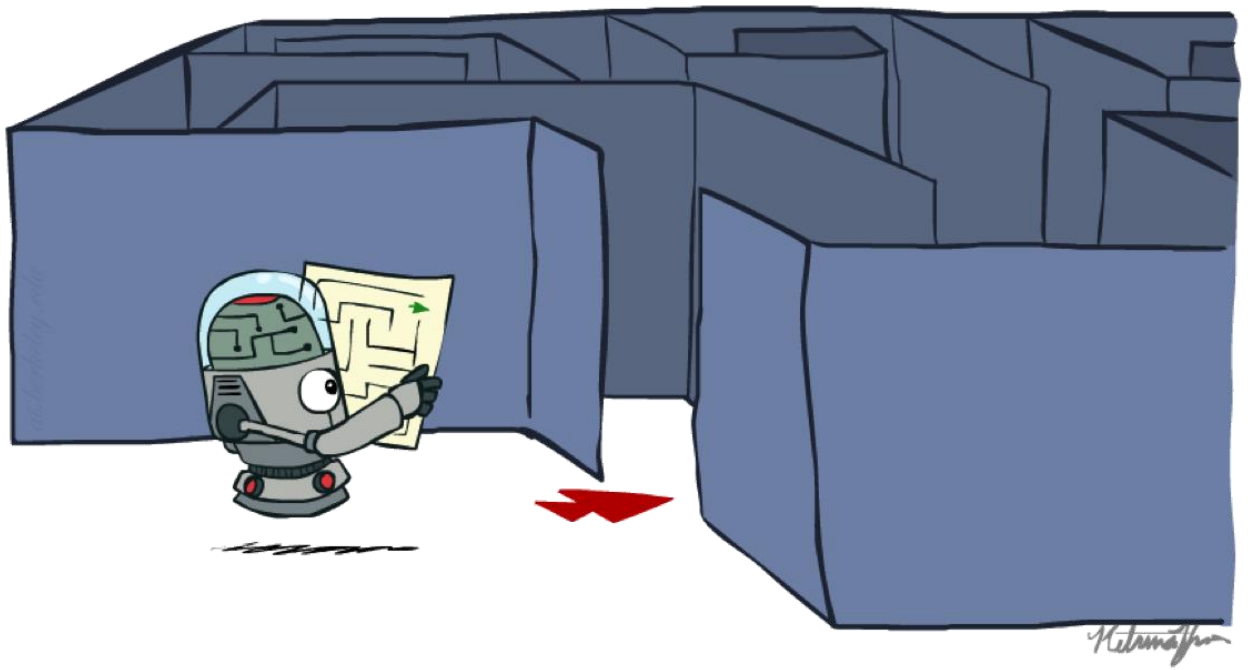
CHƯƠNG 2:

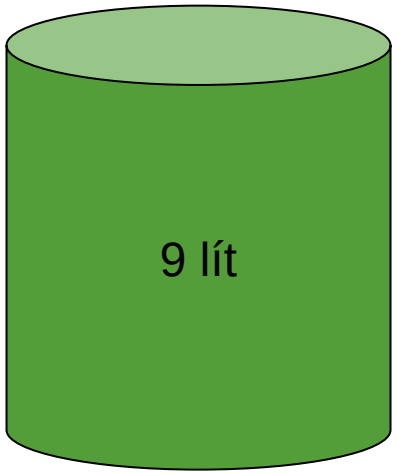
GIẢI QUYẾT VẤN ĐỀ BẰNG TÌM KIẾM



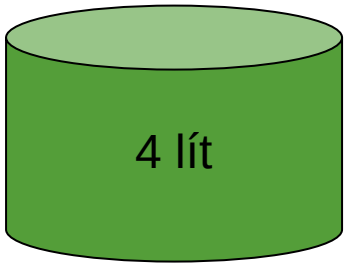
NỘI DUNG

1. Giới thiệu một số bài toán
   - Xác định các thành phần của bài toán
   - Biểu diễn bài toán
   - Giải pháp cho các bài toán
2. Kỹ thuật tìm kiếm mù
   - Tìm kiếm rộng (breath-first search)
   - Tìm kiếm sâu (depth-first search)
   - Tìm kiếm theo độ sâu có giới hạn (depth-limited search)
   - Tìm kiếm theo chiều sâu lặp (iterative deepening depth-first search)
   - Tìm kiếm hai chiều (Bidirectional search)
   - Tìm kiếm với giá đồng nhất (Uniform-cost search)

Ví dụ: Bài toán đong nước



a



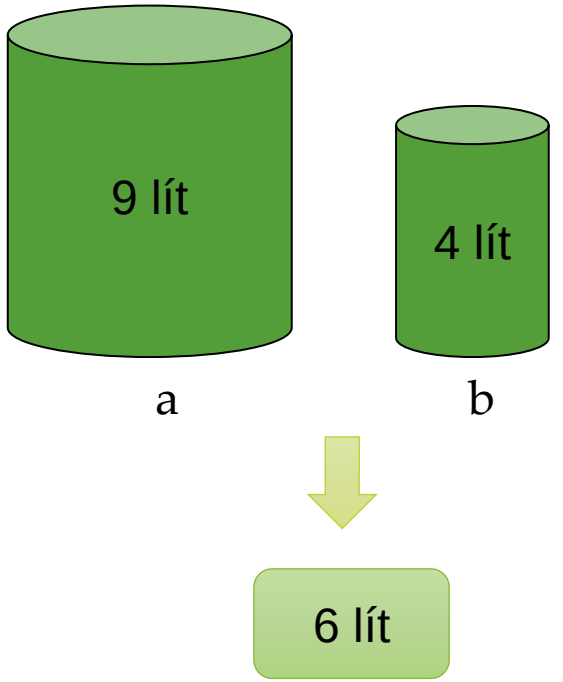
b

Cho 2 bình 9 lít và 4 lít, không có vạch chia, và 1 vòi bơm. Làm cách nào đong được 6 lít?

Ví dụ: Bài toán đong nước

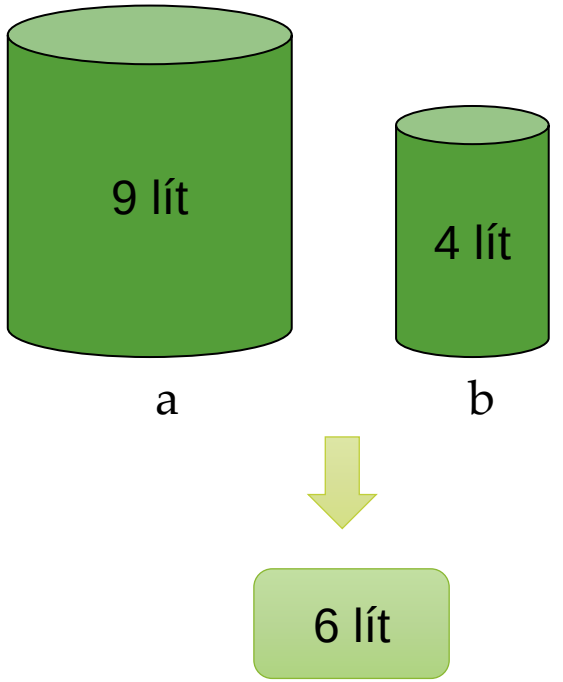
Giải pháp:

	a	b	
Start	0	0	
1	9	0	
2	5	4	
3	5	0	
4	1	4	
5	1	0	
6	0	1	
7	9	1	
8	6	4	Goal



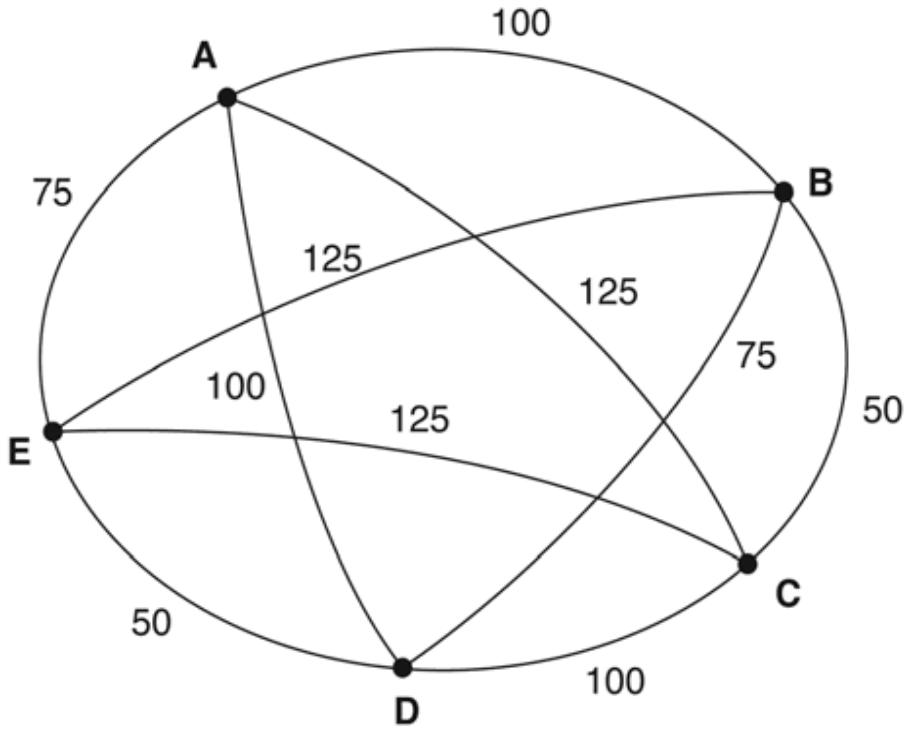
Ví dụ: Bài toán đong nước

- Xác định các thành phần của bài toán:
  - Trạng thái đầu: (0, 0)
  - Trạng thái đích: (6, x)
  - Các thao tác: đong đầy bình (từ vòi, hoặc từ bình còn lại), làm rỗng bình
  - Chi phí: 1 cho mỗi thao tác
- Tìm giải pháp: tìm dãy các thao tác chuyển từ trạng thái đầu đến trạng thái đích



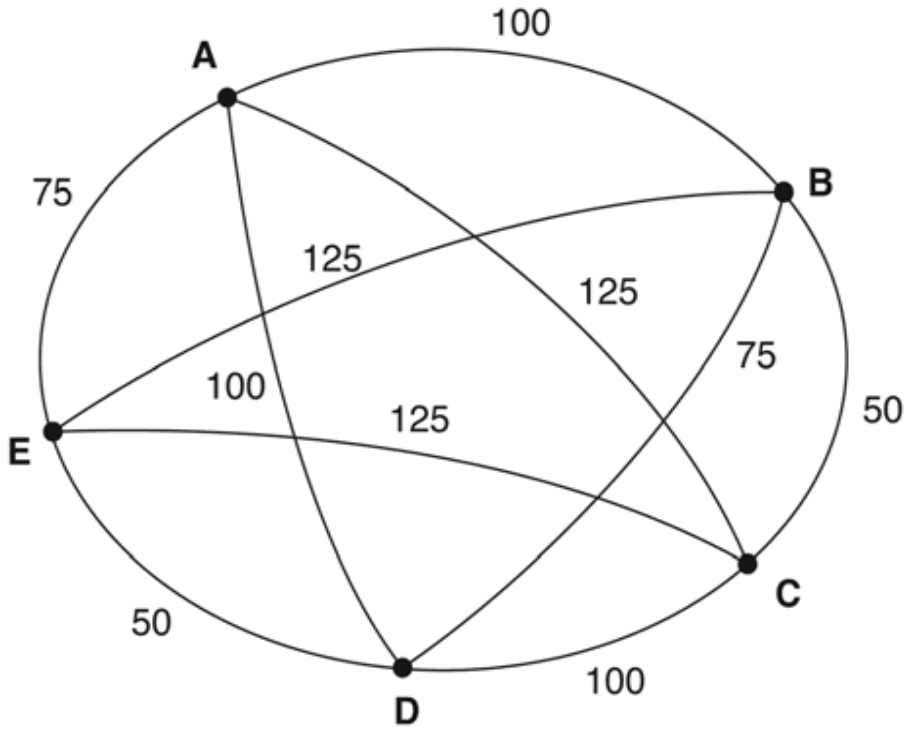
Ví dụ: Bài toán đường đi của người bán hàng (bài toán TSP)

- Xuất phát từ thành phố A, và đi qua tất cả các thành phố, kết thúc quay về A với chi phí thấp nhất.



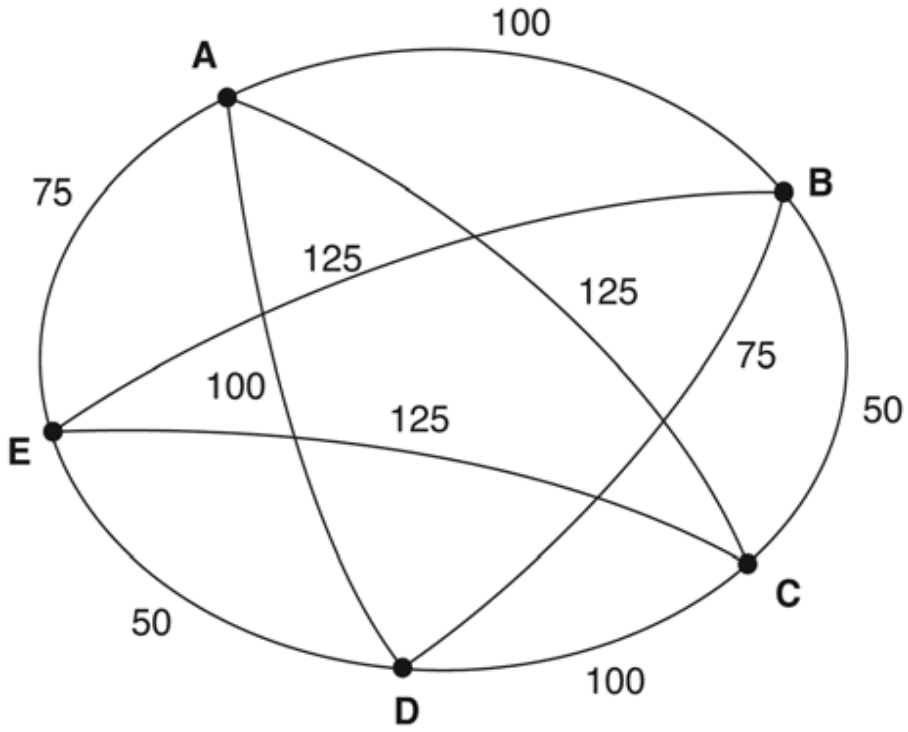
Ví dụ: Bài toán đường đi của người bán hàng (bài toán TSP)

- Xác định các thành phần của bài toán:
  - Trạng thái đầu
  - Trạng thái đích
  - Các hành động
  - Chi phí
- Giải pháp?



Ví dụ: Bài toán đường đi của người bán hàng (bài toán TSP)

- Xác định các thành phần của bài toán:
  - Trạng thái đầu: A
  - Trạng thái đích: A
  - Các hành động: đi từ thành phố này sang thành phố khác
  - Chi phí: khoảng cách giữa các thành phố
- Giải pháp?



Vấn đề đặt ra trong TTNT

- Tìm kiếm một giải pháp thỏa mãn **mục tiêu** (goal)
- Trang bị một số hành động
  - Mỗi hành động làm thay đổi môi trường, trạng thái
- Chuỗi hành động dẫn đến mục tiêu
  - Nhiều chuỗi hành động khả thi
- Việc **tìm kiếm** thực hiện thông qua các chuỗi hành động

Ví dụ

- Cờ vua (Chess)
  - Mỗi lượt đi, tìm kiếm nước đi tốt nhất để chiến thắng
- Tìm đường (Route finding)
  - Tìm kiếm một con đường đi đến đích
- Chứng minh định lý
  - Tìm kiếm chuỗi suy diễn cho một chứng minh
- Máy học
  - Tìm trong không gian khái niệm một khái niệm cho phép phân loại (phân loại thư điện tử có phải là thư rác, phân loại chuỗi DNA, phân loại lừa đảo thẻ tín dụng của tất cả giao dịch hàng ngày...)

