Welcome to LeetCode Discuss. Please read the **FAQ** (/discuss/faq) to help yourself making the best use of Discuss. x

Search

Ask a Question (../ask?cat=224)

All Activity (../activity) Questions (../questions) Unanswered (../unanswered) Tags (../tags) Categories (../categories)

Users (../users) My Updates (../updates)

Back to Problem (/problems/maximal-squ

Easy DP solution in C++ with detailed explanations (8ms, O(n^2) time and O(n) space)

3,280 views

+106 votes

Well, this problem desires for the use of dynamic programming. They key to any DP problem is to come up with the state equation. In this problem, we define the state to be **the maximal size of the square that can be achieved at point** `(i, j)`, denoted as `P[i][j]`. Remember that we use **size** instead of square as the state (`square = size^2`).

Now let's try to come up with the formula for `P[i][j]`.

First, it is obvious that for the topmost row (`i = 0`) and the leftmost column (`j = 0`), `P[i][j] = matrix[i][j]`. This is easily understood. Let's suppose that the topmost row of `matrix` is like `[1, 0, 0, 1]`. Then we can immediately know that the first and last point can be a square of size `1` while the two middle points cannot make any square, giving a size of `0`. Thus, `P = [1, 0, 0, 1]`, which is the same as `matrix`. The case is similar for the leftmost column. Till now, the boundary conditions of this DP problem are solved.

Let's move to the more general case for `P[i][j]` in which `i > 0` and `j > 0`. First of all, let's see another simple case in which `matrix[i][j] = 0`. It is obvious that `P[i][j] = 0` too. Why? Well, since `matrix[i][j] = 0`, no square will contain `matrix[i][j]`. According to our definition of `P[i][j]`, `P[i][j]` is also `0`.

Now we are almost done. The only unsolved case is `matrix[i][j] = 1`. Let's see an example.

Suppose `matrix = [[0, 1], [1, 1]]`, it is obvious that `P[0][0] = 0, P[0][1] = P[1][0] = 1`, what about `P[1][1]`? Well, to give a square of size larger than `1` in `P[1][1]`, all of its three neighbors (left, up, left-up) should be non-zero, right? In this case, the left-up neighbor `P[0][0] = 0`, so `P[1][1]` can only be 1, which means that it contains the square of itself.

Now you are near the solution. In fact, `P[i][j] = min(P[i - 1][j], P[i][j - 1], P[i - 1][j - 1]) + 1` in this case.

Taking all these together, we have the following state equations.

1. `P[0][j] = matrix[0][j]` (topmost row);
2. `P[i][0] = matrix[i][0]` (leftmost column);
3. For `i > 0` and `j > 0`: if `matrix[i][j] = 0`, `P[i][j] = 0`; if `matrix[i][j] = 1`, `P[i][j] = min(P[i - 1][j], P[i][j - 1], P[i - 1][j - 1]) + 1`.

Putting them into codes, and maintain a variable `maxsize` to record the maximum size of the square we have seen, we have the following (unoptimized) solution.

```
int maximalSquare(vector<vector<char>>& matrix) {
    int m = matrix.size();
    if (!m) return 0;
    int n = matrix[0].size();
    vector<vector<int> > size(m, vector<int>(n, 0));
    int maxsize = 0;
    for (int j = 0; j < n; j++) {
        size[0][j] = matrix[0][j] - '0';
        maxsize = max(maxsize, size[0][j]);
    }
    for (int i = 1; i < m; i++) {
        size[i][0] = matrix[i][0] - '0';
        maxsize = max(maxsize, size[i][0]);
    }
    for (int i = 1; i < m; i++) {
        for (int j = 1; j < n; j++) {
            if (matrix[i][j] == '1') {
                size[i][j] = min(size[i - 1][j - 1], min(size[i - 1][j], size[i][j - 1
                maxsize = max(maxsize, size[i][j]);
            }
        }
    }
    return maxsize * maxsize;
}
```

Welcome to LeetCode Discuss.

This is a place to post interview questions (/discuss/ask?cat=246) or ask questions related to OJ problems (/problems).

Please **read the FAQ** (/discuss/faq) to help yourself making the best use of Discuss.

