Event Hopping (events)

BY EVGENY VIHROV (LATVIA)

Subtask 1. For every event, you can switch to at most one other event.

Consider a directed graph where events correspond to nodes and there is an edge from event i to event j iff you can switch from i to j. This graph can be efficiently constructed in $\mathcal{O}(N \log N)$ with a sweep line since there are at most N edges.

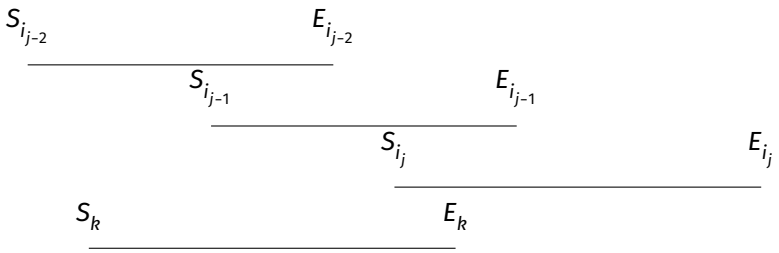
For simplicity, assume that there is only one component. Note that there is a case that prevents the graph from being a directed tree. If there are two events a and b with the same endpoint, i.e. $E_a = E_b$ and $S_a \leq S_b$, we have to handle queries that end at the shorter event b separately. In any optimal sequence of events, we first visit event a and then event b. Therefore, we can solve this special case by replacing queries of the form $(-, b)$ with $(-, a)$. This allows us to reduce the graph to a directed tree.

To answer a query (s, e), we need to check that e is a parent of s and then compute the minimum number of event switches using the depth of e and s. We can precompute the depth of each node in linear time with a DFS and with an inorder traversal of the tree we can answer in constant time whether e is a parent of s or not. Therefore, the whole precomputation can be done in linear time and we can solve this subtask in $\mathcal{O}(N \log N + Q)$.

Subtask 2. $N \leq 1\,000$ and $Q \leq 100$

This subtask can be solved by brute force. If we do a naive BFS for each query, we get a solution with a runtime of $\mathcal{O}(QN^2)$.

For the following subtasks we need an important observation. Assume that we attend an event s and want to attend an event e. Let $s = i_1, i_2, \ldots, i_l = e$ be a sequence of events with a minimum number of event switches. We can prove that if $j > 2$, we can always replace in this sequence event i_{j-1} with event k such that it is possible to switch from k to i_j and S_k is the smallest possible. Note that $E_{i_{j-2}} < S_{i_j}$ (otherwise we could switch directly from i_{j-2} to i_j) and $S_{i_{j-1}} \leq E_{i_{j-2}}$ (since we can switch from i_{j-2} to i_{j-1}). Combining it with the fact that $S_k \leq S_{i_{j-1}}$ and $E_k \geq S_{i_j}$, we can conclude that it is possible to switch from i_{j-2} to k.



Subtask 3. $N \leq 5\,000$

This subtask can be solved by precomputing all possible answers. How do we do this faster than $\mathcal{O}(N^3)$?

Let us assume that all events are sorted in non-decreasing order by their end time and let switches(j, i) be the minimum number of event switches if we attend event j and want to attend event i. We now compute all answers with a fixed ending event i by sweeping over the events in decreasing order of their index. Furthermore, we use a set where we store the pairs (switches$(j, i), j$) for all events j to which we can currently switch. If we want to find switches(k, i), we have to find an event j with minimal switches(j, i), so that we can switch from k to j. We can quickly find this event using our set. Thus, it is possible to precompute all answers in $\mathcal{O}(N^2 \log N)$ and answer queries in constant time.

It is also possible to avoid a set and improve the runtime to $\mathcal{O}(N^2)$ by noticing that switches$(-, i)$ is a monotone function.

Alternatively, we can use some slow precomputation to find the event k from the observation above and answer queries using binary jumping.

Subtask 4. $Q \leq 100$

This subtask can be solved by precomputing for each event i the event k such that it is possible to switch from k to i and S_k is the smallest possible. To find event k fast, we can use a segment tree/sparse table that allows us to query the minimum start time of all events that end in $[S_i, E_i]$. Alternatively, we can iterate over all events in non-decreasing order of their end time and use a monotonic stack to find event k with a binary search.

We can now answer each query (s, e) in linear time by starting at event e and „switching backward“ with the precomputed information until we reach event s. This leads to a solution with complexity $\mathcal{O}(QN)$.

Subtask 5. No event is completely contained in another event, i.e. there are no two events $i \neq j$ with $S_i \leq S_j < E_j \leq E_i$.

Assume that we want to answer a query (s, e). It is always optimal to switch to an event k with maximum E_k. The constraints in this subtask ensure that we can apply the same argument as in the above observation. Therefore, we can find with a sweep line for each event i the event k such that we can switch from i to k and E_k is the largest possible. We can now compute an array next$[m][i]$ that stores the event that we reach if we start at event i and perform 2^m event switches. This allows us to answer queries in $\mathcal{O}(\log N)$ using binary jumping.

Subtask 6. No further constraints.

To solve this subtask, we combine the solutions of the previous two subtasks. That means we first precompute the event k such that S_k is the smallest possible and we can switch from event k to event i using a sweep line and a segment tree/sparse table. Then we compute an array prev$[m][i]$ that stores the event that we reach if we start at event i and perform 2^m 'backward event switches.' The final complexity is $\mathcal{O}((N + Q) \log N)$.