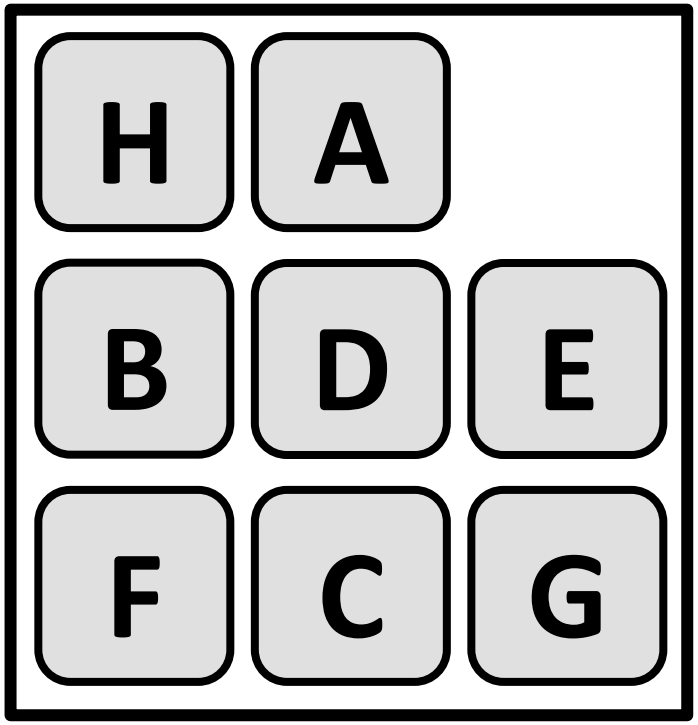
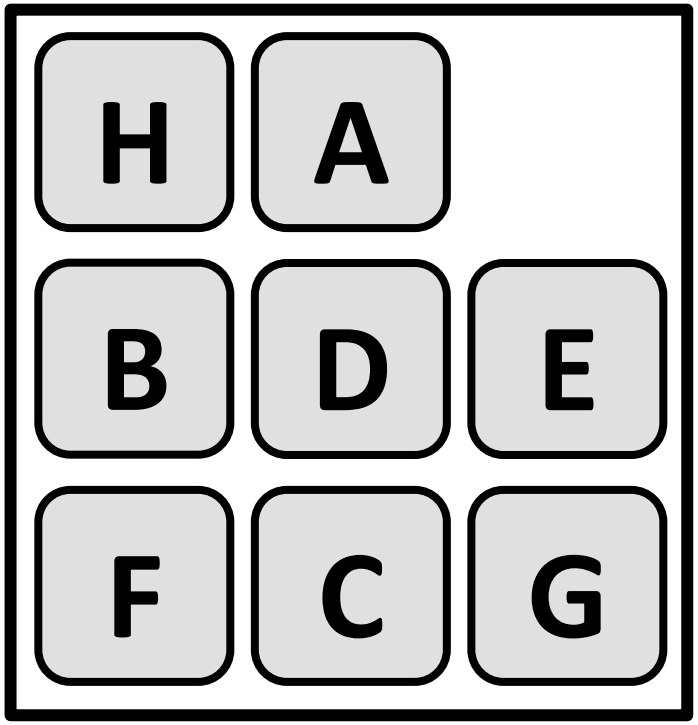
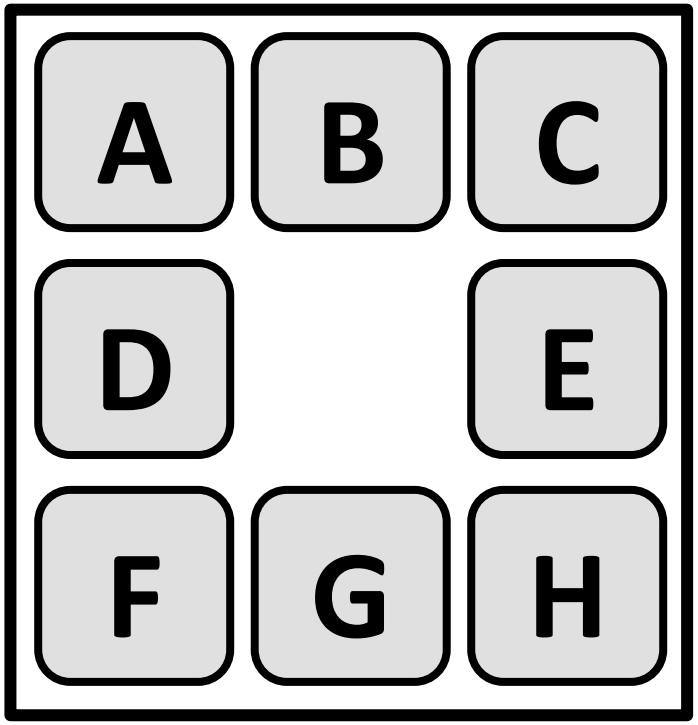
COMP10020
Introduction to Programming II
Putting It All Together - 8s Puzzle

Dr. Brian Mac Namee
brian.macnamee@ucd.ie
School of Computer Science
University College Dublin

THE 8S PUZZLE


H
A
B
D
E
F
C
G

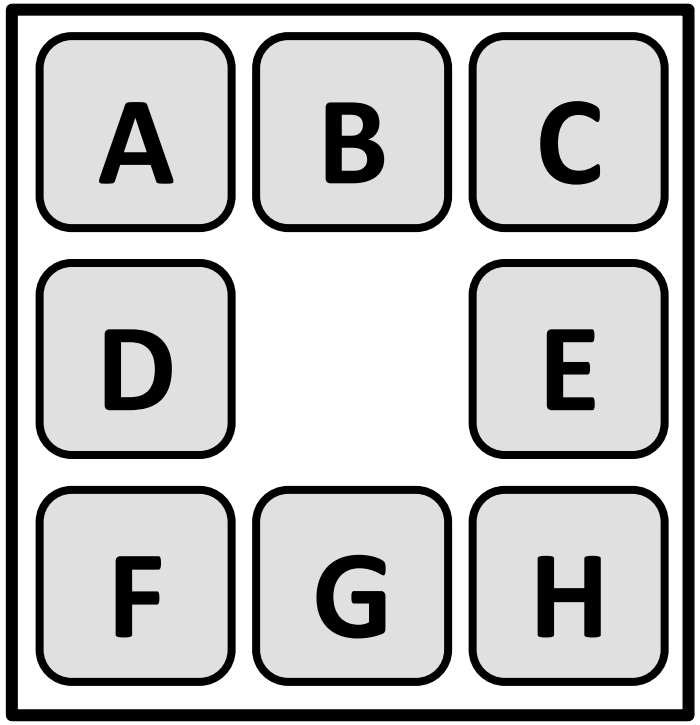

H
A
B
D
E
F
C
G


A
B
C
D
E
F
G
H

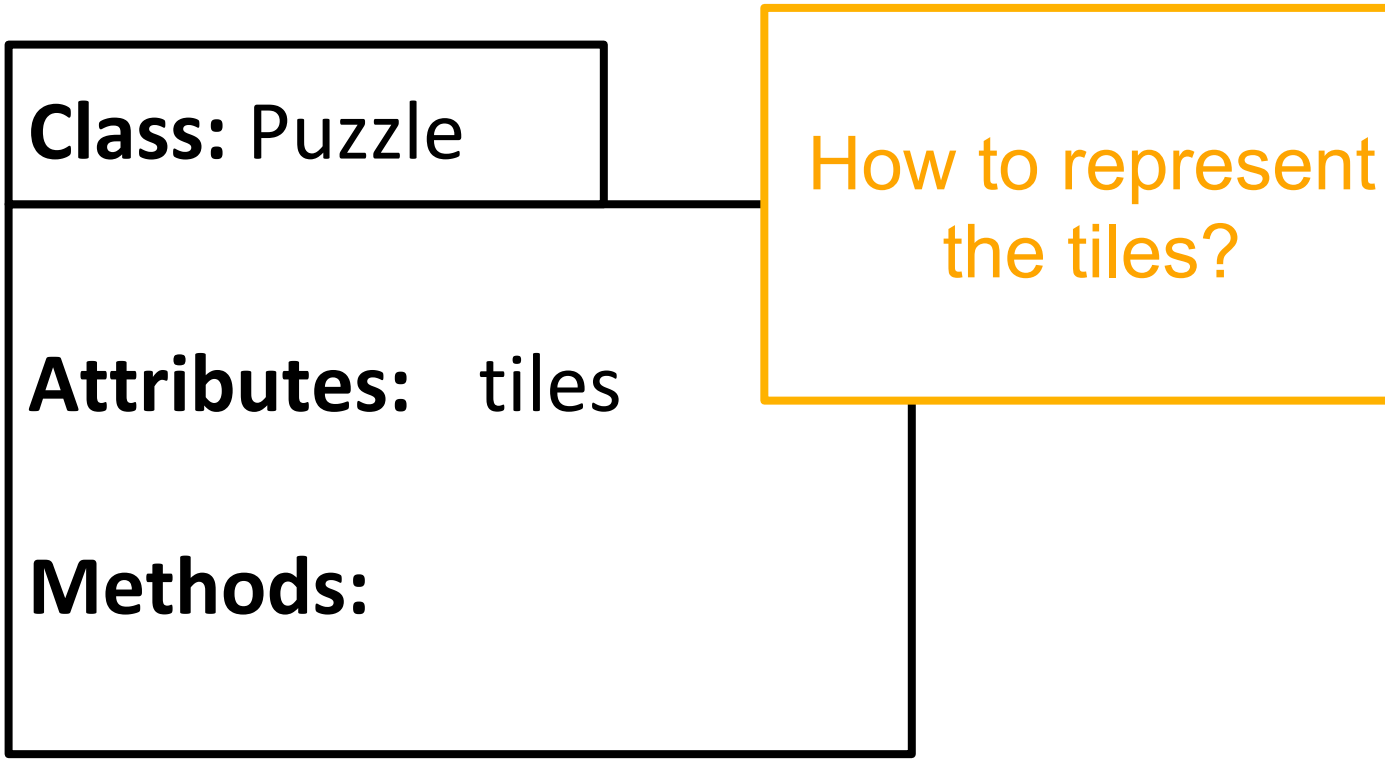

10 9 3
2 5 15 7
8 14 1 11
12 13 4 6


孔明
曹操
張飛
劉備
關羽
趙雲
兵
兵
兵
兵
華容道
放曹遊戲

REPRESENTING THE PUZZLE BOARD


A
B
C
D
E
F
G
H

8s Puzzle


Class: Puzzle
Attributes: tiles
Methods:
How to represent the tiles?

