

Problem Solving by Searching

Lecture #3
9/02/08

Announcements

- By now, you should have read Chapters 1-3 of your text
- Read Chapter 4 for Thursday

The missionaries and cannibals problem¹

- Goal: transport 3 missionaries and 3 cannibals from the left bank of a river to the right bank.
- Constraints:
 - Whenever cannibals outnumber missionaries, the missionaries get eaten
 - The boat can hold at most two people
 - The boat can't cross the river empty
 - But both missionaries & cannibals can operate the boat

1) I apologize for the politically incorrect nature of this problem – I prefer wolves & sheep – but it's a classic problem and you need to recognize it by its common name

How do you solve this problem?

- It's not an equation
- It's not a continuous function to regress
- It's a set of discrete states connected by operators
- An agent "searches" for an answer by trying sequences of actions – and this is what we will formalize

Problem solving agents

- Goal Formulation
 - Desired state of the world.
- Problem Formulation
 - States and actions to consider in view of goal.
- Search
 - Determine the possible sequence of actions that lead to the states of known values and then choosing the best sequence.
- Execute
 - Given the solution, perform the actions.
- Assumption:
 - Environment is fully observable, deterministic
 - Agent knows the effects of its actions

Back to our example

- A state description that allows us to describe our goal:

(M_L, C_L, B)

Where:

M_L : number of missionaries on left bank

C_L : number of cannibals on left bank

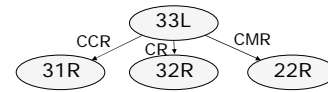
B: location of boat (L,R)

- Initial state: (3,3,L) Goal: (0,0,R)

Graph formulation of the search problem

- Nodes: states of the problem.
- Edges: there is an edge from state u to state v if there is an action that takes you from v to u
 - What are the edges for missionaries and cannibals problem?
 - Can you apply all actions from any node?
- Problem is now to find a path from the start state (3,3,L) to the goal state (0,0,R).
- In general, paths have costs associated with them
 - What cost is appropriate for missionaries & cannibals?
- The problem is to find the lowest cost path from the initial state to the goal.

The search graph

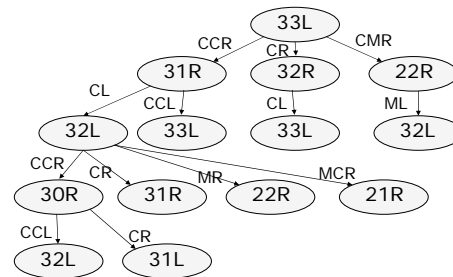


Actions (operators):
 CCR - transport two cannibals to the right bank
 MCL - transport a missionary and a cannibal to the left bank

At each step...

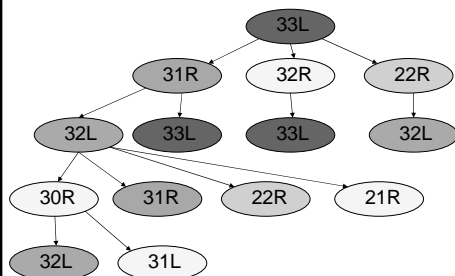
- Check to see if solution is legal
 - Can't have more cannibals than missionaries on either bank
- Compute cost of current path
 - How many boat trips have we made?
- Figure out which actions are possible
 - Possible next steps

The (partially expanded) search graph



State = (ML, CL, B)

Repeated states

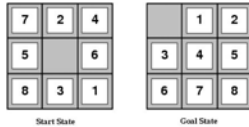


The search graph is not necessarily a tree!

M&C Solution

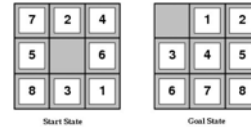
State (left bank)	Operator	Result State
33L	CC→	31R
31R	C←	32L
32L	CC→	30R
30R	C←	31L
31L	MM→	11R
11R	MC←	22L
22L	MM→	02R
02R	C←	03L
03L	CC→	01R
01R	C←	02L
02L	CC→	00R

The 8 puzzle



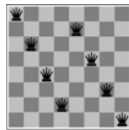
- States?
- Initial state?
- Actions?
- Goal test?
- Path cost?

