Trí tuệ nhân tạo

Tuần 3

Giảng viên: Trần Đức Minh

Nội dung trình bày

- Chiến lược tìm kiếm với độ sâu giới hạn
- Chiến lược tìm kiếm với độ sâu lặp
- Chia để trị
- Xây dựng đồ thị Và/Hoặc

Chiến lược tìm kiếm với độ sâu giới hạn

- **Chiến lược tìm kiếm với độ sâu giới hạn** chính là thuật toán DFS nhưng chỉ áp dụng đến một độ sâu xác định trước.
- Chiến lược này được đưa ra nhằm khắc phục nhược điểm của thuật toán DFS đó là nếu cây tìm kiếm chứa nhánh vô hạn thì ta có thể bị mắc kẹt ở nhánh đó và không thể tìm ra kết quả của bài toán.

Thuật toán tìm kiếm với độ sâu giới hạn

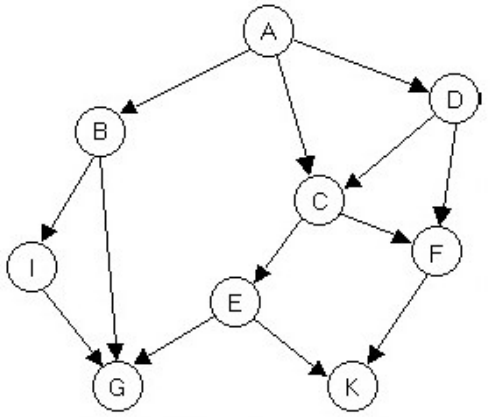
- Ta sử dụng stack **open** (có cấu trúc LIFO) để lưu giữ các trạng thái được sinh ra nhưng chưa được khảo sát.
- Hàm **father(X)** dùng để lưu lại cha của đỉnh X trên đường đi.
  - Ví dụ: Cú pháp **father(X)** ← **Y** tức là đỉnh cha của đỉnh X là đỉnh Y
- Hàm **depth(T)** dùng để đánh độ sâu cho trạng thái T nào đó.
  - Ví dụ: **depth(T)** ← **2** tức là độ sâu của trạng thái T là 2
- Xác định trạng thái bắt đầu **S**
- Xác định tập các trạng thái kết thúc **GD**
- Xác định các toán tử chuyển trạng thái

Thuật toán tìm kiếm với độ sâu giới hạn

- **DEPTH-LIMITED-SEARCH(d)**
  - Đưa trạng thái bắt đầu **S** vào **open**;
  - depth(S) ← 0;
  - **loop do**
    - Nếu **open =** ∅ thì thông báo Không tìm thấy kết quả. **Kết thúc thuật toán**.
    - Lấy 1 trạng thái từ stack **open** đưa vào biến trạng thái **X**, sau đó loại bỏ trạng thái đó khỏi stack.
    - Nếu **X** ∈ **GD** thì thông báo tìm kiếm thành công. **Kết thúc thuật toán**.
    - Nếu **X** ∉ **GD và depth(X) <= d**
      - Với mỗi trạng thái Y_i được sinh ra bởi trạng thái **X** thông qua các toán tử
        - Đưa trạng thái Y_i vào stack **open**
        - **father(Y_i) ← X**
        - **depth(Y_i) ← depth(X) + 1**
  - **end loop**

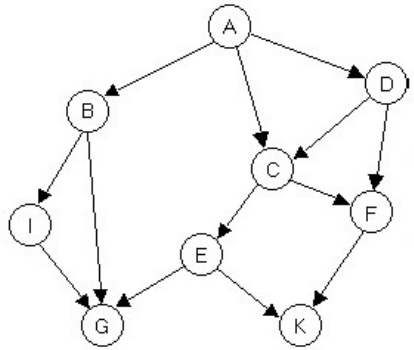
Thuật toán tìm kiếm với độ sâu giới hạn

- Ví dụ 1: Sử dụng thuật toán tìm kiếm với độ sâu giới hạn để duyệt đồ thị bên dưới. Biết rằng
  - A là trạng thái bắt đầu
  - F là trạng thái kết thúc
  - Độ sâu giới hạn bằng 2
  - Đầu mũi tên chỉ trạng thái được sinh bởi trạng thái ở đuôi mũi tên.



Thuật toán tìm kiếm với độ sâu giới hạn

- open=[A]; depth(A) = 0;
- X = A; open=[B,C,D]; father[B,C,D] = A; depth[B,C,D] = 1;
- X = B; open=[I,G,C,D]; father[I,G] = B; depth[I,G] = 2;
- X = I; open=[G^I,G^B,C,D]; father[G^I] = I; depth(G^I) = 3;
- X = G^I; open=[G^B,C,D]; ***// Bị loại do chiều sâu > 2***
- X = G^B; open=[C,D];
- X = C; open=[E,F,D]; father[E,F] = C; depth[E,F] = 2;
- X = E; open=[G^E,K,F,D]; father[G^E,K] = E; depth[G^E,K] = 3;
- X = G^E; open=[K,F,D]; ***// Bị loại do chiều sâu > 2***
- X = K; open=[F,D]; ***// Bị loại do chiều sâu > 2***
- X = F; => Tìm kiếm thành công



Thuật toán tìm kiếm với độ sâu giới hạn

- **Nhận xét**
  - Chiến lược này kết hợp được các ưu điểm của các chiến lược tìm kiếm theo bề rộng và tìm kiếm theo chiều sâu.
  - Tuy nhiên, nếu kết quả nằm ở sâu hơn so với độ sâu d thì ta sẽ không tìm thấy kết quả này.
- **Đánh giá**
  - Tính đầy đủ: Lời giải bài toán luôn tìm được nếu b (độ phân nhánh của cây) là hữu hạn và ta chọn độ sâu MAX đủ lớn.
  - Độ phức tạp về thời gian:
    - $b^0 + b^1 + b^2 + \ldots + b^d = O(b^d)$
  - Độ phức tạp về không gian lưu trữ: O(b.d)
  - Tính tối ưu: Thuật toán luôn tìm ra lời giải trong độ sâu MAX (nếu có) nhưng chưa chắc đã ít trạng thái trung gian nhất (đường đi từ trạng thái bắt đầu đến trạng thái kết thúc).

