struct Node{ //构造链表Node结构

int val;

Node \*prev, \*next;

Node(int val): val(val), prev(nullptr), next(nullptr) {} //初始化

};

class MyLinkedList {

public:

MyLinkedList(): head(nullptr), tail(nullptr),size(0){} //初始化链表

int get(int index) { //通过getNode函数返回第index个节点的地址，return 该节点的值

if(getNode(index))

return getNode(index) -> val;

return -1;

}

void addAtHead(int val) { //在head添加节点

auto node = new Node(val); //创建一个Node实例，下同

++ size; //链表长度加一，下同

if(head == nullptr) //如果链表为空，新加的node既是head也是tail，下同

{

head = node;

tail = node;

}

else

{

node -> next = head; //常规的添加头节点步骤，参照代码后的图解

head -> prev = node;

head = node;

}

}

void addAtTail(int val) { //在tail添加节点

auto node = new Node(val);

++ size;

if(tail == nullptr)

{

head = node;

tail = node;

}

else

{

node -> prev = tail; //常规的添加尾节点步骤，参照代码后的图解

tail -> next = node;

tail = node;

}

}

void addAtIndex(int index, int val) {

if(index > size) return; //如果索引大于链表长度，无效索引

if(index == size) //若索引等于链表长度，相当于添加尾节点，直接调用先前定义好的函数

{

addAtTail(val);

return;

}

if(index <= 0) //若索引小于链表长度，本题题目的bug，我们需要将它看成添加头节点，直接调用先前定义好的函数

{

addAtHead(val);

return;

}

auto node = new Node(val); //添加在非head非tail的位置的情况

auto nextNode = getNode(index); //过程参照代码后的图解

nextNode -> prev -> next = node;

node -> prev = nextNode -> prev;

node -> next = nextNode;

nextNode-> prev = node;

++ size;

}

void deleteAtIndex(int index) { //删除节点

if(auto node = getNode(index)) //若该节点不为nullptr，进行以下步骤

{

if(node == head) //若该节点为head，指针head更新为原来head的下一个点的位置

{

head = head -> next;

if(head != nullptr) head -> prev = nullptr; //若新head不为nullptr，将head的prev指针设为空，删除的节点的next指针设为空，即两者断开。

node -> next = nullptr;

}

if(node == tail) //若该节点为tail，与上一步类似

{

tail = tail -> prev;

if(tail != nullptr) tail -> next = nullptr;

node -> prev = nullptr;

}

//若目标节点上或下的指针还不为nullptr，说明指针还未独立出来，需要做以下操作

if(node -> next != nullptr) node -> next -> prev = node -> prev;

if(node -> prev != nullptr) node -> prev -> next = node -> next;

delete node;

-- size;

}

}

private:

Node\* getNode(int index) //获得目标节点位置，因为是双向链表，通过判断目标点位置在前半段还是后半段来决定从head开始搜索还是从tail搜索

{

if(index >= size || index < 0) return nullptr;

Node\* node;

int i;

if(size/2 >= index)

{

i = index;

node = head;

while(i -- > 0)

{

node = node -> next;

}

}

else

{

i = size - index - 1;

node = tail;

while(i -- > 0)

node = node -> prev;

}

return node;

}

private:

Node\* head;

Node\* tail;

int size;

};