Towers of Hanoi

The Tower of Hanoi is a classic mathematical puzzle that 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 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: 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.
2. Only the topmost disk can be moved: You cannot move a disk that is below another disk. Only the disk that is visible at the top of one of the stacks can be moved in a turn.
3. No disk may be placed on top of a smaller disk: This is a crucial rule. You cannot place a larger disk onto a smaller disk. Disks must always be stacked in order of decreasing size from bottom to top.

Introduction

Let's denote the set of disks as $D = \{1,2 \dots, n\}$, where 1 represents the smallest disk and *n* the largest. The rods can be represented as the set $T = \{1,2,3\}$ A move is a step in the process, denoted by $m \in Z^+$. Total number of moves required for solving puzzle is $2^n - 1$.

Tower Function (`Tower`):

Tower (t, p, m) is represented as:

$$Tower\colon T \times Z^+ \times Z^+ \to D \cup \{0\}$$

For a given tower *t*, position *p*, and move *m*, *Tower*(*t*,*p*,*m*) returns the size of the disk at that position and move, or 0 if no disk is present.

Disks Function (`disks`)

disks (t, m) is represented as:

$$disks\colon T \times Z^+ \to Z^+$$

For a given tower *t* and move *m*, *disks*(*t*,*m*) returns the number of disks on tower *t* at move *m*.

Move Function (`move`)

*The **move(t1, t2, t3, m)** function defines the conditions for a valid move between two towers at a given step. It can be expressed as:*

$$move: T \times T \times T \times Z^{+} \rightarrow \{true, false\}$$

This indicates that the **move** function takes four arguments: three towers (denoted as *t*1, *t*2, and *t*3) from the set of towers *T*, and a move number *m* from the set of positive integers Z^{+}. It returns a boolean value indicating whether the move is possible (true) or impossible (false)

Constraints Enforced by the Move Function

Disk Count Adjustment:

The function adjusts the count of disks on the source tower *t*1 and the destination tower *t*2 after the move:

$$disks(t1, m+1) = disks(t1, m) - 1$$
$$disks(t2, m+1) = disks(t2, m) + 1$$

No Change In Third Tower:

The function ensures that the disk being moved is smaller than the top disk (if any) on the destination tower:

$$Tower(t1, disks(t1, m), m) < Tower(t2, disks(t2, m), m)$$

This rule is crucial as it enforces the constraint that a larger disk cannot be placed on top of a smaller disk.

Towers Configuration Update:

The function ensures that each tower satisfy the conditions:

$$\forall p \in Z^{+}: (1 \leq p \leq n) \rightarrow$$
$$(Tower(t1, p, m+1) = \begin{cases} 0, & \text{p} = \text{disks(t1, m)} \\ Tower(t1, p, m), & \text{else} \end{cases} \wedge$$
$$Tower(t2, p, m+1) = \begin{cases} \text{Tower(t1, disks(t1, m)m)}, & p = disks(t2, m+1) \\ Tower(t2, p, m), & \text{else} \end{cases} \wedge$$
$$Tower(t3, p, m) = Tower(t3, p, m+1))$$

Formal Definition

The formal definition of the **move** function, incorporating all the conditions mentioned above, is as follows:

$$move(t1, t2, t3, m) \equiv disks(t1, m+1) = disks(t1, m) - 1$$
$$\wedge\, disks(t2, m+1) = disks(t2, m) + 1$$
$$\wedge disks(t3, m) = disks(t3, m)$$
$$\wedge Tower(t1, disks(t1, m), m) < Tower(t2, disks(t2, m), m)$$
$$\forall p \in Z^{+}: (1 \le p \le n) \rightarrow$$
$$(Tower(t1, p, m+1) = \begin{cases} 0, & p = disks(t1, m) \\ Tower(t1, p, m), & else \end{cases}$$
$$\wedge\, Tower(t2, p, m+1) = \begin{cases} Tower(t1, disks(t1, m)m), & p = disks(t2, m+1) \\ Tower(t2, p, m), & else \end{cases}$$
$$\wedge\, Tower(t3, p, m) = Tower(t3, p, m+1))$$

