CS 201 Data Structures II – Spring 2024

Instructor: Dr. Muhammad Mobeen Movania

Quiz 3 - Solution

Name: ______________________________________ **Date:** _____________________

Regn. No. : _____________

There are two questions in this quiz. Each question carries 5 marks.

Q1) Give a Θ(n) time non-recursive procedure that reverses a singly linked list of n elements. The procedure should use no more than constant storage beyond that needed for the list itself? (5 marks)

```
class ListNode:
    def __init__(self, value=0, next=None):
        self.value = value
        self.next = next

def reverseLinkedList(head):
    # Check for an empty list or a list with only one element
    if head is None or head.next is None:
        return head

    # Initialize pointers
    previous = None
    current = head
    next_node = None

    # Iterate through the list
    while current is not None:
        # Save the next node
        next_node = current.next

        # Reverse the link
        current.next = previous

        # Move to the next pair of nodes
        previous = current
        current = next_node

    # Previous now points to the new head of the reversed list
    return previous
```

The idea is simple. We iterate through the list, reversing the direction of each link as we go. The previous pointer is used to keep track of the reversed portion of the list, and current is used to traverse the original list. The next_node variable is used to temporarily store the next node in the original list before updating the link.

Q2) Write a SkiplistList method, absorb(l2), that takes as an argument a SkiplistList, l2, empties it and appends its contents, in order, to the receiver. For example, if l1 contains a,b, c and l2 contains d, e, f , then after calling l1.absorb(l2), l1 will contain a,b, c, d, e, f and l2 will be empty. This method should run in O(logn) time.

Algorithm: To implement the absorb function in SkipListList
Step 1: Find the last node of the receiver SkiplistList L1.
Step 2: Find the first node of the SkiplistList L2.
Step 3: Update pointers to append the contents of L2 to the end of the receiver.
Step 4: Empty L2 after absorption.

//C++ implementation:
```
#include <iostream>
#include <vector>
#include <random>

using namespace std;

// Node structure for SkipList
struct Node {
    int value;
    vector<Node*> forward;

    Node(int val, int level) : value(val), forward(level, nullptr) {}
};

class SkiplistList {
private:
    Node* head;
    int max_level;

public:
    SkiplistList() {
        max_level = 1;
        head = new Node(-1, max_level);
    }

    // Find the last node of the SkiplistList
    Node* findLast() {
        Node* current = head;
        for (int i = max_level - 1; i >= 0; --i) {
            while (current->forward[i] != nullptr)
                current = current->forward[i];
        }
        return current;
    }

    // Find the first node of the SkiplistList
    Node* findFirst() {
        return head->forward[0];
    }
    // Absorb method
    void absorb(SkiplistList& l2) {
        Node* last_node = findLast();
        Node* first_node_l2 = l2.findFirst();

        if (first_node_l2 != nullptr) {
            last_node->forward[0] = first_node_l2;
            last_node = l2.findLast();

            if  (last_node  !=  nullptr  &&  last_node->forward.size()  >
max_level)
                max_level = last_node->forward.size();

            // Empty l2
            l2.head->forward.clear();
            l2.head->forward.resize(l2.max_level, nullptr);
        }
    }

    // Print method for debugging
    void printList() {
        Node* current = head->forward[0];
        while (current != nullptr) {
            cout << current->value << " ";
            current = current->forward[0];
        }
        cout << endl;
    }

    // Destructor
    ~SkiplistList() {
        Node* current = head;
        while (current != nullptr) {
            Node* temp = current;
            current = current->forward[0];
            delete temp;
        }
    }
};

int main() {
    SkiplistList l1;
    l1.head->forward[0] = new Node(1, 1);
    l1.max_level = 1;

    SkiplistList l2;
    l2.head->forward[0] = new Node(4, 1);
    l2.head->forward[0]->forward[0] = new Node(5, 1);
    l2.head->forward[0]->forward[0]->forward[0] = new Node(6, 1);
    l2.max_level = 3;
    cout << "Before absorption:" << endl;
    cout << "l1: ";
    l1.printList();
    cout << "l2: ";
    l2.printList();

    l1.absorb(l2);

    cout << "\nAfter absorption:" << endl;
    cout << "l1: ";
    l1.printList();
    cout << "l2: ";
    l2.printList();  // Should be empty

    return 0;
}
```