CMSC 341

Extensible Hashing

Motivations

- Another way to handle data that is too large to be stored in the primary memory
  - Most records have to be stored in disk
  - Disk read/write operations much more expensive than operations in main memory
    - e.g., disk access = 200,000 instructions (p.165)
  - Regular hash tables need to exam several disk blocks when collisions occur
  - want to limit number of disk accesses for find/insert operations

Basic Ideas

- Basic ideas:
  - Hash the key of each record into a reasonably long integer to avoid collision
    - adding 0's to the left so they have the same length
  - Build a directory
    - The directory is stored in the primary memory
    - Each entry in the directory points to a leaf
    - Directory is extensible
  - Each leaf contains M records,
    - Stored in one disk block
    - Share the same D leading digits

Directory and Leaves

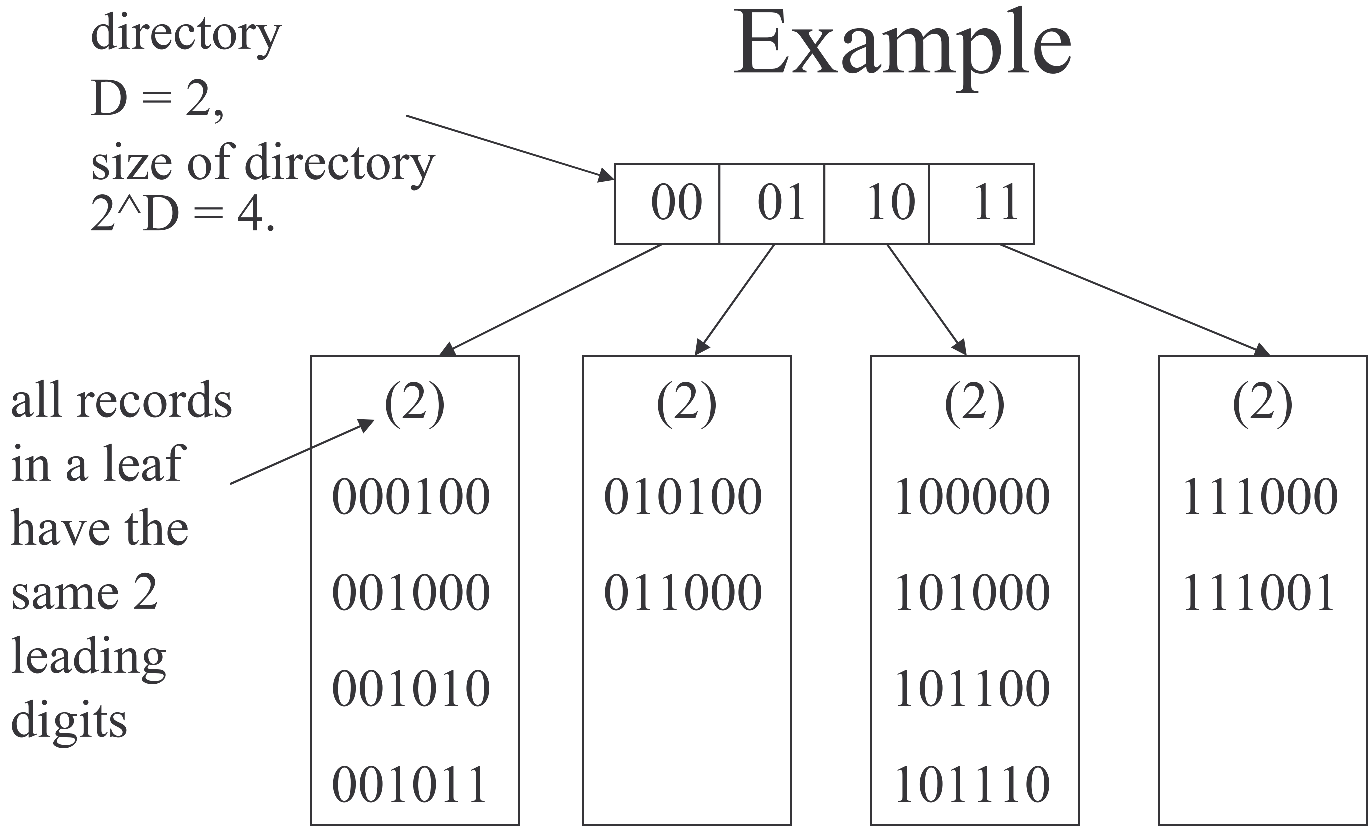
Directory: also called “root”, stored in the main memory

- D: the number of bits for each entry in the directory
- Size of the directory: 2^D

Leaf: Each leaf stores up to M elements

- M = block size /record size
- d_L: number of leading digits in common for all elements in leaf L.
- $d_L <= D$. (This will become clear shortly)