Chiến lược tìm kiếm với độ sâu lặp

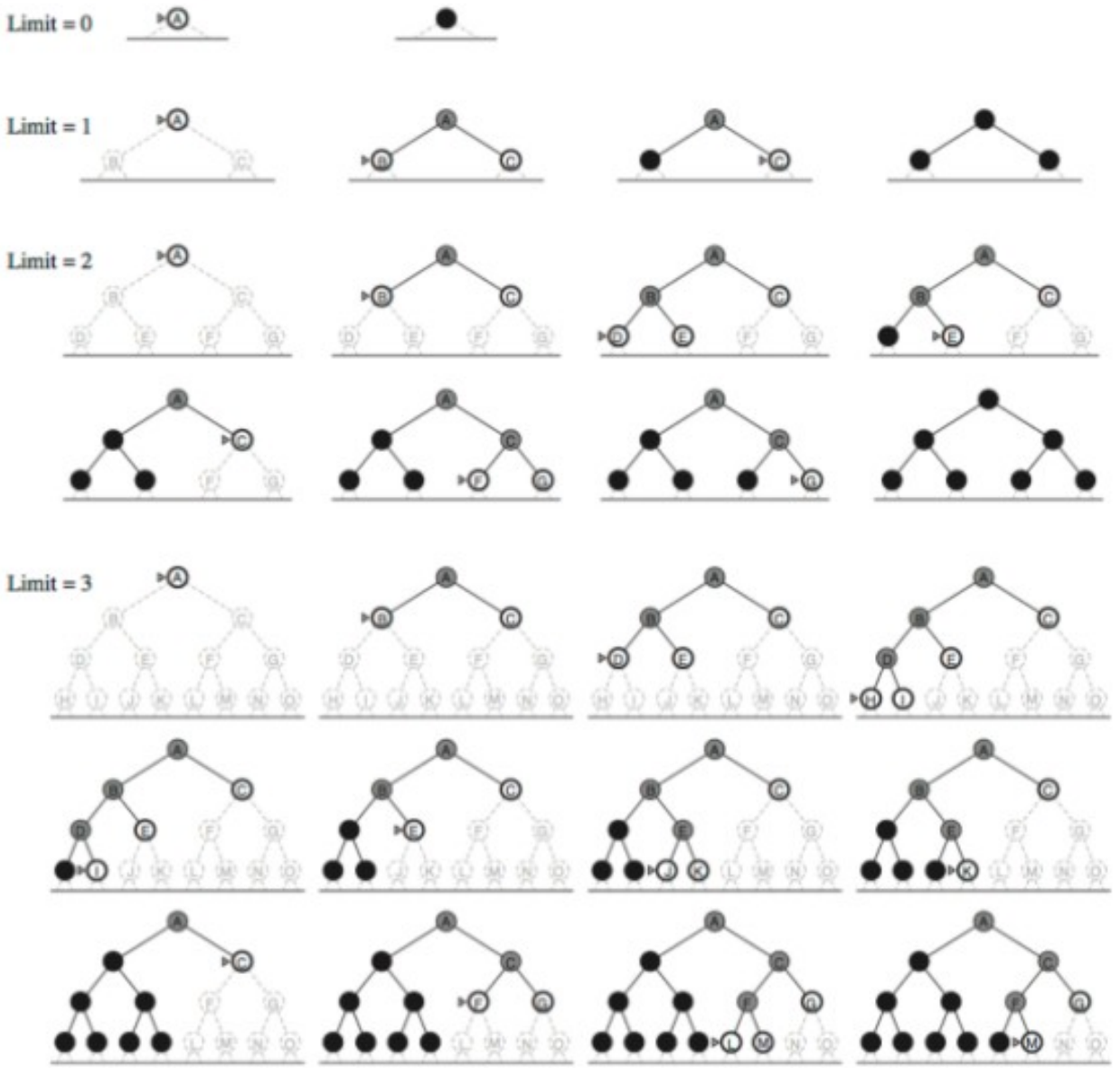
- **Chiến lược tìm kiếm với độ sâu lặp** sử dụng chiến lược tìm kiếm độ sâu giới hạn đến một **độ sâu d** nào đó. Nếu không tìm ra kết quả, chiến lược tiếp tục tăng độ sâu lên **d+1** rồi lại tiếp tục tìm kiếm với độ sâu giới hạn **d+1**. Quá trình trên được lặp lại cho đến độ sâu **max** nào đó.

Thuật toán tìm kiếm với độ sâu lặp

- Xác định độ sâu tối đa **MAX**

- **DEPTH-DEEPENING-SEARCH**
  - For **d** ←**0** to **MAX** do
    - DEPTH-LIMITED-SEARCH(d)
    - Nếu tìm kiếm thành công thì **Kết thúc thuật toán**