The 8 puzzle



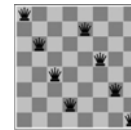
- States? Integer location of each tile
- Initial state? Any state
- Actions? (tile, direction)
 - where direction is one of {Left, Right, Up, Down}
- Goal test? Check whether goal configuration is reached
- Path cost? Number of actions to reach goal
- Is the search graph a tree?

8 queens problem



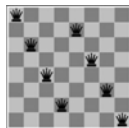
- Incremental vs. complete state formulation:
 - Incremental formulation starts with an empty state and involves operators that augment the state description
 - A complete state formulation starts with all 8 queens on the board and moves them around

8 queens problem: representation is key



- Incremental formulation
- States? Any arrangement of 0 to 8 queens on the board
 - Initial state? No queens
 - Actions? Add queen in empty square
 - Goal test? 8 queens on board and none attacked
 - Path cost? None
- 3×10^{14} possible sequences to investigate
- Is the search graph a tree?

8 queens problem: a better representation

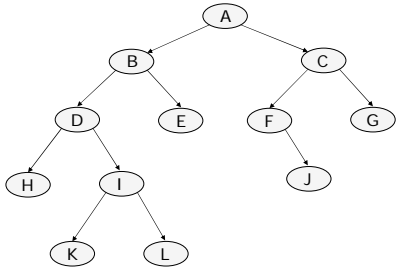


- Another Incremental formulation:
- States? n (between 0 and 8) queens on the board, one in each of the n left-most columns; no queens attacking each other.
 - Initial state? No queens
 - Actions? Add queen in leftmost empty column such that it does not attack any of the queens already on the board.
 - Goal test? 8 queens on board
- 2057 possible sequences to investigate

Real world problems

- Route Finding
- Traveling Salesperson Problem (TSP)
- VLSI Layout
- Robot Navigation.

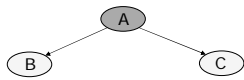
Navigating through a search tree



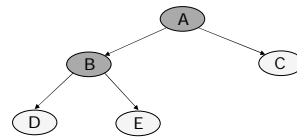
Navigating through a search tree



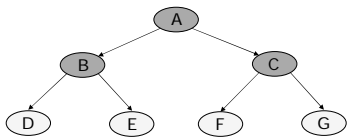
Navigating through a search tree



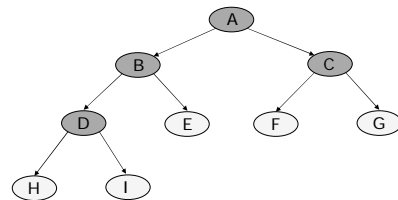
Navigating through a search tree



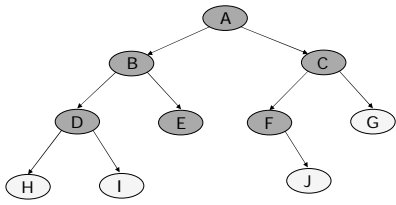
Navigating through a search tree



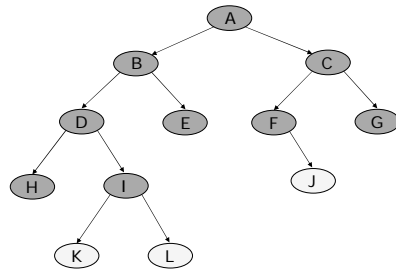
Navigating through a search tree



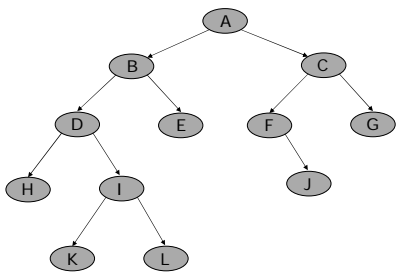
Navigating through a search tree



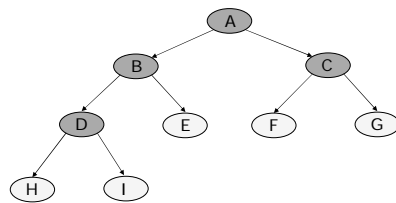
Navigating through a search tree



Navigating through a search tree




Unexpanded nodes: the fringe



At every point in the search process we keep track of a list of nodes that haven't been expanded yet: the fringe

