1. Count nodes of linked list

class Solution

{

public:

//Function to count nodes of a linked list.

int getCount(struct Node\* head){

int count = 0;

while(head!=NULL)

{

count++;

head=head->next;

}

return count;

}

};

1. Linked list insertion

class Solution{

public:

//Function to insert a node at the beginning of the linked list.

Node \*insertAtBegining(Node \*head, int x) {

Node \*num = new Node(x);

num->next = head;

return num;

}

//Function to insert a node at the end of the linked list.

Node \*insertAtEnd(Node \*head, int x) {

Node \*res = new Node(x);

if(head==NULL)

return res;

Node \*cpy = head;

while(head->next!=NULL)

{

head=head->next;

}

head->next=res;

return cpy;

}

};

1. Doubly liked list insertion at given position

void addNode(Node \*head, int pos, int data)

{

Node\* res = new Node(data);

while(pos--)

{

head = head->next;

}

if(head->next==NULL)

{

head->next=res;

res->prev=head;

}

else

{

res->prev=head;

res->next=head->next;

head->next->prev=res;

head->next=res;

}

}

1. Insert in middle of linked list

Node\* insertInMiddle(Node\* head, int x)

{

Node\* end=head, \*mid = head;

Node\* res = new Node(x);

if(head->next==NULL)

{

head->next=res;

return head;

}

while(end->next!=NULL && end->next->next!=NULL)

{

mid=mid->next;

end=end->next->next;

}

res->next=mid->next;

mid->next=res;

return head;

}

1. Identical Linked lists

bool areIdentical(struct Node \*head1, struct Node \*head2)

{

while(head1!=NULL && head2!=NULL)

{

if(head1->data!=head2->data)

return false;

head1=head1->next;

head2=head2->next;

}

if(head1==NULL && head2==NULL)

return true;

else

return false;

}

1. Delete without head pointer

class Solution

{

public:

//Function to delete a node without any reference to head pointer.

void deleteNode(Node \*del)

{

while(del->next->next!=NULL)

{

del->data = del->next->data;

del=del->next;

}

del->data = del->next->data;

del->next=NULL;

}

};

1. Remove duplicates from sorted linked list

Node \*removeDuplicates(Node \*head)

{

if(head->next==NULL) return head;

Node \*tail = head;

while(tail->next!=NULL)

{

while(tail->next!=NULL && tail->data==tail->next->data)

{

tail->next=tail->next->next;

}

if(tail->next==NULL) break;

tail=tail->next;

}

return head;

}

1. Remove duplicates from an unsorted linked list

class Solution

{

public:

//Function to remove duplicates from unsorted linked list.

Node \* removeDuplicates( Node \*head)

{

set<int> s;

Node \*tail = head;

s.insert(tail->data);

if(tail->next==NULL) return head;

while(tail->next->next!=NULL)

{

if(s.find(tail->next->data)!=s.end())

{

tail->next=tail->next->next;

}

else

{

tail=tail->next;

s.insert(tail->data);

}

}

if(s.find(tail->next->data)!=s.end())

{

tail->next=tail->next->next;

}

return head;

}

};

1. Merge two sorted linked lists

Node\* sortedMerge(Node\* head1, Node\* head2)

{

Node \*tail1=head1, \*tail2=head2, \*head;

if(head1->data>head2->data)

{

head = head2;

tail2=tail2->next;

}

else

{

head=head1;

tail1=tail1->next;

}

Node \*tail=head;

while(tail1!=NULL && tail2!=NULL)

{

if(tail1->data>tail2->data)

{

tail->next=tail2;

tail2=tail2->next;

}

else

{

tail->next=tail1;

tail1=tail1->next;

}

tail=tail->next;

}

if(tail1!=NULL) tail->next = tail1;

else tail->next = tail2;

return head;

}

1. Nth node from end of liked list

int getNthFromLast(Node \*head, int n)

{

Node \*start = head;

while(n--)

