# 原 CS61B Homework5作业回顾（附代码）

2018年03月07日 11:13:04 vincent_chan7 阅读数：408 标签： CS61b Java

HW5一开始做不出来的同学可以继续先看Lecture，Encapsulated List那一节老师会回顾一下HW5，对作业有启发思路的作用。

Part1.

HW5的DListNode增加一个myList的field，用于确认node位于哪个list，同时也为后面的isValidNode()服务。

remove()之后需要将myList, prev, next 统统指向null，不然这个node的prev和next仍指向前后，后导致后面如果insertAfter出错。

Part2.

insert(), union(), intersection()等方法中的对比需要使用compareTo(), 因为求并交集时要比较字典序lexicographically，比如“3”在“5”的前面还是后面。回
.equals(), compareTo()的用法

| | |
|---|---|
| a==b | 比较a和b是否指向同一个object |
| a.equals(b) | 验证a和b是否相等。允许a.equals(null) |
| a.compareTo(b) | 比较a和b的字典序，1. a<b, 返回负数; 2.a=b,返回0； 3.a>b,返回正数. a.compareTo(null)会返回NullPointException |

用了compareTo()之后要将item强制转换成Compareable
DListNode

```
/* DListNode.java */

package com.homework5;

/**
 *  A DListNode is a mutable node in a DList (doubly-linked list).
 **/

public class DListNode extends ListNode {

  /**
   *  (inherited)  item references the item stored in the current node.
   *  (inherited)  myList references the List that contains this node.
   *  prev references the previous node in the DList.
   *  next references the next node in the DList.
   *
   *  DO NOT CHANGE THE FOLLOWING FIELD DECLARATIONS.
   **/

  protected DListNode prev;
  protected DListNode next;

  /**
   *  DListNode() constructor.
   *  @param i the item to store in the node.
   *  @param l the list this node is in.
   *  @param p the node previous to this node.
   *  @param n the node following this node.
   */
  DListNode(Object i, DList l, DListNode p, DListNode n) {
    item = i;
    myList = l;
    prev = p;
    next = n;
  }

  /**
   *  isValidNode returns true if this node is valid; false otherwise.
   *
   *  @return true if this node is valid; false otherwise.
   *
   *  Performance:  runs in O(1) time.
   */
  public boolean isValidNode() {
    return myList != null;
  }

  /**
   *  next() returns the node following this node.  If this node is invalid,
   *  throws an exception.
   *
   *  @return the node following this node.
   *  @exception InvalidNodeException if this node is not valid.
   *
   *  Performance:  runs in O(1) time.
   */
  public ListNode next() throws InvalidNodeException {
    if (!isValidNode()) {
      throw new InvalidNodeException("next() called on invalid node");
    }
    return next;
  }

  /**
   *  prev() returns the node preceding this node.  If this node is invalid,
   *  throws an exception.
   *
   *  @return the node preceding this node.
   *  @exception InvalidNodeException if this node is not valid.
   *
   *  Performance:  runs in O(1) time.
   */
  public ListNode prev() throws InvalidNodeException {
    if (!isValidNode()) {
      throw new InvalidNodeException("prev() called on invalid node");
    }
    return prev;
  }

  /**
   *  insertAfter() inserts an item immediately following this node.  If this
   *  node is invalid, throws an exception.
   *
   *  @param item the item to be inserted.
   *  @exception InvalidNodeException if this node is not valid.
   *
   *  Performance:  runs in O(1) time.
   */
  public void insertAfter(Object item) throws InvalidNodeException {
    if (!isValidNode()) {
      throw new InvalidNodeException("insertAfter() called on invalid node");
    }
    // Your solution here.  Will look something like your Homework 4 solution,
    //   but changes are necessary.  For instance, there is no need to check if
    //   "this" is null.  Remember that this node's "myList" field tells you
    //   what DList it's in.  You should use myList.newNode() to create the
    //   new node.
      DListNode new_node = ((DList)myList).newNode(item,(DList)myList,this,this.next);
      this.next.prev = new_node;
      this.next = new_node;
      this.myList.size++;
  }

  /**
   *  insertBefore() inserts an item immediately preceding this node.  If this
   *  node is invalid, throws an exception.
   *
   *  @param item the item to be inserted.
   *  @exception InvalidNodeException if this node is not valid.
   *
   *  Performance:  runs in O(1) time.
   */
  public void insertBefore(Object item) throws InvalidNodeException {
    if (!isValidNode()) {
      throw new InvalidNodeException("insertBefore() called on invalid node");
    }
    // Your solution here.  Will look something like your Homework 4 solution,
    //   but changes are necessary.  For instance, there is no need to check if
    //   "this" is null.  Remember that this node's "myList" field tells you
    //   what DList it's in.  You should use myList.newNode() to create the
    //   new node.
      DListNode new_node = ((DList)myList).newNode(item,(DList)myList,this.prev,this);
      this.prev.next = new_node;
      this.prev = new_node;
      this.myList.size++;
  }

  /**
   *  remove() removes this node from its DList.  If this node is invalid,
   *  throws an exception.
   *
   *  @exception InvalidNodeException if this node is not valid.
   *
   *  Performance:  runs in O(1) time.
   */
  public void remove() throws InvalidNodeException {
    if (!isValidNode()) {
      throw new InvalidNodeException("remove() called on invalid node");
    }
    // Your solution here.  Will look something like your Homework 4 solution,
    //   but changes are necessary.  For instance, there is no need to check if
    //   "this" is null.  Remember that this node's "myList" field tells you
    //   what DList it's in.
      this.prev.next = this.next;
      this.next.prev = this.prev;
      this.myList.size--;



    // Make this node an invalid node, so it cannot be used to corrupt myList.
    myList = null;
    // Set other references to null to improve garbage collection.
    next = null;
    prev = null;
  }

}
```

