Design Laboratory

Online facilities for supporting and evaluating teaching-learning activities.

Min-Heap Data Structure

By

Yerra Pranay Hasan

[11CS30042]

Guide: Prof. Chittaranjan Mandal

November 24, 2015

I. INTRODUCTION

Data structures and algorithms have operations which are specific to it and can be used to execute various tasks for which it was designed. Generation and evaluation of programming assignments involving these data structures and algorithms can be sufficiently automated to avoid manual verification, which becomes tedious as the class size grows. The aim is to integrate this feature of generation and evaluation in `WBCM`, wherein student can use it to practice or evaluate himself/herself.

A. Motivation

Automated generation and evaluation of assignments is desired. The following are among the few motives for such a scheme:

- It is advantageous for the TAs since their efforts of grading gets much lower.
- Students get rapid feedback about what they did wrong(by practicing) and they will get an idea of how many points theyll earn at the end when they actually attempt a quiz.
- Lenient and strict TAs are now not a problem since the evaluation is automated and hence consistent.
- Plagiarism chances have drastically reduced.

B. Challenges

We present an inexhaustive list of challenges below:

1. `Building an intuitive interface:` Cycles of feedback and implementation will be required to settle on an interface intuitive enough to use.
2. `Correction Scheme:` We are recording the moves performed by the student to answer a question. But these moves are only for the students enabling to check themselves

without resetting and repeating the whole process of answering that question. So, the recorded moves won't be used for evaluation. The evaluation is done by matching the final data-structure created by the student and the correct data-structure simulated by the machine.

3. `Initial Data Structure Generation Scheme:` The initially data-structure is randomly generated using a seed with a predefined range on the number of elements. A seed is set so that the same data-structure is displayed to all the users at that moment. This can be customized as per a test/practice session thereby generating different random sequences for different students.

4. `Fixed set vs large numbers of initial DS:` The can be customized easily after a final decision.

II. IMPLEMENTATION

The implementation of Binary Search Tree was built and based on the same interface the following data structure Heap has been built. Other data structure AVL tree also is being built.

