AGENTS AND ENVIRONMENTS

What is AI in reality?

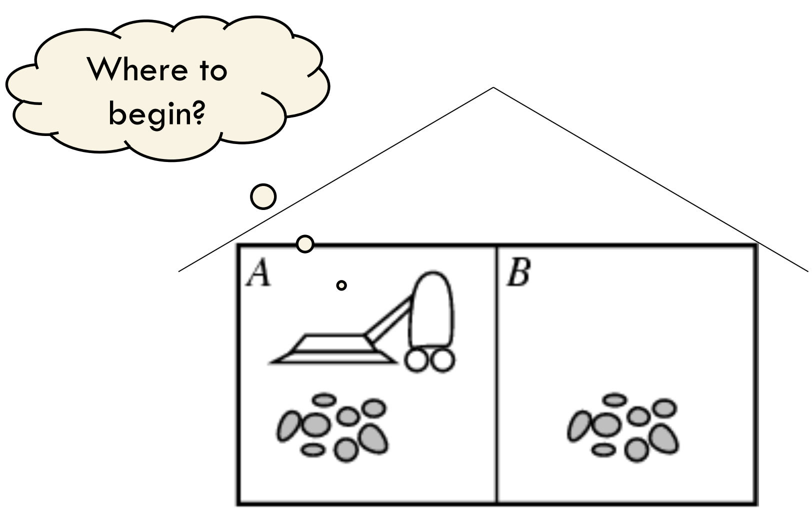
- "AI is our attempt to create a 'machine' that thinks (or acts) humanly (or rationally)"

Think like a human Cognitive Modeling	**Think rationally** Logic-based Systems
Act like a human Turing Test	**Act rationally** Rational Agents

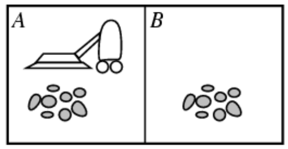
Today

- Reading
  - Artificial Intelligence: A modern approach (AIMA) Section 2.1-2.3, 3.1
- Goals
  - Rational agents
  - Task environment
  - Uninformed search

How do we create an intelligent vacuum?

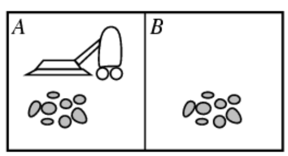


Agents



- An agent is any thing that perceives the world through sensors and acts on the world through actuators.

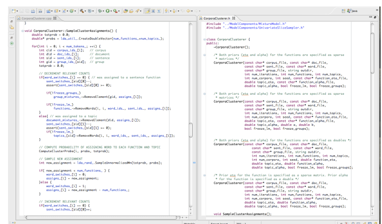
Agents



- An agent is any thing that perceives the world through sensors and acts on the world through actuators.
- percepts - which room, dirt in the room
- actions - Left, Right, Suck, Do Nothing

Agents

- An agent is any thing that perceives the world through sensors and acts on the world through actuators.



Agents

- An agent is any thing that perceives the world through sensors and acts on the world through actuators.

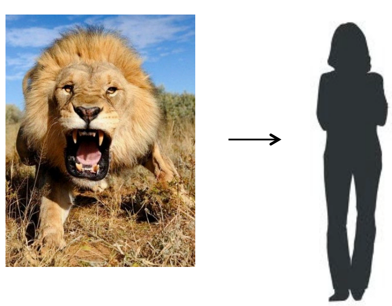
What is rationality?

Sing a song
Run
Smile
Run and scream

So what makes an agent rational?

Rational agents

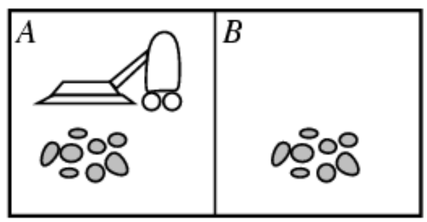
- For each percept sequence, a rational agent chooses an action that maximizes its performance measure given evidence from percept (sequence) and prior knowledge



