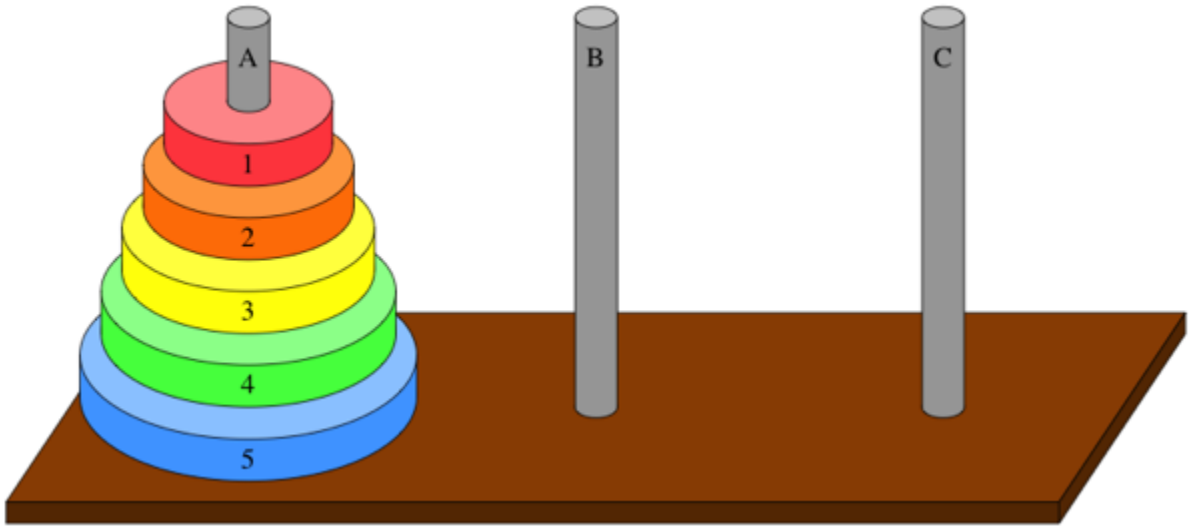
Prolog Assignment 2: Matilda Knapp: 4/20/2022

Abstract:

This is the second prologue assignment, it focuses on the tower of hanoi problem. One for three disks, one for four disks, and one for five disks. This is based on state spaced problem solving. State spaced problems have a beginning and end state, and multiple possible moves. To me this is similar to playing a board game like chess. I'm putting a visual here for what the Towers of Hanoi problem looks like in real life. There will be a link at the end to show the source of the photo.



(This is a picture of a five disk tower of hanoi problem).

Simple Look at Problems:

- I = {((S M L) () ())} [3 disk problem]
- G = {(() () (S M L))}
- O = { M12, M13, M21, M23, M31, M32}

- I = {((S M L H) () ())} [4 disk problem]
- G = {(() () (S M L H))}
- O = { M12, M13, M21, M23, M31, M32}

- I = {((T S M L H) () ())} [5 disk problem]
- G = {(() () (T S M L H))}
- O = { M12, M13, M21, M23, M31, M32}

– M12 - move a disk from peg 1 to peg 2
– M13 - move a disk from peg 1 to peg 3
– M21 - move a disk from peg 2 to peg 1
– M23 - move a disk from peg 2 to peg 3
– M31 - move a disk from peg 3 to peg 1
– M32 - move a disk from peg 3 to peg 2

Task 3: One Move Predicate and a Unit Test.

---

state space operator code:

```
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:

```
%  inspector.pl compiled 0.00 sec, 6 clauses
% c:/Users/roses/Desktop/Prolog Set/toh.pl compiled 0.02 sec, 13 clauses
?- 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 Unit Tests.

---

state space operator code all six:

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

m13([Tower1Before,Tower2,Tower3Before],[Tower1After,Tower2,Tower
3After]) :-
     Tower1Before = [H|T],
     Tower1After = T,
     Tower3Before = L,
     Tower3After = [H|L].

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

```
m23([Tower1,Tower2Before,Tower3Before],[Tower1,Tower2After,Tower
3After]) :-
     Tower2Before = [H|T],
     Tower2After = T,
     Tower3Before = L,
     Tower3After = [H|L].

m31([Tower1Before,Tower2,Tower3Before],[Tower1After,Tower2,Tower
3After]) :-
     Tower3Before = [H|T],
     Tower3After = T,
     Tower1Before = L,
     Tower1After = [H|L].

m32([Tower1,Tower2Before,Tower3Before],[Tower1,Tower2After,Tower
3After]) :-
     Tower3Before = [H|T],
     Tower3After = T,
     Tower2Before = L,
     Tower2After = [H|L].
```

All Unit Move 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 = [[s,m,l,h],[t],[]],
     trace('','TowersBefore',TowersBefore),
     m21(TowersBefore,TowersAfter),
     trace('','TowersAfter',TowersAfter).

