Browser Branched Navigation Using Tree-Like Session History

Raden Rifqi Rahman – 13520166
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13520166@std.stei.itb.ac.id

***Abstract*—Browsers save users' browsing history either globally in all-time browsing history or locally in session history. The session history is a browsing history that is local to a browsing context or a tab. It consists of a sequence of documents. Users sometimes accidentally click the wrong link or Uniform Resource Locator (URL). With the structure of sequence of documents, this accidental click may result in browser navigating the user on such URL and session history deleting some of its entries unintentionally. To address this issue, we can implement a branched navigation model. Branched navigation model is described as a navigation model that allows users to navigate backwards in one way, but multiple ways forwards. It is implemented by taking advantage of tree-like session history in contrast to linear session history with linear navigation model. Branched navigation model allows us to branch a session history entry without having to delete any of the existing entries or create a new browsing context. Consequently, branched navigation model compensates the carelessness of users that may click a URL unintentionally.**

***Keywords*—Branched navigation, session history, browser.**

I. Introduction

A browser works by sending and receiving data or information from the web. Sending and receiving such data requires us to do certain actions, including clicking a link and entering a web address or a Uniform Resource Locator (URL) to the browser address bar. Upon receiving data, browsers may display a new web page to our screen. Browsers may also alter or update the currently displayed page. Such pages are usually present in terms of Hypertext Markup Language (HTML) document and Extensible Markup Language (XML) document. Oftentimes, we want to move from a certain document or URL to another document or URL while browsing. We rarely want to stay at the same page for an extensive amount of time.

Users move to a new URL frequently. Moving to a URL is said to navigate to that URL. While navigating, a browser keeps track of what the user visits for possible later use. Every page or URL the user visited is stored in a browsing history. That means browser enables users to navigate back to any page they have visited without having to remember its URL. Even though a browsing history is useful, browsers allow user to do a parallel browsing. Huang and White [1] states that "parallel browsing describes a behavior where users visit web pages in multiple concurrent threads." Most browsers allow this by providing tabs.

With the existence of parallel browsing and tabs, users can open and view multiple web pages at a time. Nonetheless, browsers have such behavior that they save users' browsing history globally, meaning the browsing history of multiple tabs are saved in a single browsing history in the browser. It may be inconvenient for some users. When they browse with multiple tabs, the browsing history for each tab may mix with one another. To tackle this problem, browsers introduce another kind of history, namely the session history.

A session history consists of a sequence of *Documents* in a browsing context [2]. A *Document* is an object which represents an XML or HTML document. On the other hand, a browsing context is a place where such Document objects are displayed as a web page to the user [2]. It is roughly analogous to a tab or window in a browser. While having separate session histories can be helpful, session history comes with its own limitations. Since the structure of the session history itself is a sequence of Documents, it is unable to save every page the user visits. The session history will have to pop off or delete some of its contents eventually. For example, suppose a user browses three pages in sequence, say, page A, page B, and page C. Currently, the user is at page C. Now suppose they navigate back to page B and will return to page C afterwards. If by any chance, here, the user clicks a link to another page accidentally, say page D, the browser will navigate the user to page D, replacing page C from the session history by page D. In consequence, the session history loses page C forever and now contains page D instead. This can be troublesome for the user.

Most users know how to retrieve their lost page via the browsing history, i.e., the all-time history. Still, we know that the browsing history mixes up all session histories. Moreover, a study [1] claims that "57.4% of tab sessions involve parallel browsing," meaning a lot of users use more than a single tab while browsing. Although it is possible to look up for the page in the browsing history manually, it is likely to take quite a lot of time.

To address this issue, we need session history to preserve important pages in one way or another. Without having to redesign a browser entirely, we can implement a branched navigation model. Branched navigation means user can navigate backwards in a single way, but multiple ways forwards. Instead of using a sequence of Document objects, branched navigation encourages session history entries to wrap those Document objects inside a tree-like structure.

II. TERMINOLOGY

A. Node

Nodes are connected data that is spread throughout memory [3]. A node can contain any kind of data depending on how we define it to be. Therefore, aside from primitive data types, such as integers and characters, a node can hold a more complex data type or structure. Regardless of what data type it holds, a node always contains *at least* one *link* to another node.

