

**INT404 Group Project**

Topic: Production System for solving Towers of Hanoi (for any no. of Disc upto 8)

|  |  |  |  |
| --- | --- | --- | --- |
|  | Name | Registration no. | Roll no. |
| 1. | Aayush Golecha | 11813731 | 29 |
| 2. | Rutvik Surani | 11813785 | 30 |
| 3. | Saurav Kumar | 11812088 | 31 |

**Work Distribution:**

**Report:** made by Aayush Golecha

**Program and Production rules**: made by Rutvik Surani

**State space and Rule applier:** made by Saurav Kumar

**Contents**

1. Introduction
2. Representing the Tower of Hanoi
3. Graphical Representation
4. Production System for Tower of Hanoi
5. Notations
6. Types of Solution
7. Applications

**Introduction**

The Tower of Hanoi (also called the Tower of Brahma or Lucas' Tower and sometimes pluralized as Towers) is a mathematical Game or puzzle. It consists of three rods and a number of disks of different sizes, which can slide onto any rod. The puzzle starts with the disks in a neat stack in ascending order of size on one rod, the smallest at the top, thus making a conical shape.

The objective of the puzzle is to move the entire stack to another rod, obeying the following simple rules:

1. Only one disk can be moved at a time.
2. Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack or on an empty rod.
3. No larger disk may be placed on top of a smaller disk.

With 3 disks, the puzzle can be solved in 7 moves. The minimal number of moves required to solve a Tower of Hanoi puzzle is 2*n* − 1, where *n* is the number of disks.

The Puzzle was Invented by the French mathematician Edouard Lucas  in 1883. Numerous myths regarding the ancient and mystical nature of the puzzle popped up almost immediately. There is a story about an Indian temple in Kashi Vishwanath which contains a large room with three time-worn posts in it, surrounded by 64 golden disks.  Brahmin priests, acting out the command of an ancient prophecy, have been moving these disks in accordance with the immutable rules of Brahma since that time. The puzzle is therefore also known as the Tower of  Brahma puzzle. According to the legend, when the last move of the puzzle is completed, the world will end.

**Representing the Tower of Hanoi**

The Tower of Hanoi puzzle involves moving a pile of different size disks from one peg to another using an intermediate peg. Only one disk at a time can be moved, a disk can only be moved if it is the top disk on a pile, and a larger disk can never be placed on a smaller one. Figure 1 shows the initial and goal states of a three-disk problem.

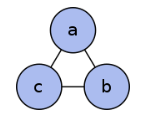
****

The most straight-forward axiomatization of this problem consists of an operator for moving each disk between each pair of pegs. For the three-disk problem, this axiomatization requires 18 operators. Table 1 shows the operator for moving disk C, the largest disk, from peg 1 to 3. The preconditions require that disk C is initially on peg 1 and that neither disk A nor B are on peg 1 or 3. This representation is far from the most concise one, but it is used in this paper to simplify the exposition. The basic ideas described in this paper apply to and have been tested on more compact representations.

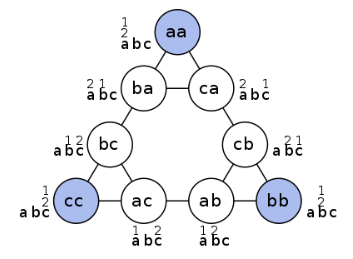
The size of the problems in the Tower of Hanoi puzzle vary based on the number of disks. The number of possible states for a given puzzle with n disks is 3n since each disk can be on one of the three pegs. The state space for the three-disk puzzle is shown. Each node represents a state and is labeled with the a picture of the state, and each arrow represents an operator that can be applied to reach the adjacent state. A solution to the three-disk problem given above consists of any path through the state space that starts at the initial state and terminates at the goal state. The shortest solution follows the path along the diagonal between the initial and goal states.

**Graphical Representation**

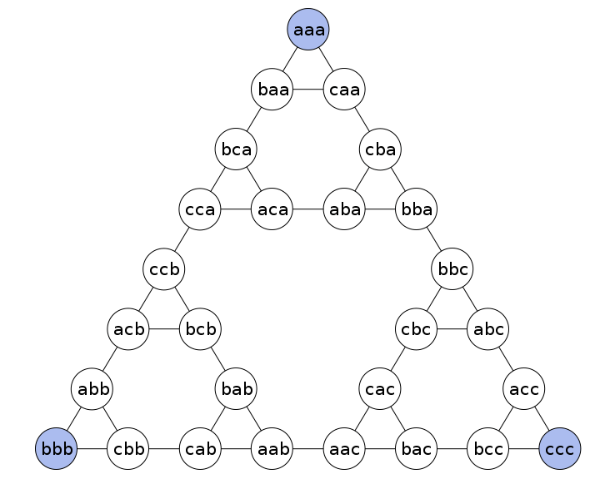
The game can be represented by an  undirected graph, the nodes representing distributions of disks and the edges representing moves. For one disk, the graph is a triangle:



The graph for two disks is three triangles connected to form the corners of a larger triangle. A second letter is added to represent the larger disk. Clearly it cannot initially be moved. The topmost small triangle now represents the one-move possibilities with two disks:



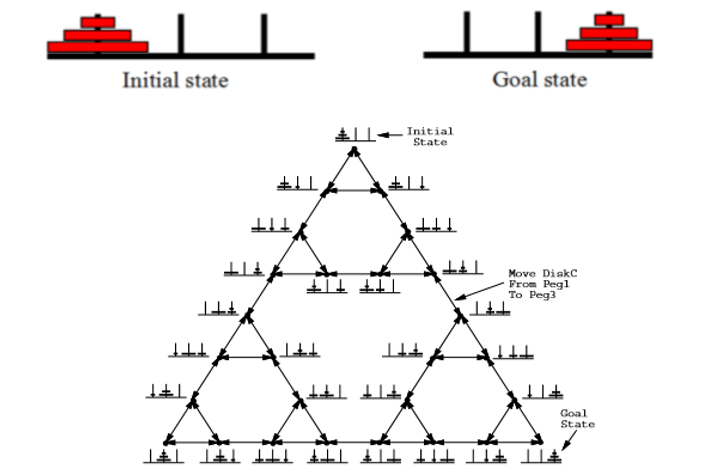
The nodes at the vertices of the outermost triangle represent distributions with all disks on the same peg. For h + 1 disks, take the graph of h disks and replace each small triangle with the graph for two disks. For three disks the graph is:



**Production System for Tower of Hanoi**

Production System is the basic layout of the AI based solution of the problem. First, we define and analyse the problem. After this the production system is used to generate and apply rules to the system to obtain an optimal solution.

**State Space:**



**Notations**

**Pickup Rod(i) :** Pick up disk i from its current location and hold it. It is assumed that there is no disk on the top.

**Putdown Rod(i)** :Place disk i down at some location and record the new location for i.

**stack(i,j)** :Place disc i on the top of disk j. we assume that the size of disk j is bigger than i only then we can put I on top of j.

**unstack(i,j) :**Remove disk i from the top of j. we assume that there is no disk above i. therefore we can pick it up and place it somewhere else.

**DISK(A): disk “A”**

**Rod(1): rod 1**

**Actions and effects:**

|  |  |
| --- | --- |
| Action | Meaning |
| Stack(d(i),d(j),n(R)) | Stack disk(i) on disk(j) in rod(R) |
| Unstack(d(i),d(j),n(R)) | Unstack disk(i) from Disk(j) from rod(R) |
| Pickup(d(i),n(R)) | Pickup disk(i) from rod(R) |
| Putdown(d(i),n(R)) | Putdown disk(i) in rod(R)) |

**Production Rules:**

**General Rules:**

* if the smaller disk is placed on large disk in Rod(i) then

-> unstack(small disk, large disk, Rod(i))

* if small disk is holding by arm of planning then

-> stack(small disk, large disk, Rod(i)).

* if a disk(W) alone in Rod(i) then

->pickup(disk(W),Rod(i))

* if holding(disk(W) then

-> putdown(disk(W),Rod(i))

**Main Rules:**

Rule.no1: if disk(A) in rod(1) then -> pickup(disk(A),rod(1)) and putdown(disk(A),rod(2)).

Rule.no2: if disk(A) in Rod(1) then-> pickup(disk(A),Rod(1)) and putdown(disk(A),Rod(3)).

Rule.no3: if disk(A) in Rod(2) then-> pickup(disk(A),Rod(2)) and putdown(disk(A),Rod(1)).

Rule.no4: if disk(A) in Rod(2) then-> pickup(disk(A),Rod(2)) and putdown(disk(A),Rod(3)).

Rule.no5: if disk(A) in Rod(3) then-> pickup(disk(A),Rod(3)) and putdown(disk(A),Rod(1)).

Rule.no6: if disk(A) in Rod(3) then ->pickup(disk(A),Rod(3)) and putdown(disk(A),Rod(2)).

Rule.no7: if disk(B) in Rod(1) and not(disk(A),Rod(2)) then

-> pickup(disk(B),Rod(1)) and putdown(disk(B),Rod(2)).

Rule.no8: if disk(B) in Rod(1) and not(disk(A),Rod(3)) then

-> pickup(disk(B),Rod(1)) and putdown(disk(B),Rod(3)).