All categories (../)

OJ (../oj) (25,021)
Two Sum (../oj/two-sum) (327)
Add Two Numbers (../oj/add-two-numbers) (186)
Longest Substring Without Repeating Characters (../oj/longest-substring-without-repeating-characters) (205)
Median of Two Sorted Arrays (../oj/median-of-two-sorted-arrays) (139)
Longest Palindromic Substring (../oj/longest-palindromic-substring) (129)
ZigZag Conversion (../oj/zigzag-conversion) (254)
Reverse Integer (../oj/reverse-integer)
String to Integer (atoi) (../oj/string-to-integer-atoi) (179) (176)
Palindrome Number (../oj/palindrome-number) (177)
Regular Expression Matching (../oj/regular-expression-matching) (139)
Container With Most Water (../oj/container-with-most-water) (45)

Now let's try to optimize the above solution. As can be seen, each time when we update `size[i][j]`, we only need `size[i][j - 1]`, `size[i - 1][j - 1]` (at the previous left column) and `size[i - 1][j]` (at the current column). So we do not need to maintain the full `m*n` matrix. In fact, keeping two columns is enough. Now we have the following optimized solution.

```
int maximalSquare(vector<vector<char>>& matrix) {
    int m = matrix.size();
    if (!m) return 0;
    int n = matrix[0].size();
    vector<int> pre(m, 0);
    vector<int> cur(m, 0);
    int maxsize = 0;
    for (int i = 0; i < m; i++) {
        pre[i] = matrix[i][0] - '0';
        maxsize = max(maxsize, pre[i]);
    }
    for (int j = 1; j < n; j++) {
        cur[0] = matrix[0][j] - '0';
        maxsize = max(maxsize, cur[0]);
        for (int i = 1; i < m; i++) {
            if (matrix[i][j] == '1') {
                cur[i] = min(cur[i - 1], min(pre[i - 1], pre[i])) + 1;
                maxsize = max(maxsize, cur[i]);
            }
        }
        swap(pre, cur);
        fill(cur.begin(), cur.end(), 0);
    }
    return maxsize * maxsize;
}
```

Now you see the solution is finished? In fact, it can still be optimized! In fact, we need not maintain two vectors and one is enough. If you want to explore this idea, please refer to the answers provided by @stellari below. Moreover, in the code above, we distinguish between the `0`-th row and other rows since the `0`-th row has no row above it. In fact, we can make all the `m` rows the same by padding a `0` row on the top (in the following code, we pad a `0` on top of `dp`). Finally, we will have the following short code :) If you find it hard to understand, try to run it using your pen and paper and notice how it realizes what the two-vector solution does using only one vector.

```
int maximalSquare(vector<vector<char>>& matrix) {
    if (matrix.empty()) return 0;
    int m = matrix.size(), n = matrix[0].size();
    vector<int> dp(m + 1, 0);
    int maxsize = 0, pre = 0;
    for (int j = 0; j < n; j++) {
        for (int i = 1; i <= m; i++) {
            int temp = dp[i];
            if (matrix[i - 1][j] == '1') {
                dp[i] = min(dp[i], min(dp[i - 1], pre)) + 1;
                maxsize = max(maxsize, dp[i]);
            }
            else dp[i] = 0;
            pre = temp;
        }
    }
    return maxsize * maxsize;
}
```

This solution, since posted, has been suggested various improvements by kind people. For a more comprehensive collection of the solutions, please visit my technical blog (http://www.cnblogs.com/jcliBlogger/p/4548751.html).

solution-sharing (../tag/solution-sharing) dynamic-programming (../tag/dynamic-programming) cpp (../tag/cpp) easy-understand (../tag/easy-understand)

asked (../38489/easy-solution-with-detailed-explanations-8ms-time-and-space) **Jun 2** in **Maximal Square (../oj/maximal-square)** by **jianchao.li.fighter (../user/jianchao.li.fighter)** (58,950 points)
edited **Jun 13** by **jianchao.li.fighter (../user/jianchao.li.fighter)**

Answer comment

nice work! Should pre set to 0 for i = 1? Otherwise, pre is the dp of the end of the previous line.

commented (../38489/easy-solution-with-detailed-explanations-8ms-time-and-space?show=41004#c41004) Jun 18 by **cfjmonkey (../user/cfjmonkey)** reply

In the last solution, `pre` is set to `0` to include the case for the first row.

commented (../38489/easy-solution-with-detailed-explanations-8ms-time-and-space?show=41012#c41012) Jun 18 by **jianchao.li.fighter (../user/jianchao.li.fighter)** reply

