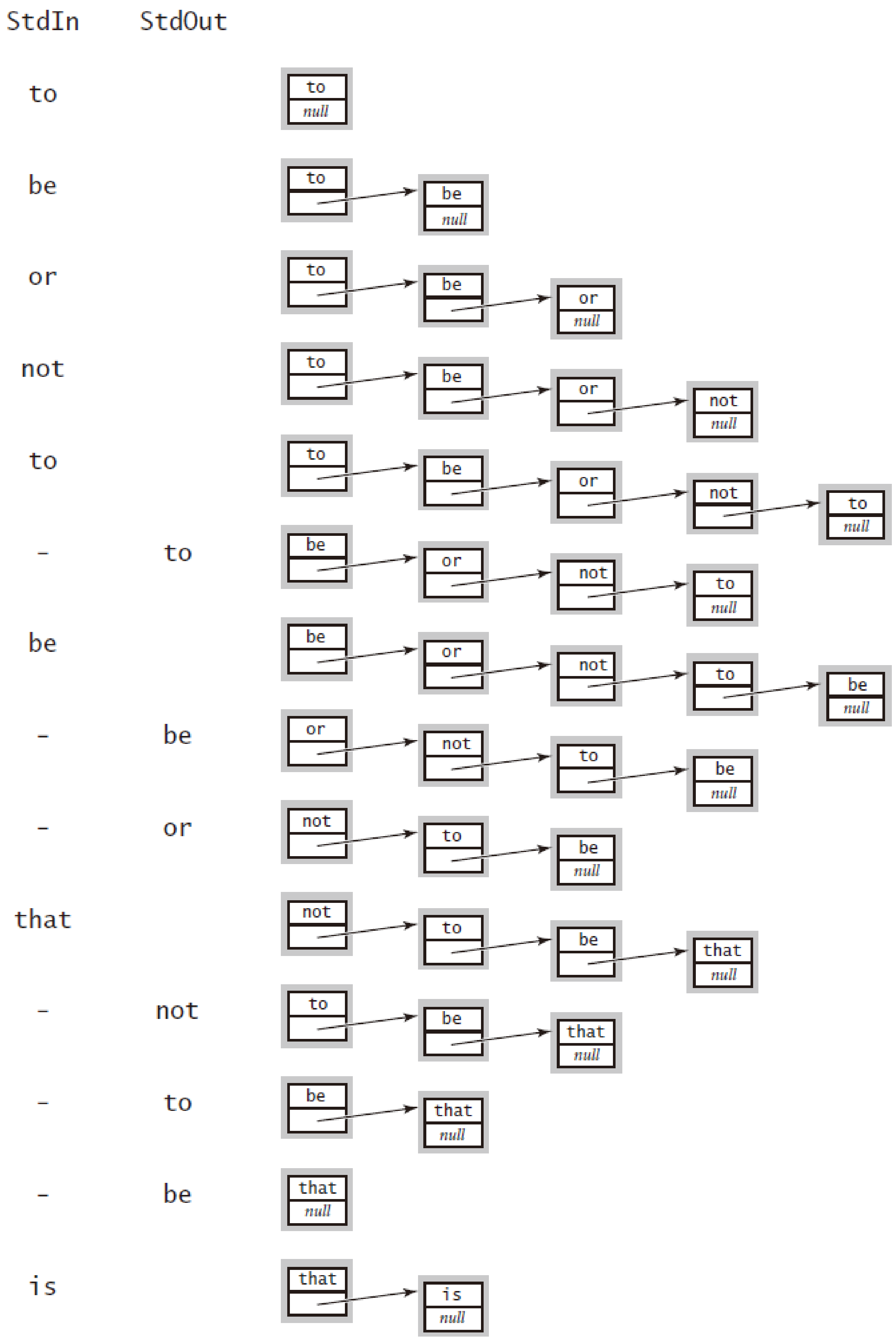
> 2018年10月31日

队列是一种先进先出`(FIFO)`的数据结构

1，队列的链表实现

```java
public class ListQueue<Item> implements Iterable<Item> {
    private class Node {
        Item item;
        Node next;
    }

    private Node first;
    private Node last;
    private int N;

    public boolean isEmpty() {
        return first == null;
    }

    public int size() {
        return N;
    }

    public void enqueue(Item item) {
        Node oldLast = last;
        last = new Node();
        last.item = item;
        last.next = null;
        if (isEmpty()) {
            first = last;
        } else {
            oldLast.next = last;
        }
        N++;
    }

    public Item dequeue() {
        Item item = first.item;
        first = first.next;
        if (isEmpty()) {
            last = null;
        }
        N--;
        return item;
    }

    @Override
    public Iterator<Item> iterator() {
        return new ListIterator(first);
    }

    private class ListIterator implements Iterator<Item> {
        private Node current;

        public ListIterator(Node first) {
            current = first;
        }

        @Override
        public boolean hasNext() {
            return current != null;
        }

        @Override
        public void remove() {
            throw new UnsupportedOperationException();
        }

        @Override
        public Item next() {
            if (!hasNext()) {
                throw new NoSuchElementException();
            }
            Item item = current.item;
            current = current.next;
            return item;
        }
    }
}
```


StdIn
StdOut
to
be
or
not
to
- to
be
- be
- or
that
- not
- to
- be
is
null

2，队列的数组实现

```java
public class ResizingArrayQueue<Item> implements Iterable<Item> {
    private Item[] a = (Item[]) new Object[2];
    private int N;
    private int first;
    private int last;

    public boolean isEmpty() {
        return N == 0;
    }

    public int size() {
        return N;
    }

    private void resize(int max) {
        Item[] temp = (Item[]) new Object[max];
        for (int i = 0; i < N; i++) {
            temp[i] = a[(first + i) % a.length];
