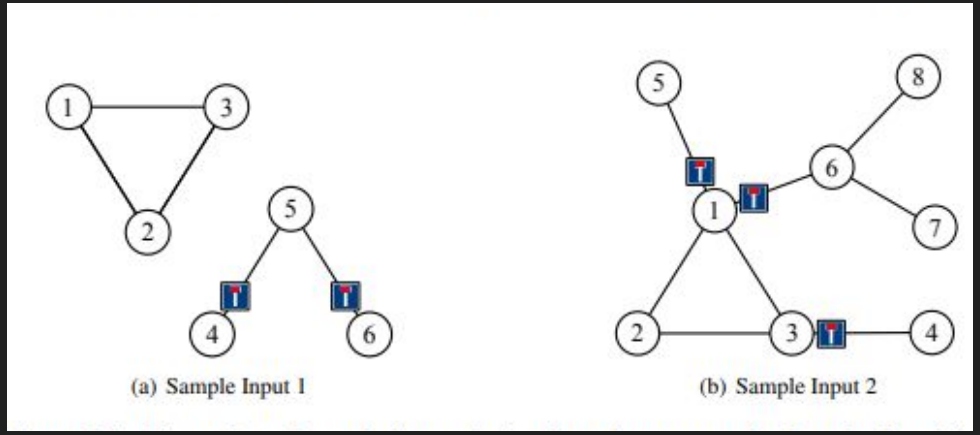
Problem E
Dead End Detector

Bella Mancini, Jon Mayer, Ryan Ozzello

Problem Overview

- Road map is a collection of locations connected by two way streets.
- Goal is to place dead end signs to avoid U-turns.
- A dead end sign goes at each vertex on an edge where a U-turn is needed to return back to that spot.
- Takes a list of tuples (the edges) as the input, will output an (v,w) where v is the is the entrance to the road (edge) where the sign should be located.



Example Inputs and Outputs

Sample Input 1

```
6 5        #Vertices #Edges
1 2        Edge1 (v,w)
1 3        Edge2 (v,w)
2 3        ...
4 5
5 6
```

Sample Output 1

```
2          #Signs
4 5        Sign1 from v to w
6 5        Sign2 from v to w
           ...
```

Sample Input 2

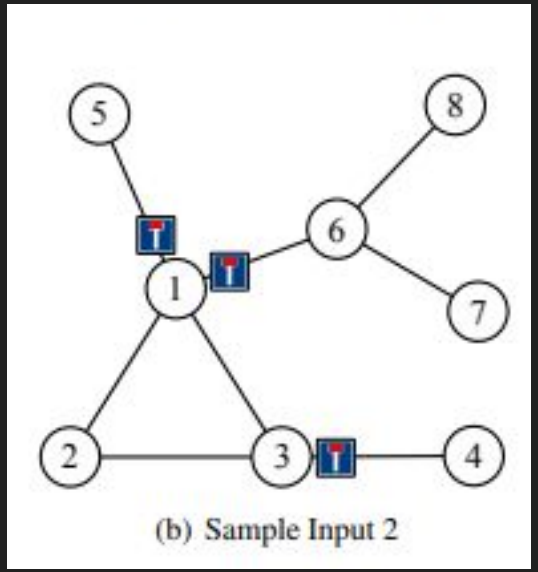
```
8 8
1 2
1 3
2 3
3 4
1 5
1 6
6 7
6 8
```