Class Attribute

```
class Puzzle8():

    blankChar = "  "

    # Creat the tile board as a list
    def __init__(self):
      self.tiles = [["A", "B", "C"], ["D", Puzzle8.blankChar, "E"],
                                                    ["F", "G", "H"]]

    # Print the game board
    def printTiles(self):
        print("-------------")
      print("| " + self.tiles[0][0] + " | " + self.tiles[0][1] + " | " + self.tiles[0][2]  + " |")
      print("-------------")
      print("| " + self.tiles[1][0] + " | " + self.tiles[1][1] + " | " + self.tiles[1][2]  + " |")
      print("-------------")
      print("| " + self.tiles[2][0] + " | " + self.tiles[2][1] + " | " + self.tiles[2][2]  + " |")
      print("-------------")
```

Class Attribute

```
class Puzzle8():

    blankChar = " "

    # Creat the tile board as a list
    def __init__(self):
      self.tiles = [["A", "B", "C"], ["D", Puzzle8.blankChar, "E"],
                                                  ["F", "G", "H"]]

    # Print the game board
    def printTiles(self):
        print("-------------")
      print("| " + self.tiles[0][0] + " | " + self.tiles[0][1] + " | " + self.tiles[0][2]  + " |")
      print("-------------")
      print("| " + self.tiles[1][0] + " | " + self.tiles[1][1] + " | " + self.tiles[1][2]  + " |")
      print("-------------")
      print("| " + self.tiles[2][0] + " | " + self.tiles[2][1] + " | " + self.tiles[2][2]  + " |")
      print("-------------")
```

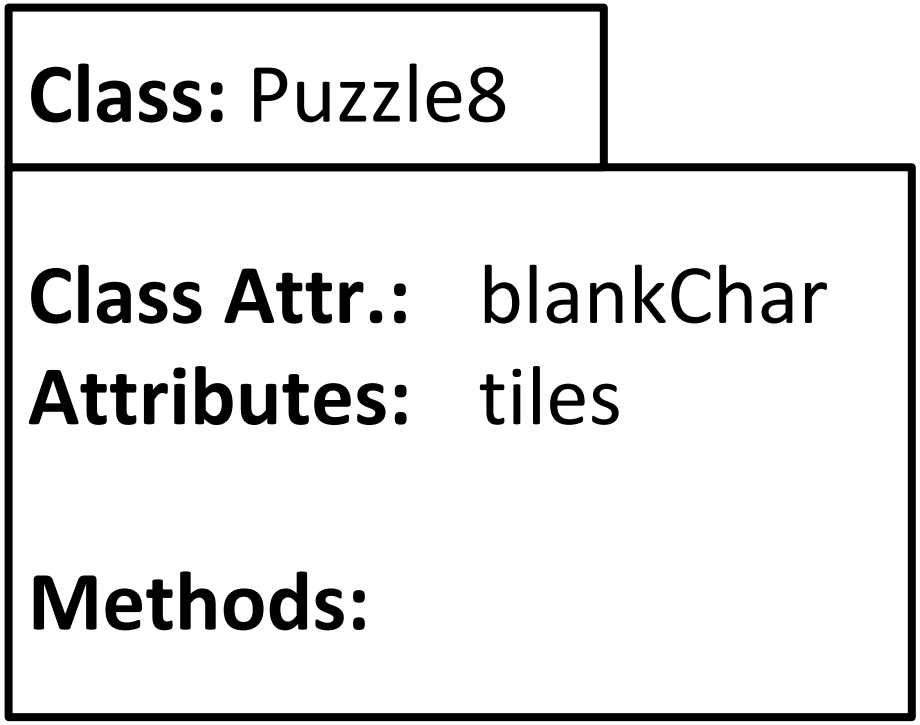
An attribute declared here is a class attribute - all instances of this class will share this variable

Class Attribute

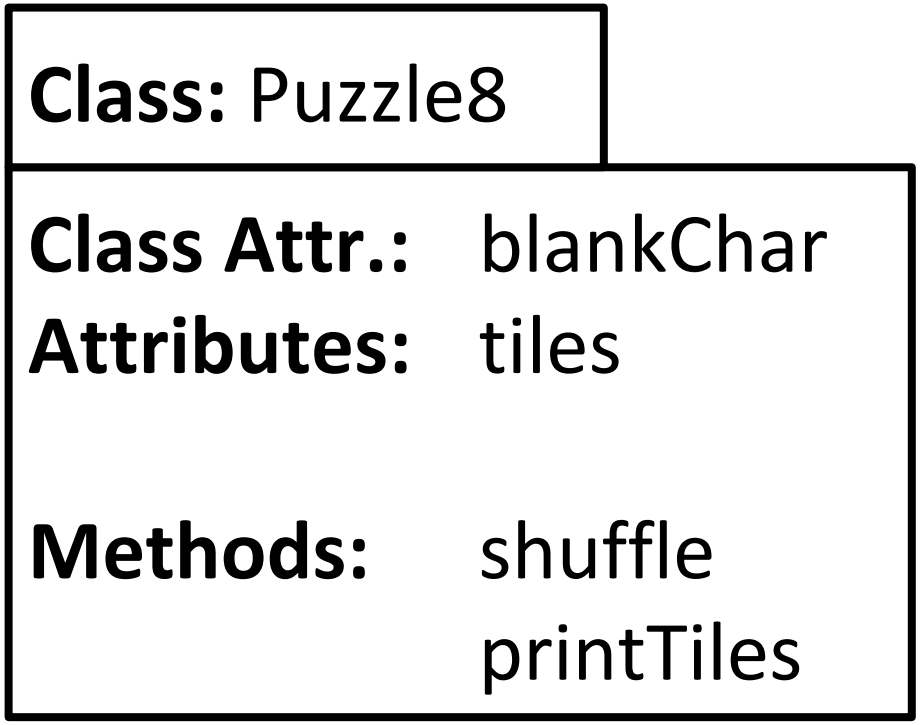
```
class Puzzle8():

    blankChar = " "

    # Creat the tile board as a list
    def __init__(self):
      self.tiles = [["A", "B", "C"], ["D", Puzzle8.blankChar, "E"],
                                                  ["F", "G", "H"]]

    # Print the game board
    def printTiles(self):
        print("-------------")
      print("| " + self.tiles[0][0] + " | " + self.tiles[0][1] + " | " + self.tiles[0][2]  + " |")
      print("-------------")
      print("| " + self.tiles[1][0] + " | " + self.tiles[1][1] + " | " + self.tiles[1][2]  + " |")
      print("-------------")
      print("| " + self.tiles[2][0] + " | " + self.tiles[2][1] + " | " + self.tiles[2][2]  + " |")
      print("-------------")
```

Great for constant values etc that instances of an object might share

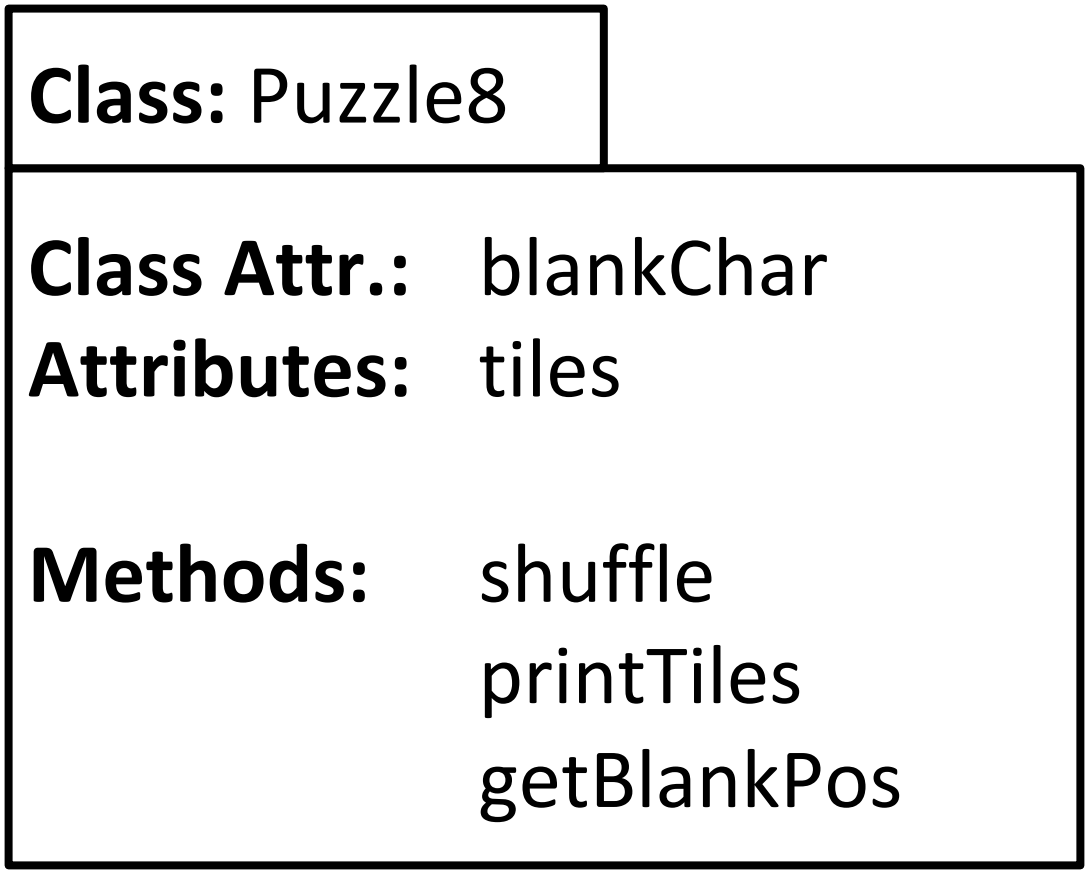
8s Puzzle


Class: Puzzle8
Class Attr.: blankChar
Attributes: tiles
Methods:

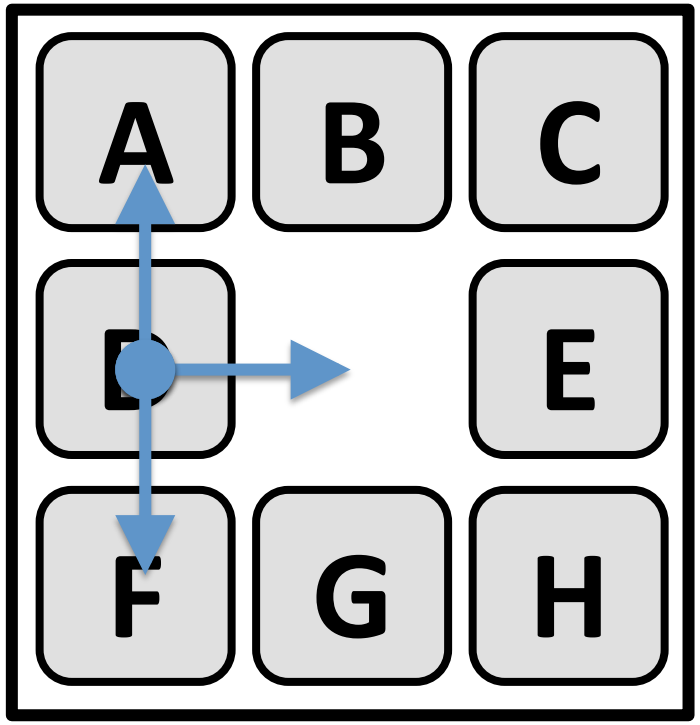
8s Puzzle

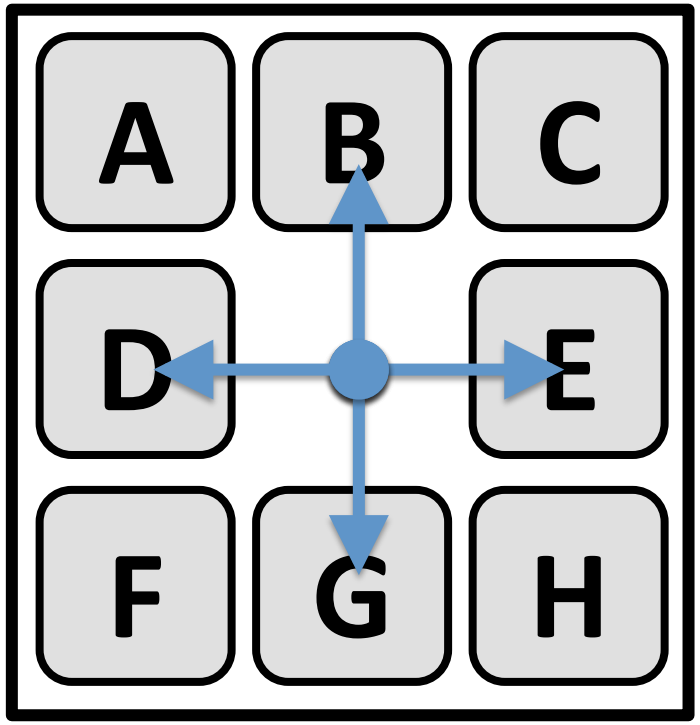

Class: Puzzle8
Class Attr.: blankChar
Attributes: tiles
Methods: shuffle
printTiles

8s Puzzle


Class: Puzzle8
Class Attr.: blankChar
Attributes: tiles
Methods: shuffle
printTiles
getBlankPos

MAKING MOVES ON THE PUZZLE BOARD


A
B
C
D
E
F
G
H


A
B
C
D
E
F
G
H

8s Puzzle

Class: Puzzle8

Class Attr.: blankChar

Attributes: tiles

Methods:

shuffle	moveBlankLeft
printTiles	moveBlankRight
getBlankPos	moveBlankUp
moveBlank	moveBlankDown

CHECKING FOR A WINNER

8s Puzzle

Class: Puzzle8

Class Attr.: blankChar

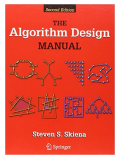
Attributes: tiles

Methods:

shuffle	moveBlankLeft
printTiles	moveBlankRight
getBlankPos	moveBlankUp
moveBlank	moveBlankDown
matchTemplate	

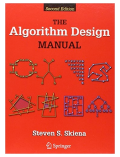
SOLVING THE PUZZLE

How to Design Algorithms



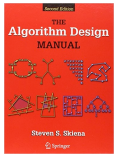
1. Do I really understand the problem?
2. Can I find a simple algorithm or heuristic for the problem?
3. Is my problem in the catalog of well known algorithmic problems (e.g. those in The Algorithm Design Manual)?
4. Are there special cases of the problem that I know how to solve exactly?
5. Which of the standard algorithm design paradigms are most relevant to my problem?
6. Am I still stumped?

How to Design Algorithms

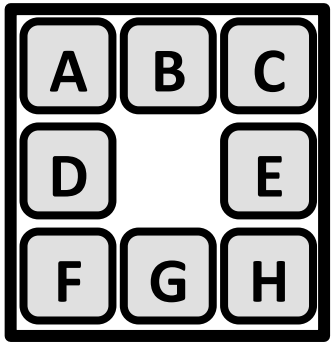


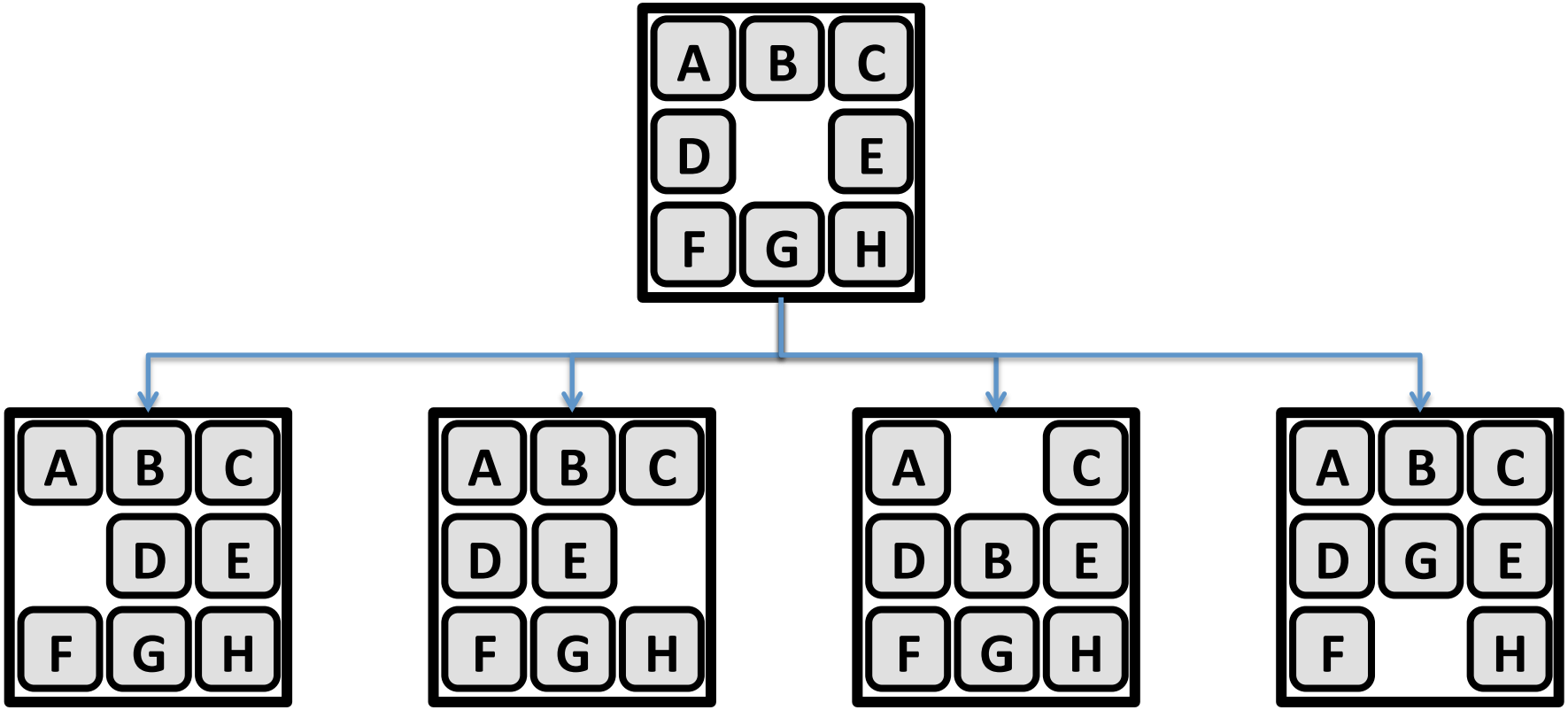
1. Do I really understand the problem?
   - What exactly does the input consist of?
   - What exactly are the desired results or output?
   - Can I construct an example input small enough to solve by hand? What happens when I try to solve it?
   - How important is it to my application that I always find an exact, optimal answer? Can I settle for something that is usually pretty good?
   - How large will a typical instance of my problem be? Will I be working on 10 items? 1,000 items? 1,000,000 items?

