Lecture 14: Direct Files

Last Day: Updating Sorted Files
- Differential Files
- Bloom Filters

Today: Hash File Reorganization
- Static Reorganization
- Dynamic Reorganization
- Examples

Folk + Zoellick, ch .

Handout on hashing.

Hash File Reorganization

- As a hash file becomes full, collisions increase, and average retrieval time increases.
- So, the file needs to be enlarged.
- There are two basic approaches:
  - Static reorganization (all at once).
  - Dynamic reorganization (incremental).

- So, reorganization is done after business hours,
  eg, at night or on weekends

- Reorganizations are infrequent,
  but massive (ie, the entire file is affected)

Dynamic Reorganization

- The file is reorganized incrementally after each insert or delete.
- Data is accessible during reorganization
- So, reorganization can be done any time, ie, during business hours
- Reorganizations are frequent but small.

Static Reorganization

- Typically, when a hash file is almost full we simply double the number of buckets, From N to $2N$, and change the hash function from $K \bmod N$ to $K \bmod 2N$

- Each record in the file is hashed to a new bucket.

- ie, A record with key K moves
  From bucket $K \bmod N$
  to bucket $K \bmod 2N$

- Note:

$$\underbrace{K \bmod 2N}_{\text{new bucket}} = \begin{cases} (K \bmod N) + N \\ \text{or} \\ \underbrace{(K \bmod N)}_{\text{old bucket}} \end{cases}$$

Example

Doubling the number of buckets from 10 to 20

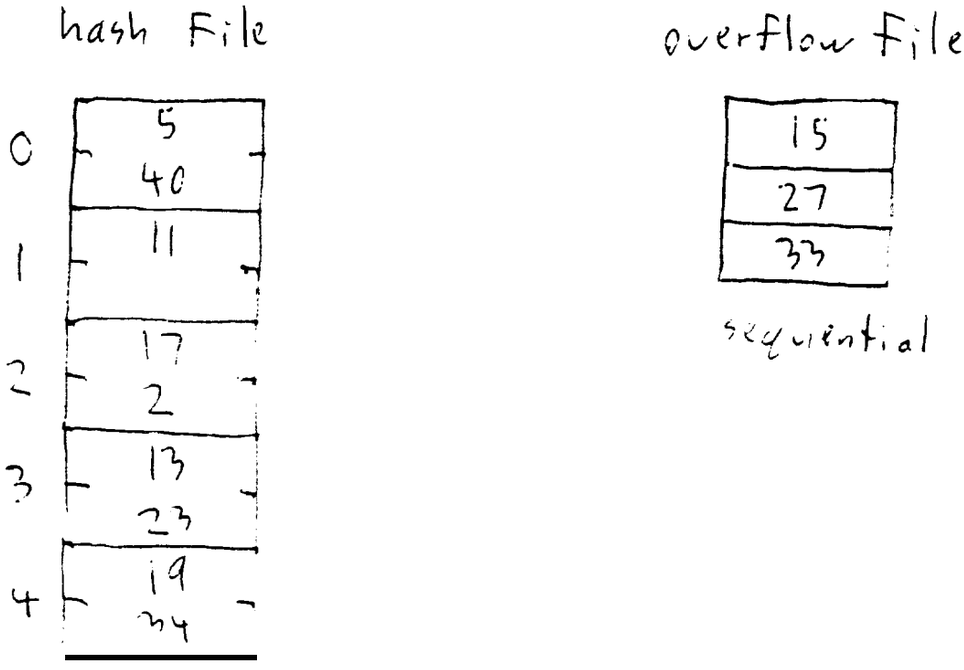
K	K mod 10	K mod 20
3	3	3
13	3	13
23	3	3
33	3	13
43	3	3
53	3	13

Note:

$$K \bmod 20 = \begin{cases} (K \bmod 10) + 10 \\ \text{or} \\ (K \bmod 10) \end{cases}$$

Thus, the contents of bucket i are redistributed among two buckets i and $i+10$.

Example: Static Reorganization



bucket size = 2

hash (k) = k mod 5

An almost Full hash file

with a sequential overflow file.

