

DSA Questions

Array

Q . Find Unique Number (single Number)

136. Single Number

Easy 11.6K 441

Companies

Given a **non-empty** array of integers `nums`, every element appears *twice* except for one. Find that single one.

You must implement a solution with a linear runtime complexity and use only constant extra space.

Example 1:

Input: `nums = [2,2,1]`
Output: `1`

Example 2:

Input: `nums = [4,1,2,1,2]`
Output: `4`

Example 3:

Input: `nums = [1]`
Output: `1`

```
1 class Solution {
2 public:
3     int singleNumber(vector<int>& nums) {
4         int n = nums.size();
5         int ans = 0 ;
6         for(int i = 0 ;i<n ; i++){
7             ans = ans^nums[i];
8         }
9         return ans ;
10    }
11 }
12 ;;
```

Console

Q. Sort Colors

75. Sort Colors

Medium 15.2K 541

Companies

Given an array `nums` with `n` objects colored red, white, or blue, sort them **in-place** so that objects of the same color are adjacent, with the colors in the order red, white, and blue.

We will use the integers `0`, `1`, and `2` to represent the color red, white, and blue, respectively.

You must solve this problem without using the library's sort function.

Example 1:

Input: `nums = [2,0,2,1,1,0]`
Output: `[0,0,1,1,2,2]`

Example 2:

Input: `nums = [2,0,1]`
Output: `[0,1,2]`

Constraints:

```
1 class Solution {
2 public:
3
4     void sortColors(vector<int>& a) {
5         int n = a.size();
6
7         int l = 0 , m = 0 , h = n-1;
8         while(m <= h){
9             if(a[m] == 0){
10                 swap(a[l],a[m]);
11                 l++ ;
12                 m++ ;
13             }else if(a[m] == 1){
14                 m++ ;
15             }else if(a[m] == 2){
16                 swap(a[h],a[m]);
17                 h-- ;
18             }
19         }
20     }
21 }
22 ;;
```

Console

Q. Maximum sum of Subarray (Kadane's Algorithm)

53. Maximum Subarray

Medium



25.3K

1.2K



Companies

Given an integer array `nums`, find the contiguous subarray (containing at least one number) which has the largest sum and return *its sum*.

A **subarray** is a **contiguous** part of an array.

Example 1:

Input: `nums = [-2,1,-3,4,-1,2,1,-5,4]`

Output: 6

Explanation: `[4,-1,2,1]` has the largest sum = 6.

Example 2:

Input: `nums = [1]`

Output: 1

Example 3:

Input: `nums = [5,4,-1,7,8]`

Output: 23

```
2 public:
3     int maxSubArray(vector<int>& nums) {
4         int sum = 0 ;
5         int ans = nums[0] ;
6         int n = nums.size() ;
7
8         for(int i = 0; i<n ;i++){
9
10            if (sum<0)
11                sum = 0 ;
12
13            sum += nums[i];
14            ans = max(sum , ans);
15        }
16        return ans;
17    }
18 };
```

Console



Run

Q. Maximum Product Subarray

152. Maximum Product Subarray

Medium



13.8K

414



Companies

Given an integer array `nums`, find a contiguous non-empty subarray within the array that has the largest product, and return *the product*.

The test cases are generated so that the answer will fit in a **32-bit** integer.

A **subarray** is a contiguous subsequence of the array.

Example 1:

Input: `nums = [2,3,-2,4]`

Output: 6

Explanation: `[2,3]` has the largest product 6.

Example 2:

Input: `nums = [-2,0,-1]`

Output: 0

Explanation: The result cannot be 2, because `[-2,-1]` is not a subarray.

```
1 class Solution {
2 public:
3     int maxProduct(vector<int>& nums) {
4         int n = nums.size();
5         int ans = nums[0];
6         int prod = 1 ;
7         int x = 1 ;
8         int ma = ans;
9         int mi = ans ;
10
11        for(int i = 1 ; i < n ; i++){
12            if(nums[i]<0){
13                swap(ma , mi);
14            }
15            ma = max(nums[i] , ma * nums[i]);
16            mi = min(nums[i] , mi * nums[i]);
17            ans = max(ma , ans);
18        }
19
20
21        return ans ;
22    }
23 }
24 };
```

Console

Q. Best Time to Buy and Sell Stock I

121. Best Time to Buy and Sell Stock

Easy

20.4K 656

Companies

You are given an array `prices` where `prices[i]` is the price of a given stock on the i^{th} day.

You want to maximize your profit by choosing a **single day** to buy one stock and choosing a **different day in the future** to sell that stock.

Return the *maximum profit* you can achieve from this transaction. If you cannot achieve any profit, return `0`.

Example 1:

Input: `prices = [7,1,5,3,6,4]`
Output: `5`
Explanation: Buy on day 2 (price = 1) and sell on day 5 (price = 6), profit = 6 - 1 = 5.
Note that buying on day 2 and selling on day 1 is not allowed because you must buy before you sell.

Example 2:

Input: `prices = [7,6,4,3,1]`
Output: `0`
Explanation: In this case, no transactions are done and the max profit = 0.

```
1 class Solution {
2 public:
3     int maxProfit(vector<int>& prices) {
4         int n = prices.size();
5         int buy = prices.at(0) ;
6         int pro = 0 ;
7         for(int i =0 ; i<n ; i++){
8             if(buy > prices[i]){
9                 buy = prices[i];
10            } else if(prices[i]-buy > pro){
11                pro = prices[i]-buy;
12            }
13        }
14        return pro ;
15    }
16 };
```

Q. Best Time to Buy and Sell Stock II

122. Best Time to Buy and Sell Stock II

Medium

9.3K 2.5K

Companies

You are given an integer array `prices` where `prices[i]` is the price of a given stock on the i^{th} day.

On each day, you may decide to buy and/or sell the stock. You can only hold **at most one** share of the stock at any time. However, you can buy it then immediately sell it on the **same day**.

Find and return the *maximum profit* you can achieve.

Example 1:

Input: `prices = [7,1,5,3,6,4]`
Output: `7`
Explanation: Buy on day 2 (price = 1) and sell on day 3 (price = 5), profit = 5 - 1 = 4.
Then buy on day 4 (price = 3) and sell on day 5 (price = 6), profit = 6 - 3 = 3.
Total profit is 4 + 3 = 7.

Example 2:

Input: `prices = [1,2,3,4,5]`
Output: `4`
Explanation: Buy on day 1 (price = 1) and sell on day 5 (price = 5), profit = 5 - 1 = 4.

```
1 class Solution {
2 public:
3     int maxProfit(vector<int>& prices) {
4         int n = prices.size();
5         int buy = prices[0];
6         int profit = 0 ;
7         int netProfit = 0;
8
9         for(int i = 1 ; i <n ; i++){
10             if(buy < prices[i]){
11                 profit = prices[i]-buy;
12                 buy = prices[i] ;
13                 netProfit += profit ;
14             } else{
15                 buy = prices[i];
16             }
17         }
18
19         return netProfit ;
20     }
21 };
```

Q. Container with Most Water

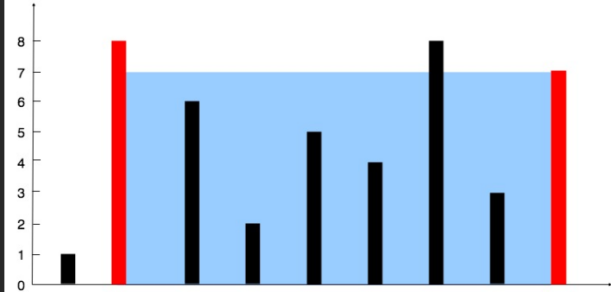
You are given an integer array `height` of length `n`. There are `n` vertical lines drawn such that the two endpoints of the i^{th} line are $(i, 0)$ and $(i, \text{height}[i])$.

Find two lines that together with the x-axis form a container, such that the container contains the most water.

Return the maximum amount of water a container can store.

Notice that you may not slant the container.

Example 1:



Input: `height = [1,8,6,2,5,4,8,3,7]`

Output: 49

Explanation: The above vertical lines are represented by array `[1,8,6,2,5,4,8,3,7]`. In this case, the max area of water (blue section) the container can contain is 49.

```
1 class Solution {
2 public:
3     int maxArea(vector<int>& height) {
4         int l = height.size() - 1;
5         int area;
6         int b = 0;
7         int maxA = 0;
8
9         int i = 0;
10
11         while(i < l){
12             b = min (height[i], height[l]);
13             area = (l-i) * b;
14             maxA = max (maxA, area);
15
16             if(height[i] < height[l]){
17                 i++;
18             }else{
19                 l--;
20             }
21         }
22         return maxA;
23     }
24 }
```

Q. Set Matrix Zero

73. Set Matrix Zeroes

Medium



9.3K

555



Companies

Given an $m \times n$ integer matrix `matrix`, if an element is 0, set its entire row and column to 0's.

