Lecture 8, Oct 4, 2023

Hash Tables

- A hash table is an implementation of a dictionary that uses *hashing*
- Idea: If the set of possible keys U is small, we can use a direct access table $T[0..u-1]$, so that item with key $k \in U$ is stored at $T[k]$
  - This gives us $\Theta(1)$ search, insert, and delete, at the cost of using a lot of memory – each possible key must have a slot associated with it, regardless of whether there's something stored there
  - If U is large, then it will be impractical or impossible to do this
  - Can we do better?
- Let m be the size of the hash table $T[0..m-1]$ and let n be the number of keys in S (we typically choose m so that it is $\Theta(n)$)

Definition

Given a set U of possible keys and a hash table of size m, a *hash function* h is a function that takes a key in U to an index in the table, i.e.

$$k \in U \to h(k) \in \{\, 0, 1, \ldots, m-1 \,\}$$

We say that the key k *hashes* into the slot $h(k)$ of T.

- The basic idea is that given a key k, we will store it at the slot $h(k)$
- Since m is smaller than the size of U, inevitably we will eventually have two distinct k hashing to the same $h(k)$; this is called a *hash collision*
  - One way to get around this is hashing with chaining, where each slot in T stores a linked list of all the keys with the same hash; on a collision, simply add the item to the start of the list
- With hash chaining, we have $\Theta(1)$ insertion, $\Theta(1)$ deletion (assuming we already have a pointer to the element), but $\Theta(n)$ search since in the worst case, all the keys can end up being hashed to the same slot
- Intuitively we want a hash function that spreads out the keys; we state this as the Simple Uniform Hashing Assumption (SUHA): any key $k \in U$ is equally likely to hash into any of the m slots of T, independent of all other keys
  - The probability of k hashing into i is $\dfrac{1}{m}$ regardless of i
- Starting from an empty T and inserting n keys, by SUHA we expect $\dfrac{n}{m}$ keys in each slot on average; $\dfrac{n}{m} = \alpha$ is called the *load factor* of the table
  - If we perform a search for k, either the key is not in the table or it is; if the key is in the table, we expect $\dfrac{\alpha}{2}$ comparisons on average; if it is not, we expect α comparisons
  - Therefore the search on average takes $\Theta(\alpha)$
- As long as we keep n within a constant factor of m, we have constant $\Theta(1)$ time on all operations!
  - n grows with each insertion, but if it gets too large we can resize the table
- One way to implement such a hashing function is to simply take $h(k) = k \mod m$ where m is a prime number; then if we assume k is uniformly distributed, this hash function will satisfy SUHA