DList

```
/* DList.java */

package com.homework5;

/**
 *  A DList is a mutable doubly-linked list ADT.  Its implementation is
 *  circularly-linked and employs a sentinel node at the head of the list.
 *
 *  DO NOT CHANGE ANY METHOD PROTOTYPES IN THIS FILE.
 **/

public class DList extends List {

  /**
   *  (inherited)  size is the number of items in the list.
   *  head references the sentinel node.
   *  Note that the sentinel node does not store an item, and is not included
   *  in the count stored by the "size" field.
   *
   *  DO NOT CHANGE THE FOLLOWING FIELD DECLARATION.
   **/

  protected DListNode head;

  /* DList invariants:
   *  1)  head != null.
   *  2)  For every DListNode x in a DList, x.next != null.
   *  3)  For every DListNode x in a DList, x.prev != null.
   *  4)  For every DListNode x in a DList, if x.next == y, then y.prev == x.
   *  5)  For every DListNode x in a DList, if x.prev == y, then y.next == x.
   *  6)  For every DList l, l.head.myList = null.  (Note that l.head is the
   *      sentinel.)
   *  7)  For every DListNode x in a DList l EXCEPT l.head (the sentinel),
   *      x.myList = l.
   *  8)  size is the number of DListNodes, NOT COUNTING the sentinel,
   *      that can be accessed from the sentinel (head) by a sequence of
   *      "next" references.
   **/

  /**
   *  newNode() calls the DListNode constructor.  Use this method to allocate
   *  new DListNodes rather than calling the DListNode constructor directly.
   *  That way, only this method need be overridden if a subclass of DList
   *  wants to use a different kind of node.
   *
   *  @param item the item to store in the node.
   *  @param list the list that owns this node.  (null for sentinels.)
   *  @param prev the node previous to this node.
   *  @param next the node following this node.
   **/
  protected DListNode newNode(Object item, DList list,
                              DListNode prev, DListNode next) {
    return new DListNode(item, list, prev, next);
  }

  /**
   *  DList() constructs for an empty DList.
   **/
  public DList() {
    // Your solution here.  Similar to Homework 4, but now you need to specify
    //   the `list' field (second parameter) as well.
      head = newNode(null,null,null,null);
      head.next = head;
      head.prev = head;
      size = 0;
  }

  /**
   *  insertFront() inserts an item at the front of this DList.
   *
   *  @param item is the item to be inserted.
   *
   *  Performance:  runs in O(1) time.
   **/
  public void insertFront(Object item) {
    // Your solution here.  Similar to Homework 4, but now you need to specify
    //   the `list' field (second parameter) as well.
      DListNode new_node = newNode(item,this,head,head.next);
      head.next.prev = new_node;
      head.next = new_node;
      size++;
  }

  /**
   *  insertBack() inserts an item at the back of this DList.
   *
   *  @param item is the item to be inserted.
   *
   *  Performance:  runs in O(1) time.
   **/
```

```
  public void insertBack(Object item) {
// Your solution here.  Similar to Homework 4, but now you need to specify
//   the `list' field (second parameter) as well.
    DListNode new_node = newNode(item,this,head.prev,head);
    head.prev.next = new_node;
    head.prev = new_node;
    size++;
  }

  /**
   *  front() returns the node at the front of this DList.  If the DList is
   *  empty, return an "invalid" node--a node with the property that any
   *  attempt to use it will cause an exception.  (The sentinel is "invalid".)
   *
   *  DO NOT CHANGE THIS METHOD.
   *
   *  @return a ListNode at the front of this DList.
   *
   *  Performance:  runs in O(1) time.
   */
  public ListNode front() {
    return head.next;
  }

  /**
   *  back() returns the node at the back of this DList.  If the DList is
   *  empty, return an "invalid" node--a node with the property that any
   *  attempt to use it will cause an exception.  (The sentinel is "invalid".)
   *
   *  DO NOT CHANGE THIS METHOD.
   *
   *  @return a ListNode at the back of this DList.
   *
   *  Performance:  runs in O(1) time.
   */
  public ListNode back() {
    return head.prev;
  }

  /**
   *  toString() returns a String representation of this DList.
   *
   *  DO NOT CHANGE THIS METHOD.
   *
   *  @return a String representation of this DList.
   *
   *  Performance:  runs in O(n) time, where n is the length of the list.
   */
  public String toString() {
    String result = "[  ";
    DListNode current = head.next;