B. Sequence

We refer to *sequence* as a collection of ordered elements that share the same type. Reference [4] defines sequence as an abstract data type that represents a collection of values, each of which may appear more than once. In other terms, this data type is commonly called *linked list*.

The term *linked list* is highly related to the term *link* in a node. A sequence or a linked list consists of a list of nodes. Because a node is spread throughout memory, each node in a sequence has a *link* that points to the *next* node. Hence, the order of elements in a sequence is determined by those links, with the last element not having any links to any nodes. Fig. 1 shows a diagram which represents a sequence of the first five natural numbers.

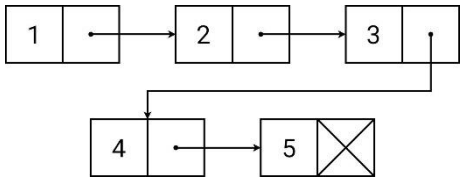

Fig. 1. A diagram representing a sequence of the first five natural numbers.

Each element within a sequence has its own unique *index*. The index of an element is a number that identifies the order of elements within a sequence. We refer to the first element of a sequence as having an index of 0. Therefore, in Fig. 1 we refer the node containing 1 to be at index 0, the node containing 2 to be at index 1, and so on.

C. Tree

Tree is another node-based data structure besides sequence. The main difference between a sequence and a tree is the link of its nodes. Each node in a sequence can only contain one link, whereas tree nodes may have links to multiple nodes [5]. Hence, each node in a tree can point to multiple nodes, namely the child nodes. The *children* of a node *A* are all nodes that *A* points to. Consequently, *A* is the *parent* of all its children. For clarity, a tree consisting of eight nodes can be visualized as shown in Fig. 2.

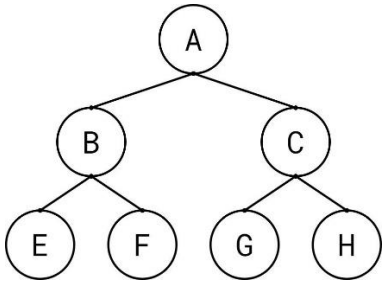

Fig. 2. A tree consisting of eight nodes.

The uppermost node, which is the node that no other nodes point to, is called the *root* [5]. In Fig. 2, we call node *A* as the *root* of the whole tree. We also call nodes *B* and *C* as the children of node *A*. Likewise, *E* is a child of *B* and *G* is a child of *C*, but *F* is neither a child of *C* nor *A*. To avoid confusion, the visualization of a tree is highly analogous to a family tree. As in a family tree, *E* is a *sibling* of *F*, and vice versa.

Some other terminologies we are interested to are *descendants* and *ancestors*, *level* and *height*, and *subtree*. As well as a family tree, a node may have descendants and ancestors. According to [5], a node's descendants are "*all* the nodes that stem from a node", whereas a node's ancestors are "all the nodes that it stems from." In our visualization (Fig. 2), *A* is the ancestor of every other node because all nodes except *A* stems from *A*. Thus, nodes *B*-*G* are called the descendants of *A*.

Each node of a tree rests on a particular *level*. All nodes that lie in the same row are said to have the same level. However, multiple references [5], [6] define the level for each row differently. Reference [5] states that the root node, which in our case is *A*, lies in the first level of the tree; whereas [6] defines the root has a level of zero. To unify this distinction, we refer to the root node as having the first level. Aside from levels, trees also have *height*. The height of the tree is the maximum levels that the tree has [6].

The children of the root node of a tree are also a tree. These trees are called *subtree* from the current tree. Therefore, the tree in Fig. 2 has two subtrees with *B* and *C* as their root nodes, as shown in Fig. 3.

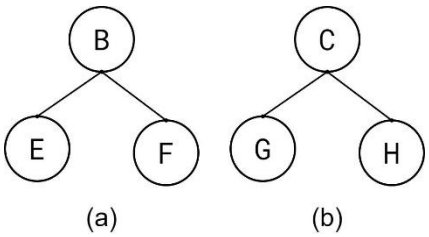

Fig. 3. Subtrees of tree in Fig. 2. (a) The first subtree. (b) The second subtree.

