DS–Autumn 2023 — Homework 1Solutions

王子隆, SID 2221411126

2023 年 10 月 13 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

```
/** Array based list.
*  @author zilong
*/
```

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

```
    }


}
```

2. 链表 Intlist.Java

```
    public class IntList {

    private int first;
    private IntList rest;

    /** A List with first FIRST0 and rest REST0. */
    public IntList(int first0, IntList rest0) {
        first = first0;
        rest = rest0;
    }

    /** A List with null rest, and first = 0.*/
    public IntList() {
        /* NOTE: public IntList () { }  would also work. */
        this(0, null);
    }


    /**
    * Returns a list consisting of the elements of A followed by the
    * *  elements of B.  May modify items of A.
    */

    public static IntList dmerge(IntList A, IntList B) {
        if (A == null) {
            A = B;
            return A;
        }
        IntList prt = A;
        while (prt.rest != null) {
            prt = prt.rest;
        }
        prt.rest = B;
        return A;
    }

    /**
    * Returns a list consisting of the elements of A followed by the
    * * elements of B.  May NOT modify items of A.
```

```
    */

    public static IntList merge(IntList A, IntList B) {
        if (A == null) {
            return B;
        }
        if (A.rest == null) {
            return new IntList(A.first, B);
        }
        return new IntList(A.first, merge(A.rest, B));
    }
}
```

3. 双向循环链表 LinkedListDeque.java

```
/** DLLists based list.
*  @author zilong
*/

/* Invariants:
addLast: The next item we want to add, will go into position size
getLast: The item we want to return is in position size - 1
size: The number of items in the list should be size.
*/

public class LinkedListDeque<T> implements List<T>{

    /* Double-ended queues are sequence containers with dynamic sizes
    that can be expanded or contracted on both ends (either its front or its
        back).*/

    private class StuffNode {
        private StuffNode prev;
        private T item;
        private StuffNode next;
        public StuffNode(StuffNode f, T i, StuffNode n) {
            prev = f;
            item = i;
            next = n;
        }
        /** Returns the ith item of this IntList. */
        public T get(int i) {
            if (i == 0) {
                return item;
            }
            return next.get(i - 1);
```

```
        }
    }

    /* The first item (if it exists) is at sentinel.next. */
    private StuffNode sentinel;
    private int size;

    public LinkedListDeque() {
        size = 0;
        sentinel = new StuffNode(null, null, null);
        sentinel.next = sentinel;
        sentinel.prev = sentinel;
    }

    public LinkedListDeque(T x) {
        sentinel = new StuffNode(null, null, null);
        sentinel.next = new StuffNode(sentinel,x,sentinel);
        sentinel.prev = sentinel.next;
        size = 1;
    }

    /** Creates a deep copy of other.*/
    /*public LinkedListDeque(LinkedListDeque other) {
        size = 0;
        sentinel = new StuffNode(null, null, null);
        sentinel.next = sentinel;
        sentinel.prev = sentinel;
        for (int i = 0; i < other.size(); i++) {
            this.addLast((T) other.get(i + 1));
        }
    }*/

    /** Adds an item of type T to the front of the deque.*/
    public void addFirst(T item) {
        size = size + 1;
        sentinel.next = new StuffNode(sentinel, item, sentinel.next);
        if (size == 1) {
            sentinel.prev = sentinel.next;
        }
        sentinel.next.next.prev = sentinel.next;
    }

    /** Adds an item of type T to the back of the deque.*/
    public void addLast(T item) {
        size = size + 1;
        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
            space.
97      * Once all the items have been printed, print out a new line.*/
98      public void printDeque() {
99          StuffNode p = sentinel.next;
100         if (size == 0) {
101             System.out.println("The deque is empty.");
102         }
103         for (int i = 0; i < size; i++) {
104             System.out.print(p.item + " ");
105             p = p.next;
106         }
107         System.out.println("\n");
108     }
109
110     /**Removes and returns the item at the front of the deque.
111     * If no such item exists, returns null.*/
112     public T removeFirst() {
113         if (size == 0) {
114             return null;
115         }
116         T front = sentinel.next.item;
117         sentinel.next.next.prev = sentinel;
118         sentinel.next = sentinel.next.next;
119         size = size - 1;
120         return front;
121     }
```

```
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
        item, and so forth.
137     * If no such item exists, returns null. Must not alter the deque!*/
138     public T get(int index) {
139         StuffNode p = sentinel;
140         if (index > size) {
141             return null;
142         }
143         for (int i = 0; i <= index; i++) {
144             p = p.next;
145         }
146         //System.out.println(p.item);
147         return p.item;
148     }
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,
                currentNode.next);
183         currentNode.next = newNode;
184     }
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  }
```