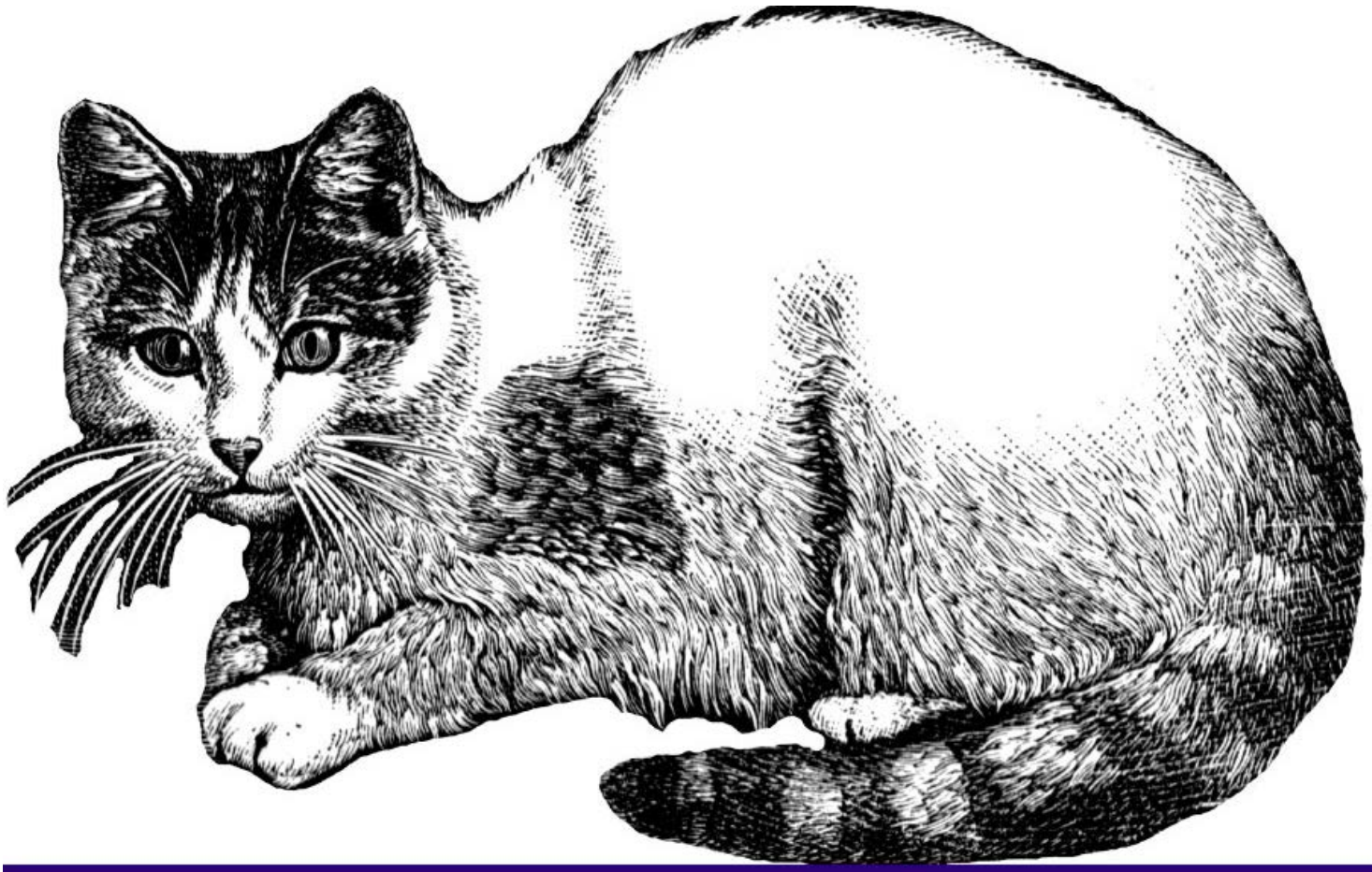

State Space Problem Solving

Prolog Programming Assignment 2

Miguel Cruz

Prolog Programming Assignment #2: State Space Problem Solving

Abstract

This assignment leads you through the development of a state space problem solver for the Towers of Hannoi problem in Prolog.

