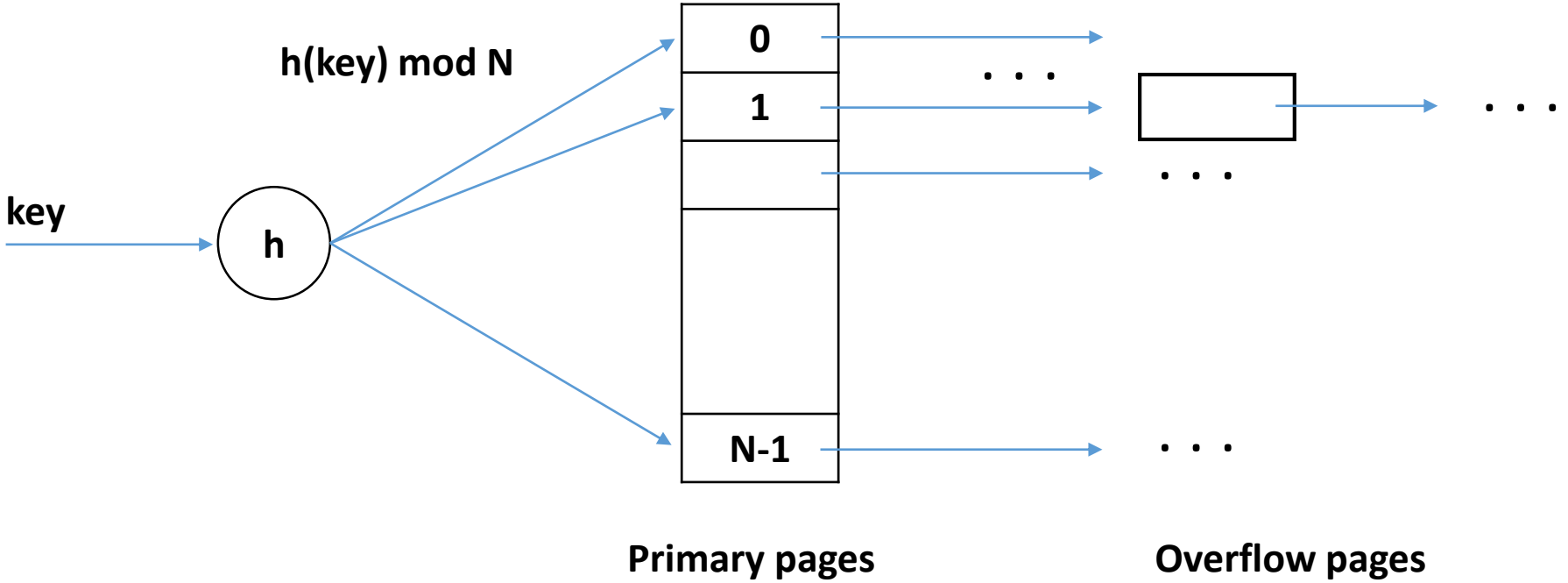
Introduction to Database Systems

Exercises: Indexing(Part 2)