    while (current != head) {
      result = result + current.item + "  ";
      current = current.next;
    }
    return result + "]";
  }

  private static void testInvalidNode(ListNode p) {
    System.out.println("p.isValidNode() should be false: " + p.isValidNode());
    try {
      p.item();
      System.out.println("p.item() should throw an exception, but didn't.");
    } catch (InvalidNodeException lbe) {
      System.out.println("p.item() should throw an exception, and did.");
    }
    try {
      p.setItem(new Integer(0));
      System.out.println("p.setItem() should throw an exception, but didn't.");
    } catch (InvalidNodeException lbe) {
      System.out.println("p.setItem() should throw an exception, and did.");
    }
    try {
      p.next();
      System.out.println("p.next() should throw an exception, but didn't.");
    } catch (InvalidNodeException lbe) {
      System.out.println("p.next() should throw an exception, and did.");
    }
    try {
      p.prev();
      System.out.println("p.prev() should throw an exception, but didn't.");
    } catch (InvalidNodeException lbe) {
      System.out.println("p.prev() should throw an exception, and did.");
    }
    try {
      p.insertBefore(new Integer(1));
      System.out.println("p.insertBefore() should throw an exception, but " +
                         "didn't.");
    } catch (InvalidNodeException lbe) {
      System.out.println("p.insertBefore() should throw an exception, and did."
                         );
    }
    try {
      p.insertAfter(new Integer(1));
      System.out.println("p.insertAfter() should throw an exception, but " +
                         "didn't.");
    } catch (InvalidNodeException lbe) {
      System.out.println("p.insertAfter() should throw an exception, and did."
                         );
    }
    try {
      p.remove();
      System.out.println("p.remove() should throw an exception, but didn't.");
    } catch (InvalidNodeException lbe) {
      System.out.println("p.remove() should throw an exception, and did.");
    }
  }

  private static void testEmpty() {
    List l = new DList();
    System.out.println("An empty list should be [  ]: " + l);
    System.out.println("l.isEmpty() should be true: " + l.isEmpty());
    System.out.println("l.length() should be 0: " + l.length());
    System.out.println("Finding front node p of l.");
    ListNode p = l.front();
    testInvalidNode(p);
    System.out.println("Finding back node p of l.");
    p = l.back();
    testInvalidNode(p);
    l.insertFront(new Integer(10));
    System.out.println("l after insertFront(10) should be [  10  ]: " + l);
  }

  public static void main(String[] argv) {
    testEmpty();
    List l = new DList();
    l.insertFront(new Integer(3));
    l.insertFront(new Integer(2));
    l.insertFront(new Integer(1));
    System.out.println("l is a list of 3 elements: " + l);
    try {
      ListNode n;
      int i = 1;
      for (n = l.front(); n.isValidNode(); n = n.next()) {
        System.out.println("n.item() should be " + i + ": " + n.item());
        n.setItem(new Integer(((Integer) n.item()).intValue() * 2));
        System.out.println("n.item() should be " + 2 * i + ": " + n.item());
        i++;
      }
      System.out.println("After doubling all elements of l: " + l);
      testInvalidNode(n);

      i = 6;
      for (n = l.back(); n.isValidNode(); n = n.prev()) {
    System.out.println("n.item() should be " + i + ": " + n.item());
    n.setItem(new Integer(((Integer) n.item()).intValue() * 2));
    System.out.println("n.item() should be " + 2 * i + ": " + n.item());
    i = i - 2;       }
      System.out.println("After doubling all elements of l again: " + l);
      testInvalidNode(n);

      n = l.front().next();
      System.out.println("Removing middle element (8) of l: " + n.item());
      n.remove();
      System.out.println("l is now: " + l);
      testInvalidNode(n);
      n = l.back();
      System.out.println("Removing end element (12) of l: " + n.item());
      n.remove();
      System.out.println("l is now: " + l);
      testInvalidNode(n);

      n = l.front();
      System.out.println("Removing first element (4) of l: " + n.item());
      n.remove();
      System.out.println("l is now: " + l);
      testInvalidNode(n);
    } catch (InvalidNodeException lbe) {
      System.err.println ("Caught InvalidNodeException that should not happen."
                          );
      System.err.println ("Aborting the testing code.");
    }
  }
}
```

