Tìm kiếm

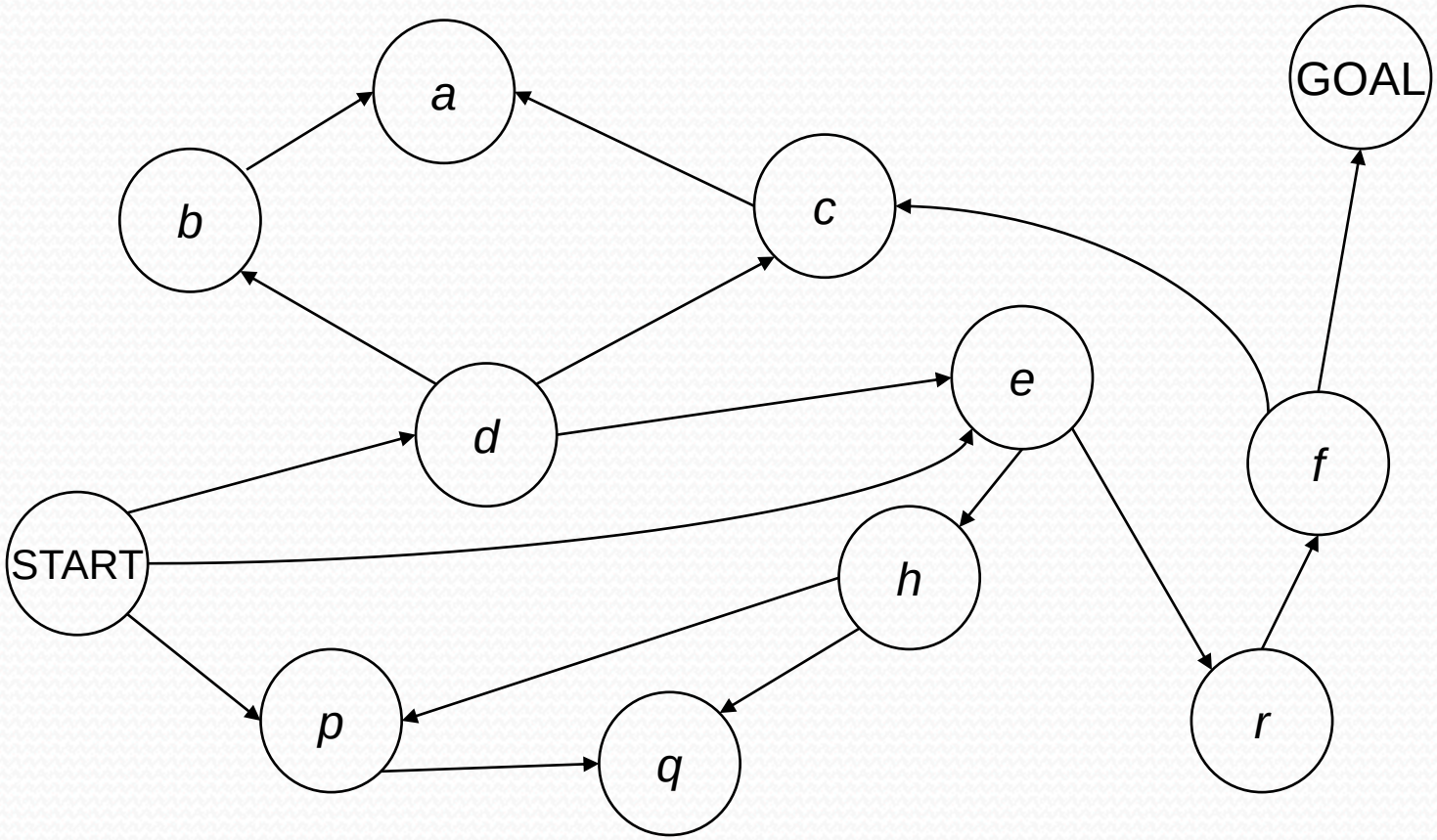
Tô Hoài Việt
Khoa Công nghệ Thông tin
Đại học Khoa học Tự nhiên TPHCM
thviet@fit.hcmuns.edu.vn

Ref: http://www.cs.cmu.edu/~awm/tutorials

Tổng quát

- Bài toán tìm kiếm
- Tìm kiếm Theo chiều Rộng
- Tính tối ưu, Tính đầy đủ, Độ phức tạp thời gian và không gian
- Cây Tìm kiếm
- Tìm kiếm Theo chiều Sâu

Một bài toán Tìm kiếm



Làm sao để đi từ S đến G? Và số biến đổi có thể ít nhất là gì?

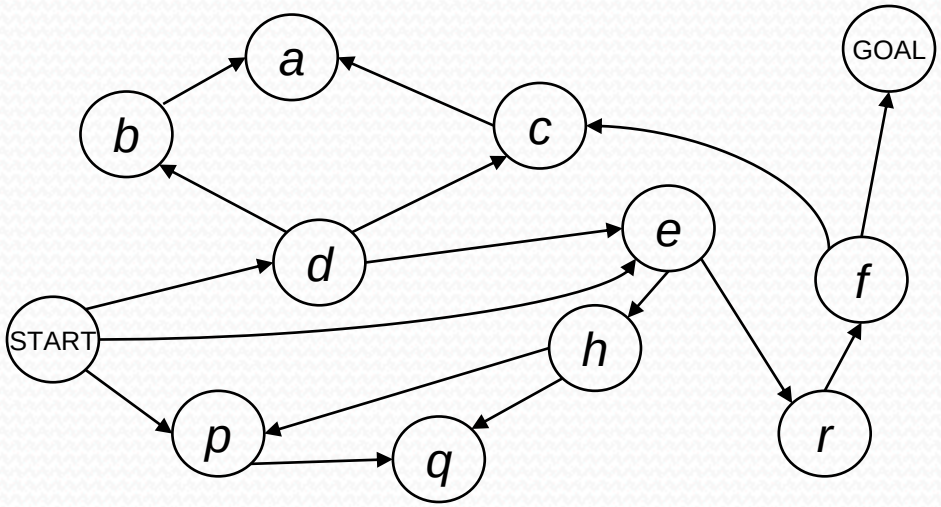
Hình thức hoá một bài toán tìm kiếm

Một bài toán tìm kiếm có năm thành phần:

Q , *S* , *G* , **succs** , **cost**

- *Q* là một tập hữu hạn các trạng thái.
- $S \subseteq Q$ một tập khác rỗng các trạng thái ban đầu.
- $G \subseteq Q$ một tập khác rỗng các trạng thái đích.
- **succs** : *Q* → *P(Q)* là một hàm nhận một trạng thái làm đầu vào và trả về kết quả là một tập trạng thái. **succs**(*s*) nghĩa là "tập các trạng thái có thể đến từ s trong một bước".
- **cost** : *Q* , *Q* → *Số Dương* là một hàm nhận hai trạng thái, *s* và *s'*, làm đầu vào. Nó trả về chi phí một bước của việc di chuyển từ *s* đến *s'*. Hàm chi phí chỉ được định nghĩa khi *s'* là trạng thái con của *s*.

Bài toán Tìm kiếm



Q = {START, *a* , *b* , *c* , *d* , *e* , *f* , *h* , *p* , *q* , *r* , GOAL}

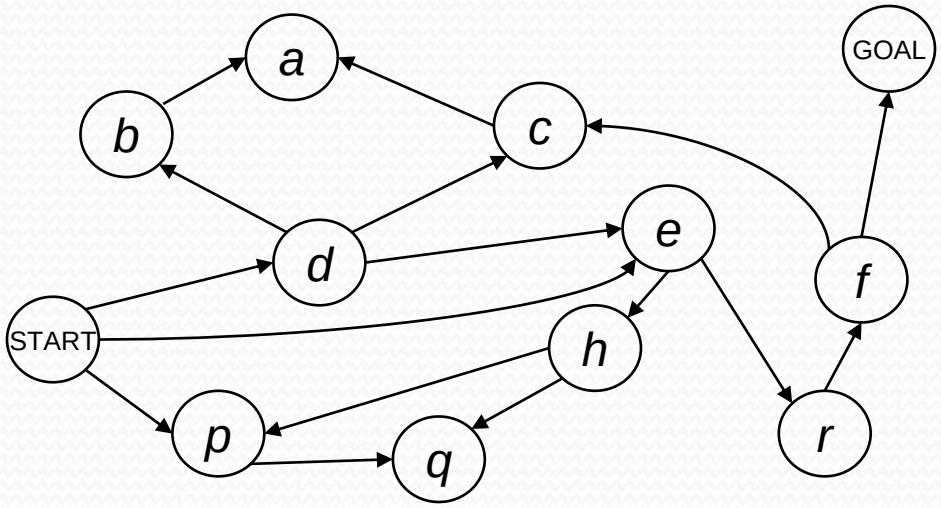
S = { START }

G = { GOAL }

succs(*b*) = { *a* }
succs(*e*) = { *h* , *r* }
succs(*a*) = NULL … etc.

cost(*s*,*s'*) = 1 cho tất cả các biến đổi

Bài toán Tìm kiếm



Q = {START, *a* , *b* , *c* , *d* , *e* , *f* , *h* , *p* , *q* , *r* , GOAL}

S = { START }

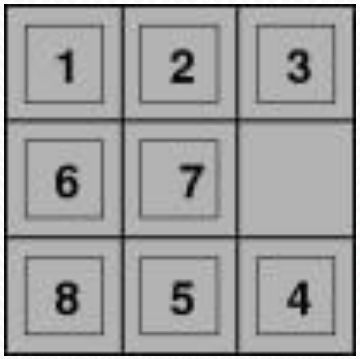
G = { GOAL }

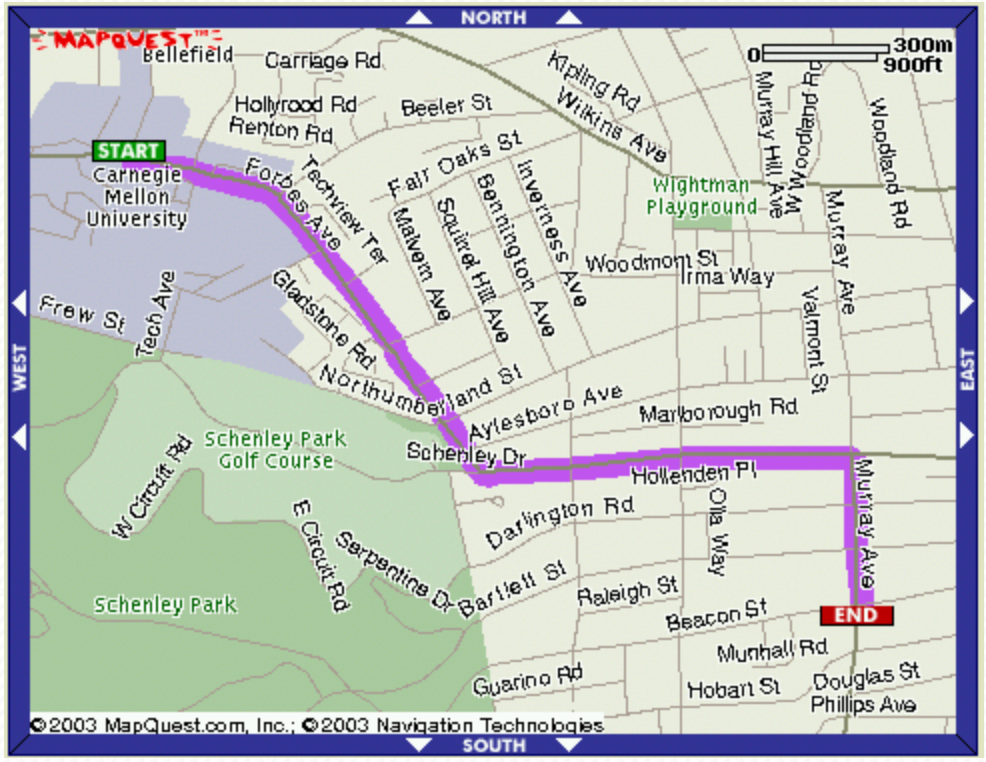
succs(*b*) = { *a* }
succs(*e*) = { *h* , *r* }
succs(*a*) = NULL … etc.

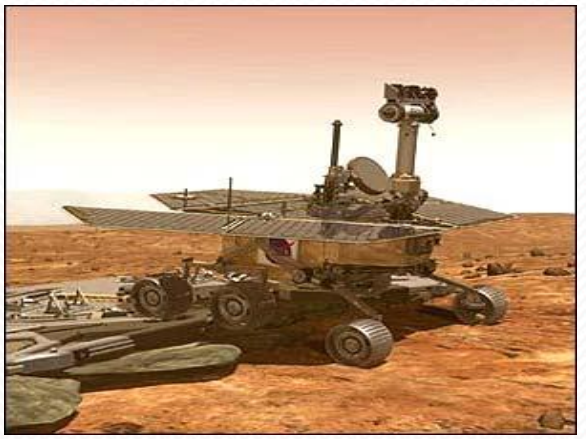
cost(*s*,*s'*) = 1 cho tất cả các biến đổi

Tại sao ta quan tâm? Bài toán nào giống như vậy?

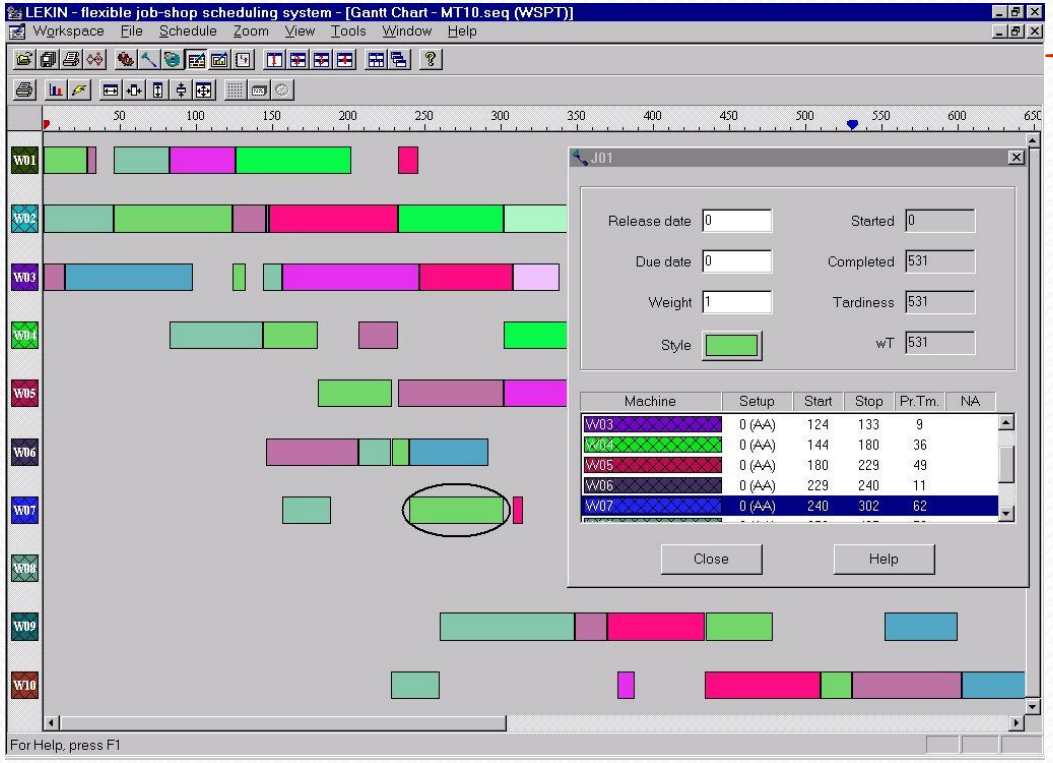
Các Bài toán Tìm kiếm







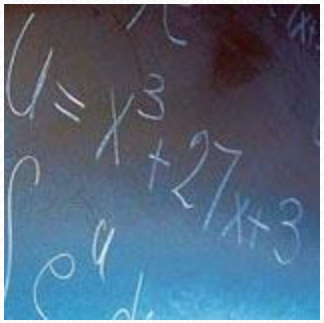
Các Bài toán Tìm kiếm



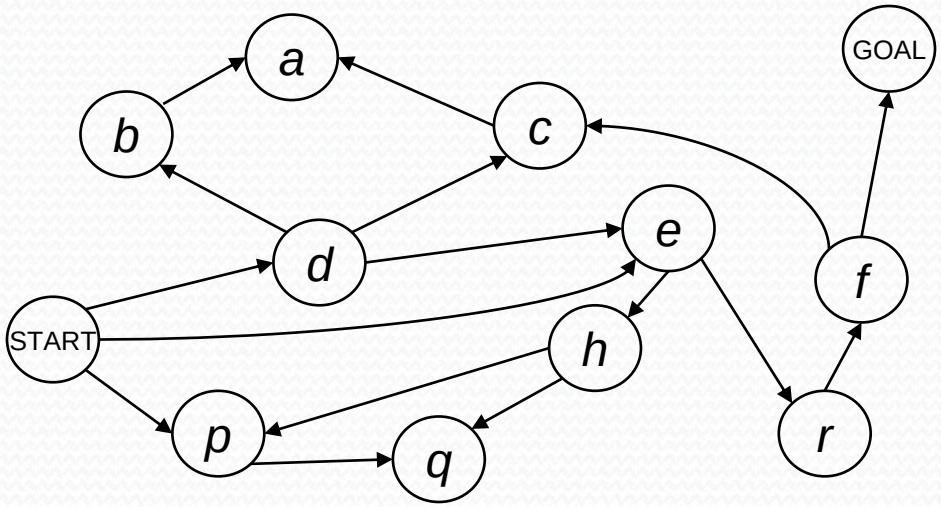
Lập lịch

8-Hậu

Giải toán



Tìm kiếm Theo Chiều Rộng



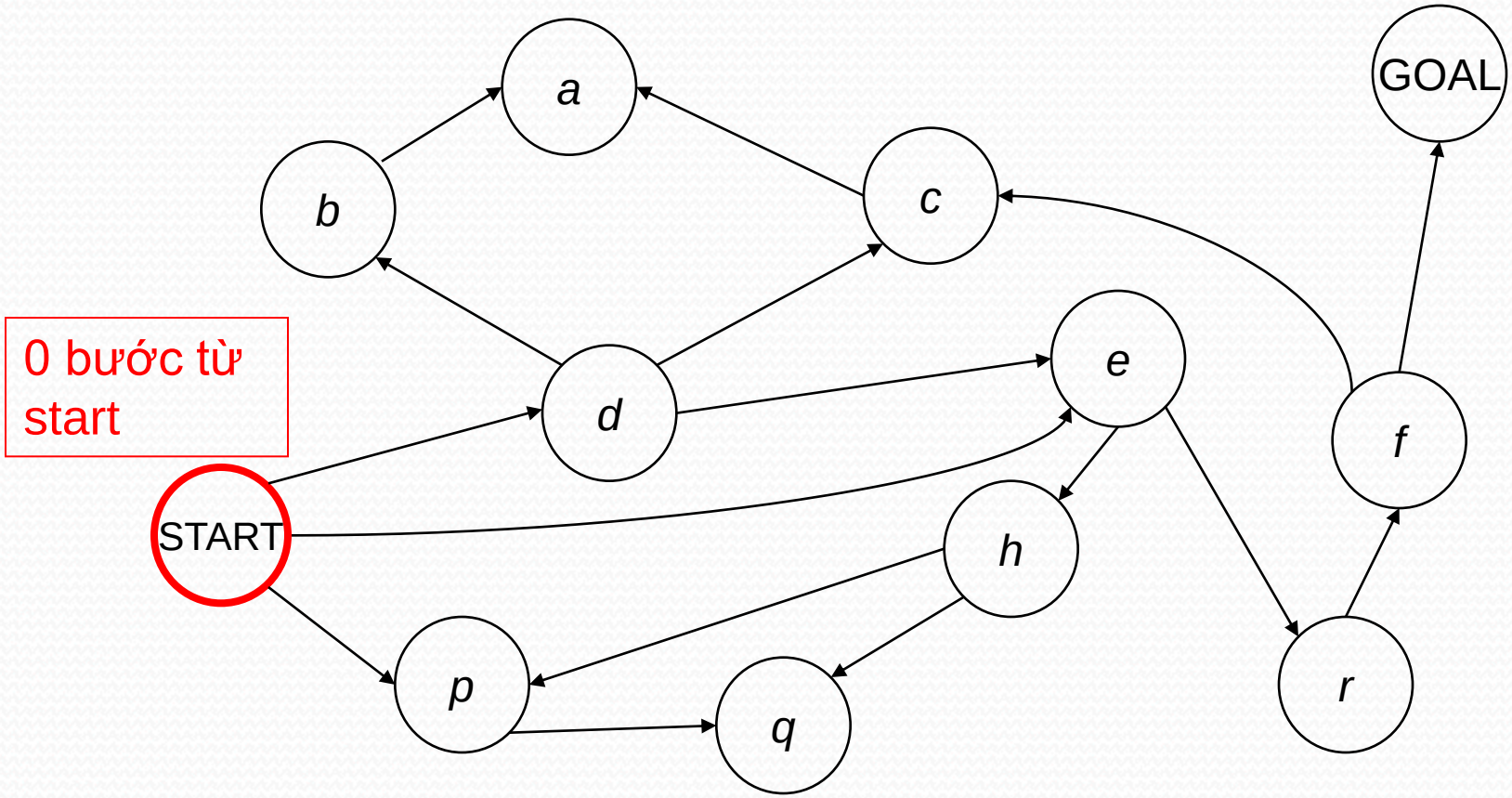
Gán nhãn tất cả trạng thái có thể đi đến được từ S trong 1 bước nhưng không thể đi đến được trong ít hơn 1 bước.

