|  |  |  |  |
| --- | --- | --- | --- |
| 283 | [Move Zeroes](https://leetcode.com/problems/move-zeroes/) | 49.2% | Easy |

Copy non-zeros to the beginning, and then insert zeroes at the end.

|  |  |  |  |
| --- | --- | --- | --- |
| 325 | [Maximum Size Subarray Sum Equals k](https://leetcode.com/problems/maximum-size-subarray-sum-equals-k/) | 42.1% | Medium |

Hash table of (sum(0,end\_index), end\_index), look for curr\_sum-target in hash table.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 301 | [Remove Invalid Parentheses](https://leetcode.com/problems/remove-invalid-parentheses/) | 35.0% | Hard |  |

1. BFS on number of parentheses removed with visited hash set and linked list queue. Check if each is valid.
2. Calculate # removable left/right parenthesis and count # open parenthesis to avoid invalid string. DFS on each char from left that either use it or not use it if it is parenthesis (o.t. append it).

if (c == '(') {

dfs(s, i + 1, res, sb, rmL - 1, rmR, open); // not use (

dfs(s, i + 1, res, sb.append(c), rmL, rmR); // use (

} else if (c == ')') {

dfs(s, i + 1, res, sb, rmL, rmR - 1, open); // not use )

dfs(s, i + 1, res, sb.append(c), rmL, rmR, open - 1); // use )

} else {

dfs(s, i + 1, res, sb.append(c), rmL, rmR, open);

}

|  |  |  |  |
| --- | --- | --- | --- |
| 67 | [Add Binary](https://leetcode.com/problems/add-binary/) | 31.6% | Easy |

Start from right to left bitwise.

Sum = a + b + carry

sb.append(Integer.toString(sum & 1));

carry = (sum >> 1) ;

|  |  |  |  |
| --- | --- | --- | --- |
| 311 | [Sparse Matrix Multiplication](https://leetcode.com/problems/sparse-matrix-multiplication/) | 50.6% | Medium |

A sparse matrix can be represented as a sequence of rows, each of which is a sequence of (column-number, value) pairs of the nonzero values in the row.

for(int i = 0; i < m; i++) {

for(int k = 0; k < n; k++) {

if (A[i][k] != 0) {

for (int j = 0; j < nB; j++) {

if (B[k][j] != 0) C[i][j] += A[i][k] \* B[k][j];

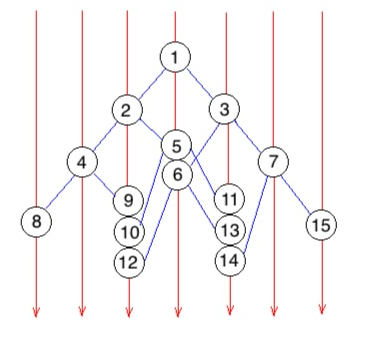
}

}

}

}

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 314 | [Binary Tree Vertical Order Traversal](https://leetcode.com/problems/binary-tree-vertical-order-traversal/) | 36.1% | Medium |  |

BFS instead of DFS to keep order of each list (downwards).

Keep queue of both nodes and column # corresponded.

map.get(col).add(node.val);

if (node.left != null) {

q.add(node.left);

cols.add(col - 1);

min = Math.min(min, col - 1);

}

if (node.right != null) {

q.add(node.right);

cols.add(col + 1);

max = Math.max(max, col + 1);

}

|  |  |  |  |
| --- | --- | --- | --- |
| 273 | [Integer to English Words](https://leetcode.com/problems/integer-to-english-words/) | 21.7% | Hard |

1. Index array instead of if statements

**private** **final** String[] LESS\_THAN\_20 = {"", "One", "Two", "Three", "Four", "Five", "Six", "Seven", "Eight", "Nine", "Ten", "Eleven", "Twelve", "Thirteen", "Fourteen", "Fifteen", "Sixteen", "Seventeen", "Eighteen", "Nineteen"};

**private** **final** String[] TENS = {"", "Ten", "Twenty", "Thirty", "Forty", "Fifty", "Sixty", "Seventy", "Eighty", "Ninety"};

**private** **final** String[] THOUSANDS = {"", "Thousand", "Million", "Billion"};

1. Handle piece of length 3 (num%1000, num \=1000)

**if** (num == 0)

**return** "";

**else** **if** (num < 20)

**return** LESS\_THAN\_20[num] + " ";

**else** **if** (num < 100)

**return** TENS[num / 10] + " " + helper(num % 10);

**else**

**return** LESS\_THAN\_20[num / 100] + " Hundred " + helper(num % 100);

|  |  |  |  |
| --- | --- | --- | --- |
| 17 | [Letter Combinations of a Phone Number](https://leetcode.com/problems/letter-combinations-of-a-phone-number/) | 33.7% | Medium |

1. Iterative (BFS)

For each digit added, remove and copy every element in the queue and add the possible letter to each element, then add the updated elements back into queue again. Repeat this procedure until all the digits are iterated.

**for**(int i =0; i<digits.length();i++){

int x = Character.getNumericValue(digits.charAt(i));

**while**(ans.peek().length()==i){

String t = ans.remove();

**for**(char s : mapping[x].toCharArray())

ans.add(t+s);

}

}

1. Recursive (DFS)

**private** **void** **combination**(String prefix, String digits, **int** offset, List<String> ret)

**if** (offset >= digits.length()) {

ret.add(prefix);

**return**;

}

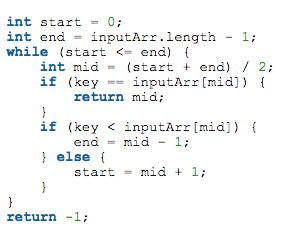
String letters = KEYS[(digits.charAt(offset) - '0')];

**for** (**int** i = 0; i < letters.length(); i++) {

combination(prefix + letters.charAt(i), digits, offset + 1, ret);

}

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 278 | [First Bad Version](https://leetcode.com/problems/first-bad-version/) | 24.9% | Easy |  |

Binary Search

1. Recursive

if (isBadVersion(middle)) {

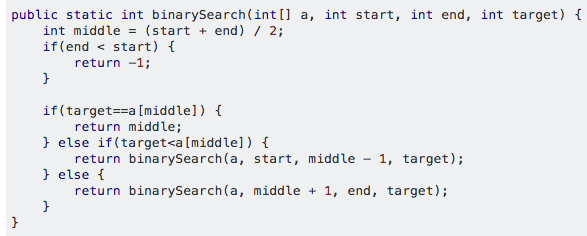
if (!isBadVersion(middle-1)) {

return middle;

}

return binarySearch(start,middle-1);

} else {



return binarySearch(middle + 1, end);

}

1. Iterative

int start = 1, end = n;

while (start <= end) {

int mid = start + (end-start) / 2;

if (!isBadVersion(mid)) start = mid + 1;

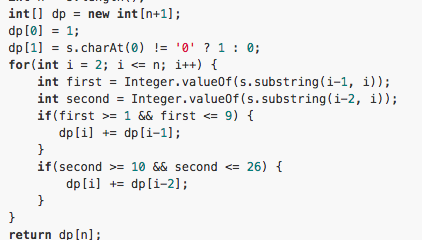
else end = mid-1;

}

return start;

|  |  |  |  |
| --- | --- | --- | --- |
| 91 | [Decode Ways](https://leetcode.com/problems/decode-ways/) | 19.3% | Medium |

DP from start to end:

Base: string of len 0 has 1 decode way.

