The University of Saskatchewan

Saskatoon, Canada

Department of Computer Science

CMPT 280– Intermediate Data Structures and Algoirthms

Assignment 3

Date Due: February 6, 2018, 10:00pm

Total Marks: 37

1 Submission Instructions

- Assignments must be submitted using Moodle.
- Responses to written (non-programming) questions must be submitted in a PDF file, plain text file (`.txt`), Rich Text file (`.rtf`), or MS Word's `.doc` or `.docx` files. Digital images of handwritten pages are also acceptable, provided that they are **clearly** legible.
- Programs must be written in Java.
- If you are using IntelliJ (or similar development environment), do not submit the workspace (project). Hand in only those files identified in Section 5. Export your `.java` source files from the workspace and submit only the `.java` files.
- No late assignments will be accepted. See the course syllabus for the full late assignment policy for this class.

2 Background

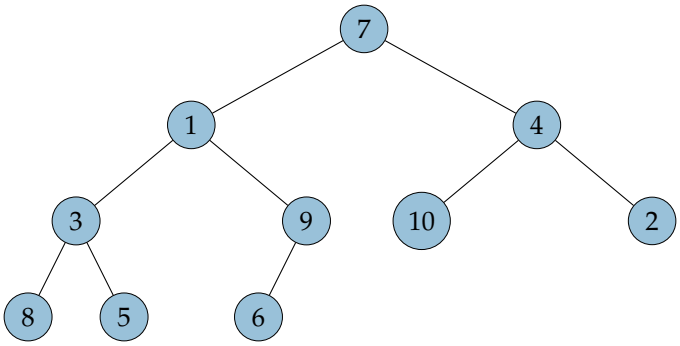
2.1 Arrayed Trees

Question 1 is about a bounded binary tree implementation. You should remember binary trees from CMPT 145 (or similar course) – they are trees in which each node has at most two children. What you probably didn't know is that binary trees can be stored using an array, rather than a linked structure. In such an array, the contents of the root node are stored in offset 1 of the array (offset 0 is unused). The contents of the children of the node whose contents are stored at offset i are stored at offset $2i$ and $2i + 1$, respectively. Thus, the left child of the root is at offset $2 \times 1 = 2$, the right child of the root is at offset $2 \times 1 + 1 = 3$, the left child of the left child of the root is at offset $2 \times 2 = 4$, and so on. The parent of the node whose contents are at offset i, is at offset $i/2$ (integer division). Thus, the parent of node at offset 7 is at offset 3.

Example 1
Here is the array representation of a tree storing the elements 1 through 10, in no particular order. The array

-	7	1	4	3	9	10	2	8	5	6

is a representation of the tree:



Note that we do not allow any unused entries in the array between used ones. All the data items in the array are stored contiguously. This means that we can represent only a particular subset of binary trees with this representation. Namely, it is the set of trees where all levels except possibly the last level are complete (full) and the nodes in the bottom level are all as far to the left as possible. You might be thinking that this is too restrictive and not very useful because we can't represent **all** binary trees with this data structure. However, as we will see on future assignments, this array-based tree data structure is highly useful and efficient as a basis for implementing certain other important data structures.

Also note that if we read off the items from left to right in each level of the tree, starting from the top level, we get the items in the same order as they appear in the array.

2.2 m-ary Trees

An m-ary tree is one in which a node may have up to m children. Your `lib280-asn3` library has a class called `BasicMAryTree280<I>` which implements an m-ary tree. It has some similarities with `LinkedSimpleTree280<I>` in that, like `LinkedSimpleTree280<I>`, you have to build up larger trees from smaller trees, rather than inserting individual elements, because since m-ary trees have no defined structure in general and thus there is no obvious algorithm for automatically deciding where a new element should go. You will use this class in Question 2. More details and some examples on how to use this class are/were provided in Tutorial 3.

3 Your Tasks

Question 1 (20 points):

Your task is to write a Java class called `ArrayedBinaryTreeWithCursors280<I>` which extends and implements the abstract class `ArrayedBinaryTree280<I>` (provided in the `lib280-asn3.tree` package as part of `lib280-asn3`). This week's lab will also talk more about array-based trees.

