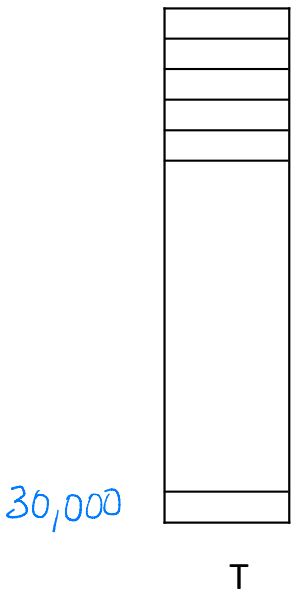
Hash Code

$g(\text{“}c_{k-1}\ c_{k-2} \ldots c_2\ c_1\ c_0\text{”}) \rightarrow \text{integer}$

$g(\text{“}c_{k-1}\ c_{k-2} \ldots c_2\ c_1\ c_0\text{”}) =$

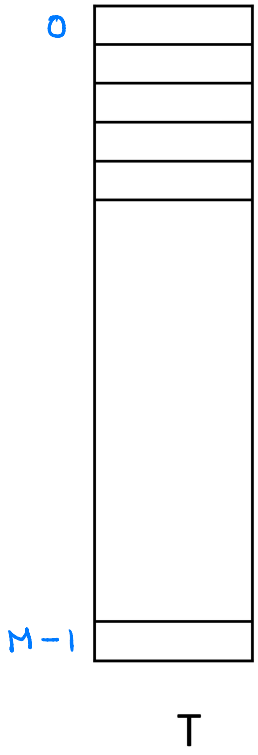


Polynomial Hash Code

Polynomial: $p(x) = a_{k-1}x^{k-1} + a_{k-2}x^{k-2} + \ldots + a_1x^1 + a_0$

$g(\text{“}c_{k-1}\ c_{k-2} \ldots c_2\ c_1\ c_0\text{”}) =$

Compression Map



$f(\text{integer}) \longrightarrow \{0, 1, \ldots, M-1\}$

Hash Function with Polynomial Hash Code

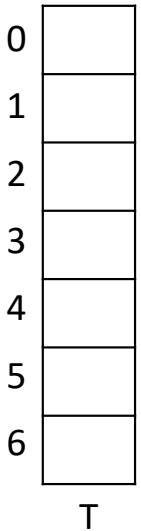
$h(\text{“}c_{k-1}\ c_{k-2}\ \ldots\ c_0\text{”}) = (\ldots((((\text{int})c_{k-1})x + (\text{int})c_{k-2})x + (\text{int})\ c_{k-3})x + \ldots + (\text{int})c_1)x + (\text{int})c_0) \bmod M$

Hash Function with Polynomial Hash Code

$h(\text{“}c_{k-1}\ c_{k-2}\ \ldots\ c_0\text{”}) = (\ldots((((\text{int})c_{k-1})x + (\text{int})c_{k-2})x + (\text{int})\ c_{k-3})x + \ldots + (\text{int})c_1)x + (\text{int})c_0) \bmod M$

Algorithm polynomialHashFunction(“c_{k-1},c_{k-2}, … c_0”,x,M)
Input: String “c_{k-1},c_{k-2}, … c_0”, value x, size M (M is a prime number) of the hash table
Output: value of the hash function for input string

Collision Resolution: Separate Chaining



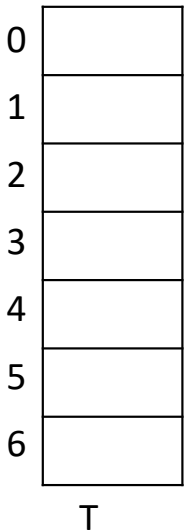
h(k) = k mod 7

Records to store
in the table

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

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

Collision Resolution: Open Addressing



$h(k) = k \bmod 7$

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)