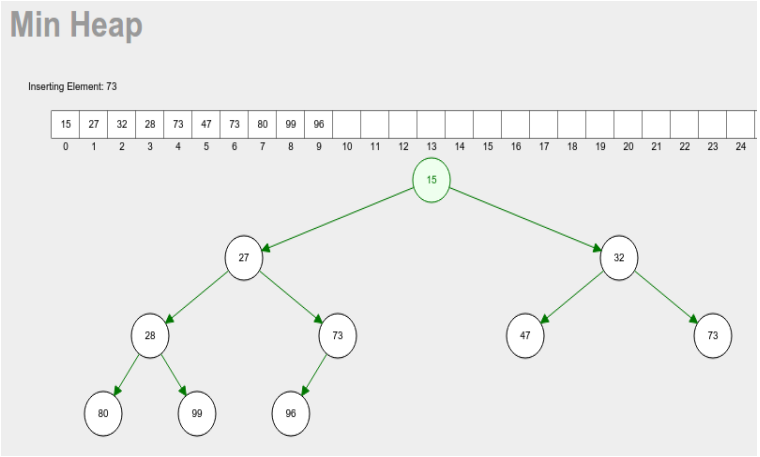


FIG. 1: Min Heap

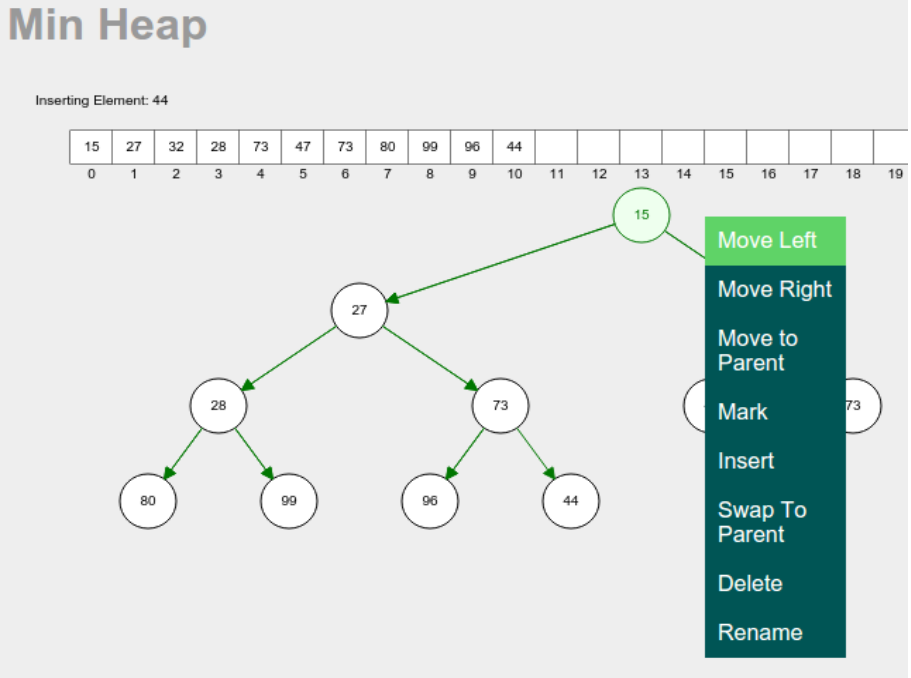


FIG. 2: Operation Menu

A. Overview and Operations

The student could left click at the current highlighted node and choose among the operations provided. The move will be recorded in the text box allotted so as to enable the student to self check his/her own moves.

B. Traversal Operations

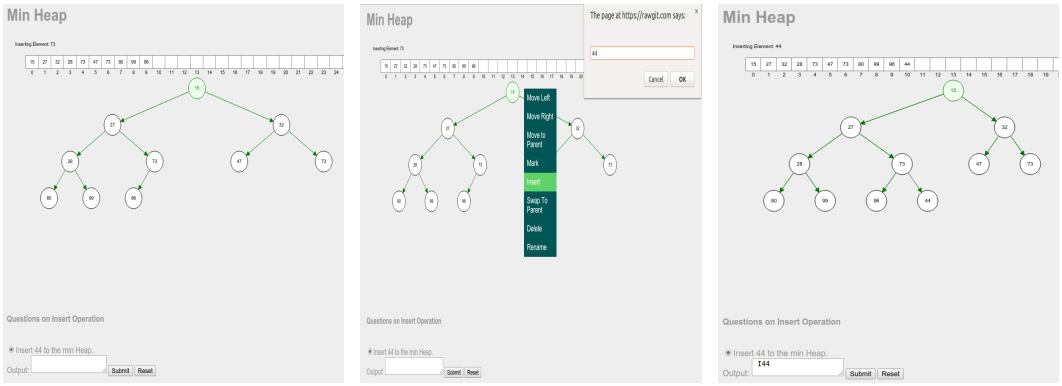
Miscellaneous question scheme deals with traversal operations.

1. Click on highlighted node and select **Move Left** to visit the left child (if any).
2. Click on highlighted node and select **Move Right** to visit the Right child (if any).
3. Click on highlighted node and select **Move to Parent** to visit the Parent (if any).
4. Click on highlighted node and select **Mark** to mark the present node(as visited for example).
5. Click on highlighted node and select **Swap to Parent** to swap the current highlighted node with the parent node(if any).
6. Click on highlighted node and select **Rename** to enter the new value of the highlighted node.

C. Insert Operation

1. Right click on any node and click on **Insert** and insert the value of the new node in the prompt. This inserts a new node at the end of the heap.
2. Then use **Move to Parent** to place the node such that the resultant structure satisfies heap structuring property.

Below figures demonstrate insertion of 44 into the Min Heap.



