15 - Map Data Structures

Joseph Afework
CS 241

Dept. of Computer Science
California Polytechnic State University, Pomona, CA

Agenda

- Intro
- Operations
- Implementation
- Performance

Reading Assignment

- Read Chapter 28
  - Chapter 28 (Read about: **Map Data Structures**)

Maps

A **Map** is an abstract data structure (ADT)

Storage Properties:

- Data is stored in the form of a key-value pair
- Keys cannot be duplicated…. ***generally***

Representation:

- Map = {(**key**, **value**), (5,A), (7,B), (2,C)}

KEYS	VALUES
Jan	327.2
Feb	368.2
Mar	197.6
Apr	178.4
May	100.0
Jun	69.9
Jul	32.3
Aug	37.3
Sep	19.0
Oct	37.0
Nov	73.2
Dec	110.9
Annual	1551.0

Usage

- Key is used to determine where in the structure the value (object) is stored.
- Key can be seen as the address to an object

KEYS	VALUES
Jan	327.2
Feb	368.2
Mar	197.6
Apr	178.4
May	100.0
Jun	69.9
Jul	32.3
Aug	37.3
Sep	19.0
Oct	37.0
Nov	73.2
Dec	110.9
Annual	1551.0

- Maps are also called:
  - Hashmap (c)
  - Dictionary (java)
  - Associative Array (javascript, php)

Operation Interfaces

1. get(key) - Search Operation

2. put(key, value) - Insert Operation

3. remove(key) - Delete Operation

Implementation

A map can be implemented using known data structures.

- Arrays (Vector, ArrayList)
- **Linked-list**
- **Binary search trees**
- **Hash Tables**

Search

```
ArrayList list = new ArrayList()

get(key)
        For (element e in list)
                If e.key == key
                        Return e.value
        Return null
```

Runtime: O(n).... not very good…

Question: Is there a better way to do this?

Insert

```
ArrayList list = new ArrayList()

put(key, value)

    For (element e in list)

        If e.key == key

            e.value = value

            Return

    list.add(new Element(key,value))
```

Runtime: O(n).... still not very good…

Question: Is there a better way to do this?

Delete

```
ArrayList list = new ArrayList()

put(key, value)

        For (element e in list)

                If e.key == key

                        list.remove(e)

                        return
```

Runtime: O(n).... still not very good…

Question: Is there a better way to do this?

Linked List Based Map

Map (Linked List)	Worst Case
Insert	O(n)
Delete	O(n)
Search	O(n)

Balanced Binary Search Tree Based Map

Map (Balanced BST)	Worst Case
Insert	O(logn)
Delete	O(logn)
Search	O(logn)

Hash Tables

Can we do better than log(n)?..... **YES!**

- Instead of iterating over all of the data to find the object with the key..O(n).., with **hashing**, we try to find the element directly (or close by).

- **Hashing** is a process in which a key can be transformed into a key that can be used to directly address the target object.

Hashing

Hash Table/Array: an array ***A*** of size ***N***

Hash Function: a function ***h*** that will map each key to a specific index of the hash table/array

Goal: Run the key through the hash function… output will be an index that maps to a specific index in the Hash Table. Store the value at this index.

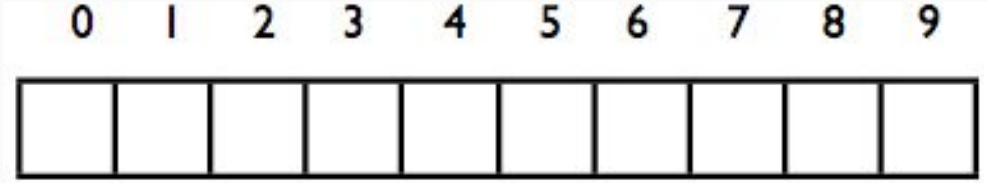
Load Factor = n/N.. n is the total number of elements in the hash table

Note: You decide the size of the table N and the hash function

ICE 15.1 Hashing

Keys: int

N = 10



h(k) = k % 10 …. k % 10 is the remainder of k/10

Add (2,A), (13,B), (15,C), (88,D), (200,E), (100,F) to the hash table….

Collisions

Two keys result in hashing to the same slot…

(200,E) and (100,F) hash to slot 0

Remember this...

Hashing Contd.

- Keys don’t need to be integers, the hashing function needs to be able to map key to an address (usually an int).
- **Memory vs Time Trade-off**
  - If you have no memory restrictions, make the hash table large… O(1) runtimes…
    - You may also adjust the hash function, and hash table size to optimize for memory constraints...
  - If you have no time restrictions, stick to a linked list, and search sequentially for keys...