Example: Static Reorganization (cont.)

- Double the size of the hash file.
  ie, add 5 more buckets

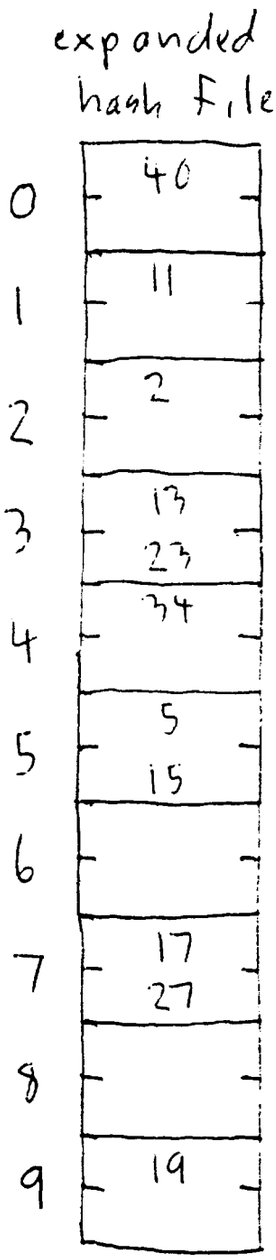
- For each old bucket, i, redistribute the contents to buckets i and i+5, using

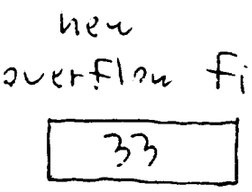
  $$\text{hash}(k) = k \bmod 10$$

- Create a new (empty) overflow file.

- Insert each record from the old overflow file into the hash file. Records that collide are appended to the new overflow file.

Example: Static Reorganization (cont.)





New file contents

Hash File Contraction

- When the hash file becomes less than half full (because of deletions), we can rehash the records to a file of half the size (thus freeing up disk space).

- Thus, $N \longrightarrow N/2$

  $k \bmod N \longrightarrow k \bmod \frac{N}{2}$

- In this case, buckets i and $i + N/2$ are <u>merged</u> into a single bucket, i

- Running the previous example in reverse illustrates contraction.

Disadvantages of Static Reorganization

- It is expensive (time consuming).
- Data in the file is unavailable to users during reorganization.
- So, must be done after business hours, eg, at night or on weekends
- But, this is not possible for 7×24 businesses, ie, businesses that operate 7 days a week, & 24 hours a day.
  eg, airline reservations, 24-hour banking.
- (But, static reorganization is <u>simple</u>.)

