Sets, Dictionaries & Hash Tables

CS16: Introduction to Data Structures & Algorithms
Seny Kamara - Spring 2018
Outline

- Sets
- Dictionaries
- Hash Tables
- Ex: Jumble
Sets

- Collection of elements that are
  - distinct
  - unordered (unlike lists or arrays)
Set ADT

- **add**(object):
  - adds object to set if not there
- **remove**(object):
  - removes object from set if there
- **boolean contains**(object):
  - checks if object is in set
- **int size()**:
  - returns number of objects in set
- **boolean isEmpty()**:
  - returns TRUE if set is empty; FALSE otherwise
- **list enumerate()**:
  - returns list of objects in set (in arbitrary order)
Set Data Structure

- How can we implement a Set?
- Expandable array
  - add (to end): $O(1)$
  - contains (scan): $O(n)$
  - remove (find & compress): $O(n)$
- Can we do better?
Dictionary

- Collection of key/value pairs
  - distinct keys
  - unordered
- Supports value lookup by key
- AKA a **map**
  - maps keys to values
- ex: name → address; word → definition
Dictionary ADT

- add(key, value):
  - adds key/value pair to dict.

- object get(key):
  - returns value mapped to key

- remove(key):
  - removes key/value pair

- int size():
  - returns number key/value pairs

- boolean isEmpty():
  - returns TRUE if dict. is empty; FALSE otherwise
Dictionary Data Structure

‣ How can we implement a Dictionary?
‣ Expandable array
  ‣ add (to end): $O(1)$
  ‣ contains (scan): $O(n)$
  ‣ remove (find & compress): $O(n)$
‣ Can we do better?
Hash Table

- Dictionary data structure
- Built with
  - array
  - hash function: function that mixes and shrinks

![Diagram of hash table with input space and output space](image-url)
Idea

- Choose hash function \( h: X \rightarrow Y \) with
  - input space \( X \): universe of keys
  - output space \( Y \): array indices
- Store `value` at location \( h(key) \) of array
- Problem
  - \( h \) can map multiple values to same index/location
  - if \( Y < X \), collisions will happen!
  - values will be overwritten
Idea

› Possible solution
  › store multiple values at each array location
  › called a bucket: list, expandable array, …
  › This solution is called Chaining

› Other possible solutions
  › linear probing, quadratic probing
  › …
Hash Table

```
function add(key, value):
    index = h(key)
    table[index].append(key, value)

function get(key):
    index = h(key)
    for (k, v) in table[index]:
        if k = key:
            return v
    error("key not found")
```

table: array
h: hash function

$O(1)$ if hash is $O(1)$
depends on bucket size
**Hash Table — Add**

**Keys:** Banner IDs

\[ h(key) = key \mod 7 \]

- B00943855
  - Kaila Jeter
- B00238494
  - Alejandro Molina
- B00472885
  - David Laidlaw
- B00231924
  - Lauren Ho

**Array of buckets w/ key/value pairs**

- B00472885
  - David Laidlaw
- B00231924
  - Lauren Ho
- B00239625
  - Sophie Saskin
- B00943855
  - Kaila Jeter
- B00238494
  - Alejandro Molina
- B00745911
  - Chantal Toupin
- B00543163
  - Surbhi Madan
What is the worst-case run time of Get?
Hash Table

- Running time of Get
  - approximately **size of largest bucket**
  - \( n \) students, table of size \( m \)
  - **if** keys get mapped randomly:
    - each bucket has size \( O(n/m) = O(n) \)

- Example:
  - **150** students, table of size **7**
  - **if** IDs get mapped randomly:
    - each bucket has size **150 / 7**
Q: Can we do better than $O(n)$?
Beating $O(n)$ — Idea #1

- **Idea:** use larger table
- Banner IDs have 8 digits so max ID is $99,999,999$
- Use table of size $100,000,000$
- With hash function $h(key) = key$
  - are there any collisions?
    - no collisions! every pair gets its own array cell
  - What is run time of Get?
    - $O(1)$
- What if we only store **150** students?
Beating O(n) — Idea #2

- **Idea:** use table of exact size
- If we know we will store 150 students
  - use table of size 150
  - with hash function \( h(\text{key}) = \text{key} \mod 150 \)
  - works if keys/IDs are completely random
- What if keys/IDs are not random?
  - what if next year all banner IDs are multiples of 150?
  - all IDs would map to cell 0
Since keys are not necessarily random, we make the hash function random
Banner ID Hashing

Form groups of 10

Activity #1

5 min
Banner ID Hashing

Activity #1
Banner ID Hashing

Activity #1

4 min
Banner ID Hashing

Activity #1

3 min
Banner ID Hashing

Activity #1

2 min
Banner ID Hashing

Activity #1

1 min
Banner ID Hashing

Activity #1
Universal Hash Function

- Setup:
  - choose **prime** $p$ larger than expected capacity
  - choose 4 numbers $a_1, a_2, a_3, a_4$ **at random** in $[0, p-1]
  - Hash(key):
    - break key into 4 parts
      - $k_1, k_2, k_3, k_4$
      - $h(key) = \sum_{i=1}^{4} a_i \cdot k_i \mod 151$

- Setup:
  - $p=151$
  - $a_1=12, a_2=43, a_3=105, a_4=83$
  - Hash(B00238918)
    - $k_1=00, k_2=23, k_3=89, k_4=18$
    - $h(B00238918) = 50$
Hash Table

- Hash table + universal hash function
  - Get is $O(1)$ expected time

What is expected time?

