Lecture 24 LinkedList

1. Insertion, Deletion, Searching

<https://ide.codingblocks.com/s/447315>

#include <iostream>

using namespace std;

class node{

public:

    int data;

    node \*next;

    node(int d)

    {

        data = d;

        next = NULL;

    }

};

void insertionAtHead(node\*&head, int data)

{

    if(head==NULL)

    {

        head = new node(data);

        return;

    }

    node \*n = new node(data);

    n -> next = head;

    head = n;

}

int length(node\*head)

{

    int len = 0;

    while(head!=NULL)

    {

        head = head->next;

        len++;

    }

    return len;

}

void insertionAtTail(node\*&head, int data)

{

    if(head==NULL)

    {

        head = new node(data);

        return;

    }

    node\*temp = head;

    while(temp->next!=NULL)

    {

        temp = temp->next;

    }

    temp->next = new node(data);

    return;

}

void insertionAtMid(node\*&head, int data, int p)

{

    if(head==NULL || p==0)

    {

        insertionAtHead(head, data);

    }

    else if(p>length(head))

    {

        insertionAtTail(head, data);

    }

    else

    {

        int jump = 1;

        node\*temp = head;

        while(jump <= p-1)

        {

            temp = temp->next;

            jump++;

        }

        node \*n = new node(data);

        n->next = temp->next;

        temp->next = n;

    }

}

void deleteAtHead(node\*&head)

{

    if(head==NULL)

    {

        return;

    }

    node\*temp = head;

    head = head->next;

    delete temp;

    return;

}

void deleteAtTail(node\*&head)

{

    node\*prev = NULL;

    node\*temp = head;

    while(temp->next!=NULL)

    {

        prev = temp;

        temp = temp->next;

    }

    delete temp;

    prev->next = NULL;

    return;

}

void print(node\*head)

{

    node \*temp = head;

    while(temp!=NULL)

    {

        cout<<temp->data<<"-->";

        temp = temp->next;

    }

    cout<<endl;

}

bool searchIterative(node\*head, int key)

{

    while(head!=NULL)

    {

        if(head->data==key)

        {

            return true;

        }

        head = head->next;

    }

    return false;

}

bool searchRecursive(node\*head, int key)

{

    // base case

    if(head==NULL)

    {

        return false;

    }

    if(head->data==key)

    {

        return true;

    }

    else

    {

        return searchRecursive(head->next, key);

    }

}

int main() {

    node\* head = NULL;

    insertionAtHead(head, 5);

    insertionAtHead(head, 1);

    insertionAtHead(head, 3);

    insertionAtMid(head, 6, 2);

    insertionAtTail(head, 4);

    deleteAtHead(head);

    deleteAtTail(head);

    if(searchIterative(head, 10))

    {

        cout<<"Present"<<endl;

    }

    else

    {

        cout<<"Not"<<endl;

    }

    if(searchRecursive(head, 6))

    {

        cout<<"Present"<<endl;

    }

    else

    {

        cout<<"Not"<<endl;

    }

    print(head);

    return 0;

}

1. <https://leetcode.com/problems/reverse-linked-list/>

class Solution {

public:

ListNode\* reverseList(ListNode\* head) {

ListNode \* curr = head;

ListNode \* prev = NULL;

ListNode \* N;

while(curr!=NULL)

{

N = curr->next;

curr->next = prev;

prev = curr;

curr = N;

}

return prev;

}

};

1. <https://leetcode.com/problems/middle-of-the-linked-list/>

class Solution {

public:

ListNode\* middleNode(ListNode\* head) {

ListNode\*fast = head;

ListNode\*slow = head;

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

{

fast = fast->next->next;

slow = slow->next;

}

return slow;

}

};

1. <https://leetcode.com/problems/merge-two-sorted-lists/>

class Solution {

public:

ListNode\* mergeTwoLists(ListNode\* l1, ListNode\* l2) {

if(l1==NULL)

{

return l2;

}

if(l2==NULL)

{

return l1;

}

ListNode \*l3;

if(l1->val <= l2->val)

{

l3=l1;

l3->next = mergeTwoLists(l1->next, l2);

}

else

{

l3=l2;

l3->next = mergeTwoLists(l1, l2->next);

}

return l3;

}

};