R4 Hashing

3.23 https://github.com/GUMI-21/MIT6.006_note

Comparison Model

The comparison model of computation acts on a set of *comparable* objects.
Then the *worst-case number of comparisons* that must be made by any comparison search algorithm will be *the height of the algorithm's decision tree*, i.e., the length of any longest root to leaf path.

Direct Access Arrays

integer keys
Now suppose we want to store a set of n items, each associated with a unique integer key in the bounded range from 0 to some large number u − 1
We can store the items in a length u direct access array, where each array slot i contains an item associated with integer key i, if it exists.

- When u is very large compared to the number of items being stored, storing a direct access array can be wasteful, or even impossible on modern machines.
  For example, suppose you wanted to support the set find(k) operation on ten-letter names using a direct access array. The space of possible names would be u ≈ 2610 ≈ 9.5 × 1013; even storing a bit array of that length would require 17.6 Terabytes of storage space

Hashing

Is it possible to get the performance benefits of a direct access array while using only linear O(n) space when n << u?
if h(k1) = h(k2), we say that the hashes of k1 and k2 *collide*

- how to solve collide
  The first strategy is called *open addressing*, which is the way most hash tables are actually implemented, but such schemes can be difficult to analyze. We will adopt the second strategy called *chaining*.

chainning

Each hash table index holds a pointer to a chain.
It is common to implement a chain using a linked list or dynamic array, but any implementation

will do, as long as each operation takes no more than linear time.

Hash Functions

- Division Method(bad)
  h(k) = (k mod m), or in Python, k % m.
- Universal Hashing(good):
  see note LEC4