**Algorithms Memo (LeetCode Problem #)**

目次

[**共通のHashMap高速探索 (49)** 3](#_Toc146230586)

[**HashMapのvalueによる個数数え上げ (347)** 3](#_Toc146230587)

[**Priority Queue順位付け変更 (347)** 3](#_Toc146230588)

[**HashSetによる高速数列探し (128)** 3](#_Toc146230589)

[**挟み込み (11, 42)** 4](#_Toc146230590)

[**左から見た可能性、右から見た可能性、総合 (42)** 4](#_Toc146230591)

[**Tree全探査 by Queue (226)** 5](#_Toc146230592)

[**Sliding Window 何も起きない場合は単にスライド (424)** 5](#_Toc146230593)

[**配列の比較** 6](#_Toc146230594)

[**char to int** 6](#_Toc146230595)

[**map.size()を利用した、一致した要素数え上げ** 6](#_Toc146230596)

[**Dequeを使って、次候補の確保** 6](#_Toc146230597)

[**注意) Integer等Object Data Typeの比較** 7](#_Toc146230598)

[**Stackを使った深さ優先探索** 7](#_Toc146230599)

[**StackやQueueに要素ではなくindex保存** 8](#_Toc146230600)

[**Tree Mapによる二要素Sort** 8](#_Toc146230601)

[**Review:** 8](#_Toc146230602)

# **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()) |

# **Dequeを使って、次候補の確保**

|  |
| --- |
| Deque<Integer> q = new LinkedList<>();  for (int i = 0; i < n; i++) {      if (!q.isEmpty() && q.peekFirst() < i - k + 1) q.pollFirst();      while (!q.isEmpty() && nums[q.peekLast()] < nums[i]) q.pollLast();      q.offer(i);      if (i >= k - 1) ans[j++] = nums[q.peekFirst()];  } |

# **注意) Integer等Object Data Typeの比較**

|  |
| --- |
| List<Integer> stack;      List<Integer> minValues = new LinkedList<>();  //input  stack.get(0).equals(minValues.get(0)) |

# **Stackを使った深さ優先探索**

|  |
| --- |
| private void backtrack(int openN, int closedN, int n) {      if (openN == closedN && closedN == n) {          Iterator vale = stack.iterator();          String temp = "";          while (vale.hasNext()) {              temp = temp + vale.next();          }          res.add(temp);      }      if (openN < n) {          stack.push('(');          backtrack(openN + 1, closedN, n);          stack.pop();      }      if (closedN < openN) {          stack.push(')');          backtrack(openN, closedN + 1, n);          stack.pop();      }  } |

# **StackやQueueに要素ではなくindex保存**

|  |
| --- |
| Stack<Integer> stack = new Stack<>();  for (int currDay = 0; currDay < temperatures.length; currDay++) {      while (          !stack.isEmpty() &&          temperatures[currDay] > temperatures[stack.peek()]      ) {          int prevDay = stack.pop();          ans[prevDay] = currDay - prevDay;      }      stack.add(currDay);  } |

# **Tree Mapによる二要素Sort**

|  |
| --- |
| Map<Integer, Double> mp = new TreeMap<>();  for (int i = 0; i < position.length; i++) {      double time = (target - position[i]) \* 1.0 / speed[i];      mp.put(Integer.valueOf(position[i]), Double.valueOf(time));  } |

# **Review:**

Sliding Window: 239