

Dynamic Hashing

NDBI007: Practical class 4

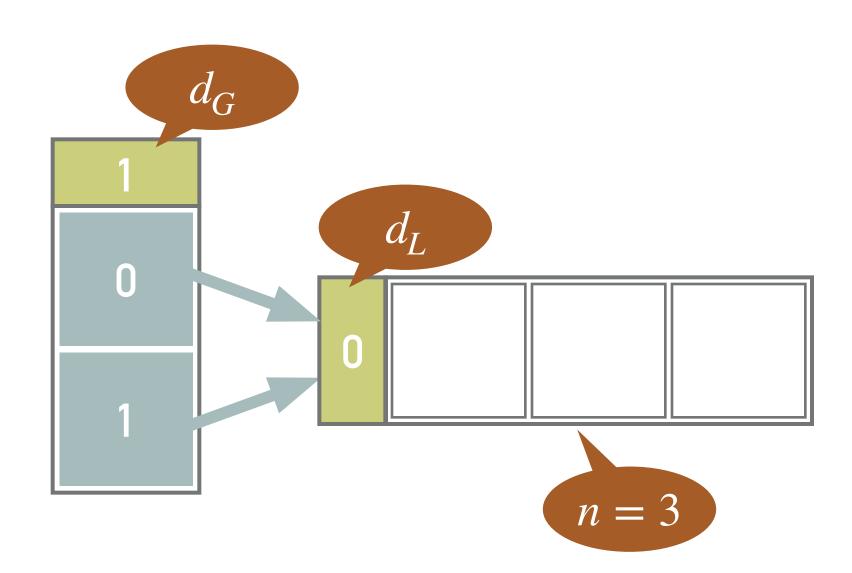
Dynamic Hashing

- Static forms of hashing lose its good performance as the table utilization comes to its maximum
- * Conversely, dynamic hashing algorithms allow to increase the size of the table with increasing number of stored records

- * Fagin
- * Litwin
- * LHPE-RL

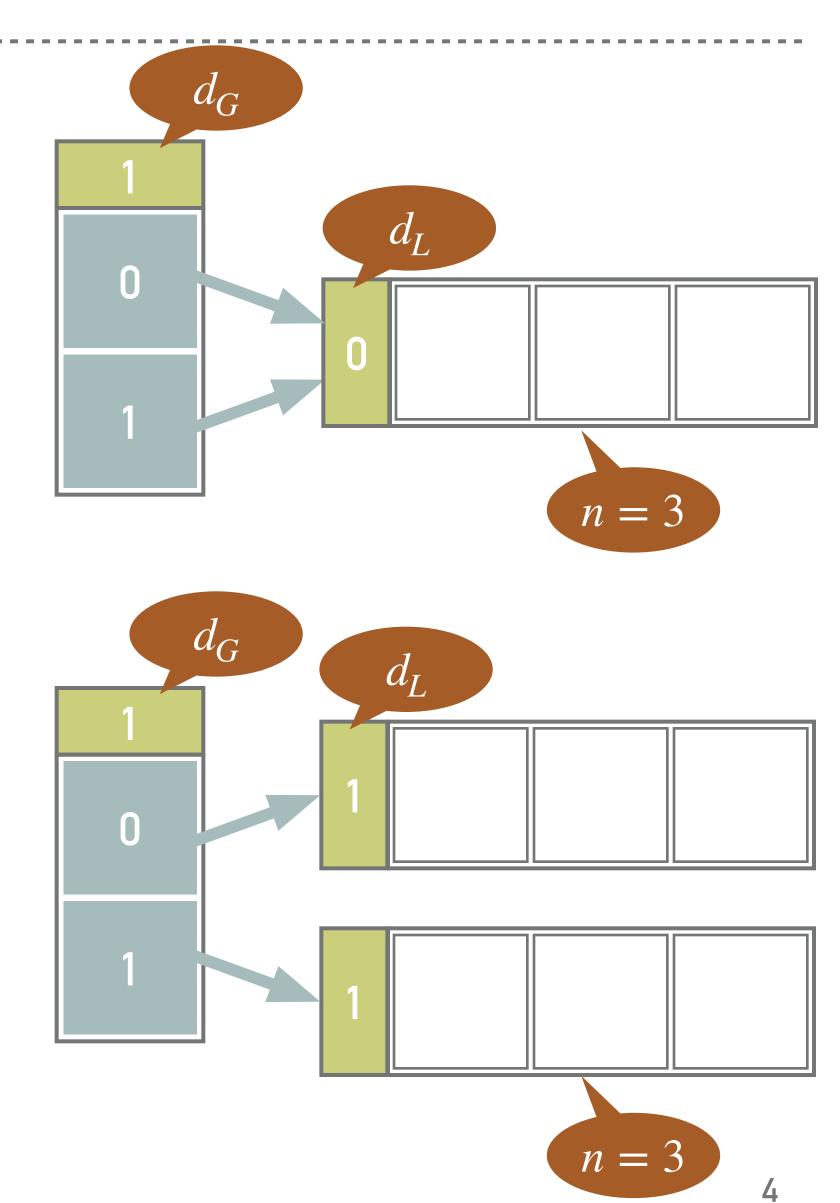
Fagin

- Directory
 - * List of entries in the main memory that points to the pages in the primary file
 - * Global depth d_G number of least significant bits of the hash h(k) needed to address an entry in the directory
- Primary file
 - * Distributed collection of pages stored in the secondary memory, i.e., continuous space is not required
 - * Each page has a constant size *n*
 - * Each page remembers local depth d_L number of least significant bits of the hash h(k) common to all records
 - * $2^{d_G-d_L}$ tells how many directory entries points to the particular page in the primary file



Fagin

- * Overflowing causes a change in the structure of the directory and primary file
 - * $d_L < d_G$ the particular page can be split, i.e., the page is split and d_L incremented
 - * $d_L=d_G$ the *directory* must be *expanded*, i.e., the directory is doubled and d_G incremented
- * Inserting or searching for a record with key k
 - * Compute k' = h(k)
 - * Convert k' into directory entry k'' by leaving the d_G least significant bits
 - * The pointer in the corresponding entry points to the page where the record should be inserted / searched