Task 3: One Move Predicate and a Unit Test

State Space Operator Implementation

```
m12([Tower1Before,Tower2Before,Tower3],[Tower1After,Tower2After,Tower3]):-
        Tower1Before =[H|T],
        Tower1After = T,
        Tower2Before = L,
        Tower2After = [H|L].
```

Unit Test Code

```
test__m12 :-
        write('Testing: move_m12\n'), TowersBefore =
        [[t,s,m,l,h],[],[]],
        trace('','TowersBefore',TowersBefore),
        m12(TowersBefore,TowersAfter),
        trace('','TowersAfter',TowersAfter).
```

Unit Test Demo

```
?- test__m12.
Testing: move_m12
TowersBefore =
[[t,s,m,l,h],[],[]]
TowersAfter = [[s,m,l,h],
[t],[]] true.
```

Task 4: The Remaining Five Move Predicates and a Unit Tests

State Space Operator Implementation

```
m12([Tower1Before,Tower2Before,Tower3],[Tower1After,Tower2After,Tower3]):-
        Tower1Before =[H|T],
        Tower1After = T,
        Tower2Before = L,
        Tower2After = [H|L].


m13([Tower1Before,Tower2,Tower3Before],[Tower1After,Tower2,Tower3After]):-
        Tower1Before = [H|T],
        Tower1After = T,
        Tower3Before = L,
        Tower3After = [H|L].


m23([Tower1,Tower2Before,Tower3Before],[Tower1,Tower2After,Tower3After]):-
        Tower2Before = [H|T],
        Tower2After = T,
        Tower3Before = L,
        Tower3After = [H|L].


m21([Tower1Before,Tower2Before,Tower3],[Tower1After,Tower2After,Tower3]):-
        Tower2Before = [H|T],
        Tower2After = T,
        Tower1Before= L,
        Tower1After = [H|L].


m31([Tower1Before,Tower2,Tower3Before],[Tower1After,Tower2,Tower3After]):-
        Tower3Before = [H|T],
        Tower3After = T,
        Tower1Before = L,
        Tower1After = [H|L].


m32([Tower1,Tower2Before,Tower3Before],[Tower1,Tower2After,Tower3After]):-
        Tower3Before = [H|T],
        Tower3After = T,
        Tower2Before = L,
        Tower2After = [H|L].
```

Unit Test Code

```
test__m12 :-
        write('Testing: move_m12\n'),
        TowersBefore = [[t,s,m,l,h],[],[]],
        trace('','TowersBefore',TowersBefore),
        m12(TowersBefore,TowersAfter),
        trace('','TowersAfter',TowersAfter).



test__m13 :-
        write('Testing: move_m13\n'),
        TowersBefore = [[t,s,m,l,h],[],[]],
        trace('','TowersBefore',TowersBefore),
        m13(TowersBefore,TowersAfter),
        trace('','TowersAfter',TowersAfter).



test__m21 :-
        write('Testing: move_m21\n'),
        TowersBefore = [[],[t,s,m,l,h],[]],
        trace('','TowersBefore',TowersBefore),
        m21(TowersBefore,TowersAfter),
        trace('','TowersAfter',TowersAfter).



test__m23 :-
        write('Testing: move_m23\n'),
        TowersBefore = [[],[t,s,m,l,h],[]],
        trace('','TowersBefore',TowersBefore),
        m23(TowersBefore,TowersAfter),
        trace('','TowersAfter',TowersAfter).



test__m31 :-
        write('Testing: move_m31\n'),
        TowersBefore = [[],[],[t,s,m,l,h]],
        trace('','TowersBefore',TowersBefore),
        m31(TowersBefore,TowersAfter),
        trace('','TowersAfter',TowersAfter).



test__m32 :-
        write('Testing: move_m32\n'),
        TowersBefore = [[],[],[t,s,m,l,h]],
        trace('','TowersBefore',TowersBefore),
        m32(TowersBefore,TowersAfter),
        trace('','TowersAfter',TowersAfter).
```

Unit Test Demo

```
?- test__m12.
Testing: move_m12 TowersBefore =
[[t,s,m,l,h],[],[]] TowersAfter =
[[s,m,l,h],[t],[]] true.

?- test__m21.
Testing: move_m21 TowersBefore =
[[],[t,s,m,l,h],[]] TowersAfter =
[[t],[s,m,l,h],[]] true.

?- test__m13.
Testing: move_m13 TowersBefore =
[[t,s,m,l,h],[],[]] TowersAfter =
[[s,m,l,h],[],[t]] true.

?- test__m31.
Testing: move_m31 TowersBefore =
[[],[],[t,s,m,l,h]] TowersAfter =
[[t],[],[s,m,l,h]] true.

?- test__m23.
Testing: move_m23 TowersBefore =
[[],[t,s,m,l,h],[]] TowersAfter =
[[],[s,m,l,h],[t]] true.

?- test__m32.
Testing: move_m32 TowersBefore =
[[],[],[t,s,m,l,h]] TowersAfter =
[[],[t],[s,m,l,h]] true.
```