Disk Count Constraint

This assertion ensures that the number of disks on each tower at each step does not exceed the total number of disks *n* and is represented as:

$$\forall t \in T, \forall m \in Z^{+}: (0 \le m < 2^{n} - 1) \rightarrow (0 \le disks(t, m) \le n)$$

Possible Moves Assertion

This assertion ensures that for each step, there exists a valid move that follows the puzzle's rules:

$$\forall i \in Z+: (0 \le i < 2n - 1) \rightarrow \exists t1, t2, t3 \in T: (t1/t2/t3) \wedge move(t1, t2, t3, i)$$

Start State Assertions

The start state is defined using assertions to establish the initial configuration of the disks on the towers. For instance,

for 3 disks:

$$
\begin{aligned}
&Tower(1,1,0) = 3 \\
&\wedge Tower(1,2,0) = 2 \\
&\wedge Tower(1,3,0) = 1 \\
&\wedge disks(1,0) = 3 \\
&\wedge disks(2,0) = 0 \\
&\wedge disks(3,0) = 0 \\
&\wedge \forall t \in \{2,3\} \forall p \in \{1,2,3\} : Tower(t,p,0) = 0
\end{aligned}
$$

for 4 disks:

$$
\begin{aligned}
&Tower(1,1,0) = 4 \\
&\wedge Tower(1,2,0) = 3 \\
&\wedge Tower(1,3,0) = 2 \\
&\wedge Tower(1,4,0) = 1 \\
&\wedge disks(1,0) = 4 \\
&\wedge disks(2,0) = 0 \\
&\wedge disks(3,0) = 0 \\
&\wedge \forall t \in \{2,3\} \forall p \in \{1,2,3,4\} : Tower(t,p,0) = 0
\end{aligned}
$$

End State Assertions

The end state assertions define the final desired configuration of the disks on the towers.

for 3 disks:

$$
\begin{aligned}
&Tower(3,1,7) = 3 \\
&\wedge Tower(3,2,7) = 2 \\
&\wedge Tower(3,3,7) = 1 \\
&\wedge disks(1,7) = 0 \\
&\wedge disks(2,7) = 0 \\
&\wedge disks(3,7) = 3 \\
&\wedge \forall t \in \{1,2\} \forall p \in \{1,2,3\} : Tower(t,p,7) = 0
\end{aligned}
$$

for 4 disks:

$$
\begin{aligned}
&Tower(1,1,15) = 4 \\
&\wedge Tower(1,2,15) = 3 \\
&\wedge Tower(1,3,15) = 2 \\
&\wedge Tower(1,4,15) = 1 \\
&\wedge disks(1,15) = 0 \\
&\wedge disks(2,15) = 0 \\
&\wedge disks(3,15) = 4 \\
&\wedge \forall t \in \{2,3\} \forall p \in \{1,2,3,4\} : Tower(t,p,15) = 0
\end{aligned}
$$