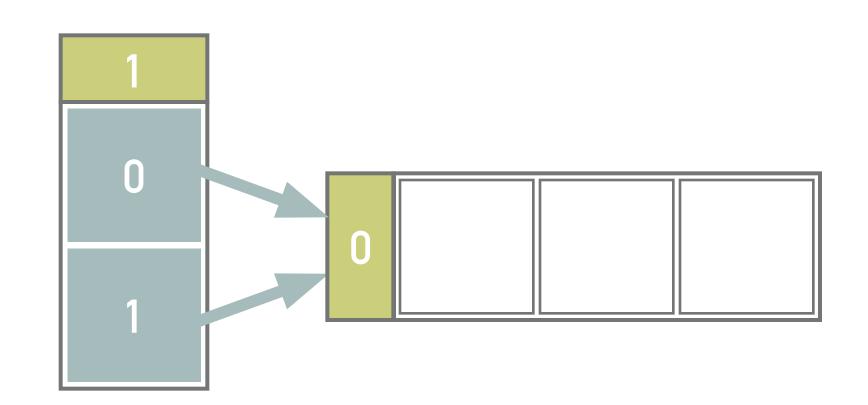


Example 4.1

Insert records with keys 20, 11, and 8

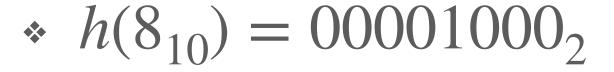
*
$$h(20_{10}) = 00010100_2$$

 Using the least significant bit of key 20, that is 0, the corresponding record is inserted into the page using entry 0

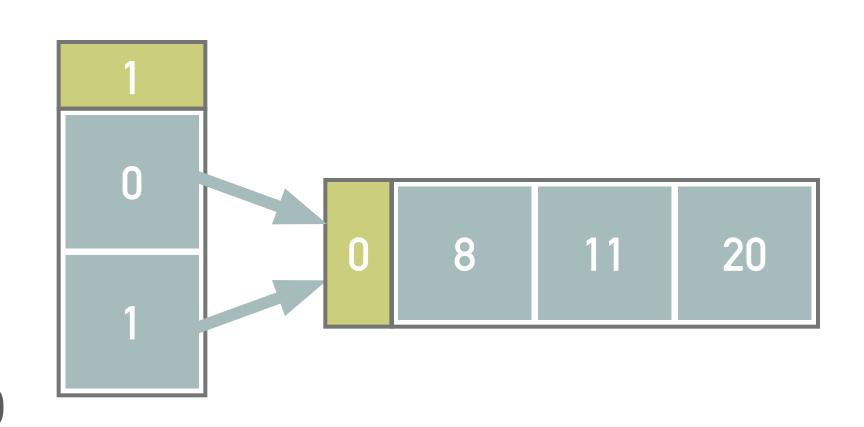


*
$$h(11_{10}) = 00001011_2$$

* Record with key 11 is stored to the same page using entry 1



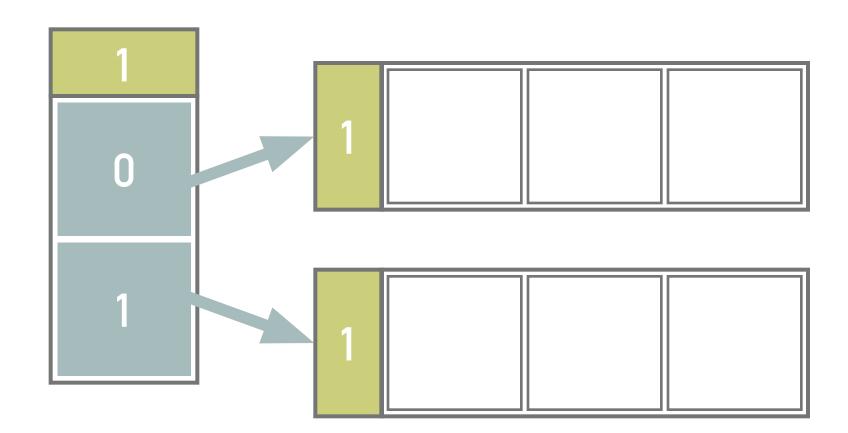
* Record with key 8 is inserted into the same page using entry 0

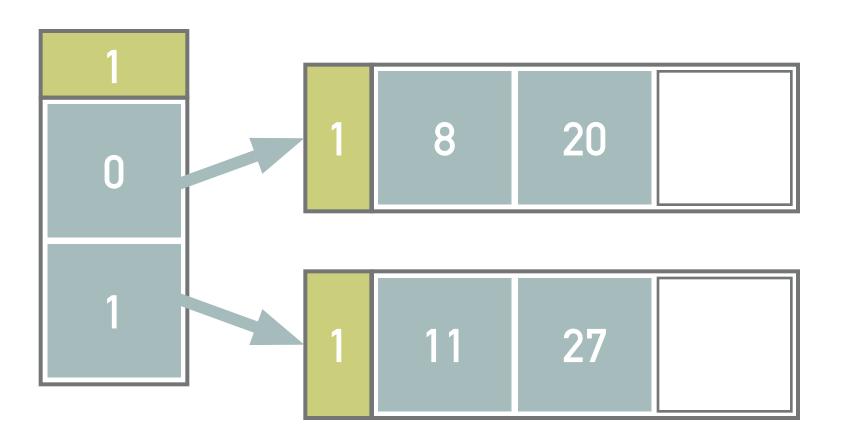


Example 4.2: Splitting a Page

* Insert record with key 27 (into the structure from example 4.1)

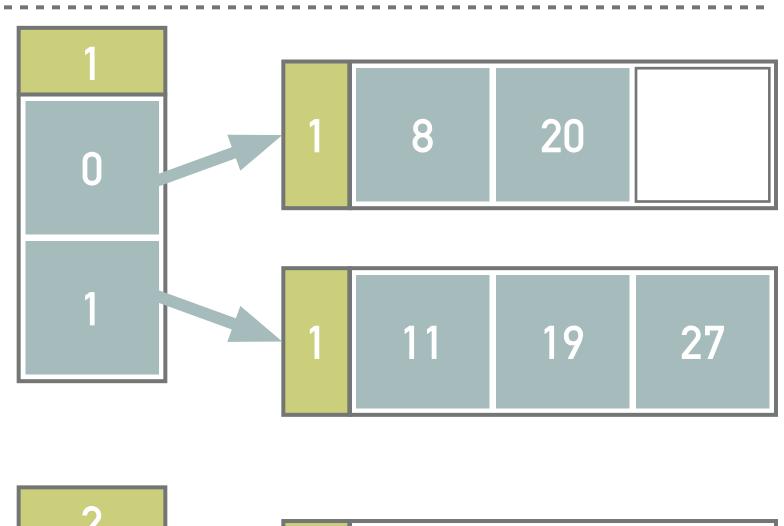
- * $h(27_{10}) = 00011011_2$
- Page is overflown
 - * The local value d_L of the page is less than the global value d_G of the directory
 - * Therefore we can split the page into two new pages and increment d_{L} values of both the pages
- Finally, we reinsert the records previously allocated into the page being split
 - * After the reinsert, the even keys are stored in the page referenced from the zero-th directory entry while the off records are referenced from the first entry

