CSE 331: Software Design & Implementation

Section Squares

1. How Do I Love Tree, Let Me Count the Ways

Recall our definition of a binary search tree from HW Weave:

$$\begin{aligned} \textbf{type}\ \mathsf{BST} := &\quad \mathsf{empty} \\ \mid &\quad \mathsf{node}(x : \mathbb{Z},\ S : \mathsf{BST},\ T : \mathsf{BST}) \end{aligned}$$

Suppose that we wanted to have a way to refer to a specific node in a BST. One way to do so would be to give directions from the root to that node. If we define these types:

$$\begin{aligned} \textbf{type}\ \mathsf{Dir} &:= \mathsf{S} \mid \mathsf{T} \\ \textbf{type}\ \mathsf{Path} &:= \mathsf{List}\langle \mathsf{Dir} \rangle \end{aligned}$$

then a Path tells you how to get to a particular node where each step along the path (item in the list) would be a direction pointing you to keep going down the left (S) or right (T) branch of the tree.

For example, $\mathsf{cons}(\mathsf{S}, \mathsf{cons}(\mathsf{T}, \mathsf{nil}))$ says to select the "S" (left) child of the parent and then the "T" (right) child of that node, giving us a grand-child of the root node.

(a) Define a function "find(p : Path, T : BST)" that returns the node (a BST) at the path from the root of T or undefined if there is no such node.

(b) Define a function "remove(p : Path, T : BST)" that returns T except with the node at the given path replaced by empty.

In this section, we will look at an application that includes both a client and a server. To get started, check out the starter code using the command

```
git clone
https://gitlab.cs.washington.edu/cse331-24su/cse331-24su-materials/sec-squares.git
```

To install the modules `cd server` and run `npm install --no-audit`, then `cd ../client` and run the same. Then, `cd server` and start it with `npm run start`. Then, `cd client` and do the same. The server running in the `client` directory is serving JavaScript that runs in the browser, while the server running in the `server` directory is running the server-side JavaScript. Point your browser at `http://localhost:8080`. You should see the same "Animal Trivia" application from Section Chatbot, now rewritten to use React for the client portion.

2. All I'm Asking Is For a Little Correct

Currently, when you first open the website, the application seems be attempting to load a new question indefinitely.

Open the Chrome developer tools and refresh the page. You will notice that you are getting an 'Invalid JSON' error.

(a) Take a look at the `doNewQuestionResp` method in `App.tsx`. What are the parameters? Now take a look at the `newQuestion` method in `routes.ts`. What is that data being passed to the client? Does it match what the client is expecting? Fix the server code so that the client no longer receives the 'Invalid JSON' error.

(b) The server is now sending the correct data; however, the website still loads indefinitely when attempting to get a new question. Take a look at how we are updating the state in `doNewQuestionResp`. What is the issue with the way we are updating the state? Fix the issue so that the website no longer loads indefinitely when attempting to get a new question.

Verify that you can now get a new question when you reload the page.

(c) Now, when you submit your answer, nothing changes in the UI.

Open the Chrome developer tools and take a look at the network tab. Refresh the page, type in an answer, and click "Submit". Verify that the client is making a request to the server to check the answer and is receiving a response. What is the response? What is the client doing with the response?

Fix the client code so that the UI updates when the client receives a response from the server.

Verify that the UI updates when you submit an answer.

(d) The UI now updates when you submit an answer. However, it always says the answer is incorrect. Take a look at the parameters we are passing to the server in `doCheckAnswer`. What is the issue with the way we are passing the parameters? Fix the issue so that the server can check the answer correctly.

Verify that the UI now correctly displays whether the answer is correct or incorrect.

3. Click-state

Add a call to `console.log(JSON.stringify(this.state))` at the top of the `render` method in `App.tsx`. Then, run the application through one cycle from the initial page to a new question. You should see the state printed out multiple times in the browser console.

Draw a diagram showing the sequence of states that were printed out and what events (e.g., a button click or a fetch response) caused each transition between states. No need to write down states that are `undefined`, unless you find it useful to keep track of.

Each character typed into the answer box will generate an update to the state. You don't need to write out every one of those states. Just write the last one.

4. Take Out the Papers and the Flash

You will notice that, when we click on the button for a new question, the UI flashes. As you saw in the previous problem, this occurs because we are redrawing the UI twice: once immediately when the button is clicked and a second time when the fetch completes.

Change the application to skip updating the state after the button click.

Verify that this eliminates the flashing.