Thinking Recursively

GTGTC Info Session

- "Are you passionate about computer science, education or both? Do you want to encourage young women interested in learning coding skills?
- Come to the ***Girls Teaching Girls to Code*** Info Session! We're going to be discussing our organization, what we do, and how you can get involved.
- We welcome all levels of experience. Learn about Code Camp, our biggest event, as well as other opportunities throughout the year.
- Hope to see you there!

Women's Community Center (Fire Truck House)
Wednesday, January 25 7-8 PM

Recursive Problem-Solving

```
if (problem is sufficiently simple) {
   Directly solve the problem.
   Return the solution.
 } else {
   Split the problem up into one or more smaller
      problems with the same structure as the original.
   Solve each of those smaller problems.
   Combine the results to get the overall solution.
   Return the overall solution.
}
```

```
int digitalRootOf(int value);
int sumOfDigitsOf(int value);

int sumOfDigitsOf(int value) {
    if (value < 10) {
        return value;
    } else {
        return sumOfDigitsOf(value / 10) + (value % 10);
    }
}

int digitalRootOf(int value) {
    if (value < 10) {
        return value;
    } else {
        return digitalRootOf(sumOfDigitsOf(value));
    }
}
```

```
string reverseOf(const string& text) {
    if (text == "") {
        return "";
    } else {
        return reverseOf(text.substr(1)) + text[0];
    }
}
```

```
int bestCoverageFor(const Vector<int>& populations) {
    if (populations.size() == 0) {
        return 0;
    } else if (populations.size() == 1) {
        return populations[0];
    } else {
        Vector<int> allButFirst    = tailOf(populations);
        Vector<int> allButFirstTwo = tailOf(allButFirst);

        int withFirst = populations[0] +
                        bestCoverageFor(allButFirstTwo);
        int withoutFirst = bestCoverageFor(allButFirst);

        return max(withFirst, withoutFirst);
    }
}
```

What's Going On?

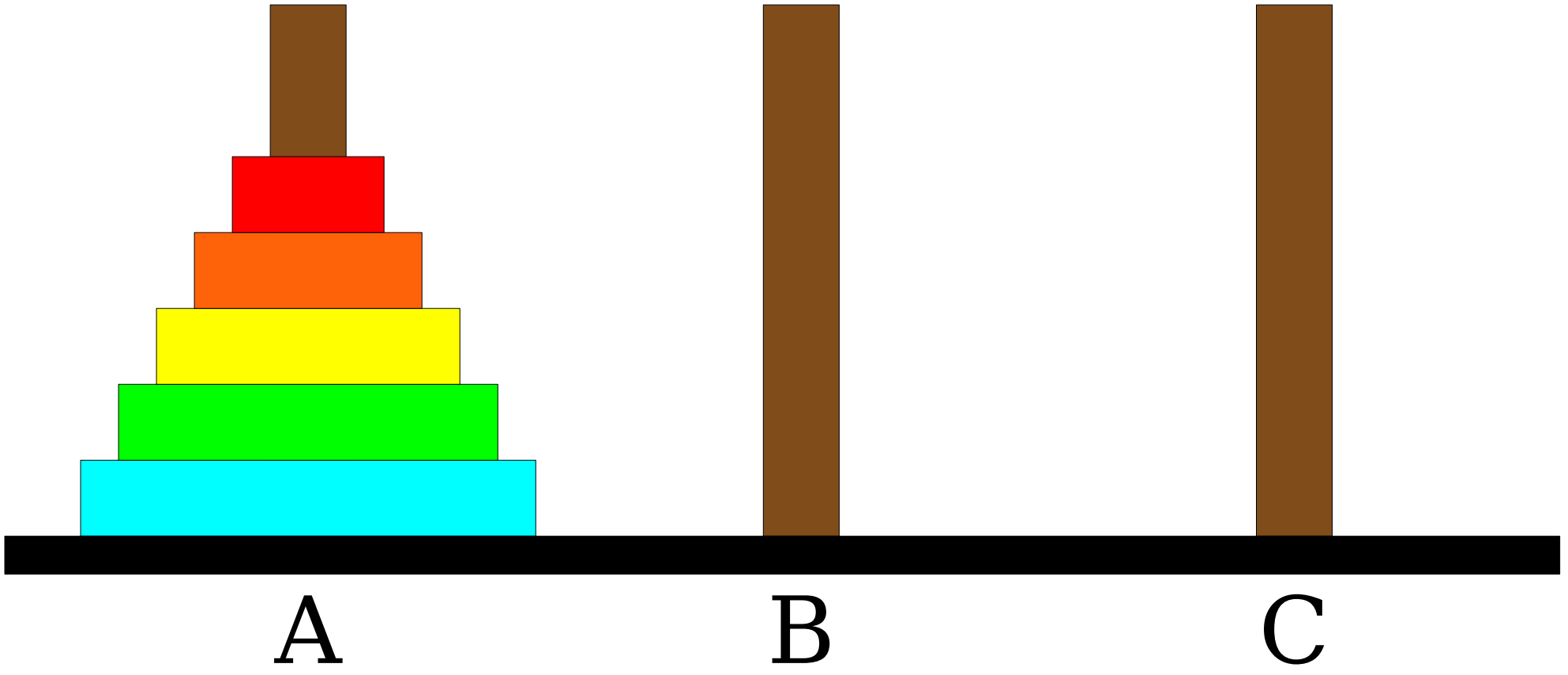
- Recursion solves a problem by continuously simplifying the problem until it becomes simple enough to be solved directly.
- The ***recursive step*** makes the problem slightly simpler.
- The ***base case*** is what ultimately makes the problem solvable – it guarantees that when the problem is sufficiently simple, we can just solve it directly.

Recursive Problem-Solving

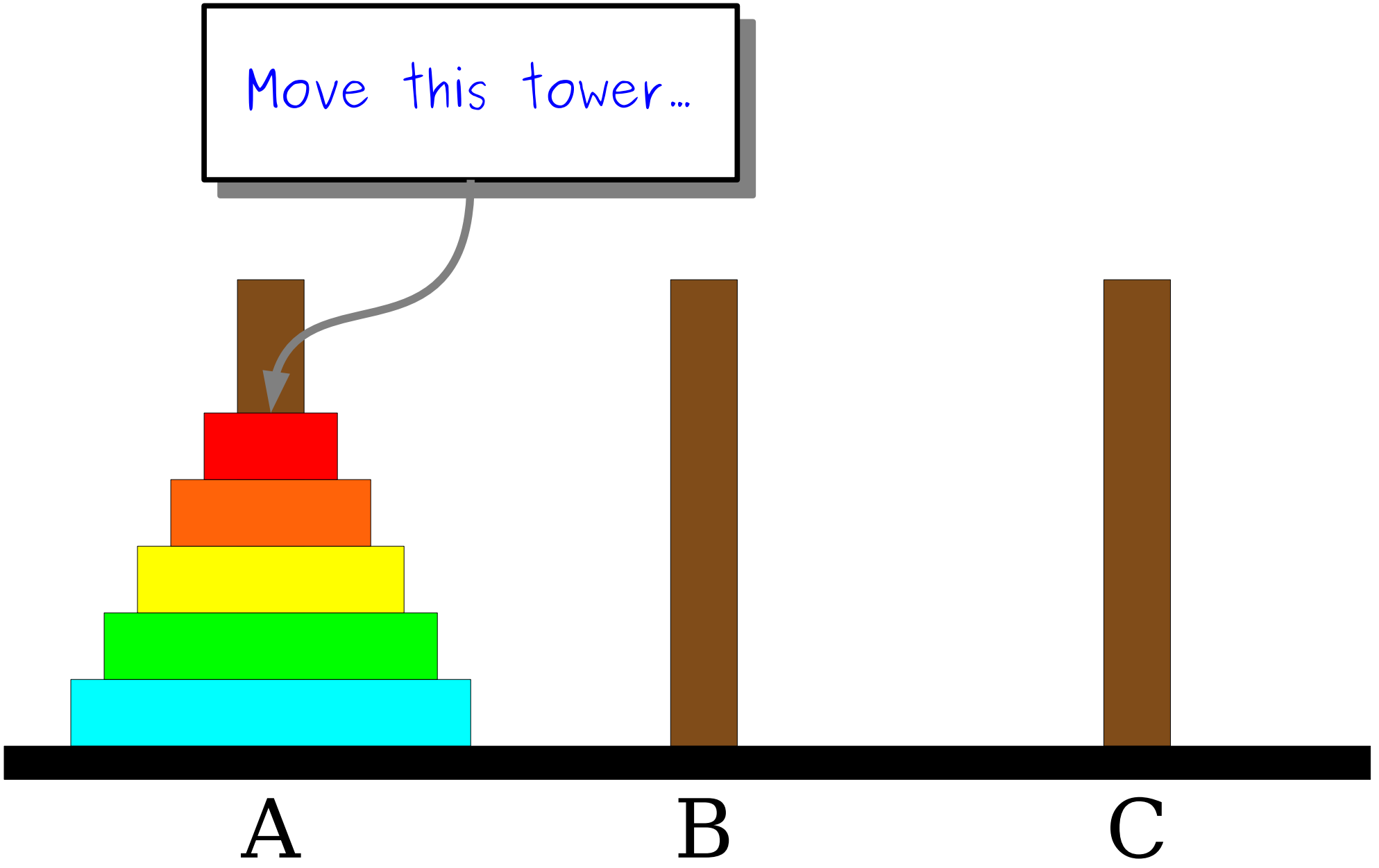
```
if (problem is sufficiently simple) {
   Directly solve the problem.
   Return the solution.
 } else {
   Split the problem up into one or more smaller
      problems with the same structure as the original.
   Solve each of those smaller problems.
   Combine the results to get the overall solution.
   Return the overall solution.
}
```

