Computer Based Test

1. *For each question, there will be a testcase (e.g. the testcase for question 1 is* testcase1.txt) *Please note that the testcase you received may not necessarily to be the same as the test-case used for grading. Please consider edge cases.*
2. *Let your program read the corresponding testcase and print the result to a corresponding text file(e.g. the output for question 2 should be* output2.txt*).*
3. *For each question, we have provided a corresponding answer (e.g assume you're solving problem 3, the corresponding answer is* ans3.txt*), and your output should be same as answer.*
4. *For each question, your program file should be named* DS{problem number}_{student ID}.c. *(e.g. assume your student ID is* 7112056067 *and you're solving question 3, your program file should be* DS3_7112056067.c)
5. *Each question is scored independently, but partial credit is not awarded; full credit is given only for complete correctness.*
6. *You can submit for three times.*

1 Problem 1

Johnny wants to go around the city to visit each of his friends. There is a friend on every street, so he wants to drive through each street exactly once to make sure he meets everyone. To make things easier, he wants to start from home and end up back at home. The streets and intersections in the city follow these rules:

- The city has up to 1994 streets, each with a unique number (starting from 1).
- All streets are bidirectional.
- Each street connects exactly two intersections (possibly the same one, forming a self-loop).
- Intersections are numbered from 1 to 44.
- No two streets share the same street number.

Rules

Help Johnny find a round trip that starts and ends at his home, traversing each street exactly once.

- The city map is fully connected — from any street, it is possible to reach any other street.
- Streets are narrow; once Johnny enters a street, he cannot make a U-turn or go back.
- Johnny's house is located at the endpoint with the smaller intersection number on the first street listed in the input.
- If multiple valid round trips exist, output the one with the lexicographically smallest sequence of street numbers.
- If no such round trip exists, output the following message:

```
Round trip does not exist.
```

Input Format

Input file consists of several blocks. Each block describes one town. Each line in the block contains three integers x, y, z, where x > 0 and y > 0 are the numbers of junctions which are connected by the street number z. The end of the block is marked by the line containing x = y = 0.At the end of the input file there is an empty block, x = y = 0.

Output Format

The output file consists of 2 line blocks corresponding to the blocks of the input file. The first line of each block contains the sequence of street numbers (single members of the sequence are separated by space) describing Johnny's round trip. If the round trip cannot be found the corresponding output block contains the message ‘Round trip does not exist.’. The second line of each block is empty.

Sample Input

```
1 2 1
2 3 2
3 1 6
1 2 5
2 3 3
3 1 4
0 0
1 2 1
2 3 2
1 3 3
2 4 4
0 0
0 0
```

Sample Output

```
1 2 3 5 4 6

Round trip does not exist.
```

2 Problem 2

You find yourself trapped inside a secret underground research facility, composed of multiple levels of rooms and corridors. The facility is structured as a 3D grid with H floors, each floor containing R rows and C columns of rooms.

Rules

To complete this task, you must implement a breadth-first search (BFS) using a manually constructed queue based on a singly linked list.

You are **not allowed to use standard library containers** such as `queue`, `deque`, `list`, `vector`, etc.

- You must use the following structure, or else you will receive a score of 0.

```
    struct Node {
    int z, x, y;              // Coordinate
    char val;                 //('S', 'E', '.', '#')
    bool visited;
    Node* neighbors[6];
    Node* next;               // for BFS queue
};
```

- And then use a queue to perform the BFS, you must implement the following operations:

- `enqueue()`
- `dequeue();`
- `bool isEmpty();`

Input Format

Each test case begins with three integers H, R, and C ($1 \leq H, R, C \leq 30$), the number of levels, rows, and columns, respectively.

Then H blocks follow, each describing a single floor with R lines of C characters each. These characters represent:

- `S` —your starting position
- `E` —the exit
- `.` —open space (can move into)
- `#` —blocked room (impassable)

There is one empty line separating each floor.

The input ends when a line containing `0 0 0` is encountered.

Output Format

For each test case, output one of the following:
$Escaped\ in\ X\ minute(s)$ or $Trapped$!
Where `X` is the minimum number of minutes required to escape.

Sample Input

```
3 4 5
S....
.###.
.##..
###.#

#####
#####
##.##
##...

#####
#####
#.###
####E

1 3 3
S##
#E#
###

0 0 0
```

Sample Output

```
Escaped in 11 step(s).
Trapped!
```

3 Problem 3

The Card Master is playing with patterns of numbers on a circular table of cards. He places m cards in a circle, each marked with a positive integer. Beginning with the first card, he examines cards one by one in clockwise order.

He compares each current card with the previously examined card. Let x be the previous card's number and y be the current card. Let $g = \gcd(x, y)$, the greatest common divisor of x and y.

The Master remembers the last computed GCD value. If the GCD of the current pair is equal to the previously computed GCD value, and this GCD is greater than 1, he discards the current card y.

This process continues until no more cards can be discarded based on this rule. The Master records the order in which the cards are removed.

- You need to represent the cards as a doubly circular linked list using the following structure, or else you will receive a score of 0.
- You shouldn't modify the fields in Node or LinkedList. However, you can modify the values if you want.
- Traversing the card and constructing it with an array is not allowed.

```
struct Node {
    int order, num;
    struct Node *next, *prev;
};

struct LinkedList {
    struct Node *head;
};
```

Input Format

- The first line contains m $(1 \leq m \leq 10^3)$ —the number of cards.
- The second line contains m integers $a_1, a_2, \ldots, a_m$ $(1 \leq a_i \leq 10^6)$ —the values on the cards.

Output Format

For each test case, output one line:

- First print an integer k —the number of discarded cards.
- Then print k integers —the indices of discarded cards, in order of removal.

Sample Input

```
2
6
12  18  24  30  40  50
4
7  14  21  28
```

Sample Output

```
2  3  4
3  3  2  4
```