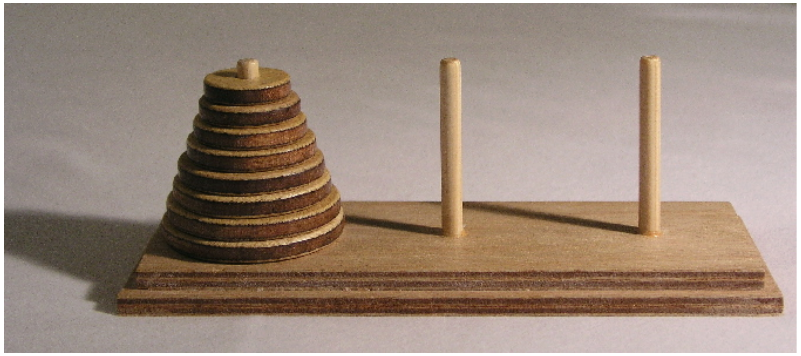
Tower of Hanoi

Victor Milenkovic

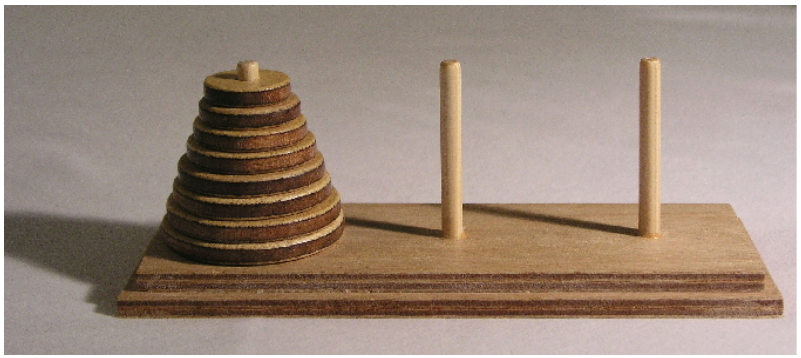
Department of Computer Science
University of Miami

CSC220 Programming II – Spring 2022

Tower of Hanoi

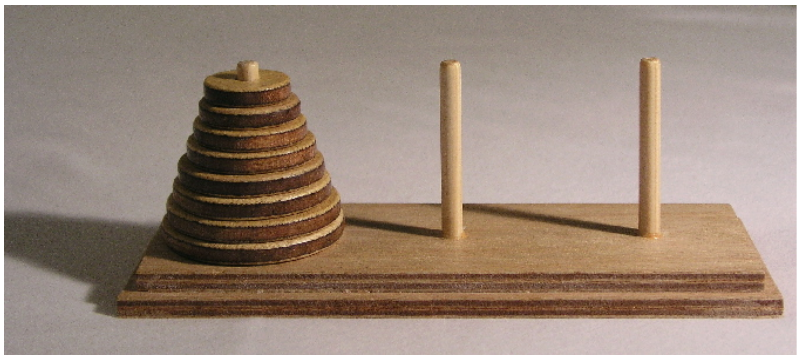


Tower of Hanoi



The Tower of Hanoi puzzle is all about stacks.

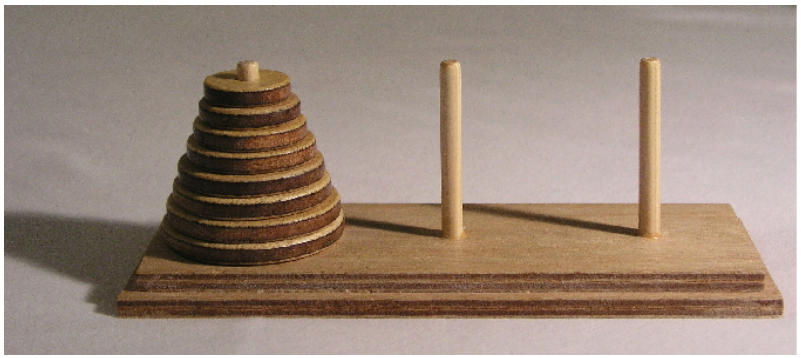
Tower of Hanoi



The Tower of Hanoi puzzle is all about stacks.