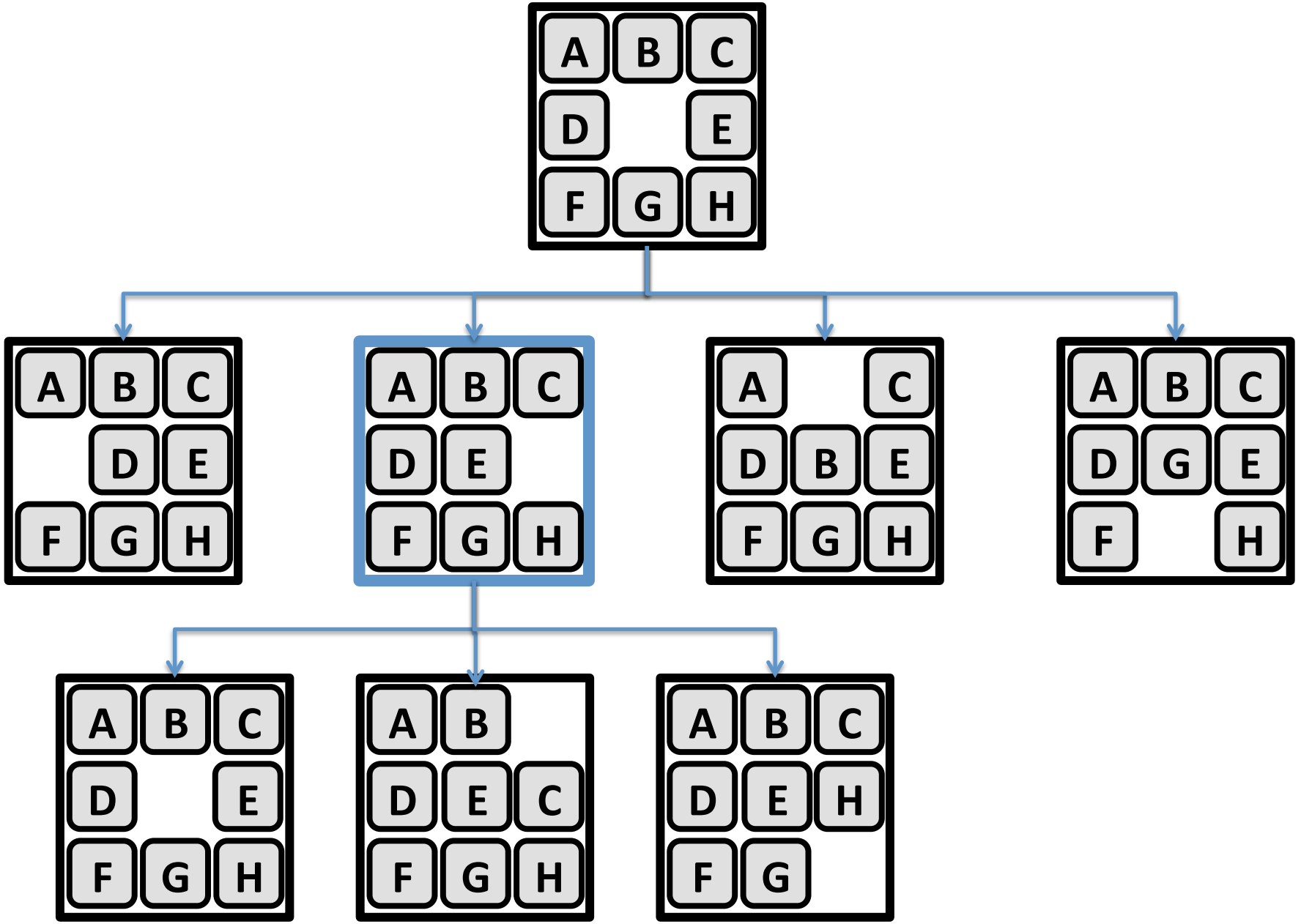
How to Design Algorithms

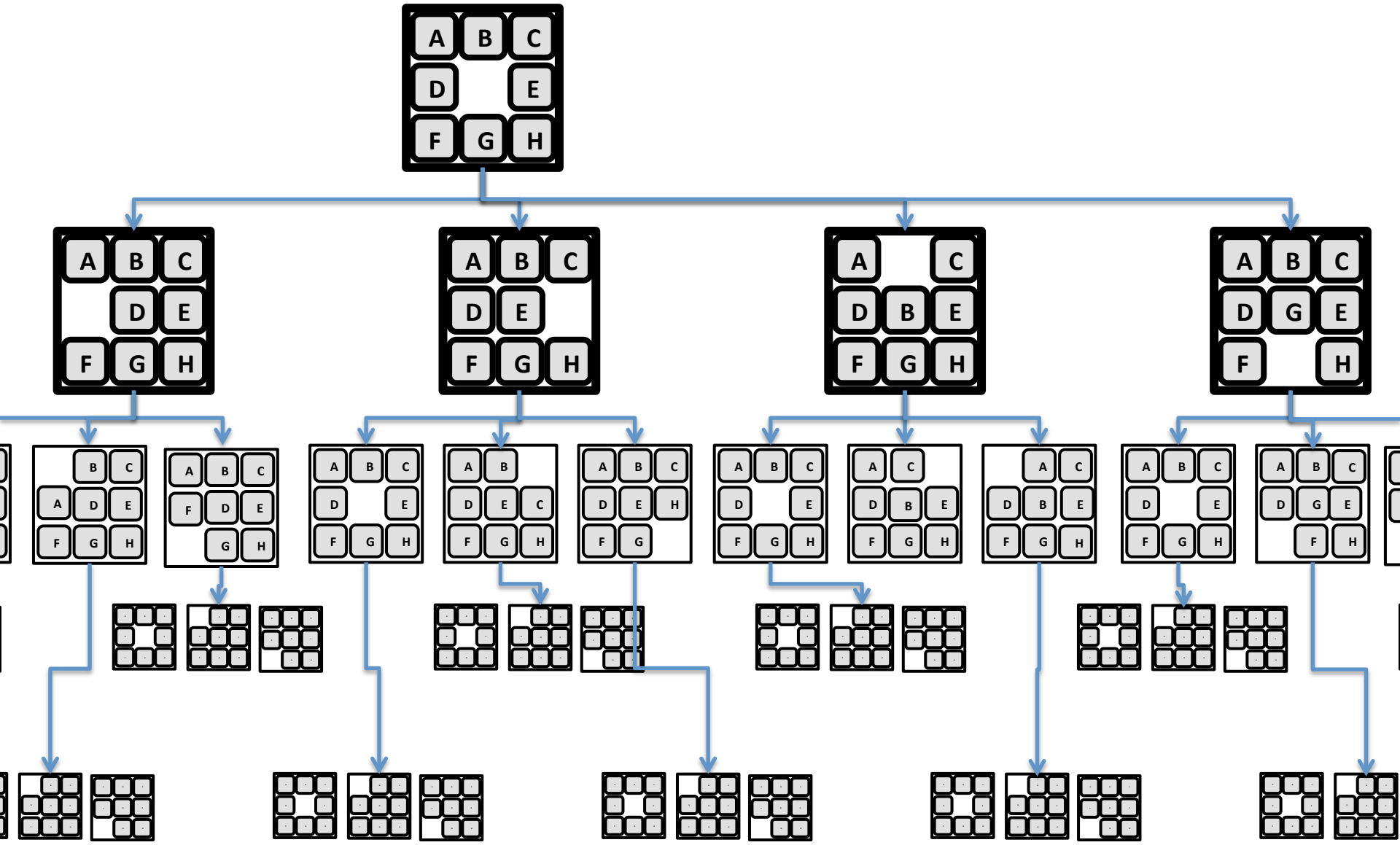


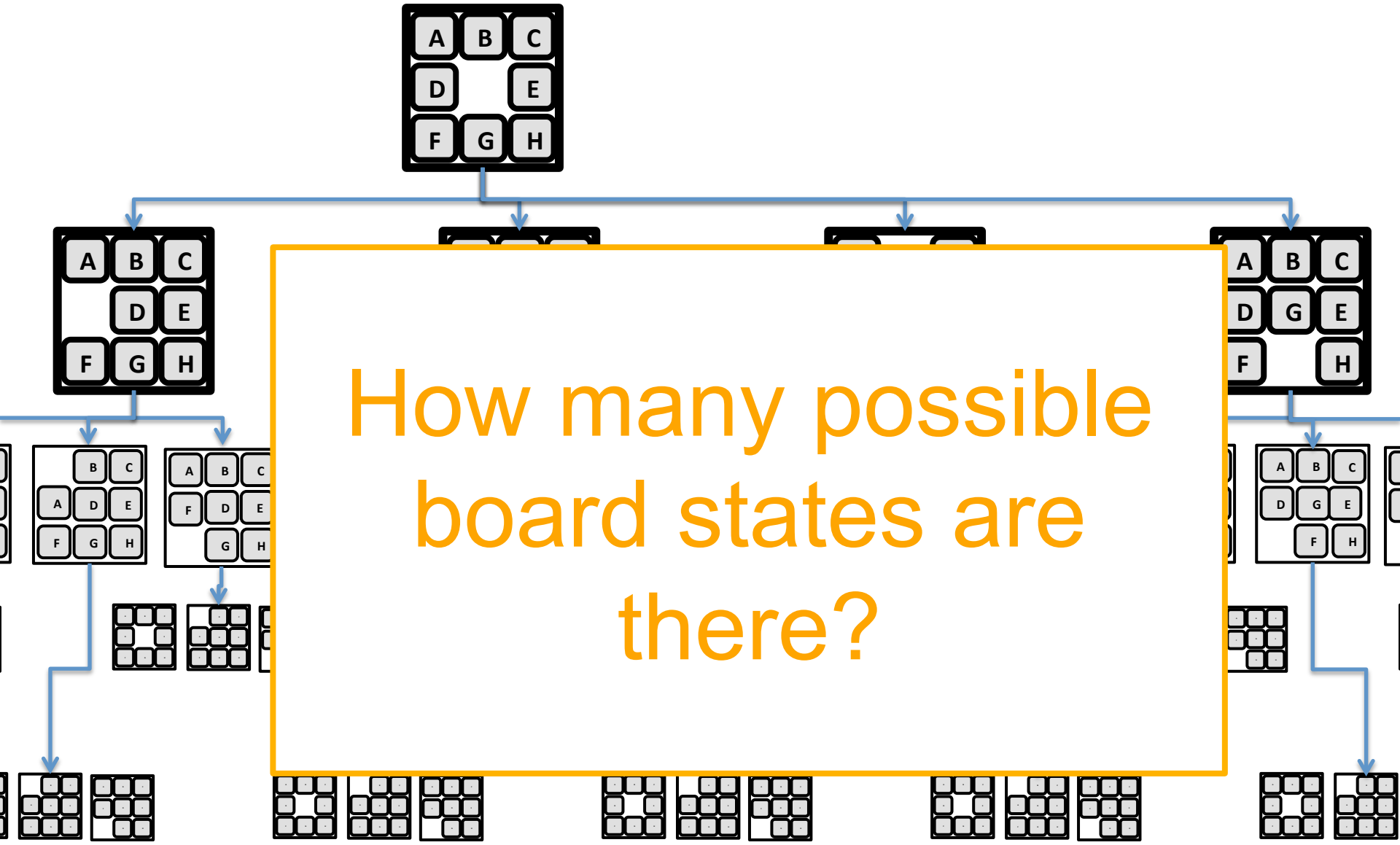
- How important is speed in my application? Must the problem be solved within one second? One minute? One hour? One day?
- How much time and effort can I invest in implementing my algorithm? Will I be limited to simple algorithms that can be coded up in a day, or do I have the freedom to experiment with a couple of approaches and see which is best?
- Am I trying to solve a numerical problem? A graph algorithm problem? A geometric problem? A string problem? A set problem? Might my problem be formulated in more than one way? Which formulation seems easiest?

