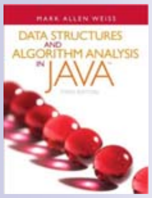
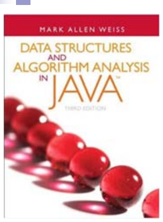


FLORIDA INTERNATIONAL UNIVERSITY

Hashing (II)

Dr. Antonio L. Bajuelos

FIU School of Computing & Information Sciences



COP-3530 - Data Structures

Module #5: Hashing (part II)

Outline:

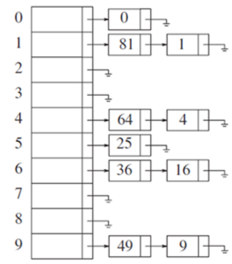
- **The collision resolution:**
  - **(II) Open addressing**
    - **General ideas**
    - **Advantages vs Disadvantages**
    - **(a) Linear probing**
      - **Primary clustering**
      - **Expected number of probing**
      - **Complexity analysis and Java code**

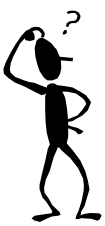
2

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?

Open Addressing

Main idea: use empty space in the table

Important points:

- <u>All items are stored in the hash table itself</u>.
- 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.
- **Deletion** (**lazy deletion**): When a key is deleted the slot is marked as **DELETED**.
- **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.

Open Addressing

- 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.

- All items are stored in the hash table itself.
- The function **c(i)** is used to resolve collisions.
- Similarly, to find item with the key **k**, we examine the same sequence of locations in the same order.
- For a given hash function **h(key)**, the only difference in the **open addressing collision resolution** techniques is in the definition of the function **c(i)**.

5

Open Addressing

- **Advantages of Open Addressing:**
  - All items are stored in the hash table itself. There is no need for another data structure.
- **Disadvantages of Open Addressing:**
  - The keys of the objects to be hashed must be distinct.
  - Dependent on choosing a proper table size.
  - Requires the use of a three-state (**EMPTY**, **OCCUPIED**, **DELETED**) flag in each cell.

6

Open Addressing Linear probing

Linear function: c(i) = i

- If **h(key)** = **key % TableSize** is already occupied then
  - try **(h(key) + 1) % TableSize**. If occupied...
  - try **(h(key) + 2) % TableSize**. If occupied...
  - try **(h(key) + 3) % TableSize**. If occupied...
- **Example:** insert keys **{89, 18, 49, 58, 69}**

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

- Note that this table is relatively empty but blocks of occupied cells start forming. This effect, known as **primary clustering**, means that **keys tend to cluster around table locations that they originally hash to.**

Open Addressing. Linear probing

If position **h(key) = key mod TableSize** is **occupied** then

Apply the **linear probing**

ith probe was **(h(key) + i) % TableSize**, i =1, 2, 3, 4, ...

- **Example:** insert {5, 15, 6, 3, 27, 8}

0	1	2	3	4	5	6	7	8	9
					5				
					5	15			
					5	15	6		
			3		5	15	6		
			3		5	15	6	27	
			3		5	15	6	27	8

Primary clustering

Open Addressing. Linear probing

- **insert** finds a free table position using a **linear probe function**
- What about **find**?
  - Must use same **probe function** to follow the path for the data
  - **Unsuccessful search** when reach **empty position**
- What about **delete**?
  - Must use “lazy” deletion.
  - Marker indicates “no data here, but don’t stop probing”
  - “Real” deletion (clean table) – off-line process (rehash)

0	8
1	109
2	10
3	/
4	/
5	/
6	/
7	/
8	38
9	19

- **Example:**

find(109)= 1

find(58)= null (T[8],T[9],T[0],T[1], and T[2] ≠ 58, T[3]=null)

delete(38) ⇨ T[8] = “no data, don’t stop” ⇦ Lazy Deletion!

find(8), T[8] ? 8, no data, move to next

T[9] ? 8, 19 ≠ 8, move to next

T[0] ? 0, 0 = 0, YES!, find(8) = 0

9

Open Addressing. Linear probing

(h(key) + i) % TableSize

- **Trivial fact:** For any λ **< 1** (**N < TableSize**), **linear probing** will find an empty cell. **So, no infinite loop unless table is full**.

- **Non-trivial fact:**
  - For **insertions and searches** the **expected number of probes using linear probing** is:
  - For **unsuccessful searches**:

$$\frac{1}{2}\left(1+\frac{1}{(1-\lambda)^2}\right)$$

Example: For λ=1/2 the # of probes < 2.5
For λ=1/4 the # of probes < 1.38

  - For **successful searches**:

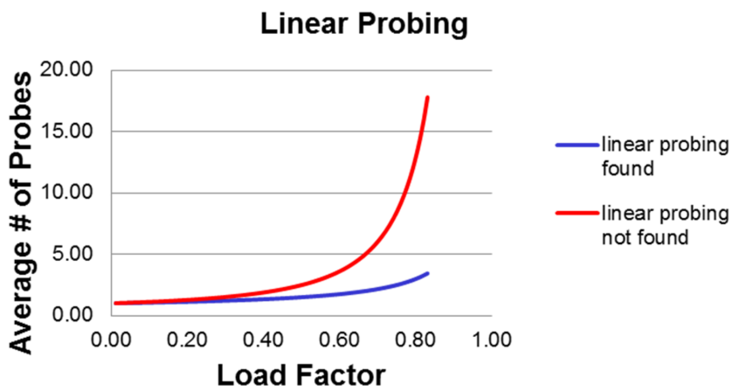
$$\frac{1}{2}\left(1+\frac{1}{(1-\lambda)}\right)$$

Example: For λ=1/2 the # of probes < 1.5
For λ=1/4 the # of probes < 1.16

10

Open Addressing. Linear probing

- **Facts!**
  - Need to leave sufficient empty space in the table to get good performance
  - Linear-probing performance degrades rapidly as table gets full (i.e. when $\lambda \rightarrow 1$ then the number of probes is increased)



11

Linear probing. Java Code

- **See Java Code for Linear probing in:**

  http://algs4.cs.princeton.edu/34hash/LinearProbingHashST.java.html

  Authors: Robert Sedgewick and Kevin Wayne

- **Hashing visualization in:**

  http://iswsa.acm.org/mphf/openDSAPerfectHashAnimation/perfectHashAV.html

12