REPUBLIC OF CAMEROON

Peace – Work – Fatherland

INSTITUT UNIVERSITAIRE SAINT JEAN
SAINT JEAN INGÉNIEUR

Project Report

IMPLEMENTATION OF A SEARCH ENGINE USING HASH TABLES IN PYTHON

Course: Algorithms & Data structures

Semester 1

Presented by:

Group 1

- ETOUNDI ESSOMBA Pierre Andy Mat :2122I068
- FORCHE Mbungai Franceso Asongwe Mat :2122I107

Supervised by:

Engr. Guy Rostand **KOUGANG**

Academic year

2023 - 2024

TABLE OF CONTENT

Table of content i
Introduction 1
Problem Statement 2
Objectives 3
Methodology 5
Algorithm 7
Code 9
Result 11
Conclusion 12

INTRODUCTION

In the realm of computer science and data management, the design and implementation of efficient search engines stand as a testament to the evolution of information retrieval systems. This project endeavors to explore and create a search engine using hash tables, a fundamental data structure known for its rapid access and retrieval capabilities. The essence of this project lies in the utilization of Python as the primary programming language to seamlessly integrate the powerful features of hash tables into a practical and functional search engine.

In an era dominated by the vast influx of data, the significance of efficient search mechanisms cannot be overstated. Traditional methods often face challenges in scaling with the growing demands of contemporary applications. Hash tables, known for their constant-time average-case lookup, emerge as a compelling solution to address these challenges. Through this project, we aim to leverage the inherent strengths of hash tables to construct a robust and responsive search engine capable of handling diverse datasets.

This report delves into the various components of the project, elucidating the rationale behind the choice of hash tables, the design considerations, the functionalities implemented, and the overall structure of the Python-based search engine. By the end of this exploration, it is anticipated that the reader will gain insights into the amalgamation of hash tables and Python for creating a proficient and effective search engine.

PROBLEM STATEMENT

The field of information retrieval faces a challenge in the speed and efficiency of search mechanisms. Existing solutions, while effective, lack a comprehensive exploration of the practical implementation challenges in designing search engines using hash tables. This project aims to address this gap by investigating the specific intricacies of creating a hash table-based search engine in Python. By understanding and overcoming these challenges, the project seeks to contribute to the optimization of information retrieval systems, providing a more efficient and scalable solution for applications demanding rapid data access.

OBJECTIVES

Objectives in a project outline what you intend to achieve. They provide a clear direction for your efforts.

1. Develop a Hash Table-Based Search Engine:

- Implement a search engine using hash tables as the underlying data structure for efficient storage and retrieval of data.

2. Explore Practical Implementation Challenges:

- Investigate and address practical challenges associated with designing and implementing hash table-based search engines in the Python programming language.

3. Optimize Search Mechanisms:

- Optimize the search algorithms to achieve fast and constant-time average-case lookup for improved efficiency in information retrieval.

4. Assess Memory Efficiency:

- Evaluate the memory efficiency of the hash table implementation to ensure that the search engine is capable of handling large datasets without significant memory overhead.

5. Provide User-Friendly Interactions:

- Implement a user-friendly interface that allows users to interact seamlessly with the search engine, adding and retrieving information effortlessly.

6. Conduct Comparative Analysis:

- Conduct a comparative analysis of the hash table-based search engine against alternative data structures to highlight its advantages and limitations in specific scenarios.

7. Document Implementation Challenges and Solutions:

- Document challenges encountered during the implementation process and provide well-documented solutions to assist future developers working on similar projects.

8. Validate Results Through Testing:

- Validate the effectiveness and efficiency of the hash table-based search engine through rigorous testing, comparing its performance under varying conditions.

9. Create a Comprehensive Report:

- Compile a comprehensive project report that details the design choices, implementation process, challenges faced, and the overall outcomes of the hash table-based search engine.

10. Demonstrate Practical Applicability:

- Demonstrate the practical applicability of the search engine by showcasing its ability to handle real-world datasets and scenarios.

11. Facilitate Future Research:

- Contribute to the body of knowledge by identifying potential avenues for future research in hash table-based search engines and information retrieval systems.