- When the monks have solved a puzzle with 64 disks,

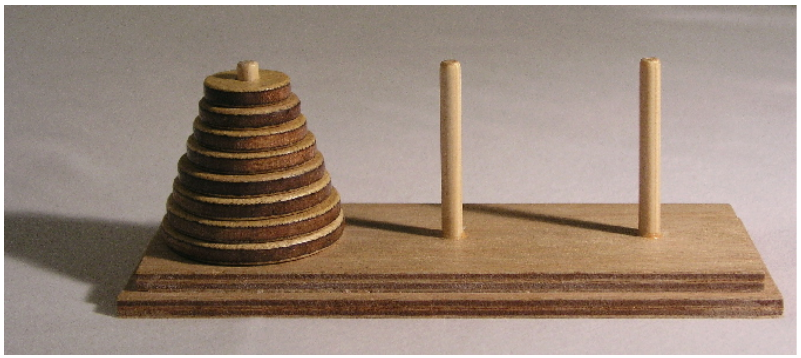
Tower of Hanoi



The Tower of Hanoi puzzle is all about stacks.

- When the monks have solved a puzzle with 64 disks,
- the world will end.

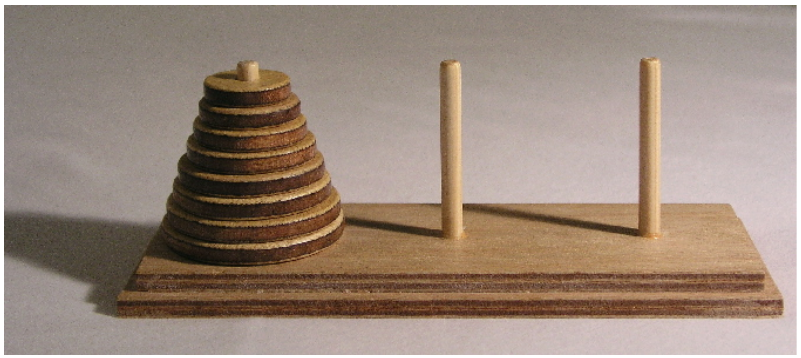
Tower of Hanoi



The Tower of Hanoi puzzle is all about stacks.

- When the monks have solved a puzzle with 64 disks,
- the world will end.
- Fortunately, it requires about $9 \cdot 10^{18}$ moves.

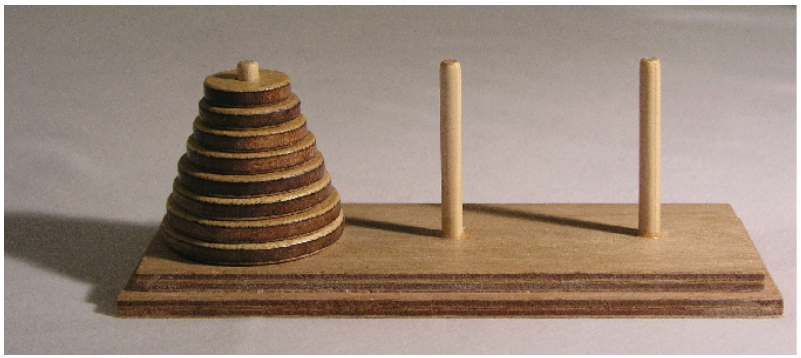
Tower of Hanoi



The Tower of Hanoi puzzle is all about stacks.

- When the monks have solved a puzzle with 64 disks,
- the world will end.
- Fortunately, it requires about $9 \cdot 10^{18}$ moves.
- Contrast the Arthur C. Clark's story The Nine Billion Names of God.

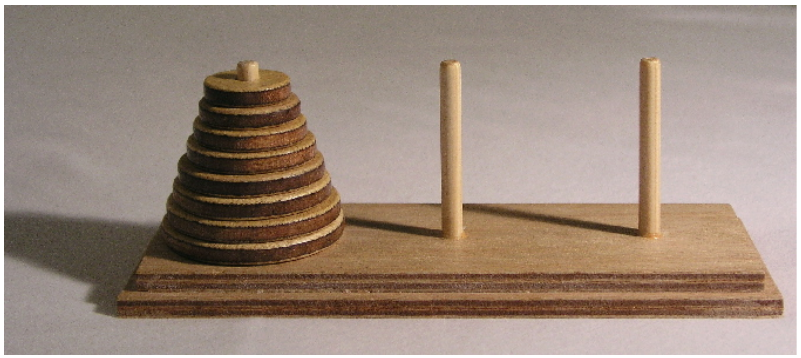
Tower of Hanoi



The Tower of Hanoi puzzle is all about stacks.

- When the monks have solved a puzzle with 64 disks,
- the world will end.
- Fortunately, it requires about $9 \cdot 10^{18}$ moves.
- Contrast the Arthur C. Clark's story The Nine Billion Names of God.
- Even a 1950s computer can complete the required task in weeks.

Tower of Hanoi



The Tower of Hanoi puzzle is all about stacks.

