Lecture 27 **Deletion** & Rehashing

November 18, 2021
Thursday

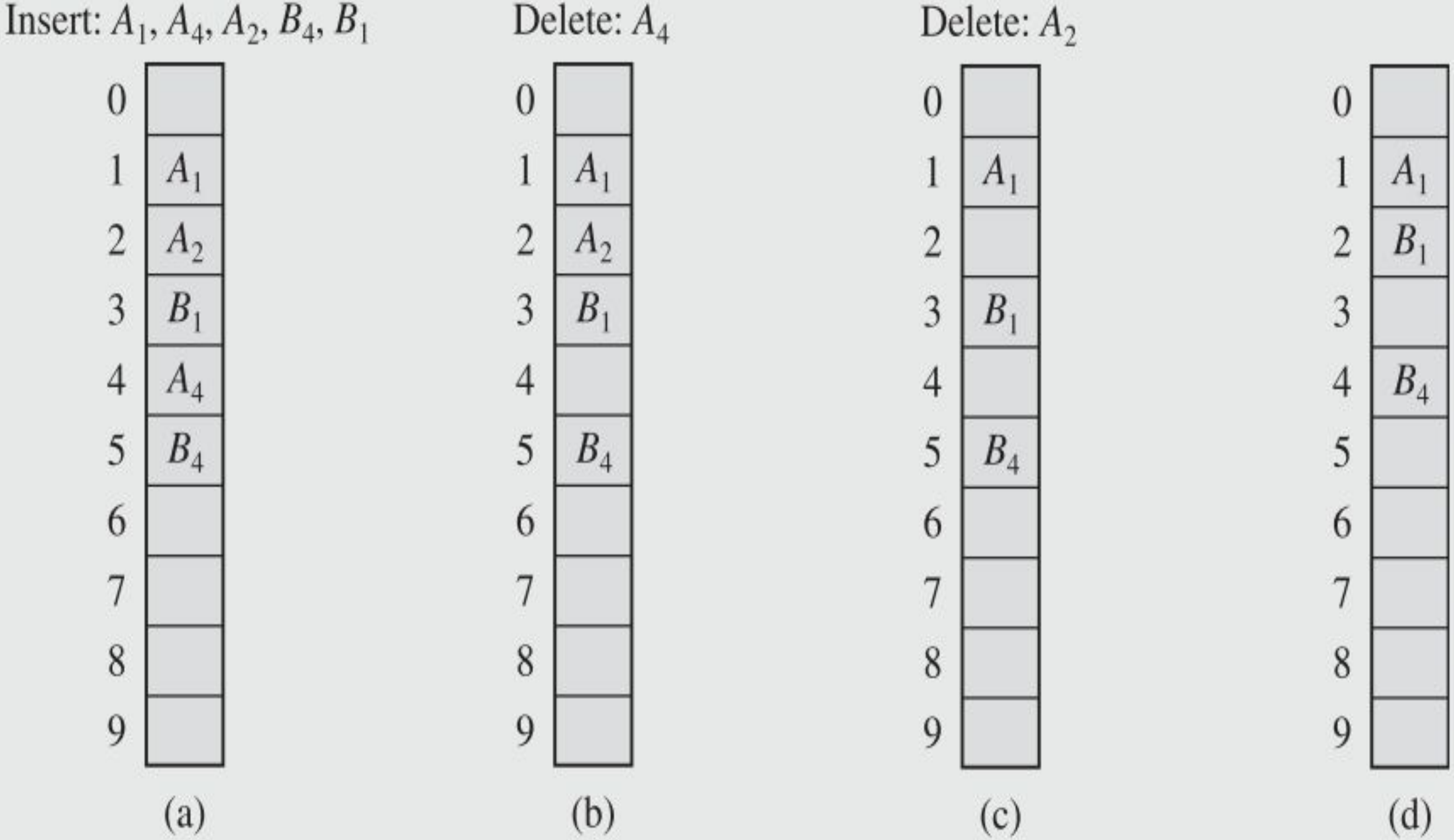
LOAD FACTOR

Load factor is the ratio between number of elements in the table and total size of the table.

$$\text{Loading factor} \quad \mathrm{LF} = \frac{\text{number of elements in the table}}{\text{table size}}$$

