CSC2001F: Data Structures II

Omowunmi Isafiade

Email: omowunmi.isafiade@uct.ac.za

Office: Room 306

Outline

- Quadratic probing – A method of Collision Resolution
- Load Factor (re-visited)
- Rehashing and Overflow (The problem of Table Size)
- Secondary Clustering
- Double hashing – A method of Collision Resolution
- Separate Chaining – A method of Collision Resolution
- Summary

Quadratic Probing – Table Size Issue

- Recall Example: with quadratic probing we have the problem of inserting 53
- Issue…
  - Choice of Table Size & Load Factor (>= 0.5)
- Solution:
  - If table size is prime and load factor is never >0.5
- Advantage: we can always insert a new item and no cell is probed more than twice during an access.

Quadratic Probing – Table Size & Load Factor

- What happens if the table size is too small
- What happens if quadratic probing cannot resolve the collision?
- Possible Solutions:
  - Adjust the load factor of the hash table by expanding the table size (**Rehashing**)
  - Requires that the load factor satisfies a constraint (<= threshold value)
  - Set pre-conditions for the table size

Methods of Collision Resolution?

- **Recall # 1**: **Linear probing (LP)**
  - Probe alternative locations successively (H+1, H+2, H+3,....)
  - Primary clustering (**problem - expensive**)
- **Recall # 2**: **Quadratic probing (QP)**
  - Probe alternative locations away from original probe point H → (H+1, H+4, H+9 ...)
  - **Resolves primary clustering**
  - BUT!!! Results in secondary clustering

Reflection: In LP, each probe tries a different cell. Does QP always guarantee that? And if table is not full does QP always guarantee an insertion? (**Table size-prime & load factor < 0.5**)

Secondary Clustering

- **Note:** Secondary clustering is a consequence of quadratic probing
- Since items probe the same alternative cells during collision resolution
- How do we resolve this?

"Approach characteristics to quadratic probing whereby elements that hash out to the same position probe the same alternative cells"

0	10
1	
2	
3	
4	337
5	617
6	123
7	93
8	17
9	
10	63

Secondary cluster formation due to probing of the Same alternative cells to resolve collision

How do we resolve secondary clustering?

Resolving Secondary Clustering?

- **Alternatives** to quadratic probing that circumvent secondary clustering
  - **Double Hashing** – **a method of collision resolution**
    - Does not suffer from secondary clustering
    - A second hash function is used to drive the collision resolution (uses **two hash functions**)
  - **Separate Channing Hashing – a method of collision resolution**
    - “Space efficient alternative. Uses a combination of an array and linked lists”
    - Less sensitive to high load factors

What is Double Hashing?

- **Double Hashing** – **a method of collision resolution**
  - Uses two hash functions, h1 and h2
  - A **second hash function** (h2) is used to drive the collision resolution (uses **two hash functions**)
    - **h1(k)** is the position where the function hash out to (the evaluated index value)
    - **h2(k)** determines the **probing sequence (offset)** for specific locations to check (i.e for insertion)
- **Note:**
  - keys could have different probing sequence (offset)
  - **Contrast to quadratic probing** where same alternative cells are probed to resolve collision
  - In linear probing **h2(k)** is always 1

Double Hashing – How it works

```
DoubleHashingInsert( K )
If (table is full) throw an exception
    probe = h1 (k);
    offset = h2 (k);
While (table [probe] occupied)
    probe = (probe + offset) mod m;
table[probe] = k;
```

Note: offset is determined by h2(k), so it can be different for different keys (dynamic)

Double Hashing

- Has many of the same (dis)advantages as linear probing
- **BUT**! Distributes key more uniformly than linear probing (no clusters formed)
- If “m” is prime, every position in the hash table eventually be examined
- **Note:** Avoid “cycling back” – you tend to cycle back when your **offset, h2(k),** divides m

Double Hashing - Observations

- Assumption: every probe looks at a random location in the hash table
- **Load factor is less than 1** ($\alpha < 1$)
- $1-\alpha$ fraction of the table is empty
- Less sensitive to high load factors
- Expected number of probes required to find an empty location (unsuccessful search is $1/(1-\alpha)$)

Double Hashing - Example

- Using double hashing, insert the following keys {337, 123, 617, 93, 63,17, 37,43, 77} into a hash table of size 13
- Hash function: **h1** = k mod m
- Hash function: **h2** = 8 – (k mod 8)
- Hash function computation gives...?

k	337	123	617	93	63	17	37	43	77
h1(k)									
h2(k)									

Double Hashing – Example (Solution)

- Using double hashing, insert the following keys {337, 123, 617, 93, 63,17, 37,43, 77} into a hash table of **size 13**
- Hash function: **h1** = k mod m
- Hash function (**Note: offset**): **h2** = 8 – (k mod 8)
- Hash function computation gives…?

k	337	123	617	93	63	17	37	43	77
h1(k)	12	6	6	2	11	4	11	4	12
h2(k)			7				3	5	3

Double Hashing – Example (Solution)

