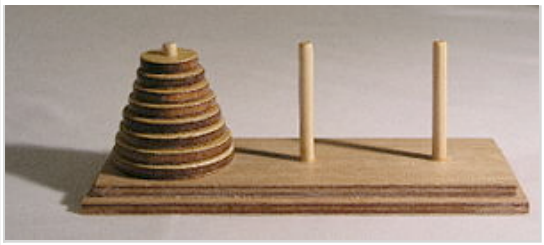
Tower of Hanoi

From Wikipedia, the free encyclopedia
(Redirected from Towers of hanoi)

The **Tower of Hanoi** or **Towers of Hanoi** (also known as **The Towers of Benares**) 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 neatly stacked in order of size on one rod, the smallest at the top, thus making a conical shape.



A model set of the Towers of Hanoi (with 8 disks)

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

- Only one disk may be moved at a time.
- Each move consists of taking the upper disk from one of the pegs and sliding it onto another rod, on top of the other disks that may already be present on that rod.
- No disk may be placed on top of a smaller disk.

An animated solution of the **Tower of Hanoi** puzzle for *T(4,3)*.

Contents

- 1 Origins
- 2 Solution
  - 2.1 Simple solution
  - 2.2 Recursive solution
  - 2.3 Non-recursive solution
  - 2.4 Binary solutions
  - 2.5 Gray code solution
- 3 Long solutions
- 4 Graphical representation
- 5 Applications
- 6 Four pegs and beyond
  - 6.1 Description of the presumed-optimal solution
  - 6.2 Two stacks and more stacks
- 7 In popular culture
  - 7.1 In video games
- 8 See also
- 9 Notes
- 10 External links
  - 10.1 Online demonstrations
  - 10.2 Algorithm
  - 10.3 Events

Origins

The puzzle was invented by the French mathematician Édouard Lucas in 1883. There is a legend about a Vietnamese or Indian temple which contains a large room with three time-worn posts in it surrounded by 64 golden disks. The priests of Brahma, acting out the command of an ancient prophecy, have been moving these disks, in accordance with the rules of the puzzle. According to the legend, when the last move of the puzzle is completed, the world will end. The puzzle is therefore also known as the Tower of Brahma puzzle. It is not clear whether Lucas invented this legend or was inspired by it. The Tower of Hanoi is a problem often used to teach beginning programming, in particular, as an example of a simple recursive algorithm.


Flag Tower of Hanoi

If the legend were true, and if the priests were able to move disks at a rate of one per second, using the smallest number of moves, it would take them 2^{64}-1 seconds or roughly 600 billion years (operation taking place is $\frac{2^{64}-1}{60 \times 60 \times 24 \times 365.2425}$).[1]

There are many variations on this legend. For instance, in some tellings, the temple is a monastery and the priests are monks. The temple or monastery may be said to be in different parts of the world — including Hanoi, Vietnam, and may be associated with any religion. In some versions, other elements are introduced, such as the fact that the tower was created at the beginning of the world, or that the priests or monks may make only one move per day.

The Flag Tower of Hanoi may have served as the inspiration for the name.

Solution

Most toy versions of the puzzle have 8 disks. The game seems impossible to many novices, yet is solvable with a simple algorithm:

Simple solution

The following solution is a simple solution for the toy puzzle.

Alternate moves between the smallest piece and a non-smallest piece. When moving the smallest piece, always move it in the same direction (either to the left or to the right, but be consistent). If there is no tower in the chosen direction, move it to the opposite end. When the turn is to move the non-smallest piece, there is only one legal move.

Recursive solution

As in many mathematical puzzles, finding a solution is made easier by solving a slightly more general problem: how to move a tower of h (h=height) disks from a starting peg **f** (f=from) onto a destination peg **t** (t=to), **r** being the remaining third peg and assuming **t** ≠ **f**. First, observe that the problem is symmetric for permutations of the names of the pegs (symmetric group S_3). If a solution is known moving from peg **f** to peg **t**, then, by renaming the pegs, the same solution can be used for every other choice of starting and destination peg. If there is only one disk (or even none at all), the problem is trivial. If h=1, then simply move the disk from peg **f** to peg **t**. If h>1, then somewhere along the sequence of moves, the largest disk must be moved from peg **f** to another peg, preferably to peg **t**. The only situation that allows this move is when all smaller h-1 disks are on peg **r**. Hence, first all h-1 smaller disks must go from **f** to **r**. Subsequently move the largest disk and finally move the h-1 smaller disks from peg **r** to peg **t**. The presence of the largest disk does not impede any move of the h-1 smaller disks and can temporarily be ignored. Now the problem is reduced to moving h-1 disks from one peg to another one, first from **f** to **r** and subsequently from **r** to **t**, but the same method can be used both times by renaming the pegs. The same strategy can be used to reduce the h-1 problem to h-2, h-3, and so on until only one disk is left. This is called recursion. This algorithm can be schematized as follows. Identify the disks in order of increasing size by the natural numbers from 0 up

to but not including h. Hence disk 0 is the smallest one and disk h-1 the largest one.

