**Algorithms Memo (LeetCode Problem #)**

目次

[**共通のHashMap高速探索 (49)** 1](#_Toc142776374)

[**HashMapのvalueによる個数数え上げ (347)** 1](#_Toc142776375)

[**Priority Queue順位付け変更 (347)** 2](#_Toc142776376)

[**HashSetによる高速数列探し (128)** 2](#_Toc142776377)

[**挟み込み (11, 42)** 2](#_Toc142776378)

[**左から見た可能性、右から見た可能性、総合 (42)** 3](#_Toc142776379)

# **共通のHashMap高速探索 (49)**

|  |
| --- |
| HashMap<String, List<String>> map = new HashMap<>();  for (String s : strs) {      int[] hash = new int[26];      for (char c : s.toCharArray()) {          hash[c - 'a']++;      }      String key = new String(Arrays.toString(hash));      map.computeIfAbsent(key, k -> new ArrayList<>());      map.get(key).add(s);  } |

# **HashMapのvalueによる個数数え上げ (347)**

|  |
| --- |
| for (int num : nums) map.put(num, map.getOrDefault(num, 0) + 1); |

# **Priority Queue順位付け変更 (347)**

※Valueによる並び替え(降順)

|  |
| --- |
| PriorityQueue<Map.Entry<Integer, Integer>> pq = new PriorityQueue<>((a, b) -> b.getValue() - a.getValue()); |

※元々Priority Queueは昇順での並びで先頭が決まるのでb – aとし、降順を実装した。

# **HashSetによる高速数列探し (128)**

|  |
| --- |
| HashSet<Integer> set = new HashSet<>();  int ans = 1;  for (int num : nums) set.add(num);  for (int num : nums) {      if (!set.contains(num - 1)) {          int count = 1;          while (set.contains(num + 1)) {              num++;              count++;          }          ans = Math.max(count, ans);      }  } |

if (!set.contains(num - 1))

により、数列の最初の項のみの処理にできる

※双方向で考える必要がない

# **挟み込み (11, 42)**

|  |
| --- |
| int l = 0;  int r = height.length - 1;  while (l < r) {      if (height[l] < height[r]) {          int pre = height[l];          while (l < r && height[l] <= pre) {              l++;          }      } else {          int pre = height[r];          while (l < r && height[r] <= pre) {              r--;          }      }  } |

両側から挟み込み、条件を減らしていく

# **左から見た可能性、右から見た可能性、総合 (42)**

|  |
| --- |
| int left[] = new int[heights.length], right[] = new int[heights.length], max =      heights[0], c = 0;  for (int i = 0; i < heights.length; i++) {      left[i] = Math.max(heights[i], max);      max = left[i];  }  max = heights[heights.length - 1];  for (int i = heights.length - 1; i >= 0; i--) {      right[i] = Math.max(heights[i], max);      max = right[i];  }  for (int i = 0; i < heights.length; i++) {      c = c + Math.min(left[i], right[i]) - heights[i];  } |

片側ずつ、可能性の最大を出し、総合して結果を出す。