Thuật toán tìm kiếm với độ sâu lặp



Thuật toán tìm kiếm với độ sâu lặp

- **Nhận xét**
  - Chiến lược này kết hợp được các ưu điểm của các chiến lược tìm kiếm theo bề rộng và tìm kiếm theo chiều sâu.
    - Cũng như tìm kiếm theo bề rộng, tìm kiếm sâu lặp luôn tìm ra kết quả (nếu bài toán có kết quả), miễn là ta chọn độ sâu MAX đủ lớn.
    - Tìm kiếm sâu lặp chỉ cần không gian nhớ như tìm kiếm theo độ sâu.
    - Trong tìm kiếm sâu lặp, ta phải phát triển lặp lại nhiều lần cùng một trạng thái. Điều đó làm cho ta có cảm giác rằng, tìm kiếm sâu lặp lãng phí nhiều thời gian. Thực ra thời gian tiêu tốn cho phát triển lặp lại các trạng thái là không đáng kể so với thời gian tìm kiếm theo bề rộng.
      - Ví dụ: với b = 10 và d = 5
        - Số lượng node được sinh trong quá trình tìm kiếm với độ sâu giới hạn là: 1 + 10 + 100 + 1,000 + 10,000 + 100,000 = 111,111
        - Số lượng node được sinh trong quá trình tìm kiếm với độ sâu lặp là: 6 + 50 + 400 + 3,000 + 20,000 + 100,000 = 123,456

Thuật toán tìm kiếm với độ sâu lặp

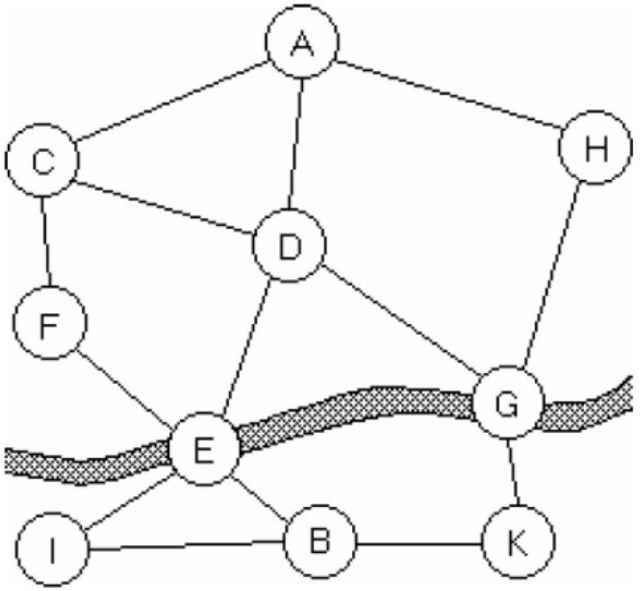
- **Đánh giá**
  - Tính đầy đủ: Lời giải bài toán luôn tìm được nếu b (độ phân nhánh của cây) là hữu hạn và ta chọn độ sâu MAX đủ lớn.
  - Độ phức tạp về thời gian:
    - $(d+1)b^0 + db^1 + (d-1)b^2 + \ldots + b^d = O(b^d)$
  - Độ phức tạp về không gian lưu trữ: O(b.d)
  - Tính tối ưu: Thuật toán luôn tìm ra lời giải với ít trạng thái trung gian nhất (đường đi từ trạng thái bắt đầu đến trạng thái kết thúc).

Chia để trị

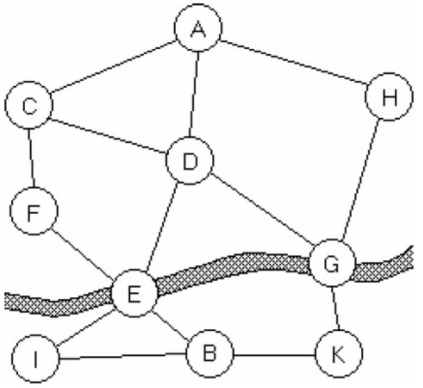
- **Chia để trị** là phương pháp giải quyết một vấn đề phức tạp bằng cách **quy vấn đề phức tạp đó về các vấn đề con**.
- Các vấn đề con này lại có thể tiếp tục chia thành các vấn đề con khác nữa. Và việc này được lặp đi lặp lại nhiều lần cho đến khi các vấn đề này có thể giải quyết được.

Chia để trị

- Ví dụ: Tìm đường đi từ điểm A đến điểm B trong bản đồ giao thông. Trong thành phố có 2 chiếc cầu để đi qua sông là điểm E và điểm G.
  - Đường đi từ điểm A đến điểm B có thể được quy về **2 phương án**
    - Đi từ A đến B qua E
    - Đi từ A đến B qua G

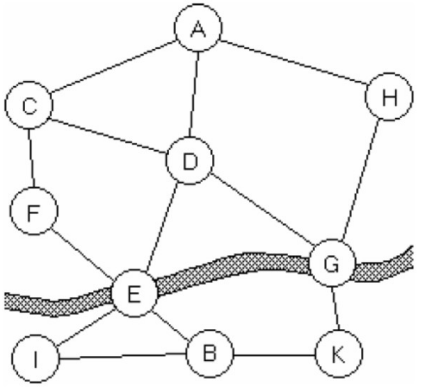


Chia để trị



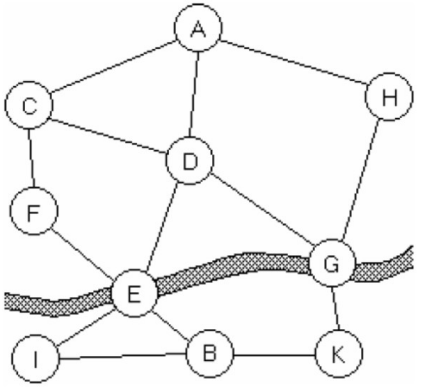
- Ví dụ:
  - Phương án đi từ A đến B qua E được chia thành 2 phương án nhỏ khác
    - Tìm đường đi từ A đến E
    - **VÀ** tìm đường đi từ E đến B
  - Phương án đi từ A đến B qua G cũng được chia thành 2 phương án nhỏ khác
    - Tìm đường đi từ A đến G
    - **VÀ** tìm đường đi từ G đến B