Sample Output 2

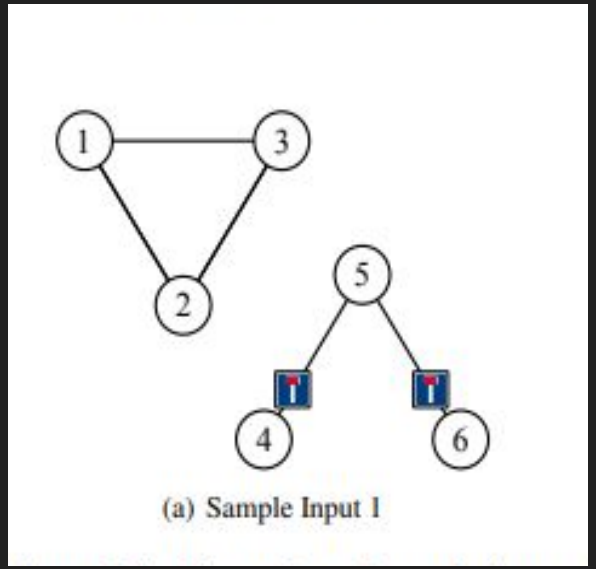
```
3
1 5
1 6
3 4
```

Approach: Trimming the Graph

- Dead end sign will either be 1) just outside of a cycle in a graph, 2) at the end of a branch (leaf nodes), or 3) there won't be any signs.
- Dead end signs will be placed somewhere on the **branches** of the graph.
- Trim the graph leaf-node by leaf-node, keeping track of each step.



(b) Sample Input 2

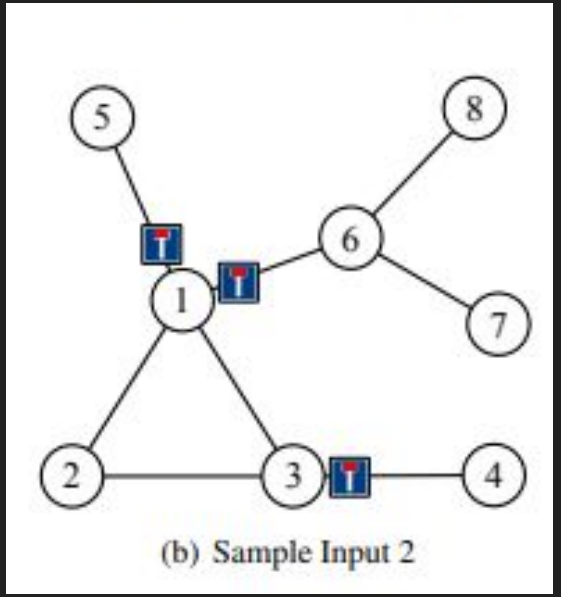


(a) Sample Input 1

Approach: Details

Three main stages:

1. Get the **edge count at each vertex** and store the counts in a mutable Map(“Vertex” -> Count).
2. **Trim the graph keeping track** of the which edges are removed at which step.
3. Detect cases and corner cases to **simplify the problem to the 3 possible sign placements**.

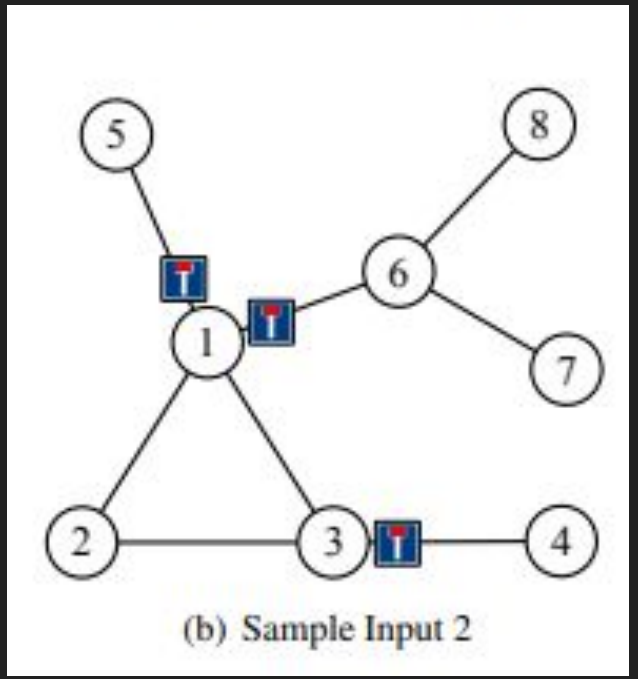


Example (“Normal”): cycles and branches

Count edges of each vertex:
scala.collection.mutable.Map(((1 -> 4), (2 -> 2), (3 -> 3), (4 -> 1), (5 -> 1), (6 -> 3), (7 -> 1), (8 -> 1))

Trim and keep track:
Steps: List(
Step 1: List((5,1), (8,6), (7,6), (4,3)) ,
Step 2: List((6,1))

Detect: Any edges left over (Yes)? Steps empty (No)? Connected (Yes)? So, what branch edges are on the cycle?
List(
Step 1: List((5,1), (8,6), (7,6), (4,3)) ,
Step 2: List((6,1)
)



