Uninformed search strategies use only information in problem definition
- Breadth-first search
- Depth-first search
- Depth-limited and iterative deepening search
A search strategy is defined by picking the order of node expansion – nodes are taken from the frontier.

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 $\infty$)
function TREE-SEARCH(\textit{problem}) \textbf{returns} a solution, or failure
initialize the frontier using the initial state of \textit{problem}
\textbf{loop do}
  \textbf{if} the frontier is empty \textbf{then return} failure
  choose a leaf node and remove it from the frontier
  \textbf{if} the node contains a goal state \textbf{then return} the corresponding solution
  expand the chosen node, adding the resulting nodes to the frontier

“Arad” is a repeated state!
Repeated states

- Failure to detect repeated states can turn a **linear** problem into an **exponential** one!
Augment TREE-SEARCH with a new data-structure:

- the explored set (closed list), which remembers every expanded node
- newly expanded nodes already in explored set are discarded
Breadth-first search

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

- Expand shallowest unexpanded node
- Implementation:
  - frontier is a FIFO queue, i.e., new successors go at end
Breadth-first search

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

- Expand shallowest unexpanded node
- **Implementation:**
  - frontier is a FIFO queue, i.e., new successors go at end
Breadth-first search algorithm

function BREADTH-FIRST-SEARCH(problem) returns a solution, or failure

node ← a node with State = problem.INITIAL-STATE, Path-Cost = 0
if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
frontier ← a FIFO queue with node as the only element
explored ← an empty set
loop do
  if EMPTY?(frontier) then return failure
  node ← POP(frontier)  /* chooses the shallowest node in frontier */
  add node.STATE to explored
  for each action in problem.ACTIONS(node.STATE) do
    child ← CHILD-NODE(problem, node, action)
    if child.STATE is not in explored or frontier then
      if problem.GOAL-TEST(child.STATE) then return SOLUTION(child)
    frontier ← INSERT(child, frontier)
Properties of breadth-first search

- **Complete?** Yes (if $b$ is finite)
- **Time?** $b + b^2 + b^3 + \ldots + b^d = O(b^d)$ (worst-case: regular $b$-ary tree of depth $d$)
- **Space?** $O(b^d)$ (keeps every node in memory)
- **Optimal?** Yes (if cost = 1 per step, then a *solution* is optimal if it is closest to the start node)

*Space* is the bigger problem (more than time)
Depth-first search

- Expand deepest unexpanded node

**Implementation:**
- frontier = LIFO queue, i.e., put successors at front
Depth-first search

- Expand deepest unexpanded node
- Implementation:
  - frontier = LIFO queue, i.e., put successors at front
Depth-first search

- Expand deepest unexpanded node
- **Implementation:**
  - frontier = LIFO queue, i.e., put successors at front
Depth-first search

- Expand deepest unexpanded node

**Implementation:**

- frontier = LIFO queue, i.e., put successors at front
Depth-first search

- Expand deepest unexpanded node
- Implementation:
  - frontier = LIFO queue, i.e., put successors at front
Depth-first search

- Expand deepest unexpanded node

**Implementation:**

- frontier = LIFO queue, i.e., put successors at front
Depth-first search

- Expand deepest unexpanded node

**Implementation:**

- frontier = LIFO queue, i.e., put successors at front
Depth-first search

- Expand deepest unexpanded node
- **Implementation:**
  - frontier = LIFO queue, i.e., put successors at front

![Depth-first search diagram]
Depth-first search

- Expand deepest unexpanded node

**Implementation:**
- frontier = LIFO queue, i.e., put successors at front
Depth-first search

- Expand deepest unexpanded node

**Implementation:**
- frontier = LIFO queue, i.e., put successors at front
Depth-first search

- Expand deepest unexpanded node
- **Implementation:**
  - frontier = LIFO queue, i.e., put successors at front
Depth-first search

- Expand deepest unexpanded node
- **Implementation:**
  - frontier = LIFO queue, i.e., put successors at front
Properties of depth-first search

- **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
Mid-Lecture Problem

- Compare breadth-first and depth-first search.
  - When would breadth-first be preferable?
  - When would depth-first be preferable?
Breadth-First:
- When completeness is important.
- When optimal solutions are important.

Depth-First:
- When solutions are dense and low-cost is important, especially space costs.
Depth-limited search

This is depth-first search with depth limit \( l \), i.e., nodes at depth \( l \) have no successors

**Recursive implementation:**

```plaintext
function DEPTH-LIMITED-SEARCH(problem, limit) returns a solution, or failure/cutoff
   return RECURSIVE-DLS(MAKE-NODE(problem.INITIAL-STATE), problem, limit)

function RECURSIVE-DLS(node, problem, limit) returns a solution, or failure/cutoff
   if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
   else if limit = 0 then return cutoff
   else
      cutoff_occurred? ← false
      for each action in problem.ACTIONS(node.STATE) do
         child ← CHILD-NODE(problem, node, action)
         result ← RECURSIVE-DLS(child, problem, limit - 1)
         if result = cutoff then cutoff_occurred? ← true
         else if result ≠ failure then return result
      if cutoff_occurred? then return cutoff else return failure
```

20/01/2017 Michael Herrmann Inf2D 3
Iterative deepening search

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 search \( l = 0 \)
Iterative deepening search \( l = 0 \)
Iterative deepening search $l = 0$

Limit = 2

Diagram showing the iterative deepening search process with limit 2.
Iterative deepening search $l = 0$
Iterative deepening search

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

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

Some cost associated with generating upper levels multiple times

Example: For $b = 10$, $d = 5$,

- $N_{BFS} = 10 + 100 + 3,000 + 10,000 + 100,000 = 111,110$
- $N_{IDS} = 50 + 400 + 3,000 + 20,000 + 100,000 = 123,450$

Overhead $= (123,450 - 111,110)/111,110 = 11\%$
Properties of iterative deepening search

- Complete? Yes
- Time? \( (d)b + (d - 1)b^2 + \ldots + (1)b^d = O(b^d) \)
- Space? \( O(bd) \)
- Optimal? Yes, if step cost = 1
Uniform cost search (UCS)

Step costs are not uniform.

Details: home work.
### Summary of Algorithms

<table>
<thead>
<tr>
<th>Criterion</th>
<th>Breadth-First</th>
<th>Uniform-Cost</th>
<th>Depth-First</th>
<th>Depth-Limited</th>
<th>Iterative Deepening</th>
</tr>
</thead>
<tbody>
<tr>
<td>Complete?</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Time</td>
<td>$O(b^d)$</td>
<td>$O(b^{\lceil C^*/\epsilon \rceil})$</td>
<td>$O(b^m)$</td>
<td>$O(b^l)$</td>
<td>$O(b^d)$</td>
</tr>
<tr>
<td>Space</td>
<td>$O(b^d)$</td>
<td>$O(b^{\lceil C^*/\epsilon \rceil})$</td>
<td>$O(bm)$</td>
<td>$O(bl)$</td>
<td>$O(bd)$</td>
</tr>
<tr>
<td>Optimal?</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Variety of *uninformed* search strategies:
- breadth-first
- depth-first
- depth limited
- iterative deepening

Iterative deepening search uses only linear space and not much more time than other uninformed algorithms.