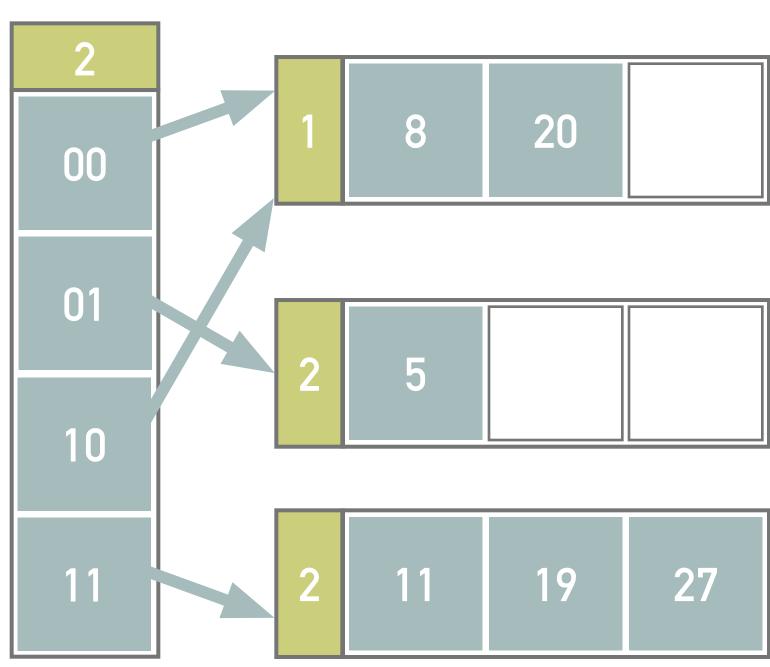




Example 4.3: Expanding the Directory

- * Insert records with keys 19 and 5 into the structure from example 4.2
- * $h(19_{10}) = 00010011_2$
 - * After inserting record with key 19, a page is filled
- * $h(5_{10}) = 00000101_2$
 - * The insert of the record having key 5 causes:
 - * Expanding the directory, i.e., $d_L = d_G$
 - * Splitting of the second page, i.e., $d_L = 2$
 - * Reinserting of records with keys 5, 11, 19, and 27
- * $h(11_{10}) = 00001011_2$
- * $h(19_{10}) = 00010011_2$
- * $h(27_{10}) = 00011011_2$

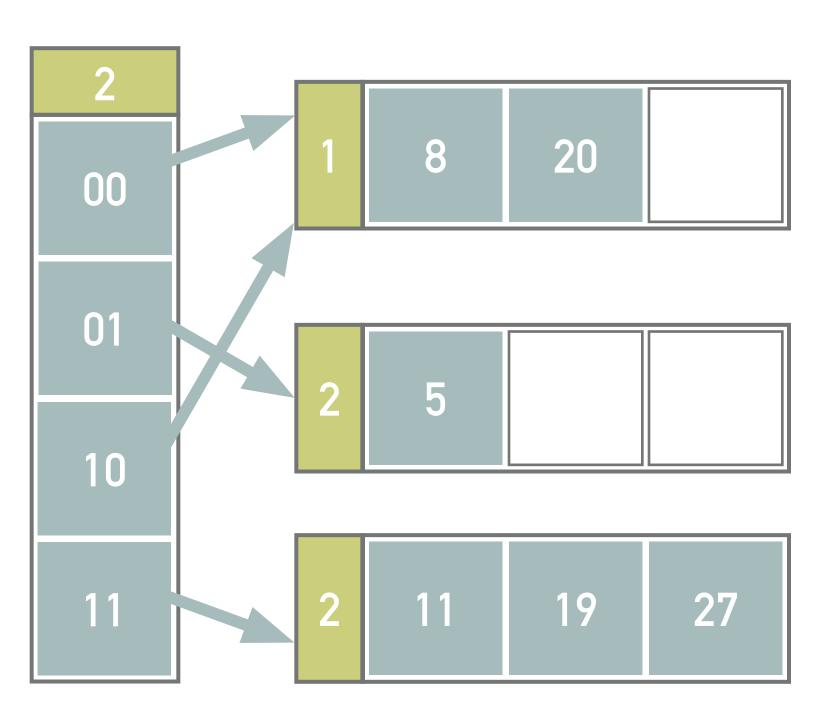




Exercise 4.4

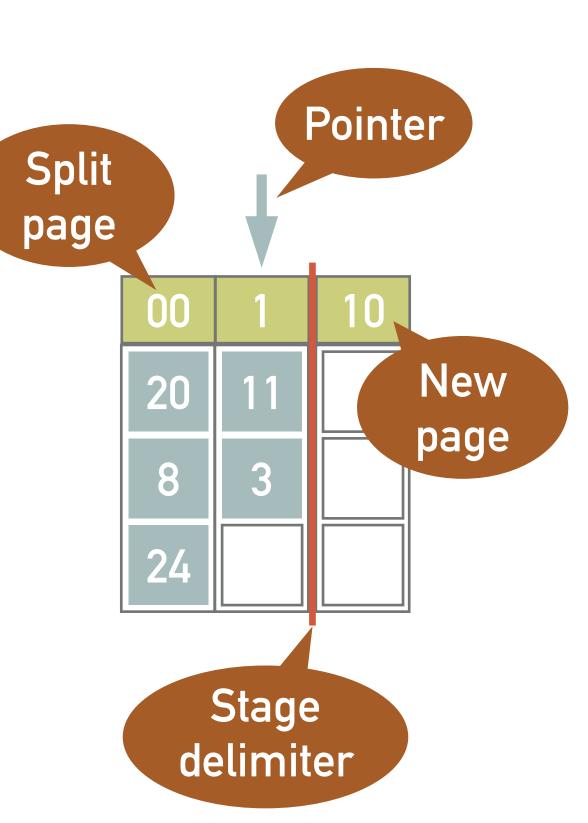
 Insert records with keys 28 and 44 into the following structure

