

Cuckoo Hashing

Arpita Korwar

May 12, 2010

1 Introduction

This document is written to serve as an aide for reading the seminal paper Cuckoo hashing [PR04].

Cuckoo hashing is used to store a dynamic set $S \subseteq U$, $|S| = n$, on which the operations `insert`, `lookup` and `delete` can be performed. This scheme worst case $O(1)$ time for `lookup` and `delete` operations. It takes expected amortized time of $O(1)$ and $O(n)$ space (infact, it takes $< 3n$ space).

It needs a source of $O(n^2)$ -universal family of hash functions.

2 The Cuckoo hashing scheme

Two tables T_1 and T_2 , each of size r are used. Two hash functions h_1 and h_2 are picked uniformly and randomly from the family of n^2 universal hash functions. Each element x in the set S is stored in either T_1 or in T_2 at the locations $T_1(h_1(x))$ or in $T_2(h_2(x))$, but never in both the locations. Hence to lookup an item, we only need to check these two locations. Similarly for deletion. Hence both the operations take $O(1)$ worst case time. The size of each table, r is chosen to be $(1 + \epsilon)n$. Hence, the space required for the algorithm is slightly larger than $2n$.

3 Insertion algorithm

When a new element x added to S we put it in the location $T_1(h_1(x))$. If there was no element in that location, the insertion procedure stops at that point. But if there was an element y in that location, it gets “kicked out”. We now have to insert it into the table T_2 at the location $T_2(h_2(y))$. Thus, when an element gets kicked out, we have to insert it

into its alternate location in the other table. We continue this process until an empty location is found. But it may so happen we keep revisiting the same locations repeatedly, e.g. when $h_1(x_1) = h_1(x_2) = h_1(x_3)$ and $h_2(x_1) = h_2(x_2) = h_2(x_3)$, and x_1 is inserted, the elements displace each other in the sequence $x_1x_2x_3x_1x_2\dots$. To overcome this situation, we stop the insertion loop after some *MaxLoop* iterations and rehash the tables. The algorithm for insertion is as follows:

```

insert( $x$ )
{
  if lookup( $x$ ) = true, return;
  loop MaxLoop times
     $x \leftrightarrow T_1(h_1(x))$ ; if  $x == \text{null}$  return;
     $x \leftrightarrow T_2(h_2(x))$ ; if  $x == \text{null}$  return;
  end loop;
  rehash();
  insert( $x$ );
}

```

In each iteration there are a max. of 2 displacements. Hence before a rehash, a max. of 2MaxLoop displacements can occur. In the rehash procedure, two hash functions for h_1 and h_2 are picked from the family of universal hash functions. The elements are again inserted according to the new hash values. Hence, rehashing takes $O(n) \cdot \langle \text{Expected insertion time for 1 element} \rangle$.

We have already proved the space bound and the time bound for lookup and delete. We will now prove that the expected amortized time for insertion is $O(1)$.

4 Analysis

We will first study the behavior of the insertion procedure.

4.1 Behavior of the insertion procedure

In this subsection, we study the behavior of the procedure when it is allowed to run infinite number of iterations.

1. The simplest case is when no previously visited cell is visited again. In this case, the sequence of nestless keys x_1, x_2, \dots ends with a vacant cell. We call this sequence *leg 1*.

Even when a repetition does occur, we call the part of the sequence until this repetition as leg 1.

2. What happens when a previously visited cell is revisited? In the sequence of nestless keys, the element x_i gets displaced the second time, at index j , i.e. the elements x_i and x_j are the same. Now, x_i would go to its previously occupied location (which is its only alternative address), thus kicking out x_{i-1} . Hence, $x_{j+1} = x_{i-1}$. x_{i-1} would kick out x_{i-2} and so on until $x_1 (= x_{i+j-1})$ becomes nestless again. We call this part of the sequence of nestless keys (from index j to index $i + j - 2$) as *leg 2*.
3. When x_1 becomes nestless again, we try to put it in its alternate location in T_2 . Again, a sequence of displacements may occur, until an element x_l visits a previously visited cell or gets a vacant cell. The subsequence from index $i + j - 1$ to l is called *leg 3*. If leg 2 occurs, then leg 3 also occurs.
4. If x_l visits a previously visited cell, then we get a closed loop. We cannot hope to accommodate all the keys using the present hash functions h_1 and h_2 , because, before x_1 arrived, all the cells in this loop were occupied. x_1 hashed to locations in this loop. The sequence of nestless keys would continue forever. When a closed loop occurs, we say *leg 4* occurs.

