Smallest Range Covering Elements From K Lists

solved by	Senan
Platform	LeetCode
difficulty	Hard
# Serial	632
tags	Heaps Sliding Window Vector
language	C++
solved on	@13/10/2024
link	https://leetcode.com/problems/smallest-range-covering-elements-from-k-lists/description/
Completion	☑

Intuition

The problem asks to find the smallest range that includes at least one number from each list. The idea is to treat this as a merging problem where you consider all elements from all lists as a combined stream, then find the smallest subarray (range) in that merged list such that it covers all the lists.

Approach

1. **Merging Lists**: First, flatten all the lists by keeping track of which list each number belongs to. We do this by creating a list of pairs, where each pair consists of a number and its corresponding list index.
2. **Sorting**: Next, sort this flattened list by the numbers, which helps in searching for ranges.
3. **Sliding Window**: Use the sliding window approach to maintain a valid range that includes at least one number from each list. Expand the window until all lists are covered, and then contract it while maintaining the coverage, updating the best (smallest) range found.
4. **Frequency Map**: Use a frequency map to count how many numbers from each list are present in the current window.

Complexity

Time Complexity:

- **Merging and Sorting**: Flattening all the lists takes $O(n)$, where $O(n)$ is the total number of elements across all lists. Sorting the merged list takes $O(n \log n)$.
- **Sliding Window**: The sliding window approach involves iterating through the merged list once, which takes $O(n)$.

Thus, the overall time complexity is $O(n \log n)$, where n is the total number of elements across all lists.

Space Complexity:

- We store the merged list and the frequency map, so the space complexity is $O(n)$, where n is the total number of elements across all lists.

Code

```
vector<int> smallestRange(vector<vector<int>>& lists) {
    vector<pair<int, int>> merge;
    for (int i = 0; i < lists.size(); i++) {
        for (int num : lists[i]) {
            merge.push_back({num, i});
        }
    }

    sort(merge.begin(), merge.end());

    unordered_map<int, int> freq;
    int start = 0, coveredLists = 0;
    int bestStart = 0, bestEnd = INT_MAX;

    for (int end = 0; end < merge.size(); end++) {
        freq[merge[end].second]++;
        if (freq[merge[end].second] == 1) {
            coveredLists++;
        }

        while (coveredLists == lists.size()) {
            int currentRange = merge[end].first - merge[start].first;
            if (currentRange < bestEnd - bestStart) {
                bestStart = merge[start].first;
                bestEnd = merge[end].first;
            }

            freq[merge[start].second]--;
            if (freq[merge[start].second] == 0) {
                coveredLists--;
            }
            start++;
        }
    }

    return {bestStart, bestEnd};
}
```