CS 2210 Data Structures and Algorithms
Second Programming Assignment

Due Date: November 19, 11:55 pm
Total marks: 20

In this assignment you will implement a multimedia ordered dictionary that allows repeated keys using a binary search tree.

1 Classes to Implement

You need to implement the following Java classes: `MultimediaItem`, `NodeData`, `BSTNode`, `BSTOrderedDictionary`, and `Query`. You can implement more classes if you need to. **You must write all the code yourself**. You **cannot** use code from the textbook, the Internet, or any other sources: however, you may implement the algorithms discussed in the lectures.

1.1 Class BSTNode

This class represents the nodes of the binary search tree implementing the ordered dictionary. This class has the following instance variables:

- `BSTNode parent`: a reference to the parent of this node
- `BSTNode leftChild`: a reference to the left child of this node
- `BSTNode rightChild`: a reference to the right child of this node
- `NodeData data`: a reference to the information stored in this node.

This class must have the following two constructors:

- `BSTNode()`: creates a new BSTNode object in which all its instance variables have value null. This constructor is used when creating a leaf node.
- `BSTNode(BSTNode newParent, BSTNode newLeftChild, BSTNode newRightChild, NodeData newData)`: creates a new BSTNode in which the instance variables take the values specified in the corresponding parameters.

For this class you must implement all and only the following `public` methods:

- `BSTNode getParent()`: returns the parent of this node
- `BSTNode getLeftchild()`: returns the left child of this node
- `BSTNode getRightChild()`: returns the right child of this node
- `NodeData getData()`: returns the NodeData object stored in this node
- `setParent(BSTNode newParent)`: sets the parent of this node to the specified value
- `setLeftChild(BSTNode newLeftChild)`: sets the left child of this node to the specified value
- `setRightChild(BSTNode new rightChild)`: sets the right child of this node to the specified value

- `setData(NodeData newData)`: stores the given NodeData object in this node
- `boolean isLeaf()`: returns true if `this` node is a leafl returns false otherwise.

You can implement any other methods that you want to in this class, but they must be declared as `private` methods (i.e. not accessible to other classes).

1.2 Class NodeData

This class represents the data items that will be stored in the internal nodes of the binary search tree implementing the ordered dictionary. Remember that in a leaf node the data and children are null, but the parent is not null, unless the leaf node is the root of the tree. Each object of this class will have two instance variables:

- `String name`: this is the key attribute for the data stored in a node.
- `ArrayList<MultimediaItem> media`: list of multimedia objects associated with the key attribute of this node.

The constructor for this class must be of the following form:

- `NodeData(String newName)`: creates the new NodeData object with the given key attribute and an empty media list.

For this class you must implement all and only the following `public` methods:

- `void add(MutimediaItem newItem)`: adds newItem to the media list of this object
- `String getName()`: returns the `name` attribute of this object.
- `ArrayList<MultimediaItem> getMedia()`: returns the media list stored in this object.

You can implement any other methods that you want to in this class, but they must be declared as `private` methods.

1.3 Class MultimediaItem

This class represents each one of the multimedia objects stored in a node of the binary search tree. This class has two instance variables:

- `String content`: a descriptor of the multimedia content. This could be text, the name of an audio file, the name of an image file, or the name of an html document.
- `int type`: the type of information represented by instance variable `content`. The values that this instance variable can take are these:
  - 1: if `content` is text
  - 2: if `content` is the name of an audio file
  - 3: if `content` is the name of an image file
  - 4: if `content` is the name of an html document.

The constructor for this class must be of the following form:

- `MultimediaItem(String newContent, int newType)`: creates the new `NodeData` object whose instance variables have the values specified by the parameters.

For this class you must implement all and only the following `public` methods:

- `String getContent()`: returns the `content` of this node.
- `int getType()`: returns the `type` of this node

You can implement any other methods that you want to in this class, but they must be declared as `private` methods.

1.4 Class BSTOrderedDictionary

This class implements an ordered dictionary using a binary search tree. In the binary search tree **only the internal nodes will store information**. The key for an internal node will be the `name` attribute of the `NodeData` object stored in that node. To implement this class, you need to declare it as follows:

`public class BSTOrderedDictionary implements OrderedDictionaryADT}`

This class has these instance variables:

- `BSTNode root`: the root of the binary search tree.
- `int numInternalNodes`: the number of internal nodes in this tree.