D. Browsing Context

According to [7], a *browsing context* is "the environment in which a browser displays a *Document*." In modern browsers, a browsing context is a tab, a window, or parts of the page, such as *frame* and *iframe*. We mostly refer to a browsing context as a single tab. However, it is not limited to a tab in general. The HTML Living Standard [2] states that "a browsing context has a session history, which lists the *Document* objects that the browsing context has presented, is presenting, or will present." A *Document* is an interface which "represents any web page loaded in the browser and serves as an entry point into the web page's content" [8].

E. Session History

We refer to *session history* as a collection of *session history entries*. A *session history entry* contains a URL and a Document object or null [2]. With linear navigation model, the session history entries are organized in a sequence structure. Since a sequence can be visualized with a diagram in Fig. 1, we can also visualize a session history using such diagram. We may simplify

the visualization of the structure of each entry by giving each a label. For instance, a session history consisting of three entries with document *A*, *B*, and *C* is visualized in Fig. 4.

Fig. 4. Linear session history consisting of three entries.

Note that a sequence's elements are ordered by default. Nonetheless, the session history entries are also ordered. This means the order of entry in the session history is determined by the order of the element in the sequence. In consequence, the session history depicted in Fig. 4 shows that a user browses the document with label *A*, then navigates to *B* and *C* in order.

III. METHODS

A. Tree-Like Session History

Most browsers implement linear session history, i.e., each session history entry is arranged in a sequence. As a result, linear session history disallows us to branch an entry. Suppose we surfed four documents in the web in sequence. By the time we load the last document, our session history may look like this.

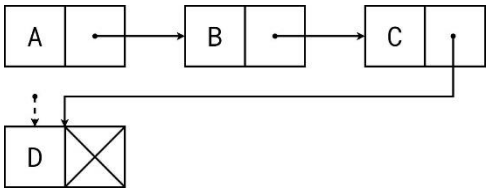

Fig. 5. Session history after surfing four documents.

The dashed arrow in the figure indicates the current history entry or the document we are currently in. Hereafter, assume we are forgetting something and need to look up document *B*. By using linear navigation, we can navigate backwards directly from *D* to *B*. At this moment, the session history still contains documents *C* and *D* for us to navigate forwards as shown in Fig. 6(a). However, documents *C* and *D* are only preserved for as long as we navigate forwards (or further backwards). This implies that by the second we try to *branch* the history at any point, the browser will remove all session history entries from that point *forwards*. As an example, after navigating backwards to *B*; if we go to another document *N*, the session history will result in Fig. 6(b).

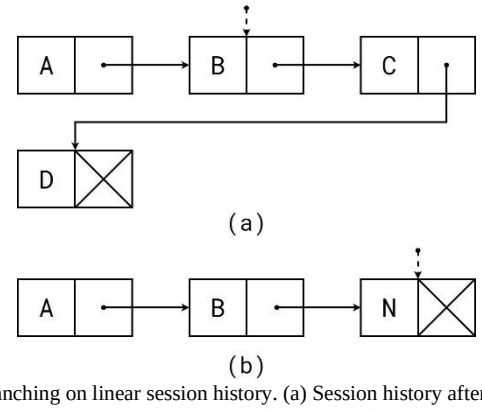

Fig. 6. Branching on linear session history. (a) Session history after navigating backwards to B. (b) Result of navigating from B to N.

Instead of linear session history, we can implement a tree-like session history. In contrast with linear history illustrated in Fig. 5, our session history tree will look like the following.

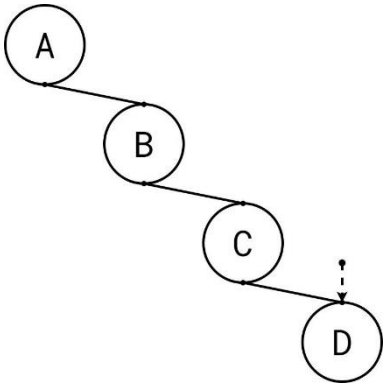

Fig. 7. Session history tree equivalent to Fig. 5.

By using the tree data structure, we enable each entry in the session history to have links to multiple entries. Hence, the session history will be capable to append a new entry at any point which is equivalent to branching the history.

