



# Static Hashing

---

*NDBI007: Practical class 3*





# Hashing

---

- ❖ Hashing is an effective method for key-value association
- ❖ In optimal situation, we need only one memory access to retrieve the values for a given key
- ❖ Nevertheless, mapping a larger domain of keys into much smaller storage leads to collisions
  - ❖ I.e., data from two different keys should be stored on the same address
- ❖ Collision can be solved in a number of different ways:
  - ❖ Separate chaining
  - ❖ Open addressing
  - ❖ *Perfect hashing*, i.e., avoiding collisions completely
    - ❖ Choosing hashing function (process) that does not create collision on a given key set

# Perfect Hashing

---

- ❖ Examples:
  - ❖ *Cormack*
  - ❖ *Larson & Kalja*
- ❖ Both methods are also members of the static hashing family
  - ❖ I.e., not designed to be used for rapidly growing number of data

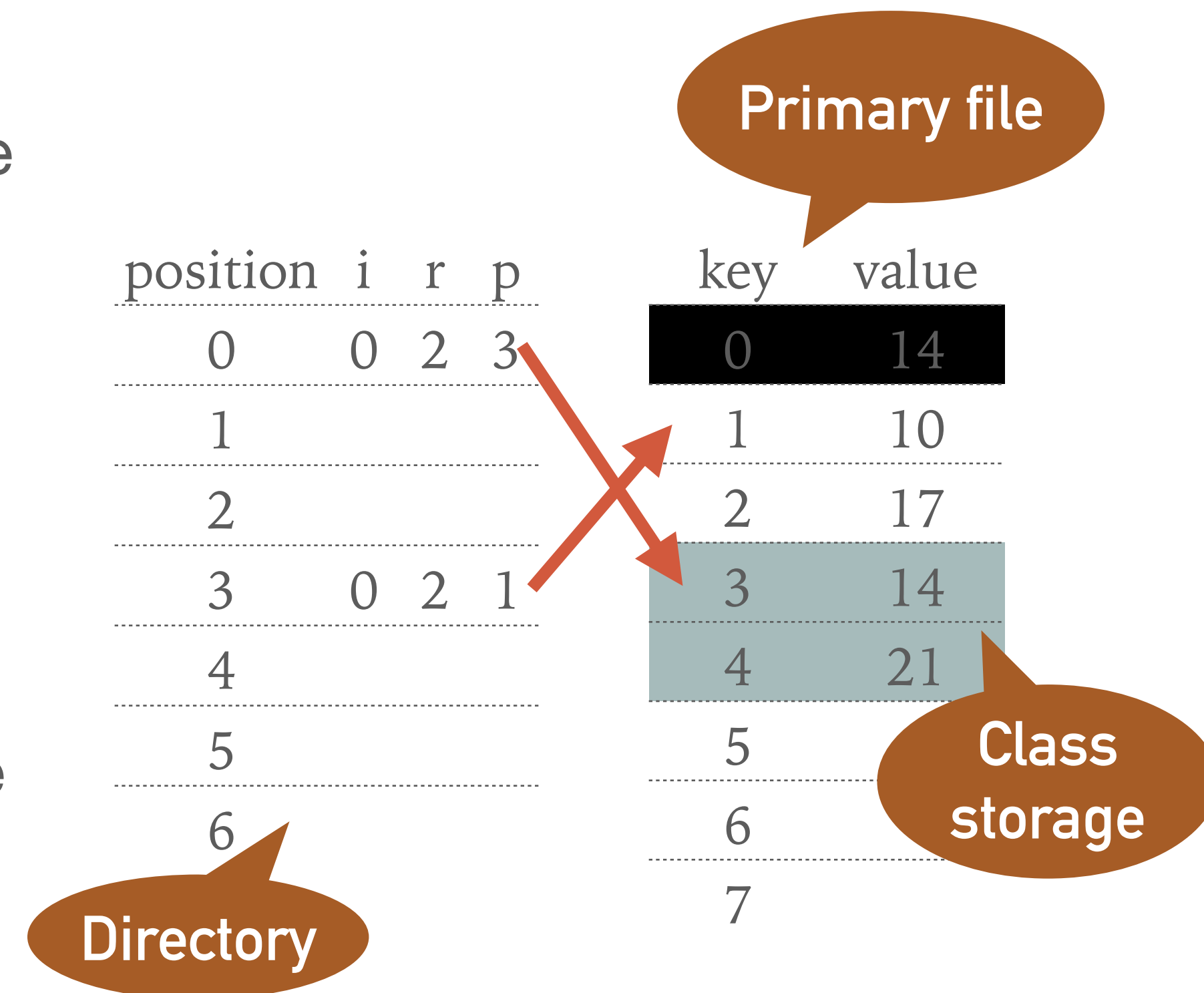
# Cormack

---

- ❖ Perfect static hashing method based on *Divide and Conquer*
  - ❖ Divide set of all records to be hashed into smaller subsets
  - ❖ Find a perfect hashing function for each small subset of records independently on each other
- ❖ *Primary hash function*  $h(k, s)$  hashes given key  $k$  into *directory* of size  $s$ 
  - ❖ E.g.,  $h(k, s) = k \bmod s$
- ❖ *Secondary hashing function*  $h_i(k, r)$  address *collisions* of the primary hashing function
  - ❖  $i$  - index of used hashing function
  - ❖  $r$  - number of referenced records in the hash table
  - ❖ E.g.,  $h_i(k, r) = (k \gg i) \bmod r$

# Cormack (Continued)

- ❖ For each directory, we have to remember its parameters:
  - ❖  $s$  - *size* of the directory, i.e., how many records can be stored there
  - ❖  $i$  - *index* of locally perfect *hashing function* to be used
  - ❖  $r$  - *number of collisions* in the primary file
  - ❖  $p$  - *pointer* to start of the primary file
- ❖ The directory has a fixed size and its change is generally not possible
  - ❖ Unless all the stored records are reinserted
