AGENTS AND SEARCH

What is AI?

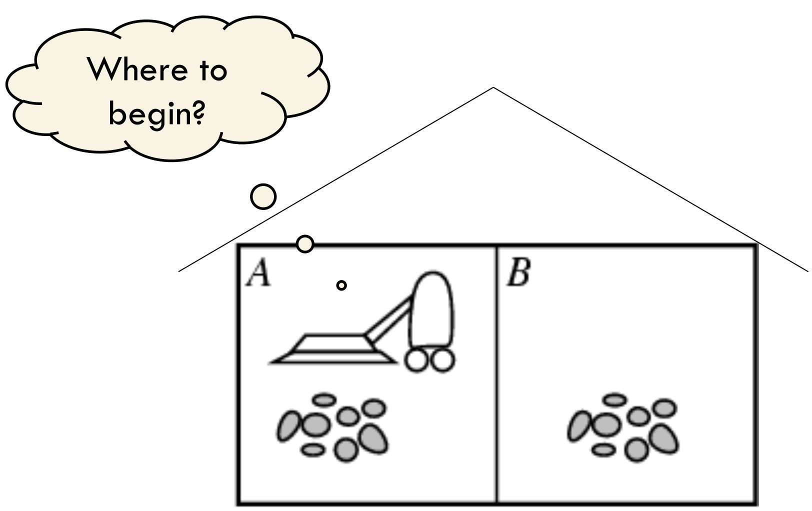
- "AI is our attempt to create a 'machine' that thinks (or acts) humanly (or rationally)"

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

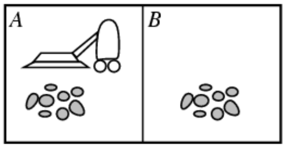
Today

- Reading
  - Skim AIMA Section 2.1-2.3
  - Read AIMA Section 3.1 – 3.4 (can skim 3.2)
- Objectives
  - What's a rational agent?
  - Uninformed search
    - Formulating the search problem
    - State-space search
    - Analyze complexity of search

How do we create an intelligent vacuum?

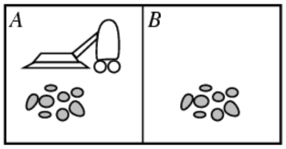


Agents



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

Agents

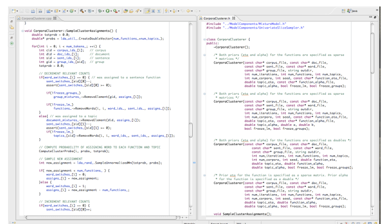


- An agent is any thing that perceives the world through sensors and acts on the world through actuators.

- percepts - which room, dirt in the room
- actions - Left, Right, Pick Up Dirt, Do Nothing

Agents

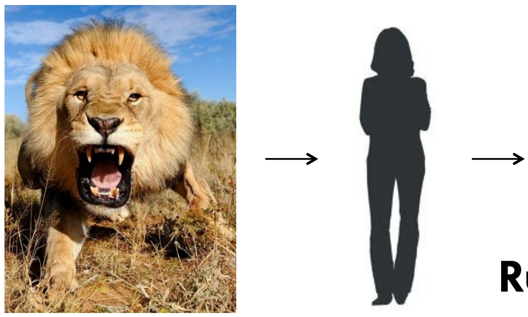
- An agent is any thing that perceives the world through sensors and acts on the world through actuators.



Agents

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

What is rationality?

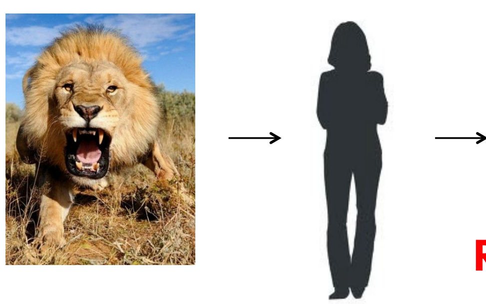


Sing a song?
Run?
Smile?
Run and scream?

So what makes an agent rational?

