Assignment 2:

Min-Max d-Heap

COS 212

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

Department of Computer Science
Deadline: 16/05/2021 at 23:00

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 are NOT allowed to import any of Java's built-in data structures. Doing so will result in a mark of 0. You may only make use of native arrays where applicable. If you require additional data structures, you will have to implement them yourself.
- Only the output of your program will be considered for marks.
- Your code may be inspected to ensure that you've followed the instructions.
- If your code does not compile, you will be awarded a mark of 0.
- Read the entire assignment before you start coding.
- You will be afforded three upload opportunities.

Plagiarism:

The Department of Computer Science considers plagiarism as 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 consent) and copying material (such as text or program code) 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` (from the main page of the University of Pretoria site, follow the *Library* quick link, and then choose the *Plagiarism* option under the *Services* menu). If you have any form of question regarding this, please ask one of the lecturers, to avoid any misunderstanding. Also note that the OOP principle of code re-use does not mean that you should copy and adapt code to suit your solution.

After completing this assignment:

Upon successful completion of this assignment you will have implemented an efficient double-ended priority queue using a min-max d-heap.

Problem Description

A priority queue is an abstract data type similar to a regular queue, but where each element has a "priority" associated with it, in addition to the stored data. In a priority queue, an element with high priority is always served before an element with low priority. If two elements have the same priority, they are served according to their order in the queue.

A double-ended priority queue is a data structure similar to a priority queue, but where efficient removal of both the maximum and minimum (highest and lowest priority) elements is allowed, according to some ordering of the keys stored in the structure.

Priority queues can be implemented as heaps, since storing the highest priority element as the root of the heap ensures $O(1)$ access. For double-ended priority queues, we would need to ensure that both the highest and the lowest priority elements are immediately accessible. This can be accomplished by storing two heaps: a min-heap and a max-heap. Alternatively, memory efficiency can be improved by combining the two heaps into a single structure, knows as the **min-max heap**.

Min-Max Heap

A min-max heap is a complete binary tree data structure which combines the usefulness of both a min-heap and a max-heap, that is, it provides $O(1)$ retrieval and $O(\log n)$ removal of both the minimum and maximum elements in it. This makes the min-max heap a very useful data structure to implement a double-ended priority queue.

The min-max heap property is: each node at an *even* level in the tree is less than all of its descendants, while each node at an *odd* level in the tree is greater than all of its descendants. Thus, min-max heaps are made of alternating min (or even) and max (or odd) levels. The root element has the smallest priority key in the min-max heap. One of the two elements in the second level, which is a max (or odd) level, has the largest priority key in the min-max heap.

See an example min-max heap in Fig. 1. Smallest element, 6, is the root of the tree. Largest element, 87, is the child of the root. Even levels constitute a min-heap, while the odd levels constitute a max-heap.

Algorithms for basic Min-Max heap operations are explained below.

Insertion

1. Append the required key to the array representing the min-max heap. This may break the min-max heap property, thus the heap has to be adjusted.
2. If the key is inserted at max level:
   a) Compare the inserted key with its min-level parent:
      i. If the key is larger than the parent, then it is larger than all other keys present at min levels. Thus, min-property is satisfied. Now, ensure that max-property is satisfied by iteratively or recursively swapping the key with its grandparents if key $>$ grandparent.
      ii. If the key is smaller than the parent, swap it with the parent to restore max-property. Now, ensure that min-property is satisfied by iteratively or recursively swapping the key with its grandparents if key $<$ grandparent.

