CS208 · Algorithm Design and Analysis

何泽安 (He Zean)
12011323

May. 6, 2022

Assignment 6

Ch.6 - Ex.2

(a)

	Week 1	Week 2	Week 3
h	100	1	1
ℓ	1	1	1

Consider such simple example, it's trivial that the best strategy is to choose the high-stress job in week 1, and choose low-stress jobs in week 2 and 3, say $\{\text{high} - \text{low} - \text{low}\}$, which will gain $100 + 1 + 1 = 102$. But the wrong algorithm will start at week 1, finding that $(h_2 = 1) < (\ell_1 + \ell_2 = 2)$ and thus step into the *else* branch, and do the similar choice in week 2 and 3, finally we will get $\{\text{low} - \text{low} - \text{low}\}$ which is $1 + 1 + 1 = 3$.

(b)

Suppose there are n weeks in total. We define the function $OPT(i)$ as the value of the optimal choice in the first i weeks, with h_i and ℓ_i given. We also define that $OPT(i) = h_i = \ell_i = 0$ for $i \in (-\infty, 0] \cup (n, +\infty)$ to avoid overflowing array bounds. Now that by the question, once we choose to do a high-stress job in a week, we cannot choose any job in the week before that week, aka. the value of week $i - 1$ is 0. But if we choose a low-stress job in one week, the $i - 1$ weeks before can have an optimal choice $OPT(i - 1)$ which will no affect week i. In short:

$$OPT(i) = \max\{OPT(i-1) + \ell_i, OPT(i-2) + h_i\}$$

If we do the memorize job, aka. save them into an array, s.t. when we are calculating $OPT(i)$, we will have all the computed value $OPT(1 \cdots i-1)$ saved in an array, therefore, in each iteration we can use $O(1)$ to calculate one item of OPT. Finally, to get the value of optimal choice for the n weeks, we need to compute $OPT(1 \cdots n)$ which is $O(n)$, quiet efficient! (Note: collecting $2n$ numbers of h_i and ℓ_i takes another $O(n)$, which is omitted in the following code.)

```
def get(arr: List[int], i: int):
    return 0 if i < 0 or i >= len(arr) else arr[i]


def find_opt(n: int,  # there are n weeks in total
             hs: List[int],  # len(hs) == n, hs[i] = h_i
```

```
            ls: List[int],  # len(ls) == n, ls[i] = l_i
        ):
    # since this is python, it's nature for us to use 0-indexed array here
    opt = [0] * n  # init for OPT memorize
    for i in range(n):
        opt[i] = max(get(opt, i - 1) + get(ls, i),
                     get(opt, i - 2) + get(hs, i))
    return opt[-1]
```