Hash Function

- Pick a hash function that uniformly distributes keys into the hash table.
  - Goal is to have a 1/N chance of hashing to the same slot (Good hash function). **- Universal Hashing Property**
- Pick a N (size of hash table) large enough to distribute the keys…
  - The larger the N, the better the performance… not including memory constraints…
- Example
  - H(k) = i mod N… a prime number for N helps spread out the hashed values….

Collisions

A collision happens when two keys hash to the same storage index of the hash table…

Solution:

- Chaining - Bucket array (hash table of linked lists)
- Linear Probing

Chaining

How it works: Store all the elements in a linked list that is found at the hashed index. Iterate over all the elements, and match on the key. (Similar to linked list search) to locate the element you want.

- If the linked list is unbounded, then you can handle unlimited collisions…
  - Runtime starts to slant towards O(k) where k is the number of keys that collide to the same index.

- Uses additional space for the linked list….
  - Wasted space

Linear Probing

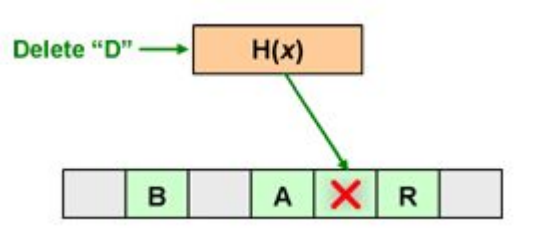
Insert: Instead of using a linked list, try to store the element in the hash table at a nearby index. i+1... i+2... in the first available empty spot.

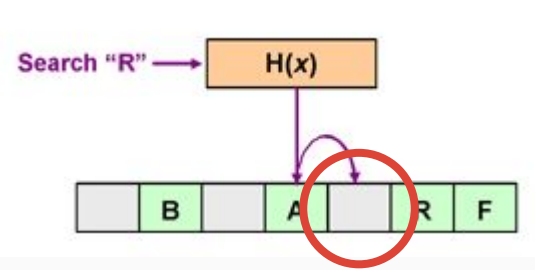
Search: Start at the index of hash(key), If the element is not found, continue checking neighbors until empty key is found, or starting index is reached again (loop around from end to start of hash table...).

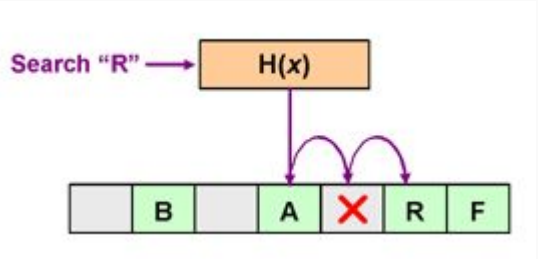
Delete: Search for the element (using Search), once found, remove the index.

Note: Delete has the potential to break Search...

Linear Probing Contd.

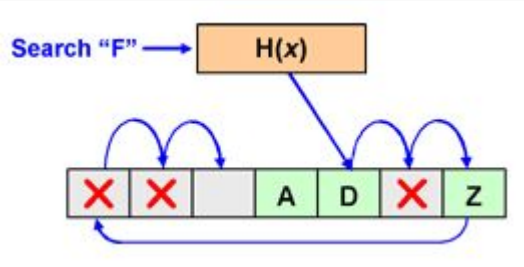


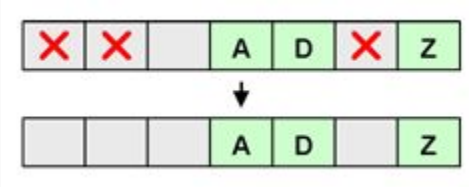


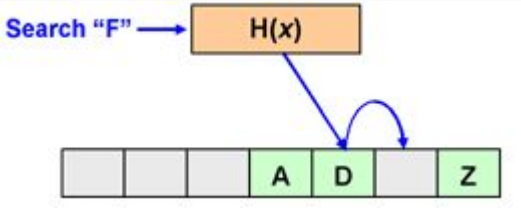


Pollution

- Too many deletion markers could result in a polluted table… a polluted table will yield O(n) search times...
- Only fix is to reconstruct the hash table….







Hash Table Based Map

Map (Hash Table)	Worst Case
Insert	O(1)
Delete	O(1)
Search	O(1)

Conclusion

- Maps are useful because of how they can be used to improve lookup operation (Search), when compared to other linear structures (*array, set*).

- Picking the right data structure for implementation could result in a runtime of O(1) for operations….

References

http://www.vogella.com/tutorials/JavaDatastructures/article.html#map

http://www.cs.rmit.edu.au/online/blackboard/chapter/05/documents/contribute/chapter/05/linear-probing.html

http://www.bowdoin.edu/~ltoma/teaching/cs210/fall09/Slides/Maps.pdf