Daily Coding Problem #9

Problem

This problem was asked by Airbnb.

Given a list of integers, write a function that returns the largest sum of non-adjacent numbers. Numbers can be `0` or negative.

For example, `[2, 4, 6, 2, 5]` should return 13, since we pick 2, 6, and 5. `[5, 1, 1, 5]` should return `10`, since we pick 5 and 5.

Follow-up: Can you do this in O(N) time and constant space?

Solution

This problem seems easy from the surface, but is actually quite tricky. It's tempting to try to use a greedy strategy like pick

the largest number (or first), then the 2nd-largest if it's non-adjacent and so on, but these don't work -- there will always be some edge case that breaks it.

Instead, we should look at this problem recursively. Say we had a function that already returns the largest sum of non-adjacent integers on smaller inputs. How could we use it to figure out what we want?

Say we used this function on our array from `a[1:]` and `a[2:]`. Then our solution should be `a[1:]` OR `a[0] + a[2:]`, whichever is largest. This is because choosing `a[1:]` precludes us from picking `a[0]`. So, we could write a straightforward recursive solution like this:

```
def largest_non_adjacent(arr):
    if not arr:
        return 0

    return max(
            largest_non_adjacent(arr[1:]),
            arr[0] + largest_non_adjacent(arr[2:]))
```

However, this solution runs in $O(2^n)$ time, since with each call, we're making two further recursive calls. We could memoize the results, or use dynamic programming to store, in an array, the largest sum of non-adjacent numbers from index `0` up to that point. Like so:

```
def largest_non_adjacent(arr):
    if len(arr) <= 2:
        return max(0, max(arr))

    cache = [0 for i in arr]
    cache[0] = max(0, arr[0])
    cache[1] = max(cache[0], arr[1])

    for i in range(2, len(arr)):
```

```
        num = arr[i]
        cache[i] = max(num + cache[i - 2], cache[i - 1])
    return cache[-1]
```

This code should run in O(n) and in O(n) space. But we can improve this even further. Notice that we only ever use the last two elements of the cache when iterating through the array. This suggests that we could just get rid of most of the array and just store them as variables:

```
def largest_non_adjacent(arr):
    if len(arr) <= 2:
        return max(0, max(arr))

    max_excluding_last= max(0, arr[0])
    max_including_last = max(max_excluding_last, arr[1])

    for num in arr[2:]:
        prev_max_including_last = max_including_last

        max_including_last = max(max_including_last, max_excluding_last + num)
        max_excluding_last = prev_max_including_last

    return max(max_including_last, max_excluding_last)
```