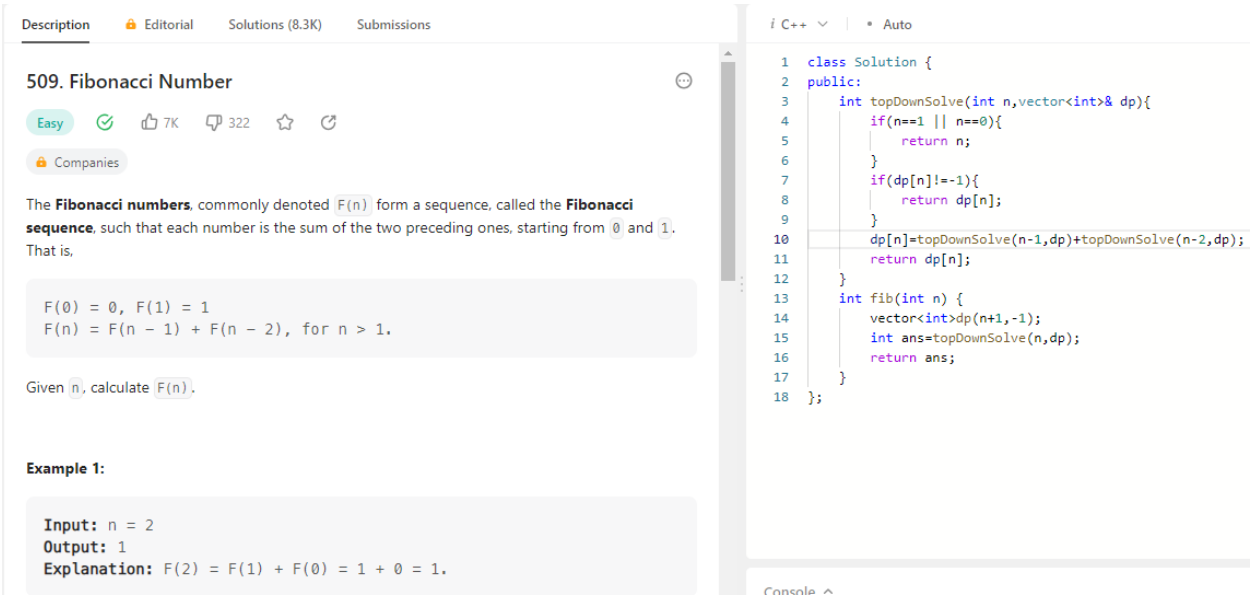
Dynamic Programming – 1 5/10/23

Dynamic Programming Class -1

1) top down approach



Description Editorial Solutions (8.3K) Submissions

509. Fibonacci Number

Easy 7K 322

Companies

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

```
F(0) = 0, F(1) = 1
F(n) = F(n - 1) + F(n - 2), for n > 1.
```

Given `n`, calculate `F(n)`.

Example 1:

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

C++ Auto

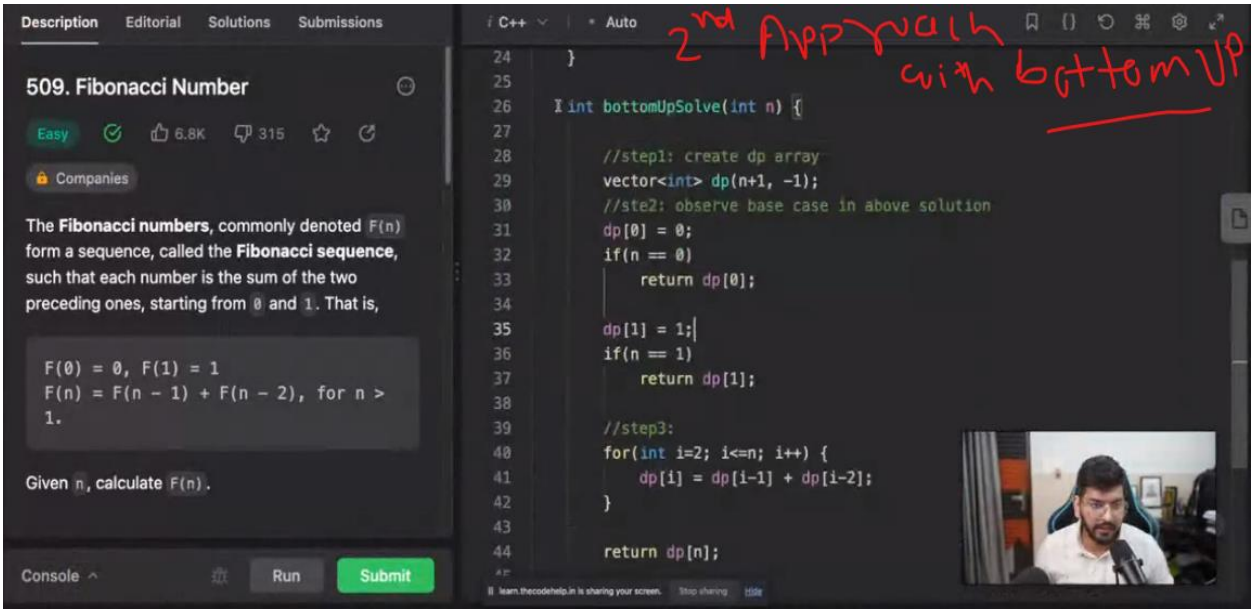
```
class Solution {
public:
    int topDownSolve(int n,vector<int>& dp){
        if(n==1 || n==0){
            return n;
        }
        if(dp[n]!=-1){
            return dp[n];
        }
        dp[n]=topDownSolve(n-1,dp)+topDownSolve(n-2,dp);
        return dp[n];
    }
    int fib(int n) {
        vector<int>dp(n+1,-1);
        int ans=topDownSolve(n,dp);
        return ans;
    }
};
```

Console

Tc=O(n)+O(n)

Sc=O(n)

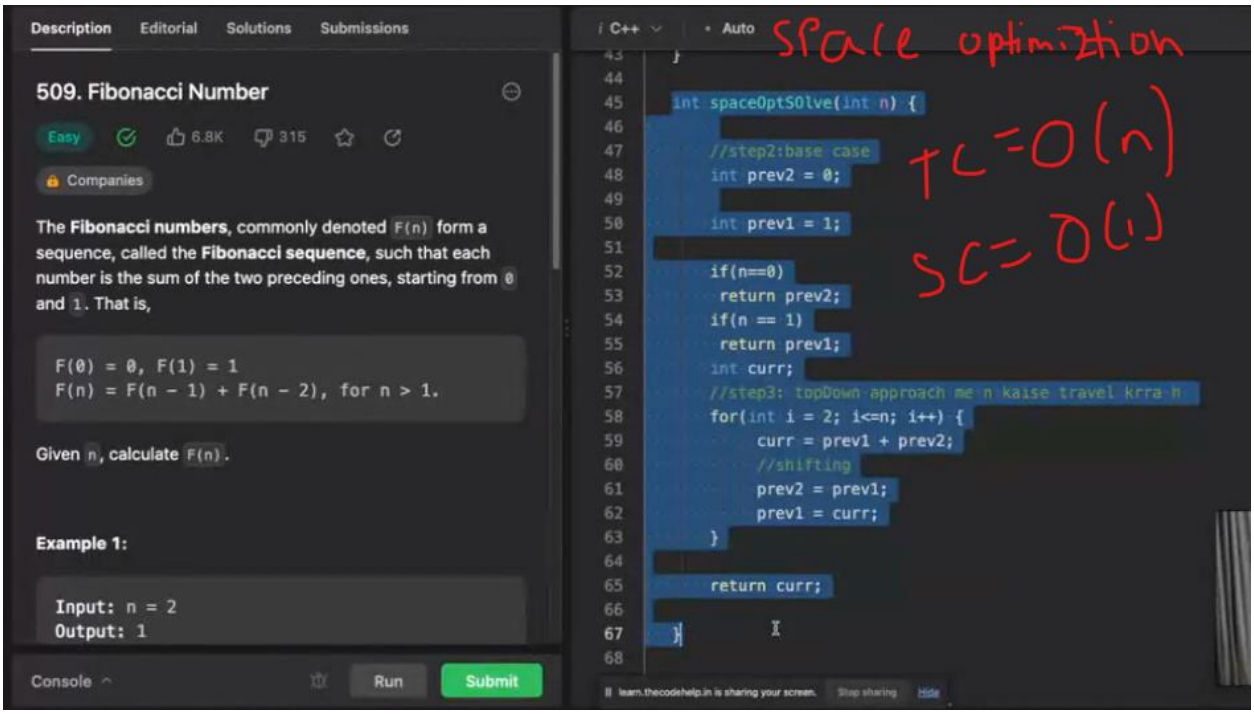
2) Bottom up approach



Tc=O(n)

Sc=O(n)

Space Optimization

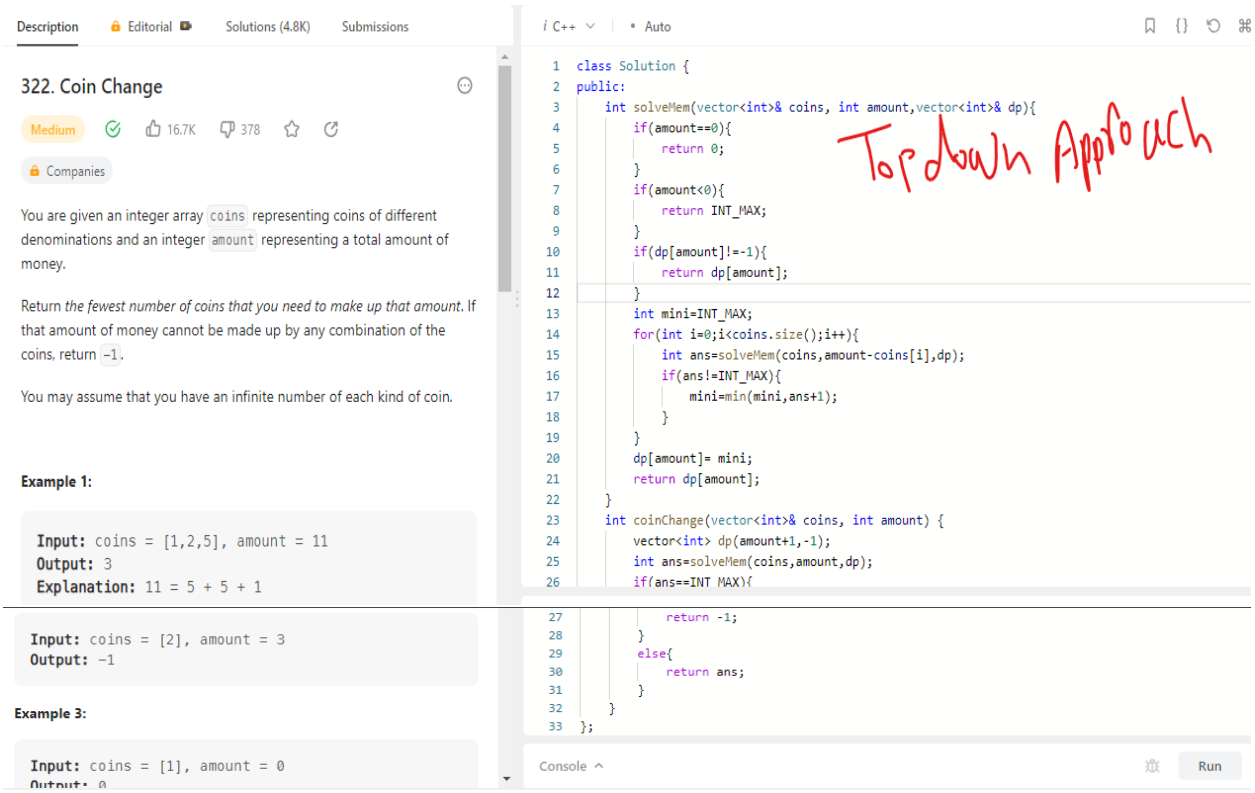


Dynamic Programming Class -2

Q) coin change using recursion (Important question LeetCode 322)

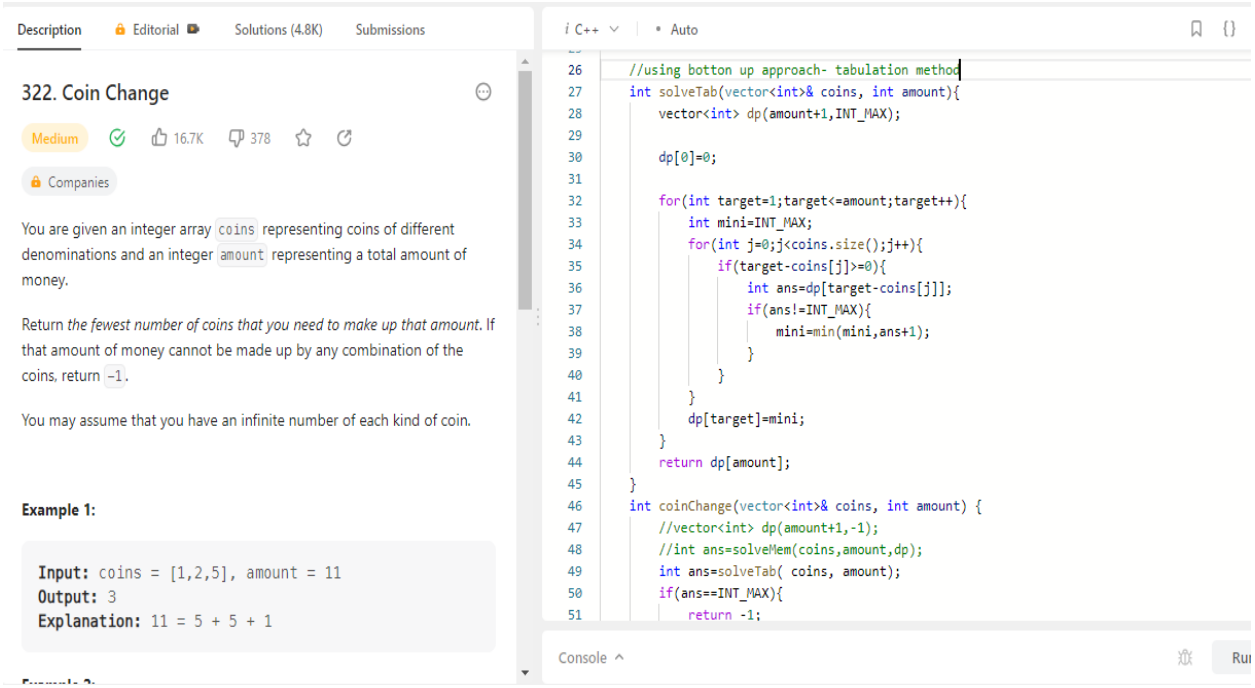
```cpp
#include <iostream>
#include <vector>
#include <limits.h>

using namespace std;

int solveUsingRecursion(vector<int>& coins,int amount){
    if(amount==0){
        return 0;
    }
    if(amount<0){
        return INT_MAX;
    }

    int mini=INT_MAX;

    for(int i=0;i<coins.size();i++){
        int ans=solveUsingRecursion(coins,amount-coins[i]);
        if(ans!=INT_MAX){
            mini=min(mini,ans + 1);
        }
    }
    return mini;

}

int main()
{
    vector<int>coins={1,2,5};
    int amount=11;
    int ans=solveUsingRecursion(coins,amount);
    cout<<"minimum coin is :"<<ans;
    return 0;
}
```

```
PS E:\C++Code> g++ .\f77.cpp
PS E:\C++Code> .\a.exe
minimum coin is :3
```



Tc=O(amount)

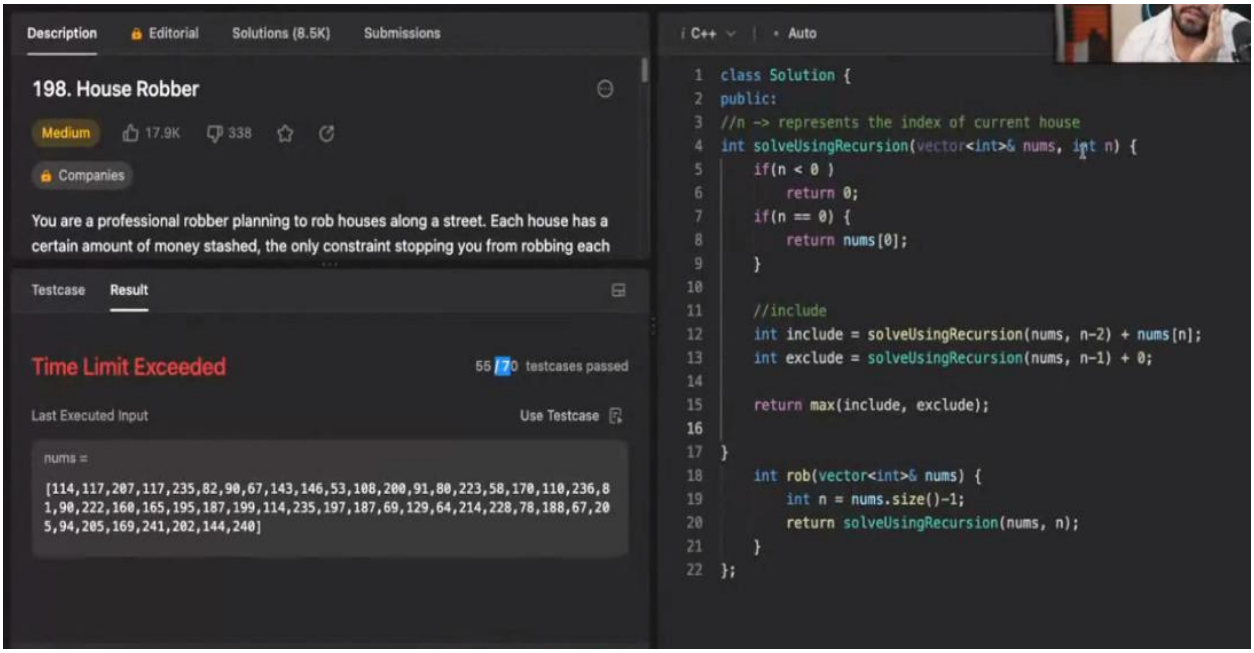
2nd approach bottom approach (tabulation method)



Tc=O(n)

Q) 198. House Robber (Importan Question)

Approach 1st using recursion



Tc=expontial

```
//using memosiation(top down approach - DP)
int solveUsingMem(vector<int>& nums,int n,vector<int>& dp){
     if(n==0){
        return nums[0];
    }
    if(n<0){
        return 0;
    }

    if(dp[n]!=-1){
        return dp[n];
    }
    int include=solveUsingMem(nums,n-2,dp)+nums[n];
    int exclude=solveUsingMem(nums,n-1,dp)+0;

    dp[n]=max(include,exclude);
    return dp[n];
}
```

Tc=O(n) ,Sc=O(n)+O(n)

```cpp
//using bottom up Approach
int solveUsingTabulation(vector<int>& nums,int n){
    vector<int>dp(n+1,0);
    dp[0]=nums[0];

    for(int i=1;i<=n;i++){
        int temp=0;

        if(i-2>=0){
            temp=dp[i-2];
        }
        int include=temp+nums[i];
        int exclude=dp[i-1]+0;

        dp[i]=max(include,exclude);
    }
    return dp[n];
}
```

Tc=O(n) ,ScO(n)

198. House Robber

Medium 18.3K 343

Companies

You are a professional robber planning to rob houses along a street. Each house has a certain amount of money stashed, the only constraint stopping you from robbing each of them is that adjacent houses have security systems connected and **it will automatically contact the police if two adjacent houses were broken into on the same night**.

Given an integer array `nums` representing the amount of money of each house, return *the maximum amount of money you can rob tonight **without alerting the police***.

