Tricks of the Trade (trade)

by Lukas Michel and Tobias Lenz

Subtask 1. $N \leq 200$

In this subtask we have enough time to check each possible segment by computing the sum of the K largest sale values and subtracting from this the sum of the costs of the segment. We can implement this check in $O(N)$ by using the *nth_element* function, but even computing it via sorting in $O(N \log N)$ is fine. Moreover, we can also compute the optimal indices by marking those elements that are at least as large as the K-th largest element of each such segment, where we reset all markings whenever we find a new segment with a higher profit.

Overall the runtime of this solution is $O(N^3)$ or $O(N^3 \log N)$.

Subtask 2. $N \leq 6\,000$

In this subtask we improve the above idea by handling each segment in $O(\log N)$ time. For this, we fix the left endpoint of the segment, and we iterate through every possible right endpoint. At the same time, we keep a priority queue with the K largest elements of the segment, and we also keep track of the sum of these elements as well as the costs of the current segment. This allows us to compute the profit of the current segment in time $O(\log N)$.

To compute the optimal indices, for each segment with the maximum profit that we encounter we can store the segment along with its K-th largest element. We will denote the K-th largest element of segment $[\ell, r]$ by $t_{[\ell,r]}$. An index i is then optimal if and only if there is a maximum profit segment $[\ell, r]$ with $\ell \leq i \leq r$ and $s_i \geq t_{[\ell,r]}$. Computing all such indices can be done efficiently with a minimum segment tree or a sweep line approach, both in time $O(N + S \log N)$ where S is the number of segments with maximum profit.

Overall this leads to a runtime of $O(N^2 \log N)$.

Subtask 3. $K = 2$

For $K = 2$, we observe that in any maximum profit segment $[\ell, r]$ you have to sell the robots ℓ and r to the other contestants.

This allows us to iterate through all possible right endpoints r while we maintain the maximum profit that we can gain from picking any left endpoint $\ell < r$. This profit is $s_r + m_r$ where $m_r := \max_{1 \leq \ell < r} s_\ell - \sum_{i=\ell}^{r} c_i$. We can update m_r in $O(1)$ whenever we move one step to the right by noting that

$$m_r = \max\{m_{r-1} - c_r, s_{r-1} - c_{r-1} - c_r\}.$$

This allows us to compute the maximum profit in $O(N)$.

To compute the optimal indices, we can simply store all right endpoints that are part of a maximum profit segment and then repeat the procedure backwards to obtain all left endpoints of maximum profit segments. Together, these two sets form the set of optimal indices by our observation.

Subtask 4. $K \leq 200$

Let $p(r, k)$ denote the maximum achievable profit if we buy some segment $[\ell, r]$ and sell k robots of this segment. Then, the maximum profit is $\max_{1 \leq r \leq N} p(r, K)$. As base cases, we have $p(r, 0) = 0$ and $p(0, k) = -\infty$ for $k > 0$. Then, we can calculate p recursively for $r, k > 0$ as

$$p(r, k) = \max\{p(r-1, k), p(r-1, k-1) + s_i\} - c_i.$$

This is because we can choose whether or not to sell the i-th robot. This formula can be evaluated in $O(NK)$ using DP.

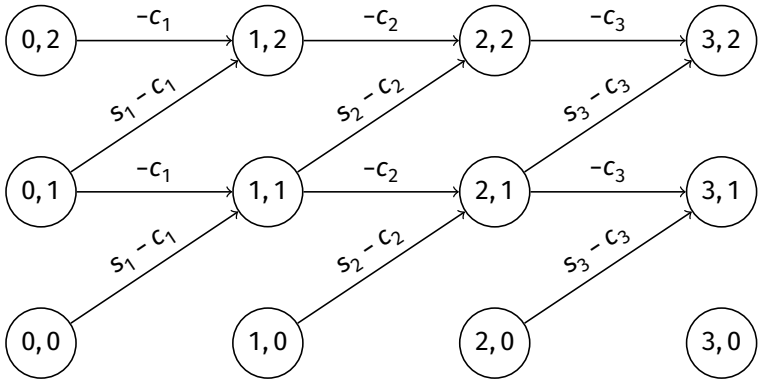
To find the optimal indices, we reconstruct all optimal DP transitions. If a transition $(i, k) \to (i + 1, k + 1)$ appears in an optimal solution, then the i-th robot is part of an optimal transaction.

Subtask 5. No further constraints.

We can visualize the DP of the previous subtask as a directed acyclic graph with nodes (r, k) and edges

- $(i - 1, k) \to (i, k)$ with weight $-c_i$ *and*
- $(i - 1, k) \to (i, k + 1)$ with weight $s_i - c_i$.

