1

10 August 2023 11:42 PM

Difficulty: **Category:** **Successful Submissions:** 78,360+

Three Number Sum

Write a function that takes in a non-empty array of distinct integers and an integer representing a target sum. The function should find all triplets in the array that sum up to the target sum and return a two-dimensional array of all these triplets. The numbers in each triplet should be ordered in ascending order, and the triplets themselves should be ordered in ascending order with respect to the numbers they hold.

If no three numbers sum up to the target sum, the function should return an empty array.

Sample Input

```
array = [12, 3, 1, 2, -6, 5, -8, 6]
targetSum = 0
```

Sample Output

```
[[-8, 2, 6], [-8, 3, 5], [-6, 1, 5]]
```

```
//APPROACH :
// so first we initialize a result 2D vector.
// then we sort the given vector of integers.
// we are running the for loop size times of integers vector.
// we are initializing left, right and sum.
// our target is left + right + array[i] = target or l + r =target - arra[i]
// if for i satisfies this eq we push the triplets into result array and increment and
// decrement left and right resp. else we check if l+r is less than then we move l forward
// else if target + array[i] > l+r then decrement r.
// this keeps on working till left < right.
// at the end we return the desired 2D vector.
```

Solution 1 | Solution 2 | Solution 3

```
#include <vector>
using namespace std;

vector<vector<int>> threeNumberSum(vector<int> array, int targetSum) {
  // Write your code here.


  vector<vector<int>> result;
  int n = array.size();

  sort(array.begin(), array.end());

  for(int i = 0; i<n; i++){
    int left = i+1, right = n-1;
    while(left<right){
      int sum = array[left]+array[right];
    if(sum == targetSum - array[i]){
      result.push_back({array[i], array[left], array[right]});
      left++; right--;
    }
    else if(sum<targetSum-array[i])
        left ++;
    else
        right --;
  }
    }

  return result;
}
```

2

10 August 2023 11:51 PM

Difficulty: **Category:** **Successful Submissions:** 71,457+

Smallest Difference

Write a function that takes in two non-empty arrays of integers, finds the pair of numbers (one from each array) whose absolute difference is closest to zero, and returns an array containing these two numbers, with the number from the first array in the first position.

Note that the absolute difference of two integers is the distance between them on the real number line. For example, the absolute difference of -5 and 5 is 10, and the absolute difference of -5 and -4 is 1.

You can assume that there will only be one pair of numbers with the smallest difference.

Sample Input

```
arrayOne = [-1, 5, 10, 20, 28, 3]
arrayTwo = [26, 134, 135, 15, 17]
```

Sample Output

```
[28, 26]
```

Solution 1 **Solution 2** **Solution 3**

```cpp
#include <vector>
using namespace std;

vector<int> smallestDifference(vector<int> arrayOne, vector<int> arrayTwo) {
  // Write your code here.
  vector<int> result;
  int n = arrayOne.size();
  int m = arrayTwo.size();
  int diff ;
  int mainDiff=INT_MAX;
  int one,two;
  for(int i =0 ; i<n; i++){
    for(int j =0 ; j<m; j++){
        diff = abs(arrayOne[i] - arrayTwo[j]);

        if(diff < mainDiff){
          mainDiff = diff;
          one =arrayOne[i]; two= arrayTwo[j];
        }

    }
  }
  cout << one <<endl;
  cout << two <<endl;
  result.push_back(one);
  result.push_back(two);

  return result;
}

```

3

10 August 2023 11:51 PM

Difficulty: **Category:** **Successful Submissions:** 70,666+

Move Element To End

You're given an array of integers and an integer. Write a function that moves all instances of that integer in the array to the end of the array and returns the array.

The function should perform this in place (i.e., it should mutate the input array) and doesn't need to maintain the order of the other integers.

Sample Input

```
array = [2, 1, 2, 2, 2, 3, 4, 2]
toMove = 2
```

Sample Output

```
[1, 3, 4, 2, 2, 2, 2, 2] // the numbers 1, 3, and 4 could be ordered dif
```

Solution 1 Solution 2 Solution 3

```cpp
#include <vector>

using namespace std;
//APPROACH :
// first we check if the ith value is equal to toMove int or not. if not increase i
// else proceed with if part i.e. check a[i] with a[l] if they are not same, just swap.
// else don't increase i as ith is still we want to move at last. so just decrement
// l.
// run this code till i is less than or equal to l ptr.

vector<int> moveElementToEnd(vector<int> array, int toMove) {
  // Write your code here.
  int n = array.size();
  int i =0, l=n-1;

  while(i<=l){
    if(array[i] == toMove){
      if(array[i]!= array[l]){
        swap(array[i], array[l]);
        i++;l--;
      }
      else
        l--;
    }
    else
        i++;
  }

  vector<int>::iterator it;
  for(it=array.begin(); it!=array.end(); it++){
    cout << *it <<" ";
  }
  return array;
}
```

4

10 August 2023 11:51 PM

Difficulty: **Category:** **Successful Submissions:** 63,227+

Monotonic Array

Write a function that takes in an array of integers and returns a boolean representing whether the array is monotonic.

An array is said to be monotonic if its elements, from left to right, are entirely non-increasing or entirely non-decreasing.

Non-increasing elements aren't necessarily exclusively decreasing; they simply don't increase. Similarly, non-decreasing elements aren't necessarily exclusively increasing; they simply don't decrease.

Note that empty arrays and arrays of one element are monotonic.

Sample Input

```
array = [-1, -5, -10, -1100, -1100, -1101, -1102, -9001]
```

Sample Output

```
true
```

Solution 1 **Solution 2** **Solution 3**

```cpp
using namespace std;

bool isMonotonic(vector<int> array) {
  // Write your code here.
  int n = array.size();

    if(array.size() ==1 || array.size()==0)
      return true;

    if(array[0]<array[n-1]){
      cout << "inside if" <<endl;
      for(int i =0 ; i<n;){
        cout <<"value i= " <<i <<endl;
        if(array[i+1]<array[i])
            break;
        i++;
        if(i==n-1)
            return true;
      }

    }
    else {
      cout << "inside else" <<endl;
      for(int i =0 ; i<n;){
        cout <<"value i= " <<i <<endl;
        if(array[i+1]>array[i])
            break;
        i++;
        if(i==n-1)
            return true;
      }
    }


  return false;
}

```

5

10 August 2023 11:51 PM

Difficulty: **Category:** **Successful Submissions:** 45,971+

Spiral Traverse

Write a function that takes in an n x m two-dimensional array (that can be square-shaped when n == m) and returns a one-dimensional array of all the array's elements in spiral order.

Spiral order starts at the top left corner of the two-dimensional array, goes to the right, and proceeds in a spiral pattern all the way until every element has been visited.

Sample Input

```
array = [
  [1,   2,  3, 4],
  [12, 13, 14, 5],
  [11, 16, 15, 6],
  [10,  9,  8, 7],
]
```

Sample Output

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16]
```

Solution 1 Solution 2 Solution 3

```cpp
using namespace std;

vector<int> spiralTraverse(vector<vector<int>> array) {
  // Write your code here.
  //size of 2d array
  int rowSize = array.size();  //this will give the total rows.
  int colSize = array[0].size();  // no. of col.
  int arraySize = rowSize * colSize;
  int startingRow = 0;
  int endingRow = rowSize-1;
  int startingColumn = 0;
  int endingColumn = colSize -1;

  vector<int> result;    //resultant array to save the result

  while(startingRow <=endingRow && startingColumn<=endingColumn){
    //left to right
    for(int c= startingColumn; c<=endingColumn; c++)
          result.push_back(array[startingRow][c]);
    //top to bottom
    for(int r = startingRow+1; r<=endingRow; r++)
          result.push_back(array[r][endingColumn]);
    // when one perimeter is completed and inside section only has one row
    // or one col so to avoid double couting we are adding a check.
    if(startingRow==endingRow || startingColumn==endingColumn)
        break;
    //right to left
    for(int c = endingColumn-1; c>=startingColumn ; c--)
          result.push_back(array[endingRow][c]);
    //bottom to top
    for(int r = endingRow-1; r>startingRow; r--)
          result.push_back(array[r][startingColumn]);
    startingRow++;
    startingColumn++;
    endingRow--;
    endingColumn--;
  }
  return result;
}
```

Solution 1 | Solution 2 | Solution 3

```cpp
using namespace std;
void traverse2D(vector<vector<int>> &array, int &lc, int &rc, int &tr, int &br, vector<int> &result){

  if(lc>rc || tr>br)
      return;
  for(int c=lc; c<=rc; c++)
      result.push_back(array[tr][c]);
    //top to bottom (top)
    for(int r=tr+1; r<=br; r++)
      result.push_back(array[r][rc]);
    //break point if it was single matrix either vertically or horizontally
    if(lc==rc || tr==br)
        return;
    //right to left(bottom)
    for(int c = rc-1; c>=lc; c--)
      result.push_back(array[br][c]);
    //bottom to top(bottom)
    for(int r=br-1; r>tr; r--)
      result.push_back(array[r][lc]);
    lc++;rc--;tr++;br--;
    traverse2D(array, lc, rc, tr, br, result);


}

vector<int> spiralTraverse(vector<vector<int>> array) {
  // Write your code here.
  int leftCol= 0;
  int rightCol= array[0].size() -1;
  int topRow = 0;
  int bottomRow = array.size() -1;

  vector<int> result;

  traverse2D(array, leftCol, rightCol, topRow, bottomRow, result);

  return result;
}
```

6

10 August 2023 11:51 PM

Solution 1 | Solution 2 | Solution 3

```cpp
using namespace std;
int longestPeak(vector<int> array) {
  // Write your code here.
  int len = array.size();
  int longestPeakLength = 0;

  int i =1;

  if(array.size()==0)
      return 0;
  while(i<len-1){
    //when peak is found
    if(array[i]>array[i-1] && array[i]>array[i+1]){
      int left = i-1;
      while(left>=0 && array[left]< array[left+1])
          left--;
      int right = i+1;
      while(right<=len-1 && array[right]<array[right-1])
          right++;

      int currentPeakL = right-left-1;
      longestPeakLength = max(currentPeakL, longestPeakLength);

      i=i+2;
    }
    else{
      i++;
      continue;
    }
  }

  return longestPeakLength;
}
```

```cpp
//APPROACH :
// we will look for the peak and then we will expand it to left and then to
// right. for peak a[i] should be greater than its previous as well as next
// value. if peak is found expan it to left i.e. initiate a left index to
// i-1 and check till left>=0 that a[left] < a[left+1], then check for
right
// to by initiating a right index to i+1. this will give us left and right
// index value we can find current peak length by (right-left-1). then we
can
// choose out from current and already stored in longestPeakL. then we
will increment
// i as new peak of larger length can be there. so instead of incrementing
// i with 1. we already know if peak is there the next value is already
small
// than that of a[i]. so increment i with i+2. if peak is not found keep
// on incrementing i with 1.
```

Difficulty: **Category:** [redacted] **Successful Submissions:** 46,885+

Longest Peak

Write a function that takes in an array of integers and returns the length of the longest peak in the array.

A peak is defined as adjacent integers in the array that are **strictly** increasing until they reach a tip (the highest value in the peak), at which point they become **strictly** decreasing. At least three integers are required to form a peak.

For example, the integers `1, 4, 10, 2` form a peak, but the integers `4, 0, 10` don't and neither do the integers `1, 2, 2, 0`. Similarly, the integers `1, 2, 3` don't form a peak because there aren't any strictly decreasing integers after the `3`.

Sample Input

```
array = [1, 2, 3, 3, 4, 0, 10, 6, 5, -1, -3, 2, 3]
```

Sample Output

```
6 // 0, 10, 6, 5, -1, -3
```

7

10 August 2023 11:51 PM

Difficulty: **Category:** **Successful Submissions:** 46,586+

Array Of Products

Write a function that takes in a non-empty array of integers and returns an array of the same length, where each element in the output array is equal to the product of every other number in the input array.

In other words, the value at `output[i]` is equal to the product of every number in the input array other than `input[i]`.

Note that you're expected to solve this problem without using division.

Sample Input

```
array = [5, 1, 4, 2]
```

Sample Output

```
[8, 40, 10, 20]
// 8 is equal to 1 x 4 x 2
// 40 is equal to 5 x 4 x 2
// 10 is equal to 5 x 1 x 2
// 20 is equal to 5 x 1 x 4
```

Solution 1 **Solution 2** **Solution 3**

```cpp
#include <vector>

using namespace std;

vector<int> arrayOfProducts(vector<int> array) {
  // Write your code here.
  // brute force approach takes n sqaured time.
  // non optimal solution.
  // if you are dealing with such problems, first think is
  // go for brute force and then analyze it to come up with
  // more optimal solution.
  int n =  array.size();

  vector<int> result;

  for(int i=0; i<n; i++){
    int prod =1;
    for(int j=0; j<n; ){
      if(i==j)
        j++;
      else{
        prod = prod*array[j];
        j++;
      }

    }
    result.push_back(prod);
  }

  return result;
}
```

```
// more optimal approach.
// to find product at ith pos. we can multiply prod of
// elements at left pos with prod of elem at right pos.
// so first we will create two arrays initialize it with
// 1. and then we will declare a variable prod starts with 1
// and before incrementing in array elem we will feed product
// at that pos in left array and the prod = prod*array[i];
```

Solution 1 | Solution 2 | Solution 3

```
#include <vector>
using namespace std;
vector<int> arrayOfProducts(vector<int> array) {
  // Write your code here.
  int n = array.size();
  int left[n];
  int right[n];
  vector<int> result;
  // initializing both arrays with 1.
  for(int i=0; i<n ; i++){
    left[i]=1;
    right[i]=1;
  }
  // feeding left array with prod of all left elements except with element at j.
  int j;
  int prod=1;
  for(j=0; j<n;j++){
    left[j]=prod;
    cout <<left[j]<<endl;
    prod = prod*array[j];
  }
  //now j is at end of array so we will fill for right element
  // in right array in reverse dir.
  prod =1; // reseting prod
  //feeding right array with prd of all right elem except
  // with element at j.
  for(j=n-1;j>=0; j--){
    right[j]=prod;
    prod = prod*array[j];
  }
  // now result of element at i is. result of prod of
  // all left with prod of right elem.
  // left array at 0 contains prod of all other left elem
  // right array at 0 contains prod of all right elem.
  // that's why in result array we are multiplying elem
  // at k in left with right array.
  for(int k=0; k<n; k++){
    int res = left[k]*right[k];
    result.push_back(res);
  }
  return result;
```

8

10 August 2023 11:51 PM

Difficulty: **Category:** **Successful Submissions:** 48,006+

First Duplicate Value

Given an array of integers between `1` and `n`, inclusive, where `n` is the length of the array, write a function that returns the first integer that appears more than once (when the array is read from left to right).

In other words, out of all the integers that might occur more than once in the input array, your function should return the one whose first duplicate value has the minimum index.

If no integer appears more than once, your function should return `-1`.

Note that you're allowed to mutate the input array.

Sample Input #1

```
array = [2, 1, 5, 2, 3, 3, 4]
```

Sample Output #1

```
2 // 2 is the first integer that appears more than once.
// 3 also appears more than once, but the second 3 appears after the se
```

Solution 1 **Solution 2** **Solution 3**

```cpp
#include <vector>
using namespace std;

//approach:
//first we create a vector container and loop through the
// given array and for every element we checked in our hash
// table if that element is there or not, if not insert it
// in the table and if found return that. cause if found
// that mean this element is first duplicate occuring
// element in the list. there can be others duplicate too but
// we have to find the first one.

int firstDuplicateValue(vector<int> array) {
  vector<int> elements;
  int n = array.size();
  vector<int>::iterator it;

  for(int i=0; i<n; i++){
    int key = array[i];

    it = find(elements.begin(),elements.end(),key);

    if(it!= elements.end())
      return key;

    elements.push_back(key);
  }

  return -1;

}
```

9

10 August 2023 11:51 PM

Difficulty: **Category:** **Successful Submissions:** 30,306+

Merge Overlapping Intervals

Write a function that takes in a non-empty array of arbitrary intervals, merges any overlapping intervals, and returns the new intervals in no particular order.

Each interval `interval` is an array of two integers, with `interval[0]` as the start of the interval and `interval[1]` as the end of the interval.

Note that back-to-back intervals aren't considered to be overlapping. For example, `[1, 5]` and `[6, 7]` aren't overlapping; however, `[1, 6]` and `[6, 7]` *are* indeed overlapping.

Also note that the start of any particular interval will always be less than or equal to the end of that interval.

Sample Input

```
intervals = [[1, 2], [3, 5], [4, 7], [6, 8], [9, 10]]
```

Sample Output

```
[[1, 2], [3, 8], [9, 10]]
// Merge the intervals [3, 5], [4, 7], and [6, 8].
// The intervals could be ordered differently.
```

```
//approach:
// we sort the 2D array.
// we store the first 1st interval in temp
// then we check if temp[1] < int[i][0], no overlapping
// else there is overlapping and we update temp cause we have
// to again check for the next subarray.
// we update temp using min element and max function.
// min from temp itself. and max from temp[1] and int[i][1]
```

Solution 1 Solution 2 Solution 3

```cpp
#include <vector>
using namespace std;
vector<vector<int>> mergeOverlappingIntervals(vector<vector<int>> intervals) {
  // Write your code here.
  vector<vector<int>> result;
  // sorting it as some cases may fail
  // example - (3,4) (1,2) here acc to our approach
  // 4>3 so we should update our temp interval
  // but if we see those intervals are not overlapping
  // that's why we need to sort it down.
  sort(intervals.begin(), intervals.end());
  int n = intervals.size();

