DSA CASE STUDY

DATA STRUCTURE - Bloom filter

Team Members:

- **CB.SC.U4CSE24709 (Data loading, persistence, traversal, and output file generation)**
- **CB.SC.U4CSE24716 (Dynamic updates (add/delete), reinitialization, and structure organization)**
- **CB.SC.U4CSE24718(Insertion, lookup, hash index visualization)**
- **CB.SC.U4CSE24739 (Initialization, hashing, clearing Bloom filter, URL normalization)**

INTRODUCTION –

WHAT IS BLOOM FILTER?

A bloom filter is a probabilistic data structure which is extremely space efficient. It is based on hashing. This data structure is mainly used to look up and tell whether a particular element is present in the set or not. The elements or items are not directly added to the set instead a hash of elements is added to the set.

When we are searching for an item or element in the bloom filter there is a little probability of false positives (i.e. sometimes it will tell that the item is present even though it is not present in the set) but it will never give false negatives (i.e. if it says the item or element is not present in the set then it is not present in the set for sure).

WHY IT WAS INVENTED?

Bloom filter is invented to solve the problem of set membership queries using a probabilistic data structure which is space efficient and time efficient especially while dealing with large databases.

REAL WORL APPLICATIONS OF BLOOM FILTER

There are several real-world applications of bloom filter. Bloom filters are widely used in large scale databases where fast membership checking and low memory usage are critical.

1.EMAIL AND USERNAME AVAILABILITY

When a user tries to sign up in google or some other platform it looks up the database to ensure whether the username or email is already present (which is a slow process) so to increase the process speed it will maintain a bloom filter and look up the bloom filter first to ensure the email or username availability.

2.WEB CRAWLER / SEARCH ENGJNE

Web crawlers use search engine to determine whether the URL is already visited or not.

3.DATABASES AND BIG DATA SYSTEMS

Used to check whether the key is present in the database table or disk file before performing an expensive lookup.

4.SPAM FILTERING

Bloom filter maintains a list of known spammer addresses and domains.

5.NETWORK AND SECURITY SYSTEMS

Bloom filter stores a list of malicious IPs, domains or URLs.

BASIC OPERATIONS OF BLOOM FILTER

There are mainly two basic operations of bloom filter. They are

1.INSERTION

A bloom filter begins with a bit array of predefined size m. It also has k independent hash functions. When we insert an element, it is passed through every hash function which generates an index between 0 to m-1. All the bits at the k indices generated by hash functions are set to 1.

2.MEMBERSHIP TEST (CHECKING FOR AN ELEMENT)

To lookup for an element in bloom filter we pass the element through the same hash functions and see every bit corresponding is1 if any bit is 0 then the element is not present in the bloom filter for sure but if all the bits are 1 then there is a possibility that the element is present already (not sure because of false positives).

Note: Deletion is not possible in bloom filter as changing a bit can also affect other elements. To support deletion, we use counting bloom filter in which every bit is replaced by a counter.

TIME COMPLEXITY

- **INSERTION:**
  - Compute k hash functions
  - Set k bits in the array
  - Complexity: O(k)
- **QUERY / MEMBERSHIP CHECK:**
  - Compute k hash functions
  - Check k bits in the array
  - Complexity: O(k)

NOTE:

- K is usually small (1-3)
- So, operations are effectively constant time (O (1))

WHY O(k) DOMINATES

- Hash computation and bit manipulation are the main steps; other operations are negligible.

SPACE COMPLEXITY

- Uses a bit array of size m → O(m) bits
- Does not store the actual elements → very memory-efficient
- For n elements and desired false positive probability p, the optimal size of the array is:

$$m = -\frac{n \ln p}{(\ln 2)^2}$$

- Optimal number of hash functions:

$$k = \frac{m}{n} \ln 2$$

So, space depends on number of elements and desired accuracy, not on the size of elements themselves.

ADTs USED IN BLOOM FILTER

BIT ARRAY (OR BIT VECTOR)

➤ Definition

A Bit Array is a fixed-size array of bits (0s and 1s).

Each bit represents whether a certain “position” has been marked by any element inserted.

