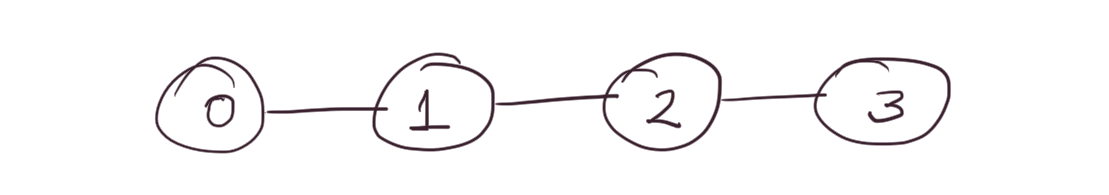
WKWebView History Documentation

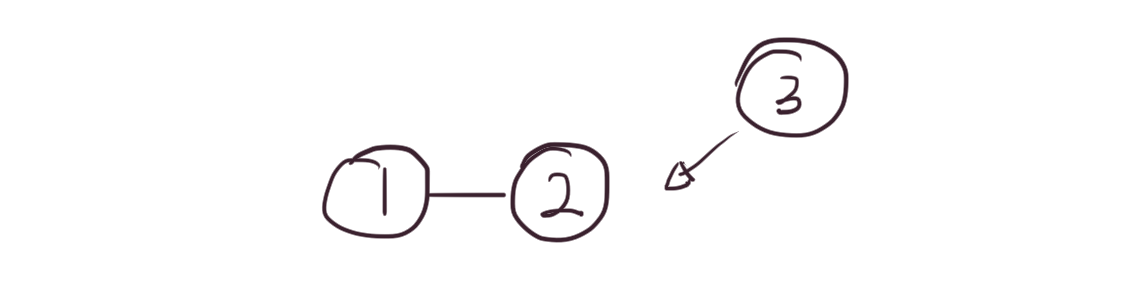
# Introduction

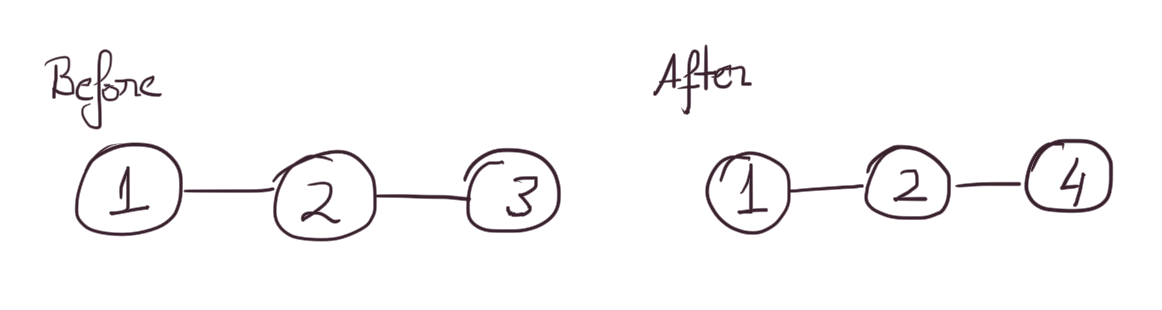
We will call the WKWebView, simply, webview. Each webview holds a list of its back and forward items. These are essentially history items. This is a guide of how I take those history items and put them in a General History for all webviews.

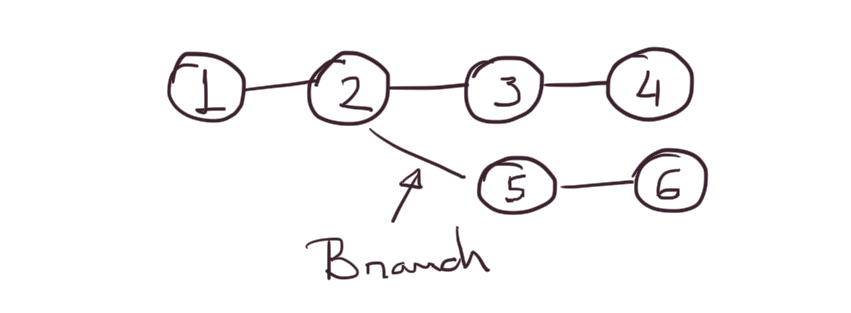
As you might have realized, there are two concepts to distinguish between. The webview history, represented by its list of back and forward items. And the General History that contains the history entries from all webviews.

When thinking about the webview history, it helps to picture a chain of nodes, like the one below:

This chain changes with every new visit. It might add new nodes, or remove some. The challenge is to determine what should be added to the General History.   
  
There are 3 cases that need to be handled.

The first one is called **ADD**. Here is an example:

The second one is called **REPLACE**. Sometimes a node is updated with a new url. That happens for example in the case of a redirect. The redirect url is added and then it is replaced with the url of the actual page. It looks something like this:

The third one is called **BRANCH**. This requires a new term. Each webview keeps an index of which item it is currently at, in order to determine what to load next, when you press the back or forward button. If this index is somewhere inside the chain (not the last node), the chain has more than one node, and the new page is not at index + 1, then we have a branch. You can visualize it like this:   
Now that we have these concepts, we can head into implementation details. (There is something that you might think is missing from the introduction. Namely, how do items from the webview’s list end up in the General History. An explanation is coming. Don’t worry.)