Example 1:

Input: nums = [1,2,3,1]
Output: 4
Explanation: Rob house 1 (money = 1) and then rob house 3 (money = 3).
Total amount you can rob = 1 + 3 = 4.

```cpp
class Solution {
public:


    //using recursion
    int solveUsingRecursion(vector<int>& nums,int n){
        if(n==0){
            return nums[0];
        }
        if(n<0){
            return 0;
        }

        int include=solveUsingRecursion(nums,n-2)+nums[n];
        int exclude=solveUsingRecursion(nums,n-1)+0;

        return max(include,exclude);
    }

    //-----------------------------------------------------//

    //using memosiation(top down approach - DP)
    int solveUsingMem(vector<int>& nums,int n,vector<int>& dp){
        if(n==0){
            return nums[0];
        }
```

Console

Tc=O(n) ,Sc=(1)

Dynamic Programming Class – 3

Q) Panting fence problem problem using recursion (IMP)

```cpp
#include <iostream>

using namespace std;
int solveUsingRecursion(int n,int k){
    if(n==1){
        return k;
    }
    if(n==2){
        return k+k*(k-1);
    }

    int ans=(solveUsingRecursion(n-2,k)+solveUsingRecursion(n-1,k)) *(k-1);
    return ans;
}
int main()
{
    int n=4;
    int k=3;
    int ans=solveUsingRecursion(n,k);
    cout<<"ans is :"<<ans;
    return 0;
}
```

```
PS E:\C++Code> g++ .\f77.cpp
PS E:\C++Code> .\a.exe
ans is :66
```

Time complexity : exponential

2nd approach solve panting fence problem using (top dowan - dp)

```cpp
#include <iostream>
#include <vector>

using namespace std;

int solveUsingMem(int n,int k,vector<int>& dp){
    if(n==1){
        return k;
    }
    if(n==2){
        return k+k*(k-1);
    }

    if(dp[n]!=-1){
        return dp[n];
    }
    dp[n]=(solveUsingMem(n-2,k,dp)+solveUsingMem(n-1,k,dp)) *(k-1);
    return dp[n];
}
int main()
{
    int n=4;
    int k=3;
    vector<int>dp(n+1,-1);
    int ans=solveUsingMem(n,k,dp);
    cout<<"ans is :"<<ans;
    return 0;
}
```

```
PS E:\C++Code> g++ .\f77.cpp
PS E:\C++Code> .\a.exe
ans is :66
```

Time complexity : O(n)

3rd approach solve panting fence problem using (bottom up - dp)

```cpp
#include <iostream>
#include <vector>

using namespace std;

int solveUsingTab(int n,int k){
    vector<int>dp(n+1,0);

    dp[1]=k;
    dp[2]=k+k*(k-1);

    for(int i=3;i<=n;i++){
        dp[i]=(dp[i-2]+dp[i-1]) *(k-1);
    }

    return dp[n];
}
int main()
{
    int n=4;
    int k=3;
    vector<int>dp(n+1,-1);
    int ans=solveUsingTab(n,k);
    cout<<"ans is :"<<ans;
    return 0;
}
```

```
PS E:\C++Code> g++ .\f77.cpp
PS E:\C++Code> .\a.exe
ans is :66
```

4th approach solve panting fence problem using (space optimization - dp)

```cpp
#include <iostream>
#include <vector>

using namespace std;

int solveUsingSpaceOptimization(int n,int k){
    vector<int>dp(n+1,0);

    int prev2=k;
    int prev1=k+k*(k-1);

    for(int i=3;i<=n;i++){
        int curr=(prev2+prev1) *(k-1);
        prev2=prev1;
        prev1=curr;
    }
    return prev1;
}

int main()
{
    int n=4;
    int k=3;

    int ans=solveUsingSpaceOptimization(n,k);
    cout<<"ans is :"<<ans;
    return 0;
}
```

```
PS E:\C++Code> g++ .\f77.cpp
PS E:\C++Code> .\a.exe
ans is :66
```

Time complexity : O(n)

Space complexity : O(1)

Q) 0/1 Knapsack Problem and Dynamic Programming (Imp)

1st approach using recursion

```cpp
#include <iostream>

using namespace std;

int solveUsingRecursion(int weight[],int value[],int index,int capacity){
    if(index==0){
        if(weight[0]<=capacity){
            return value[0];
        }
        else{
            return 0;
        }
    }

    int include =0;
    if(weight[index]<=capacity){
        include=value[index]+solveUsingRecursion(weight,value,index-1,capacity-
weight[index]);
    }

    int exclude=0+solveUsingRecursion(weight,value,index-1,capacity);

    int ans=max(include,exclude);
    return ans;

}
int main()
{
    int weight[]={4,5,1};
    int value[]={1,2,3};
    int n=3;
    int capacity=4;

    int ans=solveUsingRecursion(weight,value,n-1,capacity);
    cout<<"knapsack ans:"<<ans;
    return 0;
}
```

```
PS E:\C++Code> g++ .\f77.cpp
PS E:\C++Code> .\a.exe
knapsack ans:3
```

2nd approach using top down(memoization)

```
#include <iostream>
#include <vector>

using namespace std;

int solveUsingMem(int weight[],int value[],int index,int capacity,vector<
vector<int> >&dp){

    if(index==0){
        if(weight[0]<=capacity){
            return value[0];
        }
        else{
            return 0;
        }
    }

    if(dp[index][capacity]!=-1){
        return dp[index][capacity];
    }

    int include =0;

    if(weight[index]<=capacity){
        include=value[index]+solveUsingMem(weight,value,index-1,capacity-
weight[index],dp);
    }

    int exclude=0+solveUsingMem(weight,value,index-1,capacity,dp);

    dp[index][capacity]=max(include,exclude);

    return dp[index][capacity];

}
int main()
{
    int weight[]={4,5,1};
    int value[]={1,2,3};
    int n=3;
    int capacity=4;
```

```
    vector< vector<int> >dp(n,vector<int>(capacity+1,-1));

    int ans=solveUsingMem(weight,value,n-1,capacity,dp);
    cout<<"knapsack ans:"<<ans;
    return 0;
}
```

```
PS E:\C++Code> g++ .\f77.cpp
PS E:\C++Code> .\a.exe
knapsack ans:3
```

3rd approach using tabulation (bottom top)

```cpp
#include <iostream>
#include <vector>

using namespace std;

int solveUsingTabulation(int weight[],int value[],int n,int capacity){
    vector< vector<int> >dp(n,vector<int>(capacity+1,0));

    for(int w=weight[0];w<=capacity;w++){
        if(weight[0]<=capacity){
            dp[0][w]=value[0];
        }
        else{
            dp[0][w]=0;
        }
    }

    for(int index=1;index<n;index++){
        for(int wt=0;wt<=capacity;wt++){
            int include=0;
            if(weight[index]<=wt){
                include=value[index]+dp[index-1][wt-weight[index]];
            }
            int exclude=0+dp[index-1][wt];
            dp[index][wt]=max(include,exclude);
        }
    }
    return dp[n-1][capacity];

}
int main()
{
    int weight[]={4,5,1};
    int value[]={1,2,3};
    int n=3;
    int capacity=4;

    int ans=solveUsingTabulation(weight,value,n,capacity);
    cout<<"knapsack ans:"<<ans;
    return 0;
}
```


PS E:\C++Code> g++ .\f77.cpp
PS E:\C++Code> .\a.exe
knapsack ans:3

4th approach space optimization

```cpp
#include <iostream>
#include <vector>

using namespace std;

int solveUsingSpaceOptimization(int weight[],int value[],int n,int capacity){
    vector<int>prev(capacity+1,0);
    vector<int>curr(capacity+1,0);

    for(int w=weight[0];w<=capacity;w++){
        if(weight[0]<=capacity){
            prev[w]=value[0];
        }
        else{
            prev[w]=0;
        }
    }

    for(int index=1;index<n;index++){
        for(int wt=0;wt<=capacity;wt++){
            int include=0;
            if(weight[index]<=wt){
                include=value[index]+prev[wt-weight[index]];
            }
            int exclude=0+prev[wt];
            curr[wt]=max(include,exclude);
        }
        prev=curr;
    }
    return prev[capacity];

}
int main()
{
    int weight[]={4,5,1};
    int value[]={1,2,3};
    int n=3;
    int capacity=4;

    int ans=solveUsingSpaceOptimization(weight,value,n,capacity);
    cout<<"knapsack ans:"<<ans;
    return 0;
}
```

```
PS E:\C++Code> g++ .\f77.cpp
PS E:\C++Code> .\a.exe
knapsack ans:3
```

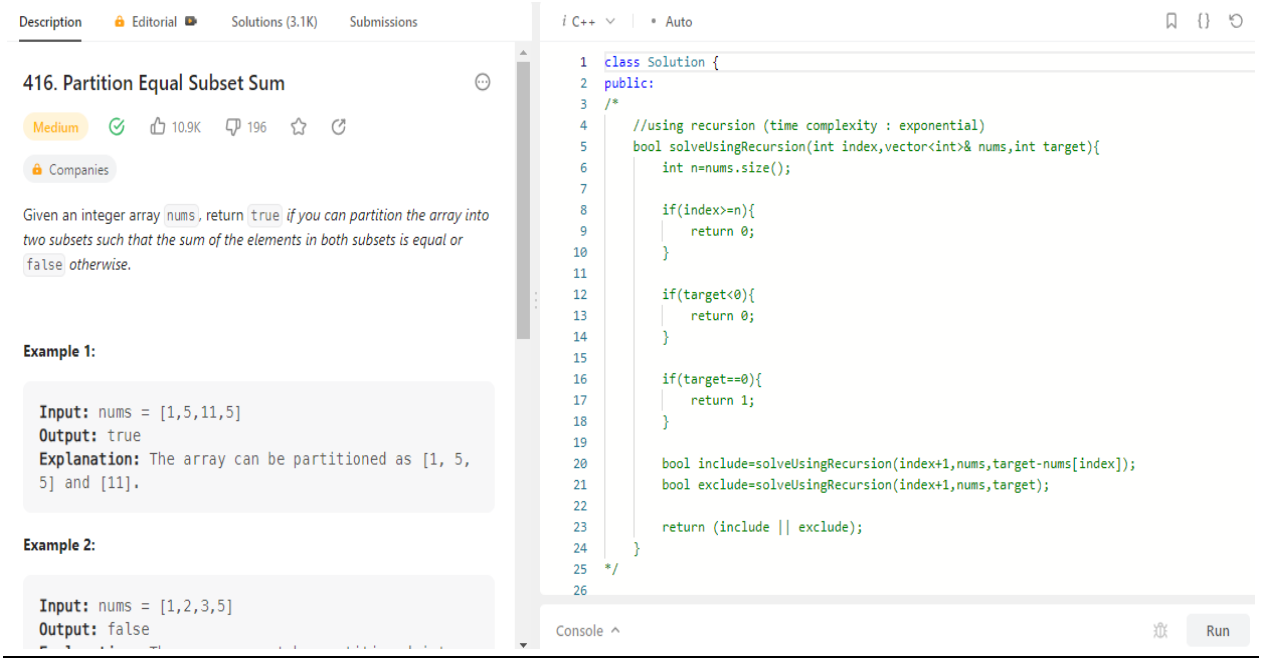
5th approach using space optimization

```cpp
#include <iostream>
#include <vector>

using namespace std;

int solveUsingSpaceOptimization(int weight[],int value[],int n,int capacity){
    //vector<int>prev(capacity+1,0);
    vector<int>curr(capacity+1,0);

    for(int w=weight[0];w<=capacity;w++){
        if(weight[0]<=capacity){
            curr[w]=value[0];
        }
        else{
            curr[w]=0;
        }
    }

    for(int index=1;index<n;index++){
        for(int wt=capacity;wt>=0;wt--){
            int include=0;
            if(weight[index]<=wt){
                include=value[index]+curr[wt-weight[index]];
            }
            int exclude=0+curr[wt];
            curr[wt]=max(include,exclude);
        }
        //prev=curr;
    }
    return curr[capacity];
}
int main()
{
    int weight[]={4,5,1};
    int value[]={1,2,3};
    int n=3;
    int capacity=4;

    int ans=solveUsingSpaceOptimization(weight,value,n,capacity);
    cout<<"knapsack ans:"<<ans;
    return 0;
}
```

```
PS E:\C++Code> .\a.exe
knapsack ans:3
```

Dynamic Programming Class - 4

Q) partition equal subset sum



Description | Editorial | Solutions (3.1K) | Submissions

416. Partition Equal Subset Sum

Medium | 10.9K | 196

Companies

Given an integer array `nums`, return `true` *if you can partition the array into two subsets such that the sum of the elements in both subsets is equal or* `false` *otherwise.*

Example 1:

```
Input: nums = [1,5,11,5]
Output: true
Explanation: The array can be partitioned as [1, 5, 5] and [11].
```

Example 2:

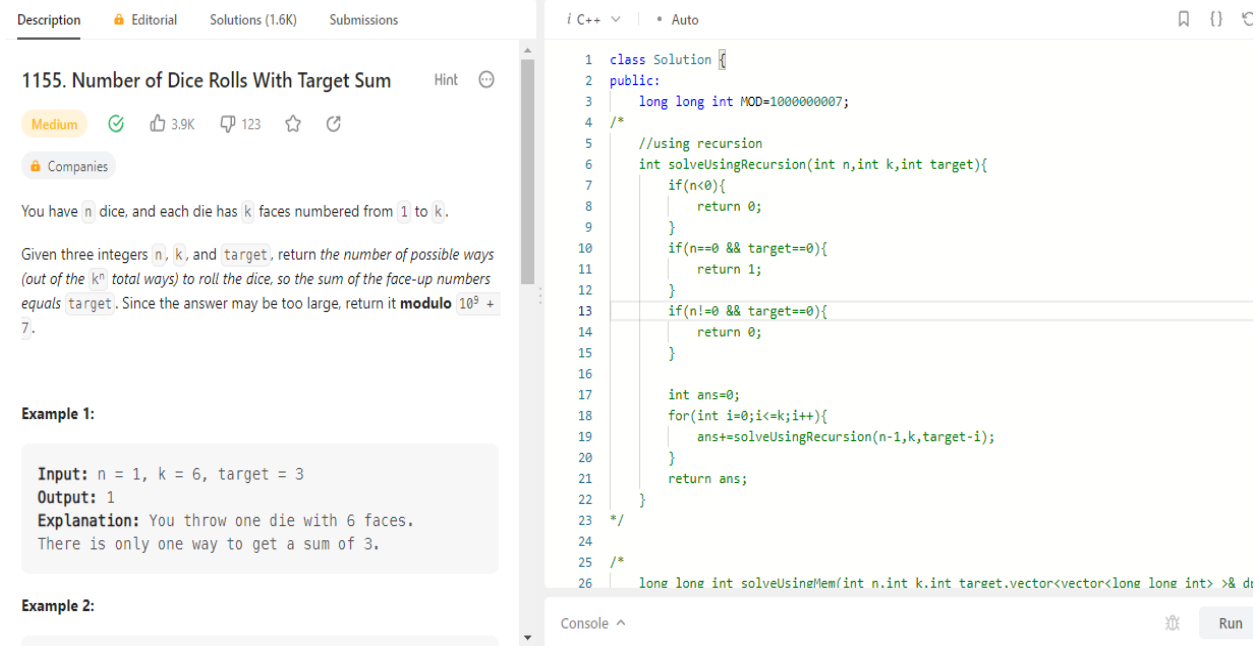
```
Input: nums = [1,2,3,5]
Output: false
```

C++ | Auto

```cpp
class Solution {
public:
/*
    //using recursion (time complexity : exponential)
    bool solveUsingRecursion(int index,vector<int>& nums,int target){
        int n=nums.size();

        if(index>=n){
            return 0;
        }

        if(target<0){
            return 0;
        }

        if(target==0){
            return 1;
        }

        bool include=solveUsingRecursion(index+1,nums,target-nums[index]);
        bool exclude=solveUsingRecursion(index+1,nums,target);

        return (include || exclude);
    }
*/
```

Console | Run

Q) 1155. Number of Dice Rolls With Target Sum



Description | Editorial | Solutions (1.6K) | Submissions

1155. Number of Dice Rolls With Target Sum Hint

Medium | 3.9K | 123

Companies

You have `n` dice, and each die has `k` faces numbered from `1` to `k`.

Given three integers `n`, `k`, and `target`, return *the number of possible ways (out of the* k^n *total ways) to roll the dice, so the sum of the face-up numbers equals* `target`. Since the answer may be too large, return it **modulo** $10^9 + 7$.

Example 1:

```
Input: n = 1, k = 6, target = 3
Output: 1
Explanation: You throw one die with 6 faces.
There is only one way to get a sum of 3.
```

Example 2:

C++ | Auto

```cpp
class Solution {
public:
    long long int MOD=1000000007;
/*
    //using recursion
    int solveUsingRecursion(int n,int k,int target){
        if(n<0){
            return 0;
        }
        if(n==0 && target==0){
            return 1;
        }
        if(n!=0 && target==0){
            return 0;
        }

        int ans=0;
        for(int i=0;i<=k;i++){
            ans+=solveUsingRecursion(n-1,k,target-i);
        }
        return ans;
    }
*/

/*
    long long int solveUsingMem(int n,int k,int target,vector<vector<long long int> >& d
```

Console | Run

Dynamic Programming – II

Dynamic Programming Class -5

375. Guess Number Higher or Lower II

Hint

Medium 1.9K 2K

Companies

We are playing the Guessing Game. The game will work as follows:

I pick a number between `1` and `n`.

You guess a number.

If you guess the right number, **you win the game**.

If you guess the wrong number, then I will tell you whether the number I picked is **higher or lower**, and you will continue guessing.

Every time you guess a wrong number `x`, you will pay `x` dollars. If you run out of money, **you lose the game**.

Given a particular `n`, return *the minimum amount of money you need to **guarantee a win regardless of what number I pick***.

Example 1:

```cpp
class Solution {
public:
/*
    //using recursion
    int solveUsingRecursion(int start,int end){
        if(start>=end){
            return 0;
        }
        int ans=INT_MAX;

        for(int i=start;i<=end;i++){
            ans=min(ans,i+max(solveUsingRecursion(start,i-1),solveUsingRecursion(i+1,end)));
        }
        return ans;
    }
*/

/*
    //using memoization
    int solveUsingMem(int start,int end,vector<vector<int> >& dp){
        if(start>=end){
            return 0;
        }
        int ans=INT_MAX;

        if(dp[start][end] !=-1){
```

Console Run

Description Editorial Solutions Submissions

1130. Minimum Cost Tree From Leaf Values

Hint

Medium 4K 258

Companies

Given an array `arr` of positive integers, consider all binary trees such that:

- Each node has either `0` or `2` children;
- The values of `arr` correspond to the values of each **leaf** in an in-order traversal of the tree.
- The value of each non-leaf node is equal to the product of the largest leaf value in its left and right subtree, respectively.

Among all possible binary trees considered, return *the smallest possible sum of the values of each non-leaf node*. It is guaranteed this sum fits into a **32-bit** integer.

A node is a **leaf** if and only if it has zero children.

C++ Auto

```cpp
class Solution {
public:
/*
    //using recursion
    int solvingUsingRecursion(vector<int>& arr ,map<pair<int,int> ,int>& maxi,int left ,int right){
        if(left==right){
            return 0;
        }
        int ans=INT_MAX;

        for(int i=left;i<right;i++){
            ans=min(ans,maxi[{left,i}]*maxi[{i+1,right}]
            + solvingUsingRecursion(arr,maxi,left,i)
            +solvingUsingRecursion(arr,maxi,i+1,right));
        }
        return ans;
    }
*/

/*
    //using memoization
    int solvingUsingMem(vector<int>& arr ,map<pair<int,int> ,int>& maxi,int left ,int right,vector<vector<in
        if(left==right){
            return 0;
        }
        int ans=INT_MAX;
```

Console Run

Dynamic Programming Class – 6

Description | Editorial | Solutions (3.9K) | Submissions

1143. Longest Common Subsequence

Hint

Medium | 11.4K | 135

Companies

Given two strings `text1` and `text2`, return *the length of their longest* ***common subsequence***. If there is no **common subsequence**, return `0`.

A **subsequence** of a string is a new string generated from the original string with some characters (can be none) deleted without changing the relative order of the remaining characters.

- For example, `"ace"` is a subsequence of `"abcde"`.

A **common subsequence** of two strings is a subsequence that is common to both strings.

Example 1:

```
Input: text1 = "abcde", text2 = "ace"
Output: 3
Explanation: The longest common subsequence is
```

C++ | Auto

```cpp
class Solution {
public:
/*
    //using recursion
    int solveUsingRecursion(string a,string b,int i,int j){
        if(i==a.length()){
            return 0;
        }
        if(j==b.length()){
            return 0;
        }

        int ans=0;

        if(a[i]==b[j]){
            ans=1+solveUsingRecursion(a,b,i+1,j+1);
        }
        else{
            ans=0+max(solveUsingRecursion(a,b,i,j+1),solveUsingRecursion(a,b,i+1,j));
        }
        return ans;
    }
*/
/*
    //using memoization
    int solveUsingMem(string& a,string& b,int i,int j,vector<vector<int> >& dp){
```

Console | Run

Description | Editorial | Solutions (2.6K) | Submissions

516. Longest Palindromic Subsequence

Medium | 8.6K | 314

Companies

Given a string `s`, find *the longest palindromic* ***subsequence****'s length in* `s`.

A **subsequence** is a sequence that can be derived from another sequence by deleting some or no elements without changing the order of the remaining elements.

