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

Cơ Sở Trí Tuệ Nhân Tạo

Tác Nhân Tìm Kiếm

Trình bày: Lê Ngọc Thành
Bộ Môn Khoa Học Máy Tính

Thành Phố Hồ Chí Minh

Nội dung

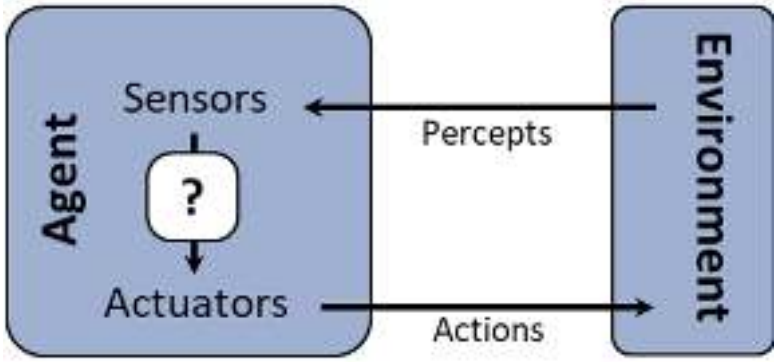
- **Tác nhân lên kế hoạch (planning agent)**
- Bài toán tìm kiếm (bài toán lên kế hoạch)
- Giải quyết bài toán bằng tìm kiếm mù
  - Thuật toán Depth-First Search
  - Thuật toán Breadth-First Search
  - Thuật toán Uniform Cost Search
  - Một số thuật toán khác
- Đánh giá các thuật toán

Tác nhân thông minh (hợp lý)

- Tác nhân (agent) là một thực thể có khả năng nhận thức và sau đó là hành động.

$$f: P^* \rightarrow A$$

- Tác nhân hợp lý (rational agent) sẽ hành động để đạt được kết quả tốt nhất.
  - Sau khi nhận thức có thể suy nghĩ/suy diễn.
  - Sau khi nhận thức có thể phản xạ mà không cần suy nghĩ.



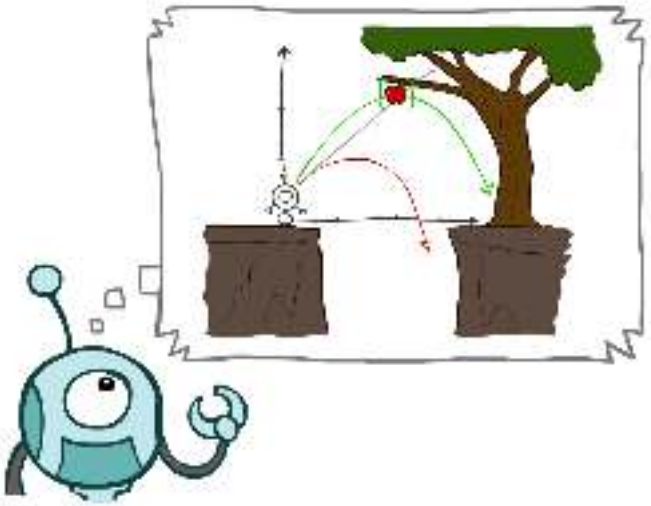
Tác nhân hợp lý

- Kết quả tốt nhất?
  - Dựa vào định nghĩa/tiêu chí
  - Trong môi trường hiện có
  - Thời gian ra quyết định
  - Tri thức đang có hiện tại

Idiot attempts to cross street

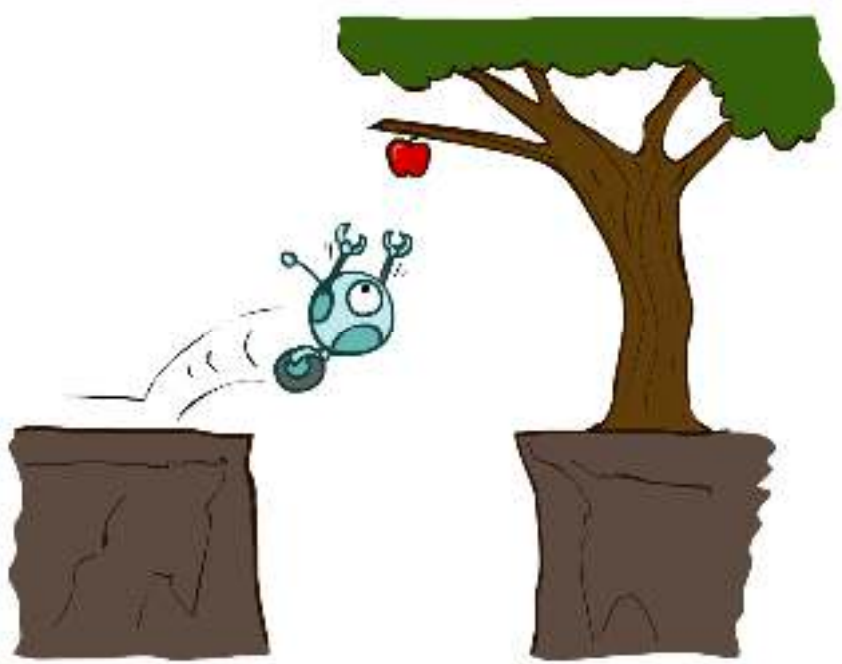
Lên kế hoạch

- Lên kế hoạch để giải quyết bài toán
  - Dựa trên hiểu biết về môi trường đang có
  - "Suy nghĩ" để đưa ra *kế hoạch* hành động nhằm đạt được *kết quả tốt nhất* (giải được bài toán).
  - Thực hiện theo kế hoạch.



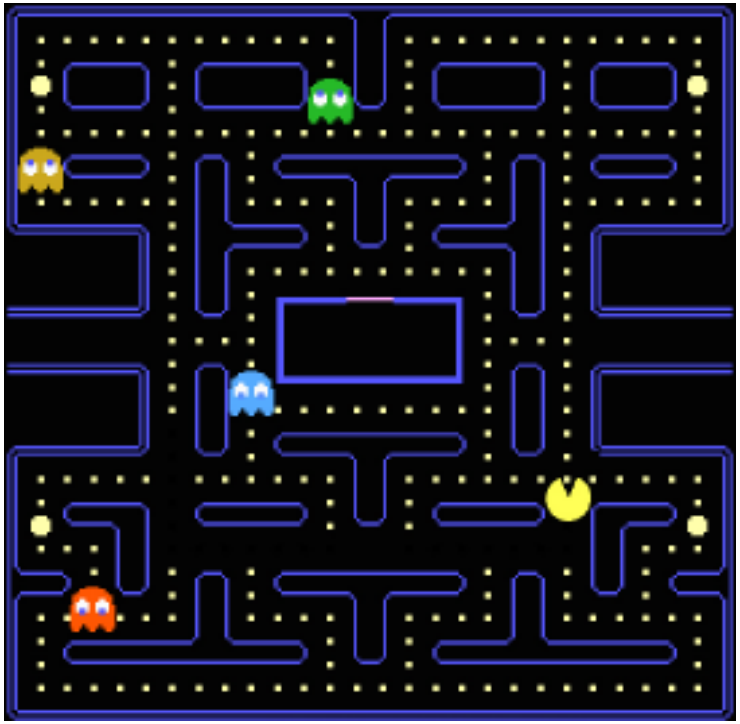
Thực hiện theo kế hoạch

- Hành động = Thực hiện theo kế hoạch



Thế giới Pac-man

- Lên kế hoạch?



Thế giới Pac-man

- Pacman muốn ăn hết thức ăn với số bước đi ít nhất.
- Đầu tiên, Pacman lên kế hoạch trong đầu: xem xét các kế hoạch khác nhau và chọn ra kế hoạch tốt nhất
  - Trong đầu Pacman cần có mô hình về "thế giới": "thế giới" sẽ thay đổi như thế nào khi Pacman thực hiện các hành động
- Sau đó, Pacman thực thi kế hoạch này.

Tác nhân giải quyết bài toán

- Tác nhân giải quyết bài toán (problem-solving agent):
  - Khi cần phải thực hiện hành động hợp lý mà không phải ngay lập tức, tác nhân sẽ lập kế hoạch trong đầu (một tuần tự các bước để đến trạng thái kết thúc).
  - Tiến trình tính toán thực hiện bên dưới được gọi là tìm kiếm (searching)

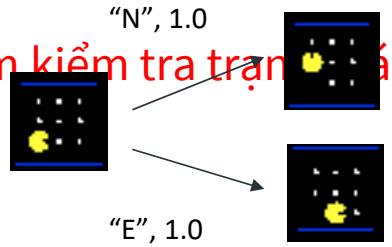
Nội dung

- Hệ thống lên kế hoạch (planning agent)
- **Bài toán tìm kiếm (bài toán lên kế hoạch)**
- Giải quyết bài toán bằng tìm kiếm mù
  - Thuật toán Depth-First Search
  - Thuật toán Breadth-First Search
  - Thuật toán Uniform Cost Search
  - Một số thuật toán khác
- Đánh giá các thuật toán

Bài toán tìm kiếm

- Một bài toán tìm kiếm bao gồm:
  - Không gian trạng thái

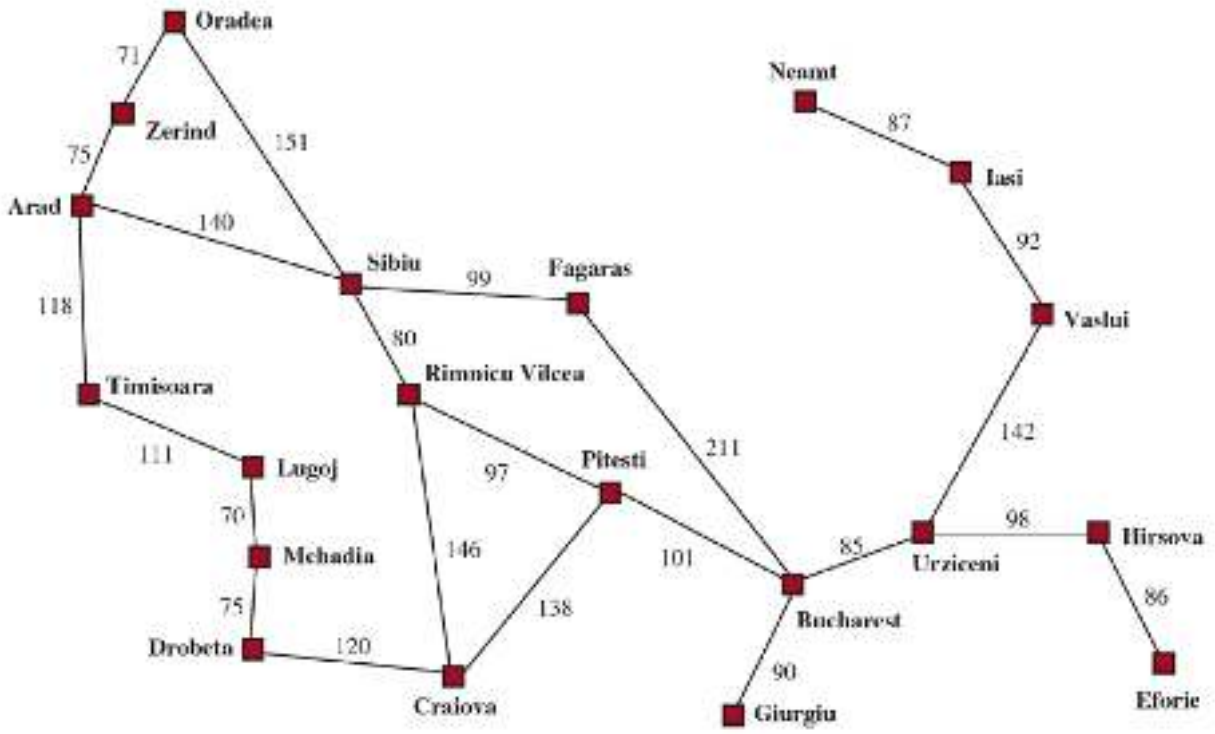
  - Hàm "successor"
    - Cho biết từ một trạng thái sẽ có thể đi đến những trạng thái kế nào (với hành động nào và chi phí bao nhiêu)



  - Trạng thái bắt đầu, và hàm kiểm tra trạng thái đích

Bài toán tìm kiếm

- Xác định các thông tin trong bài toán tìm đường đi từ Arad đến Bucharest:

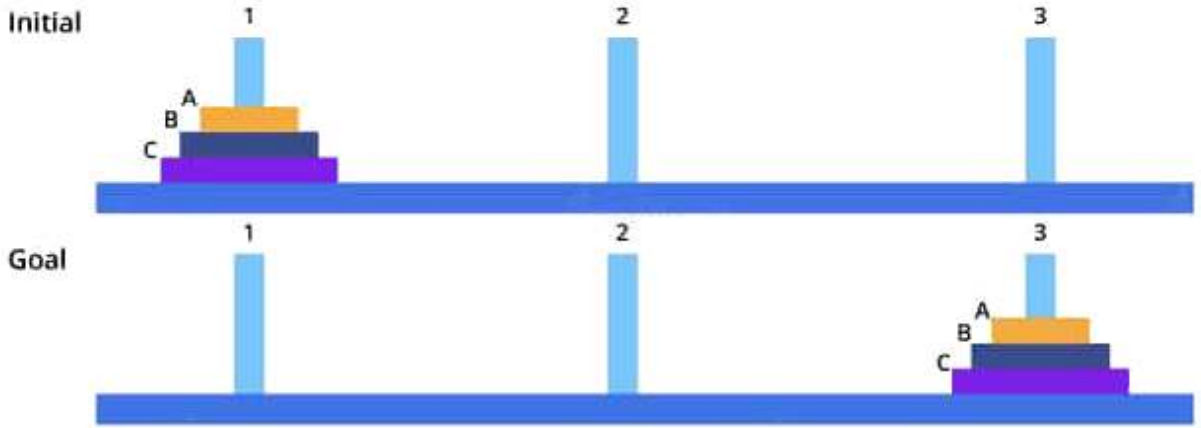


Mô hình

- Bài toán tìm kiếm = mô hình hóa về thế giới và giải quyết nó thông qua mô hình.

Bài toán tìm kiếm

- Một **lời giải** là một kế hoạch gồm một chuỗi các hành động mà sẽ chuyển từ trạng thái bắt đầu sang trạng thái đích.



Bài toán Tháp Hà Nội

Không gian trạng thái

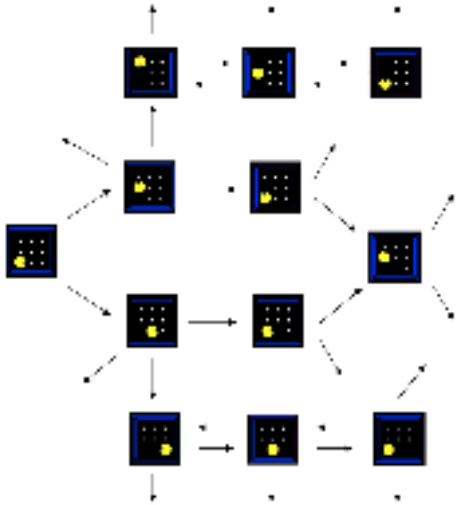
Trạng thái thế giới thực bao gồm mọi chi tiết của môi trường (vd, vị trí của Pacman, hướng quay mặt của Pacman, màu sắc Pacman, …)

Trạng thái trong bài toán tìm kiếm chỉ gồm các thông tin cần thiết cho việc lên kế hoạch

- Bài toán: Đường đi
  - Trạng thái: vị trí (x,y)
  - Hành động: NSEW
  - Successor: cập nhật 1 vị trí
  - Mục tiêu: (x,y)=END
- Bài toán: Ăn thức ăn
  - Trạng thái: {(x,y), vị trí thức ăn}
  - Hành động: NSEW
  - Successor: cập nhật vị trí và trạng thái thức ăn
  - Mục tiêu: thức ăn ăn hết

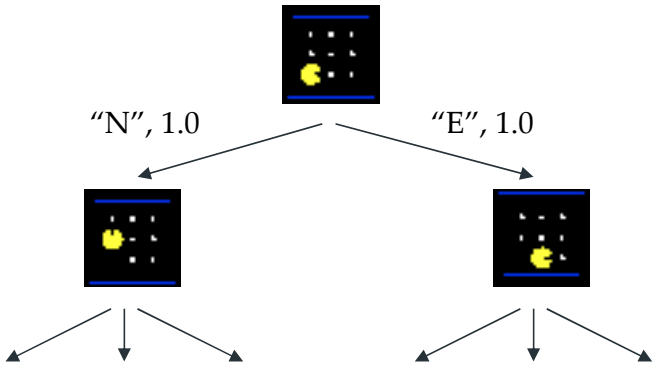
Đồ thị (không gian) trạng thái

- Là một cách biểu diễn của bài toán tìm kiếm
  - Mỗi đỉnh ứng với một trạng thái
  - Các cạnh đi ra từ một đỉnh ứng với hàm successor tại đỉnh đó
- Trong đồ thị trạng thái, mỗi trạng thái chỉ xuất hiện một lần
- Ta sẽ không xây dựng cả đồ thị này vì thường sẽ rất lớn, nhưng đồ thị này sẽ giúp ích cho việc hình dung



Cây tìm kiếm

- Cây tìm kiếm giúp ta dễ xác định các trạng thái có thể tại mỗi thời điểm.

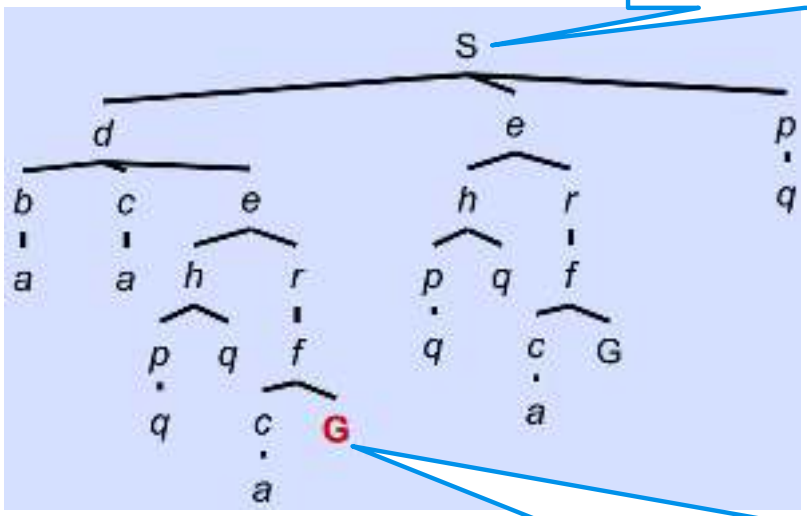


Bắt đầu

Trạng thái tương lai

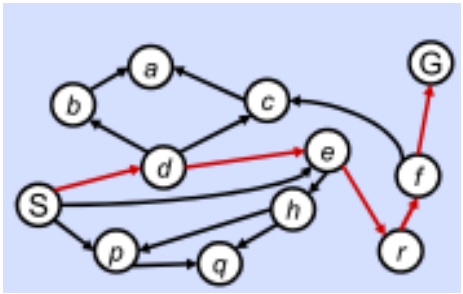
Cây tìm kiếm và kế hoạch

Đỉnh gốc là trạng thái bắt đầu (hiện tại)



Các tương lai có thể xảy ra

Đồ thị trạng thái tương ứng

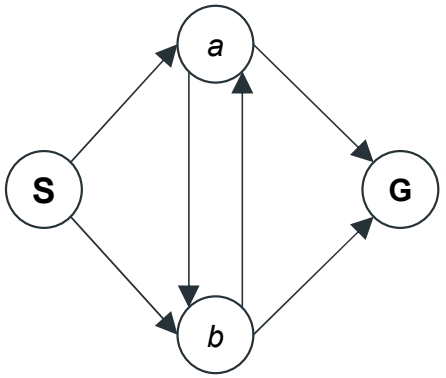


Một đỉnh của cây không chỉ là một trạng thái mà là **một kế hoạch**: một đường đi từ S đến trạng thái này

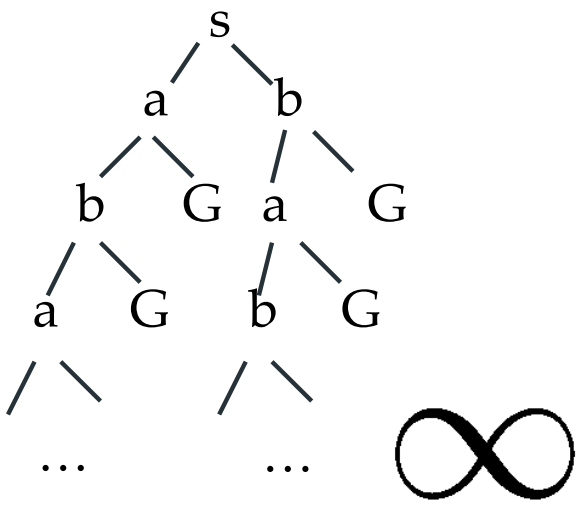
Do đó, trong cây, một trạng thái có thể xuất hiện nhiều lần

Cây tìm kiếm và đồ thị trạng thái

- Cho đồ thị trạng thái:



- Cây tìm kiếm từ S?



