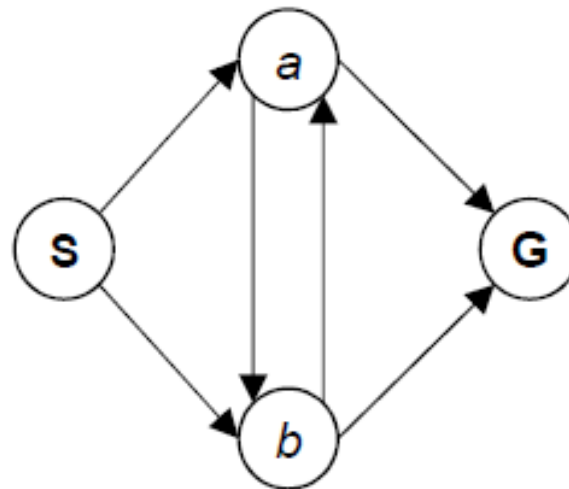


Cây tìm kiếm

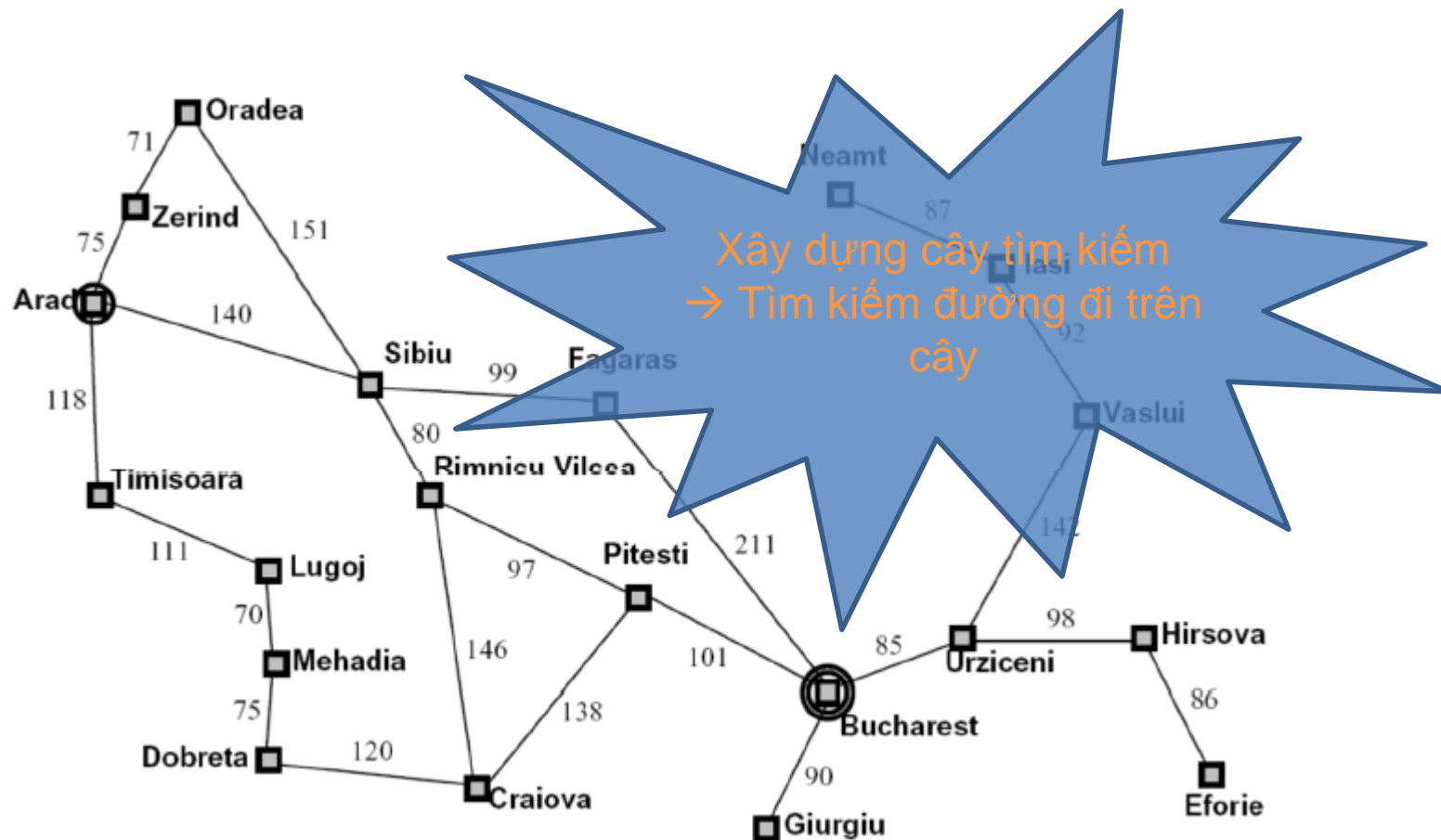


Câu hỏi

- Có bao nhiêu node trên cây tìm kiếm tương ứng với đồ thị trạng thái sau?



