HOMEWORK → palindrome linked list

Week 5 - Data Structures - 26-08-2017

- Implement function to check if a linked list is a palindrome.

T-TALK

- Modify LL? → yes
- Receive Input? → Head of LL
- Expected Output? → Boolean
- If the input is NULL? → Throw exception
- Characters shown? → ASCII
- Type of Linked List? → Singly
- Consider spaces? → yes

E - EXAMPLES

NULL → exception
a → NULL → true
a → b → a → NULL → true
a → _ → b → a → NULL → false
a → b → c → NULL → false
a → _ → a → _ → a → NULL → true

B - BRUTE FORCE

[a] → [b] → [b] → [a] → NULL

- ✓ Get length O(n)
- ✓ convert to string
- ✓ check is string is palindrome

for (int i = 0; i < str.length/2; i++)
→ check char of the beginning and at the end.
→ If different return FALSE

Time complexity ≈ O(n)
Space complexity = O(n)

O - OPTIMIZATION

OPTION 1

✓ Use stack, However still will have space of O(n)

OPTION 2

✓ Reverse half of LL
✓ No additional space needed

[c] → [b] → [b] → [a] → NULL

* be careful for odd

[c] → [b] → [c] → [b] → [a] → NULL

WALK THROUGH

- check for Null
- Use two pointers to get to the center
- Reverse second half of the list
- Compare
  - If fast pointer != NULL move slow pointer to next

① a → b → c → b → a → NULL

② a → b → b → a → NULL

① a → b → NULL
& ② a → b → NULL

I- IMPLEMENTATION

```
// Definition singly LL.
struct ListNode {
    int val;
    ListNode *next;
    ListNode(int x) : val(x), next(NULL) {}
};

bool isPalindrome(ListNode *head) {
  if (head == NULL){
      throw invalid_argument("NULL");
  }
  if (head->next == NULL){
     return true;
  }
  ListNode * prev = head;
  ListNode * slow = head;
  ListNode * fast = head;
  while (fast != NULL && fast.next != NULL) {
     fast = fast->next->next
     prev = slow
     slow = slow->next;
  }
  if (fast != NULL){
     slow = slow->next; //for odd cases
  }
  ListNode * second = reverse(slow);
  prev->next = NULL

  while (head != NULL && second != NULL)
     if (head->val != second->val){
         retur false;
     }
     head = head->next;
     second = second->next;
  }

  return true;
}
```

```
ListNode reverse(ListNode head){
  ListNode * prev = NULL
  ListNode * curr = head;
  ListNode* nxt;
  while (curr != NULL){
     nxt = curr->next;
     curr->next = prev
     prev = cur;
     curr = nxt;
  }
  return prev;
```

T-TEST

- NULL → throws exception
- ✓ a → NULL → true
- ✓ a → b → a → true
  - a → a
  - a → a
- ✓ a → b → b → a → true
  - a → b
  - a → b
- ✓ a → b → c → false
  - a → a
  - c → c