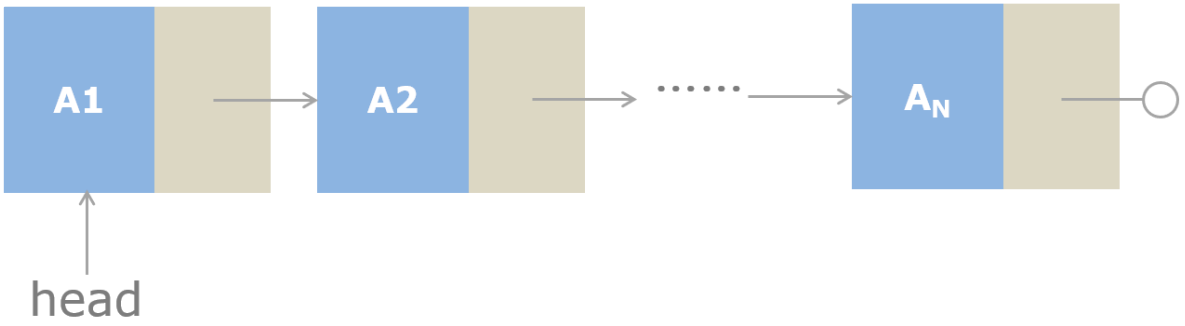
链表 Linked List

在编程问题中，有许多可以供我们选择的介质来储存数据。而本章将要介绍的是基础的链表类型。

简介

链表是一个特殊的表，每一个节点 Node 的地址都储存在前一个节点里，但第一个节点的位置是独立储存。因为这个特性，链表可以很容易地被遍历。



链表

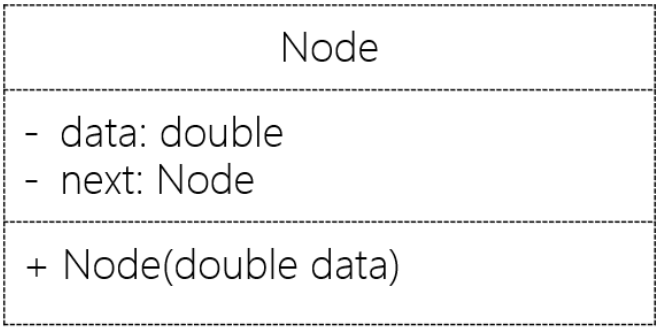
值得注意的是，**在插入数据的方面，链表比数组 array 更快；而在寻找元素的方面，数组比链表更快。**

操作	数组	链表
访问元素	$O(1)$	$O(n)$
增删元素	$O(n)$	$O(1)$

结构

Linked List Implementation

Class: Node



Setters and getters are not listed.

节点结构

Linked List Implementation

Class: List

List
- head: Node
+ List() + isEmpty(): boolean + insertNode(int index, double x): Node + findNode(double x): Node + removeNode(double x): Node + displayList(): void

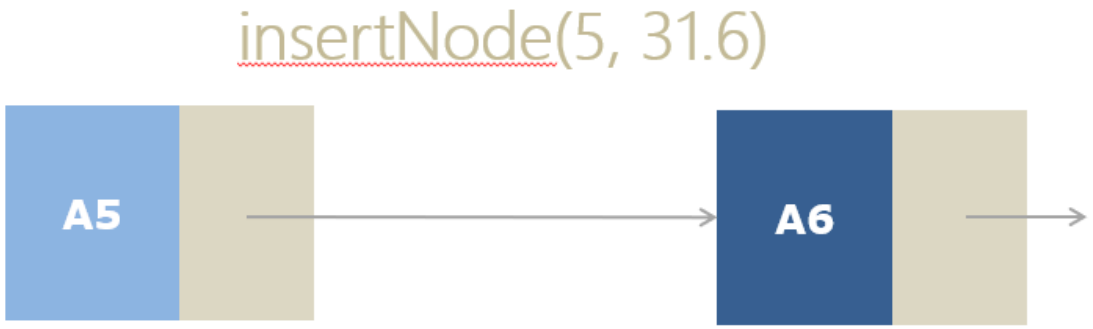
Setters and getters are not listed.

