Hashing

Tufts University

Cheat of the day

- For $0 < a < b$ it holds that $\frac{a-1}{b-1} < \frac{a}{b}$
- For $0 < c < 1$ it holds that $\sum_i c^i = \frac{1}{1-c}$

Container data structures

Dictionary data structure:

INSERT(key, additional information) Adds pair to DS

SEARCH(key) Returns the additional information associated to key

DELETE(key) Remove pair (key,info) from DS

Applications? a TON

Search files Find all copies of file ...

Webmail Given user/password hash and return data

Verification Sending messages that others verify

Dictionaries Where did I store my keys?

Detecting plagiarism Find common substrings in long texts

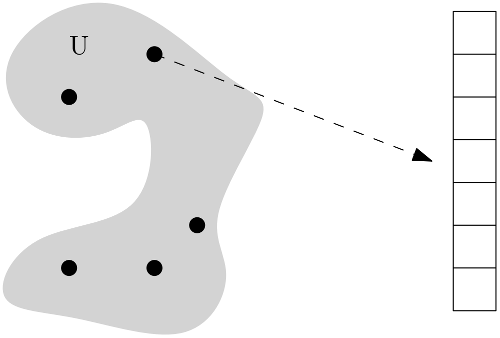
Comparison to other data structures

	Array (unsorted)	Array (sorted)	Linked list	AVL	Hash
Search	$\Theta(n)$	$\Theta(\log n)$	$\Theta(n)$	$\Theta(\log n)$	$\Theta(1)$
Insert	$\Theta(1)$	$\Theta(n)$	$\Theta(1)$	$\Theta(\log n)$	$\Theta(1)$
Delete	$\Theta(1)$	$\Theta(n)$	$\Theta(1)$	$\Theta(\log n)$	$\Theta(1)$

(Delete counts time to remove after search has been executed)

Today's goal: let's prove $\Theta(1)$ bounds for hashing

Hashing: review



Assumptions:

- User inserts n elements in hash table

  n known in advance

- Each key lies in a *universe* U (possibly infinite set)
- We create a table of m *buckets*
- We assume a hash function $h : U \to [0, \dots m]$
- We insert/search/delete (k, i) in $h(k)$-th bucket

Collisions: chaining

0	$\rightarrow$ NIL
1	$\rightarrow Obj_5 \rightarrow Obj_3$
2	$\rightarrow$ NIL
3	$\rightarrow Obj_1$
4	$\rightarrow Obj_2 \rightarrow Obj_4$
5	$\rightarrow$ NIL
6	$\rightarrow$ NIL

Chaining Keep all elements in a linked list

- New element is inserted at front
- When searching we traverse list
- Key goal: chains of similar length

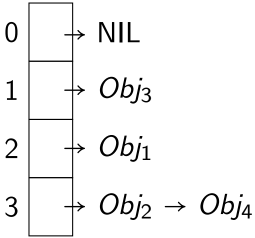
Collisions: open addressing

Obj_3
Obj_1
Obj_2
Obj_4

Open addressing If occupied try a different position

- $h(\text{key})$ = permutation of m
  i.e., $h(\text{key}_5) = 4, 6, 2, 5, 1, 3, 0$
- Search in that order for insertion/deletions
- Beware of deleted elements
- Key goal: have table mostly empty

Summary



When using a hash table you must define:

- What is the key, what extra information we store
- Number of buckets m

  $m = .75n$ is default for Java
  $m = \Theta(n)$ is good
- How to resolve collisions (open addressing or chaining)

  Beware! If open addressing we need $m \geq n$
  For Comp 160 it rarely matters

On the hash function h

- h must be easy to compute

  Assume h runs in $\Theta(1)$ time
- h must be deterministic

  $h(i)$ should return always the same
- For open addressing h should generate a permutation

  $h(\text{key}, i)$ returns i-th element of permutation

  First try $h(\text{key}, 0)$, then $h(\text{key}, 1)$, and so on
- Worst-case analysis of hashing is terrible

  All keys land in same bucket $\Rightarrow \Theta(n)$ runtime
- h should behave like a random function

  Simple uniform hashing (SUH for short)
  - Each key is equally likely to be hashed to any slot
  - Independent of past or future insertions

Runtime analysis for **Chaining**

Lemma

Under SUH, the expected number of elements in a bucket is n/m

Proof.

