2/27/2017

Assignment Hat V5

Algorithms and Datastructure 1 – Year 2 Mathematical Engineering – InHolland University of Applied Science

Project by:

572481 - Adu, Stephen and 570027- Andreicha, Semida

Table of Contents

[1. Introduction 1](#_Toc475800421)

[1.1 Problem Description 1](#_Toc475800422)

[2. Assignment Research Questions 2](#_Toc475800423)

[2.1 Main Research question 2](#_Toc475800424)

[2.2 Research sub-questions 2](#_Toc475800425)

[3. Research Methodology 2](#_Toc475800426)

[4. Results 3](#_Toc475800427)

[5. Conclusion 3](#_Toc475800428)

[6. Appendices 4](#_Toc475800429)

# Introduction

This assignment implements various elements from the Algorithms and Datastructures 1 University course. The main aspects that are involved are the implementation a generic data structure, analyzing the theoretical complexity of algorithms, and validating the complexity of algorithms in running time experiments.

## Problem Description

A Hat is a data structure that can be used to retrieve random elements. For instance, it can be used to draw names. It supports the following API[[1]](#footnote-1):

Public class Hat<Item>

Hat() *create a new empty hat*

boolean isEmpty() *is the hat empty?*

int size() *number of items in the hat*

void give(Item item) *add an item to the hat*

Item take() *delete a random item from the hat and return it*

It is needed to find the best implementation for an application with the class Hat[[2]](#footnote-2), and create an efficient implementations of the required methods. As well the determinacies of the average case time complexity[[3]](#footnote-3) of each of the methods. In addition experiments with doubling ratio should be used to verify the time complexity of the implementation.

# Assignment Research Questions

The main research question is assigned to the main focus of this assignment, which is the application Hat and its complexity implantations. The main research question is then further divided into four research sub-questions, which will help answer the main research question.

## 2.1 Main Research question

The main research question for this assignment is:

How to create the implementation for an application with the most efficient time complexity for each method implemented in class Hat and how to test this implementation?

## 2.2 Research sub-questions

The main research sub-questions for this assignment is:

1. What implementation is needed for the class Hat in order to create reasonably efficient implementations of the required methods?
2. What simple test client should be implemented to test all of the application’s implementations?
3. What is the average case time complexity of each of the methods?
4. By using the doubling ratio experiments [[4]](#footnote-4)what is the time complexity of the application’s implementation?

# Research Methodology

The methodology explored in this assignment reflects the class Hat and its complexity of implementations. By means of collecting data the research will then attempt to develop an application which has the best average time complexity during runtime. The process of development has several phases that are being implemented to get a final result. One of the phases is based on data collection. This is the point at which all the information necessary to proceed with any of the further steps, is going to be gathered. The second phase of the research then fully focuses on the development of the software programme. In the last phase of the development tests are used to determine if the chosen implementation has the best average time complexity. Subsequently, each one of these steps, will help answer the research sub-questions which in turn will provide an answer to the assignment’s research question.

The first step into development begins with **data collection**, which is based on the information presented during the university lectures of Algorithms and Datastructures 1. During the lectures the basics of implementing basic data structures were thought. As well some other resources used to gather information regarding the complexity levels were from an Algorithms textbook (Sedgewick & Wayne, 2011).

Then the process of development shifts towards the second step which is the **software development**, where the application will contain several main classes. One of them is the Hat class which will hold methods that can create a new empty hat, check if the hat empty, check the number of items in the hat, add an item to the hat, and delete a random item from the hat and return it. While the second class will be the client test class here objects will be added into the Hat and methods will be acting upon them. Such ones are adding and removing from the hat. A third class is the Timer, which will help determine the time complexity efficiency of each algorithm.

The tertiary step into development is the **tests** implementation, which are to help find the average time complexity for the developed software by the use of the doubling ratio experiment. Different implementations are to be done for the class Hat in order to find the best results in an average time complexity experiments. Those implementations vary among HatLinkedList, HatHashMapArrayList and TreeMap.

