Chapter 3 - Direct File Organization

LOCATING INFORMATION

- Problem: desire for rapid access to large volumes of data.
- Solution: simulating the associative human storage and retrieval processes.

Three ways to organize a file for direct access:

1- The key is a unique address
  - Ex: 9-digit SSN as key requires 1 billion table entries
    - space tradeoff
  - Ex: 4 digit employee number requires 1000 table entries
    - not suitable for dynamic environments

2- The key converts to a unique address
  - Similar to finding records in contiguous locations:
    - Location A[i] = base_address + (i-1) * element_size
  - Ex: flight number + day of year combination in an airline reservation system
    - Think about the effects of concatenation order on the efficiency.

3- The key converts to a probable address (Hashing)
  - Key space is larger than the address space (in contrast the previous two)
  - Multiple key values are mapped to a single address value:

Hash(key) ® probable address

Home address: the initial probable address for locating a record in a table.

Desired properties of a hash function:

- evenly distributes the keys among the addresses.
- executes efficiently.

A **collision** occurs when two distinct keys map to the same address.

Hashing is then composed of two aspects:

- a function
- a collision resolution method

Hashing Functions

- **Key mod N**

  N is the table size.
- **Key mod P**

  P is the smallest prime number ≥ N.
- **Truncation (Substringing)**

  Select any "appropriate" digits of the given key.
- **Folding**

  Folding by boundary and folding by shifting.
- **Squaring**

  ... followed by truncating a portion of the result.
- **Radix Conversion**

  The key is converted to base 10.
- **Alphabetical Keys**

  Alphabetic or alphanumeric key values can be input to a hashing function if the values are *interpreted* as integers. (Think about character data type in C.)

Collisions

A hashing function that has a large number of collisions or synonyms is said to exhibit primary clustering.

Aim: reduce the number of collisions

Solution 1: change the hashing functions.

Solution 2: reduce the load factor (i.e., # of stored records / total # storage locations)

- Load factor is a measure of storage utilization.
- As the load factor increases, the likelihood of a collision increases.
  - e.g., compare with the number of cars in city traffic.
- Time-space Tradeoff:
  - increased load factor
  - more space, decreased load factor more collisions
- Collisions typically increase rapidly when the packing factor goes beyond about 90 percent.

Collision Resolution

Changing the hashing function or decreasing load factor } may *reduce* the number of collisions, but will usually not *eliminate* them.

Therefore;

we need a procedure to position a synonym at another location.

i.e., we want to place a synonym in a location that requires minimum number of additional probes from its home address.

Probe: an access to a distinct location.

Several mechanisms for resolving collisions:

- Collision resolution with links
  - Disadv: additional space required for links.
- Collision resolution without links
  - "implied links" by applying a convention, or set of rules for deciding where to go next. (Disadv: additional probes)
  - Static positioning of records
  - Dynamic positioning of records
- Collision resolution with pseudolinks

Coalesced Hashing (Direct Chaining)

- is a collision method that uses pointers to connect the elements of a synonym chain.
- obtains its name from what occurs when we attempt to insert a record with a home address that is already occupied by a record from a chain with a different home address.
  - two chains with records having different home addresses coalesce or grow together.

See Figure 3.4

Since coalesced chains will require more probes than noncoalesced chains, we want to minimize coalescing to improve retrieval performance.

- compute the average number of probes for both cases.

Insertions made at the bottommost (highest address) empty location as a matter of convention. In searching for an empty location, an available space pointer is continually decremented from its current position until either an empty location or end of table is found.

Deletions require moving a record later (actually the last element) in the probe chain into the position of the deleted record. (assume that coalescing has occured!)

Variants

may be classified in three ways:

- The table organization (whether or not a separate overflow area is used.)
- The manner of linking a colliding item into a chain.
- The manner of choosing unoccupied locations.

1- Modifying the table organization.

The table is divided into a primary area and an overflow area.

Address factor = primary area / total table size

For a fixed amount of storage, as the address factor decreases,
the overflow size increases,
which reduces the coalescing but increases the number of collisions.
(An address factor of 0.86 yields nearly optimal retrieval performance for most load factors)

LISCH (Late Insertion Coalesced Hashing): insert new records at the end of probe chain.
LICH (Late Insertion Coalesced Hashing): insert new records at the end of probe chain and as oppose to LISCH there is a separate overflow area.

2- Vary the position in which new records are inserted into a probe chain.

EISCH: inserts a new record into a position on the probe chain *immediately* after the record stored at its home address.

See Figure 3.6

The rational behind early insertion is to reduce the amount of coalescing the two synonym chains may have.

3- Vary the way in which we choose an unoccupied location for inserting new records.

- we chose an unoccupied location from the bottom of the storage area above.
  - by concentrating all the overflow items in one area of the table, we increase the number of collisions and thereby degrade the performance.
- choose a location at random (REISCH - Random Early Insertion Standard)
- move records not stored at its home address (DCWC - Direct chaining without coalescing)

Progressive Overflow (Linear Probing)

Eliminates additional storage needed for the link fields in coalesced hashing.
It uses a convention for where to search next instead of a physical link.

- if a location is occupied, we then look at the next location to see if it is empty.

Secondary clustering occurs with a hashing scheme like this in which

- the incrementing function is a constant
- or depends only upon the home address of a record.

(Similar to bunch of cars in a highway)

Deletion is done by marking the deleted records with a tombstones.

An Example: The keys of the records are: **27, 18, 29, 28, 39, 13,** and **16**
and the hashing function is **Hash(key) = key mod 11**.

Use of Buckets

We can reduce the number of required accesses by storing multiple records at one file address.

When a storage location may hold multiple records, it is referred to as a bucket.

Bucketing factor: the number of buckets that may be stored in a bucket.

An Example: The keys of the records are: **27, 18, 29, 28, 39, 13,** and **16**
and the hashing function is **Hash(key) = key mod 11**.

Linear Quotient

- As oppose to progressive overflow, we use a variable increment instead of a constant increment of one.
  - variable increment reduces secondary clustering.
  - the increment is a function of the key being inserted.
    - it is actually viewed as another hashing function, therefore linear quotient is a member of double hashing methods.

$$H_2 = \text{Quotient}(\text{Key} / P) \bmod P$$

P is the prime number table size? Why does it need to be a prime?

- Synonyms are not usually on the same probe chain (as in PO) with linear quotient.

An Example: The keys of the records are: **27, 18, 29, 28, 39, 13,** and **16**
and the hashing function is **Hash(key) = key mod 11**.