Day15: (String)

1. Reverse Words in a String

151. Reverse Words in a String

Medium 2484 3635 Add to List Share

Given an input string `s`, reverse the order of the **words**.

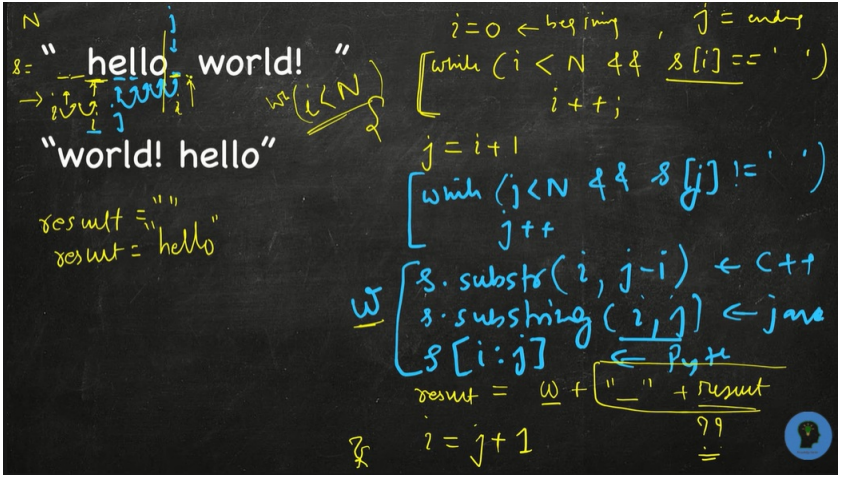
A **word** is defined as a sequence of non-space characters. The **words** in `s` will be separated by at least one space.

Return *a string of the words in reverse order concatenated by a single space.*

Note that `s` may contain leading or trailing spaces or multiple spaces between two words. The returned string should only have a single space separating the words. Do not include any extra spaces.

Example 1:

```
Input: s = "the sky is blue"
Output: "blue is sky the"
```




```java
class Solution {
    public String reverseWords(String s) {
        String res = new String();
        int i = 0;
        int n = s.length();
        while(i<n){
            while(i<n && s.charAt(i)==' ') i++;
            if(i>=n) break;
            int j = i + 1;
            while(j<n && s.charAt(j)!=' ') j++;
            String sub = s.substring(i,j);
            if(res.length()==0) res = sub;
            else res = sub + " " + res;
            i = j + 1;
        }
        return res;
    }
}
```

2. Longest Palindrome in a string

5. Longest Palindromic Substring

Medium 14316 842 Add to List Share

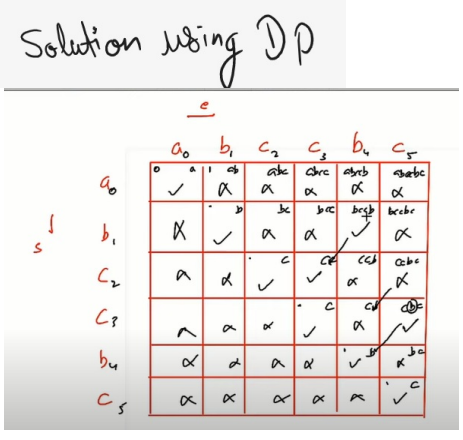
Given a string `s`, return *the longest palindromic substring* in `s`.

Example 1:

```
Input: s = "babad"
Output: "bab"
Note: "aba" is also a valid answer.
```

Example 2:

```
Input: s = "cbbd"
Output: "bb"
```




```java
class Solution {
    public String longestPalindrome(String s) {
        if(s==null || s.length()<1) return "";
        int start = 0, end = 0;
        for(int i=0; i<s.length(); i++){
            int len1 = expandAroundCenter(s,i,i);
            int len2 = expandAroundCenter(s,i,i+1);
            int len = Math.max(len1,len2);
            if(len>end-start){
                start = i - (len-1)/2;
                end = i + len/2;
            }
        }
        return s.substring(start,end+1);
    }
    private int expandAroundCenter(String s, int li, int ri){
        while(li>=0 && ri<s.length() && s.charAt(li)==s.charAt(ri)){
            li--;
            ri++;
        }
        return ri-li-1;
    }
}
```

Complexity Analysis

- Time complexity : $O(n^2)$. Since expanding a palindrome around its center could take $O(n)$ time, the overall complexity is $O(n^2)$.
- Space complexity : $O(1)$.

3. Roman Number to Integer and vice versa

13. Roman to Integer

Easy 2238 164 Add to List Share

Roman numerals are represented by seven different symbols: `I`, `V`, `X`, `L`, `C`, `D` and `M`.

Symbol	Value
I	1
V	5
X	10
L	50
C	100
D	500
M	1000

For example, `2` is written as `II` in Roman numeral, just two one's added together. `12` is written as `XII`, which is simply `X + II`. The number `27` is written as `XXVII`, which is `XX + V + II`.

Roman numerals are usually written largest to smallest from left to right. However, the numeral for four is not `IIII`. Instead, the number four is written as `IV`. Because the one is before the five we subtract it making four. The same principle applies to the number nine, which is written as `IX`. There are six instances where subtraction is used:

- `I` can be placed before `V` (5) and `X` (10) to make 4 and 9.
- `X` can be placed before `L` (50) and `C` (100) to make 40 and 90.
- `C` can be placed before `D` (500) and `M` (1000) to make 400 and 900.

Given a roman numeral, convert it to an integer.

Example 1:

```
Input: s = "III"
Output: 3
```

Example 2:

```
Input: s = "IV"
Output: 4
```

Example 3:

```
Input: s = "IX"
Output: 9
```

Example 4:

```
Input: s = "LVIII"
Output: 58
Explanation: L = 50, V= 5, III = 3.
```

Example 5:

```
Input: s = "MCMXCIV"
Output: 1994
Explanation: M = 1000, CM = 900, XC = 90 and IV = 4.
```

Constraints:

- `1 <= s.length <= 15`
- `s` contains only the characters `('I', 'V', 'X', 'L', 'C', 'D', 'M')`.
- It is **guaranteed** that `s` is a valid roman numeral in the range `[1, 3999]`.

```java
class Solution {
    public int romanToInt(String s) {
        Map<Character,Integer> map = new HashMap<>();
        map.put('I',1);
        map.put('V',5);
        map.put('X',10);
        map.put('L',50);
        map.put('C',100);
        map.put('D',500);
        map.put('M',1000);

        int result = 0;
        for(int i=0; i<s.length(); i++){
            if(i>0 && map.get(s.charAt(i)) > map.get(s.charAt(i-1))){
                result += map.get(s.charAt(i)) - 2*map.get(s.charAt(i-1));;
            }else{
                result += map.get(s.charAt(i));
            }
        }
        return result;

    }
}
```

12. Integer to Roman

Medium 2239 3513 Add to List Share

Roman numerals are represented by seven different symbols: `I`, `V`, `X`, `L`, `C`, `D` and `M`.

Symbol	Value
I	1
V	5
X	10
L	50
C	100
D	500
M	1000

For example, `2` is written as `II` in Roman numeral, just two one's added together. `12` is written as `XII`, which is simply `X + II`. The number `27` is written as `XXVII`, which is `XX + V + II`.

```
class Solution {
    public String intToRoman(int num) {
        String[] thousands = new String[]{"","M","MM","MMM"};
        String[] hundreds = {"","C","CC","CCC","CD","D","DC","DCC","DCCC","CM"};
        String[] tens = {"","X","XX","XXX","XL","L","LX","LXX","LXXX","XC"};
        String[] units = {"","I","II","III","IV","V","VI","VII","VIII","IX"};

        return thousands[num/1000]+
            hundreds[(num % 1000)/100]+
            tens[(num % 100)/10]+
            units[num % 10];
    }
}
```


12. Integer to Roman

Medium 2239 3513 Add to List Share

Roman numerals are represented by seven different symbols: `I`, `V`, `X`, `L`, `C`, `D` and `M`.

Symbol	Value
I	1
V	5
X	10
L	50
C	100
D	500
M	1000


```
int[] values = {1000,900,500,400,100,90,50,40,10,9,5,4,1};
    String[] strs = {"M","CM","D","CD","C","XC","L","XL","X","IX","V","IV","I"};

    StringBuilder sb = new StringBuilder();

    for(int i=0;i<values.length;i++) {
        while(num >= values[i]) {
            num -= values[i];
            sb.append(strs[i]);
        }
    }
    return sb.toString();
}
```

12. Integer to Roman

Medium 2239 3513 Add to List Share

Roman numerals are represented by seven different symbols: `I`, `V`, `X`, `L`, `C`, `D` and `M`.

Symbol	Value
I	1
V	5
X	10
L	50
C	100
D	500
M	1000

For example, `2` is written as `II` in Roman numeral, just two one's added together. `12` is written as `XII`, which is simply `X + II`. The number `27` is written as `XXVII`, which is `XX + V + II`.

Roman numerals are usually written largest to smallest from left to right. However, the numeral for four is not `IIII`. Instead, the number four is written as `IV`. Because the one is before the five we subtract it making four. The same principle applies to the number nine, which is written as `IX`. There are six instances where subtraction is used:

- `I` can be placed before `V` (5) and `X` (10) to make 4 and 9.
- `X` can be placed before `L` (50) and `C` (100) to make 40 and 90.
- `C` can be placed before `D` (500) and `M` (1000) to make 400 and 900.

Given an integer, convert it to a roman numeral.

Example 1:

```
Input: num = 3
Output: "III"
```

Example 2:

```
Input: num = 4
Output: "IV"
```

Example 3:

```
Input: num = 9
Output: "IX"
```

```java
class Solution {
    public String intToRoman(int num) {
        StringBuilder sb = new StringBuilder();
            if(num>=1000){
                num = process(num, sb, 1000,'M');
            }
            if(num>=500){
                if(num>=900){
                    sb.append("CM");
                    num = num-900;
                }else
                num = process(num, sb, 500,'D');
            }
            if(num>=100){
                if(num>=400){
                    sb.append("CD");
                    num = num-400;
                }else
                num = process(num, sb, 100,'C');
            }
            if(num>=50){
                if(num>=90){
                    sb.append("XC");
                    num = num - 90;
                }else
                num = process(num, sb, 50, 'L');
            }
            if(num>=10){
                if(num >= 40){
                    sb.append("XL");
                    num = num - 40;
                } else
                num = process(num, sb, 10, 'X');
            }
            if(num>=5){
                if(num==9) {
                    sb.append("IX");
                    num = 0;
                }
                else num = process(num, sb, 5, 'V');
            }
            if(num>=1){
                if(num==4) {
                    sb.append("IV");
                    num = 0;
                }
                else num = process(num, sb, 1, 'I');
            }
        return sb.toString();
    }

    public int process(int num, StringBuilder sb, int x, char symbol){
        int temp = num/x;
        num = num%x;
        while(temp>0){
            sb.append(symbol);
            temp--;
        }
        return num;
    }
}
```

4. Implement ATOI/STRSTR

8. String to Integer (atoi)

Medium 822 2313 Add to List Share

Implement the `myAtoi(string s)` function, which converts a string to a 32-bit signed integer (similar to C/C++'s `atoi` function).

The algorithm for `myAtoi(string s)` is as follows:

1. Read in and ignore any leading whitespace.
2. Check if the next character (if not already at the end of the string) is `'-'` or `'+'`. Read this character in if it is either. This determines if the final result is negative or positive respectively. Assume the result is positive if neither is present.
3. Read in next the characters until the next non-digit character or the end of the input is reached. The rest of the string is ignored.
4. Convert these digits into an integer (i.e. `"123" -> 123`, `"0032" -> 32`). If no digits were read, then the integer is `0`. Change the sign as necessary (from step 2).
5. If the integer is out of the 32-bit signed integer range $[-2^{31}, 2^{31} - 1]$, then clamp the integer so that it remains in the range. Specifically, integers less than -2^{31} should be clamped to -2^{31}, and integers greater than $2^{31} - 1$ should be clamped to $2^{31} - 1$.
6. Return the integer as the final result.

Example 1:

```
Input: s = "42"
Output: 42
Explanation: The underlined characters are what is read in, the caret is the current reader position.
Step 1: "42" (no characters read because there is no leading whitespace)
         ^
Step 2: "42" (no characters read because there is neither a '-' nor '+')
         ^
Step 3: "42" ("42" is read in)
           ^
The parsed integer is 42.
Since 42 is in the range [-2^31, 2^31 - 1], the final result is 42.
```

Example 2:

```
Input: s = "   -42"
Output: -42
Explanation:
Step 1: "___-42" (leading whitespace is read and ignored)
            ^
Step 2: "   -42" ('-' is read, so the result should be negative)
             ^
Step 3: "   -42" ("42" is read in)
               ^
The parsed integer is -42.
Since -42 is in the range [-2^31, 2^31 - 1], the final result is -42.
```

Example 3:

```
Input: s = "4193 with words"
Output: 4193
Explanation:
Step 1: "4193 with words" (no characters read because there is no leading whitespace)
         ^
Step 2: "4193 with words" (no characters read because there is neither a '-' nor '+')
         ^
Step 3: "4193 with words" ("4193" is read in; reading stops because the next character is a non-digit)
             ^
The parsed integer is 4193.
Since 4193 is in the range [-2^31, 2^31 - 1], the final result is 4193.
```

Example 4:

```
Input: s = "words and 987"
Output: 0
Explanation:
Step 1: "words and 987" (no characters read because there is no leading whitespace)
         ^
Step 2: "words and 987" (no characters read because there is neither a '-' nor '+')
         ^
Step 3: "words and 987" (reading stops immediately because there is a non-digit 'w')
         ^
The parsed integer is 0 because no digits were read.
Since 0 is in the range [-2^31, 2^31 - 1], the final result is 0.
```

```java
class Solution {
    public int myAtoi(String s) {
        int i = 0;
        int result = 0;
        int sign = 1;
        //base
        if (s.length() == 0) return 0;

        //discard white space
        while (i < s.length() && s.charAt(i) == ' ') i++;

        //check the sign
        if (i < s.length() && (s.charAt(i) == '+' || s.charAt(i) == '-'))
            sign = (s.charAt(i++) == '-') ? -1 : 1;

        // proceed only if the char is digits
        while (i < s.length() && s.charAt(i) >= '0' && s.charAt(i) <= '9'){
            // Since we are doing r =  r * 10 + digit formula
            // when r > max / 10 if u do * 10 it will overflow
            // if r == max / 10 then any number + 7 will overflow
            // same case for under flow also
            // both case we should handle
            if (result > Integer.MAX_VALUE / 10 ||
                (result == Integer.MAX_VALUE / 10 && s.charAt(i) - '0' > Integer.MAX_VALUE % 10))
                return (sign == 1) ? Integer.MAX_VALUE : Integer.MIN_VALUE;

            result = result * 10 + (s.charAt(i++) - '0');
        }

        return result * sign;
    }
}
```

5. Longest Common Prefix

14. Longest Common Prefix

Easy 5943 2588 Add to List Share

Write a function to find the longest common prefix string amongst an array of strings.

If there is no common prefix, return an empty string `""`.

Example 1:

```
Input: strs = ["flower","flow","flight"]
Output: "fl"
```

Example 2:

```
Input: strs = ["dog","racecar","car"]
Output: ""
Explanation: There is no common prefix among the input
strings.
```

6. Rabin Karp

Day-16: String [Continued]

28. Implement strStr()

Easy 3144 2954 Add to List Share

Implement strStr().

Return the index of the first occurrence of needle in haystack, or `-1` if `needle` is not part of `haystack`.

Clarification:

What should we return when `needle` is an empty string? This is a great question to ask during an interview.

For the purpose of this problem, we will return 0 when `needle` is an empty string. This is consistent to C's strstr() and Java's indexOf().

Example 1:

```
Input: haystack = "hello", needle = "ll"
Output: 2
```

Example 2:

```
Input: haystack = "aaaaa", needle = "bba"
Output: -1
```

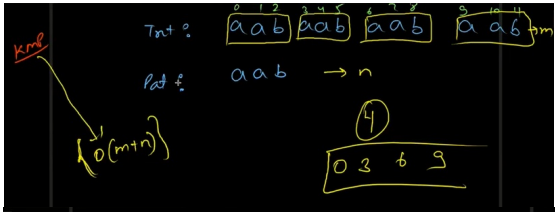
Example 3:

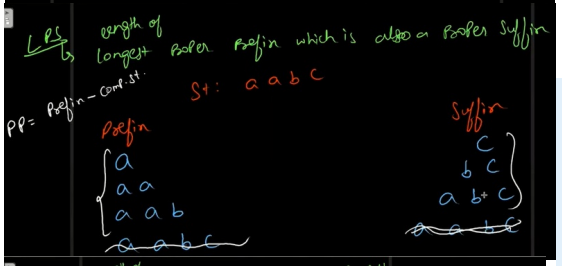
```
Input: haystack = "", needle = ""
Output: 0
```

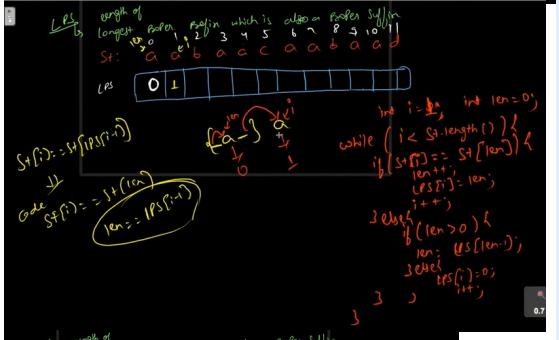
Constraints:

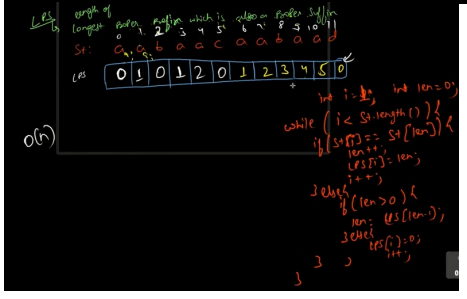
- `0 <= haystack.length, needle.length <= 5 * 10^4`
- `haystack` and `needle` consist of only lower-case English characters.

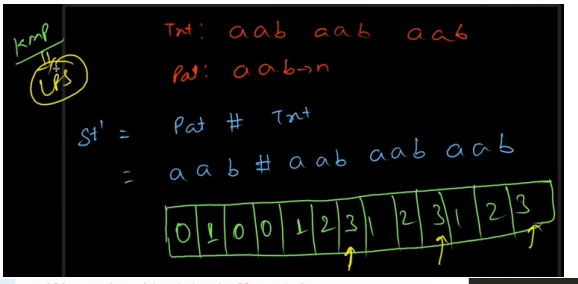
KMP algo / LPS(pi) array












```java
public static void main(String[] args) {
    Scanner scn = new Scanner(System.in);
    String s1 = scn.nextLine();
    String s2 = scn.nextLine();
    solution(s1, s2);
}

public static void solution(String txt, String pat) {
    int[] lps = preprocess(pat);
    int i = 0;
    int j = 0;
    while (i < txt.length()) {
        if (txt.charAt(i) == pat.charAt(j)) {
            i++;
            j++;
            if (j == pat.length()) {
                System.out.println(i - j);
                j = lps[j - 1];
            }
        } else {
            if (j > 0) {
                j = lps[j - 1];
            } else {
                i++;
            }
        }
    }
}
private static int[] preprocess(String p) {
    int[] lps = new int[p.length()];
    int i = 1;
    int len = 0;
    while (i < p.length()) {
        if (p.charAt(i) == p.charAt(len)) {
            len++;
            lps[i] = len;
            i++;
        } else {
            if (len > 0) {
                len = lps[len - 1];
            } else {
                i++;
            }
        }
    }
    return lps;
}
}
```

```java
class Solution {
    public int strStr(String haystack, String needle) {
        if (haystack == null || needle == null) return -1;
        int nLen = needle.length();
        int hLen = haystack.length();
        if (nLen == 0) return 0;
        if (hLen == 0) return -1;
        int[] table = kmpLookupTable(needle);
        int i = 0;
        int j = 0;
        while (i < hLen && j < nLen) {
            if (haystack.charAt(i) == needle.charAt(j)) {
                i++;
                j++;
            } else {
                if (j > 0) {
                    j = table[j - 1];
                } else {
                    i++;
                }
            }
        }
        if (j == nLen) return i - j;
        return -1;
    }
    private int[] kmpLookupTable(String s) {
        int[] table = new int[s.length()];
        int i = 1,index = 0;
        while (i < s.length()) {
            if (s.charAt(i) == s.charAt(index)) {
                table[i] = index + 1;
                index++;
                i++;
            } else {
                if (index > 0) {
                    index = table[index - 1];
                } else {
                    table[i] = 0;
                    i++;
                }
            }
        }
        return table;
    }
}
```

Minimum Characters required to make a String Palindromic

Medium | 119 | 16 | Add to favorites

Asked In: AMAZON MICROSOFT

Given an string **A**. The only operation allowed is to insert characters in the beginning of the string.

Find how many minimum characters are needed to be inserted to make the string a palindrome string.

Input Format

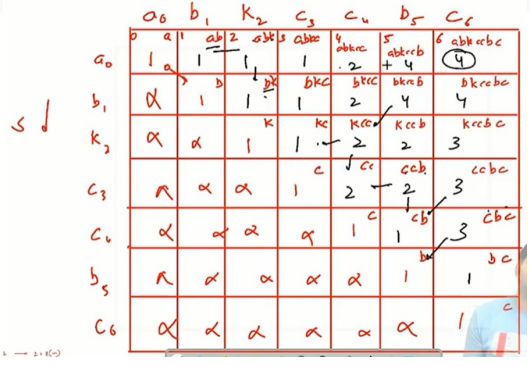
The only argument given is string A.

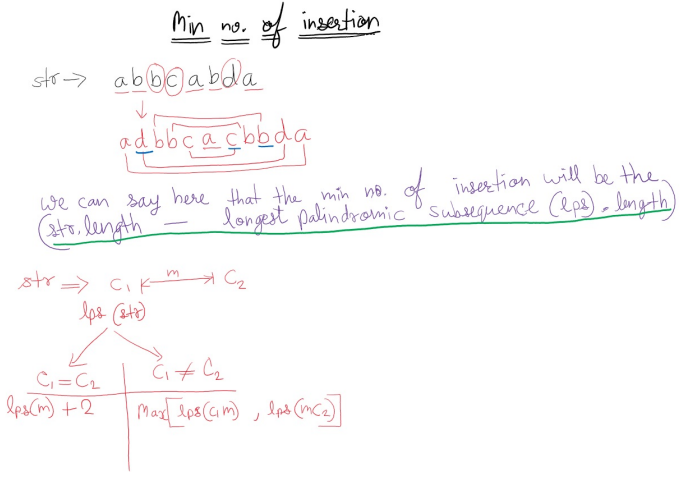
Output Format

Return the minimum characters that are needed to be inserted to make the string a palindrome string.

For Example

```
Input 1:
    A = "ABC"
Output 1:
    2
    Explanation 1:
        Insert 'B' at beginning, string becomes: "BABC".
        Insert 'C' at beginning, string becomes: "CBABC".

Input 2:
    A = "AACECAAAA"
Output 2:
    2
    Explanation 2:
        Insert 'A' at beginning, string becomes: "AAACECAAAA".
        Insert 'A' at beginning, string becomes: "AAAACECAAAA".
```






```java
import java.io.*;
import java.util.*;

public class Main {

    public static int solution(String str) {
        int[][] dp = new int[str.length()][str.length()];

        for(int gap = 0; gap < dp.length; gap++){
            for(int si = 0, ei = gap ;ei < dp.length; si++, ei++){
                if(gap == 0){
                    dp[si][ei] = 1;
                }else{
                    if(str.charAt(si) == str.charAt(ei)){
                        dp[si][ei] = dp[si + 1][ei - 1] + 2;
                    }else{
                        dp[si][ei] = Math.max(dp[si + 1][ei], dp[si][ei - 1]);
                    }
                }
            }
        }

        int ans = str.length() - dp[0][dp.length - 1];
        return ans;
    }

    public static void main(String[] args) {
        Scanner scn = new Scanner(System.in);
        String str = scn.next();
        System.out.println(solution(str));
    }
}
```

242. Valid Anagram

Easy 3573 189 Add to List Share

Given two strings `s` and `t`, return `true` *if* `t` *is an anagram of* `s`, *and* `false` *otherwise.*

Example 1:

```
Input: s = "anagram", t = "nagaram"
Output: true
```

Example 2:

```
Input: s = "rat", t = "car"
Output: false
```

```java
class Solution {
    public boolean isAnagram(String s, String t) {
        if (s.length() != t.length()) return false;

        int[] char_counts = new int[26];
        for (int i=0; i<s.length(); i++) {
            char_counts[s.charAt(i)-'a']++;
            char_counts[t.charAt(i)-'a']--;
        }

        for (int count : char_counts) {
            if (count != 0) {
                return false;
            }
        }

        return true;
    }
}
```

```java
import java.util.*;

public class Main {

    public static boolean solution(String s1, String s2){
        HashMap<Character, Integer> map = new HashMap<>();
        for(int i = 0; i < s1.length(); i++){
            char ch = s1.charAt(i);
            map.put(ch, map.getOrDefault(ch, 0) + 1);
        }

        for(int i = 0; i < s2.length(); i++){
            char ch = s2.charAt(i);

            if(map.containsKey(ch) == false){
                return false;
            } else if(map.get(ch) == 1){
                map.remove(ch);
            } else {
                map.put(ch, map.get(ch) - 1);
            }
        }

        return map.size() == 0;
    }
    public static void main(String[] args) {
        Scanner scn = new Scanner(System.in);
        String s1 = scn.next();
        String s2 = scn.next();
        System.out.println(solution(s1,s2));
    }

}
```

38. Count and Say

Medium 950 2733 Add to List Share

The **count-and-say** sequence is a sequence of digit strings defined by the recursive formula:

- `countAndSay(1) = "1"`
- `countAndSay(n)` is the way you would "say" the digit string from `countAndSay(n-1)`, which is then converted into a different digit string.

To determine how you "say" a digit string, split it into the **minimal** number of groups so that each group is a contiguous section all of the **same character.** Then for each group, say the number of characters, then say the character. To convert the saying into a digit string, replace the counts with a number and concatenate every saying.

For example, the saying and conversion for digit string `"3322251"`:

"3322251"
two 3's, three 2's, one 5, and one 1
2 3 + 3 2 + 1 5 + 1 1
"23321511"

Given a positive integer `n`, return *the* `n^th` *term of the* ***count-and-say*** *sequence.*

Example 1:

```
Input: n = 1
Output: "1"
Explanation: This is the base case.
```

Example 2:

```
Input: n = 4
Output: "1211"
Explanation:
countAndSay(1) = "1"
countAndSay(2) = say "1" = one 1 = "11"
countAndSay(3) = say "11" = two 1's = "21"
countAndSay(4) = say "21" = one 2 + one 1 = "12" + "11" = "1211"
```

```java
class Solution {
    public String countAndSay(int n) {
        String s = "1";
        if (n == 1)
            return s;
        for (int i = 1; i < n; i++) {
            s = helper(s);
        }
        return s;
    }

    private String helper(String num) {
        StringBuilder n = new StringBuilder();
        int i = 0, j = 0;
        while (i < num.length()) { // get each character count
            while (j < num.length() && num.charAt(j) == num.charAt(i)) {
                j++;
            }
            int freq = j-i;
            n.append(freq).append(num.charAt(i)); //store freq followed by character to stringbuilder
            i = j;
        }
        return n.toString();
    }
}
```

```cpp
class Solution {
public:
    string countAndSay(int n) {
        if(n==1) return "1";
        if(n==2) return "11";
        string s="11";
        for(int i=3;i<=n;i++){
            string t="";
            s=s+'&';
            int c=1;
            for(int j=1;j<s.length();j++){
                if(s[j]!=s[j-1]){
                    t=t+to_string(c);
                    t=t+s[j-1];
                    c=1;
                }
                else c++;
            }
            s=t;
        }
        return s;
    }
};
```

165. Compare Version Numbers

Medium 984 1810 Add to List Share

Given two version numbers, `version1` and `version2`, compare them.

Version numbers consist of **one or more revisions** joined by a dot `'.'`. Each revision consists of **digits** and may contain leading **zeros**. Every revision contains **at least one character**. Revisions are **0-indexed from left to right**, with the leftmost revision being revision 0, the next revision being revision 1, and so on. For example `2.5.33` and `0.1` are valid version numbers.

To compare version numbers, compare their revisions in **left-to-right order**. Revisions are compared using their **integer value ignoring any leading zeros**. This means that revisions `1` and `001` are considered **equal**. If a version number does not specify a revision at an index, then **treat the revision as** `0`. For example, version `1.0` is less than version `1.1` because their revision 0s are the same, but their revision 1s are `0` and `1` respectively, and `0 < 1`.

Return the following:

- If `version1 < version2`, return `-1`.
- If `version1 > version2`, return `1`.

Return the following:

- If `version1 < version2`, return `-1`.
- If `version1 > version2`, return `1`.
- Otherwise, return `0`.

Example 1:

```
Input: version1 = "1.01", version2 = "1.001"
Output: 0
Explanation: Ignoring leading zeroes, both "01" and "001" represent the same integer "1".
```

Example 2:

```
Input: version1 = "1.0", version2 = "1.0.0"
Output: 0
Explanation: version1 does not specify revision 2, which means it is treated as "0".
```

Example 3:

```
Input: version1 = "0.1", version2 = "1.1"
Output: -1
Explanation: version1's revision 0 is "0", while version2's revision 0 is "1". 0 < 1, so version1 < version2.
```

Example 4:

```
Input: version1 = "1.0.1", version2 = "1"
Output: 1
```

Example 5:

```
Input: version1 = "7.5.2.4", version2 = "7.5.3"
Output: -1
```

Constraints:

- `1 <= version1.length, version2.length <= 500`
- `version1` and `version2` only contain digits and `'.'`.
- `version1` and `version2` **are valid version numbers**.
- All the given revisions in `version1` and `version2` can be stored in a **32-bit integer**.

```java
class Solution {
    public int compareVersion(String version1, String version2) {
        String[] ver1=version1.split("\\.");
        String[] ver2=version2.split("\\.");
        int first=0, second=0;
        while(first<ver1.length||second<ver2.length){
            int val1=first<ver1.length?Integer.valueOf(ver1[first]):0;
            int val2=second<ver2.length?Integer.valueOf(ver2[first]):0;
            if(val1>val2){
                return 1;

            }else if (val1<val2){
                return -1;
            }else {
                first++;
                second++;
            }
        }
        return 0;
    }
}
```

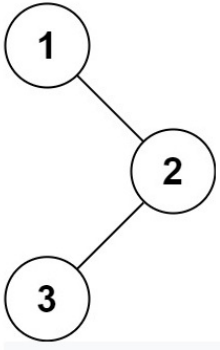
Day-17: Binary Tree

94. Binary Tree Inorder Traversal

Easy 6178 260 Add to List Share

Given the `root` of a binary tree, return *the inorder traversal of its nodes' values.*

Example 1:




```
Input: root = [1,null,2,3]
Output: [1,3,2]
```

Example 2:

```
Input: root = []
Output: []
```

```java
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode() {}
 *     TreeNode(int val) { this.val = val; }
 *     TreeNode(int val, TreeNode left, TreeNode right) {
 *         this.val = val;
 *         this.left = left;
 *         this.right = right;
 *     }
 * }
 */
class Solution {
    public List<Integer> inorderTraversal(TreeNode root) {
        List<Integer> res = new ArrayList<>();
        helper(root,res);
        return res;
    }
    public void helper(TreeNode root, List<Integer> res){
        if(root != null){
            helper(root.left,res);
            res.add(root.val);
            helper(root.right,res);
        }
    }
}
```

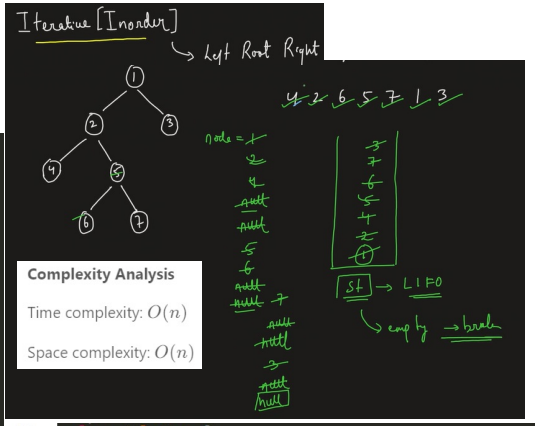
Complexity Analysis

Time complexity: $O(n)$

- The time complexity is $O(n)$ because the recursive function is $T(n) = 2 \cdot T(n/2) + 1$.

