Advanced Data Structures Jan-Apr 2012

Lecture 8 — 10 Feb, 2012

Lecturer: Venkatesh Raman *Scribe: Esha Ghosh*

1 Overview

In the last lecture we we discussed hashing as a solution to dictionary/membership problem. Results on hashing were presented with emphasis on static perfect hashing via FKS.

In this lecture we discuss about dynamic Cuckoo hashing.

We follow the lecture notes of [2] in this lecture.

2 Cuckoo - Dynamic Hashing

Cuckoo hashing is named on the nesting habit of the Cuckoo bird, which is known to place its eggs in the unattended nest of another bird. This hashing technique is due to [1].
In dictionary/membership problem, we want to keep a set S of n items from a universe U. For the membership problem, the goal is to create a data structure that allows us to ask whether a given item x is in S or not. The problems have two versions: *static* and *dynamic*. In the static version, S is predetermined and never changes. In the dynamic version allows items to be inserted to and removed from S.

First we list the dictionary operations as follows:

- *membership(x)* – determine $x \in S$ or not, and return information associated with x.
- *insert(x)* – (dynamic only)
- *delete(x)* – (dynamic only)

Cuckoo hashing achieves $O(1)$ *expected amortized* time for insertion and $O(1)$ *worst-case* time for membership and deletion queries using two hash functions h_1 and h_2, picked uniformly at random (uar) from k-universal hash families, where $k = O(\log n)$.

The table T is of size $m \geq 2n$. (We will use $m = 4n$.) We will analyse the running time of the three dictionary opretations in details.

Membership(x):
Throughout the life of the data structure, it maintains the following invariant: an item x that has already been inserted is either at $T[h_1(x)]$ or at $T[h_2(x)]$. Thus, a membership query takes at most two probes in the worst case. Hence the worst case running time for membership is $O(1)$.

Delete(x):
Finding whether x exists in the hash table requires at most two probes (We have already observed this invariant). If x is found in the table, we just delete it. Hence the worst case running time for deletion is $O(1)$.

Insert(x):
To insert an element x, we carry out the following process.

Cuckoo Hashing Algorithm

1. Compute $h_1(x)$
2. IF $T[h_1(x)]$ is empty, insert x in $T[h_1(x)]$. STOP.
   ELSE SWAP$(x, T[h_1(x)])$.
   - IF $T[h_2(x)]$ is empty, insert x in $T[h_2(x)]$. STOP.
   - ELSE SWAP$(x, T[h_2(x)])$
3. Repeat until we find an empty spot, or until we have evicted $6 \log n$ items. In the later case, we pick a new pair of hash functions and rehash.

2.1 Analysis of the Algorithm

We analyse the time required to insert an elememt using Cuckoo hashing. Consider the process of inserting an item x_1. Let $x_1, x_2, \ldots, x_t$ be the sequence of items, with the exception of x_1, that are ejected during the process, in the order they are ejected.

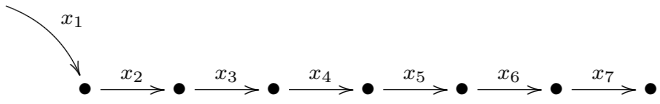
There are three possible situations that might arise during the process of insertion. We will analyze the running time in all three cases. Visualizations of the three different behaviors are given in figure 1.

The key observation is that when inserting a new element, we never examine more than $6 \log n$ items. Since our functions are $6 \log n$-independent, we can treat them as truly random functions.

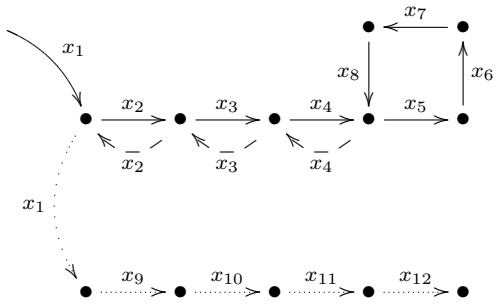
- **Case (a): No Cycle.** The process of ejection continues, without coming back to previously visited cell, until it finds an empty cell. Let us calculate the probability that the insertion process ejects t items. The process carries out the first eviction if and only if $T[h_1(x_1)]$ is occupied. By the union bound, this event has probability at most

$$\sum_{x \in S, x \neq x_1} (\Pr[h_1(x) = h_1(x_1)] + \Pr[h_2(x) = h_1(x_1)]) \leq 2\frac{|S|}{|T|} = 2\frac{n}{4n} = \frac{1}{2}.$$

