SDE SHEET STRIVER

DAY 6

1.Find intersection point of Y LinkedList

ANS. 1st approach (using two stacks and then traversing from back TS=O(m)+O(n) SC=O(m)+O(n)) :

ListNode \*getIntersectionNode(ListNode \*headA, ListNode \*headB) {

stack<ListNode \*> temp1;

stack<ListNode \*> temp2;

while (headA)

{

temp1.push(headA);

headA=headA->next;

}

while (headB)

{

temp2.push(headB);

headB=headB->next;

}

ListNode \*last=nullptr;

while (temp1.top()==temp2.top())

{

last=temp1.top();

if (!temp1.empty())

temp1.pop();

if (!temp2.empty())

temp2.pop();

if (temp1.empty() || temp2.empty())

break;

}

return last;

}

2nd approach (counting length and then traversing larger LL upto distance O(m)+O(m-n)+O(n)) :

int length(ListNode \*p)

{

int count=0;

while (p)

{

count++;

p=p->next;

}

return count;

}

ListNode \*getIntersectionNode(ListNode \*headA, ListNode \*headB) {

int m=length(headA);

int n=length(headB);

int d=n-m;

if (m>n)

{

ListNode \*temp=headA;

headA=headB;

headB=temp;

d=m-n;

}

int i=0;

for (i=0;i<d;i++)

{

headB=headB->next;

}

while (headA && headB)

{

if (headA==headB)

return headA;

headA=headA->next;

headB=headB->next;

}

return nullptr;

}

3rd approach ( O(m)+(n)+(m-n)) :

ListNode \*getIntersectionNode(ListNode \*headA, ListNode \*headB) {

ListNode \*d1=headA,\*d2=headB;

while (d1!=d2)

{

if (d1==nullptr)

d1=headB;

else

d1=d1->next;

if (d2==nullptr)

d2=headA;

else

d2=d2->next;

}

return d1;

}

2.Detect a Cycle in Linked List

ANS. 1st approach (Brute force : using hash table TC=O(N), SC=O(N)) :

2nd approach (using fast and slow pointer method O(n) ) :

bool hasCycle(ListNode \*head) {

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

while (fast)

{

fast=fast->next;

slow=slow->next;

if (fast==nullptr)

return false;

else

fast=fast->next;

if (fast==slow)

return true;

}

return false;

}

2.1 Happy Number

ANS. 1st approach (applying same concept of fast and slow pointer):

bool isHappy(int n) {

int slow=n,fast=n;

do

{

slow=sum\_0f\_squares(slow);

fast=sum\_0f\_squares(fast);

fast=sum\_0f\_squares(fast);

}while(slow!=fast);

return (slow==1)?true:false;

}

return false;

else

q=q->next;

if (p==q)

return true;

}

return false;

}

3.Reverse Nodes in k-Group

ANS. 1st approach (O(n/k)) :

int length(ListNode \*p)

{

int count=0;

while (p)

{

count++;

p=p->next;

}

return count;

}

ListNode\* reverseKGroup(ListNode\* head, int k) {

if (head==nullptr || k==1) return head;

int count=length(head);

ListNode \*dummy=new ListNode();

dummy->next=head;

ListNode \*cur=dummy,\*nex=dummy,\*pre=dummy;

while (count>=k)

{

cur=pre->next;

nex=cur->next;

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

{

cur->next=nex->next;

nex->next=pre->next;

pre->next=nex;

nex=cur->next;

}

pre=cur;

count-=k;

}

return dummy->next;

}

2nd approach (Recursive) :

ListNode \*r=nullptr,\*q=node,\*p=head;

while (p!=node)

{

r=q;

q=p;

p=p->next;

q->next=r;

}

return q;

}

ListNode \*reverseKGroup(ListNode \*head, int k) {

// 1. Generate nodes for the current group;

ListNode \*cur = head;

for (int i = 0; i < k; ++i) {

if (!cur) return head;

cur = cur->next;

}

// 2. Reverse the nodes for the current group;

ListNode \*newHead = reverseBefore(head, cur);

// 3. Reverse the nodes for the next group;

head->next = reverseKGroup(cur, k);

return newHead;

}

4.Palindrome Linked List

ANS. 1st approach (reversing the LL and then cheking the elements TC=O(N)+O(N)+O(N), SC=O(N)) :

bool isPalindrome(ListNode\* head) {

ListNode \*p=head;

vector<int> temp;

while (p)

{

temp.push\_back(p->val);

p=p->next;

}

p=head;

ListNode \*q=nullptr,\*r=nullptr;

while (p)

{

r=q;

q=p;

p=p->next;

q->next=r;