(b) Sample Input 2

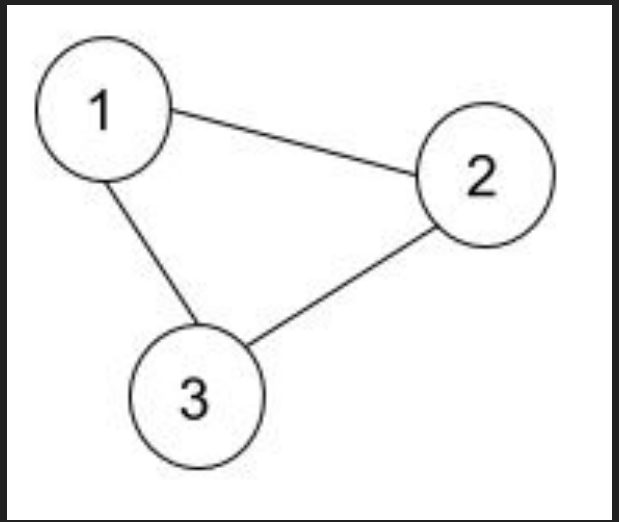
Example (corner case): cycles only

Count edges of each vertex:
scala.collection.mutable.Map(((1 -> 2), (2 -> 2), (3 -> 2))

Trim and keep track:
Steps: List()

Detect: Any edges left over (Yes)? Steps empty (Yes)? Connected (Yes)? So it is a...

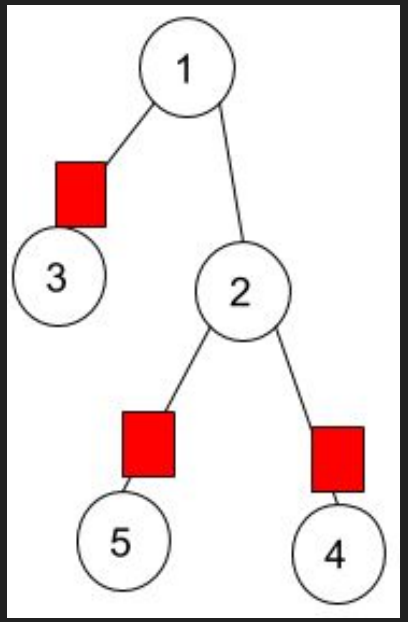
- cycle or empty graph since nothing trimmed



Example (corner case): branches only (tree)

Count edges of each vertex:
scala.collection.mutable.Map(((1 -> 2), (2 -> 3), (3 -> 1), (4 -> 1), (5 -> 1))

Trim and keep track:
Steps: List(
Step 1: List((3,1), (5,2), (4,2)) ,
Step 2: List((2,1))
Detect: Any edges left over (No)? Connected (Yes)? So, what edges are on a leaf?
List(
Step 1: List((3,1), (5,2), (4,2)) ,
Step 2: List((2,1)
)

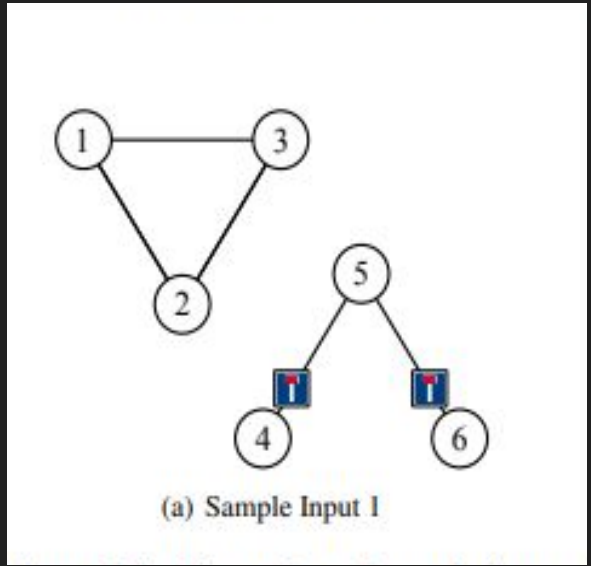


Example (corner case): disconnected graph

Count edges of each vertex:
scala.collection.mutable.Map(((1 -> 2), (2 -> 2), (3 -> 2), (4 -> 1), (5 -> 2), (6 -> 1))

