Hashing

Dr. Yingwu Zhu
What do I have?

• Recall order of magnitude of searches
  – Linear search $O(n)$
  – Binary search $O(\log_2 n)$
  – Balanced binary tree search $O(\log_2 n)$
  – Unbalanced binary tree can degrade to $O(n)$
Hash Tables

• Sometime faster search is needed
  – Solution: use **hashing**
  – Value of key field fed into a hash function
  – Location in a hash table is calculated
Hashing

• Key to hashing
  – The hash function $h(x)$
Hash Functions

• Simple function could be to mod the value of the key by some arbitrary integer

```c
int h(int i)
{
    return i % someInt;
}
```

• Note the max number of locations in the table will be same as `someInt`

• Note that we have traded space for performance
  – Table must be considerably larger than number of items anticipated
Collision Problem

- Collision: same value returned by $h(i)$ for different values of $i$
  - $h(i) = i \mod 31$
Hash Functions

• Strategies for improved performance
  – Increase table capacity (less collisions)
  – Use a different collision resolution technique
  – Devise a different hash function
Hash Table Capacity

• Size of table must be 1.5 to 2 times the size of the number of items to be stored
• Otherwise probability of collisions is too high
• Sometimes may be hard to get the estimate of the number of items
Solution #1: Linear Probing

- **Insertion**
  - Linear search begins at collision location
  - Continues until empty slot found for insertion
- **When retrieving a value**
  - Linear probe until found
  - If empty slot encountered then value is not in table
- **If deletions permitted**
  - Slot can be marked so it will not be empty and cause an invalid linear probe
Example: Linear Probing

• $h(x) = x \mod 31$, the hash table has size of 31
  – Insertion order of 620, 64, 128, 467, 777, 35, 127, 282
• Use linear probing to solve collision
Solution #2: Quadratic Probing

• Linear probing can result in primary clustering.
• Consider quadratic probing
  – Probe sequence from location $i$ is $i + 1, i - 1, i + 2^2, i - 2^2, i + 3^2, i - 3^2, ...$
  – Exercise: using quadratic probing to solve
  – Drawback: Secondary clusters can still form.
Solution #3: Double Hashing

• Double hashing
  – Use a second hash function to determine probe sequence
  – Two hash functions
    • $h_1(x) = i$
    • $h_2(x) = k$
    • Probing sequence $i, i+k, i+2k, \ldots$
Example: Double Hashing

• $h(x) = x \% 31$, the hash table has size of 31
  – Insertion order of 620, 64, 128, 467, 777, 35, 127, 282
• Exercise: Use double hashing to solve collision
  – $h_1(x) = x \% 31$
  – $h_2(x) = 17 - (x \% 17)$
Solution 4: Chaining

- **Chaining**
  - Table is a list of head nodes to linked lists
  - When item hashes to location, it is added to that linked list

![Diagram showing a table with linked lists]
Improve the Hash Function

• Ideal hash function
  – Simple to evaluate
  – Scatters items uniformly throughout table (reducing collision)

• Modulo arithmetic not so good for strings
  – Possible to manipulate numeric (ASCII) value of first and last characters of a name
Do you know any good hash function?

- **MD5** hashing, $h(x)=16$ bytes
- **SHA-1** hashing, $h(x)=20$ bytes
- Hope you spend some time on googling these two to get a taste!!!!
Review

• Why Hashing?
• What does hashing do?
• One problem of hashing: collision
  – Degrade search performance
• 3 strategies to improve hashing performance
• Collision Strategies
• How to evaluate if a hash function is good?