Experimental Method - Planning

December 3, 2017

WONG, SAI MAN
TIGERSTRÖM, GABRIEL

Contents

Chapter 1

Experimental Method - Planning

Throughout the paper, we describe our plan and process to conduct experiments on four queue-implementations. Our task comes from [1], and Appendix A shows an overview of the requirements we must complete.

1.1 Purpose

A data structure represent a model to store data in a methodical manner [2]. Software engineers use queues in software developer to manage arbitrary entities in a first-in-first-out (FIFO) order. For example, developers use a queue to develop a memory buffer, or scheduling in an operating system.

Higher level programming languages often provide a standardized library with finished implementations, such as a queue. Software developers use these implementations frequently, and sometimes take the underlying architecture for granted. Thus, an inexperienced developer possess a limited knowledge of the theory and technical aspects of the used implementations.

Our purpose was to raise awareness of the features in standardized libraries, which developers and engineers often take for granted. Thus, we applied the scientific method to conduct experiments and evaluate four versions of a priority queue implementation. Consequently, we wanted to produce a credible and scientific work, so that other students and engineers can learn from and further develop.

1.2 Prestudy

Developers use the data structure, or model, linked list to store data linearly. The simple linked list consists of entities. An entity stores data and includes a connection to the next entity in the list. Computer scientist and developers refer the entity as a node or an element, and the connection as a pointer. That is, an element stores data and consists of a pointer to the next element in the list. Thus, developers

frequently use linked list implementations to store and manage data linearly, for example, in a queue implementation.

Our task was to evaluate four variations of a list-based data structure to represent a queue [1]. Each element compromises of a number to represent its priority in the queue. For clarification, an element with a low number represents higher priority, because the element lies closer to the present time and therefore ready for execution. We must implement the queues in the low-level programming language `C`, and the following list presents the implementation specifications:

1. **Singly linked list** – Insertion of new elements takes place in the head.
2. **Doubly linked list** – Insertion of new elements takes place in the rear.
3. **Doubly linked list** – Insertion of new elements takes place based on the mean of the first and last element's priority.
   - In the head – The new element's priority is higher than the mean, that is,
     numerical value $<$ mean value.
   - In the rear – The new element's priority is lower than the mean, that is, numerical value $\geq$ mean value.
4. **Scheduling queue** – The queue consists of an FIFO-queue for each priority in the $[0, 40]$ interval.

Computer scientists evaluate algorithms in terms of space and time, such as execution time and memory usage. That is, we gather a large quantity of these metric to determine best, average and worst cases. Thus, we asked ourselves: "Do these queues achieve similar performance, or does the performance vary depending on the circumstance?"

A queue's purpose is to enqueue and dequeue elements with an FIFO policy. Because of the FIFO policy, we can assume with a correct implementation and theory that the dequeue-operation should take $\mathcal{O}(1)$ time. However, the enqueue-operation is more sensitive to the element's priority, because it must insert each element in the correct spot in the queue. Thus, the implementations can perform differently due to the distribution of the priorities.

1.3 Indentify Relevant Methods

As mentioned in the last section, the most relevant metrics to examine algorithms with are time and space, such as execution time and memory usage. These metrics are all quantifiable. In contrast to, for example, interviews and surveys, which are

common methods in qualitative studies [3, 4]. That is, qualitative studies reaches conclusions from the quality of the data rather quantity. Thus, this study uses an experimental quantitative method to try to get a deeper insight about queues. However, we use both inductive and deductive method to reason about the algorithms.

We reason with a deductive method to explore the theory within computer science, such as algorithm complexity theory. That is, we can reach theoretical conclusions from complexity theory about the algorithms. For example, it is possible to determine an algorithm's performance based solely on theoretical knowledge. On the contrary, the usage of deductive methods can also yield unexpected results when put into practice. Thus, we generate realistic data from our experiments, and use an inductive method to further analyze it. That is, to explore if we can notice any observable patters in the data with help of models.

1.4 Experimental Plan

We verify, validate and examine the implementation's performance in form of time and space. Also, we analyze and test the best, average and worst case for each implementation. Finally, we evaluate these based on both theoretical knowledge and practical observations from the experiments. This is a must, because if it is not performed rigorously and correctly, then the study loses credibility.