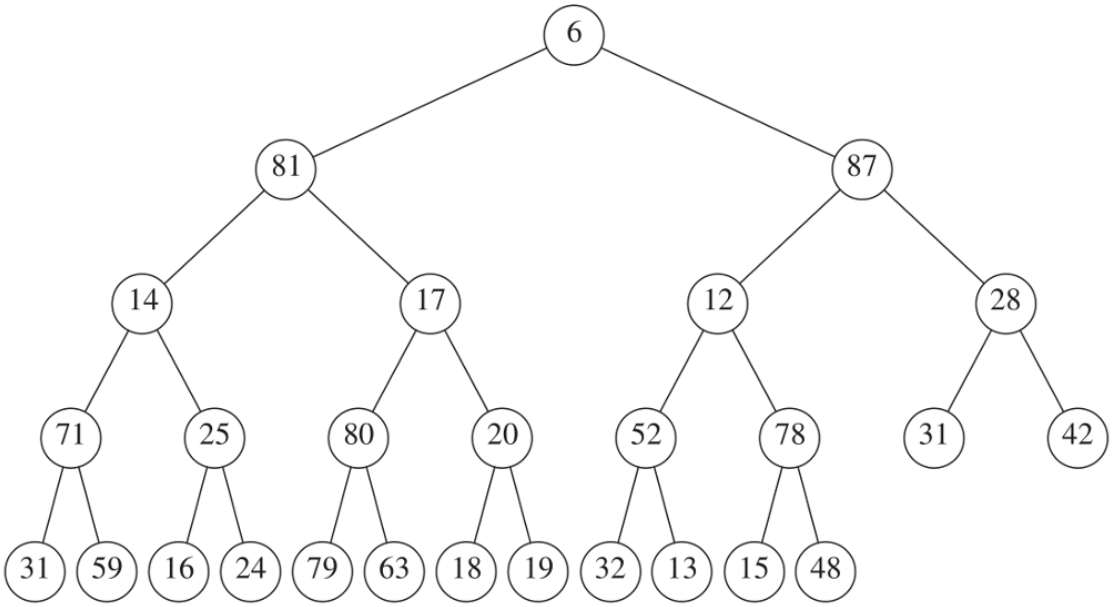

Figure 1: Min-Max Heap

3. If the key is inserted at min level:

   a) Compare the inserted key with its max-level parent:

      i. If the key is smaller than the parent, then it is smaller than all other keys present at max levels. Thus, max-property is satisfied. Now, ensure that min-property is satisfied by iteratively or recursively swapping the key with its grandparents if key $<$ grandparent.

      ii. If the key is larger than the parent, swap it with the parent to restore min-property. Now, ensure that max-property is satisfied by iteratively or recursively swapping the key with its grandparents if key $>$ grandparent.

Consider Fig. 1. The first available position at the last level will be used for insertion. Thus, whatever value is inserted, it will be inserted as a left child of 31. The level of the new node j will be 4, which is a min level. Consider the following two scenarios:

1. Value 1 is inserted. Compare 1 to 31: 1 is smaller, thus it is smaller than all values on the max levels. Thus, comparisons to min-level grandparents need to be made. Compare to 28: 1 is smaller, swap the nodes. Compare 1 to 6: 1 is smaller, swap the nodes. Thus, 1 becomes the new root of the tree.

2. Value 55 is inserted. Compare 55 to 31: 55 is larger, thus 31 and 55 are swapped. Comparisons to max-level grandparents need to be made. Compare 55 to 87: 55 is smaller, thus we can assume that it satisfies the max-property, and stop the algorithm.

Deletion

Since min-max heap represents a double-ended queue, we need to provide only two kinds of deletion operations: delete minimum element, and delete maximum element. Deletion from a min-max heap follows an algorithm similar to that of an ordinary heap: the value to be deleted is overwritten by the last element of the heap (last leaf of the last level). Once the replacement has been made, the new root value is trickled down the heap until both the min- and the max-properties are satisfied. The algorithm to remove the smallest element can be summarised as follows:

1. Overwrite the root node with the last node of the heap.

2. Trickle the new root down recursively:

a) Find the index i of the smallest child/grandchild of node j.

   i. If i is a child of j, and key of $i <$ key of j, swap nodes i and j.

   ii. Else, if i is the grandchild of j, and key of $i <$ key of j, swap nodes i and j, and compare j to its new parent. If key of $j >$ parent, swap j with parent.

b) Repeat until the min-max heap property is satisfied.

The procedure for removing the maximum node is very similar, except that the relational operators are reversed.

For examples of deletion, consider Fig. 1.

1. Suppose min value, 6, is removed. The root is overwritten by 48 (last leaf on the last level), and the last leaf is deleted/set to null. Now 48 is the new root, and has to be trickled down the tree. Find the smallest child/grandchild of 48: it is 12. 12 is a grandchild of 48, the two nodes are swapped: 12 is now the root, and 48 is the left child of 87. Is $48 > 87$? No: continue with the trickle down algorithm. Find the smallest child/grandchild of 48. It is 13. 13 is a grandchild, swap 48 with 13. Now, 13 is the left child of 87, and 48 is the right child of 52. Is $48 > 52$? No. Because we have reached the bottom of the tree, we can stop the algorithm.
2. Suppose max value, 87, is removed. 87 is overwritten by 48 (last leaf on the last level – this example does not stack on the previous one), and the last leaf is deleted/set to null. Now 48 is the right child of 6, and has to be trickled down the tree. Find the largest child/grandchild of 48: it is 78. 78 is a grandchild of 48, the two nodes are swapped: now, 78 is the left child of 6, and 48 is the right child of 12. Is $48 < 12$? No: continue with the trickle down algorithm. Find the largest child/grandchild of 48. It is 15. Is $15 > 48$? No, thus no swap is made, and we can stop the algorithm.

