İTÜ

BLG 223E - Recitation 1
Linked Lists

Wednesday, October, 16, 2024

Res. Assists. Meral Kuyucu & Ali Esad Uğur
korkmazmer@itu.edu.tr, ugura20@itu.edu.tr

P1: Bare Bones Linked List Implementation

- Implement a linked list with basic CRUD capabilities:
  - **Add Node**: Insert at the end of the list
  - **//Insert Node**: Insert at a specific index
  - **Read Node**: Retrieve data by index
  - **//Replace Node**: Update data at a given index
  - **Remove Node**: Delete a node by index
  - **Destroy List**: Delete entire list

Node Definition:

```
struct Node {
        int data;
        Node* next;
};
```

P1: Bare Bones Linked List Implementation

- Use the following function prototypes for your implementation:

```
Node* add_to_linked_list(Node* head, int value);
int read_from_linked_list(Node* head, int index);
Node* remove_from_linked_list(Node* head, int index);
void destroy_linked_list(Node* head);
```

P2: Sorting a Linked List

- **Problem:**
  - Given a linked list of elements sorted by absolute value, sort the values based on actual values. (Hint: You do not have to do pairwise comparison.)
- **Approach:**
  - Traverse the list and move negative values to the front while keeping positive values at the back.
- **Algorithm:**
  - Iterate through the list.
  - If a negative value is found after a positive value, move the negative value to the front.
  - The final list will be sorted by actual values.

P2: Sorting a Linked List

- **Input-Output Example:**
- **Input:**
  - 1 ⇒ -2 ⇒ 3 ⇒ -4 ⇒ 5
- **Process -2:**
  - -2 ⇒ 1 ⇒ 3 ⇒ -4 ⇒ 5
- **Process -4:**
  - -4 ⇒ -2 ⇒ 1 ⇒ 3 ⇒ 5
- **Output:**
  - -4 ⇒ -2 ⇒ 1 ⇒ 3 ⇒ 5

P3: Middle Element of a Linked List

- **Problem:**
  - Find the middle element of a singly linked list in one pass.
- **Approach:**
  - Use the **Tortoise and Hare** algorithm (two pointers: slow and fast).
    - The slow pointer moves one step at a time.
    - The fast pointer moves two steps at a time.
    - When fast reaches the end, slow will be at the middle.

P4: Similar Elements Between Linked Lists

- **Problem:**
  - Given two linked lists, find the common elements and return them as a new list.
- **Approach:**
  - Traverse the first list and for each node, check if it exists in the second list.
  - If a common element is found, add it to a new list.
  - No duplicates assumed.

P4: Similar Elements Between Linked Lists

- **Input-Output Example:**
  - **Input (List 1):** [1] -> [3] -> [5] -> [7] -> [9]
  - **Input (List 2):** [2] -> [3] -> [6] -> [7] -> [10]
  - **Output (List 3):** [3] -> [7]

P5: Palindrome Linked Lists

- **Problem:**
  - Check whether a given linked list is palindrome or not.
  - A list is palindrome if it is read the same forward and backward
- **Approach:**
  - Find the middle of the linked list (like in P3).
  - Reverse the second half.
  - Compare the two halves of the linked list.

P5: Palindrome Linked Lists

- **Input-Output Example:**
  - **Input (List 1):** [1] -> [2] -> [3] -> [2] -> [1]
  - **Output (List 1):** True
  - **Input (List 2):** [1] -> [2] -> [3] -> [4] -> [5]
  - **Output (List 2):** False
  - **Input (List 3):** [1] -> [9] -> [9] -> [1]
  - **Output (List 3):** True

P6: Rotate a Doubly Linked List

- **Problem:**
  - Given a doubly linked list, rotate it to the right or left by a given number of nodes.
- **Approach:**
  - Find the kth node of the list (k is the given number).
  - Move the remaining list partition to the beginning of the whole list.

P6: Rotate a Doubly Linked List

- **Input-Output Example:**
  - **Input (List 1):** [1] -> [2] -> [4] -> [5] -> [6] , k = 2
  - **Output (List 1):** [4] -> [5] -> [6] -> [1] -> [2]