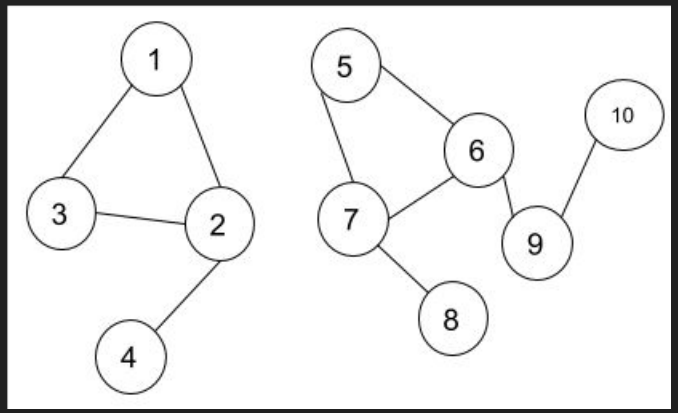
Trim and keep track:
List(
Step 1: List((4,5), (6,5)))
Detect: Any edges left over (Yes)? Steps empty (No)? Connected (No)? So...

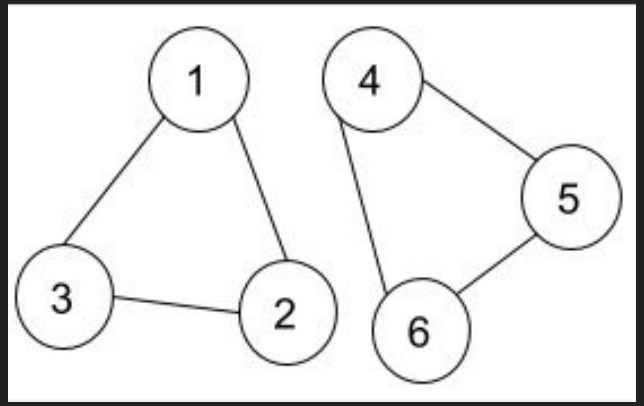
- Cycle Vertices: List(1,2,3)
- Find separate connected graphs and treat accordingly.

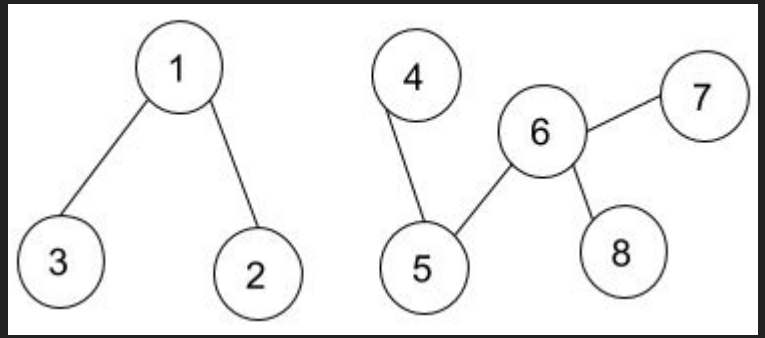


(a) Sample Input 1

Cases we don’t care about are disconnected graphs that contain connected graphs of the same type.