B. Branching

Branching a session history entry means creating a new branch on such entry in our session history. Although it is not possible to branch an entry in a linear session history, we still can branch our session history. We can bypass the limitation to branch linear session history with parallel browsing. Note that each of Fig. 5, 6(a), and 6(b) is a linear session history of a *single* browsing context. Therefore, we can create a new browsing context to branch the history from document *B* in Fig. 6(a). In other words, we may open a new tab to display the document *N*. Hence, we will not modify the session history in Fig. 6(a). In contrast with Fig. 6(b), with a new browsing context, our session history will end up looking like Fig. 8 instead.

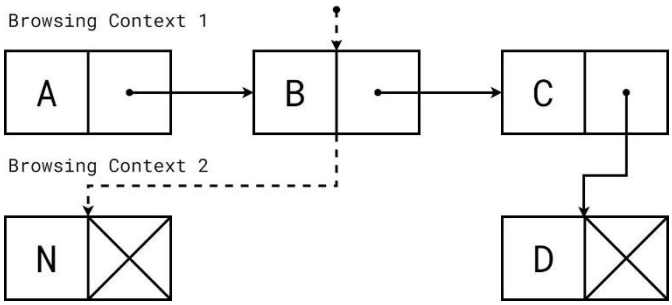

Fig. 8. Branching linear session history using multiple browsing context.

The dashed arrow in the figure indicates that we branch the first browsing context at document *B* to a new browsing context with document *N*. In other words, we open document *N* in a new tab whilst staying in document *B* in the old tab. By expanding our browsing contexts, we effectively solve our branching problem. However, such branching model comes with its own limitations. One of the issues is the disconnected history on the second browsing context. We observe that document *N* is the only entry that session history of browsing context 2 contains, whereas it should come after document *A* since it is opened after

A considering the sequence of document in browsing context 1's session history. Although such behavior may be useful in some cases, there is another issue. If we need to open a new tab or create a new browsing context every time we branch a history, the workload for our browser will grow up very quickly, resulting in more computational resource we must provide for the browser.

Instead of replacing the link of a session history entry node or creating a new browsing context to branch as if in linear session history, we can simply add a new link to another new node from the entry we are currently in. With a similar scenario as before, we can navigate backwards directly from *D* (Fig. 7) to *B*. If we visualize these steps, we will end up with a session history that looks like the one in Fig. 9.

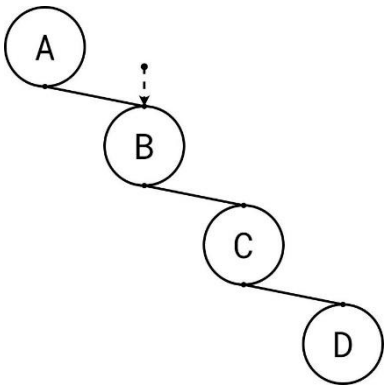

Fig. 9. Session history tree after navigating to B.

So far, our session history tree behaves identically as linear session history. As we observed, after navigating backwards to document *B*, our session history in Fig. 8 looks identical with the one in Fig. 6(a). Nonetheless, it will show a difference when we try to branch the history. Continuing our scenario, we like to open document *N* at this point. Since we are not in the lowermost entry of our history, opening a new document means we are about to branch the session history. To do a history branching in our session history tree, we have to do the following steps.

First, we create a node of the new entry containing the document *N*. Afterwards, we link the current entry, which in this case is *B*, to the new entry as its child node. At the moment, our session history tree will look like the following figure.

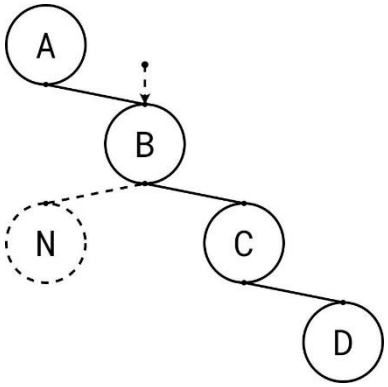

Fig. 10. Preparing to branch an entry in session history tree.

With a new entry created and linked to our tree, we are ready to navigate to the new entry, namely the document *N*. In comparison with previous results on linear session history, the session history tree maintains document *C* and *D* as its entries as shown in Fig. 11.