Task 5: Valid State Predicate and Unit Test

```
% --- valid_state(S) :: S is a valid

state valid_state(S):-


        S= [T1,T2,T3],
        valid_tower(T1),
        valid_tower(T2),
        valid_tower(T3).
        valid_tower(T):-
        length(T,L1),
        L1<2.




valid_tower(T):-

        T=[A1|[A2|_]],
        before(A1,A2).


%--before(A,B) :A comes before B on a valid stack. Helper
function for valid_state(S).
before(t,s).
before(t,m).
before(t,l).
before(t,h).
before(s,m).
before(s,l).
before(s,h).
before(m,l).
before(m,h).
before(l,h).
```

Unit Test Program Demo

```
test__valid_state :-

        write('Testing: valid_state
        \n'), test__vs([[l,t,s,m,h],
        [],[]]),
        test__vs([[t,s,m,l,h],[],[]]),
        test__vs([[],[h,t,s,m],[l]]),
        test__vs([[],[t,s,m,h],[l]]),
        test__vs([[],[h],[l,m,s,t]]),
        test__vs([[],[h],[t,s,m,l]]).
```

Unit Test Program

```
?- test__valid_state.
Testing: valid_state
[[l,t,s,m,h],[],[]] is
invalid.[[t,s,m,l,h],[],
[]] is valid.[[],
[h,t,s,m],[l]] is invalid.
[[],[t,s,m,h],[l]] is
valid.[[],[h],[l,m,s,t]]
is invalid.[[],[h],
[t,s,m,l]] is valid.
true .
```

Task 6: Defining the write_sequence predicate

```
% --- write_sequence_reversed(S) :: Write the sequence, given by
S,
% --- expanding the tokens into meaningful strings.
write_solution(S) :-
nl, write('Solution ...'), nl, nl,
reverse(S,R),
write_sequence(R),nl.
write_sequence(L):-
L=[H|T],
T=[],
write_move(H).
write_sequence(L):-
L=[H|T],
write_move(H),
write_sequence(T).
write_move(T):-
T=m12, write('Transfer a disk from tower 1 to tower 2.'),nl.
write_move(T):-
T=m13, write('Transfer a disk from tower 1 to tower 3.'),nl.
write_move(T):-
T=m21, write('Transfer a disk from tower 2 to tower 1.'),nl.
write_move(T):-
T=m23, write('Transfer a disk from tower 2 to tower 3.'),nl.
write_move(T):-
T=m31, write('Transfer a disk from tower 3 to tower 1.'),nl.
write_move(T):-
T=m32, write('Transfer a disk from tower 3 to tower 2.'),nl.
```

Unit Test Program Code

```
test__write_sequence :-
write('First test of write_sequence ...'), nl,
write_sequence([m31,m12,m13,m21]),
write('Second test of write_sequence ...'), nl,
write_sequence([m13,m12,m32,m13,m21,m23,m13]).
```

Unit Test Program Demo

```
?- test__write_sequence.
First test of write_sequence ...
Transfer a disk from tower 3 to tower 1.
Transfer a disk from tower 1 to tower 2.
Transfer a disk from tower 1 to tower 3.
Transfer a disk from tower 2 to tower 1.
Second test of write_sequence ...
Transfer a disk from tower 1 to tower 3.
Transfer a disk from tower 1 to tower 2.
Transfer a disk from tower 3 to tower 2.
Transfer a disk from tower 1 to tower 3.
Transfer a disk from tower 2 to tower 1.
Transfer a disk from tower 2 to tower 3.
Transfer a disk from tower 1 to tower 3.
true .
```

