University of Washington
Department of Computer Science and Engineering
CSE 417, Winter 2020

February 25, 2020

Homework 7 problem 2

Problem 2 (10 points) Task Choice:

Suppose that each week you have the choice of a high stress task, a low stress task, or no task. If you take a high stress task in week i, you are not allowed to take any task in week $i+1$. For n weeks, the high stress tasks have payoff $h_1, \ldots, h_n$, and the low stress tasks have payoff $l_1, \ldots, l_n$, and not doing a task has payoff 0. Give an algorithm which given the two lists of payoffs, maximizes the value of tasks that are performed over n weeks. The run time of the algorithm should be polynomial in n.

Answer:

algorithm:

```
# G = [v1, v2, v3 ...]
def job_choice(h,l):
    n = len(h)
    dp = [[0,0,0] for i in range(n+1)]
    for i in range(0, n):
        dp[i+1][0] = max(dp[i][0], dp[i][1], dp[i][2])
        dp[i+1][1] = max(dp[i][0], dp[i][1]) + l[i]
        dp[i+1][2] = max(dp[i][0], dp[i][1]) + h[i]
    print(dp)
    return max(dp[n])
```

Correctness proof by induction:

base case:

By defination, when i = 0, no work can be done in the week 0, hence the optimal payoff for either high stress, low stress and no task is all zero.

induction hypothesis:

For $i > 0$:
The optimal payoff for choosing no task:
max(dp[i][0], dp[i][1], dp[i][2])
The optimal payoff for choosing low stress task:
max(dp[i][0], dp[i][1]) + l[i]
The optimal payoff for choosing high stress task:
max(dp[i][0], dp[i][1]) + h[i]

(dp[i] as the optimal solution of subproblem till day i)

According to the context of problem, there are only three possibilities considering each day:

(1) choose no task: Any task day can be followed by a no task day, hence the optimal payoff is exactly the optimal payoff of the day before, since no payoff added on the day i, therefore the optimal payoff should be the maximum of each state: max(dp[i-1][0], dp[i-1][1], dp[i-1][2]).

(2) choose low stress task: A low stress day can only follow a non-high stress day, according to the problem. Therefore the optimal payoff should be the maximum of value built upon the optimal payoff of the day before with states of low stress or no task: max(dp[i-1][0], dp[i-1][1]) + l[i]

(3) choose high stress task: A high stress day can only follow a non-high stress day, according to the problem. Therefore the optimal payoff should be the maximum of value built upon the optimal payoff of the day before with states of low stress or no task: max(dp[i-1][0], dp[i-1][1]) + h[i]

Since the function dp(i) always correctly choose the optimal result till day i , and dp(i)'s result built opon value of dp[i-1], hence by choosing one with maximum total payoff, the function garantees to correctly find the local optimal result. Adding to final accumulated payoff of day n which achieves the full problem.

complexity:
Since on each subproblem, the algorithm only traces and compares constant states of the result on the day before with a constant time. And there are n subproblem (n days) in total, therefore the time complexity is $O(n)$