String of len 1 has 1 way if first char is not 0.

Recursive: current digit i is 1-9

Dp[i]+=dp[i-1]

Number [i-1,i] is valid (10-26)

Dp[i]+=dp[i-2]

|  |  |  |  |
| --- | --- | --- | --- |
| 253 | [Meeting Rooms II](https://leetcode.com/problems/meeting-rooms-ii/) | 38.7% | Medium |

Arrays.sort(intervals, **new** Comparator<Interval>() {

**public** **int** **compare**(Interval a, Interval b) { **return** a.start - b.start; }

});

1. Min heap

Sort the intervals based on starting time of events.  
Keep a min heap based on ending time of events.

**if** (intervals[i].start >= ends.peek()) {

*// no overlap, then should poll out and update smallest end.*

ends.poll();

}

*// Add the merged interval or only the new required interval.*

ends.offer(intervals[i].end);

**return** ends.size();

1. Event queue: Event class with start, end, type.

Sort the start/end events together and process increasingly.

Count the depth (depth ++ when start, depth-- when ends), Record max depth

|  |  |  |  |
| --- | --- | --- | --- |
| 10 | [Regular Expression Matching](https://leetcode.com/problems/regular-expression-matching/) | 24.0% | Hard |

1, If p.charAt(j) == s.charAt(i) : dp[i][j] = dp[i-1][j-1];

2, If p.charAt(j) == '.' : dp[i][j] = dp[i-1][j-1];

3, If p.charAt(j) == '\*':

here are two sub conditions:

1. if p.charAt(j-1) != s.charAt(i) : dp[i][j] = dp[i][j-2]

//in this case, a\* only counts as empty

2 if p.charAt(i-1) == s.charAt(i) or p.charAt(i-1) == '.':

dp[i][j] = dp[i-1][j] //in this case, a\* counts as multiple a

or dp[i][j] = dp[i][j-1] // in this case, a\* counts as single a

or dp[i][j] = dp[i][j-2] // in this case, a\* counts as empty

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 15 | [3Sum](https://leetcode.com/problems/3sum/) | 21.5% | Medium |  |

O(n2): for each possible first triplet, do two sum.

**for** (int i = 0; i < num.length-2; i++) {

**if** (i == 0 || (i > 0 && num[i] != num[i-1])) {

int lo = i+1, hi = num.length-1, sum = 0 - num[i];

**while** (lo < hi) {

**if** (num[lo] + num[hi] == sum) {

res.add(Arrays.asList(num[i], num[lo], num[hi]));

**while** (lo < hi && num[lo] == num[lo+1]) lo++;

**while** (lo < hi && num[hi] == num[hi-1]) hi--;

lo++; hi--;

} **else** **if** (num[lo] + num[hi] < sum) lo++;

**else** hi--;

}

}

}

|  |  |  |  |
| --- | --- | --- | --- |
| 277 | [Find the Celebrity](https://leetcode.com/problems/find-the-celebrity/) | 35.3% | Medium |

The definition of a celebrity is that all the other n - 1 people know him/her but he/she does not know any of them.

1. Iterative compare the person with candidate, switch candidate if knows(candidate,i).
2. Check if the potential candidate is real.

suppose the candidate after the first for loop is person k, it means 0 to k-1 cannot be the celebrity, because they know a previous or current candidate. Also, since k knows no one between k+1 and n-1, k+1 to n-1 can not be the celebrity either. Therefore, k is the only possible celebrity, if there exists one.

**int** candidate = 0;

**for**(**int** i = 1; i < n; i++){

**if**(knows(candidate, i))

candidate = i;

}

**for**(**int** i = 0; i < n; i++){

**if**(i != candidate && (knows(candidate, i) || !knows(i, candidate))) **return** -1;

}

**return** candidate;

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 157 | [Read N Characters Given Read4](https://leetcode.com/problems/read-n-characters-given-read4/) | 29.0% | Easy |  |

The API: int read4(char \*buf) reads 4 characters at a time from a file.

it returns 3 if there is only 3 characters left in the file.

 implement the function int read(char \*buf, int n) that reads *n* characters from the file.

Key is to determine how many bits to copy from read buf to our buffer. Math.min()

**while** (!eof && total < n) {

**int** count = read4(tmp);

*// check if it's the end of the file*

eof = count < 4;

*// get the actual count*

count = Math.min(count, n - total);

*// copy from temp buffer to buf*

**for** (**int** i = 0; i < count; i++)

buf[total++] = tmp[i];

}

**return** total;

|  |  |  |  |
| --- | --- | --- | --- |
| 158 | [Read N Characters Given Read4 II - Call multiple times](https://leetcode.com/problems/read-n-characters-given-read4-ii-call-multiple-times/) | 24.2% | Hard |

Keep global variables to store previous read records.

**private** **int** buffPtr = 0; [*//stores*](file://stores) *the previous position copied to buffer*

**private** **int** buffCnt = 0; [*//stores*](file://stores) *the previous position read by API*

**private** **char**[] buff = **new** **char**[4];[*//stores*](file://stores) *the last buff read*

**public** **int** **read**(**char**[] buf, **int** n) {

**int** ptr = 0;

**while** (ptr < n) {

**if** (buffPtr == 0) {

buffCnt = read4(buff);

}

**while** (ptr < n && buffPtr < buffCnt) {

buf[ptr++] = buff[buffPtr++];

}

*// all chars in buff used up, set pointer to 0*

**if** (buffPtr == buffCnt) buffPtr = 0;

*// read4 returns less than 4, end of file*

**if** (buffCnt < 4) **break**;

}

**return** ptr;

}

|  |  |  |  |
| --- | --- | --- | --- |
| 297 | [Serialize and Deserialize Binary Tree](https://leetcode.com/problems/serialize-and-deserialize-binary-tree/) | 32.8% | Hard |

Recursively build the string from tree or reversely.

**public** **class** **Codec** {

**private** **static** **final** String spliter = ",";

**private** **static** **final** String NN = "X";

*// Encodes a tree to a single string.*

**public** String **serialize**(TreeNode root) {

StringBuilder sb = **new** StringBuilder();

buildString(root, sb);

**return** sb.toString();

}

**private** **void** **buildString**(TreeNode node, StringBuilder sb) {

**if** (node == **null**) {

sb.append(NN).append(spliter);

} **else** {

sb.append(node.val).append(spliter);

buildString(node.left, sb);

buildString(node.right,sb);

}

}

*// Decodes your encoded data to tree.*

**public** TreeNode **deserialize**(String data) {

Deque<String> nodes = **new** LinkedList<>();

nodes.addAll(Arrays.asList(data.split(spliter)));

**return** buildTree(nodes);

}

**private** TreeNode **buildTree**(Deque<String> nodes) {

String val = nodes.remove();

**if** (val.equals(NN)) **return** **null**;

**else** {

TreeNode node = **new** TreeNode(Integer.valueOf(val));

node.left = buildTree(nodes);

node.right = buildTree(nodes);

**return** node;

}

}

}

|  |  |  |  |
| --- | --- | --- | --- |
| 200 | [Number of Islands](https://leetcode.com/problems/number-of-islands/) | 33.8% | Medium |

***Example 1:***  '1's (land) and '0's (water),

11110  
11010  
11000  
00000

Answer: 1

***Example 2:***

11000  
11000  
00100  
00011

Answer: 3

For each island, DFS from it towards 4 directions and mark them as 0. Sink the islands next to each other and increment only once.