The constructor for this class must be of the following form

- `BSTOrderedDictionary()`: creates an empty `BSTOrderedDictionary` in which the `root` is a leaf `BSTNode` and `numInternalNodes = 0`.

In this class you must implement all the `public` methods specified in the `OrderedDictionaryADT` interface:

- `BSTNode getRoot()`: returns the root of this tree.
- `int getNumInternalNodes()`: returns the value of `numInternalNodes`.
- `ArrayList<MultimediaItem> get (BSTNode r, String key)`: finds in the tree with root `r` the `BSTNode` with the given key attribute `key` and it returns the list of `MultimediaItem` objects stored in it; it returns null if no node in the tree stores the given key.
- `void put (BSTNode r, String name, String content, int type)`: adds the given information to the tree with root `r`. We will allow the tree to store two data items with the same key attribute: If the tree already has a node p with key attribute equal to `name` then a new `MultimediaItem` object storing the given `content` and `type` is added to the `ArrayList` stored in p. If the tree does not have a node storing `name` then a new internal node is added to the binary search tree storing a `NodeData` object with key attribute `name` and an `ArrayList` storing a `MultimediaItem` object containing the given `content` and `type`.

  if a new internal node is created the value of `numInternalNodes` must be increased;
- `void remove(BSTNode r, String key) throws DictionaryException`: removes from the binary search tree with root `r` the `BSTNode` storing the given key attribute `key`. The method throws a `DictionaryException` if no node stores the given `key`.

  If an internal node was removed, the value of `numInternalNodes` must be decreased.

- `void remove(BSTNode r, String key, int type) throws DictionaryException`: removes from the tree with root `r` all `MultimediaItem` objects stored in the `ArrayList` of the node with key attribute `key` with the given `type`. If after removing these `MultimediaItem` objects the `ArrayList` becomes empty, the `BSTNode` with key attribute `key` must be removed from the tree.

  If no node has key attribute `key` then this method must throw a `DictionaryException`. If an internal node was removed, the value of `numInternalNodes` must be decreased.

- `NodeData smallest(BSTNode r)`: returns the `NodeData` object storing the smallest key in the tree with root `r`; returns null if the tree is empty.

- `NodeData largest(BSTNode r) throws DictionaryException`: returns the `NodeData` object storing the largest key in the tree with root `r`; returns null if the tree is empty.

- `NodeData successor (BSTNode r, String key)`: returns the `NodeData` objects stored in the successor of the node storing key attribute `key` in the tree with root `r`; returns null if the successor does not exist.

- `NodeData predecessor (BSTNode r, String key)`: returns the `NodeData` objects stored in the predecessor of the node storing key attribute `key` in the tree with root `r`; returns null if the predecessor does not exist.

You can implement any other methods that you want to in this class, but they must be declared as `private` methods.

1.5 Class Query

This class implements a text-based application that allows the user to access the information in the multimedia ordered dictionary. This class contains the `main` method, declared with the usual method header:

```
public static void main(String[] args)
```

The input to the program will be a file containing the information that is to be stored in the ordered dictionary. Therefore, to run the program you will type this command:

`java Query` *inputFile*

where *inputFile* is the name of the file containing the input for the program. Your `main` method must read the input file and it will store the information stored in the input file in the binary search tree specified in the previous section. Section 1.5.1 specifies the format of the input file.

After reading the input file and creating the corresponding `BSTOrderedDictionary` your `main` method will have a loop in which it will repeatedly ask the user to enter a command which the program will process as explained below, until the user enters the command `end` that will terminate the program.

To read user commands you **must** use method

```
String read (String label)
```

from the provided `StringReader` class. It is very important that you use this method, as otherwise the TA's will not be able to test your program. The above method prints on the screen the label supplied as parameter and then it reads one line of input from the keyboard; the line read is returned

as a String to the invoking method. So, for example, to ask the user to type a command you might use this code fragment in your program:

```
StringReader keyboard = new StringReader();
String line = keyboard.read("Enter next command:  ");
```

1.5.1 User Commands

The commands that the user can enter and which your program must process are explained below.