Nội dung

- Hệ thống lên kế hoạch (planning agent)
- Bài toán tìm kiếm (bài toán lên kế hoạch)
- **Giải quyết bài toán bằng tìm kiếm mù**
  - Thuật toán Depth-First Search
  - Thuật toán Breadth-First Search
  - Thuật toán Uniform Cost Search
  - Một số thuật toán khác
- Đánh giá các thuật toán

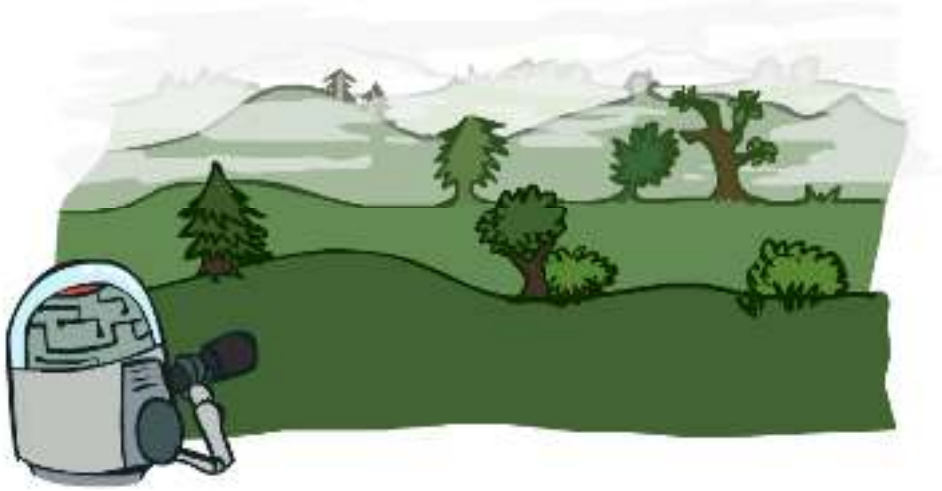
Tìm kiếm mù

- Tìm kiếm mù (blind search/uninformed search):
  - Không có được manh mối trạng thái đích còn bao xa.
  - Ví dụ: người muốn đến Vịnh Hạ Long nếu không biết địa hình sẽ không biết nên chọn thành phố Hà Nội hay Hồ Chi Minh để từ đó đi tiếp.

Các thuật toán tìm kiếm

Ý tưởng:

- Từ trạng thái bắt đầu, tiến hành phát triển dần cây tìm kiếm (*lập kế hoạch*) theo một chiến lược nào đó.
- Làm cho đến khi tìm được trạng thái đích.



Các thuật toán tìm kiếm

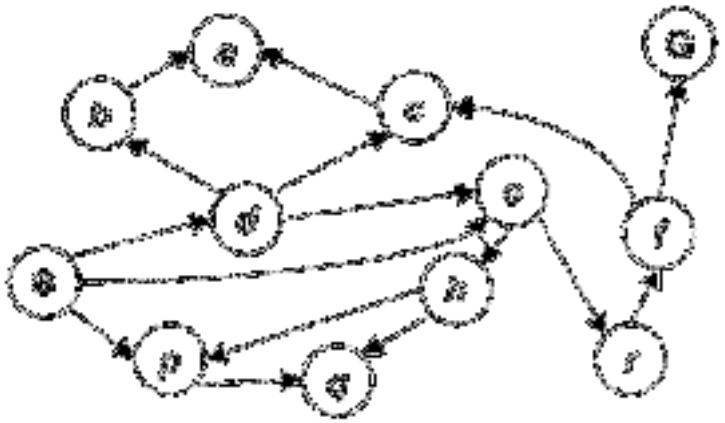
Cụ thể:

- Trong quá trình tìm kiếm, ta sẽ có biến fringe lưu danh sách các kế hoạch tiềm năng (ứng với các node lá của cây đang xây dựng).
- Ở mỗi vòng lặp, ta sẽ lấy ra từ fringe một kế hoạch theo một chiến lược nào đó và mở rộng ra (dựa vào hàm successor), rồi thêm các kế hoạch mới này vào fringe (ứng với việc phát triển cây)
- Cứ thế cho đến khi kế hoạch được lấy ra đến được trạng thái đích, hoặc trong fringe không còn gì nữa (không tìm được lời giải)

Thuật toán Depth-First Search (DFS)

Thuật toán DFS

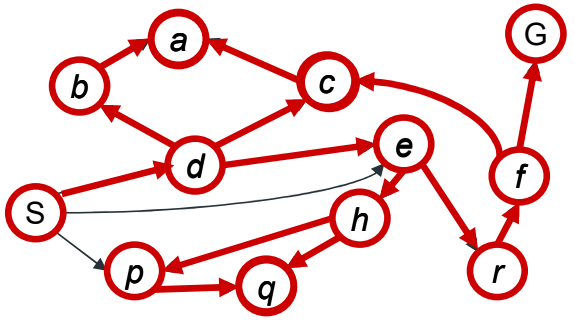
- Chiến lược chọn kế hoạch ở fringe để mở rộng: chọn kế hoạch **"sâu" nhất**
- Có thể dùng stack để cài đặt fringe
- Chạy DFS với đồ thị ở dưới:

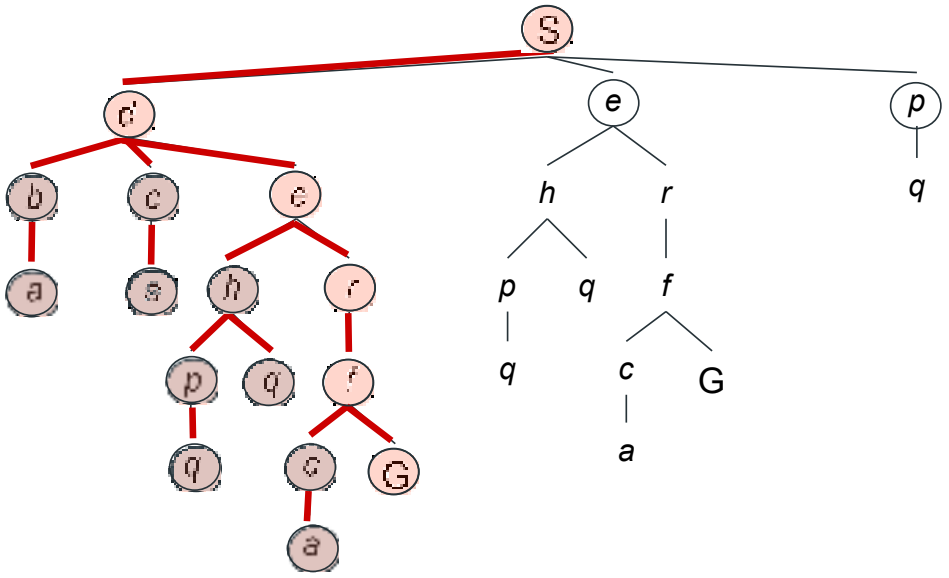


Thuật toán DFS

Chiến lược: Mở rộng trạng thái ở sâu trước

Cài đặt: Fringe thoe cơ chế LIFO của stack



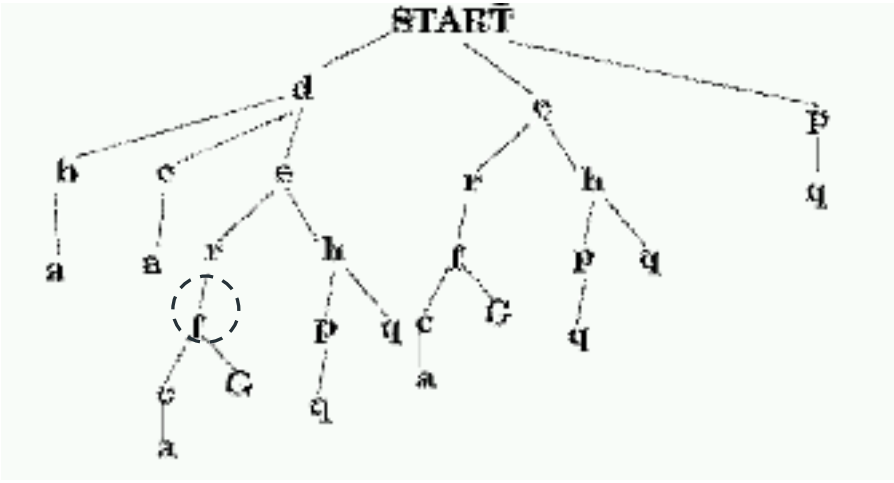


Thuật toán DFS

Ta dùng một cấu trúc dữ liệu gọi là Path để biểu diễn đường đi từ START đến trạng thái hiện tại.

VD. Path P = <START, d, e, r >

Cùng với mỗi node trên đường đi, chúng ta phải nhớ những con nào ta vẫn có thể mở. VD. tại điểm sau, ta có



P = <START (expand=e , p) ,

d (expand = NULL) ,

e (expand = h) ,

r (expand = f) >

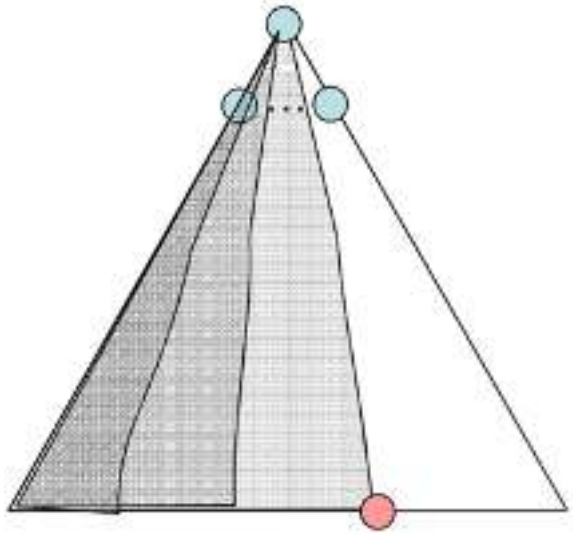
Thuật toán DFS

```
Đặt P = <START (expand = succs(START))>
While (P khác rỗng và top(P) không là đích)
        if mở rộng của top(P) rỗng
        then
                loại bỏ top(P) (“pop ngăn xếp”)
        else
                gọi s một thành viên của mở rộng của top(P)
                loại s khỏi mở rộng của top(P)
                tạo một mục mới trên đỉnh đường đi P:
                        s (expand = succs(s))
If P rỗng
        trả về FAILURE
Else
        trả về đường đi chứa trạng thái của P
```

Thuật toán này có thể được viết gọn dưới dạng đệ qui, dùng ngăn xếp của chương trình để cài đặt P.

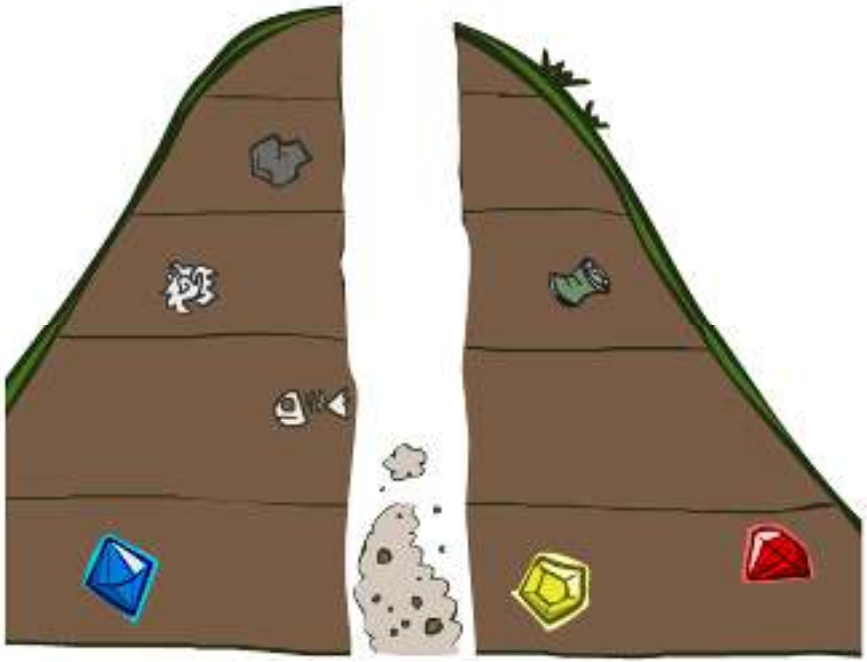
Nhận xét về DFS

- DFS duyệt cây theo hướng từ trái qua phải (hoặc từ phải qua trái tùy theo cách cài đặt)



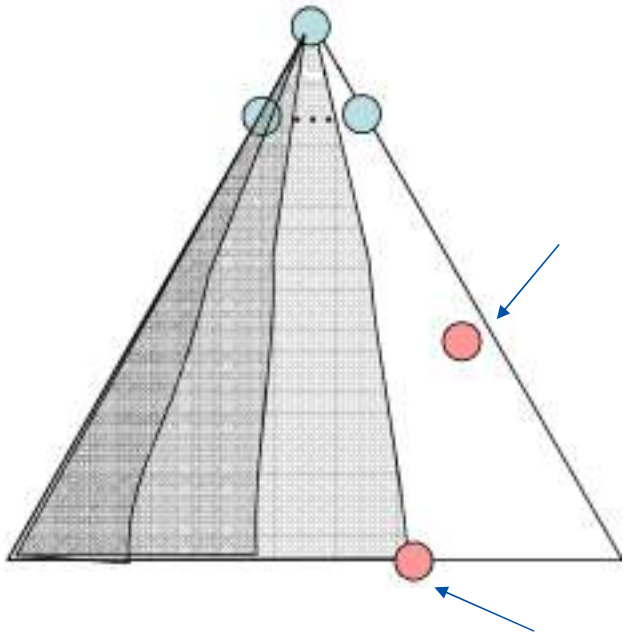
Nhận xét về DFS

- DFS có đảm bảo tìm được lời giải (nếu tồn tại)?
  - Nếu trong đồ thị có vòng lặp thì DFS sẽ có thể bị đào sâu mãi mãi.



Nhận xét về DFS

- DFS có đảm bảo tìm được lời giải tối ưu (nếu tồn tại)?
  - Không, DFS tìm được lời giải ở phía trái nhất của cây (nếu không gặp vấn đề bị đào sâu mãi mãi)

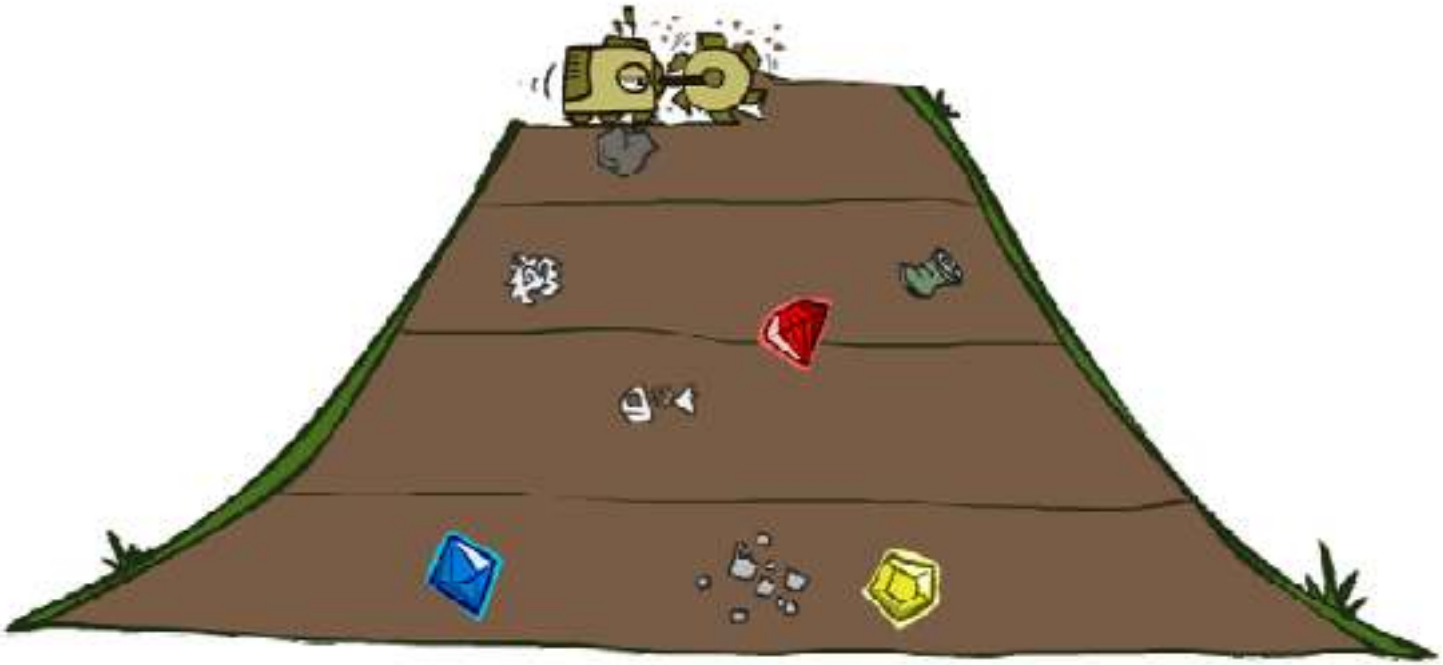


Minh họa DFS

Nội dung

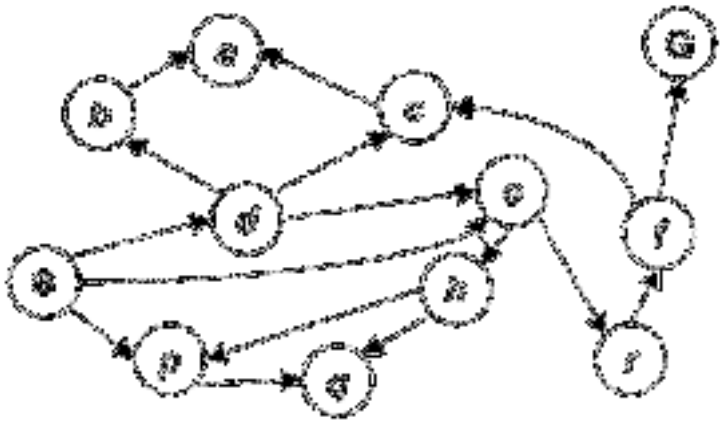
- Hệ thống lên kế hoạch (planning agent)
- Bài toán tìm kiếm (bài toán lên kế hoạch)
- **Giải quyết bài toán bằng tìm kiếm mù**
  - Thuật toán Depth-First Search
  - Thuật toán Breadth-First Search
  - Thuật toán Uniform Cost Search
  - Một số thuật toán khác
- Đánh giá các thuật toán

Thuật toán Breadth-First Search (BFS)

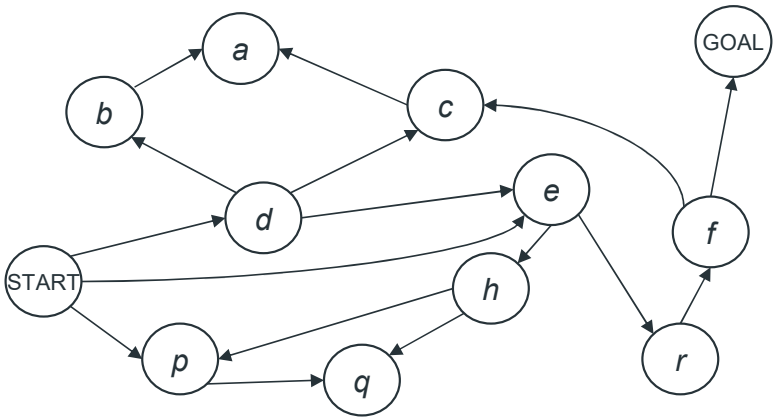


Thuật toán BFS

- Chiến lược chọn kế hoạch ở fringe để mở rộng: chọn kế hoạch **"nông" nhất**
- Có thể dùng queue để cài đặt fringe
- Chạy BFS với đồ thị ở dưới:



Thuật toán BFS



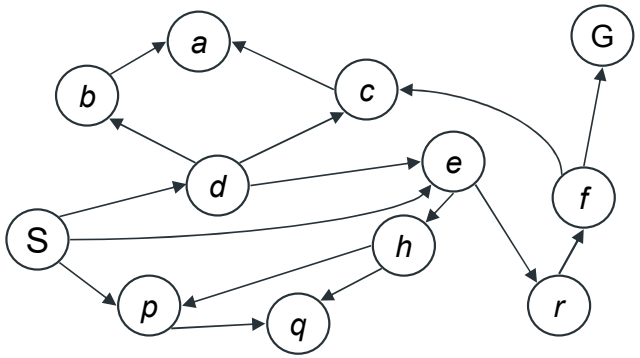
Gán nhãn tất cả trạng thái có thể đi đến được từ S trong 1 bước nhưng không thể đi đến được trong ít hơn 1 bước.

