This method is about how to represent the state space.

In n pegs, m disks hanoi problem, one state can be represented (naively) as a 2d-list, one dimension is pegs, the other dimension is the disks on certain peg.

E.g. 3 pegs, 3 disks hanoi problem's starting state can be represented like this:

start_state = [[3, 2, 1], [ ], [ ]]
If we move the smallest disk from peg 0 to peg 1, the state change to:
state_after_first_move = [[3, 2], [1], [ ]]

If we are going to avoid repeating, which means avoid coming back to an already searched state (which is obviously wasting time). We will need a buffer to hold all the states we already traveled through.

traveled_buff = [start_state, state_after_first_move]

And when we are planing our next move, we look though the traveled_buff, and cut off the move that lead us back to states in that buff, because we don't want to move back.

For example: if we are in state_after_first_move, we would not want to move back to start_state, because it's in traveled_buff, and it's obviously wasting our time to move one disk back and forth.

But, this would involved in comparing lot of states hold in the traveled_buff with the state you are considering. And this could be very time consuming, since the traveled_buff is becoming larger and larger with every steps you moved.

Here comes another example: Remember the vacuum cleaner example you used last week:

Suppose we are the vacuum cleaner located in B2, and our target is the dirt located in C4, and we are going to search the space to find shortest way lead us to that target.

	A	B	C
1			
2		VC	
3			
4			Dirt

How do you represent this state? My first idea is we can use a 2d list again, one for “longitude” and one for “latitude”.

Like, currently, the world can be described as:

```
[       [ 0, 0, 0 ],
        [ 0, 1, 0 ],
        [ 0, 0, 0 ],
        [ 0, 0, 2 ]
]
```

1 is where our vacuum cleaner is, and 2 is where the dirt is.

And then I realised it is not appropriate to describe this 2d list as a single state.

Why? Because we can use this 2d list to mark all the states we have been traveled through. In another word, this list can be used as a traveled_buff.

For example, if we move right one step and go down two steps until we reached the dirt. Our list can be marked as:

```
[       [ 0, 0, 0 ],
        [ 0,-1,-1],
        [ 0, 0,-1],
        [ 0, 0, 1 ]
]
```

1 is where we are, -1 is where we already searched, and since there are no dirt (2) on map, our job is done.

The advantages to use list like this to represent traveled_buff is, when we want to know if it is deja vu, we don’t need to do a lot of compare among a very long buffer list, we can simply go to the corresponding coordinate and see if there is an -1.

And then I thought, if there are a way to describe the traveled_buff of hanoi like that, then we will no longer need to do a lot of compare to check if we are coming back.

And then I found a solution:

The key is, we want to have a list, such that we can address a single state on that list with one set of coordinate. E.g. In the vacuum cleaner example, we can address to the starting state with coordinate (1, 1)

This means, every possible coordinates for that list, is mapped to every possible states in your problem world.

So the next question is going to be, how many possible states are there for n pegs, m disks hanoi problem?

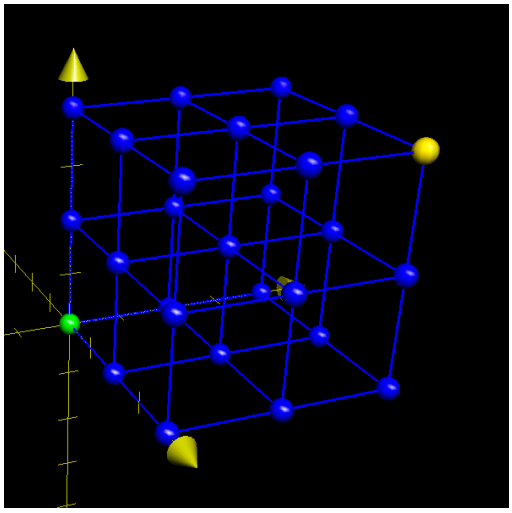
The answer is m ^ n.

Let's consider it this way:
We have n empty pegs, and m disks ready to put on them to create a new state. We first choose a peg for the biggest disk, there are n possible way, then we choose a peg for the second biggest disk, there are also n possible way. No matter what peg you choose now, it won't influence what you are going to choose next, because what you are going to choose next is always for a smaller disk.

So, we can build a n dimension array, with scale [0~m-1] for each dimension for our hanoi problem.

Now, what our problem become?

Let's consider a 3 pegs, 3 disks hanoi problem again. Our problem become like this: 3 ^ 3 = 27 possible states in total,
each dimension corresponding to the position (peg) of one of our disk.



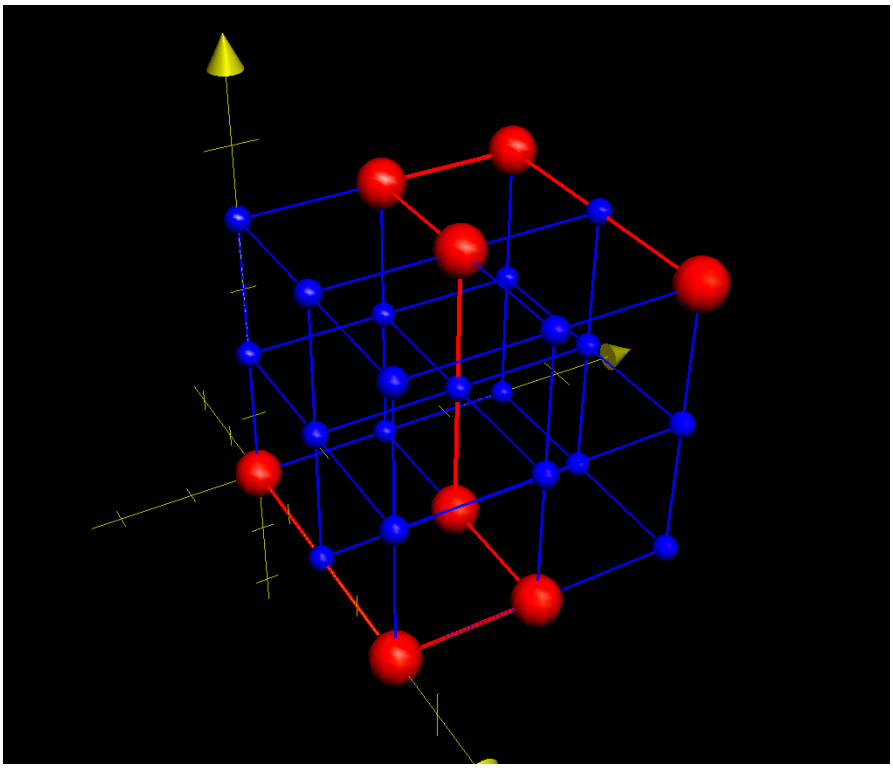
The blue balls on marked all possible state, and our target is moving from the green ball to the yellow ball.

For example: one possible solution

[[0, 0, 0], [2, 0, 0], [2, 1, 0], [1, 1, 0], [1, 1, 2], [0, 1, 2], [0, 2, 2], [2, 2, 2]],

Which means starting with all the disks in peg 0: [0, 0, 0];
Then move smallest disk to pegs 2: [2, 0, 0];
Then move second biggest disk to pegs 1: [2, 1, 0];
etc.

Can then be represented as:



The advantage of that is: every time we ask ourselves: are we coming back and forth if I move to that state? We no longer need to compare a lot of states in your old buffer, we can simply go to the coordinate in your new buffer, and see if there are a mark. And this can save us a lot of time in search.

In another word, ideally (without considering the state space could be sometime enormous), we can change any state based searching problem, to an n-dimensional maze problem.

The code solving hanoi problem using this method can be found in my github.