Set

```
package com.homework5;/* Set.java */


/**
 *  A Set is a collection of Comparable elements stored in sorted order.
 *  Duplicate elements are not permitted in a Set.
 **/
public class Set {
  /* Fill in the data fields here. */
    List setList;

  /**
   * Set ADT invariants:
   *  1)  The Set's elements must be precisely the elements of the List.
   *  2)  The List must always contain Comparable elements, and those elements
   *      must always be sorted in ascending order.
   *  3)  No two elements in the List may be equal according to compareTo().
   **/

  /**
   *  Constructs an empty Set.
   *
   *  Performance:  runs in O(1) time.
   **/
  public Set() {
    // Your solution here.
      setList = new DList();
  }

  /**
   *  cardinality() returns the number of elements in this Set.
   *
   *  Performance:  runs in O(1) time.
   **/
  public int cardinality() {
   // Replace the following line with your solution.
    return setList.length();
  }

  /**
   *  insert() inserts a Comparable element into this Set.
   *
   *  Sets are maintained in sorted order.  The ordering is specified by the
   *  compareTo() method of the java.lang.Comparable interface.
   *
   *  Performance:  runs in O(this.cardinality()) time.
   **/
  public void insert(Comparable c) {
   // Your solution here.
    if (setList.isEmpty()){
      setList.insertFront(c);
    } else {
      ListNode current = setList.front();
      try {

      while(current != setList.back() && ((Comparable)(current.item)).compareTo(c)<0){
        current = current.next();
      }
      if(((Comparable)(current.item)).compareTo(c) > 0){
        current.insertBefore(c);
      } else if(current == setList.back() && ((Comparable)(current.item)).compareTo(c) < 0){
        current.insertAfter(c);
      }

      } catch (InvalidNodeException e){
        System.out.println("Exception caught");
      }
    }
  }

  /**
   *  union() modifies this Set so that it contains all the elements it
   *  started with, plus all the elements of s.  The Set s is NOT modified.
   *  Make sure that duplicate elements are not created.
   *
   *  Performance:  Must run in O(this.cardinality() + s.cardinality()) time.
   *
   *  Your implementation should NOT copy elements of s or "this", though it
   *  will copy _references_ to the elements of s.  Your implementation will
   *  create new _nodes_ for the elements of s that are added to "this", but
   *  you should reuse the nodes that are already part of "this".
   *
   *  DO NOT MODIFY THE SET s.
   *  DO NOT ATTEMPT TO COPY ELEMENTS; just copy _references_ to them.
   **/
  public void union(Set s) {
   // Your solution here.
    if(s != null && !s.setList.isEmpty()){
      ListNode curNode = this.setList.front();
      ListNode sNode = s.setList.front();
      Comparable curItem, sItem;
      try {
        while (curNode.isValidNode() && sNode.isValidNode()){
          curItem = (Comparable)curNode.item;
          sItem = (Comparable)sNode.item;
          if (curItem.compareTo(sItem) == 0){
            curNode = curNode.next();
            sNode = sNode.next();
          }else if(curItem.compareTo(sItem) < 0){
            curNode = curNode.next();
          } else {
            curNode.insertBefore(sNode);
            curNode = curNode.next();
          }
        }