~~**Sing a song**~~
~~**Run**~~
~~**Smile**~~
Run and scream

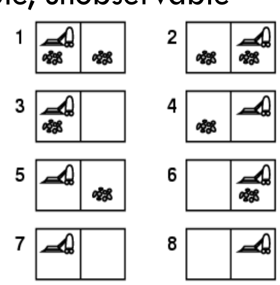
Rational agents

- For each percept sequence, a rational agent chooses an action that maximizes its performance measure given evidence from percept (sequence) and prior knowledge



Characterizing the task environment

- Now that we've defined a rational agent we want to specify the sort of environment in which that agent operates:
  - fully-observable, partially observable, unobservable
  - single agent vs. multi-agent
  - deterministic vs. stochastic
  - discrete vs. continuous



Solving problems by Searching

Search

- We have a rational agent. But how does the agent actually achieve its goal?
- Search for a solution, i.e. *a sequence of actions that leads from the initial state to the goal state*
- Uninformed search algorithms
  - Uses no information beyond problem
  - Assumes a discrete environment
  - Offline exploration

Step One: Formulate the search problem

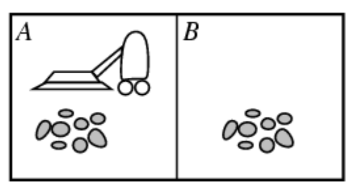
A well-defined search problem includes:

- states
- initial state
- actions
- successor function
- goal test
- path cost (reflects performance measure)

Induce the state space graph

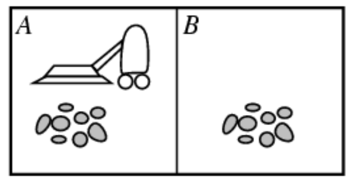
Example: Vacuum world

- states?
- initial state?
- actions?
- successor function?
- goal test?
- path cost?



Example: Vacuum world

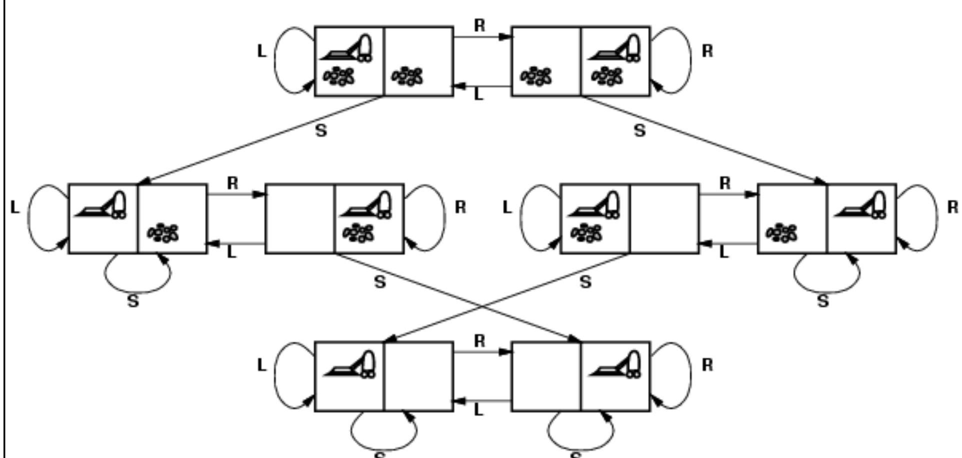
- states?
- initial state?
- actions?
- successor function?
- goal test?
- path cost?



There are 8 states: all possible configuration of dirt and position of the vacuum. There are five actions: Left, Right, Up, Down, NoOp. The successor function is just the resulting state after taking the action. The goal state is no dirt and the vacuum in either A or B. The path cost is 1 per action.

