Weekly Contest - 12

475. Heaters

Solved

Medium · Topics · Companies

Winter is coming! During the contest, your first job is to design a standard heater with a fixed warm radius to warm all the houses.

Every house can be warmed, as long as the house is within the heater's warm radius range.

Given the positions of `houses` and `heaters` on a horizontal line, return *the minimum radius standard of heaters so that those heaters could cover all houses.*

Notice that all the `heaters` follow your radius standard, and the warm radius will be the same.

Example 1:

> **Input:** houses = [1,2,3], heaters = [2]
> **Output:** 1
> **Explanation:** The only heater was placed in the position 2, and if we use the radius 1 standard, then all the houses can be warmed.

Example 2:

> **Input:** houses = [1,2,3,4], heaters = [1,4]
> **Output:** 1
> **Explanation:** The two heaters were placed at positions 1 and 4. We need to use a radius 1 standard, then all the houses can be warmed.

Example 3:

> **Input:** houses = [1,5], heaters = [2]
> **Output:** 3

Fixed warm radius
Binary Search Approach -
Sort both the arrays -> houses, heaters

Maximum radius of a heater can be
= Max – Min position of house OR
max - min position of heaters

Minimum radius = 0.
We binary search in range min to max heaters radius

```
public static int findRadius(int[] houses, int[] heaters) {
    Arrays.sort(houses);
    Arrays.sort(heaters);
    int start = 0;
    int end = Math.max(
            Math.abs(houses[0] - heaters[heaters.length-1]),
            Math.abs(heaters[0] - houses[houses.length-1])
    );

    while (start < end) {
        int radius = start + (end - start) / 2;
        if (canWarmAllHouses(radius, houses, heaters))
            end = radius;
        else start = radius + 1;
    }
    return end;
}
// O(houses.length + heaters.length)
private static boolean canWarmAllHouses(int rad, int[] houses, int[] heaters) {
    int hi = 0;
    for (int house : houses) {
        while (hi < heaters.length && Math.abs(house - heaters[hi]) > rad)
            hi++;
        if (hi >= heaters.length) // ran out of heaters
            return false;
    }
    return true;
}
```

468. Validate IP Address

Solved

Medium · Topics · Companies

Given a string `queryIP`, return `"IPv4"` if IP is a valid IPv4 address, `"IPv6"` if IP is a valid IPv6 address or `"Neither"` if IP is not a correct IP of any type.

A valid IPv4 address is an IP in the form `"x1.x2.x3.x4"` where `0 <= xi <= 255` and `xi` **cannot contain** leading zeros. For example, `"192.168.1.1"` and `"192.168.1.0"` are valid IPv4 addresses while `"192.168.01.1"`, `"192.168.1.00"`, and `"192.168@1.1"` are invalid IPv4 addresses.

A valid IPv6 address is an IP in the form `"x1:x2:x3:x4:x5:x6:x7:x8"` where:

- `1 <= xi.length <= 4`
- `xi` is a **hexadecimal string** which may contain digits, lowercase English letter (`'a'` to `'f'`) and upper-case English letters (`'A'` to `'F'`).
- Leading zeros are allowed in `xi`.

For example, `"2001:0db8:85a3:0000:0000:8a2e:0370:7334"` and `"2001:db8:85a3:0:0:8A2E:0370:7334"` are valid IPv6 addresses, while `"2001:0db8:85a3::8A2E:037j:7334"` and `"02001:0db8:85a3:0000:0000:8a2e:0370:7334"` are invalid IPv6 addresses.

Example 1:

> **Input:** queryIP = "172.16.254.1"
> **Output:** "IPv4"
> **Explanation:** This is a valid IPv4 address, return "IPv4".

Example 2:

```
public static String validIPAddress(String queryIP) {
    if (queryIP.contains(".") && !queryIP.startsWith(".") && !queryIP.endsWith(".")) {
        String[] x = queryIP.split( regex: "\\.");
        if (x.length == 4)
            return isIPv4(x) ? "IPv4" : "Neither";
    } else if (queryIP.contains(":") && !queryIP.startsWith(":") && !queryIP.endsWith(":")) {
        String[] x = queryIP.split( regex: ":");
        if (x.length == 8)
            return isIPv6(x) ? "IPv6" : "Neither";
    }
    return "Neither";
}
private static boolean isIPv4(String[] x) {
    for (String xi : x) {
        // can be 0 to 255
        if (xi.isEmpty() || xi.length() > 3) return false;
        // cannot start with 0 but can contain only 0
        if (xi.length() > 1 && xi.charAt(0) == '0') return false;

        for (char c : xi.toCharArray()) // contains digit only
            if (!Character.isDigit(c)) return false;
        // check if the number is below 256
        if (Integer.parseInt(xi) > 255) return false;
    }
    return true;
}
private static boolean isIPv6(String[] x) {
    for (String xi : x) {
        // length of xi is between 1 and 4
        if (xi.isEmpty() || xi.length() > 4) return false;
        // can start with 0
        // contains 0-9, a-f, A-F
        for (char c : xi.toCharArray()) {
            if (!(c >= '0' && c <= '9') &&
                    !(c >= 'a' && c <= 'f') &&
                    !(c >= 'A' && c <= 'F'))
                return false;
        }
    } return true;
}
```

474. Ones and Zeroes

Solved

Medium · Topics · Companies

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

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

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

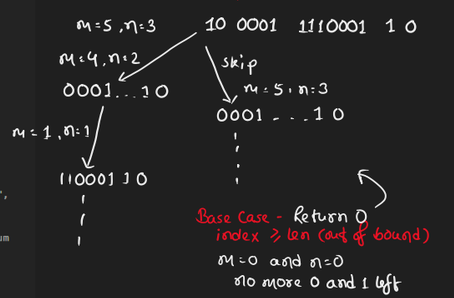
Example 1:

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

Example 2:

> **Input:** strs = ["10","0","1"], m = 1, n = 1
> **Output:** 2
> **Explanation:** The largest subset is {"0", "1"}, so the answer is 2.

For each str in strs, we have 2 options
1) Take 2) Skip




```
public static int findMaxForm(String[] strs, int m, int n) {
    int len = strs.length;

    int[][] zeroOneCount = new int[len][2];
    for (int i=0; i<len; i++) {
        for (char c : strs[i].toCharArray()) {
            if (c == '0') zeroOneCount[i][0]++;
            else zeroOneCount[i][1]++;
        }
    }
    int[][][] dp = new int[len+1][m+1][n+1];
    for (int[][] dr : dp)
        for (int[] d : dr)
            Arrays.fill(d, val: -1);
    return solve(dp, zeroOneCount, idx: 0, len, m, n);
}
private static int solve(int[][][] dp, int[][] counts, int idx, int len,
                         int zeroCount, int oneCount) {
    if (idx >= len) return 0; // out of bound

    if (dp[idx][zeroCount][oneCount] != -1)
        return dp[idx][zeroCount][oneCount];

    int skip = solve(dp, counts, idx: idx+1, len, zeroCount, oneCount);
    int take = 0;
    if (zeroCount >= counts[idx][0] && oneCount >= counts[idx][1]) {
        take = 1 + solve(dp, counts, idx: idx+1, len,
                zeroCount: zeroCount-counts[idx][0], oneCount: oneCount-counts[idx][1]);
    }
    return dp[idx][zeroCount][oneCount] = Math.max(take, skip);
}
```