



Daily Coding Problem #51

Problem

This problem was asked by Facebook.

Given a function that generates perfectly random numbers between 1 and k (inclusive), where k is an input, write a function that shuffles a deck of cards represented as an array using only swaps.

It should run in $O(N)$ time.

Hint: Make sure each one of the $52!$ permutations of the deck is equally likely.

Solution

The most common mistake people make when implementing this shuffle is something like this:

- Iterate through the array with an index i
- Generate a random index j between 0 and $n - 1$
- Swap $A[i]$ and $A[j]$

That code would look something like this:

```
def shuffle(arr):  
    n = len(arr)  
    for i in range(n):  
        j = randint(0, n - 1)  
        arr[i], arr[j] = arr[j], arr[i]  
    return arr
```

This looks like it would reasonably shuffle the array. However, the issue with this code is that it slightly biases certain outcomes. Consider the following array: $[a, b, c]$. At each step i , we have three different possible outcomes: switching the element at i with any other index in the array.

Since we swap up to three times, we have $3^3 = 27$ possible (and equally likely) outcomes. But there are only 6 outcomes, and they all need to be equally likely:

- [a, b, c]
- [a, c, b]
- [b, a, c]
- [b, c, a]
- [c, a, b]
- [c, b, a]

6 doesn't divide into 27 evenly, so it must be the case that some outcomes are over-represented. Indeed, if we run this algorithm a million times, we see some skew:

```
(2, 1, 3): 184530
(1, 3, 2): 185055
(3, 2, 1): 148641
(2, 3, 1): 185644
(3, 1, 2): 147995
(1, 2, 3): 148135
```

Recall that we want every permutation to be equally likely: in other words, any element should have a $1/n$ probability to end up in any spot. To make sure each element has $1/n$ probability of ending up in any spot, we can do the following:

- Iterate through the array with an index i
- Generate a random index j between i and $n - 1$
- Swap $A[i]$ and $A[j]$

Why does this generate a uniform distribution? Let's use a loop invariant to prove this.

Our loop invariant will be the following: at each index i of our loop, all indices before i have an equally random probability of being any element from our array.

Consider $i = 1$. Since we are swapping $A[0]$ with an index that spans the entire array, $A[0]$ has an equally uniform probability of being any element in the array. So our invariant is true in this case.

Assume our loop invariant is true until i and consider the loop at $i + 1$. Then we should calculate the probability of some element ending up at index $i + 1$. That's equal to the probability of not picking that element up until i and then choosing it.

All the remaining prospective elements must not have been picked yet, which means it avoided being picked from 0 to i . That's a probability of $(n - 1 / n) * (n - 2 / n - 1) * \dots * (n - i - 1 / n - i)$.

Finally, we need to actually choosing it. Since there are $n - i$ remaining elements to choose from,

Finally, we need to randomly choose an element since there are $n - i - 1$ remaining elements to choose from, that's a probability of $1 / (n - i)$.

Putting them together, we have a probability of $(n - 1 / n) * (n - 2 / n - 1) * \dots * (n - i - 1 / n - i) * (1 / n - i)$. Notice that everything beautifully cancels out and we are left with a probability of $1 / n!$

Here's what the code looks like:

```
def shuffle(arr):  
    n = len(arr)  
    for i in range(n - 1):  
        j = randint(i, n - 1)  
        arr[i], arr[j] = arr[j], arr[i]  
    return arr
```

P.S. This algorithm is called the Fisher-Yates shuffle.

© Daily Coding Problem 2019

[Privacy Policy](#)

[Terms of Service](#)

[Press](#)