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.

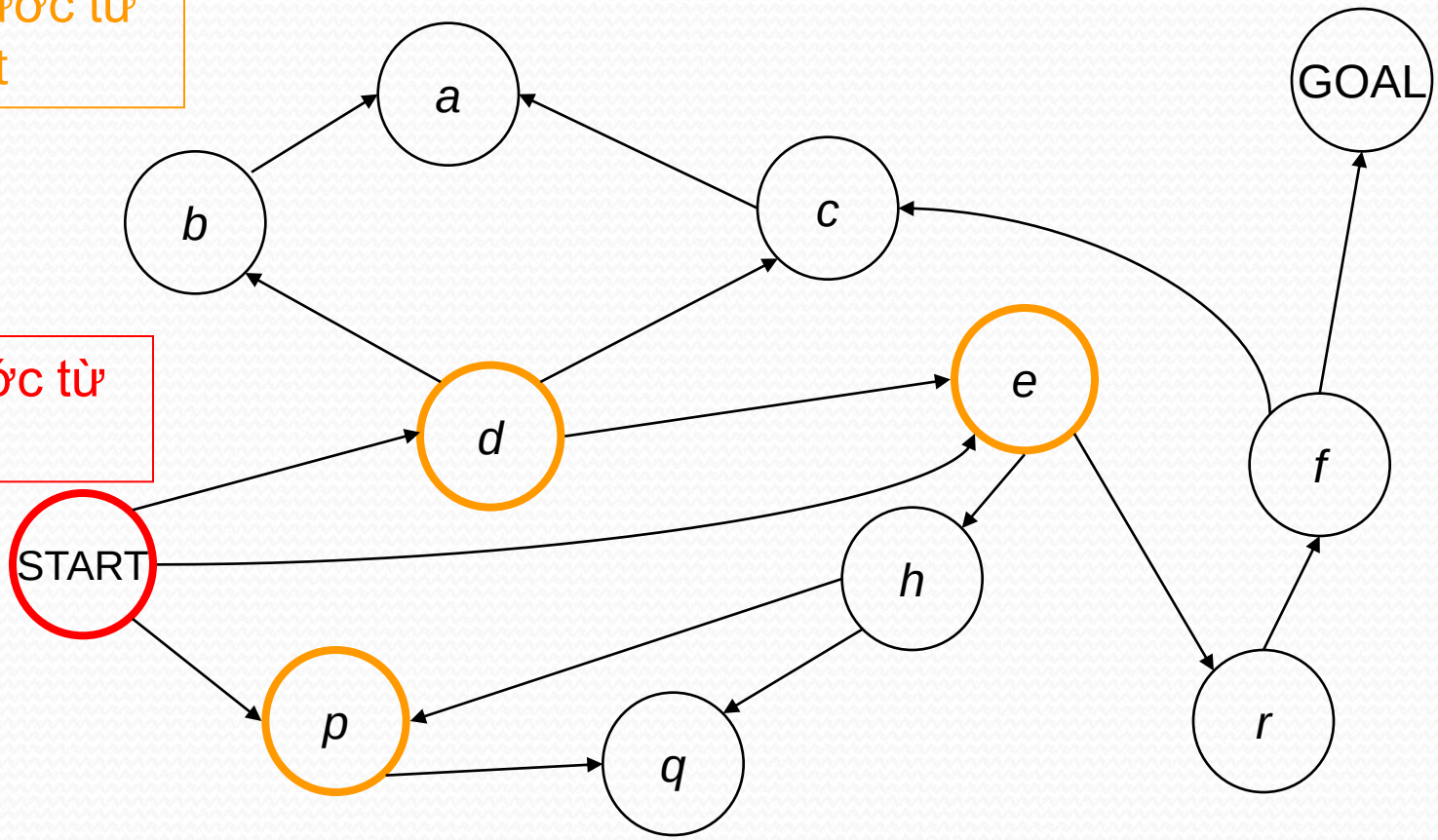
Tìm kiếm Theo Chiều Rộng



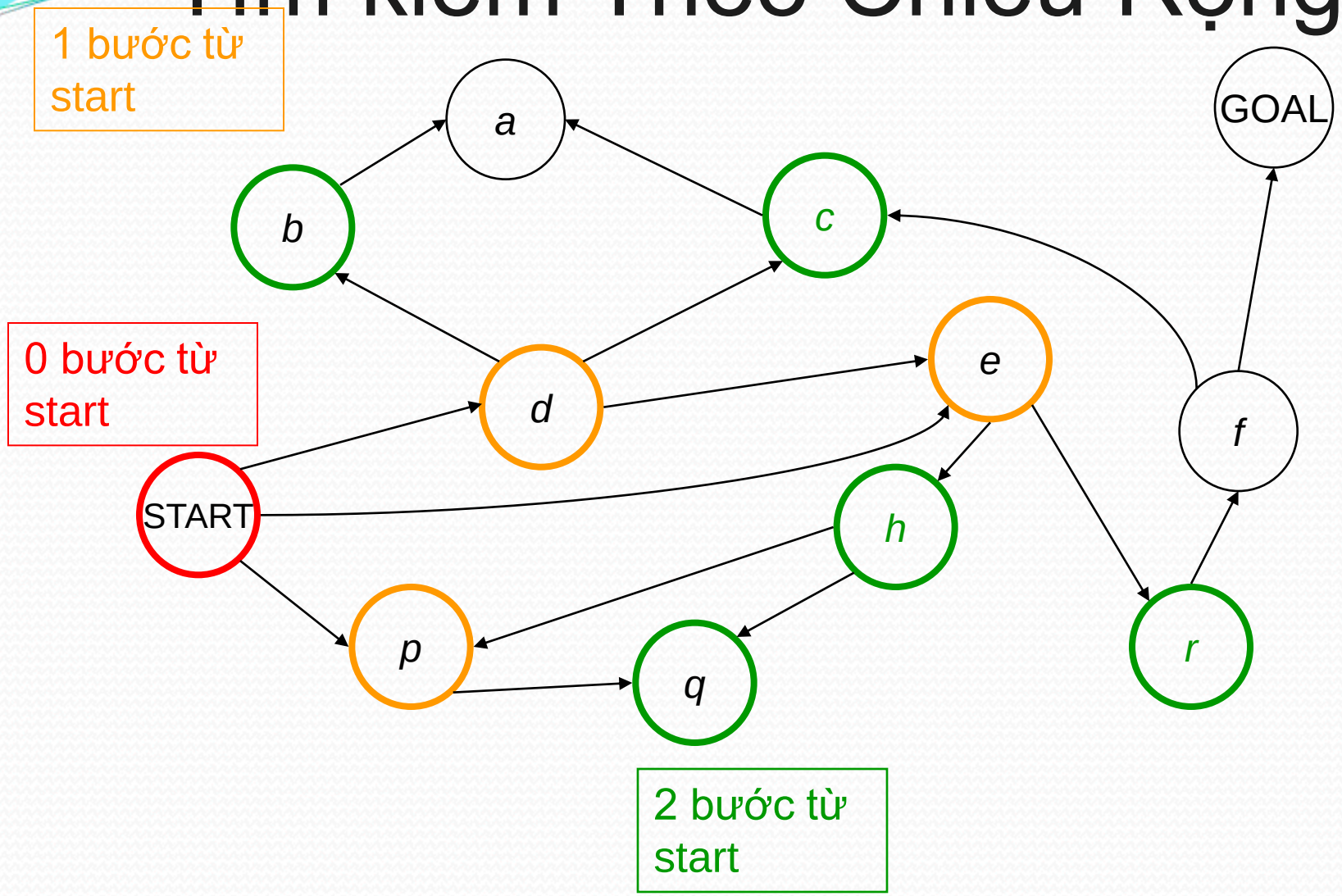
Tìm kiếm Theo Chiều Rộng

1 bước từ start

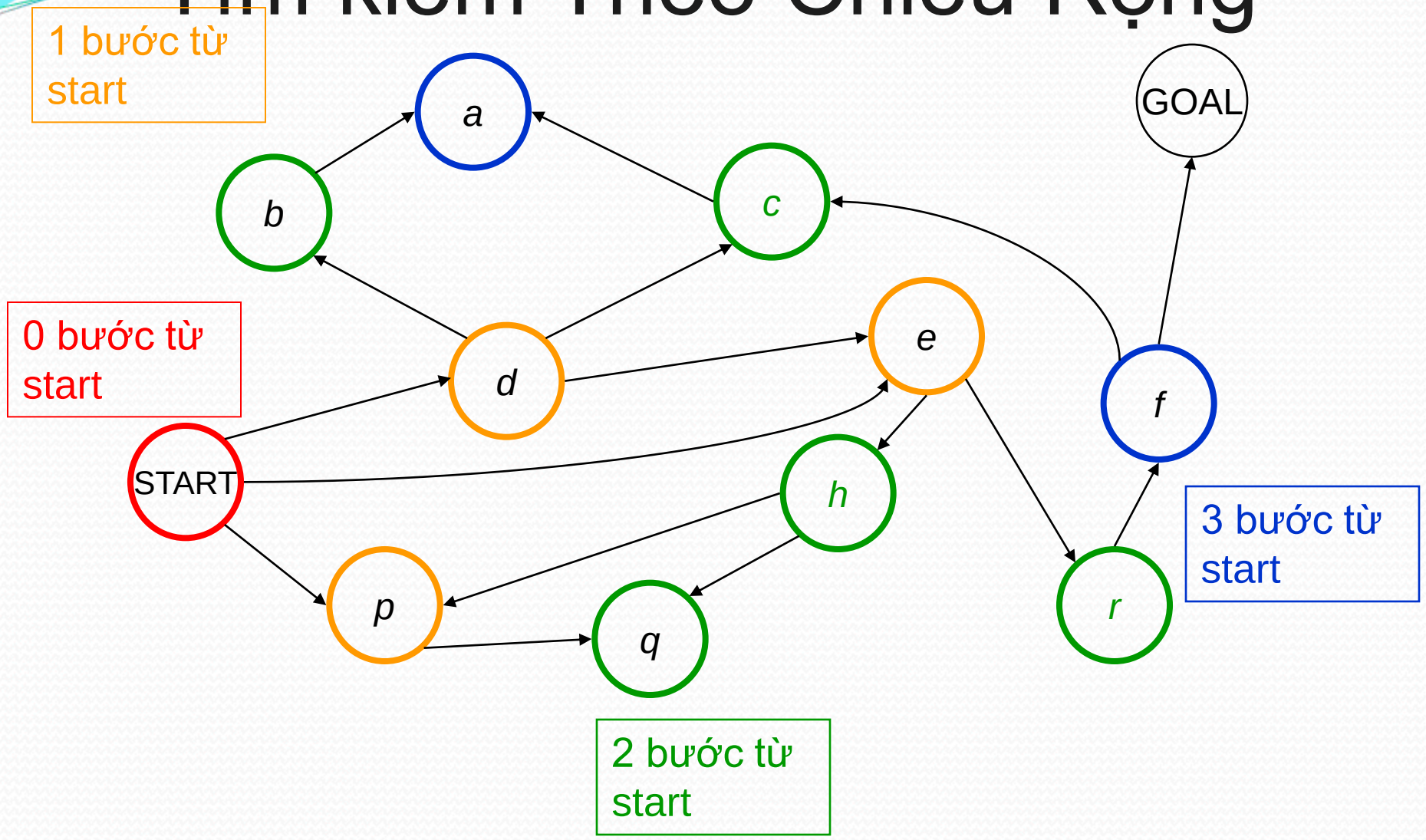
0 bước từ start



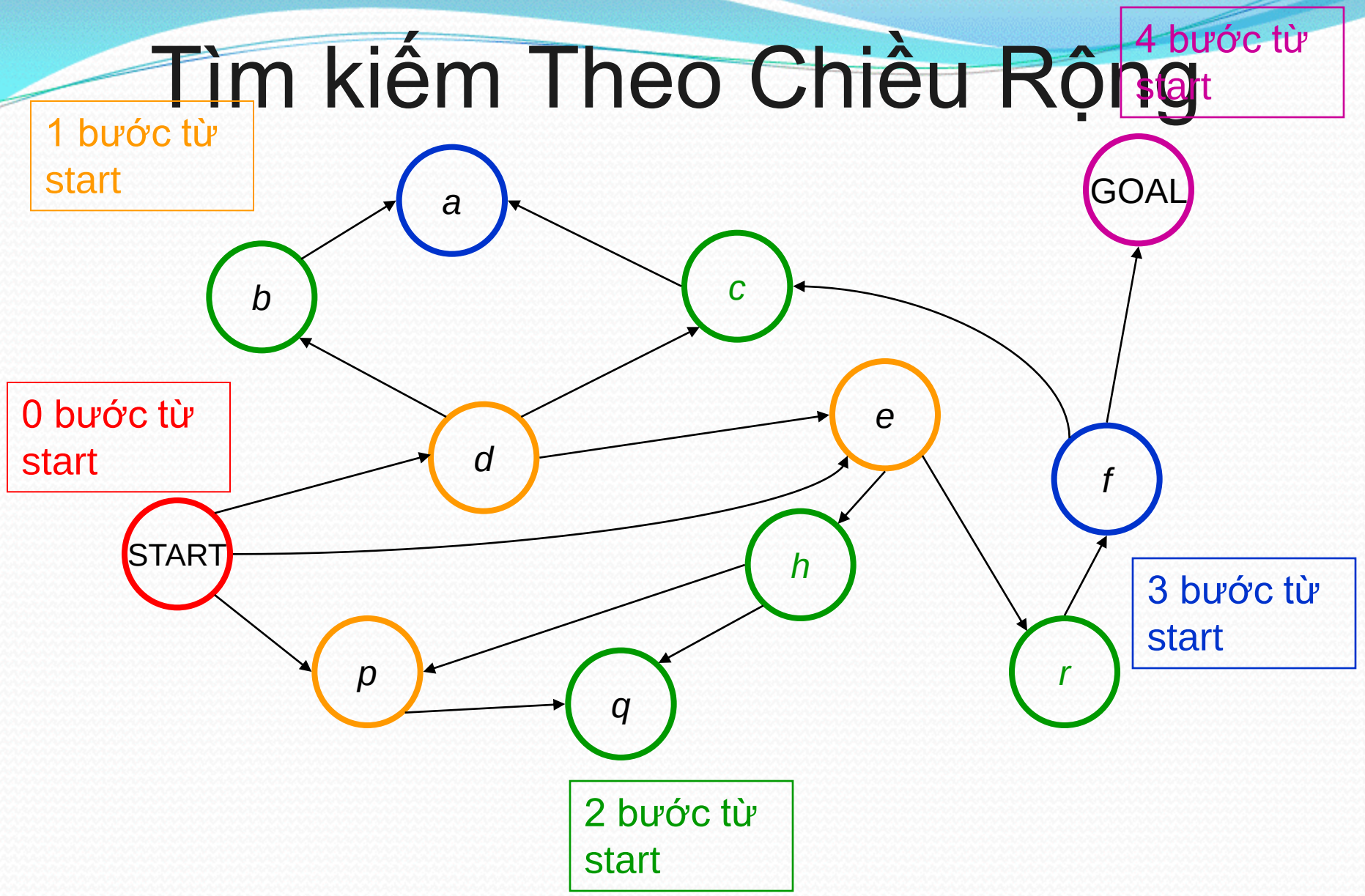
Tìm kiếm Theo Chiều Rộng



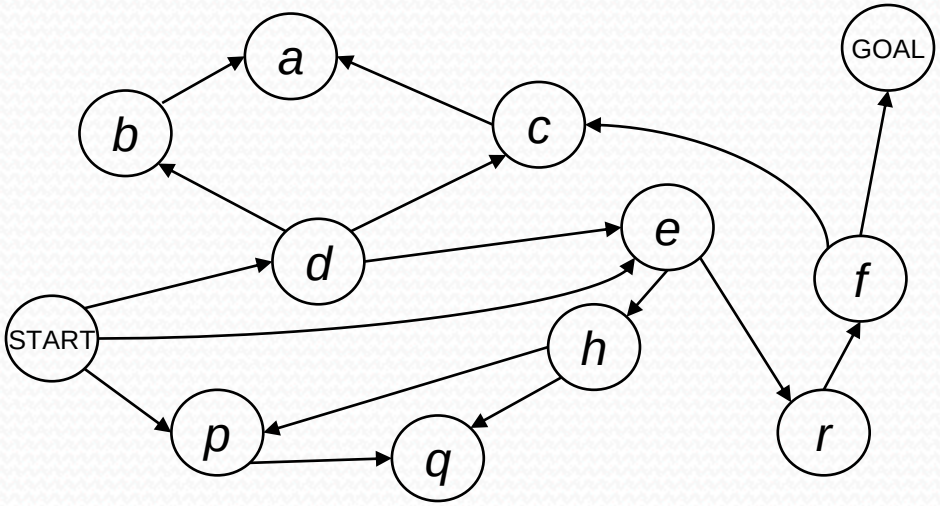
Tìm kiếm Theo Chiều Rộng



Tìm kiếm Theo Chiều Rộng



Ghi nhớ đường đi!

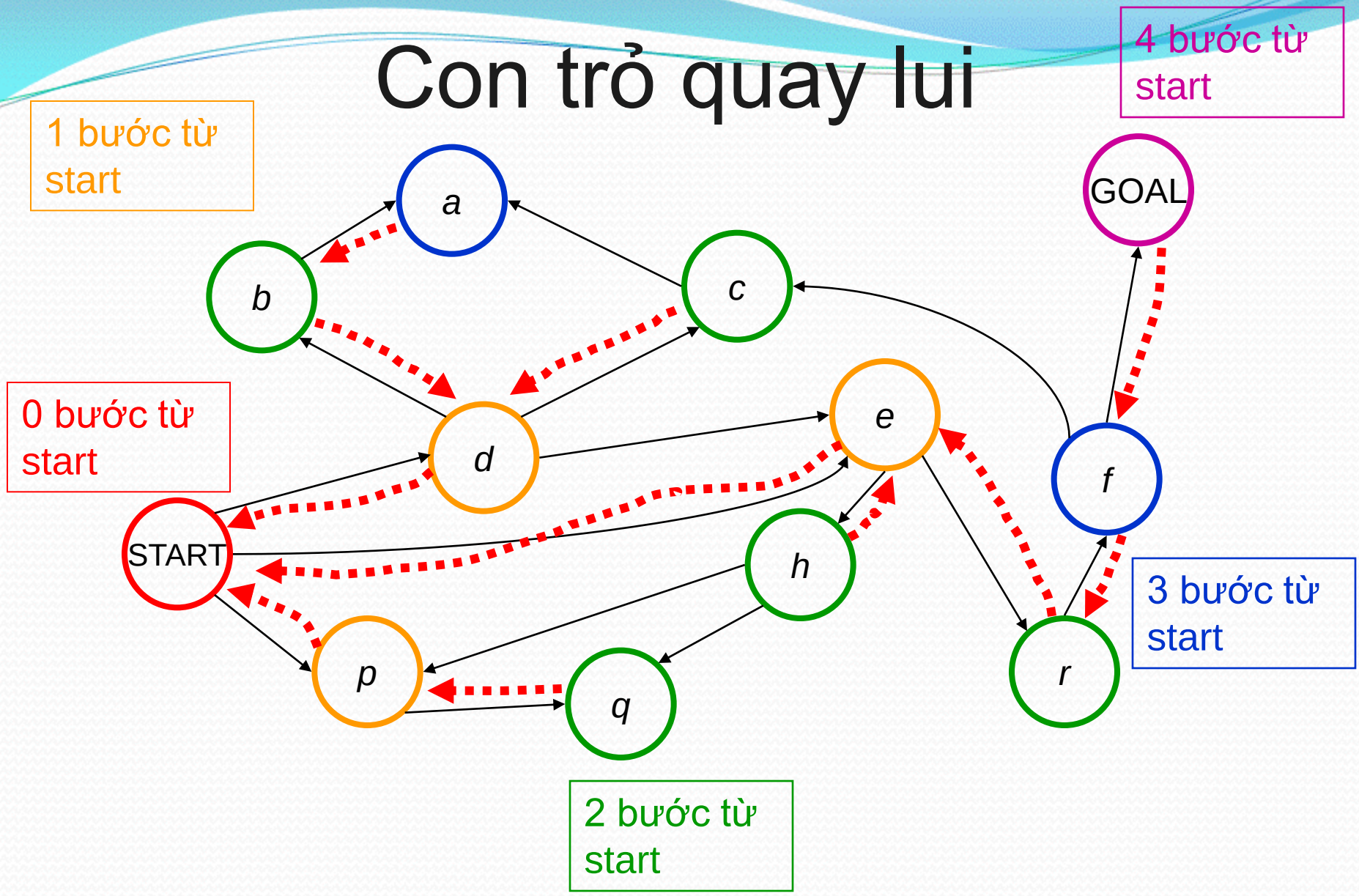


Ngoài ra, khi gán nhãn một trạng thái, ghi nhận trạng thái trước đó. Ghi nhận này được gọi là *con trỏ quay lui*. Lịch sử trước đó được dùng để phát sinh con đường lời giải, khi đã tìm được đích:

