```
import java.io.*;
import java.math.*;
import java.security.*;
import java.text.*;
import java.util.*;
import java.util.concurrent.*;
import java.util.function.*;
import java.util.regex.*;
import java.util.stream.*;
import static java.util.stream.Collectors.joining;
import static java.util.stream.Collectors.toList;

class Result {

    /*
     * 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
     */

    public static String familyTree(List<List<String>> parent_child, String request) {
        Set<String> people = new HashSet<>();

        Map<String, Set<String>> spouses = new HashMap<>();
        Map<String, Set<String>> children = new HashMap<>();
        Map<String, Set<String>> parents = new HashMap<>();
```

```
Map<String, Set<String>> grandchildren = new HashMap<>();
Map<String, Set<String>> grandparents = new HashMap<>();
Map<String, Set<String>> greatgrandchildren = new HashMap<>();
Map<String, Set<String>> greatgrandparents = new HashMap<>();
Map<String, Set<String>> siblings = new HashMap<>();
Map<String, Set<String>> cousins = new HashMap<>();
Map<String, Set<String>> secondcousins = new HashMap<>();
Map<String, Set<String>> piblings = new HashMap<>();
Map<String, Set<String>> grandpiblings = new HashMap<>();
Map<String, Set<String>> niblings = new HashMap<>();
Map<String, Set<String>> grandniblings = new HashMap<>();

parent_child.stream().forEach(people::addAll);
people.stream().forEach(person -> {
    spouses.put(person, new HashSet<>());
    children.put(person, new HashSet<>());
    parents.put(person, new HashSet<>());
    grandchildren.put(person, new HashSet<>());
    grandparents.put(person, new HashSet<>());
    greatgrandchildren.put(person, new HashSet<>());
    greatgrandparents.put(person, new HashSet<>());
    siblings.put(person, new HashSet<>());
    cousins.put(person, new HashSet<>());
    secondcousins.put(person, new HashSet<>());
    piblings.put(person, new HashSet<>());
    grandpiblings.put(person, new HashSet<>());
    niblings.put(person, new HashSet<>());
    grandniblings.put(person, new HashSet<>());
});
```

```
parent_child.stream()
   .forEach(pair -> children.get(pair.get(0)).add(pair.get(1)));
for (String person1 : people) {
   for (String person2 : people) {
      if (!person1.equals(person2) &&
         children.get(person1).parallelStream()
         .anyMatch(s -> children.get(person2).contains(s))) {
         children.get(person1).addAll(children.get(person2));
         children.get(person2).addAll(children.get(person1));
         spouses.get(person1).add(person2);
         spouses.get(person2).add(person1);
      }
   }
}

people.stream()
   .forEach(parent -> new HashSet<String>(children.get(parent)).stream()
         .forEach(child -> children.get(parent).addAll(spouses.get(child))));

people.stream()
   .forEach(parent -> children.get(parent).stream()
         .forEach(child -> parents.get(child).add(parent)));

people.stream()
   .forEach(grandparent -> children.get(grandparent).stream()
         .forEach(child -> grandchildren.get(grandparent).addAll(children.get(child))));

people.stream()
```

```
    .forEach(grandparent -> grandchildren.get(grandparent).stream()
            .forEach(grandchild -> grandparents.get(grandchild).add(grandparent)));

people.stream()
    .forEach(greatgrandparent -> grandchildren.get(greatgrandparent).stream()
            .forEach(grandchild ->
                    greatgrandchildren.get(greatgrandparent).addAll(children.get(grandchild))));

people.stream()
    .forEach(greatgrandparent -> greatgrandchildren.get(greatgrandparent).stream()
            .forEach(greatgrandchild -> greatgrandparents.get(greatgrandchild).add(greatgrandparent)));

people.stream()
    .forEach(parent -> children.get(parent).stream()
            .forEach(child -> siblings.get(child).addAll(children.get(parent))));

people.stream()
    .forEach(grandparent -> grandchildren.get(grandparent).stream()
            .forEach(grandchild ->
                    cousins.get(grandchild).addAll(grandchildren.get(grandparent))));

people.stream()
    .forEach(greatgrandparent -> greatgrandchildren.get(greatgrandparent).stream()
            .forEach(greatgrandchild ->