  //saving first sub array in the temp vector.
  vector<int> temp = intervals[0];
  // so starting with i=1
  int i=1;

  if(n==0 || n==1){
    return result;
  }

  while(i<n){
    if(temp[1]<intervals[i][0]){
      result.push_back(temp);
      temp = intervals[i];
      i++;
    }else{
      //update temp
      temp[0]= *min_element(temp.begin(),temp.end());
      // as temp[1] is already bigger than intervals[i][0]
      // then only this else part is executing that's why
      // we ignored that from our max part criteria.
      temp[1]= max(temp[1],intervals[i][1]);
      i++;
    }
  }
  result.push_back(temp);
  return result;
}
```

10

10 August 2023 11:51 PM

Difficulty: **Category:** **Successful Submissions:** 3,069+

Best Seat

You walk into a theatre you're about to see a show in. The usher within the theatre walks you to your row and mentions you're allowed to sit anywhere within the given row. Naturally you'd like to sit in the seat that gives you the most space. You also would prefer this space to be evenly distributed on either side of you (e.g. if there are three empty seats in a row, you would prefer to sit in the middle of those three seats).

Given the theatre row represented as an integer array, return the seat index of where you should sit. Ones represent occupied seats and zeroes represent empty seats.

You may assume that someone is always sitting in the first and last seat of the row. Whenever there are two equally good seats, you should sit in the seat with the lower index. If there is no seat to sit in, return -1. The given array will always have a length of at least one and contain only ones and zeroes.

Sample Input

```
seats = [1, 0, 1, 0, 0, 0, 1]
```

Sample Output

```
4
```

Solution 1 | Solution 2 | Solution 3

```cpp
using namespace std;
#include <algorithm>
int bestSeat(vector<int> seats) {
  // Write your code here.
  int n = seats.size();
  vector<pair<int,int>> vect;
  if(n==0 || n==1)
      return -1;
  for(int i=0; i<n;){
    int count =0; int flag = true; int saveIndex;
    //while starts
    if(seats[i]==0){
      while(seats[i]==0){
      if(flag==true)
          {saveIndex = i;
          flag=false;}
      count++;
      i++;
    } //while ends
    vect.emplace_back(saveIndex, count);
    }
    else
        i++;
  } //for ends
  const auto maxCount = max_element(vect.begin(),
            vect.end(),
            [](const auto& lhs, const auto& rhs) { return lhs.second < rhs.second; });
  int res;
if(maxCount==vect.end())
    return -1;
else{
  if(maxCount->second %2 ==1){
    int odd = abs(maxCount->second/2 +1) -1;
    res = maxCount->first + odd;
  }
  else{
    int even = (maxCount->second/2) - 1;
    res = maxCount->first + even;
  }
}
return res;
```

11

10 August 2023 11:51 PM

Difficulty: **Category:** **Successful Submissions:** 5,980+

Zero Sum Subarray

You're given a list of integers `nums`. Write a function that returns a boolean representing whether there exists a zero-sum subarray of `nums`.

A zero-sum subarray is any subarray where all of the values add up to zero. A subarray is any contiguous section of the array. For the purposes of this problem, a subarray can be as small as one element and as long as the original array.

Sample Input

```
nums = [-5, -5, 2, 3, -2]
```

Sample Output

```
True // The subarray [-5, 2, 3] has a sum of 0
```

Solution 1 **Solution 2** **Solution 3**

```cpp
using namespace std;
//APPROACH : we check for empty or any 1 element.
// then we keep track of sum from i=0 to array size-1.
// if we get two sum equal that means [x+1, y] is the
// subarray that sum is 0.
// NOTE : We can use simple vector instead of vector pair
// we used pair so that if subarray is req. we can return
// it that as well.
bool zeroSumSubarray(vector<int> nums) {
  // Write your code here.
  int n = nums.size();
  //if array has no element.
  if(n==0)
    return false;
  // if array has element 0, then subarray can be small as
  // one element.
  for(auto x : nums)
    if(x==0)
      return true;
  int i=0;
  int sum=0;
  vector<pair<int,int>> vect;
  while(i<n){
    sum = sum + nums[i];
    i++;
    vect.emplace_back(sum, i);
  }
  //sorting the vect based on 1st element of pair
  sort(vect.begin(), vect.end());
  //after sorting, if pair(firstElement) 2 corresponding
  // values are equal or 0 is there then there is
  // subarray which sum is 0.
  for(int i=0; i<vect.size();){
    if((vect[i].first ==vect[i+1].first)||vect[i].first==0)
      return true;
    else{
      i++;
    }
  }
  return false;
```

12

11 August 2023 12:15 AM

Missing Numbers

You're given an unordered list of unique integers `nums` in the range `[1, n]`, where `n` represents the length of `nums + 2`. This means that two numbers in this range are missing from the list.

Write a function that takes in this list and returns a new list with the two missing numbers, sorted numerically.

Sample Input

```
nums = [1, 4, 3]
```

Sample Output

```
[2, 5]  // n is 5, meaning the completed list should be [1, 2, 3, 4, 5]
```

Solution 1 Solution 2 Solution 3

```cpp
using namespace std;

// APPROACH : we first run through the given vector and
// pushed every element in temp vect.
// we have to find 2 missing element in range [1,n],
// n = nums.size()+2 then we use find function in temp
// vector and check from [1,n], if key was found just
// increment else push that key cause that is the missing
// element and increment i.

vector<int> missingNumbers(vector<int> nums) {
  // Write your code here.
  int n = nums.size();
  vector<int> temp;
  vector<int> main;

  //pushed every item in nums to temp vector
  for(int i=0; i<n; i++){
    int ele = nums[i];
    temp.push_back(ele);
  }
  //checking from key 1 to key n+2. if not present push
  // that key in result vector.
 int i=1;
  vector<int>::iterator it;
  while(i<=n+2){
    int key=i;
    it = find(temp.begin(),temp.end(), key);
    if(it!=temp.end()) //key found
      i++;
    else
    {
      main.push_back(key);
      i++;
    }
  }

  for(int i=0; i<2; i++){
    cout << main[i] <<endl;
  }
  return main;
```

13

11 August 2023 12:15 AM

Majority Element

Write a function that takes in a non-empty, unordered `array` of positive integers and returns the array's majority element without sorting the array and without using more than constant space.

An array's majority element is an element of the array that appears in over half of its indices. Note that the most common element of an array (the element that appears the most times in the array) isn't necessarily the array's majority element; for example, the arrays `[3, 2, 2, 1]` and `[3, 4, 2, 2, 1]` both have `2` as their most common element, yet neither of these arrays have a majority element, because neither `2` nor any other element appears in over half of the respective arrays' indices.

You can assume that the input array will always have a majority element.

Sample Input

```
array = [1, 2, 3, 2, 2, 1, 2]
```

Sample Output

```
2 // 2 occurs in 4/7 array indices, making it the majority element
```

Solution 1 | Solution 2 | Solution 3

```cpp
using namespace std;

int majorityElement(vector<int> array) {
  // Write your code here.
  // to get the total no. of indices
  int length =array.size();
  int majority = abs(length/2);
  int majorityElement =0;
  int max=0;
  //pair vector
  vector<pair<int, int>> pairVector;

  for(auto element : array){
    bool found = false;
    //By using auto&, we ensure that we are modifying the
    // actual pairs in the vector, rather than making
    // copies of them.
    for(auto& pair : pairVector){
      if(pair.first == element){
        found =true;
        pair.second++;
        break;
      }
    }
    if(!found)
        pairVector.push_back(make_pair(element , 1));
  }
  //iterating and finding the element with greatest occurance
  // as inpur array has majority element for sure then the
  // element having maximum occurance is the majority element.
  // if there was a chance that array may not have majority
  //element then we would have checked it indices/2.
  for(auto& pair:pairVector){
    if(pair.second > max){
      max = pair.second;
      majorityElement = pair.first;
    }
  }
  return majorityElement;
}
```

14

11 August 2023 12:15 AM

Difficulty: **Category:** **Successful Submissions:** 1,284+

Sweet And Savory

You're hosting an event at a food festival and want to showcase the best possible pairing of two dishes from the festival that complement each other's flavor profile.

Each dish has a flavor profile represented by an integer. A negative integer means a dish is sweet, while a positive integer means a dish is savory. The absolute value of that integer represents the intensity of that flavor. For example, a flavor profile of -3 is slightly sweet, one of -10 is extremely sweet, one of 2 is mildly savory, and one of 8 is significantly savory.

You're given an array of these dishes and a target combined flavor profile. Write a function that returns the best possible pairing of two dishes (the pairing with a total flavor profile that's closest to the target one). Note that this pairing must include one sweet and one savory dish. You're also concerned about the dish being too savory, so your pairing should never be more savory than the target flavor profile.

All dishes will have a positive or negative flavor profile; there are no dishes with a 0 value. For simplicity, you can assume that there will be at most one best solution. If there isn't a valid solution, your function should return `[0, 0]`. The returned array should be sorted, meaning the sweet dish should always come first.

Sample Input #1

```
dishes = [-3, -5, 1, 7]
target = 8
```

Sample Output #1

```
[-3, 7] // The combined profile of 4 is closest without going over
```

Solution 1 **Solution 2** **Solution 3**

```cpp
using namespace std;

vector<int> sweetAndSavory(vector<int> dishes, int target) {
  // Write your code here.
  if(dishes.size()==0)
    return {0,0};
  vector<int> sweet, savory;
  for(int dish:dishes){
    if(dish<0)
      sweet.push_back(dish);
    else
      savory.push_back(dish);
  }
  //greater<int>() to sort it in descending order, just for sweet
  // as they are negative values.
  sort(sweet.begin(), sweet.end(), greater<int>());
  sort(savory.begin(), savory.end());
  int sweetIndex= 0;
  int savoryIndex =0 ;
  vector<int> bestPair(2,0);
  //setting initial diff to max value of int.
  int minDifference = INT_MAX;
  while(sweetIndex < sweet.size() && savoryIndex < savory.size()){
    int currentSum = sweet[sweetIndex] + savory[savoryIndex];

    if(currentSum <=target){
      int currentDifference = target-currentSum;
      if(currentDifference < minDifference){
        minDifference = currentDifference;
        int dish_1 = sweet[sweetIndex];
        int dish_2 = savory[savoryIndex];
        bestPair[0] = dish_1;
        bestPair[1] = dish_2;
      }
      savoryIndex++;
    }
    else
      sweetIndex++;
  }

  return bestPair;
```

15

11 August 2023 12:15 AM

Solution 1 Solution 2 Solution 3

```cpp
#include <vector>
using namespace std;
#include <iostream>
#include<algorithm>

class BST {
public:
  int value;
  BST *left;
  BST *right;
  BST(int val) {
    value = val;
    left = nullptr;
    right = nullptr;
  }
  BST &insert(int val) {
    if (val < value) {
      if (left == nullptr) {
        left = new BST(val);
      } else {
        left->insert(val);
      }
    } else {
      if (right == nullptr) {
        right = new BST(val);
      } else {
        right->insert(val);
      }
    }
    return *this;
  }
```

Difficulty: **Category:** **Successful Submissions:** 27,490+

BST Construction

Write a `BST` class for a Binary Search Tree. The class should support:

- Inserting values with the `insert` method.
- Removing values with the `remove` method; this method should only remove the first instance of a given value.
- Searching for values with the `contains` method.

Note that you can't remove values from a single-node tree. In other words, calling the `remove` method on a single-node tree should simply not do anything.

Each `BST` node has an integer `value`, a `left` child node, and a `right` child node. A node is said to be a valid `BST` node if and only if it satisfies the BST property: its `value` is strictly greater than the values of every node to its left; its `value` is less than or equal to the values of every node to its right; and its children nodes are either valid `BST` nodes themselves or `None` / `null`.

```
33   bool contains(int val) {
34     if (val == value) {
35       return true;
36     } else if (val < value) {
37       if (left == nullptr) {
38         return false;
39       } else {
40         return left->contains(val);
41       }
42     } else {
43       if (right == nullptr) {
44         return false;
45       } else {
46         return right->contains(val);
47       }
48     }
49   }
50
51   BST &remove(int val) {
52     removeHelper(val, nullptr);
53     return *this;
54   }
```

```
56 private:
57   void removeHelper(int val, BST *parent) {
58     if (val < value) {
59       if (left != nullptr) {
60         left->removeHelper(val, this);
61       }
62     } else if (val > value) {
63       if (right != nullptr) {
64         right->removeHelper(val, this);
65       }
66     }
67     // if the node is root node.
68     else {
69       if (left != nullptr && right != nullptr) {
70         value = right->getMinValue();
71         right->removeHelper(value, this);
72       } else if (parent == nullptr) {
73         if (left != nullptr) {
74           value = left->value;
75           right = left->right;
76           left = left->left;
77         } else if (right != nullptr) {
78           value = right->value;
79           left = right->left;
80           right = right->right;
81         } else {// This is a single-node tree; do nothing
82         }
83       } else if (parent->left == this) {
84         parent->left = (left != nullptr) ? left : right;
85       } else if (parent->right == this) {
86         parent->right = (left != nullptr) ? left : right;
87       }
88     }
89   }
90   int getMinValue() {
91     if (left == nullptr) {
92       return value;
93     } else {
94       return left->getMinValue();
95     }
96   }
```

16

11 August 2023 12:15 AM

Difficulty: **Category:** **Successful Submissions:** 41,482+

Validate BST

Write a function that takes in a potentially invalid Binary Search Tree (BST) and returns a boolean representing whether the BST is valid.

Each `BST` node has an integer `value`, a `left` child node, and a `right` child node. A node is said to be a valid `BST` node if and only if it satisfies the BST property: its `value` is strictly greater than the values of every node to its left; its `value` is less than or equal to the values of every node to its right; and its children nodes are either valid `BST` nodes themselves or `None` / `null`.

A BST is valid if and only if all of its nodes are valid `BST` nodes.

Sample Input

```
tree =   10
       /    \
      5      15
    /   \   /   \
   2     5 13   22
 /           \
1            14
```

Sample Output

```
true
```

Solution 1 **Solution 2** **Solution 3**

```
#include <limits.h>
class BST {
public:
  int value;
  BST *left;
  BST *right;

  BST(int val);
  BST &insert(int val);
};


bool validateBstHelper(BST *root, int min, int max){
    if(root==NULL)
      return true;
  // means node value is outside of left and right value
  // so the tree is not a valid BST.
  //here root->Value >=max cause that needs to be lesser than the max value.
  // and when running on right subtree, current node value should be greater or equal
  // to the minimum value(prev node value).
  // as root value should always be greater than the value of left subtree
  // and equal to or lesser than right subtree.
    if(root->value>=max || root->value<min)
      return false;

    //root->left becomes current root node, and for that its max value
    // is the root node value. so current node needs to be less than
  // previous root node value.
    bool isLeftValid = validateBstHelper(root->left, min, root->value);
    bool isRightValid = validateBstHelper(root->right, root->value, max);

  return isLeftValid && isRightValid;

}
bool validateBst(BST *tree) {
  // Write your code here.
  return validateBstHelper(tree, INT_MIN, INT_MAX);
  return false;
}
```

17

11 August 2023 12:15 AM

Difficulty: **Category:** **Successful Submissions:** 39,986+

BST Traversal

Write three functions that take in a Binary Search Tree (BST) and an empty array, traverse the BST, add its nodes' values to the input array, and return that array. The three functions should traverse the BST using the in-order, pre-order, and post-order tree-traversal techniques, respectively.

If you're unfamiliar with tree-traversal techniques, we recommend watching the Conceptual Overview section of this question's video explanation before starting to code.

Each `BST` node has an integer `value`, a `left` child node, and a `right` child node. A node is said to be a valid `BST` node if and only if it satisfies the BST property: its `value` is strictly greater than the values of every node to its left; its `value` is less than or equal to the values of every node to its right; and its children nodes are either valid `BST` nodes themselves or `None` / `null`.

Sample Input

```
tree =   10
       /    \
      5      15
    /   \      \
   2     5      22
 /
1
array = []
```

Sample Output

```
inOrderTraverse: [1, 2, 5, 5, 10, 15, 22] // where the array is the in
preOrderTraverse: [10, 5, 2, 1, 5, 15, 22] // where the array is the i
postOrderTraverse: [1, 2, 5, 5, 22, 15, 10] // where the array is the
```

Solution 1 | Solution 2 | Solution 3

```cpp
#include <vector>
using namespace std;

class BST {
public:
  int value;
  BST *left;
  BST *right;

  BST(int val);
};

//left -> push the data -> right
void inOrderTraverse(BST *tree, vector<int> &array) {
  // Write your code here.
  if(tree!=nullptr){
    inOrderTraverse(tree->left, array);
    array.push_back(tree->value);
    inOrderTraverse(tree->right, array);
  }
}
//push the data -> left -> right
void preOrderTraverse(BST *tree, vector<int> &array) {
  // Write your code here.
  if(tree!=nullptr){
    array.push_back(tree->value);
    preOrderTraverse(tree->left, array);
    preOrderTraverse(tree->right, array);
  }
}
//left -> right -> push the data.
void postOrderTraverse(BST *tree, vector<int> &array) {
  // Write your code here.
  if(tree!=nullptr){
    postOrderTraverse(tree->left, array);
    postOrderTraverse(tree->right, array);
    array.push_back(tree->value);
  }
}

```

18.1

11 August 2023 01:27 AM

Difficulty: **Category:** **Successful Submissions:** 33,606+

Min Height BST

Write a function that takes in a non-empty sorted array of distinct integers, constructs a BST from the integers, and returns the root of the BST.

The function should minimize the height of the BST.

You've been provided with a `BST` class that you'll have to use to construct the BST.

Each `BST` node has an integer `value`, a `left` child node, and a `right` child node. A node is said to be a valid `BST` node if and only if it satisfies the BST property: its `value` is strictly greater than the values of every node to its left; its `value` is less than or equal to the values of every node to its right; and its children nodes are either valid `BST` nodes themselves or `None` / `null`.

A BST is valid if and only if all of its nodes are valid `BST` nodes.

Note that the `BST` class already has an `insert` method which you can use if you want.

Sample Input