Integer to Roman (../oj/integer-to-roman) (74)
Roman to Integer (../oj/roman-to-integer) (110)
Longest Common Prefix (../oj/longest-common-prefix) (157)
3Sum (../oj/3sum)
3Sum Closest (164)
(../oj/3sum-closest)
Letter (45)
Combinations of a Phone Number (../oj/letter-combinations-of-a-phone-number) (123)
4Sum (../oj/4sum)
Remove Nth (80)
Node From End of List (../oj/remove-nth-node-from-end-of-list) (141)
Valid Parentheses (../oj/valid-parentheses) (154)
Merge Two Sorted Lists (../oj/merge-two-sorted-lists) (129)
Generate Parentheses (../oj/generate-parentheses) (140)
Merge k Sorted Lists (../oj/merge-k-sorted-lists) (138)
Swap Nodes in Pairs (../oj/swap-nodes-in-pairs) (128)
Reverse Nodes in k-Group (../oj/reverse-nodes-in-k-group)
Remove (94)
Duplicates from Sorted Array (../oj/remove-duplicates-from-sorted-array) (163)
Remove Element (../oj/remove-element) (170)
Implement strStr() (../oj/implement-strstr) (143)
Divide Two Integers (../oj/divide-two-integers) (106)
Substring with Concatenation of All Words (../oj/substring-with-concatenation-of-all-words) (90)
Next Permutation (../oj/next-permutation) (90)
Longest Valid Parentheses (../oj/longest-valid-parentheses) (97)
Search in Rotated Sorted Array (../oj/search-in-

excellent, thanks for sharing

commented (../38489/easy-solution-with-detailed-explanations-8ms-time-and-space?show=46769#c46769) Jul 18 by **spider8 (../user/spider8)** reply

i think you do not make the [i][j] = min(P[i - 1][j], P[i][j - 1], P[i - 1][j - 1]) + 1 sense or i'm too stupid to get it ....can you explain to me again?

commented (../38489/easy-solution-with-detailed-explanations-8ms-time-and-space?show=52066#c52066) Aug 15 by **yearss (../user/yearss)** reply

2 Answers

+13 votes

Best answer

Good O(n) memory solution. A minor optimization could be to use two pointers to pre and cur and only swap those pointers to avoid vector copying.

Also, if you think about it, it is actually enough to use one vector only instead of two. The whole purpose of maintaining two arrays is that we want to keep the information of pre[i-1]. So we just need to use another variable to keep track of its value:

```
class Solution {
public:
int maximalSquare(vector<vector<char>>& matrix) {
    int nr = matrix.size(); if (nr == 0) return 0;
    int nc = matrix[0].size(); if (nc == 0) return 0;

    vector<int> dp(nc+1, 0);

    int maxsize = 0;

    int last_topleft = 0;  // This is 'pre[i-1]' for the current element

    for (int ir = 1; ir <= nr; ++ir) {
        for (int ic = 1; ic <= nc; ++ic) {
            int temp = dp[ic];      // This is 'pre[i-1]' for the next element
             if (matrix[ir-1][ic-1] == '0') dp[ic] = 0;
            else {
                dp[ic] = min(min(dp[ic], dp[ic-1]), last_topleft) + 1;
                maxsize = max(maxsize, dp[ic]);
            }
            last_topleft = temp;  // update 'pre[i-1]'
        }
    }
    return maxsize * maxsize;
}
};
```

answered (../38489/easy-solution-with-detailed-explanations-8ms-time-and-space?show=38504#a38504) **Jun 2** by **stellari (../user/stellari)** (45,820 points)
selected **Jun 2** by **jianchao.li.fighter (../user/jianchao.li.fighter)**

ask related question comment

Wow, nice observations! I have updated the vector copy of my code to be `swap(pre, cur);` . Do you think it meets your idea? I also mention your one-vector solution in the post above. Thank you!

commented (../38489/easy-solution-with-detailed-explanations-8ms-time-and-space?show=38515#c38515) Jun 2 by **jianchao.li.fighter (../user/jianchao.li.fighter)** reply

Well, actually what I have in mind is to have two pointers to pre and cur as such: vector *pp = &pre, *pc = &cur, and always use* (pp)[i] instead of pre[i]. When we need to swap two vectors, simply do swap(pp, pc). This way, there would be no data copying involved at all. swap(pre, cur) still does data copying behind the scene, so I think it is no better than your original implementation. However, I do agree that using the pointer method may slightly reduce readability. Maybe in a real interview, a good strategy is to write the code in your original way, but point it out to the interviewer that the swapping can actually be optimized as above.

commented (../38489/easy-solution-with-detailed-explanations-8ms-time-and-space?show=38522#c38522) Jun 2 by **stellari (../user/stellari)** reply

Hi, stellari. I haven taken your idea and updated my code as follows.