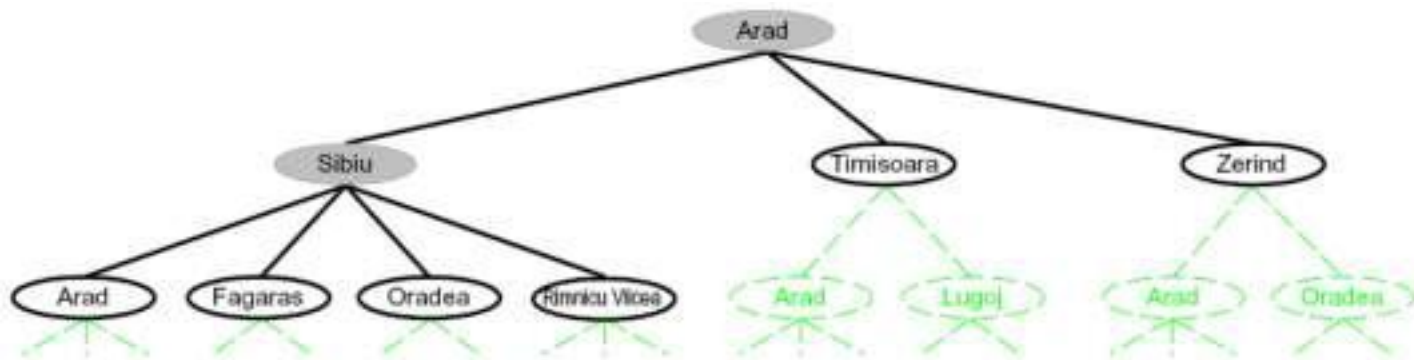
Tìm đường đi với bản đồ Romania



Tìm đường đi với bản đồ Romania

- Không gian trạng thái: các thành phố
- Hành động: di chuyển theo đường đi
- Hàm trạng thái kế tiếp: các thành phố kề nó
- Trạng thái bắt đầu: Arad
- Hàm kiểm tra trạng thái đích: $s == \text{Bucharest?}$

Tìm kiếm trên cây



- Mở rộng dần các node tiềm năng.
- Đảm bảo nơi lưu giữ thứ tự các node cần được xem xét (gọi là fringe)
- Cố gắng mở rộng ít node nhất có thể.

Thuật toán tìm kiếm trên cây

Input: **problem** – thông tin bài toán

strategy – chiến lược duyệt

Output: **solution** – nếu tồn tại lời giải; **failure** – nếu ngược lại

Khởi tạo cây tìm kiếm với trạng thái bắt đầu của **problem**

Loop

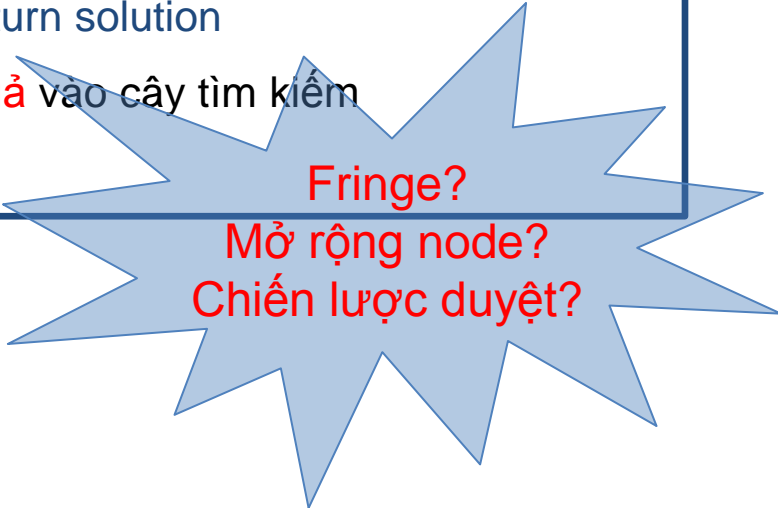
If không còn ứng viên để mở rộng then return failure

Chọn một node lá để mở rộng dựa theo **strategy**

If node được chọn là trạng thái đích then return solution

else **mở rộng node và thêm các node kết quả** vào cây tìm kiếm

End

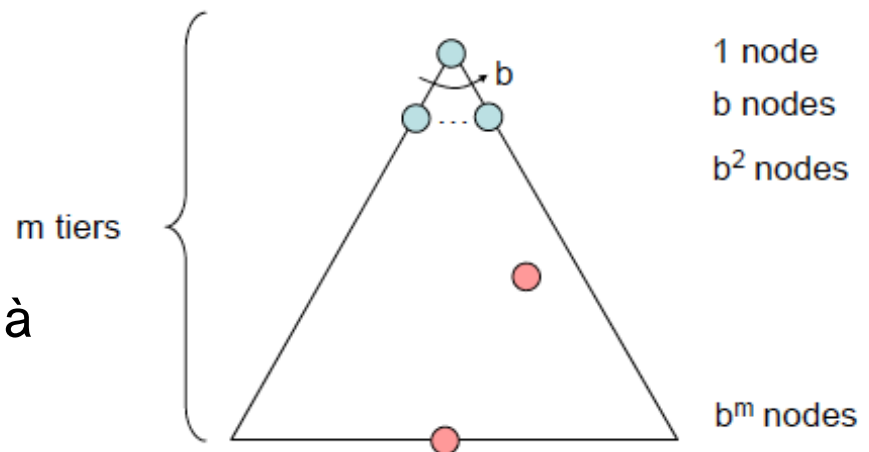


Fringe?
Mở rộng node?
Chiến lược duyệt?

Các thuộc tính của thuật toán tìm kiếm

- Đầy đủ (complete): đảm bảo tìm thấy lời giải nếu thực sự tồn tại lời giải?
- Tối ưu (optimal): Đảm bảo tìm thấy đường đi có chi phí thấp nhất?
- Độ phức tạp tính toán (time complexity)
- Độ phức tạp không gian lưu trữ (space complexity)
- Giả sử:
 - Số nhánh rẽ ở mỗi node là b
 - Độ sâu tối đa của cây là m
 - Các lời giải nằm ở các độ sâu khác nhau
- Khi đó, số lượng node trên cả cây là

