UNINFORMED SEARCH

Today

- Uninformed search
  - Formulating the search problem
  - State-space search
  - Analyze complexity of search

State-space search

- 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

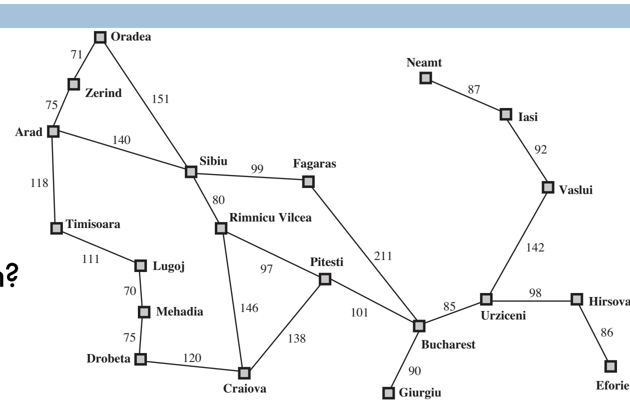
A well-defined search problem includes:

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

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

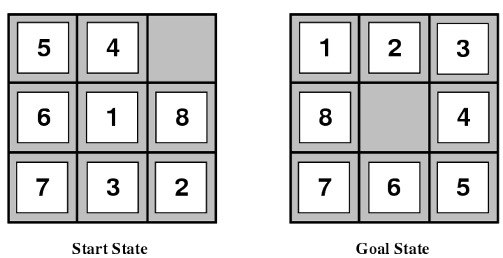
Step One: Path to Bucharest

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



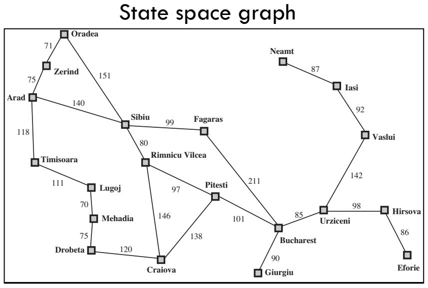
Step One: 8-puzzle

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

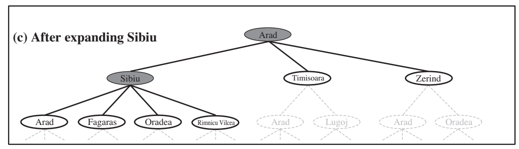


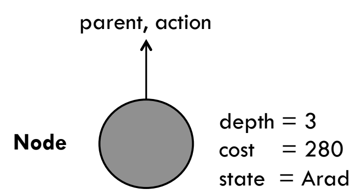
State space graph vs. Search tree

State space graph



Search tree





Step Two: Search

- If actions are reversible, redundancy in search tree
- TreeSearch
  - Does not keep track of explored nodes
  - Infinite search tree
- GraphSearch
  - Keeps track of explored nodes
  - Search tree limited to size of state space

