Hashing
Deletion algorithms

D.E ZEGOUR
École Supérieure d'Informatique
ESI

Hashing / Deletion algorithms

Linear probing

Let i be the element to delete (d).

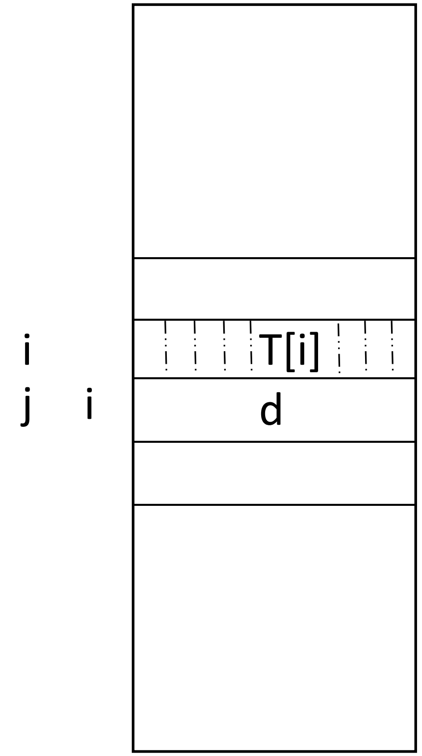
Primary address of d : any possible address

