TRACING THE WEB: A STACK-BASED BROWSING HISTORY

A PROJECT REPORT

Submitted by

SHYAM KUMAR SHAH[RA211026010371]
PRATHAM SHRIVASTAV [RA211026010366]
ANJALI SATIJA [RA2211026010390]

for the course 21CSC201J Data Structures and Algorithms

Under the Guidance of

Dr. A Maheshwari

Assistant Professor, Department of Computing Technologies

In partial satisfaction of the requirements for the degree of

BACHELOR OF TECHNOLOGY
in
COMPUTER SCIENCE ENGINEERING

SCHOOL OF COMPUTING

COLLEGE OF ENGINEERING AND TECHNOLOGY

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

KATTANKULATHUR - 603203

NOVEMBER 2023

SRM INSTITUTION OF SCIENCE AND TECHNOLOGY
KATTANKULATHUR-603203

BONAFIDE CERTIFICATE

Certified that the 21CSC201J Data Structures and Algorithms course project report titled **"Tracing the web: A Stack-Based Browsing History"** is the bonafide work done by **Shyam Kumar Shah [RA2211026010371], Pratham Shrivastav [RA2211026010366] and Anjali Satija [RA2211003011390]** who carried out under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form part of any other work.

Dr. A Maheshwari
Faculty In-Charge,
Assistant Professor,
Department of Computing Technologies,
SRMIST.

Dr. Annie Uthra
Professor and Head,
Department of Computing Technologies,
SRMIST.

ABSTRACT

In an era defined by the ubiquity of the internet, web browsing has evolved into an essential daily activity. Users navigate a vast and intricate digital landscape, engaging with a multitude of websites and services, often following nonlinear, interconnected paths. Traditional web browser history mechanisms offer linear, one-dimensional perspectives on user activity. However, this approach is often inadequate for retrace complex and dynamic web journeys.

This paper presents a transformative paradigm for managing web browsing history through a stack-based model, which enhances the navigation and understanding of online exploration. The "Tracing the Web" system introduces the concept of web stack frames, where each frame encapsulates a unique browsing context. Users can effectively navigate through these frames, allowing them to revisit and comprehend intricate interconnections between web pages, applications, and services. This novel approach significantly improves the ability to trace and comprehend complex web journeys.

We begin by outlining the architectural framework of the "Tracing the Web" system, highlighting the principles behind the stack-based model, its adaptability to different web browsers, and its seamless integration into the user experience. The system leverages innovative algorithms and user-friendly interfaces to empower individuals to manage their online history more efficiently.

Our experimental results illustrate the tangible benefits of this stack-based browsing history system. Users reported improved recall, comprehension, and efficiency when retracing their web exploration, as compared to traditional linear history mechanisms. The stack-based approach enables users to better contextualize their online experiences, fostering a more comprehensive understanding of the relationships between visited web pages and services.

In conclusion, "Tracing the Web" represents a fundamental shift in the way users interact with web content, offering an advanced and intuitive method to retrace and comprehend complex online journeys. By focusing on stack-based browsing history, this innovative system empowers users to explore the intricate web landscape with greater depth and clarity, opening up new possibilities for research, productivity, and knowledge management in the digital age.

ACKNOWLEDGEMENT

We express our heartfelt thanks to our honorable **Vice Chancellor Dr. C. MUTHAMIZHCHELVAN**, for being the beacon in all our endeavors.

We would like to express my warmth of gratitude to our **Registrar Dr.A Maheshwari** for his encouragement.

We express our profound gratitude to our **Dean (College of Engineering and Technology) Dr. T. V.Gopal,** for bringing out novelty in all executions.

We would like to express my heartfelt thanks to Chairperson, School of Computing **Dr. Revathi Venkataraman,** for imparting confidence to complete my course project

We wish to express my sincere thanks to **Course Audit Professors Dr. Vadivu. G, Professor, Department of Data Science and Business Systems and Dr. Sasikala. E Professor, Department of Data Science and Business Systems** and **Course Coordinators** for their constant encouragement and support.

We are highly thankful to our my Course project Faculty

Dr. Nancy P, Assistant professor, Department of Computing Technologies, for his/her assistance, timely suggestion and guidance throughout the duration of this course project.

