```
#include <bits/stdc++.h>

using namespace std;



string familyTree(vector<vector<string>> parent_child, string request) {
    int n = parent_child.size(), m;

    unordered_map<string, int> index;
    vector<string> names;

    for(int i = 0, ix = 0; i < n; ++i) {
        if(index.find(parent_child[i][0]) == index.end()) {
            index[parent_child[i][0]] = ix++;
            names.push_back(parent_child[i][0]);
        }
        if(index.find(parent_child[i][1]) == index.end()) {
            index[parent_child[i][1]] = ix++;
            names.push_back(parent_child[i][1]);
        }
    }

    m = names.size();

    int tree[m][m], spouse[m];
    pair<int, int> relationship[m];

    memset(tree, 0, sizeof(tree));
    memset(spouse, -1, sizeof(spouse));
```

```
for(int i = 0; i < m; ++i)
   relationship[i] = make_pair(0, 0);

for(int i = 0; i < n; ++i) {
   tree[index[parent_child[i][0]]][index[parent_child[i][1]]] = 1;
   tree[index[parent_child[i][1]]][index[parent_child[i][0]]] = -1;
}

// find spouses...
queue<int> q;
set<int> s;
vector<int> p;
int f;
q.push(0);
s.insert(0);

while(!q.empty()) {
   f = q.front();
   q.pop();

   p.clear();
   for(int i = 0; i < m; ++i) {
      if(tree[f][i] == -1) {
         p.push_back(i);
      }

      if(!tree[f][i] || s.find(i) != s.end()) {
         continue;
      }
```

```
        q.push(i);
        s.insert(i);
    }

    if(p.size() > 1) {
        spouse[p[0]] = p[1];
        spouse[p[1]] = p[0];
    }
}

string start = request.substr(0, request.find(' ')), target = request.substr(request.find(' ') + 1);

s.clear();
q.push(index[start]);
s.insert(index[start]);

while(!q.empty()) {
    f = q.front();
    q.pop();

    for(int i = 0; i < m; ++i) {
        if(!tree[f][i] || s.find(i) != s.end())
            continue;

        if(tree[f][i] < 0) {
            relationship[i] = make_pair(relationship[f].first-1, relationship[f].second);
        }else{
            relationship[i] = make_pair(relationship[f].first+1, relationship[f].second+1);
```

```
        }

        q.push(i);
        s.insert(i);
    }

    if(spouse[f] != -1 && s.find(spouse[f]) == s.end()) {
        q.push(spouse[f]);
        s.insert(spouse[f]);
        relationship[spouse[f]] = relationship[f];
    }
}

pair<int, int> t = relationship[index[target]];
if(t.first == 0) {
    // sibling, cousin, second cousin...
    if(t.second == 1) {
        return "sibling";
    }else if(t.second == 2) {
        return "cousin";
    }else if(t.second == 3) {
        return "second cousin";
    }
}else if(t.first == 1) {
    // child, nibling
    if(t.second == 1) {
        return "child";
    }else if(t.second == 2) {
        return "nibling";
```

```
    }
  }else if(t.first == 2) {
    // grandchild, grandnibling
    if(t.second == 2) {
      return "grandchild";
    }else if(t.second == 3) {
      return "grandnibling";
    }
  }else if(t.first == 3) {
    // great-grandchild, great-grandnibling
    if(t.second == 3) {
      return "great-grandchild";
    }else if(t.second == 4) {
      return "great-grandnibling";
    }
  }else if(t.first == -1) {
    // parent, pibling
    if(t.second == 0) {
      return "parent";
    }else if(t.second == 1) {
      return "pibling";
    }
  }else if(t.first == -2) {
    // grandparent, grandpibling
    if(t.second == 0) {
      return "grandparent";
    }else if(t.second == 1) {
      return "grandpibling";
    }
```

```
    }else if(t.first == -3) {
        // great-grandparent, great-grandpibling
        if(t.second == 0) {
            return "great-grandparent";
        }else if(t.second == 1) {
            return "great-grandpibling";
        }
    }

    return "spouse";
}

int main() {
    int n;
    cin >> n;
    vector<vector<string>> pc (n);

    for(int i = 0; i < n; ++i) {
        pc[i].resize(2);
        cin >> pc[i][0] >> pc[i][1];
    }

    string a, b, request;
    cin >> a >> b;
    request = a + " " + b;

    ofstream fout(getenv("OUTPUT_PATH"));

    fout << familyTree(pc, request) << endl;
```

```
    fout.close();


    return 0;
}
```