DELETION

- How can we remove data from a hash table?
- With a chaining method, deleting an element leads to the deletion of a node from a linked list holding the element.
- What about Close Hashing methods?
    - A deletion operation may require a more careful treatment of collision resolution.
        - Except for the rare occurrence when a perfect hash function is used.
    -


Insert: A_1, A_4, A_2, B_4, B_1
Delete: A_4
Delete: A_2
0
1
2
3
4
5
6
7
8
9
A_1
A_2
B_1
A_4
B_4
(a)
(b)
(c)
(d)

DELETION

- If we leave deleted keys in the table with markers indicating that they are not valid elements of the table
  - Then any subsequent search for an element does not terminate prematurely.
- When a new key is inserted, it overwrites a key that is only a space filler.

DELETION

- However, for a large number of deletions and a small number of additional insertions the table becomes overloaded with deleted records.
  - This increases the search time, cause now you are testing the deleted elements.
- A table needs to be purged after a certain number of deletions by moving the undeleted elements to the cells occupied by the deleted elements.
- Cells with deleted elements that are not overwritten by this procedure are marked as free.

Lecture 27

Deletion

&

Rehashing

November 18, 2021
Thursday

REHASHING

- When a table becomes full, insertion of new elements becomes impossible.
  a. When it reaches a saturation level searching becomes slower as well.
- In such a case, we can
  a. Allocate a larger table.
  b. Modify the hash function.
  c. Hash all items from the old table to new table.
  d. Discard the old table.
  e. Use the new table with new hash function.

REHASHING

- The new size of the table can be
  a. The double of the old table size.
  b. Can be a prime closest to the double size of the old.
  c. Can be the current size + predefined value.
  d. Random value can be chosen as well.
- All discussed methods can use rehashing, after which they can continue processing data using the same methods of hashing and collision resolution.

Cuckoo Hashing

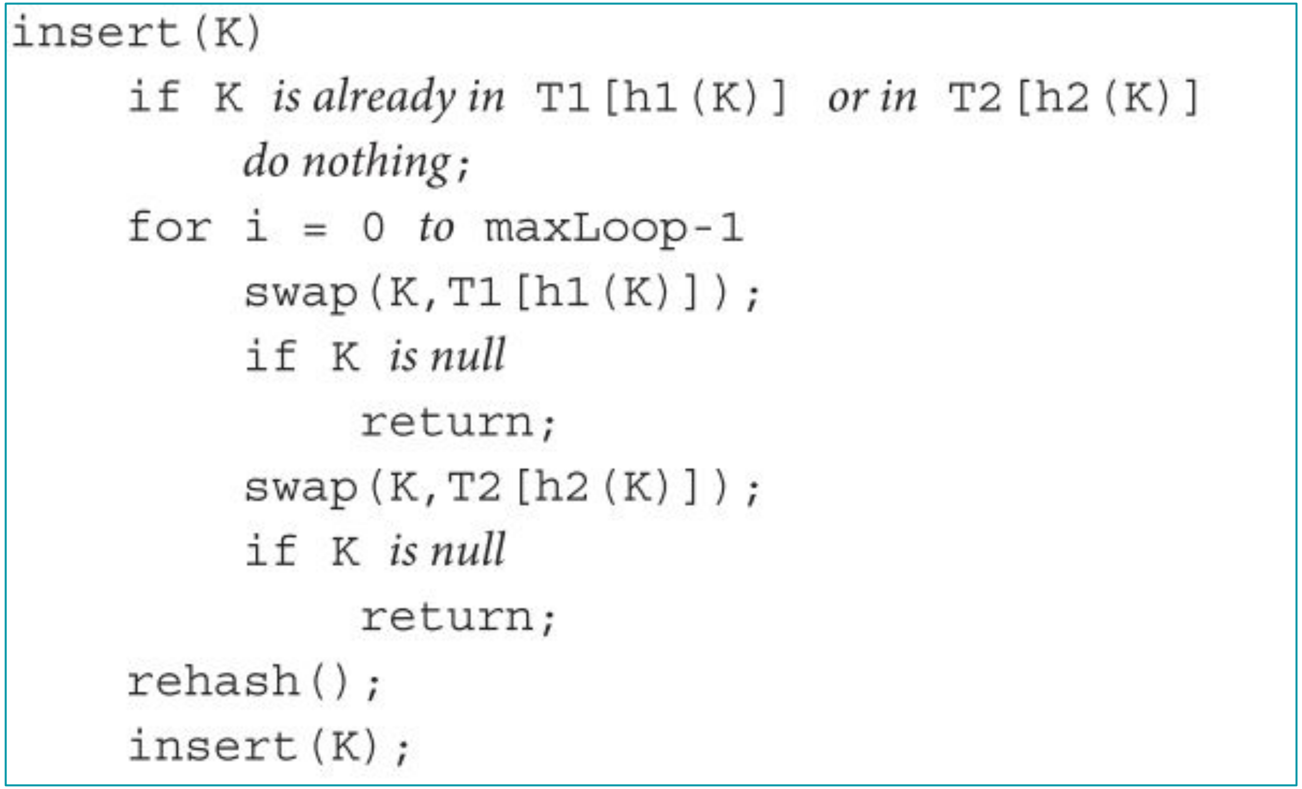
- The Cuckoo hashing uses two tables T_1 and T_2 and two hash functions h_1 and h_2.
- To insert a key K_1, table T_1 is checked with function h_1.
  - If position $T_1[h_1(K_1)]$ is free, the key is inserted there.
  - If the position is occupied by a key K_2
    - K_2 is removed to make room for K1
      - Then an attempt is made to place K_2 using h_2 in T_2 T2[h2(K2)].
  - If the position is occupied by K_3, then K_3 is removed to place K_2.
  - K_3 is placed in T_1 using h_1: $T_1 [h_1 (K_3)]$.