rotated-sorted-array)
Search for a (80)
Range (../oj/search-
for-a-range) (119)
Search Insert
Position (../oj/search-
insert-position) (108)
Valid Sudoku
(../oj/valid-sudoku)
Sudoku Solver (117)
(../oj/sudoku-solver)
Count and Say (92)
(../oj/count-and-say)
Combination (151)
Sum
(../oj/combination-
sum) (99)
Combination Sum II
(../oj/combination-
sum-ii) (76)
First Missing Positive
(../oj/first-missing-
positive) (90)
Trapping Rain Water
(../oj/trapping-rain-
water) (85)
Multiply Strings
(../oj/multiply-strings)
Wildcard (80)
Matching
(../oj/wildcard-
matching) (75)
Jump Game II
(../oj/jump-game-ii)
Permutations (75)
(../oj/permutations)
Permutations II (120)
(../oj/permutations-ii)
Rotate Image (111)
(../oj/rotate-image)
Anagrams (94)
(../oj/anagrams)(111)
Pow(x, n) (../oj/powx-
n) (110)
N-Queens (../oj/n-
queens) (83)
N-Queens II (../oj/n-
queens-ii) (58)
Maximum Subarray
(../oj/maximum-
subarray) (109)
Spiral Matrix
(../oj/spiral-matrix)
Jump Game (101)
(../oj/jump-game)
Merge Intervals (113)
(../oj/merge-
intervals) (108)
Insert Interval
(../oj/insert-interval)
Length of Last (75)
Word (../oj/length-of-
last-word) (127)
Spiral Matrix II
(../oj/spiral-matrix-ii)
Permutation (86)
Sequence
(../oj/permutation-
sequence) (85)
Rotate List
(../oj/rotate-list) (86)
Unique Paths
(../oj/unique-paths)

```
int maximalSquare(vector<vector<char>>& matrix) {
    if (matrix.empty()) return 0;
    int m = matrix.size(), n = matrix[0].size();
    vector<int> pre(m, 0), cur(m, 0);
    auto ppre = &pre, pcur = &cur;
    int maxsize = 0;
    for (int i = 0; i < m; i++) {
        (*ppre)[i] = matrix[i][0] - '0';
        maxsize = max(maxsize, pre[i]);
    }
    for (int j = 1; j < n; j++) {
        (*pcur)[0] = matrix[0][j] - '0';
        maxsize = max(maxsize, (*pcur)[0]);
        for (int i = 1; i < m; i++) {
            if (matrix[i][j] == '1') {
                (*pcur)[i] = min((*pcur)[i - 1], min((*ppre)[i - 1], (*ppre)[i])) +
                maxsize = max(maxsize, (*pcur)[i]);
            }
        }
        swap(ppre, pcur);
        fill((*pcur).begin(), (*pcur).end(), 0);
    }
    return maxsize * maxsize;
}
```

I guess it is what you want? Thank you for your nice suggestions!

commented (../38489/easy-solution-with-detailed-explanations-8ms-time-and-space?show=38525#c38525) Jun 2 by **jianchao.li.fighter (../user/jianchao.li.fighter)** reply

> *swap(pre, cur) still does data copying behind the scene*

No it doesn't. See this (http://en.cppreference.com/w/cpp/container/vector/swap2) and this (http://en.cppreference.com/w/cpp/container/vector/swap).

commented (../38489/easy-solution-with-detailed-explanations-8ms-time-and-space?show=38605#c38605) Jun 3 by **StefanPochmann (../user/StefanPochmann)** reply

top left can also be checked directly by comparing if it is '1' in the matrix. Assume k = min(dp[ic], dp[ic-1]); Then dp[ic] = (matrix[ir-k-1][ic-k-1] == '1') ? k+1 : k;

commented (../38489/easy-solution-with-detailed-explanations-8ms-time-and-space?show=38630#c38630) Jun 3 by **zhaohui0 (../user/zhaohui0)** reply

Good point! Thank you for this suggestion!

commented (../38489/easy-solution-with-detailed-explanations-8ms-time-and-space?show=38643#c38643) Jun 3 by **stellari (../user/stellari)** reply

Thank you for your tips.

commented (../38489/easy-solution-with-detailed-explanations-8ms-time-and-space?show=38648#c38648) Jun 3 by **jianchao.li.fighter (../user/jianchao.li.fighter)** reply

+3 votes

I think there is no need to use external matrix or vector. we can update the matrix itself.

```
class Solution {
public:
    int maximalSquare(std::vector<std::vector<char> > &matrix) {
        if(matrix.empty())
            return 0;
        int rows = matrix.size(), cols = matrix[0].size();
        char maxSize = '0';
        for (int j = 0; j < cols; ++j)
            if (matrix[0][j] == '1') {
                maxSize = '1';
                break;
            }
        for (int i = 1; i < rows; ++i) {
            maxSize = std::max(maxSize, matrix[i][0]);
            for (int j = 1; j < cols; ++j)
                if (matrix[i][j] == '1') {
                    matrix[i][j] = std::min(matrix[i - 1][j - 1], std::min(matrix[i
                    maxSize = std::max(maxSize, matrix[i][j]);
                }
        }
        return (maxSize - '0') * (maxSize - '0');
    }
};
```

