**CS-222-01: Assignment 3 – Binary Max Heap**

Aiden Wells

CS-222-01

Professor Das

05/11/2024

1. **Introduction**

In this paper, I provide information on the implementation of a Binary Max Heap in Java.

1. **Binary Max Heap**

A Binary Max Heap is an Abstract Data Type (ADT) that is used to store a list of values maintaining the highest value at the top of the list. The heap is implemented using an array. The highest value will always be stored at the 0 index in the array.

1. **Binary Max Heap Properties**

The Binary Max Heap will have the following properties:

* **heaparray:** an array of integers.
* **size:** a variable used to store the size of the heap.
* **maxsize:** a variable used to create the array.

1. **Binary Max Heap Methods**

The Binary Max Heap will have the following methods:

* **MaxHeap:** this is the constructor method. It takes one argument which will be assigned to maxsize, which will be used to create heaparray. The size property is set to 0.
* **insertintoarray:** this method is used to insert a new value into the heap. A new value is inserted into the first empty node. If the new value is greater than the parent node, then swap the values. You have to repeat this process, until there are no swaps.
* **extractmax:** this method is used to extract the maximum value at index 0 from the array. When the max value is extracted, it is replaced with the last value in the array. Heapify is then called to restructure the array.
* **heapify:** this method restructures the array after extracting the max value. Heapify starts at the top of the heap at index 0. The method compares the parent value to the left child and right child values. If the left child or the right child is larger than the parent, then the largest child will be swapped with the parent. Heapify is called recursively until no swaps occur.
* **display\_vertical:** this method is based on the display method in the UBST class for printing a binary tree vertically.
* **display\_recursive:** this method is from the UBST class. The method is called from display\_vertical. It is a recursive method used to print nodes going down the tree.
* **visualize:** (extra credit) this is a method that will display the binary tree horizontally.

1. **Process**

I followed this process in developing the binary max heap:

1. First, I studied how a binary max heap is supposed to work. I read about heaps in ZYBooks, and I looked at code samples online.
2. Second, I developed a plan of the properties and methods that needed to be written for the binary max heap.
3. Then, I started writing code. I first created the MaxHeap class and a Driver class (for testing). I created the properties for MaxHeap and the Constructor method. I tested these. I then wrote the insertintoarray method for inserting a value. I tested more. I then wrote the extractmax and heapify methods. These were difficult. I looked at code samples on the web to figure out how to do this. I had to do a lot of testing to get this right. I then implemented the display and display\_recursive methods from the UBST class. This was pretty easy. Finally, I wrote the visualize method for printing the heap horizontally. This was difficult and took a lot of time.
4. I tested while writing the code, and I tested after I was done writing the code.
5. I discovered a few bugs, which I fixed.
6. **Code**

The following is the code for the MaxHeap class and the Driver class.

1. **MaxHeap class:**

// Aiden Wells

// axw1@stmarys-ca.edu

// 415-301-1273

// CS-222-01

// Professor Das

// Assignment 3: Binary Max Heap

// 05-11-2024

// MaxHeap class

public class MaxHeap {

// Three properties: heaparray, size, and maxsize.

private int[] heaparray;

private int size;

private int maxsize;

// Constructor method: Create new array of size arraysize.

public MaxHeap(int arraysize){

maxsize = arraysize;

size = 0;

heaparray = new int[maxsize];

}

// Method for inserting value into heap.

public void insertintoarray(int x){

heaparray[size] = x;

int temp = 0;

if(size > 0){

int indexval = size;

int parentindex = (indexval - 1)/2;

// If value is greater than parent value, swap with parent.

while(x > heaparray[parentindex]){

temp = heaparray[parentindex];

heaparray[parentindex] = heaparray[indexval];

heaparray[indexval] = temp;

indexval = parentindex;

parentindex = (indexval - 1)/2;

}

}

// Increment size

size++;

}

// extractmax method is used to extract the maximum value at index 0 from the

// array. When maxvalue is extracted, it is replaced with the last value in the

// array. Heapify is then called to restructure the array.