                    secondcousins.get(greatgrandchild).addAll(greatgrandchildren.get(greatgrandparent))));

people.stream().forEach(person -> {
    siblings.get(person).remove(person);
    siblings.get(person).removeAll(spouses.get(person));
```

```
    cousins.get(person).remove(person);
    cousins.get(person).removeAll(spouses.get(person));
    cousins.get(person).removeAll(siblings.get(person));

    secondcousins.get(person).remove(person);
    secondcousins.get(person).removeAll(spouses.get(person));
    secondcousins.get(person).removeAll(siblings.get(person));
    secondcousins.get(person).removeAll(cousins.get(person));
});

people.stream()
    .forEach(parent -> children.get(parent).stream()
            .forEach(child -> piblings.get(child).addAll(siblings.get(parent))));

people.stream()
    .forEach(grandparent -> grandchildren.get(grandparent).stream()
            .forEach(grandchild ->
                    grandpiblings.get(grandchild).addAll(siblings.get(grandparent))));

people.stream()
    .forEach(nibling -> piblings.get(nibling).stream()
             .forEach(pibling -> niblings.get(pibling).add(nibling)));

people.stream()
    .forEach(grandnibling -> grandpiblings.get(grandnibling).stream()
             .forEach(grandpibling -> grandniblings.get(grandpibling).add(grandnibling)));

// System.out.println("Spouses: " + spouses);
```

```
// System.out.println("Children: " + children);
// System.out.println("Parents: " + parents);
// System.out.println("Grandchildren: " + grandchildren);
// System.out.println("Grandparents: " + grandparents);
// System.out.println("Great Grandchildren: " + greatgrandchildren);
// System.out.println("Great Grandparents: " + greatgrandparents);
// System.out.println("Siblings: " + siblings);
// System.out.println("Cousins: " + cousins);
// System.out.println("Second Cousins: " + secondcousins);
// System.out.println("Piblings: " + piblings);
// System.out.println("Grandpiblings: " + grandpiblings);
// System.out.println("Niblings: " + niblings);
// System.out.println("Grandniblings: " + grandniblings);

String[] names = request.split("\\s+");
String key = names[0];
String value = names[1];

if (spouses.get(key).contains(value)) {
   return "spouse";
}
if (children.get(key).contains(value)) {
   return "child";
}
if (parents.get(key).contains(value)) {
   return "parent";
}
if (grandchildren.get(key).contains(value)) {
   return "grandchild";
```

```
}
if (grandparents.get(key).contains(value)) {
   return "grandparent";
}
if (greatgrandchildren.get(key).contains(value)) {
   return "great-grandchild";
}
if (greatgrandparents.get(key).contains(value)) {
   return "great-grandparent";
}
if (siblings.get(key).contains(value)) {
   return "sibling";
}
if (cousins.get(key).contains(value)) {
   return "cousin";
}
if (secondcousins.get(key).contains(value)) {
   return "second cousin";
}
if (piblings.get(key).contains(value)) {
   return "pibling";
}
if (grandpiblings.get(key).contains(value)) {
   return "grandpibling";
}
if (niblings.get(key).contains(value)) {
   return "nibling";
}
if (grandniblings.get(key).contains(value)) {
```

```
            return "grandnibling";
        }

        return "sibling";
    }

}

public class Solution {
    public static void main(String[] args) throws IOException {
        BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(System.in));
        BufferedWriter bufferedWriter = new BufferedWriter(new
FileWriter(System.getenv("OUTPUT_PATH")));

        int numInputs = Integer.parseInt(bufferedReader.readLine().trim());

        List<List<String>> parent_child = new ArrayList<>();

        IntStream.range(0, numInputs).forEach(i -> {
            try {
                parent_child.add(
                    Stream.of(bufferedReader.readLine().replaceAll("\\s+$", "").split(" "))
                        .collect(toList())
                );
            } catch (IOException ex) {
                throw new RuntimeException(ex);
            }
        });
```

```
        String request = bufferedReader.readLine();

        String result = Result.familyTree(parent_child, request);

        bufferedWriter.write(result);
        bufferedWriter.newLine();

        bufferedReader.close();
        bufferedWriter.close();
    }
}
```