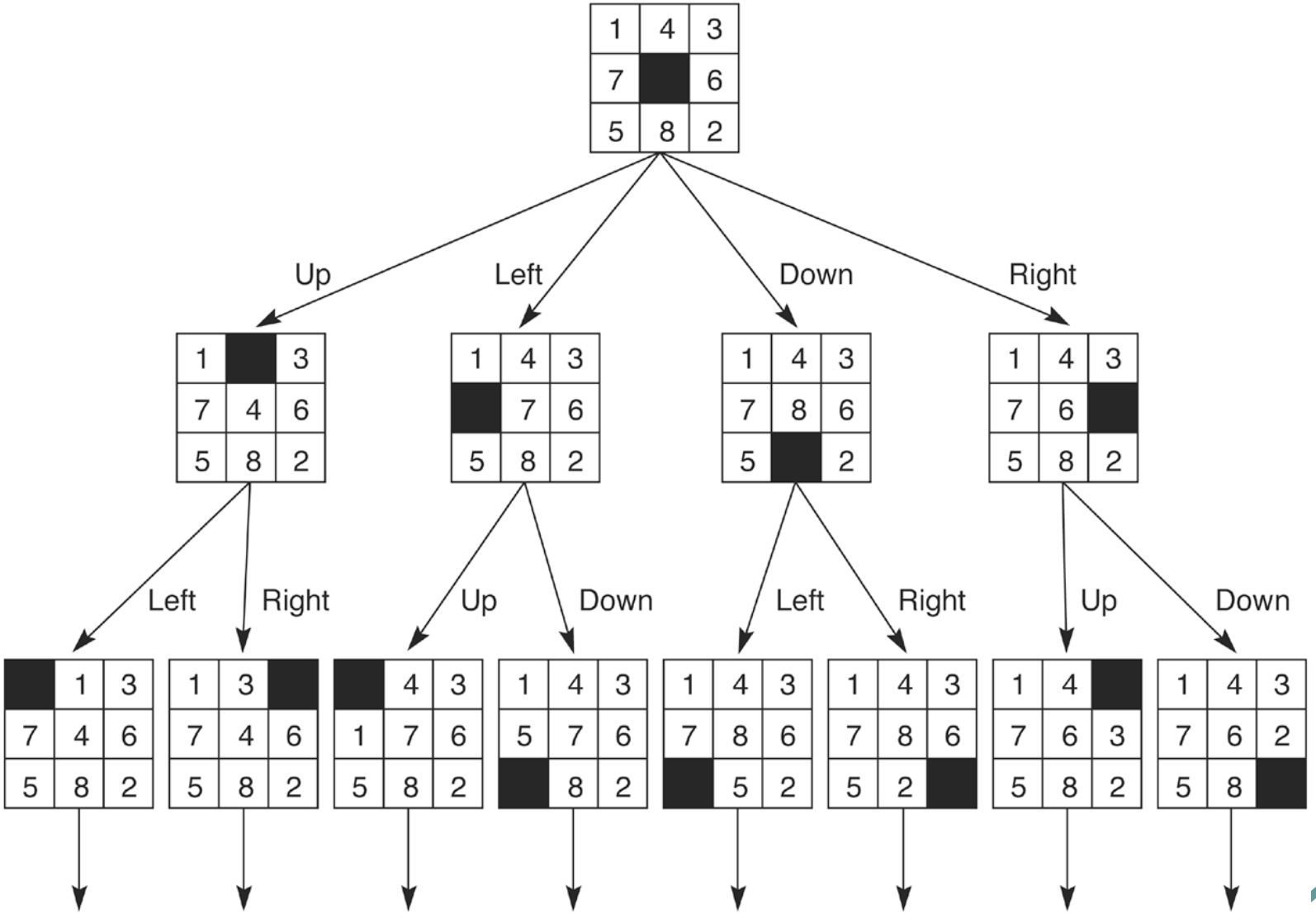
# Tìm kiếm không gian trạng thái

# Tìm kiếm không gian trạng thái

- Định nghĩa bài toán dưới dạng một không gian trạng thái và dùng một giải thuật tìm kiếm để tìm lời giải
- Không gian trạng thái bao gồm:
  - Một *không gian trạng thái* là một tập các trạng thái
  - Một tập các *thao tác* có thể chuyển từ trạng thái này sang trạng thái khác
- Một không gian bài toán có thể được coi như một đồ thị trong đó các đỉnh là các trạng thái, các cạnh là các thao tác

