UNINFORMED SEARCH

Today

- Reading
  - Read AIMA 3.1-3.4
- Objectives
  - Uninformed search
    - Formulating the search problem
    - State-space search
    - Analyze complexity of search

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:

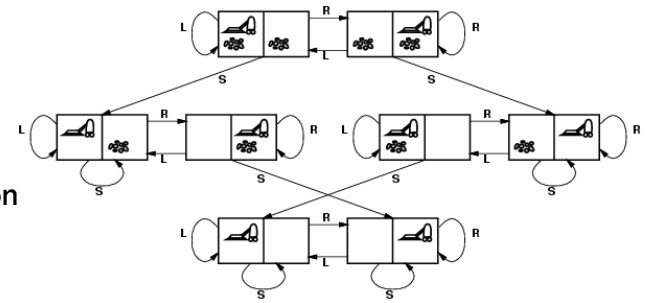
- states
- initial state
- actions
- successor function

Induce the state space graph

- goal test
- path cost (reflects performance measure)

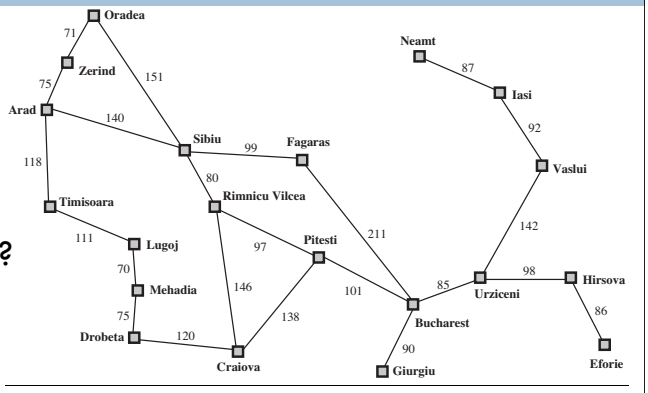
Step One: Vacuum world

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



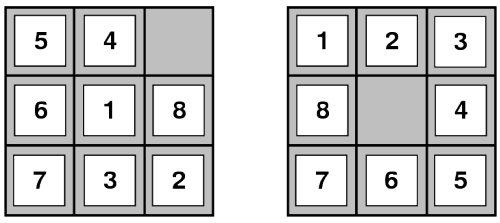
Step One: Path to Bucharest

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

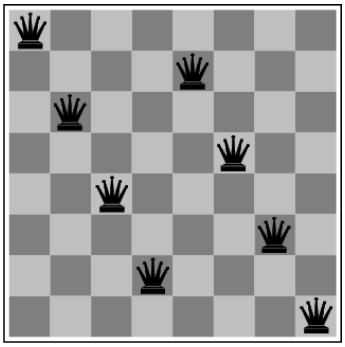


Step One: 8-puzzle

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



Step One: 8-queens puzzle



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

Step Two: Search

- Basic Algorithm
  - 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)

