11869 SOAP Response

With increasing availability of broadband internet and significant progress in Web technology, Web Services are playing a crucial role in application development. The basic idea behind Web Service is, the server hosting the services can be queried and it will send back the response based on the query. Similar to all other forms of communication, a protocol must exist on how the messages should be communicated. One popular protocol is SOAP (Simple Object Access Protocol). As the name suggests, it deals with sending objects across client and server machine. Since objects are represented differently in different languages, the objects are sent in a common text based format known as XML. In this problem, we will be analyzing a simplified version of SOAP messages in XML format and evaluate queries based on the message.

The XML messages will be formatted as below:

i) Each line consists of either an opening tag or a closing tag.

ii) Each tag will have a name.

iii) Each opening tag can have zero or more properties. The properties will have a value assigned to them. These names and values will have at least one character. The property names in individual tags will be unique. See examples below for exact formatting.

iv) Closing tags will not have any properties.

v) Tags can be nested inside another tag.

Example XML message and description:

<one valid="true" active="alive">
<anestedtag>
</anestedtag>
</one>

In the above example, we have a tag named '*one*' which has two properties, namely '*valid*' and '*active*'. The property named '*valid*' has a value of '*true*' and '*active*' has a value of '*alive*'. Each property information is separated from each other by a single space and there is also a space after tag name if the tag has any property. No other space occurs in the XML text other than those mentioned earlier. There is a tag named '*anestedtag*' which is nested inside the tag named '*one*'. Note that, this tag doesn't have any properties defined. Closing tags contain the same name of the corresponding opening tag, but is preceded by the '/' character.

You will be given a simplified soap message, containing only start and end tags with property information embedded in the start tag as the above format. Then you will have some query with some property name under some tag. You need output the value of the property under the specified tags or report such property is undefined.

Input

The first line of input will be a positive integer $T \leq 10$, where T denotes the number of test cases. Each case starts with a positive integer $n \leq 1000$ and n will always be even. The next n lines each describes an opening or a closing tag. These lines will have at most 1000 characters. The last line of the input will always be the closing tag for the tag opened in the first line. No opening tag name will occur more than once. All input will be syntactically valid, that means all opening tags will also contain a corresponding closing tags in the appropriate line. The next line will contain a positive integer $q \leq 1000$. Each of the next q lines will describe a query. The query will consist of tag names separated by a dot (`.`) followed by few characters in the format '["$<$ *prop name* $>$"]', where $<$ *prop name* $>$ is one or more character forming a property name. See sample input for exact formatting.

All *names* and *values* in the input will consist of alphabets only.

Output

Each case of output will contain the case number in one line. Then there will be q lines, each corresponding to the property value given in the query. If that particular property does not exist, the output will be '`Undefined`' without the quotes. Note that, the input queries will always be syntactically valid, but they may refer to tags and properties that are not defined. If two tags are separated by a '`.`', it indicates that the tag to the right should be contained within the left tag. See sample **input/output** for further clarification. In output, there should not be any leading or trailing spaces in any of the line.

Sample Input

```
1
8
<a good="true" wants="no">
<b>
<c contest="running">
<d ballon="no">
</d>
</c>
</b>
</a>
4
a.b.c.d["ballon"]
a.c["contest"]
a.b.c["contest"]
c["contest"]
```

Sample Output

```
Case 1:
no
Undefined
running
Undefined
```