Unique Paths II (../oj/unique-paths-ii) (93)
Minimum Path Sum (../oj/minimum-path-sum) (74) (78)
Valid Number (../oj/valid-number)
Plus One (../oj/plus-one) (133) (125)
Add Binary (../oj/add-binary) (122)
Text Justification (../oj/text-justification)
Sqrt(x) (../oj/sqrtx) (64) (88)
Climbing Stairs (../oj/climbing-stairs)
Simplify Path (../oj/simplify-path) (146)
Edit Distance (../oj/edit-distance) (46)
Set Matrix Zeroes (../oj/set-matrix-zeroes) (52) (76)
Search a 2D Matrix (../oj/search-a-2d-matrix) (105)
Sort Colors (../oj/sort-colors)
Minimum Window Substring (../oj/minimum-window-substring) (123)
Combinations (../oj/combinations) (76)
Subsets (../oj/subsets) (115) (120)
Word Search (../oj/word-search)
Remove Duplicates from Sorted Array II (../oj/remove-duplicates-from-sorted-array-ii) (114) (90)
Search in Rotated Sorted Array II (../oj/search-in-rotated-sorted-array-ii) (41)
Remove Duplicates from Sorted List II (../oj/remove-duplicates-from-sorted-list-ii) (97)
Remove Duplicates from Sorted List (../oj/remove-duplicates-from-sorted-list) (98)
Largest Rectangle in Histogram (../oj/largest-rectangle-in-histogram) (54)
Maximal Rectangle (../oj/maximal-rectangle) (44)
Partition List (../oj/partition-list)
Scramble String (../oj/scramble-string) (79)
Merge Sorted Array (../oj/merge- (44)

answered (../38489/easy-solution-with-detailed-explanations-8ms-time-and-space?show=38506#a38506) **Jun 2** by **prime_tang (../user/prime_tang)** (17,290 points)
edited **Jun 2** by **prime_tang (../user/prime_tang)**

ask related question comment

Hi, your solution is much more memory efficient. From the perspective of problem solving (less time and less memory), it is certainly more desirable. However, personally I would like to keep the input unchanged in functions with return values, which I guess is a good choice in practice :)

commented (../38489/easy-solution-with-detailed-explanations-8ms-time-and-space?show=38512#c38512) Jun 2 by **jianchao.li.fighter (../user/jianchao.li.fighter)** reply

As memory efficient as this solution is, I think it is generally best to avoid any direct modification on the original array. In most situations, updating the input matrix this way is an unexpected behavior, and may cause hard-to-catch bugs. Moreover, you may not even able to get the correct result due to data-type incompatibility. Here, the input is a 2d vector of CHAR, which typically holds values in the range of -128~127. So what if the side of the maximal square exceeds 127? The program would naturally fail, but since the basic logic of your program is correct, it may take a while for the tester to realize that this bug is actually caused by overflowing.

Also, your interviewer may notice this problem earlier than you do and may use this to attack you, and it won't look good if you fail to discover the problem even after he brings it about.

IMHO, we might as well avoid these problems since day 1 by not touching the original array.

commented (../38489/easy-solution-with-detailed-explanations-8ms-time-and-space?show=38526#c38526) Jun 2 by **stellari (../user/stellari)** reply

Hi, stellari, thank you very much for your guidance!

commented (../38489/easy-solution-with-detailed-explanations-8ms-time-and-space?show=38541#c38541) Jun 3 by **prime_tang (../user/prime_tang)** reply

sorted-array) (145)
Gray Code (../oj/gray-code) (99)
Subsets II (../oj/subsets-ii) (67)
Decode Ways (../oj/decode-ways)
Reverse Linked (86) List II (../oj/reverse-linked-list-ii) (104)
Restore IP Addresses (../oj/restore-ip-addresses) (83)
Binary Tree Inorder Traversal (../oj/binary-tree-inorder-traversal)
Unique Binary (85) Search Trees II (../oj/unique-binary-search-trees-ii) (69)
Unique Binary Search Trees (../oj/unique-binary-search-trees) (83)
Interleaving String (../oj/interleaving-string) (83)
Validate Binary Search Tree (../oj/validate-binary-search-tree) (150)
Recover Binary Search Tree (../oj/recover-binary-search-tree) (70)
Same Tree (../oj/same-tree)
Symmetric Tree (120) (../oj/symmetric-tree)
Binary Tree (134) Level Order Traversal (../oj/binary-tree-level-order-traversal)
Binary Tree (111) Zigzag Level Order Traversal (../oj/binary-tree-zigzag-level-order-traversal) (91)
Maximum Depth of Binary Tree (../oj/maximum-depth-of-binary-tree)
Construct (115) Binary Tree from Preorder and Inorder Traversal (../oj/construct-binary-tree-from-preorder-and-inorder-traversal)
Construct Binary (73) Tree from Inorder and Postorder Traversal (../oj/construct-binary-tree-from-inorder-and-postorder-traversal)
Binary Tree (56) Level Order

