Greedy Python with different explanation

50
VIEWS

2

Last Edit: September 14, 2019 6:48 PM

dridon 2

My solution is taken from mud2man and attixZhang. It took me some time to understand what they said so I wanted to try and break it down a little further. I want to offer my explanation to help others that may find themselves having trouble.

I break this problem down in two three parts:

1. Determine possibility of a solution
2. Determine operations to reach target
3. Minimize Error

The first one is straightforward, the other two take a little thinking. Lets talk about each step.

Determine possibility of a Solution

Take the following two values:

```
minSum = sum(floor(p) for p in prices)
maxSum = sum(ceil(p) for p in prices)
```

These are the minimum and maximum possible summations possible with `Round(p)`. The former occurs by summing all the floored prices and the latter from summing all the ceiled prices. Now if target falls between these two values the solution is possible:

```
if minV <= target <= maxV:
   #solution possible
else:
    return "-1"
```

In fact, not only is the solution possible. It is guaranteed. To understand, let us take a look at the second part.

Determine operations to reach target

So first remember `minV`, `maxV` and `target` are integer values. So the difference between any of these is an integer value. Particularly the following:

```
K = target - minV
```

This is the number of ceiling operations we must conduct. To understand, let us imagine the prices array:

```
prices = [ p1, p2, p3, ..., pN]
```

Now for a `pi` label its `floor(pi)` as `fpi`. This gives us the following array:

```
floored_prices = [ fp1, fp2, fp3, ..., fpN ]
```

Remember, that `minV = sum(floored_prices)`. So basically if we could change `floored_prices` such that we have `K` added to `minV` then we a sum to the target. One way to have this would be to add `1` to any `K` elements in `floored_prices`:

```
floored_prices_plus_K = [ fp1+1, ..., fK+1, ..., fpN ]
```

We achieve this by ceiling the first K (or really any K) values instead of flooring them when generating the list. Remember `ceil(p) == floor(p) + 1`:

```
floored_prices_plus_K = [ ceil(p1), ..., ceil(pK),floor(p(K+1)), ..., floor(pN) ]
```

This array will give us `minV + K == target`. But we are not at the answer yet. This is where the "sorting" or "k smallest errors" come in to play.

Minimize Error

Since we know that any `K` elements ceiled instead of floored will give us the summation of the target we only need to pick the ones that will give us the lowest error. This is not difficult. Lets call the flooring error `floorError(p) -> p - floor(p)`. Now map prices using this error:

```
floorErrors = [ floorError(p1], floorError(p2), ... floorError(pN) ]
```

Now sort it and take the largest `K` errors. If we use the ceil error `ceilError(p) -> ceil(p) - p` instead of `floorError` for these values and sum all these values, get the minimum error. The intuition is easily understood by taking a look at two actual numbers. Take `3.6` and `5.7` (arbitrarily generated). The floor errors are `0.6` and `0.7` respectively but their ceil errors are `0.4` and `0.3` respectively. So the prices with the biggest `floorErrror` will have the smallest `ceilError` (unless all errors are 0.5). This is actually fairly easy to prove mathematically:

```
0.0 <= fE1, fE2 <= 1.0  # floor errors
fE1 > fE2
1.0 - fE1 < 1.0 - fE2
# but 1.0 - fE1 = cE1 (the ceil error)
cE1 < cE2
```

A bit pedantic but hopefully it helps those who don't get the other solutions right away:

```
class Solution:
    def minimizeError(self, prices: List[str], target: int) -> str:
        maxV, minV, errors = 0,0, []

        # gather minV, maxV and errors
        for p in prices:
            fp = float(p)
            f, c = math.floor(fp), math.ceil(fp)
            minV, maxV = minV + f, maxV + c
            fError, cError = fp - f, c - fp
            errors.append((fError, cError))

        # lets make sure this is actually possible
        if target < minV or target > maxV:
            return "-1"

        # The number of prices that need to be rounded up (rest are rounded down)
        ceilCount = target - minV

        # Floor errors are enough to give us what we need
        errors = sorted(errors, reverse=True)

        #min error
        minError = sum(e[1] for e in errors[:ceilCount]) + sum(e[0] for e in erro

        # return the error
        return "{:.3f}".format(minError)
```