- When the monks have solved a puzzle with 64 disks,
- the world will end.
- Fortunately, it requires about $9 \cdot 10^{18}$ moves.
- Contrast the Arthur C. Clark's story The Nine Billion Names of God.
- Even a 1950s computer can complete the required task in weeks.

Here is a good place to practice.

Implementing ToH with StackInterface

We can use three StackInterface<Integer> to store the current situation:

Implementing ToH with StackInterface

We can use three StackInterface<Integer> to store the current situation:

- a: 4 3 2 1
- b:
- c:

Implementing ToH with StackInterface

We can use three StackInterface<Integer> to store the current situation:

- a: 4 3 2 1
- b:
- c:

Here, the top of the stack is on the right. (Turn your head sideways!)

Implementing ToH with StackInterface

We can use three StackInterface<Integer> to store the current situation:

- a: 4 3 2 1
- b:
- c:

Here, the top of the stack is on the right. (Turn your head sideways!)
A move is a push and a pop. In fact, you can do it all at once like this

Implementing ToH with StackInterface

We can use three StackInterface<Integer> to store the current situation:

- a: 4 3 2 1
- b:
- c:

Here, the top of the stack is on the right. (Turn your head sideways!)
A move is a push and a pop. In fact, you can do it all at once like this

- b.push(a.pop());

Implementing ToH with StackInterface

We can use three StackInterface<Integer> to store the current situation:

- a: 4 3 2 1
- b:
- c:

Here, the top of the stack is on the right. (Turn your head sideways!)
A move is a push and a pop. In fact, you can do it all at once like this

- b.push(a.pop());

with the result

Implementing ToH with StackInterface

We can use three StackInterface<Integer> to store the current situation:

- a: 4 3 2 1
- b:
- c:

Here, the top of the stack is on the right. (Turn your head sideways!)
A move is a push and a pop. In fact, you can do it all at once like this

- b.push(a.pop());

with the result

- a: 4 3 2
- b: 1
- c:

Displaying the Game

Displaying the Game

Question: how can we print out the contents of one of the stacks as it appears here, such as

Displaying the Game

Question: how can we print out the contents of one of the stacks as it appears here, such as

- 4 3 2

for stack a?

Displaying the Game

Question: how can we print out the contents of one of the stacks as it appears here, such as

- 4 3 2

for stack a?

- We can only peek at the top element 2.

Displaying the Game

Question: how can we print out the contents of one of the stacks as it appears here, such as

- 4 3 2

for stack a?

- We can only peek at the top element 2.
- We can't see the 4 and the 3.

Displaying the Game

Question: how can we print out the contents of one of the stacks as it appears here, such as

- 4 3 2

for stack a?

- We can only peek at the top element 2.
- We can't see the 4 and the 3.
- Hint: use a helper stack.

Displaying the Game

Question: how can we print out the contents of one of the stacks as it appears here, such as

- 4 3 2

for stack a?

- We can only peek at the top element 2.
- We can't see the 4 and the 3.
- Hint: use a helper stack.

To print out stack a:

Displaying the Game

Question: how can we print out the contents of one of the stacks as it appears here, such as

- 4 3 2

for stack a?

- We can only peek at the top element 2.
- We can't see the 4 and the 3.
- Hint: use a helper stack.

To print out stack a:

- while stack a is not empty

Displaying the Game

Question: how can we print out the contents of one of the stacks as it appears here, such as

- 4 3 2

for stack a?

- We can only peek at the top element 2.
- We can't see the 4 and the 3.
- Hint: use a helper stack.

To print out stack a:

- while stack a is not empty
  - pop from a and push the result on helper

Displaying the Game

Question: how can we print out the contents of one of the stacks as it appears here, such as

- 4 3 2

for stack a?

- We can only peek at the top element 2.
- We can't see the 4 and the 3.
- Hint: use a helper stack.

To print out stack a:

- while stack a is not empty
  - pop from a and push the result on helper
- while stack helper is not empty

Displaying the Game

Question: how can we print out the contents of one of the stacks as it appears here, such as

- 4 3 2

for stack a?

- We can only peek at the top element 2.
- We can't see the 4 and the 3.
- Hint: use a helper stack.

To print out stack a:

- while stack a is not empty
  - pop from a and push the result on helper
- while stack helper is not empty
  - pop from helper, print it, push on a

Top-Down Planning

Top-Down Planning

So we can program a simple Tower of Hanoi game for people to play.

Top-Down Planning

So we can program a simple Tower of Hanoi game for people to play.

But can we program a Tower of Hanoi solver?

Top-Down Planning

So we can program a simple Tower of Hanoi game for people to play.

But can we program a Tower of Hanoi solver?

A stack is useful for TOP DOWN design or planning.