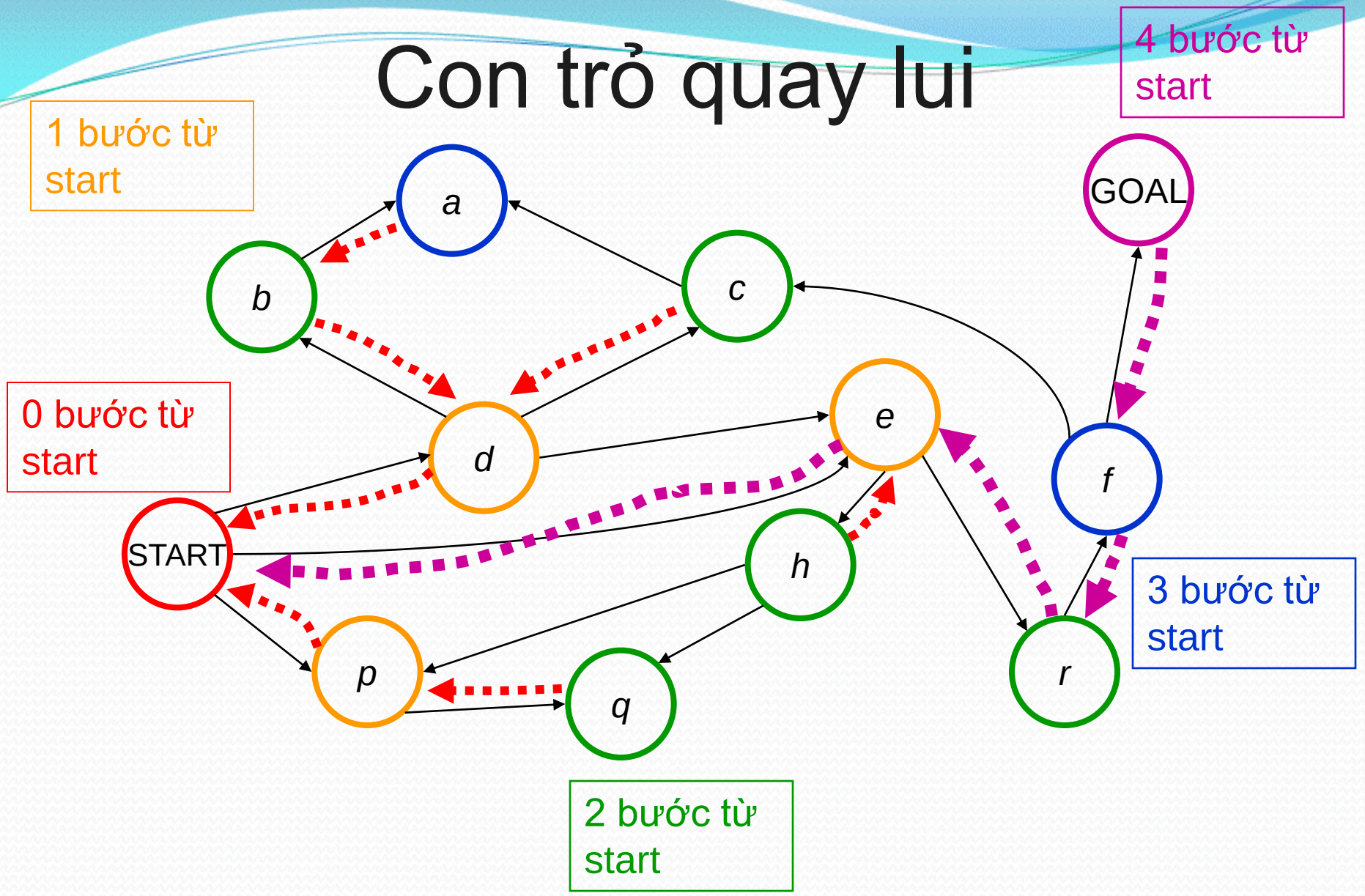
"Tôi đã đến đích. Tôi thấy mình đã ở f trước đó. Và tôi đã ở *r* trước khi tới *f*. Và…

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

Con trỏ quay lui



Con trỏ quay lui



Bắt đầu Tìm kiếm Theo chiều Rộng

Với bất kỳ trạng thái *s* nào đã gán nhãn, ghi nhớ:

•*previous*(*s*) là trạng thái trước đó trên đường đi ngắn nhất từ trạng thái START đến *s*.

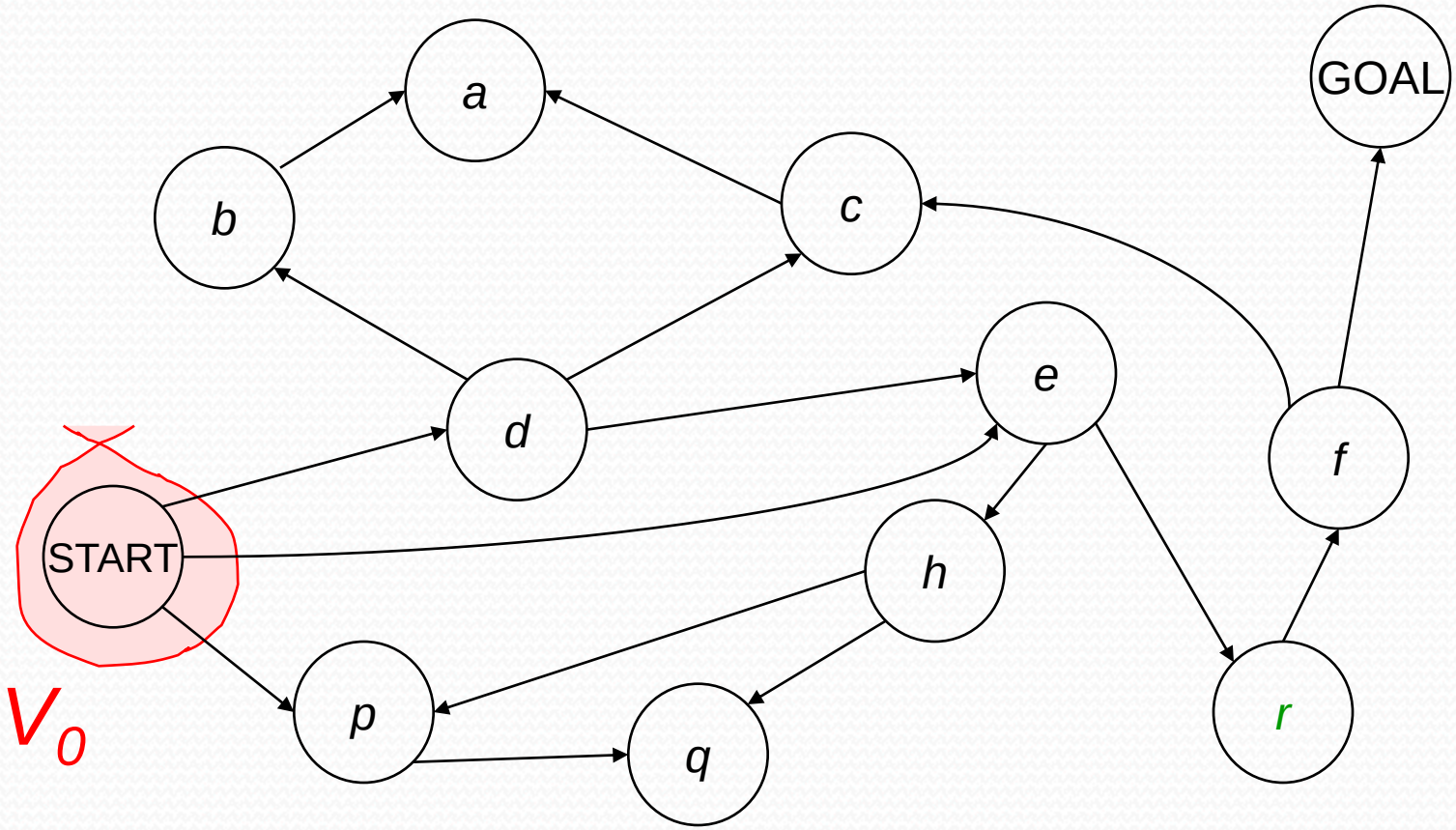
Trong vòng lặp thứ *k* của thuật toán ta bắt đầu với V_k được định nghĩa là tập các trạng thái mà từ trạng thái start đi đến có đúng *k bước*

Sau đó, trong suốt vòng lặp, ta sẽ tính V_{k+1}, được định nghĩa là tập các trạng thái mà từ trạng thái start đi đến có đúng *k+1 bước*

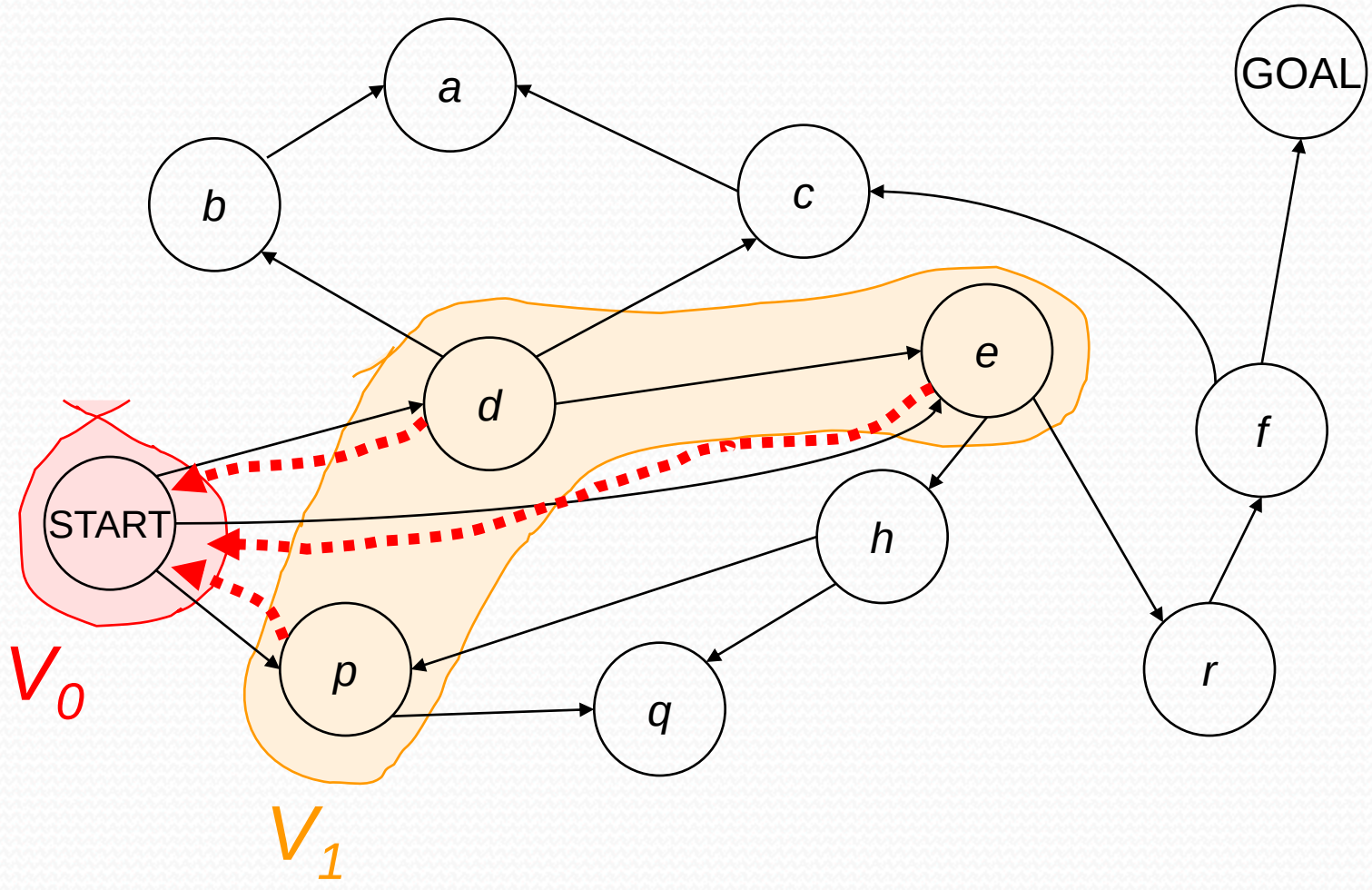
Chúng ta bắt đầu với $k = 0$, V_0 = {START} và định nghĩa, *previous*(*START*) = *NULL*

Sau đó ta sẽ thêm vào những trạng thái một bước từ START vào V_1. Và tiếp tục.