Rule.no9: if disk(B) in Rod(2) and not(disk(A),Rod(1)) then

->pickup(disk(B),Rod(2)) and d putdown(disk(B),Rod(1)).

Rule.no10: if disk(B) in Rod(2) and not(disk(A),Rod(3)) then

->pickup(disk(B),Rod(2)) and putdown(disk(B),Rod(3)).

Rule.no11: if disk(B) in Rod(3) and not(disk(A),Rod(1)) then

->pickup(disk(B),Rod(3)) and putdown(disk(B),Rod(1)).

Rule.no12: if disk(B) in Rod(3) and not(disk(A), Rod(2)) then

->pickup(disk(B),Rod(3)) and putdown(disk(B),Rod(2)).

Rule.no13: if disk(C) in Rod(1) and not(disk(A),Rod(2)) and not(disk(B),Rod(2)) then

-> pickup(disk(C),Rod(1)) and putdown(disk(C),Rod(2)).

Rule.no14: if disk(C) in Rod(1) and not(disk(A),Rod(3)) and not(disk(B),Rod(3)) then

-> pickup(disk(C),Rod(1)) and putdown(disk(C),Rod(3)).

Rule.no15: if disk(C) in Rod(2) and not(disk(A),Rod(1)) and not(disk(B),Rod(1)) then

->pickup(disk(C),Rod(2)) and putdown(disk(C),Rod(1)).

Rule.no16: if disk(C) in Rod(2) and not(disk(A),Rod(3)) and not(disk(B),Rod(3)) then

->pickup(disk(C),Rod(2)) and putdown(disk(C),Rod(3)).

Rule.no17: if disk(C) in Rod(3) and not(disk(A),Rod(1)) and not(disk(B),Rod(1)) then

->pickup(disk(C),Rod(3)) and putdown(disk(C),Rod(1)).

Rule.no18: if disk(C) in Rod(3) and not(disk(A),Rod(2)) and not(disk(B),Rod(3)) then

->pickup(disk(C),Rod(3)) and putdown(disk(C),Rod(2)).

**Rule Applier:**

Action(1) Unstack(disk(A),disk(B),Rod(1)) disk(A),disk(B),disk(C),on(disk(C),table,Rod(1)),on(disk(B),disk(C),Rod(1)),clear(disk(B)), HOLDING(disk(A)).

Action(2) Putdown(disk(A),Rod(3)) disk(A),disk(B),disk(C),on(disk(C),table,Rod(1)),on(disk(B),disk(C),Rod(1)),on(disk(A),table,Rod(3)), clear(disk(A)),clear(disk(B)),HAND EMPTY.

Action(3) Unstack(disk(B),disk(C),Rod(1)) disk(A),disk(B),disk(C),on(disk(C),table,Rod(1)),on(disk(A),table,Rod(3)),clear(disk(A)), clear(disk(C)),HOLDING(disk(B))

Action(4) Putdown(disk(B),Rod(2)) disk(A),disk(B),disk(C),on(disk(A),table,Rod(3)),on(disk(B),table,Rod(2)),on(disk(C),table,Rod(1)), clear(disk(A)),clear(disk(B)), HAND EMPTY.

Action(5) Pickup(disk(A),Rod(3)) disk(A),disk(B),disk(C),on(disk(B),table,Rod(2)),on(disk(C),table,Rod(1)),clear(disk(B)), HOLDING(disk(A)).

Action(6) Stack(disk(A),disk(B),Rod(2))disk(A),disk(B),disk(C),on(disk(B),table,Rod(2)),on(disk(C),table,Rod(1)),on(disk(A),disk(B),Rod(2)),clear(disk(A)), clear(disk(C)), HAND EMPTY.

Action(7) Pickup(disk(C),Rod(1)) disk(A),disk(B),disk(C),on(disk(B),table,Rod(2)),on(disk(A),disk(B),Rod(2)),clear(disk(A)), HOLDING(disk(C)).

Action(8) Putdown(disk(C),Rod(3)) disk(A),disk(B),disk(C),on(disk(B),table,Rod(2)),on(disk(A),disk(B),Rod(2)),on(disk(C),table,Rod(3)),clear(disk(A)), clear(disk(C)),HAND EMPTY.

Action(9) Unstack(disk(A),disk(B),Rod(2)) disk(A),disk(B),disk(C),on(disk(B),table,Rod(2)),on(disk(C),table,Rod(3)),clear(disk(B)), clear(disk(C)),HOLDING(disk(A)).

Action(10) Putdown(disk(A),Rod(1)) disk(A),disk(B),disk(C),on(disk(A),table,Rod(1)),on(disk(B),table,Rod(2)),on(disk(C),table,Rod(3)),clear(disk(A)), clear(disk(B)), clear(disk(C)),HAND EMPTY.