**int** y; *// The height of the given grid*

**int** x; *// The width of the given grid*

**char**[][] g; *// The given grid, stored to reduce recursion memory usage*

*/\*\**

*\* Given a 2d grid map of '1's (land) and '0's (water),*

*\* count the number of islands.*

*\**

*\* This method approaches the problem as one of depth-first connected*

*\* components search*

*\* @param grid, the given grid.*

*\* @return the number of islands.*

*\*/*

**public** **int** **numIslands**(**char**[][] grid) {

*// Store the given grid*

*// This prevents having to make copies during recursion*

g = grid;

*// Our count to return*

**int** c = 0;

*// Dimensions of the given graph*

y = g.length;

**if** (y == 0) **return** 0;

x = g[0].length;

*// Iterate over the entire given grid*

**for** (**int** i = 0; i < y; i++) {

**for** (**int** j = 0; j < x; j++) {

**if** (g[i][j] == '1') {

dfs(i, j);

c++;

}

}

}

**return** c;

}

*/\*\**

*\* Marks the given site as visited, then checks adjacent sites.*

*\**

*\* Or, Marks the given site as water, if land, then checks adjacent sites.*

*\**

*\* Or, Given one coordinate (i,j) of an island, obliterates the island*

*\* from the given grid, so that it is not counted again.*

*\**

*\* @param i, the row index of the given grid*

*\* @param j, the column index of the given grid*

*\*/*

**private** **void** **dfs**(**int** i, **int** j) {

*// Check for invalid indices and for sites that aren't land*

**if** (i < 0 || i >= y || j < 0 || j >= x || g[i][j] != '1') **return**;

*// Mark the site as visited*

g[i][j] = '0';

*// Check all adjacent sites*

dfs(i + 1, j);

dfs(i - 1, j);

dfs(i, j + 1);

dfs(i, j - 1);

}

|  |  |  |  |
| --- | --- | --- | --- |
| 282 | [Expression Add Operators](https://leetcode.com/problems/expression-add-operators/) | 29.3% | Hard |

Examples:

"123", 6 -> ["1+2+3", "1\*2\*3"]

"232", 8 -> ["2\*3+2", "2+3\*2"]

"105", 5 -> ["1\*0+5","10-5"]

"00", 0 -> ["0+0", "0-0", "0\*0"]

"3456237490", 9191 -> []

Backtracking with state params: current incrementing path, length, left (cur eval result), cur (cur expression), pos (position of digits array).

**void** **dfs**(List<String> ret, **char**[] path, **int** len, **long** left, **long** cur, **char**[] digits, **int** pos, **int** target) {

**if** (pos == digits.length) {

**if** (left + cur == target) ret.add(**new** String(path, 0, len));

**return**;

}

**long** n = 0;

**int** j = len + 1;

**for** (**int** i = pos; i < digits.length; i++) {

n = n \* 10 + digits[i] - '0';

path[j++] = digits[i];

path[len] = '+';

dfs(ret, path, j, left + cur, n, digits, i + 1, target);

path[len] = '-';

dfs(ret, path, j, left + cur, -n, digits, i + 1, target);

path[len] = '\*';

dfs(ret, path, j, left, cur \* n, digits, i + 1, target);

**if** (digits[pos] == '0') **break**;

}

}

**public** List<String> **addOperators**(String num, **int** target) {

List<String> ret = **new** LinkedList<>();

**if** (num.length() == 0) **return** ret;

**char**[] path = **new** **char**[num.length() \* 2 - 1];

**char**[] digits = num.toCharArray();

**long** n = 0;

**for** (**int** i = 0; i < digits.length; i++) {

n = n \* 10 + digits[i] - '0';

path[i] = digits[i];

dfs(ret, path, i + 1, 0, n, digits, i + 1, target);

**if** (n == 0) **break**;

}

**return** ret;

}

|  |  |  |  |
| --- | --- | --- | --- |
| 76 | [Minimum Window Substring](https://leetcode.com/problems/minimum-window-substring/) | 24.7% | Hard |

For example,  
**S** = "ADOBECODEBANC"  
**T** = "ABC"

Minimum window is "BANC".

**if**(S.empty() || T.empty()){

**return** result;

}

unordered\_map<**char**, **int**> map;

unordered\_map<**char**, **int**> window;

**for**(**int** i = 0; i < T.length(); i++){

map[T[i]]++;

}

**int** minLength = INT\_MAX;

**int** letterCounter = 0;

**for**(**int** slow = 0, fast = 0; fast < S.length(); fast++){

**char** c = S[fast];

**if**(map.find(c) != map.end()){

window[c]++;

**if**(window[c] <= map[c]){

letterCounter++;

}

}

**if**(letterCounter >= T.length()){

**while**(map.find(S[slow]) == map.end() || window[S[slow]] > map[S[slow]]){

window[S[slow]]--;

slow++;

}

**if**(fast - slow + 1 < minLength){

minLength = fast - slow + 1;

result = S.substr(slow, minLength);

}

*// shrink the window here*

window[S[slow]]--;

slow++;

letterCounter--;

}

}

**return** result;

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 257 | [Binary Tree Paths](https://leetcode.com/problems/binary-tree-paths/) | 37.3% | Easy |  |

public List<String> binaryTreePaths(TreeNode root) {

List<String> answer = **new** ArrayList<String>();

**if** (root != null) searchBT(root, "", answer);

**return** answer;

}

private **void** searchBT(TreeNode root, String path, List<String> answer) {

**if** (root.left == null && root.right == null) answer.add(path + root.val);

**if** (root.left != null) searchBT(root.left, path + root.val + "->", answer);

**if** (root.right != null) searchBT(root.right, path + root.val + "->", answer);

}

|  |  |  |  |
| --- | --- | --- | --- |
| 23 | [Merge k Sorted Lists](https://leetcode.com/problems/merge-k-sorted-lists/) | 26.8% | Hard |

1. Priority queue

**public** **class** **Solution** {

**public** ListNode **mergeKLists**(List<ListNode> lists) {

**if** (lists==**null**||lists.size()==0) **return** **null**;

PriorityQueue<ListNode> queue= **new** PriorityQueue<ListNode>(lists.size(),**new** Comparator<ListNode>(){

**@Override**

**public** **int** **compare**(ListNode o1,ListNode o2){

**if** (o1.val<o2.val)

**return** -1;

**else** **if** (o1.val==o2.val)

**return** 0;

**else**

**return** 1;

}

});

ListNode dummy = **new** ListNode(0);

ListNode tail=dummy;

**for** (ListNode node:lists)

**if** (node!=**null**)

queue.add(node);

**while** (!queue.isEmpty()){

tail.next=queue.poll();

tail=tail.next;

**if** (tail.next!=**null**)

queue.add(tail.next);

}

**return** dummy.next;

}

}

2.Binary Search

**public** **class** **Solution** {

**public** ListNode **mergeTwoLists**(ListNode l1, ListNode l2) {

**if** (l1 == null) **return** l2;

**if** (l2 == null) **return** l1;

ListNode head=null;