➤ Purpose in Bloom Filter

It acts as the main storage of the Bloom Filter — the entire filter is built upon it.

➤ Operations

Operation	Description
SetBit(i)	Sets the bit at position i to 1 (marking that some element maps here).
getBit(i)	Returns the current value (0 or 1) at position i.
clear()	Resets all bits to 0 (optional).

➤ Example

Bit Array (size = 10):

[0, 1, 0, 0, 1, 0, 0, 1, 0, 0]

If setBit (1) and setBit (4) are called, positions 1 and 4 become 1.

HASH FUNCTIONS

➤ Definition

A hash function maps an input (like a string or number) to an integer index in the bit array.

In a Bloom Filter, multiple hash functions are used.

➤ Purpose in Bloom Filter

Each hash function determines which bit(s) in the array to set or check for a particular element.

➤ Operations

Operation Description

hash (item, i) Returns an integer index in the bit array for the ith hash function.

➤ Example

If the bit array length = 10 and item = "apple",

hash1("apple") = 2

hash2("apple") = 5

hash3("apple") = 8

Then bits 2, 5, and 8 are set to 1.

SET CONCEPTUAL ADTs

➤ **Definition**

A Set ADT is a collection of unique elements that supports add, remove, and contains operations.

➤ **Purpose in Bloom Filter**

The Bloom Filter mimics the behavior of a Set — it allows checking whether an element may belong to the set.

However, the elements themselves are not actually stored.

➤ **Operations**

Operation	Description
add(item)	Adds an element to the Bloom Filter by hashing it and setting the corresponding bits.
contains(item)	Checks if an element may be in the set by verifying that all relevant bits are 1.

BOOLEAN

➤ Definition

A Boolean is a primitive data type with only two possible values: true or false.

➤ Purpose in Bloom Filter

Used to return results from membership checks.

➤ **Operations**

Operation	Description
true	Indicates the element may be present (all bits checked were 1).
false	Indicates the element is definitely not present (at least one bit was 0).

WORKING OF BLOOM FILTER:

It consists of two main parts — a bit array and multiple hash functions.

- The bit array is initialized with all bits set to zero.
- Each hash function independently maps an input element to a specific position in the bit array.
- When inserting an element, the element is passed through all the hash functions.
- Each hash function generates an index, and the bits at those positions in the bit array are set to one.
- This process is repeated for every element added to the filter, causing more bits to turn from zero to one.
- To check if an element exists, it is hashed again using the same hash functions.
- The Bloom Filter checks whether all the bits at the corresponding hash positions are set to one.
- If all bits are one, the element is considered “possibly present” in the set.
- If any one of the bits is zero, the element is “definitely not present.”
- False positives can occur when multiple elements set the same bits to one.
- Bloom Filters never produce false negatives, meaning if it says “not present,” the result is always correct.

- The probability of false positives depends on the size of the bit array, the number of hash functions, and how many elements are inserted.
- A larger bit array and an optimal number of hash functions can reduce false positives.
- Bloom Filters are commonly used in applications requiring fast membership checks with limited memory.
- They are used in web caching to check if a URL has been visited before.
- They are also used in databases to check if data might exist before accessing the disk.
- In network security, Bloom Filters help identify potentially malicious URLs or IP addresses efficiently.
- One limitation of Bloom Filters is that deletion is difficult since clearing a bit may remove information about other elements.
- Another limitation is that they do not store the actual elements, only the probability of their membership.
- Despite these limitations, Bloom Filters are highly useful in large-scale systems needing millions of fast lookups.