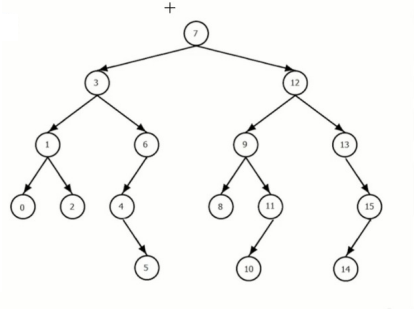
Space complexity: $O(n)$

- The worst case space required is $O(n)$, and in the average case it's $O(\log n)$ where n is number of nodes.




```java
class Solution {
    public List<Integer> inorderTraversal(TreeNode root) {
        List<Integer> res = new ArrayList<>();
        Stack<TreeNode> st = new Stack<>();
        TreeNode curr = root;
        while(curr != null || !st.isEmpty()){
            while(curr != null){
                st.push(curr);
                curr = curr.left;
            }
            curr = st.pop();
            res.add(curr.val);
            curr = curr.right;
        }
        return res;
    }
}
```

1. print :
   a. if left is null
   b. if threade is cut down then print current node
2. Mark of left subtree is completely processed or not :
   if threade already exists



1. right -> if left not exist or either left subtree is completely proces
2. something mark so that we get to know whether left subtree is completely processed of not.

```java
public static TreeNode getRightMostNode(TreeNode leftNode,TreeNode curr){

    while(leftNode.right != null && leftNode.right != curr){
        leftNode = leftNode.right;
    }

    return leftNode;
```

```java
while(curr != null){
    TreeNode leftNode = curr.left;
    if(leftNode == null){
        ans.add(curr.val);
        curr = curr.right;
    }else{

        TreeNode rightMostNode = getRightMostNode(leftNode,curr

        if(rightMostNode.right == null){  // threade create
            rightMostNode.right = curr;
            curr = curr.left;
        }else{  // threade cut down
            rightMostNode.right = null;
            ans.add(curr.val);
            curr = curr.right;
        }
    }
```

Approach 3: Morris Traversal

In this method, we have to use a new data structure-Threaded Binary Tree, and the strategy is as follows:

> Step 1: Initialize current as root
>
> Step 2: While current is not NULL,
>
> If current does not have left child
>
> a. Add current's value
>
> b. Go to the right, i.e., current = current.right
>
> Else
>
> a. In current's left subtree, make current the right child of the rightmost node
>
> b. Go to this left child, i.e., current = current.left

For example:

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

First, 1 is the root, so initialize 1 as current, 1 has left child which is 2, the current's left subtree is

```
    2
   / \
  4   5
```

So in this subtree, the rightmost node is 5, then make the current(1) as the right child of 5. Set current = cuurent.left (current = 2). The tree now looks like:

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

For current 2, which has left child 4, we can continue with thesame process as we did above

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

then add 4 because it has no left child, then add 2, 5, 1, 3 one by one, for node 3 which has left child 6, do the same as above. Finally, the inorder taversal is [4,2,5,1,6,3].

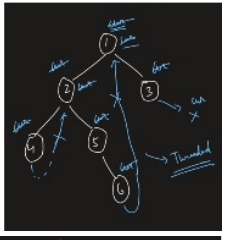
```java
class Solution {
    public List<Integer> inorderTraversal(TreeNode root) {
        List<Integer> res = new ArrayList<>();
        TreeNode curr = root;
        TreeNode pre;
        while (curr != null) {
            if (curr.left == null) {
                res.add(curr.val);
                curr = curr.right; // move to next right node
            } else { // has a left subtree
                pre = curr.left;
                while (pre.right != null) { // find rightmost
                    pre = pre.right;
                }
                pre.right = curr; // put cur after the pre node
                TreeNode temp = curr; // store cur node
                curr = curr.left; // move cur to the top of the new tree
                temp.left = null; // original cur left be null, avoid infinite loops
            }
        }
        return res;
    }
}
```

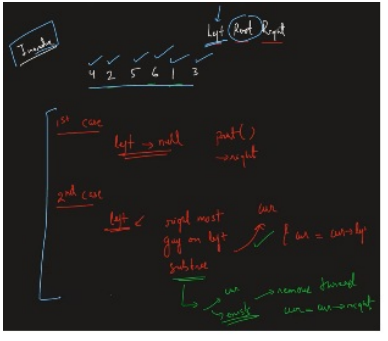
Time complexity: $O(n)$

- To prove that the time complexity is $O(n)$, the biggest problem lies in finding the time complexity of finding the predecessor nodes of all the nodes in the binary tree. Intuitively, the complexity is $O(n \log n)$, because to find the predecessor node for a single node related to the height of the tree. But in fact, finding the predecessor nodes for all nodes only needs $O(n)$ time. Because a binary Tree with n nodes has $n-1$ edges, the whole processing for each edges up to 2 times, one is to locate a node, and the other is to find the predecessor node. So the complexity is $O(n)$.

Space complexity: $O(1)$

- Extra space is only allocated for the ArrayList of size n, however the output does not count towards the space complexity.



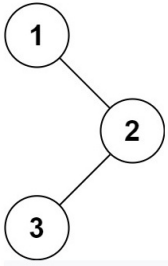



```java
class Solution {
    public List<Integer> inorderTraversal(TreeNode root) {
        List<Integer> inorder = new ArrayList<Integer>();

        TreeNode cur = root;
        while(cur != null) {
            if(cur.left == null) {
                inorder.add(cur.val);
                cur = cur.right;
            }
            else {
                TreeNode prev = cur.left;
                while(prev.right != null && prev.right != cur) {
                    prev = prev.right;
                }

                if(prev.right == null) {
                    prev.right = cur;
                    cur = cur.left;
                }
                else {
                    prev.right = null;
                    inorder.add(cur.val);
                    cur = cur.right;
                }
            }
        }
        return inorder;
    }
}
```

Pre Order Morris Traversal In Binary Tree | Using O(1) Space

Given the `root` of a binary tree, return *the preorder traversal of its nodes' values.*

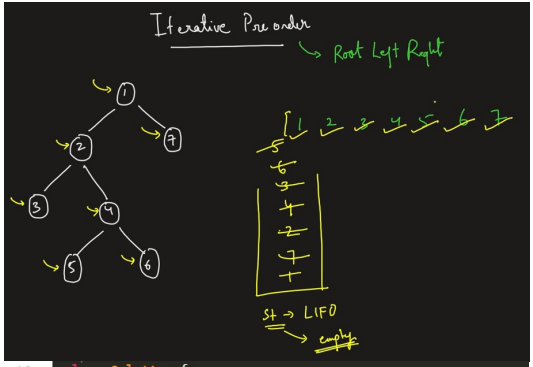
Example 1:




```
Input: root = [1,null,2,3]
Output: [1,2,3]
```

Example 2:

```
Input: root = []
Output: []
```




```java
class Solution {
    public List<Integer> preorderTraversal(TreeNode root) {
        List<Integer> preorder = new ArrayList<Integer>();
        if(root == null) return preorder;
        Stack<TreeNode> st = new Stack<TreeNode>();
        st.push(root);
        while(!st.isEmpty()){
            root = st.pop();
            preorder.add(root.val);
            if(root.right != null){
                st.push(root.right);
            }
            if(root.left!= null){
                st.push(root.left);
            }
        }
        return preorder;
    }
}
```

```java
class Solution {
    public List<Integer> preorderTraversal(TreeNode root) {
        List<Integer> res = new ArrayList<Integer>();
        helper(root,res);
        return res;
    }
    public void helper(TreeNode root, List<Integer> res){
        if(root!=null){
            res.add(root.val);
            helper(root.left,res);
            helper(root.right,res);
        }
    }
}
```

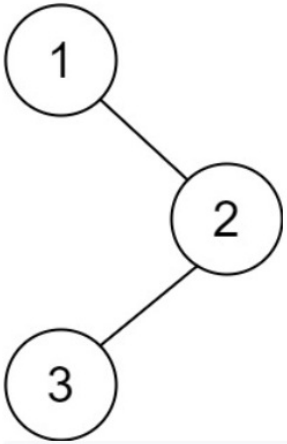
```java
public static TreeNode rightMostNode(TreeNode node, TreeNode curr) {
  while (node.right != null && node.right != curr) {
    node = node.right;
  }
  return node;
}

public static ArrayList<Integer> morrisPreTraversal(TreeNode Treenode) {
  ArrayList<Integer> ans = new ArrayList<>();
  TreeNode curr = Treenode;
  while (curr != null) {
    TreeNode leftTreeNode = curr.left;
    if (leftTreeNode == null) { // left null
      ans.add(curr.val);
      curr = curr.right;
    } else {
      TreeNode rmost = rightMostNode(leftTreeNode, curr);
      if (rmost.right == null) { // thread Creation
        rmost.right = curr;
        ans.add(curr.val);
        curr = curr.left;
      } else { // thread Break
        rmost.right = null;
        curr = curr.right;
      }
    }
  }
```

145. Binary Tree Postorder Traversal

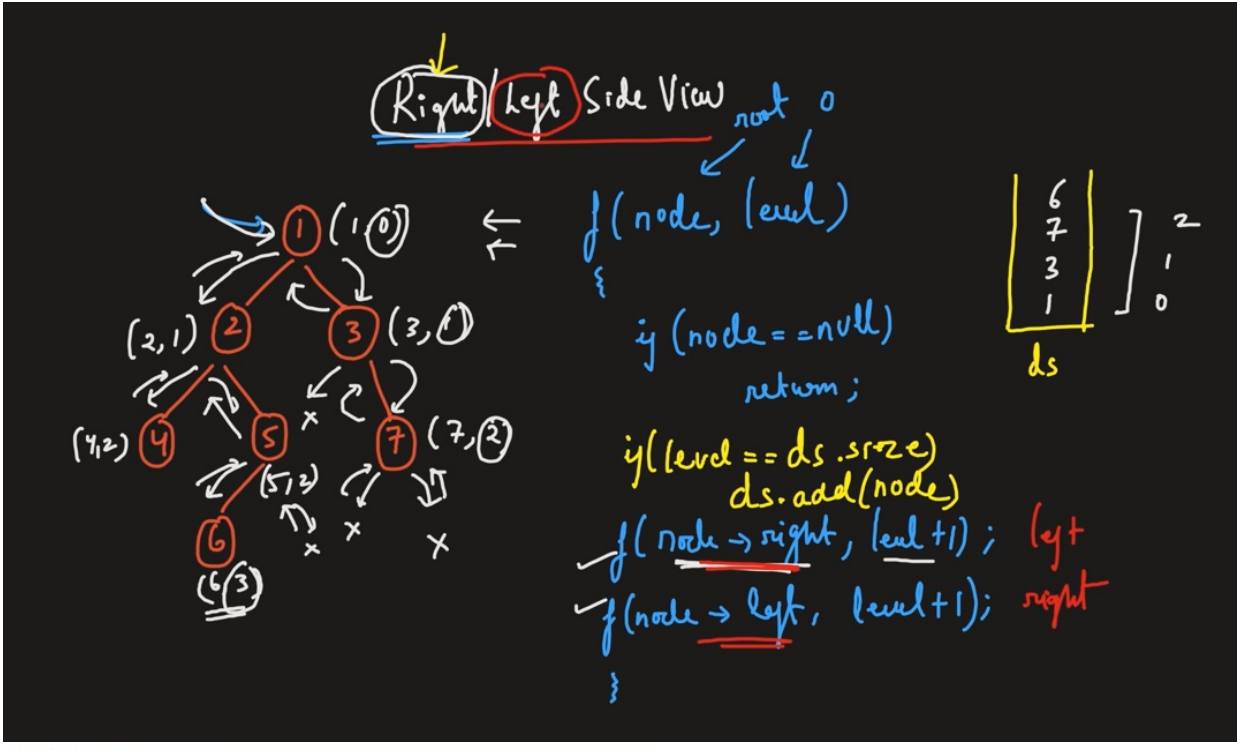
Easy 3355 128 Add to List Share

Given the `root` of a binary tree, return *the postorder traversal of its nodes' values.*

Example 1:




```
Input: root = [1,null,2,3]
Output: [3,2,1]
```

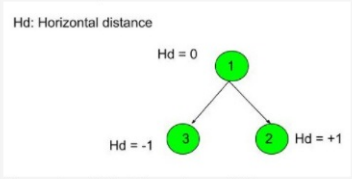



```java
public class Solution {
    public List<Integer> rightSideView(TreeNode root) {
        List<Integer> result = new ArrayList<Integer>();
        rightView(root, result, 0);
        return result;
    }

    public void rightView(TreeNode curr, List<Integer> result, int currDepth){
        if(curr == null){
            return;
        }
        if(currDepth == result.size()){
            result.add(curr.val);
        }
        rightView(curr.right, result, currDepth + 1);
        rightView(curr.left, result, currDepth + 1);

    }
}
```

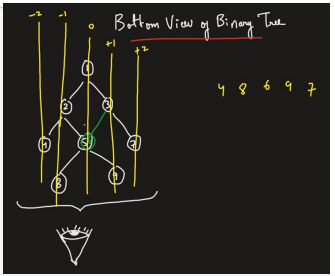
Bottom View of Binary Tree

Medium Accuracy: 45.32% Submissions: 92201 Points: 4

```
Input:
      1
    /   \
   3     2
Output: 3 1 2
Explanation:
First case represents a tree with 3 nodes
and 2 edges where root is 1, left child of
1 is 3 and right child of 1 is 2.
```



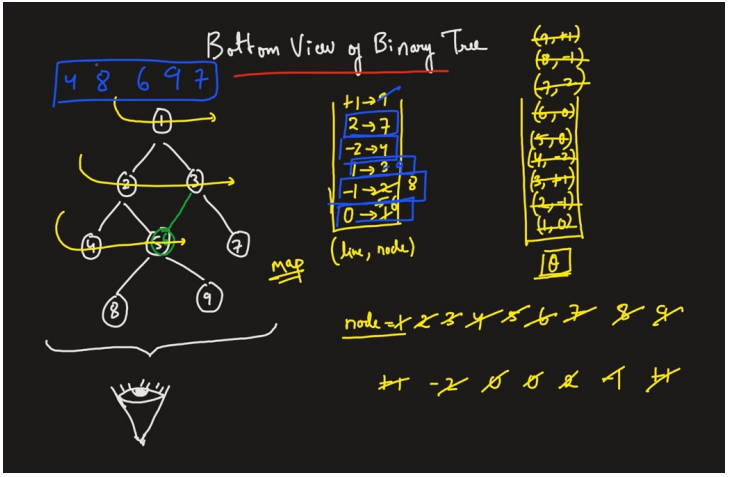

```
Thus nodes of the binary tree will be
printed as such 3 1 2.
```



Example 2:

```
Input:
        10
      /    \
     20    30
    /  \
   40   60
Output: 40 20 60 30
```

Your Task:
This is a functional problem, you **don't** need to care about input, just complete the function **bottomView**() which takes the root node of the tree as input and returns an array containing the bottom view of the given tree.




```java
class Solution
{
    //Function to return a list containing the bottom view of the given tree.
    public ArrayList <Integer> bottomView(Node root)
    {
        ArrayList<Integer> ans = new ArrayList<>();
        if(root == null) return ans;
        Map<Integer, Integer> map = new TreeMap<>();
        Queue<Node> q = new LinkedList<Node>();
        root.hd = 0;
        q.add(root);
        while(!q.isEmpty()) {
            Node temp = q.remove();
            int hd = temp.hd;
            map.put(hd, temp.data);
            if(temp.left != null) {
                temp.left.hd = hd - 1;
                q.add(temp.left);
            }
            if(temp.right != null) {
                temp.right.hd = hd + 1;
                q.add(temp.right);
            }
        }

        for (Map.Entry<Integer,Integer> entry : map.entrySet()) {
            ans.add(entry.getValue());
        }
        return ans;

    }
}
```

```cpp
class Solution {
  public:
    vector <int> bottomView(Node *root) {
        vector<int> ans;
        if(root == NULL) return ans;
        map<int,int> mpp;
        queue<pair<Node*, int>> q;
        q.push({root, 0});
        while(!q.empty()) {
            auto it = q.front();
            q.pop();
            Node* node = it.first;
            int line = it.second;
            mpp[line] = node->data;

            if(node->left != NULL) {
                q.push({node->left, line-1});
            }
            if(node->right != NULL) {
                q.push({node->right, line + 1});
            }

        }

        for(auto it : mpp) {
            ans.push_back(it.second);
        }
        return ans;
    }
};
```

Top View of Binary Tree

Medium Accuracy: 32.3% Submissions: 100k+ Points: 4

Given below is a binary tree. The task is to print the top view of binary tree. Top view of a binary tree is the set of nodes visible when the tree is viewed from the top. For the given below tree

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

Top view will be: 4 2 1 3 7
Note: Return nodes from **leftmost** node to **rightmost** node.

Example 1:

```
Input:
      1
    /   \
  2      3
Output: 2 1 3
```

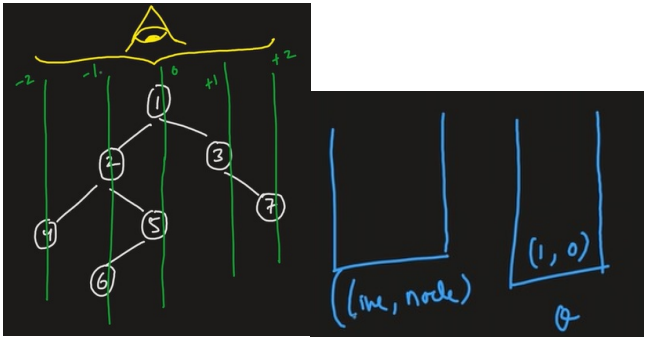
Example 2:

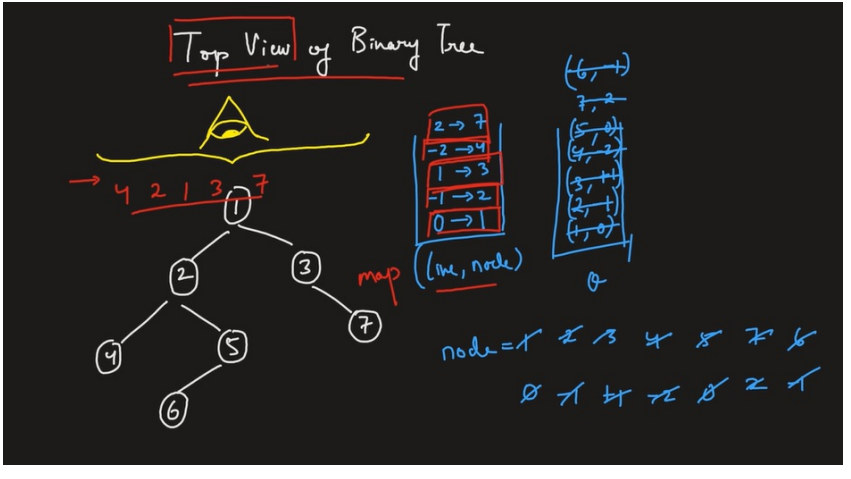
```
Input:
        10
     /      \
  20          30
 /   \      /    \
40   60   90     100
Output: 40 20 10 30 100
```

Your Task:
Since this is a function problem. You don't have to take input. Just complete the function **topView()** that takes **root node** as parameter and returns a list of nodes visible from the top view from left to right.

Expected Time Complexity: O(N)
Expected Auxiliary Space: O(N).

Constraints:
$1 \le N \le 10^5$
$1 \le$ Node Data $\le 10^5$





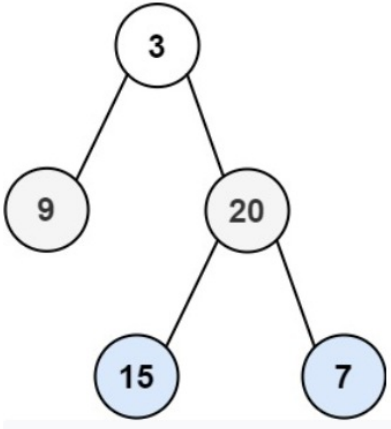

```java
class Solution
{
    //Function to return a list of nodes visible from the top view
    //from left to right in Binary Tree.
    static ArrayList<Integer> topView(Node root)
    {
        ArrayList<Integer> ans = new ArrayList<>();
        if(root == null) return ans;
        Map<Integer, Integer> map = new TreeMap<>();
        Queue<Pair> q = new LinkedList<Pair>();
        q.add(new Pair(root, 0));
        while(!q.isEmpty()) {
            Pair it = q.remove();
            int hd = it.hd;
            Node temp = it.node;
            if(map.get(hd) == null) map.put(hd, temp.data);
            if(temp.left != null) {

                q.add(new Pair(temp.left, hd - 1));
            }
            if(temp.right != null) {

                q.add(new Pair(temp.right, hd + 1));
            }
        }

        for (Map.Entry<Integer,Integer> entry : map.entrySet()) {
            ans.add(entry.getValue());
        }
        return ans;

    }
}
```

102. Binary Tree Level Order Traversal

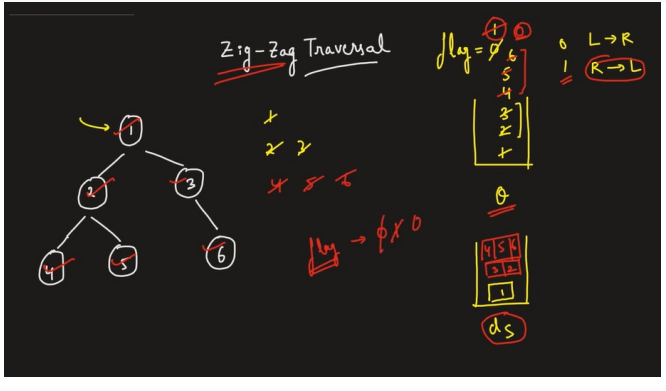
Medium 6367 127 Add to List Share

Given the `root` of a binary tree, return *the level order traversal of its nodes' values.* (i.e., from left to right, level by level).

Example 1:




```
Input: root = [3,9,20,null,null,15,7]
Output: [[3],[9,20],[15,7]]
```




```java
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode() {}
 *     TreeNode(int val) { this.val = val; }
 *     TreeNode(int val, TreeNode left, TreeNode right) {
 *         this.val = val;
 *         this.left = left;
 *         this.right = right;
 *     }
 * }
 */
class Solution {
    public List<List<Integer>> levelOrder(TreeNode root) {
        Queue<TreeNode> q = new LinkedList<>();
        List<List<Integer>> res = new ArrayList<List<Integer>>();
        if(root==null) return res;
        q.add(root);
        while(!q.isEmpty()){
            int n = q.size();
            List<Integer> tem = new ArrayList<Integer>();
            if(n==0) break;
            while(n-- > 0){
                TreeNode temp = q.poll();
                tem.add(temp.val);
                if(temp.left != null) q.add(temp.left);
                if(temp.right != null) q.add(temp.right);
            }
            res.add(tem);
        }
        return res;
    }
}
```

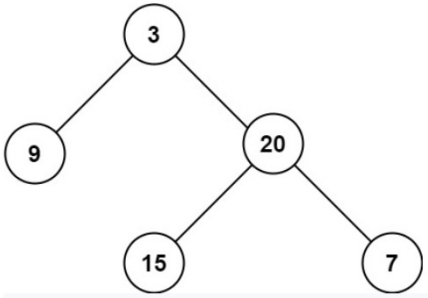
104. Maximum Depth of Binary Tree

Easy 5234 109 Add to List Share

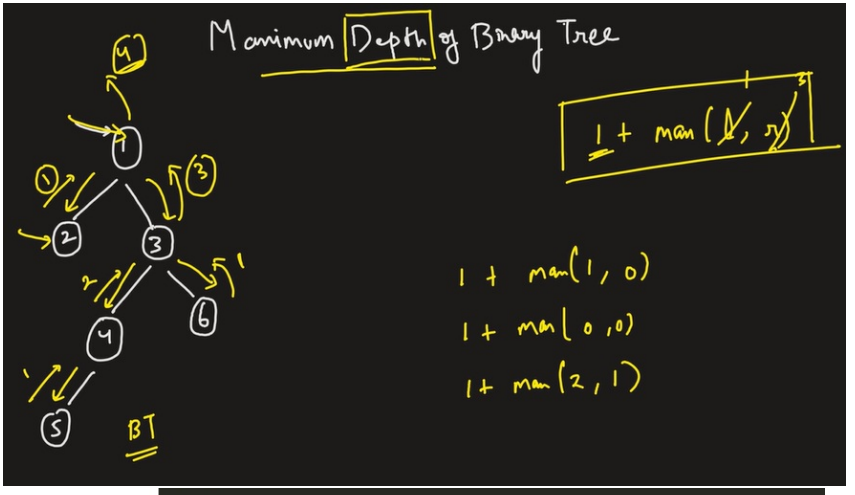
Given the `root` of a binary tree, return *its maximum depth*.

A binary tree's **maximum depth** is the number of nodes along the longest path from the root node down to the farthest leaf node.

