

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

```
1 public class ListQueue<Item> implements Iterable<Item> {
2     private class Node {
3         Item item;
4         Node next;
5     }
6
7     private Node first;
8     private Node last;
9     private int N;
10
11     public boolean isEmpty() {
12         return first == null;
13     }
14
15     public int size() {
16         return N;
17     }
18
19     public void enqueue(Item item) {
20         Node oldLast = last;
21         last = new Node();
22         last.item = item;
23         last.next = null;
24         if (isEmpty()) {
25             first = last;
26         } else {
27             oldLast.next = last;
28         }
29         N++;
30     }
31
32     public Item dequeue() {
33         Item item = first.item;
```

```

34     first = first.next;
35     if (isEmpty()) {
36         last = null;
37     }
38     N--;
39     return item;
40 }
41
42 @Override
43 public Iterator<Item> iterator() {
44     return new ListIterator(first);
45 }
46
47 private class ListIterator implements Iterator<Item> {
48     private Node current;
49
50     public ListIterator(Node first) {
51         current = first;
52     }
53
54     @Override
55     public boolean hasNext() {
56         return current != null;
57     }
58
59     @Override
60     public void remove() {
61         throw new UnsupportedOperationException();
62     }
63
64     @Override
65     public Item next() {
66         if (!hasNext()) {
67             throw new NoSuchElementException();
68         }
69         Item item = current.item;
70         current = current.next;
71         return item;
72     }
73 }
74 }

```

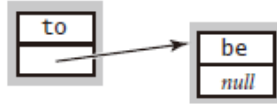
StdIn

StdOut

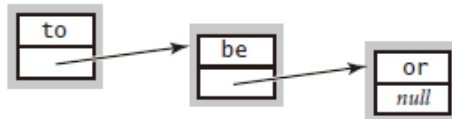
to



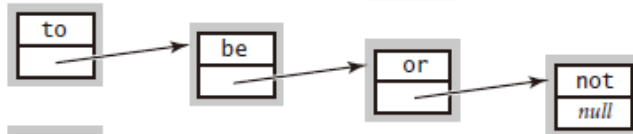
be



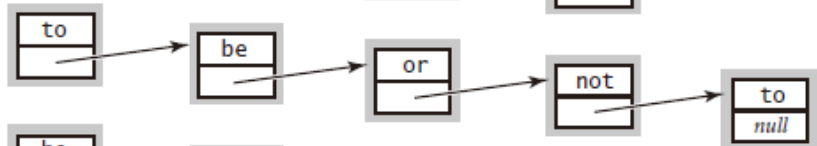
or



not

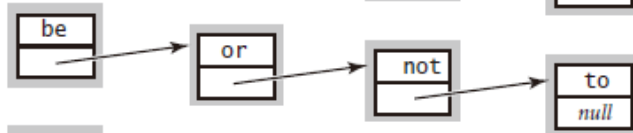


to

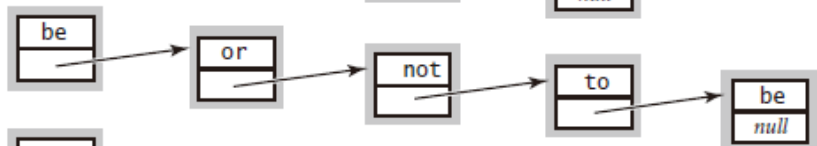


-

to

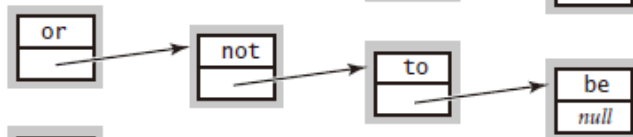


be



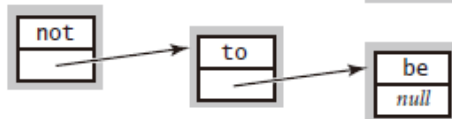
-

be

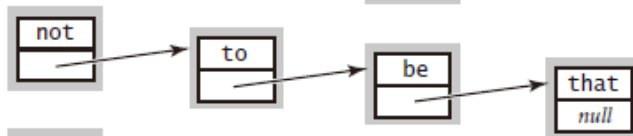


-

or

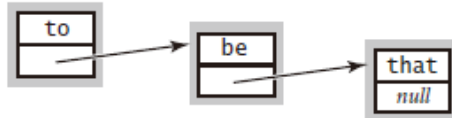


that



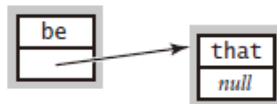
-

not



-

to



-

be



is



2, 队列的数组实现

```
1 public class ResizingArrayQueue<Item> implements Iterable<Item> {
2     private Item[] a = (Item[]) new Object[2];
3     private int N;
4     private int first;
5     private int last;
6
7     public boolean isEmpty() {
8         return N == 0;
9     }
10
11    public int size() {
12        return N;
13    }
14
15    private void resize(int max) {
16        Item[] temp = (Item[]) new Object[max];
17        for (int i = 0; i < N; i++) {
18            temp[i] = a[(first + i) % a.length];
19        }
20        a = temp;
21        first = 0;
22        last = N;
23    }
24
25    public void enqueue(Item item) {
26        if (N == a.length) {
27            resize(2 * a.length);
28        }
29        a[last++] = item;
30        if (last == a.length) {
31            //环形数组,到底了从头计数
32            last = 0;
33        }
34        N++;
35    }
36
37    public Item dequeue() {
38        if (isEmpty()) {
```

```

39         throw new NoSuchElementException();
40     }
41     Item item = a[first];
42     //避免对象游离, 即保存一个不需要的对象的引用
43     a[first] = null;
44     first++;
45     N--;
46     if (first == a.length) {
47         //环形数组, 到底了从头开始
48         first = 0;
49     }
50     if (N > 0 && N == a.length / 4) {
51         resize(a.length / 2);
52     }
53     return item;
54 }
55
56 @Override
57 public Iterator<Item> iterator() {
58     return new ArrayIterator();
59 }
60
61 private class ArrayIterator implements Iterator<Item> {
62     private int i = 0;
63
64     @Override
65     public boolean hasNext() {
66         return i < N;
67     }
68
69     @Override
70     public void remove() {
71         throw new UnsupportedOperationException();
72     }
73
74     @Override
75     public Item next() {
76         if (!hasNext()) {
77             throw new NoSuchElementException();
78         }
79         Item item = a[(first + i) % a.length];
80         i++;

```

```
81         return item;
82     }
83 }
84 }
```

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。

```
1     public int lastRemainingSolution(int n, int m) {
2         Queue<Integer> queue = new LinkedList<>();
3         for (int i = 0; i < n; i++) {
4             queue.offer(i);
5         }
6         int count = 0;
7         int result = -1;
8         while (!queue.isEmpty()) {
9             if (count == m - 1) {
10                 result = queue.poll();
11                 count = 0;
12             }
13             if (!queue.isEmpty()) {
14                 queue.offer(queue.poll());
15             }
16             count++;
17         }
18         return result;
19     }
```