Traversal II
(../oj/binary-tree-
level-order-traversal-
ii) (81)
Convert Sorted Array
to Binary Search
Tree (../oj/convert-
sorted-array-to-
binary-search-tree)
Convert Sorted (74)
List to Binary Search
Tree (../oj/convert-
sorted-list-to-binary-
search-tree) (76)
Balanced Binary
Tree (../oj/balanced-
binary-tree) (127)
Minimum Depth of
Binary Tree
(../oj/minimum-depth-
of-binary-tree) (130)
Path Sum (../oj/path-
sum) (134)
Path Sum II
(../oj/path-sum-ii)
Flatten Binary (102)
Tree to Linked List
(../oj/flatten-binary-
tree-to-linked-list)
Distinct (126)
Subsequences
(../oj/distinct-
subsequences) (54)
Populating Next
Right Pointers in
Each Node
(../oj/populating-next-
right-pointers-in-
each-node) (134)
Populating Next
Right Pointers in
Each Node II
(../oj/populating-next-
right-pointers-in-
each-node-ii) (111)
Pascal's Triangle
(../oj/pascals-
triangle) (116)
Pascal's Triangle II
(../oj/pascals-
triangle-ii) (120)
Triangle
(../oj/triangle) (113)
Best Time to Buy
and Sell Stock
(../oj/best-time-to-
buy-and-sell-stock)
Best Time to (114)
Buy and Sell Stock II
(../oj/best-time-to-
buy-and-sell-stock-ii)
Best Time to Buy (71)
and Sell Stock III
(../oj/best-time-to-
buy-and-sell-stock-
iii) (66)
Binary Tree
Maximum Path Sum
(../oj/binary-tree-
maximum-path-sum)
Valid Palindrome (83)
(../oj/valid-
palindrome) (121)
Word Ladder II

(../oj/word-ladder-ii)
Word Ladder (109)
(../oj/word-ladder)
Longest (133)
Consecutive
Sequence
(../oj/longest-consecutive-sequence) (97)
Sum Root to Leaf
Numbers (../oj/sum-root-to-leaf-numbers) (89)
Surrounded Regions
(../oj/surrounded-regions) (151)
Palindrome
Partitioning
(../oj/palindrome-partitioning) (95)
Palindrome
Partitioning II
(../oj/palindrome-partitioning-ii) (58)
Clone Graph
(../oj/clone-graph)
Gas Station (101)
(../oj/gas-station)
Candy (100)
(../oj/candy) (114)
Single Number
(../oj/single-number)
Single Number II (92)
(../oj/single-number-ii) (82)
Copy List with
Random Pointer
(../oj/copy-list-with-random-pointer)
Word Break (128)
(../oj/word-break)
Word Break II (150)
(../oj/word-break-ii)
Linked List (140)
Cycle (../oj/linked-list-cycle) (101)
Linked List Cycle II
(../oj/linked-list-cycle-ii) (98)
Reorder List
(../oj/reorder-list)
Binary Tree (113)
Preorder Traversal
(../oj/binary-tree-preorder-traversal)
Binary Tree (109)
Postorder Traversal
(../oj/binary-tree-postorder-traversal)
LRU Cache (154)
(../oj/lru-cache) (191)
Insertion Sort List
(../oj/insertion-sort-list) (129)
Sort List (../oj/sort-list) (182)
Max Points on a Line
(../oj/max-points-on-a-line) (132)
Evaluate Reverse
Polish Notation
(../oj/evaluate-reverse-polish-notation) (131)

Reverse Words in a String (../oj/reverse-words-in-a-string)	(227)
Maximum Product Subarray (../oj/maximum-product-subarray)	(91)
Find Minimum in Rotated Sorted Array (../oj/find-minimum-in-rotated-sorted-array)	(124)
Find Minimum in Rotated Sorted Array II (../oj/find-minimum-in-rotated-sorted-array-ii)	(86)
Min Stack (../oj/min-stack)	(195)
Binary Tree Upside Down (../oj/binary-tree-upside-down)	(17)
Read N Characters Given Read4 (../oj/read-n-characters-given-read4)	(26)
Read N Characters Given Read4 II - Call multiple times (../oj/read-n-characters-given-read4-ii-call-multiple-times)	(16)
Longest Substring with At Most Two Distinct Characters (../oj/longest-substring-with-at-most-two-distinct-characters)	(23)
Intersection of Two Linked Lists (../oj/intersection-of-two-linked-lists)	(155)
One Edit Distance (../oj/one-edit-distance)	(24)
Find Peak Element (../oj/find-peak-element)	(106)
Missing Ranges (../oj/missing-ranges)	(27)
Maximum Gap (../oj/maximum-gap)	(54)
Compare Version Numbers (../oj/compare-version-numbers)	(151)
Fraction to Recurring Decimal (../oj/fraction-to-recurring-decimal)	(64)
Two Sum II - Input array is sorted (../oj/two-sum-ii-input-array-is-sorted)	(6)
Excel Sheet Column Title (../oj/excel-sheet-column-title)	(88)
Majority Element (../oj/majority-element)	(132)