Note all the computations and illustrate the solution



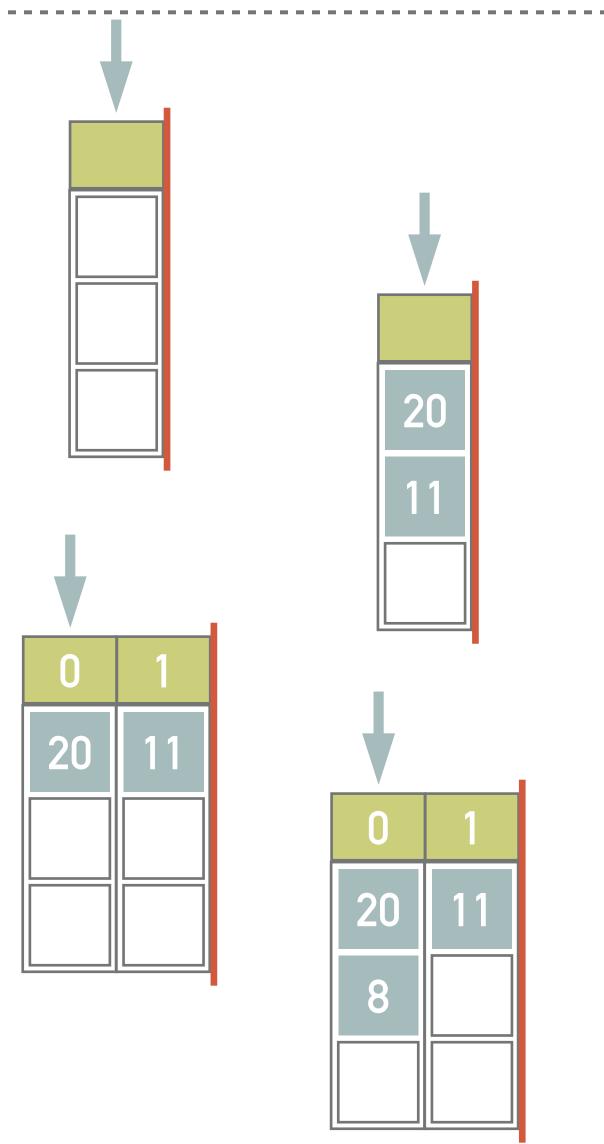
Litwin

- * Directory-less schema that avoids *exponentially increasing* size of the directory, but we need a continuous space in the secondary memory
 - Addition of a single page after pre-defined condition
- * The primary file is *linearly expanded* with time (stages), i.e., adding one page after another
 - * Stage d starts with $s=2^d$ pages and ends when the number reaches $s=2^{d+1}$ (i.e., stage d+1 begins)
- * During the stage, a *split pointer* $p \in \{0,...,2^d-1\}$ identifies the next page to be split
 - * At the beginning of stage d, the pointer points to page 0
 - * After every split operation, the *pointer is incremented* by 1, or moved to the start when we enter a new page
 - * Records from page p (and overflow records) will be distributed between split pages p and p+s using $h_{d+1}(k)$
 - * If a page overflows before its time to split, overflow page will be utilized
- * At each stage, we have two types of hash functions
 - * $h_d(k)$ for pages not yet split, i.e., the least significant d bits of the hashed value h(k) are used
 - * $h_{d+1}(k)$ for the already split pages



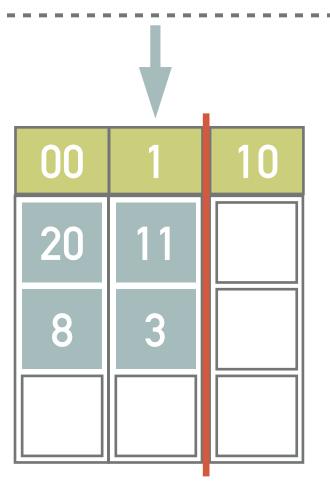
Example 4.5

- * Insert records with keys 20, 11, and 8 into an empty primary file
 - * I.e., start the stage d=0 with one page (capacity 3 records), h(k)=p, p=0
 - * Pre-defined condition: Splitting occurs after 2 inserts
- * The records with key 20 and 11 are inserted into the 0th page disregarding the value of the key
 - * d = 0 bits of the keys are used at this points
- * We have inserted 2 keys, therefore splitting occurs (a new page is created)
 - * The records from 0th page are redistributed using the least significant bit of the hashed key
 - $* h(20_{10}) = 00010100_2$
 - * $h(11_{10}) = 00001011_2$
 - * Because $p = 2^i$ is reached, the stage changes to d = 1, p = 0
- * Now, we use d=1 bit for not yet split pages and d+1 bits for split pages
 - * The record with key 8 is inserted into the page 0 using the least significant bit
 - * $h(8_{10}) = 00001000_2$

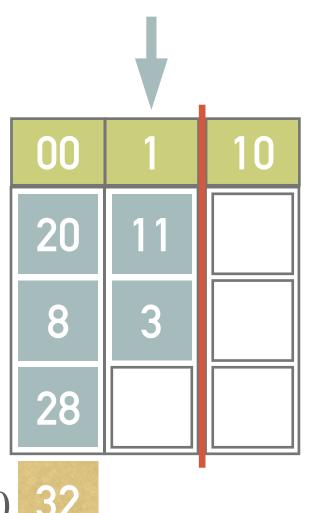


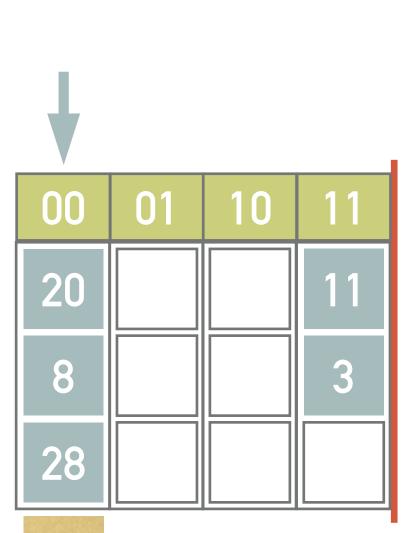
Example 4.6

- * Insert records with keys 3, 28, and 32 into the structure from example 4.5
- A record with key 3 will now be inserted into page 1
 - * $h(3_{10}) = 00000011_2$
- * We have already inserted 2 records in the stage d=1, therefore page p=0 is split into pages $p_0=00$, $p_1=10$
- * Next, we will insert a pair of records with keys 28 and 32
- * $h(28_{10}) = 00011100_2$
 - * Because $h_1(00011100_2)=0$ and 0 < p, it is necessary to address the keys using 2 least significant bits, i.e., $h_2(00011100_2)=00$, and the key belongs in the page 00
 - * $h(32_{10}) = 00100000_2$
 - The key 32 belongs to the same page, but that is already filled and thus overflows
 - Finally, the page 1 is split
 - * Since the number of pages reaches $s = 2^{1+1} = 4$, the second stage is initiated, i.e., d = 2, p = 0



