

Storage Structures

Lecture 8

Memory Hierarchies

- Primary Storage
 - Registers
 - Cache memory
 - RAM
- Secondary Storage
 - Magnetic disks
 - Magnetic tape
 - CDROM (read-only / write-once)
 - Flash memory

Secondary Storage Characteristics

- Random access versus sequential access
- Read-Write, Write-once, Read-only
- Character versus block data access

Storage of Databases

- Need data to be stored “permanently” or *persistently* for long periods of time
- Usually too big to fit in main memory
- Low cost of storage per unit of data and the definition of “very large databases”
- Primary and secondary (auxiliary) file organizations

File Organizations

- Relations usually stored in files as logical “records” and read in terms of physical “blocks”
- File organization refers to the way records are stored in terms of blocks and the way blocks are placed on the storage medium and interlinked.
- Types of organizations
 - Unsorted
 - Sorted
 - Hashing

Records

- Represents a tuple in a relation
- A file is a *sequence* of records
- Records could be either *fixed-length* or *variable-length*
- Records comprise of a sequence of *fields* (column, attribute)

Blocks

- Refer to physical units of storage in storage devices (Example: Sectors in hard disks, page in virtual memory)
- Usually (but not necessarily) stores records of a single file
- Of fixed length, based on physical characteristics of the storage/computing device and operating system
- Storage device is either defragmented or fragmented depending on whether contiguous sets of records lie in contiguous blocks

Blocking Factor

The number of records that are stored in a block is called the “blocking factor”. Blocking factor is constant across blocks if record length is fixed, or variable otherwise.

If B is block size and R is record size, then blocking factor is:

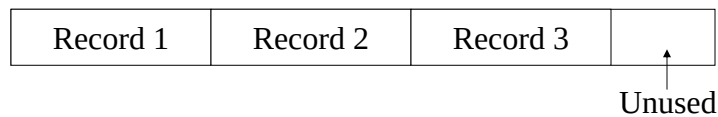
$$\text{bfr} = \lfloor B/R \rfloor$$

Since R may not exactly divide B, there could be some left-over space in each block equal to:

$$B - (\text{bfr} * R) \text{ bytes.}$$

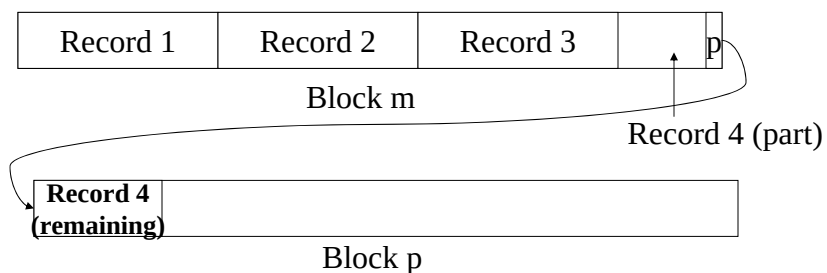
Spanned and Unspanned Records

When extra space in blocks are left unused, the record organization is said to be “unspanned”.



Spanned and Unspanned Records

In “spanned” record storage, records can be split so that the “span” across blocks.



Spanned and Unspanned Records

When record size is greater than block size (i.e. $R > B$), use of “spanned” record storage is compulsory.

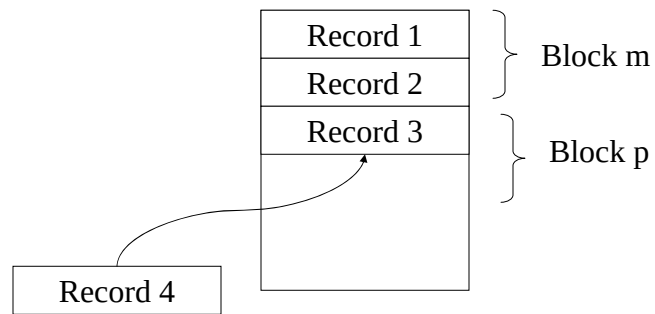
Average Blocking Factor

For variable-length records, with the use of record spanning, the “average” record length can be used in *bfr* computation to compute the number of blocks required for a file of records.

$$b = \lceil (r / bfr) \rceil$$

where *b* is the number of blocks required to store a file having *r* records of variable length with the overall blocking factor *bfr*.