CODE

```
import tkinter as tk

from tkinter import ttk, messagebox, filedialog

import hashlib

import time

import json

import os

import csv

import re
```

```
SAFE_URLS_FILE = "safe_urls.json"

# ---------------- BLOOM FILTER CLASS ----------------
class BloomFilter:
    def __init__(self, size=1000, hash_count=5):
        self.size = size
        self.hash_count = hash_count
        self.bit_array = [0] * size
        self.safe_urls = set()

    def add(self, item):
        item = item.lower()
        self.safe_urls.add(item)
        for i in range(self.hash_count):
            index = self.hash_item(item, i)
            self.bit_array[index] = 1

    def check(self, item):
        item = item.lower()
        for i in range(self.hash_count):
            index = self.hash_item(item, i)
            if self.bit_array[index] == 0:
                return False
        return True
```

```
    def hash_item(self, item, i):
        h = hashlib.sha256()
        h.update((item + str(i)).encode())
        return int(h.hexdigest(), 16) % self.size

    def clear_filter(self):
        self.bit_array = [0] * self.size
        self.safe_urls.clear()

# ---------------- APPLICATION CLASS ----------------
class BloomFilterApp:
    def __init__(self, root):
        self.root = root
        self.root.title("Bloom Filter URL Safety Checker")
        self.root.geometry("850x750")
        self.root.config(bg="#F8FAFC")

        self.bf = BloomFilter()
        self.load_safe_urls()
        self.history = []

        self.url_pattern = re.compile(
            r'^(https?://)'
```

```
        r'(([A-Za-z0-9-]+\.)+[A-Za-z]{2,})'
        r'(/[A-Za-z0-9\-._~:/?#[\]@!$&\'()*+,;=%]*)?$'
    )

    self.create_gui()

def normalize_url(self, url):
    url = url.strip().lower()
    if not url.startswith("http://") and not url.startswith("https://"):
        url = "https://" + url
    return url

def load_safe_urls(self):
    if os.path.exists(SAFE_URLS_FILE):
        with open(SAFE_URLS_FILE, "r") as f:
            urls = json.load(f)
            self.default_safe_urls = [self.normalize_url(u) for u in urls]
    else:
        self.default_safe_urls = [
            "https://google.com",
            "https://www.wikipedia.org",
            "https://www.python.org",
            "https://mail.google.com",
            "https://stackoverflow.com"
```

```
            ]
            self.save_safe_urls()
        for url in self.default_safe_urls:
            self.bf.add(url)

    def save_safe_urls(self):
        with open(SAFE_URLS_FILE, "w") as f:
            json.dump(sorted(list(self.bf.safe_urls)), f)

    def create_gui(self):
        title = tk.Label(self.root, text="Bloom Filter URL Safety Checker",
                         font=("Helvetica", 16, "bold"), bg="#F8FAFC",
fg="#0F172A")
        title.pack(pady=10)

        frame_input = tk.Frame(self.root, bg="#F8FAFC")
        frame_input.pack(pady=10)
        tk.Label(frame_input, text="Enter URL:", font=("Helvetica", 12),
bg="#F8FAFC").grid(row=0, column=0, padx=5)
        self.url_entry = tk.Entry(frame_input, width=50, font=("Helvetica", 12))
        self.url_entry.grid(row=0, column=1, padx=5)
        tk.Button(frame_input, text="Check", command=self.check_url,
                  bg="#0EA5E9", fg="white", font=("Helvetica", 11, "bold"),
width=10).grid(row=0, column=2, padx=5)
```

```
        self.result_label = tk.Label(self.root, text="Result: Waiting for input...",
                                     font=("Helvetica", 12, "italic"), bg="#F8FAFC",
fg="#334155")
        self.result_label.pack(pady=10)


        self.explain_var = tk.BooleanVar()
        tk.Checkbutton(self.root, text="Explain Working",
variable=self.explain_var,
                       bg="#F8FAFC", font=("Helvetica", 10)).pack()


        columns = ("URL", "Status", "Time")
        self.tree = ttk.Treeview(self.root, columns=columns, show="headings",
height=8)
        for col in columns:
            self.tree.heading(col, text=col)
            self.tree.column(col, anchor="center", width=250)
        self.tree.pack(pady=10)


        btn_frame = tk.Frame(self.root, bg="#F8FAFC")
        btn_frame.pack(pady=10)
        tk.Button(btn_frame, text="Add Safe URL",
command=self.add_safe_url,
                  bg="#22C55E", fg="white", font=("Helvetica", 10, "bold"),
width=15).grid(row=0, column=0, padx=5)
        tk.Button(btn_frame, text="Export Report",
command=self.export_report,
```

```
            bg="#F97316", fg="white", font=("Helvetica", 10, "bold"),
width=15).grid(row=0, column=1, padx=5)

        tk.Button(btn_frame, text="Exit", command=self.root.quit,

            bg="#DC2626", fg="white", font=("Helvetica", 10, "bold"),
width=10).grid(row=0, column=2, padx=5)


        tk.Label(self.root, text="Safe URLs:", font=("Helvetica", 12, "bold"),
bg="#F8FAFC").pack(pady=(10, 0))

        self.safe_listbox = tk.Listbox(self.root, width=70, height=6)

        self.safe_listbox.pack(pady=5)

        self.update_safe_listbox()


        url_btn_frame = tk.Frame(self.root, bg="#F8FAFC")

        url_btn_frame.pack(pady=(0, 10))

        tk.Button(url_btn_frame, text="Delete Selected",
command=self.delete_selected_url,

            bg="#F87171", fg="white", font=("Helvetica", 10, "bold"),
