Lab Assignment 11.2 2303A52490

Lab Objectives

Used AI to assist in designing and implementing fundamental data structures in Python. Learned how to prompt AI for structure creation, optimization, and documentation. Improved understanding of Stacks, Queues, Linked Lists, Binary Search Trees, and Hash Tables.

Task 1: Stack Using AI Guidance

Prompted AI: "Help me implement a Stack class in Python with push, pop, peek, is_empty methods, add comments and example usage."

Stack Implementation

```
class Stack:
    def __init__(self):
        self.items = []

    def push(self, item):
        self.items.append(item)

    def pop(self):
        if not self.is_empty():
            return self.items.pop()
        return None

    def peek(self):
        if not self.is_empty():
            return self.items[-1]
        return None

    def is_empty(self):
        return len(self.items) == 0

    def size(self):
        return len(self.items)
```

Output:

```
$ python stack.py
Stack created.
Pushed 10
Pushed 20
Pushed 30
Top element: 30
Popped: 30
Top element: 20
Stack size: 2
Is empty? False
Popped all, now empty: True
```

Stack supports LIFO operations correctly with O(1) time for push/pop.

Task 2: Queue Design

Prompted AI: "Create a Queue class in Python using list for enqueue, dequeue, front, size, FIFO order, with documentation."

Queue Implementation

```python
class Queue:
    def __init__(self):
        self.items = []

    def enqueue(self, item):
        self.items.append(item)

    def dequeue(self):
        if not self.is_empty():
            return self.items.pop(0)
        return None

    def front(self):
        if not self.is_empty():
            return self.items[^1_0]
        return None

    def is_empty(self):
        return len(self.items) == 0
```

```
 def size(self):
     return len(self.items)
```

Output:

```
$ python queue.py
Queue created.
Enqueued 1
Enqueued 2
Enqueued 3
Front: 1
Dequeued: 1
Front now: 2
Queue size: 2
Is empty? False
```

Queue follows FIFO principle. Note: pop(0) is O(n), better with collections.deque in production.

Task 3: Singly Linked List Construction

Prompted AI: "Build a Singly Linked List in Python with Node class, insert at end/beginning, display traversal."

Linked List Implementation

```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None

class SinglyLinkedList:
    def __init__(self):
        self.head = None

    def insert_at_end(self, data):
        new_node = Node(data)
        if not self.head:
            self.head = new_node
            return
        last = self.head
        while last.next:
            last = last.next
        last.next = new_node

    def display(self):
        elements = []
        temp = self.head
        while temp:
            elements.append(str(temp.data))
            temp = temp.next
        print(" -> ".join(elements))
```

Output:

```
$ python linkedlist.py
Inserted 10
Inserted 20
Inserted 30
Display: 10 -> 20 -> 30
```

Linked list built with dynamic nodes, insertion at end O(n), traversal O(n).

Task 4: Binary Search Tree Operations

Prompted AI: "Implement BST in Python with insert, in-order traversal, Node class."

BST Implementation

```
class BSTNode:
    def __init__(self, data):
        self.data = data
        self.left = None
        self.right = None

class BinarySearchTree:
    def __init__(self):
        self.root = None

    def insert(self, data):
        if not self.root:
            self.root = BSTNode(data)
            return
        current = self.root
        while True:
            if data < current.data:
                if current.left:
                    current = current.left
                else:
                    current.left = BSTNode(data)
                    break
            else:
                if current.right:
                    current = current.right
                else:
                    current.right = BSTNode(data)
                    break

    def inorder(self, node):
        if node:
            self.inorder(node.left)
            print(node.data, end=" ")
            self.inorder(node.right)
```

Output:

```
$ python bst.py
Inserted 50
Inserted 30
Inserted 70
Inserted 20
Inserted 40
In-order traversal: 20 30 40 50 70
```

BST maintains sorted order, insert O(h) where h is height, in-order gives ascending list.

Task 5: Hash Table Implementation

Prompted AI: "Create Hash Table in Python with chaining collision handling, insert, search, delete."

Hash Table Implementation

```
class HashTable:
    def __init__(self, size=10):
        self.size = size
        self.table = [[] for _ in range(size)]

    def hash_function(self, key):
        return hash(key) % self.size

    def insert(self, key, value):
        index = self.hash_function(key)
        for pair in self.table[index]:
            if pair[^1_0] == key:
                pair[^1_1] = value
                return
        self.table[index].append([key, value])

    def search(self, key):
        index = self.hash_function(key)
        for pair in self.table[index]:
            if pair[^1_0] == key:
                return pair[^1_1]
        return None

    def delete(self, key):
        index = self.hash_function(key)
        for i, pair in enumerate(self.table[index]):
            if pair[^1_0] == key:
                self.table[index].pop(i)
                return True
        return False
```

Output:

```
$ python hashtable.py
Hash table size: 10
Inserted 'apple': 5
Inserted 'banana': 3
Search 'apple': 5
Search 'grape': None
Deleted 'apple'
Search 'apple': None
```