Smallest Range

You have `k` lists of sorted integers in ascending order. Find the **smallest** range that includes at least one number from each of the `k` lists.

We define the range [a,b] is smaller than range [c,d] if `b-a or a if b-a == d-c .`

Example 1:

```
Input: [[4,10,15,24,26], [0,9,12,20], [5,18,22,30]]
Output: [20,24]
Explanation:
List 1: [4, 10, 15, 24,26], 24 is in range [20,24].
List 2: [0, 9, 12, 20], 20 is in range [20,24].
List 3: [5, 18, 22, 30], 22 is in range [20,24].
```

Note:

1. The given list may contain duplicates, so ascending order means >= here.
2. 1 k
3. -10^5 value of elements 5.
4. **For Java users, please note that the input type has been changed to List<List<Integer>>. And after you reset the code template, you'll see this point.**

Solution 1

Image you are merging k sorted array using a heap. Then everytime you pop the smallest element out and add the next element of that array to the heap. By keep doing this, you will have the smallest range.

```
public int[] smallestRange(int[][] nums) {
  PriorityQueue<Element> pq = new PriorityQueue<Element>(new Comparator<Element>() {
   public int compare(Element a, Element b) {
    return a.val - b.val;
   }
  });
  int min = Integer.MAX_VALUE, max = Integer.MIN_VALUE;
  for (int i = 0; i < nums.length; i++) {
   Element e = new Element(i, 0, nums[i][0]);
   pq.offer(e);
   max = Math.max(max, nums[i][0]);
  }
  int range = Integer.MAX_VALUE;
  int start = -1, end = -1;
  while (pq.size() == nums.length) {

   Element curr = pq.poll();
   if (max - curr.val < range) {
    range = max - curr.val;
    start = curr.val;
    end = max;
   }
   if (curr.idx + 1 < nums[curr.row].length) {
    curr.idx = curr.idx + 1;
    curr.val = nums[curr.row][curr.idx];
    pq.offer(curr);
    if (curr.val > max) {
     max = curr.val;
    }
   }
  }

  return new int[] { start, end };
 }

 class Element {
  int val;
  int idx;
  int row;

  public Element(int r, int i, int v) {
   val = v;
   idx = i;
   row = r;
  }
 }
```

written by xmztzt original link here

Solution 2

Yes. The idea is just similar to Merge K Sorted List. Keep a priority queue of iterators/pointers which points to the current head of a row.

```
class Solution {
public:
    vector<int> smallestRange(vector<vector<int>>& nums) {
        typedef vector<int>::iterator vi;

        struct comp {
            bool operator()(pair<vi, vi> p1, pair<vi, vi> p2) {
                return *p1.first > *p2.first;
            }
        };

        int lo = INT_MAX, hi = INT_MIN;
        priority_queue<pair<vi, vi>, vector<pair<vi, vi>>, comp> pq;
        for (auto &row : nums) {
            lo = min(lo, row[0]);
            hi = max(hi, row[0]);
            pq.push({row.begin(), row.end()});
        }

        vector<int> ans = {lo, hi};
        while (true) {
            auto p = pq.top();
            pq.pop();
            ++p.first;
            if (p.first == p.second)
                break;
            pq.push(p);

            lo = *pq.top().first;
            hi = max(hi, *p.first);
            if (hi - lo < ans[1] - ans[0])
                ans = {lo, hi};
        }
        return ans;
    }
};
```

written by Aeonaxx original link here

Solution 3

The idea is to sort all the elements in the k lists and run a sliding window over the sorted list, to find the minimum window that satisfies the criteria of having atleast one element from each list.

```
public static int[] smallestRange(List<List<Integer>> nums) {
        List<int[]> list = IntStream.range(0, nums.size())
                .mapToObj( i -> nums.get(i).stream().map(x -> new int[]{x, i}))
                .flatMap(y -> y)
                .sorted(Comparator.comparingInt(p -> p[0])).collect(toList());
        int[] counts = new int[nums.size()];
        BitSet set = new BitSet(nums.size());
        int start = -1;
        int[] res = new int[2];
        for(int i = 0; i < list.size(); i++) {
            int[] p = list.get(i);
            set.set(p[1]);
            counts[p[1]] += 1;
            if(start == -1) { start = 0; }
            while(start < i && counts[list.get(start)[1]] > 1) {
                counts[list.get(start)[1]]--;
                start++;
            }
            if(set.cardinality() == nums.size()) {
                if( (res[0] == 0 && res[1] == 0) || (list.get(i)[0] - list.get(star
t)[0]) < res[1] - res[0]) {
                    res[0] = list.get(start)[0];
                    res[1] = list.get(i)[0];
                }
            }
        }
        return res;
    }
```

written by johnyrufus16 original link here

From Leetcoder.