Top-Down Planning

So we can program a simple Tower of Hanoi game for people to play.

But can we program a Tower of Hanoi solver?

A stack is useful for TOP DOWN design or planning.

In this case, you need a stack of moves (StackInterface<Move>).

Top-Down Planning

So we can program a simple Tower of Hanoi game for people to play.

But can we program a Tower of Hanoi solver?

A stack is useful for TOP DOWN design or planning.

In this case, you need a stack of moves (StackInterface<Move>).

- Push a multiple-disk move on the stack.

Top-Down Planning

So we can program a simple Tower of Hanoi game for people to play.

But can we program a Tower of Hanoi solver?

A stack is useful for TOP DOWN design or planning.

In this case, you need a stack of moves (StackInterface<Move>).

- Push a multiple-disk move on the stack.
- While the stack is not empty:

Top-Down Planning

So we can program a simple Tower of Hanoi game for people to play.

But can we program a Tower of Hanoi solver?

A stack is useful for TOP DOWN design or planning.

In this case, you need a stack of moves (StackInterface<Move>).

- Push a multiple-disk move on the stack.
- While the stack is not empty:
  - Pop a move.

Top-Down Planning

So we can program a simple Tower of Hanoi game for people to play.

But can we program a Tower of Hanoi solver?

A stack is useful for TOP DOWN design or planning.

In this case, you need a stack of moves (StackInterface<Move>).

- Push a multiple-disk move on the stack.
- While the stack is not empty:
  - Pop a move.
  - If it is an easy, single-disk move,

Top-Down Planning

So we can program a simple Tower of Hanoi game for people to play.

But can we program a Tower of Hanoi solver?

A stack is useful for TOP DOWN design or planning.

In this case, you need a stack of moves (StackInterface<Move>).

- Push a multiple-disk move on the stack.
- While the stack is not empty:
  - Pop a move.
  - If it is an easy, single-disk move,
    - do it;

Top-Down Planning

So we can program a simple Tower of Hanoi game for people to play.

But can we program a Tower of Hanoi solver?

A stack is useful for TOP DOWN design or planning.

In this case, you need a stack of moves (StackInterface<Move>).

- Push a multiple-disk move on the stack.
- While the stack is not empty:
  - Pop a move.
  - If it is an easy, single-disk move,
    - do it;
  - else,

Top-Down Planning

So we can program a simple Tower of Hanoi game for people to play.

But can we program a Tower of Hanoi solver?

A stack is useful for TOP DOWN design or planning.

In this case, you need a stack of moves (StackInterface<Move>).

- Push a multiple-disk move on the stack.
- While the stack is not empty:
  - Pop a move.
  - If it is an easy, single-disk move,
    - do it;
  - else,
    - break it into submoves

Top-Down Planning

So we can program a simple Tower of Hanoi game for people to play.

But can we program a Tower of Hanoi solver?

A stack is useful for TOP DOWN design or planning.

In this case, you need a stack of moves (StackInterface<Move>).

- Push a multiple-disk move on the stack.
- While the stack is not empty:
  - Pop a move.
  - If it is an easy, single-disk move,
    - do it;
  - else,
    - break it into submoves
    - and push the submoves onto the stack.

Solving Tower of Hanoi

Solving Tower of Hanoi

- Let's apply top down planning to the Tower of Hanoi.

Solving Tower of Hanoi

- Let's apply top down planning to the Tower of Hanoi.
  - The big move is 4ac.

Solving Tower of Hanoi

- Let's apply top down planning to the Tower of Hanoi.
  - The big move is 4ac.
  - That means "move the top four disks from peg a to peg c".

Solving Tower of Hanoi

- Let's apply top down planning to the Tower of Hanoi.
  - The big move is 4ac.
  - That means "move the top four disks from peg a to peg c".
- How can 4ac be broken into submoves?

Solving Tower of Hanoi

- Let's apply top down planning to the Tower of Hanoi.
  - The big move is 4ac.
  - That means "move the top four disks from peg a to peg c".
- How can 4ac be broken into submoves?
  - How about 3ab, 1ac, 3bc?

Solving Tower of Hanoi

- Let's apply top down planning to the Tower of Hanoi.
  - The big move is 4ac.
  - That means "move the top four disks from peg a to peg c".
- How can 4ac be broken into submoves?
  - How about 3ab, 1ac, 3bc?
- Does that make sense?

Solving Tower of Hanoi

- Let's apply top down planning to the Tower of Hanoi.
  - The big move is 4ac.
  - That means "move the top four disks from peg a to peg c".
- How can 4ac be broken into submoves?
  - How about 3ab, 1ac, 3bc?
- Does that make sense?
  - 3ab: move the top three disks (3, 2, 1) from peg a to peg b.