You must do it *in place*.

Example 1:

1	1	1		1	0	1
1	0	1	→	0	0	0
1	1	1		1	0	1

Input: `matrix = [[1,1,1],[1,0,1],[1,1,1]]`

Output: `[[1,0,1],[0,0,0],[1,0,1]]`

Example 2:

```
1 class Solution {
2 public:
3     void setZeroes(vector<vector<int>>& matrix) {
4         int r = matrix.size();
5         int c = matrix[0].size();
6         set<int> row;
7         set<int> column;
8
9         for(int i = 0; i < r; i++){
10             for(int j = 0; j < c; j++){
11                 if (matrix[i][j] == 0){
12                     row.insert(i);
13                     column.insert(j);
14                 }
15             }
16         }
17
18         for(auto i : row){
19             for(int j = 0; j < c; j++){
20                 matrix[i][j] = 0;
21             }
22         }
23         for(auto i : column){
24             for(int j = 0; j < r; j++){
25                 matrix[j][i] = 0;
26             }
27         }
28     }
29 }
30 };
```

Q. Pascal's Triangle I


118. Pascal's Triangle

Easy 8K 266

Companies

Given an integer `numRows`, return the first `numRows` of **Pascal's triangle**.

In **Pascal's triangle**, each number is the sum of the two numbers directly above it as shown:



Example 1:

Input: `numRows = 5`
Output: `[[1], [1,1], [1,2,1], [1,3,3,1], [1,4,6,4,1]]`

```
1 class Solution {
2 public:
3     vector<vector<int>> generate(int numRows) {
4         vector<vector<int>> ans (numRows) ;
5
6         for(int i = 0 ; i<numRows ; i++){
7             ans[i]= vector<int>(i+1);
8             for (int j = 0 ; j<i+1; j++){
9                 if(j>0 && j < i){
10                     ans[i][j] = ans[i-1][j-1] + ans[i-1][j];
11                 }else{
12                     ans[i][j] = 1;
13                 }
14             }
15         }
16         return ans ;
17     }
18 };
```

Q. Pascal's Triangle II


119. Pascal's Triangle II

Easy 3.2K 281

Companies

Given an integer `rowIndex`, return the `rowIndexth` (**0-indexed**) row of the **Pascal's triangle**.

In **Pascal's triangle**, each number is the sum of the two numbers directly above it as shown:



Example 1:

Input: `rowIndex = 3`
Output: `[1,3,3,1]`

```
1 class Solution {
2 public:
3     vector<int> getRow(int rowIndex) {
4         vector<vector<int>> ans (rowIndex+1) ;
5
6         for(int i = 0 ; i<=rowIndex ; i++){
7             ans[i]= vector<int>(i+1);
8             for (int j = 0 ; j<i+1; j++){
9                 if(j>0 && j < i){
10                     ans[i][j] = ans[i-1][j-1] + ans[i-1][j];
11                 }else{
12                     ans[i][j] = 1;
13                 }
14             }
15         }
16
17         return ans[rowIndex] ;
18     }
19 };
20 };
```

Q . Spiral Matrix

54. Spiral Matrix

Medium 9.1K 925 ☆ ↻

Companies

Given an $m \times n$ matrix, return all elements of the matrix in spiral order.

Example 1:

1	→	2	→	3
4	→	5		↓
↑		↓		↓
7	←	8	←	9

Input: matrix = [[1,2,3],[4,5,6],[7,8,9]]

Output: [1,2,3,6,9,8,7,4,5]

Example 2:

1	→	2	→	3	→	4
5	→	6	→	7		↓
↑						↓

```
2 public:
3     vector<int> spiralOrder(vector<vector<int>>& matrix) {
4         int row = matrix.size();
5         int col = matrix[0].size();
6         vector<int> ans ;
7
8         int l = 0 , r = col-1 , t = 0 , b = row -1 , d = 0;
9
10        while (l<=r && t<=b){
11            switch(d){
12                case 0 :
13                    for(int i = l ; i<= r ; i++){
14                        ans.push_back(matrix[t][i]);
15                    }
16                    t++;
17                    d = 1;
18                    break;
19                case 1 :
20                    for(int i = t ; i<= b ; i++){
21                        ans.push_back(matrix[i][r]);
22                    }
23                    r--;
24                    d = 2;
25                    break;
26                case 2 :
27                    for(int i = r ; i>= l ; i--){
28                        ans.push_back(matrix[b][i]);
29                    }
30                    b--;
31                    d = 3;
32                    break;
33                case 3 :
34                    for(int i = b ; i>= t ; i--){
35                        ans.push_back(matrix[i][l]);
36                    }
37                    l++;
38                    d = 0 ;
39                    break ;
40            }
41        }
42        return ans ;
43    }
```

Console

Run

Strings

Searching And Sorting

Q. Merge Sort

```
void merge(int arr[], int l, int m, int r)
{
    // Your code here
    int len1 = m - l + 1 ;
    int len2 = r-m ;
    int first[len1] ;
    int second [len2] ;

    int k = l ;
    // cout<<"first " ;
    for(int i = 0 ; i<len1 ; i++){
        first[i] = arr[k++] ;
        // cout<<first[i]<<" " ;
    }
    // cout<<endl ;
    k = m + 1 ;
    // cout<<"second " ;
    for(int i = 0 ; i<len2 ; i++){
        second[i] = arr[k++] ;
        // cout<<second[i]<<" " ;
    }
    // cout<<endl ;

    // merge two sorted array
    int index1 = 0 ;
    int index2 = 0 ;
    k=l ;
    while(index1 < len1 && index2 <len2){
        if(first[index1]<=second[index2]){
            arr[k++] = first[index1++] ;
        }else{
            arr[k++] = second[index2++] ;
        }
    }

    while(index1 < len1){
        arr[k++] = first[index1++] ;
    }
```

```

    }
    while(index2 < len2){
        arr[k++] = second[index2++] ;
    }
}

```

```

void mergeSort(int arr[], int l, int r)
{
    //code here
    if(l>=r){
        return ;
    }
    int mid = l + (r-l)/2 ;
    // cout<<l <<r <<endl;

    mergeSort(arr , l , mid) ;
    mergeSort(arr , mid+1 , r) ;

    merge(arr , l , mid , r) ;

}

```

Q. Quick sort

```

int partition (int arr[], int low, int high)
{
    // Your code here

    int pivot = arr[low] ;

    int cnt = 0 ;
    for(int i = low+1 ; i<=high ; i++){
        if(pivot >= arr[i]){
            cnt++ ;
        }
    }
    int pivotindex = low + cnt ;

    swap(arr[pivotindex] , arr[low]);

    // correct left or right part

    int i = low , j = high ;

```

```

while(i<pivotindex && j>pivotindex){
    while(arr[i]<pivot){
        i++ ;
    }
    while(arr[j]> pivot){
        j-- ;
    }

    if(i<pivotindex && j>pivotindex){
        swap(arr[i++], arr[j--]) ;
    }
}

return pivotindex ;
}

void quickSort(int arr[], int low, int high)
{
    // code here
    if(low>= high)
        return ;

    int p = partition(arr , low , high);

    quickSort(arr , low , p-1) ;

    quickSort(arr , p+1 , high) ;
}

```

Recursion

Q. Fibonacci Number (509)

DescriptionEditorialSolutions (8.8K)Submissions

509. Fibonacci Number

Easy✔7.3K👍325☆🔄

Companies

The **Fibonacci numbers**, commonly denoted $F(n)$ form a sequence, called the **Fibonacci sequence**, such that each number is the sum of the two preceding ones, starting from 0 and 1. That is,

$$F(0) = 0, F(1) = 1$$
$$F(n) = F(n - 1) + F(n - 2), \text{ for } n > 1.$$

Given n , calculate $F(n)$.

Example 1:

Input: $n = 2$
Output: 1
Explanation: $F(2) = F(1) + F(0) = 1 + 0 = 1$.

Example 2:

Input: $n = 3$
Output: 2
Explanation: $F(3) = F(2) + F(1) = 1 + 1 = 2$.