We extend my gratitude to our **HoD Ms Pusplata, C-Tech** and my Departmental colleagues for their Support.

Finally, we thank our parents and friends near and dear ones who directly and indirectly contributed to the successful completion of our project. Above all, I thank the almighty for showering his blessings on me to complete my Course project.

TABLE OF CONTENTS

1. INTRODUCTION

In the ever-evolving digital landscape of the World Wide Web, web browsers serve as our gateway to a vast repository of information and experiences. As we navigate the intricate maze of websites, pages, and content, our journey leaves behind a digital breadcrumb trail, commonly referred to as browsing history. While traditional browsing history tools provide a linear list of visited websites, a new approach known as “Tracing the Web” reimagines this concept by implementing a stack-based browsing history. This innovative approach brings a fresh perspective to how we navigate and interact with the web, offering users an intuitive and dynamic way to explore online content while maintaining a contextual awareness of their digital journey. In this article, we will delve into the concept of stack-based browsing history, exploring its implications for user experience, information retrieval, and the ever-expanding realm of web exploration. Join us as we uncover the potential of this cutting-edge approach and its impact on the way we interact with the digital landscape.

2. PROBLEM STATEMENT

In today's digital age, web browsers are essential tools for navigating the vast expanse of the internet. However, managing the history of visited web pages can be a challenging task. Users often need an efficient way to retrace their steps through the web to revisit previously viewed pages. To address this, a stack-based browsing history system is needed.

Problem Requirements:

You are tasked with designing and implementing a stack-based browsing history system that allows users to:

1. Navigate Web Pages: Users should be able to visit web pages and have the system record each page's URL in a history stack.

2. Go Back: Users should be able to go back to the previously visited page by popping the stack. Each time the user navigates to a new page, the current page should be pushed onto the stack.

3. Go Forward: Users should be able to go forward to the next visited page if they have previously gone back.

4. Display History: Users should be able to view their browsing history in chronological order.

5. Handle Bookmarks: Allow users to mark specific pages as bookmarks for easy access.

6. Manage the Stack: Ensure that the stack does not grow indefinitely, and limit the number of pages stored in the browsing history.

Implementation Guidelines:

1. Use a stack data structure to maintain the browsing history.

2. Implement functions or methods to perform the required operations: visit a page, go back, go forward, display history, and manage bookmarks.

3. Ensure that the stack has a maximum size to prevent excessive memory usage.

4. Implement a user interface (GUI or command-line) to interact with the browsing history system.

5. Consider implementing data persistence to save browsing history and bookmarks between sessions.

6. Provide clear user feedback and error handling.

Evaluation:

Your solution will be evaluated based on the following criteria:

1. Correctness and functionality of the browsing history system.

2. Efficiency and resource management, especially regarding memory usage and stack management.

3. User interface design (if applicable) and user experience.

4. Implementation of bookmarks and their accessibility.

5. Handling of edge cases and robust error management.

6. Documentation and code readability.

Note: This problem statement is designed for a software development project that could involve programming in various languages and may require design considerations, data structures, and user interface development.

3. STACKS

A stack is a fundamental data structure in computer science and programming that follows the Last-In, First-Out (LIFO) principle. It is a collection of elements with two primary operations: pushing (adding) elements onto the stack and popping (removing) elements from the top of the stack. Stacks are used in a wide range of applications and algorithms due to their simplicity and efficiency.

Here's a detailed explanation of stacks:

1. Fundamental Concept:

- A stack is a linear data structure that represents a collection of elements with two principal operations: push and pop.

- Elements are added to the top of the stack, and they are removed from the top as well, following the LIFO principle.

2. Operations:

- Push: This operation is used to add an element to the top of the stack.

- Pop: This operation is used to remove the top element from the stack.

- Peek (or Top): Retrieve the top element without removing it.

3. Key Characteristics:

- Stacks are often implemented using arrays or linked lists, where each element is a node in the linked list.

- Stacks have a fixed or dynamic size, depending on the implementation.

- The top of the stack is the only point of access for adding and removing elements.

4. Common Use Cases:

- Function Call Stack: Stacks are used to manage function calls in most programming languages. When a function is called, its context is pushed onto the stack, and when it returns, it’s popped off.

