第七次作业说明

0. 实现双向链表中用到的节点类 (Node)

- 成员变量
  - `data`: 该节点储存的数据
  - `next`: 该节点的下一个节点
  - `prev`: 该节点的前一个结点
  - 其它你认为需要的成员变量
- 方法（参数中省去了`self`）
  - `get_data()`: 获取节点当前的`data`
  - `get_next()`: 获取节点当前的`next`
  - `get_prev()`: 获取节点当前的`prev`
  - `set_data(new_data)`: 为节点设置一个新的`data`
  - `set_next(new_next)`: 为节点设置一个新的`next`
  - `set_prev(new_prev)`: 为节点设置一个新的`prev`
  - 其它你认为必要的方法

1. 实现一个无序链表 (UnorderedList)

使用上一步中创建好的`Node`类实现一个无序链表。

- 成员变量
  - `head`: 链表头元素
  - `tail`: 链表尾元素
  - 其它你认为需要的成员变量
- 方法（参数中省去了`self`）
  - `push_front(item)`: 在链表的头部插入一个元素
  - `push_back(item)`: 在链表的尾部插入一个元素
  - `insert(index, item)`: 在`index`的位置**前**插入`item`，插入后`item`的坐标应该是`index`，例如对于`[1, 2, 3, 4]`，`insert(2, 666)`后链表变为`[1, 2, 666, 3, 4]`
  - `size()`: 返回链表的长度，返回值为`int`类型
  - `exist(item)`: 寻找链表中是否有`item`，返回类型为`bool`类型
  - `remove(item)`: 删除链表中所有值等于`item`的元素，返回删除的数量（如果没有`item`这个元素则返回`0`），返回类型为`int`
  - `index(item)`: 查找链表中所有值等于`item`的元素的位置（从0开始计数），返回一个`int`的`list`
  - `pop(pos=-1)`: 删除`pos`位置的元素，`pos`应当有一个默认值，如果不填`pos`参数，默认删除最后一个元素。注意该方法有返回值，应当返回被删除的`item`本身
  - `__str__()`: 实现将你的无序列表转换成字符串的操作，和`python`中的`str(list)`的形式相同，形如`[1, 2, 3, 4]`
  - `__getitem__(index)`: 让你的无序列表能使用中括号访问（即`lst[2]`这样的形式），和`list`类似
  - 其他你认为必要的方法

2. 实现一个有序链表 (OrderedList)

`OrderedList`应当继承自`UnorderedList`，默认顺序为从小到大，保证每次插入后整个链表还是有序的。

- 方法
  - `exist(item)`: 寻找链表中是否有`item`，返回类型为`bool`类型。由于此时链表是有序的，可以对该方法进行优化，请实现一个优化后的算法。
  - `add(item)`: 在链表中插入一个元素，返回插入后该元素的坐标，返回类型为`int`。
  - 注意，此时父类`OrderedList`的`push_back`和`push_front`方法其实已经没用了，但仍然可以被调用，我们不会测试这个事情，但是你可以手动把这两个方法重定向到你新实现的`add`方法上。

3. 用UnorderedList实现一个Stack类

注意`pop()`应当返回被弹出的元素本身。

4. 用UnorderedList实现一个Queue类

注意`dequeue()`应当返回出队的元素本身。