- `get k`

  If the `BSTOrderedDictionary` has a node p with key attribute `k`, then each one of the `MultimediaItem` objects stored in the `ArrayList` stored in that node will be processed in the following manner. Let `D` be one of these `MultimediaItem` objects, then

  - if `D.type = 1`, then your program must print `D.content` on the screen,
  - if `D.type = 2`, then your program must play the audio file whose name is stored in `D.content`,
  - if `D.type = 3`, then your program must display the image stored in the file whose name is stored in `D.content`.
  - if `D.type = 4` , then your program must display the content of the web page specified in `D.content`.

  If there is no node with key attribute `k` then the program must print the message:

  `"Key k is not in the ordered dictionary"`

  and then it must print (i) the key attribute of the predecessor of p (if p has a predecessor), and (ii) the key attribute of the successor of p (if p has a successor).

  So, for example, consider an ordered dictionary storing the following `NodeData` objects. We denote a `NodeData` object as `[key,<(content1,type1),(content2,type2),...>]`.

  - `["computer",<("An electronic machine frequently used by Computer Science students",1),("computer.gif",3)>]`
  - `["flower",<("http://www.csd.uwo.ca/flower.html",4)>]`
  - `["tree",<("tree.jpg",3),("tree.gif",3)>]`
  - `["ping",<("ping.wav",2)>]`
  - `["matrix",<("matrix.gif",3)>]`

  if in the above ordered dictionary the user enters the command `get ping` your program must play the file "ping.wav". If the user enters the command `get computer`, your program it must print the text "An electronic machine frequently used by Computer Science students" and it must display the image in file "computer.gif". For the command `get flower` your program must display the content of the webpage "http://www.csd.uwo.ca/flower.html".

  If the user enters the command `get homework` your program should print the following:

  ```
  The word homework is not in the ordered dictionary.
  Preceding word:  flower
  Following word:  matrix
  ```

  If the user enters the command `get abacus` your program should print:

  ```
  The word abacus is not in the ordered dictionary.
  Preceding word:
  Following word:  computer
  ```

- `remove k`

  Removes from the ordered dictionary the node with key attribute `k`, or if no node stores this key your program must print

  `No record in the ordered dictionary has key k.`

- `add k c t`

  If no node in the tree stores key `k` then a new node p with that key is added to the tree; otherwise your program finds the node p storing key k. A `MultimediaItem` string content `c` and type `t` is added to the `ArrayList` of p.

- `next k d`

  This command must find the node p with key attribute `k` or, if this node does not exist, the leaf node p where the key attribute `k` should have been stored. Then your program must find the `d` successor nodes of p (if they exist) and print the key attributes of all these nodes in lexicographic increasing order.

  For example, for the ordered dictionary described above, if the user enters `next computer 2`, your program must print

  `computer flower matrix`

  If the user enters `next ab 1`, your program must print

  `computer`

  If the user enters `next ping 3`, your program must print

  `ping tree`

  If the user enters `next set 2`, your program must print

  `"There are no keys larger than or equal to set"`

- `prev k d`

  This command must find the node p with key attribute `k` or, if this node does not exist, the leaf node p where the key attribute `k` should have been stored. Then your program must find the `d` predecessor nodes of p (if they exist) and print the key attributes of all these nodes in lexicographic decreasing order.

  For example, for the ordered dictionary described above, if the user enters `prev computer 2`, your program must print

  `computer`

  If the user enters `prev ab 1`, your program must print

  `"There are no keys smaller than or equal to set"`

  If the user enters `prev ping 3`, your program must print

  `ping matrix flower computer`

- `first`

  This command must print the smallest key attribute in the ordered dictionary. For example, for the above ordered dictionary the command `first` must print: `computer`.

  If the ordered dictionary is empty your program must print

  `"The ordered dictionary is empty"`

- `next k d`

  This command must find the node p with key attribute `k` or, if this node does not exist, the leaf node p where the key attribute `k` should have been stored. Then your program must find the `d` successor nodes of p (if they exist) and print the key attributes of all these nodes in lexicographic increasing order.

  For example, for the ordered dictionary described above, if the user enters `next computer 2`, your program must print

  ```
  computer flower matrix
  ```

  If the user enters `next ab 1`, your program must print

  ```
  computer
  ```

  If the user enters `next ping 3`, your program must print

  ```
  ping tree
  ```

  If the user enters `next set 2`, your program must print

  ```
  "There are no keys larger than or equal to set"
  ```

- `last`

  This command must print the largest key attributes in the ordered dictionary. For example, for the above ordered dictionary the command `last` must print: `tree`.