Z3 SMT Solution

3 disks

Code:

```
((declare-fun Tower(Int Int Int)Int)
(declare-fun disks(Int Int) Int)
;;Start State
(assert
   (and
      (=
         (disks 1 0)
         3
      )
      (=
         (disks 2 0)
         0
      )
      (=
         (disks 3 0)
         0
      )
      (=
         (Tower 1 1 0)
         3
      )
      (=
         (Tower 1 2 0)
         2
      )
      (=
         (Tower 1 3 0)
         1
      )
```

```
      (forall ((t Int) (p Int))
         (implies
            (and
               (<= 2 t 3)
               (<= 1 p 3)
            )
            (=
               (Tower t p 0)
               0
            )
         )
      )
   )
)
;;End State
(assert
   (and
      (=
         (disks 1 7)
         0
      )
      (=
         (disks 2 7)
         0
      )
      (=
         (disks 3 7)
         3
      )
      (=
         (Tower 3 1 7)
         3
      )
      (=
         (Tower 3 2 7)
         2
```

```
      )
      (=
        (Tower 3 3 7)
        1
      )
      (forall ((t Int) (p Int))
        (implies
          (and
            (<= 1 t 2)
            (<= 1 p 3)
          )
          (=
            (Tower t p 7)
            0
          )
        )
      )
    )
)
(define-fun move((t1 Int) (t2 Int) (t3 Int) (m Int)) Bool
  (and
    (=
      (disks t1 (+ m 1))
      (-
        (disks t1 m)
        1
      )
    )
    (=
      (disks t2 (+ m 1))
      (+
        (disks t2 m)
        1
      )
    )
    (=
```

```
        (disks t3 (+ m 1))
        (disks t3 m)
      )
      (<
        (Tower t1 (disks t1 m) m)
        (Tower t2 (disks t2 m) m)
      )
      (forall ((p Int))
        (implies
          (<= 1 p 3)
          (and
            (=
              (Tower t1 p (+ m 1))
              (ite
                (= p (disks t1 m))
                0
                (Tower t1 p m)
              )
            )
            (=
              (Tower t2 p (+ m 1))
              (ite
                (= p (disks t2 (+ m 1)))
                (Tower t1 (disks t1 m) m)
                (Tower t2 p m)
              )
            )
            (=
              (Tower t3 p m)
              (Tower t3 p (+ m 1))
            )
          )
        )
      )
    )
)
```

```
(assert
   (forall ((t Int) (m Int))
      (implies
         (and
            (<= 1 t 3)
            (<= 0 m 6)
         )
         (<= 0 (disks t m) 3)
      )
   )
)
(assert (forall ((i Int))
   (implies
      (<= 0 i 6)
      (exists ((t1 Int) (t2 Int) (t3 Int))
         (and
            (<= 1 t1 3)
            (<= 1 t2 3)
            (<= 1 t3 3)
            (distinct t1 t2 t3)
            (move t1 t2 t3 i)
         )
      )
   )
))
(check-sat)
(echo "Step 0:")
(get-value(
   (Tower 1 1 0)
   (Tower 1 2 0)
   (Tower 1 3 0)
   (Tower 2 1 0)
   (Tower 2 2 0)
   (Tower 2 3 0)
   (Tower 3 1 0)
   (Tower 3 2 0)
```

```
    (Tower 3 3 0)
))
     (echo "Step 1:")
(get-value(
    (Tower 1 1 1)
    (Tower 1 2 1)
    (Tower 1 3 1)
    (Tower 2 1 1)
    (Tower 2 2 1)
    (Tower 2 3 1)
    (Tower 3 1 1)
    (Tower 3 2 1)
    (Tower 3 3 1)
))
     (echo "Step 2:")
(get-value(
    (Tower 1 1 2)
    (Tower 1 2 2)
    (Tower 1 3 2)
    (Tower 2 1 2)
    (Tower 2 2 2)
    (Tower 2 3 2)
    (Tower 3 1 2)
    (Tower 3 2 2)
    (Tower 3 3 2)
))
     (echo "Step 3:")
(get-value(
    (Tower 1 1 3)
    (Tower 1 2 3)
    (Tower 1 3 3)
    (Tower 2 1 3)
    (Tower 2 2 3)
    (Tower 2 3 3)
    (Tower 3 1 3)
    (Tower 3 2 3)
```

```
    (Tower 3 3 3)
))
     (echo "Step 4:")
(get-value(
    (Tower 1 1 4)
    (Tower 1 2 4)
    (Tower 1 3 4)
    (Tower 2 1 4)
    (Tower 2 2 4)
    (Tower 2 3 4)
    (Tower 3 1 4)
    (Tower 3 2 4)
    (Tower 3 3 4)
))
     (echo "Step 5:")
(get-value(
    (Tower 1 1 5)
    (Tower 1 2 5)
    (Tower 1 3 5)
    (Tower 2 1 5)
    (Tower 2 2 5)
    (Tower 2 3 5)
    (Tower 3 1 5)
    (Tower 3 2 5)
    (Tower 3 3 5)
))
     (echo "Step 6:")
(get-value(
    (Tower 1 1 6)
    (Tower 1 2 6)
    (Tower 1 3 6)
    (Tower 2 1 6)
    (Tower 2 2 6)
    (Tower 2 3 6)
    (Tower 3 1 6)
    (Tower 3 2 6)
```

```
    (Tower 3 3 6)
))
     (echo "Step 7:")
(get-value(
    (Tower 1 1 7)
    (Tower 1 2 7)
    (Tower 1 3 7)
    (Tower 2 1 7)
    (Tower 2 2 7)
    (Tower 2 3 7)
    (Tower 3 1 7)
    (Tower 3 2 7)
    (Tower 3 3 7)
))
```