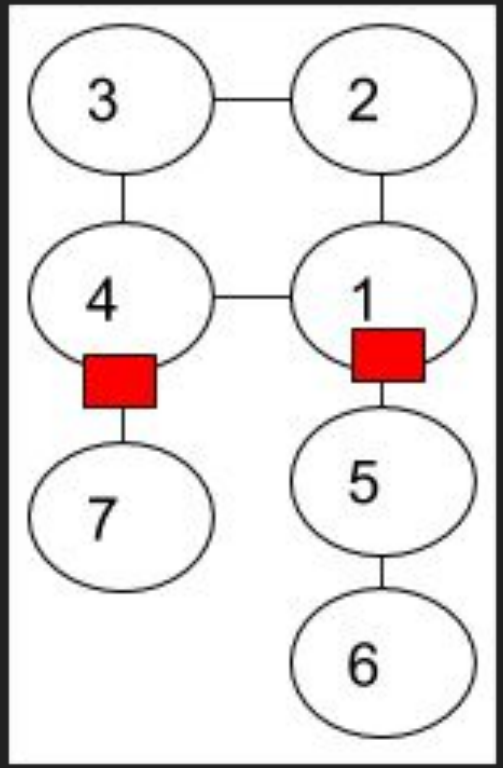




Results: a cycle with branches

Input

7 7
1 2
2 3
3 4
4 1
5 1
6 5
7 4



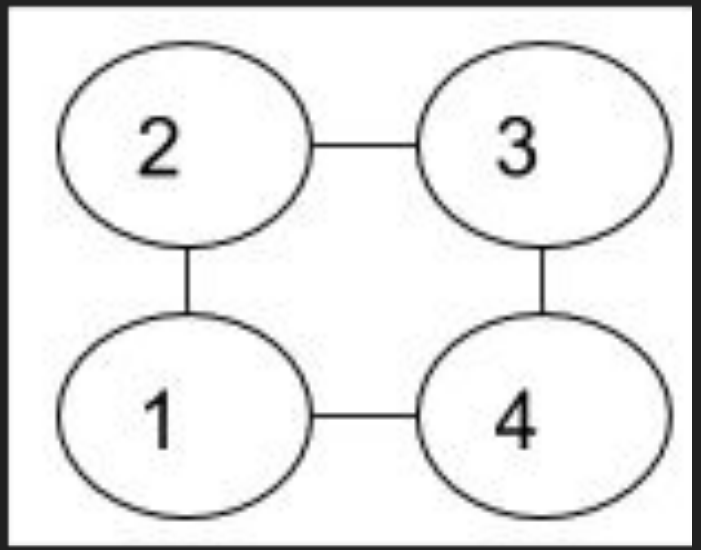
Output

2
4 7
1 5

Results: a cycle

Input

4 4
1 2
2 3
3 4
4 1



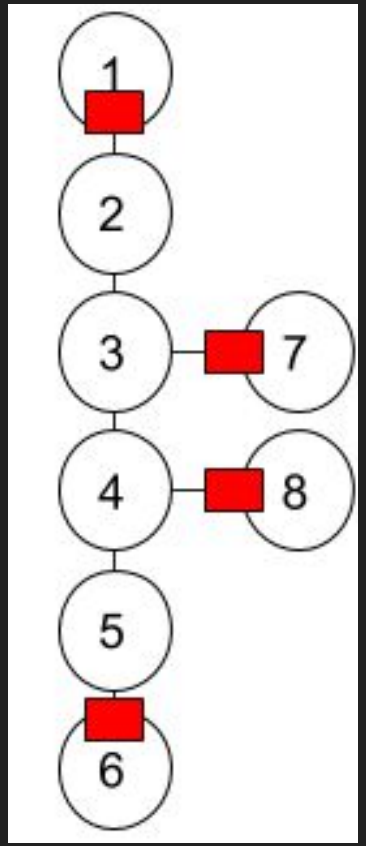
Output

0

Results: a tree

Input

8 7
1 2
2 3
3 4
4 5
5 6
3 7
4 8



Output

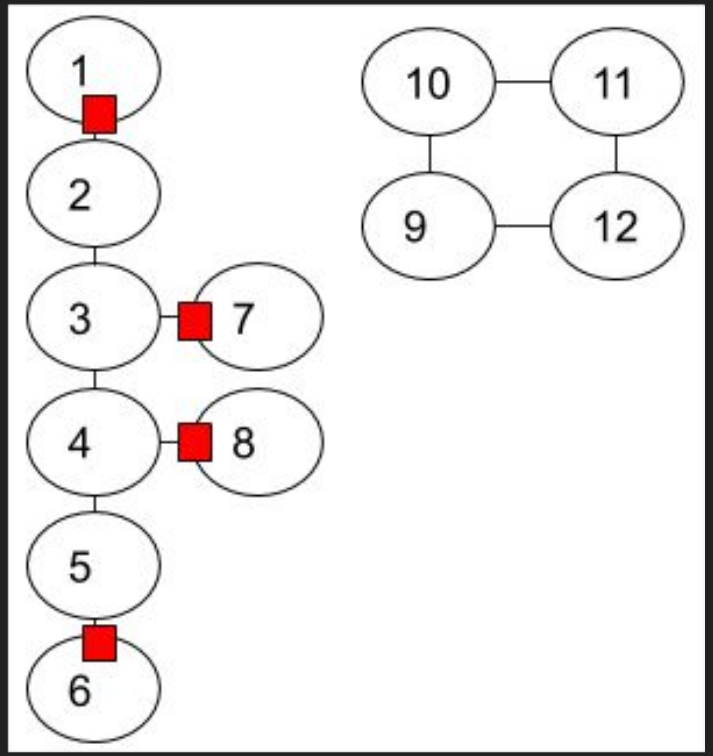
4
8 4
7 3
1 2
6 5

Results: A disconnected graph with a tree and cycle

