COL106 - Data Structures and Algorithms

Map ADT

Hashtables

The Map ADT (Dictionary / Associative array)

- Model a "searchable" collection of key-value entries
- Operations: Searching, deleting and inserting items.
- Multiple entries with same key are not allowed.
- Applications:
  - Student record database — key = entry number, value = student name
  - compiler
  - web: key = URL, value = webpage content
- Methods: size(), isEmpty(), get(k), put(k, v), remove(k), keySet(), values(), entrySet()

Only require comparisons for equality (no order required)

Implementing Map ADT

java.util.Map
java.util.Dictionary

- Arrays, LinkedList (inefficient)
- Unordered sequence:
  - get(k), remove(k) takes O(n) time
  - put(k,v) takes O(1) time
  - Could be useful if there are not too many get(), remove() ops required.
    (e.g. log files in a computer)
- Ordered sequence (say array):
  - get(k) takes O(log n) time, put(k,v) & remove(k) take O(n) time
  - Good if all you need is search.

Hashtables

Direct addressing: Array indexed by key ⇒ takes $O(1)$ time for all operations, but $O(r)$ space.
- e.g: COL106 registry.
  - ↳ r is range of keys.

Hash Table: – $O(1)$ expected time
- ① Hash function
- ② Array (table) – $O(n+m)$ space where m is table size, n is number of entries.

Store item (k, o) at index $i = h(k)$

Example: Let keys be entry numbers of COL106 students.

2022 PH 10140 ⟶ 140

Hash function: take last 3 digits: "Collision"

How to deal with alphanumeric keys?

Birthday Paradox

What is the probability that there are 2 people in this room who share a birthday?

340

365

Birthday Paradox

What is the probability that there are 2 people in this room who share a birthday?

$P(B)$ = Prob. that at least 2 people share a birthday

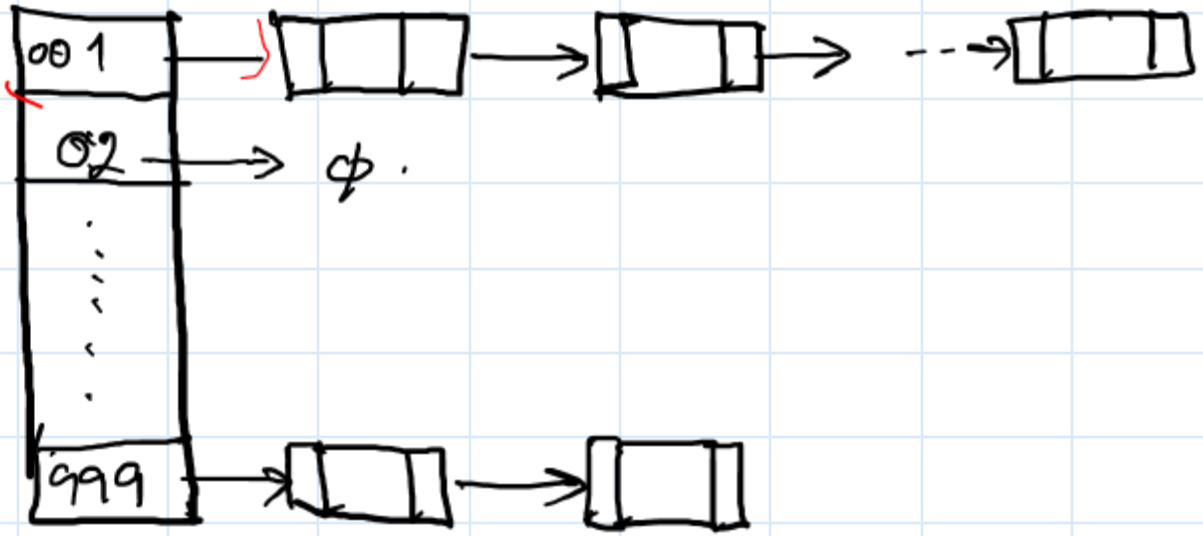
$$= 1 - \left(\frac{365}{365} \times \frac{364}{365} \times \cdots \cdot \times \frac{364 - 340}{365} \right)$$

$$= 1 - 1.07 \times 10^{-118} \approx 1$$

No matter what hash function you use, there will be collisions

Collision Resolution

- What if I have two keys hashing to same location?
- **Chaining:** Have an array of links indexed by keys, having list of items with same key.



