# 赋值2

课程作业的价值**60%**:A部分占30%，B部分占30%
**个人工作**

## 学习成果

*学生将能够理解*
1.1数据结构
1.2数据结构的应用
1.3面向对象的程序设计概念
程序测试的方法
*学生将获得以下技能:*
2.1数据抽象
2.2数据结构的使用
2.3用高级面向对象语言进行更高级的编程
程序测试和文档编制
*学生将获得以下技能:*
3.1自我管理
3.2学习
3.3沟通
3.4解决问题
3.5信息技术

## 提交需求

提交的作业应压缩成。zip文件，压缩文件中应包含以下文件:

一份包含所有类文本的*Microsoft Word*文档格式的*报告*。文件名格式:*学生证*+CHC5223_CW2_Report.docx
· 一个包含项目的。zip*文件*:可运行的*jar*文件(如果可用)和所有程序的源文本(.java)，包括所提供的
文件名格式:学生证+*CHC5223_ CW2_Files.zip*

## Part A -二叉查找树

## 一般要求

所有的编程都必须符合“Java约定和编程指南”——参见模块Moodle网站。

你必须将所有类的源代码以文本或图像的形式粘贴到你的报告中。

## Introduction

这部分作业的题目是二**叉查找树**。
接口*IMemberDB*描述了保存成员对象数据库的抽象数据类型(ADT)的方法。

您必须将*IMemberDB*实现为分配2的二叉搜索树。

你**不能**对这些接口做任何更改。

# 需求

## *基本规则*

你必须在完成所有任务后创建一个可执行项目。
一个Java类应该分别在一个. Java文件中定义。

在报告中，每个任务的源代码，连同相应的解释，应该单独呈现。

不遵守这些规则将导致**零分**。

## *任务1*

您必须创建一个名为*MemberBST*的Java类来实现*IMemberDB接口*。
你必须使用二叉查找树，但它不需要*自平衡*。

你**不能**在提交中封装现有集合的实现。例如，你不能创建一个*TreeMap*对象并从你的类中调用该对象的方法。不遵守这一规定将导致该部分**分数**为**零**。

**提示:***使用String.compareTo*按字典顺序比较字符串。你可以把大写和小写当成不同的。(哈希码在这个赋值中没有位置。)

方法既可以迭代实现，也可以递归实现。你不能仅仅通过构建一个新的树来实现方法*remove*。
您可以使用提供的*remove方法的源代码*，基于*Oberon-2中的面向对象编程*，Hanspeter Mössenböck Springer-*Verlag* 1993，第78页，由David Lightfoot转录成Java(参见附录)。
*MemberBST*的构造函数必须将字符串“二叉搜索树”打印到*System.out*。

注意，线性查找的时间复杂度为*O(n)*，而应该使用二叉查找树，其时间复杂度为*O(logn)*。

**5分**

你必须在报告中给出你的设计和实现的清晰的原理和详细的解释。

**5分**

## *任务2*

你必须适当地使用断言(assert语句)来保护操作的前提条件。记住要为你的项目启用断言检查。
你必须在报告中对你的设计和实现给出清晰的原理和详细的解释。

**2马克**

## *任务3*

必须让你的类*记录*监控信息，可以保存在文本文件中，也可以调用*System.out.println*。
它必须记录(至少):

- 对于每个成员的*添加*(put)，对于每个成员的*检索*(get)，对于每个成员的删除(remove):
  - o*成员名称;*

- 访问树的节点序列。
- 将你的日志粘贴到你的报告中。

我们提供了一个主程序*CHC5223.java*供你在这个作业中使用。
文件的名称设置在静态变量*filename*中。
样本文件*sampleMembersUK.csv*和*sampleMembersUS.csv*每个文件包含500个这种格式的成员。

**3是**

你必须在报告中给出你的设计和实现的清晰的原理和详细的解释。

**3马克**

## *Task 4*

你必须为你的实现设计一个测试计划。一定要检查(在许多其他情况下)。

- 删除叶子节点能够正确地工作
- 删除一个后代节点能够正确地工作
- 删除一个有两个后代的节点能够正确地工作

**2标志**

你必须在报告中给出你的设计和实现的清晰的原理和详细的解释。

**3马克**

## *任务5*

通过使用提供的主程序或其他方法，你必须测试你的*MemberBST*。
您必须在报告中对您的测试给出明确的证据和详细的解释，包括您的测试计划、使用的测试数据、预期结果和实际结果。您必须显示您的实际结果和从您的日志文件或IDE的输出窗格中复制的日志信息。不要简单地说“测试通过了”，或者类似的。