- Step 1: Insert 337
- Step 2: Insert 123
- **Step 3**: Insert 617 (**collision!**)
  - Prob + offset = 6+7 =0 (so goes to 0)
- Step 4: Insert 93
- Step 5: Insert 63
- Step 6: Insert 17
- **Step 7**: Insert 37 (**collision!**)
  - Prob + offset = 11+3 = 1 (so goes to 0)
- **Step 8**: Insert 43 (**collision!**)
  - Prob + offset = 4+5 = 9 (so goes to 9)
- **Step 9**: Insert 77 (**collision!**)
  - Prob + offset = 12 + 3 = 2 (**occupied!!!)**
  - 2 + 3 = 5 **(insert!!!)**

0	617
1	37
2	93
3	
4	17
5	77
6	123
7	
8	
9	43
10	
11	63
12	337

Double Hashing – Example (Solution)

- Using double hashing, insert the following keys {337, 123, 617, 93, 63,17, 37,43, 77} into a hash table of **size 13**

k	337	123	617	93	63	17	37	43	77
h1(k)	12	6	6	2	11	4	11	4	12
h2(k)			7				3	5	3

0	617
1	37
2	93
3	
4	17
5	77
6	123
7	
8	
9	43
10	
11	63
12	337

Resolving Secondary Clustering?

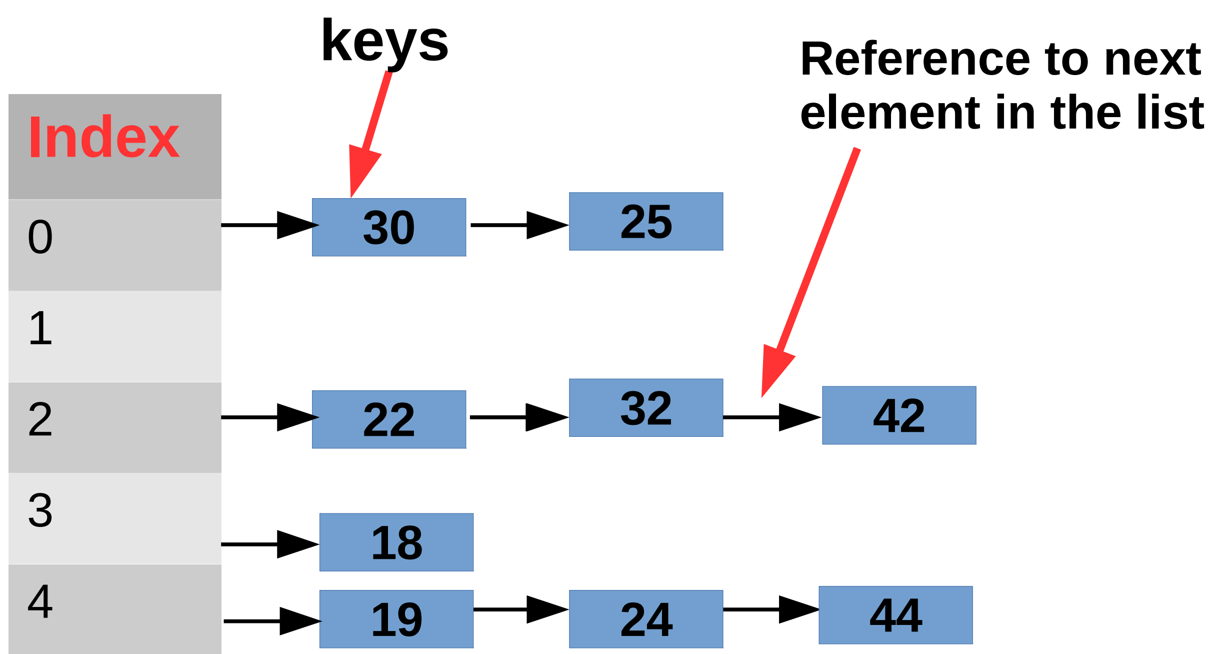
- **Alternatives** to quadratic probing that circumvent secondary clustering
  - **Double Hashing** – **a method of collision resolution**
    - Does not suffer from secondary clustering
    - A second hash function is used to drive the collision resolution (uses **two hash functions**)
  - **Separate Channing Hashing** – **a method of collision resolution**
    - "Space efficient alternative. Uses a combination of an array and linked lists"
    - Less sensitive to high load factors

Separate Chaining Hashing

- **Separate Channing Hashing – a method of collision resolution**
  - Maintains an array of linked lists
  - For an array of linked lists, the hash function tells us which list to insert an item X
  - And during a find operation, which list contains X
  - **AIM**: Although searching linked lists is a linear operation, if the lists are short the search time will be very fast

Separate Chaining Example

- Using separate chaining insert the keys {22, 32, 18, 19 and 30, 25, 42, 24} into a hash table of size 5 using the hash function h(k) = k mod m
- h(22) = 2
- h(32) = 2
- h(18) = 3
- h(19) = 4
- h(30) = 0
- h(25) = 0
- h(42) = 2
- h(24) = 4
- h(44) = 4



