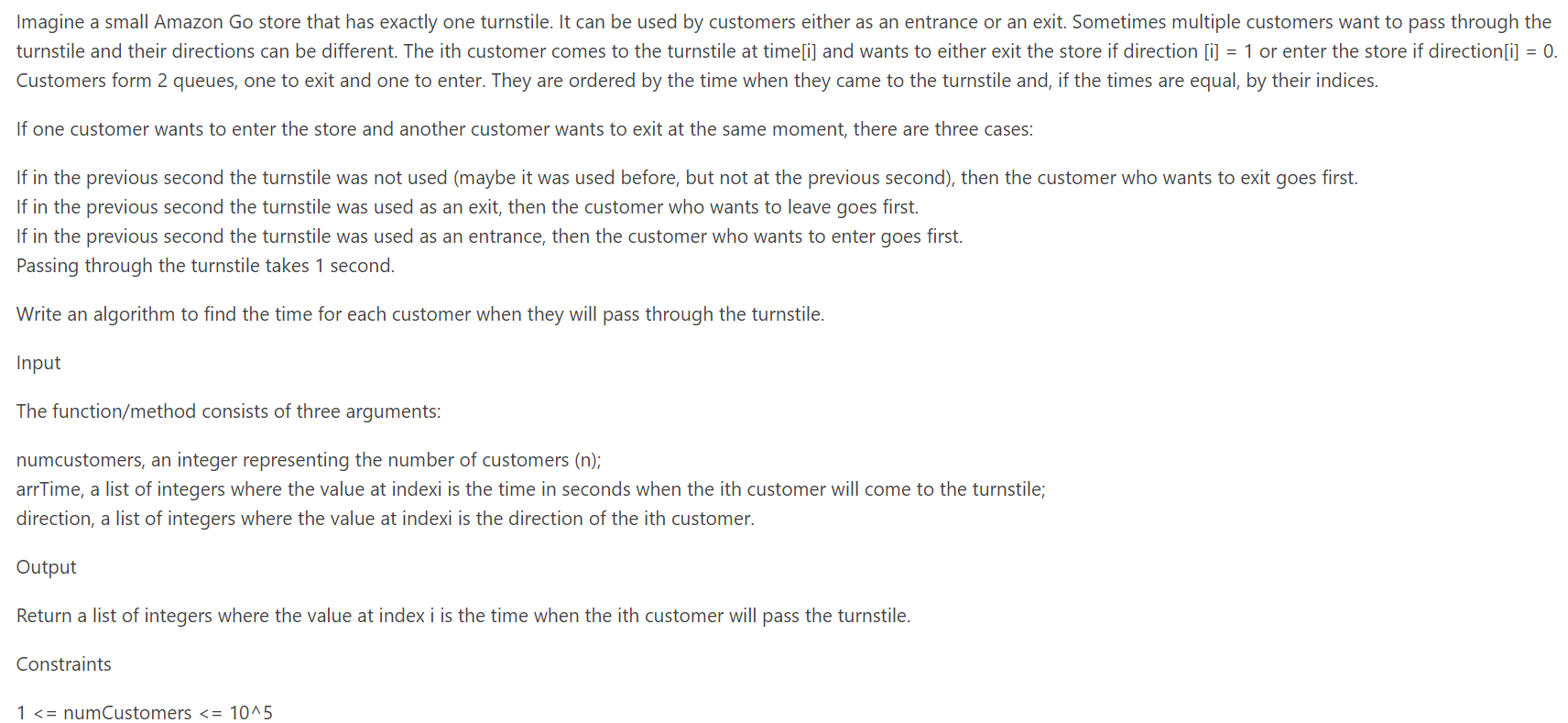
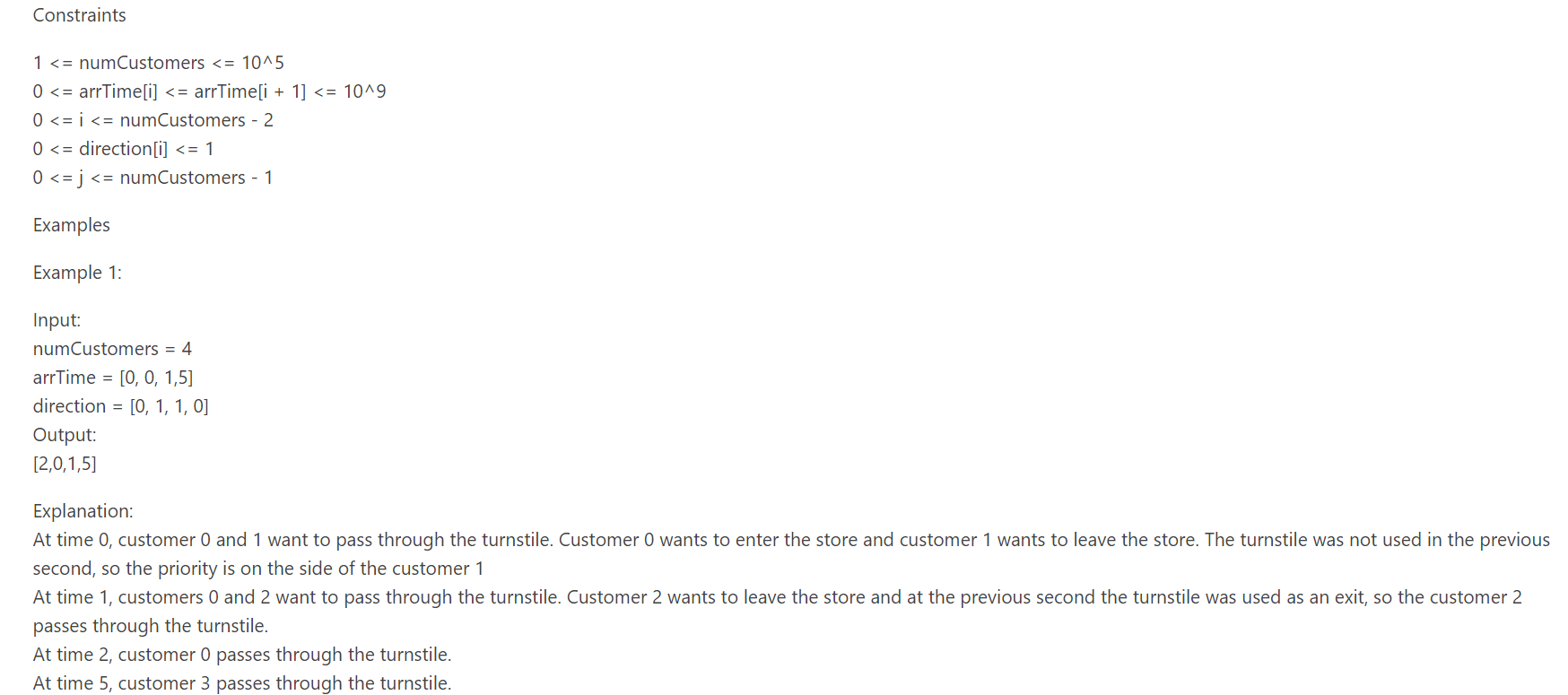
Amazon | OA 2020 | Turnstile