**5马克**

## *任务6*

你必须对你实现的二叉查找树的*时间效率*和*空间*效率进行评论。

你必须在报告中给出明确的理由和详细的解释。

**2马克**

**总分30分**

# B部分-图形和寻路

这部分作业2的主题是图和寻路。

## 一般要求

所有的编程都必须符合“Java约定和编程指南”。
你必须将所有类的源代码粘贴到你的报告中，作为*文本*，**而不是**图像。
你必须只用数组实现所有必要的数据结构。

*在Java中编程时,通常使用Java类库中的集合类。然而,如果你需要用其他语言编程,这样的类,或它们的等价物,将不一定可用。*

### *任务1*

找到或设计一个交通网络——不一定是真实的，但必须是现实的。它必须是一个无向连通的图，没有环。它必须至少有**8个**节点。

每个节点都应该有一个名称(其中不能有空格)和*x*、*y*位置(0⩽x < 256和0⩽y < 256)，表示该节点在地图上以256 ×256网格(y向下递增)表示的大致位置。

链接必须包含它所连接的节点之间的距离信息。距离可以用合适的长度单位(如公里)或时间单位(如分钟)来测量。

它们之间的链接必须是这样的:有几对节点由不止一条路由连接起来。

你必须勾画出你的网络，并将其包含在你的报告中。草图必须显示每个节点标注其名称，并位于草图上的*x*和*y*位置。草图必须显示每个链接标注其距离。你可以手绘草图，但如果你这样做，你需要扫描它以包含在你的报告中。

**2标志**

### *任务2*

你必须使用以下语法将你的网络表示为文本文件:
“电台”*名称*x y

“链接”*站站距离*
每个站在*链接*线引用之前，必须先在*站*线上定义好。在你的报告中包含文本文件的内容。

**2标志**

### *任务3*

你必须实现Java类*StackInt, QueueInt, ListInt*和*SetInt*。这些将分别是抽象类*AbsStackInt, AbsQueueInt, AbsListInt, AbsSetInt的子类*，它们的来源在附录中提供和显示。这些被作为*抽象类*(被“子类化”)而不是作为*接口*(被“实现”)给出，部分原因是它们都是“有界的”，即由于被数组实现而具有有限的容量，同时也允许我们给你一些关于如何实现它们的提示。

类的方法所需的行为在注释中以前置条件和后置条件的形式表示。

您必须在报告中对您的设计和实现给出清晰的原理和详细的解释。

**4分**

## *Task 4*

必须适当使用断言(assert语句)来保护操作的前提条件。记住要为你的项目启用断言检查。

**1马克**

## *任务5*

通过使用*JUnit*或其他方式，您必须测试您的*StackInt, QueueInt, ListInt*和*SetInt的实现*。要做到这一点，创建每个类的对象，其中容量设置为一个较低的值，如10。您将在稍后部署类时将其设置为较大的值。
你必须在报告中对你的设计和实现给出清晰的证据和详细的解释，包括测试计划、使用的测试数据、预期结果和实际结果。您的实际结果必须以截图图像或从开发工具包的输出窗格中复制的文本的形式显示。

**4分**

## *任务6*

您必须创建一个Java类*StationInfo*来实现Java接口*IStationInfo*(已提供)。

**1马克**

## *任务7*

您必须创建一个Java类来定义*邻接矩阵*
final double NO_LINK = double . max_value;*//错误*的*double*。*MAX_VALUE* int numStations;// 0 <= numStations <= capacity

Double distance[][];
其中，对于所有i, j在0 <=数量<=容量

距离[i][j]为从编号为i的站点到编号为j的站点的距离距离[i][j]等于距离[j][i]

距离[i][i] = NO_LINK

## *任务8*

并定义一个*StationInfo*类的对象列表，保存在数组中。

## *任务9*

您必须在这个类中创建一个Java方法，该方法读取包含网络文本描述的文本文件，并构建相应的站点列表和链接矩阵。

你必须在这个类中执行尽可能多的有效性检查，并报告任何错误。在你的报告中包括这门课的文本。
你必须在报告中对你的设计和实现给出清晰的理由和详细的解释。

**2标志**

## *任务10*

在使用程序从文本文件中保存的数据构建列表和矩阵后，使用print语句显示它们的内容。将结果复制到你的报告中。
你必须在报告中给出你的设计和实现的清晰的原理和详细的解释。

