Trường Đại học Khoa học Tự nhiên
Khoa Công nghệ Thông tin

TÀI LIỆU LÝ THUYẾT TRÍ TUỆ NHÂN TẠO

## Chủ đề 1

# CÁC GIẢI PHÁP TÌM KIẾM MÙ

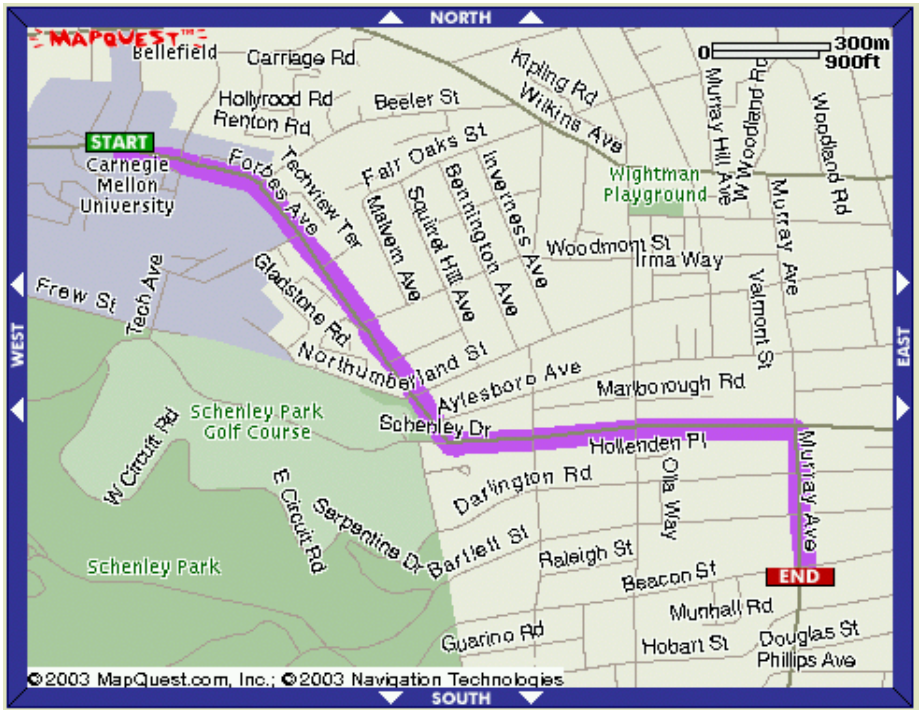
Giảng viên: Vũ Thanh Hưng
Email: vthung@fit.hcmus.edu.vn

HKI/2012-2013

# NỘI DUNG

- Định nghĩa bài toán tìm kiếm
- Nhóm thuật toán tìm kiếm theo chiều rộng
  - Tìm kiếm theo chiều rộng
  - Tìm kiếm theo chiều rộng chi phí nhỏ nhất
  - Tìm kiếm chi phí đồng nhất
- Nhóm thuật toán tìm kiếm theo chiều sâu
  - Tìm kiếm theo chiều sâu
  - Tìm kiếm chiều sâu giới hạn
- Một số thuật toán tìm kiếm khác
- Tài liệu tham khảo

# ỨNG DỤNG THỰC TIỄN

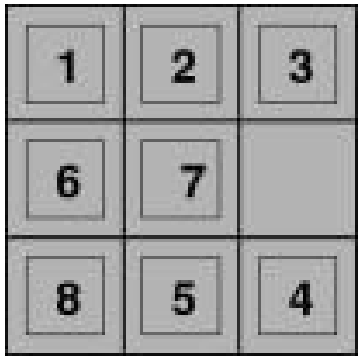



Tìm đường đi trong thành phố

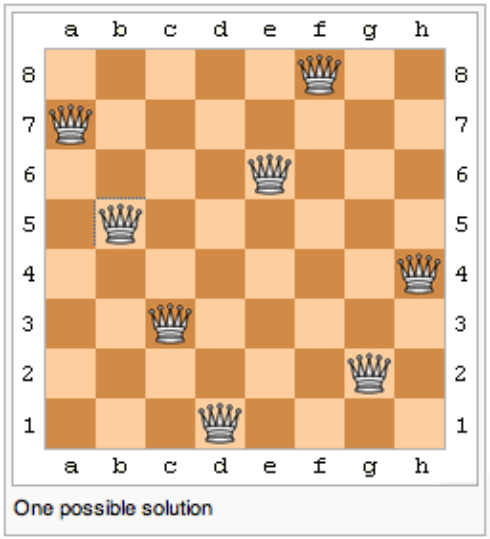
Tìm đường mê cung

Robot tự vận hành

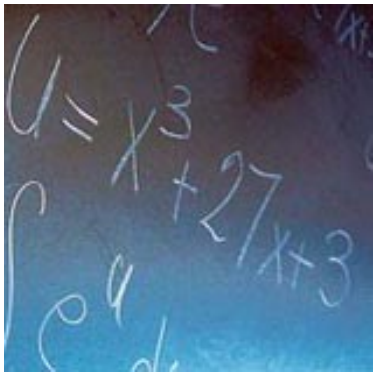
# ỨNG DỤNG THỰC TIỄN




8-puzzles




8-queens

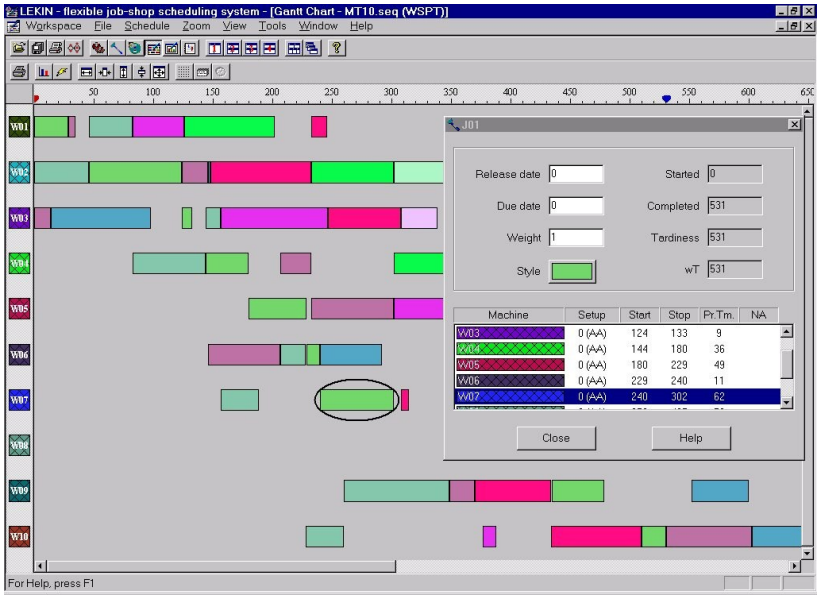


Giải toán

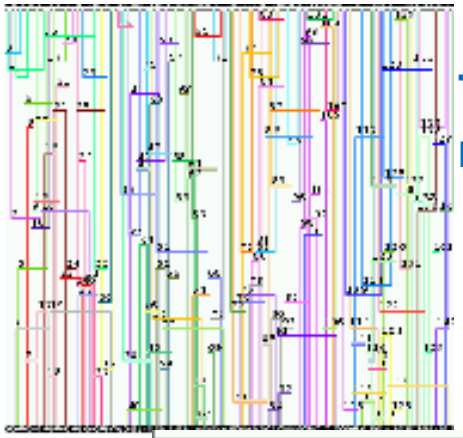
Tháp Hà Nội

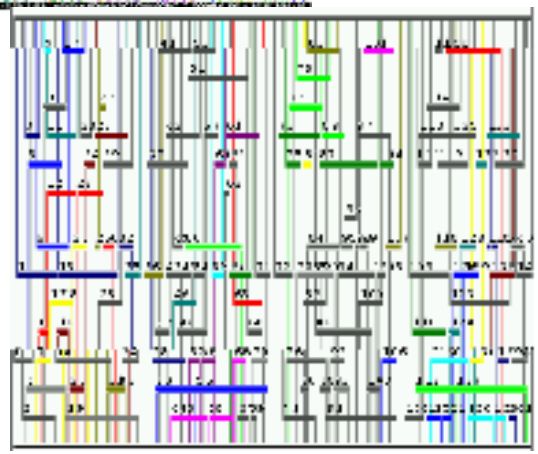
Giải câu đó

# ỨNG DỤNG THỰC TIỄN




Bài toán lập lịch





Thiết kế mạch điện

Lắp ráp tự động

# PHÁT BIỂU
# BÀI TOÁN TÌM KIẾM

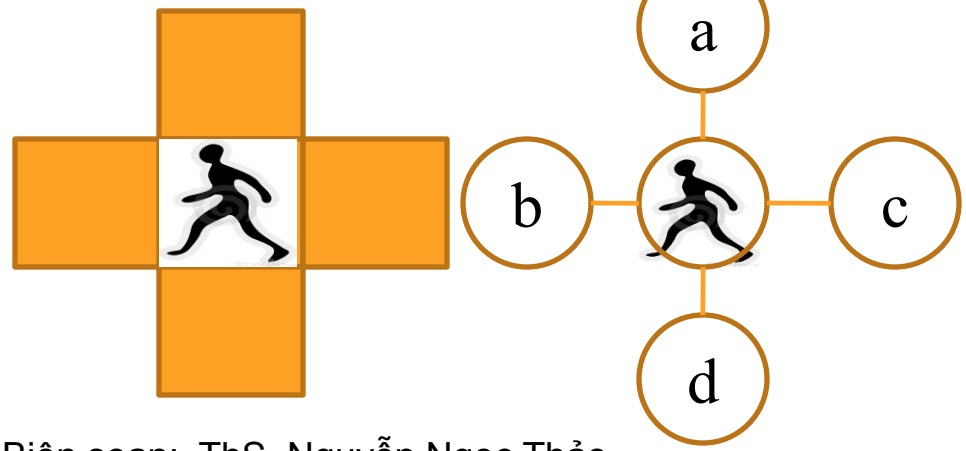
# BÀI TOÁN TÌM KIẾM

***Đề bài:*** *Giả sử bạn bị bịt mắt và thả vào giữa một mê cung. Chiến thuật (tìm kiếm) của bạn như thế nào để có thể tìm được lối ra (sẽ để biển Goal). Trong trường hợp:*

*-Bạn chỉ có một mình?*

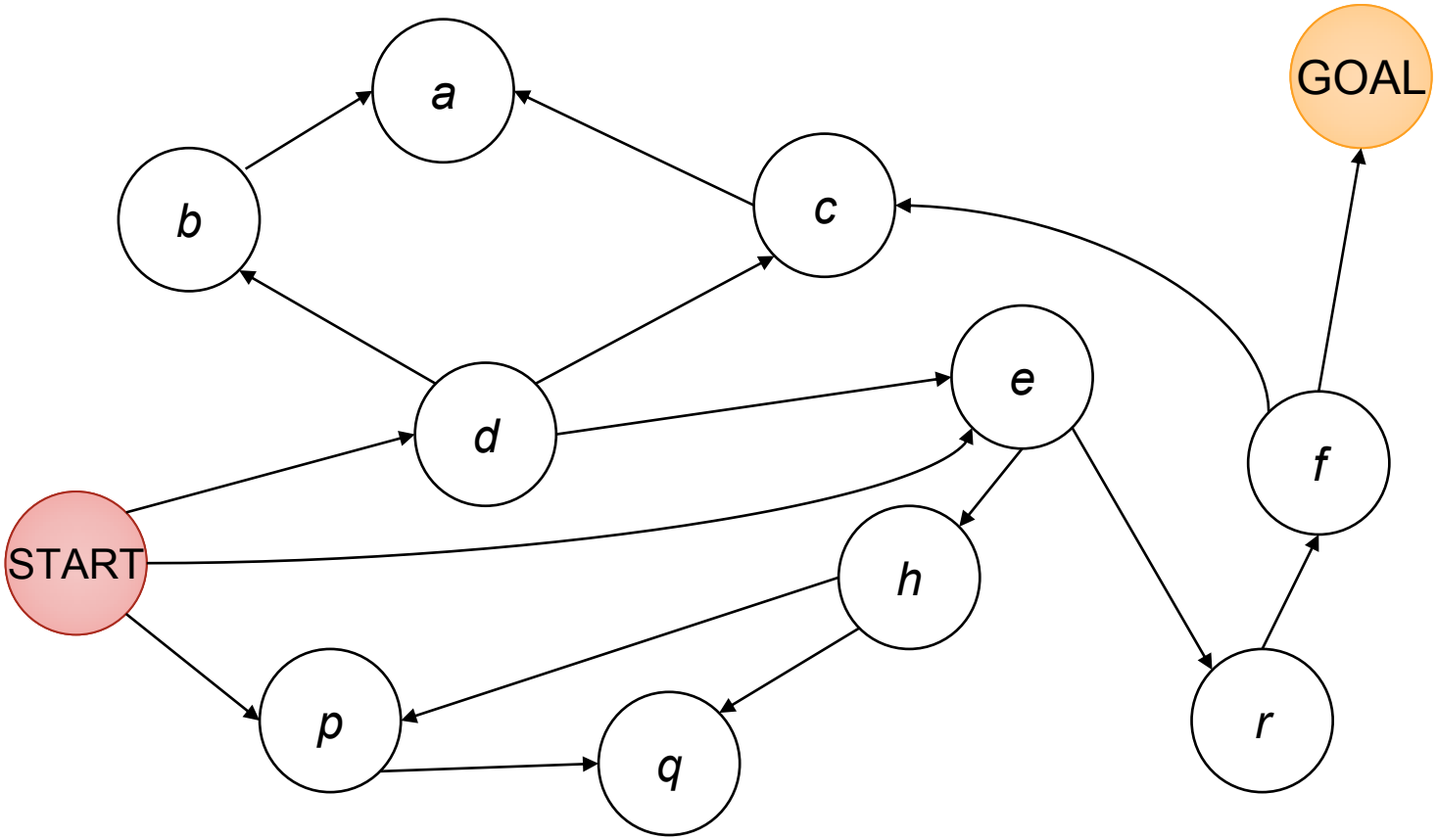
*-Bạn có một nhóm 10 người, có thể giao tiếp với nhau.*

*-Bạn biết hướng ra.*





# BÀI TOÁN TÌM KIẾM




*Làm thế nào để đi từ S đến G? Và số bước chuyển ít nhất là bao nhiêu?*

# BÀI TOÁN TÌM KIẾM

- Bài toán Tìm kiếm có năm thành phần:

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

  - Q: tập hữu hạn các trạng thái
  - $S \subseteq Q$: tập khác rỗng gồm các trạng thái ban đầu
  - $G \subseteq Q$: tập khác rỗng gồm các trạng thái đích

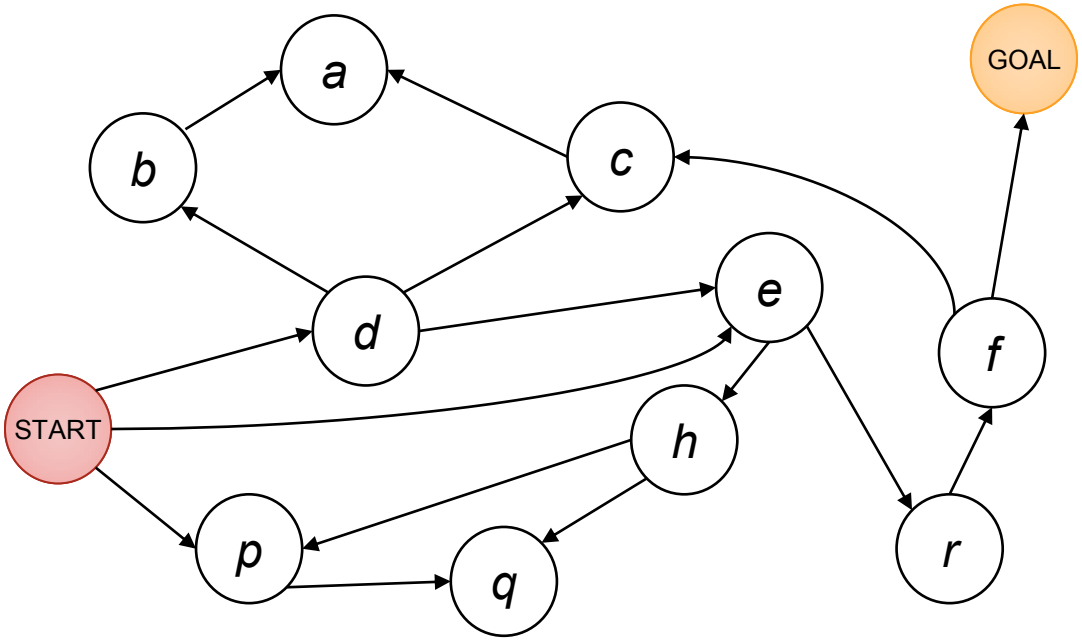
# BÀI TOÁN TÌM KIẾM

- Bài toán Tìm kiếm có năm thành phần:

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

  - **succs**(*s*): là hàm nhận một trạng thái *s* làm đầu vào và trả về một tập các trạng thái có thể đến từ s trong một bước.
  - **cost**(*s, s'*): là hàm nhận hai trạng thái *s* và *s'* làm đầu vào và trả về chi phí di chuyển một bước từ *s* đến *s'*. Hàm chi phí chỉ được định nghĩa khi *s'* là trạng thái con của *s*.

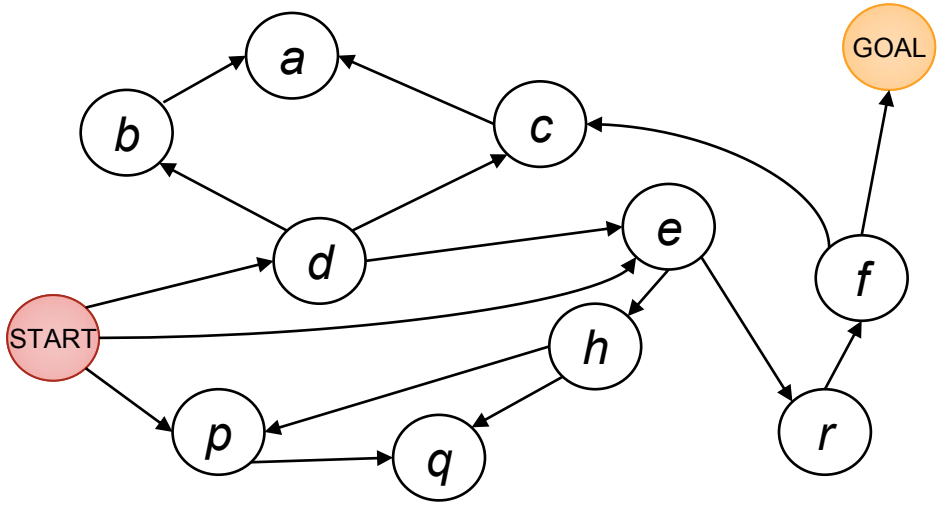
# BÀI TOÁN TÌM KIẾM




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

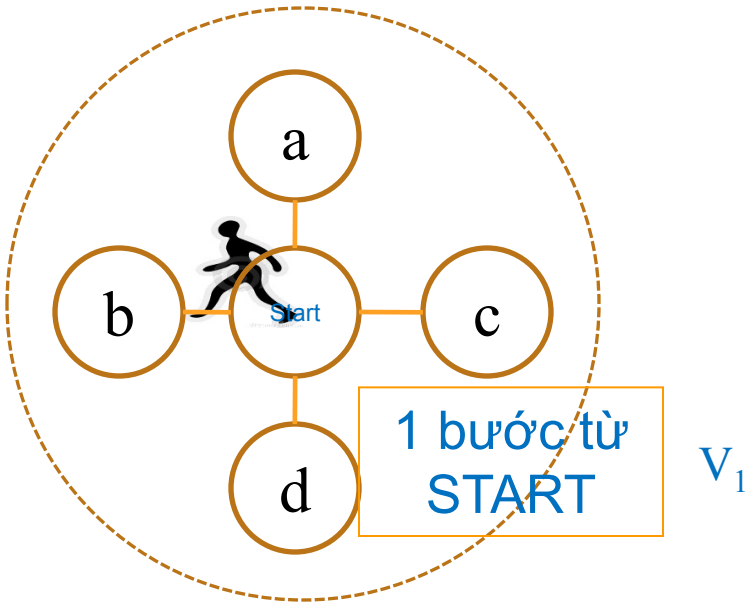
# TÌM KIẾM THEO CHIỀU RỘNG (Breadth-First Search – BFS)

# TÌM KIẾM CHIỀU RỘNG

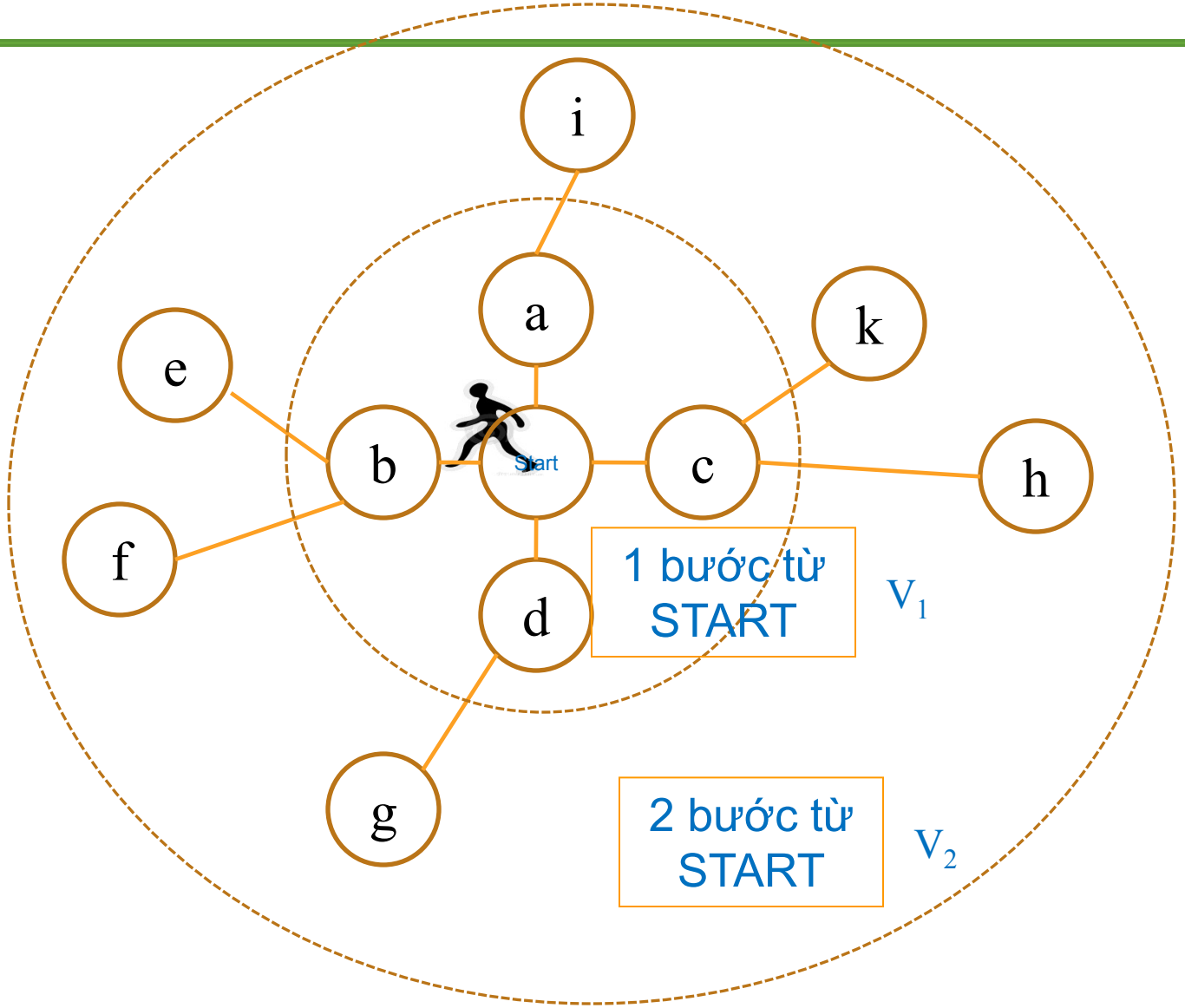



- Gán nhãn mọi trạng thái có thể đến được từ *S* trong 1 bước (nhưng không thể đến được trong ít hơn 1 bước).
- Tiếp đó gán nhãn mọi trạng thái đến được từ S trong 2 bước (nhưng không đến được trong ít hơn 2 bước).
- Tiếp đó gán nhãn cho mọi trạng thái đến được từ S trong 3 bước (nhưng không đến được trong ít hơn 3 bước)
- …cứ thế cho đến khi đi đến trạng thái Goal

