Given a linked list of line segments, remove middle points

Given a linked list of co-ordinates where adjacent points either form a vertical line or a horizontal line. Delete points from the linked list which are in the middle of a horizontal or vertical line.

Examples:

```
Input:  (0,10)->(1,10)->(5,10)->(7,10)
                                 |
                               (7,5)->(20,5)->(40,5)
Output: Linked List should be changed to following
        (0,10)->(7,10)
                  |
                (7,5)->(40,5)
The given linked list represents a horizontal line from (0,10)
to (7, 10) followed by a vertical line from (7, 10) to (7, 5),
followed by a horizontal line from (7, 5) to (40, 5).

Input:     (2,3)->(4,3)->(6,3)->(10,3)->(12,3)
Output: Linked List should be changed to following
    (2,3)->(12,3)
There is only one vertical line, so all middle points are removed.
```

Source: Microsoft Interview Experience

We strongly recommend to minimize the browser and try this yourself first.

The idea is to keep track of current node, next node and next-next node. While the next node is same as next-next node, keep deleting the next node. In this complete procedure we need to keep an eye on shifting of pointers and checking for NULL values.

Following are C/C++ and Java implementations of above idea.

C/C++

```
// C program to remove intermediate points in a linked list
// that represents horizontal and vertical line segments
#include <stdio.h>
#include <stdlib.h>

// Node has 3 fields including x, y coordinates and a pointer
// to next node
struct node
{
    int x, y;
    struct node *next;
};

/* Function to insert a node at the beginning */
void push(struct node ** head_ref, int x,int y)
{
    struct node* new_node =
           (struct node*) malloc(sizeof(struct node));
    new_node->x  = x;
    new_node->y  = y;
    new_node->next = (*head_ref);
    (*head_ref)  = new_node;
}

/* Utility function to print a singly linked list */
void printList(struct node *head)
{
    struct node *temp = head;
    while (temp != NULL)
```

```
    {
        printf("(%d,%d)-> ", temp->x,temp->y);
        temp = temp->next;
    }
    printf("\n");

}

// Utility function to remove Next from linked list
// and link nodes after it to head
void deleteNode(struct node *head, struct node *Next)
{
    head->next = Next->next;
    Next->next = NULL;
    free(Next);
}

// This function deletes middle nodes in a sequence of
// horizontal and vertical line segments represented by
// linked list.
struct node* deleteMiddle(struct node *head)
{
    // If only one node or no node...Return back
    if (head==NULL || head->next ==NULL || head->next->next==NULL)
        return head;

    struct node* Next = head->next;
    struct node *NextNext = Next->next ;

    // Check if this is a vertical line or horizontal line
    if (head->x == Next->x)
    {
        // Find middle nodes with same x value, and delete them
        while (NextNext !=NULL && Next->x==NextNext->x)
        {
            deleteNode(head, Next);

            // Update Next and NextNext for next iteration
            Next = NextNext;
            NextNext = NextNext->next;
        }
    }
    else if (head->y==Next->y) // If horizontal line
    {
        // Find middle nodes with same y value, and delete them
        while (NextNext !=NULL && Next->y==NextNext->y)
        {
            deleteNode(head, Next);

            // Update Next and NextNext for next iteration
            Next = NextNext;
            NextNext = NextNext->next;
        }
    }
    else  // Adjacent points must have either same x or same y
    {
        puts("Given linked list is not valid");
        return NULL;
    }

    // Recur for next segment
    deleteMiddle(head->next);

    return head;
}

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

    push(&head, 40,5);
    push(&head, 20,5);
```

```
    push(&head, 10,5);
    push(&head, 10,8);
    push(&head, 10,10);
    push(&head, 3,10);
    push(&head, 1,10);
    push(&head, 0,10);
    printf("Given Linked List: \n");
    printList(head);

    if (deleteMiddle(head) != NULL);
    {
        printf("Modified Linked List: \n");
        printList(head);
    }
    return 0;
}
```