**2标志**

## *任务11*

必须编写Java方法(如附录中所示)，以使用附录中给出的算法，从网络的给定节点执行*深度优先遍历*，并从同一节点执行宽度优先遍历。在你的报告中包括方法文本和由此产生的节点名称序列。

您必须在报告中对您的设计和实现给出清晰的原理和详细的解释。

**4分**

## *任务12*

您必须实现*Dijkstra算法*(如附录中所示)，利用您已经构造的数据结构来查找和显示网络中两个站点之间的最短路径。你还必须“测量”你的实现来计算while循环的迭代次数

你必须在报告中对你的设计和实现给出清晰的原理和详细的解释。

**4分**

## *任务13*

你必须给出Dijkstra算法和A*算法之间的区别的清晰的基本原理和详细的解释，并说明它们的行为在你的网络中会有什么不同。

**4分**

**总分30分**

## 获得帮助

我们鼓励您要求进一步澄清这项任务需要什么。请尽量在正常的联系时间做这件事，避免在截止日期前的最后一周请求这样的帮助。
你可以和其他人讨论作业中的要求和材料，但你创作的必须是你自己的作品。
注意避免合谋。

在报告中声明除模块教学团队以外的任何帮助。

## 反馈

除了我们在正常时间间隔内提供的书面反馈外，您将能够获得快速、简短、口头的形成性反馈，并帮助您在实践课上修改工作。

## Appendix: *remove*

```
private class Node {
   private Member data;
   private Node left, right;
   public Node(Member s) {data = s; left = null; right = null;}
} // Node

private Node root;

public Member remove(String name){
     // based on Object-Oriented Programming in Oberon-2
     // Hanspeter Mössenböck Springer-Verlag 1993, page 78
     // transcribed into Java by David Lightfoot

     // put assert statement for preconditions here

     Node parent = null, del, p = null, q = null;
     Member result;
     del = root;
     while (del != null && !del.data.getName().equals(name)) {
      parent = del;
      if (name.compareTo(del.data.getName()) < 0)
         del = del.left;
      else
        del = del.right;
     }// del == null || del.data.getName().equals(name))

   if(del != null) { // del.data.getName().equals(name)
     // find the pointer p to the node to replace del
     if (del.right == null) p = del.left;
     else if (del.right.left == null) {
        p = del.right; p.left = del.left;
     } else {
        p = del.right;
        while (p.left != null) {q = p; p = p.left;}
           q.left = p.right; p.left = del.left; p.right = del.right;
        }
        if(del == root) root = p;
        else if (del.data.getName().compareTo(parent.data.getName()) < 0)
           parent.left = p;
        else parent.right = p;
        // reduce size of tree by one
        result = del.data;
     }
     else result = null;

     return result;
   } // remove
```

# 抽象类AbsListInt

```
包CHC5223;//随便什么
/ * *
 *抽象类，由表示整型列表的类构成子类*
 你可以为此更改包名，但你不应该以任何其他方式修改它。
 *
 * /
抽象公共类AbsListInt {
   Protected int list[];
   Protected int size;// 0 <= size <=容量
   Protected final int容量;

   / * *
    * @param capacity——此列表的最大容量* @post创
    建了当前大小为0的新列表*/
   public AbsListInt(int capacity){
   //实现int值的有界列表this。Capacity =
   容量;
      这一点。Size = 0;
      List = new int[capacity];
   }

   public int getCapacity(){ 返 回 容 量 ;}
   public int getSize(){返回大小;}

   / * *
   * @param要添加的n个节点
   * @pre getSize() != getCapacity()
   * @post n已添加到list
   * /
   Abstract public void append(int n);
    / * *
    * @param x——要寻找的值
    * @pre true
    * @如果x在列表中则返回true */
   Abstract public Boolean contains(int x);

}
```

## 附录抽象类AbsQueueInt

包CHC5223;//随便什么

/ * *

*抽象类被表示整型栈的类子类*

你可以为此更改包名，但你不应该以任何其他方式修改它。

*

* /

抽象公共类AbsQueueInt {

```
protected int queue[];
protected int size; // 0 <= size <= capacity
protected final int capacity;

public AbsQueueInt(int capacity){
   this.capacity = capacity;
   this.size = 0;
   this.queue = new int[capacity];
}
```

public int getCapacity(){ 返 回 容 量 ;}public int getSize(){返回大小;}

/ * *

* @param要添加的n个节点

* @pre getSize() != getCapacity()

* @post n已经添加到队列的后面

* /

abstract public void addToBack(int n);

/ * *

@pre getSize() != 0

* @队列前面的post元素被移除* @被移除的返回值*/ abstract public int removefromFront();

