LINKED LIST

/\*

\* Search (iterative)

\*

\* Search for a key in a linked list return the position where it is found .

\* if not found , return-1.

\*/

public class IterativeSearchLinkedList {

// Search Linked List

public static int itrSearchLL(int key){

int position=0;

Node temp=head;

while(temp!=null){

if(temp.data==key){

return position;

}

temp=temp.next;

position++;

}

return -1;

}

public static void main(String[] args) {

IterativeSearchLinkedList ll = new IterativeSearchLinkedList();

ll.addFirst(5);

ll.addFirst(4);

ll.addFirst(3);

ll.addFirst(2);

ll.addMid(2,10);

ll.addLast(6);

ll.addLast(7);

ll.addMid(0, 50);

printLL();

System.out.println("\nSize of LinkedList : " + ll.size);

removeFirst();

printLL();

System.out.println("\nSize of LinkedList : " + ll.size);

removeLast();

printLL();

System.out.println("\nSize of LinkedList : " + ll.size);

System.out.println("Position: " + itrSearchLL(4));

}

public static class Node{

int data;

Node next;

Node(int data){

this.data = data;

this.next = null;

}

}

public static Node head;

public static Node tail;

public static int size=0;

// Remove in first

public static int removeFirst(){

if (size==0) {

System.out.println("Linked List is Empty");

return Integer.MIN\_VALUE;

}

if(size==1){

int data = head.data;

head = tail = null;

size=0;

return data;

}

int data = head.data;

head = head.next;

size--;

return data;

}

// remove Last

public static int removeLast(){

if (size==0) {

System.out.println("Linked List is Empty");

return Integer.MIN\_VALUE;

}

if(size==1){

int data = tail.data;

head = tail = null;

size=0;

return data;

}

Node prev = head;

for(int i=0; i<size-2; i++){

prev = prev.next;

}

int data = tail.data;

prev.next= null;

tail = prev;

size--;

return data;

}

// Add in middles

public void addMid(int index, int data){

if (index==0) {

addFirst(data);

return;

}

Node nextNode = new Node(data);

size++;

Node temp = head;

int i=0;

while (i<index-1) {

temp=temp.next;

i++;

}

nextNode.next=temp.next;

temp.next = nextNode;

}

// add First

public void addFirst(int data){

Node nextNode = new Node(data);

size++;

if(head==null){

head=tail=nextNode;

return;

}

nextNode.next= head;

head=nextNode;

}

// add Last

public void addLast(int data){

Node nextNode = new Node(data);

size++;

if(head==null){

head=tail=nextNode;

return;

}

tail.next= nextNode;

tail=nextNode;

}

// Print Linked List

public static void printLL(){

if (head==null) {

System.out.println("LinkedList is a Emtey");

return;

}

Node temp = head;

while (temp!=null) {

System.out.print(temp.data+ " ");

temp = temp.next;

}

}

}

/\*

\* Output

\* 50 2 3 10 4 5 6 7

\* Size of LinkedList : 8

\* 2 3 10 4 5 6 7

\* Size of LinkedList : 7

\* 2 3 10 4 5 6

\* Size of LinkedList : 6

\* Position: 3

\*/

public class LinkedList {

public static class Node {

int data;

Node next;

Node(int data){

this.data = data;

this.next = null;

}

}

public static Node head;

public static Node tail;

// add -> first, last

public void addFist(int data) {

// step1: create new node

Node newNode = new Node(data);

if (head == null) {

head = tail = newNode;

return;

}

// step2: newNode next = head

newNode.next = head; //link

}

public static void main(String[] args) {

LinkedList list = new LinkedList();

list.addFist(1);

list.addFist(2);

}

}

3. public class LinkedListIsPalindrome {

public static class Node{

int data;

Node next;

Node(int data){

this.data=data;

this.next=null;

}

}

public static Node head, tail;

// print Linked List

public static void printLL(){

Node temp=head;

while (temp!=null) {

System.out.print(temp.data + " ");

temp=temp.next;

}

System.out.println();

}

// add first

public static void addFirst(int data){

Node nextNode = new Node(data);

if (head==null) {

head = tail = nextNode;

return;

}

nextNode.next = head;

head = nextNode;

}

// Slow-Fast Approach

public static Node findMid(Node head){

Node slow = head;

Node fast = head;

while (fast != null && fast.next != null) {

slow = slow.next;

fast = fast.next.next;

}

return slow; // slow is my midNode

}

public static boolean checkPalindrome(){

if (head==null || head.next == null) {

return true;

}

// step1 - find mid

Node midNode = findMid(head);

// step2 - reverse 2nd half

Node prev = null;

Node curr = midNode;

Node next;

while (curr != null) {

next = curr.next;

curr.next = prev;

prev = curr;

curr = next;

}

Node right = prev; // right half head

Node left = head;

//step3 - check left half & right half

while (right != null) {

if (left.data != right.data) {

return false;

}

left = left.next;

right= right.next;

}

return true;

}

public static void main(String[] args) {

addFirst(1);

addFirst(2);

addFirst(2);

addFirst(1);

printLL();

System.out.println(checkPalindrome());

}

}

/\*

* Output:
* 1 2 2 1
* true \*/