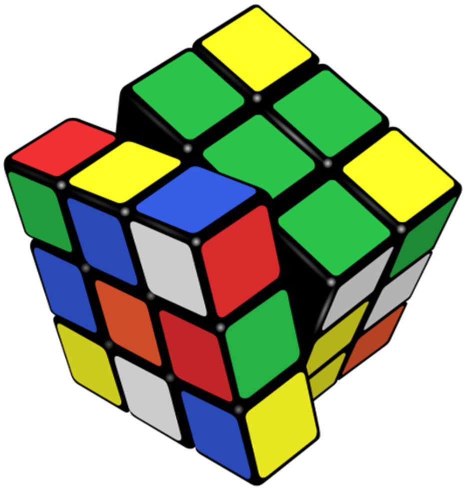


Solving Problems by Searching

[RN2] Sec 3.1-3.5
[RN3] Sec 3.1-3.4

CS486/686
University of Waterloo
Lecture 2: Sept 13, 2012

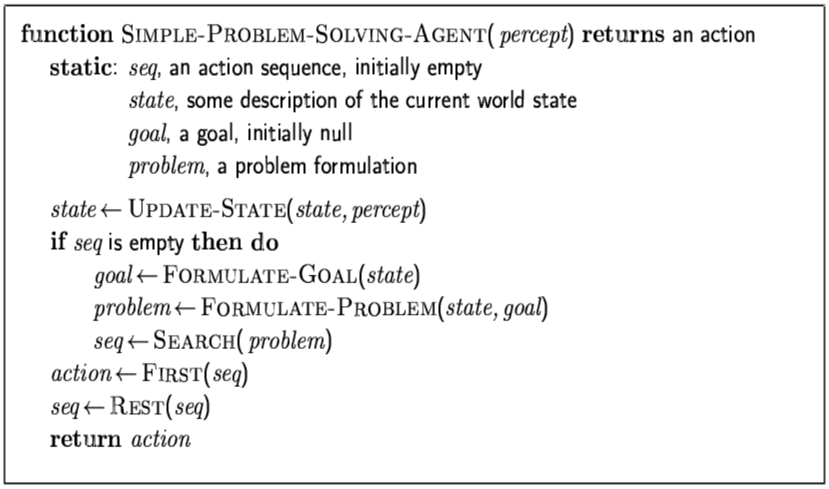
Outline

- Problem solving agents and search
- Examples
- Properties of search algorithms
- Uninformed search
  - Breadth first
  - Depth first
  - Iterative Deepening

Introduction

- Search was one of the first topics studied in AI
  - Newell and Simon (1961) *General Problem Solver*
- Central component to many AI systems
  - Automated reasoning, theorem proving, robot navigation, VLSI layout, scheduling, game playing,…

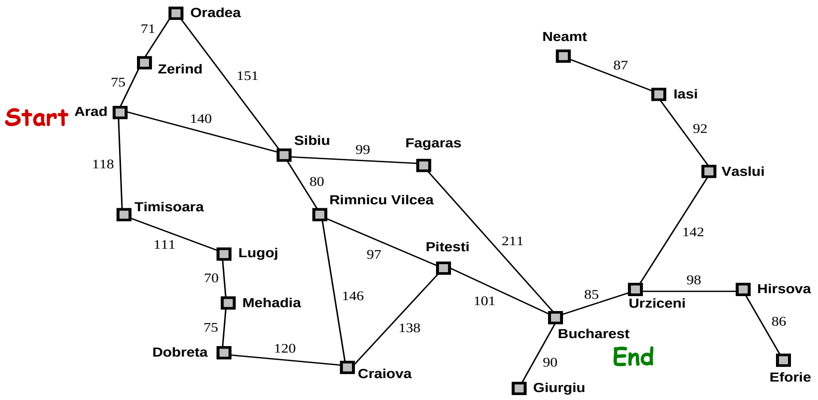
Problem-solving agents



```
function SIMPLE-PROBLEM-SOLVING-AGENT(percept) returns an action
   static: seq, an action sequence, initially empty
           state, some description of the current world state
           goal, a goal, initially null
           problem, a problem formulation

   state ← UPDATE-STATE(state, percept)
   if seq is empty then do
       goal ← FORMULATE-GOAL(state)
       problem ← FORMULATE-PROBLEM(state, goal)
       seq ← SEARCH(problem)
   action ← FIRST(seq)
   seq ← REST(seq)
   return action
```