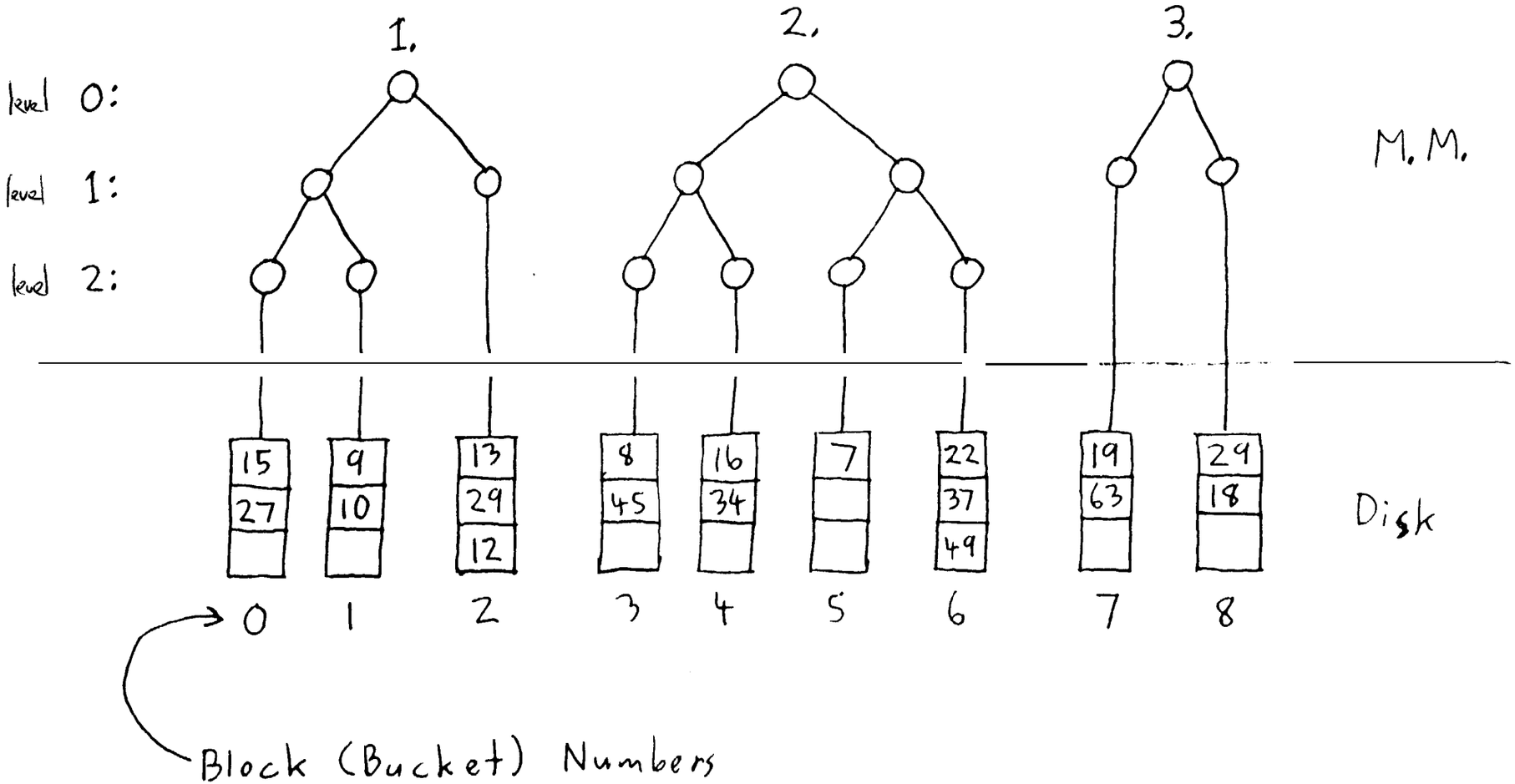
Dynamic Reorganization

- Involves a new approach to collision resolution
- No address probing, & no overflow area.
- Instead, when a bucket overflows, it is "split" into two buckets, thus adding a new bucket to the hash file.
- Issues:
  - How to split a full bucket.
  - Redistributing records between old & new buckets.
  - Minimizing file accesses during retrieval.
  - Handling record deletions (ie, "merging" buckets when they are not full enough)

- There are several methods of dynamic hash-file reorganization:
  - Virtual Hashing
  - Dynamic Hashing ← today's lecture
  - Extendible Hashing
  - Hashing

- Dynamic Hashing (Folk & Zoellick § 11.6.1)
  - Store a set of binary trees in main memory (MM).
  - Each leaf of the tree points to a bucket in the hash file.

Dynamic Hash File: Example



Implementation of Trees in M.M

- Each tree can be stored as an array of records in M.M.