Solving Tower of Hanoi

- Let's apply top down planning to the Tower of Hanoi.
  - The big move is 4ac.
  - That means "move the top four disks from peg a to peg c".
- How can 4ac be broken into submoves?
  - How about 3ab, 1ac, 3bc?
- Does that make sense?
  - 3ab: move the top three disks (3, 2, 1) from peg a to peg b.
  - 1ac: move the top one disk (4) from peg a to peg c.

Solving Tower of Hanoi

- Let's apply top down planning to the Tower of Hanoi.
  - The big move is 4ac.
  - That means "move the top four disks from peg a to peg c".
- How can 4ac be broken into submoves?
  - How about 3ab, 1ac, 3bc?
- Does that make sense?
  - 3ab: move the top three disks (3, 2, 1) from peg a to peg b.
  - 1ac: move the top one disk (4) from peg a to peg c.
  - 3bc: move the top three disks (3, 2, 1) from peg b to peg c.

Solving Tower of Hanoi

- Let's apply top down planning to the Tower of Hanoi.
  - The big move is 4ac.
  - That means "move the top four disks from peg a to peg c".
- How can 4ac be broken into submoves?
  - How about 3ab, 1ac, 3bc?
- Does that make sense?
  - 3ab: move the top three disks (3, 2, 1) from peg a to peg b.
  - 1ac: move the top one disk (4) from peg a to peg c.
  - 3bc: move the top three disks (3, 2, 1) from peg b to peg c.
- Any move with one disk, such as 1ab, is easy, and we can do it right away.

Solving Tower of Hanoi

- Let's apply top down planning to the Tower of Hanoi.
  - The big move is 4ac.
  - That means "move the top four disks from peg a to peg c".
- How can 4ac be broken into submoves?
  - How about 3ab, 1ac, 3bc?
- Does that make sense?
  - 3ab: move the top three disks (3, 2, 1) from peg a to peg b.
  - 1ac: move the top one disk (4) from peg a to peg c.
  - 3bc: move the top three disks (3, 2, 1) from peg b to peg c.
- Any move with one disk, such as 1ab, is easy, and we can do it right away.
- All others can be broken into three submoves.

4-disk Solution

So let's solve the 4-disk Tower of Hanoi.

4-disk Solution

So let's solve the 4-disk Tower of Hanoi.

- Here is out initial move stack:

4-disk Solution

So let's solve the 4-disk Tower of Hanoi.

- Here is out initial move stack:
  - 4ac

4-disk Solution

So let's solve the 4-disk Tower of Hanoi.

- Here is out initial move stack:
  - 4ac
- Pop the 4ac.

4-disk Solution

So let's solve the 4-disk Tower of Hanoi.

- Here is out initial move stack:
  - 4ac
- Pop the 4ac.
- It's hard, so break it down into 3ab, 1ac, 3bc.

4-disk Solution

So let's solve the 4-disk Tower of Hanoi.

- Here is out initial move stack:
  - 4ac
- Pop the 4ac.
- It's hard, so break it down into 3ab, 1ac, 3bc.
- Push these on the stack,

4-disk Solution

So let's solve the 4-disk Tower of Hanoi.

- Here is out initial move stack:
  - 4ac
- Pop the 4ac.
- It's hard, so break it down into 3ab, 1ac, 3bc.
- Push these on the stack,
- but in reverse order because we want to to 3ab first!

4-disk Solution

So let's solve the 4-disk Tower of Hanoi.

- Here is out initial move stack:
  - 4ac
- Pop the 4ac.
- It's hard, so break it down into 3ab, 1ac, 3bc.
- Push these on the stack,
- but in reverse order because we want to to 3ab first!
  - 3ab <– top of stack
    1ac
    3bc

4-disk Solution

So let's solve the 4-disk Tower of Hanoi.

- Here is out initial move stack:
  - 4ac
- Pop the 4ac.
- It's hard, so break it down into 3ab, 1ac, 3bc.
- Push these on the stack,
- but in reverse order because we want to to 3ab first!
  - 3ab <– top of stack
    1ac
    3bc
- Pop the 3ab.

4-disk Solution

So let's solve the 4-disk Tower of Hanoi.

- Here is out initial move stack:
  - 4ac
- Pop the 4ac.
- It's hard, so break it down into 3ab, 1ac, 3bc.
- Push these on the stack,
- but in reverse order because we want to to 3ab first!
  - 3ab <– top of stack
    1ac
    3bc
- Pop the 3ab.
- It's hard, so break it down into 2ac, 1ab, 2cb.