Hash-based indexing

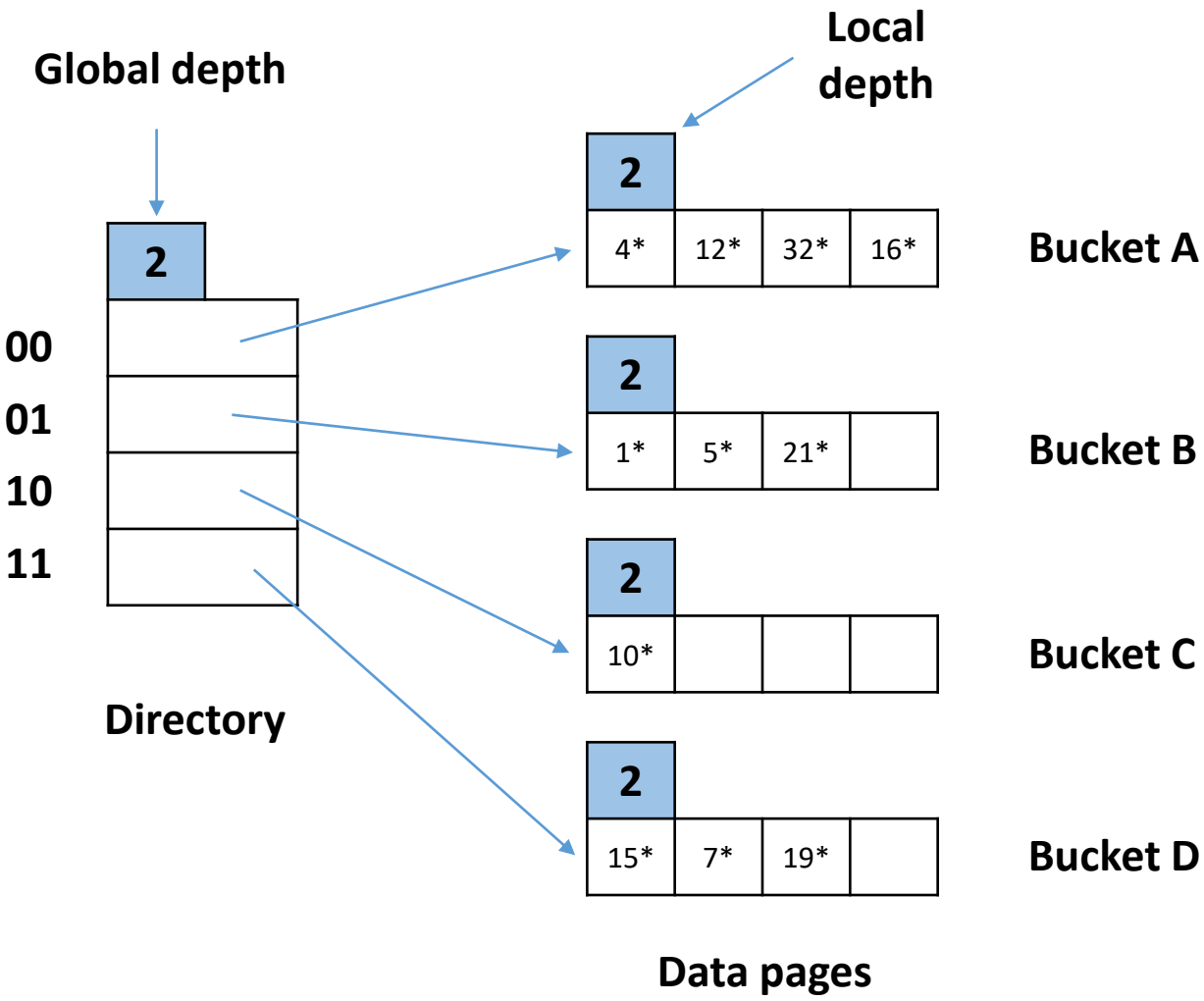
- They are excellent for **equality** selections.
  - We **do not** use it for **range** searches.
- We will have a look at **static** and **dynamic** (extendible) hash-based indexes.
  - Static indexes can lead to long overflow chains.
  - Two solutions to overcome this problem:
    - Extendible Hashing scheme, which doesn‘t use overflow pages,
    - Linear Hashing scheme, which rarely has more than two overflow pages in a chain.

