Indian Institute of Information Technology and Management – Kerala (IIITM-K)

M.Sc. in Computer Science

Data Structures and Algorithms

(Mini-Project 1)

ENTITLED

(A Web-History Tracking Solution using Linked List)

Submitted By:

Group 1

M.Sc. Computer Science

Dated on: November 8th, 2020

ACKNOWLEDGEMENT

It is our privilege to express our sincerest regards to our project coordinator, ***Mr. Alex P James****, for his valuable inputs, able guidance, encouragement, whole-hearted cooperation and constructive criticism throughout the duration of our project. We deeply express our sincere thanks to our* ***Miss Aswani A R*** *&* ***Miss Jessy Scaria*** *for being there for any adverse situation while the development of the project on the topic* ***“A Web-History Tracking Solution using Linked List”*** *towards the Mini-Project 1 of the subject as the Data Structures and Algorithms. We take this opportunity to thank all our lecturers who have directly or indirectly helped our project. Last but not the least we express our thanks to our whole team for their cooperation and supporting each other throughout the development of the project.*

The list of the team-members, involved in the development of the solution, is given on the next page!

Team Members

(Group-1)

Idea, Planning, and Coordination among Team	Ravi Prakash Nitul A U Krishnendu M B Mohamed Haris P M Deepak C Varghese Christy M Thomas Merin Jose Murli Krishna
Programing Team	Ravi Prakash Nitul A U Shivam Krishnendu M B Mohamed Haris P M Ananthkrishnan A Aleena Rodrigues Muhammed Shahil
Report Making Team	Ravi Prakash Deepak C Varghese Christy M Thomas Aleena Rodrigues Ananthu P S Ajul Krishna K V Arya Mol M Govind G S Sanil Sadanandan M Shaghin P C Alan Sunny Murli Krishna Jithin J K
Other Valuable Efforts	Ananthkrishnan A Merin Jose B Sahithi Reddy Sudheesh G L Arya Rajan Lekshmi S

Introduction

We were provided with the following problem statements:

Problem Statement

"You are a python developer specialized in the linked list. The customer has a very specific need that requires the extensive use of a Linked List. They have a requirement to build a tracking algorithm that can store and track the list of pages you visited while browsing the internet along with timestamp information. Write a program that takes in the website addresses or links and corresponding timestamp information. There should be an option to clear the tracked information and also have an option to backup history. The program should be able to also identify duplicate entries. After every 100 links you visit, the program refreshes the memory and starts from scratch."

Required parts of submission:

1. Come up with 5 solutions to this problem within your group.
2. Select the best solution and provide half a page write-up on the reasons for selecting it. This half-page write-up needs to be submitted through google Forms available in the google classroom submission link.
3. Submit a report written in a word document, including your python code, and the outputs obtained running in Google co-lab. Hint - you can make use of a simple, Double, or Circular Linked List.

Problem Definition

We are going to build a tracking algorithm that can store and track the list of pages we visited while browsing the internet along with timestamp information. In this way, after looking at the problem statement, we can define it in the following manner:

Objective:

Our objective is to:

- Develop an ***efficient*** solution that can track the user's browsing history.
- Solution must implement the concepts of ***Linked Lists***.

Requirement:

There should be some mechanism for -

- *storing* ***details*** *of visited web-address (URLs)*
- *removing* any tracked record details
- *backing up the tracked record*
- detecting *duplicate* records
- *restoring* the available backup (***additional feature***)
- keeping the *maximum tracked records* in memory ***not*** *more than 100* in count

Scopes of this solution:

The solution can have the following scopes:

1. It can be useful in history storage for web-browsers
2. An efficient method can lead to faster accessibility
3. Its backup feature can be useful in case history is removed accidentally
4. Limited storage can prove to be better for the health of the system memory
   and many more...

Planning

We are required to come up with at least **five** different solutions for the same problem. Obviously, there can be many different solutions for a given problem, but deciding what can be considered to be the best five is really a difficult task. In order to resolve this thing, we needed to find the answers to a few most important questions.

Frequently Asked Questions:

1. How can we import data (URL & timestamp)?
2. How is the solution important in the real world and how to resolve it programmatically?
3. Should we develop different solutions for different browsers? Why yes or why no?
4. Should there be some uniqueness for the tracked history for every visitor/user?
5. Do we need to collect the URLs or will it be given as input?
6. How to tackle duplicate entries?
7. Do we need to design the program in such a way that the duplicate entries will be removed the moment they are entered?
8. How to back up and clear the history?
9. Are we taking browsing history as input?
10. Which type of linked list will be best in the solution: traditional singly linked list or a circular link?
11. How to deal with the situation if the tracked history reaches 100 in number?
12. What are the pros of our solutions?

    etc.

After finding the answers to these questions, we came up with many different solutions for the given problem.

Suggested Solutions

We have analyzed various possible solutions which can be implemented to fulfill the requirement. All the possible outcomes have been discussed in detail within the group to identify the best suitable approach. Both advantages and disadvantages have been analyzed for all the approaches using sample models. Parameters like accessibility, Memory utilization, processing time, programming complexity, and resource management were considered.

We figured out some of the most important, that were required to be kept in mind while solving the problem programmatically.

Program Requirements:

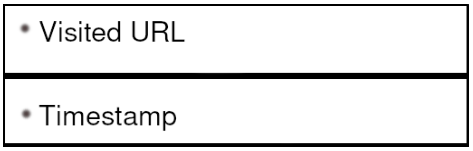
1. Time	->	Parameter
2. URL	->	Parameter
3. Node Count	->	Parameter
4. Count	->	Function
5. Insertion	->	Function
6. Deletion	->	Function
7. Backup	->	Function
8. Duplicacy Detection	->	Function
9. Searching	->	Function
10. Restore	->	Function

After the planning phase, finally we came up with the following **five approaches** to solve the given problem -

1. *Linear-Storage Mechanism*
2. *Branch-Storage Mechanism*
3. *Hash-Map-Based Solution*
4. *Circular-Doubly Linked-List*
5. ***Bucket-Storage Mechanism***

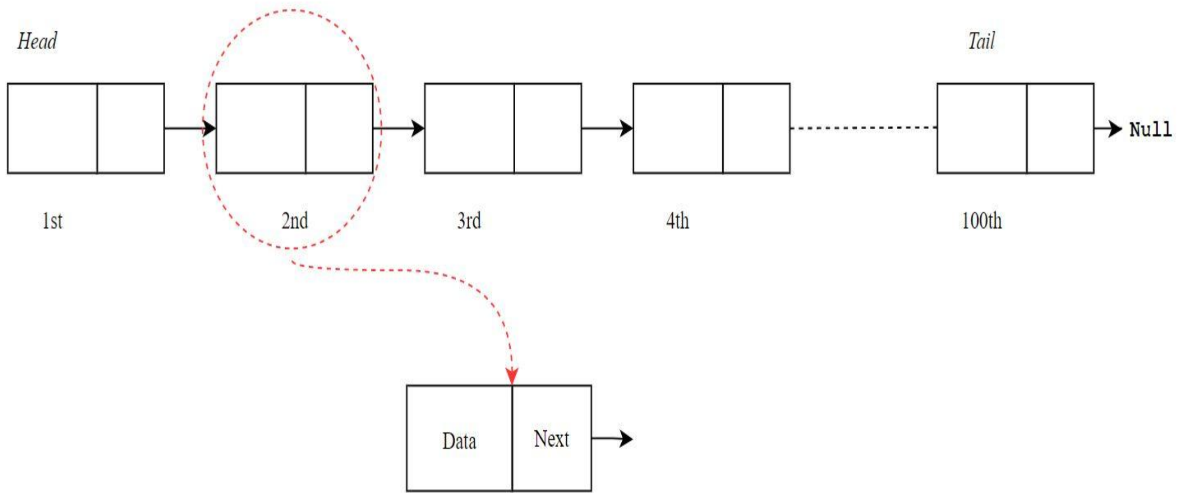
We consider our data to be encapsulated in one place with two fields mainly:

1. Visited URL
2. Timestamp



(Data)

1. Linear-Storage Mechanism



(Data in Linear-Storage)

The **Linear-Storage Mechanism** makes the fundamental use of traditional *singly linked lists*. A singly linked list is a type of linked list that is **unidirectional**. Due to being unidirecional, it can only be traversed in a single direction i.e. from the first node or *head* to the last node, generally called *tail*. That gives us an idea that it can be a possible approach to store the tracked history of visited web-addresses, using linked lists. A single node contains **data** and **a reference** to the next node which helps in maintaining the structure of the list. The last node tends to have a *NONE* as reference that signifies the end of the list.

