Dynamic Space Efficient Hashing

Janno Peterson
Institute of Computer Science
University of Tartu

Abstract
There exist implementations of space efficient hash tables and dynamically resizable hash tables, but none of the commonly used hash tables can do both at the same time, i.e., grow closely with the number of elements in the hash table. The article “Dynamic Space Efficient Hashing” [1] proposes a new approach for this problem – DySECT (Dynamic Space Efficient Cuckoo Table). The experiments regarding this implementation show great results when working with hash tables that are up to 98% filled. This essay will give an overview of the article in question.

Introduction
For a hash table to be space efficient (i.e. use as much of the table’s capacity as possible) and still have good performance, it should have an optimally big capacity for the number of elements it contains. In scenarios, where the final number of elements in the hash table is unknown, space efficiency can only be achieved if the hash table is dynamically resizable. This, however, cannot be efficiently achieved with the current techniques available for hashing and migration [1]. A lot of currently used space efficient hash table implementations lose in space efficiency, i.e. grow in size, each time the table is resized [1]. This is because the common way to grow a hash table is to create additional and bigger hash tables and then migrate the elements from the original table to the new one.

Instead of this, the authors propose an implementation of bucket cuckoo hashing along with splitting the hash table into subtables, so that the subtables could be resized independently without affecting the whole hash table and every element in it. With this, the hash table can grow with smaller increments and always remain space-efficient.

Cuckoo hashing
Cuckoo hashing uses a hashing technique where there are H hash functions (h₁, …, h_H) [2]. They are used to calculate H possible positions for each element [2].

The element is stored in one of those positions. If during an insertion, all of the positions for an element are occupied, then the elements, that occupy its position, can be moved around to make room for the inserted element (by using another possible position for the occupant) [2].

In bucket cuckoo hashing, the hash table’s cells are combined into buckets of size B. Instead of H cells, the elements will be assigned to H buckets. The element will be stored in the bucket with the most amount of free space. The element can be stored in each cell of its designated bucket. Finding and erasing operations have a guaranteed constant running time, because since there are a constant of H buckets for each element and B cells in each bucket, then H*B cells need to be searched to find the element.

DySECT (Dynamic Space Efficient Cuckoo Table)
The DySECT hash table, that the authors proposed, is split into T subtables. Each subtable contains buckets and each bucket can contain up to B elements. Like in cuckoo hashing, each element is associated with H buckets.

When the table contains enough elements and should be resized, only a single subtable is grown. The subtable that is resized, is migrated into a table of twice its original size. The subtables in the whole hash table are grown in an order from first to last – this ensures that no subtable can be more than twice the size of any other subtable.

For the experiments, the authors used three hash functions (H=3) and a bucket size of 8 (B=8). These values were chosen, because the authors
conducted various tests and these values had better performance than other options.

To reduce the time it takes to compute the hash functions for each operation, the authors decided to use just one 64-bit hash function, which they split into two 32-bit values (since 32-bit values are enough to address the values for the buckets in the hash tables). Furthermore, to get the third hash value, they used double hashing with the two 32-bit values that they already had.

**Experiments**

In order to compare the performance and efficiency of DySECT, the authors also implemented other hash tables, which could also dynamically increase their size in small increments. They implemented variants of linear probing, robin hood hashing, and bucket cuckoo hashing (which is similar to DySECT).

Five tests were conducted. The first and second tests measured, how much the fill ratio of hash tables affects the performance of insertion and find operations. Test 1 focused on static hash tables, while test 2 focused on dynamic hash tables. Results regarding find operations of test 1 are represented on Figure 1.

![Figure 1. Performance of successful find operations [1].](image1)

For the find operations, DySECT’s performance was unsurprisingly constant – regardless of how full the table was.

For insertions into dynamic tables (test 2), DySECT performed far better than other methods. The more the tables were filled with elements, the better DySECT’s numbers were, compared to others. At 97.5% load factor, the DySECT performed 2.7 times faster than the next best method (cuckoo hashing with subtables). The results for insertion operations in test 2 are shown in Figure 2.

![Figure 2. Insertions into a dynamic growing table enforcing a minimum load factor δmin [1].](image2)

Test 3 measured the performance of hash tables for a practical test of counting the number of word occurrences in a text. The average time for operations was measured for various minimum load factors. The results resembled the results of test 2.

Test 4 measured the performance of hash tables under mixed workloads (different combinations of insert, find and erase operations). Again, DySECT performed better than any other method.

Test 5 was conducted to get an approximation of what is the maximum load bound of DySECT. The results show that it is close to the load bound of static cuckoo hashing.

**Conclusion**

The authors showed, that it is possible to implement a dynamically growing hash table, which not only performs well, but is also space efficient. Results show that in scenarios, where the hash table is very full (95% and more), DySECT is multiple times faster than simpler approaches.

**Acknowledgements**

The author of this essay has received the IT Academy Specialization Stipend for the Autumn semester of academic year 2017/18.

**References**