Output:

```
Step 0:
(((Tower 1 1 0) 3)
 ((Tower 1 2 0) 2)
 ((Tower 1 3 0) 1)
 ((Tower 2 1 0) 0)
 ((Tower 2 2 0) 0)
 ((Tower 2 3 0) 0)
 ((Tower 3 1 0) 0)
 ((Tower 3 2 0) 0)
 ((Tower 3 3 0) 0))
Step 1:
(((Tower 1 1 1) 3)
 ((Tower 1 2 1) 2)
 ((Tower 1 3 1) 0)
 ((Tower 2 1 1) 0)
 ((Tower 2 2 1) 0)
 ((Tower 2 3 1) 0)
 ((Tower 3 1 1) 1)
 ((Tower 3 2 1) 0)
 ((Tower 3 3 1) 0))
Step 2:
(((Tower 1 1 2) 3)
 ((Tower 1 2 2) 0)
 ((Tower 1 3 2) 0)
 ((Tower 2 1 2) 2)
 ((Tower 2 2 2) 0)
```

((Tower 2 3 2) 0)
((Tower 3 1 2) 1)
((Tower 3 2 2) 0)
((Tower 3 3 2) 0))
Step 3:
(((Tower 1 1 3) 3)
((Tower 1 2 3) 0)
((Tower 1 3 3) 0)
((Tower 2 1 3) 2)
((Tower 2 2 3) 1)
((Tower 2 3 3) 0)
((Tower 3 1 3) 0)
((Tower 3 2 3) 0)
((Tower 3 3 3) 0))
Step 4:
(((Tower 1 1 4) 0)
((Tower 1 2 4) 0)
((Tower 1 3 4) 0)
((Tower 2 1 4) 2)
((Tower 2 2 4) 1)
((Tower 2 3 4) 0)
((Tower 3 1 4) 3)
((Tower 3 2 4) 0)
((Tower 3 3 4) 0))
Step 5:
(((Tower 1 1 5) 1)
((Tower 1 2 5) 0)
((Tower 1 3 5) 0)
((Tower 2 1 5) 2)
((Tower 2 2 5) 0)
((Tower 2 3 5) 0)
((Tower 3 1 5) 3)
((Tower 3 2 5) 0)
((Tower 3 3 5) 0))
Step 6:
(((Tower 1 1 6) 1)
((Tower 1 2 6) 0)
((Tower 1 3 6) 0)
((Tower 2 1 6) 0)
((Tower 2 2 6) 0)
((Tower 2 3 6) 0)
((Tower 3 1 6) 3)
((Tower 3 2 6) 2)
((Tower 3 3 6) 0))
Step 7:
(((Tower 1 1 7) 0)
((Tower 1 2 7) 0)
((Tower 1 3 7) 0)
((Tower 2 1 7) 0)

```
((Tower 2 2 7) 0)
((Tower 2 3 7) 0)
((Tower 3 1 7) 3)
((Tower 3 2 7) 2)
((Tower 3 3 7) 1))
```