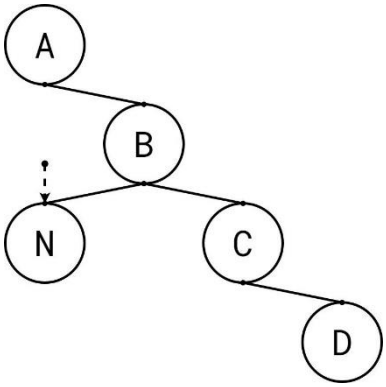

Fig. 11. Branching session history tree on entry B.

The figure shows us the result of branching a history entry within a session history tree. As opposed to linear session history, the session history tree still fits inside a *single* browsing context without popping off any of its entries.

C. Branched Navigation

Aside from its similarities with linear session history, tree-like session history has its own uniqueness and advantage. The session history tree enables us to do a *branched navigation*. Branched navigation model is described as a navigation model that allows users to navigate backwards in one way, but multiple ways forwards. It means whenever a user stays at a session history entry which has multiple branches or children, they may navigate to any of the branches or children, or even all their descendants down further.

On the contrary, linear session history implements linear navigation model. Linear navigation model is described as a navigation model in which users always navigate in one way either forwards or backwards. For example, suppose we have a far more complex session history for both linear and tree-like session history.

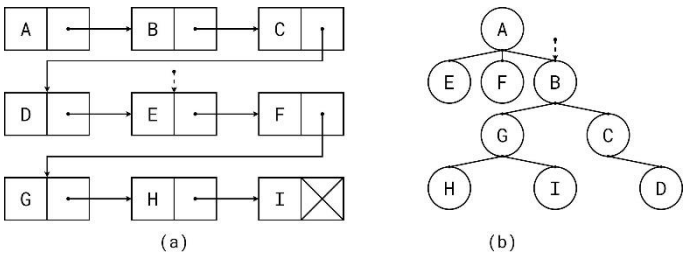

Fig. 12. Complex session histories. (a) Complex linear session history. (b) Complex session history tree.

The figure above shows us two rather complex session histories, both of which consisting of nine entries. By applying linear navigation on session history in Fig. 12(a), we know how browsers will handle the two directions of navigation. If we navigate backwards, it means we are trying to reach either document *A*, *B*, *C*, or *D*. Likewise, we are trying to reach either document *F*, *G*, *H*, or *I* whenever we want to navigate forwards. In accordance with this behavior, we can define navigating

backwards *once* as going back from *E* to *D*. Similarly, we can define navigating forwards *once* as going forward from *E* to *F*. By expanding this definition of navigation, we now have what is called the linear navigation. As another example, it is said to navigate *twice* backwards if we are going from entry *E* to *C* and it is also said to navigate *three steps* forwards when we are moving to *H*.

On the other hand, we may apply linear navigation to the tree-like session history in Fig. 12(b). For a session history tree, navigating backwards implies moving to any of the ancestor nodes of the entry node we currently stay at. In Fig. 12(b), navigating backwards is only possible to document *A* since it is the only ancestor of the entry we are currently in, which is *B*. However, it is not so clear how does forwards navigation work within session history tree. Since *B* has two children namely *G* and *C*, we cannot define exactly how is navigating forwards done in session history tree. If we define navigating forwards *once* from *B* as going to *G*, then navigating from *B* to *C* is not defined at all since we are not able to reach *C* from *G*. Therefore, we need to apply our branched navigation model on this session history tree.

The branched navigation model introduces a new method of navigating forwards. As stated, we have multiple ways to navigate forwards. Linearly, there are three ways of forwards navigation in Fig. 12(b) which are *G-H*, *G-I*, and *C-D*. Still, we cannot pick any of them because we will never reach the others if we do. With that being said, the branched navigation model *groups* all of descendants of the current entry node as a collection of entries. Consequently, forwards navigation in Fig. 12(b) refers to going forward to any of the entries *G*, *C*, *H*, *I*, and *D* regardless of the order. Thus, the *n-steps* forwards navigation is not defined in branched navigation model.

Browsers should handle forwards navigation in branch navigation model differently. If a user tries to navigate one step forwards, there are two possibilities that browsers can handle differently. One possible scenario is if there is only one child of the current session history entry. In that case, browsers can directly navigate the user to the child entry as if it was a linear session history. Another scenario is when there are multiple children of the current entry. In this case, browsers have to ask the user for which child they like to navigate to. For instance, suppose a user try to navigate one step forwards in Fig. 12(b). Since *B* has two children, *G* and *C*, we need to ask the user to ensure whether they want to navigate to *G* or *C*.