Example: Traveling in Romania



Formulate Goal

Get to Bucharest

Formulate Problem

Initial state: In(Arad)

Actions: Drive between cities

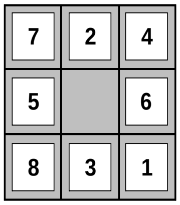
Goal Test: In(Bucharest)?

Path cost: Distance between cities

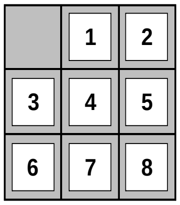
Find a solution

Sequence of cities: Arad, Sibiu, Fagaras, Bucharest

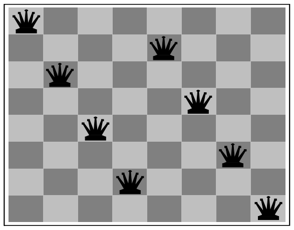
Examples of Search Problems



Start State



Goal State



States: Locations of 8 tiles and blank

Initial State: Any state

Succ Func: Generates legal states that result from trying 4 actions (blank up, down, left, right)

Goal test: Does state match desired configuration

Path cost: Number of steps

States: Arrangement of 0 to 8 queens on the board

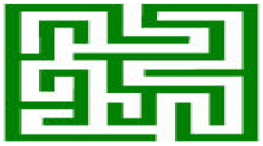
Initial State: No queens on the board

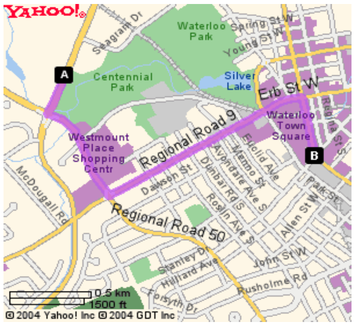
Succ Func: Add a queen to an empty space

Goal test: 8 queens on board, none attacked

Path cost: none

More Examples





Common Characteristics

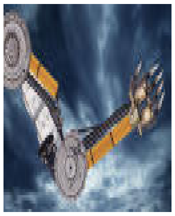
- All of those examples are
  - Fully observable
  - Deterministic
  - Sequential
  - Static
  - Discrete
  - Single agent

- Can be tackled by **simple** search techniques

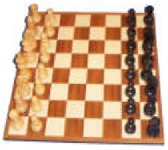
Cannot tackle these yet…

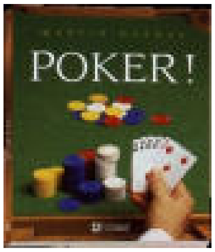
Chance

Infinite number of states



Games against
an adversary





Hidden states

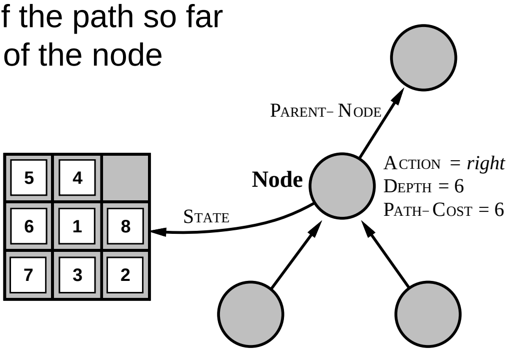
All of the
above

Searching

- We can formulate a search problem
  - Now need to find the solution
- We can visualize a state space search in terms of trees or graphs
  - Nodes correspond to states
  - Edges correspond to taking actions
- We will be studying search trees
  - These trees are constructed “on the fly” by our algorithms