Input

12 11
1 2
2 3
3 4
4 5
5 6
3 7
4 8
...

9 10
10 11
11 12
12 9



Output

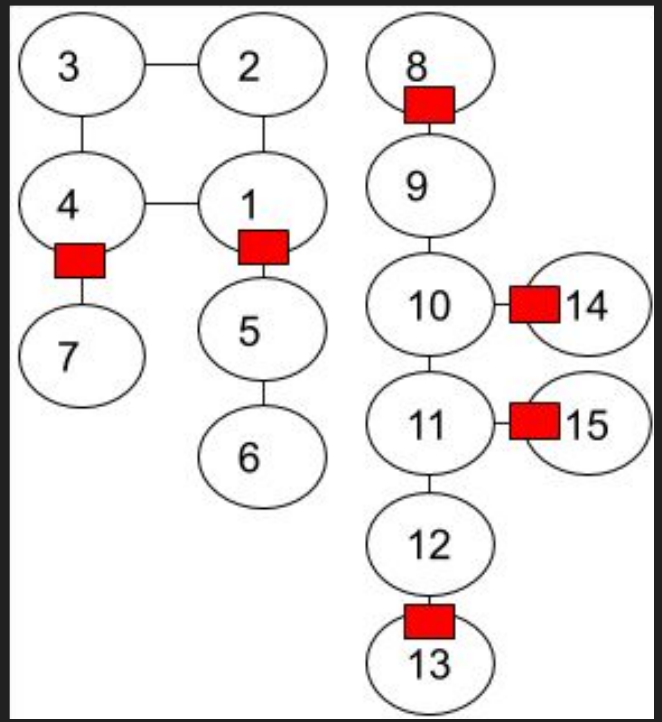
4
8 4
7 3
1 2
6 5

Results: A tree and a cycle with branches

Input

15 14
1 2
2 3
3 4
4 1
5 1
6 5
7 4
...

8 9
9 10
10 11
11 12
12 13
10 14
11 15



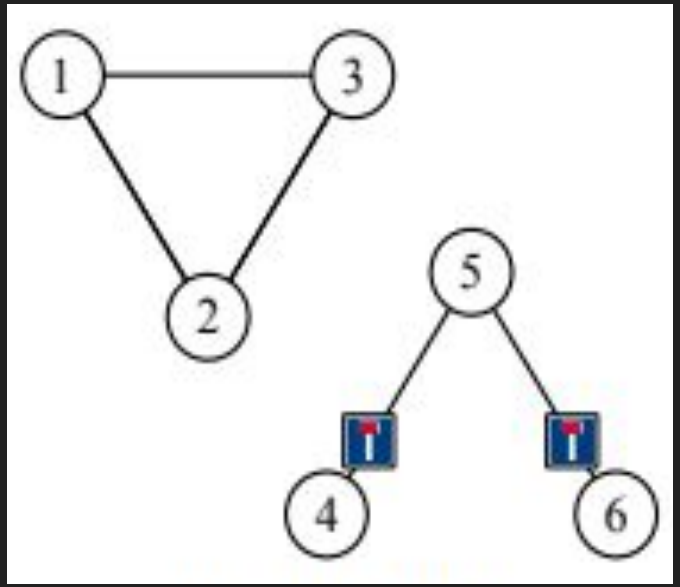
Output

6
4 7
1 5
8 9
14 10
13 12
15 11

Results: a disconnected graph with a cycle and a tree

Input

6 5
1 2
1 3
2 3
4 5
5 6



Output

2
4 5
6 5