Chia để trị

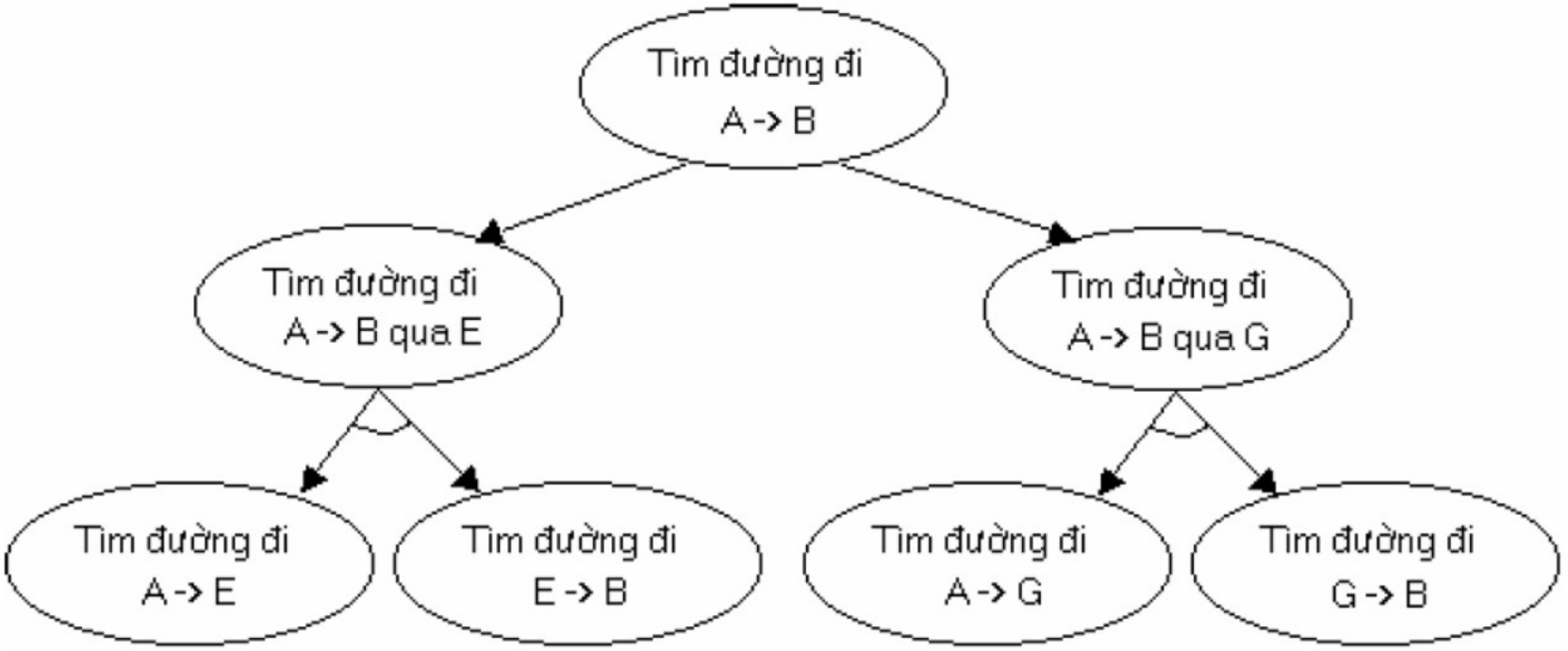


- Ví dụ:
  - Phương án đi từ A đến B qua E được chia thành 2 phương án nhỏ khác
    - Tìm đường đi từ A đến E
    - **VÀ** tìm đường đi từ E đến B
  - Phương án đi từ A đến B qua G cũng được chia thành 2 phương án nhỏ khác
    - Tìm đường đi từ A đến G
    - **VÀ** tìm đường đi từ G đến B

Chia để trị



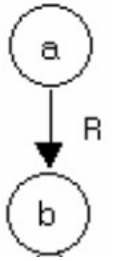
- Ví dụ:
  - Các quá trình chia nhỏ của ví dụ có thể được biểu diễn bằng một đồ thị có tên là **Đồ thị Và/Hoặc**



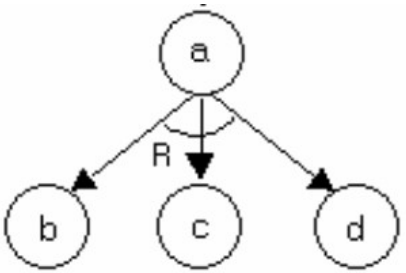
Xây dựng đồ thị Và/Hoặc

- Mỗi phương án ứng với một đỉnh của đồ thị
- Nếu một toán tử quy một phương án về một phương án khác, chẳng hạn

  R : a → b, thì trong đồ thị sẽ có **cung gán nhãn đi từ đỉnh a tới đỉnh b**.



