Hashing

Class 19

Dictionary

- dictionary is the name for a container with the following characteristics
  - a key which identifies an entry
  - a value corresponding with the key consisting of one or more fields of data
  - i.e., it contains key-value pairs
- and with the following behaviors
  - find a specific key in the dictionary
    - Boolean find
    - retrieve the value associated with a particular key
  - insert a new key and its value into the dictionary
    - if the key exists, overwrite the value associated with it
  - remove a key and its associated value from the dictionary
- is_empty, etc

Implementation

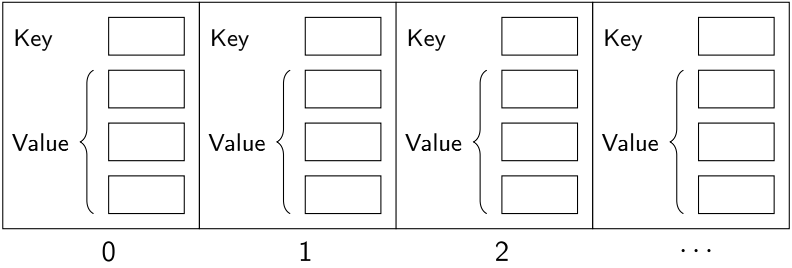
- a dictionary can be implemented in many different ways, e.g.,
  - an array of key-value pairs maintained in key order
  - a binary search tree of key-value pairs ordered by key
- but usually a dictionary implies a hashed implementation

Hashing

- there are a number of variations of hashing
  - static vs dynamic
  - open chaining vs various probing strategies
- we could spend weeks on all the variations
- we will only consider static open chaining

Hash Table

- a hash table is an array (vector) of structures
- the array size m is fixed (in static hashing)
- one field is the key
- the other field(s) are collectively the value
- there are n possible keys



Hash Function

- a hash function is used to decide where to place a key-value pair in the array
- a hash function is a function:

$$f(\text{key}) = h$$

  where

$$h \in \{0, 1, \ldots, m-1\}$$

- there are many hash functions

Hash Functions

if the key is an unsigned integer, a possible hash function is

$$f(\text{key}) = \text{key mod } 33$$

here, $m = 33$, and every possible unsigned integer is mapped to an index in the range $0 \ldots 32$

if the key is an ASCII string, a possible hash function is

$$f(\text{key}) = \text{ord}(\text{toupper}(\text{key.at}(0))) - \text{0x41}$$

here, $m = 26$, and every possible alphabetic string is mapped to an index in the range $0 \ldots 25$

Hash Functions

- literally thousands of hash functions have been invented
- designing them is an art form
- the primary desiderata of the hash function are:
  - it must distribute the keys as evenly as possible among all m indices
  - it must be fast to compute
- the hash function is tightly coupled to the size m of the hash table

Analysis

- assuming
  - a perfect hash function
  - correctly sized hash table
  - well-behaved data
- push: $T(n) \in \Theta(1)$
- pop: $T(n) \in \Theta(1)$
- find: $T(n) \in \Theta(1)$

Problem

- alas, the world is not perfect
- what if the hash table size m is smaller than the number of keys n?

Problem

- alas, the world is not perfect
- what if the hash table size m is smaller than the number of keys n?

- by the pigeonhole principle at least two keys will be mapped to the same index
- this is a collision (aka hash clash)

- collisions can happen even if m is larger than n (how?)
- thus, we must handle collisions

Collision Example

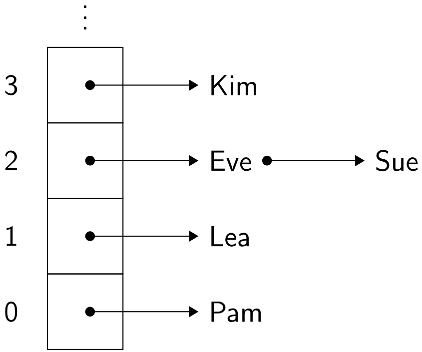
- keys are ASCII strings
- hash table size is 13
- hash function is sum(ord(characters)) mod table_size

Name	Function	Index
Eve	$(69 + 118 + 101) \bmod 13$	2
Lea	$(76 + 101 + 97) \bmod 13$	1
Kim	$(75 + 105 + 109) \bmod 13$	3
Pam	$(80 + 97 + 109) \bmod 13$	0
Sue	$(83 + 117 + 101) \bmod 13$	2

