顺序表

- 顺序表
    - 简单顺序表
    - 索引顺序表
        - 数据可以很不规则
        - 数据物理排列可以不要求
        - 索引的格式规整
    - 实现方式
        - 一体式
        - 分离式
        - 更改是否方便为准
    - 扩容问题
        - 每次定量增长：节省空间，操作频繁
        - 每次按比例增长：浪费
    - 操作：
        - 增加
            - 保序尾端插入
            - 非保序
            - 保序
        - 删除：和增加类似
    - python-list操作：
        - 分离技术实现的动态表
        - 空表：8个位置
        - 插入满： 扩大四倍
        - 如果已经很大（50000）：加一倍
- 链表
    - 分类
    - 单向链表： 案例ly01.py
- 链表相关操作
    - is_empty() 判断链表是否为空
    - length() 返回链表的长度
    - travel() 遍历
    - add(item) 在头部添加一个节点
    - append(item) 在尾部添加一个节点
    - insert(pos, item) 在指定位置pos添加节点
    - remove(item) 删除一个节点
    - search(item) 查找节点是否存在
- 链表 vs 顺序表
    - 访问元素： n, 1

- 头部插入： 1， n
- 尾部 : n,1
- 中间插入： n, n

单向列表

- 单位元素的定义
  - 一般定义为结构或者类
  - 必须包含至少两个内容：数据和指针

```
class SingleNode(object):
    def __init__(self, item):
        self.item = item

        self.next = None
```

- 遍历算法
  - 采取循环，一般用while
  - 只要下一个元素不为空就继续
- 求长度算法
  - 在head中存储长度
  - 每次求长度遍历一次

```
class SingleLinkList(object):

    def __init__(self):
        self._head = None



    def isEmpty(self):
        return self._head == None

    def length(self):
        cur = self._head

        count = 0

        while cur != None:
            count += 1
```

```
            cur = cur.next

        return count

    def travel(self):
        cur = self._head

        while cur:
            print(cur.item)
            cur = cur.next

        return None

    def addFirst(self, item):
        node = SingleNode(item)

        node.next = self._head

        self._head = node


    def append(self, item):
        node = SingleNode(item)

        if self.isEmpty():
            self._head = node

        else:
            cur = self._head
            while cur.next:
                cur = cur.next
            cur.next = node

    def insert(self, pos, item):
        if pos <= 0:
            self.addFirst(item)
        elif pos > (self.length()-1):
            self.append(item)
        else:
            node = SingleNode(item)
```

```
            count = 0

            pre = self._head
            while count<(pos-1):
                count += 1
                pre = pre.next

            node.next = pre.next
            pre.next = node

    def remove(self, item):
        cur = self._head
        pre = None

        while cur != None:
            if cur.item == item:
                if not pre:
                    self._head = cur.next
                else:
                    pre.next = cur.next

                break
            else:
                pre = cur
                cur = cur.next

    def search(self, item):
        cur = self._head
        while cur != None:
            if cur.item == item:
                return True
            cur = cur.next
        return False



if __name__ == "__main__":
    sll = SingleLinkList()
    sll.addFirst(10)
    sll.addFirst(20)
```

```
sll.append(30)
sll.insert(2,4)

print("Length of sll is {0}".format(sll.length()))

sll.travel()

print(sll.search(30))
print(sll.search(32))

sll.remove(20)
print("Length of sll is {0}".format(sll.length()))
sll.travel()
```