# Results

The results of the development of the assignment Hat, which results in a data structure that can be used to retrieve random elements, has been developed in sequential steps which involved: **data collection**, which gathered information for the **development of the software.** Then the **tests** implementation to find the average time complexity for the developed software.

During the **data collection** session a lot of information has been collected, which was to be further used in the development of the application. It was noticed the implementation of the application’s classes as well as the time complexity. The time complexity is scalable from to (Preez, B. D. ,n.d.). The application Hat has an initial inclination towards the complexity , which is a linear function. However it was deduced that the time complexity is better suited. This means that the running time grows at most proportional to .

To implement all the information that is collected, the **software development** is done by the Java [[5]](#footnote-6)programming language. The first steps into development began with the creation of the methods in the class Hat that have to be implemented for the application to be functional. They are: Hat(), *create a new empty hat*; isEmpty(), checks if the *hat empty*; size(), *number of items in the hat*; give(Item item), *adds an item to the* hat; take(), *deletes a random item from the hat and returns it*.

The results of the **tests** which were ran on the software developed showed the complexity of not only the application but the methods used in the application as well. The tests were done using the doubling ratio experiment for each implementation. The results are to be found under Table 1 in the appendices Section. It was concluded that the best algorithm implementation was done by HatLinkedList, having a Give() similar to a linear time complexity with an order of growth of . While having a Take() with a time complexity and order of growth of which is similar to a constant.

# Conclusion

The Main Research Question for this assignment has the focus on the development of the application Hat (Section 2.1). Then, from the Main Research Question, research sub-questions were derive to help provide the path to the answer of the main question for this research paper (Section 2.2). The conclusion for this research assignment will be written in the following structure: firstly the Research Sub-Questions will be answered and then the Main Research Question will be answered.

The **first sub-question** is referring to the gathering of information of what the supplication needs to have in the class Hat. As can be seen in Section 4 several methods had to be implemented such as: Hat(), *create a new empty hat*; isEmpty(), checks if the *hat empty*; size(), *number of items in the hat*; give(Item item), *adds an item to the* hat; take(), *deletes a random item from the hat and returns it*. For each one of these …

The answer to the first sub-question leads to an answer to the **second sub-question**. The simple test client has several …. A description of the test client that has been created to test all the applications implementations can be found in Section 4.

The way in which the **third sub-question** is answered is by running tests on the software and seeing the outcome graph. Section 4 has a detailed description of the tests ran to find the complexity of each method. …

In the **fourth sub-question**, the doubling ratio experiments were used to determine time complexity of the application’s implementation. …

In order to answer the **main research question**, it was needed to understand what is necessary for …

# Bibliography

Sedgewick, R., & Wayne, K., (2011). Algorithms (Fourth ed.) Princeton University.

Preez, B. D. (n.d.). The simple Big-O Notation Post. Retrieved February 26, 2017, from <https://www.javacodegeeks.com/2011/04/simple-big-o-notation-post.html>

# Appendices

1. Table – Implementations with the result of the doubling ratio experiment

|  |  |  |
| --- | --- | --- |
| Name | Take() implementation | Give() implementation |
| LinkedList |  |  |
| HashMap |  |  |
| Tree-Map |  |  |

1. API – An *Application Programming Interface* is a set of routines, protocols and tools for building software. [↑](#footnote-ref-1)
2. Class Hat - A *class* is the blueprint from which individual objects are created. Class Hat is a data structure that can be used to retrieve random elements (such as String [words], Integers [numbers], etc.). [↑](#footnote-ref-2)
3. Average Case Time Complexity – Of an algorithm is the amount of some computational resource (time) used by an algorithm, averaged over all possible inputs. [↑](#footnote-ref-3)
4. Doubling Ratio Experiments - An experiment that allows a user to analyze time taken for an algorithm to run, predicting how long it takes for the algorithm to do the task given the amount of data it has to handle. [↑](#footnote-ref-4)
5. Java – A general-purpose computer programming language designed to produce programs that will run on any computer system. [↑](#footnote-ref-6)