Note: The reader is encouraged to work with some examples of his own. To confirm that he has understood the semantics correctly, he needs to confirm that with the values of i, j and l that he got for the sequence of nestless keys, the number of cells participating is $l - i$ and that there are $l - i + 1$ elements.

4.2 Probability bounds

Recall that our aim is to find expected amortized insertion time. This depends on the expected number iterations of the insertion procedure. We would like to find the probability that the insertion procedure executes the t th iteration. Also, probability of a rehash is equal to the probability that the *MaxLoop* th iteration is executed.

4.2.1 Probability of the t th iteration

When $t > \text{MaxLoop}$ this probability is 0. When t th iteration is executed, the sequence of nestless keys is $2t$. When $t \leq \text{MaxLoop}$, the t th iteration is executed in the following cases:

1. If a closed loop has occurred then the sequence would loop forever, and definitely till the t th iteration. Thus, we have reached leg 4 of the sequence.
2. We see a sequence of $2t$ nestless keys, but no closed loop has occurred yet. i.e. we are in leg 1, 2 or 3.

To find the probability of a closed loop of length $\leq 2t$: The sample space is the Cartesian product of all possible pairs (h_1, h_2) and all possible sets of cardinality n .

One way of approaching this problem is by fixing h_1 and h_2 - and finding the probability of atleast one loop occurring. But this approach would not allow us to exploit the property of universality of hash functions.

So, we follow another approach. We fix our loop starting with x_1 and we ask for the probability that this loop occurs. Then, we will use union bound to bound the probability that any loop occurs.

What are the properties of a loop? It is a sequence of nestless keys x_1, x_2, \dots that has reached leg 4. This sequence of nestless keys has a set distinct elements that occur in some sequence. If there are v distinct elements in the sequence, then there are only $v - 1$ cell locations to store them.

We partition the loops according to the number of distinct elements in that loop, i.e. v , ($3 \leq v \leq 2t$). We bound the probability of loops with v distinct keys. There are less than $v^3 r^{v-1} n^{v-1}$ such loops (v^2 values for i and j , v values for x_l to hash to, ${}^n P_{v-1} < n^{v-1}$ sequences of distinct elements and ${}^r P_{v-1} < r^{v-1}$ cell locations). Each of the addresses $h_1(x_1), h_2(x_1), h_1(x_2), h_2(x_2), \dots$ have fixed values in this case. Hence there are $2v$ such restrictions where one value is chosen amongst r possible values. Hence probability of this loop is $\frac{1}{r^{2v}}$. Hence,

$$\begin{aligned}
\Pr[\text{closed loop}] &\leq \sum_{v=3}^{2t} \frac{1}{r^{2v}} \cdot v^3 r^{v-1} n^{v-1} \\
&= \frac{1}{rn} \sum_{v=3}^{2t} v^3 \left(\frac{n}{r}\right)^v \\
&< O\left(\frac{1}{n^2}\right) \sum_{v=3}^{\infty} v^3 \left(\frac{n}{r}\right)^v
\end{aligned}$$

In the last step, since $r > (1 + \epsilon)n$, $\frac{1}{rn} < \frac{1}{(1+\epsilon)n^2}$. $v^3 \left(\frac{n}{r}\right)^v$ is a converging series, and thus, gives a constant. hence $\Pr[\text{closed loop}] = O\left(\frac{1}{n^2}\right)$.

Sequence of $2t$ nestless keys such that a closed loop has not occurred yet:

Lemma 1. *If there is a sequence of $2t$ nestless keys such that a closed loop has not occurred yet, then there exists a subsequence of distinct keys of length $\geq \frac{2t-1}{3}$ starting with x_1 .*

The proof is in the paper. From this lemma, \Pr [sequence of $2t$ nestless keys] $\leq \Pr$ [sequence of $\frac{2t-1}{3}$ distinct nestless keys starting with x_1].