The Towers of Hanoi Problem

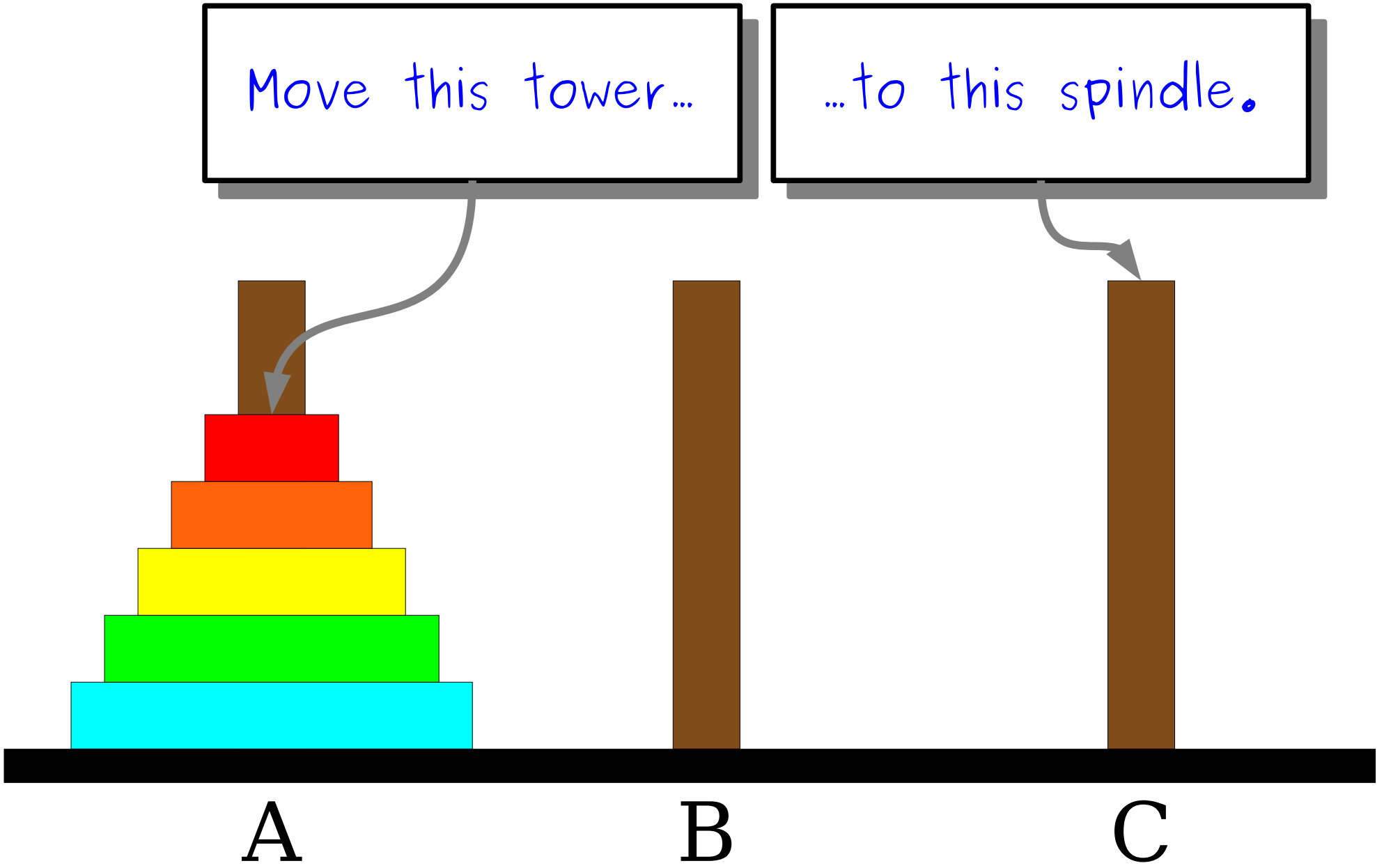
Towers of Hanoi



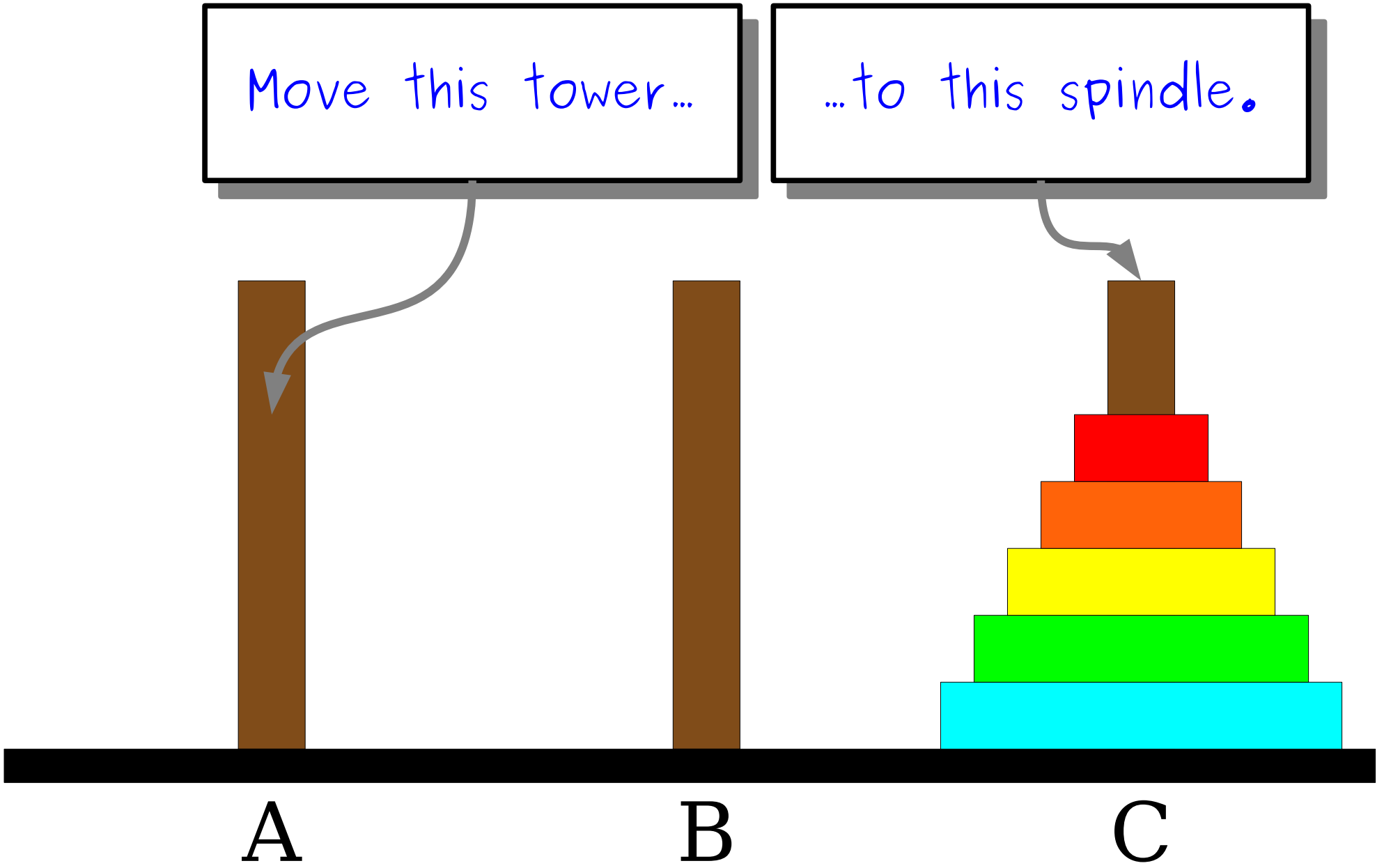
Towers of Hanoi



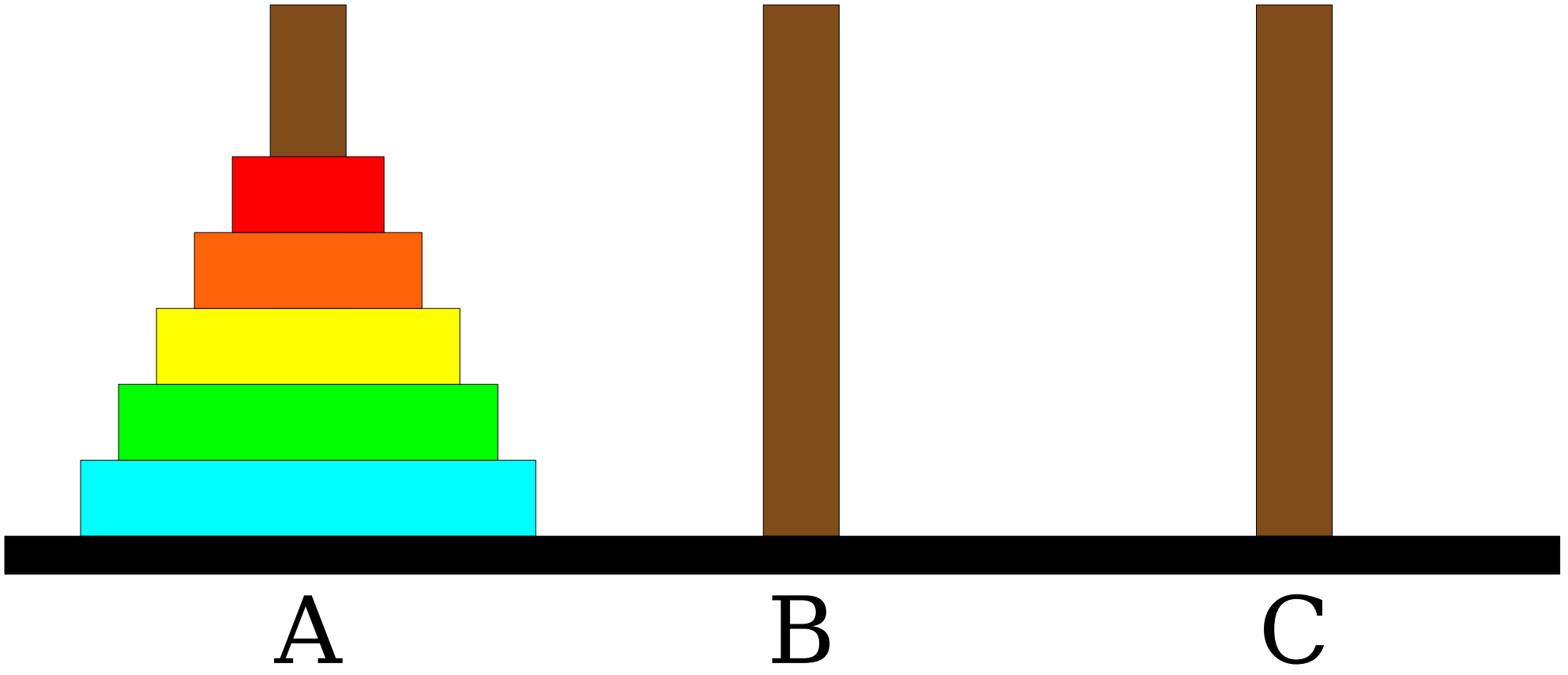
Towers of Hanoi



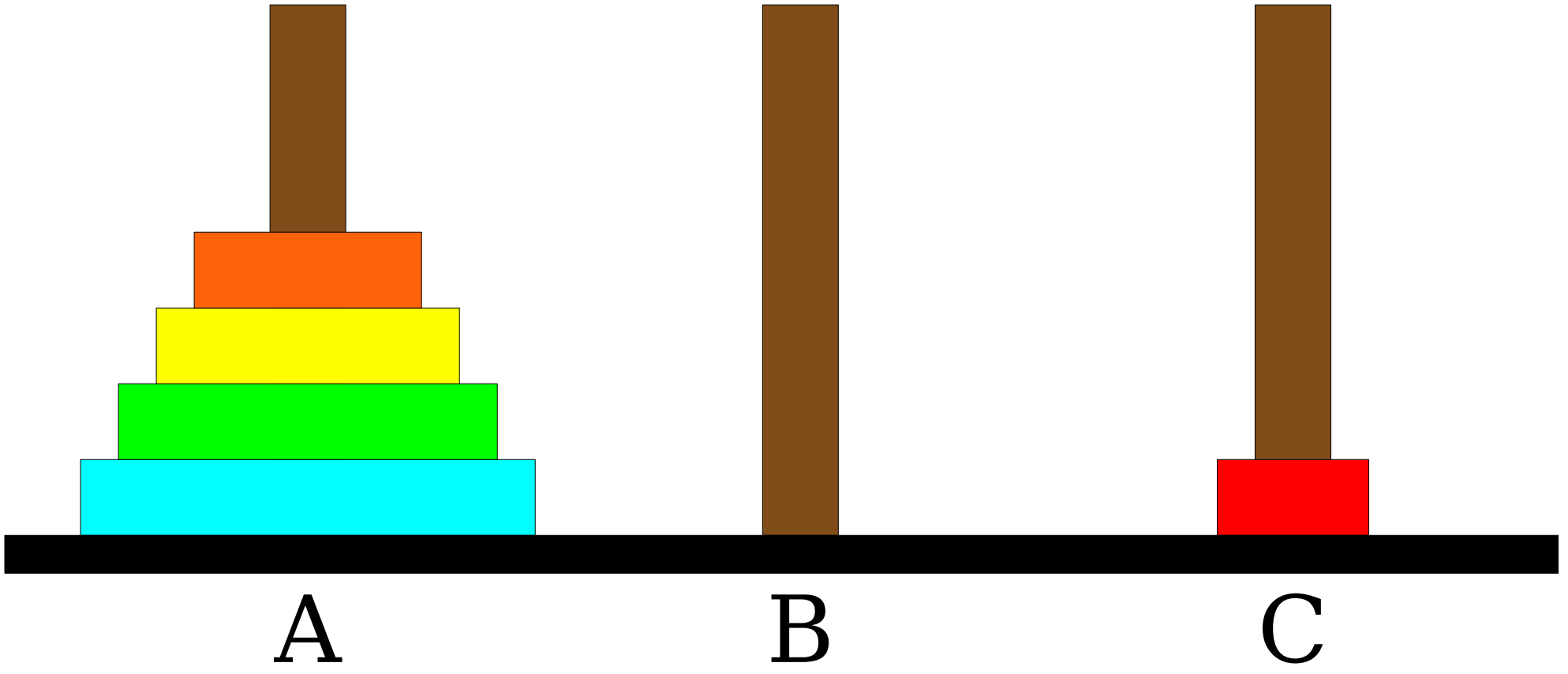
Towers of Hanoi



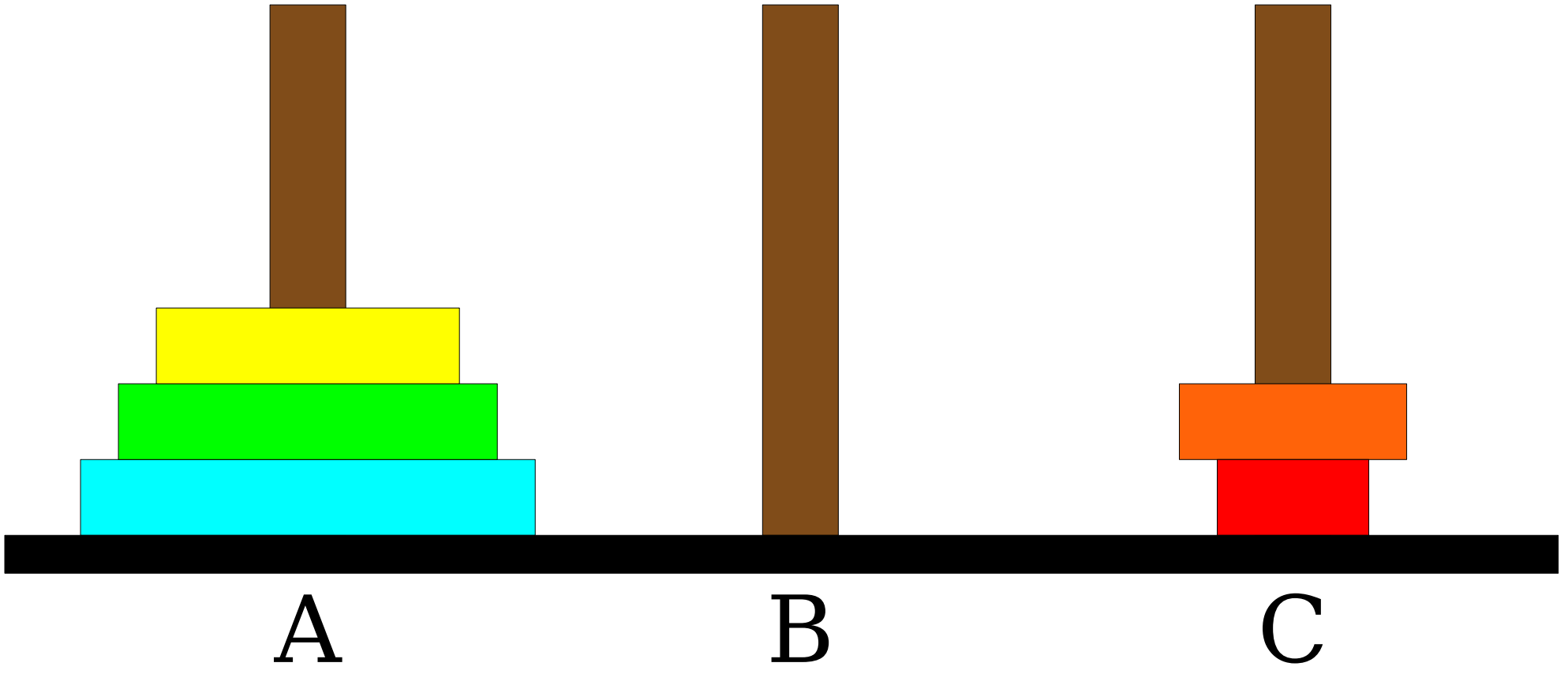
Towers of Hanoi



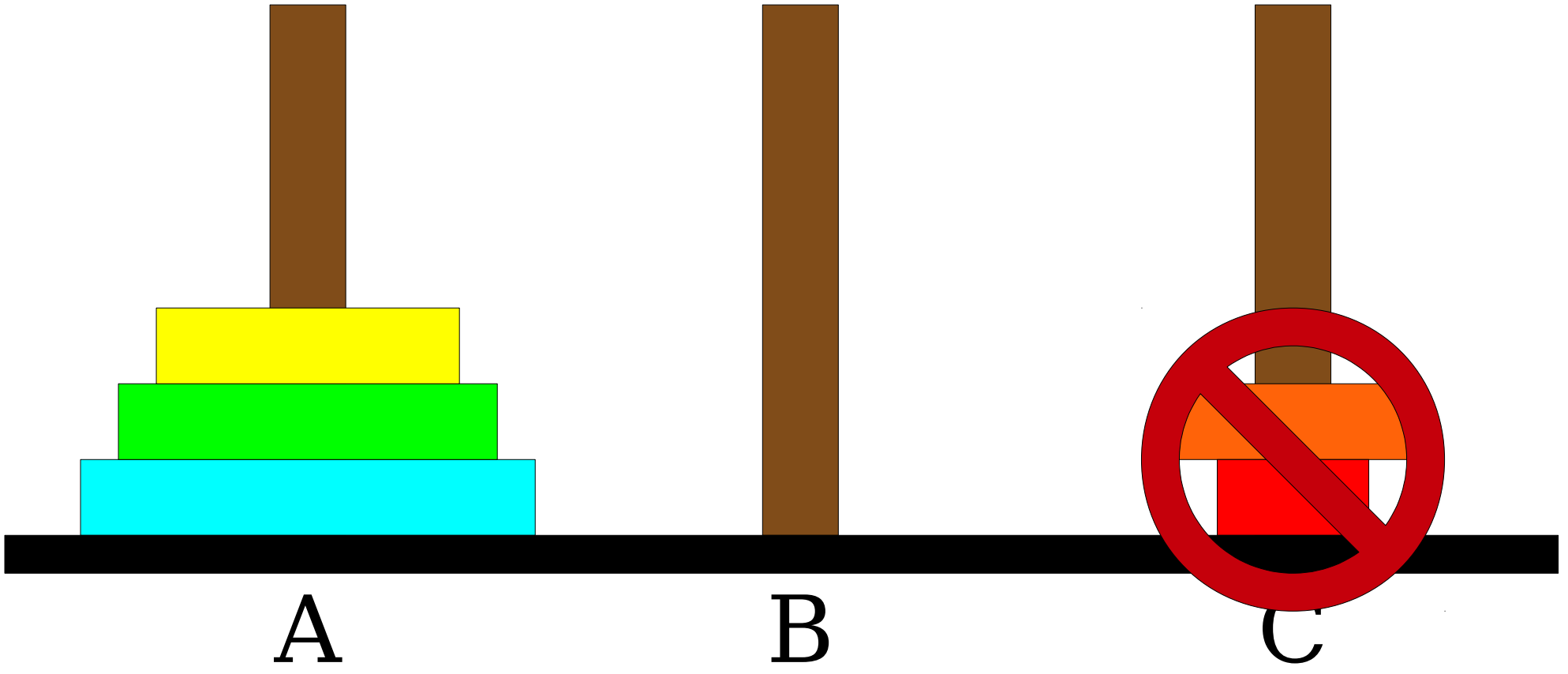
Towers of Hanoi



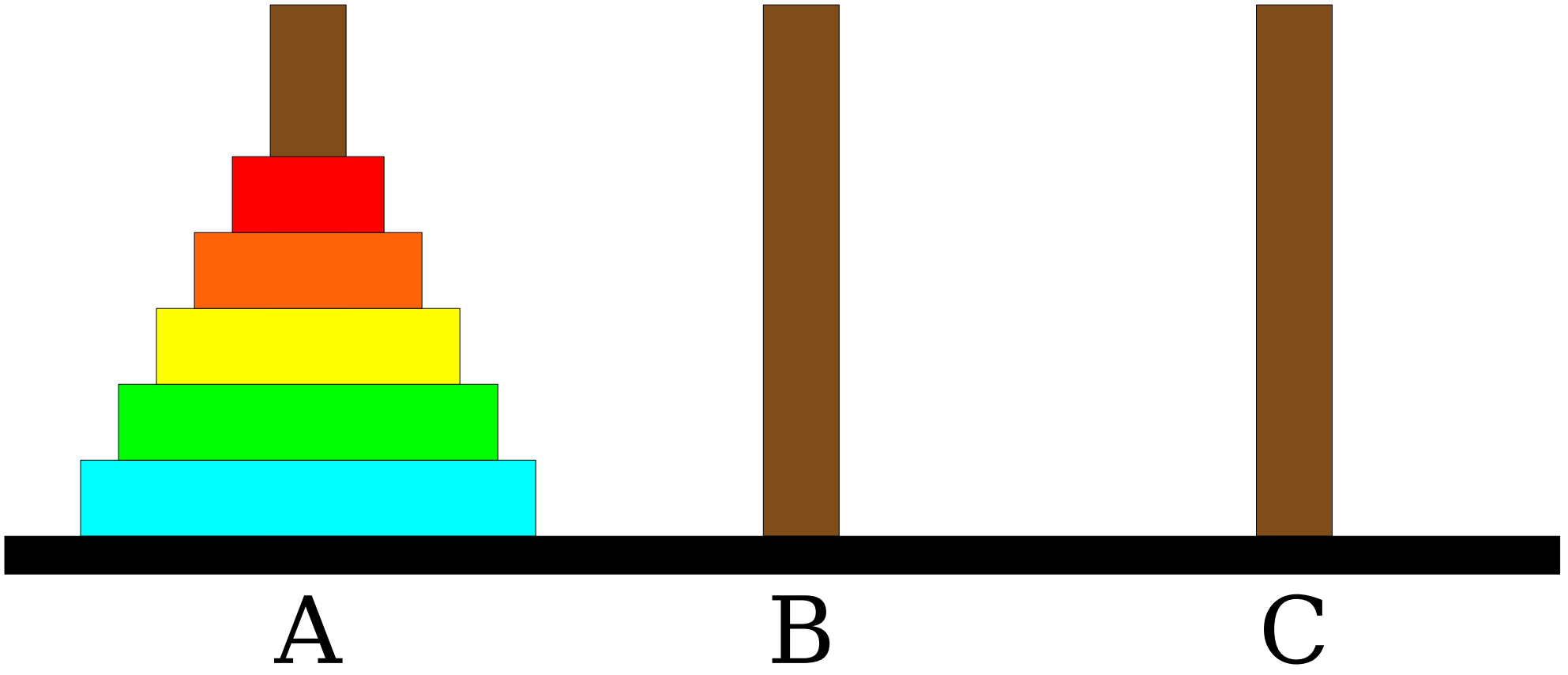
Towers of Hanoi



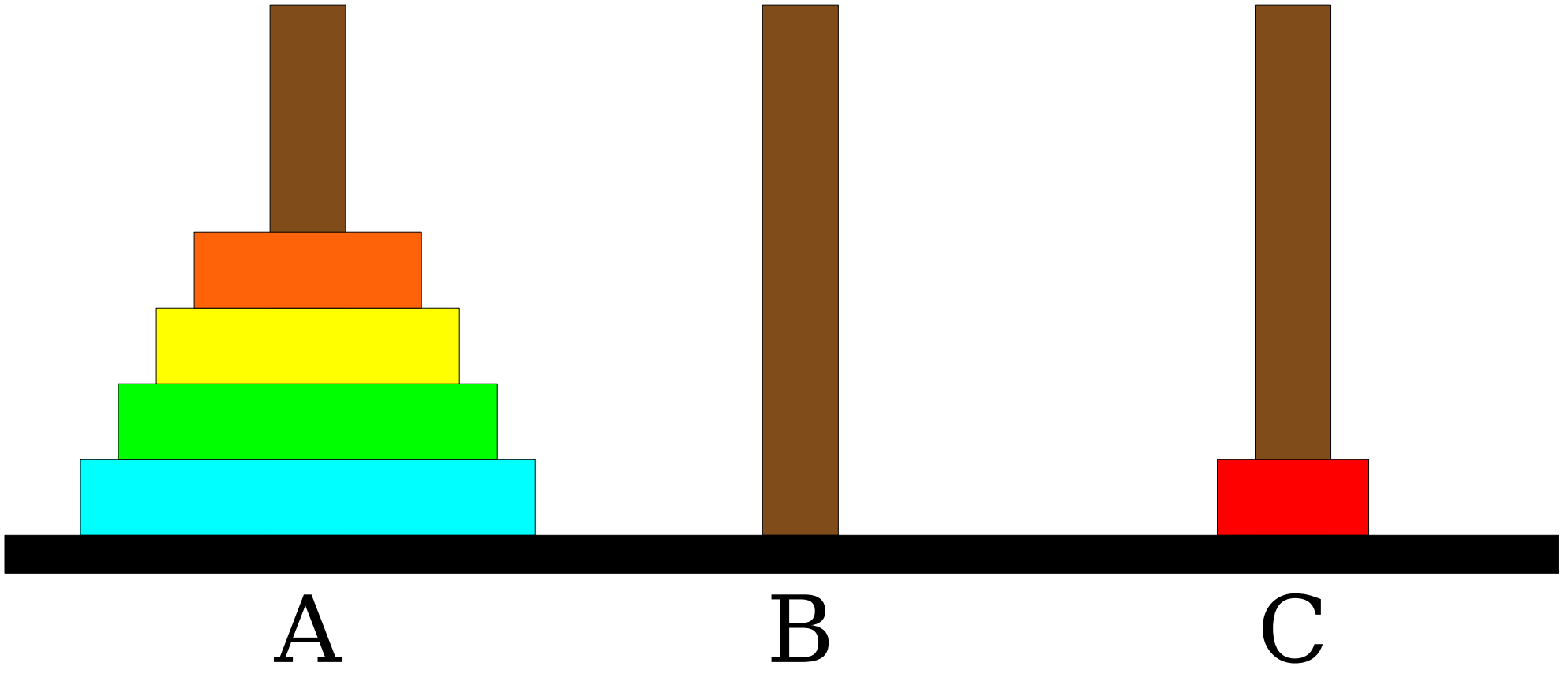
Towers of Hanoi



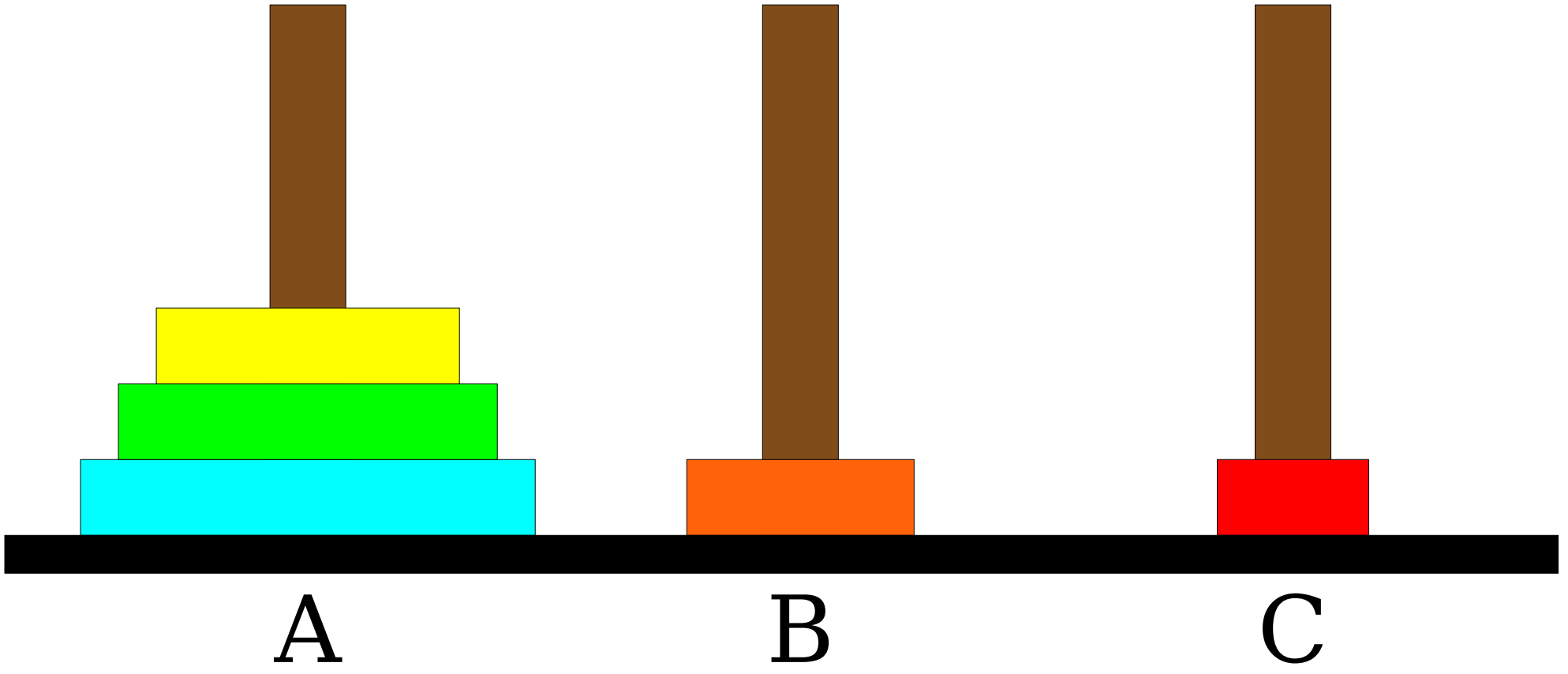
Towers of Hanoi



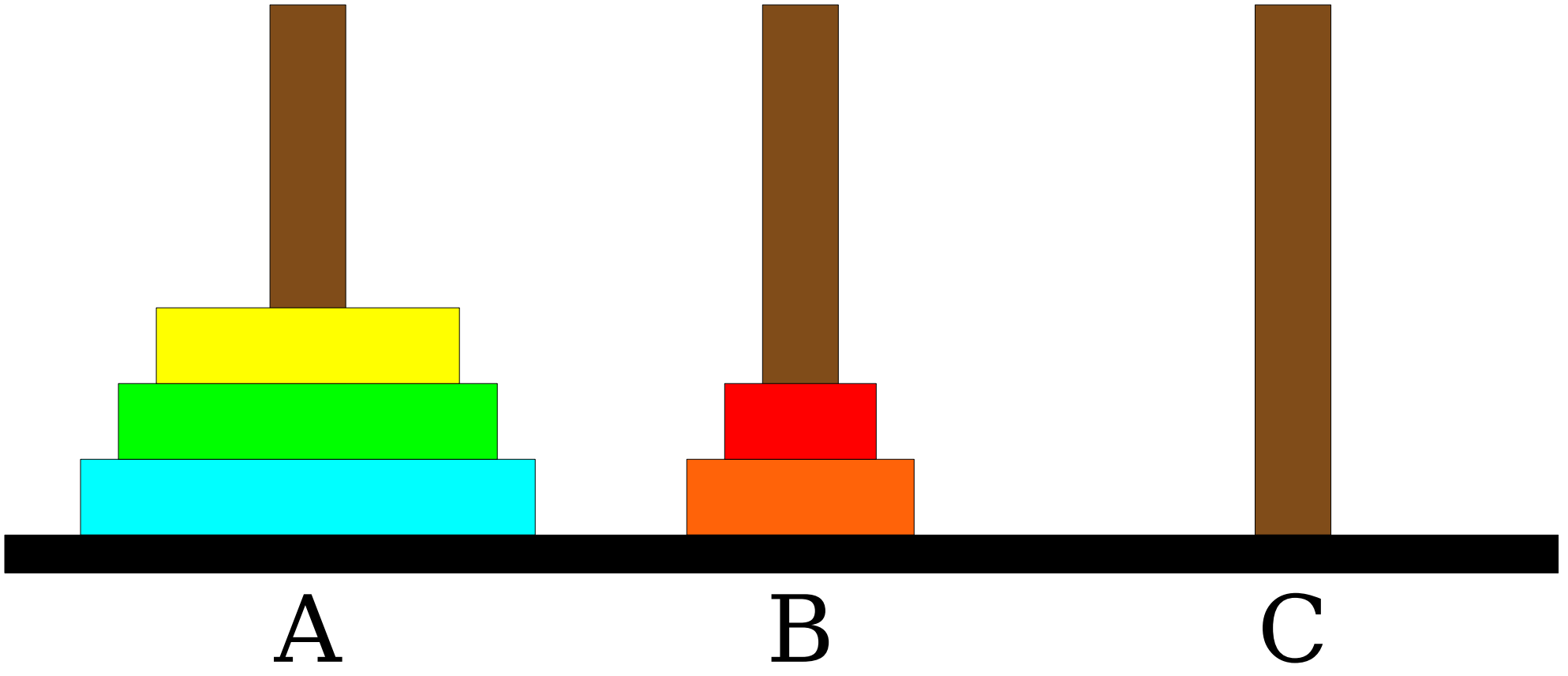
Towers of Hanoi



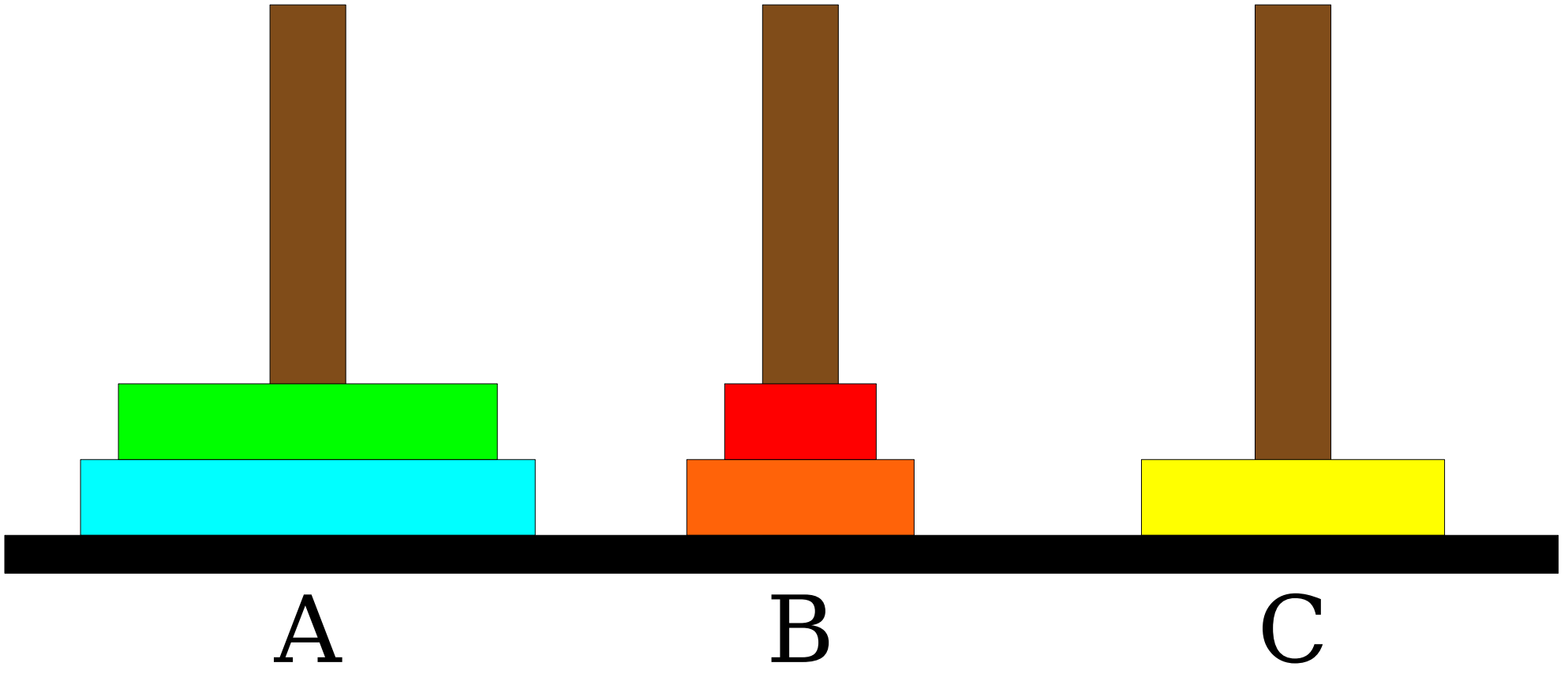
Towers of Hanoi



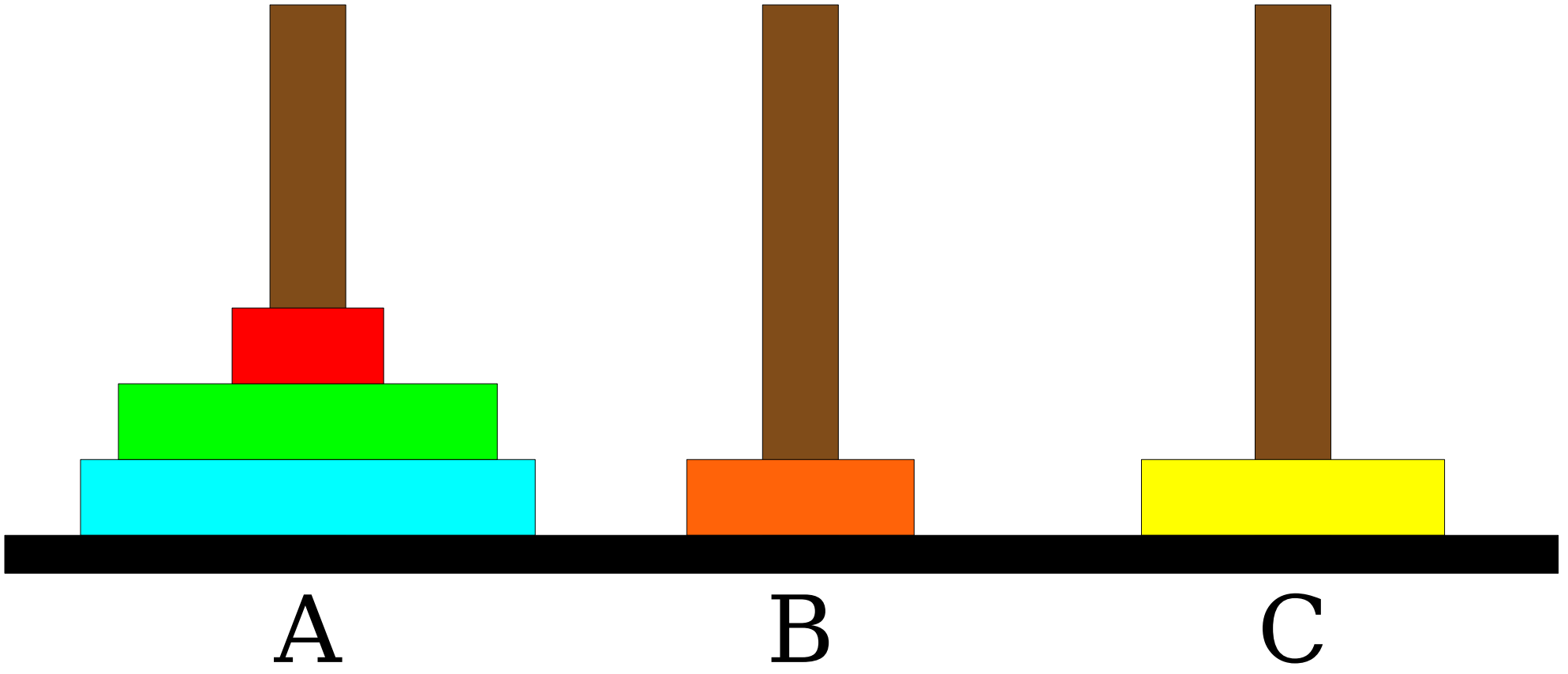
Towers of Hanoi



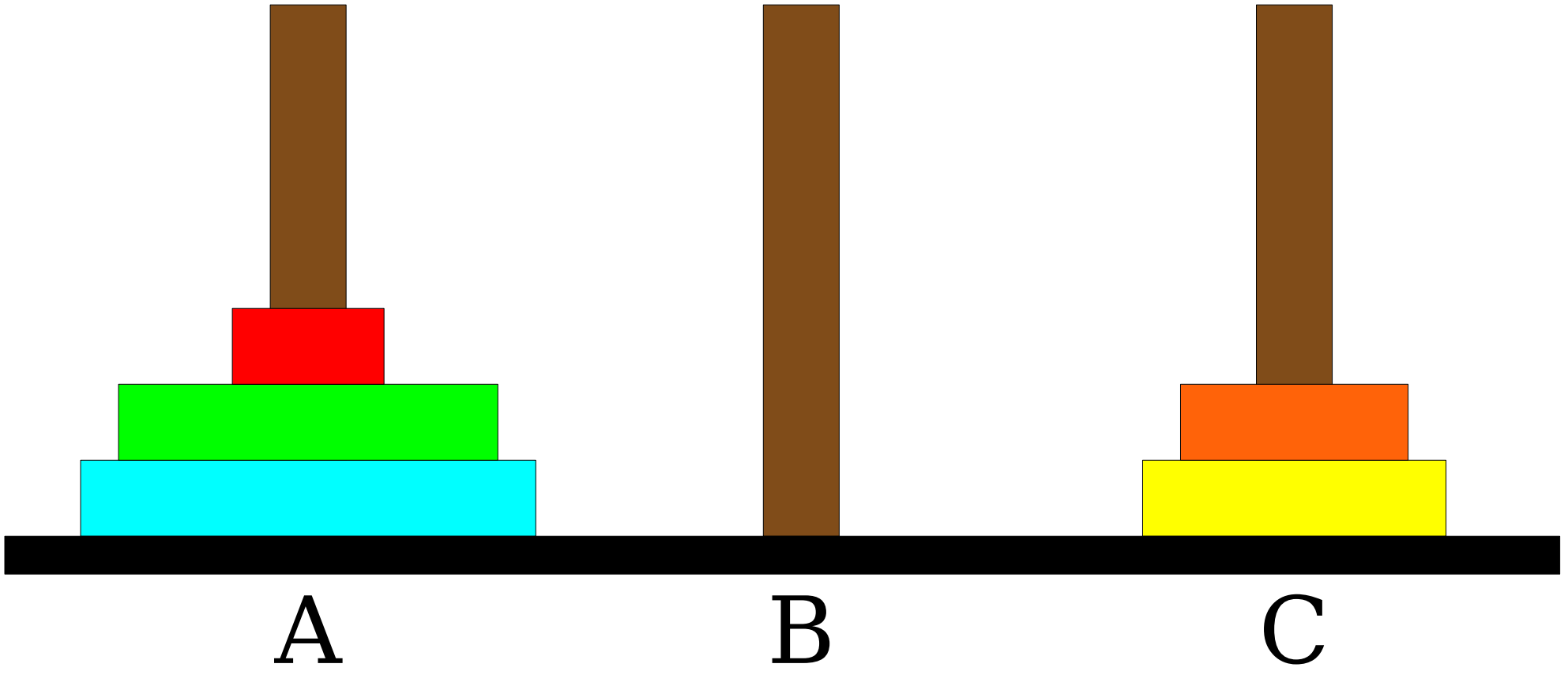
Towers of Hanoi



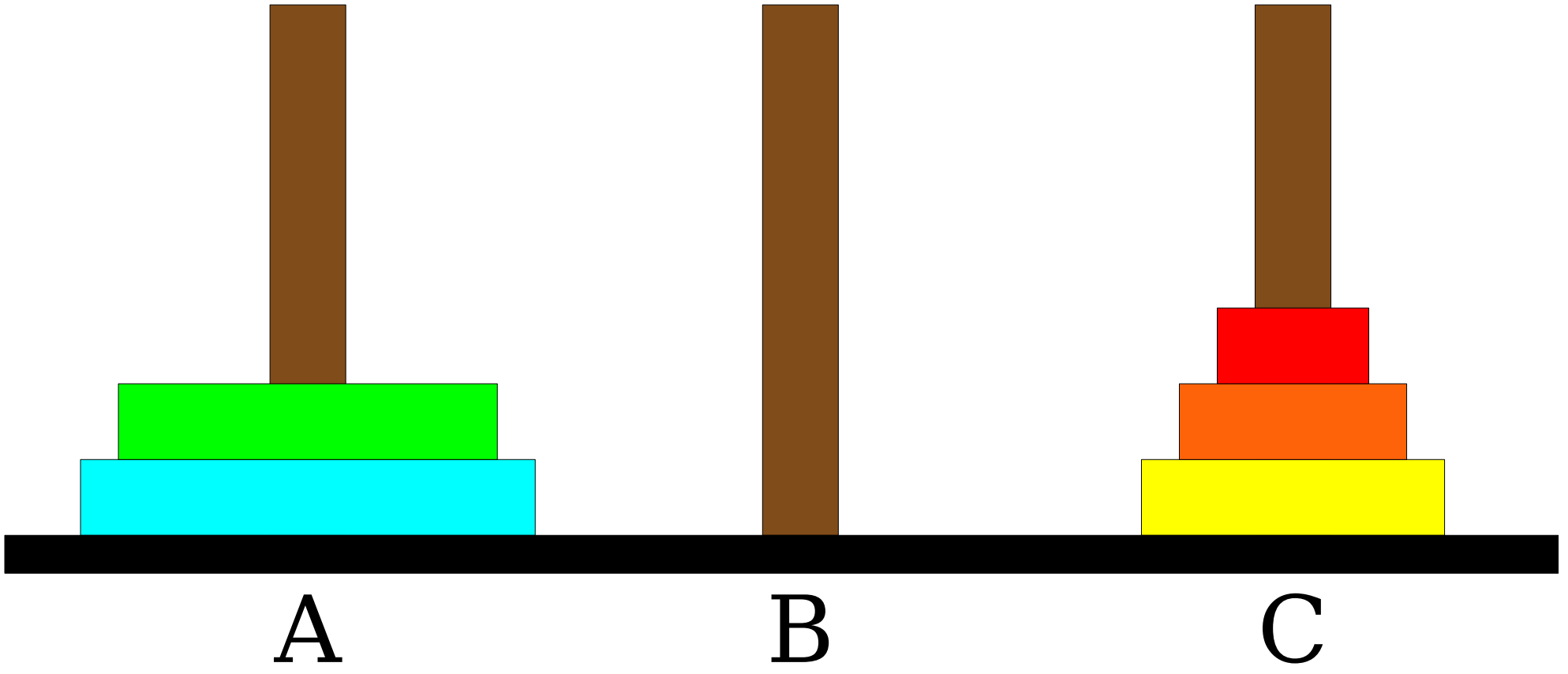
Towers of Hanoi



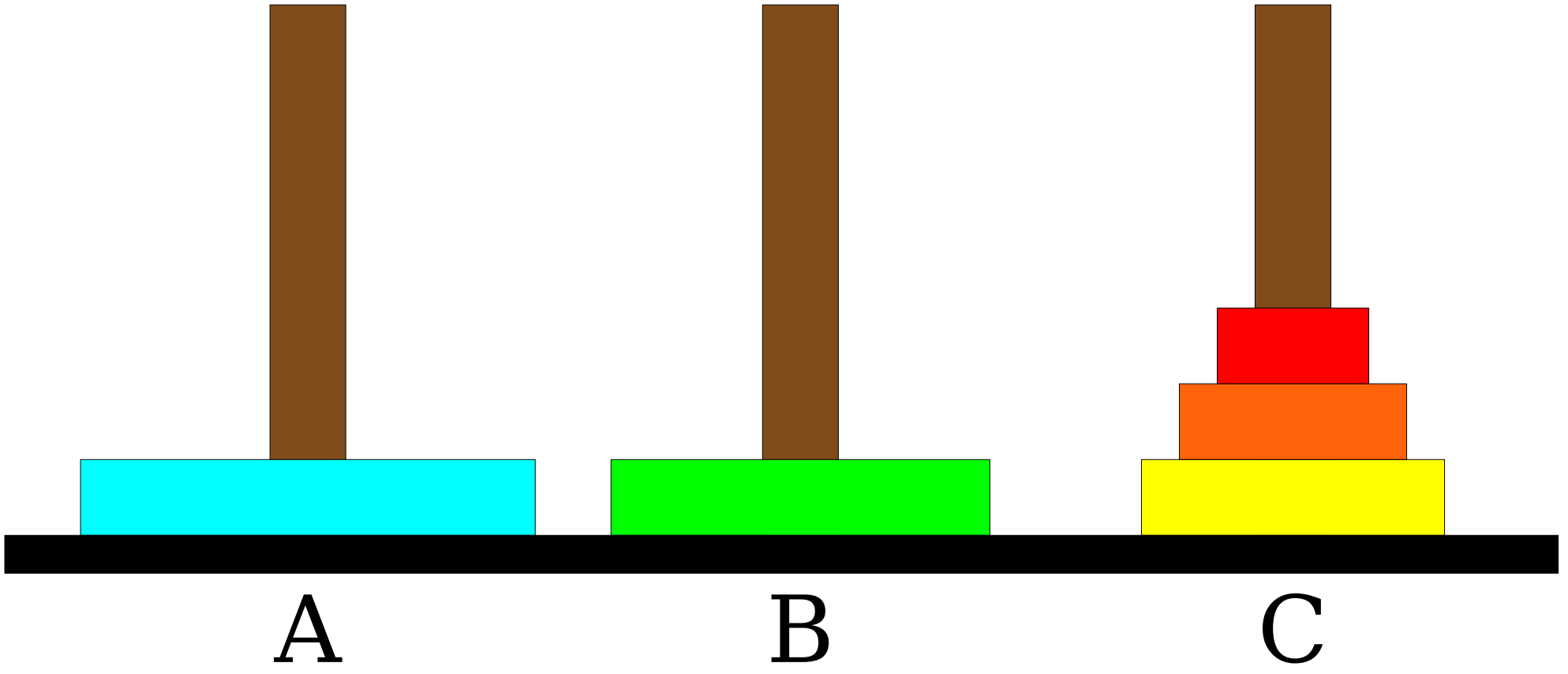
Towers of Hanoi



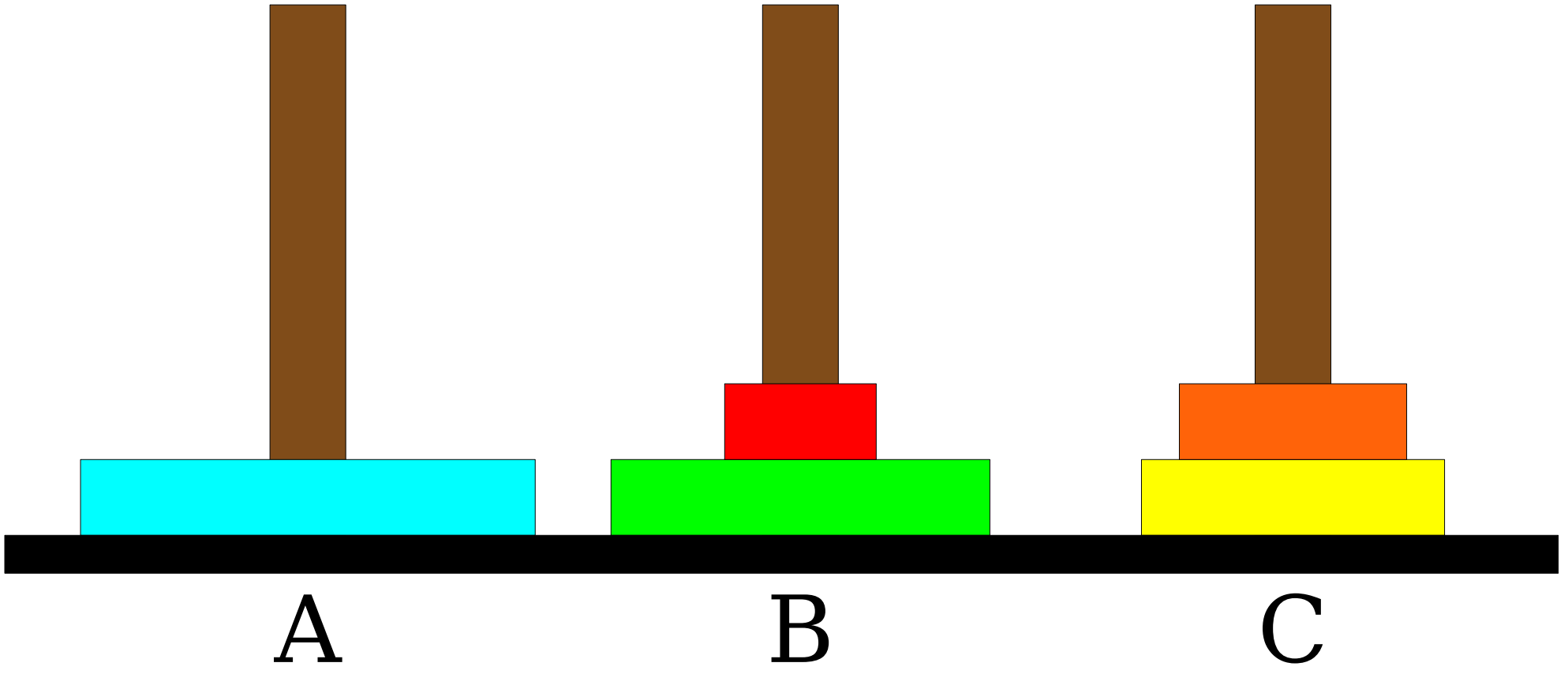
Towers of Hanoi



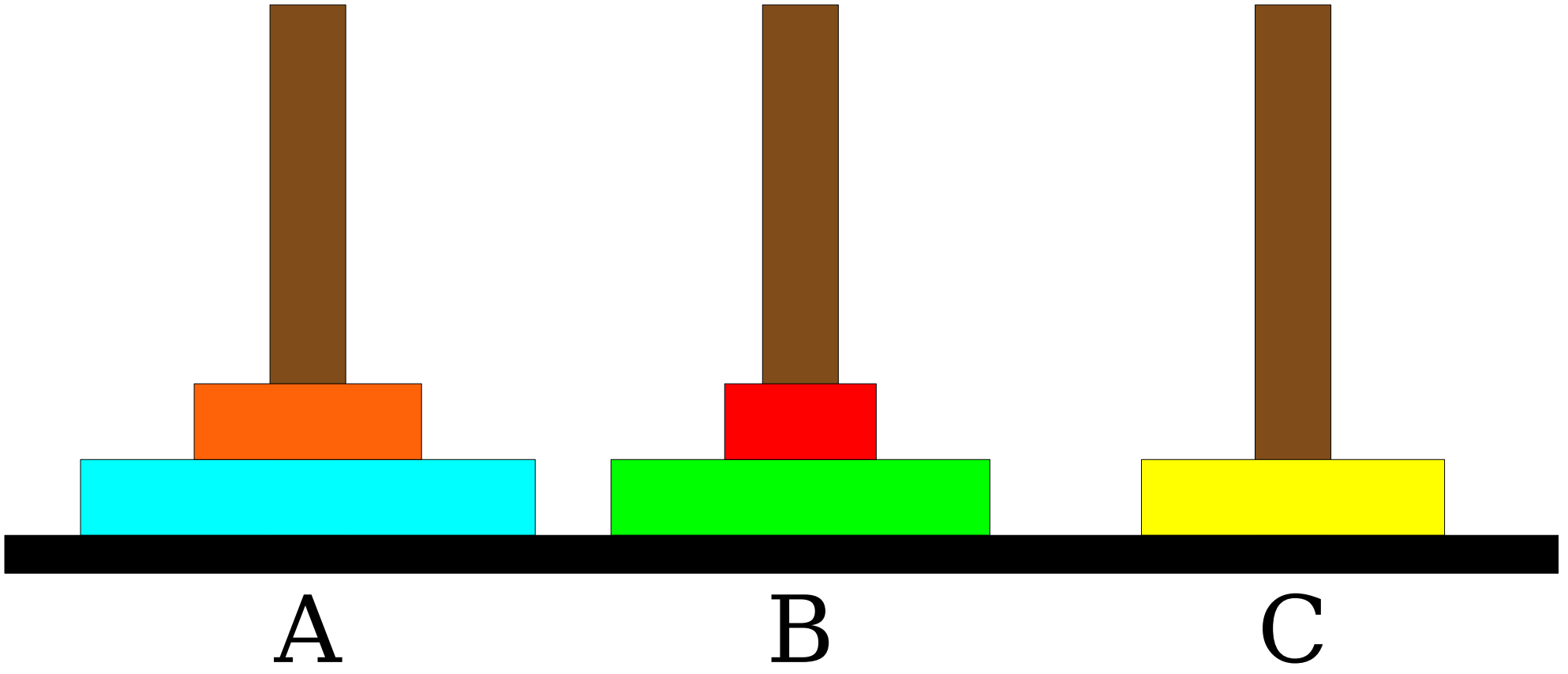
Towers of Hanoi



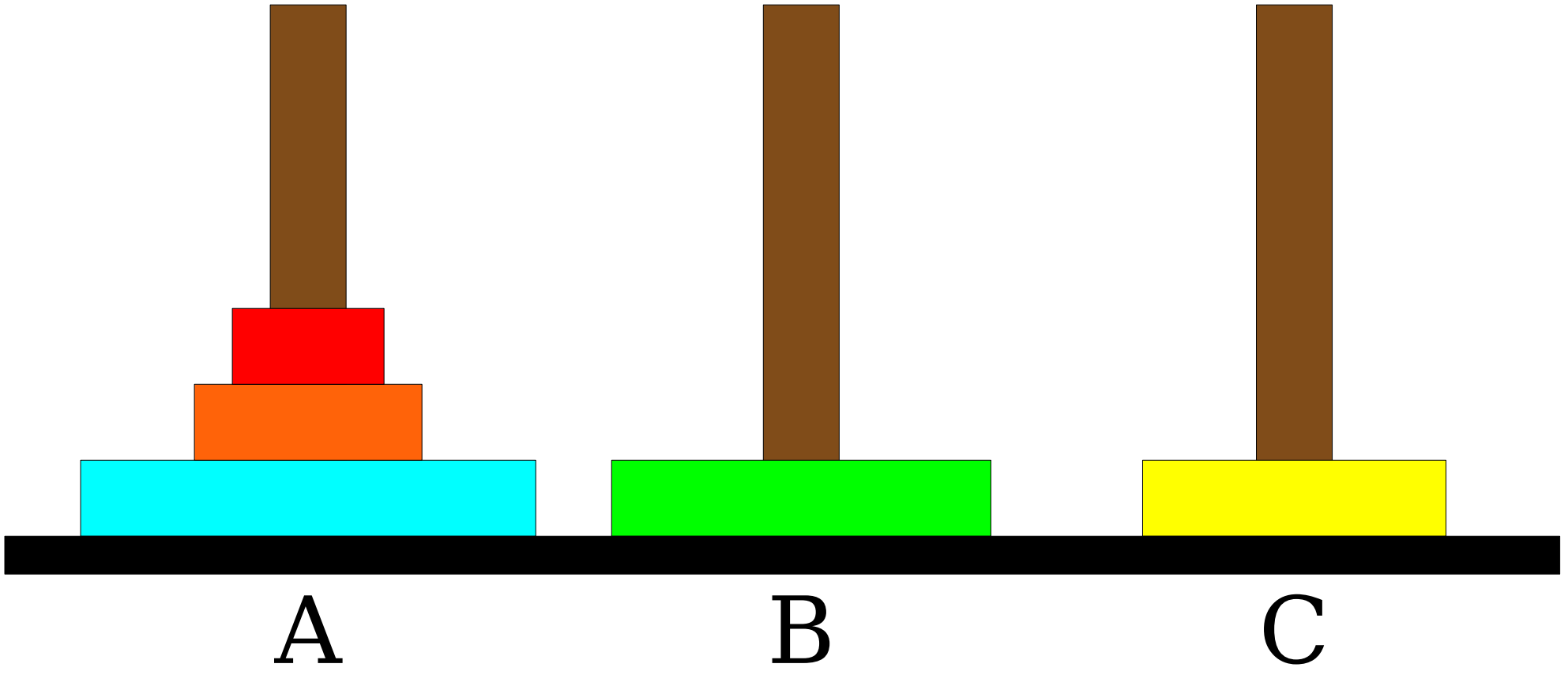
Towers of Hanoi



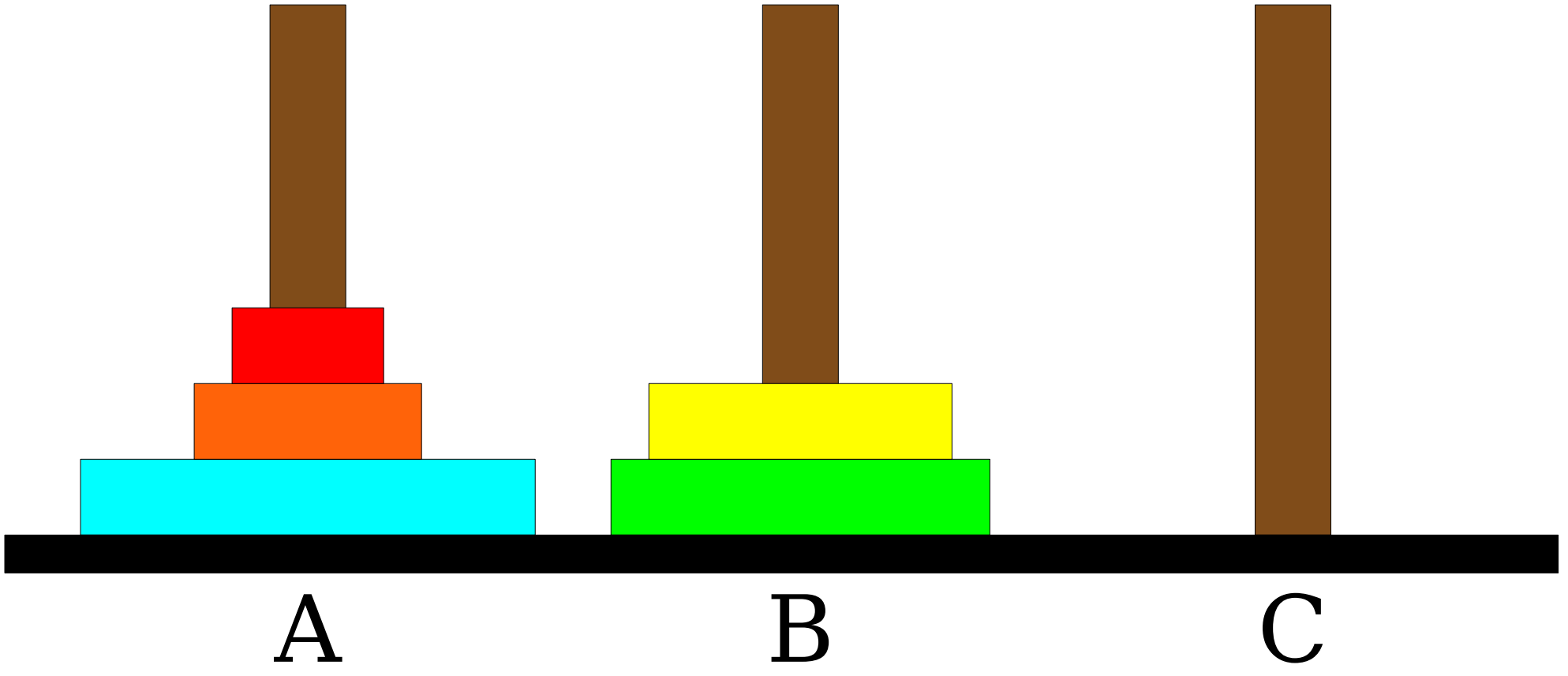
Towers of Hanoi



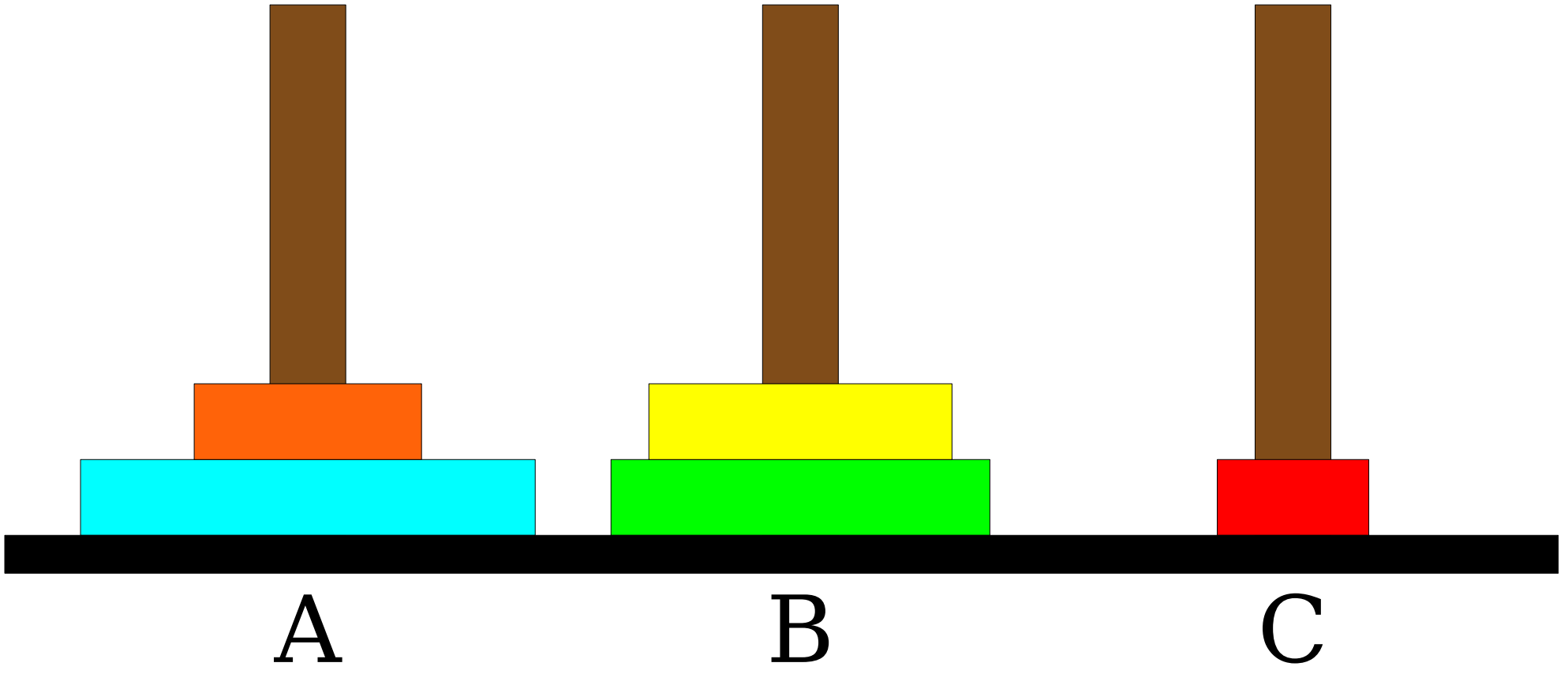
Towers of Hanoi



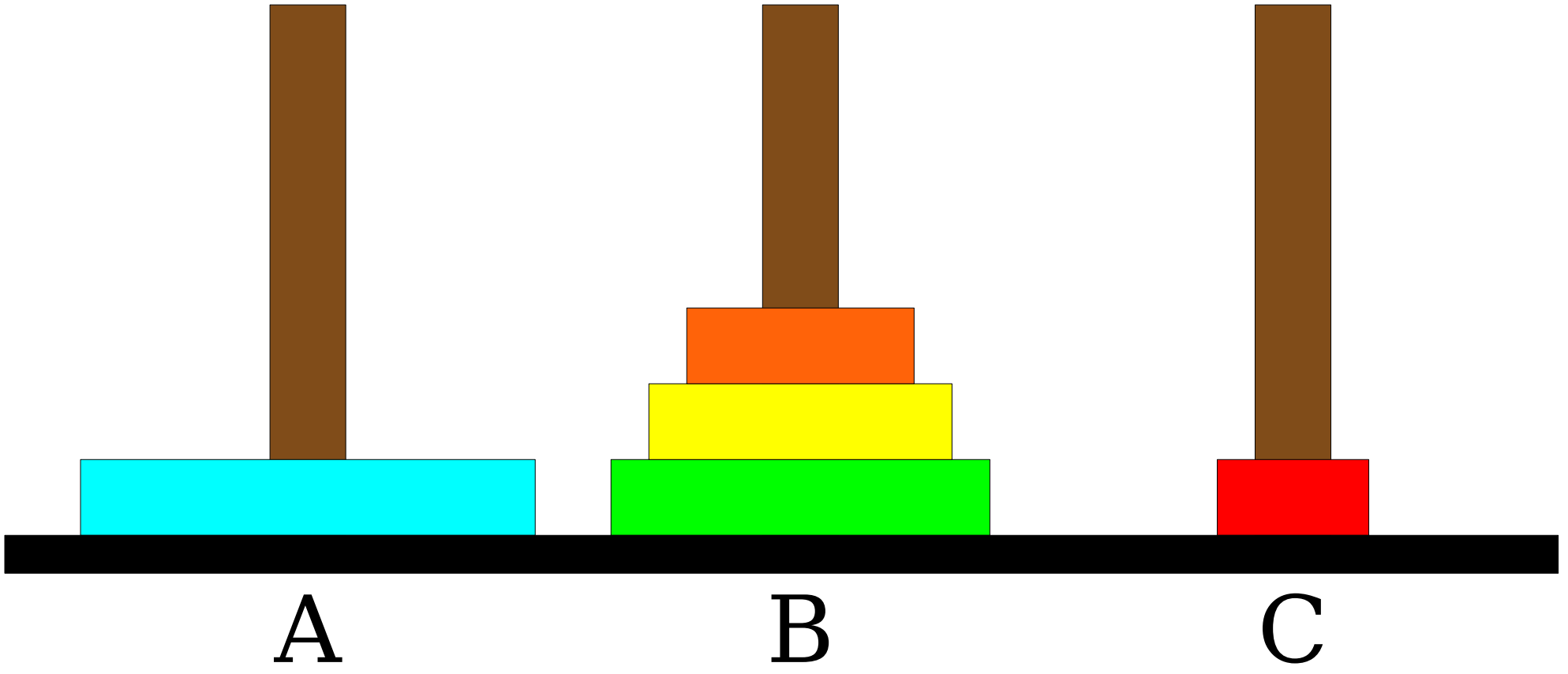
Towers of Hanoi



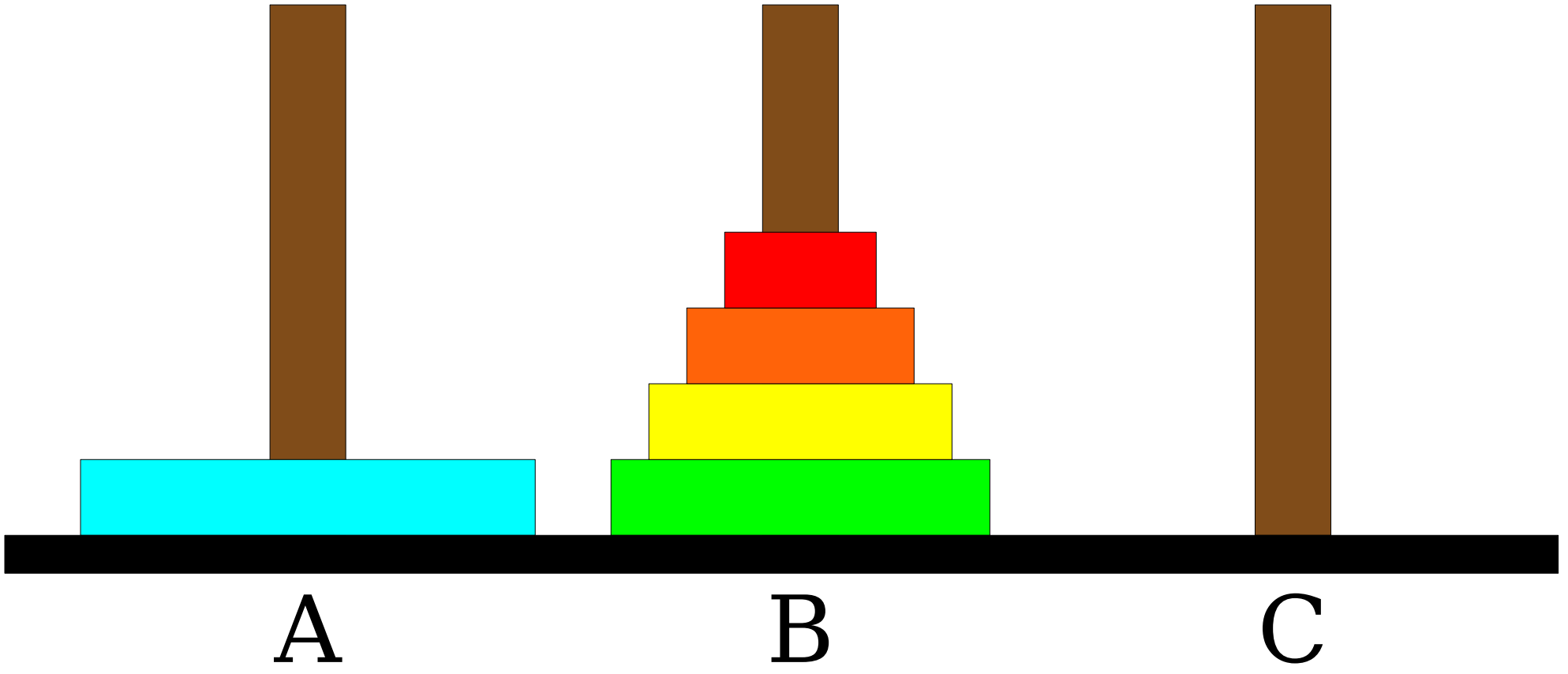
Towers of Hanoi



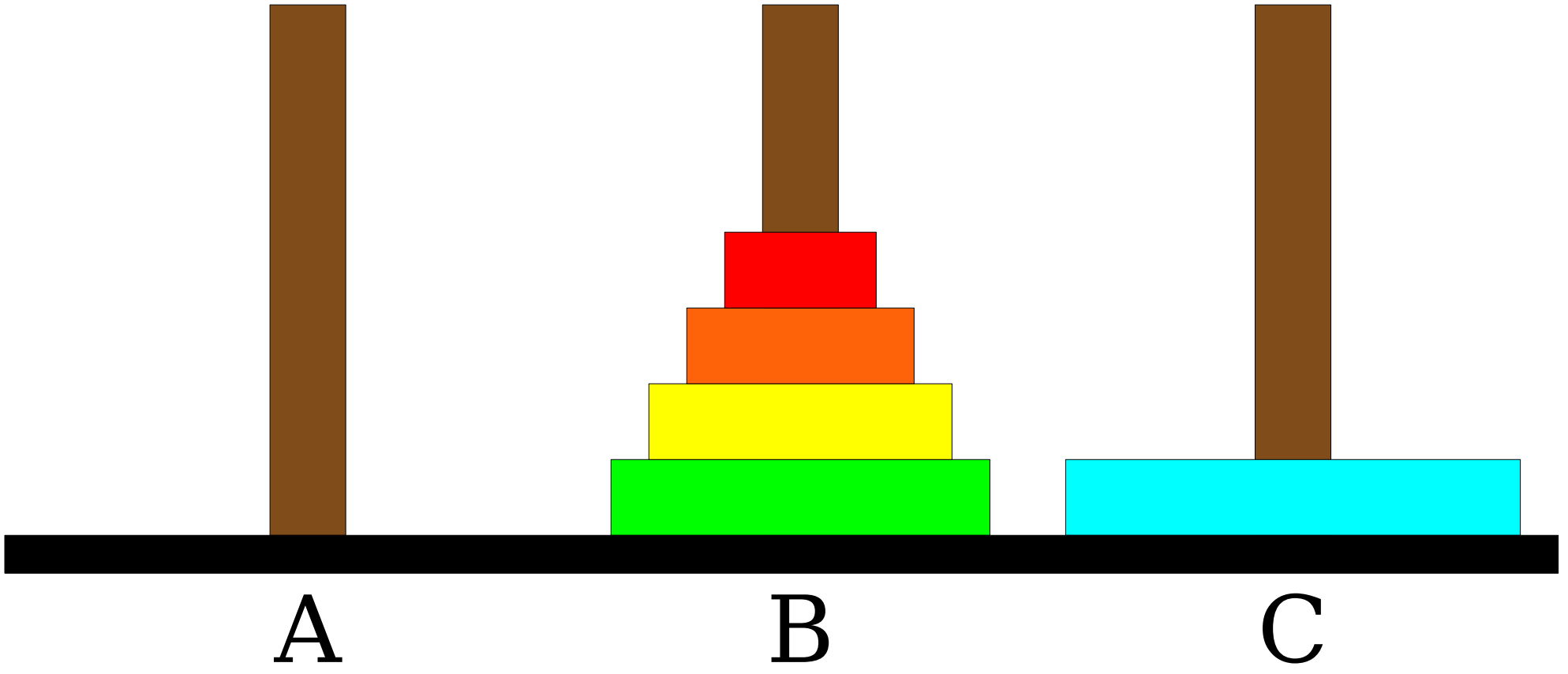
Towers of Hanoi



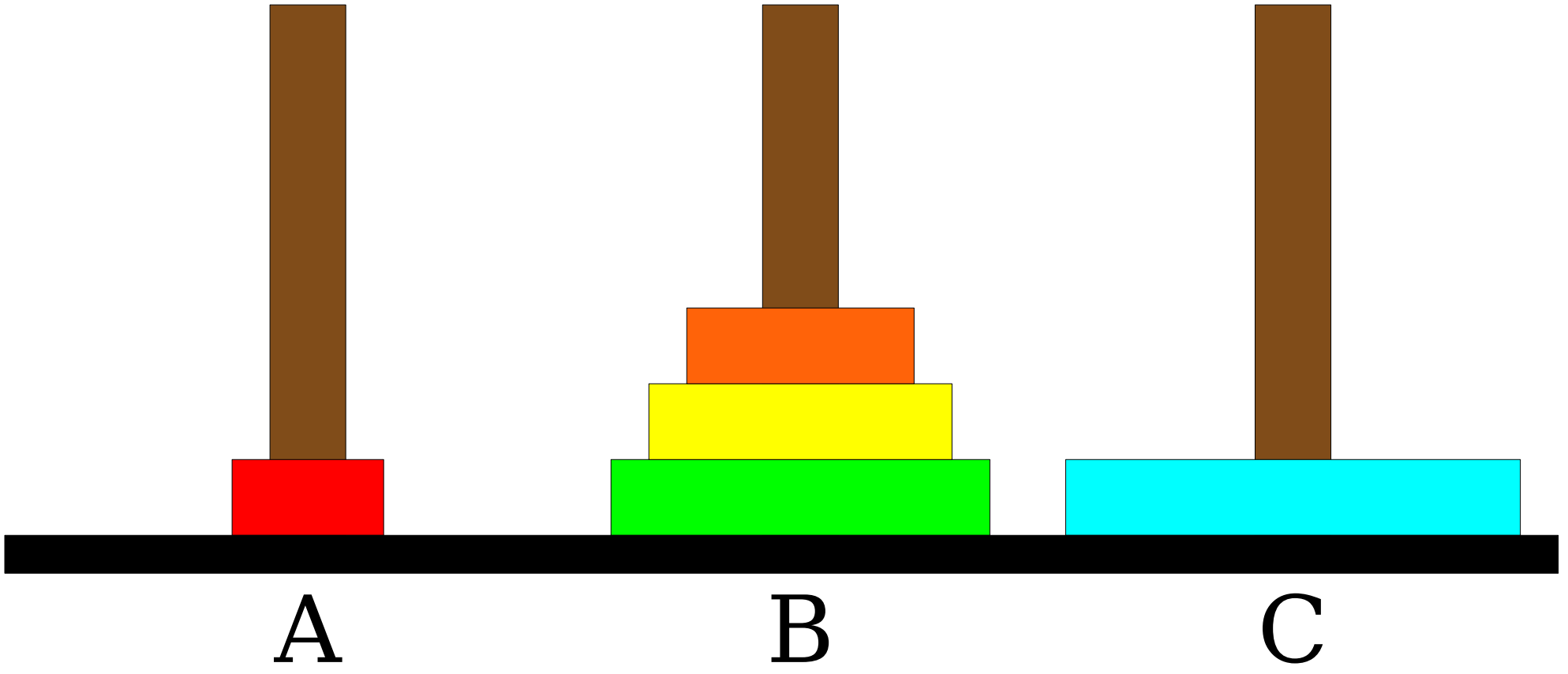
Towers of Hanoi



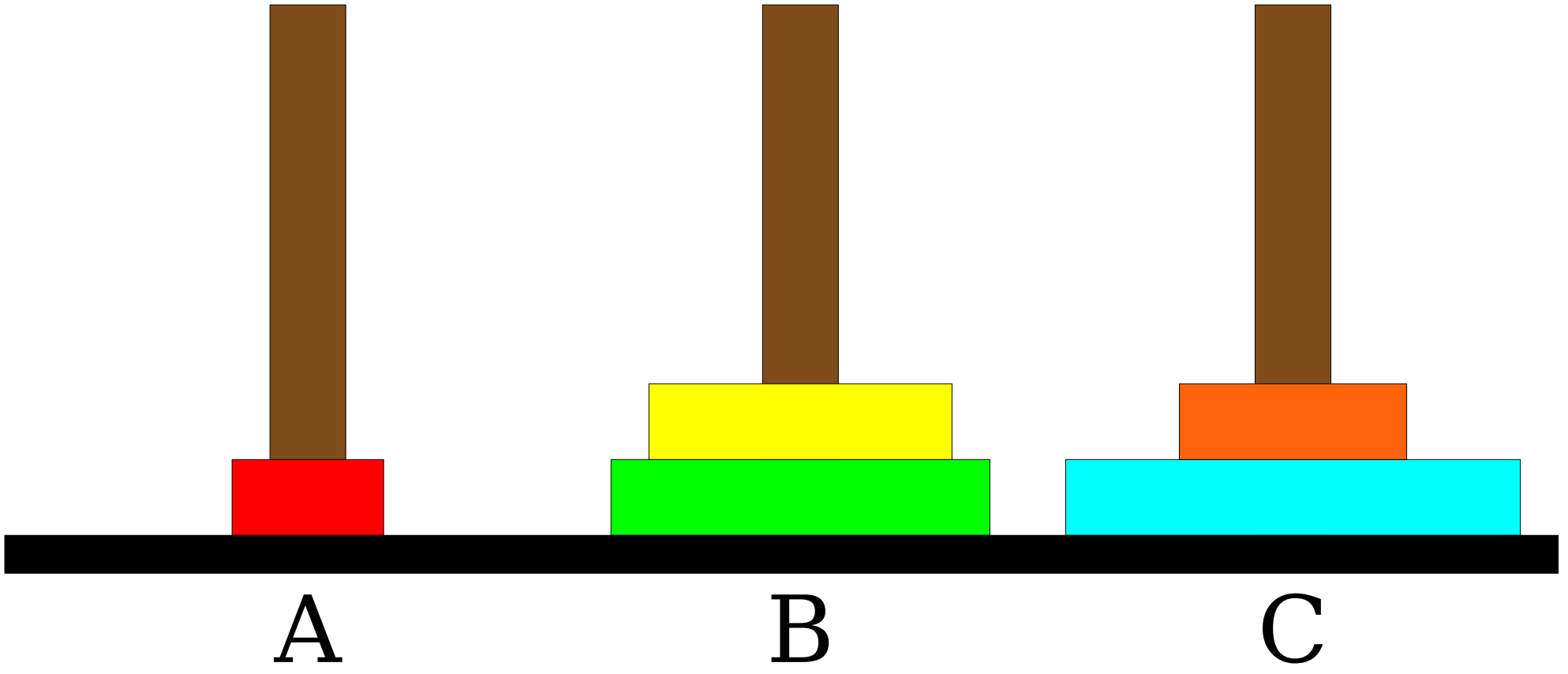
Towers of Hanoi



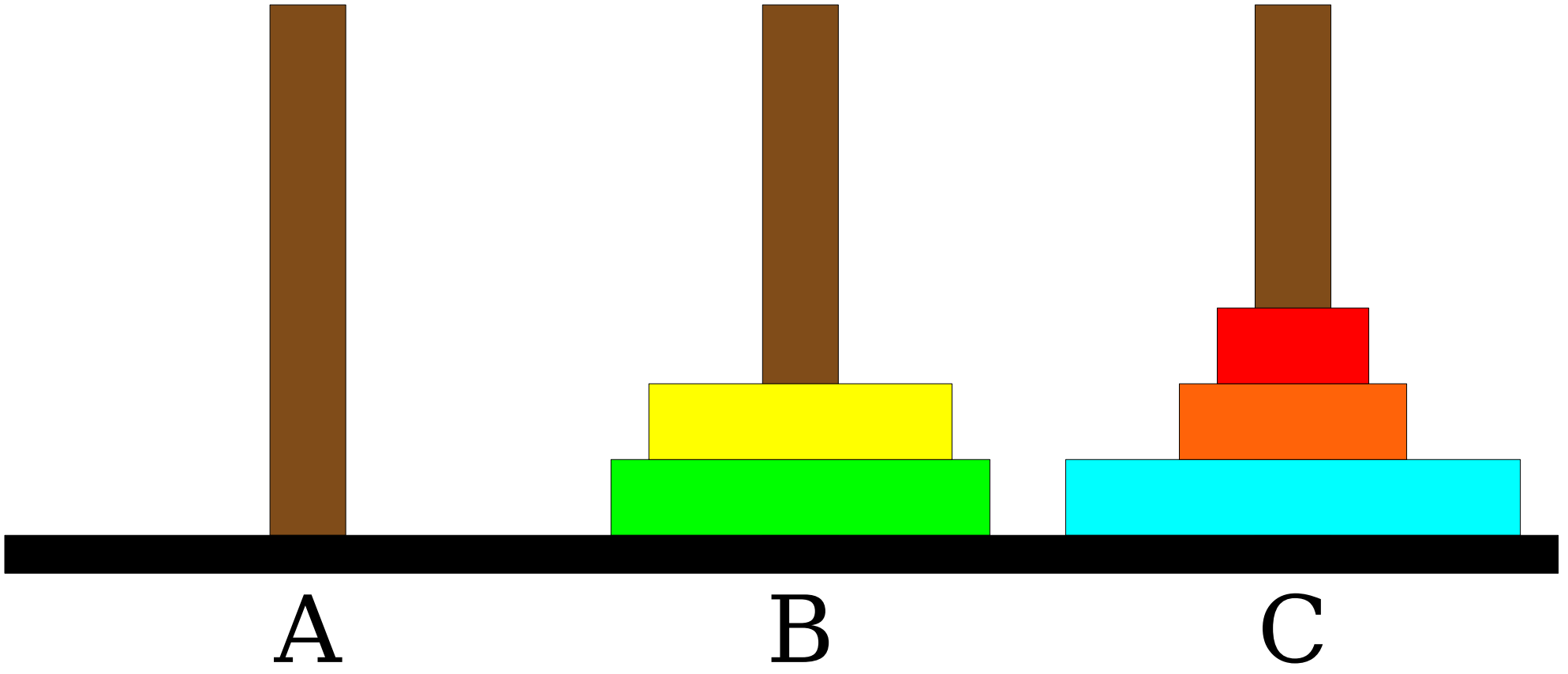
Towers of Hanoi



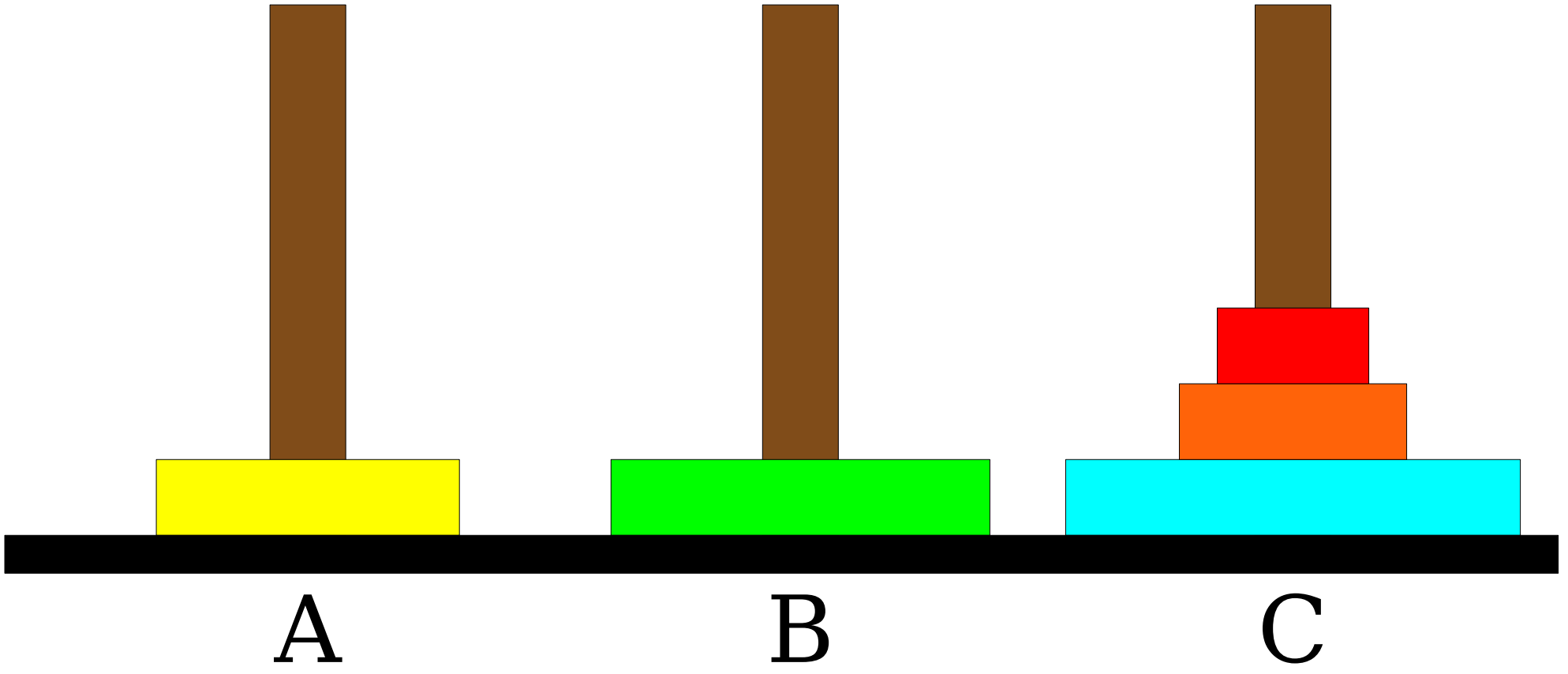
Towers of Hanoi



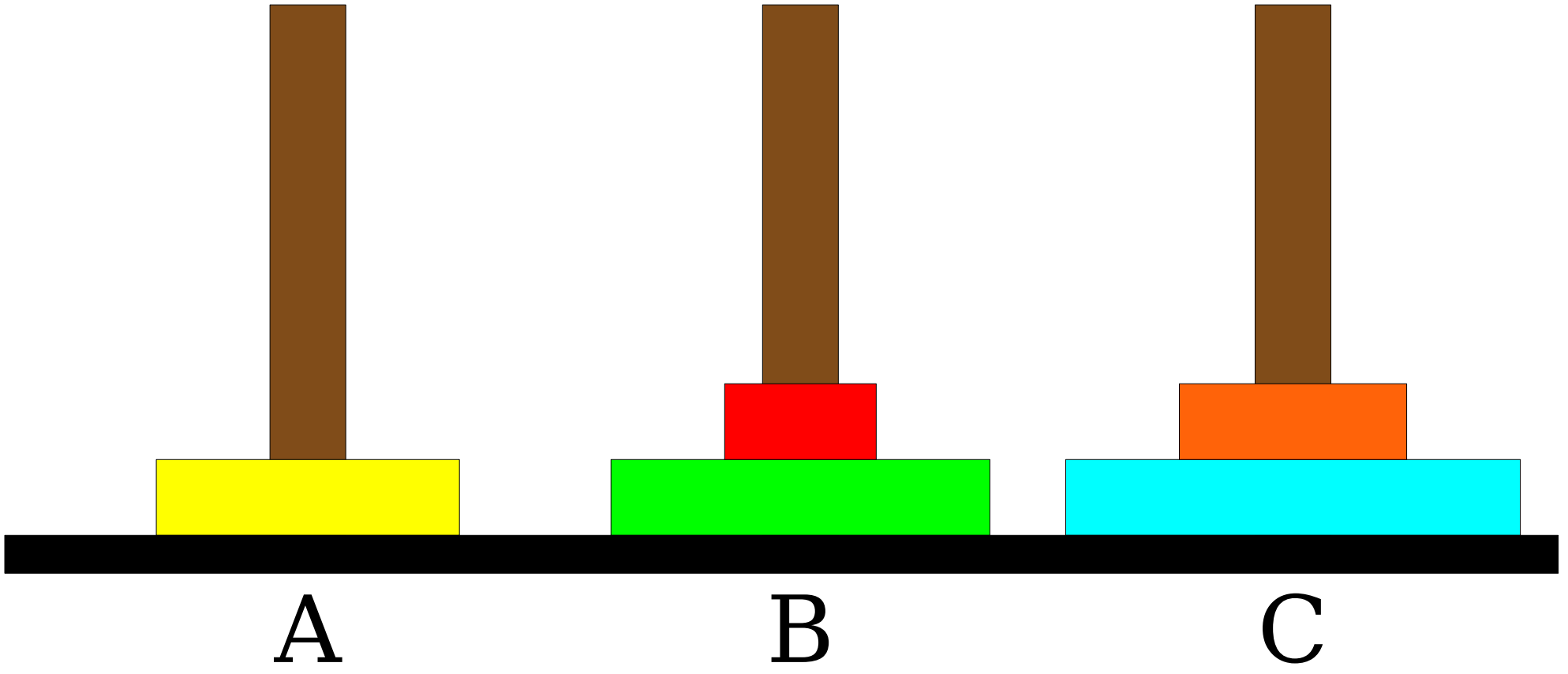
Towers of Hanoi



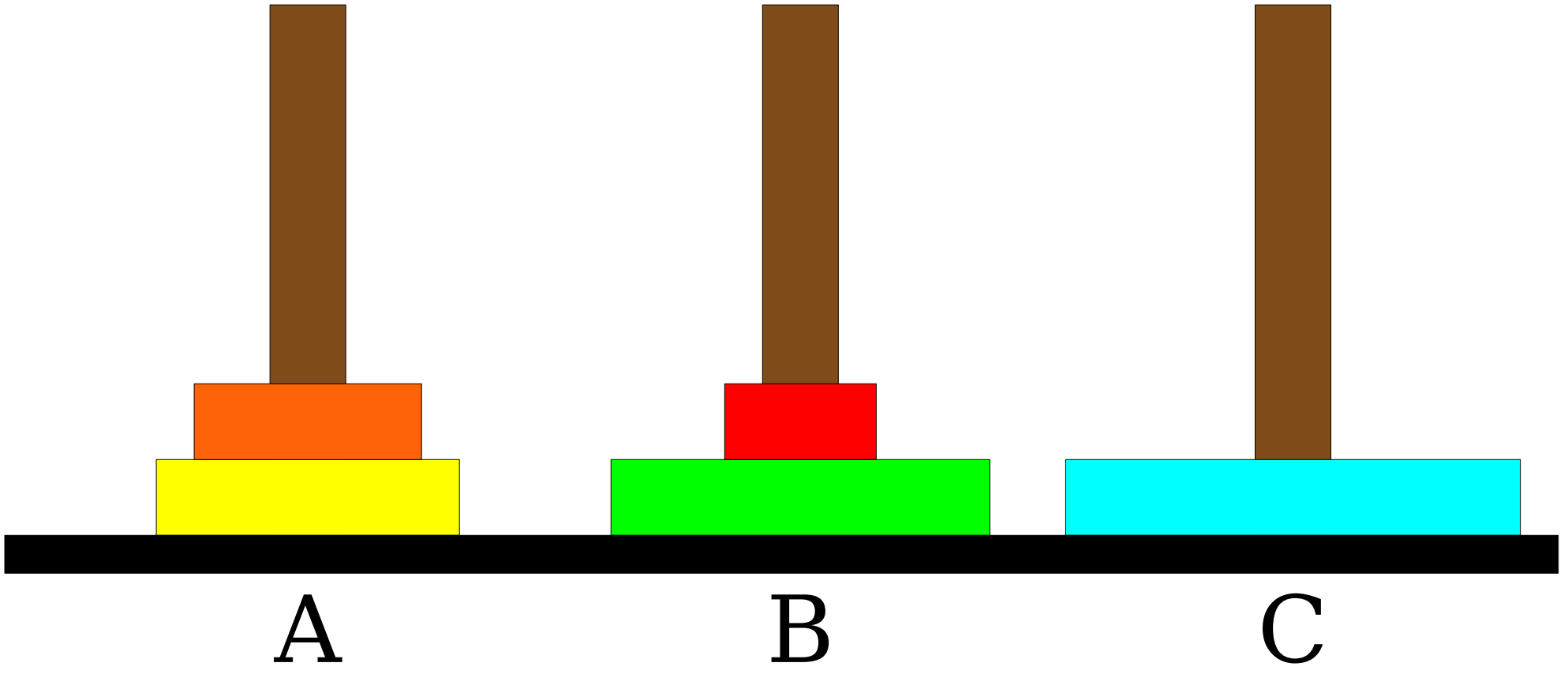
Towers of Hanoi



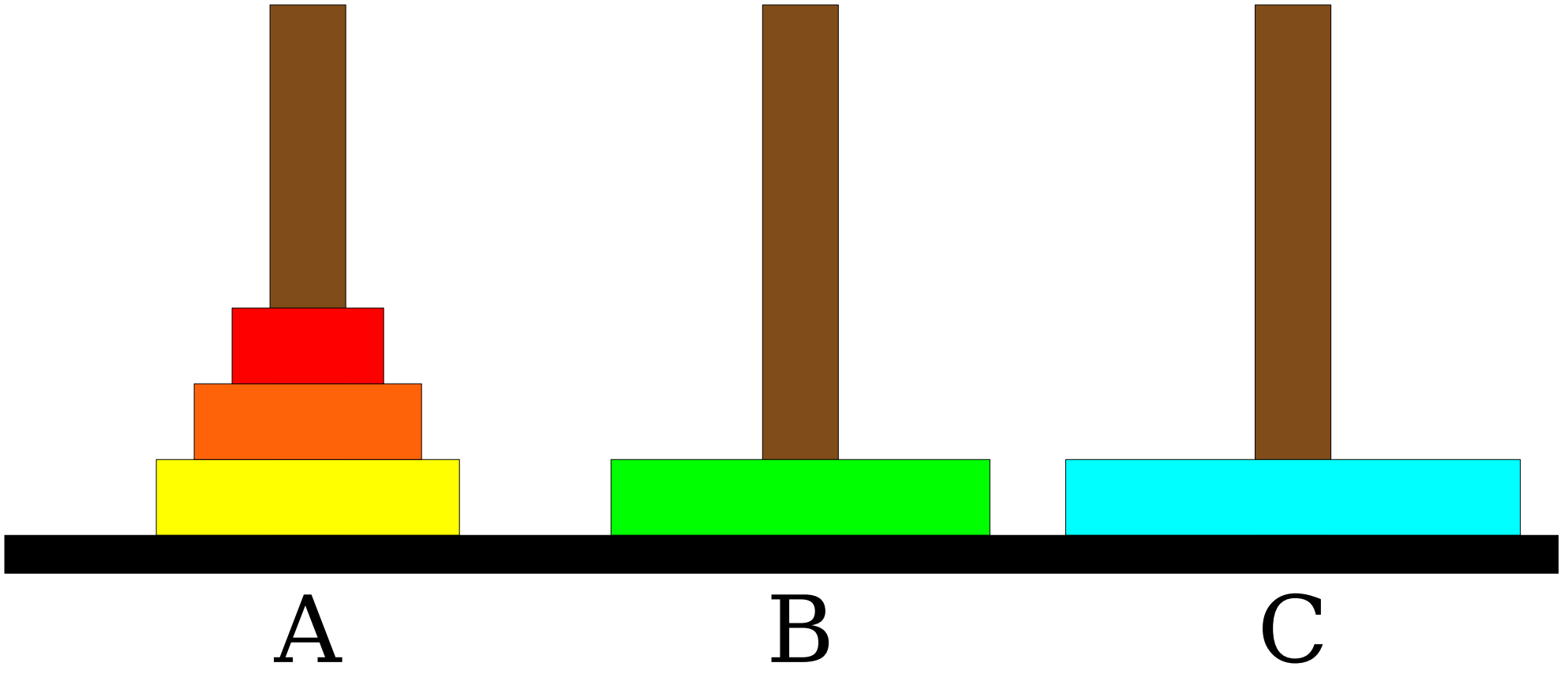
Towers of Hanoi



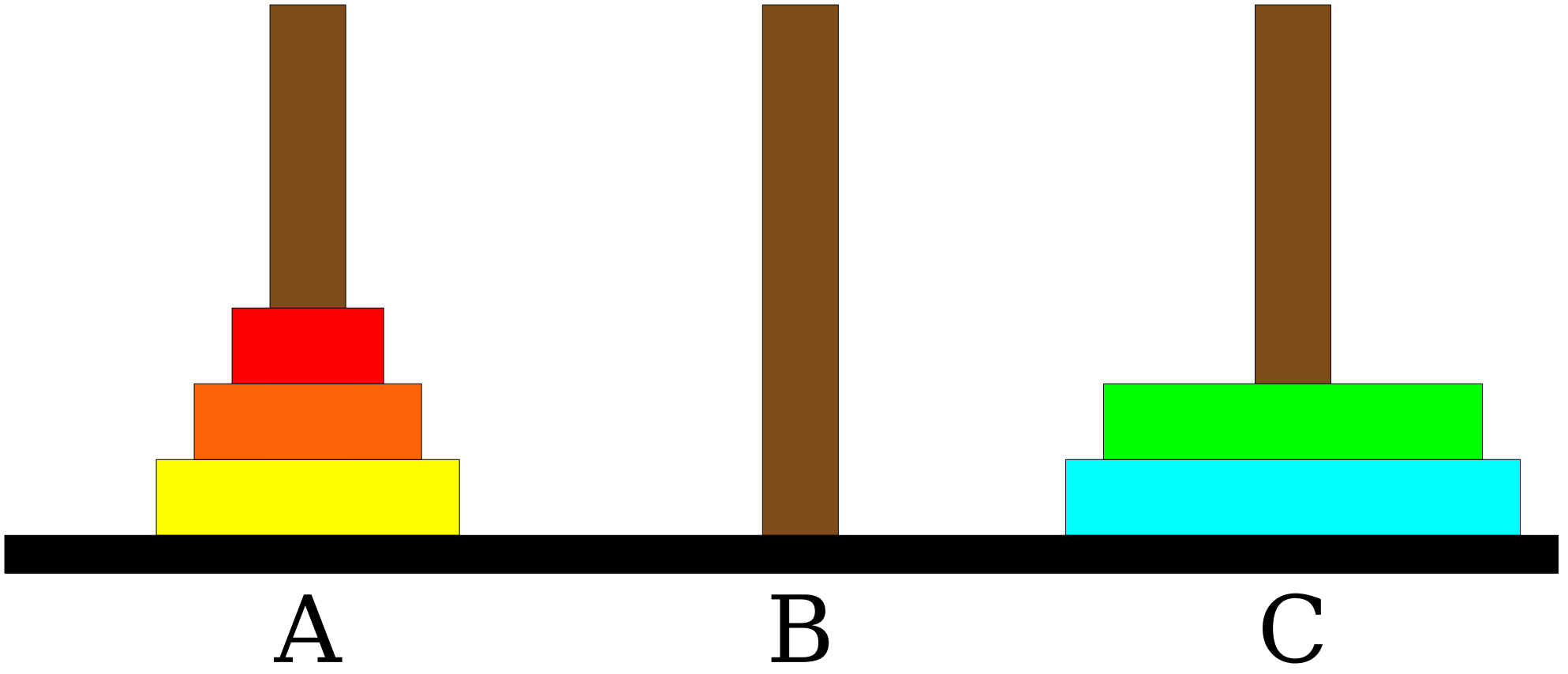
