VIVEKANAND EDUCATION SOCIETY'S INSTITUTE OF TECHNOLOGY
DEPARTMENT OF INFORMATION TECHNOLOGY

Browser History Management using Stack and Queue

By - Gitesh Peswani-47
Jatin Rohra-53
Saloni Wadhawani-63

Guided by - Ms. Kajal Jewani

Content :

- Introduction to the Project
- Problem Statement
- Objectives of the Project
- Scope of the Project
- Requirements of the System (Hardware, Software)
- ER Diagram of the Proposed System
- Data Structure & Concepts Used
- Algorithm Explanation
- Time and Space Complexity
- Front End
- .Challenges and Solutions
- Future Scope
- Code
- Output Screenshots
- Conclusion
- References (in IEEE Format)

Introduction to Project:

- Imagine a browser as a journey through web pages. To keep track of where you've been and where you can go, we use two powerful data structures: stacks and queues.

- Stack: Think of it as a pile of plates. The last plate you put on is the first one you take off. In a browser, the most recent page you visited is at the top of the stack. This helps you go back to the previous page.

- Queue: Imagine a line at a ticket counter. The first person in line is the first one served. In a browser, the first page you visited is at the front of the queue. This helps you go forward to the next page in your browsing history.

- By using these structures together, browsers can efficiently manage your navigation and let you explore the web seamlessly.

Problem Statement

- Navigation Complexity: Users struggle to efficiently navigate back and forth between previously visited pages in a browser session, especially during non-sequential browsing.
- Limited Sequential History: Browsers often maintain a linear history log, making it difficult to revisit specific pages without retracing every step.
- Stack-Based Navigation: The "back" and "forward" buttons can benefit from using a stack, where the most recently visited pages can be pushed and popped to streamline navigation.
- Queue for Chronological History: A queue can help maintain a proper chronological history of all visited pages, allowing users to revisit any page from their entire session, not just the most recent ones.
- Efficiency Trade-Offs: Managing both navigation (via a stack) and long-term history storage (via a queue) must be optimized for memory usage and performance, ensuring that the system remains responsive and scalable.
- Data Privacy and Storage: The system should balance the need for efficient browsing history management with user privacy, ensuring that sensitive data is handled appropriately and securely stored.

Objectives of the project

- **Include a credit or citation Efficiently store and retrieve visited URLs**: Implement a data structure that can quickly store and access recently visited URLs.

- **Provide seamless navigation**: Enable users to easily go back and forth between visited pages using a simple interface.

- **Handle complex browsing scenarios**: Accommodate actions like opening new tabs, closing tabs, and restoring previous sessions.

- **Optimize memory usage:** Minimize memory consumption while. maintaining a reasonable history length.

- **Ensure user privacy**: Protect user privacy by securely storing browsing history data.

SCOPE OF THE PROJECT

- **Back Navigation:** The system should allow the user to navigate backward through their browsing history, revisiting previously visited web pages.

- **Forward Navigation:** The system should also allow the user to navigate forward through their browsing history, resuming from where they left off.

- **History Management:** The system should efficiently manage the browsing history, storing and retrieving web pages as needed.

- **Memory Management:** For large history lists, implement efficient memory management techniques to avoid performance issues.

- **Cross-Browser Compatibility:** Ensure that the application works seamlessly across different browsers and platforms.

- **User Experience:** Focus on providing a smooth and intuitive user experience, especially when dealing with complex navigation scenarios.

REQUIREMENTS

Technical Requirements:

- Programming language: Python, Java, C++, or JavaScript.
- Data structures: Stack and Queue.
- Database: MySQL, MongoDB, or SQLite..
- Operating System: Windows, macOS, or Linux