  If the ordered dictionary is empty your program must print

  `"The ordered dictionary is empty"`

- `size`

  This command prints the number of internal nodes in the binary search tree implementing the ordered dictionary. For example, for the above ordered dictionary the command `size` must print

  `There are 5 keys in the ordered dictionary"`

- `end`

  This command terminates the program.

- If an invalid command is entered your program must print the message

  `"Invalid command"`

1.5.2 Format of the Input File

The input file is a text file. The first line contains a string; this is the key attribute of the first `NodeData` object t to be stored in the ordered dictionary. The second line is a string s that is processed as follows:

- If s is the name of an audio file then a `MultimediaItem` object with attributes `content =` s and `type = 2` is created and stored in the `ArrayList` of t.
- If s is the name of an image file then a `MultimediaItem` object with attributes `content =` s and `type = 3` is created and stored in the `ArrayList` of t.
- If s is the name of an html file then a `MultimediaItem` object with attributes `content =` s and `type = 4` is created and stored in the `ArrayList` of t.
- Otherwise, a a `MultimediaItem` object with attributes `content =` s and `type = 1` is created and stored in the `ArrayList` of t.

A node storing t is created and set as the root of the tree.

Each pair P of lines in the rest of the input file is processed in the same manner: the first line in the pair specifies a key attribute and the second line specifies the content of a `MultimediaItem` object. Note that if a node already exists with the same key attribute as specified by the first line in a pair, then no new `BSTNode` is created; instead the `MultimediaItem` corresponding to the second line in the pair is added to the `ArrayList` of that node.

If the second line of a pair P of lines from the input file contains only one string of the form `x.y`, then

- if `y = "wav"` or `y = "mid"` then `type = 2`. We will consider only audio files with these two extensions.
- if `y = "jpg"` or `y = "gif"` then `type = 3`. We will consider only these two types of image files.

- if `y = "html"` then `type = 4`.

For example, the above ordered dictionary could be constructed with the following input file:

```
matrix matrix.gif computer An electronic machine frequently used by Computer
Science students flower http://www.csd.uwo.ca/flower.html tree tree.jpg tree
tree.gif computer computer.gif ping ping.wav
```

1.5.3 Important Notes and Classes Provided

- If the main method is `public static void main(String[] args)`, then the name of the input file will be stored in `args[0]`.
- The key attribute of each node must be converted to lower case before being stored in the dictionary, so that capitalization does not matter when looking for a key in the dictionary.
- To play an audio file you must use the provided Java class `SoundPlayer.java`; you will use method `play(String fileName)` to play the named audio file.
- To display an image, you must use the provided Java class `PictureViewer.java`; you will use method `show(String fileName)` to display a `.jpg` or `.gif` file.
- To display a webpage you must use the provided Java class `ShowHTML.java`; you will use method `show(String url)` to render the page on the screen.
- A `MultimediaException` will be thrown by methods `play`, `run` and `show` of classes `SoundPlayer.java`, `PictureViewer.java`, and `Show HTML.java` if the named files cannot be found or if they cannot be processed. Your program must catch these exceptions and print appropriate messages.

  We provide you with a java class called `Sample.java` that illustrates how to use methods `play`, `run` and `show` from the above classes. Study this class so you know how to use these methods.
- If you use Eclipse, to ensure that it will find all the image, sound, html, and txt files, put all these files in the same directory; then in Eclipse go to Run, Run Configurations, select Arguments, on Working directory select "Other" and click on "File System" and choose the directory that contains your files.

Hint. You might find useful the StringTokenizer Java class and methods `toLowerCase()` and `endsWith(String prefix)` from class String.

2 Testing your Program

We will run a test program called `TestDict` to check that your implementation of `OrderedDictionary` has the properties specified above. We will supply you with a copy of `TestDict` to test your implementation. We will also run other tests on your software to check whether it works properly.

3 Coding Style

Your mark will be based partly on your coding style. Among the things that we will check, are

- Variable and method names should be chosen to reflect their purpose in the program.

