BM267 - Introduction to Data Structures

14. Hashing

Ankara University

Computer Engineering Department

Bulent Tugrul

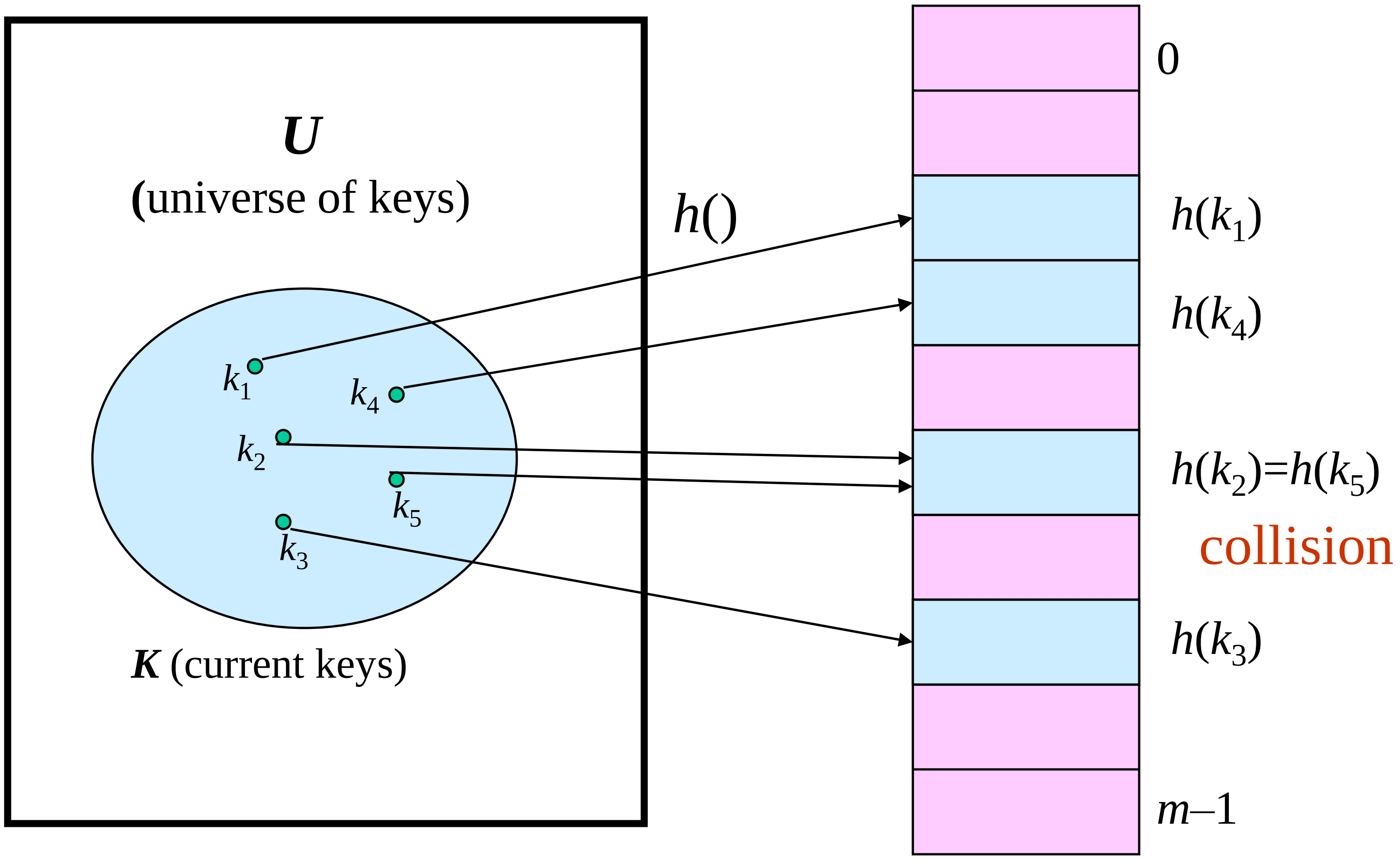
Hashing

- A dictionary is an abstract type. A set of operations searching, insertion, and deletion are defined on its element.
- The elements of this set can be of numbers, characters, character strings and so on.
- Typically, tables have several fields, each responsible for keeping a particular type of information about an entity.
- For example, a student record may contain fields for student's ID, name, date of birth and major and so on.