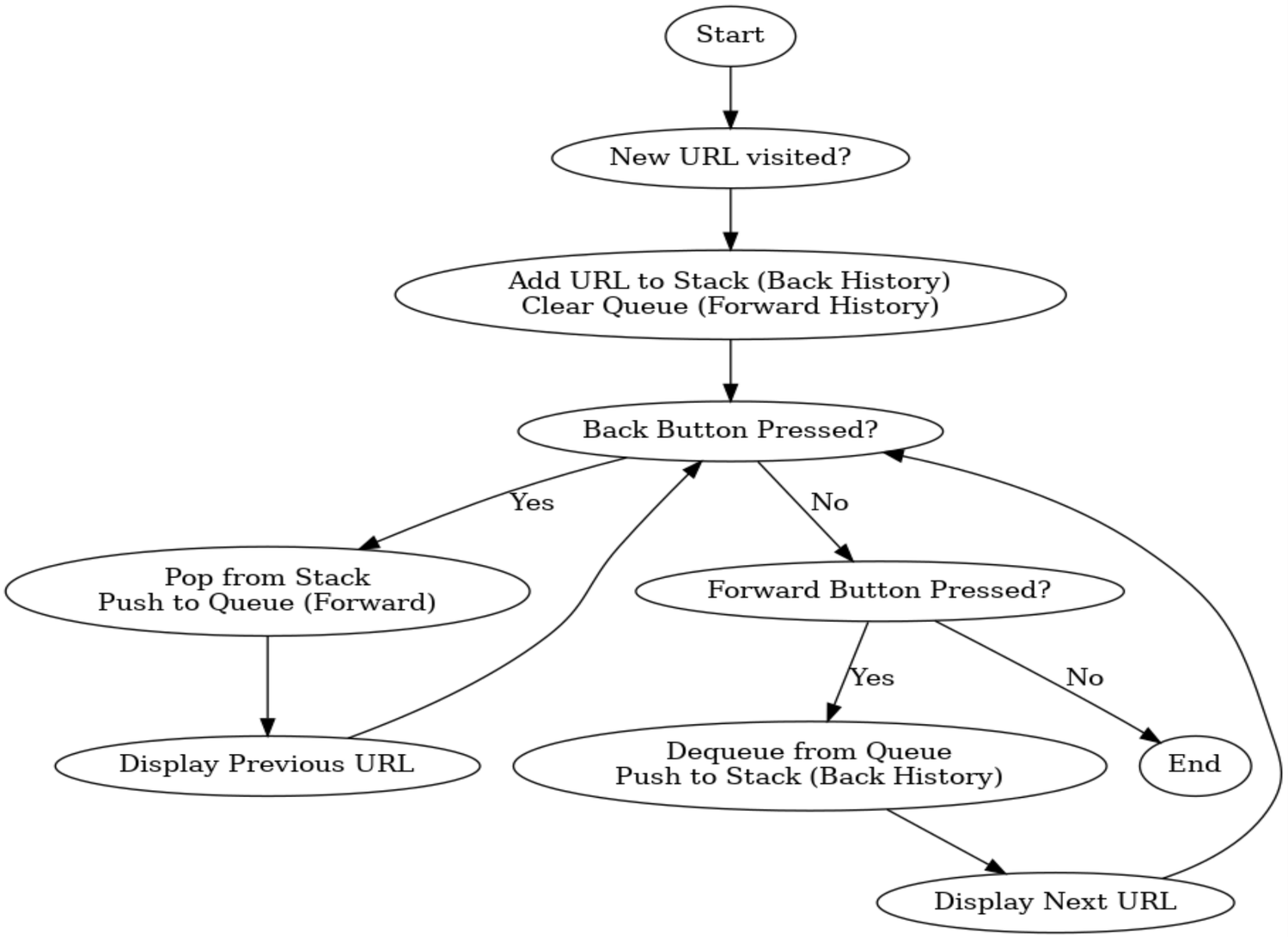
Hardware Requirements:

- Processor: Intel Core i3 or equivalent.
- RAM: 4 GB or more.
- Storage: 256 GB or more.

Software Requirements:

- IDE: Eclipse, Visual Studio, or IntelliJ.
- Database management system.
- Browser emulator (optional).