$$1 + b^2 + \dots + b^m = O(b^m)$$



Nội dung

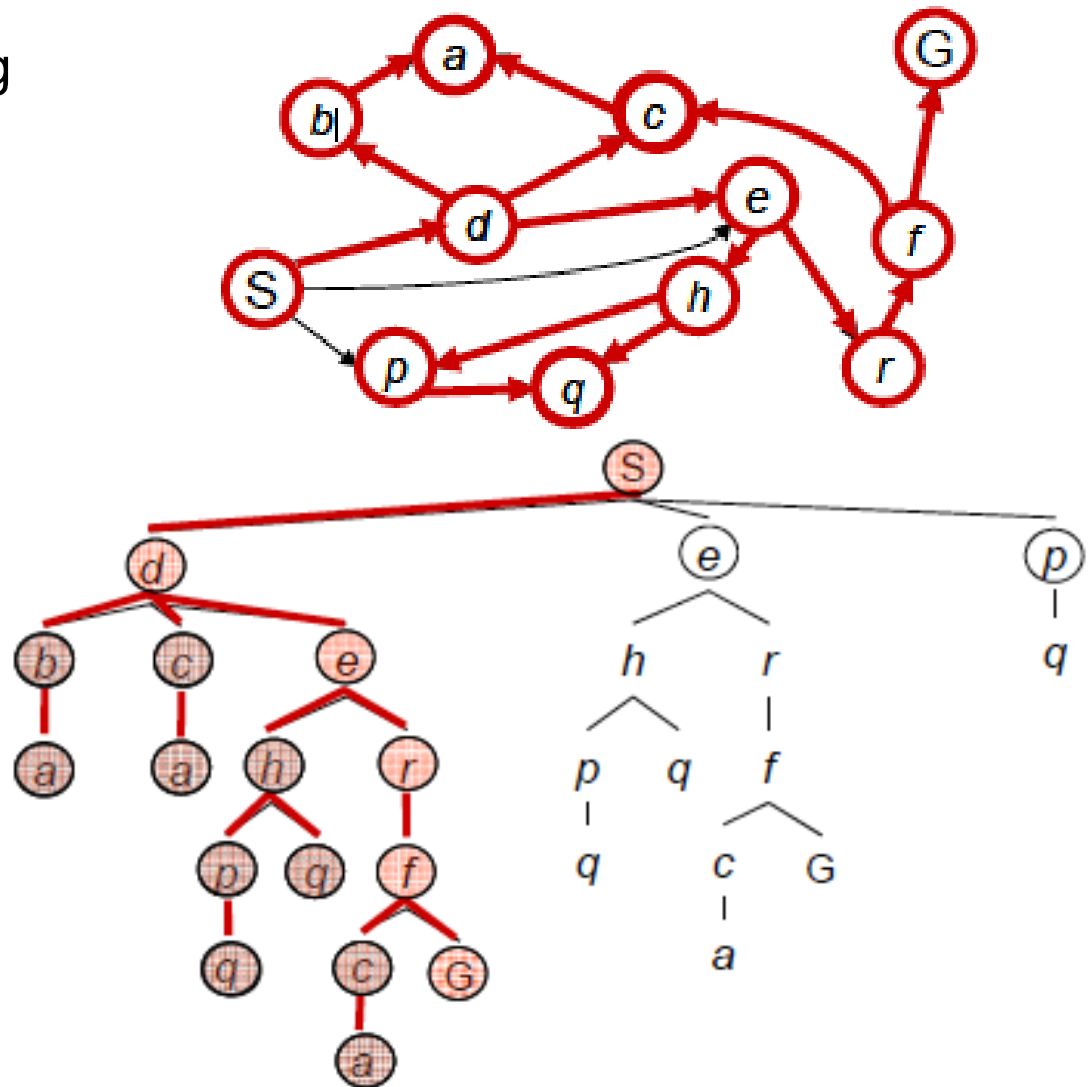
- Agent
- Bài toán tìm kiếm
- **Tìm kiếm mù (uninformed search)**
- Tìm kiếm có hiểu biết (informed search)

Tìm kiếm mù

- Tìm kiếm theo chiều sâu (**D**epth-**F**irst **S**earch)
- Tìm kiếm theo chiều rộng (**B**reath-**F**irst **S**earch)
- Tìm kiếm với lặp sâu dần (**I**terative **D**eepening)
- Tìm kiếm với chi phí đồng nhất (**U**niform **C**ost **S**earch)

Tìm kiếm theo chiều sâu (DFS)

- **Chiến lược duyệt:** mở rộng node sâu nhất trước
- **Fringe:** sử dụng ngăn xếp

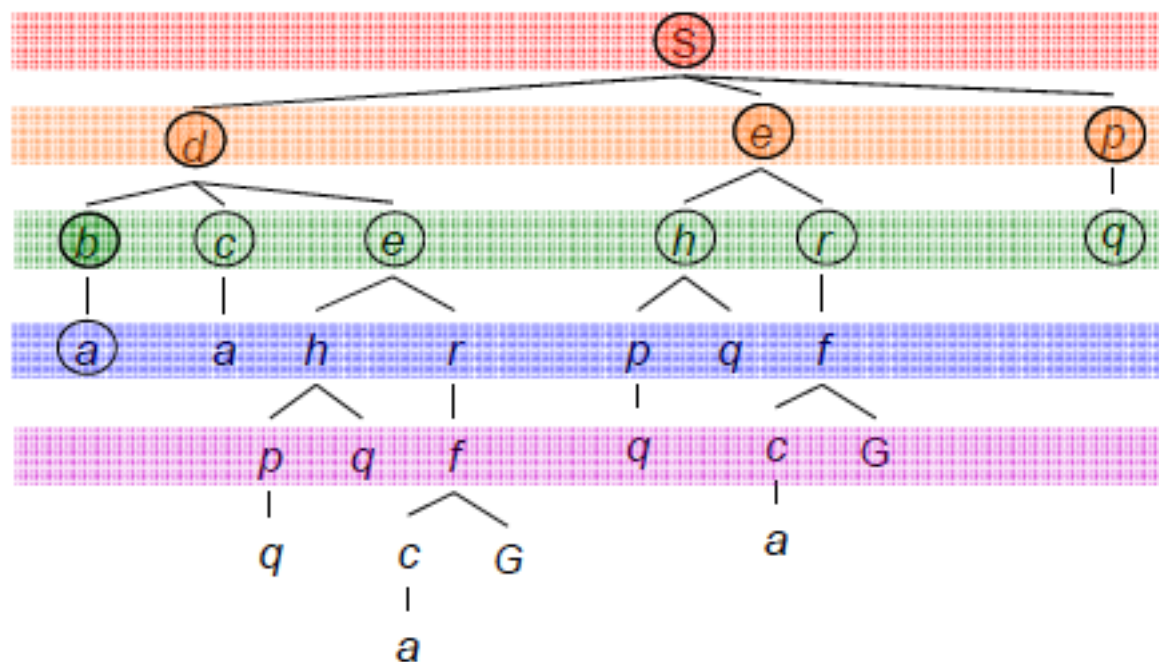
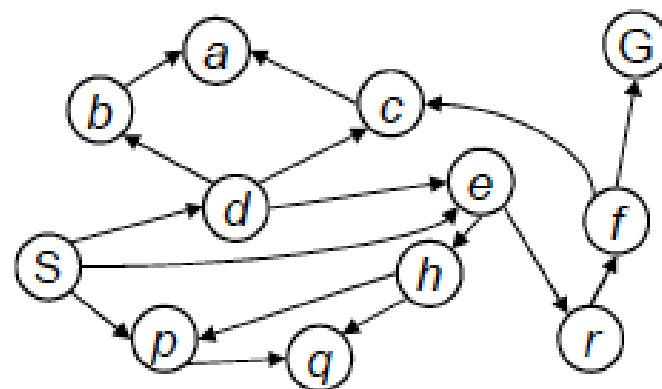