Static hashing



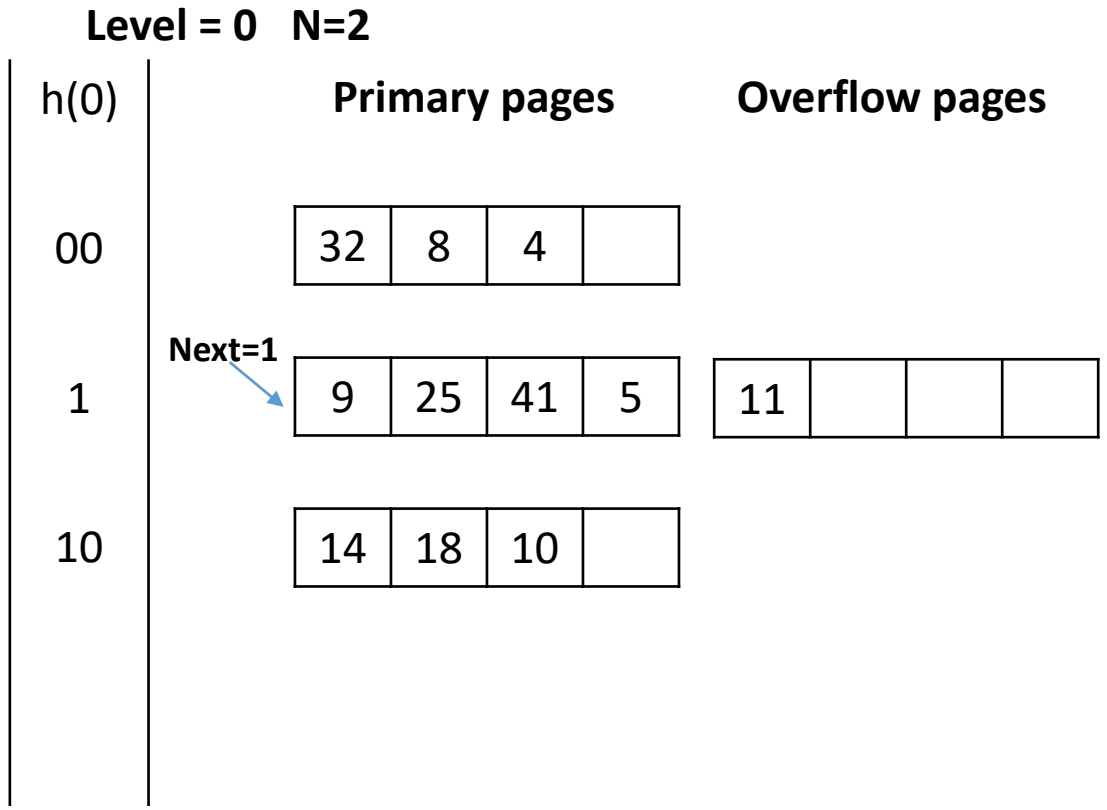
- The pages containing the data can be viewed as a collection of buckets, with one primary page and possibly additional overflow pages per bucket.
- We apply a hash function *h* to identify to which bucket the key belongs.
- The hash function must distribute values in the domain of the search field uniformly over the collection of buckets (on the interval 0 ... N-1).
  - h(k) = (a * k + b); a and b are constants
- We can get a long chain of overflow pages, resulting in poor performance.

Extendible hashing



- Directory with pointers to buckets
  - Result of a hash function tells us into which bucket an entry goes to
  - Split only the the bucket that overflows
  - Double directory if necessary
  - Doubling the directory doubles the amount of pointers
- Most of the times spliting the bucket does not demand doubling the directory
  - Compare local and global depth
  - Local depth can never exceed global depth
- How do we know to which bucket the entry belongs to?
  - Binary format of new entry
  - Last X bits tell us the appropriate bucket
    - X = global depth
  - We can also calculate modulo 2^X
    - Example: 15 mod 2^2 = 3 -> goes to the fourth bucket, because we count from 0 onwards (that means -> result of the modulo + 1)