LEET CODE : <https://leetcode.com/discuss/interview-question/853053/>





Should cover all cases,

import java.util.\*;

public class Solution {

public static void main(String[] args) {

// TODO Auto-generated method stub

Solution s = new Solution();

// int[] result = s.getTimes(4, new int[] { 0, 0, 1, 5 }, new int[] { 0, 1, 1, 0 });  
int[] result = s.getTimes(5, new int[] { 0, 1, 1, 3, 3 }, new int[] { 0, 1, 0, 0, 1 });  
System.out.println(Arrays.toString(result));  
}

public int[] getTimes(int numCustomers, int[] arrTime, int[] direction) {

int[] result = new int[numCustomers];

Map<Integer, List<int[]>> map = new HashMap<Integer, List<int[]>>();

Queue<int[]> entryQ = new ArrayDeque<int[]>();

Queue<int[]> exitQ = new ArrayDeque<int[]>();

int i = 0;

int turnstileDirection = -1; // default unused-1

for (int a : arrTime) {

if (!map.containsKey(a)) {

map.put(a, new ArrayList<int[]>());

}

map.get(a).add(new int[] { i, direction[i] });

i++;

}

i = 0; // used to determine timeStamp,

while (numCustomers > 0) {

if (map.containsKey(i)) {

List<int[]> customers = map.get(i);

for (int[] customer : customers) {

if (customer[1] == 1) {

exitQ.add(customer);