Thuộc tính của DFS

- Cách thức mở rộng các node
 - Từ bên trái nhất của cây trước
 - Có thể phải duyệt toàn bộ cây!
 - Nếu m là xác định, độ phức tạp tính toán xấu nhất là $O(b^m)$
- Độ phức tạp không gian lưu trữ:
 - Chỉ lưu các node anh/em của các node trên đường đi đang xét, $O(bm)$
- Đầy đủ?
 - Giá trị m có thể không xác định, chỉ đầy đủ nếu ngăn được chu trình
- Tối ưu?
 - Không, vì luôn tìm thấy lời giải “trái nhất”, bất chấp độ sâu hay chi phí.

Tìm kiếm theo chiều rộng (BFS)

- **Chiến lược duyệt:** mở rộng node ở mức cạn nhất trước
- **Fringe:** sử dụng hàng đợi



Thuộc tính của BFS

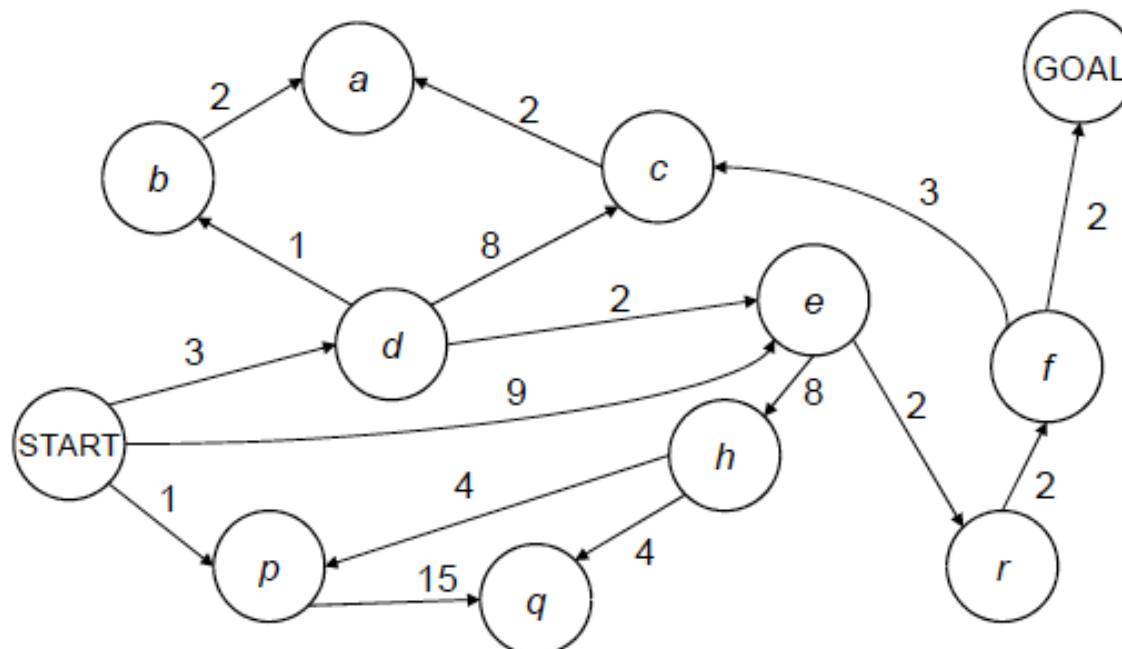
- Cách thức mở rộng các node:
 - Xử lý tất cả các node ở phía trên lời giải “cạn nhất”
 - Gọi độ sâu của lời giải “cạn nhất” là s , thời gian tìm kiếm là $O(b^s)$
- Độ phức tạp không gian lưu trữ
 - Lưu tất cả các node ở tầng cuối cùng, $O(b^s)$
- Đầy đủ?
 - Có, vì s là xác định nếu tồn tại lời giải
- Tối ưu?
 - Chỉ khi tất cả các chi phí đều bằng nhau

Lặp sâu dần (ID hoặc IDS)

- Ý tưởng: kết hợp ưu điểm về không gian lưu trữ của DFS và độ phức tạp tính toán của BFS
- Triển khai:
 - Cài đặt DFS với độ sâu giới hạn
 - Thực thi DFS với độ sâu giới hạn là 1. Nếu không tìm thấy lời giải...
 - Thực thi DFS với độ sâu giới hạn là 2. Nếu không tìm thấy lời giải...
 -

