Smart XML Analyzer

Software Engineer - Complex algorithms

Intro

Imagine that you are writing a simple web crawler that locates a user-selected element on a web site with frequently changing information. You regularly face an issue that the crawler fails to find the element after minor page updates. After some analysis you decided to make your analyzer tolerant to minor website changes so that you don't have to update the code every time.

It would be best to view the attached HTML page, imagining that you need to find the "Everything OK" button on every page.

Requirements

Write a program that analyzes HTML and finds a specific element, even after changes, using a set of extracted attributes. We've prepared a sample HTML page ("original" below) and 4 simple difference cases: first, second, third, fourth ("diff-case" below) for you (download as a single pack). Please open the pages in browser to see what we mean by minor website changes. The target element that needs to be found by your program is the green "Everything OK" button. Any user can easily find this button visually, even when the site changes. Original contains a button with attribute **id="make-everything-ok-button"**. This **id** is the only exact criteria, to find the target element in the input file.

The program must consume the original page to collect all the required information about the target element. Then the program should be able to find this element in diff-case HTML document that differs a bit from the original page. Original and diff-case HTML documents should be provided to the program in each run - no persistence is required.

Consider HTML samples, as regular XML files. No image/in-browser app analysis is needed. No CSS/JS analysis is needed (CSS/JS files are provided just for demo).

Remember that Working Software is the main goal so something simple that works is generally better, than a complex unfinished solution.

Must have

1. We need to see your own code. No borrowed code is allowed. However, handy libraries are not forbidden.
2. The application must be smart enough to find the target element, at least for the provided cases. However, a good algorithm should be agnostic and flexible to handle cases beyond the provided samples. At the same time we don't expect absolute reliability of the search algorithm; it must build some similarity level and may fail in some specific cases.
3. Tool execution should look like:
    - <platform> <program_path> <input_origin_file_path> <input_other_sample_file_path>
      Where:
        - <platform> - the chosen language/platform;
        - <program_path> - path to the executable app;
        - <input_origin_file_path> - origin sample path to find the element with attribute id="make-everything-ok-button" and collect all the required information;
        - <input_other_sample_file_path> - path to diff-case HTML file to search a similar element;
    - For example:
        - >java -cp <your_bundled_app>.jar <input_origin_file_path> <input_other_sample_file_path>
        - >python <your_bundled_script>.py <input_origin_file_path> <input_other_sample_file_path>
        - >node <your_bundled_script>.js <input_origin_file_path> <input_other_sample_file_path>
4. Output should be a XML path to the element within the diff-case HTML file. It can be XPath or an absolute path in a form that you like (for example: html > body > div > div[1] > div > a). Output can be provided into a file or the standard output.
5. Target completion time is 2 hours. We would rather see what you were able to do in 2 hours than a full-blown algorithm you've spent days implementing. Note that in addition to quality, time used is also factored into scoring the task.

Nice to have

1. Provide the target element id for collecting the initial information through application parameters, so we can search any element with id (different from the provided in the original samples). We have a