

# DS–Autumn 2023 — Homework 1 Solutions

王子隆, SID 2221411126

2023 年 9 月 22 日

## 摘要

线性表功能实现及应用

Merge(L1,L2)

Dispose (L1, 12)

Sort (L)

Insert(L,a,b) 不/带头链表L,在元素值a之前插入b

应用一元多项式的加减法（链表）

检测环形链表证明链表有无环

增加bool数据类型判断环形链表是否遍历完毕

hanoi塔

数据库（交并差叉积）

半矩阵存储

## 1. 线性表功能实现及应用

(a) 功能实现

Merge(L1,L2)

Dispose (L1, 12)

Sort (L)

Insert(L,a,b) 不/带头链表L,在元素值a之前插入b

(b) 应用

一元多项式的加减法（链表）

### 1.1 数组、链表、双向循环链表建立

**Solutions:**

#### 1. 数组 ArrayDeque.java

```
1  /** Array based list.
2  *   @author zilong
3  */
```

```
4
5  /* Invariants:
6  addLast: The next item we want to add, will go into position size
7  getLast: The item we want to return is in position size - 1
8  size: The number of items in the list should be size.
9  */
10 public class ArrayDeque<T> implements List<T>{
11     private T[] items;
12     private int size;
13
14     /** Creates an empty array deque. */
15     public ArrayDeque() {
16         items = (T[]) new Object[8];
17         size = 0;
18     }
19
20     /** Resizes the underlying array to the target capacity. */
21     private void resize(int capacity) {
22         T[] a = (T[]) new Object[capacity];
23         System.arraycopy(items, 0, a, 0, size);
24         items = a;
25     }
26     /** Adds an item of type T to the front of the deque.*/
27     public void addFirst(T item) {
28         if (size == items.length) {
29             //this.resize(size + 1);
30             this.resize(size + 1);
31         }
32         //for (int i = size; i > 0; i--) {
33         //    items[i] = items[i - 1 ];
34         //}
35         System.arraycopy(items, 0, items, 1, size);
36         items[0] = item;
37         size = size + 1;
38     }
39
40     /** Inserts X into the back of the list. */
41     public void addLast(T item) {
42         if (size == items.length) {
43             //this.resize(size + 1);
44             this.resize(size + 1);
45         }
46         items[size] = item;
47         size = size + 1;
48     }
49 }
```

```
50
51  /** Returns true if deque is empty, false otherwise.*/
52  public boolean isEmpty() {
53      if (size == 0) {
54          return true;
55      }
56      return false;
57  }
58
59  /** Prints the items in the deque from first to last, separated by a
60   * space.
61   * Once all the items have been printed, print out a new line.*/
62  public void printDeque() {
63      if (size == 0) {
64          System.out.println("The deque is empty.\n");
65      }
66      for (int i = 0; i < size; i++) {
67          System.out.print(items[i] + " ");
68      }
69      System.out.print("\n");
70  }
71
72  /** Gets the ith item in the list (0 is the front).
73   * If no such item exists, returns null. Must not alter the deque! */
74  public T get(int i) {
75      return items[i];
76  }
77
78  public T getFirst() {
79      return get(0);
80  }
81
82  public T getLast() {
83      int lastActualItemIndex = size - 1;
84      return items[lastActualItemIndex];
85  }
86
87  /** Returns the number of items in the list. */
88  public int size() {
89      return size;
90  }
91
92  /** Removes and returns the item at the front of the deque.
93   * If no such item exists, returns null. */
94  public T removeFirst() {
95      if (size == 0) {
```

```
95         return null;
96     }
97     T front = get(0);
98
99
100     //for (int i = 0; i < size - 1; i++) {
101     //    items[i] = items[i + 1];
102     //}
103
104     System.arraycopy(items, 1, items, 0, size - 1);
105
106     items[size - 1] = null; //this may not right.
107
108     size = size - 1;
109     if (size <= items.length * 0.25 && size > 16) {
110         resize((int) (items.length * 0.25));
111     }
112     return front;
113 }
114
115 /** Deletes item from back of the list and
116  * returns deleted item. */
117 public T removeLast() {
118     if (size == 0) {
119         return null;
120     }
121     T back = get(size - 1);
122     items[size - 1] = null;
123     size = size - 1;
124     if (size <= items.length * 0.25 && size > 16) {
125         resize((int) (items.length * 0.25)+1);
126     }
127     return back;
128 }
129
130
131 /** Inserts item into given position.*/
132 public void insert(T x, int position) {
133     T[] newItems = (T[]) new Object[items.length + 1];
134
135     System.arraycopy(items, 0, newItems, 0, position);
136     newItems[position] = x;
137
138     System.arraycopy(items, position, newItems, position + 1,
139         items.length - position);
140     items = newItems;
```

```
140     }
141
142
143 }
```