The following is a procedure for moving a tower of h disks from a peg **f** onto a peg **t**, with **r** being the remaining third peg:

- Step 1: If h>1 then first use this procedure to move the h-1 smaller disks from peg **f** to peg **r**.
- Step 2: Now the largest disk, i.e. disk h-1 can be moved from peg **f** to peg **t**.
- Step 3: If h>1 then again use this procedure to move the h-1 smaller disks from peg **r** to peg **t**.

By means of mathematical induction, it is easily proven that the above procedure requires the minimal number of moves possible, and that the produced solution is the only one with this minimal number of moves.

Using recurrence relations, the exact number of moves that this solution requires can be calculated by: $2^h - 1$. This result is obtained by noting that steps 1 and 3 take T_{h-1} moves, and step 2 takes one move, giving $T_h = 2T_{h-1} + 1$.

The algorithm can be written elegantly in functional programming languages, such as OCaml:

```
let rec move_tower n a b c = match n with
        | 1 -> [(a,c)]
        | _ -> (move_tower (n-1) a c b) @ (move_tower 1 a b c) @ (move_tower (n-1) b a c);;
```

or Lisp

```
(defun hanoitowers (disc src aux dst)
  (cond ((> disc 0)
         (hanoitowers (- disc 1) src dst aux)
         (princ  (list "Move" disc "from" src "to" dst))
         (hanoitowers (- disc 1) aux src dst))))
```

Recursive solutions in imperative programming languages typically resemble the following Pascal code:

```
procedure Hanoi(n: integer; from, dest, by: char);
Begin
    if (n=1) then
        writeln('Move the plate from ', from, ' to ', dest)
    else begin
        Hanoi(n-1, from, by, dest);
        Hanoi(1, from, dest, by);
        Hanoi(n-1, by, dest, from);
    end;
End;
```

A very clean implementation in java:

```
TowersOfHanoi(4, 1, 3, 2);

private void TowersOfHanoi(int numRings, int source, int temp, int dest) {
        if (numRings == 1)
            System.out.println("Move from peg " + source + " to " + temp + ".");
        else {
            TowersOfHanoi(numRings-1, source, dest, temp);
            System.out.println("Move from peg " + source + " to " + temp + ".");
            TowersOfHanoi(numRings-1, dest, temp, source);
        }
}
```

Implementations in many other languages may be found at the Hanoimania! (http://www.kernelthread.com/hanoi/) website.

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.

In alternate moves:

- move the smallest disk to the peg it has not recently come from.
- move another disk legally (there will be one possibility only)

For the very first move, the smallest disk goes to peg t if h is odd and to peg r if h is even.

So if the number of disks is even the solution will start:

1. Move disk 0 from peg f to peg r ignoring peg t.
2. Move disk 1 from peg f to peg t ignoring peg r.
3. Move disk 0 from peg r to peg t ignoring peg f.
4. Move disk 2 from peg f to peg r ignoring peg t.
5. Move disk 0 from peg t to peg f ignoring peg r.
6. Move disk 1 from peg t to peg r ignoring peg f.
7. Move disk 0 from peg f to peg r ignoring peg t.
8. Move disk 3 from peg f to peg t ignoring peg r.
9. Move disk 0 from peg r to peg t ignoring peg f.
10. Move disk 1 from peg r to peg f ignoring peg t.
11. Move disk 0 from peg t to peg f ignoring peg r.
12. Move disk 2 from peg r to peg t ignoring peg f.
13. Move disk 0 from peg f to peg r ignoring peg t.
14. Move disk 1 from peg f to peg t ignoring peg r.
15. Move disk 0 from peg r to peg t ignoring peg f.
16. Move disk 4 from peg f to peg r ignoring peg t.

etc.

Also observe that:

- Disks whose ordinals have even parity move in the same sense as the smallest disk.
- Disks whose ordinals have odd parity move in opposite sense.
- If h is even, the remaining third peg during successive moves is t, r, f, t, r, f, etc.
- If h is odd, the remaining third peg during successive moves is r, t, f, r, t, f, etc.

