

Code Challenge

Backend Software Engineer

Consider the following XML sample intended to model a directed graph:

```
<graph>
  <id>g0</id>
  <name>The Graph Name</name>
  <nodes>
    <node>
      <id>a</id>
      <name>A name</name>
    </node>
    ...
    <node>
      <id>e</id>
      <name>E name</name>
    </node>
  </nodes>
  <edges>
    <node>
      <id>e1</id>
      <from>a</from>
      <to>e</to>
      <cost>42</cost>
    </node>
    ...
    <node>
      <id>e5</id>
      <from>a</from>
      <to>a</to>
      <cost>0.42</cost>
    </node>
  </edges>
</graph>
```

Write a program in your favorite language to:

1. Download a text file containing an XML file like the one described.
2. Parse the file to make sure it is syntactically and semantically correct subject to the following restrictions:
 - There must be an `<id>` and `<name>` for the `<graph>`.
 - Assume the `<nodes>` group will always come before the `<edges>` group.
 - There must be at least one `<node>` in the `<nodes>` group.

- All nodes must have different `<id>` tags.
- For every `<edge>`, there must be a single `<from>` tag and a single `<to>` tag, corresponding to nodes that must have been defined before.
- The `<cost>` tag is optional and can provide an arbitrary non-negative floating point number. If it's not present, then the cost for the `<edge>` defaults to zero.

You'd have chosen an XML parsing library for your favorite programming language to tackle this task. Please explain why you chose the particular library.

3. Propose a normalized SQL schema to model this graphs in PostgreSQL using standard SQL data types only. Please explain briefly each attribute and relationship you propose. You can use SQL documentation facilities and comments in order to do that.
4. Write an SQL query that finds cycles in a given graph, according to the data model you proposed on item (3). You can use standard SQL99 or PL/pgSQL functions.
5. Suppose there's a frontend application where end-users can create queries having this JSON form:

```
{
  "queries": [
    {
      "paths": {
        "start": "a",
        "end": "e"
      },
    },
    {
      "cheapest": {
        "start": "a",
        "end": "e"
      }
    },
    {
      "cheapest": {
        "start": "a",
        "end": "h"
      }
    },
    ...
  ]
}
```

Write a program that receives this structure via standard input, queries the database, and answers via standard output.

The possible queries are to be interpreted as follows:

- There can be many interleaved paths and cheapest queries in the queries JSON array.
- A single JSON document should be emitted, holding all the answers.
- Each **paths** query must return a list of possible paths between the **start** and **end** nodes referenced by **id**, if there are paths between them; an empty list if there aren't any paths from **start** to **end**. If the graph has cycles, they are to be ignored.
- Each **cheapest** query must return the cheapest path between the **start** and **end** nodes referenced by **id**, if there's at least one path; a false value if there aren't any paths from **start** to **end**. If the graphs has cycles, they are to be ignored.

A sample answer could be:

```
{
  "answers": [
    {
      "paths": {
        "from": "a",
        "to": "e",
        "paths": [ [ "a", "b", "e" ], [ "a", "e" ] ]
      },
    },
    {
      "cheapest": {
        "from": "a",
        "to": "e",
        "path": [ "a", "e" ]
      }
    },
    {
      "cheapest": {
        "from": "a",
        "to": "h",
        "path": false
      }
    },
    ...
  ]
}
```

assuming there are paths a->b->e and a->e and the cheapest path is a->e, while there is no path from a to h.

Requirements:

- Use your favorite programming language.
- Choose a JSON parsing/serialization library from your favorite programming language to tackle this task.
- Please explain why you chose this particular library.
- Please explain how you solved the “find all paths” and the “cheapest path” problems.

Submission:

- You will have seven days to deliver your code.
- Please provide a link to the repository containing your code and instructions on how to run the application locally.
- Include a README file with additional setup or usage instructions if needed.

Evaluation Criteria:

- Code organization and structure.
- Correct implementation of the required features.
- Adherence to best practices and coding standards.
- Code readability and maintainability.