```

```
              while (sNode.isValidNode()){                       sItem = (Comparable)sNode.item;
                  setList.insertBack(sItem);
                  sNode = sNode.next();
              }
          } catch (InvalidNodeException e){
              System.out.println("Union Failed");
          }
      }

  }

  /**
   *  intersect() modifies this Set so that it contains the intersection of
   *  its own elements and the elements of s.  The Set s is NOT modified.
   *
   *  Performance:  Must run in O(this.cardinality() + s.cardinality()) time.
   *
   *  Do not construct any new ListNodes during the execution of intersect.
   *  Reuse the nodes of "this" that will be in the intersection.
   *
   *  DO NOT MODIFY THE SET s.
   *  DO NOT CONSTRUCT ANY NEW NODES.
   *  DO NOT ATTEMPT TO COPY ELEMENTS.
   **/
  public void intersect(Set s) {
    // Your solution here.
      if (!s.setList.isEmpty() && s != null){
          ListNode curNode = this.setList.front();
          ListNode sNode = s.setList.front();
          Comparable curItem, sItem;
          try {
            while(curNode.isValidNode() && sNode.isValidNode()){
                curItem = (Comparable)curNode.item;
                sItem = (Comparable)sNode.item;
                if (curItem.compareTo(sItem) == 0){
                    curNode = curNode.next();
                    sNode = sNode.next();
                } else if (curItem.compareTo(sItem) < 0) {
                    if (curNode != setList.back()){
                        curNode = curNode.next();
                        curNode.prev().remove();
                    } else{
                        curNode.remove();
                    }
                } else {
                    sNode = sNode.next();
                }

          }

          while (curNode.isValidNode()){
                if (curNode != setList.back()){
                    curNode = curNode.next();
                    curNode.prev().remove();
                } else {
                    curNode.remove();
                }
          }

          }catch (InvalidNodeException e){
              System.out.println("Intersect Failed");
          }
      }
  }

  /**
   *  toString() returns a String representation of this Set.  The String must
   *  have the following format:
   *    {  } for an empty Set.  No spaces before "{" or after "}"; two spaces
   *              between them.
   *    {  1  2  3  } for a Set of three Integer elements.  No spaces before
   *           "{" or after "}"; two spaces before and after each element.
   *           Elements are printed with their own toString method, whatever
   *           that may be.  The elements must appear in sorted order, from
   *           lowest to highest according to the compareTo() method.
   *
   *  WARNING:  THE AUTOGRADER EXPECTS YOU TO PRINT SETS IN _EXACTLY_ THIS
   *            FORMAT, RIGHT UP TO THE TWO SPACES BETWEEN ELEMENTS.  ANY
   *            DEVIATIONS WILL LOSE POINTS.
   **/
  public String toString() {
    // Replace the following line with your solution.
      String p = "{ ";
      ListNode curNode = setList.front();
      try{
          while(curNode.isValidNode()){
              p = p + curNode.item + "  ";
              curNode = curNode.next();
          }
      }catch (InvalidNodeException e){
          System.out.println("toString() Failed");
      }
      p = p + "}";
      return p;
  }

  public static void main(String[] argv) {
    Set s = new Set();
    s.insert(new Integer(3));
    s.insert(new Integer(4));
    s.insert(new Integer(3));
    System.out.println("Set s = " + s);

    Set s2 = new Set();
    s2.insert(new Integer(4));
    s2.insert(new Integer(5));
    s2.insert(new Integer(5));
    System.out.println("Set s2 = " + s2);

    Set s3 = new Set();
    s3.insert(new Integer(5));
    s3.insert(new Integer(3));
    s3.insert(new Integer(8));
    System.out.println("Set s3 = " + s3);

    s.union(s2);
    System.out.println("After s.union(s2), s = " + s);

    s.intersect(s3);
    System.out.println("After s.intersect(s3), s = " + s);

    System.out.println("s.cardinality() = " + s.cardinality());
    // You may want to add more (ungraded) test code here.
  }
}
```

