COMP 2611, DATA STRUCTURES
LECTURE 16

HASHING TECHNIQUES

- Deleting a Key
- Resolving Collisions

Hashing

- Use *hash* function, *h*, to convert *key* to a valid location in the array. When this is done, the array is called a *hashtable*.

- For example, h = key % 12 + 1

- Where will 52 hash to?

- 52 will hash to, 52 % 12 + 1 = 5. Location 5 is empty, so 52 is inserted in location 5.

				52							
1	2	3	4	5	6	7	8	9	10	11	12

Empty Locations

- Initialize the elements of the hashtable with a value that indicates empty. If the keys are positive integers, 0 can be used to indicate empty.

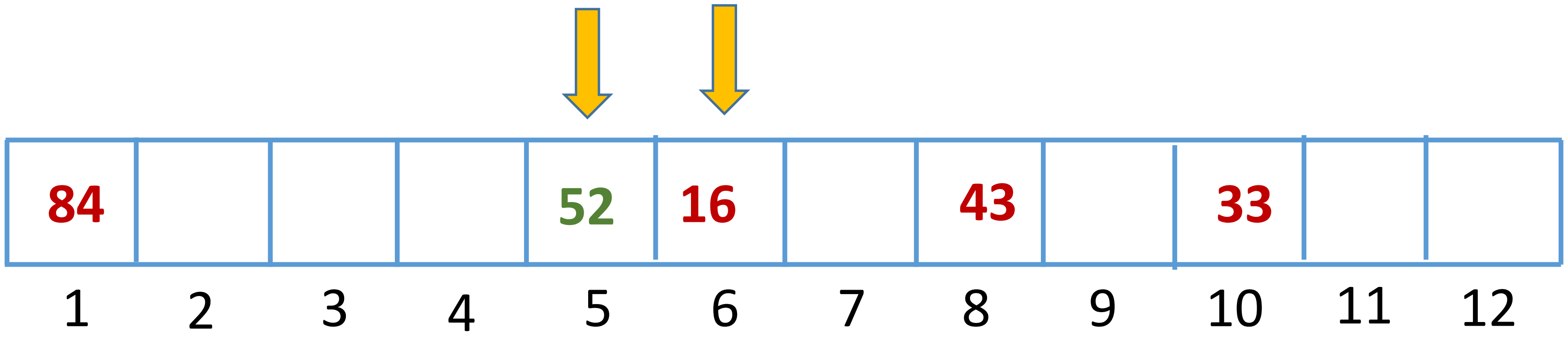
Hashing

33 → 33 % 12 + 1 = 10

84 → 84 % 12 + 1 = 1

43 → 43 % 12 + 1 = 8

- Insert 33, 84, and 43.
- Insert 16.
- 16 % 12 + 1 = 5. But, location 5 already has 52.
- This is referred to as a *collision*.
- Where to insert 16?



Hashing

- Insert 59 and 31.
- 59 % 12 + 1 = 12. Location 12 is empty. So, 59 is placed in location 12.
- 31 % 12 + 1 = 8. But, location 8 already has 43. Collision!
- We try the next location, 9. It is empty so 31 is placed in location 9,

84				52	16		43	31	33		59
1	2	3	4	5	6	7	8	9	10	11	12

Hashing

- Insert 23.

- 23 % 12 + 1 = 12. Location 12 is occupied. So, we try the next location.