```
array = [1, 2, 5, 7, 10, 13, 14, 15, 22]
```

Sample Output

```
        10
      /    \
     2      14
   /  \   /   \
  1    5 13   15
        \       \
         7      22
```

18.2

11 August 2023 01:31 AM

Solution 1 Solution 2 Solution 3

```cpp
using namespace std;

class BST {
public:
  int value;
  BST *left;
  BST *right;

  BST(int value) {
    this->value = value;
    left = nullptr;
    right = nullptr;
  }

  void insert(int value) {
    if (value < this->value) {
      if (left == nullptr) {
        left = new BST(value);
      } else {
        left->insert(value);
      }
    } else {
      if (right == nullptr) {
        right = new BST(value);
      } else {
        right->insert(value);
      }
    }
  }
};
```

```cpp
void minHeightBstHelper(int start, int last, vector<int>&arr, BST *&root){
  // recursion stop condition.

  if(start<last){
    // mid will optimize the height of the BST.
    // and it will be updated for every recursive call.
    int mid = (start+last)/2;
    // if starting root is null, set arr[mid] to root, and that will be the root
    // of the tree that will be created.
    if(!root)
      root = new BST(arr[mid]);
      // if root is not null, run insert on root object while passing the value v[mid].
    else
      root->insert(arr[mid]);

    // now basically we want to run the above approach on the left array of mid and
    // right side of mid.
    // for left, starting point is 0 and ending point is mid.
    minHeightBstHelper(start, mid, arr, root);
    //for right, starting point is mid+1, and ending is last.
    minHeightBstHelper(mid+1 , last ,arr, root);
  }
}
BST *minHeightBst(vector<int> array) {
  // Write your code here.
  // initiating a root with nullptr
  BST *root = nullptr;
  //starting point, last point , array, and root.
  minHeightBstHelper(0, array.size(), array, root);

  return root;
}
```

19

11 August 2023 01:27 AM

Difficulty: **Category:** **Successful Submissions:** 27,254+

Find Kth Largest Value In BST

Write a function that takes in a Binary Search Tree (BST) and a positive integer `k` and returns the kth largest integer contained in the BST.

You can assume that there will only be integer values in the BST and that `k` is less than or equal to the number of nodes in the tree.

Also, for the purpose of this question, duplicate integers will be treated as distinct values. In other words, the second largest value in a BST containing values `{5, 7, 7}` will be `7` —not `5`.

Each `BST` node has an integer `value`, a `left` child node, and a `right` child node. A node is said to be a valid `BST` node if and only if it satisfies the BST property: its `value` is strictly greater than the values of every node to its left; its `value` is less than or equal to the values of every node to its right; and its children nodes are either valid `BST` nodes themselves or `None` / `null`.

Sample Input

```
tree =   15
       /    \
      5      20
    /   \   /   \
   2     5 17   22
 /   \
1     3
k = 3
```

Sample Output

```
17
```

Solution 1 **Solution 2** **Solution 3**

```cpp
using namespace std;

// This is an input class. Do not edit.
class BST {
public:
  int value;
  BST *left = nullptr;
  BST *right = nullptr;

  BST(int value) { this->value = value; }
};

void inorderTrav(BST*root, vector<int>&nodes){
  if(root){
    inorderTrav(root->left, nodes);
    nodes.push_back(root->value);
    inorderTrav(root->right, nodes);
  }
}

int findKthLargestValueInBst(BST *tree, int k) {
  // Write your code here.

  vector<int> nodes;
  inorderTrav(tree, nodes);

  int length = nodes.size();
  int kthLargestValue = nodes[length-k];

  //kth largest means kth value from the last side of the
  // inorder Traversal of the BST.



  return kthLargestValue;
}

```

20.1

11 August 2023 01:27 AM

Difficulty: **Category:** **Successful Submissions:** 18,678+

Reconstruct BST

The pre-order traversal of a Binary Tree is a traversal technique that starts at the tree's root node and visits nodes in the following order:

1. Current node
2. Left subtree
3. Right subtree

Given a non-empty array of integers representing the pre-order traversal of a Binary Search Tree (BST), write a function that creates the relevant BST and returns its root node.

The input array will contain the values of BST nodes in the order in which these nodes would be visited with a pre-order traversal.

Each `BST` node has an integer `value`, a `left` child node, and a `right` child node. A node is said to be a valid `BST` node if and only if it satisfies the BST property: its `value` is strictly greater than the values of every node to its left; its `value` is less than or equal to the values of every node to its right; and its children nodes are either valid `BST` nodes themselves or `None` / `null`.

Sample Input

```
preOrderTraversalValues = [10, 4, 2, 1, 5, 17, 19, 18]
```

Sample Output

```
        10
      /    \
     4      17
   /   \      \
  2     5     19
 /           /
1           18
```

20.2

11 August 2023 01:34 AM

Solution 1 Solution 2 Solution 3

```cpp
#include<stack>
#include <vector>
#include<algorithm>
#include <iostream>
using namespace std;

// This is an input class. Do not edit.
class BST {
public:
  int value;
  BST *left = nullptr;
  BST *right = nullptr;

  BST(int value) { this->value = value; }
};

void preBST(BST*&root, vector<int>&pre){
    //creating a stack
  stack<BST *> st;
  int i=0;
  BST *p; //to keep track of nodes.
  BST *temp; //for creating new nodes.
  int length = pre.size(); // run the loop till i<length
  //manually create root node, then repeating steps
  root = new BST(pre[i++]); //initiate root with pre[i] and increament i.
  root->left=root->right=nullptr;
  p = root;
  //we will use while cause not every time i is incremented.
  while(i<length){
    if(pre[i]<p->value){
      temp = new BST(pre[i++]);
      temp->left = temp->right =nullptr;
      p->left = temp;
      st.push(p);  //pushing address of current p in stack
      p=temp;
    }
    else{
      if(!st.empty() && pre[i]<st.top()->value ){
        temp = new BST(pre[i++]);
        temp->left = temp->right =nullptr;
        p->right =temp;
        p = temp;  //when a right child is created, don't push the address.
      }
      else if(!st.empty() && pre[i]>=st.top()->value){
        p= st.top();  // else take a address using top and make p point to it, don't increment i,
        st.pop();     // and delete that element which address we are using, using pop method.
        }             // so that the node deleted from stack is the current node,
                  //and then the current element is check for the same.
      else {
        temp=new BST(pre[i++]);
        temp->left = temp->right =nullptr;
        p->right = temp;
        p = temp;
      }
    }
  }
}

BST *reconstructBst(vector<int> preOrderTraversalValues) {
  // Write your code here.
  BST *root = nullptr;

  preBST(root, preOrderTraversalValues);

  return root;
}

```

21

11 August 2023 01:27 AM

Difficulty: **Category:** **Successful Submissions:** 45,924+

Invert Binary Tree

Write a function that takes in a Binary Tree and inverts it. In other words, the function should swap every left node in the tree for its corresponding right node.

Each `BinaryTree` node has an integer `value`, a `left` child node, and a `right` child node. Children nodes can either be `BinaryTree` nodes themselves or `None` / `null`.

Sample Input

```
tree =    1
        /    \
       2      3
     /   \  /   \
    4     5 6    7
  /   \
 8     9
```

Sample Output

```
        1
     /     \
    3       2
  /   \   /   \
 7     6 5     4
           /   \
          9     8
```

Solution 1 **Solution 2** **Solution 3**

```cpp
#include <vector>
using namespace std;

class BinaryTree {
public:
  int value;
  BinaryTree *left;
  BinaryTree *right;

  BinaryTree(int value);
  void insert(vector<int> values, int i = 0);
  void invertedInsert(vector<int> values, int i = 0);
};

void invertBinaryTree(BinaryTree *tree) {
  // Write your code here.
 if(tree==nullptr)
   return ;
else{
   invertBinaryTree(tree->left);
  invertBinaryTree(tree->right);

  //swapping part;
  BinaryTree *temp = tree->left;
  tree->left = tree->right;
  tree->right = temp;
}
}

```

22

11 August 2023 01:27 AM

Difficulty: **Category:** **Successful Submissions:** 24,882+

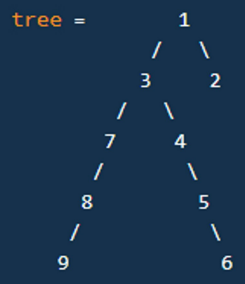
Binary Tree Diameter

Write a function that takes in a Binary Tree and returns its diameter. The diameter of a binary tree is defined as the length of its longest path, even if that path doesn't pass through the root of the tree.

A path is a collection of connected nodes in a tree, where no node is connected to more than two other nodes. The length of a path is the number of edges between the path's first node and its last node.

Each `BinaryTree` node has an integer `value`, a `left` child node, and a `right` child node. Children nodes can either be `BinaryTree` nodes themselves or `None` / `null`.

Sample Input



Sample Output

```
6 // 9 -> 8 -> 7 -> 3 -> 4 -> 5 -> 6
// There are 6 edges between the
// first node and the last node
// of this tree's longest path.
```

```cpp
using namespace std;

// This is an input class. Do not edit.
class BinaryTree {
public:
  int value;
  BinaryTree *left;
  BinaryTree *right;

  BinaryTree(int value) {
    this->value = value;
    left = nullptr;
    right = nullptr;
  }
};

int max(int a, int b){
  return (a>b)?a:b;
}
int height(BinaryTree *tree, int &ans){
  if(tree==nullptr)
    return 0;
  int leftH = height(tree->left,ans);
  int rightH = height(tree->right,ans);
    //total path for each
  int total = 1+leftH+rightH;
  ans = max(ans,total);

  return 1+max(leftH, rightH);
}

int binaryTreeDiameter(BinaryTree *tree) {
  // Write your code here.
  if(!tree)
    return -1;
  int ans = INT_MIN;
  height(tree,ans);
  ans--;
  return ans;
}
```

23.1

11 August 2023 01:27 AM

Difficulty: **Category:** **Successful Submissions:** 24,792+

Find Successor

Write a function that takes in a Binary Tree (where nodes have an additional pointer to their parent node) as well as a node contained in that tree and returns the given node's successor.

A node's successor is the next node to be visited (immediately after the given node) when traversing its tree using the in-order tree-traversal technique. A node has no successor if it's the last node to be visited in the in-order traversal.

If a node has no successor, your function should return `None` / `null`.

Each `BinaryTree` node has an integer `value`, a `parent` node, a `left` child node, and a `right` child node. Children nodes can either be `BinaryTree` nodes themselves or `None` / `null`.

Sample Input

```
tree =
            1
          /   \
         2     3
       /   \
      4     5
     /
    6
node = 5
```

Sample Output

```
1
// This tree's in-order traversal order is:
// 6 -> 4 -> 2 -> 5 -> 1 -> 3
// 1 comes immediately after 5.
```

23.2

11 August 2023 01:39 AM

Solution 1 Solution 2 Solution 3

```
using namespace std;
// This is an input class. Do not edit.
class BinaryTree {
public:
  int value;
  BinaryTree *left = nullptr;
  BinaryTree *right = nullptr;
  BinaryTree *parent = nullptr;
  BinaryTree(int value) { this->value = value; }
};
BinaryTree *RightMost(BinaryTree *root){
  while(root!=nullptr && root->right!=nullptr){
    root = root->right;
  }
  return root;
}
BinaryTree *findSuccessor(BinaryTree *tree, BinaryTree *node) {
  // Write your code here.
    // CASE 1 : when node->right is NULL. then go till the parent
    // p->left==node, if not update node with node->parent.
    // CASE 3 : here we will check the case 3 also, node is the
    // rightmost child. as rightmost ->right will be null too, so the case 1
  // and case 2 don't collide that's why we  will check for rightmost first inside case 1.
    if(node->right==nullptr){

      //case 3:
      BinaryTree* rightmostNode = RightMost(tree);
      if(rightmostNode==node)
        return nullptr;

      //case 1:
      while(node->parent!=nullptr){
      if(node->parent->left == node){
          cout << node->parent->value;
          return node->parent;
      }
      node = node->parent;
    }
    }
```

```

    //CASE 2 : WHEN node->right is not null, then successor is same
    // as bst inorder successor.
    if(node->right !=nullptr){
      if(node->right->left==nullptr)
        return node->right;
      node = node->right;
      while(node->left!=nullptr){
        node = node->left;
      }
      return node;
    }
    //CASE 3 : WHEn node is the rightmost child , return nullptr



  return nullptr;
}
```

24

11 August 2023 01:27 AM

Difficulty: **Category:** **Successful Submissions:** 19,422+

Height Balanced Binary Tree

You're given the root node of a Binary Tree. Write a function that returns `true` if this Binary Tree is height balanced and `false` if it isn't.

A Binary Tree is height balanced if for each node in the tree, the difference between the height of its left subtree and the height of its right subtree is at most `1`.

Each `BinaryTree` node has an integer `value`, a `left` child node, and a `right` child node. Children nodes can either be `BinaryTree` nodes themselves or `None` / `null`.

Sample Input

```
tree = 1
     /   \
    2     3
  /   \     \
 4     5     6
     /   \
    7     8
```

Sample Output

```
true
```

Solution 1 **Solution 2** **Solution 3**

```cpp
using namespace std;

// This is an input class. Do not edit.
//APPROACH :
// What I did is, just calculated height for each subtree.
// height is heighest length b/w root to the leaf node.
// we find the diff b/w each subtree height and stored each answer
// based on the ans condition and when the functions gets
// over we check the value of ans, if that was greater than
// 1, means it wasn't height balanced.
class BinaryTree {
public:
  int value;
  BinaryTree *left = nullptr;
  BinaryTree *right = nullptr;

  BinaryTree(int value) { this->value = value; }
};
int max(int a, int b){
  return (a>b)?a:b;
}

int heightBalanced(BinaryTree *tree, int &ans){
  if(!tree)
    return false;
  int leftH = heightBalanced(tree->left, ans);
  int rightH = heightBalanced(tree->right,ans);
  int diff = abs(leftH - rightH);
  ans = max(ans,diff);
  return 1+max(leftH, rightH);
}

bool heightBalancedBinaryTree(BinaryTree *tree) {
  // Write your code here.
  int ans= INT_MIN;
  heightBalanced(tree, ans);
  if(ans>1)
    return false;
  return true;
}
```

25

11 August 2023 01:27 AM

Difficulty: **Category:** **Successful Submissions:** 3,144+

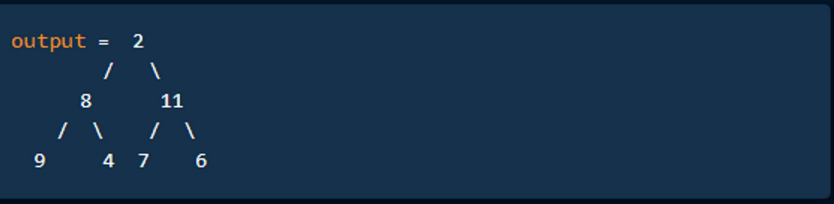
Merge Binary Trees

Given two binary trees, merge them and return the resulting tree. If two nodes overlap during the merger then sum the values, otherwise use the existing node.

Note that your solution can either mutate the existing trees or return a new tree.

Sample Input

Sample Output



Solution 1 | Solution 2 | Solution 3

```cpp
using namespace std;
// This is an input class. Do not edit.
class BinaryTree {
public:
  int value;
  BinaryTree *left = nullptr;
  BinaryTree *right = nullptr;
  BinaryTree(int value) {
    this->value = value;
  }
};
// if we are modifying tree 1, then the comparison will be
// done with tree2, means we will check if tree2->left is there
// but tree1->left is not there, then to add that tree2->left
// we will balance both left child by creating new node
// initializing with 0 same with right.
// cause if tree1->right is there, tree2->right is not there
// there is nothing we have to do as we are returning tree1
// itself.
void preOrder(BinaryTree* &root1, BinaryTree* root2){
   if(!root2)   // first time this cond. will true, this will become false when
     return ;
  if(root1->left==nullptr && root2->left)
    root1->left =new BinaryTree(0); // create with 0 to balance all the childs.
  //doing same for right.
  if(root1->right==nullptr && root2->right)
    root1->right = new BinaryTree(0);
  root1->value +=root2->value;
  preOrder(root1->left,root2->left);
  preOrder(root1->right,root2->right);
}
BinaryTree* mergeBinaryTrees(BinaryTree* tree1, BinaryTree* tree2) {
  // Write your code here.
  if(tree1 ==nullptr)
    return tree2;
  if(tree2 == nullptr)
    return tree1;
  BinaryTree* mainRoot = tree1;
  preOrder(tree1, tree2);
  return mainRoot;
}
```

26

11 August 2023 07:47 PM

Difficulty: 🟦 **Category:** [redacted] **Successful Submissions:** 3,137+

Symmetrical Tree 🟢 ☆

Write a function that takes in a Binary Tree and returns if that tree is symmetrical. A tree is symmetrical if the left and right subtrees are mirror images of each other.

Each `BinaryTree` node has an integer `value`, a `left` child node, and a `right` child node. Children nodes can either be `BinaryTree` nodes themselves or `None` / `null`.

Sample Input