Example



N = 12, each key is an integer of 6 bits

M = 4

find operation

1. Use the first D digits of the key to find the entry in the directory;
2. Fin the address of the leaf
3. Read the leaf

Time performance:

- O(1) disk access
- Time for searching the record in the leaf in the main memory is negligible.

insert operation

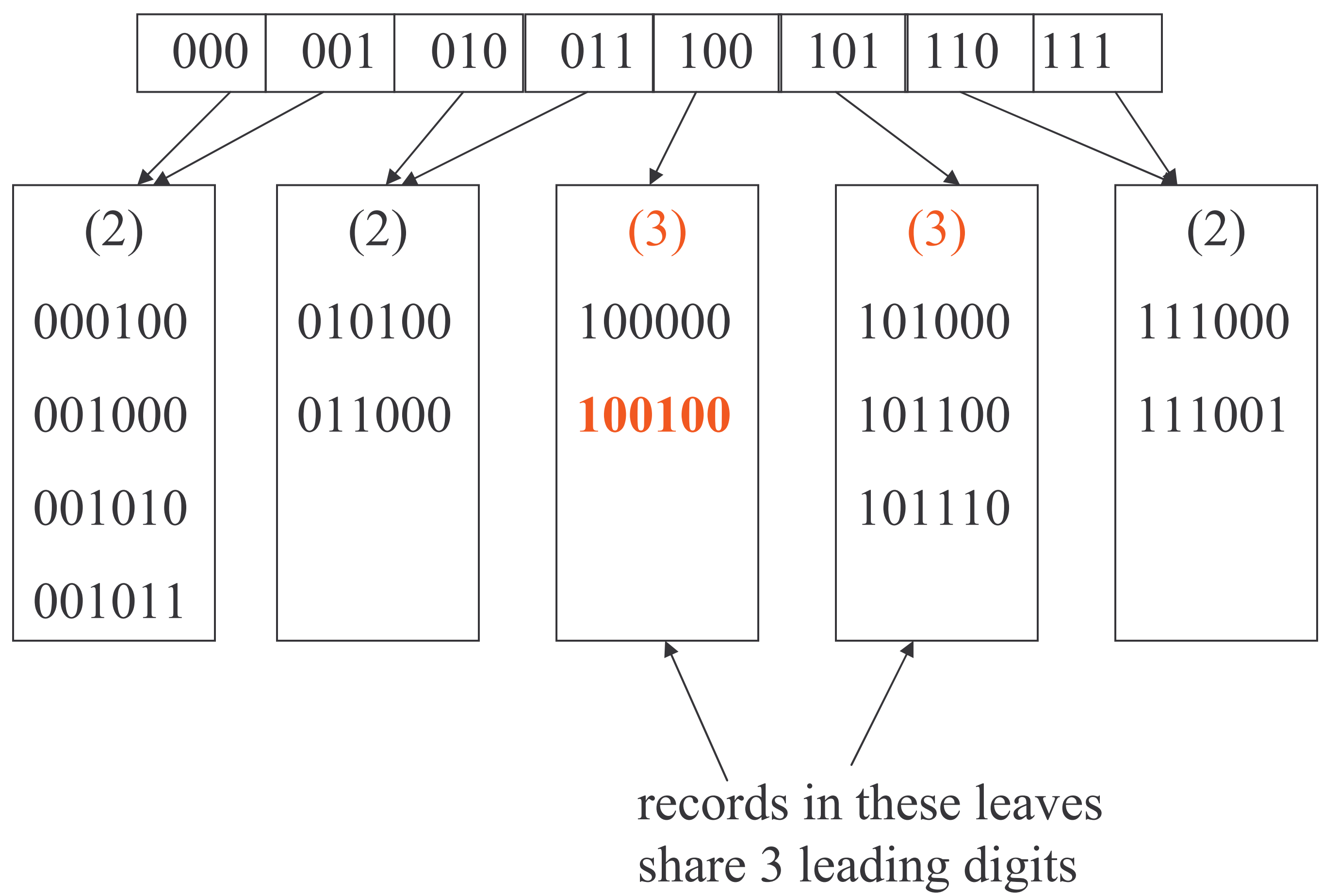
1. Find and read the leaf
2. If the leaf has room, insert the record, write back;
3. Else
   - split the leaf into two;
   - update the directory if necessary;
   - write back the leaf or leaves

insert operation

insert 100100, split (in the middle) the 3rd leaf

extend the directory (now D = 3, with8 entries)

