Solution for task ‘candies‘

1 Subtask 1

Perform a naive simulation of the operations.

2 Subtask 2

Use prefix sums to determine the number of candies in each box as if the capacity was infinite. Take the minimum of this value and the capacity of the box.

3 Subtask 3

Use a lazy propagation segment tree to maintain the minimum and maximum for each range. However, we need to make sure that negative values are reset to zero. For a given segment tree node, let a represent the minimum value of that node, and b represent the maximum value of that node.

- Case 1: $a \geq 0$. In this case, all values in that node are already correct. Return immediately.
- Case 2: $b < 0$. In this case, we update the value of this node by $-b$, and proceed to case 3.
- Case 3: $a < 0, b \geq 0$. Recursively update the children.

A similar procedure can be done to ensure that the total number of candies stays below the capacity.

It appears that this procedure can take very long to return. Indeed, if all the values are different, this procedure can take up to $O(n \log n)$ time for a single update. However, the amortized runtime is $O(\log^2 n)$ per update. The amortized runtime can be proven by keeping track of the number of boxes which have a different number of candies from its neighbor.

4 Subtask 4

The key idea behind this subtask can be summarized as follows:

- Find the last point in time in which the box was empty or full.

Let's begin with some observations. Define $p[i] = v[0] + v[1] + \ldots + v[i]$. If

$$\max(p[i], p[i+1], \ldots, p[q-1]) - \min(p[i], p[i+1], \ldots, p[q-1]) \geq c[j] \quad (1)$$

Then there exists a point in time after time i where box j was either full or empty. For each box j, we can binary search for the largest i in which (1) holds. From here, we can calculate the last point in time in which the box was empty or full.

5 Full solution

The full solution is very similar to that of subtask 4.

The main difference is that the values of $p[i]$ are no longer the same for every box. To deal with this, we use a lazy propagation segment tree to update the values of $p[i]$ as the boxes change. This reduces the time complexity to $O(n \log^2 q + q \log q)$. By doing the binary search on the segment tree directly, it can be reduced to $O(n \log q + q \log q)$, but this is not required for the full solution.