} else {

entryQ.add(customer);

}

}

}

if (entryQ.isEmpty() && exitQ.isEmpty()) {

turnstileDirection = -1;

i++;

continue;

}

int[] current;

if (!entryQ.isEmpty() && !exitQ.isEmpty()) {

switch (turnstileDirection) {

case -1: // unused

current = exitQ.poll();

result[current[0]] = i;

numCustomers--;

turnstileDirection = 1;

i++;

break;

case 1: // used for exit

current = exitQ.poll();

result[current[0]] = i;

numCustomers--;

turnstileDirection = 1;

i++;

break;

case 0: // used to entry

current = entryQ.poll();

result[current[0]] = i;

numCustomers--;

turnstileDirection = 0;

i++;

break;

}

continue;

}

if (!entryQ.isEmpty()) {

current = entryQ.poll();

result[current[0]] = i;

numCustomers--;

turnstileDirection = 0;

i++;

continue;

}

if (!exitQ.isEmpty()) {

current = exitQ.poll();

result[current[0]] = i;

numCustomers--;

turnstileDirection = 1;

i++;

continue;

}

}

return result;

}

}

ALTERNATE :

Code should be refactored.

public static List CustomerPassingTurnstile(int numCustomers, List arrTime, List direction) {

HashMap<Integer, List<Integer>> timeToCustomerMap = new HashMap<>();

for (int i = 0; i < arrTime.size(); i++) {

int time = arrTime.get(i);

if (!timeToCustomerMap.containsKey(time)) {

timeToCustomerMap.put(time, new ArrayList<Integer>());

}

timeToCustomerMap.get(time).add(i);

}

int timeInterval = 0;

Queue<Integer> queue = new ArrayDeque<>();

List<Integer> customerOrder = Arrays.asList(new Integer[numCustomers]);

int previousDirection = -1;

int customerRemaining = numCustomers;

while (customerRemaining > 0) {

if (!timeToCustomerMap.containsKey(timeInterval)) {

if(!queue.isEmpty()) {

int customer = queue.poll();

customerOrder.set(customer, timeInterval);

previousDirection = direction.get(customer);

customerRemaining--;

}

timeInterval += 1;

continue;

}

List<Integer> customers = timeToCustomerMap.get(timeInterval);

if (customers.size() == 1 && queue.isEmpty()) { // only one customer no one waiting

int customer = customers.get(0);

customerOrder.set(customer, timeInterval);

previousDirection = direction.get(customer);

customerRemaining--;

timeInterval += 1;

continue;

}

boolean turnStileUsed = false;

if (customers.size() >= 1) {

for (int i = 0; i < customers.size(); i++) {

int customerInArr = customers.get(i);

int arrCustdirection = direction.get(customerInArr);

switch (previousDirection) {

case 0:

if(arrCustdirection == 0) {

customerOrder.set(customerInArr, timeInterval);

previousDirection = arrCustdirection;

customerRemaining--;

turnStileUsed=true;

break;

}

if(!queue.isEmpty()) {

int customerInQueue = queue.peek();

int qCustdirection = direction.get(customerInQueue);

if (qCustdirection == 0) {

customerOrder.set(timeInterval, queue.poll());

previousDirection = qCustdirection;

customerRemaining--;

turnStileUsed=true;

}

}

queue.offer(customerInArr);

break;

default:

if(arrCustdirection == 1) {

customerOrder.set(customerInArr, timeInterval);

previousDirection = arrCustdirection;

customerRemaining--;

turnStileUsed=true;

break;

}

if(!queue.isEmpty()) {

int customerInQueue = queue.peek();

int qCustdirection = direction.get(customerInQueue);

if (qCustdirection == 1) {

customerOrder.set(queue.poll(), timeInterval);

previousDirection = qCustdirection;

customerRemaining--;

turnStileUsed=true;

}

}

queue.offer(customerInArr);

break;

}

}

}

if (!turnStileUsed) {

if(!queue.isEmpty()) {

previousDirection = direction.get(queue.peek());

customerOrder.set(queue.poll(), timeInterval);

customerRemaining--;

} else {

previousDirection = -1;

}

}

timeInterval += 1;

}

return customerOrder;

}