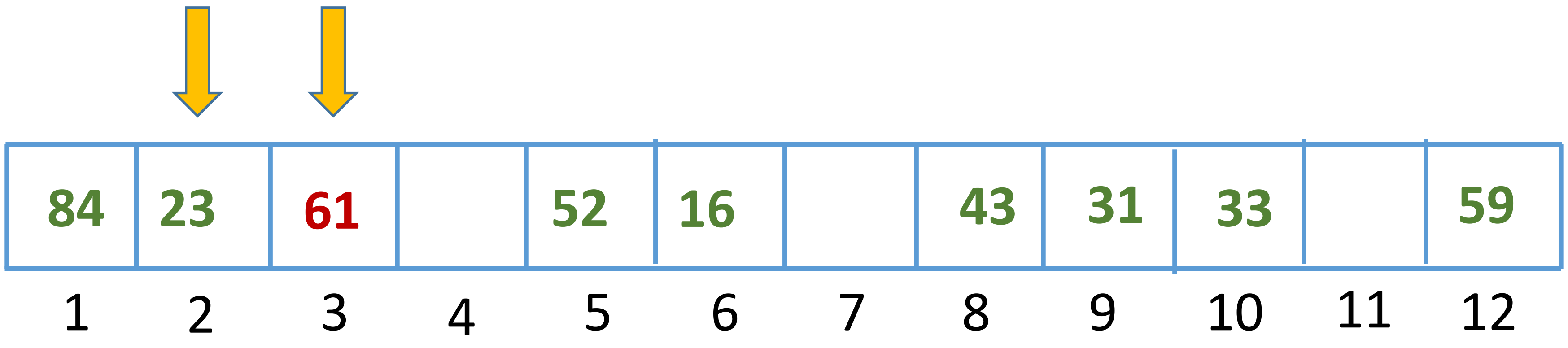
- Treat the table as circular so the next location is 1. But, location 1 is occupied. So, we try the next location, location 2.

- Location 2 is empty so 23 is placed there.

84	23			52	16		43	31	33		59
1	2	3	4	5	6	7	8	9	10	11	12

Hashing

- Insert 61.
- 61 % 12 + 1 = 2. Location 2 is occupied. So, we try the next location.
- Location 3 is empty so 61 is placed there.



Searching for a Key

- Let’s search for 23.
- 23 % 12 + 1 = 12. Location 12 is occupied but not by 23.
- We try the next location, 1. Location 1 is occupied but not by 23
- We try the next location 2. Location 2 contains the key we are looking for.

How would we know if the table does NOT contain the key?

84	23	61		52	16		43	31	33		59
1	2	3	4	5	6	7	8	9	10	11	12

Deleting a Key

- Once a key is found in the hashtable it can be deleted by assigning a value that signifies "deleted". For example, if the keys are positive integers, -1 can be placed in a deleted location.

- For example, let's delete 84.

84	23	61		52	16		43	31	33		59
1	2	3	4	5	6	7	8	9	10	11	12

Dealing with a Deleted Key

- When searching for a key (e.g., 23), a deleted key is treated as if it is a valid key.
- When inserting a key, it can be inserted in the first empty location or in the first deleted location.
- For example, 71 can be inserted in Location 1.

-1	23	61		52	16		43	31	33		59
1	2	3	4	5	6	7	8	9	10	11	12

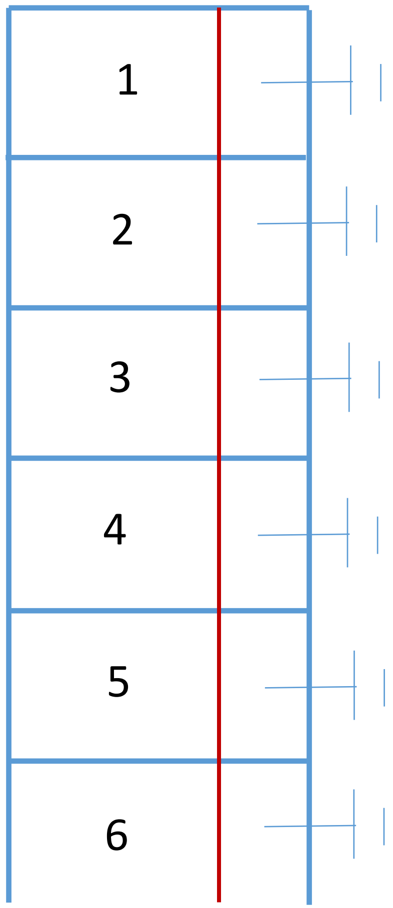
Resolving Collisions

- Linear probing (*location* = *location* + 1) ✓
- Chaining
- Quadratic probing
- Linear probing with double hashing

