Practical 4

COS212

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA
Denkleiers • Leading Minds • Dikgopolo tša Dihlalefi

Department of Computer Science

Deadline: 07/04/2023 at 23:59

Marks: 200

1 General instructions:

- This assignment should be completed individually; no group effort is allowed.
- Be ready to upload your assignment well before the deadline, as no extension will be granted.
- You may not import any of Java's built-in data structures. Doing so will result in a mark of zero. You may only make use of native arrays where applicable. If you require additional data structures, you must implement them yourself.
- If your code does not compile, you will be awarded a zero mark. Only the output of your program will be considered for marks, but your code may be inspected for the presence or absence of certain prescribed features.
- All submissions will be checked for plagiarism.
- Read the entire assignment before you start coding.
- You will be afforded five upload opportunities.
- Make sure your IDE did not create any packages or import any libraries which are not allowed.

2 Plagiarism

The Department of Computer Science considers plagiarism a serious offence. Disciplinary action will be taken against students who commit plagiarism. Plagiarism includes copying someone else's work without consent, copying a friend's work (even with permission) and copying material from the Internet. Copying will not be tolerated in this course. For a formal definition of plagiarism, the student is referred to http://www.library.up.ac.za/plagiarism/index.htm. **If you have any questions regarding this, please ask one of the lecturers to avoid misunderstanding.** Also, note that the OOP principle of code reuse does not mean you should copy and adapt code to suit your solution.

3 Outcomes

Upon completing this assignment, you will have implemented the following:

- a min heap represented as an array.
- a way to change the number of children per node for the heap.

4 Background

A min-heap is a binary tree where the value of each node is less than or equal to the values stored in each of its children. The tree is also perfectly balanced, and the leaves are all in the leftmost positions. This concept is then combined with a d-heap. A d-heap is a heap data structure, but instead of every node having 2 children, every node has d children. The heap is stored inside and can be thought of as the breadth-first traversal representation of the heap.

4.1 Example

Given this array of values, the heaps are shown for d = 2 in Figure 1 and d = 3 in Figure 2.

```
[1,2,3,4,5,6,7,8,9,10]
```

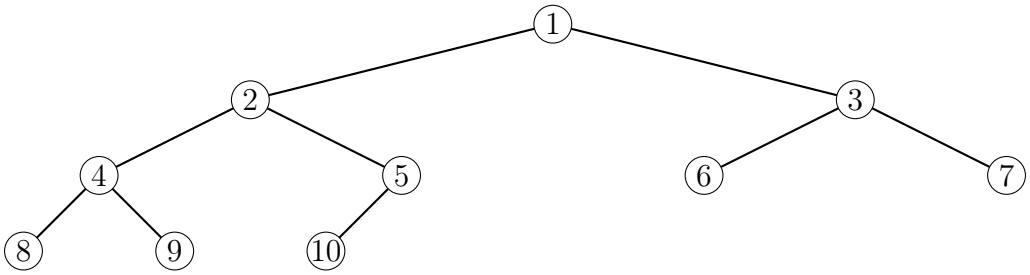

Figure 1: d=2

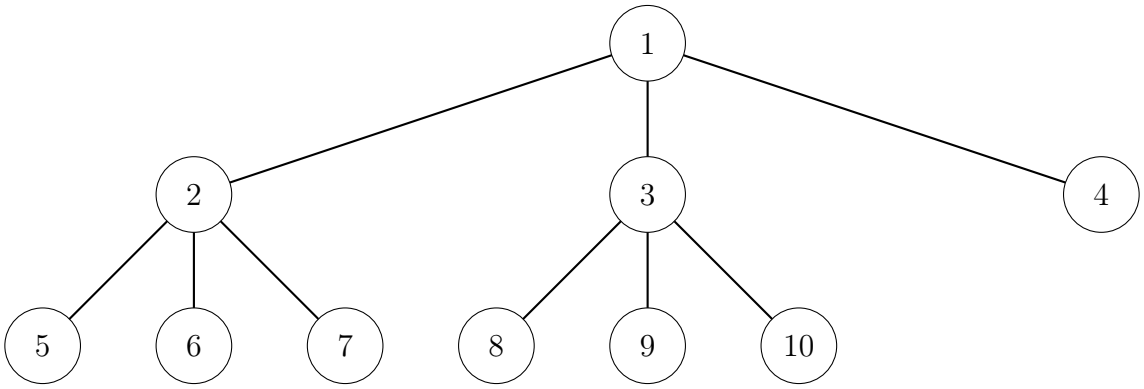

Figure 2: d=3

5 Tasks

5.1 Tips

- This practical uses a generic class T which extends comparable. It is really important that you use compareTo() and equals() when comparing elements and not ==,< and >.
- This practical uses a T[]; when coding this, you will realise you can't create arrays of generic classes (T[]). Instead, you must create a Comparable[] and then cast this to a T[].
- The UML given is very limited and only lists the functions that will be tested on Fitchfork. It will be very difficult to complete this Practical without helper functions. Thus, I really recommend that you make use of them.

