Check if given Linked List is Plaindrome

Check if the given Linked List is Palindrome

Problem Statement: Given the head of a singly linked list, return true if it is a palindrome.

Examples:

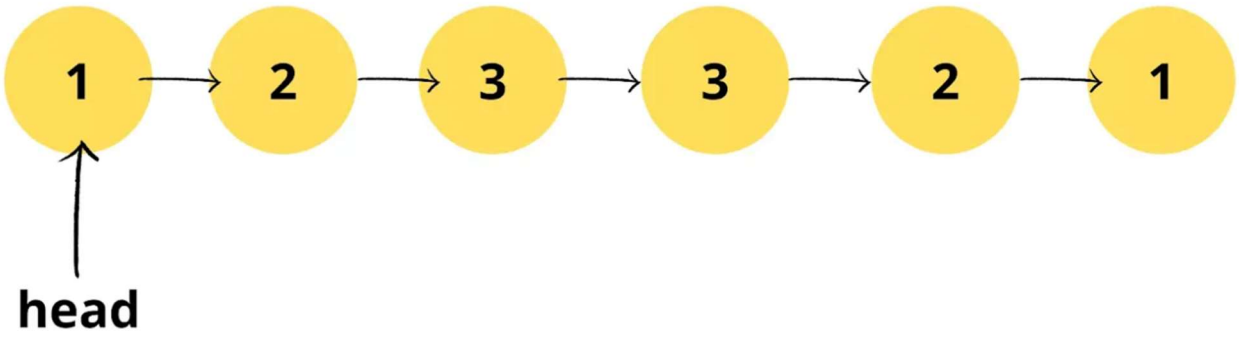
Example 1:

Input: head = [1,2,3,3,2,1]

Output:

true

Explanation: If we read elements from left to right, we get [1,2,3,3,2,1]. When we read elements from right to left, we get [1,2,3,3,2,1]. Both ways list remains same and hence, the given linked list is palindrome.



Example 2:

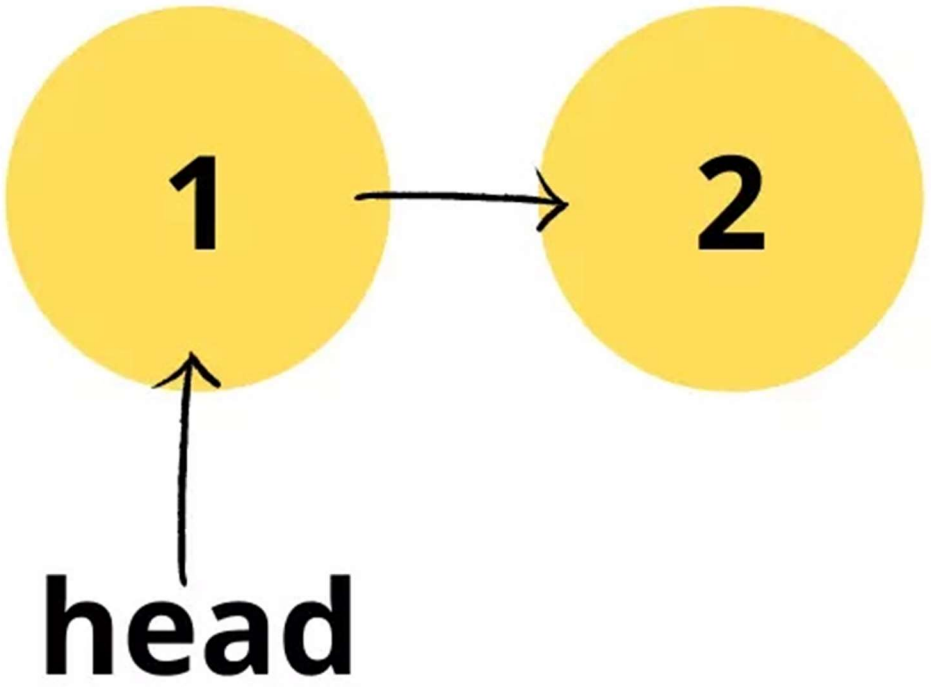
Input:

head = [1,2]

Output:

false

Explanation: When we read elements from left to right, we get [1,2]. Reading from right to left, we get a list as [2,1]. Both are different and hence, the given linked list is not palindrome.



Solution: Using the extra data structure

Approach:

We can store elements in an array. Then check if the given array is a palindrome. How to check if an array is a palindrome?

Let's take a string, say "level" which is a palindrome. Let's observe a thing.

Letter	Position	Letter	Position
l	0	l	4
e	1	e	3
v	2	v	2

So we can see that each index letter is the same as (length-each index -1) letter.

The same logic required to check an array is a palindrome.

Following are the steps to this approach.

- Iterate through the given list to store it in an array.
- Iterate through the array.
- For each index in range of n/2 where n is the size of the array
- Check if the number in it is the same as the number in the n-index-1 of the array.

Code:

- C++ Code
- Java Code
- Python Code

```cpp
#include<bits/stdc++.h>
using namespace std;

class node {
    public:
        int num;
        node* next;
        node(int val) {
            num = val;
            next = NULL;
        }
};

void insertNode(node* head,int val) {
    node* newNode = new node(val);
    if(head == NULL) {
        head = newNode;
        return;
    }

    node* temp = head;
```

```
    while(temp->next != NULL) temp = temp->next;

    temp->next = newNode;
    return;
}

bool isPalindrome(node* head) {
    vector<int> arr;
    while(head != NULL) {
        arr.push_back(head->num);
        head = head->next;
    }
    for(int i=0;i<arr.size()/2;i++)
        if(arr[i] != arr[arr.size()-i-1]) return false;
    return true;
}

int main() {
    node* head = NULL;
    insertNode(head,1);
    insertNode(head,2);
    insertNode(head,3);
    insertNode(head,2);
    insertNode(head,1);
    isPalindrome(head)? cout<<"True" : cout<<"False";
    return 0;
}
```

Output: True

Time Complexity: O(N)

Reason: Iterating through the list to store elements in the array.

Space Complexity: O(N)

Reason: Using an array to store list elements for further computations.