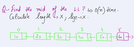
Lecture 29
Monday 11/06/23 - 1:00PM

Find all 1 in a linked list

① Access 1st
② Access 2nd
③ Access 3rd

Q: Find the mid of the list? in $O(n)$ time.

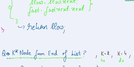
Calculate length of x, dep i=0



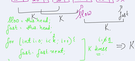
count = 0; temp = first node;
while(temp != null){
count++;
}

public Node getMidNode() {
Node slow = this.head;
Node fast = this.head;
while (fast.next != null && fast.next != null){
slow = slow.next;
fast = fast.next.next;
}
return slow;
}

Q: 2nd Node from end of list? , k=1, k=2, k=3



$O(n) \rightarrow$ time.
$O(1) \rightarrow$ space.



slow = slow.next;
fast = fast.next;
for (int i=1; i<k; i++){
fast = fast.next;
}
while(fast != null){
slow = slow.next;
fast = fast.next;
}
return slow.data;

$\Rightarrow$ K distance

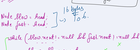
$O(n)$ time
$O(1)$ space

Q: Detect Cycle in linked list?

while (slow.next != null && fast.next != null && fast.next.next != null){
slow = slow.next;
fast = fast.next.next;
if (slow == fast){
return true;
}
}
return false;

$\Rightarrow$ return false

Floyd - Cycle - Detection algo



$\rightarrow m$
$\rightarrow n$.
$O(m+n)$

Merge 2 Sorted lists

linkedList thisL = new LinkedList();
while (this.temp != null && other.temp != null){
if (this.temp.data < other.temp.data){
thisL.addNext(this.temp.data);
this.temp = this.temp.next;
} else {
thisL.addNext(other.temp.data);
other.temp = other.temp.next;
}
}
while (this.temp != null){
thisL.addNext(this.temp.data);
this.temp = this.temp.next;
}
while (other.temp != null){
thisL.addNext(other.temp.data);
other.temp = other.temp.next;
}
return thisL;
}

Merge Sort

Q: Implement sort a LL?

Middle node (getMiddle()) ;
this.mergeSort(this.getMiddle());
LinkedList first = new LinkedList(this.getHead, mid);
LinkedList second = new LinkedList(this.getNext(mid));
first = first.mergeSort();
second = second.mergeSort();
first = first.merge(second);
return merged;

Q: Linked List as stack?

Linked as a Stack?

push(10);
push(20);
push(30);

Queue DR using L.L.2

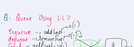
enqueue(10);
enqueue(20);
enqueue(30);

Queue at 1:45pm

Q: Reverse DR in ① → Queue

public void reverseDR(){
reverseDR(this.head, 0);
}

private void reverseDR(Node left, Node right, int level){
if (right == null){
return;
}
reverseDR(left, right.next, level+1);
if (level >= this.size()/2){
int temp = left.data;
left.data = right.data;
right.data = temp;
left = left.next;
}
}

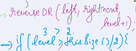


public void reverseDR(){
reverseDR(this.head, 0);
}

private void reverseDR(Node left, Node right, int level){
if (right == null){
return;
}
reverseDR(left, right.next, level);
if (level >= this.size()/2){
int temp = left.data;
left.data = right.data;
right.data = temp;
left = left.next;
}
}

int[] arr = {1, 2, 3, 4, 5};

Count of distinct rectangles



Count(int[] arr){
int[] arr = {1, 2, 3, 4, 5};
int left = 0;
int right = arr.length - 1;
int count = 0;
while (left < right){
if (arr[left] * arr[right] < area){
count = count + (right - left);
left++;
} else {
right--;
}
}

$\rightarrow O(n) + C$
$\rightarrow O(1)$ SC

LinkedList
thisL
thisNext