00	1	10
20	11	
8	3	
28		

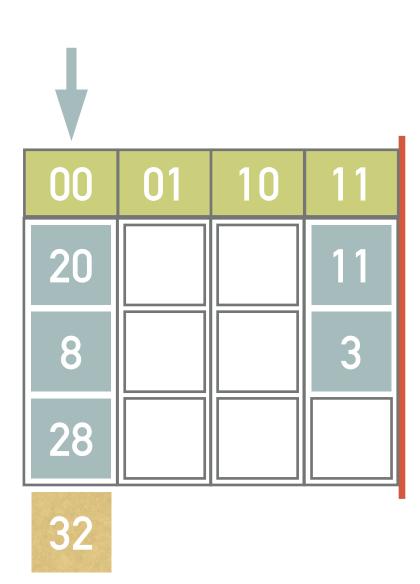




Exercise 4.7

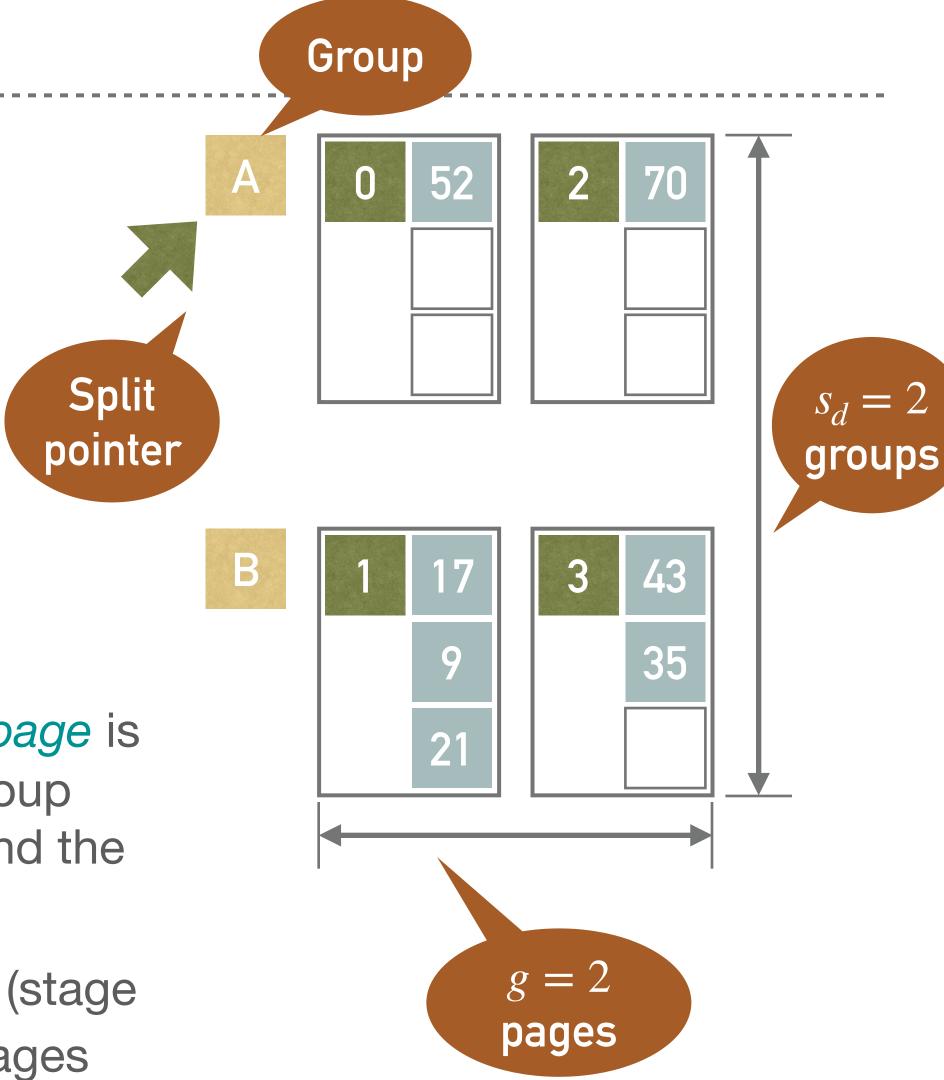
- Insert records with keys 31, 51, 74, and 13 into the following structure
 - * I.e., start the stage d=2 with s=4 pages (capacity 3 records), h(k)=k, p=0
 - * Pre-defined condition: Splitting occurs after 2 inserts

Note all the computations and illustrate the solution



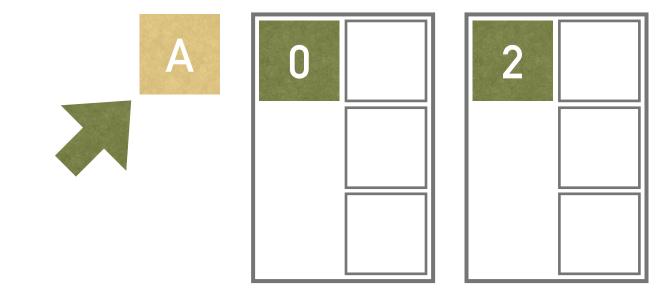
LHPE-RL

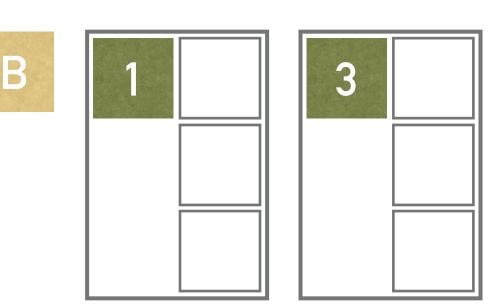
- Simplified version of LHPE
- * At the stage d, the primary file consists of p_d pages
 - * Each page has capacity b
 - * Pages are grouped into $s_d = p_d \div g$ groups
 - * Each *group* has *g* pages
- * When a *predefined condition* is met (e.g., after *L* insertions), a *new page* is inserted at the end of the primary file and records in pages in the group pointed to the *split pointer* are redistributed between these pages and the new page (being the new member of the group)
- * When the last group is redistributed, the file is (virtually) *reorganized* (stage d+1) so that all the pages are again sorted into $s_{d+1}=p_{d+1}\div g$ pages
- * $p_{d+1} = \lceil s_d \cdot (g+1) \div g \rceil \cdot g$



