**Linked List Palindrome**

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

**T- Talk (/Listen/Clarify)**

-Does capitalization matter? Yes

-Does the function return a boolean value? Yes

-What type of information is stored in the data part of each node? Any ASCII character is allowed

-What is the input is null? Throw an exception

**E-Examples (/Test/TDD, Out of the Box)**

dictionary: { pear, salmon, foot, prints, footprints, leave, you, sun, girl, enjoy }

|  |  |  |
| --- | --- | --- |
| Case | Input | Output |
| Capitalization | N->o->o->n | False |
| Null | Null | Throws an exception |
| Is a palindrome | n->o->o->n | True |

**B- Brute Force**

-create another linked list that is the reverse of the linked list provided.

-compare the each spot in both linked lists and see if the values of the nodes are the same

-if you come across a node where the values are different, then return false. It is not a palindrome.

-if all the spots in both linked lists and see if the values of the nodes are the same return true

Time and space complexity: O(n)

**O- Optimize**

-Traverse to the middle of the linked list and reverse the second half.

-Have a pointer that starts at the beginning and the middle.

-Simultaneously move the two pointers and compare the values to see if they are the same at each point

-if they are return true, if not return false

Time complexity: O(n)

Space complexity: O(1)

**W- Walk Through**

**-**create a method that takes in a linked list and returns a boolean value

**-**if the input string contains one word and that word is in the dictionary return true

-start at the first letter of the input string and check if it is in the dictionary.

-keep increasing the letters checked in the input string until a word is found in the dictionary

-if not found return false

-if found store it in a variable.

-repeat this process for the remainder of the string by calling the same method and passing in the remainder of the input string and the dictionary

**I- Implement**

bool isPalindrome (node \*head) {

if (head == NULL) {

return true;

}

node \*p = head->next;

node \*trail = head;

//create a linked list in the reverse order of the given //linked list

while (p->next != NULL) {

node \*temp = p->next;

temp->next = trail;

trail = temp;

p = p->next;

}

//create pointers to traverse the given linkedlist and //the reversed one

node \*pointer = head;

node \*reversePointer = trail;

//check if the spot in the given linkedlist matches the //value in the reversed one

while (pointer != NULL) {

//if doesn't match then isn't a palindrome

if (pointer->data != reversePointer->data) {

return false;

}

pointer = pointer->next;

reversePointer = reversePointer->next;

}

//is a palindrome

return true;

}

**T- Test**

|  |  |  |
| --- | --- | --- |
| Case | Input | Output |
| Capitalization | N->o->o->n | False |
| Null | Null | Throws an exception |
| Is a palindrome | n->o->o->n | True |