}

## 抽象类AbsSetInt

```
包CHC5223;//随便什么
/ * *
 *抽象类被表示整型栈的类子类*
 你可以为此更改包名，但你不应该以任何其他方式修改它。
 *
 * /



抽象公共类AbsSetInt {

    Protected int set[];
    Protected int size;// 0 <= size <= capacity
    protected final int capacity;

     / * *
     * @param capacity——此队列的最大容量
     * @预容量>= 0
     * @post创建了当前大小为0的新集合
     * /
    public AbsSetInt(int capacity){
        这一点。容量=容量;
        这一点。Size = 0;
        这一点。Set = new int[容量];
    }
    public int getCapacity(){返回容量;}
    public int getSize(){返回size;}

    / * *
     * @param x——要寻找的值
     * @pre true
     * @return true iff x在列表中*/
    Abstract public Boolean contains(int x);

    / * *
     * @param要添加的n个节点
     @ precontains (n) || getSize() != getCapacity()
     * @post包含(n)
     * /
    Abstract public void include(int n);

     / * *
     * @pre true
     * @post !包含(n)
     * /
     Abstract public void exclude(int n);

}
```

# 抽象类AbsStackInt

```
包CHC5223;//随便什么
/ * *
 *抽象类被表示整型栈的类子类*
 你可以为此更改包名，但你不应该以任何其他方式修改它。
 *
 * /


抽象公共类AbsStackInt {

   Protected int stack[];
   Protected int size;// 0 <= size <=容量
   受保护的最终int容量;

    / * *
    * @param capacity——此队列的最大容量
    * @预容量>= 0
    * @post当前大小为0的新堆栈已经创建
    * /
   public AbsStackInt(int容量){
      这一点。容量=容量;
      这一点。Size = 0;
   }  Stack = new int[容量];
   public int getCapacity(){返回容量;}
   public int getSize(){返回size;}

   / * *
    * @param要添加的n个节点
    * @pre getSize() != getCapacity()
    * @post n已经被压入栈顶
    * /
   Abstract public void push(int n);

    / * *
    @pre getSize() != 0
    * @post栈顶元素已被移除
    * @被删除的返回值*/
    Abstract public int pop();

   / * *
    @pre getSize() != 0
    * @栈顶返回值*/
   Abstract public int peek();

}
```

# Interface IStationInfo

```
 * Interface to be implemented by class(es) that represent
 * information about stations
 *
 * You may change the package name for this, but you should not
 * modify it in any other way.
 *
 */
public interface IStationInfo {

   /**
    * @return the name of the station
    */
   String getName();

   /**
    * @return x position -- 0 <= getxPos() < 256
    */
   int getxPos();

   /**
    * @return y position -- 0 <= getyPos() < 256
    */
   int getyPos();
}
```

## Appendix: breadth-first traversal, beginning with a specified start station

```
Let Q be an empty queue of Stations
Let L be an empty list of Stations
Add the start station to the back of Q
while Q is not empty do
    Remove the Station at the front of Q, call this Station S
    if S is not in list L then
            Append S to L
            for each station S2 that is adjacent to S do
                    if S2 is not in the list L then
                            Add S2 to the back of queue Q
                    endif
            endfor
    endif
endwhile
return L
```

## Appendix: depth-first traversal, beginning with a specified start station.

```
Let S be a stack of Stations
Create an initially empty list of Stations, which we will call L
Push the start station on to the stack S
while S is not empty do
    Pop the Station at the top of the stack. Call this Station T.
    if T is not already in list L then
            add it to the back of that list
    endif
    for each station T2 that is adjacent to T do
            if T2 is not in the list L then
                    push it on to the top of the stack S
            endif
    endfor
endwhile
return L
```

## Appendix: Dijkstra's algorithm for shortest path in a network

```
set Closed to be empty
add all nodes in the graph to Open.
set the g-value of Start to 0, and the g-value of all the other nodes to ∞
set previous to be none for all nodes.
while End is not in Closed do
   let X be the node in Open that has the lowest g-value (highest priority)
   remove X from Open and add it to Closed.
   if X is not equal to End then
      for each node N that is adjacent to X in the graph, and also in Open do
         let g' = g-value of X + cost of edge from N to X
         if g' is less than the current g-value of N then
            change the g-value of N to g'
            make N's previous pointer point to X
         endif
      endfor
   endif
endwhile
```

通过跟随“前一个”指针找到到终点的前一个节点，前一个节点到那个前一个节点，以此类推，重构从开始到结束的最短路径。