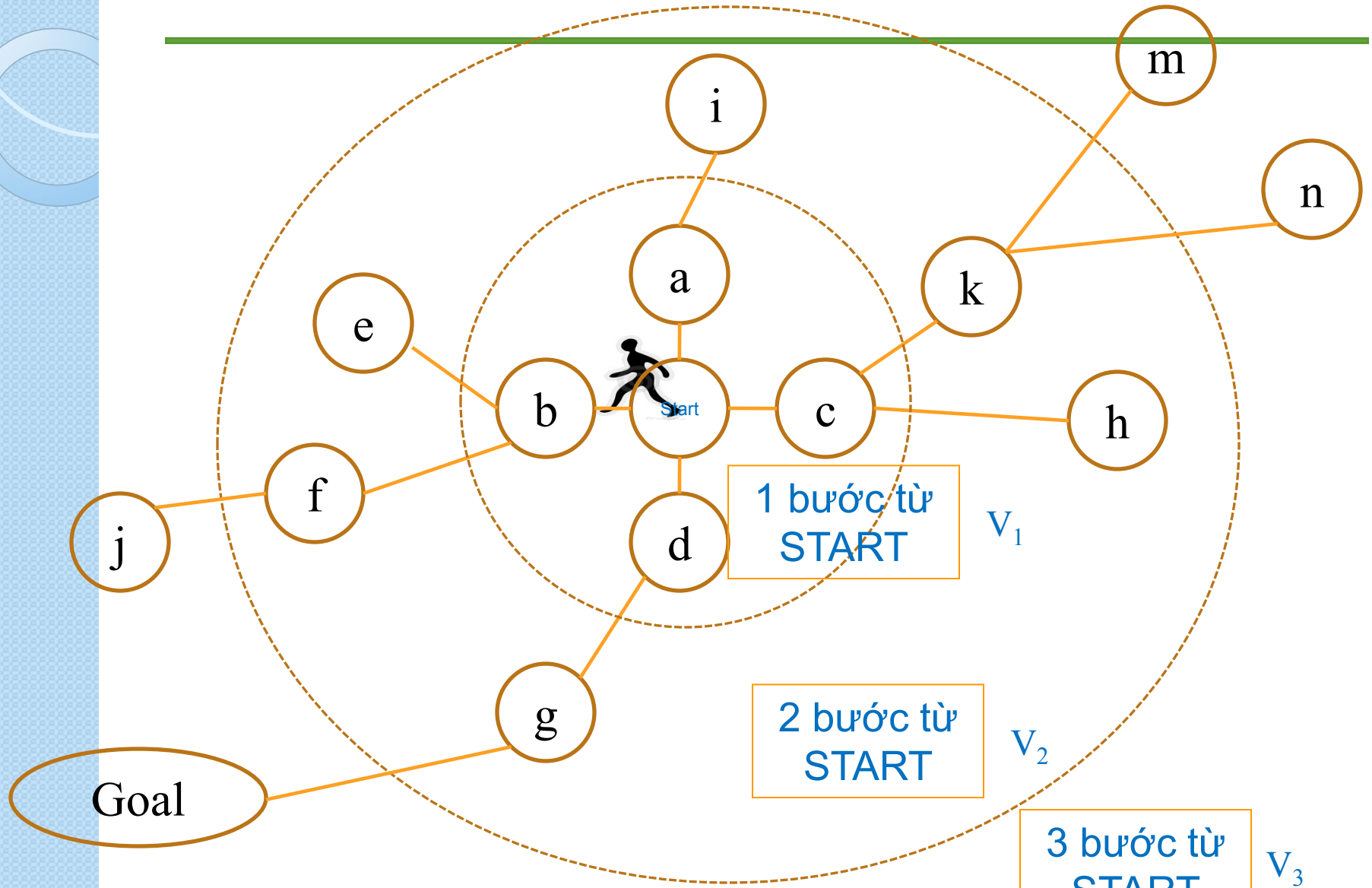
# TÌM KIẾM CHIỀU RỘNG



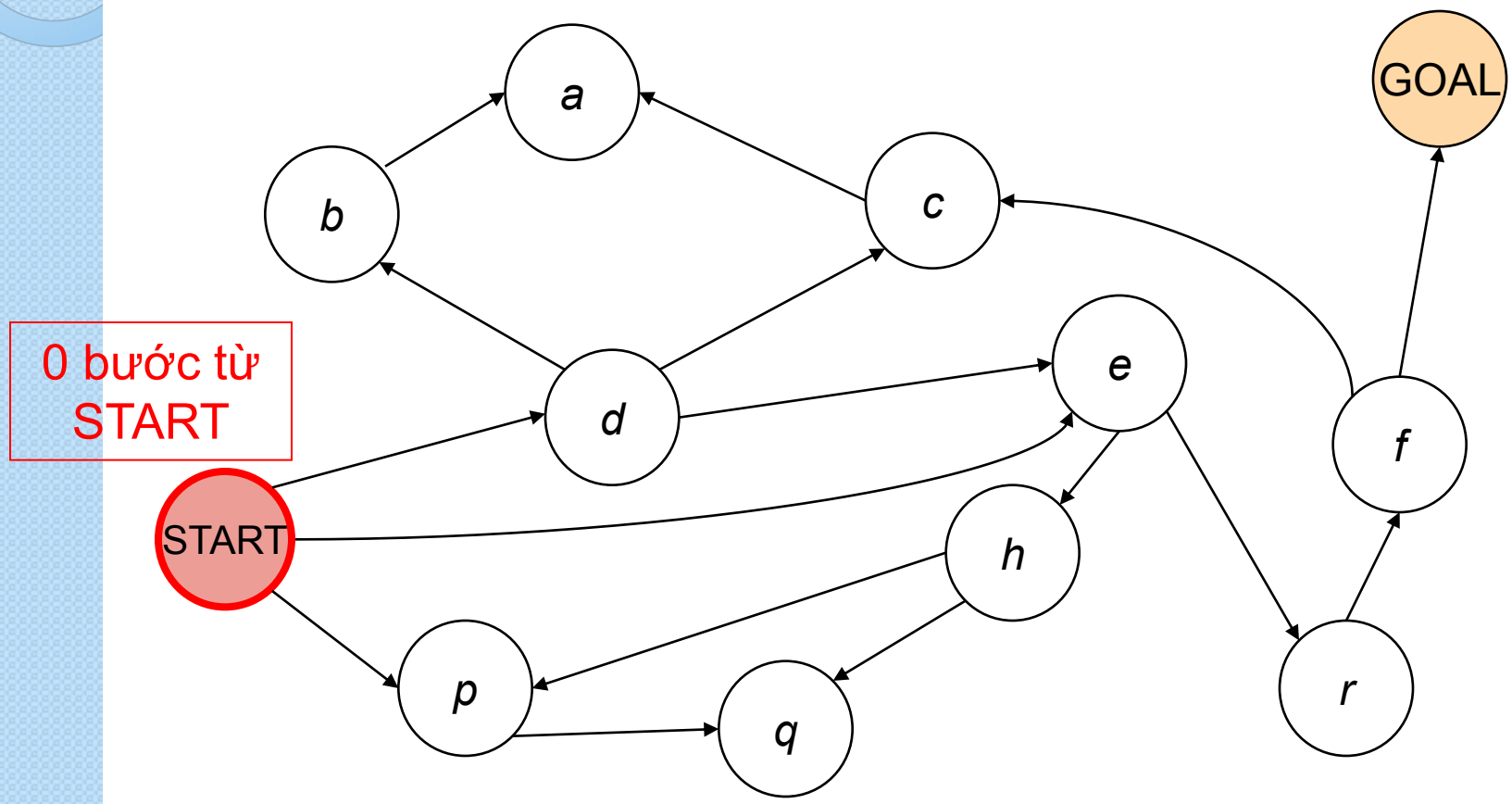
# TÌM KIẾM CHIỀU RỘNG



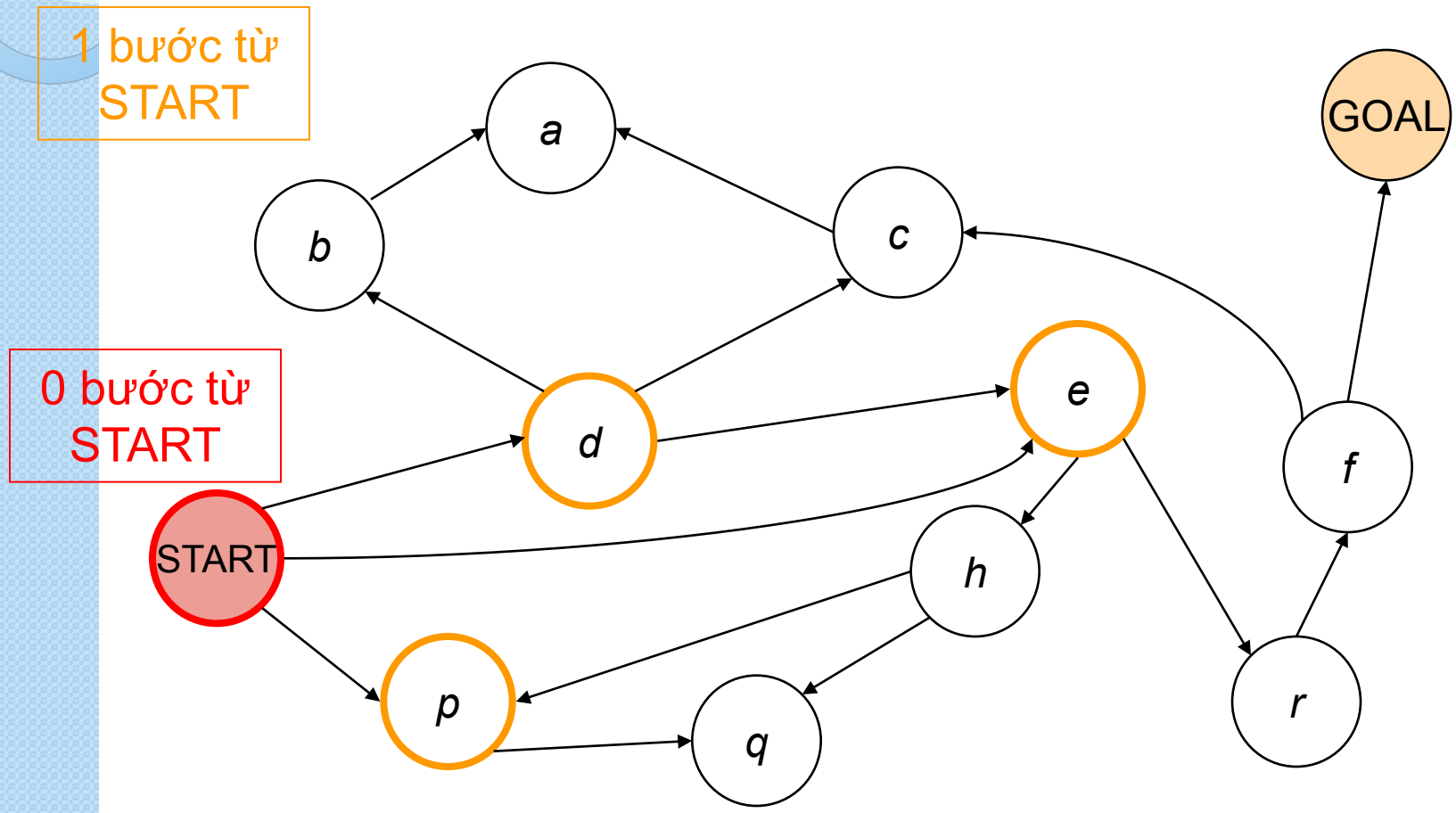
# TÌM KIẾM CHIỀU RỘNG



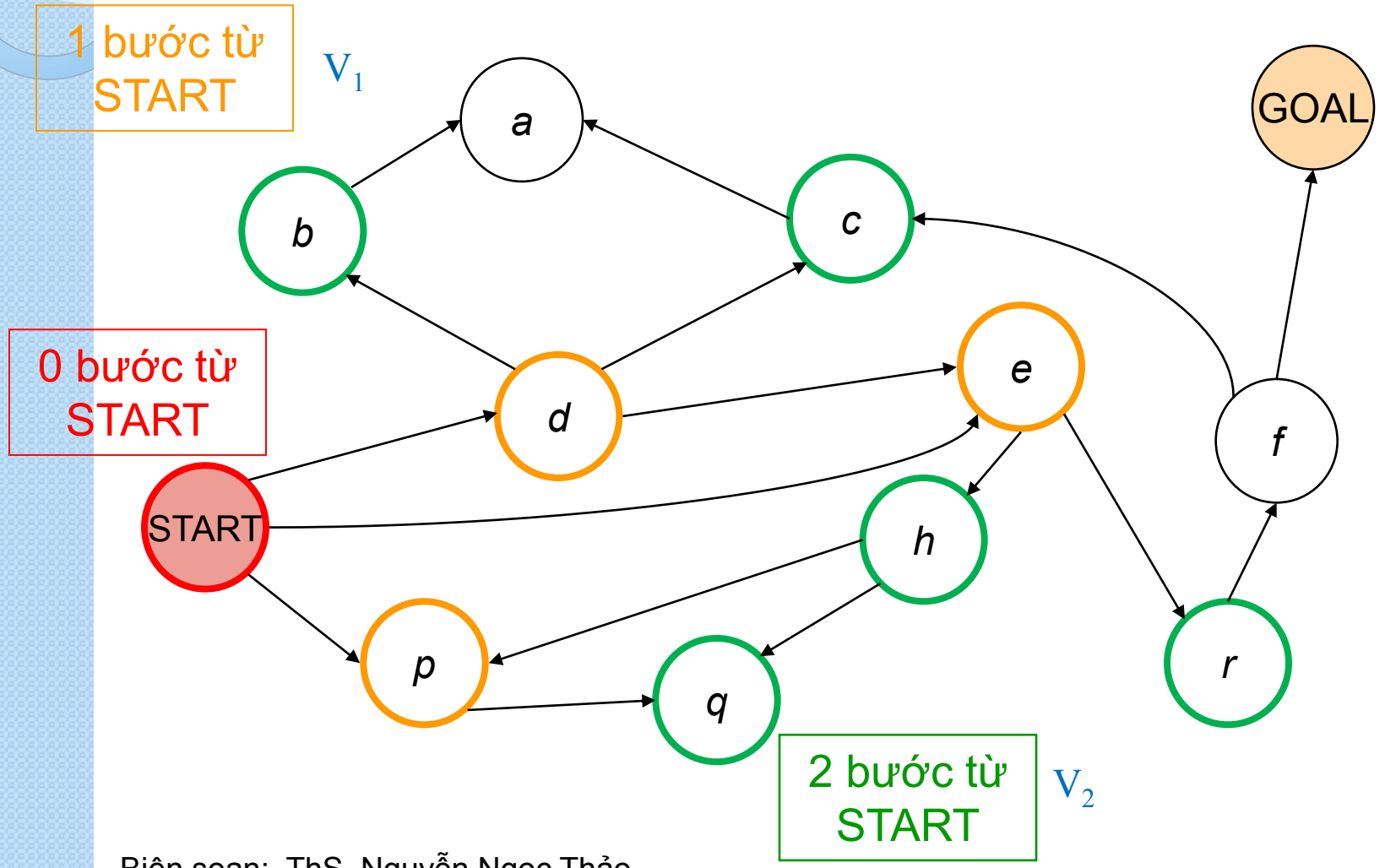
# TÌM KIẾM CHIỀU RỘNG



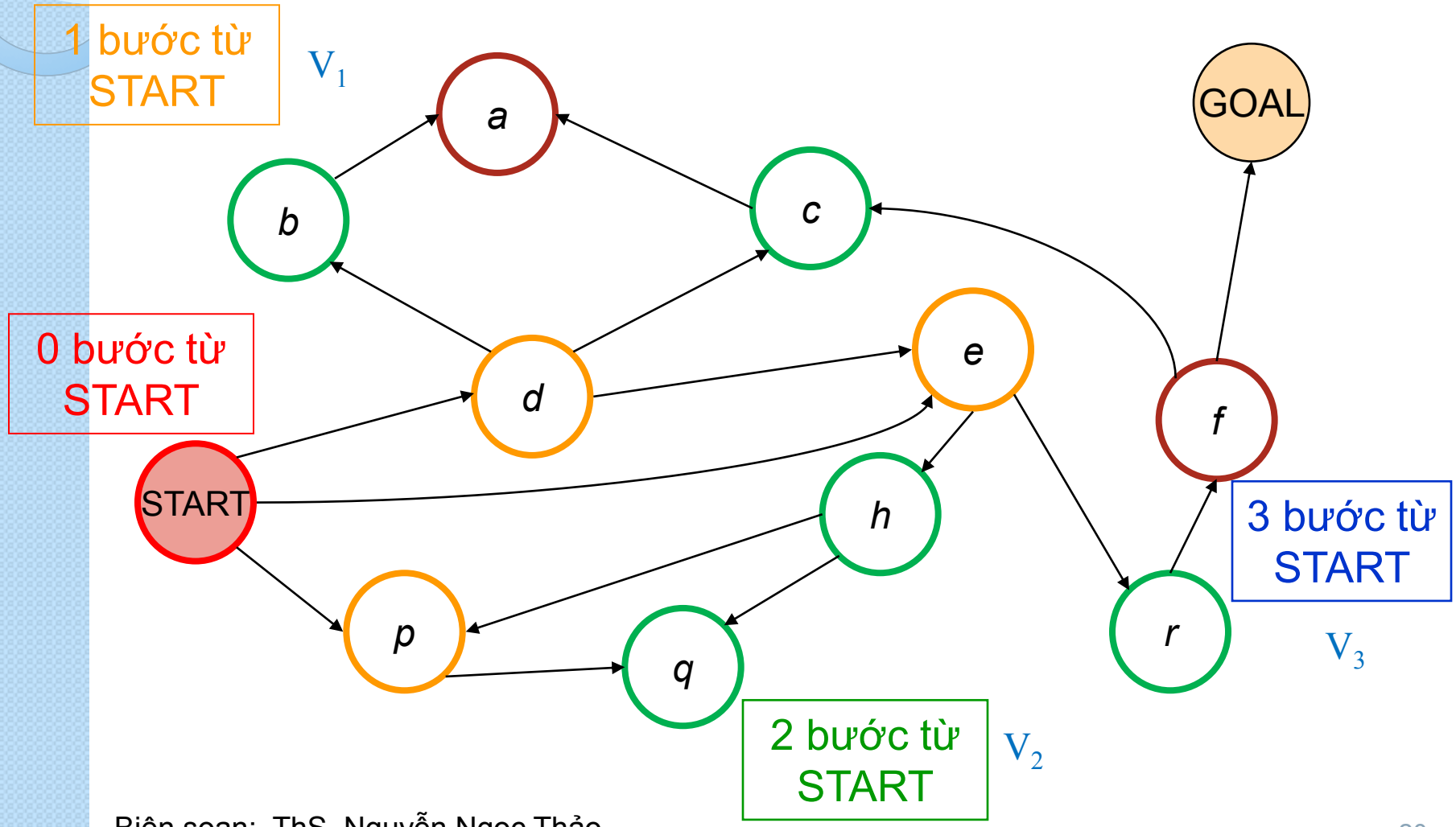
# TÌM KIẾM CHIỀU RỘNG



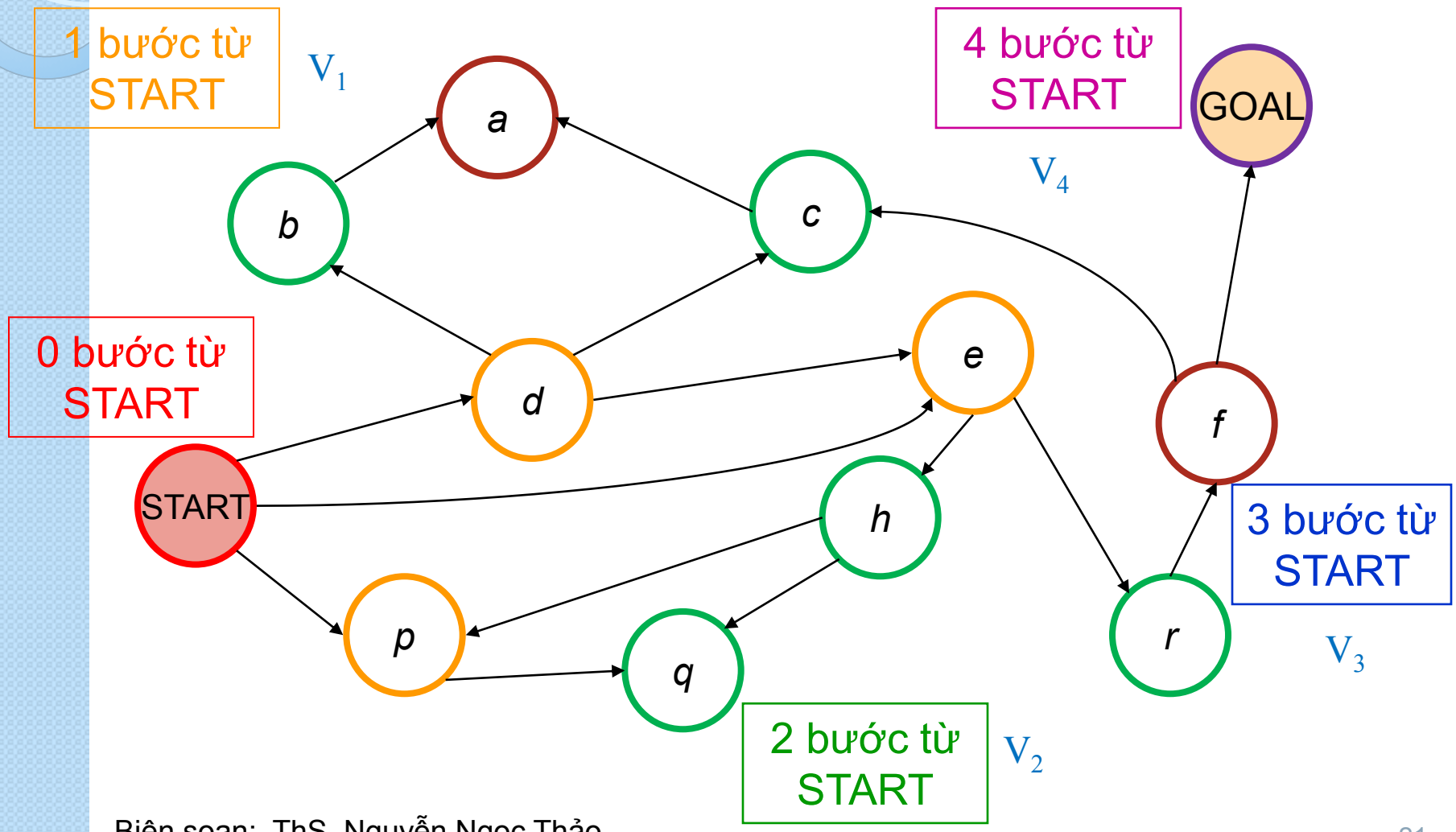
# TÌM KIẾM CHIỀU RỘNG



# TÌM KIẾM CHIỀU RỘNG



# TÌM KIẾM CHIỀU RỘNG

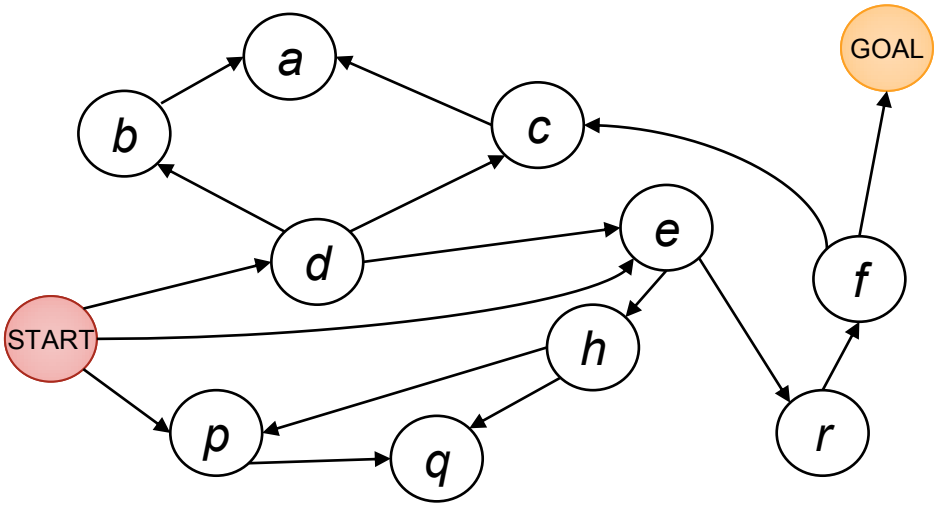


# ĐƯỜNG ĐI?

- **Nếu như bạn cần hướng dẫn thành viên khác ra khỏi mê cung.**

  **→ Vậy đường đi như thể nào?**

# GHI NHỚ ĐƯỜNG ĐI

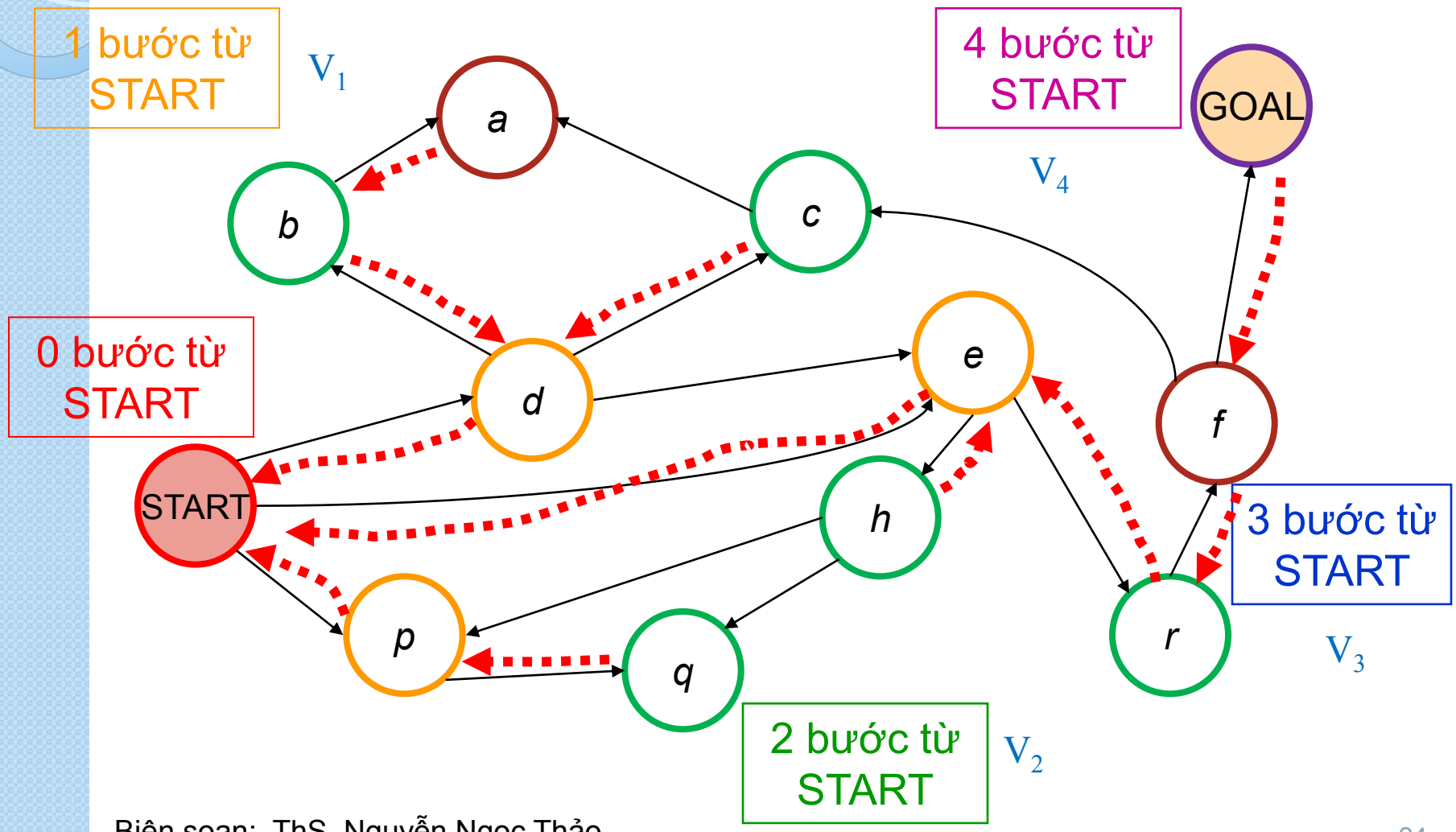



- Khi gán nhãn một trạng thái, cần ghi nhận trạng thái trước đó. Ghi nhận này được gọi là *con trỏ quay lui* (backpointer). Lịch sử các trạng thái trước dùng để phát sinh lời giải đường đi khi đã tìm thấy đích.

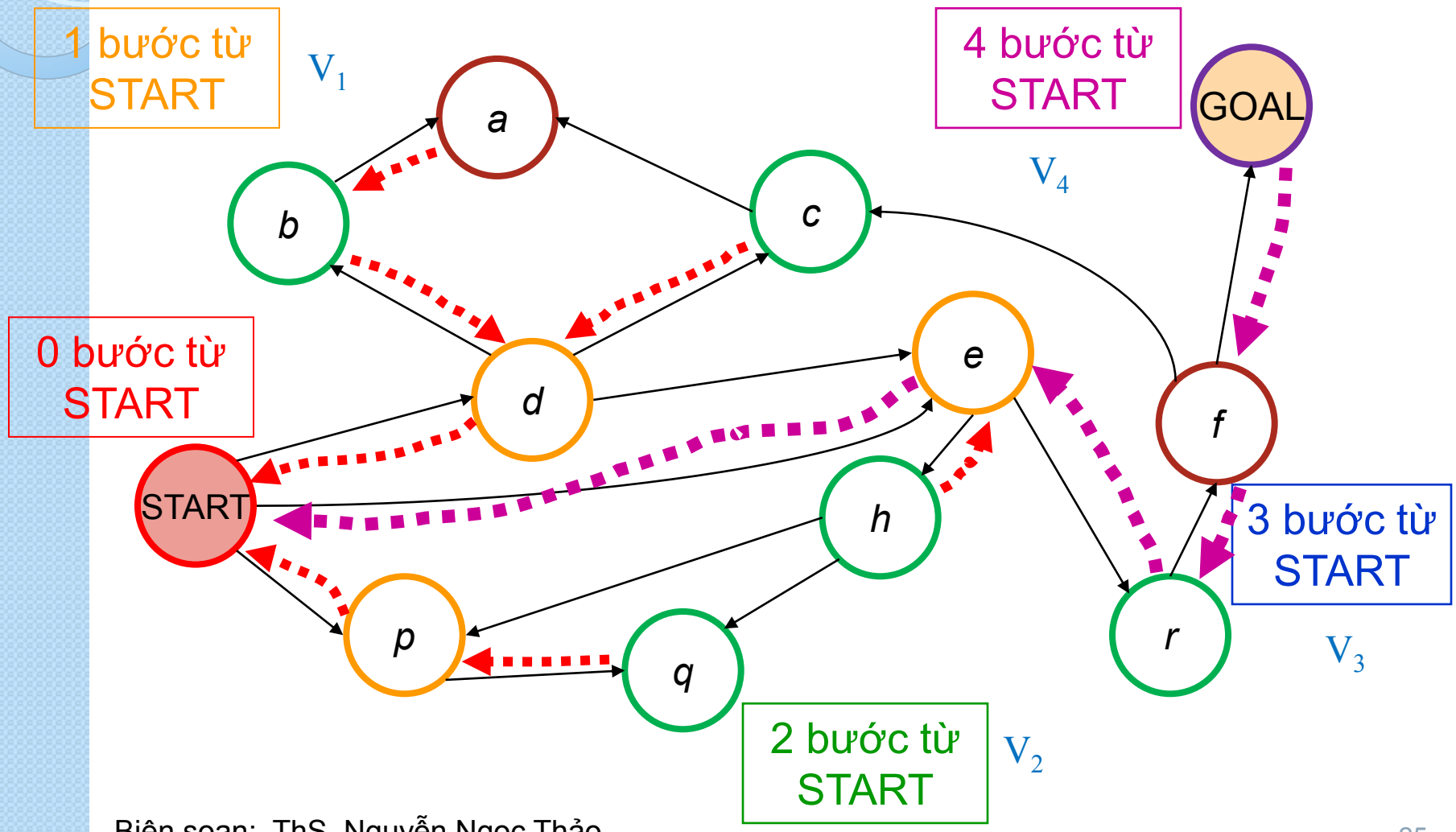
  "Tôi đã đến đích. Tôi thấy mình đã ở f trước khi đến đích. Và tôi đã ở r trước khi ở f. Và tôi…

  … vậy lời giải đường đi là $S \rightarrow e \rightarrow r \rightarrow f \rightarrow G$"

# TÌM KIẾM CHIỀU RỘNG



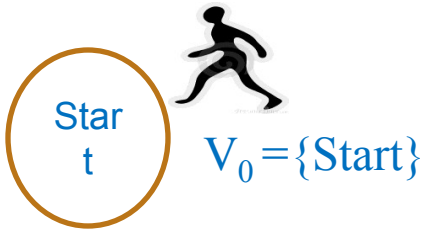
# TÌM KIẾM CHIỀU RỘNG



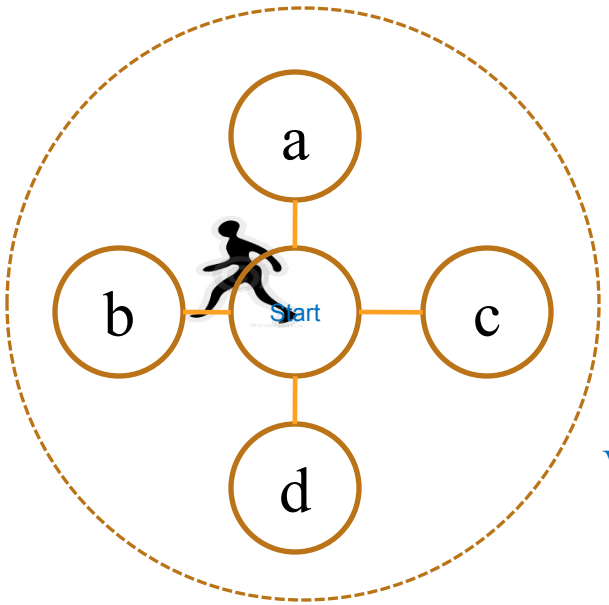
# TÌM KIẾM CHIỀU RỘNG

- Với trạng thái *s* bất kì đã gán nhãn:
  - *previous*(*s*) là trạng thái trước đó trên đường đi ngắn nhất từ trạng thái START đến *s*.
  - Tại vòng lặp thứ *k* của thuật toán, bắt đầu với $V_k$, là tập các trạng thái mà đường đi ngắn nhất từ START đi đến chúng có đúng *k* bước.
  - Tiếp đó, trong vòng lặp, tính $V_{k+1}$, là tập các trạng thái mà đường đi ngắn nhất từ START có đúng *k*+1 bước.
  - Ta bắt đầu với $k = 0$, $V_0 = \{START\}$ và định nghĩa *previous*(START) = NULL
  - Tiếp đó, thêm vào những trạng thái đi một bước từ START vào $V_1$. Và cứ thế tiếp diễn.

