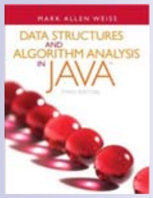
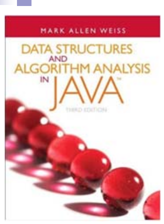


FLORIDA INTERNATIONAL UNIVERSITY

Hashing
(III)

Dr. Antonio L. Bajuelos

FIU School of Computing & Information Sciences



COP-3530 - Data Structures

Module #5: Hashing (part III)

Outline:

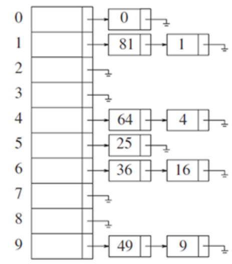
- **The collision resolution:**
  - **(II) Open addressing**
    - **(b) Quadratic probing**
      - **Secondary clustering**
      - **Expected number of probing**
      - **Complexity analysis and Java code**
    - **(c) Double hashing**
      - **General idea**
      - **Improving the key distribution**
    - **Linear vs Quadratic vs Double hashing**
    - **AVL vs Hash table**

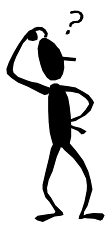
2

Remember…

Separate chaining

- **Advantages**
  - Used when memory is of concern, easily implemented.
- **Disadvantages**
  - **Parts of the table/array might never be used.**
  - As chains get longer, search time increases to **O(n)** in the worst case.





Next Question:

- Is there a way to use the "**unused**" space in the table/array instead of using chains to make more space?

3

Remember… Open Addressing

Main idea: use empty space in the table

Important points:

- All items are stored in the hash table itself.
- In addition to the cell data (if any), each cell keeps one of the three states: **EMPTY, OCCUPIED, DELETED**.
- While inserting, if a collision occurs, alternative cells are tried until an empty cell is found.
- **Probe sequence**: A probe sequence is the sequence of array indexes that is followed in searching for an empty cell during an insertion, or in searching for a key during find or delete operations.
- The most common probe sequences are of the form:

  h_i(key) = (h(key) + c(i)) mod TableSize,

  where i = 0, 1, …, TableSize-1 and c(0) = 0.

  Linear probing - linear function: c(i) = i

4

Open Addressing. Quadratic probing

- **Quadratic probing** is a collision resolution method that eliminates the **primary clustering** problem of **linear probing.**
- We can avoid **primary clustering** by changing the **probe** function:
  - **c(i) = i** by **c(i) = i²** $(\text{or } c(i) = a_1 i^2 + a_2 i + a_3)$

If **h(key)** = **key % TableSize** is <u>already occupied</u> then

- the **probe sequence** is:
  - 1^{st} probe: **(h(key) + 1^2) % TableSize**
  - 2^{nd} probe: **(h(key) + 2^2) % TableSize**
  - 3^{rd} probe: **(h(key) + 3^2) % TableSize**
  - ...
  - i^{th} probe: **(h(key) + i^2) % TableSize**

5

Open Addressing. Quadratic probing

h(key) = key % TableSize

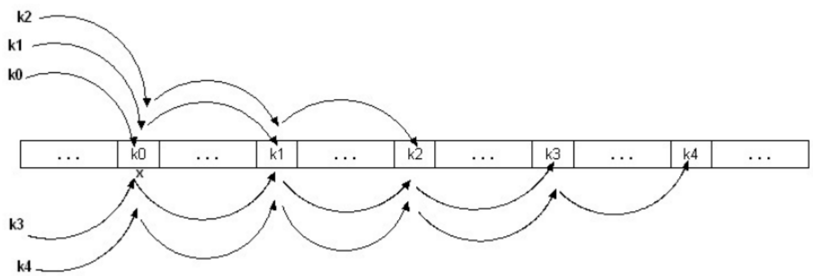
- and the **probe sequence** is:
  - 1^{st} probe: **(h(key) + 1^2) % TableSize**
  - 2^{nd} probe: **(h(key) + 2^2) % TableSize**
  - 3^{rd} probe: **(h(key) + 3^2) % TableSize**
  - ...
  - i^{th} probe: **(h(key) + i^2) % TableSize**
- **Example:** insert {89,18,49,58,69}