/ C++Auto

```
1 class Solution {
2     int fibonacci(int n , vector<int>& dp){
3         if( n <= 1){
4             dp[n] = n ;
5             return n;
6         }
7         if(dp[n] != -1){
8             return dp[n] ;
9         }
10        dp[n] = fibonacci(n-1 , dp) + fibonacci(n-2 , dp) ;
11        return dp[n] ;
12    }
13 public:
14     int fib(int n) {
15         vector<int> dp (n+1 , -1) ;
16         return fibonacci(n , dp) ;
17     }
18 }
19 ;
```

DescriptionEditorialSolutions (8.8K)Submissions

509. Fibonacci Number

Easy✔7.3K👍325☆🔄

Companies

The **Fibonacci numbers**, commonly denoted $F(n)$ form a sequence, called the **Fibonacci sequence**, such that each number is the sum of the two preceding ones, starting from 0 and 1. That is,

$$F(0) = 0, F(1) = 1$$
$$F(n) = F(n - 1) + F(n - 2), \text{ for } n > 1.$$

Given n , calculate $F(n)$.

Example 1:

Input: $n = 2$
Output: 1
Explanation: $F(2) = F(1) + F(0) = 1 + 0 = 1$.

Example 2:

Input: $n = 3$
Output: 2
Explanation: $F(3) = F(2) + F(1) = 1 + 1 = 2$.

/ C++Auto

```
1 class Solution {
2     int fibonacci(int n , vector<int>& dp){
3         if( n <= 1){
4             dp[n] = n ;
5             return n;
6         }
7         if(dp[n] != -1){
8             return dp[n] ;
9         }
10        dp[n] = fibonacci(n-1 , dp) + fibonacci(n-2 , dp) ;
11        return dp[n] ;
12    }
13 public:
14     int fib(int n) {
15         vector<int> dp (n+1 , -1) ;
16         return fibonacci(n , dp) ;
17     }
18 }
19 ;
```

Q. Pow(x, n)

DescriptionEditorialSolutions (6.4K)Submissions

50. Pow(x, n)

Medium

8.6K8.4K

Companies

Implement `pow(x, n)`, which calculates `x` raised to the power `n` (i.e., x^n).

Example 1:

Input: `x = 2.00000, n = 10`
Output: `1024.00000`

Example 2:

Input: `x = 2.10000, n = 3`
Output: `9.26100`

Example 3:

Input: `x = 2.00000, n = -2`
Output: `0.25000`
Explanation: $2^{-2} = 1/2^2 = 1/4 = 0.25$

Constraints:

i C++ | Auto

```
1 class Solution {
2     double solve(double x, int n){
3         if(n == 0)
4             return 1 ;
5
6         if(n==1)
7             return x ;
8
9         double ans = solve(x , n/2) ;
10
11         if(n%2==0){
12             return ans * ans ;
13         }else{
14             return x * ans * ans ;
15         }
16     }
17 public:
18     double myPow(double x, int n) {
19         double ans = 0 ;
20         if(n < 0){
21             x = 1/x ;
22             n = abs(n) ;
23         }
24         ans = solve(x , n) ;
25         return ans ;
26     }
27 };
```

Q. Subsets

DescriptionEditorialSolutions (8.5K)Submissions

78. Subsets

Medium

15.5K224

Companies

Given an integer array `nums` of **unique** elements, return *all possible subsets (the power set)*.

The solution set **must not** contain duplicate subsets. Return the solution in **any order**.

Example 1:

Input: `nums = [1,2,3]`
Output: `[[], [1], [2], [1,2], [3], [1,3], [2,3], [1,2,3]]`

Example 2:

Input: `nums = [0]`
Output: `[[], [0]]`

Constraints:

- `1 <= nums.length <= 10`
- `-10 <= nums[i] <= 10`
- All the numbers of `nums` are **unique**.

i C++ | Auto

```
1 class Solution {
2     void solve(int i , vector<int>& nums , vector<vector<int>>& ans , vector<int> temp){
3         if(i == nums.size()){
4             ans.push_back(temp) ;
5             return ;
6         }
7
8         // exclude element
9         solve(i+1 , nums , ans , temp) ;
10
11         // include element
12         temp.push_back(nums[i]) ;
13         solve(i+1 , nums , ans , temp) ;
14     }
15 public:
16     vector<vector<int>> subsets(vector<int>& nums) {
17         vector<vector<int>> ans ;
18         vector<int> temp ;
19         solve (0 , nums , ans , temp) ;
20         return ans ;
21     }
22 };
```

Console ^

Run

Q. Subset Sums

Subset Sums

Medium Accuracy: 72.55% Submissions: 82K+ Points: 4

Done with this problem? Now use these skills to apply for a Job in Job-A-Thon 21!

Given a list **arr** of **N** integers, print sums of all subsets in it.

Example 1:

Input:
N = 2
arr[] = {2, 3}

Output:
0 2 3 5

Explanation:
When no elements is taken then Sum = 0.
When only 2 is taken then Sum = 2.
When only 3 is taken then Sum = 3.
When element 2 and 3 are taken then
Sum = 2+3 = 5.

```
1 // Driver Code Starts
2 class Solution
3 {
4     void solve(int index , int sum , vector<int>& arr , int n , vector<int>& ans){
5         if(index == n){
6             ans.push_back(sum) ;
7             return ;
8         }
9         solve(index+1 , sum , arr , n , ans) ;
10        // include element
11        sum += arr[index] ;
12        solve(index+1 , sum , arr , n , ans) ;
13    }
14    public:
15        vector<int> subsetSums(vector<int> arr, int N)
16        {
17            // Write Your Code here
18            vector<int> ans ;
19            int sum = 0 ;
20            solve(0 , sum , arr , N , ans) ;
21            return ans ;
22        }
23    };
24    // } Driver Code Ends
```

Custom Input

Compile & Run

End

BackTracking

Q. Letter Combinations of a Phone Number

DescriptionEditorialSolutions (11.4K)Submissions


17. Letter Combinations of a Phone Number

Medium✔16.9K892

Companies

Given a string containing digits from 2-9 inclusive, return all possible letter combinations that the number could represent. Return the answer in **any order**.

A mapping of digits to letters (just like on the telephone buttons) is given below. Note that 1 does not map to any letters.



Example 1:

Input: digits = "23"

Output: ["ad", "ae", "af", "bd", "be", "bf", "cd", "ce", "cf"]

i C++ | Auto

```
1 class Solution {
2     void solve(string digit , string output ,int index ,vector<string>& ans){
3         // string output ;
4         string mapping[10] = {"", "", "abc", "def", "ghi", "jkl", "mno", "pqrs", "tuv", "wxyz"} ;
5         if(index == digit.length()){
6             ans.push_back(output);
7             return ;
8         }
9         int number = digit[index] - '0' ;
10        string value = mapping[number] ;
11        for(int i =0 ; i<value.length(); i++){
12            output.push_back(value[i]);
13            solve(digit , output ,index+1 , ans );
14            output.pop_back() ;
15        }
16    }
17
18 public:
19     vector<string> letterCombinations(string digits) {
20         vector<string> ans ;
21         string output ;
22         if(digits.length() == 0){
23             return ans ;
24         }
25         int index = 0 ;
26         solve (digits , output , index,  ans ) ;
27         return ans ;
28     }
29 };
30
```

Ln 11, Col 31

Console ^

Run

Submit

Q. Permutations

DescriptionEditorialSolutions (8.4K)Submissions

46. Permutations

Medium✔17.6K282

Companies

Given an array `nums` of distinct integers, return *all the possible permutations*. You can return the answer in **any order**.

Example 1:

Input: nums = [1,2,3]

Output: [[1,2,3], [1,3,2], [2,1,3], [2,3,1], [3,1,2], [3,2,1]]

Example 2:

Input: nums = [0,1]

Output: [[0,1], [1,0]]

Example 3:

Input: nums = [1]

Output: [[1]]

i C++ | Auto

```
1 class Solution {
2     void solve(int idx,vector<vector<int>>&ans,vector<int>&nums){
3         if(nums.size()==idx){
4             ans.push_back(nums);
5             return ;
6         }
7
8         for(int i=idx;i<nums.size();i++){
9             swap(nums[idx],nums[i]);
10            solve(idx+1 , ans,nums);
11            //backtracking
12            swap(nums[idx],nums[i]);
13        }
14    }
15 public:
16     vector<vector<int>> permute(vector<int>& nums) {
17         vector<vector<int>> ans ;
18         solve(0,ans,nums);
19         return ans;
20     }
21 };

```

Q. Rat In a Maze

Rat In a Maze

Last Updated: 11 Jul, 2023

Hard 0/120

6 upvotes



Problem Statement

Suggest Edit

You are given a $N \times N$ maze with a rat placed at ' $mat[0][0]$ '. Find all paths that rat can follow to reach its destination i.e. $mat[N-1][N-1]$. The directions in which the rat can move are 'U'(up), 'D'(down), 'L' (left), 'R' (right).