想对作者说点什么？ 我来说一句

**CS61BHomework3作业回顾（附代码）** 127

Part 1.smoosh()函数，目的是将数组中出现的所有数字产生重复的部分删除，例如1... 来自： everest115的博客

**CS61B Homework6作业回顾（附代码）** 92

本次作业重点是hascode和compress function，评价一个compress function好坏的标... 来自： everest115的博客

**CS61B+CS170** 697

UCB的本科课程CS61B和CS170是利用java语言对数据结构与算法进行了详细的讲… 来自： 一只小包子的博客

**Berkeley CS61B_Data_Structures Video Lecture** 1350

recorded from: http://webcast.berkeley.edu/course_details.php?seriesid=19069783… 来自： ramboisme的专栏

**CS61B sp2018笔记 | Lists** 187

Lists 1. IntLists 下面我们来一步一步的实现List类，首先你可以实现一个最简单… 来自： 隐秀_

下载 **final review for berkeley cs61b** 07-2

关于伯克利公开课cs61b的期末复习文档

**（原创笔记）加州伯克利大学CS61b数据结构（Java描述）一：对象** 2326

OOP(object-oriented programming): object: a repository of data; class: type of objec… 来自： Emacsor的博客

**CS61B Homework9作业回顾（附代码）** 57

本次作业主要学会运用Disjoin Sets，DisjointSets的class作业已经写好给出。class… 来自： everest115的博客

**cs61a spring 2018 Container笔记和补充 - CSDN博客**

个人分类: CS61A Spring 2018 版权声明:欢迎交流学习...separating items with commas: [a], [a, b, c...HW5—开…

**cs61a 2018 spring Lab4 Lists and Data Abstraction ..._CSDN博客**

cs61a 2018 spring Lab4 Lists and Data Abstraction...('Stanford', 34.05, 118.25) >>> closer_...term) return temp…

**CS61B sp2018笔记 | Introduction to Java** 126

Introduction to Java Essentials 1. Reading 1.1 Hello World 这门课程虽然是用Ja… 来自： 隐秀_

**文章热词** java记录目录树 javaweb 博客源码 java 用户登录挤下线 java开发直播平台 int数字比较 java

**相关热词** cs61b是什么 cs61b课程 cs61b作业 cs61b字幕

**LeetCode刷题笔记——LeetCode使用介绍** 2.8万

转载自http://blog.csdn.net/tostq 又到了一年毕业就业季了，三年前的校招季我逃… 来自： seabiscuityj的博客

麦哲思科技任甲林 关注 303篇文章

温柔狠角色 关注 264篇文章

zmycoco2 关注 444篇文章

**CS61B sp2018笔记 | Efficient Programming - 隐秀_ - CSDN博客**

05-14 Efficient C Programming Techniques.pdf Efficient...Spring IoC容器浅析及简单实现 Linux 基本操作总结(.….