ListNode former=null;

**while** (l1!=null&&l2!=null) {

**if** (l1.val>l2.val) {

**if** (former==null) former=l2; **else** former.next=l2;

**if** (head==null) head=former; **else** former=former.next;

l2=l2.next;

} **else** {

**if** (former==null) former=l1; **else** former.next=l1;

**if** (head==null) head=former; **else** former=former.next;

l1=l1.next;

}

}

**if** (l2!=null) l1=l2;

former.next=l1;

**return** head;

}

**public** ListNode **mergeKLists**(List<ListNode> lists) {

**if** (lists.size()==0) **return** null;

**if** (lists.size()==1) **return** lists.**get**(0);

**if** (lists.size()==2) **return** mergeTwoLists(lists.**get**(0), lists.**get**(1));

**return** mergeTwoLists(mergeKLists(lists.subList(0, lists.size()/2)),

mergeKLists(lists.subList(lists.size()/2, lists.size())));

}

}

101. Symmetric Tree

1

/ \

2 2

/ \ / \

3 4 4 3

**public** **boolean** **isSymmetric**(TreeNode root) {

**if**(root==**null**) **return** **true**;

**return** isMirror(root.left,root.right);

}

**public** **boolean** **isMirror**(TreeNode p, TreeNode q) {

**if**(p==**null** && q==**null**) **return** **true**;

**if**(p==**null** || q==**null**) **return** **false**;

**return** (p.val==q.val) && isMirror(p.left,q.right) && isMirror(p.right,q.left);

}

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 543 | [Diameter of Binary Tree](https://leetcode.com/problems/diameter-of-binary-tree/) | 42.8% | Easy |  |

**public** **class** **Solution** {

**int** max;

**public** **int** **diameterOfBinaryTree**(TreeNode root) {

max = 0;

height(root);

**return** max;

}

**int** **height**(TreeNode root){

**if**(root==**null**)**return** -1;

**int** leftH = height(root.left);

**int** rightH = height(root.right);

**int** height = Math.max(leftH,rightH)+1;

max = Math.max(max,leftH+rightH+2);

**return** height;

}

}

|  |  |  |  |
| --- | --- | --- | --- |
| 572 | [Subtree of Another Tree](https://leetcode.com/problems/subtree-of-another-tree/) | 42.1% | Easy |

**public** **class** **Solution** {

**public** **boolean** **isSubtree**(TreeNode s, TreeNode t) {

**if** (s == **null**) **return** **false**;

**if** (isSame(s, t)) **return** **true**;

**return** isSubtree(s.left, t) || isSubtree(s.right, t);

}

**private** **boolean** **isSame**(TreeNode s, TreeNode t) {

**if** (s == **null** && t == **null**) **return** **true**;

**if** (s == **null** || t == **null**) **return** **false**;

**if** (s.val != t.val) **return** **false**;

**return** isSame(s.left, t.left) && isSame(s.right, t.right);

}

}

581. Shortest Unsorted Continuous Subarray

**public** **int** **findUnsortedSubarray**(**int**[] A) {

**int** n = A.length, beg = -1, end = -2, min = A[n-1], max = A[0];

**for** (**int** i=1;i<n;i++) {

max = Math.max(max, A[i]);

min = Math.min(min, A[n-1-i]);

**if** (A[i] < max) end = i;

**if** (A[n-1-i] > min) beg = n-1-i;

}

**return** end - beg + 1;

}

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 26 | [Remove Duplicates from Sorted Array](https://leetcode.com/problems/remove-duplicates-from-sorted-array/) | 35.5% | Easy |  |

**class** **Solution** {

**public**:

**int** **removeDuplicates**(**int** A[], **int** n) {

**if**(n < 2) **return** n;

**int** id = 1;

**for**(**int** i = 1; i < n; ++i)

**if**(A[i] != A[i-1]) A[id++] = A[i];

**return** id;

}

};

|  |  |  |  |
| --- | --- | --- | --- |
| 168 | [Excel Sheet Column Title](https://leetcode.com/problems/excel-sheet-column-title/) | 25.5% | Easy |

public class Solution {

public String convertToTitle(int n) {

String res = "";

while(n>0){

n--;

res = (char)(n%26+'A')+res;

n = n/26;

}

return res;

}

}

|  |  |  |  |
| --- | --- | --- | --- |
| 161 | [One Edit Distance](https://leetcode.com/problems/one-edit-distance/) | 31.0% | Medium |

1. Find the first char that is different in s and t.
2. Delete (from s or t)/Replace (in s or t) and determine the equality of the rest of the string.

**public** **boolean** **isOneEditDistance**(String s, String t) {

**for** (**int** i = 0; i < Math.min(s.length(), t.length()); i++) {

**if** (s.charAt(i) != t.charAt(i)) {

**if** (s.length() == t.length()) *// s has the same length as t, so the only possibility is replacing one char in s and t*

**return** s.substring(i + 1).equals(t.substring(i + 1));

**else** **if** (s.length() < t.length()) *// t is longer than s, so the only possibility is deleting one char from t*

**return** s.substring(i).equals(t.substring(i + 1));

**else** *// s is longer than t, so the only possibility is deleting one char from s*

**return** t.substring(i).equals(s.substring(i + 1));

}

}

*//All previous chars are the same, the only possibility is deleting the end char in the longer one of s and t*

**return** Math.abs(s.length() - t.length()) == 1;

}

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 285 | [Inorder Successor in BST](https://leetcode.com/problems/inorder-successor-in-bst/) | 36.0% | Medium |  |

1. Iterative

public TreeNode inorderSuccessor(TreeNode root, TreeNode p) {

TreeNode res = null;

while(root!=null) {

if(root.val > p.val) {

res = root;

root = root.left;

}

else root = root.right;

}

return res;

}

**public** TreeNode **inorderPredecessor** (TreeNode root, TreeNode p) {

TreeNode pre = **null**;

**while**(root!=**null**) {

**if**(root.val < p.val) {

pre = root;

root = root.right;

}

**else** root = root.left;

}

**return** pre;

}

1. Recursive

Successor

**public TreeNode** successor**(TreeNode root, TreeNode p) {**

**if (root == null)**

**return null;**

**if (root.val <= p.val) {**

**return successor(root.right, p);**

**} else {**

**TreeNode left = successor(root.left, p);**

**return (left != null) ? left : root;**

**}**

**}**

Predecessor

**public TreeNode** predecessor**(TreeNode root, TreeNode p) {**

**if (root == null)**

**return null;**

**if (root.val >= p.val) {**

**return predecessor(root.left, p);**

**} else {**

**TreeNode right = predecessor(root.right, p);**

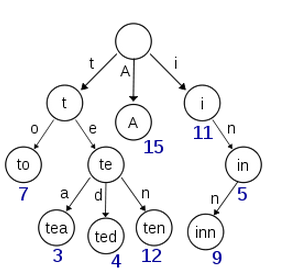
**return (right != null) ? right : root;**

**}**

**}**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 211 | [Add and Search Word - Data structure design](https://leetcode.com/problems/add-and-search-word-data-structure-design/) | 21.9% | Medium |  |

208. Implement Trie (Prefix Tree)



Use an array of length 26 to represent hash map of a-z as children set of a given node.