Hashing

- At least one field called **key** is used for identifying entities (the student's ID).
- Hashing is based on the idea of distributing ***n*** keys among a one-dimensional array H[0…m-1] called a hash table.
- The distribution is done by computing the value of some predefined functions ***h*** called hash function.
- This function assigns an integer between 0 and m-1, called hash address, to a key.

Hashing



U
(universe of keys)
k_1
k_4
k_2
k_5
k_3
K (current keys)
$h()$
0
$h(k_1)$
$h(k_4)$
$h(k_2)=h(k_5)$
collision
$h(k_3)$
$m-1$

Hashing

- The hash function depends on the key type.
- For example, if keys are positive integers, hash function can be of the form ***`h(k)= k mod m.`*** This function assures that the result is always between 0 and m-1.
- In general, hash function needs to satisfy two requirement:
  - ➢ A hash function needs to distribute keys among the cells of the hash table as evenly as possible. (Because of this requirement the value of m is usually chosen to be prime.)
  - ➢ A hash function has to be easy to compute

Hashing

- If we choose a hash table's size m to be smaller than the number of keys n, we will get collisions (two (or more) keys being hashed into same cell of the hash table).
- In fact, in the worst case, all the keys could be hashed to the same cell of the hash table.
- With an appropriately chosen size of the hash table and a good hash function, this situation will happen rarely.
- Still, every hashing scheme must have a collision resolution mechanism.
- There are two version of hashing: open hashing (separate chaining) and closed hashing (open addressing)

Collisions

- The goal of the hash function is to distribute the keys in an apparently in a random way to prevent mapping different keys to the same table index.

- Total prevention is not possible in practice.

- When $\boldsymbol{h}(\boldsymbol{x}) = \boldsymbol{h}(\boldsymbol{y})$ for $\boldsymbol{x} \neq \boldsymbol{y}$, this is called a **collision**
- Collisions occur when different elements are mapped to the same cell.

Collisions

- As the number of elements in the table increases, the likelihood of a *collision* increases - so make the table **as large as possible.**
- If the table size is 100, and all the hashed keys are divisible by 10, there will be many collisions!
    - Particularly bad if table size is a power of a small integer such as 2 or 10
- Therefore, make the table size **a prime number**

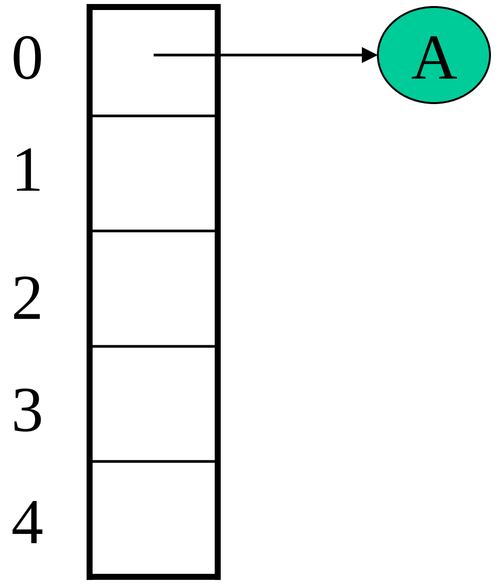
Collisions handling

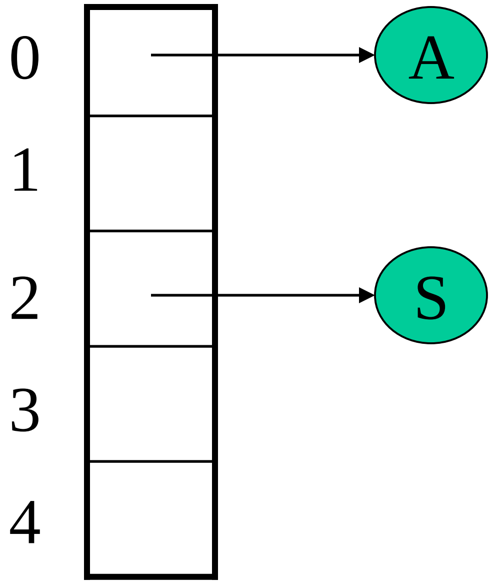
- Open Hashing (Separate chaining): Create an array of linked list, so that the item can be inserted into the linked list if collision occurs.
- Closed Hashing (Open Addressing): Search the array in some systematic way for an empty cell and insert the new item there if collision occurs.