Two Sum III - Data structure design (../oj/two-sum-iii-data-structure-design)	(15)
Excel Sheet Column Number (../oj/excel-sheet-column-number)	(91)
Factorial Trailing Zeroes (../oj/factorial-trailing-zeroes)	(64)
Binary Search Tree Iterator (../oj/binary-search-tree-iterator)	(81)
Dungeon Game (../oj/dungeon-game)	(79)
Combine Two Tables (../oj/combine-two-tables)	(11)
Second Highest Salary (../oj/second-highest-salary)	(33)
Nth Highest Salary (../oj/nth-highest-salary)	(19)
Rank Scores (../oj/rank-scores)	(26)
Largest Number (../oj/largest-number)	(125)
Consecutive Numbers (../oj/consecutive-numbers)	(21)
Employees Earning More Than Their Managers (../oj/employees-earning-more-than-their-managers)	(16)
Duplicate Emails (../oj/duplicate-emails)	(19)
Customers Who Never Order (../oj/customers-who-never-order)	(13)
Department Highest Salary (../oj/department-highest-salary)	(32)
Department Top Three Salaries (../oj/department-top-three-salaries)	(30)
Reverse Words in a String II (../oj/reverse-words-in-a-string-ii)	(21)
Repeated DNA Sequences (../oj/repeated-dna-sequences)	(126)
Best Time to Buy and Sell Stock IV (../oj/best-time-to-buy-and-sell-stock-iv)	(47)
Rotate Array (../oj/rotate-array)	(225)
Reverse Bits (../oj/reverse-bits)	

Number of 1 (161)
Bits (../oj/number-of-1-bits) (148)
Word Frequency
(../oj/word-frequency) (16)
Valid Phone
Numbers (../oj/valid-phone-numbers) (12)
Transpose File
(../oj/transpose-file)
Tenth Line (17)
(../oj/tenth-line) (11)
Delete Duplicate
Emails (../oj/delete-duplicate-emails) (20)
Rising Temperature
(../oj/rising-temperature) (27)
House Robber
(../oj/house-robber)
Binary Tree (122)
Right Side View
(../oj/binary-tree-right-side-view) (124)
Number of Islands
(../oj/number-of-islands) (125)
Bitwise AND of
Numbers Range
(../oj/bitwise-and-of-numbers-range) (83)
Happy Number
(../oj/happy-number)
Remove Linked (135)
List Elements
(../oj/remove-linked-list-elements) (108)
Count Primes
(../oj/count-primes)
Isomorphic (97)
Strings
(../oj/isomorphic-strings) (119)
Reverse Linked List
(../oj/reverse-linked-list) (130)
Course Schedule
(../oj/course-schedule) (94)
Implement Trie
(Prefix Tree)
(../oj/implement-trie-prefix-tree) (87)
Minimum Size
Subarray Sum
(../oj/minimum-size-subarray-sum) (89)
Course Schedule II
(../oj/course-schedule-ii) (68)
Add and Search
Word - Data
structure design
(../oj/add-and-search-word-data-structure-design) (67)
Word Search II
(../oj/word-search-ii)
House Robber II (49)
(../oj/house-robber-ii)
Shortest (70)
Palindrome
(../oj/shortest-

palindrome) (78)

Kth Largest Element in an Array (../oj/kth-largest-element-in-an-array) (81)

Combination Sum III (../oj/combination-sum-iii) (80)

Contains Duplicate (../oj/contains-duplicate) (81)

The Skyline Problem (../oj/the-skyline-problem) (51)

Contains Duplicate II (../oj/contains-duplicate-ii) (90)

Contains Duplicate III (../oj/contains-duplicate-iii) (57)

Maximal Square (../oj/maximal-square) (54)

Count Complete Tree Nodes (../oj/count-complete-tree-nodes) (80)

Rectangle Area (../oj/rectangle-area) (82)

Basic Calculator (../oj/basic-calculator) (72)

Implement Stack using Queues (../oj/implement-stack-using-queues) (90)

Invert Binary Tree (../oj/invert-binary-tree) (113)

Basic Calculator II (../oj/basic-calculator-ii) (65)

Summary Ranges (../oj/summary-ranges) (111)