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

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

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

Overall Demo:

```
%  inspector.pl compiled 0.00 sec, 6 clauses
% c:/Users/roses/Desktop/Prolog Set/toh.pl compiled 0.00 sec, 23 clauses
?- test__m12.
Testing: move_m12
TowersBefore = [[t,s,m,l,h],[],[]]
TowersAfter = [[s,m,l,h],[t],[]]
true.
```

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

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

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

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

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

Task 5: Valid State Predicate and Unit Test.

Unit Test Code for validity testing:

```
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]]).
```

```
test__vs(S) :-
     valid_state(S),
     write(S), write(' is valid.'), nl.
test__vs(S) :-
     write(S), write(' is invalid.'), nl.
```

Valid State Code:

```
valid_state([A|[B|[C]]]) :-
     valid(A),
     valid(B),
     valid(C).
     valid([]).
     valid([t]).
     valid([s]).
     valid([m]).
     valid([l]).
     valid([h]).
     valid([t,h]).
     valid([t,l]).
     valid([t,m]).
     valid([t,s]).
     valid([s,h]).
     valid([s,l]).
     valid([s,m]).
     valid([m,h]).
     valid([m,l]).
     valid([l,h]).
     valid([t,s,m]).
     valid([t,s,l]).
     valid([t,s,h]).
     valid([t,m,l]).
     valid([t,m,h]).
     valid([t,l,h]).
     valid([s,m,l]).
     valid([s,m,h]).
     valid([s,l,h]).
     valid([m,l,h]).
```

```
valid([t,s,m,l]).
valid([t,s,m,h]).
valid([t,s,l,h]).
valid([t,m,l,h]).
valid([s,m,l,h]).
valid([t,s,m,l,h]).
```

Validity Code Demo:

```
?- 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.

Unit Test Code for write sequence:

```
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]).
```

Write Sequence Code:

```
write_sequence([]).
write_sequence([H|T]) :-
     sequence(H,E),
     write(E),nl,
     write_sequence(T).
sequence(m12,Write) :-
     Write = 'Transfer a disk from tower 1 to tower 2.\n'.
sequence(m13,Write) :-
     Write = 'Transfer a disk from tower 1 to tower 3.\n'.
```

```
sequence(m21,Write) :-
     Write = 'Transfer a disk from tower 2 to tower 1.\n'.
sequence(m23,Write) :-
     Write = 'Transfer a disk from tower 2 to tower 3.\n'.
sequence(m31,Write) :-
     Write = 'Transfer a disk from tower 3 to tower 1.\n'.
sequence(m32,Write) :-
     Write = 'Transfer a disk from tower 3 to tower 2.\n'.
```

Write Sequence 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.

Intermediate output with english like demo:

```
?- 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 = [[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
```

Only english like demo:

```
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
```

Questions:

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

The length of my program's solution to the three disk problem is 14 moves.

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

The shortest solution to the three disk problem is 7 moves.

3. How do you account for the discrepancy?

The program tries to cover all options which causes it to take more steps then it needs to.

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

English Solution Demo:

*Note Full output was so long that it got cut off by prolog, and I can’t scroll up anymore.

```
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?

40 moves in total, it's a lot.

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

The shortest possible solution is 15 moves.

Task 8:Looking for Bugs

Complete Source Code:

```
% --- File: towers_of_hanoi.pl
% --- Line: Program to solve the Towers of Hanoi problem
:- consult('inspector.pl').
```

```
% --- make_move(S,T,SSO) :: Make a move from state S to state T
by SSO
make_move(TowersBeforeMove,TowersAfterMove,m12) :-
     m12(TowersBeforeMove,TowersAfterMove).
make_move(TowersBeforeMove,TowersAfterMove,m13) :-
     m13(TowersBeforeMove,TowersAfterMove).
make_move(TowersBeforeMove,TowersAfterMove,m21) :-
     m21(TowersBeforeMove,TowersAfterMove).
make_move(TowersBeforeMove,TowersAfterMove,m23) :-
     m23(TowersBeforeMove,TowersAfterMove).
make_move(TowersBeforeMove,TowersAfterMove,m31) :-
     m31(TowersBeforeMove,TowersAfterMove).
make_move(TowersBeforeMove,TowersAfterMove,m32) :-
     m32(TowersBeforeMove,TowersAfterMove).

% --- ADD HERE the six state space operators
m12([Tower1Before,Tower2Before,Tower3],[Tower1After,Tower2After,
Tower3]) :-
     Tower1Before = [H|T],
     Tower1After = T,
     Tower2Before = L,
     Tower2After = [H|L].

m13([Tower1Before,Tower2,Tower3Before],[Tower1After,Tower2,Tower
3After]) :-
     Tower1Before = [H|T],
     Tower1After = T,
     Tower3Before = L,
     Tower3After = [H|L].

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

```
m23([Tower1,Tower2Before,Tower3Before],[Tower1,Tower2After,Tower
3After]) :-
     Tower2Before = [H|T],
     Tower2After = T,
     Tower3Before = L,
     Tower3After = [H|L].

m31([Tower1Before,Tower2,Tower3Before],[Tower1After,Tower2,Tower
3After]) :-
     Tower3Before = [H|T],
     Tower3After = T,
     Tower1Before = L,
     Tower1After = [H|L].

m32([Tower1,Tower2Before,Tower3Before],[Tower1,Tower2After,Tower
3After]) :-
     Tower3Before = [H|T],
     Tower3After = T,
     Tower2Before = L,
     Tower2After = [H|L].

