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 }

**ANS:**

public class MergeArrays {

public static void mergeArrays(int[] X, int[] Y) {

int m = X.length;

int n = Y.length;

// Move all elements of X to the end

int k = m - 1;

for (int i = m - 1; i >= 0; i--) {

if (X[i] != 0) {

X[k] = X[i];

k--;

}

}

// Merge X and Y

int i = k + 1; // Starting index of X

int j = 0; // Starting index of Y

int idx = 0; // Current index to insert elements in X

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

if (X[i] < Y[j]) {

X[idx] = X[i];

i++;

} else {

X[idx] = Y[j];

j++;

}

idx++;

}

// If there are remaining elements in Y[]

while (j < n) {

X[idx] = Y[j];

j++;

idx++;

}

}

public static void main(String[] args) {

int[] X = {2, 4, 6, 8, 0, 0, 0}; // Example X array with vacant cells

int[] Y = {1, 3, 5}; // Example Y array

mergeArrays(X, Y);

// Print the merged array X

for (int num : X) {

System.out.print(num + " ");

}

}

}

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

**ANS:**

public class MaxSumPath {

public static int maxSumPath(int[] X, int[] Y) {

int m = X.length;

int n = Y.length;

int i = 0, j = 0; // Pointers for arrays X and Y respectively

int sumX = 0, sumY = 0; // Variables to keep track of the sum

int result = 0; // Variable to store the maximum sum

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

if (X[i] < Y[j]) {

sumX += X[i++];

} else if (X[i] > Y[j]) {

sumY += Y[j++];

} else { // If both elements are equal, we have a common element

result += Math.max(sumX, sumY) + X[i]; // Add the maximum sum between the two arrays along with the common element

sumX = 0;

sumY = 0;

i++;

j++;

}

}

// Add remaining elements of X if any

while (i < m) {

sumX += X[i++];

}

// Add remaining elements of Y if any

while (j < n) {

sumY += Y[j++];

}

// Add the remaining sums to the result

result += Math.max(sumX, sumY);

return result;

}

public static void main(String[] args) {

int[] X = {3, 6, 7, 8, 10, 12, 15, 18, 100};

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

int maxSum = maxSumPath(X, Y);

System.out.println("The maximum sum path is: " + maxSum);

}

}

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

**ANS**:

import java.util.HashMap;

public class WordCount {

public static void main(String[] args) {

String inputString = "This is a sample input string with some sample words.";

// Create a HashMap to store word counts

HashMap<String, Integer> wordCountMap = new HashMap<>();

// Split the input string into words using whitespace as delimiter

String[] words = inputString.split("\\s+");

// Iterate through each word and update the word count in the HashMap

for (String word : words) {

// Convert word to lowercase to ensure case-insensitivity

String lowercaseWord = word.toLowerCase();

// Remove punctuation (if needed)

lowercaseWord = lowercaseWord.replaceAll("[^a-zA-Z]", "");

// If the word is already present in the HashMap, increment its count, otherwise add it with count 1

wordCountMap.put(lowercaseWord, wordCountMap.getOrDefault(lowercaseWord, 0) + 1);

}

// Print the word counts

System.out.println("Word counts:");

for (String word : wordCountMap.keySet()) {

System.out.println(word + ": " + wordCountMap.get(word));

}

}

}

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

**ANS:**

import java.util.HashMap;

import java.util.Map;

public class DuplicateCharacters {

public static void main(String[] args) {

String inputString = "hello world";

// Create a HashMap to store character counts

Map<Character, Integer> charCountMap = new HashMap<>();

// Convert the input string to lowercase (to treat 'H' and 'h' as the same)

inputString = inputString.toLowerCase();

// Iterate through each character in the input string

for (char c : inputString.toCharArray()) {

// If the character is not a whitespace

if (c != ' ') {

// If the character is already present in the map, increment its count, otherwise add it with count 1

charCountMap.put(c, charCountMap.getOrDefault(c, 0) + 1);

}

}

// Print the duplicate characters along with their counts

System.out.println("Duplicate characters:");

for (Map.Entry<Character, Integer> entry : charCountMap.entrySet()) {

if (entry.getValue() > 1) {

System.out.println(entry.getKey() + ": " + entry.getValue());

}

}

}

}