Towers of Hanoi



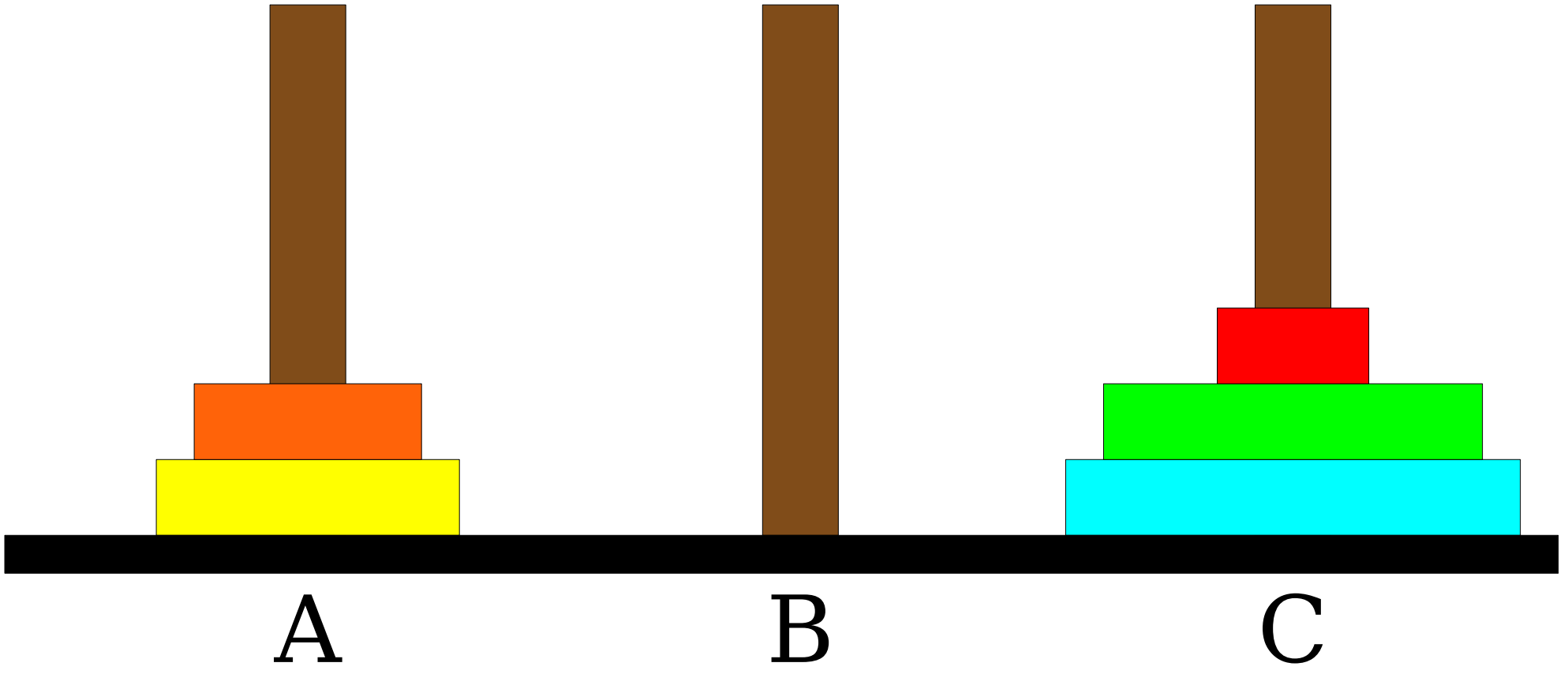
Towers of Hanoi



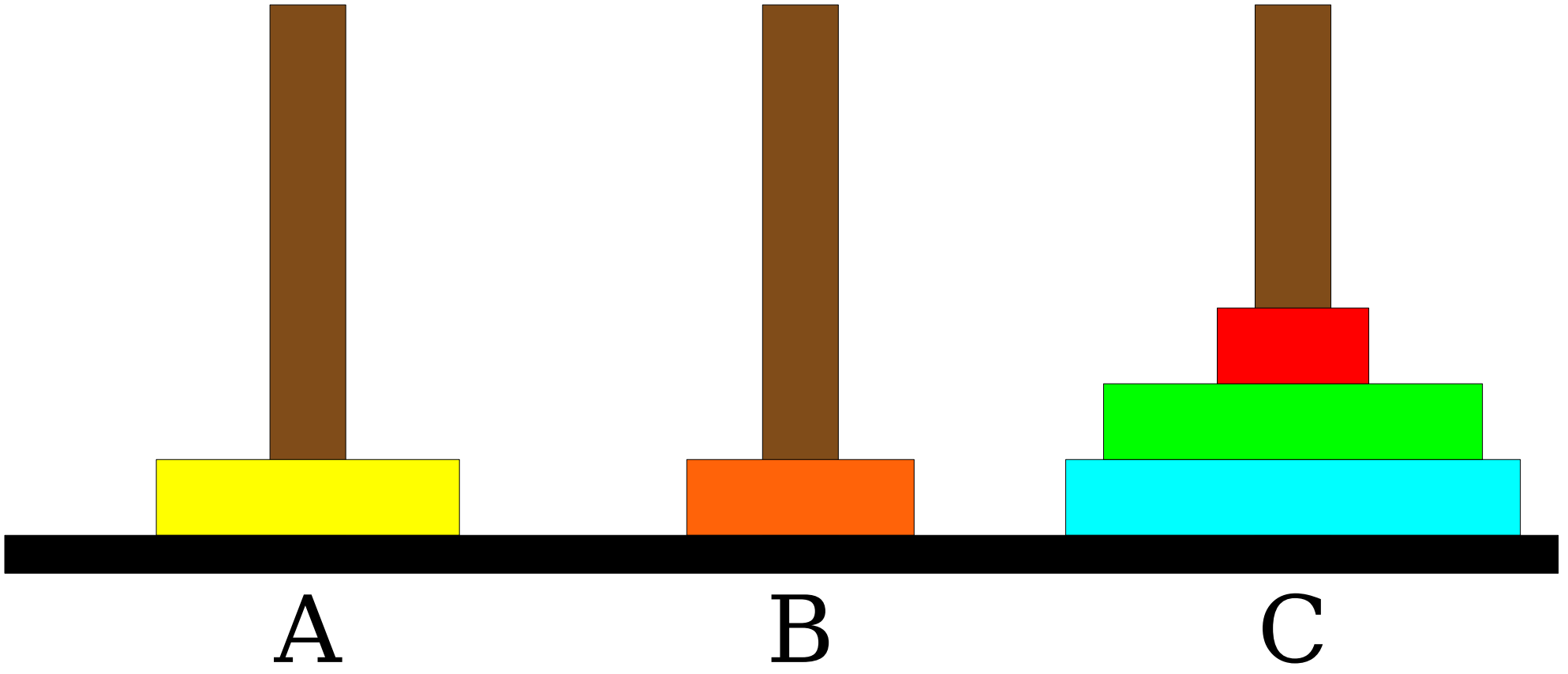
Towers of Hanoi



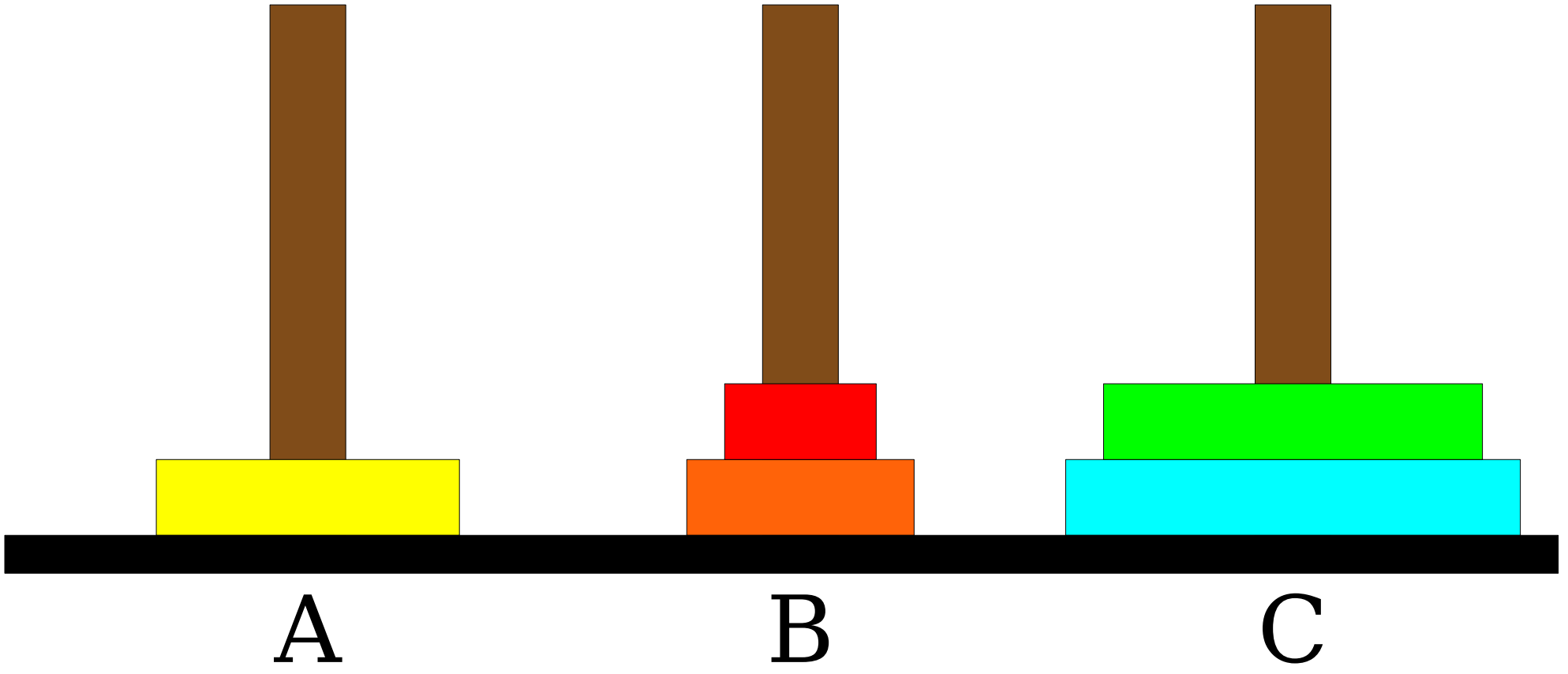
Towers of Hanoi



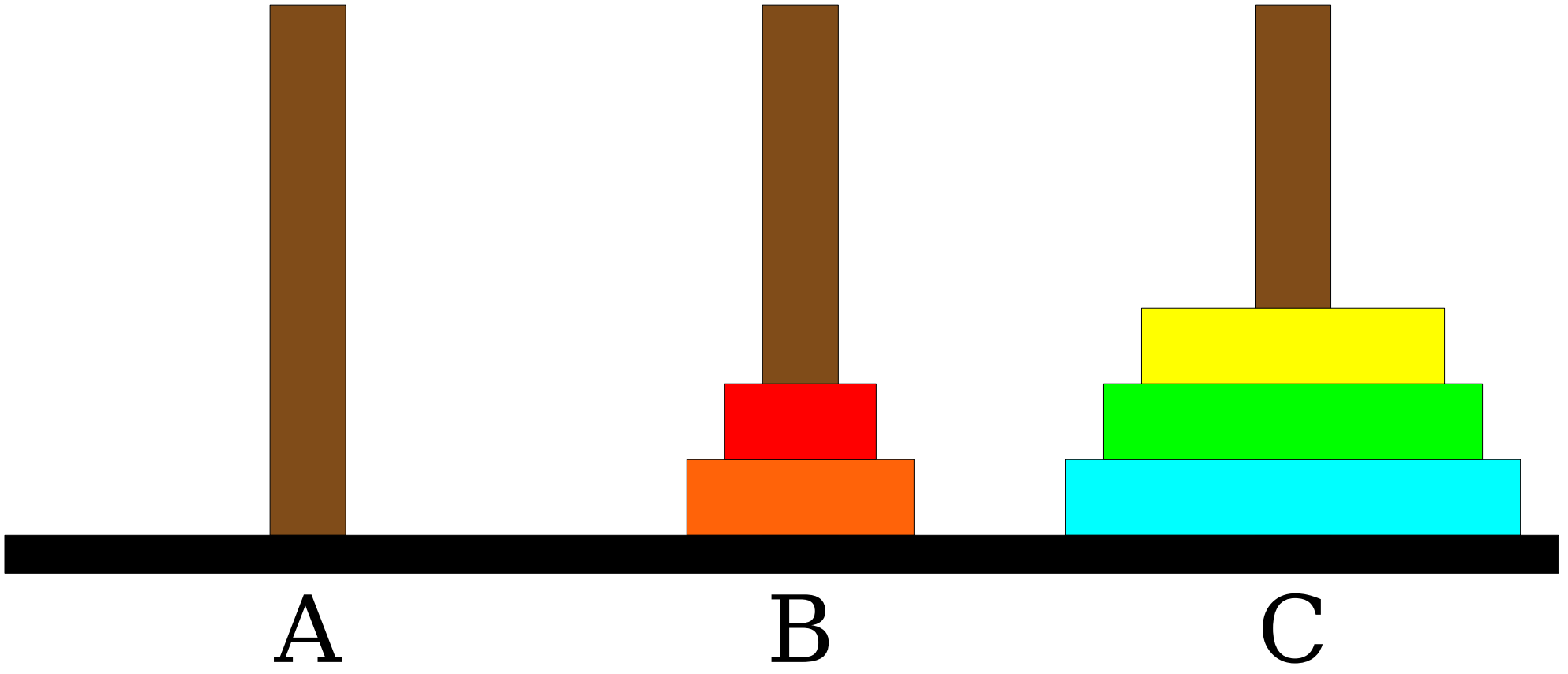
Towers of Hanoi



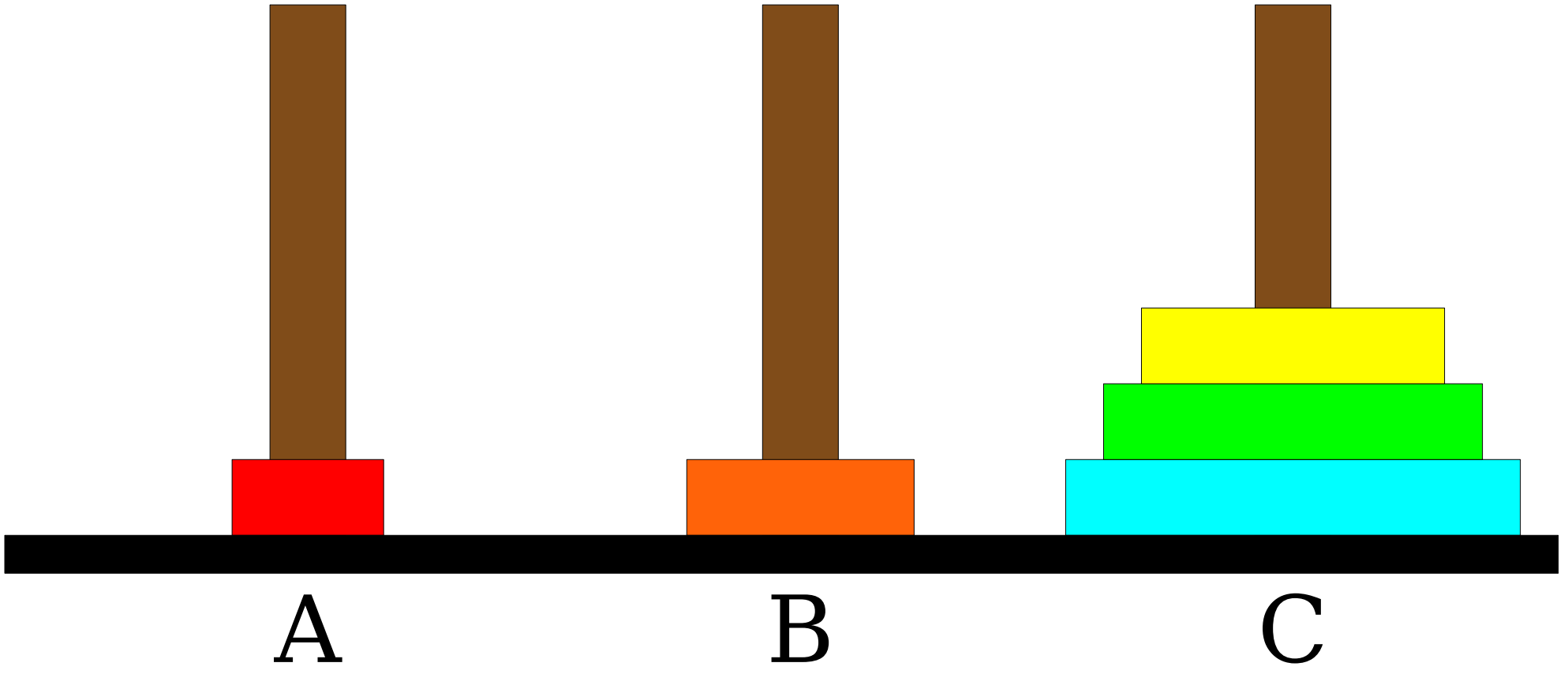
Towers of Hanoi



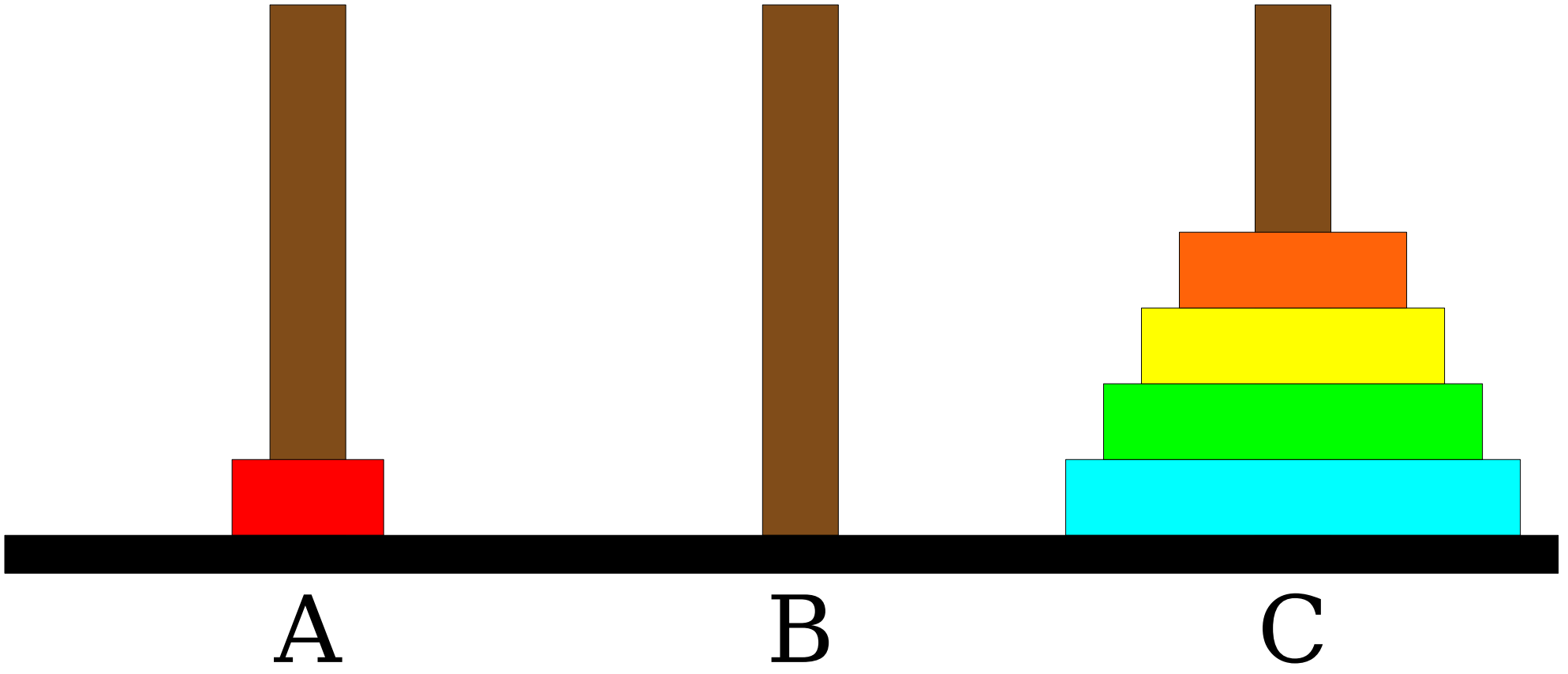
Towers of Hanoi



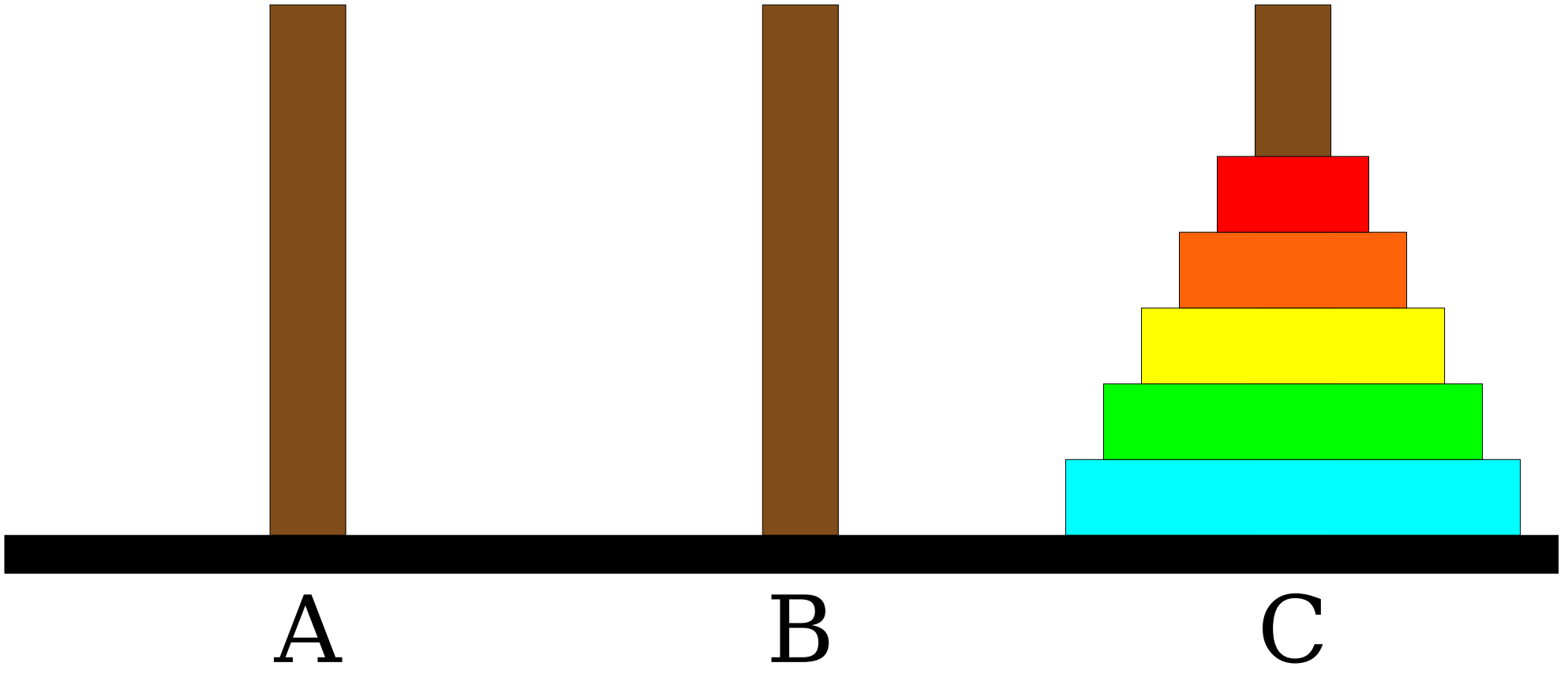
Towers of Hanoi



Towers of Hanoi



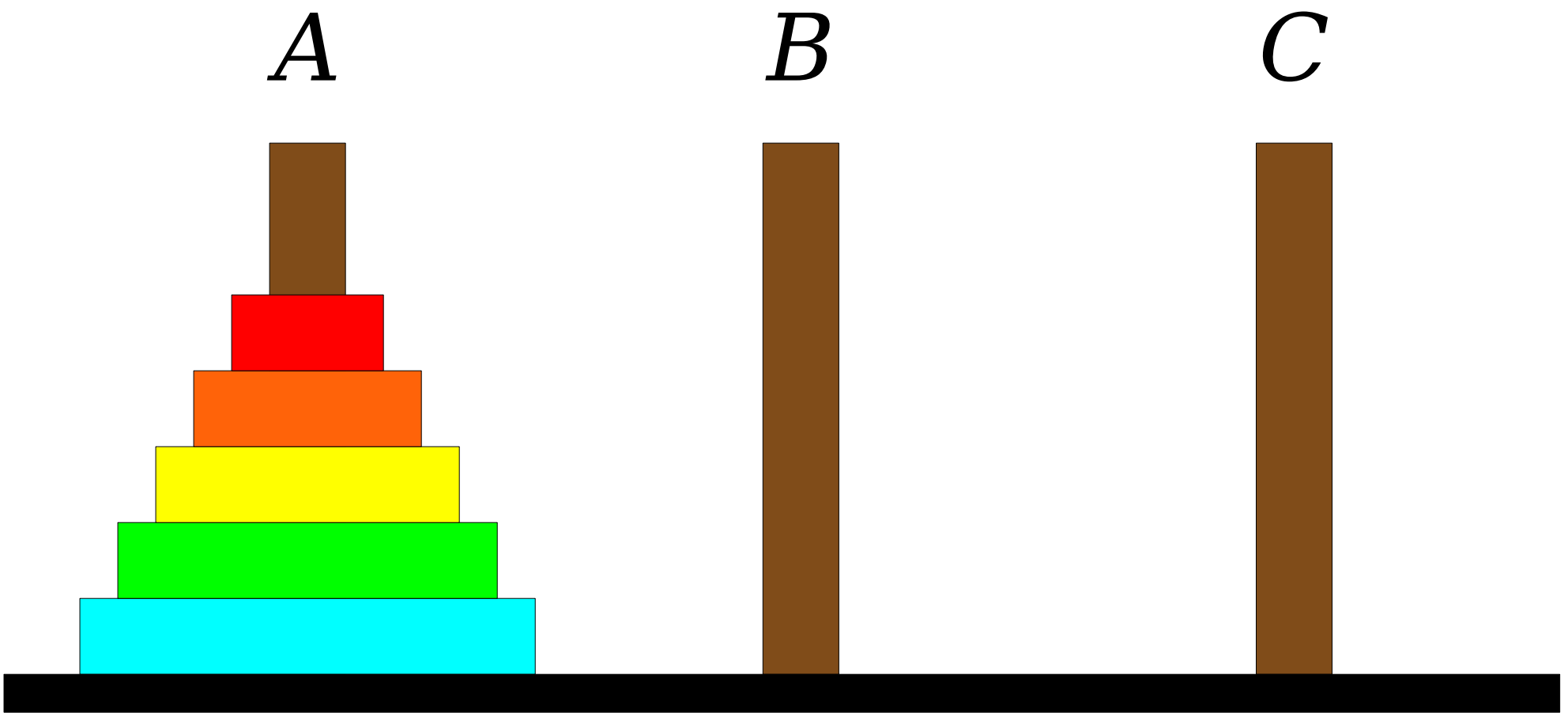
Towers of Hanoi



Solving the Towers of Hanoi

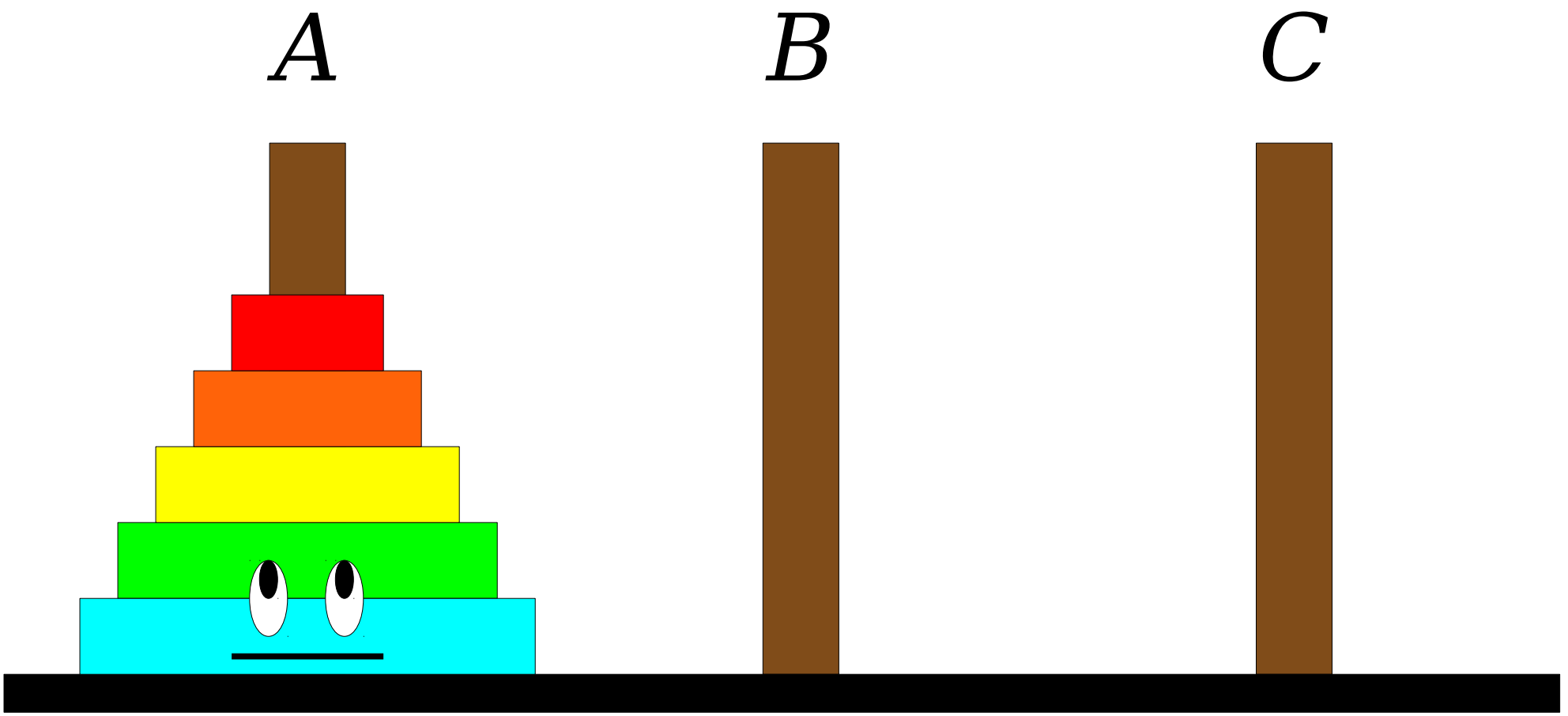
*Thanks to **Grant Sanderson** for the animation idea.*

Solving the Towers of Hanoi



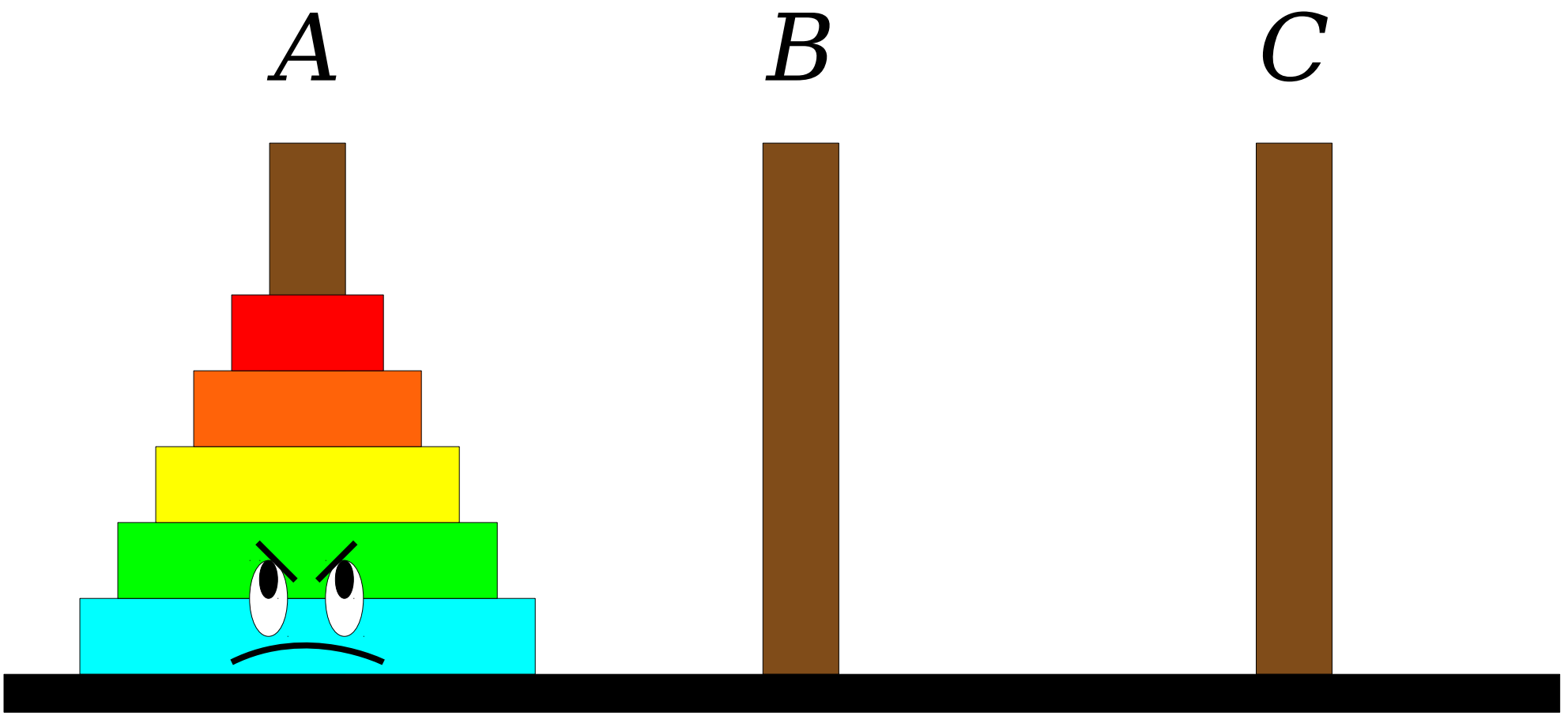
*Thanks to **Grant Sanderson** for the animation idea.*

Solving the Towers of Hanoi



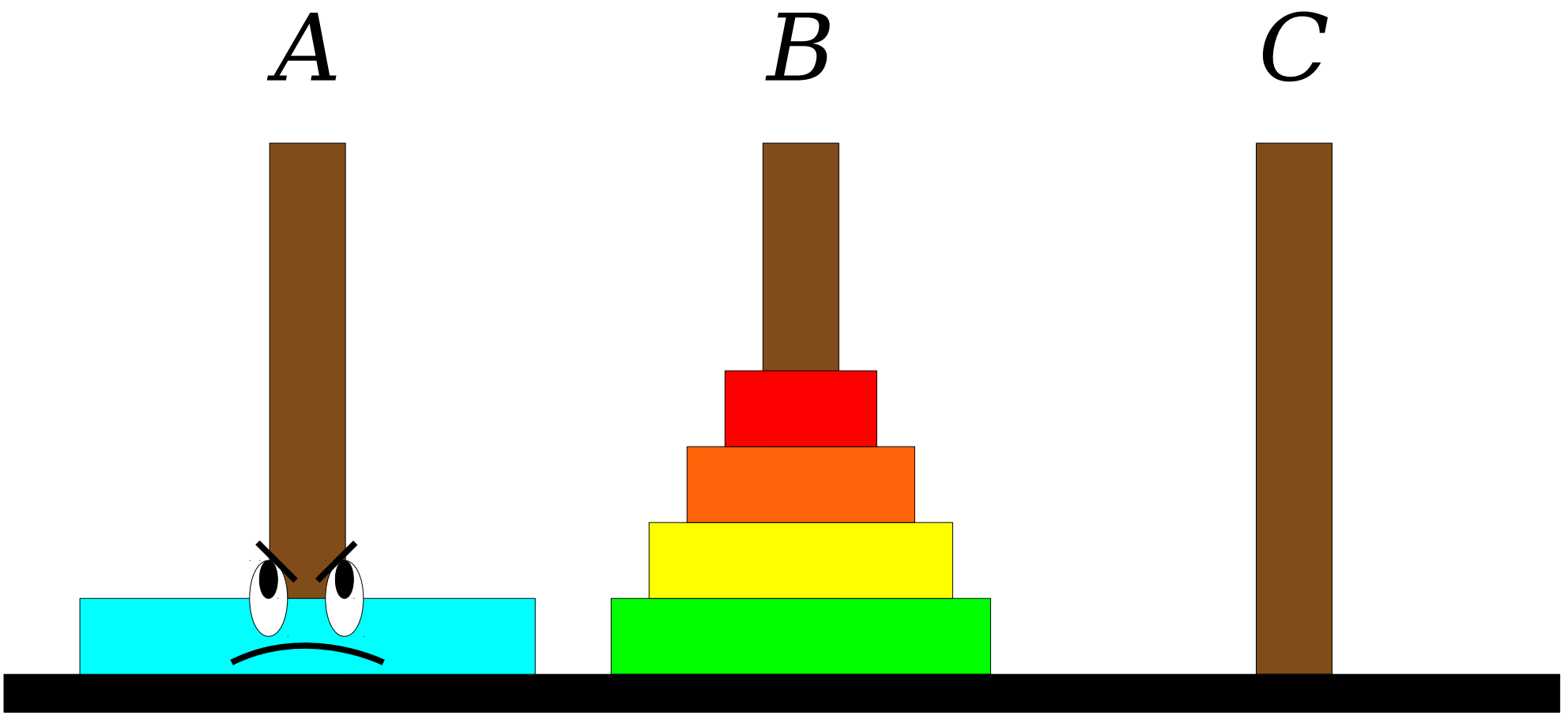
*Thanks to **Grant Sanderson** for the animation idea.*

Solving the Towers of Hanoi



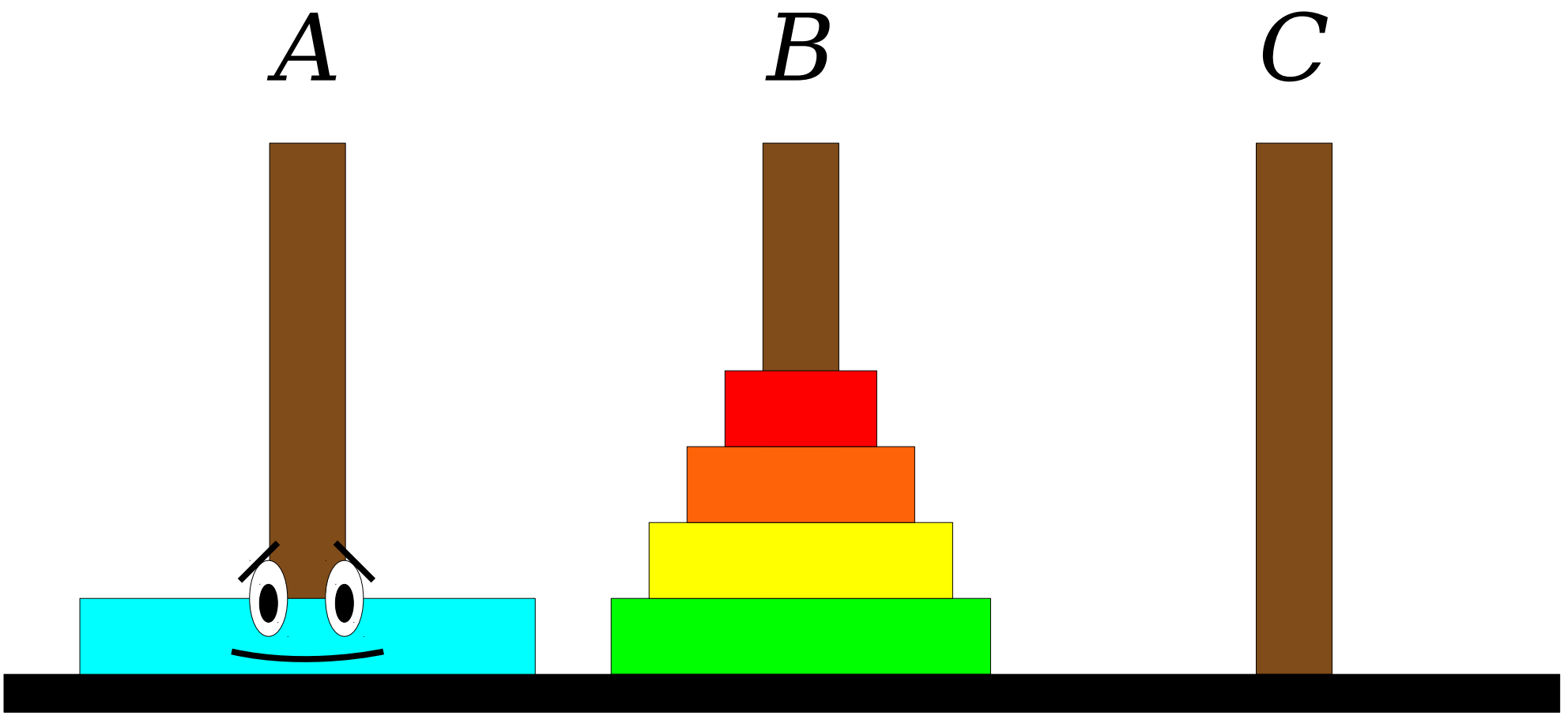
*Thanks to **Grant Sanderson** for the animation idea.*

Solving the Towers of Hanoi



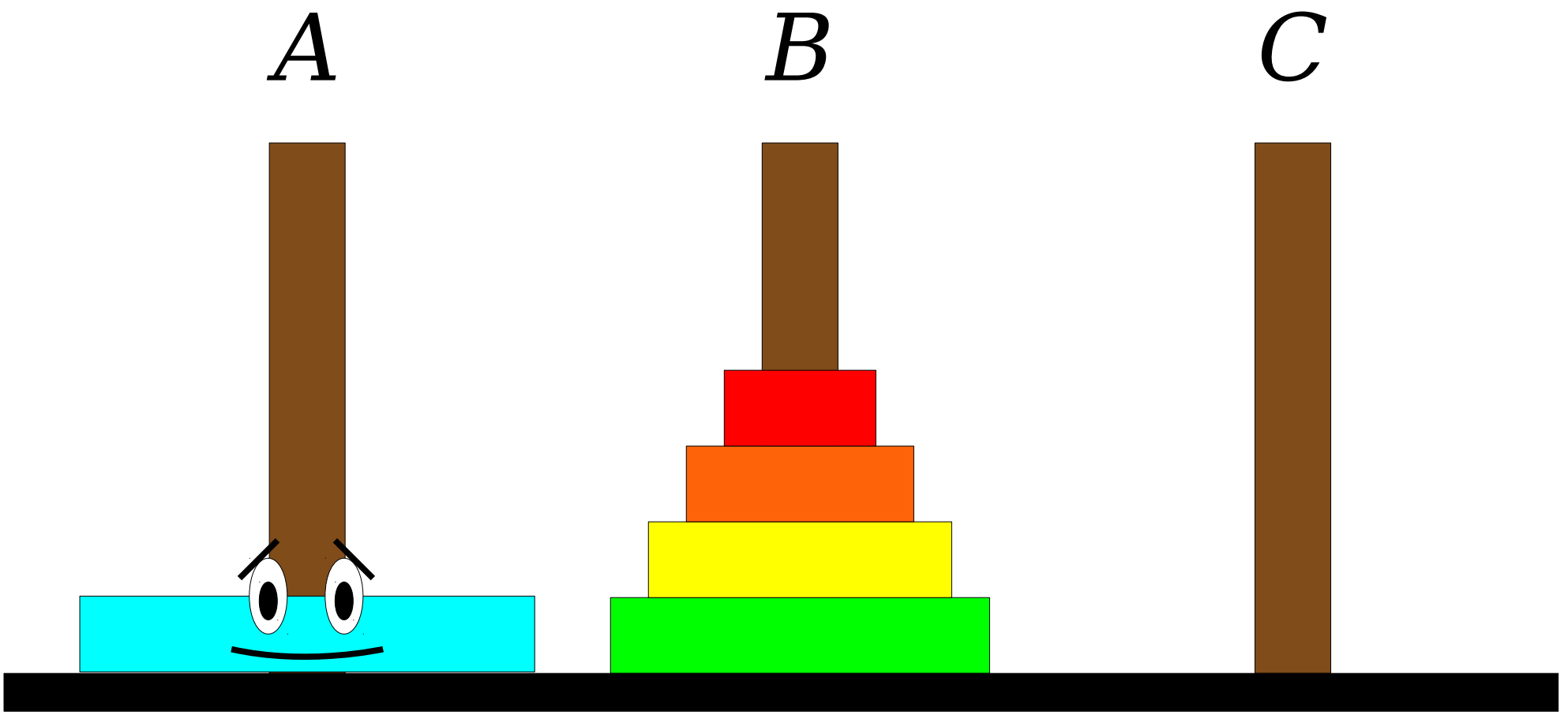
*Thanks to **Grant Sanderson** for the animation idea.*

Solving the Towers of Hanoi



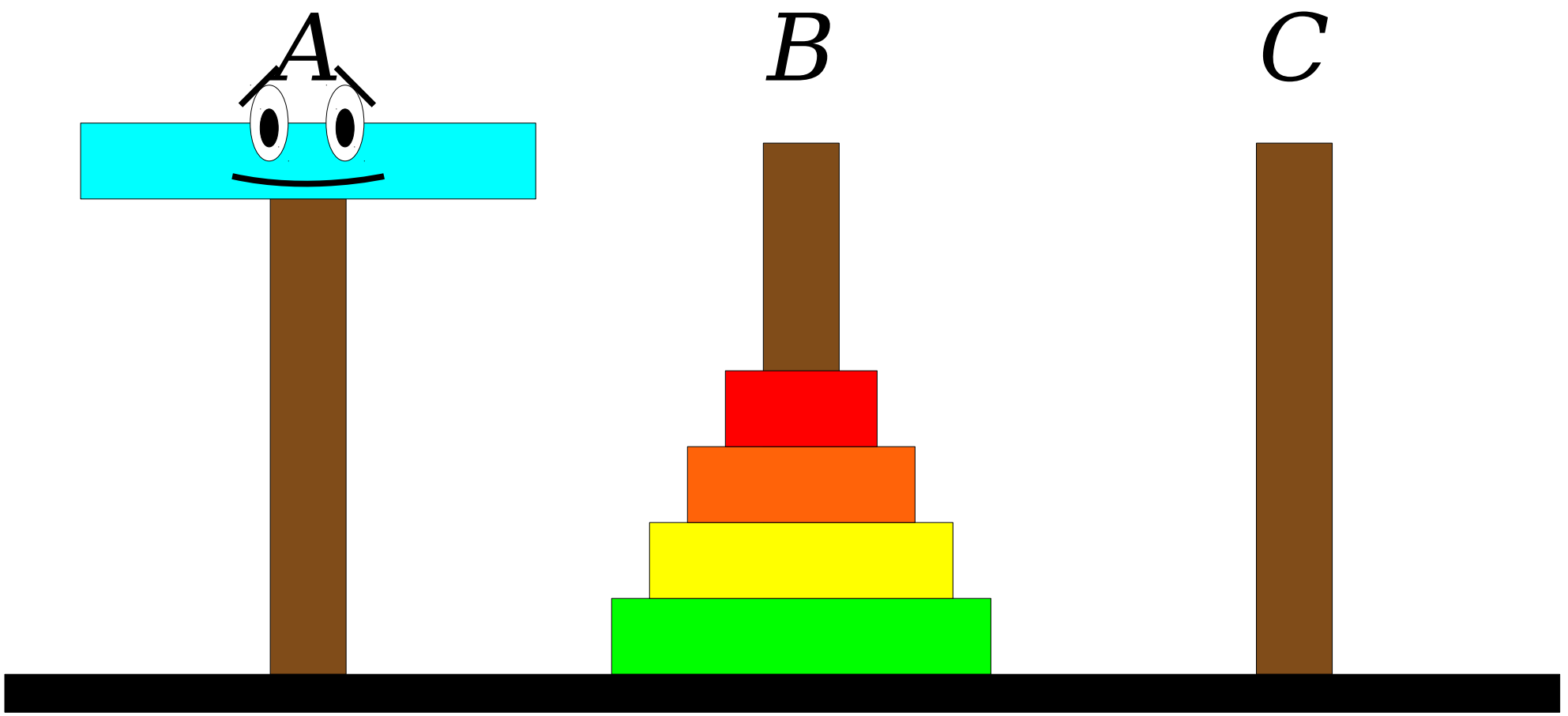
*Thanks to **Grant Sanderson** for the animation idea.*

Solving the Towers of Hanoi



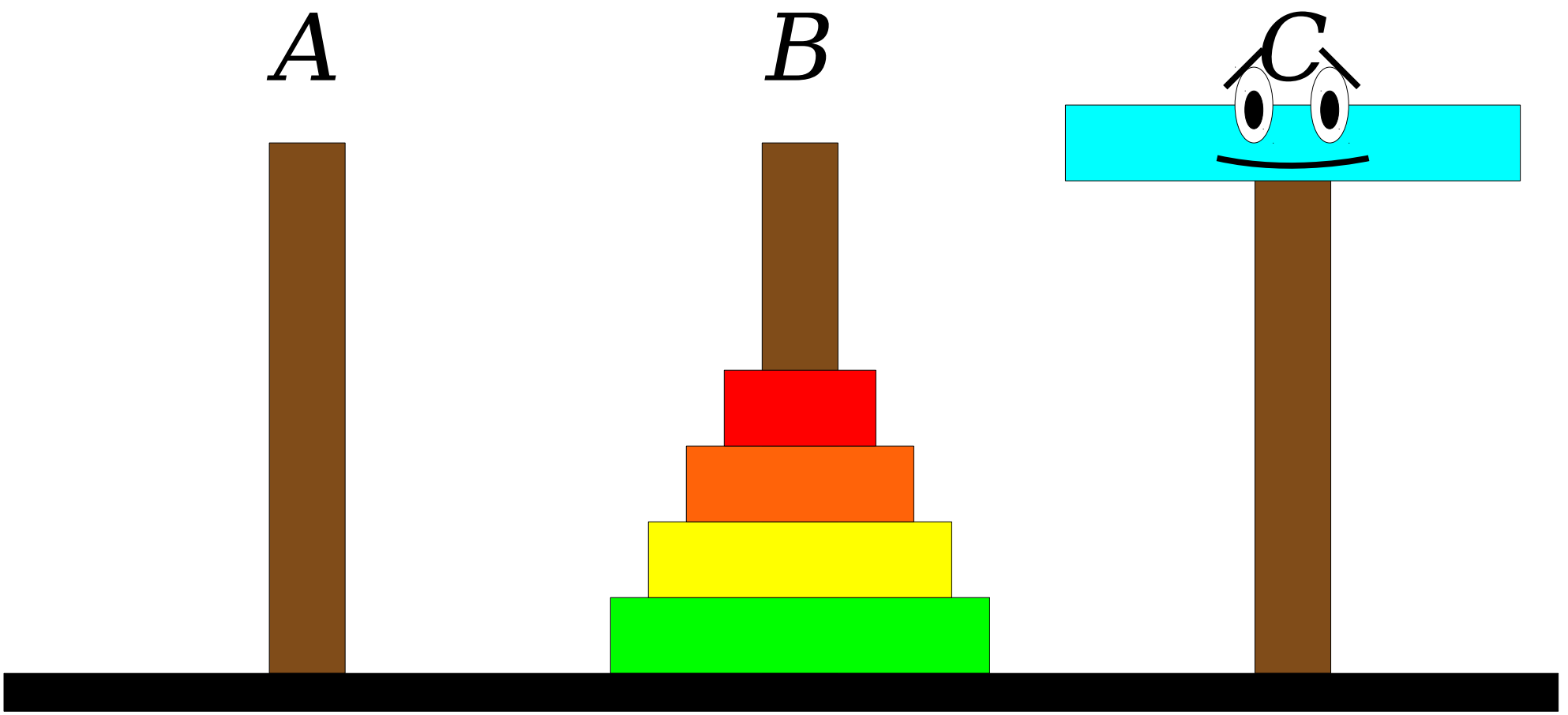
Step One: Move the four smaller disks from Spindle *A* to Spindle *B*.

Solving the Towers of Hanoi



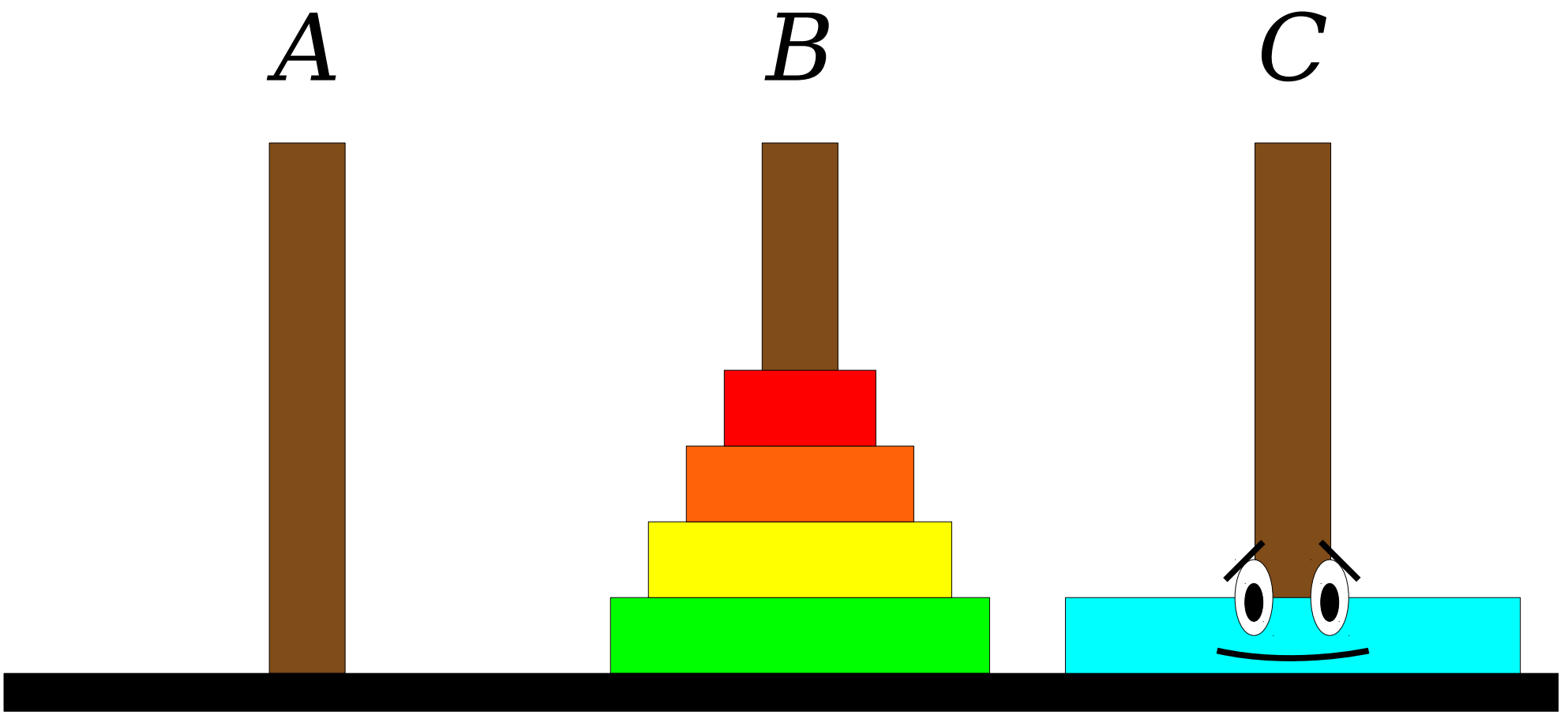
Step One: Move the four smaller disks from Spindle *A* to Spindle *B*.

Solving the Towers of Hanoi



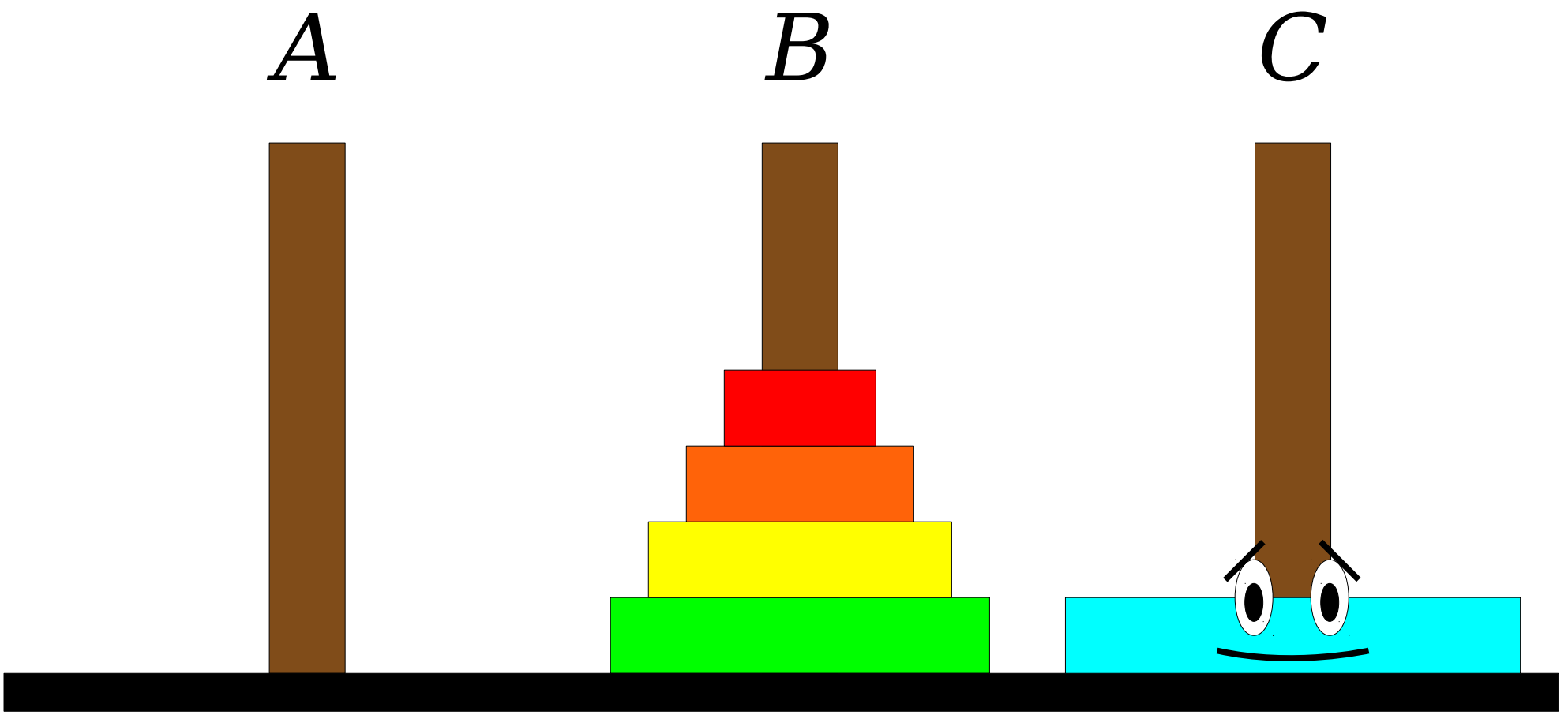
Step One: Move the four smaller disks from Spindle *A* to Spindle *B*.

Solving the Towers of Hanoi



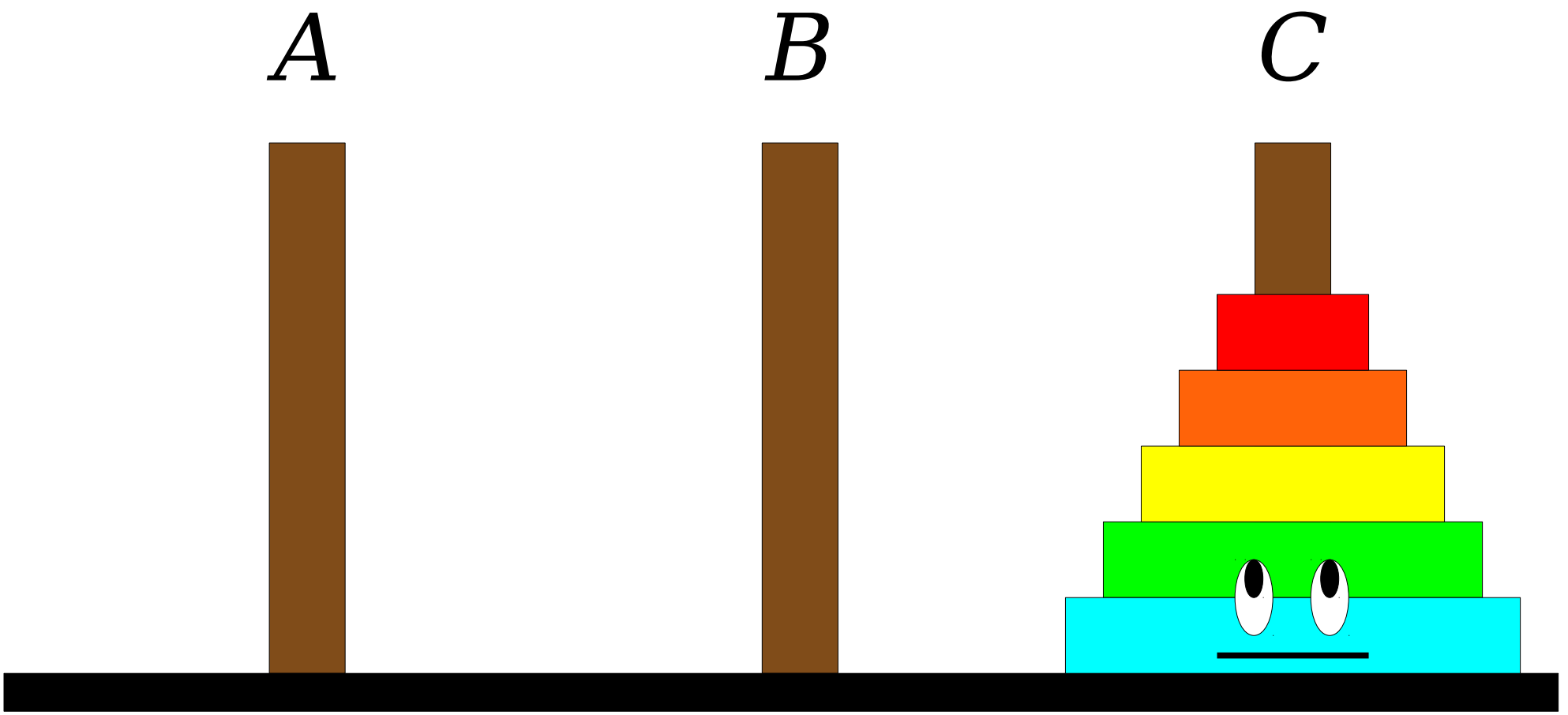
Step One: Move the four smaller disks from Spindle *A* to Spindle *B*.