Methods to be Implemented

Some of the work of implementing `ArrayedBinaryTreeWithCursors280<I>` has already been done, but there is more to do.

Firstly, there are several methods in `ArrayedBinaryTreeWithCursors280<I>` which are defined but not implemented. You must implement these methods. These methods can be easily identified by their "TODO" comments.

Secondly, since `ArrayedBinaryTreeWithCursors280<I>` implements some other interfaces, there are several methods required by these interfaces that also need to be implemented but have not yet been finished. The method headers for these methods have already been generated but the method bodies are empty. The definitions of these methods in their respective interfaces document what these methods are supposed to do. These methods can be identified by their "TODO" commands as well as the `@Override` directive above the method headers. Many of these methods are defined by `LinearIterator280`, which is inherited through `Dict280<I>`. These are the same linear iterator methods that you wrote for `LinkedList280<I>` on assignment 1. The rest of these methods are defined by `Cursor280<I>`.

There is already a regression test included in `ArrayedBinaryTreeWithCursors280<I>`. Your completed implementation of the arrayed binary tree must pass the given regression test. If all the regression tests are successful, the only output should be: `Regression test complete`.

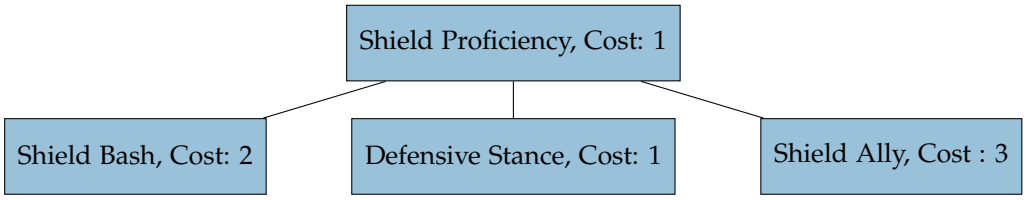
You may not modify any of the existing code in the provided `ArrayedBinaryTreeWithCursors280.java` **file** (including the regression test) but you can add to it. **You may also not modify any other files within** `lib280-asn3`.

Implementation Hints

- You don't need to declare an array instance variable in `ArrayedBinaryTreeWithCursors280<I>` to hold the data items. There is already one inherited from `ArrayedBinaryTree280<I>`. You should look at the other inherited instance variables too!
- One of your first tasks will be to start implementing the `insert` method and decide where the new element should be inserted. If you think about it, there's really only one place it can go...
- The algorithm for deleting an element is to replace the element to be deleted by the right-most element in the bottom level of the tree, then delete the right-most element in the bottom level of the tree.
- Not sure how a linear iterator works on a tree? If you think about it, there is only one reasonable way to define a linear ordering on the elements of an array-based binary tree.
- Reminder: the `items` array (defined in the abstract class `ArrayedBinaryTree280<I>`) represents the nodes of the tree. You are storing the **contents** of nodes in the array. There is no node class. It is very important that the contents of the root are stored in offset 1 and we don't use offset 0 of the array, otherwise, the given formulae for finding the child or parent of a node at offset i will not work correctly.

Question 2 (17 points):

In video games, especially those in the roleplaying genre, it is common that characters in the game are advanced in power through the use of a *skill tree*. Generally, a skill tree defines the prerequisite for the various skills that your character in the game might acquire. For example, in a hypothetical game, if the *Shield Bash*, *Defensive Stance*, and *Shield Ally* skills all require that your character first have the skill *Shield Proficiency*, then this might be represented by the following skill tree:



More formally, a skill in the skill tree can only be gained if the character first gains all of the skills which are ancestors of that skill in the tree.[1]

Your task in this question is to write a class called `SkillTree` which extends `BasicMAryTree280<Skill>` (an m-ary tree of `Skill` objects; a complete `Skill.java` is provided). A template for the `SkillTree` class is provided. It contains a constructor and a couple of useful methods. You will add additional methods to this class in the following steps, which you should complete in order:

(a) Write a `main()` method in the `SkillTree` class in which you construct your own skill tree for your own hypothetical video game. Your tree must contain at least 10 skills. However, for the sanity of everyone involved, try to keep it under 15 skills. Be creative! There is no reason why any two students should hand in exactly the same (or even very similar) skill trees, nor should you just duplicate the skill tree shown in the sample output. Print your tree to the console using the `toStringByLevel()` method inherited from `BasicMAryTree280`.