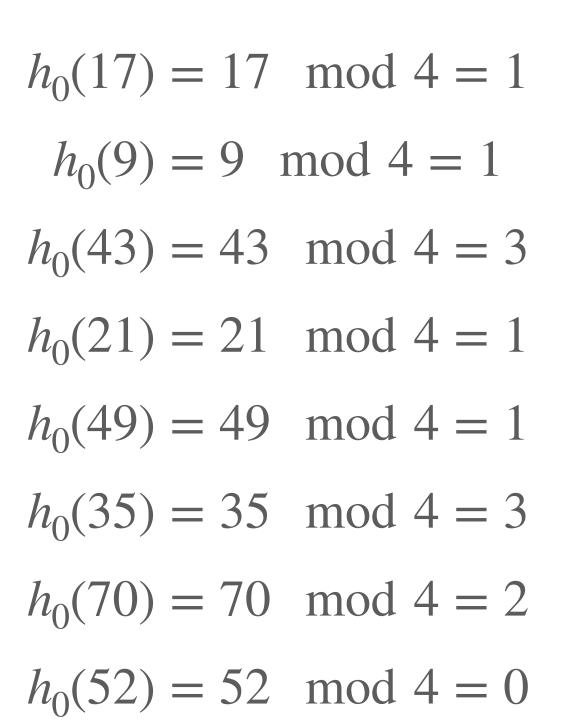
Example 4.8

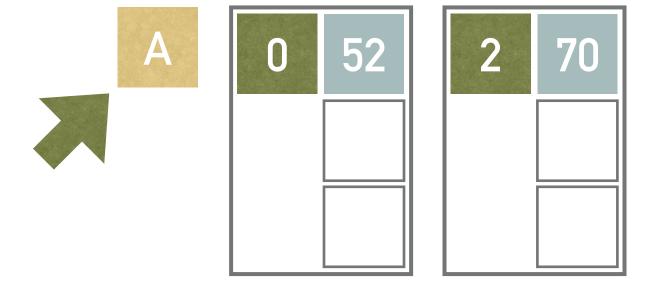
- * Insert records with keys 17, 9, 43, 21, 49, 35, 70, 52, 40, 13, 5, 80 into the following empty structure
 - * Stage d = 0
 - * The initial number of groups $s_0 = 2$
 - * Page capacity b = 3
- Hash function
 - $h_0(k) = k \mod 4$
 - Determines into which of 4 initial pages a record is inserted at the beginning
 - $h_1(k) = k \mod 3$
 - * Determines where the records are inserted when a group splits for the first time
 - * $h_2(k) = \lfloor k \div 3 \rfloor \mod 3$
 - * Determines where the records are inserted when a group splits for the second time
- * We are going to split regularly after two inserts, i.e., L=2
 - * Having $p_0 = s_0 \cdot g = 4$ pages, the first split occurs after insertion of $p_0 \cdot L = 8$ records

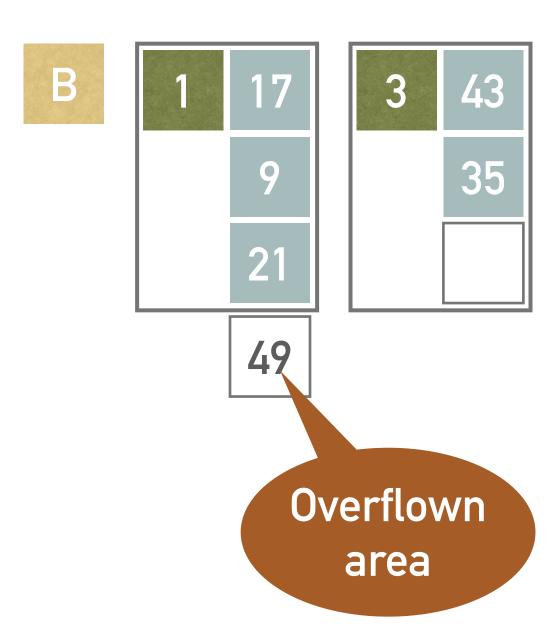




* Inserts of the first 8 keys, i.e., 17, 9, 43, 21, 49, 35, 70, and 52 are not interesting since these are inserted where the h_0 function says

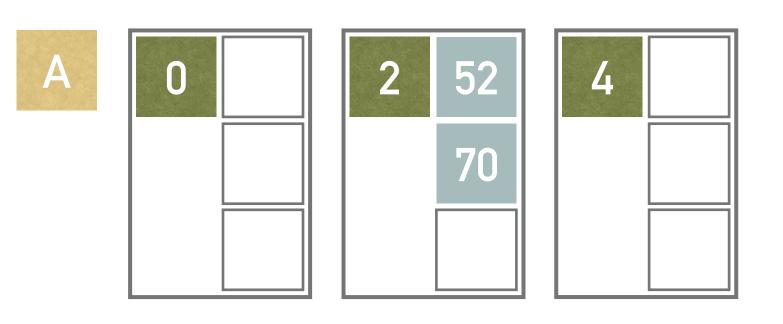


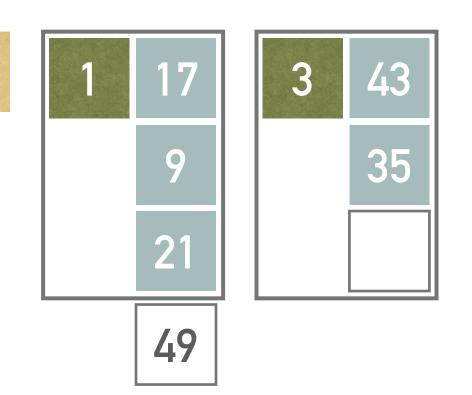




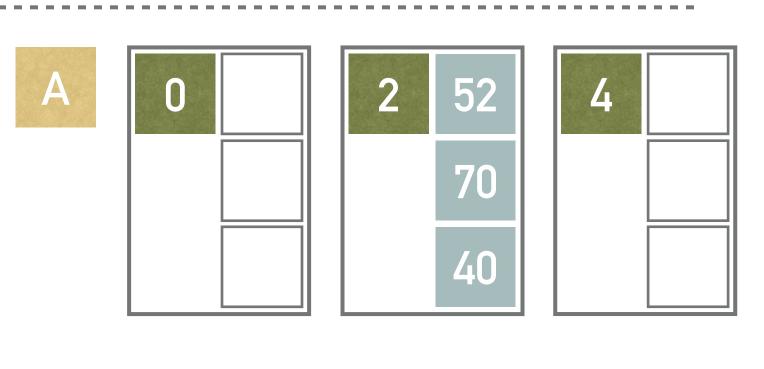
* The only problem is with key 49 which is assigned to an (already full) page 1