Example 1:

```
Input: s = "bbbab"
Output: 4
Explanation: One possible longest palindromic
subsequence is "bbbb".
```

Example 2:

C++ | Auto

```cpp
class Solution {
public:
    int usingSpaceOptimization(string& a,string& b){

        vector<int>next(b.length()+1,0);
        vector<int>curr(b.length()+1,0);

        for(int i=a.length()-1;i>=0;i--){
            for(int j=b.length()-1;j>=0;j--){
                int ans=0;

                if(a[i]==b[j]){
                    ans=1+next[j+1];
                }
                else{
                    ans=0+max(curr[j+1],next[j]);
                }
                curr[j]= ans;
            }
            next=curr;
        }
        return next[0];
    }
    int longestPalindromeSubseq(string s) {
        string first=s;
        reverse(s.begin(),s.end());
```

Console

72. Edit Distance

Medium 13.1K 155

Companies

Given two strings `word1` and `word2`, return *the minimum number of operations required to convert `word1` to `word2`*.

You have the following three operations permitted on a word:

- Insert a character
- Delete a character
- Replace a character

Example 1:

```
Input: word1 = "horse", word2 = "ros"
Output: 3
Explanation:
horse -> rorse (replace 'h' with 'r')
rorse -> rose (remove 'r')
```

```cpp
class Solution {
public:
/*
    //using recursion
    int solveUsingRecursion(string& a, string& b,int i,int j){
        if(i==a.length()){
            return b.length()-j;
        }

        if(j==b.length()){
            return a.length()-i;
        }

        int ans=0;

        if(a[i]==b[j]){
            ans=solveUsingRecursion(a,b,i+1,j+1);
        }
        else{
            int insert=1+solveUsingRecursion(a,b,i,j+1);
            int deleted=1+solveUsingRecursion(a,b,i+1,j);
            int replace=1+solveUsingRecursion(a,b,i+1,j+1);
            ans=min(insert,min(deleted,replace));
        }
        return ans;
    }
```

Console

Dynamic Programming Class – 7

300. Longest Increasing Subsequence

Medium 17.9K 342

Companies

Given an integer array `nums`, return *the length of the longest* ***strictly increasing subsequence***.

Example 1:

```
Input: nums = [10,9,2,5,3,7,101,18]
Output: 4
Explanation: The longest increasing subsequence is [2,3,7,101], therefore the length is 4.
```

Example 2:

```
Input: nums = [0,1,0,3,2,3]
Output: 4
```

```cpp
class Solution {
public:

    //using recursion
    int solveUsingRecursion(vector<int>& arr,int curr,int prev){
        if(curr >= arr.size()){
            return 0;
        }

        int include=0;
        if(prev==-1 || arr[curr] > arr[prev]){
            include=1+solveUsingRecursion(arr,curr+1,curr);
        }

        int exclude=0+solveUsingRecursion(arr,curr+1,prev);

        int ans =max(include,exclude);
        return ans;
    }

    //using memoization
    int solveUsingMem(vector<int>& arr,int curr,int prev,vector<vector<int> >& dp){
        if(curr >= arr.size()){
            return 0;
        }
```

Console Run

1691. Maximum Height by Stacking Cuboids

Hint

Hard 943 28

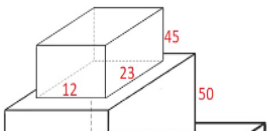
Companies

Given `n` `cuboids` where the dimensions of the `i`th cuboid is `cuboids[i] = [width_i, length_i, height_i]` (**0-indexed**). Choose a **subset** of `cuboids` and place them on each other.

You can place cuboid `i` on cuboid `j` if `width_i <= width_j` and `length_i <= length_j` and `height_i <= height_j`. You can rearrange any cuboid's dimensions by rotating it to put it on another cuboid.

Return *the* ***maximum height*** *of the stacked* `cuboids`.

Example 1:

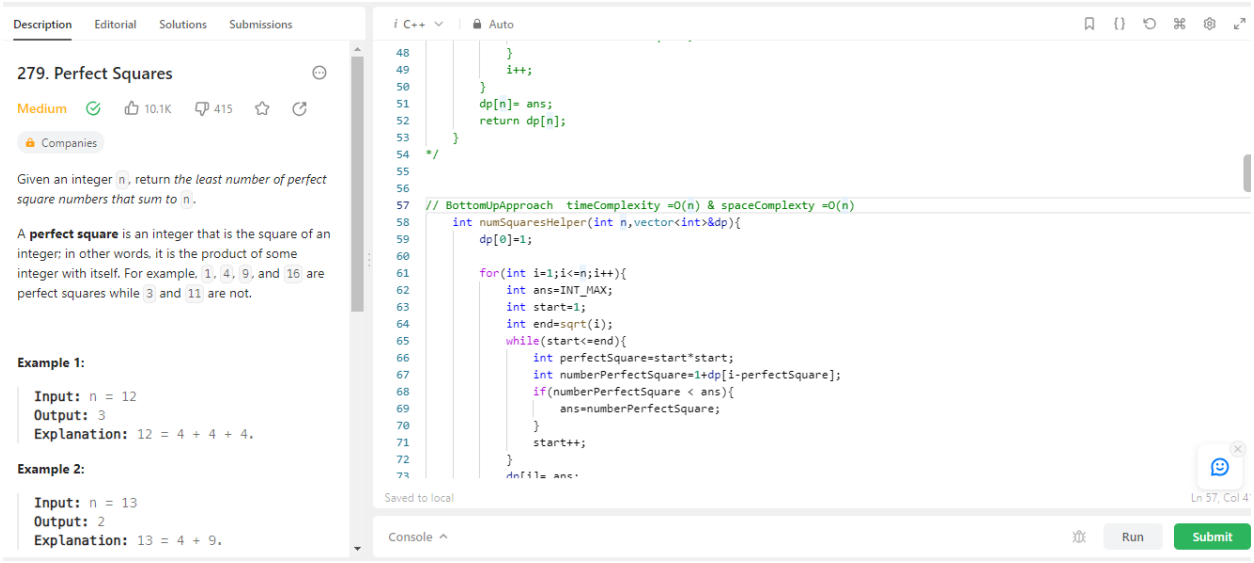



```cpp
class Solution {
public:
    bool check(vector<int>& a ,vector<int>& b){
        if(b[0]<=a[0] && b[1]<=a[1] && b[2]<=a[2]){
            return true ;
        }
        else{
            return false;
        }
    }

    //using space optimization
    int solveUsingTabSO(vector<vector<int>>& arr) {
        int n = arr.size();
        //vector<vector<int> > dp(n+1, vector<int>(n+1, 0));
        vector<int> currRow(n+1, 0);
        vector<int> nextRow(n+1, 0);

        for(int curr = n-1; curr>=0; curr--) {
            for(int prev = curr - 1; prev >= -1; prev--) {
                //include
                int include = 0;
                if(prev == -1 || check(arr[curr], arr[prev]))
                    include = arr[curr][2] + nextRow[curr + 1];
```

Console Run

DP – Assignments

Q)perfect squares



Description Editorial Solutions Submissions

279. Perfect Squares

Medium 10.1K 415

Companies

Given an integer `n`, return *the least number of perfect square numbers that sum to* `n`.

A **perfect square** is an integer that is the square of an integer; in other words, it is the product of some integer with itself. For example, `1`, `4`, `9`, and `16` are perfect squares while `3` and `11` are not.

Example 1:

```
Input: n = 12
Output: 3
Explanation: 12 = 4 + 4 + 4.
```

Example 2:

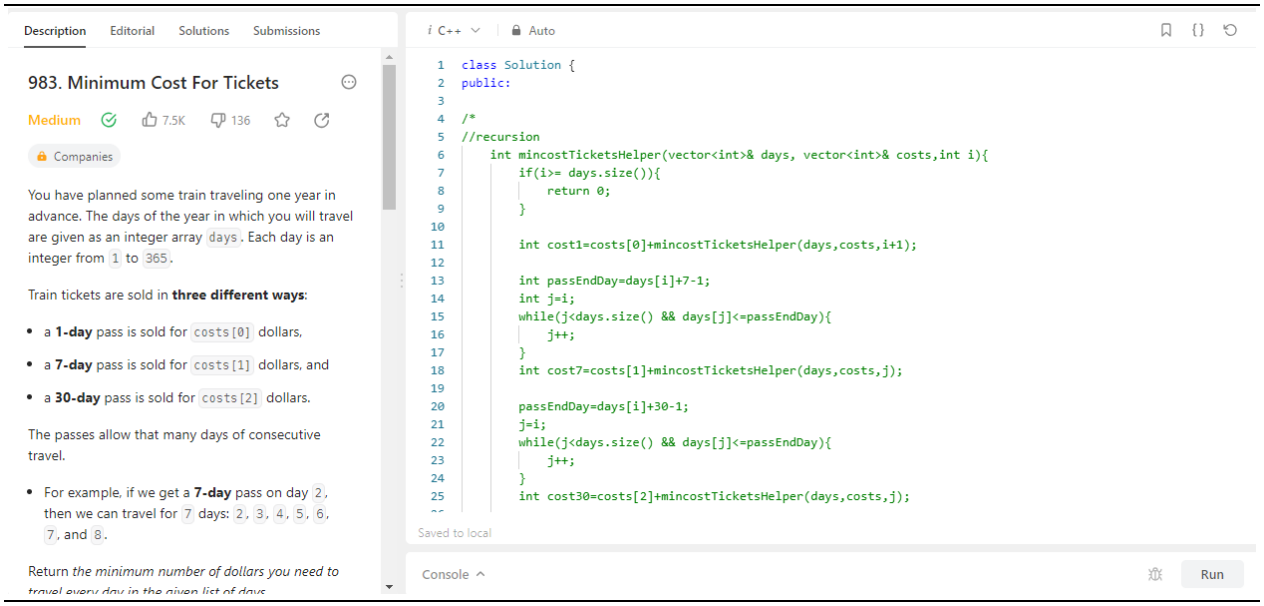
```
Input: n = 13
Output: 2
Explanation: 13 = 4 + 9.
```

C++ Auto

```
        }
        i++;
      }
      dp[n]= ans;
      return dp[n];
    }
*/

// BottomUpApproach  timeComplexity =O(n) & spaceComplexty =O(n)
    int numSquaresHelper(int n,vector<int>&dp){
        dp[0]=1;

        for(int i=1;i<=n;i++){
            int ans=INT_MAX;
            int start=1;
            int end=sqrt(i);
            while(start<=end){
                int perfectSquare=start*start;
                int numberPerfectSquare=1+dp[i-perfectSquare];
                if(numberPerfectSquare < ans){
                    ans=numberPerfectSquare;
                }
                start++;
            }
```

Saved to local Ln 57, Col 4

Console Run Submit



Description Editorial Solutions Submissions

983. Minimum Cost For Tickets

Medium 7.5K 136

Companies

You have planned some train traveling one year in advance. The days of the year in which you will travel are given as an integer array `days`. Each day is an integer from `1` to `365`.

Train tickets are sold in **three different ways**:

- a **1-day** pass is sold for `costs[0]` dollars,
- a **7-day** pass is sold for `costs[1]` dollars, and
- a **30-day** pass is sold for `costs[2]` dollars.

The passes allow that many days of consecutive travel.

- For example, if we get a **7-day** pass on day `2`, then we can travel for `7` days: `2`, `3`, `4`, `5`, `6`, `7`, and `8`.

Return *the minimum number of dollars you need to travel every day in the given list of days*

C++ Auto

```
class Solution {
public:

/*
//recursion
    int mincostTicketsHelper(vector<int>& days, vector<int>& costs,int i){
        if(i>= days.size()){
            return 0;
        }

        int cost1=costs[0]+mincostTicketsHelper(days,costs,i+1);

        int passEndDay=days[i]+7-1;
        int j=i;
        while(j<days.size() && days[j]<=passEndDay){
            j++;
        }
        int cost7=costs[1]+mincostTicketsHelper(days,costs,j);

        passEndDay=days[i]+30-1;
        j=i;
        while(j<days.size() && days[j]<=passEndDay){
            j++;
        }
        int cost30=costs[2]+mincostTicketsHelper(days,costs,j);
```

Saved to local

Console Run

Description Editorial Solutions Submissions

122. Best Time to Buy and Sell Stock II

Medium 12.6K 2.6K

Companies

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

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

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

Example 1:

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

C++ Auto

```cpp
class Solution {
public:
/*
    //using recurrsion
    int solve(vector<int>&prices,int i,int buy){
        if(i>=prices.size()){
            return 0;
        }

        int profit=0;
        if(buy){
            int buyItProfit=-prices[i]+solve(prices,i+1,0);
            int skipProfit=solve(prices,i+1,1);
            profit=max(buyItProfit,skipProfit);
        }
        else{
            int sellItProfit=prices[i]+solve(prices,i+1,1);
            int skipProfit=solve(prices,i+1,0);
            profit=max(sellItProfit,skipProfit);
        }
        return profit;
    }
*/

/* using top down
```

Saved to local

Console

Description Editorial Solutions Submissions

123. Best Time to Buy and Sell Stock III

Hard 9.1K 165

Companies

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

Find the maximum profit you can achieve. You may complete **at most two transactions**.

Note: You may not engage in multiple transactions simultaneously (i.e., you must sell the stock before you buy again).

Example 1:

```
Input: prices = [3,3,5,0,0,3,1,4]
Output: 6
Explanation: Buy on day 4 (price = 0) and sell on day 6 (price = 3), profit = 3-0 = 3.
Then buy on day 7 (price = 1) and
```

C++ Auto

```cpp
class Solution {
public:
/*
 //using recurrsion
    int solve(vector<int>&prices,int i,int buy,int limit){
        if(i>=prices.size() || limit==0){
            return 0;
        }

        int profit=0;
        if(buy){
            int buyItProfit=-prices[i]+solve(prices,i+1,0,limit);
            int skipProfit=solve(prices,i+1,1,limit);
            profit=max(buyItProfit,skipProfit);
        }
        else{

            int sellItProfit=prices[i]+solve(prices,i+1,1,limit-1);
            int skipProfit=solve(prices,i+1,0,limit);
            profit=max(sellItProfit,skipProfit);
        }
        return profit;
    }
*/

```

Saved to local

Console Run

Description | Editorial | Solutions (2.2K) | Submissions

188. Best Time to Buy and Sell Stock IV

Hard | 7.1K | 202

Companies

Q(123)

You are given an integer array `prices` where `prices[i]` is the price of a given stock on the `i`th day, and an integer `k`.

Find the maximum profit you can achieve. You may complete at most `k` transactions: i.e. you may buy at most `k` times and sell at most `k` times.

Note: You may not engage in multiple transactions simultaneously (i.e., you must sell the stock before you buy again).

Example 1:

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

Example 2:

```
Input: k = 2, prices = [3,2,6,5,0,3]
```

C++ | Auto

```cpp
class Solution {
public:
/*
    //using recurrsion
    int solve(vector<int>&prices,int i,int buy,int limit){
        if(i>=prices.size() || limit==0){
            return 0;
        }

        int profit=0;
        if(buy){
            int buyItProfit=-prices[i]+solve(prices,i+1,0,limit);
            int skipProfit=solve(prices,i+1,1,limit);
            profit=max(buyItProfit,skipProfit);
        }
        else{

            int sellItProfit=prices[i]+solve(prices,i+1,1,limit-1);
            int skipProfit=solve(prices,i+1,0,limit);
            profit=max(sellItProfit,skipProfit);
        }
        return profit;
    }
*/

```

Saved to local

Console | Run

Description | Editorial | Solutions | Submissions

714. Best Time to Buy and Sell Stock with Transaction Fee

Hint

Medium | 6.8K | 190

Companies

You are given an array `prices` where `prices[i]` is the price of a given stock on the `i`th day, and an integer `fee` representing a transaction fee.

Find the maximum profit you can achieve. You may complete as many transactions as you like, but you need to pay the transaction fee for each transaction.

Note:

- You may not engage in multiple transactions simultaneously (i.e., you must sell the stock before you buy again).
- The transaction fee is only charged once for each stock purchase and sale.

C++ | Auto

```cpp
class Solution {
public:
/*
    //using recurrsion
    int solve(vector<int>&prices,int i,int buy ,int &fee){
        if(i>=prices.size()){
            return 0;
        }

        int profit=0;
        if(buy){
            int buyItProfit=-prices[i]+solve(prices,i+1,0,fee);
            int skipProfit=solve(prices,i+1,1,fee);
            profit=max(buyItProfit,skipProfit);
        }
        else{
            int sellItProfit=prices[i]+solve(prices,i+1,1,fee)-fee;
            int skipProfit=solve(prices,i+1,0,fee);
            profit=max(sellItProfit,skipProfit);
        }
        return profit;
    }

*/

```

Saved to local

Console | Run

Description Editorial Solutions Submissions

337. House Robber III

Medium 8.2K 132

Companies

The thief has found himself a new place for his thievery again. There is only one entrance to this area, called `root`.

Besides the `root`, each house has one and only one parent house. After a tour, the smart thief realized that all houses in this place form a binary tree. It will automatically contact the police if **two directly-linked houses were broken into on the same night**.

Given the `root` of the binary tree, return *the maximum amount of money the thief can rob* ***without alerting the police***.

Example 1:

C++ Auto

```cpp
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode() : val(0), left(nullptr), right(nullptr) {}
 *     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
 *     TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left), right(right) {}
 * };
 */
class Solution {
public:
    //Recurrsion
    int solveRecurrsion(TreeNode* &root){
        if(!root){
            return 0;
        }

        int robThisHouse=0,dontRobThisHouse=0;
        robThisHouse += root->val;
        if(root->left){
            robThisHouse +=solveRecurrsion(root->left->left) + solveRecurrsion(root->left->right);
        }
        if(root->right){
```

TC = O(n)
SC = O(n)

Saved to local

Console Run

Description Editorial Solutions Submissions

486. Predict the Winner

Medium 5.6K 260

Companies

You are given an integer array `nums`. Two players are playing a game with this array: player 1 and player 2.

Player 1 and player 2 take turns, with player 1 starting first. Both players start the game with a score of `0`. At each turn, the player takes one of the numbers from either end of the array (i.e., `nums[0]` or `nums[nums.length - 1]`) which reduces the size of the array by `1`. The player adds the chosen number to their score. The game ends when there are no more elements in the array.

Return `true` if Player 1 can win the game. If the scores of both players are equal, then player 1 is still the winner, and you should also return `true`. You may assume that both players are playing optimally.

Example 1:

C++ Auto

```cpp
class Solution {
public:

    int solve(vector<int>& nums,int start,int end){
        if(start==end){
            return nums[start];
        }

        int diffByStart=nums[start]-solve(nums,start+1,end);
        int diffByEnd=nums[end]-solve(nums,start,end-1);
        return max(diffByStart,diffByEnd);
    }

    int solveTopDown(vector<int>& nums,int start,int end, vector<vector<int>>&dp){
        if(start==end){
            return nums[start];
        }

        if(dp[start][end]!=-1){
            return dp[start][end];
        }
        int diffByStart=nums[start]-solveTopDown(nums,start+1,end,dp);
        int diffByEnd=nums[end]-solveTopDown(nums,start,end-1,dp);
        dp[start][end]= max(diffByStart,diffByEnd);
        return dp[start][end];
```

Saved to local

Description Editorial Solutions Submissions

139. Word Break

Medium 16.4K 709

Companies

Given a string `s` and a dictionary of strings `wordDict`, return `true` if `s` can be segmented into a space-separated sequence of one or more dictionary words.

Note that the same word in the dictionary may be reused multiple times in the segmentation.

Example 1:

```
Input: s = "leetcode", wordDict = ["leet","code"]
Output: true
Explanation: Return true because "leetcode" can be segmented as "leet code".
```

Example 2:

C++ Auto

```cpp
class Solution {
public:
    bool check(vector<string>& wordDict,string &s){
        for(auto i:wordDict){
            if(s==i){
                return true;
            }
        }
        return false;
    }
    bool solve(string s, vector<string>& wordDict,int start){
        if(start==s.size()){
            return true;
        }

        string word="";
        bool flag=false;

        for(int i=start;i<s.size();i++){
            word+=s[i];
            if(check(wordDict,word)){
                flag=flag || solve(s,wordDict,i+1);
            }
        }
        return flag;
    }
```

Saved to local

Console Run

Description Editorial Solutions Submissions

494. Target Sum

Medium 10.3K 332

Companies

You are given an integer array `nums` and an integer `target`.

You want to build an **expression** out of nums by adding one of the symbols `'+'` and `'-'` before each integer in nums and then concatenate all the integers.

- For example, if `nums = [2, 1]`, you can add a `'+'` before `2` and a `'-'` before `1` and concatenate them to build the expression `"+2-1"`.

Return the number of different **expressions** that you can build, which evaluates to `target`.