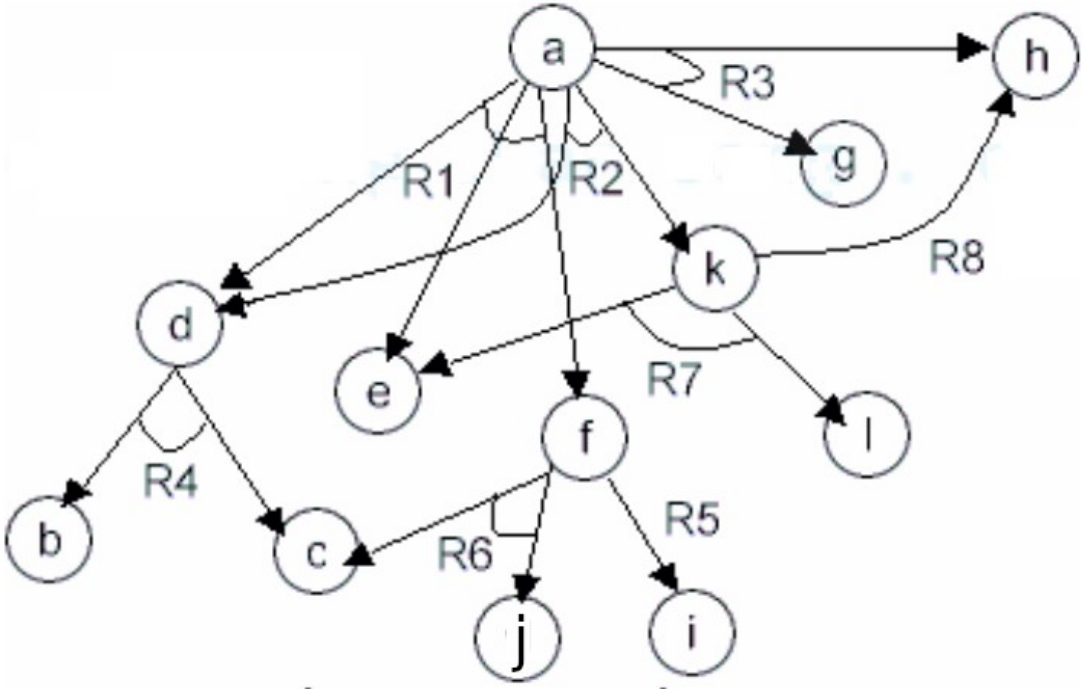
- Nếu một toán tử quy một phương án về một số phương án con, chẳng hạn
  - **R : a → b, c, d** thì ta biểu diễn tập các phương án con **{b, c, d}** và toán tử **R** như hình bên và nói
    - **b, c, d** là các **đỉnh kề** với **đỉnh a** theo **toán tử R**.



Xây dựng đồ thị Và/Hoặc

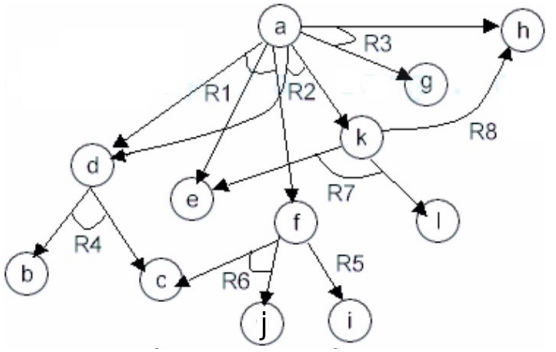
- Ví dụ: Xây dựng đồ thị và/hoặc cho không gian trạng thái sau
  - Trạng thái ban đầu là **a**.
    - Đây là phương án gốc cần giải quyết
  - Tập các toán tử quy về các phương án con như sau
    - **R1 : a → d, e, f**
    - **R2 : a → d, k**
    - **R3 : a → g, h**
    - **R4 : d → b, c**
    - **R5 : f → i**
    - **R6 : f → c, j**
    - **R7 : k → e, l**
    - **R8 : k → h**
  - Tập các trạng thái kết thúc (các bài toán sơ cấp) là **T = { b, c, e, j, l }**

Xây dựng đồ thị Và/Hoặc

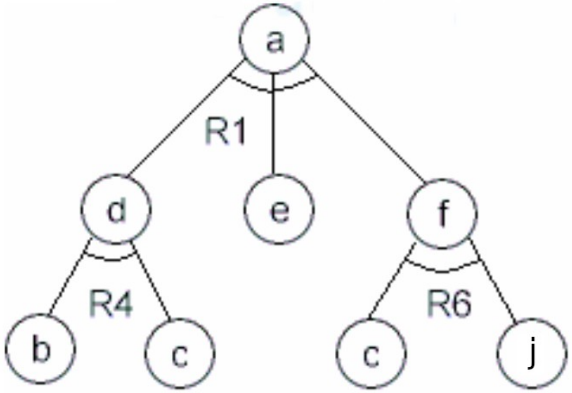


- Trong đồ thị này ta nói
  - **Các đỉnh d, e, f kề với đỉnh a theo toán tử R1**.
  - Các đỉnh c, j kề với đỉnh f theo toán tử R6.
  - …

Xây dựng đồ thị Và/Hoặc

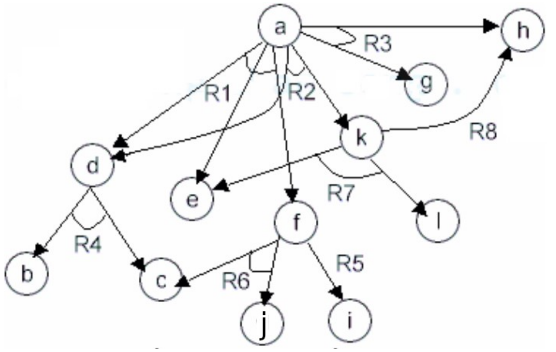


- Bằng cách áp dụng liên tiếp các toán tử, ta có thể đưa phương án cần giải quyết về một tập các phương án con.
- Ví dụ:
  - Nếu ta áp dụng các toán tử R1, R4, R6 thì phương án a sẽ được quy về các phương án con b, c, e, j và đây là các trạng thái kết thúc. Do đó ta sẽ xây dựng được một **cây nghiệm**.