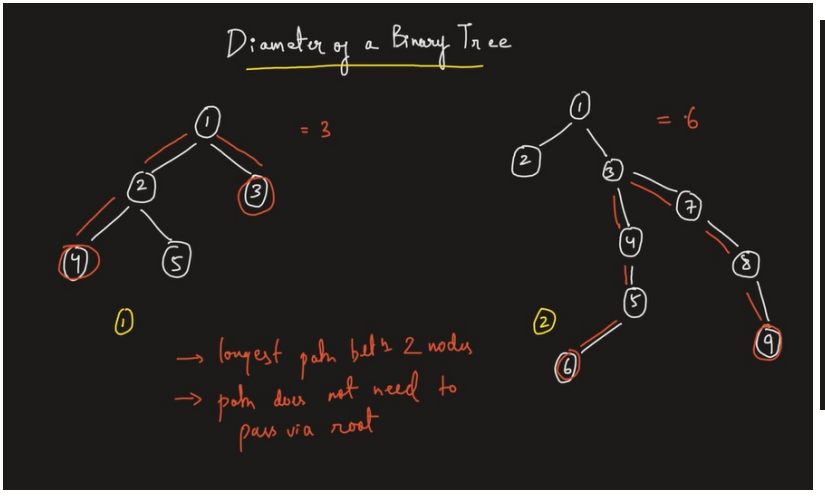
Example 1:

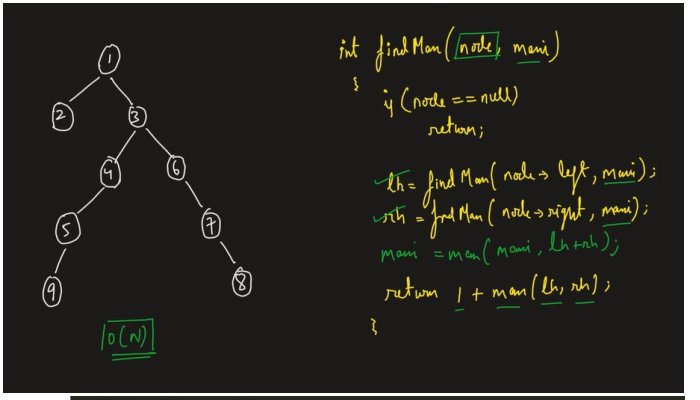



```
Input: root = [3,9,20,null,null,15,7]
Output: 3
```

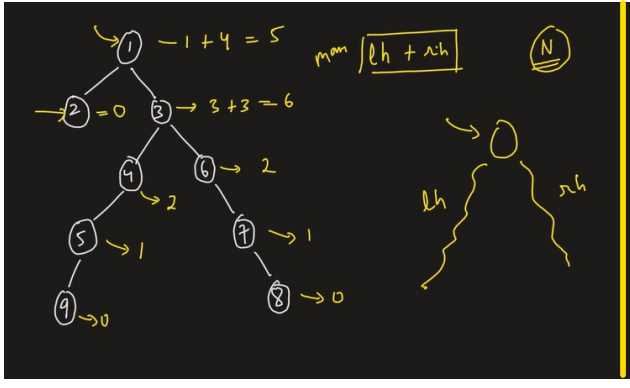


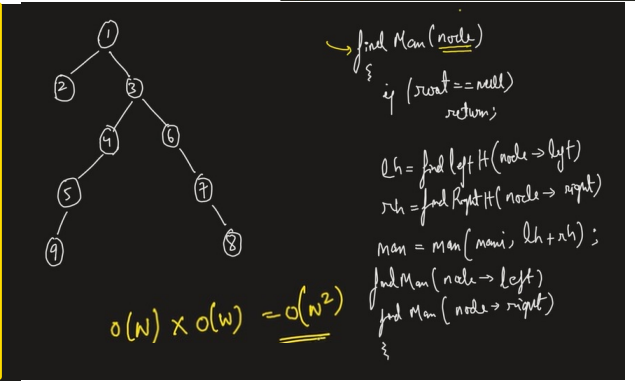

```java
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode() {}
 *     TreeNode(int val) { this.val = val; }
 *     TreeNode(int val, TreeNode left, TreeNode right) {
 *         this.val = val;
 *         this.left = left;
 *         this.right = right;
 *     }
 * }
 */
class Solution {
    public int maxDepth(TreeNode root) {

        if(root == null) return 0;

        int lh = maxDepth(root.left);
        int rh = maxDepth(root.right);

        return 1+Math.max(lh,rh);
    }
}
```



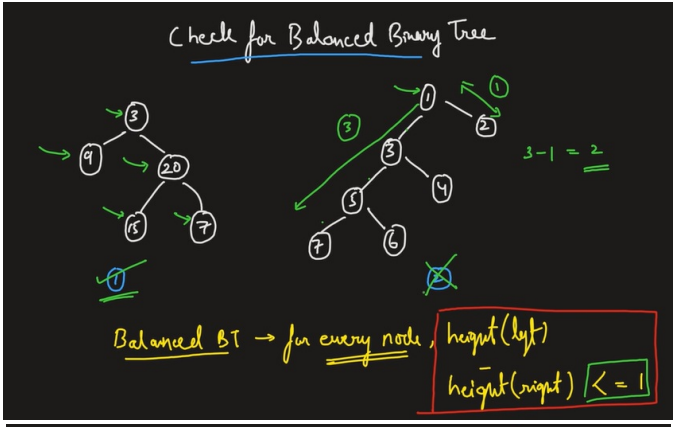


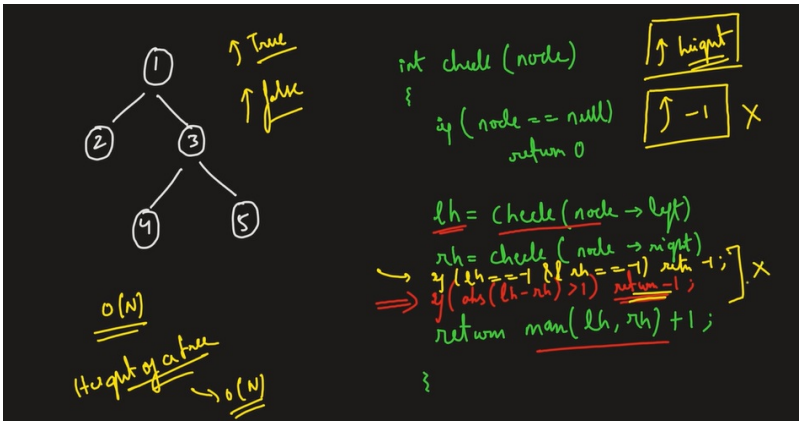

```java
public class Solution {
    public int diameterOfBinaryTree(TreeNode root) {
        int[] diameter = new int[1];
        height(root, diameter);
        return diameter[0];
    }

    private int height(TreeNode node, int[] diameter) {
        if (node == null) {
            return 0;
        }
        int lh = height(node.left, diameter);
        int rh = height(node.right, diameter);
        diameter[0] = Math.max(diameter[0], lh + rh);
        return 1 + Math.max(lh, rh);
    }
}
```

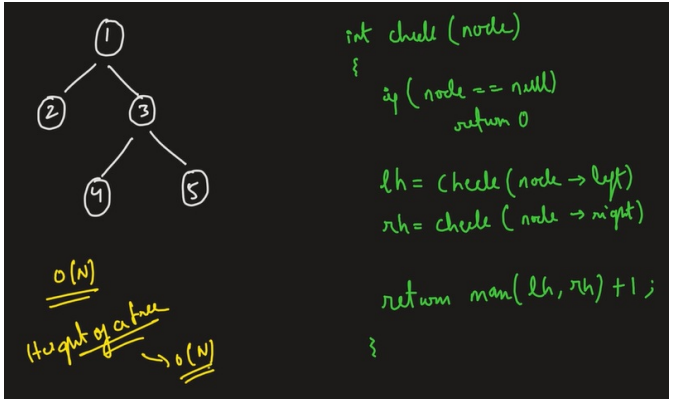




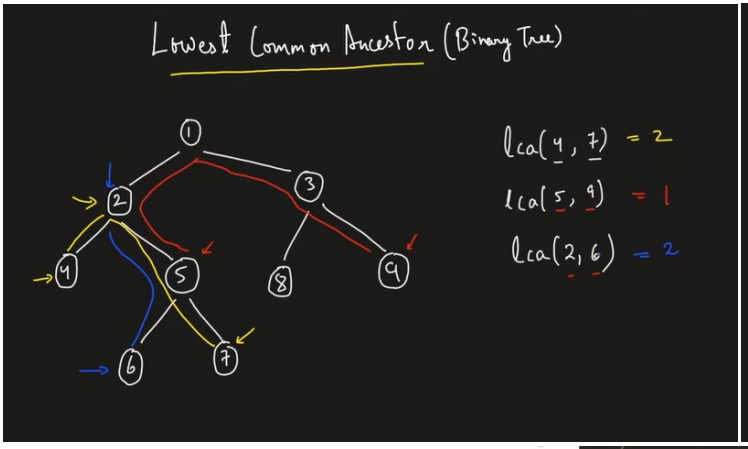
Check if the Binary tree is height-balanced or not

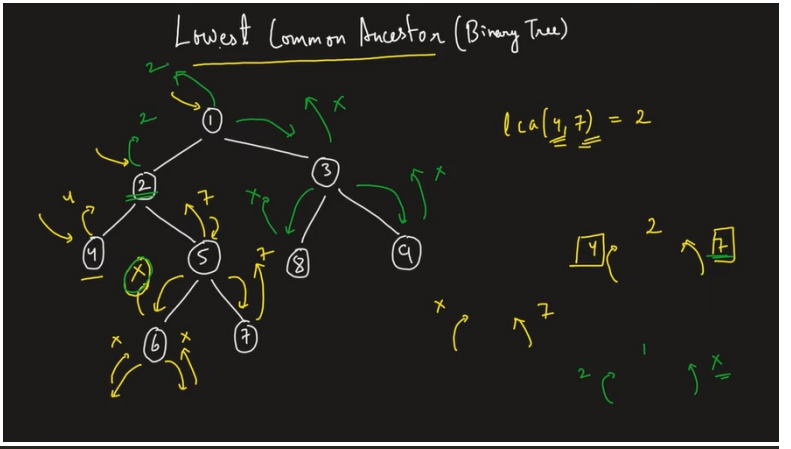







```java
class Solution {
    public boolean isBalanced(TreeNode root) {
        return dfsHeight (root) != -1;
    }
    int dfsHeight (TreeNode root) {
        if (root == null) return 0;

        int leftHeight = dfsHeight (root.left);
        if (leftHeight == -1) return -1;
        int rightHeight = dfsHeight (root.right);
        if (rightHeight == -1) return -1;

        if (Math.abs(leftHeight - rightHeight) > 1)  return -1;
        return Math.max(leftHeight, rightHeight) + 1;
    }
}
```






```java
class Solution {
    public TreeNode lowestCommonAncestor(TreeNode root, TreeNode p, TreeNode q) {
        if(root==null || root==p || root==q) return root;
        TreeNode left = lowestCommonAncestor(root.left,p,q);
        TreeNode right = lowestCommonAncestor(root.right,p,q);
        if(left == null) return right;
        else if(right == null) return left;
        else return root;
    }
}
```

100. Same Tree

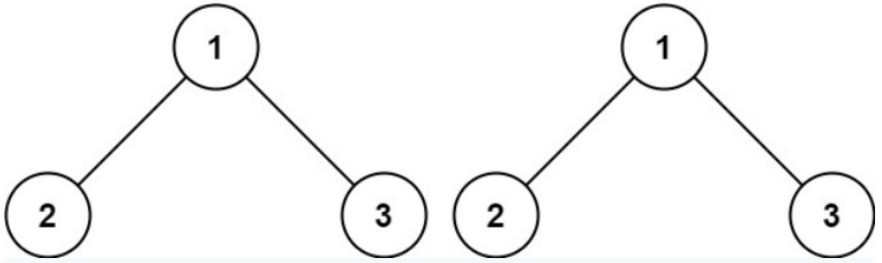
Check if two trees are identical or not

Easy 4372 110 Add to List Share

Given the roots of two binary trees `p` and `q`, write a function to check if they are the same or not.

Two binary trees are considered the same if they are structurally identical, and the nodes have the same value.

Example 1:




```
Input: p = [1,2,3], q = [1,2,3]
Output: true
```

```java
class Solution {
    public boolean isSameTree(TreeNode p, TreeNode q) {
        if(p==null || q==null){
            return (p==q);
        }
        return (p.val == q.val) && isSameTree(p.left,q.left) && isSameTree(p.right,q.right);
    }
}
```

124. Binary Tree Maximum Path Sum

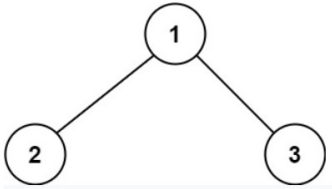
Hard 7656 467 Add to List Share

A **path** in a binary tree is a sequence of nodes where each pair of adjacent nodes in the sequence has an edge connecting them. A node can only appear in the sequence **at most once**. Note that the path does not need to pass through the root.

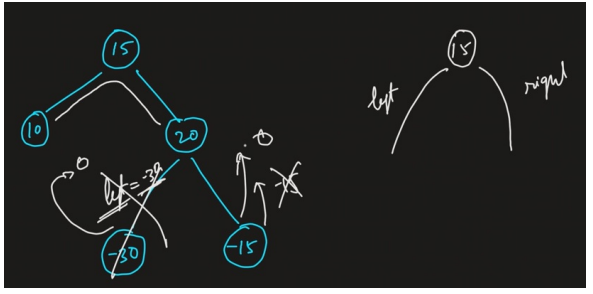
The **path sum** of a path is the sum of the node's values in the path.

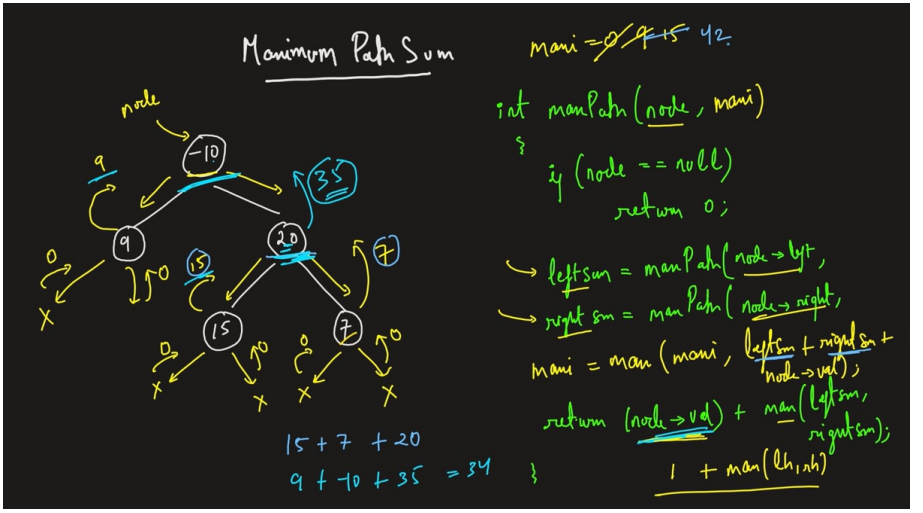
Given the `root` of a binary tree, return *the maximum **path sum** of any **non-empty** path.*

Example 1:




```
Input: root = [1,2,3]
Output: 6
Explanation: The optimal path is 2 -> 1 -> 3 with a path sum of 2 + 1 + 3 = 6.
```





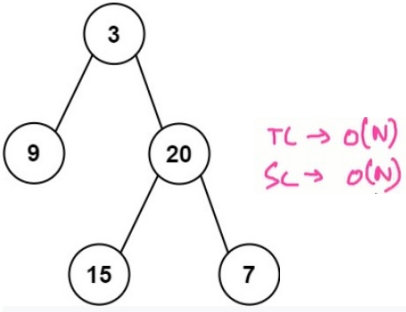

```java
class Solution {
    public int maxPathSum(TreeNode root) {
        int maxVal[] = new int[1];
        maxVal[0] = Integer.MIN_VALUE;
        maxPathDown(root,maxVal);
        return maxVal[0];
    }
    private int maxPathDown(TreeNode node, int maxVal[]){
        if(node==null) return 0;
        int left = Math.max(0,maxPathDown(node.left,maxVal));
        int right = Math.max(0,maxPathDown(node.right,maxVal));
        maxVal[0] = Math.max(maxVal[0],left+right+node.val);
        return Math.max(left,right) + node.val;
    }
}
```

105. Construct Binary Tree from Preorder and Inorder Traversal

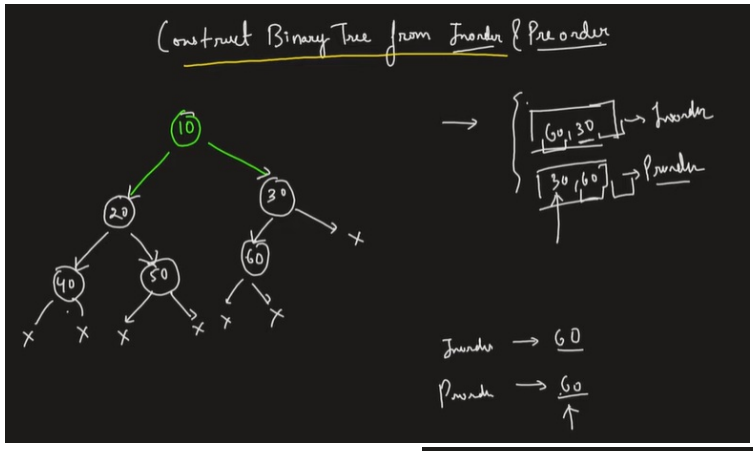
Medium 6928 174 Add to List Share

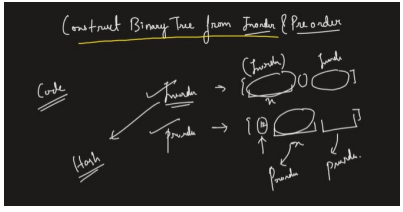
Given two integer arrays `preorder` and `inorder` where `preorder` is the preorder traversal of a binary tree and `inorder` is the inorder traversal of the same tree, construct and return *the binary tree*.

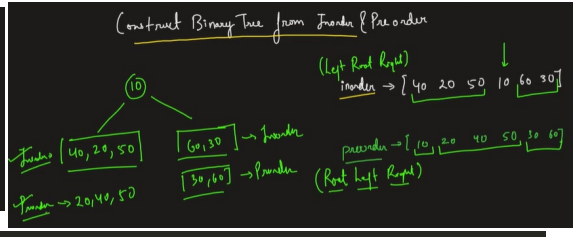
Example 1:

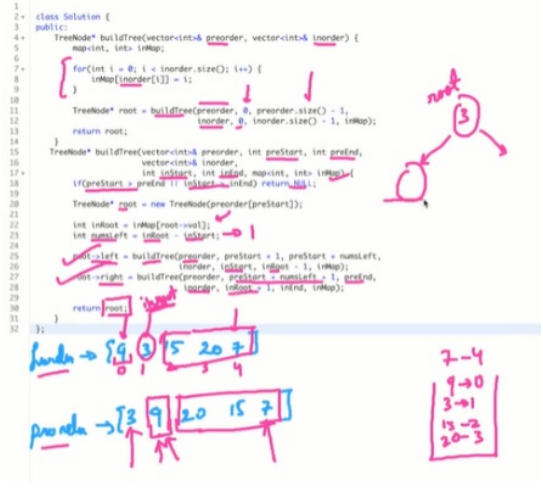


Input: preorder = [3,9,20,15,7], inorder = [9,3,15,20,7]
Output: [3,9,20,null,null,15,7]

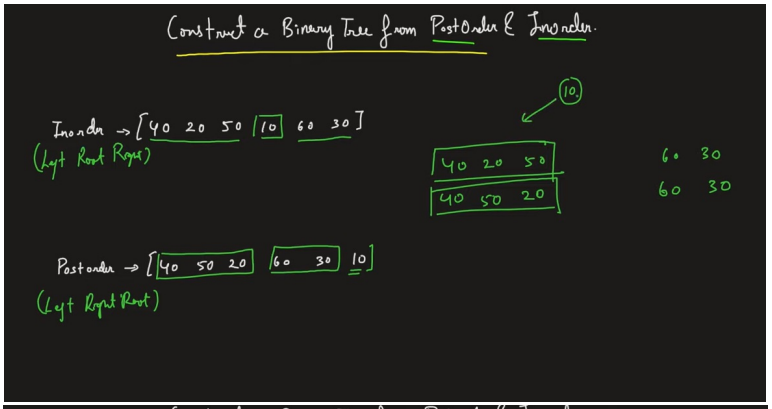


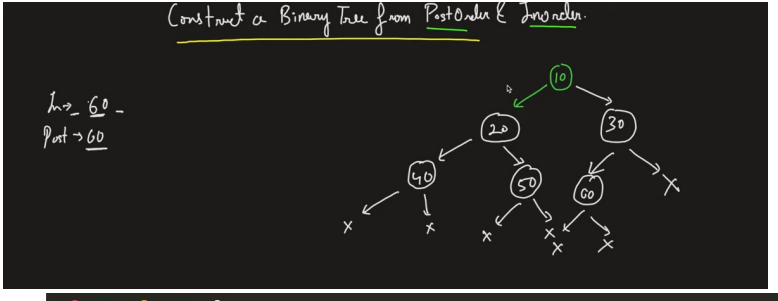


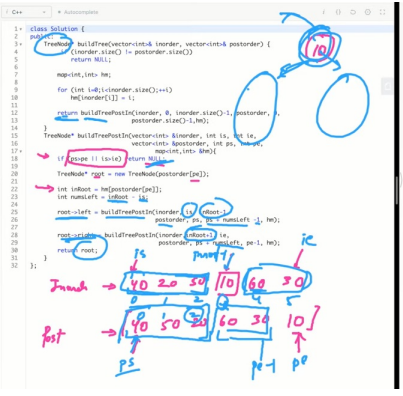


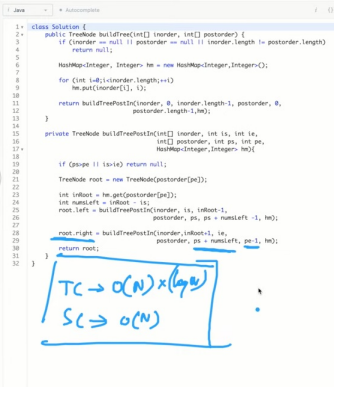



```java
class Solution {
    public TreeNode buildTree(int[] preorder, int[] inorder) {
        Map<Integer,Integer> inMap = new HashMap<Integer,Integer>();
        for(int i=0; i<inorder.length; i++){
            inMap.put(inorder[i],i);
        }
        TreeNode root = buildTree(preorder, 0, preorder.length-1, inorder, 0, inorder.length-1, inMap);
        return root;
    }
    public TreeNode buildTree(int[] preorder, int preStart, int preEnd, int[] inorder, int inStart, int inEnd, Map<Integer,Integer> inMap){
        if(preStart>preEnd || inStart>inEnd) return null;
        TreeNode root = new TreeNode(preorder[preStart]);
        int inRoot = inMap.get(root.val);
        int numsLeft = inRoot-inStart;

        root.left = buildTree(preorder, preStart+1, preStart+numsLeft, inorder, inStart, inRoot-1, inMap);
        root.right = buildTree(preorder, preStart+numsLeft+1, preEnd, inorder, inRoot+1, inEnd, inMap);

        return root;
    }
}
```









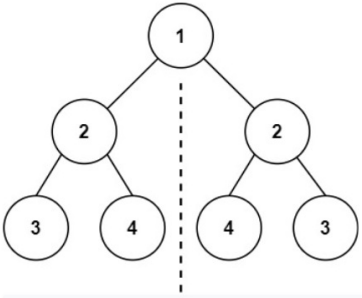

```java
class Solution {
    public TreeNode buildTree(int[] inorder, int[] postorder) {
        if(inorder==null || postorder==null || inorder.length!=postorder.length) return null;
        HashMap<Integer,Integer> hm = new HashMap<Integer,Integer>();
        for(int i=0; i<inorder.length; ++i){
            hm.put(inorder[i],i);
        }
        return buildTreePostIn(inorder, 0, inorder.length-1, postorder, 0, postorder.length-1, hm);
    }
    private TreeNode buildTreePostIn(int[] inorder, int is, int ie, int[] postorder, int ps, int pe, HashMap<Integer,Integer> hm){
        if(ps>pe || is>ie) return null;
        TreeNode root = new TreeNode(postorder[pe]);

        int inRoot = hm.get(postorder[pe]);
        int numsLeft = inRoot - is;

        root.left = buildTreePostIn(inorder, is, inRoot-1, postorder, ps, ps+numsLeft-1, hm);
        root.right = buildTreePostIn(inorder, inRoot+1, ie, postorder, ps+numsLeft, pe-1, hm);

        return root;
    }
}
```

101. Symmetric Tree

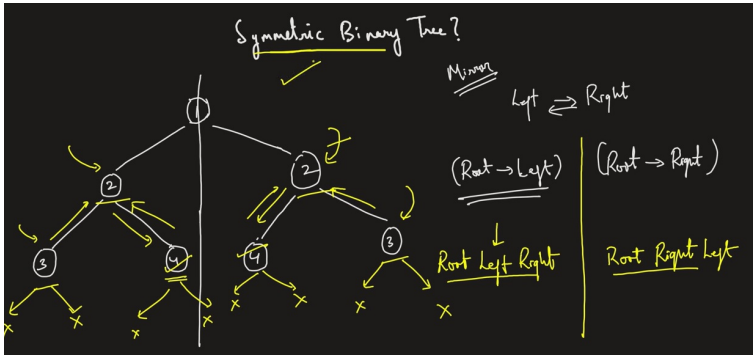
Easy 7823 193 Add to List Share

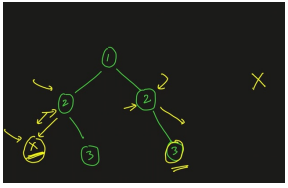
Given the `root` of a binary tree, *check whether it is a mirror of itself* (i.e., symmetric around its center).

Example 1:




```
Input: root = [1,2,2,3,4,4,3]
Output: true
```





```cpp
 * };
 */
class Solution {
public:
    bool isSymmetric(TreeNode* root) {
        return root==NULL || isSymmetricHelp(root->left, root->right);
    }
    bool isSymmetricHelp(TreeNode* left, TreeNode* right){
        if(left==null || right==null)
            return left==right;

        if(left.val!=right.val) return false;

        return isSymmetricHelp(left.left, right.right)
               &&
               isSymmetricHelp(left.right, right.left);
    }

};
```

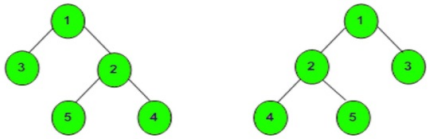
TC → O(N)
SC → O(N)

```cpp
 *         this.val = val;
 *         this.left = left;
 *         this.right = right;
 *     }
 * }
 */
class Solution {
    public boolean isSymmetric(TreeNode root) {
        return root==null || isSymmetricHelp(root.left, root.right);
    }

    private boolean isSymmetricHelp(TreeNode left, TreeNode right){
        if(left==null || right==null)
            return left==right;

        if(left.val!=right.val) return false;

        return isSymmetricHelp(left.left, right.right)
               &&
               isSymmetricHelp(left.right, right.left);
    }

}
```

Mirror Tree

Easy Accuracy: 63.43% Submissions: 100k+ Points: 2

Given a Binary Tree, convert it into its mirror.



Mirror Trees

Example 1:

```
Input:
      1
    /  \
   2    3
Output: 2 1 3
Explanation: The tree is
   1    (mirror)  1
 /  \    =>      /  \
3    2          2    3
The inorder of mirror is 2 1 3
```

```
139 void mirror(Node* root)
140 {
141       // Your Code Here
142     if(!root) return;
143     mirror(root->left);
144     mirror(root->right);
145     swap(root->left,root->right);
146 }
```

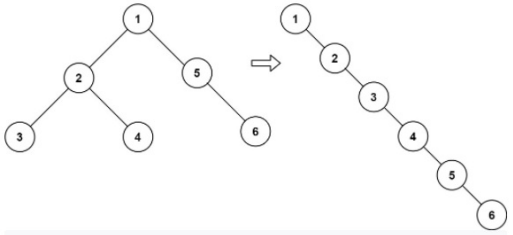
114. Flatten Binary Tree to Linked List

Medium 5862 433 Add to List Share

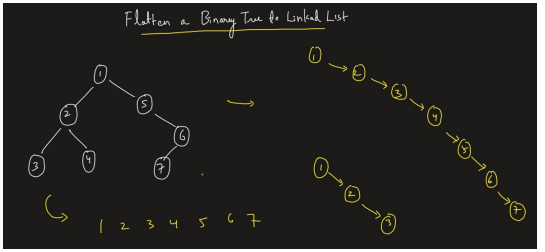
Given the `root` of a binary tree, flatten the tree into a "linked list":

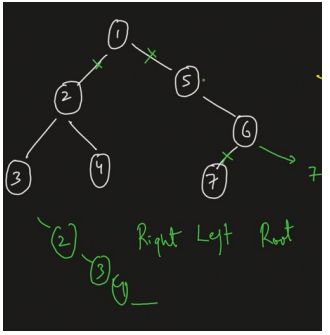
- The "linked list" should use the same `TreeNode` class where the `right` child pointer points to the next node in the list and the `left` child pointer is always `null`.
- The "linked list" should be in the same order as a **pre-order traversal** of the binary tree.

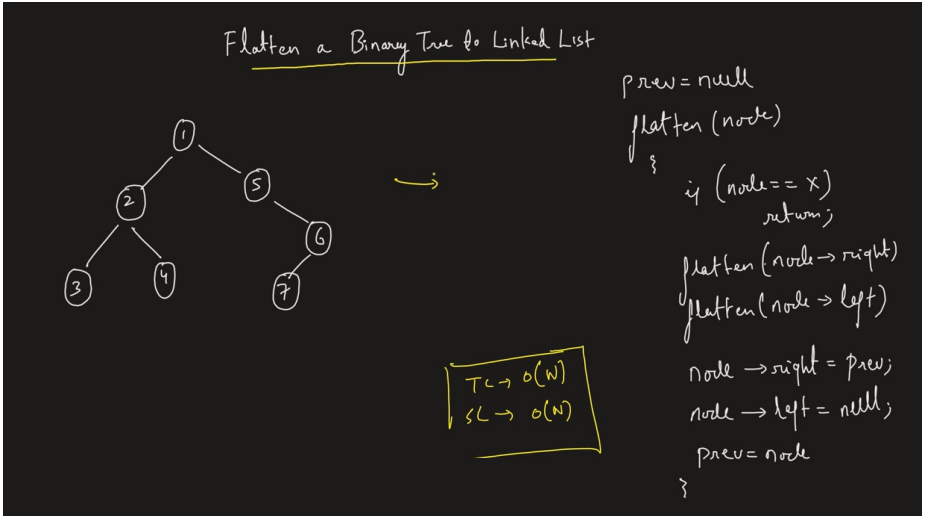
Example 1:



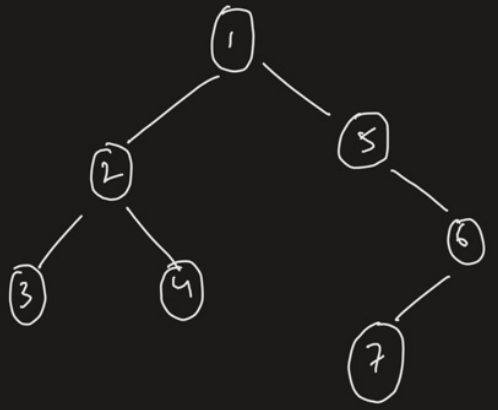
```
Input: root = [1,2,5,3,4,null,6]
Output: [1,null,2,null,3,null,4,null,5,null,6]
```



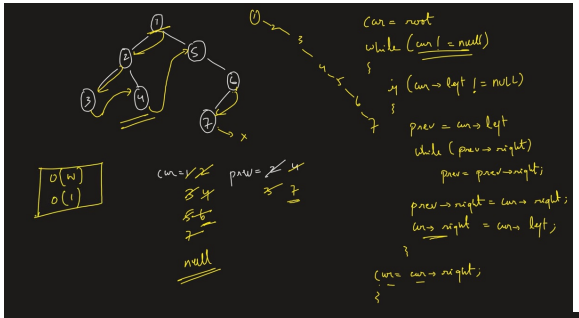


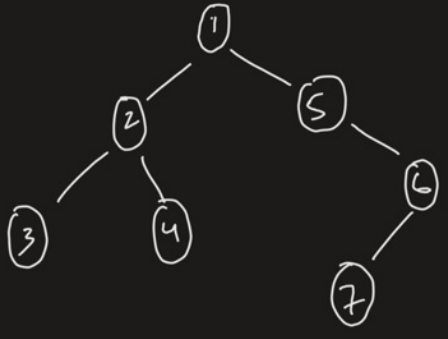


Flatten a Binary Tree to Linked List




```
st.push(root)
while(!st.empty())
{
    cur = st.top() ; st.pop();

    if(cur->right)
        st.push(cur->right)
    if(cur->left)
        st.push(cur->left)

    if(!st.empty())
        cur->right = st.top();
    cur->left = null;
}
```






```
cur = root
while (cur != null)
{
    if (cur → left != NULL)
    {
        prev = cur → left
        while (prev → right)
            prev = prev → right;

        prev → right = cur → right;
        cur → right = cur → left;
    }
cur = cur → right;
}
```

Day-20: Binary Search Tree

116. Populating Next Right Pointers in Each Node

Medium 4772 203 Add to List Share

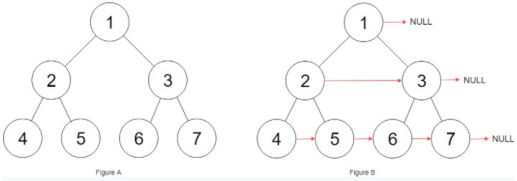
You are given a **perfect binary tree** where all leaves are on the same level, and every parent has two children. The binary tree has the following definition:

```
struct Node {
  int val;
  Node *left;
  Node *right;
  Node *next;
}
```

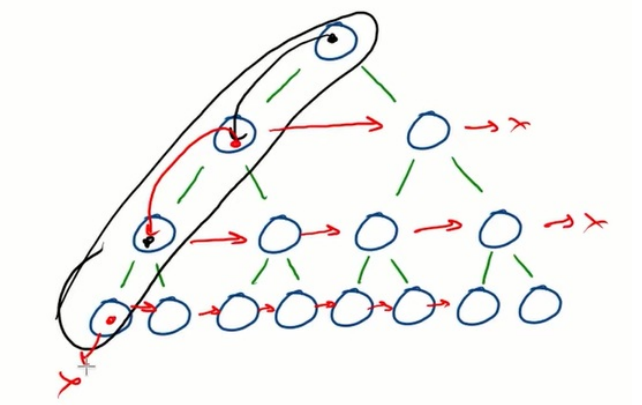
Populate each next pointer to point to its next right node. If there is no next right node, the next pointer should be set to `NULL`.

Initially, all next pointers are set to `NULL`.

