Problem

Given a sequence S with N elements.

We need to find a subsequence with 4 elements with pattern:

A B A B

where \( A \neq B \)

Solution

Let's first consider a brute force solution, where we look at:

- All pairs of indices \((i, j)\) where \( S_i = S_j \). There are \( O(N^2) \) such pairs.
- All pairs of indices \((u, v)\) where \( S_u = S_v \). There are \( O(N^2) \) such pairs.

There are \( O(N^4) \) indices \( i j u v \). If \( i < u < j < v \), then we have found a solution.

We observe that if \( i < u < j < v \) forms a solution, then any \( i' \) satisfying \( S_{i'} = S_i \) and \( i' < i \) also forms a solution, since \( i' < i < u < j < v \) and \( S_{i'} = S_i = S_j \).

Thus, instead of looking at all \( O(N^2) \) pairs of indices \((i, j)\), we only look at all the pairs \((i_{\text{min}}, j)\) where \( i_{\text{min}} \) is the minimum index such that \( S_{i_{\text{min}}} = S_j \). There is only \( O(N) \) pairs of \((i_{\text{min}}, j)\).

Similarly, we only need to look at the pairs of indices \((u, v_{\text{max}})\) where \( v_{\text{max}} \) is the maximum index such that \( S_u = S_{v_{\text{max}}} \). There are also \( O(N) \) such pairs.

Thus, we have improved our solution to \( O(N^2) \) with some pre-processing:

- For each value \( x \), stores the smallest index \( i_{\text{min}}(x) \) where \( S_{i_{\text{min}}(x)} = x \), and the largest index \( i_{\text{max}}(x) \) where \( S_{i_{\text{max}}(x)} = x \).
- Loop through all index \( j \) and \( u \). Let \( i = i_{\text{min}}(S_j) \) and \( v = i_{\text{max}}(S_u) \). If \( i < u < j < v \) and \( A_u \neq A_j \), then we have found a solution.

Improve to \( O(N \times \log N) \)

We re-state the problem as follows:

- Given \( O(N) \) segments \([i, j]\).
- Given \( O(N) \) queries \((u, v)\). We need to check if there exist any segment such that \( i < u < j < v \).
This problem can be solved efficiently as follows:

- For each segment \([i, j]\), we create 2 events:
  - At \(time = i\), we add a new segment \([i, j]\) to our data structure.
  - At \(time = j\), we remove the segment \([i, j]\) from our data structure. Note that this segment must be previously added.
- For each query \((u, v)\), we create 1 event:
  - At \(time = u\), we query if there is a segment \([i, j]\) in our data structure, such that:
    - \(A_u \neq A_j\)
    - \(j < v\)

We sort all events according to time. This will make sure that we do not need to check for the condition \(i < u < j\), since the segment will only exist in our data structure at the time of query iff \(i < u < j\).

To check efficiently if there is at least one segment with \(j < v\), we can store segments in a Segment Tree.