Sau đó gán nhãn tất cả trạng thái có thể đi đến được từ S trong 2 bước nhưng không thể đi đến được trong ít hơn 2 bước.

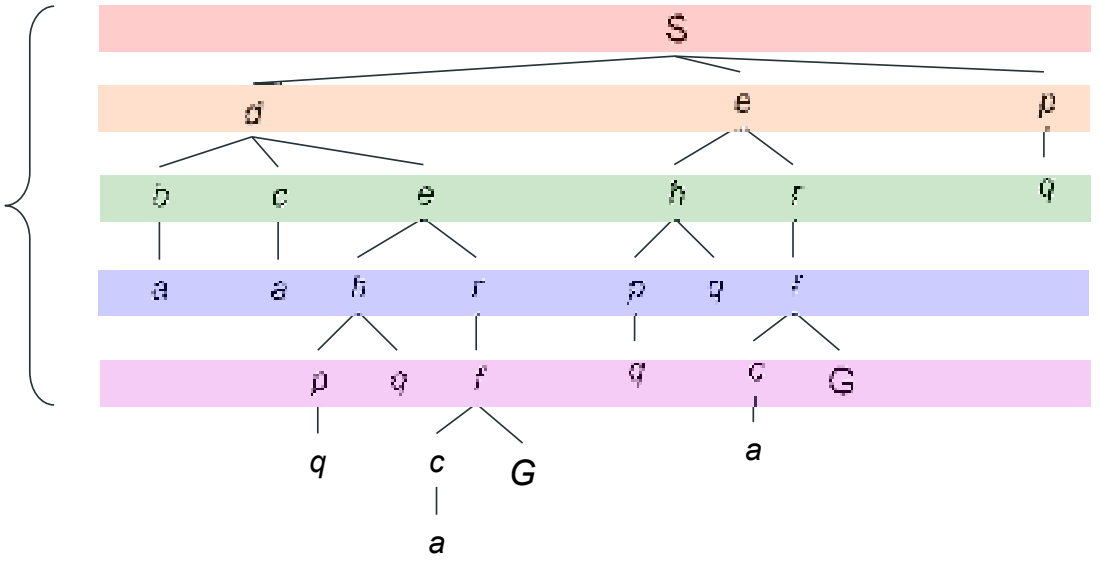
Sau đó gán nhãn tất cả trạng thái có thể đi đến được từ S trong 3 bước nhưng không thể đi đến được trong ít hơn 3 bước.

V.v… đến khi trạng thái Goal được đi đến.

