Behavior of Circular Arrays Under Successive Rounds of Uniform Insertions and Deletions

Diogo Andrade        Gábor Rudolf

May 7, 2003

Introduction

In this paper we present a simple experiment modeling the behavior of a dynamic hash table of approximately constant size. The basic idea is to consider a circular array and alternately insert and delete an uniformly chosen random element (this approach was first suggested by Knuth). We study the long-term behavior of this system focusing on the insertion times and clustering phenomena. The paper represents a work in progress: we implemented the model and performed a large number of experiments to collect sufficient data and gain an understanding of the process but much of the theoretical work is still to be done.

1 Description of the basic experiment and implementation details

In this section we will briefly describe the implementation and specify how the experiments were done.

1.1 Implementation

There is nothing particularly new or difficult about the implementation of a circular array. The point of this section is to specify some details to clarify how the experiments were done.

There are four main points that need to be specified: Insertion, Deletion, Initialization and Data gathering. In the following we present a brief overview of these:
INSERTION: For the insertion we are given a position and a value. If the position is already occupied, the current element is inserted in that position and the existing elements in the array are shifted (rotated, since the array is circular) to the right until an empty position is reached.

DELETION: For the deletion we are given a position. If the current position is empty, we select a new one. If the current position is occupied, we simply set the element in that position to a special empty value.

INITIALIZATION: For the initialization only the number of elements to be inserted is specified. Naturally, this number must be smaller than the size of the array. Three different kinds of initialization were implemented:

1. Uniformly random: for this initialization shifts are not allowed since we want to simulate a uniformly random distribution of the elements in the array. So we stay in a loop that generates a random position and tries to insert the element there and if the position is occupied we generate another position and so on. The initialization ends when the number of elements to be inserted is reached.

2. One chunk: for this initialization an initial position is given and the elements are inserted one after the other. This initialization is particularly interesting for convergence tests.

3. Successive Insertions: for this initialization the positions are chosen randomly and the elements are inserted, shifting normally if it is necessary. This corresponds to building a hash table.

DATA GATHERING: Four different kinds of data were analyzed during the running of the program:

1. Histogram of Shifts: The number of elements shifted during an insertion is a measure of the cost of the insertion operation. During a round of tests we count the number of shifts in each iteration and store it in a histogram to evaluate the distribution of the shifts.

2. Number of clusters: A cluster is a contiguous block of elements. The number of clusters is a measure of how evenly the elements are distributed in the array. On every
insertion and deletion this number was updated and stored\(^1\) in data arrays.

3. **Sum of the Log of the Gaps**: A gap is a contiguous block of empty cells between two elements in the array. The sum of the log of the gaps is also a clustering measure. The bigger this sum the more evenly distributed the elements will be. To be more precise we actually used the sum of \(\log(gap + 1)\) since we also wanted to count gaps of size 1. This measure also had to be recalculated on every update on the structure.

4. **Histogram of Cluster sizes**: The histogram of clusters sizes is computed at the end of the execution of the program. This measure is closely related to the number of shifts that is expected to be done in each insertion. We will prove this relation later in this paper.

### 2 Behavior of a single array over time

(The following results are for \(10^6\) elements stored with a load factor of 0.75; the behavior is very similar for other parameter values.) Both the clustering measure and the number of clusters converges but while the 'stationary state' is identical, the initial behavior is different for the various initial arrays: When we start out with a single chunk, it takes about 200000 iterations to break it up; the clustering measure also reaches its final value around this time. Starting with a random array clustering will occur and the steady state is reached in approximately 100000 iterations. The most interesting case is initialization by successive insertions as it models the building of a hash table. The 'stationary state' is again reached after about 100000 iterations and the long-term clustering properties are better than those of the original array. Similar results for other array sizes and load factors suggest the conjecture that the stationary state is reached in \(O(n)\) iterations.