Binary solutions

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 $\#2^n$-1, 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.
- 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 1's or 0's means that the corresponding disks are all on the same peg).
- A bit with a different value to the previous one means that the corresponding disk is one position to the left or right of the previous one. Whether it is left or right is determined by this rule:
    - Assume that the initial peg is on the left and the final peg is on the right.

- Also assume "wrapping" - so the right peg counts as one peg "left" of the left peg, and vice versa.
- Let n be the number of greater disks that are located on the same peg as their first greater disk and add 1 if the largest disk is on the left peg. If n is even, the disk is located one peg to the left, if n is odd, the disk located one peg to the right.

For example, in an 8-disk Hanoi:

- Move #0)
  - The largest disk is 0, so it is on the left (initial) peg.
  - All other disks are 0 as well, so they are stacked on top of it. Hence all disks are on the initial peg.
- Move #2^8-1)
  - The largest disk is 1, so it is on the right (final) peg.
  - All other disks are 1 as well, so they are stacked on top of it. Hence all disks are on the final peg and the puzzle is complete.
- Move #0b11011000)
  - The largest disk is 1, so it is on the right (final) peg.
  - Disk two is also 1, so it is stacked on top of it, on the right peg.
  - Disk three is 0, so it is on another peg. Since n is odd, it is one peg to the right, i.e. on the middle peg.
  - Disk four is 1, so it is on another peg. Since n is still odd, it is one peg to the right, i.e. on the right peg.
  - Disk five is also 1, so it is stacked on top of it, on the right peg.
  - Disk six is 0, so it is on another peg. Since n is even, the disk is one peg to the left, i.e. on the middle peg.
  - Disks seven and eight are also 0, so they are stacked on top of it, on the middle peg.

The above algorithm can be coded in Scheme as follows:

```
(define (conf m h f t) ; m=move number, h=height of tower, f=starting peg, t=destination peg
 ; Identify the pegs by the numbers 0, 1 and 2.
 (let loop ((prev-zero? #t) (mask (arithmetic-shift 1 (sub1 h))) (rotation (- t f)) (f f))
  (if (zero? mask) ()
   (let ((zero-bit? (zero? (bitwise-and mask m))) (mask (arithmetic-shift mask -1)))
    (if (eq? prev-zero? zero-bit?) (cons f (loop zero-bit? mask (- rotation) f))
     (let ((f (modulo (+ f rotation) 3)))
      (cons f (loop zero-bit? mask rotation f))))))))

; This procedure produces a list of the positions of the disks in order of decreasing size.
; Example:

(conf #e6022e20 80 0 2) ; in less than a milisecond on a plain 1.8 GHz personal computer:
;--> (01111111100001201201221111201112012000)
; #e6022e20 = 602, which is about the number of [[Avogadro]].
```

The source and destination pegs for the *m*th move can also be found elegantly from the binary representation of *m* using bitwise operations. To use the syntax of the C programming language, the *m*th move is from peg `(m&m-1)%3` to peg `((m|m-1)+1)%3`, where the disks begin on peg 0 and finish on peg 1 or 2 according as whether the number of disks is even or odd. Furthermore the disk to be moved is determined by the number of times the move count (m) can be divided by 2 (i.e. the number of zero bits at the right), counting the first move as 1 and identifying the disks by the numbers 0, 1, 2 etc in order of increasing size. This permits a very fast non-recursive computer implementation to find the positions of the disks after m moves without reference to any previous move or distribution of disks:

```
; h : total number of disks
; m : move counter, starting with 1 for the first move.
; f : starting peg; the pegs are identified by the numbers 0, 1 and 2.
; t : destination peg
; d : disk (numbered 0, 1, 2, etc in order of increasing size)
; Function =quotient= takes two integer numbers and computes their quotient rounded to integer towa
; (rot3 m) : sense of rotation of the remaining third peg during move m.
; (rotd d) : sense of rotation of disk d.
; mcnt : number of moves disk d has made after a total of m moves.
; from : the peg a disk is taken from during move m.
; onto : the peg a disk is put onto during move m.
; thrd : the remaining third peg. (- 3 from onto)
; disk : disk being moved.
; conf : position of disk d after a total of m moves.

(define (exp2 n        ) (expt   2 n))
(define (mod2 n        ) (modulo n 2))
(define (mod3 n        ) (modulo n 3))
(define (pari n        ) (add1 (mod2 (add1 n))))
(define (rotd   h d f t) (mod3 (* (- t f) (pari (- h d)))))
(define (rot3   h   f t) (rotd h 0 f t))
(define (mcnt m   d    ) (quotient (+ m (exp2 d)) (exp2 (add1 d))))
(define (thrd m h   f t) (mod3 (- f (* m (rot3 h f t)))))
(define (onto m h   f t) (mod3 (- (thrd m h f t) (rotd h (disk m) f t))))
(define (from m h   f t) (mod3 (+ (thrd m h f t) (rotd h (disk m) f t))))
(define (conf m h d f t) (mod3 (+ f (* (rotd h d f t) (mcnt m d)))))
(define (disk m        ) (sub1 (bit-count (bitwise-xor m (sub1 m)))))
```