Example 1:




```
Input: root = [1,2,3,4,5,6,7]
Output: [1,#,2,3,#,4,5,6,7,#]
Explanation: Given the above perfect binary tree (Figure A),
your function should populate each next pointer to point to
its next right node, just like in Figure B. The serialized
output is in level order as connected by the next pointers,
with '#' signifying the end of each level.
```



```java
class Solution {
  public Node connect(Node root) {
      Node black = root;
      while(black!=null && black.left!=null){
          Node n = black;
          while(true){
              n.left.next = n.right;
              if(n.next==null) break;
              n.right.next = n.next.left;
              n = n.next;
          }
          black = black.left;
      }
      return root;
  }
}
```

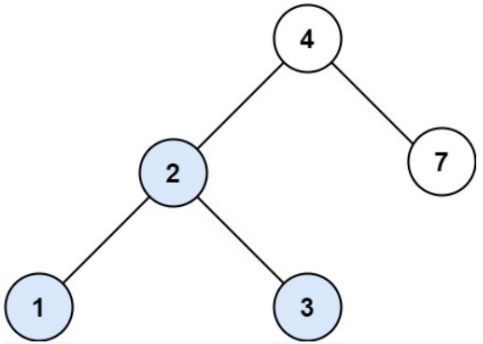
700. Search in a Binary Search Tree

Easy 2062 137 Add to List Share

You are given the `root` of a binary search tree (BST) and an integer `val`.

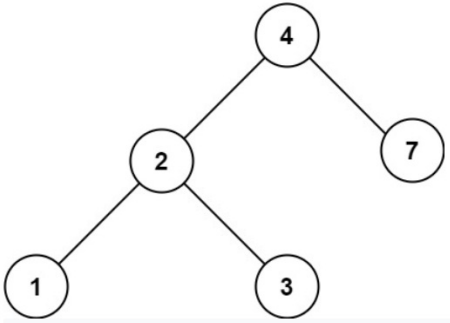
Find the node in the BST that the node's value equals `val` and return the subtree rooted with that node. If such a node does not exist, return `null`.

Example 1:




```
Input: root = [4,2,7,1,3], val = 2
Output: [2,1,3]
```

Example 2:




```
Input: root = [4,2,7,1,3], val = 5
Output: []
```

```java
class Solution {
    public TreeNode searchBST(TreeNode root, int val) {
        TreeNode curr = root;
        while(curr!=null){
            if(curr.val==val){
                return curr;
            }else if(curr.val>val){
                curr = curr.left;
            }else{
                curr = curr.right;
            }
        }
        return curr;
    }
}
```

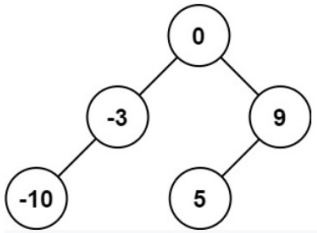
108. Convert Sorted Array to Binary Search Tree

Easy 5216 320 Add to List Share

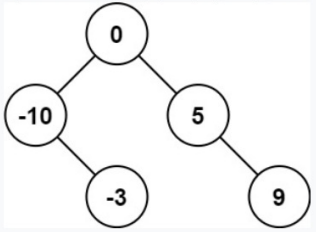
Given an integer array `nums` where the elements are sorted in **ascending order**, convert *it to a **height-balanced** binary search tree.*

A **height-balanced** binary tree is a binary tree in which the depth of the two subtrees of every node never differs by more than one.

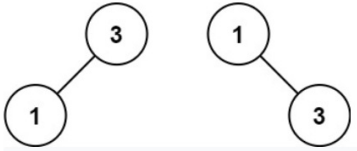
Example 1:



Input: nums = [-10,-3,0,5,9]
Output: [0,-3,9,-10,null,5]
Explanation: [0,-10,5,null,-3,null,9] is also accepted:



Example 2:



Input: nums = [1,3]
Output: [3,1]
Explanation: [1,3] and [3,1] are both a height-balanced BSTs.

```java
class Solution {
    public TreeNode sortedArrayToBST(int[] nums) {
        return createBST(nums, 0, nums.length - 1);
    }

    public TreeNode createBST(int[] nums, int start, int end) {
        if(start > end){
            return null;
        }
        int mid = start + (end - start) / 2;
        TreeNode root = new TreeNode(nums[mid]);
        root.left = createBST(nums, start, mid - 1);
        root.right = createBST(nums, mid + 1, end);
        return root;
    }
}
```

98. Validate Binary Search Tree

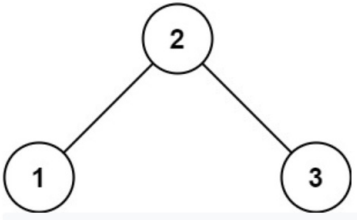
Medium 7931 783 Add to List Share

Given the `root` of a binary tree, *determine if it is a valid binary search tree (BST).*

A **valid BST** is defined as follows:

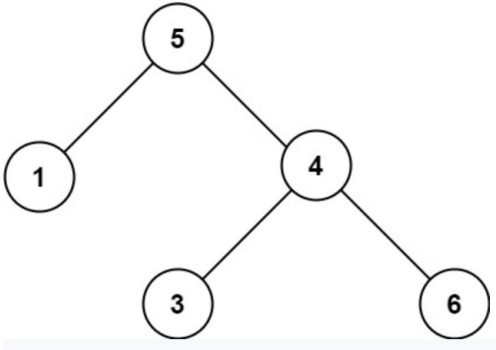
- The left subtree of a node contains only nodes with keys **less than** the node's key.
- The right subtree of a node contains only nodes with keys **greater than** the node's key.
- Both the left and right subtrees must also be binary search trees.

Example 1:

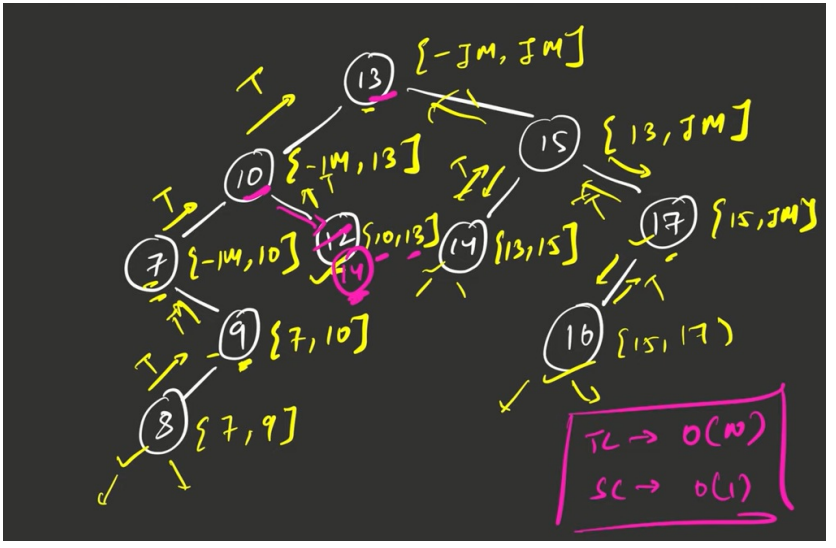



```
Input: root = [2,1,3]
Output: true
```

Example 2:




```
Input: root = [5,1,4,null,null,3,6]
Output: false
Explanation: The root node's value is 5 but its right child's
value is 4.
```




```java
class Solution {
    public boolean isValidBST(TreeNode root) {
        return isValidBSTHelper(root,Long.MIN_VALUE,Long.MAX_VALUE);
    }
    public boolean isValidBSTHelper(TreeNode root, long minVal, long maxVal){
        if(root==null) return true;
        if(root.val>=maxVal || root.val<=minVal) return false;
        return isValidBSTHelper(root.left,minVal,root.val) && isValidBSTHelper(root.right,root.val,maxVal);
    }
}
```

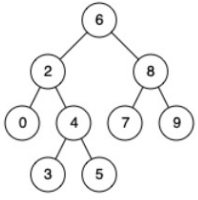
235. Lowest Common Ancestor of a Binary Search Tree

Easy 4377 160 Add to List Share

Given a binary search tree (BST), find the lowest common ancestor (LCA) of two given nodes in the BST.

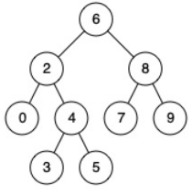
According to the definition of LCA on Wikipedia: "The lowest common ancestor is defined between two nodes `p` and `q` as the lowest node in `T` that has both `p` and `q` as descendants (where we allow **a node to be a descendant of itself**)."

Example 1:




```
Input: root = [6,2,8,0,4,7,9,null,null,3,5], p = 2, q = 8
Output: 6
Explanation: The LCA of nodes 2 and 8 is 6.
```

Example 2:




```
Input: root = [6,2,8,0,4,7,9,null,null,3,5], p = 2, q = 4
Output: 2
Explanation: The LCA of nodes 2 and 4 is 2, since a node can
be a descendant of itself according to the LCA definition.
```

Example 3:

```
Input: root = [2,1], p = 2, q = 1
Output: 2
```

```java
class Solution {
    public TreeNode lowestCommonAncestor(TreeNode root, TreeNode p, TreeNode q) {
        if(root == null) return null;
        int curr = root.val;
        if(curr<p.val && curr<q.val){
            return lowestCommonAncestor(root.right, p, q);
        }
        if(curr>p.val && curr>q.val){
            return lowestCommonAncestor(root.left, p, q);
        }
        return root;
    }
}
```

Predecessor and Successor

Easy Accuracy: 46.01% Submissions: 35

Find the inorder predecessor/successor of a given Key in BST.

There is BST given with root node with key part as integer only. You need to find the inorder successor and predecessor of a given key. In case, if the either of predecessor or successor is not found print -1.

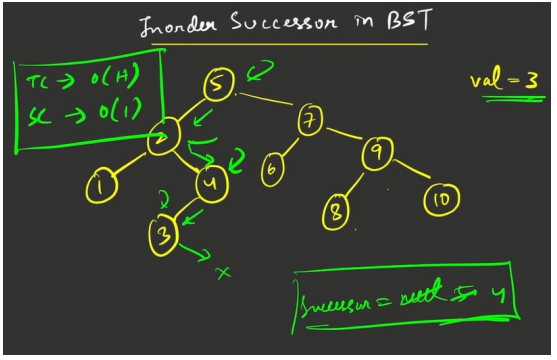
Input:
The first line of input contains an integer T denoting the number of test cases. Then T test cases follow. Each test case contains n denoting the number of edges of the BST. The next line contains the edges of the BST. The last line contains the key.

Output:
Print the predecessor followed by successor for the given key. If the predecessor or successor is not found print -1.

Constraints:
1<=T<=100
1<=n<=100
1<=data of node<=100
1<=key<=100

Example:
Input:
2
6
50 30 L 30 20 L 30 40 R 50 70 R 70 60 L 70 80 R
65
6
50 30 L 30 20 L 30 40 R 50 70 R 70 60 L 70 80 R
100

Output:
60 70
80 -1

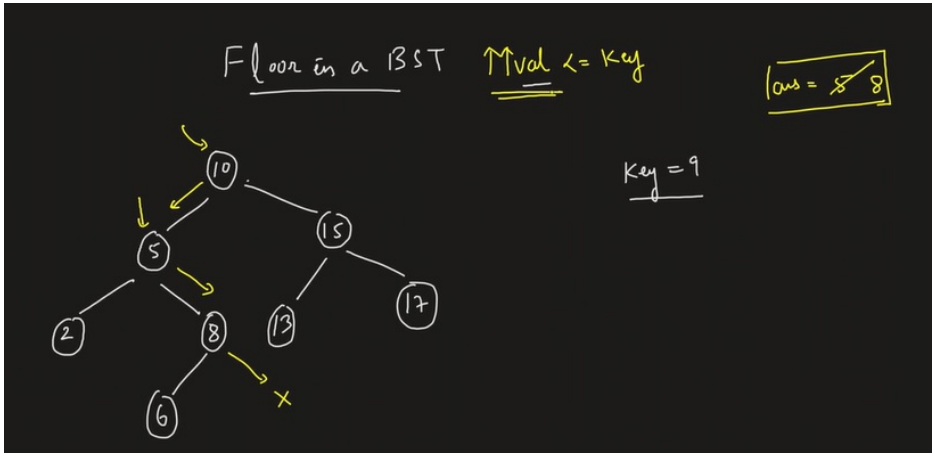



```java
class Solution {
    public TreeNode inorderSuccessor(TreeNode root, TreeNode p) {
        TreeNode successor = null;
        while (root != null) {
            if (p.val >= root.val) {
                root = root.right;
            } else {
                successor = root;
                root = root.left;
            }
        }
        return successor;
    }
}
```

```java
class Solution {
    public TreeNode inorderPredecessor(TreeNode root, TreeNode p) {
        TreeNode predecessor = null;
        while (root != null) {
            if (p.val < root.val) {
                root = root.left;
            } else {
                predecessor = root;
                root = root.right;
            }
        }
        return predecessor;
    }
}
```

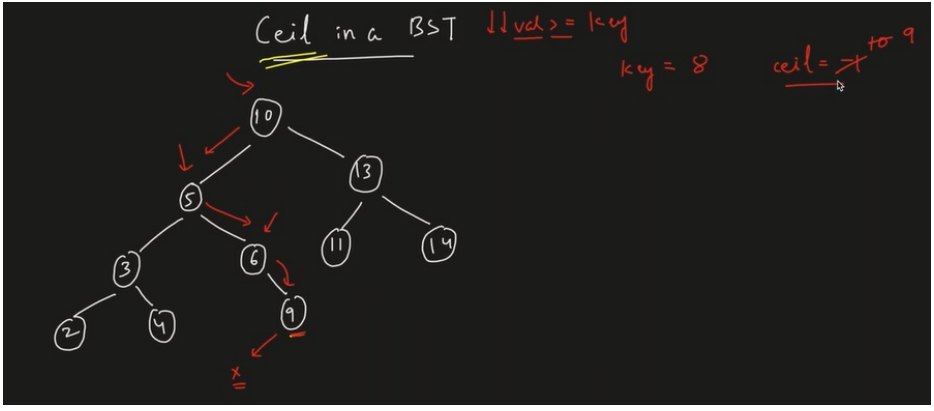
Day21: (BinarySearchTree)

Floor in a Binary Search Tree

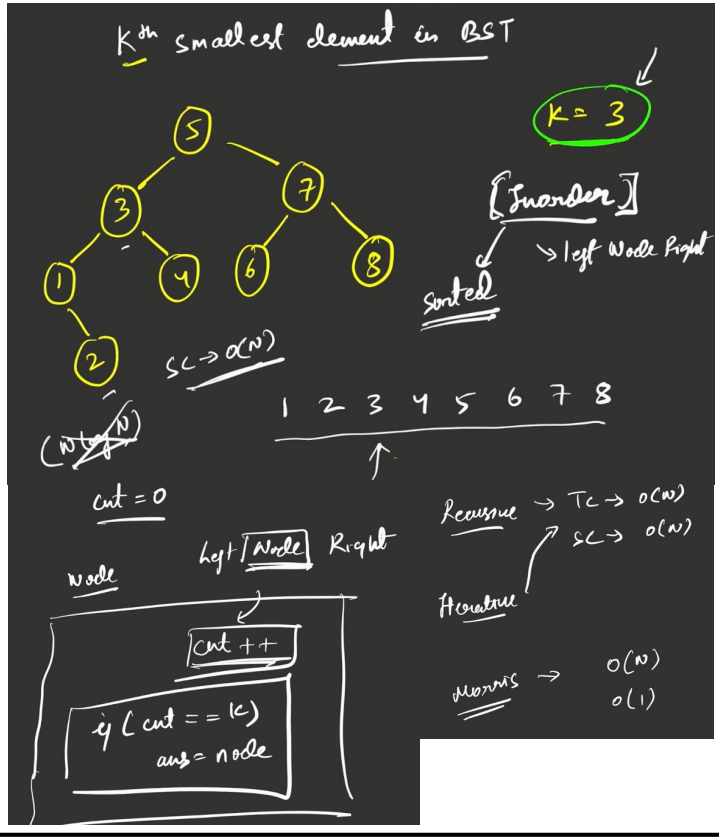



```java
public class Solution {

    public static int floorInBST(TreeNode<Integer> root, int key) {
        int floor = -1;
        while (root != null) {
            if (root.data == key) {
                floor = root.data;
                return floor;
            }

            if (key > root.data) {
                floor = root.data;
                root = root.right;
            }
            else {
                root = root.left;
            }
        }
        return floor;
    }
}
```

Ceil in a Binary Search Tree



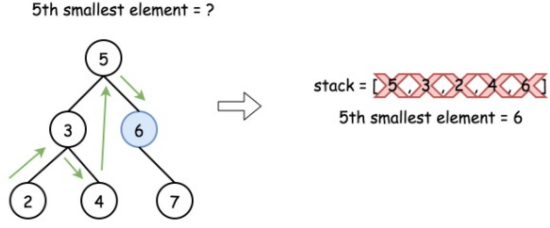

```java
public class Solution {

    public  static int findCeil(TreeNode<Integer> root, int key) {

        int ceil = -1;
        while (root != null) {
            if (root.data == key) {
                ceil = root.data;
                return ceil;
            }

            if (key > root.data) {
                root = root.right;
            }
            else {
                ceil = root.data;
                root = root.left;
            }
        }
        return ceil;

    }
}
```



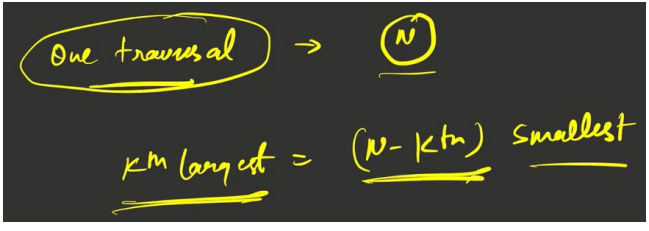

```java
class Solution {
  public int kthSmallest(TreeNode root, int k) {
    LinkedList<TreeNode> stack = new LinkedList<TreeNode>();
    while (true) {
      while (root != null) {
        stack.add(root);
        root = root.left;
      }
      root = stack.removeLast();
      if (--k == 0) return root.val;
      root = root.right;
    }
  }
}
```

Approach 2: Iterative Inorder Traversal

The above recursion could be converted into iteration, with the help of stack. This way one could speed up the solution because there is no need to build the entire inorder traversal, and one could stop after the kth element.



Kth largest element in BST



Binary Search Tree Iterator

173. Binary Search Tree Iterator

Medium 4421 340 Add to List Share

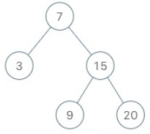
Implement the `BSTIterator` class that represents an iterator over the **in-order traversal** of a binary search tree (BST):

- `BSTIterator(TreeNode root)` Initializes an object of the `BSTIterator` class. The `root` of the BST is given as part of the constructor. The pointer should be initialized to a non-existent number smaller than any element in the BST.
- `boolean hasNext()` Returns `true` if there exists a number in the traversal to the right of the pointer, otherwise returns `false`.
- `int next()` Moves the pointer to the right, then returns the number at the pointer.

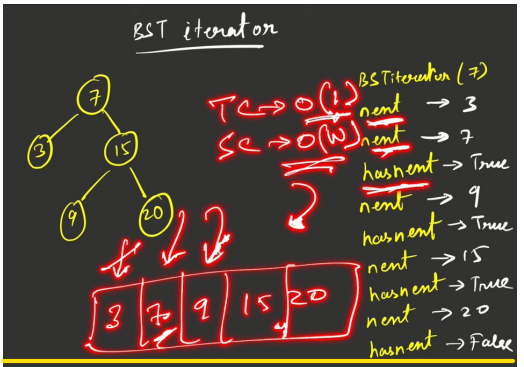
Notice that by initializing the pointer to a non-existent smallest number, the first call to `next()` will return the smallest element in the BST.

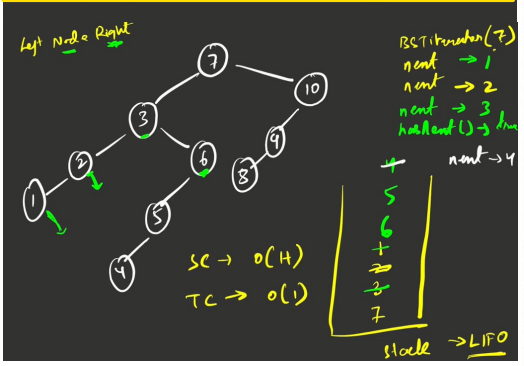
You may assume that `next()` calls will always be valid. That is, there will be at least a next number in the in-order traversal when `next()` is called.

Example 1:




```
Input
["BSTIterator", "next", "next", "hasNext", "next", "hasNext",
"next", "hasNext", "next", "hasNext"]
[[[7, 3, 15, null, null, 9, 20]], [], [], [], [], [], [],
[], []]
Output
[null, 3, 7, true, 9, true, 15, true, 20, false]

Explanation
BSTIterator bSTIterator = new BSTIterator([7, 3, 15, null,
null, 9, 20]);
bSTIterator.next();    // return 3
bSTIterator.next();    // return 7
bSTIterator.hasNext(); // return True
bSTIterator.next();    // return 9
bSTIterator.hasNext(); // return True
bSTIterator.next();    // return 15
bSTIterator.hasNext(); // return True
bSTIterator.next();    // return 20
bSTIterator.hasNext(); // return False
```






```java
public class BSTIterator {
    private Stack<TreeNode> stack = new Stack<TreeNode>();

    public BSTIterator(TreeNode root) {
        pushAll(root);
    }

    /** @return whether we have a next smallest number */
    public boolean hasNext() {
        return !stack.isEmpty();
    }

    /** @return the next smallest number */
    public int next() {
        TreeNode tmpNode = stack.pop();
        pushAll(tmpNode.right);
        return tmpNode.val;
    }

    private void pushAll(TreeNode node) {
        for (; node != null; stack.push(node), node = node.left);
    }
}

/**
 * Your BSTIterator object will be instantiated and called as such:
 * BSTIterator obj = new BSTIterator(root);
 * int param_1 = obj.next();
 * boolean param_2 = obj.hasNext();
 */
```

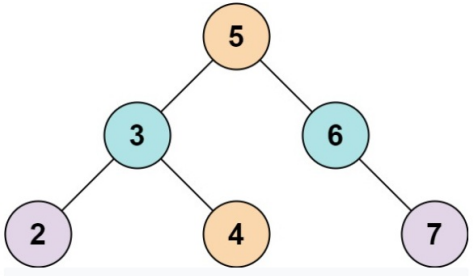
Two Sum In BST | Check if there exists a pair with Sum K

653. Two Sum IV - Input is a BST

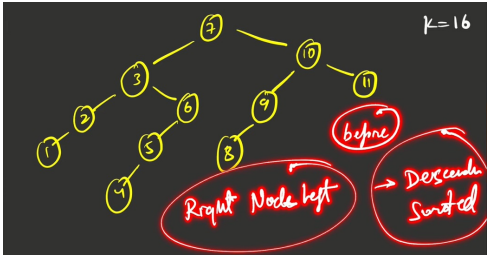
Easy 2895 182 Add to List Share

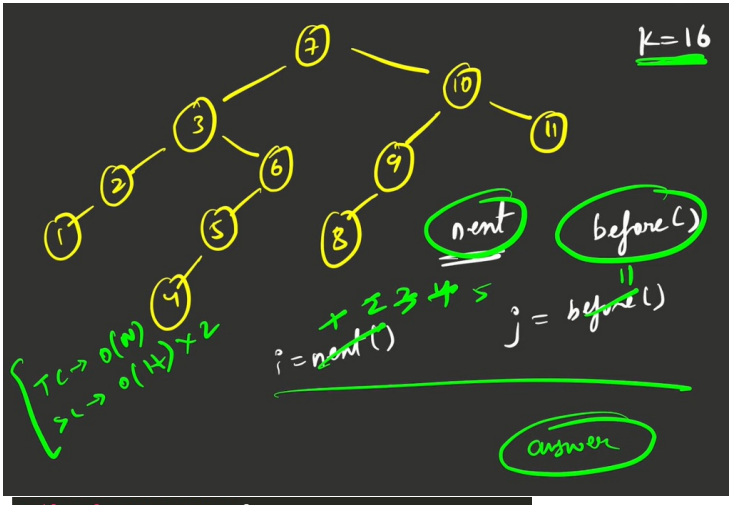
Given the `root` of a Binary Search Tree and a target number `k`, return *`true`* *if there exist two elements in the BST such that their sum is equal to the given target.*

Example 1:




```
Input: root = [5,3,6,2,4,null,7], k = 9
Output: true
```






```java
public class BSTIterator {
    private Stack<TreeNode> stack = new Stack<TreeNode>();
    boolean reverse = true;

    public BSTIterator(TreeNode root, boolean isReverse) {
        reverse = isReverse;
        pushAll(root);
    }

    /** @return whether we have a next smallest number */
    public boolean hasNext() {
        return !stack.isEmpty();
    }

    /** @return the next smallest number */
    public int next() {
        TreeNode tmpNode = stack.pop();
        if(reverse == false) pushAll(tmpNode.right);
        else pushAll(tmpNode.left);
        return tmpNode.val;
    }

    private void pushAll(TreeNode node) {
        while(node != null) {
            stack.push(node);
            if(reverse == true) {
                node = node.right;
            } else {
                node = node.left;
            }
        }
    }
}
```

```java
class Solution {
    public boolean findTarget(TreeNode root, int k) {
        if(root == null) return false;
        BSTIterator l = new BSTIterator(root, false);
        BSTIterator r = new BSTIterator(root, true);

        int i = l.next();
        int j = r.next();
        while(i<j) {
            if(i + j == k) return true;
            else if(i + j < k) i = l.next();
            else j = r.next();
        }
        return false;
    }
}
```

1373. Maximum Sum BST in Binary Tree

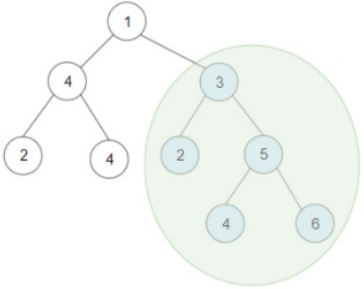
Hard 774 103 Add to List Share

Given a **binary tree** `root`, return *the maximum sum of all keys of* ***any*** *sub-tree which is also a Binary Search Tree (BST).*

Assume a BST is defined as follows:

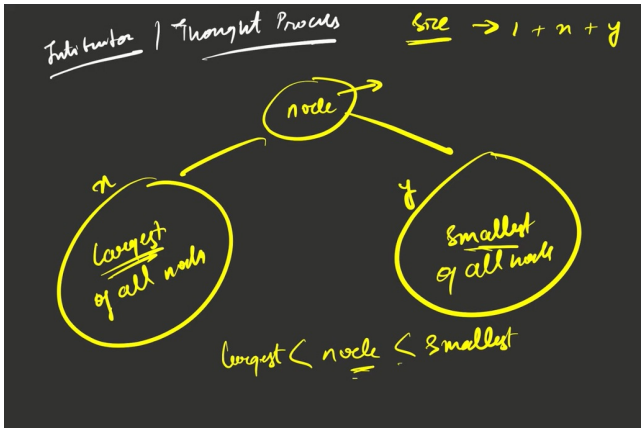
- The left subtree of a node contains only nodes with keys **less than** the node's key.
- The right subtree of a node contains only nodes with keys **greater than** the node's key.
- Both the left and right subtrees must also be binary search trees.