{

if(head==NULL) return -1;

head=head->next;

}

while(head!=NULL)

{

head=head->next;

start=start->next;

}

return start->data;

}

1. Swap kth nodes from ends

Node \*swapkthnode(Node\* head, int num, int k)

{

if(k>num) return head;

Node \*f = head, \*s=head, \*cpy=head;

if(k==1||k==num)

{

while(s->next->next!=NULL)

s=s->next;

s->next->next=f->next;

head = s->next;

s->next= f;

f->next=NULL;

return head;

}

int k2 = num + 1 - k-2;

k=k-2;

while(k--)

f=f->next;

while(k2--)

s=s->next;

Node \*f1 = f->next, \*s1 = s->next;

if(f1==s)

{

s->next = s1->next;

s1->next = s;

f->next = s1;

}

else if(f==s1)

{

f->next = f1->next;

f1->next = f;

s->next = f1;

}

else

{

f->next = f1->next;

s->next = s1->next;

f1->next = s->next;

s1->next = f->next;

f->next=s1;

s->next=f1;

}

return head;

}

1. Reverse a linked list

class Solution

{

public:

//Function to reverse a linked list.

struct Node\* reverseList(struct Node \*head)

{

Node \*prev = NULL, \*nex = head->next;

while(nex!=NULL)

{

head->next = prev;

prev = head;

head=nex;

nex=nex->next;

}

head->next=prev;

return head;

}

};

1. Detect loop in linked list

class Solution

{

public:

//Function to check if the linked list has a loop.

bool detectLoop(Node\* head)

{

Node \*s = head, \*f = head;

while(f!=NULL && f->next!=NULL)

{

f = f->next->next;

s = s->next;

if(f==s) return true;

}

return false;

}

};

1. Find length of loop

int countNodesinLoop(struct Node \*head)

{

Node \*s = head, \*f = head;

while(f!=NULL && f->next!=NULL)

{

f = f->next->next;

s = s->next;

if(f==s) break;

}

int t = 0;

while(f!=NULL && f->next!=NULL)

{

f = f->next;

t++;

if(f==s) break;

}

return t;

}

1. Remove loop in linked list

class Solution

{

public:

//Function to remove a loop in the linked list.

void removeLoop(Node\* head)

{

Node \*s = head, \*f = head;

while(f!=NULL && f->next!=NULL)

{

f = f->next->next;

s = s->next;

if(f==s) break;

}

if(f==NULL ||f->next==NULL) return;

s=head;

while(f!=s)

{

f = f->next;

s=s->next;

}

while(f->next!=s)

f=f->next;

f->next=NULL;

return;

}

1. Rotate a linked list

class Solution

{

public:

//Function to rotate a linked list.

Node\* rotate(Node\* head, int k)

{

Node \*tail = head;

while(tail->next!=NULL)

tail = tail->next;

tail->next = head;

while(k--)

{

head=head->next;

tail=tail->next;

}

tail->next=NULL;

return head;

}

};

1. Add two numbers represented by linked lists

class Solution

{

public:

//Function to add two numbers represented by linked list.

struct Node\* reverseList(struct Node \*head)

{

Node \*prev = NULL, \*nex = head->next;

while(nex!=NULL)

{

head->next = prev;

prev = head;

head=nex;

nex=nex->next;

}

head->next=prev;

return head;

}

struct Node\* addTwoLists(struct Node\* first, struct Node\* second)

{

first = reverseList(first), second = reverseList(second);

Node \*f=first, \*s=second, \*head, \*tail;

int carry = 0;

while(first!=NULL && second!=NULL)

{

int t = first->data+second->data+carry;

first->data = t%10;

second->data = t%10;

carry = t/10;

first = first->next;

second = second ->next;

}

if(first==NULL)

{

while(second!=NULL)

{

int t = second->data + carry;

second->data=t%10;

carry=t/10;

second=second->next;

}

head = s;