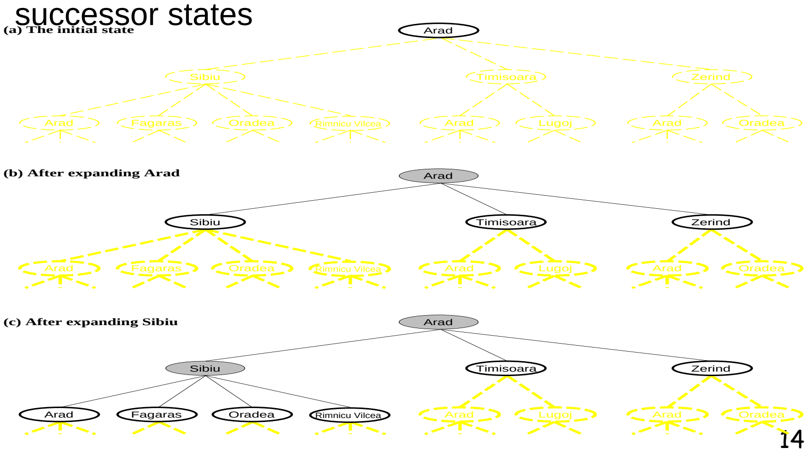
Data Structures for Search

- Basic data structure: Search Node
  - State
  - Parent node and operator applied to parent to reach current node
  - Cost of the path so far
  - Depth of the node



Expanding Nodes

- Expanding a node
  - Applying all legal operators to the state contained in the node and generating nodes for all corresponding successor states



Generic Search Algorithm

1. Initialize search algorithm with initial state of the problem
2. **Repeat**
    1. If no candidate nodes can be expanded, **return failure**
    2. Choose leaf node for expansion, according to search strategy
    3. If node contains a goal state, **return solution**
    4. Otherwise, expand the node, by applying legal operators to the state within the node. Add resulting nodes to the tree

---

Implementation Details

- We need to only keep track of nodes that need to be expanded (fringe)
    - Done by using a (prioritized) **queue**

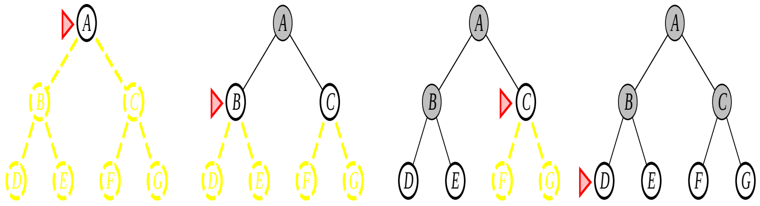
1. Initialize queue by inserting the node corresponding to the initial state of the problem
2. Repeat
    1. If queue is empty, **return failure**
    2. Dequeue a node
    3. If the node contains a goal state, **return solution**
    4. Otherwise, expand node by applying legal operators to the state within. Insert resulting nodes into queue

Search algorithms differ in their queuing function!

Breadth-first search

All nodes on a given level are expanded before any nodes on the next level are expanded.

Implemented with a FIFO queue



A B,C C,D,E D,E,F,G

Evaluating search algorithms

- Completeness: Is the algorithm guaranteed to find a solution if a solution exists?
- Optimality: Does the algorithm find the optimal solution (Lowest path cost of all solutions)
- Time complexity
- Space complexity

Variables

b	Branching factor
d	Depth of shallowest goal node
m	Maximum length of any path in the state space

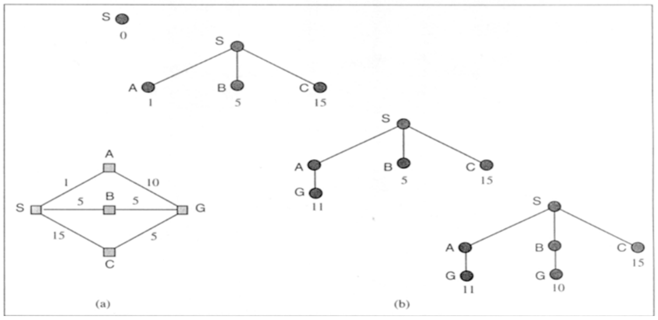
Judging BFS

- **Complete:**
  - Yes, if b is finite
- **Optimal:**
  - Yes, if all costs are the same
- **Time:**
  - $1+b+b^2+b^3+\ldots+b^d = O(b^d)$
- **Space:**
  - $O(b^d)$