单向循环列表

- 案例ly02.py

```
class SingleCycLinkedlist(object):
    def __init__(self):
        self._head = None

    def is_empty(self):
        return self._head == None

    def length(self):
        if self.is_empty():
            return 0
        count = 1
        cur = self._head
        while cur.next != self._head:
            count += 1
            cur = cur.next
        return count

    def travel(self):
        if self.is_empty():
            return
        cur = self._head
        print(cur.item)
```

```
        while cur.next != self._head:
            cur = cur.next
            print(cur.item)
```

```
    def addFirst(self, item):
        node = SingleNode(item)
        if self.is_empty():
            self._head = node
            node.next = self._head
        else:
            node.next = self._head
            cur = self._head
            while cur.next != self._head:
                cur = cur.next
            cur.next = node
            self._head = node

    def append(self, item):
        node = SingleNode(item)
        if self.is_empty():
            self._head = node
            node.next = self._head
        else:
            cur = self._head
            while cur.next != self._head:
                cur = cur.next
            cur.next = node
            node.next = self._head

    def insert(self, pos, item):
        if pos <= 0:
            self.addFirst(item)
        elif pos > (self.length()-1):
            self.append(item)
        else:
            node = SingleNode(item)
            cur = self._head
            count = 0
            while count < (pos-1):
                count += 1
```

```
            cur = cur.next
        node.next = cur.next
        cur.next = node

    def remove(self, item):
        if self.is_empty():
            return
        cur = self._head
        pre = None
        if cur.item == item:
            if cur.next != self._head:
                while cur.next != self._head:
                    cur = cur.next
                cur.next = self._head.next
                self._head = self._head.next
            else:
                self._head = None
        else:
            pre = self._head
            while cur.next != self._head:
                if cur.item == item:
                    pre.next = cur.next
                    return
                else:
                    pre = cur
                    cur = cur.next
            if cur.item == item:
                pre.next = cur.next

    def search(self, item):
        if self.is_empty():
            return False
        cur = self._head
        if cur.item == item:
            return True
        while cur.next != self._head:
            cur = cur.next
            if cur.item == item:
                return True
        return False
```

```
if __name__ == "__main__":
    ll = SingleCycLinkedlist()
    ll.addFirst(1)
    ll.addFirst(2)
    ll.append(3)
    ll.insert(2, 4)
    ll.insert(4, 5)
    ll.insert(0, 6)
    print("length: {0}".format(ll.length()))
    ll.travel()
    print(ll.search(3))
    print(ll.search(7))
    ll.remove(1)
    print("length:",ll.length())
    ll.travel()
```

双向列表

- 案例ly03.py
- 单位元素定义

```
class Node(object):
    def __init__(self, item):
        self.item = item
        self.next = None
        self.prev = None
```

  - 代码实现

```
class DLinkList(object):
    def __init__(self):
        self._head = None

    def is_empty(self):
        return self._head == None

    def length(self):
        cur = self._head
```

```
        count = 0
        while cur != None:
            count += 1
            cur = cur.next
        return count

    def travel(self):
        cur = self._head
        while cur != None:
            print( cur.item)
            cur = cur.next

    def add(self, item):
        node = Node(item)
        if self.is_empty():
            self._head = node
        else:
            node.next = self._head
            self._head.prev = node
            self._head = node

    def append(self, item):
        node = Node(item)
        if self.is_empty():
            self._head = node
        else:
            cur = self._head
            while cur.next != None:
                cur = cur.next
            cur.next = node
            node.prev = cur
```

```
def search(self, item):
    cur = self._head
    while cur != None:
        if cur.item == item:
            return True
        cur = cur.next
    return False

def insert(self, pos, item):
    if pos <= 0:
        self.add(item)
    elif pos > (self.length() - 1):
```

```
            self.append(item)
        else:
            node = Node(item)
            cur = self._head
            count = 0
            # 移动到指定位置的前一个位置
            while count < (pos - 1):
                count += 1
                cur = cur.next
            # 将node的prev指向cur
            node.prev = cur
            # 将node的next指向cur的下一个节点
            node.next = cur.next
            # 将cur的下一个节点的prev指向node
            cur.next.prev = node
            # 将cur的next指向node
            cur.next = node

    def remove(self, item):
        if self.is_empty():
            return
        else:
            cur = self._head
            if cur.item == item:
                if cur.next == None:
                    self._head = None
                else:
                    cur.next.prev = None
                    self._head = cur.next
                return
            while cur != None:
                if cur.item == item:
                    cur.prev.next = cur.next
                    cur.next.prev = cur.prev
                    break
                cur = cur.next

if __name__ == "__main__":
    ll = DLinkList()
    ll.add(1)
```

```
ll.add(2)
ll.append(3)
ll.insert(2, 4)
ll.insert(4, 5)
ll.insert(0, 6)
print( "length: ", ll.length())
ll.travel()
print( ll.search(3))
print( ll.search(4))
ll.remove(1)
print( "length:", ll.length())
ll.travel()
```