- Expression Evaluation: Stacks can be used to evaluate arithmetic expressions, postfix (Reverse Polish) notation, and infix-to-postfix conversion.

- Undo and Redo Functionality: Stacks can be used to implement undo and redo operations in applications.

- Backtracking Algorithms: Stacks are useful in backtracking algorithms, such as finding paths in a maze.

7. Stack Implementations:

- Stacks can be implemented using arrays or linked lists. The choice of implementation depends on the specific use case and requirements.

- Linked lists are often preferred when the size of the stack is not known in advance, as they can grow or shrink dynamically.

In summary, a stack is a simple yet powerful data structure that is essential in many aspects of computer science and programming. It is particularly useful for managing function calls, tracking state changes, and solving various algorithmic problems where a last-in, first-out order is required. Understanding stacks is fundamental for software developers and computer scientists alike.

4. PUSH OPERATION

Push operation can be performed in the below steps

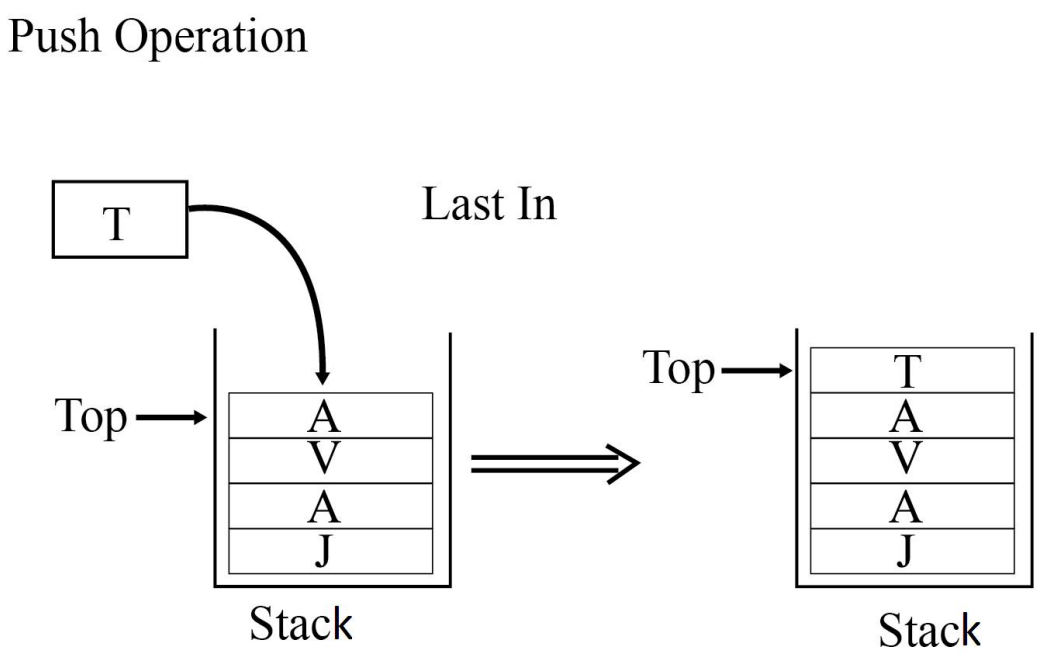
Step 1 – Checks stack has some space or stack is full.

Step 2 – If the stack has no space then display "overflow" and exit.

Step 3 – If the stack has space then increase top by 1 to point next empty space.

Step 4 – Adds item to the newly stack location, where top is pointing.

Step 5 – PUSH operation performed successfully.



Code Snippet of PUSH Operation

```
public void push(int x)
    {
        if (top >= (MAX – 1)) {
            System.out.println("Stack Overflow");
        }
        else {
            a[++top] = x;
            System.out.println("Item pushed into stack = "+x);
        }
    }
```

