```
import argparse
import networkx as nx
import random
import matplotlib.pyplot as plt
from networkx.utils import arbitrary_element
from satispy import Variable, Cnf
from satispy.solver import Minisat

"""
==========================================================================
  Complete the following function.
==========================================================================
"""

def solve(num_wizards, num_constraints, wizards, constraints):
    """
    Write your algorithm here.
    Input:
        num_wizards: Number of wizards
        num_constraints: Number of constraints
        wizards: An array of wizard names, in no particular order
        constraints: A 2D-array of constraints,
                     where constraints[0] may take the form ['A', 'B',
'C']i

    Output:
        An array of wizard names in the ordering your algorithm
returns
    """
    SATDict = {}
    #SATDict has key: tuple, value: Variable(tuple) <-- this just
returns the name but sets it up for SAT
    exp = Variable('empty')
    solver = Minisat()
    for con in constraints:
        tups = [(con[0], con[1]), (con[1], con[0]), (con[0], con[2]),
(con[2], con[0]), (con[1], con[2]), (con[2], con[1])]
        for t in tups:
            if t not in list(SATDict.keys()):
                SATDict[t] = Variable(t)

        # AB = SATDict[(con[0], con[1])]
        # BA = SATDict[(con[1], con[0])]
        # BC = SATDict[(con[1], con[2])]
        # CB = SATDict[(con[2], con[1])]
        # AC = SATDict[(con[0], con[2])]
        # CA = SATDict[(con[2], con[0])]
        # (AB & AC & BC) | (AB & CA & CB) | (BA & AC & BC) | (BA & CA
& CB) & (AB ^ BA) & (AC ^ CA) & (CB ^ BC)
```

```
        texp = (((SATDict[(con[0], con[1])] & SATDict[(con[0],
con[2])] & SATDict[(con[1], con[2])]) | \
                (SATDict[(con[0], con[1])] & SATDict[(con[2], con[1])]
& SATDict[(con[2], con[1])]) | \
                (SATDict[(con[1], con[0])] & SATDict[(con[0], con[2])]
& SATDict[(con[1], con[2])]) | \
                (SATDict[(con[1], con[0])] & SATDict[(con[2], con[0])]
& SATDict[(con[2], con[1])])) & \
                ((SATDict[(con[0], con[1])] ^ SATDict[(con[1],
con[0])]) & \
                (SATDict[(con[1], con[2])] ^ SATDict[(con[2],
con[1])]) & \
                (SATDict[(con[0], con[2])] ^ SATDict[(con[2],
con[0])])))

        exp = exp & texp
    for w1 in wizards:
        for w2 in wizards:
            for w3 in wizards:
                tups = [(w1, w2), (w2, w3), (w1, w3)]
                for t in tups:
                    if t not in list(SATDict.keys()):
                        SATDict[t] = Variable(t)
                texp2 = ((SATDict[tups[0]] & SATDict[tups[1]]) >>
SATDict[tups[2]])
                exp = exp & texp2
    solution = solver.solve(exp)
    """
    need to process solution before feeding to top sort
    solution[tuple] will return true or false assignment
    """
    solvedSAT = []
    for key in SATDict.keys():
        if solution[SATDict[key]]:
            solvedSAT.append(key)
    sol = topologicalSort(solvedSAT)
    print("fuck me")
    return sol

def topologicalSort(solvedSAT):
    G = nx.DiGraph()
    for var in solvedSAT:
        s, t = var
        try:
            G.add_edge(s, t)
            cycle = nx.find_cycle(G, t)
            G.remove_edge(s, t)
        except nx.NetworkXNoCycle:
            continue;
    generator = nx.topological_sort(G)
```

```
    final = []
    for wiz in generator:
        final.append(wiz)
    return final




"""
================================================================================
   No need to change any code below this line
================================================================================
"""

def read_input(filename):
    with open(filename) as f:
        num_wizards = int(f.readline())
        num_constraints = int(f.readline())
        constraints = []
        wizards = set()
        for _ in range(num_constraints):
            c = f.readline().split()
            constraints.append(c)
            for w in c:
                wizards.add(w)

    wizards = list(wizards)
    return num_wizards, num_constraints, wizards, constraints

def write_output(filename, solution):
    with open(filename, "w") as f:
        for wizard in solution:
            f.write("{0} ".format(wizard))

if __name__=="__main__":
    parser = argparse.ArgumentParser(description = "Constraint
Solver.")
    parser.add_argument("input_file", type=str, help = "___.in")
    parser.add_argument("output_file", type=str, help = "___.out")
    args = parser.parse_args()

    num_wizards, num_constraints, wizards, constraints =
read_input(args.input_file)
    solution = solve(num_wizards, num_constraints, wizards,
constraints)
    write_output(args.output_file, solution)
```