```
tree =    1
        /   \
       2     2
      / \   / \
     3   4 4   3
    /     \     \
   5       6     5
```

Sample Output

```
True
```

Solution 1 **Solution 2** **Solution 3**

```cpp
using namespace std;
// This is an input class. Do not edit.
//APPROACH : run inorder Traversal and store in a vector
// then check if vec[i] = vec[last](i++, last--) till
// i==last.
class BinaryTree {
public:
  int value;
  BinaryTree *left = nullptr;
  BinaryTree *right = nullptr;

  BinaryTree(int value) {
    this->value = value;
  }
};

void inOrder(BinaryTree* root, vector<int>&vec){
  if(root!=nullptr){
    inOrder(root->left, vec);
    vec.push_back(root->value);
    inOrder(root->right, vec);
  }
}

bool symmetricalTree(BinaryTree* tree) {
  // Write your code here.
  vector<int> v;
  inOrder(tree, v);
  int left =0;
  int right = v.size()-1;
  while(left<right){
    if(v[left]!=v[right])
      return false;
    left++;
    right--;
  }
  return true;
}
```

27

11 August 2023 07:47 PM

Difficulty: **Category:** **Successful Submissions:** 1,213+

Split Binary Tree

Write a function that takes in a Binary Tree with at least one node and checks if that Binary Tree can be split into two Binary Trees of equal sum by removing a single edge. If this split is possible, return the new sum of each Binary Tree, otherwise return 0. Note that you do not need to return the edge that was removed.

The sum of a Binary Tree is the sum of all values in that Binary Tree.

Each `BinaryTree` node has an integer `value`, a `left` child node, and a `right` child node. Children nodes can either be `BinaryTree` nodes themselves or `None` / `null`.

Sample Input

```
tree =     1
         /   \
        3     -2
      /  \   /  \
     6   -5 5   2
   /
  2
```

Sample Output

```
  // Remove the edge to the left of the root node,
6 // creating two trees, each with sums of 6
```

Solution 1 | Solution 2 | Solution 3

```cpp
using namespace std;// This is an input class. Do not edit.
class BinaryTree {
public:
  int value;
  BinaryTree *left = nullptr;
  BinaryTree *right = nullptr;
  BinaryTree(int value) { this->value = value; }
};
int findTreeSum(BinaryTree *tree){
  if(!tree)
    return 0;
  return tree->value + findTreeSum(tree->left)+ findTreeSum(tree->right);
}
int canSplit(BinaryTree* root, int desiredSum, bool &flag){
  if(!root || flag) //exit as soon as flag turns true.
    return 0;
  int leftSum = canSplit(root->left, desiredSum, flag);
  int rightSum = canSplit(root->right, desiredSum, flag);
  int subtreeSum = leftSum + rightSum + root->value; //for each node the sum is left child
  if(subtreeSum == desiredSum){
    flag = true;
  }
  return subtreeSum;
}
int splitBinaryTree(BinaryTree *tree) {
  // Write your code here.
  int treeSum = findTreeSum(tree);
  int desiredSum = treeSum/2;
  bool flag =false;
  if(!tree)
    return -1;
  if(treeSum%2==1)
    return 0;
  else{
    canSplit(tree, desiredSum, flag);
    if(flag) //the function canSplit turned flag to true, means split was possible so retur
      return desiredSum;
    else return 0;
  }
  return -1;
```

28

11 August 2023 07:47 PM

Difficulty: **Category:** **Successful Submissions:** 43,438+

Max Subset Sum No Adjacent

Write a function that takes in an array of positive integers and returns the maximum sum of non-adjacent elements in the array.

If the input array is empty, the function should return `0`.

Sample Input

```
array = [75, 105, 120, 75, 90, 135]
```

Sample Output

```
330 // 75 + 120 + 135
```

Solution 1 Solution 2 Solution 3

```cpp
#include <vector>
using namespace std;

//APPROACH : approach is pretty basic, we use dynamic Pro
// and build smaller solution then fill up our final solution
// we write 2 base cases, then using that we create a new
// array of same size, and till that index we find the
// maximum sum with no, then come up with a formula
// maxSum[i] = max(maxSum[i-1], maxSum[i-2]+arr[i]);

int maxSubsetSumNoAdjacent(vector<int> array) {
  // Write your code here.
  int length = array.size();
  vector<int> maxSums;
  //base cases :
  if(length==0)
    return 0;
  else if(length==1)
    return array[0];
  else if(length==2){
    return (array[0]>array[1]) ? array[0] : array[1];
  }
  else {

    maxSums.push_back(array[0]);
    maxSums.push_back((array[0]>array[1]) ? array[0] : array[1]);
    int i=2;
    while(i<length){
      maxSums.push_back(max(maxSums[i-1], (maxSums[i-2]+array[i])));
      i++;
    }
  }

  int result = maxSums[length-1];
  return result;
}
```

29

11 August 2023 07:47 PM

Difficulty: **Category:** **Successful Submissions:** 34,218+

Number Of Ways To Make Change

Given an array of distinct positive integers representing coin denominations and a single non-negative integer `n` representing a target amount of money, write a function that returns the number of ways to make change for that target amount using the given coin denominations.

Note that an unlimited amount of coins is at your disposal.

Sample Input

```
n = 6
denoms = [1, 5]
```

Sample Output

```
2 // 1x1 + 1x5 and 6x1
```

Solution 1 **Solution 2** **Solution 3**

```cpp
#include <vector>
using namespace std;
// This is an optimization ques which uses overlapping solution
// and we are using top down DP[memoization].

//APPROACH : we make an array of size n+1, initializing with 0 except
// vec[0]= 1, which is a base case as there is 1 way to make change 0, by no
// using any coin. so this array represents the target at the end
// and for every denom we are getting closer to the result.
// at the end, value at the n index (which represents n)
// is the solution. suppose if target is 10, we pass 1st denom
// and check from 0 till 10 , when next denom is passed
// value at 10 index of vector is the ways to make change of that coin
// with 1st denom only.
// if denom is greater than target move to next index.
// if denom <=target  ways[amount] = ways[amount]+ways[amount-denom]
// which is a overlapping solution.

int numberOfWaysToMakeChange(int n, vector<int> denoms) {
  // Write your code here.
  vector<int> ways(n+1, 0); //size of n+1, initializing with 0
  ways[0] =1 ; //except for base case.

  if(n==0)
    return 1;

  for(int denom: denoms){
    for(int amount =0; amount <= n; amount++){
      if(denom <=amount)
        ways[amount] += ways[amount-denom];
    }
  }


  return ways[n];
}

```

57.1

11 August 2023 08:50 PM

Min Number Of Coins For Change

Given an array of positive integers representing coin denominations and a single non-negative integer `n` representing a target amount of money, write a function that returns the smallest number of coins needed to make change for (to sum up to) that target amount using the given coin denominations.

Note that you have access to an unlimited amount of coins. In other words, if the denominations are `[1, 5, 10]`, you have access to an unlimited amount of `1`s, `5`s, and `10`s.

If it's impossible to make change for the target amount, return `-1`.

Sample Input

```
n = 7
denoms = [1, 5, 10]
```

Sample Output

```
3 // 2x1 + 1x5
```

Solution 1 | Solution 2 | Solution 3

```cpp
#include <vector>
using namespace std;

int minNumberOfCoinsForChange(int n, vector<int> denoms) {
  // Write your code here.

  //first we create a array of size target+1.
  vector<int> targetChange(n+1, INT_MAX);
  targetChange[0] = 0;
  // now we don't need to initialize 0th with 1, as in previous
  // question we were asked about ways , now here it is actual
  // minimum no. of coins required to achieve that change.
  // so with 1 rupee coin, 0 coins are required to make
  // change of rupee 0, so we initialize 0th with 0 and the rest
  //with int max. cause we need a min if we would have initialized it with
  // 0, then everytime 0 would have been selected during the min function.
  // now the coins required for the selected denom to make that
  // change is equal to 1+ arr(i-denom). then we select
  // the minimum coins from the current denom and from prev denom.
  // everytime a new denom is passed we do not consider it as an individual
  // suppose 1st 1 is passed and then next 5 is passed second time
  // the concept is [1,5] we are checking for.
  if(denoms.size()==0)
    return -1;
  for(int amount : denoms){
      for(int i=amount; i<=n; i++ ){
        if(targetChange[i-amount]!=INT_MAX){
        targetChange[i] = min(targetChange[i], 1+targetChange[i-amount]);
      }
        else
          targetChange[i] = min(targetChange[i], targetChange[i-amount]);
      }
  }

  return targetChange[n]==INT_MAX ? -1 : targetChange[n];
}

```

57.2

11 August 2023 09:10 PM

Solution 1 **Solution 2** Solution 3

```cpp
#include <vector>
using namespace std;

int minNumberOfCoinsForChange(int n, vector<int> denoms) {
  // Write your code here.

  //first we create a array of size target+1.
  vector<int> targetChange(n+1, n+1);
  targetChange[0] = 0;
  // now we don't need to initialize 0th with 1, as in previous
  // question we were asked about ways , now here it is actual
  // minimum no. of coins required to achieve that change.
  // so with 1 rupee coin, 0 coins are required to make
  // change of rupee 0.
  if(denoms.size()==0)
    return -1;
  for(int amount : denoms){
      for(int i=amount; i<=n; i++ ){
          targetChange[i] = min(targetChange[i], 1+targetChange[i-amount]);
      }
  }

  return targetChange[n]==n+1 ? -1 : targetChange[n];
}

```

30

11 August 2023 07:47 PM

Difficulty: **Category:** [redacted] **Successful Submissions:** 24,808+

Levenshtein Distance

Write a function that takes in two strings and returns the minimum number of edit operations that need to be performed on the first string to obtain the second string.

There are three edit operations: insertion of a character, deletion of a character, and substitution of a character for another.

Sample Input

```
str1 = "abc"
str2 = "yabd"
```

Sample Output

```
2 // insert "y"; substitute "c" for "d"
```

```
  // as the question again asks for the minimum operations, so our approach will be
  // using dynamic programming and use simpler solution and then get to the final solution.

  // we use 2d vector in such questions. where rows and columns represents string 1 and 2
  // length respectively. it should look like this.
  // 0 1 2 3 4 ..
  // 1
  // 2
  // 3
  // the approach is, operation required to make emty string to empty string is 0.
  // the next is how many operations required to make str1[1] like str2[1], it takes
  // 1 operation and when we do the same for other in i=1th row, we will find out the pattern
  // if(str1[r-1]==str2[c-1]), then vect[r][c]= vect[r-1][c-1](the diagonal)
  // else 1+min(vect[r][c-1], vect[r-1][c], vect[r-1][c-1]) (the above, the left, and the diagonal)
  // r and c are the rows and col. we take r-1 and c-1 in string comparison, cause
  // as we take one extra 0 so str1[1] will check for the 2nd element.
```

Solution 1 **Solution 2** **Solution 3**

```
#include<vector>
using namespace std;

int levenshteinDistance(string str1, string str2) {
  // Write your code here.

  vector<vector<int>> operationTable(str2.size()+1, vector<int>(str1.size()+1, 0) );
  //2d array with str2.size +1, no. of rows and each row initialize with vector of
  // size str1.size()+1, and each value is 0.
  // now first column should be initialize with 0 1 2 3 4.. whatever the size of str2 string
  // first row should be initialize with 0 1 2 3 .. whatever the size of str1 string

  //this code will fill the matrix.
  for(int i=0; i<=str2.size(); i++){
    for(int j=0; j<=str1.size(); j++){
      operationTable[i][j] = j;
    }
    operationTable[i][0] = i;
  }

  for(int i=1; i<=str2.size(); i++){
    for(int j=1; j<=str1.size(); j++){
      if(str2[i-1]==str1[j-1])
        operationTable[i][j] = operationTable[i-1][j-1];
      else
        operationTable[i][j]=1+ min(operationTable[i-1][j-1], min(operationTable[i][j-1], operationTable[i-1][j]));
    }
  }

  return operationTable[str2.size()][str1.size()];

}

```

31

11 August 2023 07:47 PM

Difficulty: **Category:** **Successful Submissions:** 19,673+

Number Of Ways To Traverse Graph

You're given two positive integers representing the width and height of a grid-shaped, rectangular graph. Write a function that returns the number of ways to reach the bottom right corner of the graph when starting at the top left corner. Each move you take must either go down or right. In other words, you can never move up or left in the graph.

For example, given the graph illustrated below, with `width = 2` and `height = 3`, there are three ways to reach the bottom right corner when starting at the top left corner:

```
 _ _
|_|_|
|_|_|
|_|_|
```

1. Down, Down, Right
2. Right, Down, Down
3. Down, Right, Down

Note: you may assume that `width * height >= 2`. In other words, the graph will never be a 1x1 grid.

Sample Input

```
width = 4
height = 3
```

Sample Output

```
10
```

Solution 1 Solution 2 Solution 3

```cpp
using namespace std;
// 1st approach is recursive approach.
// where we start from end point and go left and up.
// the box should look like this.
// 1,1  1,2   1,3    so 4,3 is end point we are starting from there
// 2,1  2,2   2,3    and when we move left width is decreasing by 1
// 3,1  3,2   3,3    height remains same. then height decreases, width
// 4,1  4,2   4,3    remains same on moving up.

// the base case is when height or width is 1 return 1.
// as you can see to come at 1,2 there is only 1 way from 1,1
// and same for 2,1 only 1 way from 1,1
// but this algorithm takes O(2^(n+m)) time which is not the optimal
// solution, approach 2 is in solution 2.
int numberOfWaysToTraverseGraph(int width, int height) {
  // Write your code here.
  if(width==1 || height==1)
    return 1;
  return numberOfWaysToTraverseGraph(width-1, height) + //to left from end
         numberOfWaysToTraverseGraph(width, height-1); //to top

  //we are adding both cause to reach at 4,3 the max ways to reach there is
  // from 3,3 and 4,2. similarly to reach at 4,2 2 ways are 4,1 and 3,2.....
}

// this algorithm might also break for larger test cases.
```

Solution 1 **Solution 2** Solution 3

```cpp
using namespace std;
// the next approach is top down dynamic programming(memoization) , where
// we should store the intermediate results in a 2d array and before making
// a call we are checking if that result is already there or not, if not store it
// move to the next call.

int graphTraverseHelper(int width, int height, vector<vector<int>> &v){
  if(width==1 || height==1)
    return 1;
  if(v[width][height]==0){
    v[width][height] = graphTraverseHelper(width-1, height, v) +graphTraverseHelper(width, height-1, v);
  }

  return v[width][height];
}


int numberOfWaysToTraverseGraph(int width, int height) {
  // Write your code here.
  vector<vector<int>> vect(width+1, vector<int>(height+1,0));
   return graphTraverseHelper(width, height, vect);
  return -1;
}
```

32

11 August 2023 07:47 PM

Difficulty: **Category:** **Successful Submissions:** 32,979+

Kadane's Algorithm

Write a function that takes in a non-empty array of integers and returns the maximum sum that can be obtained by summing up all of the integers in a non-empty subarray of the input array. A subarray must only contain adjacent numbers (numbers next to each other in the input array).

Sample Input

```
array = [3, 5, -9, 1, 3, -2, 3, 4, 7, 2, -9, 6, 3, 1, -5, 4]
```

Sample Output

```
19 // [1, 3, -2, 3, 4, 7, 2, -9, 6, 3, 1]
```

Solution 1 Solution 2 Solution 3

```cpp
#include <vector>
using namespace std;

int kadanesAlgorithm(vector<int> array) {
  // Write your code here.
  // we are storing max sum for each index[adjacent] as we want
  // sum from adjacent subarray and check if that sum is greater
  // than array[i] then push sum else push array[i]. and storing it in temp
  // and for each value we will be checking if new sum is greater than
  // the one in max variable then we will update it.
  vector<int> temporaryMaxSum;

  if(array.size()==0)
    return 0;
  if(array.size()==1)
    return array[0];
  temporaryMaxSum.push_back(array[0]);
  int i=1;
  int maxSum = array[0];
  while(i<array.size()){
    int sumAt_i = array[i] + temporaryMaxSum[i-1]; //adjacent sum

    if(array[i] > sumAt_i)
      temporaryMaxSum.push_back(array[i]);
    else
      temporaryMaxSum.push_back(sumAt_i);

    if(temporaryMaxSum[i] > maxSum)
      maxSum = temporaryMaxSum[i];
    i++;
  }

  cout << maxSum;
  return maxSum;
}
```

33.1

11 August 2023 07:47 PM

Stable internships

A company has hired N interns to each join one of N different teams. Each intern has ranked their preferences for which teams they wish to join, and each team has ranked their preferences for which interns they prefer.

Given these preferences, assign 1 intern to each team. These assignments should be "stable," meaning that there is no unmatched pair of an intern and a team such that both that intern and that team would prefer they be matched with each other.

In the case there are multiple valid stable matchings, the solution that is most optimal for the interns should be chosen (i.e. every intern should be matched with the best team possible for them).

Your function should take in 2 2-dimensional lists, one for interns and one for teams. Each inner list represents a single intern or team's preferences, ranked from most preferable to least preferable. These lists will always be of length N, with integers as elements. Each of these integers corresponds to the index of the team/intern being ranked. Your function should return a 2-dimensional list of matchings in no particular order. Each matching should be in the format [internIndex, teamIndex].

Sample Input