Rational agents

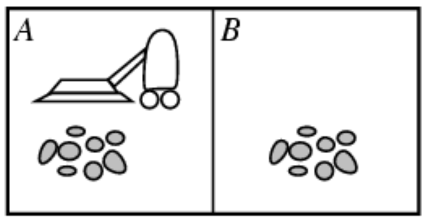
- For each percept sequence, a rational agent chooses an action that maximizes its performance measure given evidence from percept (sequence) and prior knowledge



~~**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



Framework for the rest of this course

Task	Action Sequence	Percepts/Prior knowledge	Performance Measure	What strategy should agent employ to maximize p.m.?
Find route to location	Set of street directions	Current location, traffic, map of area	Positive if route ends at desired location	Search
Predict patient has breast cancer	Treatment plan	mammogram, patient information	Positive if treatment plan matches patient condition	Bayesian network
Should I wait to eat at this restaurant?	Wait or don't wait?	Quality of restaurant, past behavior,…	Higher value if decision fits observed past behavior of humans	Decision tree

Solving problems by Searching

State-space search

- State-space search is one of the earliest techniques employed in AI (~1950s)
- Canonical examples
  - 1850s: The 8-queens puzzle
  - 1870s: The n-puzzle (similar to 2048 today)
  - 1960s: Missionaries and cannibals
- Real-life examples
  - Airline flights
  - VLSI Layout
  - Metabolic pathways

State-space 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

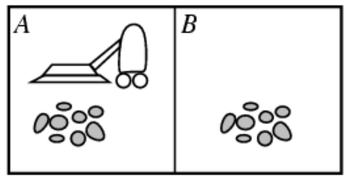
A well-defined search problem includes:

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

(states, initial state, actions, successor function:) Induce the state space graph

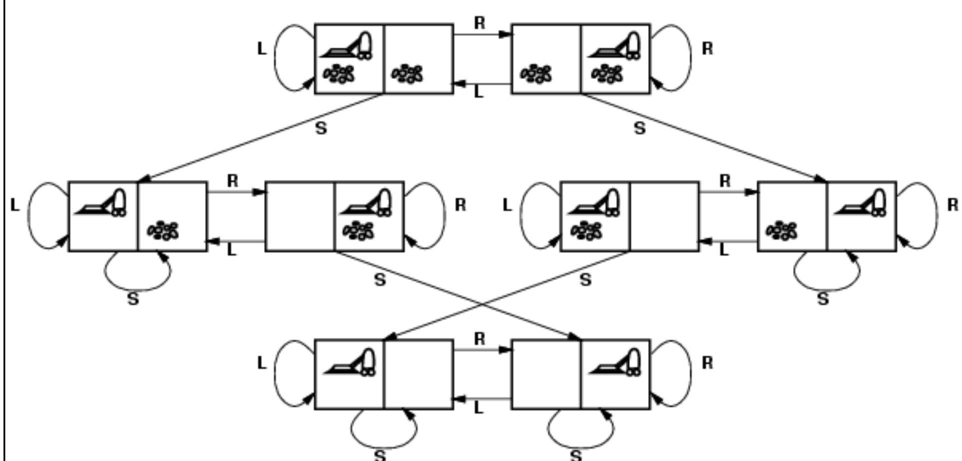
Step One: Vacuum world

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



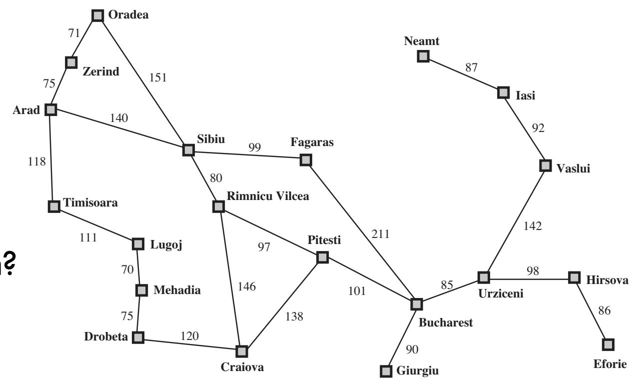
Step One: Vacuum world

The state space graph:



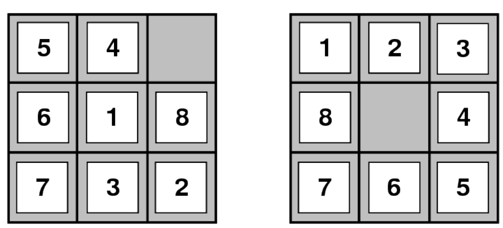
Step One: Path to Bucharest

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



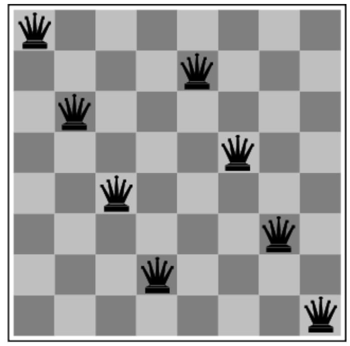
Step One: 8-puzzle

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



Start State Goal State

Step One: 8-queens puzzle

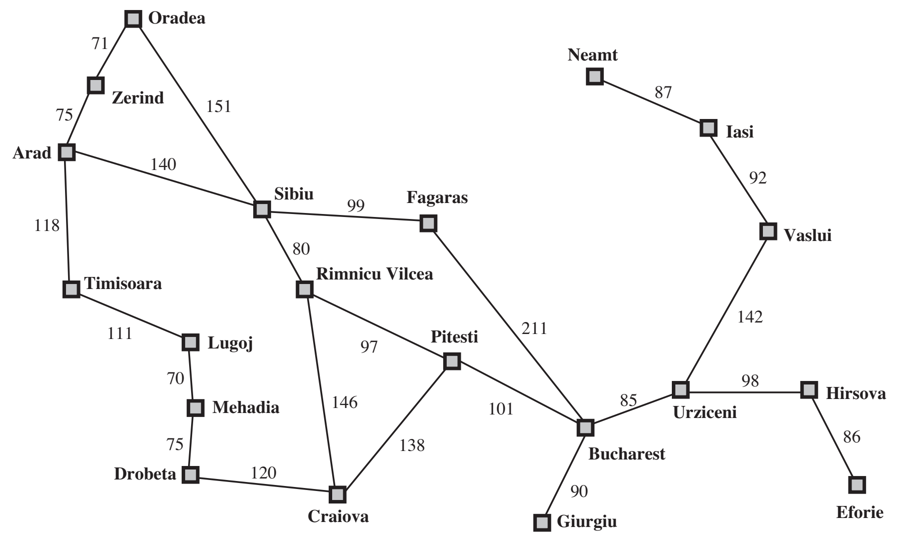


- states?
- initial state?
- actions?
- successor function?
- goal test?
- path cost?
- What does the state space look like?

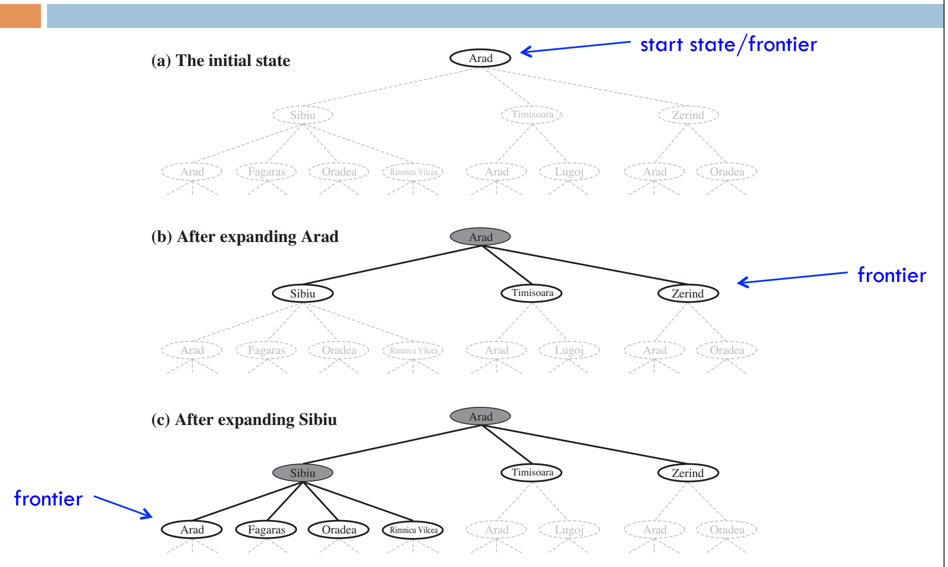
Step Two: Search

- Basic Idea
  - Pick a node
  - If not goal state
    - expand node by generating all its successors
    - mark node as explored
  - Repeat till goal found
- Necessary data structures
  - frontier - nodes that were generated but not yet expanded
  - (explored - nodes that have been expanded)