# Xây $V_{k+1}$ từ $V_k$

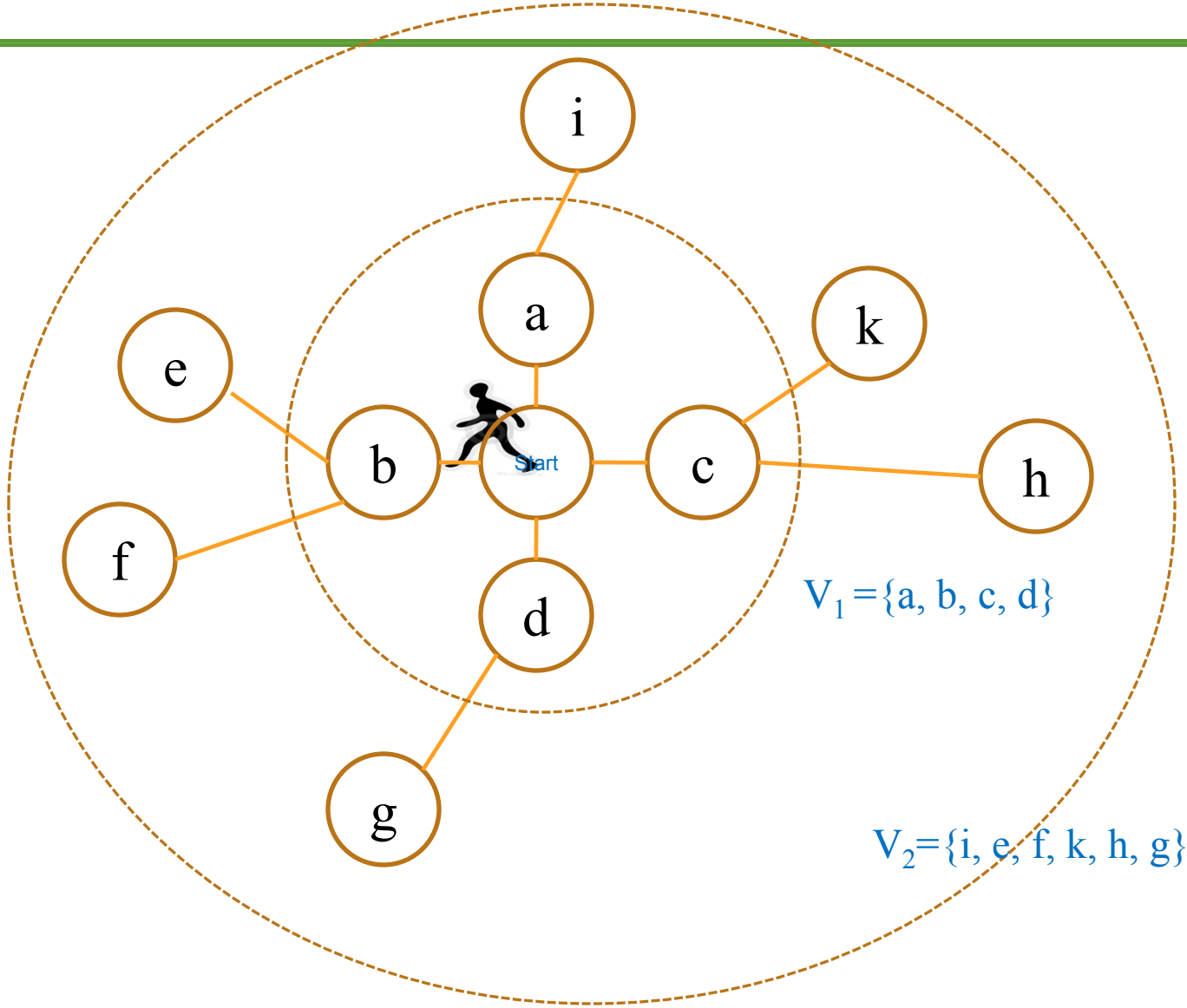


# Xây $V_{k+1}$ từ $V_k$

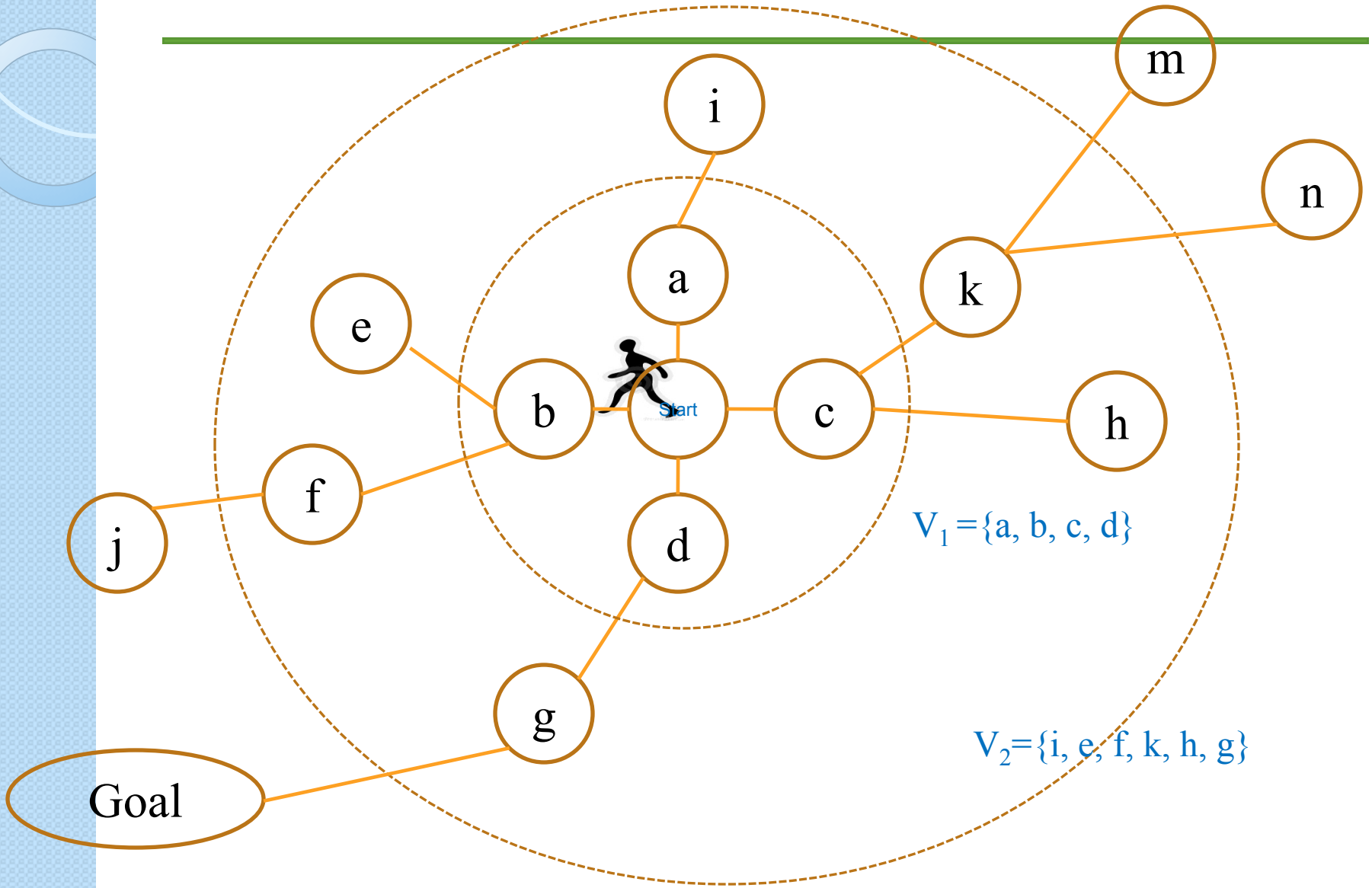



$V_1 = \{a, b, c, d\}$

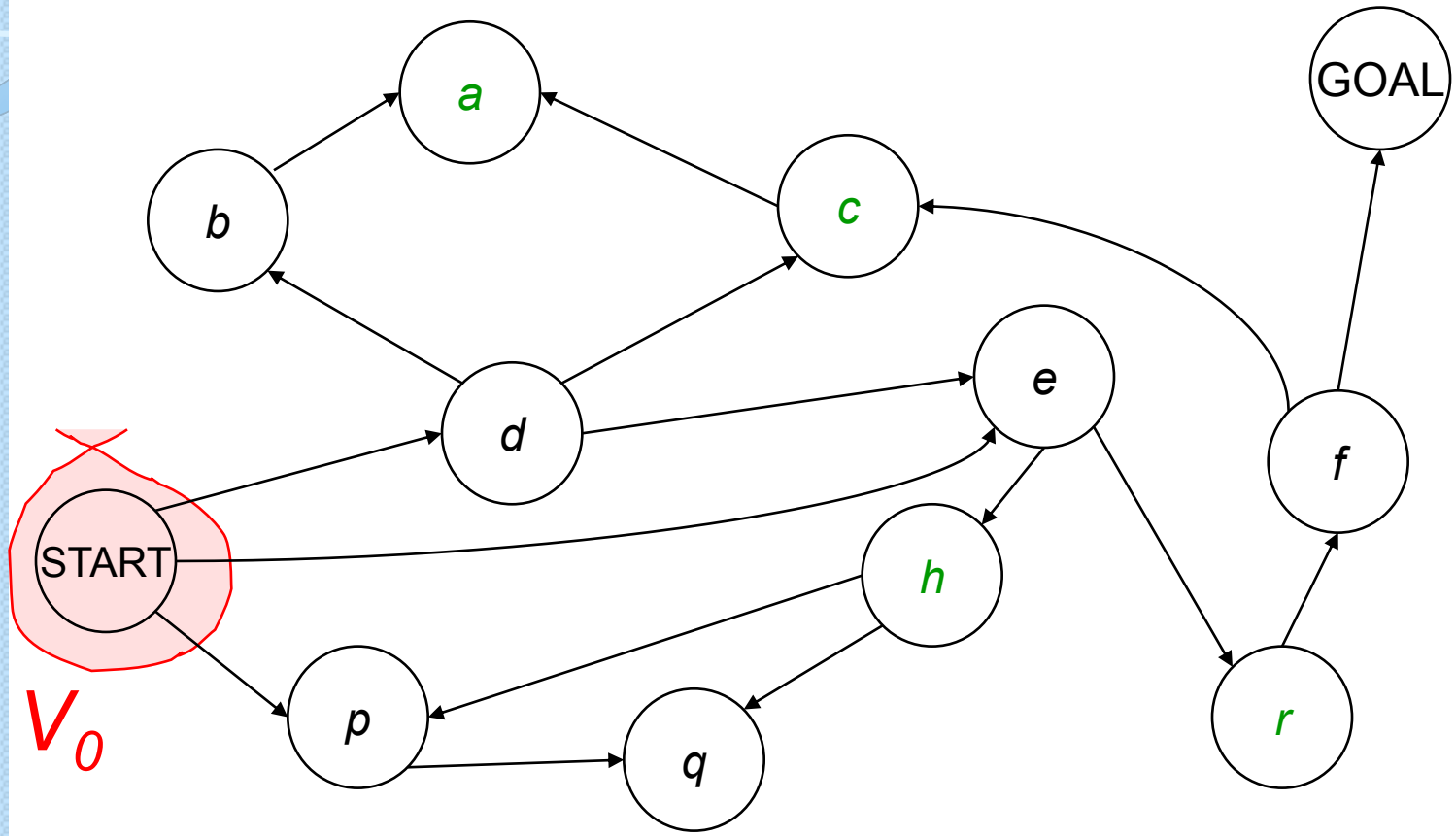
# TÌM KIẾM CHIỀU RỘNG



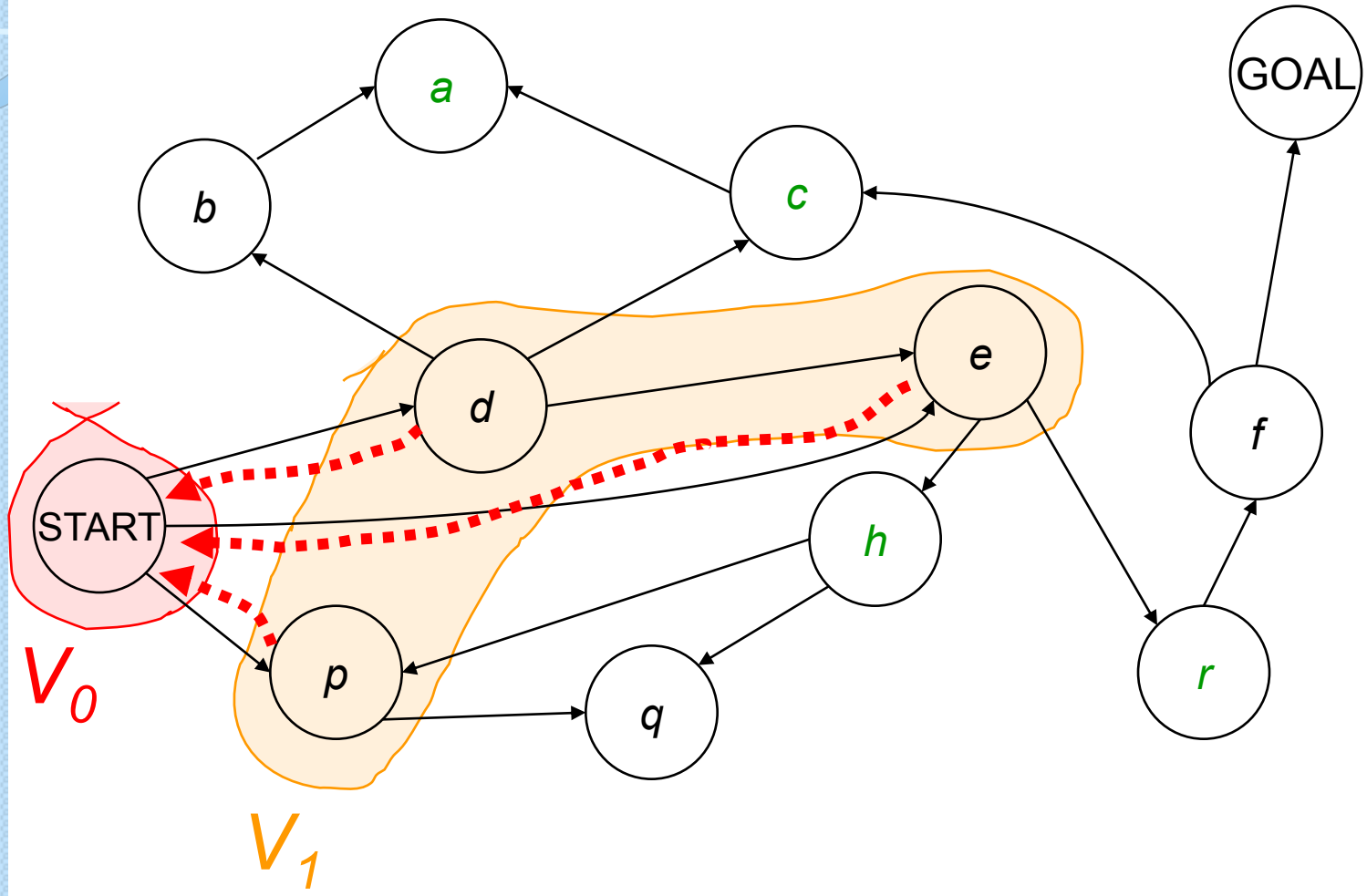
# TÌM KIẾM CHIỀU RỘNG



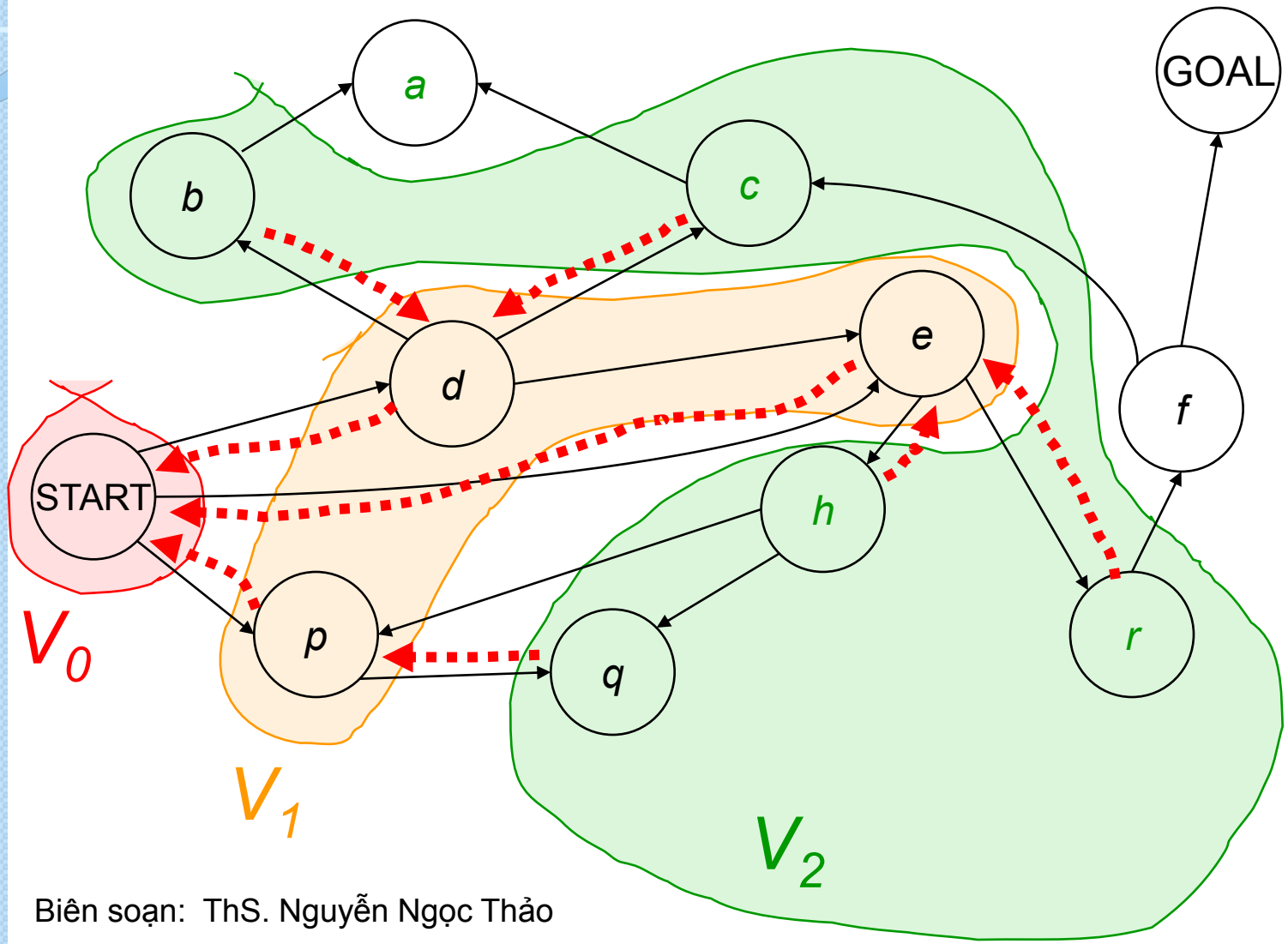
# TÌM KIẾM CHIỀU RỘNG



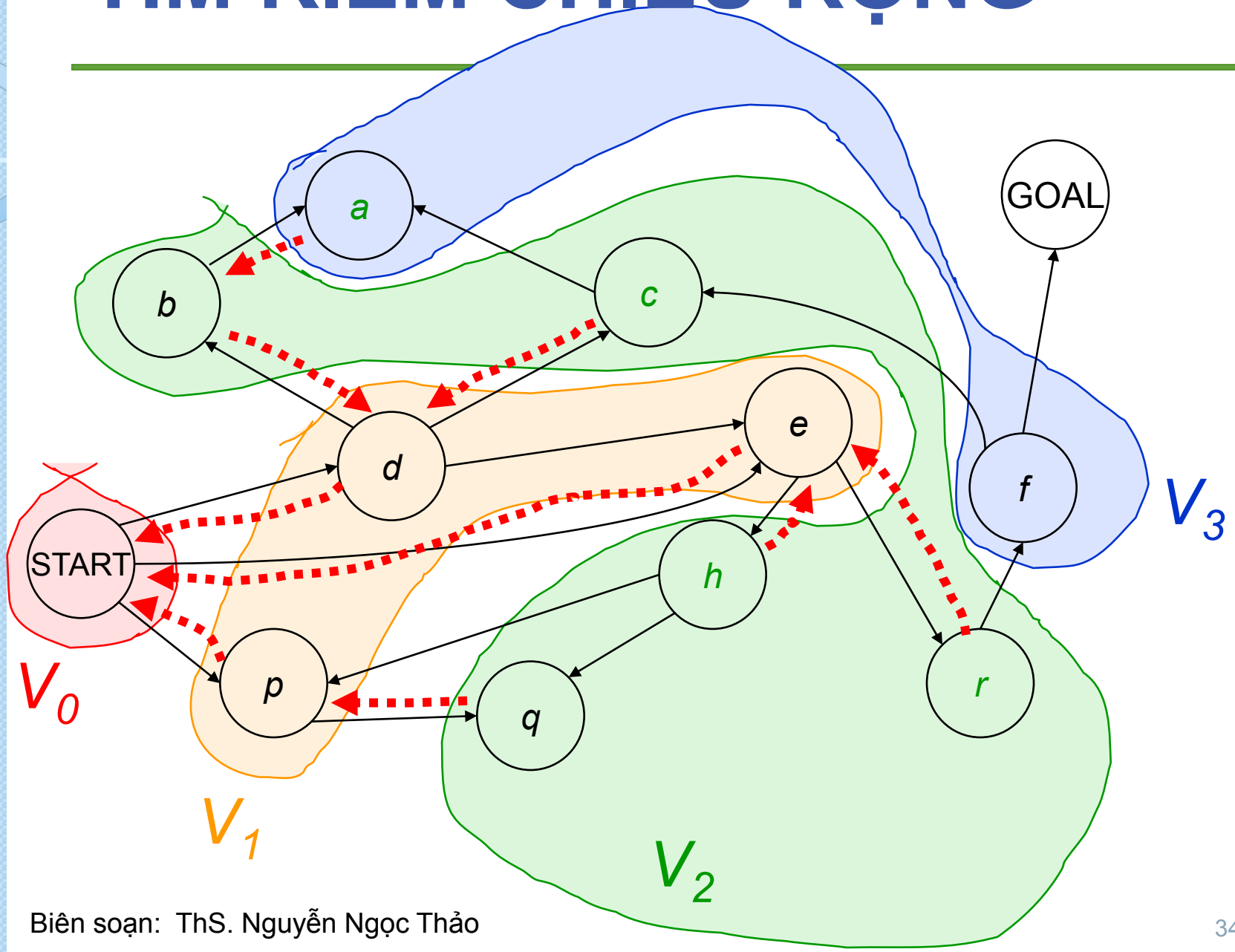
# TÌM KIẾM CHIỀU RỘNG



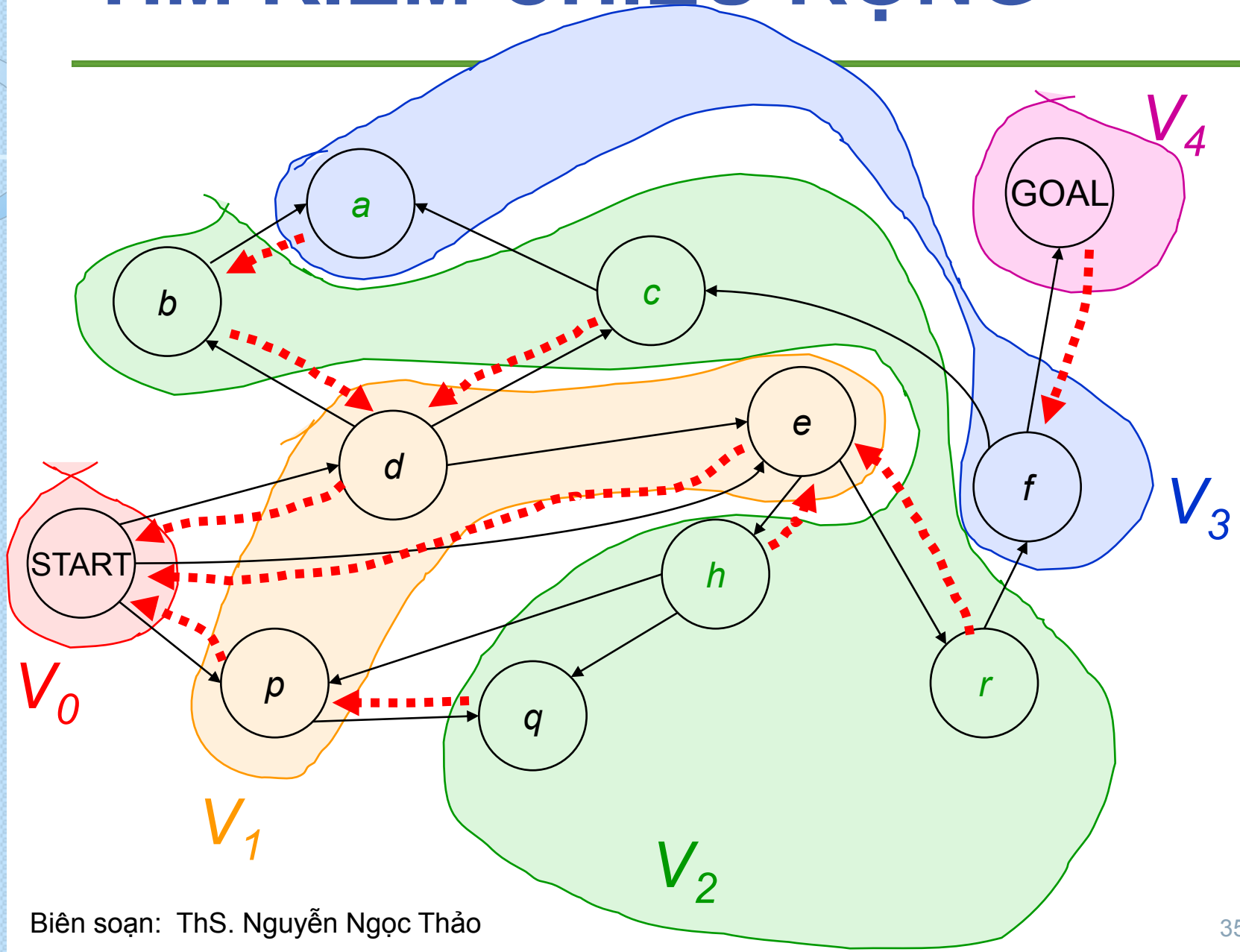
# TÌM KIẾM CHIỀU RỘNG



# TÌM KIẾM CHIỀU RỘNG



# TÌM KIẾM CHIỀU RỘNG



# MÔ TẢ THUẬT TOÁN

k:

…

Với mỗi trạng thái *s* trong $V_k$

Với mỗi trạng thái *s'* trong **succs**(*s*)

Nếu *s'* chưa được gán nhãn

…

Thêm *s'* vào $V_{k+1}$

…

# MÔ TẢ THUẬT TOÁN

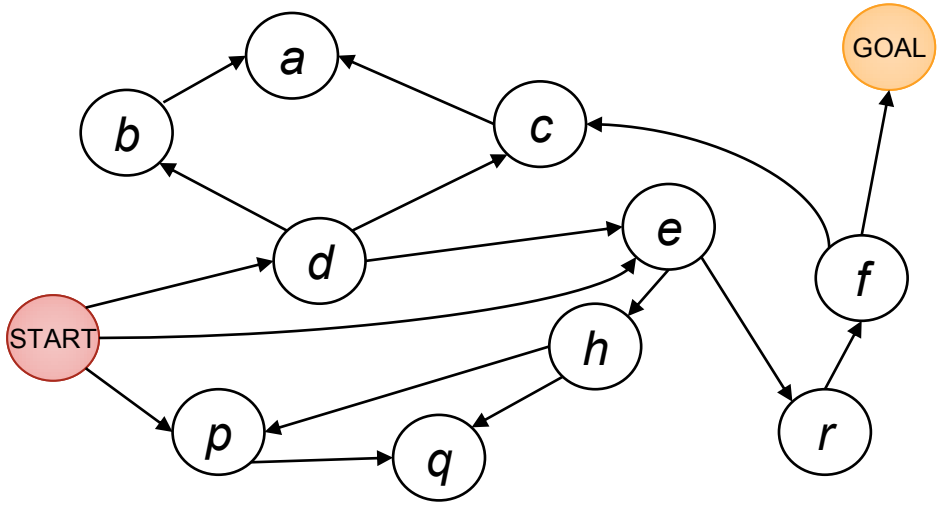
$V_0$ := $S$ (tập các trạng thái khởi đầu)
*previous*(*START*) := *NIL*
$k$ := *0*
**while** (không có trạng thái đích trong $V_k$ và $V_k$ không rỗng) **do**
    $V_{k+1}$ := tập rỗng
    Với mỗi trạng thái $s$ trong $V_k$
        Với mỗi trạng thái $s'$ trong **succs**($s$)
            Nếu $s'$ chưa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

# TÌM KIẾM CHIỀU RỘNG


Giả sử không gian tìm kiếm cho phép ta lấy trạng thái trước đó một cách dễ dàng.