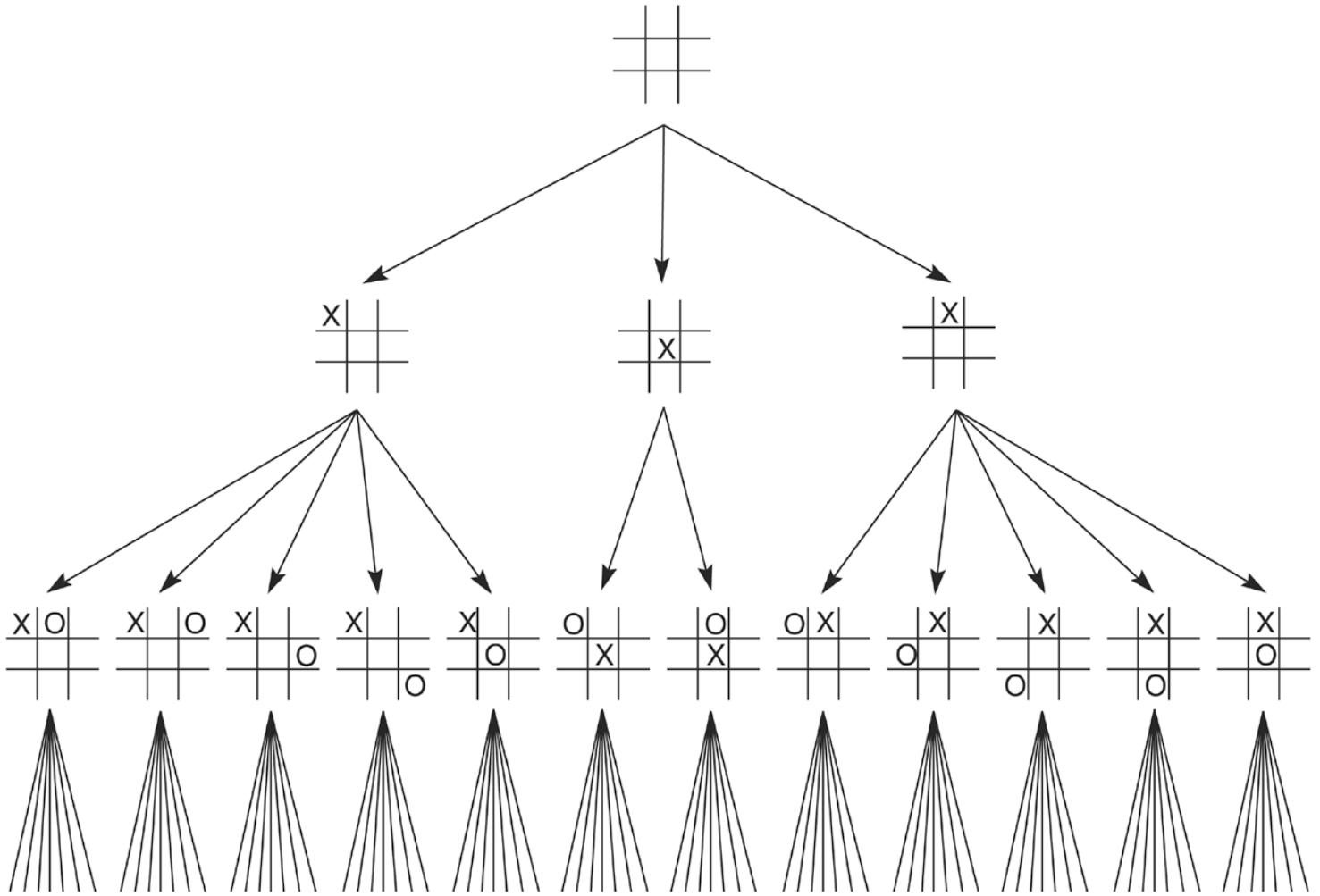
# Không gian trạng thái của 8-puzzle



# Kích thước không gian tìm kiếm: 8/16-puzzle

- 8-puzzle: 8! = 40,320 trạng thái
- 16-puzzle: 16! =20,922,789,888,000 ≈ $10^{13}$ trạng thái
- Thao tác: chuyển các ô
- Đơn giản hóa: giả sử chỉ chuyển các ô trống
- Các thao tác hợp lệ: lên, xuống, trái, phải
- Giữ ô trống trên bàn cờ
- Không gian trạng thái bao gồm hai đồ thị con độc lập