4 disks

Code:

```
(declare-fun Tower(Int Int Int)Int)
(declare-fun disks(Int Int) Int)
;;Start State
(assert
   (and
      (=
         (disks 1 0)
         4
      )
      (=
         (disks 2 0)
         0
      )
      (=
         (disks 3 0)
         0
      )
      (=
         (Tower 1 1 0)
         4
      )
      (=
         (Tower 1 2 0)
         3
      )
      (=
         (Tower 1 3 0)
         2
      )
      (=
         (Tower 1 4 0)
```

```
        1
      )
      (forall ((t Int) (p Int))
        (implies
          (and
            (<= 2 t 3)
            (<= 1 p 4)
          )
          (=
            (Tower t p 0)
            0
          )
        )
      )
    )
)
;;End State
(assert
  (and
    (=
      (disks 1 15)
      0
    )
    (=
      (disks 2 15)
      0
    )
    (=
      (disks 3 15)
      4
    )
    (=
      (Tower 3 1 15)
      4
    )
    (=
      (Tower 3 2 15)
      3
    )
    (=
```

```
        (Tower 3 3 15)
        2
      )
      (=
        (Tower 3 4 15)
        1
      )
      (forall ((t Int) (p Int))
        (implies
          (and
            (<= 1 t 2)
            (<= 1 p 4)
          )
          (=
            (Tower t p 15)
            0
          )
        )
      )
    )
)
(define-fun move((t1 Int) (t2 Int) (t3 Int) (m Int)) Bool
  (and
    (=
      (disks t1 (+ m 1))
      (-
        (disks t1 m)
        1
      )
    )
    (=
      (disks t2 (+ m 1))
      (+
        (disks t2 m)
        1
      )
    )
    (=
      (disks t3 (+ m 1))
      (disks t3 m)
```

```
      )
      (<
         (Tower t1 (disks t1 m) m)
         (Tower t2 (disks t2 m) m)
      )
      (forall ((p Int))
         (implies
            (<= 1 p 4)
            (and
               (=
                  (Tower t1 p (+ m 1))
                  (ite
                     (= p (disks t1 m))
                     0
                     (Tower t1 p m)
                  )
               )
               (=
                  (Tower t2 p (+ m 1))
                  (ite
                     (= p (disks t2 (+ m 1)))
                     (Tower t1 (disks t1 m) m)
                     (Tower t2 p m)
                  )
               )
               (=
                  (Tower t3 p m)
                  (Tower t3 p (+ m 1))
               )
            )
         )
      )
   )
)
(assert
   (forall ((t Int) (m Int))
      (implies
         (and
            (<= 1 t 3)
            (<= 0 m 14)
```

```
          )
          (<= 0 (disks t m) 4)
        )
    )
)
(assert (forall ((i Int))
    (implies
        (<= 0 i 14)
        (exists ((t1 Int) (t2 Int) (t3 Int))
            (and
                (<= 1 t1 3)
                (<= 1 t2 3)
                (<= 1 t3 3)
                (distinct t1 t2 t3)
                (move t1 t2 t3 i)
            )
        )
    )
))
(check-sat)
(echo "Step 0:")
(get-value(
    (Tower 1 1 0)
    (Tower 1 2 0)
    (Tower 1 3 0)
    (Tower 1 4 0)
    (Tower 2 1 0)
    (Tower 2 2 0)
    (Tower 2 3 0)
    (Tower 2 4 0)
    (Tower 3 1 0)
    (Tower 3 2 0)
    (Tower 3 3 0)
    (Tower 3 4 0)
))
(echo "Step 1:")
(get-value(
    (Tower 1 1 1)
    (Tower 1 2 1)
    (Tower 1 3 1)
```

```
    (Tower 1 4 1)
    (Tower 2 1 1)
    (Tower 2 2 1)
    (Tower 2 3 1)
    (Tower 2 4 1)
    (Tower 3 1 1)
    (Tower 3 2 1)
    (Tower 3 3 1)
    (Tower 3 4 1)
))
(echo "Step 2:")
(get-value(
    (Tower 1 1 2)
    (Tower 1 2 2)
    (Tower 1 3 2)
    (Tower 1 4 2)
    (Tower 2 1 2)
