Delete nodes which have a greater value on right side

Given a singly linked list, remove all the nodes which have a greater value on right side.

We strongly recommend that you click here and practice it, before moving on to the solution.

Examples:

a) The list 12->15->10->11->5->6->2->3->NULL should be changed to 15->11->6->3->NULL. Note that 12, 10, 5 and 2 have been deleted because there is a greater value on the right side.

When we examine 12, we see that after 12 there is one node with value greater than 12 (i.e. 15), so we delete 12.
When we examine 15, we find no node after 15 that has value greater than 15 so we keep this node.
When we go like this, we get 15->6->3

b) The list 10->20->30->40->50->60->NULL should be changed to 60->NULL. Note that 10, 20, 30, 40 and 50 have been deleted because they all have a greater value on the right side.

c) The list 60->50->40->30->20->10->NULL should not be changed.

Method 1 (Simple)

Use two loops. In the outer loop, pick nodes of the linked list one by one. In the inner loop, check if there exist a node whose value is greater than the picked node. If there exists a node whose value is greater, then delete the picked node.
Time Complexity: O(n^2)

Method 2 (Use Reverse)

Thanks to Paras for providing the below algorithm.
1. Reverse the list.
2. Traverse the reversed list. Keep max till now. If next node < max, then delete the next node, otherwise max = next node. 3. Reverse the list again to retain the original order. Time Complexity: O(n) Thanks to R.Srinivasan for providing below code.

C

```
// C program to delete nodes which have a greater value on
// right side
#include <stdio.h>
#include <stdlib.h>

/* structure of a linked list node */
struct node
{
     int data;
     struct node *next;
};

/* prototype for utility functions */
void reverseList(struct node **headref);
void _delLesserNodes(struct node *head);

/* Deletes nodes which have a node with greater value node
  on left side */
void delLesserNodes(struct node **head_ref)
{
    /* 1) Reverse the linked list */
    reverseList(head_ref);

    /* 2) In the reversed list, delete nodes which have a node
```

```
       with greater value node on left side. Note that head
       node is never deleted because it is the leftmost node.*/
    _delLesserNodes(*head_ref);

    /* 3) Reverse the linked list again to retain the
       original order */
    reverseList(head_ref);
}

/* Deletes nodes which have greater value node(s) on left side */
void _delLesserNodes(struct node *head)
{
     struct node *current = head;

     /* Initialize max */
     struct node *maxnode = head;
     struct node *temp;

     while (current != NULL && current->next != NULL)
     {
         /* If current is smaller than max, then delete current */
         if(current->next->data < maxnode->data)
         {
             temp = current->next;
             current->next = temp->next;
             free(temp);
         }

         /* If current is greater than max, then update max and
            move current */
         else
         {
             current = current->next;
             maxnode = current;
         }
     }
}

/* Utility function to insert a node at the begining */
void push(struct node **head_ref, int new_data)
{
     struct node *new_node =
               (struct node *)malloc(sizeof(struct node));
     new_node->data = new_data;
     new_node->next = *head_ref;
     *head_ref = new_node;
}

/* Utility function to reverse a linked list */
void reverseList(struct node **headref)
{
     struct node *current = *headref;
     struct node *prev = NULL;
     struct node *next;
     while (current != NULL)
     {
          next = current->next;
          current->next = prev;
          prev = current;
          current = next;
     }
     *headref = prev;
}

/* Utility function to print a linked list */
void printList(struct node *head)
{
     while (head != NULL)
     {
        printf("%d ", head->data);
        head=head->next;
     }
     printf("\n");
```

```
}

/* Driver program to test above functions */
int main()
{
    struct node *head = NULL;

    /* Create following linked list
      12->15->10->11->5->6->2->3 */
    push(&head,3);
    push(&head,2);
    push(&head,6);
    push(&head,5);
    push(&head,11);
    push(&head,10);
    push(&head,15);
    push(&head,12);

    printf("Given Linked List \n");
    printList(head);

    delLesserNodes(&head);

    printf("Modified Linked List \n");
    printList(head);

     return 0;
}
```

Java

```
// Java program to delete nodes which have a greater value on
// right side
class LinkedList
{
    Node head;  // head of list

    /* Linked list Node*/
    class Node
    {
        int data;
        Node next;
        Node(int d)  { data = d; next = null; }
    }

    /* Deletes nodes which have a node with greater
       value node on left side */
    void delLesserNodes()
    {
        /* 1.Reverse the linked list */
        reverseList();

        /* 2) In the reversed list, delete nodes which
           have a node with greater value node on left
           side. Note that head node is never deleted
           because it is the leftmost node.*/
        _delLesserNodes();

        /* 3) Reverse the linked list again to retain
           the original order */
        reverseList();
    }

    /* Deletes nodes which have greater value node(s)
       on left side */
    void _delLesserNodes()
    {
        Node current = head;

        /* Initialise max */
```

```
        Node maxnode = head;
        Node temp;

        while (current != null && current.next != null)
        {
            /* If current is smaller than max, then delete
               current */
            if (current.next.data < maxnode.data)
            {
                temp = current.next;
                current.next = temp.next;
                temp = null;
            }

            /* If current is greater than max, then update
               max and move current */
            else
            {
                current = current.next;
                maxnode = current;
            }
        }
    }

    /* Utility functions */

    /* Inserts a new Node at front of the list. */
    void push(int new_data)
    {
        /* 1 & 2: Allocate the Node &
                  Put in the data*/
        Node new_node = new Node(new_data);

        /* 3. Make next of new Node as head */
        new_node.next = head;

        /* 4. Move the head to point to new Node */
        head = new_node;
    }

    /* Function to reverse the linked list */
    void reverseList()
    {
        Node current = head;
        Node prev = null;
        Node next;
        while (current != null)
        {
            next = current.next;
            current.next = prev;
            prev = current;
            current = next;
        }
        head = prev;
    }

    /* Function to print linked list */
    void printList()
    {
        Node temp = head;
        while (temp != null)
        {
            System.out.print(temp.data+" ");
            temp = temp.next;
        }
        System.out.println();
    }

    /* Drier program to test above functions */
    public static void main(String args[])
    {
        LinkedList llist = new LinkedList();
```

```
        /* Constructed Linked List is 12->15->10->11->
           5->6->2->3 */
        llist.push(3);
        llist.push(2);
        llist.push(6);
        llist.push(5);
        llist.push(11);
        llist.push(10);
        llist.push(15);
        llist.push(12);

        System.out.println("Given Linked List");
        llist.printList();

        llist.delLesserNodes();

        System.out.println("Modified Linked List");
        llist.printList();
    }
} /* This code is contributed by Rajat Mishra */
```

Output:

```
Given Linked List
12 15 10 11 5 6 2 3
Modified Linked List
15 11 6 3
```

Source:
http://geeksforgeeks.org/forum/topic/amazon-interview-question-for-software-engineerdeveloper-about-linked-lists-6