

## 10. Regular Expression Matching

Hard



10.7K

1.8K



Companies

Given an input string `s` and a pattern `p`, implement regular expression matching with support for `.` and `*` where:

- `.` Matches any single character.
- `*` Matches zero or more of the preceding element.

The matching should cover the **entire** input string (not partial).

### Example 1:

**Input:** `s = "aa", p = "a"`

**Output:** `false`

**Explanation:** "a" does not match the entire string "aa".

### Example 2:

**Input:** `s = "aa", p = "a*"`

**Output:** `true`

**Explanation:** '\*' means zero or more of the preceding element, 'a'. Therefore, by repeating 'a' once, it becomes "aa".

### Example 3:

**Input:** `s = "ab", p = ".*"`

**Output:** `true`

**Explanation:** ".\*" means "zero or more (\*) of any character (.)".

```
1 class Solution {
2     public boolean isMatch(String s, String p) {
3         int m = s.length(), n = p.length();
4         boolean[][] dp = new boolean[m+1][n+1];
5         dp[0][0] = true; // empty pattern matches empty string
6
7         // initialize first row (empty string)
8         for (int j = 1; j <= n; j++) {
9             if (p.charAt(j-1) == '*')
10                dp[0][j] = dp[0][j-2];
11        }
12
13        // fill in remaining cells
14        for (int i = 1; i <= m; i++) {
15            for (int j = 1; j <= n; j++) {
16                if (s.charAt(i-1) == p.charAt(j-1) || p.charAt(j-1) == '.') {
17                    dp[i][j] = dp[i-1][j-1];
18                } else if (p.charAt(j-1) == '*') {
19                    // Two cases:
20                    // 1. Previous character in p is not a special character
21                    // e.g. "ab*cd*" vs "abdc" where * is matching "c"
22                    dp[i][j] = dp[i][j-2];
23                    // 2. Previous character in p is a special character
24                    // e.g. "ab*" vs "ab" where * is matching "a"
25                    dp[i][j] = dp[i][j-2] || (p.charAt(j-2) == s.charAt(i-1) || p.charAt(j-2) == '.' && dp[i-1][j]);
26                }
27            }
28        }
29        return dp[m][n];
30    }
31 }
```

### Testcase Result

Accepted

Runtime: 0 ms

Case 1

Case 2

Case 3

Input

s =

"aa"

p =

"a"

Output

false

Expected

Console



Run

Submit

## 1406. Stone Game III

Hint ⋮

Hard



1.7K

47



Companies

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

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

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

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

Assume Alice and Bob **play optimally**.

Return `"Alice"` if Alice will win, `"Bob"` if Bob will win, or `"Tie"` if they will end the game with the same score.

## Example 1:

**Input:** `values = [1,2,3,7]`**Output:** `"Bob"`**Explanation:** Alice will always lose. Her best move will be to take three piles and the score become 6. Now the score of Bob is 7 and Bob wins.

## Example 2:

**Input:** `values = [1,2,3,-9]`**Output:** `"Alice"`**Explanation:** Alice must choose all the three piles at the first move to win and leave Bob with negative score.

If Alice chooses one pile her score will be 1 and the next move Bob's

```

1 class Solution {
2     public String stoneGameIII(int[] stoneValue) {
3         int n = stoneValue.length;
4         Integer[] memo = new Integer[n];
5         int dif = f(stoneValue, n, 0, memo);
6         if (dif > 0) {
7             return "Alice";
8         } else if (dif < 0) {
9             return "Bob";
10        } else {
11            return "Tie";
12        }
13    }
14    private int f(int[] stoneValue, int n, int i, Integer[] memo) {
15        if (i == n) {
16            return 0;
17        }
18        if (memo[i] != null) {
19            return memo[i];
20        }
21        int max = Integer.MIN_VALUE;
22        for (int j = 1; j <= 3; j++) {
23            int dif = f(stoneValue, n, i + j, memo);
24            if (dif > max) {
25                max = dif;
26            }
27        }
28        memo[i] = max;
29        return max;
30    }
31 }
```

Testcase Result

Accepted Runtime: 0 ms

• Case 1 • Case 2 • Case 3

Input

stoneValue =  
[1,2,3,7]

Output

"Bob"

Expected

"Bob"

Console



Run

Submit

Description

Editorial

Solutions (12K)

Submissions

## 9. Palindrome Number

Hint

Easy

9.6K

2.5K



Companies

Given an integer  $x$ , return `true` if  $x$  is a *palindrome*, and `false` otherwise.

### Example 1:

**Input:**  $x = 121$ **Output:** `true`**Explanation:** 121 reads as 121 from left to right and from right to left.

### Example 2:

**Input:**  $x = -121$ **Output:** `false`**Explanation:** From left to right, it reads -121. From right to left, it becomes 121-. Therefore it is not a palindrome.

### Example 3:

**Input:**  $x = 10$ **Output:** `false`**Explanation:** Reads 01 from right to left. Therefore it is not a palindrome.

### Constraints:

- $-2^{31} \leq x \leq 2^{31} - 1$

i Java

Auto

```
1 class Solution {
2     public boolean isPalindrome(int x) {
3         if(x==0){
4             return true;
5         }
6
7         if(x<0 || x%10==0){
8             return false;
9         }
10
11         int temp = x;
12         int rev = 0;
13         while(temp!=0){
14             int rem = temp%10;
15             temp = temp/10;
16
17             rev = rev*10+rem;
18         }
19     }
```

Testcase

Result

**Accepted** Runtime: 0 ms

• Case 1

• Case 2

• Case 3

Input

 $x =$ 

121

Output

true

Expected

Console



Run

Submit