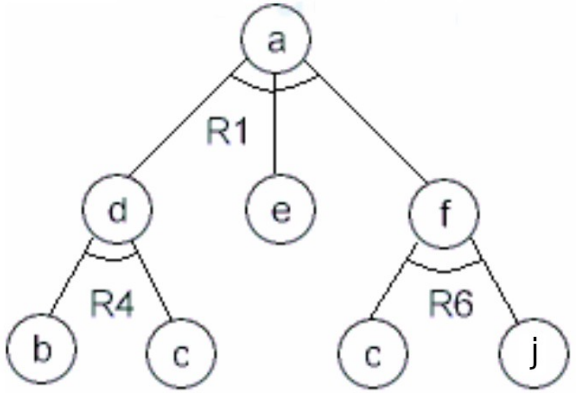


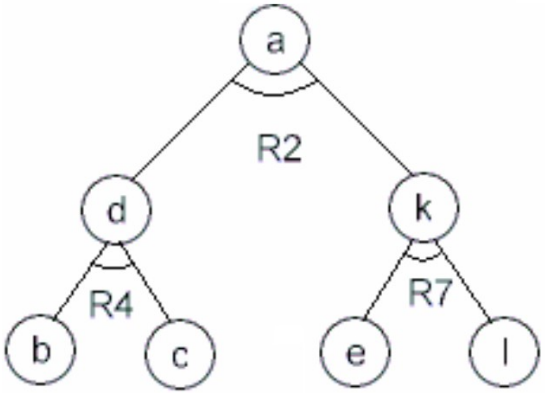
Xây dựng đồ thị Và/Hoặc

Cây nghiệm

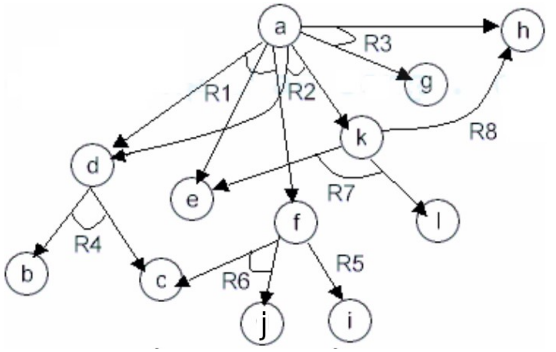


- **Cây nghiệm** là một cây trong đó
  - Gốc của cây ứng với phương án cần giải.
  - Tất cả các lá của cây là các đỉnh kết thúc (đỉnh ứng với các trạng thái kết thúc hay bài toán sơ cấp).
  - Nếu u là đỉnh trong của cây, thì các đỉnh con của u là các đỉnh kề u theo một toán tử nào đó.
- Ví dụ cây nghiệm



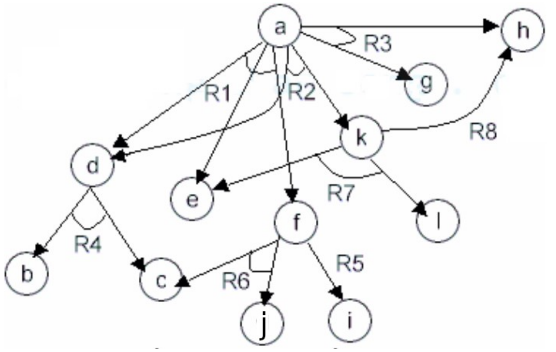


Xây dựng đồ thị Và/Hoặc



- **Đỉnh giải được** được xác định theo cách đệ quy như sau
  - Các đỉnh kết thúc (đỉnh ứng với các trạng thái kết thúc hay bài toán sơ cấp) là các đỉnh giải được.
  - Nếu u không phải là đỉnh kết thúc, nhưng có một toán tử R sao cho tất cả các đỉnh kề u theo R đều giải được thì u giải được.
- Ví dụ
  - Đỉnh b, c, e, j, l là các đỉnh kết thúc nên là các đỉnh giải được.
  - Đỉnh f là đỉnh giải được do đỉnh c, j kề với đỉnh f theo toán R6 là các đỉnh giải được.

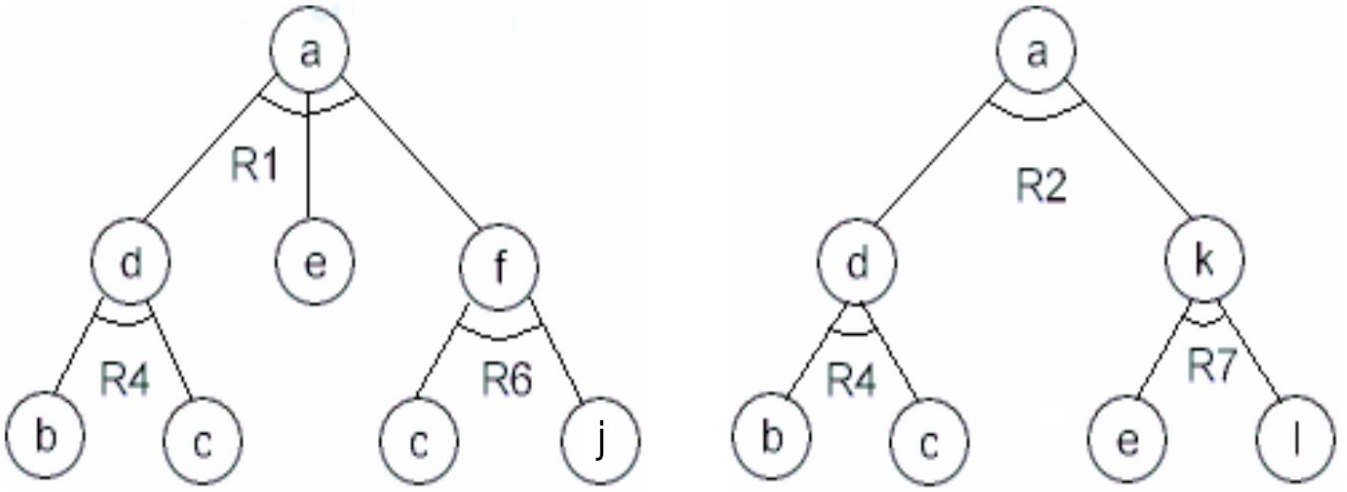
Xây dựng đồ thị Và/Hoặc



- **Đỉnh không giải được** được xác định theo cách đệ quy như sau
  - Các đỉnh không phải là đỉnh kết thúc và không có đỉnh kề, là các đỉnh không giải được.
  - Nếu u không phải là đỉnh kết thúc và với mọi toán tử R áp dụng được tại u đều có ít nhất một đỉnh v kề với u theo R không giải được thì u không giải được.
- Ví dụ
  - Đỉnh g, h là các đỉnh không giải được do chúng không thuộc tập đỉnh kết thúc và không có đỉnh kề.
  - Đỉnh a là đỉnh không giải được nếu áp dụng toán tử R3, do đỉnh g kề với đỉnh a theo toán tử R3 là đỉnh không giải được.

