Searching with data structures

Gerben Aalvanger
Studentnumber: 3987051
Utrecht University

Erik Visser
Studentnumber: 3470806
Utrecht University

William Kos
Studentnumber: 3933083
Utrecht University

Sam van der Wal
Studentnumber: 3962652
Utrecht University

18 maart 2015
Teacher: Peter de Waal

1 Introduction

Searching values and storing them in memory is a key concept in computer science. For optimal speed and complexity, data is stored in a lot of different data structures, for example a tree or a skiplist. Every data structure has its own advantages, some perform better on insertion, while other perform better on finding a specific value. In this document we will describe how we will compare data structures and actually compare them. We will start by proposing a research question and some sub-questions. Secondly we specify the problem we want to research and the scope of our project. After that we explain our experiments in terms of criteria, test data and scenarios. And last we will show in different ways our test results and compare them to eachother and end with a conclusion.

2 Research description

In this research the question "Which data structure has the shortest duration on given actions" has been put central. We have used 4 data structures to get the answer to this question. To get the best insight in this question with 4 data structures we had chosen to take data structures which differ a lot from each other, namely:

- Lists
- Balanced trees
- Hash tables
- Min-max heaps

The list and hash table are both straight forward data structures, but there are multiple implementations possible for balanced trees and min-max heaps. For the balanced tree we had chosen to use the AVL tree because it is similar to the red-black tree, but known to be faster on lookups.

The min-max heap implementation we have used in our experiments is the interval heap. Our initial idea was to use the original min-max heap implementation but after implementing this algorithm it turned out that it did not work according to what was described in the paper we found. Therefore the decision was made to use the interval heap since the complexity for the individual actions are the same as with the min-max heap. All the actions used in our research will be described in the next section.

Part of the research was performing statistics on the results gained from the experiments. The focus laid at the average and the standard deviation of the performed actions on special cases. Because the same tests were performed multiple times we have chosen for both the average and the standard deviation to gain a good insight in the overall results retrieved from the experiments.

3 Experiment description

We will seperate the experiment in several section:

3.1 Build

In the build we have "build" the different datastructures. For the build operation we have build from15 different unsorted datasets, this is divided in datasets of size 10.000, 100.00 and 1.000.000, each size occuring 5 times. For each of these 15 options we have also tested the sorted and reverse sorted set. Each dataset is tested 30 times. This means that we have conducted all tests with 1350 distinct datasets.

3.2 Search

We have searched in the pre-build datastructures of section 3.1. The amount of searches is half the amount of elements in the datastructure. All the values of the searches are determined random and are garanteed to be in the datastructure. Like in the build section we have performed all search-tests 30 times.

We wanted to know if finding the minimum or maximum through the getMin (see section 3.5) and getMax (see section 3.6) methods of the datastructures are faster then searching for the minimum or maximum value with this search method. This test is conducted 1000 times on the datasets described in section 3.1

3.3 Insert

We have constructed new datastructures from scratch by repeatedly inserting new elements. Like in section 3.1 we have tested the insert actions for 15 sets of elements (set sizes: 10.000, 100.000, 1.000.000) and each of these tests have been performed 30 times.

3.4 Delete

We have performed delete operations on the pre-build datastructures of section 3.1. The amount of deletes is half the amount of elements in the datastructure. All the values of the deletes are determined random and are garanteed to be in the datastructure. Like in the build section we have performed all delete-tests 30 times.

3.5 getMin

We performed getMin operations on the pre-build datastructures of section 3.1. The amount of getMin operations in a test is 1000 since it returns always the same value. GetMin returns the lowest value in the datastructures. Like in the build section we performed all getMin-tests 30 times.

3.6 getMax

We performed getMax operations on the pre-build datastructures of section 3.1. The amount of getMax operations in a test is 1000 since it returns always the same value. GetMax returns the highest value in the datastructures. Like in the build section we performed all getMax-tests 30 times.

3.7 extractMin

We performed extractMin operations on the pre-build datastructures of section 3.1. The amount of extractMin operations in a test is half the amount of the elements in the datastructure. ExtractMin returns the lowest value in the datastructures and then deletes this entry. Like in the build section we performed all extractMin-tests 30 times.

3.8 extractMax

We performed extractMax operations on the pre-build datastructures of section 3.1. The amount of extractMax operations in a test is half the amount of the elements in the datastructure. ExtractMax returns the highest value in the datastructures and then deletes this entry. Like in the build section we performed all extractMax-tests 30 times.