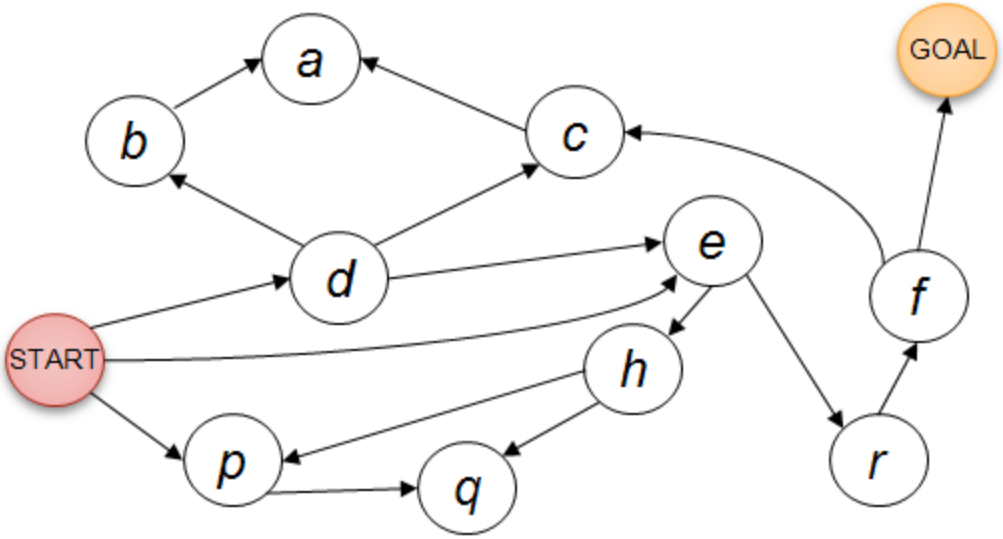
- Có cách nào khác để thực hiện BFS không?
- Và ta có thể tránh việc lưu trữ những gì mà ta đã từng lưu?

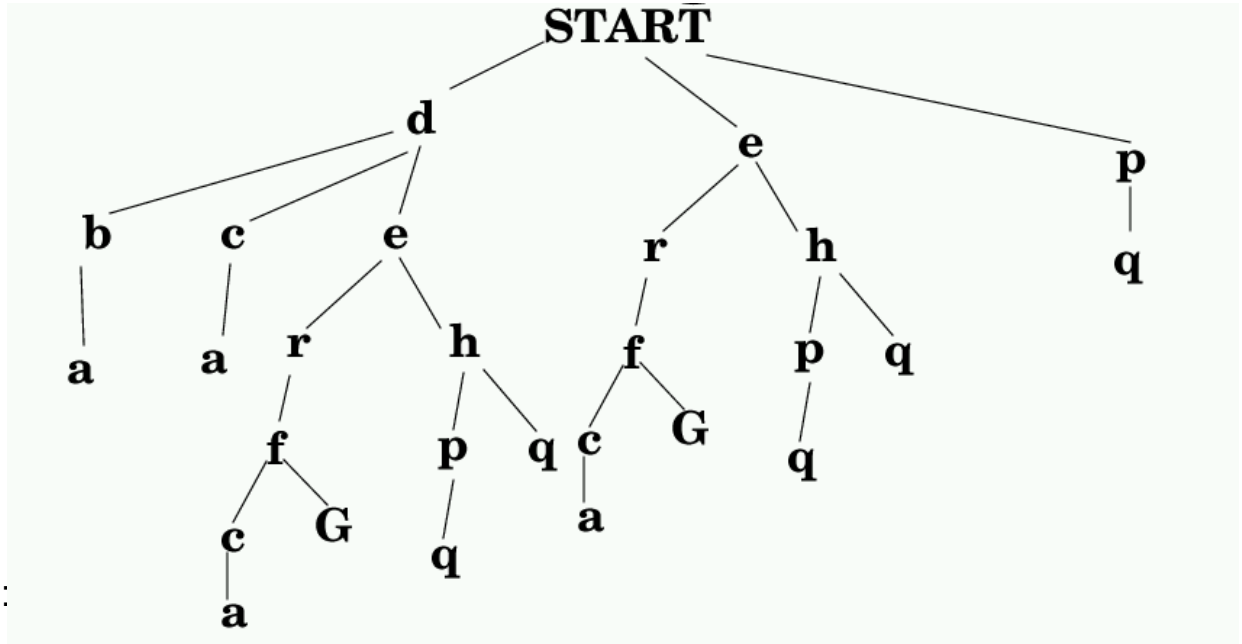
# MỘT CÁCH KHÁC: ĐI LUI




- Gán nhãn mọi trạng thái có thể đi đến *G* trong 1 bước nhưng không thể đi đến trong ít hơn 1 bước.
- Gán nhãn mọi trạng thái có thể đi đến *G* trong 2 bước nhưng không thể đi đến trong ít hơn 2 bước.
- … cho đến khi đi đến trạng thái START
- Các nhãn "số bước đến đích" xác định đường đi ngắn nhất. Không cần thêm thông tin lưu trữ.

# CÂY TÌM KIẾM - BFS



# MỘT SỐ NHẬN XÉT

| Depth | Nodes | Time | Memory |
|---|---|---|---|
| 2 | 110 | .11 milliseconds | 107 kilobyes |
| 4 | 11,110 | 11 milliseconds | 10.6 megabytes |
| 6 | $10^6$ | 1.1 seconds | 1 gigabytes |
| 8 | $10^8$ | 2 minutes | 103 gigabytes |
| 10 | $10^{10}$ | 3 hours | 10 terabytes |
| 12 | $10^{12}$ | 13 days | 1 petabytes |
| 14 | $10^{14}$ | 3.5 years | 99 petabytes |
| 16 | $10^{16}$ | 350 years | 10 exabytes |

Time and memory requirements for breadth-first search. The numbers shown assume branching factor b = 10; 1 million nodes/second; 1000 bytes/node.

Page 83, chapter 3 - S. Russel and P. Norvig, Artificial Intelligence – A Modern Approach. *Third Edition.* 2010.

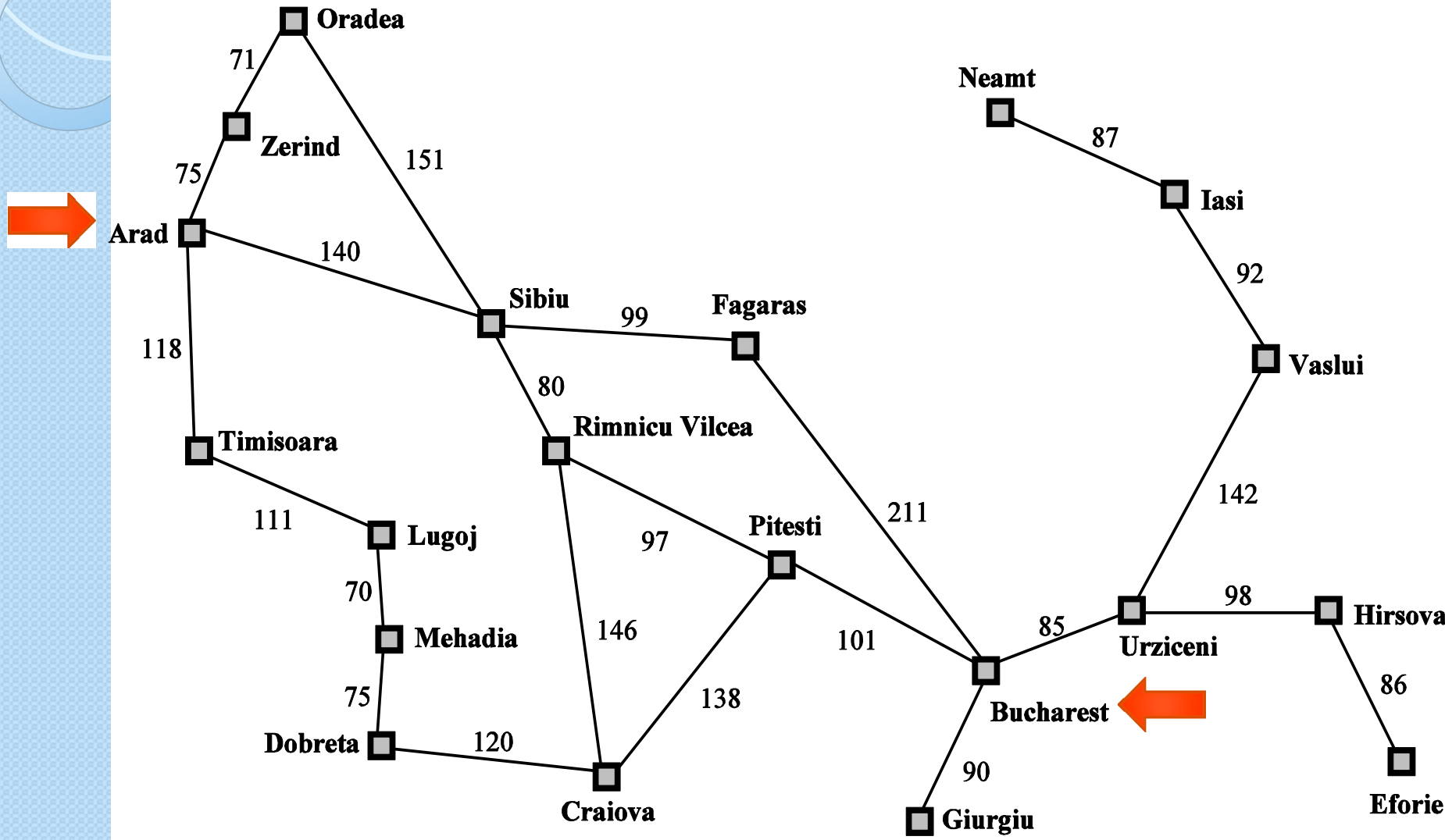
# MỘT SỐ NHẬN XÉT

- Vẫn chạy tốt khi có nhiều hơn một trạng thái đích hoặc nhiều hơn một trạng thái bắt đầu.
- Thuật toán thực hiện tiến tới từ trạng thái bắt đầu. Những thuật toán thực hiện tiến tới từ trạng thái bắt đầu gọi là *suy diễn tiến*.
- Thuật toán này rất giống với thuật toán Dijkstra's.
- Có thể thực hiện lùi dần từ đích. Thuật toán thực hiện lùi dần từ đích gọi là *suy diễn lùi*.
- Lùi và tiến. Phương pháp nào tốt hơn?

# BÀI TẬP VÍ DỤ

- Cho bản đồ các thành phố ở Rumani (slide kế tiếp), một người muốn đi du lịch từ Arad đến Bucharest bằng đường bộ. Hãy xác định lộ trình qua các thành phố sao cho số thành phố phải đi qua là ít nhất.

# BÀI TẬP VÍ DỤ


Oradea
71
Zerind
75
151
Arad
140
Sibiu
99
Fagaras
118
80
Rimnicu Vilcea
Timisoara
111
Lugoj
97
Pitesti
211
70
Mehadia
146
101
75
138
Dobreta
120
Craiova
90
Giurgiu
Bucharest
85
Urziceni
98
Hirsova
86
Eforie
142
Vaslui
92
Iasi
87
Neamt

# BÀI TẬP VÍ DỤ

- $V_0$ = {Arad}
- $V_1$ = {Sibiu, Timisoara, Zerind}
- $V_2$ = {Faragas, Rimnicu Vilcea, Lugoj, Oradea}
- $V_3$ = {**Bucharest**, Pitesti, Craiova, Mehadia, ~~Sibiu~~}

Đã tìm thấy Goal, dừng thuật toán.

Đường đi là: Arad $\rightarrow$ Sibiu $\rightarrow$ Faragas $\rightarrow$ Bucharest

Chi phí: 450

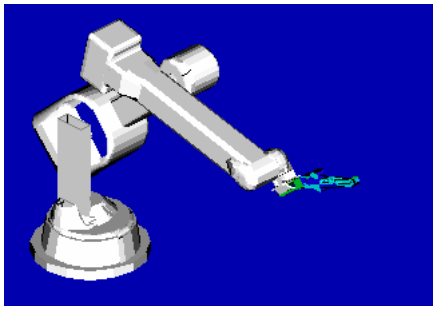
* Sibiu đã có trong $V_1$ thì không đưa vào $V_3$ nữa.

# KHÔNG ÁP DỤNG TÌM KIẾM

Trò chơi đối kháng

Yếu tố ngẫu nhiên


Trạng thái ẩn



Trạng thái vô hạn

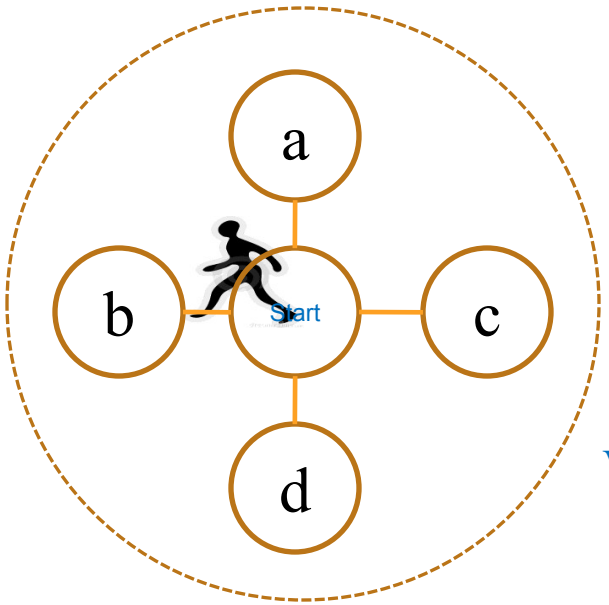
Yếu tố đồng đội

# Tìm đường tp HCM


**Tìm đường ngắn nhất từ nhà bạn đến trường**

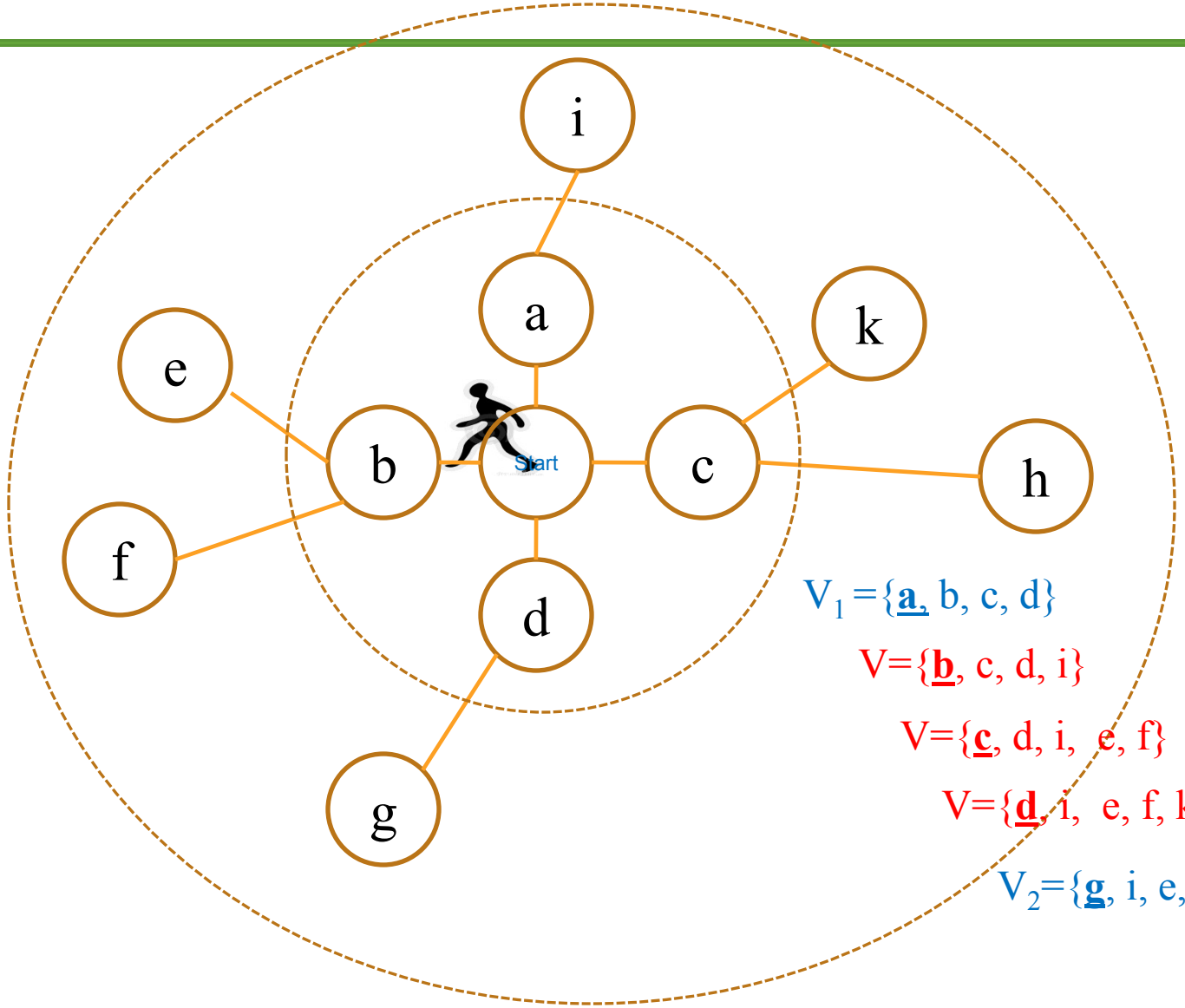
Biên soạn: ThS. Nguyễn Ngọc Thảo

# Xây $V_{k+1}$ từ $V_k$




$V_1 = \{a, b, c, d\}$

# TÌM KIẾM CHIỀU RỘNG



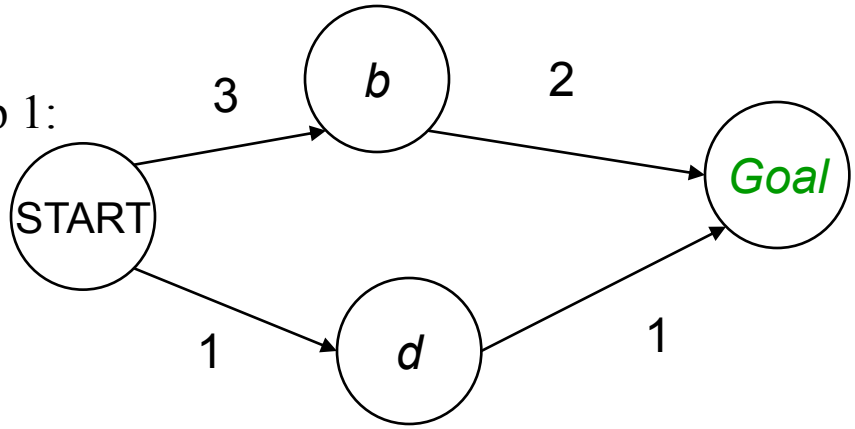
# TÌM KIẾM THEO CHIỀU RỘNG VỚI CHI PHÍ THẤP NHẤT

## (Least Cost Breadth-First Search – LCBFS)

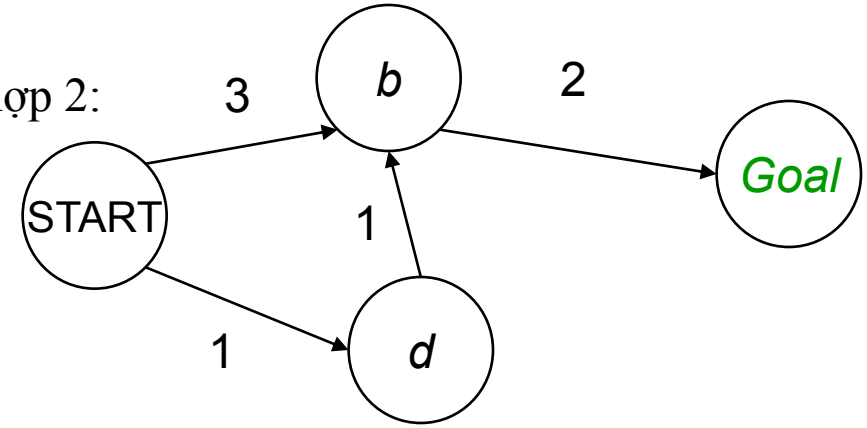
Biên soạn: ThS. Nguyễn Ngọc Thảo

# BFS có tìm đđi ít chi phí?

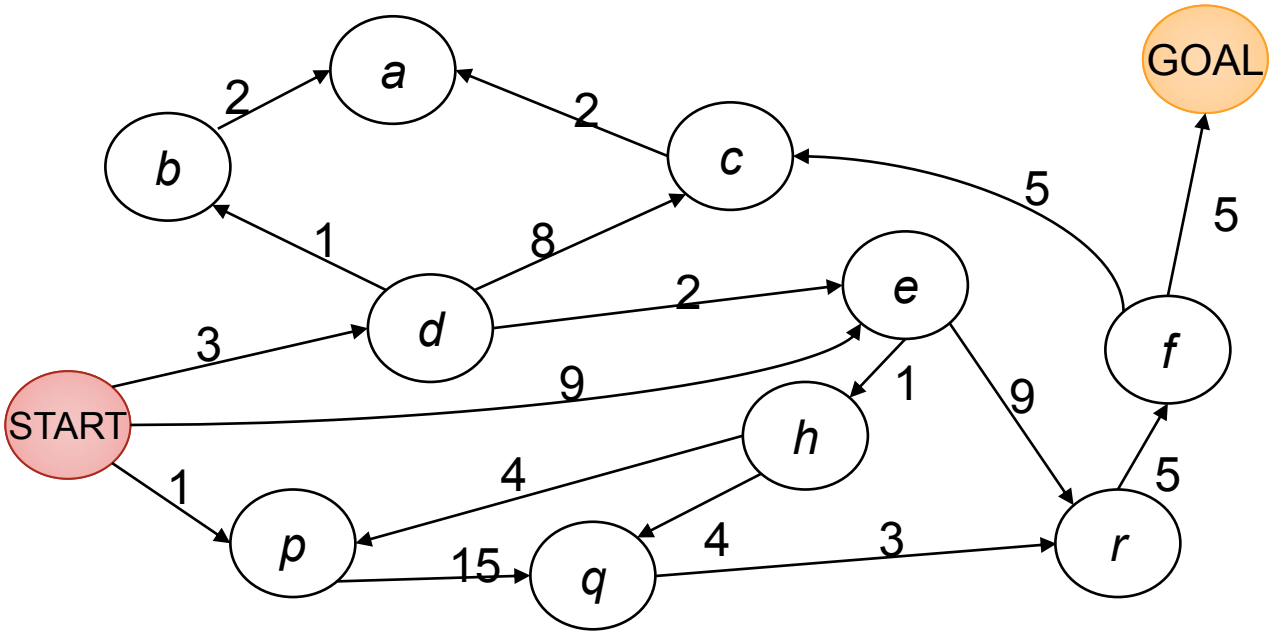
Trường hợp 1:




Trường hợp 2:



# CHI PHÍ ĐƯỜNG ĐI




- Chú ý rằng BFS tìm đường đi ngắn nhất theo số bước chuyển. Nó không tìm đường đi có chi phí ít nhất.
- Xét thuật toán có khả năng tìm đường đi ngắn nhất. Tại vòng lặp thứ *k*, với trạng thái *S* bất kì, viết hàm *g*(*s*) là chi phí của đường đi có chi phí nhỏ nhất đến S trong *k* bước hoặc ít hơn.

# MÔ TẢ THUẬT TOÁN

$V_k$ = tập trạng thái có thể đến trong đúng $k$ bước và đường đi $k$ bước có chi phí ít hơn bất kì đường đi nào có chiều dài nhỏ hơn $k$. Nói cách khác, $V_k$ là tập các trạng thái có giá trị thay đổi trong vòng lặp trước.