Have a isWord flag for each node (maybe only prefix of some word).

**class** **TrieNode** {

**public** **char** val;

**public** **boolean** isWord;

**public** TrieNode[] children = **new** TrieNode[26];

**public** **TrieNode**() {}

TrieNode(**char** c){

TrieNode node = **new** TrieNode();

node.val = c;

}

}

**public** **class** **Trie** {

**private** TrieNode root;

**public** **Trie**() {

root = **new** TrieNode();

root.val = ' ';

}

**public** **void** **insert**(String word) {

TrieNode ws = root;

**for**(**int** i = 0; i < word.length(); i++){

**char** c = word.charAt(i);

**if**(ws.children[c - 'a'] == **null**){

ws.children[c - 'a'] = **new** TrieNode(c);

}

ws = ws.children[c - 'a'];

}

ws.isWord = **true**;

}

**public** **boolean** **search**(String word) {

TrieNode ws = root;

**for**(**int** i = 0; i < word.length(); i++){

**char** c = word.charAt(i);

**if**(ws.children[c - 'a'] == **null**) **return** **false**;

ws = ws.children[c - 'a'];

}

**return** ws.isWord;

}

**public** **boolean** **startsWith**(String prefix) {

TrieNode ws = root;

**for**(**int** i = 0; i < prefix.length(); i++){

**char** c = prefix.charAt(i);

**if**(ws.children[c - 'a'] == **null**) **return** **false**;

ws = ws.children[c - 'a'];

}

**return** **true**;

}

}

Regular expression with “.”

**public** **boolean** **search**(String word) {

**return** helper(word, 0, root);

}

**private** **boolean** **helper**(String s, **int** index, TrieNode p) {

**if** (index >= s.length()) **return** p.isWord;

**char** c = s.charAt(index);

**if** (c == '.') {

**for** (**int** i = 0; i < p.child.length; i++)

**if** (p.child[i] != **null** && helper(s, index + 1, p.child[i]))

**return** **true**;

**return** **false**;

} **else** **return** (p.child[c - 'a'] != **null** && helper(s, index + 1, p.child[c - 'a']));

}

|  |  |  |  |
| --- | --- | --- | --- |
| 121 | [Best Time to Buy and Sell Stock](https://leetcode.com/problems/best-time-to-buy-and-sell-stock/) | 40.5% | Easy |

1. Keep the min price, max profit sold at day i, max of max profit.

public class Solution {

public int maxProfit(int[] prices) {

if(prices.length ==0)

return 0;

int min = prices[0];

int max\_profit = -1;

for(int price:prices){

min = Math.min(min,price);

max\_profit = Math.max(max\_profit, price-min);

}

return max\_profit;

}

}

1. If the array is difference of prices, keep

public class Solution {

public int maxProfit(int[] prices) {

int maxCur = 0, maxSoFar = 0;

for(int i = 1; i < prices.length; i++) {

maxCur = Math.max(prices[i] - prices[i-1], maxCur+ prices[i] - prices[i-1]);

//include ending at i-1 or not

maxSoFar = Math.max(maxCur, maxSoFar);

}

return maxSoFar;

}

}

492. Construct the Rectangle

**public** **int**[] constructRectangle(**int** area) {

**int** w = (**int**)Math.sqrt(area);

**while** (area%w!=0) w--;

**return** **new** **int**[]{area/w, w};

}

39. Combination Sum

Given a **set** of candidate numbers (***C***) **(without duplicates)** and a target number (***T***), find all unique combinations in ***C*** where the candidate numbers sums to ***T***.

The **same** repeated number may be chosen from ***C*** unlimited number of times.

**public** List<List<Integer>> combinationSum(**int**[] nums, **int** target) {

List<List<Integer>> list = **new** ArrayList<>();

Arrays.sort(nums);

backtrack(list, **new** ArrayList<>(), nums, target, 0);

**return** list;

}

**private** **void** **backtrack**(List<List<Integer>> list, List<Integer> tempList, **int** [] nums, **int** remain, **int** start){

**if**(remain < 0) **return**;

**else** **if**(remain == 0) list.add(**new** ArrayList<>(tempList));

**else**{

**for**(**int** i = start; i < nums.length && nums[i]<=remain; i++){ //prune

tempList.add(nums[i]);

backtrack(list, tempList, nums, remain - nums[i], i); *// not i + 1 because we can reuse same elements*

tempList.remove(tempList.size() - 1);

}

}

}

40. Combination Sum II

Given a collection of candidate numbers (***C***) and a target number (***T***), find all unique combinations in ***C*** where the candidate numbers sums to ***T***.

Each number in ***C*** may only be used **once** in the combination.

**public** List<List<Integer>> combinationSum2(**int**[] nums, **int** target) {

List<List<Integer>> list = **new** ArrayList<>();

Arrays.sort(nums);

backtrack(list, **new** ArrayList<>(), nums, target, 0);

**return** list;

}

**private** **void** **backtrack**(List<List<Integer>> list, List<Integer> tempList, **int** [] nums, **int** remain, **int** start){

**if**(remain < 0) **return**;

**else** **if**(remain == 0) list.add(**new** ArrayList<>(tempList));

**else**{

**for**(**int** i = start; i < nums.length; i++){

**if**(i > start && nums[i] == nums[i-1]) **continue**; *// skip duplicates*

tempList.add(nums[i]);

backtrack(list, tempList, nums, remain - nums[i], i + 1);

tempList.remove(tempList.size() - 1);

}

}

}

78. Subsets

**public** **List**<**List**<Integer>> subsets(int[] nums) {

**List**<**List**<Integer>> **list** = **new** ArrayList<>();

Arrays.sort(nums);

backtrack(**list**, **new** ArrayList<>(), nums, 0);

**return** **list**;

}

**private** void backtrack(**List**<**List**<Integer>> **list** , **List**<Integer> tempList, int [] nums, int start){

**list**.add(**new** ArrayList<>(tempList));

**for**(int i = start; i < nums.length; i++){

tempList.add(nums[i]);

backtrack(**list**, tempList, nums, i + 1);

tempList.remove(tempList.size() - 1);

}

}

Subsets II (contains duplicates) : <https://leetcode.com/problems/subsets-ii/>

**public** **List**<**List**<Integer>> subsetsWithDup(int[] nums) {

**List**<**List**<Integer>> **list** = **new** ArrayList<>();

Arrays.sort(nums);

backtrack(**list**, **new** ArrayList<>(), nums, 0);

**return** **list**;

}

**private** void backtrack(**List**<**List**<Integer>> **list**, **List**<Integer> tempList, int [] nums, int start){

**list**.add(**new** ArrayList<>(tempList));

**for**(int i = start; i < nums.length; i++){

**if**(i > start && nums[i] == nums[i-1]) **continue**; *// skip duplicates*

tempList.add(nums[i]);

backtrack(**list**, tempList, nums, i + 1);

tempList.remove(tempList.size() - 1);

}

}

46. Permutations

Given a collection of **distinct** numbers, return all possible permutations.