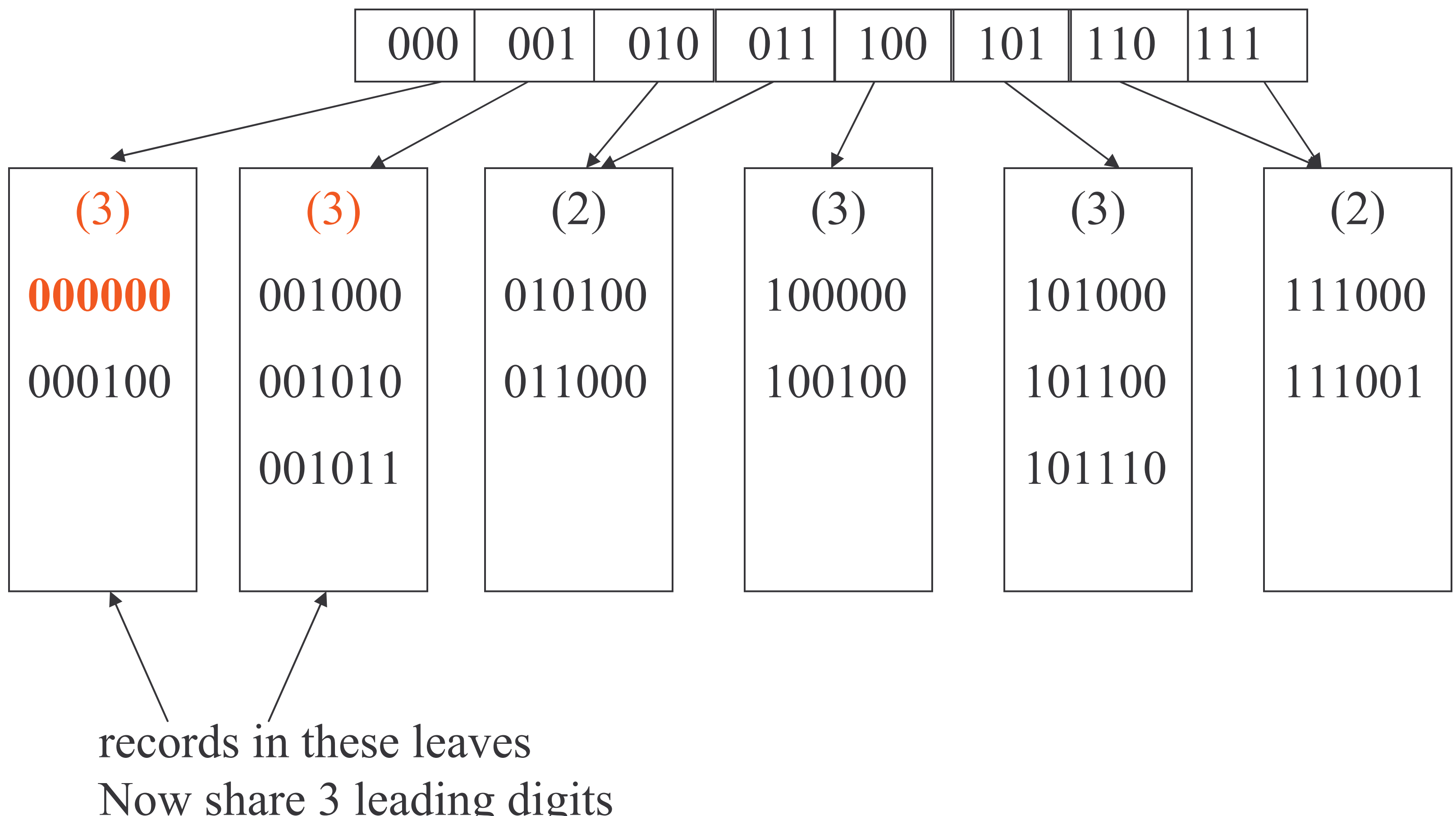
Other leaves are pointed by two adjacent directory entries



insert operation

insert 000000, split the 1st leaf

the directory is not extended since the original leaf is pointed by two directory entries



Some Issues

- Some time, more than one directory split may be needed when inserting one record.

  E.g, insert 111010, 111011, then 111100 into the original example.

(2)	(2)	(4)	(4)
111000	111000	111000	111100
111001	111001	111001	
	111010	111010	
	111011	111011	

original leaf

after inserting 111010 and 111011, No split

after inserting 11100, 2 splits, now D = 4 because 4 digits are needed to distinguish the 5 keys

Some Issues

- Duplicate keys (different original keys hashed to the same integer keys: collision)
  - Ok if fewer than M duplicates
  - Doesn't work if more than M duplicates (one directory entry cannot point to more than one page)
- Time performance
  - Expected # of leaves: $(N/M)\log_2 e$.
    - Average leaf is $\ln 2 = 0.69$ full (same as B-trees).
  - Expected size of the directory (2^D): $O(N^{1+1/M}/M)$.
    - May be large if M is small (i.e., records are large)
    - Let leaves store pointers to the records, not records themselves – adding a second disk access