Thuật ngữ

- **Trạng thái (States)**: "nơi" việc tìm kiếm có thể ghé qua
- **Không gian trạng thái (Search space)**: tập các trạng thái
- **Đường đi (Search path)**
  - Chuỗi trạng thái mà tác tử đã đi qua
  - Khái niệm về tác tử:
    - An agent is anything that can be viewed as perceiving its environment through sensors and acting upon the environment through effectors (Russell và Norvig)

Thuật ngữ

- **Giải pháp (Solution)**
  - Trạng thái giải quyết được vấn đề đặt ra
    - Có thể được biết trước hoặc là có một tính chất kiểm tra được
  - Có thể nhiều hơn một giải pháp

- **Chiến lược (Strategy)**
  - Chọn trạng thái kế tiếp từ trạng thái hiện hành

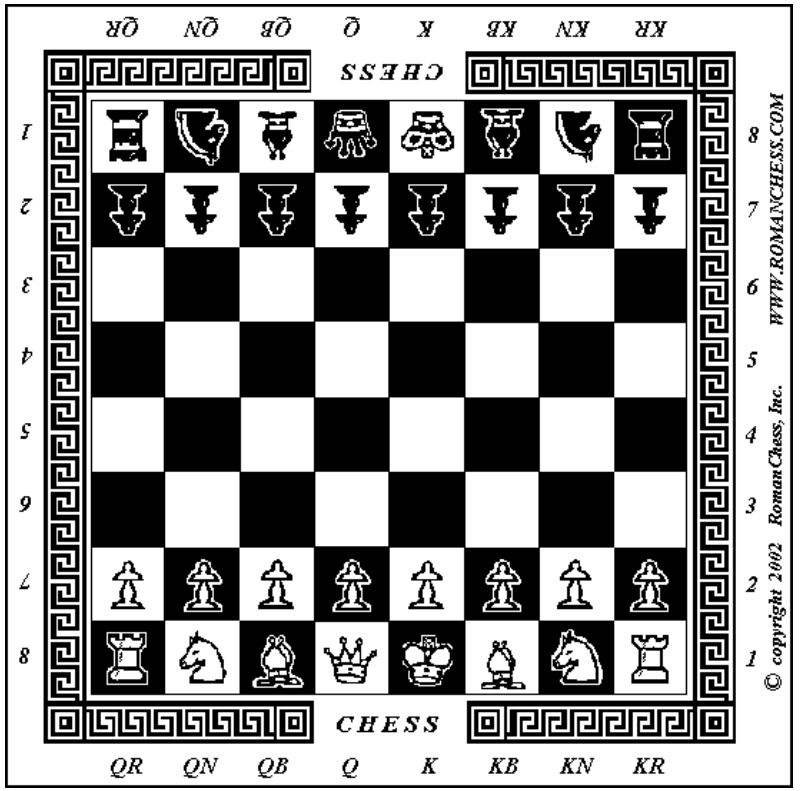
- **Giá thành/Chi phí (Cost)**
  - Chi phí phải trả khi thực hiện giải pháp
  - Đa số các bài toán mong muốn tìm giải pháp có giá thành nhỏ nhất

Mô tả vấn đề

- Vấn đề P được mô tả bằng bộ 5 <St, I, O, G, C>
  1. Tập các trạng thái: St (states)
  2. Trạng thái bắt đầu I (Initial state)
     - Nơi bắt đầu tìm kiếm
  3. Phép toán O (Operators)
     - Hàm sinh ra trạng thái kế tiếp từ trạng thái hiện hành
     - Còn được gọi là hành động
  4. Hàm kiểm tra mục tiêu G (Goal test)
     - Kiểm tra đã tìm thấy được mục tiêu?
  5. Chi phí đường đi: C (cost)
- Chiến lược tìm kiếm sử dụng phép toán để chọn trạng thái kế tiếp

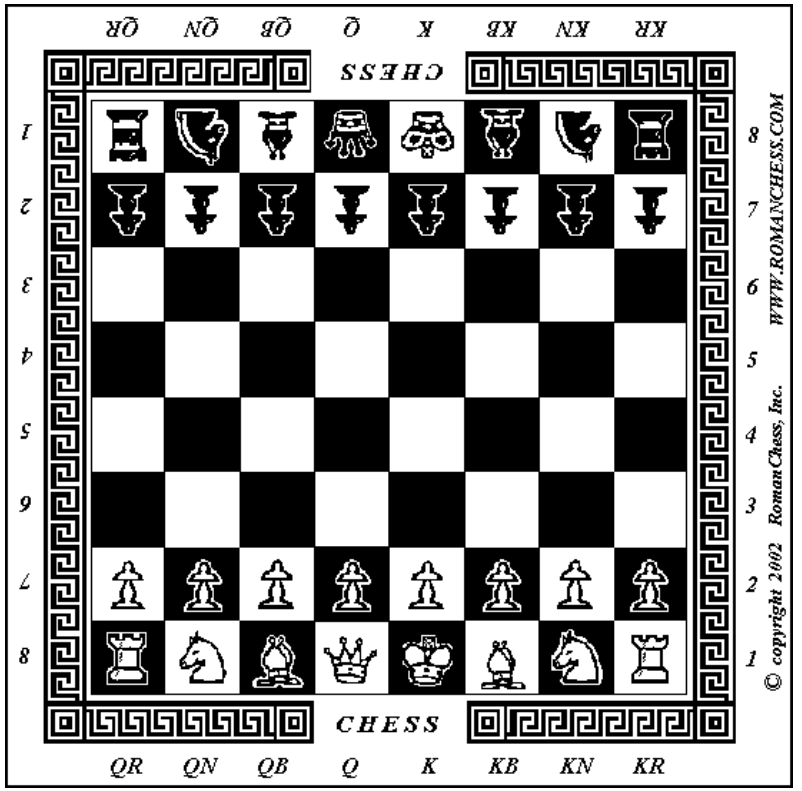
Ví dụ: Cờ vua

- Tập các trạng thái?
- Trạng thái bắt đầu?
- Phép toán?
- Kiểm tra mục tiêu?
- Chi phí?



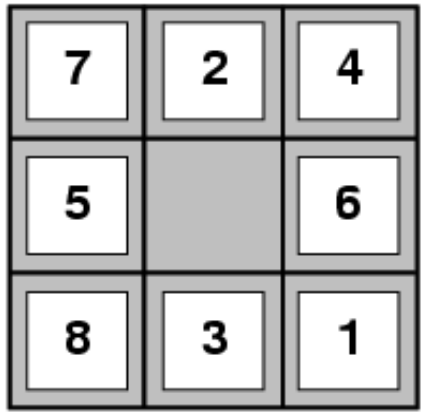
Ví dụ: Cờ vua

- Tập các trạng thái?
- Trạng thái bắt đầu (hình bên phải)
- Phép toán
  - Nước di chuyển hợp lệ của các quân cờ
- Kiểm tra mục tiêu
  - Chiếu hết
    - Vua có thể di chuyển được nữa không?
- Chi phí?

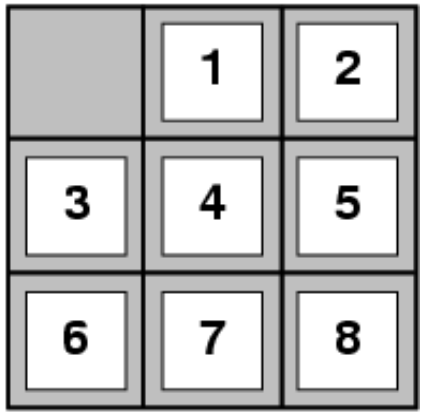


Ví dụ: Trò chơi 8 ô số (8-Puzzle)

- Cần xác định:
  - Các trạng thái?
  - Trạng thái bắt đầu?
  - Các hành động?
  - Mục tiêu?
  - Chi phí đường đi?



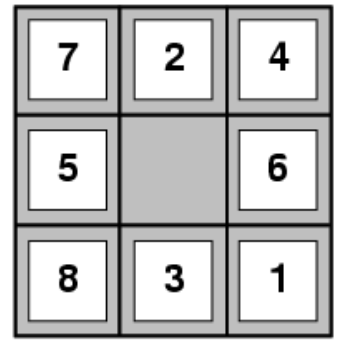
Start State



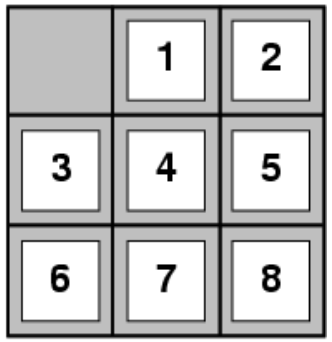
Goal State

Ví dụ: Trò chơi 8 ô số (8-Puzzle)

- Phân tích:
  - Các trạng thái: vị trí khác nhau của các ô số
  - Trạng thái bắt đầu: 1 trạng thái bất kì trong tập các trạng thái
  - Các hành động: di chuyển ô trống sang trái phải, lên, xuống
  - Mục tiêu: Goal state
  - Chi phí đường đi: 1 (mỗi lần di chuyển)



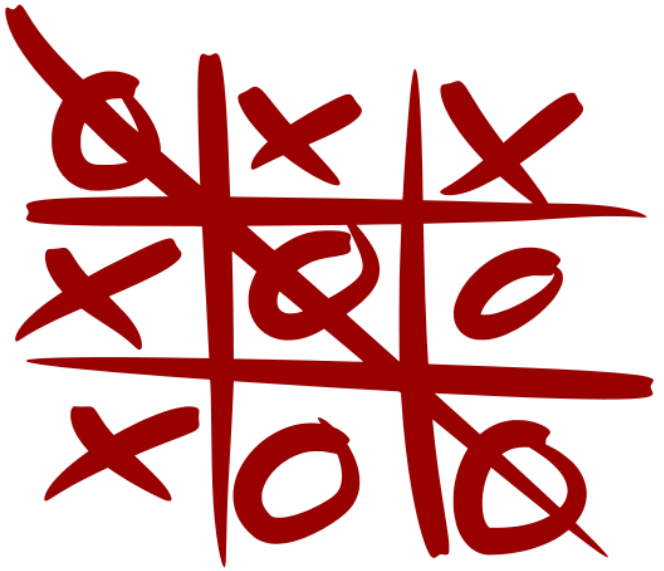
Start State



Goal State

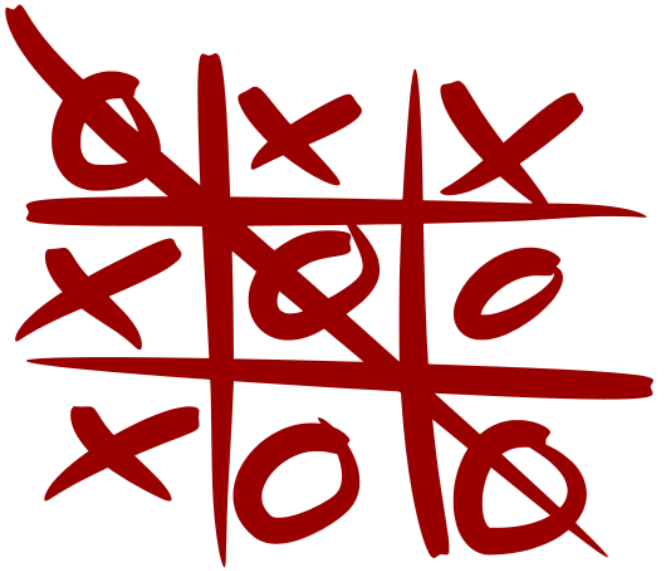
Ví dụ: Trò chơi Tic-tac-toe

- Tic-tac-toe là một trò chơi phổ biến có 9 ô, 3x3.
- Hai người chơi, người thứ nhất dùng ký hiệu O, người kia dùng ký hiệu X, lần lượt điền ký hiệu của mình vào các ô.
- Người thắng là người tạo được đầu tiên một dãy ba ký hiệu của mình, ngang dọc hay chéo đều được (tương tự cờ ca-rô, không giới hạn trong 9 ô, người nào đạt được một dãy 5 thì thắng)



Ví dụ: Trò chơi Tic-tac-toe

- Phân tích:
  - Các trạng thái: mỗi lượt đi sẽ tạo nên 1 trạng thái mới
  - Trạng thái bắt đầu: 9 ô rỗng
  - Các hành động: đi X hoặc O
  - Mục tiêu: dãy 3 ký hiệu giống nhau (ngang, dọc hoặc chéo)
  - Chi phí: 1 (mỗi lượt đi)



Không gian trạng thái (KGTT)

- Một trong những cách để giải quyết các bài toán TTNT là sử dụng Không gian trạng thái (KGTT) để biểu diễn tri thức
- Biểu diễn KGTT là gì?
  - KGTT bao gồm các trạng thái, kết nối với nhau bằng các hành động
  - Từ một trạng thái ban đầu, một hành động có thể chuyển trạng thái đó sang một trạng thái khác

Giải pháp cho các bài toán

- Biểu diễn bài toán trên KGTT để thực hiện tìm kiếm một đường đi lời giải/ một trạng thái đích mong muốn
- Các bước xây dựng KGTT:
  - Bắt đầu từ một trạng thái đã cho (trạng thái ban đầu)
  - Kiểm tra xem có phải là trạng thái đích
  - Mở rộng 1 trong các trạng thái
  - Nếu có nhiều khả năng, chọn 1 khả năng nào đó
  - Quy trình: chọn, kiểm tra và mở rộng thực hiện đến khi tìm ra giải pháp hoặc không thể tiếp tục mở rộng

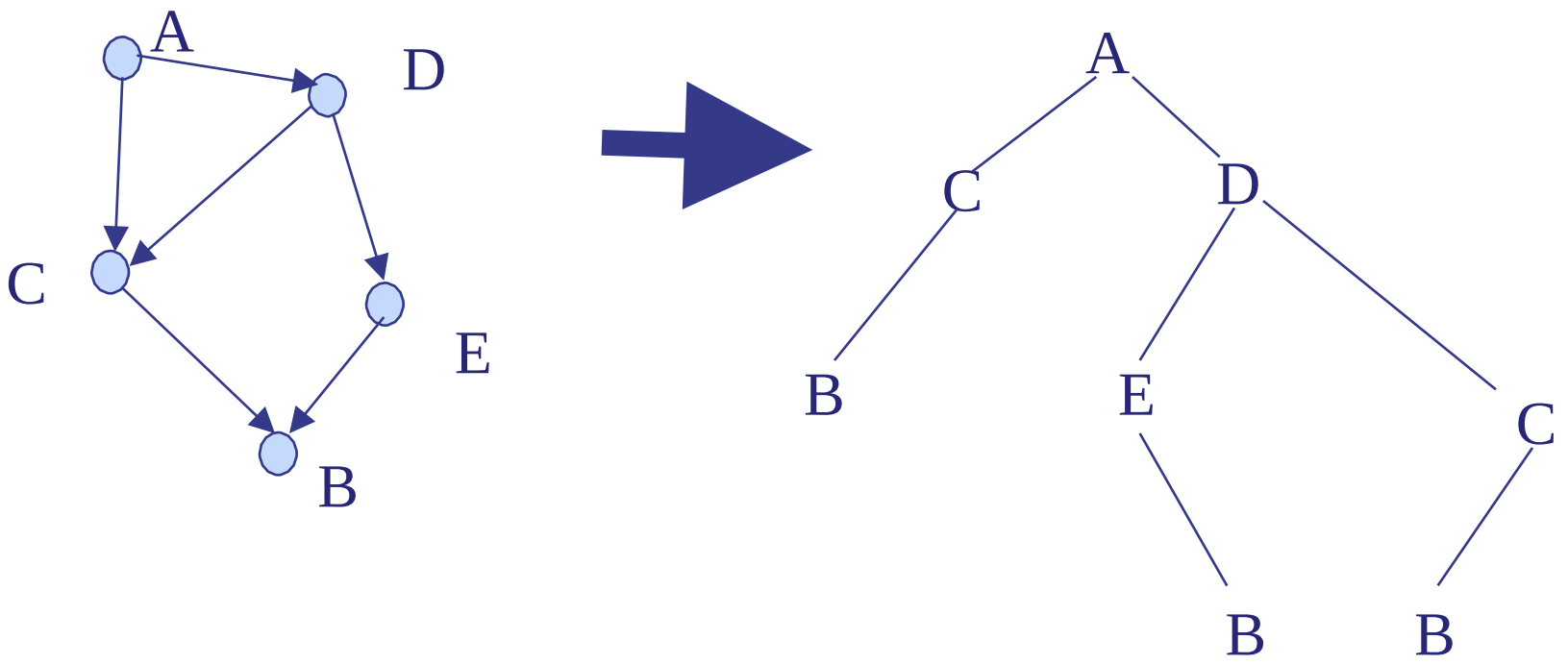
  => quy trình xây dựng cây tìm kiếm
- Tập các trạng thái được mở rộng sẽ có nhiều lựa chọn, việc chọn trạng thái nào để mở rộng được gọi là **chiến lược tìm kiếm**

Giải pháp cho các bài toán

- Cây tìm kiếm:
  - Mô tả KGTT
  - Nút gốc: nút tìm kiếm tương ứng với trạng thái ban đầu
  - Nút lá: tương ứng với các trạng thái không thể mở rộng (không phát sinh con)
- Lưu ý:
  - KGTT không hoàn toàn giống cây tìm kiếm
  - KGTT phải thể hiện các khả năng diễn biến của bài toán.
  - Cần xác định một nút sẽ biểu diễn những thông tin gì?
- Quá trình tìm kiếm tức là tìm đường đi trên cây tìm kiếm (chuỗi các trạng thái) dẫn đến trạng thái mong muốn

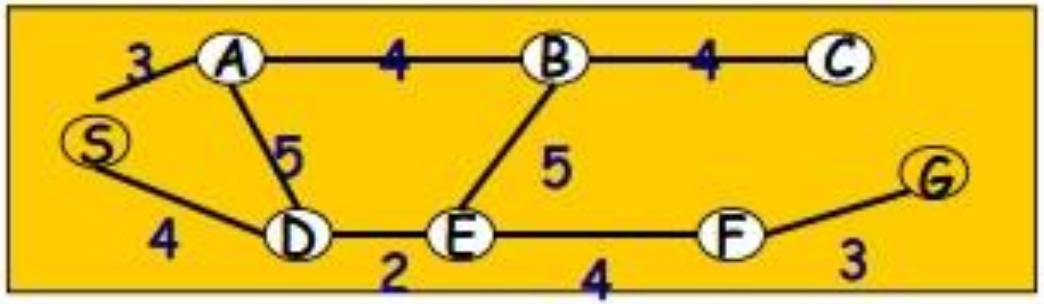
Giải pháp cho các bài toán

- Ví dụ cây tìm kiếm:

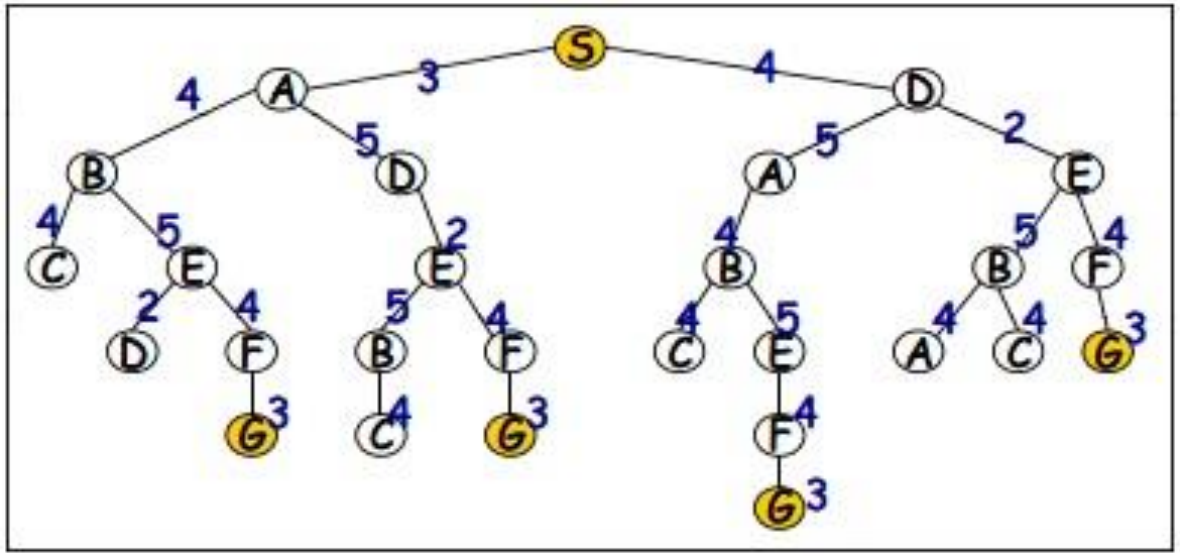


Giải pháp cho các bài toán

Ví dụ: Đồ thị không gian trạng thái

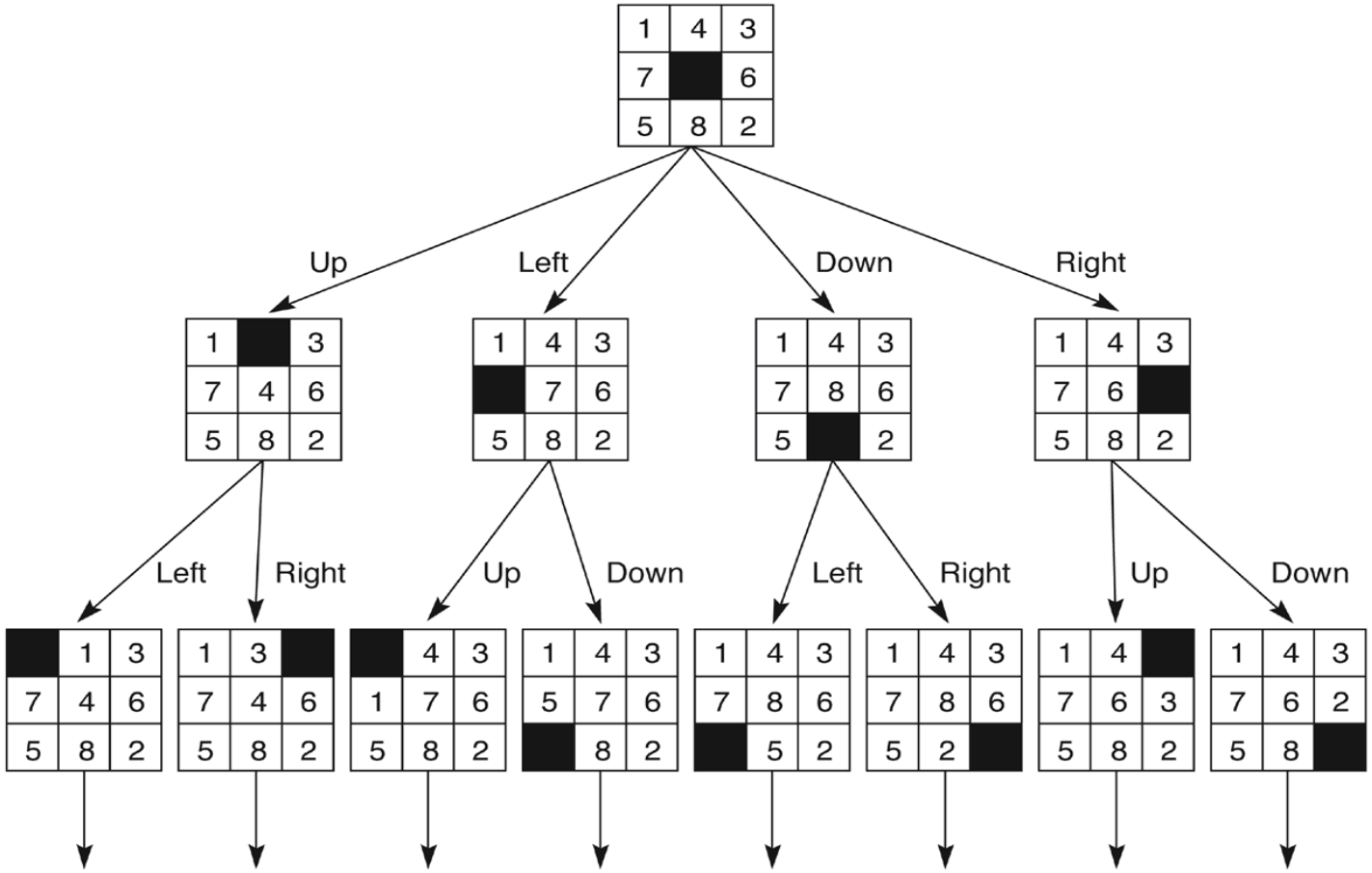


Ví dụ: Cây tìm kiếm



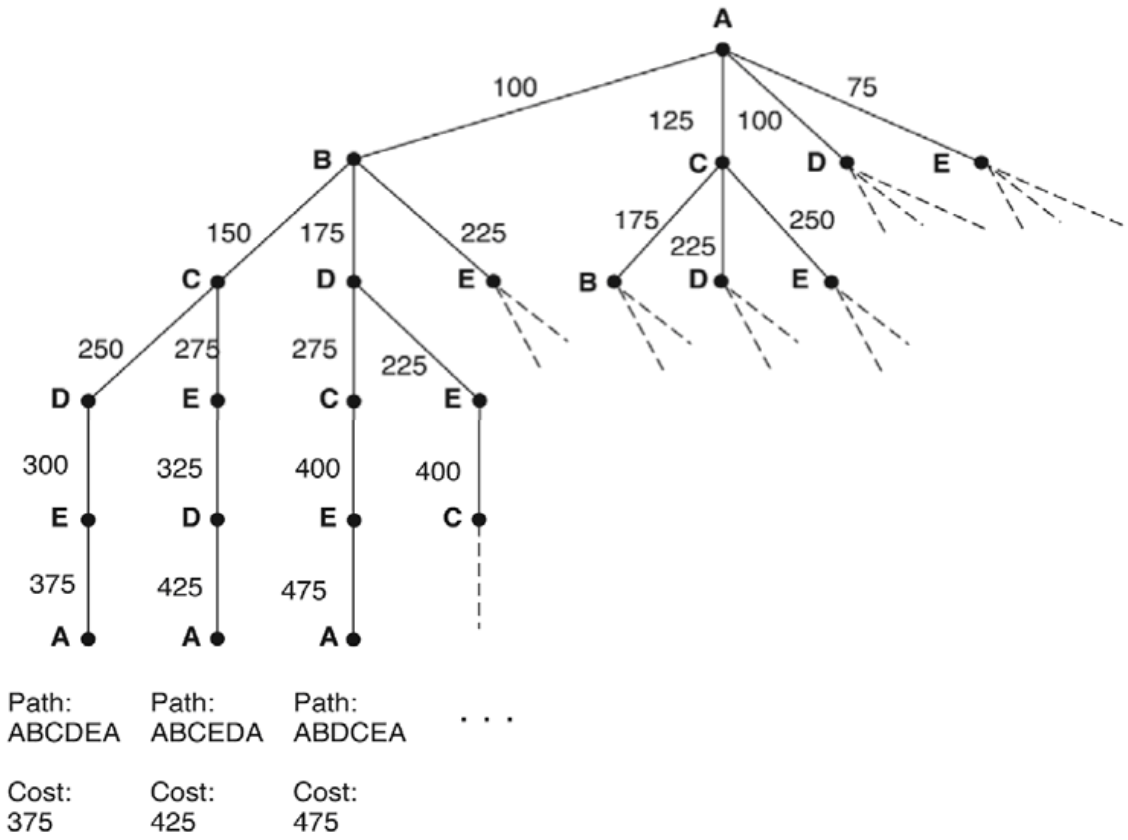
Ví dụ biểu diễn bài toán trên KGTT

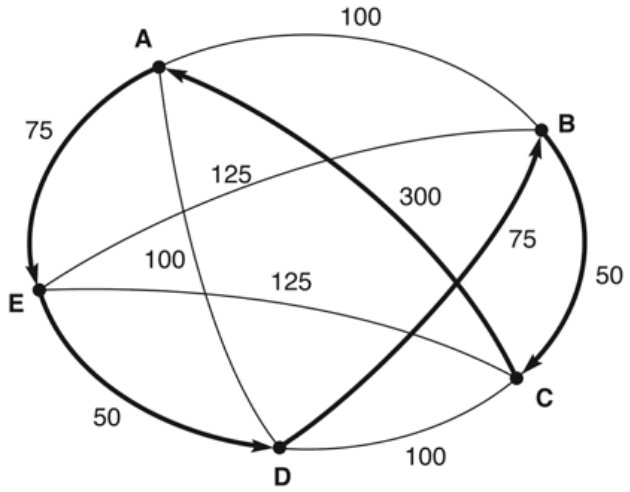
- Trò chơi 8-Puzzle



Ví dụ biểu diễn bài toán trên KGTT

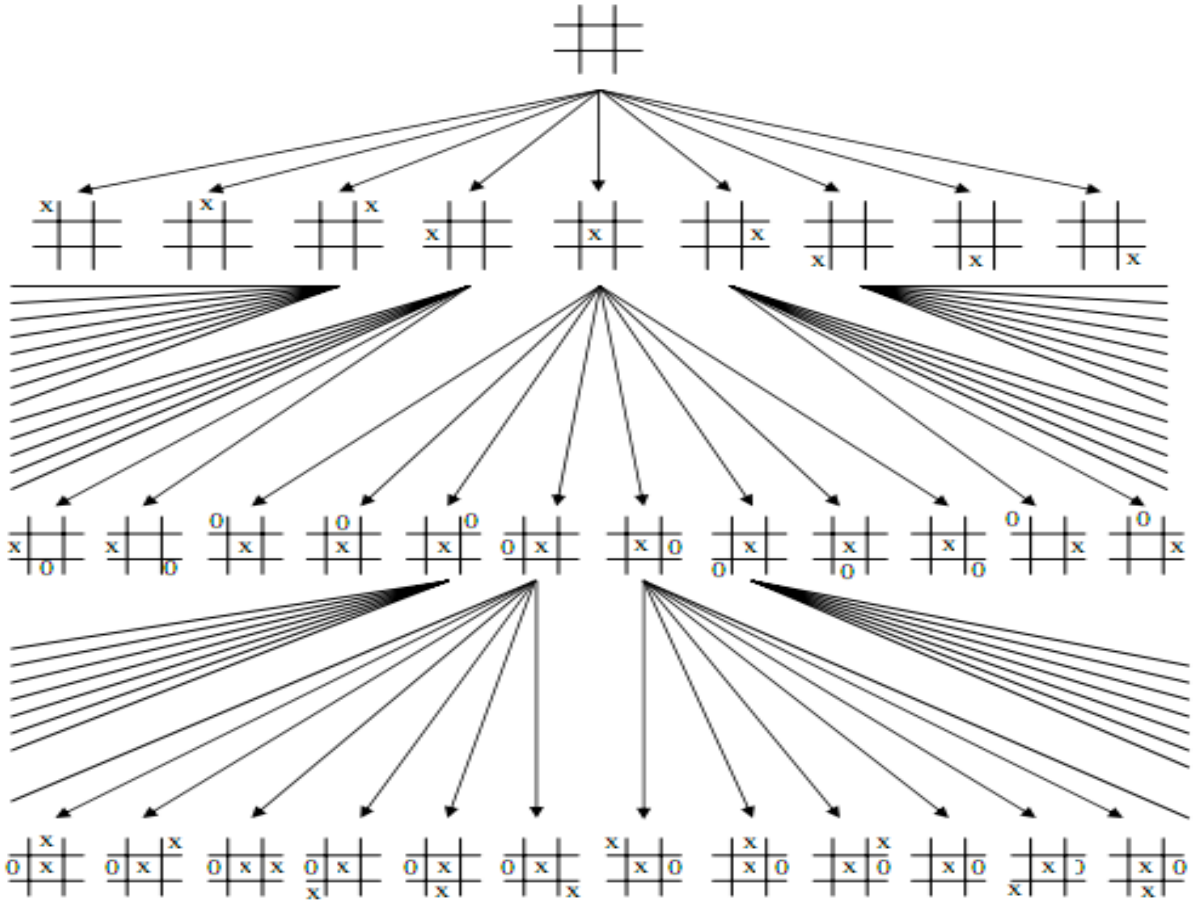
- Bài toán TSP





Ví dụ biểu diễn bài toán trên KGTT

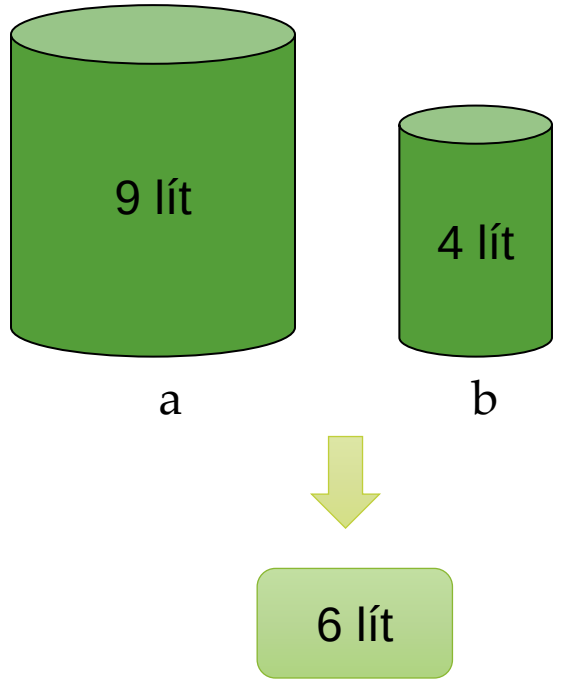
- Trò chơi Tic-tac-toe:



Ví dụ biểu diễn bài toán trên KGTT

- Bài toán đong nước?

(Cho 2 bình 9 lít và 4 lít, không có vạch chia, và 1 vòi bơm. Làm cách nào đong được 6 lít?)