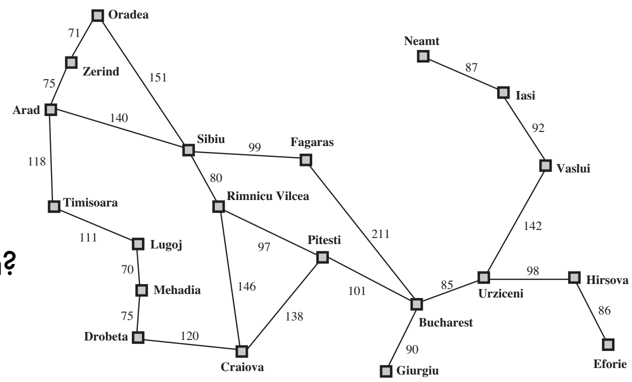
Example: Vacuum world

The state space graph:



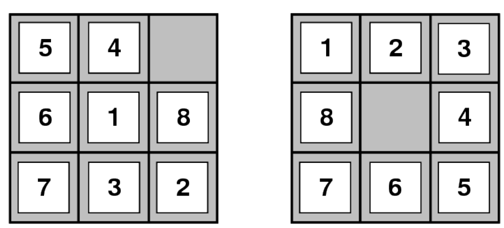
Example: Path to Bucharest

- states?
- initial state?
- actions?
- successor function?
- goal test?
- path cost?
- What does the state space graph look like?



Example: 8-puzzle

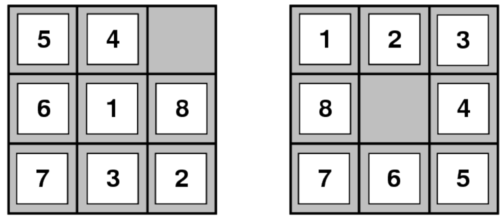
- states?
- initial state?
- actions?
- successor function?
- goal test?
- path cost?
- What does the state space look like?



Start State Goal State

Example: 8-puzzle

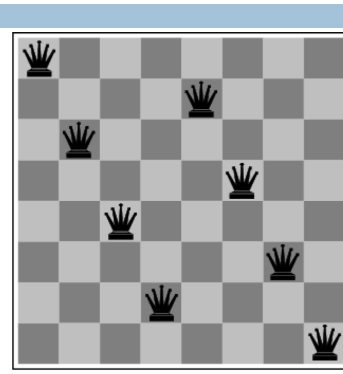
- states?
- initial state?
- actions?
- successor function?
- goal test?
- path cost?
- What does the state space look like?



Start State Goal State

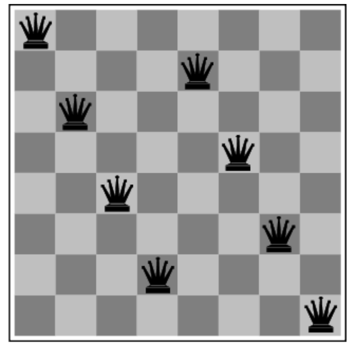
states – all possible configurations of the 8 tiles and the blank space
actions – move the blank space UP, DOWN, LEFT, RIGHT
path cost – a cost of 1 per action

Example: 8-queens puzzle



- states?
- initial state?
- actions?
- successor function?
- goal test?
- path cost?
- What does the state space look like?