	Empty Table	After 89	After 18	After 49	After 58	After 69
0				49	49	49
1						
2					58	58
3						69
4						
5						
6						
7						
8			18	18	18	18
9		89	89	89	89	89

6

Open Addressing. Quadratic probing

- **Important points:**
  - **Quadratic probing** is better than **linear probing** because it eliminates **primary clustering**.
  - However, if $h(k_1) = h(k_2)$ the **probing sequences** for k_1 and k_2 are exactly the same. This sequence of locations is called a **secondary clustering**.
  - **Example:**



Open Addressing. Quadratic probing

- **Bad News:**
  - For **quadratic probing** is **NO** guarantee of finding an empty cell once the table gets more than half full, or even before the table gets half full if the table size is not prime.

Theorem: if the table is half empty ($\lambda < 1/2$) and the TableSize is prime, then we are always guaranteed to be able to insert a new element.

Proof: (see textbook, page 182)

Quadratic Probing. Java Code

- **See Java Code for Quadratic Probing in:**

http://users.cis.fiu.edu/~weiss/dsaajava/code/DataStructures/QuadraticProbingHashTable.java

Author: Mark A. Weiss

9

Open Addressing. Double Hashing

General Idea:

- Given **two good hash functions u** and **v**, it is very unlikely that for some **key**, **u(key) == v(key)**
- So make the probe function **f(i) = i*v(key)**

Finding a position…

h(key) = key % TableSize

- and the **probe sequence** is:
  - 1st probe: **(u(key) + 1*v(key)) % TableSize**
  - 2nd probe: **(u(key) + 2*v(key)) % TableSize**
  - 3rd probe: **(u(key) + 3*v(key)) % TableSize**
  - …
  - ith probe: **(u(key) + i*v(key)) % TableSize**

Detail: Make sure **v(key)** cannot be **0**

10

Open Addressing. Double Hashing

- If we have
  - **u(key) = key** and,
  - **v(key) = *R* − (key mod *R*)**, with ***R*** a prime smaller than **TableSize**, will work well.
- **Example:**
  - If we choose ***R* = 7**, then to insert {89,18,49,58,69} we have:

i^{th} probe: **(key + i*(7 − (key % 7)) % TableSize**

	Empty Table	After 89	After 18	After 49	After 58	After 69
0						69
1						
2						
3					58	58
4						
5						
6				49	49	49
7						
8			18	18	18	18
9		89	89	89	89	89

Open Addressing. Double Hashing Analysis

- By simple intuition we can suppose that if each probe is “jumping” by **v(key)** each time, we “leave the neighborhood” and “go different places from other initial collisions”
- But we could still have a problem like in **quadratic probing** where we are not “safe” (infinite loop despite room in table)

Good News:

- It is known that this **cannot happen** in at least one case:
  - **u(key) = key % p**
  - **v(key) = q − (key % q)**
  - **2 < q < p**
  - **p and q are prime**

Open Addr. Improving the key distribution

	Linear Probing	Quadratic Probing	Double Hashing
0	49	49	69
1	58		
2	69	58	
3		69	58
4			
5			
6			49
7			
8	18	18	18
9	89	89	89

Comparison: Linear, Quadratic, and Double

- **Separate chaining** hashing requires the use of links, which costs some memory, and the standard method of implementing calls on memory allocation routines, which typically are expensive.
- **Quadratic probing** is only slightly more difficult to implement and gives good performance in practice. An insertion can fail if the table is half empty, but this is not likely.
- **Double hashing** eliminates primary and secondary clustering, but the computation of a second hash function can be costly.
- In most cases **quadratic probing is the fastest method.**

Comparison: AVL-tree vs. Hash Table

	AVL-tree	HashTable
Ins, Del, Find complexity	O(logN)	O(1) in average
Find Min/Max	Yes	No
Items in a range	Yes	No
Sorted Input	Very Bad (many rotations)	No problems

- **Recommendation:**
  - Use **Hash Table** if there is any suspicion of **SORTED input** & **NO ordering** information is required.

15

16