Task 7: Run the program to solve the 3 disk problem

```
 ?- solve.
PathSoFar = [[[s,m,l],[],[]]]
Move = m12
NextState = [[m,l],[s],[]]
PathSoFar = [[[s,m,l],[],[]],[[m,l],[s],[]]]
Move = m12
NextState = [[l],[m,s],[]]
Move = m13
NextState = [[l],[s],[m]]
PathSoFar = [[[s,m,l],[],[]],[[m,l],[s],[]],[[l],[s],[m]]] Move =
m12
NextState = [[],[l,s],[m]]
Move = m13
NextState = [[],[s],[l,m]]
Move = m21
NextState = [[s,l],[],[m]]
PathSoFar =
[[[s,m,l],[],[]],[[m,l],[s],[]],[[l],[s],[m]],[[s,l],[],[m]]]
Move = m12
NextState = [[l],[s],[m]]
Move = m13
NextState = [[l],[],[s,m]]
PathSoFar =
[[[s,m,l],[],[]],[[m,l],[s],[]],[[l],[s],[m]],[[s,l],[],[m]],
[[l ],[],[s,m]]]
Move = m12
NextState = [[],[l],[s,m]]
PathSoFar =
[[[s,m,l],[],[]],[[m,l],[s],[]],[[l],[s],[m]],[[s,l],[],[m]],
[[l ],[],[s,m]],[[],[l],[s,m]]]
Move = m21
NextState = [[l],[],[s,m]]
Move = m23
NextState = [[],[],[l,s,m]]
Move = m31
NextState = [[s],[l],[m]]
PathSoFar =
[[[s,m,l],[],[]],[[m,l],[s],[]],[[l],[s],[m]],[[s,l],[],[m]],
[[l ],[],[s,m]],[[],[l],[s,m]],[[s],[l],[m]]]
Move = m12
NextState = [[],[s,l],[m]]
PathSoFar =
[[[s,m,l],[],[]],[[m,l],[s],[]],[[l],[s],[m]],[[s,l],[],[m]],
[[l ],[],[s,m]],[[],[l],[s,m]],[[s],[l],[m]],[[],[s,l],[m]]]
Move = m21
NextState = [[s],[l],[m]]
Move = m23
NextState = [[],[l],[s,m]]
Move = m31
NextState = [[m],[s,l],[]]
PathSoFar =
[[[s,m,l],[],[]],[[m,l],[s],[]],[[l],[s],[m]],[[s,l],[],[m]],
[[l ],[],[s,m]],[[],[l],[s,m]],[[s],[l],[m]],[[],[s,l],[m]],[[m],
[s, l],[]]]
Move = m12
NextState = [[],[m,s,l],[]]
Move = m13
NextState = [[],[s,l],[m]]
Move = m21
NextState = [[s,m],[l],[]]
PathSoFar =
[[[s,m,l],[],[]],[[m,l],[s],[]],[[l],[s],[m]],[[s,l],[],[m]],
[[l ],[],[s,m]],[[],[l],[s,m]],[[s],[l],[m]],[[],[s,l],[m]],[[m],
[s, l],[]],[[s,m],[l],[]]]
Move = m12
NextState = [[m],[s,l],[]]
Move = m13
NextState = [[m],[l],[s]]
PathSoFar =
[[[s,m,l],[],[]],[[m,l],[s],[]],[[l],[s],[m]],[[s,l],[],[m]],
[[l ],[],[s,m]],[[],[l],[s,m]],[[s],[l],[m]],[[],[s,l],[m]],[[m],
[s, l],[]],[[s,m],[l],[]],[[m],[l],[s]]]
Move = m12
NextState = [[],[m,l],[s]]
PathSoFar =
[[[s,m,l],[],[]],[[m,l],[s],[]],[[l],[s],[m]],[[s,l],[],[m]],
[[l ],[],[s,m]],[[],[l],[s,m]],[[s],[l],[m]],[[],[s,l],[m]],[[m],
[s, l],[]],[[s,m],[l],[]],[[m],[l],[s]],[[],[m,l],[s]]]
Move = m21
NextState = [[m],[l],[s]]
Move = m23
NextState = [[],[l],[m,s]]
Move = m31
NextState = [[s],[m,l],[]]
PathSoFar =
[[[s,m,l],[],[]],[[m,l],[s],[]],[[l],[s],[m]],[[s,l],[],[m]],
[[l ],[],[s,m]],[[],[l],[s,m]],[[s],[l],[m]],[[],[s,l],[m]],[[m],
[s, l],[]],[[s,m],[l],[]],[[m],[l],[s]],[[],[m,l],[s]]],[[s],
[m,l],[] ]]
Move = m12
NextState = [[],[s,m,l],[]]
PathSoFar =
[[[s,m,l],[],[]],[[m,l],[s],[]],[[l],[s],[m]],[[s,l],[],[m]],
[[l ],[],[s,m]],[[],[l],[s,m]],[[s],[l],[m]],[[],[s,l],[m]],[[m],
[s, l],[]],[[s,m],[l],[]],[[m],[l],[s]],[[],[m,l],[s]]],[[s],
[m,l],[] ],[[],[s,m,l],[]]]
Move = m21
```

```
NextState = [[s],[m,l],[]]
Move = m23
NextState = [[],[m,l],[s]]
Move = m13
NextState = [[],[m,l],[s]]
Move = m21
NextState = [[m,s],[l],[]]