All uninformed search methods will have exponential time complexity ☹

Uniform Cost Search

- A variation of breadth-first search
  - Instead of expanding shallowest node it expands the node with lowest path cost
  - Implemented using a priority queue



C* is cost of optimal solution

ε is minimum action cost

Optimal Yes

Complete if $\varepsilon > 0$

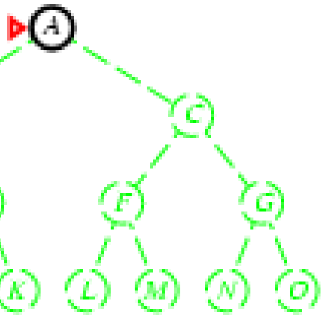
Time: $O(b^{\text{ceiling}(C^*/\varepsilon)})$

Space: $O(b^{\text{ceiling}(C^*/\varepsilon)})$

Depth-first search

The deepest node in the current fringe of the search tree is expanded first.

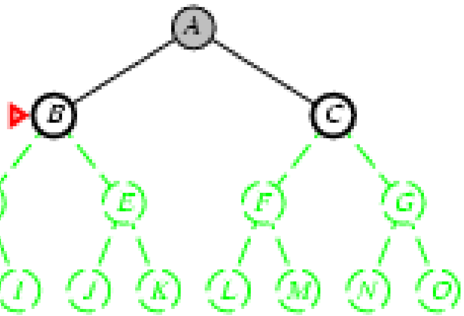
Implemented with a stack (LIFO queue)



Depth-first search

The deepest node in the current fringe of the search tree is expanded first.

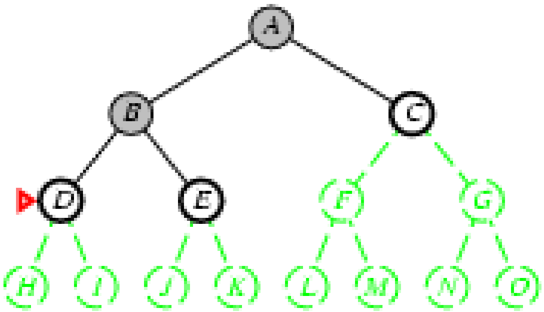
Implemented with a stack (LIFO queue)



Depth-first search

The deepest node in the current fringe of the search tree is expanded first.

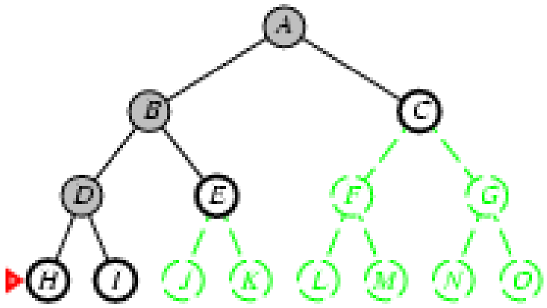
Implemented with a stack (LIFO queue)



Depth-first search

The deepest node in the current fringe of the search tree is expanded first.

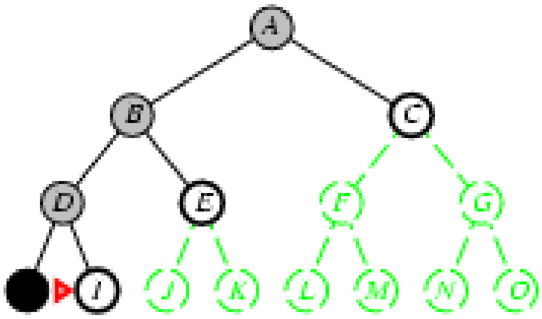
Implemented with a stack (LIFO queue)



Depth-first search

The deepest node in the current fringe of the search tree is expanded first.

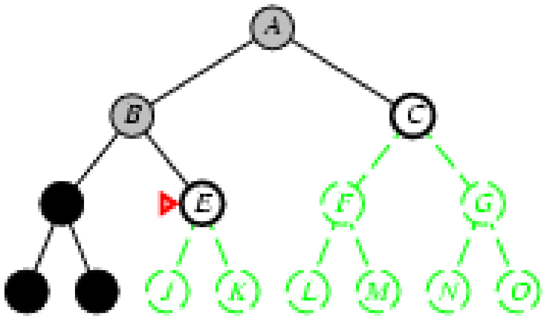
Implemented with a stack (LIFO queue)



Depth-first search

The deepest node in the current fringe of the search tree is expanded first.