Example 1:

```
Input: nums = [1,1,1,1,1], target = 3
Output: 5
```

C++ Auto

```cpp
class Solution {
public:
    int solveRecurrsion(vector<int>& nums, int target,int i){
        if(i==nums.size() ){
            return target==0 ? 1 : 0;
        }

        int plus=solveRecurrsion(nums, target-nums[i],i+1);
        int minus=solveRecurrsion(nums, target+nums[i],i+1);
        return plus+minus;
    }

    int solveTopDown(vector<int>& nums, int target,int i, map<pair<int,int>,int>&dp){
        if(i==nums.size() ){
            return target==0 ? 1 : 0;
        }

        if(dp.find({i,target})!=dp.end()){
            return dp[{i,target}];
        }
        int plus=solveTopDown(nums, target-nums[i],i+1,dp);
        int minus=solveTopDown(nums, target+nums[i],i+1,dp);
        return dp[{i,target}]= plus+minus;
    }
```

Saved to local

Console

Description Editorial Solutions Submissions

474. Ones and Zeroes

Medium 5.2K 433

Companies

You are given an array of binary strings `strs` and two integers `m` and `n`.

Return *the size of the largest subset of `strs` such that there are **at most** `m` `0`'s and `n` `1`'s in the subset.*

A set `x` is a **subset** of a set `y` if all elements of `x` are also elements of `y`.

Example 1:

> **Input:** strs = ["10","0001","111001","1","0"], m = 5, n = 3
> **Output:** 4
> **Explanation:** The largest subset with at most 5 0's and 3 1's is {"10", "0001", "1", "0"}, so the answer is 4.

C++ Auto

```cpp
class Solution {
public:
    void convertsToNumStrs(vector<string>& strs,vector<pair<int,int>>&numStr){
        for(auto str:strs){
            int zeros=0,ones=0;
            for(auto ch:str){
                if(ch=='0'){
                    ++zeros;
                }
                else{
                    ones++;
                }
            }
            numStr.push_back({zeros,ones});
        }
    }
    int solveRecurrsion(vector<pair<int,int>>&numStr,int i,int m, int n){
        if(i==numStr.size()){
            return 0;
        }

        int zeros=numStr[i].first;
        int ones=numStr[i].second;

        int include=0,exclude=0;
```

Restored from local Upgrade to Cloud Saving

Console

Description Editorial Solutions Submissions

5. Longest Palindromic Substring

Hint

Medium 27.9K 1.6K

Companies

Given a string `s`, return *the longest palindromic substring* in `s`.

Example 1:

> **Input:** s = "babad"
> **Output:** "bab"
> **Explanation:** "aba" is also a valid answer.

Example 2:

> **Input:** s = "cbbd"
> **Output:** "bb"

Constraints:

C++ Auto

```cpp
class Solution {
public:
    int maxlen=1,start=0;
    bool solve(string& s,int i,int j){
        if(i>=j){
            return true;
        }
        bool flag=false;
        if( s[i]==s[j]){
            flag=solve(s,i+1,j-1);
        }
        if(flag){
            int currlen=j-i+1;
            if(currlen>maxlen){
                maxlen=currlen;
                start=i;
            }
        }
        return flag;
    }

    //using topdown

    bool solveTopDown(string& s,int i,int j,vector<vector<int>>&dp){
        if(i>=j){
```

Saved to local

Console

Description Editorial Solutions Submissions

1402. Reducing Dishes

Hint

Hard 3.1K 293

Companies

A chef has collected data on the `satisfaction` level of his `n` dishes. Chef can cook any dish in 1 unit of time.

Like-time coefficient of a dish is defined as the time taken to cook that dish including previous dishes multiplied by its satisfaction level i.e. `time[i] * satisfaction[i]`.

Return the maximum sum of **like-time coefficient** that the chef can obtain after preparing some amount of dishes.

Dishes can be prepared in **any** order and the chef can discard some dishes to get this maximum value.

Example 1:

C++ Auto

```cpp
class Solution {
public:
    int solveRecurrsion(vector<int>& satisfaction,int i,int time){
        if(i==satisfaction.size()){
            return 0;
        }
        int includeLikeTimeCoefficient=time*satisfaction[i]+solveRecurrsion(satisfaction,i+1,time+1);
        int excludeLikeTimeCoefficient=solveRecurrsion(satisfaction,i+1,time);

        return max(includeLikeTimeCoefficient,excludeLikeTimeCoefficient);
    }

    int solveTopDown(vector<int>& satisfaction,int i,int time,vector<vector<int>>&dp){
        if(i==satisfaction.size()){
            return 0;
        }

        if(dp[i][time]!=-1){
            return dp[i][time];
        }
        int includeLikeTimeCoefficient=time*satisfaction[i]+solveTopDown(satisfaction,i+1,time+1,dp);
        int excludeLikeTimeCoefficient=solveTopDown(satisfaction,i+1,time,dp);

        return dp[i][time]= max(includeLikeTimeCoefficient,excludeLikeTimeCoefficient);
    }
```

Saved to local

Description Editorial Solutions (3.7K) Submissions

140. Word Break II

Hard 6.5K 520

Companies

Given a string `s` and a dictionary of strings `wordDict`, add spaces in `s` to construct a sentence where each word is a valid dictionary word. Return all such possible sentences in **any order**.

Note that the same word in the dictionary may be reused multiple times in the segmentation.

Example 1:

> **Input:** `s = "catsanddog", wordDict = ["cat","cats","and","sand","dog"]`
> **Output:** `["cats and dog","cat sand dog"]`

Example 2:

> **Input:** `s = "pineapplepenapple", wordDict = ["apple","pen","applepen","pine","pineapple"]`
> **Output:** `["pine apple pen apple","pineapple pen apple","pine applepen apple"]`

C++ Auto

```cpp
class Solution {
public:
    vector<string> solve(string s, unordered_map<string,bool>&dict,int i){
        if(i==s.size()){
            return{""};
        }

        vector<string>ans;
        string word;

        for(int j=i;j<s.size();j++){
            word.push_back(s[j]);
            if(dict.find(word)==dict.end()){
                continue;
            }

            auto right=solve(s,dict,j+1);
            for(auto eachRightPart :right){
                string endPart;
                if(eachRightPart.size()>0){
                    endPart=" "+eachRightPart;
                }
                ans.push_back(word+endPart);
            }
        }
```

Saved to local

Console Run

Description | Editorial | Solutions (2.1K) | Submissions

115. Distinct Subsequences

Hard | 6.3K | 276

Companies

Given two strings s and t, return *the number of distinct* ***subsequences*** *of* s *which equals* t.

The test cases are generated so that the answer fits on a 32-bit signed integer.

Example 1:

Input: s = "rabbbit", t = "rabbit"
Output: 3
Explanation:
As shown below, there are 3 ways you can generate "rabbit" from s.
rabbbit
rabbbit
rabbbit

Example 2:

C++ | Auto

```cpp
class Solution {
public:
    int solveRecurrsion(string& s, string& t,int i,int j){
        if(j==t.size()){
            return 1;
        }
        if(i==s.size()){
            return 0;
        }
        int ans=0;
        if(s[i]==t[j]){
            ans+=solveRecurrsion(s,t,i+1,j+1);
        }
        ans+=solveRecurrsion(s,t,i+1,j);
        return ans;
    }

    int solveTopDown(string& s, string& t,int i,int j,vector<vector<int>>&dp){
        if(j==t.size()){
            return 1;
        }
        if(i==s.size()){
            return 0;
        }
```

Saved to local

Console

Description | Editorial | Solutions | Submissions

97. Interleaving String

Medium | 7.8K | 450

Companies

Given strings `s1`, `s2`, and `s3`, find whether `s3` is formed by an **interleaving** of `s1` and `s2`.

An **interleaving** of two strings `s` and `t` is a configuration where `s` and `t` are divided into `n` and `m` substrings respectively, such that:

- $s = s_1 + s_2 + ... + s_n$
- $t = t_1 + t_2 + ... + t_m$
- $|n - m| <= 1$
- The **interleaving** is $s_1 + t_1 + s_2 + t_2 + s_3 + t_3 + ...$ or $t_1 + s_1 + t_2 + s_2 + t_3 + s_3 + ...$

Note: `a + b` is the concatenation of strings `a` and `b`.

C++ | Auto

```cpp
class Solution {
public:
    int solveRecurrsion(string& s1, string& s2, string& s3,int i,int j,int k){
        if(i==s1.size() && j==s2.size() && k==s3.size()){
            return true;
        }

        bool flag=false;
        if(i<s1.size() && s1[i]==s3[k]){
            flag=flag || solveRecurrsion(s1,s2,s3,i+1,j,k+1);
        }
        if(j<s2.size() && s2[j]==s3[k]){
            flag=flag || solveRecurrsion(s1,s2,s3,i,j+1,k+1);
        }
        return flag;
    }

    int solveTopDown(string& s1, string& s2, string& s3,int i,int j,int k,vector<vector<vector<int>>>&dp){
        if(i==s1.size() && j==s2.size() && k==s3.size()){
            return true;
        }
        if(dp[i][j][k]!=-1){
            return dp[i][j][k];
        }
        bool flag=false;
```

Saved to local

Console | Run

Description Editorial Solutions Submissions

1312. Minimum Insertion Steps to Make a String Palindrome

Hint

Hard 4.7K 61

Companies

Given a string `s`. In one step you can insert any character at any index of the string.

Return *the minimum number of steps* to make `s` palindrome.

A **Palindrome String** is one that reads the same backward as well as forward.

Example 1:

```
Input: s = "zzazz"
Output: 0
Explanation: The string "zzazz" is already palindrome we do not need any insertions.
```

C++ Auto

```cpp
class Solution {
public:
    int usingSpaceOptimization(string& a,string& b){

        vector<int>next(b.length()+1,0);
        vector<int>curr(b.length()+1,0);

        for(int i=a.length()-1;i>=0;i--){
            for(int j=b.length()-1;j>=0;j--){
                int ans=0;

                if(a[i]==b[j]){
                    ans=1+next[j+1];
                }
                else{
                    ans=0+max(curr[j+1],next[j]);
                }
                curr[j]= ans;
            }
            next=curr;
        }
        return next[0];
    }
    int longestPalindromeSubseq(string s) {
        string first=s;
```

Saved to local

Console

Problem List Dynamic Layout

Description Editorial Solutions Submissions

1671. Minimum Number of Removals to Make Mountain Array

Hint

Hard 1.5K 20

Companies

You may recall that an array `arr` is a **mountain array** if and only if:

- `arr.length >= 3`
- There exists some index `i` (**0-indexed**) with `0 < i < arr.length - 1` such that:
  - `arr[0] < arr[1] < ... < arr[i - 1] < arr[i]`
  - `arr[i] > arr[i + 1] > ... > arr[arr.length - 1]`

Given an integer array `nums`, return *the* ***minimum*** *number of elements to remove to make* `nums` *a* ***mountain array***.

C++ Auto

```cpp
class Solution {
public:
    int solveOptimal(vector<int> &arr,vector<int> &lis){
        if(arr.size()==0){
            return 0;
        }

        vector<int>ans;

        lis.push_back(1);

        ans.push_back(arr[0]);

        for(int i=1;i<arr.size();i++){

            if(arr[i]>ans.back()){
                ans.push_back(arr[i]);

                lis.push_back(ans.size());
            }
            else{
            int index=lower_bound(ans.begin(),ans.end(),arr[i])-ans.begin();
            ans[index]=arr[i];

            lis.push_back(index+1);
```

Saved to local

Description Editorial Solutions Submissions

354. Russian Doll Envelopes

Hard 5.7K 138

Companies

You are given a 2D array of integers `envelopes` where `envelopes[i] = [wᵢ, hᵢ]` represents the width and the height of an envelope.

One envelope can fit into another if and only if both the width and height of one envelope are greater than the other envelope's width and height.

Return *the maximum number of envelopes you can Russian doll (i.e., put one inside the other)*.

Note: You cannot rotate an envelope.

Example 1:

```
Input: envelopes = [[5,4],[6,4],[6,7],[2,3]]
Output: 3
Explanation: The maximum number of
```

C++ Auto

```cpp
class Solution {
public:
    int solveRecurrsion(vector<vector<int>>& envelopes,int prev,int i){
        if(i==envelopes.size()){
            return 0;
        }

        int include=INT_MIN;
        if(prev==-1 || envelopes[prev][0]<envelopes[i][0] && envelopes[prev][1]<envelopes[i][1] ){
            include=1+solveRecurrsion(envelopes,i,i+1);
        }
        int exclude=solveRecurrsion(envelopes,prev,i+1);
        return max(include,exclude);
    }
    int solveTopDown(vector<vector<int>>& envelopes,int prev,int i,vector<vector<int>>&dp){
        if(i==envelopes.size()){
            return 0;
        }

        if(dp[prev+1][i]!=-1){
            return dp[prev+1][i];
        }
        int include=INT_MIN;
        if(prev==-1 || envelopes[prev][0]<envelopes[i][0] && envelopes[prev][1]<envelopes[i][1] ){
            include=1+solveTopDown(envelopes,i,i+1,dp);
```

Saved to local

Console Run

Description Editorial Solutions Submissions

312. Burst Balloons

Hard 8.6K 218

Companies

You are given `n` balloons, indexed from `0` to `n - 1`. Each balloon is painted with a number on it represented by an array `nums`. You are asked to burst all the balloons.

If you burst the `iᵗʰ` balloon, you will get `nums[i - 1] * nums[i] * nums[i + 1]` coins. If `i - 1` or `i + 1` goes out of bounds of the array, then treat it as if there is a balloon with a `1` painted on it.

Return *the maximum coins you can collect by bursting the balloons wisely*.

Example 1:

```
Input: nums = [3,1,5,8]
Output: 167
Explanation:
nums = [3,1,5,8] --> [3,5,8] -->
```

C++ Auto

```cpp
class Solution {
public:
    int solveRecurrsion(vector<int>& nums,int start,int end){
        if(start>end){
            return 0;
        }

        int coins=INT_MIN;
        for(int i=start;i<=end;++i){
            coins=max(coins,nums[start-1]*nums[i]*nums[end+1]
                +solveRecurrsion(nums,start,i-1)
                +solveRecurrsion(nums,i+1,end)
            );
        }
        return coins;
    }

    int solveTopDown(vector<int>& nums,int start,int end,vector<vector<int>>&dp){
        if(start>end){
            return 0;
        }
        if(dp[start][end]!=-1){
            return dp[start][end];
        }
        int coins=INT_MIN;
```

Saved to local

Console

Description Editorial Solutions Submissions

877. Stone Game

Medium 3.1K 2.8K

Companies

Alice and Bob play a game with piles of stones. There are an **even** number of piles arranged in a row, and each pile has a **positive** integer number of stones `piles[i]`.

The objective of the game is to end with the most stones. The **total** number of stones across all the piles is **odd**, so there are no ties.

Alice and Bob take turns, with **Alice starting first**. Each turn, a player takes the entire pile of stones either from the **beginning** or from the **end** of the row. This continues until there are no more piles left, at which point the person with the **most stones wins**.

Assuming Alice and Bob play optimally, return `true` *if Alice wins the game, or* `false` *if Bob wins.*

Example 1:

C++ Auto

```cpp
class Solution {
public:
    bool stoneGame(vector<int>& piles) {
        return true;
    }
};
```

Saved to local

Console

Description Editorial Solutions Submissions

1140. Stone Game II

Hint

Medium 2.7K 601

Companies

Alice and Bob continue their games with piles of stones. There are a number of piles **arranged in a row**, and each pile has a positive integer number of stones `piles[i]`. The objective of the game is to end with the most stones.

Alice and Bob take turns, with Alice starting first. Initially, `M = 1`.

On each player's turn, that player can take **all the stones** in the **first** `X` remaining piles, where `1 <= X <= 2M`. Then, we set `M = max(M, X)`.

The game continues until all the stones have been taken.

Assuming Alice and Bob play optimally, return the maximum number of stones Alice can get.

C++ Auto

```cpp
class Solution {
public:
    int solveRecurrsion(vector<int>& piles,int i,int M,int alice){
        if(i==piles.size()){
            return 0;
        }

        int ans=alice? INT_MIN :INT_MAX;
        int total=0;

        for(int X=1;X<=2*M;X++){
            if(i+X-1>=piles.size()){
                break;
            }

            total+=piles[i+X-1];

            if(alice){
                ans=max(ans,total+solveRecurrsion(piles,i+X,max(X,M),!alice));
            }
            else{
                ans=min(ans,solveRecurrsion(piles,i+X,max(X,M),!alice));
            }
        }
        return ans;
```

Saved to local

Console

Description | Editorial | Solutions | Submissions

1406. Stone Game III

Hint

Hard | 2.1K | 68

Companies

Alice and Bob continue their games with piles of stones. There are several stones **arranged in a row**, and each stone has an associated value which is an integer given in the array `stoneValue`.

Alice and Bob take turns, with Alice starting first. On each player's turn, that player can take `1`, `2`, or `3` stones from the **first** remaining stones in the row.

The score of each player is the sum of the values of the stones taken. The score of each player is `0` initially.

The objective of the game is to end with the highest score, and the winner is the player with the highest score and there could be a tie. The game continues until all the stones have been taken.

Assume Alice and Bob **play optimally**.

Return `"Alice"` *if Alice will win.* `"Bob"` *if Bob will*

C++ | Auto

```cpp
class Solution {
public:
    int solveRecurrsion(vector<int>& sv,int i){
        if(i==sv.size()){
            return 0;
        }
        int ans=INT_MIN;
        int total=0;

        for(int X=1;X<=3;X++){
            if(i+X-1>=sv.size()){
                break;
            }

            total+=sv[i+X-1];
            ans=max(ans,total - solveRecurrsion(sv,i+X));
        }
        return ans;
    }

    int solveTopDown(vector<int>& sv,int i,vector<int>&dp){
        if(i==sv.size()){
            return 0;
        }
```

Saved to local

Console

Graphs-I 9/11/23

Graphs Class - 1

Q) Write a program for adjacency matrix

```cpp
#include <iostream>
#include <queue>

using namespace std;

int main()
{
    int n;
    cout<<"enter the number of nodes :"<<endl;
    cin>>n;

    vector<vector<int> >adj(n,vector<int>(n,0));

    int e;
    cout<<"enter the number of edges :"<<endl;
    cin>>e;

    for(int i=0;i<e;i++){
        int u,v;
        cin>>u>>v;
        adj[u][v]=1;
    }

    cout<<"print the adjance: "<<endl;

    for(int i=0;i<n;i++){
        for(int j=0;j<n;j++){
            cout<<adj[i][j]<<" ";
        }
        cout<<endl;
    }
    return 0;
}
```

```
PS E:\C++Code> g++ .\f77.cpp
PS E:\C++Code> .\a.exe
enter the number of nodes :
3
enter the number of edges :
6
0 1
1 0
1 2
2 1
0 2
2 0
print the adjance:
0 1 1
1 0 1
1 1 0
```

Q) write program for adjacency list using undirect graph

```cpp
#include <iostream>
#include <vector>
#include <unordered_map>
#include <list>

using namespace std;

class Graph{

    public:
    unordered_map<int,list<int> >adjList;

    void addEdge(int u,int v,bool direction){

        adjList[u].push_back(v);

        if(direction==0){
            adjList[v].push_back(u);
        }
    }

    void printAdjacencyList(){
        for(auto node :adjList){
            cout<<node.first<<"-> ";
            for(auto neighbours: node.second){
                cout<<neighbours<<", ";
            }
            cout<<endl;
        }
    }
};

int main()
{
    Graph g;

    g.addEdge(0,1,0);
    g.addEdge(1,2,0);
    g.addEdge(0,2,0);

    g.printAdjacencyList();
    return 0;
}
```

```
PS E:\C++Code> g++ .\f77.cpp
PS E:\C++Code> .\a.exe
2-> 1, 0,
1-> 0, 2,
0-> 1, 2,
```

Q) write program for adjacency list using direct graph

```cpp
#include <iostream>
#include <vector>
#include <unordered_map>
#include <list>

using namespace std;

class Graph{

    public:
    unordered_map<int,list<int> >adjList;

    void addEdge(int u,int v,bool direction){

        adjList[u].push_back(v);
        if(direction==0){
            adjList[v].push_back(u);
        }
    }

    void printAdjacencyList(){
        for(auto node :adjList){
            cout<<node.first<<"-> ";
            for(auto neighbours: node.second){
                cout<<neighbours<<", ";
            }
            cout<<endl;
        }
    }
};

int main()
{
    Graph g;

    g.addEdge(0,1,1);
    g.addEdge(1,2,1);
    g.addEdge(0,2,1);

    g.printAdjacencyList();
    return 0;
}
```


PS E:\C++Code> g++ .\f77.cpp
PS E:\C++Code> .\a.exe
1-> 2,
0-> 1, 2,

Q)write program for weighted graph

```cpp
#include <iostream>
#include <vector>
#include <unordered_map>
#include <list>

using namespace std;

class Graph{

    public:
    unordered_map<int,list<pair<int,int> > >adjList;

    void addEdge(int u,int v,int weight,bool direction){

        adjList[u].push_back({v,weight});
        if(direction==0){
            adjList[v].push_back({u,weight});
        }
    }

    void printAdjacencyList(){
        for(auto node :adjList){
            cout<<node.first<<"-> ";
            for(auto neighbours: node.second){
                cout<<"("<<neighbours.first<<", "<<neighbours.second<<"),";
            }
            cout<<endl;
        }
    }
};

int main()
{
    Graph g;

    g.addEdge(0,1,5,1);
    g.addEdge(1,2,8,1);
    g.addEdge(0,2,6,1);

    g.printAdjacencyList();
    return 0;
}
```

```
PS E:\C++Code> g++ .\f77.cpp
PS E:\C++Code> .\a.exe
1-> (2, 8),
0-> (1, 5),(2, 6),
```