**CS61A 系列课程笔记(一) - CSDN博客**

2018年05月06日 19:33:54 哈哈哈哈士奇VIP 阅读...a,b=0,1 ... for i in range(2,n+1): ....cs61a 2018 spring Lab4 …

下载 **CS61B 教材2 Algorithms 4th Edition** 04-10

CS61B 教材2 Algorithms 4th Edition - Robert Sedgewick and Kevin Wayne 高清英文原版

下载 **【伯克利CS61B教材——Java 学习教程】 Head First Java 中文版** 05-08

中文版的伯克利CS61B教材——经典Java 学习教程 《Head First Java 第二版 》

**cs61a课程总结--lecture7 递归（和一种数据结构）** 520

数据结构 tuple 是一种至类型 来自： vczhfan的专栏

**CS61A 系列课程笔记（一）** 👁 1338

嗯 今天刚看第二周的课程，大量的 reading 材料我是真的要崩溃了，全英。。。。 … 来自： 柠檬黄先生的博客

**C++实现控制台版2048（内附彩蛋）** 👁 124

前言　　之前做过一个JavaScript版本的2048游戏，最近在学习C++，昨天晚上突然… 来自： 隐秀_

**cs61a 2018 spring Lab4 Lists and Data Abstraction 笔记** 👁 89

list Lists are Python data structures that can store multiple values. list comprehensi… 来自： Siucaan

**Homework6** 👁 31

智能巡逻兵游戏规则游戏规则 使用WSAD或方向键上下左右移动player，进入巡逻兵… 来自： qq_36335897的…

**【深入浅出】| 基于深度学习的机器翻译（附PDF+视频下载）** 👁 1012

微信公众号关键字全网搜索最新排名【机器学习算法】：排名第一【机器学习】：排… 来自： 机器学习算法与…

**leetcode全部题目答案** 👁 1.3万

32. Longest Valid Parentheses Given a string containing just the characters '(' and ')… 来自： 陈善亮的博客

**LeetCode 答案(Easy)（1-100）** 👁 1984

7 . Reverse IntegerReverse digits of an integer.Example1: x = 123, return 321 Exa… 来自： CapMiachael的…

**漫画：如何实现抢红包算法?** 👁 1.2万

… 来自： 程序员小灰的博客

**微信红包的随机算法是怎样实现的?** 👁 1.9万

导语：今天看到有人问：关于微信红包的随机算法！就查阅资料看了一下“微信红包… 来自： 精彩人生从此刻…

**LeetCode51——N-Queens** 👁 1396

LeetCode51——N-Queens 求N皇后问题的所有解。 经典的算法，从大三刚开始学… 来自： 上善若水

**CS61B Homework4作业回顾（附代码）** 👁 122

这次作业没有测试代码，一亩地上有测试数据链接:http://pan.baidu.com/s/1dD5ql4P… 来自： everest115的博客

**java compareTo() 用法注意点** 👁 1.9万

转自：http://www.2cto.com/kf/201305/210466.html compareTo就是比较两个值，如… 来自： 风随星月

**compareTo返回值为-1 、 1 、 0 的排序问题** 👁 1.2万

首先，先看代码内容：（希望大家自己可以运行尝试，以加深记忆和理解）　packag… 来自： 狼郎

**达内课程-集合之LinkedList** 👁 401

达内课程-集合之LinkedList 来自： Errol's Blog

**Java中Collections.sort()排序详解** 👁 3.5万

java中Collections.sort排序详解java基础—— Collections.sort的两种用法，简单明了 … 来自： 薛瑄的博客

### 代码回顾：编程中一些经验性的问题

186

一、结构体编程：(1) 结构体初始化 今天在编程过程中。一个结构体定义 struct bss… 来自： Something of a…

### 下载 Data Structures Lecture Notes (UCB CS61b)

10-20

Data Structures Lecture Notes (UCB CS61b) Data Structures Lecture Notes (UCB CS61b)

### 一次git事故的回顾和学习

240

此次回顾是针对之前在上海开发过程中，吕鹏提交的代码被我覆盖掉的事故。希望通… 来自： 渣va搬砖路

### 下载 中国中文信息学会发布《2018知识图谱发展报告》

09-08

中国中文信息学会发布《2018知识图谱发展报告》报告目录如下（文末附下载链接） 第一章 知识表示与建模 第二章 知识表示学习 第三章 实体识别与链接 第四章 实体关系学习 第…

