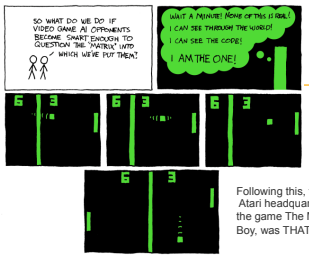
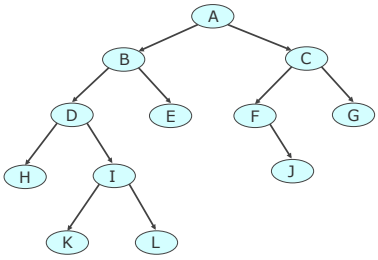
Uninformed Search



Russell and Norvig chap. 3

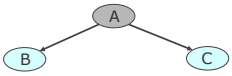
Following this, the pong paddle went on a mission to destroy Atari headquarters and, due to a mixup, found himself inside the game The Matrix Reloaded.
Boy, was THAT ever hard to explain to him.

Navigating through a search tree

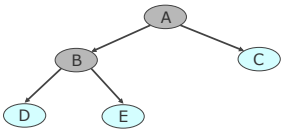


Navigating through a search tree

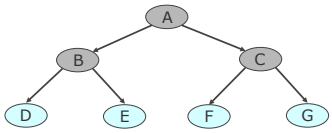
Navigating through a search tree



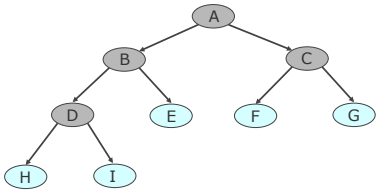
Navigating through a search tree



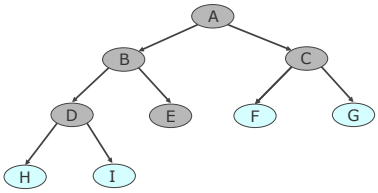
Navigating through a search tree



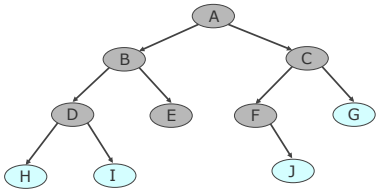
Navigating through a search tree



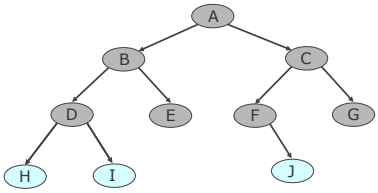
Navigating through a search tree



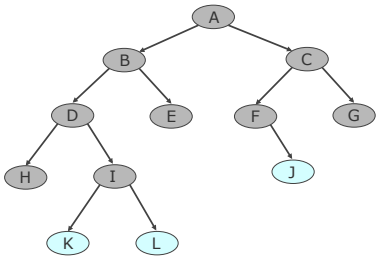
Navigating through a search tree



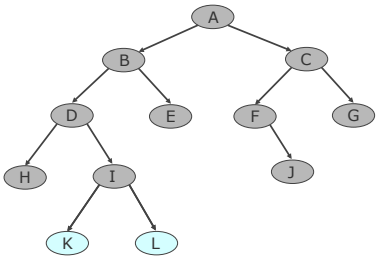
Navigating through a search tree



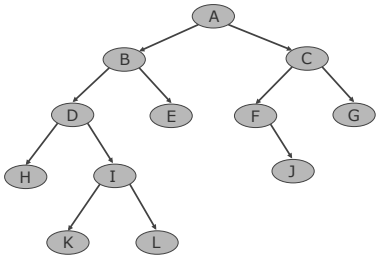
Navigating through a search tree



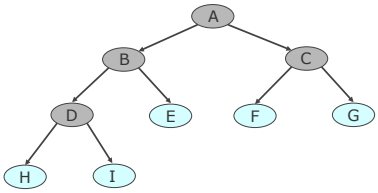
Navigating through a search tree



Navigating through a search tree



Unexpanded nodes: the frontier



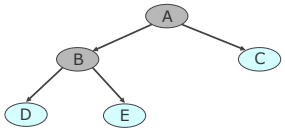
At every point in the search process we keep track of a list of nodes that haven't been expanded yet: the frontier

Tree search

Initial state

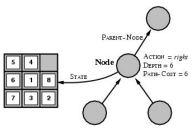
function TREE-SEARCH(*problem*) **return** a solution or failure
  Initialize frontier using the initial state of problem
  do
    if the frontier is empty **then return** *failure*
    choose leaf node from the frontier
    if node is a goal state **then return** *solution*
    else expand the node and add resulting nodes to the frontier

Tree search



function TREE-SEARCH(*problem*) **return** a solution or failure
  Initialize frontier using the initial state of problem
  do
    if the frontier is empty **then return** *failure*
    choose leaf node from the frontier
    if node is a goal state **then return** *solution*
    else expand the node and add resulting nodes to the frontier

What's in a node



- State
- Parent
- Action (the action that got us from the parent)
- Depth
- Path-Cost (total cost to get to the node)

Why do we need the parent and action information?