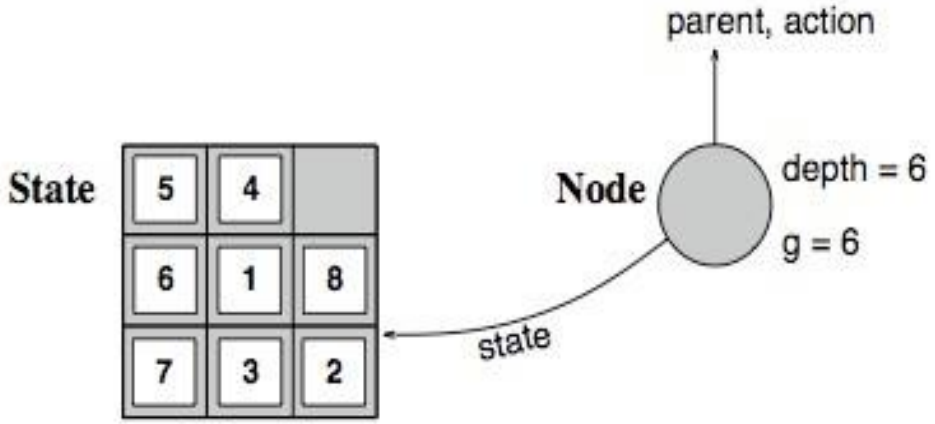


Giải pháp cho các bài toán

- Cấu trúc dữ liệu cho 1 nút gồm 5 thành phần có thể:
  - Trạng thái tương ứng
  - Nút cha: là nút tạo ra nút đang được xét
  - Các thao tác/hành động: được áp dụng để tạo ra nút hiện tại
  - Độ sâu: số nút tính từ nút gốc đến nút hiện tại – 1
  - Chi phí đường đi

Giải pháp cho các bài toán

- Trạng thái vs. Nút
  - 1 trạng thái tượng trưng cho 1 tình trạng thực tế của bài toán
  - 1 nút là một cấu trúc dữ liệu tạo thành cây tìm kiếm, bao gồm: các nút cha, con, độ sâu, chi phí đường đi
  - Trạng thái không có khái niệm: cha, con, độ sâu, hoặc chi phí đường đi
  - Lưu ý: 2 nút khác nhau có thể mô tả cho cùng 1 trạng thái



Đánh giá phương pháp tìm kiếm

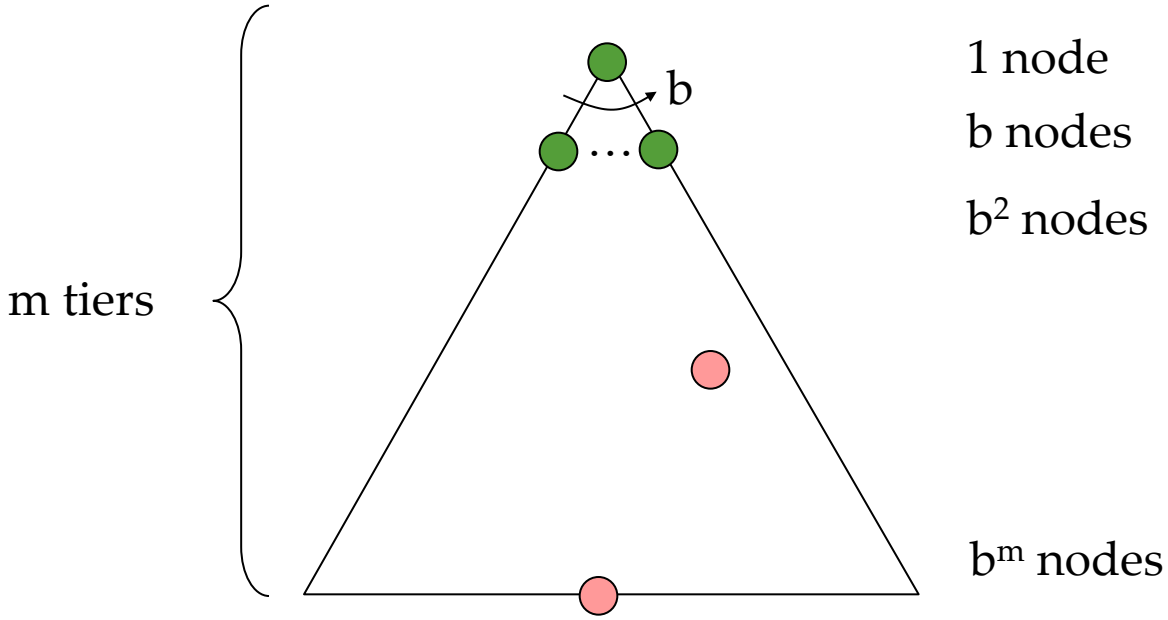
- Phương pháp tìm kiếm xác định thứ tự triển khai của các đỉnh trong cây tìm kiếm
- Các giải thuật tìm kiếm thường được đánh giá dựa trên 4 tiêu chí sau:
  - Tính trọn vẹn: Liệu nó luôn tìm ra nghiệm không nếu bài toán tồn tại nghiệm.
  - Độ phức tạp thời gian: giải thuật có mất nhiều thời gian không?
  - Độ phức tạp không gian: yêu cầu bộ nhớ lưu trữ?
  - Tính tối ưu: Giải thuật có đảm bảo tìm ra nghiệm với hàm chi phí ít nhất không?

Đánh giá phương pháp tìm kiếm (tt)

- Độ phức tạp thời gian và độ phức tạp không gian của giải thuật tìm kiếm lời giải có thể đánh giá dựa trên kích thước đầu vào của giải thuật. Các tham số kích thước đầu vào có thể là:

  + b: số nhánh tối đa của một nút, hay là số phép chuyển trạng thái tối đa của một trạng thái tổng quát

  + m – độ sâu tối đa của cây tìm kiếm (m có thể là vô hạn)



Các kỹ thuật tìm kiếm

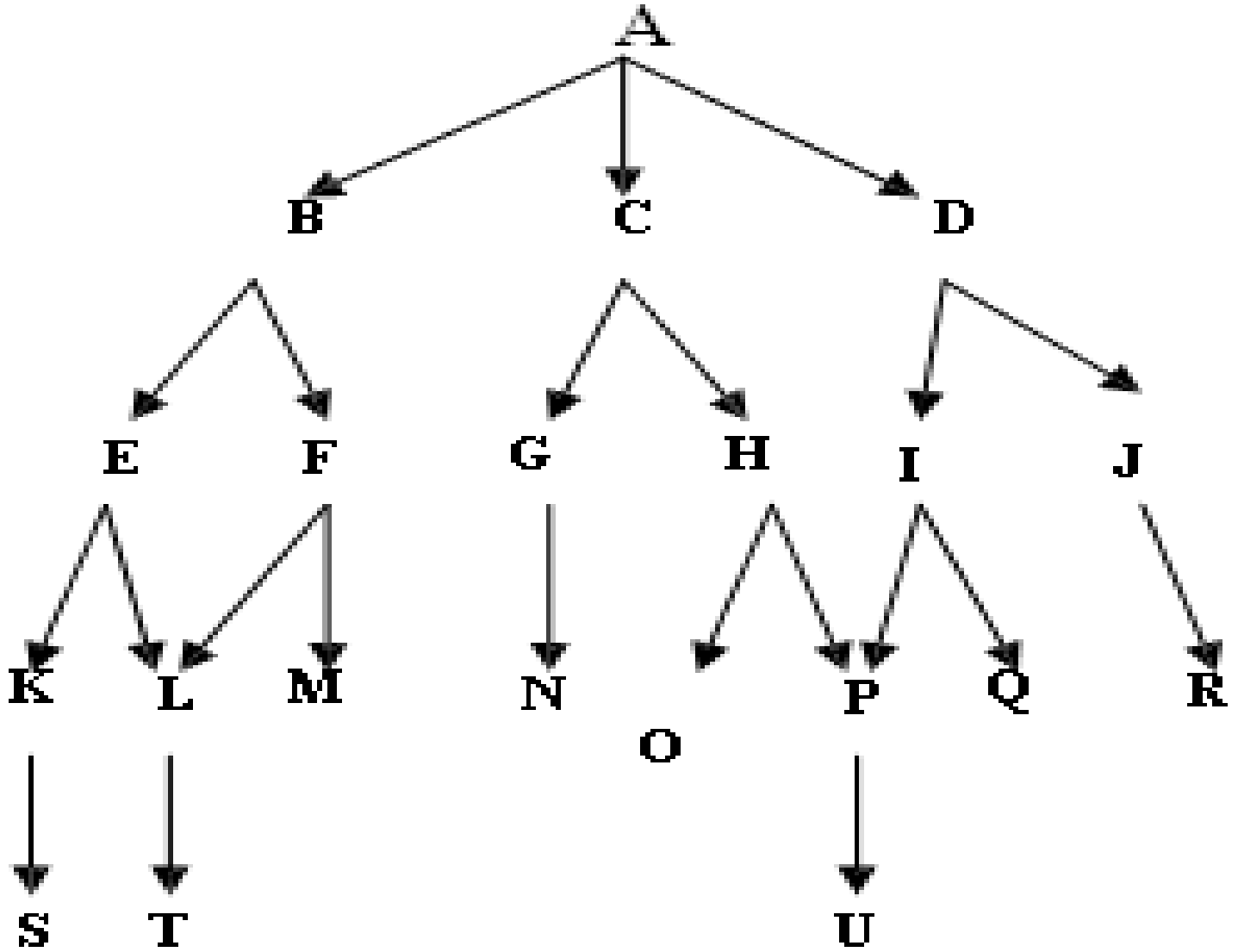
- Tìm kiếm mù (uninformed/blind search):
  - Không hiểu biết gì về đối tượng
  - Duyệt qua để tìm được đối tượng cần tìm
- Tìm kiếm dựa trên kinh nghiệm (informed/ heuristic search):
  - Dựa vào kinh nghiệm và sự hiểu biết để xây dựng hàm đánh giá hướng dẫn tìm kiếm

Tìm kiếm mù

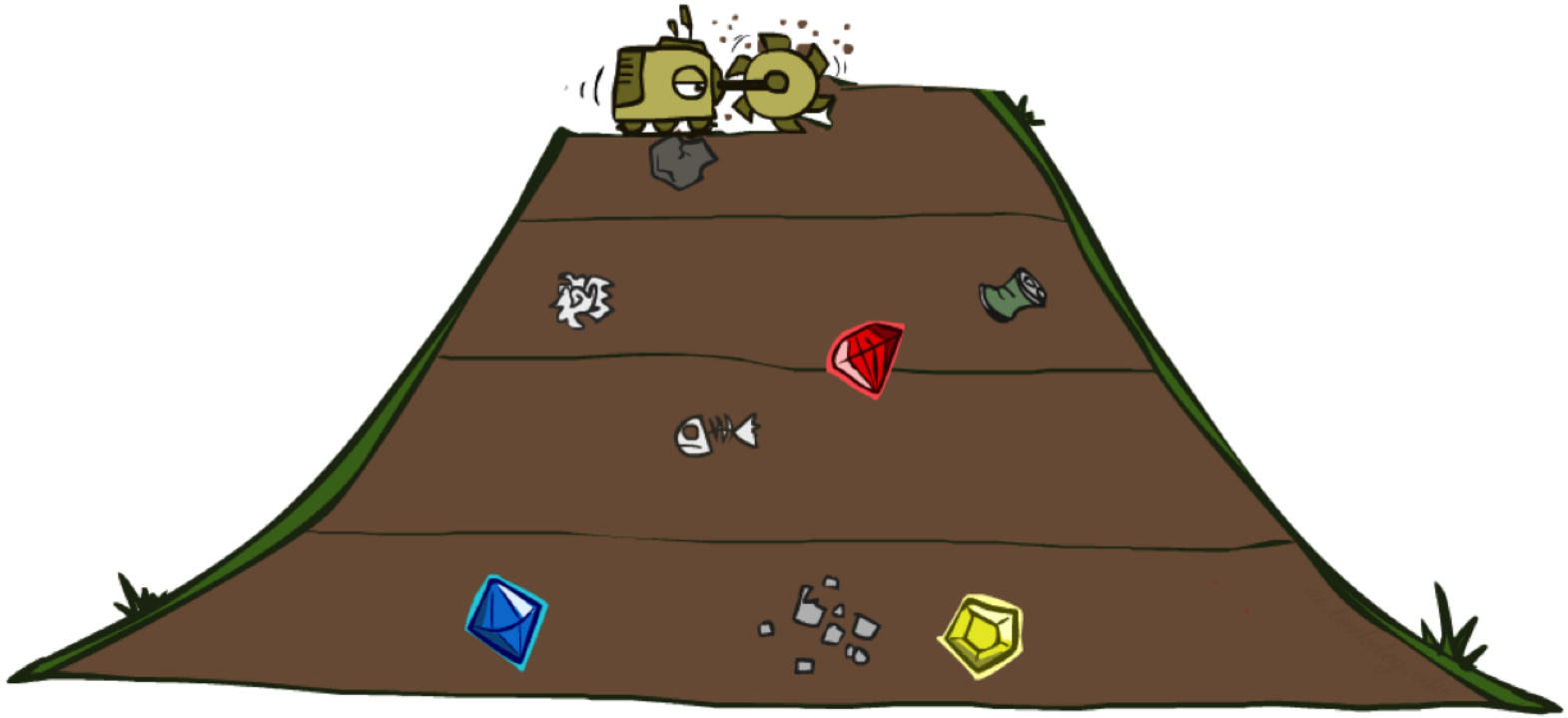
- Tìm kiếm rộng (breath-first search)
- Tìm kiếm sâu (depth-first search)
- Tìm kiếm theo độ sâu có giới hạn (depth-limited search)
- Tìm kiếm theo chiều sâu lặp (iterative deepening depth-first search)
- Tìm kiếm hai chiều (Bidirectional search)
- Tìm kiếm với giá đồng nhất (Uniform-cost search)

Tìm kiếm mù

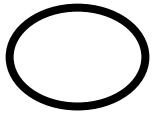
- Xét KGTT sau:



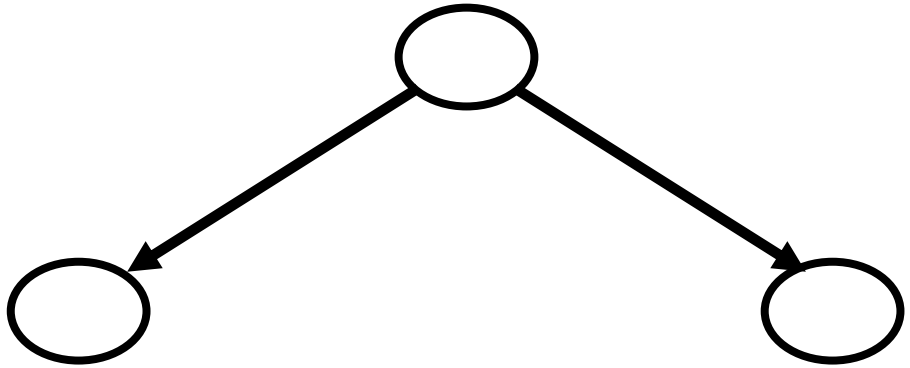
Tìm kiếm rộng (breath-first search - BFS)



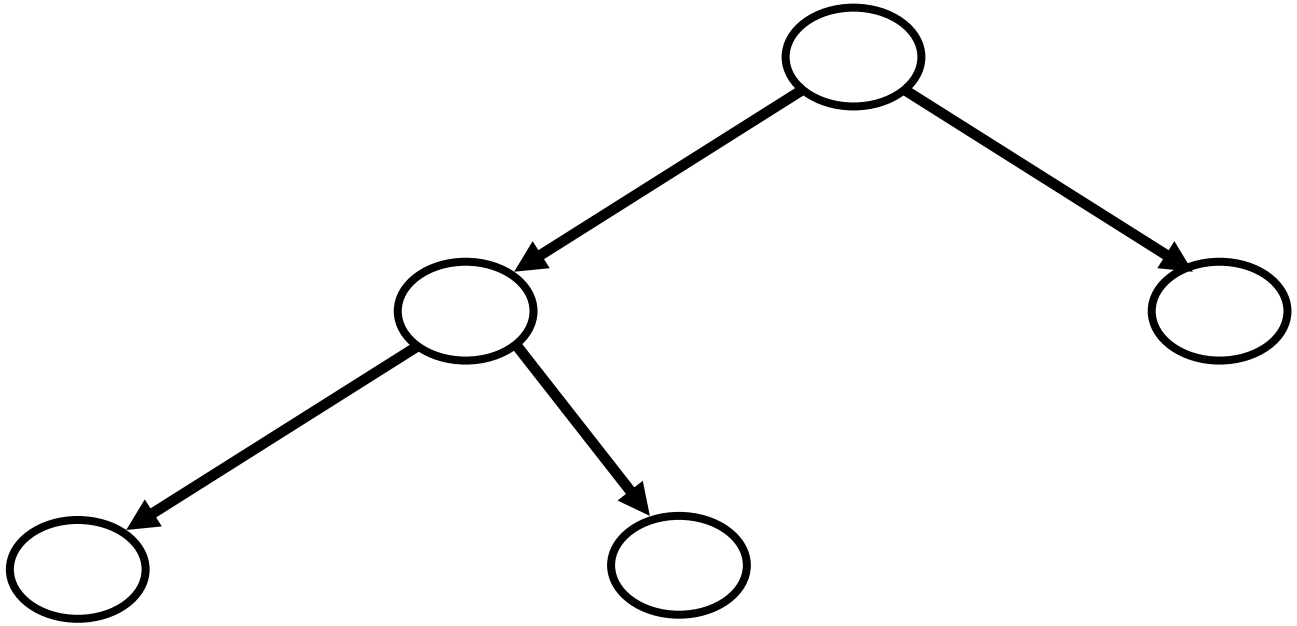
Tìm kiếm rộng (breath-first search - BFS)