Solving the Towers of Hanoi



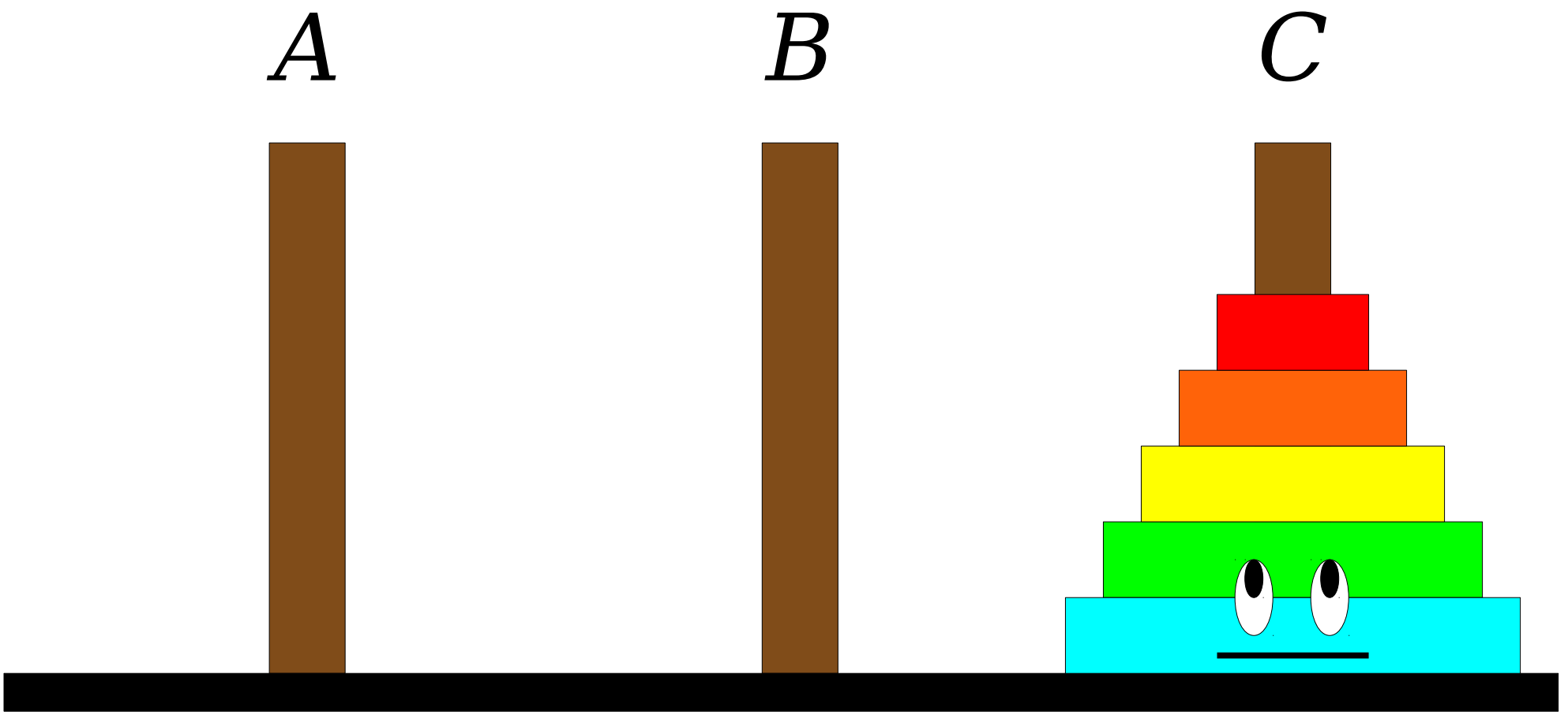
Step One: Move the four smaller disks from Spindle *A* to Spindle *B*.
Step Two: Move the blue disk from Spindle *A* to Spindle *C*.

Solving the Towers of Hanoi



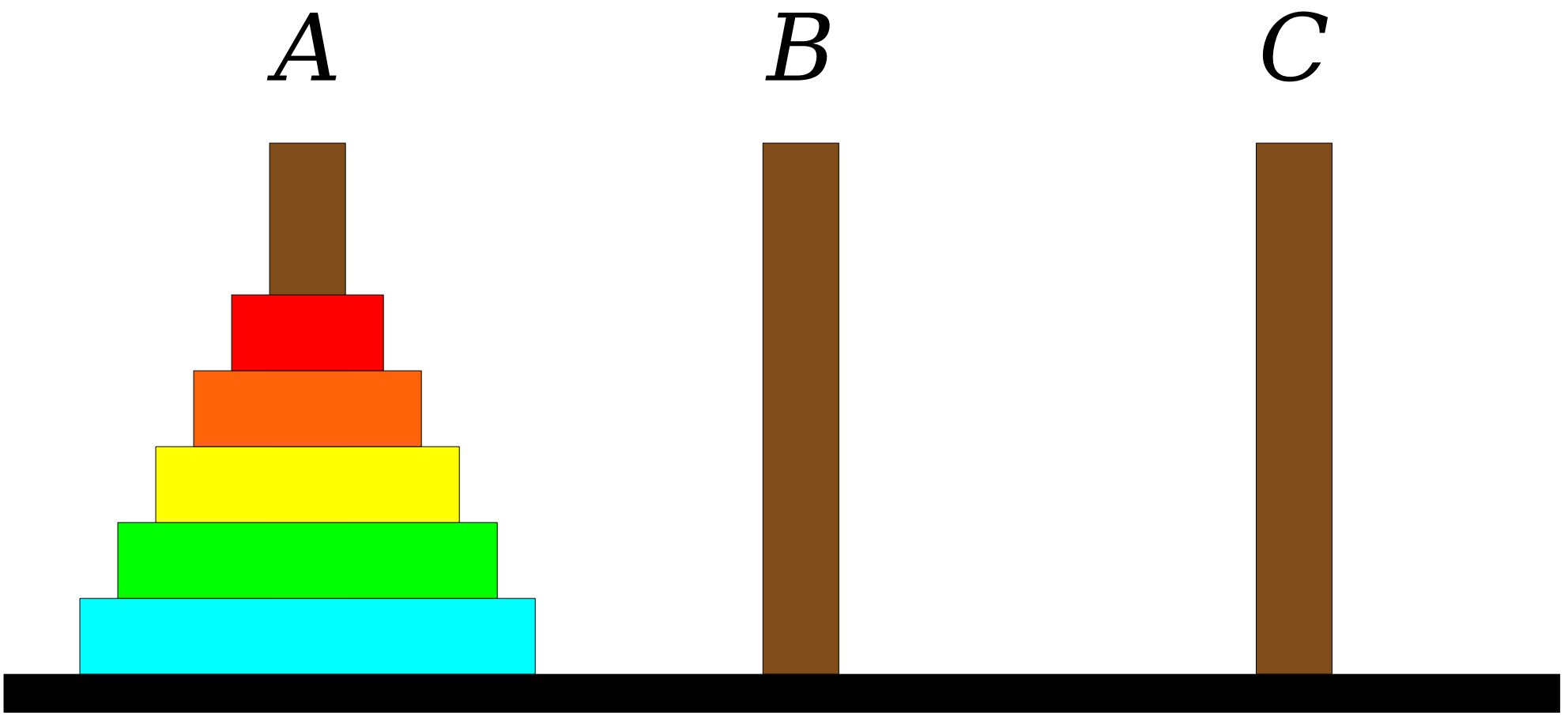
Step One: Move the four smaller disks from Spindle *A* to Spindle *B*.
Step Two: Move the blue disk from Spindle *A* to Spindle *C*.

Solving the Towers of Hanoi

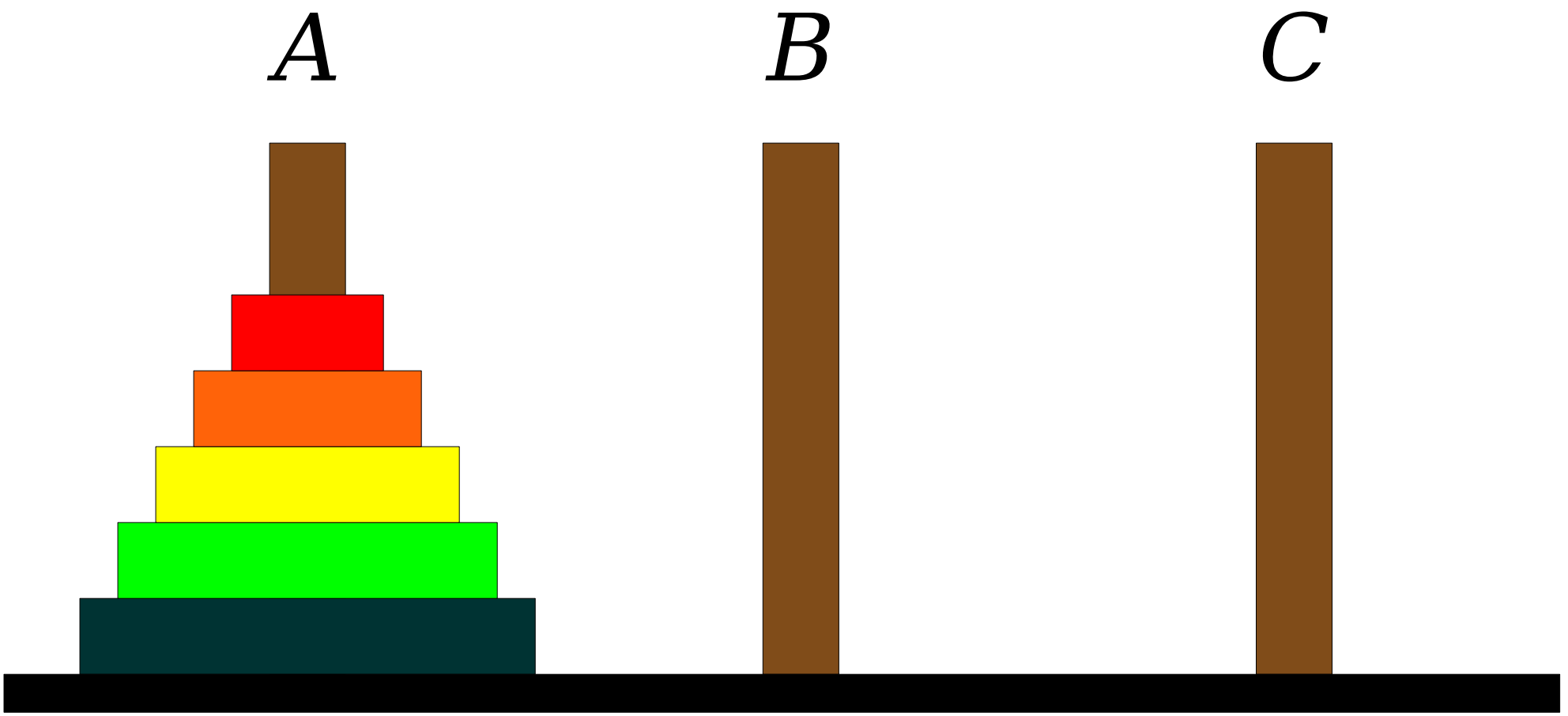


Step One: Move the four smaller disks from Spindle *A* to Spindle *B*.
Step Two: Move the blue disk from Spindle *A* to Spindle *C*.
Step Three: Move the four smaller disks from Spindle *B* to Spindle *C*.

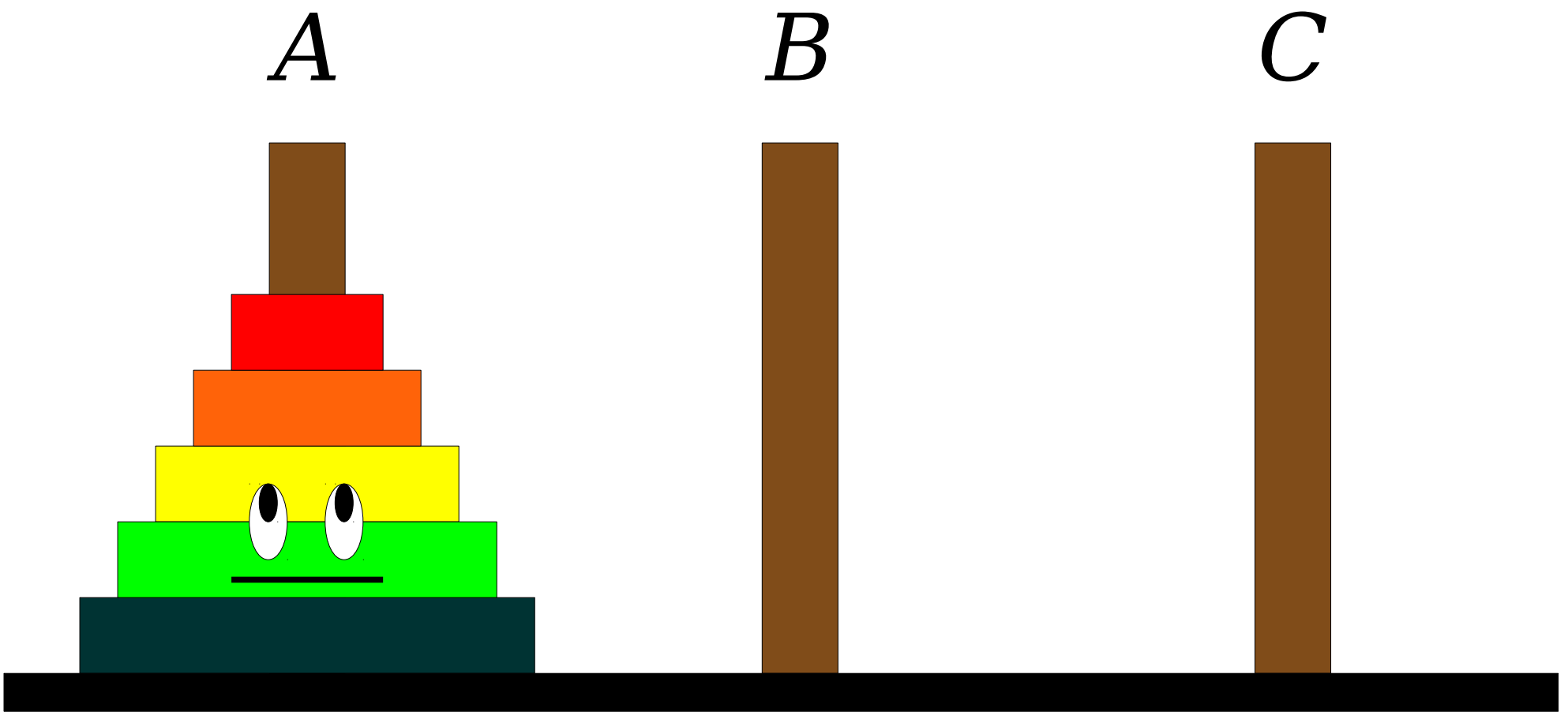
Solving the Towers of Hanoi



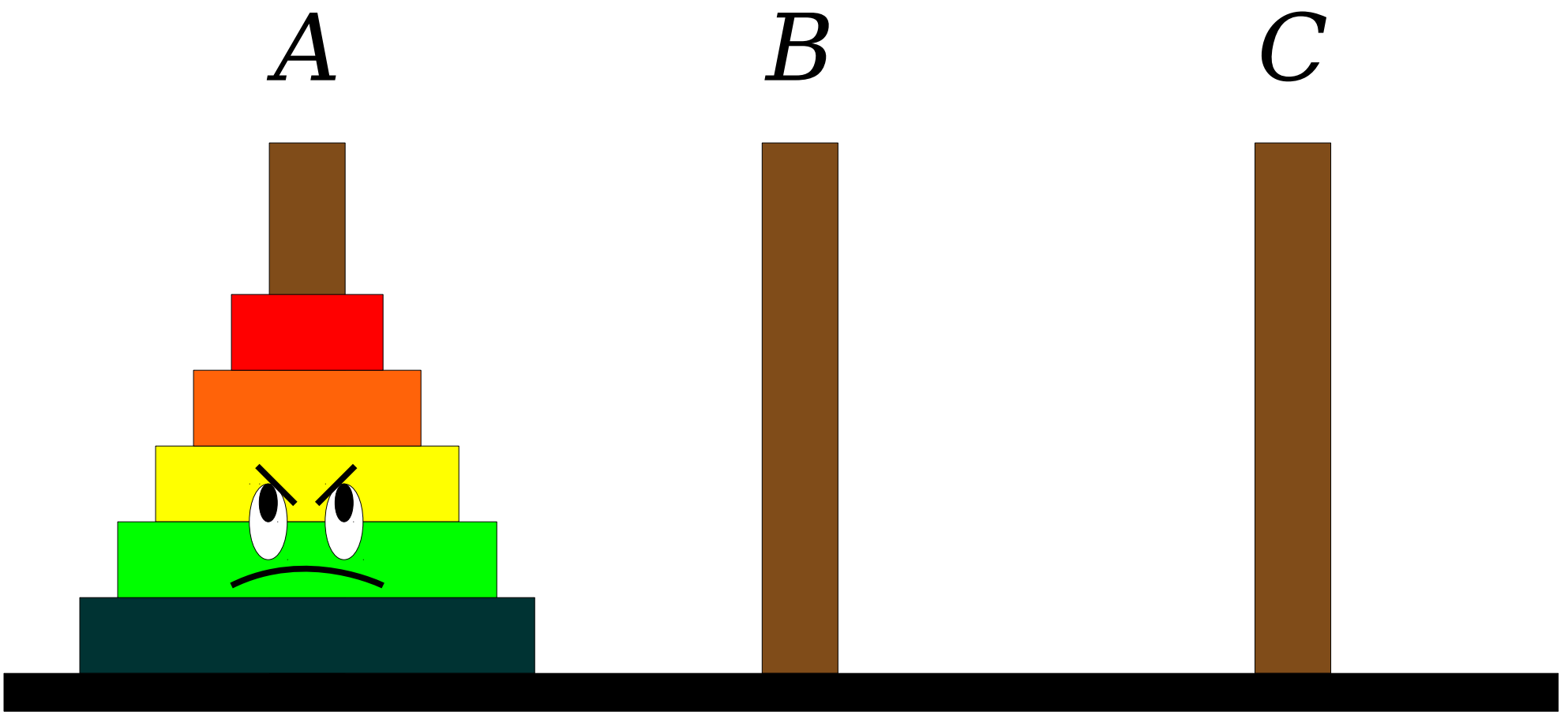
Solving the Towers of Hanoi



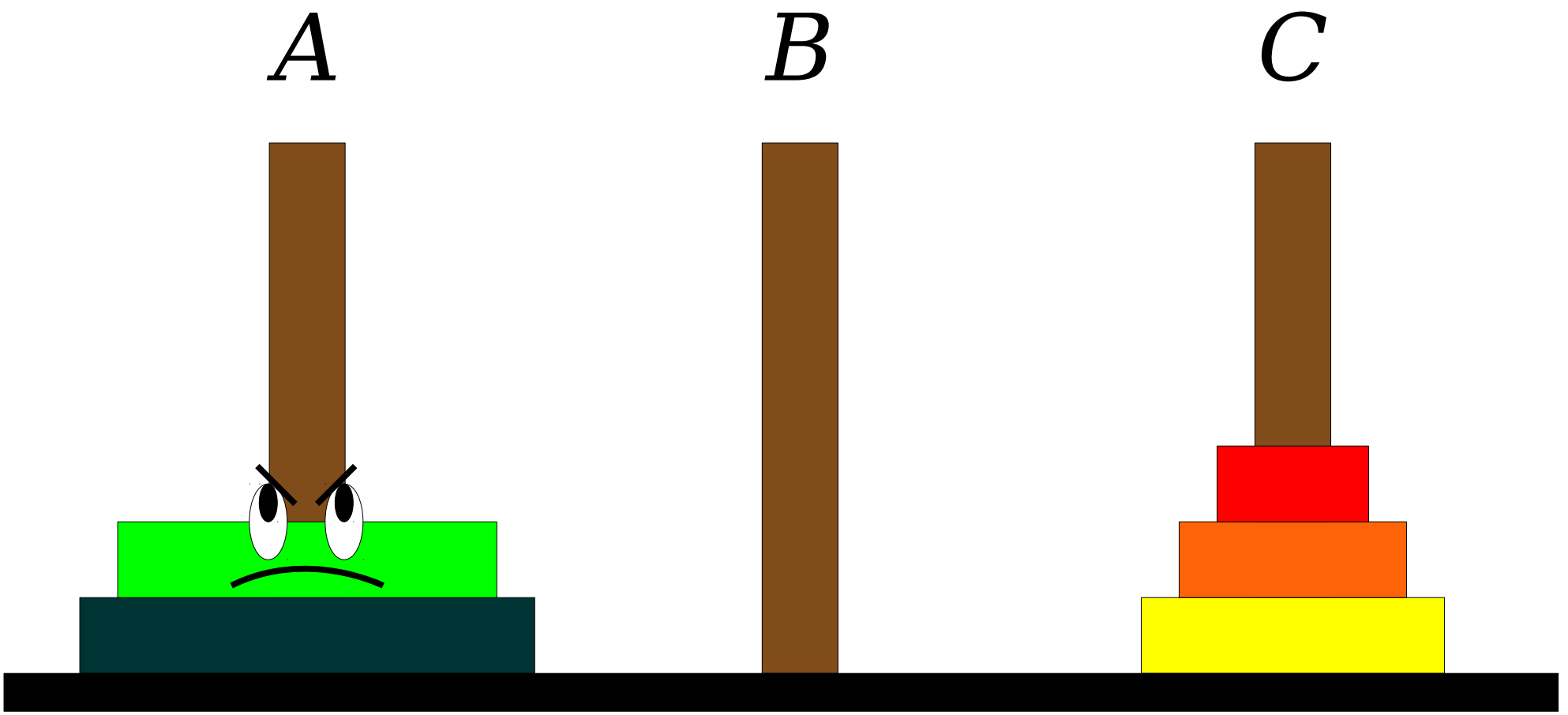
Solving the Towers of Hanoi



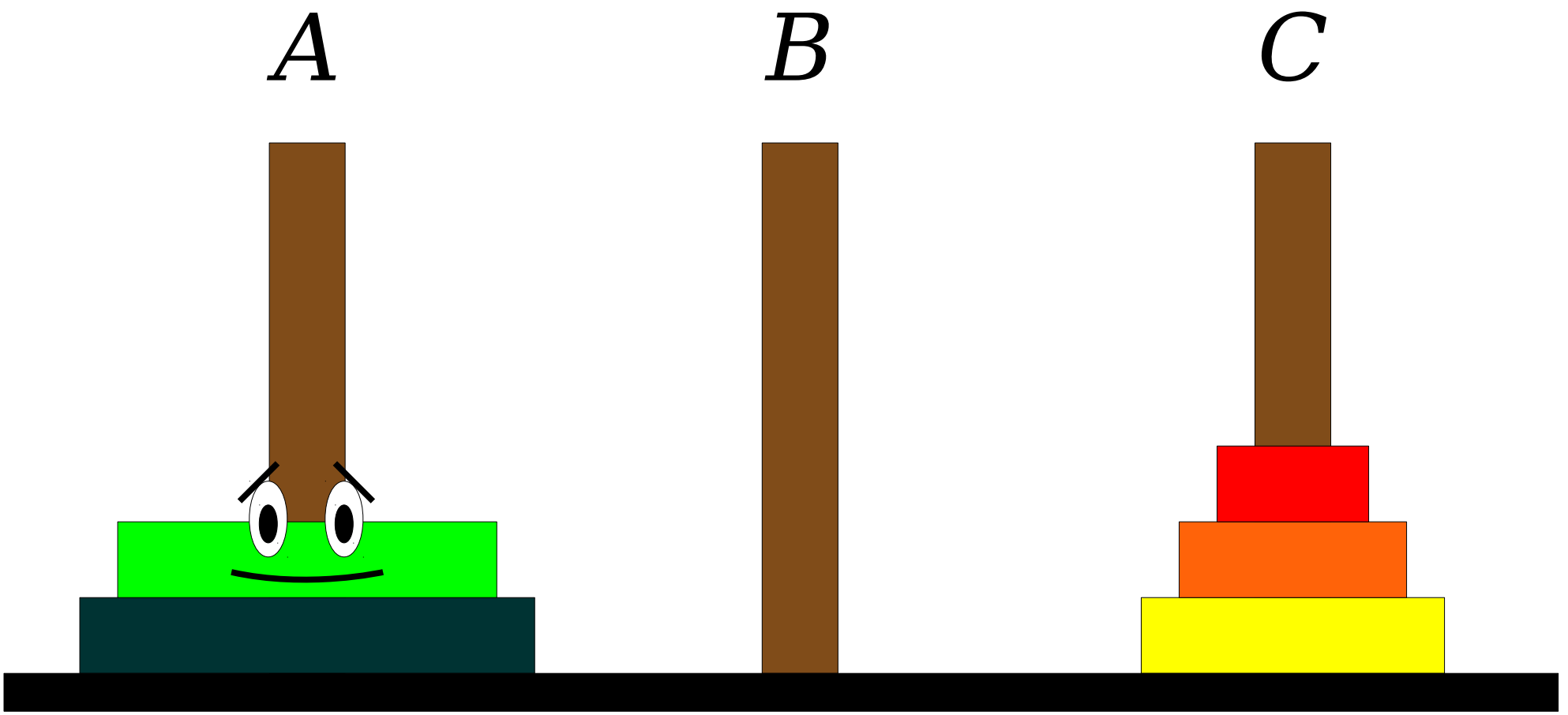
Solving the Towers of Hanoi



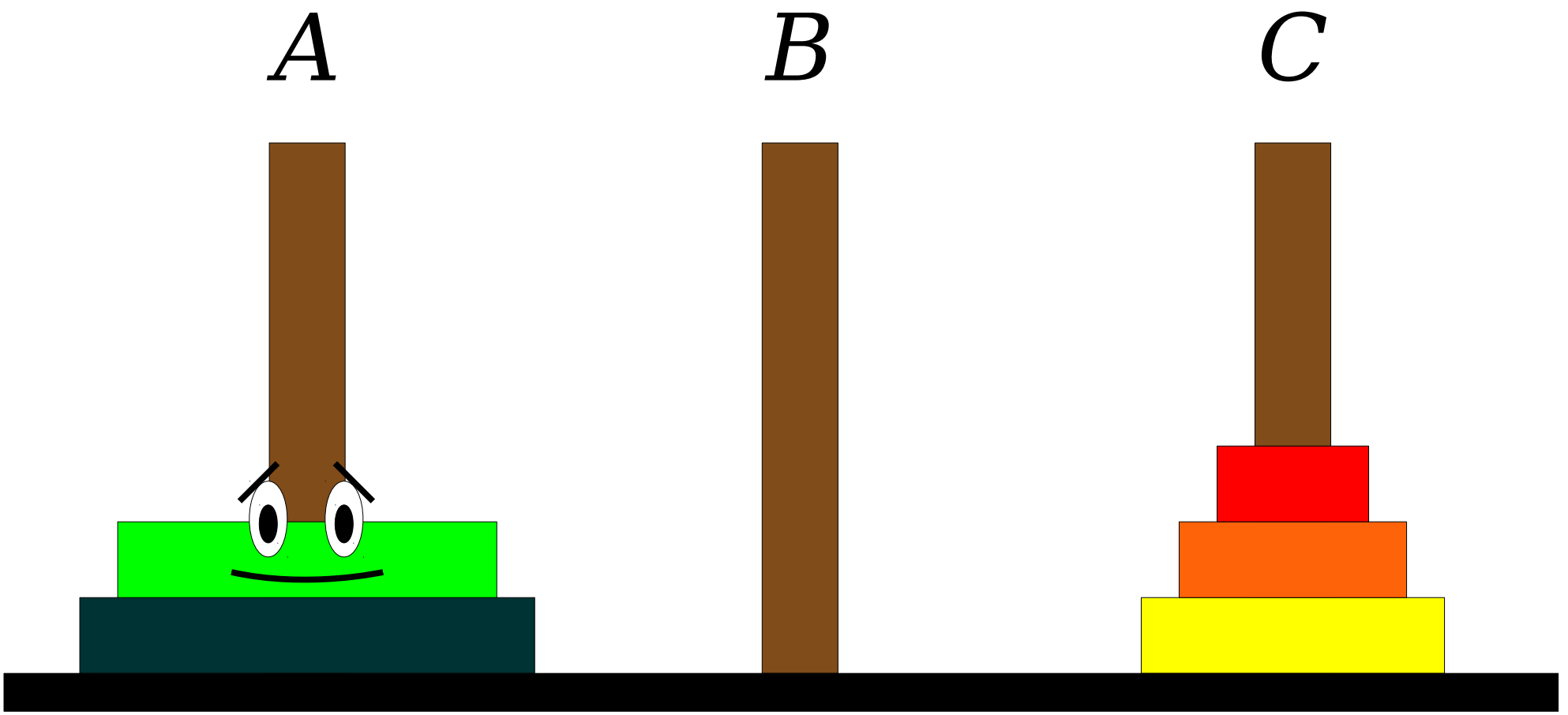
Solving the Towers of Hanoi



Solving the Towers of Hanoi

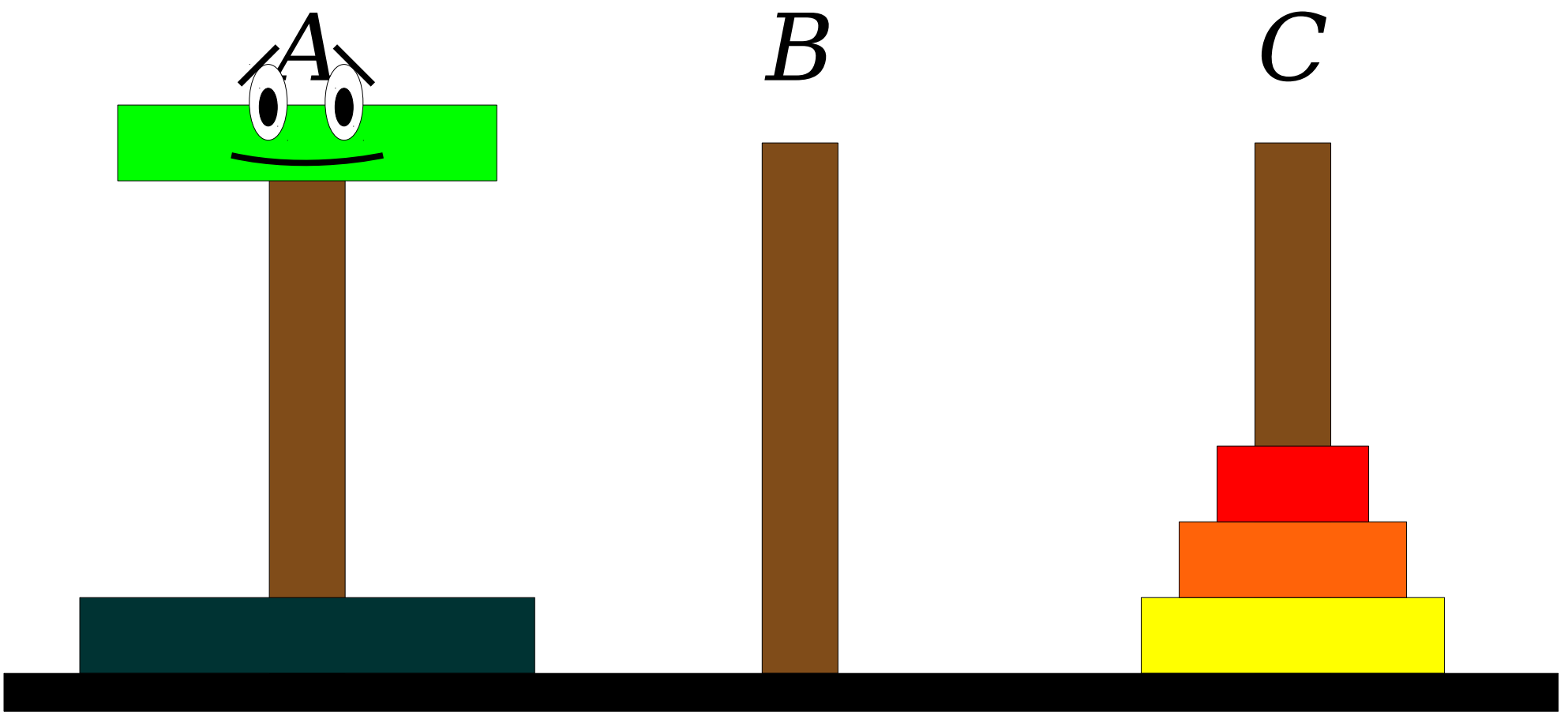


Solving the Towers of Hanoi



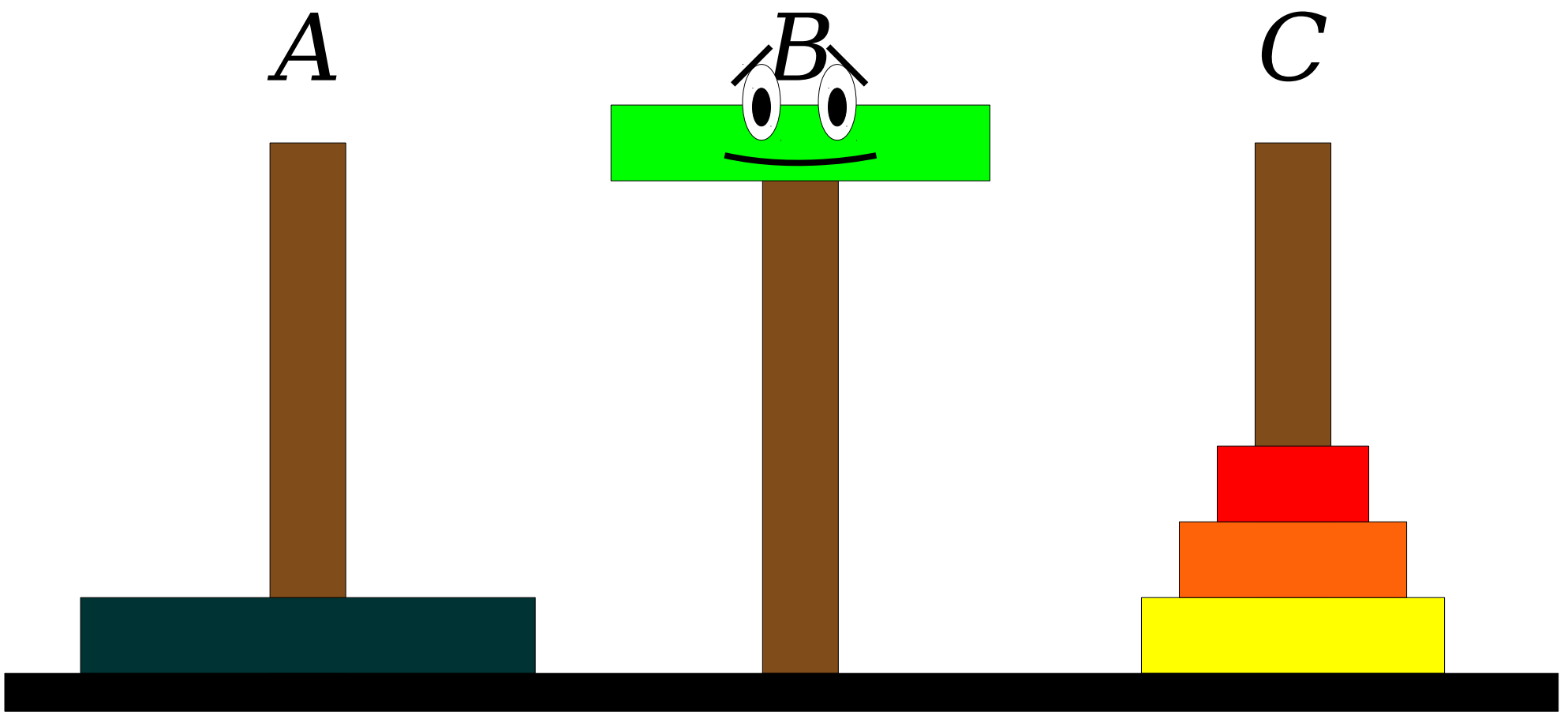
Step One: Move the three smaller disks from Spindle *A* to Spindle *C*.

Solving the Towers of Hanoi



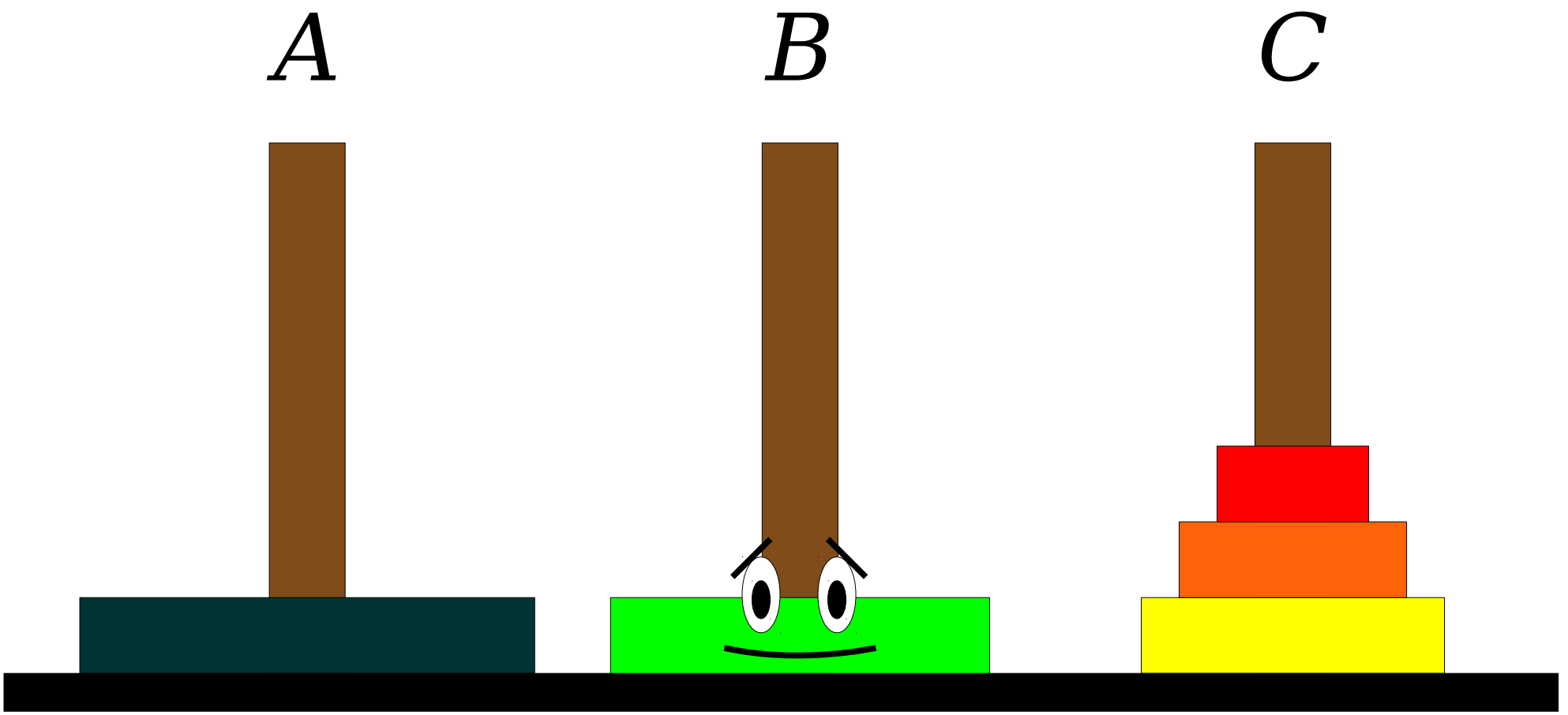
Step One: Move the three smaller disks from Spindle *A* to Spindle *C*.

Solving the Towers of Hanoi



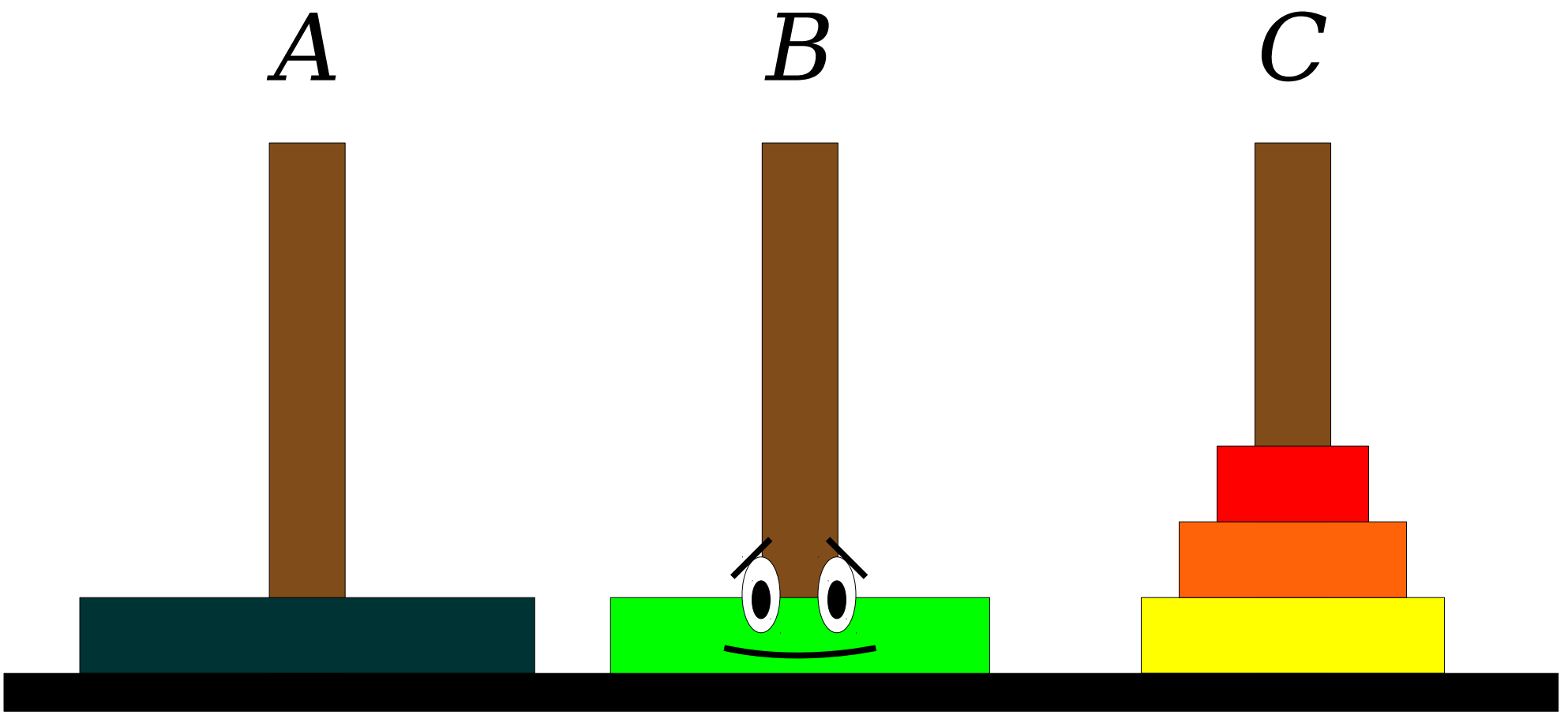
Step One: Move the three smaller disks from Spindle *A* to Spindle *C*.

Solving the Towers of Hanoi



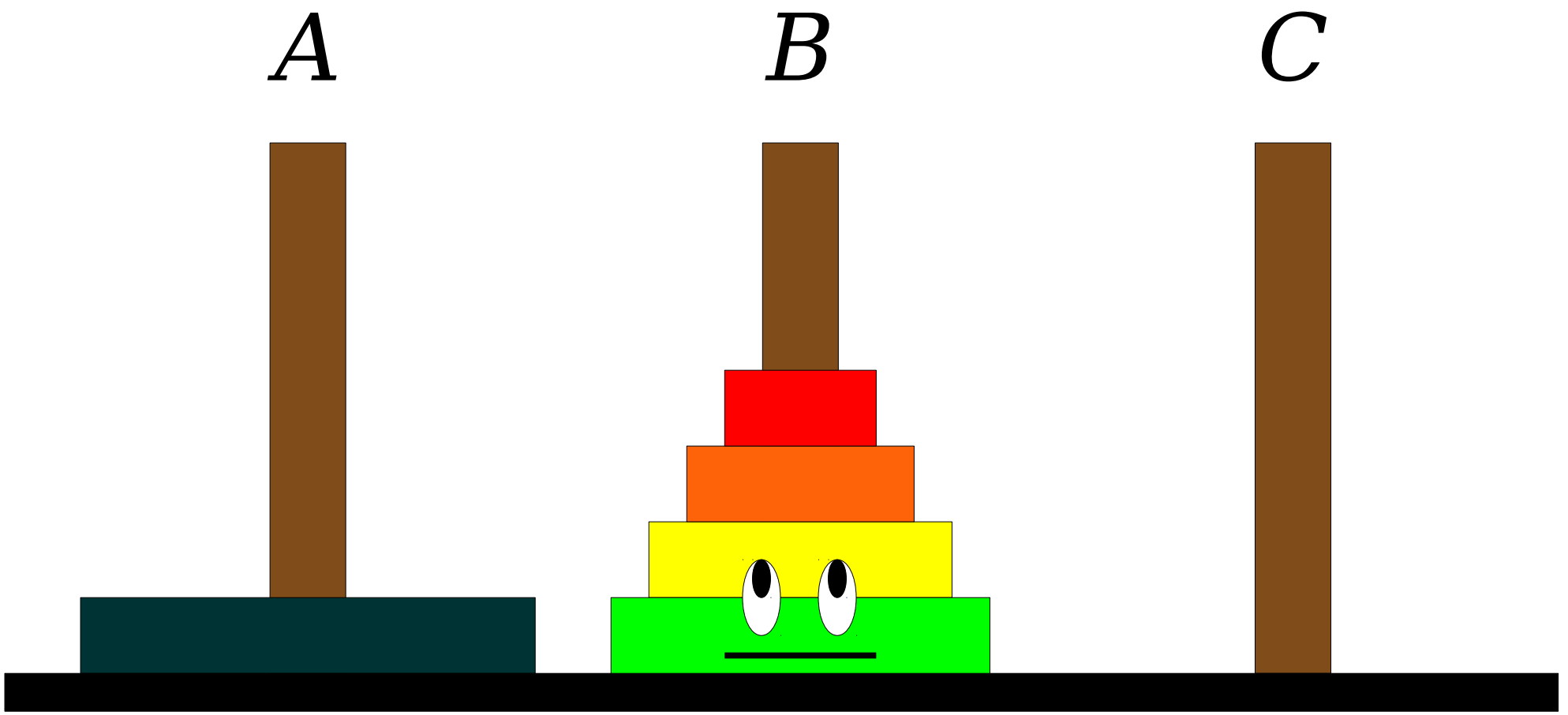
Step One: Move the three smaller disks from Spindle *A* to Spindle *C*.

Solving the Towers of Hanoi



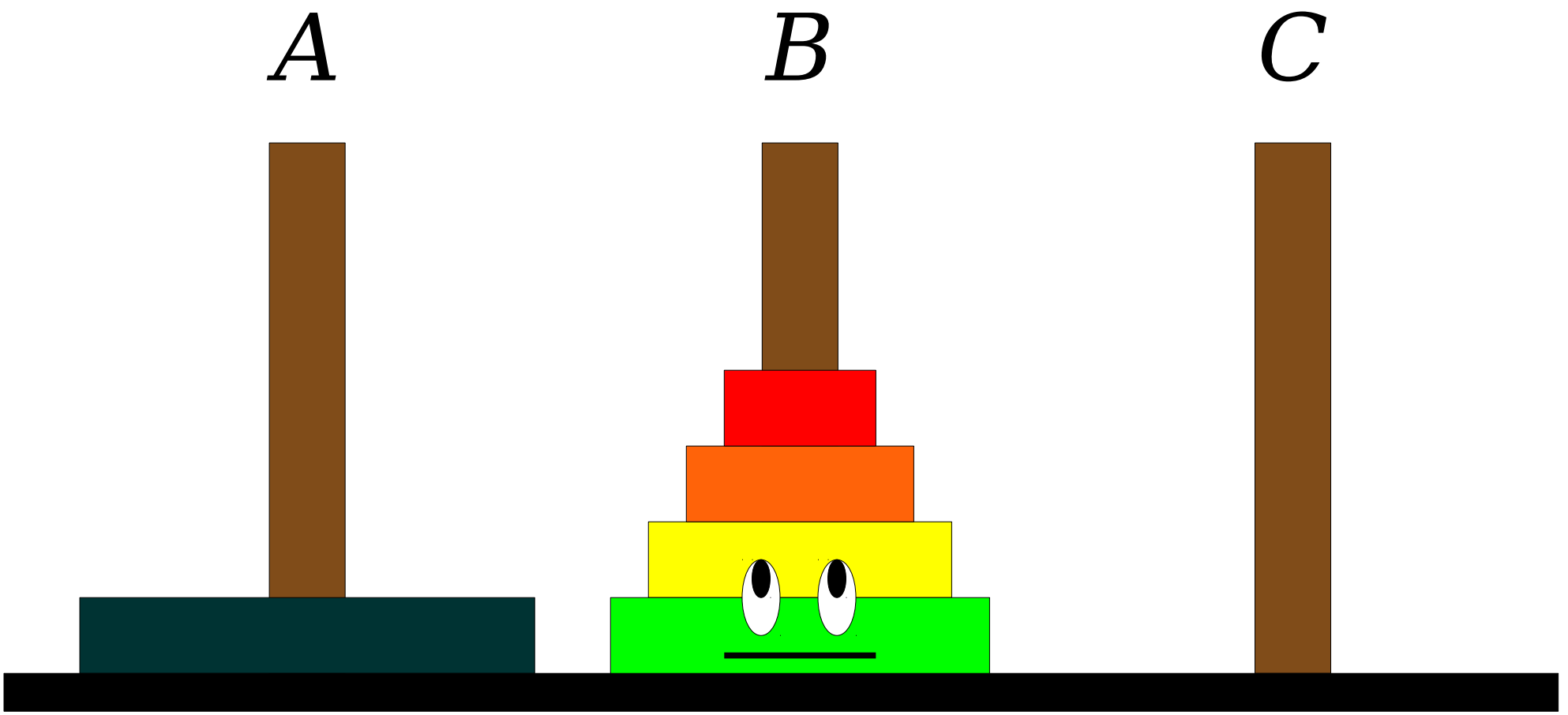
Step One: Move the three smaller disks from Spindle *A* to Spindle *C*.
Step Two: Move the green disk from Spindle *A* to Spindle *B*.

Solving the Towers of Hanoi



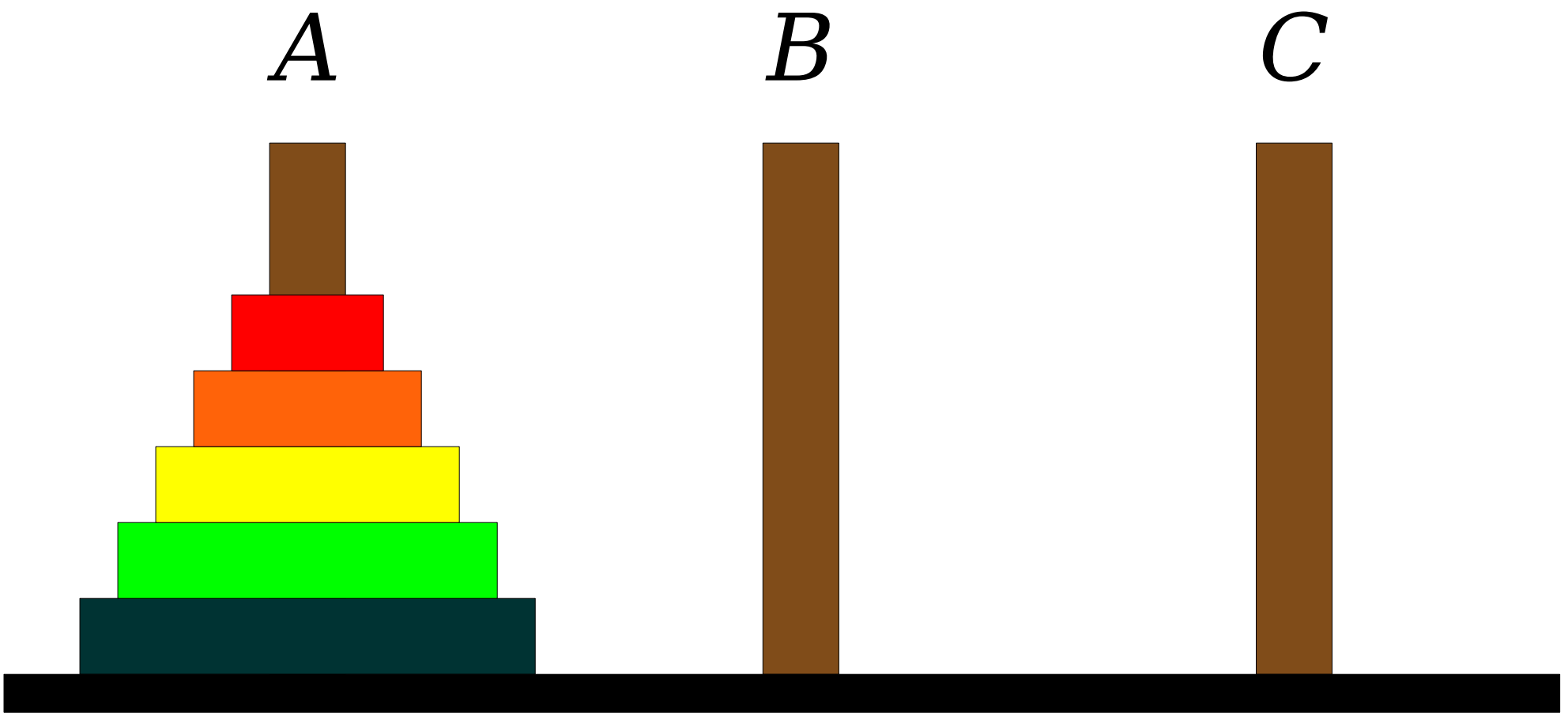
Step One: Move the three smaller disks from Spindle *A* to Spindle *C*.
Step Two: Move the green disk from Spindle *A* to Spindle *B*.

Solving the Towers of Hanoi

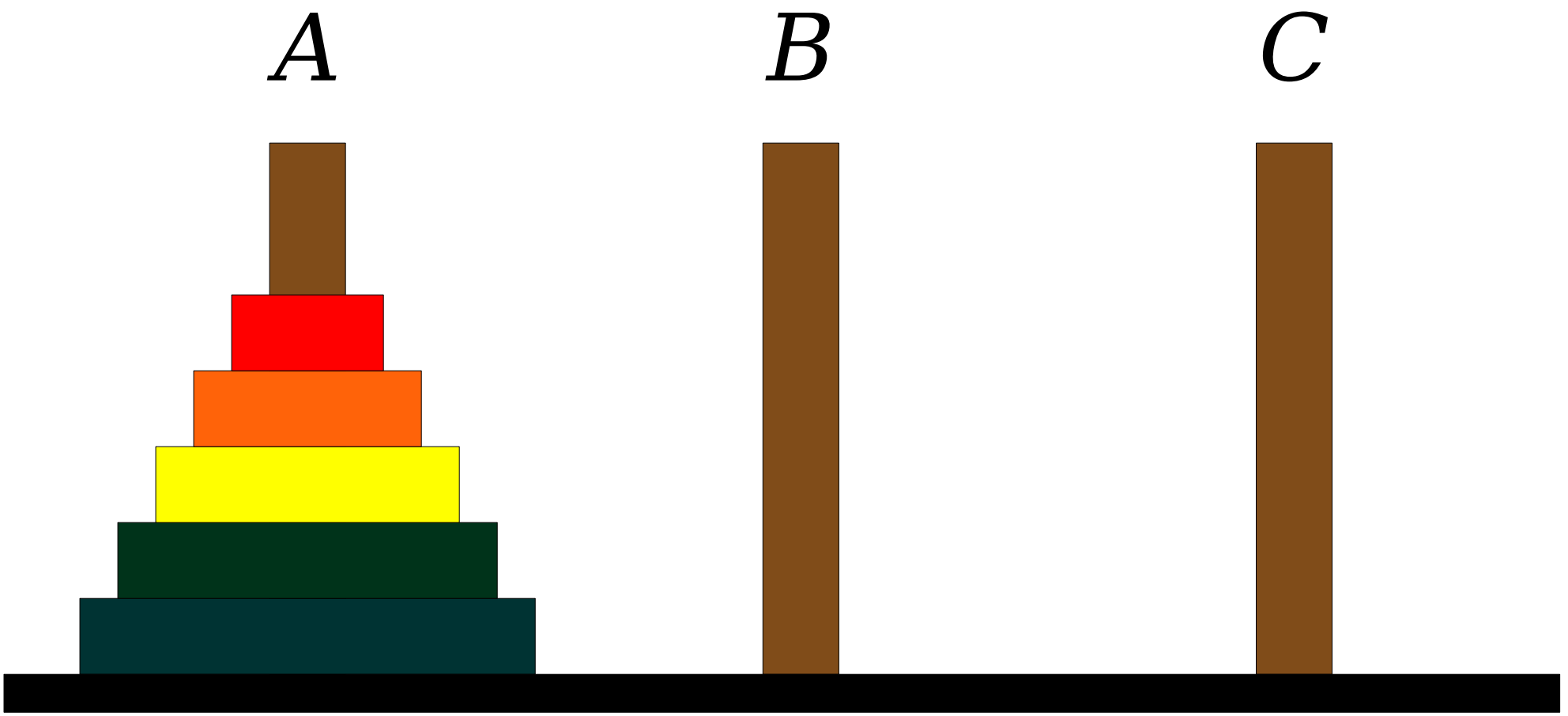


Step One: Move the three smaller disks from Spindle *A* to Spindle *C*.
Step Two: Move the green disk from Spindle *A* to Spindle *B*.
Step Three: Move the three smaller disks from Spindle *C* to Spindle *B*.