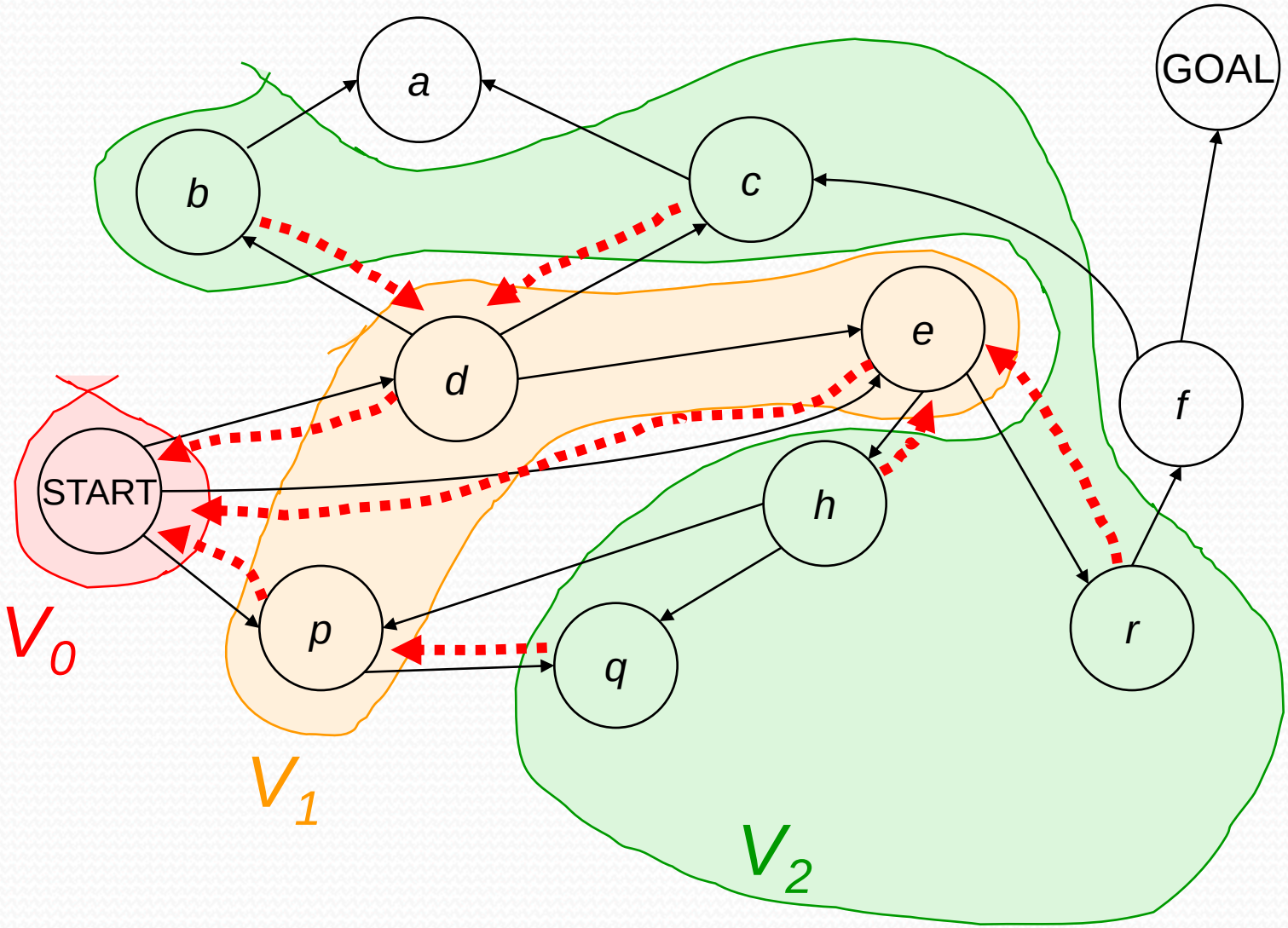
BFS



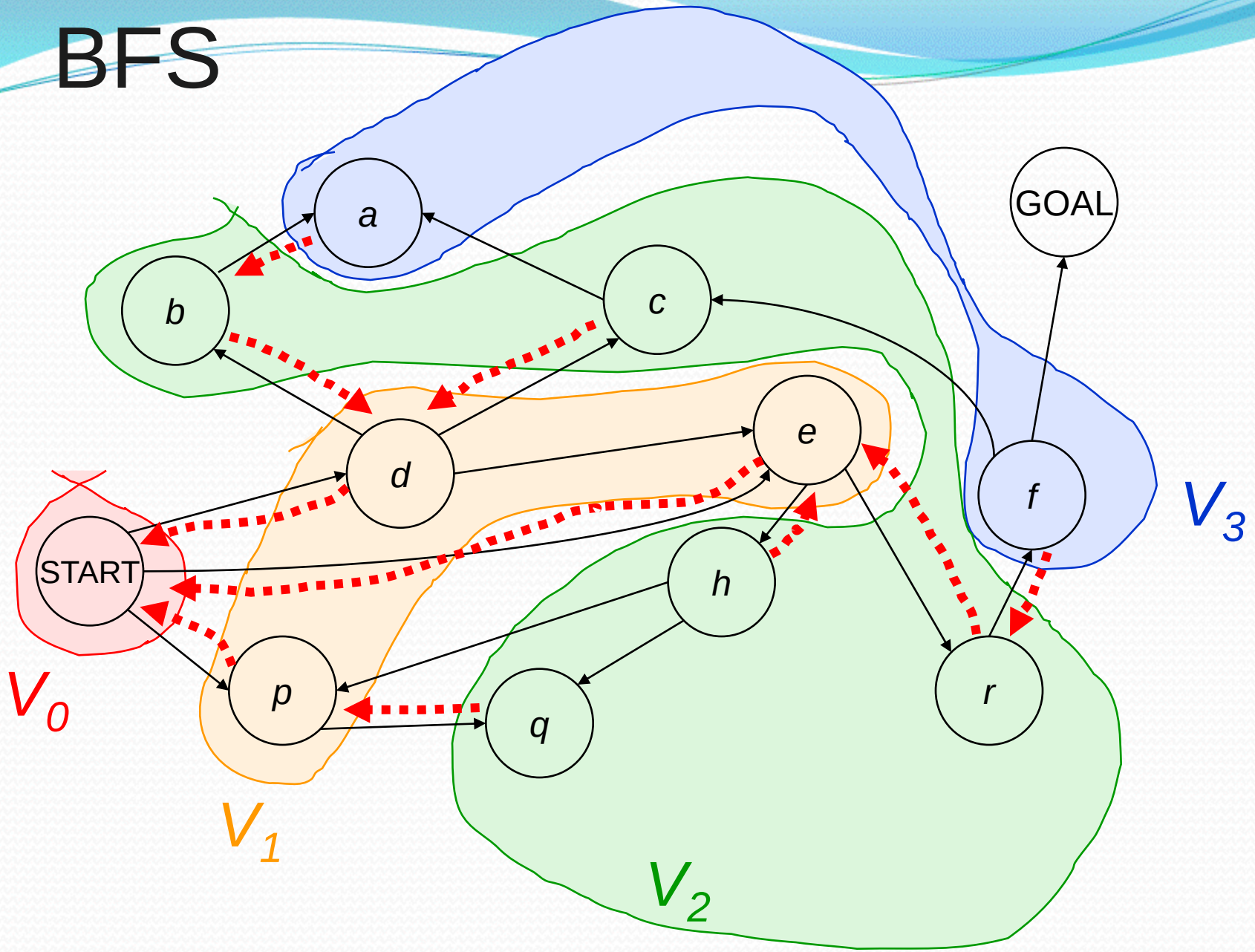
BFS



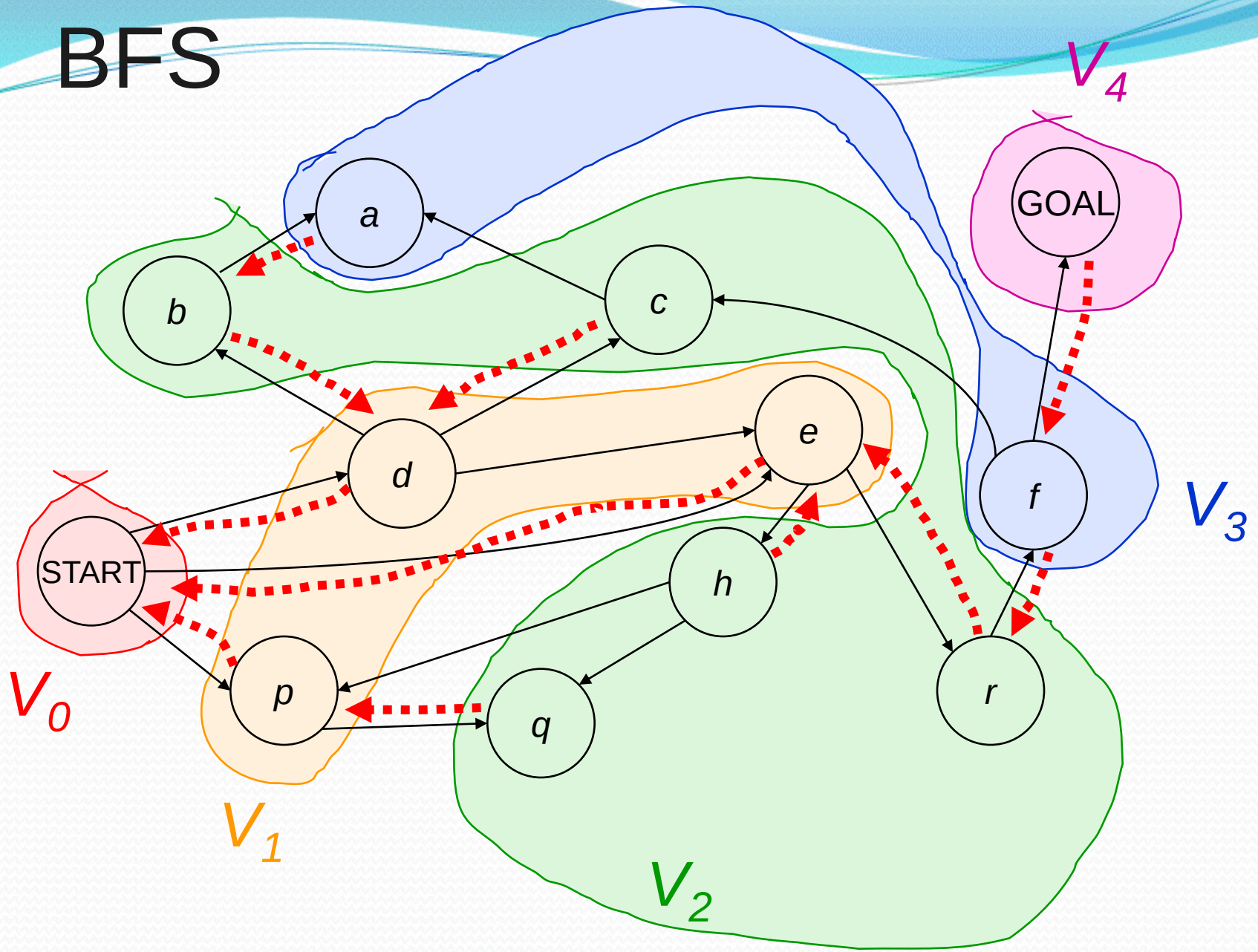
BFS



BFS



BFS



Tìm kiếm Theo Chiều Rộng

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*

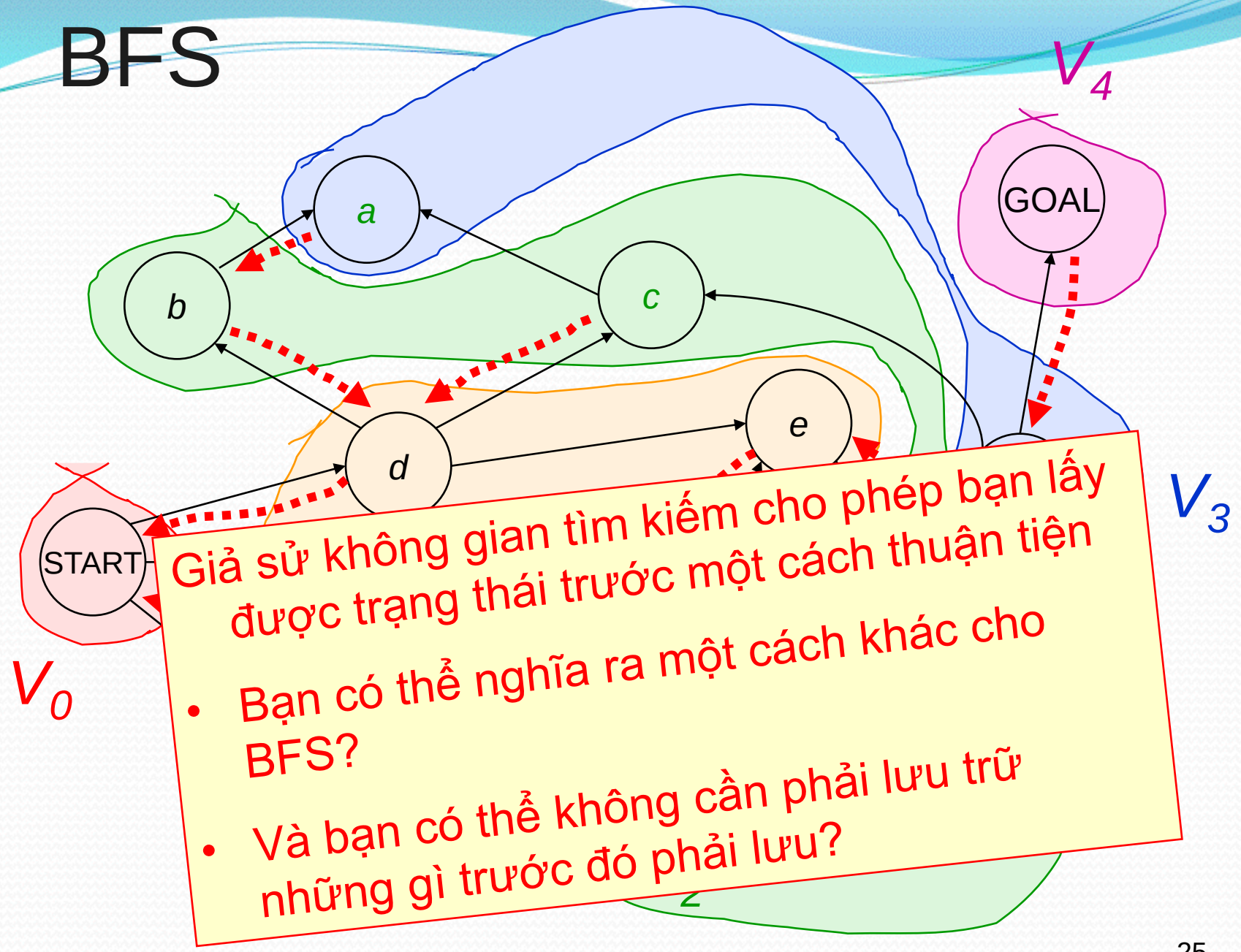
Thêm *s'* vào V_{k+1}

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).

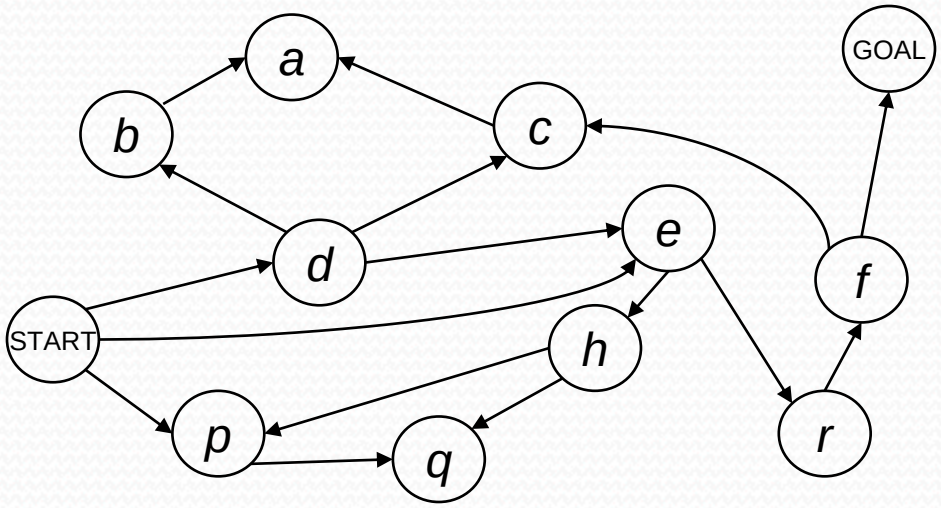
BFS



Giả sử không gian tìm kiếm cho phép bạn lấy được trạng thái trước một cách thuận tiện

- Bạn có thể nghĩa ra một cách khác cho BFS?
- Và bạn có thể không cần phải lưu trữ những gì trước đó phải lưu?

Một cách khác: Đi lui



Gán nhãn tất cả các trạng thái có thể đến G trong 1 nhưng không thể đi đến nó trong ít hơn 1 bước.

Gán nhãn tất cả các trạng thái có thể đến G trong 2 nhưng không thể đi đến nó trong ít hơn 2 bước.

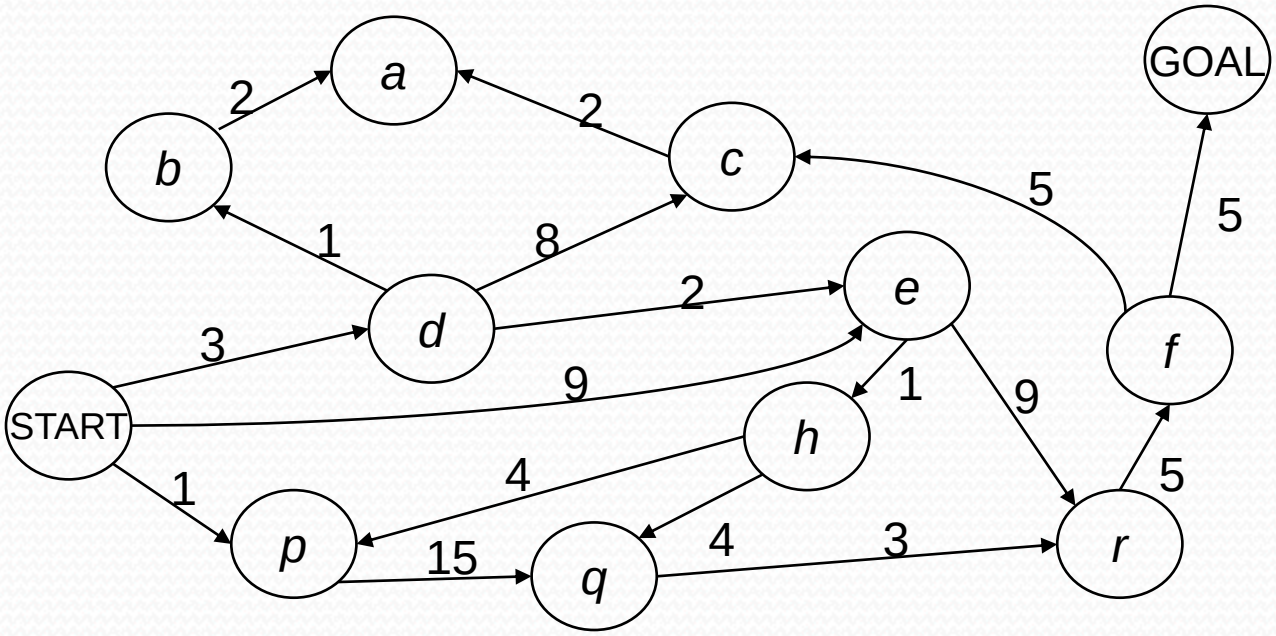
V.v. … cho đến khi đến start.

Nhãn "số bước tới đích" xác định đường đi ngắn nhất. Không cần thêm thông tin lưu trữ.

Các chi tiết của Theo Chiều Rộng

- Vẫn tốt nếu có nhiều hơn một trạng thái đích.
- Vẫn tốt nếu có nhiều hơn một trạng thái đầu.
- Thuật toán này hoạt động theo kiểu tiến từ đầu. Thuật toán nào hoạt động theo kiểu tiến từ đầu được gọi là *suy diễn tiến*.
- Bạn cũng có thể hoạt động quay lui từ đích.
- Thuật toán này rất giống thuật toán Dijkstra.
- Bất kỳ thuật toán nào hoạt động theo kiểu quay lui từ đích được gọi là *suy diễn lùi*.
- Lùi so với tiến. Cái nào tốt hơn?

Chi phí chuyển đổi



Lưu ý rằng BFS tìm đường đi ngắn nhất theo số biến đổi. Nó không tìm thấy đường đi có chi phí ít nhất.

Bây giờ chúng ta xem xét một thuật toán tìm đường đi chi phí thấp nhất. Trong vòng lặp thứ *k*, với bất kỳ trạng thái *S* nào, đặt *g*(*s*) là chi phí đường đi có chi phí nhỏ nhất đến *S* trong *k* bước hay ít hơn.

Theo Chiều Rộng Chi phí Thấp nhất

V_k = tập các trạng thái có thể đến được trong đúng *k* bước, và với nó đường đi k-bước chi phí thấp nhất thì ít chi phí hơn bất kỳ đường đi nào có độ dài nhỏ hơn *k*. Nói cách khác, V_k = tập trạng thái mà giá trị của nó thay đổi so với vòng lặp trước.

V_0 := *S* (tập trạng thái đầu)

previous(*START*) := *NIL*

g(*START*) = *0*

k := *0*

while (V_k khác rỗng) **do**

V_{k+1} := rỗng

Với mỗi *s* trong V_k

Với mỗi *s'* trong **succs**(*s*)

Nếu *s'* chưa được gán nhãn

HAY nếu *g*(*s*) + *Cost*(*s*,*s'*) < *g*(*s'*)

Đặt *previous*(*s'*) := *s*

Đặt *g*(*s'*) := *g*(*s*) + *Cost*(*s*,*s'*)

Thêm *s'* vào V_{k+1}

k := *k+1*

Nếu GOAL chưa gán nhãn, thoát FAILURE

Nglại xây dựng lời giải theo: Đặt S_k là trạng thái thứ *k* trên đường đi ngắn nhất. Định nghĩa S_k = GOAL, và với mọi *i* <= *k*, định nghĩa S_{i-1} = *previous*(S_i).

Tìm kiếm Chi phí Đồng nhất

- Một cách tiếp cận BFS đơn giản về mặt khái niệm khi có chi phí chuyển đổi
- Dùng hàng đợi ưu tiên

Hàng đợi Ưu tiên

Một hàng đợi ưu tiên là một cấu trúc dữ liệu trong đó ta có thể thêm và lấy các cặp (thing, value) với các toán tử sau:

Init-PriQueue(PQ)	khởi tạo PQ rỗng.
Insert-PriQueue(PQ, thing, value)	thêm *(thing, value)* vào hàng đợi.
Pop-least(PQ)	trả về cặp *(thing, value)* với giá trị thấp nhất, và loại bỏ nó khỏi hàng đợi.

Hàng đợi Ưu tiên

Một hàng đợi ưu tiên là một cấu trúc dữ liệu trong đó ta có thể thêm và lấy các cặp (thing, value) với các toán tử sau:

Init-PriQueue(PQ)	khởi tạo PQ rỗng.
Insert-PriQueue(PQ, thing, value)	thêm *(thing, value)* vào hàng đợi.
Pop-least(PQ)	trả về cặp *(thing, value)* với giá trị thấp nhất, và loại bỏ nó khỏi hàng đợi.

Hàng đợi Ưu tiên có thể được cài đặt theo một cách sao cho chi phí của các toán tử thêm và lấy là

O(*log*(*số mục trong hàng đợi ưu tiên*))

Rất rẻ (dù không tuyệt đối, nhưng rẻ không tin được!)

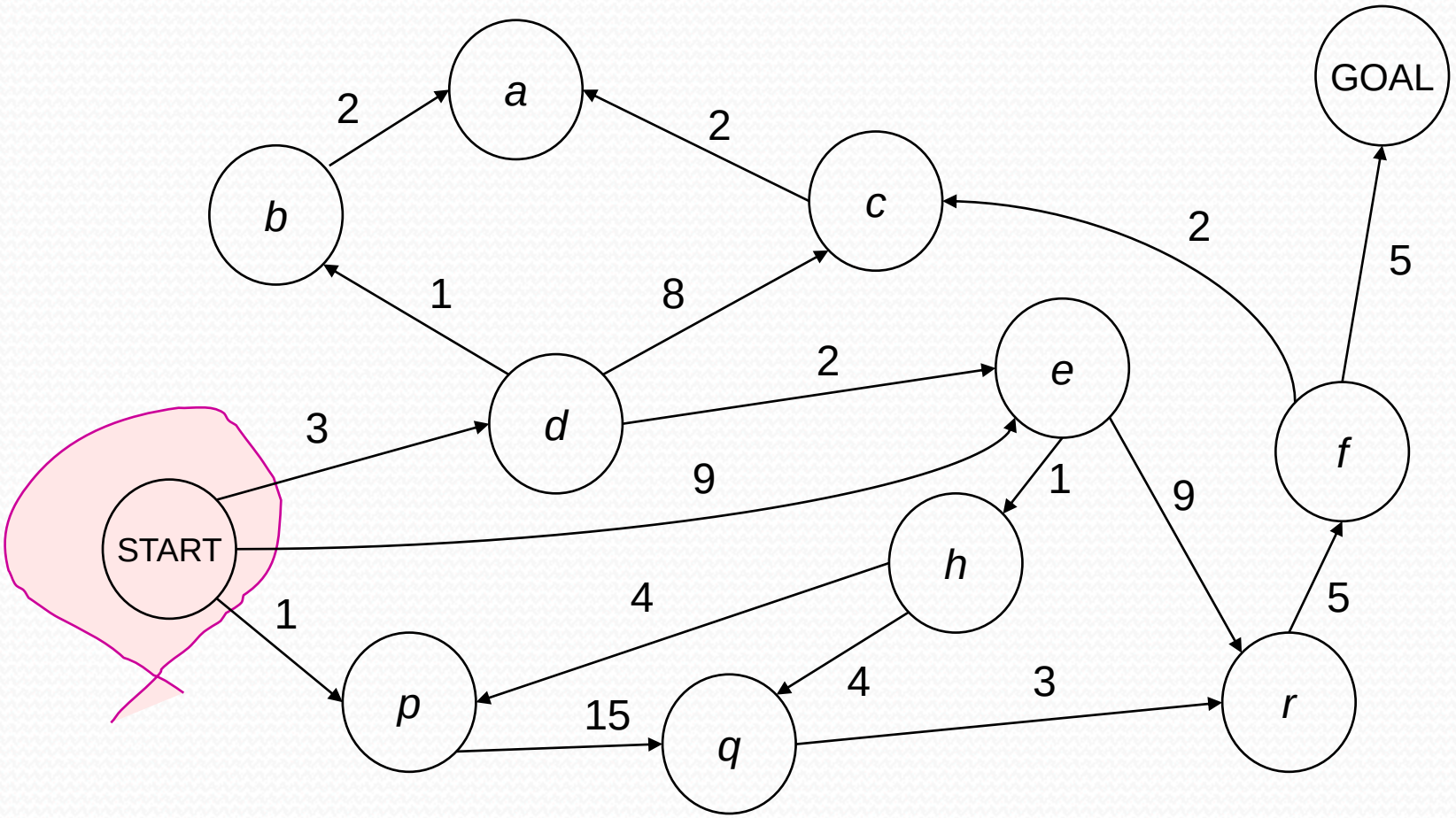
Tìm kiếm Chi phí Đồng nhất (UCS)

- Một cách tiếp cận BFS đơn giản về mặt khái niệm khi có chi phí chuyển đổi
- Dùng hàng đợi ưu tiên

  PQ = Tập trạng thái đã được mở hay đang đợi mở

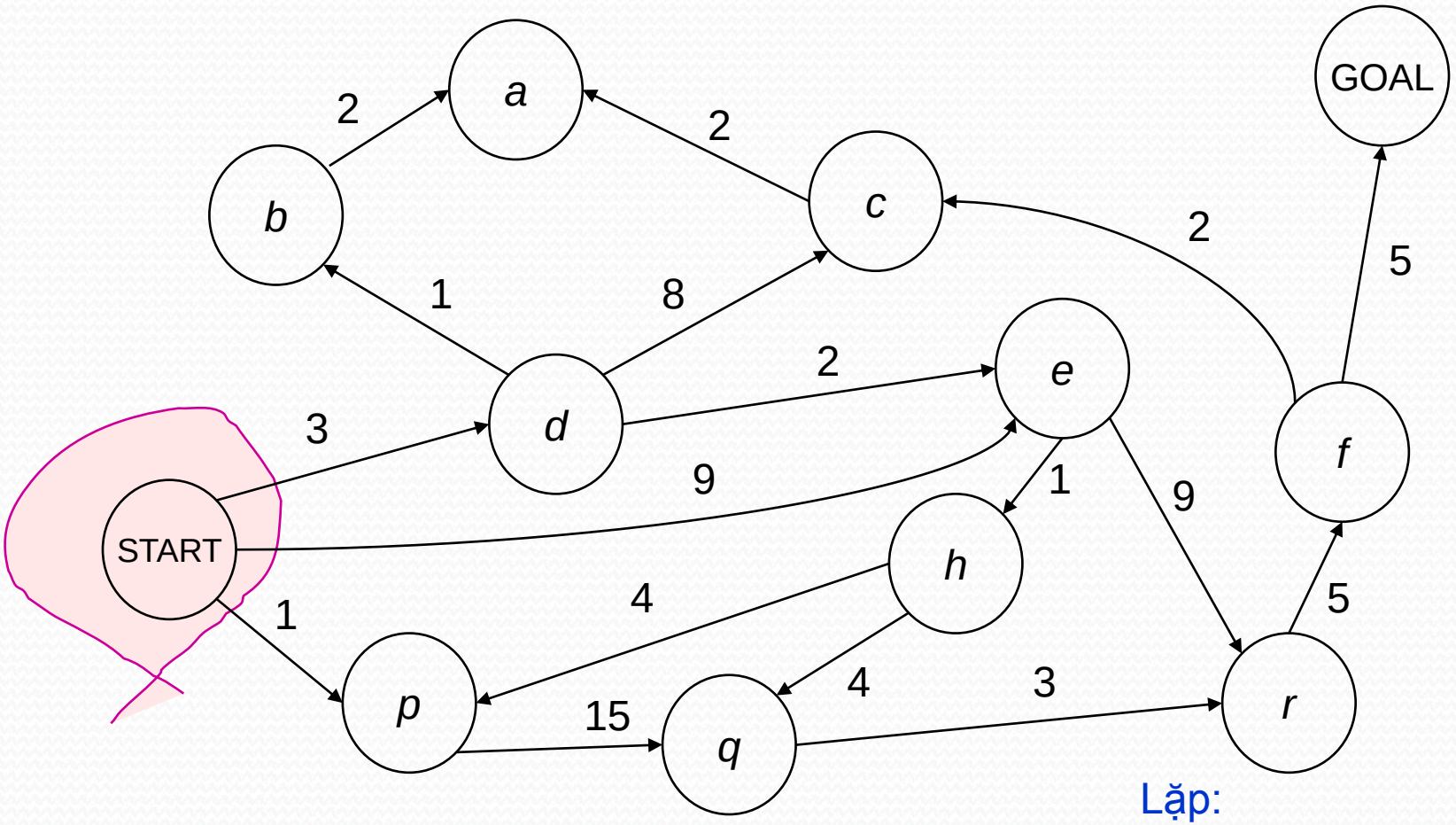
  Độ ưu tiên của trạng thái *s* = *g(s)* = chi phí đến *s* dùng đường đi cho bởi con trỏ quay lui.