Q) write program for Generic Type

```cpp
#include <iostream>
#include <vector>
#include <unordered_map>
#include <list>

using namespace std;
template <typename T>

class Graph{

    public:
    unordered_map<T,list<T> > adjList;

    void addEdge(T u,T v,bool direction){

        adjList[u].push_back(v);
        if(direction==0){
            adjList[v].push_back(u);
        }
    }

    void printAdjacencyList(){
        for(auto node :adjList){
            cout<<node.first<<"-> ";
            for(auto neighbours: node.second){
                cout<<neighbours<<", ";
            }
            cout<<endl;
        }
    }
};
int main()
{
    Graph <char>g;

    g.addEdge('a','b',0);
    g.addEdge('b','c',0);
    g.addEdge('a','c',0);

    g.printAdjacencyList();
    return 0;
}
```

```
PS E:\C++Code> g++ .\f77.cpp
PS E:\C++Code> .\a.exe
c-> b, a,
b-> a, c,
a-> b, c,
```

Adjacency matrix : space complexity : worst case $O(n^2)$

space complexity : Average case $O(v^2)$

Adjacency List : space complexity : worst case $O(v^2)$

space complexity : Average case $O(v + e)$

Q) write program for BFS Traversal

```cpp
#include <iostream>
#include <vector>
#include <unordered_map>
#include <list>
#include <queue>

using namespace std;

template <typename T>

class Graph{

    public:
    unordered_map<T,list<T> > adjList;

    void addEdge(T u,T v,bool direction){

        adjList[u].push_back(v);
        if(direction==0){
            adjList[v].push_back(u);
        }
    }

    void printAdjacencyList(){
        for(auto node :adjList){
            cout<<node.first<<"-> ";
            for(auto neighbours: node.second){
                cout<<neighbours<<", ";
            }
            cout<<endl;
        }
    }

    void bfs(T src){
        queue<T>q;
        unordered_map<T,bool>visited;

        q.push(src);
        visited[src]=true;
```

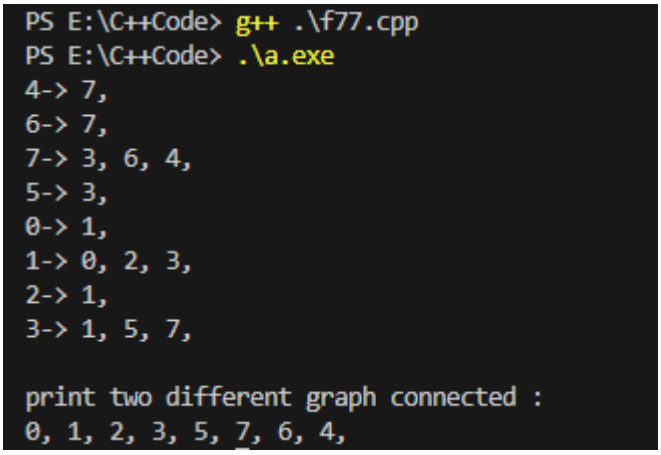
```
        while(!q.empty()){
            int frontNode=q.front();
            q.pop();
            cout<<frontNode<<", ";

            for(auto neighbours : adjList[frontNode]){
                if(!visited[neighbours]){
                    q.push(neighbours);
                    visited[neighbours]=true;
                }
            }
        }
    }
};

int main()
{
    Graph <int>g;

    g.addEdge(0,1,0);
    g.addEdge(1,2,0);
    g.addEdge(1,3,0);
    g.addEdge(3,5,0);
    g.addEdge(3,7,0);
    g.addEdge(7,6,0);
    g.addEdge(7,4,0);

    g.printAdjacencyList();
    cout<<endl<<"print bfs "<<endl;
    g.bfs(0);

    return 0;// time Complexity :O(v+e)//v is number of node,e is number of edge
}
```

```
PS E:\C++Code> g++ .\f77.cpp
PS E:\C++Code> .\a.exe
4-> 7,
6-> 7,
7-> 3, 6, 4,
5-> 3,
0-> 1,
1-> 0, 2, 3,
2-> 1,
3-> 1, 5, 7,

print bfs
0, 1, 2, 3, 5, 7, 6, 4,
```

Q) write program for two diferent graph connected (BFS)

```cpp
#include <iostream>
#include <vector>
#include <unordered_map>
#include <list>
#include <queue>

using namespace std;

template <typename T>

class Graph{

    public:
    unordered_map<T,list<T> > adjList;

    void addEdge(T u,T v,bool direction){

        adjList[u].push_back(v);
        if(direction==0){
            adjList[v].push_back(u);
        }
    }

    void printAdjacencyList(){
        for(auto node :adjList){
            cout<<node.first<<"-> ";
            for(auto neighbours: node.second){
                cout<<neighbours<<", ";
            }
            cout<<endl;
        }
    }

    void bfs(T src,unordered_map<int,bool>& visited){
        queue<T>q;

        q.push(src);
        visited[src]=true;
```

```
        while(!q.empty()){
            int frontNode=q.front();
            q.pop();
            cout<<frontNode<<", ";

            for(auto neighbours : adjList[frontNode]){
                if(!visited[neighbours]){
                    q.push(neighbours);
                    visited[neighbours]=true;
                }
            }
        }
    }
};

int main()
{
    Graph <int>g;
    int n=8;

    g.addEdge(0,1,0);
    g.addEdge(1,2,0);
    g.addEdge(1,3,0);
    g.addEdge(3,5,0);
    g.addEdge(3,7,0);
    g.addEdge(7,6,0);
    g.addEdge(7,4,0);

    g.printAdjacencyList();
    cout<<endl<<"print two different graph connected :"<<endl;

    unordered_map<int,bool>visited;
    for(int i=0;i<n;i++){
        if(!visited[i]){
            g.bfs(i,visited);
        }
    }

    return 0;
}
```



Time complexity :O(v+e) //liner time complexity

Q) write program for BFS traversal & DFS traversal with two different graph

```cpp
#include <iostream>
#include <vector>
#include <unordered_map>
#include <list>
#include <queue>

using namespace std;

template <typename T>

class Graph{

    public:
    unordered_map<T,list<T> > adjList;

    void addEdge(T u,T v,bool direction){

        adjList[u].push_back(v);
        if(direction==0){
            adjList[v].push_back(u);
        }
    }

    void printAdjacencyList(){
        for(auto node :adjList){
            cout<<node.first<<"-> ";
            for(auto neighbours: node.second){
                cout<<neighbours<<", ";
            }
            cout<<endl;
        }
    }

    void bfs(T src,unordered_map<int,bool>& visited){
        queue<T>q;

        q.push(src);
        visited[src]=true;
```

```
        while(!q.empty()){
            int frontNode=q.front();
            q.pop();
            cout<<frontNode<<", ";

            for(auto neighbours : adjList[frontNode]){
                if(!visited[neighbours]){
                    q.push(neighbours);
                    visited[neighbours]=true;
                }
            }
        }
    }

    void dfs(T src,unordered_map<int,bool>& visited){
        cout<<src<<", ";
        visited[src]=true;

        for(auto neighbour: adjList[src]){
            if(!visited[neighbour]){
                dfs(neighbour,visited);
            }
        }
    }
};

int main()
{
    Graph <int>g;
    int n=5;

    g.addEdge(0,1,0);
    g.addEdge(1,3,0);
    g.addEdge(0,2,0);
    g.addEdge(2,4,0);

    g.printAdjacencyList();
    cout<<endl<<"print two different graph connected (BFS TRAVERSAL):"<<endl;

    unordered_map<int,bool>visited;
    for(int i=0;i<n;i++){
        if(!visited[i]){
            g.bfs(i,visited);
        }
    }
```

```
    cout<<endl<<"print two different graph connected (DFS TRAVERSAL):"<<endl;
    unordered_map<int,bool>visited2;
    for(int i=0;i<n;i++){
        if(!visited2[i]){
            g.dfs(i,visited2);
        }
    }
    return 0;
}
```

```
print two different graph connected :
0, 1, 2, 3, 5, 7, 6, 4,
PS E:\C++Code> g++ .\f77.cpp
PS E:\C++Code> .\a.exe
4-> 2,
0-> 1, 2,
1-> 0, 3,
3-> 1,
2-> 0, 4,

print two different graph connected (BFS TRAVERSAL):
0, 1, 2, 3, 4,
print two different graph connected (DFS TRAVERSAL):
0, 1, 3, 2, 4,
```

Graphs Class – 2

Q) write program for check cycle graph using BFS (undirect graph)

```
#include <iostream>
#include <vector>
#include <unordered_map>
#include <list>
#include <queue>

using namespace std;

template <typename T>

class Graph{

    public:
    unordered_map<T,list<T> > adjList;

    void addEdge(T u,T v,bool direction){

        adjList[u].push_back(v);
        if(direction==0){
            adjList[v].push_back(u);
        }
    }

    void printAdjacencyList(){
        for(auto node :adjList){
            cout<<node.first<<"-> ";
            for(auto neighbours: node.second){
                cout<<neighbours<<", ";
            }
            cout<<endl;
        }
    }

    bool checkCyclicUsingBfs(int src,unordered_map<int,bool>& visited){
        queue<int>q;
        unordered_map<int,bool>parent;

        q.push(src);
        visited[src]=true;
        parent[src]=-1;
```

```
        while(!q.empty()){
            int frontNode=q.front();
            q.pop();

            for(auto neighbour : adjList[frontNode]){
                if(!visited[neighbour]){
                    q.push(neighbour);

                    visited[neighbour]=true;
                    parent[neighbour]=frontNode;
                }
                else{
                    if(visited[neighbour] && neighbour!=parent[frontNode]){
                        return true;
                    }
                }
            }
        }
        return false;
    }
};

int main()
{
    Graph <int>g;
    int n=5;

    g.addEdge(0,1,0);
    g.addEdge(1,2,0);
    g.addEdge(2,3,0);
    g.addEdge(3,4,0);
    g.addEdge(4,0,0);

    g.printAdjacencyList();
    cout<<endl;

    bool ans=false;

    unordered_map<int,bool>visited;
```

```
    for(int i=0;i<n;i++){
        if(!visited[i]){
            ans=g.checkCyclicUsingBfs(i,visited);
            if(ans==true){
                break;
            }
        }
    }

    if(ans=true){
        cout<<"cycle is present"<<endl;
    }
    else{
        cout<<"cycle is NOT present"<<endl;
    }
    return 0;
}
```

```
PS E:\C++Code> g++ .\f77.cpp
PS E:\C++Code> .\a.exe
4-> 3, 0,
0-> 1, 4,
1-> 0, 2,
2-> 1, 3,
3-> 2, 4,

cycle is present
```

Time complexity=O(v+e)

Q) write program for check cycle graph using BFS & DFS (undirect graph)

```cpp
#include <iostream>
#include <vector>
#include <unordered_map>
#include <list>
#include <queue>

using namespace std;

template <typename T>

class Graph{

    public:
    unordered_map<T,list<T> > adjList;

    void addEdge(T u,T v,bool direction){

        adjList[u].push_back(v);
        if(direction==0){
            adjList[v].push_back(u);
        }
    }

    void printAdjacencyList(){
        for(auto node :adjList){
            cout<<node.first<<"-> ";
            for(auto neighbours: node.second){
                cout<<neighbours<<", ";
            }
            cout<<endl;
        }
    }

    bool checkCyclicUsingBfs(int src,unordered_map<int,bool>& visited){
        queue<int>q;
        unordered_map<int,bool>parent;

        q.push(src);
        visited[src]=true;
        parent[src]=-1;
```

```
        while(!q.empty()){
            int frontNode=q.front();
            q.pop();

            for(auto neighbour : adjList[frontNode]){
                if(!visited[neighbour]){
                    q.push(neighbour);

                    visited[neighbour]=true;
                    parent[neighbour]=frontNode;
                }
                else{
                    if(visited[neighbour] && neighbour!=parent[frontNode]){
                        return true;
                    }
                }
            }
        }
        return false;
    }

    bool checkCyclicUsingDfs(int src,unordered_map<int,bool>& visited,int
parent){
        visited[src]=true;

        for(auto neighbour : adjList[src]){
            if(!visited[neighbour]){
                bool checkAageKaAns=checkCyclicUsingDfs(neighbour,visited,src);
                if(checkAageKaAns==true){
                    return true;
                }

                if(visited[neighbour] && neighbour!=parent){
                    return true;
                }
            }
        }
        return false;
    }
};
```

```cpp
int main()
{
    Graph <int>g;
    int n=5;

    g.addEdge(0,1,0);
    g.addEdge(1,2,0);
    g.addEdge(2,3,0);
    g.addEdge(3,4,0);
    g.addEdge(4,0,0);

    g.printAdjacencyList();
    cout<<endl;

    bool ans=false;

    unordered_map<int,bool>visited;
    for(int i=0;i<n;i++){
        if(!visited[i]){
            ans=g.checkCyclicUsingBfs(i,visited);
            if(ans==true){
                break;
            }
        }
    }

    if(ans=true){
        cout<<"cycle is present"<<endl;
    }
    else{
        cout<<"cycle is NOT present"<<endl;
    }

    //check cycle using DFS
    bool ans2=false;

    unordered_map<int,bool>visited2;
    for(int i=0;i<n;i++){
        if(!visited2[i]){
            ans2=g.checkCyclicUsingDfs(i,visited,-1);
            if(ans2==true){
                break;
            }
        }
    }
```

```
    if(ans2=true){
        cout<<"cycle is present"<<endl;
    }
    else{
        cout<<"cycle is NOT present"<<endl;
    }
    return 0;
}
```

```
PS E:\C++Code> g++ .\f77.cpp
PS E:\C++Code> .\a.exe
4-> 3, 0,
0-> 1, 4,
1-> 0, 2,
2-> 1, 3,
3-> 2, 4,

cycle is present
cycle is present
```

Q) check cycle direct graph using DFS

```
#include <iostream>
#include <vector>
#include <unordered_map>
#include <list>
#include <queue>

using namespace std;

template <typename T>

class Graph{

    public:
    unordered_map<T,list<T> > adjList;

    void addEdge(T u,T v,bool direction){

        adjList[u].push_back(v);
        if(direction==0){
            adjList[v].push_back(u);
        }
    }

    void printAdjacencyList(){
        for(auto node :adjList){
            cout<<node.first<<"-> ";
            for(auto neighbours: node.second){
                cout<<neighbours<<", ";
            }
            cout<<endl;
        }
    }

    bool checkCyclicDirectGraphUsingDfs(int src,unordered_map<int,bool>&
visited,unordered_map<int,bool> dfsVisited){

        visited[src]=true;
        dfsVisited[src]=true;
```

```
        for(auto neighbour: adjList[src]){
            if(!visited[neighbour]){
                bool
aageKaAns=checkCyclicDirectGraphUsingDfs(neighbour,visited,dfsVisited);

                if(aageKaAns==true){
                    return true;
                }
            }

            if(visited[neighbour]==true && dfsVisited[neighbour]==true){
                return true;
            }
        }

        dfsVisited[src]=false;
        return false;
    }
};

int main()
{
    Graph <int>g;
    int n=5;

    g.addEdge(0,1,1);
    g.addEdge(1,2,1);
    g.addEdge(2,3,1);
    g.addEdge(3,4,1);
    g.addEdge(4,0,1);

    g.printAdjacencyList();
    cout<<endl;

    bool ans=false;

    unordered_map<int,bool>visited;
    unordered_map<int,bool>dfsVisited;
```

```
    for(int i=0;i<n;i++){
        if(!visited[i]){
            ans=g.checkCyclicDirectGraphUsingDfs(i,visited,dfsVisited);
            if(ans==true){
                break;
            }
        }
    }

    if(ans == true)
        cout << "Cycle is Present" << endl;
    else
        cout << "Cycle Absent" << endl;


    return 0;
}
```

```
PS E:\C++Code> g++ .\f77.cpp
PS E:\C++Code> .\a.exe
4-> 0,
0-> 1,
1-> 2,
2-> 3,
3-> 4,

Cycle is Present
```

Graphs Class – 3

Q) write program for Topological Sort Order using DFS

```
#include <iostream>
#include <vector>
#include <unordered_map>
#include <list>
#include <queue>
#include <stack>

using namespace std;

template <typename T>

class Graph{

    public:
    unordered_map<T,list<T> > adjList;

    void addEdge(T u,T v,bool direction){

        adjList[u].push_back(v);
        if(direction==0){
            adjList[v].push_back(u);
        }
    }

    void printAdjacencyList(){
        for(auto node :adjList){
            cout<<node.first<<"-> ";
            for(auto neighbours: node.second){
                cout<<neighbours<<", ";
            }
            cout<<endl;
        }
    }

    void topologicalSortUsingDFS(int src,unordered_map<int,bool>&
visited,stack<int>& ans){
        visited[src]=true;
```

```
        for(auto neighbour :adjList[src]){
            if(!visited[neighbour]){
                topologicalSortUsingDFS(neighbour,visited,ans);
            }
        }
        ans.push(src);
    }
};

int main()
{
    Graph <int>g;
    int n=8;

    g.addEdge(0,1,1);
    g.addEdge(1,2,1);
    g.addEdge(2,3,1);
    g.addEdge(3,4,1);

    g.addEdge(3,5,1);
    g.addEdge(4,6,1);
    g.addEdge(5,6,1);
    g.addEdge(6,7,1);

    g.printAdjacencyList();
    cout<<endl;

    unordered_map<int,bool>visited;
    stack<int> ans;

    for(int i=0;i<n;i++){
        if(!visited[i]){
            g.topologicalSortUsingDFS(i,visited,ans);
        }
    }

    cout << "topological sort order :"<<endl;
    while(!ans.empty()){
        cout<<ans.top()<<" ,";
        ans.pop();
    }
    return 0;
}
```

```
PS E:\C++Code> g++ .\f77.cpp
PS E:\C++Code> .\a.exe
6-> 7,
5-> 6,
4-> 6,
0-> 1,
1-> 2,
2-> 3,
3-> 4, 5,

topological sort order :
0 ,1 ,2 ,3 ,5 ,4 ,6 ,7 ,
```

Q) write program for Topological Sort Order using BFS

```
#include <iostream>
#include <vector>
#include <unordered_map>
#include <list>
#include <queue>
#include <stack>

using namespace std;

template <typename T>

class Graph{

    public:
    unordered_map<T,list<T> > adjList;

    void addEdge(T u,T v,bool direction){

        adjList[u].push_back(v);
        if(direction==0){
            adjList[v].push_back(u);
        }
    }

    void printAdjacencyList(){
        for(auto node :adjList){
            cout<<node.first<<"-> ";
            for(auto neighbours: node.second){
                cout<<neighbours<<", ";
            }
            cout<<endl;
        }
    }

    void topologicalSortUsingDFS(int src,unordered_map<int,bool>&
visited,stack<int>& ans){
        visited[src]=true;
```

```
        for(auto neighbour :adjList[src]){
            if(!visited[neighbour]){
                topologicalSortUsingDFS(neighbour,visited,ans);
            }
        }
        ans.push(src);
    }

    void topologicalSortUsingBFS(int n,vector<int>& ans){
        queue<int>q;
        unordered_map<int,int> indegree;

        for(auto i: adjList){
            int src=i.first;
            for(auto neighbour: i.second){
                indegree[neighbour]++;
            }
        }

        for(int i=0;i<n;i++){
            if(indegree[i]==0){
                q.push(i);
            }
        }

        while(!q.empty()){
            int fNode=q.front();
            q.pop();

            ans.push_back(fNode);

            for(auto neighbour :adjList[fNode]){
                indegree[neighbour]--;

                if(indegree[neighbour]==0){
                    q.push(neighbour);
                }
            }
        }
    }
};
```

```cpp
int main()
{
    Graph <int>g;
    int n=8;

    g.addEdge(2,4,1);
    g.addEdge(2,5,1);
    g.addEdge(4,6,1);
    g.addEdge(5,3,1);

    g.addEdge(3,7,1);
    g.addEdge(6,7,1);
    g.addEdge(7,0,1);
    g.addEdge(7,1,1);

    g.printAdjacencyList();
    cout<<endl;

    vector<int>ans;
    g.topologicalSortUsingBFS(n,ans);

    cout << "topological sort order using BFS :"<<endl;
    for(int i=0;i<ans.size();i++){
        cout<<ans[i]<<", ";
    }
    return 0;
}
```

```
PS E:\C++Code> g++ .\f77.cpp
PS E:\C++Code> .\a.exe
7-> 0, 1,
6-> 7,
2-> 4, 5,
4-> 6,
5-> 3,
3-> 7,

topological sort order using BFS :
2, 4, 5, 6, 3, 7, 0, 1,
```