We also notice that the entries *E* and *F* are not accessible from *B*, meaning we cannot navigate directly from *B* to either *E* or *F*. This is an expected behavior of branched navigation model that we don't have to worry about. Recall that in Fig. 6(a) and 6(b) branching a linear session history results in deleting all entries that is on the same index or higher. It means that all entries with the same indices are not always related with each other. By analogy with tree-like session history, each entry that lies on the same level is not always directly related with one another. Despite sharing the same parent, sibling entries don't necessarily share the same information or context. Hence, it is reasonable if we are restricted to directly navigate to *E* and *F* in Fig. 12(b), or any of their children entry if any. It is also useful to separate different environment or section of our session history.

IV. Impact and Discussion

A. Advantages and Disadvantages

Branched navigation model has its own advantages against linear navigation model. We saw that branched navigation model with session history tree allows us to branch a history entry without having to delete any of the existing entries or create a new browsing context. Branched navigation model also grants us the ability separate a different environment or workspace in our browsing session within a single browsing context. Among those advantages, we recognize that there are some disadvantages on branched navigation model as well.

Our example in Fig. 12 informs us a minor disadvantage of branched navigation model. In branched navigation model with session history tree, we cannot define the flow of forwards navigation by default. In case of having multiple branches from the current session history entry, we always need to confirm which branch the user wants to navigate to. As an impact, if we apply the branched navigation model to modern browsers, they have to change the flow or algorithm of forwards navigation slightly. By default, browsers forward button works by navigating the user one step forwards if there are any entry forwards or is disabled when no more entry exists after the current entry. However, since we cannot define the n-steps forwards navigation, we will have to change how this button works.

The forward button in branch navigation model navigates the user differently. First, we group all descendants of the current entry. Then, we ask the user to choose one of the descendants to navigates to. Nonetheless, there is a certain priority to determine how are those descendants served to the user to choose. Note that each node in the tree rests on a certain level. By using the information of each entry's level, we can pick the most relevant entries first for the user to choose. In Fig. 12(b), we know that the entries *G* and *C* are the children of the current entry, *B*. Hence, it is highly likely that they are the most relevant entry to the current entry. Thus, we assume the user would like to navigate to any of them. However, it is not always the case. In case the user wants to navigate to *H*, *I*, or *D* directly, we allow the user to expand either *G* or *C*, or both, so that the user finds the exact entry they want to navigate to. For example, Fig. 13 shows how the user can navigate directly to entry *I*.

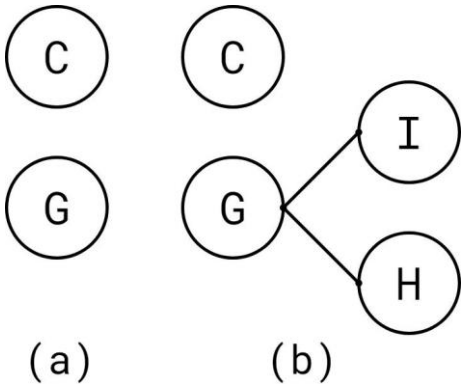

Fig. 13. Branched navigation to entry I. (a) The first entries for the user to choose. (b) User expands G to reach I.

Apart from the minor advantage of forwards navigation, branched navigation model also has a major disadvantage. We notice that in Fig. 10, branching an entry in session history tree does not involve deleting or popping off any of the existing entries. In consequence, visiting 100 different documents in any order or branch results in a session history tree with 100 entry nodes. If we let it be, the session history will eventually grow larger quickly and takes a lot of space or memory. This is a new issue to take care of.

B. Saving Space

Space constraint is a new issue and major disadvantage that the branched model navigation introduce. Because the session history tree never deletes its entries, a single browsing context implementing branched navigation model can take a lot of space equivalent to parallel browsing. Therefore, it is considerably good for session history tree to delete its entries sometimes. However, we need to be careful on which entry to delete.