Java

```
// Java program to remove middle points in a linked list of
// line segments,
class LinkedList
{
    Node head;  // head of list

    /* Linked list Node*/
    class Node
    {
        int x,y;
        Node next;
        Node(int x, int y)
        {
            this.x = x;
            this.y = y;
            next = null;
        }
    }

    // This function deletes middle nodes in a sequence of
    // horizontal and vertical line segments represented
    // by linked list.
    Node deleteMiddle()
    {
        // If only one node or no node...Return back
        if (head == null || head.next == null ||
            head.next.next == null)
            return head;

        Node Next = head.next;
        Node NextNext = Next.next;

        // check if this is vertical or horizontal line
        if (head.x == Next.x)
        {
            // Find middle nodes with same value as x and
            // delete them.
            while (NextNext != null && Next.x == NextNext.x)
            {
                head.next = Next.next;
                Next.next = null;

                // Update NextNext for the next iteration
                Next = NextNext;
                NextNext = NextNext.next;
            }
        }

        // if horizontal
        else if (head.y == Next.y)
        {
```

```
            // find middle nodes with same value as y and
            // delete them
            while (NextNext != null && Next.y == NextNext.y)
            {
                head.next = Next.next;
                Next.next = null;

                // Update NextNext for the next iteration
                Next = NextNext;
                NextNext = NextNext.next;
            }
        }

        // Adjacent points should have same x or same y
        else
        {
            System.out.println("Given list is not valid");
            return null;
        }

        // recur for other segment

        // temporarily store the head and move head forward.
        Node temp = head;
        head = head.next;

        // call deleteMiddle() for next segment
        this.deleteMiddle();

        // restore head
        head = temp;

        // return the head
        return head;
    }

    /*  Given a reference (pointer to pointer) to the head
        of a list and an int, push a new node on the front
        of the list. */
    void push(int x, int y)
    {
        /* 1 & 2: Allocate the Node &
                  Put in the data*/
        Node new_node = new Node(x,y);

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

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


    void printList()
    {
        Node temp = head;
        while (temp != null)
        {
            System.out.print("("+temp.x+","+temp.y+")->");
            temp = temp.next;
        }
        System.out.println();
    }


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

        llist.push(40,5);
        llist.push(20,5);
        llist.push(10,5);
```

```
        llist.push(10,8);
        llist.push(10,10);
        llist.push(3,10);
        llist.push(1,10);
        llist.push(0,10);

        System.out.println("Given list");
        llist.printList();

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

Python

```
# Python program to remove middle points in a linked list of
# line segments,
class LinkedList(object):
    def __init__(self):
        self.head = None

    # Linked list Node
    class Node(object):
        def __init__(self, x, y):
            self.x = x
            self.y = y
            self.next = None

    # This function deletes middle nodes in a sequence of
    # horizontal and vertical line segments represented
    # by linked list.
    def deleteMiddle(self):
        # If only one node or no node...Return back
        if self.head == None or self.head.next == None or self.head.next.next == None:
            return self.head
        Next = self.head.next
        NextNext = Next.next
        # check if this is vertical or horizontal line
        if self.head.x == Next.x:
            # Find middle nodes with same value as x and
            # delete them.
            while NextNext != None and Next.x == NextNext.x:
                self.head.next = Next.next
                Next.next = None
                # Update NextNext for the next iteration
                Next = NextNext
                NextNext = NextNext.next
        elif self.head.y == Next.y:
            # find middle nodes with same value as y and
            # delete them
            while NextNext != None and Next.y == NextNext.y:
                self.head.next = Next.next
                Next.next = None
                # Update NextNext for the next iteration
                Next = NextNext
                NextNext = NextNext.next
        else:
            # Adjacent points should have same x or same y
            print "Given list is not valid"
            return None
        # recur for other segment
        # temporarily store the head and move head forward.
        temp = self.head
        self.head = self.head.next
        # call deleteMiddle() for next segment
```

```
        self.deleteMiddle()
        # restore head
        self.head = temp
        # return the head
        return self.head

    # Given a reference (pointer to pointer) to the head
    # of a list and an int, push a new node on the front
    # of the list.
    def push(self, x, y):
        # 1 & 2: Allocate the Node &
        # Put in the data
        new_node = self.Node(x, y)
        # 3. Make next of new Node as head
        new_node.next = self.head
        # 4. Move the head to point to new Node
        self.head = new_node

    def printList(self):
        temp = self.head
        while temp != None:
            print "(" + str(temp.x) + "," + str(temp.y) + ")->",
            temp = temp.next
        print ''

# Driver program
llist = LinkedList()
llist.push(40,5)
llist.push(20,5)
llist.push(10,5)
llist.push(10,8)
llist.push(10,10)
llist.push(3,10)
llist.push(1,10)
llist.push(0,10)

print "Given list"
llist.printList()

if llist.deleteMiddle() != None:
    print "Modified Linked List is"
    llist.printList()

# This code is contributed by BHAVYA JAIN
```

Output:

```
Given Linked List:
(0,10)-> (1,10)-> (3,10)-> (10,10)-> (10,8)-> (10,5)-> (20,5)-> (40,5)->
Modified Linked List:
(0,10)-> (10,10)-> (10,5)-> (40,5)->
```

Time Complexity of the above solution is O(n) where n is number of nodes in given linked list.

Exercise:

The above code is recursive, write an iterative code for the same problem.