

CSC 301: Analysis of Algorithms

Lecture 14: Hash Tables

William Rebelsky
[rebelskyw]

Grinnell College

Wednesday 21st February, 2024

Reminders

- Homework 2 grades will be released soon. A 4 is required for an S
- Homework 3 is due Tonight
- Homework 4 has been released on gradescope, it is due next Wednesday
- Quiz 3 grades have been released, an 8 is required for an S

Table of Contents

1 Hash Tables

2 Hash Functions

Direct-Address Tables

- Works well when number of potential keys is small and each element has a distinct key
- Search, Insert, Delete take $O(1)$ (in the worst case)
- Space is $O(\text{—universe of potential keys—})$

Hash Tables

- Extension of direct-address tables when the number of potential keys is large or the set of keys stored is small compared to the number of potential keys
- Storage space is $O(\text{—number of keys used—})$
- Searching takes $O(1)$ (expected time)

Hash Tables ADT

- Needs to support Mapping/dictionaries
- Needs to support the following operations
 - Insert(Key,Val) (or Set): add Key,Val pair to the table
 - Search(Key) (or Get): return Val
 - Delete(Key) (or Remove): remove Key,Val pair from table

Hashing

- Element with key k hashes to slot $h(k)$ which is the hash value of key k when the hashing function h is applied
- Two keys may hash to the same spot: Collision
- There will always be a potential collision

Table of Contents

1 Hash Tables

2 Hash Functions

Independent Uniform Hashing

- An Ideal hash function returns $h(k)$ that is randomly and identically chosen from the range $\{0, \dots, m-1\}$ such that every call $h(k)$ on the same input k is identical
- This is called an independent uniform hash function or a random oracle
- Theoretical abstraction, not feasible in practice

Dealing with Collisions

Two Main solutions:

- Chaining
 - Each nonempty slot points to a linked list
 - Load factor: n/m or the average number of elements stored in a chain. α
- Probing with open addressing
 - All elements are stored in the table itself
 - Load factor $\alpha < 1$
 - When a new element is to be inserted, if it collides, try it's next preferred location

Chaining

- Worst case running time for Insertion: $O(1)$
- Assuming doubly linked lists, Deletion is also $O(1)$
- Searching (find):
 - Worst Case behavior: all n keys hash to the same slot. Searching is $\Theta(n)$ plus the time for to compute the hash function
 - Average Case behavior (assuming independent uniform hashing):
 - Expected length of the list at $T[j]$ is n/m
 - Assume calculating the hash value is $O(1)$
 - Theorem 11.1: An unsuccessful search takes $\Theta(1 + \alpha)$ on average
 - Theorem 11.2: A successful search takes $\Theta(1 + \alpha)$ on average
 - If we assume number of elements is proportional to number of hash slots ($n=O(m)$), then we have that Searching is $O(1+m/m)=O(1)$.

Probing

- Every key has a probe sequence that is a permutation of the possible slots
- When searching, search until element is found or a blank is found
- Deleting is complicated, need to have a special value so that it doesn't stop a search early
- Deleting also means search times no longer depend on α
- Double Hashing: $h(k, i) = (h_1(k) + ih_2(k)) \bmod m$. $h_2(k)$ must be relatively prime to m
- Linear Probing: $h(k, i) = (h_1(k) + i) \bmod m$

Probing

- Corr 11.7: Inserting takes on average at most $1/(1 - \alpha)$ probes
- Searching (find):
 - Worst Case behavior: Searching is $\Theta(m)$ plus the time for to compute the hash function
 - Average Case behavior (assuming independent uniform hashing):
 - Theorem 11.6: Expected number of probes is at most $1/(1 - \alpha)$
 - Theorem 11.8: Expected number of probes in a successful search is at most $\frac{1}{\alpha} \ln \frac{1}{1-\alpha}$
 - Theorem 11.2: A successful search takes $\Theta(1 + \alpha)$ on average
 - If we assume number of elements is proportional to number of hash slots ($n=O(m)$), then we have that Searching is $O(1+m/m)=O(1)$.
- Theorem 11.9: If h_1 is 5-independent and $\alpha < \frac{2}{3}$ then searching, inserting, and deleting take expected time of $O(1)$ in linear probing

Expanding Hash Tables

- If you need to expand a hash table (e.g. in probing when the table is full):
 - Choice 1: extend the size of the table by 1. Adds time to every call
 - Choice 2: extend the size of the table by doubling: Adds time rarely, amortized cost is constant over number of keys in the table
 - Choice 2 also allows for maintaining the $2/3$ fraction more easily

Choosing Hash Functions

- Random Oracle (or as close as possible)
- Generally designed to handle keys that are 32 or 64 bit integers or a short vector of nonnegative integers

Static Hashing

- Single, fixed hash function
- Not recommended any more
- Division Method: $h(k) = k \bmod m$
- Multiplication method: $h(k) = \lfloor m(kA \bmod 1) \rfloor$

Random Hashing

- Multiple hash functions, randomly select the hash function at the start of each execution
- Universal family based on number theory:
 - Choose a prime p such that every possible key is less than p (then $p \geq m$)
 - let a be in $\{0, \dots, p-1\}$ and b be in $\{1, \dots, p-1\}$
 - Define $h_{ab}(k) = ((ak + b) \bmod p) \bmod m$
 - $p(p-1)$ hash functions

Practicing with Hash Functions

For the following hash functions and keys, apply the hash functions, resolve collisions using chaining, and think about why they may not be useful in practice

- $h(k) = k \bmod 9$, keys=5,28,19,15,20,33,12,17,10
- hashing a string as the average of the ascii values, keys="a", "z", "az", "by", "din"

For the following keys, Illustrate the result of inserting the keys using linear probing $h(k, i) = (k + i) \bmod m$ and double hashing with $h_1(k) = k$ and $h_2(k) = 1 + (k \bmod (m - 1))$

- $m=11$, keys=10,22,31,4,15,28,17,88,59