Вам дано описание наследования классов в следующем формате.
<имя класса 1> : <имя класса 2> <имя класса 3> ... <имя класса k>
Это означает, что **класс 1** отнаследован от **класса 2**, **класса 3**, и т.

Или эквивалентно записи:

```
class Class1(Class2, Class3 ... ClassK):

    pass
```

Класс **A** является **прямым предком** класса **B**, если **B** отнаследован от **A:**

```
class B(A): pass
```

Класс **A** является **предком** класса **B**, если

- **A** = **B**;
- **A** - прямой предок **B**
- существует такой класс **C**, что **C** - прямой предок **B** и **A** - предок **C**
    - Например:
    - ```
      class B(A):
          pass

      class C(B):
          pass

      # A -- предок C
      ```

    Вам необходимо отвечать на запросы, является ли один класс предком другого класса
    - **Важное примечание:**
    - Создавать классы не требуется.
    - Мы просим вас промоделировать этот процесс, и понять существует ли путь от одного класса до другого.

Формат входных данных

- В первой строке входных данных содержится целое число **n** - число классов.
- В следующих **n** строках содержится описание наследования классов. В **i**-й строке указано от каких классов наследуется **i**-й класс. Обратите внимание, что класс может ни от кого не наследоваться. Гарантируется, что класс не наследуется сам от себя (прямо или косвенно), что класс не наследуется явно от одного класса более одного раза.
- В следующей строке содержится число **q** - количество запросов.
- В следующих **q** строках содержится описание запросов в формате **<имя класса 1> <имя класса 2>**.
  Имя класса – строка, состоящая из символов латинского алфавита, длины не более 50.

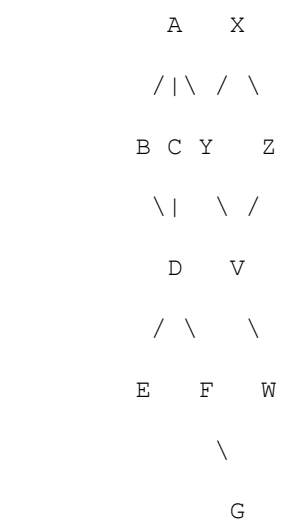
Формат выходных данных

- Для каждого запроса выведите в отдельной строке слово "**Yes**", если **класс 1** является предком **класса 2**, и "**No**", если не является.
      ```

- **Sample Input:**
- 4
- A
- B : A
- C : A
- D : B C
- 4
- A B
- B D
- C D
- D A

- **Sample Output:**
- Yes
- Yes
- Yes
- No

Отладил свой алгоритм на следующих входных данных




```
lst_mro = [  # список введённых строк

    'G : F',  # сначала отнаследуем от F, потом его объявим, корректный алгоритм все равно правильно обойдёт граф, независимо что было раньше: наследование или объявление

    'A',

    'B : A',

    'C : A',

    'D : B C',

    'E : D',

    'F : D',

    # а теперь другая ветка наследования
```

```
    'X',

    'Y : X A',  # свяжем две ветки наследования для проверки, обошла ли
рекурсия предков Z и предков Y в поисках A

    'Z : X',

    'V : Z Y',

    'W : V',

]



lst_q = [  # список введённых запросов

    'A G',      # Yes   # A предок G через B/C, D, F

    'A Z',      # No    # Y потомок A, но не Y

    'A W',      # Yes   # A предок W через Y, V

    'X W',      # Yes   # X предок W через Y, V

    'X QWE',    # No    # нет такого класса QWE

    'A X',      # No    # классы есть, но они нет родства :)

    'X X',      # Yes   # родитель он же потомок

    '1 1',      # No    # несуществующий класс

]
```

Листинг

```
n = int(input())
lst_in = []
lst_qq = []
for i in range(n):
    lst_in.append(input())
q = int(input())
for i in range(q):
    lst_qq.append(input())
def split_and_insert(element):
    items = element.split(':')  # Разделение элемента на две части
    prefix = items[0]  # Часть элемента перед символом ":"
    suffix = items[1].strip()  # Часть элемента после символа ":", очищенная
от лишних пробелов
    # Создание новых элементов списка
    new_elements = [f"{prefix}: {item}" for item in suffix.split()]
    return new_elements
def check(lst_mro, lst_q):
    ans = False
    query = lst_q.split(" ")
    j = 0
    while j < len(lst_mro):
        child = lst_mro[j].split(" : ")[0]
        if (query[0] == query[1] and (query[0] in [i[0] for i in lst_mro] or
query[0] in [i[-1] for i in lst_mro])):
            return True
        if query[1] == child and len(lst_mro[j]) > 1:
            query[1] = lst_mro[j][-1]
```

```
            if j != lst_mro.index(lst_mro[-1]) and child[0] == lst_mro[j +
1][0]:
                pos = lst_mro[j + 1].rpartition(" ")
                ans = check(lst_mro, query[0] + ' ' + pos[2])
            j = 0
        j = j + 1
    if ans:
        return True
    return ans
def graph(lst_mro, lst_q):
    for i in lst_mro:
        if i.count(" ") > 2:
            pos = split_and_insert(i)
            for j in pos:
                lst_mro.insert(lst_mro.index(i),  j)
            lst_mro.remove(i)
    for i in lst_q:
        if check(lst_mro, i):
            print("Yes")
        else:
            print("No")

graph(lst_in, lst_qq)
```