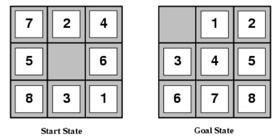
Search

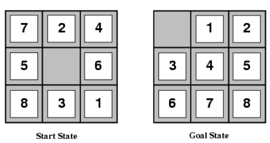
Dr. Melanie Martin
CS 4480
September 14, 2012

Example: The 8-puzzle



- states?
- actions?
- goal test?
- path cost?

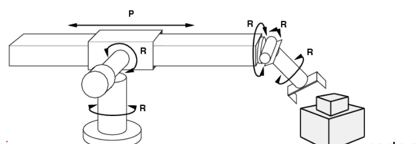
Example: The 8-puzzle



- states? locations of tiles
- actions? move blank left, right, up, down
- goal test? = goal state (given)
- path cost? 1 per move

[Note: optimal solution of *n*-Puzzle family is NP-hard]

Example: robotic assembly



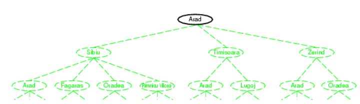
- states ... parts of the object to be assembled
- actions?: continuous motions of robot joints
- goal test?: complete assembly
- path cost?: time to execute

Tree search algorithms

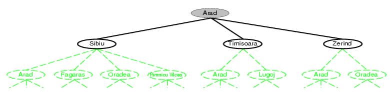
- Basic idea:
  - offline, simulated exploration of state space by generating successors of already-explored states (a.k.a.~expanding states)

```
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
```

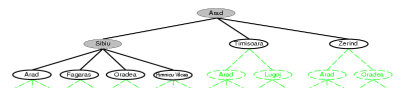
Tree search example



Tree search example



Tree search example



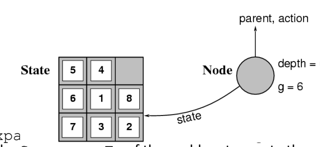
Implementation: general tree search

```
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 ← INSERTALL(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
```

Implementation: states vs. nodes

- A state is a (representation of) a physical configuration
- A node is a data structure constituting part of a search tree includes state, parent node, action, path cost *g(x)*, depth



- The Expa ious fields and using the SuccessorFn of the problem to create the corresponding states.

Search strategies

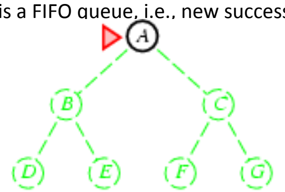
- A search strategy is defined by picking the order of node expansion
- Strategies are evaluated along the following dimensions:
  - completeness: does it always find a solution if one exists?
  - time complexity: number of nodes generated
  - space complexity: maximum number of nodes in memory
  - optimality: does it always find a least-cost solution?
- Time and space complexity are measured in terms of
  - *b:* maximum branching factor of the search tree
  - *d:* depth of the least-cost solution
  - *m*: maximum depth of the state space (may be ∞)

Uninformed search strategies

- Uninformed search strategies use only the information available in the problem definition
- Breadth-first search
- Uniform-cost search
- Depth-first search
- Depth-limited search

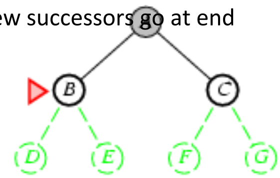
Breadth-first search

- Expand shallowest unexpanded node
- Implementation:
  - *fringe* is a FIFO queue, i.e., new successors go at end



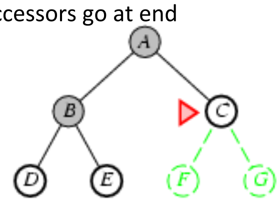
Breadth-first search

- Expand shallowest unexpanded node
- Implementation: *fringe* is a FIFO queue, i.e., new successors go at end



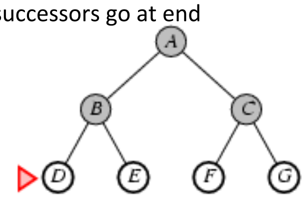
Breadth-first search

- Expand shallowest unexpanded node
- Implementation: *fringe* is a FIFO queue, i.e., new successors go at end



Breadth-first search

- Expand shallowest unexpanded node
- Implementation: *fringe* is a FIFO queue, i.e., new successors go at end



Properties of breadth-first search

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

- Space is the bigger problem (more than time)