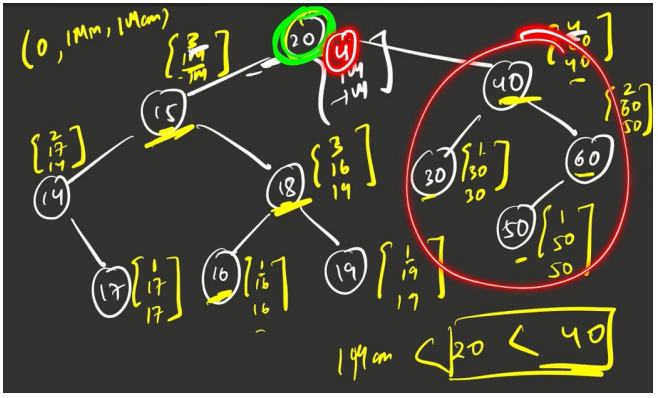
Example 1:




```
Input: root =
[1,4,3,2,4,2,5,null,null,null,null,null,null,4,6]
Output: 20
Explanation: Maximum sum in a valid Binary search tree is
obtained in root node with key equal to 3.
```

Largest BST in Binary Tree






```java
class NodeValue {
    public int maxNode, minNode, maxSize;

    NodeValue(int minNode, int maxNode, int maxSize) {
        this.maxNode = maxNode;
        this.minNode = minNode;
        this.maxSize = maxSize;
    }
};
class Solution {
    private NodeValue largestBSTSubtreeHelper(TreeNode root) {
        // An empty tree is a BST of size 0.
        if (root == null) {
            return new NodeValue(Integer.MAX_VALUE, Integer.MIN_VALUE, 0);
        }

        // Get values from left and right subtree of current tree.
        NodeValue left = largestBSTSubtreeHelper(root.left);
        NodeValue right = largestBSTSubtreeHelper(root.right);

        // Current node is greater than max in left AND smaller than min in right, it is a BST.
        if (left.maxNode < root.val && root.val < right.minNode) {
            // It is a BST.
            return new NodeValue(Math.min(root.val, left.minNode), Math.max(root.val, right.maxNode),left.maxSize + right.maxSize + 1);
        }

        // Otherwise, return [-inf, inf] so that parent can't be valid BST
        return new NodeValue(Integer.MIN_VALUE, Integer.MAX_VALUE, Math.max(left.maxSize, right.maxSize));
    }

    public int largestBSTSubtree(TreeNode root) {
        return largestBSTSubtreeHelper(root).maxSize;
    }
}
```

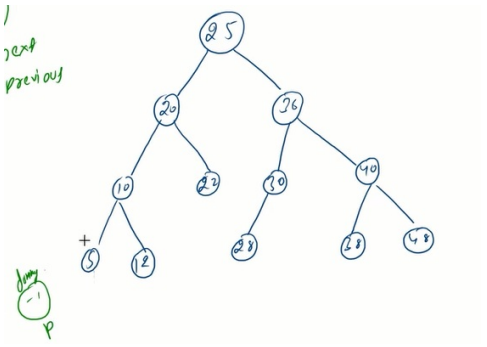
```java
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode() {}
 *     TreeNode(int val) { this.val = val; }
 *     TreeNode(int val, TreeNode left, TreeNode right) {
 *         this.val = val;
 *         this.left = left;
 *         this.right = right;
 *     }
 * }
 */
```

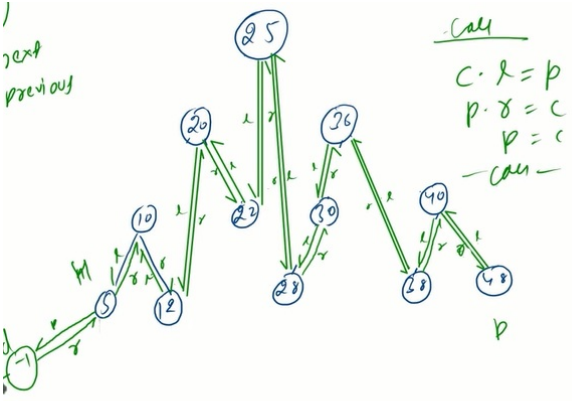
TC → O(N)
SC → O(1)

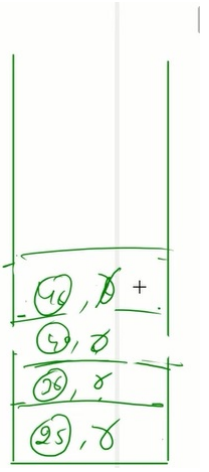
Day-22: Trees [Miscellaneous]

Convert Binary Search Tree to Sorted Doubly Linked List

1st method








```java
import java.util.Scanner;

public class Main {
  public static Scanner scn = new Scanner(System.in);

  public static class Node {
    int val = 0;
    Node left = null;
    Node right = null;

    Node(int val) {
      this.val = val;
    }
  }

  private static Node prev = null;

  private static void bToDLL_(Node root) {
    if (root == null) return;

    bToDLL_(root.left);

    prev.right = root;
    root.left = prev;
    prev = root;

    bToDLL_(root.right);
  }
  public static Node bToDLL(Node root) {
    Node dummy = new Node(-1);
    prev = dummy;
    bToDLL_(root);

    Node head = dummy.right;
    dummy.right = head.left = null;

    // circular
    head.left = prev;
    prev.right = head;

    return head;
  }

    node = node.right;
    if (node == head) {
      break;
    }
  }
}

public static Node constructFromInOrder_(int[] in, int si, int ei) {
  if (si > ei)
    return null;

  int mid = (si + ei) / 2;
  Node node = new Node(in[mid]);

  node.left = constructFromInOrder_(in, si, mid - 1);
  node.right = constructFromInOrder_(in, mid + 1, ei);

  return node;
}

public static Node constructFromInOrder(int[] inOrder) {
  int n = inOrder.length;
  return constructFromInOrder_(inOrder, 0, n - 1);
}

public static void solve() {
  int n = scn.nextInt();
  int[] in = new int[n];
  for (int i = 0; i < n; i++)
    in[i] = scn.nextInt();

  Node root = constructFromInOrder(in);
  root = bToDLL(root);
  display(root);
}

public static void main(String[] args) {
  solve();
}
}
```

Sample Input

```
7
1 2 3 4 5 6 7
```

Sample Output

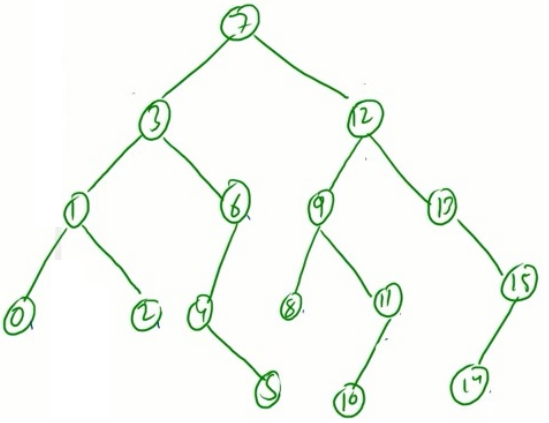
```
1 2 3 4 5 6 7
```

Convert Binary Search Tree to Sorted Doubly Linked List Using Stack Method

```java
import java.util.Scanner;
import java.util.LinkedList;

public class Main {
  public static Scanner scn = new Scanner(System.in);

  public static class Node {
    int val = 0;
    Node left = null;
    Node right = null;

    Node(int val) {
      this.val = val;
    }
  }

  public static void addAllLeftNodes(Node curr, LinkedList<Node> st) {
    while (curr != null) {
      st.addFirst(curr);
      curr = curr.left;
    }
  }
  public static Node bToDLL(Node root) {
    LinkedList<Node> st = new LinkedList<>(); // add first , remove first
    addAllLeftNodes(root, st);

    Node dummy = new Node(-1);
    Node prev = dummy;

    while (st.size() != 0) {
      Node curr = st.removeFirst();
      prev.right = curr;
      curr.left = prev;
      prev = curr;
      addAllLeftNodes(curr.right, st);
    }
    Node head = dummy.right;
    dummy.right = head.left = null;
    // for circular
    head.left = prev;
    prev.right = head;

    return head;
  }
  public static void display(Node node) {
    Node head = node;
    while (node != null) {
      System.out.print(node.val + " ");
      node = node.right;
      if (node == head) {
        break;
      }
    }
  }
  public static Node constructFromInOrder_(int[] in, int si, int ei) {
    if (si > ei)
      return null;

    int mid = (si + ei) / 2;
    Node node = new Node(in[mid]);

    node.left = constructFromInOrder_(in, si, mid - 1);
    node.right = constructFromInOrder_(in, mid + 1, ei);

    return node;
  }

  public static Node constructFromInOrder(int[] inOrder) {
    int n = inOrder.length;
    return constructFromInOrder_(inOrder, 0, n - 1);
  }

  public static void solve() {
    int n = scn.nextInt();
    int[] in = new int[n];
    for (int i = 0; i < n; i++)
      in[i] = scn.nextInt();

    Node root = constructFromInOrder(in);
    root = bToDLL(root);
    display(root);
  }

  public static void main(String[] args) {
    solve();
  }
}
```

Convert Binary Search Tree to Sorted Doubly Linked List

```java
import java.util.Scanner;

public class Main {
  public static Scanner scn = new Scanner(System.in);

  public static class Node {
    int val = 0;
    Node left = null;
    Node right = null;

    Node(int val) {
      this.val = val;
    }
  }


public static Node getrightMostNode(Node node, Node curr){
    while(node.right != null && node.right!=curr){
        node = node.right;
    }
    return node;
}

public static Node bToDLL(Node root){
    Node dummy = new Node(-1);
    Node prev = dummy,curr=root;
    while(curr!=null){
        Node left = curr.left;
        if(left==null){
            // create links
            prev.right = curr;
            curr.left = prev;
            prev = curr;
            // move
            curr = curr.right;
        }else{
            Node rightMostNode = getrightMostNode(left,curr);
            if(rightMostNode.right == null){// thread creation area
                rightMostNode.right = curr;
                curr = curr.left;
            }else{// thread destroy area
                rightMostNode.right = null;
                prev.right = curr;
                curr.left = prev;
                prev = curr;

                // move
                curr = curr.right;
            }
        }
    }
    Node head = dummy.right;
    dummy.right = head.left = null;
    // for circular
    prev.right = head;
    head.left = prev;
    return head;
}
```




```java
public static void display(Node node){
    Node head = node;
    while(node!=null){
        System.out.print(node.val + " ");
        node = node.right;
        if(node == head){
            break;
        }
    }
}
  public static Node constructFromInOrder_(int[] in, int si, int ei) {
    if (si > ei)
      return null;

    int mid = (si + ei) / 2;
    Node node = new Node(in[mid]);

    node.left = constructFromInOrder_(in, si, mid - 1);
    node.right = constructFromInOrder_(in, mid + 1, ei);

    return node;
  }

  public static Node constructFromInOrder(int[] inOrder) {
    int n = inOrder.length;
    return constructFromInOrder_(inOrder, 0, n - 1);
  }

  public static void solve() {
    int n = scn.nextInt();
    int[] in = new int[n];
    for (int i = 0; i < n; i++)
      in[i] = scn.nextInt();

    Node root = constructFromInOrder(in);
    root = bToDLL(root);
    display(root);
  }

  public static void main(String[] args) {
    solve();
  }
}
```

295. Find Median from Data Stream

Hard 5695 104 Add to List Share

The **median** is the middle value in an ordered integer list. If the size of the list is even, there is no middle value and the median is the mean of the two middle values.

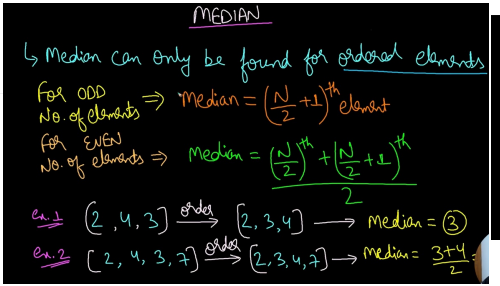
- For example, for `arr = [2,3,4]`, the median is `3`.
- For example, for `arr = [2,3]`, the median is `(2 + 3) / 2 = 2.5`.

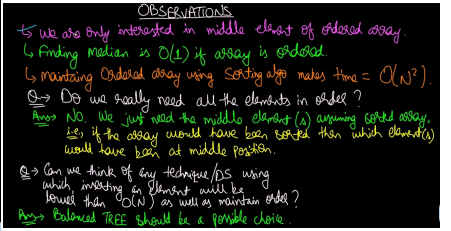
Implement the MedianFinder class:

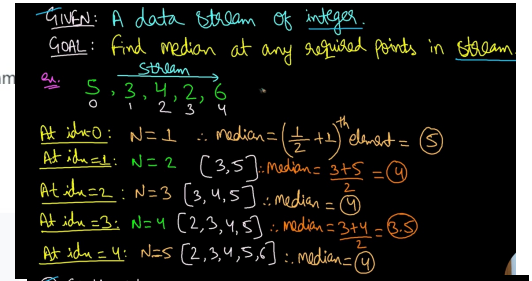
- `MedianFinder()` initializes the `MedianFinder` object.
- `void addNum(int num)` adds the integer `num` from the data stream to the data structure.
- `double findMedian()` returns the median of all elements so far. Answers within 10^{-5} of the actual answer will be accepted.

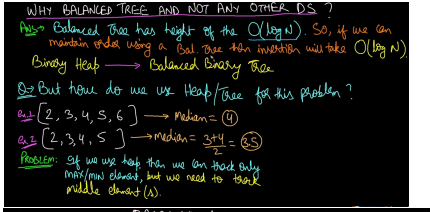
Example 1:

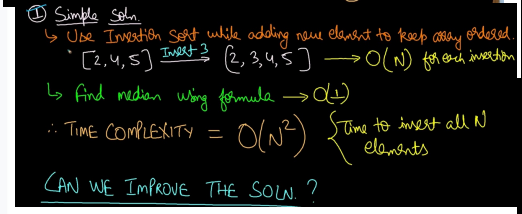
```
Input
["MedianFinder", "addNum", "addNum", "findMedian", "addNum", "findMedian"]
[[], [1], [2], [], [3], []]
Output
[null, null, null, 1.5, null, 2.0]

Explanation
MedianFinder medianFinder = new MedianFinder();
medianFinder.addNum(1);    // arr = [1]
medianFinder.addNum(2);    // arr = [1, 2]
medianFinder.findMedian(); // return 1.5 (i.e., (1 + 2) / 2)
medianFinder.addNum(3);    // arr[1, 2, 3]
medianFinder.findMedian(); // return 2.0
```

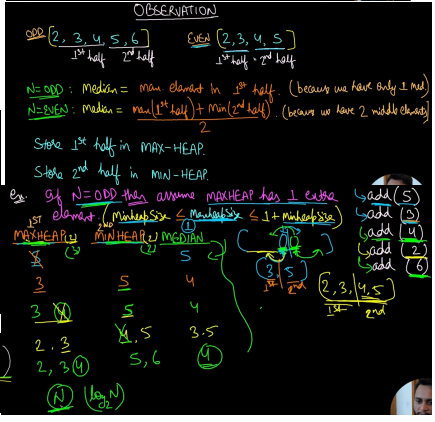


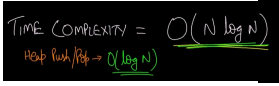












```cpp
class MedianFinder{
public:
    priority_queue<int> maxheap;// 1st half(left half)
    priority_queue<int,vector<int>,greater<int>> minheap;//2nd half(right half)
// initialize your data structure
    MedianFinder(){

    }
    void addNum(int num){
        int lsize = maxheap.size();
        int rsize = minheap.size();
        if(lsize==0){
            maxheap.push(num);
        }else if(lsize==rsize){
            if(num<minheap.top()){
                maxheap.push(num);
            }else{
                int temp = minheap.top();
                minheap.pop();
                minheap.push(num);
                maxheap.push(temp);
            }
        }else{
            if(minheap.size()==0){
                if(num>maxheap.top()){
                    minheap.push(num);
                }else{
                    int temp = maxheap.top();
                    maxheap.pop();
                    maxheap.push(num);
                    minheap.push(temp);
                }
            }else if(num>=minheap.top()){
                minheap.push(num);
            }else{
                if(num<maxheap.top()){
                    int temp = maxheap.top();
                    maxheap.pop();
                    maxheap.push(num);
                    minheap.push(temp);
                }else{
                    minheap.push(num);
                }
            }
        }
    }
    double findMedian(){
        int lsize = maxheap.size();
        int rsize = minheap.size();
        if(lsize > rsize){
            return double(maxheap.top());
        }else{
            return (double(maxheap.top())+double(minheap.top()))/2;
        }
    }
};
```

```java
class MedianFinder {

        /** initialize your data structure here. */
    PriorityQueue<Integer> minQueue;
    PriorityQueue<Integer> maxQueue;
    int size1 = 0;
    int size2 = 0;
    public MedianFinder() {
        minQueue = new PriorityQueue<>();
        maxQueue = new PriorityQueue<>(Collections.reverseOrder());
        size1 = 0;
        size2 = 0;
    }

    public void addNum(int num) {
        if(size1 == 0){
            minQueue.add(num);
            size1++;
            return;
        }
        if(size1 >= size2){
            if(num > minQueue.peek()){
                int poll = minQueue.poll();
                minQueue.add(num);
                maxQueue.add(poll);
                size2++;
            }else{
                maxQueue.add(num);
                size2++;
            }
        }else{
            if(maxQueue.peek() > num){
                int poll = maxQueue.poll();
                minQueue.add(poll);
                maxQueue.add(num);
            }else{
                minQueue.add(num);
            }
            size1++;
        }
    }

    public double findMedian() {
        if(size1 == size2){
            return (minQueue.peek() * 1.0 + maxQueue.peek() * 1.0)/2;
        }else{
            if(size1 > size2){
                return minQueue.peek();
            }else{
                return maxQueue.peek();
            }
        }
    }
}
```

K-th largest element in a stream

703. Kth Largest Element in a Stream

Easy 1710 1006 Add to List Share

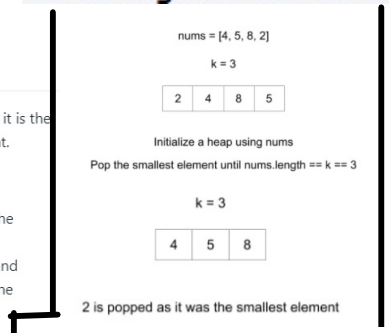
Design a class to find the k^{th} largest element in a stream. Note that it is the k^{th} largest element in the sorted order, not the k^{th} distinct element.

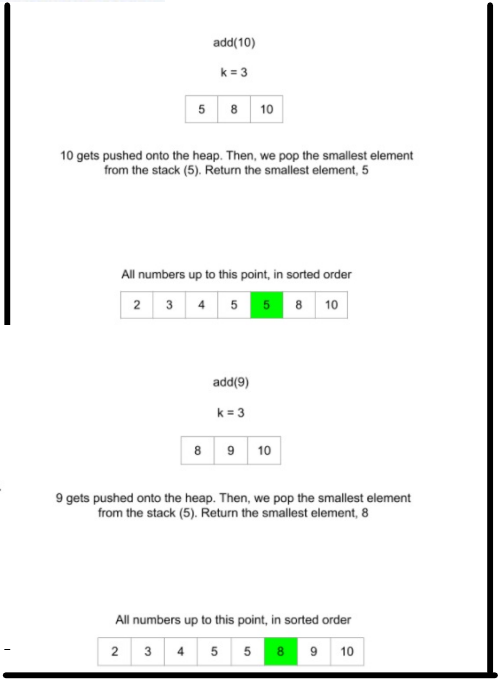
Implement `KthLargest` class:

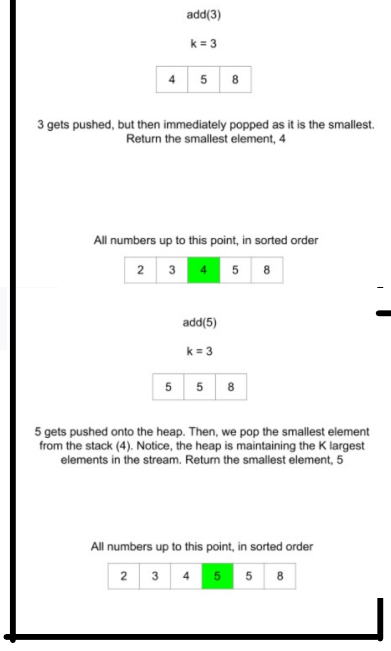
- `KthLargest(int k, int[] nums)` Initializes the object with the integer `k` and the stream of integers `nums`.
- `int add(int val)` Appends the integer `val` to the stream and returns the element representing the k^{th} largest element in the stream.

Example 1:

```
Input
["KthLargest", "add", "add", "add", "add", "add"]
[[3, [4, 5, 8, 2]], [3], [5], [10], [9], [4]]
Output
[null, 4, 5, 5, 8, 8]

Explanation
KthLargest kthLargest = new KthLargest(3, [4, 5, 8, 2]);
kthLargest.add(3);   // return 4
kthLargest.add(5);   // return 5
kthLargest.add(10);  // return 5
kthLargest.add(9);   // return 8
kthLargest.add(4);   // return 8
```








```java
class KthLargest {
    private static int k;
    private PriorityQueue<Integer> heap;

    public KthLargest(int k, int[] nums) {
        this.k = k;
        heap = new PriorityQueue<>();

        for (int num: nums) {
            heap.offer(num);
        }

        while (heap.size() > k) {
            heap.poll();
        }
    }

    public int add(int val) {
        heap.offer(val);
        if (heap.size() > k) {
            heap.poll();
        }

        return heap.peek();
    }
}
```

Distinct Numbers in Window

Medium | 96 | 6 | Add to favorites

Asked In: AMAZON

Problem Description

You are given an array of **N** integers, $A_1, A_2, ..., A_N$ and an integer **B**. Return the of count of distinct numbers in all windows of size **B**.

Formally, return an array of size **N-B+1** where **i'th** element in this array contains number of distinct elements in sequence $A_i, A_{i+1}, ..., A_{i+B-1}$.

NOTE: if **B** > **N**, return an empty array.

Input Format

First argument is an integer array A

Second argument is an integer B.

Output Format

Return an integer array.

Example Input

Input 1:

```
A = [1, 2, 1, 3, 4, 3]
B = 3
```

Input 2:

```
A = [1, 1, 2, 2]
B = 1
```

Example Output

Output 1:

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

Output 2:

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

Example Explanation

Explanation 1:

```
A=[1, 2, 1, 3, 4, 3] and B = 3
All windows of size B are
[1, 2, 1]
[2, 1, 3]
[1, 3, 4]
[3, 4, 3]
So, we return an array [2, 3, 3, 2].
```

Explanation 2:

```
Window size is 1, so the output array is [1, 1, 1, 1].
```

```java
public static ArrayList<Integer> solution(int[] arr, int k){
    ArrayList<Integer> list = new ArrayList<>();
    HashMap<Integer,Integer> map = new HashMap<>();

    for(int i=0; i<k-1; i++){
        map.put(arr[i], map.getOrDefault(arr[i],0)+1);
    }
    for(int j=0,i=k-1; i<arr.length;){
        map.put(arr[i], map.getOrDefault(arr[i],0)+1);// acquire

        list.add(map.size());// work

        // release
        int freq = map.get(arr[j]);
        if(freq==1){
            map.remove(arr[j]);
        }else{
            map.put(arr[j],freq-1);
        }
        i++;
        j++;
    }

    return list;
}
```

```java
public static ArrayList<Integer> solution(int[] arr, int k) {
    ArrayList<Integer> list = new ArrayList<>();
    HashMap<Integer, Integer> map = new HashMap<>();

    int i = 0;
    while(i <= k - 2){
        map.put(arr[i], map.getOrDefault(arr[i], 0) + 1);
        i++;
    }

    i--;
    int j = -1;

    while(i < arr.length - 1){
        i++;
        map.put(arr[i], map.getOrDefault(arr[i], 0) + 1); // acquire

        list.add(map.size()); // work

        // release
        j++;
        int freq = map.get(arr[j]);
        if(freq == 1){
            map.remove(arr[j]);
        } else {
            map.put(arr[j], freq - 1);
        }
    }


    return list;
}
```

215. Kth Largest Element in an Array

Medium 7410 421 Add to List Share

Given an integer array `nums` and an integer `k`, return *the* k^{th} *largest element in the array.*

Note that it is the k^{th} largest element in the sorted order, not the k^{th} distinct element.

Example 1:

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

Example 2:

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

```java
class Solution{
    public int findKthLargest(int[] nums, int k){
        PriorityQueue<Integer> pq = new PriorityQueue<>();
        for(int el:nums){
            pq.add(el);
            if(pq.size()>k){
                pq.poll();
            }
        }
        return pq.poll();
    }
}
```

733. Flood Fill

Easy 2949 307 Add to List Share

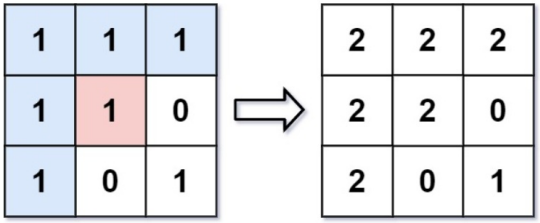
An image is represented by an `m x n` integer grid `image` where `image[i][j]` represents the pixel value of the image.

You are also given three integers `sr`, `sc`, and `newColor`. 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 `newColor`.

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

Example 1:




```
Input: image = [[1,1,1],[1,1,0],[1,0,1]], sr = 1, sc = 1,
newColor = 2
Output: [[2,2,2],[2,2,0],[2,0,1]]
Explanation: From the center of the image with position (sr, sc)
= (1, 1) (i.e., the red pixel), all pixels connected by a path of
the same color as the starting pixel (i.e., the blue pixels) are
colored with the new color.
Note the bottom corner is not colored 2, because it is not 4-
directionally connected to the starting pixel.
```

Example 2:

```
Input: image = [[0,0,0],[0,0,0]], sr = 0, sc = 0, newColor = 2
Output: [[2,2,2],[2,2,2]]
```

Constraints:

- `m == image.length`
- `n == image[i].length`
- `1 <= m, n <= 50`
- `0 <= image[i][j], newColor < 2^16`
- `0 <= sr < m`
- `0 <= sc < n`

Approach #1: Depth-First Search [Accepted]

Intuition

We perform the algorithm explained in the problem description: paint the starting pixels, plus adjacent pixels of the same color, and so on.

Algorithm

Say `color` is the color of the starting pixel. Let's floodfill the starting pixel: we change the color of that pixel to the new color, then check the 4 neighboring pixels to make sure they are valid pixels of the same `color`, and of the valid ones, we floodfill those, and so on.

We can use a function `dfs` to perform a floodfill on a target pixel.

```java
class Solution {
    public int[][] floodFill(int[][] image, int sr, int sc, int newColor) {
        int color = image[sr][sc];
        if (color != newColor) dfs(image, sr, sc, color, newColor);
        return image;
    }
    public void dfs(int[][] image, int r, int c, int color, int newColor) {
        if (image[r][c] == color) {
            image[r][c] = newColor;
            if (r >= 1) dfs(image, r-1, c, color, newColor);
            if (c >= 1) dfs(image, r, c-1, color, newColor);
            if (r+1 < image.length) dfs(image, r+1, c, color, newColor);
            if (c+1 < image[0].length) dfs(image, r, c+1, color, newColor);
        }
    }
}
```

Complexity Analysis

- Time Complexity: $O(N)$, where N is the number of pixels in the image. We might process every pixel.
- Space Complexity: $O(N)$, the size of the implicit call stack when calling `dfs`.

Flood Fill

Flood Fill

Easy

Prev Next

1. You are given a number n, representing the number of rows.
2. You are given a number m, representing the number of columns.
3. You are given n*m numbers, representing elements of 2d array a. The numbers can be 1 or 0 only.
4. You are standing in the top-left corner and have to reach the bottom-right corner. Only four moves are allowed 't' (1-step up), 'l' (1-step left), 'd' (1-step down) 'r' (1-step right). You can only move to cells which have 0 value in them. You can't move out of the boundaries or in the cells which have value 1 in them (1 means obstacle)
5. Complete the body of floodfill function - without changing signature - to print all paths that can be used to move from top-left to bottom-right.

Note1 -> Please check the sample input and output for details
Note2 -> If all four moves are available make moves in the order 't', 'l', 'd' and 'r'

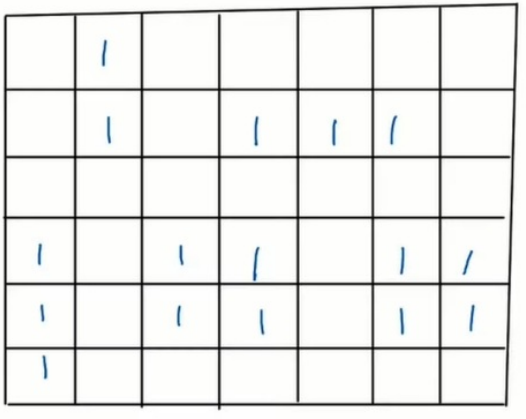
Sample Input

```
3 3
0 0 0
1 0 1
0 0 0
```

Sample Output

```
rddr
```

```java
import java.io.*;
import java.util.*;

public class Main {

  public static void main(String[] args) {
    Scanner scn = new Scanner(System.in);
    int n = scn.nextInt();
    int m = scn.nextInt();
    int[][] arr = new int[n][m];
    for (int i = 0; i < n; i++) {
      for (int j = 0; j < m; j++) {
        arr[i][j] = scn.nextInt();
      }
    }
    boolean[][] visited = new  boolean[n][m];
    floodfill(arr, 0, 0, "", visited);
  }

  // asf -> answer so far
  public static void floodfill(int[][] maze, int row, int col, String psf, boolean[][] visited) {

    if (row < 0 || col < 0 || row == maze.length || col == maze[0].length || maze[row][col] == 1 || visited[row][col] == true) {
      return;
    }
    if (row == maze.length - 1 && col == maze[0].length - 1) {
      System.out.println(psf);
      return;
    }
    visited[row][col] = true;
    floodfill(maze, row - 1, col, psf + "t", visited);
    floodfill(maze, row, col - 1, psf + "l", visited);
    floodfill(maze, row + 1, col, psf + "d", visited);
    floodfill(maze, row, col + 1, psf + "r", visited);
    visited[row][col] = false;
  }
}
```



Graphs - Part 1

133. Clone Graph

Medium 4153 2018 Add to List Share

Given a reference of a node in a **connected** undirected graph.

Return a **deep copy** (clone) of the graph.

Each node in the graph contains a value (`int`) and a list (`List[Node]`) of its neighbors.

```
class Node {
    public int val;
    public List<Node> neighbors;
}
```

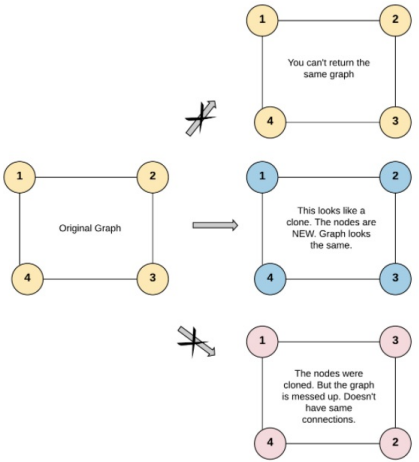
Test case format:

For simplicity, each node's value is the same as the node's index (1-indexed). For example, the first node with `val == 1`, the second node with `val == 2`, and so on. The graph is represented in the test case using an adjacency list.

An adjacency list is a collection of unordered **lists** used to represent a finite graph. Each list describes the set of neighbors of a node in the graph.

The given node will always be the first node with `val = 1`. You must return the **copy of the given node** as a reference to the cloned graph.

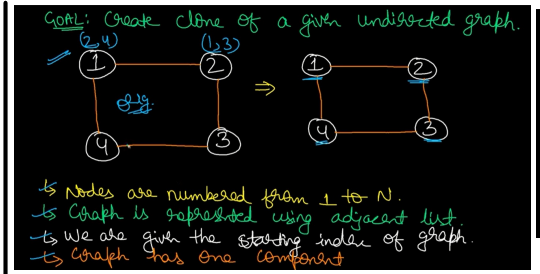
Example 1:

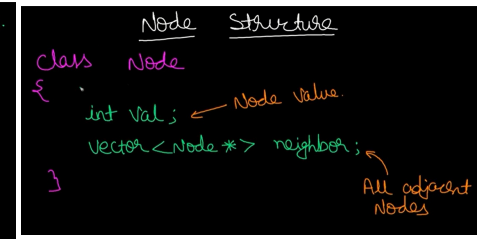


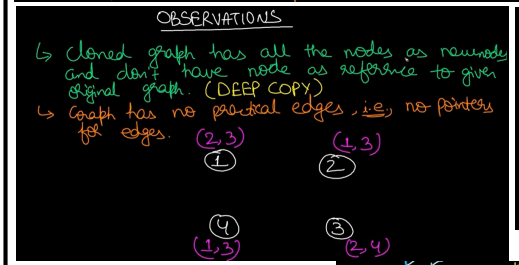

```
Input: adjList = [[2,4],[1,3],[2,4],[1,3]]
Output: [[2,4],[1,3],[2,4],[1,3]]
Explanation: There are 4 nodes in the graph.
1st node (val = 1)'s neighbors are 2nd node (val = 2) and 4th node (val = 4).
2nd node (val = 2)'s neighbors are 1st node (val = 1) and 3rd node (val = 3).
3rd node (val = 3)'s neighbors are 2nd node (val = 2) and 4th node (val = 4).
4th node (val = 4)'s neighbors are 1st node (val = 1) and 3rd node (val = 3).
```

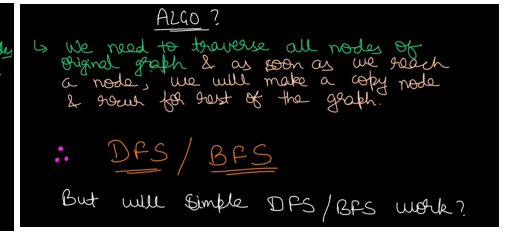
Example 2:

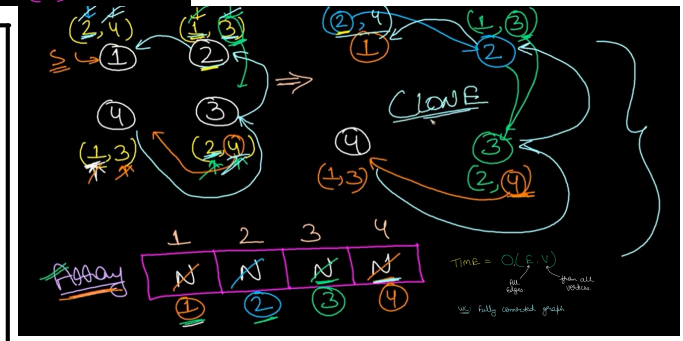
```
Input: adjList = [[]]
Output: [[]]
Explanation: Note that the input contains one empty list. The graph consists of only one node with val = 1 and it does not have any neighbors.
```











```cpp
/*
// Definition for a Node.
class Node {
public:
    int val;
    vector<Node*> neighbors;
    Node() {
        val = 0;
        neighbors = vector<Node*>();
    }
    Node(int _val) {
        val = _val;
        neighbors = vector<Node*>();
    }
    Node(int _val, vector<Node*> _neighbors) {
        val = _val;
        neighbors = _neighbors;
    }
};
*/

class Solution{
    void dfs(Node* curr, Node* node, vector<Node *> & visited){
        visited[node->val] = node;
        for(auto ele: curr->neighbors){
            if(visited[ele->val]==NULL){
                Node *newnode = new Node(ele->val);
                (node->neighbors).push_back(newnode);
                dfs(ele,newnode,visited);
            }else{
                (node->neighbors).push_back(visited[ele->val]);
            }
        }
    }
public:
    Node* cloneGraph(Node* node){
        if(node==NULL){
            return NULL;
        }
        vector<Node *> visited(1000,NULL);
        Node* copy = new Node(node->val);
        visited[node->val] = copy;
        for(auto curr: node->neighbors){
            if(visited[curr->val]==NULL){
                Node *newnode = new Node(curr->val);
                (copy->neighbors).push_back(newnode);
                dfs(curr,newnode,visited);
            }else{
                (copy->neighbors).push_back(visited[curr->val]);
            }
        }
        return copy;
    }
};
```

```java
/*
// Definition for a Node.
class Node {
    public int val;
    public List<Node> neighbors;
    public Node() {
        val = 0;
        neighbors = new ArrayList<Node>();
    }
    public Node(int _val) {
        val = _val;
        neighbors = new ArrayList<Node>();
    }
    public Node(int _val, ArrayList<Node> _neighbors) {
        val = _val;
        neighbors = _neighbors;
    }
}
*/
class Solution {
    public Node cloneGraph(Node node) {
        if(node == null) return null;
         Map<Node, Node> map = new HashMap<>();
         Queue<Node> queue = new ArrayDeque<>();

    queue.add(node);
    map.put(node, new Node(node.val, new ArrayList<>()));
    while (!queue.isEmpty()) {
        Node h = queue.poll();

        for (Node neighbor : h.neighbors) {
            if (!map.containsKey(neighbor)) {
                map.put(neighbor, new Node(neighbor.val, new ArrayList<>()));
                queue.add(neighbor);
            }
            map.get(h).neighbors.add(map.get(neighbor));
        }
    }

    return map.get(node);
    }
}
```

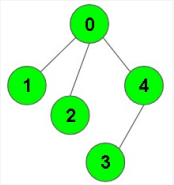
DFS of Graph

Easy Accuracy: 49.62% Submissions: 93368 Points: 2

Given a connected undirected graph. Perform a Depth First Traversal of the graph.
Note: Use recursive approach to find the DFS traversal of the graph starting from the 0th vertex from left to right according to the graph..