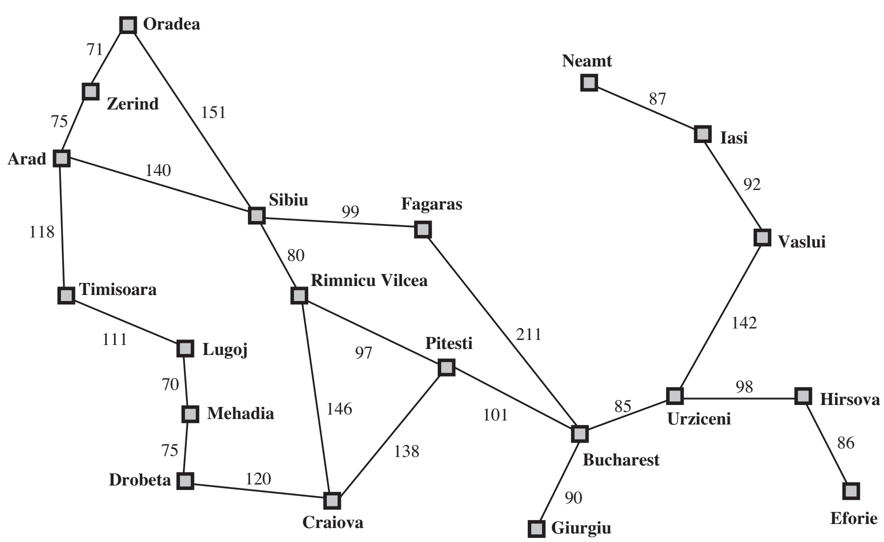
Example: 8-queens puzzle



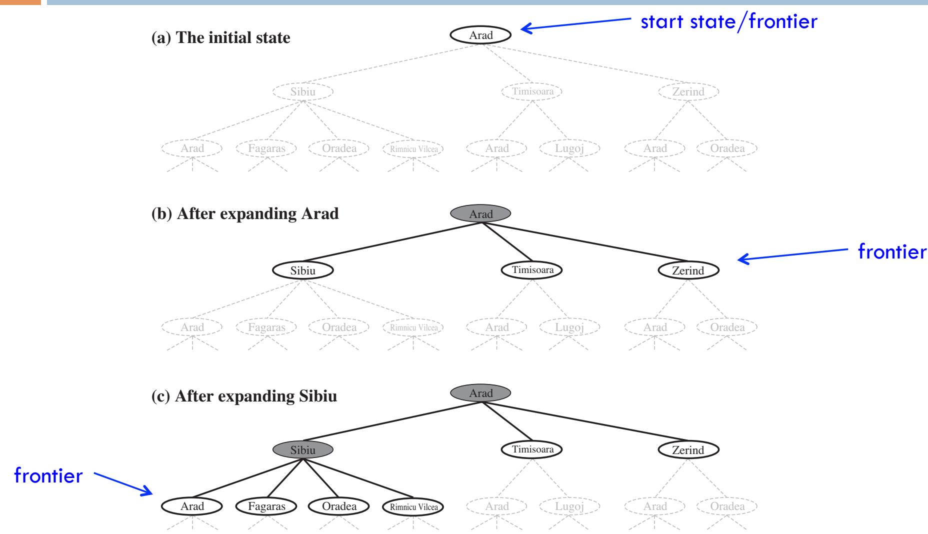
- states?
- initial state?
- actions?
- goal test?
- path cost?
- What does the state space look like?

incremental formulation: Initial state is a blank board. An action is to place a queen in the leftmost empty column (such that it is not in conflict with any previously placed queens)
Complete-state formulation: Initial state is 8 queens on the board. An action is to move a queen.
Note the path cost is irrelevant. We care only about the final configuration.

Example: path to Bucharest



Step Two: search

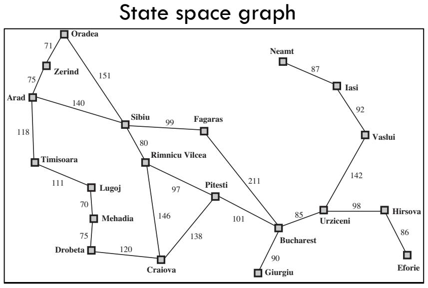


Tree-search algorithm

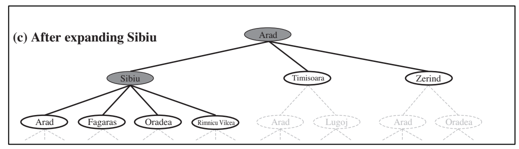
function TREE-SEARCH(*problem*, *strategy*) **returns** a solution or failure
  initialize the frontier using the initial state of *problem*
  loop do
    if the frontier is empty **return** failure
    choose node according to *strategy* and remove from frontier
    if node contains goal state **return** solution
    expand chosen node and add resulting nodes to frontier

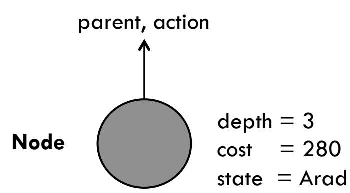
State space graph vs. Search tree

State space graph



Search tree





States versus nodes

- A state is a symbolic representation
- A node is a data structure
- Multiple nodes can point to the same state
- Keep an explored list which stores already-visited nodes