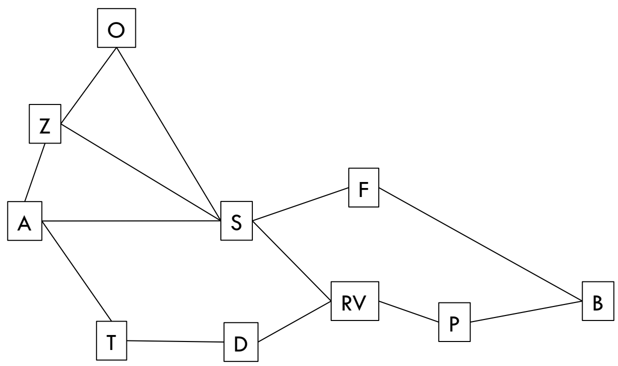
Step Two: GraphSearch

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



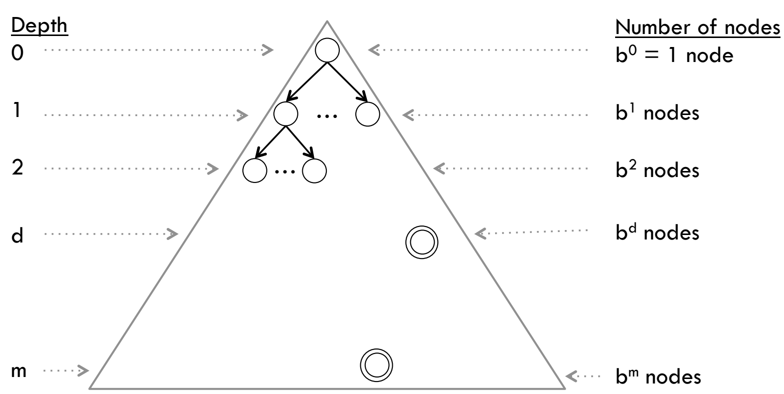
Breadth-first search (BFS)

Analyzing search algorithms

4 criteria for analyzing algorithms on board:

Notation

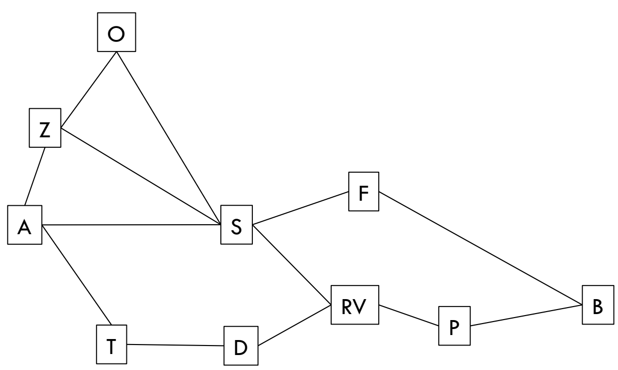
- b – branching factor, i.e. max number of successors of any node
- d – depth of the shallowest goal node
- m – maximum length of any path in state space



Analyzing BFS

Depth-first search (DFS)

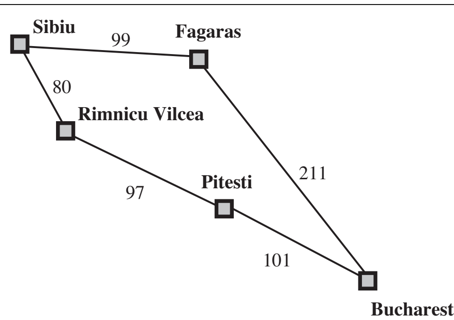
- Expand deepest unexpanded node
- Implementation: *frontier* is a LIFO queue (stack)



Analyzing DFS

Uniform-cost search

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



Uniform-cost search

Analyzing Uniform-cost search

- Let C* be the cost of the optimal solution and ε be the minimum action cost
- Time: $O(b^{C^*/\varepsilon + 1})$
- Space: $O(b^{C^*/\varepsilon + 1})$
- Complete = YES if action cost exceeds epsilon
- Optimal = YES

Depth limited DFS

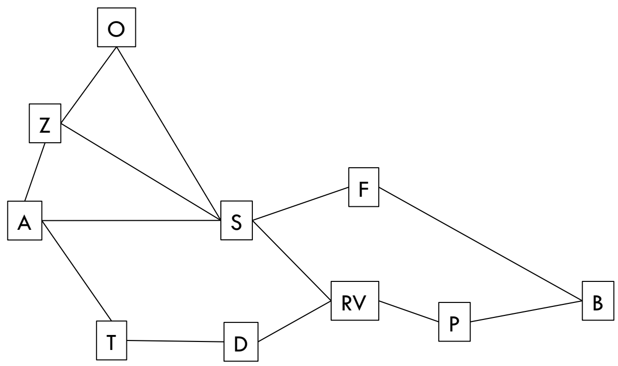
- DFS, but with a depth limit **L** specified
  - Nodes at depth **L** are treated as if they have no successors
  - We only search down to depth **L**
- Time?
  - $O(b^L)$
- Space?
  - O(bL)
- Complete?
  - No, if solution is longer than L
- Optimal
  - No, for same reasons DFS isn't

Iterative deepening search (IDS)

```
for L=0, 1, 2, …
    run depth-limited DFS with depth limit L
    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

Analysis of IDS

- Time
  - $O(b^d)$
- Space
  - O(bd)
- Complete?
  - Yes
- Optimal?
  - Yes