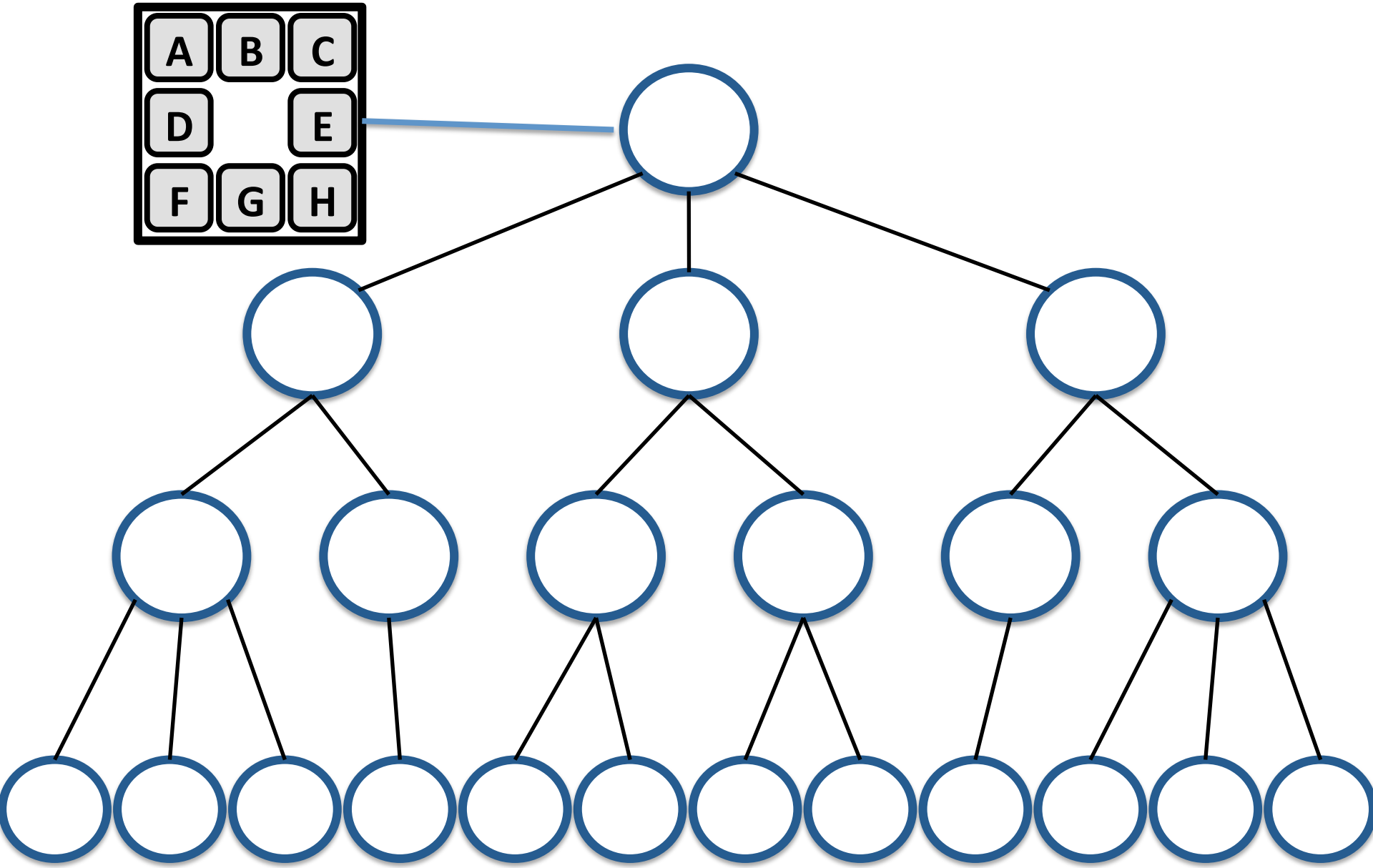

A
B
C
D
E
F
G
H

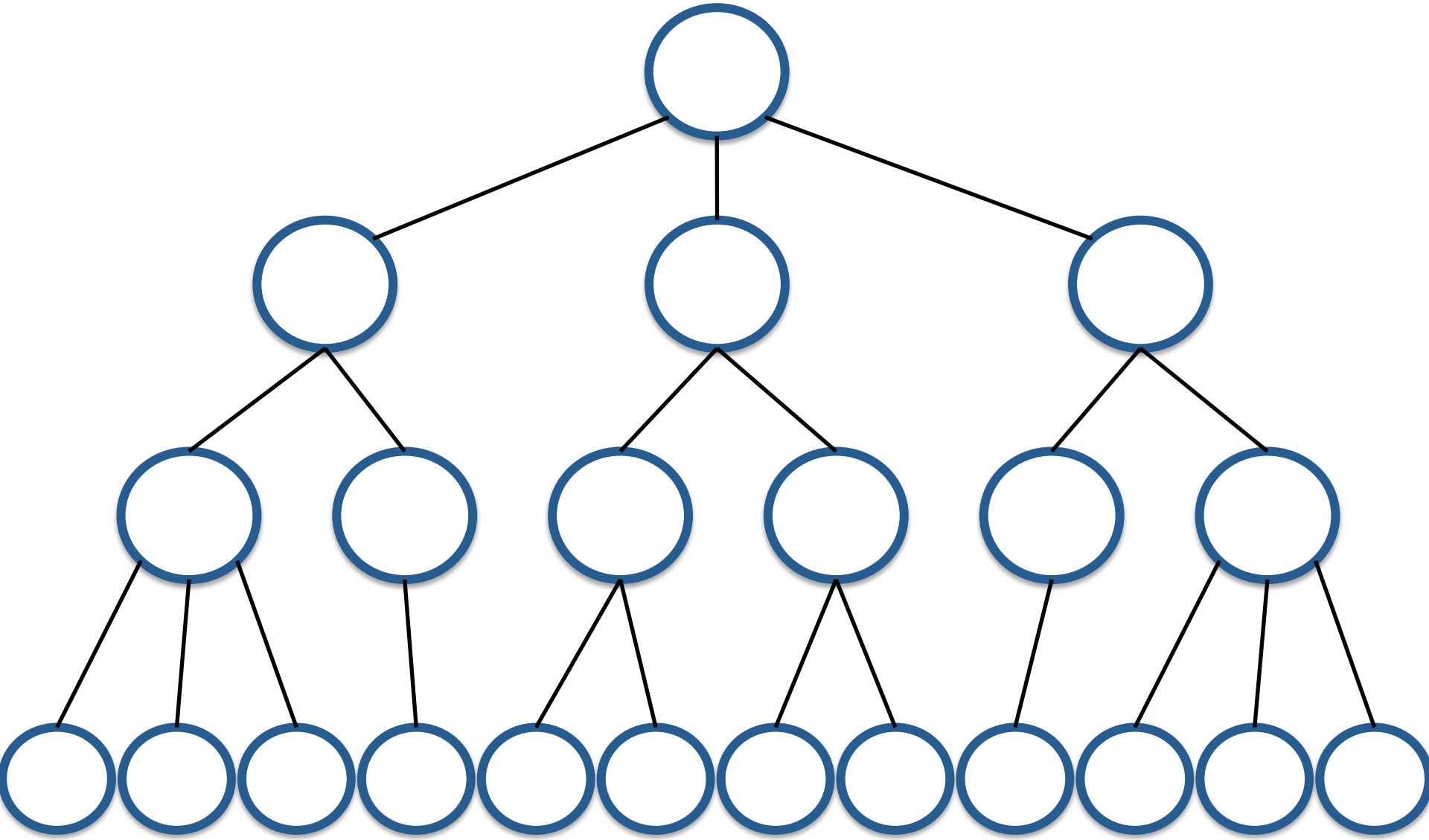

A B C
D E
F G H
A B C
D E
F G H
A B C
D E
F G H
A C
D B E
F G H
A B C
D G E
F H

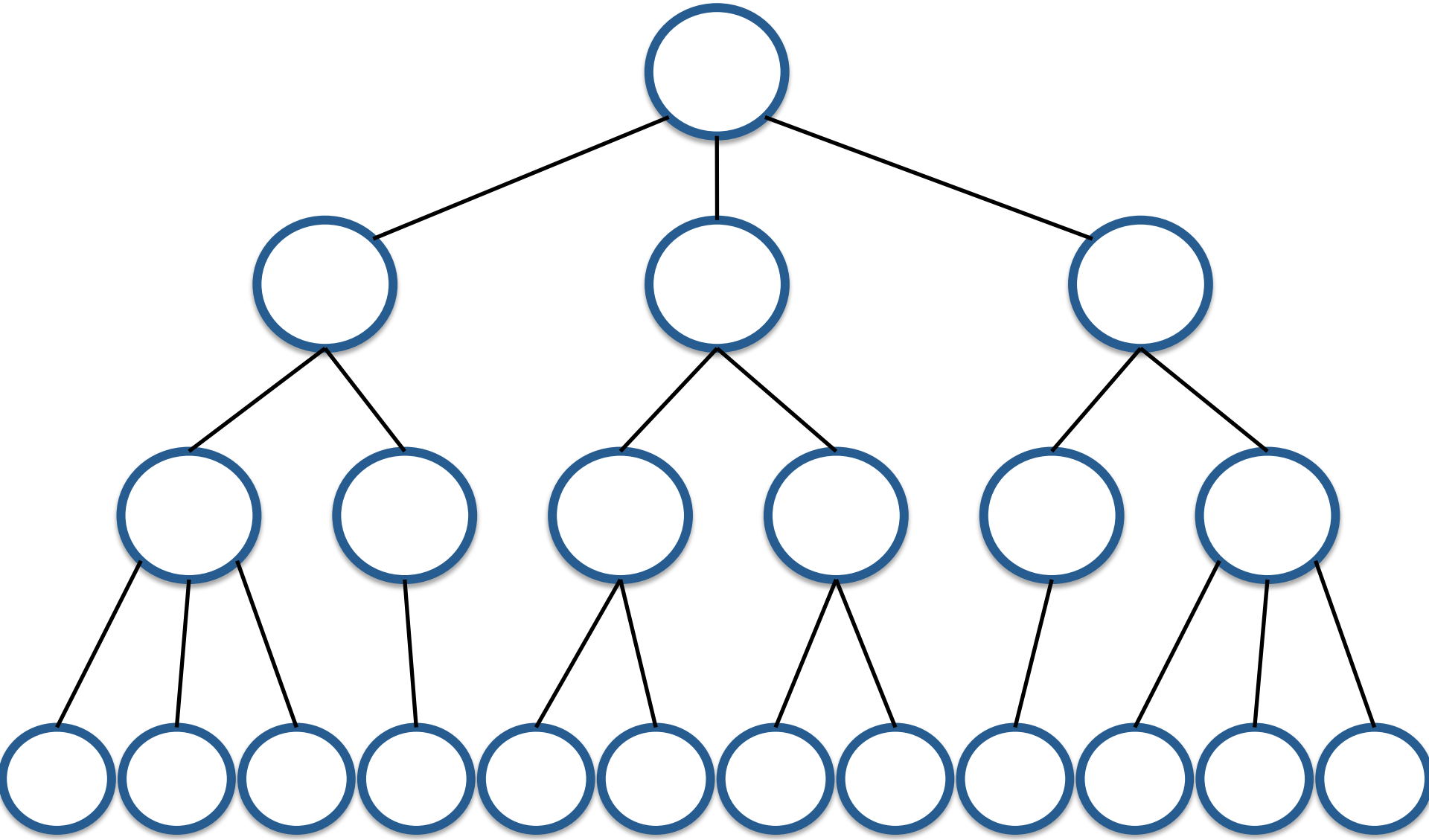

A B C
D E
F G H
A B C
D E
F G H
A B C
D E
F G H
A C
D B E
F G H
A B C
D G E
F H
A B C
D E
F G H
A B
D E C
F G H
A B C
D E H
F G