Graph-search

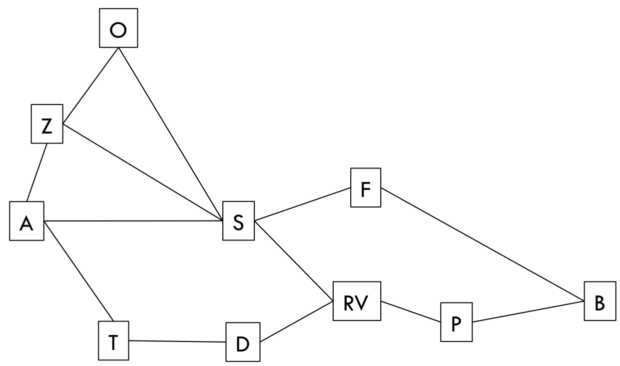
```
function GRAPH-SEARCH(problem, strategy) returns a solution or failure
    initialize the frontier using the initial state of problem
    initialize explored set to empty
    loop do
        if the frontier is empty return failure
        choose leaf node according to strategy and remove from frontier
        if node contains goal state return solution
        add node to explored set
        expand chosen node and add resulting nodes to frontier
        only if not in frontier or explored set
```

Search Strategies

A search strategy *specifies the order in which nodes are selected from the frontier* to be expanded

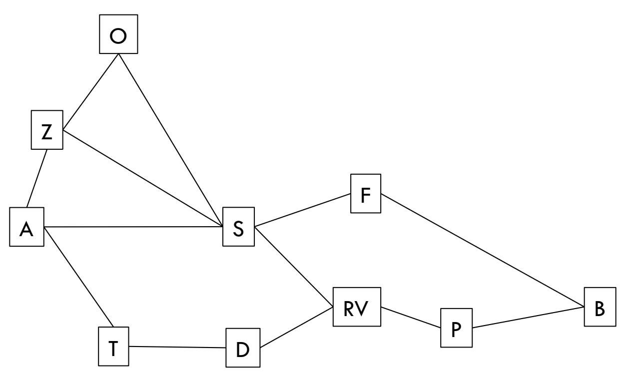
Breadth-first search (BFS)

- Expand shallowest unexpanded node
- Implementation: *frontier* is a FIFO queue



Depth-first search (DFS)

- Expand deepest unexpanded node
- Implementation: *frontier* is a LIFO queue (stack)

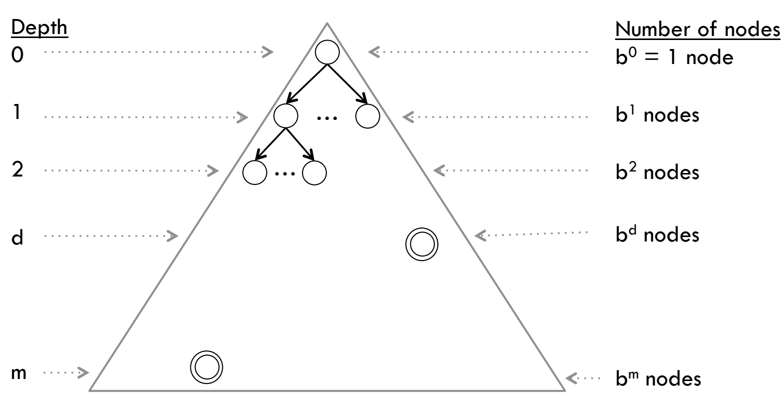


Evaluating search algorithm

- Time (Big-O)
  - approximately the number of nodes generated (frontier plus explored list)
- Space (Big-O)
  - the max # of nodes stored in memory at any time
- Complete (yes/no)
  - If a solution exists, will we find it?
- Optimal (yes/no)
  - If we return a solution, will it be the best/optimal solution, i.e. solution with lowest path cost

Recap: analyzing search algorithms

- When analyzing time and space, it is useful to define some notation:
  - b – branching factor, i.e. max number of successors of any node
  - d – depth of the shallowest goal node
  - m – maximum length of any path in state space



Analyzing BFS

- Time: $O(b^d)$
- Space: $O(b^d)$
- Complete = YES if branching factor is finite
- Optimal = YES if path cost is non-decreasing function of depth of the node
- (Useful if step costs are constant)

