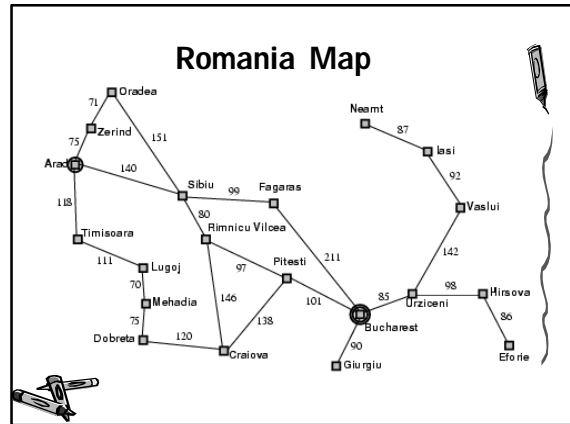






Uninformed Search

CIS*3700 (Winter 2007)



Example: Map Problem

- Problem: drive from Arad to Bucharest.
- Formulate a goal: be in Bucharest
 - Current in Arad
- Formulate the problem:
 - States: various cities on the map
 - Actions: drive between two cities
- Search for a solution:
 - Sequence of cities, e.g., Arad → Sibiu → Fagaras → Bucharest.



State Space Representation

- A set of states which contain all the possible configurations of the objects relevant to a problem.
- A set of initial states.
- A set of goal states.
- A set of actions (or operators) which allow us to move from one state to another.
- Solving a problem is to find a path that goes from an initial state to a goal state.

Water Jugs Problem

- There are two water jugs: one is 4-gallon, and the other, 3-gallon.
- Neither has any measuring markers on it.
- There is a pump that can be used to fill the jugs with water.
- You can pour water from one jug to another or onto the ground.
- How do you get exactly 2 gallons of water into the 3-gallon jug?



Water Jugs Problem

- The set of states:

$$\{(x, y) \mid 0 \leq x \leq 4, 0 \leq y \leq 3\}$$
- The set of initial states:

$$\{(0, 0)\}$$
- The set of goal states:

$$\{(x, 2) \mid 0 \leq x \leq 4\}$$
- How about the water pump and the ground?

Water Jugs Problem

- The set of actions:
 - (x,y) and $x < 4 \rightarrow (4,y)$ -- fill 4-gallon jug
 - (x,y) and $y < 3 \rightarrow (x, 3)$
 - (x,y) and $x > 0 \rightarrow (0,y)$ -- empty 4-gallon jug
 - (x,y) and $y > 0 \rightarrow (x,0)$
 - (x,y) and $x+y \geq 4$ and $y > 0 \rightarrow (4, y-(4-x))$
 - (x,y) and $x+y \geq 3$ and $x > 0 \rightarrow (x-(3-y), 3)$
 - (x,y) and $x+y \leq 4$ and $y > 0 \rightarrow (x+y, 0)$
 - (x,y) and $x+y \leq 3$ and $x > 0 \rightarrow (0, x+y)$

Water Jugs Problem

- One possible solution:

8-Puzzle Problem

- What is a suitable state space representation?

7	2	4
5		6
8	3	1

Initial State

	1	2
3	4	5
6	7	8

Goal State

Search Problems

- Toy problems:
 - Concise and exact descriptions are possible
 - Performance of algorithms can be compared
 - E.g., Vacuum world, 8-queens, and checker.
- Real-world problems:
 - People care about the solutions
 - No single agreed-upon descriptions
 - E.g., route finding (network routing and airline travel planning), VLSI design, and robot navigation.