5. POP OPERATION

POP operation can be performed in the below steps

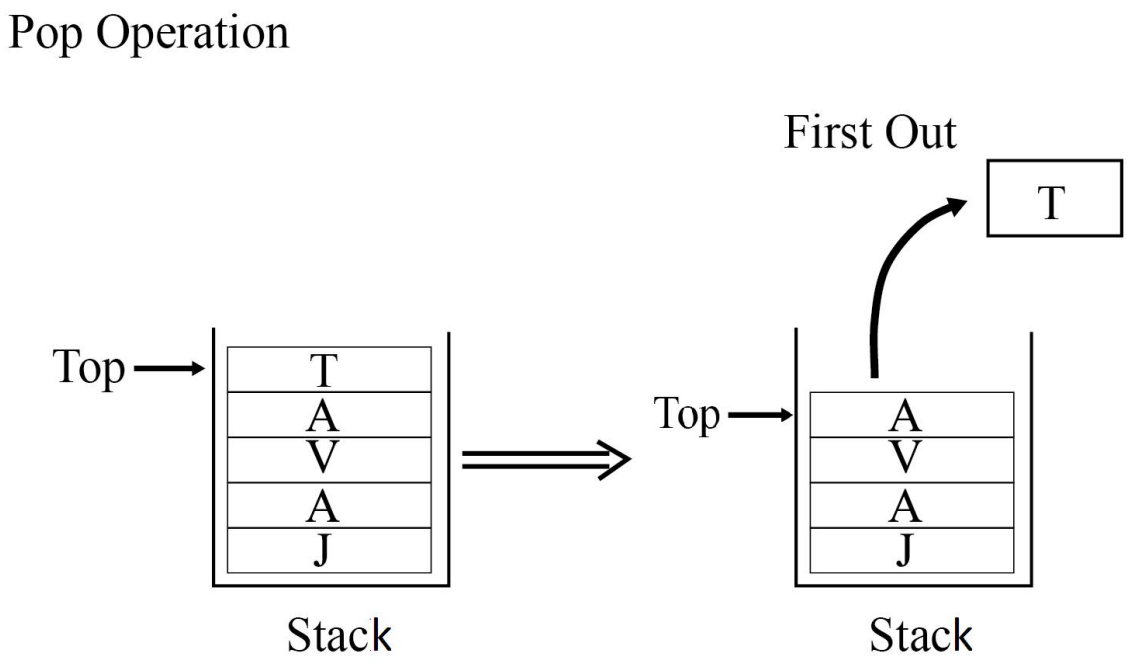
Step 1 – Checks stack has some element or stack is empty.

Step 2 – If the stack has no element means it is empty then display "underflow"

Step 3 – If the stack has element some element, accesses the data element at which top is pointing.

Step 4 – Decreases the value of top by 1.

Step 5 – POP operation performed successfully.