Flow Chart


Start
New URL visited?
Add URL to Stack (Back History)
Clear Queue (Forward History)
Back Button Pressed?
Yes
No
Pop from Stack
Push to Queue (Forward)
Forward Button Pressed?
Yes
No
Display Previous URL
Dequeue from Queue
Push to Stack (Back History)
End
Display Next URL

Data Structures and Concepts used

Data Structures:

- Stack (Back Stack)
- Queue (Forward Queue)
- Linked List (History List)
- Database(History Storage)

Concepts:

- Last-In-First-Out (LIFO) - Stack
- First-In-First-Out (FIFO) - Queue
- Depth-FirstSearch(DFS) Navigation
- Database indexing - History storage

ALGORITHM EXPLANATION

Key Data Structures:

Stack (**Backward Navigation**): Stores URLs of pages visited in backward direction.
Queue (**Forward Navigation**)**:** Stores URLs of pages visited in forward direction.

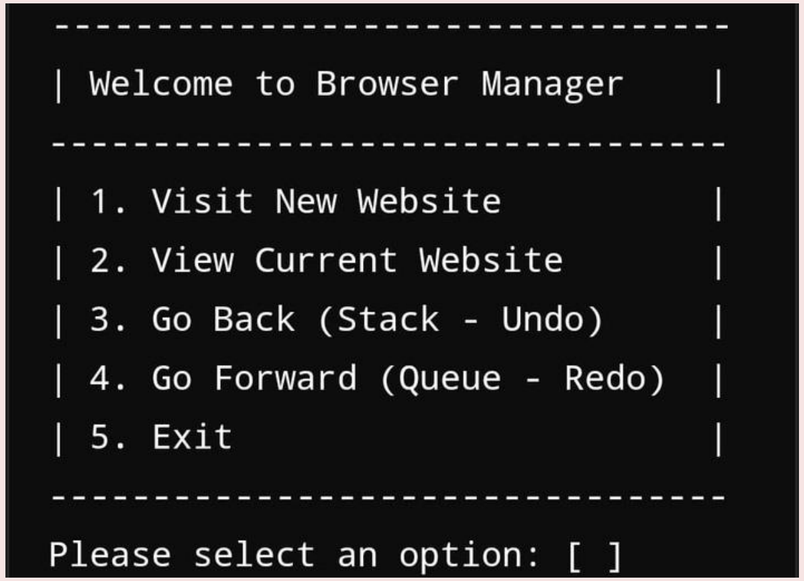
Algorithm Steps:

- New Page Visit: Push current page URL to stack, set new page URL as current, and clear queue.
- Backward Navigation: Dequeue URL from queue, set as current page URL, and push to stack.
- Forward Navigation: Pop URL from stack, set as current page URL, and enqueue to queue.
- Current Page URL: Retrieve from top of stack.
- Example: Visiting pages A, B, C, D, E, then navigating back three times, and forward twice.

Program Complexity

- **Time Complexity:** O(1) for visiting a new page, back navigation, and forward navigation.
- **Space Complexity:** O(n) for both stack and queue, where n is the number of pages visited.

FRONTEND


Welcome to Browser Manager
1. Visit New Website
2. View Current Website
3. Go Back (Stack - Undo)
4. Go Forward (Queue - Redo)
5. Exit
Please select an option: [ ]


Enter the Website URL: [example.com]
Website 'example.com' visited.
Added to history.
Press any key to return to Main Menu


Going Back to Previous Website
Previous Website: oldsite.com
Press any key to return to Main Menu


Going Forward to Next Website
Next Website: example.com
Press any key to return to Main Menu

CODE

https://github.com/GiteshPeswani/dsampr.git

Challenges and Solutions

- Challenges:
  - Handling large browsing history
  - Managing concurrent navigation
  - Handling page reloads and refreshes
  - Supporting advanced navigation features
  - Ensuring data integrity and security

- Solutions:
  - Implement bounded stack and queue
  - Use thread-safe data structures and synchronization
  - Detect and handle page reloads and refreshes
  - Extend implementation to support advanced features
  - Implement data encryption and access control

Future Scope

- Advanced features: Tab management, search history, incognito mode, synchronization, and extensions.
- Performance improvements: Optimized data structures, caching, and asynchronous operations.
- User experience enhancements: Gesture support, voice commands, and personalized recommendations.
- Integration with other services: Cloud storage, social media, and password management.