Linear hashing

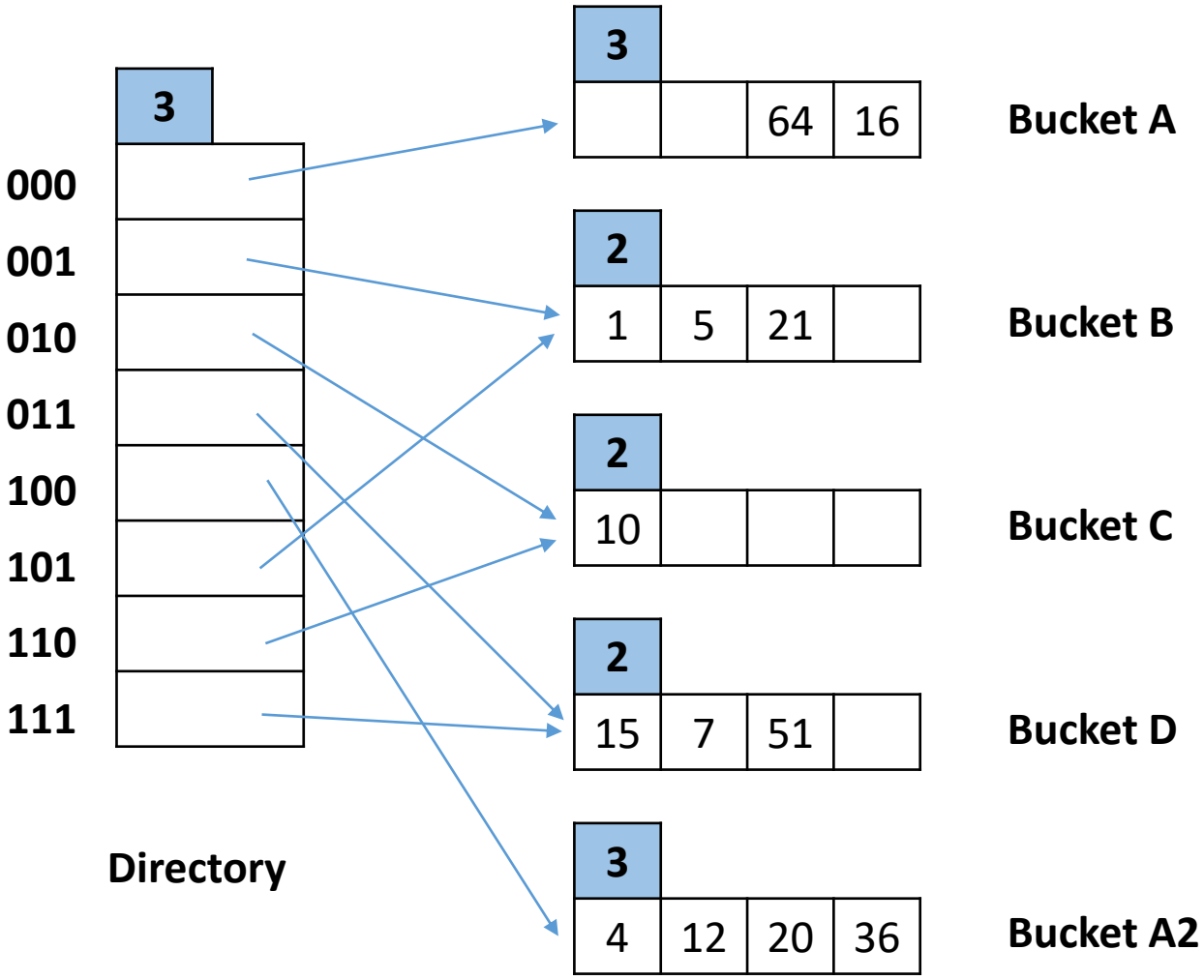


Split condition→ more than 90% occupancy

- How do we know to which bucket the entry belongs to?
  - We again check last X bits or do a modulo
  - If we don‘t find a suitable bucket, we check X+1 bits
  - X is equal to the level in this type of hashing
- If the bucket we‘re inserting into is full, we do not split the bucket, but create an overflow page
- Split of the bucket is defined by a split condition
- We do not double the amount of buckets at once, but we create new ones one by one (called a round)
- Pointer *Next* points to the bucket to be split next
  - If we split the bucket, we set *Next+1*
  - If we delete a bucket, we set *Next* to the bucket that the deleted bucket was split from
  - If directory advances a level, we set *Next = 0* (so that Next points to the first bucket)
- N represents the amount of buckets at the start of a level
- Level is increased, when Next points to the N-1th bucket and that bucket splits
  - In the example, next split would split the 2nd bucket, after which we would have 4 buckets in total, which is exactly 2N (so N was doubled)
  - When that happens => Level+1 and new N=2*N