A B C
D E
F G H

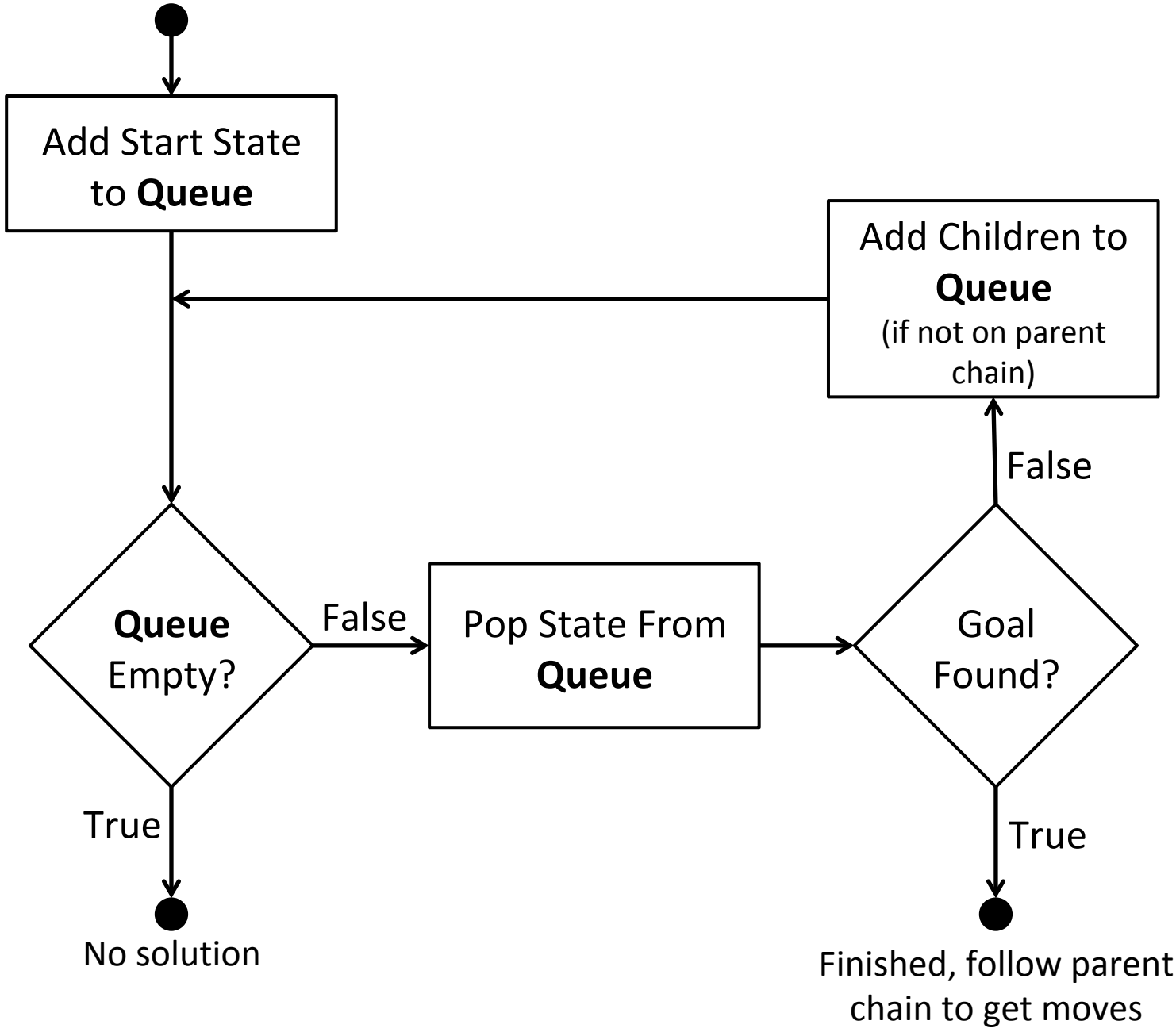

How many possible board states are there?