Majority Element II (../oj/majority-element-ii) (60)

Kth Smallest Element in a BST (../oj/kth-smallest-element-in-a-bst) (95)

Power of Two (../oj/power-of-two) (75)

Implement Queue using Stacks (../oj/implement-queue-using-stacks) (63)

Number of Digit One (../oj/number-of-digit-one) (51)

Palindrome Linked List (../oj/palindrome-linked-list) (141)

Lowest Common Ancestor of a Binary Search Tree (../oj/lowest-common-ancestor-of-a-binary-search-tree) (99)

Lowest Common Ancestor of a Binary Tree (../oj/lowest-common-ancestor-

of-a-binary-tree)	(80)
Delete Node in a Linked List (../oj/delete-node-in-a-linked-list)	(60)
Product of Array Except Self (../oj/product-of-array-except-self)	(73)
Sliding Window Maximum (../oj/sliding-window-maximum)	(78)
Search a 2D Matrix II (../oj/search-a-2d-matrix-ii)	(71)
Different Ways to Add Parentheses (../oj/different-ways-to-add-parentheses)	(56)
Valid Anagram (../oj/valid-anagram)	(106)
Shortest Word Distance (../oj/shortest-word-distance)	(19)
Shortest Word Distance II (../oj/shortest-word-distance-ii)	(14)
Shortest Word Distance III (../oj/shortest-word-distance-iii)	(17)
Strobogrammatic Number (../oj/strobogrammatic-number)	(13)
Strobogrammatic Number II (../oj/strobogrammatic-number-ii)	(20)
Strobogrammatic Number III (../oj/strobogrammatic-number-iii)	(16)
Group Shifted Strings (../oj/group-shifted-strings)	(17)
Count Univalue Subtrees (../oj/count-univalue-subtrees)	(19)
Flatten 2D Vector (../oj/flatten-2d-vector)	(21)
Meeting Rooms (../oj/meeting-rooms)	(12)
Meeting Rooms II (../oj/meeting-rooms-ii)	(34)
Factor Combinations (../oj/factor-combinations)	(15)
Verify Preorder Sequence in Binary Search Tree (../oj/verify-preorder-sequence-in-binary-search-tree)	(11)
Paint House (../oj/paint-house)	(11)
Binary Tree Paths (../oj/binary-tree-paths)	(90)

Add Digits (../oj/add-digits)	(70)
3Sum Smaller (../oj/3sum-smaller)	(22)
Single Number III (../oj/single-number-iii)	(25)
Graph Valid Tree (../oj/graph-valid-tree)	(28)
Trips and Users (../oj/trips-and-users)	(9)
Ugly Number (../oj/ugly-number)	(39)
Ugly Number II (../oj/ugly-number-ii)	(46)
Paint House II (../oj/paint-house-ii)	(22)
Palindrome Permutation (../oj/palindrome-permutation)	(16)
Palindrome Permutation II (../oj/palindrome-permutation-ii)	(18)
Missing Number (../oj/missing-number)	(92)
Alien Dictionary (../oj/alien-dictionary)	(29)
Closest Binary Search Tree Value (../oj/closest-binary-search-tree-value)	(18)
Encode and Decode Strings (../oj/encode-and-decode-strings)	(23)
Closest Binary Search Tree Value II (../oj/closest-binary-search-tree-value-ii)	(27)
Integer to English Words (../oj/integer-to-english-words)	(42)
H-Index (../oj/h-index)	(87)
H-Index II (../oj/h-index-ii)	(29)
Paint Fence (../oj/paint-fence)	(21)
Find the Celebrity (../oj/find-the-celebrity)	(14)
First Bad Version (../oj/first-bad-version)	(65)
Perfect Squares (../oj/perfect-squares)	(64)
Wiggle Sort (../oj/wiggle-sort)	(15)
Zigzag Iterator (../oj/zigzag-iterator)	(28)
Expression Add Operators (../oj/expression-add-operators)	(26)
Move Zeroes (../oj/move-zeroes)	(118)
Peeking Iterator (../oj/peeking-iterator)	(43)

- Inorder Successor in BST (../oj/inorder-successor-in-bst) (18)
- Walls and Gates (../oj/walls-and-gates) (21)
- Find the Duplicate Number (../oj/find-the-duplicate-number) (49)
- Unique Word Abbreviation (../oj/unique-word-abbreviation) (17)
- Game of Life (../oj/game-of-life) (47)
- Word Pattern (../oj/word-pattern) (52)
- Word Pattern II (../oj/word-pattern-ii) (14)
- Nim Game (../oj/nim-game) (33)
- Flip Game (../oj/flip-game) (14)
- Flip Game II (../oj/flip-game-ii) (3)

Interview Questions (../interview-questions) (93)