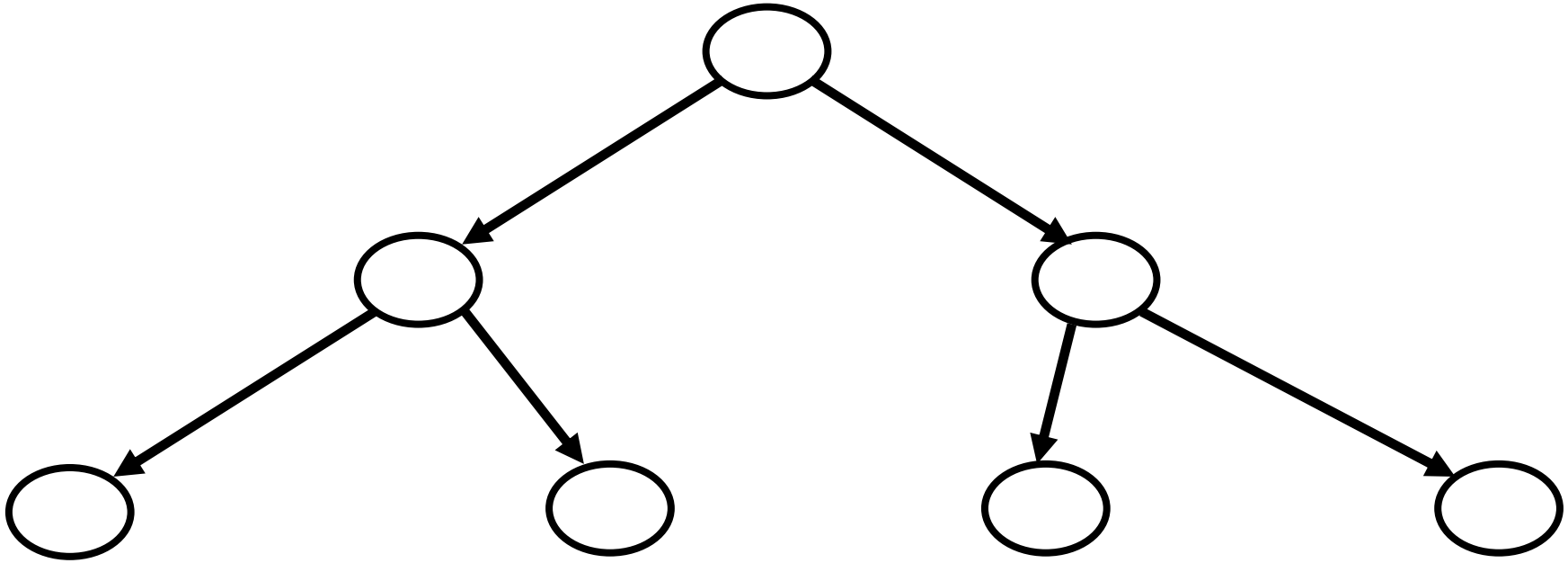
Tìm kiếm rộng (breath-first search - BFS)



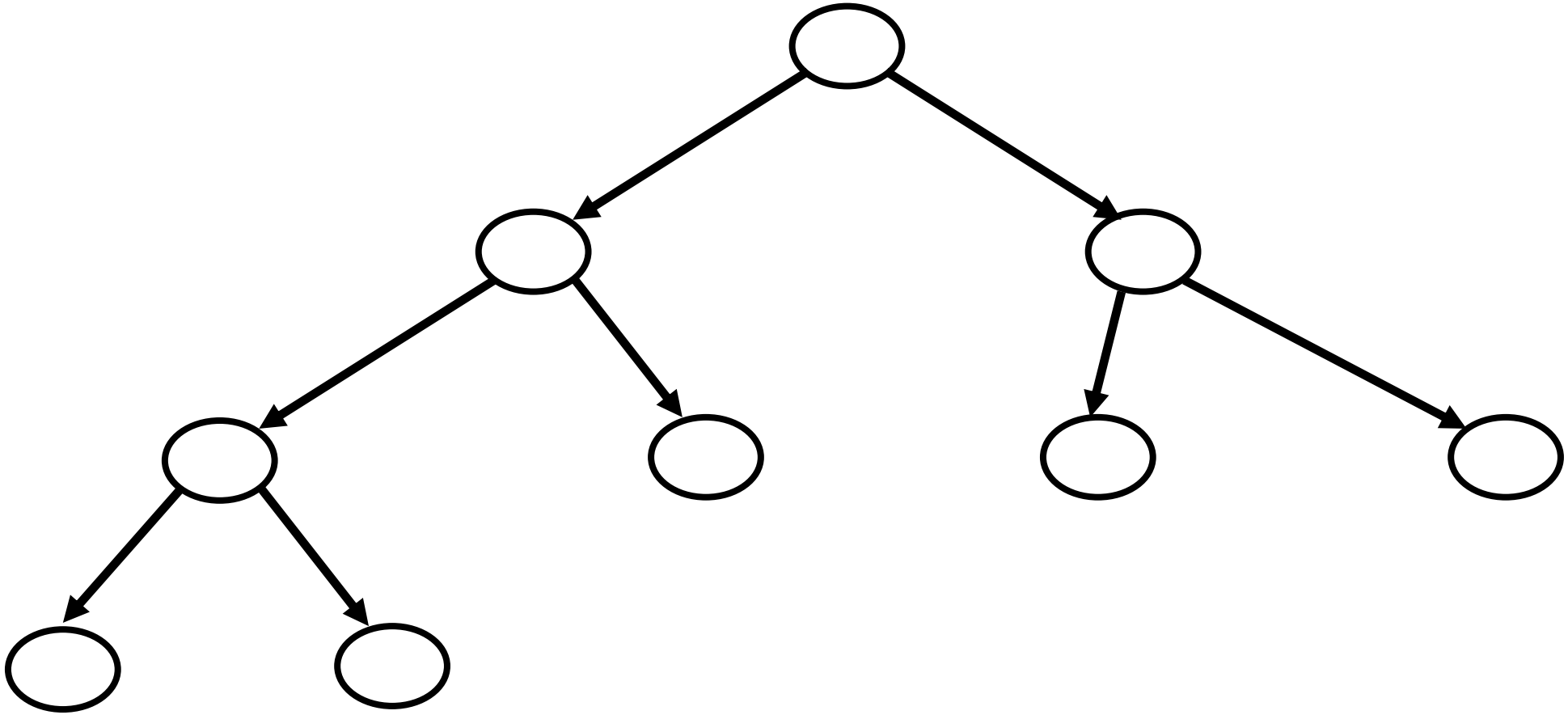
Tìm kiếm rộng (breath-first search - BFS)



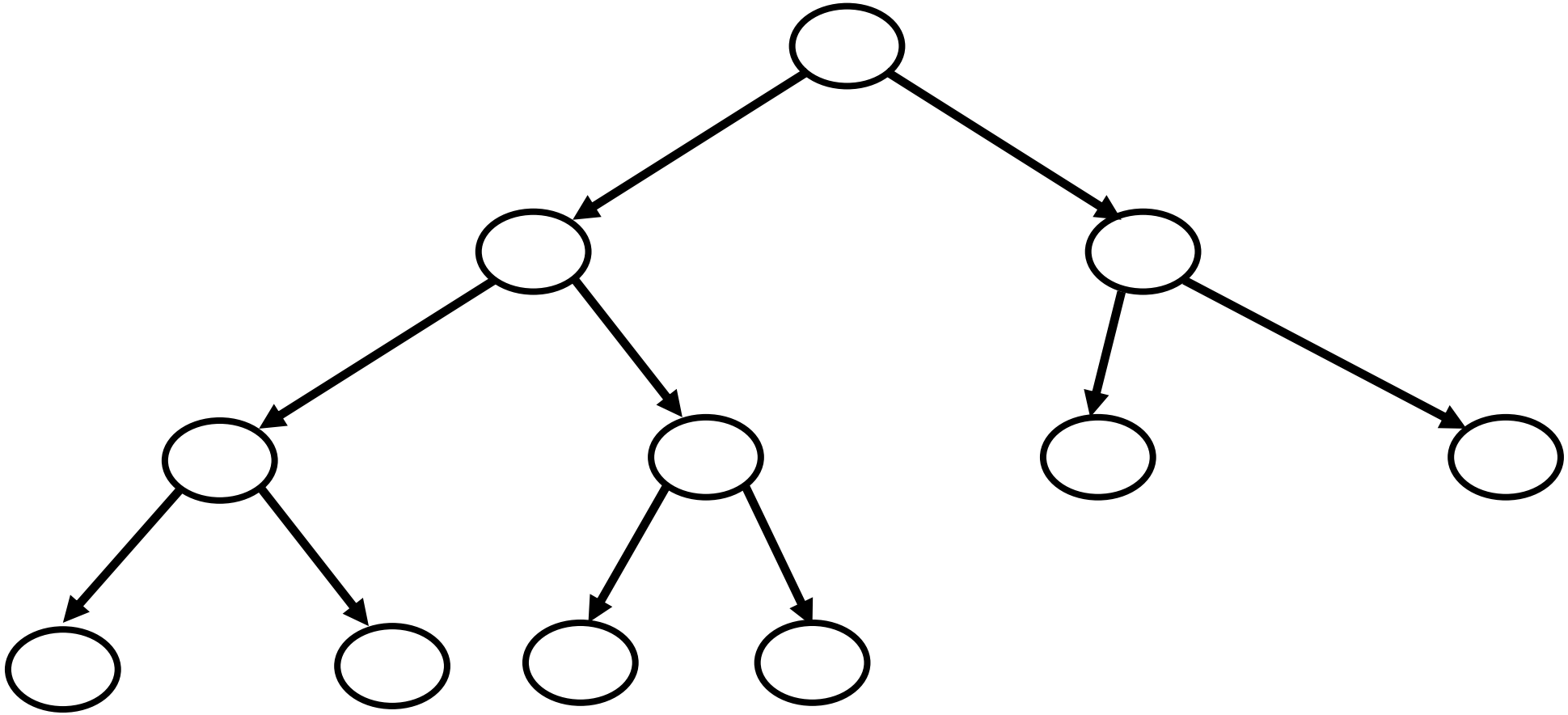
Tìm kiếm rộng (breath-first search - BFS)



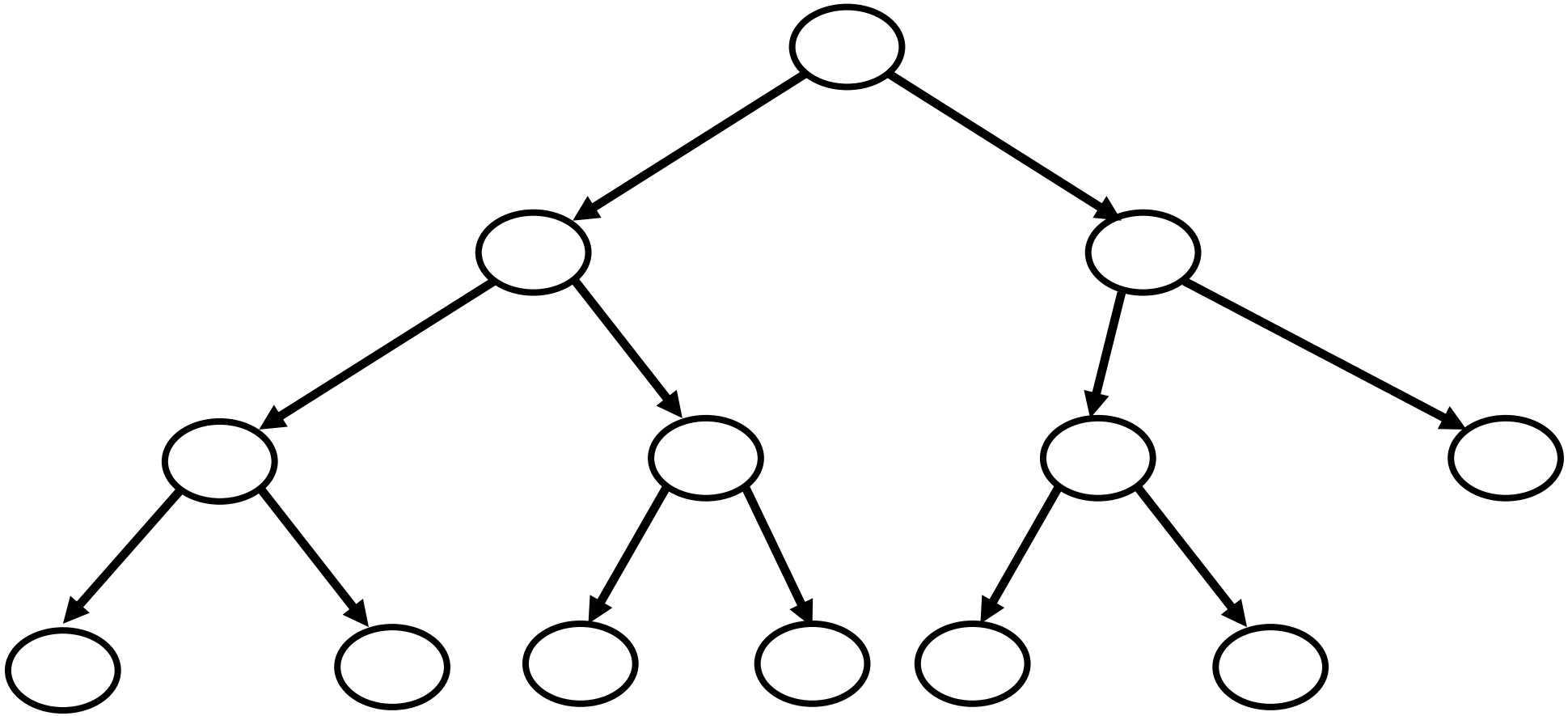
Tìm kiếm rộng (breath-first search - BFS)



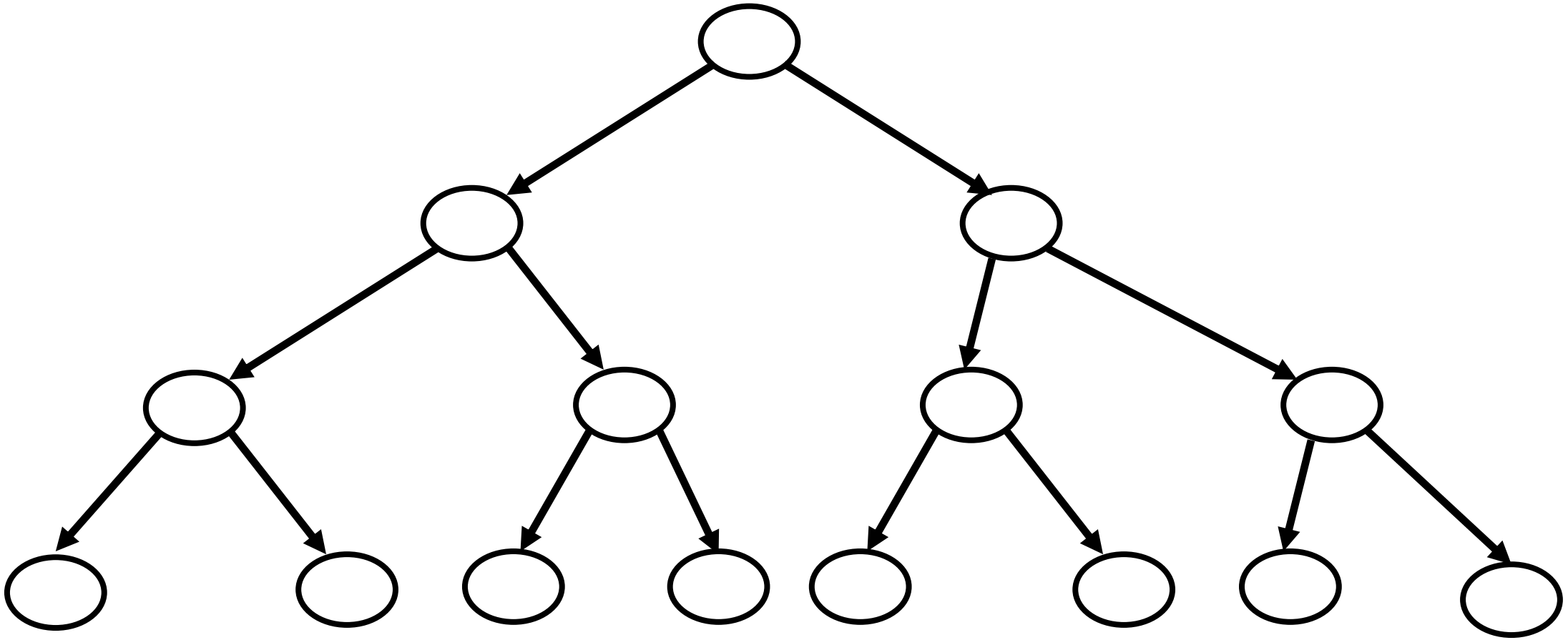
Tìm kiếm rộng (breath-first search - BFS)



Tìm kiếm rộng (breath-first search - BFS)



Tìm kiếm rộng (breath-first search - BFS)