Exercises

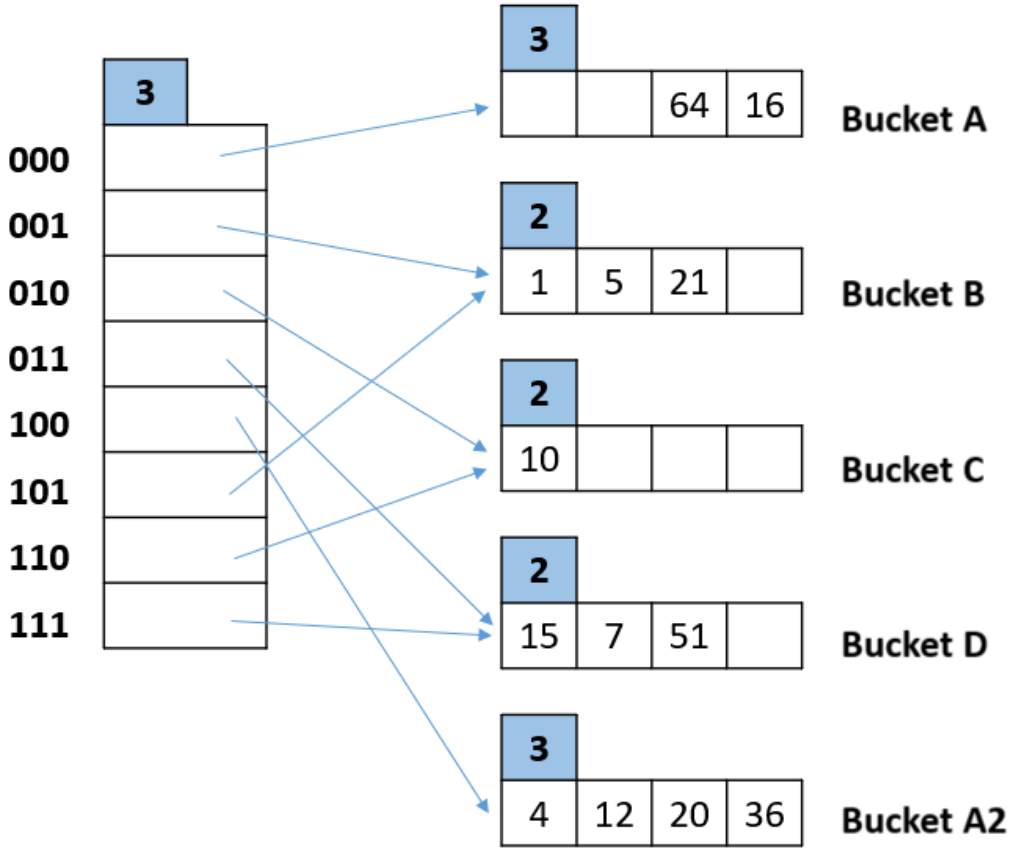
Exercise 1.1 – Extendible hash index



Consider the Extendible Hashing index shown in figure on the left.

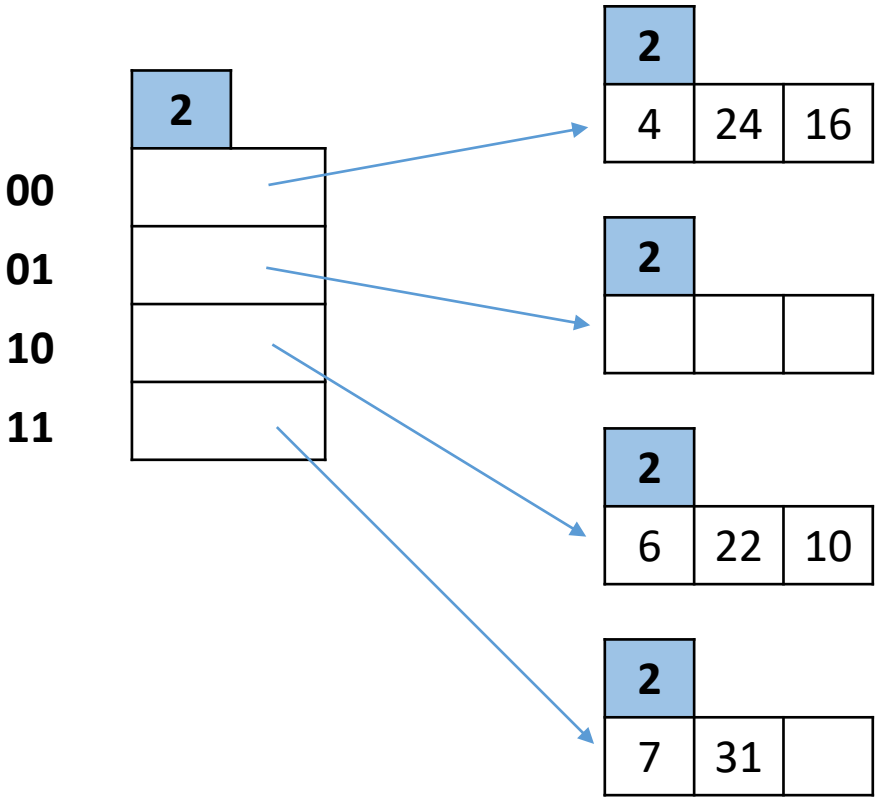
Show the index after performing the following operations:

- Insert an entry with hash value 68
- Insert entries with hash values 17 and 69 (use the original index)
- Delete an entry with hash value 21 (use the original index)
- Delete an entry with hash value 10 (use the original index)



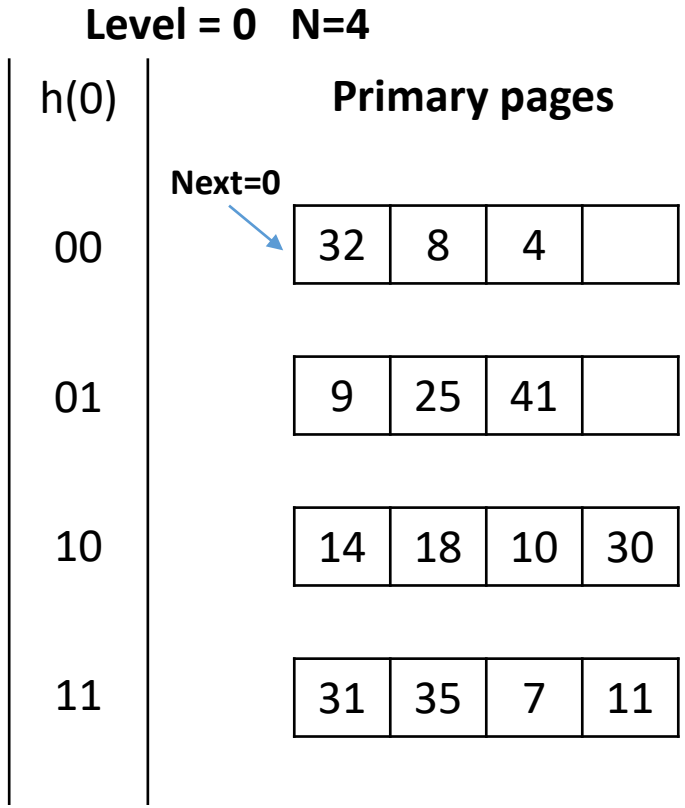
- Insert an entry with hash value 68
- Insert entries with hash values 17 and 69 (use the original index)
- Delete an entry with hash value 21 (use the original index)
- Delete an entry with hash value 10 (use the original index)

Exercise 1.2 – Extendible hash index



- Insert an entry with hash value 9, 20 and 26

Exercise 2.1 – Linear hashing index



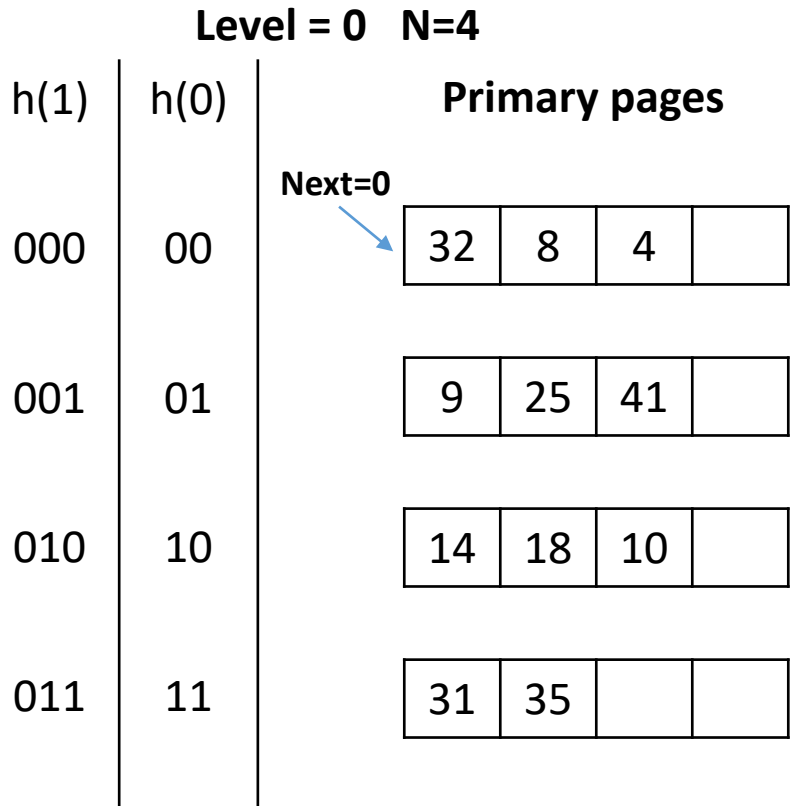
Consider the Linear Hashing index shown in figure on the left. Assume that we split whenever an overflow page is created.