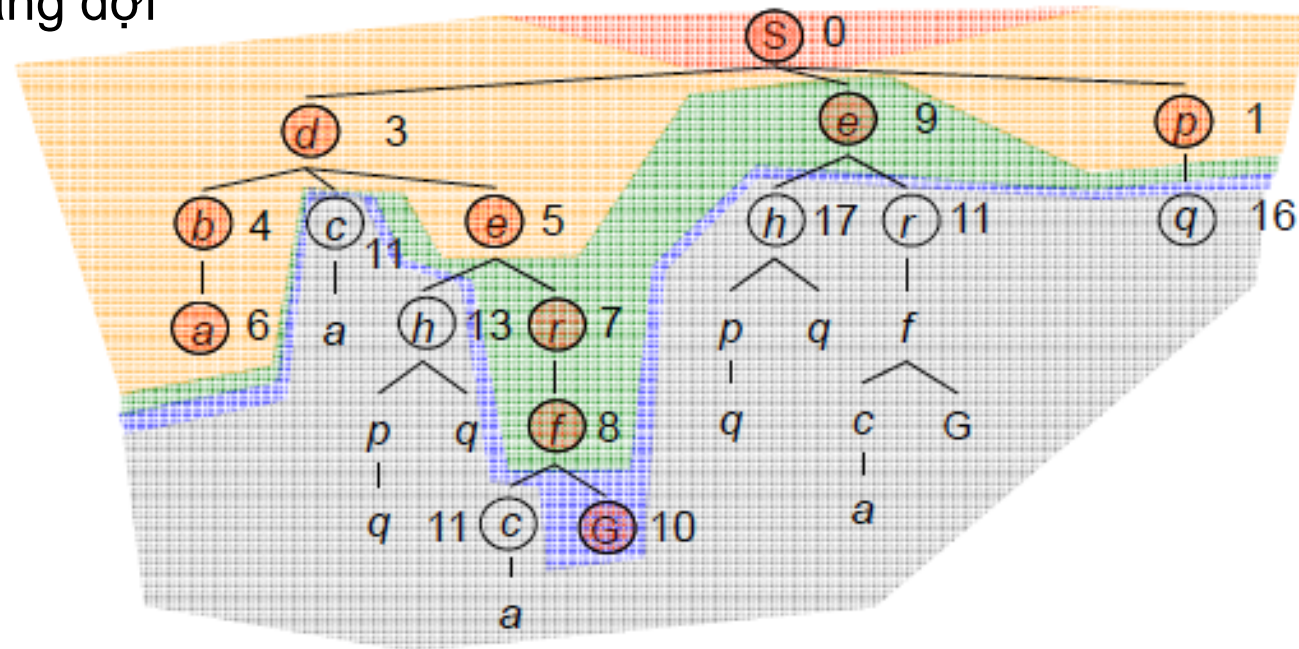
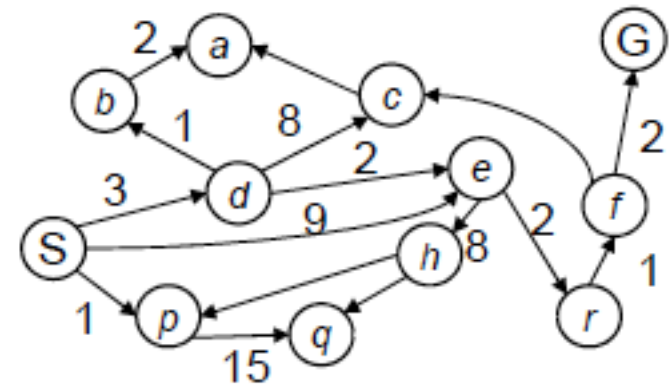
Tìm kiếm với chi phí đồng nhất (UCS)

- BFS tìm đường đi ngắn nhất theo ý nghĩa số bước di chuyển ít nhất
- BFS không tìm đường đi cho chi phí nhỏ nhất



Tìm kiếm chi phí đồng nhất (UCS)

- **Chiến lược duyệt:** mở rộng node có chi phí rẻ nhất trước
 - Là chi phí từ trạng thái bắt đầu đến node đó.
- **Fringe:** sử dụng hàng đợi có độ ưu tiên

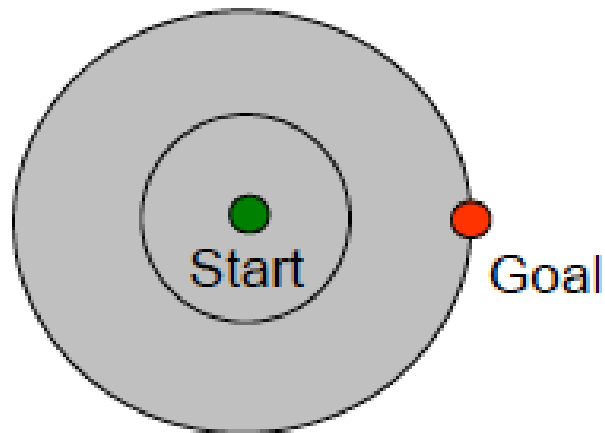


Thuộc tính của UCS

- Cách thức mở rộng các node
 - Xử lý tất cả các node có chi phí nhỏ hơn lời giải “nhỏ nhất”
 - Nếu chi phí của lời giải là C^* và các cạnh có chi phí nhỏ nhất là ϵ thì độ sâu ước chừng là C^*/ϵ
 - Độ phức tạp tính toán là $O(b^{C^*/\epsilon})$
- Độ phức tạp không gian lưu trữ
 - Phải lưu tầng xử lý cuối cùng $O(b^{C^*/\epsilon})$
- Đầy đủ?
 - Có, nếu lời giải “nhỏ nhất” có chi phí xác định và tất cả các cạnh có chi phí không âm
- Tối ưu?
 - Có