## 2. 链表 Intlist.Java

```
1     public class IntList {
2
3         private int first;
4         private IntList rest;
5
6         /** A List with first FIRST0 and rest REST0. */
7         public IntList(int first0, IntList rest0) {
8             first = first0;
9             rest = rest0;
10        }
11
12        /** A List with null rest, and first = 0.*/
13        public IntList() {
14            /* NOTE: public IntList () { } would also work. */
15            this(0, null);
16        }
17
18
19        /**
20         * Returns a list consisting of the elements of A followed by the
21         * * elements of B. May modify items of A.
22         */
23
24        public static IntList dmerge(IntList A, IntList B) {
25            if (A == null) {
26                A = B;
27                return A;
28            }
29            IntList prt = A;
30            while (prt.rest != null) {
31                prt = prt.rest;
32            }
33            prt.rest = B;
34            return A;
35        }
36
37        /**
38         * Returns a list consisting of the elements of A followed by the
39         * * elements of B. May NOT modify items of A.
```

```

40     */
41
42     public static IntList merge(IntList A, IntList B) {
43         if (A == null) {
44             return B;
45         }
46         if (A.rest == null) {
47             return new IntList(A.first, B);
48         }
49         return new IntList(A.first, merge(A.rest, B));
50     }
51 }

```

### 3. 双向循环链表 LinkedListDeque.java

```

1  /** DLLists based list.
2  *   @author zilong
3  */
4
5  /* Invariants:
6  addLast: The next item we want to add, will go into position size
7  getLast: The item we want to return is in position size - 1
8  size: The number of items in the list should be size.
9  */
10
11 public class LinkedListDeque<T> implements List<T>{
12
13     /* Double-ended queues are sequence containers with dynamic sizes
14     that can be expanded or contracted on both ends (either its front or its
15     back).*/
16
17     private class StuffNode {
18         private StuffNode prev;
19         private T item;
20         private StuffNode next;
21         public StuffNode(StuffNode f, T i, StuffNode n) {
22             prev = f;
23             item = i;
24             next = n;
25         }
26     }
27
28     /** Returns the ith item of this IntList. */
29     public T get(int i) {
30         if (i == 0) {
31             return item;
32         }
33         return next.get(i - 1);
34     }
35 }