We measure execution time for the implementation's enqueue- and dequeue-operations to determine its efficiency for best, average and worst cases. From a practical aspect, we use different stochastic distributions for the priorities to test the implementations. The space metric is also important to take into consideration. For example, to examine the memory usage of the algorithms. And, to avoid an incorrect implementation of a queue, which can result in memory leaks. We use Valgrind to monitor allocation and deallocation of memory as it is a widely accepted tool for this purpose. The following list presents the overview of our experiment plan:

1. **Understand the problem** – Gather a general understanding of queues, linked list and algorithm complexity. And, analyze how these models relates to our problem, or task, in theory and practice. Finally, set up hypotheses and assumptions based on the analysis.

2. **Methods** – Evaluate different scientific methods to apply in this study. That is, for our purpose, an experimental quantitative study is the most feasible one Finally, we discuss the findings and reach conclusions with both deductive and inductive reasoning.

3. **Break down the problem** – Use scientific methodology to set up smaller and clear requirements for the experiments.

4. **Implementation** – Implement the four variations of a queue.

5. **Verification** – Test each implementation's behavior. For example, write tests to verify that the implementation can enqueue the elements in correct order.

6. **Validation** – Analyze and validate the generated results from the implementation with theory. That is, to avoid unexpected and inaccurate output. The execution time of a specific implementation for best case tests are higher than worst case is an example of inaccurate result.

7. **Interpretation and visualize** – It is useless in science to generate data without an interpretation of it. We can represent, for example, the execution time and memory usage in graphs. That is, we use inductive method to interpret the data and reach a conclusion. Finally, then compare the latter conclusion with theoretical findings from deductive reasoning.

8. **Communication** – Present the entire work in form of a report. It is important that the study is as transparent, honest and rigorous as possible. That is, to create a credible and reproducible work.

1.5 Experimental setup

A solid foundation of software development, algorithms, data structures and complexity theory are required prerequisites to perform this experiment. In this study, we demonstrate our knowledge about these as clearly as possible.

1.5.1 System

We conduct the experiments on a Microsoft Azure virtual machine. Basically, it is a high-performance virtual private server. We use this service because we have full control over the environment. The system consists of a 64 bit architecture virtual CPU with two cores (Intel(R) Xeon(R) CPU E5-2673 v4 @ 2.30GHz), 8 GB memory and Ubuntu 16.04.3 LTS (Xenial Xerus) operating system.

1.5.2 Error Sources

Error sources are also necessary to take into account as these can influence the results of this study. That is, if we identify error sources early in the planning phase, we can then mitigate unexpected behaviors and results. Every software and computational research depend heavily on the underlying system, both hardware and software. Assume that the implementation is correct, the results depend on memory and CPU to produce accurate output, but also the operating system. Thus, we run the experiments multiple times and calculate statistical features, such as mean value and standard deviation. Standard deviation is relevant, because we can then identify how the execution time vary around the mean. For example, a large standard deviation indicates, for example, that another process irregularly consume a large amount of resources.

Another error source is poor design and implementation of models. Thus, it important to grasp and follow scientific methodology rigorously, verify and validate the implementations. It gets significantly more difficult to identify error sources, if the latter mentioned points are not taken seriously. For example, a poor implementation design results in that the researcher waste valuable time to debug. And, especially in a relative low-level programming language, such as C.

1.5.3 Implementation

A queue implementation consists primarily of enqueue- and dequeue-operations. When the queue enqueues an element, the implementation needs to allocate memory for the it and put it in the correct spot in the queue based on priority. For the dequeue-operation, the queue need to return the correct element with highest priority and free it from the memory to avoid memory leak.

We represent elements with help of data structure that can store data. And, it must also store one or two pointers. That is, the type of queue determines the amount of pointers. For example, an element in a singly linked list has one pointer, which refers to the next element in the list. And, an element in a doubly linked list has an additional pointer and it connects with the previous element. We plan to create an executable for each of the queue-implementations. Thus, for each executable we set up test cases to 1) test exertion time for best, average, worst cases and other stochastic distributions and 2) test memory usage.

1.6 Goals Reached and Experiments Completed

The bigger goal is to get a deeper knowledge about the scientific method, and apply it to evaluate four queue-implementations. In this paper we have identified what needs to be done to complete this study. Thus, the only goals to be reached is to execute the experiments, analyze the results, discuss and conclude the entire work in a smaller scientific report. Appendix A also shows gives an overview over the requirement analysis of this study.

