EECS 2011 M: Fundamentals of Data Structures

Suprakash Datta
Office: LAS 3043

Course page: `http://www.eecs.yorku.ca/course/2011M`
Also on Moodle

Hash Tables

Ch. 9.1, 9.2

Applications

- Dictionaries
- Set membership queries
- Union, intersection of sets

Note: Some slides in this presentation have been adapted from the authors' slides.

Dictionaries

set S of elements, each with an associated *key*, *value* pairs.

Complexity of lookup in a set of size n:

- Unsorted list: $\Omega(n)$ worst-case time
- Sorted list: $\Omega(\log n)$ worst-case time
- Search trees (later topic): $\Omega(\log n)$ worst-case time

Q: (how) can we do better?

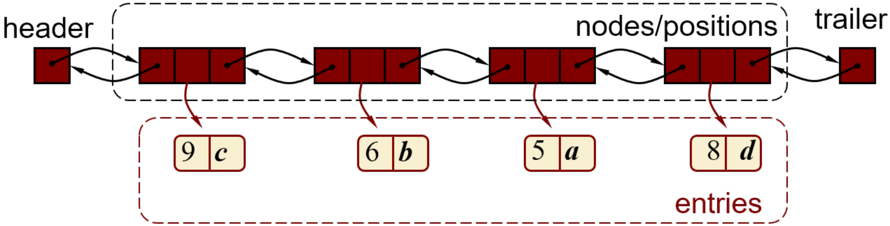
The Map ADT

A map M is a searchable collection of key-value entries

- Main operations: search, insert, and delete items
- Multiple entries with the same key are not allowed
- ADT:
  - $get(k)$: return value of entry with key k, or null
  - $put(k, v)$: insert entry (k, v); return old value associated with k, or null
  - $remove(k)$: remove entry with key k, return its value/null
  - $size()$, $isEmpty()$
  - $entrySet()$: returns an iterable collection of entries in M
  - $keySet()$: return an iterable collection of the keys in M
  - $values()$: return an iterable collection of the values in M

List-based Maps

Doubly linked list implementation:



List-based Maps

Performance:

- put, get and remove take $\theta(n)$ time since in the worst case (the item is not found) we traverse the entire sequence to look for an item with the given key
- The unsorted list implementation is effective only for small maps

Direct Access Tables

- Define a one-to-one function from keys to natural numbers
- Allocate an array large enough to have entire key space as indices
- Ignoring space and array initialization overhead, add/delete/find are all $\theta(1)$ operations

Q: Can we get "similar" times for smaller arrays?

Hash Tables

- A hash table is a data structure that can be used to make map operations faster. While worst-case is still $\theta(n)$, average case is typically $\theta(1)$
- A hash table for a given key type consists of
  - Hash function h
  - Array (called table) of size N
- the goal is to store item (k, v) at index $i = h(k)$ of the hash table where $h()$ is a hash function whose input is the key of the element and output is a location in the hash table

Hash Functions

- A hash function h maps keys of a given type to integers in a fixed interval $[0, N-1]$,
  E.g., $h(k) = k \mod N$ is a hash function for integer keys. The size N is usually chosen to be a prime.
- Hash functions are many-to-one, there can be *collisions*
- Since different keys map to the same location, need to resolve collisions carefully

Hash Functions - examples

- $h2(y) = y \mod N$
- Multiply, Add and Divide (MAD): $h2(y) = (ay + b) \mod N$, a, b non-negative integers, and $a \mod N \neq 0$
- Integer cast: We reinterpret the bits of the key as an integer
- Component sum: partition the bits of the key into components of fixed length (e.g., 16 or 32 bits) and sum the components (ignoring overflows)