In the given maze, each cell can have a value of either 0 or 1. Cells with a value of 0 are considered blocked, which means the rat cannot enter or traverse through them. On the other hand, cells with a value of 1 are open, indicating that the rat is allowed to enter and move through those cells.

Example:

```
mat:{{1, 0, 0, 0},
      {1, 1, 0, 1},
      {1, 1, 0, 0},
      {0, 1, 1, 1}}
```

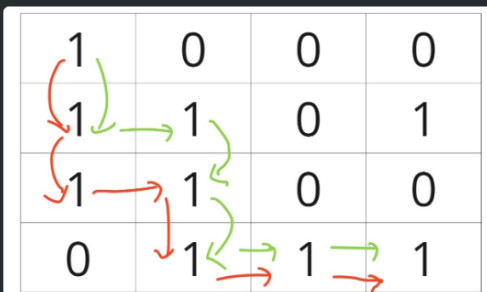
All possible paths are:

```
mat:{{1, 0, 0, 0},
      {1, 1, 0, 1},
      {1, 1, 0, 0},
      {0, 1, 1, 1}}
```

All possible paths are:

DDRRR (in red)

DRDRR (in green)



```
1
2 void findPathHelper(int i, int j, vector < vector < int >> & a, int n,
3 vector < string > & ans, string move, vector < vector < int >> & vis) {
4     if (i == n - 1 && j == n - 1) {
5         ans.push_back(move);
6         return;
7     }
8
9     // downward
10    if (i + 1 < n && !vis[i + 1][j] && a[i + 1][j] == 1) {
11        vis[i][j] = 1;
12        findPathHelper(i + 1, j, a, n, ans, move + 'D', vis);
13        vis[i][j] = 0;
14    }
15
16    // left
17    if (j - 1 >= 0 && !vis[i][j - 1] && a[i][j - 1] == 1) {
18        vis[i][j] = 1;
19        findPathHelper(i, j - 1, a, n, ans, move + 'L', vis);
20        vis[i][j] = 0;
21    }
22
23    // right
24    if (j + 1 < n && !vis[i][j + 1] && a[i][j + 1] == 1) {
25        vis[i][j] = 1;
26        findPathHelper(i, j + 1, a, n, ans, move + 'R', vis);
27        vis[i][j] = 0;
28    }
29
30    // upward
31    if (i - 1 >= 0 && !vis[i - 1][j] && a[i - 1][j] == 1) {
32        vis[i][j] = 1;
33        findPathHelper(i - 1, j, a, n, ans, move + 'U', vis);
34        vis[i][j] = 0;
35    }
36
37    return;
38 }
39
40 vector<string> ratMaze(vector<vector<int>> &mat) {
41     // Write your code here.
42     vector<string> ans ;
43     int n = mat.size() ;
44
45     vector < vector < int >> vis(n, vector < int > (n, 0));
46
47     if (mat[0][0] == 1) findPathHelper(0, 0, mat, n, ans, "", vis);
48
49     return ans;
```

```
22
23 // right
24 if (j + 1 < n && !vis[i][j + 1] && a[i][j + 1] == 1) {
25     vis[i][j] = 1;
26     findPathHelper(i, j + 1, a, n, ans, move + 'R', vis);
27     vis[i][j] = 0;
28 }
29
30 // upward
31 if (i - 1 >= 0 && !vis[i - 1][j] && a[i - 1][j] == 1) {
32     vis[i][j] = 1;
33     findPathHelper(i - 1, j, a, n, ans, move + 'U', vis);
34     vis[i][j] = 0;
35 }
36
37 }
38
39 vector<string> ratMaze(vector<vector<int>> &mat) {
40     // Write your code here.
41     vector<string> ans ;
42     int n = mat.size() ;
43
44     vector < vector < int >> vis(n, vector < int > (n, 0));
45
46     if (mat[0][0] == 1) findPathHelper(0, 0, mat, n, ans, "", vis);
47
48     return ans;
49 }
```


Q . N-Queen

DescriptionEditorialSolutions (4.4K)Submissions

51. N-Queens

Hard✔11.1K234☆🔄

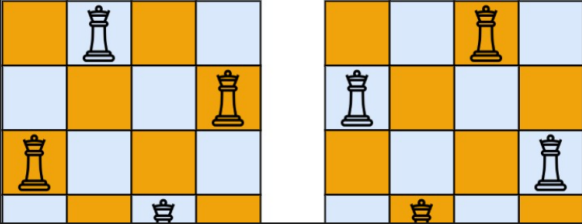
Companies

The **n-queens** puzzle is the problem of placing n queens on an $n \times n$ chessboard such that no two queens attack each other.

Given an integer n , return *all distinct solutions to the n-queens puzzle*. You may return the answer in **any order**.

Each solution contains a distinct board configuration of the n-queens' placement, where `'Q'` and `'.'` both indicate a queen and an empty space, respectively.

Example 1:



C++Auto

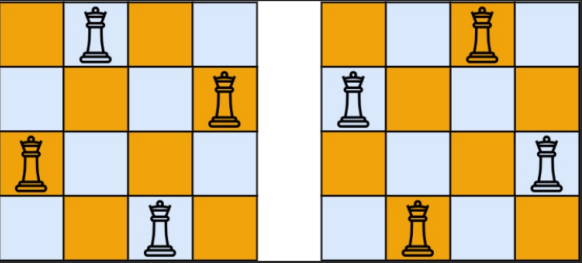
```
1 class Solution {
2     bool issafe(int row, int col, vector<vector<char>>& board, int n){
3         int x = row ;
4         int y = col ;
5         while(x >= 0){
6             if(board[x][y] == 'Q'){
7                 return false ;
8             }
9             x-- ;
10        }
11        x = row ;
12        y = col ;
13        while(x >= 0 && y <= n){
14            if(board[x][y] == 'Q'){
15                return false ;
16            }
17            x-- ;
18            y-- ;
19        }
20        x = row ;
21        y = col ;
22        while(x >= 0 && y < n){
23            if(board[x][y] == 'Q'){
24                return false ;
25            }
26            x-- ;
27            y++ ;
28        }
29        return true ;
30    }
31    void addsolution (vector<vector<char>>& board , vector<vector<string>>& ans , int n){
```

Console ^

↵Run

DescriptionEditorialSolutions (4.4K)Submissions

Example 1:



Input: $n = 4$
Output: `[["Q...", "...Q", "Q...", "...Q"], [".Q.", "Q...", "...Q", ".Q."]]`
Explanation: There exist two distinct solutions to the 4-queens puzzle as shown above

Example 2:

Input: $n = 1$
Output: `[["Q"]]`

Constraints:

- $1 \leq n \leq 9$

C++Auto

```
31 void addsolution (vector<vector<char>>& board , vector<vector<string>>& ans , int n){
32     vector<string> sol ;
33     for(int i = 0 ; i<n ; i++){
34         string temp ;
35         for(int j = 0 ; j<n ; j++){
36             temp.push_back(board[i][j]) ;
37         }
38         sol.push_back(temp) ;
39     }
40     ans.push_back(sol) ;
41 }
42 void solve(int row , vector<vector<char>>& board , vector<vector<string>>& ans , int n){
43     if(row == n ){
44         addsolution(board , ans , n) ;
45         return ;
46     }
47     for(int col = 0 ; col<n ; col++){
48         if(issafe(row , col , board , n)){
49             board[row][col] = 'Q' ;
50             solve(row+1 , board , ans , n);
51             board[row][col] = '.' ;
52         }
53     }
54 }
55 public:
56 vector<vector<string>> solveNQueens(int n) {
57     vector<vector<char>> board (n , vector<char> (n , '.')) ;
58     vector<vector<string>> ans ;
59     int row = 0 ;
60     solve(row , board , ans , n) ;
61     return ans ;
```

Console ^

↵Run

Q. Sudoku Solver

Description

Editorial

Solutions (3.2K)

Submissions

37. Sudoku Solver

Hard

8.6K 221

Companies

Write a program to solve a Sudoku puzzle by filling the empty cells.