Open Hashing (Separate Chaining)

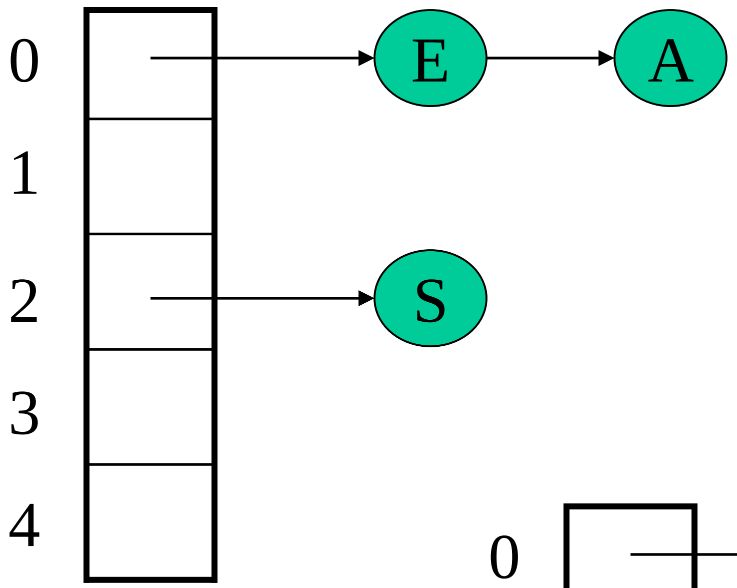
- In open hashing, keys are stored in linked list attached to cells of a hash table.

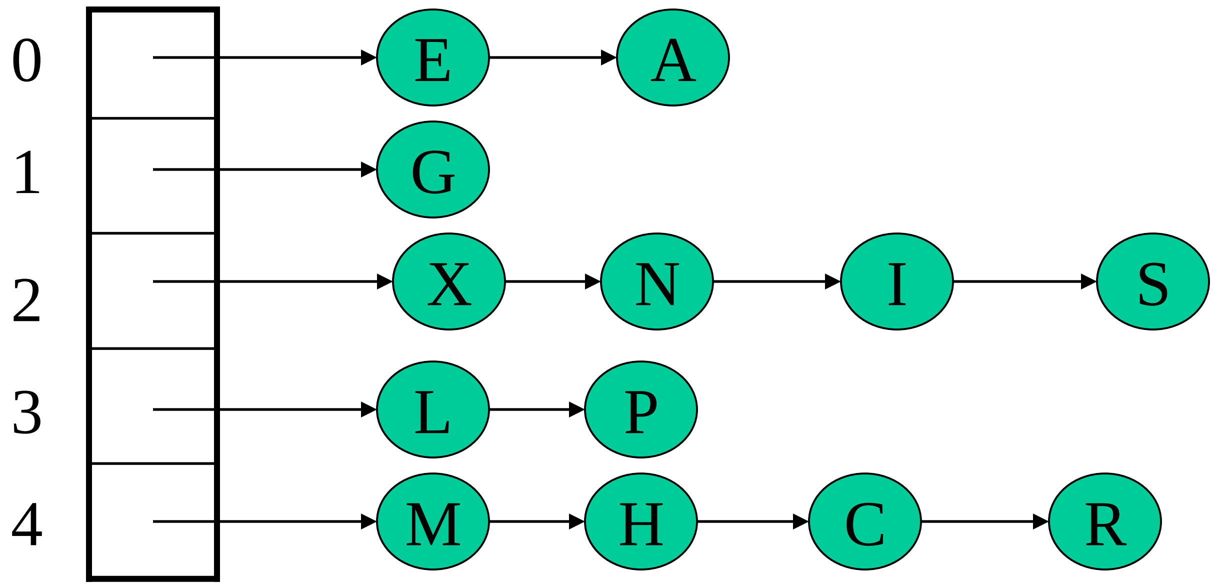
keys	A	S	E	R	C	H	I	N	G	X	M	P	L
hash value	0	2	0	4	4	4	2	2	1	2	4	3	3