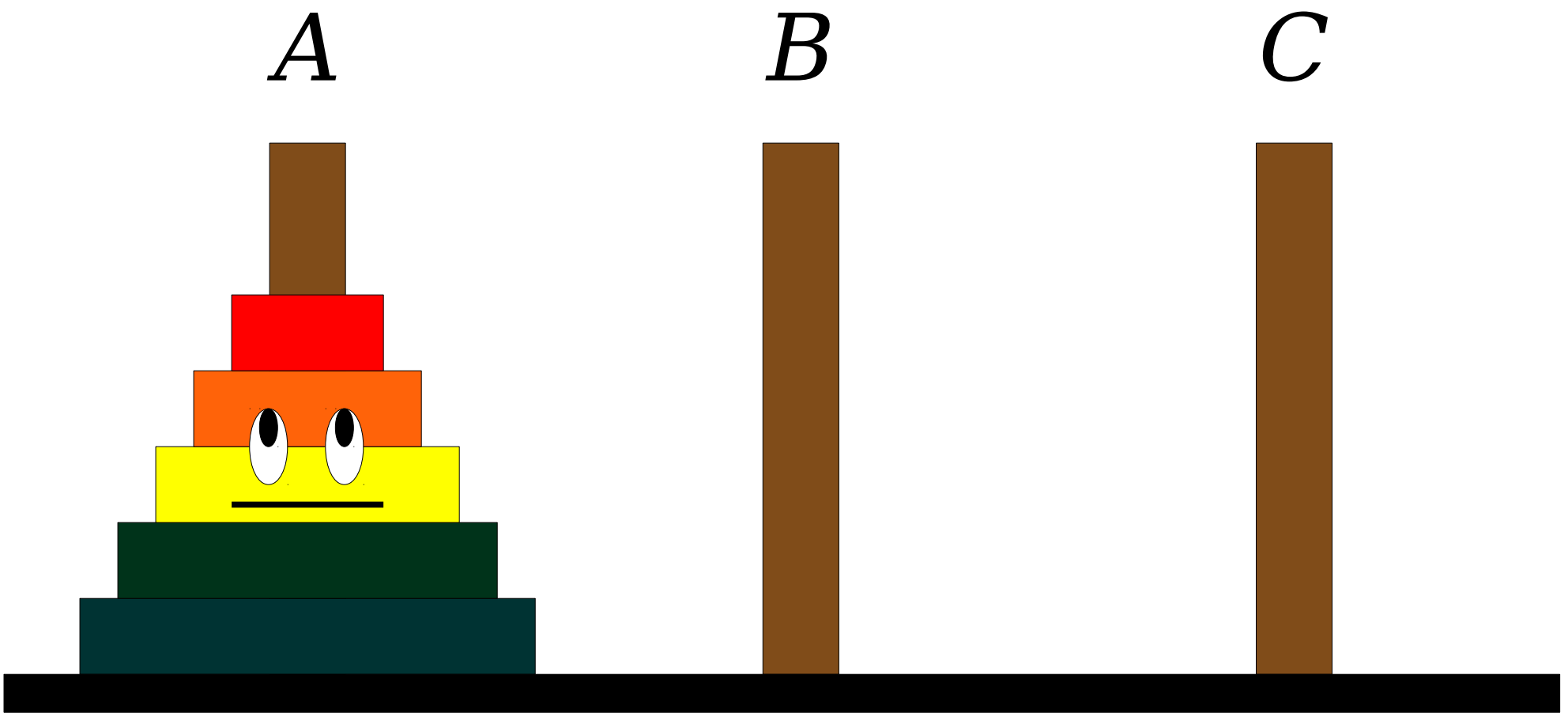
Solving the Towers of Hanoi



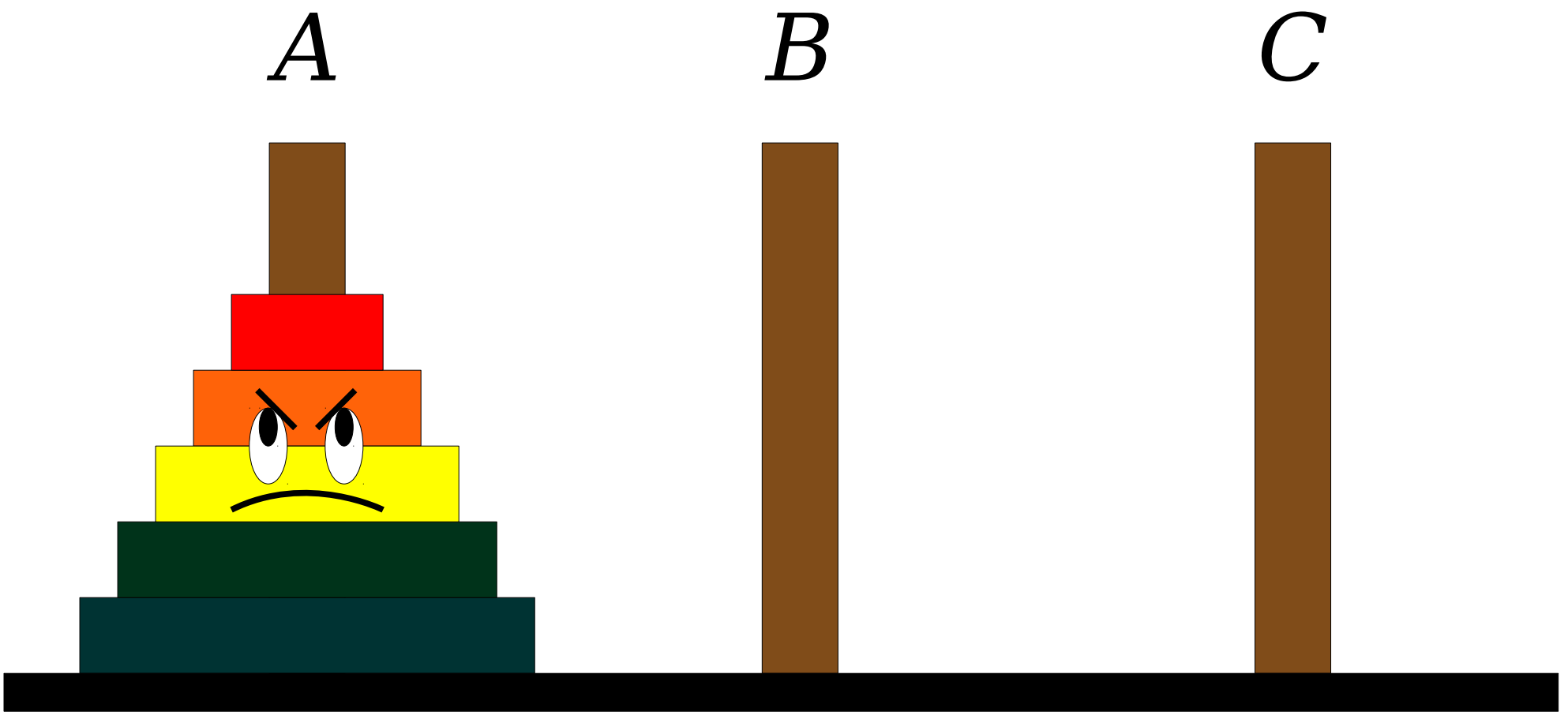
Solving the Towers of Hanoi



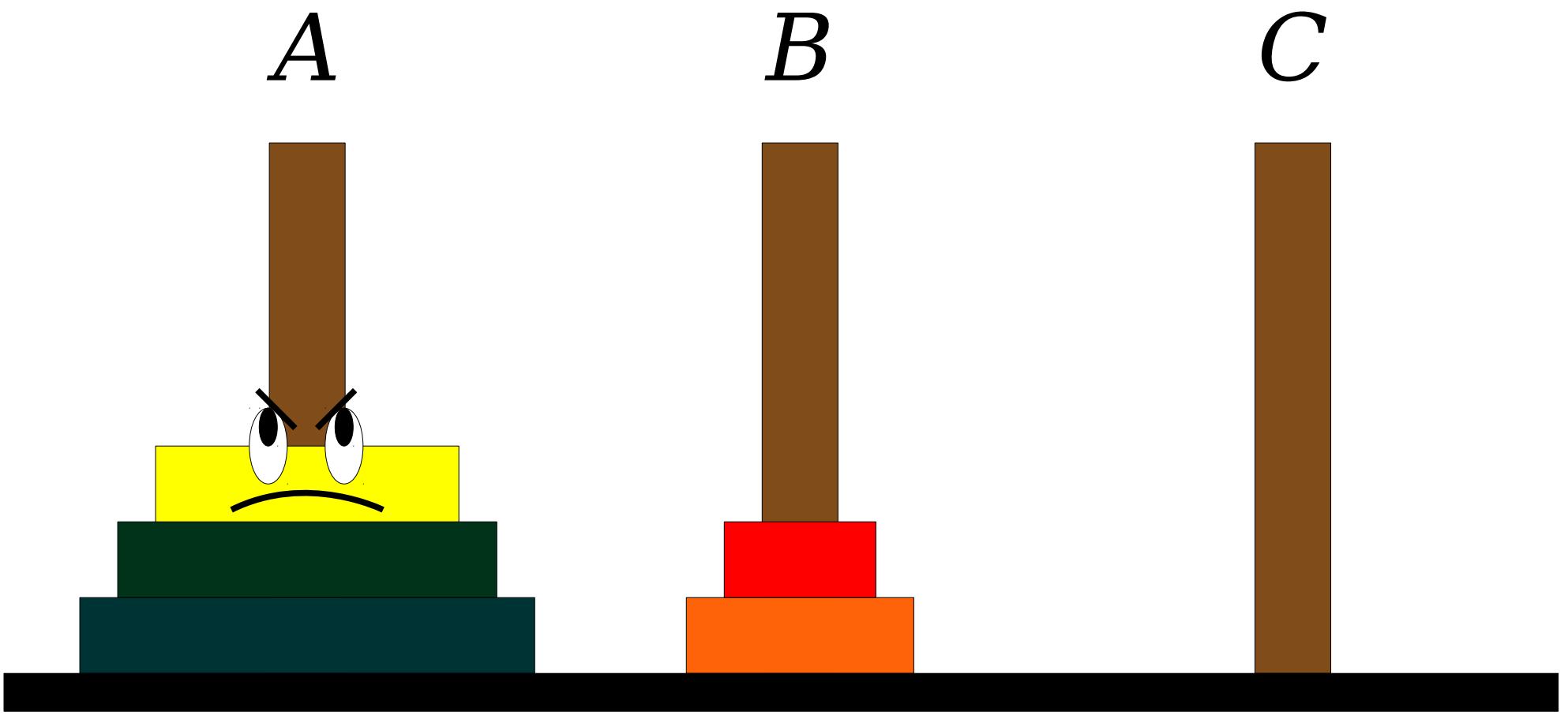
Solving the Towers of Hanoi



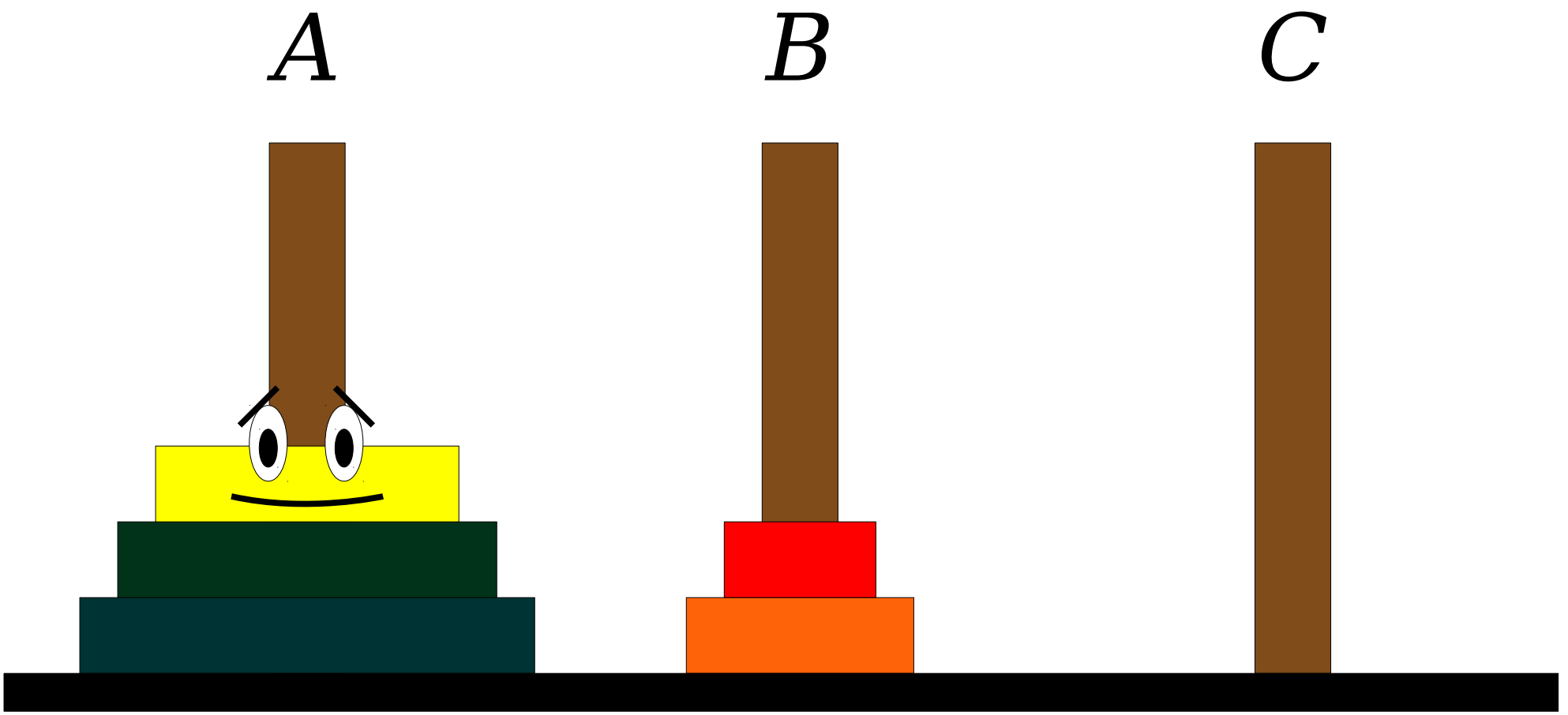
Solving the Towers of Hanoi



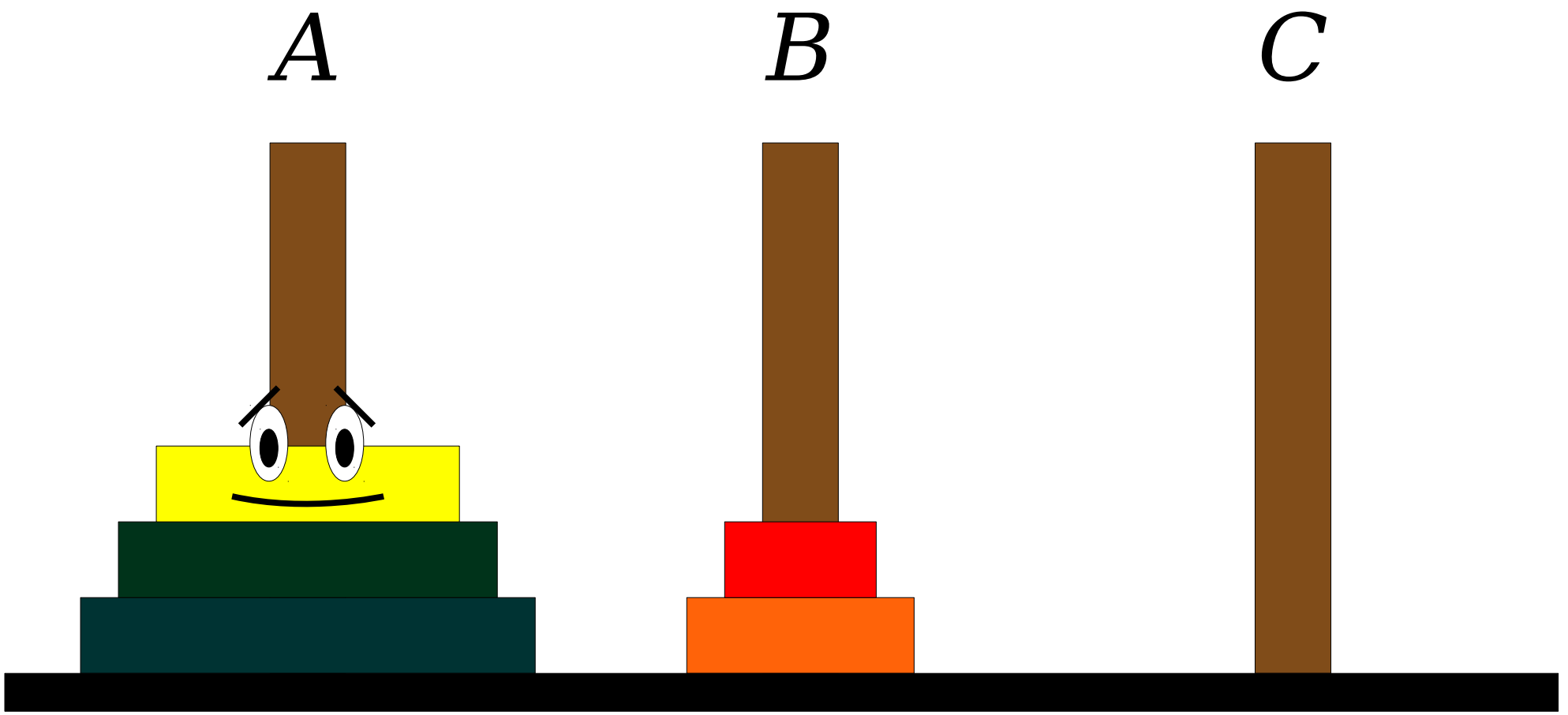
Solving the Towers of Hanoi



Solving the Towers of Hanoi

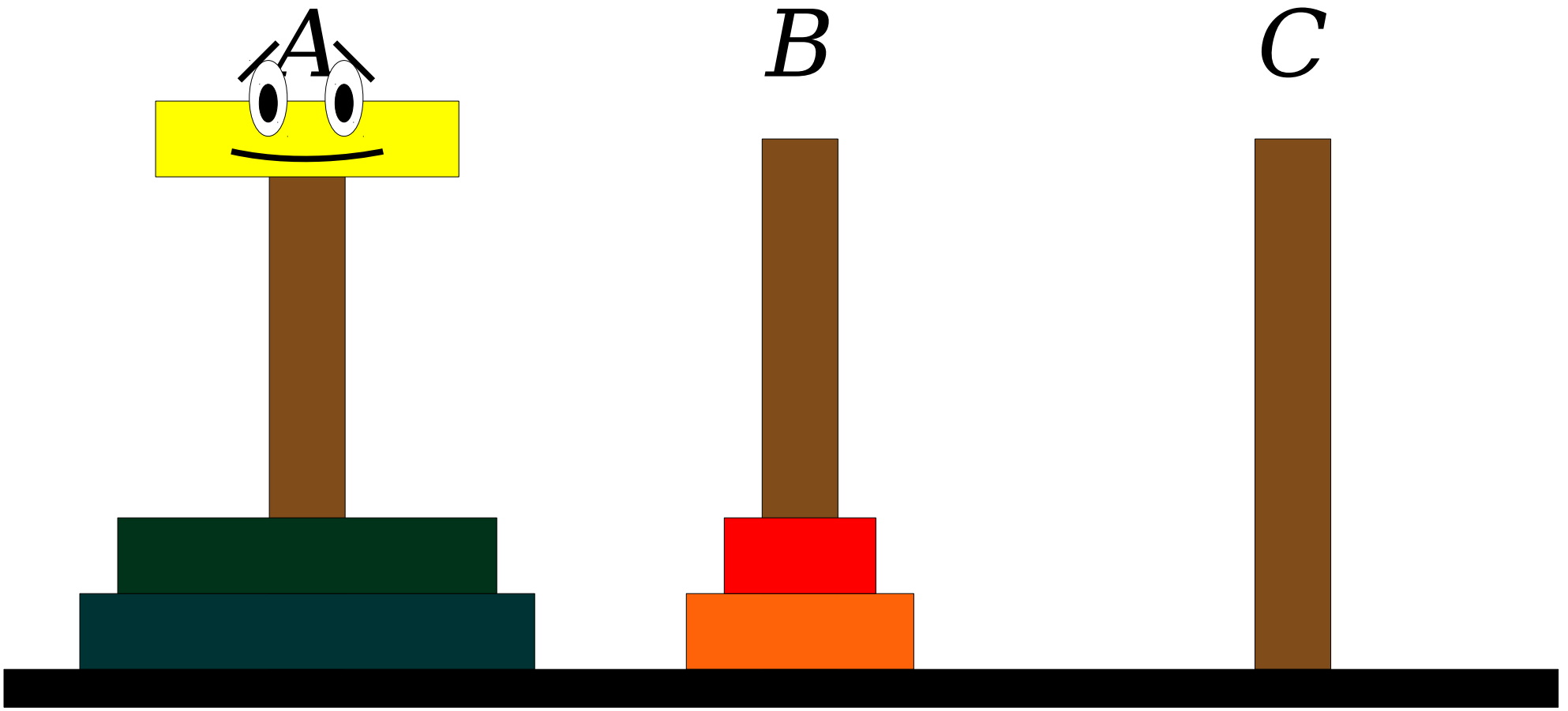


Solving the Towers of Hanoi



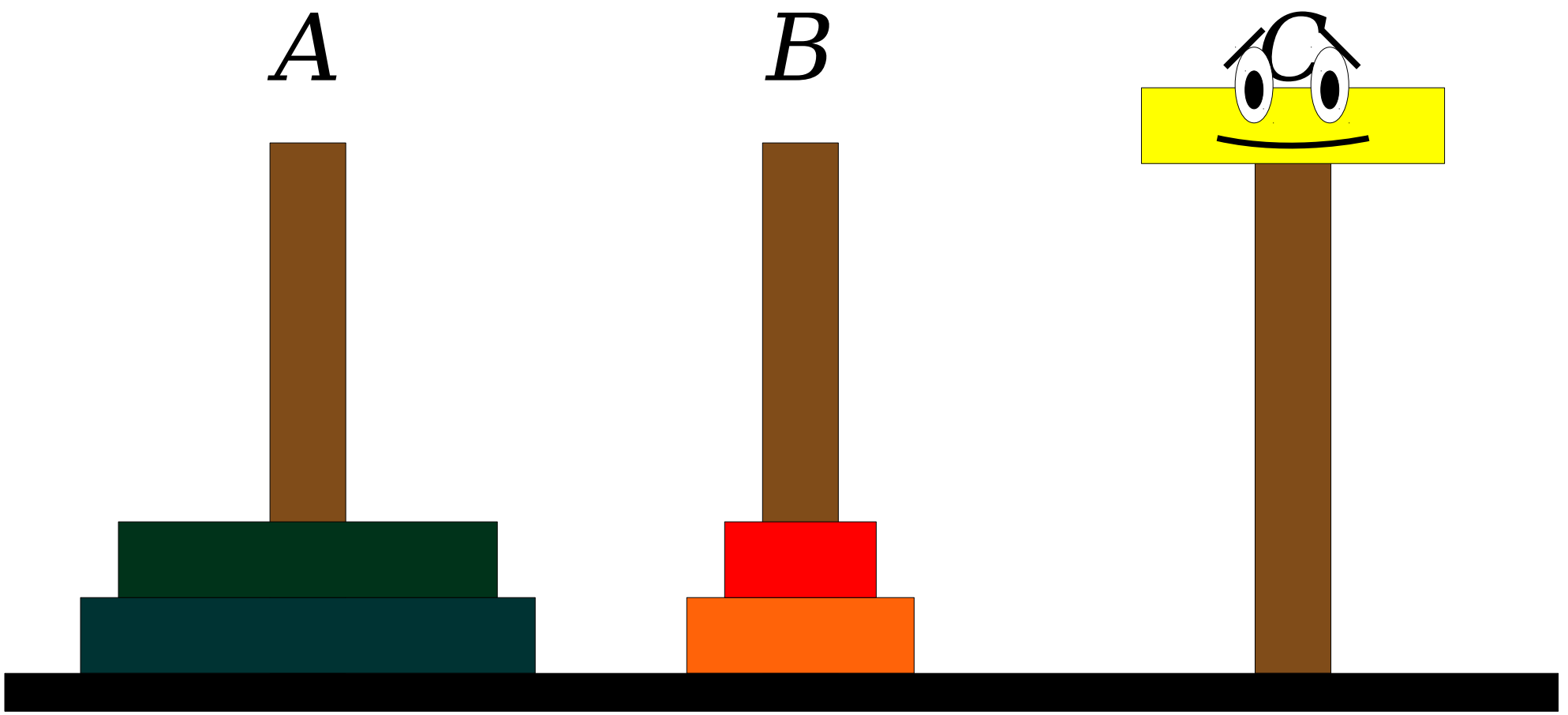
Step One: Move the two smaller disks from Spindle *A* to Spindle *B*.

Solving the Towers of Hanoi



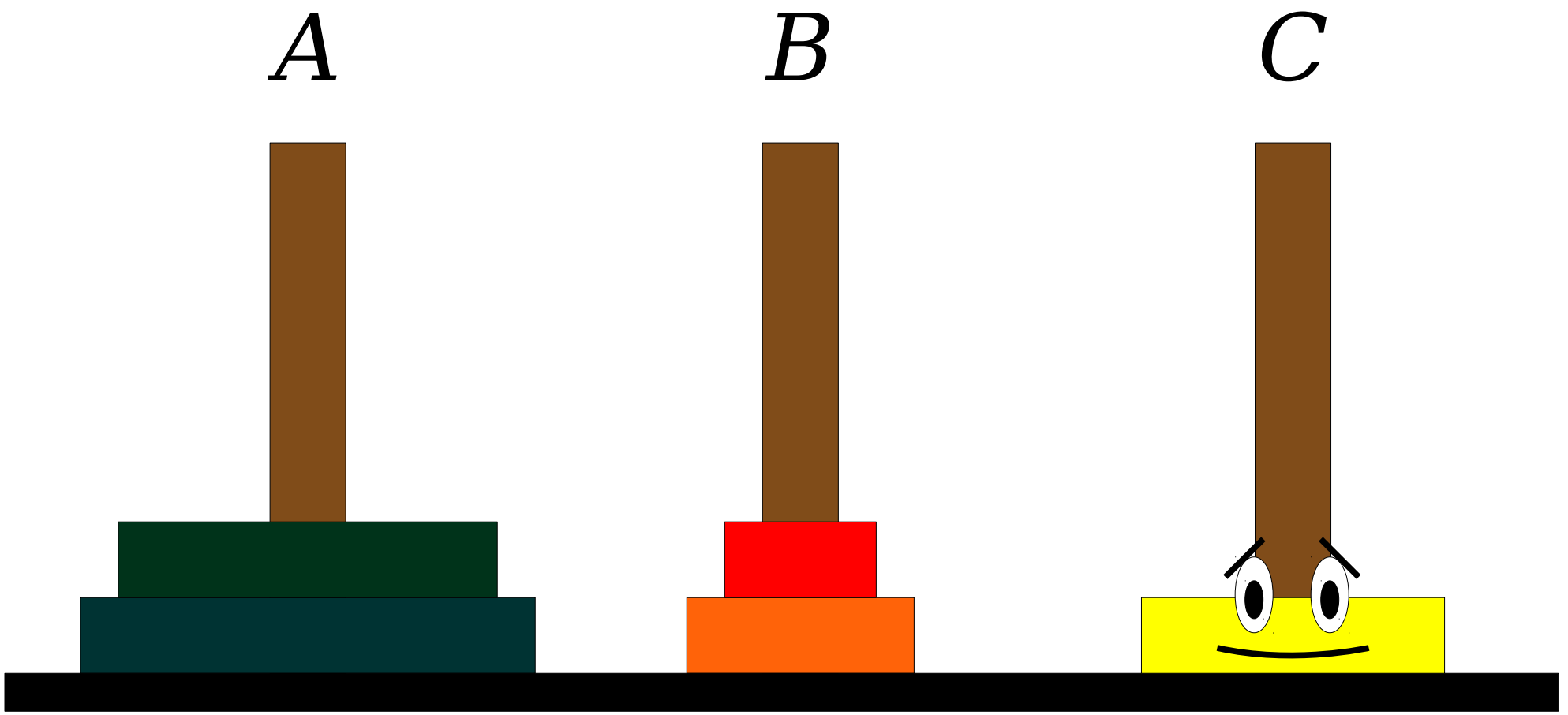
Step One: Move the two smaller disks from Spindle *A* to Spindle *B*.

Solving the Towers of Hanoi



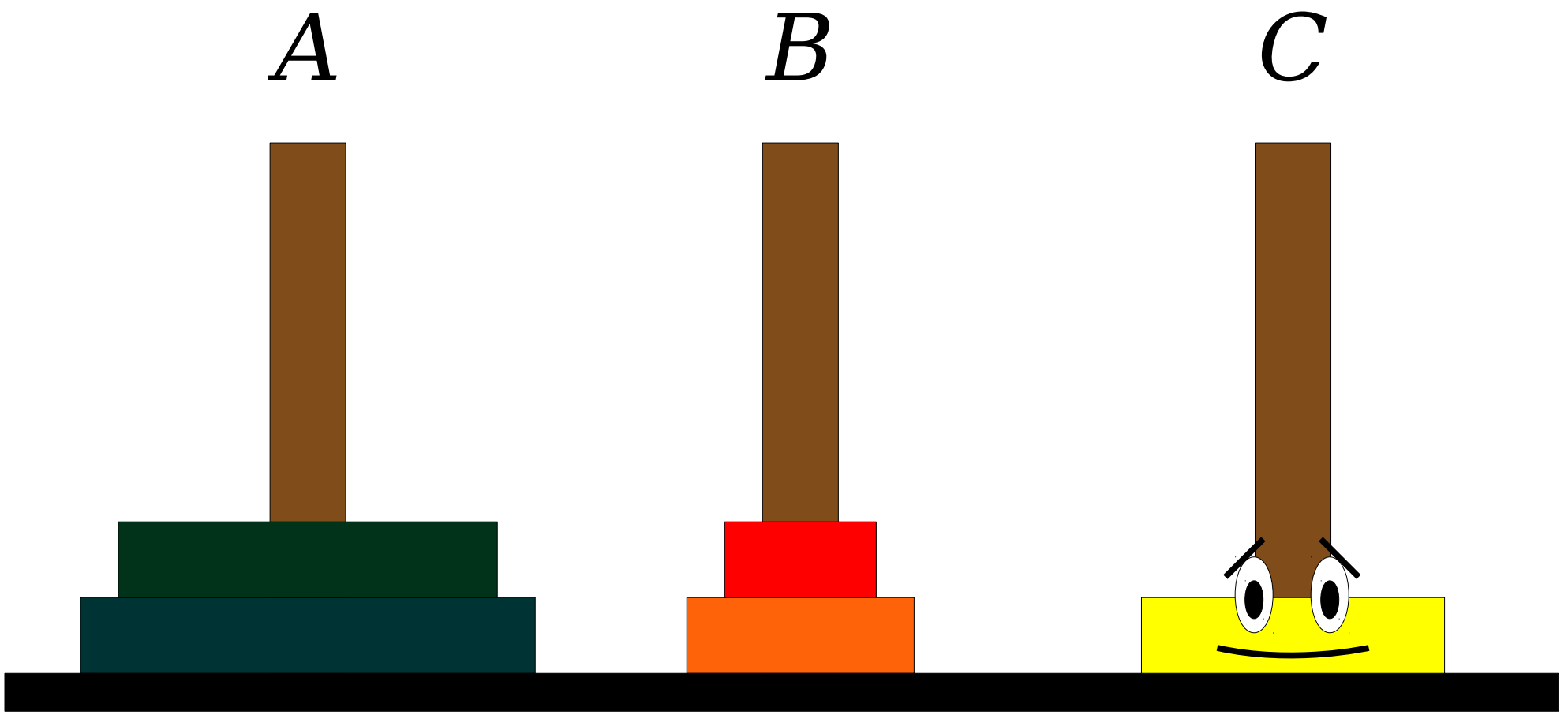
Step One: Move the two smaller disks from Spindle *A* to Spindle *B*.

Solving the Towers of Hanoi



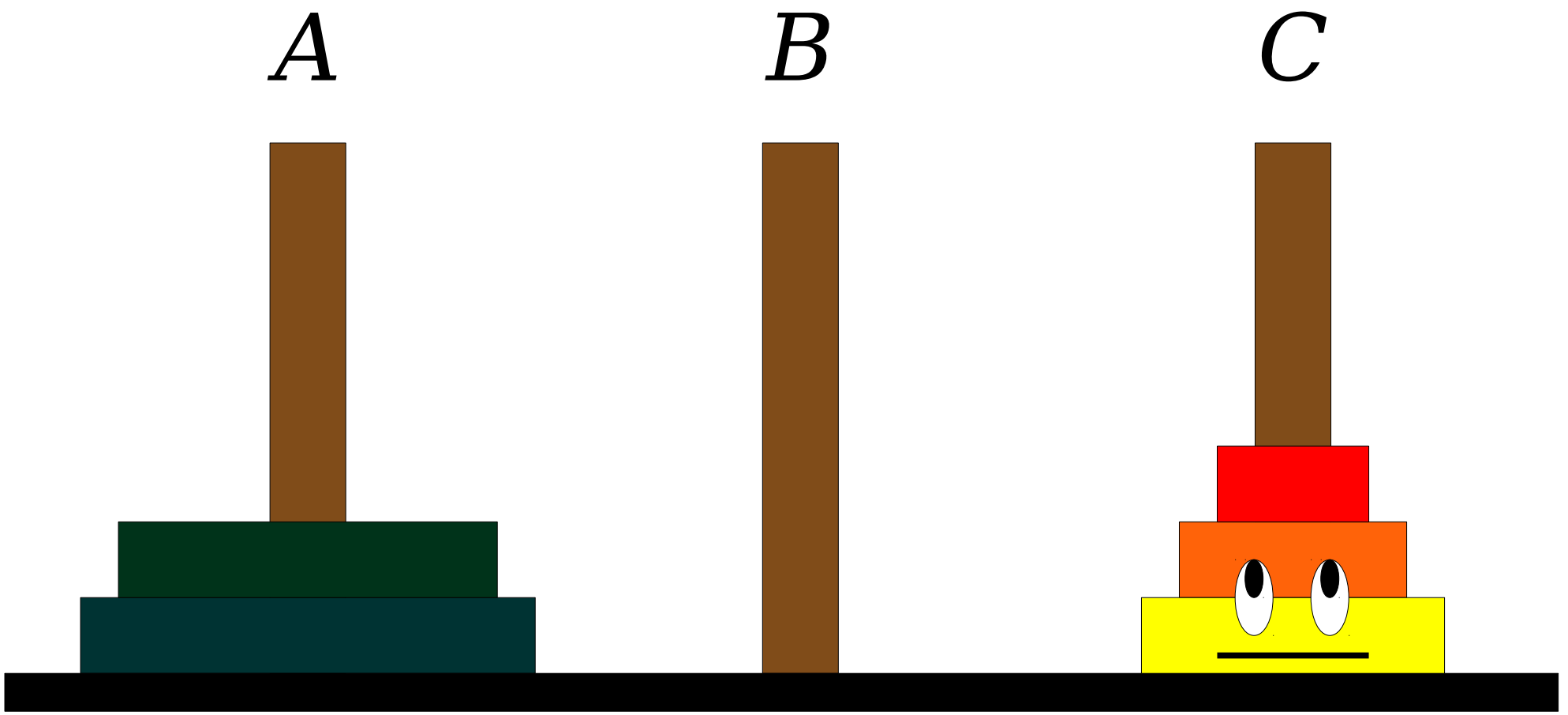
Step One: Move the two smaller disks from Spindle *A* to Spindle *B*.

Solving the Towers of Hanoi



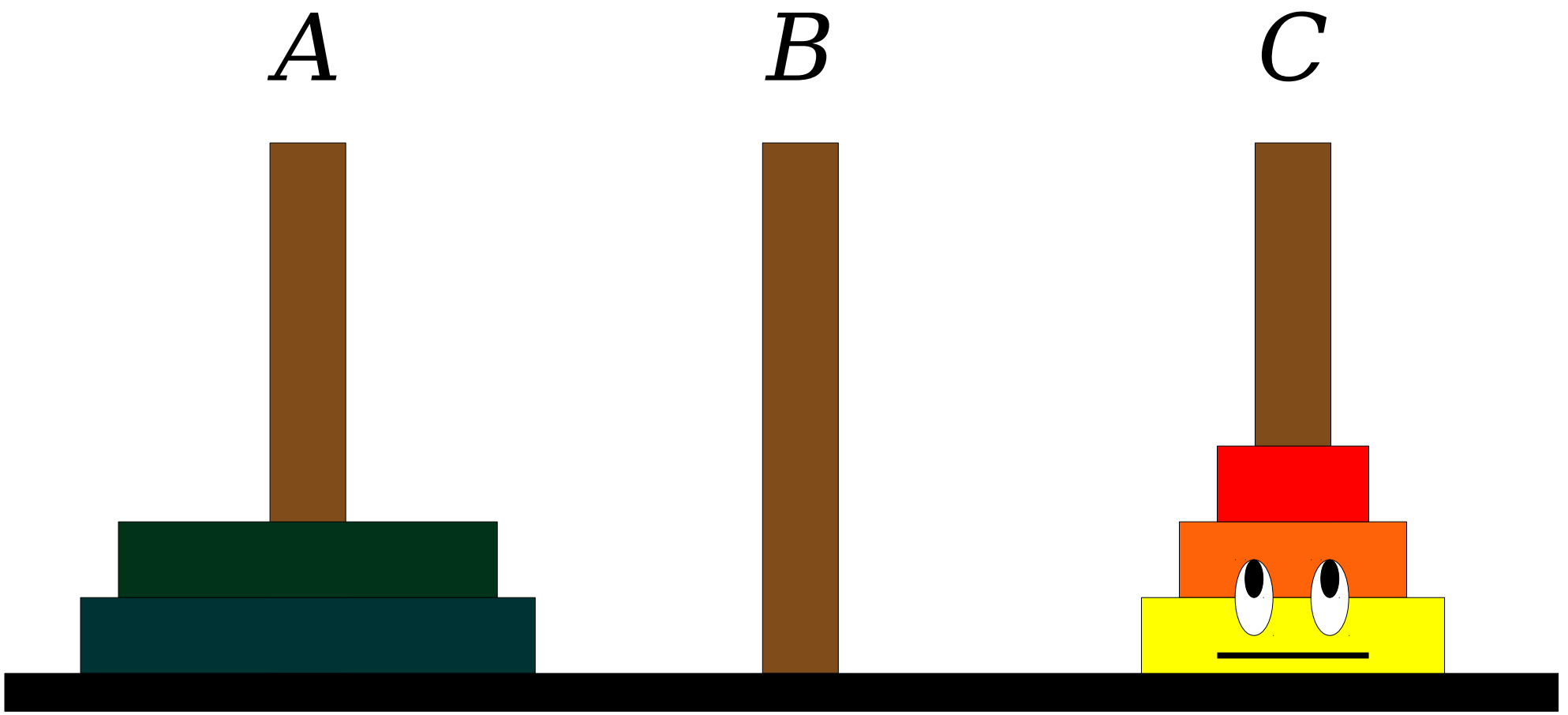
Step One: Move the two smaller disks from Spindle *A* to Spindle *B*.
Step Two: Move the yellow disk from Spindle *A* to Spindle *C*.

Solving the Towers of Hanoi



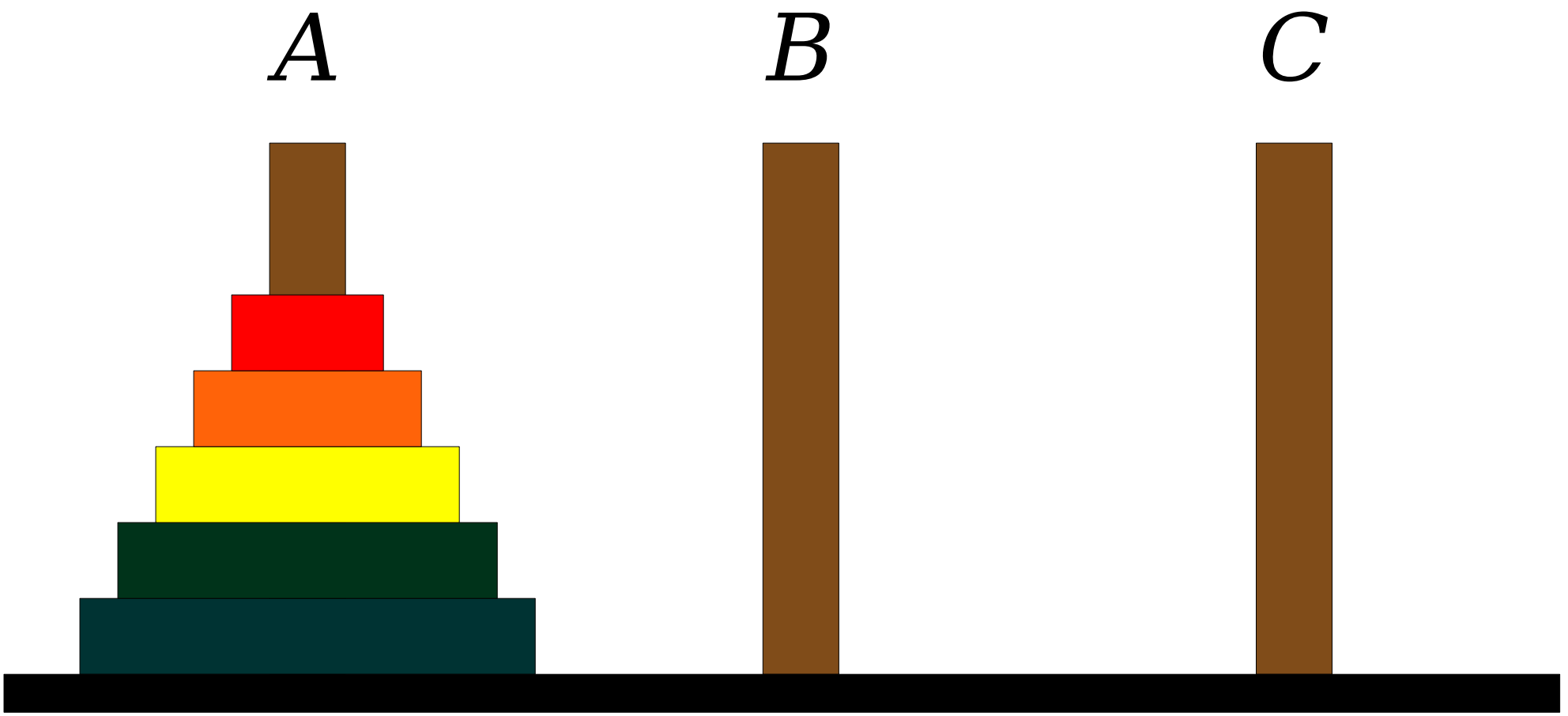
Step One: Move the two smaller disks from Spindle *A* to Spindle *B*.
Step Two: Move the yellow disk from Spindle *A* to Spindle *C*.

Solving the Towers of Hanoi

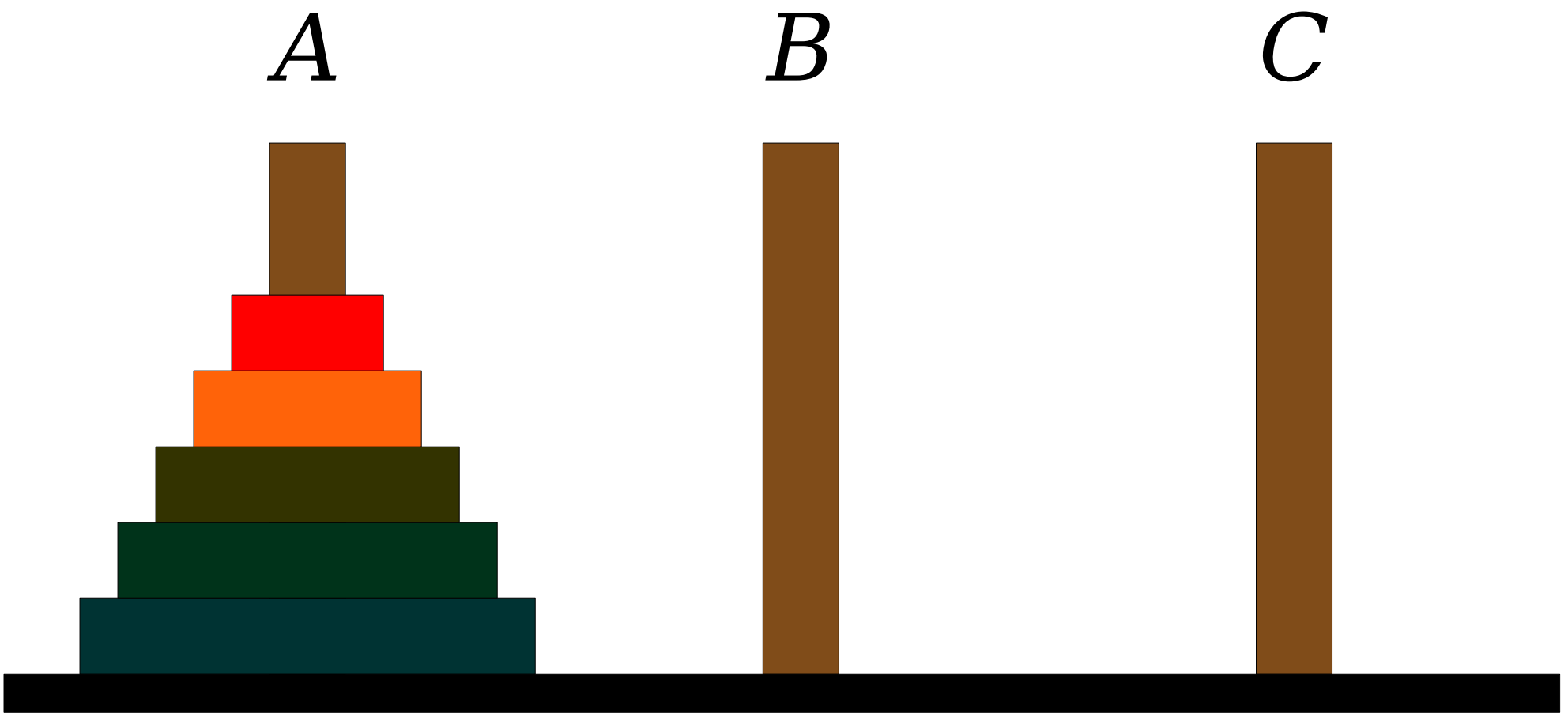


Step One: Move the two smaller disks from Spindle *A* to Spindle *B*.
Step Two: Move the yellow disk from Spindle *A* to Spindle *C*.
Step Three: Move the two smaller disks from Spindle *B* to Spindle *C*.

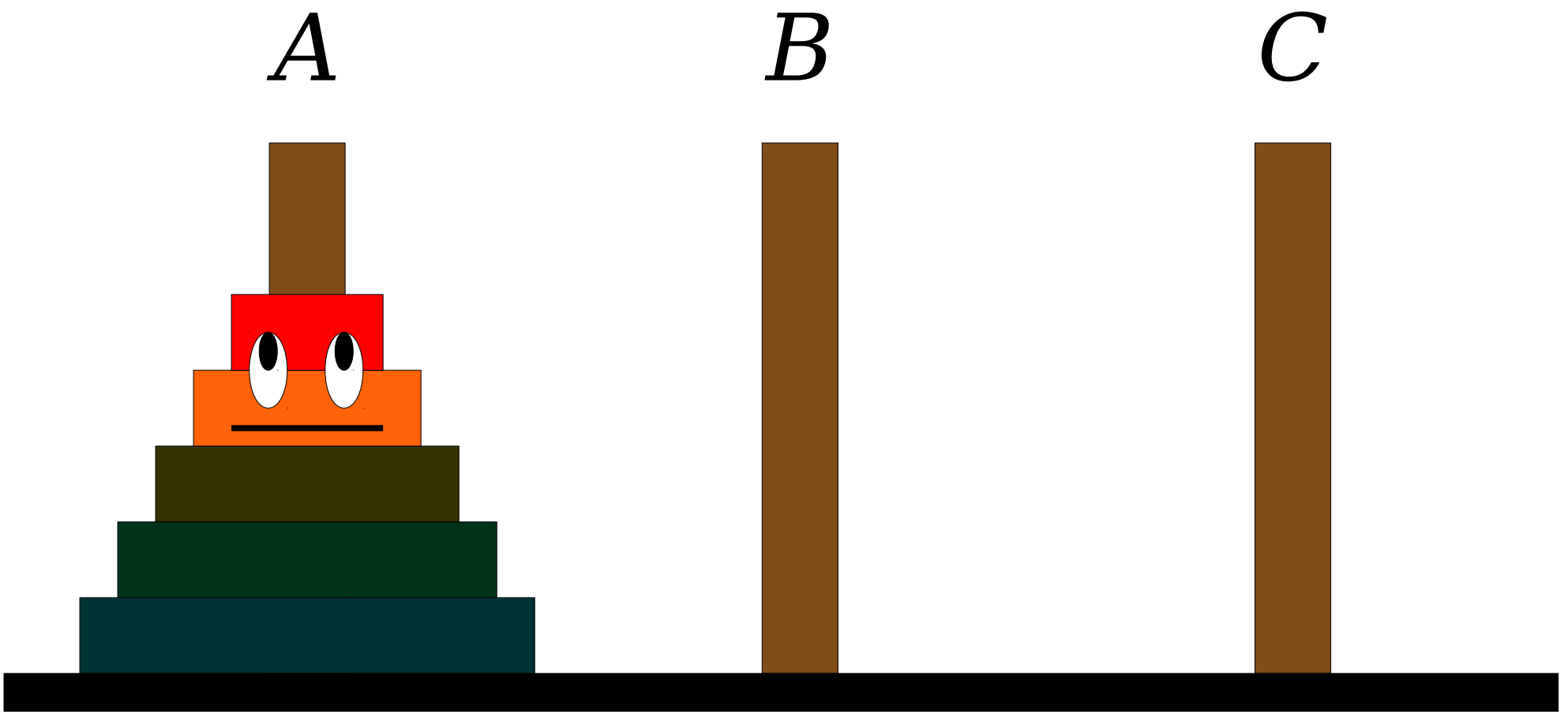
Solving the Towers of Hanoi



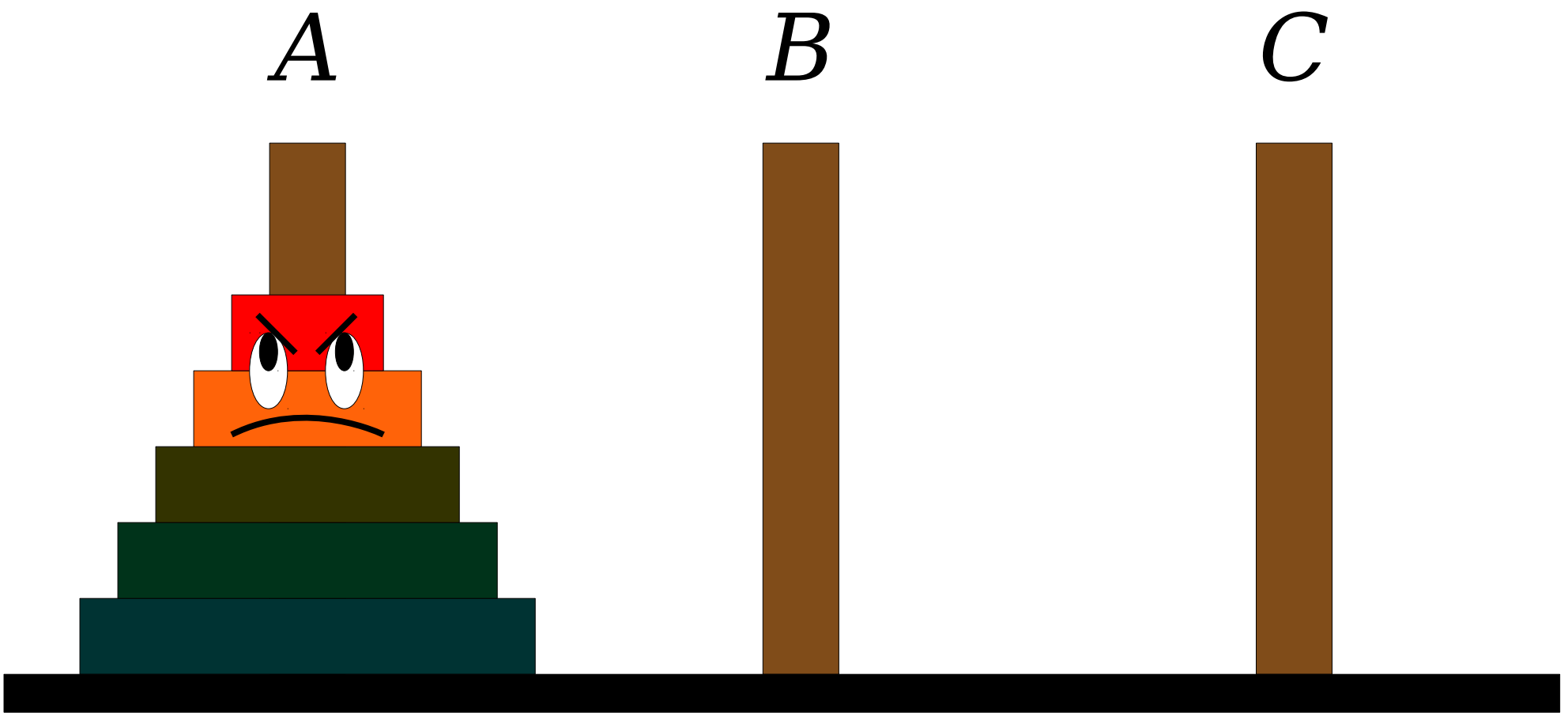
Solving the Towers of Hanoi



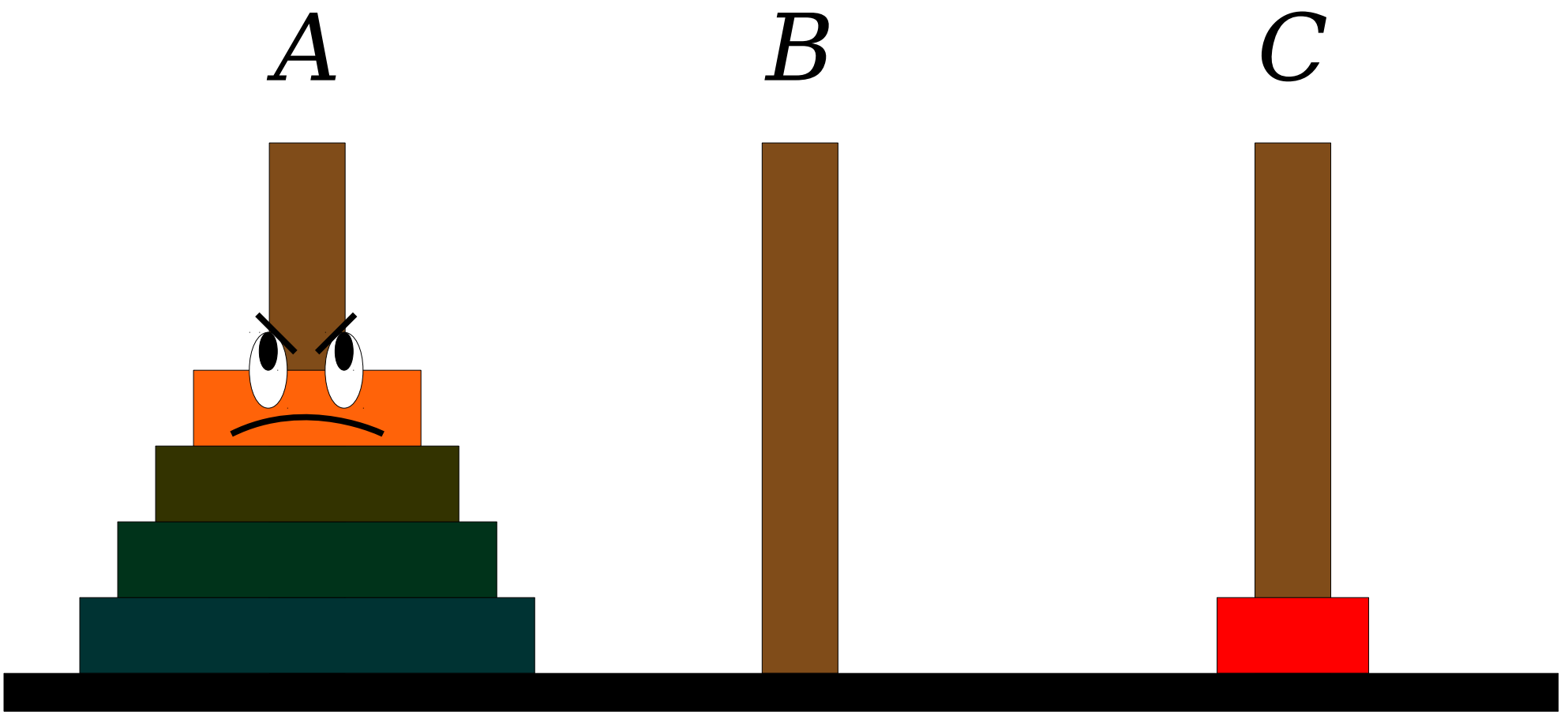
Solving the Towers of Hanoi



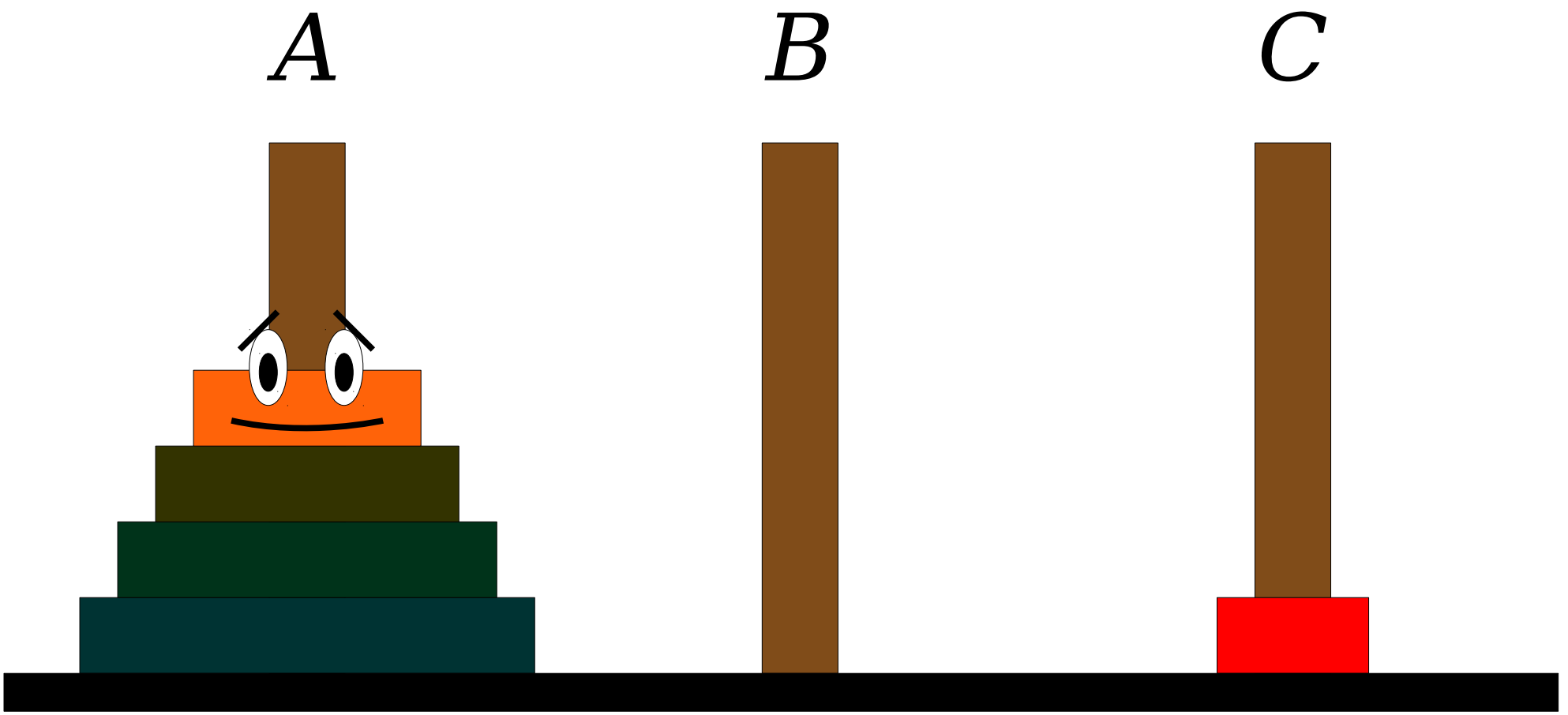
Solving the Towers of Hanoi



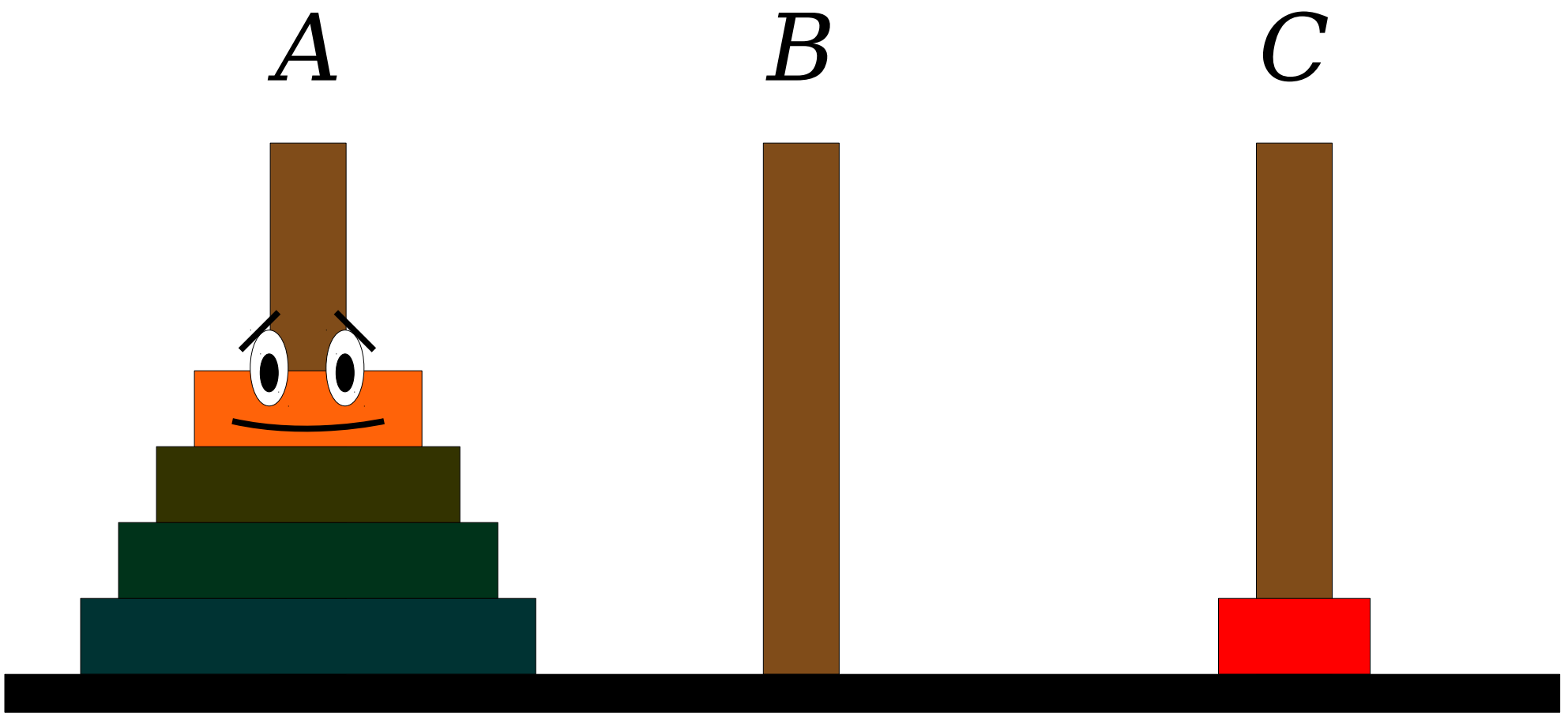
Solving the Towers of Hanoi



Solving the Towers of Hanoi

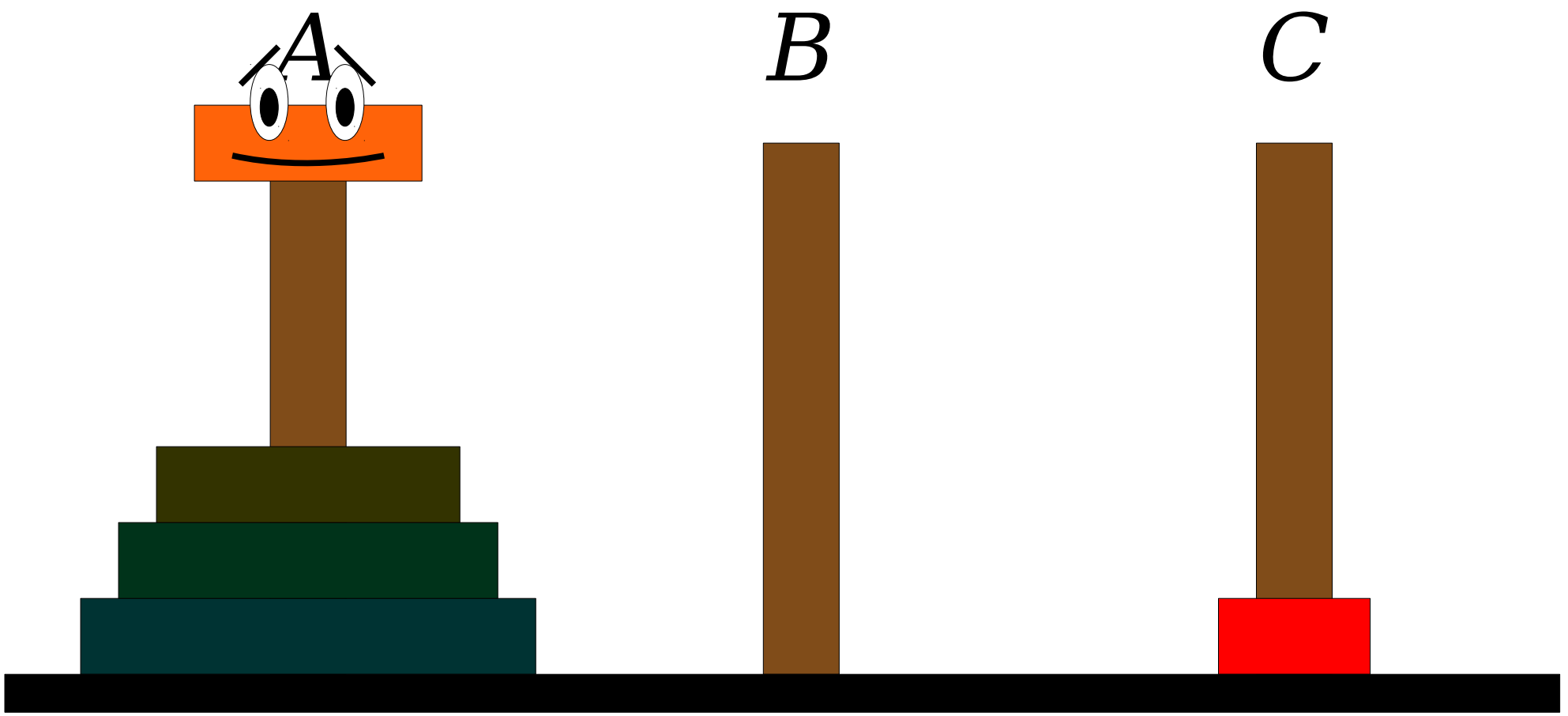


Solving the Towers of Hanoi



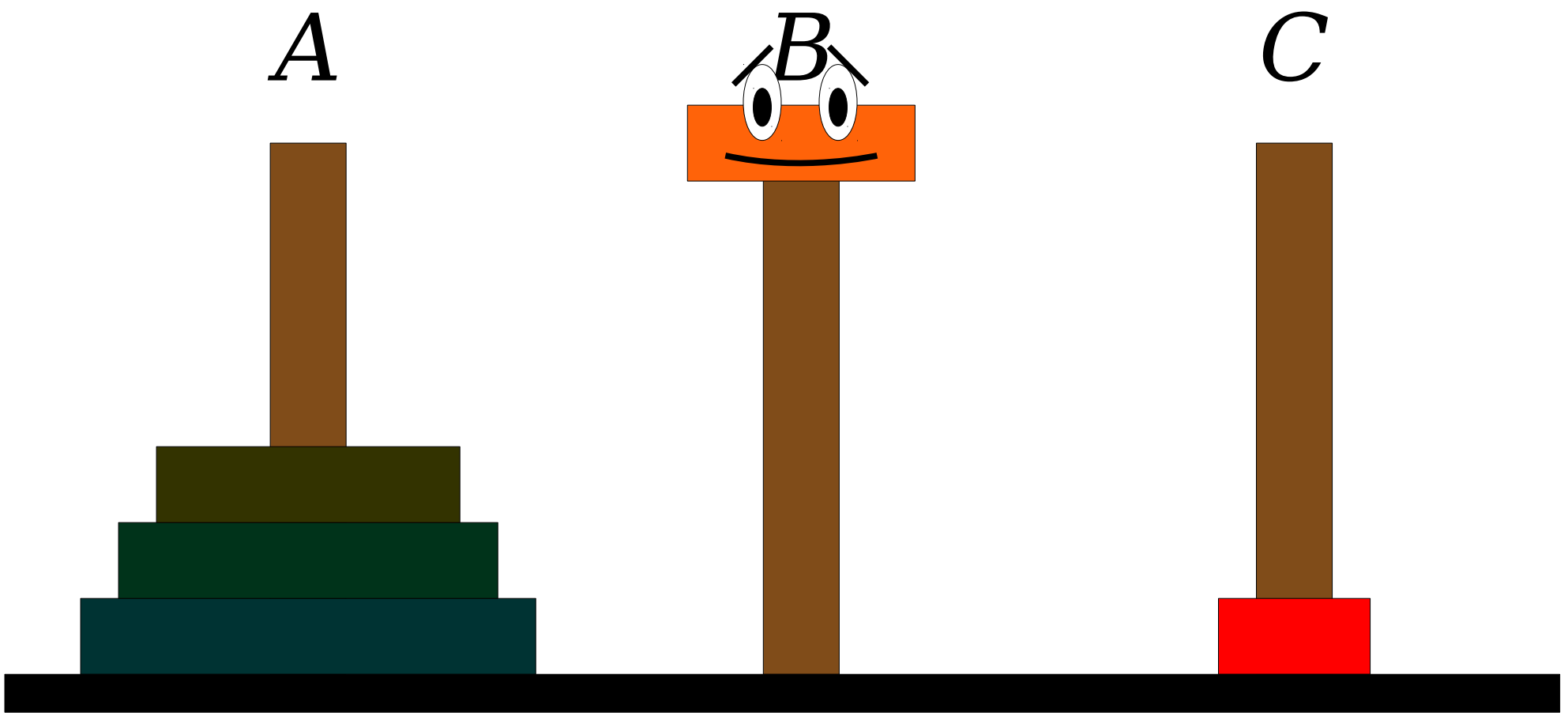
Step One: Move the smallest disk from Spindle *A* to Spindle *C*.

