Artificial Intelligence
For HEDSPI Project
Lecturer 3 - Search

Lecturers:
Le Thanh Huong
Tran Duc Khanh
Dept of Information Systems
Faculty of Information Technology - HUT

Outline
- Problem-solving agents
- Problem types
- Problem formulation
- Example problems
- Basic search algorithms
  - breadth-first search
  - depth-first search
  - depth-limited search
  - iterative deepening depth-first search

Problem-solving agents

```
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
```

Example 1: Route Planning

- Performance: Get from Arad to Bucharest as quickly as possible
- Environment: The map, with cities, roads, and guaranteed travel times
- Actions: Travel a road between adjacent cities
Example 2: Finding letters

- Replace letters by numbers from 0 to 9 such as no different letter is replaced by the same number and satisfying the following constraint:
  
  \[
  \begin{array}{cc}
  \text{SEND} & \text{CROSS} \\
  + \text{MORE} & + \text{ROADS} \\
  \text{MONEY} & \text{DANGER}
  \end{array}
  \]

Example 3: Pouring water

- Given 2 containers A(m litres), B(n litres). Finding a method to measure \( k \) litres (\( k \leq \max(m,n) \)) by 2 containers A, B and a container C
- Actions (how):
  
  \[
  C \rightarrow A; C \rightarrow B; A \rightarrow B; A \rightarrow C; B \rightarrow A; B \rightarrow C
  \]
- Conditions: no overflow, pouring all water
- Eg: \( m = 5, n = 6, k = 2 \) (what)
- Mathematical model:
  
  \[
  (x, y) \rightarrow (x', y')
  \]

Example 4: The 8-puzzle

- Trong bảng ô vuông \( n \) hàng, \( n \) cột, mỗi ô chứa 1 số nằm trong phạm vi từ 1 \( \rightarrow \) \( n^2 \)-1 sao cho không có 2 ô có cùng giá trị. Còn đúng 1 ô bị tổ. Xuat phát từ 1 cách sắp xếp nào đó của các độ của các số trong bảng, hãy dịch chuyển các ô trùng sang phải, sang trái, lên trên, xuống dưới để đưa về bảng:

```
    7 2 4
  5 6
  8 3 1
```
```
    1 2
  3 4 5
  6 7 8
```

Example 5: Hà Nội tower

- Cho 3 cọc 1,2,3. Ô cọc 1 ban đầu có \( n \) đĩa, sắp theo thứ tự to dán từ trên xuống dưới. Hãy tìm cách chuyển \( n \) đĩa đến sang cọc 3 sao cho:
  - Mỗi lần chỉ chuyển 1 đĩa
  - Ô mỗi cọc không cho phép đĩa to nằm trên đĩa con

```
    1 2 3
  4 5 6
  7 8
```

```
    1 2 3
  1
  2
  3
```
Problem types

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

Search Problem Definition

A problem is defined by four items:
1. initial state: e.g., Arad
2. actions or successor function \( S(x) \) = set of action-state pairs
   - e.g., \( S(Arad) = \{ <Arad \rightarrow Zerind, Zerind>, \ldots \} \)
3. goal test, can be
   - explicit, e.g., \( x = \text{Bucharest} \)
   - implicit, e.g., \( \text{Checkmate}(x) \)
4. path cost (additive)
   - e.g., sum of distances, number of actions executed, etc.
   - \( c(x,a,y) \) is the step cost, assumed to be \( \geq 0 \)

- A solution is a sequence of actions leading from the initial state to a goal state

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

Search tree

- Search trees:
  - Represent the branching paths through a state graph.
  - Usually much larger than the state graph.
  - Can a finite state graph give an infinite search tree?
Tree and graph

B is parent of C
C is child of B
A is ancestor of C
C is descendant of A

Convert from search graph to search tree

- We can turn graph search problems into tree search problems by:
  - replacing undirected links by 2 directed links
  - avoiding loops in path (or keeping track of visited nodes globally)

Tree search algorithms

- Basic idea:
  - 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
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 ← Insert-All(Expand(node, problem), fringe)
end loop

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 Expand function creates new nodes, filling in the various 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 $\infty$)
Uninformed search strategies

- **Uninformed search strategies** use only the information available in the problem definition.

- **Breadth-first search**
  - Expand shallowest unexpanded node
  - fringe = queue (FIFO)

- **Depth-first search**
  - Expand deepest unexpanded node
  - fringe = stack (LIFO)

- **Depth-limited search**: depth-first search with depth limit
- Iterative deepening search

---

**Breadth-first search**

- Expand shallowest unexpanded node

---

**Breadth-first search (con’t)**

- **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)

---

**Depth-first search**

- Expand deepest unexpanded node
Depth-first search (con't)

- **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 can get stuck on infinite path when a different choice would lead to a solution
  ⇒ Depth-limited search = depth-first search with depth limit $l$,
  i.e., nodes at depth $l$ have no successors

```
function DEPTH-LIMITED-SEARCH(problem, limit) returns solution/fail/cutoff
    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 search

- Problem with depth-limited search: if the shallowest goal is beyond the depth limit, no solution is found.
  ⇒ Iterative deepening search:
  1. Do a DFS which only searches for paths of length 1 or less. (DFS gives up on any path of length 2)
  2. If "1" failed, do a DFS which only searches paths of length 2 or less.
  3. If "2" failed, do a DFS which only searches paths of length 3 or less.
  4. ... and so on.

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

8-puzzle game with depth limit $l = 5$
Iterative deepening search (con't)

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

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

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

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

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 search

- **Complete?** Yes
- **Time?** $(d+1)b^0 + db^1 + (d-1)b^2 + \ldots + b^d = O(b^d)$
- **Space?** $O(bd)$
- **Optimal?** Yes, if step cost = 1
## 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>$O(b^{d+1})$</td>
<td>$O(b^{C'/c_1})$</td>
<td>$O(b^m)$</td>
<td>$O(b^l)$</td>
<td>$O(b^d)$</td>
</tr>
<tr>
<td>Space</td>
<td>$O(b^{d+1})$</td>
<td>$O(b^{C'/c_1})$</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>