```
interns = [
  [0, 1, 2],
  [1, 0, 2],
  [1, 2, 0]
]
```

```
teams = [
  [2, 1, 0],
  [1, 2, 0],
  [0, 2, 1]
]
```

Sample Output

```
// This is the most optimal solution for interns
[
  [0, 0],
  [1, 1],
  [2, 2]
]
```

```
// APPROACH : we will store all the interns in stack and pop out one at a time and will match his preferred team and increase his
// intern choices , then will go for the next intern and will check if team is already occupied or not.
// if not then we will let him match his team else we will check from team maps that which intern is
// more preferred by the team, then we will swap the team if required and will push
// that intern into intern stack again. the process will run till stack is empty
//O(n^2) time and space where n is the no. of interns and teams.
```

33.2

11 August 2023 08:05 PM

Solution 1 Solution 2 Solution 3

```cpp
#include <vector>
#include<stack>
#include <unordered_map>
using namespace std;
vector<vector<int>> stableInternships(vector<vector<int>> interns, vector<vector<int>> teams) {
unordered_map<int, int> result;
  stack<int> totalInterns;
  for(int i=0; i<interns.size(); i++ )
    totalInterns.push(i);
  // to keep a check on intern choice of team. if the value is 0, it means team 0
  // if 1 team 1 is preferred.
  vector<int> current_internChoices(interns.size(), 0);
  vector<unordered_map<int, int>> teamPreferences;
  for(vector<int> team : teams){
    unordered_map<int, int> ranks;
    for(int i=0; i<teams.size(); i++){
      ranks[team[i]] = i;
    }
    teamPreferences.push_back(ranks);
  }
  while(!totalInterns.empty()){
    int internPresent = totalInterns.top();
    totalInterns.pop();
    vector<int> intern = interns[internPresent]; //as this is a vector so storing it in vector
    int teamPreference = intern[current_internChoices[internPresent]]; // will give the team preference.
    current_internChoices[internPresent]++;
    if(result.find(teamPreference)==result.end()){ //means team is not already there.
        result[teamPreference] = internPresent;
        continue; // if continue is encountered the remaining code will not be executed, the loop will proceed to the next iteration.
    }
    int previousIntern = result[teamPreference]; // if its team has to be changed then we can push it in the stack again.
    int previousInternRank = teamPreferences[teamPreference][previousIntern];
    //teamPrefrences[teamPreference] is a map, and we are passing previousIntern
    // as key to find out the the rank of that intern in its prefered team.
    int currentInternRank =teamPreferences[teamPreference][internPresent];

    if(currentInternRank < previousInternRank){
      totalInterns.push(previousIntern);
      result[teamPreference] = internPresent;
    }
    else{
      totalInterns.push(internPresent);
    }
  }

  vector<vector<int>> matches; //as we need to return 2d array as result.
  for(auto intern : result){
    matches.push_back({intern.second , intern.first});
  }


  return matches;

}
```

34

11 August 2023 07:47 PM

Union Find

The union-find data structure is similar to a traditional set data structure in that it contains a collection of unique values. However, these values are spread out amongst a variety of distinct disjoint sets, meaning that no set can have duplicate values, and no two sets can contain the same value.

Write a `UnionFind` class that implements the union-find (also called a disjoint set) data structure. This class should support three methods:

- `createSet(value)` : Adds a given value in a new set containing only that value.
- `union(valueOne, valueTwo)` : Takes in two values and determines which sets they are in. If they are in different sets, the sets are combined into a single set. If either value is not in a set or they are in the same set, the function should have no effect.
- `find(value)` : Returns the "representative" value of the set for which a value belongs to. This can be any value in the set, but it should always be the same value, regardless of which value in the set `find` is passed. If the value is not in a set, the function should return `null` / `None`. Note that after a set is part of a union, its representative can potentially change.

You can assume `createSet` will never be called with the same value twice.

If you're unfamiliar with Union Find, we recommend watching the Conceptual Overview section of this question's video explanation before starting to code.

```
//APPROACH : creating a value is just insert into parent map and each value will be its parent
// initially. find : find till the number is a parent to itself. In union, first we have to
// find their representative basically parent, if they are diff. we check their ranks and
// based on that we make representative of the complete set.
// when a value is created using create function, its rank is set to 0. the increment and
// choosing of parent is done on the basis of the 3 cond listed in if block.
// optimal solution, time O(alpha(n)), alpha is inverse acronym function which value is less than 4 on any input.
// Do not edit the class below except for
// the constructor, push, pop, peek, and
// isEmpty methods. Feel free to add new
// properties and methods to the class.
```

Solution 1 | Solution 2 | Solution 3

```cpp
#include<unordered_map>
#include <optional>
using namespace std;

class UnionFind {
unordered_map<int, int> parents;
unordered_map<int, int> ranks;
public:
  void createSet(int value) {
    // Write your code here.
    parents[value] = value; //making parent of itself
    ranks[value] = 0;  //at creation rank will be 0
  }
  optional<int> find(int value) {
    // Write your code here.
    if(parents.find(value)==parents.end())
      return nullopt;
    int currentParent = value;
    while(currentParent != parents[currentParent])
      currentParent = parents[currentParent];

    return currentParent;
  }
  void createUnion(int valueOne, int valueTwo) {
    //base case
    if(parents.find(valueOne)==parents.end() || parents.find(valueTwo)==parents.end())
      return ;
    // Write your code here.
    int valueOneRoot = *find(valueOne);     // deferencing the function. calling the function at runtime.
    int valueTwoRoot = *find(valueTwo);
    if(ranks[valueOneRoot] > ranks[valueTwoRoot])
      parents[valueTwoRoot] = valueOneRoot;
    else if(ranks[valueOneRoot] < ranks[valueTwoRoot])
      parents[valueOneRoot] = valueTwoRoot;
    else{
      parents[valueTwoRoot] = valueOneRoot;    //ranks are equal, we are choosing one value to the parent in this case.
      ranks[valueOneRoot] = ranks[valueOneRoot] +1;
    }

  }
```

35

11 August 2023 07:47 PM

Difficulty: **Category:** **Successful Submissions:** 34,101+

Single Cycle Check

You're given an array of integers where each integer represents a jump of its value in the array. For instance, the integer `2` represents a jump of two indices forward in the array; the integer `-3` represents a jump of three indices backward in the array.

If a jump spills past the array's bounds, it wraps over to the other side. For instance, a jump of `-1` at index `0` brings us to the last index in the array. Similarly, a jump of `1` at the last index in the array brings us to index `0`.

Write a function that returns a boolean representing whether the jumps in the array form a single cycle. A single cycle occurs if, starting at any index in the array and following the jumps, every element in the array is visited exactly once before landing back on the starting index.

Sample Input

```
array = [2, 3, 1, -4, -4, 2]
```

Sample Output

```
true
```

Prompt **Scratchpad** **Solutions** **Video Explanation**

```
//APPROACH: we discover the pattern that i[0] == visited[last-1]
// to complete a cycle, this condition is not alone to claim that
// as in some cases, the cycle was repeated b/w 2 values only.
// so other condition to claim a cycle is to be no duplicates
// in visited array. if both cond are true, single cycle is true.
// we also convert the outside bound index using formula used in code.
// we didn't use only 1, as modulo works diff for +ve and -ve.
```

Solution 1 **Solution 2** **Solution 3**

```cpp
#include<vector>
#include<set>
using namespace std;
void helperFn(vector<int> &ie, vector<int> &visited, vector<int> arr){
    int currentIndex =0 ;
    int jumpIndex;
    int arraySize = arr.size();
    for(int i=0; i<arr.size();i++ ){
    int jump = currentIndex + arr[currentIndex] + arraySize;
    if(jump<0)
      jumpIndex = (jump%arraySize + arraySize)%arraySize;
    else
      jumpIndex = jump%arraySize;

    ie.push_back(currentIndex);
    visited.push_back(jumpIndex);
    currentIndex = jumpIndex;
  }
}

bool hasSingleCycle(vector<int> array) {
  // Write your code here.
  int arrSize = array.size();
  vector<int> i;
  vector<int> visited;
  helperFn(i, visited, array);
  bool duplicatesInVisited = false;
  set<int> temp(visited.begin(), visited.end());

  if(temp.size()< visited.size())
    duplicatesInVisited= true;
  else
    cout << "no";

  if(i[0]==visited[arrSize-1] && duplicatesInVisited==false)
    return true;
  else
    return false;
}
```

36

11 August 2023 07:47 PM

Difficulty: **Category:** **Successful Submissions:** 39,326+

Breadth-first Search

You're given a `Node` class that has a `name` and an array of optional `children` nodes. When put together, nodes form an acyclic tree-like structure.

Implement the `breadthFirstSearch` method on the `Node` class, which takes in an empty array, traverses the tree using the Breadth-first Search approach (specifically navigating the tree from left to right), stores all of the nodes' names in the input array, and returns it.

If you're unfamiliar with Breadth-first Search, we recommend watching the Conceptual Overview section of this question's video explanation before starting to code.

Sample Input

```
graph = A
      /  |  \
     B   C   D
    / \     / \
   E   F   G   H
      / \   \
     I   J   K
```

Sample Output

```
["A", "B", "C", "D", "E", "F", "G", "H", "I", "J", "K"]
```

Solution 1 | Solution 2 | Solution 3

```cpp
#include <vector>
#include <queue>
using namespace std;

// Do not edit the class below except
// for the breadthFirstSearch method.
// Feel free to add new properties
// and methods to the class.
class Node {
public:
  string name;
  vector<Node *> children;
  Node(string str) { name = str; }
  vector<string> breadthFirstSearch(vector<string> *array) {
    // Write your code here.
    queue<Node*> q;   // q which takes a pointer to the nodes.
    Node* currentNode = this;  //this points to the starting node.
    q.push(currentNode);
    array->push_back(currentNode->name);  // in array we pushing the name
    //that pointer is pointing to

    while(!q.empty()){
      currentNode = q.front();    // it gives back the address of the old element
      q.pop();       // pops the oldest.
      for(auto child : currentNode->children){   // for every child in children
        array->push_back(child->name);           //array of current Node pointing to
        q.push(child);
      }
    }
    return *array;       // returning the address of the array, it points to first element of array.
  }

  Node *addChild(string name) {
    Node *child = new Node(name);
    children.push_back(child);
    return this;
  }
};
```

37

11 August 2023 07:47 PM

Difficulty: **Category:** **Successful Submissions:** 37,574+

River Sizes

You're given a two-dimensional array (a matrix) of potentially unequal height and width containing only `0`s and `1`s. Each `0` represents land, and each `1` represents part of a river. A river consists of any number of `1`s that are either horizontally or vertically adjacent (but not diagonally adjacent). The number of adjacent `1`s forming a river determine its size.

Note that a river can twist. In other words, it doesn't have to be a straight vertical line or a straight horizontal line; it can be L-shaped, for example.

Write a function that returns an array of the sizes of all rivers represented in the input matrix. The sizes don't need to be in any particular order.

Sample Input

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

Sample Output

```
[1, 2, 2, 2, 5] // The numbers could be ordered differently.
```

```
// APPROACH : traverse the matrix and when you hit a 1, run dfs or bfs
// on its neighbours nodes(above, below, right, left) if they are 1 and
mark them true in the
// auxiliary visited matrix so that its easy to track them.
// or we  can directly edit the original matrix and mark the current node
// as -1 as to mark them visited, this will save our space..
// graph question common thing is visited array or matrix.
// In DFS fn check the indexes are in range, not outside the matrix using
i and j index.
// check if that is 1 again as we need that for neighbouring nodes too.
// keep track of size. when dfs is over send size inside array.
```

Solution 1 | Solution 2 | Solution 3

```cpp
#include <vector>
using namespace std;

void DFS(vector<vector<int>> &m, int x, int y, int &size){
  int height = m.size(); int width = m[0].size();
  if((x>=0 && x<height) && (y>=0 && y<width)){
    if(m[x][y]==1){
      m[x][y]=-1;
      size++;
      DFS(m, x-1, y ,size); // above
      DFS(m, x+1, y, size); //below
      DFS(m, x, y-1, size); //left
      DFS(m, x, y+1, size); //right
    }
  }
}


vector<int> riverSizes(vector<vector<int>> matrix) {
  // Write your code here.
  vector<int> result;
  int height = matrix.size();
  int width = matrix[0].size();

  for(int i=0; i<height; ++i){
    for(int j=0; j<width; ++j){
      int val = matrix[i][j];
      if(val==1){
        int size=0;
        DFS(matrix, i, j, size);
        result.push_back(size);
      }
    }
  }


  return result;
}
```

38

11 August 2023 07:47 PM

Difficulty: **Category:** **Successful Submissions:** 24,330+

Youngest Common Ancestor

You're given three inputs, all of which are instances of an `AncestralTree` class that have an `ancestor` property pointing to their youngest ancestor. The first input is the top ancestor in an ancestral tree (i.e., the only instance that has no ancestor--its `ancestor` property points to `None` / `null`), and the other two inputs are descendants in the ancestral tree.

Write a function that returns the youngest common ancestor to the two descendants.

Note that a descendant is considered its own ancestor. So in the simple ancestral tree below, the youngest common ancestor to nodes A and B is node A.

```
// The youngest common ancestor to nodes A and B is node A.
  A
 /
B
```

Sample Input

```
// The nodes are from the ancestral tree below.
topAncestor = node A
descendantOne = node E
descendantTwo = node I
          A
        /   \
       B     C
     /  \   /  \
    D    E F    G
  /  \
 H    I
```

Sample Output

```
node B
```

```
1  //APPROACH: we are just running recursion and pushing back ancestors
   address
2  // of descedant 1 and 2 separately in 2 vectors, and as each descendant is
   an
3  // ancestor of itself. the first common occuring address from both
4  // vector points to the youngest ancestor of the 2 descendants.
5  // we use set to find the common from the 2 vectors.
```

Solution 1 | Solution 2 | Solution 3

```
1  #include <vector>
2  #include<set>
3  using namespace std;
4  class AncestralTree {
5  public:
6    char name;
7    AncestralTree *ancestor;
8    AncestralTree(char name) {
9      this->name = name;
10     this->ancestor = nullptr;
11   }
12   void addAsAncestor(vector<AncestralTree *> descendants);
13 };
14 void ances(AncestralTree* value, vector<AncestralTree*>&D){
15   auto value_Ances = value;
16   D.push_back(value_Ances);
17   if(value->ancestor!=nullptr){
18     auto res = value->ancestor;
19     ances(res, D);
20   }
21 }
22 AncestralTree* findCommon(vector<AncestralTree*> v1, vector<AncestralTree*> v2){
23   set<AncestralTree*> s;
24   for(auto i : v1)
25     s.insert(i);
26   for(auto i:v2){
27     if(s.count(i)>0)
28       return i;
29   }
30   return nullptr;
31 }
32 AncestralTree *getYoungestCommonAncestor(AncestralTree *topAncestor,
33                                          AncestralTree *descendantOne,
34                                          AncestralTree *descendantTwo) {
35   // Write your code here.
36   vector<AncestralTree*> d1;
37   vector<AncestralTree*> d2;
38   ances(descendantTwo, d1);
39   ances(descendantOne, d2);
40   return findCommon(d1, d2);
41 }
```

39

11 August 2023 07:47 PM

Remove Islands

You're given a two-dimensional array (a matrix) of potentially unequal height and width containing only `0`s and `1`s. The matrix represents a two-toned image, where each `1` represents black and each `0` represents white. An island is defined as any number of `1`s that are horizontally or vertically adjacent (but not diagonally adjacent) and that don't touch the border of the image. In other words, a group of horizontally or vertically adjacent `1`s isn't an island if any of those `1`s are in the first row, last row, first column, or last column of the input matrix.

Note that an island can twist. In other words, it doesn't have to be a straight vertical line or a straight horizontal line; it can be L-shaped, for example.

You can think of islands as patches of black that don't touch the border of the two-toned image.

Write a function that returns a modified version of the input matrix, where all of the islands are removed. You remove an island by replacing it with `0`s.

Naturally, you're allowed to mutate the input matrix.

Sample Input

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

Sample Output

```
[
  [1, 0, 0, 0, 0, 0],
  [0, 0, 0, 1, 1, 1],
  [0, 0, 0, 0, 1, 0],
  [1, 1, 0, 0, 1, 0],
```

Solution 1 Solution 2 Solution 3

```cpp
#include <vector>
using namespace std;
void imageInAuxMatrix(vector<vector<int>>matrix, int row, int col, vector<vector<int>> &auxi){
  int rows = matrix.size(); int cols = matrix[0].size();
  if((row>=0 && row<rows) && (col>=0 && col<cols)){
    if(matrix[row][col] == 1 && auxi[row][col]==0){ // checking 0 so to ensure this is not visited already
      auxi[row][col] = 1;
      // run for neighbors of border and if they are 1, means they are touching the border
      // so they won't make an island.
      imageInAuxMatrix(matrix, row+1, col,auxi);
      imageInAuxMatrix(matrix, row-1, col, auxi);
      imageInAuxMatrix(matrix, row, col+1, auxi);
      imageInAuxMatrix(matrix, row, col-1, auxi);
    }
  }}
vector<vector<int>> removeIslands(vector<vector<int>> matrix) {// Write your code here.
  int rows = matrix.size();
  int cols = matrix[0].size();
  vector<vector<int>> auxMatrix(rows, vector<int>(cols, 0));
  for(int i=0; i<rows; i++){
    if(matrix[i][0]==1)
      imageInAuxMatrix(matrix, i,0 , auxMatrix);
    if(matrix[i][cols-1]==1)
      imageInAuxMatrix(matrix, i, cols-1, auxMatrix);
  }
  for(int i=1; i<cols; i++){
    if(matrix[rows-1][i]==1)
      imageInAuxMatrix(matrix, rows-1, i, auxMatrix);
    if(matrix[0][i]==1)
      imageInAuxMatrix(matrix, 0, i, auxMatrix);
  }
  // now auxMatrix contains 0 at the place of 1 in interior of matrix, so we will run loop inside the
  // interior and check if any 1 is encountered we will check in auxiMatrix at same
  // location if 0 is there then that 1 is island and change it to 0.
  for(int i=1; i<rows-1; i++){
    for(int j=1; j<cols-1; j++){
      if(matrix[i][j]==1 && auxMatrix[i][j]==0)
        matrix[i][j] = 0;
    }
  }
  return matrix;
```

```cpp
//APPROACH : the brute force is loop through the 2d matrix, when you encounter 1
// run a graph traversal and check if any of the neighbours are touching the border 1.
// if yes that is not an island.