Analyzing DFS

- Time (for Tree-Search): $O(b^m)$
- Space (for Tree-Search): $O(bm)$
- Complete = YES, if space is finite (and no circular paths), NO otherwise
- Optimal = NO

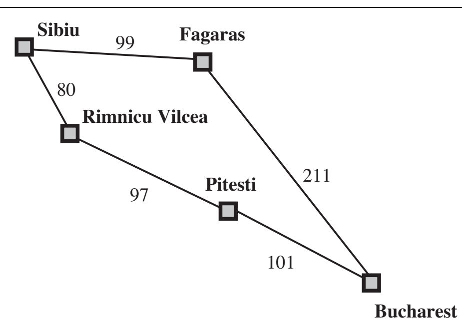
Time and memory requirements for BFS

Depth	Nodes	Time	Memory
2	1100	.11 sec	1 MB
4	111,100	11 sec	106 MB
6	10^7	19 min	10 GB
8	10^9	31 hours	1 terabyte
10	10^{11}	129 days	101 terabytes
12	10^{13}	35 years	10 petabytes
14	10^{15}	3,523 years	1 exabyte

BFS with b=10; 10,000 nodes/sec; 10 bytes/node

Uniform-cost search

- Expand node with lowest path cost
- Implementation:
  - *frontier* is a priority queue ordered by path cost



Analyzing Uniform-cost search

- Let C* be the cost of the optimal solution and ε be the minimum step cost
- Time: $O(b^{C^*/\varepsilon})$
- Space: $O(b^{C^*/\varepsilon})$
- Complete = YES if step cost exceeds epsilon
- Optimal = YES

BFS versus DFS

	Time	Space	Complete	Optimal
BFS	$O(b^d)$	$O(b^d)$	Yes	Yes
DFS	$O(b^m)$	O(bm)	Yes	No

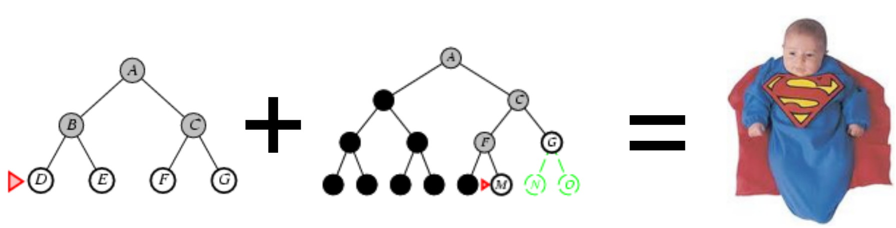
- Which strategy would you use and why?
- Brainstorm improvements to DFS and BFS

Problems with BFS and DFS

- BFS
  - memory! ☹
- DFS
  - Not optimal
  - And not even necessarily complete!

Improvements?

- Can we combined the optimality and completeness of BFS with the memory of DFS?



Depth limited DFS

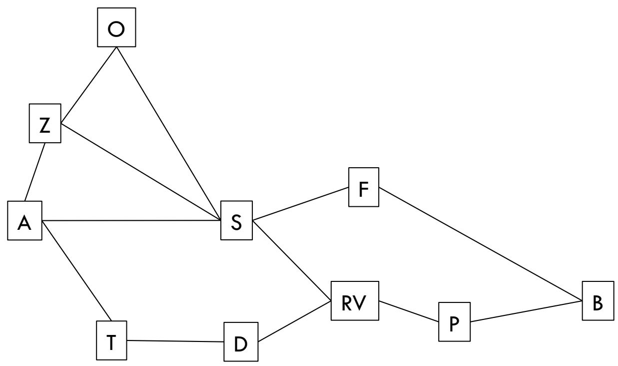
- DFS, but with a depth limit **L** specified
  - Nodes at depth **L** are treated as if they have no successors
  - We only search down to depth **L**
- Time?
  - $O(b^L)$
- Space?
  - O(bL)
- Complete?
  - No, if solution is longer than L
- Optimal
  - No, for same reasons DFS isn't

