DS-IA Linked List Tutorial

Chirag:

Hello friends, I am Chirag Jethwa and Welcome to our channel! Today we are going to solve a problem on linked list. Here goes the problem statement:

Count all the duplicate elements present in the linked list, remove the duplicates (single element and multiple elements) and sort the linked list.

So, there are basically three approaches to achieve this:

1. Maintain two loops
2. Sort the linked list and then remove duplicates
3. Use hashing

In first method, we maintain two pointers; p and q, and then use it to traverse the linked list. For current value of p node, we check if corresponding value is present in rest of the linked list. If yes, we remove it, else we proceed to next value of p directly.

In second method, we first sort the linked list and then keep checking for a particular value, till we encounter a different value. We will free those nodes which are repeated.

Third method is hashing, which is very efficient, but it has its own limitations. It is a higher level concept so we won’t touch on that yet.

For the sake of simplicity, we will solve the problem using the first approach, i.e. maintaining two loops for finding and removing duplicates in an unsorted linked list.

So, before we jump into coding, let’s first visualize the problem statement and its approach. My pal Nikhil and Arnav shall lead you into the visualization world for complete understanding!

Nikhil/Arnav: **Speak as per your Prezi, as I don’t know the contents. You both can update your dialogs here for common reference.**

Now that you all have understood how to go about the problem statement, let’s jump right into coding part with Arghyadeep and Jaydeep!

Arghyadeep:

So, hey everyone, Arghyadeep here. We are now going to scan through the code in C to understand the structure of the program and program flow.

So, first, we will import the preprocessor directives stdio.h and stdlib.h before making further progress. Now we shall define a node structure for our linked list, which is our abstract data type. We define integer type data, and a pointer to next node. We use typedef to reduce the hassle of typing struct \*node everywhere.

Now we have defined two pointers, end and start to maintain the head and tail of the linked list. We also maintain a counter to count number of elements and max variable to identify the largest value.

Now, let’s start defining functions for different operations on our linked list.

We first define the function int insert (int x, int position). It takes in the element to be inserted along with its position from the user. However, if user enters multiple values at same position, the previous nodes keep shifting right. You can see that we have defined two pointers; ptr and p. ptr is used to create new node each time the insert function is called. Now we run through various conditions checking. If the inserted element is bigger than the current value of variable max, we reinitialize value of max to inserted element. This is being done to identify the largest element in the linked list.

Next if statement is quite obvious, is for error handling when we reach end of linked list. If we are creating first node, that implies the start pointer and end pointer are the same and they both point to the first node.

Next condition check is if the element position entered is more than the total number of elements present in the linked list. If it is so, the element is added at the end of the current linked list. If the position of element entered is 1, it implies that it’s supposed to be added in the beginning and the start pointer should point to it. If the position entered doesn’t fall under any of the above conditions discussed, we add the element at the user-entered position and shuffle the links among the node. After all conditions are checked, we increment the counter by 1.

Now let’s check out the duplicate function. This function has been defined to identify and count the number of duplicates present in the linked list. We define a pointer p for traversing and few int type variables for maintaining the count of duplicates and flag variable to check if list is duplicate free.

Now we define a for loop which will run till max element encountered. We set pointer p to start node and set repeat counter to zero. Inside the while loop, we traverse till the end of the linked list and check if the variable i is equal to value of the data present at node p. If it is equal, we increment counter by 1 and retrieve the value of that node in temp variable. Now if the value of the rpt variable is more than 1, it implies the element is repeated in the linked list and we accordingly print the element along with its number of occurrences. If flag variable is 0, i.e. it does not get changed in the entire loop of complexity O(n2), it implies that there are no repeat elements.

Now, to explain the remaining functions, I would request my friend Jaydeep to kindly take this ahead.