CS 577: HW 5

Daniel Ko

Summer 2020

§1 Joe is a project manager in a software company called SuperDuo. Joe found out that programmers usually produces the highest quality codes when they are working in pair. However, the productivity of each pair of programmers is the speed of the slower programmer. So Joe needs to figure out the best strategy for pairing the programmers to generate the maximum sum of productivity for his project. Suppose that the number of programmers in Joe's project is even.

§1.1 Give a greedy strategy that maximizes the sum of the productivity of all pairs. You should describe the greedy strategy first and then write it in the form of an algorithm. Also you should analyze the computing complexity of the greedy algorithm.

The greedy strategy is to first sort the programmers in ascending order of productivity. Then, we pair the first two programmers, then the next two, etc. We sum up all the slower programmer in each pair, which will be our maximum sum of productivity.

```
Data: A list gamer of even size n, containing the productivity level of each programmer
Result: The maximum sum of productivity for pair programming
def maxProd(gamer):
    sortedGamer ← Sort gamer in ascending order of productivity
    totalProd ← 0
    for int i = 0; i < n; i+=2:
        totalProd += sortedGamer[i]
    return totalProd
```

The time complexity of maxProd is $O(n \log n)$

Proof. We first sort the list gamer of size n. Using an efficient sorting algorithm such as merge sort, this will take $O(n \log n)$ time. Then, we sum up every other element in the list, which takes $O(n)$ time. The rest of the computations take constant time. Hence, the total time complexity is $O(n \log n + n) = O(n \log n)$ □

§1.2 Prove that this greedy strategy indeed generates the maximum sum of productivity.

We will use a "Greedy Stays Ahead" argument to prove this greedy strategy.

§1.2.1 Define Your Solution

Let $A = \{a_1, a_2, \cdots, a_{n/2}\}$ denote the pairs containing the productivity values of two programmers that our algorithm will generate, i.e. $a_i = (p_1, p_2)$ where p_1 and p_2 are the productivity values of the first and second programmer of the ith pair respectively. Let $O = \{o_1, o_2, \cdots, o_{n/2}\}$ denote the pairs containing the productivity values of two programmers that an optimal algorithm will generate. Let O be ordered in ascending productivity, i.e. if $i > j$, the productivity of pair o_i will be equal to or higher than pair o_j.

§1.2.2 Define Your Measure

For our algorithm, let $\delta(a_1), \cdots, \delta(a_{n/2})$ denote the total productivity of pairs from a_1 to a_i. For example, $\delta(a_3)$ will be the sum of the productivity (the lower productivity between the two programmers) of pairs a_1, a_2, and a_3. Similiary for the optimal algorithm, the $\delta(o_1), \cdots, \delta(o_{n/2})$ denote the total productivity of pairs from o_1 to o_i.

§1.2.3 Prove Greedy Stays Ahead

We show by induction that our greedy stays ahead.

Proof. Let $P(i)$ be the predicate, "$\delta(a_i) \geq \delta(o_i)$" We define $i \in \mathbb{N}^+$.
Base Case: When $i = 1$, we know that a_1 contains the lowest productivity programmer because we paired up the two lowest productivity programmers in our algorithm. We also know that o_1 contains the lowest productivity programmer because we ordered O in ascending productivity. The least productive pair must contain the lowest productivity programmer because the productivity any pair containing the lowest productivity programmer will the productivity value of the lowest productivity programmer. Hence, the productivity of pair a_1 will be the value of lowest productivity programmer, and similarly, the productivity of pair o_1 will be the value of lowest productivity programmer. Thus, $\delta(a_1) \geq \delta(o_1)$ and $P(1)$ holds.
Inductive step: Suppose $P(k-1)$ holds. We show that $P(k)$ holds.

□

slowest has to be first no matter what lol

§1.2.4 Prove Optimality