Q) cycle detection using topological order graph using BFS

```cpp
#include <iostream>
#include <vector>
#include <unordered_map>
#include <list>
#include <queue>
#include <stack>

using namespace std;

template <typename T>

class Graph{

    public:
    unordered_map<T,list<T> > adjList;

    void addEdge(T u,T v,bool direction){

        adjList[u].push_back(v);
        if(direction==0){
            adjList[v].push_back(u);
        }
    }

    void printAdjacencyList(){
        for(auto node :adjList){
            cout<<node.first<<"-> ";
            for(auto neighbours: node.second){
                cout<<neighbours<<", ";
            }
            cout<<endl;
        }
    }
```

```
    void topologicalSortUsingDFS(int src,unordered_map<int,bool>&
visited,stack<int>& ans){
        visited[src]=true;

        for(auto neighbour :adjList[src]){
            if(!visited[neighbour]){
                topologicalSortUsingDFS(neighbour,visited,ans);
            }
        }
        ans.push(src);
    }
    void topologicalSortUsingBFS(int n,vector<int>& ans){
        queue<int>q;
        unordered_map<int,int> indegree;

        for(auto i: adjList){
            int src=i.first;
            for(auto neighbour: i.second){
                indegree[neighbour]++;
            }
        }

        for(int i=0;i<n;i++){
            if(indegree[i]==0){
                q.push(i);
            }
        }

        while(!q.empty()){
            int fNode=q.front();
            q.pop();

            ans.push_back(fNode);

            for(auto neighbour :adjList[fNode]){
                indegree[neighbour]--;

                if(indegree[neighbour]==0){
                    q.push(neighbour);
                }
            }
        }
    }
};
```

```cpp
int main()
{
    Graph <int>g;
    int n=8;

    g.addEdge(0,1,1);
    g.addEdge(1,2,1);
    g.addEdge(2,3,1);
    g.addEdge(3,1,1);

    g.printAdjacencyList();
    cout<<endl;

    vector<int>ans;
    g.topologicalSortUsingBFS(n,ans);

    if(ans.size()==n){
        cout<<"It is a valid topo sort "<<endl;
    }
    else{
        cout<<"cycle preesnt or invalid topological sort"<<endl;;
    }

    cout << "topological sort order using BFS :"<<endl;
    for(int i=0;i<ans.size();i++){
        cout<<ans[i]<<", ";
    }
    return 0;
}
```

```
PS E:\C++Code> g++ .\f77.cpp
PS E:\C++Code> .\a.exe
3-> 1,
2-> 3,
1-> 2,
0-> 1,

cycle preesnt or invalid topological sort
topological sort order using BFS :
0, 4, 5, 6, 7,
```

Graphs Class – 4

Q) write program for find shortest path using BFS

```
#include<iostream>
#include<unordered_map>
#include<list>
#include<queue>
#include<stack>

using namespace std;

class Graph{
    public :
    unordered_map<int, list<pair<int,int> > >adjList;

    void addEdge(int u,int v,int wt,bool direction){
        //dircttion = 1-> undirected graph
        //dircttion = 0-> directed graph
        adjList[u].push_back({v,wt});
        if(direction==0){
            adjList[v].push_back({u,wt});
        }
    }

    void printAdjList(){
        for(auto i: adjList){
            cout<<i.first<<"->";
            for(auto j : i.second){
                cout<<"("<<j.first<<", "<<j.second<<") ";
            }
            cout<<endl;
        }
    }

    void shortestPathBfs(int src,int dest){
        queue<int>q;
        unordered_map<int,bool>visited;
        unordered_map<int,int>parent;

        q.push(src);
        visited[src]=1;
        parent[src]=-1;


        while(!q.empty()){
```

```
            int fNode=q.front();
            q.pop();

            for(auto neighbour :adjList[fNode]){
                if(!visited[neighbour.first]){
                    q.push(neighbour.first);
                    visited[neighbour.first]=1;
                    parent[neighbour.first]=fNode;
                }
            }
        }
        stack<int>st;
        int node=dest;

        while (node!=-1)
        {
            st.push(node);
            node=parent[node];
        }

        cout<<"printing ans: "<<endl;
        while(!st.empty()){
            cout<<st.top()<<"->";
            st.pop();
        }

    }
};

int main(){

    Graph g;

    g.addEdge(0,1,1,1);
    g.addEdge(1,2,1,1);
    g.addEdge(2,3,1,1);

    g.addEdge(3,4,1,1);
    g.addEdge(0,5,1,1);
    g.addEdge(5,4,1,1);

    g.addEdge(0,6,1,1);
    g.addEdge(6,7,1,1);
    g.addEdge(7,8,1,1);
    g.addEdge(8,4,1,1);
```

```
    g.printAdjList();

    int src=0;
    int dest=4;

    g.shortestPathBfs(src,dest);
    return 0;
}
```

```
PS C:\Users\home\Desktop\DSA Code> g++ .\f77.cpp
PS C:\Users\home\Desktop\DSA Code> .\a.exe
8->(4, 1)
7->(8, 1)
6->(7, 1)
5->(4, 1)
0->(1, 1) (5, 1) (6, 1)
1->(2, 1)
2->(3, 1)
3->(4, 1)
printing ans:
0->5->4->
```

Time complexity : O(n)

Q) write program for find shortest path distance using DFS

```cpp
#include<iostream>
#include<unordered_map>
#include<list>
#include<queue>
#include<stack>
#include<limits.h>

using namespace std;

class Graph{
    public :
    unordered_map<int, list<pair<int,int> > >adjList;

    void addEdge(int u,int v,int wt,bool direction){
        //dircttion = 1-> undirected graph
        //dircttion = 0-> directed graph
        adjList[u].push_back({v,wt});
        if(direction==1){
            adjList[v].push_back({u,wt});
        }
    }

    void printAdjList(){
        for(auto i: adjList){
            cout<<i.first<<"->";
            for(auto j : i.second){
                cout<<"("<<j.first<<", "<<j.second<<") ";
            }
            cout<<endl;
        }
    }

    void topSortDfs(int src ,unordered_map<int,bool>& visited,stack<int>& ans){
        visited[src]=true;

        for(auto neighbour : adjList[src]){
            if(!visited[neighbour.first] ){
                topSortDfs(neighbour.first,visited,ans);
            }
        }
        ans.push(src);
    }
```

```
    void shortestPathDfs(int dest,stack<int>topOrder,int n){
        vector<int>dist(n,INT_MAX);

        int src=topOrder.top();
        topOrder.pop();
        dist[src]=0;

        for(auto neighbour:adjList[0]){
            if(dist[0]+neighbour.second < dist [neighbour.first] ){
                dist [neighbour.first] = dist[0]+neighbour.second;
            }
        }

        while(!topOrder.empty()){
            int topElment=topOrder.top();
            topOrder.pop();

            if(dist[topElment]!=INT_MAX){
                for(auto neighbour:adjList[topElment]){
                    if(dist[topElment]+neighbour.second < dist [neighbour.first]
){
                        dist [neighbour.first] =
dist[topElment]+neighbour.second;
                    }
                }
            }
        }
        cout<<"printing ans :"<<endl;
        for(int i=0;i<n;i++){
            cout<<i<<"->"<<dist[i]<<endl;
        }
    }
};
int main(){

    Graph g;

    g.addEdge(0,1,5,0);
    g.addEdge(0,2,3,0);
    g.addEdge(2,1,2,0);

    g.addEdge(1,3,3,0);
    g.addEdge(2,3,5,0);
    g.addEdge(2,4,6,0);
```

```
    g.addEdge(4,3,1,0);

    g.printAdjList();

    stack<int>topOrder;
    unordered_map<int,bool>visited;
    g.topSortDfs(0,visited,topOrder);

    g.shortestPathDfs(3,topOrder,5);
    return 0;
}
```

```
PS C:\Users\home\Desktop\DSA Code> g++ .\f77.cpp
PS C:\Users\home\Desktop\DSA Code> .\a.exe
4->(3, 1)
1->(3, 3)
2->(1, 2) (3, 5) (4, 6)
0->(1, 5) (2, 3)
printing ans :
0->0
1->5
2->3
3->8
4->9
```

Q) write program for find shortest path distance using dijkstra's algorithm (Imp)

```cpp
#include<iostream>
#include<unordered_map>
#include<list>
#include<queue>
#include<stack>
#include<set>
#include<limits.h>

using namespace std;

class Graph{
    public :
    unordered_map<int, list<pair<int,int> > >adjList;

    void addEdge(int u,int v,int wt,bool direction){
        //dircttion = 1-> undirected graph
        //dircttion = 0-> directed graph
        adjList[u].push_back({v,wt});
        if(direction==1){
            adjList[v].push_back({u,wt});
        }
    }

    void printAdjList(){
        for(auto i: adjList){
            cout<<i.first<<"->";
            for(auto j : i.second){
                cout<<"("<<j.first<<", "<<j.second<<") ";
            }
            cout<<endl;
        }
    }
```

```
    void shortestDistDijkstra(int src,int n){
        vector<int>dist(n,INT_MAX);

        set<pair<int,int> >st;
        dist[src]=0;
        st.insert(make_pair(0,src));

        while(!st.empty()){
            auto topElement=*(st.begin());
            int nodeDistance=topElement.first;
            int node=topElement.second;

            st.erase(st.begin());

            for(auto neighbour:adjList[node]){
                if(nodeDistance+neighbour.second < dist[neighbour.first]){
                    auto
result=st.find(make_pair(dist[neighbour.first],neighbour.first));

                    if(result!=st.end()){
                        st.erase(result);
                    }

                    dist[neighbour.first]=nodeDistance+neighbour.second;

                    st.insert(make_pair(dist[neighbour.first],neighbour.first));
                }
            }
        }

        cout<<"printing ans :"<<endl;
        for(int i=0;i<n;i++){
            cout<<dist[i]<<" ";
        }
        cout<<endl;
    }
};
int main(){

    Graph g;
```

```
    g.addEdge(6,3,2,1);
    g.addEdge(6,1,14,1);
    g.addEdge(3,1,9,1);
    g.addEdge(3,2,10,1);
    g.addEdge(1,2,7,1);
    g.addEdge(2,4,15,1);
    g.addEdge(4,3,11,1);
    g.addEdge(6,5,9,1);
    g.addEdge(4,5,6,1);

    g.printAdjList();

    g.shortestDistDijkstra(6,7);
    return 0;
}
```

```
PS C:\Users\home\Desktop\DSA Code_Next_Part> g++ .\f77.cpp
PS C:\Users\home\Desktop\DSA Code_Next_Part> .\a.exe
5->(6, 9) (4, 6)
4->(2, 15) (3, 11) (5, 6)
6->(3, 2) (1, 14) (5, 9)
1->(6, 14) (3, 9) (2, 7)
3->(6, 2) (1, 9) (2, 10) (4, 11)
2->(3, 10) (1, 7) (4, 15)
printing ans :
2147483647 11 12 2 13 9 0
```

Time complexity : O(e log v)

Graphs – II

Graphs Class – 5

547. Number of Provinces

Medium 8.6K 318

Companies

There are `n` cities. Some of them are connected, while some are not. If city `a` is connected directly with city `b`, and city `b` is connected directly with city `c`, then city `a` is connected indirectly with city `c`.

A **province** is a group of directly or indirectly connected cities and no other cities outside of the group.

You are given an `n x n` matrix `isConnected` where `isConnected[i][j] = 1` if the i^{th} city and the j^{th} city are directly connected, and `isConnected[i][j] = 0` otherwise.

Return *the total number of* ***provinces***.

Example 1:

```cpp
class Solution {
public:
    void dfs(unordered_map<int,bool>& visited,int src,vector<vector<int>>& isConnected){
        visited[src]=true;

        int size=isConnected[src].size();

        for(int col=0;col<size;col++){
            if( isConnected[src][col]==1){  // (src!=col && isConnected[src][col]==1)
                if(!visited[col]){
                    dfs(visited,col,isConnected);
                }
            }
        }
    }
    int findCircleNum(vector<vector<int>>& isConnected) {
        unordered_map<int,bool>visited;

        int count=0;
        int n=isConnected.size();

        for(int i=0;i<n;i++){
            if(!visited[i]){
                dfs(visited,i,isConnected);
                count++;
```

Description Editorial Solutions Submissions

200. Number of Islands

Medium 21.5K 468

Companies

Given an `m x n` 2D binary grid `grid` which represents a map of `'1'`s (land) and `'0'`s (water), return *the number of islands*.

An **island** is surrounded by water and is formed by connecting adjacent lands horizontally or vertically. You may assume all four edges of the grid are all surrounded by water.

Example 1:

```
Input: grid = [
  ["1","1","1","1","0"],
  ["1","1","0","1","0"],
  ["1","1","0","0","0"],
  ["0","0","0","0","0"]
]
Output: 1
```

C++ Auto

```cpp
class Solution {
    private:
    void bfs(map <pair<int,int>, bool >& visited,int row,int col,vector<vector<char>>& grid){
        queue<pair<int,int> >q;
        q.push({row,col});
        visited[{row,col}]=true;

        while(!q.empty()){
            pair<int,int>fNode=q.front();
            q.pop();

            int x=fNode.first;
            int y=fNode.second;

            int dx[]={-1,0,1,0};
            int dy[]={0,1,0,-1};

            for(int i=0;i<4;i++){
                int newX = x+dx[i];
                int newY = y+dy[i];

                if( newX >=0 && newX < grid.size() && newY >=0 && newY < grid[0].size() &&
                !visited[{newX,newY}] && grid[newX][newY]=='1'){
                    q.push({newX,newY});
                    visited[{newX,newY}]=true;
```

Saved to local

Console Run

Description Editorial Solutions Submissions

733. Flood Fill

Hint

Easy 8K 791

Companies

An image is represented by an `m x n` integer grid `image` where `image[i][j]` represents the pixel value of the image.

You are also given three integers `sr`, `sc`, and `color`. You should perform a **flood fill** on the image starting from the pixel `image[sr][sc]`.

To perform a **flood fill**, consider the starting pixel, plus any pixels connected **4-directionally** to the starting pixel of the same color as the starting pixel, plus any pixels connected **4-directionally** to those pixels (also with the same color), and so on. Replace the color of all of the aforementioned pixels with `color`.

Return *the modified image after performing the flood fill.*

C++ Auto

```cpp
class Solution {
    private:
    void dfs(int x,int y,int oldColor,int newColor,map< pair<int,int>, bool>& vis,vector<vector<int>>& arr){
        vis[{x,y}]=true;
        arr[x][y]=newColor;

        int dx[]={-1,0,1,0};
        int dy[]={0,1,0,-1};

        for(int i=0;i<4;i++){
            int newX= x+dx[i];
            int newY=y+dy[i];

            if(newX>=0 && newX<arr.size() && newY>=0 && newY<arr[0].size()
            && !vis[{newX,newY}] && arr[newX][newY]==oldColor){
                dfs(newX,newY,oldColor,newColor,vis,arr);
            }
        }
    }
public:
    vector<vector<int>> floodFill(vector<vector<int>>& image, int sr, int sc, int color) {

        int oldColor=image[sr][sc];

        map< pair<int,int>, bool> vis;
```

Saved to local

Console Run

Description Editorial Solutions Submissions

994. Rotting Oranges

Medium 11.9K 370

Companies

You are given an `m x n` `grid` where each cell can have one of three values:

- `0` representing an empty cell,
- `1` representing a fresh orange, or
- `2` representing a rotten orange.

Every minute, any fresh orange that is **4-directionally adjacent** to a rotten orange becomes rotten.

Return *the minimum number of minutes that must elapse until no cell has a fresh orange.* If *this is impossible, return* `-1`.

Example 1:

C++ Auto

```cpp
class Solution {
public:
    int orangesRotting(vector<vector<int>>& grid) {
        queue<pair<pair<int,int>,int> >q;
        vector<vector<int>> arr=grid;

        int ansTime=0;
        for(int row=0;row<grid.size();row++){
            for(int col=0;col<grid[row].size();col++){
                if(grid[row][col]==2){
                    pair<int,int>coordinate=make_pair(row,col);
                    q.push({coordinate,0});
                }
            }
        }

        while(!q.empty()){
            auto fnode=q.front();
            q.pop();
            int x=fnode.first.first;
            int y=fnode.first.second;
            int time=fnode.second;

            int dx[]={-1,0,1,0};
            int dy[]={0,1,0,-1};
```

Saved to local

Console

Graphs Class – 6

Description Editorial Solutions Submissions

207. Course Schedule

Hint

Medium 14.6K 580

Companies

There are a total of `numCourses` courses you have to take, labeled from `0` to `numCourses - 1`. You are given an array `prerequisites` where `prerequisites[i] = [a_i, b_i]` indicates that you **must** take course `b_i` first if you want to take course `a_i`.

- For example, the pair `[0, 1]`, indicates that to take course `0` you have to first take course `1`.

Return `true` if you can finish all courses. Otherwise, return `false`.

Example 1:

```
Input: numCourses = 2, prerequisites = [[1,0]]
Output: true
```

C++ Auto

```cpp
class Solution {
    public :
    bool topologicalSortUsingBFS(int n,unordered_map<int,list<int> >& adjList){
        vector<int>ans;
        queue<int>q;
        unordered_map<int,int> indegree;

        for(auto i: adjList){
            int src=i.first;
            for(auto neighbour: i.second){
                indegree[neighbour]++;
            }
        }

        for(int i=0;i<n;i++){
            if(indegree[i]==0){
                q.push(i);
            }
        }

        while(!q.empty()){
            int fNode=q.front();
            q.pop();

            ans.push_back(fNode);
```

Console

Description Editorial Solutions Submissions

210. Course Schedule II

Hint

Medium 9.6K 304

Companies

There are a total of `numCourses` courses you have to take, labeled from `0` to `numCourses - 1`. You are given an array `prerequisites` where `prerequisites[i] = [a_i, b_i]` indicates that you **must** take course `b_i` first if you want to take course `a_i`.

- For example, the pair `[0, 1]`, indicates that to take course `0` you have to first take course `1`.

Return *the ordering of courses you should take to finish all courses*. If there are many valid answers, return **any** of them. If it is impossible to finish all courses, return **an empty array**.

Example 1:

C++ Auto

```cpp
class Solution {
    private :
    vector<int> topologicalSortUsingBFS(int n,unordered_map<int,list<int> >& adjList){
        vector<int>ans;
        queue<int>q;
        unordered_map<int,int> indegree;

        for(auto i: adjList){
            int src=i.first;
            for(auto neighbour: i.second){
                indegree[neighbour]++;
            }
        }

        for(int i=0;i<n;i++){
            if(indegree[i]==0){
                q.push(i);
            }
        }

        while(!q.empty()){
            int fNode=q.front();
            q.pop();

            ans.push_back(fNode);
```

Console

Description | Editorial | Solutions (2.9K) | Submissions

127. Word Ladder

Hard | 10.8K | 1.8K

Companies

A **transformation sequence** from word `beginWord` to word `endWord` using a dictionary `wordList` is a sequence of words `beginWord -> s1 -> s2 -> ... -> sk` such that:

- Every adjacent pair of words differs by a single letter.
- Every `si` for `1 <= i <= k` is in `wordList`. Note that `beginWord` does not need to be in `wordList`.
- `sk == endWord`

Given two words, `beginWord` and `endWord`, and a dictionary `wordList`, return *the **number of words** in the **shortest transformation sequence** from `beginWord` to `endWord`, or `0` if no such sequence exists.*

Example 1:

C++ | Auto

```cpp
class Solution {
public:
    int ladderLength(string beginWord, string endWord, vector<string>& wordList) {
        queue<pair<string,int> >q;

        q.push({beginWord,1});

        unordered_set<string>st(wordList.begin(),wordList.end());

        st.erase(beginWord);

        while(!q.empty()){
            pair<string,int> fNode=q.front();
            q.pop();

            string currString=fNode.first;
            int currCount=fNode.second;

            if(currString==endWord){
                return currCount;
            }

            for(int index=0;index<currString.length();index++){
                char originalCharacter=currString[index];
```

Description | Editorial | Solutions | Submissions

1631. Path With Minimum Effort

Hint

Medium | 4.6K | 159

Companies

You are a hiker preparing for an upcoming hike. You are given `heights`, a 2D array of size `rows x columns`, where `heights[row][col]` represents the height of cell `(row, col)`. You are situated in the top-left cell, `(0, 0)`, and you hope to travel to the bottom-right cell, `(rows-1, columns-1)` (i.e., **0-indexed**). You can move **up**, **down**, **left**, or **right**, and you wish to find a route that requires the minimum **effort**.

A route's **effort** is the **maximum absolute difference** in heights between two consecutive cells of the route.