Choosing a session history entry to delete may not look straightforward. Since branched navigation model makes use of tree data structure, deleting a session history entry inside a session history tree implies deleting its node and all of its descendants. Hence, we can conclude that we cannot delete any of all ancestors of the current entry because if we do, we also delete the current entry.

In branched navigation model, every child of the current entry and its descendants need to be preserved since it is the sole purpose of the branch. At this point, we know that we have to keep all ancestors and descendants of the current entry in session history tree. That being so, the only entries we are allowed to delete are the siblings of the current entry and their descendants. Even though we know which entries to delete, we still have to determine when to delete them.

Deleting an entry from session history tree should not be done every time a user navigates because it will defeat the purpose of branched navigation model to preserve certain branches. Instead, we can delete an entry when we are sure that the user is no longer interested in the entry we are about to delete. To be assured that the user is no longer interested to the entry, we may suggest a measure to determine user's interest to each entry within a session history tree. This measure is called the *session history entry persistency*.

Thus far, we are restricted to delete all ancestors and descendants of the current session history entry. This indicates that the ancestors and descendants of the current entry are always *persistent*. On the other hand, the remaining entries do not necessarily need to be persisted. By adding additional piece of data to each entry within a session history tree, we will be able to determine when to delete an entry from the session history tree. This piece of data is called the *persistence* of an entry.

The *persistence* of an entry is a number that determines the lifetime of the entry. Every time the user navigates, all entries that need to be persistent are assigned the value of *maximum persistence* which is a configurable predetermined value. On the contrary, instead of instantly deleting the remaining entries, each of the remaining entries persistence will *decay* over time. The persistence of them will be reduced by 1 every time the user navigates. Upon reaching a persistence value of 0, the user is considerably no longer interested in such entries, and we are assured to delete them from its session history tree. For example, suppose the session history tree in Fig. 12(b) has a maximum persistence of 3 and let all entries have a persistence of 3.

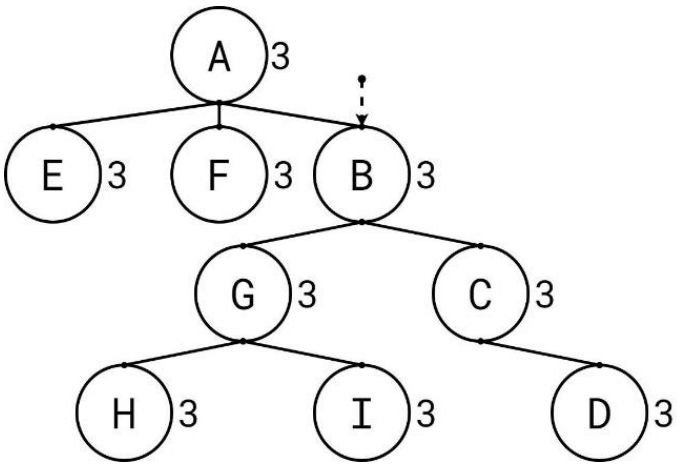

Fig. 14. Session history tree with entry persistence.

The figure above shows us a session history tree equivalent to Fig. 12(b) with the persistence of each entry. In Fig. 14, if we navigate three consecutive times without moving to document *A*, we are considered to no longer be interested in *E* and *F*. As depicted in Fig. 15(a), 15(b), and 15(c), if we navigate to *C*, *D*, and back to *B* consecutively, the persistence of *E* and *F* will decay to 0 over time.

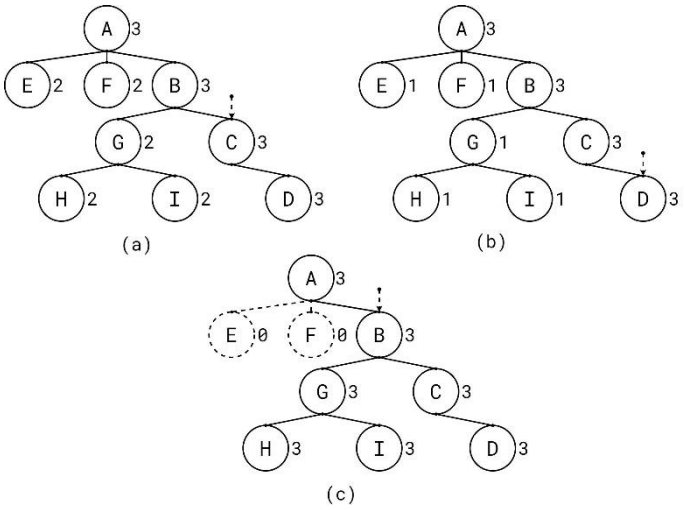

Fig. 15. Decay of entry persistence. (a) First navigation. (b) Second navigation. (c) Third navigation.