Xây dựng đồ thị Và/Hoặc

- Nhận xét
  - Một phương án giải được có thể có nhiều cây nghiệm và mỗi cây nghiệm sẽ là một cách giải bài toán đó.



Tìm kiếm trên đồ thị Và/Hoặc

- **Tìm kiếm trên đồ thị và/hoặc** nhằm xác định đỉnh ứng với phương án ban đầu là **giải được hay không giải được**, và nếu đỉnh đó là giải được thì **xây dựng cây nghiệm cho nó**.
- Ta sẽ sử dụng chiến lược tìm kiếm theo chiều sâu trên đồ thị và/hoặc để tìm lời giải cho bài toán.

Thuật toán tìm kiếm trên đồ thị Và/Hoặc

- Xây dựng hàm đệ quy **Solvable(u)**.
  - Hàm nhận giá trị **true** nếu u giải được và **false** trong trường hợp u không giải được.
- Trong hàm **Solvable(u)** có các thành phần quan trọng sau:
  - **Biến OK:** Với mỗi toán tử R áp dụng được tại u, biến OK nhận giá trị **true** nếu tất cả các đỉnh v kề u theo R đều giải được, và OK nhận giá trị **false** nếu có một đỉnh v kề u theo R không giải được.
  - **Hàm Operator(u):** Hàm ghi lại toán tử áp dụng thành công tại u, tức là Operator(u) = R nếu mọi đỉnh v kề với u theo R đều giải được.

Thuật toán tìm kiếm trên đồ thị Và/Hoặc

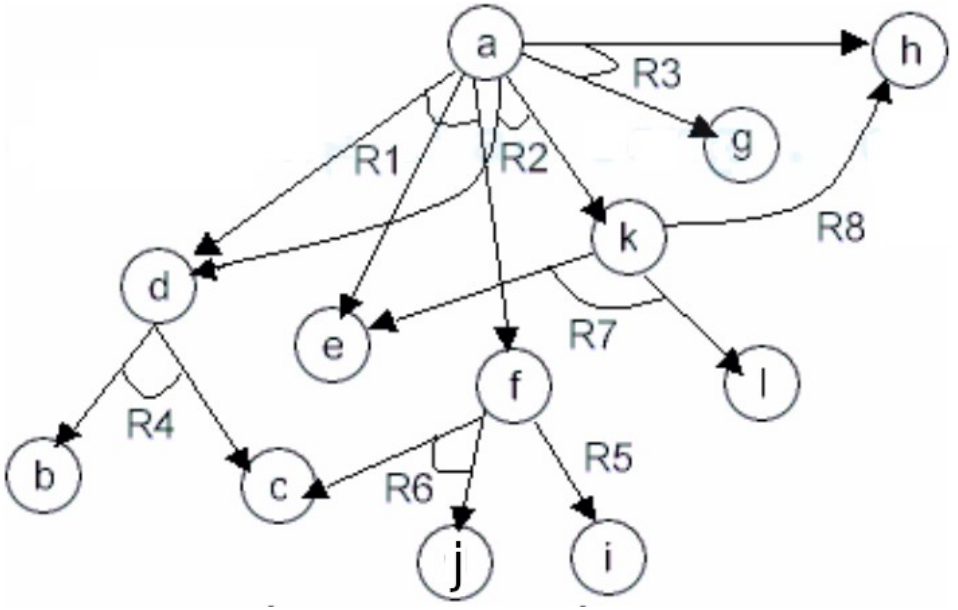
- **Solvable(u)**
  - **Nếu** u là đỉnh kết thúc **thì**
    - **return true;**
  - **Nếu** u không là đỉnh kết thúc và không có đỉnh kề **thì**
    - **return false;**
  - **for** mỗi toán tử R áp dụng được tại u **do**
    - OK ← true;
    - **for** mỗi đỉnh v kề với u theo R **do**
      - Nếu **Solvable(v)** = false thì
        - OK ← false;
        - **break**;
    - **Nếu** OK = true **thì**
      - Operator(u) ← R;
      - **return true;**
  - **return false;**

