Chapter 13

Hash Tables

Balanced Search Trees

- The efficiency of the binary search tree implementation of the ADT table is related to the tree's height
  - Height of a binary search tree of n items
    - Maximum: n
    - Minimum: $\lceil \log_2(n + 1) \rceil$
- Height of a binary search tree is sensitive to the order of insertions and deletions
- Variations of the binary search tree
  - Can retain their balance despite insertions and deletions

Hashing

- Hashing
  - Enables access to table items in time that is relatively constant and independent of the items
- Hash function
  - Maps the search key of a table item into a location that will contain the item
- Hash table
  - An array that contains the table items, as assigned by a hash function

Hashing

- A perfect hash function
  - Maps each search key into a unique location of the hash table
  - Possible if all the search keys are known
- Collisions
  - Occur when the hash function maps more than one item into the same array location
- Collision-resolution schemes
  - Assign locations in the hash table to items with different search keys when the items are involved in a collision
- Requirements for a hash function
  - Be easy and fast to compute
  - Place items evenly throughout the hash table

Hash Functions

- It is sufficient for hash functions to operate on integers
- Simple hash functions that operate on positive integers
  - Selecting digits
  - Folding
  - Modulo arithmetic
- Converting a character string to an integer
  - If the search key is a character string, it can be converted into an integer before the hash function is applied

Resolving Collisions

- Two approaches to collision resolution
  - Approach 1: Open addressing
    - A category of collision resolution schemes that probe for an empty, or open, location in the hash table
      - The sequence of locations that are examined is the probe sequence
    - Linear probing
      - Searches the hash table sequentially, starting from the original location specified by the hash function
      - Possible problem
        - » Primary clustering

Resolving Collisions

- Approach 1: Open addressing (Continued)
  - Quadratic probing
    - Searches the hash table beginning with the original location that the hash function specifies and continues at increments of 1^2, 2^2, 3^2, and so on
    - Possible problem
      - Secondary clustering
  - Double hashing
    - Uses two hash functions
    - Searches the hash table starting from the location that one hash function determines and considers every n^{th} location, where n is determined from a second hash function
- Increasing the size of the hash table
  - The hash function must be applied to every item in the old hash table before the item is placed into the new hash table

Resolving Collisions

- Approach 2: Restructuring the hash table
  - Changes the structure of the hash table so that it can accommodate more than one item in the same location
  - Buckets
    - Each location in the hash table is itself an array called a bucket
  - Separate chaining
    - Each hash table location is a linked list

The Efficiency of Hashing

- An analysis of the average-case efficiency of hashing involves the load factor
  - Load factor α
    - Ratio of the current number of items in the table to the maximum size of the array `table`
    - Measures how full a hash table is
    - Should not exceed 2/3
  - Hashing efficiency for a particular search also depends on whether the search is successful
    - Unsuccessful searches generally require more time than successful searches

The Efficiency of Hashing

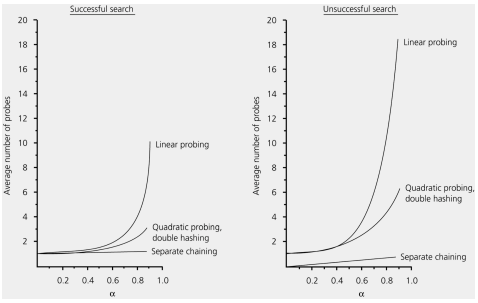

Figure 13-50

The relative efficiency of four collision-resolution methods

What Constitutes a Good Hash Function?

- A good hash function should
  - Be easy and fast to compute
  - Scatter the data evenly throughout the hash table
- Issues to consider with regard to how evenly a hash function scatters the search keys
  - How well does the hash function scatter random data?
  - How well does the hash function scatter nonrandom data?
- General requirements of a hash function
  - The calculation of the hash function should involve the entire search key
  - If a hash function uses modulo arithmetic, the base should be prime

Summary

- Hashing as a table implementation calculates where the data item should be rather than search for it