- ❖ In general, when a new item (key, value) is inserted, its class storage is moved to the end of file, expanded, new  $h_i(k, r)$  is found and all the values in the storage are reinserted
- ❖ Once the class storage is ready, the record in directory is updated



$$h(k, s) \longrightarrow \text{position}$$

$$h_i(k, r) \longrightarrow \text{order}$$

# Example 3.1: Cormack

- ❖ Insert records 14, 17, and 10 into directory of size  $s = 7$ 
  - ❖ Primary hashing function is given as  $h(k, s) = k \bmod s$
  - ❖ Secondary hashing function is  $h_i(k, r) = (k \gg i) \bmod r$
- ❖ Inserting record 14
  - ❖  $h(14, 7) = 14 \bmod 7 = 0$
  - ❖ Position 0 in the directory is empty
    - ❖ Therefore we set  $i = 0, r = 1, p = 0$  1
- ❖ Inserting record 17
  - ❖  $h(17, 7) = 17 \bmod 7 = 3$
  - ❖ Position 3 in the directory is empty
    - ❖ We append a new class storage at the end of primary file 2
    - ❖ We remember parameters  $i = 0, r = 1, p = 1$  3

position	i	r	p	key	value
0	0	1	0	0	14
1				1	
2				2	
3				3	
4				4	
5				5	
6				6	
7					

position	i	r	p	key	value
0	0	1	0	0	14
1				1	17
2				2	
3	0	1	1	3	
4				4	
5				5	
6				6	
7					

# Example 3.1: Cormack (Continued)

- ❖ Inserting record 10
  - ❖  $h(10,7) = 10 \bmod 7 = 3$
  - ❖ Position 3 already contains a record (i.e., 17) for existing class storage
  - ❖ As the class storage is located at the end of the primary file, we can easily expand it **1**
  - ❖ Given class storage has now two elements, i.e.,  $r = 2$ , and starts on position  $p = 1$  **2**
  - ❖ Finally, we need to find  $i$ , i.e.,  $h_i(k, r)$  for which there will be no collision
    - ❖  $h_0(10,2) = (10 > > 0) \bmod 2 = 10 \bmod 2 = 0$
    - ❖  $h_0(17,2) = (17 > > 0) \bmod 2 = 17 \bmod 2 = 1$  **3**
  - ❖ The records in class storage are stored in order given by secondary hashing function

position	i	r	p	key	value
0	0	1	0	0	14
1				1	17
2				2	
3	0	1	1	3	
4				4	
5				5	
6				6	
				7	

position	i	r	p	key	value
0	0	1	0	0	14
1				1	10
2				2	17
3	0	2	1	3	
4				4	
5				5	
6				6	
				7	