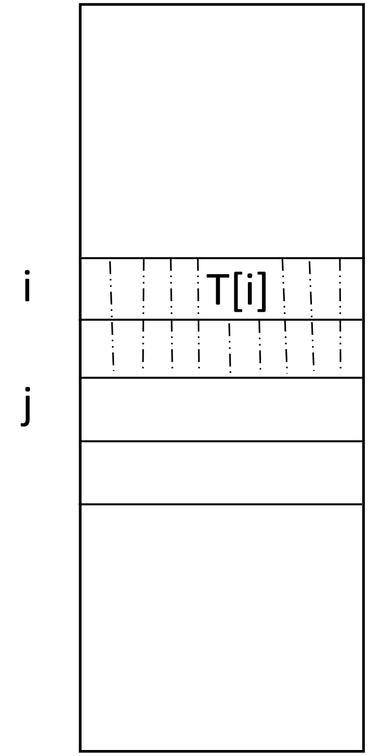
h(d) = i
h(d) < i
h(d) > i

1. Make T(i) empty.
   Let j := i.

2. i := i - 1; If i < 0 : i := i + M

3. If T(i) is empty, the algorithm ends.
Otherwise, let r := h(T(i)).





Hashing / Deletion algorithms

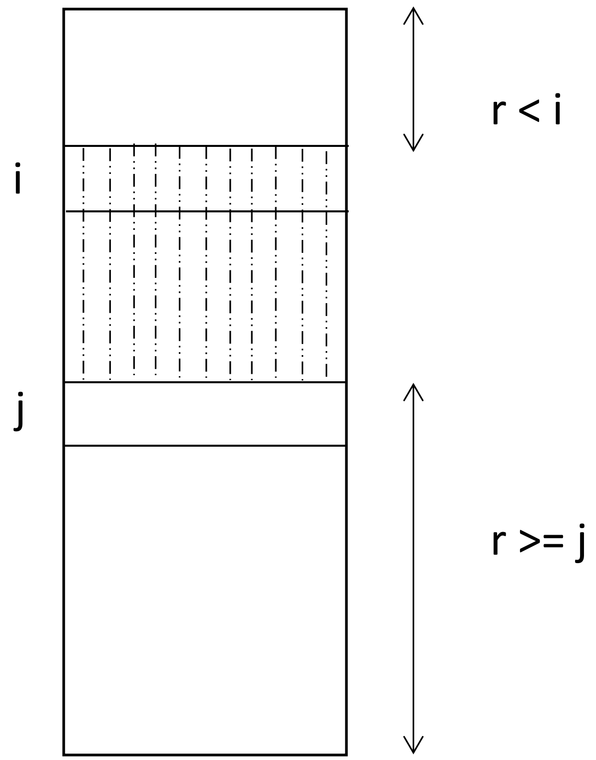
Linear probing

1. Make T(i) empty.
   Let j := i.

2. i := i - 1; If i < 0 : i := i + M

3. If T(i) is empty, the algorithm ends.
Otherwise, let r := h(T(i)).

4. CASE i < j
If r < i or r ≥ j:
    Move the element, i.e., T(j) := T(i)
    Go to 2
Otherwise, Go to 1



Hashing / Deletion algorithms

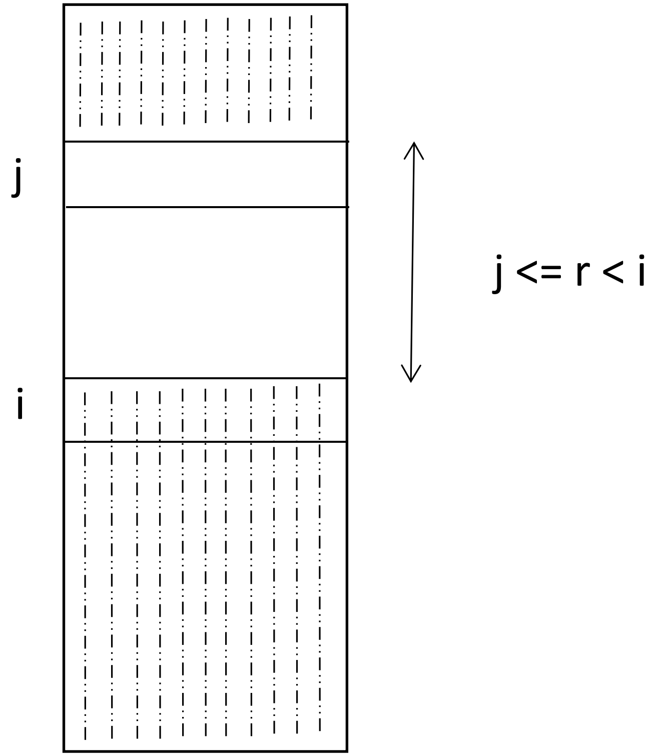
Linear probing

1. Make T(i) empty.
   Let j := i.

2. i := i - 1; If i < 0 : i := i + M

3. If T(i) is empty, the algorithm ends.
Otherwise, let r := h(T(i)).

4. CAS i > j
   If j ≤ r < i:
      Move the element, i.e., T(j) := T(i)
      Go to 2
   Otherwise, Go to 1



Hashing / Deletion algorithms

Linear probing

Deleting data e

Primary addresses : a(3), b(2), c(3), d(2), e(1)

Make T(5) empty

Previous cell (T(4)) is empty : the algorithm ends

0	d
1	c
2	b
3	a
4	
5	e

Hashing / Deletion algorithms

Linear probing

Deleting data a

Primary addresses : a(3), b(2), c(3), d(2), e(1)

- Make cell 3 empty.
- Since cell 2 is not empty, the algorithm continues.
- As b is in its primary address (h(b) = 2), it will not be moved, and the algorithm continues.

- The primary address of c is 3; c will be moved to position 3. The algorithm continues since the cell before c is not empty.

d will be moved to position 1.

e will be moved to position 0.

0	d	0	d	0		0	e
1	c	1		1	d	1	d
2	b	2	b	2	b	2	b
3	a	3	c	3	c	3	c
4		4		4		4	
5	e	5	e	5	e	5	

Hashing / Deletion algorithms

Double hashing

One cannot find an algorithm analogous to that of linear probing.

A simple method: a logical deletion (adding an erase bit).

Hashing / Deletion algorithms

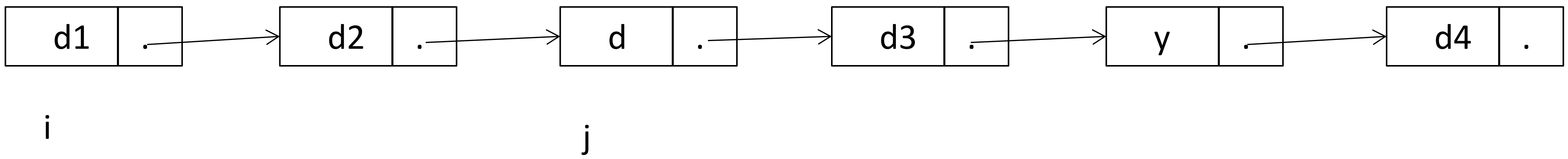
Internal chaining

Suppose we want to delete the element d.

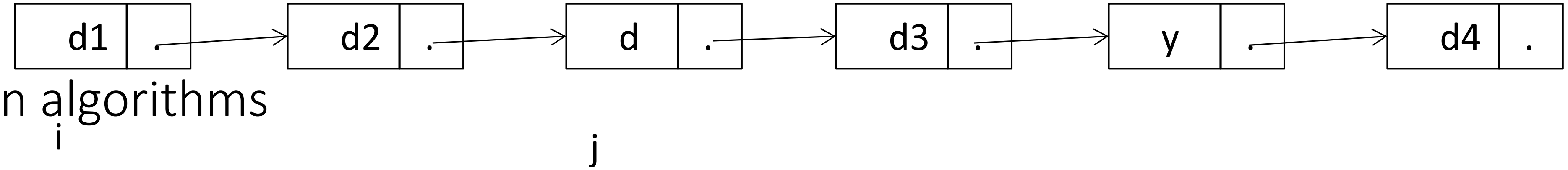
Search for the element.

Let i be its primary address, and j be the index of element d. (i can be equal to j.)

So, i is the list that contains d.