4 Weergave van eindresultaten

4.1 Tabellen

BUILD (random)

	List	**AVL Tree**	**Interval Heap**	**Hash table**
10.000	4,2 ±0,4	3 ±0	2 ±0	1 ±0
100.000	52,4 ±0,5	36,4 ±0,5	18,2 ±4	29,2 ±1,2
1.000.000	663, 6 ±48,5	1032,8 ±31,4	177 ±2,3	610,4 ±9,2

SEARCH (random)

	List	**AVL Tree**	**Interval Heap**	**Hash table**
10.000	6 ±0	1 ±0	1118,4 ±4,5	1 ±0
100.000	75,8 ±3,4	12,8 ±0,7	115007,4±3112,6	6,8 ±1,8
1.000.000	991 ±40,3	211,4 ±11,9	timeout	106,6 ±5,8

INSERT (random)

	List	**AVL Tree**	**Interval Heap**	**Hash table**
10.000	6 ±0	1 ±0	1118,4 ±4,5	1 ±0
100.000	75,8 ±3,4	12,8 ±0,7	115007,4±3112,6	6,8 ±1,8
1.000.000	991 ±40,3	211,4 ±11,9	timeout	106,6 ±5,8

DELETE (random)

	List	**AVL Tree**	**Interval Heap**	**Hash table**
10.000	16 ±0	1,2 ±0,4	not tested	1 ±0
100.000	2022,2 ±6,1	21,2 ±5,4	not tested	9,2 ±2,9
1.000.000	510015,8 ±95899,4	480 ±37,9	not tested	128 ±2,1

GETMIN

	List	**AVL Tree**	**Interval Heap**	**Hash table**
10.000	1 ±0	1 ±0	1 ±0	3230,8 ±11,1
100.000	1 ±0	1 ±0	1 ±0	37349 ±67,6
1.000.000	1 ±0	1,2±0,4	1 ±0	874572,4±3270,8

GETMAX

	List	**AVL Tree**	**Interval Heap**	**Hash table**
10.000	1 ±0	1 ±0	1 ±0	3174 ±7,9
100.000	1 ±0	1 ±0	1 ±0	46396,6 ±6468,1
1.000.000	1 ±0	1,2 ±0,4	1 ±0	905490,6 ±20239,3

EXTRACTMIN

	List	**AVL Tree**	**Interval Heap**	**Hash table**
10.000	28 ±0	1 ±0	6±0	1445,2 ±3
100.000	4012,2 ±4,7	12,8 ±3,9	72,2±0,4	164378,6 ±2414,6
1.000.000	836228 ±198929,6	109,4 ±0,8	928,8 ±13	timeout

EXTRACTMAX

	List	**AVL Tree**	**Interval Heap**	**Hash table**
10.000	4 ±0	1 ±0	6,2 ±0,4	1444,4 ±2
100.000	41 ±0	10 ±0	75±0	165799 ±299,2
1.000.000	482,6 ±25,7	110,2 ±0,7	951,6±1	timeout

4.2 Grafieken

4.3 Toelichting

4.4 Hypotheses en uitwerking statistische tests

4.4.1 Hypotheses changes

While testing the already changed second hypotheses: "The search action will be executed the fastest by the hash table and the AVL tree and the slowest by the Min-Max tree", we noticed a signficant difference between the speed of the hash table and the AVL tree. We realized that we should be more strict in our hypotheses and therefor we changed some other hypotheses before testing them. We changed the following hypotheses:

- The build action will be executed the fastest by the hash table and slowest by the Min-Max tree.
- The search action will be executed the fastest by the hash table and the AVL tree and the slowest by~~ the list and ~~the Min-Max tree.
- The insert action will be executed the fastest by the hash table~~, the AVL tree and the Min-Max tree~~ and slowest by the list.
- The delete action will be executed the fastest by the hash table ~~and the AVL tree~~ and the slowest by ~~the Min-Max tree and~~ the list.
- The getMin action will be executed the fastest by the list and the Min-Max tree and the slowest by the hash table.
- The extractMin action will be executed the fastest by ~~the AVL tree and~~ the Min-Max tree and the slowest by the hash table.
- The getMax action will be executed the fastest by the list and the Min-Max tree and the slowest by hash table.
- The extractMax action will be executed the fastest by the list and the Min-Max tree and the slowest by the hash table.

Welke veronderstellingen gaan we testen. Dit is een verdere uitwerking van de onderzoeksvraag. Minimaal 4 echt verschillende , meer mag ook. Denk aan: Vergelijkingen: Parameterinstelling 1 levert lagere looptijd parameterinstelling 2 Relaties: Er is een correlatie tussen het aantal klanten dat pizza's besteld en de looptijd van het algoritme Statistische test moeten worden uitgevoerd met Excel

5 Discussie en conclusie

6 Reflectie

6.1 In hoeverre heb je de onderzoeksvraag beantwoord?

6.2 Heb je je onderzoeksplan kunnen uitvoeren of heb je het bijgesteld? Zo ja, hoe en waarom?

6.3 Tegen welke moeilijkheden ben je aangelopen in het projetc?