```
#!/bin/python3

import math
import os
import random
import re
import sys

#
# Complete the 'familyTree' function below.
#
# The function is expected to return a STRING.
# The function accepts following parameters:
#  1. 2D_STRING_ARRAY parent_child
#  2. STRING request
#

parent = {}
parents = []
spouse = {}

def union(a, b):
    return list(set(a + b))

def intersect(a, b):
    return any(x in b for x in a)

def isSpouse(a, b):
    if a not in parents or b not in parents:
```

```
        return False
    return intersect(parent[a], parent[b])

def isParent(a, b):
    for p in parent:
        if p == a and any(b == i or spouse[b] == i for i in parent[p]):
            return True
    return False

def isGrandParent(a, b):
    if a not in parents:
        return False
    for p in parent[a]:
        if isParent(p, b):
            return True
    return False

def isGreatGrandParent(a, b):
    if a not in parents:
        return False
    for p in parent[a]:
        if isGrandParent(p, b):
            return True
    return False

def isSibling(a, b):
    for p in parent:
        if (a in parent[p] or spouse[a] in parent[p]) and (b in parent[p] or spouse[b] in parent[p]):
            return True
```

```
    return False

def isCousin(a, b):
    for p in getParents(a):
        for q in getParents(b):
            if isSibling(p, q):
                return True
    return False

def isSecondCousin(a, b):
    for p in getParents(a):
        for q in getParents(b):
            if isCousin(p, q):
                return True
    return False

def isPibling(a, b):
    for p in getParents(b):
        if isSibling(a, p):
            return True
    return False

def isGrandPibling(a, b):
    for p in parents:
        if isGrandParent(p, b) and isSibling(p, a):
            return True
    return False

def getParents(a):
```

```
    return [p for p in parent if isParent(p, a)]

def familyTree(parent_child, request):
    for p in parent_child:
        if p[0] not in [k for k in parent]:
            parent[p[0]] = []
        parent[p[0]].append(p[1])
        spouse[p[0]], spouse[p[1]] = "", ""

    parents.extend([k for k in parent])

    [a, b] = request.split()

    if isSpouse(a, b):
        return "spouse"

    for i in range(len(parents)):
        for j in range(i+1, len(parents)):
            p, q = parents[i], parents[j]
            if isSpouse(p, q):
                u = union(parent[p], parent[q])
                parent[p], parent[q] = u, u
                spouse[p], spouse[q] = q, p

    if isParent(a, b):
        return "child"

    if isParent(b, a):
        return "parent"
```

```
if isGrandParent(a, b):
    return "grandchild"

if isGrandParent(b, a):
    return "grandparent"

if isGreatGrandParent(a, b):
    return "great-grandchild"

if isGreatGrandParent(b, a):
    return "great-grandparent"

if isSibling(a, b):
    return "sibling"

if isCousin(a, b):
    return "cousin"

if isSecondCousin(a, b):
    return "second cousin"

if isPibling(a, b):
    return "nibling"

if isPibling(b, a):
    return "pibling"

if isGrandPibling(a, b):
```

```
        return "grandnibling"

    if isGrandPibling(b, a):
        return "grandpibling"



if __name__ == '__main__':
    fptr = open(os.environ['OUTPUT_PATH'], 'w')

    numInputs = int(input().strip())

    parent_child = []

    for _ in range(numInputs):
        parent_child.append(input().rstrip().split())

    request = input()

    result = familyTree(parent_child, request)

    fptr.write(result + '\n')

    fptr.close()
```