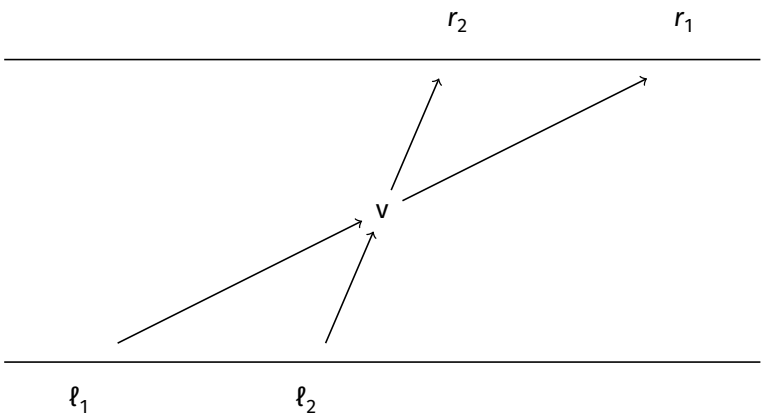
The problem of finding the maximum profit is then equivalent to finding the longest path from some node of the form $(\ell, 0)$ to some node (r, K) in this graph. For example, the graph for $N = 3$ and $K = 2$ looks like this:



For a left endpoint ℓ, we say that a right endpoint $r > \ell$ is ℓ-*optimal* if buying the segment $[\ell, r]$ achieves the maximum profit possible for this fixed left endpoint ℓ.

Lemma 1. *Let $\ell_1 < \ell_2 < r_2 < r_1$ be such that r_1 and r_2 are ℓ_1- and ℓ_2-optimal respectively. Then, r_2 is also ℓ_1-optimal, and r_1 is also ℓ_2-optimal.*

Proof. Let p_1 and p_2 be longest paths corresponding to the intervals $[\ell_1, r_1]$ and $[\ell_2, r_2]$. The paths p_1 and p_2 must intersect at some vertex v of the graph:



The paths from v to r_1 and from v to r_2 must have the same length as we could otherwise replace the part of p_1 or p_2 that comes after v with a longer path. In particular, the path from ℓ_1 to r_2 via v has the same length as p_1, and the same holds for the path from ℓ_2 to r_1 compared to p_2. Hence, buying the segment $[\ell_1, r_2]$ or $[\ell_2, r_1]$ also achieves the maximum profit. □

Let $opt(\ell)$ be the smallest ℓ-optimal right endpoint. If we can compute $opt(\ell)$ for every possible left endpoint ℓ, then the maximum overall profit is the maximum profit of the segments $[\ell, opt(\ell)]$. From the lemma above we get that $opt(\ell) \leq opt(\ell + 1)$.

This means that we can apply the divide and conquer optimization: First, we compute $opt(m)$ for $m = N/2$ by iteratively testing every possible value r. Then, to compute $opt(\ell)$ for $\ell \in [1, m - 1]$, we only need to consider $r \leq opt(m)$ as possible right endpoints, and for $\ell \in [m + 1, N]$ we only check $r \geq opt(m)$. For these intervals, we can apply the idea recursively. In total, this means that we have to check at most $O(N \log N)$ values of r.

However, during this divide and conquer algorithm, we still need an efficient way to compute the maximum profit of a segment $[\ell, r]$. Since we can compute the costs of such a segment with prefix sums (or one of many other ways), we focus on efficiently computing the sum of the K largest elements of this segment. For this, recall our approach from Subtask 2. There, we kept a set of the K largest elements of our current segment $[\ell, r]$ as well as their sum, and we were able to efficiently add elements to this segment.

In our divide and conquer algorithm, we can now always move our current segment $[\ell, r]$ to the segment where we need to know the sum of the K largest elements. However, for this we would also need to be able to remove elements from the segment. Fortunately, this is also possible: in addition to the set with the K largest elements, we can also keep a set with all the other elements of the current segment. If we now delete an element, we can delete it from the appropriate set and rebalance the two sets so that afterwards the top set contains the K largest elements once more.

With the standard analysis of the divide and conquer optimization, we can prove that ℓ and r move at most $O(N \log N)$ steps during this process, and so this algorithm runs in time $O(N \log^2 N)$. It would also have been possible to use persistent segment trees to compute the sum of the K largest elements, which would have resulted in the same complexity.

For a full solution, it remains to compute the optimal indices. In Subtask 2 we already noted that if there are S segments with maximum profit, we could do it in time $O(N + S \log N)$. However, in this subtask, it might hold that $S \in \Theta(N^2)$ which makes this approach too slow. So, we have to reduce the number of segments that we need to consider.

For this, assume that $[\ell_1, r_1]$ and $[\ell_2, r_2]$ are segments with maximum profit with $\ell_1 < \ell_2 < r_2 < r_1$ and that $i \in [\ell_1, r_1]$ is an optimal index. This means that i is one of the K largest elements in one of these two segments. In this case, we know from the lemma that $[\ell_1, r_2]$ and $[\ell_2, r_1]$ are also segments with maximum profit. Since these segments are shorter, it follows that i must also be an optimal index of one of the segments $[\ell_1, r_2]$, $[\ell_2, r_2]$, or $[\ell_2, r_1]$.

In particular, if $\ell_1 < \ell_2$ are left endpoints of segments with maximum profit with no such endpoint in between, for ℓ_1 we only need to consider right endpoints r with $opt(\ell_1) \leq r \leq opt(\ell_2)$ when computing all optimal indices. This means that we will only consider at most $2N$ segments in total, allowing us to compute all optimal indices with a two pointer approach in $O(N \log N)$.