Maximum Length of Repeated Subarray

Given two integer arrays `A` and `B`, return the maximum length of an subarray that appears in both arrays.

Example 1:

```
Input:
A: [1,2,3,2,1]
B: [3,2,1,4,7]
Output: 3
Explanation:
The repeated subarray with maximum length is [3, 2, 1].
```

Note:

1. 1
2. 0

Solution 1

The code explains itself:

```
class Solution {
    public int findLength(int[] A, int[] B) {
        if(A == null||B == null) return 0;
        int m = A.length;
        int n = B.length;
        int max = 0;
        //dp[i][j] is the length of longest common subarray ending with nums[i] and
nums[j]
        int[][] dp = new int[m + 1][n + 1];
        for(int i = 0;i <= m;i++){
            for(int j = 0;j <= n;j++){
                if(i == 0 || j == 0){
                    dp[i][j] = 0;
                }
                else{
                    if(A[i - 1] == B[j - 1]){
                        dp[i][j] = 1 + dp[i - 1][j - 1];
                        max = Math.max(max,dp[i][j]);
                    }
                }
            }
        }
        return max;
    }
}
```

Hope it helps!

written by yujun original link here

Solution 2

DP formula

```
/**
 * dp[i][j] = a[i] == b[j] ? dp[i + 1][j + 1] : 0;
 * dp[i][j] : max lenth of common subarray start at a[i] & b[j];
 */
```

Java - DP matrix

```java
class Solution {
    public int findLength(int[] a, int[] b) {
        int m = a.length, n = b.length;
        if (m == 0 || n == 0) return 0;
        int[][] dp = new int[m + 1][n + 1];
        int max = 0;
        for (int i = m - 1; i >= 0; i--)
            for (int j = n - 1; j >= 0; j--)
                max = Math.max(max, dp[i][j] = a[i] == b[j] ? 1 + dp[i + 1][j + 1]
: 0);
        return max;
    }
}
```

Java - DP array

```java
class Solution {
    public int findLength(int[] a, int[] b) {
        int m = a.length, n = b.length;
        if (m == 0 || n == 0) return 0;
        int[] dp = new int[n + 1];
        int max = 0;
        for (int i = m - 1; i >= 0; i--)
            for (int j = 0; j < n; j++)
                max = Math.max(max, dp[j] = a[i] == b[j] ? 1 + dp[j + 1] : 0);
        return max;
    }
}
```

C++ - DP array

```
class Solution {
public:
    int findLength(vector<int>& a, vector<int>& b) {
        int m = a.size(), n = b.size();
        if (!m || !n) return 0;
        vector<int> dp(n + 1);
        int res = 0;
        for (int i = m - 1; i >= 0; i--) {
            for (int j = 0; j < n; j++) {
                res = max(res, dp[j] = a[i] == b[j] ? 1 + dp[j + 1] : 0);
            }
        }
        return res;
    }
};
```

written by alexander original link here

Solution 3

```
class Solution {
    public int findLength(int[] A, int[] B) {
        int l1 = A.length, l2 = B.length, ans = 0;
        if (l1 == 0 || l2 == 0)
            return 0;
        HashMap < Integer, List < Integer >> map = new HashMap < > ();
        List < Integer > list;
        for (int i = 0; i < l1; i++) {
            int n = A[i];
            list = map.getOrDefault(n, new ArrayList<Integer>());
            list.add(i);
            map.put(n, list);
        }

        for (int i = 0; i < l2 && l2-i > ans; i++) {
            int n = B[i];
            if (map.containsKey(n)){
                list = map.get(n);
                for (int m: list) {
                    if (l1 - m < ans)
                        break;
                    int count = 1, k = m + 1;
                    for (int j = i + 1; j < l2 && k < l1; j++, k++) {
                        if (B[j] == A[k]) {
                            count++;
                        } else {
                            break;
                        }
                    }
                    ans = Math.max(ans, count);
                }
            }
        }

        return ans;
    }
}
```

Update: I have updated my code. Thanks for suggestions.

written by ashish53v original link here