### 下载 Data structure and algorithms in java

05-29

CS61B课程中的配套课本，十分经典，涵盖数据结构和算法知识，以java语言的方式实现代码，赞！

### LeetCode总结

13.3万

最近完成了www.leetcode.com的online judge中151道算法题目。除各个题目有特殊… 来自： Allan的专栏

### LeetCode刷题指南（一）

8万

以下是我个人做题过程中的一些体会： 1. LeetCode的题库越来越大，截止到目前，… 来自： Lnho的专栏

### leetcode官网及使用

6594

官网:leetcode 来自： BoomMan

### 刷了两遍LeetCode之后，我拿了9个offer

2万

官方网站：http://leetcode.com/LeetCode是一个美国的在线编程网站，上面主要收集… 来自： haimianjie2012…

### LeetCode所有题目答案汇总

3623

最近再刷leetcode题目，找到了这个大神的博客总结，收藏一下 LeetCode All in One… 来自： 蘇丶的博客

### 回顾2016，工作总结！

740

在2016年里，还记得最初自己的工作态度并不是非常的好，随着工作时间的累积，… 来自： -JackoChan

### 团队敏捷实践：迭代回顾会议

2867

迭代回顾会议（不超过1个小时） 1. 上一次的行动完成的证据。没有完成要怎么办… 来自： xuesiyuan的专栏

### leetcode的开始—两数之和，c++

1401

给定一个整数数列，找出其中和为特定值的那两个数。你可以假设每个输入都只会有… 来自： prime_lee的博客

### 我们的敏捷之路——回顾会议

638

前言我们引入敏捷已经超过2年了，经过长时间的不断尝试逐渐摸索出一套适应于我… 来自： zhaoenweiex的…

### CS61B sp2018笔记 | Generics and Autoboxing

79

1. Automatic Conversions 1.1 Autoboxing and Unboxing Java中的泛型用到了&amp;… 来自： 隐秀_

### leetcode_812_ 最大三角形面积

194

给定包含多个点的集合，从其中取三个点组成三角形，返回能组成的最大三角形的面… 来自： Snow_Jie的博客

**LeetCode85——Maximal Rectangle** 1820

初看这道题有点不知所措，但是其实是LeetCode84——Largest Rectangle in Histogr… 来自： 上善若水

**LeetCode84——Largest Rectangle in Histogram** 1470

在柱状图中找到面积最大的矩形。 来自： 上善若水

**【LeetCode--数据库】分数排名** 1874

178. Rank Scores（中等） 题目：编写一个sql语句来实现分数排名，如果两个分数… 来自： 小白壮壮的博客

**Google面经，已拿到offer哦！** 2509

我面的职位是Softwre Engineer, Tools and Infrastracture, 所以开发和测试的问题都… 来自： UU小站

下载 **软件工程作业 现代银行业务系统 UML C++** 06-02

软件工程作业 内附用C++开发的代码、UML图及文档

**肖仰华 | 知识图谱研究的回顾与展望** 894

本文转载自公众号知识工场。 本文整理自2017年1… 来自： 开放知识图谱

vincent_chan7 关注

| 原创 | 粉丝 | 喜欢 | 评论 |
|---|---|---|---|
| 11 | 3 | 2 | 0 |

等级： 博客 2 访问： 991

积分： 118 排名： 126万+

勋章： 恒

最新文章

前端学习笔记-BBC网站复刻

CS61B Homework9作业回顾（附代码）

CS61B Homework8 作业回顾（附代码）

CS61B Homework7作业回顾 (附代码)

CS61B Homework6作业回顾（附代码）

归档

| | |
|---|---|
| 2018年5月 | 1篇 |
| 2018年4月 | 1篇 |
| 2018年3月 | 9篇 |

热门文章

CS61B Homework5作业回顾（附代码）
阅读量：406

CS61BHomework3作业回顾（附代码）
阅读量：127

CS61B Homework4作业回顾（附代码）
阅读量：122

CS61B Homework6作业回顾（附代码）
阅读量：92

CS61B Homework9作业回顾（附代码）
阅读量：57