A sudoku solution must satisfy **all of the following rules**:

- Each of the digits 1-9 must occur exactly once in each row.
- Each of the digits 1-9 must occur exactly once in each column.
- Each of the digits 1-9 must occur exactly once in each of the 9 3x3 sub-boxes of the grid.

The '.' character indicates empty cells.

Example 1:

5	3		7					
6		1	9	5				
	9	8				6		
8			6					3
4		8		3				1
7			2					6
	6				2	8		
		4	1	9				5
			8			7	9	

Input: board = [["5","3"," ","7"," "," "," "," "," "],["6"," ","1","9","5"," "," "," "," "],[" ","9","8"," "," ","6"," "," "," "],["8"," "," ","6"," "," "," "," ","3"],["4"," ","8"," ","3"," "," "," ","1"],["7"," "," ","2"," "," "," "," ","6"],[" ","6"," "," "," ","2","8"," "," "],[" "," ","4","1","9"," "," "," ","5"],[" "," "," ","8"," "," ","7","9"," "]]

i C++

Auto

```
1 class Solution {
2     bool issafe(int row , int col , vector<vector<char>>& board , char val){
3         for(int i = 0 ; i<9 ; i++){
4             if(board[row][i]==val)
5                 return false ;
6
7             if(board[i][col]==val)
8                 return false ;
9
10            if(board[3*(row/3) + i/3][3*(col/3) + i%3]==val){
11                return false ;
12            }
13        }
14        return true ;
15    }
16    bool solve(vector<vector<char>>& board){
17        for(int r = 0 ; r<9 ; r++){
18            for(int c = 0 ; c<9 ; c++){
19                if(board[r][c]=='.'){
20                    for(int val = 1 ; val<=9 ; val++){
21                        if(issafe(r , c , board , val + '0')){
22                            board[r][c] = '0' + val ;
23                            if(solve(board)){
24                                return true ;
25                            }else{
26                                board[r][c] = '.' ;
27                            }
28                        }
29                    }
30                    return false ;
31                }
32            }
33        }
34        return true ;
35    }
36 }
37 public:
38     void solveSudoku(vector<vector<char>>& board) {
39         solve(board) ;
40     }
41 };
```

End

DP

Q. Climbing Stairs

DescriptionEditorialSolutions (12.6K)Submissions

70. Climbing Stairs

Easy✔19.5K636☆🔄

Companies

You are climbing a staircase. It takes n steps to reach the top.

Each time you can either climb 1 or 2 steps. In how many distinct ways can you climb to the top?

Example 1:

Input: $n = 2$
Output: 2
Explanation: There are two ways to climb to the top.
1. 1 step + 1 step
2. 2 steps

Example 2:

Input: $n = 3$
Output: 3
Explanation: There are three ways to climb to the top.
1. 1 step + 1 step + 1 step
2. 1 step + 2 steps
3. 2 steps + 1 step

C++Auto

```
1 class Solution {
2     int count(int n, int step, vector<int>& dp){
3         if(n<step){
4             return 0;
5         }
6         if(n == step){
7             return 1;
8         }
9         if(dp[step] != -1){
10            return dp[step];
11        }
12        dp[step] = count(n, step + 1, dp) + count(n, step + 2, dp);
13        return dp[step];
14    }
15 public:
16     int climbStairs(int n) {
17         vector<int> dp (n+1, -1);
18         return count(n, 0, dp);
19     }
20 };
```

Console ^

Q. Min Cost Climbing Stairs

DescriptionEditorialSolutions (4.7K)Submissions

746. Min Cost Climbing Stairs

Easy✔10K1.5K☆🔄

Companies

You are given an integer array `cost` where `cost[i]` is the cost of i^{th} step on a staircase. Once you pay the cost, you can either climb one or two steps.

You can either start from the step with index 0, or the step with index 1.

Return the minimum cost to reach the top of the floor.

Example 1:

Input: `cost = [10,15,20]`
Output: 15
Explanation: You will start at index 1.
- Pay 15 and climb two steps to reach the top.
The total cost is 15.

Example 2:

Input: `cost = [1,100,1,1,1,100,1,1,100,1]`
Output: 6
Explanation: You will start at index 0.
- Pay 1 and climb two steps to reach index 2.

C++Auto

```
1 class Solution {
2     int solve (vector<int>& cost, int step, vector<int>& dp){
3         if(step == 0){
4             return cost[0];
5         }
6         if(step == 1){
7             return cost[1];
8         }
9
10        if(dp[step] != -1){
11            return dp[step];
12        }
13
14        dp[step] = cost[step] + min(solve(cost, step-1, dp), solve(cost, step-2, dp));
15        return dp[step];
16    }
17
18 public:
19     int minCostClimbingStairs(vector<int>& cost) {
20         int n = cost.size();
21         vector<int> dp (n+1, -1);
22         int ans = min(solve(cost, n-1, dp), solve(cost, n-2, dp));
23         return ans;
24     }
25 };
```

Console ^

Q.

End

Linked List

Q. Reverse Linked List

Problem List

DescriptionEditorialSolutions (10.3K)Submissions

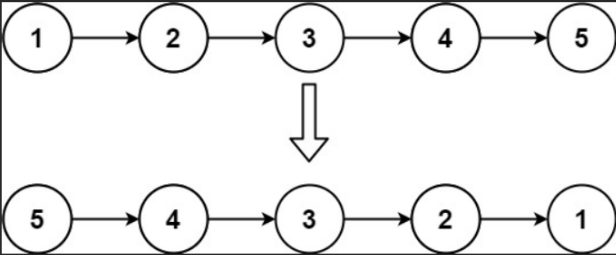
206. Reverse Linked List

Easy18.2K337

Companies

Given the `head` of a singly linked list, reverse the list, and return *the reversed list*.

Example 1:



Input: head = [1,2,3,4,5]
Output: [5,4,3,2,1]