Problem-Solving Agent

```

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
  
```

Problem Types

- Deterministic, fully observable \rightarrow single-state problem
 - Agent knows exactly which state it will be in; solution is a sequence
- Non-observable \rightarrow sensorless/conformant problem
 - Agent may have no idea where it is; solution is a sequence
- Nondeterministic and/or partially observable \rightarrow contingency problem
 - percepts provide new information about current state
 - often interleave search and execution
- Unknown state space \rightarrow exploration problem

Single-State Problem

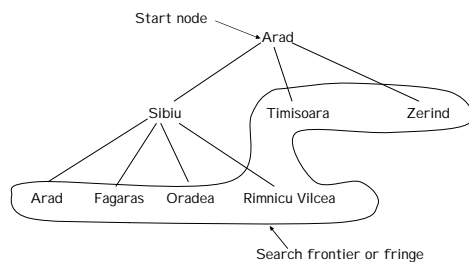
- Problem formation:
 - using the state space representation.
- Performance measure (additive)
 - Does it find a solution at all?
 - Is it a good solution (with the lowest path cost)?
 - What is the search cost in terms of time and space?

Ideas for Tree Search

- Offline, simulated exploration of state space by generating successors of already-explored states.

```
function TREE-SEARCH(problem, strategy) returns a solution, or failure
  initialize the search tree using the initial state of problem
  loop do
    if there are no candidates for expansion then return failure
    choose a leaf node for expansion according to strategy
    if the node contains a goal state then return the corresponding solution
    else expand the node and add the resulting nodes to the search tree
```

Tree Search Example



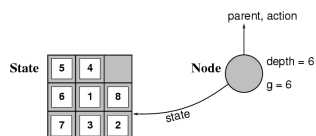
Tree Search Algorithm

```
function TREE-SEARCH(problem, fringe) returns a solution, or failure
  fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
  loop do
    if fringe is empty then return failure
    node ← REMOVE-FRONT(fringe)
    if GOAL-TEST[problem](STATE[node]) then return SOLUTION(node)
    fringe ← INSERT ALL(EXPAND(node, problem), fringe)

function EXPAND(node, problem) returns a set of nodes
  successors ← the empty set
  for each action, result in SUCCESSOR-FN[problem](STATE[node]) do
    s ← a new NODE
    PARENT-NODE[s] ← node, ACTION[s] ← action, STATE[s] ← result
    PATH-COST[s] ← PATH-COST[node] + STEP-COST(node, action, s)
    DEPTH[s] ← DEPTH[node] + 1
    add s to successors
  return successors
```

Data Structures for Tree Search

- A tree node includes state, parent node, action, path cost $g(x)$, depth.



- A fringe is a queue of ordered nodes.

Measures of Search Strategies

- Completeness: guarantee to find a solution if one exists.
- Optimality: guarantee to find the optimal solution.
- Time complexity: time taken to find a solution
 - Measured by max # of nodes generated.
- Space complexity: memory needed to perform the search
 - Measured by max # of nodes stored.

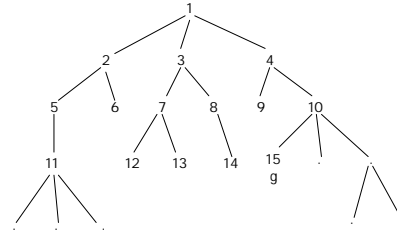
Uniformed Search Strategies

- Uninformed search strategies use only the information available in the problem definition
- Different Strategies:
 - Breadth-first search (BFS)
 - Uniform-cost search
 - Depth-first search (DFS)
 - Depth-limited search
 - Iterative deepening search



Breadth-First Search

- Maintain fringe as a queue by putting successors at the end.



Properties of BFS

- Complete? Yes (if b is finite)
- Time? $1+b+b^2+b^3+\dots+b^d + b(b^d-1) = O(b^{d+1})$
- Space? $O(b^{d+1})$ (keeps every node in memory)
- Optimal? Yes (if cost = 1 per step)