When the search graph is not a tree

- Need to avoid repeated states!
- Happens in problems with reversible operators
  Examples: missionaries and cannibals problem, sliding blocks puzzles, route finding problems.
- Detection: compare a node to be expanded to those already expanded.
- Increases memory requirements (especially for DFS): bounded by the size of the state space.

Graph Search

function GRAPH-SEARCH(*problem*) **return** a solution or failure
initialize the frontier using the initial state of problem
initialize the explored set to be empty
loop do
if the frontier is empty **then return** failure
choose a node from the frontier
if node is a goal state **then return** the corresponding solution
add the node to the explored set
expand the node, adding the resulting nodes to the frontier
(**only if not in the frontier or explored set**)

Algorithms that forget their history are doomed to repeat it

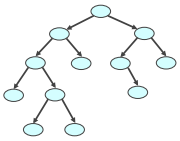
Search strategies

function GRAPH-SEARCH(*problem*) **return** a solution or failure
initialize the frontier using the initial state of problem
initialize the explored set to be empty
loop do
if the frontier is empty **then return** failure
choose a node from the frontier
if node is a goal state **then return** the corresponding solution
add the node to the explored set
expand the node, adding the resulting nodes to the frontier
(only if not in the frontier or explored set)

Search strategies differ in how a node is chosen from the frontier

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.
- Search algorithms are defined by the node expansion method:
  - Breadth-first search
  - Uniform-cost search
  - Depth-first search
  - Depth-limited search
  - Iterative deepening search.
  - Bidirectional search

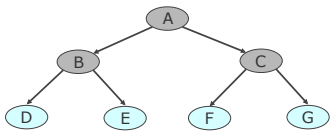


Metrics for comparing search strategies

- 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 ∞)*

Breadth-First Search (BFS)

- Expand all nodes at depth d before proceeding to depth d+1. Return the first goal node found
- Implementation: queue (FIFO).



Evaluation of BFS

- Completeness:
  - *Does it always find a solution if one exists?*

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:

$$b + b^2 + b^3 + \ldots + b^d = O(b^d)$$

Evaluation of BFS

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

$$b + b^2 + b^3 + \ldots + b^d = O(b^d)$$

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

Evaluation of BFS

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

$$b + b^2 + b^3 + \ldots + b^d = O(b^d)$$

- Space complexity:
  - Same, if each node is retained in memory
- Optimality:
  - *Does it always find the least-cost solution?*
    YES (if all actions have the same cost)

BFS evaluation

- Memory requirements are a bigger problem than its execution time.
- 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* = priority queue ordered by path cost.
- Same as BFS when all step-costs are equal.

Uniform cost search

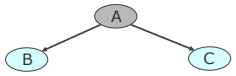
- Completeness:
  - YES
- Time complexity:
  - Assume C* the cost of the optimal solution.
  - Assume that every action costs at least ε
  - Worst-case:

$$O(b^{C^*/\varepsilon})$$

- Space complexity:
  - Same as time complexity
- Optimality:
  - YES (nodes expanded in order of increasing path cost)

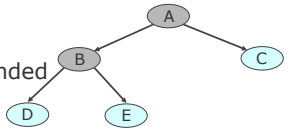
Depth First Search (DFS)

Expand deepest unexpanded node



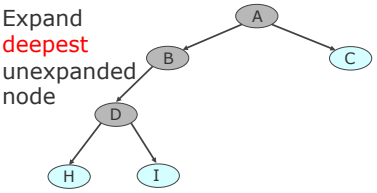
Depth First Search (DFS)

Expand deepest unexpanded node



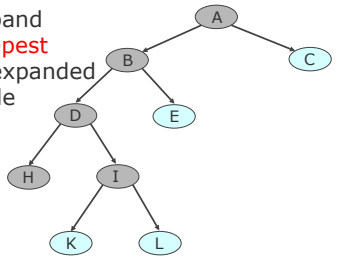
Depth First Search (DFS)

Expand deepest unexpanded node



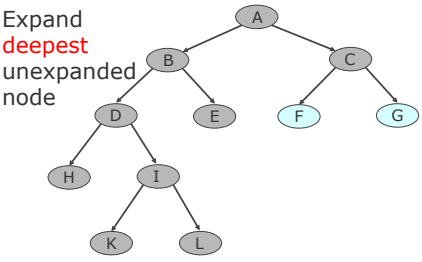
Depth First Search (DFS)

Expand deepest unexpanded node



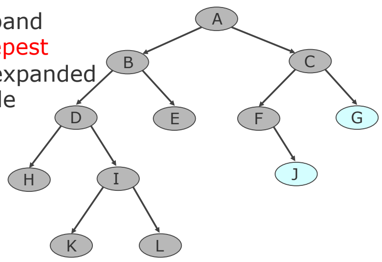
Depth First Search (DFS)