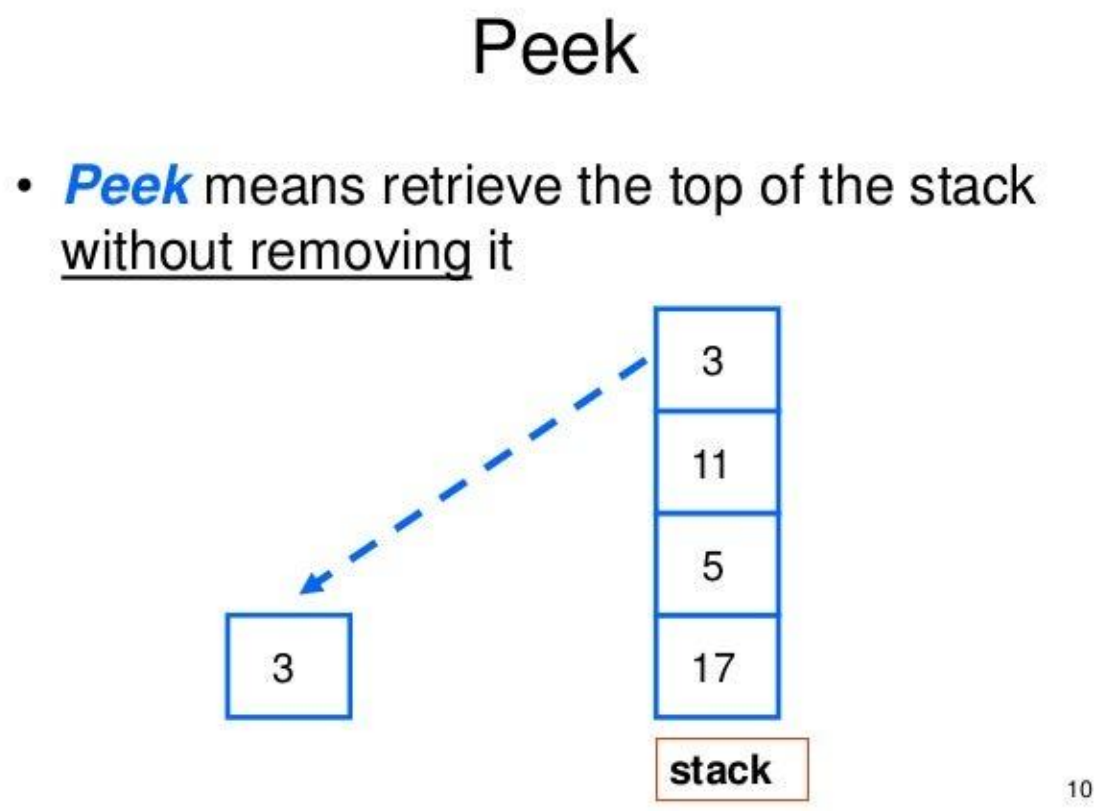


Code Snippet of POP Operation

```
public int pop()
    {
        if (top < 0) {
            System.out.println("Stack Underflow");
            return 0;
        }
        else {
            int x = a[top--];
            return x;
        }
    }
```

6. PEEK OPERATION

Using peek operation topmost element of the stack will be retrieved without deleting it from the stack. However, the Peek operation first checks if the stack is empty, i.e., if TOP = NULL, then an appropriate message is printed, else the value is returned.



Code Snippet of PEEK Operation

```
public int peek()
    {
        if (top < 0) {
            System.out.println("Stack Underflow");
            return 0;
        }
        else {
            int x = a[top];
            return x;
        }
    }
```

7. SIZE STACK & EMPTY STACK

	stack empty()	stack size()
1.	It is used to return whether the stack is empty	It is used to return the number of elements in the stack.
2.	Its syntax is –: empty();	Its syntax is –: size();
3.	Its return type is of boolean.	Its return type is of integer.
4.	It does not take any parameters.	It does not take any parameters.
5.	Its complexity is constant.	Its complexity is constant.

8. BROWSING HISTORY

Browsing history is often implemented using a stack data structure. A stack-based browsing history system allows users to navigate through web pages while maintaining a record of their visited pages. Here's an explanation of how a stack is used to manage browsing history:

1. Navigating Web Pages:

- As a user visits web pages, each page's URL is pushed onto a stack. The stack starts empty at the beginning of the browsing session.

2. Push Operation:

- When the user visits a new web page, the current page's URL is added (pushed) onto the top of the stack.

3. Going Back:

- If the user wants to go back to the previous page, they perform a "pop" operation on the stack, which removes the top URL.

- This simulates going back to the previously visited page.

4. Going Forward:

- In some browsing history implementations, if the user goes back and then decides to visit a different page, the pages that were popped off the stack (i.e., the pages they went back from) can be stored in a separate stack.

- If the user decides to go forward, URLs are popped from the separate stack and pushed onto the main stack, allowing them to simulate going forward.

5. Displaying History:

- Users can view their browsing history by examining the contents of the stack.

- The history is presented in reverse chronological order, with the most recently visited page at the top of the stack.

6. Bookmarks:

- In some browsing history systems, users can mark specific pages as bookmarks.

- Bookmarked pages may be stored separately, allowing for quicker access and retrieval.

7. Managing Stack Size:

- To prevent the stack from growing indefinitely, a maximum size is often defined. When the stack reaches its maximum size, older URLs are removed from the bottom of the stack to make room for new ones.

8. User Interface:

- Browsers typically provide user interfaces (such as the back and forward buttons) to make it convenient for users to navigate their browsing history.

Stack-based browsing history is a straightforward and efficient way to implement a user's web page navigation history. It allows users to retrace their steps and revisit previously viewed pages. By using a stack, the system ensures that the most recently visited pages are easily accessible and provides a simple mechanism for going back and forth in the browsing history. Additionally, bookmarks can be integrated for quicker access to favorite pages.