A
B
C
D
E
F
G
H

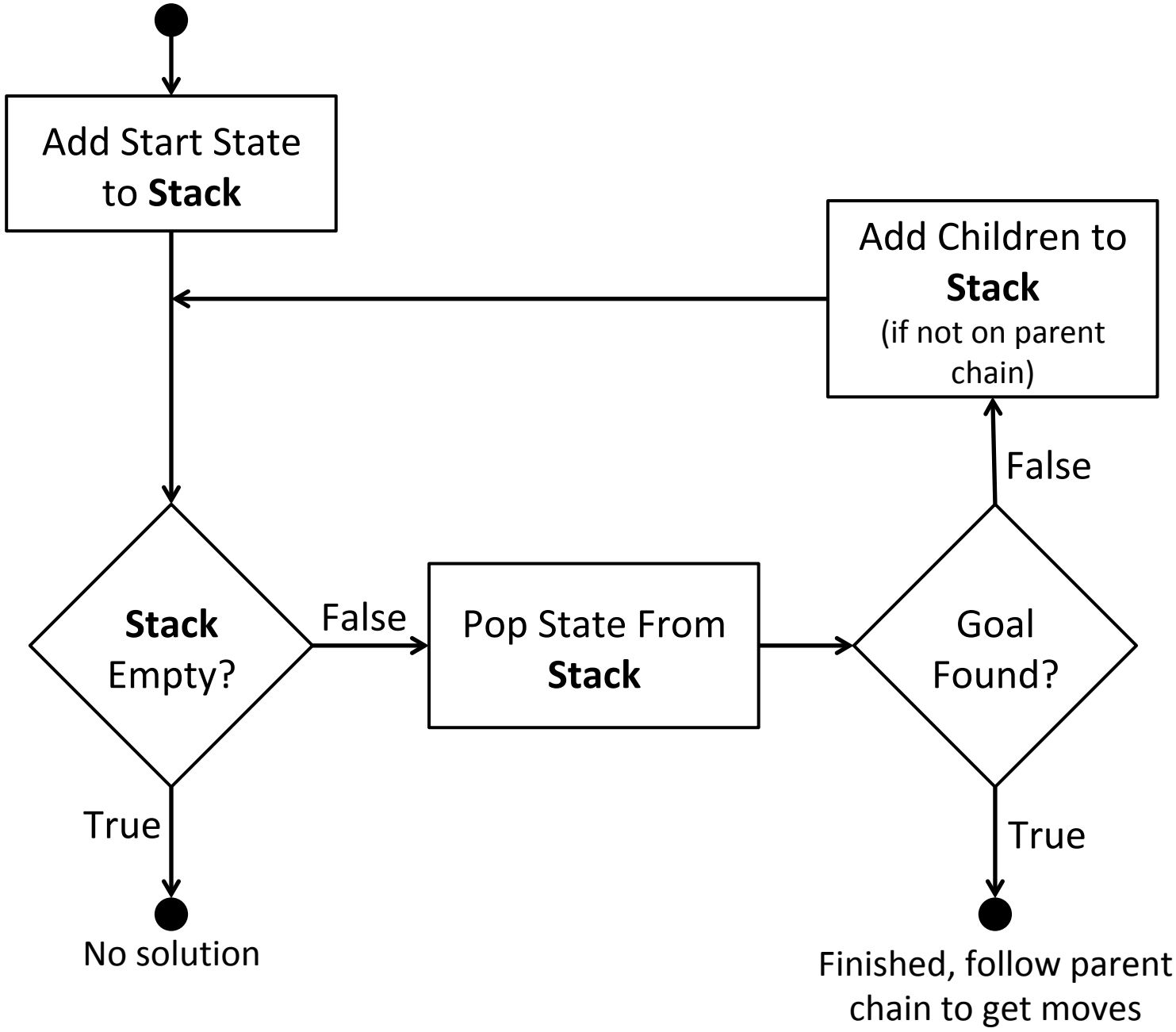




Breadth First Search

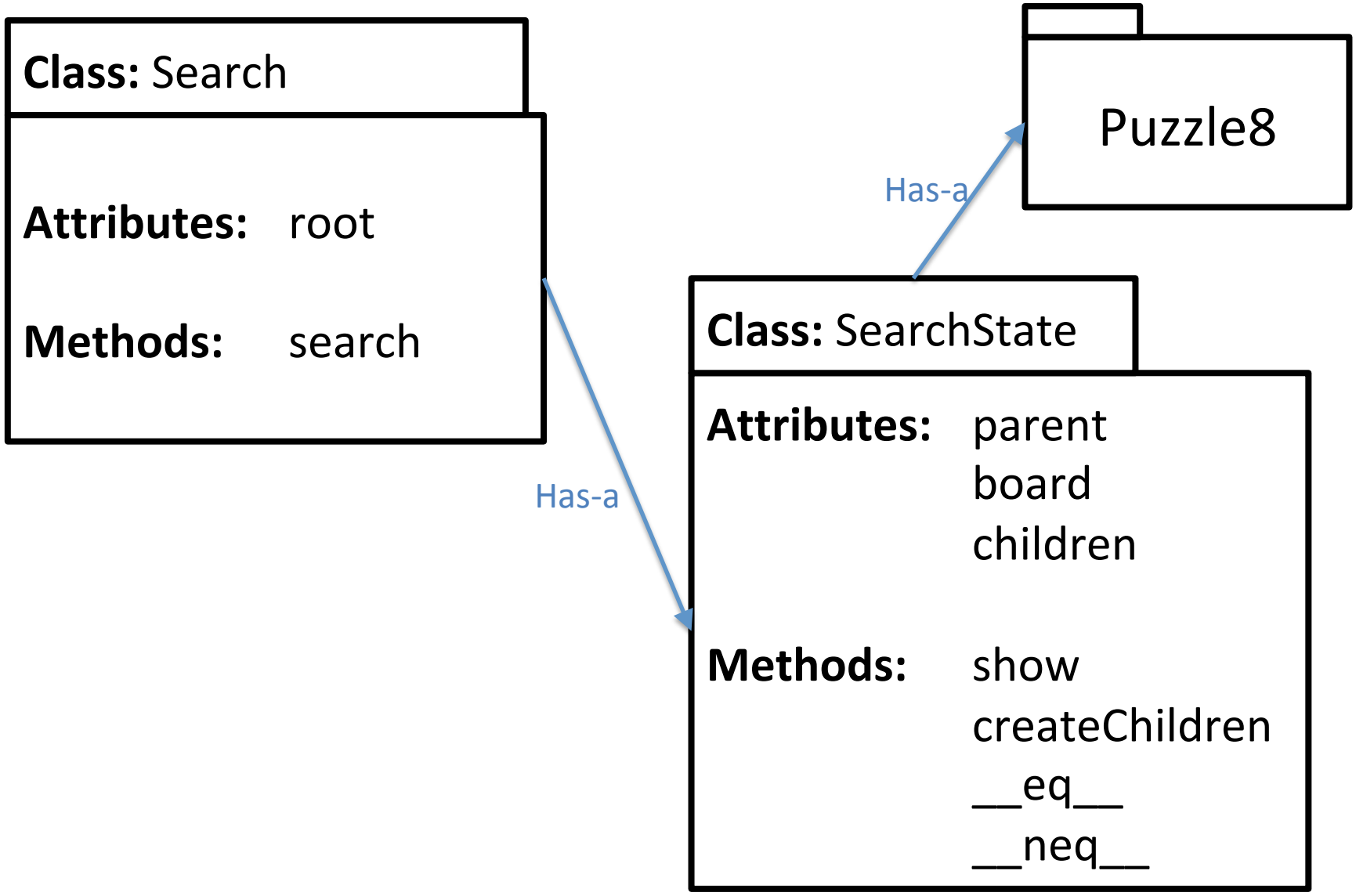


Depth First Search

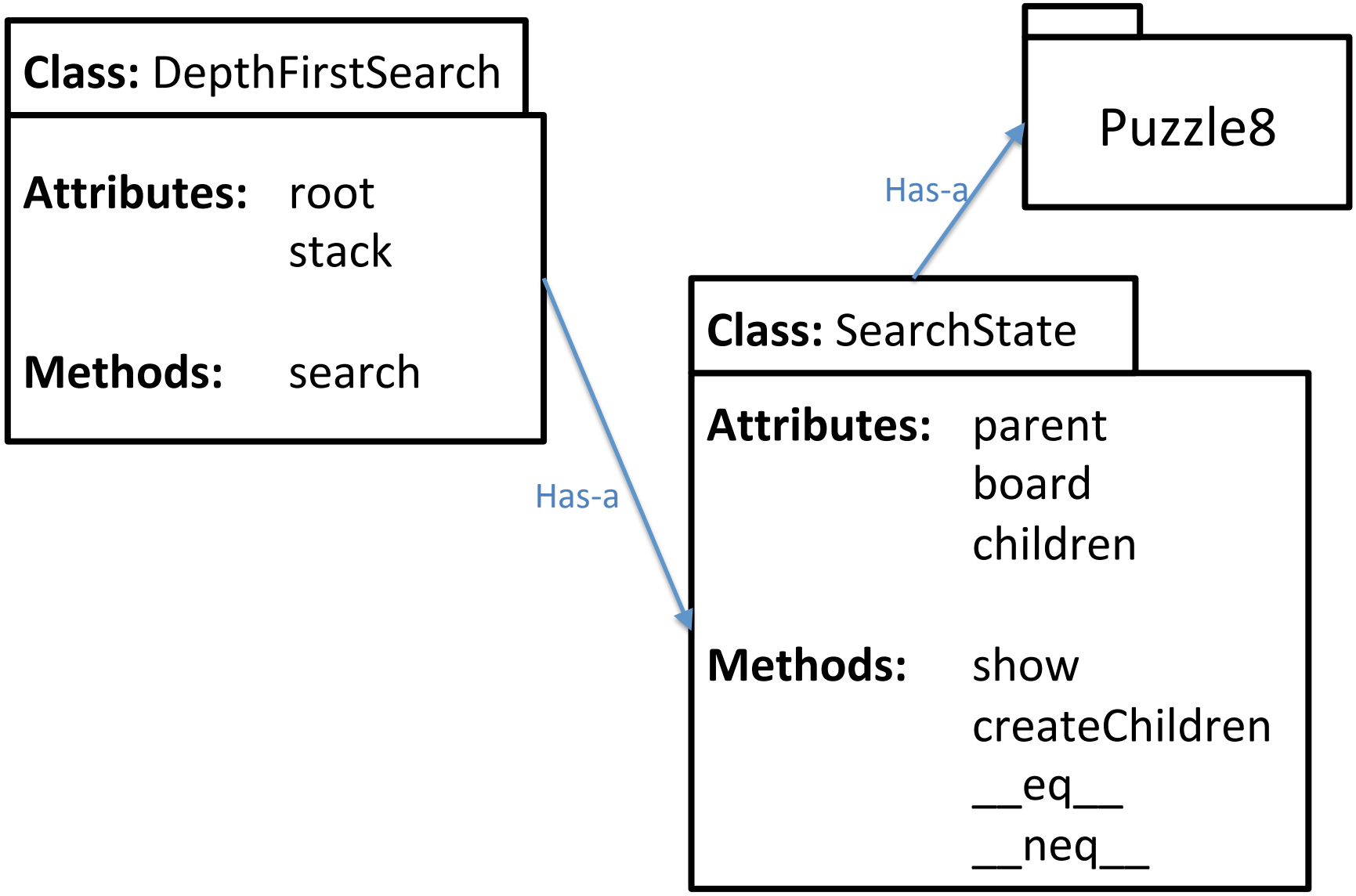


IMPLEMENTING A SEARCH ALGORITHM

Search Objects



Search Objects



Interesting Code

```
import copy

class SearchState:

    def __init__(self, board, parent = None):
        self.parent = parent
        self.board = copy.deepcopy(board)
        self.children = list()

    # Overload == and != opertors so we can easily
    # search lists for boards
    def __eq__(self, other):
        return self.board == other.board
    def __ne__(self, other):
        return self.board != other.board
```

Interesting Code

```
import copy

class SearchState:

    def __init__(self, board, parent = None)
        self.parent = parent
        self.board = copy.deepcopy(board)
        self.children = list()

    # Overload == and != opertors so we c
    # search lists for boards
    def __eq__(self, other):
        return self.board == other.board
    def __ne__(self, other):
        return self.board != other.board
```

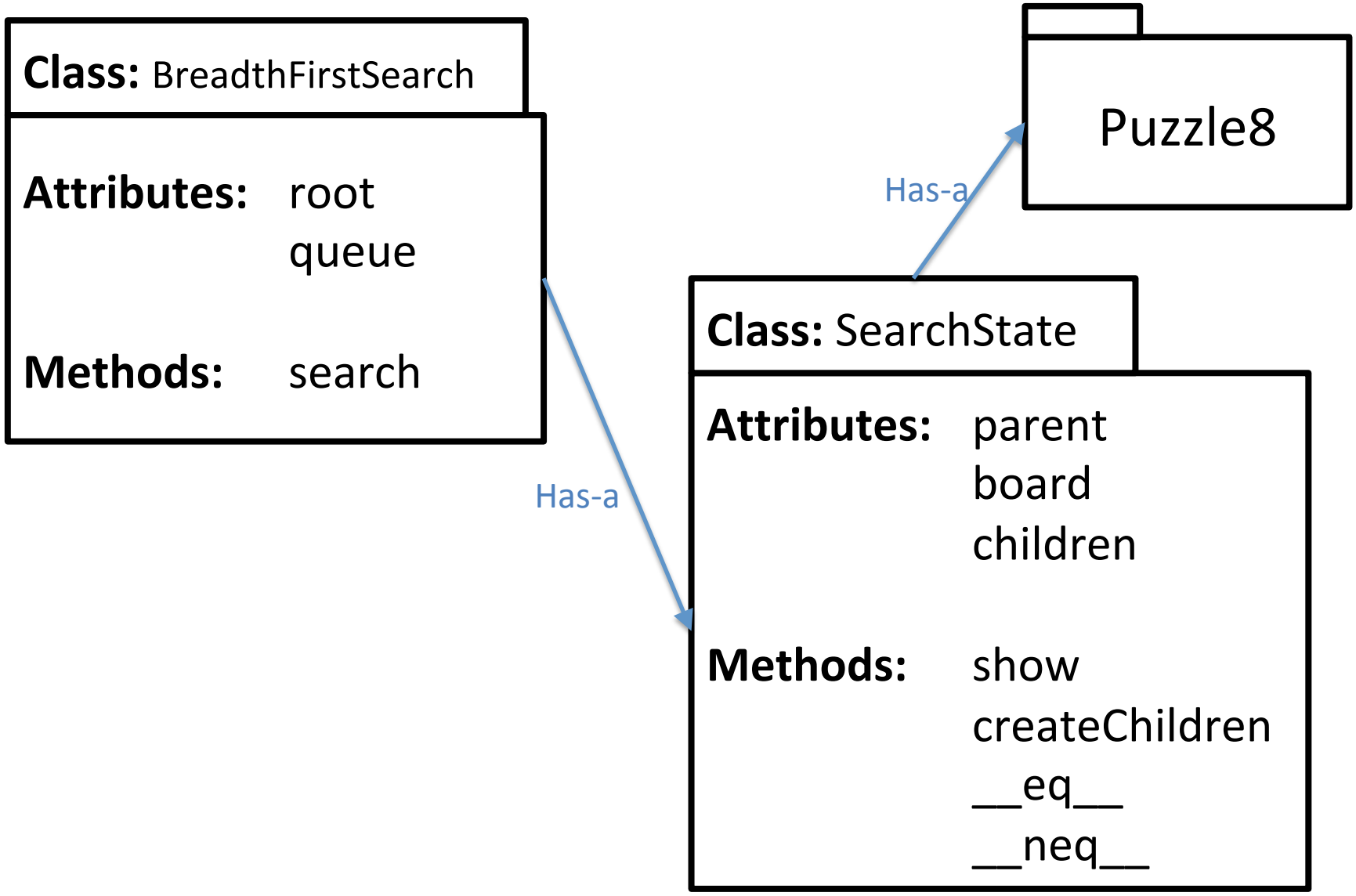
This creates a deep copy of a Python object rather than just copying a reference

Interesting Code

```
import copy

class SearchState:

    def __init__(self, board, parent = None):
        self.parent = parent
        self.board = copy.deepcopy(board)
        self.children = list()

    # Overload == and != opertors so we c
    # search lists for boards
    def __eq__(self, other):
        return self.board == other.board
    def __ne__(self, other):
        return self.board != other.board
```