The method has following pros and cons:

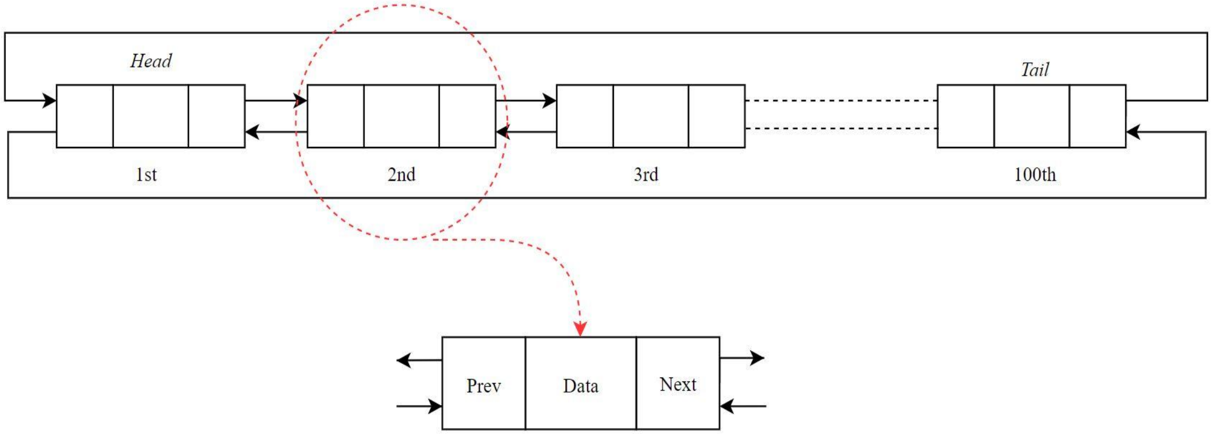
Pros:

- Easiest way to implement the linked lists while defining the data structure mechanism.
- Easy to explain and understand the mechanism.
- Very easy to debug.

Cons:

- Searching complexity is very high i.e. ***O(n)***, where the *n* is the maximum number of elements in the list.
- Operations like searching, sorting and deletion are done in a linear fashion.
- Due to being a very basic approach, an expert (as we are considered) can have many better approaches.

2. Circular-Doubly Linked List



(Data in Circular-Doubly Linked List)

The **Circular-Doubly Linked List Mechanism** implements a mixed concept of the *Circular Linked Lists* as well as the *Doubly Linked Lists*. As the name suggests, in this list any consecutive element consists of *references to both,* ***previous*** *and* ***next*** *element*. The last node and the first node have references to each other, making it a circular linked list, too. In Circular Doubly LinkedList, list can be traversed from both the directions i.e. from head to tail or from tail to head and Jumping from head to tail or from tail to head is done in constant time. And it enables Auto - Deletion (set-100)

The method has following pros and cons:

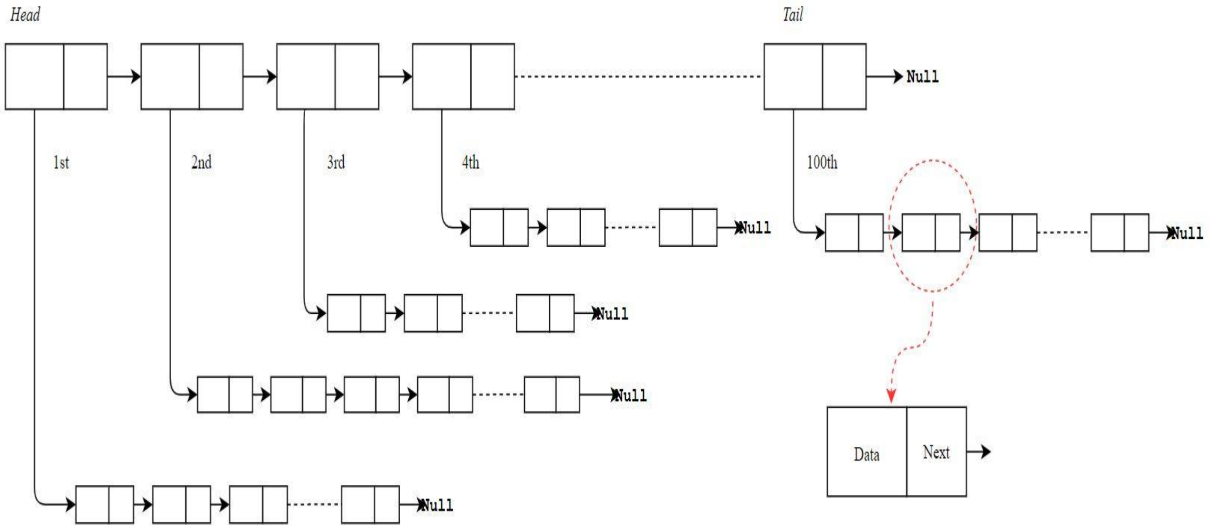
Pros:

- Easier to implement and better than a singly linked list.
- It can have a track onto the previous and next visited web-addresses.
- Debugging is not very hard
- No need to have a deletion mechanism for elements more than 100, if the maximum number of elements that can be accommodated are set to be 100 only.

Cons:

- Searching complexity is very high i.e. ***O(n)***, where the *n* is the maximum number of elements in the list.
- Operations like searching, sorting and deletion are done in a linear fashion.
- Unnecessary usage of references as its two-way tracking does not provide any bigger efficiency over the Linear-Storage Mechanism.
- The references must be handled carefully to avoid the loss of the data.

3. Branch-Storage Mechanism



(Data in Branch-Storage)

The **Branch-Storage Mechanism** makes an *extensive use of the Linked Lists*. Here, ***every NODE** of the linked list can also have a **reference to a new linked list**.* In this situation, if a NODE holds data about one parent URL, the linked list that can be referenced by it, might hold the visited URLs from the same domains. If it is done, the searching any visited web-address or the duplicate one will be far more easier.

The method has following pros and cons:

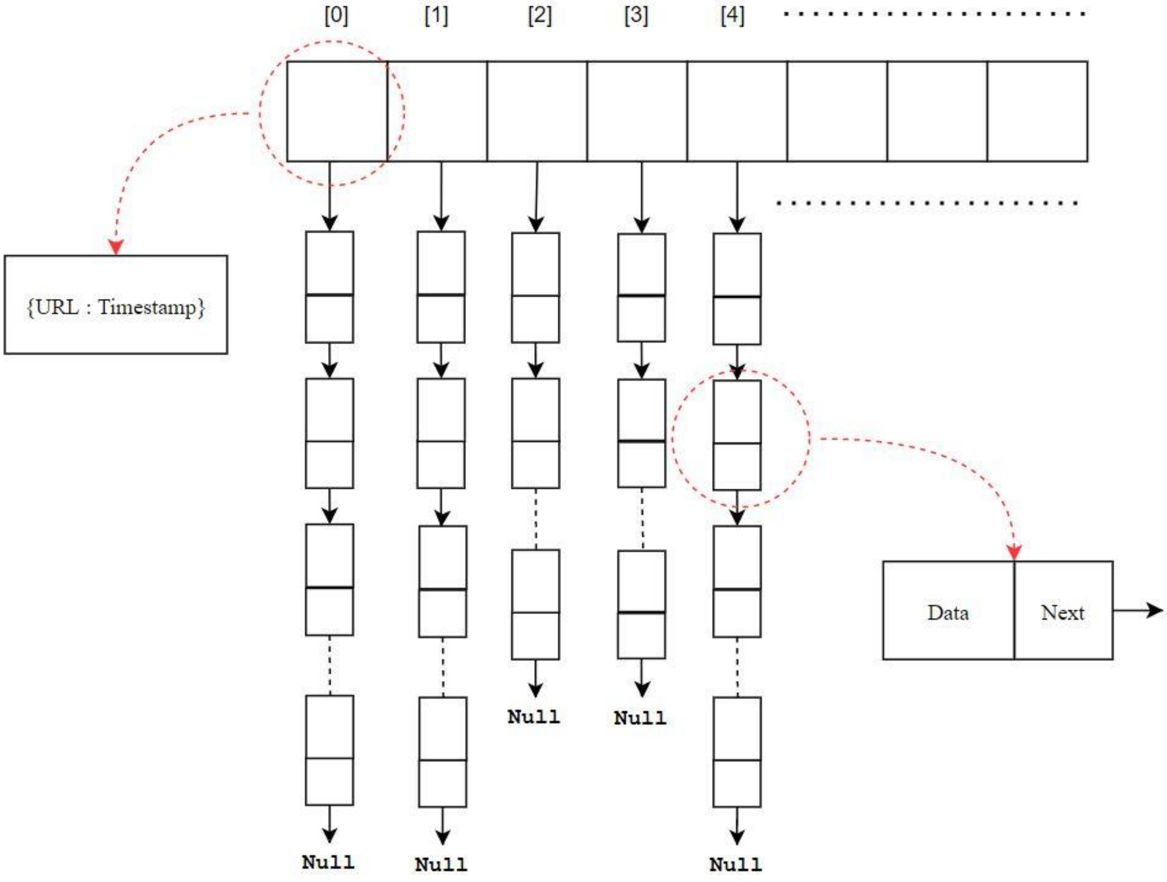
Pros:

- Purely uses linked lists concept.
- Faster operations viz. Searching, duplicate detection or deletion etc.. in terms of the time.
- An expert level approach.

Cons:

- Very complex to implement.
- Debugging is difficult.
- As any particular NODE can have the reference to any number of similar linked-lists, handling references is quite difficult.
- Limiting the number of data items to 100 is not an easy task here, and in some specific **it can turn into just a singly linked list**.

4. Hash-Table Based Solution



(Data in Hash-Table)

The **Hash-Table Based Approach** implements an array of **dictionaries** having a key-value pair as **URL** a **linked list** holding all the different timestamps when the same URL was visited.

The method has the following pros and cons:

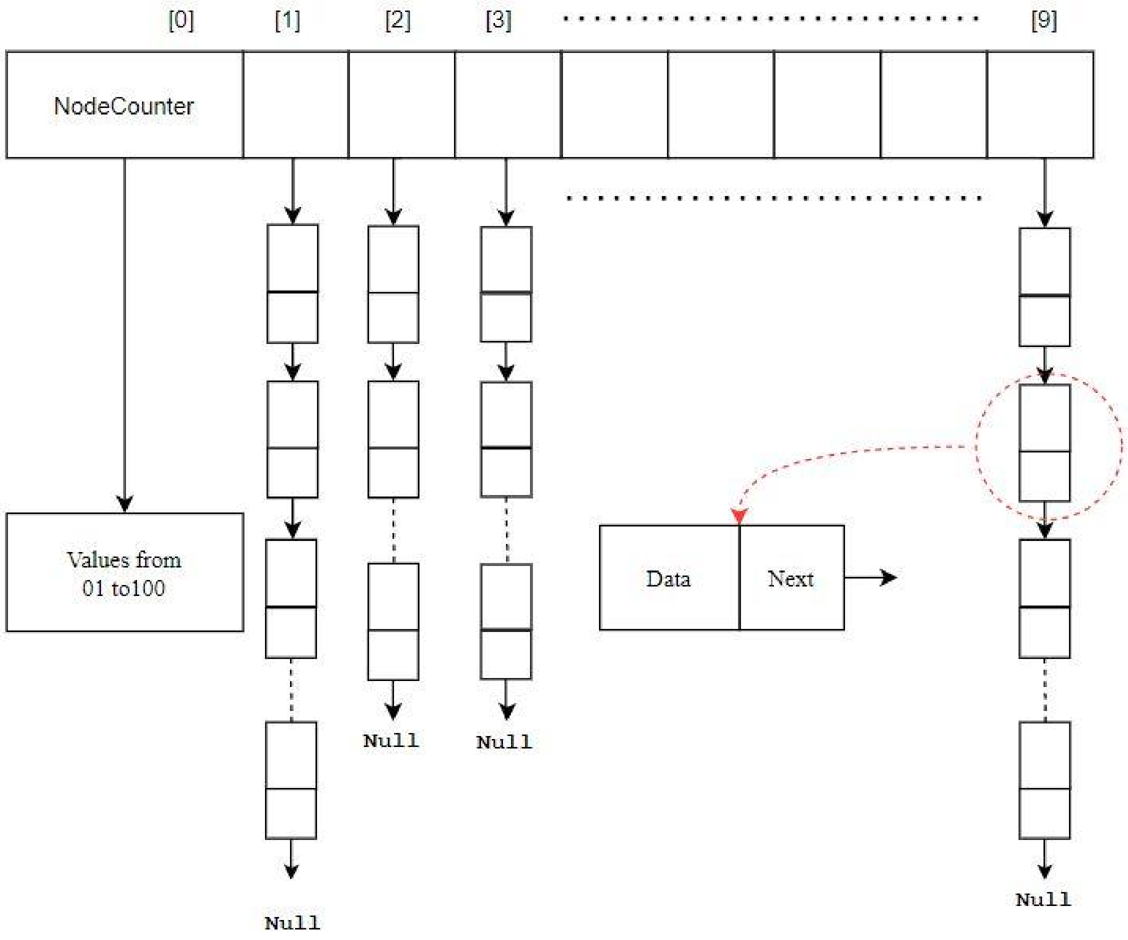
Pros:

- It can be an efficient way of searching duplicate entries.
- An innovative approach.

Cons:

- Complex to implement and debug.
- Extensive use of the array.
- Dictionaries can grow up to size 100 in the worst case and it'll not be a linked list-based approach.

5. Bucket-Storage Mechanism



(Data in Bucket-Storage)

The **Bucket-Storage Mechanism** is named after its bucket-like structure in which the history data is stored. It actually consists of a hybrid *array of **10 elements***, only. Here the first element is a numeric counter that *tracks the total number of tracked history data* leading not to exceed 100. Whereas, as far as concerned about the other 9 elements of the array, each of them hold a singly linked list.

Generally, the **first** element of the array is **indexed by 0**, and indexing of all the 10 elements in the array becomes 0, 1, 2, 3, 4, 5, 6, 7, and 9. Utilizing this notation, we can define a **hash function** that can map every new element i.e. visited web address to a single digit number lying between 1 and 9 only. The mapping is done on the basis of the timestamp i.e. adding all the digits of time until a single-digit number is obtained.

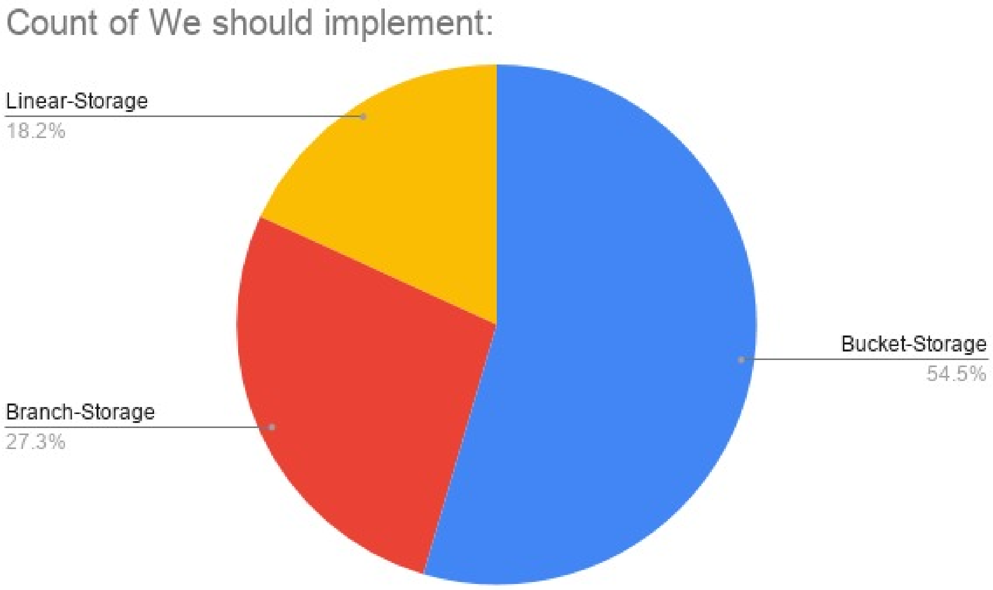
Pros:

- It has one-tenth searching complexity as compared to the Linear-Storage Mechanism and Circular-Doubly Linked List.
- Fantastic use of the hashing along with linked lists
- Debugging and implementation are not very difficult.
- Tracking the total number of historic elements is very easy as the array's first element is the counter.
- Very innovative approach and have future scope for research too.
-
- Most preferred method as per the voting.