4-disk Solution

So let's solve the 4-disk Tower of Hanoi.

- Here is out initial move stack:
  - 4ac
- Pop the 4ac.
- It's hard, so break it down into 3ab, 1ac, 3bc.
- Push these on the stack,
- but in reverse order because we want to to 3ab first!
  - 3ab <– top of stack
    1ac
    3bc
- Pop the 3ab.
- It's hard, so break it down into 2ac, 1ab, 2cb.
- Push these:

4-disk Solution

So let's solve the 4-disk Tower of Hanoi.

- Here is out initial move stack:
  - 4ac
- Pop the 4ac.
- It's hard, so break it down into 3ab, 1ac, 3bc.
- Push these on the stack,
- but in reverse order because we want to to 3ab first!
  - 3ab <– top of stack
    1ac
    3bc
- Pop the 3ab.
- It's hard, so break it down into 2ac, 1ab, 2cb.
- Push these:
  - 2ac <– top of stack
    1ab
    2cb
    1ac
    3bc

4-disk Solution

So let's solve the 4-disk Tower of Hanoi.

- Here is out initial move stack:
  - 4ac
- Pop the 4ac.
- It's hard, so break it down into 3ab, 1ac, 3bc.
- Push these on the stack,
- but in reverse order because we want to to 3ab first!
  - 3ab <– top of stack
    1ac
    3bc
- Pop the 3ab.
- It's hard, so break it down into 2ac, 1ab, 2cb.
- Push these:
  - 2ac <– top of stack
    1ab
    2cb
    1ac
    3bc
- Pop the 2ac.

4-disk Solution

So let's solve the 4-disk Tower of Hanoi.

- Here is out initial move stack:
  - 4ac
- Pop the 4ac.
- It's hard, so break it down into 3ab, 1ac, 3bc.
- Push these on the stack,
- but in reverse order because we want to to 3ab first!
  - 3ab <– top of stack
    1ac
    3bc
- Pop the 3ab.
- It's hard, so break it down into 2ac, 1ab, 2cb.
- Push these:
  - 2ac <– top of stack
    1ab
    2cb
    1ac
    3bc
- Pop the 2ac.
- It's hard, so break it down into 1ab, 1ac, 1bc.

4-disk Solution

- Push these:

4-disk Solution

- Push these:
  - 1ab
    1ac
    1bc
    1ab
    2cb
    1ac
    3bc

4-disk Solution

- Push these:
  - 1ab
    1ac
    1bc
    1ab
    2cb
    1ac
    3bc
- Now I can do four pops in a row that give me easy moves that I can do!

4-disk Solution

- Push these:
  - 1ab
    1ac
    1bc
    1ab
    2cb
    1ac
    3bc
- Now I can do four pops in a row that give me easy moves that I can do!
  - a: 4 3 2 1
  - b:
  - c:

4-disk Solution

- Push these:
  - 1ab
    1ac
    1bc
    1ab
    2cb
    1ac
    3bc
- Now I can do four pops in a row that give me easy moves that I can do!
  - a: 4 3 2 1
  - b:
  - c:
- goes to

4-disk Solution

- Push these:
  - 1ab
    1ac
    1bc
    1ab
    2cb
    1ac
    3bc
- Now I can do four pops in a row that give me easy moves that I can do!
  - a: 4 3 2 1
  - b:
  - c:
- goes to
  - a: 4
  - b: 3
  - c: 2 1

4-disk Solution

- Push these:
  - 1ab
    1ac
    1bc
    1ab
    2cb
    1ac
    3bc
- Now I can do four pops in a row that give me easy moves that I can do!
  - a: 4 3 2 1
  - b:
  - c:
- goes to
  - a: 4
  - b: 3
  - c: 2 1
- and the move stack is

4-disk Solution

- Push these:
  - 1ab
    1ac
    1bc
    1ab
    2cb
    1ac
    3bc
- Now I can do four pops in a row that give me easy moves that I can do!
  - a: 4 3 2 1
  - b:
  - c:
- goes to
  - a: 4
  - b: 3
  - c: 2 1
- and the move stack is
  - 2cb
    1ac
    3bc

4-disk Solution

- Push these:
  - 1ab
    1ac
    1bc
    1ab
    2cb
    1ac
    3bc
- Now I can do four pops in a row that give me easy moves that I can do!
  - a: 4 3 2 1
  - b:
  - c:
- goes to
  - a: 4
  - b: 3
  - c: 2 1
- and the move stack is
  - 2cb
    1ac
    3bc
- Pop 2cb.