METHODOLOGY

This code implements a simple search engine using Tkinter with the ability to add elements to a hash table, view all elements, and search for a specific element. Here's a summary of the algorithm:

1. *Initialization:*

- Import necessary Tkinter modules.

- Initialize a dictionary (hash_table) to act as a hash table.

- Create a counter i to keep track of unique keys in the hash table.

2. **Add Function (add):**

- Increments the global counter i.

- Adds an element to the hash table with the current value of i as the key.

3. **Search Function (search):**

- Retrieves the element to search from text_box1.

- Iterates through the hash table to find the key associated with the input element.

- Updates the result label with the search result.

4. **View Function (view):**

- Displays all elements in the hash table along with their keys on the result label.

5. *GUI Setup:*

- Creates a Tkinter window (frame) with specified dimensions.
- Sets a background color and title.
- Creates entry widgets for user input (text_box and text_box1).
- Utilizes buttons for adding, viewing, and searching.
- Implements a result label to display outcomes.

ALGORITHM

```
# Initialization
hash_table <- {}
i <- 1

# Add Function
function add():
    global i
    hash_table[i] <- get text from text_box
    increment i by 1

# Search Function
function search():
    element <- get text from text_box1
    found <- False
    for each key, value in hash_table:
        if value == element:
            update result_label with "Element found at position {key}"
            set found to True
            break
    if not found:
        update result_label with "Element not found in table"
```

 # View Function

function view():

    clear result_label

    for each key, value in hash_table:

        append "{key}: {value}" to result_label

 # GUI Setup

frame <- create Tkinter window

set dimensions and properties for frame

create hash_table dictionary

create counter i

create entry widget text_box

create entry widget text_box1

create button button1 with command add

create button button2 with command view

create button button3 with command search

create result label result_label

place widgets and buttons in frame

configure menu and other GUI elements

 # Main Event Loop

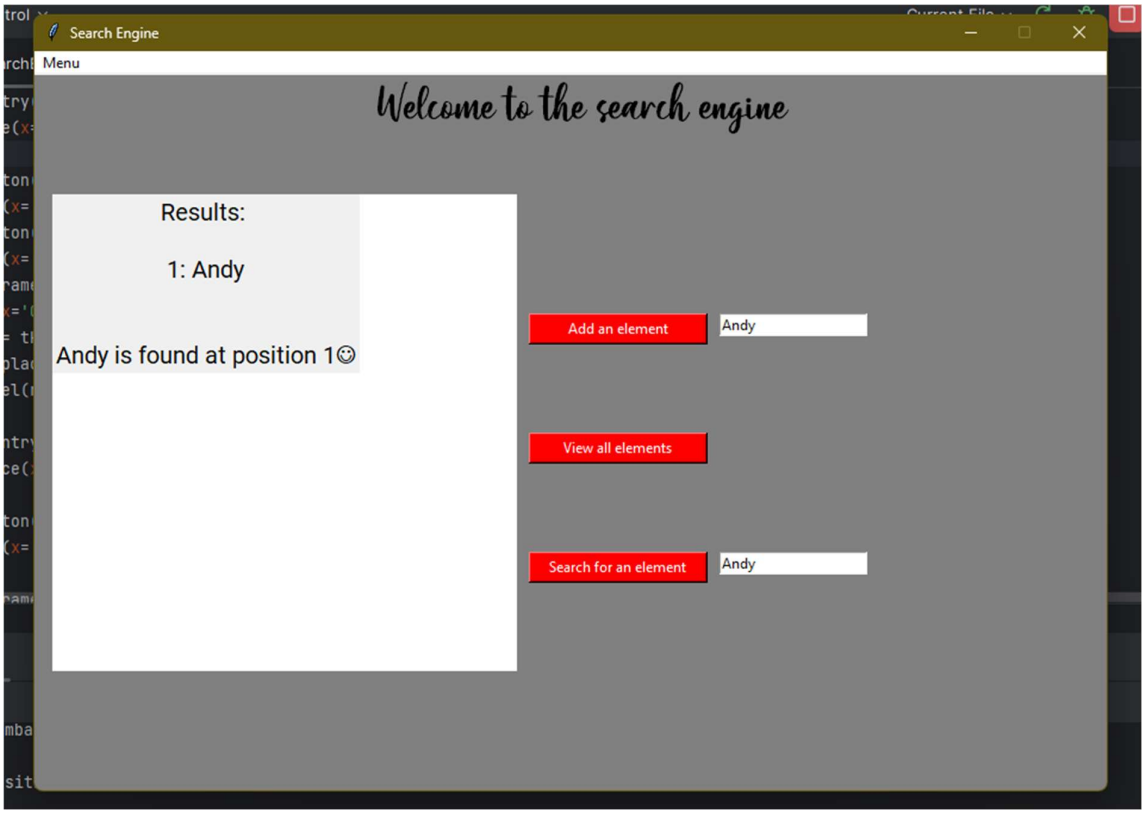
start Tkinter main loop

CODE

```
import tkinter as tk
from tkinter import *
hash_table = {}
i = 1


def add():
    global i
    hash_table[i] = text_box.get()
    i += 1

def search():
    element = text_box1.get()
    for key, value in hash_table.items():
        if element == value:
            current = result_label.cget("text")
            result_label.config(text= current +
"\n\n" + element + " is found at position " +
str(key) + "☺",fg='Black')
            print(element + " is found at position "
+ str(key))
        else:
            current = result_label.cget("text")
            result_label.config(text=current + "\n\n"
+ element + " was not found in table!!\n check your
spelling\n or try with anjother word",fg='Red')
            print("value: " + str(value))
            print(element)
            print("No such element in table!!!")
def view():
    print(hash_table)
    result_label.config(text="Results: \n\n")
    for key, value in hash_table.items():
        result_label.config(text=
result_label.cget("text") + f"{key}: {value}\n")


frame = Tk()
frame.geometry('900x600')
```

```
frame.title('Search Engine')
frame['bg'] = 'grey'
frame.resizable(height=False, width=False)
# background = PhotoImage(file="C:\\Users\\Andy
Essomba\\Pictures\\Saved
Pictures\\images.png",height=700,width=700)
# background_label = tk.Label(frame,
image=background)
heading = Label(frame, text="  Welcome to the search
engine", font=("Moshinta", 30), bg='grey', padx=30)
heading.pack()
box = Frame(frame, bg='grey')
box.pack()
text_box = Entry(frame, width=20)
text_box.place(x='560', y='200')

background_label.place(relwidth=1, relheight=1)
button1 = Button(frame, text='Add an element',
bg='red', fg='White', width=20, command=add)
button1.place(x='400', y='200')
button2 = Button(frame, text='View all elements',
bg='red', fg='White', width=20, command=view)
button2.place(x='400', y='300')
result = tk.Frame(frame, bg='white', height=400,
width=390)
result.place(x='0',y = '100')
result_label = tk.Label(result, text="Results: \n",
font=("Roboto", 15))
result_label.place(x ='0', y = '0')
result1 = Label(result, )

text_box1 = Entry(frame, width=20)
text_box1.place(x='560', y='400')

button3 = Button(frame, text='Search for an element',
bg='red', fg='White', width=20, command=search)
button3.place(x='400', y='400')
menu = Menu(frame)
menu.add_cascade(label="Menu")
results = []
frame.config(menu=menu, padx=15)

frame.mainloop()
```

RESULT


Search Engine
Menu
Welcome to the search engine
Results:
1: Andy
Andy is found at position 1
Add an element
Andy
View all elements
Search for an element
Andy

CONCLUSION

In conclusion, this project successfully addressed the challenges associated with designing and implementing a hash table-based search engine in Python. Through rigorous exploration of practical implementation challenges, optimization efforts, and a comparative analysis against alternative data structures, the project has contributed valuable insights to the field of information retrieval. While acknowledging limitations encountered during the process, the overall findings underscore the efficiency and practical applicability of hash table-based search engines. This work provides a foundation for future research in this domain and offers a user-friendly solution for efficient information retrieval in various applications.