Daily Coding Problem #295

Problem

This problem was asked by Stitch Fix.

Pascal's triangle is a triangular array of integers constructed with the following formula:

- The first row consists of the number `1`.
- For each subsequent row, each element is the sum of the numbers directly above it, on either side.

For example, here are the first few rows:

```
    1
   1 1
  1 2 1
 1 3 3 1
1 4 6 4 1
```

Given an input `k`, return the k^{th} row of Pascal's triangle.

Bonus: Can you do this using only `O(k)` space?

Solution

Solution

One straightforward solution is to apply the rule described above on each subsequent row. To make our operations simpler, we can create a right, instead of equilateral, triangle, that obeys similar properties:

```
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
```

To construct this triangle, we first initialize a `k` by `k` array, and iterate through each row, setting each value to be the sum of its top and top-left values. Finally, we return the last row.

```
def pascal(k):
    rows = [[0 for _ in range(k)] for _ in range(k)]

    rows[0][0] = 1
    for i in range(1, k):
        for j in range(i + 1):
            rows[i][j] = rows[i - 1][j - 1] + rows[i - 1][j]

    return rows[-1]
```

This will take $O(k^2)$ time and space. As indicated above, though, there is in fact a solution which only uses $O(k)$ space. Note that to calculate the values in each row, we only need access to the row immediately above it.
Now consider an arbitrary row, say `k = 3`, but prepended and appended with a zero: `[0, 1, 3, 3, 1, 0]`. It turns out that working our way backwards through this row, incrementing each value by that of its left neighbor, precisely imitates the logic seen above.

For example, in the above row, we would perform the following operations:

- Replace `0` with `1 + 0 = 1`
- Replace `1` with `3 + 1 = 4`
- Replace `3` with `3 + 3 = 6`
- Replace `3` with `1 + 3 = 4`

- Replace `1` with `1 + 0 = 1`

The first element will remain zero, but with each iteration the rest of the list will be transformed into the correct values. As a result, we can carry out this operation `k` times, and finally return the row with the first element chopped off.

```
def pascal(k):
    rows = [0 for _ in range(k + 1)]

    rows[1] = 1
    for i in range(1, k + 1):
        for j in range(i, 0, -1):
            rows[j] += rows[j - 1]

    return rows[1:]
```

Finally, the mathematicians among you may know that the `n + 1`th row in Pascal's triangle can be given by the values:

`n`^{C}`0`, `n`^{C}`1`, `...`, `n`^{C}`n`, where `n`^{C}`r` represents the number of ways of choosing `r` items out of `n`.

Therefore, we can implement `n`^{C}`r` and use it to elegantly solve our problem. While this solution also uses `O(k)` space, the factorial function will bring our run time up to `O(k`^{2}`)`.

```
from math import factorial

def nCr(n, r):
    return factorial(n) // factorial(r) // factorial(n - r)

def pascal(k):
    return [nCr(k - 1, i) for i in range(k)]
```

Privacy Policy
Terms of Service

Press