Move = m23
NextState = [[s],[l],[m]]
Move = m32
NextState = [[],[s,m,l],[]]
PathSoFar =
[[[s,m,l],[],[]],[[m,l],[s],[]],[[l],[s],[m]],[[s,l],[],[m]],[[l
],[],[s,m]],[[],[l],[s,m]],[[s],[l],[m]],[[],[s,l],[m]],[[m],[s,
l],[]],[[s,m],[l],[]],[[m],[l],[s]],[[],[m,l],[s]],[[],[s,m,l],[
]]]
Move = m21
NextState = [[s],[m,l],[]]
PathSoFar =
[[[s,m,l],[],[]],[[m,l],[s],[]],[[l],[s],[m]],[[s,l],[],[m]],[[l
],[],[s,m]],[[],[l],[s,m]],[[s],[l],[m]],[[],[s,l],[m]],[[m],[s,
l],[]],[[s,m],[l],[]],[[m],[l],[s]],[[],[m,l],[s]],[[],[s,m,l],[
]],[[s],[m,l],[]]]
Move = m12
NextState = [[],[s,m,l],[]]
Move = m13
NextState = [[],[m,l],[s]]
Move = m21
NextState = [[m,s],[l],[]]
Move = m23
NextState = [[s],[l],[m]]
Move = m23
NextState = [[],[m,l],[s]]
Move = m13
NextState = [[],[l],[m,s]]
Move = m21
NextState = [[l,m],[],[s]]
Move = m23
NextState = [[m],[],[l,s]]
Move = m31
NextState = [[s,m],[l],[]]
Move = m32
NextState = [[m],[s,l],[]]
Move = m21
NextState = [[l,s,m],[],[]]
Move = m23
NextState = [[s,m],[],[l]]
PathSoFar =
[[[s,m,l],[],[]],[[m,l],[s],[]],[[l],[s],[m]],[[s,l],[],[m]],[[l
],[],[s,m]],[[],[l],[s,m]],[[s],[l],[m]],[[],[s,l],[m]],[[m],[s,
l],[]],[[s,m],[l],[]],[[s,m],[],[l]]]
Move = m12
NextState = [[m],[s],[l]]
PathSoFar =
[[[s,m,l],[],[]],[[m,l],[s],[]],[[l],[s],[m]],[[s,l],[],[m]],[[l
],[],[s,m]],[[],[l],[s,m]],[[s],[l],[m]],[[],[s,l],[m]],[[m],[s,
l],[]],[[s,m],[l],[]],[[s,m],[],[l]],[[m],[s],[l]]]
Move = m12
NextState = [[],[m,s],[l]]
Move = m13
NextState = [[],[s],[m,l]]
PathSoFar =
[[[s,m,l],[],[]],[[m,l],[s],[]],[[l],[s],[m]],[[s,l],[],[m]],[[l
],[],[s,m]],[[],[l],[s,m]],[[s],[l],[m]],[[],[s,l],[m]],[[m],[s,
l],[]],[[s,m],[l],[]],[[s,m],[],[l]],[[m],[s],[l]],[[],[s],[m,l]
]]
Move = m21
NextState = [[s],[],[m,l]]
PathSoFar =
[[[s,m,l],[],[]],[[m,l],[s],[]],[[l],[s],[m]],[[s,l],[],[m]],[[l
],[],[s,m]],[[],[l],[s,m]],[[s],[l],[m]],[[],[s,l],[m]],[[m],[s,
l],[]],[[s,m],[l],[]],[[s,m],[],[l]],[[m],[s],[l]],[[],[s],[m,l]
],[[s],[],[m,l]]]
Move = m12
NextState = [[],[s],[m,l]]
Move = m13
NextState = [[],[],[s,m,l]]
PathSoFar =
[[[s,m,l],[],[]],[[m,l],[s],[]],[[l],[s],[m]],[[s,l],[],[m]],[[l
],[],[s,m]],[[],[l],[s,m]],[[s],[l],[m]],[[],[s,l],[m]],[[m],[s,
l],[]],[[s,m],[l],[]],[[s,m],[],[l]],[[m],[s],[l]],[[],[s],[m,l]
],[[s],[],[m,l]],[[],[],[s,m,l]]]
SolutionSoFar =
[m12,m13,m21,m13,m12,m31,m12,m31,m21,m23,m12,m13,m21,m13]
Solution ...
Transfer a disk from tower 1 to tower 2.
Transfer a disk from tower 1 to tower 3.
Transfer a disk from tower 2 to tower 1.
Transfer a disk from tower 1 to tower 3.
Transfer a disk from tower 1 to tower 2.
Transfer a disk from tower 3 to tower 1.
Transfer a disk from tower 1 to tower 2.
Transfer a disk from tower 3 to tower 1.
Transfer a disk from tower 2 to tower 1.
Transfer a disk from tower 2 to tower 3.
Transfer a disk from tower 1 to tower 2.
Transfer a disk from tower 1 to tower 3.
Transfer a disk from tower 2 to tower 1.
```

```
Transfer a disk from tower 1 to tower 3.
True
 ~/Documents/Programming/CSC-344/src/PA6/ [main*]
```