public int extractmax(){

int maxvalue = 0;

if(size>0){

maxvalue = heaparray[0];

// Move last value from array to index 0.

heaparray[0] = heaparray[size];

// Call heapify to restructure array.

heapify(0);

// Decrement size.

size--;

}

return maxvalue;

}

// heapify restructures the array after extracting the max value.

private void heapify(int x){

int temp = 0;

int largest = x;

int left\_index = (2\*x)+1;

int right\_index = (2\*x)+2;

// Check left child to see if it is larger than parent.

if(left\_index<size){

if(heaparray[left\_index]>heaparray[largest]){

largest = left\_index;

}

}

// Check right child to see if it is larger than parent and left child.

if(right\_index<size){

if(heaparray[right\_index]>heaparray[largest]){

largest = right\_index;

}

}

// If largest is left child or right child, swap with parent and call

// heapify recursively.

if(largest!=x){

temp = heaparray[largest];

heaparray[largest] = heaparray[x];

heaparray[x] = temp;

heapify(largest);

}

}

// display\_vertical is a method based on the display method in the UBST class

// for printing a binary tree vertically.

public void display\_vertical(){

display\_recursive(0, 0);

}

// Display recursive helper

private void display\_recursive(int root\_pos, int level){

if(root\_pos < size){

display\_recursive((root\_pos\*2)+2, level+1);

for(int i=0; i < level; i++){

System.out.print(" ");

}

System.out.println(heaparray[root\_pos]);

display\_recursive((root\_pos\*2)+1, level+1);

}

}

// visualize is a method that will display the binary tree horizontally.

public void visualize() {

// maxdepth is the lowest level of the tree.

int maxdepth = (int) (Math.log(size) / Math.log(2));

int y = 0;

for(int i = 0; i <= maxdepth; i++){

// Use a for loop for printing spaces.

for(int r = 0; r<((int) (Math.pow(2,(maxdepth-i)))-1); r++){

System.out.print(" ");

}

for(int x = ((int) Math.pow(2,i) - 1); x < ((int) Math.pow(2,i+1)-1); x++ ) {

if(y<size){

// This code will display integers up to 99999 centered in a 5

// character space.

if(Math.abs(heaparray[y])<10){

System.out.print(" " + heaparray[y] + " ");

}else{

if(Math.abs(heaparray[y])<100){

System.out.print(" " + heaparray[y] + " ");

}else{

if(Math.abs(heaparray[y])<1000){

System.out.print(" " + heaparray[y]);

}else{

if(Math.abs(heaparray[y])<10000){

System.out.print(" " + heaparray[y]);

}else{

System.out.print(heaparray[y]);

}

}

}

}

// Print spaces at the end of the row.

for(int r = 0; r<((int) (Math.pow(2,((maxdepth+1)-i)))-1); r++){

System.out.print(" ");

}

}

y++;

}

System.out.println("");

}

}

}

1. **Driver class:**

// Aiden Wells

// axw1@stmarys-ca.edu

// 415-301-1273

// CS-222-01

// Professor Das

// Assignment 3: Binary Max Heap

// 05-11-2024

public class Driver {

public static void main(String[] arg){

MaxHeap heap1 = new MaxHeap(100);

heap1.insertintoarray(11);

heap1.insertintoarray(22);

heap1.insertintoarray(33);

heap1.insertintoarray(44);

heap1.insertintoarray(55);

heap1.insertintoarray(66);

heap1.insertintoarray(77);

heap1.insertintoarray(88);

heap1.insertintoarray(89);

heap1.insertintoarray(90);

heap1.insertintoarray(91);

heap1.insertintoarray(92);

heap1.insertintoarray(93);

heap1.insertintoarray(94);

heap1.insertintoarray(95);

System.out.println("Max Value: " + heap1.extractmax());

System.out.println("Max Value: " + heap1.extractmax());

System.out.println("Max Value: " + heap1.extractmax());

System.out.println("Remaining Heap:");

heap1.display\_vertical();

heap1.visualize();

}

}