$V_0$ := $S$ (tập các trạng thái bắt đầu)

*previous*(*START*) := *NIL* $g(START) = 0$ $k := 0$

**while** ($V_k$ không 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 được gán nhãn

(A)

Ngược lại

Nếu $g(s) + Cost(s,s') < g(s')$

(A)

**(A)**

Đặt *previous*($s'$) := $s$

Đặt $g(s'):=g(s) + Cost(s,s')$

Thêm $s'$ vào $V_{k+1}$

$k := k+1$

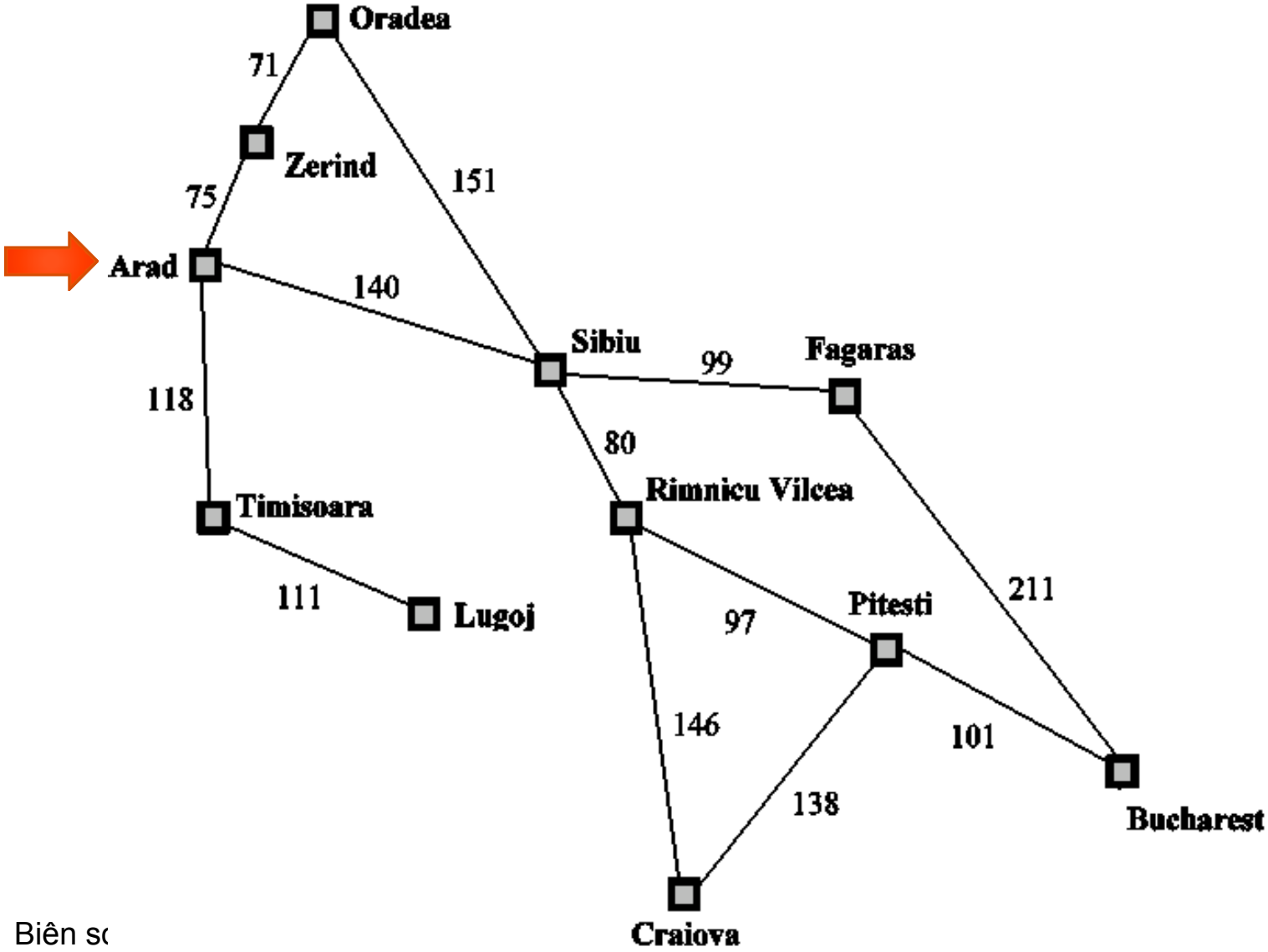
**If** GOAL không được gán nhãn thì thông báo THẤT BẠI

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

# BFS CÓ CHI PHÍ

- Chạy đến khi hết đỉnh để mở
- Mỗi đỉnh sẽ có kèm chi phí theo
- Cận nhật lại chi phí đỉnh nếu chi phí mới nhỏ hơn

# BÀI TẬP VÍ DỤ


Oradea
71
Zerind
75
151
Arad
140
Sibiu
99
Fagaras
118
80
Rimnicu Vilcea
Timisoara
111
Lugoj
97
Pitesti
211
146
101
138
Bucharest
Craiova

# BÀI TẬP VÍ DỤ

- $V_0$ = {(Arad, 0)}
- $V_1$ = {(Sibiu, 140), (Timisoara, 118), (Zerind, 75)}
- $V_2$ = {(Faragas, 239), (Rimnicu Vilcea, 220), (Lugoj, 229), (Oradea, 146)}
- $V_3$ = {(Bucharest, 450), (Pitesti, 317), (Craiova, 366), ~~(Sibiu, 297)~~}
- $V_4$ = {(Bucharest, 418), ~~(Pitesti, 504)~~}
- $V_5$ = { } rỗng, dừng thuật toán.

Đường đi là: Arad → Sibiu → Rimnicu Vilcea → Pitesti → Bucharest. Chi phí: 418

# TÌM KIẾM
# CHI PHÍ ĐỒNG NHẤT
## (Uniform-Cost Search)

# HÀNG ĐỢI ƯU TIÊN

- Cấu trúc dữ liệu cho phép thêm vào và lấy ra các cặp (thing, value) theo các thao tác:

| | |
|---|---|
| Init-PriQueue(PQ) | Khởi tạo hàng đợi ưu tiên rỗng. |
| Insert-PriQueue(PQ, thing, value) | Thêm (thing, value) vào hàng đợi |
| Pop-least(PQ) | Trả về cặp (thing, value) có giá trị nhỏ nhất và loại nó ra khỏi hàng đợi. |

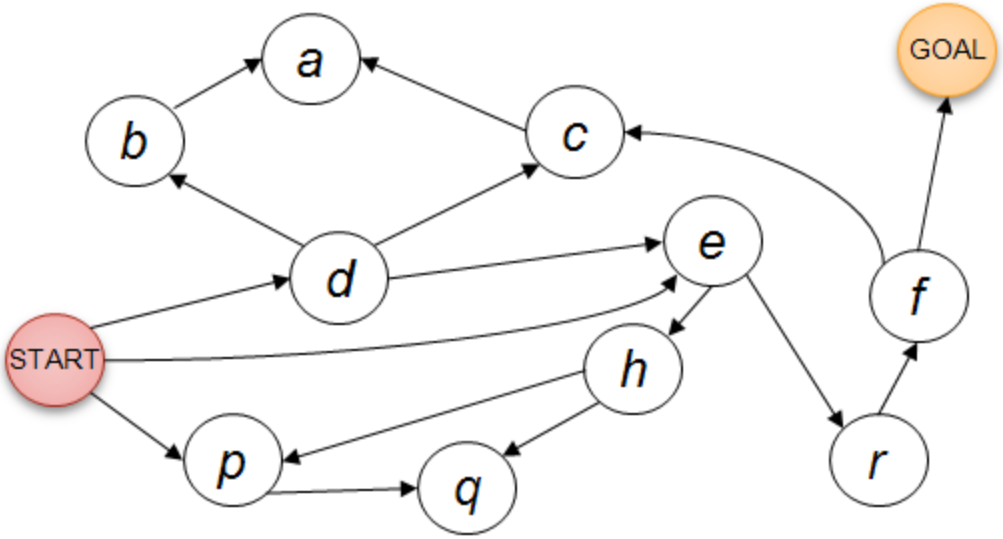
- Hàng đợi ưu tiên có thể được cài đặt sao cho chi phí thêm vào và lấy ra là:

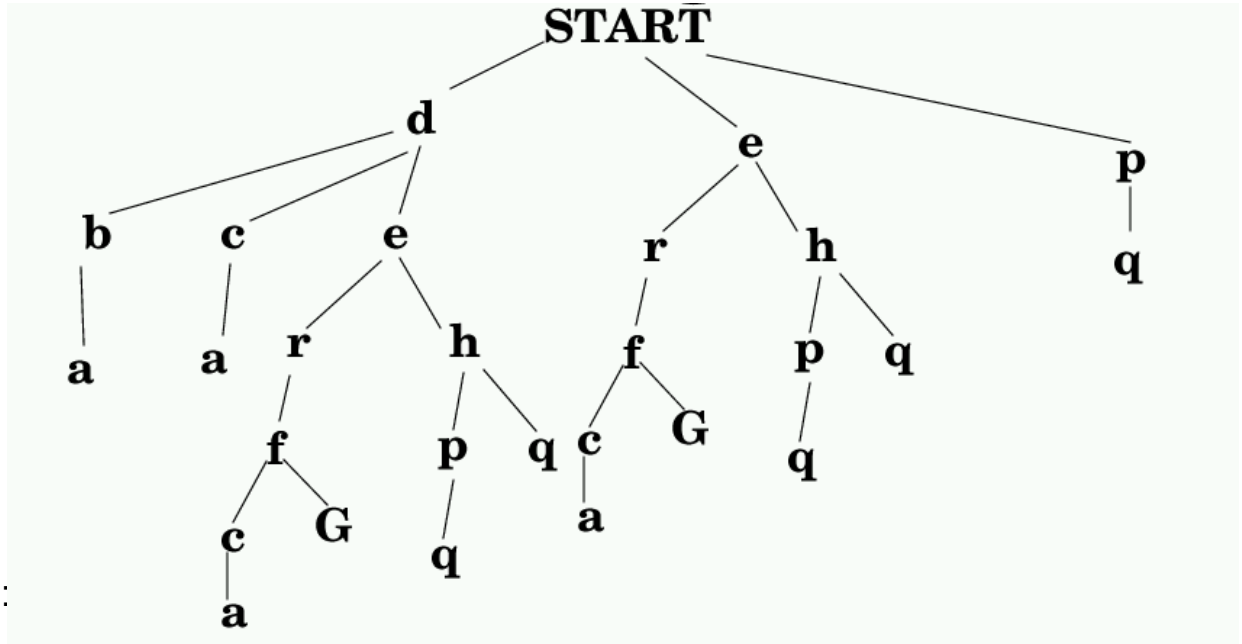
*O(log(số phần tử trong hàng đợi ưu tiên))*

# TÌM KIẾM CP ĐỒNG NHẤT

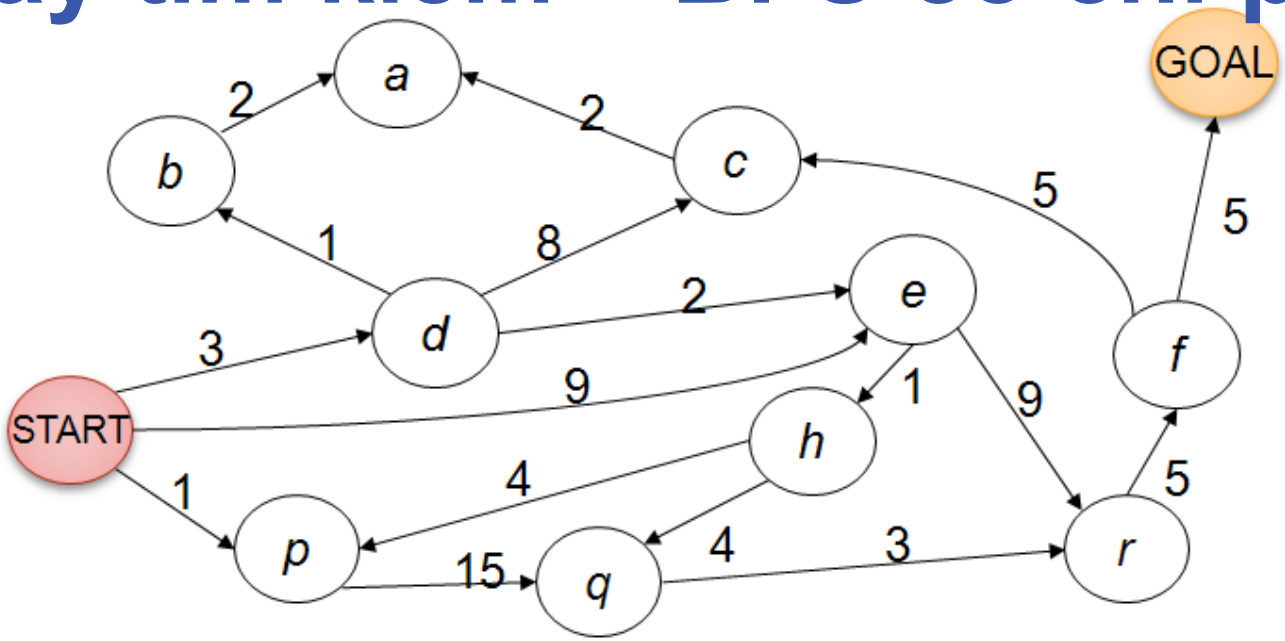
- Một hướng tiếp cận kiểu BFS đơn giản khi có chi phí trên các bước chuyển.
- Sử dụng hàng đợi ưu tiên
  - PQ = Tập trạng thái đã được mở hay đang đợi mở
  - Độ ưu tiên của trạng thái *s* = *g(s)* = chi phí đến *s* dùng đường đi cho bởi con trỏ quay lui.

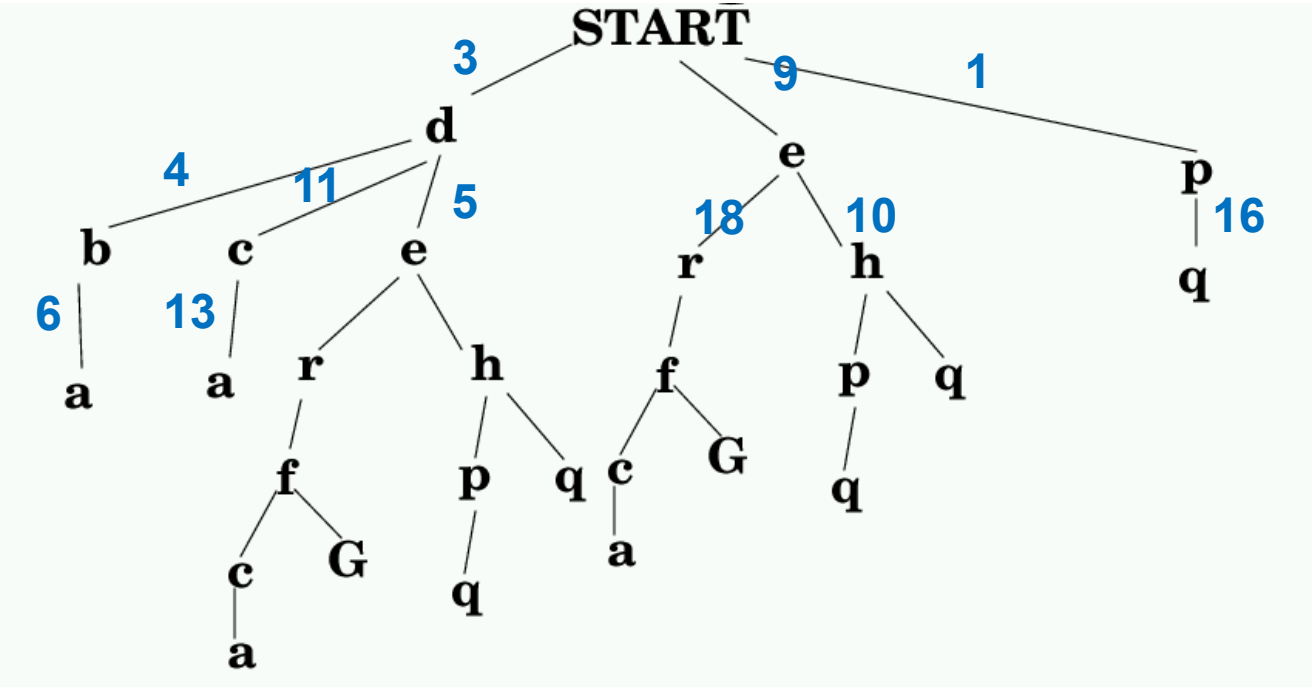
# CÂY TÌM KIẾM - BFS



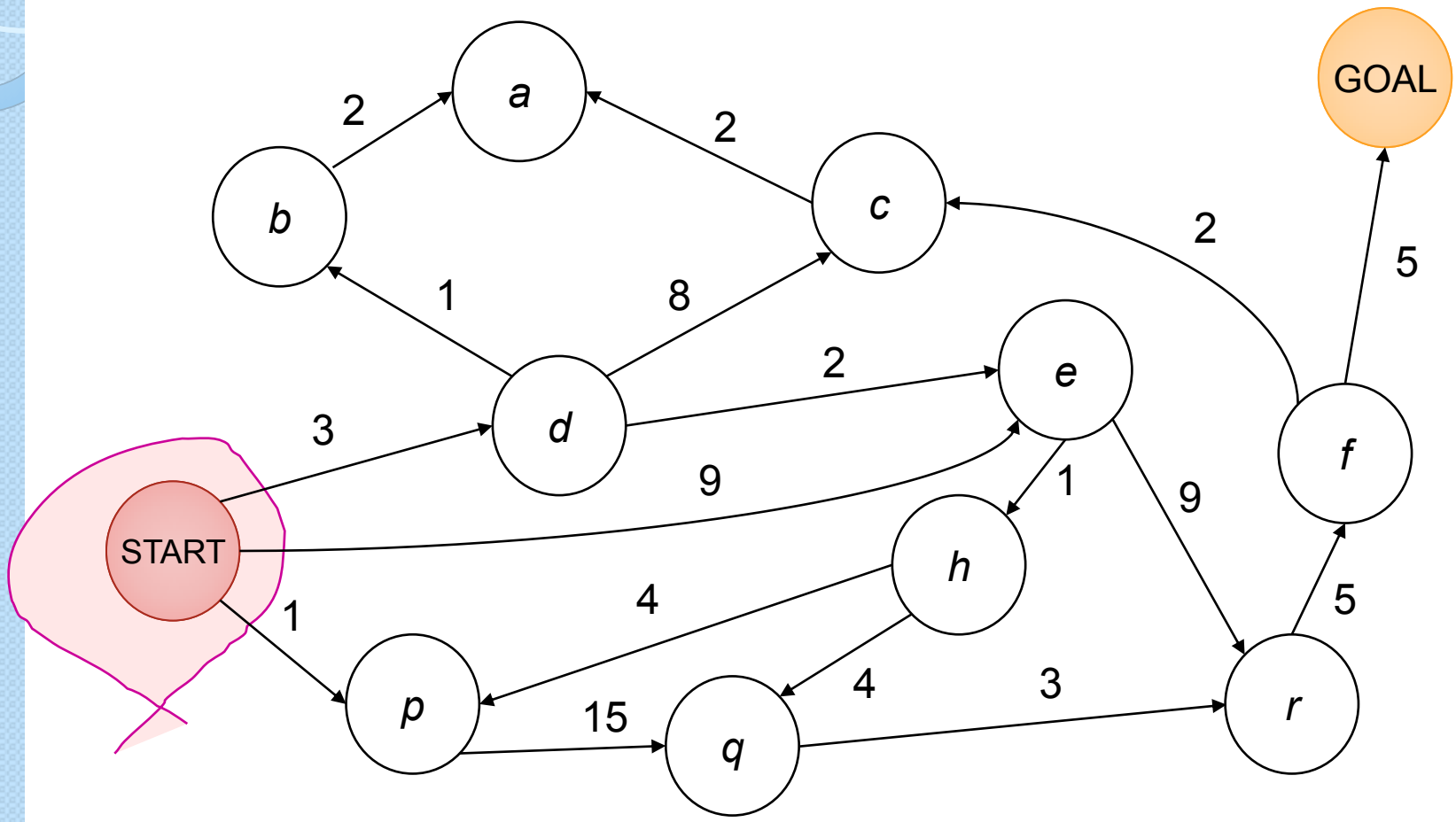



# Cây tìm kiếm – BFS có chi phí



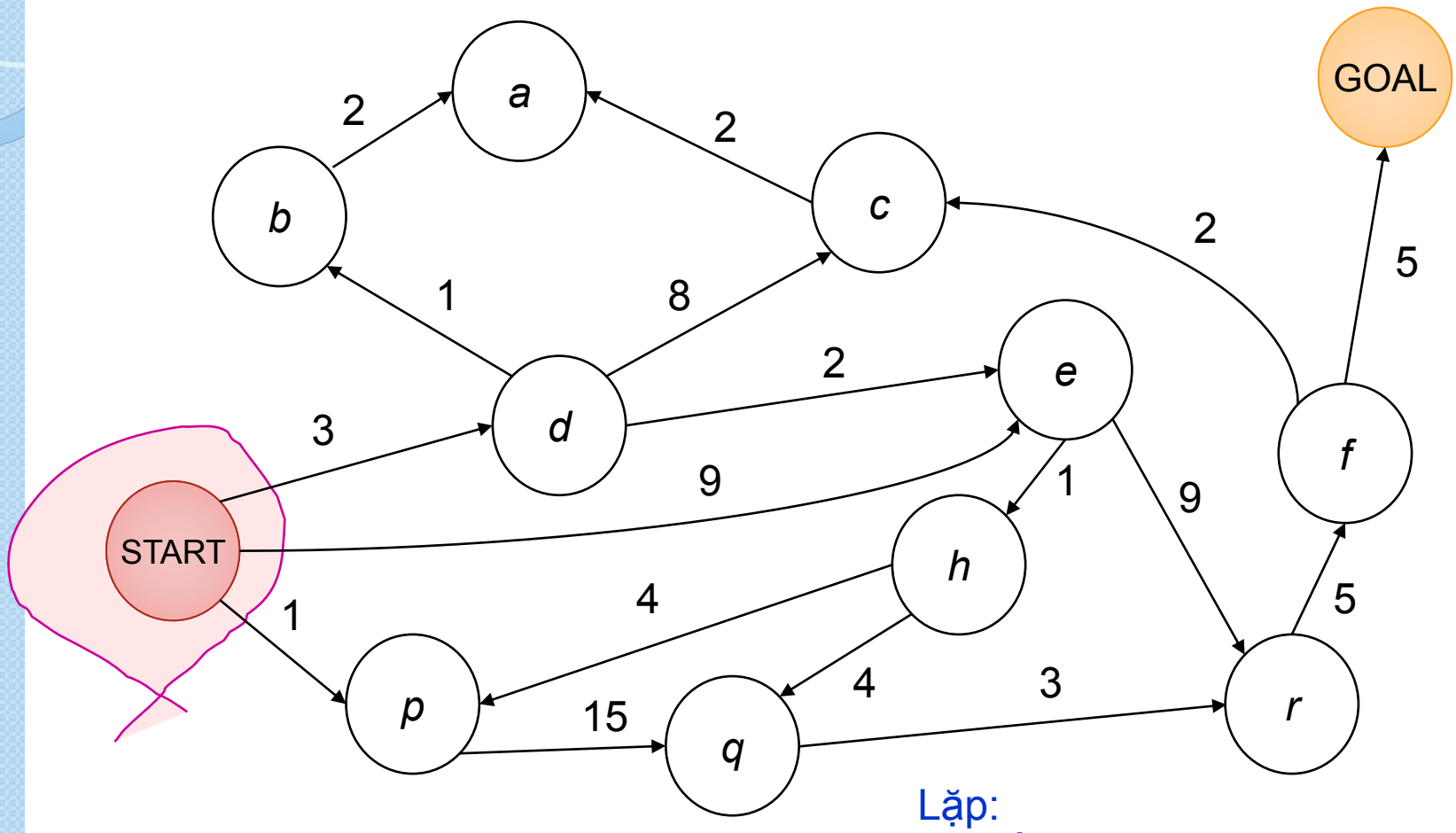



# TÌM KIẾM CP ĐỒNG NHẤT




*PQ = { (S,0) }*

# TÌM KIẾM CP ĐỒNG NHẤT

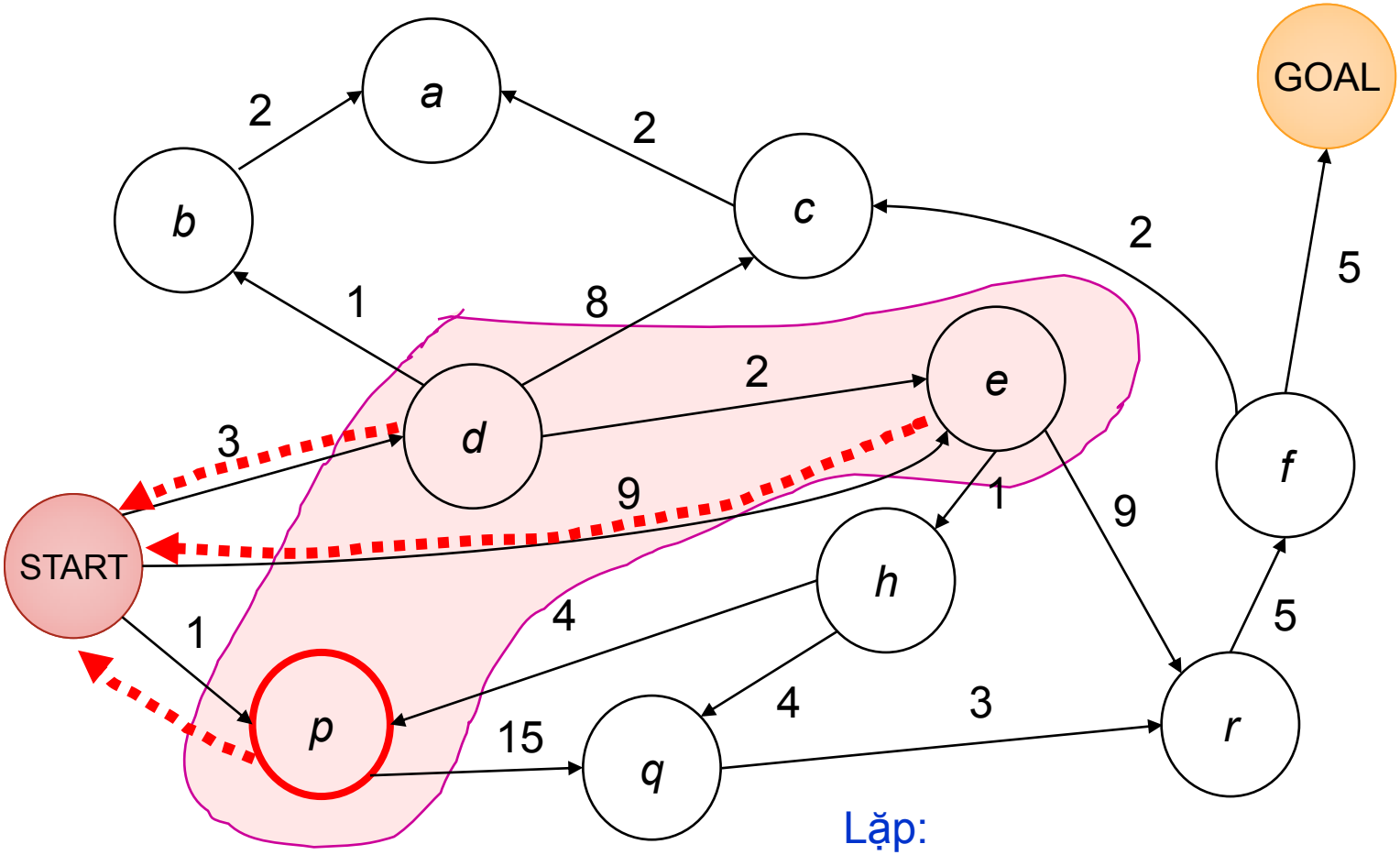



Lặp:

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

*PQ = { (S,0) }*

# TÌM KIẾM CP ĐỒNG NHẤT

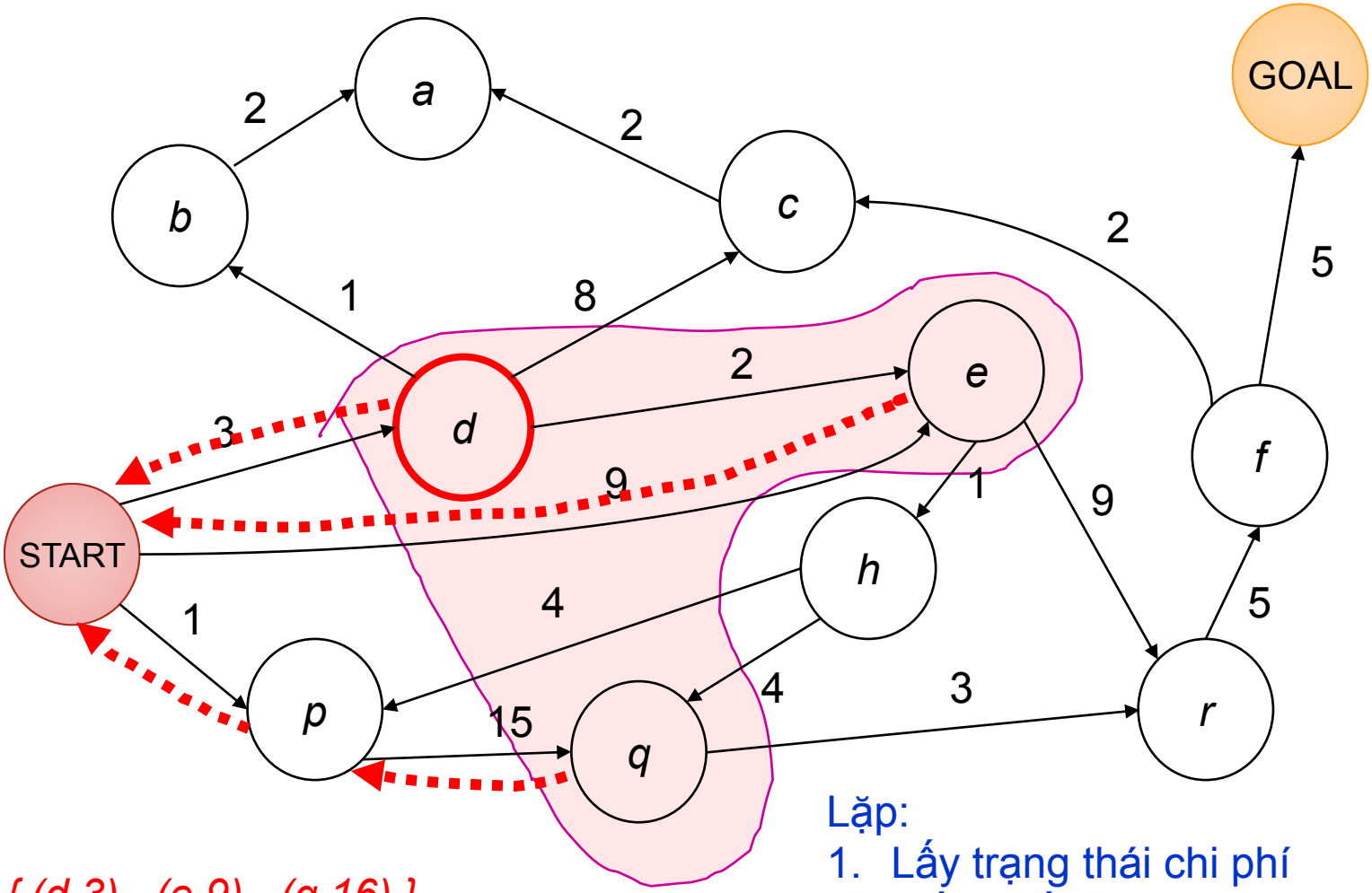



*PQ = { (p,1), (d,3) , (e,9) }*

Lặp:

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

# TÌM KIẾM CP ĐỒNG NHẤT

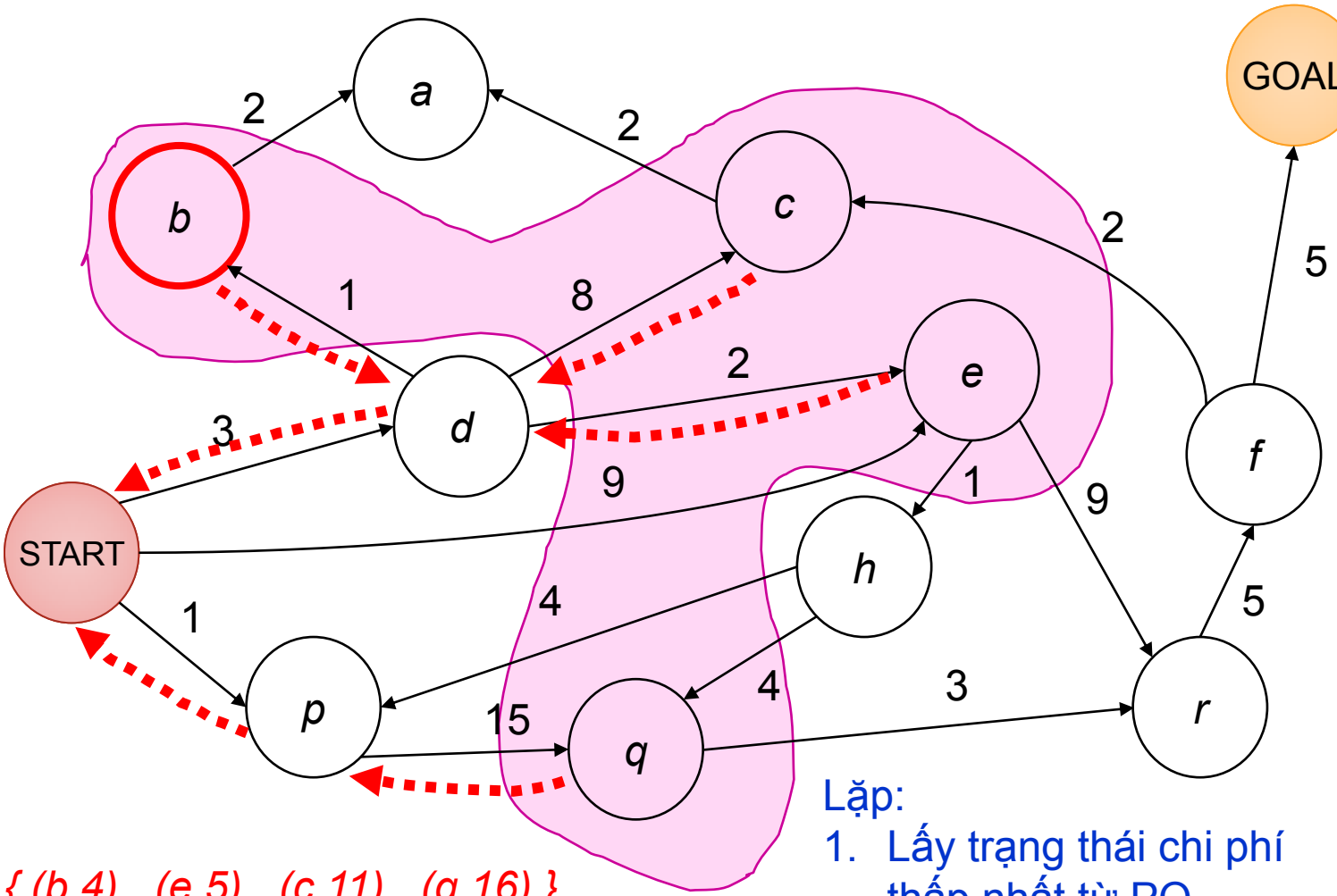



Lặp:

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

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

# TÌM KIẾM CP ĐỒNG NHẤT

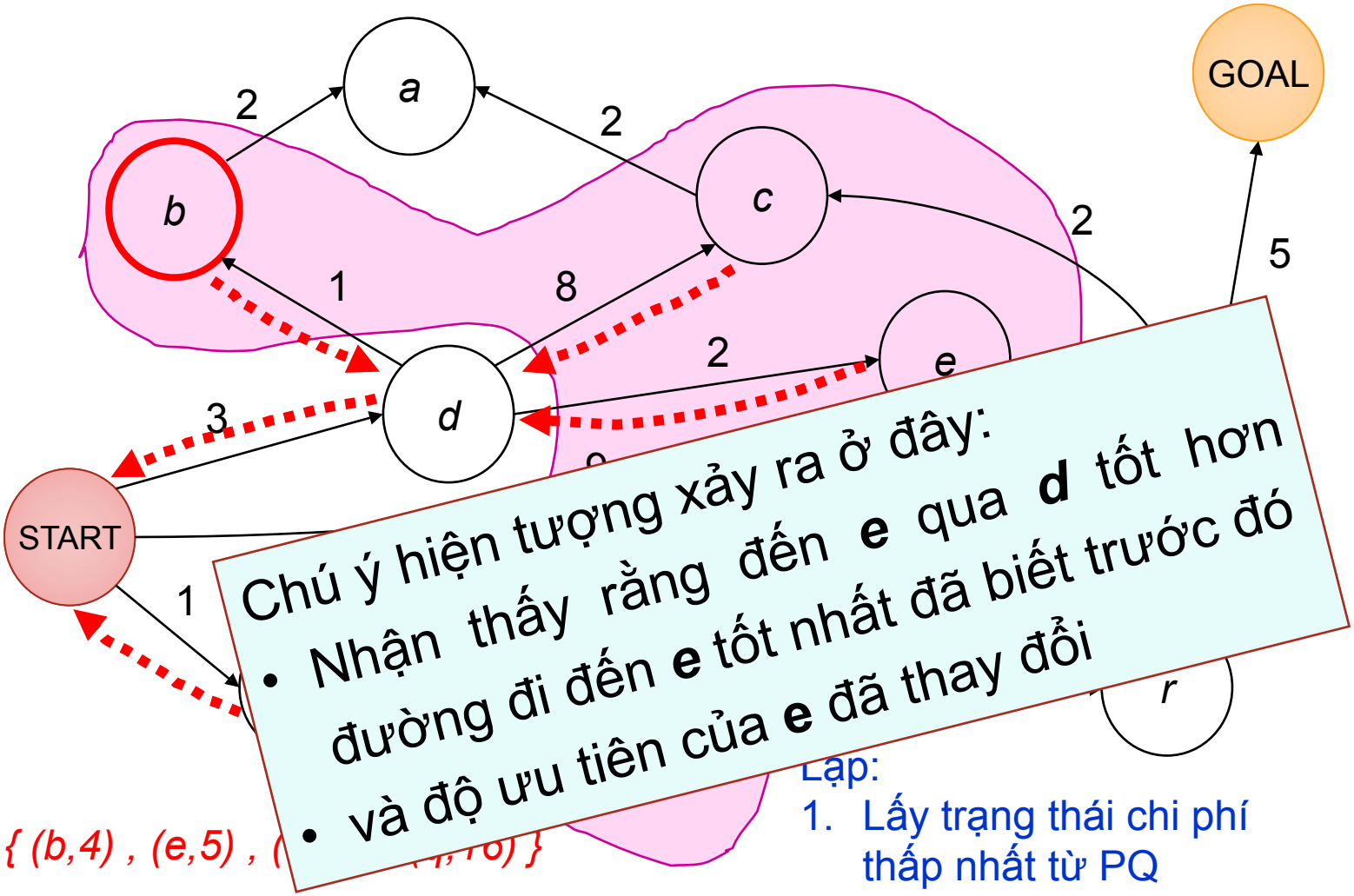



Lặp:

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

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

# TÌM KIẾM CP ĐỒNG NHẤT

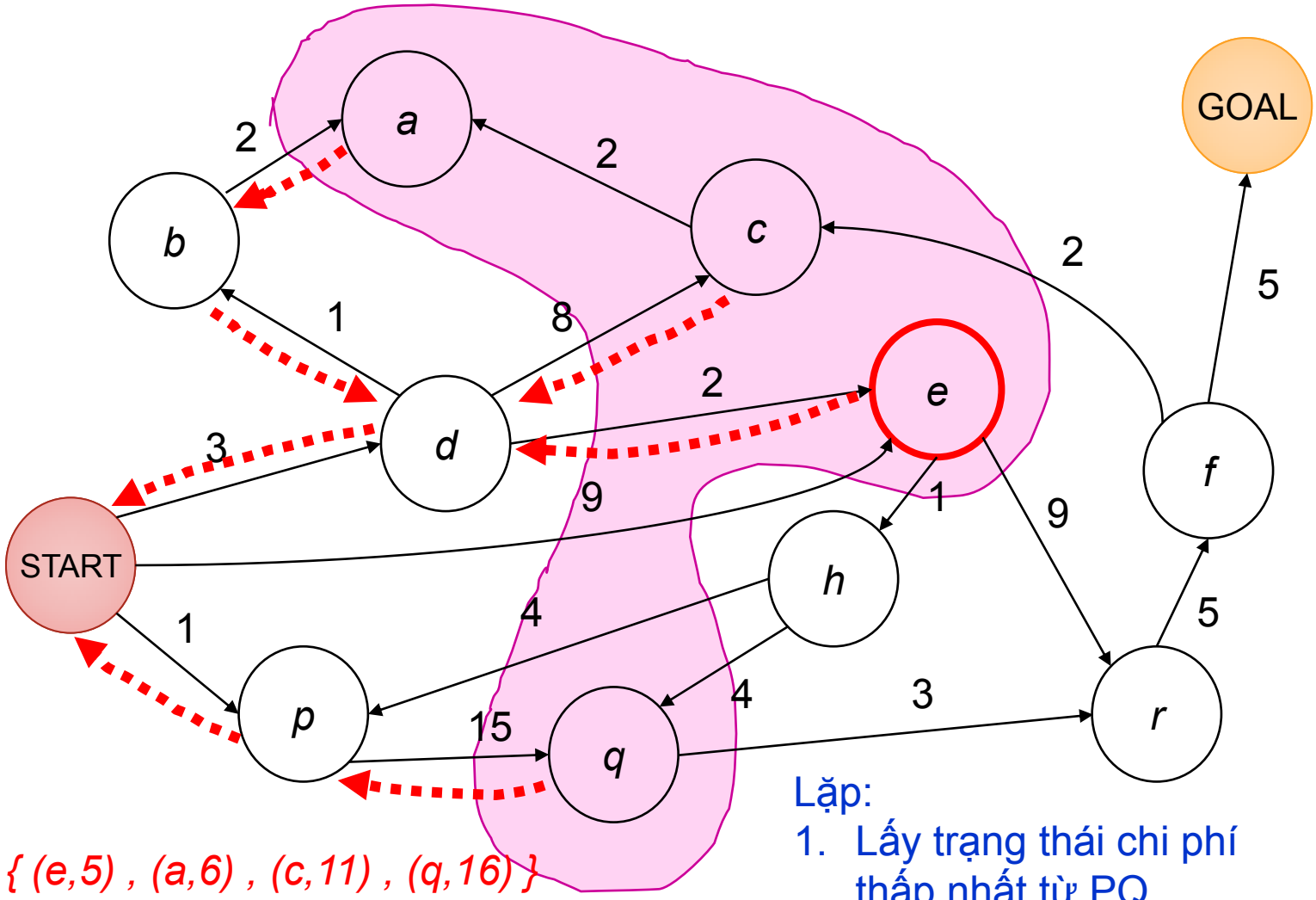



*PQ = { (b,4) , (e,5) , (*

Lặp:

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

# TÌM KIẾM CP ĐỒNG NHẤT

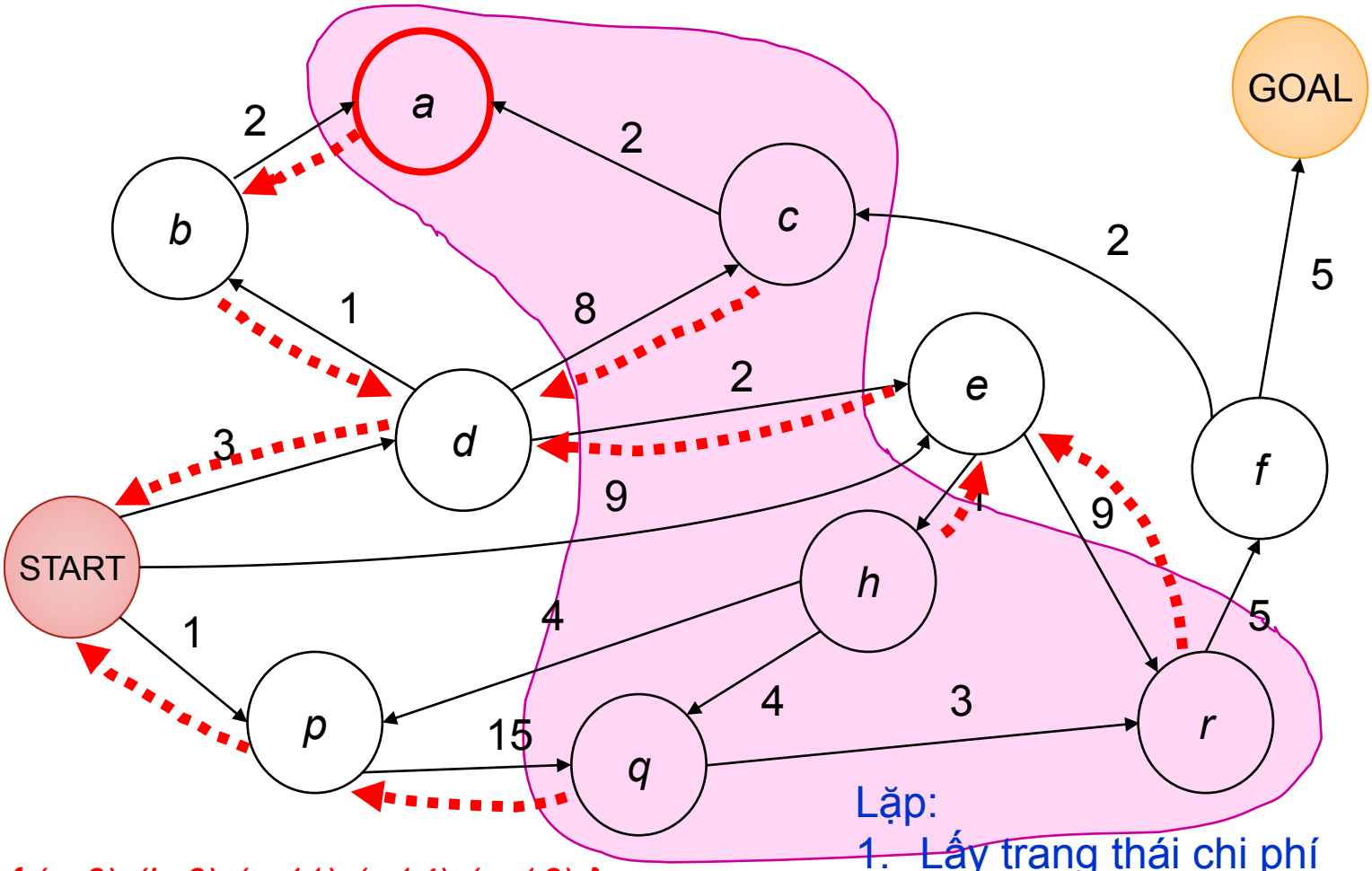



*PQ = { (e,5) , (a,6) , (c,11) , (q,16) }*

Lặp:

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

# TÌM KIẾM CP ĐỒNG NHẤT

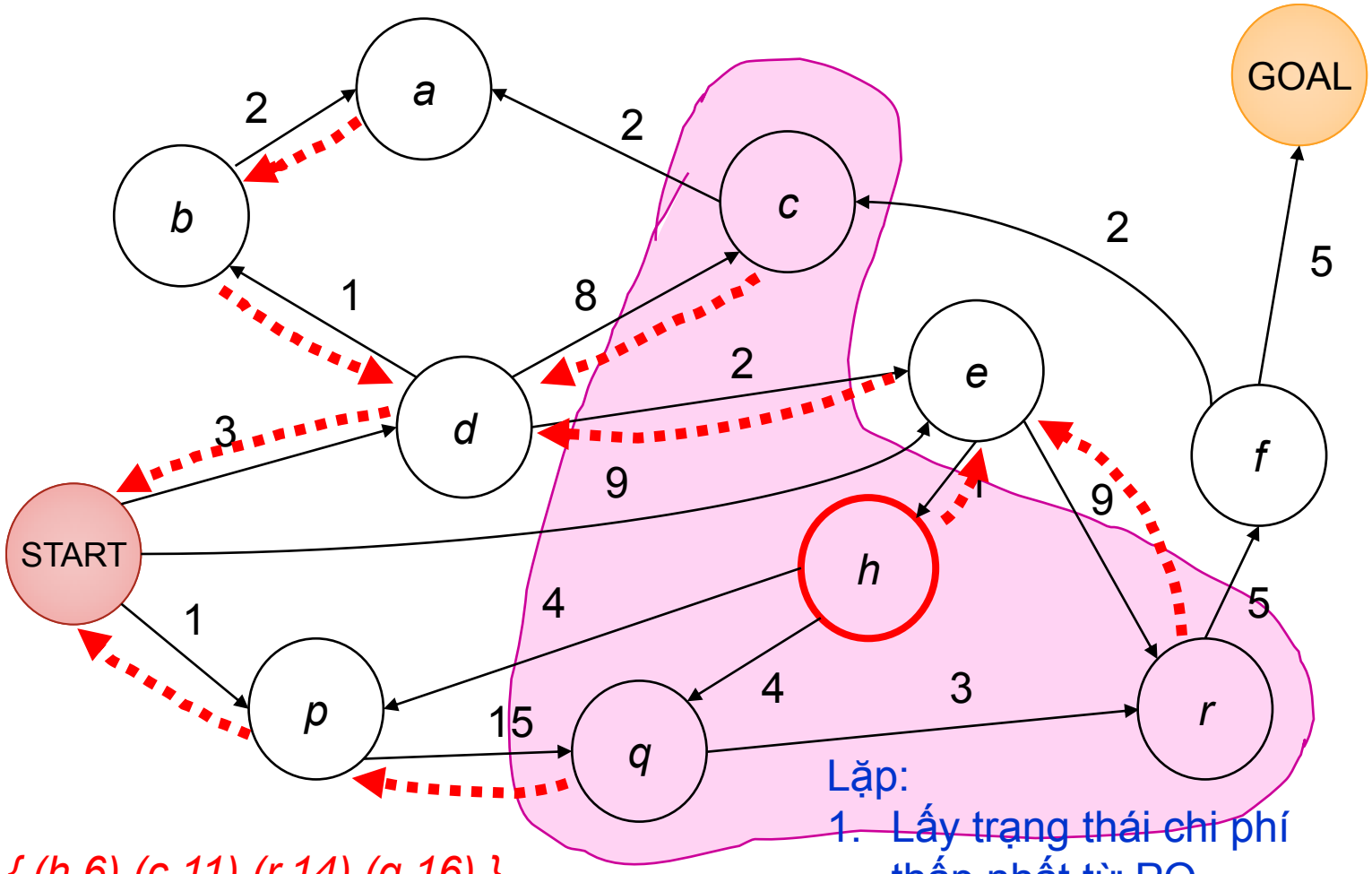



*PQ = { (a,6),(h,6),(c,11),(r,14),(q,16) }*

Lặp:

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

# TÌM KIẾM CP ĐỒNG NHẤT

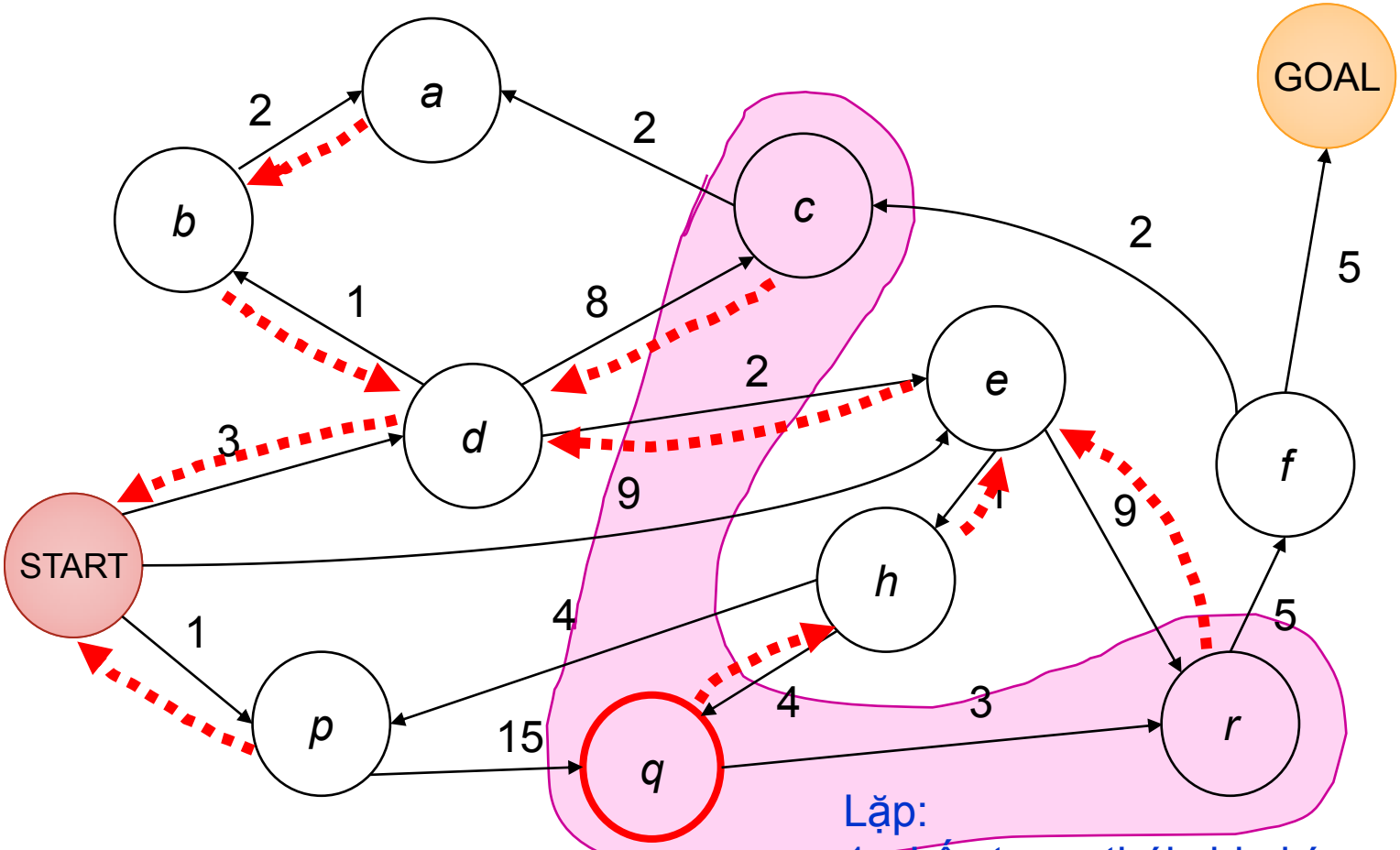



Lặp:

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

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

# TÌM KIẾM CP ĐỒNG NHẤT




Lặp:

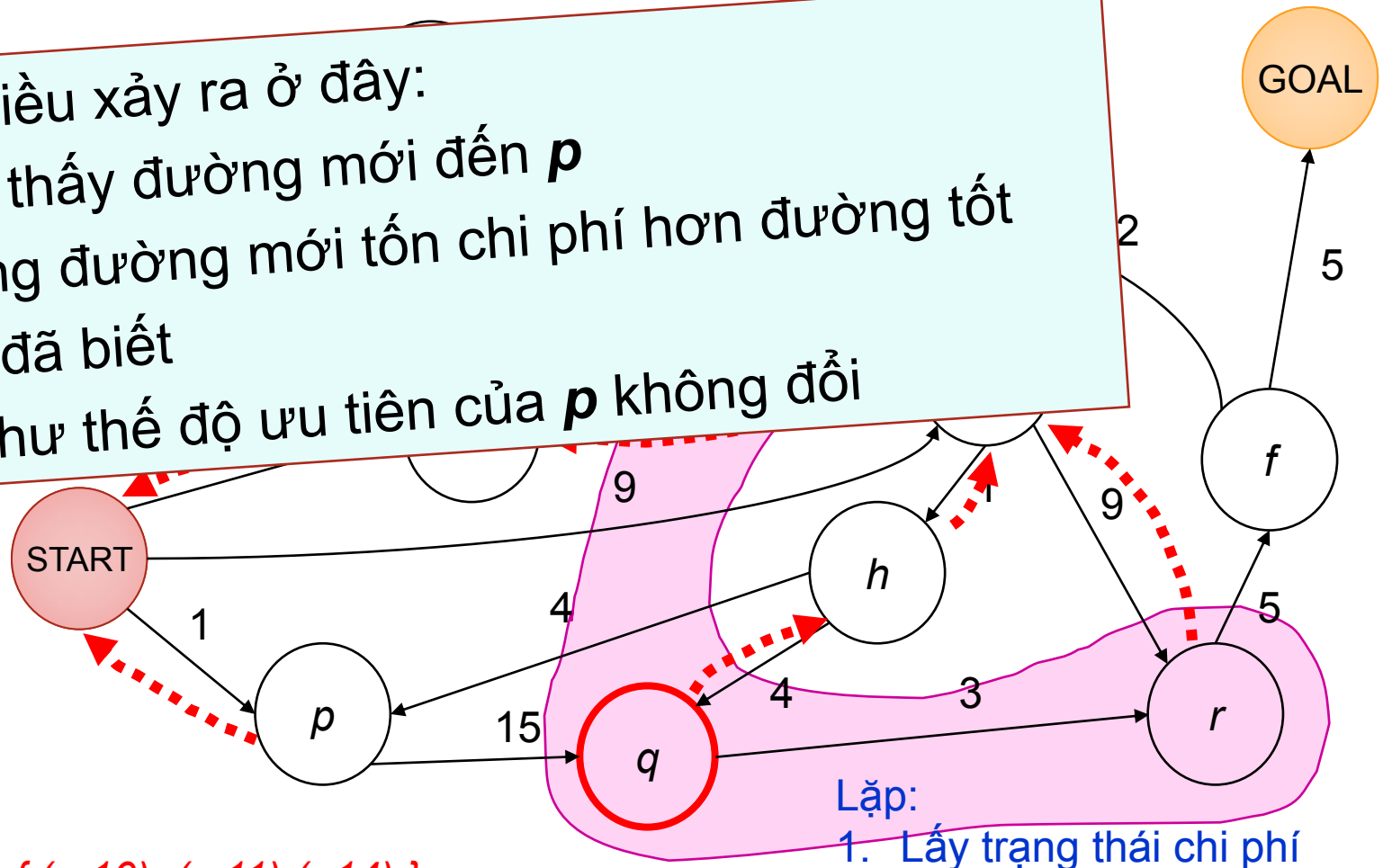
1. Lấy trạng thái chi phí thấp nhất từ PQ
2. Thêm các con

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

# TÌM KIẾM CP ĐỒNG NHẤT

Chú ý điều xảy ra ở đây:

- ***h*** tìm thấy đường mới đến ***p***
- Nhưng đường mới tốn chi phí hơn đường tốt nhất đã biết
- Và như thế độ ưu tiên của ***p*** không đổi