% --- valid_state(S) :: S is a valid state
% --- ADD HERE valid state
valid_state([A|[B|[C]]]) :-
     valid(A),
     valid(B),
     valid(C).
     valid([]).
     valid([t]).
     valid([s]).
     valid([m]).
     valid([l]).
     valid([h]).
     valid([t,h]).
     valid([t,l]).
     valid([t,m]).
     valid([t,s]).
```

```
     valid([s,h]).
     valid([s,l]).
     valid([s,m]).
     valid([m,h]).
     valid([m,l]).
     valid([l,h]).
     valid([t,s,m]).
     valid([t,s,l]).
     valid([t,s,h]).
     valid([t,m,l]).
     valid([t,m,h]).
     valid([t,l,h]).
     valid([s,m,l]).
     valid([s,m,h]).
     valid([s,l,h]).
     valid([m,l,h]).
     valid([t,s,m,l]).
     valid([t,s,m,h]).
     valid([t,s,l,h]).
     valid([t,m,l,h]).
     valid([s,m,l,h]).
     valid([t,s,m,l,h]).

% --- solve(Start,Solution) :: succeeds if Solution represents a
path
% --- from the start state to the goal state.
solve :-
     extend_path([[[s,m,l,h],[],[]]],[],Solution),
     write_solution(Solution).

     extend_path(PathSoFar,SolutionSoFar,Solution) :-
PathSoFar = [[[],[],[s,m,l,h]]|_],
showr('PathSoFar',PathSoFar),
showr('SolutionSoFar',SolutionSoFar),
Solution = SolutionSoFar.
extend_path(PathSoFar,SolutionSoFar,Solution) :-
PathSoFar = [CurrentState|_],
```

```
     showr('PathSoFar',PathSoFar),
     make_move(CurrentState,NextState,Move),
     show('Move',Move),
     show('NextState',NextState),
     not(member(NextState,PathSoFar)),
     valid_state(NextState),
     Path = [NextState|PathSoFar],
     Soln = [Move|SolutionSoFar],
     extend_path(Path,Soln,Solution).

% --- 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.

% --- HERE write_sequence
write_sequence([]).
write_sequence([H|T]) :-
     sequence(H,E),
     write(E),nl,
     write_sequence(T).
sequence(m12,Write) :-
     Write = 'Transfer a disk from tower 1 to tower 2.\n'.
sequence(m13,Write) :-
     Write = 'Transfer a disk from tower 1 to tower 3.\n'.
sequence(m21,Write) :-
     Write = 'Transfer a disk from tower 2 to tower 1.\n'.
sequence(m23,Write) :-
     Write = 'Transfer a disk from tower 2 to tower 3.\n'.
sequence(m31,Write) :-
     Write = 'Transfer a disk from tower 3 to tower 1.\n'.
sequence(m32,Write) :-
     Write = 'Transfer a disk from tower 3 to tower 2.\n'.
```

```
% --- Unit test programs
% --- HERE unit test programs
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 = [[s,m,l,h],[t],[]],
     trace('','TowersBefore',TowersBefore),
     m21(TowersBefore,TowersAfter),
     trace('','TowersAfter',TowersAfter).

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

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

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

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]]).
test__vs(S) :-
     valid_state(S),
     write(S), write(' is valid.'), nl.
test__vs(S) :-
     write(S), write(' is invalid.'), nl.

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]).

Cited:
Towers of Hanoi picture source.
https://www.google.com/search?q=towers+of+hanoi+&tbm=isch&ved=2a
hUKEwizsp3UxKv3AhXCr3IEHYHUBhcQ2-cCegQIABAA&oq=towers+of+hanoi+&
gs_lcp=CgNpbWcQAzIFCAAQgAQyBQgAEIAEMgUIABCABDIFCAAQgAQyBQgAEIAEM
gUIABCABDIFCAAQgAQyBQgAEIAEMgUIABCABDIFCAAQgAQ6BAgAEBhQ6ARY5BVg8
hhoAHAAeACAAUWIAYoGkgECMTSYAQCgAQGqAQtnd3Mtd2l6LWltZ8ABAQ&sclien
t=img&ei=TKZkYrPcKMLfytMPgambuAE&bih=722&biw=1536#imgrc=GwWhXO7C
lszDVM
```