Collision Resolution: Open Addressing

We must initialize the table

	T
0	14, d_1
1	21, d_4
2	19, d_5
3	2, d_6
4	5, d_7
5	12, d_2 ← pos
6	13, d_3

$h(14)=0$
$h(12)=5$
$h(13)=6$
$h(21)=0$
$h(19)=5$

NULL cannot be one of the possible data items!

$h(2)=2$

get(3)

$h(k) = k \bmod 7$

Lazy evaluation:

Records to store in the table

($14,d_1$)
($12,d_2$)
($13,d_3$)
($21,d_4$)
($19,d_5$)

($2,d_6$)
($5, d_7$)

remove (14) ⇐ After removing (14), (19) is not in the right spot. And the array need to be rescheduled to make sure all elements are in the right place.

This is called rehashing.

Algorithm get(k)
Input: Key k
Output: Record with key k, or null if no record has key k

pos ← h(k)
count = 0
while (T[pos] ≠ NULL) and (T[pos].getkey() ≠ k) do {
  pos ← (pos + 1) mod M
  count += 1
  if (count == n) { // everything in table is checked)
   return null }.
}
if T[pos] = null then return null
else return T[pos]

worst case: x is not in the table.
In this time, the complexity will be O(n).

Linear probing:

h(k), (h(k)+1) mod M, (h(k) + 2) mod M, ((h(k) + 3) mod M …

Linear Probing and Double Hashing

0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	

$h(k) = k \bmod 11$

Records to store in the table

$(3, d_1)$
$(14, d_2)$
$(25, d_3)$
$(5, d_4)$
$(28, d_5)$
$(91, d_6)$

0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	

Secondary hash function:
$h'(k) = q - (k \bmod q)$
for some prime value q

$h'(k) = 7 - (k \bmod 7)$

Linear probing:
$h(k)$, $(h(k)+1) \bmod M$, $(h(k) + 2) \bmod M$, $((h(k) + 3) \bmod M$ …

Double hashing:
$h(k)$, $(h(k)+h'(k)) \bmod M$, $(h(k) + 2h'(k)) \bmod M$, $((h(k) + 3h'(k)) \bmod M$ …

Double Hashing and Size of the Table

0	
1	
2	
3	
4	
5	
6	
7	

$h(k) = k \bmod 8$

Records to store in the table

$(2, d_1)$
$(6, d_2)$
$(10, d_3)$

Secondary hash function:

$h'(k) = q - (k \bmod q)$

for some prime value q

$h'(k) = 7 - (k \bmod 7)$

Double hashing:

$h(k)$, $(h(k)+h'(k)) \bmod M$, $(h(k) + 2h'(k)) \bmod M$, $((h(k) + 3h'(k)) \bmod M$ …

The size of the hash table must be a prime number.

Open Addressing: put Method (linear probing)

Algorithm put (k,data, *M*)

In: record (k,data) to insert, size M of hash table

Out: {add record (k,data) to table, or ERROR if insertion not allowed}

pos ← h(k)

count ← 0

while (T[pos] != NULL) **and** (T[pos] != DELETED) **do** {

    if T[pos].getKey() = k **then** *ERROR*

    pos ← (pos + 1) **mod** M

    count ← count + 1

    if count = M **then** *ERROR*

}

T[pos] ← (k,data)

Open Addressing: put Method (double hashing)

Algorithm put (k,data, M)