// solution Approach : we traverse 1st, last row and 1st, last col
// for every 1 is detected in main matrix, we run DFS on that and check the neighbours
// if they are 1, means that is connected to border 1 so they also don't make island.
// and run DFS again and mark 1 at the same location in auxiliary matrix.
// and when all border is over in auxi matrix all the borders 1
// and 1 connected to those 1 is 1 in that matrix and then we iterate
// the inner of matrix and if 1 is encountered and for same pos
// auxiliary matrix is 0, that is an island mark it 0 in main matrix.
```

40.1

11 August 2023 07:47 PM

Cycle In Graph

You're given a list of `edges` representing an unweighted, directed graph with at least one node. Write a function that returns a boolean representing whether the given graph contains a cycle.

For the purpose of this question, a cycle is defined as any number of vertices, including just one vertex, that are connected in a closed chain. A cycle can also be defined as a chain of at least one vertex in which the first vertex is the same as the last.

The given list is what's called an adjacency list, and it represents a graph. The number of vertices in the graph is equal to the length of `edges`, where each index `i` in `edges` contains vertex `i`'s outbound edges, in no particular order. Each individual edge is represented by a positive integer that denotes an index (a destination vertex) in the list that this vertex is connected to. Note that these edges are directed, meaning that you can only travel from a particular vertex to its destination, not the other way around (unless the destination vertex itself has an outbound edge to the original vertex).

Also note that this graph may contain self-loops. A self-loop is an edge that has the same destination and origin; in other words, it's an edge that connects a vertex to itself. For the purpose of this question, a self-loop is considered a cycle.

For a more detailed explanation, please refer to the Conceptual Overview section of this question's video explanation.

Sample Input

```
edges = [
  [1, 3],
  [2, 3, 4],
  [0],
  [],
  [2, 5],
  [],
]
```

Sample Output

```
true
// There are multiple cycles in this graph:
// 1) 0 -> 1 -> 2 -> 0
// 2) 0 -> 1 -> 4 -> 2 -> 0
// 3) 1 -> 2 -> 0 -> 1
// These are just 3 examples; there are more.
```

```
/APPROACH : we run DFS on each individual vertex. what that means is, we pass a
// vertex, that's the starting node. a cycle is there suppose from 0 we reach to 1,
// from 1 we reach to 2, then from 2 to 0 so its a cycle.
// so we push the current vertex in queue and mark it visited in the set.
// then we run a loop till queue is not empty, we took out a node, and then
// for every outbound edge from it we check if its equal to startnode, if yes its a cycle
// otherwise we check in the set if that is visited or not, if not we push in the queue and
// and do the same process. if queue is empty and then it returns false

// this fn is for individual vertex. means when 2nd vertex will be passed, all vector
// and queues will be reset.
```

40.2

11 August 2023 08:24 PM

Solution 1 Solution 2 Solution 3

```cpp
#include <vector>
#include <queue>
#include <set>
using namespace std;
bool DFS(vector<vector<int>> edges, int vertex){
  int startNode = vertex;
  queue<int> que;
  que.push(vertex);
  set<int> visited;
  while(!que.empty()){
    int curr = que.front();
    que.pop();

    visited.insert(curr);

    for(int edge : edges[curr]){
      if(edge ==startNode)
        return true;
      if(visited.find(edge)==visited.end())
        que.push(edge);
    }
  }

  return false;

  }

bool cycleInGraph(vector<vector<int>> edges) {
  // Write your code here.
  //sending each vertex
  for(int v=0; v<edges.size(); v++){
    bool res = DFS(edges, v);
    if(res)
      return true;
  }
  return false;
}
```

Solution 1 Solution 2 Solution 3

```cpp
#include <vector>
#include <queue>
#include <set>
using namespace std;
// this is an edit of first solution, in this we are using vector array instead of set.
bool DFS(vector<vector<int>> edges, int vertex){
  int startNode = vertex;
  queue<int> que;
  que.push(vertex);
  vector<int>visited(10, 0);
  while(!que.empty()){
    int curr = que.front();
    que.pop();

    visited[curr] = 1;

    for(int edge : edges[curr]){
      if(edge ==startNode)
        return true;
      if(visited[edge]==0)
        que.push(edge);
    }
  }

  return false;

  }

bool cycleInGraph(vector<vector<int>> edges) {
  // Write your code here.
  //sending each vertex
  for(int v=0; v<edges.size(); v++){
    bool res = DFS(edges, v);
    if(res)
      return true;
  }
  return false;
}
```

41.1

11 August 2023 07:47 PM

Minimum Passes Of Matrix

Write a function that takes in an integer matrix of potentially unequal height and width and returns the minimum number of passes required to convert all negative integers in the matrix to positive integers.

A negative integer in the matrix can only be converted to a positive integer if one or more of its adjacent elements is positive. An adjacent element is an element that is to the left, to the right, above, or below the current element in the matrix. Converting a negative to a positive simply involves multiplying it by `-1`.

Note that the `0` value is neither positive nor negative, meaning that a `0` can't convert an adjacent negative to a positive.

A single pass through the matrix involves converting all the negative integers that *can* be converted at a particular point in time. For example, consider the following input matrix:

```
[
  [0, -2, -1],
  [-5, 2, 0],
  [-6, -2, 0],
]
```

After a first pass, only 3 values can be converted to positives:

```
[
  [0, 2, -1],
  [5, 2, 0],
  [-6, 2, 0],
]
```

After a second pass, the remaining negative values can all be converted to positives:

```
[
  [0, 2, 1],
  [5, 2, 0],
  [6, 2, 0],
]
```

Note that the input matrix will always contain at least one element. If the negative integers in the input matrix can't all be converted to positives, regardless of how many passes are run, your function should return `-1`.

Sample Input

```
matrix = [
  [0, -1, -3, 2, 0],
  [1, -2, -5, -1, -3],
  [3, 0, 0, -4, -1],
]
```

Sample Output

```
3
```

41.2

11 August 2023 08:27 PM

Solution 1 Solution 2 Solution 3

```cpp
#include <vector>
#include <queue>
#include <utility>
#include <iostream>
using namespace std;
// these are the 4 adjacent to any cell when adding to current cell.
int x[4] = {-1, 0, 0, 1};
int y[4] = {0, -1, 1, 0};

void helperMatr(vector<vector<int>>&matrix, queue<pair<int,int>> &Q, int &pass){
    int rows = matrix.size();
    int cols = matrix[0].size();
    while(!Q.empty()){
        queue<pair<int,int>> q;
        swap(Q,q);
        while(!q.empty()){
            int row = q.front().first;
            int col = q.front().second;
            q.pop();
            for(int k=0; k<4; k++){
                if((row+x[k]>=0 && row+x[k]<rows) && (col+y[k]>=0 && col+y[k]<cols)){
                    if(matrix[row+x[k]][col+ y[k]] < 0){
                        matrix[row + x[k]][col+y[k]] *= -1;
                        pair<int,int> pp = make_pair(row+x[k], col+y[k]);
                        Q.push(pp);
                    }
                }
            }
        }
        pass++;
    }
}
```

```cpp
int minimumPassesOfMatrix(vector<vector<int>> matrix) {
  // Write your code here.
  int rows = matrix.size();
  int cols = matrix[0].size();
  queue<pair<int,int>> Q;
  //pushing every positive value's position of matrix  in queue.


  for(int i=0; i<rows; i++){
    for(int j=0; j<cols; j++){
      if(matrix[i][j]>0){
        pair<int, int> p = make_pair(i,j);
        Q.push(p);
      }
    }
  }


  int pass = -1;
  helperMatr(matrix, Q, pass);
  cout << pass;

  // checking if there is still negative after running the fun
  // if yes then we know no amount of passes can convert that negative to +ve.
  // so return -1
  for(int i=0; i<rows; i++){
   for(int j=0; j<cols; j++){
     if(matrix[i][j] < 0)
       return -1;
   }
  }
  return pass;
}
```

42

11 August 2023 08:28 PM

Two-Colorable

You're given a list of `edges` representing a connected, unweighted, undirected graph with at least one node. Write a function that returns a boolean representing whether the given graph is two-colorable.

A graph is two-colorable (also called bipartite) if all of the nodes can be assigned one of two colors such that no nodes of the same color are connected by an edge.

The given list is what's called an adjacency list, and it represents a graph. The number of vertices in the graph is equal to the length of `edges`, where each index `i` in `edges` contains vertex `i`'s siblings, in no particular order. Each individual edge is represented by a positive integer that denotes an index in the list that this vertex is connected to. Note that this graph is undirected, meaning that if a vertex appears in the edge list of another vertex, then the inverse will also be true.

Also note that this graph may contain self-loops. A self-loop is an edge that has the same destination and origin; in other words, it's an edge that connects a vertex to itself. Any self-loop should make a graph not 2-colorable.

Sample Input

```
edges = [
  [1, 2],
  [0, 2],
  [0, 1]
]
```

Sample Output

```
      // Nodes 1 and 2 must be different colors than node 0.
      // However, nodes 1 and 2 are also connected, meaning they must a
False // which is impossible with only 2 available colors.
```

```
// APPROACH : we manually push a first pair of vertex with color
// then we pop out that store it, and then check the vertex for its
// connecting edges as they should not have same color. for this
// we take a visited array of same size and for every vertex we run through
// its connecting edges ffrom list and check if current connecting edge
// is =='@' means its not visited and its color is not been assigned
// then we assign color that is opposite to the current vertex we do for
// all the connecting edges, but if some edge is already visited
// and its color is equal to that of current vertex then two colorable is
not possible
// if the loop runs without false, then the graph is two colorable.
```

Solution 1 | Solution 2 | Solution 3

```cpp
#include<vector>
#include<queue>
#include<utility>
using namespace std;
bool twoColorable(vector<vector<int>> edges) {
  // Write your code here.
  if(edges.size()==0 || edges.size()==1)
    return false;
  if(edges.size()==2)
    return true;
  vector<char> vertices(edges.size(), '@');
  queue<pair<int, char>> q;
  q.push({0, 'B'});
  vertices[0]='B';

  while(!q.empty()){
    auto x = q.front();
    q.pop();
    int vertex = x.first;
    char color = x.second;

    for(auto p: edges[vertex]){
      if(vertices[p]=='@'){
        if(color=='B')
          vertices[p]='W';
        else
          vertices[p]='B';
        q.push({p, vertices[p]});
      }
      else if(vertices[p]==color)
        return false;
      }
  }


    return true;
}
```

43

11 August 2023 08:28 PM

Difficulty: **Category:** **Successful Submissions:** 15,435+

Task Assignment

You're given an integer `k` representing a number of workers and an array of positive integers representing durations of tasks that must be completed by the workers. Specifically, each worker must complete two unique tasks and can only work on one task at a time. The number of tasks will always be equal to `2k` such that each worker always has exactly two tasks to complete. All tasks are independent of one another and can be completed in any order. Workers will complete their assigned tasks in parallel, and the time taken to complete all tasks will be equal to the time taken to complete the longest pair of tasks (see the sample output for an explanation).

Write a function that returns the optimal assignment of tasks to each worker such that the tasks are completed as fast as possible. Your function should return a list of pairs, where each pair stores the indices of the tasks that should be completed by one worker. The pairs should be in the following format: `[task1, task2]`, where the order of `task1` and `task2` doesn't matter. Your function can return the pairs in any order. If multiple optimal assignments exist, any correct answer will be accepted.

Note: you'll always be given at least one worker (i.e., `k` will always be greater than **0**).

Sample Input

```
k = 3
tasks = [1, 3, 5, 3, 1, 4]
```

Sample Output

```
[
  [0, 2], // tasks[0] = 1, tasks[2] = 5 | 1 + 5 = 6
  [4, 5], // tasks[4] = 1, tasks[5] = 4 | 1 + 4 = 5
  [1, 3], // tasks[1] = 3, tasks[3] = 3 | 3 + 3 = 6
] // The fastest time to complete all tasks is 6.
```

Solution 1 **Solution 2** **Solution 3**

```cpp
#include <vector>
#include<utility>
#include<unordered_map>
using namespace std;
// APPROACH : the fastest time to complete all tasks is the max duration + min duration
// of task then the rest will be paired accordingly. so we send all the tasks with
// their indices as pair in a vector, then we sort the vector pair indices of
// start with last (with start++, last--). in this we the least duration was
// paired with high durations to done the tasks faster.

vector<vector<int>> taskAssignment(int k, vector<int> tasks) {
  // Write your code here
  vector<pair<int,int>> temp;

  for(int i=0; i<tasks.size(); i++){
    temp.push_back({i, tasks[i]});
  }

  sort(temp.begin(), temp.end(), [](const auto&x, const auto&y){
    return x.second < y.second;
  });

  vector<vector<int>>res;

  int start=0;
  int last=tasks.size()-1;
  while(start<last){
    res.push_back({temp[start].first, temp[last].first});
    start++;
    last--;
  }
   for(auto x: res)
    cout << x[0] << " " << x[1] <<endl;

  return res;
}
```

44.1

11 August 2023 08:28 PM

Valid Starting City

Imagine you have a set of cities that are laid out in a circle, connected by a circular road that runs clockwise. Each city has a gas station that provides gallons of fuel, and each city is some distance away from the next city.

You have a car that can drive some number of miles per gallon of fuel, and your goal is to pick a starting city such that you can fill up your car with that city's fuel, drive to the next city, refill up your car with that city's fuel, drive to the next city, and so on and so forth until you return back to the starting city with 0 or more gallons of fuel left.

This city is called a valid starting city, and it's guaranteed that there will always be exactly one valid starting city.

For the actual problem, you'll be given an array of distances such that city `i` is `distances[i]` away from city `i + 1`. Since the cities are connected via a circular road, the last city is connected to the first city. In other words, the last distance in the `distances` array is equal to the distance from the last city to the first city. You'll also be given an array of fuel available at each city, where `fuel[i]` is equal to the fuel available at city `i`. The total amount of fuel available (from all cities combined) is exactly enough to travel to all cities. Your fuel tank always starts out empty, and you're given a positive integer value for the number of miles that your car can travel per gallon of fuel (miles per gallon, or MPG). You can assume that you will always be given at least two cities.

Write a function that returns the index of the valid starting city.

Sample Input

```
distances = [5, 25, 15, 10, 15]
fuel = [1, 2, 1, 0, 3]
mpg = 10
```

Sample Output

```
4
```

Solution 1 | Solution 2 | Solution 3

```cpp
#include <vector>
using namespace std;
// APPROACH : we take first city  as valid, milesLeft for car is
// equal to fuel *mpg. so if milesleft >= distance we can travel to
// next pump,if greater then miles left is left with milesleft - distance
// then we check how much next city gas station can give then we
// add those miles to milesLeft. if any case milesleft is less that
// distance that means that city is not valid, update it.

int validStartingCity(vector<int> distances, vector<int> fuel, int mpg) {
  // Write your code here.

  int validCity =0;
  int milesLeft = 0;
  int  i=0;

  while(i<distances.size()){
    milesLeft += mpg * fuel[i];  //updating milesLeft for that particular valid city.
    if(milesLeft >= distances[i])  //means enough petrol to cover that distance
      milesLeft -= distances[i];    // update milesLEft
    else{
      validCity = i + 1;          //start with new city.
      milesLeft=0;    //reset
    }

    i++;
  }

  return validCity;
}
```

44.2

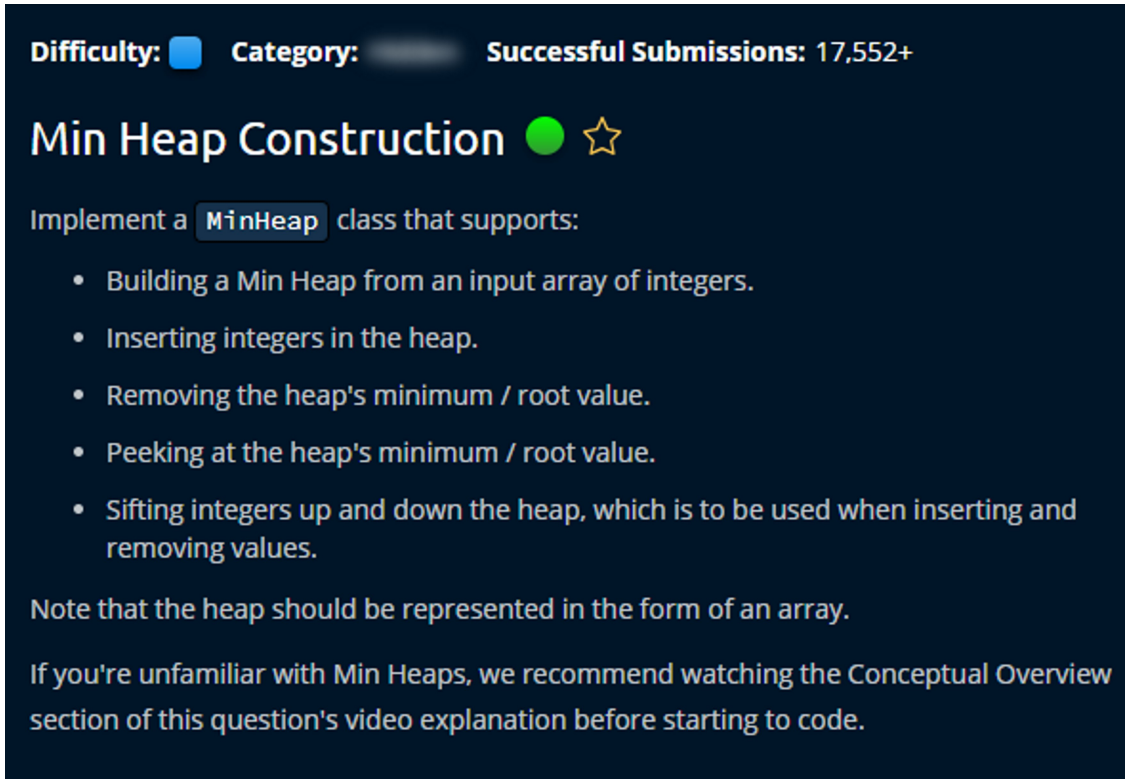
11 August 2023 08:34 PM

Solution 1 | Solution 2 | Solution 3