# Không gian trạng thái của tic-tac-toe



# Kích thước không gian tìm kiếm: tic-tac-toe

- Khởi đầu là bàn cờ trống
- Kết thúc là bàn cờ với 3 X một hàng, cột hoặc hàng chéo
- Đường đi từ khởi đầu đến kết thúc bao gồm các nước đi trong ván cờ
- Bộ từ vựng là (blank, X, O)
- $3^9$ = 19,683 cách để sắp xếp (blank, X, O) trong 9 ô trống
- Không tồn tại chu trình: tại sao?
- Biểu diễn bằng DAG (directed acyclic graph)
- 9! = 362,880 đường đi khác nhau có thể sinh ra: tại sao?

# Chiến lược tìm kiếm

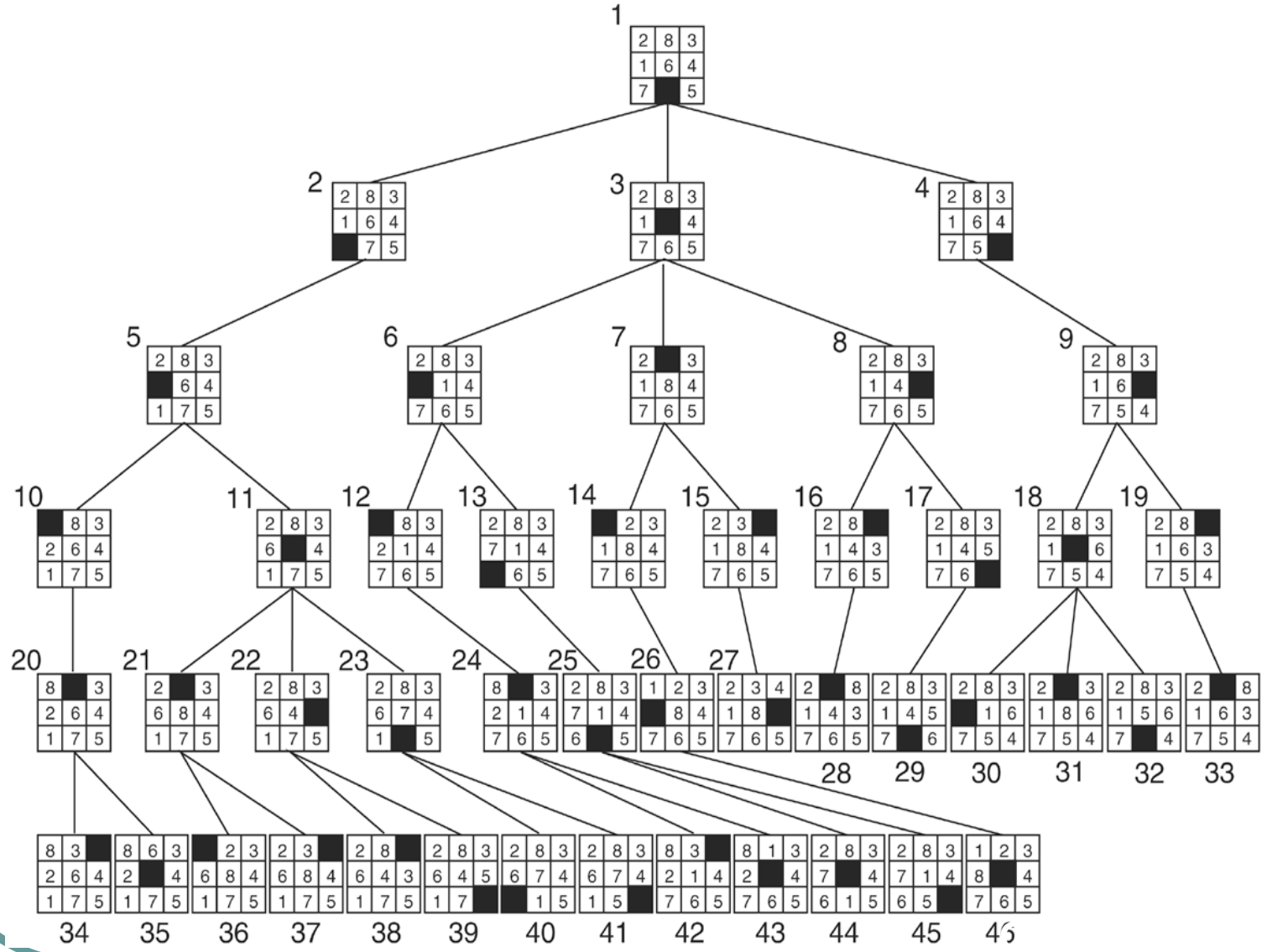
- Duyệt đồ thị từ trạng thái khởi tạo để tìm một giải pháp
- Các chiến lược khác:
  - Theo chiều sâu: thăm con trước anh em (= alg. backtrack)
  - Theo chiều rộng: thăm đồ thị theo từng mức
  - Best-first: sắp xếp các nút chưa thăm theo heuristic, tìm ứng cử viên tốt nhất cho bước tiếp theo

