Front-end engineer testing

The front-end test is made of two parts:

- HTML+CSS+JS mastery,
- General programming skills.

We use Angular, React and Vue in our products, being Angular and Vue the most common, but you are free to use other frameworks or no frameworks at all.

HTML+CSS+JS Test

Our designers made a mockup and we want to see how it feels in a first actual HTML implementation. Your task is to recreate the contents of the `table_mock.png` file using HTML+CSS+JS:

- The delivery should be a single web page. The widgets in the mock-up- like the calendar, order header, search box, etc.- are not required to be interactive. Just ensure that the appearance is the same.
- Your index page should include the contents of the `html_segment.html` and use the global function `mockAPI` to simulate the fetch of the table data. You should use the response to draw the table. I.E., do not hardcode the table contents and, instead, iterate the mockAPI promise as if it was an actual remote API call.
- For less senior positions, the test wll contain a folder named `page_shell_sample`. You can use it to save time and just add the missing widgets.
- The left side menu options should display the selected style when in hover and focus states.
- Your usage of semantic HTML will be evaluated. Use the correct tags, roles and ARIA attributes.
- All the icons of the mock are FontAwesome icons.
- The main font of the mock is,

  ```
  font-size: 14px;
  font-family: 'Circular Std';
  color: #6F6F6F;
  ```

  The font home is https://github.com/elartix/circular-std, but you can use other secure fonts if you have any trouble installing it.
- You should provide a README.md file explaining how to install and see your deliverable. If you add any test, you should also explain how to run it in the README file.

General programming skills

For the general programming test, you can use any language that you feel confortable with. However, we use Typescript, JavaScript and Python as our main languages and using those will make easier for us to check the deliverable.

We want to add to our portfolio a Music Genealogy Microservice to find the nearest common ancestor of two music styles. For example, if we ask our system for the nearest common ancestor of Death Metal and Industrial Metal, it should return Thrash metal.

In a more formal spec, the microservice will be a function with this signature (in Typescript language):

```
function findAncestor(genreA: string, genreB: string): string
```

With the rule that if `genreA` is an ancestor of `genreB`, then it should return `genreA`. If `genreA` is a root element, no ancestors, the function should return `genreA`.

To make the exercise easier, the genres will be named with letters and they will have only a single ancestor. Also, the tree will be connected and have a single root element without any ancestor.

Your program should be able to read from `stdin` and output to `stdout` the response.

The input files are as follow:

- The first line will indicate the number `n` of items and the number `m` of questions.
- The next line will indicate the root element name.
- The next `n - 1` lines will indicate the pair element name and one of its descendants. E.g. `a b` indicates that the genre `a` is a parent of `b`.
- The last `m` lines will indicate the pair that we want to know their nearest common ancestor.

For example, the input

```
4 2
a
a b
b c
b d
a b
c d
```

describes this tree

```
a --> b --> c
    | --> d
```

And request to our service what are the common ancestors of `(a,b)` and `(c, d)`.

The output of our microservice is a text stream of `m` lines with the responses. With the previous input, should return

```
a
b
```

The folder `dataset` will contain inputs to test your program.