```

```
31     }
32 }
33
34 /* The first item (if it exists) is at sentinel.next. */
35 private StuffNode sentinel;
36 private int size;
37
38 public LinkedListDeque() {
39     size = 0;
40     sentinel = new StuffNode(null, null, null);
41     sentinel.next = sentinel;
42     sentinel.prev = sentinel;
43 }
44
45 public LinkedListDeque(T x) {
46     sentinel = new StuffNode(null, null, null);
47     sentinel.next = new StuffNode(sentinel, x, sentinel);
48     sentinel.prev = sentinel.next;
49     size = 1;
50 }
51
52 /** Creates a deep copy of other.*/
53 /*public LinkedListDeque(LinkedListDeque other) {
54     size = 0;
55     sentinel = new StuffNode(null, null, null);
56     sentinel.next = sentinel;
57     sentinel.prev = sentinel;
58     for (int i = 0; i < other.size(); i++) {
59         this.addLast((T) other.get(i + 1));
60     }
61 }*/
62
63 /** Adds an item of type T to the front of the deque.*/
64 public void addFirst(T item) {
65     size = size + 1;
66     sentinel.next = new StuffNode(sentinel, item, sentinel.next);
67     if (size == 1) {
68         sentinel.prev = sentinel.next;
69     }
70     sentinel.next.next.prev = sentinel.next;
71 }
72
73 /** Adds an item of type T to the back of the deque.*/
74 public void addLast(T item) {
75     size = size + 1;
76     sentinel.prev.next = new StuffNode(sentinel.prev, item, sentinel);
```

```
77         sentinel.prev = sentinel.prev.next;
78     }
79
80     /** Returns true if deque is empty, false otherwise.*/
81     public boolean isEmpty() {
82         /* if (sentinel.next.item == null){
83             return true;
84         }
85         return false;*/
86         if (size == 0) {
87             return true;
88         }
89         return false;
90     }
91
92     /** Returns the number of items in the deque.*/
93     public int size() {
94         return size;
95     }
96     /** Prints the items in the deque from first to last, separated by a
97         space.
98     * Once all the items have been printed, print out a new line.*/
99     public void printDeque() {
100         StuffNode p = sentinel.next;
101         if (size == 0) {
102             System.out.println("The deque is empty.");
103         }
104         for (int i = 0; i < size; i++) {
105             System.out.print(p.item + " ");
106             p = p.next;
107         }
108         System.out.println("\n");
109     }
110
111     /**Removes and returns the item at the front of the deque.
112     * If no such item exists, returns null.*/
113     public T removeFirst() {
114         if (size == 0) {
115             return null;
116         }
117         T front = sentinel.next.item;
118         sentinel.next.next.prev = sentinel;
119         sentinel.next = sentinel.next.next;
120         size = size - 1;
121         return front;
122     }
```



```
122
123     /** Removes and returns the item at the back of the deque.
124     * If no such item exists, returns null.*/
125     public T removeLast() {
126         if (size == 0) {
127             return null;
128         }
129         T back = sentinel.prev.item;
130         sentinel.prev.prev.next = sentinel;
131         sentinel.prev = sentinel.prev.prev;
132         size = size - 1;
133         return back;
134     }
135
136     /** Gets the item at the given index, where 0 is the front, 1 is the next
137     item, and so forth.
138     * If no such item exists, returns null. Must not alter the deque!*/
139     public T get(int index) {
140         StuffNode p = sentinel;
141         if (index > size) {
142             return null;
143         }
144         for (int i = 0; i <= index; i++) {
145             p = p.next;
146         }
147         //System.out.println(p.item);
148         return p.item;
149     }
150
151     public T getFirst() {
152         return get(0);
153     }
154
155     public T getLast() {
156         return get(size - 1);
157     }
158
159
160     /** Same as get, but uses recursion.*/
161     public T getRecursive(int index) {
162         if (index > size) {
163             return null;
164         }
165         return sentinel.next.get(index);
166     }
```

```

167
168
169     /** Inserts item into given position.*/
170     public void insert(T item, int position) {
171         if (sentinel.next == null || position == 0) {
172             addFirst(item);
173             return;
174         }
175
176         StuffNode currentNode = sentinel.next.next;
177         while (position > 1 && currentNode.next != null) {
178             position -= 1;
179             currentNode = currentNode.next;
180         }
181
182         StuffNode newNode = new StuffNode(currentNode, item,
183             currentNode.next);
184         currentNode.next = newNode;
185     }
186 }

```

## 1.2 应用

### Solutions:

#### 1. interface List.java

```

1     public interface List<Item> {
2         /**
3          * Inserts X into the back of the list.
4          */
5         public void addLast(Item x);
6
7         /**
8          * Returns the item from the back of the list.
9          */
10        public Item getLast();
11
12        /**
13         * Gets the ith item in the list (0 is the front).
14         */
15        public Item get(int i);
16
17        /**
18         * Returns the number of items in the list.

```

```
19     */
20 public int size();
21
22 /**
23  * Deletes item from back of the list and
24  * returns deleted item.
25  */
26 public Item removeLast();
27
28 /**
29  * Inserts item into given position.
30  * Code from discussion #3
31  */
32 public void insert(Item x, int position);
33
34 /**
35  * Inserts an item at the front.
36  */
37 public void addFirst(Item x);
38
39 /**
40  * Gets an item from the front.
41  */
42 public Item getFirst();
43
44 /** Prints the list. Works for ANY kind of list. */
45 default public void print() {
46     for (int i = 0; i < size(); i = i + 1) {
47         System.out.print(get(i) + " ");
48     }
49 }
50 }
```