**QUIZ 11**

Given the list of array return array in which each element is the product of other element except ith element (try to do it without division operation)

 input: [1,2,3,4]

output:[24,12,8,6]

import java.util.Arrays;

public class ProductExceptSelf {

public static int[] productExceptSelf(int[] nums) {

int n = nums.length;

int[] leftProducts = new int[n];

int[] rightProducts = new int[n];

int leftProduct = 1;

for (int i = 0; i < n; i++) {

leftProducts[i] = leftProduct;

leftProduct \*= nums[i];

}

int rightProduct = 1;

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

rightProducts[i] = rightProduct;

rightProduct \*= nums[i];

}

int[] result = new int[n];

for (int i = 0; i < n; i++) {

result[i] = leftProducts[i] \* rightProducts[i];

}

return result;

}

public static void main(String[] args) {

int[] inputArray = {1, 2, 3, 4};

int[] outputArray = productExceptSelf(inputArray);

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

}

}

OUTPUT:

[24, 12, 8, 6]

 Given an array list return all possible permutations Input: nums = [1,4,3]

Output: [[1,4,3],[1,3,4],[4,1,3],[4,3,1],[3,1,4],[3,4,1]]

import java.util.ArrayList;

import java.util.List;

public class Permutations {

public static List<List<Integer>> permute(int[] nums) {

List<List<Integer>> result = new ArrayList<>();

List<Integer> current = new ArrayList<>();

boolean[] used = new boolean[nums.length];

backtrack(nums, used, current, result);

return result;

}

private static void backtrack(int[] nums, boolean[] used, List<Integer> current, List<List<Integer>> result) {

if (current.size() == nums.length) {

result.add(new ArrayList<>(current));

return;

}

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

if (!used[i]) {

used[i] = true;

current.add(nums[i]);

backtrack(nums, used, current, result);

used[i] = false;

current.remove(current.size() - 1);

}

}

}

public static void main(String[] args) {

int[] inputArray = {1, 4, 3};

List<List<Integer>> permutations = permute(inputArray);

System.out.println(permutations);

}

}

OUTPUT:

[[1, 4, 3], [1, 3, 4], [4, 1, 3], [4, 3, 1], [3, 1, 4], [3, 4, 1]]

Return all the clubbed words

Input: words =["mat","mate","matbellmates","bell","bellmatesbell","butterribbon","butter","ribbon"] Output: ["matbellmates","bellmatesbell","butterribbon"]

import java.util.ArrayList;

import java.util.Arrays;

import java.util.HashSet;

import java.util.List;

import java.util.Set;

public class ClubbedWords {

public static List<String> findAllClubbedWords(String[] words) {

List<String> result = new ArrayList<>();

Set<String> wordSet = new HashSet<>(Arrays.asList(words));

for (String word : words) {

wordSet.remove(word);

if (isClubbedWord(word, wordSet)) {

result.add(word);

}

wordSet.add(word);

}

return result;

}

private static boolean isClubbedWord(String word, Set<String> wordSet) {

if (word.isEmpty()) {

return true;

}

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

String prefix = word.substring(0, i);

String suffix = word.substring(i);

if (wordSet.contains(prefix) && (wordSet.contains(suffix) || isClubbedWord(suffix, wordSet))) {

return true;

}

}

return false;

}

public static void main(String[] args) {

String[] inputWords = {"mat", "mate", "matbellmates", "bell", "bellmatesbell", "butterribbon", "butter", "ribbon"};

List<String> clubbedWords = findAllClubbedWords(inputWords);

System.out.println(clubbedWords);

}

}

OUTPUT:

["matbellmates","bellmatesbell","butterribbon"]