Cuckoo Hashing

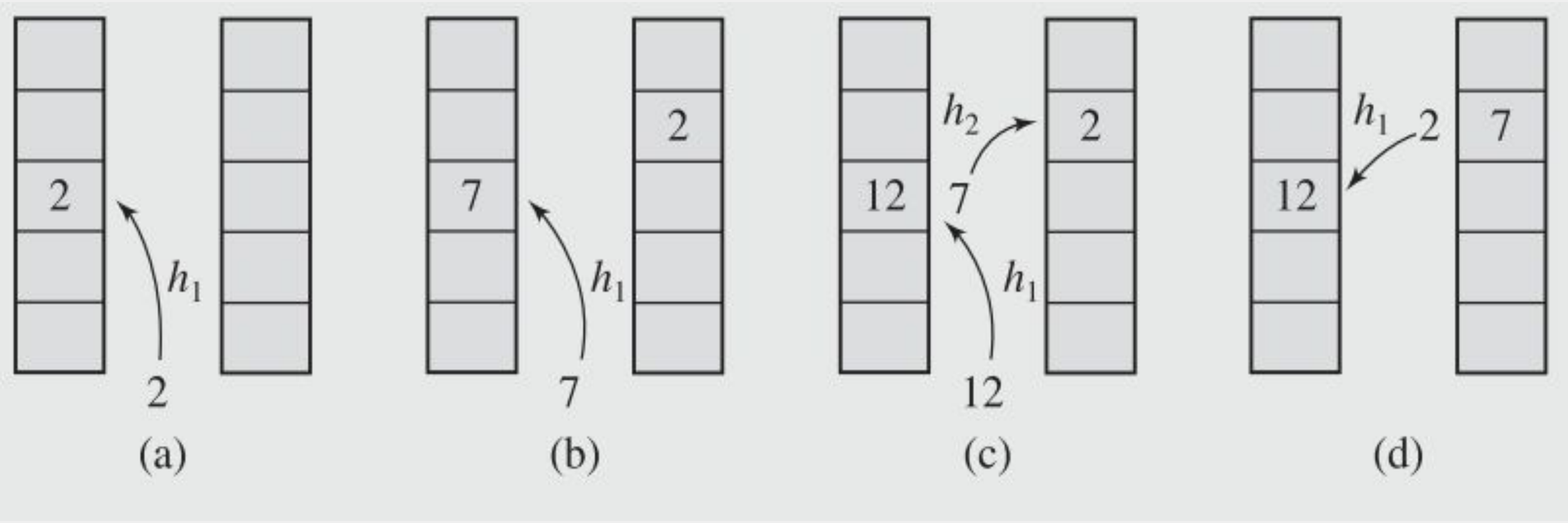
In theory, such a sequence of attempts can lead to an infinite loop if the very first position that was tried is tried again.

