

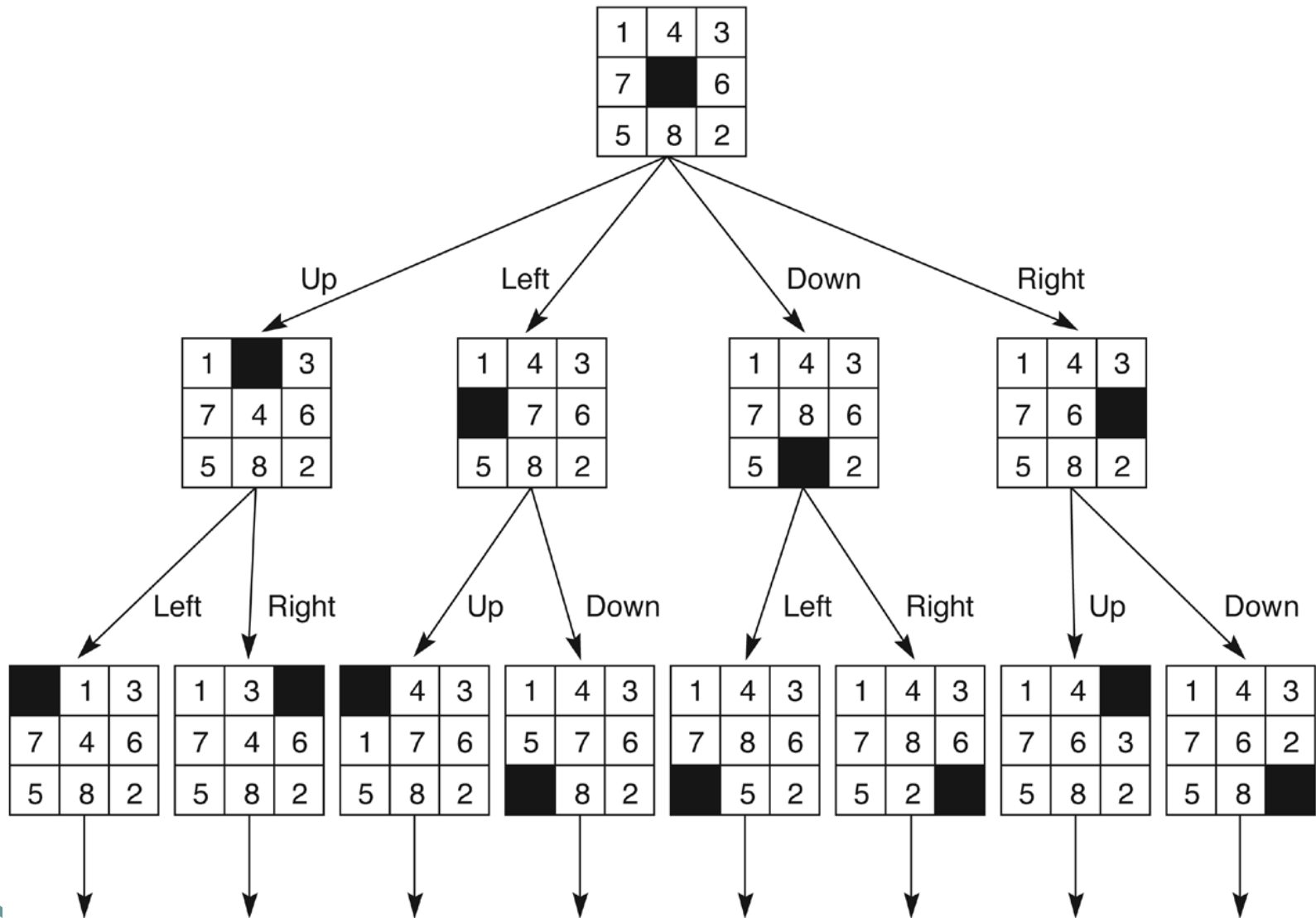
# State space search

`anhtt-fit@mail.hut.edu.vn`

# State Space Search

- Define problem in form of a state space and use a search algorithm to find a solution
- The problem space consists of:
  - a *state space* which is a set of states representing the possible configurations of the world
  - a set of *operators* which can change one state into another
- The problem space can be viewed as a graph where the states are the nodes and the arcs represent the operators.