# Example 3.2: Cormack Expanding

- ❖ Expand directory by adding record 21
- ❖  $h(21,7) = 21 \bmod 7 = 0$ 
  - ❖ Respective class storage is not located at the end of the file **1**
  - ❖ We have to move it, i.e., we set position  $p = 3$  and  $r = 2$  **2**
- ❖ Again, we need to find suitable  $i$ 
  - ❖  $h_0(14,2) = (14 > > 0) \bmod 2 = 14 \bmod 2 = 0$  **3**
  - ❖  $h_0(21,2) = (21 > > 0) \bmod 2 = 21 \bmod 2 = 1$
- ❖ Position 0 is marked as *unused space* and will be never used again as the class storage always moves on the end of the primary file **4**
- ❖ Optimization for *space reusability* could be employed\*

position	i	r	p	key	value
0	0	1	0	0	14
1				1	10
2				2	17
3	0	2	1	3	
4				4	
5				5	
6				6	
				7	

position	i	r	p	key	value
0	0	2	3	0	14
1				1	10
2				2	17
3	0	2	1	3	14
4				4	21
5				5	
6				6	
				7	

\* That is out of scope of this practical class



# Exercise 3.3

- ❖ Expand directory from example 3.2
- ❖ Insert record 28
- ❖ Primary hashing function

$$h(k, s) = k \bmod s$$

- ❖ Secondary hashing function

$$h_i(k, r) = (k \gg i) \bmod r$$

- ❖ Compute all the parameters and illustrate the directory and primary file

position	i	r	p	key	value
0	0	2	3	0	14
1				1	10
2				2	17
3	0	2	1	3	14
4				4	21
5				5	
6				6	
				7	

# Exercise 3.4

- ❖ Expand the directory from exercise 3.3 (see figure)
  - ❖ Insert record 42
  - ❖ Primary hashing function

$$h(k, s) = k \bmod s$$

- ❖ Secondary hashing function

$$h_i(k, r) = (k \gg i) \bmod r$$

- ❖ Compute all the parameters and illustrate the directory and primary file

position	i	r	p	key	value
0	0	3	3	0	14
1				1	10
2				2	17
3	0	2	1	3	21
4				4	28
5				5	14
6				6	
7				7	

- ❖ **Tip:** If you get a collision for every  $i$ , increment parameter  $r$  by 1 and try computation again



# Larson & Kalja

- ❖ The disadvantage of Cormack is the necessity of *storing the directory*
- ❖ Larson & Kalja hashing uses only a few bites instead of a directory record
- ❖ *Splits data* into pages, where each page has a *separator*
  - ❖ Record fits into certain page only when is less than the separator
    - ❖ I.e., the separator is greater than all the keys in respective page
- ❖ Pages have *limited capacity*, therefore *overflow* may occur
  - ❖ If the overflow occurs, the page *separator is updated*
    - ❖ I.e., its value is lowered
  - ❖ All the *records which do not fit* into the page any more due to the updated separator are *re-inserted*

The diagram illustrates a hash table structure with four pages. Each page is represented by a row in a table. The first column of each row is the 'Separator' value, and the subsequent columns are the 'Page number' values. The 'Separator' values are 0, 1, 2, and 3 for pages 0, 1, 2, and 3 respectively. The 'Page number' values are 10, 20, 30 for page 0; 51, 61 for page 1; 32, 37, 42 for page 2; and 100, 010, 000 for page 3. The 'Separator' values are also shown in binary (111) for each page. The 'Page number' values are shown in binary for each page. The 'Separator' values are also shown in binary (111) for each page.

0	10	20	30
111	011	110	010
1	51	61	
111	010	101	
2	32	37	42
111	100	010	000
3			
111			
4			
111			

# Example 3.5: Larson & Kalja

- ❖ Insert records 10, 20, 30, 32, 37, 42, 51, 61
- ❖ Use hash function  $h_i(k) = (k + i) \bmod 5$ 
  - ❖ To get the *number of page* in which the data should be inserted (i.e., we have 5 pages)
- ❖ Employ function  $s_i(k) = (k \gg i) \bmod 7$  to get the signatures
  - ❖  $i$  stands for the number of *previously unsuccessful inserts*
- ❖ Initial separator values are set to  $111_2$  as the maximum inserted record is  $s_i(k) = 110_2 = 6$