Unordered File Organization



Also called a “pile file”...

Pile Files

- Simplest file organization
- Records are inserted in the order of their arrival
- Usually requires secondary files for efficient search
- Insertion: very easy
- Search on non-indexed attribute very expensive

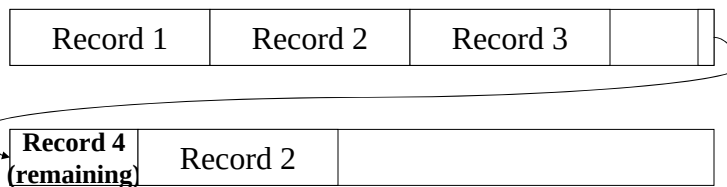
Record Deletion in Pile Files

Record 1	Record 2	Record 3	
----------	----------	----------	--

Record 1		Record 3	
----------	--	----------	--

Deletion by fragmentation.
Inefficient in terms of space usage.

Record Updation in Pile Files



When record length is fixed, records can be updated in place.

When record length is variable, marker based updation is required

Sorted Files

- File organization where records are sorted based on the value of some field called the ordering field
- Ordering field should be a key field (unique for each record) and belong to an ordinal domain (where values can be compared)
- Insertion and Deletion: both expensive
- Updation may involve physical migration of record, if ordering field is modified
- Searching: Binary search based on ordering field efficient

Binary Search

Algorithm BinarySearch(k)

1. $l \leftarrow 1, u \leftarrow b$; // where b is the number of blocks in file
2. while ($u \geq l$) do
 1. $i \leftarrow \lfloor (l+u)/2 \rfloor$
 2. read block i
 3. if $\text{key}(i) < k$
 1. $u \leftarrow i-1$
 4. else
 1. if $\text{key}(i) > k$
 1. $l \leftarrow i+1$
 2. else return block i;
3. done
4. return not-found;

Sorted Files

Record 1
Record 3
Record 7

Record 4
Record 2
Record 8

Overflow File (unsorted)

Master File (Sorted)

Making insertion and updation efficient in sorted files.

Sorted Files

- More efficient than pile-files for key-based searches
- Suitable mainly for random-access devices
- A better choice when database is mostly read-only and queries are mostly key-based retrievals

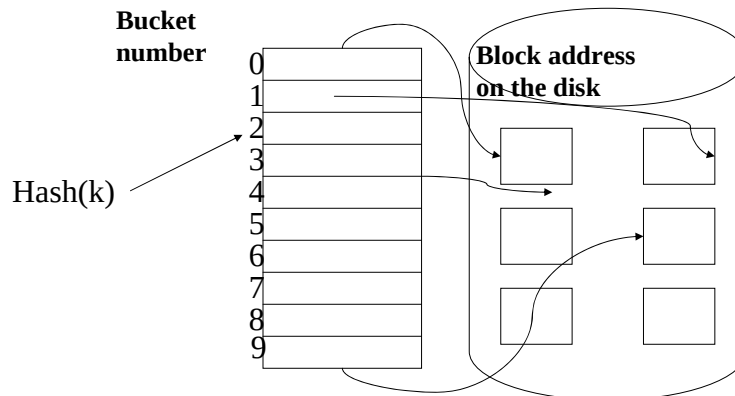
Hashing Techniques

- Provide very fast access to records on certain search conditions
- Search condition should be an equality condition on a “key” field
- Uses a “hashing function” or a “randomizing function” to map keys onto “buckets” hosting records
- Primarily suited for random-access devices

Internal (in-memory) Hashing

- Used as an internal data structure to quickly access records that are read into memory
- Uses a static array of M buckets indexed from 0 to $M-1$
- Several candidates for the hash function: $k \bmod M$, folding the key, sampling the key, etc.

