Kick Start 2019 - Round B

Analysis: Diverse Subarray

Consider finding the optimal solution that is a prefix of the input. That is, let us only consider taking ranges of trinkets [1,r] for any r. This reduces the problem to finding prefix sum maximums. For example, consider this sequence of trinkets:
1, 1, 2, 1, 3, 2, 3, 2, 2
Assume that **S**=2.
The first and second occurrence of a type can be modeled as a +1 event since we can take the trinkets. The **S**+1th occurrence can be a -**S** event, since now we cannot take any trinkets of that type. Finally, any further occurrence of that type can be a +0 since we still cannot take them. Thus, we have the following events:
+1, +1, +1, -2, +1, +1, +1, -2, +0
A prefix sum ending at index r in our events corresponds to the score for taking the subarray [1,r]. This idea can be generalized to any value of **S**, since all events until the **S**+1th can be +1 events.

Test set 1 (Visible)

A naive algorithm can be realized using the above strategy. It can be applied to each potential left point (instead of always starting at 1). This explicitly considers every possible range [l,r] and computes the number of trinkets. The events can be calculated by keeping track of how many times each type has occurred while sweeping across the trinkets and keeping a running sum of events. The count of types can be handled with a frequency table. Any map from type to count can implement this, for example, an array or hashtable. The strategy uses O(**N**) time to process the solution starting at a particular left point. There are O(**N**) left points. Thus, the time complexity is O($\mathbf{N}^2$). This is sufficient for Test set 1.

Test set 2 (Hidden)

The previous algorithm will not be fast enough for the Test set 2. Observe that the events for the lth left point are closely related to those for the l+1th left point. In particular, only events corresponding to the type of the trinket at index l change. In fact, only the -**S** for that type changes. It moves over to the next +0 event for that type if it exists. The previous -**S** event becomes a +1. This means that the events only change at a constant number of places.

Using this observation, we can think about this as a data structure problem. We require a data structure that supports two operations:

1. Change the value at an index.
2. What is the maximum prefix sum?

This can be used to maintain our events and find the best solution for each left point respectively. These operations can be achieved using a modified segment tree. Each node in the segment tree can be modified to store the sum of the elements covered by that node. Also, each node should store the maximum sum of any prefix of elements that are covered by that node. Such a tree can support both operations in O(log(**N**)) time. We can achieve a final complexity of O(**N** log(**N**)).