Bắt đầu UCS



PQ = { (S,0) }

Lặp UCS

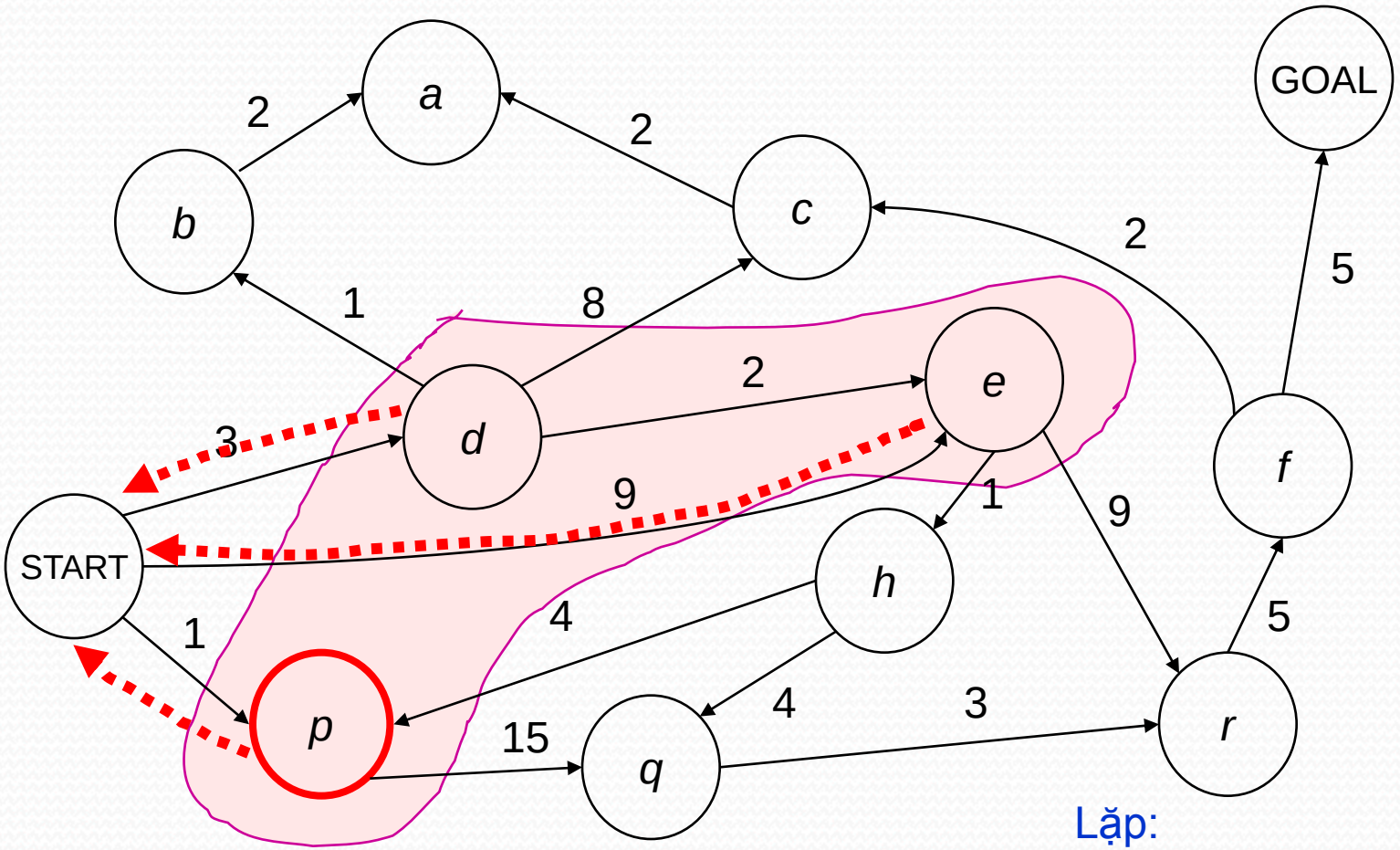


PQ = { (S,0) }

Lặp:

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

Lặp UCS

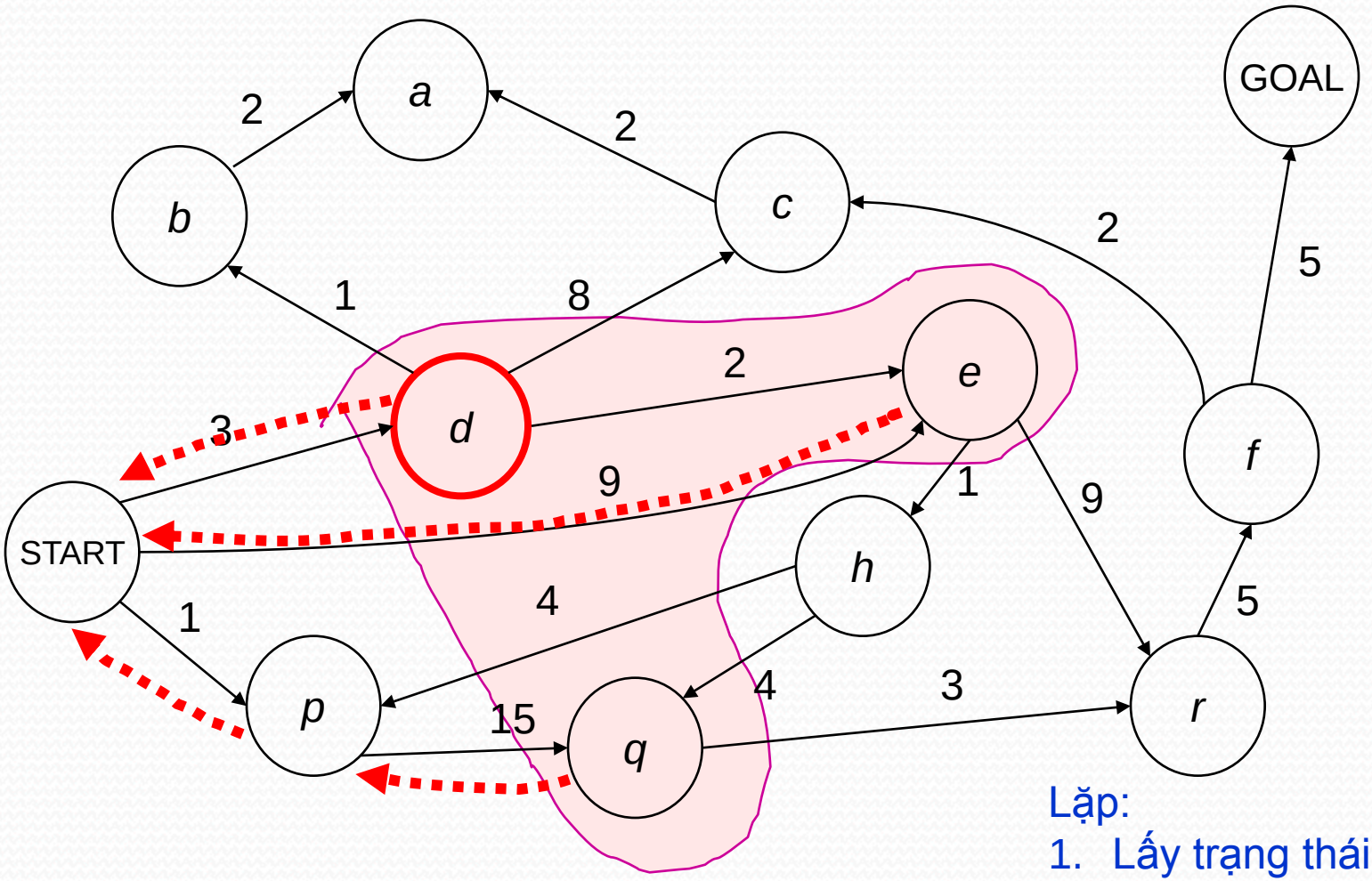


Lặp:

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

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

Lặp UCS

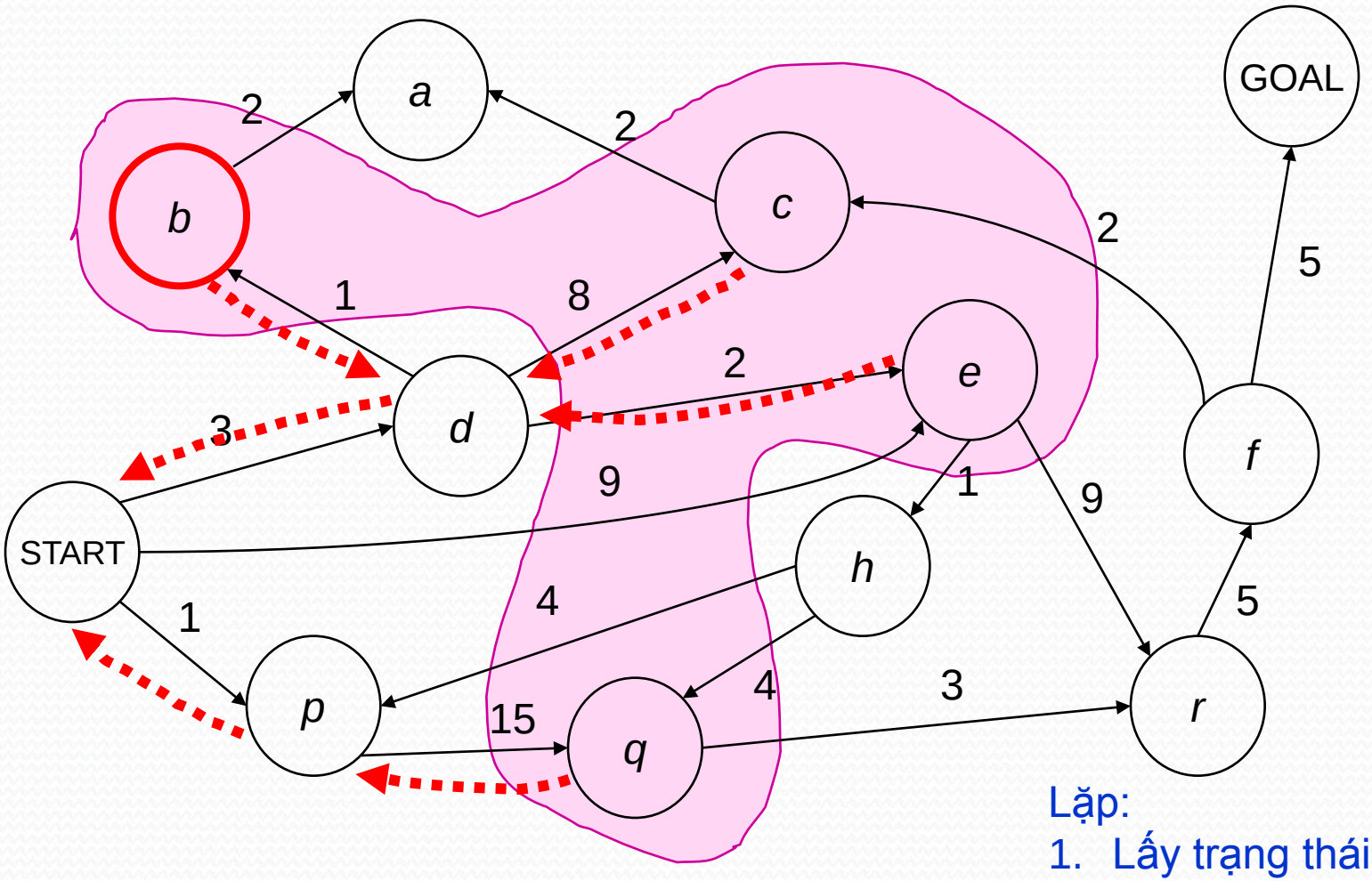


PQ = { (d,3) , (e,9) , (q,16) }

Lặp:

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

Lặp UCS

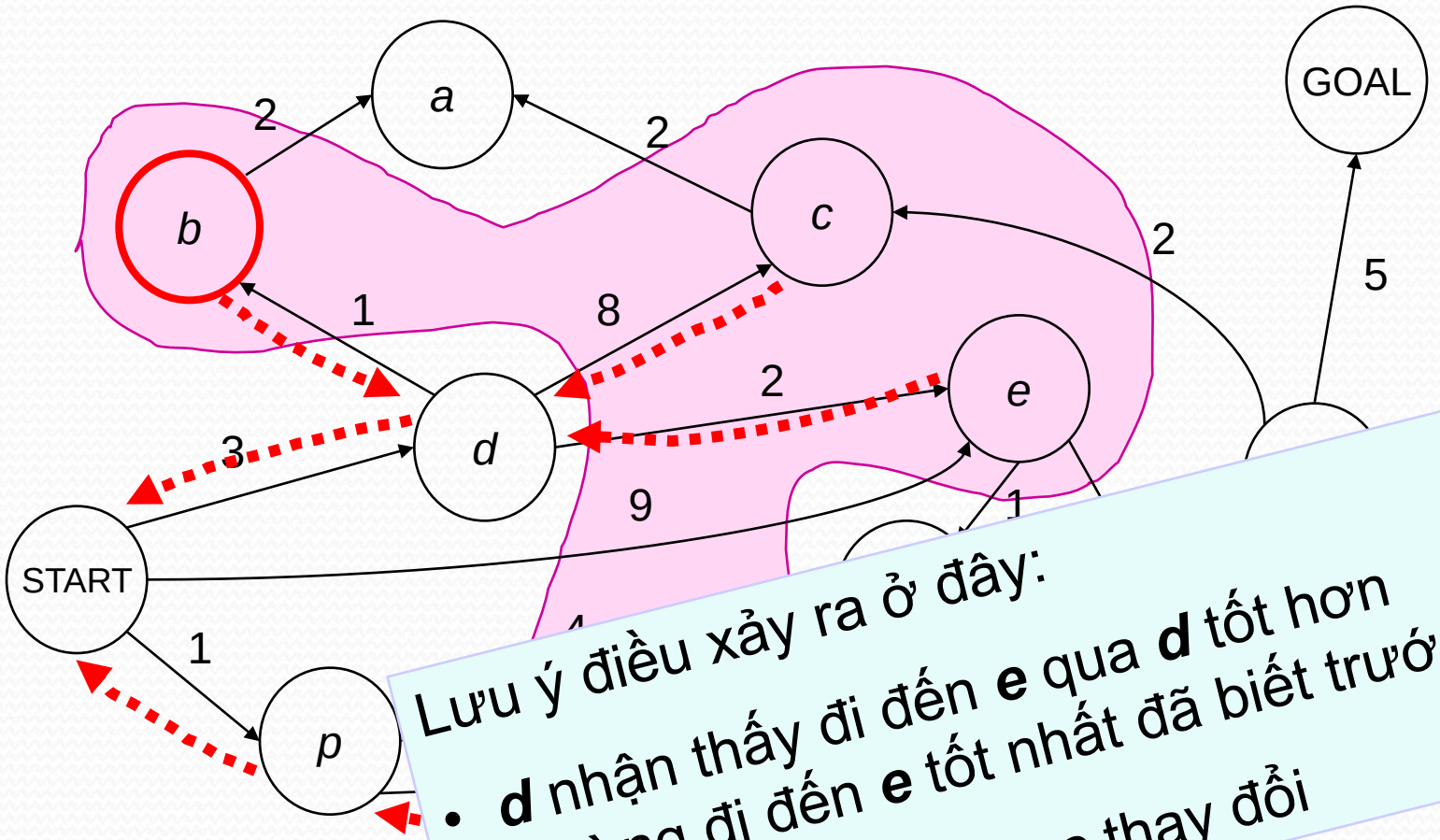


PQ = { (b,4) , (e,5) , (c,11) , (q,16) }

Lặp:

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

Lặp UCS



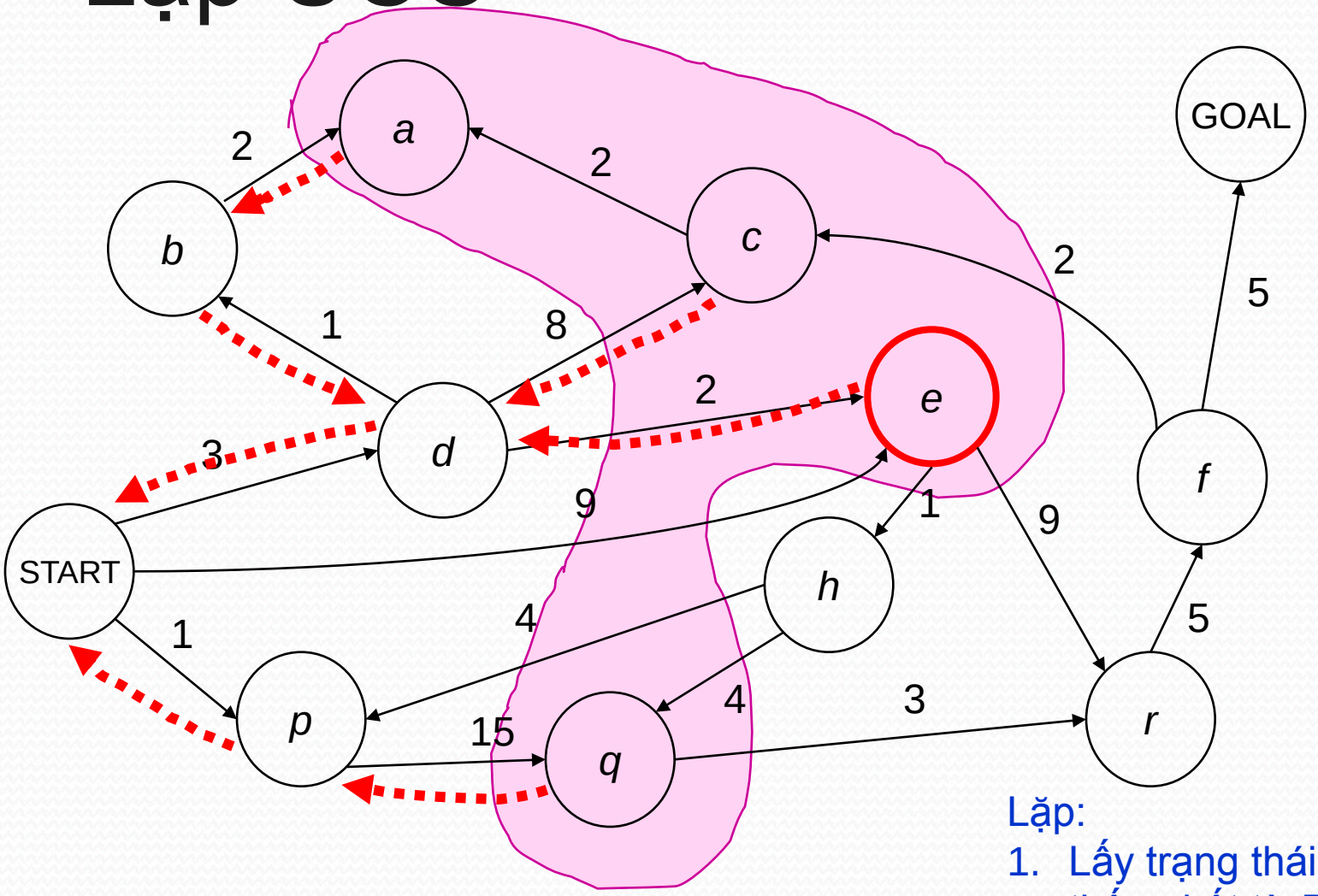
Lưu ý điều xảy ra ở đây:

- **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

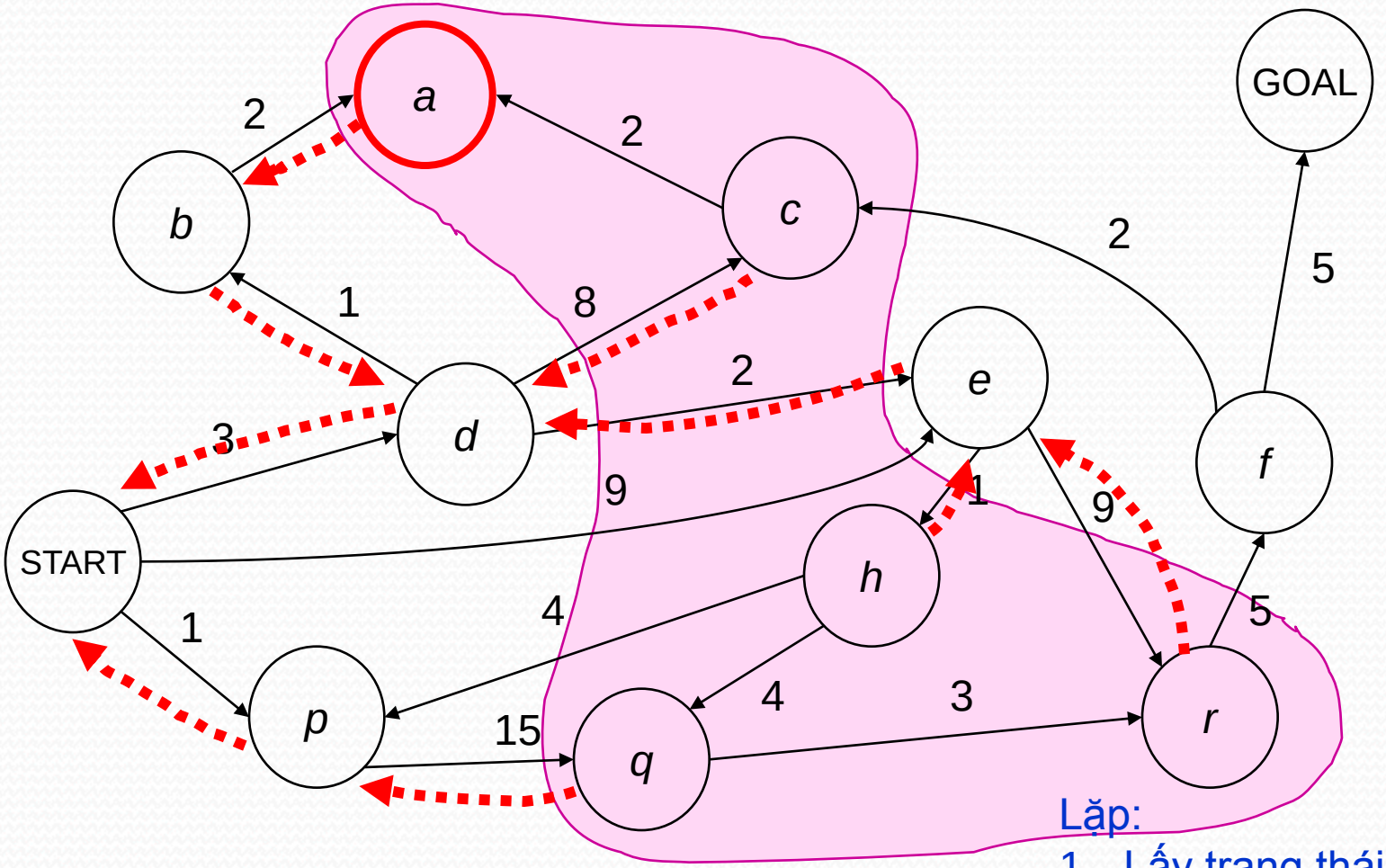


PQ = { (e,5) , (a,6) , (c,11) , (q,16) }

Lặp:

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

Lặp UCS

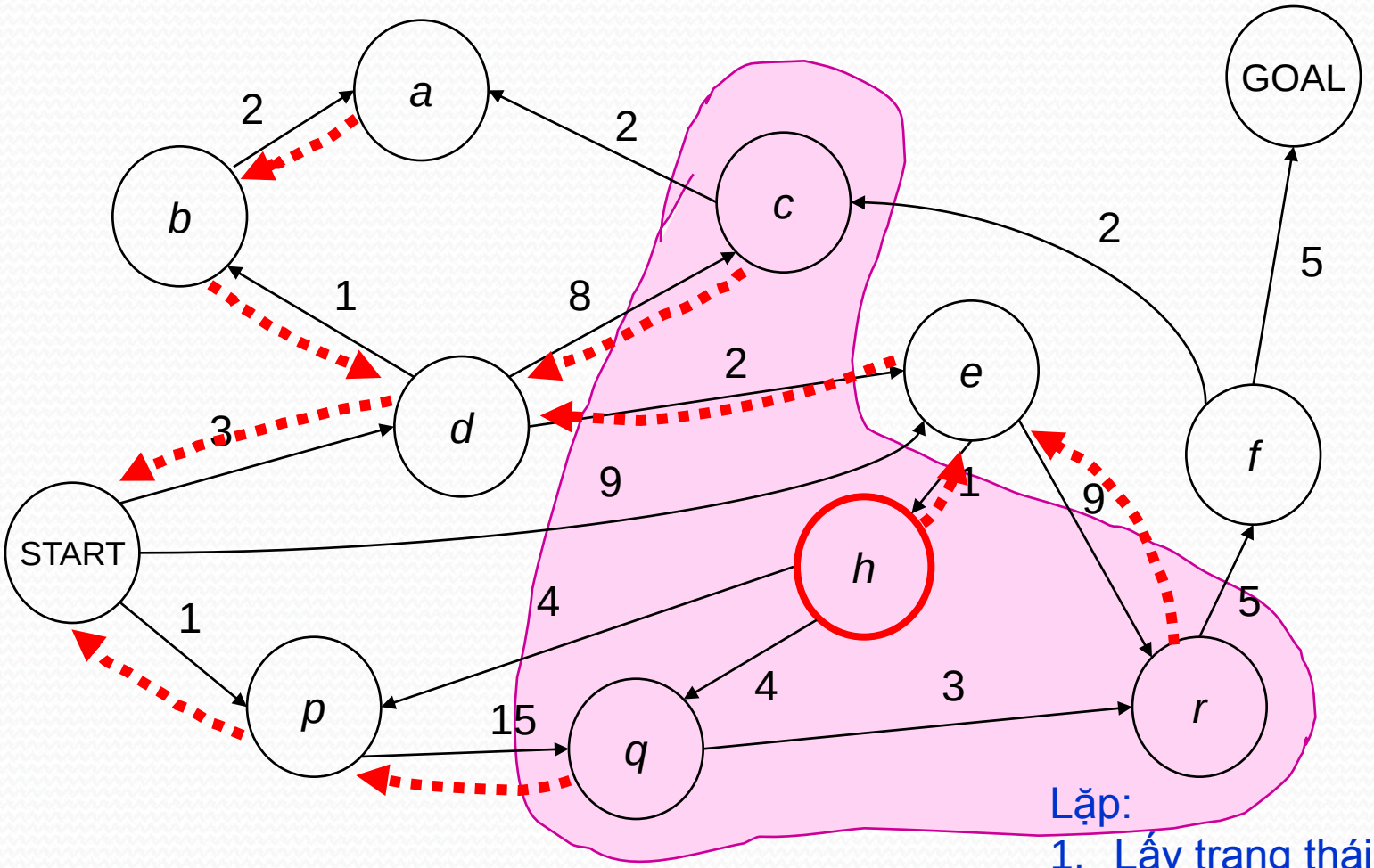


Lặp:

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

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

Lặp UCS

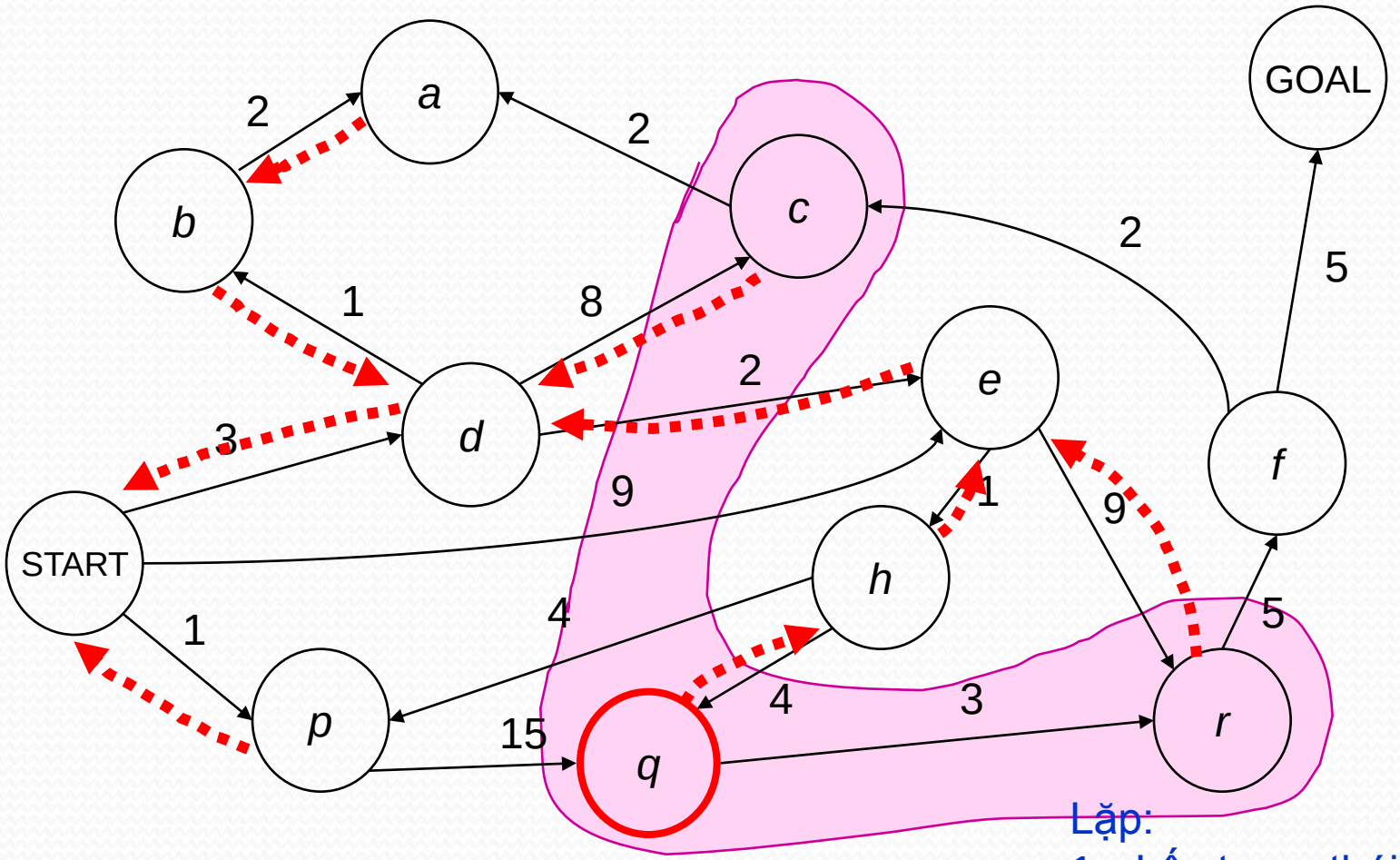


PQ = { (h,6),(c,11),(r,14),(q,16) }

Lặp:

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

Lặp UCS



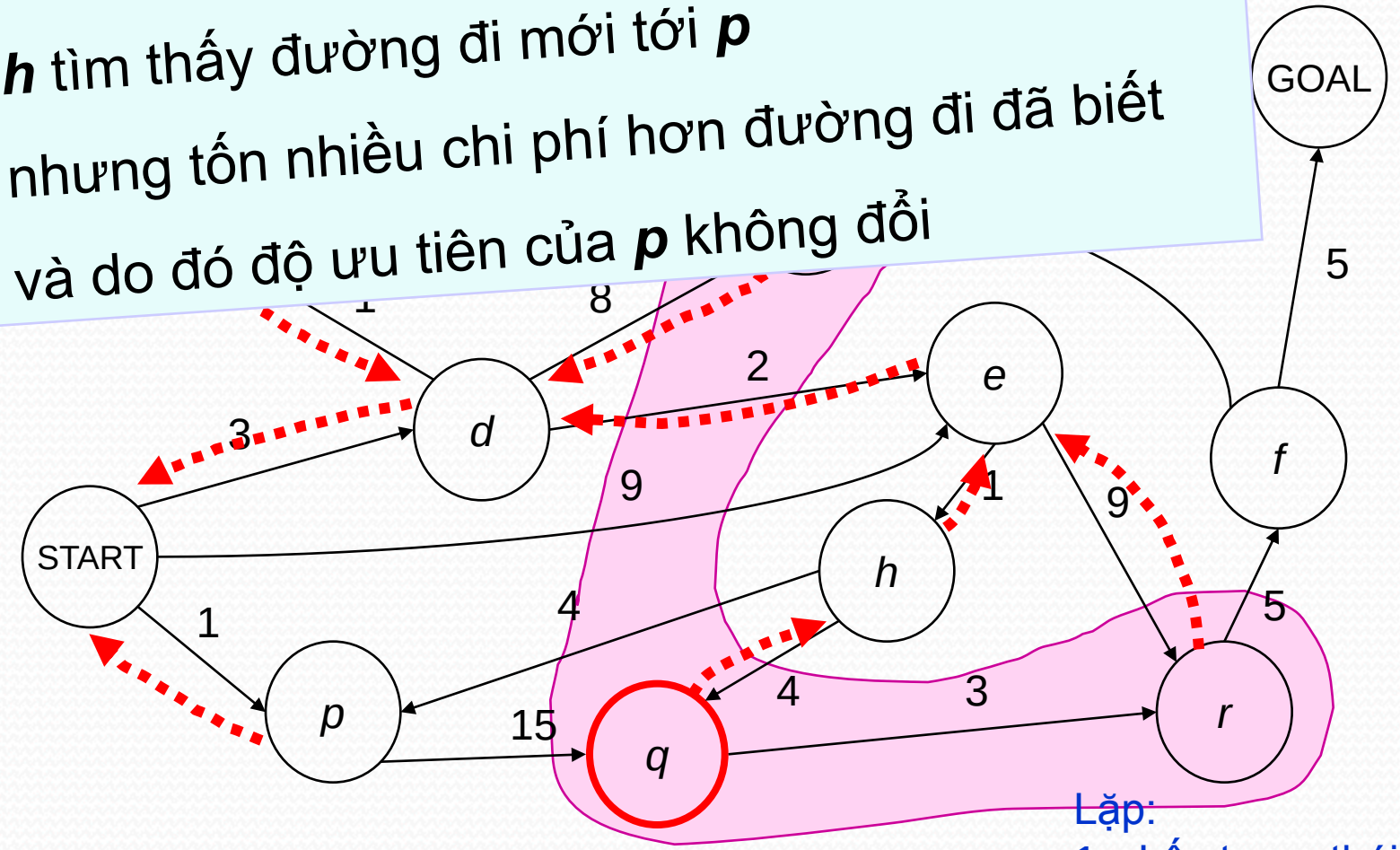
Lặp:

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

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

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

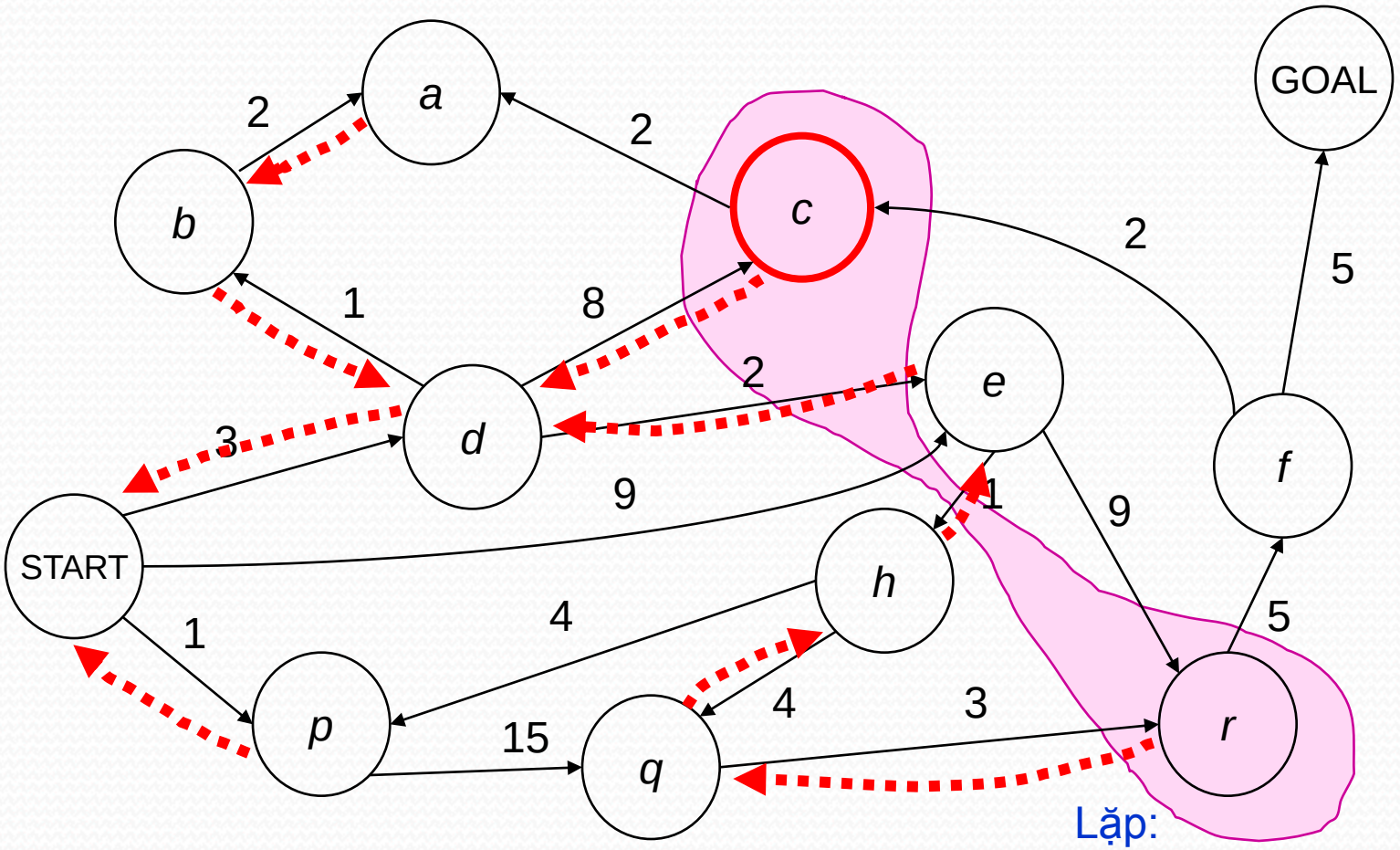


Lặp:

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

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

Lặp UCS

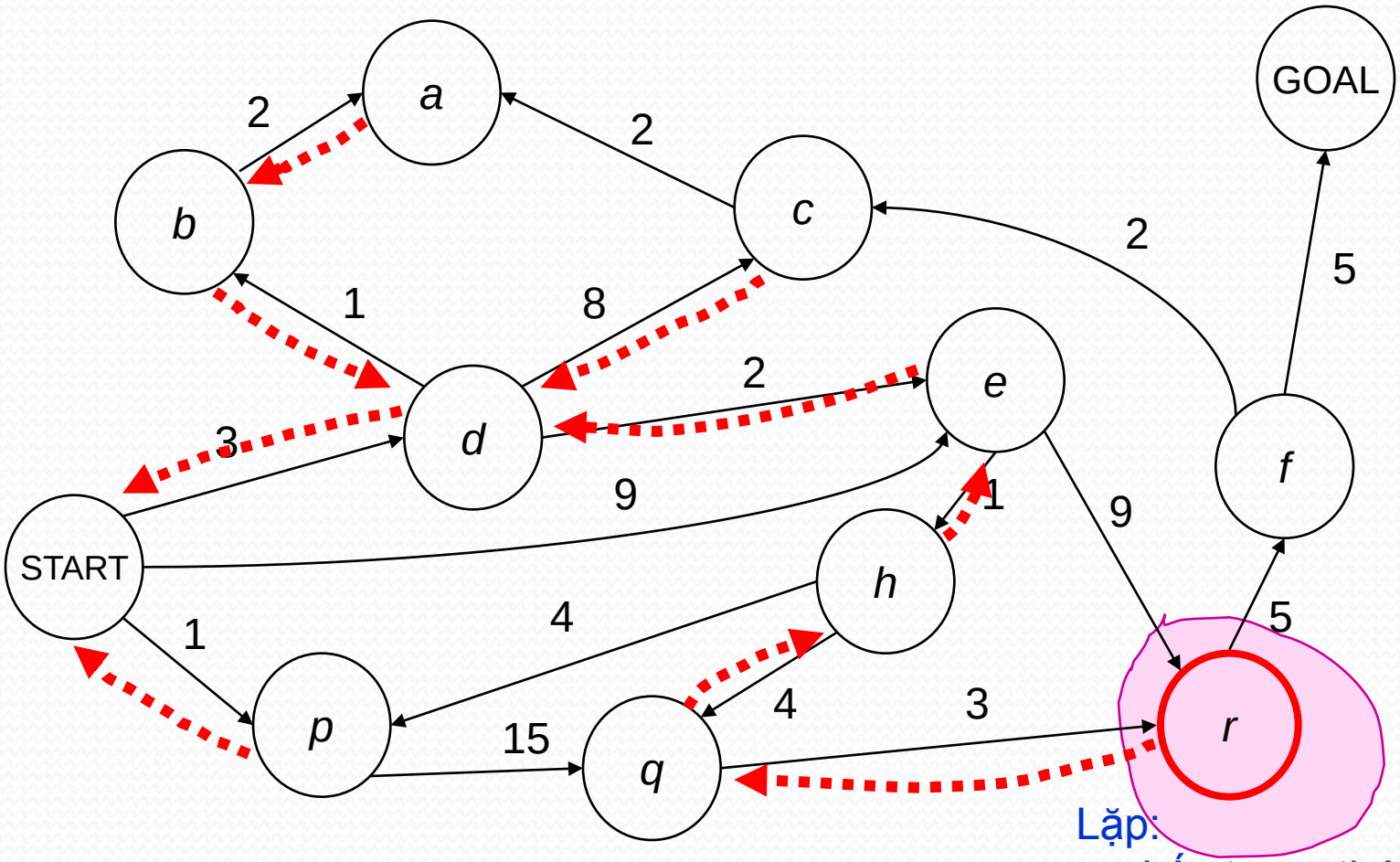


Lặp:

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

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

Lặp UCS

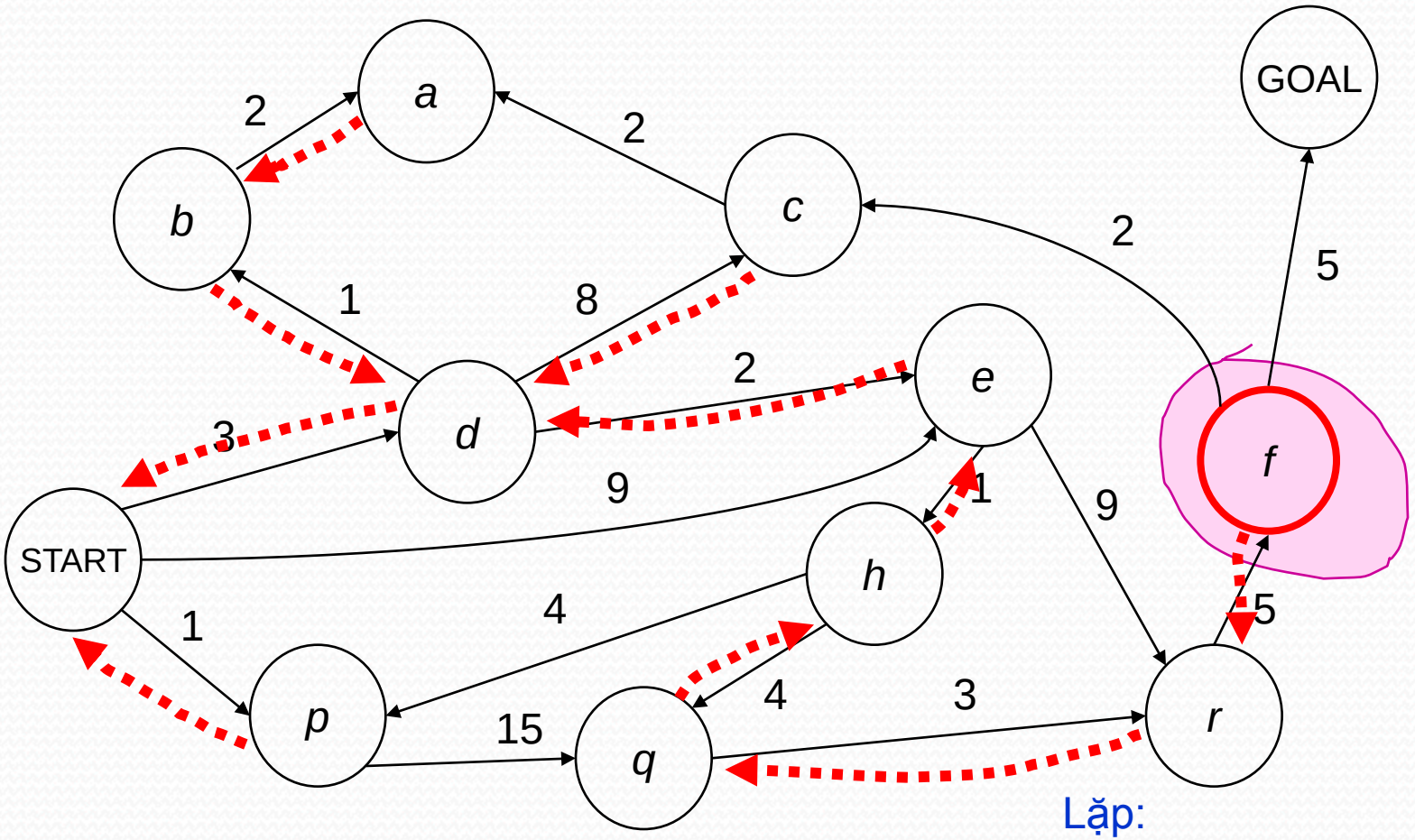


Lặp:

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

PQ = { (r,13) }

Lặp UCS

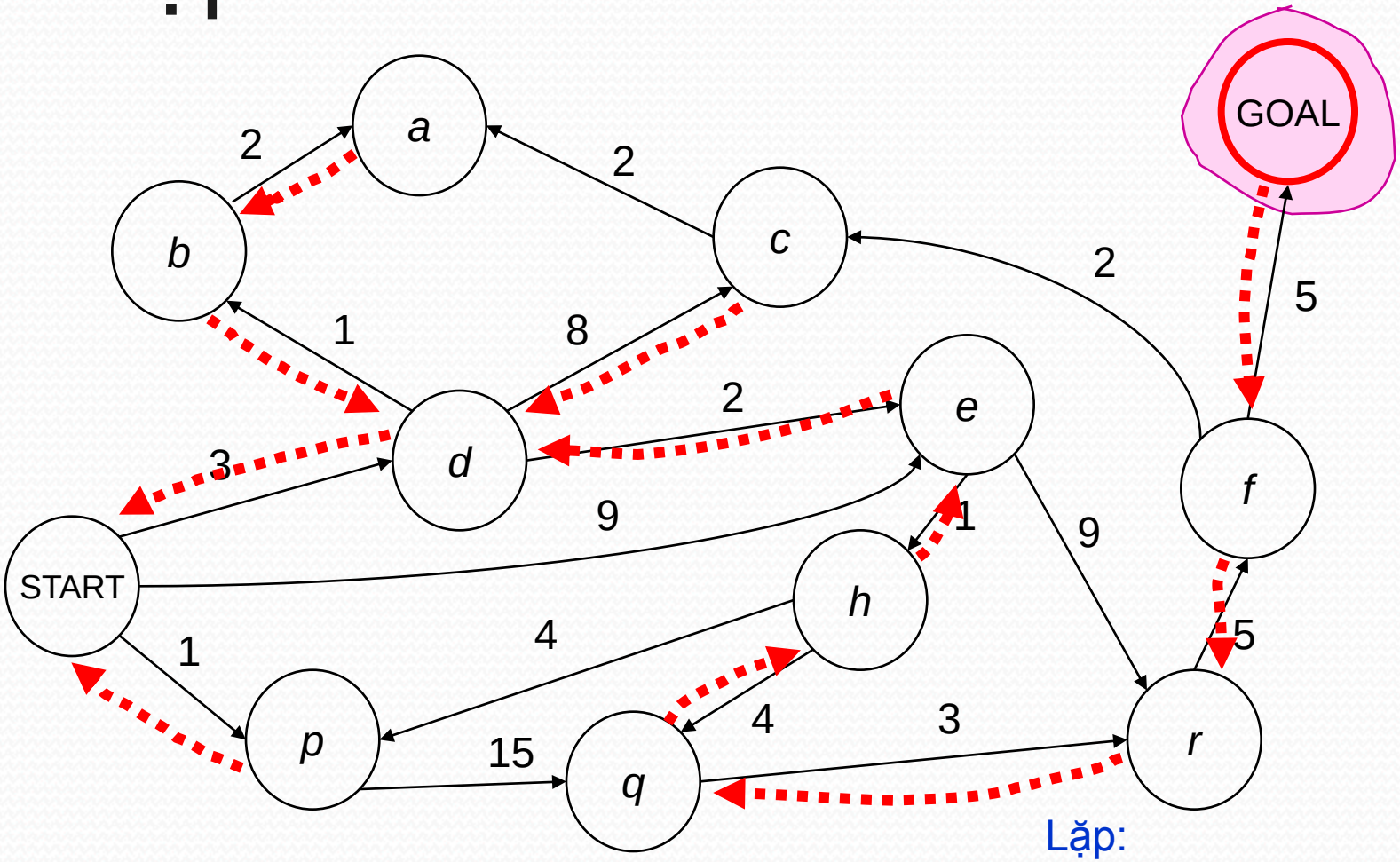


Lặp:

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

PQ = { (f,18) }

Lặp UCS

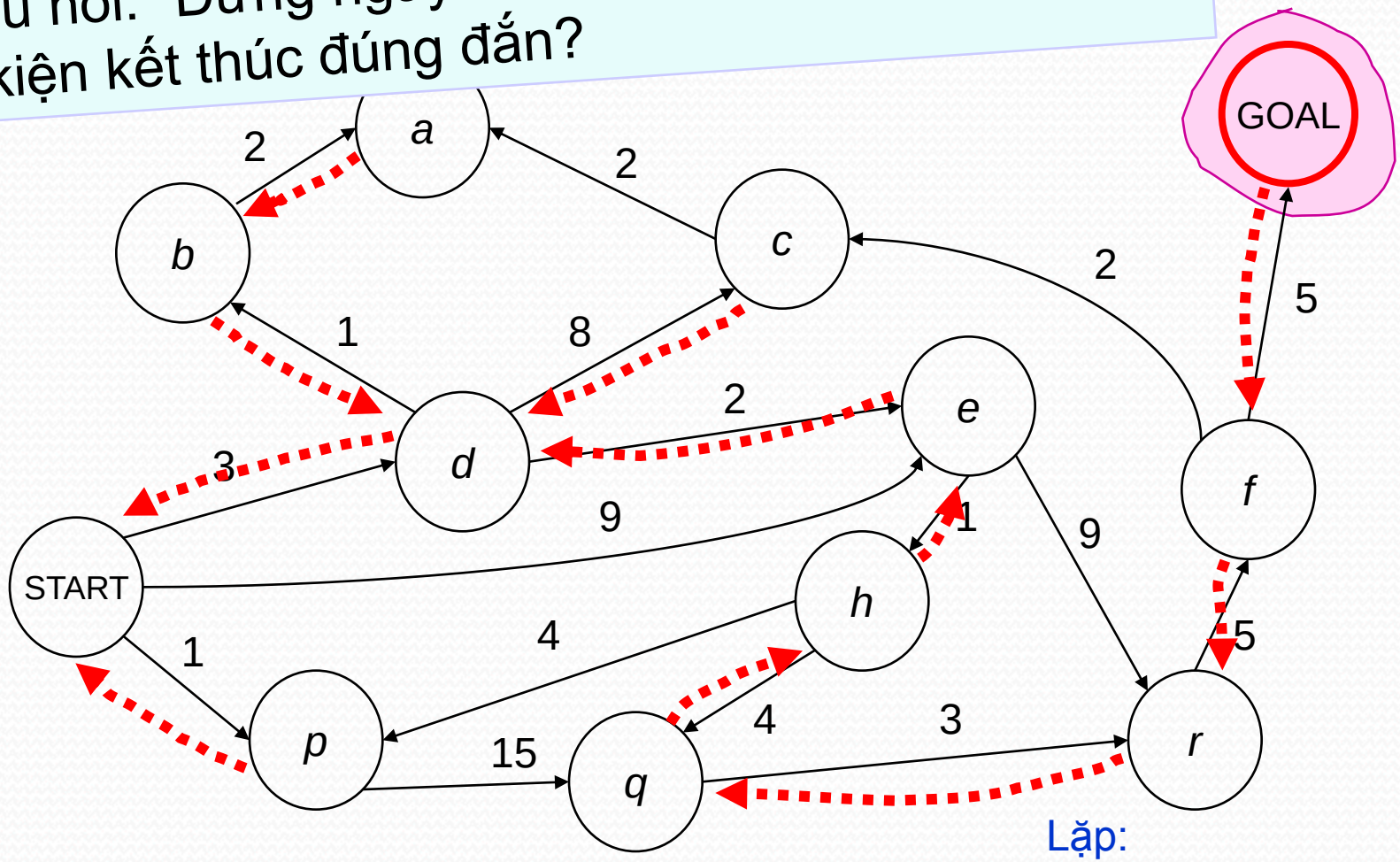


Lặp:

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

PQ = { (G,23) }

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

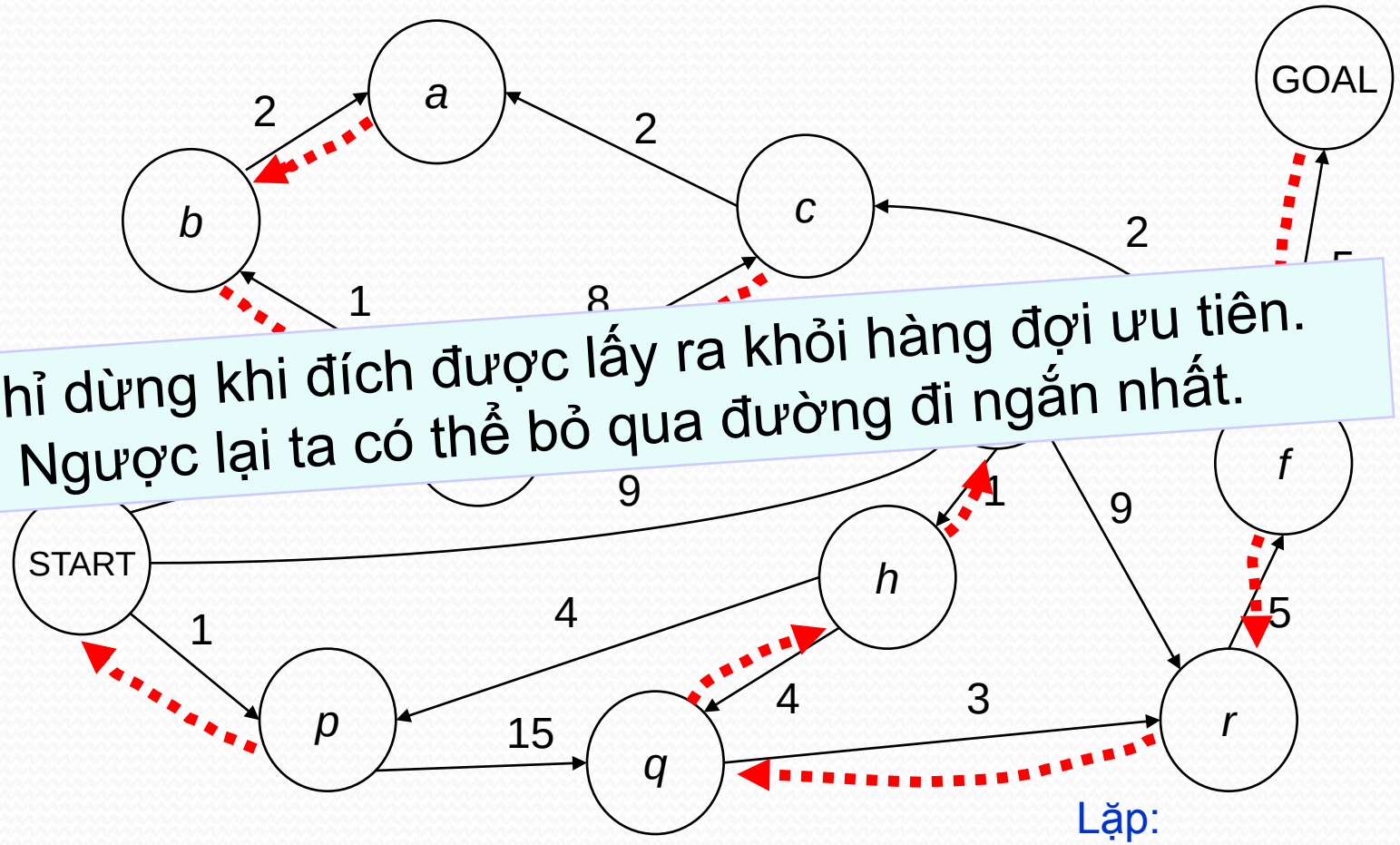


Lặp:

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

PQ = { (G,23) }

Kết thúc UCS



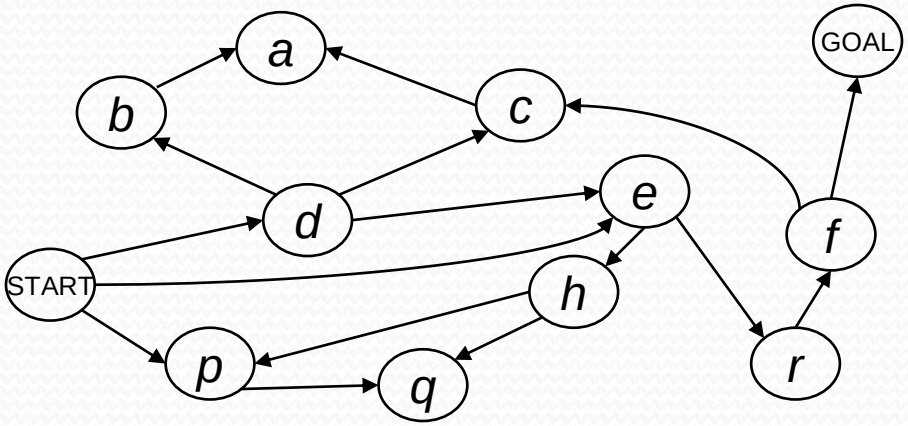
Chỉ dừng khi đích được lấy ra khỏi hàng đợi ưu tiên.
Ngược lại ta có thể bỏ qua đường đi ngắn nhất.

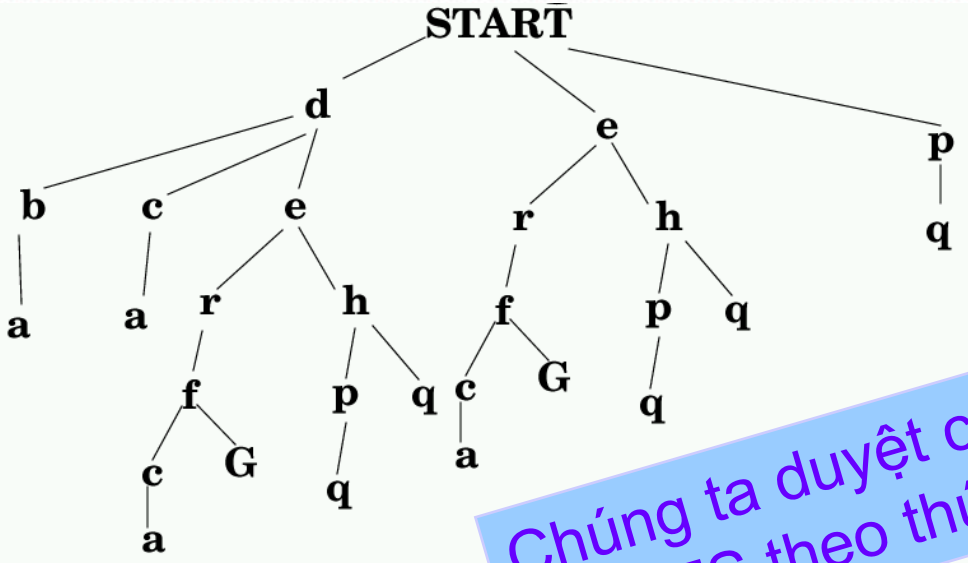
Lặp:

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

PQ = { }

Biểu diễn cây tìm kiếm





Chúng ta duyệt cây tìm kiếm với BFS theo thứ tự nào?

