Readings 11.1, 11.2, 11.3, except 11.3.3

Self test 11.1-1, 11.2-1, 11.2-2

Lecture 09

Hashing

Problem 1: Read a text file, keep track of number of occurences of each character (ASCII codes 0 - 127).
Solution $\rightarrow$ Direct-access table: store number of occurences of each character in array with 128 positions. All operations $\Theta(1)$ and memory usage small.
Problem 2: Read a data file, keep track of each integer value (from 0 to $2^{32}-1$).
Solution $\rightarrow$ Wasteful: use array with 2^{32} positions as above. Time is $\Theta(1)$ but memory required is huge. Instead, allocate array with 10000 positions (for example), and figure out how to map each integer to one position - "hashing."

universe:= set of all possible keys
hash table T:= array with m positions, each location called a "slot" or a "bucket"
hash function:= $h : U \rightarrow \{0, 1, \ldots, m-1\}$, $h(k)$ maps keys to buckets

Collisions

$$|u| > m \Rightarrow \exists\ k \neq k', h(k) = h(k')$$

Strategies for Collision Resolution

(a) pointer to new locatation / overflow

(b) rule to find the overflow

Chaining ("closed hashing")

Each location of T stores linked list of items that hash to this location.

Simple Uniform Hashing

$$P[h(x) = i] = \sum_{x \in U, h(x)=i} P[x] = \frac{1}{m} \text{ for } i = 0, 1, \ldots, m-1$$

It is equally likely for a key to go into any bucket.

$$\sum_{x \in U, h(x)=i} P[x] = \frac{1}{m}$$

Define load factor α = expected number of items in each bucket, $\alpha = \frac{n}{m}$
Random variables:
  $N(x)$ = number of elements examined on search for x
  L_i = number of elements in bucket i

$$\sum_{i=0}^{m-1} L_i = n$$

Probability space? $\rightarrow$ pick uniformly at random from U

$$
\begin{aligned}
E[T] &= \sum_{x \in U} P[x] \cdot N(x) \\
&= \sum_{i=0}^{m-1} \left(\sum_{x \in U, h(x)=i} P(x) \cdot N(x) \right) \\
&\leq \sum_{i=0}^{m-1} P[h(x) = i] \cdot L_i = \frac{1}{m} \sum_{i=0}^{m-1} L_i = \frac{n}{m} = \alpha
\end{aligned}
$$

Concern: In practical applications $|U| >> m$. So analysis above computes expectation for keys most likely not in the table. Intuitively: searching for a key not in the table requires traversing one complete linked list with average size α.