- Eve and Sue collide
- both map to the same index

Open Chaining

- one strategy to handle collisions is open chaining
- this is the only strategy we'll consider
- in open chaining, the hash table does not actually store entries
- rather, it stores linked lists of entries



Open Chaining

- find
  1. compute hash of key
  2. go to the linked list of that index
  3. search linearly for key in the list
- push (both insert and replace)
  1. compute hash of key
  2. go to the linked list of that index
  3. search linearly for key in the list
  4. insert or update entry
- pop
  1. compute hash of key
  2. go to the linked list of that index
  3. search linearly for key in the list; delete entry if found

Open Chaining

- pros
  - can tolerate an unlimited number of collisions
- cons
  - requires the machinery of linked list
  - list could be quite long — how long?

Open Chaining

- pros
  - can tolerate an unlimited number of collisions
- cons
  - requires the machinery of linked list
  - list could be quite long — how long?
  - let us assume a good hash function that evenly distributes keys
  - but some collisions occur
- now what is analysis?
- insertion:
- deletion:
- search:

Open Chaining

- pros
  - can tolerate an unlimited number of collisions
- cons
  - requires the machinery of linked list
  - list could be quite long — how long?
  - let us assume a good hash function that evenly distributes keys
  - but some collisions occur
- now what is analysis?
- insertion: $T(n) \in O(n/m)$
- deletion: $T(n) \in O(n/m)$
- search: $T(n) \in O(n/m)$

- the ratio $\alpha = n/m$ is called the load factor

Load Factor

- the load factor should be close to 1
- what if it's too small (m is big compared to n)?
  lots of unused wasted space in the hash table
- what if it's too large ($m \ll n$)?
  lots of long chains: degraded performance

Rehashing

- if the table gets too full
- 70% of the entries are occupied
- running time for operations increases, performance decreases
- solution: allocate a new table of size next prime number more than twice as big as the current table
- create a new hash function appropriate for the new size
- re-hash every element in the old table to the new table
- de-allocate old table

Hash Table Size

- truism: always use hash table whose size is a prime number
- why?
- if your keys are evenly distributed, so that every possible key is equally likely, then it makes no difference
- prime or non-prime size is irrelevant
- however, in real life many key sets are not evenly distributed

- imagine you are keeping a symbol table of objects based on where they are stored in memory — their memory address
- your computer's word size is 4 bytes
- every address is a multiple of 4
- if m happens to also be a multiple of 4, $\frac{3m}{4}$ table entries are empty, and all n entries collide in the remaining $m/4$ slots

Prime Table Size

- every key that has a common factor with table size m will be hashed to a location that is a multiple of the common factor
- to minimize collisions we must reduce the number of common factors between m and the set of keys
- how? choose m to be a number that has very few factors
- what kind of number has very few factors?

Prime Table Size

- every key that has a common factor with table size m will be hashed to a location that is a multiple of the common factor
- to minimize collisions we must reduce the number of common factors between m and the set of keys
- how? choose m to be a number that has very few factors
- what kind of number has very few factors? a prime number!

Java Hashcode

- the Java Object class has the method int hashCode()
- you can generate a hash code for any object
- when hashCode() is invoked on the same object more than once during an execution of a Java application, the hashCode method must return the same integer
- if two objects are equal according to the equals() method, then hashCode() on each of the two objects must produce the same integer result
- it is not required that if two objects are unequal according to the equals() method, then calling the hashCode method on each of the two objects must produce distinct integer results — WHY?

Java String Hashcode

- Java uses this as its string hash function

$$h(s) = \sum_{i=0}^{n-1} s[i] \times 31^{n-1-i}$$

where the sum is 32-bit addition and $s[i]$ is the utf-16 character code of the ith character of the string

- what are the keys?
- how many indices can this generate?

Hash Table Implementations

- Java HashMap, uses hashCode() to place items
- C++ map
- PHP every array is actually a hash table with keys limited to integers and strings
- Python dictionary
- Perl hash
- JavaScript all object fields are associative arrays

Array vs HashMap

- what is stored
- memory used
- ordering of elements
- duplicate elements
- access strategy

Article

https://www.techrepublic.com/article/programming-languages-facebook-open-sources-its-fast-f14-hash-table-written-in-c/