(b) Write a method in the `SkillTree` class called `skillDependencies` which takes a skill name as input and returns an instance of `LinkedList280<Skill>` which contains all the of the skills which are prerequisites for obtaining the input skill (including the input skill itself!). A `RuntimeException` exception should be thrown if the tree does not contain the given skill. A good implementation approach for this method is to use a recursive traversal of the tree to find the named skill, and then add skills to the output list as the recursion unwinds. Tutorial 3 includes some discussion of recursive traversal of m-ary trees. Add to your `main()` program a few tests of this method, and print out the lists that is returned (you can use the list's `toString()` method for this). Be sure to test the case where the named skill does not exist in the tree.

(c) Write a method in the `SkillTree` class called `skillTotalCost` which takes a skill name as input and returns the total number of skill points that a player must invest to obtain the given skill. If the named skill is not in the skill tree, then the `skillTotalCost` method should throw a `RuntimeException` exception. *Hint: this method is quite easy to implement if you make use of the previously implemented* `skillDependencies` *method.*

For example, in the above skill tree, if a character wants the *Shield Ally* skill they would need to spend 1 skill point to get *Shield Proficiency*, and then spend 3 skill points to get *Shield Ally* for an overall investment of $1 + 3 = 4$ points, so for the above tree, `skillTotalCost("Shield Ally")` should return 4. Note that the `Skill` object contains the cost of the skill.

Add to your `main()` program a few tests of `skillTotalCost`, and print out the total costs returned. Be sure to test the case where the named skill does not exist in the tree.

(d) Run your `main()` program. Cut and paste the console output to a text file and submit it with your assignment. See the sample output on the next page.

[1] In the video game world, the term "skill tree" sometimes refers to things that actually aren't trees; a noteworthy example is the skill tree in the ARPG Path Of Exile, which, if you click the link, can see is clearly **not** a tree, even though they call it that. **Here in question 2, we used the term "skill trees" to mean skill trees that are, in fact, actual trees.**

3.1 Sample Output

Here is an example of what the output of your program might look like. Remember, you are expected to be creative in designing your skill tree, and your submission should not attempt to duplicate what you see here aside from the general formatting (the formatting can be the same, but the data should be different!). Note that the formatting of output of the skill tree contents is done by the `toStringByLevel()` method of `BasicMAryTree280`.

```
My Skill Tree:

1: Slash, Cost: 1
     2: Mighty Blow, Cost: 2
     2: Shield Bash, Cost: 1
          3: Shield Charge, Cost: 2
          3: Parry, Cost: 2
               4: Shield Wall, Cost: 4
               4: -
               4: -
               4: -
          3: -
          3: -
     2: Cleave, Cost: 2
          3: Whirlwind, Cost: 3
               4: Berzerk, Cost: 5
               4: -
               4: -
               4: -
          3: -
          3: -
          3: -
     2: Mobility, Cost: 1

Dependencies for Shield Wall:
Slash, Cost: 1, Shield Bash, Cost: 1, Parry, Cost: 2, Shield Wall, Cost: 4,
Dependencies for Mobility:
Slash, Cost: 1, Mobility, Cost: 1,
Dependencies for Slash:
Slash, Cost: 1,
Dependencies for FakeSkill:
FakeSkill not found.
To get Whirlwind you must invest 6 points.
To get Mighty Blow you must invest 3 points.
To get Slash you must invest 1 points.
FakeSkill not found.
```

4 Files Provided

lib280-asn3: A copy of lib280 which includes the `ArrayedBinaryTree280` class needed for Question 1, a partially complete `ArrayedBinaryTreeWithCursors280` for Question 1, and the `BasicMAryTree280` class which is needed for Question 2.

Skill.java : A complete implementation of the `Skill` class needed for Question 2.

SkillTree.java : A template for your implementation of the `SkillTree` class in Question 2.

5 What to Hand In

You must submit the following files:

ArrayedBinaryTreeWithCursors280.java - Your completed class for Question 1.

SkillTree.java - Your completed implementation of the skill tree and associated tests.

The console output from your `SkillTree::main()` test program to question 2.