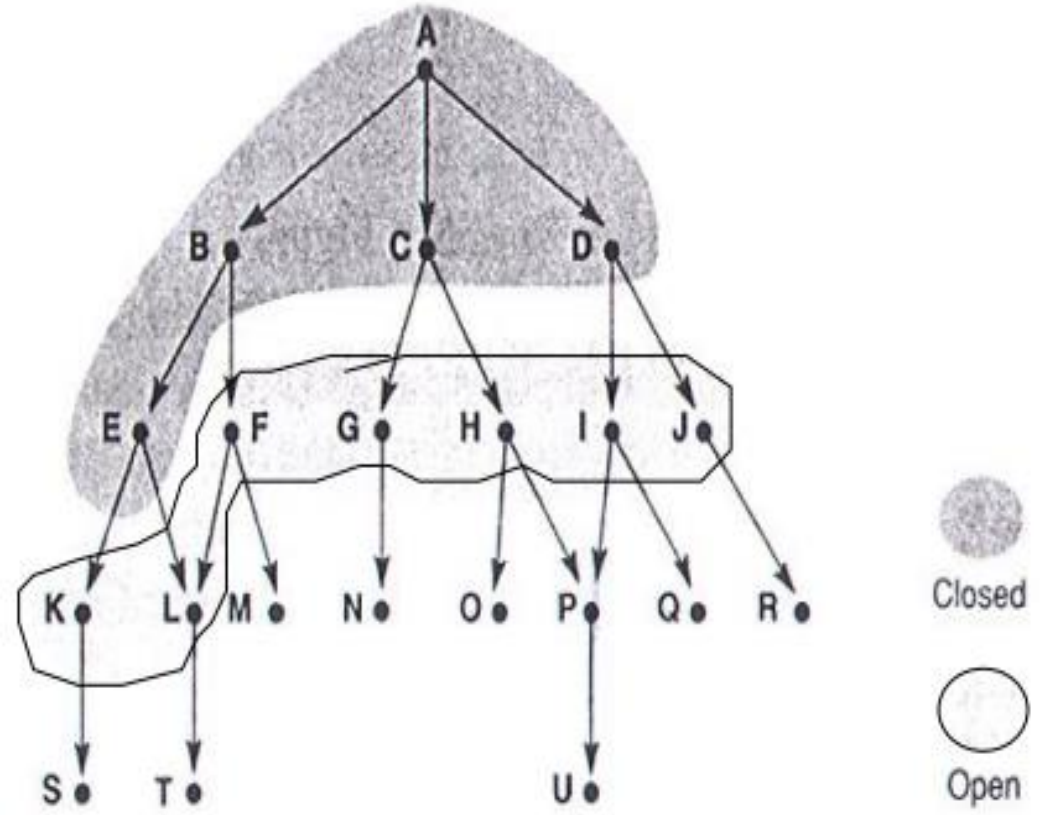


Tìm kiếm rộng (breath-first search - BFS)

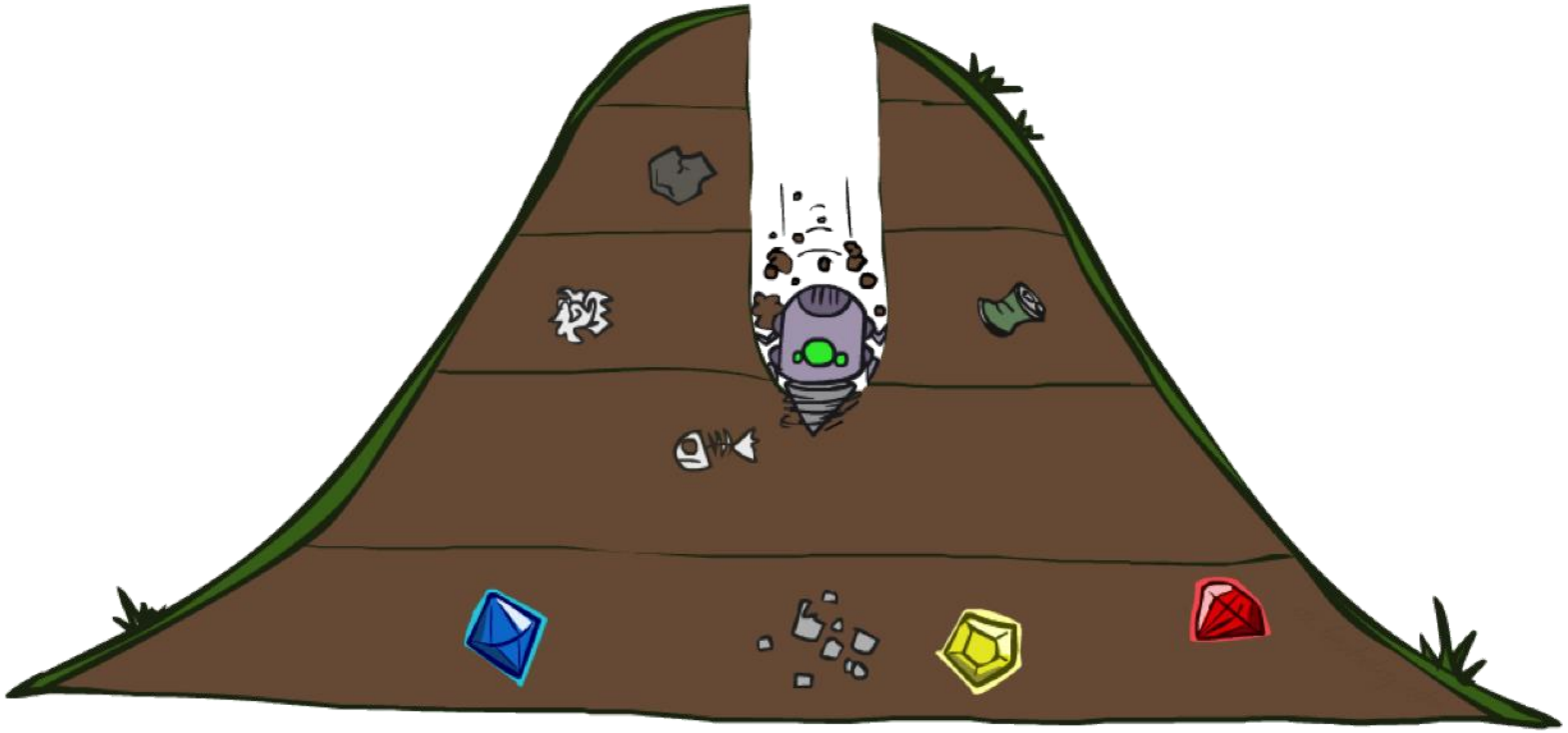
```
Procedure breadth-first-search;
   Begin                    % khởi đầu
      Queue Open:= [start], Closed:= [ ];
      While open ≠ [ ] do    % còn các trạng thái chưa khảo sát
         Begin
            Lấy một trạng thái khỏi open, gọi nó là X;
            If X là mục tiêu then trả lời kết quả (thành công) // tìm thấy đích
            else begin
                 Thực hiện các phép toán (hành động) trên X và sinh ra con của X
                 Đưa X vào closed;
                 Loại các con của X đã tồn tại open hoặc closed;
                 Đưa các con còn lại vào open; //Queue – Hàng đợi
            End;
         End;
       Trả lời kết quả (thất bại);   % không còn trạng thái nào
  End;
```

Tìm kiếm rộng (breath-first search - BFS)

- Tìm kiếm rộng:
- Các bước thực hiện tìm kiếm rộng:
  1. Open = [A]; closed = []
  2. Open = [B,C,D]; closed = [A]
  3. Open = [C,D,E,F];closed = [B,A]
  4. Open = [D,E,F,G,H];
     closed = [C,B,A]
  5. Open = [E,F,G,H,I,J];
  closed = [D,C,B,A]
  6. Open = [F,G,H,I,J,K,L];
  closed = [E,D,C,B,A]
  7. Open = [G,H,I,J,K,L,M];
        (vì L đã có trong open);
        closed = [F,E,D,C,B,A]
  8. …



Tìm kiếm sâu (depth-first search - DFS)

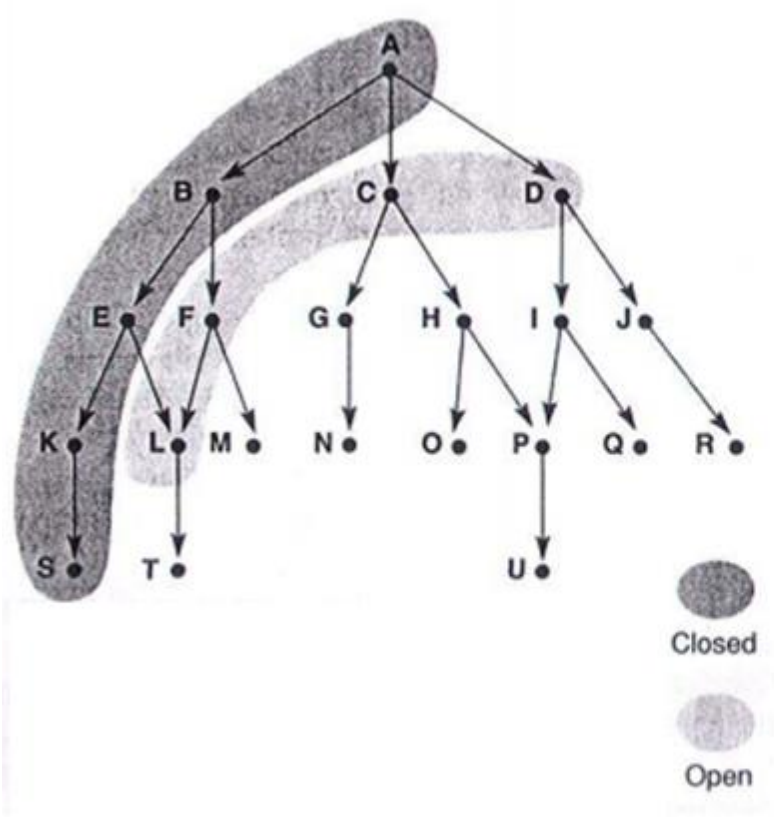


Tìm kiếm sâu (depth-first search)

```
Procedure depth – first –search;
   Begin                    % khởi đầu
      Stack Open:= [start], Closed:= [ ];
      While open ≠ [ ] do    % còn các trạng thái chưa khảo sát
         Begin
            Lấy một trạng thái khỏi open, gọi nó là X;
            If X là mục tiêu then trả lời kết quả (thành công) // tìm thấy đích
            else begin
                Thực hiện các phép toán (hành động) trên X và sinh ra con của X
                Đưa X vào closed;
                Loại các con của X đã tồn tại open hoặc closed;
                Đưa các con còn lại vào open; //Stack – Ngăn xếp
            End;
          End;
        Trả lời kết quả (thất bại);    % không còn trạng thái nào
  End;
```

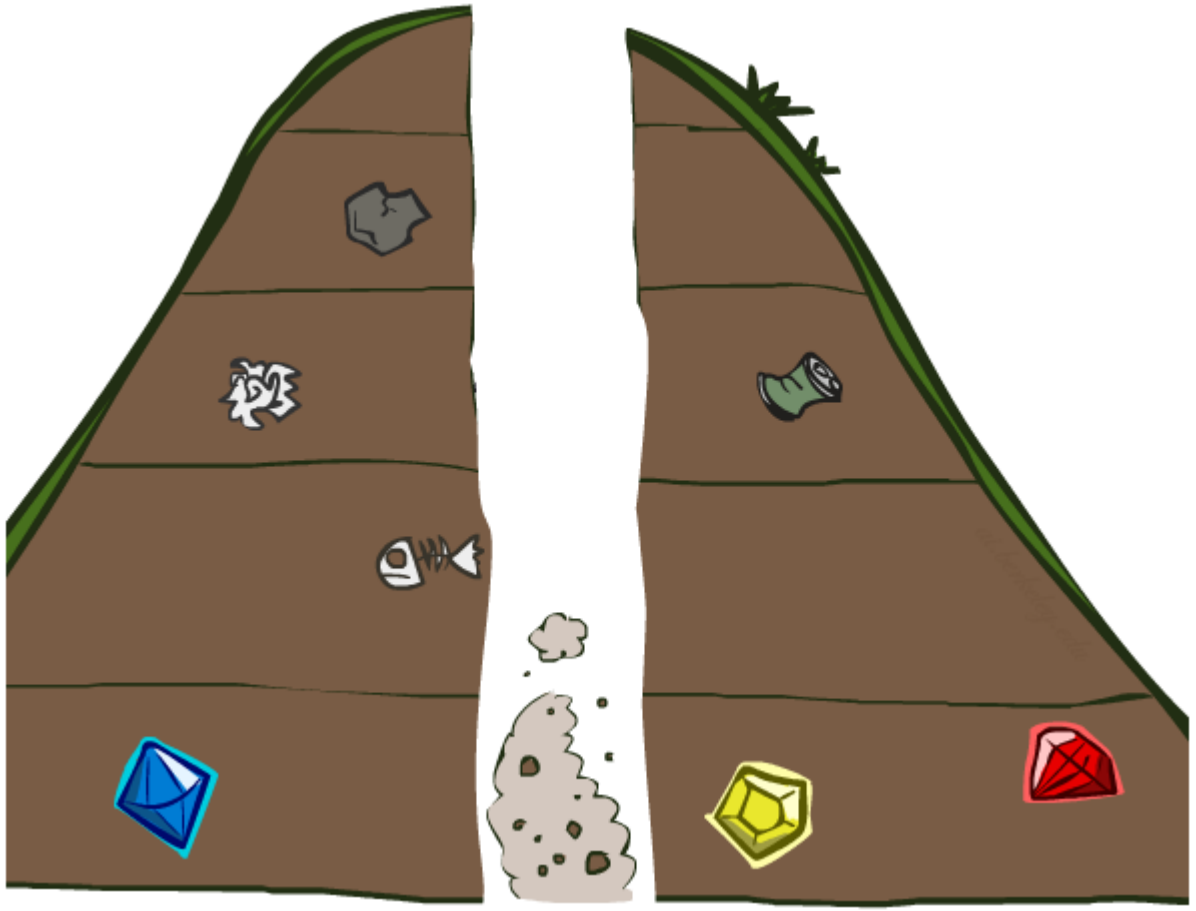
Tìm kiếm sâu (depth-first search - DFS)

- Tìm kiếm sâu:
- Các bước thực hiện tìm kiếm sâu:
    1. Open = [A]; closed = []
    2. Open = [B,C,D]; closed = [A]
    3. Open = [E,F,C,D];closed = [B,A]
    4. Open = [K,L,F,C,D];
       closed = [E,B,A]
    5. Open = [S,L,F,C,D];
       closed = [K,E,B,A]
    6. Open = [L,F,C,D];
       closed = [S,K,E,B,A]
    7. Open = [T,F,C,D];
       closed = [L,S,K,E,B,A]
    8. Open = [F,C,D];
       closed = [T,L,S,K,E,B,A]
    9. ….

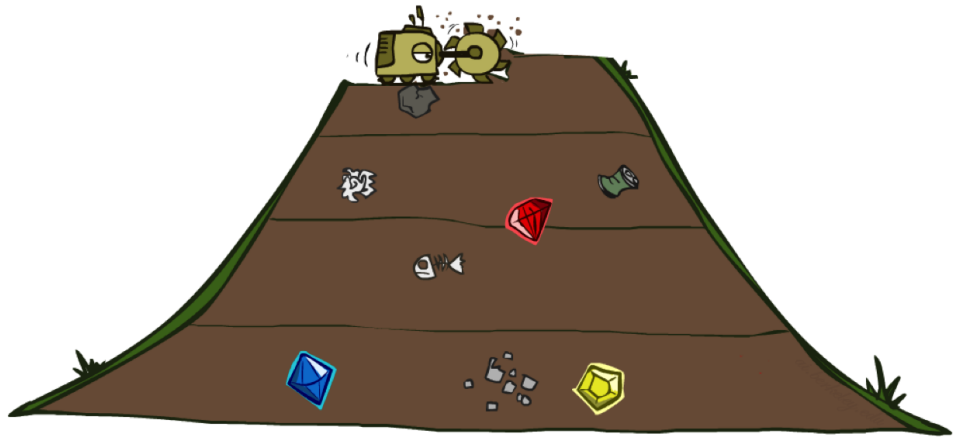


Đánh giá giải thuật tìm kiếm sâu

- Tính đầy đủ: giải thuật không chắc chắn cho lời giải của bài toán trong trường hợp không gian trạng thái của bài toán là vô hạn.
- Độ phức tạp thời gian: O(bm)
- Độ phức tạp không gian: O(b.m)
- Tính tối ưu: giải thuật tìm kiếm theo chiều sâu không cho lời giải tối ưu.



So sánh TK rộng và TK sâu



So sánh TK rộng và TK sâu

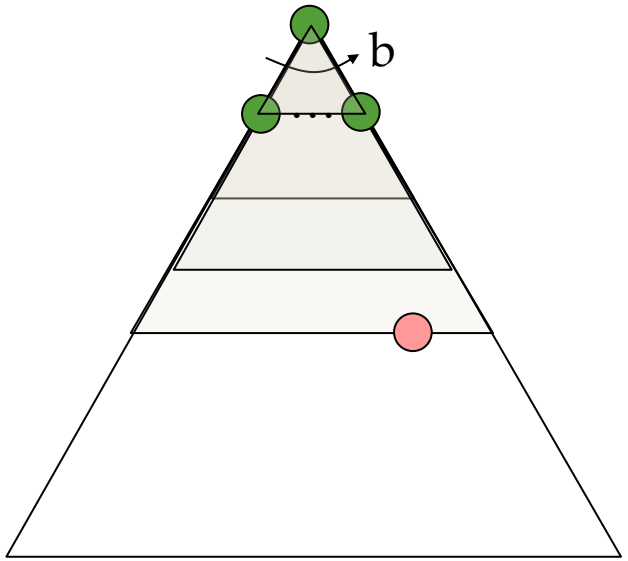
- Giả sử hệ số phân nhánh b
- Theo chiều rộng
  - Đầy đủ (bảo đảm tìm thấy giải pháp)
  - Cần bộ nhớ nhiều hơn
    - Tại độ sâu d cần: $b^0 + b^1 + b^2 + \ldots + b^d \sim b^d$ trạng thái, độ phức tạp không gian $O(b^d)$
- Theo chiều sâu
  - Không đầy đủ vì đường đi vô tận hoặc độ sâu giới hạn
  - Ít tốn bộ nhớ
    - Ở độ sâu d chỉ cần b*d trạng thái, độ phức tạp không gian O(b*d)

Tìm kiếm theo độ sâu có giới hạn (depth-limited search)

- Tương tự như tìm kiếm theo độ sâu, tuy nhiên chỉ tìm đến một độ sâu d nhất định nào đó

Tìm kiếm theo chiều sâu lặp (iterative deepening depth-first search)

- Ý tưởng: lặp lại việc tìm kiếm với độ sâu giới hạn được tăng dần
  - Mỗi lần tăng giới hạn độ sâu lên 1
  - Vd: lần thứ nhất giới hạn độ sâu = 1, nếu tìm không thấy tăng độ sâu giới hạn lên 2, ...