- $X_{ij}=1 \Leftrightarrow i$-th key lands in j-th bucket
- $E[X_{ij}] = 1/m$
- Y_j = number of elements in j-th bucket= $\sum_{i=1}^{n} X_{ij}$
- $E[Y_j] = E[\sum_{i=1}^{n} X_{ij}] = n/m = \alpha$ (**load factor**)

$\square$

Corollary Runtime of operations becomes:

Insertion $\Theta(1)$ (hash and insert at front of table)

Successful search $\Theta(1 + \alpha/2)$ (we expect to visit half the elements)

Unsuccessful search $\Theta(1 + \alpha)$ (must traverse whole list)

Delete $\Theta(1 + \alpha/2)$ (successful search $+$ $\Theta(1)$ to delete)

Runtime analysis for **Open Addressing**

- SUH is not enough for open addressing
- **Uniform Hashing**: $h(\text{key})$ will return any of the $m!$ permutations with equal probability
- Informally: SUH for $h(\text{key},0)$, $h(\text{key},1)$, $h(\text{key},2)$, and so on

Lemma

Under UH, the expected #probes in an unsuccessful search is $(1-\alpha)^{-1}$

Proof.

- $X_i = 1 \Leftrightarrow$ We look at i-position in probe sequence.
- X_0 is always $1 \Rightarrow E[X_0] = 1$
- $E[X_1] = \frac{n}{m}$ (m buckets, only n of them occupied)
- $E[X_i] = 1 \cdot \frac{n}{m} \cdot \frac{n-1}{m-1} \cdot \ldots \cdot \frac{n-(i-1)}{m-(i-1)} < \alpha^i$ (previous buckets occupied)
- $X =$ number of buckets we check
- $E[X] = E[\sum_{i=0}^{n} X_i] = \sum_{i=0}^{n} \alpha^i < \frac{1}{1-\alpha}$

Runtime analysis for **Open Addressing**

Lemma

Under UH, the expected #probes in an unsuccessful search is $(1-\alpha)^{-1}$

Corollary Runtime of operations become:

Insertion $\Theta((1-\alpha)^{-1})$ (must find an empty spot)

Unsuccessful search $\Theta((1-\alpha)^{-1})$

Successful search $O((1-\alpha)^{-1})$ (better than unsuccessful)

Delete $O((1-\alpha)^{-1})$ (successful search + $\Theta(1)$ to remove)

Examples of hash functions

Multiplication Fix a. Then $h(k) = ak \bmod m$ and keep middle bits

- Fast to compute
- Works well in practice
- May have problems if codependencies in string
- a should not be a power of 2

Universal Hashing Fix large prime p randomly choose $a, b < p$.
Then $h(k) = (ak + b \bmod p) \bmod m$

- For worst case input, $P[h(k_i) = h(k_j)] = 1/m$
- No need for SUH assumption!

Probe Sequences

- Linear probing: $h(k, i) = (h(k, 0) + i) \bmod m$
- Quadratic probing: $h(k, i) = (h(k, 0) + c \cdot i + d \cdot i^2) \bmod m$
- Double hashing: $h(k, i) = (h_1(k) + i \cdot h_2(k)) \bmod m$

Additional Practice questions

- Give pseudocode for the following operations:
  - Insertion
  - Search
  - Deletion
- Answer the previous question for both collision strategies
- What is the runtime of executing n insertions in:
  - hash table with m buckets
  - collisions handled with chaining
  - Can I choose a value of m so that overall time is $\Theta(\sqrt{n})$?
- Say I have a hash table with n students
  - key is Tufts id, extra info is their grade in course
  - How fast can I find the student with highest grade?
  - How fast can I find the student with highest Tufts id?
- How much space is needed in a hash table?
  - Assume m buckets, n elements
  - How much space if we use chaining?
  - Would answer change for open addressing?

Lots of technicalities!

Just an FYI, beyond scope of 160:

- SUH extremely unlikely to achieve
  - We can do simulate reasonably
  - Given a sequence of n hash values, hard to predict $n+1$-th
- UH impossible
  - Hard to even generate proper permutations!
  - Double hashing works well in practice
- Slightly different definition of expected runtime
  - No random choices to average
- Unfair to compare collision strategies with same bucket size
  - See recitation!
- What if n is not known in advance?
  - Must resize table if α becomes too big
  - Need to talk about *amortized* runtime