GeeksMan
Linked List
Lesson 7

Delete without head pointer

You are given a pointer/ reference to the node which is to be deleted from the linked list of **N** nodes. The task is to delete the node. Pointer/ reference to head node is not given.

Note: No head reference is given to you. It is guaranteed that the node to be deleted is not a tail node in the linked list.

Expected Time Complexity : O(1)

Expected Auxilliary Space : O(1)

Constraints:

1 <= N <= 103

```
void deleteNode(Node *node)
{
   node->data = node->next->data;
   node->next = node->next->next;
  // Your code here
}
```

Delete-without-head-pointer

Polynomial Addition

Given two polynomial numbers represented by a linked list. The task is to complete the function **addPolynomial()** that adds these lists meaning adds the coefficients who have the same variable powers.

Note: Given polynomials are sorted in decreasing order of power.

Example 1:

Input:
LinkedList1: (1,x2)
LinkedList2: (1,x3)
Output:
1x^3 + 1x^2
Note: Try to solve the question without using any extra space.

Expected Time Complexity: O(N+M)

Expected Auxiliary Space: O(1)

Constraints:

1 <= N, M <= 105

1 <= x, y <= 106

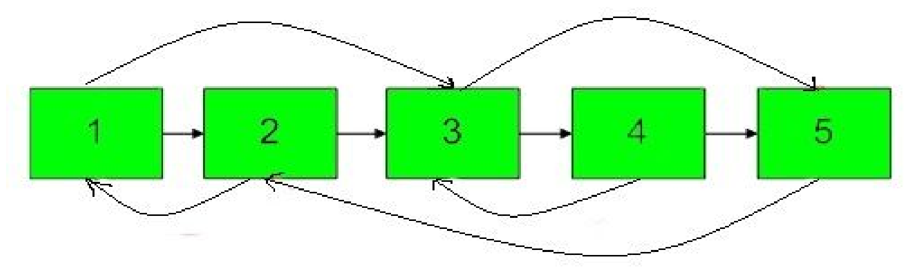
```
Node* addPolynomial(Node *p1, Node *p2)
{
  Node *res,*temp=new Node(0,0);
  res=temp;
  while(p1!=NULL && p2!=NULL)
  {
     if(p1->pow==p2->pow)
     {
```

```
                p1->coeff+=p2->coeff;
                temp->next=p1;
                p1=p1->next;
                p2=p2->next;
            }
            else if(p1->pow>p2->pow)
            {
                temp->next=p1;
                p1=p1->next;
            }
            else
            {
                temp->next=p2;
                p2=p2->next;
            }
            temp=temp->next;
        }
        if(p1)
        temp->next=p1;
        else
        temp->next=p2;
        return res->next;
        //Your code here
    }
```

polynomial-addition=
Can also be done with map O(N) space complexity.

Clone a linked list with next and random pointer

You are given a special linked list with **N** nodes where each node has a next pointer pointing to its next node. You are also given **M** random pointers , where you will be given **M** number of pairs denoting two nodes **a** and **b** **i.e.** a->arb = b.



Example 1:

Input:
N = 4, M = 2
value = {1,2,3,4}
pairs = {{1,2},{2,4}}
Output: 1
NOTE : If their is any node whose arbitrary pointer is not given then its by default null.

Expected Time Complexity : O(n)

Expected Auxilliary Space : O(1)

Constraints:

1 <= N <= 100

1 <= M <= N

1 <= a, b <= 100

clone-a-linked-list-with-next-and-random-pointer=

1)*Using Hashing*

```
Node *copyList(Node *head) {
   unordered_map<Node *,Node *>mp;
   Node *h=head,*p,*rhd;
   int i=0;
   while(head)
   {
       Node *n= new Node(head->data);
       mp[head]=n;
       if(i==0)
       {
           p=n;
           rhd=p;
           i++;
       }
       else
       {
           p->next=n;
           p=p->next;
       }
       head=head->next;
   }
   head=h;
   Node *l1,*l2;
   while(head)
   {
       l1=mp[head];
```

```
        l2=mp[head->arb];
        l1->arb=l2;
        head=head->next;

    }

    return rhd;

}
```

2)Method 2 (Uses Constant Extra Space)

```
Node *copyList(Node *head)
{
    Node *curr = head;
    while(curr != NULL)
    {
        Node *nextn = curr->next;
        curr->next = new Node(curr->data);
        curr->next->next = nextn;
        curr = nextn;
    }
    curr = head;
    Node*temp;
    while (curr)
    {
        if(curr->next)
        {
            if(curr->arb)
```

```
                {
                    curr->next->arb =  curr->arb->next;
                }
                else
                    curr->next->arb = curr->arb;

                curr=curr->next->next;
            }
            else
               curr = curr->next;
        }
        Node* new_head= head->next;

        curr = new_head;

        while (head&&new_head)
        {
            if(head->next)
            {
                head->next= head->next->next ;
            }
            if(curr->next)
            {
                curr->next =curr->next->next;
            }
            head = head->next;
            curr = curr->next;
        }
        return new_head;
    }
```

Delete nodes having greater value on right

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

Example 1:

Input:
LinkedList = 12->15->10->11->5->6->2->3
Output: 15 11 6 3
Example 2:

Input:
LinkedList = 10->20->30->40->50->60
Output: 60

Expected Time Complexity: O(N)

Expected Auxiliary Space: O(1)

Constraints:

1 <= size of linked list <= 1000

1 <= element of linked list <= 1000

Note: Try to solve the problem without using any extra space.

