Hat Assignment

Ludachris OKN 527604

Shamil Karimli 523001

**introduction**

A hat is a generic data structure that can be used to retrieve random elements. For instance, it can be used to draw names. Hat has a method draw that is used to randomly delete items from the Hat.

First of all, a class Hat was implemented with it’s various methods. these methods include: add(), isEmpty(), size() and draw(). A test client was also implemented to test these methods.

**Objective**

The aim of this assignment is to create a generic java class with it’s necessary methods. A test client to check the implementations of these methods and most of all to determine the average or worse case complexity.

**Problem description**

The worst-case complexity describes an upper bound on the worst-case time we would see when running an algorithm, average case complexity will present an upper bound on the average time we would see when running the program many times on many different inputs. [1]

Part of this assignment is to measure the complexity of the various methods for the Hat class.

**Main question**

How can the average / worst-case complexity of a class method be measured?

**Methodology**

To measure the average time complexity of a method, a quantitative research was conducted. An experiment was conducted on the methods to find out the time it takes to run each method. This experiment was done by calling the method in the test client. A line of code is inserted into the method which will check and print the time taken to run each method.

Using the double ratio experiment, the input is doubled on each run 5 consecutive times.

**Data collection and analysis**

**isEmpty() Method**

This method is used to check if an instance of a Hat is empty or has inputs in the array. Given that this method only returns true if the array is empty, this give a constant time complexity.

Result overview of time taken to run the isEmpty method with input N

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| input | 10000 | 20000 | 40000 | 80000 | 160000 |
| time (ms) | 0 | 0 | 0 | 1 | 2 |

**size() Method**

Initially, an instance of a Hat is populated by using the add() method to add inputs. The input becomes the independent variable. The time complexity of the size() was experimentally measured by checking the time it takes to return the size of the array, which is the number of input in the array.

Result overview of time taken to run the size method with input N

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| input | 10000 | 20000 | 40000 | 80000 | 160000 |
| time (ms) | 0 | 0 | 0 | 1 | 2 |

**add() Method**

This method is used to populate/add input to the array. This method takes a generic data and adds it to the array. Since the method only add an input to the array. Its theoretical time complexity is O(1). To measure the experimental time complexity, the double ratio experiment was used. The initial input is doubled each time for 5 consecutive times. The time taken to add the input to the array is noted below in a table.

Result overview of time taken to run the add method with input N

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| input | 10000 | 20000 | 40000 | 80000 | 160000 |
| time (ms) | 2 | 3 | 7 | 13 | 18 |

**draw() Method**

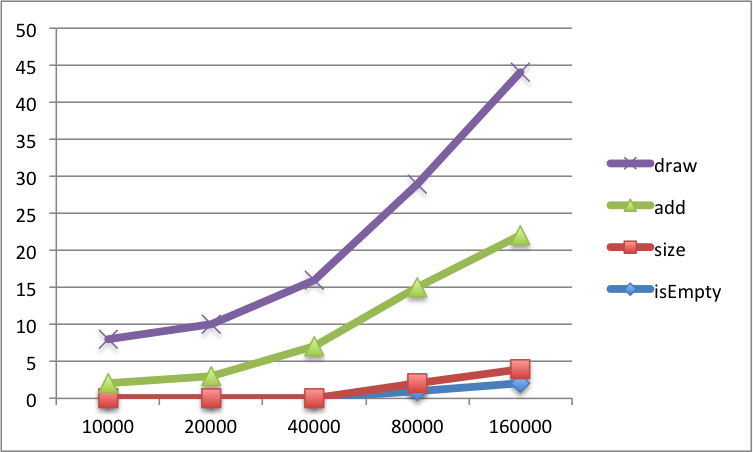
The draw method was implemented to draw an item from a hat. It goes through the array, select and delete a random item from the array and return it. The first line of this method takes a random item from the array and assign it to a variable. A for loop is then used to loop through the array, an if statement is then used to compare the random item chosen to all the items in the array. If there a match, it deletes this item from the array and return it.

To test the draw method. inside the client two lines of code are used to select two random people (people is an instance of a hat created in the test client and populated with strings using the add method) from the array and assigned a chore.

Result overview of time taken to run the draw method with input N

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| input (N) | 10000 | 20000 | 40000 | 80000 | 160000 |
| time (ms) | 6 | 7 | 9 | 14 | 22 |

**Graph of the experimental time complexity**

O(N) Time Complexity - linear time( finding the smallest or largest item in an unsorted array.

**Conclusion**

The findings above appear straightforward but become very interesting when the number of input increases. From the various results overview table, there was a clear increase in method run time when the input became bigger especially in both the add and draw method. In comparison to the worst-case complexity of an algorithm based on the theoretical point of view from the book [ Robert Sedgewick and Kevin Wayne . Algorithms Fourth Edition]. The worst-case complexity of the draw method has O(N). This happens because of the for loop used to check the input for the element selected to be deleted.

Finally, quantitative findings indicate that the time complexity of each method is proportional to the input in the draw and add method.

**Reference**

[1] [Analysis of Algorithms: Average Case Analysis](https://secweb.cs.odu.edu/~zeil/cs361/web/website/Lectures/averagecase/pages/index.html)

[1999-2006 Steven J. Zeil, Old Dominion University]

[2] Robert Sedgewick and Kevin Wayne . Algorithms Fourth Edition

[3]Cay S.Horstmann ,Gary Cornell, Core Java Volume I Fundamentals Ninth Edition

**Appendices**

**hat.java**

**hat\_test\_client.java**

**public class hat\_test\_client {**

**public static void main(String[] args) {**

**Hat<String> people = new Hat<String>();**

**people.add("Ama");**

**people.add("Kofi");**

**people.add("Yaa");**

**people.add("Adwoa");**

**System.out.println("Washing dishes: " + people.draw());**

**System.out.println("Vacuuming: " + people.draw());**

**}**

**}**

**public class Hat<T> {**

**private Object[] elements;**

**private int N;**

**private Object obj;**

**public Hat() {**

**elements = new Object[3];**

**N = 0;**

**}**

**public void add(T t) {**

**if (N < elements.length) {**

**elements[N] = t;**

**} else {**

**int newIncreasedCapacity = elements.length \* 2; // snippet from http://www.javamadesoeasy.com/2015/02/arraylist-custom-implementation.html**

**elements = Arrays.copyOf(elements, newIncreasedCapacity); //snippet from http://www.javamadesoeasy.com/2015/02/arraylist-custom-implementation.html**

**elements[N] = t;**

**}**

**++N;**

**}**

**public T draw() {**

**Object removedElement=elements[new Random().nextInt(elements.length)];**

**for(int i=0;i< N;i++){**

**elements[i]=elements[i+1];**

**}**

**N--;**

**return (T) removedElement;**

**}**

**public boolean isEmpty(){**

**return N == 0;**

**}**

**public int size(){**

**return N;**

**}**

**}**