Step Two: Search



Step Two: Search

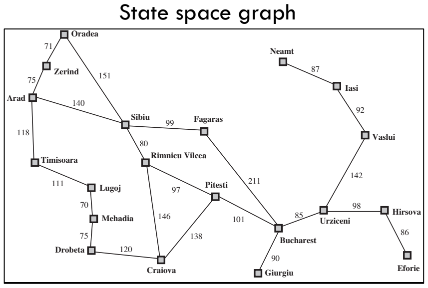


Tree-search algorithm

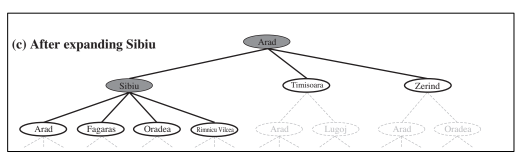
function TREE-SEARCH(*problem*, *strategy*) **returns** a solution or failure
  initialize the frontier using the initial state of *problem*
  loop do
    if the frontier is empty **return** failure
    choose node according to *strategy* and remove from frontier
    if node contains goal state **return**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

State space graph vs. Search tree

State space graph

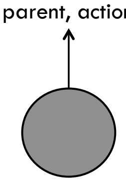


Search tree



parent, action

Node



depth = 3
cost = 280
state = Arad

Graph-search

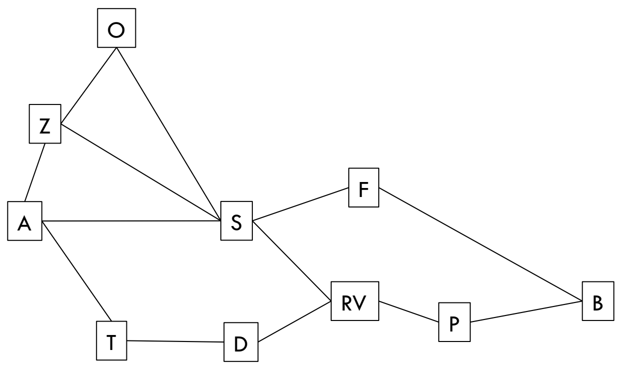
```
function GRAPH-SEARCH(problem, strategy) returns a solution or failure
    initialize the frontier using the initial state of problem
    initialize explored set to empty
    loop do
        if the frontier is empty return failure
        choose leaf node according to strategy and remove from frontier
        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

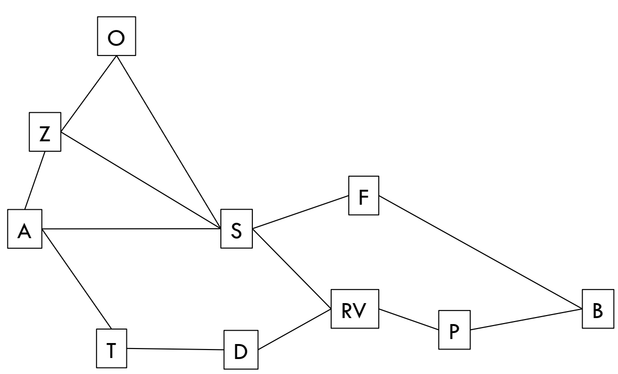
