Experiment-3

Student Name: Gaganjot Singh **UID: 22BCS14843**
Branch: BE-CSE **Section/Group:** 22BCS_JT_802-B
Semester: 5th **Date of Performance:** 10 August 2024
Subject Name: Advanced Programming **Subject Code:** 22CSP - 314

Question 3.1

1. Aim: Check the given Linked List for 'Cycle'.

2. Objective: The given Linked List has to be checked for Cycle. That is to , check if the

3. Algorithm:

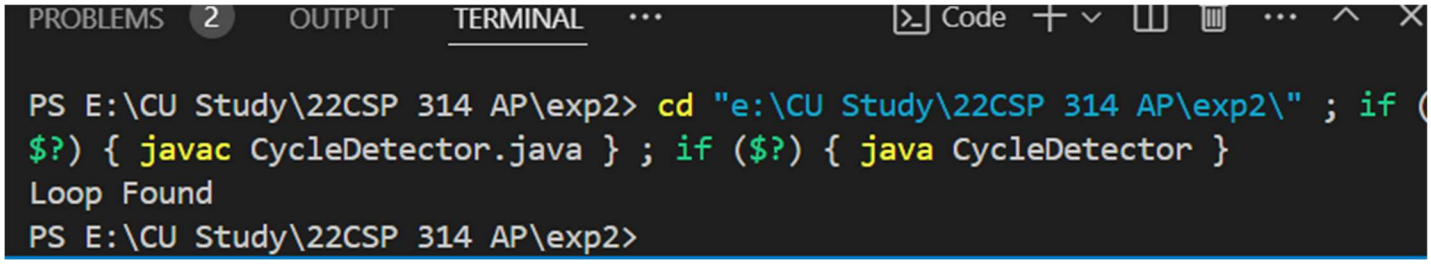
- Initialize the Linked List by Hard-coding a loop/ cycle
- Traverse the list individually and keep putting the node addresses in a Hash Table.
- At any point, if NULL is reached then return false
- If the next of the current nodes points to any of the previously stored nodes in Hash then return true.
- End

3. Implementation/Code:

```
import java.util.HashSet;
import java.util.Set;
class Node {
   int data;
   Node next;
   Node(int new_data) {
      this.data = new_data;
      this.next = null;
   }}

class CycleDetector {
   // Function that returns true if there is a loop in linked list else returns false.
   boolean detectLoop(Node head) {
      Set<Node> set = new HashSet<>();
      // loop that runs till the head is null
      while (head != null) {

         // If this node is already present
         // in hashmap it means there is a cycle
         // (Because you will be encountering the
         // node for the second time).
         if (set.contains(head))
```

```
            return true;
        set.add(head);
        head = head.next;
    }return false;   }
public static void main(String[] args) {
    // 3 -> 23 -> 7 -> 2 -> 77
    Node head = new Node(3);
    head.next = new Node(23);
    head.next.next = new Node(7);
    head.next.next.next = new Node(2);
    head.next.next.next.next = new Node(77);

    // Creating a loop resulting in the linked list:
    // 3 -> 23 -> 7 -> 2 -> 77 -> 3 -> 23 ...
    head.next.next.next.next = head;

    if (detectLoop(head))
        System.out.println("Loop Found");
    else
        System.out.println("No Loop");
}}
```

4. Output:



5. Time Complexity:

The detectLoop function iterates through the linked list once, checking each node for a cycle. The time complexity is O(n) where n is the number of nodes in the linked list.

Question 3.2

1. Aim : Reversing a given Linked List

2. Objective : The objective is to reverse the order of elements in a given linked list. It also involves changing the pointers of the nodes to solve this problem.

3. Algorithm:

- Start of the Program
- The user is prompted to enter the number of elements in the linked list.
- The user is prompted to enter each element of the linked list.
- The elements are added to the linked list using the addNode method.
- The original linked list is displayed using the displayList method.
- The linked list is reversed using the reverseList method.
- the reverseList method uses a simple iterative approach to reverse the linked list.
- It keeps track of the previous node, current node, and next node, and updates the next pointers accordingly.
- Print the reversed List.

3. Implementation/Code :

```
import java.util.Scanner;
class Node {
   int data;
   Node next;

   Node(int data) {
      this.data = data;
      this.next = null;}}

class LinkedList {
   Node head;

   void addNode(int data) {
      Node node = new Node(data);
      if (head == null) {
         head = node;}
 else {Node temp = head;
         while (temp.next != null) {
            temp = temp.next;}
```

```
            temp.next = node;} }

    void displayList(Node node) {
        while (node != null) {
            System.out.print(node.data + " ");
            node = node.next;
        } System.out.println();}

    Node reverseList(Node node) {
        Node prev = null;
        Node current = node;
        Node next = null;

        while (current != null) {
            next = current.next;
            current.next = prev;
            prev = current;
            current = next;}
return prev;}}

public class ReverseLL {
    public static void main(String[] args) {
        LinkedList list = new LinkedList();
        Scanner scanner = new Scanner(System.in);

        System.out.println("Enter the number of elements:");
        int n = scanner.nextInt();

        System.out.println("Enter the elements:");
        for (int i = 0; i < n; i++) {
            list.addNode(scanner.nextInt());}

        System.out.println("Original Linked List:");
        list.displayList(list.head);

        Node reversedHead = list.reverseList(list.head);
        System.out.println(" ");
        System.out.println("Reversed Linked List:");
        list.displayList(reversedHead);
        scanner.close();
    }}
```

4. Output:

```
PS E:\CU Study\22CSP 314 AP\exp2> cd "e:\CU Study\22CSP 314
$?) { javac ReverseLL.java } ; if ($?) { java ReverseLL }
Enter the number of elements:
6
Enter the elements:
1
56
24
0
12
0
Original Linked List:
1 56 24 0 12 0

Reversed Linked List:
0 12 0 24 56 1
```

5. Time Complexity:

The addNode method has a while loop that iterates through the linked list until the end is reached, resulting in a linear time complexity. The reverseList method also has a while loop that iterates through the linked list once, resulting in a linear time complexity. Therefore, the overall time complexity of the code is O(n).

6. Learning Outcomes:

- Both `addNode` and `reverseList` methods have a linear time complexity, O(n).
- Learnt to evaluate the time complexity of linked list operations by examining loops and their relation to the size of the list.
- Identify that the time complexity of methods with single loops through the list is directly proportional to the number of elements.
- Recognize the trade-offs between different operations and their impact on overall performance.