    (Tower 2 2 2)
    (Tower 2 3 2)
    (Tower 2 4 2)
    (Tower 3 1 2)
    (Tower 3 2 2)
    (Tower 3 3 2)
    (Tower 3 4 2)
))
(echo "Step 3:")
(get-value(
    (Tower 1 1 3)
    (Tower 1 2 3)
    (Tower 1 3 3)
    (Tower 1 4 3)
    (Tower 2 1 3)
    (Tower 2 2 3)
    (Tower 2 3 3)
    (Tower 2 4 3)
    (Tower 3 1 3)
    (Tower 3 2 3)
    (Tower 3 3 3)
    (Tower 3 4 3)
))
```

```
(echo "Step 4:")
(get-value(
    (Tower 1 1 4)
    (Tower 1 2 4)
    (Tower 1 3 4)
    (Tower 1 4 4)
    (Tower 2 1 4)
    (Tower 2 2 4)
    (Tower 2 3 4)
    (Tower 2 4 4)
    (Tower 3 1 4)
    (Tower 3 2 4)
    (Tower 3 3 4)
    (Tower 3 4 4)
))
(echo "Step 5:")
(get-value(
    (Tower 1 1 5)
    (Tower 1 2 5)
    (Tower 1 3 5)
    (Tower 1 4 5)
    (Tower 2 1 5)
    (Tower 2 2 5)
    (Tower 2 3 5)
    (Tower 2 4 5)
    (Tower 3 1 5)
    (Tower 3 2 5)
    (Tower 3 3 5)
    (Tower 3 4 5)
))
(echo "Step 6:")
(get-value(
    (Tower 1 1 6)
    (Tower 1 2 6)
    (Tower 1 3 6)
    (Tower 1 4 6)
    (Tower 2 1 6)
    (Tower 2 2 6)
    (Tower 2 3 6)
    (Tower 2 4 6)
```

```
    (Tower 3 1 6)
    (Tower 3 2 6)
    (Tower 3 3 6)
    (Tower 3 4 6)
))
(echo "Step 7:")
(get-value(
    (Tower 1 1 7)
    (Tower 1 2 7)
    (Tower 1 3 7)
    (Tower 1 4 7)
    (Tower 2 1 7)
    (Tower 2 2 7)
    (Tower 2 3 7)
    (Tower 2 4 7)
    (Tower 3 1 7)
    (Tower 3 2 7)
    (Tower 3 3 7)
    (Tower 3 4 7)
))
(echo "Step 8:")
(get-value(
    (Tower 1 1 8)
    (Tower 1 2 8)
    (Tower 1 3 8)
    (Tower 1 4 8)
    (Tower 2 1 8)
    (Tower 2 2 8)
    (Tower 2 3 8)
    (Tower 2 4 8)
    (Tower 3 1 8)
    (Tower 3 2 8)
    (Tower 3 3 8)
    (Tower 3 4 8)
))
(echo "Step 9:")
(get-value(
    (Tower 1 1 9)
    (Tower 1 2 9)
    (Tower 1 3 9)
```

```
    (Tower 1 4 9)
    (Tower 2 1 9)
    (Tower 2 2 9)
    (Tower 2 3 9)
    (Tower 2 4 9)
    (Tower 3 1 9)
    (Tower 3 2 9)
    (Tower 3 3 9)
    (Tower 3 4 9)
))
(echo "Step 10:")
(get-value(
    (Tower 1 1 10)
    (Tower 1 2 10)
    (Tower 1 3 10)
    (Tower 1 4 10)
    (Tower 2 1 10)
    (Tower 2 2 10)
    (Tower 2 3 10)
    (Tower 2 4 10)
    (Tower 3 1 10)
    (Tower 3 2 10)
    (Tower 3 3 10)
    (Tower 3 4 10)
))
(echo "Step 11:")
(get-value(
    (Tower 1 1 11)
    (Tower 1 2 11)
    (Tower 1 3 11)
    (Tower 1 4 11)
    (Tower 2 1 11)
    (Tower 2 2 11)
    (Tower 2 3 11)
    (Tower 2 4 11)
    (Tower 3 1 11)
    (Tower 3 2 11)
    (Tower 3 3 11)
    (Tower 3 4 11)
))
```

```
(echo "Step 12:")
(get-value(
    (Tower 1 1 12)
    (Tower 1 2 12)
    (Tower 1 3 12)
    (Tower 1 4 12)
    (Tower 2 1 12)
    (Tower 2 2 12)
    (Tower 2 3 12)
    (Tower 2 4 12)
    (Tower 3 1 12)
    (Tower 3 2 12)
    (Tower 3 3 12)
    (Tower 3 4 12)
))
(echo "Step 13:")
(get-value(
    (Tower 1 1 13)
    (Tower 1 2 13)
    (Tower 1 3 13)
    (Tower 1 4 13)
    (Tower 2 1 13)
    (Tower 2 2 13)
    (Tower 2 3 13)
    (Tower 2 4 13)
    (Tower 3 1 13)
    (Tower 3 2 13)
    (Tower 3 3 13)
    (Tower 3 4 13)
))
(echo "Step 14:")
(get-value(
    (Tower 1 1 14)
    (Tower 1 2 14)
    (Tower 1 3 14)
    (Tower 1 4 14)
    (Tower 2 1 14)
    (Tower 2 2 14)
    (Tower 2 3 14)
    (Tower 2 4 14)
```

```
   (Tower 3 1 14)
   (Tower 3 2 14)
   (Tower 3 3 14)
   (Tower 3 4 14)
))
(echo "Step 15:")
(get-value(
   (Tower 1 1 15)
   (Tower 1 2 15)
   (Tower 1 3 15)
   (Tower 1 4 15)
   (Tower 2 1 15)
   (Tower 2 2 15)
   (Tower 2 3 15)
   (Tower 2 4 15)
   (Tower 3 1 15)
   (Tower 3 2 15)
   (Tower 3 3 15)
   (Tower 3 4 15)
))
```