Example 1:

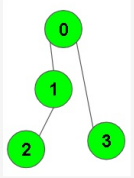
Input:



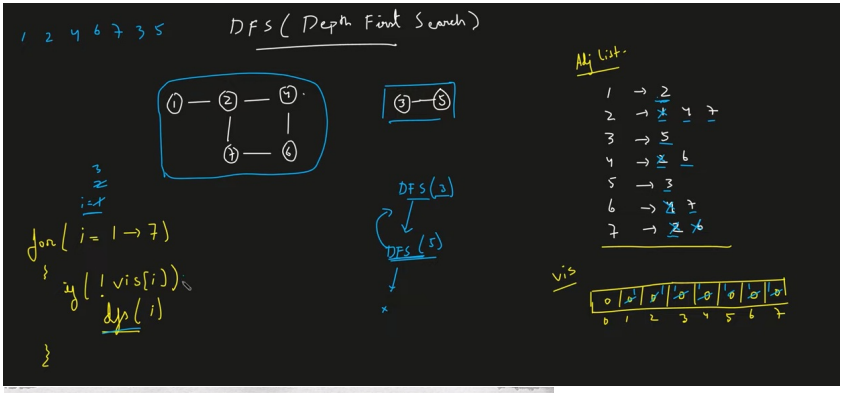
Output: 0 1 2 4 3
Explanation:
0 is connected to 1, 2, 4.
1 is connected to 0.
2 is connected to 0.
3 is connected to 4.
4 is connected to 0, 3.
so starting from 0, it will go to 1 then 2
then 4, and then from 4 to 3.
Thus dfs will be 0 1 2 4 3.

Example 2:

Input:



Output: 0 1 2 3
Explanation:
0 is connected to 1 , 3.
1 is connected to 2.
2 is connected to 1.
3 is connected to 0.
so starting from 0, it will go to 1 then 2
then back to 0 then 0 to 3
thus dfs will be 0 1 2 3.



TC will be O(N+E) and SC will be O(N+E) + O(N) + O(N)

N is time taken for visiting N nodes, and E is for traveling through adjacent nodes overall. Space for adj list, vis array and Auxiliary Space

```java
class Solution{
  public void dfs(int node, boolean vis[], ArrayList<ArrayList<Integer>> adj, ArrayList<Integer> storeDfs){
    storeDfs.add(node);
    vis[node] = true;
    for(Integer it: adj.get(node)){
      if(vis[it]==false){
        dfs(it,vis,adj,storeDfs);
      }
    }
  }
  public ArrayList<Integer> dfsOfGraph(int V, ArrayList<ArrayList<Integer>> adj){
    ArrayList<Integer> storeDfs = new ArrayList<>();
    boolean vis[] = new boolean[V+1];
    for(int i=1; i<=V; i++){
      if(vis[i] == 0){
        dfs(i,vis,adj,storeDfs);
      }
    }
    return storeDfs;
  }
}
```

BFS of graph

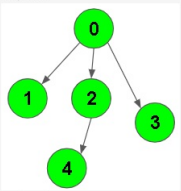
Easy Accuracy: 42.71% Submissions: 100k+ Points: 2

Given a directed graph. The task is to do Breadth First Traversal of this graph starting from 0.

Note: One can move from node u to node v only if there's an edge from u to v and find the BFS traversal of the graph starting from the 0th vertex, from left to right according to the graph. Also, you should only take nodes directly or indirectly connected from Node 0 in consideration.

Example 1:

Input:



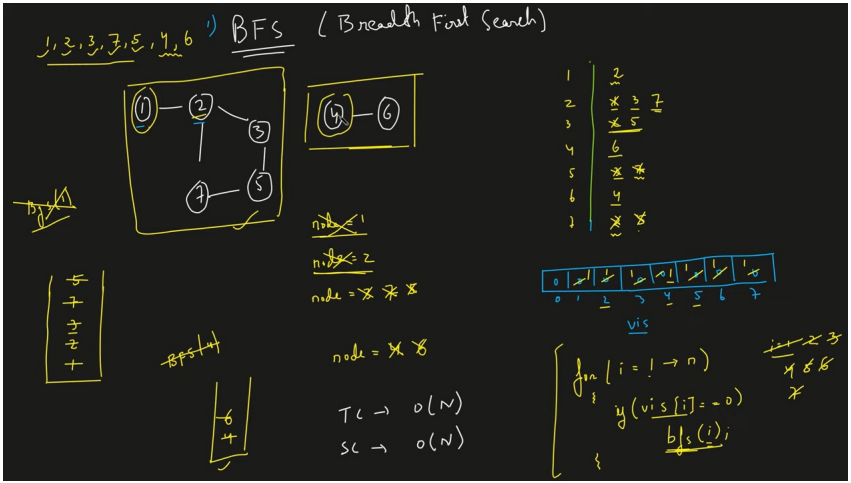
Output: 0 1 2 3 4

Explanation:

0 is connected to 1 , 2 , 3.

2 is connected to 4.

so starting from 0, it will go to 1 then 2 then 3.After this 2 to 4, thus bfs will be 0 1 2 3 4.

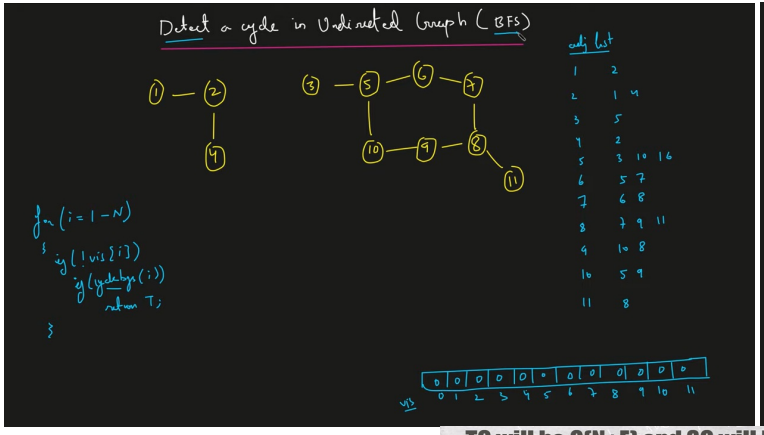


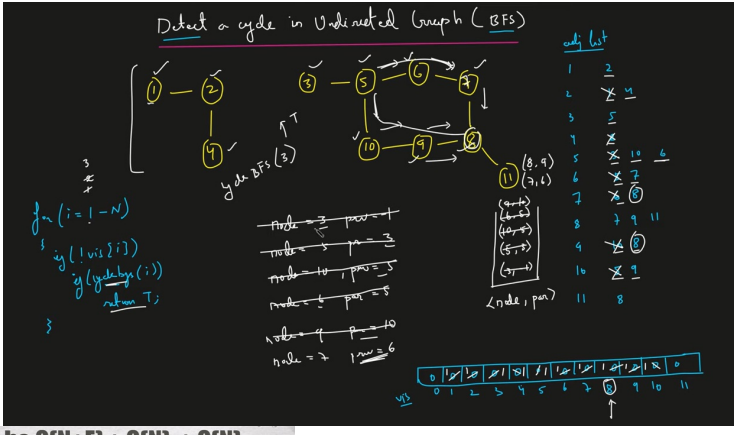
TC will be O(N+E) and SC will be O(N+E) + O(N) + O(N)

N is time taken for visiting N nodes, and E is for traveling through adjacent nodes overall. Space for adj list, vis array and queue

```java
class Solution{
  public ArrayList<Integer> bfsOfGraph(int V, ArrayList<ArrayList<Integer>> adj){
    ArrayList<Integer> bfs = new ArrayList<>();
    boolean vis[] = new boolean[V+1];

    for(int i=1; i<=V; i++){
      if(vis[i] == 0){
        Queue<Integer> q = new LinkedList<>();
        q.add(i);
        vis[i] = true;

        while(!q.isEmpty()){
          Integer node = q.poll();
          bfs.add(node);

          for(Integer it: adj.get(node)){
            if(vis[it] == false){
              vis[it] = true;
              q.add(it);
            }
          }
        }
      }
    }
    return bfs;
  }
}
```

Cycle Detection in Undirected Graph using BFS





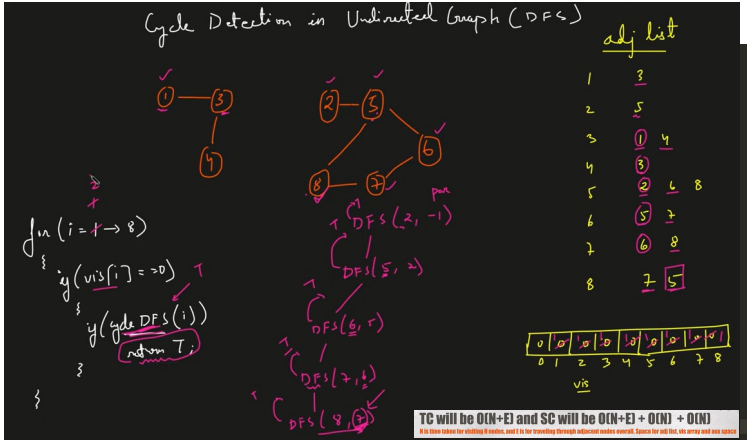
TC will be O(N+E) and SC will be O(N+E) + O(N) + O(N)

N is time taken for visiting N nodes, and E is for traveling through adjacent nodes overall. Space for adj list, vis array and queue

```java
import java.util.*;
import java.lang.*;
import java.io.*;
class GFG
{
    public static void main(String[] args) throws IOException
    {
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
        int T = Integer.parseInt(br.readLine().trim());
        while(T-->0)
        {
            String[] s = br.readLine().trim().split(" ");
            int V = Integer.parseInt(s[0]);
            int E = Integer.parseInt(s[1]);
            ArrayList<ArrayList<Integer>>adj = new ArrayList<>();
            for(int i = 0; i < V; i++)
                adj.add(i, new ArrayList<Integer>());
            for(int i = 0; i < E; i++){
                String[] S = br.readLine().trim().split(" ");
                int u = Integer.parseInt(S[0]);
                int v = Integer.parseInt(S[1]);
                adj.get(u).add(v);
                adj.get(v).add(u);
            }
            Solution obj = new Solution();
            boolean ans = obj.isCycle(V, adj);
            if(ans)
                System.out.println("1");
            else
                System.out.println("0");
        }
    }
}// } Driver Code Ends
```

```java
class Node {
    int first;
    int second;
    public Node(int first, int second) {
        this.first = first;
        this.second = second;
    }
}
class Solution{
   static boolean checkForCycle(ArrayList<ArrayList<Integer>> adj, int s, boolean vis[], int parent[]){
        Queue<Node> q =  new LinkedList<>(); //BFS
        q.add(new Node(s, -1));
        vis[s] =true;

        while(!q.isEmpty()){
            int node = q.peek().first;
            int par = q.peek().second;
            q.remove();

            for(Integer it: adj.get(node)){
                if(vis[it]==false) {
                    q.add(new Node(it, node));
                    vis[it] = true;
                }else if(par != it) return true;
            }
        }
        return false;
    }
    public boolean isCycle(int V, ArrayList<ArrayList<Integer>> adj){
        boolean vis[] = new boolean[V];
        Arrays.fill(vis,false);
        int parent[] = new int[V];
        Arrays.fill(parent, -1);

        for(int i=0;i<V;i++)
            if(vis[i]==false)
                if(checkForCycle(adj, i,vis, parent))
                    return true;

        return false;
    }
}
```

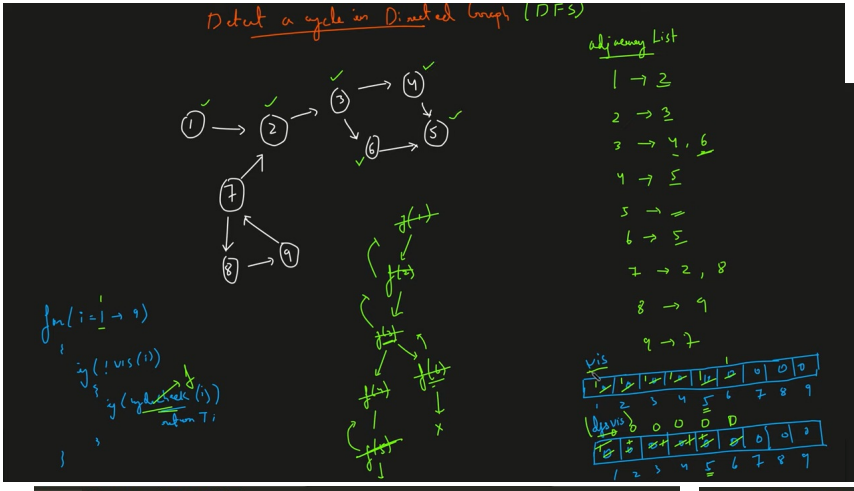
Cycle Detection in Undirected Graph using DFS

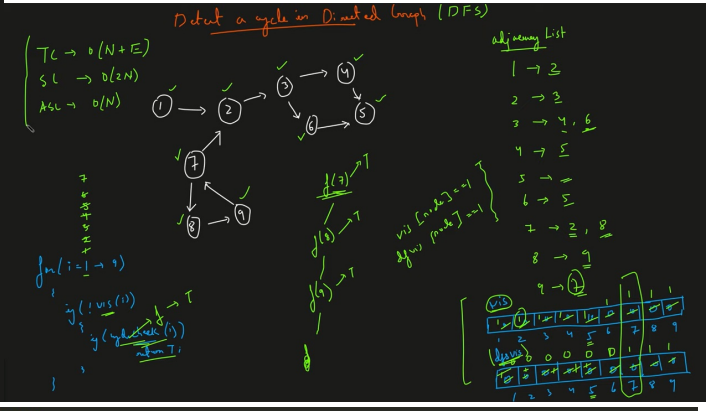



```java
class Solution{
    public boolean checkForCycle(int node, int parent, boolean vis[], ArrayList<ArrayList<Integer>> adj) {
        vis[node] = true;
        for(Integer it: adj.get(node)) {
            if(vis[it] == false) {
                if(checkForCycle(it, node, vis, adj) == true)
                    return true;
            }
            else if(it!=parent)
                return true;
        }

        return false;
    }
    // 0-based indexing Graph
    public boolean isCycle(int V, ArrayList<ArrayList<Integer>> adj){
        boolean vis[] = new boolean[V];
        for(int i = 0;i<V;i++) {
            if(vis[i] == false) {
                if(checkForCycle(i, -1, vis, adj))
                    return true;
            }
        }
        return false;
    }
}
```

```java
import java.util.*;
import java.lang.*;
import java.io.*;
class GFG{
    public static void main(String[] args) throws IOException{
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
        int T = Integer.parseInt(br.readline().trim());
        while(T--30){
            String[] s = br.readLine().trim().split(" ");
            int V = Integer.parseInt(s[0]);
            int E = Integer.parseInt(s[1]);
            ArrayList<ArrayList<Integer>>adj = new ArrayList<>();
            for(int i = 0; i < V; i++)
                adj.add(i, new ArrayList<Integer>());
            for(int i = 0; i < E; i++){
                String[] S = br.readline().trim().split(" ");
                int u = Integer.parseInt(S[0]);
                int v = Integer.parseInt(S[1]);
                adj.get(u).add(v);
                adj.get(v).add(u);
            }
            Solution obj = new Solution();
            boolean ans = obj.isCycle(V, adj);
            if(ans)
                System.out.println("1");
            else
                System.out.println("0");
        }
    }
}// } Driver Code Ends
```

Cycle Detection in Directed Graph using DFS

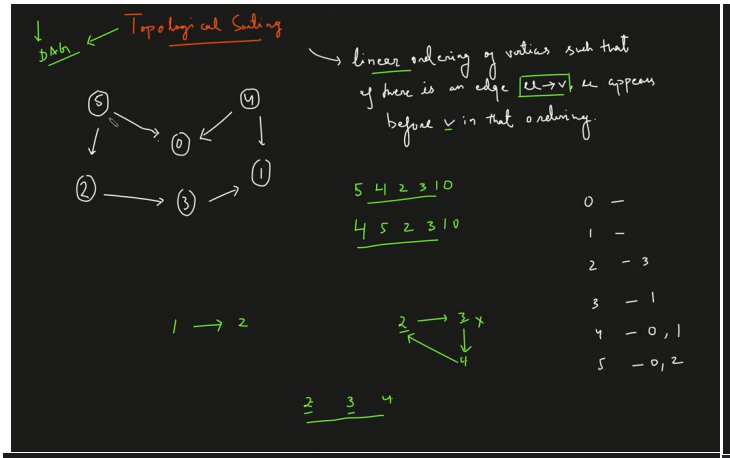


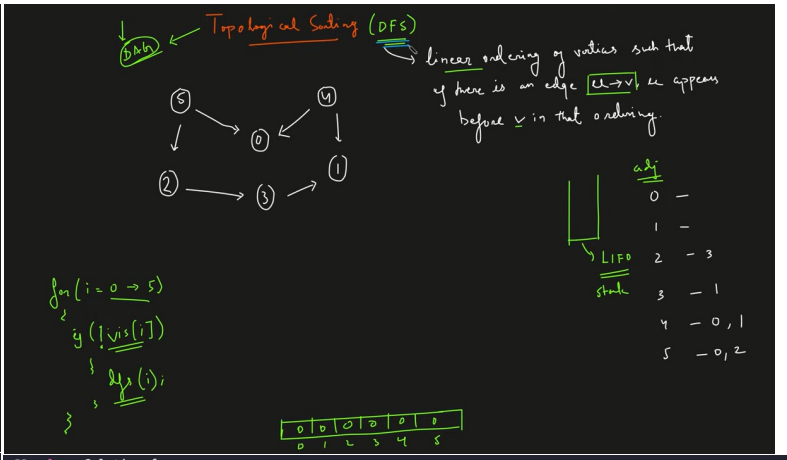


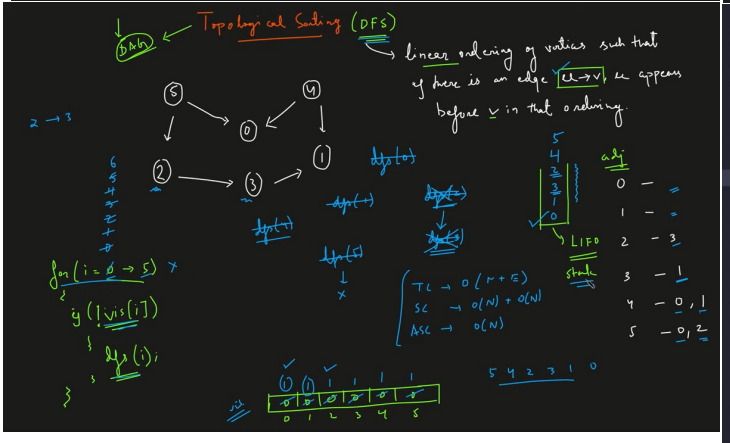

```java
import java.util.*;
import java.io.*;
import java.lang.*;

class DriverClass{
    public static void main (String[] args) {
        Scanner sc = new Scanner(System.in);
        int t = sc.nextInt();

        while(t-- > 0){
            ArrayList<ArrayList<Integer>> list = new ArrayList<>();
            int V = sc.nextInt();
            int E = sc.nextInt();
            for(int i = 0; i < V+1; i++)
                list.add(i, new ArrayList<Integer>());
            for(int i = 0; i < E; i++){
                int u = sc.nextInt();
                int v = sc.nextInt();
                list.get(u).add(v);
            }
            if(new Solution().isCyclic(V, list) == true)
                System.out.println("1");
            else System.out.println("0");
        }
    }
}
```

```java
class Solution {
    private boolean checkCycle(int node,  ArrayList<ArrayList<Integer>> adj, int vis[], int dfsVis[]) {
        vis[node] = 1;
        dfsVis[node] = 1;

        for(Integer it: adj.get(node)) {
            if(vis[it] == 0) {
                if(checkCycle(it, adj, vis, dfsVis) == true) {
                    return true;
                }
            } else if(dfsVis[it] == 1) {
                return true;
            }
        }
        dfsVis[node] = 0;
        return false;
    }
    public boolean isCyclic(int N, ArrayList<ArrayList<Integer>> adj) {
        int vis[] = new int[N];
        int dfsVis[] = new int[N];

        for(int i = 0;i<N;i++) {
            if(vis[i] == 0) {
                if(checkCycle(i, adj, vis, dfsVis) == true) return true;
            }
        }
        return false;
    }
}
```

Topological Sort (DFS)

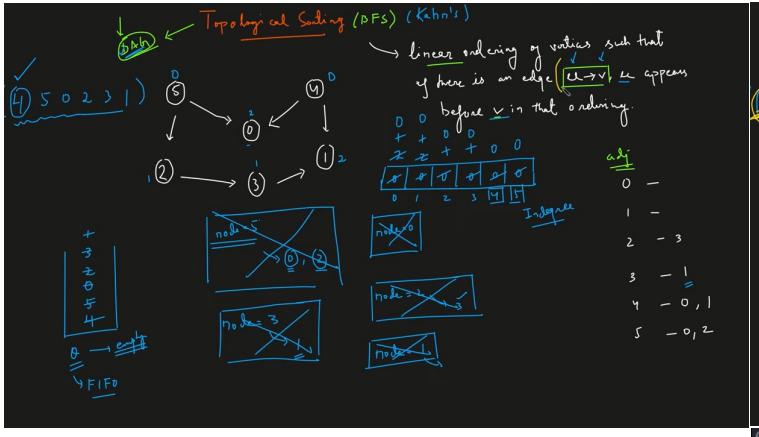


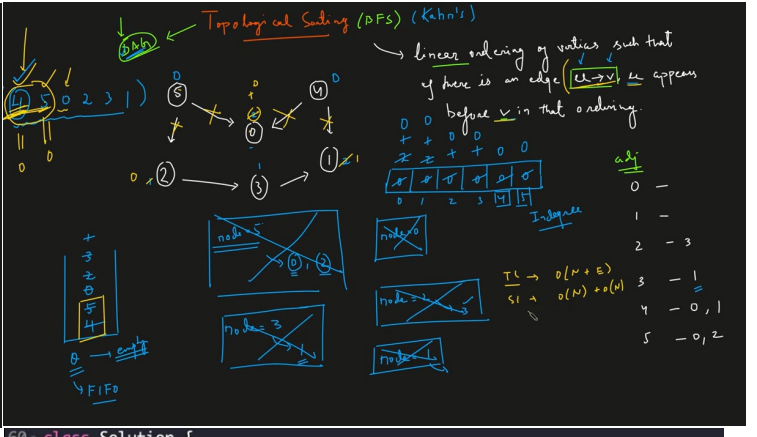





```java
class Solution {
    static void findTopoSort(int node, int vis[], ArrayList<ArrayList<Integer>> adj, Stack<Integer> st) {
        vis[node] = 1;
        for(Integer it: adj.get(node)) {
            if(vis[it] == 0) {
                findTopoSort(it, vis, adj, st);
            }
        }
        st.push(node);
    }
    static int[] topoSort(int N, ArrayList<ArrayList<Integer>> adj) {
        Stack<Integer> st = new Stack<Integer>();
        int vis[] = new int[N];

        for(int i = 0;i<N;i++) {
            if(vis[i] == 0) {
                findTopoSort(i, vis, adj, st);
            }
        }

        int topo[] = new int[N];
        int ind = 0;
        while(!st.isEmpty()) {
            topo[ind++] = st.pop();
        }
        // for(int i = 0;i<N;i++) System.out.println(topo[i] + " ");
        return topo;
    }
}
```

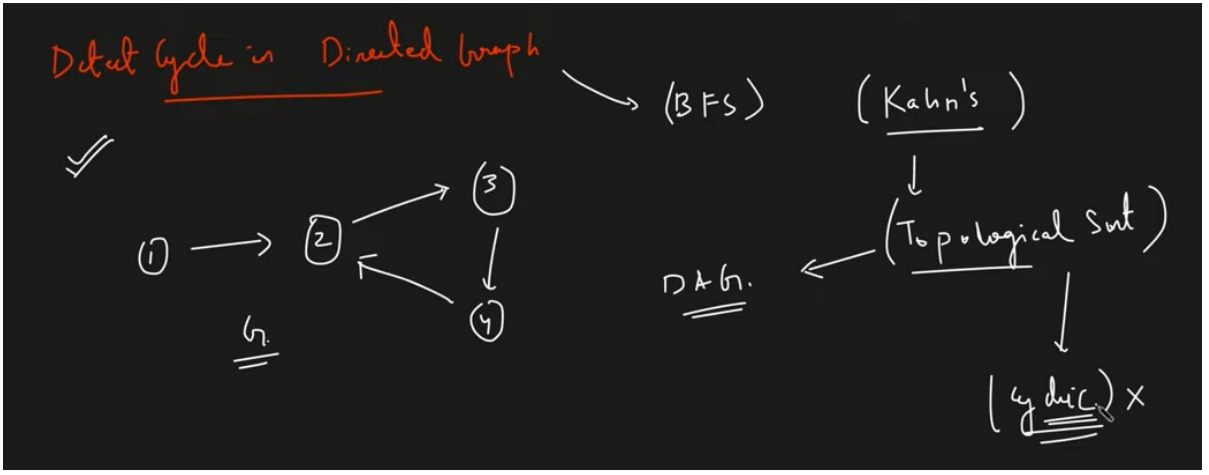
Topological Sort (BFS) | Kahn's Algorithm






```java
class Solution {
    static int[] topoSort(int N, ArrayList<ArrayList<Integer>> adj) {
        int topo[] = new int[N];
        int indegree[] = new int[N];
        for(int i = 0;i<N;i++) {
            for(Integer it: adj.get(i)) {
                indegree[it]++;
            }
        }

        Queue<Integer> q = new LinkedList<Integer>();
        for(int i = 0;i<N;i++) {
            if(indegree[i] == 0) {
                q.add(i);
            }
        }
        int ind = 0;
        while(!q.isEmpty()) {
            Integer node = q.poll();
            topo[ind++] = node;

            for(Integer it: adj.get(node)) {
                indegree[it]--;
                if(indegree[it] == 0) {
                    q.add(it);
                }
            }
        }
        return topo;
    }
```

Cycle Detection in Directed Graph using BFS(Kahn's Algorithm)




```java
import java.util.*;
import java.io.*;
import java.lang.*;

class DriverClass{
    public static void main (String[] args) {
        Scanner sc = new Scanner(System.in);
        int t = sc.nextInt();

        while(t-- > 0){
            ArrayList<ArrayList<Integer>> list = new ArrayList<>();
            int V = sc.nextInt();
            int E = sc.nextInt();
            for(int i = 0; i < V+1; i++)
                list.add(i, new ArrayList<Integer>());
            for(int i = 0; i < E; i++){
                int u = sc.nextInt();
                int v = sc.nextInt();
                list.get(u).add(v);
            }
            if(new Solution().isCyclic(V, list) == true)
                System.out.println("1");
            else System.out.println("0");
        }
    }
}
```

```java
class Solution {
    public boolean isCyclic(int N, ArrayList<ArrayList<Integer>> adj) {
        int topo[] = new int[N];
        int indegree[] = new int[N];
        for(int i = 0;i<N;i++) {
            for(Integer it: adj.get(i)) {
                indegree[it]++;
            }
        }

        Queue<Integer> q = new LinkedList<Integer>();
        for(int i = 0;i<N;i++) {
            if(indegree[i] == 0) {
                q.add(i);
            }
        }
        int cnt = 0;
        while(!q.isEmpty()) {
            Integer node = q.poll();
            cnt++;
            for(Integer it: adj.get(node)) {
                indegree[it]--;
                if(indegree[it] == 0) {
                    q.add(it);
                }
            }
        }
        if(cnt == N) return false;
        return true;
    }
}
```

Number of islands (Do in Grid and Graph both)

200. Number of Islands

Medium 11092 277 Add to List Share

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:

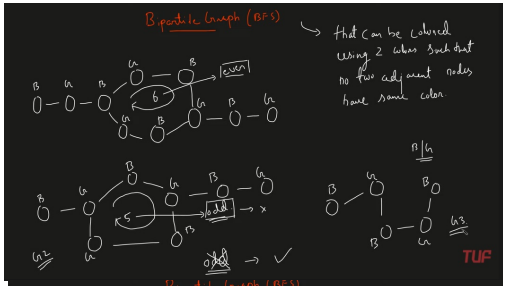
```
Input: grid = [
  ["1","1","1","1","0"],
  ["1","1","0","1","0"],
  ["1","1","0","0","0"],
  ["0","0","0","0","0"]
]
Output: 1
```

