Daily Coding Problem Blog

Daily Coding Problem #318

Problem

This problem was asked by Apple.

You are going on a road trip, and would like to create a suitable music playlist. The trip will require `N` songs, though you only have `M` songs downloaded, where `M < N`. A valid playlist should select each song at least once, and guarantee a buffer of `B` songs between repeats.

Given `N`, `M`, and `B`, determine the number of valid playlists.

Solution

First let us consider a simpler problem: creating playlists of `N` songs from `M` downloaded options, but without any buffer requirement.

We can construct a solution using dynamic programming. In particular we would like to construct a matrix, `ways`, such that `ways[i][j]` represents the number of ways of making a playlist of length `i` with `j` unique songs.

Our base case is when `i` and `j` are both zero. Here, we can consider there to be only one trivial option: a playlist with no songs. Otherwise, we can find the value of `ways[i][j]` by dividing into two cases: when the last song is new, and when the last song is a repeat.

If the last song is new, it could be any of the `m - (j - 1)` unused songs. Each of these songs can be combined with any of the playlists with one fewer song and one fewer unique song, or `ways[i - 1][j - 1]`. On the other hand, if the last song is a repeat, it must be one of the `j` songs already chosen. Each of these songs can be combined with any option provided by `ways[i - 1][j]`.

Once we built up our matrix in this way, we can take the value of `ways[n][m]` to be our solution. We can implement this as follows.

```
def valid_playlists(n, m, b):
    ways = [[0 for _ in range(m + 1)] for _ in range(n + 1)]
    ways[0][0] = 1

    for i in range(1, n + 1):
        for j in range(1, m + 1):
            ways[i][j] = ways[i - 1][j - 1] * (m - (j - 1)) + ways[i - 1][j] * j



    return ways[n][m]
```

Now let us try adding in the buffer `B` to our dynamic programming formula. If the last song is new, no change needs to be made, since it cannot possibly be a repeat.

If the song is old, on the other hand, there will be `B` options that cannot be used as the next song, specifically the last `B` songs in our playlist. Therefore the number of new playlist formed can be represented as `ways[i - 1][j] * (j - b)`. If the buffer is bigger than the number of distinct songs played so far, no repeat songs are possible, so we can use `max(j - b, 0)` to handle this case.

```
def valid_playlists(n, m, b):
    ways = [[0 for _ in range(m + 1)] for _ in range(n + 1)]
    ways[0][0] = 1

    for i in range(1, n + 1):
        for j in range(1, m + 1):
            ways[i][j] = ways[i - 1][j - 1] * (m - (j - 1)) + ways[i - 1][j] * max(j - b,
0)

    return ways[n][m]
```

The time and space complexity of this algorithm will be `O(M * N)`, since we must loop through each cell of our `M x N` matrix and perform a few calculations.

Privacy Policy

Terms of Service

Press