External Hashing



External Hashing

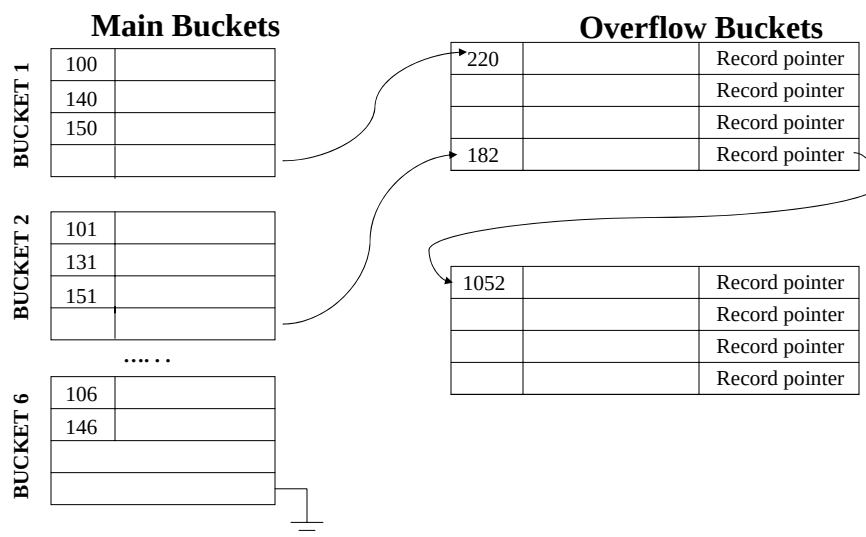
- Uses two levels of indirection: hashing to buckets and searching within buckets
- A bucket is a disk block or a set of contiguous blocks
- A bucket can hold more than one record
- Sequential search within buckets

Overflows

Occurs when a bucket receives more records than it can hold.
Overflow management techniques:

- Open addressing
 - Use the next available bucket.
- Chaining
 - Pointers to extra buckets called overflow buckets.
- Rehashing
 - Use a second hashing function to see if it can hash without overflow.

Handling Overflows with Chaining



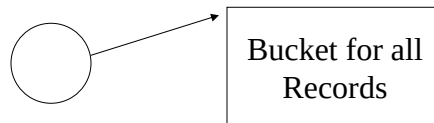
Dynamic Hashing

- Hashing where the number of buckets are fixed are called *static* hashing
- Inefficient when data set is skewed. Few overflowing buckets and large number of nearly empty buckets
- Wastage of space (in maintaining buckets) and time (sequential searching of overflow buckets)
- Extensible or Dynamic hashing: number of buckets change dynamically in accordance with data requirements

Dynamic Hashing

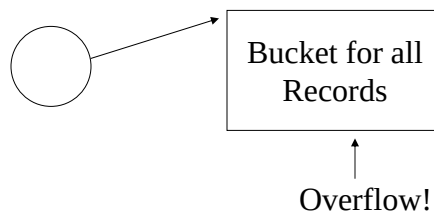
- No. of buckets may grow or shrink depending on additions and deletions
- Overall strategy:
 1. start with a single bucket.
 2. Once full, split the bucket into two.
 3. Redistribute records between the two split buckets using some method ensuring near-uniform distribution

Dynamic Hashing

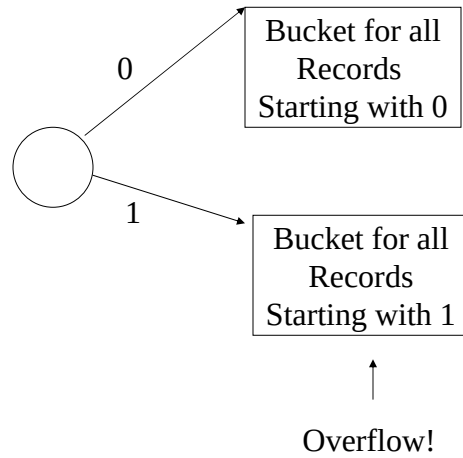


Assume keys to be made of binary strings...