Polynomial Hash Functions

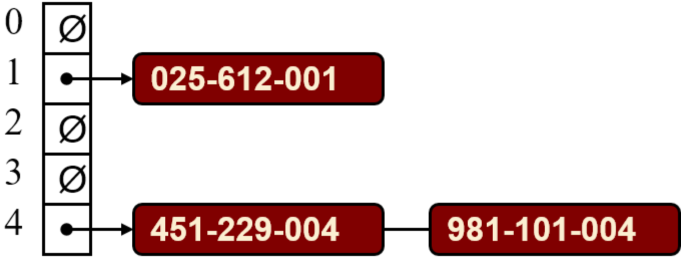
- Partition the key into fixed length (e.g., 8, 16 or 32 bits) chunks $a_0, a_1, \ldots, a_{n-1}$. Fix a constant $z > 1$.
- Evaluate (ignoring overflows)
  $p(z) = a_0 + a_1 z + a_2 z^2 + \ldots + a_{n-1} z^{n-1}$
- Especially suitable for strings
- Horner's rule for computing $p(z)$ in $\theta(n)$ time
  1. $p_0(z) = a_{n-1}$
  2. $p_i(z) = a_{n-i-1} + z p_{i-1}(z) (i = 1, 2, \ldots, n-1)$
  3. $p(z) = p_{n-1}(z)$

Clustering Behaviour of Hash Functions

- Desirable in some applications: like searching with approximate keys.
- Undesirable in some scenarios: e.g. cryptographic has functions

Handling Collisions - Techniques

- Chaining



- Open addressing with Linear Probing
- Open addressing with Double Hashing

Chaining

- A new data structure at each location
- Often a linked list

For linked list based chaining:

- Worst case time for insert: $\theta(1)$
- Worst case time for delete, search: $\Omega(n)$
- In practice, the array size can be chosen to tradeoff array initialization costs with search/delete costs

Open Addressing with Linear Probing

the colliding item is placed in another cell of the table

- Linear Probing: colliding item in the next (circularly) available table cell
- Each table cell inspected is referred to as a “probe”
- Colliding items lump together, so that future collisions cause a longer sequence of probes

Linear Probing ($h(k) = k \mod 7$)

0	1	2	3	4	5	6	
	50						Insert 50
	50				47		Insert 47
	50	43			47		Insert 43
	50	43	31		47		Insert 31
	50	43	31	36	47		Insert 36
	50	43	31	36	47	20	Insert 20
27	50	43	31	36	47	20	Insert 27

Linear Probing - Details

get(k)

- start at cell $h(k)$
- probe consecutive locations until
    - An item with key k is found, or
    - An empty cell is found, or N cells have been unsuccessfully probed

Insertions and deletions: problem

0	1	2	3	4	5	6
27	50	43	31	36	47	20

Delete 50

27		43	31	36	47	20

Delete 43

Fails?

Solution: introduce a special object, called DEFUNCT, to replace deleted elements

Linear Probing: Add and Remove

remove(k)

- search for an entry with key k
- If such an entry (k, o) is found, replace it with the special item DEFUNCT and return o
- Else, return null

put(k, o)

- throw an exception if the table is full
- start at cell h(k)
- probe consecutive cells until
  - A cell i is found that is either empty or stores DEFUNCT, or
  - N cells have been unsuccessfully probed
- We store (k, o) in cell i

Open addressing with Double Hashing

Suppose that $i = h(k)$ leads to a collision.

- uses a secondary hash function $h'(k)$ in addition to the primary hash function $h(x)$
- iteratively probe the locations $(i + jh'(k)) \mod N$ for $j = 0, 1, \ldots, N-1$
- $h'(k)$ cannot have zero values
- Choose N to be prime. Common choices for $h'(k)$:
  - $h'(k) = q - k \mod q$, where
  - $q < N$ and q is prime
- The possible values for $h'(k)$ are $1, 2, \ldots, q$

Performance of Hashing Schemes

- In the worst case, searches, insertions and removals on a hash table take $\Omega(n)$ time
- worst case: all keys collide
- The load factor $\lambda = n/N$ affects the performance of a hash table
  - chaining: performance is typically good for $\lambda < 0.9$
  - open addressing: performance is usually good for $\lambda < 0.5$
  - java.util.HashMap maintains $\lambda < 0.75$
- Open addressing can be more memory efficient than separate chaining
- However, chaining is typically as fast or faster than open addressing.

Application of Hashing

- small databases
- compilers
- browser caches
- Finding Set Intersection
  - Complexity: Average case $\theta(m + n)$
  - Alternative: Merge sort modification

Rehashing

When the load factor λ exceeds threshold, the table must be rehashed.

- A larger table is allocated (typically at least double the size)
- A new hash function is defined.
- All existing entries are copied to this new table using the new hash function.

Cryptographic Hash Functions

- Used for digital signatures. A digital signature must
  - depend on the content being signed, and
  - be short
- Must be hard to forge

Demo at `www.hashemall.com`