Hashing / Deletion algorithms



Internal chaining

The algorithm is as follows:

1. Check if there is another data element y further along in the list (starting from element j) such that the list h(y) passes through j.

2. If y does not exist, remove d from the list by adjusting the chaining, and the algorithm terminates.

3. If y exists, move it to position j. Set j := index of y and d := y, then restart from step 1.

In both cases, update the variable R as follows:
(assuming k is the index of the deleted element) IF k > R: R := k + 1 ENDIF

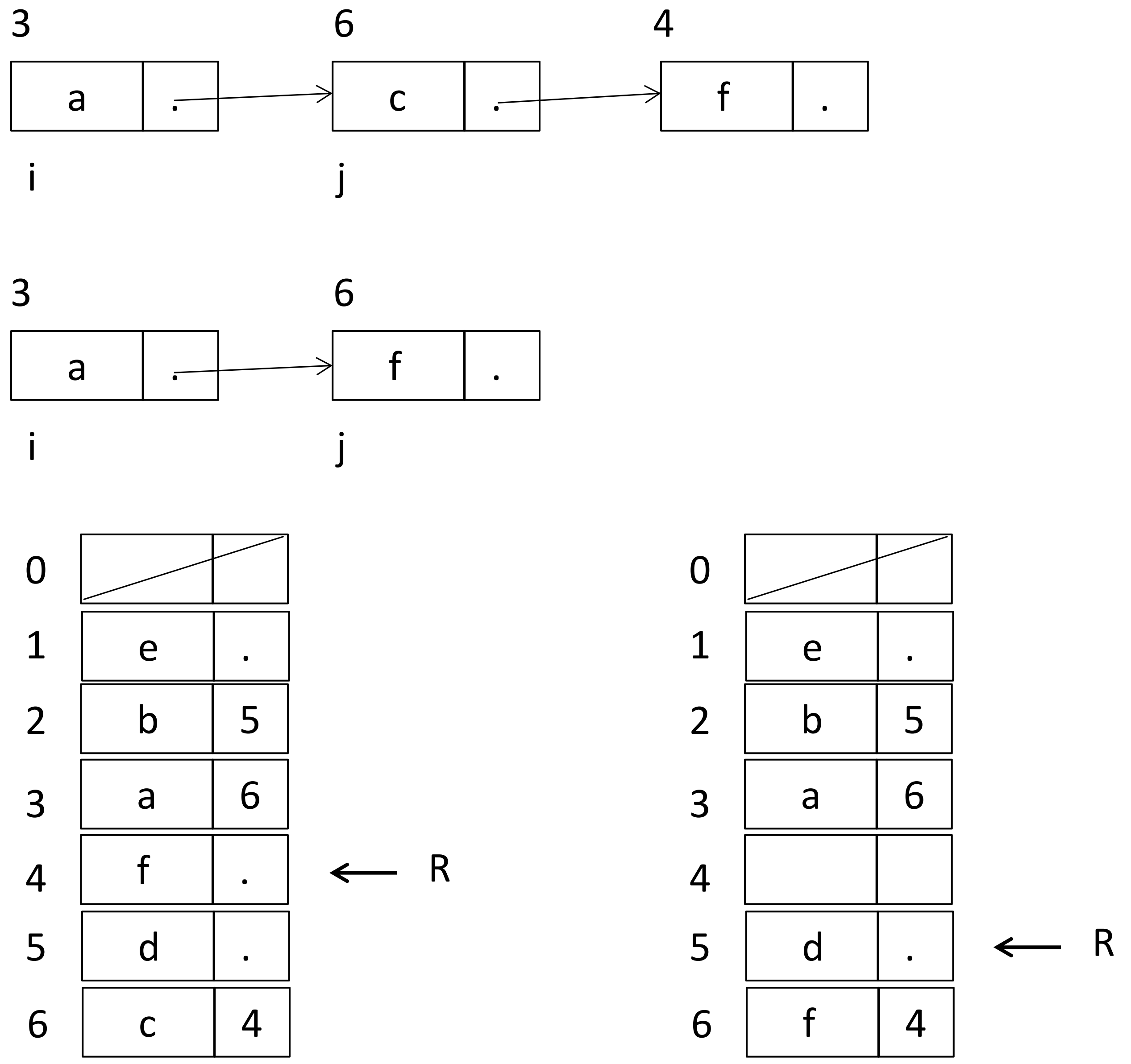
Hashing / Deletion algorithms

Internal chaining

Deleting data c

Reminder: a(3), b(2), c(3), d(2), e(1), f(3)

- c is found at j=6.
- The primary address of c is i=3.
- There exists y such that the list h(y) passes through 6 (y=f).
- Since the list that starts at 3 passes through j=6, f will be moved to position 6.



Hashing / Deletion algorithms

Separated chaining

The algorithm for deleting an element is very simple. It simply involves removing an element from a linked list.