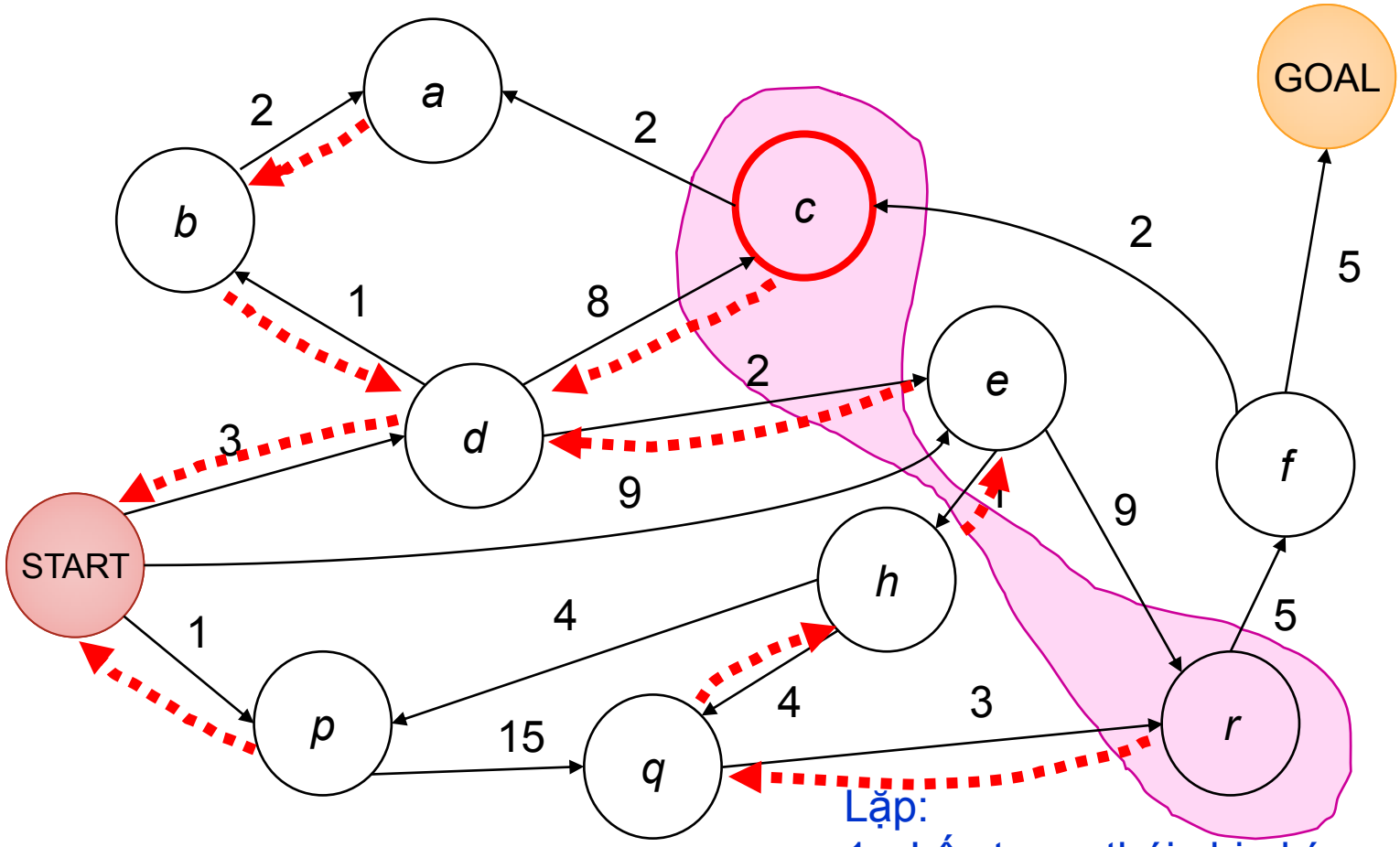



Lặp:

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

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

# TÌM KIẾM CP ĐỒNG NHẤT

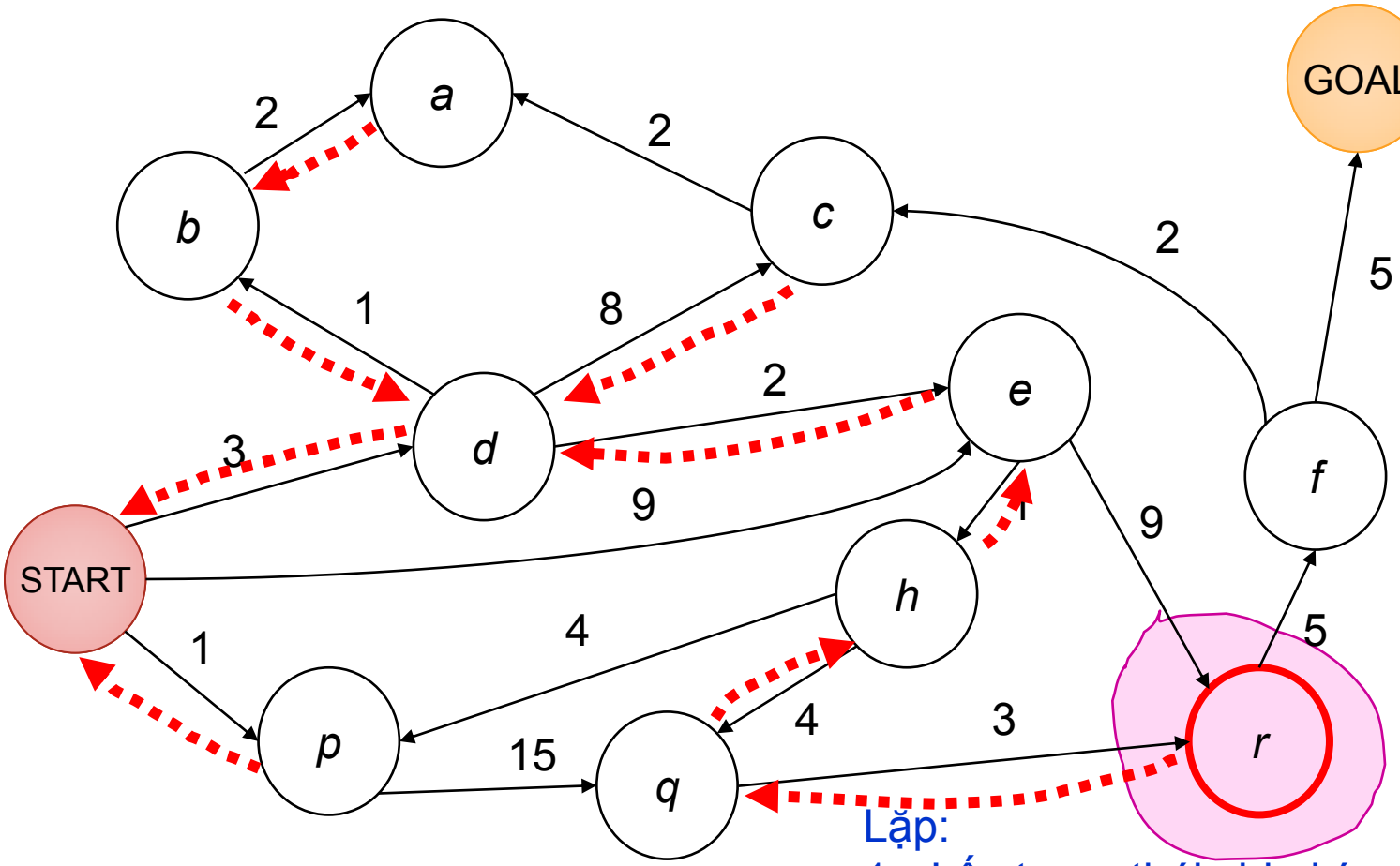



Lặp:

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

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

# TÌM KIẾM CP ĐỒNG NHẤT

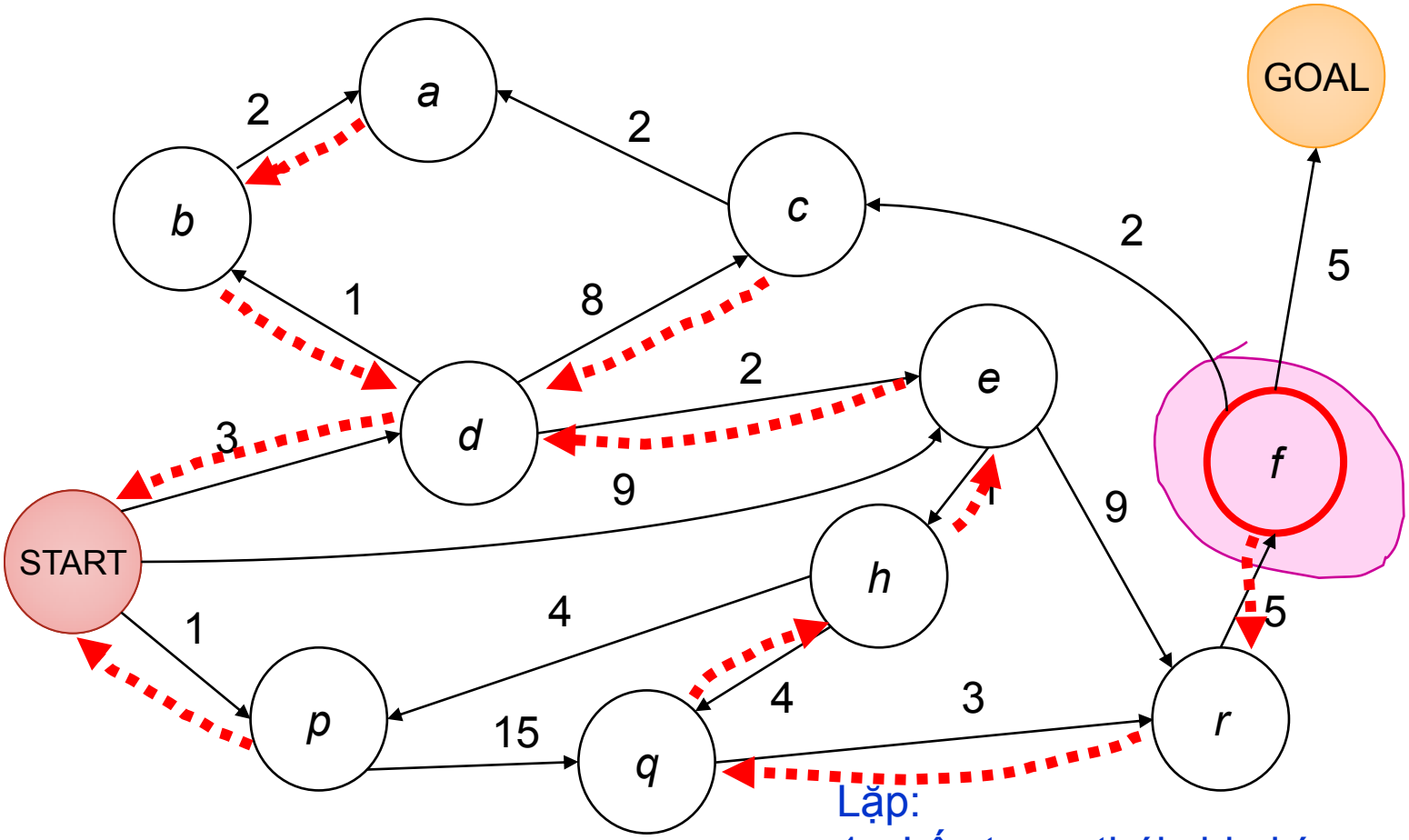



Lặp:

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

*PQ = { (r,13) }*

# TÌM KIẾM CP ĐỒNG NHẤT

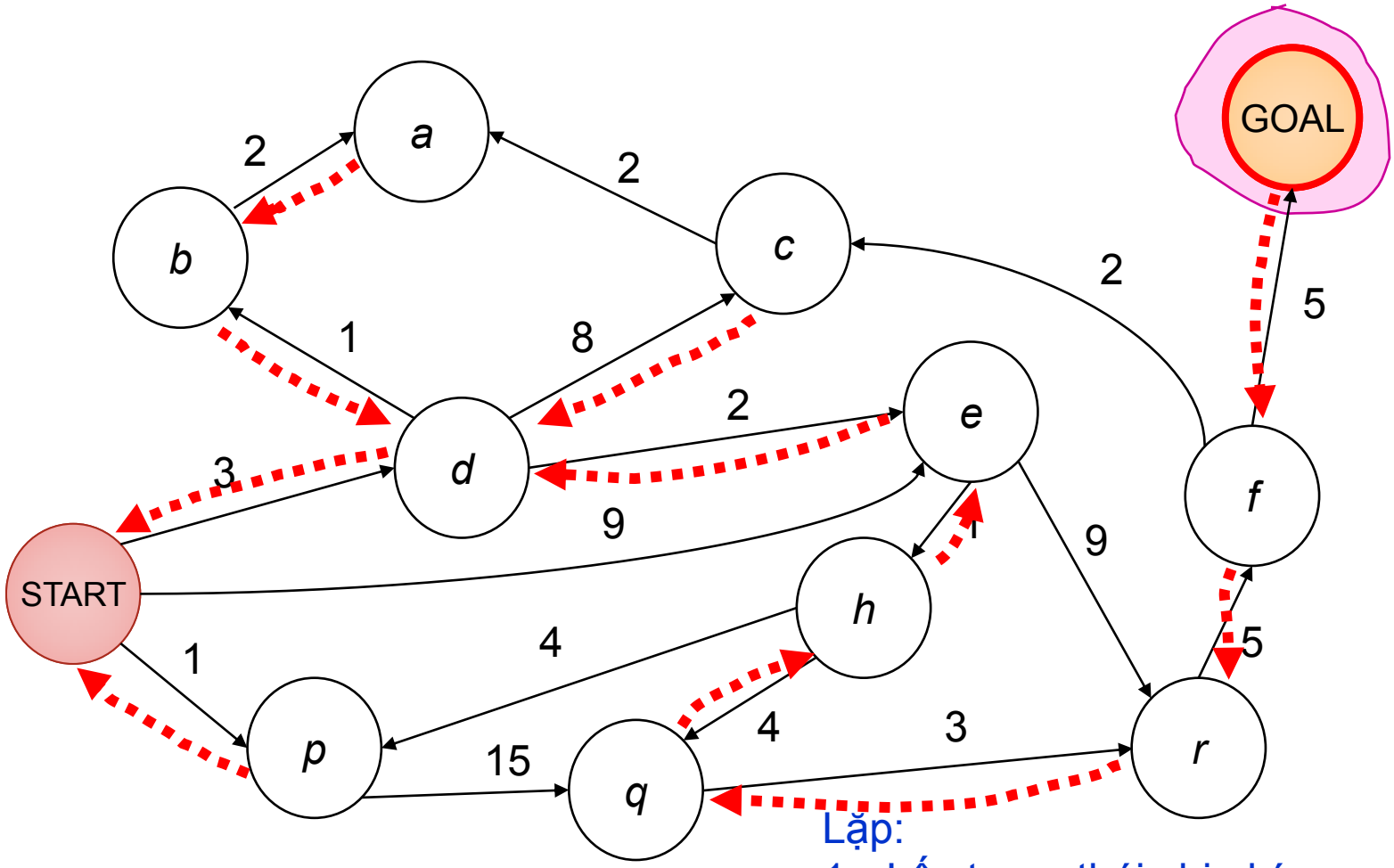



Lặp:

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

*PQ = { (f, 18) }*

# TÌM KIẾM CP ĐỒNG NHẤT



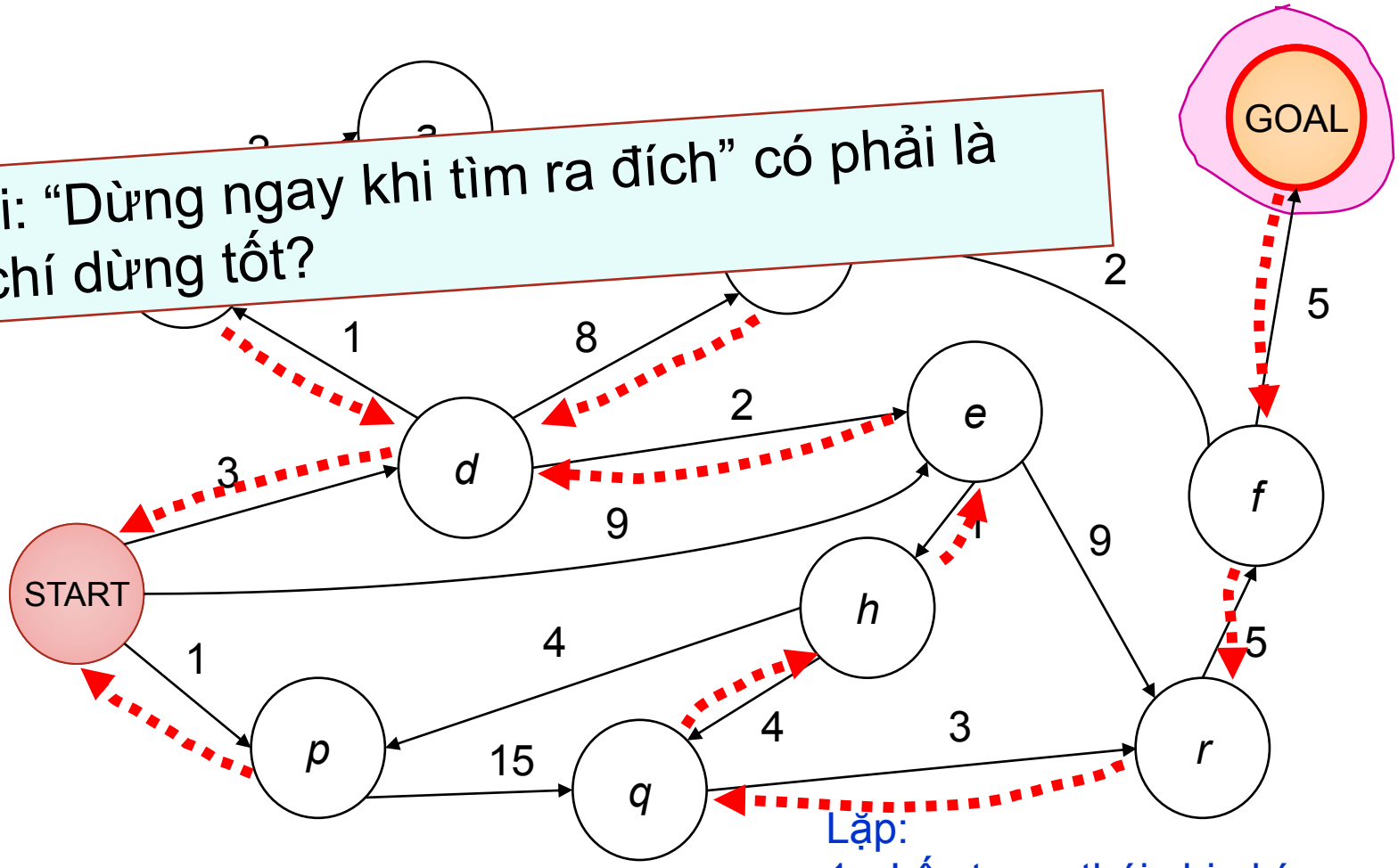

Lặp:

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

*PQ = { (G, 23) }*

# TÌM KIẾM CP ĐỒNG NHẤT

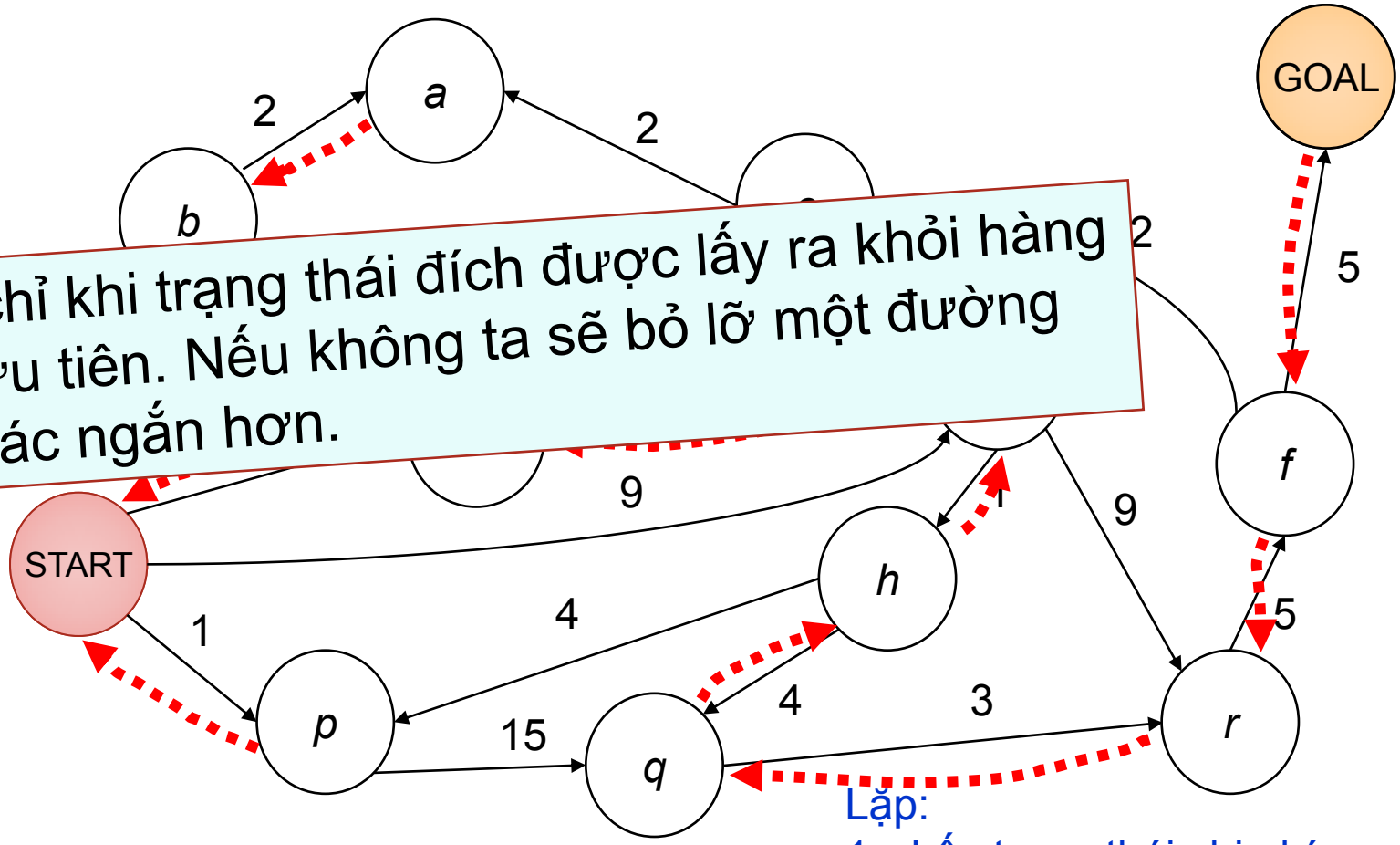
Câu hỏi: "Dừng ngay khi tìm ra đích" có phải là tiêu chí dừng tốt?




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

*PQ = { (G, 23) }*

# ĐIỀU KIỆN DỪNG



# ĐÁNH GIÁ THUẬT TOÁN

- **Tính đầy đủ**: bảo đảm tìm ra lời giải nếu có
- **Tính tối ưu**: tìm được đường đi có chi phí ít nhất
- Độ phức tạp về thời gian: số node phát sinh
- Độ phức tạp về không gian: lượng bộ nhớ sử dụng

Các biến

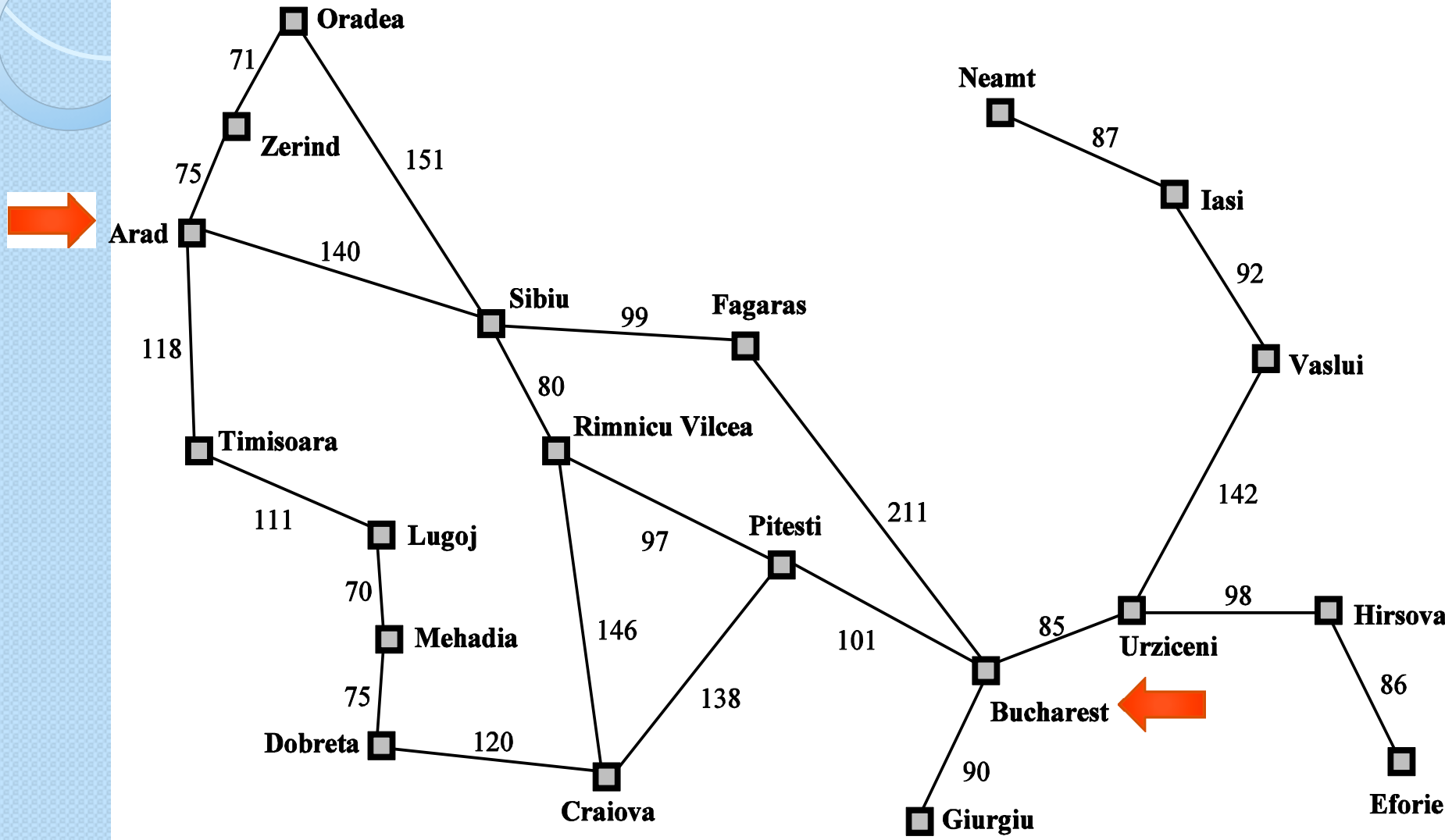
| | |
|---|---|
| N | số trạng thái của bài toán |
| B | nhân tố phân nhánh trung bình (số con trung bình) ($B>1$) |
| L | độ dài đường đi từ start đến goal với số bước ngắn nhất |

# ĐÁNH GIÁ 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 | $O(min(N,B^L))$ | $O(min(N,B^L))$ |
| UCS | Uniform Cost Search | C | C | $O(log(Q) * min(N,B^L))$ | $O(min(N,B^L))$ |

# BÀI TẬP VÍ DỤ


Oradea
71
Zerind
75
151
Arad
140
Sibiu
99
Fagaras
118
80
Rimnicu Vilcea
Timisoara
111
Lugoj
97
Pitesti
211
70
Mehadia
146
101
75
138
Dobreta
120
Craiova
90
Giurgiu
Bucharest
85
Urziceni
98
Hirsova
86
Eforie
142
Vaslui
92
Iasi
87
Neamt

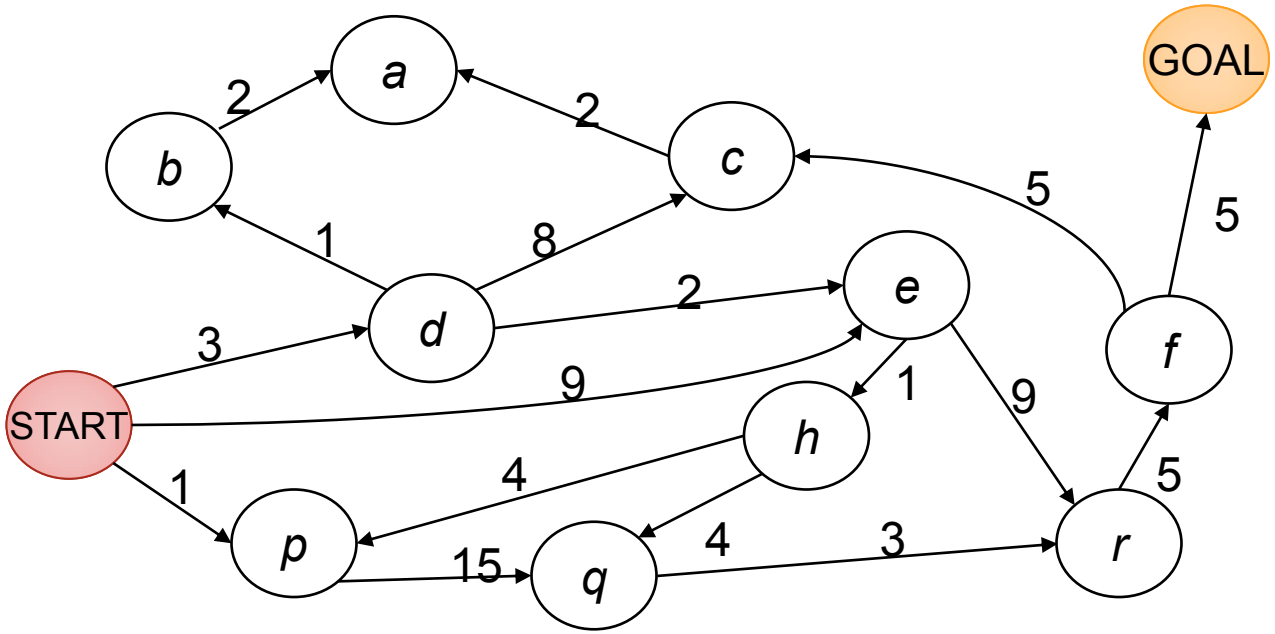
# BÀI TẬP VÍ DỤ

- PQ = {(Arad, 0)}
- PQ = {(Zerind, 75), (Timisoara, 118), (Sibiu, 140)}
- PQ = {(Timisoara, 118), (Sibiu, 140), (Oradea, 146)}
- PQ = {(Sibiu, 140), (Oradea, 146), (Lugoj, 229)}
- PQ = {(Oradea, 146), (Rimnicu Vilcea, 220), (Lugoj, 229), (Faragas, 239)}
- PQ = {(Rimnicu Vilcea, 220), (Lugoj, 229), (Faragas, 239)}
- PQ = {(Lugoj, 229), (Faragas, 239), (Pitesti, 317), (Craiova, 366)}
- PQ = {(Faragas, 239), (Mehadia, 299), (Pitesti, 317), (Craiova, 366)}
- PQ = {(Mahadia, 299), (Pitesti, 317), (Craiova, 366), (Bucharest, 450)}
- PQ = {(Pitesti, 317), (Craiova, 366), (Dobreta, 374), (Bucharest, 450)}
- PQ = {(Craiova, 366), (Dobreta, 374), (Bucharest, 418)}
- PQ = {(Dobreta, 374), (Bucharest, 418)}
- PQ = {(Bucharest, 418)}
- PQ = { }

Arad → Sibiu → Rimnucu Vilcea → Pitesti → Bucharest. Chi phí: 418

# TÌM KIẾM THEO CHIỀU SÂU (Depth-First Search – DFS)

# TÌM KIẾM THEO CHIỀU SÂU




- Một lựa chọn khác với BFS. Luôn mở từ nút vừa mới mở nhất, nếu nó có bất kì nút con chưa thử nào. Ngược lại, quay về nút trước đó trên đường đi hiện tại.