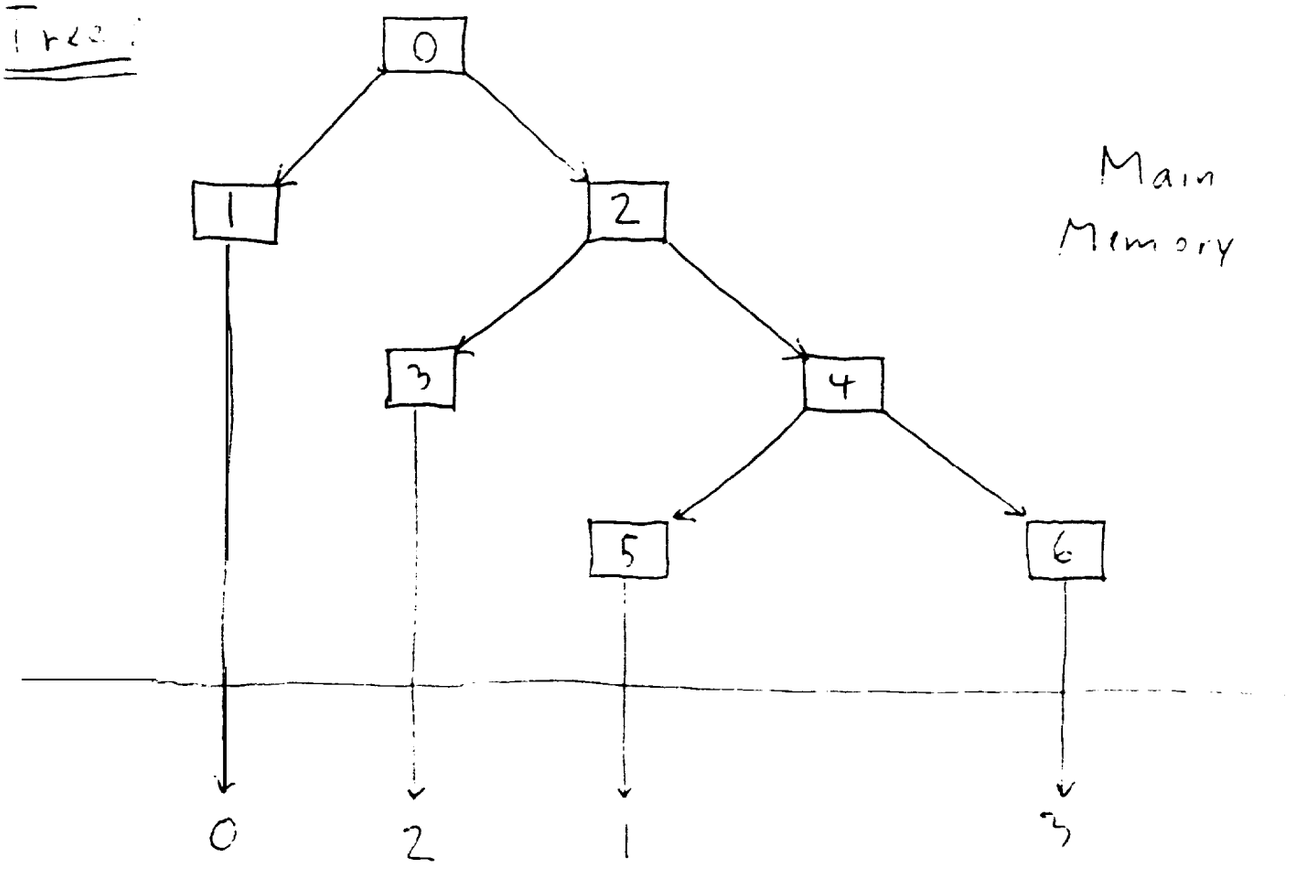
  Each record of the array represents a node of the tree.

- Records representing internal nodes store the array addresses of their children

- Records representing leaf nodes store the disk address of a bucket.

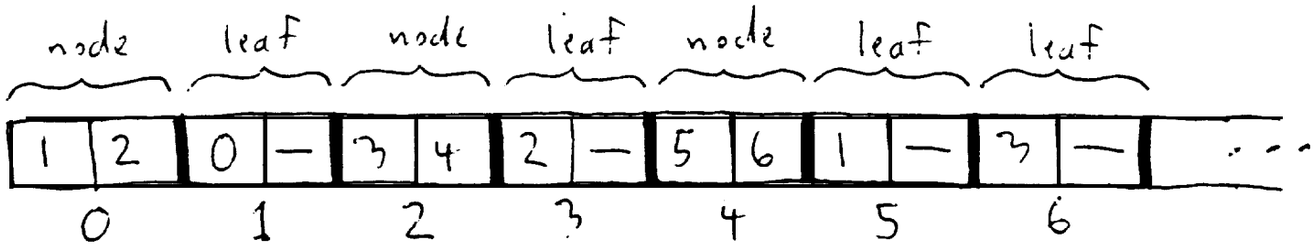
Example

Tree:



Bucket Numbers (for a file on disk)

Array:



Retrieval

To find the bucket for key k, we use two hash functions:

H(k) to identify a tree,

B(k) to specify a path through the tree from root to leaf.