Iterative deepening search (IDS)

```
for depth=0, 1, 2, …
    run depth-limited DFS
    if solution found return result
```

- Blends the benefits of BFS and DFS
  - searches in a similar order to BFS
  - but has the memory requirements of DFS
- Will find the solution when **L** is the depth of the shallowest goal

Iterative deepening search (IDS)



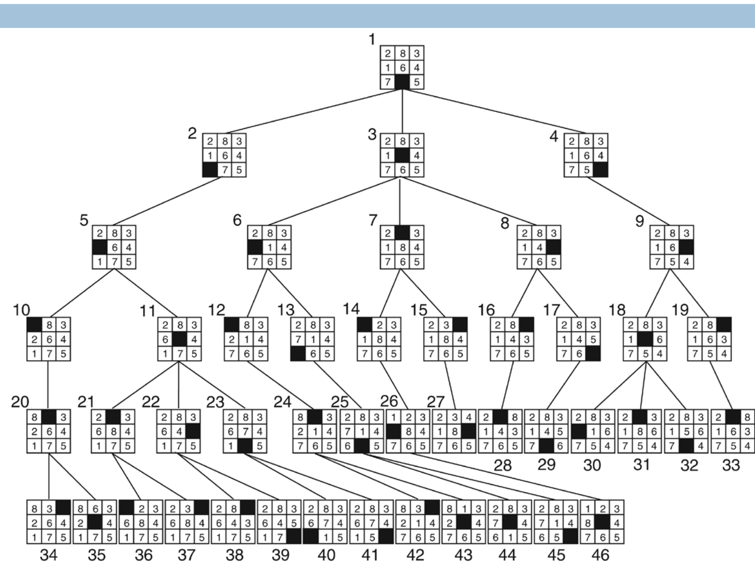
Time complexity for IDS

- L = 0: 1
- L = 1: 1 + b
- L = 2: $1 + b + b^2$
- L = 3: $1 + b + b^2 + b^3$
- …
- L = d: $1 + b + b^2 + b^3 + \ldots + b^d$
- Overall:
  - $d(1) + (d-1)b + (d-2)b^2 + (d-3)b^3 + \ldots + b^d$
  - $O(b^d)$
  - Cost of the repeat of the lower levels is subsumed by the cost at the highest level

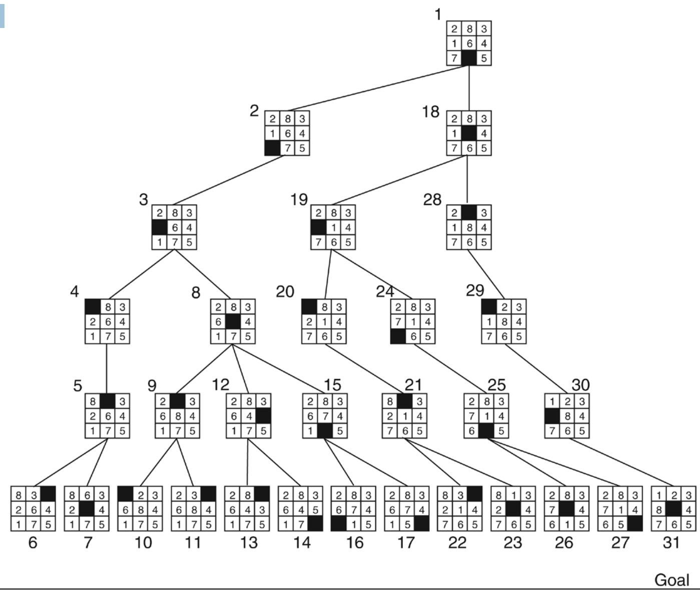
Analysis of IDS

- Space
  - O(bd)
- Complete?
  - Yes
- Optimal?
  - Yes

BFS of 8-puzzle



DFS of 8-puzzle



Summary of Uninformed Search

- Step One: Formulate the search problem
- Step Two: Search
  - Breadth-first search (queue)
  - Depth-first search (stack)
  - Uniform cost search (priority queue)
  - Iterative-deepening DFS (ID-DFS)
- Analyze search algorithms
  - Time, Space, Completeness, Optimality