Q1: Merge two arrays by satisfying given constraints

Given two sorted arrays X[] and Y[] of size m and n each where m >= n and X[] has exactly n vacant cells,

merge elements of Y[] in their correct position in array X[], i.e., merge (X, Y) by keeping the sorted order.

For example,

Input: X[] = { 0, 2, 0, 3, 0, 5, 6, 0, 0 }

Y[] = { 1, 8, 9, 10, 15 } The vacant cells in X[] is represented by 0

Output: X[] = { 1, 2, 3, 5, 6, 8, 9, 10, 15 }

SOLUTION:

import java.util.Arrays;

class Main

{

    // Function to merge `X[0… m]` and `Y[0… n]` into `X[0… m+n+1]`

    private static void merge(int[] X, int[] Y, int m, int n)

    {

        // size of `X[]` is `k+1`

        int k = m + n + 1;

        // run if X[] or Y[] has elements left

        while (m >= 0 && n >= 0)

        {

            // put the next greater element in the next free position in `X[]`

            // from the end

            if (X[m] > Y[n]) {

                X[k--] = X[m--];

            }

            else {

                X[k--] = Y[n--];

            }

        }

        // copy the remaining elements of `Y[]` (if any) to `X[]`

        while (n >= 0) {

            X[k--] = Y[n--];

        }

        // fill `Y[]` with all zeros

        Arrays.fill(Y, 0);

    }

    // The function moves non-empty elements in `X[]` in the

    // beginning and then merge them with `Y[]`

    public static void rearrange(int[] X, int[] Y)

    {

        // return if `X` is empty

        if (X.length == 0) {

            return;

        }

        // moves non-empty elements of `X[]` at the beginning

        int k = 0;

        for (int value: X)

        {

            if (value != 0) {

                X[k++] = value;

            }

        }

        // merge `X[0…k-1]` and `Y[0… n-1]` into `X[0… m-1]`

        merge(X, Y, k - 1, Y.length - 1);

    }

    public static void main (String[] args)

    {

        // vacant cells in `X[]` is represented by 0

        int[] X = { 0, 2, 0, 3, 0, 5, 6, 0, 0};

        int[] Y = { 1, 8, 9, 10, 15 };

        /\* Validate input before calling `rearrange()`

            1. Both arrays `X[]` and `Y[]` should be sorted (ignore 0's in `X[]`)

            2. Size of array `X[]` >= size of array `Y[]` (i.e., `m >= n`)

            3. Total number of vacant cells in array `X[]` = size of array `Y[]` \*/

        // merge `Y[]` into `X[]`

        rearrange(X, Y);

        // print merged array

        System.out.println(Arrays.toString(X));

    }

}

Q2:Find maximum sum path involving elements of given arrays

Given two sorted arrays of integers, find a maximum sum path involving elements of both arrays whose sum is maximum.

We can start from either array, but we can switch between arrays only through its common elements.

For example,

Input: X = { 3, 6, 7, 8, 10, 12, 15, 18, 100 }

Y = { 1, 2, 3, 5, 7, 9, 10, 11, 15, 16, 18, 25, 50 }

The maximum sum path is: 1 —> 2 —> 3 —> 6 —> 7 —> 9 —> 10 —> 12 —> 15 —> 16 —> 18 —> 100

The maximum sum is 199

SOLUTION:

Class MaximumSumPath {

// Utility function to find maximum of two integers

int max(int x, int y) { return (x > y) ? x : y; }

// This function returns the sum of elements on maximum

// path from beginning to end

int maxPathSum(int ar1[], int ar2[], int m, int n)

{

// initialize indexes for ar1[] and ar2[]

int i = 0, j = 0;

// Initialize result and current sum through ar1[]

// and ar2[].

int result = 0, sum1 = 0, sum2 = 0;

// Below 3 loops are similar to merge in merge sort

while (i < m && j < n) {

// Add elements of ar1[] to sum1

if (ar1[i] < ar2[j])

sum1 += ar1[i++];

// Add elements of ar2[] to sum2

else if (ar1[i] > ar2[j])

sum2 += ar2[j++];

// we reached a common point

else {

// Take the maximum of two sums and add to

// result

// Also add the common element of array,

// once

result += max(sum1, sum2) + ar1[i];

// Update sum1 and sum2 for elements after

// this intersection point

sum1 = 0;

sum2 = 0;

// update i and j to move to next element of

// each array

i++;

j++;

}

}

// Add remaining elements of ar1[]

while (i < m)

sum1 += ar1[i++];

// Add remaining elements of ar2[]

while (j < n)

sum2 += ar2[j++];

// Add maximum of two sums of remaining elements

result += max(sum1, sum2);

return result;

}

// Driver code

public static void main(String[] args)

{

MaximumSumPath sumpath = new MaximumSumPath();

int ar1[] = { 2, 3, 7, 10, 12, 15, 30, 34 };

int ar2[] = { 1, 5, 7, 8, 10, 15, 16, 19 };

int m = ar1.length;

int n = ar2.length;

// Function call

System.out.println(

"Maximum sum path is :"

+ sumpath.maxPathSum(ar1, ar2, m, n));

}

}

Q3: Write a Java Program to count the number of words in a string using HashMap.

SOLUTION:

// Java Program to find the occurrence

// of words in a String using HashMap.

import java.io.\*;

import java.util.HashMap;

import java.util.Map;

class GFG {

public static void main(String[] args)

{

// Declaring the String

String str = "Alice is girl and Bob is boy";

// Declaring a HashMap of <String, Integer>

Map<String, Integer> hashMap = new HashMap<>();

// Splitting the words of string

// and storing them in the array.

String[] words = str.split(" ");

for (String word : words) {

// Asking whether the HashMap contains the

// key or not. Will return null if not.

Integer integer = hashMap.get(word);

if (integer == null)

// Storing the word as key and its

// occurrence as value in the HashMap.

hashMap.put(word, 1);

else {

// Incrementing the value if the word

// is already present in the HashMap.

hashMap.put(word, integer + 1);

}

}

System.out.println(hashMap);

}

}

Q4: Write a Java Program to find the duplicate characters in a string.

SOLUTION:

public class DuplicateCharacters {

     public static void main(String[] args) {

        String string1 = "Great responsibility";

        int count;

        //Converts given string into character array

        char string[] = string1.toCharArray();

        System.out.println("Duplicate characters in a given string: ");

        //Counts each character present in the string

        for(int i = 0; i <string.length; i++) {

            count = 1;

            for(int j = i+1; j <string.length; j++) {

                if(string[i] == string[j] && string[i] != ' ') {

                    count++;

                    //Set string[j] to 0 to avoid printing visited character

                    string[j] = '0';

                }

            }

            //A character is considered as duplicate if count is greater than 1

            if(count > 1 && string[i] != '0')

                System.out.println(string[i]);

        }

    }

}