Đánh giá một thuật toán tìm kiếm

- Tính đầy đủ: thuật toán có bảo đảm tìm thấy lời giải nếu có hay không?
- Có bảo đảm tìm thấy tối ưu? (nó sẽ tìm thấy đường đi có chi phí ít nhất?)
- Độ phức tạp về thời gian
- Độ phức tạp về không gian (sử dụng bộ nhớ)

Các biến:

N	số trạng thái của bài toán
B	nhân tố phân nhánh trung bình (số con trung bình) ($B>1$)
L	đọ dài đường đi từ start đến goal với số bước ngắn nhất

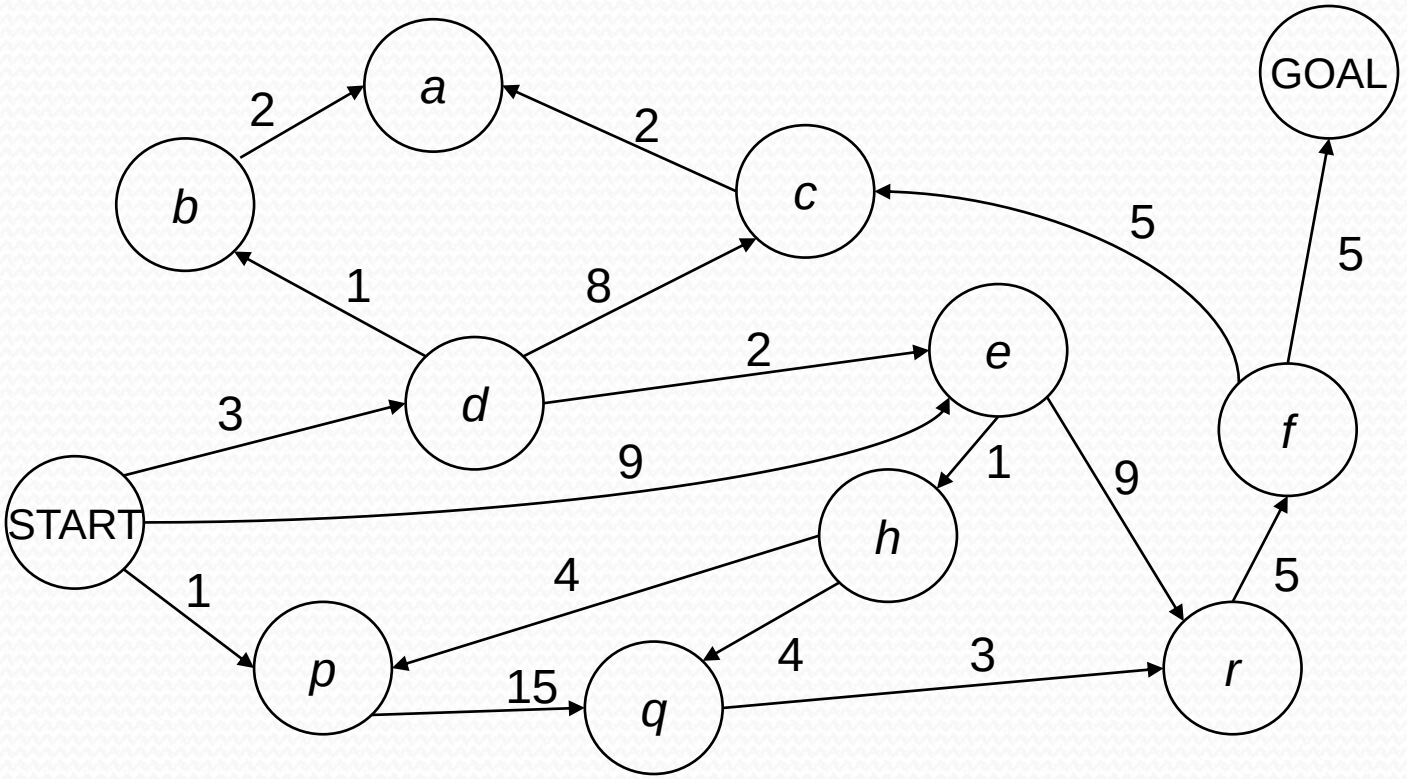
Chúng ta đánh giá thuật toán như thế nào?

Đánh giá một thuật toán

N	số trạng thái trong bài toán
B	thừa số phân nhánh trung bình (số con trung bình) ($B>1$)
L	độ dài đường đi từ start đến goal với số bước (chi phí) ít nhất
Q	kích cỡ hàng đợi ưu tiên trung bình

Thuật toán		Đủ	Tối ưu	Thời gian	Không gian
BFS	Breadth First Search	C	Nếu tất cả chuyển đổi cùng chi phí	$O(min(N,B^L))$	$O(min(N,B^L))$
LCBFS	Least Cost BFS	C	C	$O(B^L)$	$O(min(N,B^L))$
UCS	Uniform Cost Search	C	C	$O(log(Q) * min(N,B^L))$	$O(min(N,B^L))$

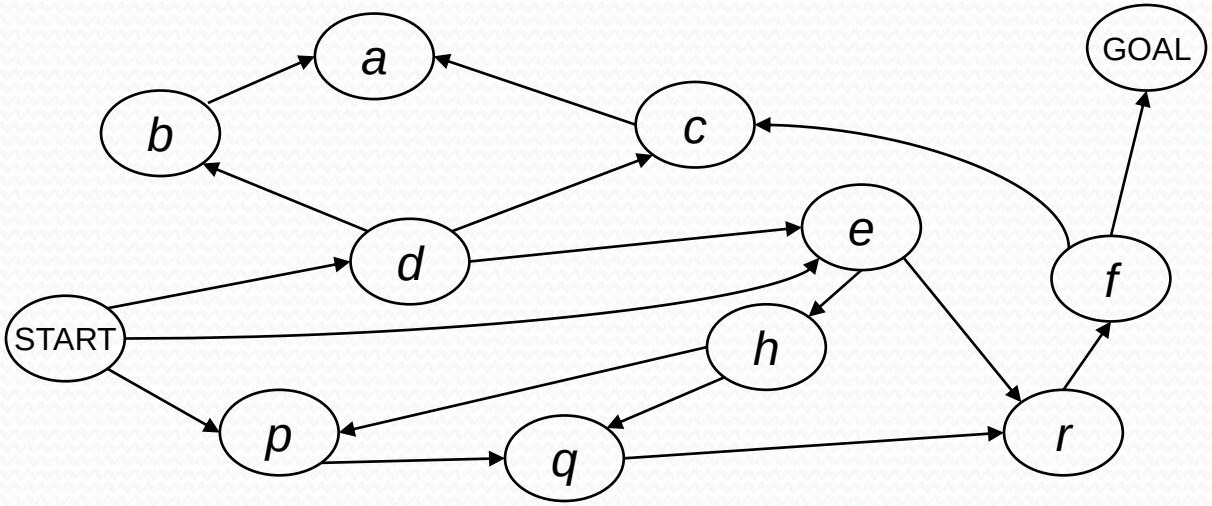
Tìm kiếm Theo Chiều Sâu



Một thay thế cho BFS. Luôn mở từ node vừa mới mở nhất, nếu nó có bất kỳ node con chưa thử nào. Ngược lại quay lại node trước đó trên đường đi.

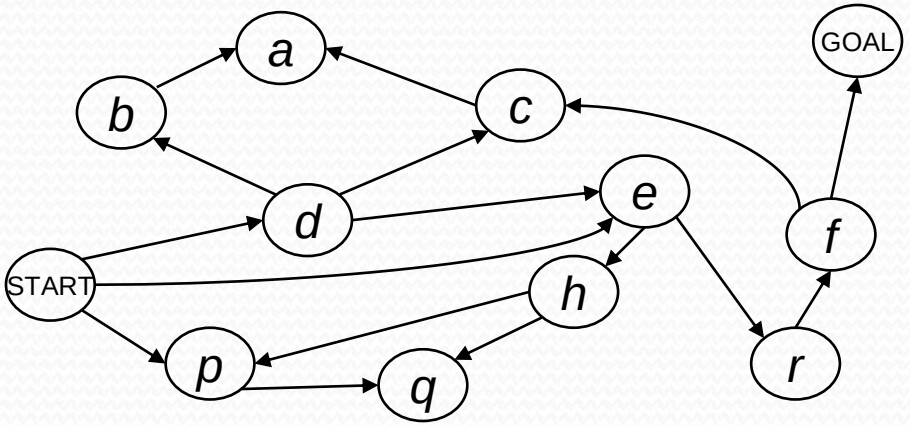
DFS trên thực tế

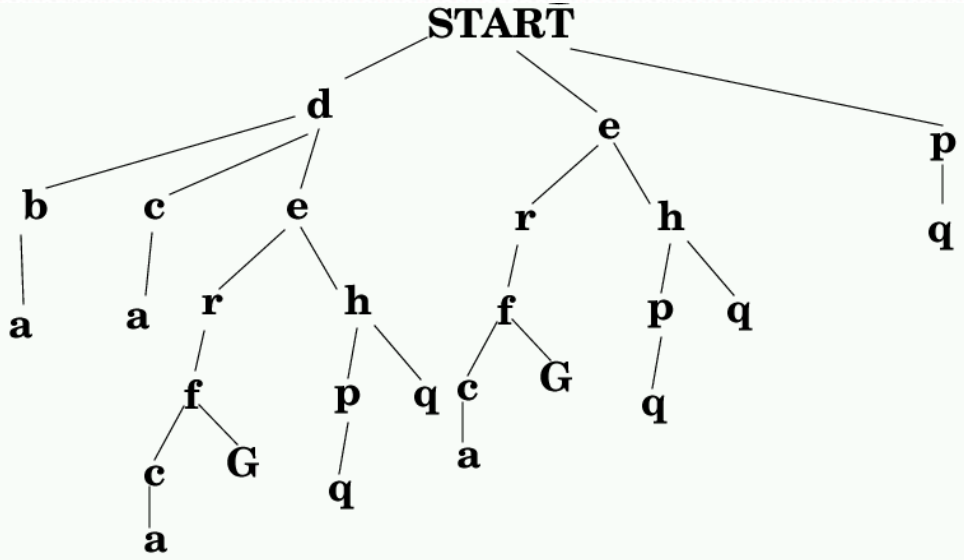
START
START *d*
START *d b*
START *d b a*
START *d c*
START *d c a*
START *d e*
START *d e r*
START *d e r f*
START *d e r f c*
START *d e r f c a*
START *d e r f* GOAL



Duyệt cây tìm kiếm DFS

Bạn có thể vẽ thứ tự mà trong đó các node của cây tìm kiếm được viếng?



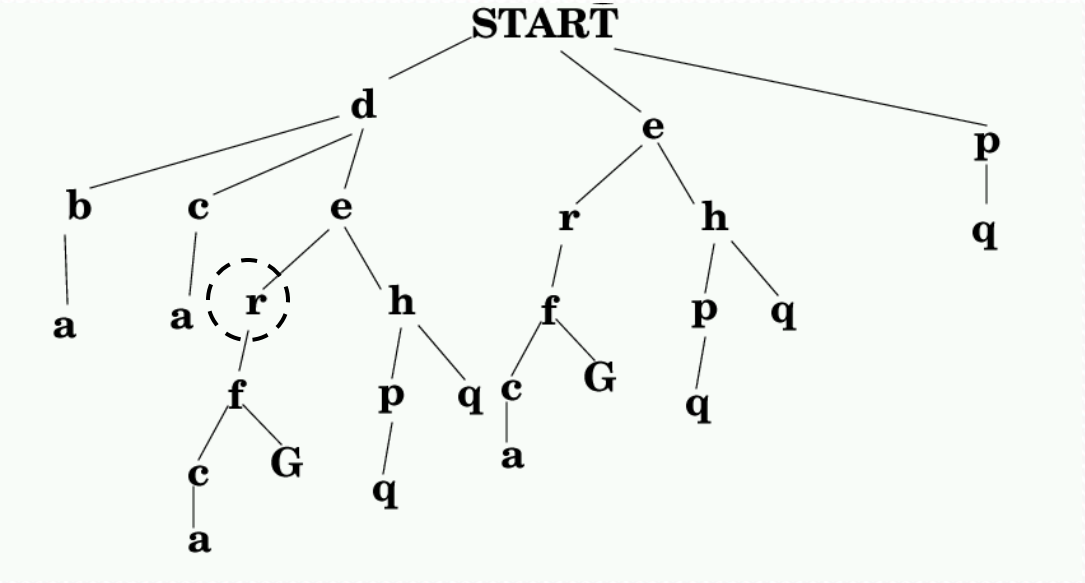


Thuật toán DFS

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.

Đánh giá một thuật toán

N	số trạng thái trong bài toán
B	thừa số phân nhánh trung bình (số con trung bình) ($B>1$)
L	độ dài đường đi từ start đến goal với số bước (chi phí) ít nhất
Q	kích cỡ hàng đợi ưu tiên trung bình

Thuật toán		Đủ	Tối ưu	Thời gian	Không gian
BFS	Breadth First Search	C	Nếu chi phí chuyển đổi như nhau	$O(min(N,B^L))$	$O(min(N,B^L))$
LCBFS	Least Cost BFS	C	C	$O(B^L)$	$O(min(N,B^L))$
UCS	Uniform Cost Search	C	C	$O(log(Q) * min(N,B^L))$	$O(min(N,B^L))$
DFS	Depth First Search				

Đánh giá một thuật toán

N	số trạng thái trong bài toán
B	thừa số phân nhánh trung bình (số con trung bình) ($B>1$)
L	độ dài đường đi từ start đến goal với số bước (chi phí) ít nhất
Q	kích cỡ hàng đợi ưu tiên trung bình

Thuật toán		Đủ	Tối ưu	Thời gian	Không gian
BFS	Breadth First Search	C	Nếu chi phí chuyển đổi như nhau	$O(min(N,B^L))$	$O(min(N,B^L))$
LCBFS	Least Cost BFS	C	C	$O(B^L)$	$O(min(N,B^L))$
UCS	Uniform Cost Search	C	C	$O(log(Q) * min(N,B^L))$	$O(min(N,B^L))$
DFS	Depth First Search	K	K	*N/A*	*N/A*

Đánh giá một thuật toán

N	số trạng thái trong bài toán
B	thừa số phân nhánh trung bình (số con trung bình) ($B>1$)
L	độ dài đường đi từ start đến goal với số bước (chi phí) ít nhất
LMAX	Độ dài đường đi dài nhất từ start đến bất cứ đâu
Q	kích cỡ hàng đợi ưu tiên trung bình

Thuật toán		Đủ	Tối ưu	Thời gian	Không gian
BFS	Breadth First Search	C	Nếu chi phí chuyển đổi như nhau	$O(min(N,B^L))$	$O(min(N,B^L))$
LCBFS	Least Cost BFS	C	C	$O(B^L)$	$O(min(N,B^L))$
UCS	Uniform Cost Search	C	C	$O(log(Q) * min(N,B^L))$	$O(min(N,B^L))$
DFS**	Depth First Search				

Giả sử Không gian Tìm kiếm không chu trình

Đánh giá một thuật toán

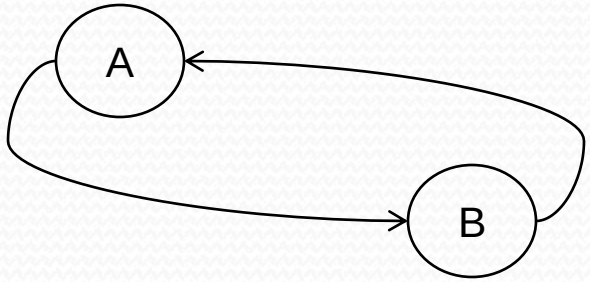
N	số trạng thái trong bài toán
B	thừa số phân nhánh trung bình (số con trung bình) ($B>1$)
L	độ dài đường đi từ start đến goal với số bước (chi phí) ít nhất
LMAX	Độ dài đường đi dài nhất từ start đến bất cứ đâu
Q	kích cỡ hàng đợi ưu tiên trung bình

Thuật toán		Đủ	Tối ưu	Thời gian	Không gian
BFS	Breadth First Search	C	Nếu chi phí chuyển đổi như nhau	$O(min(N,B^L))$	$O(min(N,B^L))$
LCBFS	Least Cost BFS	C	C	$O(B^L)$	$O(min(N,B^L))$
UCS	Uniform Cost Search	C	C	$O(log(Q) * min(N,B^L))$	$O(min(N,B^L))$
DFS**	Depth First Search	C	K	$O(B^{LMAX})$	$O(LMAX)$

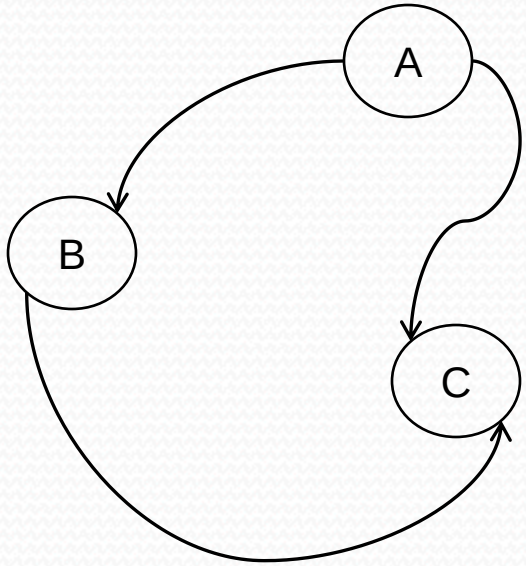
Giả sử Không gian Tìm kiếm không chu trình

Câu hỏi suy nghĩ

- Làm sao để ngăn ngừa lặp vô tận trong DFS ?



- Làm sao bắt buộc nó đưa ra một lời giải tối ưu?



Câu hỏi suy nghĩ

- Làm sao để ngăn ngừa lặp vô tận trong DFS ?
- Làm sao bắt buộc nó đưa ra một lời giải tối ưu?

Trả lời 1:

PC-DFS (Path Checking DFS):

Trả lời 2:

MEMDFS (Memorizing DFS):

Câu hỏi suy nghĩ

- Làm sao để ngăn ngừa lặp vô tận trong DFS ?
- Làm sao bắt buộc nó đưa ra một lời giải tối ưu?

Trả lời 1:

PC-DFS (Path Checking DFS):

Không gọi lại một trạng thái nếu nó đã có trên đường đi

Trả lời 2:

MEMDFS (Memorizing DFS):

Nhớ tất cả trạng thái đã mở. Không bao giờ mở hai lần.

Câu hỏi suy nghĩ

- Làm sao để ng
  ngừa lặp
  trong
- Làm s buộc
  phải trả một lời
  tối ưu?

Có khi nào PCDFS tốt hơn MEMDFS không?

Có khi nào MEMDFS tốt hơn PCDFS không?

Trả lời 1:

PC-DFS (Path Checking DFS):

Không gọi lại một trạng thái nếu nó đã có trên đường đi

Trả lời 2:

MEMDFS (Memoizing DFS):

Nhớ tất cả trạng thái đã mở. Không bao giờ mở hai lần.

N	số trạng thái trong bài toán
B	thừa số phân nhánh trung bình (số con trung bình) ($B>1$)
L	độ dài đường đi từ start đến goal với số bước (chi phí) ít nhất
LMAX	Độ dài đường đi **không chu trình** dài nhất từ start đến bất cứ đâu
Q	kích cỡ hàng đợi ưu tiên trung bình

Thuật toán		Đủ	Tối ưu	Thời gian	Không gian
BFS	Breadth First Search	C	Nếu chi phí chuyển đổi như nhau (1)	$O(min(N,B^L))$	$O(min(N,B^L))$
LCBFS	Least Cost BFS	C	C	$O(B^L)$	$O(min(N,B^L))$
UCS	Uniform Cost Search	C	C	$O(log(Q) * min(N,B^L))$	$O(min(N,B^L))$
PCDFS	Path Check DFS	C	K	$O(B^{LMAX})$	$O(LMAX)$
MEMDFS	Memoizing DFS	C	K	$O(min(N,B^{LMAX}))$	$O(min(N,B^{LMAX}))$

Lặp Sâu dần

Lặp sâu dần là một thuật toán đơn giản dùng DFS làm một thủ tục con:

1. Thực hiện DFS chỉ tìm các đường đi có độ dài 1 hay ít hơn. (DFS bỏ các đường đi nào dài hơn hay bằng 2)
2. Nếu "1" thất bại, thực hiện DFS chỉ tìm các đường đi có độ dài 2 hay ít hơn.
3. Nếu "2" thất bại, thực hiện DFS chỉ tìm các đường đi có độ dài 3 hay ít hơn.

….và tiếp tục cho đến khi thành công.

Chi phí là

$O(b^1 + b^2 + b^3 + b^4 \ldots + b^L) = O(b^L)$

Có thể tốt hơn nhiều so với DFS thông thường. Nhưng chi phí có thể lớn hơn nhiều so với số trạng thái.

Đánh giá một thuật toán

N	số trạng thái trong bài toán
B	thừa số phân nhánh trung bình (số con trung bình) ($B>1$)
L	độ dài đường đi từ start đến goal với số bước (chi phí) ít nhất
LMAX	Độ dài đường đi **không chu trình** dài nhất từ start đến bất cứ đâu
Q	kích cỡ hàng đợi ưu tiên trung bình

Thuật toán		Đủ	Tối ưu	Thời gian	Không gian
BFS	Breadth First Search	C	Nếu chi phí chuyển đổi như nhau (1)	$O(min(N,B^L))$	$O(min(N,B^L))$
LCBFS	Least Cost BFS	C	C	$O(B^L)$	$O(min(N,B^L))$
UCS	Uniform Cost Search	C	C	$O(log(Q) * min(N,B^L))$	$O(min(N,B^L))$
PCDFS	Path Check DFS	C	K	$O(B^{LMAX})$	$O(LMAX)$
MEMDFS	Memoizing DFS	C	K	$O(min(N,B^{LMAX}))$	$O(min(N,B^{LMAX}))$
ID	Iterative Deepening	Y	(1)	$O(B^L)$	$O(L)$

Điều cần nắm

- Hiểu thấu đáo về BFS, LCBFS, UCS, DFS, PCDFS, MEMDFS
- Hiểu các khái niệm về việc liệu một tìm kiếm là đầy đủ, tối ưu hay không, độ phức tạp về thời gian và không gian của nó
- Hiểu ý tưởng đằng sau lặp sâu dần.