(a) (b) (c)

(d) (e) (f)

(g) (h) (i)

(j) (k)

FIG. 3: Demonstration of Inserting 44 into Min Heap

D. Delete Operation

1. Right click on any node and click on **Delete**. This deletes the last node at the end of the heap.

2. Then using **Rename**, rename the root node with the value of the deleted node. Now using **Swap to Parent** move the root element to the bottom so that heap ordering is satisfied.

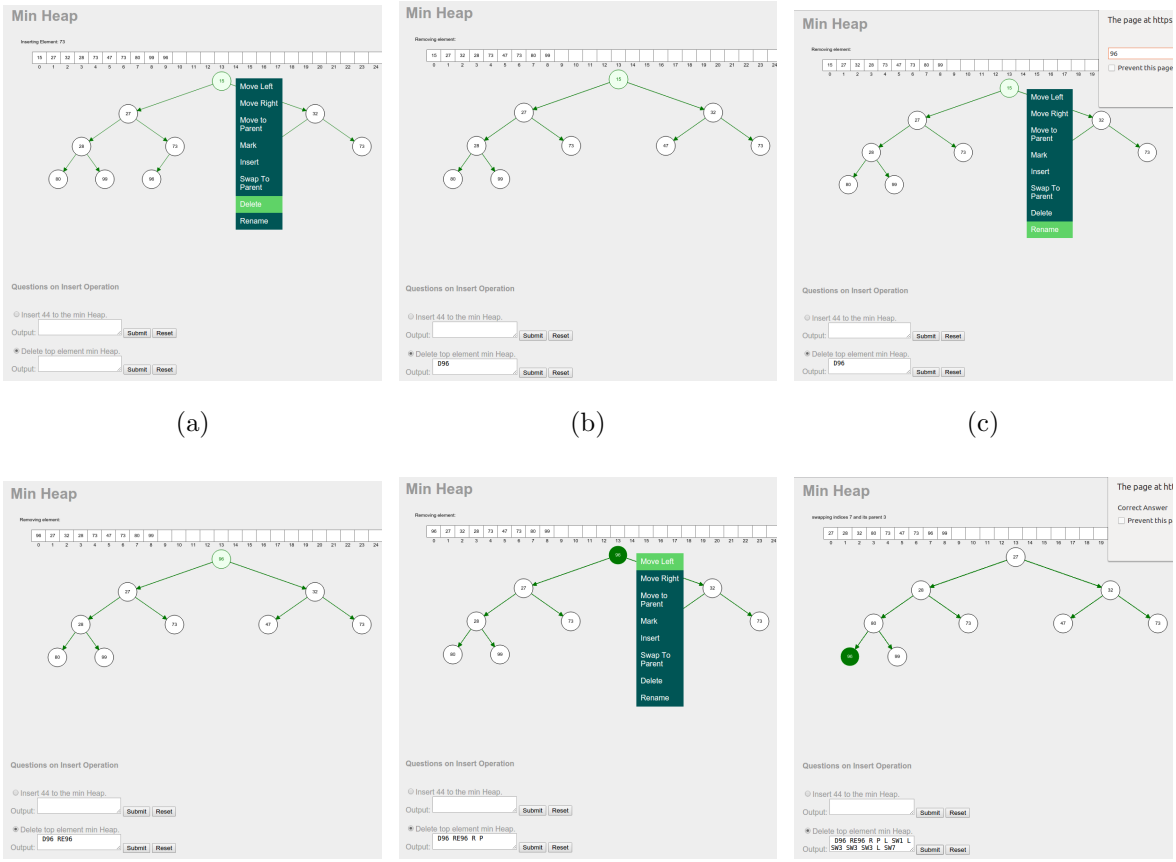


FIG. 4: Demonstration of Deleting top element in Min Heap (few steps after (e) skipped)

III. CODE STRUCTURE

This section will consist of the important code blocks which were changed in order to meet the requirements of the HEAP Data structure.