**public** **List**<**List**<Integer>> permute(int[] nums) {

**List**<**List**<Integer>> **list** = **new** ArrayList<>();

*// Arrays.sort(nums); // not necessary*

backtrack(**list**, **new** ArrayList<>(), nums);

**return** **list**;

}

**private** void backtrack(**List**<**List**<Integer>> **list**, **List**<Integer> tempList, int [] nums){

**if**(tempList.size() == nums.length){

**list**.add(**new** ArrayList<>(tempList));

} **else**{

**for**(int i = 0; i < nums.length; i++){

**if**(tempList.contains(nums[i])) **continue**; *// element already exists, skip*

tempList.add(nums[i]);

backtrack(**list**, tempList, nums);

tempList.remove(tempList.size() - 1);

}

}

}

47. Permutations II

Given a collection of numbers that might contain duplicates, return all possible unique permutations.

For example,  
[1,1,2] have the following unique permutations:

[

[1,1,2],

[1,2,1],

[2,1,1]

]

**public** **List**<**List**<Integer>> permuteUnique(int[] nums) {

**List**<**List**<Integer>> **list** = **new** ArrayList<>();

Arrays.sort(nums);

backtrack(**list**, **new** ArrayList<>(), nums, **new** boolean[nums.length]);

**return** **list**;

}

**private** void backtrack(**List**<**List**<Integer>> **list**, **List**<Integer> tempList, int [] nums, boolean [] used){

**if**(tempList.size() == nums.length){

**list**.add(**new** ArrayList<>(tempList));

} **else**{

**for**(int i = 0; i < nums.length; i++){

**if**(used[i]) **continue**;

**if**(i>0 &&nums[i-1]==nums[i] && !used[i-1]) **continue**;

used[i] = **true**;

tempList.add(nums[i]);

backtrack(**list**, tempList, nums, used);

used[i] = **false**;

tempList.remove(tempList.size() - 1);

}

}

}

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 236 | [Lowest Common Ancestor of a Binary Tree](https://leetcode.com/problems/lowest-common-ancestor-of-a-binary-tree/) | 29.7% | Medium |  |

**public** TreeNode **lowestCommonAncestor**(TreeNode root, TreeNode p, TreeNode q) {

**if**( root == p || root == q || root == **null**)

**return** root;

TreeNode left = lowestCommonAncestor( root.left, p, q);

TreeNode right = lowestCommonAncestor( root.right, p, q);

**if**(left == **null**)

**return** right;

**else** **if** (right == **null**)

**return** left;

**else**

**return** root;

}

|  |  |  |  |
| --- | --- | --- | --- |
| 49 | [Group Anagrams](https://leetcode.com/problems/group-anagrams/) | 33.7% | Medium |

public **class** **Solution** {

public List<List<String>> groupAnagrams(String[] strs) {

**if** (strs == null || strs.length == 0) **return** **new** ArrayList<List<String>>();

Map<String, List<String>> map = **new** HashMap<String, List<String>>();

**for** (String s : strs) {

char[] ca = s.toCharArray();

Arrays.sort(ca);

String keyStr = String.valueOf(ca);

**if** (!map.containsKey(keyStr)) map.put(keyStr, **new** ArrayList<String>());

map.get(keyStr).add(s);

}

**return** **new** ArrayList<List<String>>(map.values());

}

}

|  |  |  |  |
| --- | --- | --- | --- |
| 79 | [Word Search](https://leetcode.com/problems/word-search/) | 26.3% | Medium |

Given **board** =

[

['A','B','C','E'],

['S','F','C','S'],

['A','D','E','E']

]

**word** = "ABCCED", -> returns true,  
**word** = "SEE", -> returns true,  
**word** = "ABCB", -> returns false.

Could not reuse an index of the board. So set up a marker to xor with char (toggling used or unused) to avoid extra space.

**public** **boolean** **exist**(**char**[][] board, String word) {

**char**[] w = word.toCharArray();

**for** (**int** y=0; y<board.length; y++) {

**for** (**int** x=0; x<board[y].length; x++) {

**if** (exist(board, y, x, w, 0)) **return** **true**;

}

}

**return** **false**;

}

**private** **boolean** **exist**(**char**[][] board, **int** y, **int** x, **char**[] word, **int** i) {

**if** (i == word.length) **return** **true**;

**if** (y<0 || x<0 || y == board.length || x == board[y].length) **return** **false**;

**if** (board[y][x] != word[i]) **return** **false**;

board[y][x] ^= 256;

**boolean** exist = exist(board, y, x+1, word, i+1)

|| exist(board, y, x-1, word, i+1)

|| exist(board, y+1, x, word, i+1)

|| exist(board, y-1, x, word, i+1);

board[y][x] ^= 256;

**return** exist;

}

|  |  |  |  |
| --- | --- | --- | --- |
| 238 | [Product of Array Except Self](https://leetcode.com/problems/product-of-array-except-self/) | 48.6% | Medium |

{ 1, a[0], a[0]\*a[1], a[0]\*a[1]\*a[2], }

{ a[1]\*a[2]\*a[3], a[2]\*a[3], a[3], 1, }

int a[N] // This is the input

int products[N];

// Get the products below the current index

p=1;

for(int i=0;i<N;++i) {

products[i]=p;

p\*=a[i];

}

// Get the products above the curent index

p=1;

for(int i=N-1;i>=0;--i) {

products[i]\*=p;

p\*=a[i];

}

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 38 | [Count and Say](https://leetcode.com/problems/count-and-say/) | 34.0% | Easy |  |

1. 1

2. 11

3. 21

4. 1211

5. 111221

public String countAndSay(int n) {

String s = "1";

while (n-- > 1) { /\* invariant: s is nth \*/

StringBuilder next = new StringBuilder(); /\* invariant: contain cnt-say before cur \*/

for (int i = 0, cnt = 1; i < s.length(); i++, cnt++) {

if (i == s.length()-1 || s.charAt(i+1) != s.charAt(i)) {

next.append(cnt).append(s.charAt(i));

cnt = 0;

}

}

s = next.toString();

}

return s;

}

}

228. Summary Ranges

For example, given [0,1,2,4,5,7], return ["0->2","4->5","7"].

**public** List<String> **summaryRanges**(**int**[] nums) {

List<String> ret = **new** ArrayList<>();

**int** n = nums.length;

**for** (**int** i = 1, j = 0; i <= n; i++) {

**if** (i == n || nums[i - 1] != nums[i] - 1) {

ret.add(j == i - 1 ? nums[j] + "" : nums[j] + "->" + nums[i - 1]);

j = i;

}

}

**return** ret;

}

*// Encode string from "aaabccccc" -> "3ab5c"*

**public** String **encode**(String word) {

StringBuilder enc = **new** StringBuilder();

**int** n = word.length();

**for** (**int** i = 1, cnt = 1; i <= n; i++, cnt++) {

**if** (i == n || word.charAt(i - 1) != word.charAt(i)) {

**if** (cnt > 1) {

enc.append(cnt);

}

enc.append(word.charAt(i - 1));

cnt = 0;

}

}

**return** enc.length() < word.length() ? enc.toString() : word;

}

|  |  |  |  |
| --- | --- | --- | --- |
| 215 | [Kth Largest Element in an Array](https://leetcode.com/problems/kth-largest-element-in-an-array/) | 38.8% | Medium |

public int findKthLargest(int[] nums, int k) {

k = nums.length - k;

int lo = 0;

int hi = nums.length - 1;

while (lo < hi) {

final int j = partition(nums, lo, hi);

if(j < k) {

lo = j + 1;