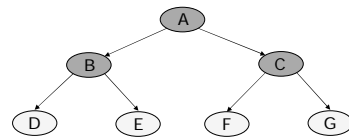
Tree search

Initial state 

```

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

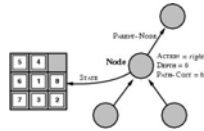
Tree search



```

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

What's in a node



- State
- Parent
- Operator (Used to generate node)
- Depth
- Path-Cost

Tree search

```

function TREE-SEARCH(problem, fringe) return a solution or failure
  fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
  loop do
    if EMPTY(fringe) then return failure
    node ← REMOVE-FIRST(fringe)
    if GOAL-TEST[problem] applied to STATE[node] succeeds
      then return SOLUTION(node)
    fringe ← INSERT-ALL(EXPAND(node, problem), fringe)
  
```

Comparing search strategies

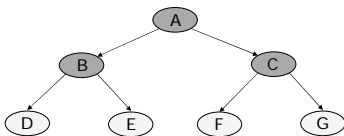
- A strategy is defined by the order of node expansion.
- Problem-solving performance is measured in four ways:
 - Completeness: *Does it always find a solution if one exists?*
 - Optimality: *Does it always find the least-cost solution?*
 - Time Complexity: *Number of nodes generated/expanded.*
 - Space Complexity: *Number of nodes stored in memory during search.*
- 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

- (a.k.a. blind search) = use only information available in problem definition.
 - When strategies can determine whether one non-goal state is better than another → informed search.
- Categories defined by expansion algorithm:
 - Breadth-first search
 - Uniform-cost search
 - Depth-first search
 - Depth-limited search
 - Iterative deepening search.
 - Bidirectional search

Breadth-First Search (BFS)

- Expand all nodes at depth *d* before proceeding to depth *d*+1.
- Implementation: FIFO queue.



Evaluation of BFS

- Completeness:
 - *Does it always find a solution if one exists?*
 - YES (if shallowest goal node is at some finite depth *d*)

Evaluation of BFS

- Completeness:
 - YES
- Time complexity:
 - Assume a state space where every state has b successors.
 - Assume solution is at depth d
 - Worst case: expand all but the last node at depth d
 - Total number of nodes expanded:

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

Evaluation of BFS

- Completeness:
 - YES
- Time complexity:
 - Total number of nodes generated:

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

- Space complexity:
 - Same, if each node is retained in memory

Evaluation of BFS

- Completeness:
 - YES
- Time complexity:
 - Total number of nodes generated:
- Space complexity:
 - Same, if each node is retained in memory
- Optimality:
 - Does it always find the least-cost solution?
If depth = cost.

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

BFS evaluation

- Memory requirements are a bigger problem than the number of operations.
- Exponential complexity search problems cannot be solved by uninformed search methods for any but the smallest instances.

DEPTH	NODES	TIME	MEMORY
2	1100	0.11 seconds	1 megabyte
4	111100	11 seconds	106 megabytes
6	10^7	19 minutes	10 gigabytes
8	10^9	31 hours	1 terabyte
10	10^{11}	129 days	101 terabytes
12	10^{13}	35 years	10 petabytes
14	10^{15}	3523 years	1 exabyte

Time and memory requirements for BFS for $b=10$, 10,000 nodes/sec; 1000 bytes/node

Uniform cost search

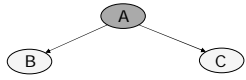
- Extension of BFS:
 - Expand node with *lowest path cost*
- Implementation: *fringe* = queue ordered by path cost.
- Same as BFS when all step-costs are equal.

Uniform cost search

- Completeness:
 - YES, if step-cost $> \epsilon$ (small positive constant)
 - What if some step costs are zero or negative?
- Time complexity:
 - Assume C^* the cost of the optimal solution.
 - Assume that every action costs at least ϵ
 - Worst-case: $O(b^{C^*/\epsilon})$
- Space complexity:
 - Same as time complexity
- Optimality:
 - nodes expanded in order of increasing path cost.
 - YES, if complete.