Implemented with a stack (LIFO queue)



Judging DFS

- **Complete:**
  - No, might get stuck going down a long path
- **Optimal:**
  - No, might return a solution which is deeper (i.e. more costly) than another solution
- **Time:**
  - $O(b^m)$, m might be larger than d
- **Space:**
  - $O(bm)$ ☺

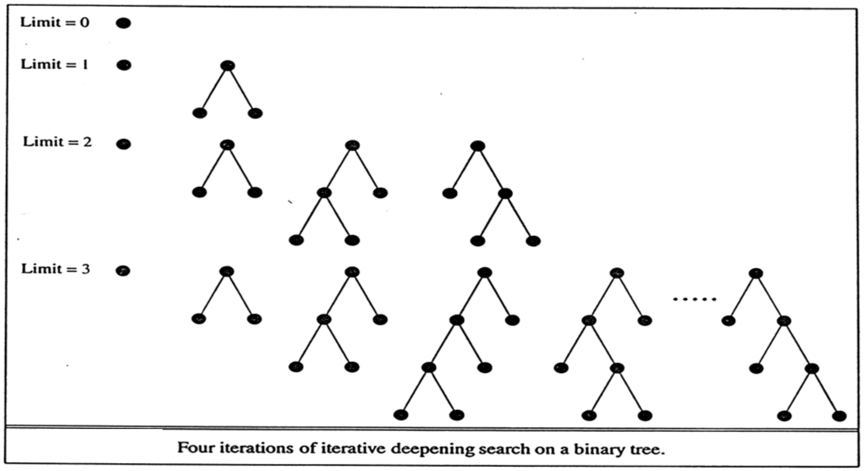
Do not use DFS if you suspect a large tree depth

Depth-limited search

- We can avoid the problem of unbounded trees by using a **depth limit**, l
  - All nodes at depth l are treated as though they have no successors
  - If possible, choose l based on knowledge of the problem
- **Time:** $O(b^l)$
- **Memory:** $O(bl)$
- **Complete?:** No
- **Optimal?:** No

Iterative-deepening

- General strategy that repeatedly does depth-limited search, but increases the limit each time



Four iterations of iterative deepening search on a binary tree.

Iterative-deepening

IDS is not as wasteful as one might think.

Note, most nodes in a tree are at the bottom level. It does not matter if nodes at a higher level are generated multiple times.

Breadth first search :
$1 + b + b^2 + \ldots + b^{d-1} + b^d$
E.g. b=10, d=5: 1+10+100+1,000+10,000+100,000 = 111,111

Iterative deepening search :
$(d+1)*1 + (d)*b + (d-1)*b^2 + \ldots + 2b^{d-1} + 1b^d$
E.g. 6+50+400+3000+20,000+100,000 = 123,456

Complete, Optimal, $O(b^d)$ time, $O(bd)$ space

Summary

- Problem formulation usually requires abstracting away real-world details to define a state space that can feasibly be explored
- Variety of uninformed search strategies
  - Assume no knowledge about the problem (general but expensive)
  - Mainly differ in the order in which they consider the states

Criteria	BFS	Uniform	DFS	DLS	IDS
Complete	Yes	Yes	No	No	Yes
Time	$O(b^d)$	$O(b^{ceiling(C^*/\varepsilon)})$	$O(b^m)$	$O(b^l)$	$O(b^d)$
Space	$O(b^d)$	$O(b^{ceiling(C^*/\varepsilon)})$	O(bm)	O(bl)	O(bd)
Optimal	Yes	Yes	No	No	Yes

Summary

- Iterative deepening uses only **linear space** and not much more time than other uninformed search algorithms
  - Use IDS when there is a large state space and the maximum depth of the solution is unknown
- Things to think about:
  - What about searching graphs?
  - Repeated states?

Next Class

- Informed search techniques