Output:

```
sat
Step 0:
(((Tower 1 1 0) 4)
 ((Tower 1 2 0) 3)
 ((Tower 1 3 0) 2)
 ((Tower 1 4 0) 1)
 ((Tower 2 1 0) 0)
 ((Tower 2 2 0) 0)
 ((Tower 2 3 0) 0)
 ((Tower 2 4 0) 0)
 ((Tower 3 1 0) 0)
 ((Tower 3 2 0) 0)
 ((Tower 3 3 0) 0)
 ((Tower 3 4 0) 0))
Step 1:
(((Tower 1 1 1) 4)
```

((Tower 1 2 1) 3)
((Tower 1 3 1) 2)
((Tower 1 4 1) 0)
((Tower 2 1 1) 1)
((Tower 2 2 1) 0)
((Tower 2 3 1) 0)
((Tower 2 4 1) 0)
((Tower 3 1 1) 0)
((Tower 3 2 1) 0)
((Tower 3 3 1) 0)
((Tower 3 4 1) 0))
Step 2:
(((Tower 1 1 2) 4)
((Tower 1 2 2) 3)
((Tower 1 3 2) 0)
((Tower 1 4 2) 0)
((Tower 2 1 2) 1)
((Tower 2 2 2) 0)
((Tower 2 3 2) 0)
((Tower 2 4 2) 0)
((Tower 3 1 2) 2)
((Tower 3 2 2) 0)
((Tower 3 3 2) 0)
((Tower 3 4 2) 0))
Step 3:
(((Tower 1 1 3) 4)
((Tower 1 2 3) 3)
((Tower 1 3 3) 0)
((Tower 1 4 3) 0)
((Tower 2 1 3) 0)
((Tower 2 2 3) 0)
((Tower 2 3 3) 0)
((Tower 2 4 3) 0)
((Tower 3 1 3) 2)
((Tower 3 2 3) 1)

((Tower 3 3 3) 0)
((Tower 3 4 3) 0))
Step 4:
(((Tower 1 1 4) 4)
((Tower 1 2 4) 0)
((Tower 1 3 4) 0)
((Tower 1 4 4) 0)
((Tower 2 1 4) 3)
((Tower 2 2 4) 0)
((Tower 2 3 4) 0)
((Tower 2 4 4) 0)
((Tower 3 1 4) 2)
((Tower 3 2 4) 1)
((Tower 3 3 4) 0)
((Tower 3 4 4) 0))
Step 5:
(((Tower 1 1 5) 4)
((Tower 1 2 5) 1)
((Tower 1 3 5) 0)
((Tower 1 4 5) 0)
((Tower 2 1 5) 3)
((Tower 2 2 5) 0)
((Tower 2 3 5) 0)
((Tower 2 4 5) 0)
((Tower 3 1 5) 2)
((Tower 3 2 5) 0)
((Tower 3 3 5) 0)
((Tower 3 4 5) 0))
Step 6:
(((Tower 1 1 6) 4)
((Tower 1 2 6) 1)
((Tower 1 3 6) 0)
((Tower 1 4 6) 0)
((Tower 2 1 6) 3)
((Tower 2 2 6) 2)