Show the index after performing the following operations:

- Insert an entry with hash value 19
- Insert an entry with hash value 17
- Delete the entry with hash value 4

Split condition → when the overflow page is added

Exercise 2.2 – Linear hashing index



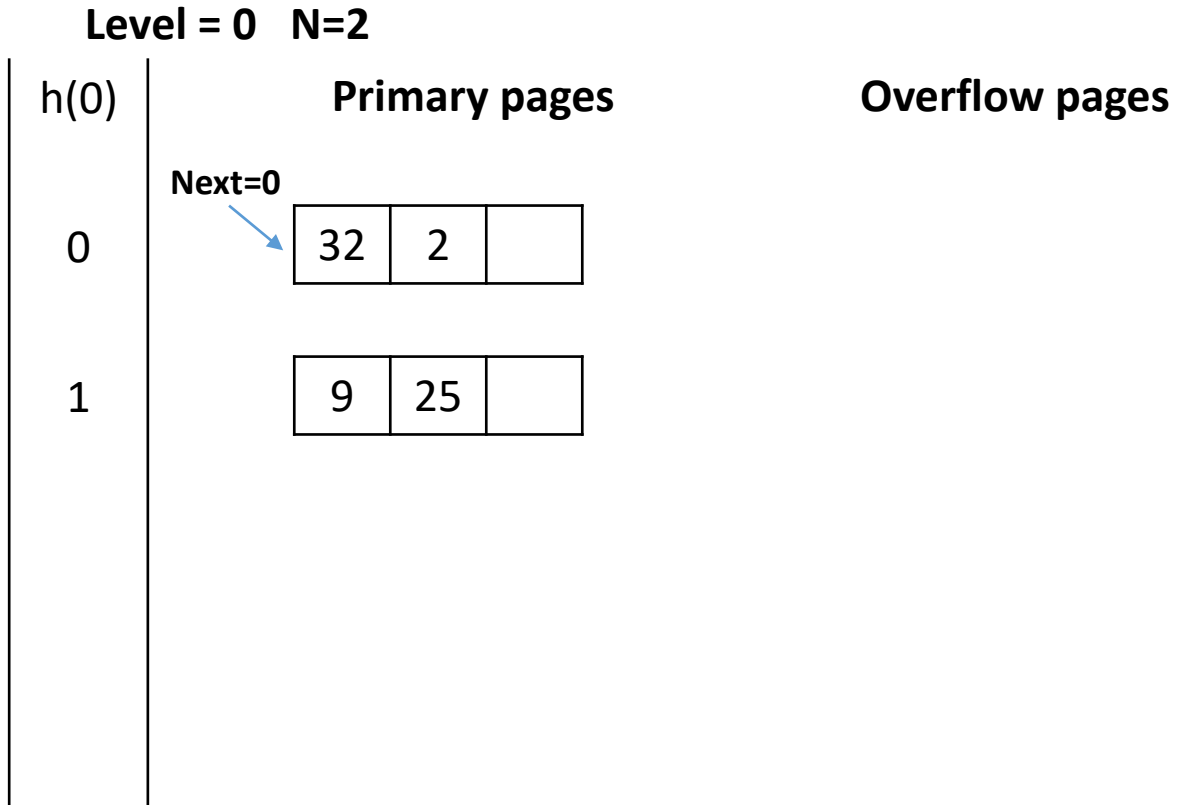
Consider the Linear Hashing index shown in figure on the left.
Show the index after performing the following operations:

- Insert an entry with hash value 7
- Insert an entry with hash value 19
- Delete the entry with hash value 4

Split condition → more than 75% occupancy

Number of items/ (number of buckets * bucket capacity)

Exercise 2.3 – Linear hashing index



Consider the Linear Hashing index shown in figure on the left.
Show the index after performing the following operations:

- Insert an entry with hash value 7
- Insert an entry with hash value 19
- Insert an entry with hash value 17

Split condition → more than 70% occupancy

Number of items/ (number of buckets * bucket capacity)