

Tutorial 2

- Feedback on submissions
 - Question 2: Representations
 - Question 4: Pseudo Code

Representations

- Should be as formal as possible
- Goal is to need as few assumptions as possible
- A good representation will make transitions easy to implement
- Constraints on the states also need to be considered
 - Checked during transitions
 - Check before or after transitions
 - Make invalid states infinitely bad
- Note that representation refers to the states of the problem, not the entire state space of the problem

Towers of Hanoi: Stacks

- Each pole is a stack
- $P[0]$, $P[1]$, $P[2]$
- Each stack is a LIFO queue with two operations:
 $\text{Push}(S,i)/\text{Pop}(S)$
- Towers of Hanoi translates to stacks nicely
- Makes some assumptions implicit: can't move a disc unless it's on the top of a stack
- Still need to define the constraint that disc can't be moved onto a smaller disc

Towers of Hanoi: Arrays

- A 3x3 array
- `Disks[Pn][Dn]`
- But you have to define the two stack operations on this array, as well as defining the constraint that disc can't be moved onto a smaller disc

Towers of Hanoi: Indexes

- An index is stored for each disc, determining the stack that each disc is on
- D0, D1, D2
- Lower space requirements than the other methods
- Much more computation required to determine legal moves

Pseudo Code

- Define function name and description
- Define input/output parameters and descriptions
- Define Subroutine functions names and descriptions
- States lists of actions to perform
- Use indentation to define blocks corresponding to flow chart boxes

Functions

- Goal test
 - Is the current state the goal state?
- Successor
 - What states can you get to from the current state?
- Update (The search algorithm)
 - What is a path from the initial state to the goal state?

Goal Test

Towers of Hanoi: Stack Representation

- Determine whether the state is the goal state
- Input is the state
- Output is whether it is the goal state

function Goal-test(*state*) **returns** *succeeds* or *failure*
 if state[0]=[] and state[1]=[] and state[2]=[1,2,3] **return** *succeeds*
 else return *failure*

Successor Function

Towers of Hanoi: Stack Representation

- Determine the legal states that can be reached from the current state
- Input is the current state
- Output is the set of new states

```
function Expand(node) return nodes
    T = []
    for m = 0 to 2:
        for n = 0 to 2:
            P = State(node)
            if n ≠ m and P[m] is not empty and (P[n] is empty or topDisk(P[n] > topDisk(P[m]))
            then Push(P[n], Pop(P[m])); add(Make-Node(P), T)
    return T
```

Update Process: Breadth First Search

- Implement the Breadth First Search strategy:
 1. Set L to be a list of the initial nodes in the problem
 2. Let n be the first node on L . If L is empty, fail.
 3. If n is a goal state, stop and return it and the path from the initial node to n .
 4. Otherwise, remove n from L and add to the end of L all of n 's children, labeling each with its path from the initial node. Return to step 2.

Update Process: Breadth First Search

```
function BFS(L) returns path
  if (not-empty(L))
    n = L(0)
    if (Goal-test(n))
      return path
    else
      expanded_nodes = Expand (n)
      // remove n from start of L, put expanded nodes at end of L
      L = [L(1:end) expanded_nodes]
  else
    return failure
```

Update Process: Depth First Search

- Implement the Depth First Search strategy:
 1. Set L to be a list of the initial nodes in the problem.
 2. Let n be the first node on L . If L is empty, fail
 3. If n is a goal state, stop and return it and the path from the initial node to n .
 4. Otherwise, remove n from L and add to the front of L all of n 's children, labeling each with its path from the initial node. Return to step 2.

Update Process: Depth First Search

```
function BFS(L) returns path
  if (not-empty(L))
    n = L(0)
    if (goal-test(n))
      return path
    else
      expanded_nodes = successor-function(n)
      // remove n from start of L, put expanded nodes at start of L
      L = [expanded_nodes L(1:end)]
  else
    return failure
```