Con:

- Requires the basic understanding to design a hash function.

Choosing the Best algorithm on the bases of the voting:



Implemented

Basic Structure:

The **Bucket-Storage Mechanism** was finalized after the voting of all the members

.

Structure of the NODE -

Parameters: 1. URL 2. TimeStamp 3. Bucket Index (highlighted thing is a User-Input)	**Datatype:** 1. String 2. DATE - TIME 3. Numeric
Methods: 1. Node() 2. printNodeData()	**Significance:** 1. Constructure 2. nodeName.printNodeData() (**Node()** will itself ask for input)

Structure of the Bucket -

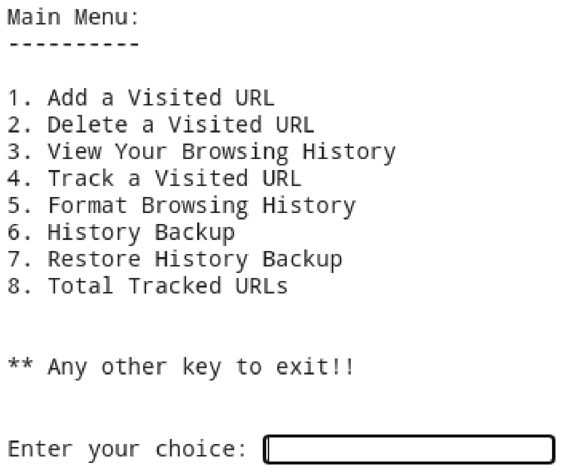
Parameters: 1. Head 2. Tail	**Datatype:** 1. NODE 2. NODE
Methods: 1. bucketLinkedList() Private Methods 1. __checkUnderflow__() Public Methods 1. addNode() 2. deleteData() 3. checkDuplicateEntries() 4. trackDateWise() 5. printList() 6. printURLOnly()	**Significance:** 1. Constructure 1. Checks if history exists or not 1. Stores new web-address details 2. Removes existing history details 3. Searches for duplicate URLs 4. Searches date-wise history 5. Displays entire history in details 6. Displays entire history URLs

Structure of the Main History Tracker -

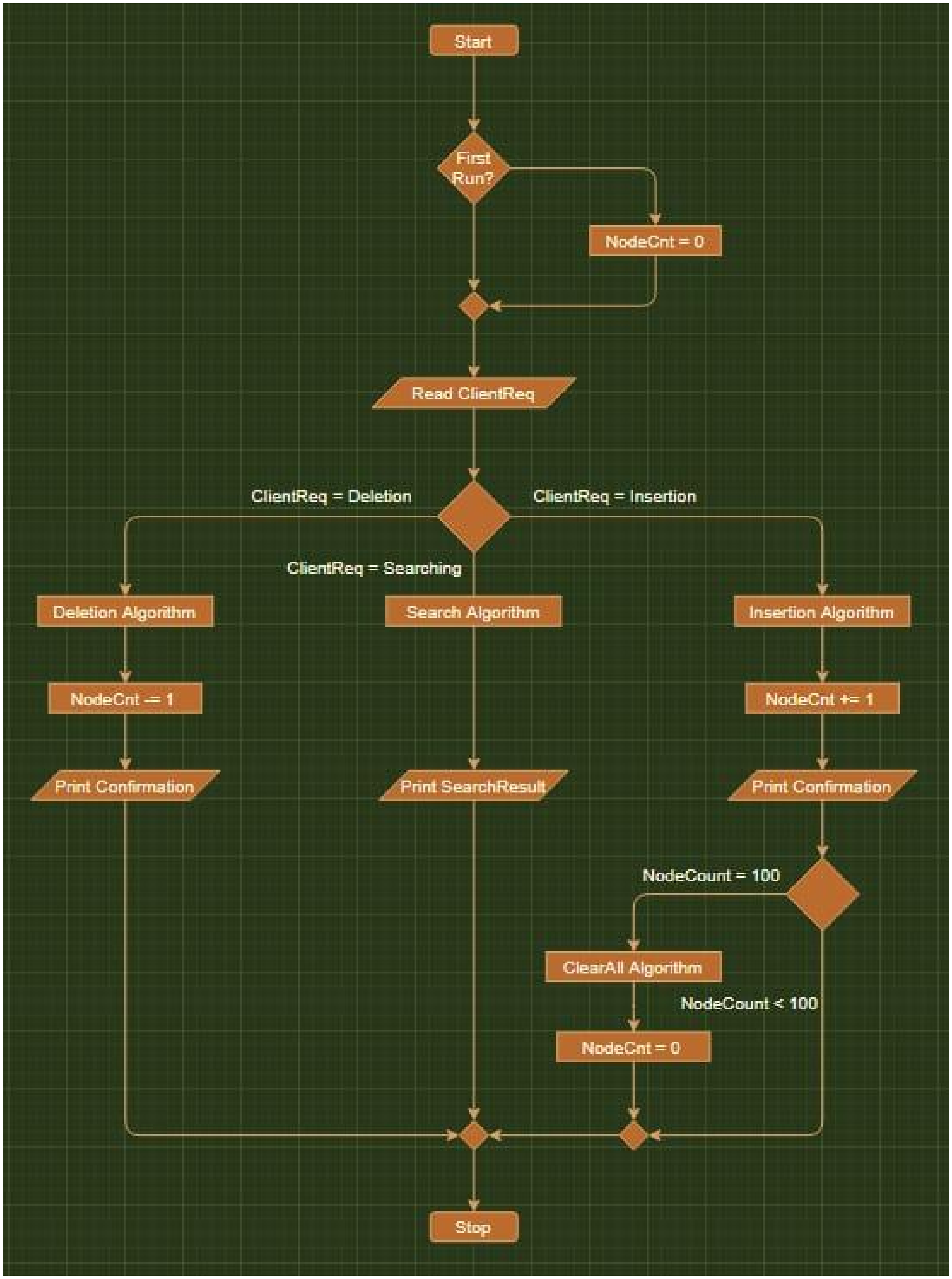
Parameters:	**Datatype:**
1. totalNodes 2. History 3. backup	1. numeric 2. Hybrid array of: a. Number (Nodes' Counter) b. Linked-Lists (Buckets) 3. Bucket
Methods: 1. historyTracker() Private Methods 1. __totalNodeCounter__() 2. __freshMemory__() 3. __checkUnderflow__() Public Methods 1. insert() 2. delete() 3. viewAll() 4. printBucket() 5. search() 6. refreshMemory() 7. backupHistory() 8. restoreBackup() 9. countHistory()	**Significance:** 1. Constructure 1. Tracks total numbers of records 2. Removes all the tracked records 3. Checks if history exists or not 1. For adding new history details 2. For removing a visited history 3. To display all browsing history 4. To print entries from 1 Bucket 5. To track the browsing history 6. drives __freshMemory__() 7. Backs up the current history 8. Recovers from the backup 9. drives __totalNodeCounter__()

Program flow:

The program serves the following features as the Main-Menu:

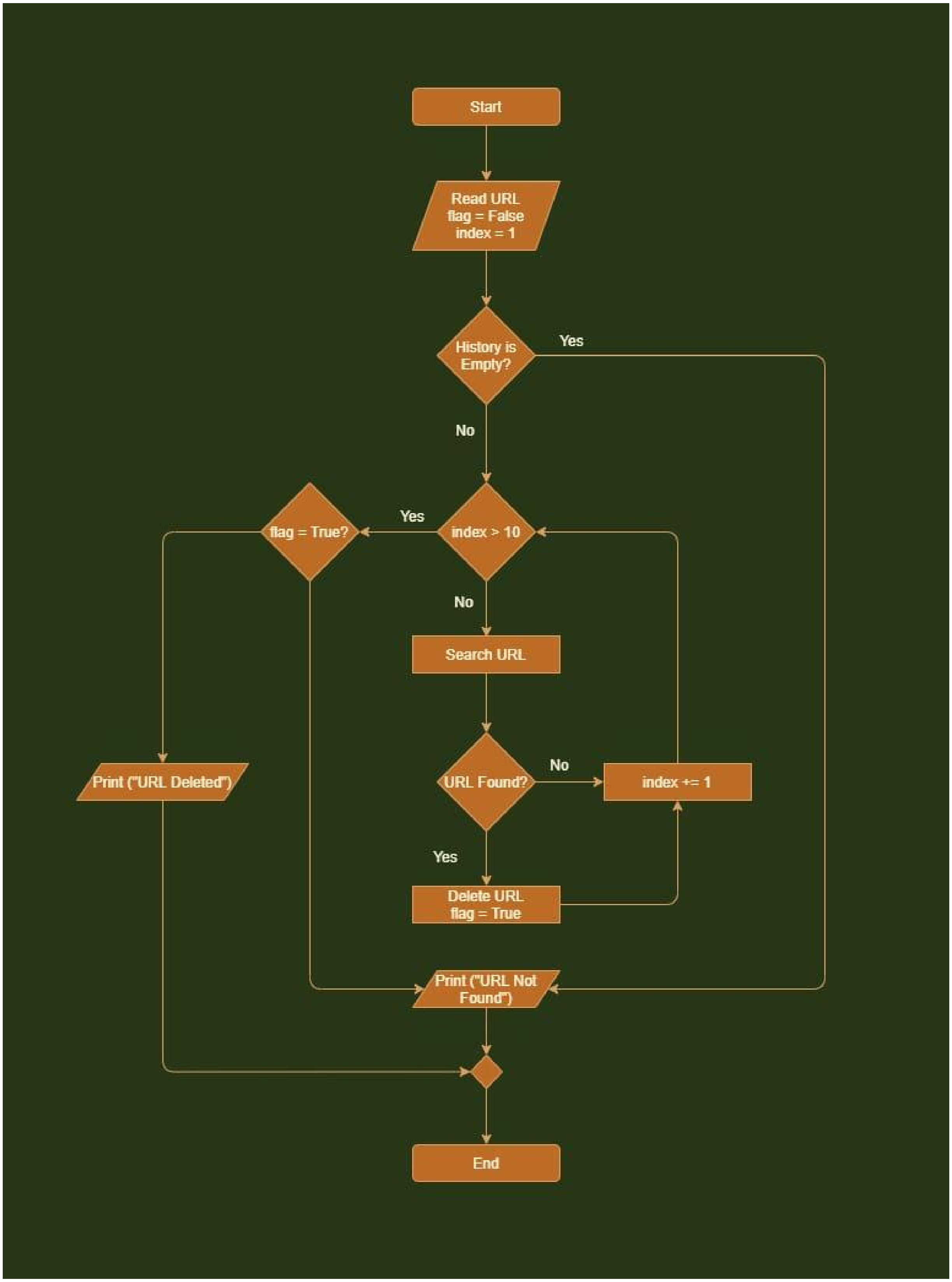


```
Main Menu:
----------

1. Add a Visited URL
2. Delete a Visited URL
3. View Your Browsing History
4. Track a Visited URL
5. Format Browsing History
6. History Backup
7. Restore History Backup
8. Total Tracked URLs


** Any other key to exit!!


Enter your choice: 
```

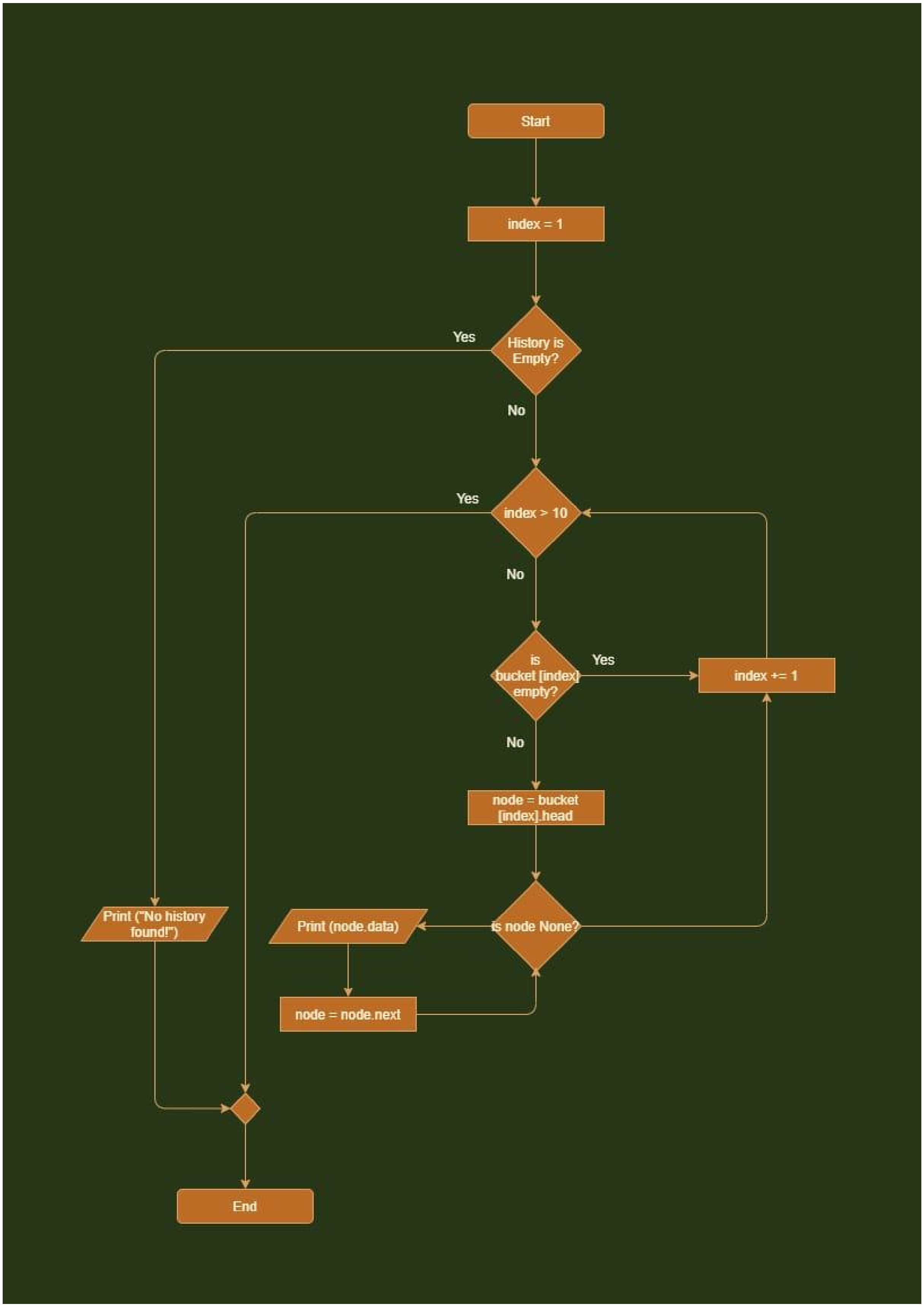

Start
First Run?
NodeCnt = 0
Read ClientReq
ClientReq = Deletion
ClientReq = Insertion
ClientReq = Searching
Deletion Algorithm
Search Algorithm
Insertion Algorithm
NodeCnt -= 1
NodeCnt += 1
Print Confirmation
Print SearchResult
Print Confirmation
NodeCount = 100
ClearAll Algorithm
NodeCount < 100
NodeCnt = 0
Stop