OUTPUT SCREENSHOT

```
Browser History Menu:
1. Visit a new URL
2. Go back
3. Go forward
4. Display browsing history
5. Exit
Enter your choice: 1
Enter the URL to visit: https://web.whatsapp.com/
Visited: https://web.whatsapp.com/

Browser History Menu:
1. Visit a new URL
2. Go back
3. Go forward
4. Display browsing history
5. Exit
Enter your choice: 1
Enter the URL to visit: https://www.youtube.com/
Visited: https://www.youtube.com/
```

```
Browser History Menu:
1. Visit a new URL
2. Go back
3. Go forward
4. Display browsing history
5. Exit
Enter your choice: 1
Enter the URL to visit: https://classroom.google.com/u/1/c
    /NzAwODU4MzQwNzE2
Visited: https://classroom.google.com/u/1/c/NzAwODU4MzQwNzE2

Browser History Menu:
1. Visit a new URL
2. Go back
3. Go forward
4. Display browsing history
5. Exit
Enter your choice: 2
Moved back to: https://www.youtube.com/
```

```
Browser History Menu:
1. Visit a new URL
2. Go back
3. Go forward
4. Display browsing history
5. Exit
Enter your choice: 3
Moved forward to: https://classroom.google.com/u/1/c/NzAwODU4MzQwNzE2

Browser History Menu:
1. Visit a new URL
2. Go back
3. Go forward
4. Display browsing history
5. Exit
Enter your choice: 4
Browsing history:
https://web.whatsapp.com/
https://www.youtube.com/
https://classroom.google.com/u/1/c/NzAwODU4MzQwNzE2
```

REFERENCES

[1] D. L. Nkweteyim, "Data Structures for Information Retrieval," IST-Africa 2014 Conference Proceedings, Paul Cunningham and Miriam Cunningham, Eds. IIMC International Information Management Corporation, 2014, pp. 1-8. ISBN: 978-1-905824-44-1.

[2] O. Aziz, T. Anees, and E. Mehmood, "An Efficient Data Access Approach With Queue and Stack in Optimized Hybrid Join," IEEE Access, vol. 9, pp. 41261-41274, Mar. 2021, doi: 10.1109/ACCESS.2021.3064202.

Literature Review

1. **Data Structures for Information Retrieval:** examines efficient ways to handle large-scale information retrieval tasks using the vector-space model (VSM). The study details the process of constructing an index for document collections, specifically focusing on the OHSUMED dataset. The paper describes the usage of binary search trees and linked lists for handling index terms, document vectors, and postings, as well as trade-offs between memory usage and processing time. Nkweteyim's approach leverages term frequency-inverse document frequency (TF-IDF) for weighting terms and utilizes cosine similarity to compare document vectors with search queries. Results from the OHSUMED dataset show that even with modest memory, large datasets can be indexed efficiently by dividing data into batches and utilizing secondary storage.

2. **An Efficient Data Access Approach With Queue and Stack in Optimized Hybrid Join:** This paper introduces two optimized algorithms, P-HYBRIDJOIN and QaS-HYBRIDJOIN, aimed at improving the efficiency of near-real-time data warehousing. These algorithms focus on reducing disk I/O costs and improving service rates in the semi-stream join process, a critical operation in Extract, Transform, Load (ETL) pipelines. The study addresses the limitations of the traditional HYBRIDJOIN algorithm, which processes data inefficiently by overwriting older data partitions in the buffer. P-HYBRIDJOIN uses two parallel disk buffers, allowing the join operation and disk loading to occur simultaneously. QaS-HYBRIDJOIN adds a stack mechanism to prioritize recent attributes for joining while still considering older attributes, increasing the number of matching records processed.

Conclusion

A browser history management system using stack and queue provides a robust solution for tracking and navigating user history. It offers features like forward/backward navigation, history list, bookmarking, and session management. Key considerations include user experience, scalability, and security.

Thank you!