WaitNoMore

Benjamin Katz

April 2022

1 Algorithm

My algorithm finds the ratio of Weight/Duration and orders those ratios from high to low. It completes the jobs in order of high to low. High ratios are associated with an urgency to print this job for two reasons. One: this is a relatively important job so it should not be delayed. Two: it is a relatively quick job so it does not cause later jobs to have a large increase in wait time. This will minimize the weighted wait times. Because this algorithm makes a local decision based on the ratio and not based on different possible holistic outcomes, it is considered greedy.

2 Proof

Let the optimal algorithm that produce the minimal weighted wait times be Denoted as O

Let $k = \sum allPreviousDurations$

- $O = ...W_i * k + W_{i+1} * (k + t_i) + W_{i+2} * (k + t_{i+1} + t_i) + ...$
- Prove: If the greedy algorithm decides to swap any two elements in O the greedy sum G will either be $<= O$
- Because later terms are only defined by the earlier terms regardless of order, all elements preceding and following the inversion remain unchanged.
- The greedy algorithm would only swap two terms if the ratio of $W_i/t_i < W_{i+1}/t_{i+1}$
- In the case where G performs a swaps it would transform O: $...W_i * k + W_{i+1} * (k + t_i) + W_{i+2} * (k + t_{i+1} + t_i) + ...$ into $...W_{i+1} * k + W_i * (k + t_{i+1}) + W_{i+2} * (k + t_{i+1} + t_i) + ...$
- (Lemma)Because the greedy did the swap we know that $W_{i+1}/t_{i+1} > W_i/t_i$
- Prove that $G <= O$ Factoring out the inversion of O and G : $W_{i+1} * k + W_i * k + W_i * t_{i+1} <= W_i * k + (W_{i+1} * t_i * W_{i+1} * k$

- Simplify: $W_i * t_{i+1} <= W_{i+1} * t_i$
- Divide both sides by $t_i * t_{i+1}$
- We get $W_{i+1}/t_{i+1} >= W_i/t_i$ which we know to be true because of the Lemma
- The swapping of elements in O due to the greedy algorithm maintains the minimized weighted wait time of optimal.
- This concludes the exchange argument QED

3 Time Complexity

The algorithm runs through every element to calculate the ratio which is a $O(n)$ operation. It sorts these weights which is an $O(n \log n)$ operation. It then iterates through this sorted ratios which is an $O(n)$ operation. The sorting dominates so the total time complexity is $O(n \log n)$