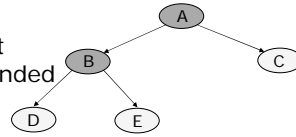
Depth First Search (DFS)

Expand deepest unexpanded node



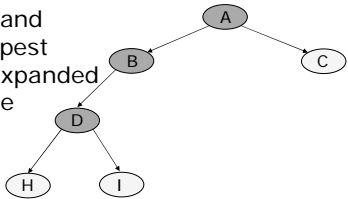
Depth First Search (DFS)

Expand deepest unexpanded node



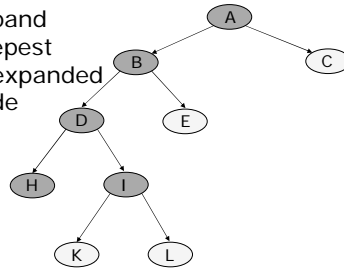
Depth First Search (DFS)

Expand deepest unexpanded node



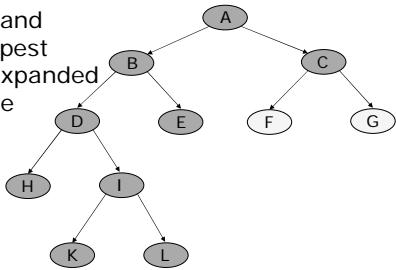
Depth First Search (DFS)

Expand deepest unexpanded node



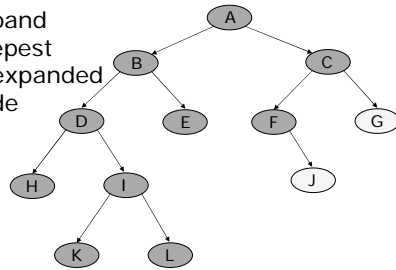
Depth First Search (DFS)

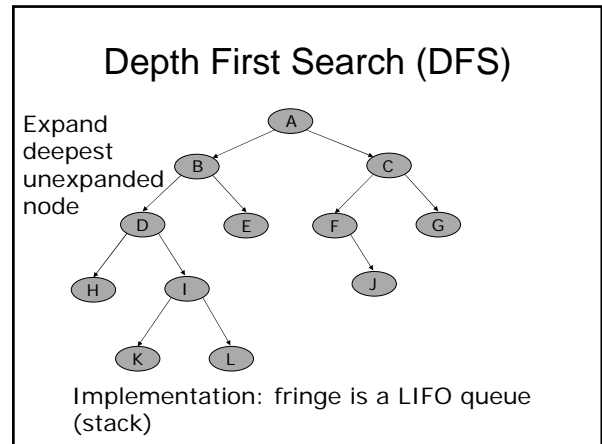
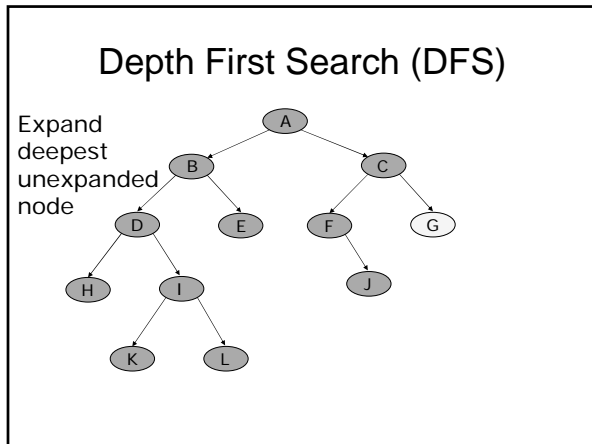
Expand deepest unexpanded node



Depth First Search (DFS)

Expand deepest unexpanded node





- ### DFS evaluation
- **Completeness:**
 - Does it always find a solution if one exists?
 - NO
 - unless search space is finite and no loops are possible.

- ### DFS evaluation
- **Completeness:**
 - NO unless search space is finite.
 - **Time complexity:**
 - Terrible if m (depth of search space) is much larger than d (depth of optimal solution)
 - But if many solutions, faster than BFS

- ### DFS evaluation
- **Completeness:**
 - NO unless search space is finite.
 - **Time complexity:** $O(b^m)$
 - **Space complexity:** $O(bm)$
 - Possible to use even less (expand one successor instead of all b).

- ### DFS evaluation
- **Completeness:**
 - NO unless search space is finite.
 - **Time complexity:** $O(b^m)$
 - **Space complexity:** $O(bm)$
 - **Optimality:** No
 - Same issues as completeness