**Algorithms Memo (LeetCode Problem #)**

目次

[**共通のHashMap高速探索 (49)** 1](#_Toc144078407)

[**HashMapのvalueによる個数数え上げ (347)** 2](#_Toc144078408)

[**Priority Queue順位付け変更 (347)** 2](#_Toc144078409)

[**HashSetによる高速数列探し (128)** 2](#_Toc144078410)

[**挟み込み (11, 42)** 3](#_Toc144078411)

[**左から見た可能性、右から見た可能性、総合 (42)** 3](#_Toc144078412)

[**Tree全探査 by Queue (226)** 4](#_Toc144078413)

[**Sliding Window 何も起きない場合は単にスライド (424)** 4](#_Toc144078414)

[**配列の比較** 5](#_Toc144078415)

[**char to int** 5](#_Toc144078416)

[**map.size()を利用した、一致した要素数え上げ** 5](#_Toc144078417)

# **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。

# **Tree全探査 by Queue (226)**

|  |
| --- |
| if (root == null) return root;  Queue<TreeNode> que = new ArrayDeque<>();  que.add(root);  while (!que.isEmpty()) {      TreeNode cur = que.poll();      if (cur.left != null) {          que.add(cur.left);      }      if (cur.right != null) {          que.add(cur.right);      }      //do something to cur  } |

# **Sliding Window 何も起きない場合は単にスライド (424)**

|  |
| --- |
| int ans = 0;          int[] maxes = new int[26];          int max = 0;          int l = 0;          for (int i = 0; i < s.length(); i++) {              int num = pint(s.charAt(i));              maxes[num] += 1;              if (maxes[num] > max) {                  max = maxes[num];              }              if (i - l + 1 - max > k) {                  maxes[pint(s.charAt(l))]--;                  l++;              }              ans = Math.max(ans, i - l + 1);          } |

# **配列の比較**

|  |
| --- |
| int[] mp1 = new int[26];  int[] mp2 = new int[26];  //do something  st1.equals(Arrays.toString(mp2)) |

# **char to int**

|  |
| --- |
| public int cint(char c) {  return (int)(c - 'a');  } |

# **map.size()を利用した、一致した要素数え上げ**

|  |
| --- |
| if (map.containsKey(right)) {      map.put(right, map.get(right) - 1);      if (map.get(right) == 0) matched++;  }  if (matched == map.size()) |