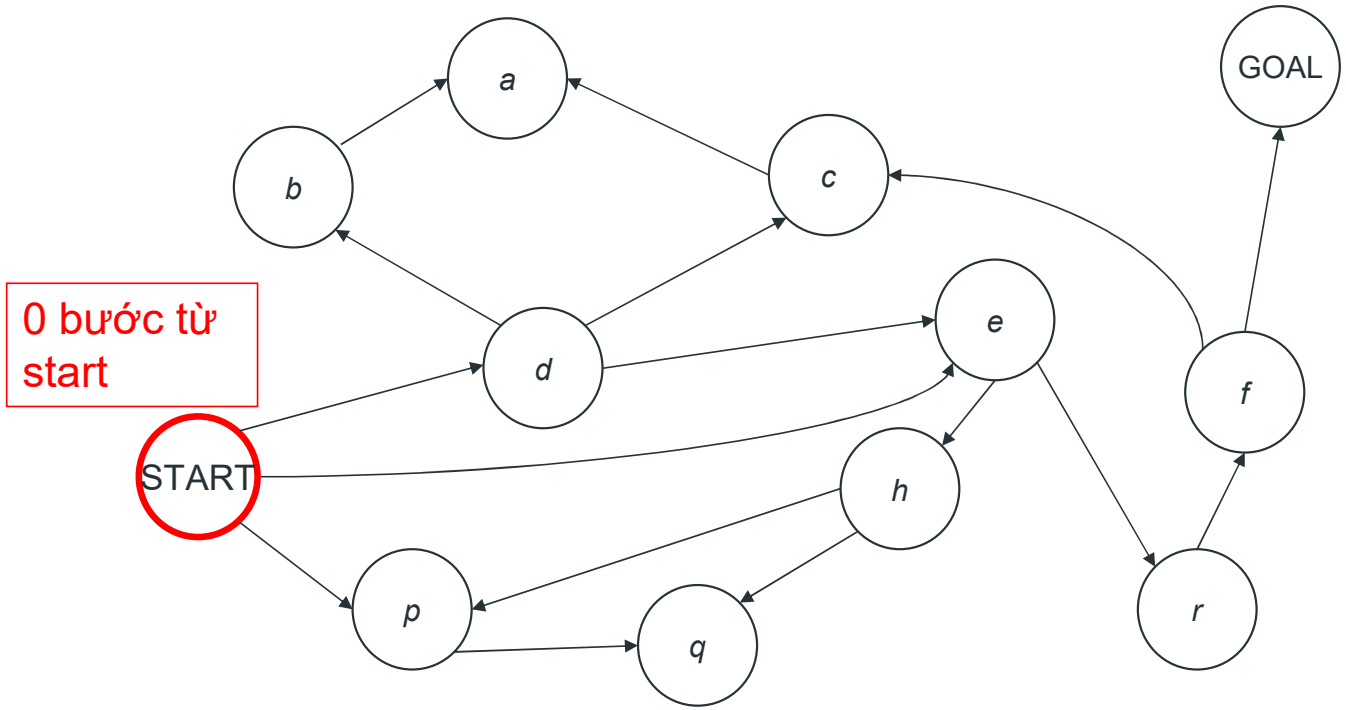
Thuật toán BFS



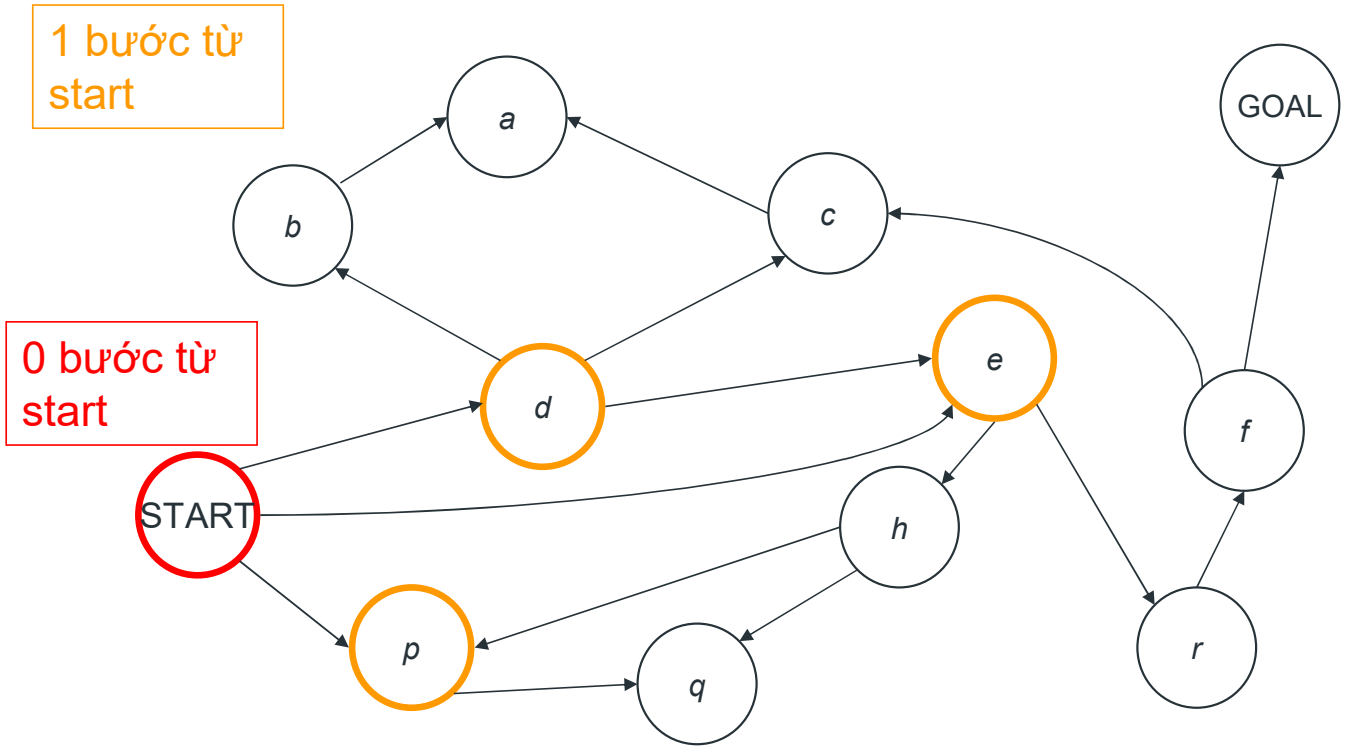
Các tầng tìm kiếm



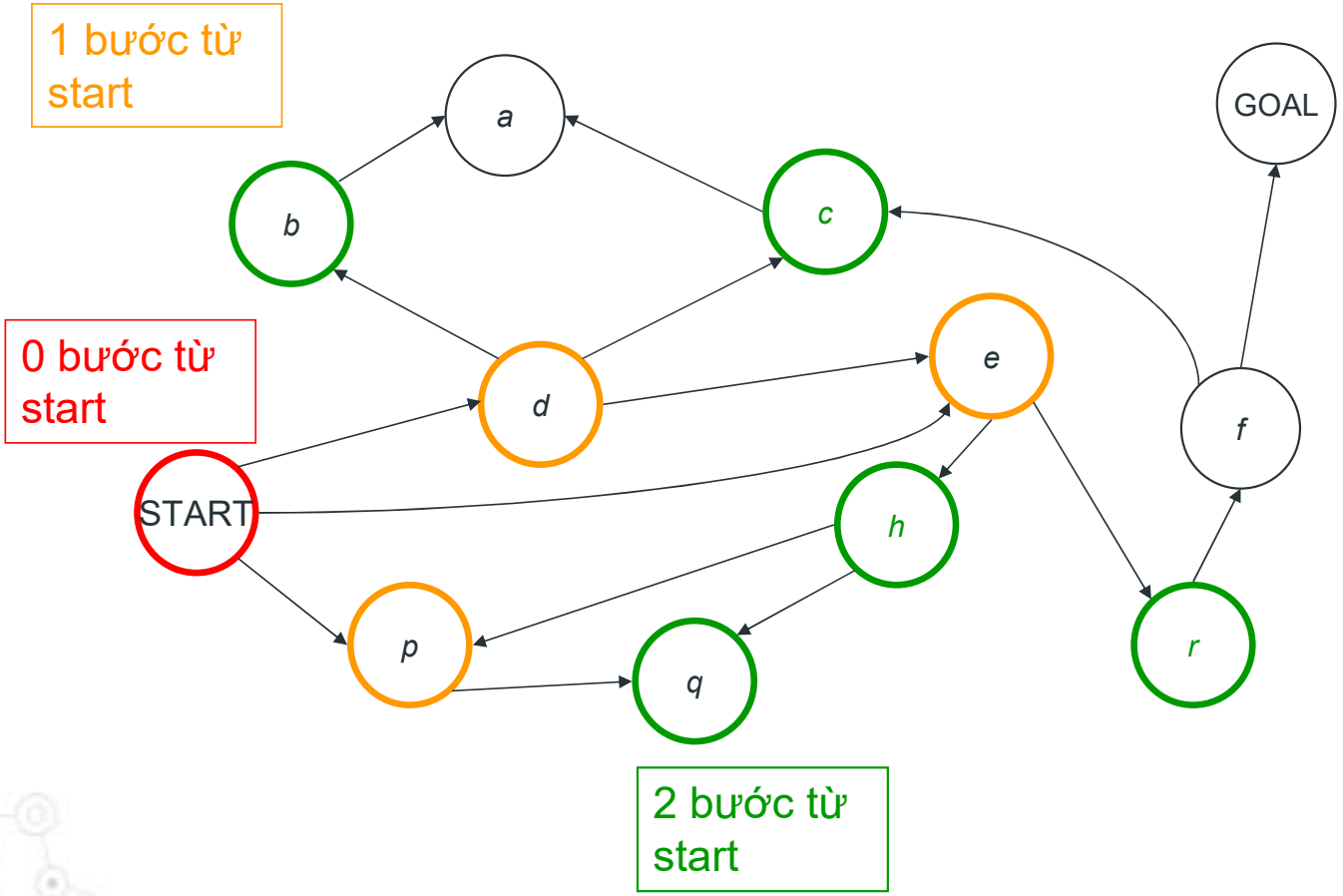
Tìm kiếm Theo Chiều Rộng



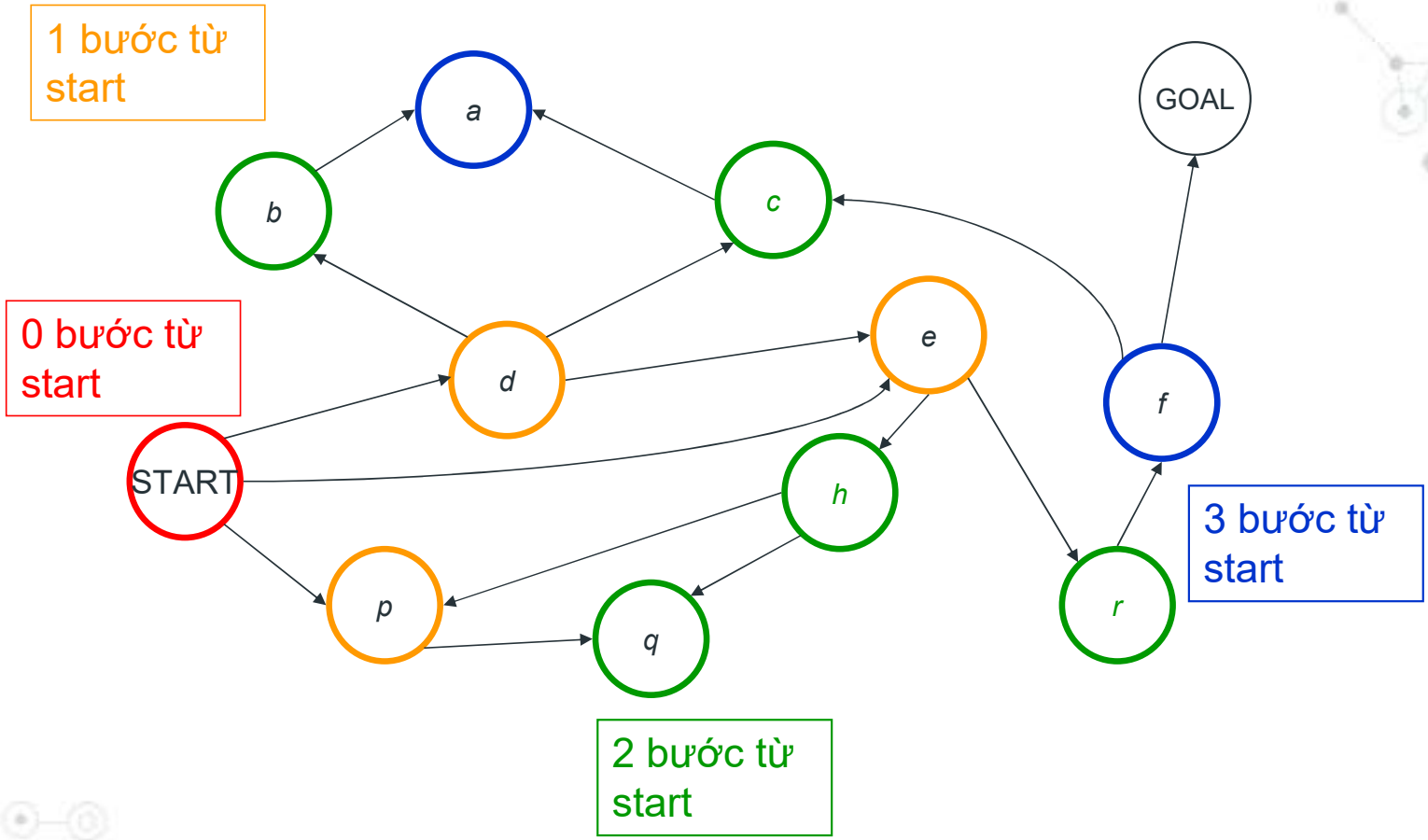
Tìm kiếm Theo Chiều Rộng



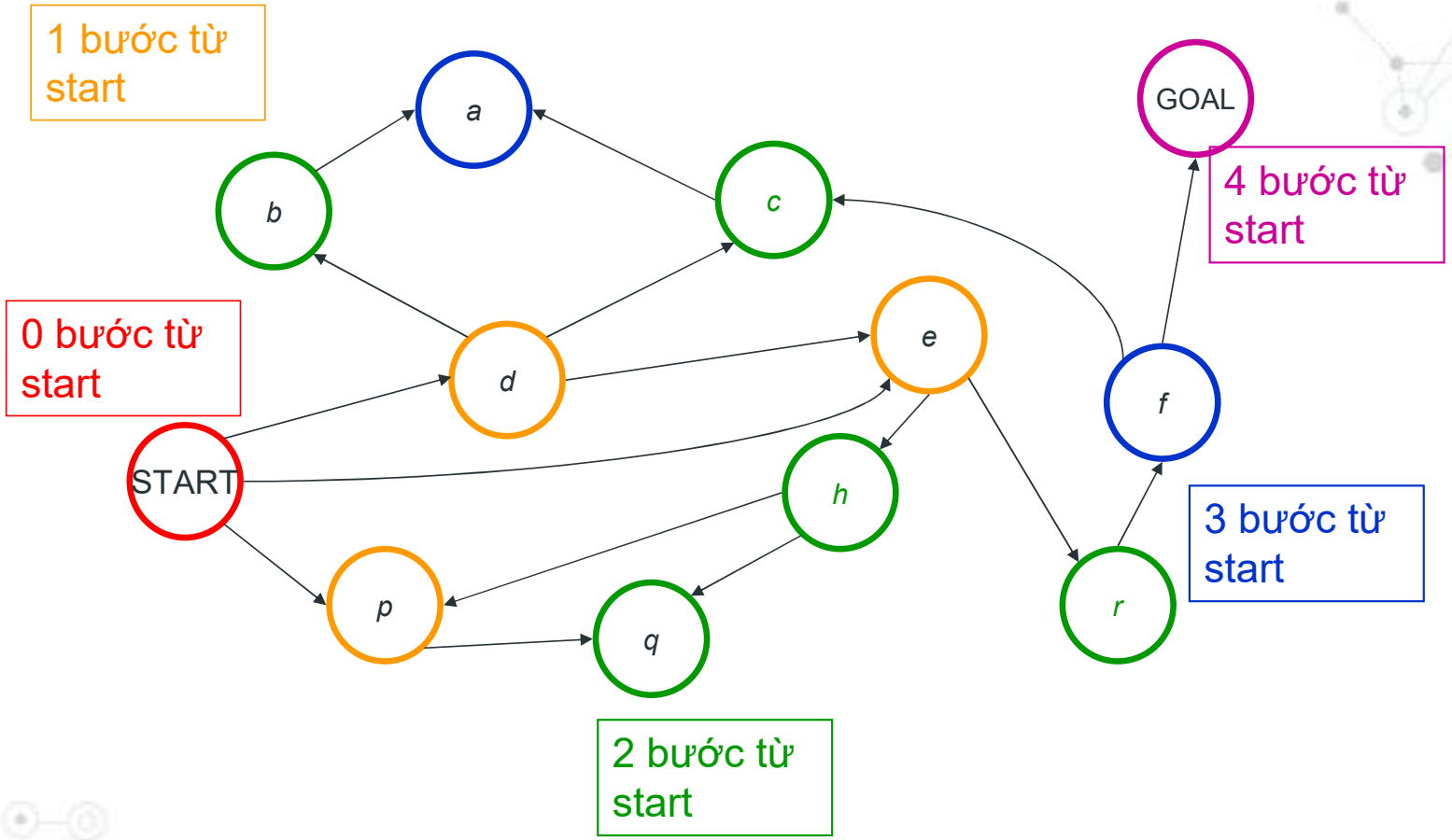
Tìm kiếm Theo Chiều Rộng



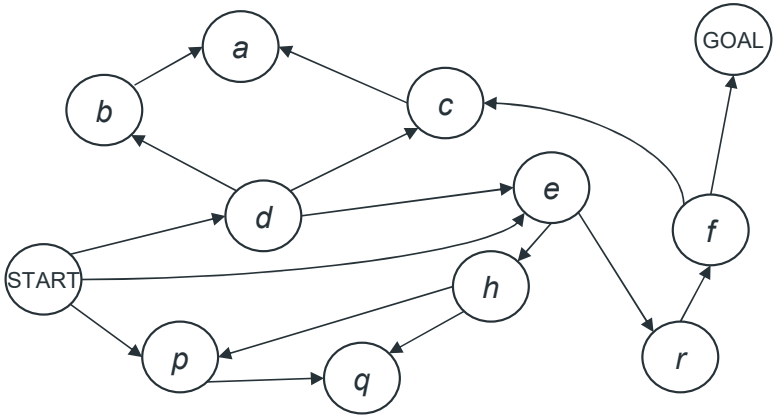
Tìm kiếm Theo Chiều Rộng



Tìm kiếm Theo Chiều Rộng



Ghi nhớ đường đi

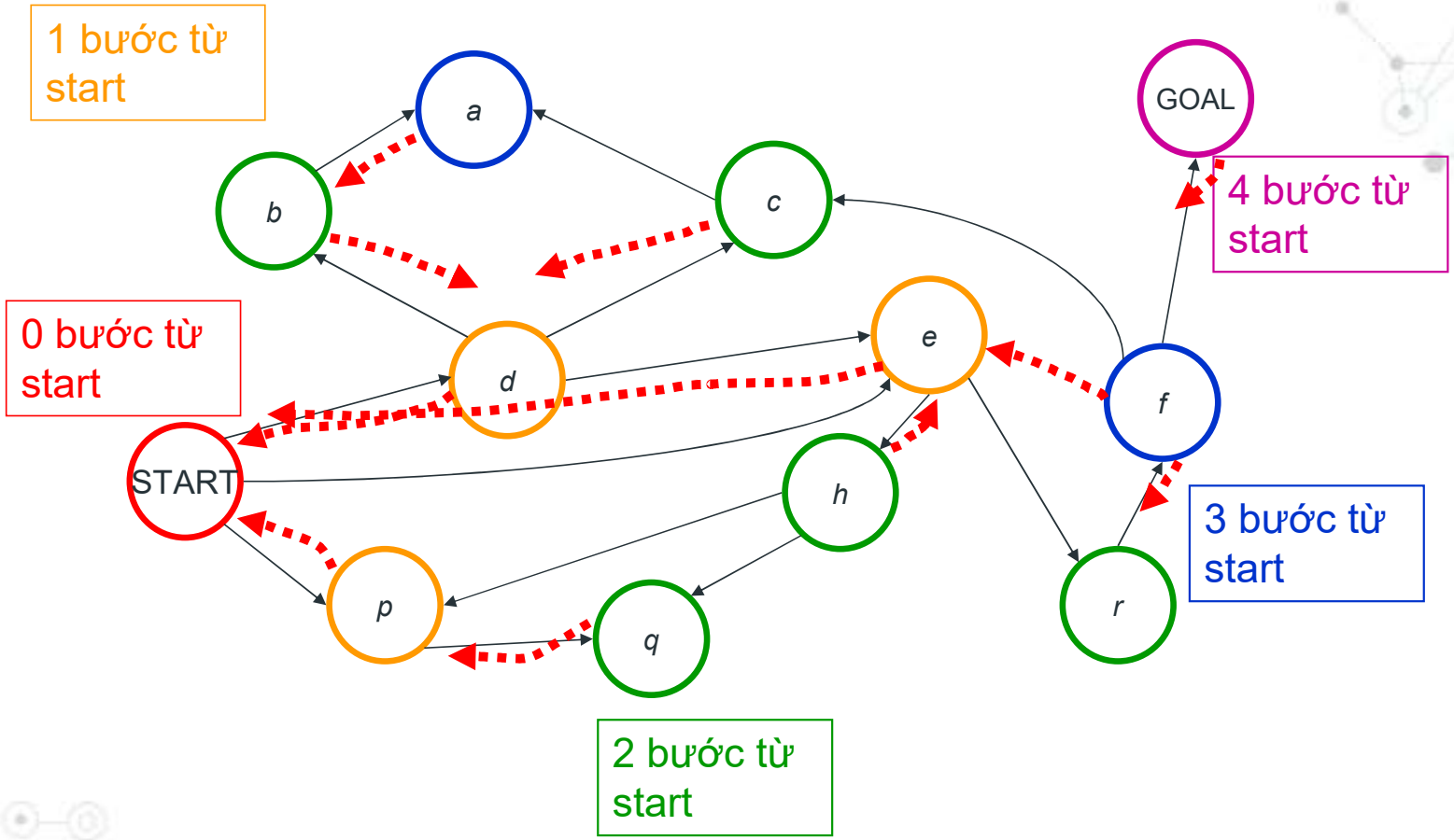


Ngoài ra, khi gán nhãn một trạng thái, ghi nhận trạng thái trước đó. Ghi nhận này được gọi là *con trỏ quay lui*. Lịch sử trước đó được dùng để phát sinh con đường lời giải, khi đã tìm được đích:

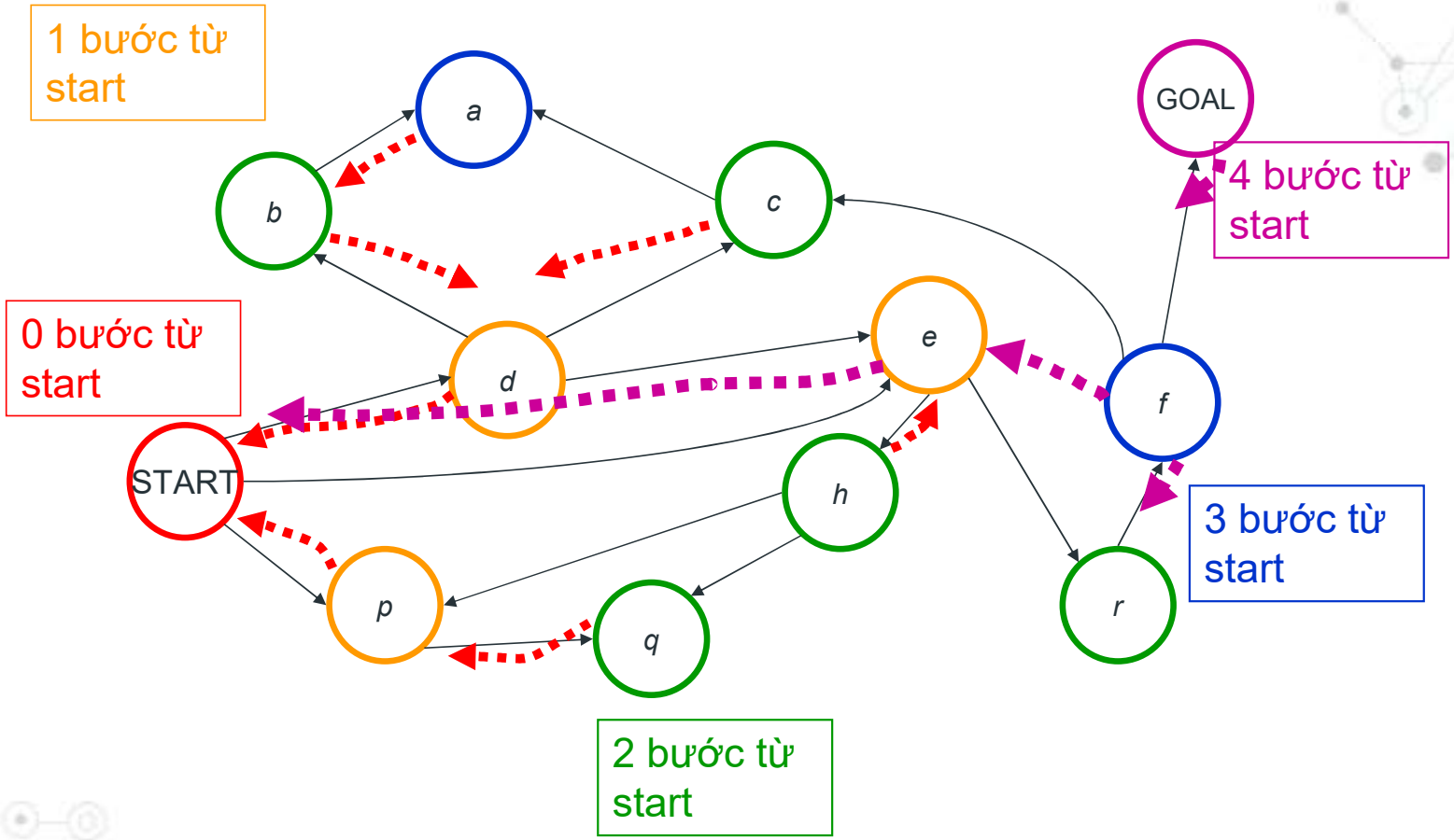
"Tôi đã đến đích. Tôi thấy mình đã ở f trước đó. Và tôi đã ở *r* trước khi tới *f*. Và…

…. do đó con đường lời giải là S → *e* → *r* → *f* → G"

Con trỏ quay lui



Con trỏ quay lui

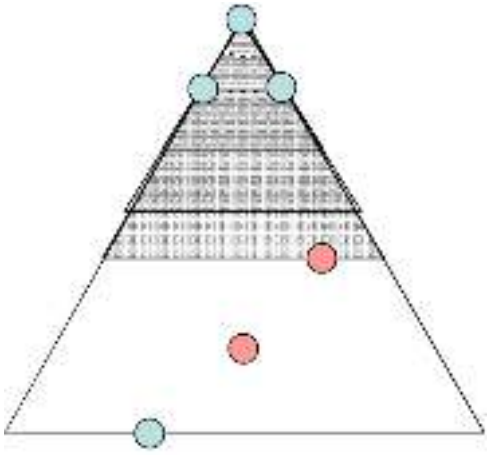


Giải thuật BFS

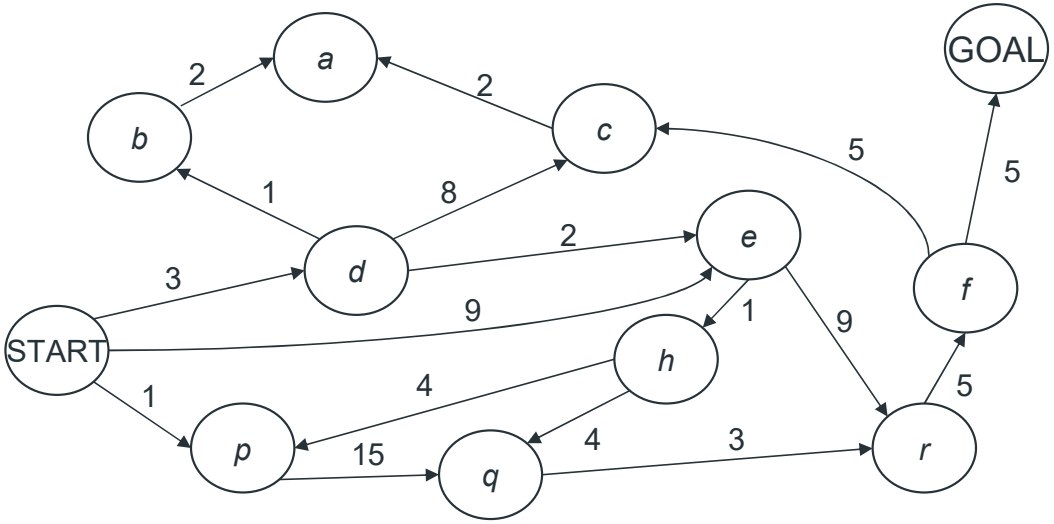
V_0 := S (tập các trạng thái ban đầu)
previous(*START*) := *NIL*
k := *0*
while (không có trạng thái đích trong V_k và V_k khác rỗng) **do**
    V_{k+1} := tập rỗng
    Với mỗi trạng thái s trong V_k
        Với mỗi trạng thái s' trong **succs**(s)
            Nếu s' chưa gán nhãn
                Đặt *previous*(s') := s
                Thêm s' vào V_{k+1}
    k := k+*1*
If V_k rỗng thì FAILURE
Else xây dựng lời giải: Đặt S_i là trạng thái thứ i trên đường đi ngắn nhất.
Định nghĩa S_k = GOAL, và với mọi i <= k, định nghĩa S_{i-1} = *previous*(S_i).

Nhận xét về BFS

- BFS duyệt cây theo hướng từ trên xuống dưới
- BFS có đảm bảo tìm được lời giải (nếu tồn tại)?
  - Có
- BFS có đảm bảo tìm được lời giải tối ưu (nếu tồn tại)?
  - BFS tìm được lời giải "nông" nhất; lời giải "nông" nhất là lời giải tối ưu khi tất cả các cạnh đều có chi phí bằng nhau



Chi phí chuyển đổi khác nhau



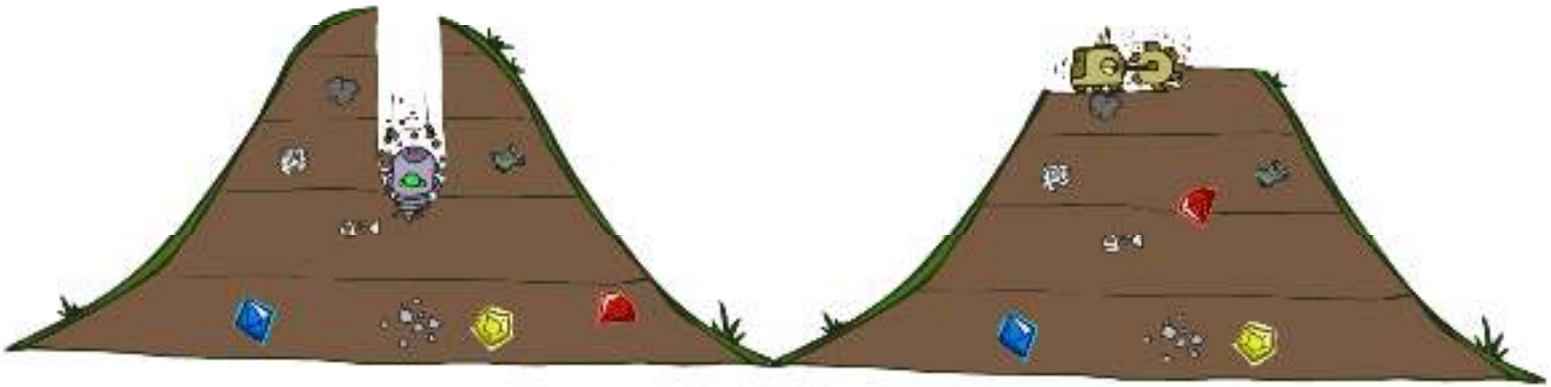
Nếu chi phí chuyển trạng thái là khác nhau thì BFS không thể trả về lời giải tối ưu.

Tại sao?

Vậy phương pháp nào để tìm được lời giải tối ưu trong trường hợp này?

