Weekly Contest - 5

402. Remove K Digits

Solved

Medium · Topics · Companies

Given string num representing a non-negative integer `num`, and an integer `k`, return *the smallest possible integer after removing* `k` *digits from* `num`.

Example 1:

> **Input:** num = "1432219", k = 3
> **Output:** "1219"
> **Explanation:** Remove the three digits 4, 3, and 2 to form the new number 1219 which is the smallest.

Example 2:

> **Input:** num = "10200", k = 1
> **Output:** "200"
> **Explanation:** Remove the leading 1 and the number is 200. Note that the output must not contain leading zeroes.

Example 3:

> **Input:** num = "10", k = 2
> **Output:** "0"
> **Explanation:** Remove all the digits from the number and it is left with nothing which is 0.

1 4 3 2 2 1 9 , K=3

Maintain a monotonic Stack

Stack → 1219

9 for(digit : number)

1 while(k>0 and ! stack empty and stack(top) > digit)

~~2~~ K:0 pop from stack

2 push digit into stack , k--;

~~3~~ K:1

~~4~~ K:2 Stack now contains the answer

1

```
class Solution {
    public String removeKdigits(String num, int k) {
        int n = num.length();
        if (k == n) return "0";

        char[] stack = new char[n];
        int stackTop = -1;
        int remove = k;

        for (char digit : num.toCharArray()) {
            while (remove > 0 && stackTop >= 0 && stack[stackTop] > digit) {
                stackTop--;
                remove--;
            }
            stackTop++;
            stack[stackTop] = digit;
        }

        int nonZeroStart = 0;
        while (stack[nonZeroStart] == '0' && nonZeroStart < n - k - 1)
            nonZeroStart++;
        return String.valueOf(stack, nonZeroStart, n - k - nonZeroStart);
    }
}
```

401. Binary Watch

Solved

Easy · Topics · Companies · Hint

A binary watch has 4 LEDs on the top to represent the hours (0-11), and 6 LEDs on the bottom to represent the minutes (0-59). Each LED represents a zero or one, with the least significant bit on the right.

- For example, the below binary watch reads `"4:51"`.

Given an integer `turnedOn` which represents the number of LEDs that are currently on (ignoring the PM), return *all possible times the watch could represent*. You may return the answer in **any order**.

The hour must not contain a leading zero.

- For example, `"01:00"` is not valid. It should be `"1:00"`.

The minute must consist of two digits and may contain a leading zero.

- For example, `"10:2"` is not valid. It should be `"10:02"`.

Try every possible time combination

Check if bit count of Hour + bit count of Minute = turned on

```
public static List<String> readBinaryWatch(int turnedOn) {  2 usages  Nazmul Hasan
    List<String> answer = new ArrayList<>();
    StringBuilder time = new StringBuilder();
    for (int hour = 0; hour < 12; hour++) {
        for (int minute = 0; minute < 60; minute++) {
            if (Integer.bitCount(hour) + Integer.bitCount(minute) == turnedOn) {
                time.setLength(0);
                time.append(hour).append(":");
                if (minute < 10) time.append(0);
                time.append(minute);
                answer.add(time.toString());
            }
        }
    } return answer;
}
```

403. Frog Jump

Solved

Hard · Topics · Companies

A frog is crossing a river. The river is divided into some number of units, and at each unit, there may or may not exist a stone. The frog can jump on a stone, but it must not jump into the water.

Given a list of `stones` positions (in units) in sorted **ascending order**, determine if the frog can cross the river by landing on the last stone. Initially, the frog is on the first stone and assumes the first jump must be `1` unit.

If the frog's last jump was `k` units, its next jump must be either `k - 1`, `k`, or `k + 1` units. The frog can only jump in the forward direction.

Example 1:

> **Input:** stones = [0,1,3,5,6,8,12,17]
> **Output:** true
> **Explanation:** The frog can jump to the last stone by jumping 1 unit to the 2nd stone, then 2 units to the 3rd stone, then 2 units to the 4th stone, then 3 units to the 6th stone, 4 units to the 7th stone, and 5 units to the 8th stone.

Example 2:

> **Input:** stones = [0,1,2,3,4,8,9,11]
> **Output:** false
> **Explanation:** There is no way to jump to the last stone as the gap between the 5th and 6th stone is too large.


```
private static HashMap<Integer, Integer> map;  4 usages

public static boolean canCross(int[] stones) {  2 usages  Nazmul Hasan
    if (stones[1] != 1) return false;
    map = new HashMap<>();
    int n = stones.length;

    for (int i=0; i<n; i++)
        map.put(stones[i], i);

    return solve(stones, idx: 1, n, prevJump: 1, new Boolean[n+1][n+1]);
}
private static boolean solve(int[] stones, int idx, int n, int prevJump, Boolean[][] dp) {
    if (idx == n-1) return true; // reached the end of the river
    if (idx >= n) return false; // out of bound

    if (dp[idx][prevJump] != null) return dp[idx][prevJump];

    // the frog can jump k-1, k, k+1 units
    boolean answer = false;
    for (int nextJump = prevJump-1; nextJump <= prevJump+1; nextJump++) {
        if (nextJump > 0) {
            // since the frog can only jump in the forward direction
            int nextStone = stones[idx] + nextJump;
            if (map.containsKey(nextStone)) // get the index of next jump
                answer = solve(stones, map.get(nextStone), n, nextJump, dp);
            if (answer) break;
        }
    }
    return dp[idx][prevJump] = answer;
}
```