# Breadth-First search

```
function breadth_first_search;

begin
   open := [Start];                                          % initialize
   closed := [ ];
   while open ≠ [ ] do                                       % states remain
      begin
         remove leftmost state from open, call it X;
            if X is a goal then return SUCCESS               % goal found
               else begin
                  generate children of X;
                  put X on closed;
                  discard children of X if already on open or closed;   % loop check
                  put remaining children on right end of open           % queue
               end
      end
   return FAIL                                               % no states left
end.
```

# Breadth-first search of the 8-puzzle



# Bài tập 1

- Viết chương trình in ra lời giải cho trò chơi 8-puzzle sử dụng giải thuật BFS.
- Các câu hỏi cần giải đáp:
  - Làm thế nào để biểu diễn một trạng thái của trò chơi 8-puzzle trong bộ nhớ?
  - Làm thế nào để so sánh hai trạng thái?
  - Làm thế nào để sinh ra các trạng thái con từ một trạng thái?
  - Làm thế nào để lưu các trạng thái trong hai tập hợp (mở và đóng)?
  - Làm thế nào để in một trạng thái ra màn hình?

# Depth first search

```
begin
   open := [Start];                                              % initialize
   closed := [ ];
   while open ≠ [ ] do                                           % states remain
      begin
         remove leftmost state from open, call it X;
         if X is a goal then return SUCCESS                      % goal found
            else begin
               generate children of X;
               put X on closed;
               discard children of X if already on open or closed;   % loop check
               put remaining children on left end of open              % stack
            end
      end;
   return FAIL                                                   % no states left
end.
```

# Theo chiều sâu vs. theo chiều rộng

- Theo chiều rộng:
  - luôn luôn tìm đường đi ngắn nhất
  - không hiệu quả nếu mức độ rẽ nhánh **B** rất cao
  - yêu cầu nhiều bộ nhớ
  - yêu cầu không gian trạng thái theo cấp số mũ: $\mathbf{B}^n$
- Theo chiều sâu:
  - Không phải lúc nào cũng tìm đường đi ngắn nhất
  - hiệu quả nếu lời giải dài
  - tuy nhiên có thể bị "lạc", với chiều sâu vô hạn
  - chỉ cần bộ nhớ cho một đường đi: $\mathbf{B} \times n$