Solving the Towers of Hanoi



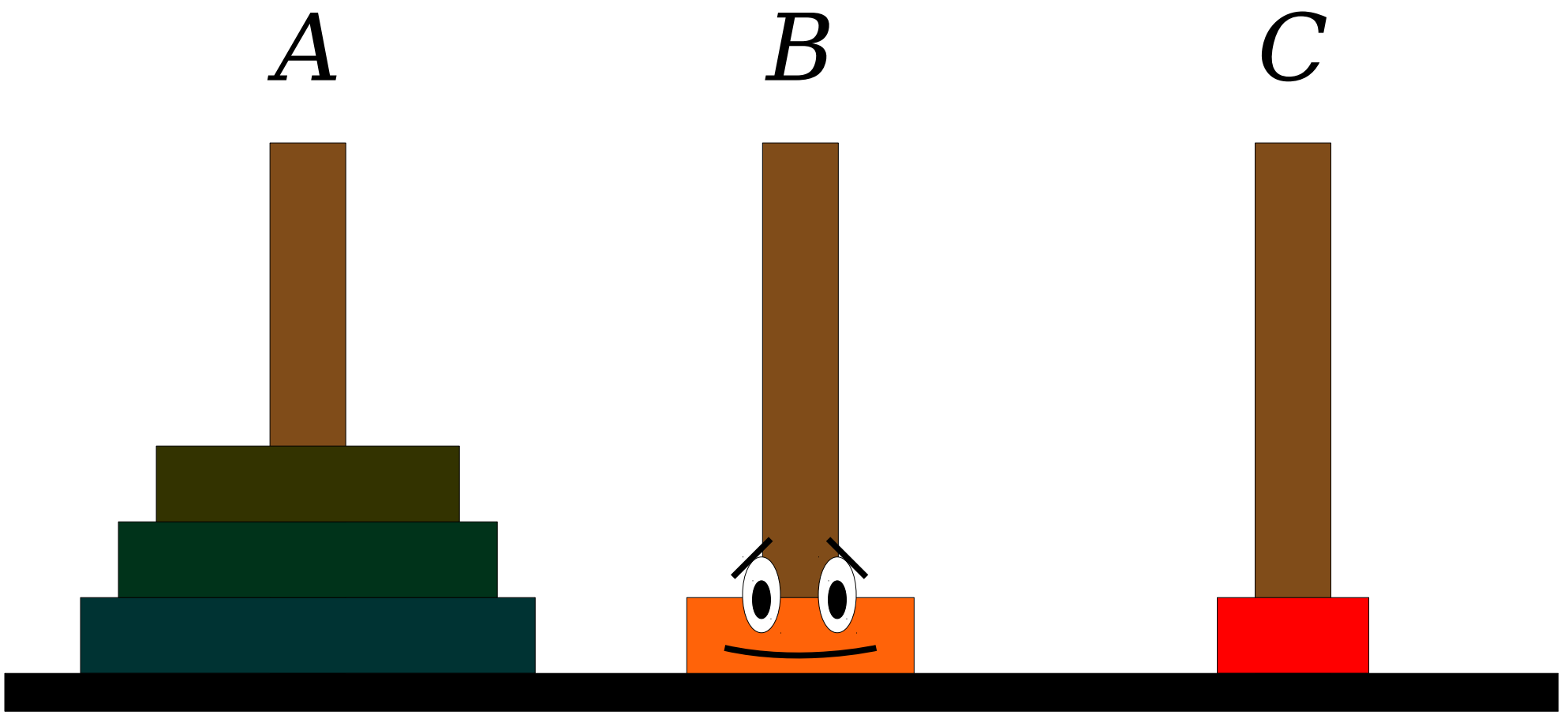
Step One: Move the smallest disk from Spindle *A* to Spindle *C*.

Solving the Towers of Hanoi



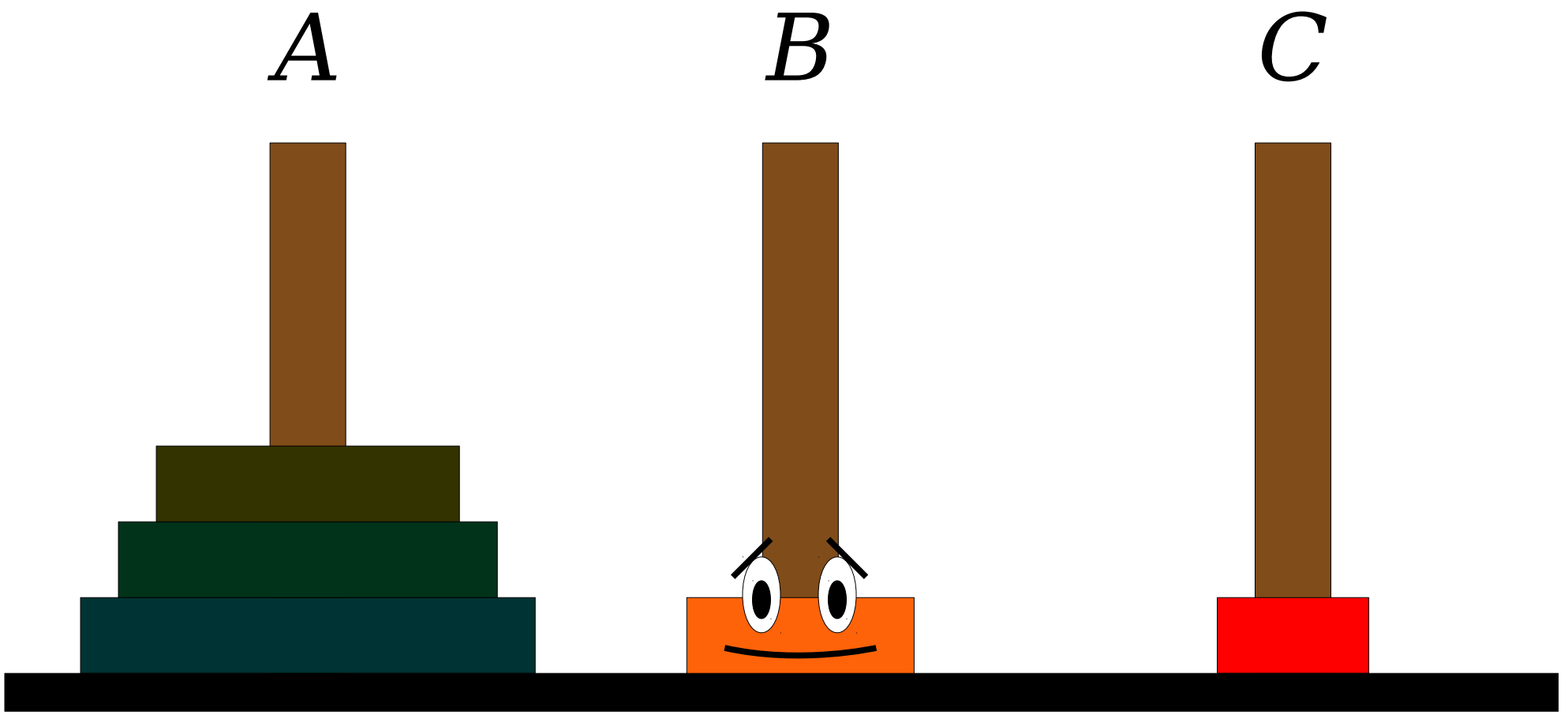
Step One: Move the smallest disk from Spindle *A* to Spindle *C*.

Solving the Towers of Hanoi



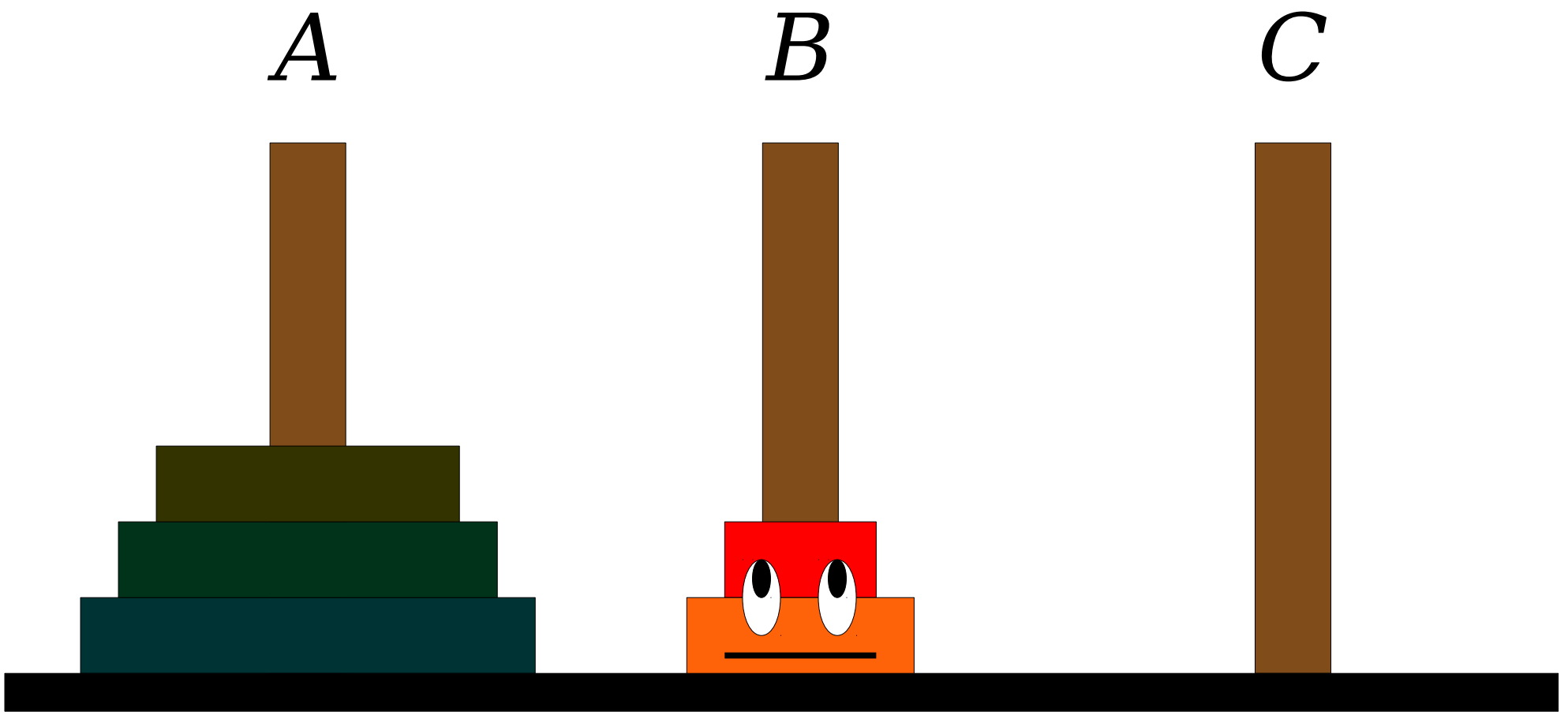
Step One: Move the smallest disk from Spindle *A* to Spindle *C*.

Solving the Towers of Hanoi



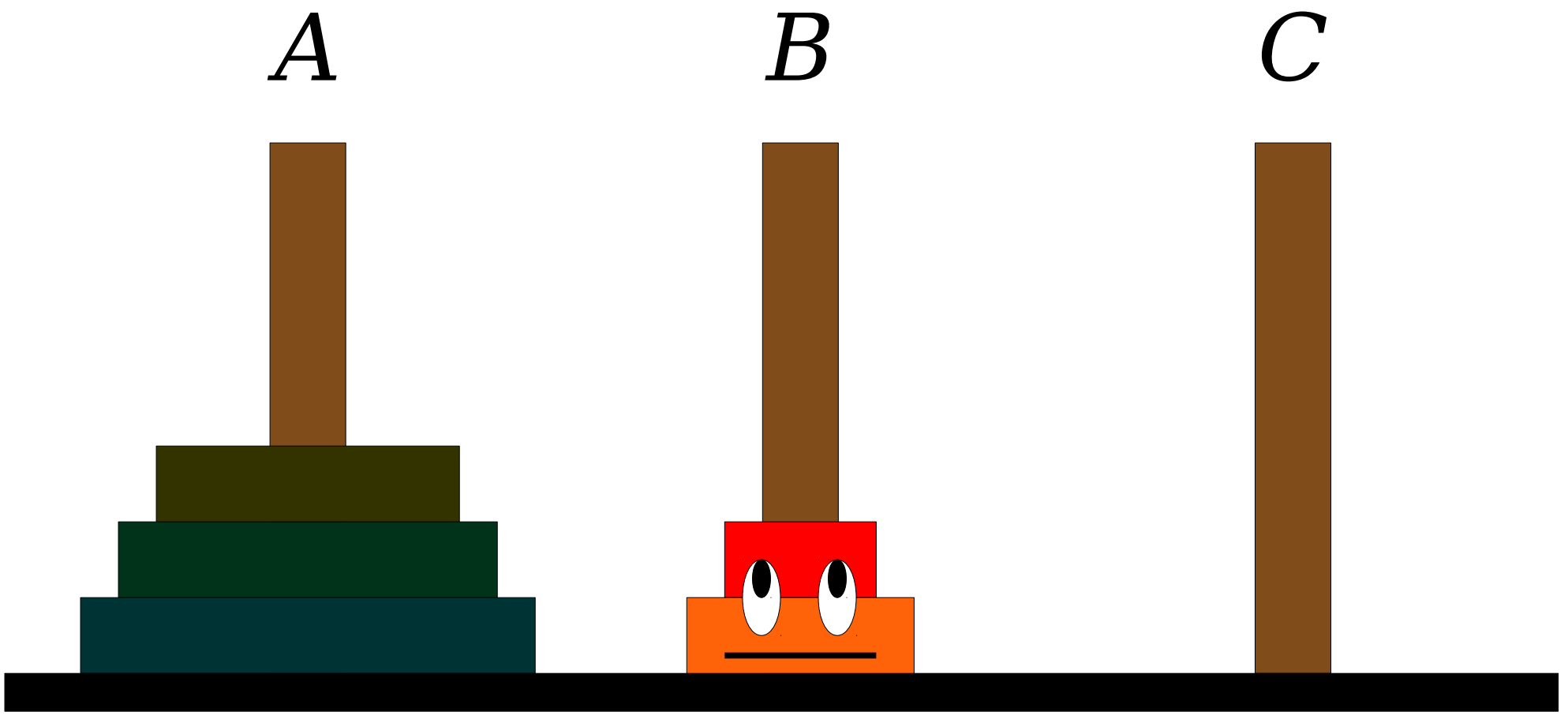
Step One: Move the smallest disk from Spindle *A* to Spindle *C*.
Step Two: Move the orange disk from Spindle *A* to Spindle *B*.

Solving the Towers of Hanoi



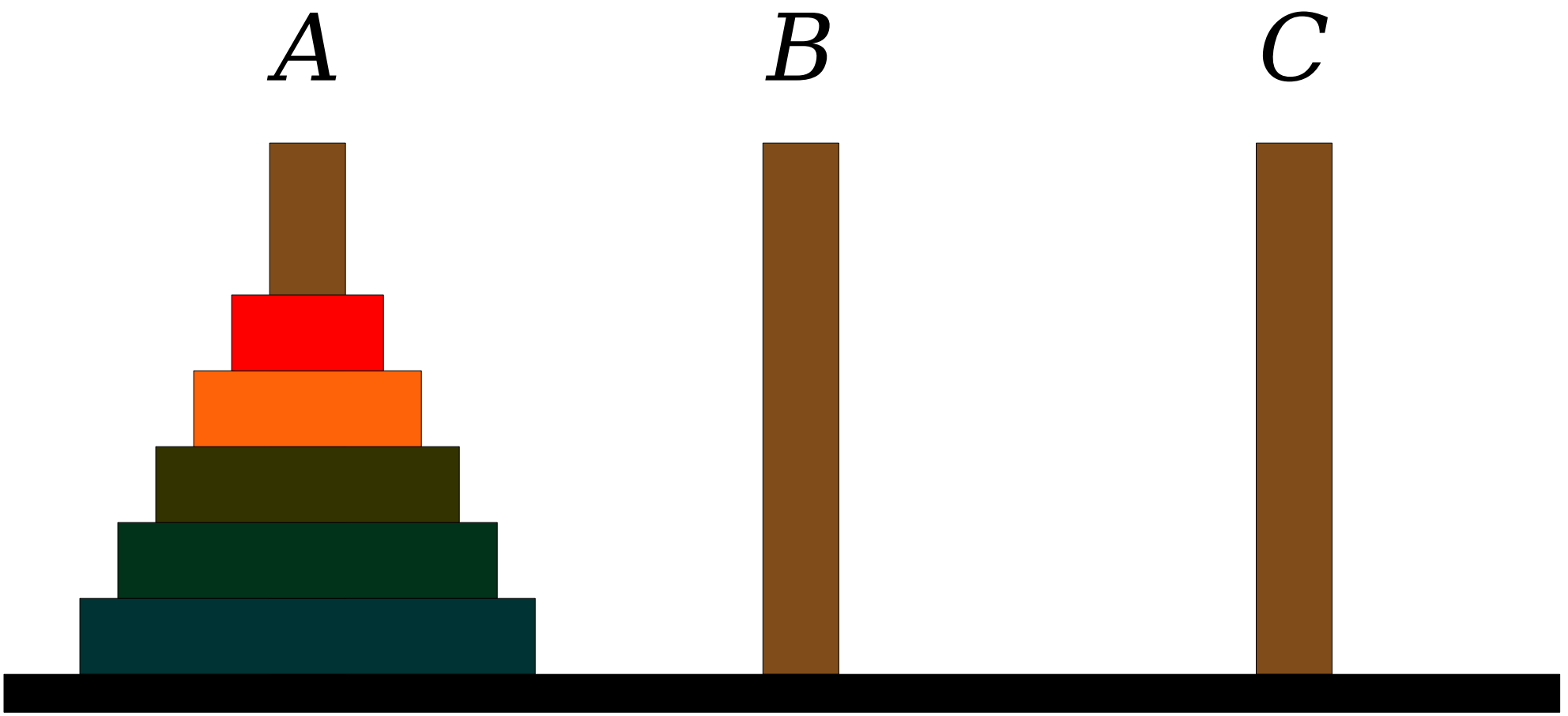
Step One: Move the smallest disk from Spindle *A* to Spindle *C*.
Step Two: Move the orange disk from Spindle *A* to Spindle *B*.

Solving the Towers of Hanoi

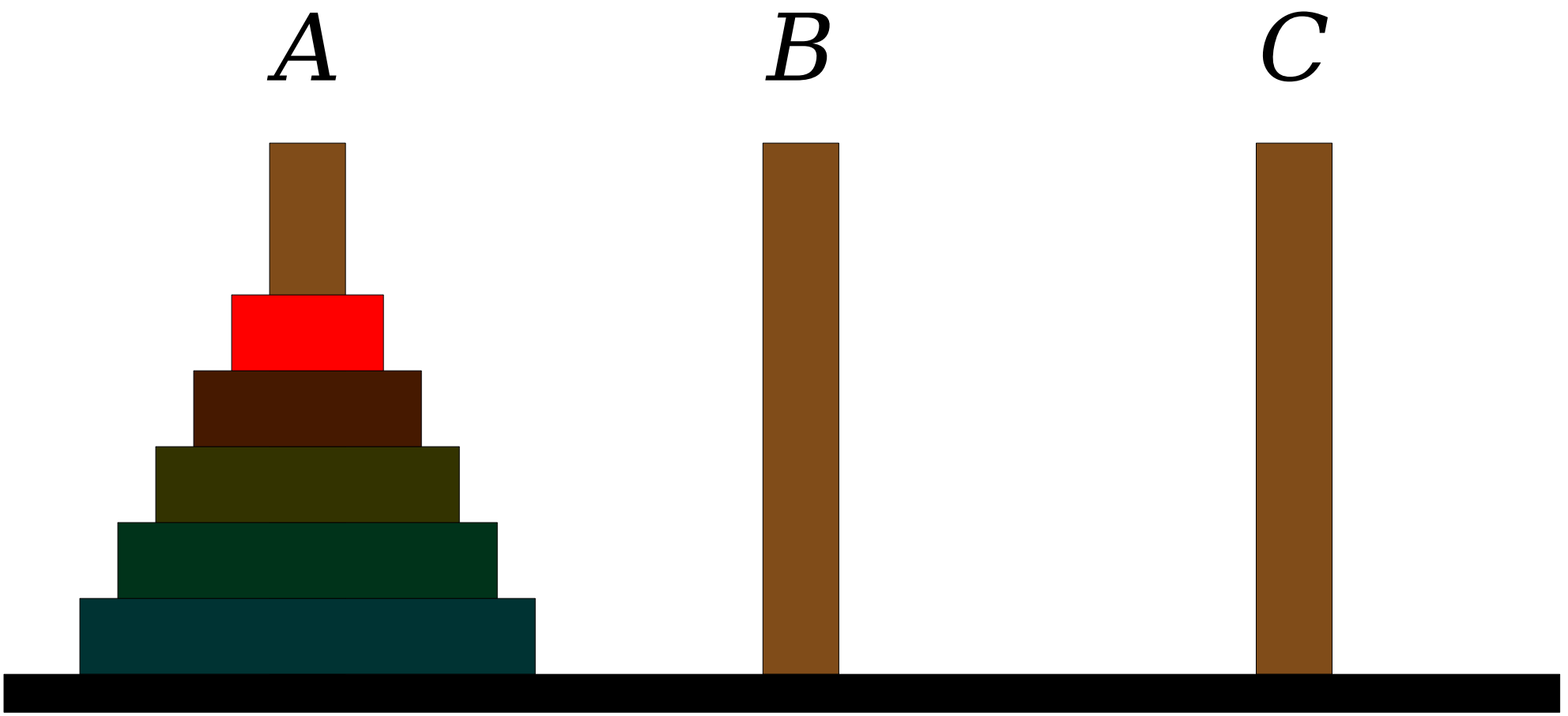


Step One: Move the smallest disk from Spindle A to Spindle C.
Step Two: Move the orange disk from Spindle A to Spindle B.
Step Three: Move the smallest disk from Spindle C to Spindle B.

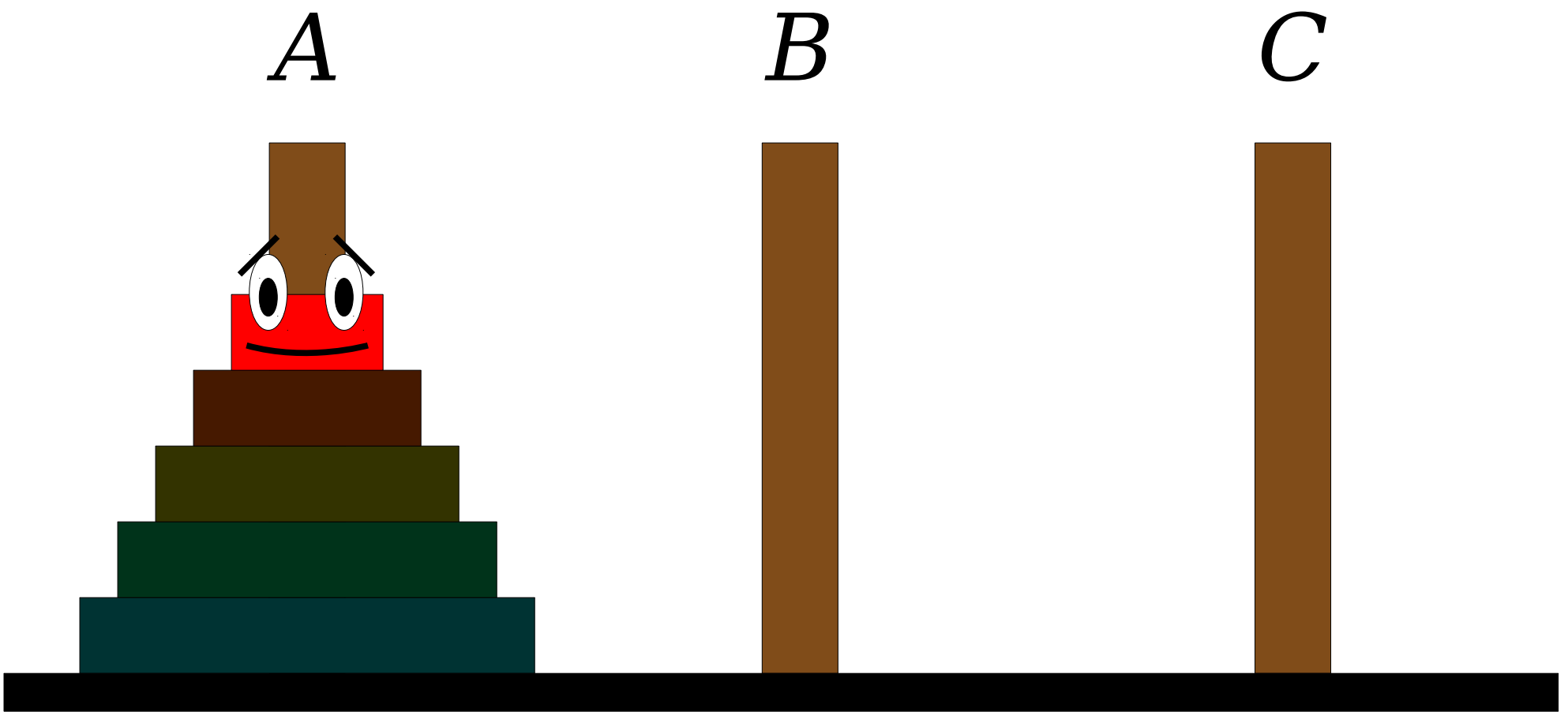
Solving the Towers of Hanoi



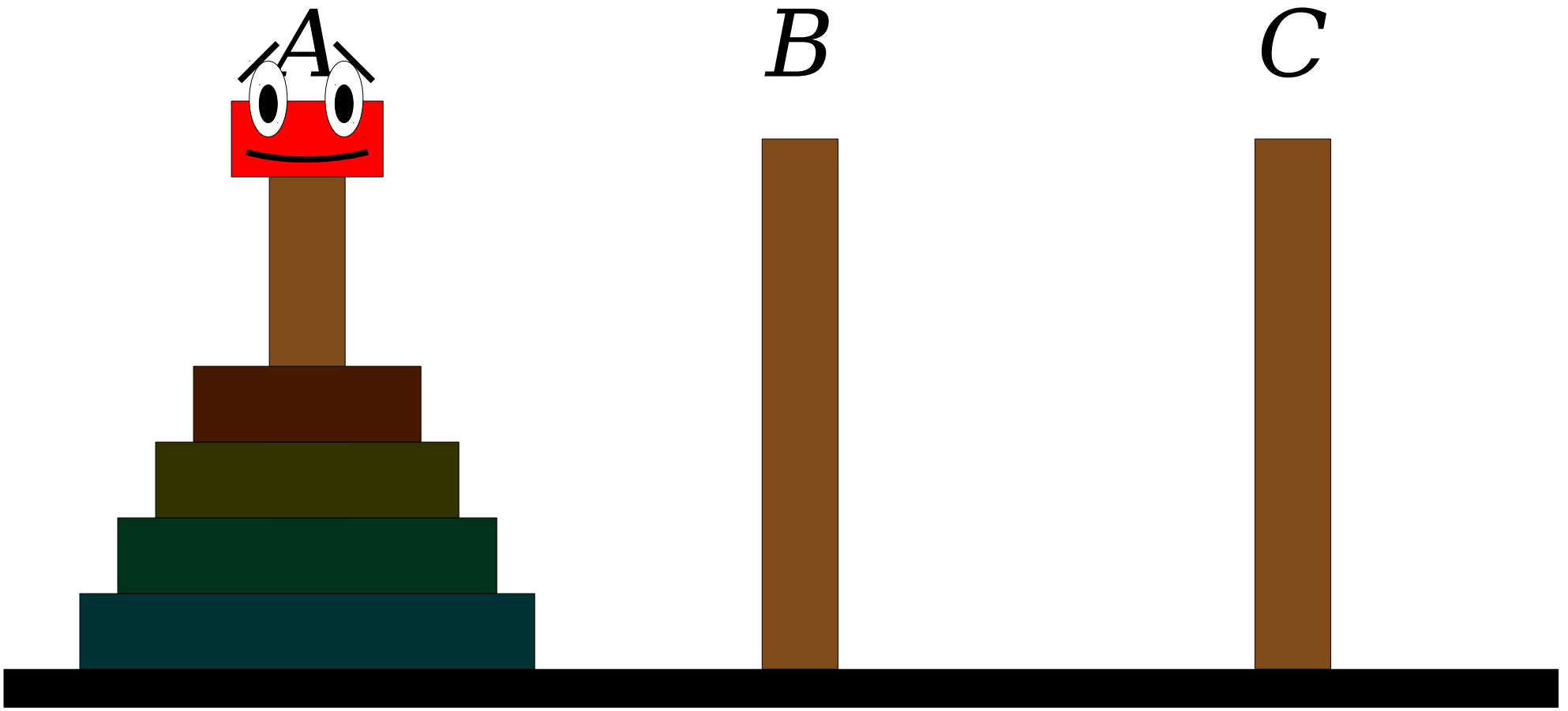
Solving the Towers of Hanoi



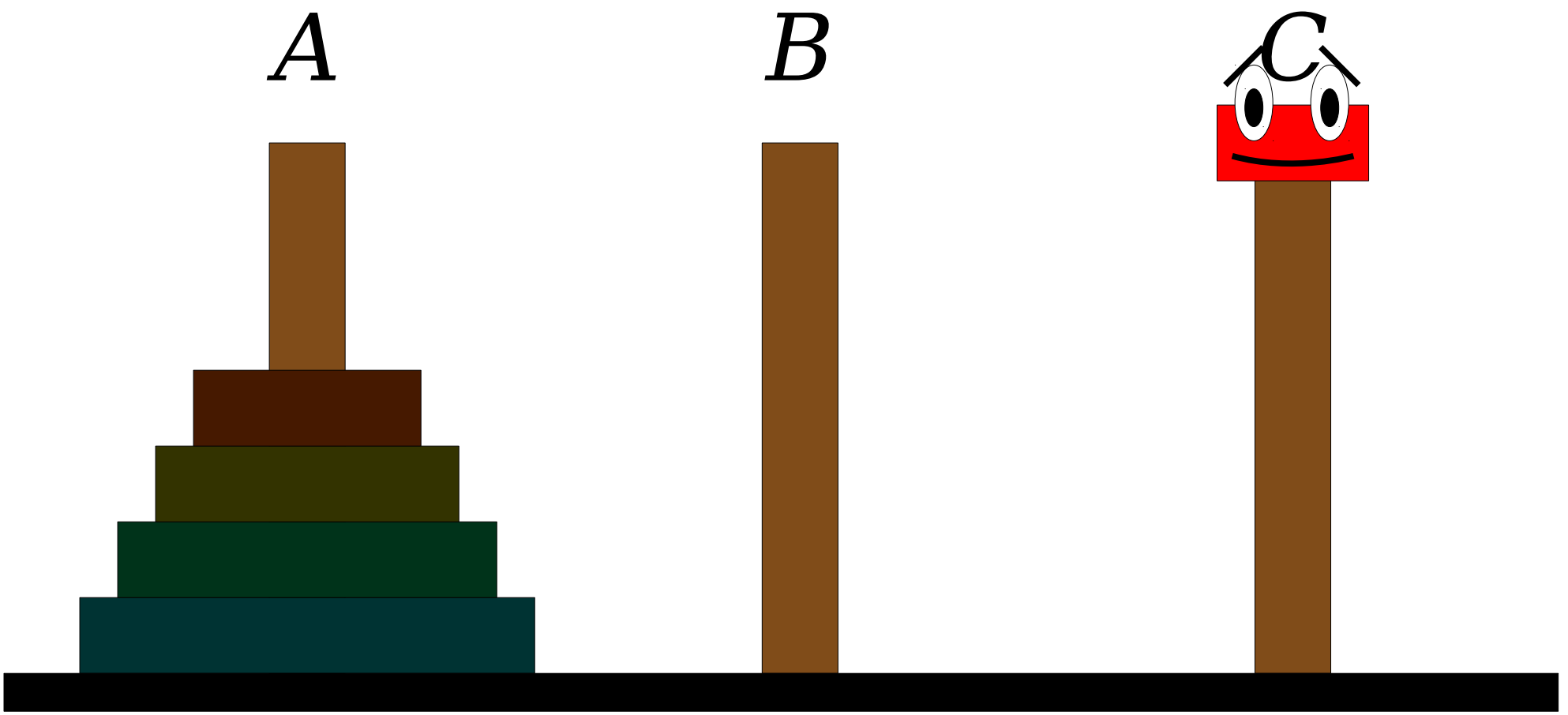
Solving the Towers of Hanoi



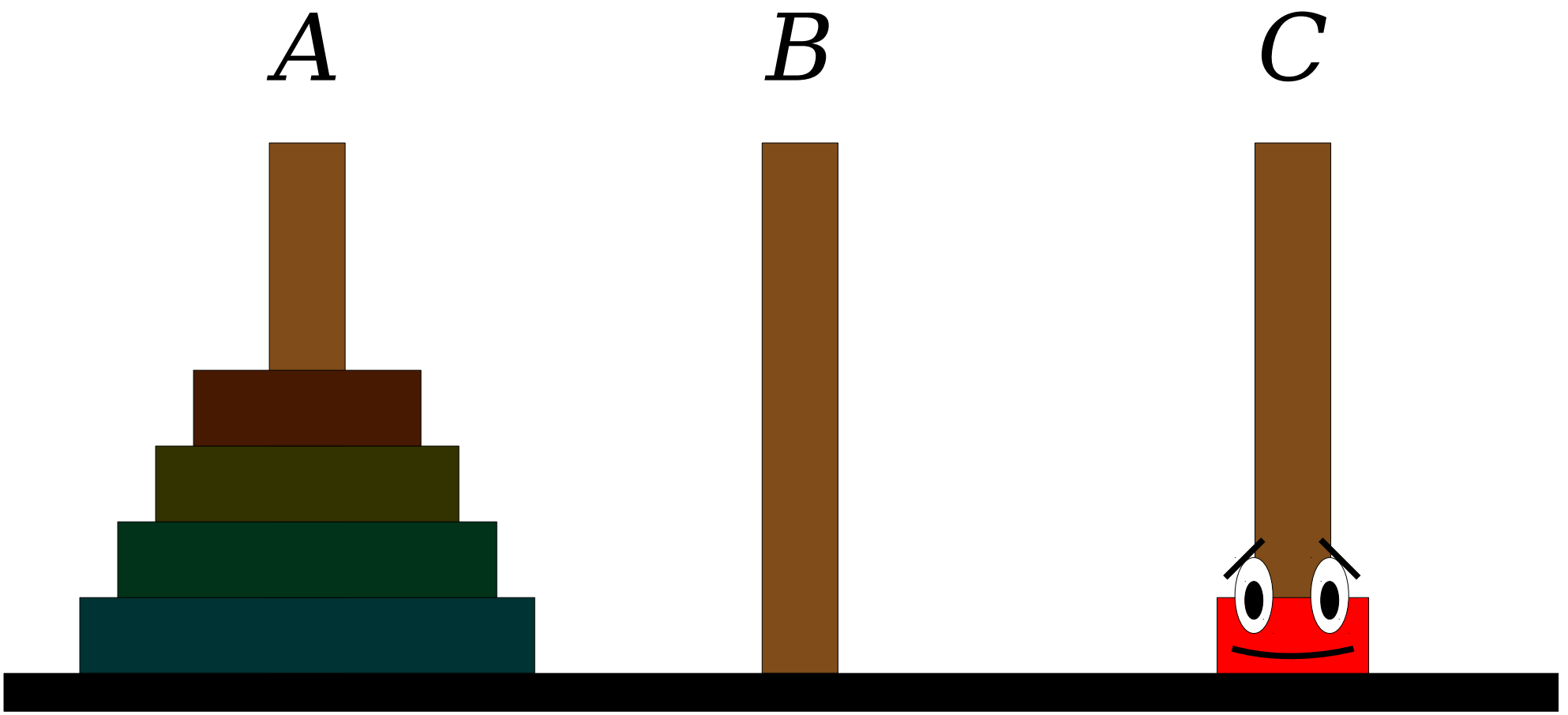
Solving the Towers of Hanoi



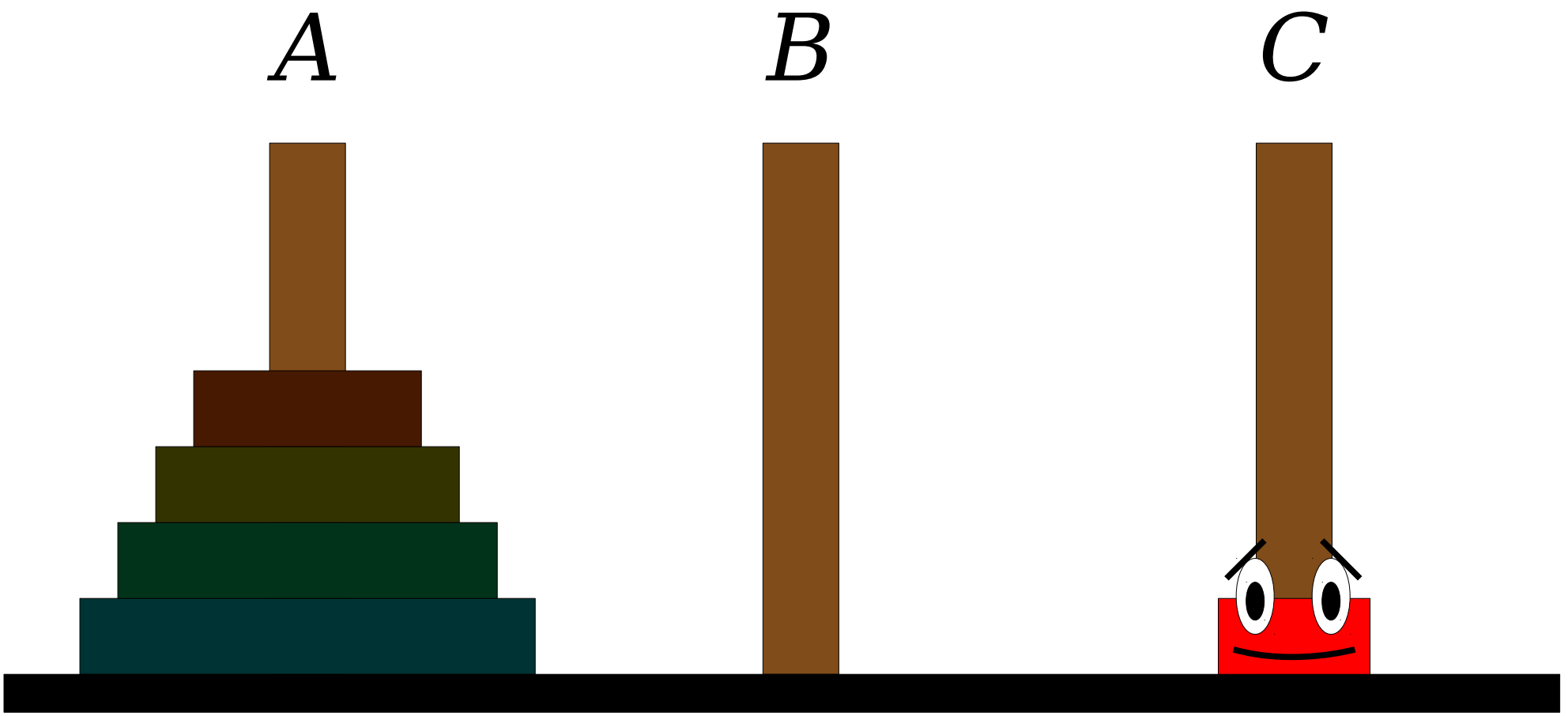
Solving the Towers of Hanoi



Solving the Towers of Hanoi



Solving the Towers of Hanoi



Only Step: Move the smallest disk from Spindle *A* to Spindle *C*.

To move the 5-disk tower from Spindle A to Spindle C:

Step One: Move the four smaller disks from Spindle A to Spindle B.
Step Two: Move the blue disk from Spindle A to Spindle C.
Step Three: Move the four smaller disks from Spindle B to Spindle C.

To move the 4-disk tower from Spindle A to Spindle B:

Step One: Move the three smaller disks from Spindle A to Spindle C.
Step Two: Move the green disk from Spindle A to Spindle B.
Step Three: Move the three smaller disks from Spindle C to Spindle B.

To move the 3-disk tower from Spindle A to Spindle C:

Step One: Move the two smaller disks from Spindle A to Spindle B.
Step Two: Move the yellow disk from Spindle A to Spindle C.
Step Three: Move the two smaller disks from Spindle B to Spindle C.

To move the 2-disk tower from Spindle A to Spindle B:

Step One: Move the smallest disk from Spindle A to Spindle C.
Step Two: Move the orange disk from Spindle A to Spindle B.
Step Three: Move the smallest disk from Spindle C to Spindle B.

To move the 1-disk tower from Spindle A to Spindle C:

Only Step: Move the smallest disk from Spindle A to Spindle C.

To move the 1-disk tower from Spindle A to Spindle C:

Only Step: Move the smallest disk from Spindle A to Spindle C.

To move an n-disk tower from Spindle A to Spindle C:

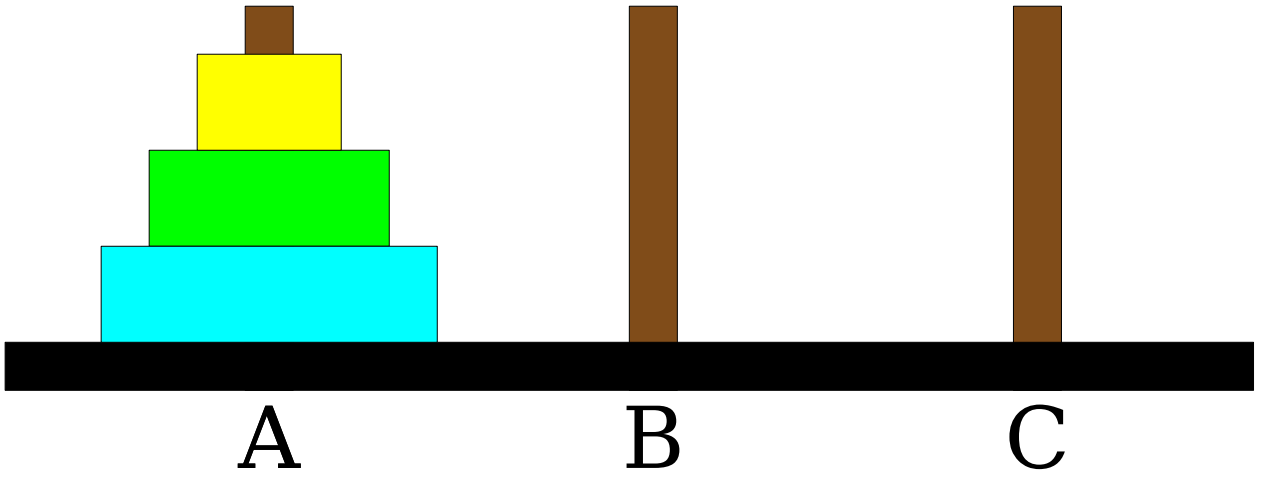
Step One: Move the (n-1) smaller disks from Spindle A to Spindle B.

Step Two: Move the nth disk from Spindle A to Spindle C.