The experiments are completed if the results are both verified and validated. Because, there is no value in scientific work to analyze inaccurate and wrong data. We explained the verification and validation process in Section 1.4 and Section 1.5.

Appendix A

Requirement Analysis

In [5], the author describes that a requirement analysis is a summary of all the requirement the researcher can find. The author clearly states that this document is *not* about how one should implement something to achieve a specific requirement. On the contrary, it should only cover the meaning of the requirement. Thus, we need to set up requirements that describes how to pass the task about the scientific method, specifically experimental quantitative method, as shown in Table A.1.

Table A.1. Requirement analysis about the scientific method

Requirement number	Requirement type	Name or source	Clarified description of the requirement, followed by what should be fulfilled	Fulfilled or not fulfilled or partly fulfilled
1	**Must** mandatory task	Purpose [6]	Ask good questions and identify the purpose. That is, why is it necessary to conduct an experimental evaluation of algorithms, and what is the application of it. The specific task description is described in [1].	Fulfilled (Section 1.1)
2	**Must** mandatory task	Prestudy [6]	Demonstrate an understanding about the task and field of study. Also, identify which parameters to use, how these can depend on and vary from each other	Fulfilled Section 1.2
3	**Must** mandatory task	Methods [6]	Identify which scientific method(s) to use. For example, quantitative, qualitative, deductive and/or inductive method	Fulfilled Section 1.3
4	**Must** mandatory task	Experimental Plan [6]	Identify experiments to conduct. Describe why these are relevant. Examine metrics to measure and elaborate on why these are relevant to this study.	Fulfilled Section 1.4

5	**Must** mandatory task	Experimental Setup [6]	Describe in details the resources needed. Identify and examine the influence of error sources, both external and internal ones. Set up an implementation, verification and validation plan.	Fulfilled Section 1.5
5	**Must** mandatory task	Goals [6]	Demonstrate how to determine when a goal is reached and an experiment is completed.	Fulfilled Section 1.6
6	**Must** mandatory task	Communication Part I [6]	Communicate and summarize the discussions of requirement 1-5 (Appendix A) in a document and hand it in to the mentor/supervisor	Fulfilled
7	**Must** mandatory task	Experimental Execution [6, 1]	Follow the discussed requirements 1-5, or only 6, to conduct the experiments on a real system. And, regularly document the findings with honesty and transparency.	
8	**Must** mandatory task	Communication Part II [7]	Summarize the entire work and its requirements, that is 1-7, in a smaller version of a technical report. Use the IMRAD-structure (Introduction, Method, Results, and Discussion). It is a common communication structure in scientific reports.	

Bibliography

[1] R. Rönngren. Uppgift läsåret 15/16 | ingenjörskunskap och ingenjörsrollen ICT (II1304) | KTH. [Online]. Available: https://www.kth.se/social/course/II1304/page/uppgift-lasaret-1516/ [Accessed: 2015-12-08]

[2] P. Deshpande and O. Kakde, *C & Data Structures*, ser. Charles River Media Computer Engineering. Charles River Media, 2004.

[3] R. Rönngren. Om experimentell och vetenskaplig metodik.pdf. [Online]. Available: https://www.kth.se/social/files/562ce17af2765431e02db7c3/Om%20experimentell%20och%20vetenskaplig%20metodik.pdf [Accessed: 2017-11-28]

[4] A. Håkansson, "Portal of research methods and methodologies for research projects and degree projects," in *The 2013 World Congress in Computer Science, Computer Engineering, and Applied Computing WORLDCOMP 2013; Las Vegas, Nevada, USA, 22-25 July*. CSREA Press USA, 2013, pp. 67–73.

[5] R. Rönngren. Enkel kravanalys/kravsammanställning. [Online]. Available: https://www.kth.se/social/course/II1304/page/kravanalys/ [Accessed: 2015-12-08]

[6] R. Rönngren. Å3 experimentell metod – planering. [Online]. Available: https://kth.instructure.com/courses/2502/assignments/11804?module_item_id=33137 [Accessed: 2017-11-29]

[7] R. Rönngren. Å3 experimentell metod – rapport. [Online]. Available: https://kth.instructure.com/courses/2502/assignments/11805?module_item_id=33138 [Accessed: 2017-11-29]