Open Hashing (Separate Chaining)





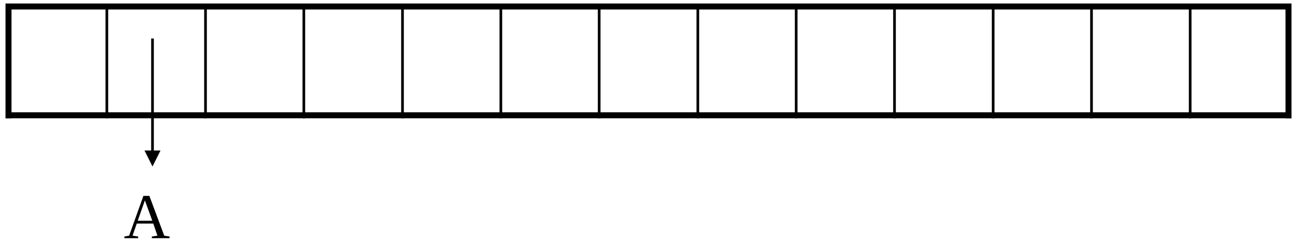
Open Hashing (Separate Chaining)

- Consider , as an example, the following list of words:

A, FOOL, AND, HIS MONEY, ARE, SOON, PARTED

- As a hash function, we will use the simple function for strings, that is we will add the positions of a word's letters in the alphabet and compute the sum's remainder after division by 13.
- We start with an empty table.
- The first key is the word "A"; its hash is