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. The number of bits present in Gray code is important, and leading zeros are not optional, unlike in positional systems.

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.

> Counting moves from 1 and identifying the disks by numbers starting from 0 in order of increasing size, the ordinal of the disk to be moved during move m is the number of times m can be divided by 2.

This technique identifies which disk to move, but not where to move it to. For the smallest disk there are always two possibilities. For the other disks there is always one possibility, except when all disks are on the same peg, but in that case either it is the smallest disk that must be moved or the objective has already been achieved. Luckily, there is a rule which does say where to move the smallest disk to. Let f be the starting peg, t the destination peg and r the remaining third peg. If the number of disks is odd, the smallest disk cycles along the pegs in the order f->t->r->f->t->r, etc. If the number of disks is even, this must be reversed: f->r->t->f->r->t etc. [1] (http://occawlonline.pearsoned.com/bookbind/pubbooks/miller2_awl/chapter4/essay1/deluxe-content.html#tower)

Long solutions

A modification of the game can be to move the tower from one peg to another peg using as many moves as possible without ever producing the same distribution of disks more than once. A simple algorithm (written in Scheme) is:

```
(define (long-move-tower h f t r)
  (if (positive? h)
   (let ((h (sub1 h)))
    (long-move-tower h f t r)
    (move-disk       h f r t)
    (long-move-tower h t f r)
    (move-disk       h r t f)
    (long-move-tower h f t r))))
```

Where procedure (move-disk d f t r) moves disk d from peg f onto peg t, ignoring peg r. The number of moves of this uniquely defined solution is 3^{height}-1 and all 3^{height} different distributions of disks are traversed (when including the starting and final distribution). This is called a Hamilton path. For this solution the disk to be moved can be found with a ternary gray code in a similar way as explained for the shortest solution. In fact there is a ternary Gray code starting with all digits 0 and ending with all digits equal 2, that lists the successive distributions of disks of a Hamilton path from peg 0 to peg 2 for a tower of h disks, each code showing the positions of the disks in decreasing order of size when read from left to right. This Gray code is uniquely defined by imposing the extra condition that each digit is switched more often than each more significant digit on the left. This is the code needed for the Tower of Hanoi.

```
(define (number->p-ary-gray-code n h p) ; n h p --> n-th h digit p-ary gray-code
  (let ((2p (* 2 p)))
   (let loop ((n n) (h h) (gc ()))
    (if (zero? h) gc
     (let ((q (quotient n p)) (r (modulo n 2p)))
      (loop q (sub1 h) (cons (if (>= r p) (- 2p r 1) r) gc)))))))

 (define (p-ary-gray-code->number gc p) ; n-th p-ary gray-code --> n
  (let loop ((gc gc) (significance (expt p (sub1 (length gc)))))
   (if (null? gc) 0
    (let ((digit (car gc)) (gc (cdr gc)))
     (let ((n (loop gc (quotient significance p))))
      (+ (* digit significance)
       (if (odd? digit) (- significance n 1) n)))))))

 (define (number->hanoian-gray-code n h) (number->p-ary-gray-code n h 3))
 (define (hanoian-gray-code->number gc) (p-ary-gray-code->number gc 3))
```

The disk to be moved is determined by the number of times the move counter can be divided by 3. Where the disk is to be moved to can easily be determined too. For every triplet of moves, move the smallest disk twice in succession in the same direction, followed by a move of one of the larger disks (only one direction possible) Between every triplet of moves reverse the direction of the smallest disk. The very first move of the smallest disk is to be made from the starting peg onto the remaining third peg. Also observe that the unused peg of each move alternates between the starting and destination peg. These statements are easily proven by mathematical induction.