9. CODE IN C LANGUAGE

```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>


// Structure to represent a visited website

struct Website {

    char url[100];

    struct Website* next;

};


// Function to create a new website node

struct Website* createWebsiteNode(const char* url) {

    struct Website* newNode = (struct Website*)malloc(sizeof(struct Website));


    if (newNode != NULL) {

        strcpy(newNode->url, url);

        newNode->next = NULL;

    }


    return newNode;

}
```

```
// Function to push a visited website to the history
struct Website* push(struct Website* history, const char* url) {
    struct Website* newNode = createWebsiteNode(url);

    if (newNode != NULL) {
        newNode->next = history;
        return newNode;
    }

    return history;
}

// Function to pop a website from the history (remove the last visited website)
struct Website* pop(struct Website* history) {
    if (history == NULL) {
        printf("Browsing history is empty.\n");
        return NULL;
    }

    struct Website* temp = history;
    history = history->next;
```

```
    free(temp);

    return history;
}

// Function to delete the entire browsing history
struct Website* deleteHistory(struct Website* history) {
    while (history != NULL) {
        struct Website* temp = history;
        history = history->next;
        free(temp);
    }
    printf("Browsing history deleted.\n");
    return NULL;
}

// Function to display the browsing history
void displayHistory(struct Website* history) {
    struct Website* current = history;
    printf("Browsing History:\n");

    while (current != NULL) {
```

```
        printf("%s\n", current->url);
        current = current->next;
    }
}

int main() {
    struct Website* history = NULL; // Initialize an empty history

    int choice = 0;
    while (choice != 6) {
        printf("\n1. Add website to history\n");
        printf("2. Display browsing history\n");
        printf("3. Delete last visited website\n");
        printf("4. Delete entire history\n");
        printf("5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        if (choice == 1) {
            char input[100];
            printf("URL: ");
            scanf("%s", input);
```

```
            history = push(history, input);
        } else if (choice == 2) {
            displayHistory(history);
        } else if (choice == 3) {
            history = pop(history);
            printf("Last visited website deleted.\n");
        } else if (choice == 4) {
            history = deleteHistory(history);
        } else if (choice == 5) {
            // Free remaining memory before exiting
           // freeHistory(history);
        } else {
            printf("Invalid choice. Please try again.\n");
        }
    }

    return 0;
}
```

10. OUTPUT

Output

```
1. Add website to history
2. Display browsing history
3. Delete last visited website
4. Delete entire history
5. Exit
Enter your choice: 1
URL: http://www.example.com/
1. Add website to history
2. Display browsing history
3. Delete last visited website
4. Delete entire history
5. Exit
Enter your choice: 1
URL: http://example.com/
1. Add website to history
2. Display browsing history
3. Delete last visited website
4. Delete entire history
5. Exit
Enter your choice: 1
URL: https://www.example.com/
1. Add website to history
2. Display browsing history
3. Delete last visited website
4. Delete entire history
5. Exit
Enter your choice: 2
Browsing History:
https://www.example.com/
http://example.com/
http://www.example.com/

1. Add website to history
2. Display browsing history
3. Delete last visited website
4. Delete entire history
```

11. CONCLUSION

The concept of a stack-based browsing history offers a unique and innovative approach to web navigation. This method introduces the idea of treating web pages as items on a stack, where each new page visited is pushed onto the stack, and navigating back involves popping the stack. This approach has several noteworthy implications and benefits.

1. Sequential and Efficient Navigation: A stack-based history system provides a highly intuitive and sequential way to navigate the web. Users can easily retrace their steps by moving back and forth through the stack, which makes the browsing experience more predictable and user-friendly.

2. Granular Control: Users have granular control over their browsing history. They can jump directly to any previously visited page by selecting it from the stack, which is especially valuable when trying to return to a specific point in their browsing session.

3. Reduced Clutter: The stack-based model inherently trims down the browsing history, as it only keeps track of the pages actively visited in the current session. This reduces clutter and allows users to focus on relevant pages.

4. Enhanced User Experience: For both novices and experienced web users, the stack-based browsing history offers an improved user experience. The stack is an intuitive metaphor, and the predictability of forward and backward navigation can reduce frustration and streamline the browsing process.

5. Web Application Integration: Implementing a stack-based browsing history into web browsers or applications is feasible and could become a valuable feature, especially for applications that require users to navigate through complex hierarchies of web pages.