Formalizing the State Space - 1

- A **state space** is a *graph*, (V, E), where V is a set of **nodes** and E is a set of **arcs**, where each arc is *directed* from a node to another node.
- Each **node** is a data structure, represents a particular state.
- Each **arc** corresponds to an instance of one of the operators. When the operator is applied to the state associated with the arc's source node, then the resulting state is the state associated with the arc's destination node.
- Each arc has a fixed, positive cost associated with it corresponding to the cost of the operator.
Formalizing the State Space - 2

- Each node has a set of successor nodes = all legal operators.
- One or more nodes are designated as start nodes.
- A goal test predicate is applied to a state to determine if its associated node is a goal node.
- A solution is a sequence of operators that is associated with a path in a state space from a start node to a goal node.
- The cost of a solution is the sum of the arc costs on the solution path.
- State-space search is searching through a state space for a solution by making explicit a sufficient portion of an implicit state-space graph to include a goal node. Hence, initially V={S}, where S is the start node; when S is expanded and so forth until …
- Each node implicitly or explicitly represents a partial solution path (and cost of the partial solution path) from the start node.
Basic Search Algorithm

**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] applied to STATE(node) succeeds return 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
Uninformed Search

*Uninformed* strategies use only the information available in the problem definition

Breadth-first search

Uniform-cost search

Depth-first search

Depth-limited search

Iterative deepening search
Simple Example - 1

S ... Initial State
   /\   \\
  1/ 5 \8
  /  \\ /
 A B C
   /\   / \\
 3/ 7 9 4 /5
  /  \\ /
 D E G .... Goal State

Nodes expanded by:

- Depth-First Search: S A D E G
  Solution found: S A G

- Breadth-First Search: S A B C D E G
  Solution found: S A G

- Uniform-Cost Search: S A D B C E G
  Solution found: S B G
  This is the only uninformed search that worries about costs.

- Iterative-Deepening Search: S A B C S A D E G
  Solution found: S A G
Simple Example - 2

Depth first LIFO queue

expanded
node          nodes list
-------------        -------------
{ S }
S               { A B C }
A               { D E G B C }
D               { E G B C }
E               { G B C }
G               { B C }

Uniform cost ordered queue

expanded
node          nodes list
-------------        -------------
{ S }
S               { A(1) B(5) C(8) }
A               { D(4) B(5) C(8) E(8) G(10) }
D               { B(5) C(8) E(8) G(10) }
B               { C(8) E(8) G(9) G(10) }
C               { E(8) G(9) G(10) G(13) }
E               { G(9) G(10) G(13) }
G               { }

Breadth first FIFO queue

expanded
node          nodes list
-------------        -------------
{ S }
S               { A B C }
A               { B C D E G }
B               { C D E G G' }
C               { D E G G' G" }
D               { E G G' G" }
E               { G G' G" }
G               { G' G" }

CSI 535 - Introduction to A.I. 6
Stefan’s Question

- Breadth first complexity $O(b^d)$
- Uniform cost complexity $O(b^{\lceil C*/\epsilon \rceil})$
- Is UCS always better than BFS?
- Recall the nodes are placed on the queue according to cumulative path costs
- If each and every action is a constant cost then $d = \lceil C*/\epsilon \rceil$
- Otherwise for UCS we order the fringe according to costs which will dominate BFS
Classic Problem #1: Eight Queen Problem
K-Queens Problem as CSP

Assume one queen in each column. Which row does each one go in?

**Variables** $Q_1, Q_2, Q_3, Q_4$

**Domains** $D_i = \{1, 2, 3, 4\}$

**Constraints**

$Q_i \neq Q_j$ (cannot be in same row)

$|Q_i - Q_j| \neq |i - j|$ (or same diagonal)

Translate each constraint into set of allowable values for its variables

E.g., values for $(Q_1, Q_2)$ are $(1, 3) \ (1, 4) \ (2, 4) \ (3, 1) \ (4, 1) \ (4, 2)$
9. Consider the classic farmer, fox, goose and grain problem. The farmer wants to move himself, the fox, the goose and the edible grain from the west to the east side of the river. Only he can row his small boat across the river and he can only take along with himself either the fox, the goose or the grain. That is, he can only take one of his items with him. If the fox is left with the goose, the goose will be eaten. If the goose is left with the grain, the grain will be eaten. You are to \textbf{formally} formulate this situation as a state space search problem. For example, you must state what variables comprise your state space.

a) Describe the representation of the state space and the goal test
b) Describe the operators that move from one state to another.

c) \textbf{Formally} specify the constraints that are applicable for each operator.
d) Describe a non-trivial admissible heuristic that is usable with the A* informed search algorithm.
More Classic Problems

- **Missionaries and Cannibals**
  There are 3 missionaries, 3 cannibals, and 1 boat that can carry up to two people on one side of a river. Goal: Move all the missionaries and cannibals across the river. Constraint: Missionaries can never be outnumbered by cannibals on either side of the river, or else the missionaries are killed. State = configuration of missionaries and cannibals and boat on each side of the river. Operators: Move boat containing some set of occupants across the river (in either direction) to the other side.

- **Remove 5 Sticks**
  Given the following configuration of sticks, remove exactly 5 sticks in such a way that the remaining configuration forms exactly 3 squares.

```
  - -
  | | |
  | | |
  | | |
  - -
```

- **Water Jug Problem**
  Given a 5-gallon jug and a 2-gallon jug, with the 5-gallon jug initially full of water and the 2-gallon jug empty, the goal is to fill the 2-gallon jug with exactly one gallon of water.
Water Jug Solution

- State = (x,y), where x = number of gallons of water in the 5-gallon jug and y is gallons in the 2-gallon jug
- Initial State = (5,0)
- Goal State = (*,1), where * means any amount
- Operators
  - (x,y) -> (0,y) ; empty 5-gal jug
  - (x,y) -> (x,0) ; empty 2-gal jug
  - (x,2) and x<=3 -> (x+2,0) ; pour 2-gal into 5-gal
  - (x,0) and x>=2 -> (x-2,2) ; pour 5-gal into 2-gal
  - (1,0) -> (0,1) ; empty 5-gal into 2-gal
- State Space (also called the Problem Space)

```
(5,0) = Start
    / \    
(3,2) (0,0)
    / \    
(3,0) (0,2)
    / /    
(1,2) (0,2)
    / /    
(1,0) (0,2)
    / /    
(0,1) = Goal
```
Uniform Cost Search
Next Class

• This class uninformed search
• Next classes are informed (heuristic) search and constraint satisfaction problems (Chapter 4, R&N)