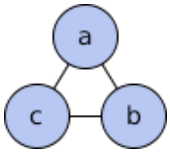
```
(define (exp3 n        ) (expt   3 n))
(define (mod3 n         ) (modulo n 3))
(define (mod4 n         ) (modulo n 4))
(define (mcnt m   d     ) (+ (* 2 (quotient m (exp3 (add1 d)))) (mod3 (quotient m (exp3 d)))))
(define (thrd m h   f t) (if (odd? m) t f))
(define (onto m h   f t) (posi m h (disk m) f t))
(define (from m h   f t) (- 3 (onto m h f t) (thrd m h f t)))
(define (posi m h d f t) (vector-ref (vector f (- 3 f t) t (- 3 f t)) (mod4 (mcnt m d))))
(define (disk m) (if (zero? (mod3 m)) (add1 (disk (quotient m 3))) 0))
```

Another modification is to move a tower from a peg back to the same peg while traversing all distributions of disks. (circular Hamilton path) There are exactly two solutions, but they mirror each other in the sense that there is in fact one path that can be traversed in both directions. Obviously, the length of the path is 3^{height}. A simple algorithm for the circular Hamilton path is:

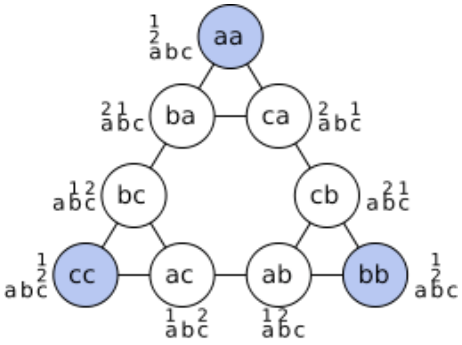
```
(define (circular-hamilton-move-tower h a b c) ; h=height. a, b and c are the three pegs.
 (if (positive? h) ; start with a tower at peg a, move tower to peg b, then to peg c and finally re
  (let ((h-1 (sub1 h)))
   (hamilton-start  h-1 a c b)   ; The largest disk is moved three times.
   (move-disk       h-1 a b c)   ; Between these moves the longest non selfcrossing path is used in
   (long-move-tower h-1 c a b)   ; partial tower consisting of the h-1 smaller disks from one peg o
   (move-disk       h-1 b c a)   ; Together the procedures hamilton-start and hamilton-finish make
   (long-move-tower h-1 a b c)   ; non selfcrossing path too. The moves made by hamiton-finish foll
   (move-disk       h-1 c a b)   ; those of hamilton-start form a longest non selfcrossing path for
   (hamilton-finish h-1 b a c)))); partial tower from peg b to peg c.

(define (hamilton-start h a b c)
 (if (positive? h)
  (let ((h-1 (sub1 h)))
   (hamilton-start  h-1 a c b)
   (move-disk       h-1 a b c)
   (long-move-tower h-1 c b a))))

(define (hamilton-finish h a b c)
 (if (positive? h)
  (let ((h-1 (sub1 h)))
   (long-move-tower h-1 a c b)
   (move-disk       h-1 a b c)
   (hamilton-finish h-1 c b a))))
```

Graphical representation

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



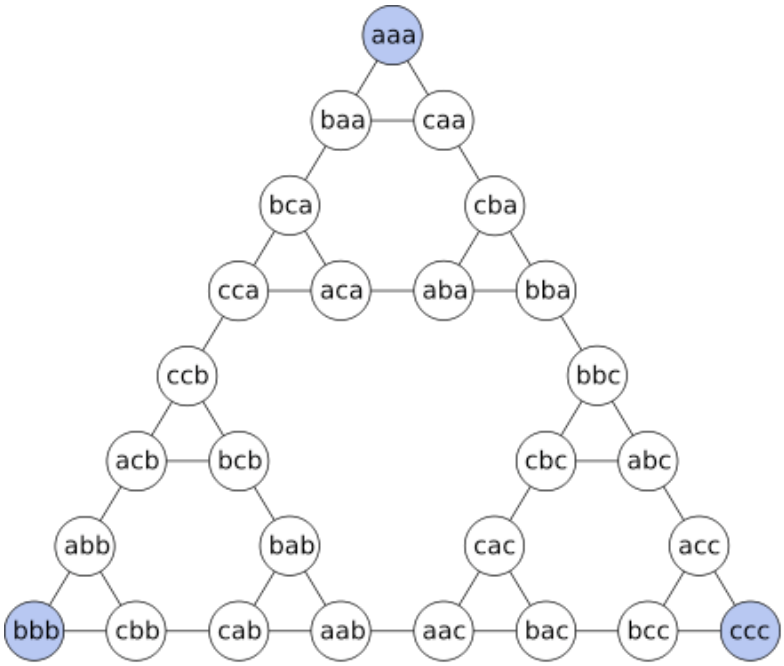
The graph for 2 disks is 3 triangles arranged in a larger triangle:



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 2 disks.