The steps depicted in the figure above tells us that after some navigations, determined by the maximum persistency, one or more entry will eventually reach 0 persistency. Hence, the entry persistency works perfectly to determine whether a user is still interested in a document or not. However, it is not mandatory to set a maximum persistency to 3. In general case, the more branches the session history has, the higher the value of maximum persistency should be to correctly measure user's interest on a document or entry. Since the user is likely to only sticks to a few branches instead of all available branches, some

branches will have to be deleted eventually to save some space regardless of how the user navigates around. By applying this measure, we solve branched navigation model space constraint issue.

V. Conclusion

Branched navigation model is described as a navigation model that allows users to navigate backwards in one way, but multiple ways forwards. It is implemented by taking advantage of tree-like session history in contrast to linear session history with linear navigation model. Branched navigation model allows us to branch a session history entry without having to delete any of the existing entries or create a new browsing context.

Branched navigation model has a known minor and major disadvantage. The minor disadvantage of the navigation model is that browser is not always certain of where to navigate user forwards. The major disadvantage of branched navigation model is the uncertainty of when to delete an entry from the session history tree. However, there is a suggested solution on this issue, namely the implementation of session history entry persistency. By assigning a persistence value for each entry within the session history tree, there is a measure to determine when an entry should be removed from the session history tree.

VI. Acknowledgment

First and foremost, I would like to praise and thank God, the Almighty, who has granted me countless blessing, knowledge, and opportunity to finish this paper. I would also like to express my gratitude to all lecturers of IF2120 Matematika Diskrit who encourage me to write this paper at the first place. Last but not least, I would also like to thank my family who supports me throughout life.

References

[1] J. Huang and R. W. White, "Parallel browsing behavior on the web," in *Proc. 21st ACM Conf. Hypertext and Hypermedia*, 2010, doi: 10.1145/1810617.1810622.

[2] *HTML Living Standard*, Dec. 8, 2021. Accessed on: Dec. 11, 2021. [Online]. Available: https://html.spec.whatwg.org/

[3] J. Wengrow, "Node-Based Data Structures," in *A Common-Sense Guide to Algorithm and Data Structures*, B. MacDonald, Ed., 2nd ed, Raleigh, North Carolina, USA: Pragmatic Bookshelf, 2020, ch. 14, pp. 225–246.

[4] Institut Teknologi Bandung. (2021). ADT List. [Online]. Available: https://cdn-edunex.itb.ac.id/32549-Algorithm--Data-Structure-Parallel-Class/34114-Week-03/17283-Lecture-Notes/1630998282515_W03_A1_ADTList.pdf

[5] J. Wengrow, "Speeding Up All the Things with Binary Search Trees," in *A Common-Sense Guide to Algorithm and Data Structures*, B. MacDonald, Ed., 2nd ed, Raleigh, North Carolina, USA: Pragmatic Bookshelf, 2020, ch. 15, pp. 247–278.

[6] K. H. Rosen, "Trees," in *Discrete Mathematics and Its Applications*, B. MacDonald, Ed., 8th ed, New York, NY, USA: McGraw-Hill Education, 2019, ch. 11, sec. 1, pp. 781–791.

[7] MDN Contributors. "Browsing context - MDN Web Docs Glossary: Definitions of Web-related terms | MDN." MDN Web Docs. Accessed on: Dec. 11, 2021. [Online]. Available: https://developer.mozilla.org/en-US/docs/Glossary/Browsing_context

[8] MDN Contributors. "Document - Web APIs | MDN." MDN Web Docs. Accessed on: Dec. 11, 2021. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/Document

Statement

I certify that this paper is my own work and is not plagiarized, based on my personal study and research and that I have acknowledged all material and sources used in its preparation, whether they be books, articles, reports, lecture notes, and any other kind of document, electronic or personal communication.

Serang, December 12, 2021

Raden Rifqi Rahman – 13520166