Action(11) Pickup(disk(B),Rod(2)) disk(A),disk(B),disk(C),on(disk(A),table,Rod(1)),on(disk(C),table,Rod(3)),clear(disk(A)), clear(disk(C)),HOLDING(disk(B)).

Action(12) Stack(disk(B),disk(C),Rod(3)) disk(A),disk(B),disk(C),on(disk(A),table,Rod(1)),on(disk(B),disk(C),Rod(3)),clear(disk(A)), clear(disk(B)),HAND EMPTY.

Action(13) Pickup(disk(A),Rod(1)) disk(A),disk(B),disk(C),on(disk(B),disk(C),Rod(3)),clear(disk(B)),HOLDING(A).

Action(14) Stack(disk(A)),Rod(3)) disk(A),disk(B),disk(C),on(disk(A),disk(B),Rod(3)),on(disk(B),disk(C),Rod(3)),HAND EMPTY.

**Types of Solution**

**Iterative solution**

A simple solution for the toy puzzle is to alternate moves between the smallest piece and a non-smallest piece. When moving the smallest piece, always move it to the next position in the same direction (to the right if the starting number of pieces is even, to the left if the starting number of pieces is odd). If there is no tower position in the chosen direction, move the piece to the opposite end, but then continue to move in the correct direction. For example, if you started with three pieces, you would move the smallest piece to the opposite end, then continue in the left direction after that. When the turn is to move the non-smallest piece, there is only one legal move. Doing this will complete the puzzle in the fewest moves.

**Recursive solution**

The key to solving a problem recursively is to recognize that it can be broken down into a collection of smaller sub-problems, to each of which *that same general solving procedure that we are seeking* applies, and the total solution is then found in some *simple* way from those sub-problems' solutions. Each of thus created sub-problems being "smaller" guarantees that the base case(s) will eventually be reached.

### Non-recursive solution

The list of moves for a tower being carried from one peg onto another one, as produced by the recursive algorithm, has many regularities. When counting the moves starting from 1, the ordinal of the disk to be moved during move *m* is the number of times *m* can be divided by 2. Hence every odd move involves the smallest disk. It can also be observed that the smallest disk traverses the pegs *f*, *t*, *r*, *f*, *t*, *r*, etc. for odd height of the tower and traverses the pegs *f*, *r*, *t*, *f*, *r*, *t*, etc. for even height of the tower. This provides the following algorithm, which is easier, carried out by hand, than the recursive algorithm.

### Gray-code solution

The binary numeral system of  Gray codes gives an alternative way of solving the puzzle. In the Gray system, numbers are expressed in a binary combination of 0s and 1s, but rather than being a standard positional numeral system, Gray code operates on the premise that each value differs from its predecessor by only one (and exactly one) bit changed.

If one counts in Gray code of a bit size equal to the number of disks in a particular Tower of Hanoi, begins at zero, and counts up, then the bit changed each move corresponds to the disk to move, where the least-significant bit is the smallest disk, and the most-significant bit is the largest.

### Binary solution

Disk positions may be determined more directly from the binary (base-2) representation of the move number (the initial state being move #0, with all digits 0, and the final state being with all digits 1), using the following rules:

* There is one binary digit (bit) for each disk.
* The most significant (leftmost) bit represents the largest disk. A value of 0 indicates that the largest disk is on the initial peg, while a 1 indicates that it's on the final peg (right peg if number of disks is odd and middle peg otherwise).
* The bitstring is read from left to right, and each bit can be used to determine the location of the corresponding disk.
* A bit with the same value as the previous one means that the corresponding disk is stacked on top the previous disk on the same peg.

(That is to say: a straight sequence of 1s or 0s means that the corresponding disks are all on the same peg.)

**Applications**

The Tower of Hanoi is frequently used in psychological research on problem solving. There also exists a variant of this task called Tower of London for neuropsychological diagnosis and treatment of executive functions.

Zhang and Norman used several isomorphic (equivalent) representations of the game to study the impact of representational effect in task design. They demonstrated an impact on user performance by changing the way that the rules of the game are represented, using variations in the physical design of the game components. This knowledge has impacted on the development of the TURF framework for the representation of human computer interaction.

The Tower of Hanoi is also used as a Backup rotation scheme when performing computer data Backups where multiple tapes/media are involved.

As mentioned above, the Tower of Hanoi is popular for teaching recursive algorithms to beginning programming students. A pictorial version of this puzzle is programmed into the emacs editor, accessed by typing M-x hanoi. There is also a sample algorithm written in prolog.

The Tower of Hanoi is also used as a test by neuropsychologists trying to evaluate frontal lobe deficits.