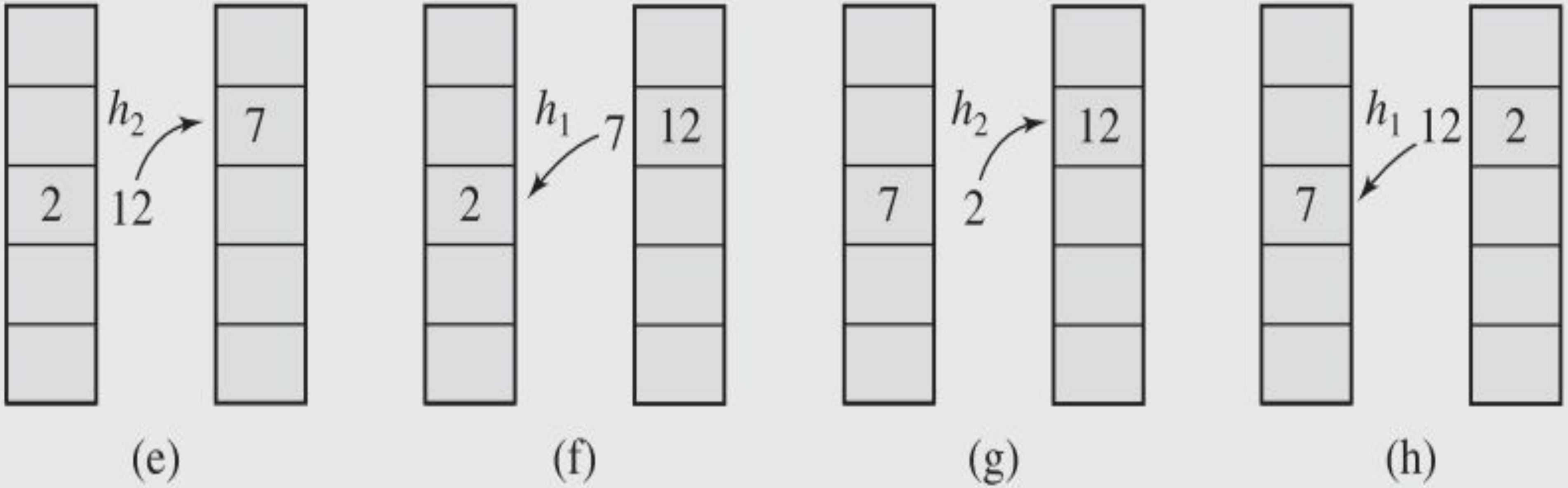
To avoid the problem, a limit on tries is set and then if it is exceeded, rehashing is performed by creating two new and larger tables, defining two new hash functions and rehashing keys from the old tables to the new ones.