For 3 disks the graph is:



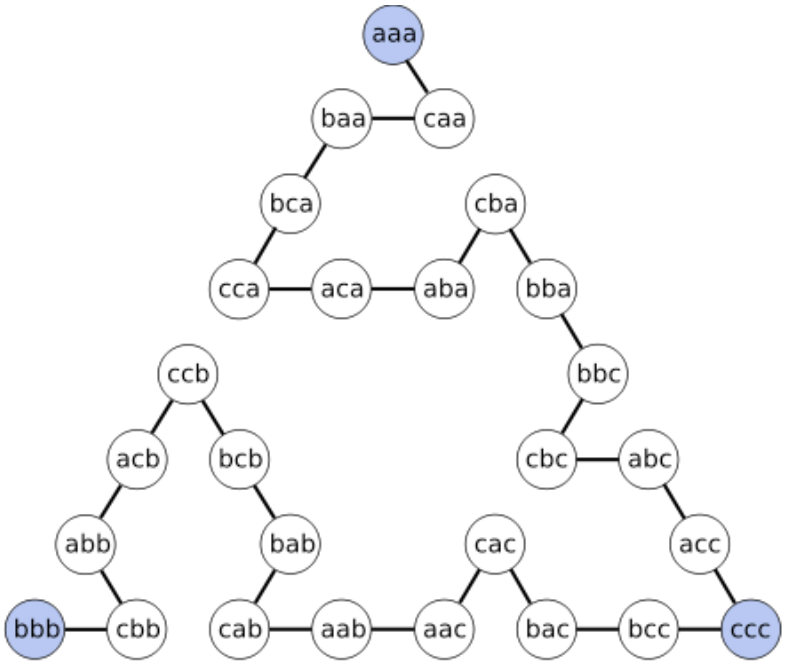
- call the pegs a, b and c
- list disk positions from left to right in order of increasing size

The sides of the outermost triangle represent the shortest ways of moving a tower from one peg to another one. The branch in the middle of the sides of the largest triangle represents a move of the largest disk. The branch in the middle of the sides of each next smaller triangle represents a move of each next smaller disk. The sides of the smallest triangles represent moves of the smallest disk.

In general, for a puzzle with n disks, there are 3^n nodes in the graph; every node has three branches to other nodes, except the three corner nodes, which have two: it is always possible to move the smallest disk to the one of the two other pegs; and it is possible to move one disk between those two pegs *except* in the situation where all disks are stacked on one peg. The corner nodes represent the three cases where all the disks are stacked on one peg. The diagram for n+1 disks is obtained by taking three copies of the n-disk diagram -- each one representing all the states and moves of the smaller disks for one particular position of the new largest disk -- and joining them at the corners with three new branches, representing the only three opportunities to move the largest disk. The resulting figure thus has 3^{n+1} nodes and still has three corners remaining with only two branches.

As more disks are added, the graph representation of the game will resemble the Fractal figure, Sierpiński triangle. It is clear that the great majority of positions in the puzzle will never be reached when using the shortest possible solution; indeed, if the priests of the legend are using the longest possible solution (without re-visiting any position) it will take them 3^{64}-1 moves, or more than 10^{23} years.

The longest non-repetitive way for three disks can be visualized by erasing the unused branches:



The circular Hamiltonian path for three disks is:

```
                ccc
                 /\          ; call the pegs a, b and c
            acc /  \ bcc     ; list disk positions from left to right in order of increasing size
               /    \
          abc /      \ bac
              \      /
       bbc  __\    /__  aac
         /  cbc  cac  \
     bba /              \ aab
         \              /
   cba  __\aba      bab/__  cab
      /                    \
  caa /       aca  bcb       \ cbb
     /       /\      /\       \
aaa /__  __ /  \ __ /  \ __  __\ bbb
      baa  bca cca ccb acb abb
```

The graphs clearly show that:

- From every arbitrary distribution of disks, there is exactly one shortest way to move all disks onto one of the three pegs.
- Between every pair of arbitrary distributions of disks there are one or two different shortest paths.
- From every arbitrary distribution of disks, there are one or two different longest non selfcrossing paths to move all disks to one of the three pegs.
- Between every pair of arbitrary distributions of disks there are one or two different longest non selfcrossing paths.
- Let N_h be the number of non selfcrossing paths for moving a tower of h disks from one peg to another one. Then:
  - $N_1=2$
  - $N_{h+1}=(N_h)^2+(N_h)^3$.
  - For example: $N_8 \approx 1.5456 \times 10^{795}$

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.

