Hash Code

$$g("c_{k-1} c_{k-2} ... c_2 c_1 c_0") \rightarrow integer$$

$$g("c_{k-1} c_{k-2} ... c_2 c_1 c_0") =$$

J -

30,000

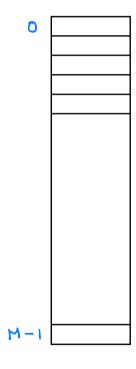
Т

Polynomial Hash Code

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

$$g("c_{k-1} c_{k-2} ... c_2 c_1 c_0") =$$

Compression Map



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

Hash Function with Polynomial Hash Code

$$h("c_{k-1} c_{k-2} ... c_0") = (... ((((int)c_{k-1})x + (int)c_{k-2})x + (int) c_{k-3})x + ... + (int)c_1)x + (int)c_0) \bmod M$$

Hash Function with Polynomial Hash Code

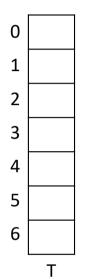
$$h("c_{k-1} c_{k-2} ... c_0") = (... ((((int)c_{k-1})x + (int)c_{k-2})x + (int) c_{k-3})x + ... + (int)c_1)x + (int)c_0) \mod M$$

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



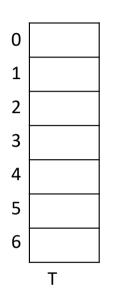
```
h(k) = k mod 7

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 \mod 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

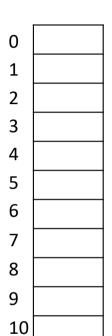
Linear probing:

```
h(k), (h(k)+1) \mod M, (h(k)+2) \mod M, ((h(k)+3) \mod M ...
```

Computer Memory

110101	10011	01100	10010	1000111	10010	011010	110011	101100	010010	000111	110010	101100	010010	000111	110010	01110	111001	1/01/
1101111	11600	ooll	0/10/1/	1000	palloll	101011	100011	100111	0/10(1)	10011	0011011	100111	Ollari	(00(11	0011011	non	000111	1110111
1011	uru	10000	oollou	01110/1	0011101	1(1011	una	0000	0011011	011101	looll	00000	001(0(1	011101	ooll	110011	10011	00(1)
1011011	100110	101110	ווווווו	110111	00011	101101	110011	(100)	orilli	111011	100011	((0(1)	וווושט	11011	100011	100101	101010	0(1110
011010	110011	101100	010010	101101	0(1100	011010	110011	101100	010010	000111	110010	101100	9 (0010	000111	10010	000(1)	0100((1	٥١١١٥
101011	100011	100111	0/1011	100011	0011011	01011	100011	100111	011011	100(11	0011011	100111	0/1011)	100(11	0011011	mon	10011	101101
lloll	unu	0000	001(0(1	ارااه	looll	IIIOII	unu	00000	<i>0</i> 01(0(1	011101			001(0(1	Ollioj (10110	DILIOD	0111)
101101	110011	((0(1)	0011111	111011	1 <i>00</i> 01	101101	110011	(10(1)	0011111	111011	100011	((0(1)	oniii i	111011	00011	[001]	(0)111	0011)
011010	110011	101100	010010	000111	110010	011010	110011	101100	010010	000111	110010	101100	0 (0010	000111	110010	(1100	001111	(1(01)
101011	100011	100111	OHOU	100011	001/01/	(0(0)	10001	1 10011	Ollon	10001	1 00110	1 100111	0/1011)	(00()(001101	10011	0111(01	(0011)
ILIOII	unu	00000	0011011	011101	looll	1(1011	unu	0000	0 001(01	01110	1 100111	0000	001(0(1	011101	looll	9110	4141	رازاتان
101101	110011	(10(1)	orulii	111011	100011	10110	1 110011	11011	ווווטס	11101	100011	11011)	orilli	1(1011	100011	111000	01010	0)6)0
011010	110011	101100	01001	00011	110010	01101	ווסטון	10110	0 (1001	0001	110010	101100	010010	00011	110010	10111	00(1)	011)/
101011	10001	100111	0/1011)	(00)	00//0/	10101	10001	10011	011011	100(1	001101	1 100111	0/10/1	100011	0011011	1000	01)(0	(010)
ILIOII	una	0000	0011011	٥١١١٥	looll	111011	unu	00000	0011011	٥١١١٥	1 100111	0000	0011011	011101	looll	11100	00(1)	01110
10110	110011	((0(1)	0011111	1(1011	10001	10110	110011	1100	orilli	[[]]	100011	((0(1)	orilli	111011	100011	(000	0/1011	(0)0)
1(1011	una	0000	0011011	٥١١١٥	looill	111011	unu	0000	0 001101	01110	l looll	0000	0011011	011101	looll	1109	(100)	(010)
10110	110011	(10(1)	001111	1(1011	100011	10110	110011	11011	OUILLI	1(10)	100011	(1011)	orilli	[[]]	100011	(000(ana	OUN

Linear Probing and Double Hashing



h(k) = k mod 11
Records to store in the table
$(3,d_1)$ $(14,d_2)$ $(25,d_3)$ $(5,d_4)$ $(28,d_5)$ $(91,d_6)$