- Comments, indenting, and white spaces should be used to improve readability.
- No instance variable should be used unless they contain data which is to be maintained in the object from call to call. In other words, variables which are needed only inside methods, whose values do not have to be remembered until the next method call, should be declared inside those methods.
- All instance variables should be declared `private`, to maximize information hiding. Any access to these variables from other classes should be done with accessor methods.

4 Marking

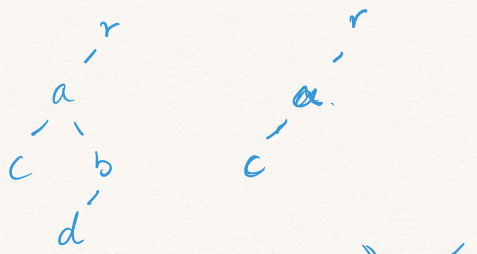
Your mark will be computed as follows.

- Program compiles, produces meaningful output: 2 marks.
- `TestDict` tests pass: 4 marks.
- `Test Query` tests pass: 4 marks
- Coding style: 2 marks.
- Ordered Dictionary implementation: 4 marks.
- `Query` program implementation: 4 marks.

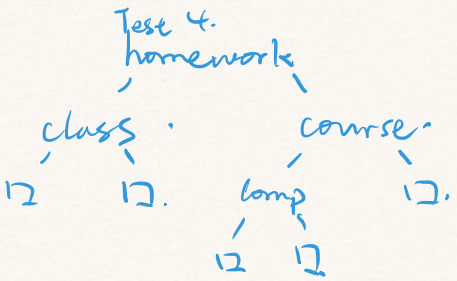
5 Submitting Your Program

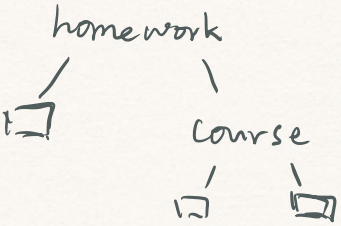
You are required to submit an electronic copy of your program through OWL. **Please do not put your code in sub-directories**. Delete any `package` lines at the top of your java classes as using packages makes it harder for the TAs to mark the assignments. **Please do not compress your files**.

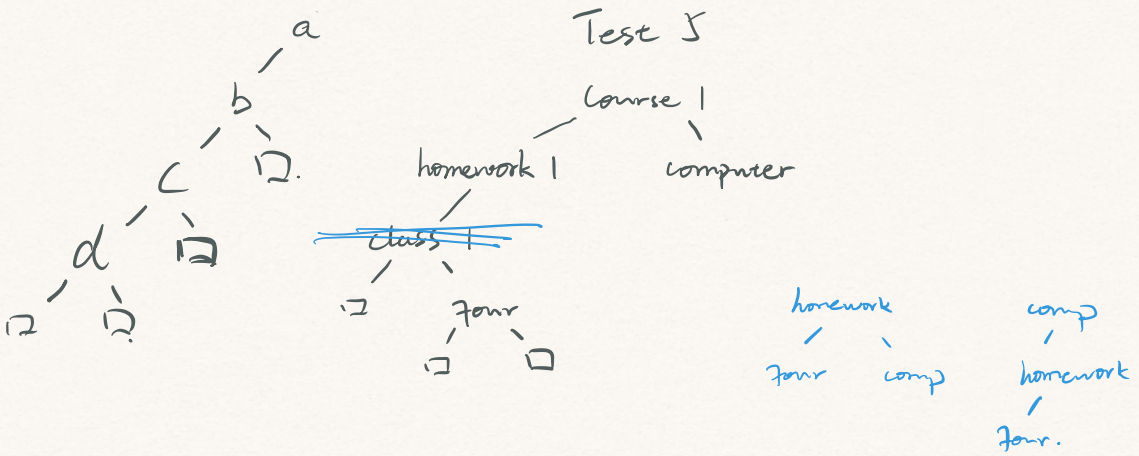
If you submit your program more than once, we will take the latest program submitted as the final version, and we will deduct marks accordingly if it is late.