Let $v = \frac{2t-1}{3}$. What does the sequence look like? It is a sequence of distinct keys x_1, x_2, x_3, \dots such that

$$h_1(x_1) = h_1(x_2), h_2(x_2) = h_2(x_3), h_1(x_3) = h_1(x_4), \dots$$

OR

$$h_2(x_1) = h_2(x_2), h_1(x_2) = h_1(x_3), h_2(x_3) = h_2(x_4), \dots$$

How many such sequences can be formed? We can either start with h_1 or with h_2 . The $v-1$ elements can be chosen in less than n^{v-1} ways. There are exactly $v-1$ equalities in the above sequence. Hence the probability that the sequence starting with h_1 (or h_2) occurs is $\frac{1}{r^{v-1}}$. Hence the probability that such sequence exists is:

$$2 \left(\frac{n}{r}\right)^{v-1} \leq \frac{2}{(1+\epsilon)^{\frac{2t-1}{3}+1}}$$

4.2.2 Expected number of iterations

From standard probability theory we know that for random variable X taking positive integer values, if $\Pr[x \geq t] = p(t)$ then $E[X] = \sum_t p(t)$. We let X = number of iterations and $p(t)$ = probability that t th iteration is executed. Then the expected number of iterations is given by

$$\begin{aligned} & 1 + \sum_{t=2}^{MaxLoop} \Pr[t \text{ iterations}] \\ = & O\left(1 + \frac{1}{1 - (1-\epsilon)^{\frac{-2}{3}}}\right) \end{aligned}$$

Thus, the expected number of iterations is a constant.

4.2.3 Expected time taken for each iteration

The expected time taken for each iteration is given by:

$$\begin{aligned} \mathbb{E}[\text{time taken for 1 insertion}] &= \Pr[\text{no rehash}] \cdot \mathbb{E}[\text{no. of iterations}] \\ &\quad + \Pr[\text{rehash}] \cdot \mathbb{E}[\text{time taken for rehash}]. \end{aligned}$$

Time taken for rehash in turn depends on insertion time. This would seem like a cyclic dependency, except that $\Pr[\text{rehash}]$ has a small value, because of which we can find the expected time for rehash and in turn expected time for insertion.

$$\begin{aligned} \Pr[\text{rehash}] &= \Pr[\text{MaxLoopth loop is executed}] \\ &= O\left(\frac{1}{n^2}\right) + \frac{2}{(1+\epsilon)^{\frac{2\text{MaxLoop}-1}{3}+1}} \end{aligned}$$

We set MaxLoop to such a value that the second term also becomes $O\left(\frac{1}{n^2}\right)$. We set $\text{MaxLoop} = \lceil 3 \log_{1+\epsilon} r \rceil$. Hence, probability of rehash in any one insertion is $O\left(\frac{1}{n^2}\right)$ and probability of rehash in n insertions is $O\left(\frac{1}{n}\right)$.

To find expected time taken for rehash: When rehashing, we insert n items. If $T(n)$ is the expected time taken to rehash, then,

$$\begin{aligned} T(n) &= \Pr[\text{no rehash in } n \text{ insertions}] \cdot \langle \text{Time taken for } n \text{ insertions} \rangle \\ &\quad + \Pr[\text{rehash in } n \text{ insertions}] \cdot T(n) \\ T(n) &= \left(1 - \frac{1}{n}\right) \cdot O(n) + \frac{1}{n} T(n) \\ \left(1 - \frac{1}{n}\right) T(n) &= \left(1 - \frac{1}{n}\right) O(n) \\ T(n) &= O(n) \end{aligned}$$

Hence, if rehash does occur, it takes $O(n)$ time. Probability of rehash at each step is $O\left(\frac{1}{n^2}\right)$. Hence expected amortized rehash time at each insertion is $O\left(\frac{1}{n}\right)$.

Hence, expected amortized time taken for each insertion is $O(1)$.

References

- [PR04] Rasmus Pagh and Flemming Friche Rodler. Cuckoo hashing. *J. Algorithms*, 51(2):122–144, 2004.