((Tower 2 3 6) 0)
((Tower 2 4 6) 0)
((Tower 3 1 6) 0)
((Tower 3 2 6) 0)
((Tower 3 3 6) 0)
((Tower 3 4 6) 0))
Step 7:
(((Tower 1 1 7) 4)
((Tower 1 2 7) 0)
((Tower 1 3 7) 0)
((Tower 1 4 7) 0)
((Tower 2 1 7) 3)
((Tower 2 2 7) 2)
((Tower 2 3 7) 1)
((Tower 2 4 7) 0)
((Tower 3 1 7) 0)
((Tower 3 2 7) 0)
((Tower 3 3 7) 0)
((Tower 3 4 7) 0))
Step 8:
(((Tower 1 1 8) 0)
((Tower 1 2 8) 0)
((Tower 1 3 8) 0)
((Tower 1 4 8) 0)
((Tower 2 1 8) 3)
((Tower 2 2 8) 2)
((Tower 2 3 8) 1)
((Tower 2 4 8) 0)
((Tower 3 1 8) 4)
((Tower 3 2 8) 0)
((Tower 3 3 8) 0)
((Tower 3 4 8) 0))
Step 9:
(((Tower 1 1 9) 0)
((Tower 1 2 9) 0)

((Tower 1 3 9) 0)
((Tower 1 4 9) 0)
((Tower 2 1 9) 3)
((Tower 2 2 9) 2)
((Tower 2 3 9) 0)
((Tower 2 4 9) 0)
((Tower 3 1 9) 4)
((Tower 3 2 9) 1)
((Tower 3 3 9) 0)
((Tower 3 4 9) 0))
Step 10:
(((Tower 1 1 10) 2)
((Tower 1 2 10) 0)
((Tower 1 3 10) 0)
((Tower 1 4 10) 0)
((Tower 2 1 10) 3)
((Tower 2 2 10) 0)
((Tower 2 3 10) 0)
((Tower 2 4 10) 0)
((Tower 3 1 10) 4)
((Tower 3 2 10) 1)
((Tower 3 3 10) 0)
((Tower 3 4 10) 0))
Step 11:
(((Tower 1 1 11) 2)
((Tower 1 2 11) 1)
((Tower 1 3 11) 0)
((Tower 1 4 11) 0)
((Tower 2 1 11) 3)
((Tower 2 2 11) 0)
((Tower 2 3 11) 0)
((Tower 2 4 11) 0)
((Tower 3 1 11) 4)
((Tower 3 2 11) 0)
((Tower 3 3 11) 0)

((Tower 3 4 11) 0))
Step 12:
(((Tower 1 1 12) 2)
((Tower 1 2 12) 1)
((Tower 1 3 12) 0)
((Tower 1 4 12) 0)
((Tower 2 1 12) 0)
((Tower 2 2 12) 0)
((Tower 2 3 12) 0)
((Tower 2 4 12) 0)
((Tower 3 1 12) 4)
((Tower 3 2 12) 3)
((Tower 3 3 12) 0)
((Tower 3 4 12) 0))
Step 13:
(((Tower 1 1 13) 2)
((Tower 1 2 13) 0)
((Tower 1 3 13) 0)
((Tower 1 4 13) 0)
((Tower 2 1 13) 1)
((Tower 2 2 13) 0)
((Tower 2 3 13) 0)
((Tower 2 4 13) 0)
((Tower 3 1 13) 4)
((Tower 3 2 13) 3)
((Tower 3 3 13) 0)
((Tower 3 4 13) 0))
Step 14:
(((Tower 1 1 14) 0)
((Tower 1 2 14) 0)
((Tower 1 3 14) 0)
((Tower 1 4 14) 0)
((Tower 2 1 14) 1)
((Tower 2 2 14) 0)
((Tower 2 3 14) 0)

((Tower 2 4 14) 0)
((Tower 3 1 14) 4)
((Tower 3 2 14) 3)
((Tower 3 3 14) 2)
((Tower 3 4 14) 0))
Step 15:
(((Tower 1 1 15) 0)
((Tower 1 2 15) 0)
((Tower 1 3 15) 0)
((Tower 1 4 15) 0)
((Tower 2 1 15) 0)
((Tower 2 2 15) 0)
((Tower 2 3 15) 0)
((Tower 2 4 15) 0)
((Tower 3 1 15) 4)
((Tower 3 2 15) 3)
((Tower 3 3 15) 2)
((Tower 3 4 15) 1))