3/26/2017

Assignment Hat

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
1.1 Problem Description 1
2. Assignment Research Questions 1
3. Research Methodology 2
4. Results 2
5. Conclusion 3
6. Bibliography 3
7. Appendices 4

1. Introduction

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

1.1 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]:

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], and create an efficient implementations of the required methods. As well the determinacies of the average case time complexity[3] of each of the methods. In addition experiments with doubling ratio should be used to verify the time complexity of the implementation.

2. Assignment Research Questions

The *main research question* for this assignment is:

What application implementation has the most efficient time complexity for each method implemented in class Hat and how to test this implementation?

The *main research sub-questions* for this assignment is:

1. What is needed for the class Hat in order to create an 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]what is the time complexity of the application's implementation?

[1] API – An *Application Programming Interface* is a set of routines, protocols and tools for building software.
[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.).
[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.
[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.

3. 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 is 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.

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.

4. 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 $O(1)$[5] to $O(2^n)$ (Preez, B. D. ,n.d.), where *N* is the number of items in the Hat. The application Hat has an initial inclination towards the complexity $O(N)$, which is a linear function. However it was deduced that the time complexity $O(logN)$ is better suited, meaning that the running time grows at most proportional to $logN$. However, it was also found that the implementation for a $logN$ is perplexing. Thus it was settled to develop an application with an implementation of both HashMap and ArrayList. Having a complexity of $O(N)$ on the Take() and for the other methods, such as Give(), isEmpty() and Size(), as constant $O(1)$.

To implement all the information that is collected, the **software development** is done by the Java [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, see implementation under Section 5 *Implementations*. They are: isEmpty(), checks if the *hat is empty* and has a complexity O(1); size(), *number of items in the hat* having a complexity of O(1). The give(Item item) method checks whether an argument already exists, if true it returns, else if false adds argument to the collection for storing array objects; and in HashMap storing as key ArrayList and size as value. The take(), *deletes a random item from the hat and returns it*. While a second class is 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 helps determine the time complexity efficiency of each algorithm. As well as the interface Hat () which creates *a new empty hat* is used.

[5] O() – Order of growth.

[6] Java – A general-purpose computer programming language designed to produce programs that will run on any computer system.

The results of the **tests,** which were ran on the application's 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 in the Appendices section. It was concluded that the implementation of the HashMap and The LinkedList as an algorithm has given the method give a linear time complexity with an order of growth of $O(1)$. While having a Take() with a time complexity and order of growth of $O(N)$which is similar to a constant. It was found that the methods isEmpty() and Size() regardless of the size of the input will always have an order of growth of $O(1)$.

5. 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 the interface Hat(), *create a new empty hat,* and several methods had to be implemented such as: 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*. Each one of these methods take part into developing the algorithm with the average time complexity.

The answer to the first sub-question leads to an answer to the **second sub-question**. The simple test client has several objects which can be added into the Hat, and the methods from the various implementations of the class Hat will be acting upon them (Section 3). Those implementations will give different results (Section 4) of the actions of adding and removing objects from the hat. 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 graphs. It is deducted that for the chosen implemented algorithm, a combination of HashMap and LinkedList, the average case complexity for the methods isEmpty() and Size() is considered to be $O(1)$. While the method Give() has an average time complexity of $O(N)$, and Take() has a time complexity of $O(N)$. 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. It was found that the most efficient algorithm with the most reasonable average time complexity was the a combination of HashMap and LinkedList, which gives a complexity of $O(N)$. Details can be found under Section 4.

In order to answer the **main research question**, it was needed to understand what is necessary for an application to implement, with the average time complexity, for the class Hat. It was deduced after data collection that there are a variety of possibilities which are connected to the time complexity, however the best choice to use when developing an application is a time complexity with a lower runtime but with a greater input size (Section 3). Then the development process of the application had in consideration one main implementation, being an algorithm combination of HashMap and ArrayList. The implementation had methods such as Take() or Give(), which helped process the algorithms. Then while running the doubling ratio experiment it was noticed that, the algorithm combination, has indeed had the best results, $O(N)$, in regards to the time complexity(Section 4).