```
10
```

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

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

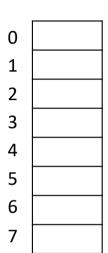
Linear probing:

$$h(k)$$
, $(h(k)+1) \mod M$, $(h(k) + 2) \mod M$, $((h(k) + 3) \mod M ...$

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 mod 8

Records to store in the table

$$(2,d_1)$$

$$(6,d_2)$$

$$(10,d_3)$$

```
Secondary hash function:

h'(k) = q - (k \mod q)

for some prime value q

h'(k) = 7 - (k \mod 7)
```

Double hashing:

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

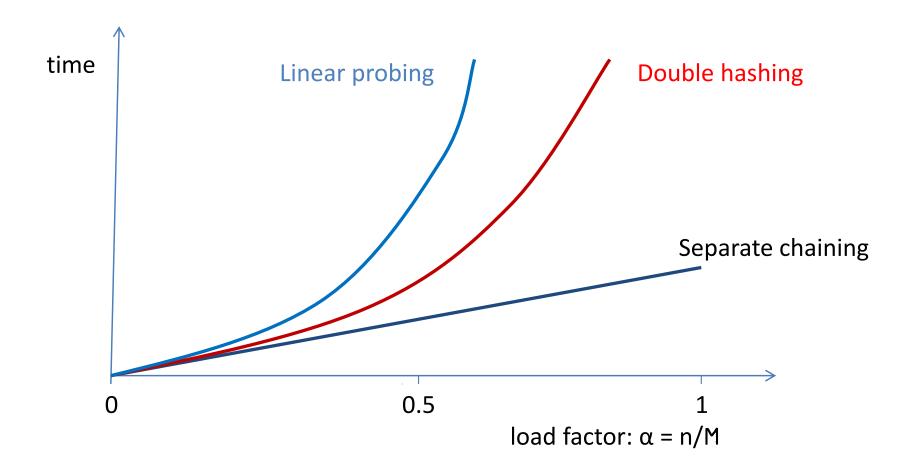
Open Addressing: put Method (linear probing)

```
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 \leftarrow h(k)
count \leftarrow 0
while (T[pos] != NULL) and (T[pos] != DELETED) do {
  if T[pos].getKey() = k then ERROR
  pos \leftarrow (pos + 1) \mod M
  count \leftarrow count + 1
  if count = Mthen ERROR
T[pos] \leftarrow (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 \leftarrow h(k)
count \leftarrow 0
while (T[pos] != NULL) and (T[pos] != DELETED) do {
  if T[pos].getKey() = k then ERROR
  pos \leftarrow (pos + h'(k)) mod M
  count \leftarrow count + 1
  if count = Mthen ERROR
T[pos] \leftarrow (k, data)
```

Average Time Complexity of get Operation



Separate chaining Linear Probing Double Hashing Average number of key comparisons

1 +
$$\alpha$$

 $\frac{1}{2}$ + $\frac{1}{(2(1 - \alpha)^2)}$
 $\frac{1}{(1 - \alpha)}$