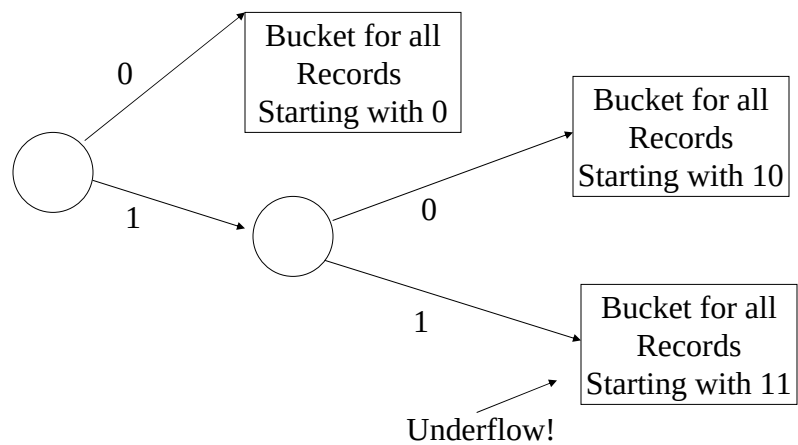
Dynamic Hashing



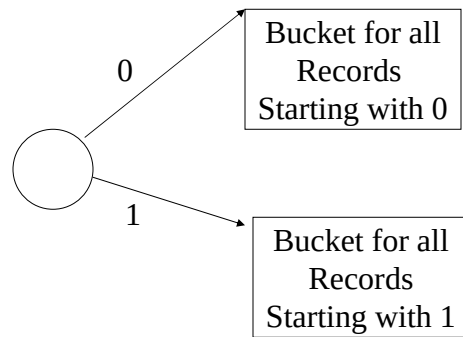
Dynamic Hashing



Dynamic Hashing



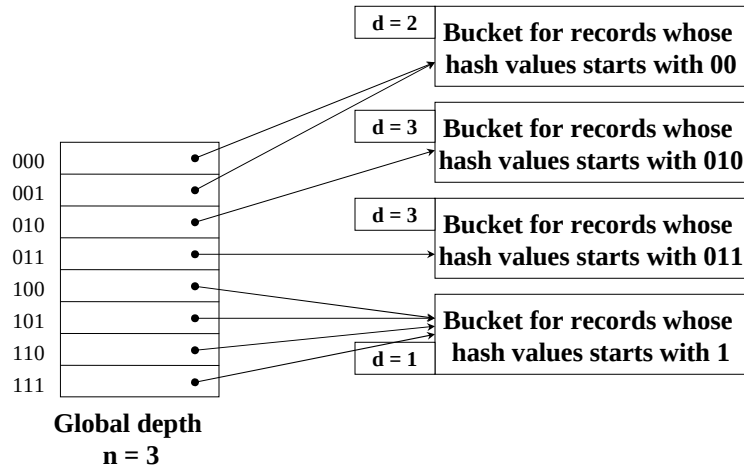
Dynamic Hashing



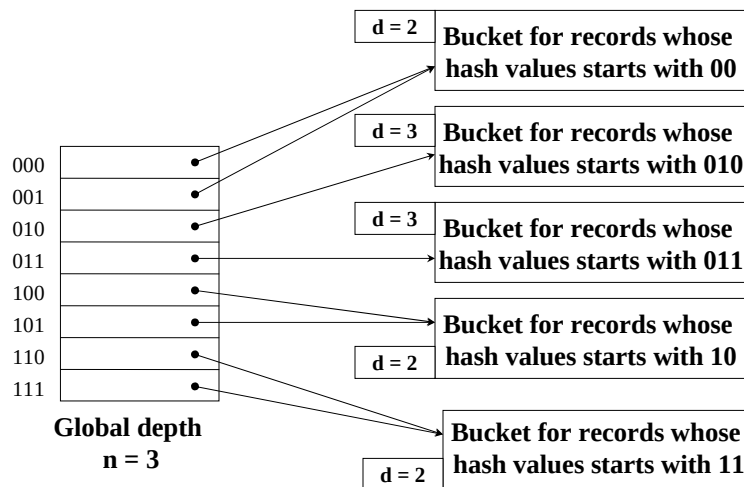
Extensible Hashing

- Uses a global directory of 2^n bucket addresses. Here, n is called the global depth of the directory.
- Higher order $d \leq n$ bits of the key are treated as index into the directory.
- Address in the directory maps to the bucket containing the record.

Extensible Hashing



Extensible Hashing



Summary

- Storage Media and their characteristics
- Records, Blocks and Files
- Spanning and Unspanning records
- Pile Files
- Sorted Files
- Hash Files
 - Static versus dynamic hashing