Hashing

Introduction

First things first…

- Project 1
  - Still working on grading
  - Definitely by tomorrow

Second things second…

- Project 2
  - try targets are now up.
  - Mail about VFSystem.find()
  - Minimum Submission
    - Entry.java and Document.java
    - Due this Friday, February 6th
    - REMEMBER MINIMUM SUBMISSION RULE!!!
  - Final Submission
    - Due Sunday, February 15th

Third things third…

- Exam 2: This Wednesday
  - Topics:
    - Java I/O
    - Recursion
    - Analysis of Algorithms
    - Searching
    - Sorting
- Review Session
  - Tonight 4-6 Building 70 Auditorium

Exam Topics

- Java I/O
  - 4 Basic Classes:
    - Reader, Writer – for character data
    - InputStream, OutputStream – for byte data
    - Wrapper Classes – for high level I/O
    - Do not memorize methods…will provide Javadocs as needed.

Exam Topics

- Recursion
  - What is recursion
  - Step through a recursive function
  - Avoid this guy…

Exam Topics

- Speaking of recursion
  - Note also that you can always turn a recursive solution into a iterative solution by creating and maintaining a "state stack"
    - Which is exactly what a recursive system does under the hood.
    - …and no, this will not be on the exam!!!

Exam Topics

- Analysis of algorithms
  - Big O
  - Big Theta
    - Difference between the two
  - Calculating
    - Loop within a loop

Exam Topics

- Searching
  - Linear Search
    - Θ (n)
  - Binary Search
    - Θ (log n)

Exam Topics

- Sorting
  - Simple Sort
    - Insertion
    - Selection
    - Bubble
    - All $\Theta(n^2)$ – average case
  - Divide and Conquer Sorts
    - Merge Sort
    - Quicksort
    - Both Θ (n log n) – average case

Exam Topics

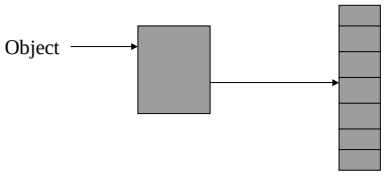
- Questions?

Searching

- Suppose are given a collection of items and we will need to see if a given object is in the collection:
  - Linear Search
    - Θ (n)
  - Binary Search
  - Binary Search Tree
    - Θ (log n)
- Can we do better?

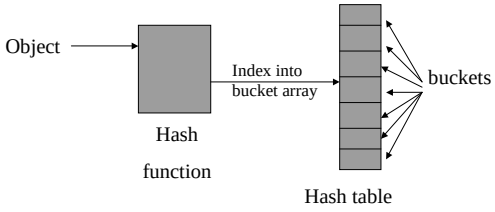
Hashing

- What if the object itself can give its location in the collection



- This is called Hashing

Hashing Terminology



About Hashing functions

- Converts object to index into bucket array.
- Goal
  - Distribute objects equally among buckets
  - Bad function
    - Add first 3 character codes of a string
  - Good function
    - Add all character codes of a string
    - Address where object is found in memory
- Should be Efficient

About Hashing functions

- Hashing rules
  - Hashing function called on same object must always return same value
  - Ideal hashing function will produce “almost random-like” values when applied on different objects.

About Hashing functions

- Ultimately, hashing function will need to fit within the bounds of an array.
  - index = (hash(O)) % n

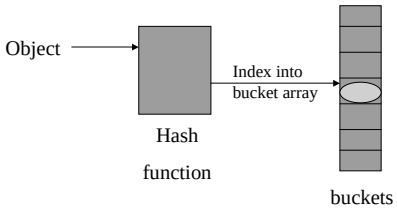
Operations on Hash tables

- Insert
  - add an object to the hash table
- Remove
  - remove an object from the hash table
- Find
  - Determine if a given object is in the hash table.

Insert

1) Apply hash function to object

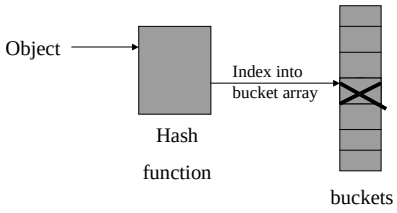
2) Add object to the index returned by the hash function



Remove

1) Apply hash function to object to see where it would be if in the hash table

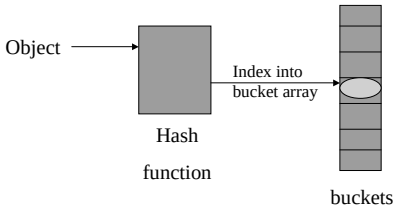
2) If item is there, remove it (and replace will a "blank object")



Find

1) Apply hash function to object to see where it would be if in the hash table

2) If item is there return true, else return false



Advantages of hashing

- Insert, Remove, Find
  - Performed in constant time
  - Time dependent only on complexity of hash function.

Collisions

- What happens if two objects hash to the same index?
  - Hash functions aren't perfect!
  - When this happens, it is called a collision.
- How do we handle collisions?