Tìm kiếm chi phí đồng nhất (UCS)

- Duyệt các trạng thái theo mọi hướng
- Không sử dụng thông tin về trạng thái đích



Nội dung

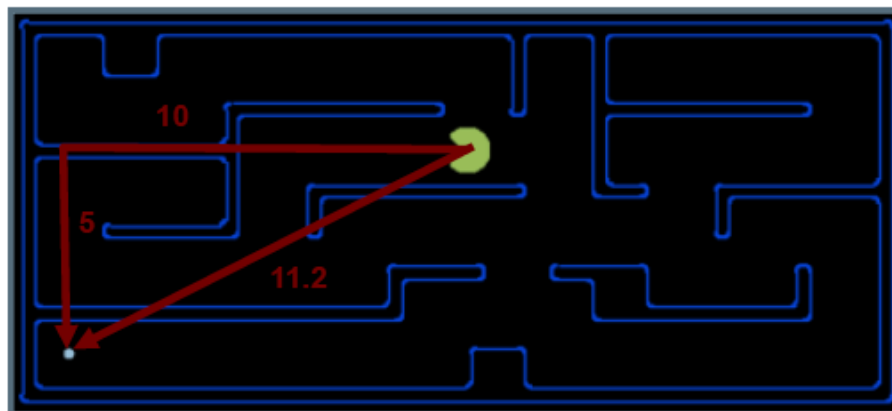
- Agent
- Bài toán tìm kiếm
- Tìm kiếm mù (uninformed search)
- **Tìm kiếm có hiểu biết (informed search)**

Tìm kiếm có hiểu biết (informed search)

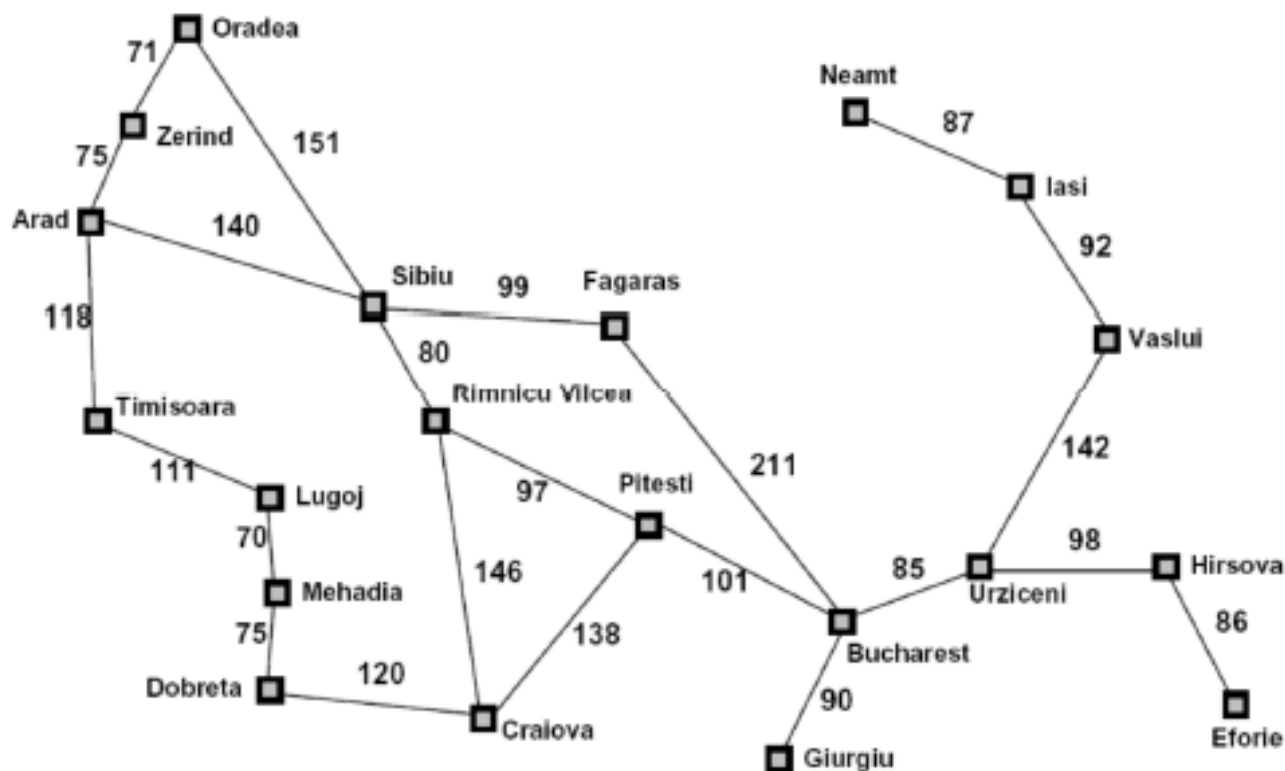
- Heuristic
- Tìm kiếm tham lam (Greedy Search)
- Thuật giải A*

Heuristic tìm kiếm

- Heuristic là một hàm ước lượng mức độ gần của một trạng thái so với trạng thái đích
 - Kí hiệu: $h(s)$, với s là trạng thái.
- Heuristic được thiết kế cho từng bài toán tìm kiếm cụ thể
- Một số ví dụ: khoảng cách Euclidean hay Manhattan cho bài tìm đường đi



Ví dụ - hàm heuristic



Straight-line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

$h(x)$

Tìm kiếm tham lam (greedy search)

- Cách thức mở rộng các node:
 - Mở rộng node mà agent nghĩ là gần đích đến nhất
 - Heuristic: ước lượng khoảng cách tới đích gần nhất cho mỗi node.
- Trường hợp phổ biến
 - Thường sẽ đưa chúng ta thẳng tới đích, nhưng không phải là chi phí tốt nhất
- Trường hợp xấu nhất:
 - Như thuật toán DFS nhưng được định hướng không tốt.