Return *the minimum **effort** required to travel from the top-left cell to the bottom-right cell.*

C++ | Auto

```cpp
class Solution {
public:
    int minimumEffortPath(vector<vector<int>>& heights) {
        priority_queue< pair<int ,pair<int,int> >,vector<pair<int,pair<int,int> > > ,greater<pair<
pq;
        vector<vector<int>> dist (heights.size(),vector<int>(heights[0].size(),INT_MAX));

        pq.push({0,{0,0}});
        dist[0][0]=0;

        while(!pq.empty()){
            auto fNode=pq.top();
            pq.pop();
            int frontNodeDifference=fNode.first;
            int x=fNode.second.first;
            int y=fNode.second.second;

            if(x==heights.size() -1 && y==heights[0].size() -1){
                return dist[x][y];
            }

            int dx[]={-1,0,1,0};
            int dy[]={0,1,0,-1};

            for(int i=0;i<4;i++){
```

Graphs Class – 7

Q) write bellman ford algorithms

```
#include<iostream>
#include<unordered_map>
#include<list>
#include<queue>
#include<stack>
#include<set>
#include<limits.h>

using namespace std;

class Graph{
    public :
    unordered_map<int, list<pair<int,int> > >adjList;

    void addEdge(int u,int v,int wt,bool direction){
        //dircttion = 1-> undirected graph
        //dircttion = 0-> directed graph
        adjList[u].push_back({v,wt});
        if(direction==1){
            adjList[v].push_back({u,wt});
        }
    }

    void printAdjList(){
        for(auto i: adjList){
            cout<<i.first<<"->";
            for(auto j : i.second){
                cout<<"("<<j.first<<", "<<j.second<<") ";
            }
            cout<<endl;
        }
    }
```

```
    void bellmanForAlgo(int n,int src){
        vector<int>dist(n,INT_MAX);
        dist[src]=0;

        for(int i=0;i<n-1;i++){
            for(auto t:adjList){
                for(auto neighbour:t.second){
                    int u=t.first;
                    int v=neighbour.first;
                    int wt=neighbour.second;

                    if(dist[u]!=INT_MAX && dist[u]+wt < dist[v]){
                        dist[v]=dist[u]+wt;
                    }
                }
            }
        }
        cout<<"printing Bellman Ford Algorithms ans :"<<endl;
        for(auto i: dist){
            cout<<i<<" ";
        }
    }
};
int main(){

    Graph g;

    g.addEdge(0,1,-1,0);
    g.addEdge(0,2,4,0);
    g.addEdge(1,2,3,0);

    g.addEdge(1,3,2,0);
    g.addEdge(1,4,2,0);
    g.addEdge(3,1,2,0);

    g.addEdge(3,2,5,0);
    g.addEdge(4,3,-3,0);

    g.printAdjList();

    g.bellmanForAlgo(5,0);
    //g.shortestDistDijkstra(6,7);
    return 0;
}
```

```
PS C:\Users\home\Desktop\DsaCodeNewNext> g++ .\f77.cpp
PS C:\Users\home\Desktop\DsaCodeNewNext> .\a.exe
4->(3, -3)
3->(1, 2) (2, 5)
1->(2, 3) (3, 2) (4, 2)
0->(1, -1) (2, 4)
printing Bellman Ford Algorithms ans :
0 -1 2 -2 1
```

Q) write program to check –ve cycle is present or absen using bellman ford algorithms

```
#include<iostream>
#include<unordered_map>
#include<list>
#include<queue>
#include<stack>
#include<set>
#include<limits.h>

using namespace std;

class Graph{
    public :
    unordered_map<int, list<pair<int,int> > >adjList;

    void addEdge(int u,int v,int wt,bool direction){
        //dircttion = 1-> undirected graph
        //dircttion = 0-> directed graph
        adjList[u].push_back({v,wt});
        if(direction==1){
            adjList[v].push_back({u,wt});
        }
    }

    void printAdjList(){
        for(auto i: adjList){
            cout<<i.first<<"->";
            for(auto j : i.second){
                cout<<"("<<j.first<<", "<<j.second<<") ";
            }
            cout<<endl;
        }
    }
```

```
    void bellmanFordAlgo(int n,int src){
        vector<int>dist(n,INT_MAX);
        dist[src]=0;

        for(int i=0;i<n-1;i++){
            for(auto t:adjList){
                for(auto neighbour:t.second){
                    int u=t.first;
                    int v=neighbour.first;
                    int wt=neighbour.second;

                    if(dist[u]!=INT_MAX && dist[u]+wt < dist[v]){
                        dist[v]=dist[u]+wt;
                    }
                }
            }
        }
        //check -ve cycle
        bool negativeCycle=false;
        for(auto t:adjList){
                for(auto neighbour:t.second){
                    int u=t.first;
                    int v=neighbour.first;
                    int wt=neighbour.second;

                    if(dist[u]!=INT_MAX && dist[u]+wt < dist[v]){
                        negativeCycle=true;
                        break;
                    }
                }
            }

        if(negativeCycle==true){
            cout<<"-ve cycle is present";
        }
        else{
            cout<<"-ve cycle is absent";
        }
        cout<<endl;
        cout<<"printing Bellman Ford Algorithms ans :"<<endl;
        for(auto i: dist){
            cout<<i<<" ";
        }
    }
};
```

```cpp
int main(){

    Graph g;

    g.addEdge(0,1,-1,0);
    g.addEdge(0,2,4,0);
    g.addEdge(1,2,3,0);

    g.addEdge(1,3,2,0);
    g.addEdge(1,4,2,0);
    g.addEdge(3,1,2,0);

    g.addEdge(3,2,5,0);
    g.addEdge(4,3,-3,0);

    g.printAdjList();

    g.bellmanFordAlgo(5,0);
    //g.shortestDistDijkstra(6,7);
    return 0;
}
```

```
PS C:\Users\home\Desktop\DsaCodeNewNext> g++ .\f77.cpp
PS C:\Users\home\Desktop\DsaCodeNewNext> .\a.exe
4->(3, -3)
3->(1, 2) (2, 5)
1->(2, 3) (3, 2) (4, 2)
0->(1, -1) (2, 4)
-ve cycle is absent
printing Bellman Ford Algorithms ans :
0 -1 2 -2 1
```

Q) write Floyd Warshall algorithms

```cpp
#include<iostream>
#include<unordered_map>
#include<list>
#include<queue>
#include<stack>
#include<set>
#include<limits.h>
#include<algorithm>

using namespace std;

class Graph{
    public :
    unordered_map<int, list<pair<int,int> > >adjList;

    void addEdge(int u,int v,int wt,bool direction){
        //dircttion = 1-> undirected graph
        //dircttion = 0-> directed graph
        adjList[u].push_back({v,wt});
        if(direction==1){
            adjList[v].push_back({u,wt});
        }
    }

    void printAdjList(){
        for(auto i: adjList){
            cout<<i.first<<"->";
            for(auto j : i.second){
                cout<<"("<<j.first<<", "<<j.second<<") ";
            }
            cout<<endl;
        }
    }
```

```
    void floydWarshall(int n){
        vector<vector<int> >dist(n,vector<int>(n,1e9));

        for(int i=0;i<n;i++){
            dist[i][i]=0;
        }

        for(auto t:adjList){
                for(auto neighbour:t.second){
                    int u=t.first;
                    int v=neighbour.first;
                    int wt=neighbour.second;

                    dist[u][v]=wt;
                }
            }

        for(int helper=0;helper<n;helper++){
            for(int src=0;src<n;src++){
                for(int dest=0;dest<n;dest++){
                    dist[src][dest]=min(dist[src][dest],
dist[src][helper]+dist[helper][dest]);
                }
            }
        }
        cout<<endl;
        cout<<"printing floyd warshall Algorithms ans :"<<endl;
        for(int i=0;i<n;i++){
            for(int j=0;j<n;j++){
                cout<<dist[i][j]<<" ";
            }
            cout<<endl;
        }
    }
};
int main(){
    Graph g;

    g.addEdge(0,1,3,0);
    g.addEdge(0,3,5,0);
    g.addEdge(1,0,2,0);

    g.addEdge(1,3,4,0);
    g.addEdge(2,1,1,0);
    g.addEdge(3,2,2,0);
```

```
    g.printAdjList();
    g.floydWarshall(4);

    return 0;
}
```

```
PS C:\Users\home\Desktop\DsaCodeNewNext> g++ .\f77.cpp
PS C:\Users\home\Desktop\DsaCodeNewNext> .\a.exe
3->(2, 2)
2->(1, 1)
1->(0, 2) (3, 4)
0->(1, 3) (3, 5)

printing floyd warshall Algorithms ans :
0 3 7 5
2 0 6 4
3 1 0 5
5 3 2 0
```

Graphs Class – 8

Q) Strongly Connected Components (Kosaraju's Algo)

```cpp
#include <iostream>

#include <vector>
#include <unordered_map>
#include <list>
#include <queue>
#include <stack>

using namespace std;


class Graph{

    public:
    unordered_map<int,list<int> > adjList;

    void addEdge(int u,int v,bool direction){

        adjList[u].push_back(v);
        if(direction==1){
            adjList[v].push_back(u);
        }
    }

    void dfs1(int src,stack<int>& s,unordered_map<int,bool>& vis){
        vis[src]=true;

        for(auto neighbour :adjList[src]){
            if(!vis[neighbour]){
                dfs1(neighbour,s,vis);
            }
        }
        s.push(src);
    }
```

```
    void dfs2(int src,unordered_map<int,bool>&
visited,unordered_map<int,list<int> >& adjNew){
        visited[src]=true;

        cout<<src<<", ";
        for(auto neighbour :adjNew[src]){
            if(!visited[neighbour]){
                dfs2(neighbour,visited,adjNew);
            }
        }

    }
```

```
    int countSSC(int n){
        stack<int>s;
        unordered_map<int,bool>visited;

        for(int i=0;i<n;i++){
            if(!visited[i]){
                dfs1(i,s,visited);
            }
        }

        unordered_map<int,list<int> > adjNew;

        for(auto t: adjList){
            for(auto neighbour:t.second){
                int u=t.first;
                int v=neighbour;

                adjNew[v].push_back(u);
            }
        }

        int count=0;
        unordered_map<int,bool>visited2;

        while(!s.empty()){
            int node=s.top();
            s.pop();
            if(!visited2[node]){
                cout<<"printing "<<count+1<<" th SSC:";
                dfs2(node,visited2,adjNew);
                cout<<endl;
                count++;
            }
        }
        return count;
    }

};
```

```cpp
int main()
{
    Graph g;
    g.addEdge(0,1,0);
    g.addEdge(1,2,0);
    g.addEdge(2,3,0);
    g.addEdge(3,0,0);
    g.addEdge(2,4,0);
    g.addEdge(4,5,0);
    g.addEdge(5,6,0);
    g.addEdge(6,4,0);
    g.addEdge(6,7,0);

    int ans =g.countSSC(8);
    cout<<"number of SSC: "<<ans;
    return 0;
}
```

```
PS E:\C++Code> g++ .\f77.cpp
PS E:\C++Code> g++ .\f77.cpp
PS E:\C++Code> .\a.exe
printing 1 th SSC:0, 3, 2, 1,
printing 2 th SSC:4, 6, 5,
printing 3 th SSC:7,
number of SSC: 3
```

Q) Bridges in Graph using Tarjan's Algorithm

```cpp
#include <iostream>
#include <vector>
#include <unordered_map>
#include <list>
#include <queue>
#include <stack>
#include <algorithm>

using namespace std;


class Graph{

    public:
    unordered_map<int,list<int> > adjList;

    void addEdge(int u,int v,bool direction){

        adjList[u].push_back(v);
        if(direction==1){
            adjList[v].push_back(u);
        }
    }
```

```
    void findBridges(int src,int parent,int & timer,vector<int>& tin,vector<int>&
low,unordered_map<int,bool>vis){
        vis[src]=true;
        tin[src]=timer;
        low[src]=timer;
        timer++;

        for(auto neighbour: adjList[src]){
            if(neighbour==parent){
                continue;
            }

            if(!vis[neighbour]){
                findBridges(neighbour,src,timer,tin,low,vis);

                low[src]=min(low[src],low[neighbour]);

                if(low[neighbour] > low[src]){
                    cout<<neighbour<<"--"<<src<<" is a bridge"<<endl;
                }
            }
            else{
                low[src]=min(low[src],low[neighbour]);
            }
        }
    }
};

int main()
{
    Graph g;
    g.addEdge(0,1,1);
    g.addEdge(0,2,1);
    g.addEdge(2,1,1);
    g.addEdge(0,3,1);
    g.addEdge(3,4,1);

    int n=5;
    int timer=0;
    vector<int> tin(n);
    vector<int> low(n);
    unordered_map<int,bool>vis;
    g.findBridges(0,-1,timer,tin,low,vis);
    return 0;
}
```

```
PS E:\C++Code> .\a.exe
4--3 is a bridge
3--0 is a bridge
```

Description | Editorial | Solutions (1K) | Submissions

1192. Critical Connections in a Network

Hint

Hard | 5.7K | 173

Companies

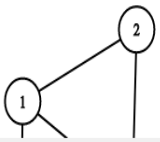
Google

There are `n` servers numbered from `0` to `n - 1` connected by undirected server-to-server `connections` forming a network where `connections[i] = [a_i, b_i]` represents a connection between servers `a_i` and `b_i`. Any server can reach other servers directly or indirectly through the network.

A *critical connection* is a connection that, if removed, will make some servers unable to reach some other server.

Return all critical connections in the network in any order.

Example 1:



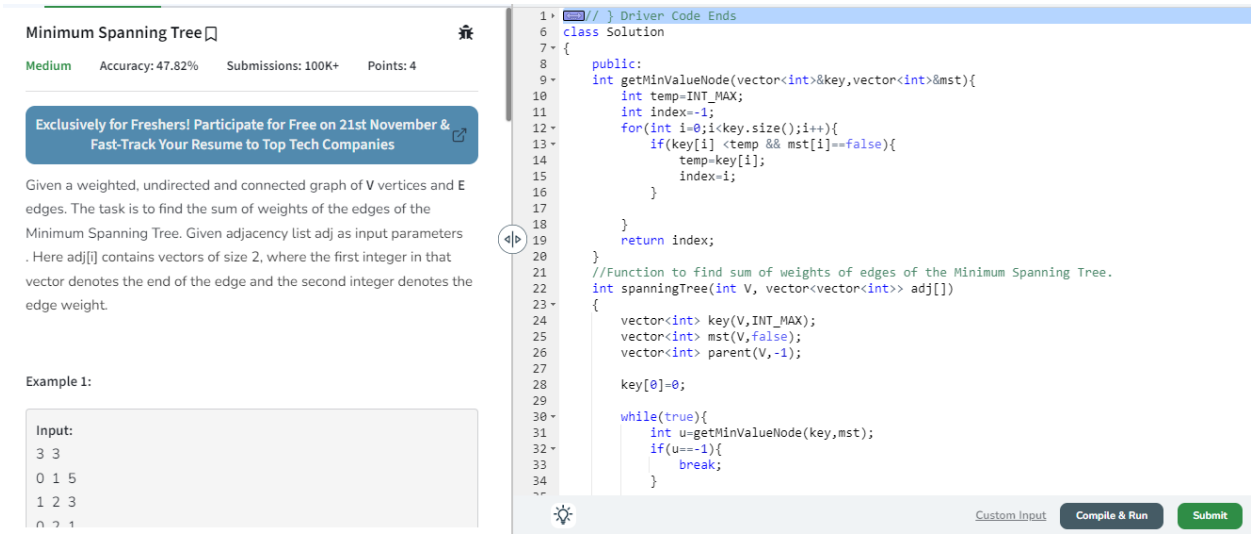
C++ | Auto

```cpp
class Solution {
public:
    void findBridges(int src,int parent,int & timer,vector<int>& tin,vector<int>& low,unordered_map<int,bool>& vis,
    vector<vector<int>>& ans,unordered_map<int,list<int> >& adjList){
        vis[src]=true;
        tin[src]=timer;
        low[src]=timer;
        timer++;

        for(auto neighbour: adjList[src]){
            if(neighbour==parent){
                continue;
            }

            if(!vis[neighbour]){
                findBridges(neighbour,src,timer,tin,low,vis,ans,adjList);

                low[src]=min(low[src],low[neighbour]);

                if(low[neighbour] > tin[src]){
                    vector<int>temp;
                    temp.push_back(neighbour);
                    temp.push_back(src);
                    ans.push_back(temp);
```

Console | Run | Submit

Graphs Assignment

Prim's Algorithms for MST



Minimum Spanning Tree

Medium Accuracy: 47.82% Submissions: 100K+ Points: 4

Exclusively for Freshers! Participate for Free on 21st November & Fast-Track Your Resume to Top Tech Companies

Given a weighted, undirected and connected graph of V vertices and E edges. The task is to find the sum of weights of the edges of the Minimum Spanning Tree. Given adjacency list adj as input parameters . Here adj[i] contains vectors of size 2, where the first integer in that vector denotes the end of the edge and the second integer denotes the edge weight.

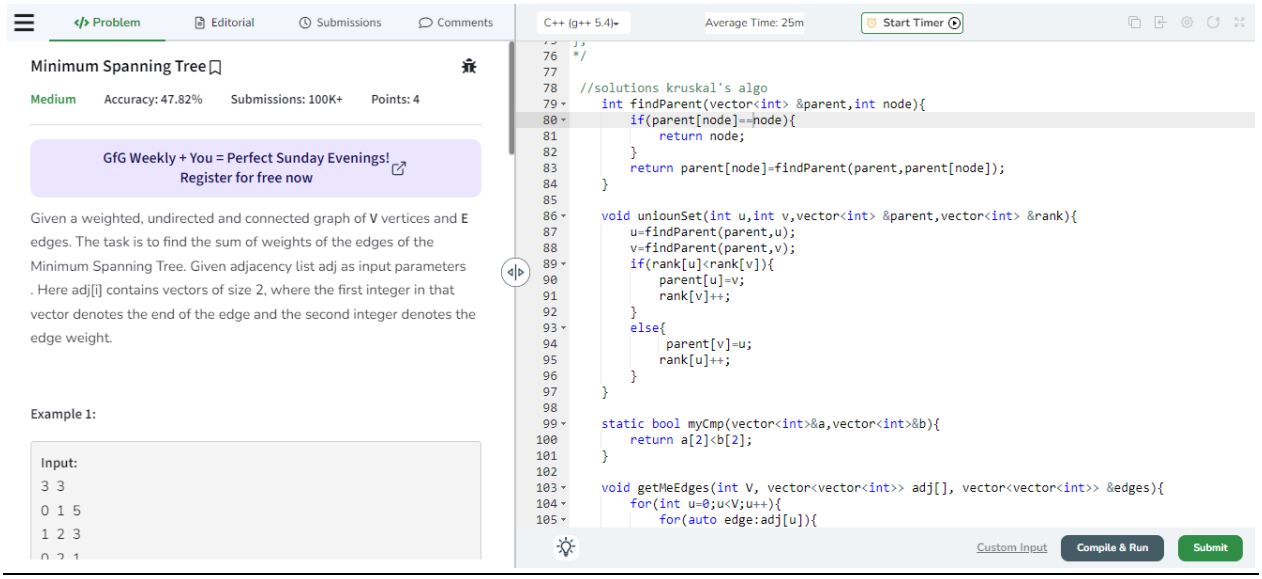
Example 1:

Input:
3 3
0 1 5
1 2 3

```
// } Driver Code Ends
class Solution
{
    public:
    int getMinValueNode(vector<int>&key,vector<int>&mst){
        int temp=INT_MAX;
        int index=-1;
        for(int i=0;i<key.size();i++){
            if(key[i] <temp && mst[i]==false){
                temp=key[i];
                index=i;
            }

        }
        return index;
    }
    //Function to find sum of weights of edges of the Minimum Spanning Tree.
    int spanningTree(int V, vector<vector<int>> adj[])
    {
        vector<int> key(V,INT_MAX);
        vector<int> mst(V,false);
        vector<int> parent(V,-1);

        key[0]=0;

        while(true){
            int u=getMinValueNode(key,mst);
            if(u==-1){
                break;
            }
```

Custom Input | Compile & Run | Submit

Kruskal's MST Algorithm



Problem | Editorial | Submissions | Comments

C++ (g++ 5.4) | Average Time: 25m | Start Timer

Minimum Spanning Tree

Medium Accuracy: 47.82% Submissions: 100K+ Points: 4

GfG Weekly + You = Perfect Sunday Evenings! Register for free now

Given a weighted, undirected and connected graph of V vertices and E edges. The task is to find the sum of weights of the edges of the Minimum Spanning Tree. Given adjacency list adj as input parameters . Here adj[i] contains vectors of size 2, where the first integer in that vector denotes the end of the edge and the second integer denotes the edge weight.