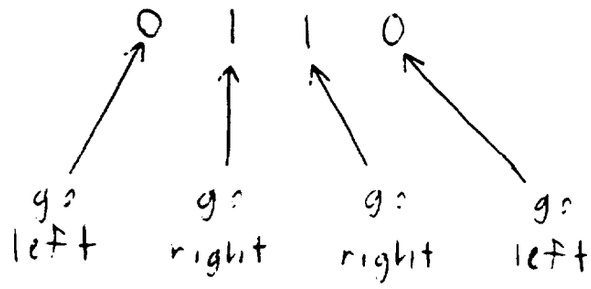
H(k) returns the MM address of a tree root.

B(k) returns a bit string that specifies a path.

- The bit string encodes a sequence of instructions for moving from the root of a tree to a leaf.

- Each 0 means go left,
- Each 1 means go right.

- eg. string:



- eg. Given $H(k) = 2$

$B(k) = 01100\ldots$

and the tree on page 14-14,

bucket 4 is retrieved.

Building a Dynamic Hash File

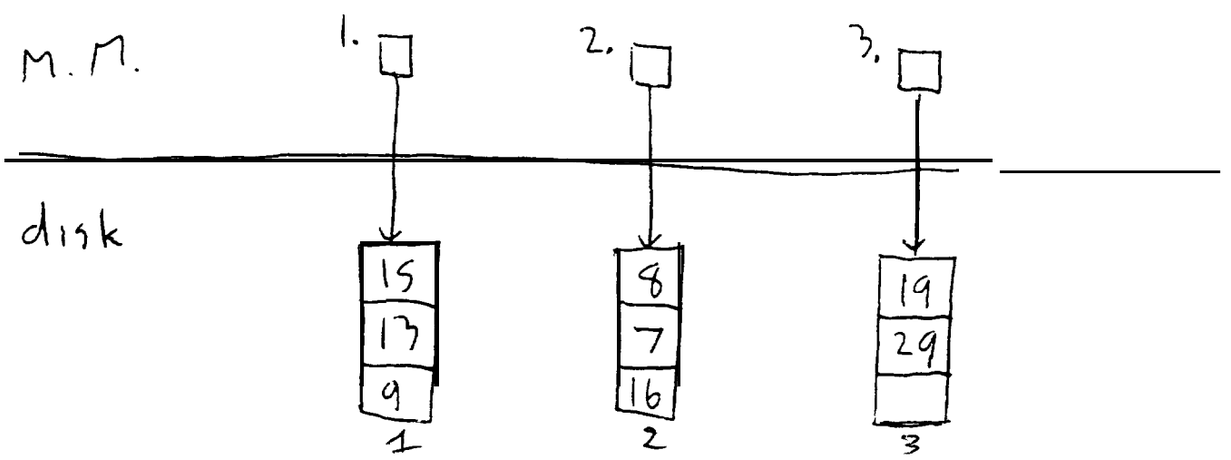
Initially, each tree is a single node.