Breadth-first search (BFS)

- 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)

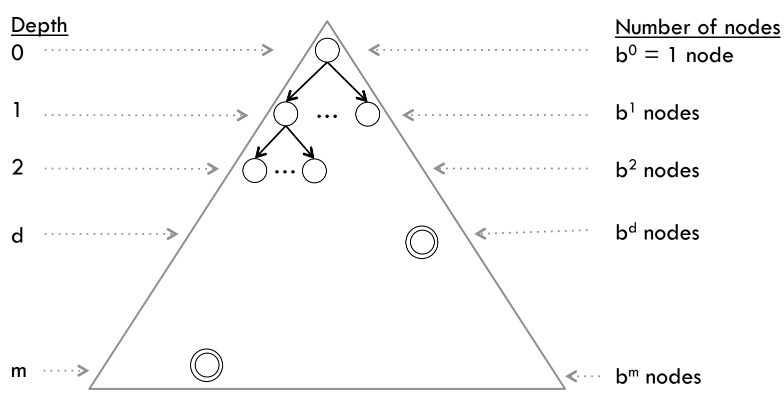


Evaluating search algorithm

- 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

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
- (Use when 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

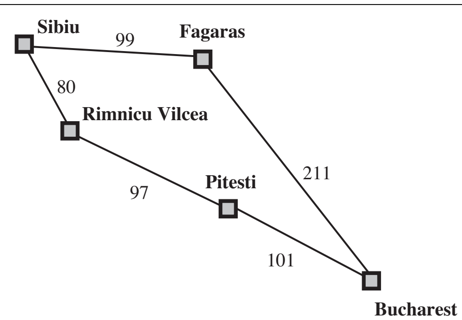
Time and memory requirements for BFS

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

Depth limited DFS

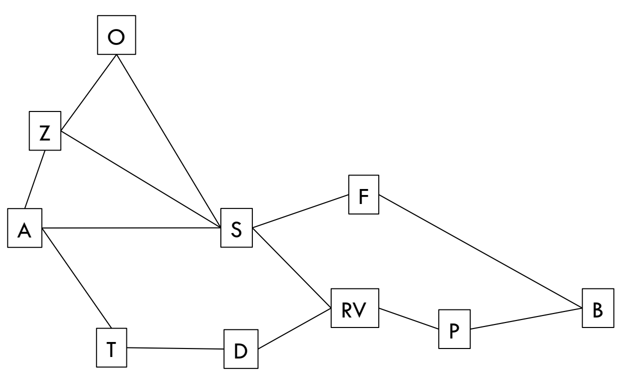
- DFS, but with a depth limit **L** specified
  - Nodes at depth **L** are treated as if they have no successors
  - We only search down to depth **L**
- Time?
  - $O(b^L)$
- Space?
  - O(bL)
- Complete?
  - No, if solution is longer than L
- Optimal
  - No, for same reasons DFS isn't

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)



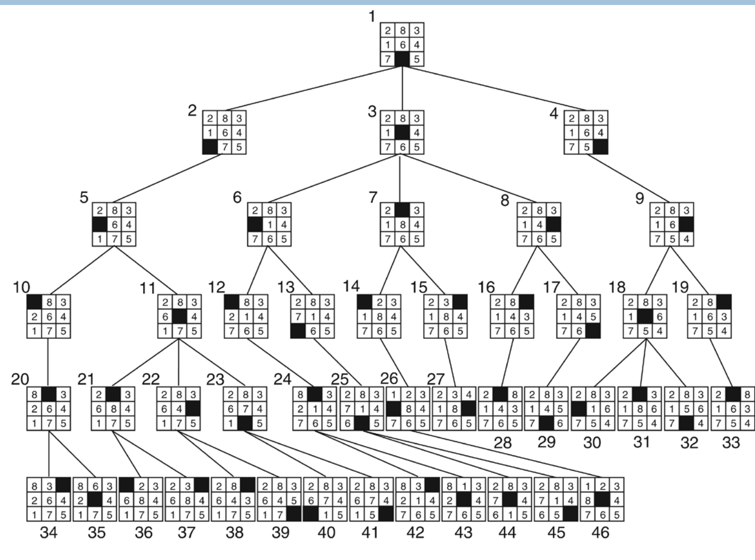
Time complexity for IDS

- $L = 0$: 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

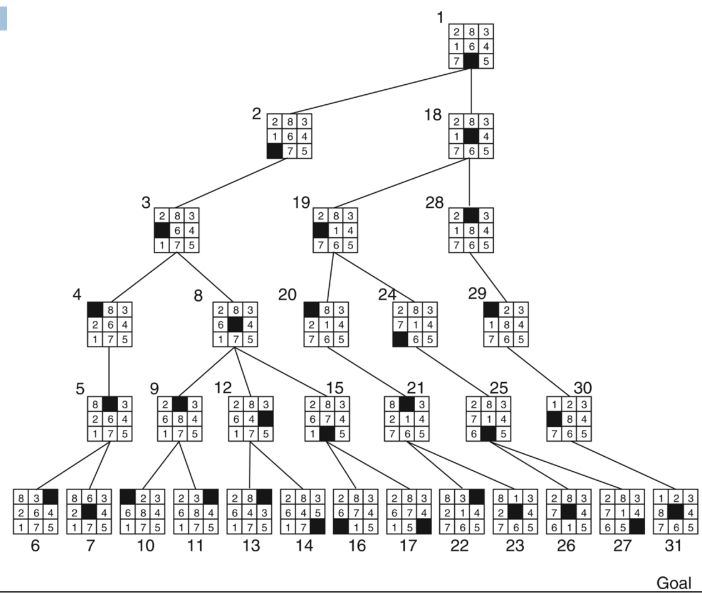
Analysis of IDS

- 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