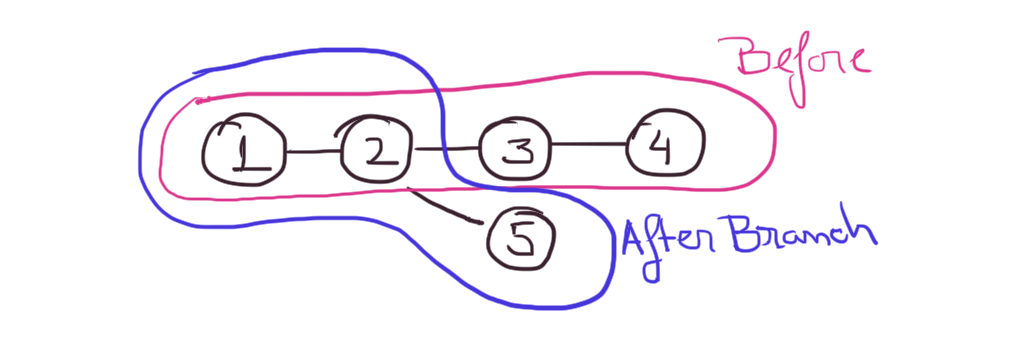
# Implementation Details

The implementation file is called `WebViewHistory`, in case you want to take a look at the code. There you will find a function called `update` that gets called every time the url of the webview changes.   
  
We face a problem, right from the start. The url changes, but the change is not immediately reflected in the webview’s list. But this is easy to solve. We can use a queue to wait for the webview to update. Inside the queue are blocks containing a function that checks if the webview has updated. If it has not, it adds another block. Otherwise it executes code and cancels all other blocks.

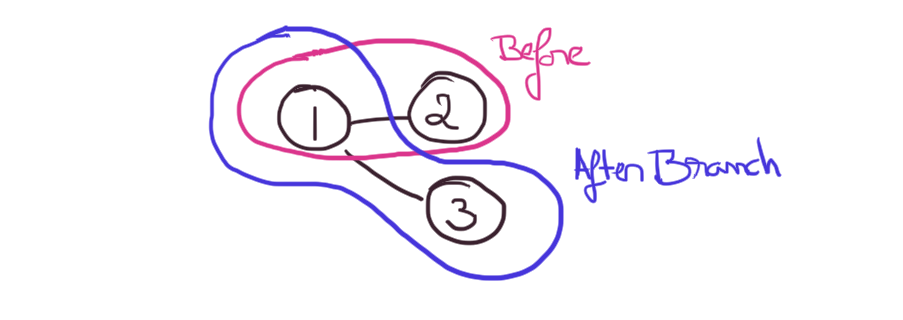
When the webview is updated I can start to look at the change and determine whether I have to deal with an **ADD**, **REPLACE**, or **BRANCH**. To do this I keep a copy of the **webview’s list** to be able to compare changes. We will call this copy, the **internal list**.   
  
There are two straightforward cases. It is easy to determine what happened if the count of the **webview list** is different from the count of the **internal list**.

In case webview\_list.count > internal\_list.count, then something was added to the **webview list**. This is obviously an **ADD**.

In case webview\_list.count < internal\_list.count, then something was removed from the webview list. This happens when a **BRANCH** occurs. I will illustrate this with a picture:



There is one case that is not that straightforward:

The case when webview\_list.count == internal\_list.count. In this case we might have either a **REPLACE**, or a **BRANCH**. The **REPLACE** might be easy to see (there is picture above, about it), but the **BRANCH** might be a little trickier. I will illustrate it with a picture:

As the picture shows, a **BRANCH** can happen, and leave the **webview list** having the exact same size.

How could we differentiate between the two. Well, with a **BRANCH**, I go from node 2 to node 1, and then from 1 to 3 (see picture above). In the case of a **REPLACE**, there is no movement. So I need to check if there was movement. If there was movement, we have a **BRANCH**, otherwise we have a **REPLACE**.

It turns out there is a simple way to determine if there was movement. There is a thing called *forwardCount*. It basically tells me how many nodes are after the current index. Looking at the picture above, at index 1, I have a forward count of 1 (there is only one node after 1, node 2). But when I execute code I am already at 3, and the *forwardCount* is 0 by then. To solve this, we can keep a *\_last\_forwardCount*, that tells us the *forwardCount* one step before the current one. Because, my *forwardCount* is 1 at node 1, the *\_last\_forwardCount* will be 1 at node 3. So a simple check for movement could be \_last\_forwardCount > 0.

This concludes the part where we detect what happened. Now let’s get into what we should do in each of the cases (**ADD**, **REPLACE**, **BRANCH**) with the new url.

# From WebView list to General History

Before I go into the details, I need to mention that there are two things the General History can do. It can add a new entry or update an existing one. So now we need to map **ADD**, **REPLACE** and **BRANCH** to these two.

**ADD** and **BRANCH** turn out to be simple. In both cases I just need to add a new entry in the General History.

**REPLACE** is a little trickier. I need to update an entry in the General History. How do I identify which entry to update. Using the url? Well, what if the same url came from another webview? It turns out that the url is not an unique identifier for an entry in the General History. A unique identifier can be timestamp the entry was added at. Due to the high precision of the timestamps, it is impossible (as far as I can see) that two entries are added at the same time.

Now that we have a unique identifier we can determine which entry in the General History to update. A few more words about this though.

The timestamp is kept in the **internal list** and the General History. It is created in both when an item is added. In case of a **REPLACE**, I use the timestamp of the item in the **internal list** (timestamp of the item I replace), to identify which entry to update in the General History.

# Conclusion

So there are 3 cases: **ADD**, **REPLACE** and **BRANCH**. They map to the operations (add, update) of the General History in the following manner:

ADD and BRANCH —> add

REPLACE —> update