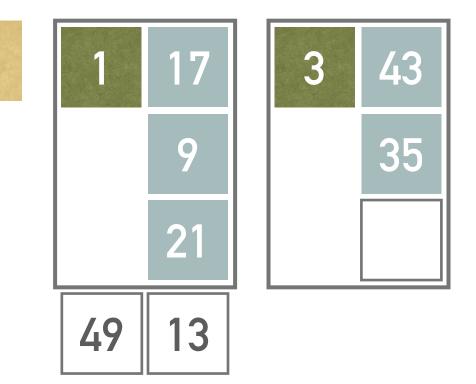
- * Having inserted 8 keys, we have to split the group pointed by the split pointer, i.e., the group A having pages 0 and 2
 - * Page 4 is added into group A
 - * Function $h_1(k)$ is applied in order to redistribute keys in the group A
 - * $h_1(k)$ returns the index of a page in a group A, i.e., $h_1(k)=0$ for the page 0, $h_1(k)=1$ for the page 2, and $h_1(k)=2$ for the page 4
 - * $h_1(52) = 52 \mod 3 = 1$, therefore key 52 goes into the page 2
 - * $h_1(70) = 70 \mod 3 = 1$, hence the key 70 goes into the page 2
 - Split pointer is incremented
- * The key in the overflow area, i.e., 49, does not belong neither to page 0 nor to page 2, thus stays where it is



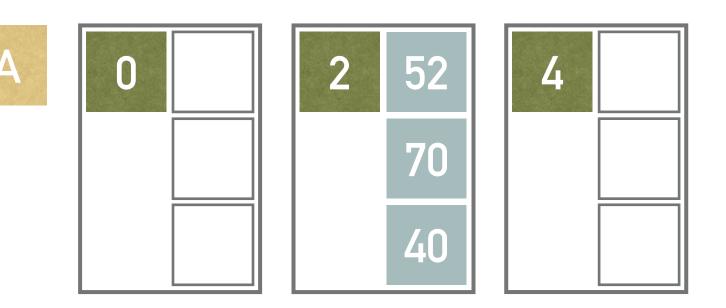


- Next, we insert record with key 40
 - * $h_0(40) = 40 \mod 4 = 0$
 - * Based on the function h_0 , the record with key 40 should be assigned to the page 0 but this page has already been split
 - * Thus we need to use h_1 which sends it into the second page in the group A
 - * $h_1(40) = 40 \mod 3 = 1$ (i.e., page 2)
- Next, we insert record with key 13
 - * $h_0(13) = 13 \mod 4 = 1$
 - * Based on the function h_0 , the record with key belongs to the page 1, which has not been split yet
 - * No need to use h_1
 - * The page 1 is already full, therefore the overflow area is used

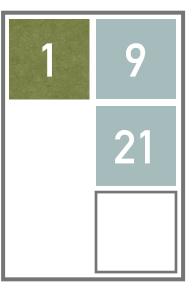


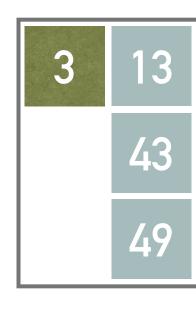


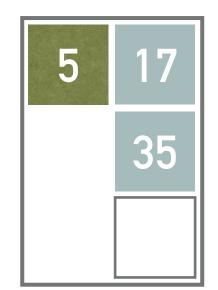
- * Once again, we have to split the group (we have already inserted L=2 records)
 - * Split pointer points to the group B, i.e., pages 1 and 3 will be split
 - Page 5 is added
 - * Function $h_1(k)$ will be applied in order to redistribute keys in the group B
 - * $h_1(17) = 17 \mod 3 = 2$, therefore goes to the page 5
 - * $h_1(9) = 9 \mod 3 = 0$, therefore goes to the page 1
 - * $h_1(21) = 21 \mod 3 = 0$, therefore goes to the page 1
 - * $h_1(43) = 43 \mod 3 = 1$, therefore goes to the page 3
 - * $h_1(35) = 35 \mod 3 = 2$, therefore goes to the page 5
 - * $h_1(49) = 49 \mod 3 = 1$, therefore goes to the page 3
 - * $h_1(13) = 13 \mod 3 = 1$, therefore goes to the page 3





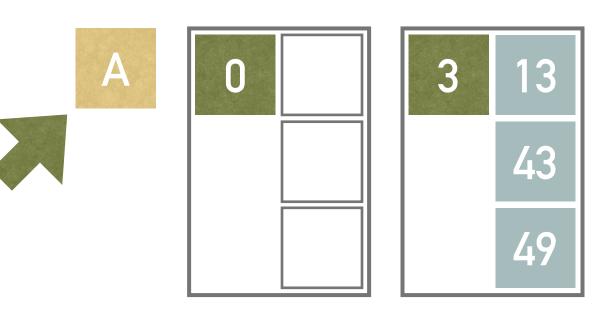


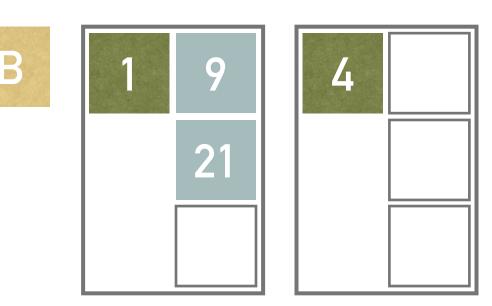


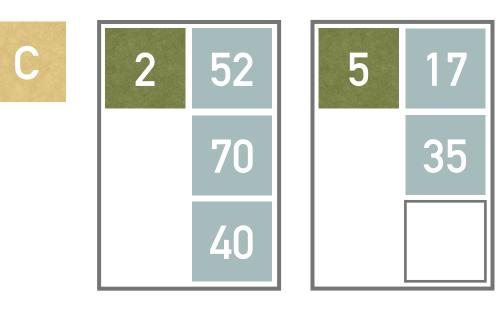


- * Having all the group processed (i.e., split), the end of the stage d=0 occurs
 - Hence, the file is reorganized into 3 groups, each having 2 pages