Open address hashing

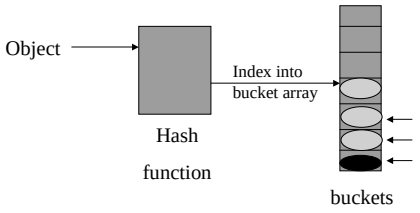
- Ways to deal with collisions
  - Open-address hashing – find another spot to put it
    - Linear Probing – go to next unfilled bucket

Linear Probing – Insert

1) Apply hash function to object to see where it should go

2) If bucket is full then, find next available bucket.

3) If no open bucket found, start again at top of hash table

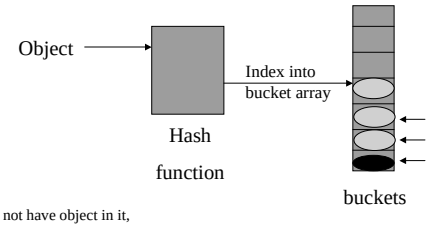


Linear Probing – Find

1) Apply hash function to object to see where it should be

2) If bucket is has item in it, return true

3) If bucket does not have object in it, but is not empty, traverse the table until either the object or an empty bucket is found

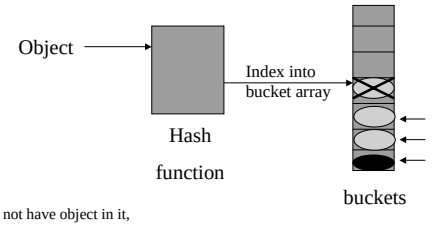


Linear Probing – Why we need the "blank" object

1) Apply hash function to object to see where it should be

2) If bucket is has item in it, return true

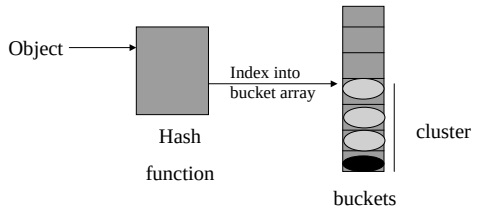
3) If bucket does not have object in it, but is not empty, traverse the table until either the object or an empty bucket is found



Linear Probing

- Clustering
  - If your hash function is less than optimal
    - Many objects hashing to the same index
    - End up with clustering
    - In the worst case
      - All objects hash to the same index
      - Must do a linear search through the hash table
      - $\Theta(n)$

Linear probing



Double Hashing

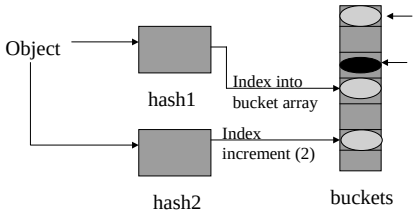
- Another way to deal with collisions
  - Open-address hashing – find another spot to put it
    - Double hashing – use a second hash function to determine how many slots forward to look

Double Hashing – Insert

1) Apply hash function to object to see where it should be

2) If bucket is full, apply a second hash function to get an increment

3) Apply increment to index until empty bucket is found

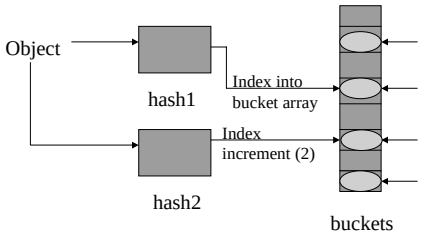


Double Hashing – Insert

1) Apply hash function to object to see where it should be

2) If bucket is full, apply a second hash function to get an increment

3) Apply increment to index until empty bucket is found



Double Hashing

- Double hashing
  - Hash function considerations
    - Must assure that increment returned by second hash function will result in all empty buckets being visited.
    - Can assure this by making the "range" of the two hashing functions to be relatively prime.
      - The two ranges have no common multiples except 1.

Double Hashing

- Double hashing
  - Hash function considerations
    - In our example, range of hash1 is 8 (size of hash table)
    - Hash2 returns a 2.
    - 8 is a multiple of 2
    - Problem!

Double Hashing

- Double hashing
  - Hash function considerations
  - Finding relatively prime numbers
    - Make the size of the hash table to be prime
    - Make the range of hash 2 to be size of hash table – 2.
      - Twin primes.
    - Note that hash2 should never return 0.

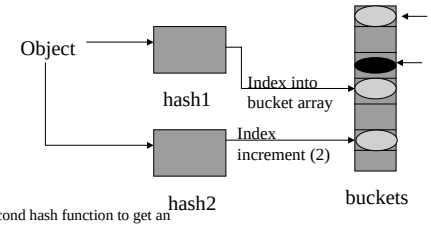
Double Hashing – Find

1) Apply hash function to object to see where it should be

2) If bucket contains object return true.

3) Else apply a second hash function to get an increment

4) Apply increment to index until object or empty bucket is found



Open-address hashing

- In case of collision
  - Find another open bucket to place your object
    - Linear Probing
      - Search for empty bucket sequentially
    - Double hashing
      - Use a second hash function to get an increment
  - Questions?

Next time

Chained Hashing

Another way to deal with collisions.