Mario Bolivar

Mjb160330

CE 3345.001

Professor Chida

Assignment 3

1. Write pseudocode to swap two adjacent elements by adjusting only the links (and not the data) using
   1. Singly linked list

If A & B are being swapped: **X1-> A-> B-> X2->**

**Loop to node before first element that will be swapped and create a temporary node pointer pointing to it (left most node : X1).**

**Get right most node and use a temp node ptr to point to it (X2 = X1.next.next.next)**

**Node B points to Node A**

**Node X1 points to Node B**

**Node A points to Node X2**

**Result: X1-> B-> A-> X2->**

**Node left = X1 (Node ptr points to X1)**

**Node right = left.next.next.next (Node ptr points to X2)**

**left.next.next.next = left.next (B points to A)**

**left.next = left.next.next (X1 points to B)**

**left.next.next.next = right (A points to X2)**

* 1. Doubly linked list

**Loop to node before first element that will be swapped and create a temporary node pointer pointing to it (left most node : X1).**

**A.prev points to B**

**B.prev points to X1**

**X1 points to B**

**X2.prev points to A**

**B.next points to A**

**A.next points to X2**

1. Write pseudocode to implement the contains routine for MyLinkedList

**Public boolean contains ( AnyType data )**

**For each item in linked list, check if current node data matches the date being searched for**

**If found return true**

**If not found in list return false**

1. The Josephus problem is the following game: N people, numbered 1 to N, are sitting in a circle.

Starting at person 1, a hot potato is passed. After M passes, the person holding the potato is eliminated, the circle closes ranks, and the game continues with the person who was sitting after the eliminated person picking up the hot potato. The last remaining person wins. Thus if M=0 and N=5, players are eliminated in order and player 5 wins. If M=1 and N=5, the order of elimination is 2,4,1,5.

1. Write a program to solve the Josephus problem for general (integer) values of M and N.

Try to make your program as efficient as possible. Make sure you dispose of cells.

**Node ptr = head;**

**//While node is not last remaining**

**while(ptr.next != ptr){**

**//Move M times**

**for(int i = 0; i < M; i++)**

**ptr = ptr.next;**

**System.out.println(ptr.data + " : was removed.");**

**//Removing node**

**Node temp = ptr.prev;**

**ptr.prev = null;**

**ptr.next.prev = temp;**

**temp.next = ptr.next;**

**ptr.next = null;**

**}**

b. What is the running time of your program? -- **O()**

1. What is the running time of the following code?

public static List<Integer> makeList( int N )

{

ArrayList<Integer> lst = new ArrayList<>();

for( int i = 0; i < N; i++) **— O(N)**

{

lst.add(i); **— O(N)**

lst.trimToSize(); **— O(N)**

}

}

Time Complexity: **= N \* (N + N) = = O()**

1. The following routine removes the first half of the list passed as a parameter:

public static void removeFirstHalf(List<?> lst)

{

int theSize = lst.size() /2

for( inti = 0; i < theSize; i++ )

lst.remove(0);

}

1. Why is theSize saved prior to entering the for loop?
   * **So that the for loops stops at the end of the first half of the list.**
2. What is the running time of removeFirstHalf if lst is an ArrayList?
   * **O().**
3. What is the running time of removeFirstHalf if lst is a LinkedLIst?
   * **O(N).**
4. Does using an iterator make removeFirstHalf faster for either type of List
   * **No.**
5. Write a function in pseudocode named removeDuplicates(), which takes a singly linked list sorted in increasing order and deletes any duplicate nodes from the list. The list should only be traversed once and the routine should not call any other routine. For example if the linked list is 11->11->11->21->43->43->60 then removeDuplicates() should convert the list to 11->21->43->60.

**Function removeDuplicates(Node head) pseudocode.**

**Node ptr = head**

**For(every node in linked list)**

**Node ptr2 = ptr (first value)**

**While(ptr2.data == ptr2.next.data) (while current and next nodes are duplicates)**

**ptr2 = ptr2.next (move to next node)**

**EndW**

**ptr.next = ptr2.next (Trim duplicates GC will clean up trash)**

**ptr = ptr.next (Move to next node after trimming duplicates if any)**

**Next (Repeat until at end of list)**

**return head**

**Input:** the **head** node of the linked list .

**Output:** Your function should return a pointer to the head of linked list with no duplicate element.