```cpp
#include <vector>
#include <cmath>
using namespace std;
//this test passed 9/10 test , 1 was failed because of the
// accurate value we get in decimal. we can pass that case
// if we can convert 3.99999999999999991 to 4.

int validStartingCity(vector<int> distances, vector<int> fuel, int mpg) {
  // Write your code here.

  int result = 0;

  for(int start= 0 ; start<distances.size(); start++){
    bool flag = true;
    float reqGalon = abs(distances[start] * 1.0/mpg);
    float leftGalon = abs(fuel[start]);
    for(int i=start+1; i<distances.size(); i++){
      float val = leftGalon - reqGalon;
      if(val >=0  ){
        leftGalon= fuel[i]+ val;
        reqGalon = distances[i]* 1.0/mpg;
      }
      else{
        reqGalon = distances[0] * 1.0/mpg;
        leftGalon = fuel[0];
        flag = false;
        break;
      }
    }
    if(flag){
      result = start;
      return result;
    }
  }

  return -1;

}
```

45.1

11 August 2023 08:28 PM



45.2

11 August 2023 08:37 PM

Solution 1 Solution 2 Solution 3

```cpp
#include <vector>
using namespace std;
//APPROACH : Heap is represented as array. heap follows complete binary tree, so always the
// height the is log(n), so the height does not increase unnecessarily
// remove- we always remove the root, and its place is taken
// by the right most child and then adjusted acc to the max or min heap.
// insert we insert at the end and then shift up .
// Do not edit the class below except for the buildHeap,
// siftDown, siftUp, peek, remove, and insert methods.
// Feel free to add new properties and methods to the class.
class MinHeap {
public:
  vector<int> heap;
  MinHeap(vector<int> vector) { heap = buildHeap(vector); }
  vector<int> buildHeap(vector<int> &vector) {
    // Write your code here.
    int firstParent = (vector.size()-2)/2;
    for(int currentIdx = firstParent; currentIdx >=0; currentIdx--){
        siftDown(currentIdx, vector.size()-1, vector);
    }
    return vector;
  }
  void siftDown(int currentIdx, int endIdx, vector<int> &heap) {
    // Write your code here.
    int childOne = 2 * currentIdx + 1;
    while(childOne <= endIdx){
      int childTwo = childOne+1 <= endIdx ? childOne+1 :-1;
      int idxToSwap;
      if(childTwo != -1 && heap[childTwo] < heap[childOne])
        idxToSwap = childTwo;
      else
        idxToSwap = childOne;
      if(heap[idxToSwap] < heap[currentIdx]){
        swap(heap[currentIdx], heap[idxToSwap]);
        currentIdx = idxToSwap;
        childOne = 2*currentIdx + 1;
      }
      else
        return;
    }
  }
  void siftUp(int currentIdx, vector<int> &heap) {
    // Write your code here.
    int parent = (currentIdx -1)/2;
    int temp = heap[currentIdx];
    while(currentIdx > 0 && temp < heap[parent]  ){
      swap(heap[currentIdx], heap[parent]);
      currentIdx = parent;
      parent = (currentIdx -1)/2;
    }
  }

  int peek() {
    // Write your code here.
    int res=  heap[0];
    return res;
  }

  int remove() {
    //Remember : only root is deleted in heap and the rightmost child take it place
    // and then we adjust its place.
    // Write your code here.
    swap(heap[0], heap[heap.size()-1]);
    int valueToRemove = heap.back(); //storing last element;
    heap.pop_back(); // deleting from last as we shifted the root to last.
    siftDown(0, heap.size()-1, heap);
    return valueToRemove;
  }

  void insert(int value) {
    // Write your code here.
    heap.push_back(value);
    siftUp(heap.size()-1, heap);

  }
};
```

46.1

11 August 2023 08:28 PM

Linked List Construction

Write a `DoublyLinkedList` class that has a `head` and a `tail`, both of which point to either a linked list `Node` or `None` / `null`. The class should support:

- Setting the head and tail of the linked list.
- Inserting nodes before and after other nodes as well as at given positions (the position of the head node is `1`).
- Removing given nodes and removing nodes with given values.
- Searching for nodes with given values.

Note that the `setHead`, `setTail`, `insertBefore`, `insertAfter`, `insertAtPosition`, and `remove` methods all take in actual `Node`s as input parameters—not integers (except for `insertAtPosition`, which also takes in an integer representing the position); this means that you don't need to create any new `Node`s in these methods. The input nodes can be either stand-alone nodes or nodes that are already in the linked list. If they're nodes that are already in the linked list, the methods will effectively be *moving* the nodes within the linked list. You won't be told if the input nodes are already in the linked list, so your code will have to defensively handle this scenario.

If you're doing this problem in an untyped language like Python or JavaScript, you may want to look at the various function signatures in a typed language like Java or TypeScript to get a better idea of what each input parameter is.

Each `Node` has an integer `value` as well as a `prev` node and a `next` node, both of which can point to either another node or `None` / `null`.

Solution 1 | Solution 2 | Solution 3

```cpp
using namespace std;
class Node {
public:
  int value;
  Node *prev;
  Node *next;
  Node(int value);
};

// Feel free to add new properties and me
class DoublyLinkedList {
public:
  Node *head;
  Node *tail;

  DoublyLinkedList() {
    head = nullptr;
    tail = nullptr;
  }
  void setHead(Node *node) {
    // Write your code here.
    if(head==nullptr){
      head = node;
      tail = node;
      return;
    }
    insertBefore(head, node);
  }
  void setTail(Node *node) {
    // Write your code here.
    if(head==nullptr){
      setHead(node);
      return;
    }
    insertAfter(tail, node);
  }
```

46.2

11 August 2023 08:40 PM

```
void insertBefore(Node *node, Node *nodeToInsert) {
  // Write your code here.
  //checking if linkedlist is empty and if yes then there
  // is no need for any insertion.
  if(nodeToInsert == head && nodeToInsert==tail)
    return ;
  remove(nodeToInsert);
  nodeToInsert->next = node;
  nodeToInsert->prev = node->prev;

  if(node->prev==nullptr){
    head = nodeToInsert;
    node->prev = nodeToInsert;
    return;
  }

  node->prev->next = nodeToInsert;
  node->prev = nodeToInsert;

}

void insertAfter(Node *node, Node *nodeToInsert) {
  // Write your code here.
  //checking if linkedlist is empty and if yes then there
  // is no need for any insertion.
  if(nodeToInsert==head && nodeToInsert==tail)
    return;
  remove(nodeToInsert);
  nodeToInsert->prev = node;
  nodeToInsert->next = node->next;
  //checking if node was the last node in linked list.
  if(node->next==nullptr){
    tail=nodeToInsert;
    node->next = nodeToInsert;
    return;
  }
  node->next->prev = nodeToInsert;
  node->next = nodeToInsert;
}
```

```
void insertAtPosition(int position, Node *nodeToInsert) {
  // Write your code here.
  if(position==1){
    setHead(nodeToInsert);
    return;
  }
  int pos= 1;
  Node *node = head;
  while(pos!= position && node!=nullptr){
    node = node->next;
    pos++;
  }
  if(node==nullptr){
    setTail(nodeToInsert);return;}
  insertBefore(node, nodeToInsert);
}

void removeNodesWithValue(int value) {
  // Write your code here.
  if(!containsNodeWithValue(value)) return; // means node is not presen
  Node *node = head;
  while(node!=nullptr){
    Node*toRemove = node;
    node=node->next;
    if(toRemove->value==value)
      remove(toRemove);
  }
}

void remove(Node *node) {
  // Write your code here.
  if(node==head){
    head= head->next;
  }
  if(node==tail){
    tail = tail->prev;
  }
  removePointers(node);
}
```

```
bool containsNodeWithValue(int value) {
  // Write your code here.
  Node *node=head;
  while(node!=nullptr && node->value!=value){
    node=node->next;
  }
  return node!=nullptr;
}
void removePointers(Node *node){
  if(node->prev!=nullptr){
    node->prev->next = node->next;
  }
  if(node->next!=nullptr){
    node->next->prev = node->prev;
  }
  node->next = nullptr;   //just modifying the  links of deleted node
  node->prev=nullptr;
}
};
```

47

11 August 2023 08:28 PM

Remove Kth Node From End

Write a function that takes in the head of a Singly Linked List and an integer `k` and removes the kth node from the end of the list.

The removal should be done in place, meaning that the original data structure should be mutated (no new structure should be created).

Furthermore, the input head of the linked list should remain the head of the linked list after the removal is done, even if the head is the node that's supposed to be removed. In other words, if the head is the node that's supposed to be removed, your function should simply mutate its `value` and `next` pointer.

Note that your function doesn't need to return anything.

You can assume that the input Linked List will always have at least two nodes and, more specifically, at least k nodes.

Each `LinkedList` node has an integer `value` as well as a `next` node pointing to the next node in the list or to `None` / `null` if it's the tail of the list.

Sample Input

```
head = 0 -> 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7 -> 8 -> 9 // the head node
k = 4
```

Sample Output

```
// No output required.
// The 4th node from the end of the list (the node with value 6) is rem
0 -> 1 -> 2 -> 3 -> 4 -> 5 -> 7 -> 8 -> 9
```

Solution 1 Solution 2 Solution 3

```cpp
#include <vector>
using namespace std;
class LinkedList {
public:
  int value;
  LinkedList *next;
  LinkedList(int value);
  void addMany(vector<int> values);
  vector<int> getNodesInArray();
};
void removeKthNodeFromEnd(LinkedList *head, int k) {
  // Write your code here.
  LinkedList *node =head;
  int total=1;  //starting with 1 as at the last node it won't increase the counter
  while(node->next!=nullptr){
    node= node->next;
    total++;
  }
  cout << total<<endl;  //length of linkedlist
  //node pointing at last node.
  int nodeToBeDeleted = total - k;
  LinkedList *n = head;
  if(nodeToBeDeleted==0){
      head->value = head->next->value;
      head->next = head->next->next;       //means it is head
      return ;
  }
  LinkedList *p = head;
  LinkedList *q = nullptr;
  for(int i=0; i<nodeToBeDeleted; i++){
      q=p;
      p=p->next;
  }
  cout << p->value <<endl;
  cout << q->value;
  // p is pointing at the node to be deleted, and q is behind that
  q->next = p->next;
  delete p;
}
```

11 August 2023 08:28 PM

Sum of Linked Lists

You're given two Linked Lists of potentially unequal length. Each Linked List represents a non-negative integer, where each node in the Linked List is a digit of that integer, and the first node in each Linked List always represents the least significant digit of the integer. Write a function that returns the head of a new Linked List that represents the sum of the integers represented by the two input Linked Lists.

Each `LinkedList` node has an integer `value` as well as a `next` node pointing to the next node in the list or to `None` / `null` if it's the tail of the list. The `value` of each `LinkedList` node is always in the range of `0 - 9`.

Note: your function must create and return a new Linked List, and you're not allowed to modify either of the input Linked Lists.

Sample Input

```
linkedListOne = 2 -> 4 -> 7 -> 1
linkedListTwo = 9 -> 4 -> 5
```

Sample Output

```
1 -> 9 -> 2 -> 2
```

Solution 1 | Solution 2 | Solution 3

```cpp
using namespace std;
// This is an input struct. Do not edit.
class LinkedList {
public:
  int value;
  LinkedList *next = nullptr;

  LinkedList(int value) { this->value = value; }
};

LinkedList *sumOfLinkedLists(LinkedList *linkedListOne,
                             LinkedList *linkedListTwo) {
  // Write your code here.
   int sum=0, carry =0;
  LinkedList* linkedListSum = new LinkedList(0);

  LinkedList *one = linkedListOne;
  LinkedList *two = linkedListTwo;
  LinkedList *three = linkedListSum;

  while(one!=nullptr && two!=nullptr){
    sum = carry + one->value + two->value;
    carry = sum>9 ? 1 : 0;
    three->next = new LinkedList(sum%10);
    one = one->next;
    two = two->next;
    three = three->next;
  }
  // now either both linkedlist are empty or any one of them is still
  // left. so we will check it separately and which every is left we
  // will just add the value in linkedlist . and at last we will check
  // if carry is still greater than 0, if yes we will add the carry
  // in list and will reset it  to 0.
  while(one!=nullptr){
    sum = one->value + carry;
    carry = sum>9 ? 1 :0;
    three->next = new LinkedList(sum%10);
    one = one->next;
    three = three->next;
  }
  while(two!=nullptr){
    sum= two->value + carry;
    carry = sum>10? 1 : 0;
    three->next = new LinkedList(sum%10);
    two = two->next;
    three = three->next;
  }

  //if carry is still left add in linkedlist and make carry 0
  if(carry>0){
    three->next = new LinkedList(carry);
  carry = 0;
}


  return linkedListSum->next;     //next is use cause first node is alwa
}
```

49.1

11 August 2023 08:28 PM

Difficulty: 🟦 **Category:** [redacted] **Successful Submissions:** 2,687+

Merging Linked Lists 🟢 ☆

You're given two Linked Lists of potentially unequal length. These Linked Lists potentially merge at a shared intersection node. Write a function that returns the intersection node or returns `None` / `null` if there is no intersection.

Each `LinkedList` node has an integer `value` as well as a `next` node pointing to the next node in the list or to `None` / `null` if it's the tail of the list.

Note: Your function should return an existing node. It should not modify either Linked List, and it should not create any new Linked Lists.

Sample Input

```
linkedListOne = 2 -> 3 -> 1 -> 4
linkedListTwo = 8 -> 7 -> 1 -> 4
```

Sample Output

```
1 -> 4 // The lists intersect at the node with value 1
```

```
DIFFERENT APPROACHES : we can insert all the nodes of 1 linked list in set
and then iterate through second list if that node is present in set
means it is the intersection point and we return it. if we reach the eend
 and the node is not in the set means there was no intersection.
This is an input struct. Do not edit.

second approach : we take both lengths the one having larger length start
iteration to that point till both linkedlist become same length. suppose
3nd is 3, 1st is 5 so we will move 1st by 2 before start iterating list 2.


3rd APPROACH : update version of second. as we know to reach intersection
point we need to travel same no. of node from ptr 1 and ptr2 we can do this
by traveling list two from 1st list ptr and list one from 2nd list ptr.
it will run till it find an interaction point) or until both pointers
reach the end of their respective linked lists. if no intersection they
will reach nullptr together as they are traversing all the nodes.
// If both one and two are not null, the code
// enters a loop that continues until one and two are the
// same. Within the loop:
// It checks if one is null. If it is null, it means it has
// reached the end of linkedListOne and needs to be redirected to
// linkedListTwo. This is done by assigning one to linkedListTwo.
// It checks if two is null. If it is null, it means it has reached the
// end of linkedListTwo and needs to be redirected to linkedListOne.
// This is done by assigning two to linkedListOne.
// If neither one nor two is null, it advances both pointers to their
// respective next nodes. Once the loop terminates, it means one and two
// have either met at the intersection point or have reached the end of
// their respective lists.
```

49.2

11 August 2023 08:48 PM

Solution 1 Solution 2 Solution 3

```cpp
using namespace std;

class LinkedList {
public:
  int value;
  LinkedList *next = nullptr;

  LinkedList(int value) {
    this->value = value;
  }
};


LinkedList* mergingLinkedLists(LinkedList* linkedListOne, LinkedList* linkedListTwo) {
  // Write your code here.
  LinkedList* one = linkedListOne;
  LinkedList* two = linkedListTwo;

  if(one==nullptr || two==nullptr)
    return nullptr;

  while(one!=two){
    one = (one==nullptr) ? linkedListTwo : one->next;
    two = (two==nullptr) ? linkedListOne : two->next;
  }

  return one;
}

```

Solution 1 Solution 2 Solution 3

```cpp
using namespace std;

// This is an input struct. Do not edit.
class LinkedList {
public:
  int value;
  LinkedList *next = nullptr;

  LinkedList(int value) {
    this->value = value;
  }
};
// TIME : O(n*n)
LinkedList* mergingLinkedLists(LinkedList* linkedListOne, LinkedList* linkedListTwo) {
  // Write your code here.
  LinkedList* one = linkedListOne;

  while(one!=nullptr){
    int oneValue = one->value;
    LinkedList* two = linkedListTwo;
    while(two!=nullptr){
      int twoValue = two->value;
      if(oneValue==twoValue){
        return one;
      }
      two = two->next;
    }
    one= one->next;
  }
  return nullptr;
}
```

50.1

11 August 2023 08:28 PM

Permutations

Write a function that takes in an array of unique integers and returns an array of all permutations of those integers in no particular order.

If the input array is empty, the function should return an empty array.

Sample Input

```
array = [1, 2, 3]
```

Sample Output

```
[[1, 2, 3], [1, 3, 2], [2, 1, 3], [2, 3, 1], [3, 1, 2], [3, 2, 1]]
```

```
APPROACH :
1st way is we take a int from array everytime
and then we create a new array from removing that element from the array.
and insert that num into perm array and recursively run that function with perm array and the new array and when the array is empty means the permutation is created for that integer so we push that into 2d array.

time complexity :
as array of n integers creates n! permutations.
and helper function is called n times and creating new array stuffs take n times.
so total = O(n!.n.n) this is the upper bound.
space :
as 2d array is of size n! with each array of n size
so total : O(n!.n)


OPTIMAL APPROACH:
the approach is similar instead of creating new array everytime, we mutate the original array by swapping elements.
so the base case is still called n! times and in that we
are copying the array which takes O(n) times.
the helper fn is called n times and swapping takes O(1).
so total = O(n!.n)  (n! have n times in copying)
space is same.