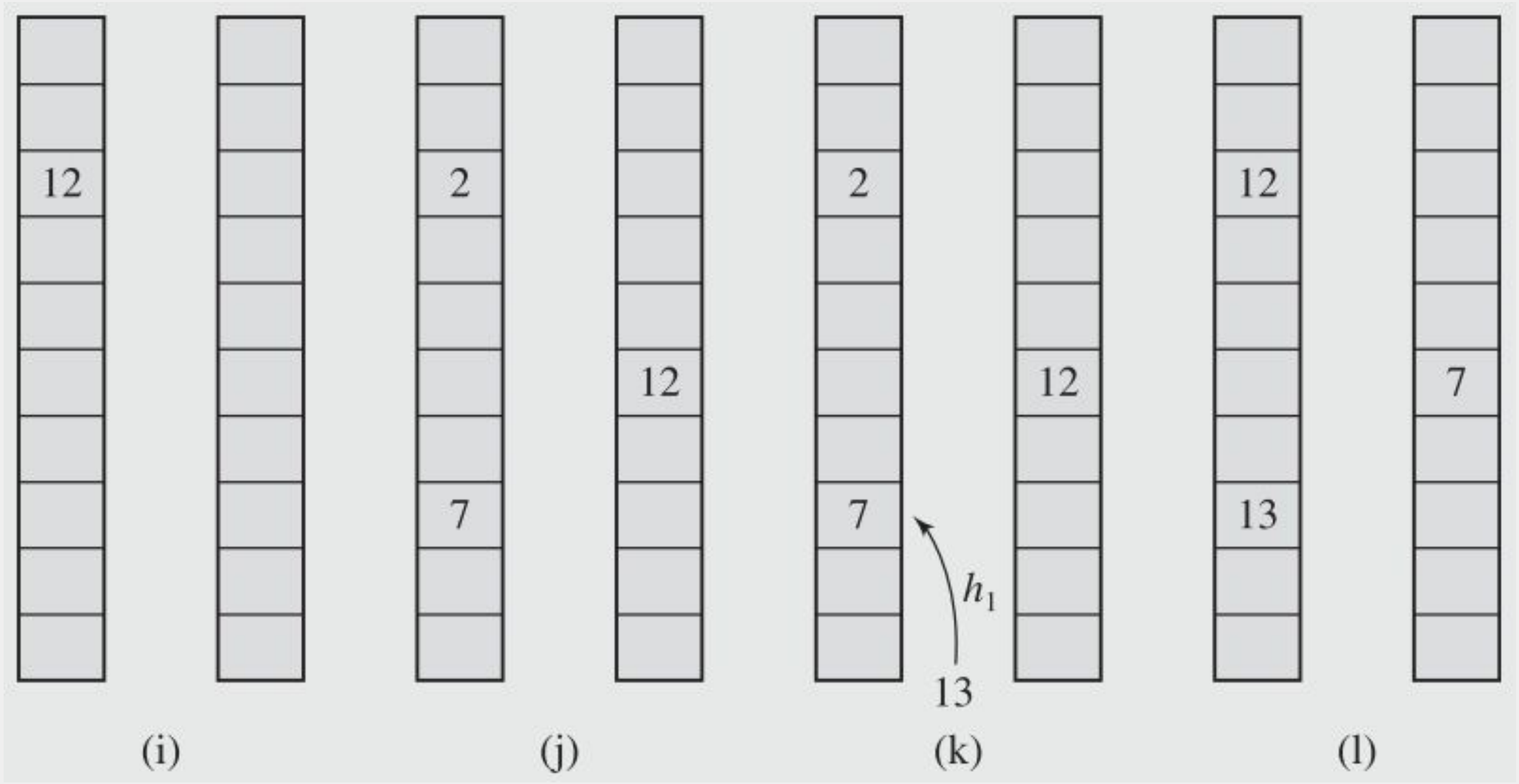
Cuckoo Hashing



```
insert(K)
    if K is already in T1[h1(K)] or in T2[h2(K)]
        do nothing;
    for i = 0 to maxLoop-1
        swap(K,T1[h1(K)]);
        if K is null
            return;
        swap(K,T2[h2(K)]);
        if K is null
            return;
    rehash();
    insert(K);
```


2
h_1
2
7
h_1
7
2
12
h_2
2
7
h_1
12
12
h_1
2
7
(a)
(b)
(c)
(d)


h_2
2
12
7
(e)
h_1
7
2
12
(f)
h_2
7
2
12
(g)
h_1
12
7
2
(h)


12
2
12
7
2
12
7
h_1
13
12
13
7
(i)
(j)
(k)
(l)

PERFECT HASHING

- All the cases discussed earlier assume that the body of the data is not precisely known.
  a. Therefore, the hash function only rarely turned out to be an ideal hash function.
- In addition, the number of keys is rarely known in advance, so the table had to be large enough to accommodate all the arriving data.
- The table size contributed to the number of collisions: a larger table has a smaller number of collisions

PERFECT HASHING

- What if the body of the data is fixed, and a hash function can be devised after the data are known.
- If such a function requires only as many cells in the table as the number of data so that no empty cell remains after hashing is completed, it is called a **minimal perfect hash function.**
- Wasting time for collision resolution and space for deleted cells are avoided in such a hash function.

PERFECT HASHING

- What if the body of the data is fixed, and a hash function can be devised after the data are known.
- If such a function requires only as many cells in the table as the number of data so that no empty cell remains after hashing is completed, it is called a **minimal perfect hash function.**
- Wasting time for collision resolution and space for deleted cells are avoided in such a hash function.