C++Auto

```
1 int val;  
2  
3 *  
4 ListNode* next;  
5 *  
6 ListNode() : val(0), next(nullptr) {}  
7 *  
8 ListNode(int x) : val(x), next(nullptr) {}  
9 *  
10 *  
11 *  
12 *  
13 *  
14 *  
15 *  
16 *  
17 *  
18 *  
19 *  
20 *  
21 *  
22 *  
23 *  
24 *  
25 *  
26 *  
27 *  
28 *  
29 *  
30 *  
31 *  
32 *  
33 *  
34 *  
35 *  
36 *  
37 *  
38 *  
39 *  
40 *  
41 *  
42 *  
43 *  
44 *  
45 *  
46 *  
47 *  
48 *  
49 *  
50 *  
51 *  
52 *  
53 *  
54 *  
55 *  
56 *  
57 *  
58 *  
59 *  
60 *  
61 *  
62 *  
63 *  
64 *  
65 *  
66 *  
67 *  
68 *  
69 *  
70 *  
71 *  
72 *  
73 *  
74 *  
75 *  
76 *  
77 *  
78 *  
79 *  
80 *  
81 *  
82 *  
83 *  
84 *  
85 *  
86 *  
87 *  
88 *  
89 *  
90 *  
91 *  
92 *  
93 *  
94 *  
95 *  
96 *  
97 *  
98 *  
99 *  
100 *  
101 *  
102 *  
103 *  
104 *  
105 *  
106 *  
107 *  
108 *  
109 *  
110 *  
111 *  
112 *  
113 *  
114 *  
115 *  
116 *  
117 *  
118 *  
119 *  
120 *  
121 *  
122 *  
123 *  
124 *  
125 *  
126 *  
127 *  
128 *  
129 *  
130 *  
131 *  
132 *  
133 *  
134 *  
135 *  
136 *  
137 *  
138 *  
139 *  
140 *  
141 *  
142 *  
143 *  
144 *  
145 *  
146 *  
147 *  
148 *  
149 *  
150 *  
151 *  
152 *  
153 *  
154 *  
155 *  
156 *  
157 *  
158 *  
159 *  
160 *  
161 *  
162 *  
163 *  
164 *  
165 *  
166 *  
167 *  
168 *  
169 *  
170 *  
171 *  
172 *  
173 *  
174 *  
175 *  
176 *  
177 *  
178 *  
179 *  
180 *  
181 *  
182 *  
183 *  
184 *  
185 *  
186 *  
187 *  
188 *  
189 *  
190 *  
191 *  
192 *  
193 *  
194 *  
195 *  
196 *  
197 *  
198 *  
199 *  
200 *  
201 *  
202 *  
203 *  
204 *  
205 *  
206 *  
207 *  
208 *  
209 *  
210 *  
211 *  
212 *  
213 *  
214 *  
215 *  
216 *  
217 *  
218 *  
219 *  
220 *  
221 *  
222 *  
223 *  
224 *  
225 *  
226 *  
227 *  
228 *  
229 *  
230 *  
231 *  
232 *  
233 *  
234 *  
235 *  
236 *  
237 *  
238 *  
239 *  
240 *  
241 *  
242 *  
243 *  
244 *  
245 *  
246 *  
247 *  
248 *  
249 *  
250 *  
251 *  
252 *  
253 *  
254 *  
255 *  
256 *  
257 *  
258 *  
259 *  
260 *  
261 *  
262 *  
263 *  
264 *  
265 *  
266 *  
267 *  
268 *  
269 *  
270 *  
271 *  
272 *  
273 *  
274 *  
275 *  
276 *  
277 *  
278 *  
279 *  
280 *  
281 *  
282 *  
283 *  
284 *  
285 *  
286 *  
287 *  
288 *  
289 *  
290 *  
291 *  
292 *  
293 *  
294 *  
295 *  
296 *  
297 *  
298 *  
299 *  
300 *  
301 *  
302 *  
303 *  
304 *  
305 *  
306 *  
307 *  
308 *  
309 *  
310 *  
311 *  
312 *  
313 *  
314 *  
315 *  
316 *  
317 *  
318 *  
319 *  
320 *  
321 *  
322 *  
323 *  
324 *  
325 *  
326 *  
327 *  
328 *  
329 *  
330 *  
331 *  
332 *  
333 *  
334 *  
335 *  
336 *  
337 *  
338 *  
339 *  
340 *  
341 *  
342 *  
343 *  
344 *  
345 *  
346 *  
347 *  
348 *  
349 *  
350 *  
351 *  
352 *  
353 *  
354 *  
355 *  
356 *  
357 *  
358 *  
359 *  
360 *  
361 *  
362 *  
363 *  
364 *  
365 *  
366 *  
367 *  
368 *  
369 *  
370 *  
371 *  
372 *  
373 *  
374 *  
375 *  
376 *  
377 *  
378 *  
379 *  
380 *  
381 *  
382 *  
383 *  
384 *  
385 *  
386 *  
387 *  
388 *  
389 *  
390 *  
391 *  
392 *  
393 *  
394 *  
395 *  
396 *  
397 *  
398 *  
399 *  
400 *  
401 *  
402 *  
403 *  
404 *  
405 *  
406 *  
407 *  
408 *  
409 *  
410 *  
411 *  
412 *  
413 *  
414 *  
415 *  
416 *  
417 *  
418 *  
419 *  
420 *  
421 *  
422 *  
423 *  
424 *  
425 *  
426 *  
427 *  
428 *  
429 *  
430 *  
431 *  
432 *  
433 *  
434 *  
435 *  
436 *  
437 *  
438 *  
439 *  
440 *  
441 *  
442 *  
443 *  
444 *  
445 *  
446 *  
447 *  
448 *  
449 *  
450 *  
451 *  
452 *  
453 *  
454 *  
455 *  
456 *  
457 *  
458 *  
459 *  
460 *  
461 *  
462 *  
463 *  
464 *  
465 *  
466 *  
467 *  
468 *  
469 *  
470 *  
471 *  
472 *  
473 *  
474 *  
475 *  
476 *  
477 *  
478 *  
479 *  
480 *  
481 *  
482 *  
483 *  
484 *  
485 *  
486 *  
487 *  
488 *  
489 *  
490 *  
491 *  
492 *  
493 *  
494 *  
495 *  
496 *  
497 *  
498 *  
499 *  
500 *  
501 *  
502 *  
503 *  
504 *  
505 *  
506 *  
507 *  
508 *  
509 *  
510 *  
511 *  
512 *  
513 *  
514 *  
515 *  
516 *  
517 *  
518 *  
519 *  
520 *  
521 *  
522 *  
523 *  
524 *  
525 *  
526 *  
527 *  
528 *  
529 *  
530 *  
531 *  
532 *  
533 *  
534 *  
535 *  
536 *  
537 *  
538 *  
539 *  
540 *  
541 *  
542 *  
543 *  
544 *  
545 *  
546 *  
547 *  
548 *  
549 *  
550 *  
551 *  
552 *  
553 *  
554 *  
555 *  
556 *  
557 *  
558 *  
559 *  
560 *  
561 *  
562 *  
563 *  
564 *  
565 *  
566 *  
567 *  
568 *  
569 *  
570 *  
571 *  
572 *  
573 *  
574 *  
575 *  
576 *  
577 *  
578 *  
579 *  
580 *  
581 *  
582 *  
583 *  
584 *  
585 *  
586 *  
587 *  
588 *  
589 *  
590 *  
591 *  
592 *  
593 *  
594 *  
595 *  
596 *  
597 *  
598 *  
599 *  
600 *  
601 *  
602 *  
603 *  
604 *  
605 *  
606 *  
607 *  
608 *  
609 *  
610 *  
611 *  
612 *  
613 *  
614 *  
615 *  
616 *  
617 *  
618 *  
619 *  
620 *  
621 *  
622 *  
623 *  
624 *  
625 *  
626 *  
627 *  
628 *  
629 *  
630 *  
631 *  
632 *  
633 *  
634 *  
635 *  
636 *  
637 *  
638 *  
639 *  
640 *  
641 *  
642 *  
643 *  
644 *  
645 *  
646 *  
647 *  
648 *  
649 *  
650 *  
651 *  
652 *  
653 *  
654 *  
655 *  
656 *  
657 *  
658 *  
659 *  
660 *  
661 *  
662 *  
663 *  
664 *  
665 *  
666 *  
667 *  
668 *  
669 *  
670 *  
671 *  
672 *  
673 *  
674 *  
675 *  
676 *  
677 *  
678 *  
679 *  
680 *  
681 *  
682 *  
683 *  
684 *  
685 *  
686 *  
687 *  
688 *  
689 *  
690 *  
691 *  
692 *  
693 *  
694 *  
695 *  
696 *  
697 *  
698 *  
699 *  
700 *  
701 *  
702 *  
703 *  
704 *  
705 *  
706 *  
707 *  
708 *  
709 *  
710 *  
711 *  
712 *  
713 *  
714 *  
715 *  
716 *  
717 *  
718 *  
719 *  
720 *  
721 *  
722 *  
723 *  
724 *  
725 *  
726 *  
727 *  
728 *  
729 *  
730 *  
731 *  
732 *  
733 *  
734 *  
735 *  
736 *  
737 *  
738 *  
739 *  
740 *  
741 *  
742 *  
743 *  
744 *  
745 *  
746 *  
747 *  
748 *  
749 *  
750 *  
751 *  
752 *  
753 *  
754 *  
755 *  
756 *  
757 *  
758 *  
759 *  
760 *  
761 *  
762 *  
763 *  
764 *  
765 *  
766 *  
767 *  
768 *  
769 *  
770 *  
771 *  
772 *  
773 *  
774 *  
775 *  
776 *  
777 *  
778 *  
779 *  
780 *  
781 *  
782 *  
783 *  
784 *  
785 *  
786 *  
787 *  
788 *  
789 *  
790 *  
791 *  
792 *  
793 *  
794 *  
795 *  
796 *  
797 *  
798 *  
799 *  
800 *  
801 *  
802 *  
803 *  
804 *  
805 *  
806 *  
807 *  
808 *  
809 *  
810 *  
811 *  
812 *  
813 *  
814 *  
815 *  
816 *  
817 *  
818 *  
819 *  
820 *  
821 *  
822 *  
823 *  
824 *  
825 *  
826 *  
827 *  
828 *  
829 *  
830 *  
831 *  
832 *  
833 *  
834 *  
835 *  
836 *  
837 *  
838 *  
839 *  
840 *  
841 *  
842 *  
843 *  
844 *  
845 *  
846 *  
847 *  
848 *  
849 *  
850 *  
851 *  
852 *  
853 *  
854 *  
855 *  
856 *  
857 *  
858 *  
859 *  
860 *  