Tree-search

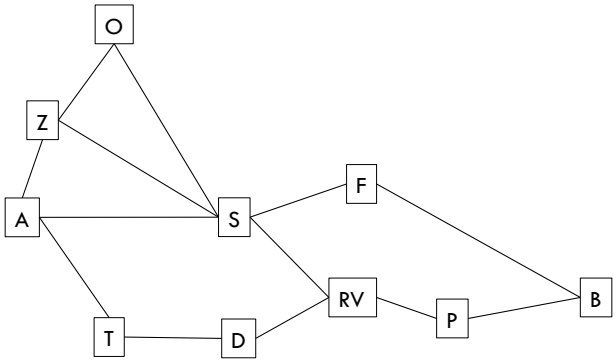
```
function TREE-SEARCH(problem, strategy) returns a solution or failure
        initialize the frontier using the initial state of problem
        loop do
                if the frontier is empty return failure
                node = pop from frontier according to strategy
                if node contains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

Search Strategies

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

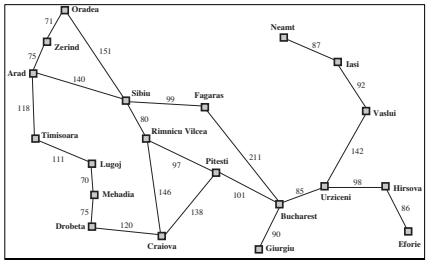
Breadth-first search (BFS)

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

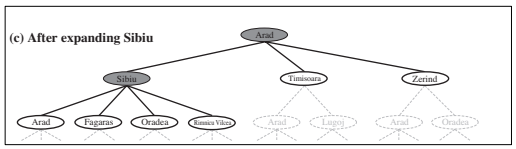


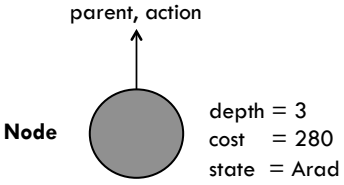
State space graph vs. Search tree

State space graph



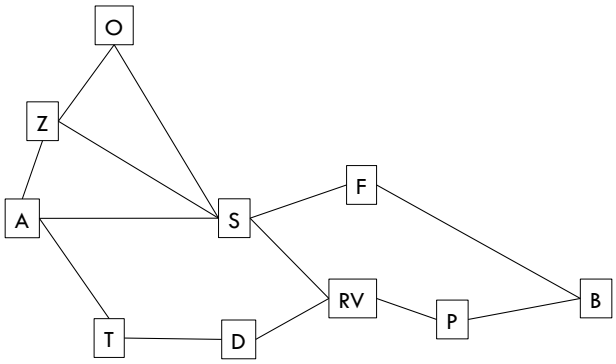
Search tree





Depth-first search (DFS)

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

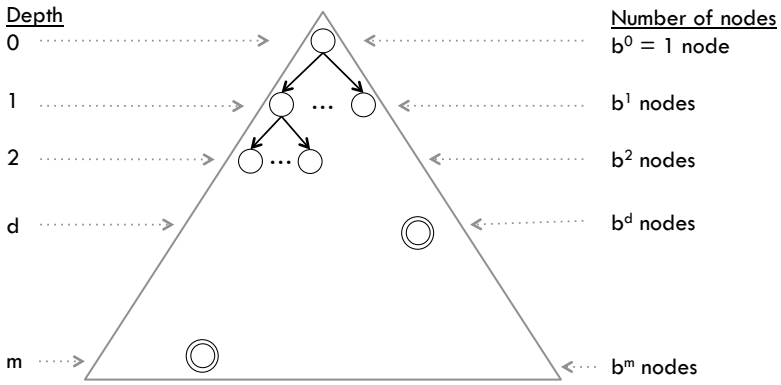


Evaluating search algorithm

- 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

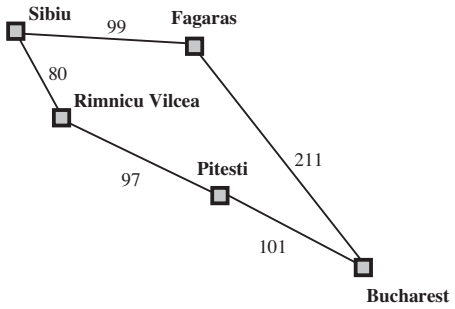
- Time: $O(b^d)$
- Space: $O(b^d)$
- Complete = YES if branching factor is finite
- Optimal = YES if path cost is non-decreasing function of depth of the node
- (Use when step costs are constant)

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

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

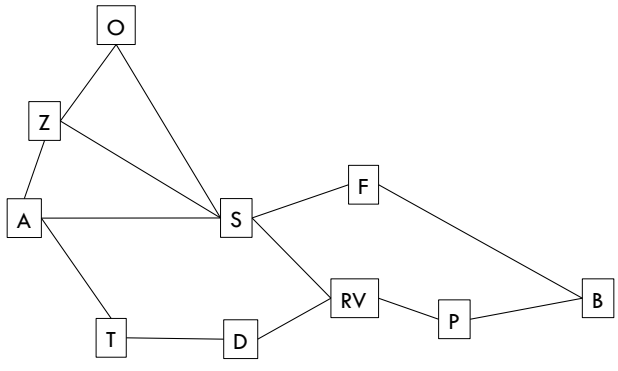
- DFS, but with a depth limit **L** specified
  - Nodes at depth **L** are treated as if they have no successors
  - We only search down to depth **L**
- Time?
  - $O(b^L)$
- Space?
  - O(bL)
- Complete?
  - No, if solution is longer than L
- Optimal
  - No, for same reasons DFS isn't

Iterative deepening search (IDS)

```
for 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

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

Analysis of IDS

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

Graph-search version 1

```
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
                node = pop from frontier according to strategy
                if node contains goal state return solution
                if node not explored
                        add node to explored set
                        expand and add successor nodes to frontier
```

Graph-search version 2

```
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
                node = pop from frontier according to strategy
                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
```

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
- Analyze search algorithms
  - Time, Space, Completeness, Optimality