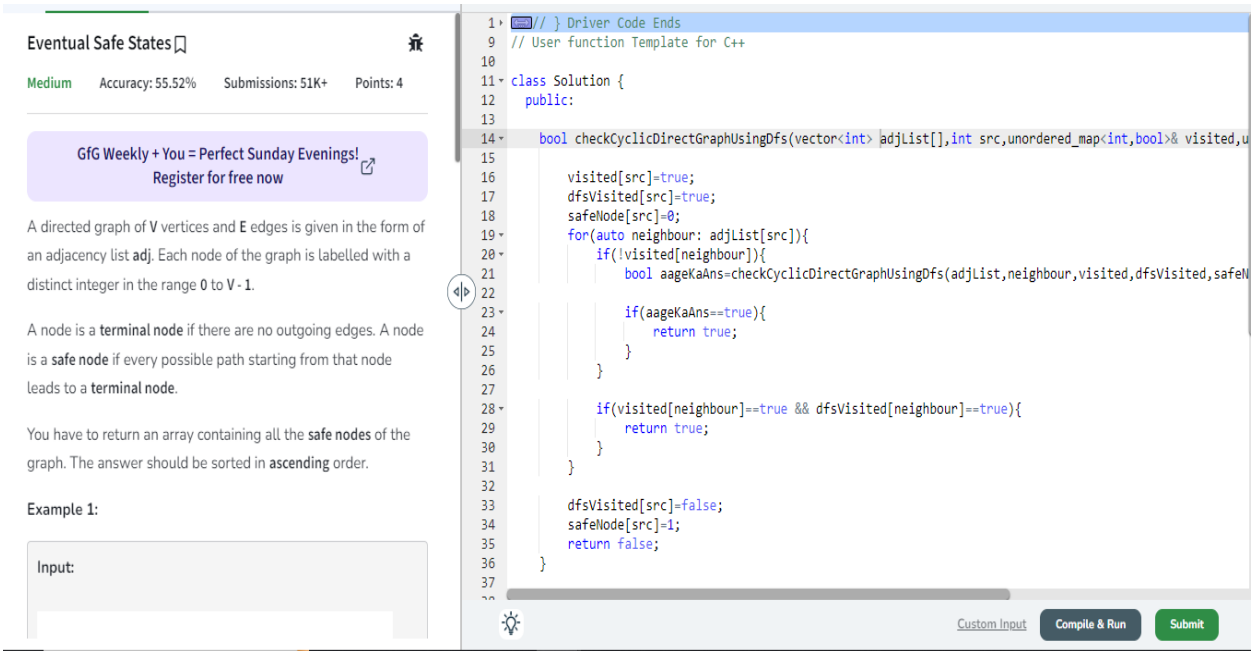
Example 1:

Input:
3 3
0 1 5
1 2 3

```
*/

//solutions kruskal's algo
    int findParent(vector<int> &parent,int node){
        if(parent[node]==node){
            return node;
        }
        return parent[node]=findParent(parent,parent[node]);
    }

    void uniounSet(int u,int v,vector<int> &parent,vector<int> &rank){
        u=findParent(parent,u);
        v=findParent(parent,v);
        if(rank[u]<rank[v]){
            parent[u]=v;
            rank[v]++;
        }
        else{
            parent[v]=u;
            rank[u]++;
        }
    }

    static bool myCmp(vector<int>&a,vector<int>&b){
        return a[2]<b[2];
    }

    void getMeEdges(int V, vector<vector<int>> adj[], vector<vector<int>> &edges){
        for(int u=0;u<V;u++){
            for(auto edge:adj[u]){
```

Custom Input | Compile & Run | Submit

Eventual Safe States



Eventual Safe States

Medium Accuracy: 55.52% Submissions: 51K+ Points: 4

GfG Weekly + You = Perfect Sunday Evenings!
Register for free now

A directed graph of **V** vertices and **E** edges is given in the form of an adjacency list **adj**. Each node of the graph is labelled with a distinct integer in the range **0** to **V - 1**.

A node is a **terminal node** if there are no outgoing edges. A node is a **safe node** if every possible path starting from that node leads to a **terminal node**.

You have to return an array containing all the **safe nodes** of the graph. The answer should be sorted in **ascending** order.

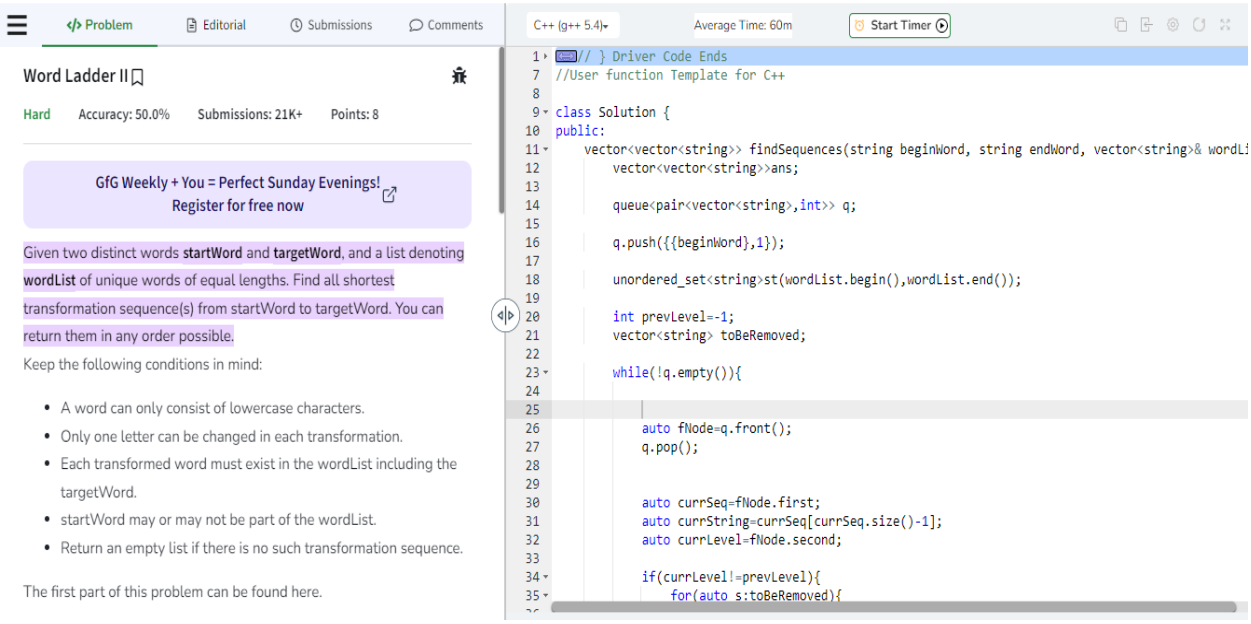
Example 1:

Input:

```
// } Driver Code Ends
// User function Template for C++

class Solution {
  public:

    bool checkCyclicDirectGraphUsingDfs(vector<int> adjList[],int src,unordered_map<int,bool>& visited,u

        visited[src]=true;
        dfsVisited[src]=true;
        safeNode[src]=0;
        for(auto neighbour: adjList[src]){
            if(!visited[neighbour]){
                bool aageKaAns=checkCyclicDirectGraphUsingDfs(adjList,neighbour,visited,dfsVisited,safeN

                if(aageKaAns==true){
                    return true;
                }
            }

            if(visited[neighbour]==true && dfsVisited[neighbour]==true){
                return true;
            }
        }

        dfsVisited[src]=false;
        safeNode[src]=1;
        return false;
    }
```

Custom Input | Compile & Run | Submit

Word Ladder II



Problem | Editorial | Submissions | Comments

C++ (g++ 5.4) | Average Time: 60m | Start Timer

Word Ladder II

Hard Accuracy: 50.0% Submissions: 21K+ Points: 8

GfG Weekly + You = Perfect Sunday Evenings!
Register for free now

Given two distinct words **startWord** and **targetWord**, and a list denoting **wordList** of unique words of equal lengths. Find all shortest transformation sequence(s) from startWord to targetWord. You can return them in any order possible.

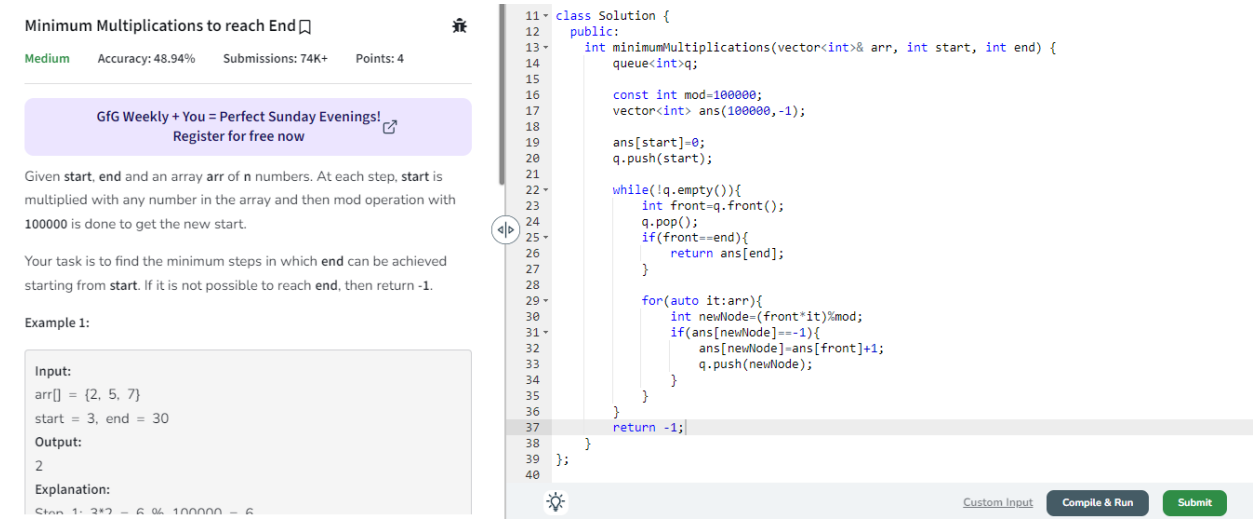
Keep the following conditions in mind:

- A word can only consist of lowercase characters.
- Only one letter can be changed in each transformation.
- Each transformed word must exist in the wordList including the targetWord.
- startWord may or may not be part of the wordList.
- Return an empty list if there is no such transformation sequence.

The first part of this problem can be found here.

```
// } Driver Code Ends
//User function Template for C++

class Solution {
public:
    vector<vector<string>> findSequences(string beginWord, string endWord, vector<string>& wordLi
        vector<vector<string>>ans;

        queue<pair<vector<string>,int>> q;

        q.push({{beginWord},1});

        unordered_set<string>st(wordList.begin(),wordList.end());

        int prevLevel=-1;
        vector<string> toBeRemoved;

        while(!q.empty()){


            auto fNode=q.front();
            q.pop();


            auto currSeq=fNode.first;
            auto currString=currSeq[currSeq.size()-1];
            auto currLevel=fNode.second;

            if(currLevel!=prevLevel){
                for(auto s:toBeRemoved){
```

Minimum Multiplications to reach End



Minimum Multiplications to reach End

Medium Accuracy: 48.94% Submissions: 74K+ Points: 4

GfG Weekly + You = Perfect Sunday Evenings!
Register for free now

Given **start**, **end** and an array **arr** of **n** numbers. At each step, **start** is multiplied with any number in the array and then mod operation with **100000** is done to get the new start.

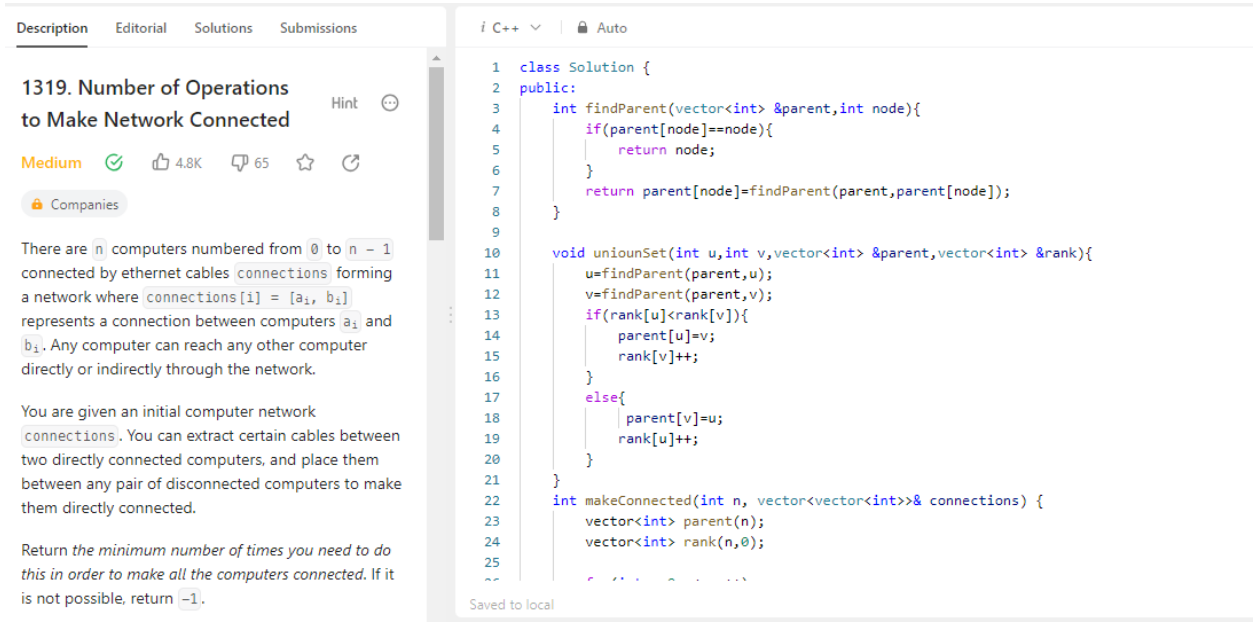
Your task is to find the minimum steps in which **end** can be achieved starting from **start**. If it is not possible to reach **end**, then return **-1**.

Example 1:

Input:
arr[] = {2, 5, 7}
start = 3, end = 30
Output:
2
Explanation:

```cpp
class Solution {
  public:
    int minimumMultiplications(vector<int>& arr, int start, int end) {
        queue<int>q;

        const int mod=100000;
        vector<int> ans(100000,-1);

        ans[start]=0;
        q.push(start);

        while(!q.empty()){
            int front=q.front();
            q.pop();
            if(front==end){
                return ans[end];
            }

            for(auto it:arr){
                int newNode=(front*it)%mod;
                if(ans[newNode]==-1){
                    ans[newNode]=ans[front]+1;
                    q.push(newNode);
                }
            }
        }
        return -1;
    }
};
```

Custom Input | Compile & Run | Submit



Description | Editorial | Solutions | Submissions

1319. Number of Operations to Make Network Connected

Hint

Medium 4.8K 65

Companies

There are `n` computers numbered from `0` to `n - 1` connected by ethernet cables `connections` forming a network where `connections[i] = [a_i, b_i]` represents a connection between computers `a_i` and `b_i`. Any computer can reach any other computer directly or indirectly through the network.

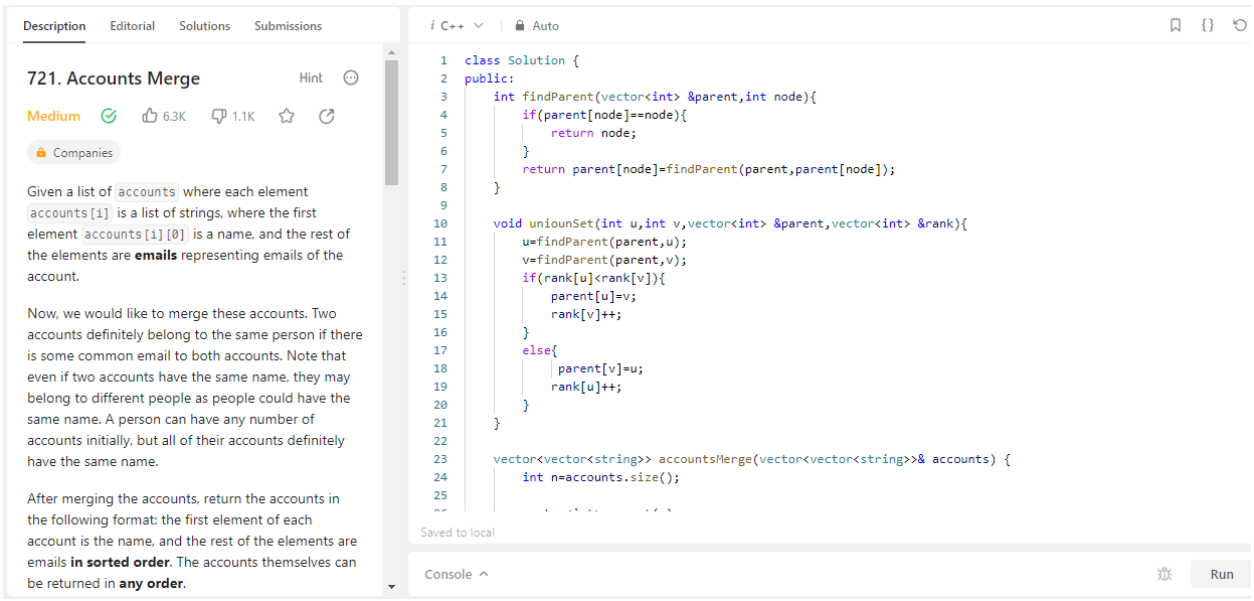
You are given an initial computer network `connections`. You can extract certain cables between two directly connected computers, and place them between any pair of disconnected computers to make them directly connected.

Return *the minimum number of times you need to do this in order to make all the computers connected.* If it is not possible, return `-1`.

C++ | Auto

```cpp
class Solution {
public:
    int findParent(vector<int> &parent,int node){
        if(parent[node]==node){
            return node;
        }
        return parent[node]=findParent(parent,parent[node]);
    }

    void uniounSet(int u,int v,vector<int> &parent,vector<int> &rank){
        u=findParent(parent,u);
        v=findParent(parent,v);
        if(rank[u]<rank[v]){
            parent[u]=v;
            rank[v]++;
        }
        else{
            parent[v]=u;
            rank[u]++;
        }
    }
    int makeConnected(int n, vector<vector<int>>& connections) {
        vector<int> parent(n);
        vector<int> rank(n,0);
```

Saved to local



Description Editorial Solutions Submissions

C++ Auto

721. Accounts Merge

Hint

Medium 6.3K 1.1K

Companies

Given a list of `accounts` where each element `accounts[i]` is a list of strings, where the first element `accounts[i][0]` is a name, and the rest of the elements are **emails** representing emails of the account.

Now, we would like to merge these accounts. Two accounts definitely belong to the same person if there is some common email to both accounts. Note that even if two accounts have the same name, they may belong to different people as people could have the same name. A person can have any number of accounts initially, but all of their accounts definitely have the same name.

After merging the accounts, return the accounts in the following format: the first element of each account is the name, and the rest of the elements are emails **in sorted order**. The accounts themselves can be returned in **any order**.

```cpp
class Solution {
public:
    int findParent(vector<int> &parent,int node){
        if(parent[node]==node){
            return node;
        }
        return parent[node]=findParent(parent,parent[node]);
    }

    void unionSet(int u,int v,vector<int> &parent,vector<int> &rank){
        u=findParent(parent,u);
        v=findParent(parent,v);
        if(rank[u]<rank[v]){
            parent[u]=v;
            rank[v]++;
        }
        else{
            parent[v]=u;
            rank[u]++;
        }
    }

    vector<vector<string>> accountsMerge(vector<vector<string>>& accounts) {
        int n=accounts.size();
```

Saved to local

Console Run

1334. Find the City With the Smallest Number of Neighbors at a Threshold Distance

Hint

Medium 2.4K 82

Companies

There are `n` cities numbered from `0` to `n-1`. Given the array `edges` where `edges[i] = [from_i, to_i, weight_i]` represents a bidirectional and weighted edge between cities `from_i` and `to_i`, and given the integer `distanceThreshold`.

Return the city with the smallest number of cities that are reachable through some path and whose distance is **at most** `distanceThreshold`, If there are multiple such cities, return the city with the greatest number.

Notice that the distance of a path connecting cities ***i*** and ***j*** is equal to the sum of the edges' weights along that path.

```cpp
class Solution {
public:
    int  shortestDistDijkstra(int src,int n,unordered_map<int, list<pair<int,int>>> &adjList,int &distanceThreshold){
        vector<int>dist(n,INT_MAX);

        set<pair<int,int> >st;
        dist[src]=0;
        st.insert(make_pair(0,src));

        int reachableCities=0;

        while(!st.empty()){
            auto topElement=*(st.begin());

            st.erase(st.begin());

            int nodeDistance=topElement.first;
            int node=topElement.second;

            if(node!=src && nodeDistance<=distanceThreshold){
                ++reachableCities;
            }


            for(auto neighbour:adjList[node]){
```

Saved to local Ln 58

Console Run Subm