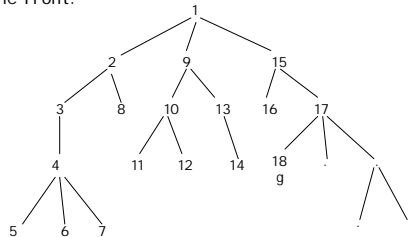
Uniform-Cost Search

- Maintain fringe as queue ordered by path cost
 - Equivalent to breadth-first if step costs are all equal
- Complete? Yes, if step cost = ϵ
- Time? # of nodes with $g =$ cost of optimal solution, $O(b^{\lceil C^*/\epsilon \rceil})$ where C^* is the cost of the optimal solution
- Space? # of nodes with $g =$ cost of optimal solution, $O(b^{\lceil C^*/\epsilon \rceil})$
- Optimal? Yes - nodes expanded in increasing order of $g(n)$



Depth-First Search

- Maintain fringe as a stack by putting successors at the front.



Properties of DFS

- Complete? No: fails in infinite-depth spaces, spaces with loops
 - Modify to avoid repeated states along path
 - complete in finite spaces
- Time? $O(b^m)$: terrible if m is much larger than d
 - but if solutions are dense, may be much faster than breadth-first
- Space? $O(bm)$, i.e., linear space!
- Optimal? No



Depth-Limited Search

- Depth-first search with depth limit l : no successors beyond this level.

```

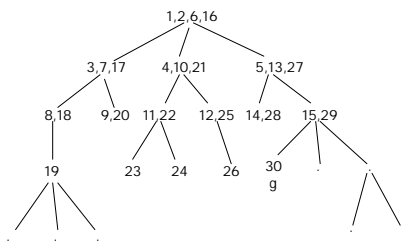
function DEPTH-LIMITED-SEARCH(problem, limit) returns soln/fail/cutoff
  RECURSIVE-DLS(MAKE-NODE(INITIAL-STATE[problem]), problem, limit)
function RECURSIVE-DLS(node, problem, limit) returns soln/fail/cutoff
  cutoff-occurred? ← false
  if GOAL-TEST[problem](STATE[node]) then return SOLUTION(node)
  else if DEPTH[node] = limit then return cutoff
  else for each successor in EXPAND(node, problem) do
    result ← RECURSIVE-DLS(successor, problem, limit)
    if result = cutoff then cutoff-occurred? ← true
    else if result ≠ failure then return result
  if cutoff-occurred? then return cutoff else return failure
    
```

Iterative Deepening

```

function ITERATIVE-DEEPENING-SEARCH(problem) returns a solution, or failure
  for depth ← 0 to ∞ do
    result ← DEPTH-LIMITED-SEARCH(problem, depth)
    if result ≠ cutoff then return result
    
```

Iterative Deepening



Iterative Deepening

- Number of nodes generated in a depth-limited search to depth d with branching factor b :

$$N_{DLS} = b^0 + b^1 + b^2 + \dots + b^{d-2} + b^{d-1} + b^d$$

- Number of nodes generated in an iterative deepening search to depth d with branching factor b :

$$N_{IDS} = (d+1)b^0 + d b^1 + (d-1)b^2 + \dots + 3b^{d-2} + 2b^{d-1} + 1b^d$$

- For $b = 10$, $d = 5$,
 - $N_{DLS} = 1 + 10 + 100 + 1,000 + 10,000 + 100,000 = 111,111$
 - $N_{IDS} = 6 + 50 + 400 + 3,000 + 20,000 + 100,000 = 123,456$

- Overhead = $(123,456 - 111,111) / 111,111 = 11\%$

Properties of Iterative Deepening

- Complete? Yes
- Time? $(d+1)b^0 + d b^1 + (d-1)b^2 + \dots + b^d = O(b^d)$
- Space? $O(bd)$
- Optimal? Yes, if step cost = 1

Summary

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening
Complete?	Yes	Yes	No	No	Yes
Time	$O(b^{d+1})$	$O(b^{C^*/\epsilon})$	$O(b^m)$	$O(b^l)$	$O(b^d)$
Space	$O(b^{d+1})$	$O(b^{C^*/\epsilon})$	$O(bm)$	$O(bl)$	$O(bd)$
Optimal?	Yes	Yes	No	No	Yes