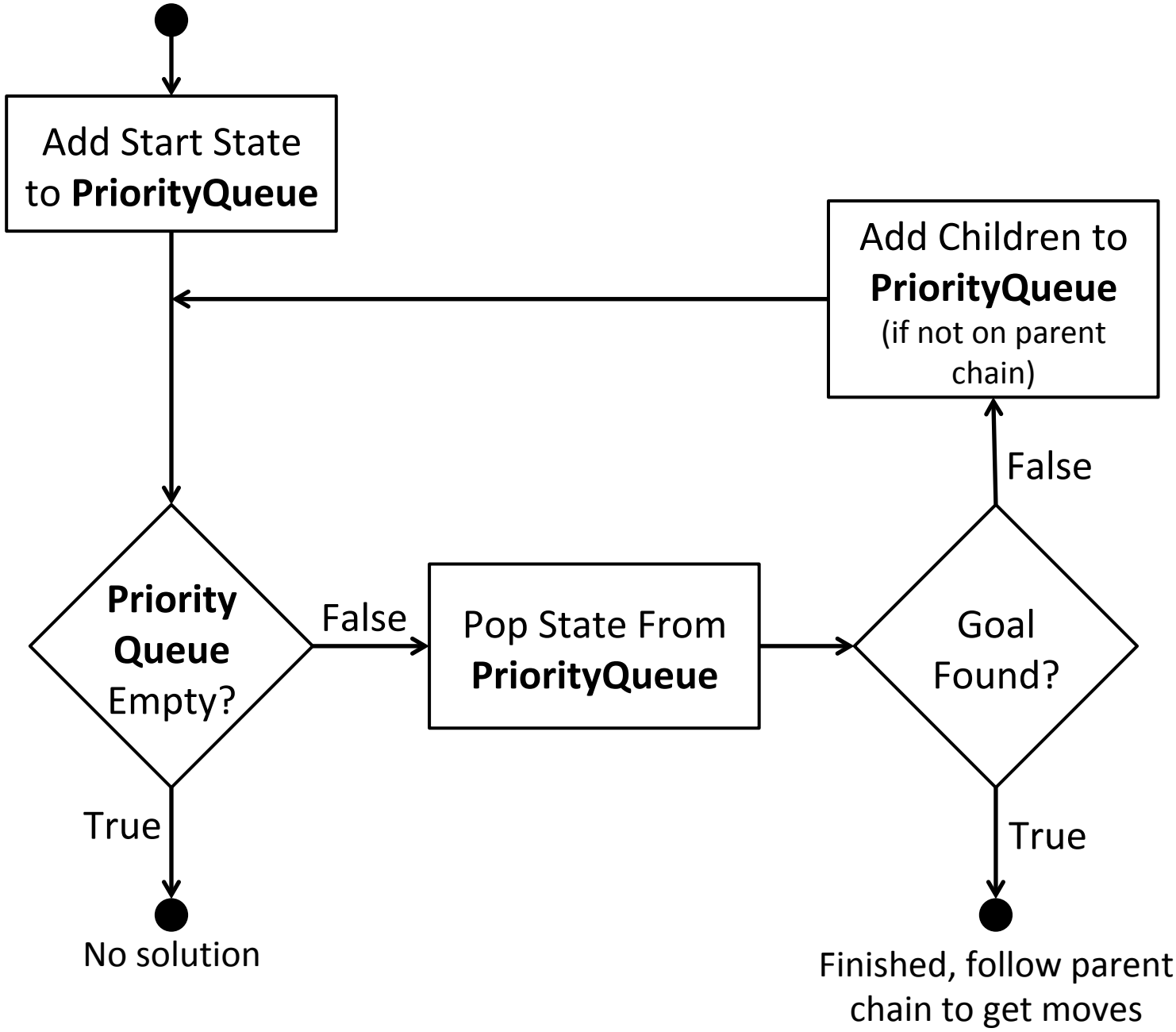
Overload == and != operators Fancy Python code!

Search Objects



CAN WE DO BETTER?

Best First Search



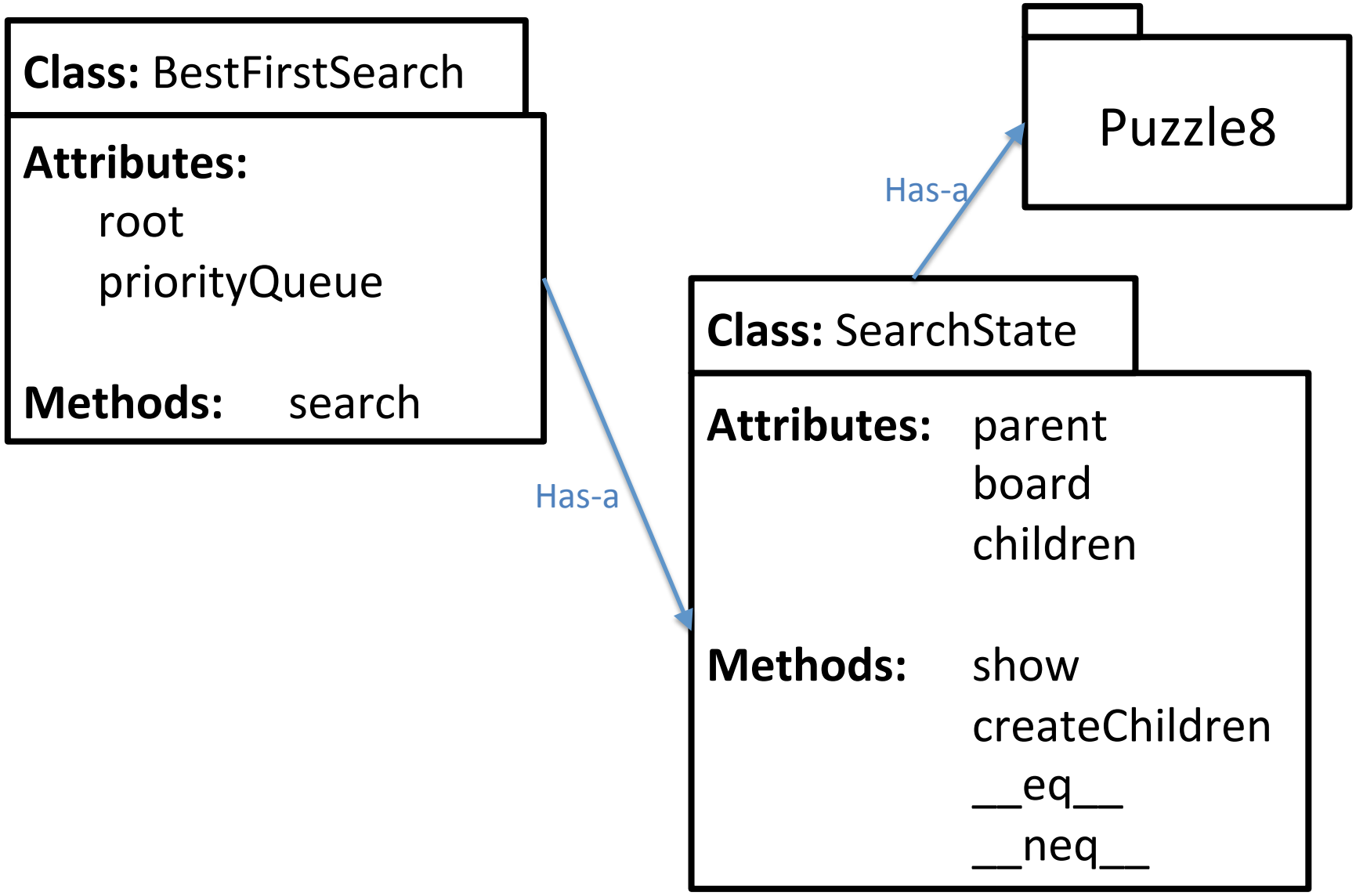
Heurisitics

To use Best First Search we need a **heuristic** that measures how good a board state is

For the 8s puzzle there are some simple ones:

- The number of tokens that are out of place.
- The sum of the distances between each token and its goal position

Search Objects



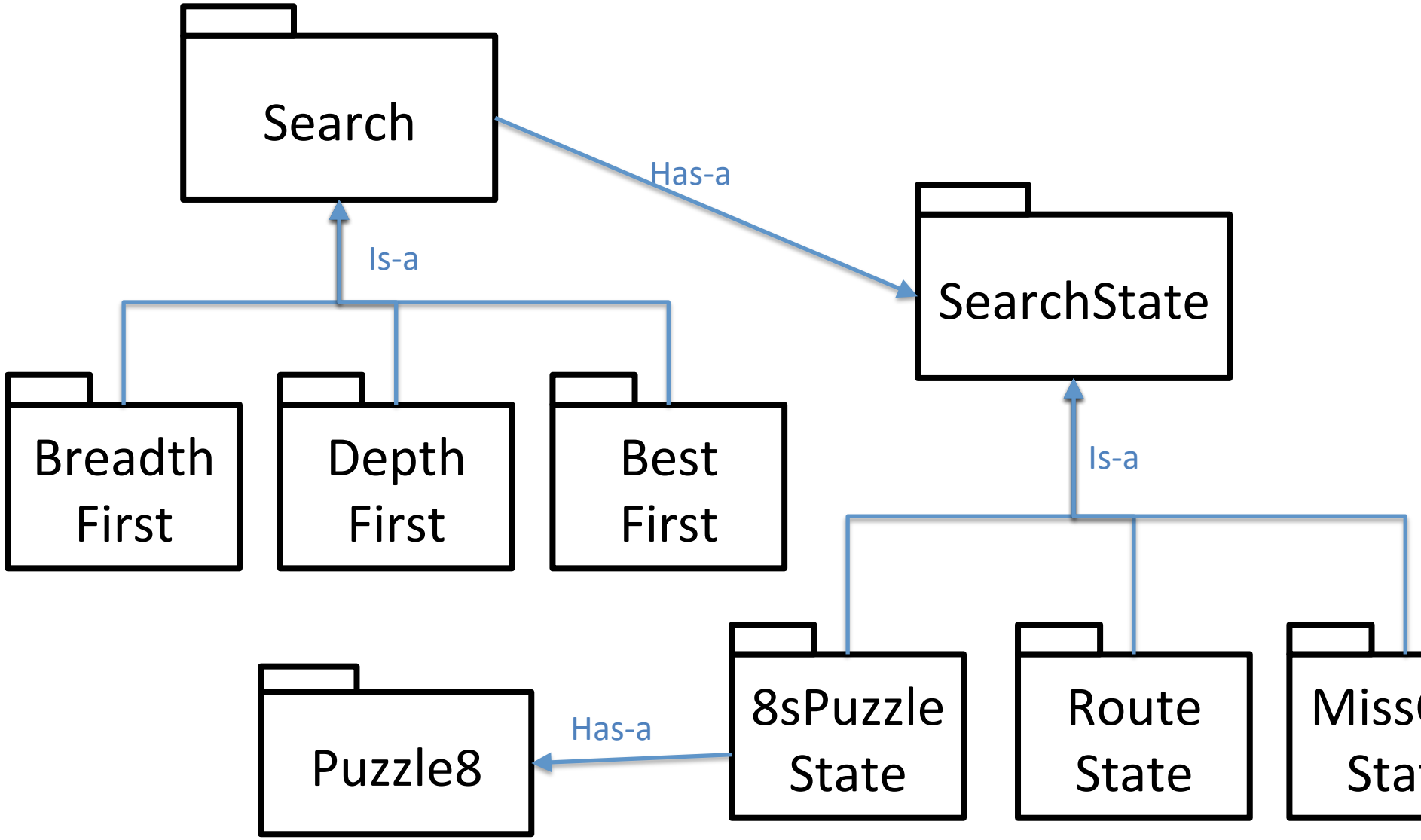
OTHER PROBLEMS

Other Problems

the same search code can be used to solve all sorts of other problems:

- Other puzzles and games (e.g. missionaries and cannibals problem)
- Route finding
- Optimization

Search Objects



SUMMARY

Summary

In these lectures we have looked at putting algorithm design and object oriented programming to build a sophisticated system