Adamson University
College of Engineering
Computer Engineering Department

Data Structure and Algorithm

Laboratory Activity No.6

Linked List

Submitted by:
Manigbas, Jeus Miguel T. <Leader>
Barredo, Alwin P.
Bela, Lorenzo Miguel D.
Callorina, Robert Victor A.
Guzon, Kean Louiz I.

Instructor:
Engr. Maria Rizette H. Sayo

November 17, 2023

I. Objectives

Introduction

A linked list is an organization of a list where each item in the list is in a separate node. Linked lists look like the links in a chain. Each link is attached to the next link by a reference that points to the next link in the chain. When working with a linked list, each link in the chain is called a Node. Each node consists of two pieces of information, an item, which is the data associated with the node, and a link to the next node in the linked list, often called next.

This laboratory activity aims to implement the principles and techniques in:

- Writing algorithms using Linked list
- Writing a python program that will perform the common operations in a doubly linked list

II. Methods

- Given the set of element {a,b,c,d,e, f } stored in a list, show the final state of the list, assuming we use the move-to-front heuristic and access the elements according to the following sequence: (a,b,c,d,e, f ,a,c, f ,b,d,e).
- Write a Python program to create a doubly linked list, append some items, count the number of items, insert a new item in front, and print in reverse order

III. Results

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

class MoveToFrontLinkedList:
    def __init__(self, elements):
        self.head = None
        self.initialize_list(elements)

    def initialize_list(self, elements):
        for element in elements:
            self.append(element)

    def append(self, data):
        new_node = Node(data)
        new_node.next = self.head
        self.head = new_node

    def move_to_front(self, key):
        current = self.head
        prev = None

        while current and current.data != key:
            prev = current
            current = current.next

        if current:
            if prev:
                prev.next = current.next
                current.next = self.head
                self.head = current

    def display(self):
        current = self.head
        while current:
            print(current.data, end=" ")
            current = current.next
        print()

elements = ['a', 'b', 'c', 'd', 'e', 'f']

access_sequence = ['a', 'b', 'c', 'd', 'e', 'f', 'a', 'c', 'f', 'b', 'd', 'e']

linked_list = MoveToFrontLinkedList(elements)
for item in access_sequence:
    linked_list.move_to_front(item)

print("Final State of the Linked List:")
linked_list.display()
```

Figure 1.1 Code

```
Final State of the Linked List:
e d b f c a
```

Figure 1.2 Code results

This Python program establishes a linked list featuring a distinct behavior referred to as "Move-to-Front heuristic." The list manages its elements and adapts their arrangement according to how frequently they are accessed.

- **Node class**

- Represents a single node in a linked list.

- **MoveToFrontLinkedList**

- This represents the linked list with the Move-To-Front behavior.

- ‘Head’ attribute initially set to ‘none’.

- **‘__init__’ method in MoveToFrontLinkedList**

- Takes a list of elements as a parameter when initializing an instance of the class.

- Invokes the initialize_list function to construct the linked list using the given elements.

- **‘initialize_list’ method**

- Goes through each element and adds them to the linked list using the append function.

- **‘append’ method**

  - Places a new node containing the specified data at the start of the linked list, following the Move-to-Front heuristic.

- **‘move_to_front’ method**

- To change the position of a node in the linked list based on a specified key. The node with this key is moved in front of the list.

- The method iterates through the linked list to find the node with the specified key by examining each node sequentially.

- If the node with the specified key is found, the `move_to_front` method adjusts pointers to move it to the front of the list by updating the "next" pointers of the preceding node and the found node.

- **‘display’ method**

- prints the elements of the linked list.

- ‘**elements’ lists**

  - Used to initialize the linked list

- **‘access_sequence’ lists**

- Order in which elements are accessed, employed to illustrate the Move-to-Front behavior.

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

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

    def append(self, data):
        new_node = Node(data)
        if not self.head:
            self.head = new_node
        else:
            current = self.head
            while current.next:
                current = current.next
            current.next = new_node
            new_node.prev = current

    def count_items(self):
        count = 0
        current = self.head
        while current:
            count += 1
            current = current.next
        return count

    def insert_at_front(self, data):
        new_node = Node(data)
        if not self.head:
            self.head = new_node
        else:
            new_node.next = self.head
            self.head.prev = new_node
            self.head = new_node

    def print_reverse(self):
        current = self.head
        while current.next:
            current = current.next

        while current:
            print(current.data, end=" ")
            current = current.prev
        print()

dll = DoublyLinkedList()

dll.append(1)
dll.append(2)
dll.append(3)

print("Appended items in a reverse order:")
dll.print_reverse()

print(f"Number of items: {dll.count_items()}")

dll.insert_at_front(0)

print("\nNew item inserted:")
dll.print_reverse()

print(f"Number of items: {dll.count_items()}")
```

Figure 2.1 Code

```
Appended items in a reverse order:
3 2 1
Number of items: 3

New item inserted:
3 2 1 0
Number of items: 4
```

Figure 2.2 Program Result

- **Node class**

- Represents a node in a doubly linked list.
- Each node contains data, a pointer to the next node (referred to as "next"), and a pointer to the previous node (referred to as "prev").

- **DoublyLinkedList class**

- Serves as a blueprint for creating instances of a doubly linked list. It encapsulates the behavior and properties of a doubly linked list.

- **Append**

- The function of this method is to add a new node to the doubly linked list.
- If the doubly linked list is empty, appending a new node through the `append` method makes it the head of the list. If the list is not empty, the append method places the new node after the last existing node, updating pointers to establish it as the new last node.

- **count_items**

- This function counts the nodes within the linked list by traversing through the list.

- **insert_at_front**

- Inserts a new node in the created linked list
- It adjusts the pointer to insert the new node at the front of the doubly linked list.

- **print_reverse**

- Traverses the list from the head to the end, then prints the data while moving backwards using the ‘prev’ pointers.

- **dll.print_reverse()**

- Prints the list in reverse order.

- **dll.count_items()**

- Counts the number of items inside the list.

- **dll.insert_at_front(0)**

- Inserts a new item (0) in front of the list.

IV. Conclusion

In this laboratory activity, we explored the implementation of linked lists, specifically focusing on doubly linked lists. We demonstrated the Move-to-Front heuristic, a strategy that optimizes access to frequently used elements in a linked list. Additionally, we developed a Python program to create and manipulate doubly linked lists, implementing functionalities such as inserting, counting, and printing in reverse order. Through these exercises, we gained a deeper understanding of linked lists and their applications.

References

[1] GeeksforGeeks. (2023, September 27). *Linked List data structure*.

https://www.geeksforgeeks.org/data-structures/linked-list/

[2] *Doubly Linked List (Python Code with Example) | FavTutor*. (n.d.). FavTutor.

https://favtutor.com/blogs/doubly-linked-list-python

[3] GeeksforGeeks. (2023, November 9). *Self organizing list move to front method*.

https://www.geeksforgeeks.org/self-organizing-list-move-front-method/