861 *  
862 *  
863 *  
864 *  
865 *  
866 *  
867 *  
868 *  
869 *  
870 *  
871 *  
872 *  
873 *  
874 *  
875 *  
876 *  
877 *  
878 *  
879 *  
880 *  
881 *  
882 *  
883 *  
884 *  
885 *  
886 *  
887 *  
888 *  
889 *  
890 *  
891 *  
892 *  
893 *  
894 *  
895 *  
896 *  
897 *  
898 *  
899 *  
900 *  
901 *  
902 *  
903 *  
904 *  
905 *  
906 *  
907 *  
908 *  
909 *  
910 *  
911 *  
912 *  
913 *  
914 *  
915 *  
916 *  
917 *  
918 *  
919 *  
920 *  
921 *  
922 *  
923 *  
924 *  
925 *  
926 *  
927 *  
928 *  
929 *  
930 *  
931 *  
932 *  
933 *  
934 *  
935 *  
936 *  
937 *  
938 *  
939 *  
940 *  
941 *  
942 *  
943 *  
944 *  
945 *  
946 *  
947 *  
948 *  
949 *  
950 *  
951 *  
952 *  
953 *  
954 *  
955 *  
956 *  
957 *  
958 *  
959 *  
960 *  
961 *  
962 *  
963 *  
964 *  
965 *  
966 *  
967 *  
968 *  
969 *  
970 *  
971 *  
972 *  
973 *  
974 *  
975 *  
976 *  
977 *  
978 *  
979 *  
980 *  
981 *  
982 *  
983 *  
984 *  
985 *  
986 *  
987 *  
988 *  
989 *  
990 *  
991 *  
992 *  
993 *  
994 *  
995 *  
996 *  
997 *  
998 *  
999 *  
1000 *  
1001 *  
1002 *  
1003 *  
1004 *  
1005 *  
1006 *  
1007 *  
1008 *  
1009 *  
1010 *  
1011 *  
1012 *  
1013 *  
1014 *  
1015 *  
1016 *  
1017 *  
1018 *  
1019 *  
1020 *  
1021 *  
1022 *  
1023 *  
1024 *  
1025 *  
1026 *  
1027 *  
1028 *  
1029 *  
1030 *  
1031 *  
1032 *  
1033 *  
1034 *  
1035 *  
1036 *  
1037 *  
1038 *  
1039 *  
1040 *  
1041 *  
1042 *  
1043 *  
1044 *  
1045 *  
1046 *  
1047 *  
1048 *  
1049 *  
1050 *  
1051 *  
1052 *  
1053 *  
1054 *  
1055 *  
1056 *  
1057 *  
1058 *  
1059 *  
1060 *  
1061 *  
1062 *  
1063 *  
1064 *  
1065 *  
1066 *  
1067 *  
1068 *  
1069 *  
1070 *  
1071 *  
1072 *  
1073 *  
1074 *  
1075 *  
1076 *  
1077 *  
1078 *  
1079 *  
1080 *  
1081 *  
1082 *  
1083 *  
1084 *  
1085 *  
1086 *  
1087 *  
1088 *  
1089 *  
1090 *  
1091 *  
1092 *  
1093 *  
1094 *  
1095 *  
1096 *  
1097 *  
1098 *  
1099 *  
1100 *  
1101 *  
1102 *  
1103 *  
1104 *  
1105 *  
1106 *  
1107 *  
1108 *  
1109 *  
1110 *  
1111 *  
1112 *  
1113 *  
1114 *  
1115 *  
1116 *  
1117 *  
1118 *  
1119 *  
1120 *  
1121 *  
1122 *  
1123 *  
1124 *  
1125 *  
1126 *  
1127 *  
1128 *  
1129 *  
1130 *  
1131 *  
1132 *  
1133 *  
1134 *  
1135 *  
1136 *  
1137 *  
1138 *  
1139 *  
1140 *  
1141 *  
1142 *  
1143 *  
1144 *  
1145 *  
1146 *  
1147 *  
1148 *  
1149 *  
1150 *  
1151 *  
1152 *  
1153 *  
1154 *  
1155 *  
1156 *  
1157 *  
1158 *  
1159 *  
1160 *  
1161 *  
1162 *  
1163 *  
1164 *  
1165 *  
1166 *  
1167 *  
1168 *  
1169 *  
1170 *  
1171 *  
1172 *  
1173 *  
1174 *  
1175 *  
1176 *  
1177 *  
1178 *  
1179 *  
1180 *  
1181 *  
1182 *  
1183 *  
1184 *  
1185 *  
1186 *  
1187 *  
1188 *  
1189 *  
1190 *  
1191 *  
1192 *  
1193 *  
1194 *  
1195 *  
1196 *  
1197 *  
1198 *  
1199 *  
1200 *  
1201 *  
1202 *  
1203 *  
1204 *  
1205 *  
1206 *  
1207 *  
1208 *  
1209 *  
1210 *  
1211 *  
1212 *  
1213 *  
1214 *  
1215 *  
1216 *  
1217 *  
1218 *  
1219 *  
1220 *  
1221 *  
1222 *  
1223 *  
1224 *  
1225 *  
1226 *  
1227 *  
1228 *  
1229 *  
1230 *  
1231 *  
1232 *  
1233 *  
1234 *  
1235 *  
1236 *  
1237 *  
1238 *  
1239 *  
1240 *  
1241 *  
1242 *  
1243 *  
1244 *  
1245 *  
1246 *  
1247 *  
1248 *  
1249 *  
1250 *  
1251 *  
1252 *  
1253 *  
1254 *  
1255 *  
1256 *  
1257 *  
1258 *  
1259 *  
1260 *  
1261 *  
1262 *  
1263 *  
1264 *  
1265 *  
1266 *  
1267 *  
1268 *  
1269 *  
1270 *  
1271 *  
1272 *  
1273 *  
1274 *  
1275 *  
1276 *  
1277 *  
1278 *  
1279 *  
1280 *  
1281 *  
1282 *  
1283 *  
1284 *  
1285 *  
1286 *  
1287 *  
1288 *  
1289 *  
1290 *  
1291 *  
1292 *  
1293 *  
1294 *  
1295 *  
1296 *  
1297 *  
1298 *  
1299 *  
1300 *  
1301 *  
1302 *  
1303 *  
1304 *  
1305 *  
1306 *  
1307 *  
1308 *  
1309 *  
1310 *  
1311 *  
1312 *  
1313 *  
1314 *  
1315 *  
1316 *  
1317 *  
1318 *  
1319 *  
1320 *  
1321 *  
1322 *  
1323 *  
1324 *  
1325 *  
1326 *  
1327 *  
1328 *  
1329 *  
1330 *  
1331 *  
1332 *  
1333 *  
1334 *  
1335 *  
1336 *  
1337 *  
1338 *  
1339 *  
1340 *  
1341 *  
1342 *  
1343 *  
1344 *  
1345 *  
1346 *  
1347 *  
1348 *  
1349 *  
1350 *  
1351 *  
1352 *  
1353 *  
1354 *  
1355 *  
1356 *  
1357 *  
1358 *  
1359 *  
1360 *  
1361 *  
1362 *  
1363 *  
1364 *  
1365 *  
1366 *  
1367 *  
1368 *  
1369 *  
1370 *  
1371 *  
1372 *  
1373 *  
1374 *  
1375 *  
1376 *  
1377 *  
1378 *  
1379 *  
1380 *  
1381 *  
1382 *  
1383 *  
1384 *  
1385 *  
1386 *  
1387 *  
1388 *  
1389 *  
1390 *  
1391 *  
1392 *  
1393 *  
1394 *  
1395 *  
1396 *  
1397 *  
1398 *  
1399 *  
1400 *  
1401 *  
1402 *  
1403 *  
1404 *  
1405 *  
1406 *  
1407 *  
1408 *  
1409 *  
1410 *  
1411 *  
1412 *  
1413 *  
1414 *  
1415 *  
1416 *  
1417 *  
1418 *  
1419 *  
1420 *  
1421 *  
1422 *  
1423 *  
1424 *  
1425 *  
1426 *  
1427 *  
1428 *  
1429 *  
1430 *  
1431 *  
1432 *  
1433 *  
1434 *  
1435 *  
1436 *  
1437 *  
1438 *  
1439 *  
1440 *  
1441 *  
1442 *  
1443 *  
1444 *  
1445 *  
1446 *  
1447 *  
1448 *  
1449 *  
1450 *  
1451 *  
1452 *  
1453 *  
1454 *  
1455 *  
1456 *  
1457 *  
1458 *  
1459 *  
1460 *  
1461 *  
1462 *  
1463 *  
1464 *  
1465 *  
1466 *  
1467 *  
1468 *  
1469 *  
1470 *  
1471 *  
1472 *  
1473 *  
1474 *  
1475 *  
1476 *  
1477 *  
1478 *  
1479 *  
1480 *  
1481 *  
1482 *  
1483 *  
1484 *  
1485 *  
1486 *  
1487 *  
1488 *  
1489 *  
1490 *  
1491 *  
1492 *  
1493 *  
1494 *  
1495 *  
1496 *  
1497 *  
1498 *  
1499 *  
1500 *  
1501 *  
1502 *  
1503 *  
1504 *  
1505 *  
1506 *  
1507 *  
1508 *  
1509 *  
1510 *  
1511 *  
1512 *  
1513 *  
1514 *  
1515 *  
1516 *  
1517 *  
1518 *  
1519 *  
1520 *  
1521 *  
1522 *  
1523 *  
1524 *  
1525 *  
1526 *  
1527 *  
1528 *  
1529 *  
1530 *  
1531 *  
1532 *  
1533 *  
1534 *  
1535 *  
1536 *  
1537 *  
1538 *  
1539 *  
1540 *  
1541 *  
1542 *  
1543 *  
1544 *  
1545 *  
1546 *  
1547 *  
1548 *  
1549 *  
1550 *  
1551 *  
1552 *  
1553 *  
1554 *  
1555 *  
1556 *  
1557 *  
1558 *  
1559 *  
1560 *  
1561 *  
1562 *  
1563 *  
1564 *  
1565 *  
1566 *  
1567 *  
1568 *  
1569 *  
1570 *  
1571 *  
1572 *  
1573 *  
1574 *  
1575 *  
1576 *  
1577 *  
1578 *  
1579 *  
1580 *  
1581 *  
1582 *  
1583 *  
1584 *  
1585 *  
1586 *  
1587 *  
1588 *  
1589 *  
1590 *  
1591 *  
1592 *  
1593 *  
1594 *  
1595 *  
1596 *  
1597 *  
1598 *  
1599 *  
1600 *  
1601 *  
1602 *  
1603 *  
1604 *  
1605 *  
1606 *  
1607 *  
1608 *  
1609 *  
1610 *  
1611 *  
1612 *  
1613 *  
1614 *  
1615 *  
1616 *  
1617 *  
1618 *  
1619 *  
1620 *  
1621 *  
1622 *  
1623 *  
1624 *  
1625 *  
1626 *  
1627 *  
1628 *  
1629 *  
1630 *  
1631 *  
1632 *  
1633 *  
1634 *  
1635 *  
1636 *  
1637 *  
1638 *  
1639 *  
16
```