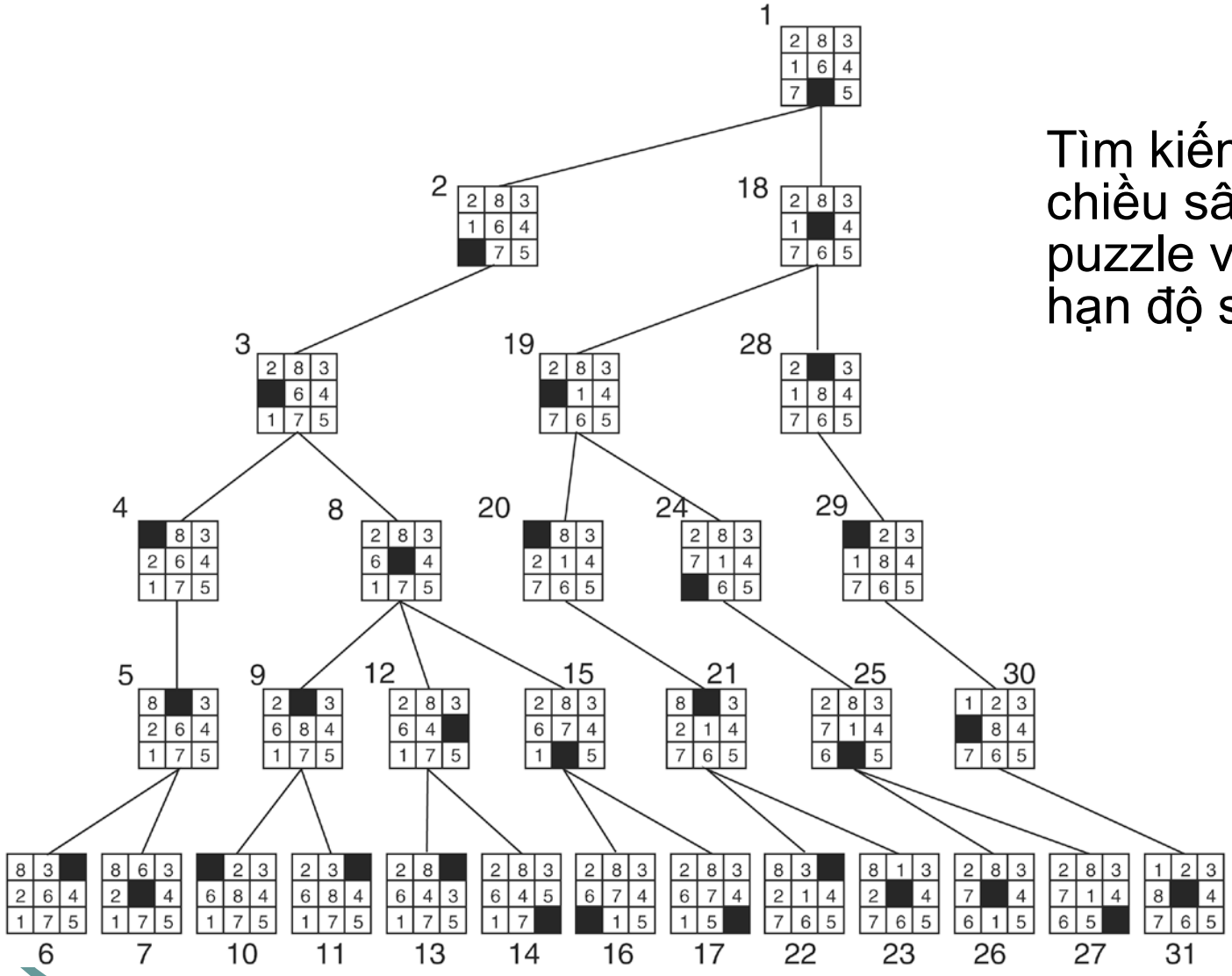
# Iterative deepening

Giải pháp thỏa hiệp:

- sử dụng tìm kiếm theo chiều sâu, nhưng
- với độ sâu tối đa trước khi đi sang mức tiếp theo

$\rightarrow$ *Deptheo-first Iterative Deepening*

# Tìm kiếm theo chiều sâu cho 8-puzzle




Tìm kiếm theo chiều sâu cho 8-puzzle với giới hạn độ sâu = 5

# Bài tập 2

- Viết lại chương trình trong Bài tập 1 sử dụng giải thuật DFS.
- So sánh lời giải sử dụng hai chiến thuật