Mock Interviews @ ACM

Solution Card

Room Draw Woes: The Sequel

Problem Statements

Housing is struggling with another problem — can you help them out? **Note:** We recommend that you first do *Room Draw Woes* and then solve this problem as a follow-up.

You have a draw group with `N` people and Housing has a specific set of room sizes `rooms` they can offer. Write a function `roomDrawTwo(N, rooms)` that outputs the *total number of ways* Housing can assign some set of rooms so that everyone is accommodated *and* no room is under capacity. (Assume there is an infinite supply of each room.)

Assumptions. You can assume `N` and the elements of `rooms` (i.e. possible room sizes) will fit in a 32-bit positive integer. Assume the total number of ways can also fit in a 32-bit integer.

As usual, you should analyze the runtime and space usage of any solution you come up with.

Source: Coin Change 2 from Leetcode.

Hints

Hint. We have to think of a way not to overcount the possible combinations. Let `ways[j][k]` represent the number of ways that you can accommodate exactly k people using only room sizes less than or equal to the jth smallest room offered. For example, if `N = 7` and `S = {1, 3, 4}`, we would have that `ways[2][5] = 2` because there are three ways (1 + 1 + 1 + 1 + 1, 1 + 1 + 3) to accommodate exactly five people with the 2 smallest room types offered (singles and triples).

With this idea, can you come up with a recurrence for `ways[j][k]`? Why wouldn't this result in overcounting?

Solution

Note: The idea to prevent overcounting is to start by allowing only the smallest room sizes, and slowly working our way up.

Taking the hint, we can see that `ways[j][k]` will be

- The number of ways (k - sj) people can be accommodated using only the j smallest room sizes — where sj is the size of the jth smallest room offered — plus
- The number of ways k people can be accommodated using only the (j - 1) smallest room sizes.

Recall that `ways[j][k]` represents the number of ways that k people can be accommodated with only the j smallest room sizes. Note also that if (k - j) is negative, then the contribution from the first bullet point will be nothing. Again, don't forget to define the base cases: we can accommodate zero people in one way, regardless of how many room sizes we are allowed. An implementation is below:

```
public int roomDrawTwo(int N, int[] rooms) {

    /* This will be our dynamic programming array: ways[j][k] is the number
       of ways that we can accommodate k people with only the j smallest room
       sizes. */
    int[][] ways = new int[rooms.length + 1][N + 1];

    ways[0][0] = 1; // A base case

    /* The dynamic programming step. */
    for (int j = 1; j <= rooms.length; j++) {
        ways[j][0] = 1; // We can always accommodate 0 people in 1 way

        for (int k = 1; k <= N; k++) {
            ways[j][k] = ways[j - 1][k]; // Second bullet point from above

            /* Add the contribution from the first bullet point. */
            int sj = rooms[j - 1];
            if (sj <= k)
                ways[j][k] += ways[j][k - sj];
        }
    }

    return ways[rooms.length][N];
}
```

Analysis

Time. Let M be the number of room types housing can offer (i.e. the size of `rooms`). The runtime is dominated by the dynamic programming step, which does an O(M) operation for every element in `min`. Altogether, this means that the algorithm's runtime is **O(MN).**

Space. The auxiliary space needed is dominated by the space needed to store the `ways` array. It has (M + 1) rows and (N + 1) columns, so the auxiliary space needed is **O(MN).**

Tips and tricks

- In Java, when you initialize an array storing an int, long, or double, every element will be auto-set to 0. If initializing a boolean array, all elements are auto-set to false.
- Notice that in this problem our dynamic programming array had to be 2D rather than 1D (as with the original *Room Draw Woes* problem). With dynamic programming, you may potentially need to use many indices to describe a subproblem! You should just think of dynamic programming as splitting up a problem into strictly smaller subproblems of the same form and using the solutions of those subproblems to reconstruct the solution of the original. (It's a lot like recursion where you store the results of each function call.)

Follow-ups

Locality. Notice that our `ways` array is first indexed by a limit on the room size (j) and then by the number of people we need to accommodate (k). Which of the statements in the body of the dynamic programming loop have good spatial locality because of this? Bad spatial locality? What if instead we switched the roles of j and k — which statements would have good spatial locality? (Read up on spatial locality on Wikipedia if you are unfamiliar with the concept.)