Problem H: Scenery

Shortest judge solution: 1753 bytes. Shortest team solution (during contest): N/A bytes.
Python solutions by the judges: none

OK, this is the difficult problem of the set...

This problem really tempts you to try some sort of a greedy solution. Let us see how that would go. Let us process time from the beginning. At any point in time, when we decide to take a photograph, we should – out of all the photographs we can currently take – choose the one for which the end time is the smallest (by a standard interchange argument).

The tricky part is to choose whether you should actually start taking a photograph. Consider an input with two photographs to take. The first one will be available in the range $[0, 5]$, the second in the range $[1, 3]$, and the time to take a single photograph is 2. Then, at time zero, we will fail if we start to take the first photograph. On the other hand, if the second range was $[2, 4]$, we would fail if we didn't start a photograph at zero. So, some sort of smarts are going to be needed.

We will describe two solutions. The first, which we will describe in some detail, comes from the paper "Scheduling Unit-time Tasks With Arbitrary Release Times and Deadlines" by Garey, Johnson, Simons, and Tarjan (SICOMP, 1981). In that paper it is shown that the problem can even be solved in $O(n \log n)$, but we felt that getting a quadratic time solution was hard enough, and that is what we describe here.

We are going to adapt the naive solution above. Notice that in order to make the photograph with the range $[1, 3]$ be even feasible, regardless of all the other photographs, we cannot start any photograph in the open range $(-1, 1)$ – if we do, it will overlap the range $[1, 3]$, and we will not be able to fit $[1, 3]$ in. We will try to generalize that observation.

Take any interval $[s, e]$, and consider all the photographs which become available no earlier than s, and become unavailable no later than e. Then, all these photographs have to fit into the interval $[s, e]$. We will now totally disregard their constraints (except that they have to be taken somewhere in $[s, e]$), and try to take them as late as possible. Note that since we disregard their constraints, they are all identical and we can just schedule them greedily. Let us look at the time C when we started taking the first (earliest) of those photographs. However we schedule these photographs in the interval, the first start time will not be later than C. If $C < s$, then we will simply not be able to take all the photographs. The interesting case is if $C - s < t$. Then, if any photograph gets started in the open interval $(C - t, s)$, we will be unable to take all the photos from the interval $[s, e]$, since we will not be able to take the first of them at C or before.

This means we can mark the interval $(C - t, s)$ as "forbidden", and no photograph can ever be started then. We can also take any forbidden intervals we found so far into account when doing the greedy scheduling from the back (which might help us create more forbidden intervals).

The full algorithm works as follows. We order all the availability times of all the photographs from latest to earliest. For each such time s, we iterate over all the end times e of the photographs, and for each interval $[s, e]$ we run the algorithm above – take all the photographs that have to be taken fully within $[s, e]$, ignore other constraints on them, assign times to them as late as possible (taking into account already created forbidden regions), and find the time C, which is the earliest possible start time of the first of those photographs. If $C < s$, return NO, and if $C - s < t$, produce a new forbidden region ending in s.

The above, implemented naively, is pretty slow. First, notice that while as written we can have a quadratic number of forbidden regions, it is trivial to collapse all the forbidden regions ending at any s into one (the one corresponding to the smallest C). Also, note that when we move from s to the previous s', we don't have to run the time assignment from scratch – we can just take the previous C value, and if the photograph starting at s' ends before some e, we just have to add one more photograph before the C (by default, at $C - t$, but possibly earlier if $C - t$ is in a forbidden region). This allows us to progress through this stage in $O(n^2)$ time.

Then, after this is done, we just run the standard "greedy" algorithm, taking the forbidden regions into account. The tricky part, of course, is to prove that this actually solves the problem. Obviously, if the greedy algorithm succeeds in taking the photographs, the answer is YES. Now, we will prove that if the greedy algorithm ends up taking a photograph after its end time, then we would have actually failed in the first phase.

Assume the greedy algorithm did take a photograph incorrectly. First, notice that if there are idle times (that is, times when no photograph is being taken by the greedy), we can assume they are all within forbidden regions. If there was an idle period without a forbidden region, let's say at time X, it means that we have had no photograph available to take. So, all the photographs that were available before X got assigned and ended before X, and so they do not affect the photographs that become available after X. So, we can just remove these photographs from the set altogether and solve the smaller problem.

Now, assume the greedy algorithm failed. Take the first photograph P_i, with a deadline of e_i, that got scheduled so it ends after e_i. If no photograph that got scheduled earlier has a deadline later than e_i, it means that the photographs up to P_i actually failed to fit into the range s_0, e_i – and so for that interval the first phase would've returned NO. If there is a photograph that has a deadline later than e_i, let P_j be the latest of those photographs. Consider all the photographs $P_{k_1}, \ldots, P_{k_l}$ that got scheduled between P_j and P_i, and let s be the earliest time at which any of those photographs (including P_j) became available. Consider the first phase for the interval s, e_i – it had to have failed (because the greedy solution failed to put these photographs into this interval).

Thus, in all the cases where the greedy algorithm fails, the first phase had to have failed for some interval as well – so, if the first phase finished successfully, we are guaranteed the greedy algorithm will return a correct answer.

An alternative approach, which we will not prove, is an approach where we modify a naive branching approach. Naively, we can process forward through time, and maintain a set of possible states, where a possible state is the time when the photograph currently being taken (if any) ends, and the set of photographs that are available to take, but not yet started. This set of states is potentially exponential in size. We branch out (from all the states where there is no photograph being taken) when a new photograph becomes available, and we branch out when a photograph taking ends (to either not start a new photograph, or to start the one with the earliest deadline from those in the state).

However, it is provable that we can actually have only one state where no photograph is being taken – when the algorithm ends up produces a new "nothing runs" state because some photograph just finished, one of the two states is strictly worse than the other. The proof, and the details of turning this into a quadratic algorithm, are left as an exercise to the reader.