NOTE : when calling a func if you are sending a address of the array, then fn parameter should be a ptr to catch it.
```

50.2

11 August 2023 08:50 PM

Solution 1 Solution 2 Solution 3

```cpp
#include <vector>
using namespace std;
//APPROACH : this can be optimized when instead of creating new array everytime
// we can swap these elements.
void helper(vector<int>perm, vector<vector<int>> *results, vector<int> arr){
  //base case
  if(arr.size()==0 && perm.size()>0)
    results->push_back(perm);
  else{
    for(int i=0; i<arr.size(); i++){
      vector<int>newArray;
      newArray.insert(newArray.end(), arr.begin(), arr.begin()+i); //exact copy of arr.
      newArray.insert(newArray.end(), arr.begin()+i+1, arr.end());
      vector<int> newPerm = perm;
      newPerm.push_back(arr[i]);
      helper(newPerm, results, newArray);
    }
  }

}

vector<vector<int>> getPermutations(vector<int> array) {
  // Write your code here.
  vector<vector<int>> results;
  helper({}, &results, array);
  return results;
}
```

Solution 1 Solution 2 Solution 3

```cpp
#include <vector>
using namespace std;
//optimized time using swap approach.

void helper(int i, vector<int>arr, vector<vector<int>> &results){
  if(i==arr.size()-1)
    results.push_back(arr);
  else{
    for(int j =i; j<arr.size(); j++){
      swap(arr[i], arr[j]);
      helper(i+1, arr, results);
      swap(arr[i], arr[j]);
    }
  }
}

vector<vector<int>> getPermutations(vector<int> array) {
  // Write your code here.
  vector<vector<int>> results;
  helper(0, array, results);


  return results;
}
```

51.1

11 August 2023 08:50 PM

Powerset

Write a function that takes in an array of unique integers and returns its powerset.

The powerset `P(X)` of a set `X` is the set of all subsets of `X`. For example, the powerset of `[1,2]` is `[[], [1], [2], [1,2]]`.

Note that the sets in the powerset do not need to be in any particular order.

Sample Input

```
array = [1, 2, 3]
```

Sample Output

```
[[], [1], [2], [3], [1, 2], [1, 3], [2, 3], [1, 2, 3]]
```

```
Approach : this approach is using iterative idea. as we know every set will have an empty subset so what we do is
for every integer we look in the result of subsets and add that integer in that and push it in the results.
iterate through every no. and for every no. iterate through
all the subsets and make a new subset and add it in the result.

TIME COMPLEXITY:
it is a lil tricky and different from what we do in every ques count the for loop and operations.
in this we see for every integer the subsets increased by double. so for n we have 2^n subsets and we have to append an element every time, there are subsets of length 0, 1....n. so on average length is n/2.
so total time = O(2^n. n/2) = O(2^n .n).
space = O(n. 2^n)



APPROACH II : it is a recursive approach.
using recursion we are sending all the subsets in 2d array
for that particular element and then return it for previous call and then the same procedure we create a new vector , iterate all the subsets , push that element in that and then push the vector in subsets 2d array.
same space and time complexity
```

51.2

11 August 2023 08:52 PM

Solution 1 Solution 2 Solution 3

```cpp
#include <vector>
using namespace std;

void helper(vector<int> arr, vector<vector<int>> &res){
  res.push_back({});
  for(int i=0; i<arr.size(); i++){
    int length = res.size();
    for(int j=0; j<length; j++){
      vector<int> newSubSet = res[j];
      newSubSet.push_back(arr[i]);
      res.push_back(newSubSet);
    }
  }
}

vector<vector<int>> powerset(vector<int> array) {
  // Write your code here.
  vector<vector<int>> res;
  helper(array, res);
  return res;
}
```

Solution 1 Solution 2 Solution 3

```cpp
#include <vector>
using namespace std;

vector<vector<int>> helper(vector<int>&array, int idx){
  if(idx<0)
    return vector<vector<int>>{{}};
  int element = array[idx];
  vector<vector<int>> subsets = helper(array, idx-1);
  int length = subsets.size();
  for(int j=0; j<length; j++){
    vector<int> currentSub = subsets[j];
    currentSub.push_back(element);
    subsets.push_back(currentSub);
  }
  return subsets;
}

vector<vector<int>> powerset(vector<int> array) {
  // Write your code here.
  return helper(array, array.size()-1); //sending last index

}
```

52

11 August 2023 08:50 PM

Search In Sorted Matrix

You're given a two-dimensional array (a matrix) of distinct integers and a target integer. Each row in the matrix is sorted, and each column is also sorted; the matrix doesn't necessarily have the same height and width.

Write a function that returns an array of the row and column indices of the target integer if it's contained in the matrix, otherwise `[-1, -1]`.

Sample Input

```
matrix = [
  [1, 4, 7, 12, 15, 1000],
  [2, 5, 19, 31, 32, 1001],
  [3, 8, 24, 33, 35, 1002],
  [40, 41, 42, 44, 45, 1003],
  [99, 100, 103, 106, 128, 1004],
]
target = 44
```

Sample Output

```
[3, 3]
```

```
brute approach:
traversing through each element and checking if that matches the target, we return the current i and j.
if the loops completes means there was no element that matches target then we return -1,-1.

time complexity : O(n*m) where n*m are total elements.


optimal -> as matrix are sorted we start with last column. if that is bigger than the target we eliminate that column and move to prev column, if that is less than target we eliminate that row and move to next row. So we are eliminating bunch of matrix by doing this.

time complexity : this is basically a zigzag traversal we go left or down. atmost we traverse n+m no.s where n is size of rows and m is size of cols. so O(n+m).
```

Solution 1 | Solution 2 | Solution 3

```cpp
#include <vector>
using namespace std;
//brute force

vector<int> searchInSortedMatrix(vector<vector<int>> matrix, int target) {
  // Write your code here.
  int rows = matrix.size();
  int cols = matrix[0].size();
  vector<int> result;

  for(int i=0; i<rows; i++){
    for(int j=0; j<cols; j++){
      if(matrix[i][j]==target){
        result.push_back(i);
        result.push_back(j);
        return result;
      }
    }
  }
  return {-1,-1};
}
```

Solution 1 | Solution 2 | Solution 3

```cpp
#include <vector>
using namespace std;
//optimal
vector<int> searchInSortedMatrix(vector<vector<int>> matrix, int target) {
  // Write your code here.
  int rows = matrix.size();
  int cols = matrix[0].size();

  int col = cols-1;
  int row = 0;

  while(row < rows && col >= 0){
    if(matrix[row][col]==target)
      return {row, col};
    else if(matrix[row][col] > target){
      col = col-1;
    }
    else if(matrix[row][col] < target){
      row = row+1;
    }
  }
  return {-1,-1};
}
```

11 August 2023 08:50 PM

Three Number Sort

You're given an array of integers and another array of three distinct integers. The first array is guaranteed to only contain integers that are in the second array, and the second array represents a desired order for the integers in the first array. For example, a second array of `[x, y, z]` represents a desired order of `[x, x, ..., x, y, y, ..., y, z, z, ..., z]` in the first array.

Write a function that sorts the first array according to the desired order in the second array.

The function should perform this in place (i.e., it should mutate the input array), and it shouldn't use any auxiliary space (i.e., it should run with constant space: `O(1)` space).

Note that the desired order won't necessarily be ascending or descending and that the first array won't necessarily contain all three integers found in the second array—it might only contain one or two.

Sample Input

```
array = [1, 0, 0, -1, -1, 0, 1, 1]
order = [0, 1, -1]
```

Sample Output

```
[0, 0, 0, 1, 1, 1, -1, -1]
```

```
so the approach i will be using is similar to count sort. but instead of using any other auxiliary space, i will mutate the array with the variables.

we count the order[0] order[1] order[2] elements in main array in first, second, third variables.
now we know order[0] occurance in main array is first.
then we use while to mutate original array for first , second  and third variables. first we insert order[0] first times, then order[1] second times, order[2] third no. of times.


time complexity:
The time complexity of vector operations such as resizing and assigning elements is generally considered to be amortized O(1). Therefore, we can safely assume that the individual assignments in the while loops have constant time complexity in the context of this code. As a result, the overall time complexity remains O(n).

space : O(1)
```

Solution 1 | Solution 2 | Solution 3

```cpp
#include <vector>
using namespace std;

vector<int> threeNumberSort(vector<int> array, vector<int> order) {
  // Write your code here.
  int size =array.size();
  int first=0;
  int second=0;
  int third=0;
  for(int  i=0; i<array.size();i++){
    if(array[i]==order[0])
      first++;
    else if(array[i]==order[1])
      second++;
    else if(array[i]==order[2])
      third++;
  }

  int i=0;
  while(first!=0){
    array[i] = order[0];
    first--;
    i++;
  }
  while(second!=0){
    array[i] = order[1];
    second--;
    i++;
  }
  while(third!=0){
    array[i] = order[2];
    third--;
    i++;
  }

  return array;
}
```

54.1

11 August 2023 08:50 PM

Min Max Stack Construction

Write a `MinMaxStack` class for a Min Max Stack. The class should support:

- Pushing and popping values on and off the stack.
- Peeking at the value at the top of the stack.
- Getting both the minimum and the maximum values in the stack at any given point in time.

All class methods, when considered independently, should run in constant time and with constant space.

```
we created a array of max size of 1000;
then for push we check if array is empty, we push it in the stack and make min and max equal to that no.
if stack is not empty, we check if the number is less than min, then we get the new min but before inserting it directly, we convert the no. as (2*number-minEle) to track of prev min. it will be useful in pop. if suppose the deleted number is the min then how we would assign the new min? if we don't convert it then it will take more than O(1) time complexity which we don't want.

for that we use the converted no., we took it out convert it into prev min using (2*minEle - no.) this is the prev min then we update the min with that and the deleted no. actuals is the minEle before the updation of minEle.
In peek we check if top element is less than min element. means it is the converted value for minEle, so we return minEle

same for max.

time complexity: O(1) for independent funs.

```

Solution 1 Solution 2 Solution 3

```cpp
#include<vector>
using namespace std;

#define MAX_SIZE 1000
// Feel free to add new properties and methods to the class.
class MinMaxStack {
private:
int top;
int arr[MAX_SIZE];
int minEle;
int maxEle;

 public:
MinMaxStack(){
  top = -1;
}
  int peek() {
    // Write your code here.
    if(top<0){
      return -1;
    }
    int num = arr[top];
    if(num<minEle){
      return minEle;
    }
    else if(num > maxEle){
      return maxEle;
    }
    else
      return arr[top];
  }
```

54.2

11 August 2023 09:02 PM

```
int pop() {
  // Write your code here.
  if(top<0)  //stack is empty
    return -1;
  int num_to_be_deleted = arr[top];
  if(num_to_be_deleted < minEle){
    int returnedValue = minEle;
    int prevMin = 2*minEle - num_to_be_deleted;
    minEle = prevMin;
    top--;
    return returnedValue;;
  }
  else if(num_to_be_deleted > maxEle){
    int prevMax = 2*maxEle - num_to_be_deleted;
    int returnedValue = maxEle;
    maxEle = prevMax;
    top--;
    return returnedValue;
  }
  else{
    return arr[top--];
  }

}
```

```
void push(int number) {
  // Write your code here.
  if(top>=999) // stack is full
    return ;
  else if(top<0){
    top++;
    arr[top] = number;
    minEle = maxEle = number;
  }
  else if(number < minEle){
    top++;
    int convert = 2*number - minEle;
    arr[top] = convert;
    minEle = number;
  }
  else if(number>maxEle){
    top++;
    int convert  =2*number- maxEle;
    arr[top] = convert;
    maxEle = number;
  }
  else{
    arr[++top] = number;
  }
}

int getMin() {
  // Write your code here.
  if(top<0)
    return -1;
  return minEle;
}

int getMax() {
  // Write your code here.
  if(top<0)
    return -1;
  return maxEle;
}
};

```

55

11 August 2023 08:50 PM

Solution 1 Solution 2 Solution 3

```cpp
#include <vector>
#include <stack>
using namespace std;

void insertInSortStack(vector<int>&stack, int key){
  if(stack.size()==0 || key>stack.back()){
    stack.push_back(key);
    return;
  }
  int top = stack.back();
  stack.pop_back();
  insertInSortStack(stack, key);
  stack.push_back(top);

}

vector<int> sortStack(vector<int>& stack) {
  // Write your code here.

  if(stack.size()==0)
    return {};

  int top = stack.back();
  stack.pop_back();

  sortStack(stack);

  insertInSortStack(stack, top);
  return stack;
}

```

Sort Stack

Write a function that takes in an array of integers representing a stack, recursively sorts the stack in place (i.e., doesn't create a brand new array), and returns it.

The array must be treated as a stack, with the end of the array as the top of the stack. Therefore, you're only allowed to

- Pop elements from the top of the stack by removing elements from the end of the array using the built-in `.pop()` method in your programming language of choice.
- Push elements to the top of the stack by appending elements to the end of the array using the built-in `.append()` method in your programming language of choice.
- Peek at the element on top of the stack by accessing the last element in the array.

You're not allowed to perform any other operations on the input array, including accessing elements (except for the last element), moving elements, etc.. You're also not allowed to use any other data structures, and your solution must be recursive.

Sample Input

```
stack = [-5, 2, -2, 4, 3, 1]
```

Sample Output

```
[-5, -2, 1, 2, 3, 4]
```

```
first we take a value out from top and store it, then pop
we do this till stack is empty recursively and while unfolding we also run a insert to sort the stack.
now stack is empty recursion starts folding for the last pop element and does the remaining work for that call. insert is called and check if stack is empty or key is greater than stack top push, else took the key is small and stack is not empty.
took out element from stack and insert key again if still its smaller again it will remove the top and will run insert again and while unfolding these call we perform push back for the element which was being removed.
and in this way stack gets sorted....

// the stack is vector here so the funs of vector was being used by us.


time complexity ;
O(N^2)

space : O(n), a recursive stack was used.
```

56

11 August 2023 08:50 PM

Solution 1 Solution 2 Solution 3

```cpp
using namespace std;

string reverseWordsInString(string str) {
  // Write your code here.
  //reverse the string
  reverse(str.begin(), str.end());

//add space at last.
  str.insert(str.end(), ' ');
  int j=0;
  //iterate
  for(int i=0; i<str.length(); i++){
    if(str[i]==' '){
      reverse(str.begin()+j, str.begin()+i);
      j=i+1;  // updating starting index for next word to reverse.
    }

  }
  str.pop_back(); // Remove spaces from the end of the
                  // word that we appended


  return str;
}
```

```
APPROACH:
reverse the string first.
the whole string is reversed, each word is at its place but reversed. now we iterate through this array and we need to reverse each word to get the original word, whenever a space is encountered we reverse from starting point till i. starting point is determined by other pointer which gets updated after word is reversed.

before iteratating we Add space at the end so that the last word is also reversed, as if no space will be there then the last word will remain reversed.

time complexity:
O(n)
O(1), space



other approaches:
1 method. using stringstream and vector
2 method. {
    i)Initially reverse each word of the given string str.
    ii) Now reverse the whole string to get the resultant
        string in desired order.
}
```

Reverse Words In String

Write a function that takes in a string of words separated by one or more whitespaces and returns a string that has these words in reverse order. For example, given the string `"tim is great"`, your function should return `"great is tim"`.

For this problem, a word can contain special characters, punctuation, and numbers. The words in the string will be separated by one or more whitespaces, and the reversed string must contain the same whitespaces as the original string. For example, given the string `"whitespaces  4"` you would be expected to return `"4  whitespaces"`.

Note that you're ***not*** allowed to to use any built-in `split` or `reverse` methods/functions. However, you ***are*** allowed to use a built-in `join` method/function.

Also note that the input string isn't guaranteed to always contain words.

Sample Input

```
string = "AlgoExpert is the best!"
```

Sample Output

```
"best! the is AlgoExpert"
```