M.M. 1. □ 2 □ 3. □

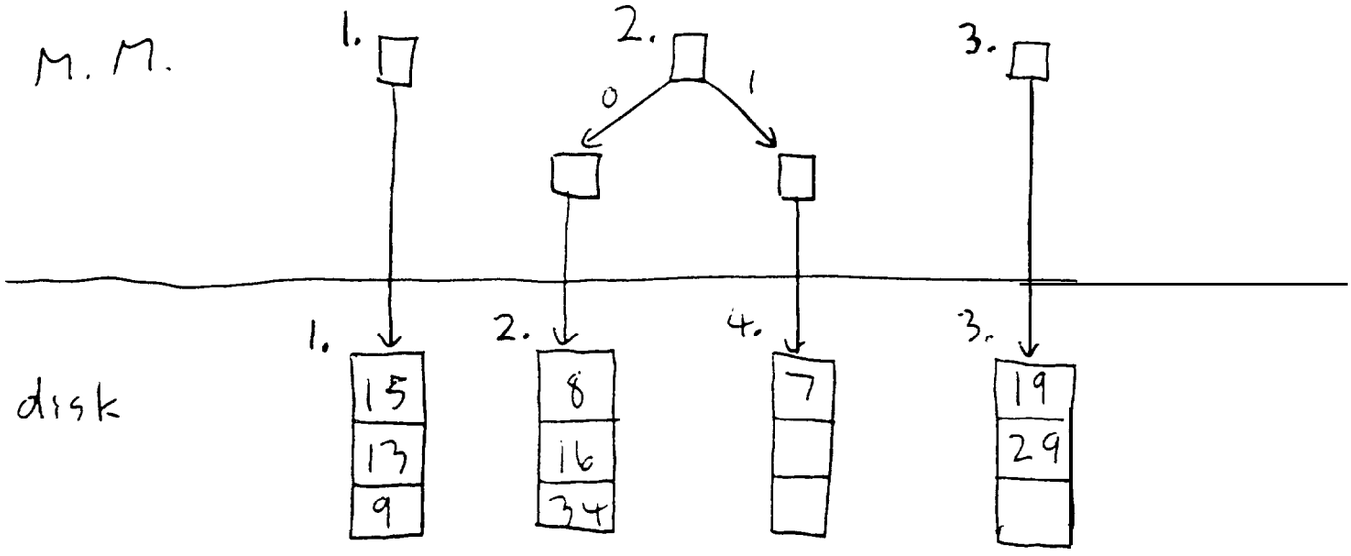
---

disk

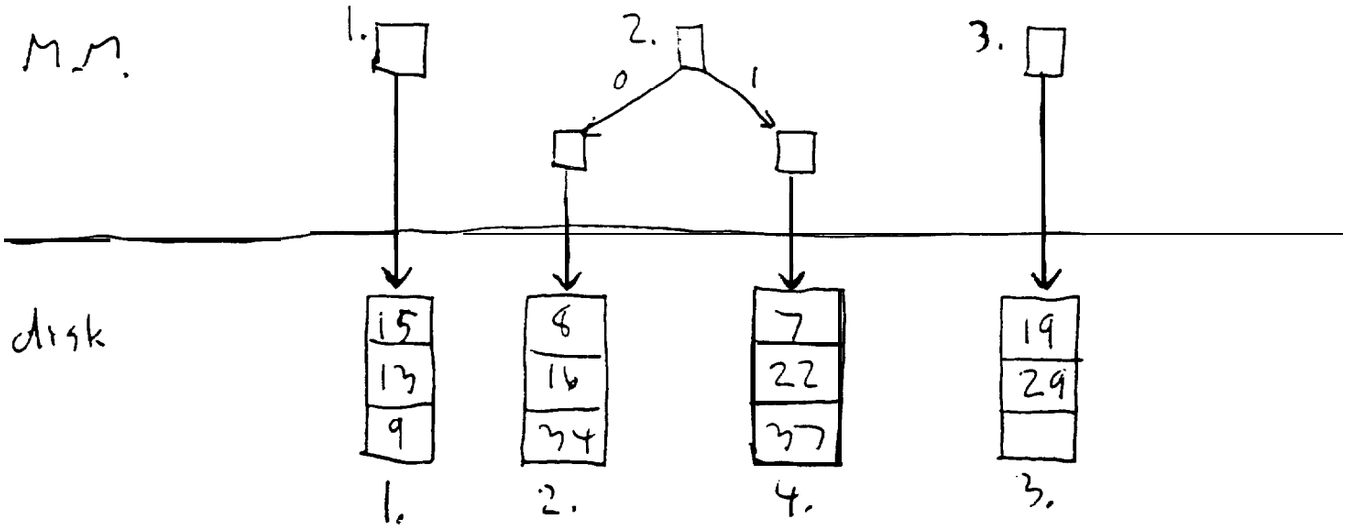
	key	H (key)	B (key)
insert:	15		0011 ...
	8	2	0001 ...
	19	3	0101 ...
	29	3	1101 ...
	7	2	1001 ...
	13	1	1000 ...
	9	1	0110 ...
	16	2	0100 ...