width=15).grid(row=0, column=0, padx=5)

        tk.Button(url_btn_frame, text="Clear All", command=self.clear_all_urls,

            bg="#EF4444", fg="white", font=("Helvetica", 10, "bold"),
width=15).grid(row=0, column=1, padx=5)


        self.explain_text = tk.Text(self.root, height=8, width=95, wrap="word",
bg="#F1F5F9", font=("Consolas", 10))

        self.explain_text.pack(pady=10)

        self.explain_text.insert(tk.END, "Explanation will appear here when
'Explain Working' is enabled.\n")
```

```
        self.explain_text.config(state=tk.DISABLED)

    # ---------------- CORE FUNCTIONS ----------------
    def check_url(self):
        url = self.normalize_url(self.url_entry.get())
        if not url:
            messagebox.showwarning("Input Error", "Please enter a URL.")
            return
        if not self.url_pattern.match(url):
            messagebox.showerror("Invalid URL", "Please enter a proper URL
starting with http:// or https://")
            return

        start = time.time()
        is_safe = self.bf.check(url)
        end = time.time()
        check_time = f"{(end - start):.4f}s"

        if is_safe:
            color, status = "#16A34A", "Safe"
            text = f"URL '{url}' is probably SAFE."
        else:
            color, status = "#DC2626", "Unsafe"
            text = f"URL '{url}' is NOT in safe list."
```

```
        self.result_label.config(text=text, fg=color)
        timestamp = time.strftime("%I:%M:%S %p")
        self.history.append((url, status, timestamp))
        self.tree.insert("", tk.END, values=(url, status, timestamp))

        if self.explain_var.get():
            self.show_explanation(url, is_safe, check_time)
        else:
            self.clear_explanation()

    def show_explanation(self, url, is_safe, check_time):
        self.explain_text.config(state=tk.NORMAL)
        self.explain_text.delete(1.0, tk.END)
        self.explain_text.insert(tk.END, f"Checking URL: {url}\n\n")
        for i in range(self.bf.hash_count):
            h = self.bf.hash_item(url.lower(), i)
            self.explain_text.insert(tk.END, f"Hash {i+1}: Index {h}\n")
        self.explain_text.insert(tk.END, f"\nResult: {'Safe' if is_safe else
'Unsafe'}")
        self.explain_text.insert(tk.END, f"\nCheck Time: {check_time}\n")
        self.explain_text.config(state=tk.DISABLED)

    def clear_explanation(self):
        self.explain_text.config(state=tk.NORMAL)
        self.explain_text.delete(1.0, tk.END)
```

```
        self.explain_text.insert(tk.END, "Explanation disabled.\n")
        self.explain_text.config(state=tk.DISABLED)

    def add_safe_url(self):
        new_url = self.normalize_url(self.url_entry.get())
        if not new_url:
            messagebox.showwarning("Input Error", "Enter a URL to add to
safe list.")
            return
        if not self.url_pattern.match(new_url):
            messagebox.showerror("Invalid URL", "Please enter a proper URL
starting with http:// or https://")
            return
        if new_url in self.bf.safe_urls:
            messagebox.showinfo("Info", f"'{new_url}' is already in the safe
list.")
            return

        self.bf.add(new_url)
        self.save_safe_urls()
        self.update_safe_listbox()
        messagebox.showinfo("Success", f"Added '{new_url}' to safe list!")

    def delete_selected_url(self):
        selected = self.safe_listbox.curselection()
```

```
        if not selected:
            messagebox.showwarning("No Selection", "Please select a URL to delete.")
            return
        url = self.safe_listbox.get(selected[0])
        confirm = messagebox.askyesno("Confirm Delete", f"Are you sure you want to delete '{url}' from safe URLs?")
        if confirm:
            self.bf.safe_urls.discard(url)
            self.save_safe_urls()
            self.update_safe_listbox()

    def clear_all_urls(self):
        confirm = messagebox.askyesno("Confirm Clear", "Are you sure you want to delete ALL safe URLs?")
        if confirm:
            self.bf.clear_filter()
            self.save_safe_urls()
            self.update_safe_listbox()

    def update_safe_listbox(self):
        self.safe_listbox.delete(0, tk.END)
        for url in sorted(self.bf.safe_urls):
            self.safe_listbox.insert(tk.END, url)
```

```
    def export_report(self):
        file_path = filedialog.asksaveasfilename(defaultextension=".csv",
                                                 filetypes=[("CSV Files", "*.csv")])
        if not file_path:
            return

        with open(file_path, "w", newline="", encoding="utf-8") as f:
            writer = csv.writer(f)
            writer.writerow(["URL", "Status", "Time"])
            for url, status, timestamp in self.history:
                writer.writerow([url, status, timestamp])
            history_urls = set(url for url, _, _ in self.history)
            for url in sorted(self.bf.safe_urls):
                if url not in history_urls:
                    writer.writerow([url, "Safe", "Not Checked"])
        messagebox.showinfo("Export Successful", f"CSV report saved to
{file_path}")


# ---------------- MAIN ----------------
if __name__ == "__main__":
    root = tk.Tk()
    app = BloomFilterApp(root)
    root.mainloop()
```