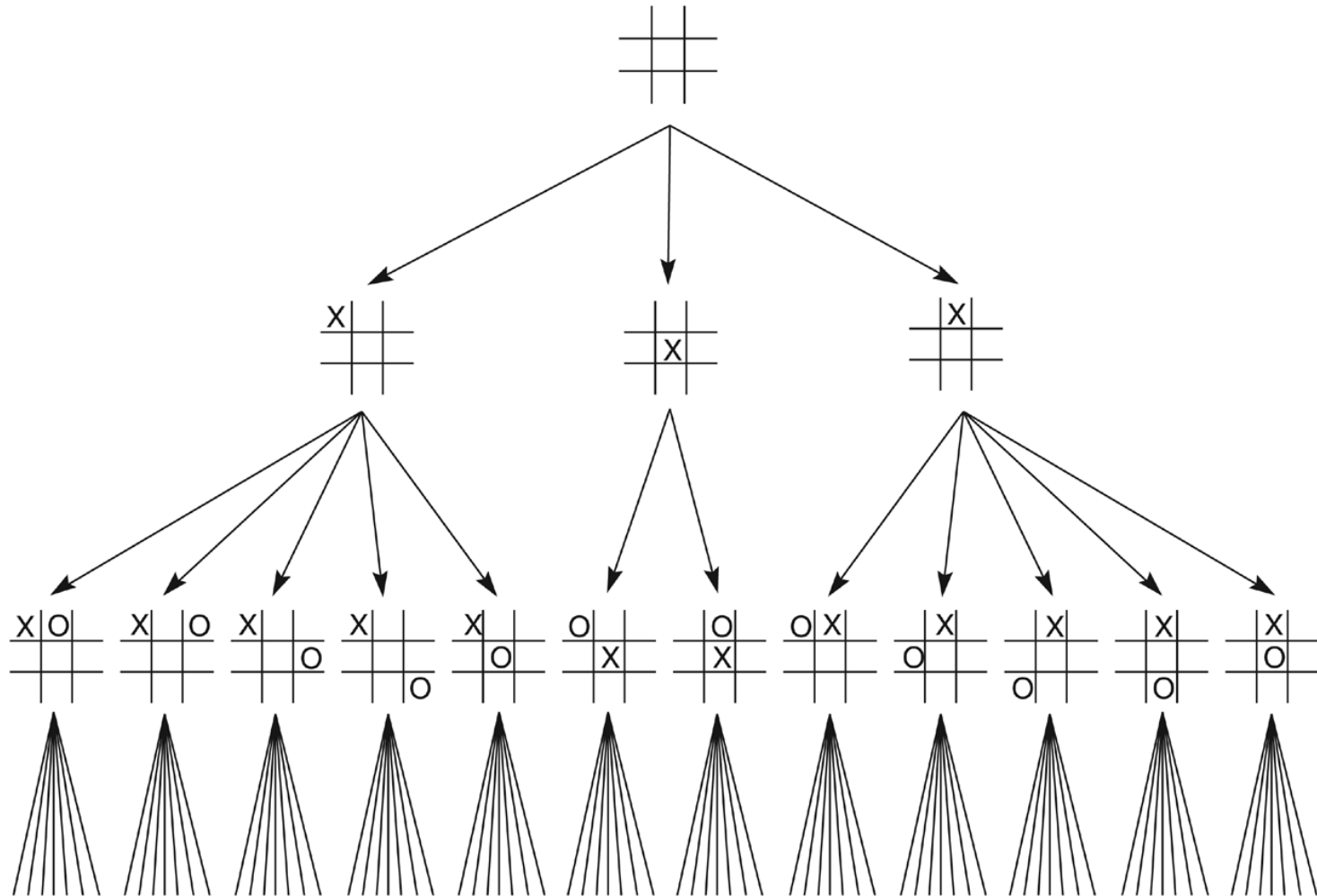
# State space of the 8-puzzle



# Size of search space: 8/16-puzzle

- 8-puzzle:  $8! = 40,320$  different states
- 16-puzzle:  $16! = 20,922,789,888,000 \approx 10^{13}$  different states
- Game works by moving tiles
- Simplification: assume only blank tile is moved
- Legal moves: blank up, down, left, right
- Keep blank tile on board
- State space consists of two disconnected subgraphs

# State space of tic-tac-toe



# Size of search space: tic-tac-toe

- Start is empty board
- Goal is board with 3 Xs in a row, column or diagonal
- Path from start to end gives a series of moves in a winning game
- Vocabulary is (blank, X, O)
- $3^9 = 19,683$  ways to arrange (blank, X, O) in 9 spaces
- No cycles possible: why?
- Represented as DAG (directed acyclic graph)
- $9! = 362,880$  different paths can be generated: why?