Q. Remove Duplicate From sorted List

DescriptionDiscussion (5)Solutions (5.1K)Submissions

83. Remove Duplicates from Sorted List

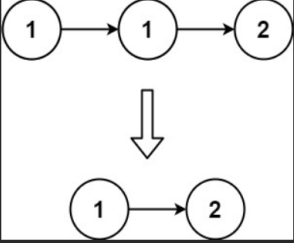
Easy

5.9K 210

Companies

Given the `head` of a sorted linked list, *delete all duplicates such that each element appears only once*. Return the linked list *sorted* as well.

Example 1:



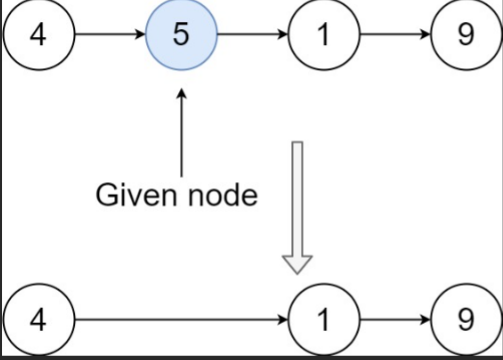
Input: head = [1,1,2]
Output: [1,2]

```
3 struct ListNode {
4     int val;
5     ListNode *next;
6     ListNode() : val(0), next(nullptr) {}
7     ListNode(int x) : val(x), next(nullptr) {}
8     ListNode(int x, ListNode *next) : val(x), next(next) {}
9 };
10
11 class Solution {
12 public:
13     ListNode* deleteDuplicates(ListNode* head) {
14         ListNode * temp = head ;
15
16         if(temp == NULL)
17             return head ;
18         while(temp->next != NULL){
19             if(temp->val == temp->next->val ){
20                 ListNode * a = temp->next->next ;
21                 // ListNode * l =a->next ;
22                 delete temp->next ;
23                 temp->next = a ;
24             }else{
25                 temp = temp->next ;
26             }
27         }
28         return head ;
29     }
30 };
31
32
33
34 ;;
```

Q. Delete Node in a Linked List

DescriptionDiscussion (65)Solutions (2.8K)Submissions

Example 1:



Input: head = [4,5,1,9], node = 5
Output: [4,1,9]
Explanation: You are given the second node with value 5, the linked list should become 4 -> 1 -> 9 after calling your function.

```
1 /**
2  * Definition for singly-linked list.
3  * struct ListNode {
4  *     int val;
5  *     ListNode *next;
6  *     ListNode(int x) : val(x), next(NULL) {}
7  * };
8  */
9 class Solution {
10 public:
11     void deleteNode(ListNode* node) {
12
13         node->val = node->next->val;
14         node->next = node->next->next;
15
16
17     }
18 };
19 ;;
```

Q . Remove Nth Node From End of List

```
class Solution {
    int length(ListNode* head){
        int i=1;
        while(head->next!=NULL){
            i++;
            head=head->next;
        }
        return i;
    }
public:
    ListNode* removeNthFromEnd(ListNode* head, int n) {
        if(head==NULL)
            return NULL;
        if(head->next==NULL&& n==1)
            return NULL;
        int l=length(head);
        if(l==n)
        {
            head=head->next;
            return head;
        }
        l=l-n;
        int i=1;
        ListNode* curr=head;
        while(i<l){
            curr=curr->next;
            i++;
        }
        ListNode* temp=curr->next;
        curr->next=curr->next->next;
        temp->next=NULL;
        delete temp;
        return head;
    }
}
```

};

Q . Middle of The Linked List

876. Middle of the Linked List


Easy 7.1K 191

Companies

Given the `head` of a singly linked list, return *the middle node of the linked list*.


If there are two middle nodes, return **the second middle node**.

Example 1:



Input: head = [1,2,3,4,5]
Output: [3,4,5]
Explanation: The middle node of the list is node 3.

Example 2:



Input: head = [1,2,3,4,5,6]

```
1 /**
2  * Definition for singly-linked list.
3  * struct ListNode {
4  *     int val;
5  *     ListNode *next;
6  *     ListNode() : val(0), next(nullptr) {}
7  *     ListNode(int x) : val(x), next(nullptr) {}
8  *     ListNode(int x, ListNode *next) : val(x), next(next) {}
9  * };
10 */
11 class Solution {
12 public:
13     ListNode* middleNode(ListNode* head) {
14         ListNode* slow = head ;
15         ListNode* fast = head->next ;
16
17         while(fast != NULL && fast->next != NULL){
18             slow = slow->next ;
19             fast = fast->next->next ;
20         }
21         if (fast == NULL){
22             return slow ;
23         }
24
25         return slow->next;
26
27     }
28 };
```

Console

Q . Linked List Cycle

141. Linked List Cycle

Easy 12.4K 1K

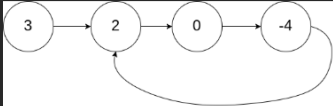
Companies

Given `head`, the head of a linked list, determine if the linked list has a cycle in it.

There is a cycle in a linked list if there is some node in the list that can be reached again by continuously following the `next` pointer. Internally, `pos` is used to denote the index of the node that tail's `next` pointer is connected to. **Note that `pos` is not passed as a parameter.**

Return `true` if there is a cycle in the linked list. Otherwise, return `false`.

Example 1:



Input: `head = [3,2,0,-4]`, `pos = 1`

Output: `true`

Explanation: There is a cycle in the linked list, where the tail connects to the 1st node (0-indexed).

C++ Auto

```

1  /**
2  * Definition for singly-linked list.
3  * struct ListNode {
4  *     int val;
5  *     ListNode *next;
6  *     ListNode(int x) : val(x), next(NULL) {}
7  * };
8  */
9  class Solution {
10 public:
11     bool hasCycle(ListNode *head) {
12         ListNode * temp = head ;
13         map<ListNode* , int> m ;
14         int i = 1 ;
15         while(temp != NULL){
16             if(m[temp] == 0){
17                 m[temp] = i++ ;
18                 temp = temp->next ;
19             }else{
20                 return true ;
21             }
22         }
23         return false ;
24     }
25 };

```

Console

Stacks

Queue

Binary Tree

Binary Search Tree

HashMap

Heap And Priority Queue

Tries

Graphs

Greedy

Algorithms & Other