The Tower of Hanoi is also used as 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 memory test by neuropsychologists trying to evaluate amnesia.

Four pegs and beyond

Although the three-peg version has a simple recursive solution as outlined above, the *optimal* solution for the Tower of Hanoi problem with four or more pegs is still an open problem. This is a good example of how a simple, solvable problem can be made dramatically more difficult by slightly loosening one of the problem constraints.

The fact that the problem with four or more pegs is an open problem does not imply that no algorithm exists for finding (all of) the optimal solutions. Simply represent the game by an undirected graph, the nodes being distributions of disks and the edges being moves (of length 1) and use Dijkstra's algorithm to find one (or all) shortest paths moving a tower from one peg onto another one. However, even smartly implemented on the fastest computer now available, this algorithm provides no way of effectively computing solutions for large numbers of disks; the program would require more time and memory than available. Hence, even having an algorithm, it remains unknown how many moves an optimal solution requires and how many optimal solutions exist for 1000 disks and 10 pegs.

Though it is not known exactly how many moves must be made, there are some asymptotic results. There is also a "presumed-optimal solution" that can be recursively applied to find a solution - see Paul Stockmeyer's survey paper (http://www.cs.wm.edu/~pkstoc/boca.ps) for an explanation and some variants of the four-peg problem.

Although it agrees with computer experiments for small numbers of disks, there is not yet a general proof that this presumed-optimal solution is in fact optimal. However, results in 2004 (http://epubs.siam.org/sam-bin/dbq/article/43101) showed that the presumed-optimal solution must be of the same order of magnitude as the optimal solution.

Description of the presumed-optimal solution

The problem for four pegs is sometimes called "Reve's puzzle". *A* solution for four (or more) pegs, which has not been proved to be optimal, is described below:

- Let n be the number of disks.
- Let r be the number of pegs.
- Define $T(n,r)$ to be the number of moves required to transfer n disks using r pegs

The algorithm can be described recursively:

1. For some k, $1 \leq k < n$, transfer the top k disks to a single other peg, taking $T(k,r)$ moves.
2. Without disturbing the peg that now contains the top k disks, transfer the remaining $n - k$ disks to the destination peg, using only the remaining $r - 1$ pegs, taking $T(n - k,r - 1)$ moves.
3. Finally, transfer the top k disks to the destination peg, taking $T(k,r)$ moves.

The entire process takes $2T(k,r) + T(n - k,r - 1)$ moves. Therefore, the count k should be picked for which this quantity is minimum.
This algorithm (with the above choice for k) is presumed to be optimal, and no counterexamples are known.

Two stacks and more stacks

A 2007 U.S. patent application purportedly discloses multistack Tower of Hanoi puzzles [Feb. 1, 2007, Serial

No.11/701,454] with two or more stacks and twice as many pegs as stacks. After beginning on a particular peg, each stack displaces and is displaced by a different colored stack on another peg when the puzzle is solved. Disks of one color also have another peg that excludes all other colors, so that there are three pegs available for each color disk, two that are shared with other colors and one that is not shared. On the shared pegs, a disk may not be placed on a different colored disk of the same size, a possibility that does not arise in the standard puzzle.

The simplest multistack game (2 x 4) has two stacks and four pegs, and it requires 3[T(n)] moves to solve where T(n) is the number of moves needed to solve a single stack classic of n disks. The game proceeds in seesaw fashion with longer and longer series of moves that alternate between colors. It concludes in reverse seesaw fashion with shorter and shorter such series of moves. Starting with the second series of three moves, these alternate series of moves double in length for the first half of the game, and the lengths are halved as the game concludes. The solution involves nesting an algorithm suitable for Tower of Hanoi into an algorithm that indicates when to switch between colors. When there are k stacks of n disks apiece in a game, and $k > 2$, it requires $k[T(n)] + T(n-1)$ moves to relocate them.

The addition of a centrally located universal peg open to disks from all stacks converts these multistack Tower of Hanoi puzzles to multistack Reve's puzzles as described in the preceding section. In these games each stack may move among four pegs, the same combination of three in the 2 x 4 game plus the central universal peg. The simplest game of this kind (2 x 5) has two stacks and five pegs. A solution conjectured to be optimal interlocks the optimal solution of the 2 x 4 puzzle with the presumed optimal solution to Reve's puzzle. It takes $R(n) + 2R(n-1) + 2$ moves, where R(n) is the number of moves in the presumed optimal Reve's solution for a stack of n disks.

