Smart Phone

Category: Sorting

Problem link: https://www.codechef.com/ZCOPRAC/problems/ZCO14003

Solution

Sort the array. If you do that, the number of people who will buy the app at the price equal to the `ith` person's budget will be `N - i`. After sorting, we can brute-force to maximize the value `budget[i] * (N - i)` over all valid `i` using a `for` loop.

SUPW

Category: Dynamic Programming

Code: submission.cpp

Problem link: https://www.codechef.com/ZCOPRAC/problems/ZCO14002

Solution

For each day, we can recursively decide whether to work on that day or not. However, if we haven't worked on a given day.

Step 1: Define a subproblem

Let `solve(idx, days)` be the minimum number of total minutes you need to work from day `idx` till day `N - 1`, such that at no point a situation arises where you go three days without doing the SUPW duty, given that you haven't worked for the past `days` days.

Step 2: Base cases

Consider the simplest cases where you know the solution to the problem.

Assume the number of minutes given for day `i` is `minutes[i]`, where `0 <= i < N`.

Consider the case of the `(N - 1)th` day which is the last day.

Then `solve(N - 1, 0) = 0` because you already worked yesterday and working again on the last day is not necessary because it will only add to the number of minutes.

Similarly `solve(N - 1, 1) = 0`

However, `solve(N - 1, 2) = minutes[N - 1]` because you didn't work for the past two days, and thus you will have to work today.

Step 3: Recurrence relation

Dynamic programming is careful brute-force. We'll try out all combinations, but ensure that we do not compute the same result again.

If `days < 2`, we have a choice regarding whether to work on the given day `idx` or not. So we compute the answer for both the cases and return their minimum

```
if days < 2:
  return min( solve(idx + 1, days + 1), solve(idx + 1, 0) + minutes[i] )
```

Otherwise (when `days` is 2), we don't have a choice, so we'll have to work on day `idx`

```
else:
  return solve(idx + 1, 0) + minutes[i]
```

Memoize

Store the return value of the `solve()` function in a memo table (perhaps an array or a vector in this case) and use it directly if already computed.

Endnote

Exercise: Argue why the base cases described above can be generalized to the following base case without affecting the recurrence:

`solve(N, 0) = solve(N, 1) = solve(N, 2) = 0`

Video Game

Category: Implementation Problem link: https://www.codechef.com/ZCOPRAC/problems/ZCO14001

Solution

Store the number of boxes in each position in an array/vector. Then just follow the instructions to update the number of boxes in each position. And then just output the elements of the array/vector.