Few Important Flowcharts

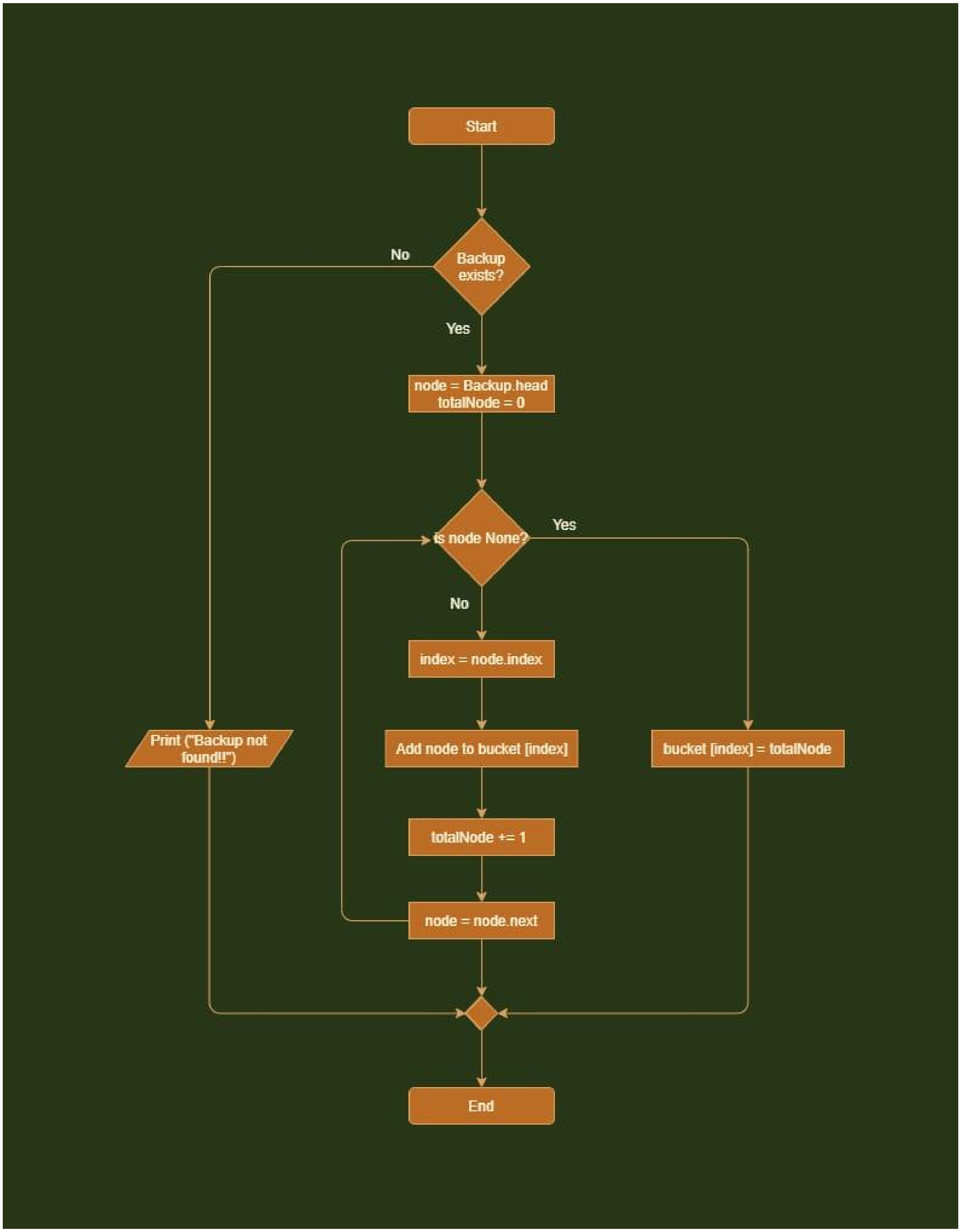
Deletion



View History

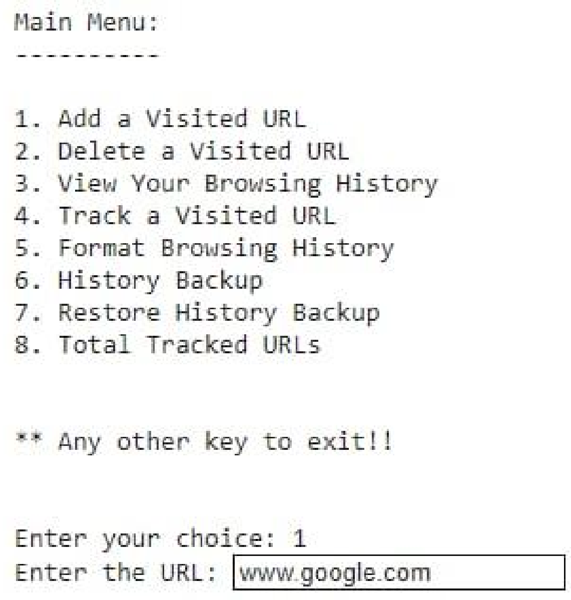


Restore History



Algorithms and Outputs of Different Methods / Functions

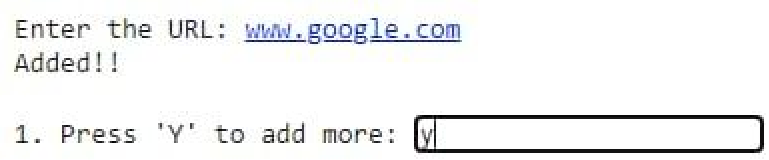
1. Insertion -



```
Main Menu:
----------

1. Add a Visited URL
2. Delete a Visited URL
3. View Your Browsing History
4. Track a Visited URL
5. Format Browsing History
6. History Backup
7. Restore History Backup
8. Total Tracked URLs


** Any other key to exit!!


Enter your choice: 1
Enter the URL: www.google.com
```

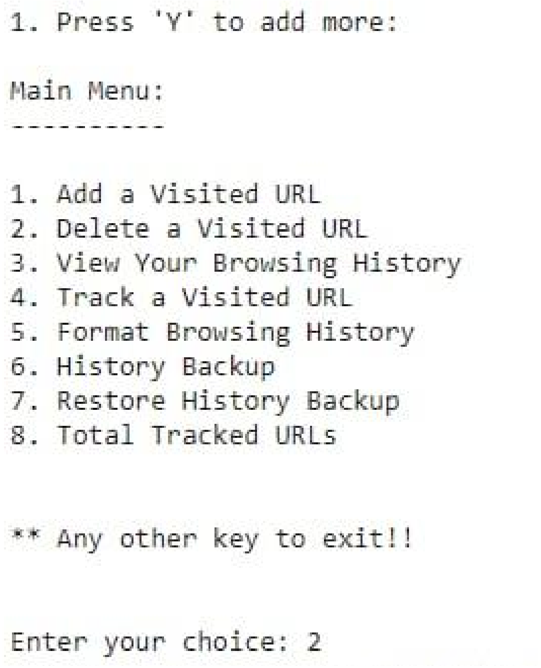
Algorithm

```
Start
model Bucket( )
n Node( )
model.insert(n)

ind n.getBucketIndex( )
temp self.history[index]
self.history[index] n
n.next temp

return
End
```



```
Enter the URL: www.google.com
Added!!

1. Press 'Y' to add more: y
```

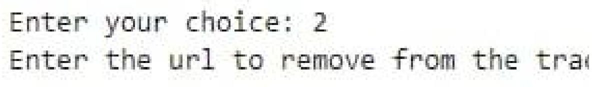
2. Deletion -



```
1. Press 'Y' to add more:

Main Menu:
----------

1. Add a Visited URL
2. Delete a Visited URL
3. View Your Browsing History
4. Track a Visited URL
5. Format Browsing History
6. History Backup
7. Restore History Backup
8. Total Tracked URLs


** Any other key to exit!!


Enter your choice: 2
```

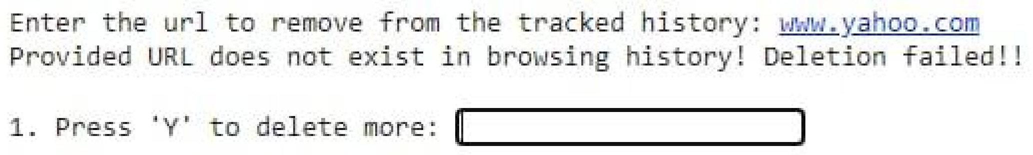
Algorithm

```
Start
URL input url
Flag False
If history is not Empty:
        For index from 1 to 9:
                Search URL in bucket[index]
                If URL is found:
                        Delete URL
                        Flag True
                        break

If flag is True:
        Print “Deleted”
else:
        print “URL Not Found”

End
```