6. 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

7. Appendices

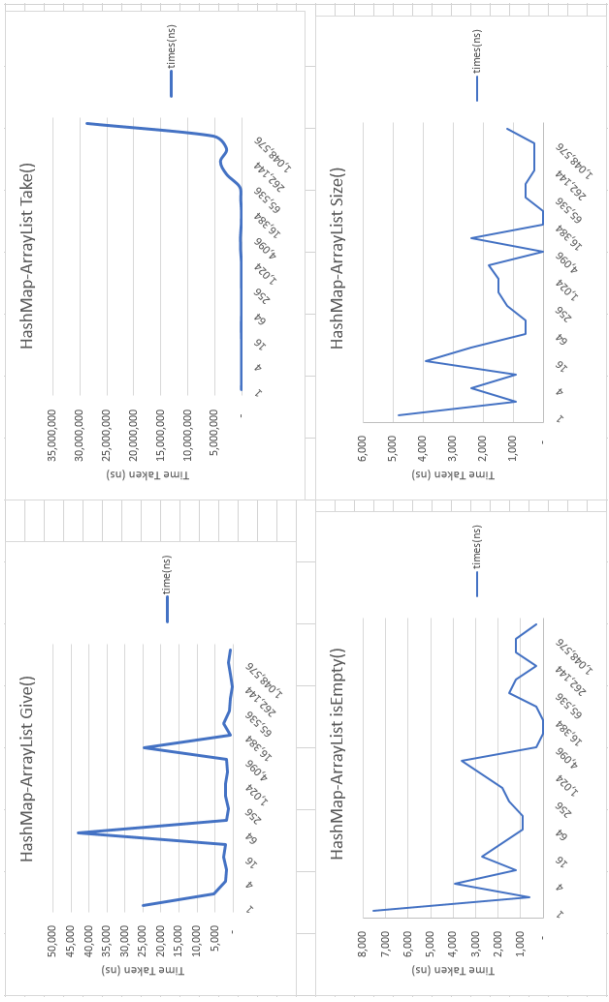
7.1 Implementations - HashMap + ArrayList Implementation Source-Code

```
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Random;

/**
 * Created by Stephen A. and Semida A.
 */
public class HatHashMapArrayListImpl<T> implements Hat<T> {

    private ArrayList<T> arrayObjects;
    private HashMap<T,Integer> integerHashMap;


    public HatHashMapArrayListImpl(){
        arrayObjects = new ArrayList<>();
        integerHashMap = new HashMap<>();
    }

    @Override
    public boolean isEmpty() {
        return arrayObjects.isEmpty();
    }

    @Override
    public int size() {
        return arrayObjects.size();
    }

    @Override
    public void give(T item) {
        if (integerHashMap.get(item) != null)
        return;                                     // Returns

        int s = arrayObjects.size();                // Object index
        arrayObjects.add(item);                     // Add Object to arrayObjects
        integerHashMap.put(item,s);                 // Add Object and Index

    }

    @Override
    public T take() {
        int index = getRandomIndex();               // Get random index
        T targetObject = arrayObjects.get(index);   // Gets random element from
arrayObjects

        arrayObjects.remove(targetObject);          // Remove object from
arrayObjects
        integerHashMap.remove(targetObject);        // Remove index from hashMap

        return targetObject;                        // Returns the Object to be
removed.

    }

    private int getRandomIndex(){
        Random r = new Random();                    // Create Random Object.
        return r.nextInt(arrayObjects.size());      // Returns random index
    }

    @Override
    public String toString() {
        return arrayObjects.toString();             // Returns the collection of
objects.
    }
}
```

7.1 Implementations - Hat Interface Source-code

```java
/**
 * Created by Stephen A. and Semida A.
 */
public interface Hat<T> {

    /**
     * @return Returns <b>true</b> if empty, <b>false</b> if more than 0 elements
     */
    boolean isEmpty();

    /**
     * @return Returns size the size
     */
    int size();

    /**
     * @param item The item to place into the collection
     */
    void give(T item);


    /**
     * @return Returns a removes and returns a random element from the collection
     */
    T take();
}
```

7.2 Table – Implementation results of the doubling ratio experiment of all the Data used during testing



HashMap-ArrayList Implementation

N	Give() Method time(ns)	Growth Ratio	Take() Method times(ns)	Growth Ratio	isEmpty() Method times(ns)	Growth Ratio	size() Method times(ns)	Growth Ratio
1	24,997		41,863	-	7,529	-	4,818	-
2	5,422	0.22	10,240	0.24	602	0.08	903	0.19
4	2,108	0.39	24,998	2.44	3,915	6.50	2,410	2.67
8	1,807	0.86	14,456	0.58	1,205	0.31	904	0.38
16	2,711	1.50	28,311	1.96	2,711	2.25	3,915	4.33
32	2,109	0.78	72,583	2.56	1,807	0.67	2,409	0.62
64	42,863	20.32	13,854	0.19	904	0.50	602	0.25
128	1,807	0.04	15,661	1.13	904	1.00	602	1.00
256	1,204	0.67	30,720	1.96	1,506	1.67	1,205	2.00
512	2,109	1.75	33,732	1.10	1,807	1.20	1,506	1.25
1,024	2,108	1.00	30,720	0.91	2,711	1.50	1,506	1.00
2,048	1,506	0.71	184,923	6.02	3,614	1.33	1,807	1.20
4,096	1,807	1.20	215,642	1.17	301	0.08	-	-
8,192	24,685	13.66	76,800	0.36	-	-	2,409	#DIV/0!
16,384	602	0.02	29,214	0.38	-	#DIV/0!	-	-
32,768	2,710	4.50	159,624	-	302	#DIV/0!	-	#DIV/0!
65,536	903	0.33	362,918	2.27	1,506	4.99	602	#DIV/0!
131,072	602	0.67	2,815,700	7.76	1,205	0.80	602	1.00
262,144	301	0.50	3,930,957	-	302	0.25	301	0.50
524,288	602	2.00	2,801,243	0.71	1,204	3.99	301	1.00
1,048,576	1,205	2.00	5,257,640	-	1,204	1.00	301	1.00
2,097,152	602	0.50	28,753,331	5.47	301	0.25	1,205	4.00
		Linear	2.6		1.8	Constant		#DIV/0!