Monsters and Swords

Author and problem setter: Renat Karimov

Subgroup $k = 1$

To solve the subgroup $k = 1$, first introduce a new notation: w_i = the minimum cost of a sword with which it is possible to kill the i-th monster. If there is no such sword for some monster, then the answer is immediately NO.

In this subgroup, we have to kill the monsters one by one in order, so each time we need to find the cost of the cheapest sword that can kill the next monster. This is exactly w_i. Therefore, to solve the problem, we just need to check n times that the number of coins we have left is greater than or equal to the next w_i. The solution works in $O(n \log n + m \log m)$

Subgroup $k = n$

To solve the subgroup with $k = n$, we need to notice that when buying a new sword, we always need to choose a sword with strength greater than the previous one. This is because we have no restriction on sword durability. Then from this fact it follows that in an optimal solution we can always use a sword until it is exhausted. That is, we only need to change swords in the situation when it cannot kill the next monster.

Using these facts, we can write a solution using the dynamic programming technique. Let $dp[i]$ = the maximum number of coins that can remain with the knight after killing the first i monsters (while it does not matter which swords were used to kill the monsters).

For convenience, let $w(i, j) = \max\limits_{i \le l \le j} w[l]$. Then the initial state is: $dp[0] = x$ and the transition is $dp[i] = \max\limits_{j} dp[j] - w(i, j-1) + r(i, j-1)$. At the same time, it must hold that $dp[j] \ge w(i, j-1)$.

To optimize this DP, we can notice that we are not interested in all states of this DP. For each sword, we can identify a prefix of monsters that it can defeat. Then, based on the facts above, we can recompute the DP only through these prefixes, and thus obtain a solution in $O(m^2 + n \log n)$.

For the full solution, we just need to optimize the dynamic programming above; this can be done, for example, using a Cartesian tree structure, then the final solution will work in $O(n \log n + m \log m)$

Subgroup k - arbitrary

To solve the subgroup with arbitrary k, we need to examine the resulting formula in more detail. Let the value i be fixed; then, as j decreases, the value of $w(i, j)$ will increase. Because of this, we would like to use the maximum stack technique. With its help, we can implement the same DP as in the case $k = n$, but with the additional condition that $i - j \le k$. This gives a solution in $O(n \log n + m \log m)$ using a Cartesian tree, HLD, or a sweep with a set.

However, we will discuss a slightly different DP in more detail. Let $dp[i]$ = the minimum amount of gold we need in order to be guaranteed to kill all monsters with indices $i...n-1$ (in increasing order of indices). Then the answer is YES only if $dp[0] \le x$.

Initially: $dp[n-1] = w[n-1]$. Transition: $dp[i] = \min\limits_{i < j \le i+k} (w(i, j-1) + \max(0, dp[j] - r(i, j-1)))$. Let $pr[i] = r(0, i)$ and rewrite the resulting formula a bit; we get $w(i, j-1) + max(0, dp[j] - pr[j-1] + pr[i])$. Then we are interested in the sign of $dp[j] - pr[j-1] + pr[i]$. It is not hard to notice that if now j is fixed and i decreases, then the value of $dp[j] - pr[j-1] + pr[i]$ only decreases, so the sign changes at most once.

Such a DP can be quickly recomputed using a maximum stack and sets, noticing the fact that $dp[i] - pr[i-1] \ge dp[i+1] - pr[i]$.

The final solution will have complexity $O((n+m)\log(n+m))$