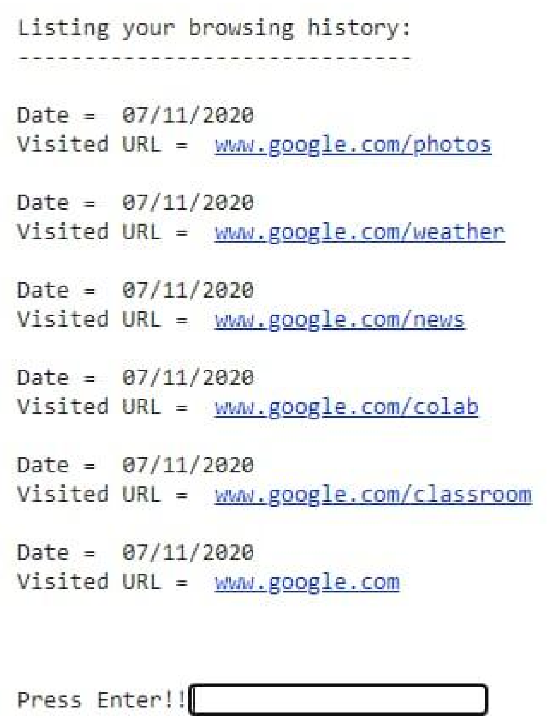
```
Enter your choice: 2
Enter the url to remove from the tra
```

In case, if user enters any invalid URL:



```
Enter the url to remove from the tracked history: www.yahoo.com
Provided URL does not exist in browsing history! Deletion failed!!

1. Press 'Y' to delete more:
```

3. Viewing the History -



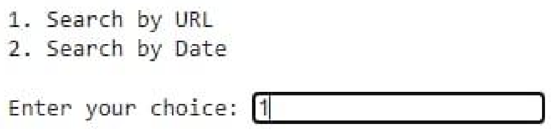
```
Listing your browsing history:
------------------------------

Date =  07/11/2020
Visited URL =  www.google.com/photos

Date =  07/11/2020
Visited URL =  www.google.com/weather

Date =  07/11/2020
Visited URL =  www.google.com/news

Date =  07/11/2020
Visited URL =  www.google.com/colab

Date =  07/11/2020
Visited URL =  www.google.com/classroom

Date =  07/11/2020
Visited URL =  www.google.com


Press Enter!!
```

Algorithm

```
Start
If history is not Empty:
    for index from 1 to 9:
        if bucket[index] is not empty:
            node bucket[index].head
            while node is node None:
                print node.data
                node node.next
        Else:
            print “No history found”
End
```

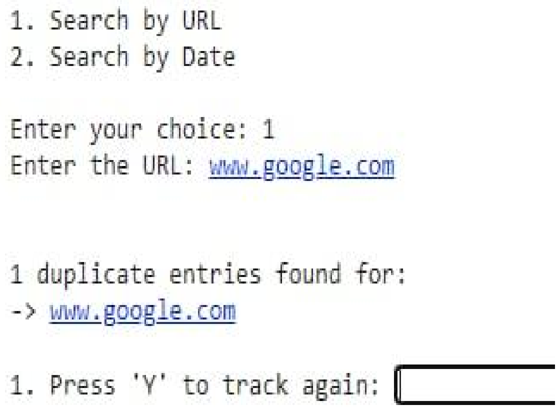
4. Searching -

To search the duplicate URLs



```
1. Search by URL
2. Search by Date

Enter your choice: 1
```

Duplicate URLs

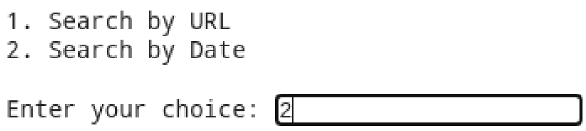


```
1. Search by URL
2. Search by Date

Enter your choice: 1
Enter the URL: www.google.com


1 duplicate entries found for:
-> www.google.com

1. Press 'Y' to track again:
```

Algorithm

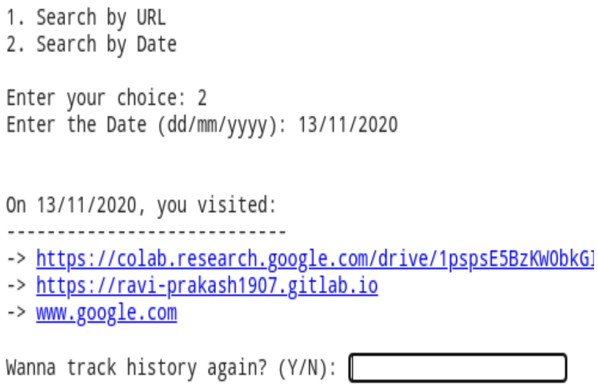
```
Start
Define Function search_duplicate_URL
(search_URL)
print(“Searching Node”)
TEMP1 head
duplicate_list [ ]
While TEMP1 not equal to None

If TEMP.URL == search_URL then
        Print(“COUNT”)
        Print (“TEMP1.URL”)
        duplicate_list += TEMP1.URL
EndIf
COUNT COUNT + 1
TEMP TEMP.next
End While loop
If duplicate_list is not Empty, then
        Print(duplicate_list)
        Return True
End If

Return False
End
```

To search URLs by Timestamp



```
1. Search by URL
2. Search by Date

Enter your choice: 2
```

Search by Timestamp



```
1. Search by URL
2. Search by Date

Enter your choice: 2
Enter the Date (dd/mm/yyyy): 13/11/2020


On 13/11/2020, you visited:
----------------------------
-> https://colab.research.google.com/drive/1pspsE5BzKW0bkG
-> https://ravi-prakash1907.gitlab.io
-> www.google.com

Wanna track history again? (Y/N):
```

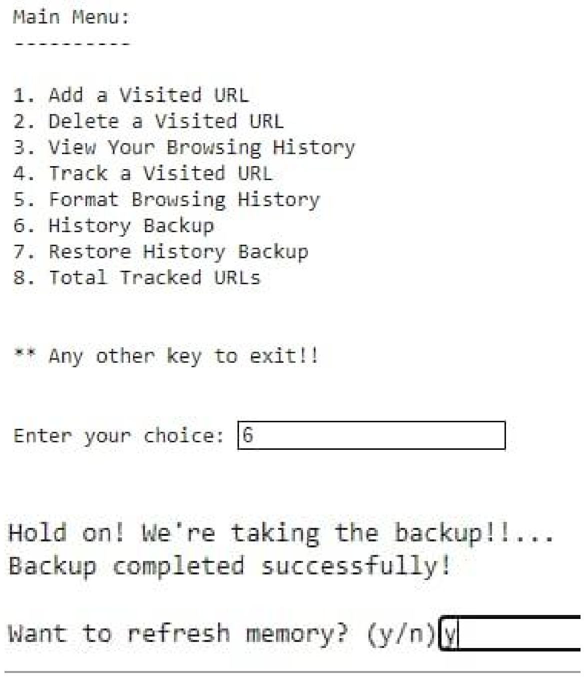
Algorithm

```
Start
Define function ‘search_by_timestamp’
print(“….Searching Node….”)
TMP1 head
TMP2 tail
Input_timestamp scan( )
searchTimestamp, search_bucketindex Node.
setTimeData(Node,
datetime.strptime(inputTimestamp,
“%d/%m/%y %H:%M:%S :”))
exact_url ‘ ’
list1 [ ]
While TMP1 ! = None

  If TMP1.bktIndx == searchBktIndx then
    if TMP1.timestamp==searchTimestamp then
        Exact_url TEMP1.URL
    Endif
  Else
    pass
  Endif
  TEMP1 TEMP1.next
End While

Return exact_url, list1
End
```

5. Backup -

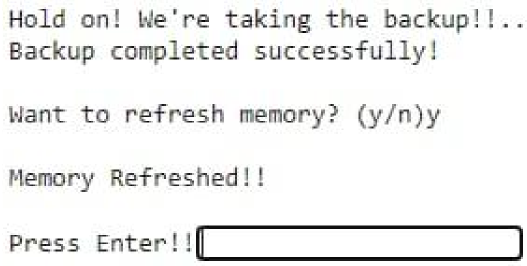
Possible ways to backup:

1 --->



```
Main Menu:
----------

1. Add a Visited URL
2. Delete a Visited URL
3. View Your Browsing History
4. Track a Visited URL
5. Format Browsing History
6. History Backup
7. Restore History Backup
8. Total Tracked URLs


** Any other key to exit!!


Enter your choice: 6


Hold on! We're taking the backup!!...
Backup completed successfully!

Want to refresh memory? (y/n)y
```

2 --->



```
Hold on! We're taking the backup!!...
Backup completed successfully!

Want to refresh memory? (y/n)y

Memory Refreshed!!

Press Enter!!
```

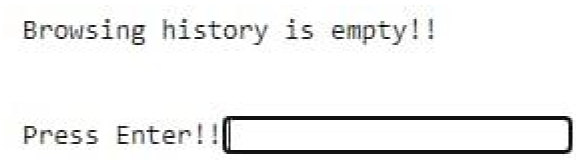
Algorithm

```
Start
TEMP1 head
duplicate_list [ ]
While TEMP1 not equal to None
If TEMP.URL <- search_URL then
        Print("COUNT")
        Print ("TEMP1.URL")
        duplicate_list += TEMP1.URL
EndIf
COUNT COUNT + 1
TEMP TEMP.next
End While loop
If duplicate_list is not Empty, then
        Print(duplicate_list)
        Return True
End If
Return False
End
```

Tackling Miscellaneous Situations

6. Output, in case, when tracked history is empty, on choosing -

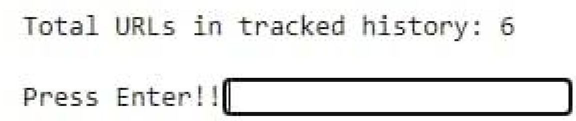
- Deletion
- Searching/Tracking
- Viewing History
- A total number of Tracked URLs
- Backup



```
Browsing history is empty!!

Press Enter!!
```

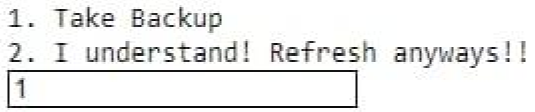
7. Total Tracked URLs -

Returns the value of the universal counter of the main bucket holder hybrid array which is stored on the initial index:



```
Total URLs in tracked history: 6

Press Enter!!
```

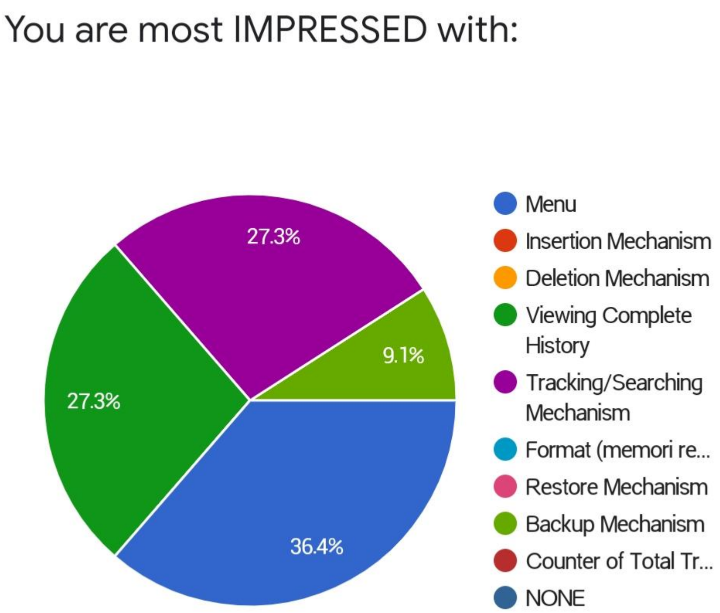
8. Formatting/Refreshing the Memory -

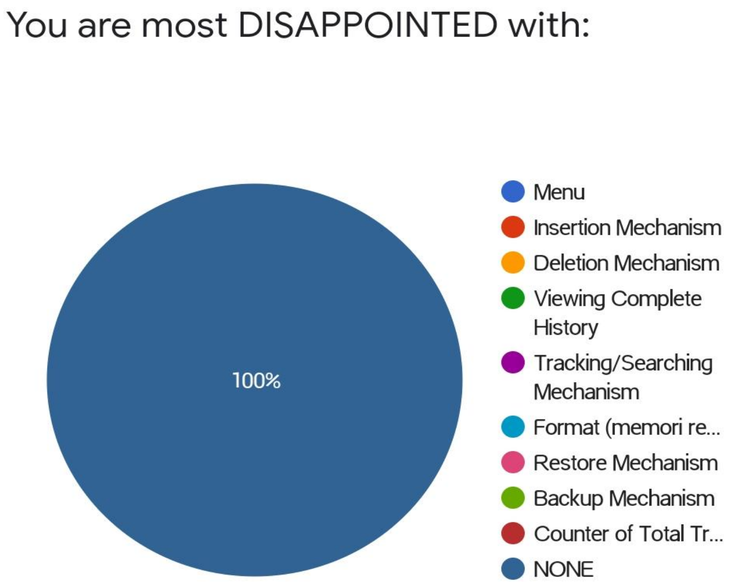


```
1. Take Backup
2. I understand! Refresh anyways!!
1
```

Endnote

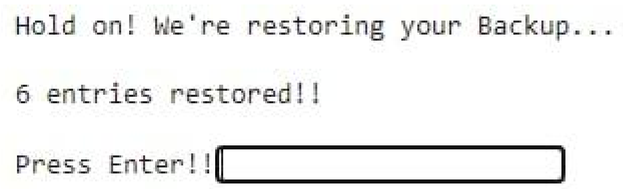
A feedback was conducted after implementing the final version of the solution that resulted as follows:


You are most IMPRESSED with:


You are most DISAPPOINTED with:

Our **Bucket-Storage Mechanism** is centered around a really very advanced level self-designed algorithm. The solution is developed after taking all the different possible aspects and situations into consideration. It not only satisfies the requirements but also provides other useful features like Restoring the Backup.

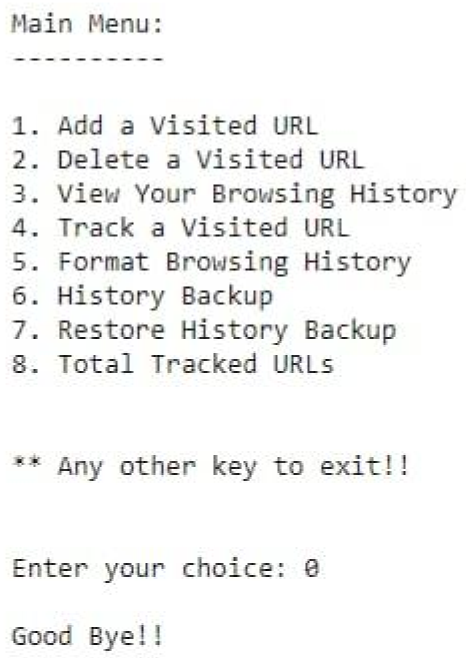
Restoring the Backed-Up history -



```
Hold on! We're restoring your Backup...

6 entries restored!!

Press Enter!!
```

Having a fantastic usage of the Object Oriented Programming concepts including *modularity* and *data-abstraction* makes this approach stand out from other commonly used methods. It also deals with all the possible adverse situations and the solution has the capabilities for error handling, as well.

The program greets the user when they exit the tracking system in the following manner -



```
Main Menu:
----------

1. Add a Visited URL
2. Delete a Visited URL
3. View Your Browsing History
4. Track a Visited URL
5. Format Browsing History
6. History Backup
7. Restore History Backup
8. Total Tracked URLs


** Any other key to exit!!


Enter your choice: 0

Good Bye!!
```

The running instance of the given solution is ready at the following link:

https://github.com/ravi-prakash1907/Data-Structures-and-Algo/blob/main/Submissions/Group%20Projects/Project1/A%20Web%20History%20Tracker.ipynb

Happy Testing!!!