Expand deepest unexpanded node



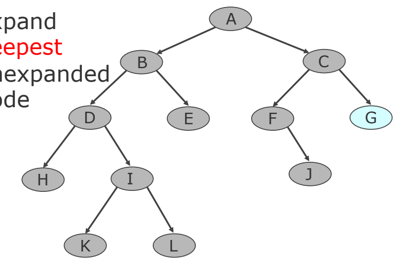
Depth First Search (DFS)

Expand
deepest
unexpanded
node



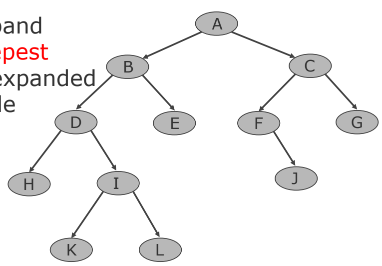
Depth First Search (DFS)

Expand
deepest
unexpanded
node



Depth First Search (DFS)

Expand
deepest
unexpanded
node



Implementation: fringe is a stack (LIFO)

DFS evaluation

- Completeness:
  - *Does it always find a solution if one exists?*

DFS evaluation

- Completeness:
  - *Does it always find a solution if one exists?*
    NO
    - *unless* search space is finite (also beware of loops if using graph search)

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, then 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: $O(bm)$
- Optimallity: No

Depth-limited search

- DFS with depth limit l.
  - Treat nodes at depth l as if they have no successors.
- Solves the infinite-path problem.
- If $l < d$ (depth of least cost solution) then incomplete.
- If $l > d$ then not optimal.
- Time complexity: $O(b^l)$
- Space complexity: $O(bl)$

Iterative Deepening Search (IDS)

- A strategy to find best depth limit l.
- Depth-Limited Search to depth 1, 2, …
- Expands from the root each time.
- Appears very wasteful, but combinatorics can be counter intuitive:

  N(DLS) = $b + b^2 + \ldots + b^{d-1} + b^d = O(b^d)$

  N(IDS) = $db + (d-1)b^2 + \ldots + 2b^{d-1} + b^d = O(b^d)$

  N(BFS) = $b + b^2 + \ldots + b^d = O(b^d)$

Iterative deepening search

function ITERATIVE_DEEPENING_SEARCH(*problem*) **return** a solution or failure

for *depth* = 0 to ∞ **do**
result ← DEPTH-LIMITED_SEARCH(*problem*, *depth*)
if *result* ≠ *cutoff* **then return** *result*

Note: depth-limited_search returns *cutoff* when it has reached the given depth without finding a solution

Evaluation of IDS

- Completeness:
  - YES (no infinite paths)

Evaluation of IDS

- Completeness:
  - YES
- Time complexity: $O(b^d)$

Evaluation of IDS

- Completeness:
  - YES
- Time complexity: $O(b^d)$
- Space complexity: $O(bd)$
  - Same as DFS

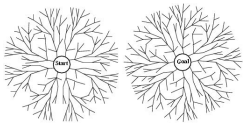
Evaluation of IDS

- Completeness:
  - YES
- Time complexity: $O(b^d)$
- Space complexity: $O(bd)$
  - Same as DFS
- Optimality:
  - YES if step cost is 1.

Iterative Deepening Search

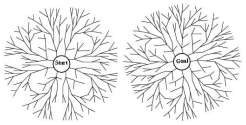
- Analogous to BFS: explores a complete layer of nodes before proceeding to the next one.
- Combines benefits of DFS and BFS.

Bidirectional Search



- Two simultaneous searches from start and goal.
  - Motivation: $b^{d/2} + b^{d/2}$ much less than b^d
- Before a node is expanded it is checked if it is in the fringe of the other search (can be done in constant time using a hash table).
- Need to keep at least one of the search trees in memory
- Time complexity: $O(b^{d/2})$.
- Space complexity: same.
- Complete and optimal (for uniform step costs) if both searches are BFS

Bidirectional Search



Issues in applying:

- The predecessor of each node should be efficiently computable.
  - When actions are easily reversible.
- Goal node: sometimes not unique or known explicitly (e.g. in chess).
- Memory consumption

Comparison of search strategies

Criterion	Breadth-First	Uniform-cost	Depth-First	Depth-limited	Iterative deepening	Bidirectional search
Complete?	YES*	YES*	NO	YES, if $l \geq d$	YES	YES*
Time	b^{d+1}	$b^{C^*/\varepsilon}$	b^m	b^l	b^d	$b^{d/2}$
Space	b^{d+1}	$b^{C^*/\varepsilon}$	bm	bl	bd	$b^{d/2}$
Optimal?	YES*	YES*	NO	NO	YES	YES