Delete-nodes-having-greater-value-on-right
1)Simple loop method

2)Reverse:

```
Node*reverse(Node*head)
{
   Node*prev=NULL,*curr=head,*nextn;
   while(curr)
   {
      nextn=curr->next;
      curr->next=prev;
      prev=curr;
      curr=nextn;
   }
   return prev;
}
Node *compute(Node *head)
{
   Node*nhead=reverse(head);
   Node*temp=nhead,*ptr=temp;
   int max=temp->data;
   while(temp)
   {
      if(temp->data<max)
      {
         ptr->next=temp->next;
         temp=temp->next;
      }
      else
      {
         max=temp->data;
         ptr=temp;
         temp=temp->next;
```

```
            }
        }
        return reverse(nhead);
    }
```

3)recursion :

```
Node *compute(Node *head)
{

if(head==NULL || head->next==NULL)
return head;
Node *t=compute(head->next);
if(head->data<t->data)
{
    return t;
    }
else
    {
        head->next=t;
        return head;
    }
 }
```

XOR Linked List

An ordinary Doubly Linked List requires space for two address fields to store the addresses of previous and next nodes. A memory efficient version of Doubly Linked List can be created using only one space for address field with every node. This memory efficient Doubly Linked List is called XOR Linked List or Memory Efficient as the list uses bit-wise XOR operation to save space for one address.

Given stream of data of size **N** for the linked list, your task is to complete the function **insert()** and **printList()**. The **insert()** function pushes (or inserts at beginning) the given data in the linked list and the **printList()** function returns the linked list as a list.

Note: There is an utility function XOR() that takes two Node pointer to get the bit-wise XOR of the two Node pointer. Use this function to get the XOR of the two pointers.

Example 1:

```
Input:
LinkedList: 9<->5<->4<->7<->3<->10
Output:
10 3 7 4 5 9
9 5 4 7 3 10
```

Example 2:

```
Input:
LinkedList: 58<->96<->31
Output:
31 96 58
58 96 31
```

Expected Time Complexity: O(N)

Expected Auxiliary Space: O(1)

Constraints:

1 <= N <= 100

```
struct Node* insert(struct Node *head, int data)
{
  if(head==NULL)
  {
     Node *newnode = new Node(data);
     head=newnode;
     newnode->npx = NULL;
     return head;
  }
  else
  {
     Node *newnode = new Node(data);
     newnode->npx = head;
     if(head->npx == NULL)
     {
        head->npx=newnode;
     }
     else
     {
        head->npx=XOR(newnode ,head->npx);
     }
     head=newnode;
     return head;
  }
```

```
            // Code here
}

vector<int> printList (struct Node *head)
{
   vector<int>v;
   Node *curr = head;
   Node *prev = NULL;
   Node *next;


   while (curr != NULL)
   {
              v.push_back(curr->data);

              next = XOR (prev, curr->npx);

              prev = curr;
       curr = next;
   }
   return v;
          // Code here
}

```

Subtract Two Numbers represented as Linked Lists

Given two linked lists that represent two large positive numbers. Subtract the smaller number from the larger one and return the difference as a linked list. Note that the input lists may be in any order, but we always need to subtract smaller from the larger one.

```
Node* subLinkedList(Node* l1, Node* l2)
{
    while(l1 && l1->data==0)
  {
      l1=l1->next;
  }
    while(l2 && l2->data==0)
  {
      l2=l2->next;
  }
int n=0,m=0;
Node *f=l1,*s=l2;
while(f!=NULL)
{
    n++;
    f=f->next;
}
 while(s!=NULL)
{
    m++;
    s=s->next;
}
// cout<<n<<m<<endl;
```

```
f=l1;
s=l2;
Node *first,*second;
if(m==n)
{
    while(f)
    {
        if(f->data==s->data)
        {
            f=f->next;
            s=s->next;
        }
        else
        {
            if(f->data>s->data)
            {
                first=l1;
                second=l2;
            }
            else
            {
                first=l2;
                second=l1;
            }
            break;
        }
    }
}
else if(m>n)
{
    first=l2;
    second=l1;
}
else
{
      first=l1;
    second=l2;
}
//reversinf first and second
Node *p=NULL,*c=first,*nx;
 while(c!=NULL)
 {
     nx=c->next;
     c->next=p;
     p=c;
     c=nx;

 }
 l1=p;

 p=NULL;
 c=second;
 while(c!=NULL)
 {
     nx=c->next;
     c->next=p;
     p=c;
     c=nx;

 }
```

```

l2=p;
c=l1;
while(l2)
{
    if(l1->data-l2->data<0)
    {
        l1->next->data--;
        l1->data=l1->data+10-l2->data;
    }
    else
    {
        l1->data-=l2->data;

    }
    l1=l1->next;
    l2=l2->next;



     }

     if(l1 && l1->data<0)
     {
         l1->next->data--;
             l1->data=l1->data+10;
     }
     p=NULL;
        while(c!=NULL)
     {
         nx=c->next;
         c->next=p;
         p=c;
         c=nx;

     }
     while(p->next && p->data==0)
     {
         p=p->next;
     }
    return p;

 }
```