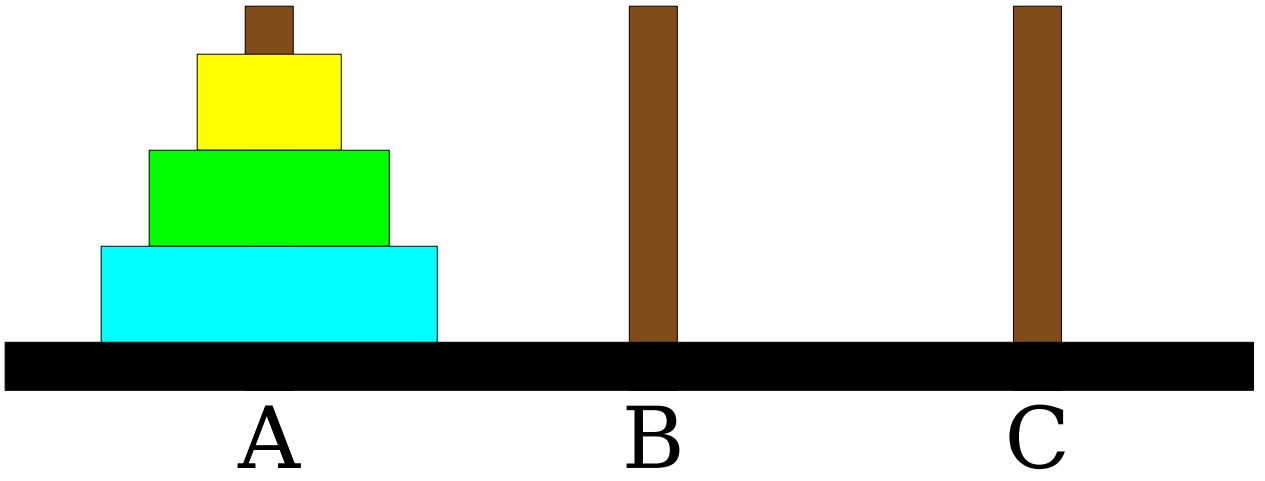
Step Three: Move the (n-1) smaller disks from Spindle B to Spindle C.

```
void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n - 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n - 1, temp, to, from);
    }
}
```

```
int main() {
    moveTower(3, 'a', 'c', 'b');
    return 0;
}
```



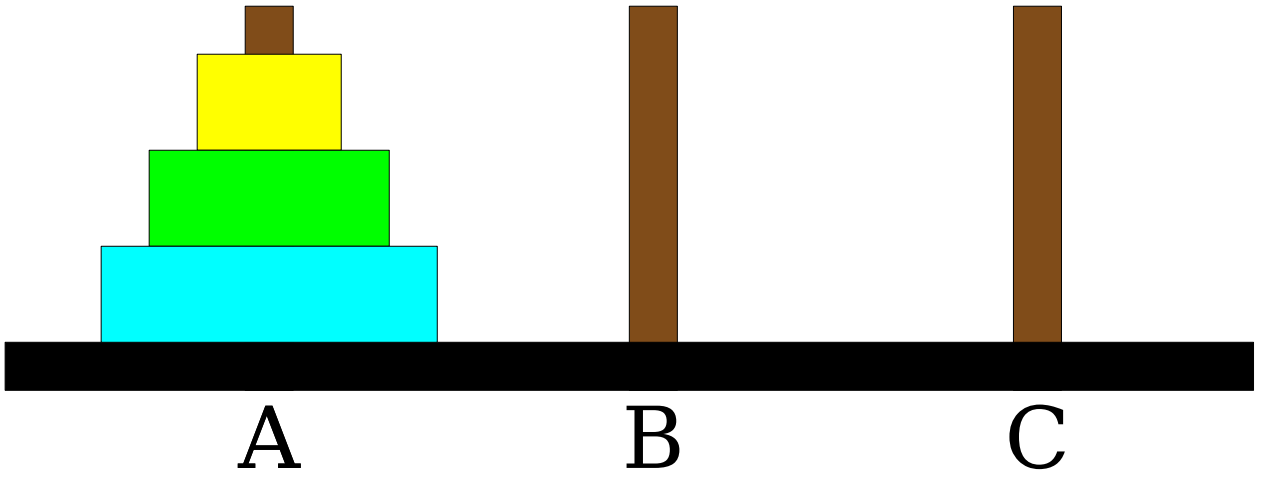
```
int main() {
    moveTower(3, 'a', 'c', 'b');
    return 0;
}
```



```
int main() {
}
```

```
void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n – 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n – 1, temp, to, from);
    }
}
```

n	3	from	a	to	c	temp	b

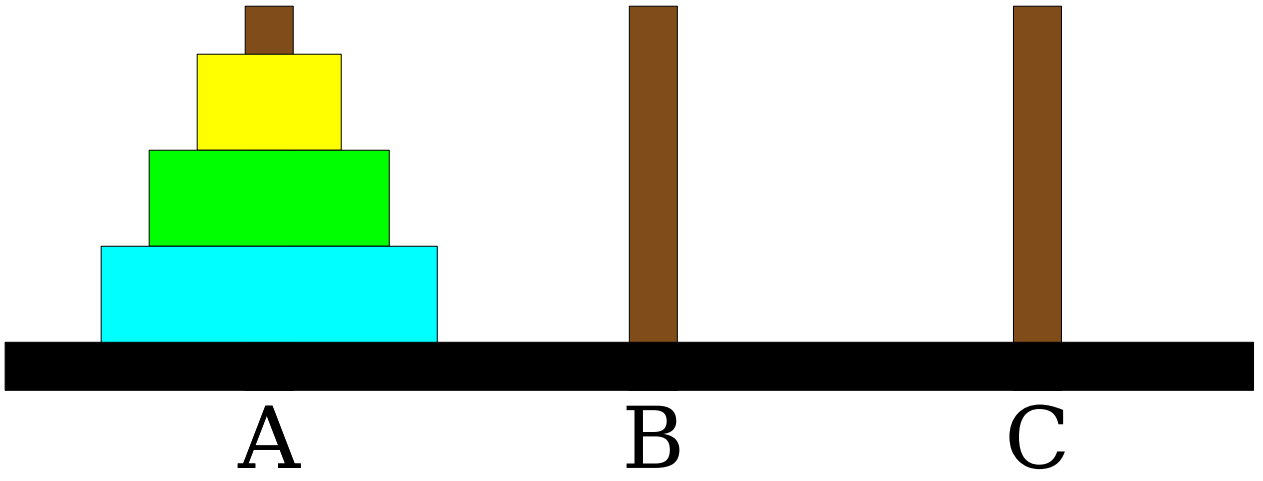


```
int main() {

}
```

```
void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n – 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n – 1, temp, to, from);
    }
}
```

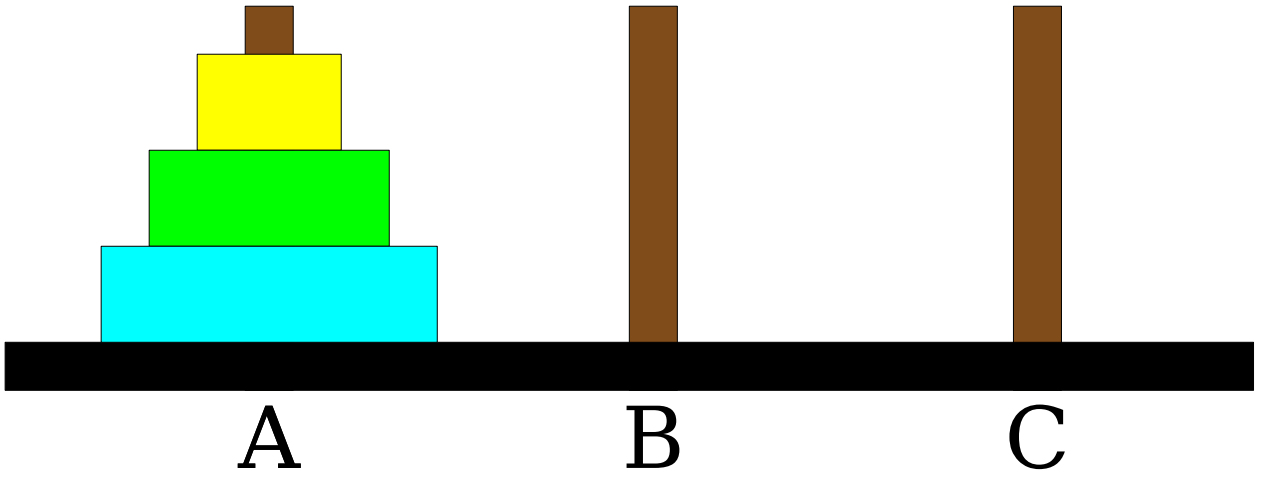
n	3	from	a	to	c	temp	b



```
int main() {
}
```

```
void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n – 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n – 1, temp, to, from);
    }
}
```

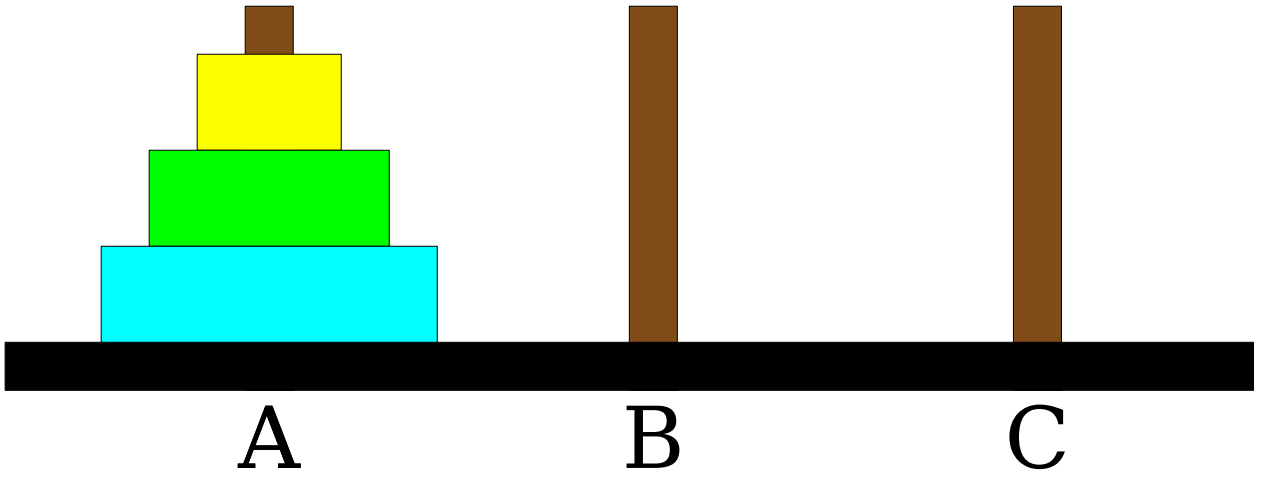
n	3	from	a	to	c	temp	b



```
int main() {
}
```

```
void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n – 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n – 1, temp, to, from);
    }
}
```

n	3	from	a	to	c	temp	b

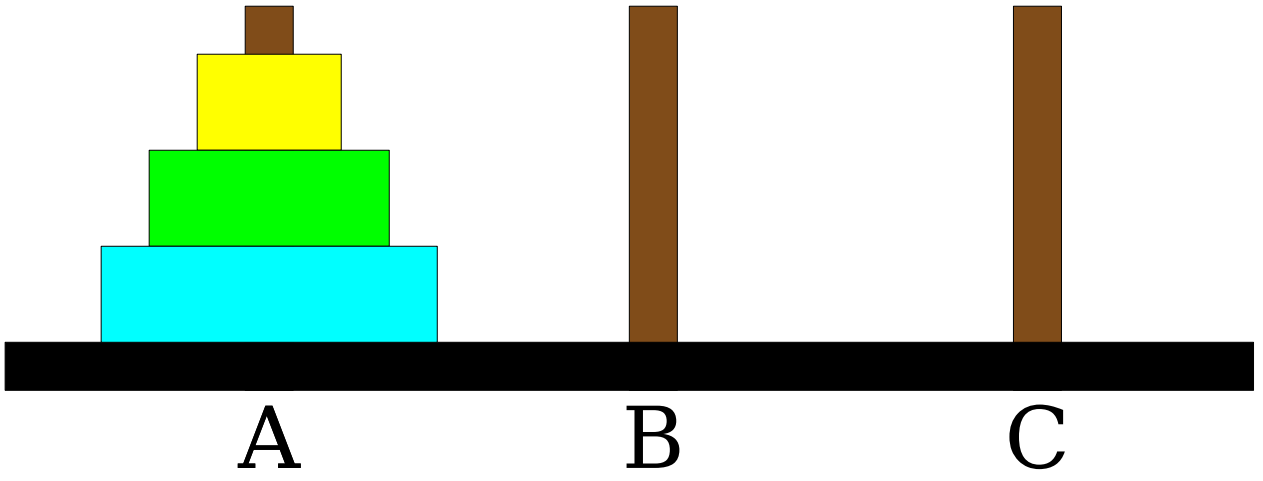


```
int main() {
}
```

```
void moveTower(int n, char from, char to, char temp) {
}
```

```
void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n – 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n – 1, temp, to, from);
    }
}
```

n	2	from	a	to	b	temp	c

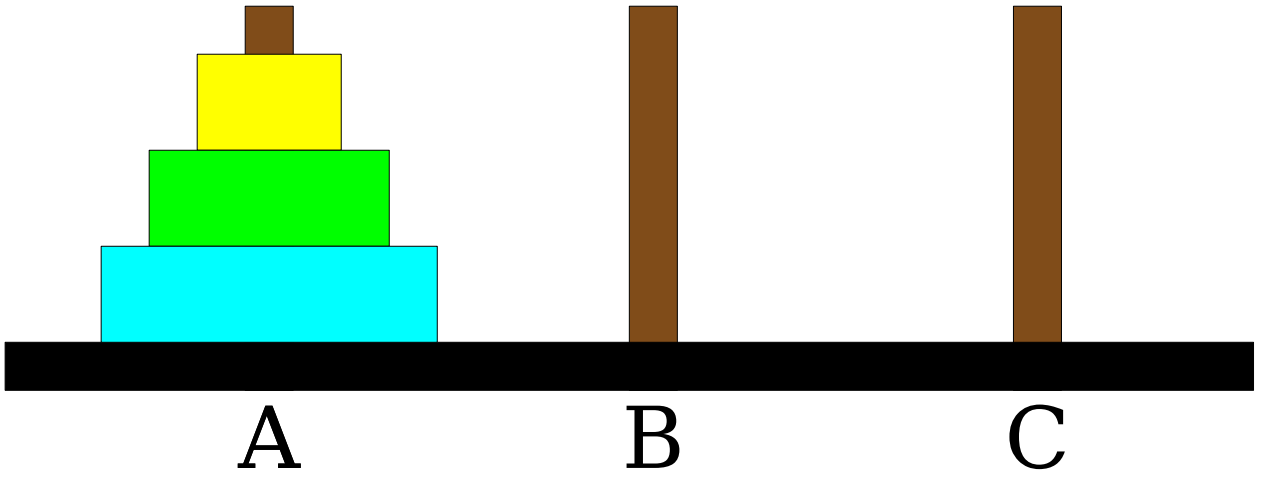


```
int main() {
}
```

```
void moveTower(int n, char from, char to, char temp) {
}
```

```
void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n – 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n – 1, temp, to, from);
    }
}
```

n	2	from	a	to	b	temp	c

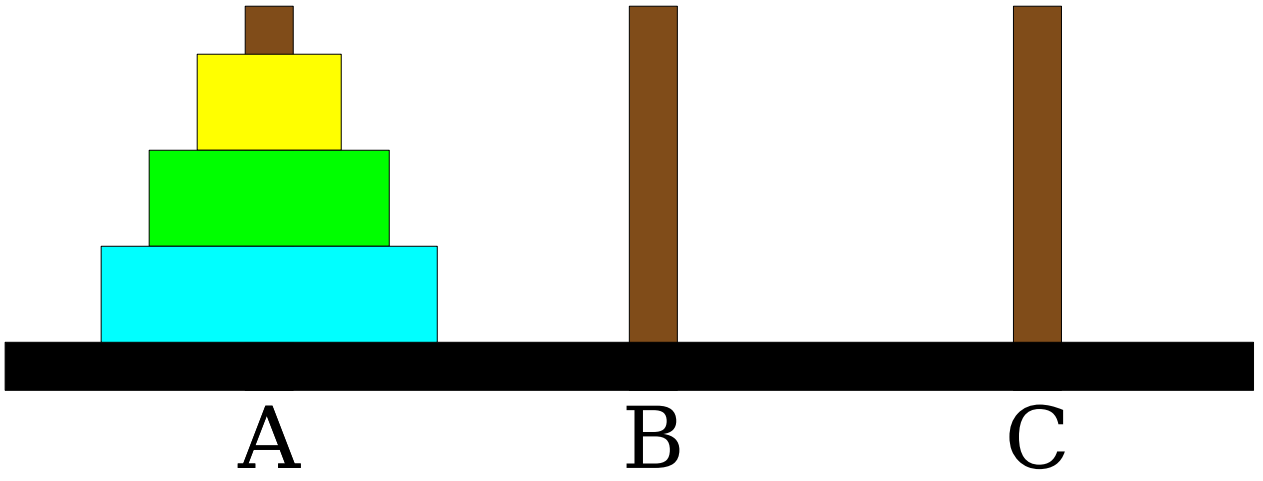


```
int main() {
}
```

```
void moveTower(int n, char from, char to, char temp) {
}
```

```
void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n – 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n – 1, temp, to, from);
    }
}
```

n	2	from	a	to	b	temp	c

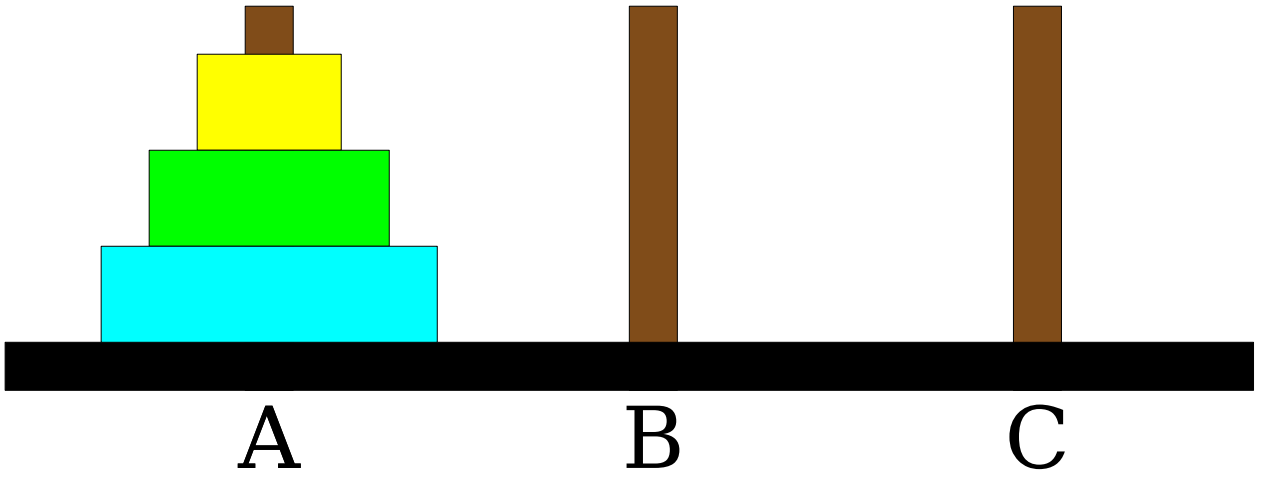


```
int main() {
}
```

```
void moveTower(int n, char from, char to, char temp) {
}
```

```
void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n - 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n - 1, temp, to, from);
    }
}
```

n	2	from	a	to	b	temp	c



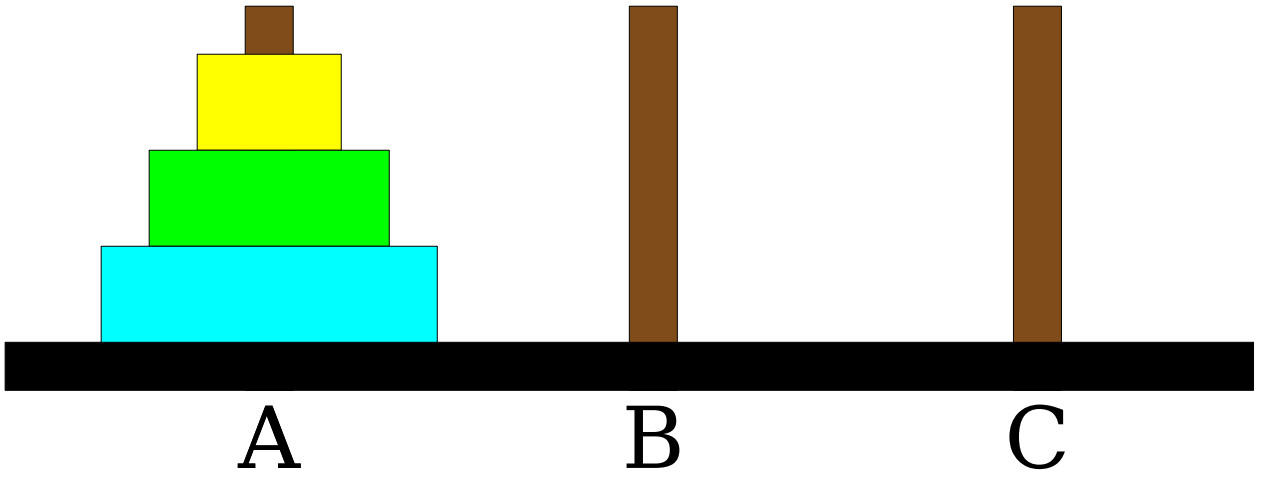
```
int main() {
}
```

```
void moveTower(int n, char from, char to, char temp) {
}
```

```
void moveTower(int n, char from, char to, char temp) {
}
```

```
void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n – 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n – 1, temp, to, from);
    }
}
```

n	1	from	a	to	c	temp	b



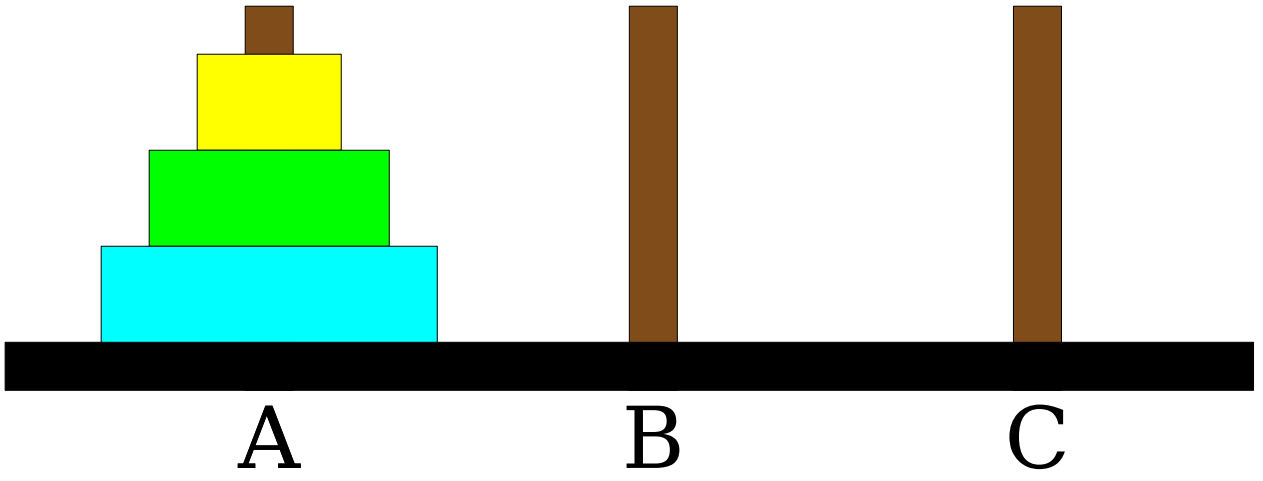
```
int main() {
}
```

```
void moveTower(int n, char from, char to, char temp) {
}
```

```
void moveTower(int n, char from, char to, char temp) {
}
```

```
void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n – 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n – 1, temp, to, from);
    }
}
```

n	1	from	a	to	c	temp	b



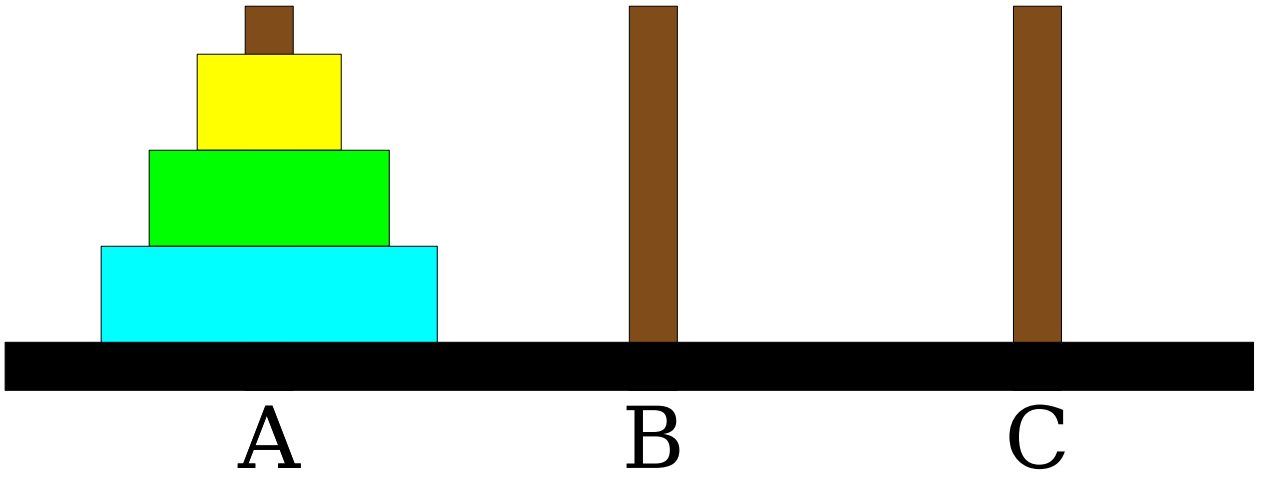
```
int main() {
}
```

```
void moveTower(int n, char from, char to, char temp) {
}
```

```
void moveTower(int n, char from, char to, char temp) {
}
```

```
void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n – 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n – 1, temp, to, from);
    }
}
```

n	1	from	a	to	c	temp	b



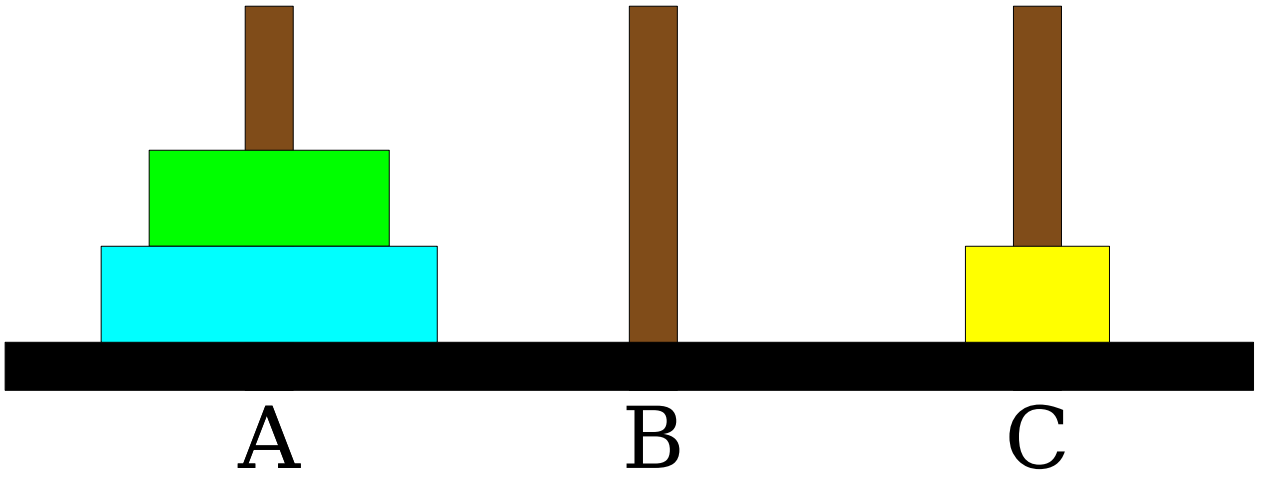
```
int main() {
}
```

```
void moveTower(int n, char from, char to, char temp) {
}
```

```
void moveTower(int n, char from, char to, char temp) {
}
```

```
void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n – 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n – 1, temp, to, from);
    }
}
```

n	1	from	a	to	c	temp	b

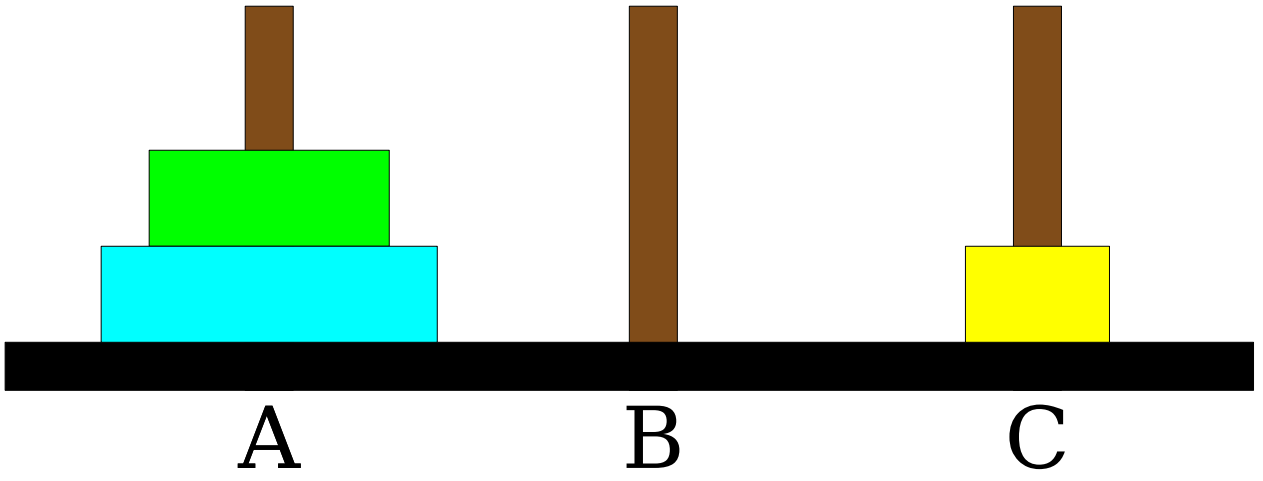


```
int main() {
}
```

```
void moveTower(int n, char from, char to, char temp) {
}
```

```
void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n - 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n - 1, temp, to, from);
    }
}
```

n	2	from	a	to	b	temp	c

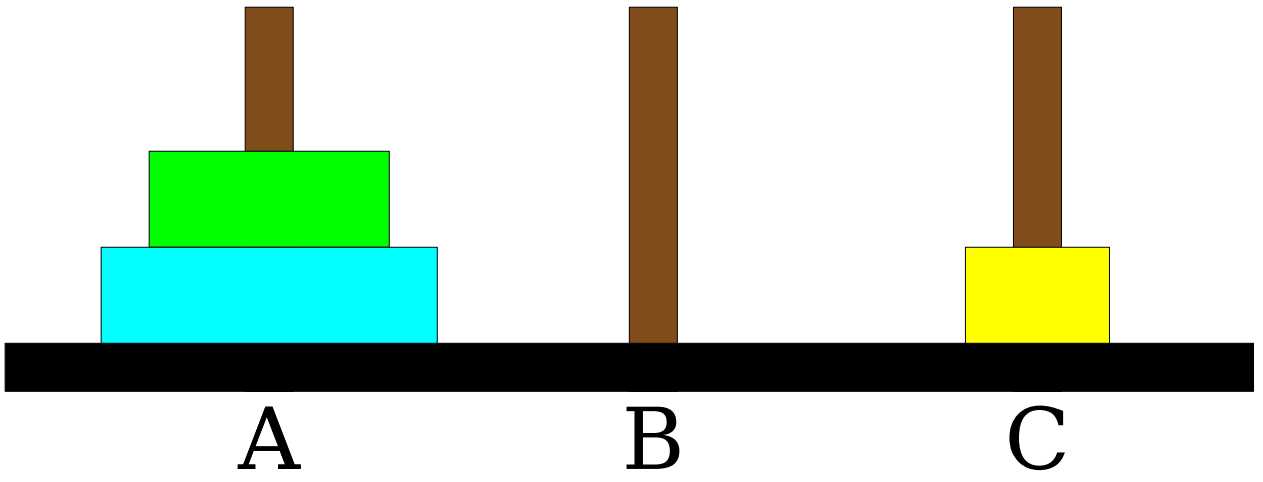


```
int main() {
}
```

```
void moveTower(int n, char from, char to, char temp) {
}
```

```
void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n – 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n – 1, temp, to, from);
    }
}
```

n	2	from	a	to	b	temp	c

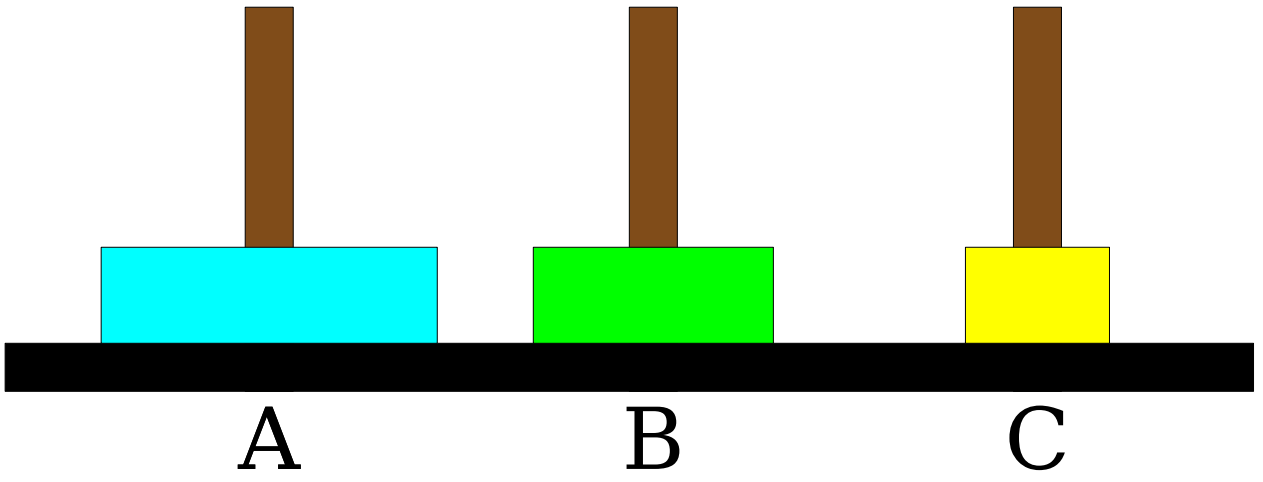


```
int main() {
}
```

```
void moveTower(int n, char from, char to, char temp) {
}
```

```
void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n – 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n – 1, temp, to, from);
    }
}
```

n	2	from	a	to	b	temp	c

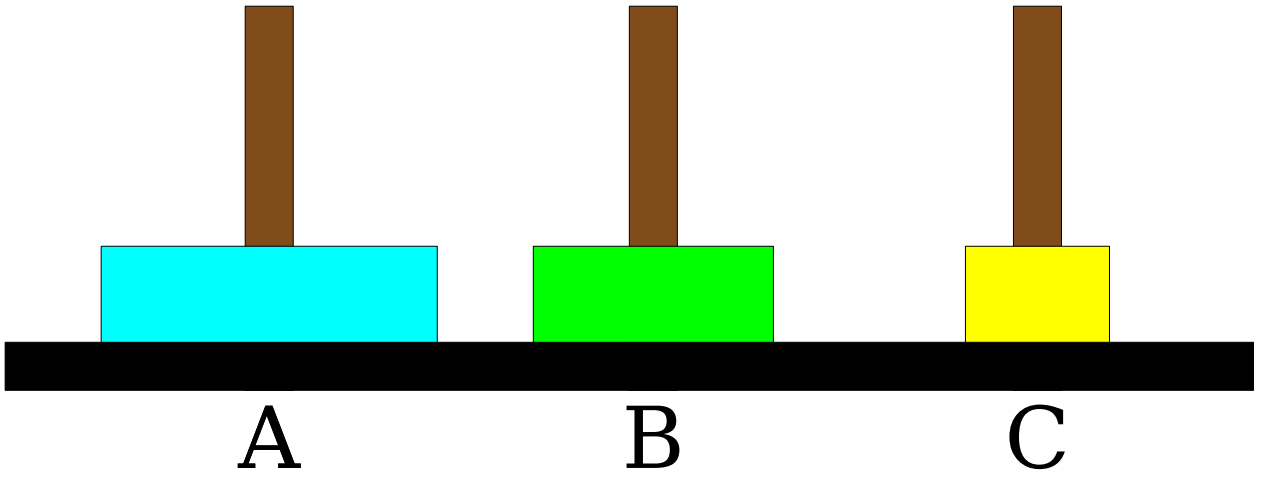


```
int main() {
}
```

```
void moveTower(int n, char from, char to, char temp) {
}
```

```
void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n – 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n – 1, temp, to, from);
    }
}
```

n	2	from	a	to	b	temp	c



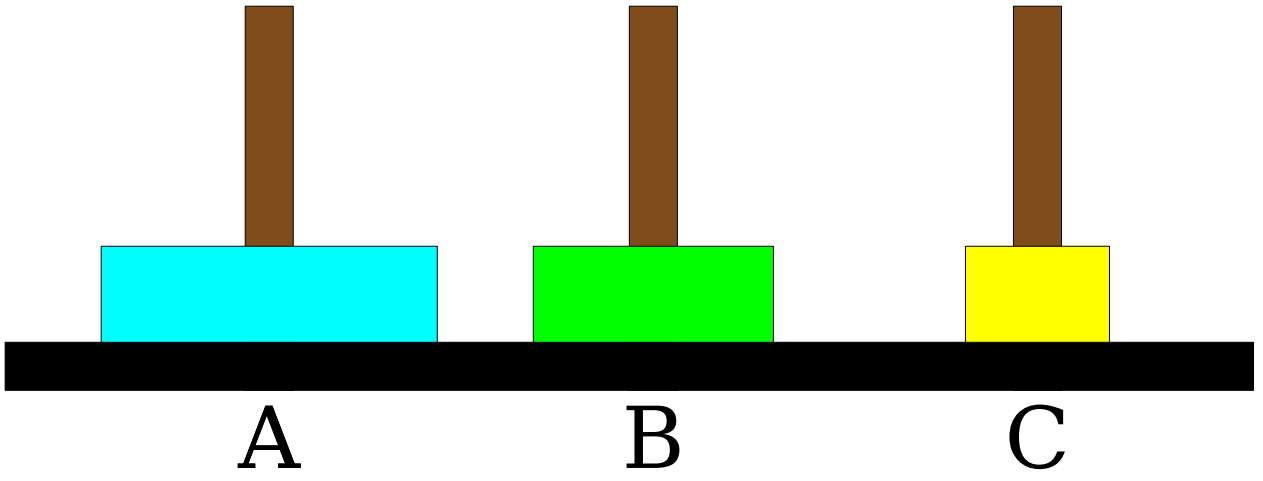
```
int main() {
}
```

```
void moveTower(int n, char from, char to, char temp) {
}
```

```
void moveTower(int n, char from, char to, char temp) {
}
```

```
void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n – 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n – 1, temp, to, from);
    }
}
```

n 1 from c to b temp a



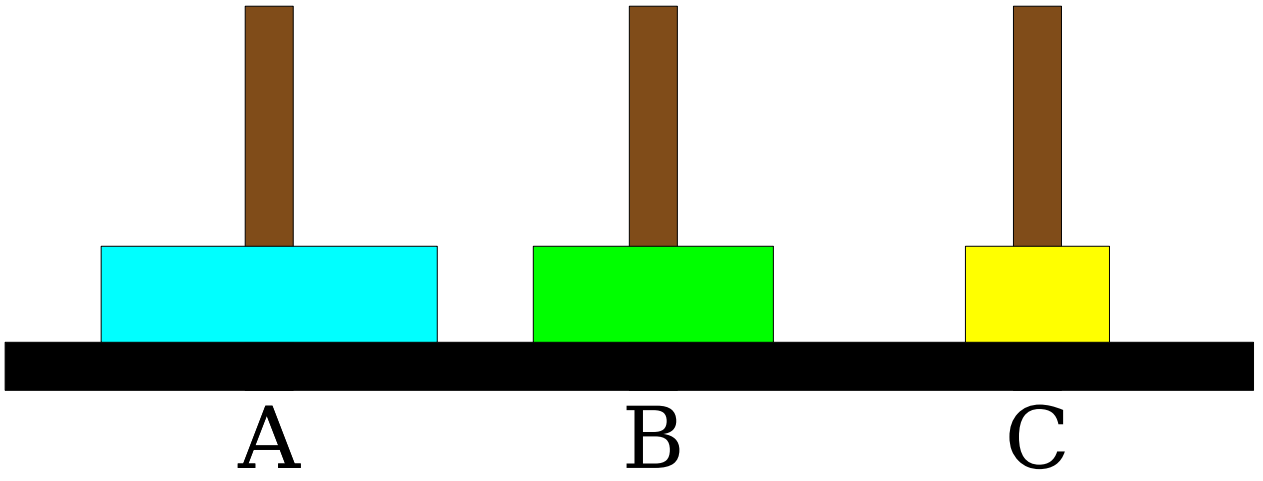
```
int main() {
}
```

```
void moveTower(int n, char from, char to, char temp) {
}
```

```
void moveTower(int n, char from, char to, char temp) {
}
```

```
void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n – 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n – 1, temp, to, from);
    }
}
```

n	1	from	c	to	b	temp	a



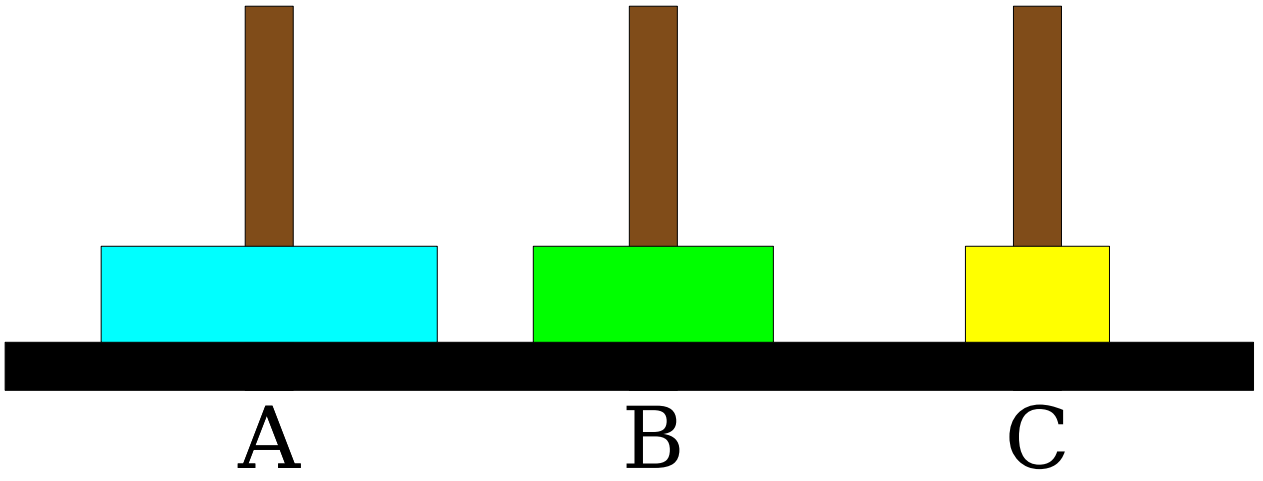
```
int main() {
}
```

```
void moveTower(int n, char from, char to, char temp) {
}
```

```
void moveTower(int n, char from, char to, char temp) {
}
```

```
void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n – 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n – 1, temp, to, from);
    }
}
```

n	1	from	c	to	b	temp	a



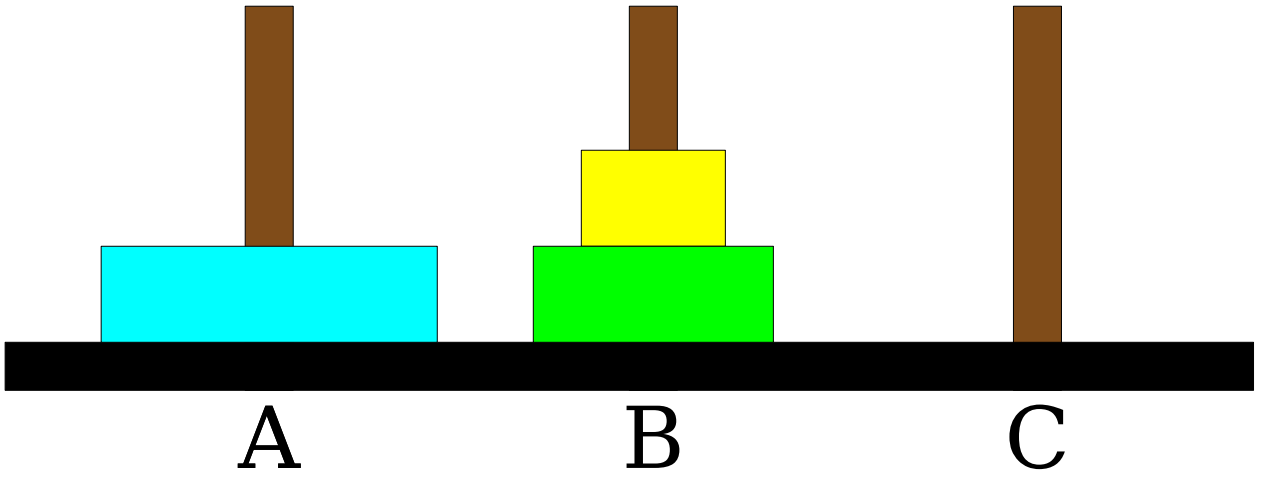
```
int main() {
}
```

```
void moveTower(int n, char from, char to, char temp) {
}
```

```
void moveTower(int n, char from, char to, char temp) {
}
```

```
void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n – 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n – 1, temp, to, from);
    }
}
```

n	1	from	c	to	b	temp	a

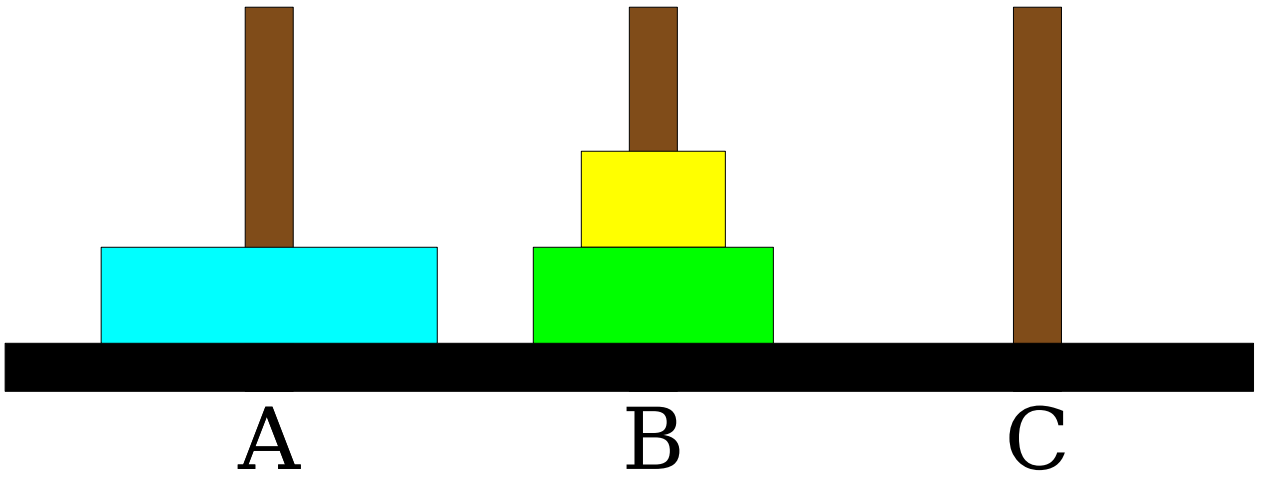


```
int main() {
}
```

```
void moveTower(int n, char from, char to, char temp) {
}
```

```
void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n – 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n – 1, temp, to, from);
    }
}
```

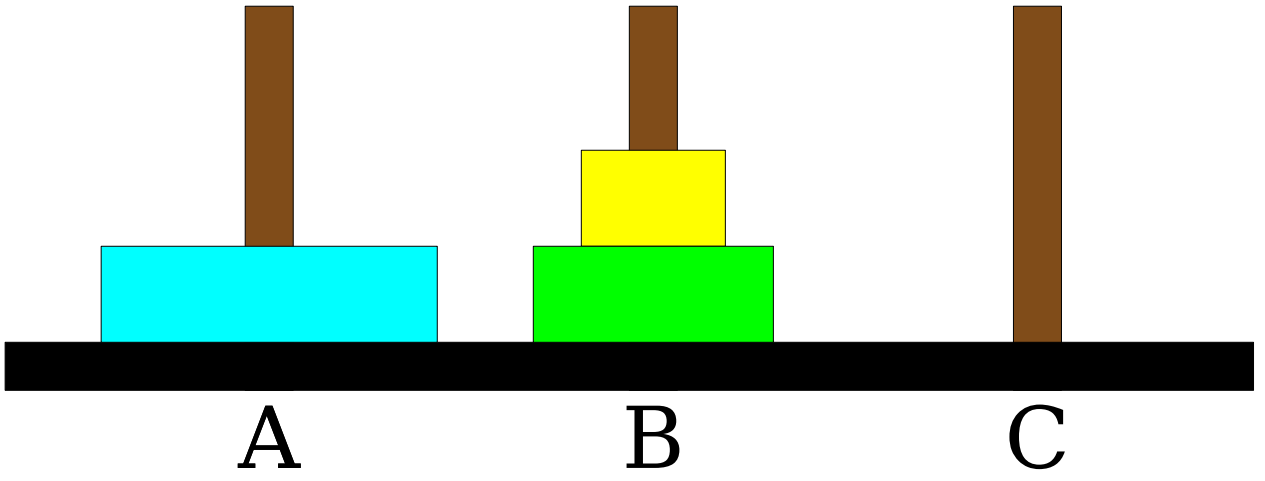
n	2	from	a	to	b	temp	c



```
int main() {
}
```

```
void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n – 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n – 1, temp, to, from);
    }
}
```

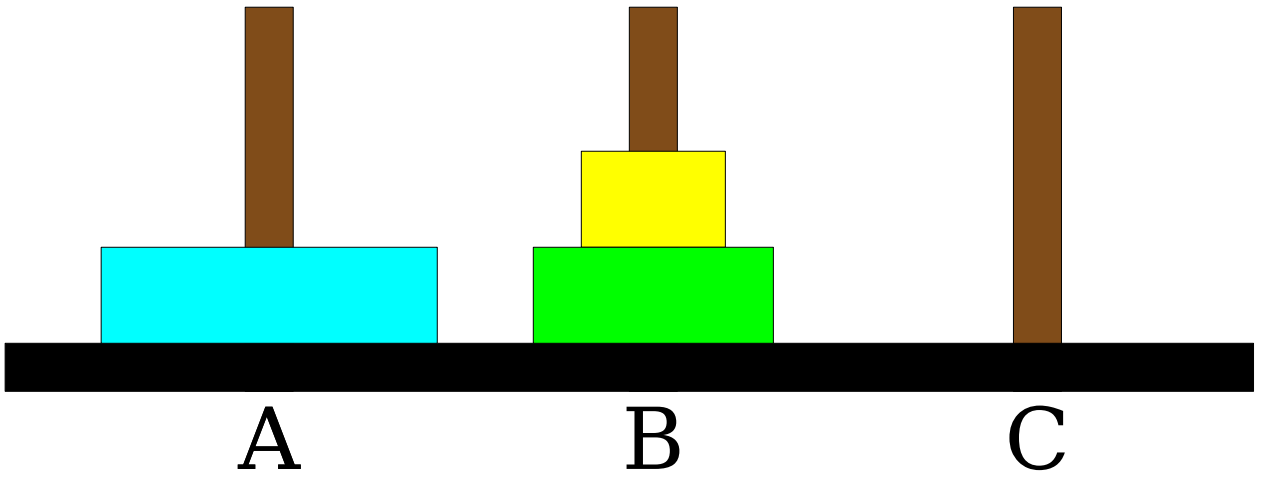
n	3	from	a	to	c	temp	b



```
int main() {
}
```

```
void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n – 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n – 1, temp, to, from);
    }
}
```

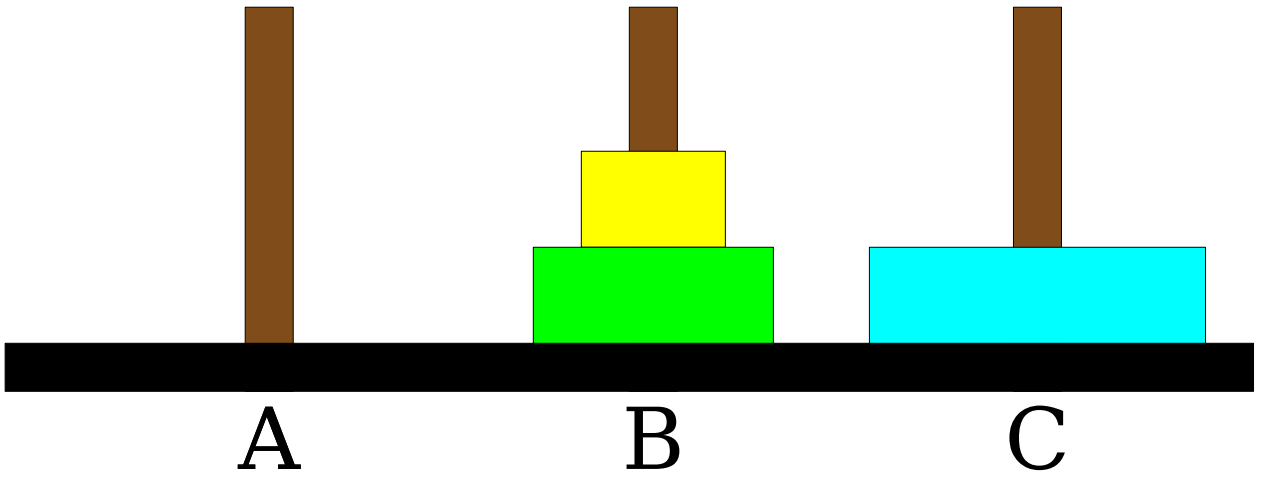
n	3	from	a	to	c	temp	b



```
int main() {
}
```

```
void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n – 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n – 1, temp, to, from);
    }
}
```

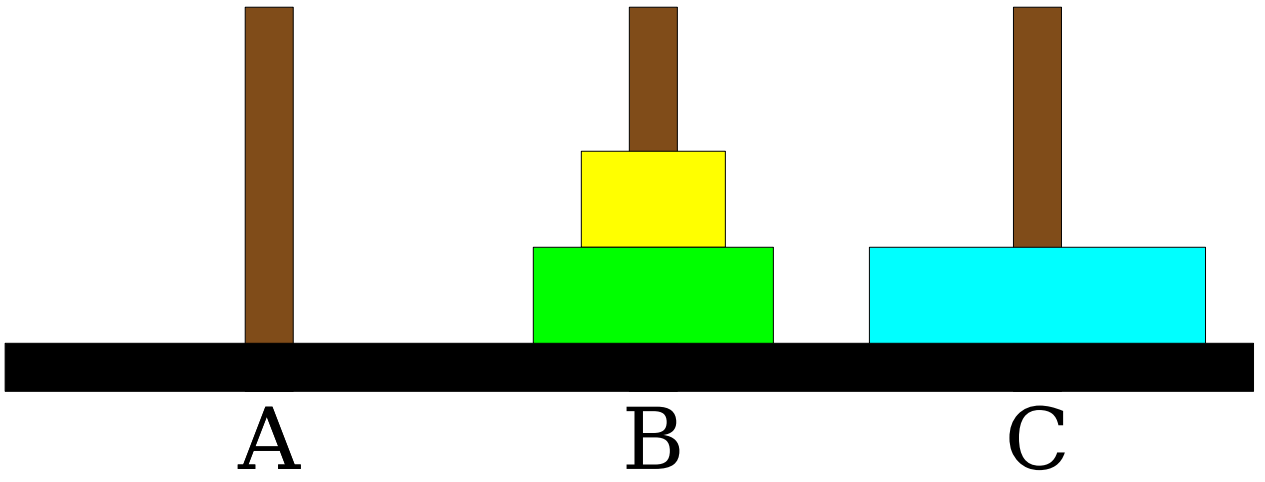
n	3	from	a	to	c	temp	b



```
int main() {
}
```

```
void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n – 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n – 1, temp, to, from);
    }
}
```

n	3	from	a	to	c	temp	b

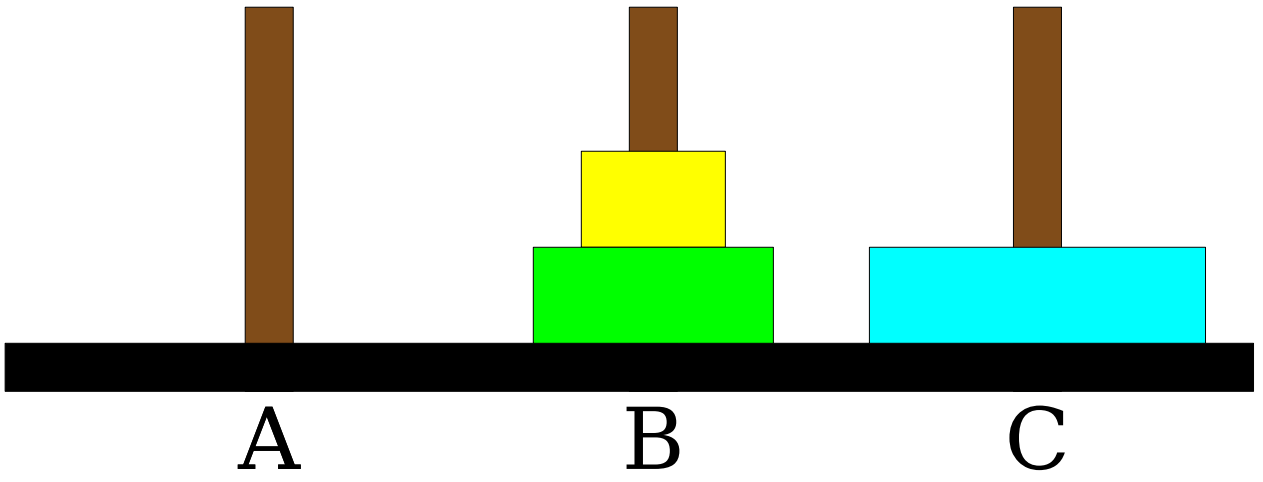


```
int main() {
}
```

```
void moveTower(int n, char from, char to, char temp) {
}
```

```
void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n – 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n – 1, temp, to, from);
    }
}
```

n	2	from	b	to	c	temp	a

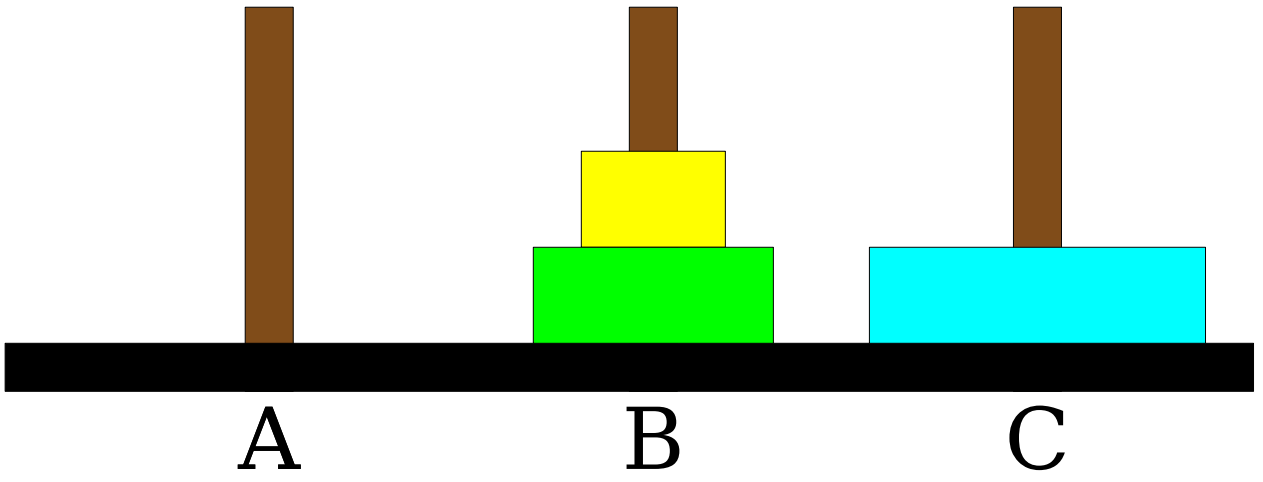


```
int main() {
}
```

```
void moveTower(int n, char from, char to, char temp) {
}
```

```
void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n – 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n – 1, temp, to, from);
    }
}
```

n	2	from	b	to	c	temp	a

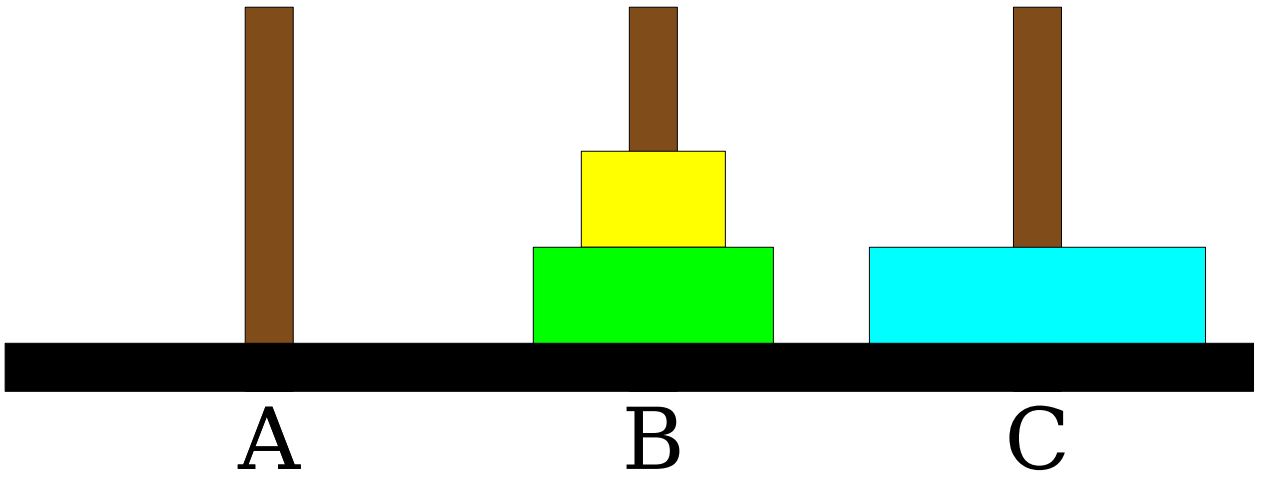


```
int main() {
}
```

```
void moveTower(int n, char from, char to, char temp) {
}
```

```
void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n – 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n – 1, temp, to, from);
    }
}
```

n	2	from	b	to	c	temp	a

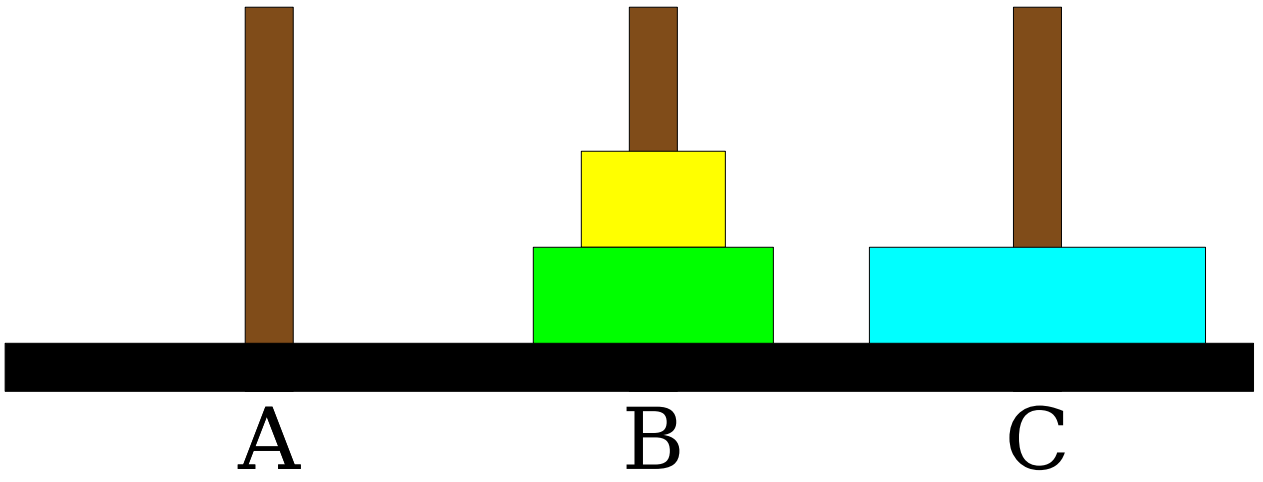


```
int main() {
```

```
void moveTower(int n, char from, char to, char temp) {
}
```

```
void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n - 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n - 1, temp, to, from);
    }
}
```

n	2	from	b	to	c	temp	a



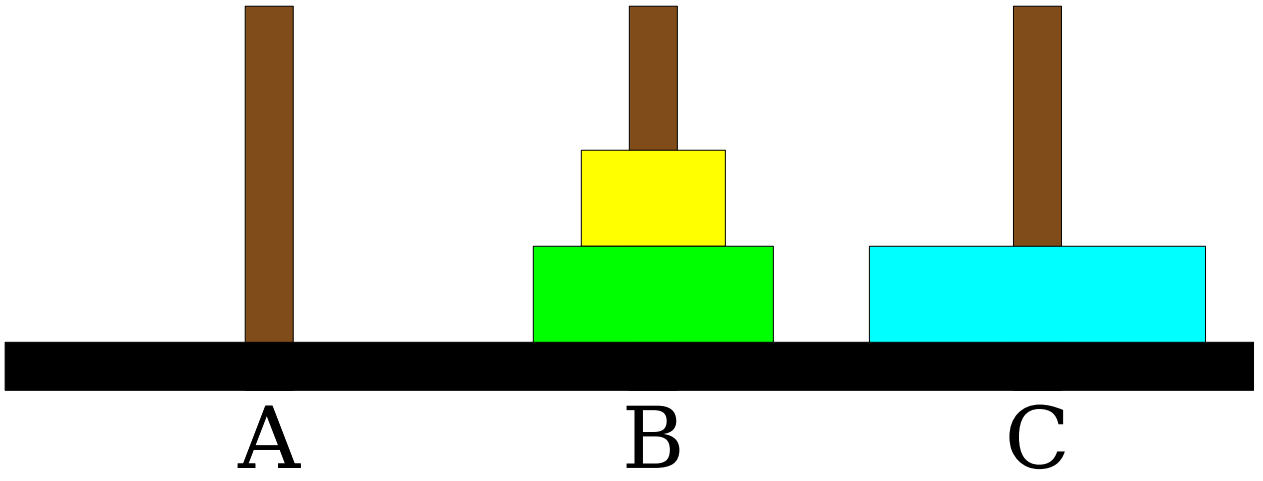
```
int main() {
}
```

```
void moveTower(int n, char from, char to, char temp) {
}
```

```
void moveTower(int n, char from, char to, char temp) {
}
```

```
void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n – 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n – 1, temp, to, from);
    }
}
```