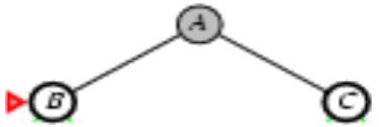


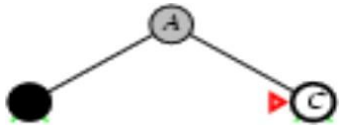
Tìm kiếm sâu lặp, l = 0

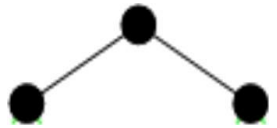
Limit = 0

Tìm kiếm sâu lặp, l = 1

Limit = 1

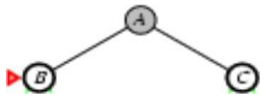


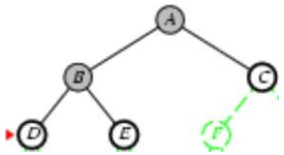


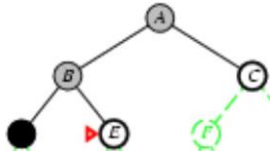


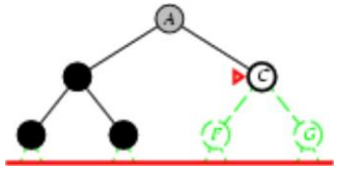
Tìm kiếm sâu lặp, l = 2

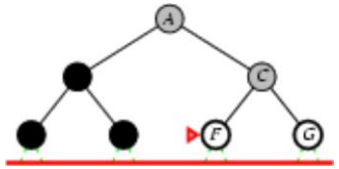
- Limit = 2

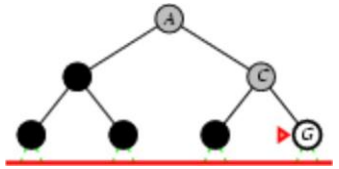


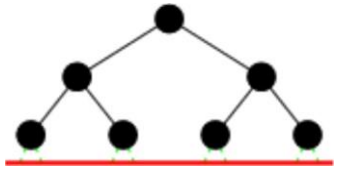




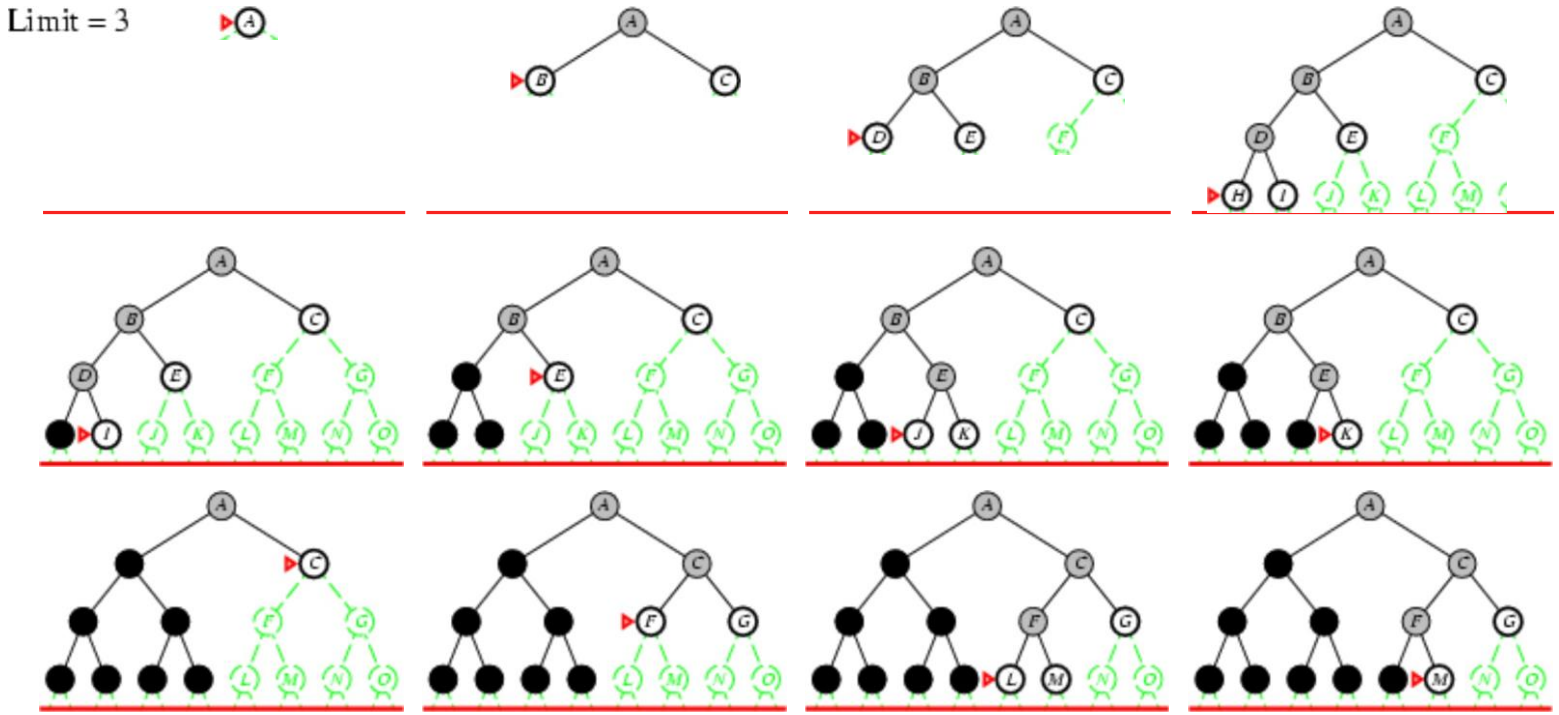




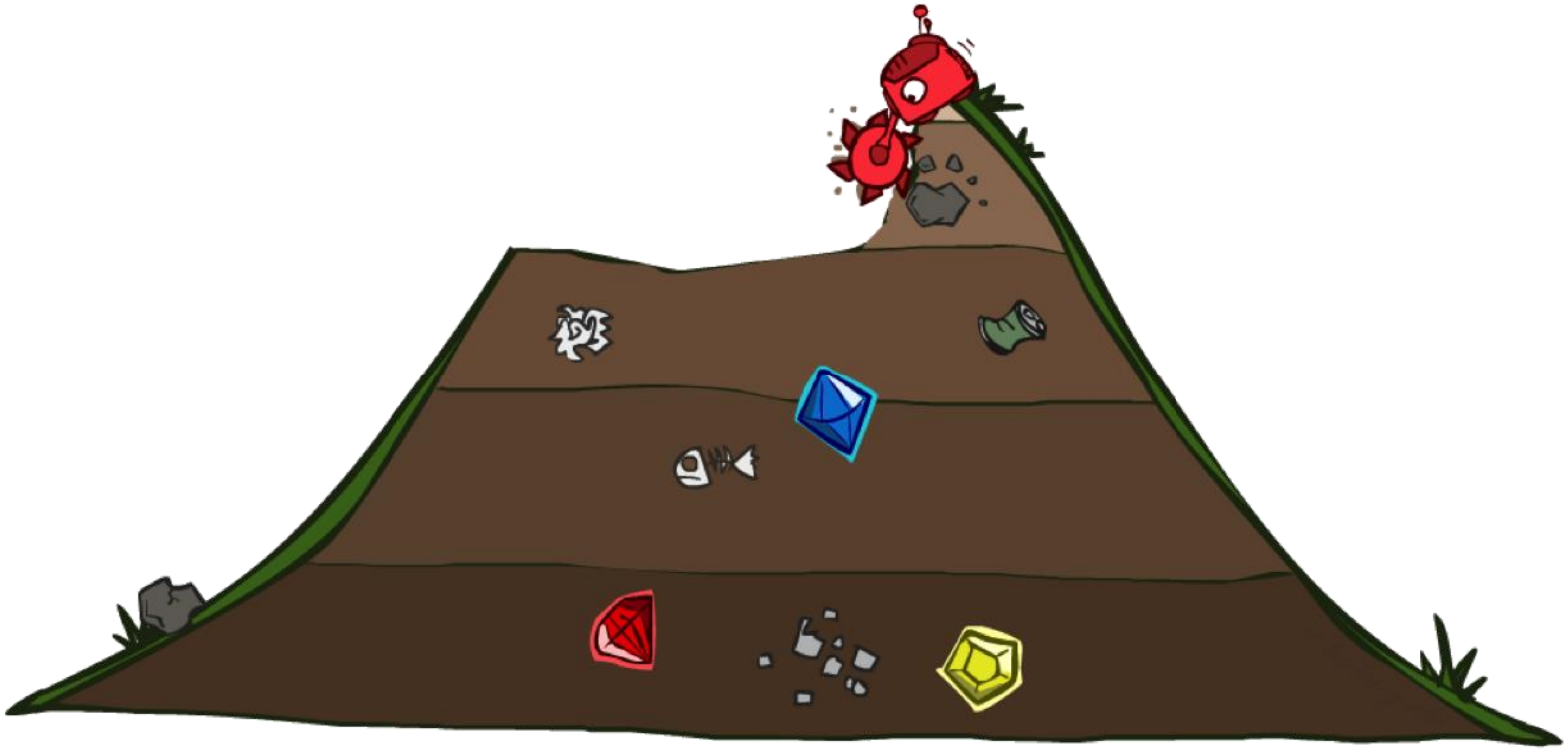




Tìm kiếm sâu lặp, l = 3



Tìm kiếm giá thành đồng nhất (Uniform-cost search)

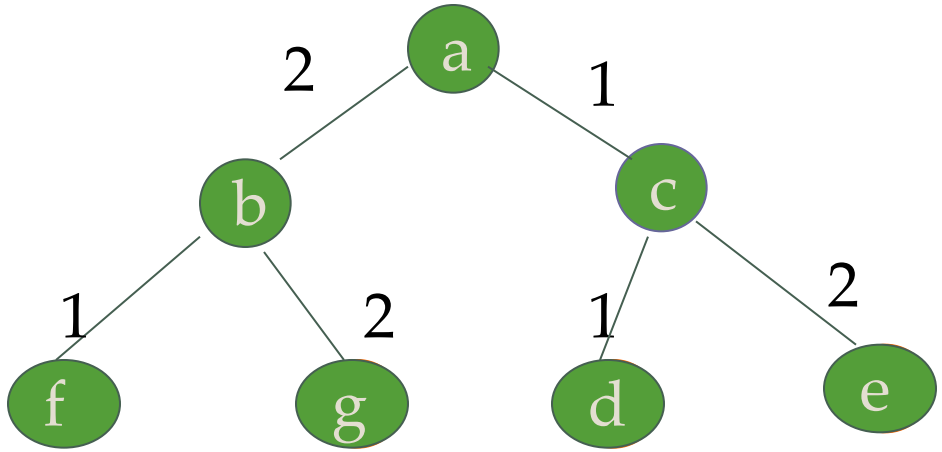


Tìm kiếm giá thành đồng nhất (Uniform-cost search)

- Tìm kiếm theo chiều rộng
  - Đảm bảo tìm ra giải pháp cho bài toán
  - Không chắc tìm ra **đường đi chi phí thấp nhất**
- Tìm kiếm giá thành đồng nhất (rất giống GT Dijsktra)
  - Chọn nút có **giá thành đường đi** đến nó thấp nhất mà triển khai
  - Đảm bảo tìm được giải pháp với chi phí thấp nhất nếu biết rằng chi phí tăng khi chiều dài đường đi tăng
  - Tối ưu và đầy đủ
  - Nhưng có thể rất chậm

Ví dụ Uniform Cost Search

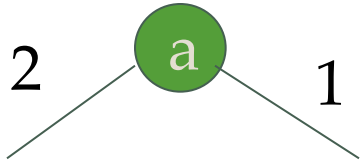
- Cho cây tìm kiếm với chi phí như sau:



Phân biệt:

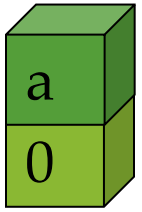
- nút được sinh ra
- nút được mở rộng (duyệt qua)

Ví dụ Uniform Cost Search

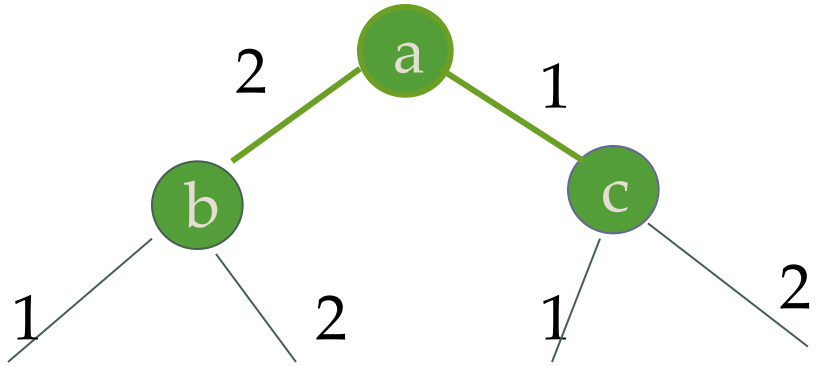


Closed list:

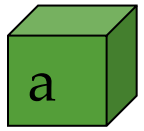
Open list:



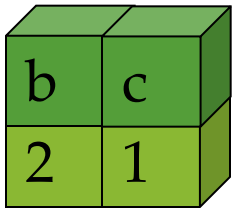
Ví dụ Uniform Cost Search



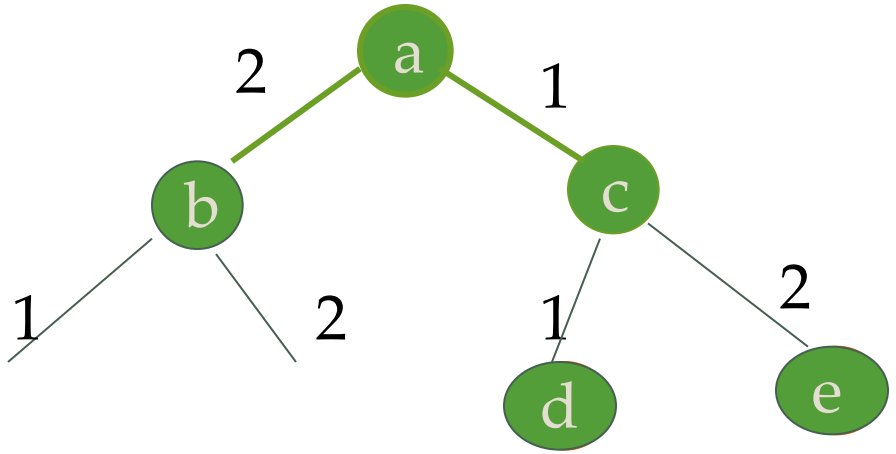
Closed list:



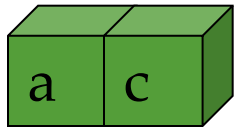
Open list:



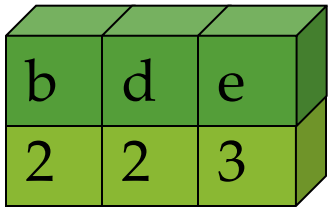
Ví dụ Uniform Cost Search



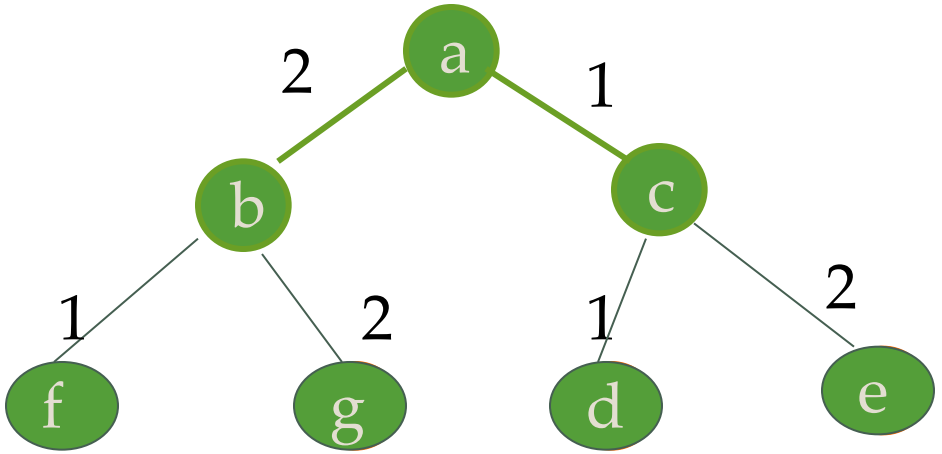
Closed list:



Open list:



Ví dụ Uniform Cost Search



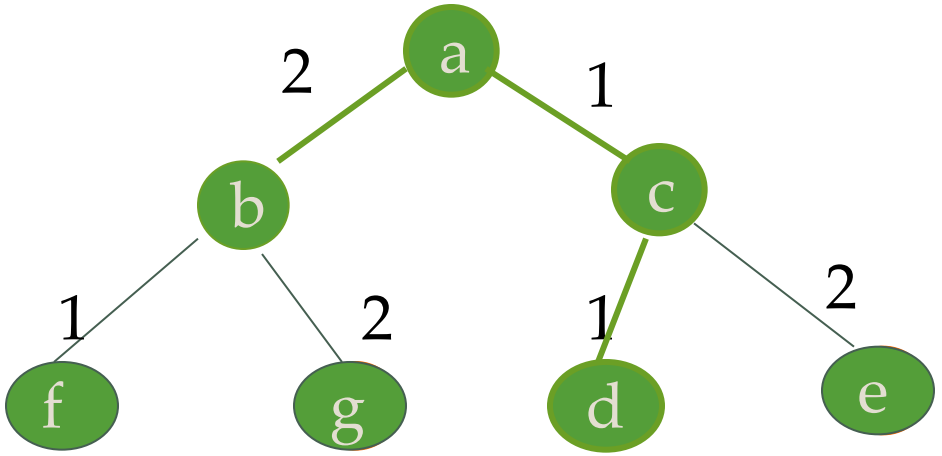
Closed list:

a	c	b

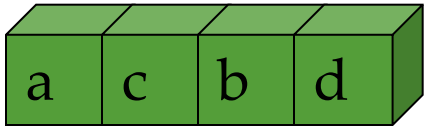
Open list:

d	e	f	g
2	3	3	4

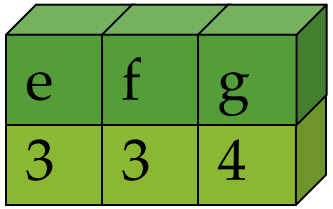
Ví dụ Uniform Cost Search



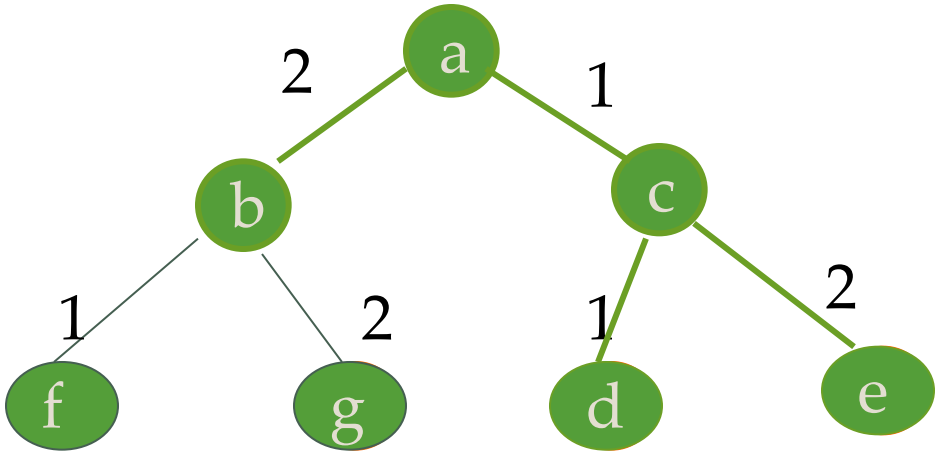
Closed list:



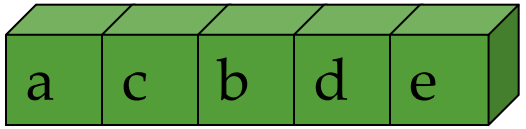
Open list:



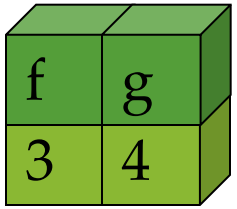
Ví dụ Uniform Cost Search



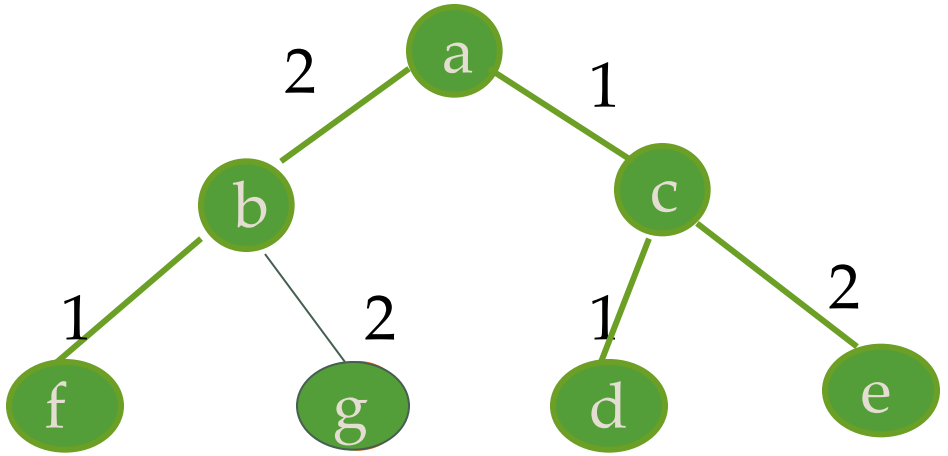
Closed list:



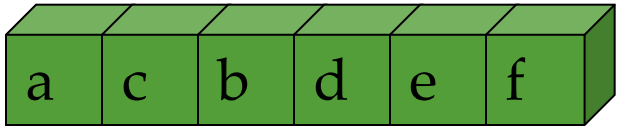
Open list:



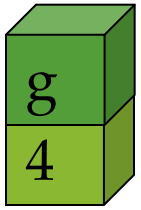
Ví dụ Uniform Cost Search



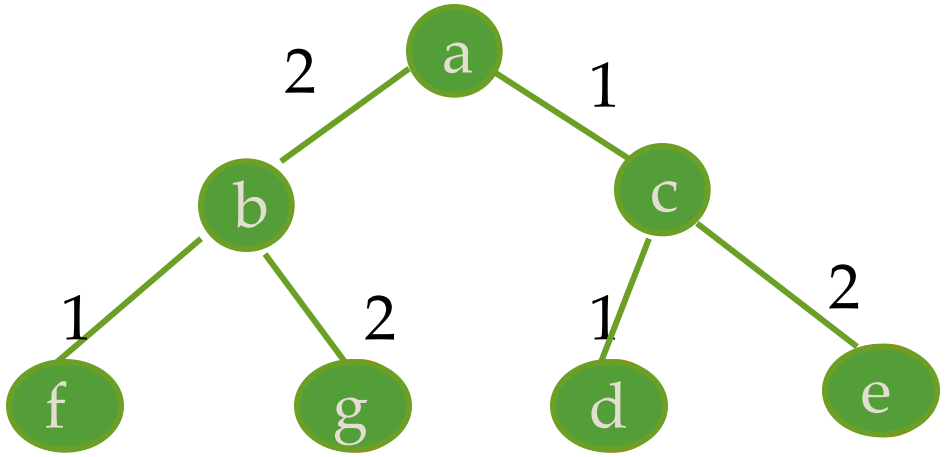
Closed list:



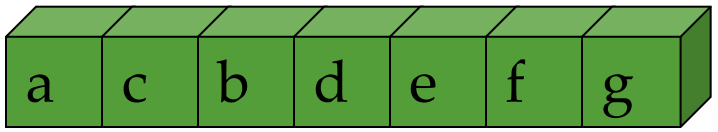
Open list:



Ví dụ Uniform Cost Search



Closed list:

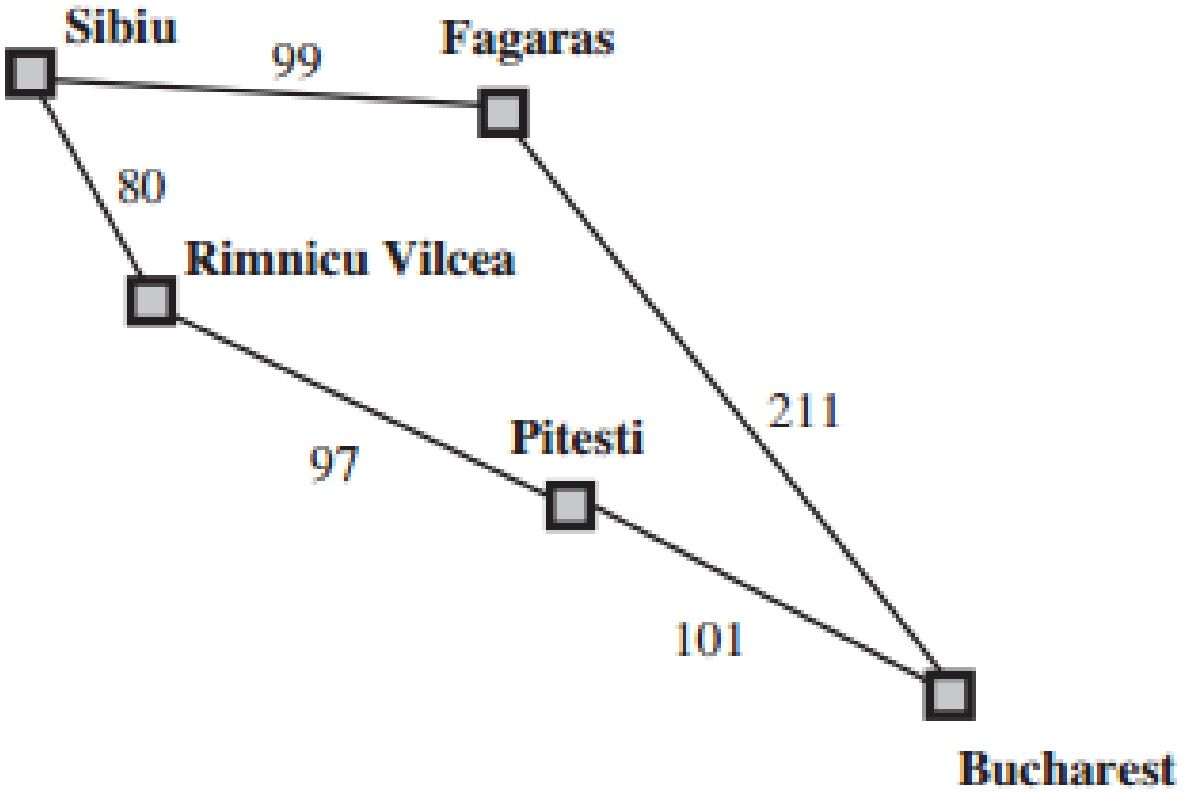


Open list:

Ví dụ Uniform Cost Search

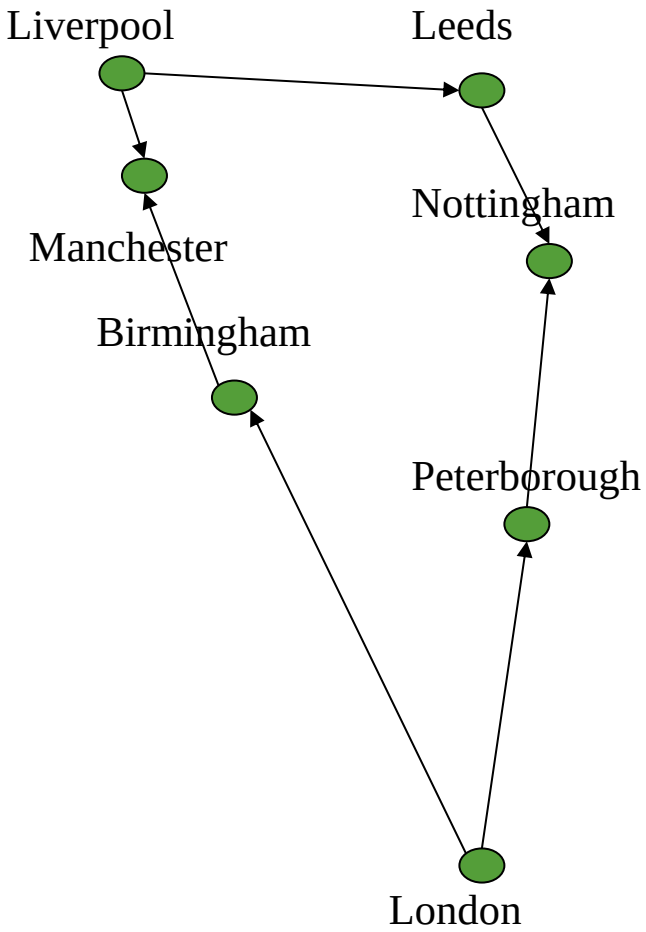
Tìm đường đi ngắn nhất từ Sibiu đến Bucharest?

B1:
Open list: [Sibiu]
Closed list: []
B2:
Open list: [80 R, 99 F]
Closed list: [Sibiu]
B3:
Open list: [99 F, 177 P]
Closed list: [Sibiu, R]
B4:
Open list: [177P, 310B]
Closed list: [Sibiu, R, F]
B5:
Open list: [278B, 310B]
Closed list: [Sibiu, R, F, P]

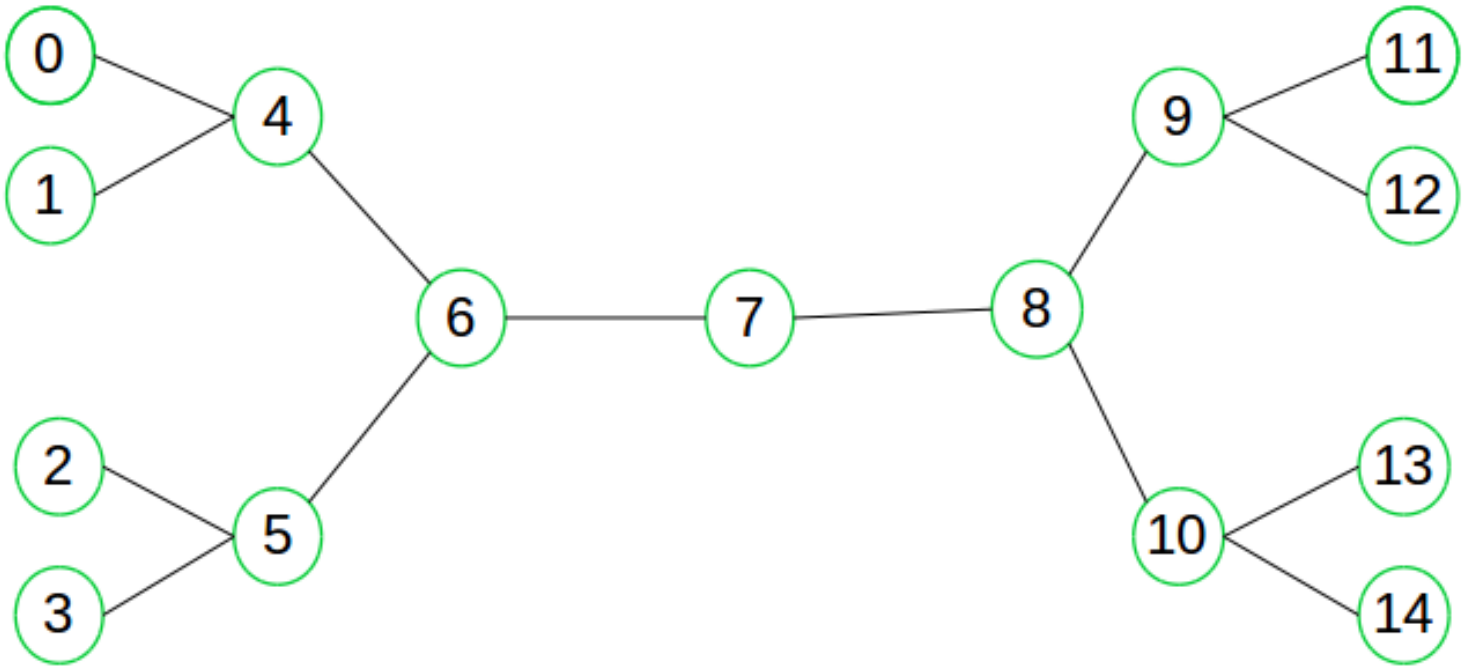


Tìm kiếm hai chiều

- Nếu biết trạng thái đích
  - Tìm kiếm tiến và lùi
  - Gặp nhau ở giữa
- Chỉ cần ½ độ sâu, độ phức tạp không gian $O(b^{d/2})$
- Khó khăn
  - Có thật sự biết được trạng thái đích hay không? Có duy nhất không? (VD: bài toán 8 hậu)
  - Phải có phép toán ngược
  - Lưu tất cả đường đi để kiểm tra xem có gặp nhau chưa
    - Tốn bộ nhớ



Tìm kiếm hai chiều



Cần kiểm tra có đường đi từ 0 tới 14?

- Thực hiện việc tìm kiếm từ 0 và từ 14
- Tìm kiếm tiến và lùi gặp nhau tại 7 => tồn tại đường đi từ 0 đến 14 (tránh được những thao tác không cần thiết)