$$h_0(10) = 10 \bmod 5 = 0 \quad s_0(10) = 10 \gg 0 \bmod 7 = 10 \bmod 7 = 3 \sim 011_2$$
$$h_0(20) = 20 \bmod 5 = 0 \quad s_0(20) = 20 \gg 0 \bmod 7 = 20 \bmod 7 = 6 \sim 110_2$$
$$h_0(30) = 30 \bmod 5 = 0 \quad s_0(30) = 30 \gg 0 \bmod 7 = 30 \bmod 7 = 2 \sim 010_2$$
$$h_0(32) = 32 \bmod 5 = 2 \quad s_0(32) = 32 \gg 0 \bmod 7 = 32 \bmod 7 = 4 \sim 100_2$$
$$h_0(37) = 37 \bmod 5 = 2 \quad s_0(37) = 37 \gg 0 \bmod 7 = 37 \bmod 7 = 2 \sim 010_2$$
$$h_0(42) = 42 \bmod 5 = 2 \quad s_0(42) = 42 \gg 0 \bmod 7 = 42 \bmod 7 = 0 \sim 000_2$$
$$h_0(51) = 51 \bmod 5 = 1 \quad s_0(51) = 51 \gg 0 \bmod 7 = 51 \bmod 7 = 2 \sim 010_2$$
$$h_0(61) = 61 \bmod 5 = 1 \quad s_0(61) = 61 \gg 0 \bmod 7 = 61 \bmod 7 = 5 \sim 101_2$$

0	10	20	30
111	011	110	010
1	51	61	
111	010	101	
2	32	37	42
111	100	010	000
3			
111			
4			
111			



# Example 3.6: Larson & Kalja – Split Page

- ❖ Insert record 40 and split a page
  - ❖  $h_0(40) = 40 \bmod 5 = 0$     $s_0(40) = 40 > 0 \bmod 7 = 40 \bmod 7 = 5 \sim 101_2$
  - ❖ Page 0 is already full
  - ❖ We sort all the records (including newly added record) according to the signature
  - ❖ We select the item having the biggest signature
    - ❖ In our particular case, the biggest signature belongs to 20
  - ❖ We update page separator to 110 (signature of 20) **1**
  - ❖ Record 20 gest out of the page
  - ❖ We insert record 40 into page 0 **2**
- ❖ As the next step, we have to reinsert record 20
  - ❖  $h_0(20) = 20 \bmod 5 = 0$     $s_0(20) = 20 > 0 \bmod 7 = 20 \bmod 7 = 6 \sim 110_2$
  - ❖ Again, we should put record 20 into page 0, but we cannot as the page separator is smaller or equal to the signature
  - ❖ We increase  $i$  and we try to reinsert record 20 once again
  - ❖  $h_1(20) = (20 + 1) \bmod 5 = 1$     $s_1(20) = (20 > 1) \bmod 7 = 3 \sim 011_2$  **3**

	0	10	40	30
<b>1</b>	110	011	101	010
	1	51	61	20
	111	010	101	011
	2	32	37	42
	111	100	010	000
	3			
	111			
	4			
	111			

# Exercise 3.7

- ❖ Apply Larson & Kalja method to insert record 41 into the structure from example 3.6
  - ❖ Note all the computations and illustrate the result
- ❖ Tip: In some cases, we can split multiple pages on a single insert

0	10	40	30
110	011	101	010
1	51	61	20
111	010	101	011
2	32	37	42
111	100	010	000
3			
111			
4			
111			



# Exercise 3.8

- ❖ Apply Larson & Kalja method to insert record 67 into the structure from exercise 3.7 (see figure)
- ❖ Note all the computations and illustrate the result
- ❖ Tip: If one page contains more records with the same signature and we need to split this page, then we may reinsert more than just a single record

0	10	40	30
110	011	101	010
1	51	61	20
110	010	101	011
2	32	37	42
110	100	010	000
3	41		
111	011		
4			
111			

# Summary

---

- ❖ Larson & Kalja method does not have to store the item's signature as its computation is often straightforward
- ❖ The whole directory consists of  $M \cdot d$  bits, where  $M$  is a number of pages and  $d$  is a separator size (in bits)
- ❖ Thanks to the smaller size, the directory should fit into primary memory (RAM)
- ❖ In contrast to Cormack, we have to sequentially scan a page (class storage) to get the value for a given key
- ❖ Both methods require appropriate selection of the primary and secondary hashing functions