Minh họa BFS

So sánh DFS và BFS

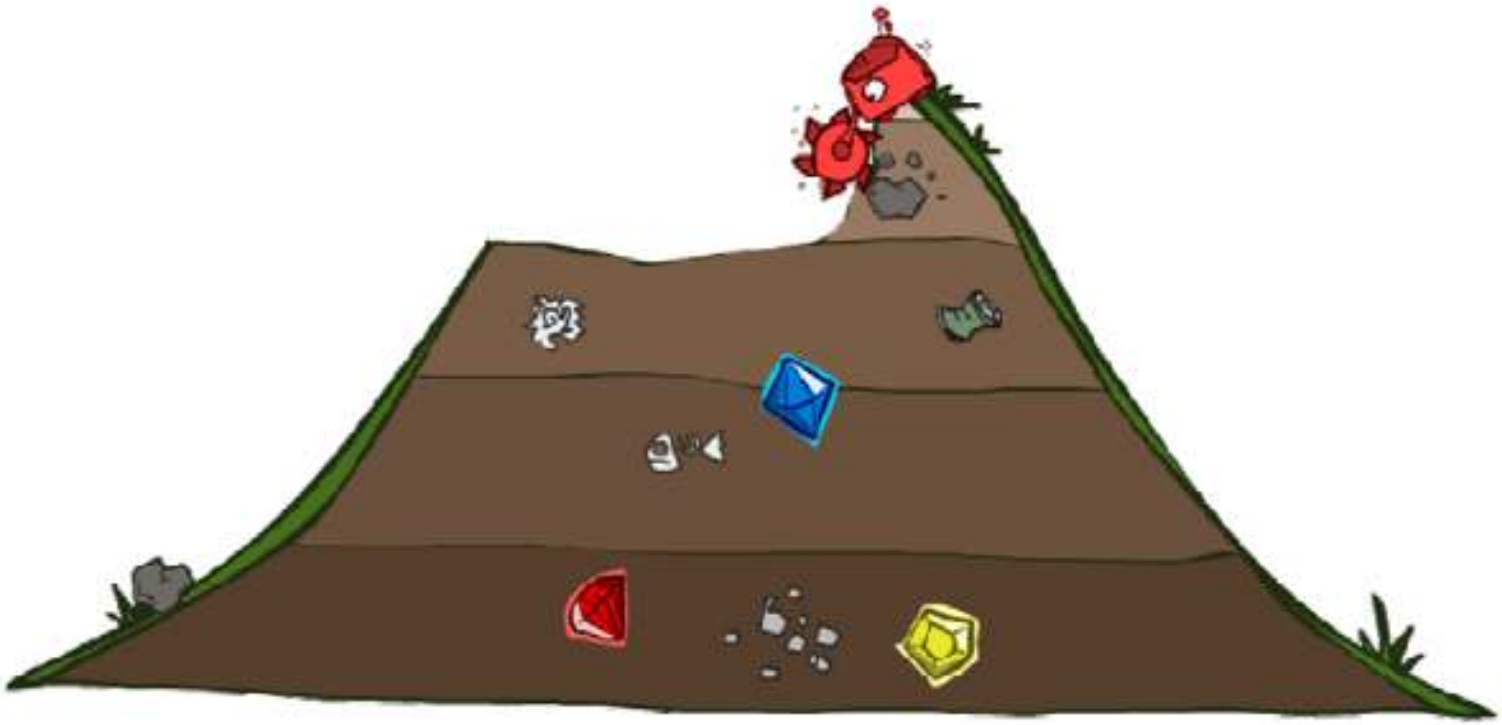


Cái nào tốt hơn?

Nội dung

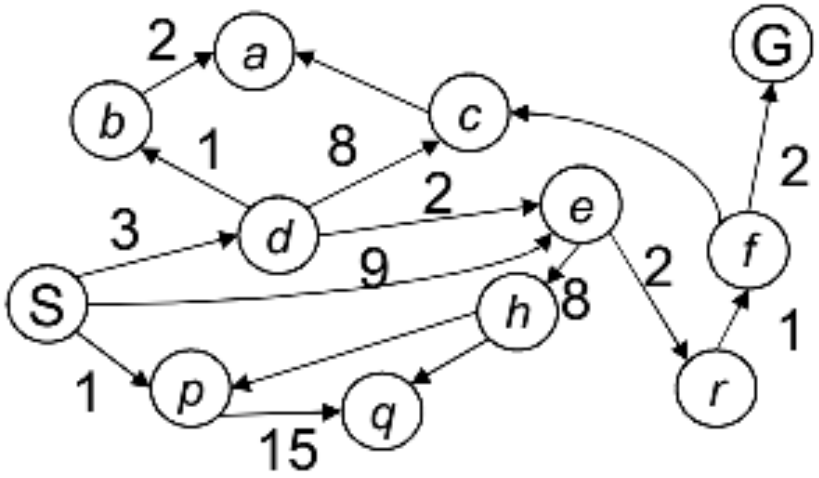
- Hệ thống lên kế hoạch (planning agent)
- Bài toán tìm kiếm (bài toán lên kế hoạch)
- **Giải quyết bài toán bằng tìm kiếm mù**
  - Thuật toán Depth-First Search
  - Thuật toán Breadth-First Search
  - Thuật toán Uniform Cost Search
  - Một số thuật toán khác
- Đánh giá các thuật toán

Thuật toán Uniform Cost Search (UCS)

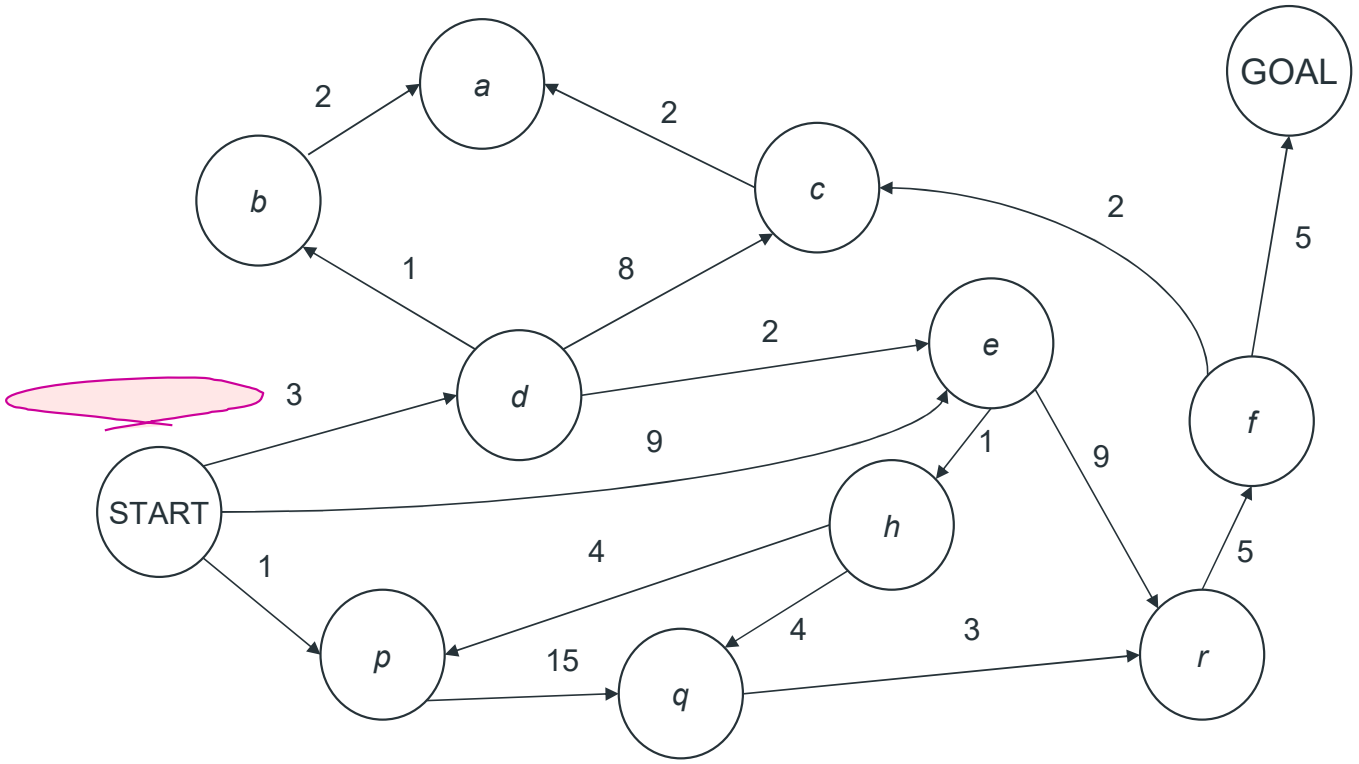


Thuật toán UCS

- Chiến lược chọn kế hoạch ở fringe để mở rộng: chọn kế hoạch **có chi phí nhỏ nhất**
- Có thể dùng priority queue để cài đặt fringe
- Chạy UCS với đồ thị ở dưới:

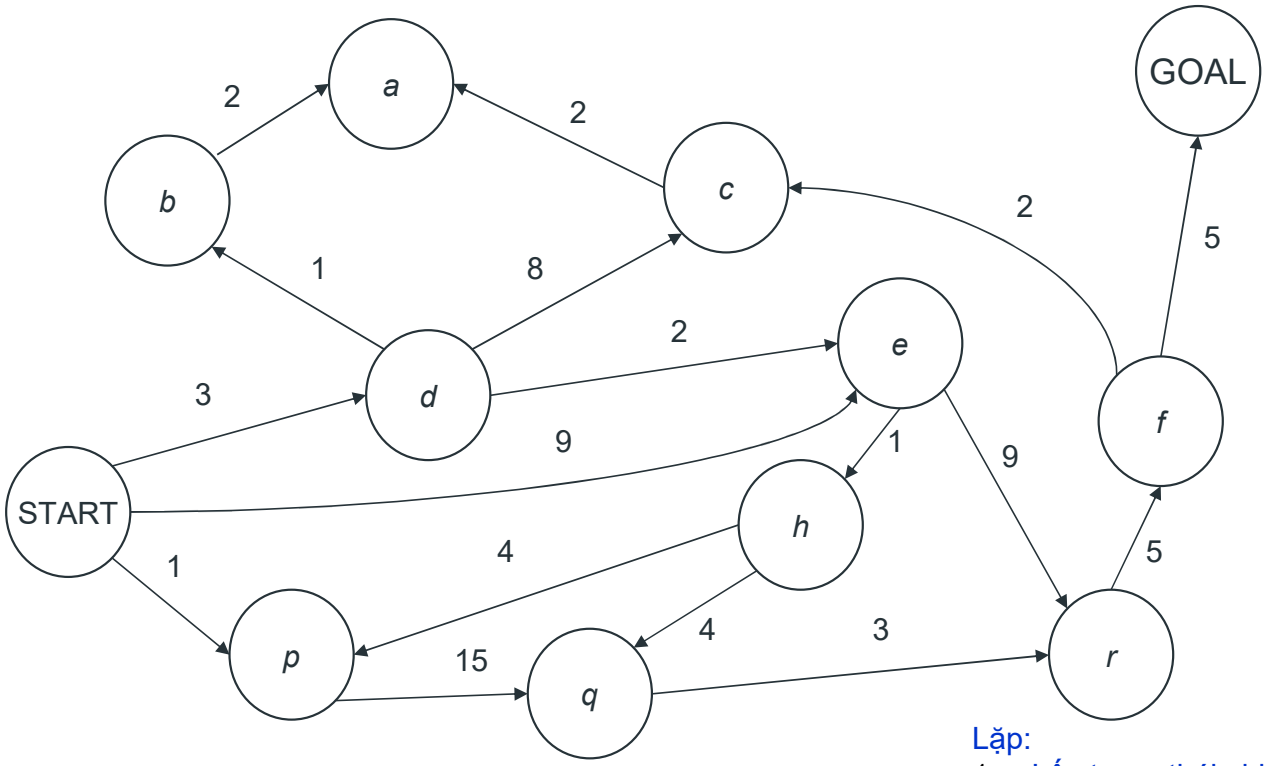


Bắt đầu UCS



PQ = { (S,0) }

Lặp UCS

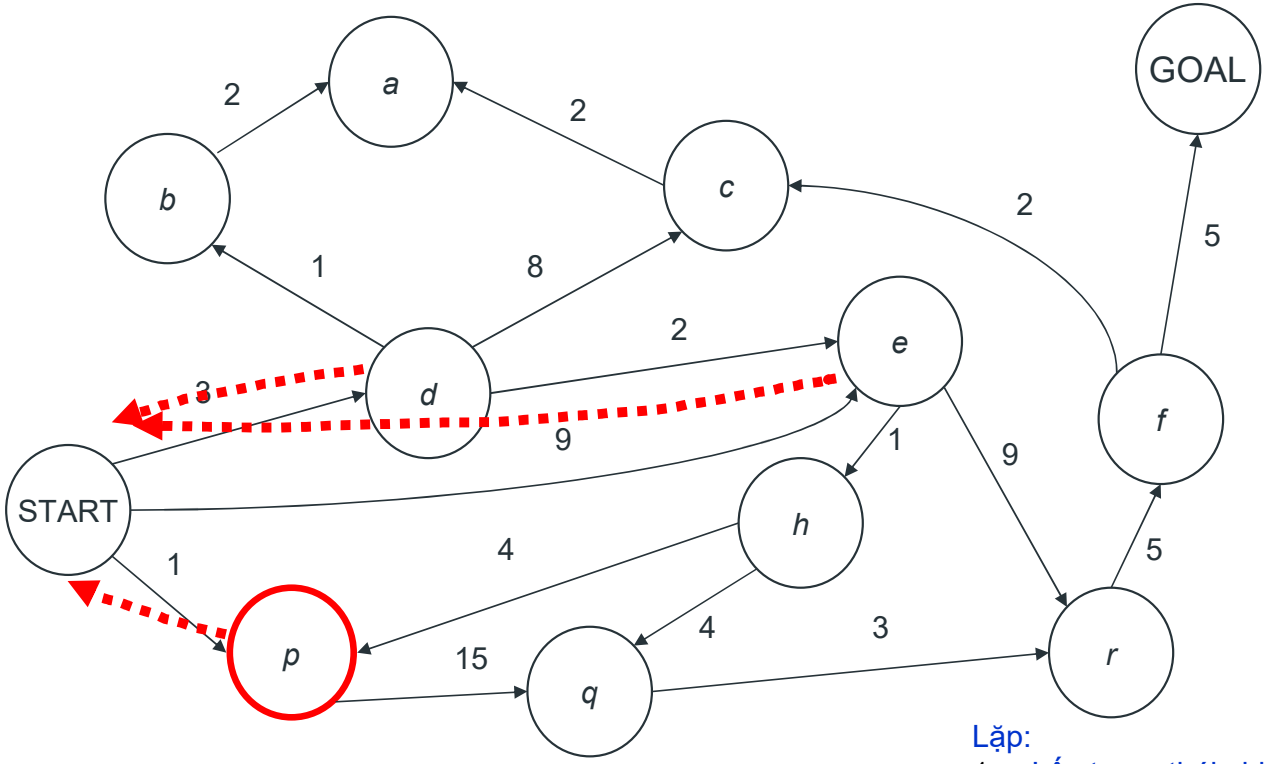


Lặp:

1. Lấy trạng thái chi phí thấp nhất từ PQ
2. Thêm các con

PQ = { (S,0) }

Lặp UCS

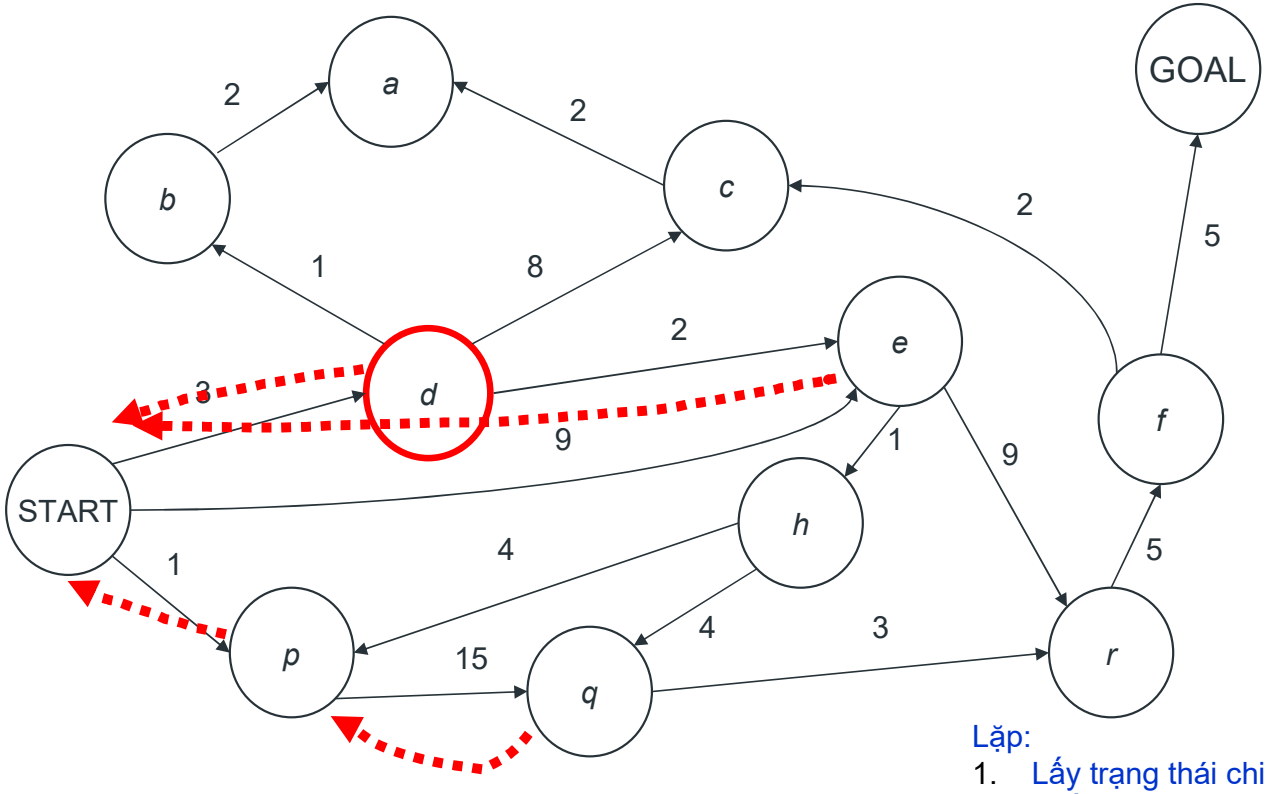


Lặp:

1. Lấy trạng thái chi phí thấp nhất từ PQ
2. Thêm các con

PQ = { (p,1), (d,3) , (e,9) }

Lặp UCS

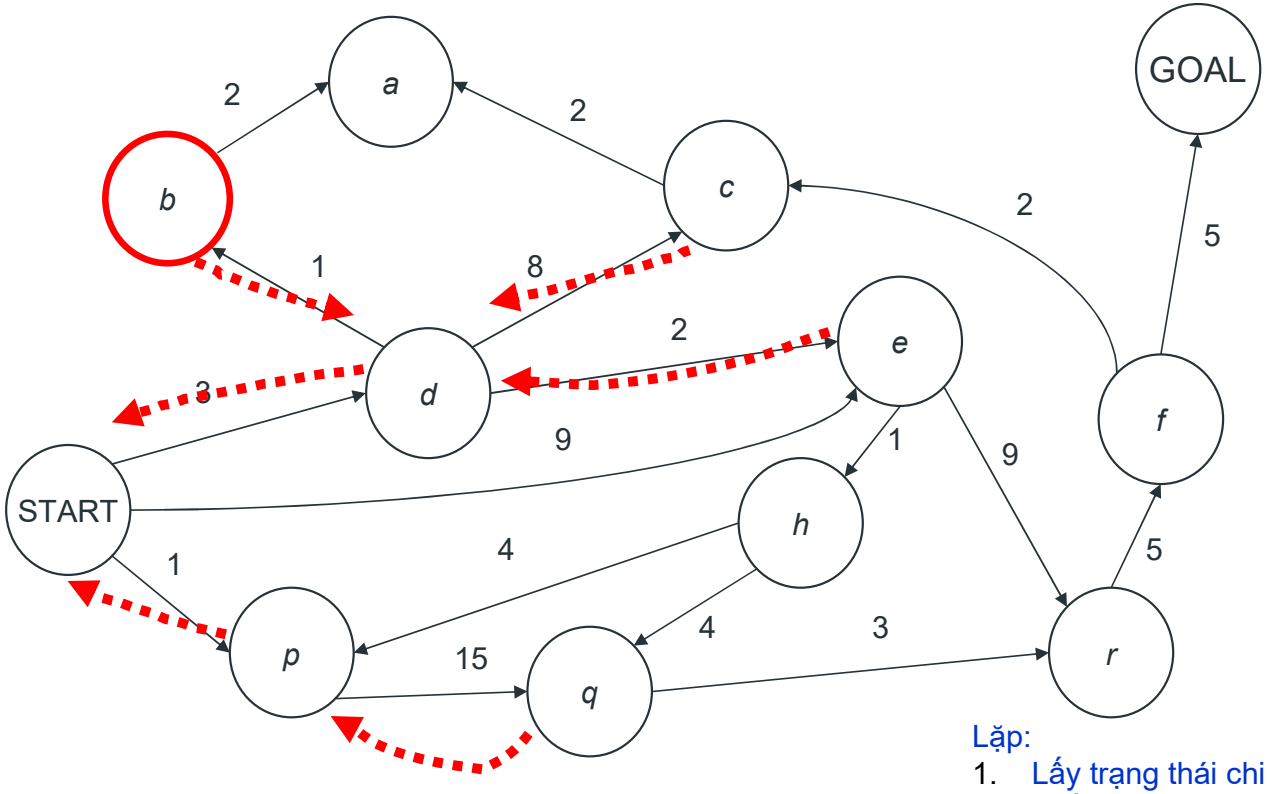


Lặp:

1. Lấy trạng thái chi phí thấp nhất từ PQ
2. Thêm các con

PQ = { (d,3) , (e,9) , (q,16) }

Lặp UCS

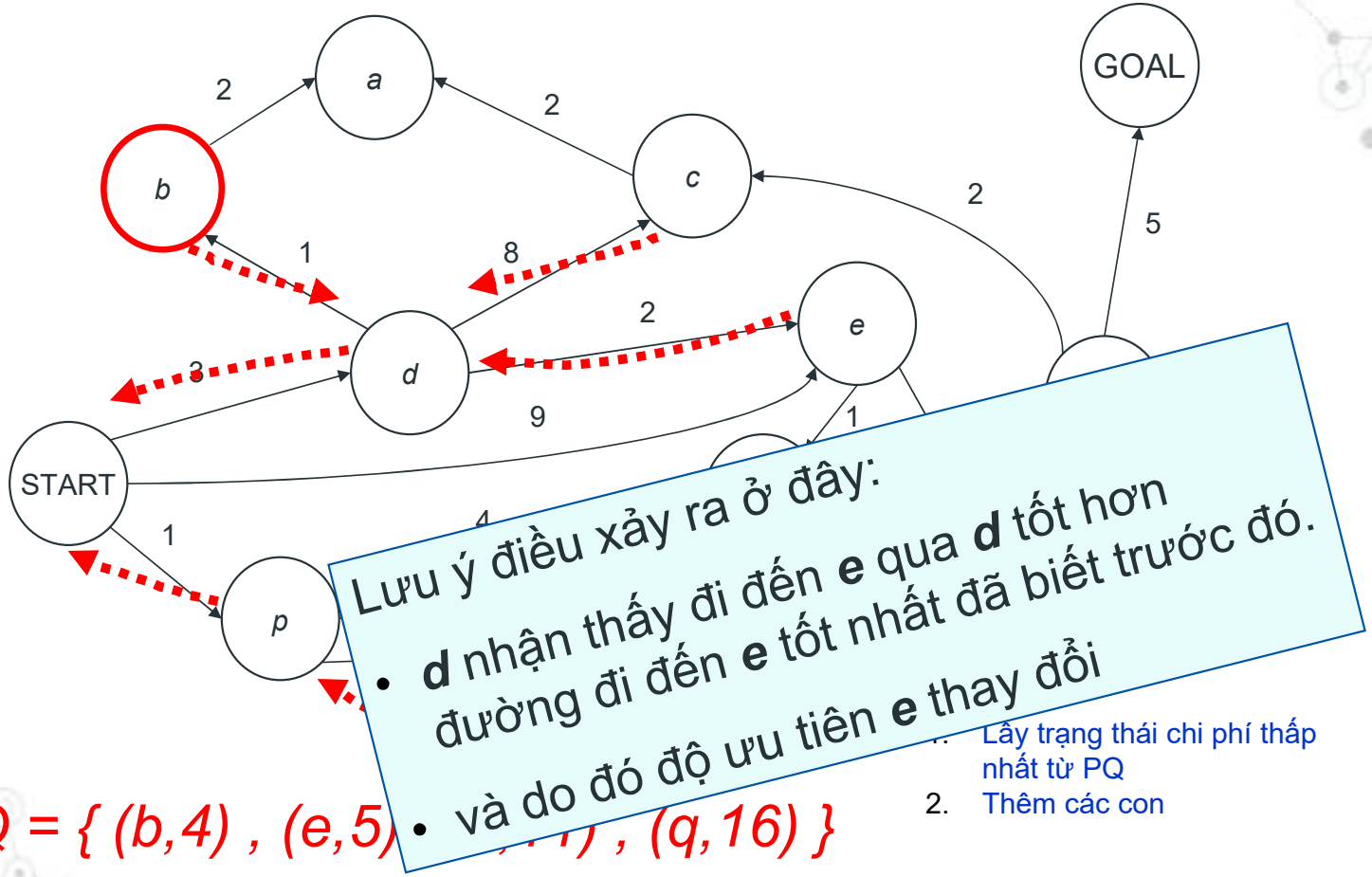


Lặp:

1. Lấy trạng thái chi phí thấp nhất từ PQ
2. Thêm các con

PQ = { (b,4) , (e,5) , (c,11) , (q,16) }

Lặp UCS



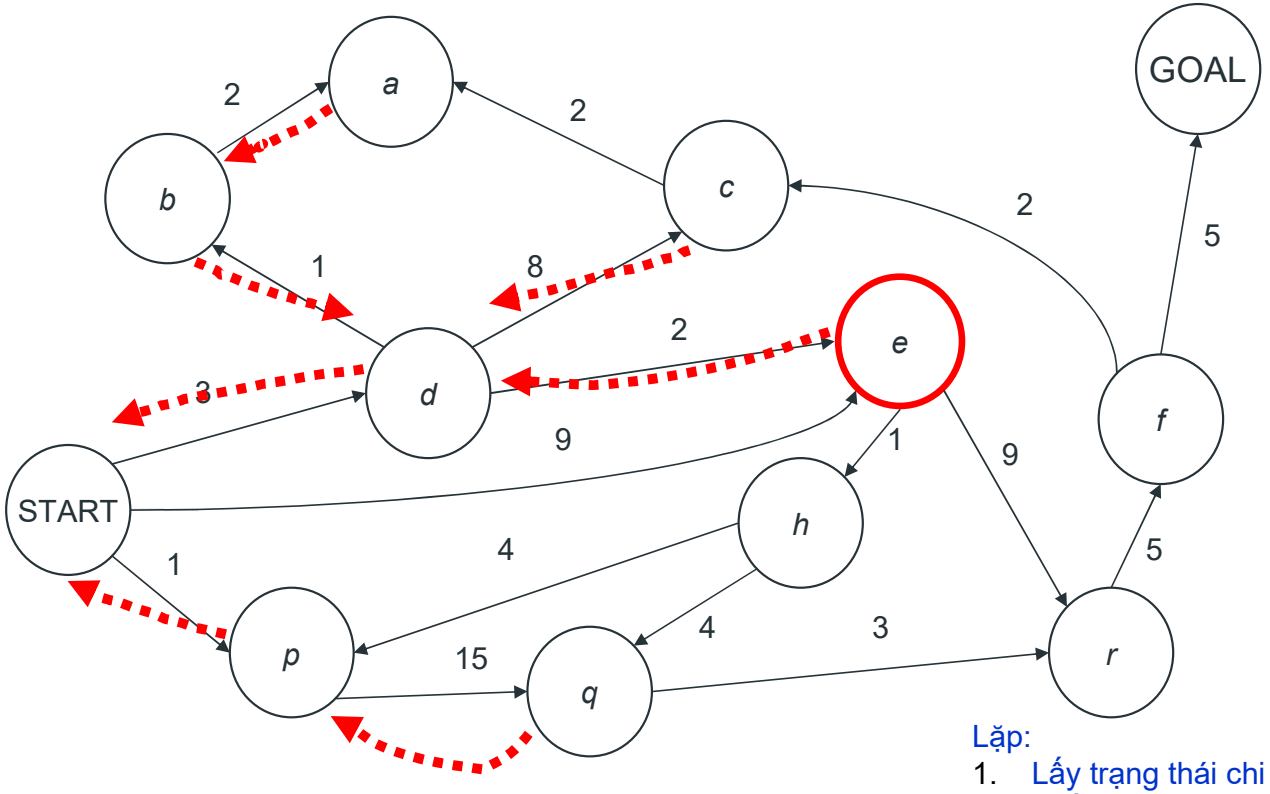
Lưu ý điều xảy ra ở đây:

- **d** nhận thấy đi đến **e** qua **d** tốt hơn đường đi đến **e** tốt nhất đã biết trước đó.
- và do đó độ ưu tiên **e** thay đổi

PQ = { (b,4) , (e,5) , (q,16) }

1. Lấy trạng thái chi phí thấp nhất từ PQ
2. Thêm các con

Lặp UCS

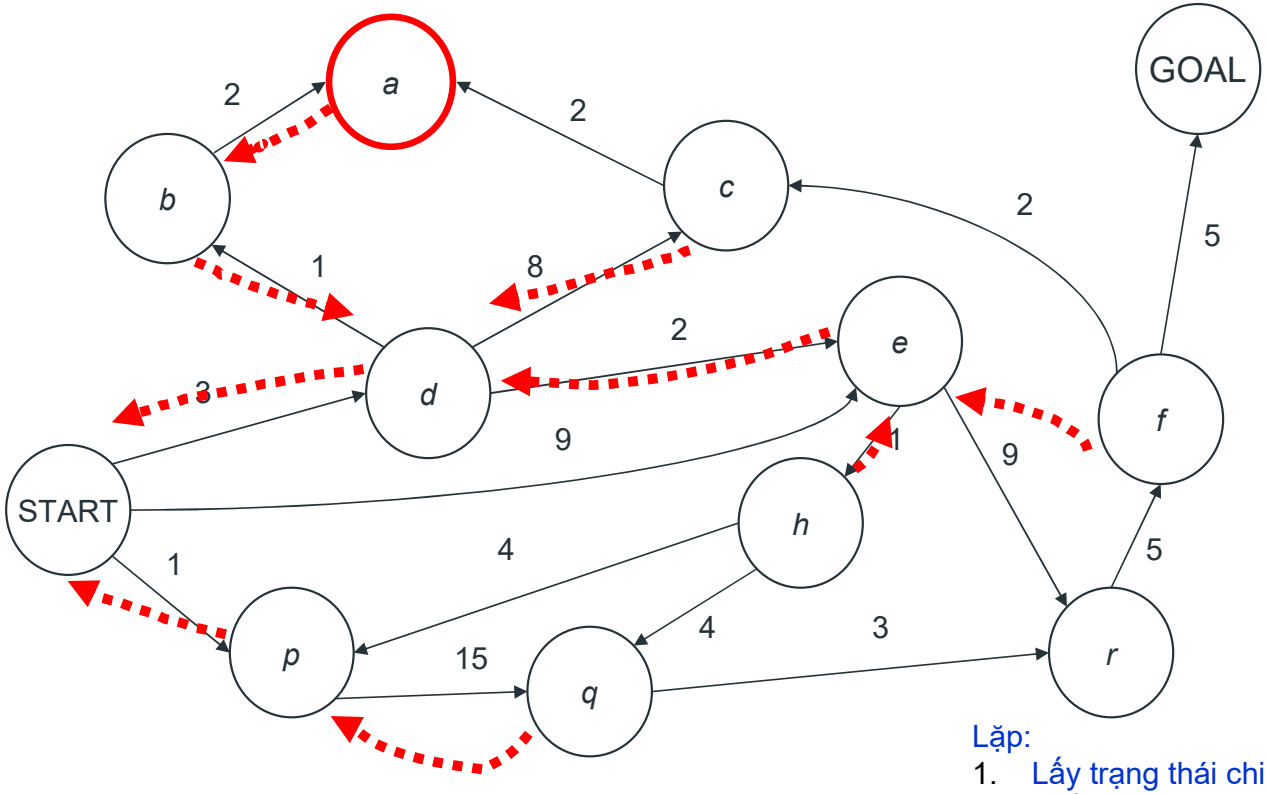


Lặp:

1. Lấy trạng thái chi phí thấp nhất từ PQ
2. Thêm các con

PQ = { (e,5) , (a,6) , (c,11) , (q,16) }

Lặp UCS

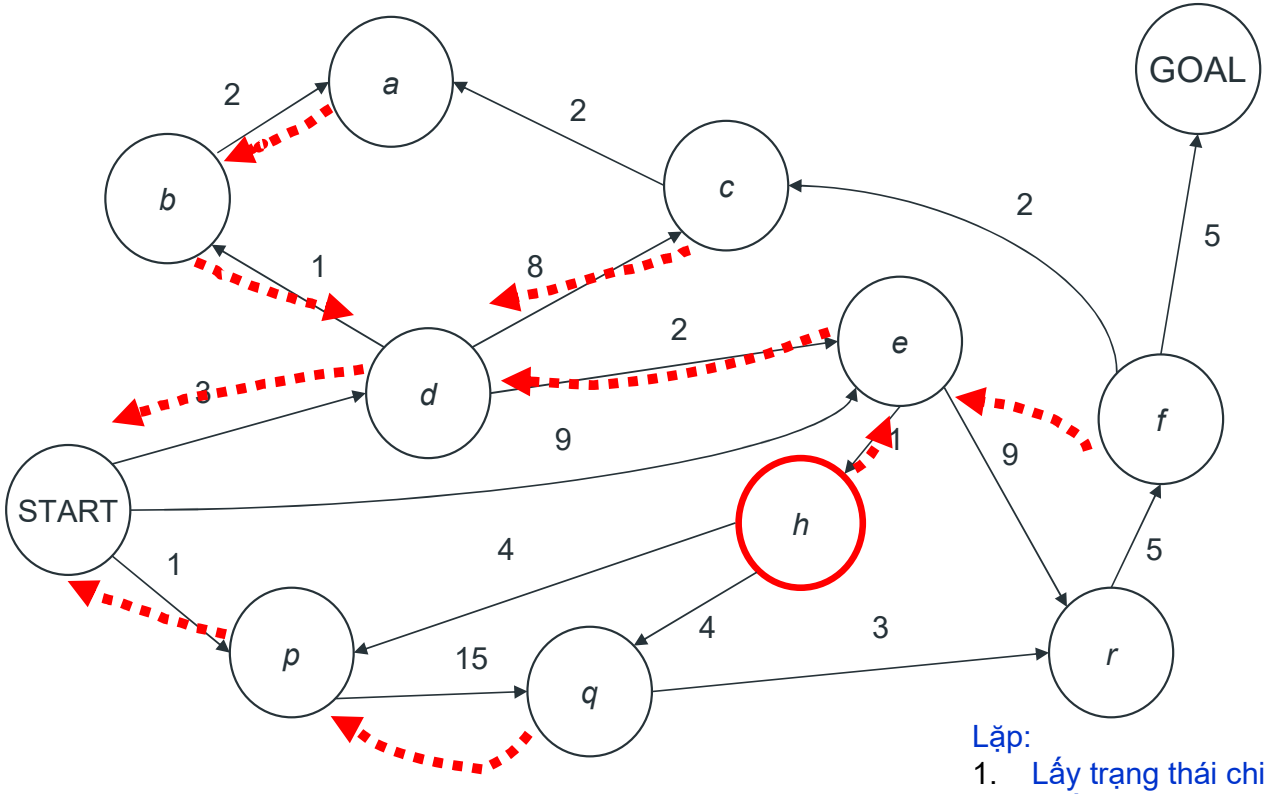


Lặp:

1. Lấy trạng thái chi phí thấp nhất từ PQ
2. Thêm các con

$PQ = \{ (a,6),(h,6),(c,11),(r,14),(q,16) \}$

Lặp UCS

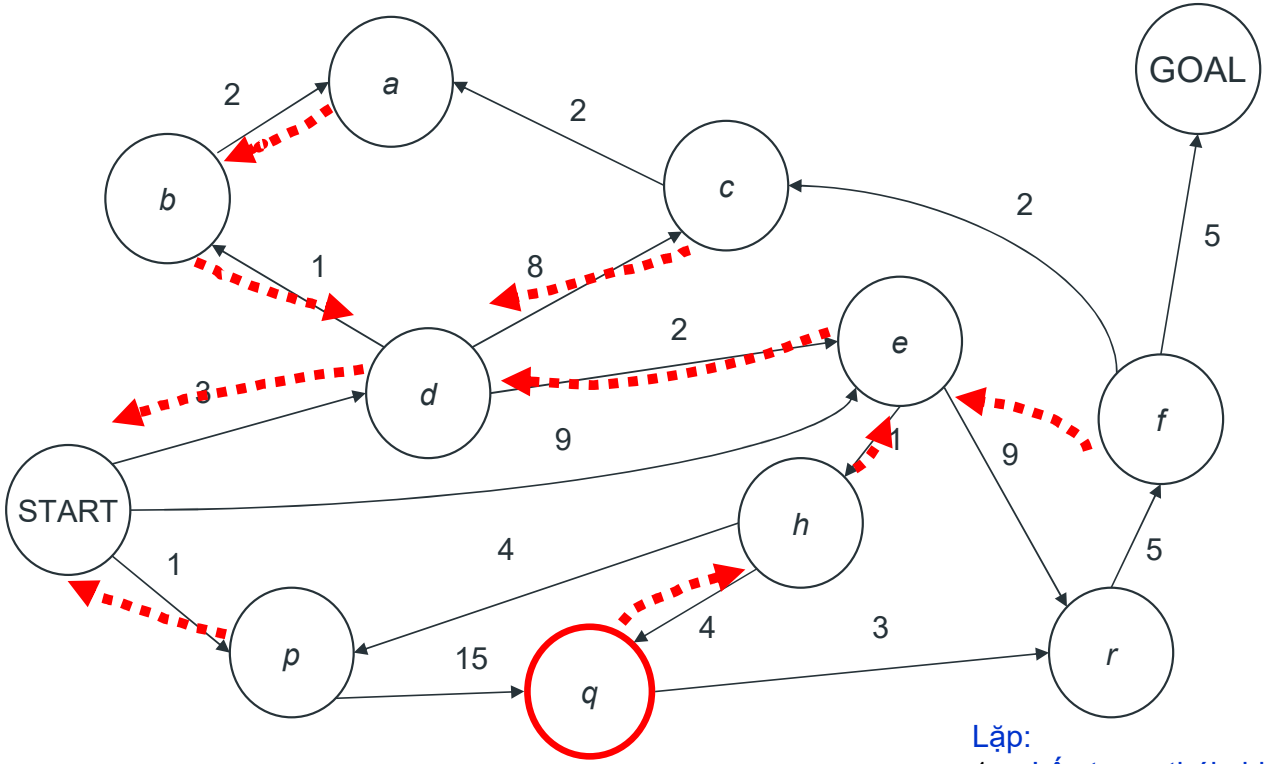


Lặp:

1. Lấy trạng thái chi phí thấp nhất từ PQ
2. Thêm các con

$PQ = \{ (h,6),(c,11),(r,14),(q,16) \}$

Lặp UCS



Lặp:

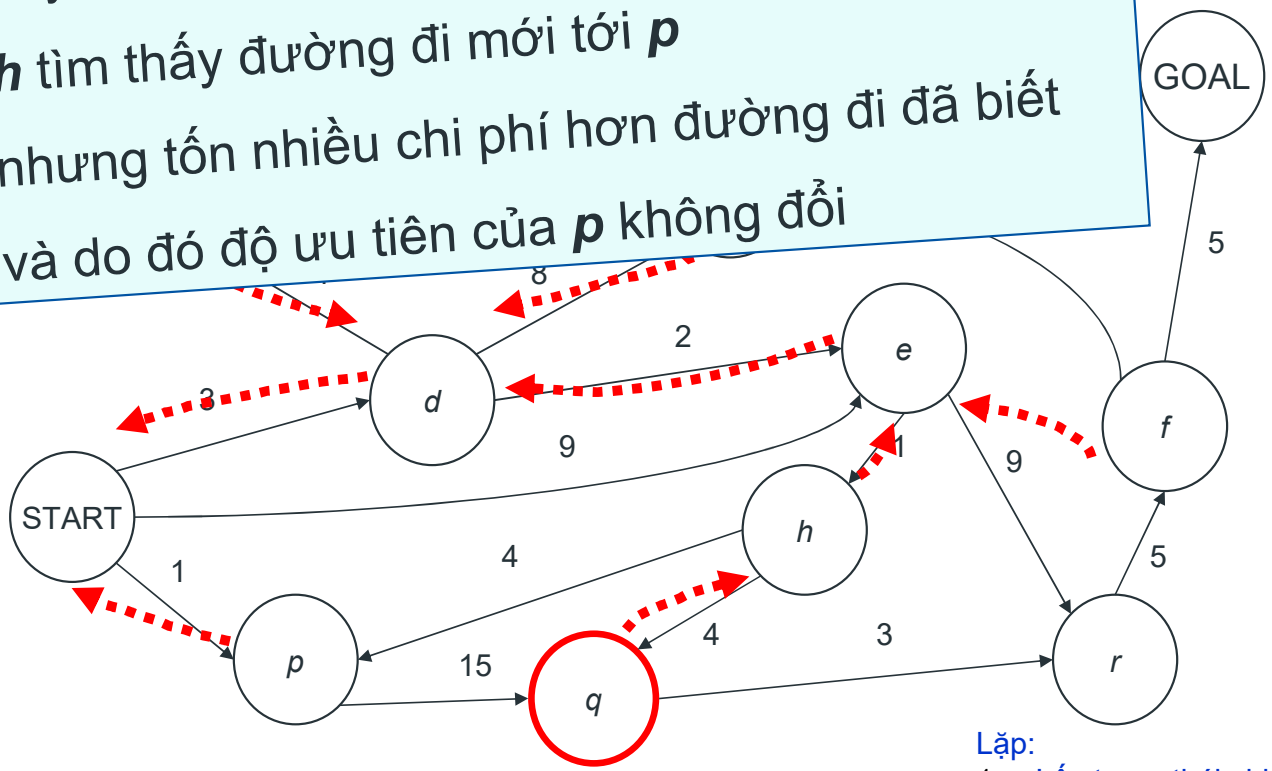
1. Lấy trạng thái chi phí thấp nhất từ PQ
2. Thêm các con

PQ = { (q,10), (c,11),(r,14) }

Lặp UCS

Lưu ý điều xảy ra ở đây:

- ***h*** tìm thấy đường đi mới tới ***p***
- nhưng tốn nhiều chi phí hơn đường đi đã biết
- và do đó độ ưu tiên của ***p*** không đổi

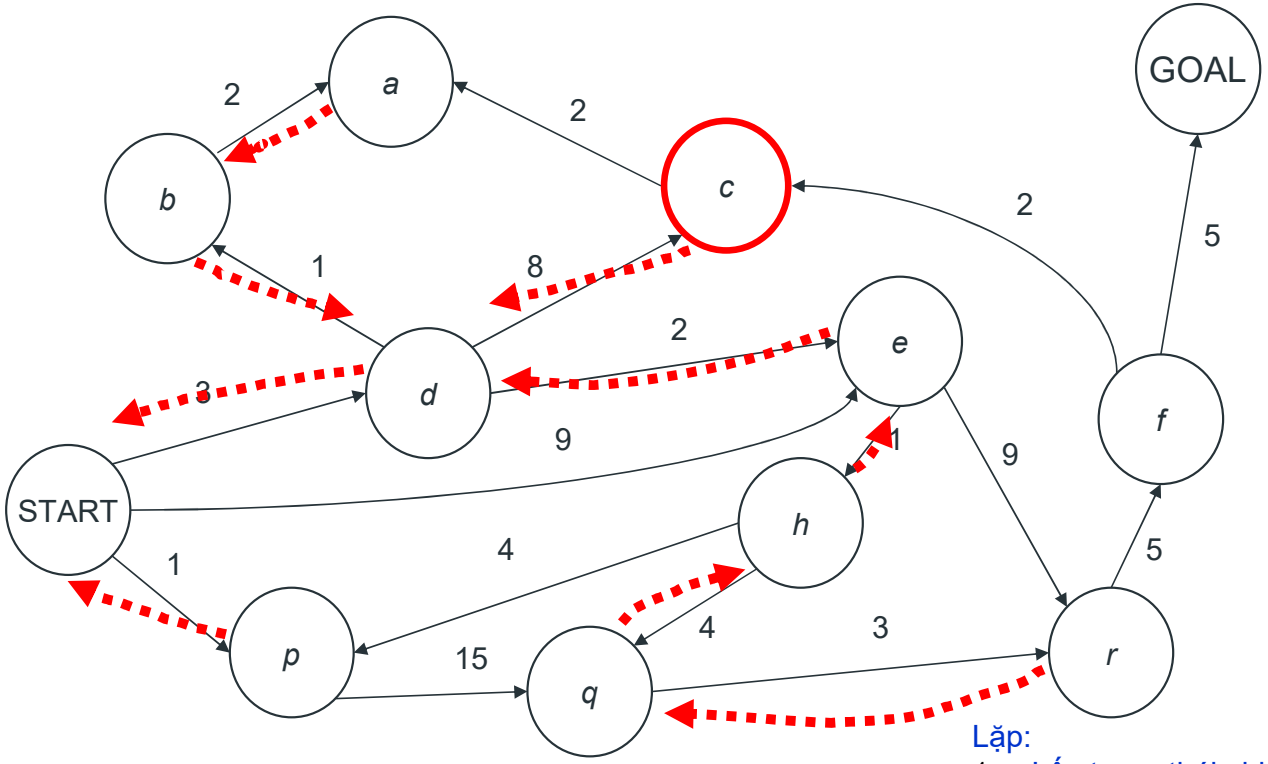


Lặp:

1. Lấy trạng thái chi phí thấp nhất từ PQ
2. Thêm các con

PQ = { (q,10), (c,11),(r,14) }

Lặp UCS

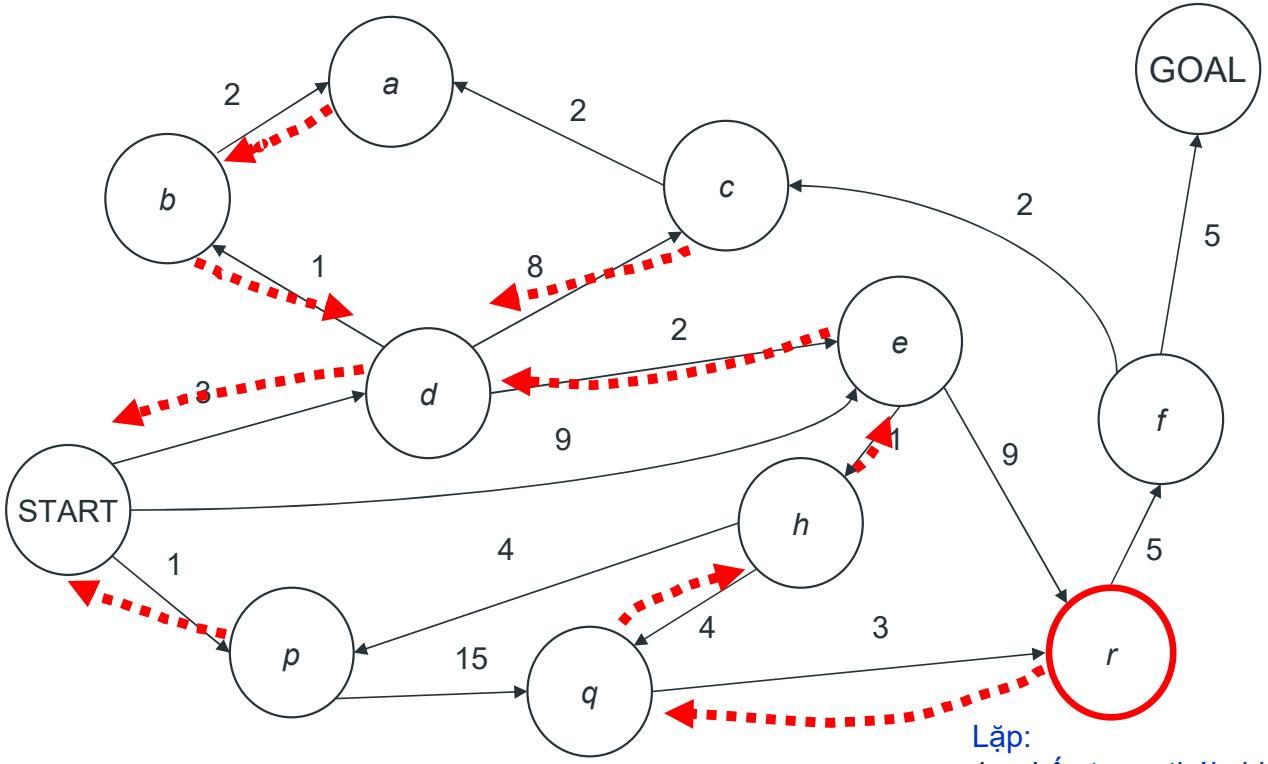


Lặp:

1. Lấy trạng thái chi phí thấp nhất từ PQ
2. Thêm các con

PQ = { (c,11),(r,13) }

Lặp UCS

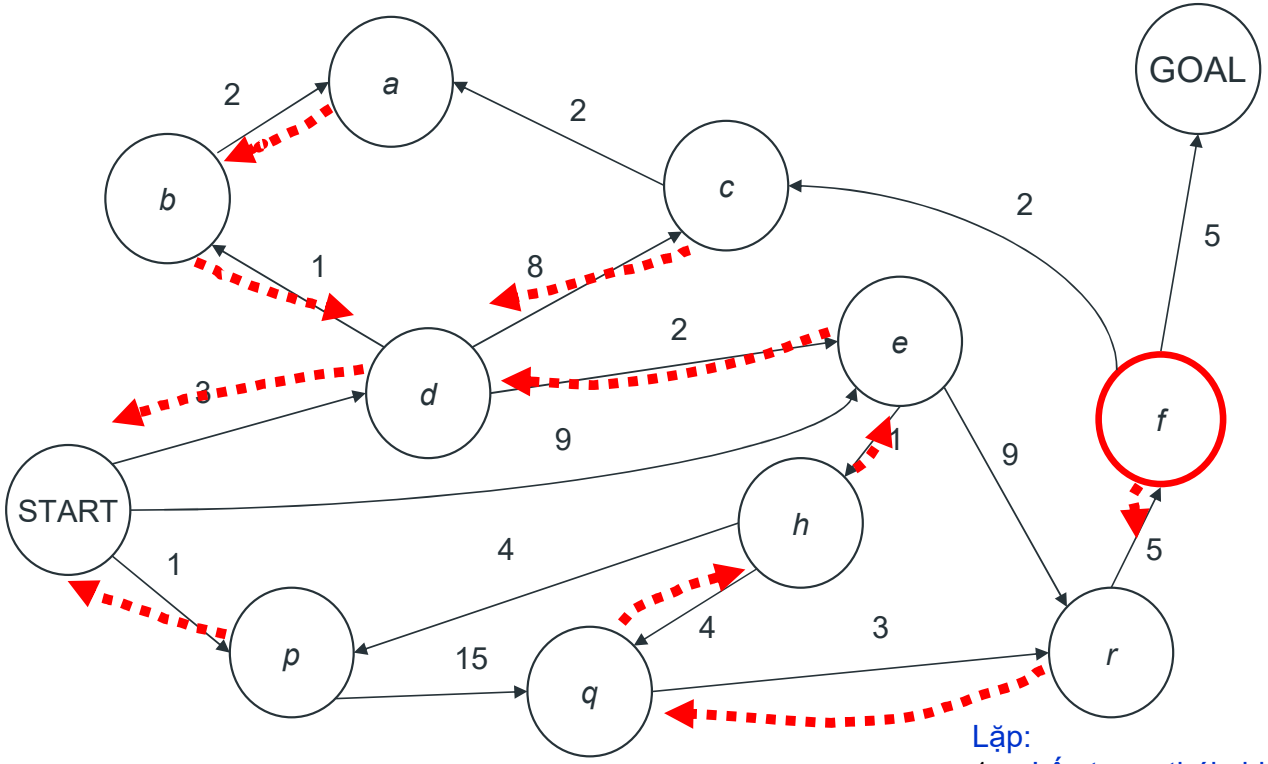


Lặp:

1. Lấy trạng thái chi phí thấp nhất từ PQ
2. Thêm các con

PQ = { (r,13) }

Lặp UCS

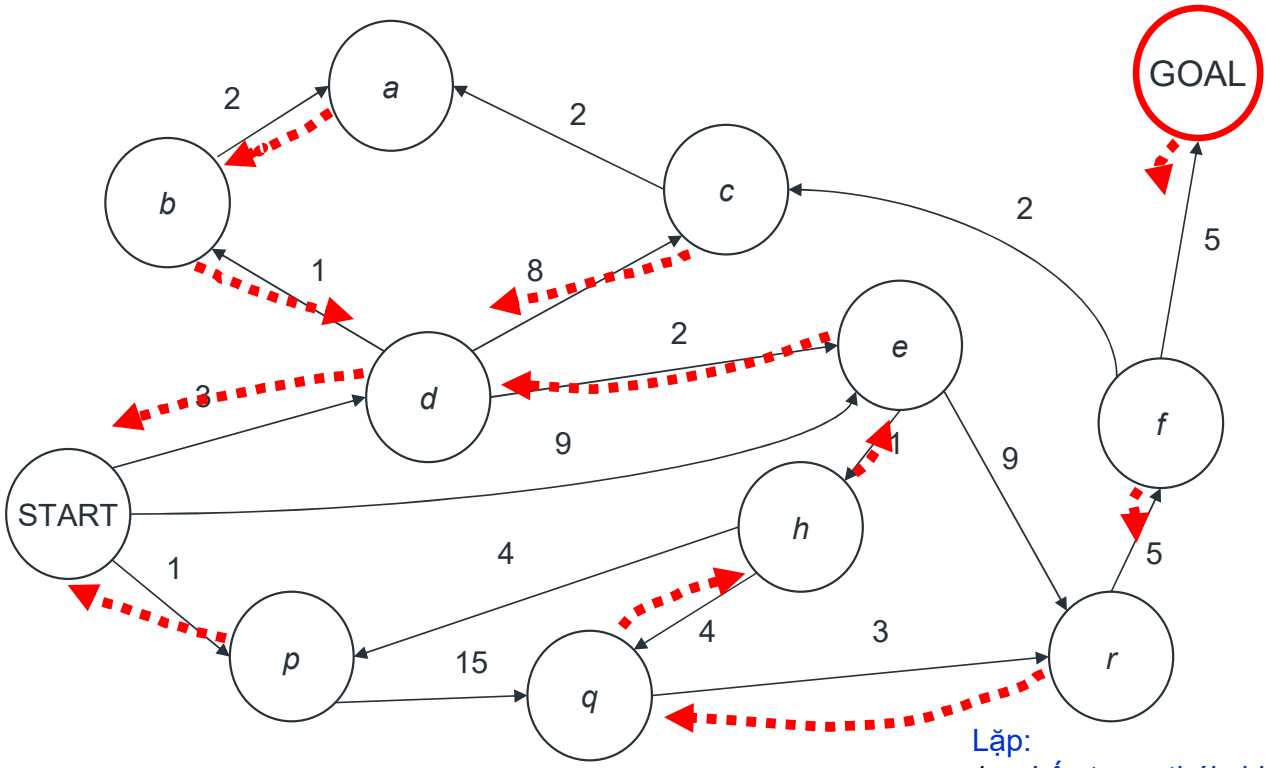


Lặp:

1. Lấy trạng thái chi phí thấp nhất từ PQ
2. Thêm các con

PQ = { (f,18) }

Lặp UCS



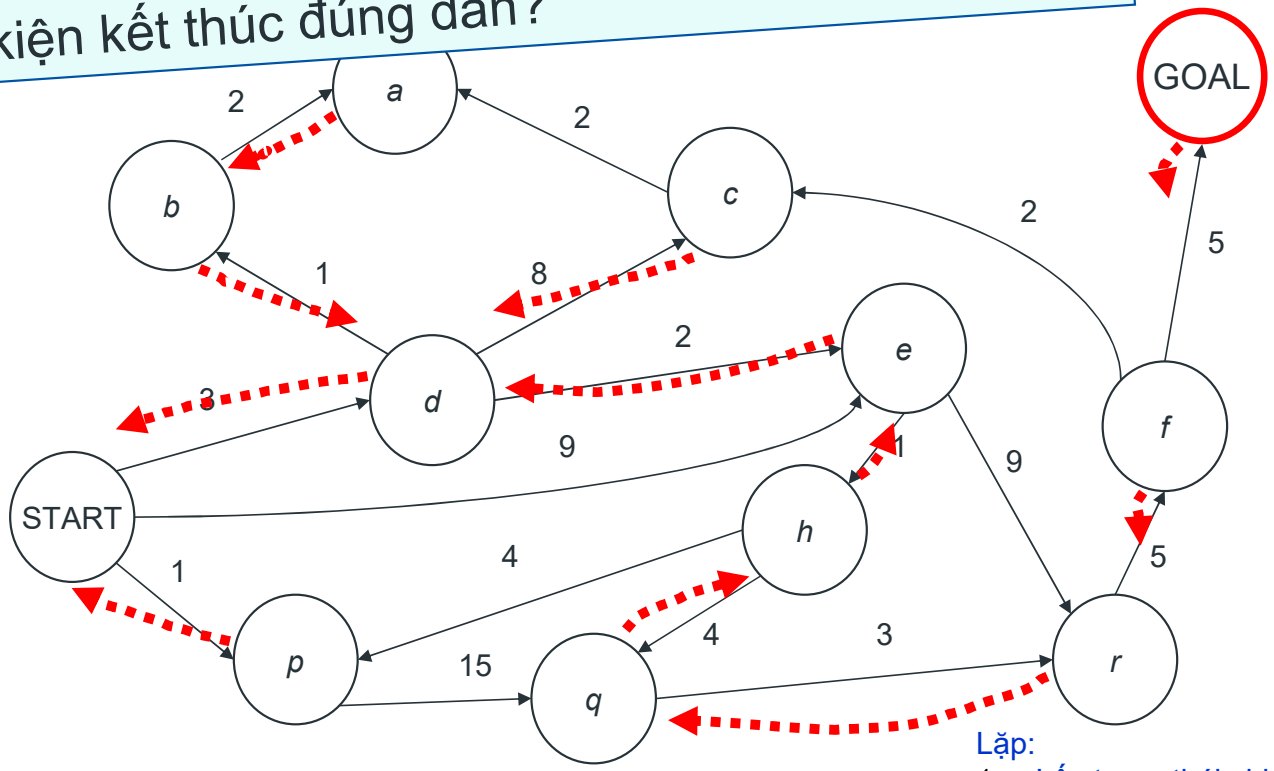
Lặp:

1. Lấy trạng thái chi phí thấp nhất từ PQ
2. Thêm các con

PQ = { (G,23) }

Lặp UCS

Câu hỏi: "Dừng ngay khi thấy đích" có là một điều kiện kết thúc đúng đắn?