4-disk Solution

- Push these:
  - 1ab
    1ac
    1bc
    1ab
    2cb
    1ac
    3bc
- Now I can do four pops in a row that give me easy moves that I can do!
  - a: 4 3 2 1
  - b:
  - c:
- goes to
  - a: 4
  - b: 3
  - c: 2 1
- and the move stack is
  - 2cb
    1ac
    3bc
- Pop 2cb.
- It's hard, so break it into 1ca, 1cb, 1ab:

4-disk Solution

- Push these:
  - 1ab
    1ac
    1bc
    1ab
    2cb
    1ac
    3bc
- Now I can do four pops in a row that give me easy moves that I can do!
  - a: 4 3 2 1
  - b:
  - c:
- goes to
  - a: 4
  - b: 3
  - c: 2 1
- and the move stack is
  - 2cb
    1ac
    3bc
- Pop 2cb.
- It's hard, so break it into 1ca, 1cb, 1ab:
  - 1ca
    1cb
    1ab
    1ac
    3bc

4-disk Solution

4-disk Solution

- Four pops of easy moves later:

4-disk Solution

- Four pops of easy moves later:
- Pegs:

4-disk Solution

- Four pops of easy moves later:
- Pegs:
  - a:

4-disk Solution

- Four pops of easy moves later:
- Pegs:
  - a:
  - b: 3 2 1
  - c: 4
- Move stack:

4-disk Solution

- Four pops of easy moves later:
- Pegs:
  - a:
  - b: 3 2 1
  - c: 4
- Move stack:
  - 3bc

4-disk Solution

- Four pops of easy moves later:
- Pegs:
  - a:
  - b: 3 2 1
  - c: 4
- Move stack:
  - 3bc
- Continue on your own!

Implementation

Implementation

- So how can you implement this idea?

Implementation

- So how can you implement this idea?
- What does the move stack contain?

Implementation

- So how can you implement this idea?
- What does the move stack contain?
- I hope the answer is obvious.

Implementation

- So how can you implement this idea?
- What does the move stack contain?
- I hope the answer is obvious.
- You need a Move class:

```
class Move {
  int fromPeg;   // 0 for a, 1 for b, 2 for c
  int toPeg;     // ditto
}
```

Implementation

- So how can you implement this idea?
- What does the move stack contain?
- I hope the answer is obvious.
- You need a Move class:

```
class Move {
  int fromPeg;   // 0 for a, 1 for b, 2 for c
  int toPeg;     // ditto
}
```

- And a derived class MoveN with an added numberToMove variable.

Implementation

- So how can you implement this idea?
- What does the move stack contain?
- I hope the answer is obvious.
- You need a Move class:

```
class Move {
  int fromPeg;   // 0 for a, 1 for b, 2 for c
  int toPeg;     // ditto
}
```

- And a derived class MoveN with an added numberToMove variable.
- And the stack is StackInterface<Move>.

Implementation

- So how can you implement this idea?
- What does the move stack contain?
- I hope the answer is obvious.
- You need a Move class:

```
class Move {
  int fromPeg;   // 0 for a, 1 for b, 2 for c
  int toPeg;     // ditto
}
```

- And a derived class MoveN with an added numberToMove variable.
- And the stack is StackInterface<Move>.
- Since a MoveN is a Move, it can go into the stack.

Running Time

Running Time

- To move a stack of n:

Running Time

- To move a stack of n:
  - Move a stack of $n - 1$.

Running Time

- To move a stack of n:
  - Move a stack of $n - 1$.
  - Move 1.

Running Time

- To move a stack of n:
  - Move a stack of $n - 1$.
  - Move 1.
  - Move a stack of $n - 1$

Running Time

- To move a stack of n:
  - Move a stack of $n - 1$.
  - Move 1.
  - Move a stack of $n - 1$
- This is similar to recursive Fibonacci.

Running Time

- To move a stack of n:
  - Move a stack of $n - 1$.
  - Move 1.
  - Move a stack of $n - 1$
- This is similar to recursive Fibonacci.
- To calculate fib(n):

Running Time

- To move a stack of n:
  - Move a stack of $n - 1$.
  - Move 1.
  - Move a stack of $n - 1$
- This is similar to recursive Fibonacci.
- To calculate fib(n):
  - Calculate fib(n-2).

Running Time

- To move a stack of n:
  - Move a stack of $n - 1$.
  - Move 1.
  - Move a stack of $n - 1$
- This is similar to recursive Fibonacci.
- To calculate fib(n):
  - Calculate fib(n-2).
  - Calculate fib(n-1).