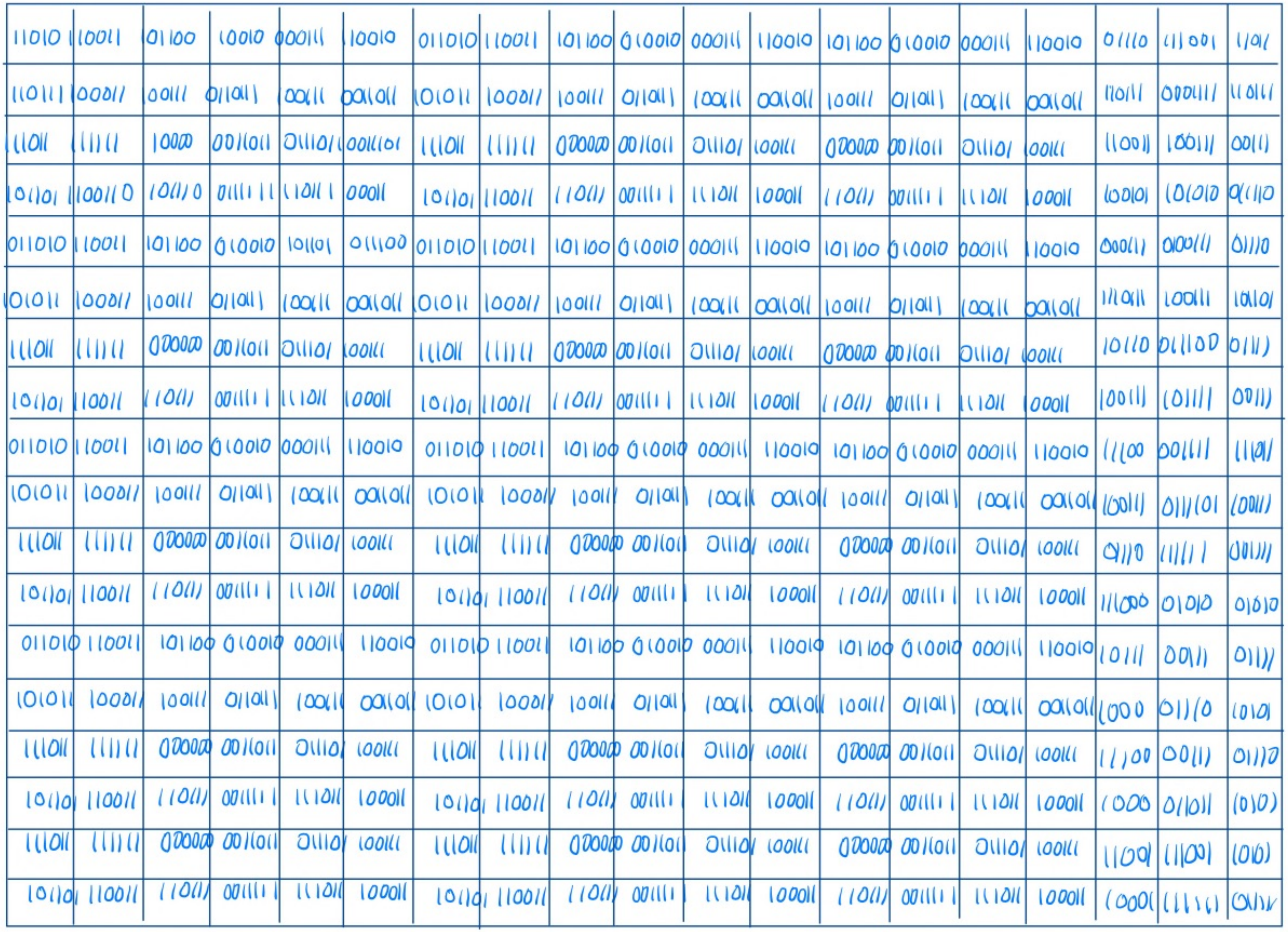
Algorithm get(k)
Input: Key k
Output: Record with key k, or
null if no record has key k

Initially every entry of T is null

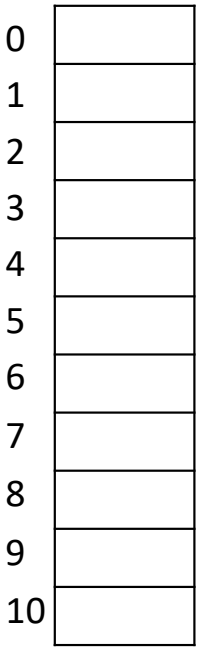
Linear probing:

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

Computer Memory



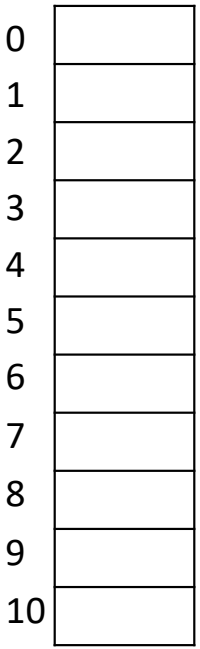
Linear Probing and Double Hashing



$h(k) = k \bmod 11$

Records to store in the table

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



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

Double hashing:

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

Double Hashing and Size of the Table

$h(k) = k \bmod 8$

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:

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

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)

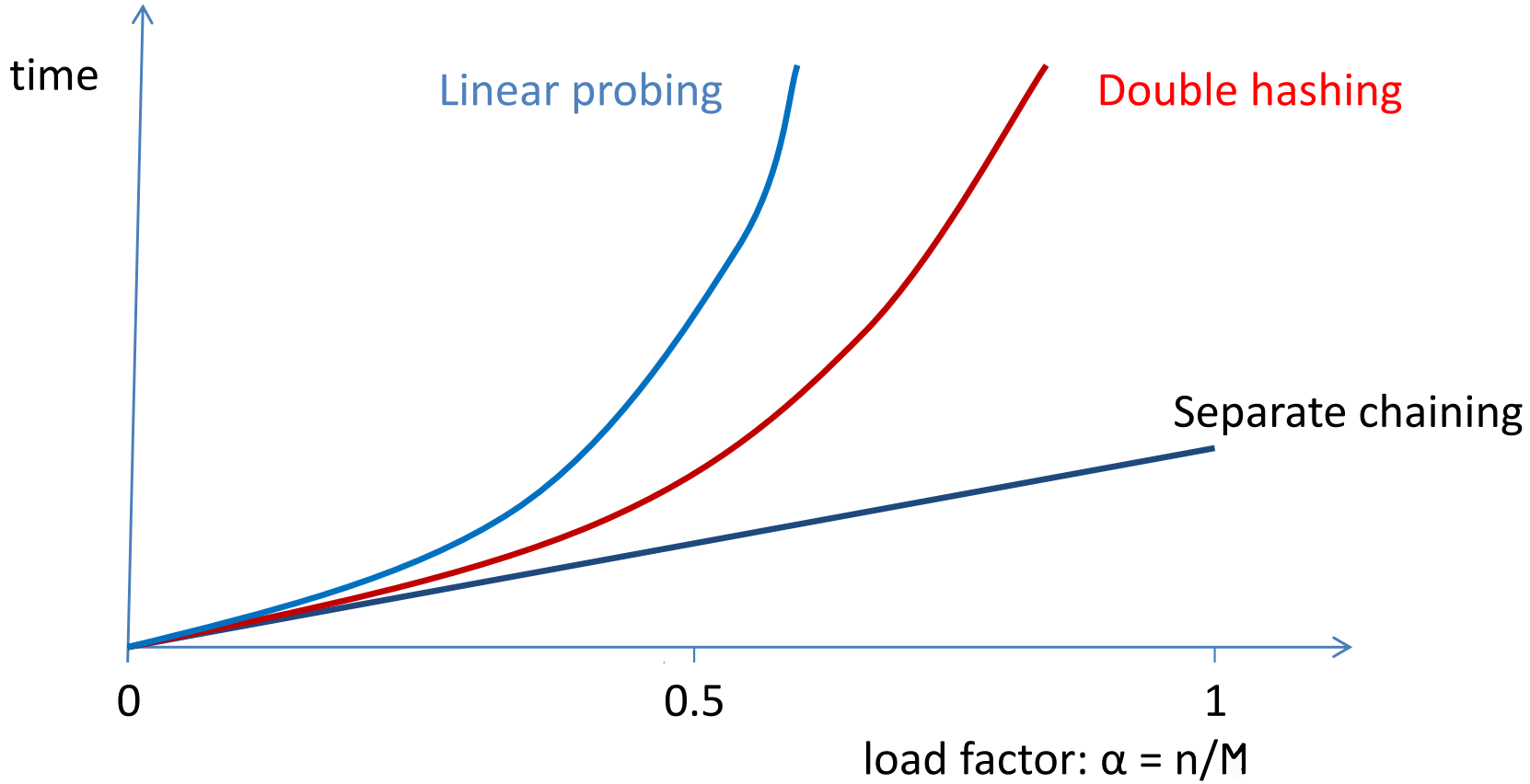
In: record (k,data) to insert, size N of hash table

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	$1/(1 - \alpha)$