}

else

{

while(first!=NULL)

{

int t = first->data + carry;

first->data=t%10;

carry=t/10;

first=first->next;

}

head = f;

}

tail =head;

while(tail->next!=NULL)

tail=tail->next;

if(carry)

{

Node\* add = new Node(1);

tail->next = add;

}

head = reverseList(head);

return head;

}

};

1. Pairwise swap of nodes in linked list

struct Node\* pairwise\_swap(struct Node\* head)

{

Node \*tail1 = head, \*tail2 = head->next;

tail1->next=tail2->next;

tail2->next=tail1;

head=tail2;

Node \*h = head;

while(tail1->next!=NULL && tail1->next->next!=NULL)

{

head=tail1;

tail1=tail1->next;

tail2=tail1->next;

tail1->next=tail2->next;

tail2->next=tail1;

head->next=tail2;

}

return h;

}

1. Check if linked list is palindrome

class Solution{

public:

//Function to check whether the list is palindrome.

bool isPalindrome(Node \*head)

{

Node \*head2 = new Node(head->data), \*cpy=head->next;

while(cpy!=NULL)

{

Node \*x = new Node(cpy->data);

x->next=head2;

head2=x;

cpy = cpy->next;

}

while(head!=NULL)

{

if(head->data!=head2->data)

return false;

head=head->next;

head2=head2->next;

}

return true;

}

};

1. Merge sort for linked list

class Solution{

public:

//Function to sort the given linked list using Merge Sort.

Node\* Merge(Node\* head1, Node\* head2)

{

Node \*tail1=head1, \*tail2=head2, \*head;

if(head1->data>head2->data)

{

head = head2;

tail2=tail2->next;

}

else

{

head=head1;

tail1=tail1->next;

}

Node \*tail=head;

while(tail1!=NULL && tail2!=NULL)

{

if(tail1->data>tail2->data)

{

tail->next=tail2;

tail2=tail2->next;

}

else

{

tail->next=tail1;

tail1=tail1->next;

}

tail=tail->next;

}

if(tail1!=NULL) tail->next = tail1;

else tail->next = tail2;

return head;

}

Node\* mergeSort(Node\* head) {

int l=0;

if(head->next==NULL) return head;

Node \*slow=head, \*fast=head;

while(fast->next!=NULL && fast->next->next!=NULL)

{

slow=slow->next;

fast=fast->next->next;

}

fast = slow->next;

slow->next=NULL;

head = mergeSort(head);

fast = mergeSort(fast);

return Merge(head, fast);

}

};

1. Given a linked list of 0s, 1s and 2s, sort it

class Solution

{

public:

//Function to sort a linked list of 0s, 1s and 2s.

Node\* segregate(Node \*head) {

if(head->next==NULL)

return head;

Node \*tail = head, \*p = head, \*cpy = head;

while(tail->next!=NULL)

tail=tail->next;

while(p->next!=tail)

{

if(p->next->data==0)

{

Node \*temp = p->next;

p->next = p->next->next;

temp->next = head;

head = temp;

}

else if(p->next->data==2)

{

Node \*temp = p->next;

p->next = p->next->next;

temp->next = tail->next;

tail->next = temp;

}

else

p = p->next;

}

if(cpy==head && cpy->data==2)

{

head = head->next;

cpy->next=tail->next;

tail->next=cpy;

}

else if(cpy->data==2)

{

Node \*temp=head;

while(temp->next!=cpy)

temp=temp->next;

temp->next=cpy->next;

cpy->next=tail->next;

tail->next=cpy;

}

if(tail->data==0)

{

Node \*temp=cpy;

while(temp->next!=tail)

temp=temp->next;

temp->next = tail->next;

tail->next = head;

head = tail;

}

return head;

}

};

1. Merge sort on double li ked list

struct Node \*sortDoubly(struct Node \*head)

{

head = mergeSort(head); //from Q20

Node \*cpy = head, \*tail = head->next;

head->prev = NULL;

while(tail!=NULL)

{

tail->prev=head;

head=head->next;

tail=tail->next;

}

return cpy;

}

1. Merge k sorted linked lists

class Solution{

public:

//Function to merge K sorted linked list.