\(^1\)Actually each unit of data was stored after an insertion-deletion pair.
3 Parameters of the stationary state

The following figures show the distribution of individual insertion times in the stationary state (i.e. a histogram showing for each $I$ the number of insertions taking time $i$ and the distribution of cluster sizes (i.e. a histogram showing for each $i$ the number of clusters of
size $i$ at a given time). These two distributions are closely related:

**Proposition.** The probability of having to make $k$ shifts after an insertion can be determined by the sizes of the clusters:

$$P[k\text{shifts}] = \frac{\text{number of clusters of size } \leq k}{\text{size}}$$

**Proof.** It is easy to see that when inserting an element the number of shifts necessary equals the number of elements following the position of insertion in the cluster it falls into (including the position itself). Therefore we have $k$ shifts if and only if the position of insertion is the $k$-th from behind in its respective cluster; every cluster of size $\leq k$ has exactly one such position. The proposition immediately follows. \qed

![Figure 3: Histogram of the shifts](image-url)
Next we look at how our case from insertion into a random array of size \( n \):

**Proposition.** *When inserting into an (infinite) random array where an element is present with probability \( p \) the number of shifts follows a geometric distribution with parameter \( 1 - p \).*

**Proof.** We have \( k \) shifts exactly if the position of insertion and the next \( k - 1 \) positions are occupied while the next is free. This happens with probability \( p^k \cdot (1 - p) \).

The next figure shows the geometric distribution compared to our experimental results.

---

![Clusters histogram](image1.png)

**Figure 4:** Histogram of the clusters sizes

![Shifts and Geometric Distribution](image2.png)

**Figure 5:** Histogram of the shifts and Geometric Distribution curves.
We can get a significantly closer fit by modifying the geometric distribution; the following figure shows a least-square fit for our data with a distribution of the following type (modified geometric):

\[ p(0) = 1 - p, \ p(1) = p(1 - q), \ p(2) = pq(1 - r), \ p(k) = pqr^{k-2}(1 - r) \]

Figure 6: Histogram of the shifts and Modified Geometric Distribution curves.

4 Dependence on load factor and array size

In this section we look at how the input parameters influence the clustering parameters and insertion times. The first two figures demonstrate that (for sufficiently large array sizes) the clustering properties and insertion times depend only on the load factor and not on the number of elements: the clustering measure grows linearly with \( n \) while the average insertion time remains constant.
The following figures show the clustering measure, average cluster size, average insertion time and their respective standard deviations as functions of the load factor for 10^6 elements (the deviation of the clustering measure has been scaled up to be visible in the figure).
Figure 9: Average shifts per iteration and Standard Deviation of the shifts VS Load Factor.

Figure 10: Average and Standard Deviation on the number of Cluster VS Load Factor
5 Connection with hash tables

Our experiment models the behavior of a dynamic hash table using linear probing under the assumption of uniform hashing. The following result is due to Knuth,1963:

Theorem. We build a hash table by consecutive insertions. The probability that the \( k \)-th insertion requires \( m \) shifts in an array of size \( N \) is:

\[
P(N, k, m) = \frac{1}{N^{k-1}} \sum_{i=m}^{N} Q(N, k, i - 1) = 1 - \frac{k - 1}{N} - \frac{1}{N^{k-1}} \sum_{i=1}^{m-1} Q(N, k, i - 1).
\]

where

\[
R(N, k, t) = \binom{k - 1}{t - 1} \left( \frac{t}{N} \right)^{t-2} \left( 1 - \frac{t}{N} \right)^{n-t-1}.
\]

This gives the number of shifts when inserting an element into a newly built hash table. The next figure provides a comparison with our long-term experimental results inserting an element into an array of the same size and with the same load factor:
It can be seen that the long-term behavior is slightly better, i.e. the clustering properties improve with successive insertions and deletions (this agrees with our results in section 2 where initialization by successive insertions corresponds to building a hash table).

**Remark.** Knuth’s formula causes considerable computational difficulties and it couldn’t be computed for load factors above 0.8. Also, the accuracy of the values for load factors between 0.75 and 0.8 is questionable, although they definitely provide lower bounds for the actual values.

### 6 Future work

The next step is to prove the convergence results of section 2. It would also be desirable to find a formula for the distribution of cluster sizes described in section 3, which would also give the distribution of insertion times. If such a formula cannot be obtained, then the modified geometric approximation could be useful to obtain bounds. Similarly, we are going to try to find the exact relationship between the load factor and the clustering parameters and insertion times. We have just started to explore the relationship with hash tables; a full distributional analysis of linear probing hashing is available (Flajolet, Poblete, Viola, 1997) which could provide interesting comparisons.