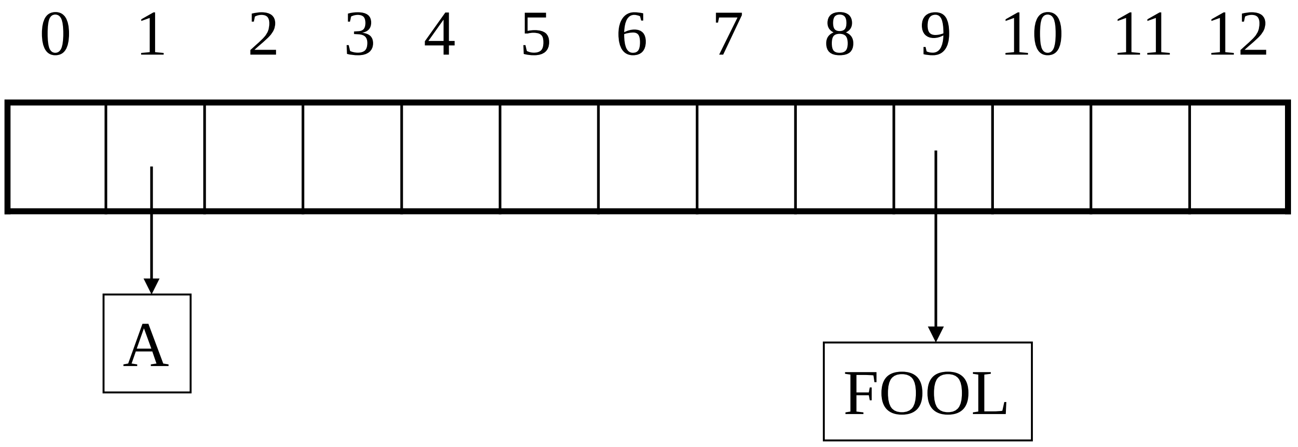
$$h(A) = 1 \bmod 13 = 1$$

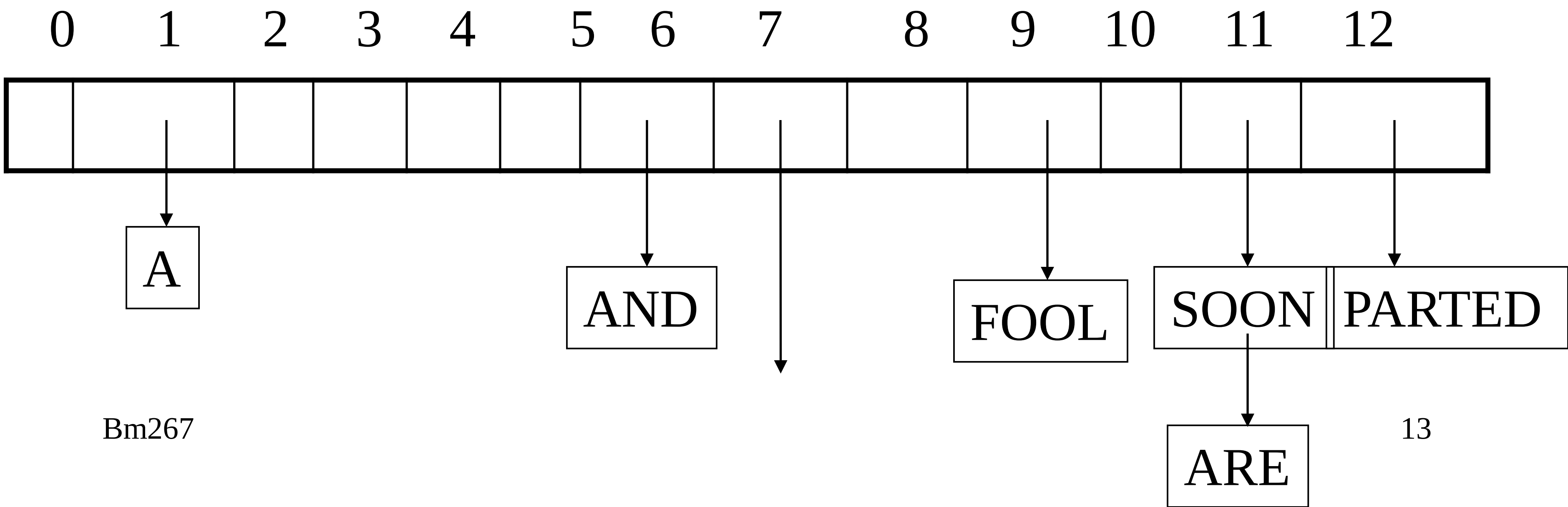


Open Hashing (Separate Chaining)

- The second key –the word FOOL; its hash is

  h(FOOL) = (6 +15 +15+12) mod 13 = 9





Open Hashing (Separate Chaining)

• A collision occurred at position 11, because h(ARE) = (1+8+5) mod 13 = 11 and h(SOON) = (19+15+15+14) mod 13 = 11.

Searching

•Apply same procedure to search key that was used for creating the table.

•If we want to search a key “KID” in the hash table, we first compute the value of the same hash function for the key: h(KID) = 11.

•Since the list attached to cell 11 is not empty, so the list may contain the key. After comparing the string KID first with SOON and then ARE, we end up with an unsuccessful search.

Open Hashing (Separate Chaining)

- The efficiency of searching depends on the length of the list.
- The length of the list depends on the table size and the quality of the hash function.
- If the hash function distributes n keys among m cells of the hash table evenly, each list will be about ***n/m*** keys long.
- The ratio α = ***n/m*** called the **load factor** of the hash table.
- The average number of pointers inspected in successful search will be $1 + \alpha/2$. In unsuccessful search, it will be α.

Open Hashing (Separate Chaining)

- The other two dictionary operations – insertion and deletion are almost identical to searching.
- Insertion are done to the front or end of the list.
- Deletion is performed by searching for a key to be deleted and then removing it from the its list.

Closed Hashing(Open Addressing)

- In closed hashing, all keys are stored in the hash table itself without the use of linked list.
- This implies that the table size m must be at least as large as the number of keys n.
- Different strategies can be employed for collision resolution. The simplest one – called ***linear probing***- checks the cell following the one where collision occurs. If that cell is empty, the new key is saved there, if not the first empty cell is searched. If the end of the table is reached, the search starts from the beginning of the table. The table is treated as a circular array.