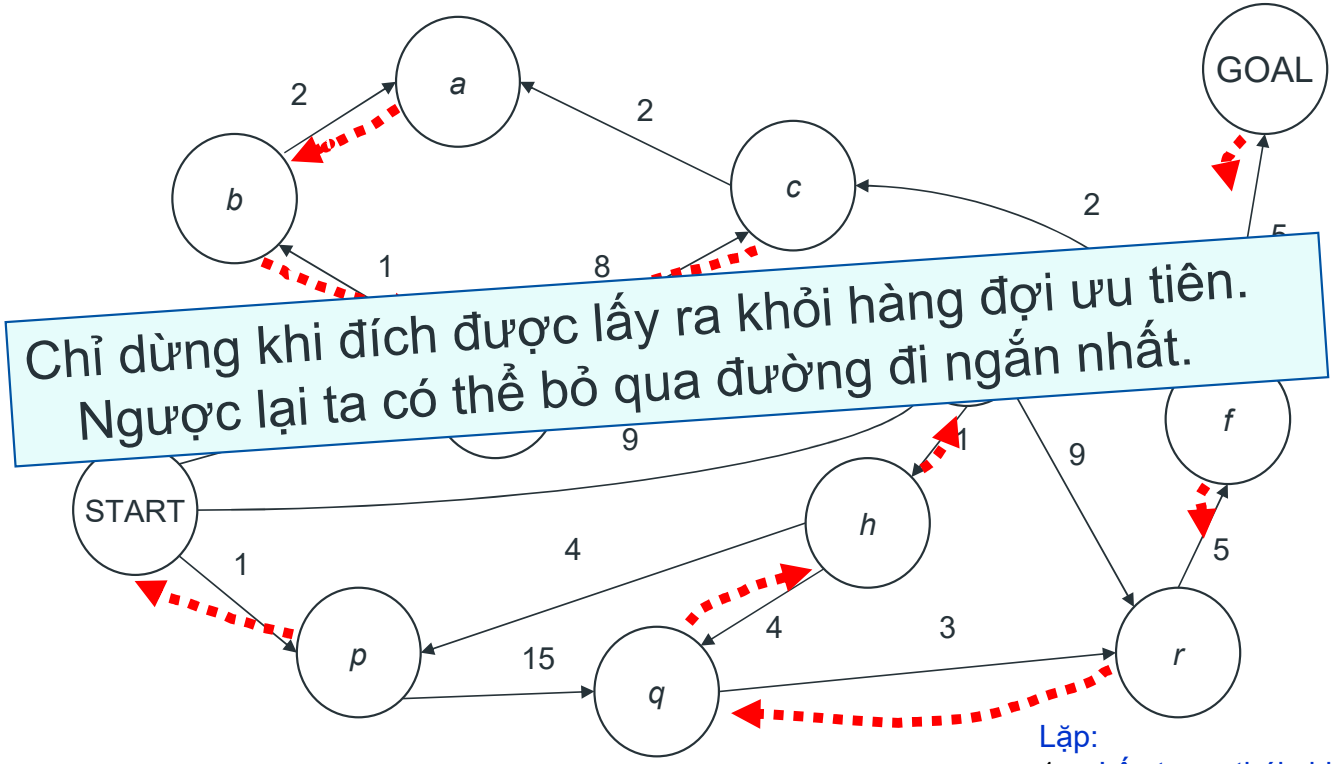


Lặp:

1. Lấy trạng thái chi phí thấp nhất từ PQ
2. Thêm các con

PQ = { (G,23) }

Lặp UCS

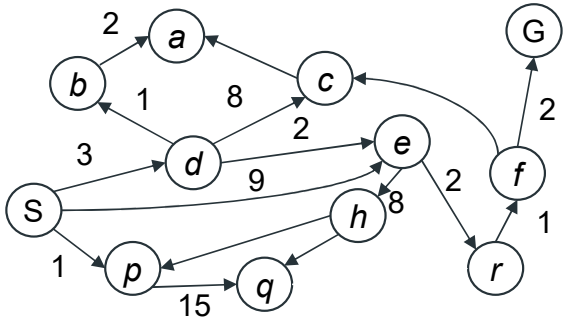


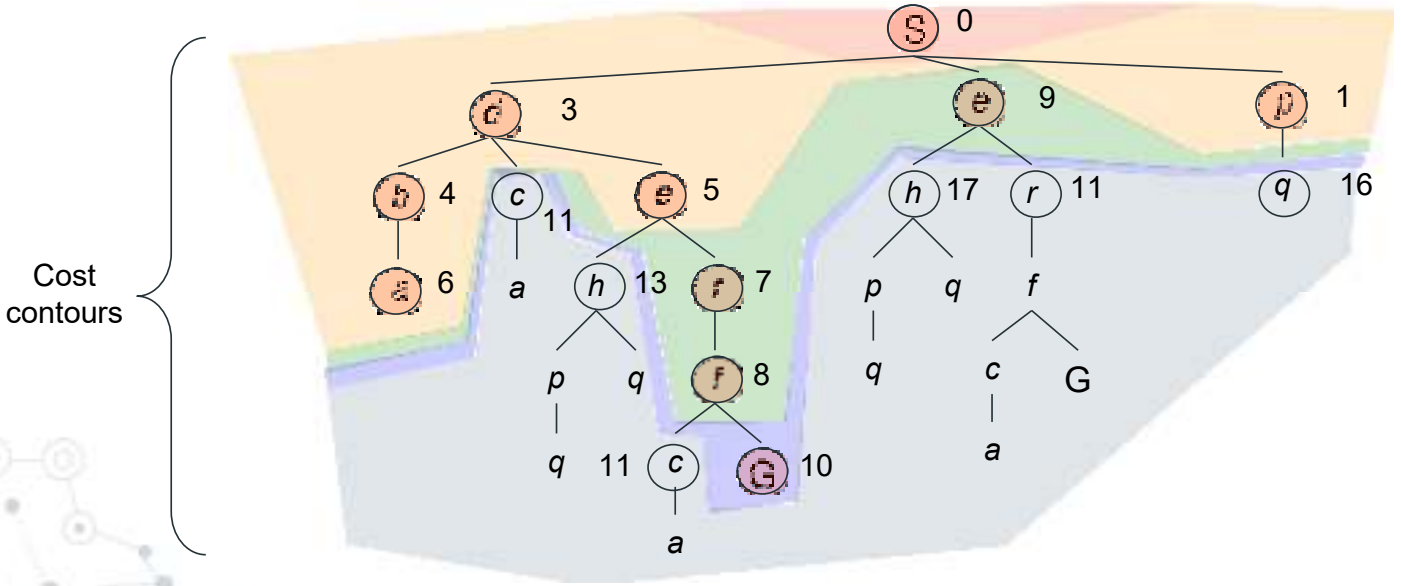
Lặp:

1. Lấy trạng thái chi phí thấp nhất từ PQ
2. Thêm các con

PQ = { }

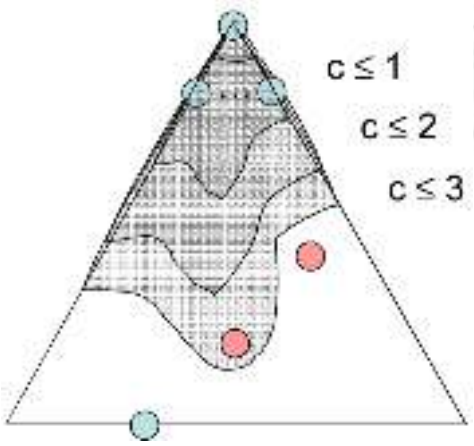
Biểu diễn cây tìm kiếm





Nhận xét về UCS

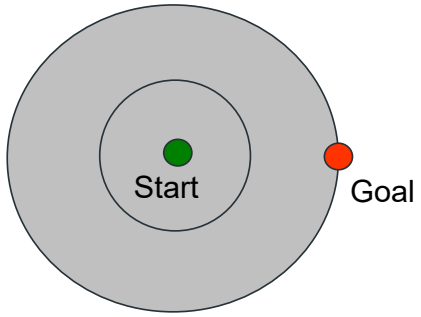
- UCS duyệt cây tương tự như BFS, nhưng UCS mở rộng theo chí phí tăng dần (thay vì mở rộng theo chiều sâu tăng dần như ở BFS)
- UCS có đảm bảo tìm được lời giải (nếu tồn tại)?
  - Có
- UCS có đảm bảo tìm được lời giải tối ưu (nếu tồn tại)?
  - Có



Nhận xét về UCS

- UCS khám phá chi phí tăng dần theo dạng đường bao.
  - Phải duyệt tất cả các trạng thái ở mọi hướng.

→ Chi phí tính toán cao

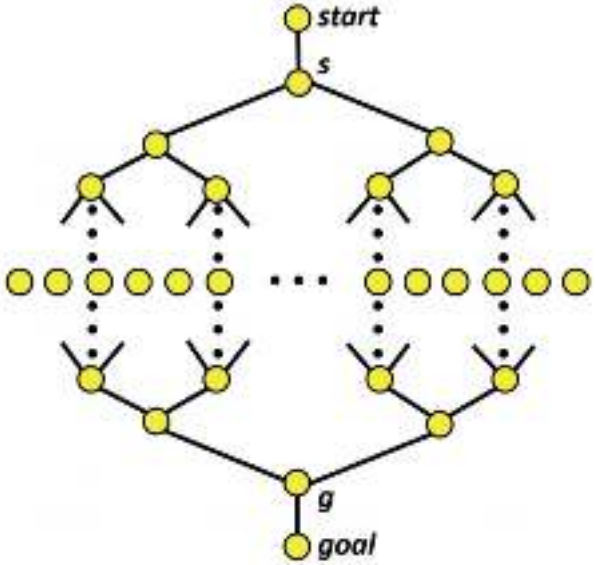


Minh họa UCS

Một số thuật toán khác

Depth-Limited Search

Iterative Deepening Search



Bidirectional search

Nội dung

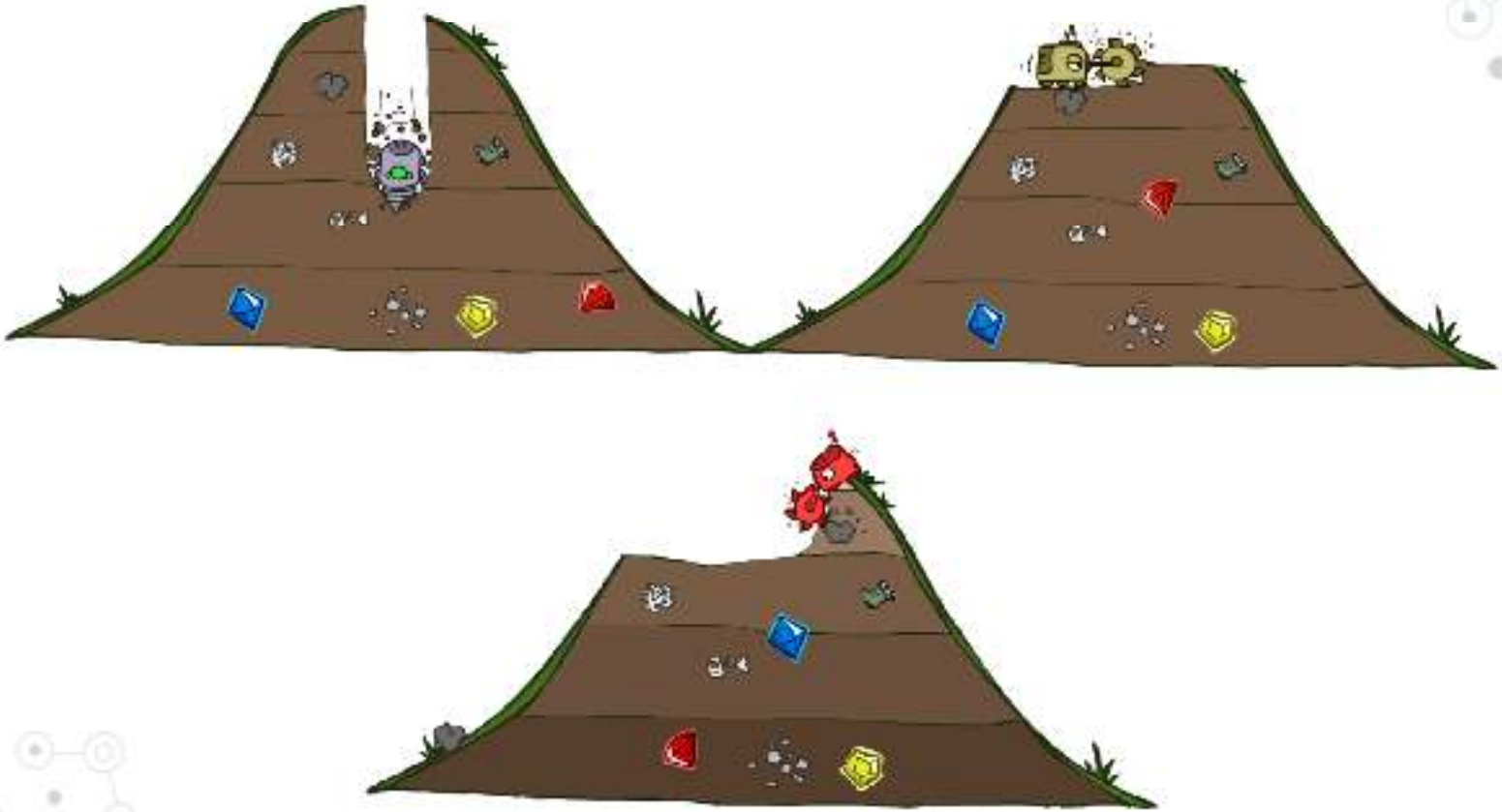
- Hệ thống lên kế hoạch (planning agent)
- Bài toán tìm kiếm (bài toán lên kế hoạch)
- Giải quyết bài toán bằng tìm kiếm mù
  - Thuật toán Depth-First Search
  - Thuật toán Breadth-First Search
  - Thuật toán Uniform Cost Search
  - Một số thuật toán khác
- **Đánh giá các thuật toán**

Phương pháp nào là DFS, BFS, UCS? (1)

Phương pháp nào là DFS, BFS, UCS? (2)

Phương pháp nào là DFS, BFS, UCS? (3)

So sánh DFS, BFS, DFS

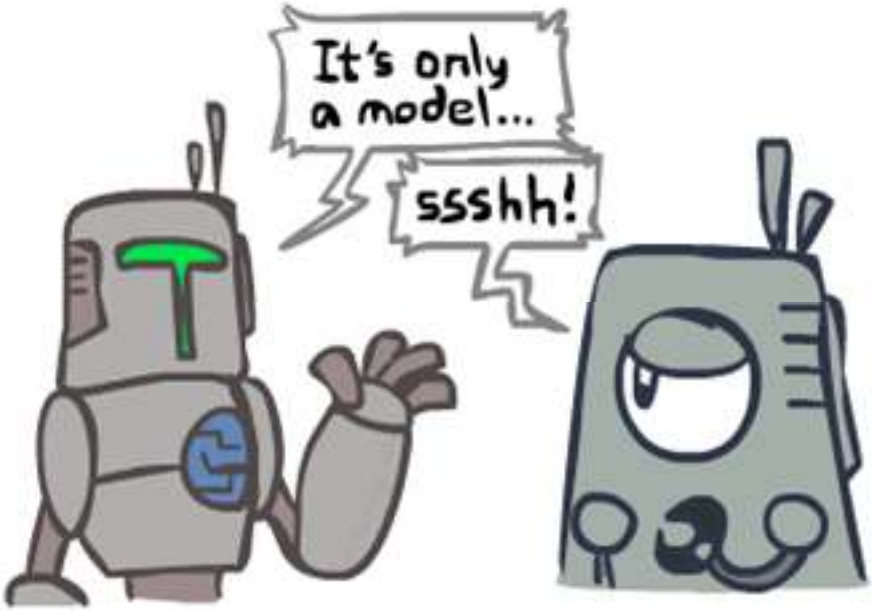


So sánh DFS, BFS, DFS

Criterion	Breadth-First	Uniform-Cost	Depth-First
Complete?	Yes[1]	Yes[1,2]	No
Optimal cost?	Yes[3]	Yes	No
Time	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(b^m)$
Space	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(bm)$

Tìm kiếm và mô hình

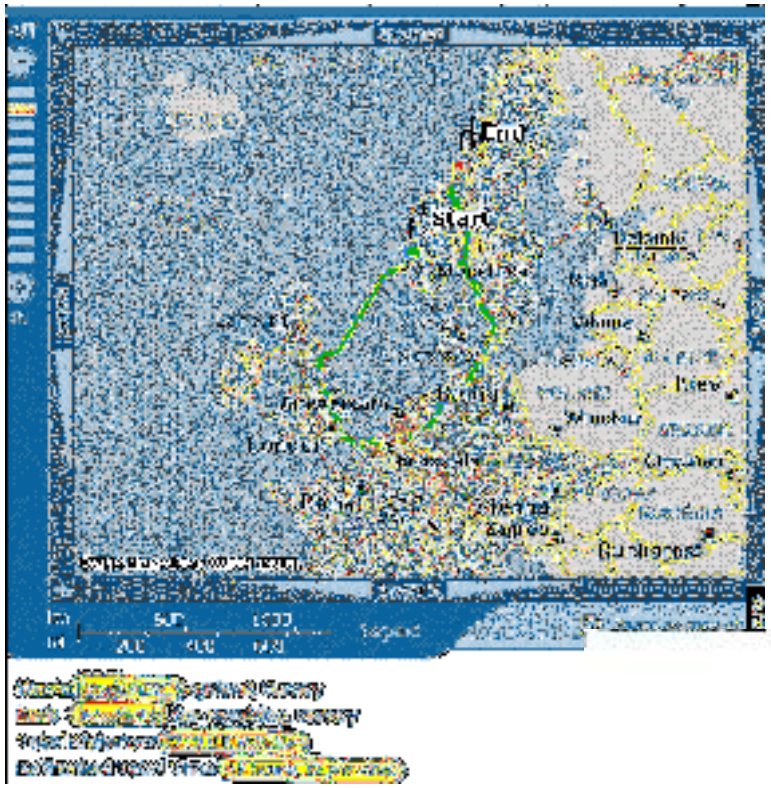
- Việc tìm kiếm chỉ thực hiện trên mô hình
  - Nó có thể tốt/phù hợp trên môn hình nhưng có thể không trên thực tế.
  - Nguyên nhân?



Ví dụ


MAPQUEST
Brier Ave
Silver Lake Blvd
McCready Ave
Hidalgo Ave
Silver Ridge Ave
Lake View Ave
Tevlot St
Bancroft Ave
Earl St
Loma Vista Pl
END
2
Cove Ave
Moore St
Oak Glen
Apex Ave
Glendale Blvd
Baxter St
Ewing St
2005 MapQuest.com, Inc.

Ví dụ



Thank you for listening

Tài liệu tham khảo

- Russell, Stuart, and Peter Norvig. "Artificial intelligence: a modern approach.", 2020.
- Lê Hoài Bắc, Tô Hoài Việt. Cơ sở Trí tuệ nhân tạo. 2014
- AI Course, Trần Trung Kiên, Nguyễn Ngọc Thảo
- UC Berkely, CS188 Intro to AI