Thuật toán tìm kiếm trên đồ thị Và/Hoặc

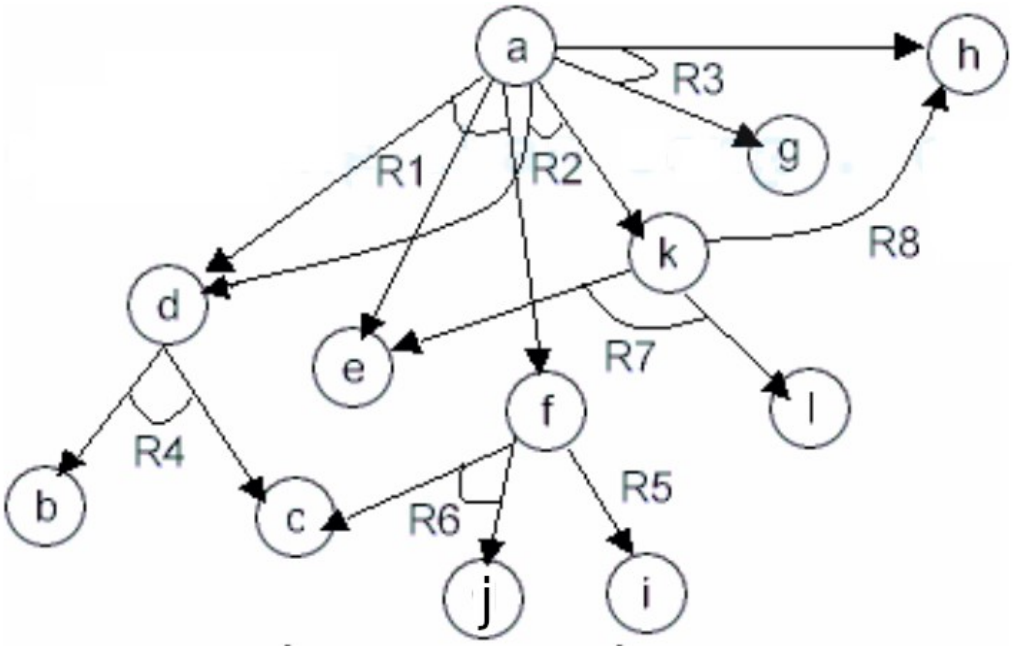
- Nhận xét
  - Thuật toán tìm kiếm theo độ sâu trên đồ thị và/hoặc sẽ xác định được phương án ban đầu là giải được hay không giải được, nếu cây tìm kiếm không có nhánh vô hạn. Nếu cây tìm kiếm có nhánh vô hạn thì chưa chắc thuật toán đã dừng. Trong trường hợp này ta nên sử dụng thuật toán tìm kiếm sâu lặp.
  - Nếu bài toán ban đầu giải được, thì bằng cách sử dụng hàm Operator ta sẽ xây dựng được cây nghiệm.

Thuật toán tìm kiếm trên đồ thị Và/Hoặc

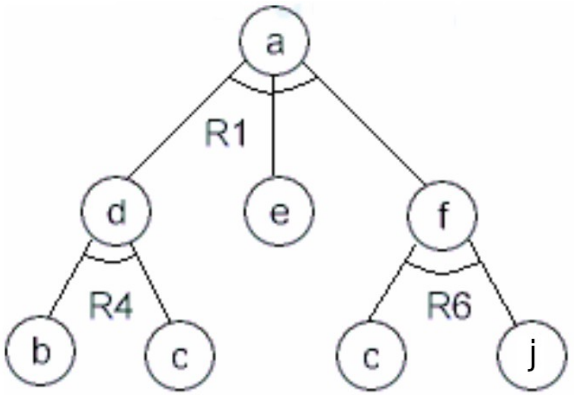
- Ví dụ:
  - Chạy hàm Solvable(a) để kiểm tra đỉnh a là đỉnh giải được hay không giải được trên đồ thị và/hoặc bên dưới. Sau đó xây dựng cây nghiệm.
    - Tập các trạng thái kết thúc (các bài toán sơ cấp) là **T = { b, c, e, j, l }**



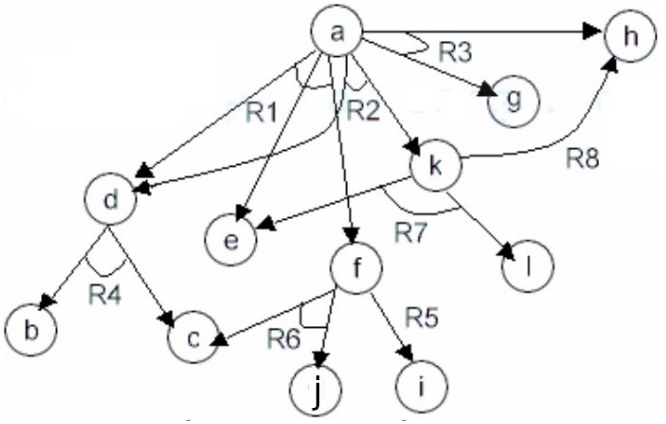
Thuật toán tìm kiếm trên đồ thị Và/Hoặc



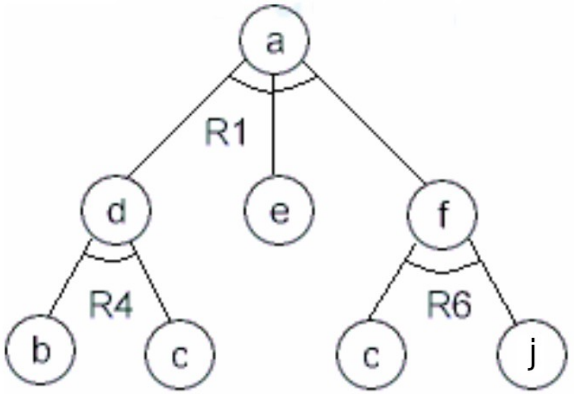
- Ví dụ:
  - Solvable(a) ← true bởi
    - Toán tử R1:
      - Solvable(d) ← true bởi
        - Toán tử R4:
          - Solvable(b) ← true
          - Solvable(c) ← true
        - Operator(d) ← R4
      - Solvable(e) ← true
      - Solvable(f) ← true bởi
        - Toán tử R5:
          - Solvable(i) ←false
        - Toán tử R6:
          - Solvable(c) ← true
          - Solvable(j) ← true
        - Operator(f) ← R6
    - Operator(a) ← R1



Thuật toán tìm kiếm trên đồ thị Và/Hoặc



- Ví dụ tìm cây nghiệm:
  - Operator(a) ← R1 và d, e, f kề với đỉnh a theo toán tử R1.
    - Operator(d) ← R4 và b, c kề với đỉnh d theo toán tử R4
      - b, c là các đỉnh kết thúc
    - e là đỉnh kết thúc
    - Operator(f) ← R6 và c, j kề với đỉnh f theo toán tử R6
      - c, j là các đỉnh kết thúc



Hết Tuần 3

Cảm ơn các bạn đã chú ý lắng nghe !!!