March 5, 2006 at 13:36

1. Introduction. The tower of Hanoi (commonly also known as the "*towers* of Hanoi"), is a puzzle invented by E. Lucas in 1883. Given a stack of disks arranged from largest on the bottom to smallest on top placed on a rod, together with two empty rods, the towers of Hanoi puzzle asks for the minimum number of moves required to move the stack from one rod to another, where moves are allowed only if they place smaller disks on top of larger disks. The puzzle with $n = 4$ pegs and n disks is sometimes known as Reve's puzzle.

The problem is isomorphic to finding a Hamiltonian path on an n-hypercube (Gardner 1957, 1959).

Given three rods and n disks, the sequence $S_1 = \{a_k\}$ giving the number of the disk ($i = 1$ to n) to be moved at the kth step is given by the remarkably simple recursive procedure of starting with the list $S_1 = \{1\}$ for a single disk, and recursively computing

$$S_n = \{S_{n-1}, n, S_{n-1}\}$$

For the first few values of n, this gives the sequences shown in the following table. A solution of the three-rod four-disk problem is illustrated above.

n	S_n
1	1
2	1, 2, 1
3	1, 2, 1, 3, 1, 2, 1
4	1, 2, 1, 3, 1, 2, 1, 4, 1, 2, 1, 3, 1, 2, 1

As the number of disks is increases (again for three rods), an infinite sequence is obtained, the first few terms of which are illustrated in the table above (Sloane's A001511). Amazingly, this is exactly the binary carry sequence plus one. Even more amazingly, the number of disks moved after the kth step is the same as the element which needs to be added or deleted in the kth addend of the Ryser formula (Gardner 1988, Vardi 1991).

As a result of the above procedure, the number of moves h_n required to solve the puzzle of n disks on three rods is given by the recurrence relation

$$h_n = 2h_{n-1} + 1$$

with $h_1 = 1$. Solving gives

$$h_n = 2^n - 1$$

i.e., the Mersenne numbers.

For three rods, the proof that the above solution is minimal can be achieved using the Lucas correspondence which relates Pascal's triangle to the Hanoi graph. While algorithms are known for transferring disks on four rods, none has been proved minimal.

A Hanoi graph can be constructed whose graph vertices correspond to legal configurations of towers of Hanoi, where the graph vertices are adjacent if the corresponding configurations can be obtained by a legal move. The puzzle itself can be solved using a binary Gray code.

Poole (1994) and Rangel-Mondragn give Mathematica routines for solving the Hanoi towers problem. Poole's algorithm works for an arbitrary disk configuration, and provides the solution in the fewest possible moves.

The minimal numbers of moves required to order $n = 1, 2, \ldots$ disk on four rods are given by 1, 3, 5, 9, 13, 17, 25, 33, … (Sloane's A007664). It is conjectured that this sequence is given by the recurrence

$$s_n = s_{n-1} + 2^x$$

with $s_1 = 1$ and x the positive floor integer solution to

$$n - 1 = \frac{1}{2}x(x+1),$$

i.e.,

$$x = \left\lfloor \frac{\sqrt{8n-7} - 1}{2} \right\rfloor.$$

This would then give the explicit formula

$$s_n = 1 + \left[n - \frac{1}{2}x(x-1) - 1 \right] 2^x.$$

This program solves the problem of Hanoi Tower in non-recursive version. Here is the overall layout of this C program:

```
#include <stdio.h>
#include <stdlib.h>
  ⟨ Type declarations 2 ⟩
  ⟨ Global variables 3 ⟩
  ⟨ Functions 4 ⟩
  main(argc, argv)
      int argc;      /* the number of command-line arguments */
      char *argv[];      /* an array of strings containing those arguments */
  {
    if (argc ≠ 2) {
      printf("Tower␣of␣Hanoi␣program.␣Non-recursive␣version.\n");
      printf("\tThere␣is␣three␣rods.(A,␣B,␣and␣C).");
      printf("␣Move␣the␣stack␣from␣A␣to␣C.\n");
      printf("Usage:␣hanoi␣<number␣of␣disks>\n");
      return 0;
    }
    hanoi(atoi(argv[1]), 'A', 'B', 'C');
    return 0;
  }
```

2. Stack. The stack is implemented by linked list.

⟨ Type declarations 2 ⟩ ≡

```
struct _node {
  void ∗element;
  struct _node ∗next;
};
```

See also section 8.

This code is used in section 1.

3.

⟨ Global variables 3 ⟩ ≡

```
struct _node ∗head = Λ;
```

This code is used in section 1.

4. Alloc a node for stack.

⟨ Functions 4 ⟩ ≡

```
static struct _node ∗talloc( )
{
  return (struct _node ∗) malloc(sizeof(struct _node));
}
```

See also sections 5, 6, 7, 9, and 10.

This code is used in section 1.

5. Initialize and deallocate a stack.

⟨ Functions 4 ⟩ +≡

```
void init_stack( )
{
  if (¬head) {
    head = talloc( );
    head→next = Λ;
  }
}
void free_stack( )
{
  struct _node ∗p;
  while (head) {
    p = head;
    head = p→next;
    free(p);
  }
}
```

6. Push and pop operation

```
⟨ Functions 4 ⟩ +≡
  void push(t)
      void *t;
  {
    struct _node *p;
    p = talloc( );
    p→element = t;
    p→next = head→next;
    head→next = p;
  }
  void *pop( )
  {
    void *value;
    struct _node *p;
    if (¬stackempty( )) {
      p = head→next;
      head→next = p→next;
      value = p→element;
      free(p);
      return value;
    }
    else {
      return Λ;
    }
  }
```

7.

```
⟨ Functions 4 ⟩ +≡
  int stackempty( )
  {
    return head→next ≡ Λ;
  }
```

8. Hanoi Tower.

```
⟨ Type declarations 2 ⟩ +≡
  struct _disk {
    int n, no;
    char from, via, to;
  };
```

9. Make a disk.

```
⟨ Functions 4 ⟩ +≡
  struct _disk ∗disk(no, n, from, via, to)
      int n, no;
      char from, via, to;
  {
    struct _disk ∗p;
    p = (struct _disk ∗) malloc(sizeof(struct _disk));
    p→no = no;
    p→n = n;
    p→from = from;
    p→via = via;
    p→to = to;
    return p;
  }
```

10. The main routine to solve the problem using stack.

```
⟨ Functions 4 ⟩ +≡
  void hanoi(n, from, via, to)
      int n;    /∗ the number of disks ∗/
      char from, via, to;    /∗ rods ∗/
  {
    struct _disk ∗p;
    init_stack( );
    push(disk(n, n, from, via, to));
    while (¬stackempty( )) {
      p = (struct _disk ∗) pop( );
      if (p→n > 1)
        ⟨ Push the processes in reverse order 11 ⟩
      else
        printf("Move␣disc␣%d:␣%c␣==>␣%c\n", p→no, p→from, p→to);
      free(p);
    }
    free_stack( );
  }
```

11.

```
⟨ Push the processes in reverse order 11 ⟩ ≡
  {
    push(disk(p→no − 1, p→n − 1, p→via, p→from, p→to));
    push(disk(p→no, 1, p→from, p→via, p→to));
    push(disk(p→no − 1, p→n − 1, p→from, p→to, p→via));
  }
```

This code is used in section 10.

12. Index.

HANOI