Linear Probing

keys	A	FOOL	AND	HIS	MONEY	ARE	SOON	PARTED
hash ad	1	9	6	10	7	11	11	12

0	1	2	3	4	5	6	7	8	9	10	11	12
	A											
	A								FOOL			
	A					AND			FOOL			
	A					AND			FOOL	HIS		
	A					AND	MONEY		FOOL	HIS		
	A					AND	MONEY		FOOL	HIS	ARE	
	A					AND	MONEY		FOOL	HIS	ARE	SOON
PARTED	A					AND	MONEY		FOOL	HIS	ARE	SOON

Linear Probing

- To search for a given key k, we start by computing *h(k)* where h is the hash function used in the table's construction.
- If the cell is empty the search is unsuccessful.
- If the cell is not empty, we must compare k with the cell: if they are equal, we have found a matching key; if they are not we compare k with a key in the next cell and continue in this manner until either we encounter a matching key or an empty cell (unsuccessful search)
- For example, if we search the word LIT in the table, first we need to calculate the hash value. We will get h(LIT) = (12 + 9 +20) mod 13 = 2 , since cell is empty the search is unsuccessful. We can stop immediately

Linear Probing

- However if we search for KID with h(KID) = (11+9+4) mod 13 = 11, we will compare KID with ARE, SOON, PARTED, and, A before we can declare the search is unsuccessful.
- While the search and insertion operations are straightforward, deletion is not.
- For example, if we simply delete the key ARE from the table, we would unable to to find key SOON afterward.
- After computing h(SOON) = 11, the algorithm would find this location empty and report the search is unsuccessful.
- The solution is to mark previously occupied locations by a special symbol to distinguish them from locations that have not been occupied.

Linear Probing

- Linear probing creates clusters. Clusters are bad news in hashing because they make operations less efficient.

- Several other collision resolution strategies have been suggested; Quadratic and Double hashing.
- **Quadratic probing** uses a hash function of the form

$$h(k, i) = (h'(k) + c_1 i + c_2 i^2) \bmod m$$

where h' is an auxiliary hash function, c_1 and c_2 are auxiliary constants, and $i = 0, 1, \ldots, m-1$.

Double Hashing

- Double Hashing is one of the best methods available for open addressing.
- Double hashing uses a hash function of the form

$$h(k,i) = (h_1(k) + ih_2(k)) \bmod m$$

where h_1 and h_2 are auxiliary hash functions.

- The initial probe is to position $h_1(k)$; successive probe positions are offset from previous positions by the amount $h_2(k) \bmod m$.
- The $h_2(k)$ must be relatively prime to the hash-table size m.
- A convenient way to ensure this condition, let m be prime and to design h_2 so that it always returns a positive integer less than m.

Double Hashing

- For example, we could choose m prime and let

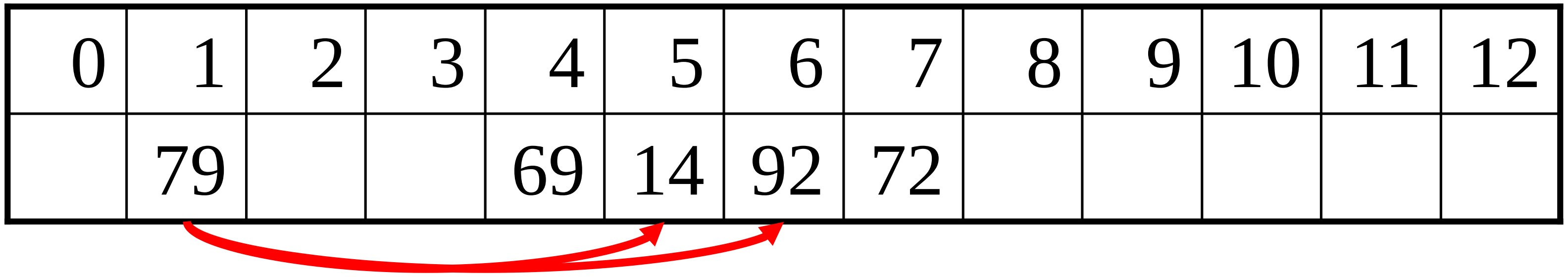
$$h_1(k) = k \bmod m,$$

$$h_2(k) = 1 + (k \bmod m')$$

where m' is chosen to be slightly less than (say m-1).

- If k =123456, m = 701 and m' =700, we have $h_1(k) = 80$ and $h_2(k) = 257$ so the first probe is at position 80, and then every 257th slot is examined.

Double Hashing



0	1	2	3	4	5	6	7	8	9	10	11	12
	79			69	14	92	72					

$h_1(k) = k \bmod 13$

$h_2(k) = 1 + (k \bmod 11)$