- remember: UHF setup picks $a_1, a_2, a_3, a_4$, at random
- for some values $h$ will do well (i.e., keys are spread)
- for others it might not (i.e., keys are clustered)
- expected time is average time over $a_1, a_2, a_3, a_4$
- measure time on all $a_1, a_2, a_3, a_4$ and take average
Hash Table

- Why does universal hashing give us $O(1)$ Gets?
  - see Chapter 1.5.2 in Dasgupta et al.
Proof of Universal Hashing
Inverses

- What is the inverse of a fraction $x/y$?
  - $y/x$ because $(x/y)(y/x) = 1$
  - inverse is whatever we need to multiply it by to get 1

- What is the inverse of an int $x$ (not 1)?
  - $1/x$ because $(x)(1/x) = 1$

- What is the “integer” inverse of an int $x$ (not 1)?
  - there is none…
  - you can’t multiply an int w/ another int to get 1 (unless 1)
Modular Arithmetic

- If working modulo some number
  - Integers can have integer inverses!
- ex: let’s work \textbf{mod 7}
  - inverse of \(2\mod 7\) is 4 because \(2 \times 4 \mod 7 = 1\)
  - inverse of \(5\mod 7\) is 3 because \(5 \times 3 \mod 7 = 1\)
- Is this always true?
  - ex: does 2 have an inverse \textbf{mod 4}?
    - \(2 \times 0 \mod 4 = 0; 2 \times 1 \mod 4 = 2\)
    - \(2 \times 2 \mod 4 = 0; 2 \times 3 \mod 4 = 2\)
    - No!
- But it is true when we work modulo a prime number
  - mod a prime, every number except 0 has a unique inverse
Analysis

- Prime $p$ is the size of array
- $x_1, x_2, x_3, x_4$ are a banner ID in chunks
- $y_1, y_2, y_3, y_4$ are another banner ID in chunks
- If IDs are different, at least 1 of the chunks are diff
- Let’s assume (wlog) it is the last one so
  - $x_4 \neq y_4$
- What is the probability that
  - $h(x_1, x_2, x_3, x_4) = h(y_1, y_2, y_3, y_4)$
What is the probability that
\[ h(x_1, x_2, x_3, x_4) = h(y_1, y_2, y_3, y_4) \]

Step #1:
- find equivalent formulation of event
- that makes the randomness explicit
- what is the randomness here?

Step #2:
- what is probability of equivalent formulation?
Step 1: Equivalent Formulation

\[ h(x_1, x_2, x_3, x_4) = h(y_1, y_2, y_3, y_4) \]

by definition

\[ a_1 x_1 + \cdots + a_4 x_4 \equiv a_1 y_1 + \cdots + a_4 y_4 \pmod{p} \]

move things

\[ a_4 x_4 - a_4 y_4 \equiv (a_1 y_1 + a_2 y_2 + a_3 y_3) - (a_1 x_1 + a_2 x_2 + a_3 x_3) \pmod{p} \]

different

just some number; let’s call it c

\[ a_4 \cdot (x_4 - y_4) \equiv c \pmod{p} \]

\[ a_4 \equiv c \cdot (x_4 - y_4)^{-1} \pmod{p} \]
Step 2: Probability of Equiv. Formulation

- So hashes are equal when
  \[ a_4 \equiv c \cdot (x_4 - y_4)^{-1} \pmod{p} \]
- But
  - \( x_4 \) and \( y_4 \) are different so \( x_4 - y_4 \neq 0 \)
  - and \( p \) is prime
  - so \( (x_4 - y_4) \) has unique inverse mod \( p \)
  - So \( c(x_4 - y_4)^{-1} \) can only take on one value
    - therefore \( a_4 \) can only take on one value
- What is the probability \( a_4 \) takes on that value?
  - \( a_4 \) is randomly chosen from \( p \) possible values so probability is \( \frac{1}{p} \)
Putting it all Together

- Prob. that some ID will collide w/ another ID
  - $\frac{1}{p} = \frac{1}{151}$
- For some ID,
  - expected # of collisions w/ all other IDs is
    - $\frac{149}{151} = 0.986...$
- Expected size of an ID’s bucket is
  - $1 + 0.986... = 1.986... = O(1)$
End of Universal Hashing Proof
Sets from Hash Tables

- We can implement sets with a hash table
- Sometimes called a Hash Set

```python
function add(object):
    index = h(object)
    table[index].append(object)

function contains(object):
    index = h(object)
    for elt in table[index]:
        if elt == object:
            return true
    return false
```
Hash Map vs. Hash Set

- **Hash Map**
  - Hash table implementation of dictionary
  - Maps keys to values
  - No ordering

- **Hash Set**
  - Hash table implementation of set
  - No keys (like hash map with keys same as values)
  - No ordering
Example: JUMBLE

- Jumble puzzle
  - given a clue and set of letters,
  - rearrange letters into word that fits clue
- Leah is making a Jumble puzzle
  - needs words for which all permutations are invalid words
  - that way there is only 1 possible solution to puzzle
- Algorithm
  - input: set of all 5-letter english words
  - output: all 5-letter words whose permutations are non-english
JUMBLE Algorithm

- Naive approach
  - For each word,
    - For each permutation of word
      - check if permutation is an english word
  - There are 5! permutations of a word…
Better approach
- Sort each English word alphabetically
- For each English word store
  - key = sorted word and value = word
  - in hash table
- Words with no valid permutations
  - are the words in single-element buckets
function jumble(words):
    output = []
    permutations = dictionary()
    for each word in words:
        sortedKey = sort the letters of “word” alphabetically
        permList = permutations.get(sortedKey) or []  // [] if empty
        permList.append(word)
        permutations.add(sortedKey, permList)
    for each word in words:
        sortedKey = sort the letters of word alphabetically
        if permutations.get(sortedKey).length == 1:
            output.append(word)
    return output