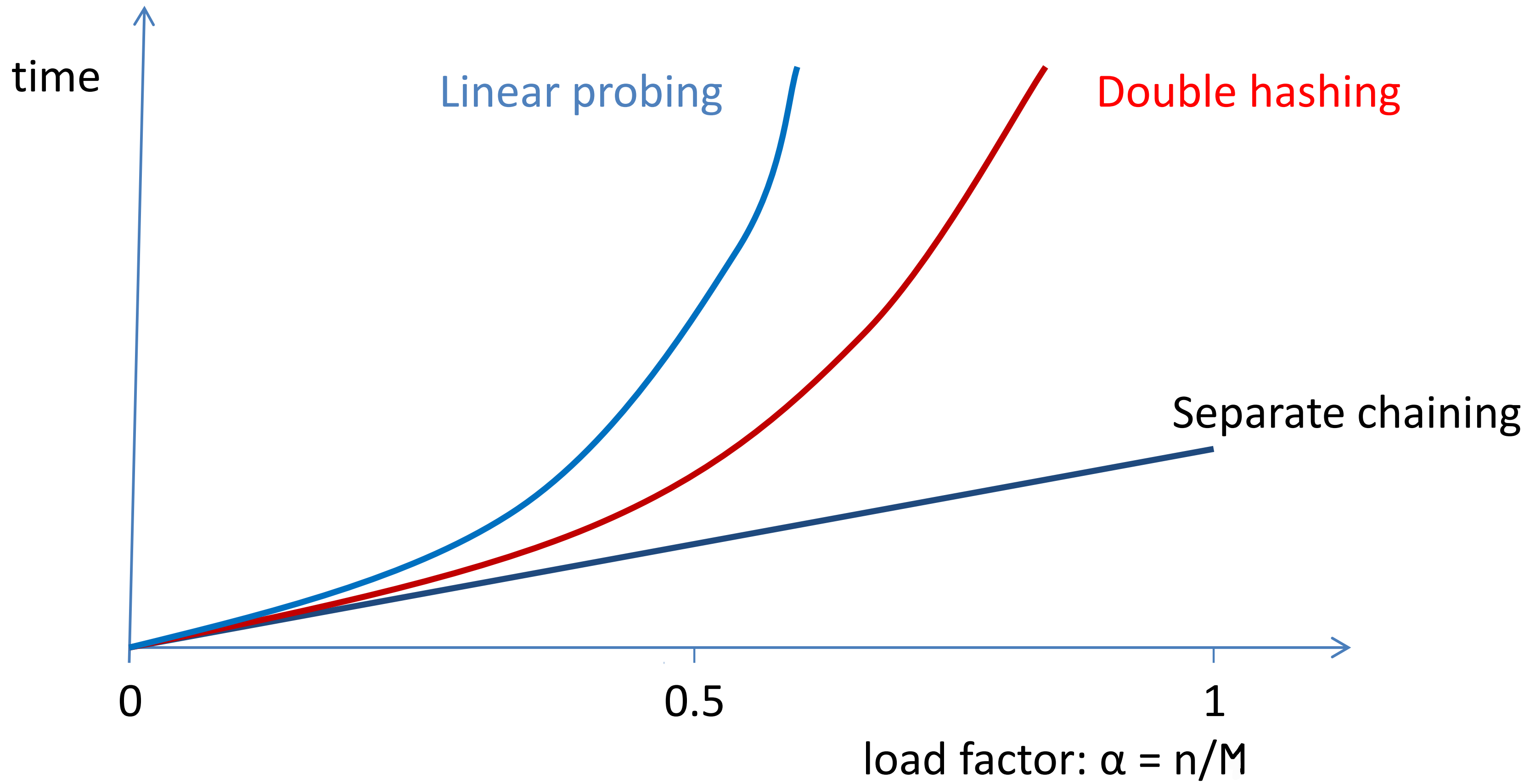
In: record (k,data) to insert, size N of hash table

Out: {add record (k,data) to table, or ERROR if insertion not allowed}

pos ← h(k)
count ← 0

while (T[pos] != NULL) **and** (T[pos] != DELETED) **do** {
    if T[pos].getKey() = k **then** *ERROR*
    pos ← (pos + h'(k)) **mod** M
    count ← count + 1
    if count = M **then** *ERROR*
}
T[pos] ← (k,data)

Average Time Complexity of get Operation



	Average number of key comparisons
Separate chaining	$1 + \alpha$
Linear Probing	$\frac{1}{2} + 1/(2(1 - \alpha)^2)$
Double Hashing	$1/(1 - \alpha)$