n 1 from b to a temp c



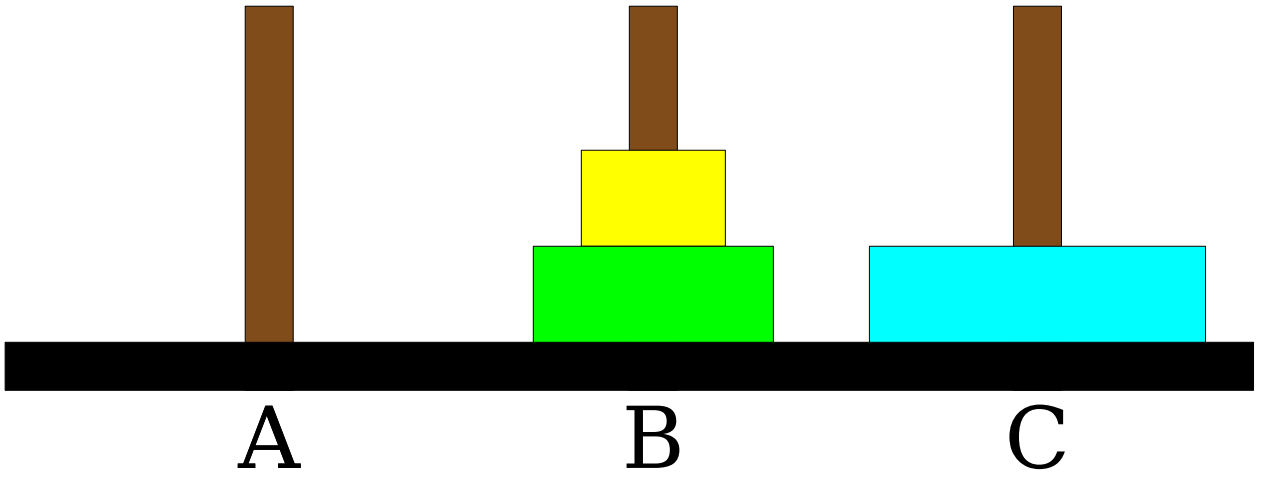
```
int main() {
}
```

```
void moveTower(int n, char from, char to, char temp) {
}
```

```
void moveTower(int n, char from, char to, char temp) {
}
```

```
void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n – 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n – 1, temp, to, from);
    }
}
```

n	1	from	b	to	a	temp	c



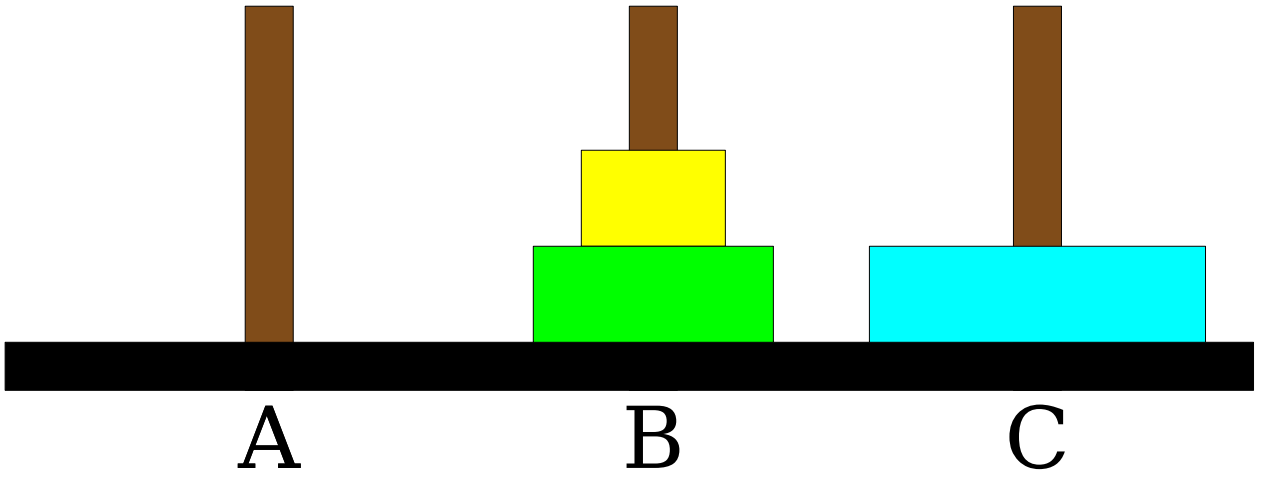
```
int main() {
}
```

```
void moveTower(int n, char from, char to, char temp) {
}
```

```
void moveTower(int n, char from, char to, char temp) {
}
```

```
void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n – 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n – 1, temp, to, from);
    }
}
```

n	1	from	b	to	a	temp	c



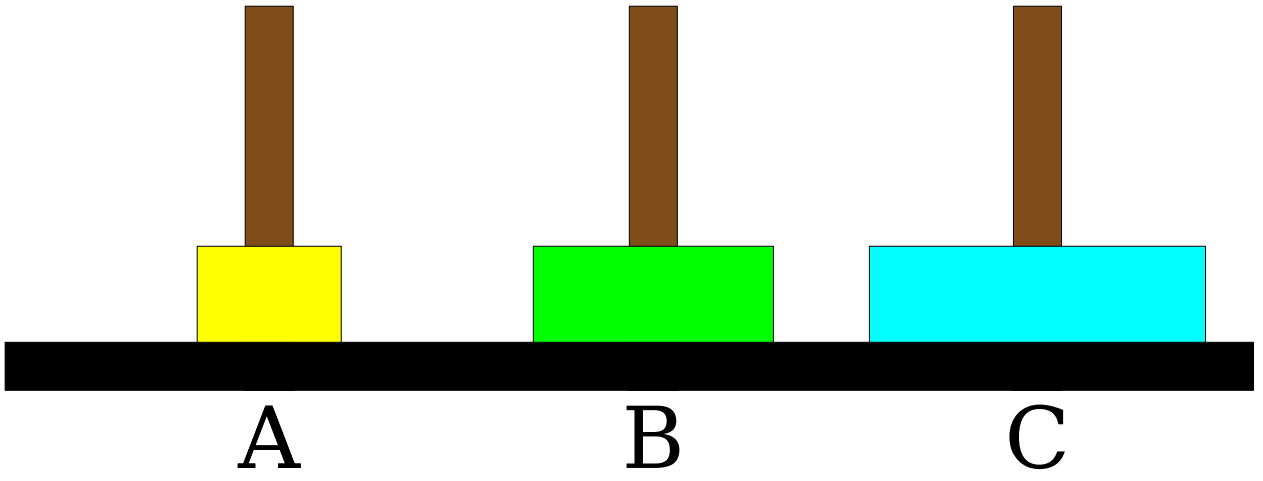
```
int main() {
}
```

```
void moveTower(int n, char from, char to, char temp) {
}
```

```
void moveTower(int n, char from, char to, char temp) {
}
```

```
void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n – 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n – 1, temp, to, from);
    }
}
```

n	1	from	b	to	a	temp	c

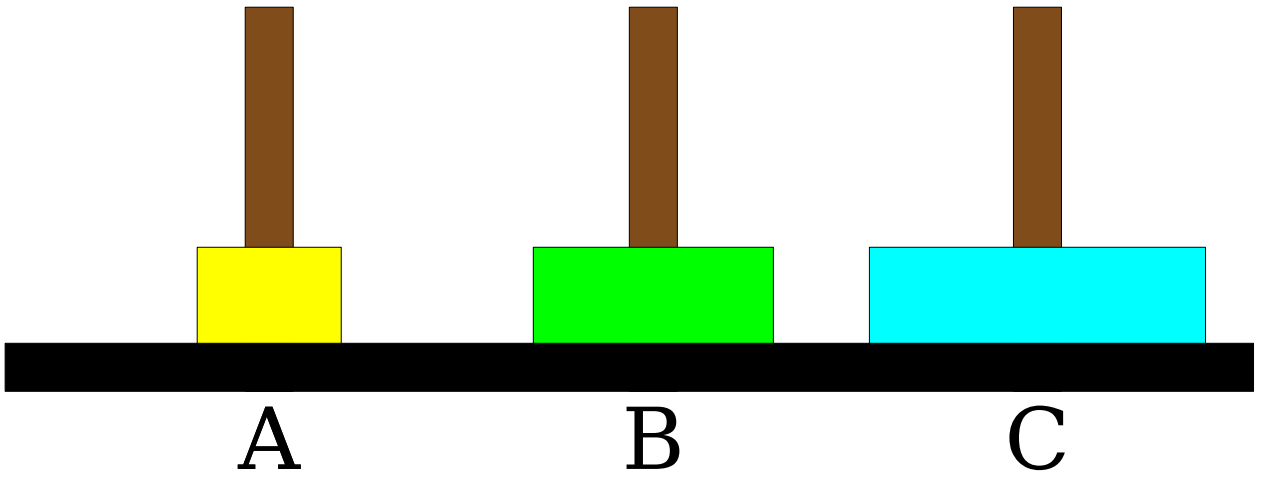


```
int main() {
```

```
void moveTower(int n, char from, char to, char temp) {
}
```

```
void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n - 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n - 1, temp, to, from);
    }
}
```

n	2	from	b	to	c	temp	a

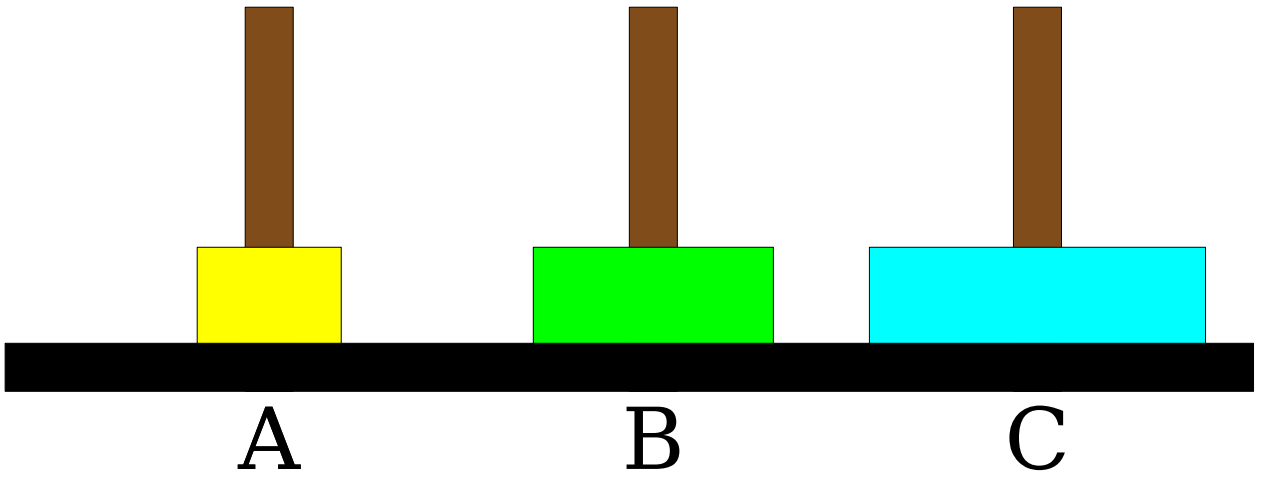


```
int main() {
}
```

```
void moveTower(int n, char from, char to, char temp) {
}
```

```
void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n – 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n – 1, temp, to, from);
    }
}
```

n	2	from	b	to	c	temp	a

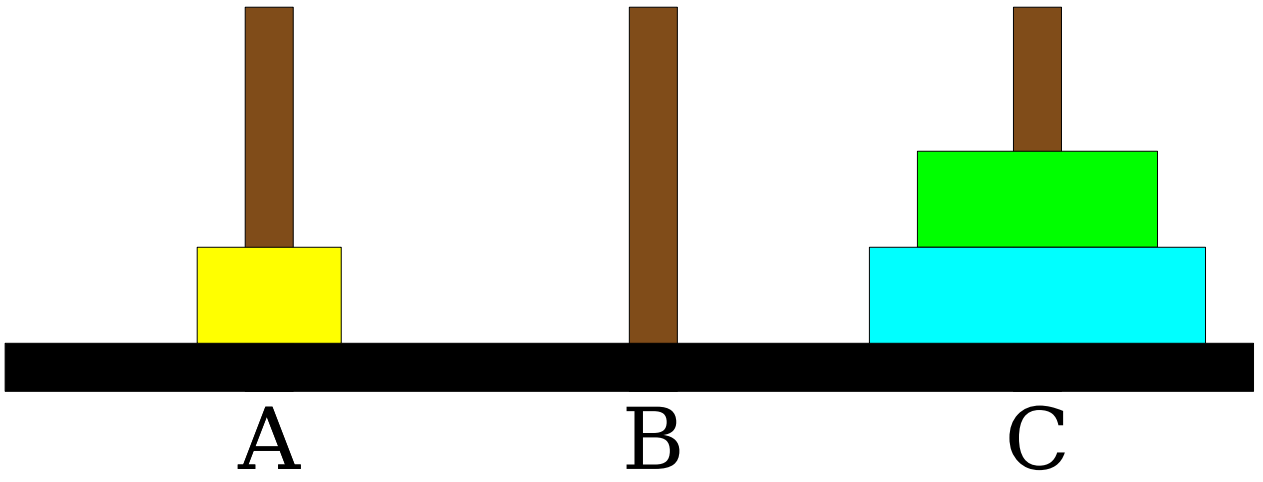


```
int main() {
```

```
void moveTower(int n, char from, char to, char temp) {
}
```

```
void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n – 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n – 1, temp, to, from);
    }
}
```

n	2	from	b	to	c	temp	a

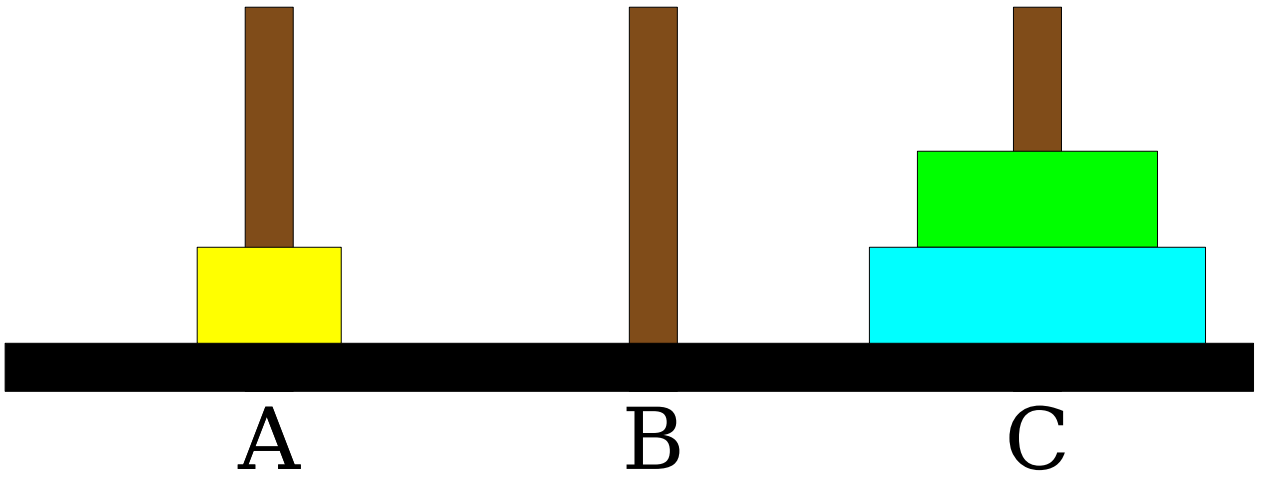


```
int main() {
```

```
void moveTower(int n, char from, char to, char temp) {
}
```

```
void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n – 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n – 1, temp, to, from);
    }
}
```

n	2	from	b	to	c	temp	a



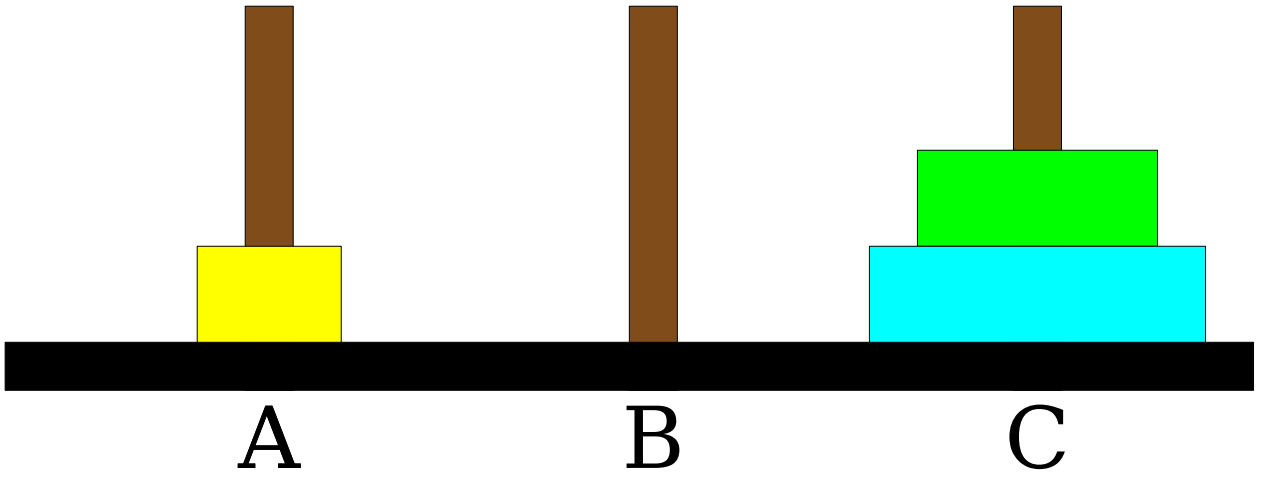
```
int main() {
}
```

```
void moveTower(int n, char from, char to, char temp) {
}
```

```
void moveTower(int n, char from, char to, char temp) {
}
```

```
void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n – 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n – 1, temp, to, from);
    }
}
```

n	1	from	a	to	c	temp	b



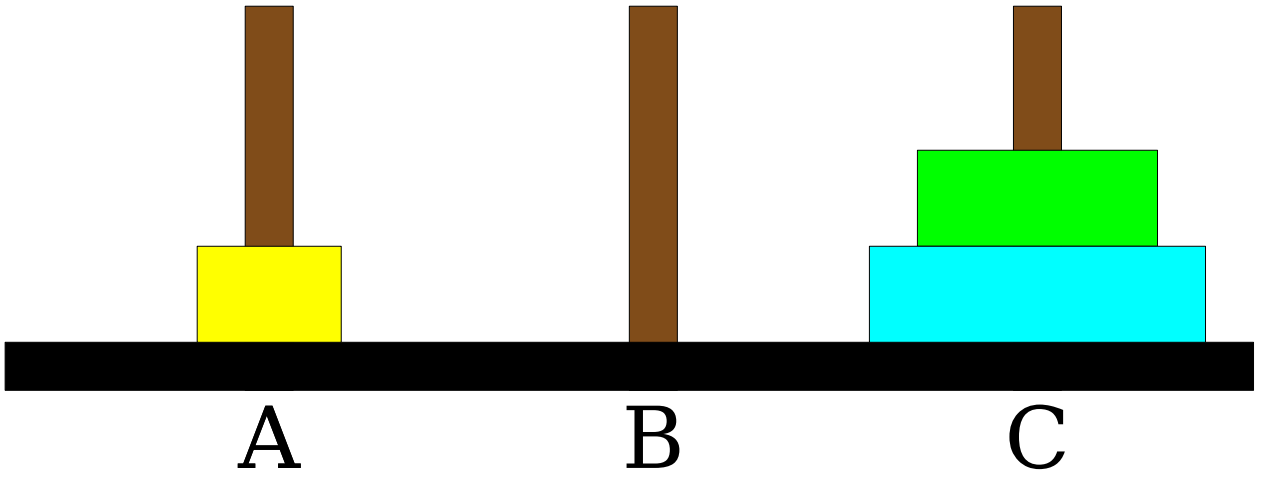
```
int main() {
}
```

```
void moveTower(int n, char from, char to, char temp) {
}
```

```
void moveTower(int n, char from, char to, char temp) {
}
```

```
void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n – 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n – 1, temp, to, from);
    }
}
```

n	1	from	a	to	c	temp	b



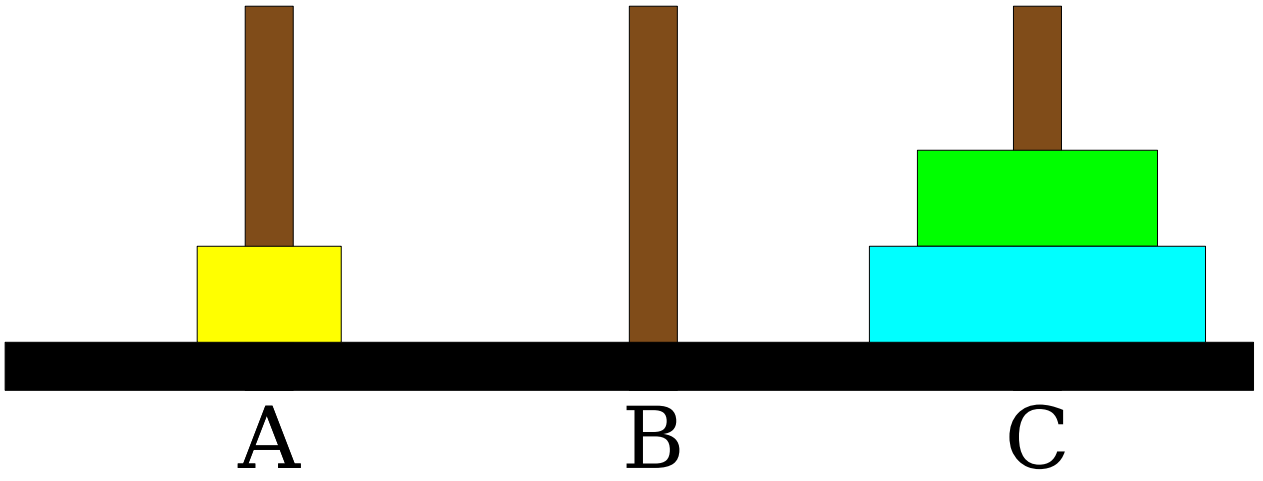
```
int main() {
}
```

```
void moveTower(int n, char from, char to, char temp) {
}
```

```
void moveTower(int n, char from, char to, char temp) {
}
```

```
void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n – 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n – 1, temp, to, from);
    }
}
```

n	1	from	a	to	c	temp	b



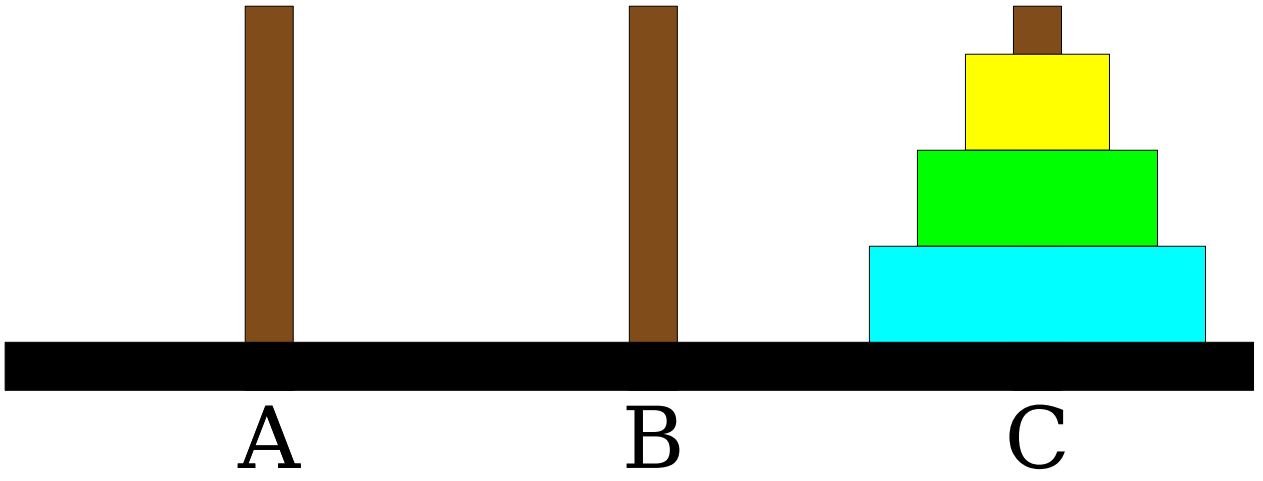
```
int main() {
}
```

```
void moveTower(int n, char from, char to, char temp) {
}
```

```
void moveTower(int n, char from, char to, char temp) {
}
```

```
void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n – 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n – 1, temp, to, from);
    }
}
```

n	1	from	a	to	c	temp	b

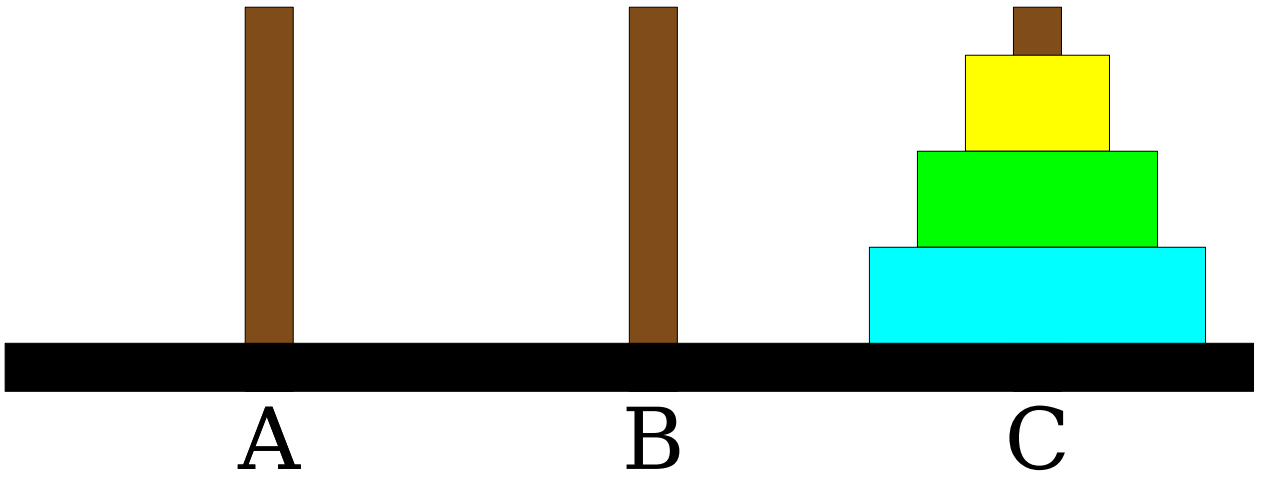


```
int main() {
}
```

```
void moveTower(int n, char from, char to, char temp) {
}
```

```
void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n – 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n – 1, temp, to, from);
    }
}
```

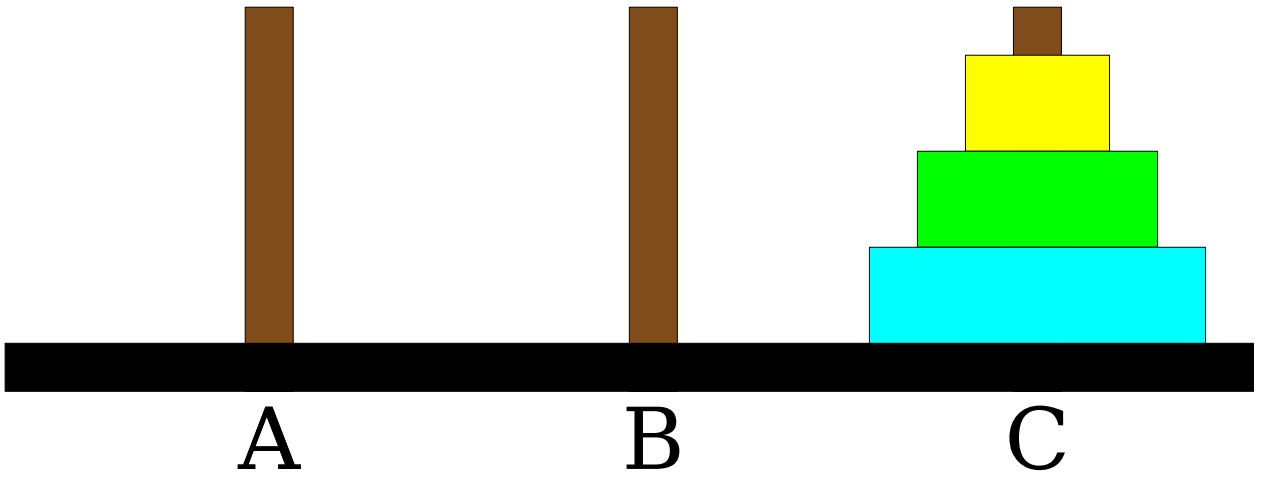
n	2	from	b	to	c	temp	a



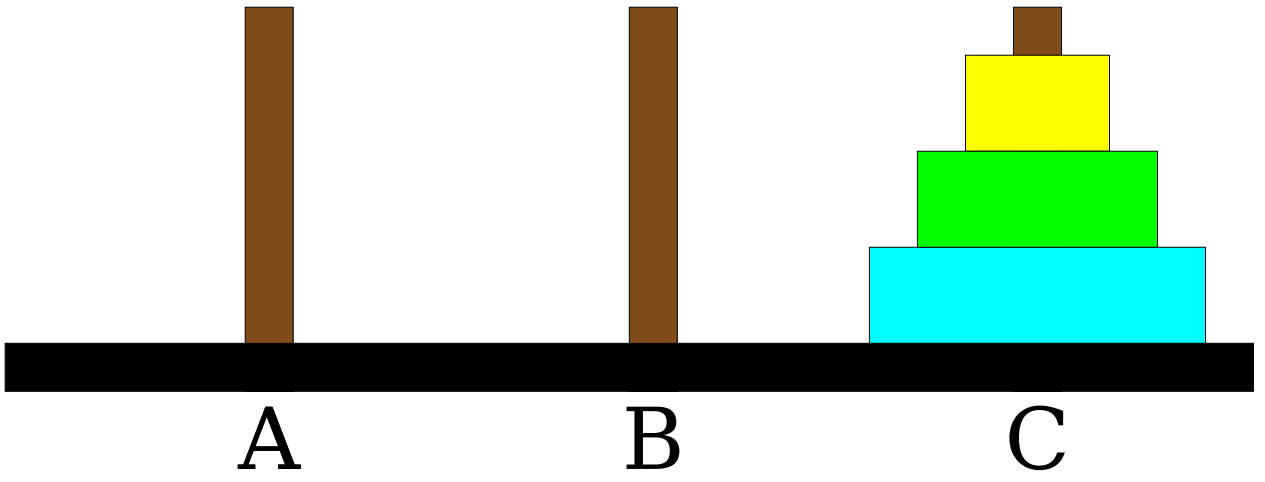
```
int main() {
}
```

```
void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n – 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n – 1, temp, to, from);
    }
}
```

n	3	from	a	to	c	temp	b



```
int main() {
    moveTower(3, 'a', 'c', 'b');
    return 0;
}
```



Emergent Behavior

- Even though each function call does very little work, the overall behavior of the function is to solve the Towers of Hanoi.
- It's often tricky to think recursively because of this ***emergent behavior***:
  - No one function call solves the entire problem.
  - Each function does only a small amount of work on its own and delegates the rest.

Writing Recursive Functions

- Although it is good to be able to trace through a set of recursive calls to understand how they work, you will need to build up an intuition for recursion to use it effectively.
- You will need to learn to trust that your recursive calls – which are to the function that you are currently writing! – will indeed work correctly.
  - Eric Roberts calls this the ***"Recursive Leap of Faith."***
- ***Everyone can learn to think recursively***. If this seems confusing now, ***don't panic!*** You'll start picking this up as we continue forward.

```
void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n - 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n - 1, temp, to, from);
    }
}
```

```
void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
        moveTower(n - 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n - 1, temp, to, from);
    }
}
```

```
void moveTower(int n, char from, char to, char temp) {
    if (n == 0) {

    } else {
        moveTower(n - 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n - 1, temp, to, from);
    }
}
```

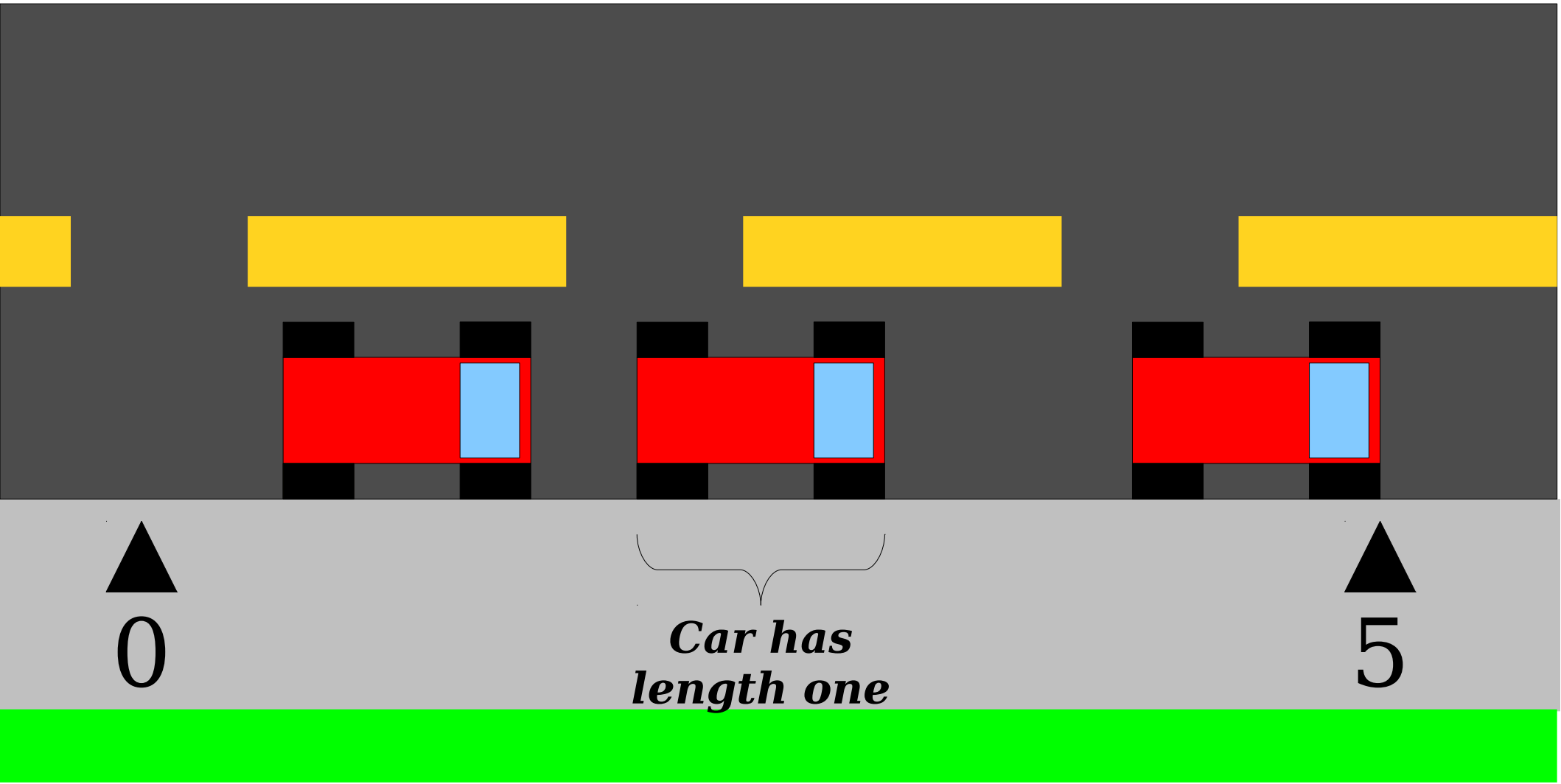
```
void moveTower(int n, char from, char to, char temp) {
    if (n > 0) {
        moveTower(n - 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n - 1, temp, to, from);
    }
}
```

```
void moveTower(int n, char from, char to, char temp) {
    if (n > 0) {
        moveTower(n - 1, from, temp, to);
        moveSingleDisk(from, to);
        moveTower(n - 1, temp, to, from);
    }
}
```

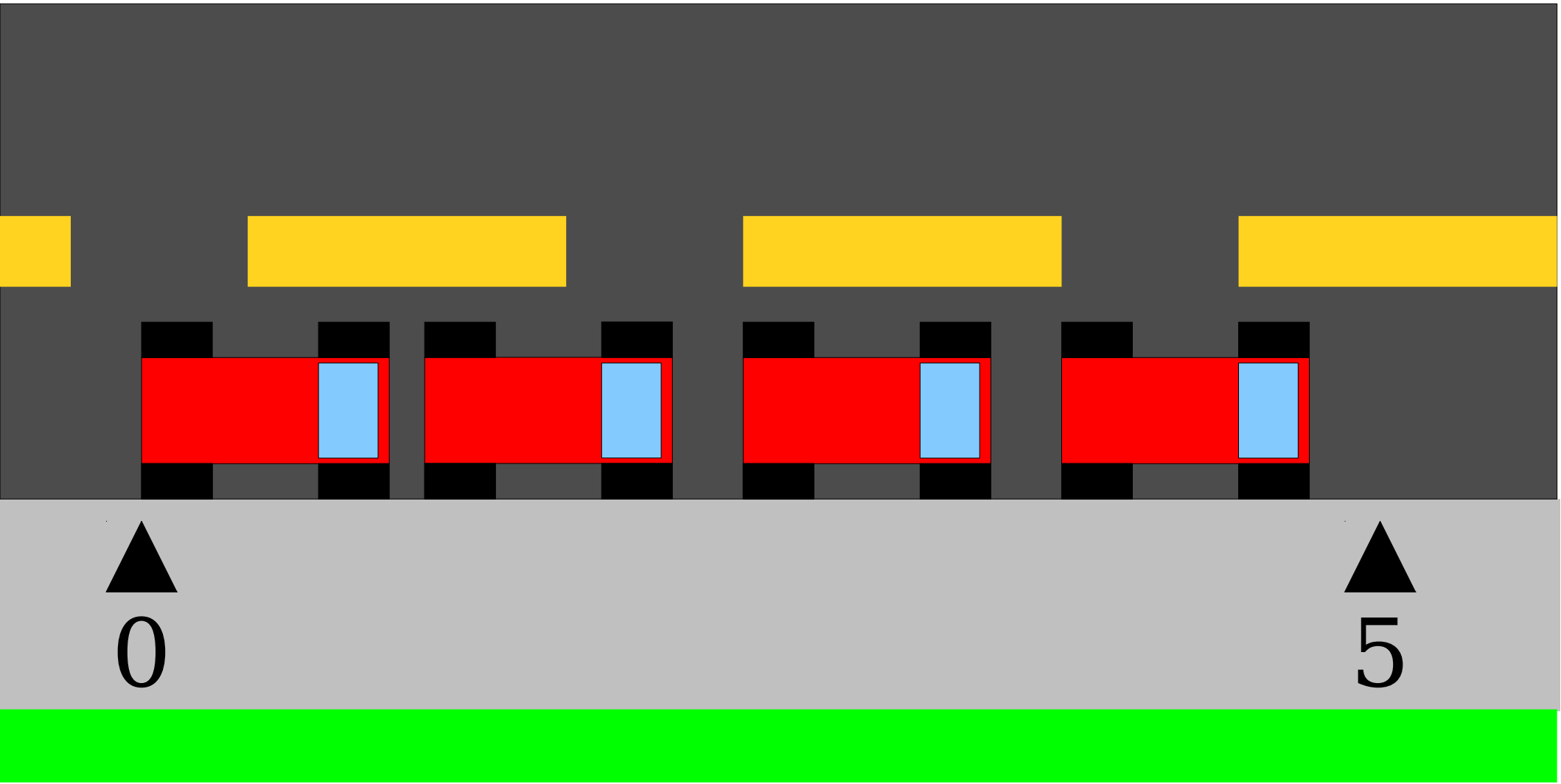
Picking a Base Case

- When choosing base cases, you should always try to pick the absolute smallest case possible.
- The simplest case is often so simple that it appears silly.
  - Solve Towers of Hanoi with no disks.
  - Add up no numbers.
  - Reverse an empty string.
- This is a skill you'll build up with practice.

Parking Randomly



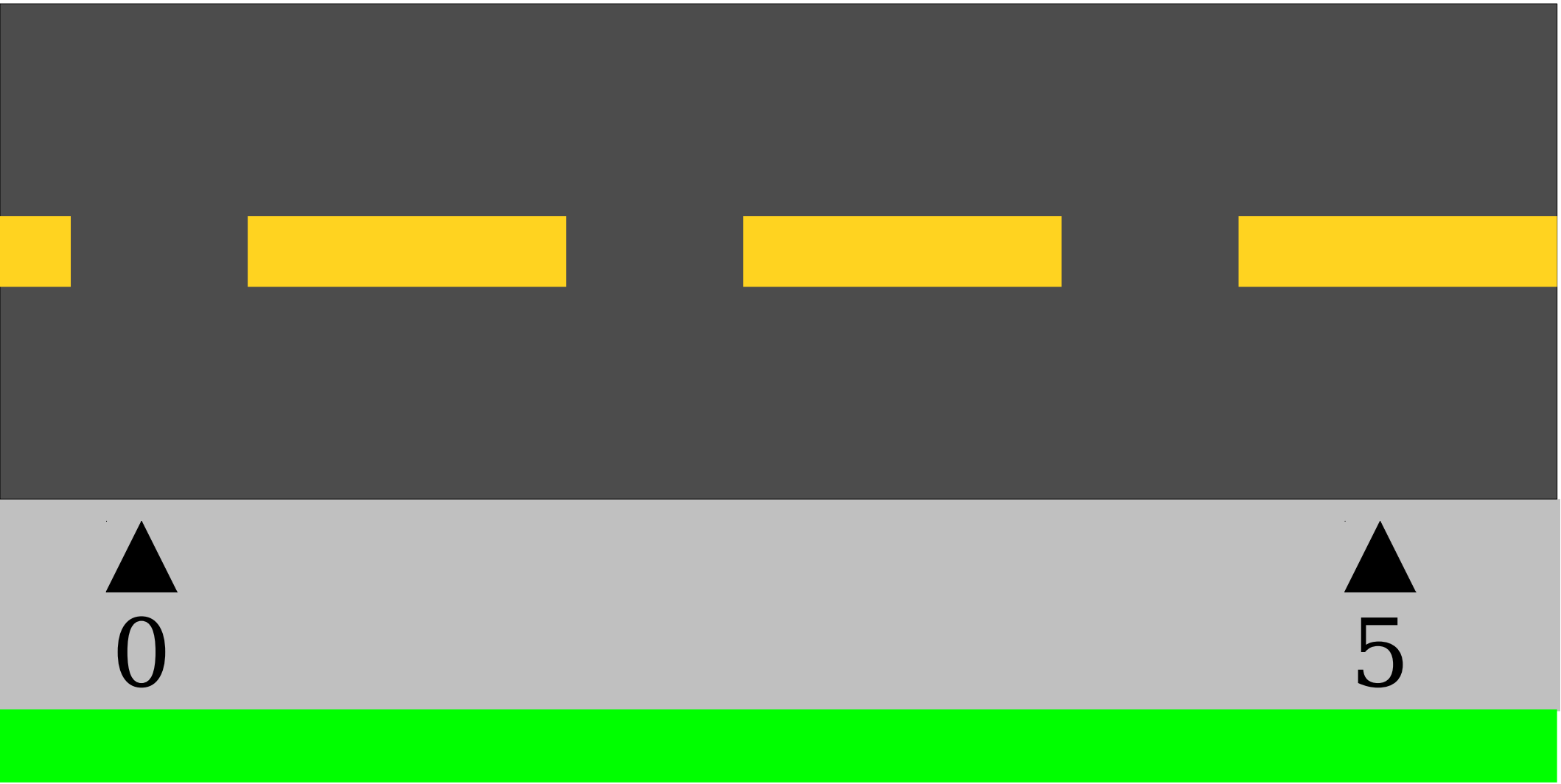
Parking Randomly



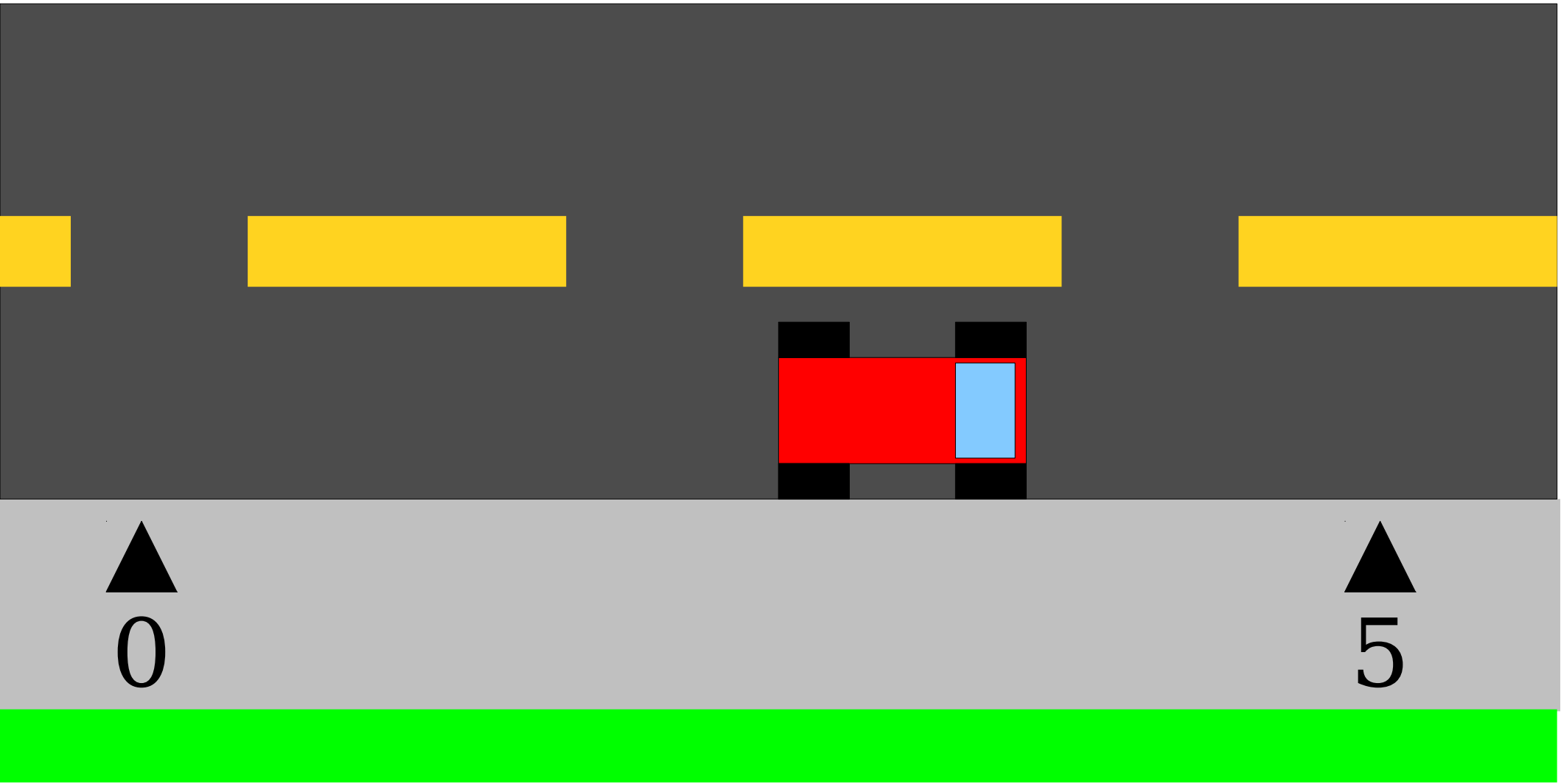
Parking Randomly

- Given a curb of length five, how many cars, on average, can park on the curb?
- We can get an approximate value through random simulation:
  - Simulate random parking a large number of times.
  - Output the average number of cars that could park.
- ***Question***: How do we simulate parking cars on the curb?

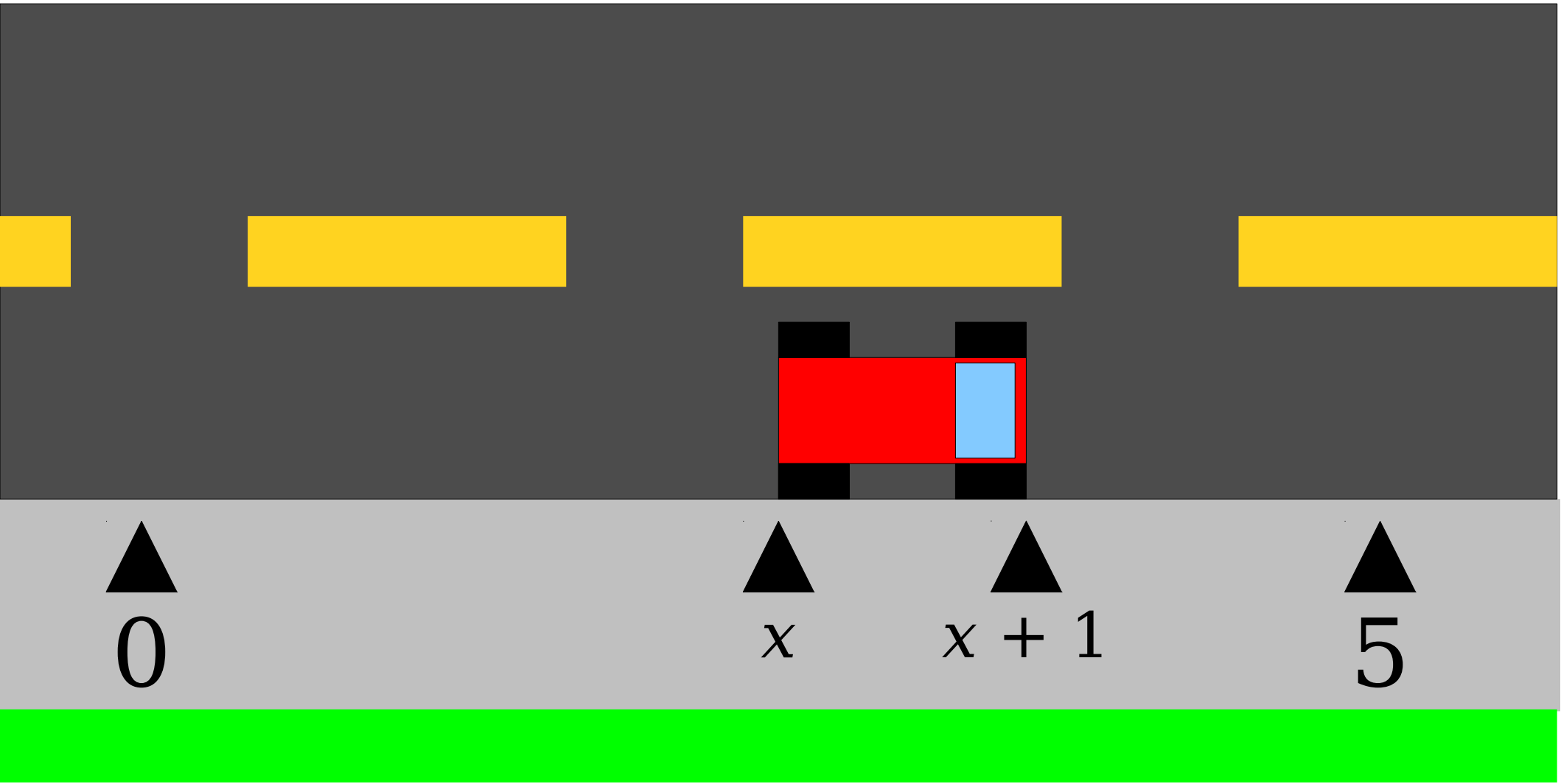
Parking Randomly



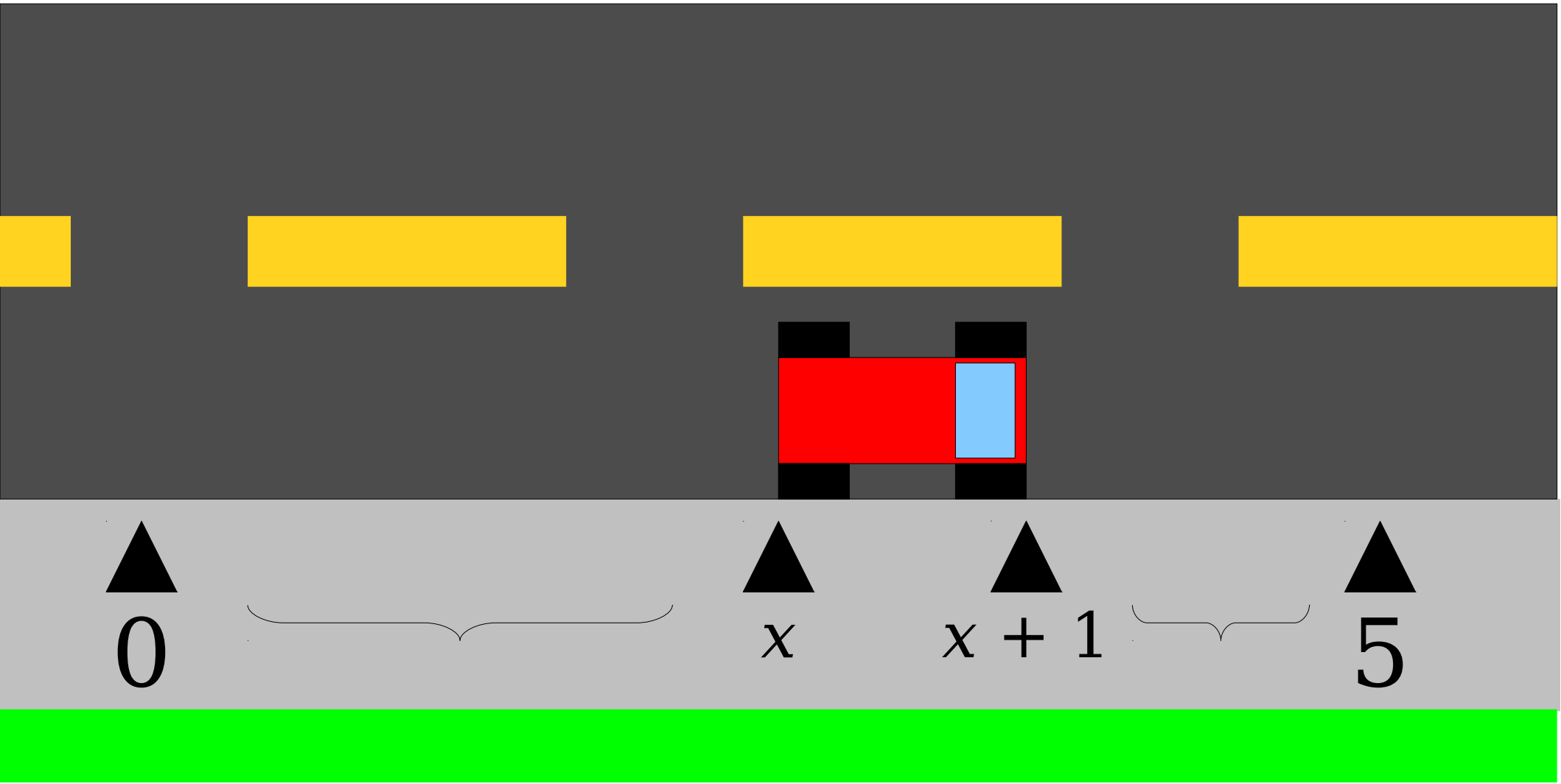
Parking Randomly



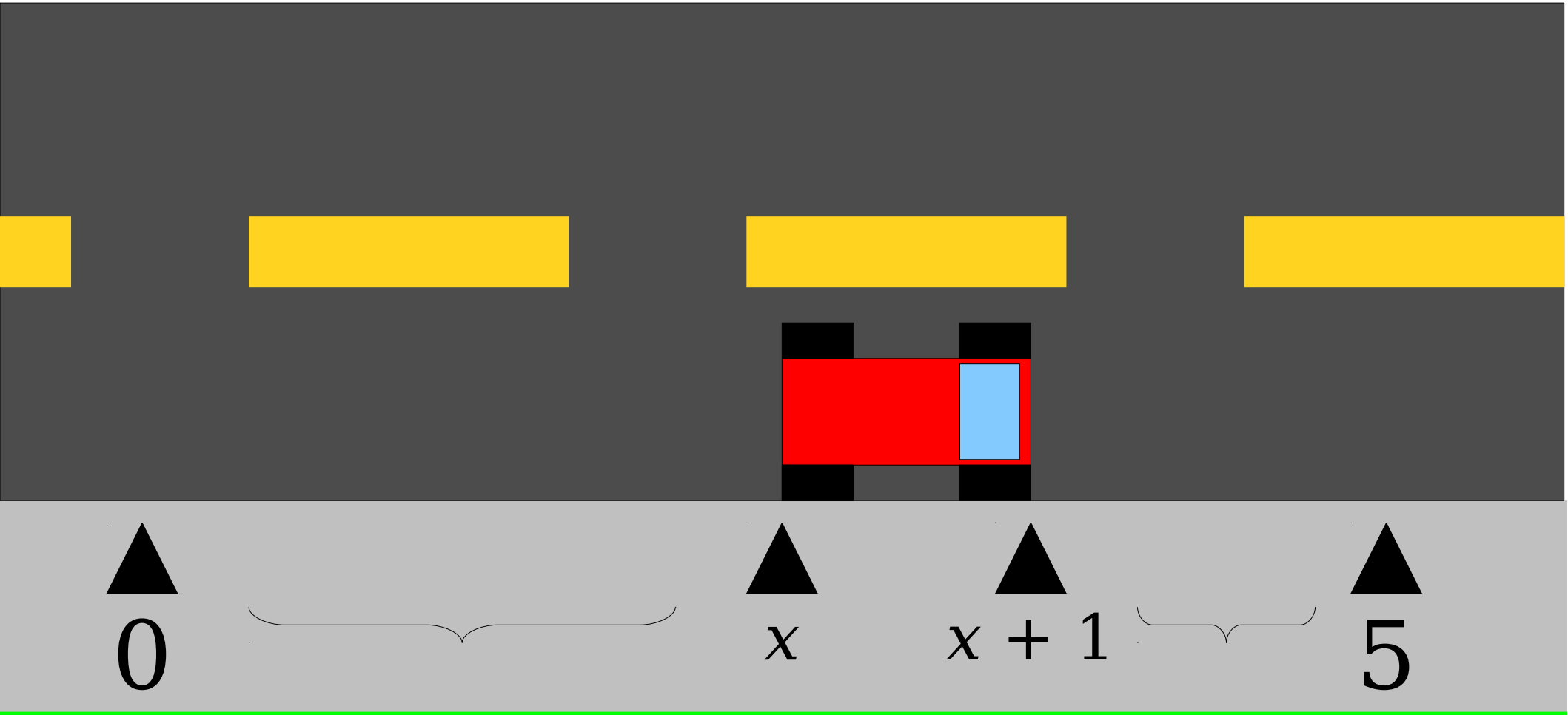
Parking Randomly



Parking Randomly



Parking Randomly



Place cars randomly in these ranges!

Parking Randomly

```
int parkRandomly(double low, double high) {
    if (high - low < 1.0) {
        return 0;
    } else {
        double position = randomReal(low, high - 1.0);
        drawCarAt(position);

        return 1 + parkRandomly(low, position) +
                   parkRandomly(position + 1.0, high);
    }
}
```

The Parking Ratio

- The average number of cars that can be parked in a range of width w for sufficiently large w is approximately

$$0.7475972\ w$$

- The constant 0.7475972... is called ***Rényi's Parking Constant***.
- For more details, visit **this link**!

So What?

- The beauty of our algorithm is the following recursive insight:

Split a problem into smaller, independent pieces and solve each piece separately.

- Many problems can be solved this way.

Next Time

- ***Graphical Recursion***
  - How do you draw a self-similar object?
- ***Exhaustive Recursion***
  - How do you generate all objects of some type?