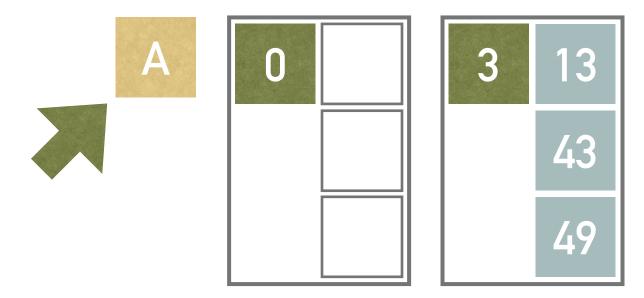
- The reorganization is only virtual
 - The page numbers are kept, we just think of the pages differently

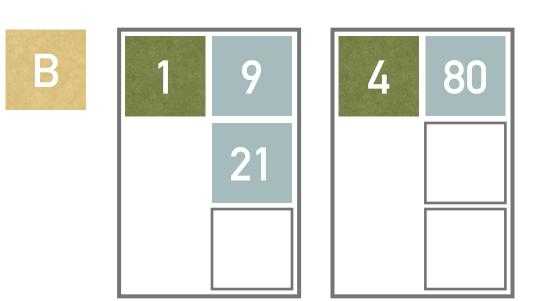


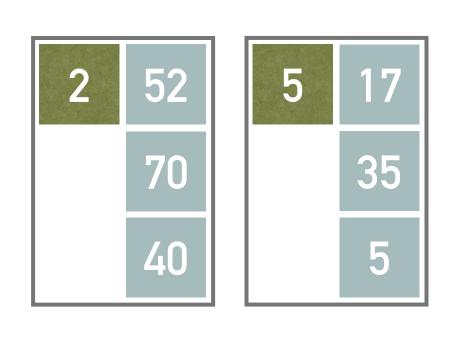




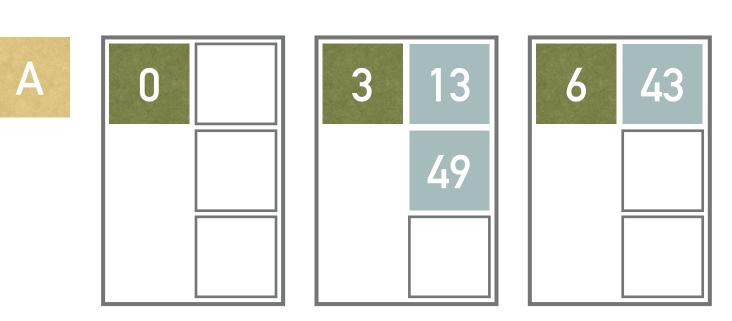
- * Now, we insert record with key 5
 - * $h_0(5) = 5 \mod 4 = 1$
 - * Based on the function h_0 , this record belongs to the page 1, which has been split once
 - * Therefore we have to use h_1
 - * $h_1(5) = 5 \mod 3 = 2$ (note that redistribution is only virtual)
 - * The record comes into page 5
- Next, we insert record with key 80
 - * $h_0(80) = 80 \mod 4 = 0$
 - * Based on the function h_0 , this record belongs to the page 0, which has been split once
 - * Therefore we have to use h_1
 - * $h_1(80) = 80 \mod 3 = 2$ (once again, redistribution is only virtual)

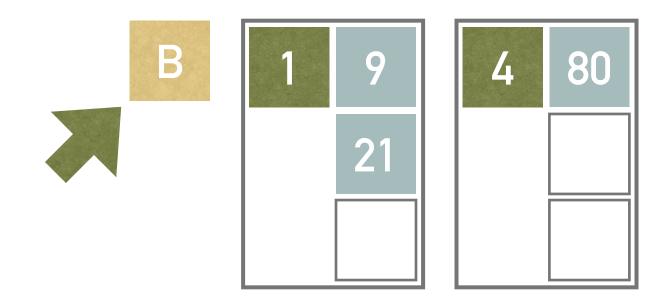


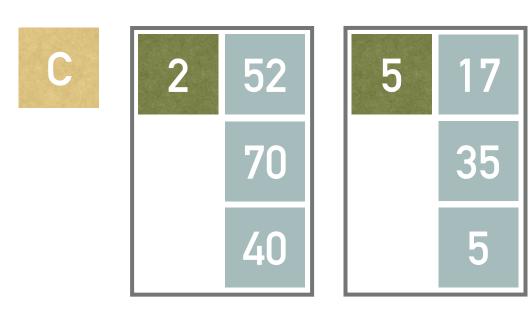




- * Having inserted additional L=2 records, we must split once again
 - * The split pointer points to the group A, i.e., pages 0 and 3
 - Page 6 is added into the group A
 - * Function $h_2(k)$ is applied in order to redistribute keys in the group A
 - * $h_2(k)$ returns the index of a page in a group A, i.e., $h_2(k)=0$ for the page 0, $h_2(k)=1$ for the page 3, $h_2(k)=2$ for the page 6
 - * $h_2(43) = \lfloor 43 \div 3 \rfloor \mod 3 = 2$ (i.e., page 6)
 - * $h_2(49) = \lfloor 49 \div 3 \rfloor \mod 3 = 1$ (i.e., page 3)
 - * $h_2(13) = \lfloor 13 \div 3 \rfloor \mod 3 = 1$ (i.e., page 3)
- Finally, the split pointer is incremented







Exercise 4.9

- Insert record with key 73 into the structure from example 4.8 (see the picture)
 - * Stage d = 1
 - * Page capacity b = 3
 - * Predefined condition L=2
 - * Hash functions:
 - * $h_0(k) = k \mod 4$
 - * $h_1(k) = k \mod 3$
 - * $h_2(k) = \lfloor k \div 3 \rfloor \mod 3$
- * Note all the computations and illustrate the solution