FUNCTIONS - TIME COMPLEXITY, SPACE COMPLEXITY

Function / Method	Time Complexity	Space Complexity	Description
`__init__(size=1000, hash_count=5)`	O(n)	Initialize bit array and safe URLs set	`__init__(size=1000, hash_count=5)`
`add(item)`	O(k)	Add URL: update bit array + insert in set	`add(item)`
`check(item)`	O(k)	Check URL: check bit array indices	`check(item)`
`hash_item(item, i)`	O(1)	Compute hash index	`hash_item(item, i)`
`clear_filter()`	O(n)	Clear Bloom Filter bit array and safe URLs set	`clear_filter()`
`__init__(root)`	O(n + m·k)	Initialize GUI, Bloom Filter, load safe URLs	`__init__(root)`
`normalize_url(url)`	O(1)	Clean/format URL	`normalize_url(url)`
`load_safe_urls()`	O(m·k)	Load safe URLs from JSON into Bloom Filter	`load_safe_urls()`
`save_safe_urls()`	O(n)	Save safe URLs from set to JSON	`save_safe_urls()`
`create_gui()`	O(1)	Build GUI interface	`create_gui()`
`check_url()`	O(k)	Check URL safety, update history & GUI	`check_url()`

`show_explanation(url, is_safe, check_time)`	O(k)	Show hash indices for explanation	`show_explanation(url, is_safe, check_time)`
`clear_explanation()`	O(1)	Clear explanation text	`clear_explanation()`
`add_safe_url()`	O(k)	Add URL to Bloom Filter and set	`add_safe_url()`
`delete_selected_url()`	O(1)	Remove URL from set + update GUI	`delete_selected_url()`
`clear_all_urls()`	O(n)	Clear Bloom Filter + set + update GUI	`clear_all_urls()`
`update_safe_listbox()`	O(n log n)	Sort safe URLs set and update Listbox	`update_safe_listbox()`
`export_report()`	O(h + n)	Export history + safe URLs to CSV	`export_report()`