Questions

1. What was the length of your program's solution to the three disk problem?

The program solved the three disk problem in 14 moves.

2. What is the length of the shortest solution to the three disk problem?

The shortest solution is 7 moves long. (M13, M12, M32, M13, M21, M23, M13)

3. How do you account for the discrepancy?

The program is not designed to find the most efficient solution, only the first path to the end it found. The program does a depth-first search, rather than a breadth-first search. The program takes the first path available to it, even if it was not a step toward an optimum solution.

Task 8: Run the program to solve the 4 disk problem

?- solve.
Solution ...
Transfer a disk from tower 1 to tower 2.
Transfer a disk from tower 1 to tower 3.
Transfer a disk from tower 2 to tower 1.
Transfer a disk from tower 1 to tower 3.
Transfer a disk from tower 1 to tower 2.
Transfer a disk from tower 3 to tower 1.
Transfer a disk from tower 1 to tower 2.
Transfer a disk from tower 3 to tower 1.
Transfer a disk from tower 2 to tower 1.
Transfer a disk from tower 1 to tower 3.
Transfer a disk from tower 1 to tower 2.
Transfer a disk from tower 3 to tower 1.
Transfer a disk from tower 1 to tower 2.
Transfer a disk from tower 1 to tower 3.
Transfer a disk from tower 2 to tower 1.
Transfer a disk from tower 1 to tower 3.
Transfer a disk from tower 2 to tower 1.
Transfer a disk from tower 3 to tower 1.
Transfer a disk from tower 1 to tower 2.
Transfer a disk from tower 1 to tower 3.
Transfer a disk from tower 2 to tower 1.
Transfer a disk from tower 1 to tower 3.
Transfer a disk from tower 2 to tower 1.
Transfer a disk from tower 3 to tower 1.
Transfer a disk from tower 1 to tower 2.
Transfer a disk from tower 3 to tower 1.
Transfer a disk from tower 2 to tower 1.
Transfer a disk from tower 1 to tower 3.
Transfer a disk from tower 1 to tower 2.
Transfer a disk from tower 3 to tower 1.
Transfer a disk from tower 1 to tower 2.
Transfer a disk from tower 1 to tower 3.
Transfer a disk from tower 2 to tower 1.
Transfer a disk from tower 1 to tower 3.
Transfer a disk from tower 2 to tower 1.
Transfer a disk from tower 3 to tower 1.
Transfer a disk from tower 1 to tower 2.
Transfer a disk from tower 1 to tower 3.
Transfer a disk from tower 2 to tower 1.
Transfer a disk from tower 1 to tower 3.
true.

Questions

1.What was the length of your program's solution to the four disk problem?
The program solved the four disk problem in 40 moves.

2. What is the length of the shortest solution to the four disk problem?

The four disk problem can be solved in 15 moves.