802. Find Eventual Safe States
We will check use dfs to check each node is safe and tag it as 1, if not we will tag it as 2, then for later nodes that could access those nodes, they will become safe or unsafe nodes accordingly. TC is O(E+V), SC is O(E+V)

```
class Solution:
    def eventualSafeNodes(self, graph: List[List[int]]) -> List[int]:
        res = []
        if not graph or not graph[0]:
          return res

        color = [0] * len(graph)
        def dfs(i):
          if color[i] > 0:
            return color[i] == 1
          color[i] = 2
          for n_i in graph[i]:
            if not dfs(n_i):
              return False
          color[i] = 1
          return True

        for i in range(len(graph)):
          dfs(i)
        for idx, v in enumerate(color):
          if v == 1:
            res.append(idx)
        return res
```

399. Evaluate Division
We will use bfs and deque to solve this question. TC is O(n^2*m), SC is O(n*n)

```
from collections import defaultdict, deque
class Solution:
    def calcEquation(self, equations: List[List[str]], values: List[float], queries: List[List[str]]) ->
List[float]:
        memo = defaultdict(lambda: defaultdict(int))
        res = []

        def bfs(start, target):
          dq = deque()
          dq.append(start)
          visited = set([start[0]])
          while dq:
            s, cur = dq.popleft()
```

```
            visited.add(s)
            for i in memo[s]:
              if i in visited:
                continue
              if i == target:
                return cur * memo[s][i]
              else:
                dq.append([i, cur * memo[s][i]])
          return -1.0

        for (a, b), v in zip(equations, values):
          memo[a][b] = v
          memo[b][a] = 1.0 / v

        for s, e in queries:
          if s not in memo or e not in memo:
            res.append(-1.0)
          else:
            if s == e:
              res.append(1.0)
            else:
              res.append(bfs([s, 1], e))
        return res
```

990. Satisfiability of Equality Equations
Go through the code twice and first for equal using union find and second for not equal. TC is O(n^2)

```
from collections import defaultdict
class Solution:
    def equationsPossible(self, equations: List[str]) -> bool:
        parents = {}
        not_parents = defaultdict(set)
        def findParent(i):
          cur = i
          while i in parents and parents[i] != i:
            i = parents[i]
          parents[cur] = i
          return i

        for e in equations:
          if e[1] == '=':
            a_parent = findParent(e[0])
            b_parent = findParent(e[3])
```

```
            if a_parent != b_parent:
              parents[a_parent] = b_parent

        for e in equations:
          if e[1] == '!':
            a_parent = findParent(e[0])
            b_parent = findParent(e[3])
            if a_parent == b_parent:
              return False
        return True
```

35. Search Insert Position

We will use binary search to get this index.

```
from bisect import *
class Solution:
    def searchInsert(self, nums: List[int], target: int) -> int:
        return bisect_left(nums, target)
```

34. Find First and Last Position of Element in Sorted Array

The same as the last one, we will use binary search. TC is O(logn)

```
from bisect import *
class Solution:
    def searchRange(self, nums: List[int], target: int) -> List[int]:
        res = []
        if not nums:
          return [-1, -1]
        idx1 = bisect_left(nums, target)
        idx2 = bisect_right(nums, target) - 1
        length = len(nums)
        if idx1 < length and nums[idx1] == target:
          res.append(idx1)
        else:
          res.append(-1)

        if idx2 < length and nums[idx2] == target:
          res.append(idx2)
        else:
          res.append(-1)
        return res
```