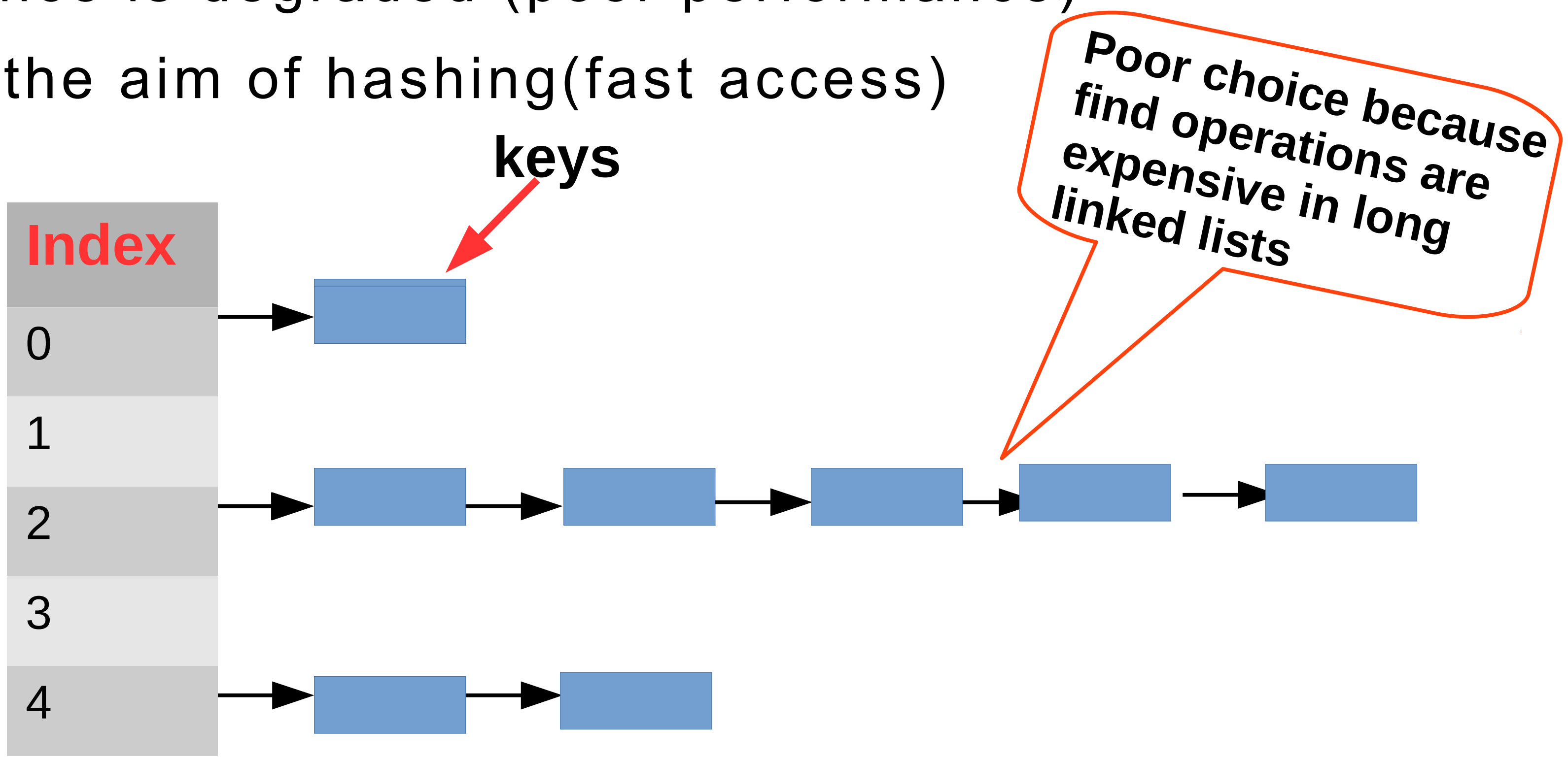
Separate Chaining Hashing: Observations

- **Load factor can be > 1.0**
  - Less sensitive to high load factor
  - **Rehashing is avoided**

- **Choose a hash function that distributes key equitably**
  - Reduces cost of searching long linked lists attached to single probe
  - Choose a sufficiently large (prime) table size to ensure that lists are short
  - E.g for an array of 2000 items, choose a prime approximately close to (2000/3).
    - i.e 701 (prime) ensures not more than 3 collisions per index

Separate Chaining Hashing: Observations

- Example: if keys are not uniformly distributed , performance is degraded (poor performance)
- Defeats the aim of hashing(fast access)



Hash Tables versus Binary Search Trees

- "Hash table useful instead of binary search tree if you do not need order statistics and are worried about non-random points"

S/N	Hash Tables	Binary Search Trees
1	Not efficient for finding minimum element	Good for finding min or max
2	Searches for strings are inefficient when the exact string is not known	Can quickly find all strings (items) within a certain range
3	O(1) on searches and inserts	O(log N) bound on searches and inserts
4	Good when no ordering is needed or when data is sorted.	Good when ordering is needed and the data is not sorted.

Next Class...

- The Priority Queue … (**Chapter 21**)

Reference Textbook:

“Data Structures & Problem Solving using Java”, 4th Ed., Mark A. Weiss.