The code for Min Heap can be found here and the running version can be viewed in browser here

```
//*****************************************************************
//****************ADMIN CALLBACK BLOCK*************************
//*****************************************************************
Heap.prototype.addControls =  function() //Inserts elements in diplay and sets callbacks
{}

Heap.prototype.newArrayCallback = function(event)
{}

Heap.prototype.insertCallback = function(event)
{}

Heap.prototype.removeSmallestCallback = function(event)
{}

Heap.prototype.buildHeapCallback = function(event)
{}

Heap.prototype.heapifyCallback = function(event)
{}

Heap.prototype.deleteCallback = function(event)
{}

Heap.prototype.clearCallback = function(event) //TODO:  Make me undoable!!
{}

//*****************************************************************
//************END OF ADMIN BLOCK ***************************
//*****************************************************************
Heap.prototype.createArray = function() //initiating the array display
{}

Heap.prototype.newArrayBox = function(event) //for admin access
{}

Heap.prototype.newArray = function(event) //for initially generating the heap
{};

Heap.prototype.insertElementBox = function(insertedValue) //for admin access
{}

Heap.prototype.insertElement = function(insertedValue) //for initially generating the heap, ADMIN METHOD
{}

Heap.prototype.removeSmallest = function(dummy) //ADMIN METHOD
```

```
47 {}

49  Heap.prototype.buildHeap = function(event) //ADMIN METHOD
    {}
51
    Heap.prototype.heapify = function(index) //ADMIN METHOD
53  {}

55  Heap.prototype.swapWithParent = function() //ADMIN METHOD
    {}
57
    Heap.prototype.pushDown = function(index) //ADMIN METHOD
59  {}

61  Heap.prototype.swap = function(index1, index2) //ADMIN METHOD
    {}
63
    //*************************************************************
65  //**************Student Methods*********************************
    //*************************************************************
67  Heap.prototype.insert = function(insertedValue) //STUDENT INSERT
    {}
69
    Heap.prototype.rename = function(value) //STUDENT RENAME
71  {}

73  Heap.prototype.delete = function() //STUDENT DELETE
    {}
75
    //*************************************************************
77  //**************Evaluation Methods******************************
    //*************************************************************
79  AnswerInsert = function(valueInserted) //EVALUATION METHOD
    {}
81
    AnswerDelete = function() //EVALUATION METHOD
83  {}

85  AnswerHeapify = function(n, i) //EVALUATION METHOD
    {}
87
    AnswerMakeHeap = function(n) //EVALUATION METHOD
89  {}

91  //**************SUPPORT METHODS*********************************
    Heap.prototype.resizeTree = function() //SUPPORT METHOD
93  {}

95  Heap.prototype.setNewPositions = function(tree, xPosition, yPosition, side) //SUPPORT METHOD
    {}
97
    Heap.prototype.animateNewPositions = function(tree) //SUPPORT METHOD
99  {}

101 Heap.prototype.resizeWidths = function(tree) //SUPPORT METHOD
    {}
```

IV. AUTO EVALUATION

Auto Evaluation is not 100% possible as a lot of different kinds of questions are possible on heap. So auto-evaluation is possible only for a basic set of operations such as Insert, Delete, MakeHeap and Heapify Node. These set of methods have been implemented (as can be seen in Evaluation Methods in the Code Structure of Section III). So some basic questions are framed to demonstrate these. Once submit is clicked for each of these questions evaluation is done automatically.

The method of auto-evaluation is done by comparing the final data structures, one generated by the student and other simulated by machine in the background. The moves shown are just for reference of the student and not for evaluation. This is because in case of moves, most of the traversal operations have to be simulated which is an extra overhead in computation. The moves method also creates a lot of unnecessary bugs.

V. CONCLUSION

Marks Distribution:

1. **Interface:** 100/100
   Array+Tree Structure

2. **Traversal Operations:** 90/100
   Rename, Swap To Parent

3. **Heap Operations:** 100/100
   Insert, Delete, Heapify, MakeHeap

4. **Admin Mode:** 95/100
   All Heap operations visualisations

5. **Auto Evaluation:** 90/100
   Insert, Delete, Heapify, MakeHeap (Can be improved using NLP techniques and also by keeping a mapping of Questions)