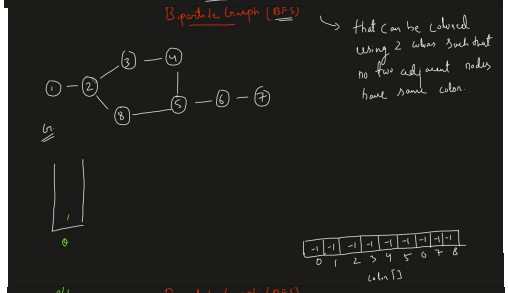
Example 2:

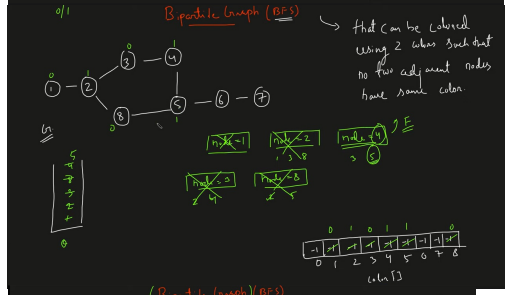
```
Input: grid = [
  ["1","1","0","0","0"],
  ["1","1","0","0","0"],
  ["0","0","1","0","0"],
  ["0","0","0","1","1"]
]
Output: 3
```

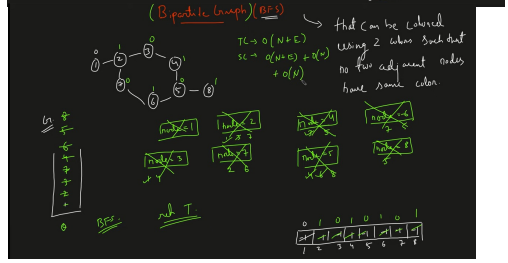
```java
class Solution {
    public int numIslands(char[][] grid) {
        int[][] visited=new int[grid.length][grid[0].length];
        int cnt=0;
        for(int i=0;i<grid.length;i++){
            for(int j=0;j<grid[0].length;j++){
                if(visited[i][j]==0 && grid[i][j]=='1'){
                    dfs(grid,i,j,visited);
                    cnt++;
                }
            }
        }
        return cnt;
    }


    public void dfs(char[][] grid,int i,int j, int[][] visited){
        if(i<0 || j<0 || i>=grid.length || j>=grid[0].length || visited[i][j] !=0 || grid[i][j]=='0') return;

            visited[i][j]=1;

            dfs(grid,i-1,j,visited);
            dfs(grid,i+1,j,visited);
            dfs(grid,i,j-1,visited);
            dfs(grid,i,j+1,visited);
    }
}
```

Bipartite Graph (BFS) | Graph Coloring










```java
import java.util.*;

class Main{
  boolean bfsCheck(ArrayList<ArrayList<Integer>> adj, int node, int color[]) {
    Queue<Integer> q = new LinkedList<>();
    q.add(node);
    color[node] = 1;
    while(!q.isEmpty()) {
      Integer nde = q.poll();

      for(Integer it: adj.get(nde)) {
        if(color[it] == -1) {
          color[it] = 1 - color[nde];
          q.add(it);
        }
        else if(color[it] == color[nde]) {
          return false;
        }
      }
    }
    return true;
  }


    boolean checkBipartite(ArrayList<ArrayList<Integer>> adj, int n){
        int color[] = new int[n];
        for(int i = 0;i<n;i++) {
          color[i] = -1;
        }
        for(int i = 0;i<n;i++) {
          if(color[i] == -1) {
            if(!bfsCheck(adj, i, color)) {
              return false;
            }
          }
        }
        return true;
    }

    public static void main(String args[]){
      int n = 7;
      ArrayList<ArrayList<Integer> > adj = new ArrayList<ArrayList<Integer> >();

      for (int i = 0; i < n; i++)
        adj.add(new ArrayList<Integer>());

      adj.get(0).add(1);
      adj.get(1).add(0);

      adj.get(1).add(2);
      adj.get(2).add(1);

      adj.get(2).add(3);
      adj.get(3).add(2);

      adj.get(4).add(3);
      adj.get(3).add(4);

      adj.get(4).add(5);
      adj.get(5).add(4);

      adj.get(4).add(6);
      adj.get(6).add(4);

      adj.get(1).add(6);
      adj.get(6).add(1);


      Main obj = new Main();
      if(obj.checkBipartite(adj, n) == true) System.out.println("Yes Bipartite");
      else System.out.println("Not Bipartite");
    }
}
```

Bipartite Graph (BFS) | Graph Coloring

785. Is Graph Bipartite?

Medium 3437 249 Add to List Share

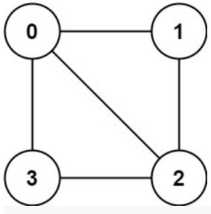
There is an **undirected** graph with `n` nodes, where each node is numbered between `0` and `n - 1`. You are given a 2D array `graph`, where `graph[u]` is an array of nodes that node `u` is adjacent to. More formally, for each `v` in `graph[u]`, there is an undirected edge between node `u` and node `v`. The graph has the following properties:

- There are no self-edges (`graph[u]` does not contain `u`).
- There are no parallel edges (`graph[u]` does not contain duplicate values).
- If `v` is in `graph[u]`, then `u` is in `graph[v]` (the graph is undirected).
- The graph may not be connected, meaning there may be two nodes `u` and `v` such that there is no path between them.

A graph is **bipartite** if the nodes can be partitioned into two independent sets `A` and `B` such that **every** edge in the graph connects a node in set `A` and a node in set `B`.

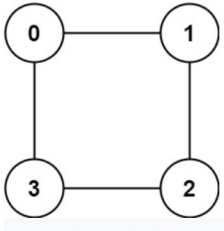
Return `true` *if and only if it is* ***bipartite***.

Example 1:




```
Input: graph = [[1,2,3],[0,2],[0,1,3],[0,2]]
Output: false
Explanation: There is no way to partition the nodes into two independent sets such that
every edge connects a node in one and a node in the other.
```

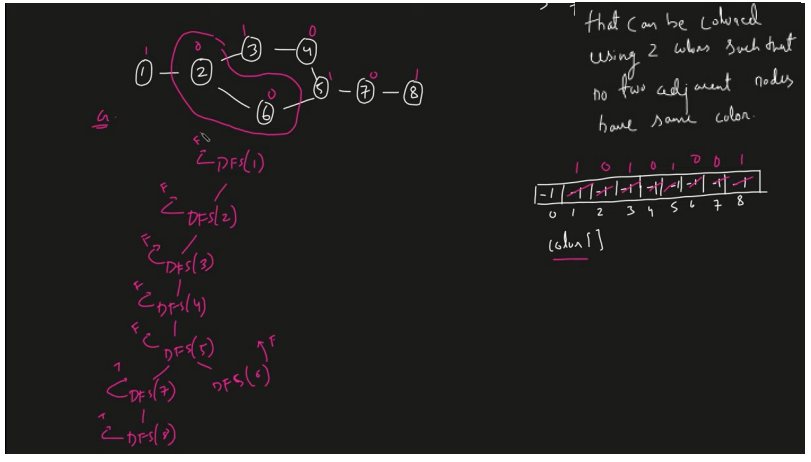
Example 2:

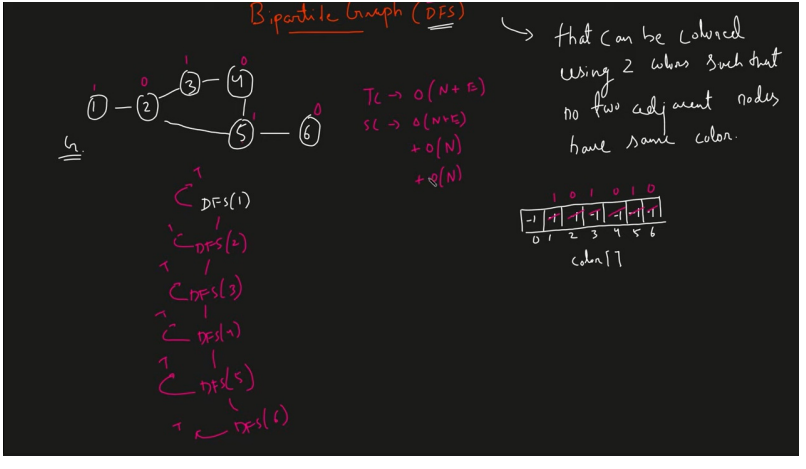



```
Input: graph = [[1,3],[0,2],[1,3],[0,2]]
Output: true
Explanation: We can partition the nodes into two sets: {0, 2} and {1, 3}.
```

```java
class Solution {
    public boolean isBipartite(int[][] graph, int node, int[] color) {
        Queue<Integer> q = new ArrayDeque<>();
        q.add(node);
        color[node] = 1;

        while (!q.isEmpty()) {
            int rem = q.remove();

            for (int nbr : graph[rem]) {
                if (color[nbr] == -1) {
                    color[nbr] = 1 - color[rem];
                    q.add(nbr);
                } else if (color[nbr] == color[rem]) {
                    return false;
                }
            }
        }
        return true;
    }
    public boolean isBipartite(int[][] graph) {
        int color[] = new int[graph.length];

        for (int i = 0; i < color.length; i++) {
            color[i] = -1;
        }

        for (int i = 0; i < graph.length; i++) {
            if (color[i] == -1) {
                if (!isBipartite(graph, i, color)) {
                    return false;
                }
            }
        }
        return true;
    }
}
```

Bipartite Graph (DFS) | Graph Coloring






```java
import java.util.*;

class Main{
  boolean dfsCheck(ArrayList<ArrayList<Integer>> adj, int node, int color[]) {
    for (Integer it : adj.get(node)) {
      if (color[it] == -1) {
        color[it] = 1 - color[node];

        if (!dfsCheck(adj, it, color))
          return false;
      }
      else if (color[it] == color[node]) {
        return false;
      }
    }
    return true;
  }

  boolean checkBipartite(ArrayList<ArrayList<Integer>> adj, int n){
    int color[] = new int[n];
    for (int i = 0; i < n; i++) {
      color[i] = -1;
    }
    for (int i = 0; i < n; i++) {
      if (color[i] == -1) {
        if (!dfsCheck(adj, i, color)) {
          return false;
        }
      }
    }
    return true;
  }
  public static void main(String args[]){
    int n = 7;
    ArrayList<ArrayList<Integer> > adj = new ArrayList<ArrayList<Integer> >();

    for (int i = 0; i < n; i++)
      adj.add(new ArrayList<Integer>());

    adj.get(0).add(1);
    adj.get(1).add(0);

    adj.get(1).add(2);
    adj.get(2).add(1);

    adj.get(2).add(3);
    adj.get(3).add(2);

    adj.get(4).add(3);
    adj.get(3).add(4);

    adj.get(4).add(5);
    adj.get(5).add(4);

    adj.get(4).add(6);
    adj.get(6).add(4);

    adj.get(1).add(6);
    adj.get(6).add(1);


    Main obj = new Main();
    if (obj.checkBipartite(adj, n) == true) System.out.println("Yes Bipartite");
    else System.out.println("Not Bipartite");
  }
}
```

Bipartite Graph (DFS) | Graph Coloring

785. Is Graph Bipartite?

Medium 3437 249 Add to List Share

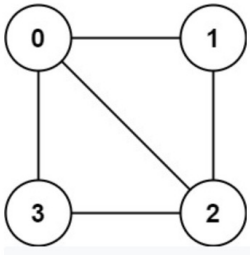
There is an **undirected** graph with `n` nodes, where each node is numbered between `0` and `n - 1`. You are given a 2D array `graph`, where `graph[u]` is an array of nodes that node `u` is adjacent to. More formally, for each `v` in `graph[u]`, there is an undirected edge between node `u` and node `v`. The graph has the following properties:

- There are no self-edges (`graph[u]` does not contain `u`).
- There are no parallel edges (`graph[u]` does not contain duplicate values).
- If `v` is in `graph[u]`, then `u` is in `graph[v]` (the graph is undirected).
- The graph may not be connected, meaning there may be two nodes `u` and `v` such that there is no path between them.

A graph is **bipartite** if the nodes can be partitioned into two independent sets `A` and `B` such that **every** edge in the graph connects a node in set `A` and a node in set `B`.

Return `true` *if and only if it is* ***bipartite***.

Example 1:



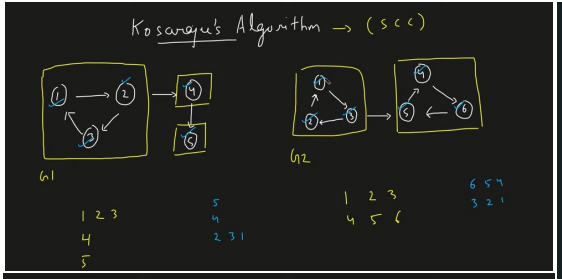

```
Input: graph = [[1,2,3],[0,2],[0,1,3],[0,2]]
Output: false
Explanation: There is no way to partition the nodes into two
independent sets such that every edge connects a node in one and a
node in the other.
```

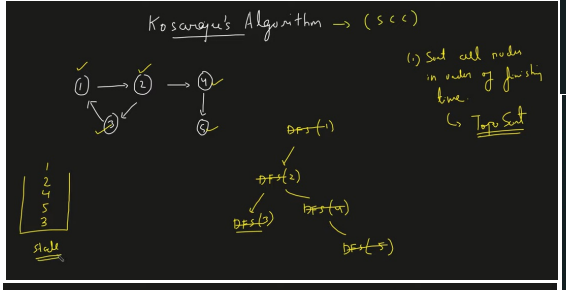
```java
class Solution {
    public boolean isBipartite(int[][] graph) {
        // Array representing the colors
        int[] colors = new int[graph.length];

        // DFS of each node
        for(int i=0; i<graph.length; i++) {
            // If uncolored, then perform DFS
            if(colors[i] == 0 && !hasEvenCycle(graph, colors, i, 1))
                return false;
        }

        return true;
    }

    // Return true when graph is bipartite
    public boolean hasEvenCycle(int[][] g, int[] colors, int node, int c) {
        // if node is colored, node color is same as sent in func definition, return true
        if(colors[node] != 0)
            return colors[node] == c;

        // Color the current node with color sent in func definition
        colors[node] = c;

        // Check for all the adjecent nodes of the current node "node"
        for(int n : g[node]) {
            if(!hasEvenCycle(g, colors, n, -c))
                return false;
        }

        return true;

    }
}
```

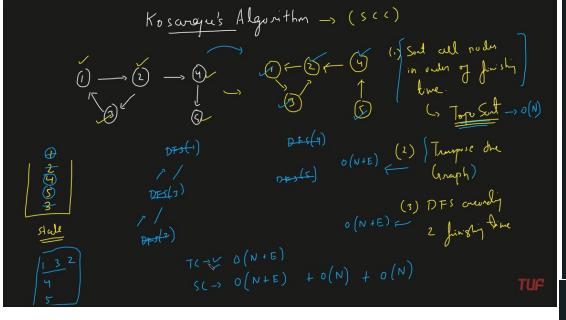
Colors:
0 -> uncolored
1 -> red
-1 -> blue

Graphs - Part 2

Kosaraju's Algorithm for Strongly Connected Components (SCC)



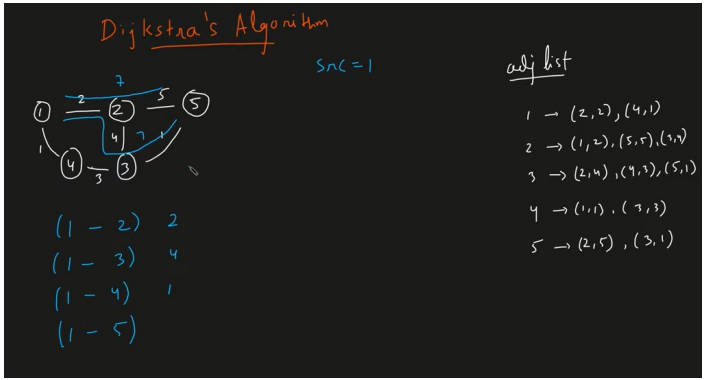


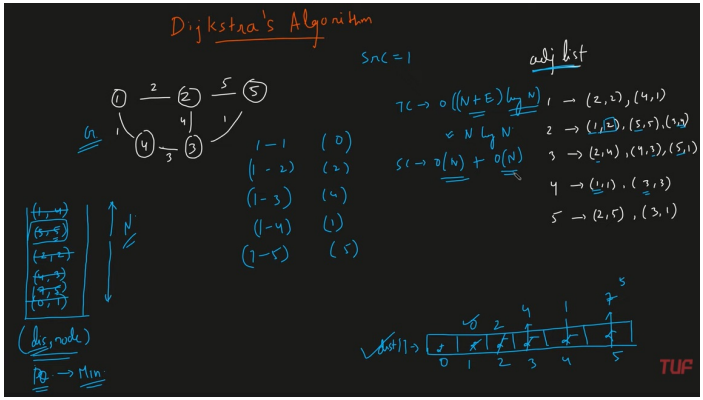



```java
import java.util.*;

class Main {
  private void dfs(int node, Stack<Integer> st, ArrayList<ArrayList<Integer>> adj, int vis[]) {
    vis[node] = 1;
    for (Integer it : adj.get(node)) {
      if (vis[it] == 0) {
        dfs(it, st, adj, vis);
      }
    }

    st.push(node);
  }

  private void revDfs(int node, ArrayList<ArrayList<Integer>> transpose, int vis[]) {
    vis[node] = 1;
    System.out.print(node + " ");
    for (Integer it : transpose.get(node)) {
      if (vis[it] == 0) {
        revDfs(it, transpose, vis);
      }
    }
  }

  void kosaRaju(ArrayList<ArrayList<Integer>> adj, int n) {
    int vis[] = new int[n];
    Stack<Integer> st = new Stack<Integer>();
    for (int i = 0; i < n; i++) {
      if (vis[i] == 0) {
        dfs(i, st, adj, vis);
      }
    }

    ArrayList<ArrayList<Integer> > transpose = new ArrayList<ArrayList<Integer> >();

    for (int i = 0; i < n; i++)
      transpose.add(new ArrayList<Integer>());

    for (int i = 0; i < n; i++) {
      vis[i] = 0;
      for (Integer it : adj.get(i)) {
        transpose.get(it).add(i);
      }
    }

    while (st.size() > 0) {
      int node = st.peek();
      st.pop();
      if (vis[node] == 0) {
        System.out.print("SCC: ");
        revDfs(node, transpose, vis);
        System.out.println();
      }
    }

  }
  public static void main(String args[]) {
    int n = 5;
    ArrayList<ArrayList<Integer> > adj = new ArrayList<ArrayList<Integer> >();
    for (int i = 0; i < n; i++)
      adj.add(new ArrayList<Integer>());

    adj.get(0).add(1);
    adj.get(1).add(2);
    adj.get(2).add(0);
    adj.get(1).add(3);
    adj.get(3).add(4);

    Main obj = new Main();
    obj.kosaRaju(adj, n);
  }
}
```

```
/*
5 5
0 1
1 2
2 0
1 3
3 4
*/
```

Dijkstra's Algorithm | Shortest Path in Undirected Graphs






```java
import java.util.*;

class Node implements Comparator<Node>{
  private int v;
  private int weight;

  Node(int _v, int _w) {
    v = _v;
    weight = _w;
  }

  Node() {}
  int getV() {
    return v;
  }
  int getWeight() {
    return weight;
  }

  @Override
  public int compare(Node node1, Node node2){
    if (node1.weight < node2.weight)
      return -1;
    if (node1.weight > node2.weight)
      return 1;
    return 0;
  }
}
```

```java
public static void main(String args[]){
  int n = 5;
  ArrayList<ArrayList<Node> > adj = new ArrayList<ArrayList<Node> >();

  for (int i = 0; i < n; i++)
    adj.add(new ArrayList<Node>());

  adj.get(0).add(new Node(1, 2));
  adj.get(1).add(new Node(0, 2));

  adj.get(1).add(new Node(2, 4));
  adj.get(2).add(new Node(1, 4));

  adj.get(0).add(new Node(3, 1));
  adj.get(3).add(new Node(0, 1));

  adj.get(3).add(new Node(2, 3));
  adj.get(2).add(new Node(3, 3));

  adj.get(1).add(new Node(4, 5));
  adj.get(4).add(new Node(1, 5));

  adj.get(2).add(new Node(4, 1));
  adj.get(4).add(new Node(2, 1));

  Main obj = new Main();
  obj.shortestPath(0, adj, n);
}
}
```

```java
class Main{
  void shortestPath(int s, ArrayList<ArrayList<Node>> adj, int N){
    int dist[] = new int[N];

    for (int i = 0; i < N; i++) dist[i] = 100000000;
    dist[s] = 0;

    PriorityQueue<Node> pq = new PriorityQueue<Node>(N, new Node());
    pq.add(new Node(s, 0));

    while (pq.size() > 0) {
      Node node = pq.poll();

      for (Node it : adj.get(node.getV())) {
        if (dist[node.getV()] + it.getWeight() < dist[it.getV()]) {
          dist[it.getV()] = dist[node.getV()] + it.getWeight();
          pq.add(new Node(it.getV(), dist[it.getV()]));
        }
      }
    }

    for (int i = 0; i < N; i++){
      System.out.print( dist[i] + " ");
    }
  }
```

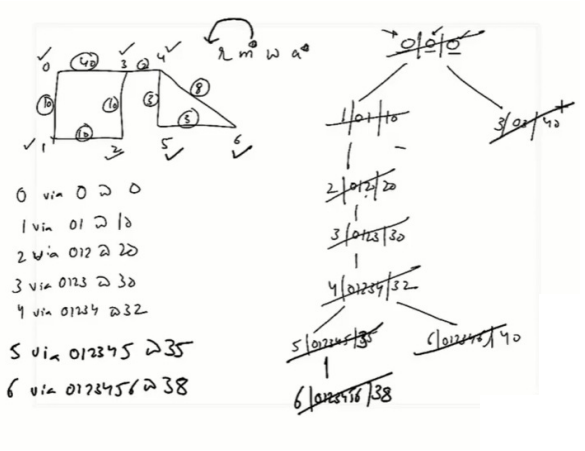
Shortest Path In Weights

Easy

Prev Next

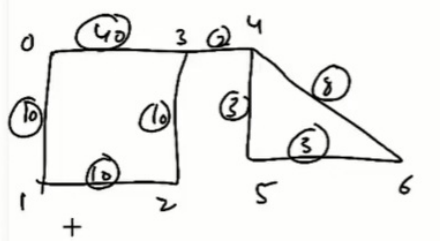
1. You are given a graph and a source vertex. The vertices represent cities and the edges represent
   distance in kms.
2. You are required to find the shortest path to each city (in terms of kms) from the source city along
   with the total distance on path from source to destinations.

Note -> For output, check the sample output and question video.



Sample Input

```
7
9
0 1 10
1 2 10
2 3 10
0 3 40
3 4 2
4 5 3
5 6 3
4 6 8
2 5 5
0
```



Sample output

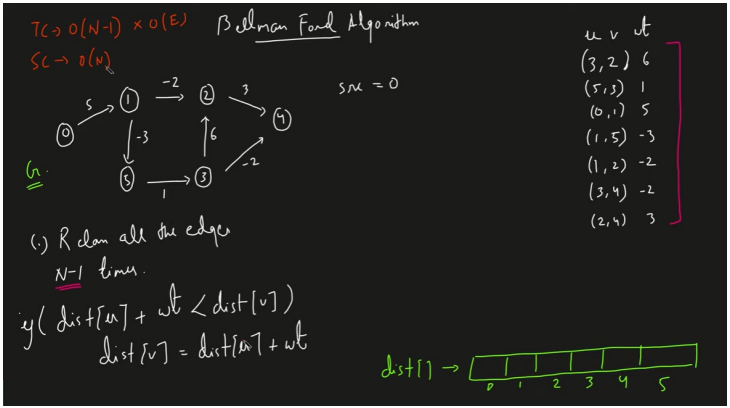
```
0 via 0 @ 0
1 via 01 @ 10
2 via 012 @ 20
5 via 0125 @ 25
4 via 01254 @ 28
6 via 01256 @ 28
3 via 012543 @ 30
```

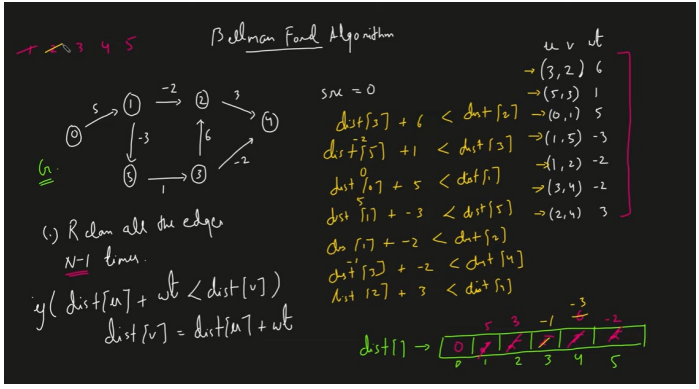
```java
import java.io.*;
import java.util.*;

public class Main {
    static class Edge {
        int src;
        int nbr;
        int wt;

        Edge(int src, int nbr, int wt) {
            this.src = src;
            this.nbr = nbr;
            this.wt = wt;
        }
    }

    static class Pair implements Comparable<Pair> {
        int v;
        String psf;
        int wsf;

        Pair(int v, String psf, int wsf){
            this.v = v;
            this.psf = psf;
            this.wsf = wsf;
        }

        public int compareTo(Pair o){
            return this.wsf - o.wsf;
        }
    }
}
```

```java
public static void main(String[] args) throws Exception {
    BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

    int vtces = Integer.parseInt(br.readLine());
    ArrayList<Edge>[] graph = new ArrayList[vtces];
    for (int i = 0; i < vtces; i++) {
        graph[i] = new ArrayList<>();
    }

    int edges = Integer.parseInt(br.readLine());
    for (int i = 0; i < edges; i++) {
        String[] parts = br.readLine().split(" ");
        int v1 = Integer.parseInt(parts[0]);
        int v2 = Integer.parseInt(parts[1]);
        int wt = Integer.parseInt(parts[2]);
        graph[v1].add(new Edge(v1, v2, wt));
        graph[v2].add(new Edge(v2, v1, wt));
    }

    int src = Integer.parseInt(br.readLine());

    PriorityQueue<Pair> queue = new PriorityQueue<>();
    queue.add(new Pair(src, src + "", 0));
    boolean[] visited = new boolean[vtces];
    while(queue.size() > 0){
        Pair rem = queue.remove();

        if(visited[rem.v] == true){
            continue;
        }
        visited[rem.v] = true;
        System.out.println(rem.v + " via " + rem.psf + " @ " + rem.wsf);

        for (Edge e : graph[rem.v]) {
            if (visited[e.nbr] == false) {
                queue.add(new Pair(e.nbr, rem.psf + e.nbr, rem.wsf + e.wt));
            }
        }
    }
}
```

Bellman Ford Algorithm | Detect Negative Weight Cycle in Graphs

```java
import java.util.*;

class Node {
    private int u;
    private int v;
    private int weight;

    Node(int _u, int _v, int _w) { u = _u; v = _v; weight = _w; }

    Node() {}

    int getV() { return v; }
    int getU() { return u; }
    int getWeight() { return weight; }
}
```

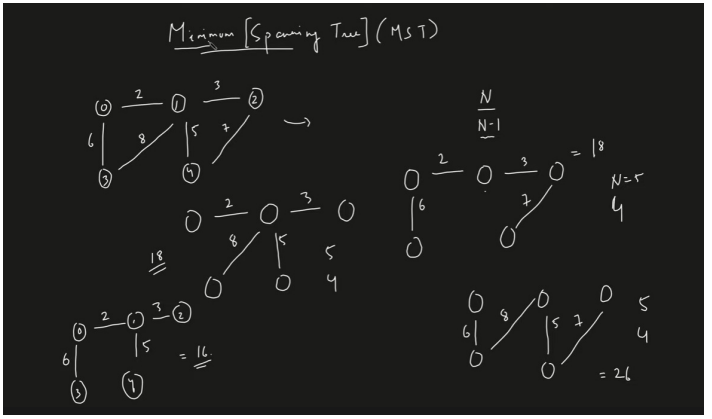


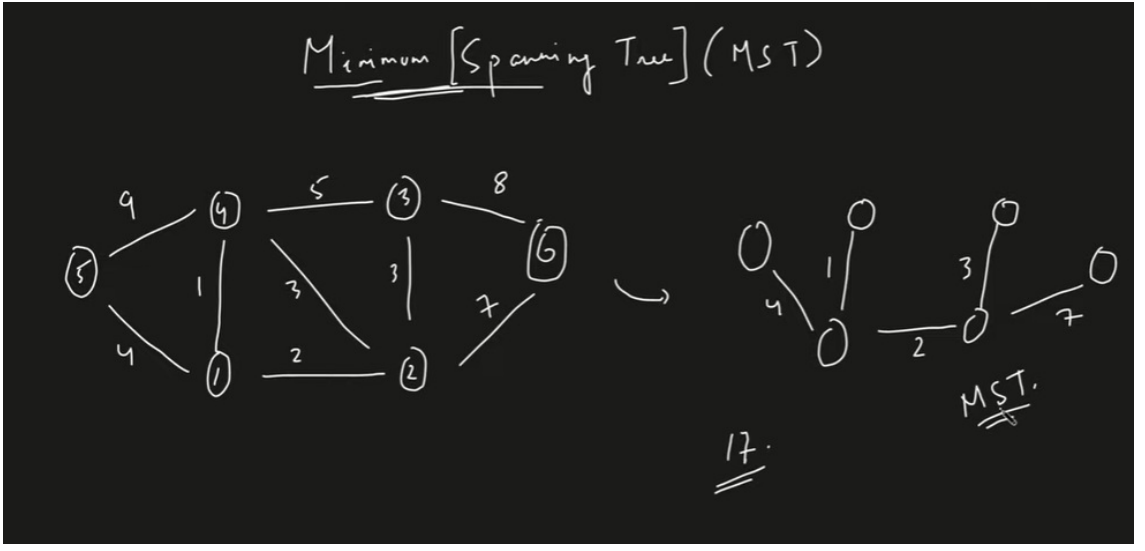



```java
class Main{
    void bellmanFord(ArrayList<Node> edges, int N, int src){
        int dist[] = new int[N];

        for(int i = 0;i<N;i++) dist[i] = 10000000;

        dist[src] = 0;
        for(int i = 1;i<=N-1;i++) {
            for(Node node : edges) {
                if(dist[node.getU()] + node.getWeight() < dist[node.getV()]) {
                    dist[node.getV()] = dist[node.getU()] + node.getWeight();
                }
            }
        }

        int fl = 0;
        for(Node node: edges) {
            if(dist[node.getU()] + node.getWeight() < dist[node.getV()]) {
                fl = 1;
                System.out.println("Negative Cycle");
                break;
            }
        }

        if(fl == 0) {
            for(int i = 0;i<N;i++) {
                System.out.println(i + " " + dist[i]);
            }
        }
    }
    public static void main(String args[]){
        int n = 6;
        ArrayList<Node> adj = new ArrayList<Node>();

        adj.add(new Node(3, 2, 6));
        adj.add(new Node(5, 3, 1));
        adj.add(new Node(0, 1, 5));
        adj.add(new Node(1, 5, -3));
        adj.add(new Node(1, 2, -2));
        adj.add(new Node(3, 4, -2));
        adj.add(new Node(2, 4, 3));

        Main obj = new Main();
        obj.bellmanFord(adj, n, 0);
    }
}
```