Thuật giải A^*

- Ý tưởng: kết hợp UCS và tìm kiếm tham lam
- UCS: thứ tự duyệt dựa theo chi phí từ trạng thái bắt đầu đến nó, $g(n)$
- Tìm kiếm tham lam: thứ tự duyệt dựa theo chi phí ước lượng từ nó đến trạng thái đích, $h(n)$
- Thuật giải A^* : thứ tự duyệt dựa theo tổng $f(n) = g(n) + h(n)$

Thuật giải A*

1. Khởi tạo tập **OPEN** = {start}, **CLOSED** = {}, $g(\text{start}) = 0$, $f(\text{start}) = h(\text{start})$
2. If **OPEN** == {} then return failure
3. Chọn trạng thái có chi phí nhỏ nhất từ **OPEN**, n. Đưa n vào **CLOSED**.
4. If n là trạng thái đích then return solution
5. Mở rộng node n. Với mỗi m là node con của n, ta có:

If m không thuộc **OPEN** hay **CLOSED** then:

Gán $g(m) = g(n) + C(n, m)$

Gán $f(m) = g(m) + h(m)$

Thêm m vào **OPEN**

If m thuộc **OPEN** hoặc **CLOSED** then:

Gán $g(m) = \min \{g(m), g(n) + \text{cost}(n, m)\}$

Gán $f(m) = g(m) + h(m)$

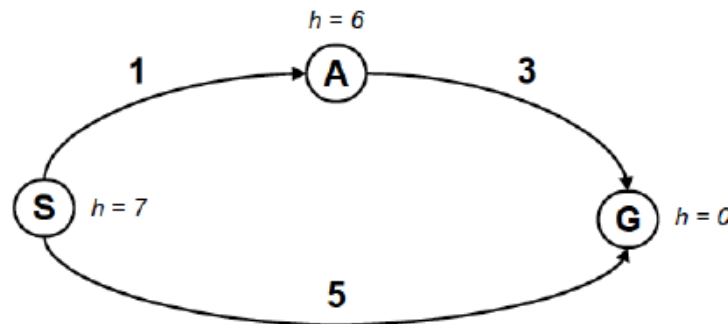
If $f(m)$ giảm và m thuộc **CLOSED** then:

Đưa m trở lại **OPEN**

End

Quay lại bước 2.

A* có tối ưu?



- Chi phí ước lượng (cho trường hợp tốt) cao hơn chi phí thực (cho trường hợp xấu)
- Chúng ta cần các ước lượng nhỏ hơn chi phí thực tế

Heuristic có thể chấp nhận

- Một heuristic là có thể chấp nhận (admissible) nếu:

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

- Trong đó, $h^*(n)$ là chi phí thực tế đến đích gần nhất
- Định lý: Nếu $h(n)$ là admissible, thuật toán A^* với cây tìm kiếm là tối ưu.

Tạo các heuristic có thể chấp nhận

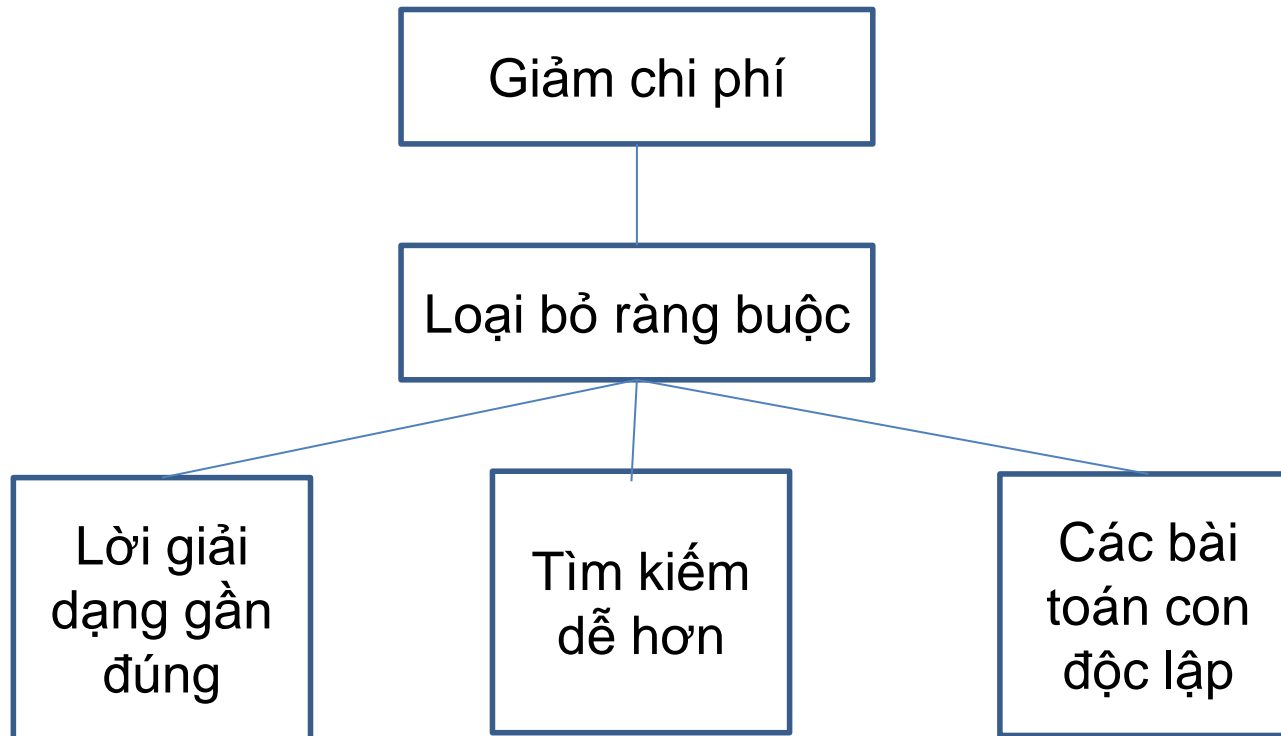
- Thông thường, **admissible heuristic** là các lời giải cho các bài toán được nới lỏng ràng buộc (relaxation), ở đó có nhiều hành động mới được cho phép
- Đôi khi các inadmissible heuristic cũng có ích