In popular culture

In the classic science fiction story *Now Inhale*, by Eric Frank Russell (*Astounding Science Fiction* April 1959, and in various anthologies), the human hero is a prisoner on a planet where the local custom is to make the prisoner play a game until it is won or lost, and then execution is immediate. The hero is told the game can be one of his own species', as long as it can be played in his cell with simple equipment strictly according to rules which are written down before and cannot change after play starts, and it has a finite endpoint. The game and execution are televised planet-wide, and watching the desperate prisoner try to spin the game out as long as possible is very popular entertainment; the record is sixteen days. The hero knows a rescue ship might take a year or more to arrive, so chooses to play Towers of Hanoi with 64 disks until rescue arrives. When the locals realize they've been had, they are angry, but under their own rules there is nothing they can do about it. They do change the rules, which will apply to any *future* prisoners. This story makes reference to the legend about the Buddhist monks playing the game until the end of the world, and refers to the game as **arkymalarky**. (The slang term "malarky", meaning nonsense, pre-dates this story by at least 30 years. [2] (http://www.phrases.org.uk/bulletin_board/7/messages/555.html))

In the movie Stranger than Fiction, a miniature Towers of Hanoi puzzle can be seen on the cluttered desk of Professor Jules Hilbert.

In the *Doctor Who* serial "The Celestial Toymaker", the Toymaker challenges the Doctor to complete the "Trilogic Game" (ten disk Hanoi) in exactly 1,023 ($2^{10} - 1$) moves.

There is a band named Towers of Hanoi (http://www.towersofhanoi.org) .

In video games

The puzzle is featured regularly in adventure and puzzle games. Since it is easy to implement, and easily-recognised, it is well-suited to use as a puzzle in a larger graphical game. Some implementations use straight disks, but others disguise the puzzle in some other form. What follows is a partial list of games which use the puzzle:

- Black & White
- The Island of Dr. Brain
- The Secret Island of Dr. Quandary

- Star Wars: Knights of the Old Republic
- Zork Zero
- The Legend of Kyrandia: Hand of Fate
- Escape from Paradise
- ECHO: Secrets of the Lost Cavern
- Clan Lord
- Mass Effect
- Azada

See also

- Baguenaudier
- Recursion (computer science)

Notes

1. **^** *1000 Play Thinks*.

External links

Online demonstrations

- *Towers of Hanoi (http://demonstrations.wolfram.com/TowersOfHanoi/)* by Jay Warendorff based on a program by Jaime Rangel-Mondragón, The Wolfram Demonstrations Project.
- A simple online demonstration. (http://www.alexmaslin.org/toh.html)

Algorithm

- Theory, recursive and a per step algorithms with Java implementation (http://www.cut-the-knot.org/Curriculum/Combinatorics/TowerOfHanoi.shtml) at cut-the-knot
- Towers Of Hanoi in c++ using Stacks (http://24bytes.com/Towers-Of-Hanoi.html) at 24bytes
- Bicolor Tower of Hanoi (http://www.cut-the-knot.org/Curriculum/Combinatorics/BiColorTowerOfHanoi.shtml) at cut-the-knot
- Hanoimania, implementations of the Tower of Hanoi problem in over 100 different programming environments and languages (http://www.kernelthread.com/hanoi/)
- Towers of Hanoi Online game (http://math.bu.edu/DYSYS/applets/hanoi.html)
- Binary Numbers and the Standard Gray Code, with a discussion of the Gray Code solution (http://occawlonline.pearsoned.com/bookbind/pubbooks/miller2_awl/chapter4/essay1/deluxe-content.html#tower)
- A variety of Tower of Hanoi Algorithms in PLT MzScheme (http://www.telefonica.net/web2/koot/hanoi.doc)
- Solution in an all web-based interactive classical BASIC environment (http://www.quitebasic.com/prj/puzzle/towers-of-hanoi/)
- Two non-recursive solutions in C++ (http://www.datastructures.info/the-towers-of-hanoi/)

Events

- [3] (http://www-mat.pfmb.uni-mb.si/toh2005/) Workshop on the Tower of Hanoi and Related Problems, 18.9. - 22.9.2005, Maribor, Slovenia

Retrieved from "http://en.wikipedia.org/wiki/Tower_of_Hanoi"
Categories: Mechanical puzzles | Articles with example code | Articles with example Haskell code | Articles with example Scheme code | Articles with example Pascal code | Articles with example Python code | Articles with example C code