Amazone Question

1)You are in charge of preparing a recently purchased lot for Amazon’s building. The lot is covered with trenches and has a single obstacle that needs to be taken down before the foundation is prepared for the building. The demolition robot must remove the obstacle before the progress can be made on the building.

Write an algorithm to determine the minimum distance required for the demolition robot to remove the obstacle. Assumptions:

• The lot is flat, except the trenches and can be represented by a 2D grid.

• The demolition robot must start at the top left corner of the lot, which is always flat, and can move on block up, down, right, left • The demolition robot cannot enter trenches and cannot leave the lot.

• The flat areas are indicated by 1,

areas with trenches are indicated by 0, and the obstacle is indicated by 9

Input: The input of the function has 3 arguments:

numRows – number of rows numColumns – number of columns lot – 2d grid of integers

Output: Return an integer that indicated the minimum distance traversed to remove the obstacle else return -1 Constraints: 1<= numRows, numColumns <= 1000

Example:  
 numRows = 3,

numColumns = 3

lot = [ [1, 0, 0], [1, 0, 0], [1, 9, 1]]

Output: 3

package Training\_stuff;

import java.util.ArrayList;

import java.util.Arrays;

import java.util.List;

public class DemolitionRobot {

public static int minimumDistance(int numRows, int numColumns, List<List<Integer>> lot) {

int[] result = {Integer.***MAX\_VALUE***}; // Using an array to store the result as a mutable container

// Find the starting position of the robot (top-left corner)

List<Integer> start = lot.stream()

.flatMap(List::stream)

.collect(ArrayList::new, (arr, value) -> {

if (value == 1) {

arr.add(value);

}

}, ArrayList::addAll);

*findMinimumDistance*(0, 0, lot, 0, result);

return (result[0] != Integer.***MAX\_VALUE***) ? result[0] : -1;

}

private static void findMinimumDistance(int row, int col, List<List<Integer>> lot, int distance, int[] result) {

if (row < 0 || row >= lot.size() || col < 0 || col >= lot.get(0).size() || lot.get(row).get(col) == 0) {

return; // Invalid move

}

if (lot.get(row).get(col) == 9) {

result[0] = Math.*min*(result[0], distance); // Update the result with the minimum distance

return;

}

// Mark the current cell as visited

int originalValue = lot.get(row).get(col);

lot.get(row).set(col, 0);

// Explore all possible moves (up, down, left, right)

*findMinimumDistance*(row - 1, col, lot, distance + 1, result);

*findMinimumDistance*(row + 1, col, lot, distance + 1, result);

*findMinimumDistance*(row, col - 1, lot, distance + 1, result);

*findMinimumDistance*(row, col + 1, lot, distance + 1, result);

// Restore the original value of the cell

lot.get(row).set(col, originalValue);

}

public static void main(String[] args) {

int numRows = 3;

int numColumns = 3;

List<List<Integer>> lot = Arrays.*asList*(

Arrays.*asList*(1, 0, 0),

Arrays.*asList*(1, 0, 0),

Arrays.*asList*(1, 9, 1)

);

int result = *minimumDistance*(numRows, numColumns, lot);

System.***out***.println("Output: " + result);

}

}

2) Amazon is planning to release a new order prioritization algorithm to optimize fulfilling Prime deliveries on time. All orders (Prime or otherwise) are given an alphanumeric ID code. However, Prime orders are given additional metadata that consists of a space-delimited list of lowercase English letters, whereas non-Prime orders are given metadata that consists only of a space-delimited string of positive integers.

Each order is therefore defined as their alphanumeric id code, followed by a space, followed by their space-delimited metadata.

You have been tasked with sorting a list of all orders in the order queue to assist in prioritization of fulfillment. They should be sorted according to the following

order: 1. The Prime orders should be returned first, sorted by lexicographic sort of the alphabetic metadata.

2. Only in the case of ties, the identifier should be used as a backup sort.

3. The remaining non-Prime orders must all come after, in the original order they were given in the input. Write a function or method to return a list of strings representing the correctly prioritized orders according to this system.

w Input The input to the function/method consists of one argument: orderList, a list of strings representing all of the orders.

Output Return a list of strings representing the correctly Examples

Input: orderList = [zld 93 12] [fp kindle book] [10a echo show] [17g 12 25 6] [ab1 kindle book] [125 echo dot second generation]

Output: [125 echo dot second generation] [10a echo show] [ab1 kindle book] [fp kindle book] [zld 93 12] [17g 12 25 6]

Explanation: There are four Prime orders (lines with words) in this order list. Because "echo" comes before “kindle”, those lines should come first, with “dot" coming before “show”. Because two lines have the metadata “kindle book”, their tie should be broken by the identifier, where “ab1" comes before "fp".

Finally, the non-Prime numeric orders should come last, in the original order, they were in the

input.Amazon is planning to release a new order prioritization algorithm to optimize fulfilling Prime deliveries on time. All orders (Prime or otherwise) are given an alphanumeric ID code.

However, Prime orders are given additional metadata that consists of a space-delimited list of lowercase English letters, whereas non-Prime orders are given metadata that consists only of a space-delimited string of positive integers

import java.util.ArrayList;

import java.util.Collections;

import java.util.Comparator;

import java.util.List;

public class OrderPrioritization {

public static List<String> prioritizeOrders(List<String> orderList) {

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

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

for (String order : orderList) {

String[] orderParts = order.split("\\s+", 2);

String orderId = orderParts[0];

String metadata = orderParts.length > 1 ? orderParts[1] : "";

if (metadata.matches(".\*[a-zA-Z].\*")) {

primeOrders.add(order);

} else {

nonPrimeOrders.add(order);

}

}

// Sort prime orders by metadata and then by identifier

Collections.sort(primeOrders, Comparator.comparing((String o) -> o.split("\\s+", 2)[1])

.thenComparing(Comparator.comparing(o -> o.split("\\s+", 2)[0])));

// Combine sorted prime orders and non-prime orders

primeOrders.addAll(nonPrimeOrders);

return primeOrders;

}

public static void main(String[] args) {

List<String> orderList = List.of(

"zld 93 12", "fp kindle book", "10a echo show", "17g 12 25 6", "ab1 kindle book", "125 echo dot second generation"

);

List<String> result = prioritizeOrders(orderList);

for (String order : result) {

System.out.println(order);

}

}

}