Node\* merge(Node\* head1, Node\* head2)

{

if(head1==NULL) return head2;

Node \*tail1=head1, \*tail2=head2, \*head;

if(head1->data>head2->data)

{

head = head2;

tail2=tail2->next;

}

else

{

head=head1;

tail1=tail1->next;

}

Node \*tail=head;

while(tail1!=NULL && tail2!=NULL)

{

if(tail1->data>tail2->data)

{

tail->next=tail2;

tail2=tail2->next;

}

else

{

tail->next=tail1;

tail1=tail1->next;

}

tail=tail->next;

}

if(tail1!=NULL) tail->next = tail1;

else tail->next = tail2;

return head;

}

Node \* mergeKLists(Node \*arr[], int K)

{

Node \*ans = arr[0];

for(int i=1; i<K; i++)

{

ans = merge(ans, arr[i]);

}

return ans;

}

};

1. Intersection point of y shaped lists

int intersectPoint(Node\* head1, Node\* head2)

{

Node \*tail = head1->next, \*trash = new Node(-5);

while(tail!=NULL)

{

head1->next=trash;

head1=tail;

tail=tail->next;

}

head1->next=trash;

while(head2!=NULL)

{

if(head2->next==trash) return head2->data;

head2=head2->next;

}

return -1;

}

1. Clone a linked list with next and random pointer

class Solution

{

public:

Node \*copyList(Node \*head)

{

Node \*cpy=head;

while(cpy!=NULL)

{

Node \*t = new Node(cpy->data);

t->next = cpy->next;

cpy->next = t;

cpy = cpy->next->next;

}

cpy = head;

/\*while(cpy!=NULL)

{

cout<<cpy->data<<" ";

cpy=cpy->next;

}\*/

while(cpy!=NULL)

{

if(cpy->arb==NULL)

cpy->next->arb=NULL;

else

cpy->next->arb = cpy->arb->next;

cpy=cpy->next->next;

}

Node \*tail=head->next, \*head2 = head->next;

cpy = head;

while(cpy!=NULL && tail!=NULL)

{

cpy->next=tail->next;

if(cpy->next==NULL)

{

tail->next=NULL;

break;

}

cpy = cpy->next;

tail->next = cpy->next;

tail = tail->next;

}

return head2;

}

};

1. Add two numbers by linked list(incomplete)

Node\* reverse(Node \*head)

{

Node \*prev = NULL, \*nex = head->next;

while(nex!=NULL)

{

head->next = prev;

prev = head;

head=nex;

nex=nex->next;

}

head->next=prev;

return head;

}

Node\* addSameSize(Node\* first, Node\* second, int\* carry)

{

first=reverse(first), second=reverse(second);

Node \*cpy=first;

//\*carry = 0;

while(first!=NULL && second!=NULL)

{

int t = first->data + second->data + \*carry;

first->data = t%10;

\*carry = t/10;

first = first->next;

second = second ->next;

}

return reverse(cpy);

}

//This function is called after the smaller list is added to the sublist of

//bigger list of same size. Once the right sublist is added, the carry

//must be added to left side of larger list to get the final result.

void addCarryToRemaining(Node\* head1, Node\* curr, int\* carry, Node\*\* result)

{

Node \*cpy=\*result;

while(cpy->next!=NULL)

cpy=cpy->next;

head1 = reverse(head1);

Node \*cpy2 = head1;

while(cpy2!=NULL && carry)

{

int t = cpy2->data+\*carry;

cpy2->data = t%10;

\*carry = t/10;

cpy2 = cpy2->next;

}

cpy2 = head1;

head1=reverse(head1);

cpy2->next=cpy;

\*result = head1;

}