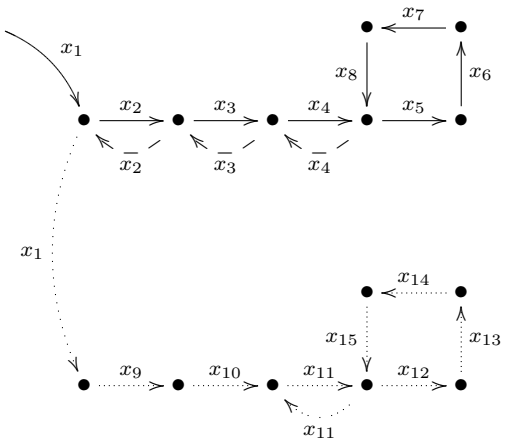
  Similarly, the process carries out the second if and only if it has carried out the first eviction, and the cell for the first evicted element is occupied. Thus, by the same reasoning, it carries



(a) no cycle



(b) one cycle



(c) two cycles

Figure 1: Three different behaviors of the process of inserting element x_1.

out the second eviction with probability at most 2^{-2}. And we can argue that the process carries out the t^{th} eviction with probability at most 2^{-t}. Therefore, the expected running time of this case is $\sum_{t=1}^{\infty} t2^{-t} = O(1)$.

Also, $\Pr[Rehash] \leq 2^{-6\log n} = \frac{1}{n^6}$.

- **Case (b): One Cycle.** In this case, for some j, the other cell that x_j can be in is occupied by a previously ejected item x_i. Then x_i will be ejected a second time, and will go to the other position it can be found in–namely, back to the location of x_{i-1}. Thus, $x_i, x_{i-1}, \ldots, x_1$ will be ejected in that order, and x_1 will be sent to $T[h_2(x)]$, and the sequence continues and eventually finds an empty cell. We claim that in the sequence $x_1, x_2, \ldots, x_t$ of ejected items, there exists at least one path of length at least $t/3$ that starts with x_1. The sequence can be partitioned into 3 parts — the solid line part, the dashed line part, and the dotted line part — and one of them must contain at least $t/3$ items. By the same reasoning as in the previous case, we have that the probability that the insertion process ejects all these items is at most $2^{-t/3}$. So, the expected running time in this case is at most $\sum_{t=1}^{\infty} t2^{-t/3} = O(1)$.

  Also, $\Pr[Rehash] \leq 2^{\frac{-6\log n}{3}} = \frac{1}{n^2}$

- **Two Cycles:**This is the case where even after a series of ejections $x_i, x_{i-1}, \ldots, x_1$, when x_1 is sent to $T[h_2(x)]$, the sequence continues and runs into a previously visited cell. We shall calculate the probability of a sequence of length t with two cycles.

  - The first item in the sequence is x_1.
  - There are at most $t-1$ items in the sequence, each of which is drawn from a set of size n. Thus, the number of sequences starting at x_1 is at most n^{t-1}.
  - There are at most t choices for the first loop occurs, at most t choices for where this loop returns on the path so far, and at most t choices for when the second loop occurs and t choices for where this second loop returns on the path.
  - Additionally, this pattern can occur anywhere in our table (data structure) of hash values. While the first hash value is $h_1(x_1)$, the remaining $t-1$ values are unconstrained. Our table is of size $m = 4n$, so there are $(4n)^{t-1}$ possibilities here.

  Thus, there are at most $t^4 n^{t-1}(4n)^{t-1}2^t$ configurations. Now let us consider the total number of configurations possible. The hash functions have a range of size $m = 4n$, so the number of configurations possible is $(4n)^t \cdot (4n)^t$. Thus, the probability that a two-cycle configuration occurs is
  $\Pr[\textit{Two-cycle configuration of sequence length } t \textit{ happens}] \leq \frac{t^4 n^{t-1}(4n)^{t-1}2^t}{(4n)^{2t}} = \frac{t^4}{4n^2 2^t}$.
  Therefore, the probability that a two-cycle occurs at all is at most

$$\sum_{t=2}^{\infty} \frac{t^4}{4n^2 2^t} = \frac{1}{4n^2} \sum_{t=2}^{\infty} \frac{t^4}{2^t} = \frac{1}{4n^2} \cdot O(1) = O\left(\frac{1}{n^2}\right).$$

Now we calculate tha amortized expected cost of one insertion using a simple recurrence relation. First, note that an insertion can cause $O(\frac{1}{n^2})$ rehashing. Therefore, for n insertions, $\Pr[Rehash] \leq O(\frac{1}{n})$. So with probabilty $1 - O(\frac{1}{n})$, n insertions succeeds. Hence we get the recurrence relation,

$\mathrm{E}(n \text{ insertions}) = [1 - O(\frac{1}{n})]O(n) + \frac{1}{n}\ \mathrm{E}(n \text{ insertions})$
Solving this, we get, $\frac{(n-1)}{n}\mathrm{E}(n \text{ insertions}) = O(n)$, i.e, $\mathrm{E}(n \text{ insertions}) = O(n)$.

Therefore expected amortized cost of one insertion is $O(1)$.

References

[1] R. Pagh, F. Rodler, *Cuckoo Hashing*, Journal of Algorithms, 51(2004), p. 122-144.

[2] Erik Demaine, David Wilson *6.851: Advanced Data Structures*, Lecture 14, Spring 2010.