} else if (j > k) {

hi = j - 1;

} else {

break;

}

}

return nums[k];

}

private int partition(int[] a, int lo, int hi) {

int i = lo;

int j = hi;

while(i<j) {

while(i < hi && (a[i]<= a[lo]))

i++;

while(j > lo && (a[lo] <= a[j]))

j--;

if(i < j) {

swap(a, i, j);

}

}

swap(a, lo, j);

return j;

}

private void swap(int[] a, int i, int j) {

final int tmp = a[i];

a[i] = a[j];

a[j] = tmp;

}

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 209 | [Minimum Size Subarray Sum](https://leetcode.com/problems/minimum-size-subarray-sum/) | 30.1% | Medium |  |

public int minSubArrayLen(int s, int[] nums) {

int p1=0;

int p2 =0;

int sum = 0;

int minLen = Integer.MAX\_VALUE;

while(p2<nums.length){

sum+=nums[p2++];

while(sum>=s){

minLen = Math.min(minLen,p2-p1);

sum-=nums[p1++];

}

}

return (minLen == Integer.MAX\_VALUE) ? 0 : minLen;

}

|  |  |  |  |
| --- | --- | --- | --- |
| 57 | [Insert Interval](https://leetcode.com/problems/insert-interval/) | 27.2% | Hard |

Change newInterval in place (change start/end or set it to null)

public List<Interval> insert(List<Interval> intervals, Interval newInterval) {

int start = newInterval.start;

int end = newInterval.end;

List<Interval> res = new ArrayList<Interval>();

boolean addMerge = false;

if(intervals.size()==0)

res.add(newInterval);

for(int j =0;j<intervals.size();j++){

Interval i = intervals.get(j);

if(newInterval==null||i.end<newInterval.start){

res.add(i);

}

else if(i.start>newInterval.end){

res.add(newInterval);

res.add(i);

newInterval=null;

}

else{

newInterval.end = Math.max(i.end,newInterval.end);

newInterval.start = Math.min(i.start,newInterval.start);

}

}

if (newInterval != null)

res.add(newInterval);

return res;

}

**public** List<Interval> **insert**(List<Interval> intervals, Interval newInterval) {``

**int** i=0;

**while**(i<intervals.size() && intervals.**get**(i).end<newInterval.start) i++;

**while**(i<intervals.size() && intervals.**get**(i).start<=newInterval.end){

newInterval = **new** Interval(Math.min(intervals.**get**(i).start, newInterval.start), Math.max(intervals.**get**(i).end, newInterval.end));

intervals.remove(i);

}

intervals.add(i,newInterval);

**return** intervals;

}

|  |  |  |  |
| --- | --- | --- | --- |
| 71 | [Simplify Path](https://leetcode.com/problems/simplify-path/) | 24.9% | Medium |

public String simplifyPath(String path) {

Deque<String> stack = new ArrayDeque<>();

Set<String> skip = new HashSet<>(Arrays.asList("..",".",""));

for (String dir : path.split("/")) {

if (dir.equals("..") && !stack.isEmpty()) stack.pop();

else if (!skip.contains(dir)) stack.push(dir);

}

String res = "";

while(!stack.isEmpty()) res ="/"+stack.pop() + res ;

return res.isEmpty() ? "/" : res;

}

Stack is deprecated. Use deque (ArrayDeque) as double-ended queue/stack.

dequeA.add ("element 1"); //add element at tail

dequeA.addFirst("element 2"); //add element at head

dequeA.addLast ("element 3"); //add element at tail

Object firstElement = dequeA.remove();

Object firstElement = dequeA.removeFirst();

Object lastElement = dequeA.removeLast();  
Object firstElement = dequeA.element();

Object firstElement = dequeA.getFirst();

Object lastElement = dequeA.getLast();

|  |  |  |  |
| --- | --- | --- | --- |
| 146 | [LRU Cache](https://leetcode.com/problems/lru-cache/) | 17.3% | Hard |

get(key) - Get the value (will always be positive) of the key if the key exists in the cache, otherwise return -1.  
put(key, value) - Set or insert the value if the key is not already present. When the cache reached its capacity, it should invalidate the least recently used item before inserting a new item.

Keep a hash table that keeps track of the keys and its values in the double linked list.

It takes constant time to add and remove nodes from the head or tail.

**class** **DLinkedNode** {

**int** key;

**int** **value**;

DLinkedNode pre;

DLinkedNode post;

}

*/\*\**

*\* Always add the new node right after head;*

*\*/*

**private** **void** **addNode**(DLinkedNode node){

node.pre = head;

node.post = head.post;

head.post.pre = node;

head.post = node;

}

*/\*\**

*\* Remove an existing node from the linked list.*

*\*/*

**private** **void** **removeNode**(DLinkedNode node){

DLinkedNode pre = node.pre;

DLinkedNode post = node.post;

pre.post = post;

post.pre = pre;

}

*/\*\**

*\* Move certain node in between to the head.*

*\*/*

**private** **void** **moveToHead**(DLinkedNode node){

**this**.removeNode(node);

**this**.addNode(node);

}

*// pop the current tail.*

**private** DLinkedNode **popTail**(){

DLinkedNode res = tail.pre;

**this**.removeNode(res);

**return** res;

}

**private** Hashtable<Integer, DLinkedNode>

cache = **new** Hashtable<Integer, DLinkedNode>();

**private** **int** count;

**private** **int** capacity;

**private** DLinkedNode head, tail;

**public** **LRUCache**(**int** capacity) {

**this**.count = 0;

**this**.capacity = capacity;

head = **new** DLinkedNode();

head.pre = null;

tail = **new** DLinkedNode();

tail.post = null;

head.post = tail;

tail.pre = head;

}

**public** **int** **get**(**int** key) {

DLinkedNode node = cache.**get**(key);

**if**(node == null){

**return** -1; *// should raise exception here.*

}

*// move the accessed node to the head;*

**this**.moveToHead(node);

**return** node.**value**;

}

**public** **void** **set**(**int** key, **int** **value**) {

DLinkedNode node = cache.**get**(key);

**if**(node == null){

DLinkedNode newNode = **new** DLinkedNode();

newNode.key = key;

newNode.**value** = **value**;

**this**.cache.put(key, newNode);

**this**.addNode(newNode);

++count;

**if**(count > capacity){

*// pop the tail*

DLinkedNode tail = **this**.popTail();

**this**.cache.remove(tail.key);

--count;

}

}**else**{

*// update the value.*

node.**value** = **value**;

**this**.moveToHead(node);

}

}

|  |  |  |  |
| --- | --- | --- | --- |
| 13 | [Roman to Integer](https://leetcode.com/problems/roman-to-integer/) | 45.2% | Easy |

Input : "XIV"

Return : 14

Input : "XX"

Output : 20

If current num[i] is smaller than next num[i+1], num[i] should be subtracted. Otherwise, add num[i]. Finally add the last bit anyway.

