CS 240 — Lecture 13 & 14

Bartosz Antczak *Instructor: Eric Schost* *February 14, 16, 2017*

13.1 Hashing

Hashing is another implementation of a dictionary.

Structure

We first outline a requirement: for any given $M \in \mathbb{N}$ (i.e., any piece of data), there exists a key as an integer with $0 \leq k < M$. This data structure is an array of values A with size M. We first consider the hash function:

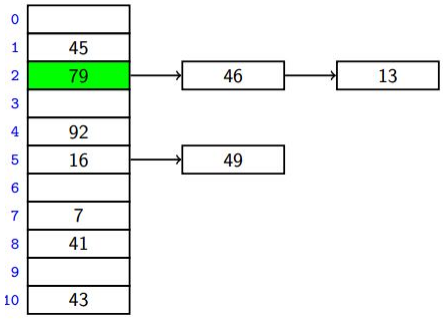
$$h : U \to \{0, 1, \cdots, M-1\}$$

Where U is some universe. Generally h is not injective, so many keys can map to the same integer.
Its structure is very simple: it is an array T of size M, and any item with key k is stored in $T[h(k)]$. In theory, all of its methods (insert, search, and delete) should cost $O(1)$.
Immediately, however, we spot an issue: how do we deal with collisions? (i.e., when $h(k_1) = h(k_2)$). There are two solutions: *buckets* and *open addressing*:

13.1.1 Buckets (or Chaining)

This method involves every table entry being what's called a "bucket" containing zero or more key-value pairs. The simplest approach is to structure each bucket using an unordered linked list.

Example 13.1.1. *A hash table using buckets*



Analysis

- `search`: $\Theta(1 + \alpha)$ average case, $\Theta(n)$ worst-case
- `insert`: $O(1)$ worst-case, since we always insert in front
- `delete`: same cost as search

13.1.2 Open Addressing

The main idea is that each hash table enty holds only one item but any key k can go in multiple locations. To search for and insert an element into the table, we follow a *probe sequence* of possible locations for key k. The simplest idea is what's called linear probing. *An example can be seen on the course slides.*

13.1.3 Double Hashing

Say we have two hash functions that are independent. The probability that $h_1(k) = a$ and $h_2(k) = b$ for any particular a and b is:

$$\frac{1}{M^2}$$

For double hashing, we define $h(k, i) = h_1(k) + i \cdot h_2(k) \mod M$. The basic methods (search, insert, and delete) work just like for linear probing.

13.1.4 Cuckoo Hashing

This is a relatively new idea, discovered in 2001. Again, we use two independent hash functions h_1 and h_2.