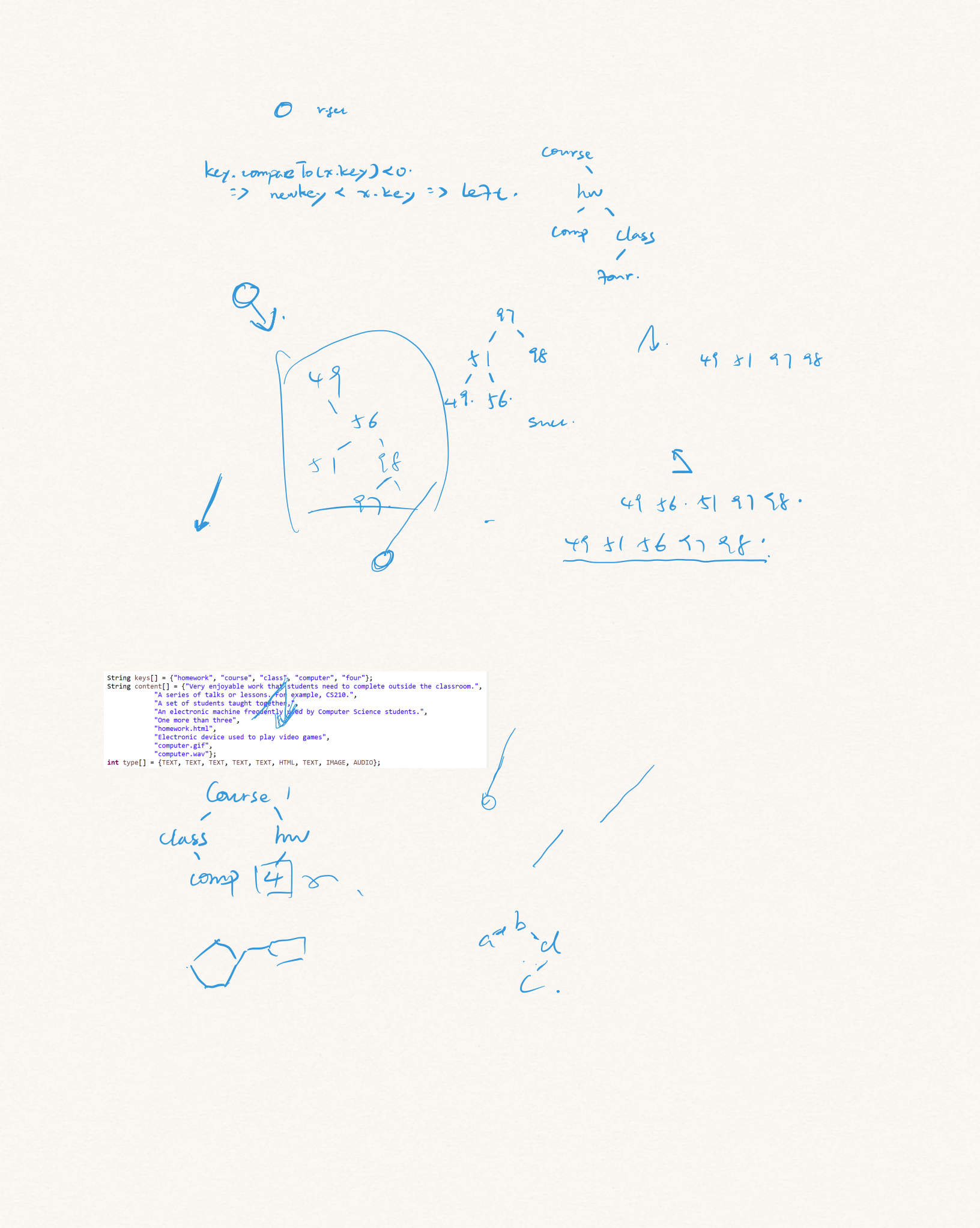


largest left child.
^
r
^
smallest right child.










```
String keys[] = {"homework", "course", "class", "computer", "four"};
String content[] = {"Very enjoyable work that students need to complete outside the classroom.",
            "A series of talks or lessons. For example, CS210.",
            "A set of students taught together.",
            "An electronic machine frequently used by Computer Science students.",
            "One more than three",
            "homework.html",
            "Electronic device used to play video games",
            "computer.gif",
            "computer.wav"};
int type[] = {TEXT, TEXT, TEXT, TEXT, TEXT, HTML, TEXT, IMAGE, AUDIO};
```