IV = 5 – 1, VI = 5 + 1

**public** **int** **romanToInt**(String s) {

**int** nums[]=**new** **int**[s.length()];

**for**(**int** i=0;i<s.length();i++){

**switch** (s.charAt(i)){

**case** 'M':

nums[i]=1000;

**break**;

**case** 'D':

nums[i]=500;

**break**;

**case** 'C':

nums[i]=100;

**break**;

**case** 'L':

nums[i]=50;

**break**;

**case** 'X' :

nums[i]=10;

**break**;

**case** 'V':

nums[i]=5;

**break**;

**case** 'I':

nums[i]=1;

**break**;

}

}

**int** sum=0;

**for**(**int** i=0;i<nums.length-1;i++){

**if**(nums[i]<nums[i+1])

sum-=nums[i];

**else**

sum+=nums[i];

}

**return** sum+nums[nums.length-1];

}

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 33 | [Search in Rotated Sorted Array](https://leetcode.com/problems/search-in-rotated-sorted-array/) | 32.1% | Medium |  |

The idea is that when rotating the array, there must be one half of the array that is still in sorted order.  
For example, 6 7 1 2 3 4 5, the order is disrupted from the point between 7 and 1. So when doing binary search, we can make a judgement that which part is ordered and whether the target is in that range, if yes, continue the search in that half, if not continue in the other half.

**public** **class** **Solution** {

**public** **int** **search**(**int**[] nums, **int** target) {

**int** start = 0;

**int** end = nums.length - 1;

**while** (start <= end){

**int** mid = (start + end) / 2;

**if** (nums[mid] == target)

**return** mid;

**if** (nums[start] <= nums[mid]){

**if** (target < nums[mid] && target >= nums[start])

end = mid - 1;

**else**

start = mid + 1;

}

**if** (nums[mid] <= nums[end]){

**if** (target > nums[mid] && target <= nums[end])

start = mid + 1;

**else**

end = mid - 1;

}

}

**return** -1;

}

}

|  |  |  |  |
| --- | --- | --- | --- |
| 44 | [Wildcard Matching](https://leetcode.com/problems/wildcard-matching/) | 19.8% | Hard |

dp[i][j] denotes whether s[0....i-1] matches p[0.....j-1],

First, we need to initialize dp[i][0], i= [1,m]. All the dp[i][0] should be false because p has nothing in it.

Then, initialize dp[0][j], j = [1, n]. In this case, s has nothing, to get dp[0][j] = true, p must be '\*', '\*', '\*\*',etc. Once p.charAt(j-1) != '\*', all the dp[0][j] afterwards will be false.

**public** **boolean** **isMatch\_2d\_method**(String s, String p) {

**int** m=s.length(), n=p.length();

**boolean**[][] dp = **new** **boolean**[m+1][n+1];

dp[0][0] = **true**;

**for** (**int** i=1; i<=m; i++) {

dp[i][0] = **false**;

}

**for**(**int** j=1; j<=n; j++) {

**if**(p.charAt(j-1)=='\*'){

dp[0][j] = **true**;

} **else** {

**break**;

}

}

**for**(**int** i=1; i<=m; i++) {

**for**(**int** j=1; j<=n; j++) {

**if** (p.charAt(j-1)!='\*') {

dp[i][j] = dp[i-1][j-1] && (s.charAt(i-1)==p.charAt(j-1) || p.charAt(j-1)=='?');

} **else** {

dp[i][j] = dp[i-1][j] || dp[i][j-1];

}

}

}

**return** dp[m][n];

}

For each element in s  
If \*s==\*p or \*p == ? which means this is a match, then goes to next element s++ p++.  
If p=='\*', this is also a match, but one or many chars may be available, so let us save this \*'s position and the matched s position.  
If not match, then we check if there is a \* previously showed up,  
       if there is no \*,  return false;  
       if there is an \*,  we set current p to the next element of \*, and set current s to the next saved s position.  
  
e.g.  
  
abed  
?b\*d\*\*  
  
a=?, go on, b=b, go on,  
e=\*, save \* position star=3, save s position ss = 3, p++  
e!=d,  check if there was a \*, yes, ss++, s=ss; p=star+1  
d=d, go on, meet the end.  
check the rest element in p, if all are \*, true, else false;

**boolean** **comparison**(String str, String pattern) {

**int** s = 0, p = 0, match = 0, starIdx = -1;

**while** (s < str.length()){

*// advancing both pointers*

*// match: s\_last\_match\_begin*

*// starIdx: p\_last\_asterisk\_pos*

**if** (p < pattern.length() && (pattern.charAt(p) == '?' || str.charAt(s) == pattern.charAt(p))){

s++;

p++;

}

*// \* found, only advancing pattern pointer*

**else** **if** (p < pattern.length() && pattern.charAt(p) == '\*'){

starIdx = p;

match = s;

p++;

}

*// last pattern pointer was \*, advancing string pointer*

**else** **if** (starIdx != -1){

p = starIdx + 1;

match++;

s = match;

}

*//current pattern pointer is not star, last patter pointer was not \**

*//characters do not match*

**else** **return** **false**;

}

*//check for remaining characters in pattern*

**while** (p < pattern.length() && pattern.charAt(p) == '\*')

p++;

**return** p == pattern.length();

}

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 380 | [Insert Delete GetRandom O(1)](https://leetcode.com/problems/insert-delete-getrandom-o1/) | 39.0% | Medium |  |

1. insert(val): Inserts an item val to the set if not already present.
2. remove(val): Removes an item val from the set if present.
3. getRandom: Returns a random element from current set of elements. Each element must have the **same probability** of being returned.

HashMap (int, set of int) to store the values and its locations (if dups). ArrayList stores values at specific locations.

Remove will remove the element and swap the last element to the empty location and update locations.

ArrayList<Integer> nums;

HashMap<Integer, Set<Integer>> locs;

java.util.Random rand = **new** java.util.Random();

*/\*\* Initialize your data structure here. \*/*

**public** **RandomizedSet**() {

nums = **new** ArrayList<Integer>();

locs = **new** HashMap<Integer, Set<Integer>>();

}

*/\*\* Inserts a value to the set. Returns true if the set did not already contain the specified element. \*/*

**public** boolean **insert**(**int** val) {

boolean contain = locs.containsKey(val);

**if** ( ! contain ) locs.put( val, **new** HashSet<Integer>() );

locs.**get**(val).add(nums.size());

nums.add(val);

**return** ! contain ;

*/\*\* Removes a value from the set. Returns true if the set contained the specified element. \*/*

**public** boolean **remove**(**int** val) {

boolean contain = locs.containsKey(val);

**if** ( ! contain ) **return** false;

**int** loc = locs.**get**(val).iterator().next();

locs.**get**(val).remove(loc);

**if** (loc < nums.size() - 1 ) {

**int** lastone = nums.**get**(nums.size() - 1 );

nums.**set**( loc , lastone );

locs.**get**(lastone).remove(nums.size() - 1);

locs.**get**(lastone).add(loc);

}

nums.remove(5.size() - 1);

**if** (locs.**get**(val).isEmpty()) locs.remove(val);

**return** true;

}

*/\*\* Get a random element from the set. \*/*

**public** **int** **getRandom**() {

**return** nums.**get**( rand.nextInt(nums.size()) );

}

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 128 | [Longest Consecutive Sequence](https://leetcode.com/problems/longest-consecutive-sequence/) | 36.5% | Hard |  |

476. Number Complement

unsigned mask = ~0;

**while** (num & mask) mask <<= 1;

**return** ~mask & ~num;

258. Add Digits

**public** **int** **addDigits**(**int** num) {

**return** num==0?0:(num%9==0?9:(num%9));

}