Running Time

- To move a stack of n:
  - Move a stack of $n - 1$.
  - Move 1.
  - Move a stack of $n - 1$
- This is similar to recursive Fibonacci.
- To calculate fib(n):
  - Calculate fib(n-2).
  - Calculate fib(n-1).
  - Add them.

Running Time

- To move a stack of n:
  - Move a stack of $n - 1$.
  - Move 1.
  - Move a stack of $n - 1$
- This is similar to recursive Fibonacci.
- To calculate fib(n):
  - Calculate fib(n-2).
  - Calculate fib(n-1).
  - Add them.
- Not surprisingly, solving Tower of Hanoi is also exponential time.

Running Time

- To move a stack of n:
  - Move a stack of $n - 1$.
  - Move 1.
  - Move a stack of $n - 1$
- This is similar to recursive Fibonacci.
- To calculate fib(n):
  - Calculate fib(n-2).
  - Calculate fib(n-1).
  - Add them.
- Not surprisingly, solving Tower of Hanoi is also exponential time.
  - Exactly $2^n - 1$ moves.

Running Time

- To move a stack of n:
  - Move a stack of $n - 1$.
  - Move 1.
  - Move a stack of $n - 1$
- This is similar to recursive Fibonacci.
- To calculate fib(n):
  - Calculate fib(n-2).
  - Calculate fib(n-1).
  - Add them.
- Not surprisingly, solving Tower of Hanoi is also exponential time.
  - Exactly $2^n - 1$ moves.
  - Which is $O(2^n)$.

Running Time

- To move a stack of n:
  - Move a stack of $n - 1$.
  - Move 1.
  - Move a stack of $n - 1$
- This is similar to recursive Fibonacci.
- To calculate fib(n):
  - Calculate fib(n-2).
  - Calculate fib(n-1).
  - Add them.
- Not surprisingly, solving Tower of Hanoi is also exponential time.
  - Exactly $2^n - 1$ moves.
  - Which is $O(2^n)$.
- For the monks solving a stack of 64:

Running Time

- To move a stack of n:
  - Move a stack of $n - 1$.
  - Move 1.
  - Move a stack of $n - 1$
- This is similar to recursive Fibonacci.
- To calculate fib(n):
  - Calculate fib(n-2).
  - Calculate fib(n-1).
  - Add them.
- Not surprisingly, solving Tower of Hanoi is also exponential time.
  - Exactly $2^n - 1$ moves.
  - Which is $O(2^n)$.
- For the monks solving a stack of 64:
  - $2^{63} \approx 9 \cdot 10^{18}$ moves.

Running Time

- To move a stack of n:
  - Move a stack of $n-1$.
  - Move 1.
  - Move a stack of $n-1$
- This is similar to recursive Fibonacci.
- To calculate fib(n):
  - Calculate fib(n-2).
  - Calculate fib(n-1).
  - Add them.
- Not surprisingly, solving Tower of Hanoi is also exponential time.
  - Exactly $2^n - 1$ moves.
  - Which is $O(2^n)$.
- For the monks solving a stack of 64:
  - $2^{63} \approx 9 \cdot 10^{18}$ moves.
  - At one move per second.

Running Time

- To move a stack of n:
  - Move a stack of $n-1$.
  - Move 1.
  - Move a stack of $n-1$
- This is similar to recursive Fibonacci.
- To calculate fib(n):
  - Calculate fib(n-2).
  - Calculate fib(n-1).
  - Add them.
- Not surprisingly, solving Tower of Hanoi is also exponential time.
  - Exactly $2^n - 1$ moves.
  - Which is $O(2^n)$.
- For the monks solving a stack of 64:
  - $2^{63} \approx 9 \cdot 10^{18}$ moves.
  - At one move per second.
  - 292 billion years.

Summary

Summary

- Tower of Hanoi has a natural representation using three StackInterface<Integer>.

Summary

- Tower of Hanoi has a natural representation using three StackInterface<Integer>.
- Printing or displaying uses a *helper* stack.

Summary

- Tower of Hanoi has a natural representation using three StackInterface<Integer>.
- Printing or displaying uses a *helper* stack.
- Solving Tower of Hanoi requires *top-down* planning.

Summary

- Tower of Hanoi has a natural representation using three StackInterface<Integer>.
- Printing or displaying uses a *helper* stack.
- Solving Tower of Hanoi requires *top-down* planning.
- Which we can implement using a StackInterface<Move>.

Summary

- Tower of Hanoi has a natural representation using three StackInterface<Integer>.
- Printing or displaying uses a *helper* stack.
- Solving Tower of Hanoi requires *top-down* planning.
- Which we can implement using a StackInterface<Move>.
- This solution uses $O(2^n)$ time.