链表结构

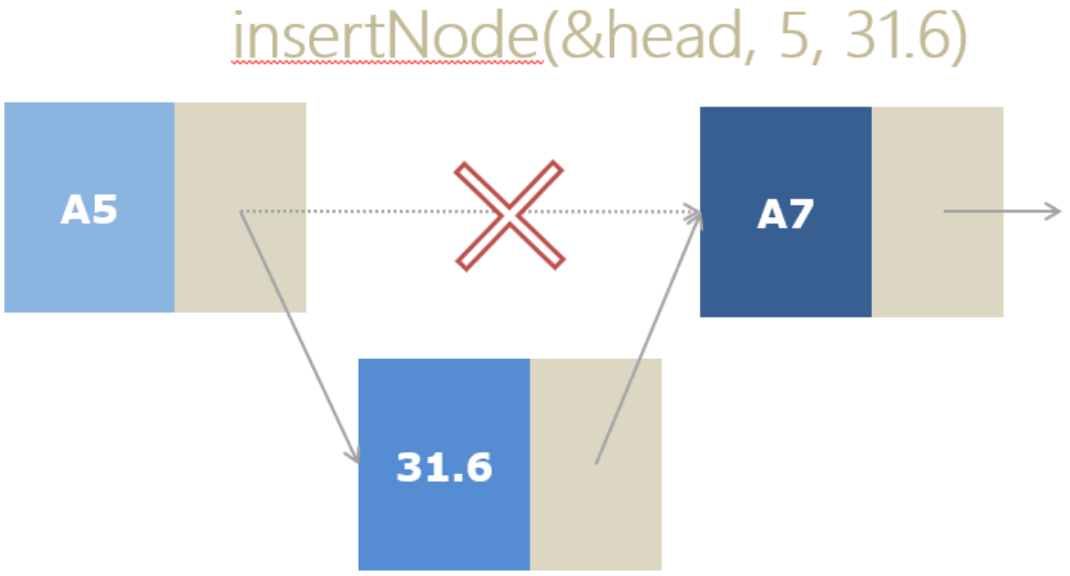
插入

当插入一个数据的时候拥有三种情况：

1. $index = 0$ 的时候，实际上是在创建开头的节点，因此可以直接新建。
2. $index > 0$，假设链表的第一个元素位置是0，则在此位置插入元素是从第 $index - 1$ 个元素的后面插入
   - 在链表的末尾插入元素，在创建节点后妥善安排连接于此节点的上一个节点的指针即可
   - 在链表的中间插入元素，在创建好元素后移除旧指针再次创建新指针即可



末尾插入



中间插入

寻找与删除

在删除元素后，需要完成的是删除旧指针与创立新指针。需要特别注意的是，查询和删除都是针对链表内第一个符合目标大小的元素而操作，如果表内有多个 3 ，而这两个操作都是针对第一个出现的 3 而负责。

代码实现

节点

```
public class Node {
    private double data;
    private Node next;

    public Node(double data) {
        this.data = data;
        this.next = null;
    }

    public double getData() {
        return this.data;
    }

    public Node getNext() {
        return this.next;
    }

    public void setNext(Node next) {
        this.next = next;
    }

}
```

链表

```
/*Add some comments by yourself.*/
public class List {

    private Node head;
    public List() {
```

```
        this.head = null;
    }
    public boolean isEmpty() {
        return this.head == null;
    }
    public Node insertNode(int index, double x) {
        if(index < 0)
            return null;
        int currIndex = 1;
        Node currNode = this.head;
        while(currNode != null && index > currIndex) {
            currNode = currNode.getNext();
            currIndex ++;
        }
        if(index > 0 && currNode == null)
            return null;

        Node newNode = new Node(x);
        if(index == 0) {
            newNode.setNext(this.head);
            this.head = newNode;
        }
        else {
            newNode.setNext(currNode.getNext());
            currNode.setNext(newNode);
        }
        return newNode;
    }
    public Node findNode(double x) {
        for(Node currNode = head; currNode != null; currNode =
currNode.getNext())
            if(currNode.getData() == x)
                return currNode;
        return null;
    }
    public Node removeNode(double x) {
        if(head == null)
            return null;
```

```
        Node currNode = head;
        if(head.getData() == x) {
            this.head = head.getNext();
            return currNode;
        }
        for(; currNode.getNext() != null; currNode =
currNode.getNext())
            if(currNode.getNext().getData() == x) {
                Node matchNode = currNode.getNext();
                currNode.setNext(matchNode.getNext());
                return matchNode;
            }
        return null;
    }
    public void displayList() {
        Node currNode = head;
        if(currNode != null) {
            System.out.print(currNode.getData());
            for(currNode = currNode.getNext(); currNode !=
null; currNode = currNode.getNext()) {
                System.out.print(" -> " + currNode.getData());
            }
        }
        System.out.println();
    }
}
```