# DUYỆT CÂY TÌM KIẾM DFS

START

START *d*

START *d b*

START *d b a*

START *d c*

START *d c a*

START *d e*

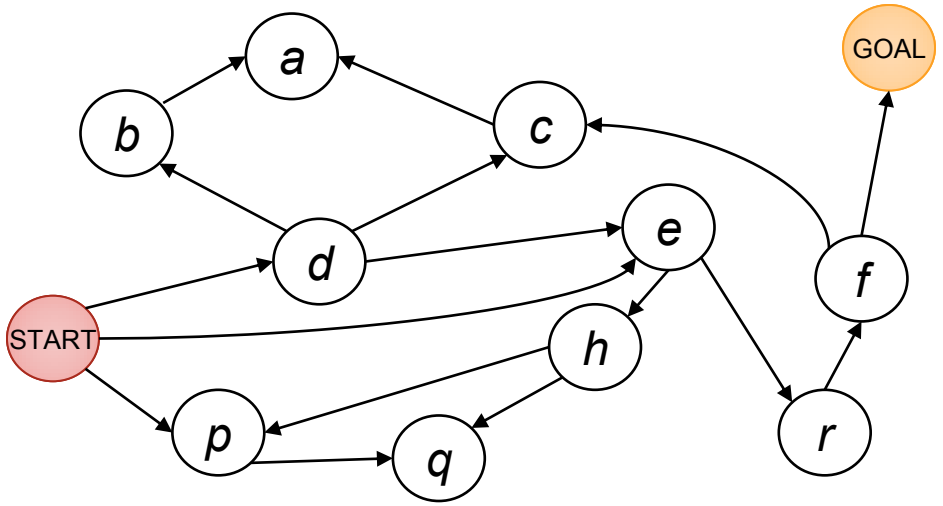
START *d e r*

START *d e r f*

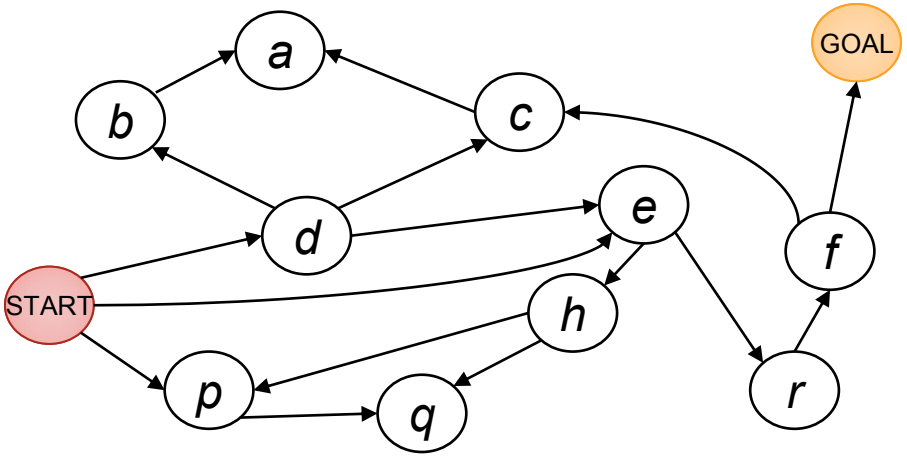
START *d e r f c*

START *d e r f c a*

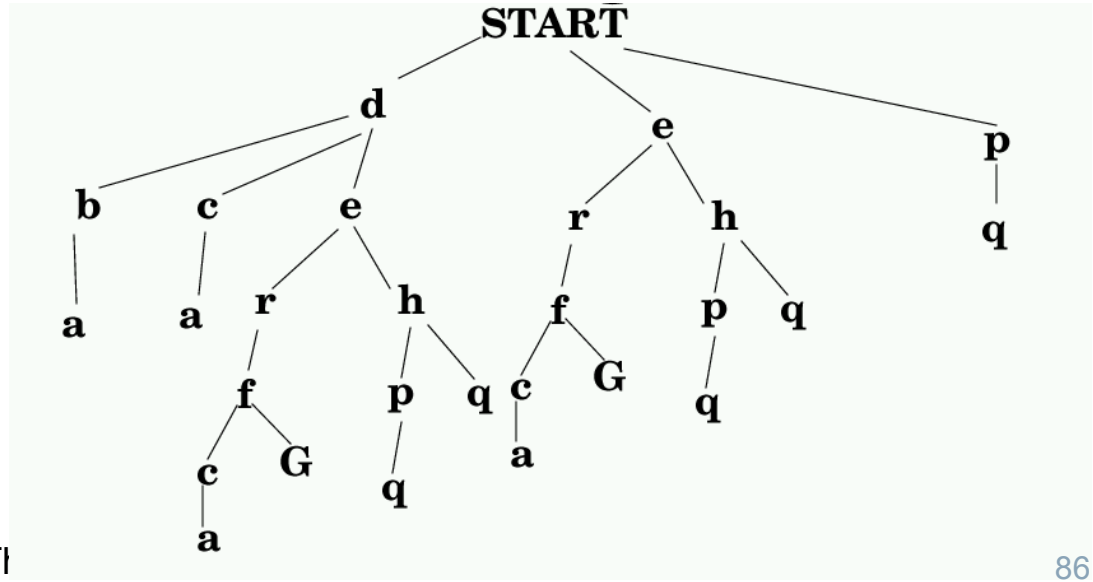
START *d e r f* GOAL



# DUYỆT CÂY TÌM KIẾM DFS

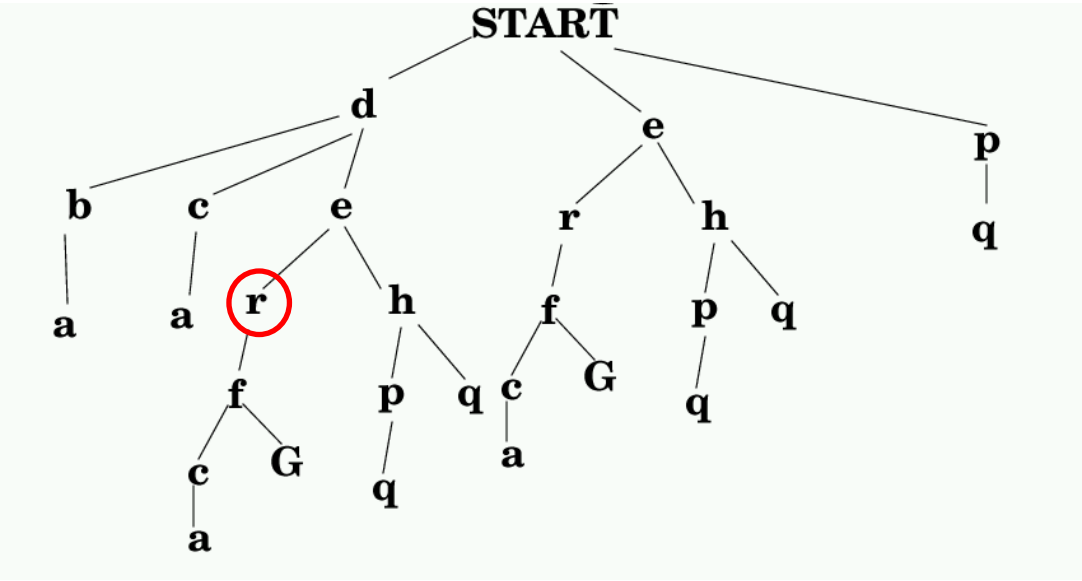



Hãy chỉ ra thứ tự mà các nút trên cây tìm kiếm được viếng?



# TÌM KIẾM THEO CHIỀU SÂU

- Sử dụng một cấu trúc dữ liệu, gọi là **Path**, để biểu diễn đường đi từ START đến trạng thái hiện tại.
  - Ví dụ: Path P = <START, d, e, r>
- Với mỗi nút trên đường đi, cần ghi nhớ những nút con nào vẫn còn có thể mở. Ví dụ, tại điểm bên dưới, ta có:




P = <START (expand=e , p),
d (expand = NULL),
e (expand = h),
r (expand = f) >

# TÌM KIẾM THEO CHIỀU SÂU

Gọi *P* = <START (expand = succs(START)>
While (*P* không rỗng và top(*P*) không phải là đích)
    if  expand của top(*P*) rỗng
    then
        loại bỏ top(*P*) (“lấy khỏi ngăn xếp”)
    else
        gọi *s* là một thành viên của expand của top(P)
        loại bỏ *s* ra khỏi expand của top(P)
        tạo một mục mới trên đỉnh của đường đi P:
            *s*(expand = succs(*s*))
If *P* is empty
    trả về FAILURE
Else
    trả về đường đi chứa các trạng thái trong *P*

Thuật toán này có thể viết dưới dạng đệ qui, dùng ngăn xếp để cài đặt *P*.

# ĐÁNH GIÁ 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 | $O(min(N,B^L))$ | $O(min(N,B^L))$ |
| UCS | Uniform Cost Search | C | C | $O(log(Q) * min(N,B^L))$ | $O(min(N,B^L))$ |
| DFS | Depth First Search | | | | |

# ĐÁNH GIÁ 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 | $O(min(N,B^L))$ | $O(min(N,B^L))$ |
| 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 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 | $O(min(N,B^L))$ | $O(min(N,B^L))$ |
| UCS | Uniform Cost Search | C | C | $O(log(Q) * min(N,B^L))$ | $O(min(N,B^L))$ |
| DFS | De... Sea... | | | | |

Giả sử Không gian Tìm kiếm không chu trình

# ĐÁNH GIÁ 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 | $O(min(N,B^L))$ | $O(min(N,B^L))$ |
| UCS | Uniform Cost Search | C | C | $O(log(Q) * min(N,B^L))$ | $O(min(N,B^L))$ |
| DFS | Depth First | C | K | $O(B^{LMAX})$ | $O(LMAX)$ |

Giả sử Không gian Tìm kiếm không chu trình

# CÂU HỎI SUY NGHĨ

- Làm thế nào ngăn chặn DFS lặp vô tận?

- Làm thế nào bắt buộc DFS đưa ra lời giải tối ưu?

# CÂU HỎI SUY NGHĨ

- Làm thế nào ngăn chặn DFS lặp vô tận?

Trả lời 1:
PC-DFS (Path Checking DFS)

- Làm thế nào bắt buộc DFS đưa ra lời giải tối ưu?

Trả lời 2:
MEMDFS (Memorizing DFS)

# CÂU HỎI SUY NGHĨ

- Làm thế nào ngăn chặn DFS lặp vô tận?

Trả lời 1:
PC-DFS (Path Checking DFS)
Không đệ qui một trạng thái nếu trạng thái này đã nằm trong đường đi hiện tại.

- Làm thế nào bắt buộc DFS đưa ra lời giải tối ưu?

Trả lời 2:
MEMDFS (Memorizing DFS)
Ghi nhớ mọi trạng thái đã mở.
Không mở hai lần.

# CÂU HỎI SUY NGHĨ

- Làm thế nào
  ngăn c
  lặ
  thế nào bắt
  buộc DFS đưa ra
  lời giải 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 đệ qui một trạng thái nếu trạng thái này đã nằm trong đường đi hiện tại.

**Trả lời 2:**

MEMDFS (Memorizing DFS)
Ghi nhớ mọi trạng thái đã mở.
Không mở hai lần.

# ĐÁNH GIÁ THUẬT TOÁN

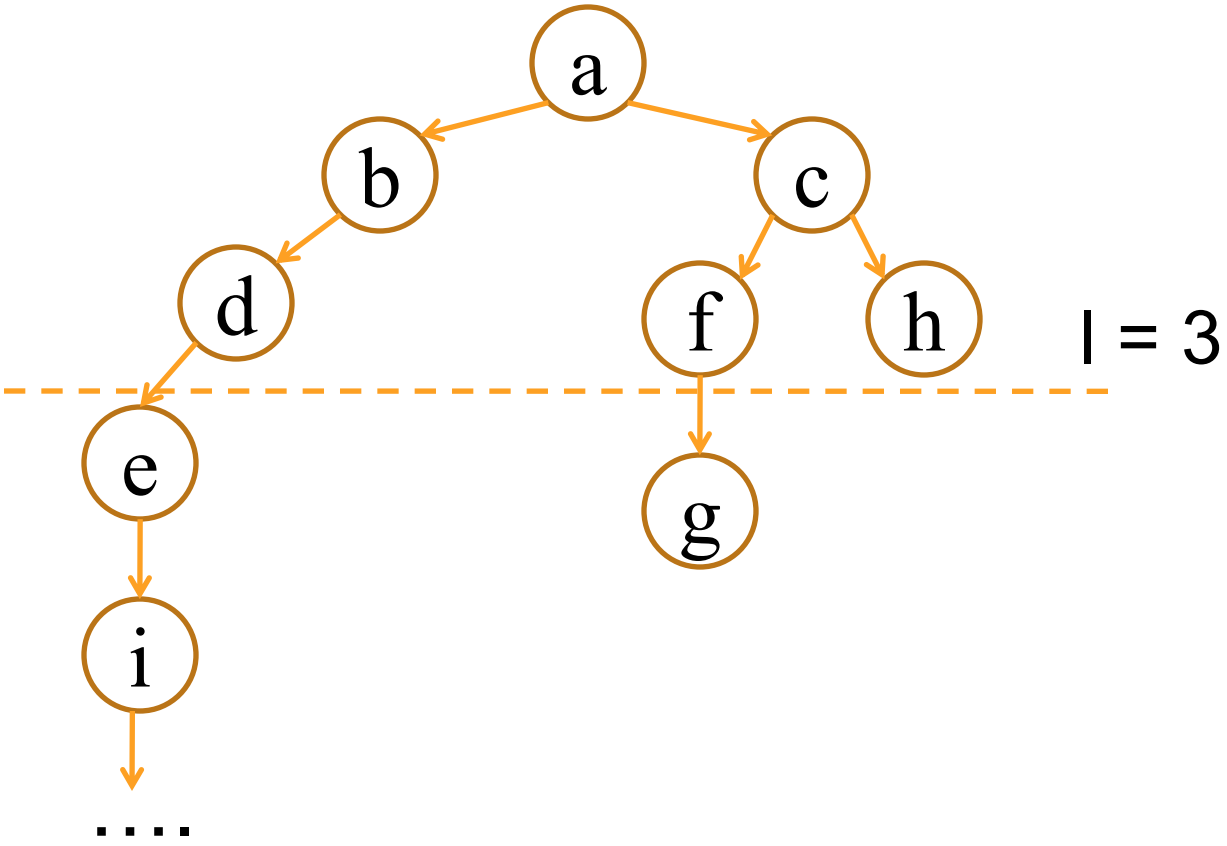
| | |
|---|---|
| N | số trạng thái trong bài toán |
| B | thừa số phân nhánh trung bình (số con trung bình) ($B>1$) |
| L | độ dài đường đi từ start đến goal với số bước (chi phí) ít nhất |
| LMAX | Độ dài đường đi **không chu trình** dài nhất từ start đến bất cứ đâu |
| Q | kích cỡ hàng đợi ưu tiên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

# TÌM KIẾM CHIỀU SÂU GIỚI HẠN

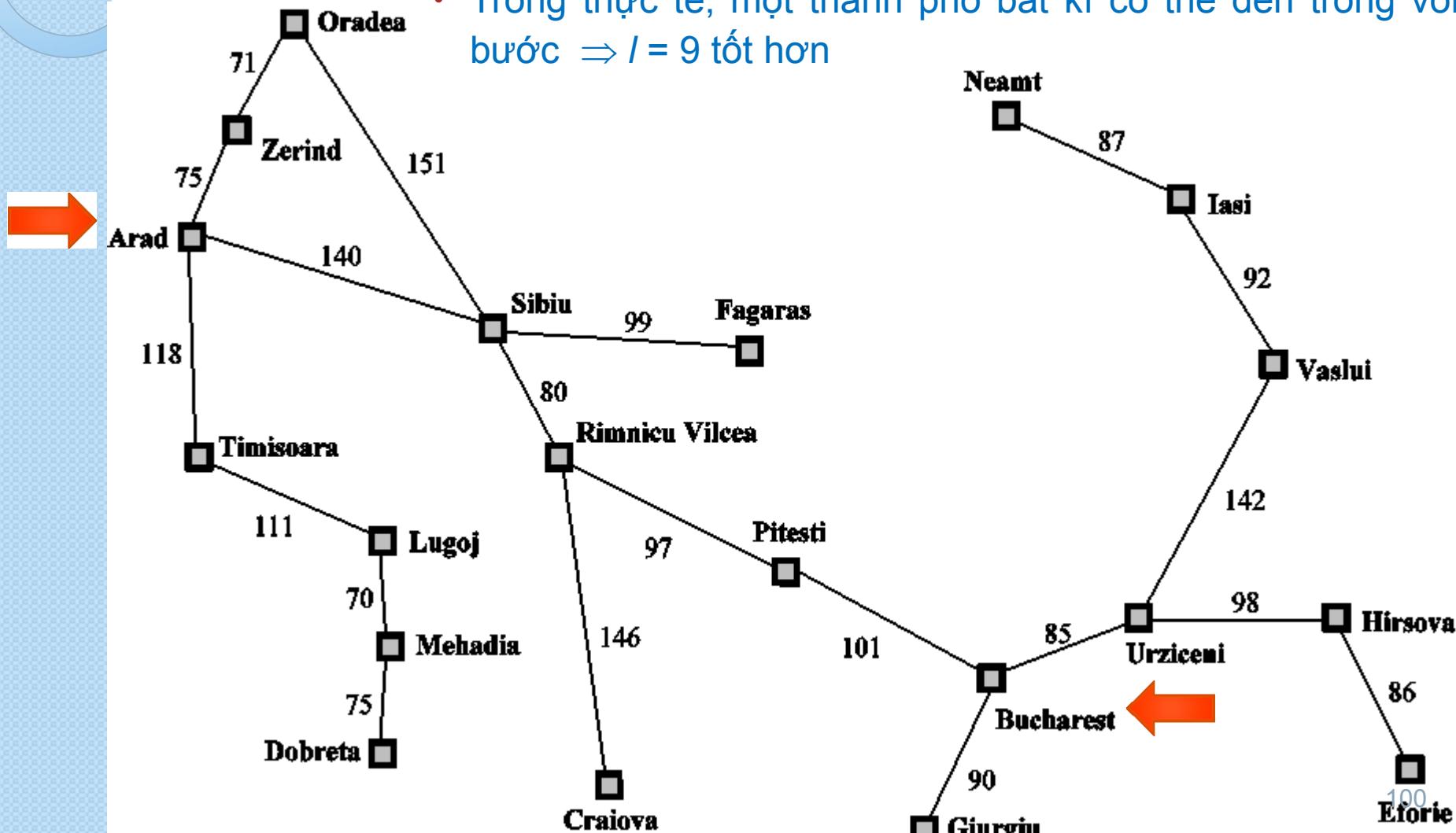
- Tránh trường hợp cây có đường đi sâu vô tận bằng cách đặt ra giới hạn độ sâu $l$.
  - Không đầy đủ nếu chọn $l < d$
  - Có thể không tối ưu nếu chọn $l > d$.
- Giới hạn độ sâu phụ thuộc vào tri thức nhận biết về bài toán.

# TÌM KIẾM CHIỀU SÂU GIỚI HẠN



# BÀI TẬP VÍ DỤ

- Bản đồ Romania có 20 thành phố $\Rightarrow l = 19$
- Trong thực tế, một thành phố bất kì có thể đến trong vòn bước $\Rightarrow l = 9$ tốt hơn



# BÀI TẬP VÍ DỤ

- Kết quả thực hiện bằng PC-DFS, chọn đỉnh đi tiếp theo thứ tự bảng chữ cái.

| STT | Đường đi | Danh sách mở của nút hiện tại |
|---|---|---|
| 1 | Arad | {Sibiu, Timisoara, Zerind} |
| 2 | Arad → Sibiu | {Faragas, Oradea, Riminicu Vilcea} |
| 3 | Arad → Sibiu → Faragas | {NULL} |
| 4 | Arad → Sibiu | {Oradea, Riminicu Vilcea} |
| 5 | Arad → Sibiu → Oradea | {Zerind} |
| 6 | Arad → Sibiu → Oradea → Zerind | {NULL} (Arad đang nằm trên đường đi} |
| 7 | Arad → Sibiu → Oradea | {NULL} |

# BÀI TẬP VÍ DỤ

- Kết quả thực hiện bằng PC-DFS, chọn đỉnh đi tiếp theo thứ tự bảng chữ cái.

| STT | Đường đi | Danh sách mở của nút hiện tại |
|---|---|---|
| 8 | Arad → Sibiu | {Riminicu Vilcea} |
| 9 | Arad → Sibiu → Riminicu Vilcea | {Craiova, Pitesti} |
| 10 | Arad → Sibiu → Riminicu Vilcea → Craiova | {NULL} |
| 11 | Arad → Sibiu → Riminicu Vilcea | {Pitesti} |
| 12 | Arad → Sibiu → Riminicu Vilcea → Pitesti | {Bucharest} |
| 13 | Arad → Sibiu → Riminicu Vilcea → Pitesti → Bucharest | GOAL. Chi phí: 418 |

Arad → Sibiu → Riminicu Vilcea → Pitesti → Bucharest. Chi phí: 418

# TÌM KIẾM LẶP SÂU DẦN
# (Iterative Deepening Search – IDS)

Biên soạn: ThS. Nguyễn Ngọc Thảo

# TÌM KIẾM LẶP SÂU DẦN

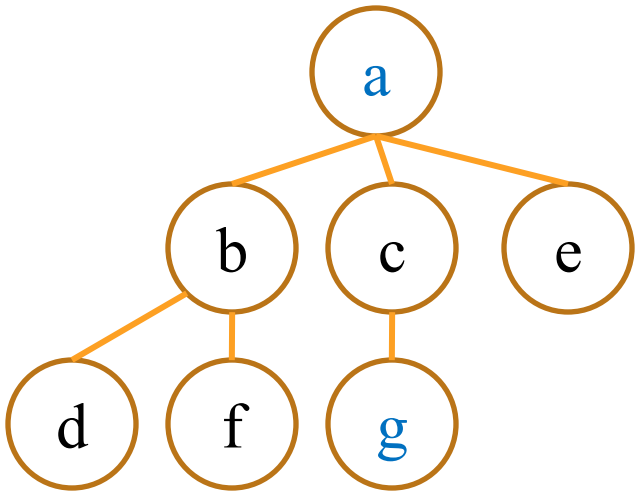
- Lặp sâu dần là một thuật toán đơn giản sử dụng DFS làm thủ tục con:
  1. Thực hiện DFS chỉ tìm các đường đi có độ dài $\leq 1$ (bỏ qua mọi đường đi có độ dài 2).
  2. Nếu "1" thất bại, thực hiện DFS chỉ tìm các đường đi có độ dài $\leq 2$.
  3. Nếu "2" thất bại, thực hiện DFS chỉ tìm các đường đi có độ dài $\leq 3$.
- ….và tiếp tục cho đến khi thành công.

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

# Ví dụ - không chu trình

- DFS: a b d f c g e

- IDFS
  - Sâu 1: a
  - Sâu 2: a b c e
  - Sâu 3 a b d f c g e
  - Lặp tới độ sâu ngắn nhất tới đích



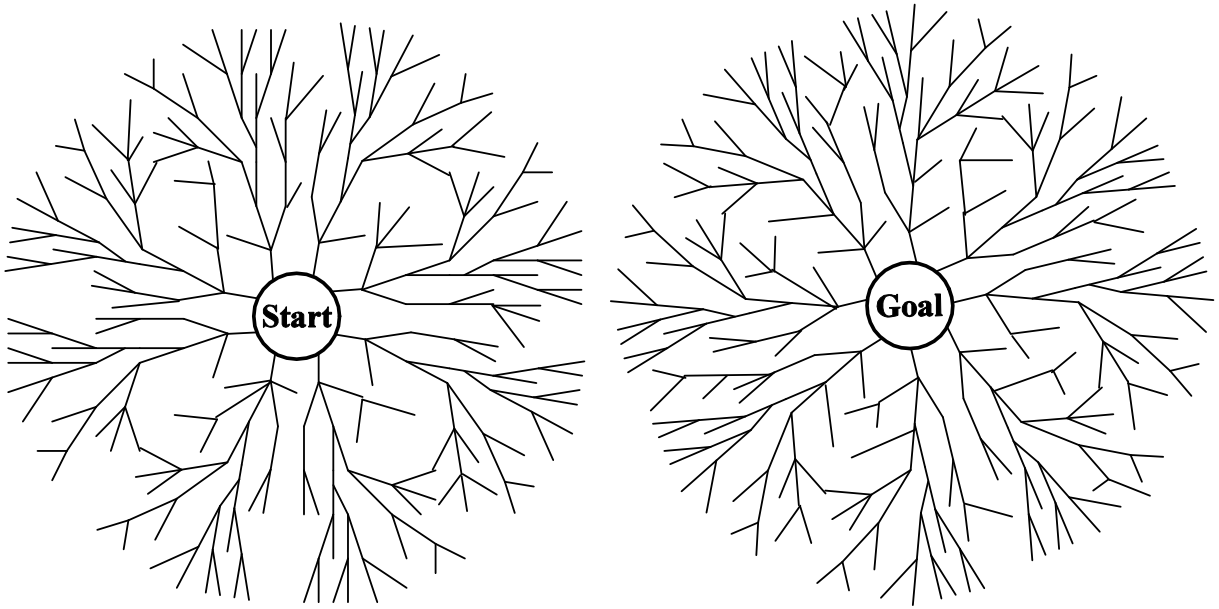
# ĐÁNH GIÁ THUẬT TOÁN

| | |
|---|---|
| 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 | $O(min(N,B^L))$ | $O(min(N,B^L))$ |
| 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 | C | (1) | $O(B^L)$ | $O(L)$ |

# TÌM KIẾM HAI CHIỀU

- Chạy đồng thời hai lượt tìm kiếm, một lượt đi tới từ trạng thái bắt đầu và một lượt đi lui từ trạng thái đích, dừng khi cùng gặp nhau tại một nút.



# TỔNG KẾT

- Nắm vững ý tưởng của các thuật toán tìm kiếm mù
- Biết các khái niệm và tiêu chí để đánh giá thuật toán: tính đầy đủ, tính tối ưu, độ phức tạp về thời gian và không gian.

# TÀI LIỆU THAM KHẢO

- Tài liệu bài giảng môn học
- Moore's tutorial:

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

- Chapter 3. S. Russel and P.Norvig, Artificial Intelligence – A Modern Approach. *Third Edition.* 2010

# KẾT THÚC CHỦ ĐỀ