# Search Strategies

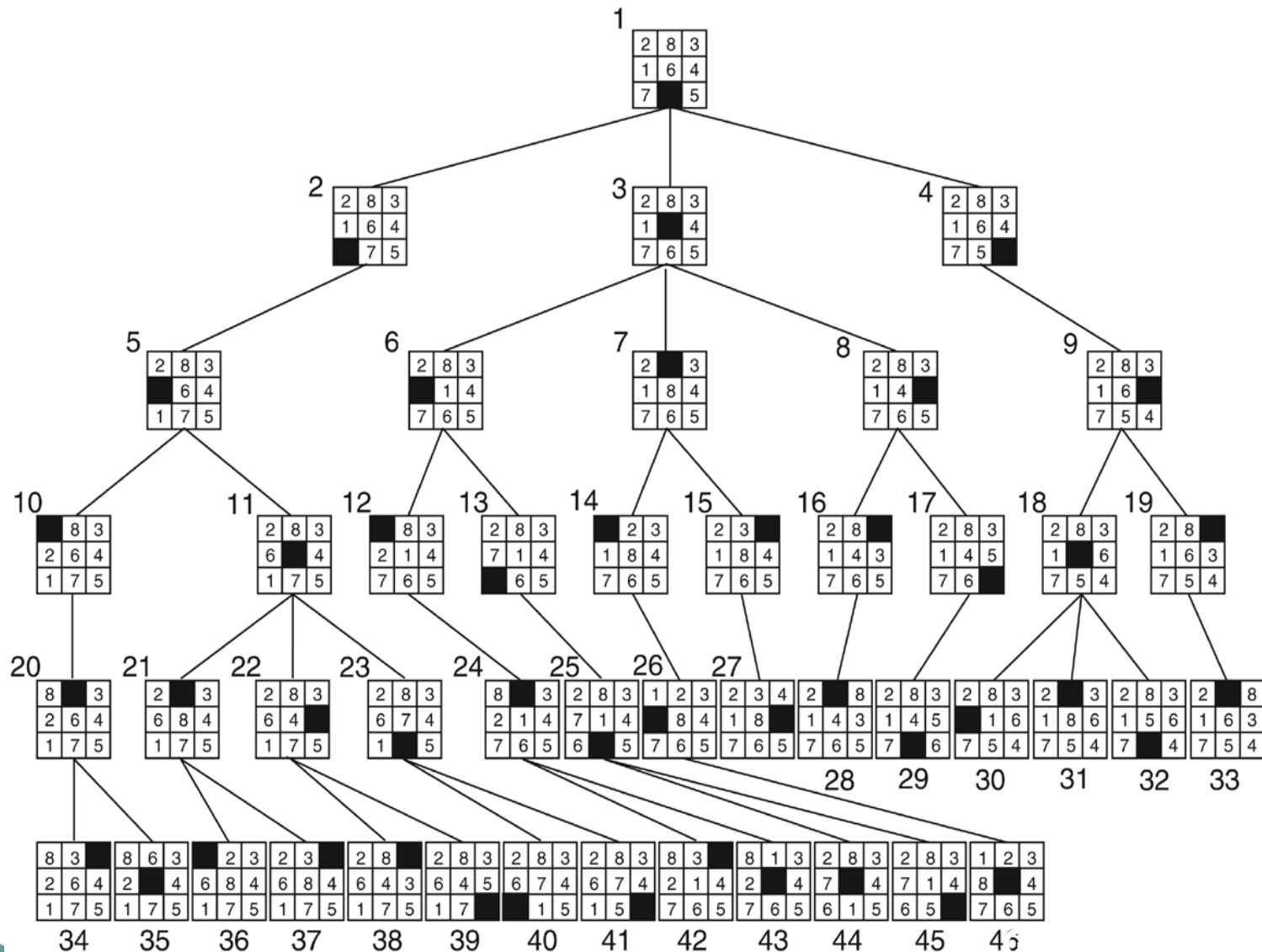
- Traverse the graph from an initial state to find a goal
- Alternative search strategies:
  - Depth-first: visit children before siblings (= alg. backtrack)
  - Breadth-first: visit graph level-by-level
  - Best-first: order unvisited nodes through heuristic, finding best candidate for next step

# 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;  
                put remaining children on right end of open  
            end  
        end  
    end  
    return FAIL  
end.  
% no states left
```



# Breadth-first search of the 8-puzzle



# Quiz 1

- Write a program to print out solutions for the 8-puzzle game using the BFS algorithm.
- Question to solve:
  - How to represent a state of 8-puzzle game in memory?
  - How to compare two states?
  - How to generate sub-states from a state?
  - How to store states in two collections (open and closed)?
  - How to print a state in the screen?

# 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;
        put remaining children on left end of open % loop check
                                                    % stack
      end
    end
  end;
  return FAIL
end.
```