To find/insert/delete an element, lookup position in table. & search/insert/delete the element in the linked list of the hashed slot

$h: U \rightarrow \{0, \ldots m-1\}$ → m = size of hash table.

Analysis of Hashing

$h(k) = [\text{rand}(k)] \to m$

- Element with key k stored in slot $h(k)$.

$$h: U \longrightarrow \{0, 1, \ldots m-1\}$$

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

- Assume time taken to compute $h(k)$ is $\Theta(1)$.
- What is a good hash function?
  - distributes keys "evenly" among slots, easy to compute, less space.
  - ideally slot is picked uniformly at random & hash
    - ↳ not actually
    - we need to know where a key got mapped to!
    - ↳ also not easy to store
  - Simple uniform function (assumption)
  - Load factor: $\alpha = \frac{n}{m}$ → # elements, → size of table

$m = O(n)$

Analysis of Hashing

$h: U \to \{0, 1, \ldots, m-1\}$

$h(k) \to$

- Simple Uniform hashing leads to a list of length $\alpha = \frac{n}{m}$ on average.

  $\Rightarrow$ Expected search time = $O(1+\alpha)$ → worst case unsuccessful search

- Efficiency relies on choice of hash function!

What about expected search time for a succesful search?

$h(k) = i$ $\quad$ $t[i]$.

$O(1+\frac{\alpha}{2})$

Analysis of Hashing

- Simple Uniform hashing leads to a list of length $\alpha = \frac{n}{m}$ on average.

$\Rightarrow$ Expected search time = $O(1+\alpha)$ → worst case unsuccessful search

- Efficiency relies on choice of hash function!

What about expected search time for a succesful search?

- You computed a hash & found a hit. How many elements did you have to look at?

$O(1+\frac{\alpha}{2})$

Obs: Expected length of list when i-th element was inserted = $\frac{i}{m}$

a b c a+bz+cz² c a b c+az+bz² z

Hash functions = Hash code + Compression map

key → integer integer → [0, m-1]

Hash Code: (2021 CS 1101)

SC

- Integer cast (interpret bits of key as integer)
- Component sum (partition bits of key into components & add up)
- Memory address (interpret memory address as integer)
- Polynomial accumulation (partition key to components of fixed length & evaluate polynomial) → $a_0, \ldots a_{n-1}$

$$p(z) = a_0 + a_1 z + \cdots + a_{n-1} z^{n-1}$$

numbers

Horner's rule: $p(z) = a_0 + z(a_1 + z(a_2 + \cdots))$

$P_0(z) \leftarrow a_{n-1}$, $P_i(z) \leftarrow a_{n-i-1} + z P_{i-1}(z)$

$z = 33$ gives ≤ 6 collisions on a set of 50,000 English words

Compression Maps

1. $h(k) = k \pmod m$ (simple division)

choosing $m = b^e$ (bad!) 2^e

Typically m is chosen to be prime - gives a uniform distribution.

not close to a power of 2.

Compression Maps

(1) $h(k) = k \pmod m$ (simple division)

choosing $m = b^e$ (bad!)

Typically m is chosen to be prime - gives a uniform distribution.

not close to a power of 2.

(2) $h(k) = \lfloor m (kA \bmod 1) \rfloor$ where $0 < A < 1$

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

Eg: $A = \frac{\sqrt{5} - 1}{2}$

(Fibonacci hashing)

(Knuth TAOCP vol.2)

Compression maps

(3) "MAD" (Multiply, Add & Divide)

$h(k) = |ak + b| \bmod N$

a not a multiple of N.

(often used in pseudo random number generators)

(4) Universal Hashing: $\mathcal{H} = \{h : U \to \{0, \ldots, m-1\}\}$

$\mathcal{H}$ is universal if for any randomly chosen h from $\mathcal{H}$, & keys k_x, k_y

$$Pr[h(k_x) = h(k_y)] \leq 1/m$$

Handling Collisions

$h : U \to \{0, 1, \ldots, m\}$

Open addressing: In chaining all elements were stored outside the hash table.

$A[i]$

- Put all the elements in the table $\Rightarrow$ $n \leq m$
- Have a systematic way to probe elements of the table.
- Modify hash function:

$h : U \times \{0, 1, \ldots, m-1\} \to \{0, 1, \ldots, m-1\}$

specify the probe number

- Probe sequence:

$\langle h(k,0), h(k,1), \ldots, h(k,m-1) \rangle$.

- h gives a sequence of slots examined for a given key.