Construction

Using the trickle down algorithm described in the previous section, the Floyd's heapifying algorithm can be adapted to convert an arbitrary array into a valid min-max heap. The procedure must differentiate between min- and max-levels, and iteratively or recursively construct the min-max heap in bottom-up fashion.

d-Heap

A `d-heap` is a heap structure that allows for up to d children per node. One of the motivations behind implementing such a structure is that the more children each node can have, the more nodes can be contained on fewer levels. A binary heap is a 2-heap, $d = 2$. A ternary heap is a 3-heap, $d = 3$. An example of a min `d-heap`, $d = 3$, is shown in Fig. 2.

The `d-heap` consists of an array of n items, each of which has a priority associated with it. These items may be viewed as the nodes in a complete d-ary tree, listed in breadth first traversal order: the item at position 0 of the array forms the root of the tree, the items at positions 1 through d are its children, etc. Just like the binary heap, the `d-heap` is always interpreted as a perfectly balanced tree with no gaps between the nodes on each level. To derive the location of the parent/grandparent of the item at position i (for any $i > 0$), you will have to generalise the binary heap formulae given in the textbook. According to the heap property, in a min-heap, each item has a priority that is at least as large as its parent; in a max-heap, each item has a priority that is no larger than its parent.

Your task

For this assignment, you will implement a data structure that combines the `d-heap` with the min-max heap. In other words, you have to implement a fully-functional `min-max d-heap`.

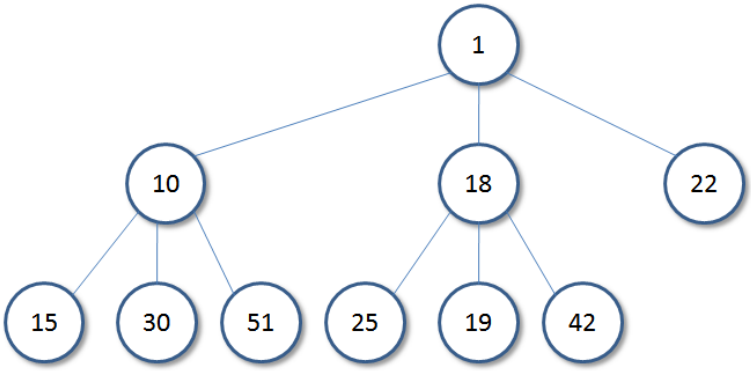

Figure 2: d-Heap, d = 3

This assignment is divided into a number of steps. You must implement all of the steps. It is strongly advised but not required that you implement this assignment in the order in which the steps are given.

Step 1: Node class

Download the archive `assignment2Code.zip`. You will have to complete the `Node` class which was given to you. Each `Node` object represents a node in the min-max `d-heap`, and thus should store data of arbitrary type, as well as a priority value (key) which determines the location of the node in the data structure.

You are also given the file `Test.java`. You must write your test code in this file. It will be overwritten for marking purposes.

Step 2: MinMaxDHeap class

The `MinMaxDHeap` class represents the min-max `d-heap`. You have to complete this class by implementing all of the methods. All comparisons/structuring should be based on the integer keys. Refer to the comments in the code for the description of the individual methods. You will have to implement the following functionality:

- Construction of a min-max d-heap object with an arbitrary d value.
- Insertion of data items based on the provided keys (priorities).
- Read-only access to min/max priority elements.
- Conversion of the min-max d-heap object to a printable `String` object.
- Removal of min/max priority elements, with the correct restructuring of the heap.
- Construction of the min-max d-heap from an arbitrary array of Node objects.
- Re-construction of the min-max heap for a new value of d. I.e., if the current value of d is 2, you should be able to, given a new value of d such as 3, restructure the current min-max d-heap accordingly.
- Clearing the heap, i.e. removal of all Node objects.

Write code to test your implementation in `Test.java`. For the sake of simplicity, your functions should not throw any exceptions.

Implementation Issues

Do not rename any of the classes or methods provided to you. You will also have to include a makefile for your implementation.

Submission instructions

Once you are satisfied that everything is working, you must tar or zip all of your Java code, including the makefiles, into one archive named uXXX.tar (alternatively, uXXX.zip), where XXX is your student number. Make sure your archive is valid before you submit it for marking, and that it does not contain any subfolders.

Go to the website `https://ff.cs.up.ac.za` and log in with your CS website credentials.

Select COS 212 amongst your modules and find the assignment link called Assignment 2 and submit the appropriate archive to the upload for automarking before the deadline.