5.2 minDHeap<T>

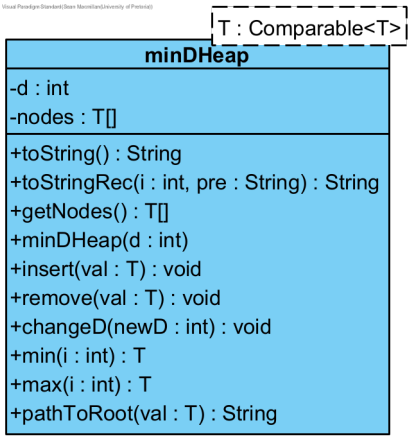

Figure 3: UML

- Members
  - d:int
    - This is the d value of the heap. This means that every node in the heap can have at most d children.
  - nodes:T[]
    - This is the array used to store the nodes in the heap.
    - If the heap is empty, this will be an array of size 0.
    - The length of this array should always match the number of nodes in the heap. Thus if nodes are added or removed, then the array should be resized.
- Functions
  - toString(), toStringRec()
    - These functions are given to you. Don't change them, as they will be used to mark your practical.
    - You are encouraged to use them when creating a Test Main as they print the heap in a way which is very readable.
  - getNodes():T[]
    - This function is given to you and only returns the array of values.
    - This is also used to mark your practical, so don't change it.
  - minDHeap(d:int)
    - This is the constructor for the heap.
    - You may assume the passed-in parameter will be strictly positive.

- insert(val:T):void
  - This function inserts an element into the heap.
  - Make sure that the heap properties are maintained after the insert and that the array is the correct size.
  - This is also referred to as "Enqueuing".
- remove(val:T):void
  - This function removes an element from the heap.
  - Make sure that the heap properties are maintained after the removal and that the array is the correct size.
  - This is also referred to as "Dequeuing", but this time any node can be removed, not just the root.
- changeD(newD:int):void
  - This function changes the number of children a node can have.
  - This function will use a similar approach to the Floyd algorithm for typifying arrays.
    - Change the d value to the new passed-in value.
    - Pretend that the array is already a heap with the correct d value.
    - Start from the last non-leaf node and swap down until it is in the correct position.
    - Continue this moving from the last non-leaf node to the root.
- min(i:int):T
  - This function returns the minimum value in a sub-heap.
  - The passed-in parameter is the index of the sub-heap's root.
  - For example, calling min(1) on the heap in Figure 2 returns 2.
  - If the tree is empty or the index is invalid, return null.
  - *Tip: This is a min-heap. Think about where the minimum value for a heap is located and use this to return the min value.*
- max(i:int):T
  - This function returns the maximum value in a sub-heap.
  - The passed-in parameter is the index of the sub-heap's root.
  - For example, calling max(1) on the heap in Figure 2 returns 7.
  - If the tree is empty or the index is invalid, return null.
  - *Tip: This is a min-heap. Think about the places where the max value will be located. A recursive function will be the easiest.*
- pathToRoot(val:T):String
  - This function prints the path in the heap from a node to the tree's root.
  - If the node is not found in the tree, return an empty string.
  - The value of each node is printed inside square brackets, which are then appended to each other. Don't add a new line to the end of the String.
  - Look at the given Main.java and out.txt for an example.

6 Submission instructions

Do not submit any custom function classes.
Your code should be able to be compiled and run with the following commands:
javac *.java
java Main

Once you are satisfied that everything is working, you must create an archive of all your Java code into one archive called uXXXXXXXX.{tar.gz/tar/zip} where X is your student number. Submit your code for marking under the appropriate link before the deadline. **Make sure your archive consists of only java files and no folders or subfolders.**

Please ensure that you thoroughly test your code before submitting it. Just because your code runs with local testing does not mean that your code works for every situation. **DO NOT USE FITCH FORK AS A DEBUGGER**

7 Submission checklist

- minDHeap.java

8 Allowed imports

- You are not allowed to import any libraries.