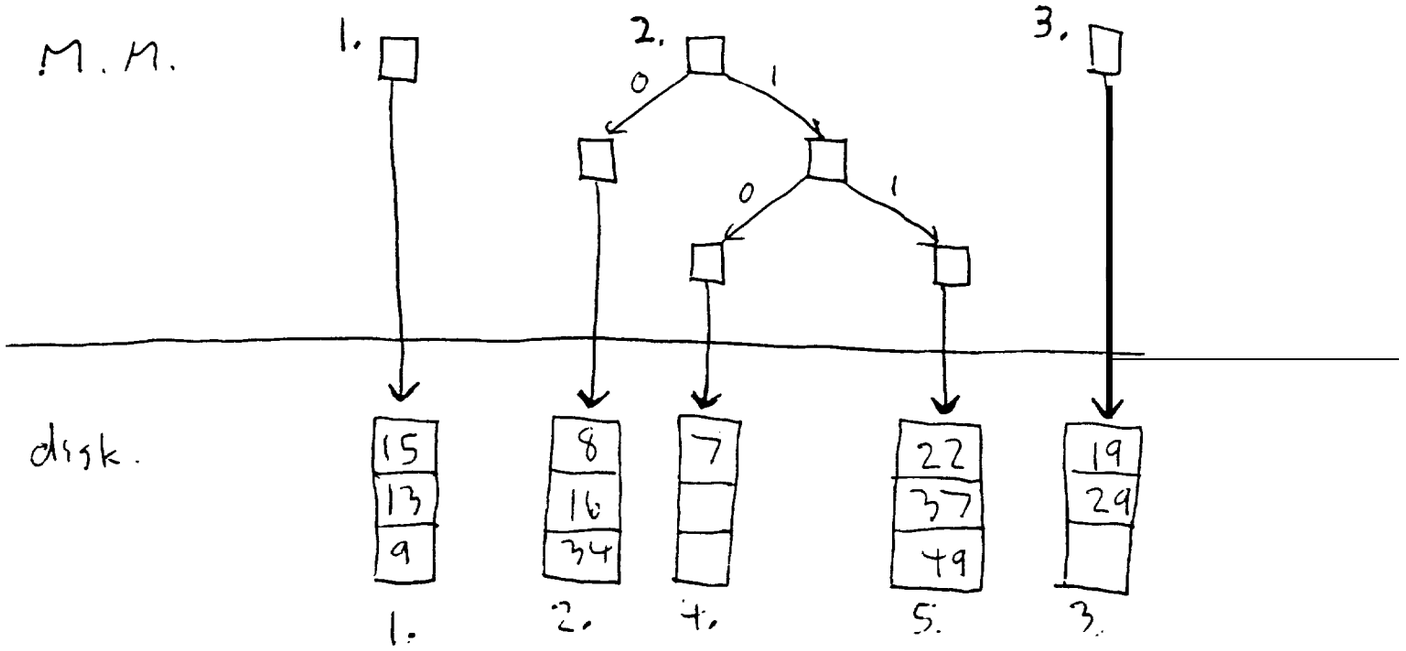
	key	H(key)	B(key)
insert	34	2	0 1 1 1 ...



	key	H(key)	B(key)
insert	22	2	1 1 1 0 ...
	37	2	1 1 1 1 ...



	key	H(key)	B(key)
insert:	49	2	1110 ...



Retrieve	H(key)	B(key)	
7	2	1001 ...	(success)
37	2	1111 ...	(success)
45	2	0001 ...	(Fail)

cost = 1 file access each.

Summary of Direct Files (i.e. hash files)

Advantages

(1) Any record can be retrieved in one or two disk accesses on average.

(2) Records can be inserted and deleted without moving other records.
(Contrast with sequential files.)

Disadvantages

(1) Need to deal with collisions.
- This has led to much research and development.
- Many collision-resolution methods.

(2) Not good for sequential retrieval (i.e., retrieving records in order of key value)
- eg. print all student records ordered by last name.

(3) Difficult to retrieve records in a given range of key values.
- eg. Print all students whose last name begins with B.