Sự nới lỏng (relaxation)

- Làm sao để có một heuristic tốt?
 - Lý tưởng thì $h(s) = h^*(s)$. Điều này khó như giải quyết bài toán gốc
 - Thay vì dùng $h^*(s)$ của bài toán gốc, chúng ta dùng $h^*(s)$ của bài toán dễ hơn
 - Heuristic tốt liên quan đến mô hình hóa, không phải xây dựng thuật toán



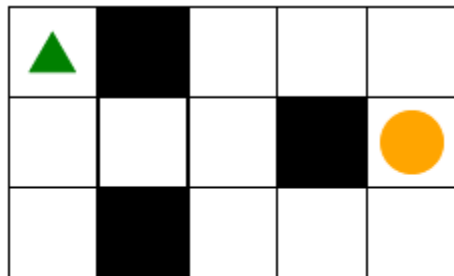
Sự nói lỏng



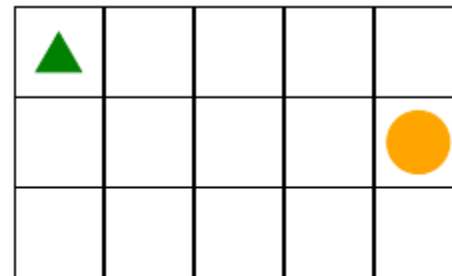
- Kết hợp các heuristic bằng hàm max

Lời giải dạng gần đúng

- Mục tiêu: di chuyển từ ô hình tam giác đến hình tròn
- Nói lỏng: bỏ các bức tường đi



Hard



Easy

- Bài toán sau khi nói lỏng: h^* chính là khoảng cách Mahattan
 - → Sử dụng $h^*(n)$ này để làm heuristic

Tìm kiếm dễ hơn

- Bài toán gốc
 - Trạng thái bắt đầu: 1
 - Hành động walk: từ s đến $s+1$ (chi phí là 1)
 - Hành động tram: từ s đến $2s$ (chi phí là 2)
 - Trạng thái đích: n
 - **Ràng buộc: số hành động tram không được nhiều hơn walk**
- Trạng thái: **(vị trí, #walk - #tram)**
 - Số trạng thái từ $O(n)$ thành $O(n^2)$

Tìm kiếm dễ hơn

- Giữ các thông tin bài toán gốc, nhưng loại bỏ ràng buộc
- Trạng thái sau nói lỏng: (location)
- Bài toán sau khi nói lỏng: tìm đường đi từ thành phố 1 đến thành phố n .
 - Hàm heuristic là h^* của bài toán nói lỏng. Có 2 giải pháp:
 - GP1: Hàm $h^*(\text{location})$ của quy hoạch động
 - GP2: $g(\text{location})$ của UCS nhưng tìm đường từ trạng thái đích tới trạng thái bắt đầu.

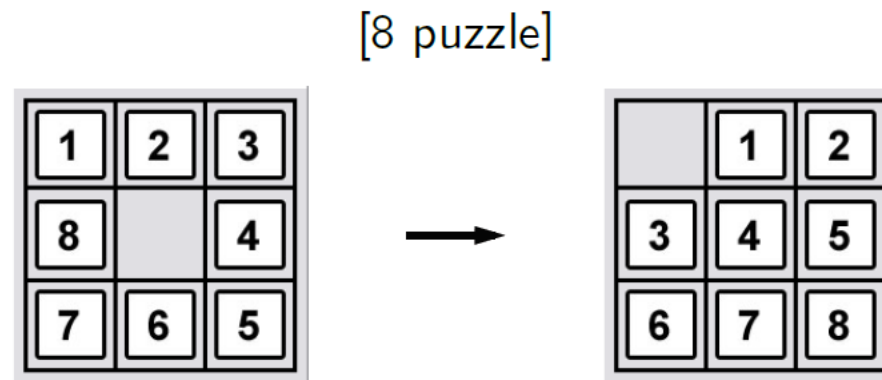
Start state: n

Walk action: from s to $s - 1$ (cost: 1)

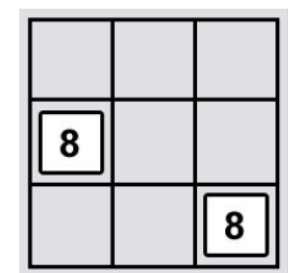
Tram action: from s to $s/2$ (cost: 2)

Goal state: 1

Bài toán con độc lập



- Bài toán gốc: các mảnh ghép không được chồng lên nhau
- Nói lỏng: Các mảnh ghép có thể chồng lên nhau.
- Bài toán sau khi nói lỏng: 8 bài toán con độc lập nhau.
 - Mỗi bài toán con tương ứng với một dạng gần đúng (slide trước).
 - Hàm heuristic là hàm tổng của các bài toán con.



Nhắc lại

- Các bài toán nói lỏng được sử dụng để lấy giá trị cho hàm heuristic $h(s)$ nhằm định hướng tìm kiếm. Các lời giải của bài toán nói lỏng không phải là lời giải của bài toán gốc.

Tài liệu tham khảo

- *Artificial Intelligence: A Modern Approach*, 3rd Edition, S. Russel and P. Norvig, Pearson Education Inc., 2010
- *Techniques in Artificial Intelligence* (SMA 5504) , MIT OpenCourseWare, Massachusetts Institute of Technology
- *Artificial Intelligence: Principles and Techniques*, Stanford courses, Autumn 2015.