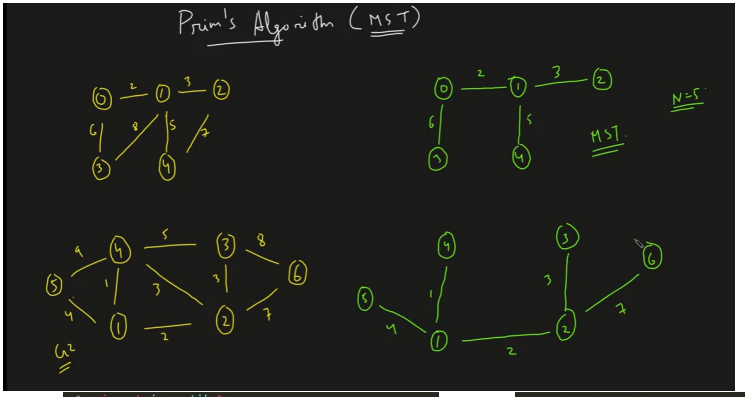
Floyd Warshall Algorithm

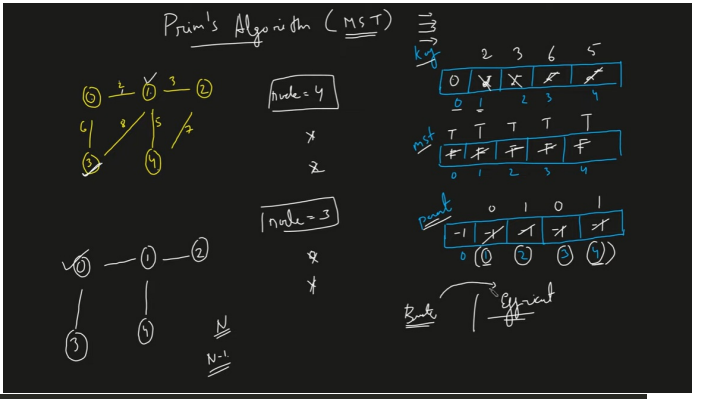
Minimum Spanning Tree Explanation





Prims Algorithm | Minimum Spanning Tree






```java
import java.util.*;

class Node implements Comparator<Node>{
    private int v;
    private int weight;

    Node(int _v, int _w) { v = _v; weight = _w; }

    Node() {}

    int getV() { return v; }
    int getWeight() { return weight; }

    @Override
    public int compare(Node node1, Node node2){
        if (node1.weight < node2.weight)
            return -1;
        if (node1.weight > node2.weight)
            return 1;
        return 0;
    }
}

```

```java
    public static void main(String args[]){

      int n = 5;
      ArrayList<ArrayList<Node> > adj = new ArrayList<ArrayList<Node> >();

      for (int i = 0; i < n; i++)
        adj.add(new ArrayList<Node>());

      adj.get(0).add(new Node(1, 2));
      adj.get(1).add(new Node(0, 2));

      adj.get(1).add(new Node(2, 3));
      adj.get(2).add(new Node(1, 3));

      adj.get(0).add(new Node(3, 6));
      adj.get(3).add(new Node(0, 6));

      adj.get(1).add(new Node(3, 8));
      adj.get(3).add(new Node(1, 8));

      adj.get(1).add(new Node(4, 5));
      adj.get(4).add(new Node(1, 5));

      adj.get(2).add(new Node(4, 7));
      adj.get(4).add(new Node(2, 7));

      Main obj = new Main();
      obj.primsAlgo(adj, n);
    }
}
```

```java
class Main {
    void primsAlgo(ArrayList<ArrayList<Node>> adj, int N){
        int key[] = new int[N];
        int parent[] = new int[N];
        boolean mstSet[] = new boolean[N];
        for(int i = 0;i<N;i++) {
          key[i] = 100000000;
          mstSet[i] = false;
        }

        PriorityQueue<Node> pq = new PriorityQueue<Node>(N, new Node());
        key[0] = 0;
        parent[0] = -1;
        pq.add(new Node(key[0], 0));
      // Run the loop till all the nodes have been visited
        // because in the brute code we checked for mstSet[node] == false while computing the minimum
        // but here we simply take the minimal from the priority queue, so a lot of times a node might be taken twice
        // hence its better to keep running till all the nodes have been taken.
        // try the following case:
        // Credits: Srejan Bera
        // 6 7
        // 0 1 5
        // 0 2 10
        // 0 3 100
        // 1 3 50
        // 1 4 200
        // 3 4 250
        // 4 5 50
        while(!pq.isEmpty()) {
          int u = pq.poll().getV();
          mstSet[u] = true;

          for(Node it: adj.get(u)) {
            if(mstSet[it.getV()] == false && it.getWeight() < key[it.getV()]) {
              parent[it.getV()] = u;
              key[it.getV()] = it.getWeight();
              pq.add(new Node(it.getV(), key[it.getV()]));
            }
          }
        }
        for(int i = 1;i<N;i++) {
          System.out.println(parent[i] + " - " + i);
        }
    }
```

Disjoint Set | Union By Rank and Path Compression

Kruskal's Algorithm

```java
import java.util.*;

class Node{
  private int u;
  private int v;
  private int weight;

  Node(int _u, int _v, int _w) {
    u = _u;
    v = _v;
    weight = _w;
  }
  Node() {}
  int getV() {
    return v;
  }
  int getU() {
    return u;
  }
  int getWeight() {
    return weight;
  }
}

class SortComparator implements Comparator<Node> {
  @Override
  public int compare(Node node1, Node node2){
    if (node1.getWeight() < node2.getWeight())
      return -1;
    if (node1.getWeight() > node2.getWeight())
      return 1;
    return 0;
  }
}
```

```java
  public static void main(String args[]){
    int n = 5;
    ArrayList<Node> adj = new ArrayList<Node>();

    adj.add(new Node(0, 1, 2));
    adj.add(new Node(0, 3, 6));
    adj.add(new Node(1, 3, 8));
    adj.add(new Node(1, 2, 3));
    adj.add(new Node(1, 4, 5));
    adj.add(new Node(2, 4, 7));

    Main obj = new Main();
    obj.KruskalAlgo(adj, n);
  }
}
```

```java
class Main{
  private int findPar(int u, int parent[]) {
    if (u == parent[u]) return u;
    return parent[u] = findPar(parent[u], parent);
  }
  private void union(int u, int v, int parent[], int rank[]) {
    u = findPar(u, parent);
    v = findPar(v, parent);
    if (rank[u] < rank[v]) {
      parent[u] = v;
    }
    else if (rank[v] < rank[u]) {
      parent[v] = u;
    }
    else {
      parent[v] = u;
      rank[u]++;
    }
  }
  void KruskalAlgo(ArrayList<Node> adj, int N){
    Collections.sort(adj, new SortComparator());
    int parent[] = new int[N];
    int rank[] = new int[N];

    for (int i = 0; i < N; i++) {
      parent[i] = i;
      rank[i] = 0;
    }

    int costMst = 0;
    ArrayList<Node> mst = new ArrayList<Node>();
    for (Node it : adj) {
      if (findPar(it.getU(), parent) != findPar(it.getV(), parent)) {
        costMst += it.getWeight();
        mst.add(it);
        union(it.getU(), it.getV(), parent, rank);
      }
    }
    System.out.println(costMst);
    for (Node it : mst) {
      System.out.println(it.getU() + " " + it.getV());
    }
  }
  public static void main(String args[]){
```

Day-25: Dynamic Programming – Part 1

Max Product Subarray || Leetcode

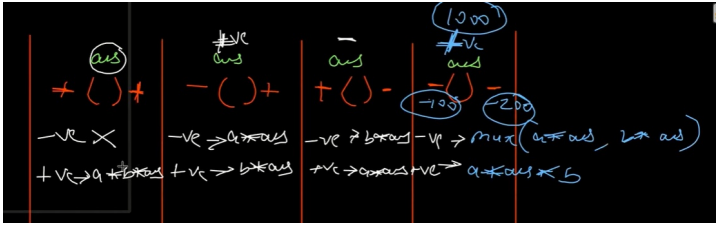
152. Maximum Product Subarray

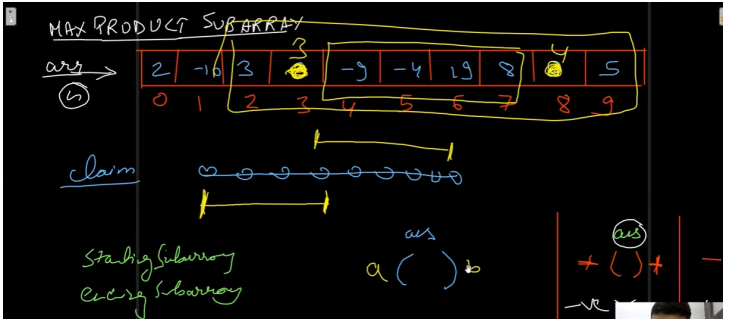
Medium 8954 279 Add to List Share

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

It is **guaranteed** that the answer will fit in a **32-bit** integer.

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





Example 1:

```
Input: nums = [2,3,-2,4]
Output: 6
Explanation: [2,3] has the largest product 6.
```

Example 2:

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

```java
class Solution {
    public int maxProduct(int[] nums) {
        int ans = Integer.MIN_VALUE;
        int cprod = 1;
        for(int i=0; i<nums.length; i++){
            cprod *= nums[i];
            ans = Math.max(ans,cprod);
            if(cprod == 0){
                cprod = 1;
            }
        }
        cprod = 1;
        for(int i=nums.length-1; i>=0; i--){
            cprod *= nums[i];
            ans = Math.max(ans,cprod);
            if(cprod == 0){
                cprod = 1;
            }
        }
        return ans;
    }
}
```

Longest Increasing Subsequence | Dynamic Programming

300. Longest Increasing Subsequence

Medium 9634 197 Add to List Share

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

A **subsequence** is a sequence that can be derived from an array by deleting some or no elements without changing the order of the remaining elements. For example, `[3,6,2,7]` is a subsequence of the array `[0,3,1,6,2,2,7]`.

Example 1:

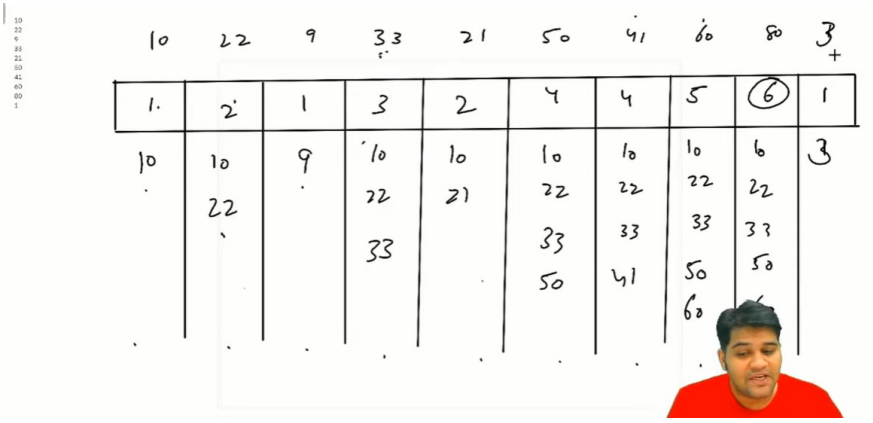
```
Input: nums = [10,9,2,5,3,7,101,18]
Output: 4
Explanation: The longest increasing subsequence is [2,3,7,101],
therefore the length is 4.
```

Example 2:

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

Example 3:

```
Input: nums = [7,7,7,7,7,7,7]
Output: 1
```




```java
class Solution {
    public int lengthOfLIS(int[] nums) {
        int[] dp = new int[nums.length];
        int omax = 0;
        for(int i=0; i<dp.length; i++){
            int max = 0;
            for(int j=0; j<i; j++){
                if(nums[j]<nums[i]){
                    if(dp[j]>max){
                        max = dp[j];
                    }
                }
            }
            dp[i] = max + 1;
            if(dp[i]>omax) omax = dp[i];
        }
        return omax;
    }
}
```

1143. Longest Common Subsequence

Longest Common Subsequence Dynamic Programming

Medium 4855 56 Add to List Share

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:

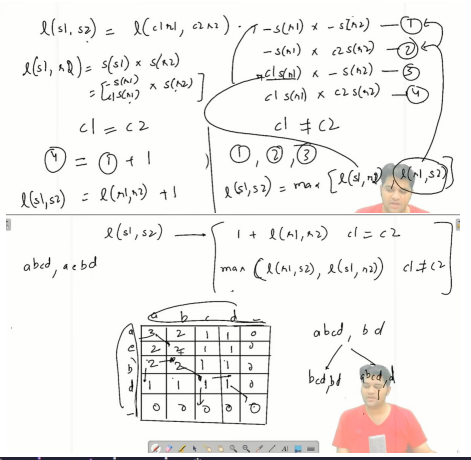
```
Input: text1 = "abcde", text2 = "ace"
Output: 3
Explanation: The longest common subsequence is "ace" and its
length is 3.
```

Example 2:

```
Input: text1 = "abc", text2 = "abc"
Output: 3
Explanation: The longest common subsequence is "abc" and its
length is 3.
```

Example 3:

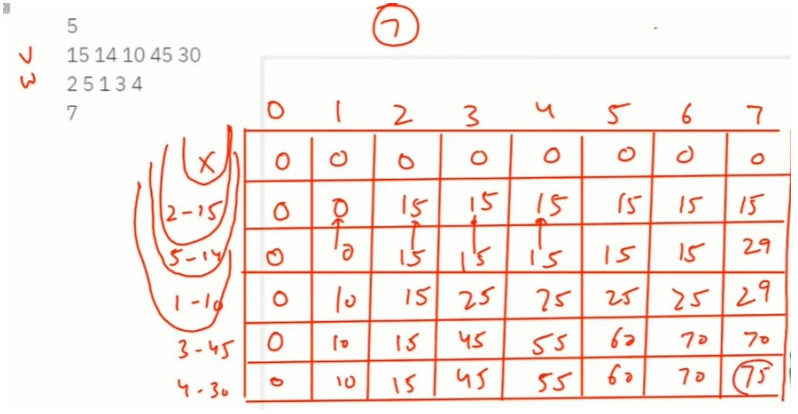
```
Input: text1 = "abc", text2 = "def"
Output: 0
```



```java
import java.io.*;
import java.util.*;

public class Main {
    public static void main(String[] args) throws Exception {
        Scanner scn = new Scanner(System.in);
        String s1 = scn.nextLine();
        String s2 = scn.nextLine();

        int[][] dp = new int[s1.length()+1][s2.length()+1];
        for(int i=dp.length-2; i>=0; i--){
            for(int j=dp[0].length-2; j>=0; j--){
                char c1 = s1.charAt(i);
                char c2 = s2.charAt(j);
                if(c1==c2){
                    dp[i][j] = 1 + dp[i+1][j+1];
                }else{
                    dp[i][j] = Math.max(dp[i+1][j],dp[i][j+1]);
                }
            }
        }
        System.out.println(dp[0][0]);
    }
}
```

0-1 Knapsack

0-1 Knapsack Problem Dynamic Programming




```java
import java.io.*;
import java.util.*;

public class Main {

    public static void main(String[] args) throws Exception {
        Scanner scn = new Scanner(System.in);
        int n = scn.nextInt();
        int[] vals = new int[n];
        int[] wts = new int[n];
        for(int i=0; i<vals.length; i++){
            vals[i] = scn.nextInt();
        }
        for(int i=0; i<wts.length; i++){
            wts[i] = scn.nextInt();
        }
        int caps = scn.nextInt();
        int[][] dp = new int[n+1][caps+1];
        for(int i=1; i<dp.length; i++){
            for(int j=1; j<dp[0].length; j++){

                dp[i][j] = dp[i-1][j];// when i doesnt bats
                if(j>=wts[i-1]){
                    int rcap = j-wts[i-1];
                    if(dp[i-1][rcap]+vals[i-1] > dp[i-1][j]){
                        dp[i][j] = dp[i-1][rcap]+vals[i-1];
                    }else{
                        dp[i][j] = dp[i-1][j];
                    }
                }
            }
        }
        System.out.println(dp[dp.length-1][dp[0].length-1]);
    }
}
```

Print All Results In 0-1 Knapsack

Medium

1. You are given a number n, representing the count of items.
2. You are given n numbers, representing the values of n items.
3. You are given n numbers, representing the weights of n items.
3. You are given a number "cap", which is the capacity of a bag you've.
4. You are required to calculate and print the maximum value that can be created in the bag without overflowing it's capacity.
5. Also, you have to print the indices of items that should be selected to make maximum profit.
6. You have to print all such configurations.

Note -> Each item can be taken 0 or 1 number of times. You are not allowed to put the same item again and again.

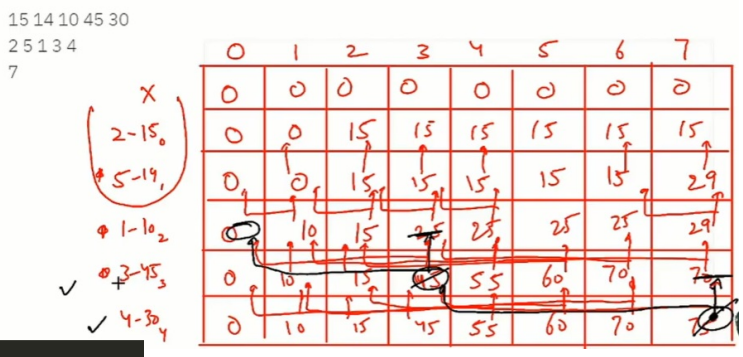
Example

Sample Input

```
5
15 14 10 45 30
2 5 1 3 4
7
```

Sample Output

```
75
3 4
```




```java
import java.io.*;
import java.util.*;

public class Main {
  public static class Pair {
    int i;
    int j;
    String psf;
    Pair(int i, int j, String psf) {
      this.i = i;
      this.j = j;
      this.psf = psf;
    }
  }

  public static void main(String[] args) throws Exception {
    BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
    int n = Integer.parseInt(br.readLine());

    int[] values = new int[n];
    String str1 = br.readLine();
    for (int i = 0; i < n; i++) {
      values[i] = Integer.parseInt(str1.split(" ")[i]);
    }

    int[] wts = new int[n];
    String str2 = br.readLine();
    for (int i = 0; i < n; i++) {
      wts[i] = Integer.parseInt(str2.split(" ")[i]);
    }

    int cap = Integer.parseInt(br.readLine());

    //write your code here
    int[][] dp = new int[values.length + 1][cap + 1];
    for (int i = 1; i < dp.length; i++) {
      for (int j = 1; j < dp[0].length; j++) {
        dp[i][j] = dp[i - 1][j];

        if (j >= wts[i - 1] &&  (dp[i - 1][j - wts[i - 1]] + values[i - 1] > dp[i - 1][j])) {
          dp[i][j] = dp[i - 1][j - wts[i - 1]] + values[i - 1];
        }
      }
    }
    int ans = dp[values.length][cap];
    System.out.println(ans);

    ArrayDeque<Pair> que = new ArrayDeque<>();
    que.add(new Pair(values.length, cap, ""));

    while (que.size() > 0) {
      Pair rem = que.removeFirst();
      if (rem.i == 0 || rem.j == 0) {
        System.out.println(rem.psf);
      } else {
        int exc = dp[rem.i - 1][rem.j];
        if (rem.j >= wts[rem.i - 1]) {
          int inc = dp[rem.i - 1][rem.j - wts[rem.i - 1]] + values[rem.i - 1];
          if (dp[rem.i][rem.j] == inc) {
            que.add(new Pair(rem.i - 1, rem.j - wts[rem.i - 1], (rem.i - 1) + " " + rem.psf));
          }
        }
        if (dp[rem.i][rem.j] == exc) {
          que.add(new Pair(rem.i - 1, rem.j, rem.psf));
        }
      }
    }

  }
}
```

Edit Distance | Recursion | Dynamic Programming

Edit Distance

Hard

1. You are given two strings s1 and s2.
2. You have to find the minimum number of operations needed to convert s1 to s2.
   Operations allowed are -
   Insert - You can insert any character in s1.
   Remove - You can remove any character in s1.
   Replace - You can replace any character in s1 with any other character.

Constraints

1

Format

Input

Two strings s1 and s2

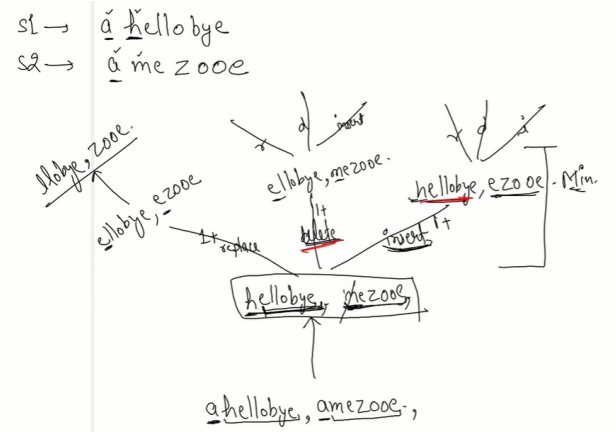
Example

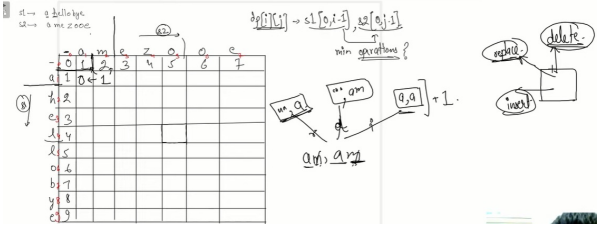
Sample Input

pepperatcoding
pepcoding

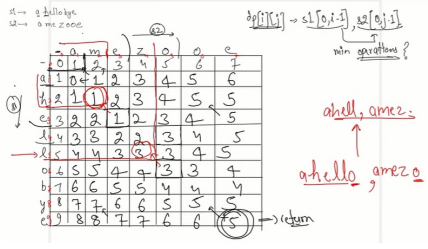
Sample Output

5






```java
import java.io.*;
import java.util.*;

public class Main {

    public static int solution(String s1, String s2) {
        //write your code here
        int[][] dp = new int[s1.length()+1][s2.length()+1];
        for(int i=0; i<dp.length; i++){
            for(int j=0; j<dp[0].length; j++){
                if(i==0){
                    dp[i][j] = j;
                }else if(j==0){
                    dp[i][j] = i;
                }else{
                    if(s1.charAt(i-1)==s2.charAt(j-1)){
                        dp[i][j] = dp[i-1][j-1];
                    }else{
                        dp[i][j] = 1 + Math.min(dp[i-1][j-1],Math.min(dp[i-1][j],dp[i][j-1]));
                    }
                }
            }
        }
        return dp[dp.length-1][dp[0].length-1];
    }

    public static void main(String[] args) {
        Scanner scn = new Scanner(System.in);
        String s1 = scn.next();
        String s2 = scn.next();
        System.out.println(solution(s1,s2));
    }

}
```



Maximum Sum Increasing Subsequence | Dynamic Programming | Explanation and Code

Maximum Sum Increasing Subsequence

Easy

1. You are given a number n, representing the number of elements.
2. You are given n numbers, representing the contents of array of length n.
3. You are required to print the sum of elements of the increasing subsequence with maximum sum for the array.

Example

Sample Input

```
10
10
22
9
33
21
50
41
60
80
1
```

Sample Output

```
255
```

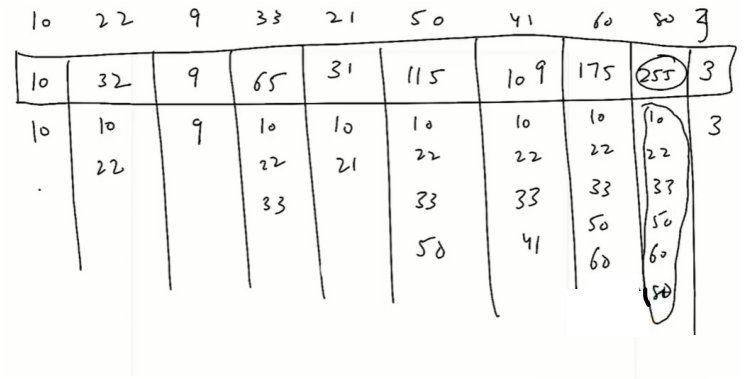
Format

Input

A number n
.. n more elements

Output

A number representing the sum of elements of the increasing subsequence with maximum sum for the array.




```java
import java.io.*;
import java.util.*;

public class Main {
    public static void main(String[] args) {
        Scanner scn = new Scanner(System.in);
        int n = scn.nextInt();
        int[] arr = new int[n];
        for(int i=0; i<n; i++){
            arr[i] = scn.nextInt();
        }

        int omax = Integer.MIN_VALUE;
        int[] dp = new int[n];

        for(int i=0; i<dp.length; i++){
            Integer max = null;
            for(int j=0; j<i; j++){
                if(arr[j]<=arr[i]){
                    if(max==null){
                        max = dp[j];
                    }
                    if(dp[j]>max){
                        max = dp[j];
                    }
                }
            }

            if(max == null) dp[i] = arr[i];
            else dp[i] = arr[i] + max;

            if(dp[i]>omax) omax = dp[i];
        }
        System.out.println(omax);
    }
}
```

Matrix Chain Multiplication Dynamic Programming

Matrix Chain Multiplication

Medium

1. You are given an array(arr) of positive integers of length N which represents the dimensions of N-1 matrices such that the ith matrix is of dimension arr[i-1] x arr[i].
2. You have to find the minimum number of multiplications needed to multiply the given chain of matrices.

Constraints

2

Format

Input

A number N
arr1
arr2.. N integers

Output

Check the sample output and question video.

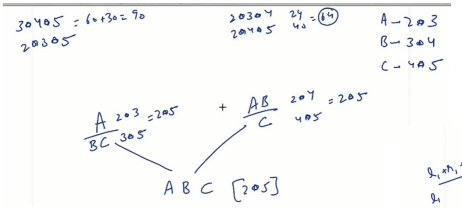
Example

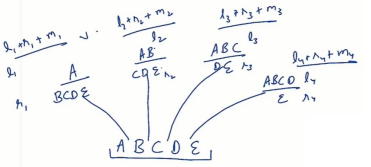
Sample Input

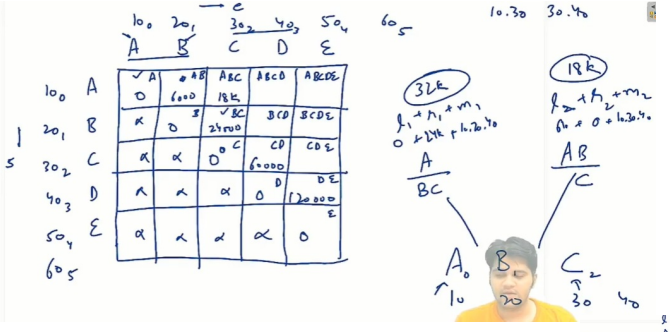
3
1
2
3

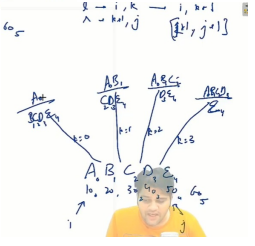
Sample Output

6









```java
import java.io.*;
import java.util.*;
public class Main {
    public static int mcm(int[] arr){
        int[][] dp = new int[arr.length-1][arr.length-1];
        for(int g=0; g<arr.length; g++){
            for(int i=0, j=g; j<dp[0].length; i++,j++){
                if(g==0){
                    dp[i][j] = 0;
                }else if(g==1){
                    dp[i][j] = arr[i]*arr[j]*arr[j+1];
                }else{
                    int min = Integer.MAX_VALUE;
                    for(int k=i; k<j; k++){
                        // dp -> i,k left half and k+1,j right half
                        // arr -> i * k+1  left half and  k+1 * j+1 right half
                        int lc = dp[i][k];
                        int rc = dp[k+1][j];
                        int mc = arr[i] * arr[k+1] * arr[j+1];
                        int tc = lc+rc+mc;
                        if(tc<min) min = tc;
                    }
                    dp[i][j] = min;
                }
            }
        }
        return dp[0][dp.length-1];
    }
    public static void main(String[] args) {
        Scanner scn = new Scanner(System.in);
        int n = scn.nextInt();
        int[] arr = new int[n];
        for(int i = 0 ; i < n; i++){
            arr[i] = scn.nextInt();
        }
        System.out.println(mcm(arr));
    }
}
```

Ones and Zeroes

474. Ones and Zeroes

Medium 2422 310 Add to List Share

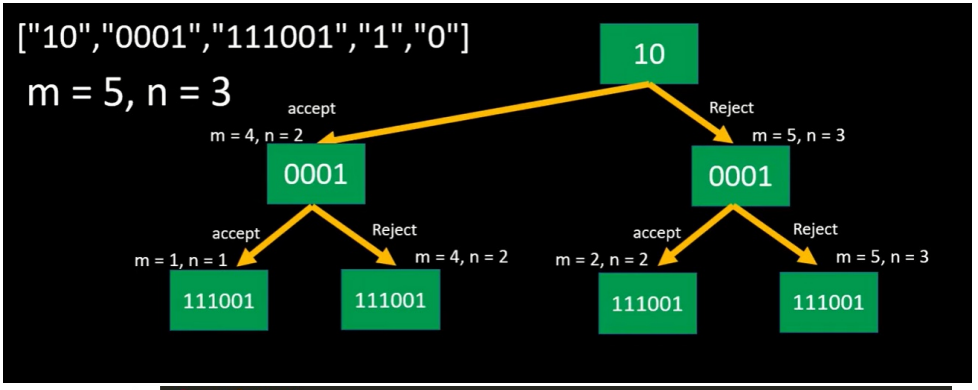
You are given an array of binary strings `strs` and two integers `m` and `n`.

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.
Other valid but smaller subsets include {"0001", "1"} and {"10",
"1", "0"}.
{"111001"} is an invalid subset because it contains 4 1's,
greater than the maximum of 3.
```




```java
class Solution {
    int[][] dp;
    public int findMaxForm(String[] strs, int m, int n) {
        dp= new int[m+1][n+1];
        for(String s:strs){
            int[] count = count(s);
            //zero m-count[0]  ---- 0
            //one n - count[1] ---- 0
            for(int zero=m;zero>=count[0];zero--){
                for(int one=n; one>=count[1]; one--){
                    dp[zero][one] = Math.max(dp[zero-count[0]][one-count[1]] +1 , dp[zero][one]);
                }
            }
        }
        return dp[m][n];
    }
    int[] count(String s){
        int[] count=new int[2];
        for(char c: s.toCharArray()){
            count[c-'0']++;
        }
        return count;
    }
}
```