        }
        a = temp;
        first = 0;
        last = N;
    }

    public void enqueue(Item item) {
        if (N == a.length) {
            resize(2 * a.length);
        }
        a[last++] = item;
        if (last == a.length) {
            //环形数组,到底了从头计数
            last = 0;
        }
        N++;
    }

    public Item dequeue() {
        if (isEmpty()) {
            throw new NoSuchElementException();
        }
        Item item = a[first];
        //避免对象游离，即保存一个不需要的对象的引用
        a[first] = null;
        first++;
        N--;
        if (first == a.length) {
            //环形数组，到底了从头开始
            first = 0;
        }
        if (N > 0 && N == a.length / 4) {
                resize(a.length / 2);
            }
            return item;
        }

        @Override
        public Iterator<Item> iterator() {
            return new ArrayIterator();
        }

        private class ArrayIterator implements Iterator<Item> {
            private int i = 0;

            @Override
            public boolean hasNext() {
                return i < N;
            }

            @Override
            public void remove() {
                throw new UnsupportedOperationException();
            }

            @Override
            public Item next() {
                if (!hasNext()) {
                    throw new NoSuchElementException();
                }
                Item item = a[(first + i) % a.length];
                i++;
                return item;
            }
        }
    }
```

3，队列的应用

- 圆圈中最后剩下的数字 题目：0, 1, ..., n-1这n个数字排成一个圆圈，从数字0开始每次从这个圆圈里删除第m个数字。求出这个圆圈里剩下的最后一个数字。

```java
    public int lastRemainingSolution(int n, int m) {
        Queue<Integer> queue = new LinkedList<>();
        for (int i = 0; i < n; i++) {
            queue.offer(i);
        }
        int count = 0;
        int result = -1;
        while (!queue.isEmpty()) {
            if (count == m - 1) {
                result = queue.poll();
                count = 0;
            }
            if (!queue.isEmpty()) {
                queue.offer(queue.poll());
            }
            count++;
        }
        return result;
    }
```