% no states left

# Depth-first vs. breadth-first

- Breadth-first:
  - always finds shortest path
  - inefficient if branching factor **B** is very high
  - memory requirements high
  - exponential space for states required:  $B^n$
- Depth-first:
  - does not always find shortest path
  - efficient if solution path is known to be long
  - but can get „lost“ in (infinitely) deep paths
  - only memory for states of one path needed:  $B \times n$

# Iterative Deepening

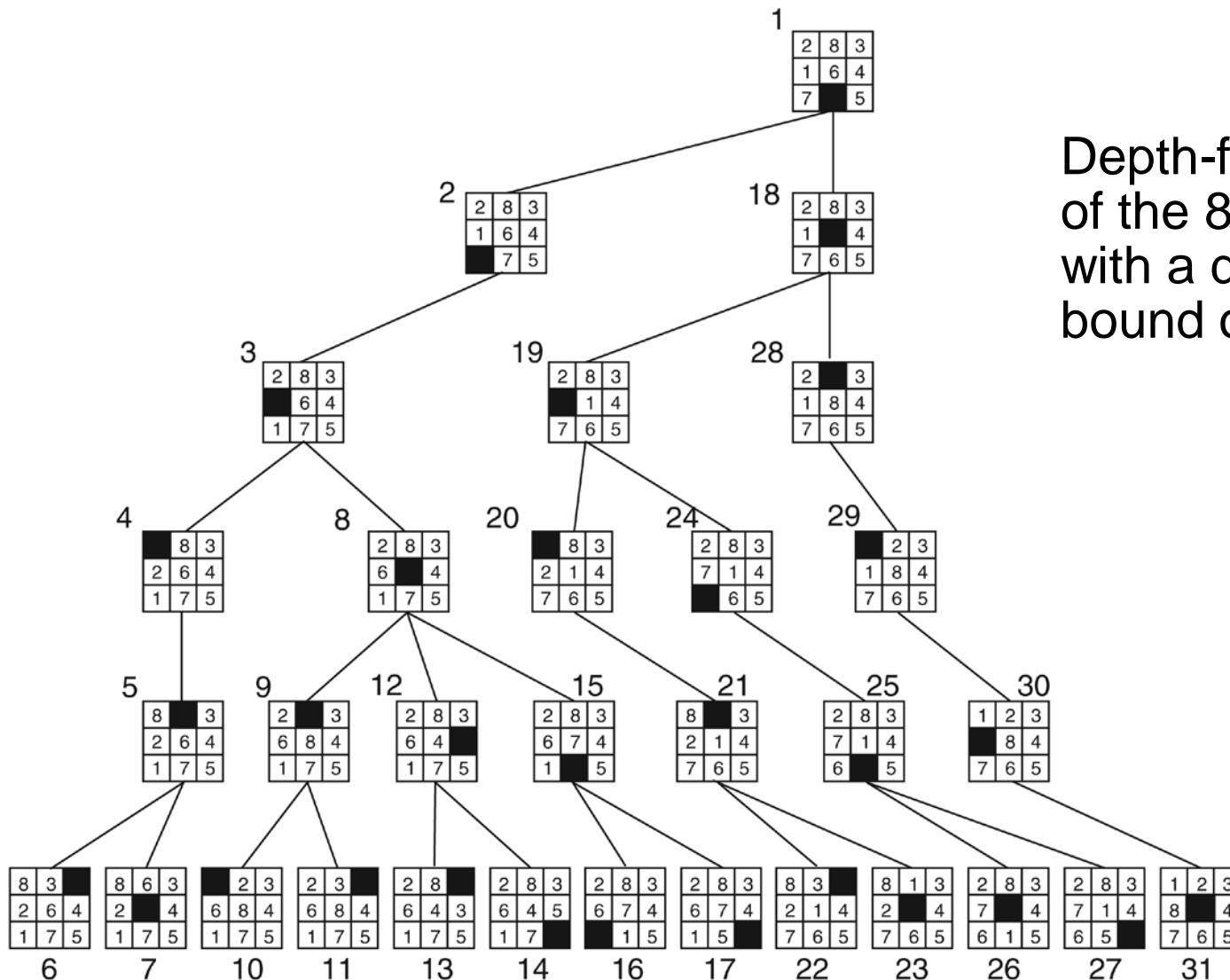
Compromise solution:

- use depth-first search, but
- with a maximum depth before going to next level

→ *Depth-first Iterative Deepening*

# Depth-first search of the 8-puzzle

Depth-first search  
of the 8-puzzle  
with a depth  
bound of 5



## Quiz 2

- Rewrite the program in Quiz 1 using the DFS algorithm.
- Compare the solution given by the two strategies.