}

ListNode \*newhead=q;

int i=0;

for (i=0;i<temp.size();i++)

{

if (temp[i]!=newhead->val)

return false;

newhead=newhead->next;

}

return true;

}

2nd approach (storing values in vector and checking that vector : TC=O(N)+O(N), SC=O(N)) :

bool isPalindrome(ListNode\* head) {

ListNode \*p=head;

vector<int> temp;

while (p)

{

temp.push\_back(p->val);

p=p->next;

}

int j=temp.size()-1;

for (int i=0;i<j;i++,j--)

{

if (temp[i]!=temp[j])

return false;

}

return true;

}

3rd approach (using stack and checking the values : TC-O(N),SC-O(N)) :

bool isPalindrome(ListNode\* head) {

stack<ListNode \*> st;

ListNode \*temp=head;

while (temp)

{

st.push(temp);

temp=temp->next;

}

while (!st.empty())

{

if (st.top()->val!=head->val)

return false;

st.pop();

head=head->next;

}

return true;

}

4rd approach (finding middle and reversing the LL from one rigth to the middle node and then traversing simultaneously : TC=O(N),SC=O(1)) :

ListNode\* reverse(ListNode \*head)

{

ListNode \*pre=nullptr,\*next=nullptr;

while (head!=nullptr)

{

next=head->next;

head->next=pre;

pre=head;

head=next;

}

return pre;

}

bool isPalindrome(ListNode\* head) {

if (head==nullptr || head->next==nullptr)

return true;

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

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

{

slow=slow->next;

fast=fast->next->next;

}

slow->next=reverse(slow->next);

slow=slow->next;

while (slow!=nullptr)

{

if (head->val!=slow->val)

return false;

head=head->next;

slow=slow->next;

}

return true;

}

5.Find the starting point of the Loop of LinkedList

ANS. 1st approach (uisng Hash table TC=O(N), SC=O(N)) :

Diagram, schematic

Description automatically generated

2nd approach (using slow and fast pointer (floyd tortoise method) O(n)) :

proof ()

ListNode \*detectCycle(ListNode \*head) {

if (head==nullptr)

return nullptr;

ListNode \*p=head,\*q=head;

do

{

p=p->next;

q=q->next;

if (q==nullptr || q->next==nullptr)

return nullptr;

else

q=q->next;

} while (p && q && p!=q);

p=head;

while (p!=q)

{

p=p->next;

q=q->next;

}

return q;

}

6.Flattening of a LinkedList

ANS. 1st approach (merging list one by one from right : O(all nodes)) :

Node\* Merge(Node \*a,Node \*b)

{

Node \*temp=new Node(0);

Node \*res=temp;

while (a && b)

{

if (a->data<b->data)

{

temp->bottom=a;

temp=temp->bottom;

a=a->bottom;

}

else

{

temp->bottom=b;

temp=temp->bottom;

b=b->bottom;

}

}

if (a){temp->bottom=a;}

else{temp->bottom=b;}

return res->bottom;

}

Node \*flatten(Node \*root)

{

if (root==nullptr || root->next==nullptr)

return root;

root->next=flatten(root->next);

root=Merge(root,root->next);

return root;

}

7.Rotate a LinkedList

ANS. 1st approach (Brute force : O(k\*n)) :

ListNode\* rotateRight(ListNode\* head, int k) {

if (head==nullptr ||head->next==nullptr)

return head;

for (int i=0;i<k;i++)

{

ListNode \*p=head,\*q=nullptr;

while (p->next!=nullptr)

{

q=p;

p=p->next;

}

p->next=head;

q->next=nullptr;

head=p;

}

return head;

}

2nd approach ( O(N\*K%N)+O(N)) :

int length(ListNode \*p)

{

int count=0;

while (p)

{

count++;

p=p->next;

}

return count;

}

ListNode\* rotateRight(ListNode\* head, int k) {

if (head==nullptr ||head->next==nullptr)

return head;

int len=length(head);

int rem=k%len;

for (int i=0;i<rem;i++)

{

ListNode \*p=head,\*q=nullptr;

while (p->next!=nullptr)

{

q=p;

p=p->next;

}

p->next=head;

q->next=nullptr;

head=p;

}

return head;

}

3rd approach ( O(N)+O(N-N%K)) :

ListNode\* rotateRight(ListNode\* head, int k) {

if (head==nullptr ||head->next==nullptr)

return head;

int len=1;

ListNode \*c=head,\*p=head;

while (c->next!=nullptr)

{

len++;

c=c->next;

}

c->next=head;

k=k%len;

k=len-k;

for (int i=0;i<k-1;i++)

{

p=p->next;

}

head=p->next;

p->next=nullptr;

return head;

}