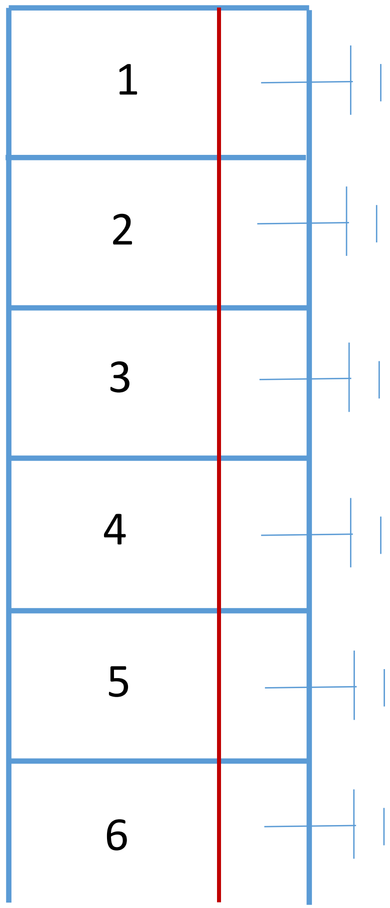
Chaining

- All keys that hash to the same location are held on a linked list.

Hashtable with 12 Locations



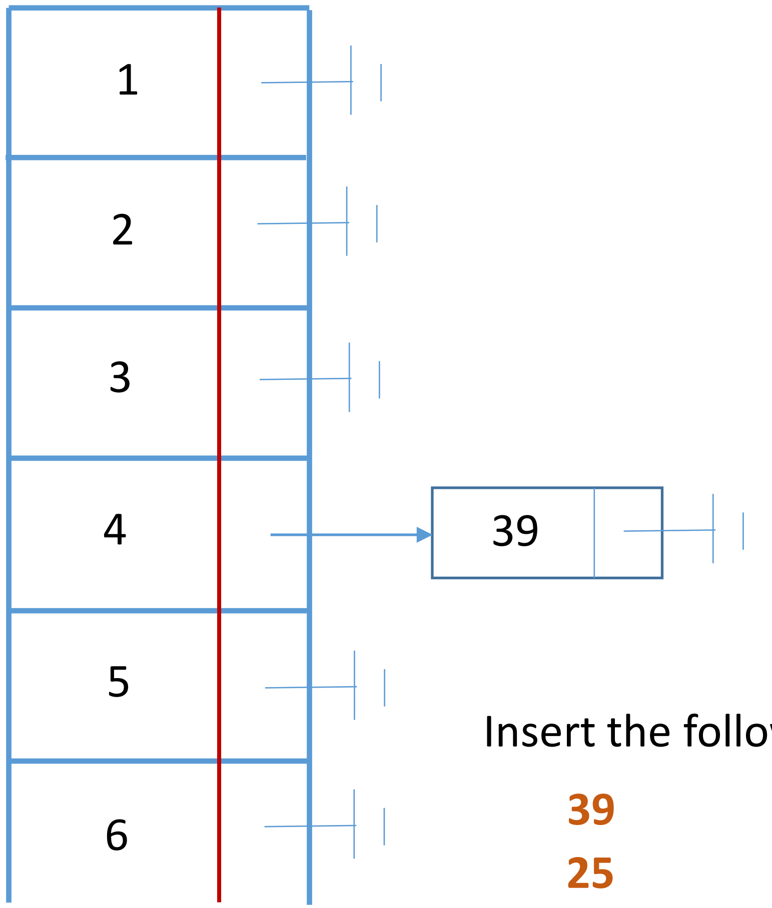
Hashtable with 12 Locations



Insert the following keys:

39

Hashtable with 12 Locations

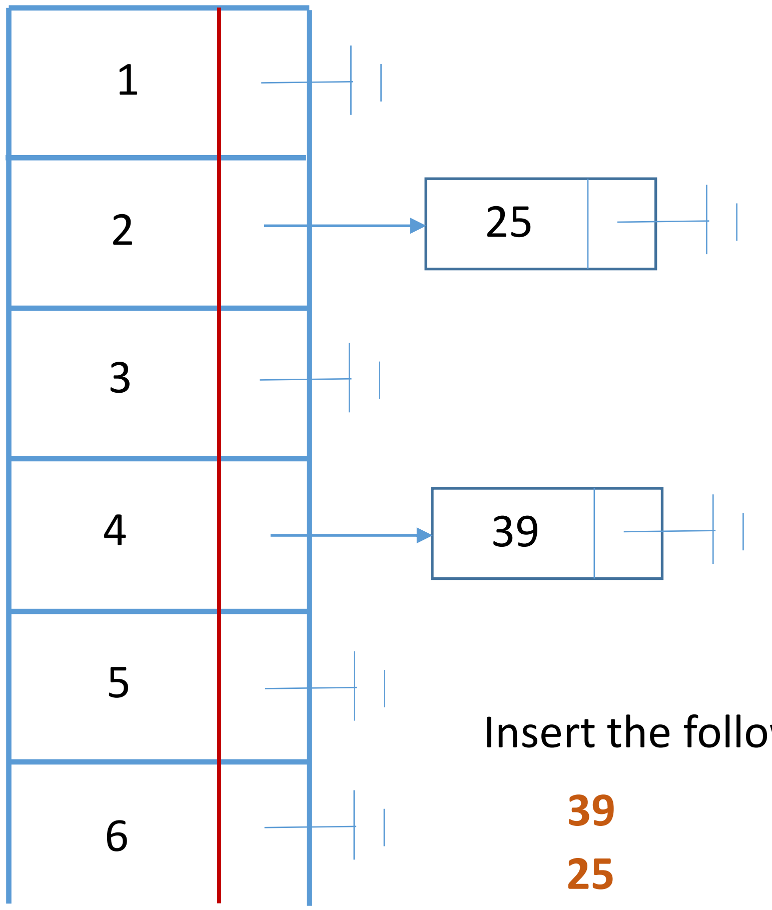


Insert the following keys:

39

25

Hashtable with 12 Locations



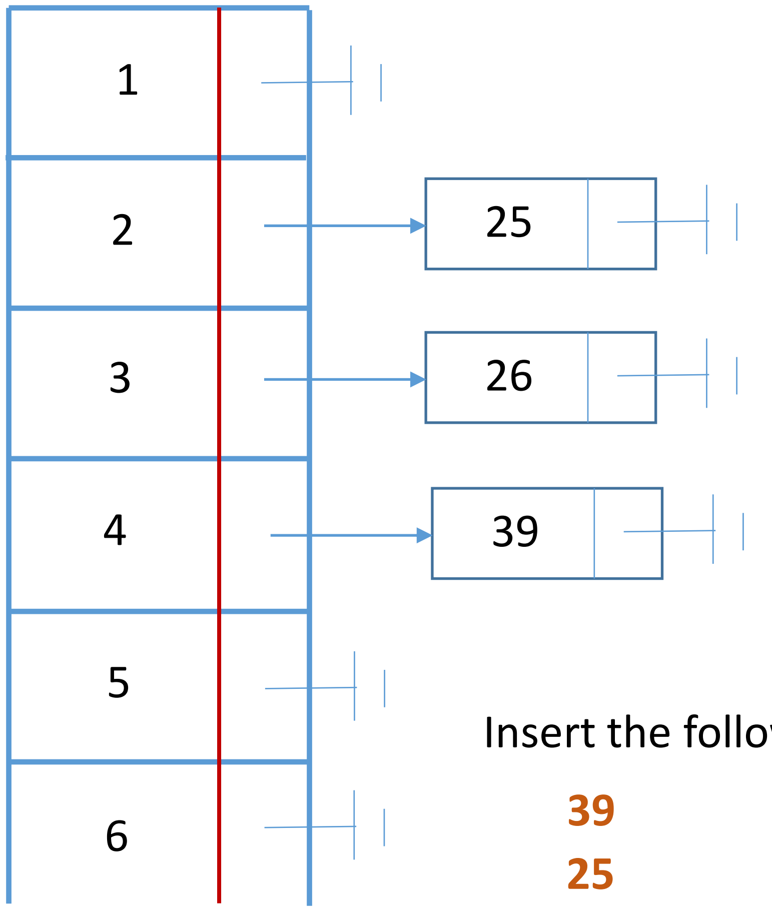
Insert the following keys:

39

25

26

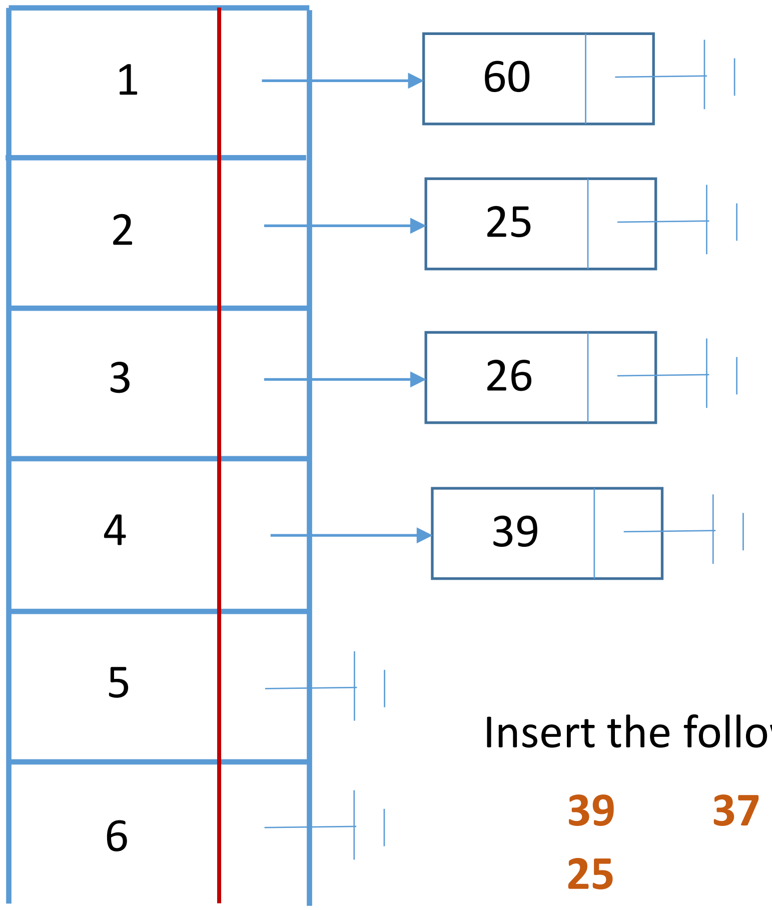
Hashtable with 12 Locations



Insert the following keys:

39
25
26
60

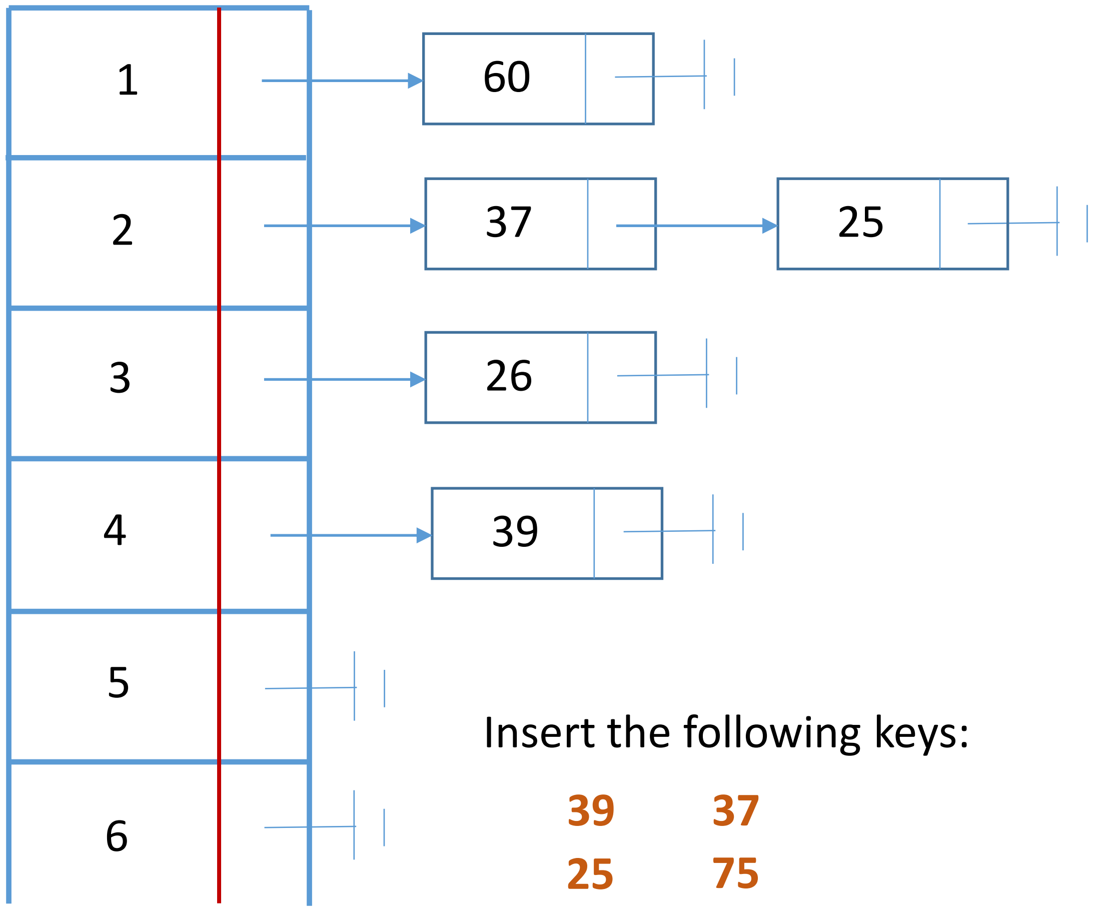
Hashtable with 12 Locations



Insert the following keys:

39 37
25
26
60

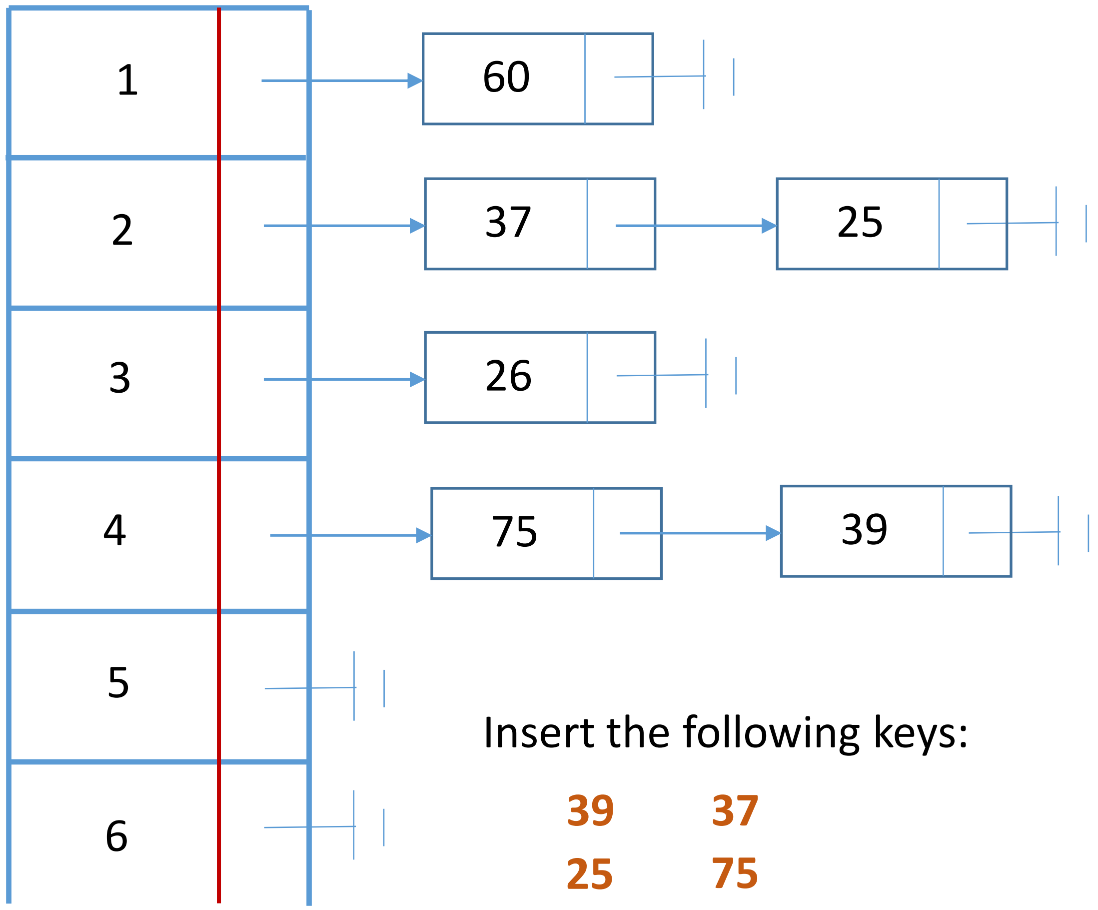
Hashtable with 12 Locations



Insert the following keys:

39 37
25 75
26
60

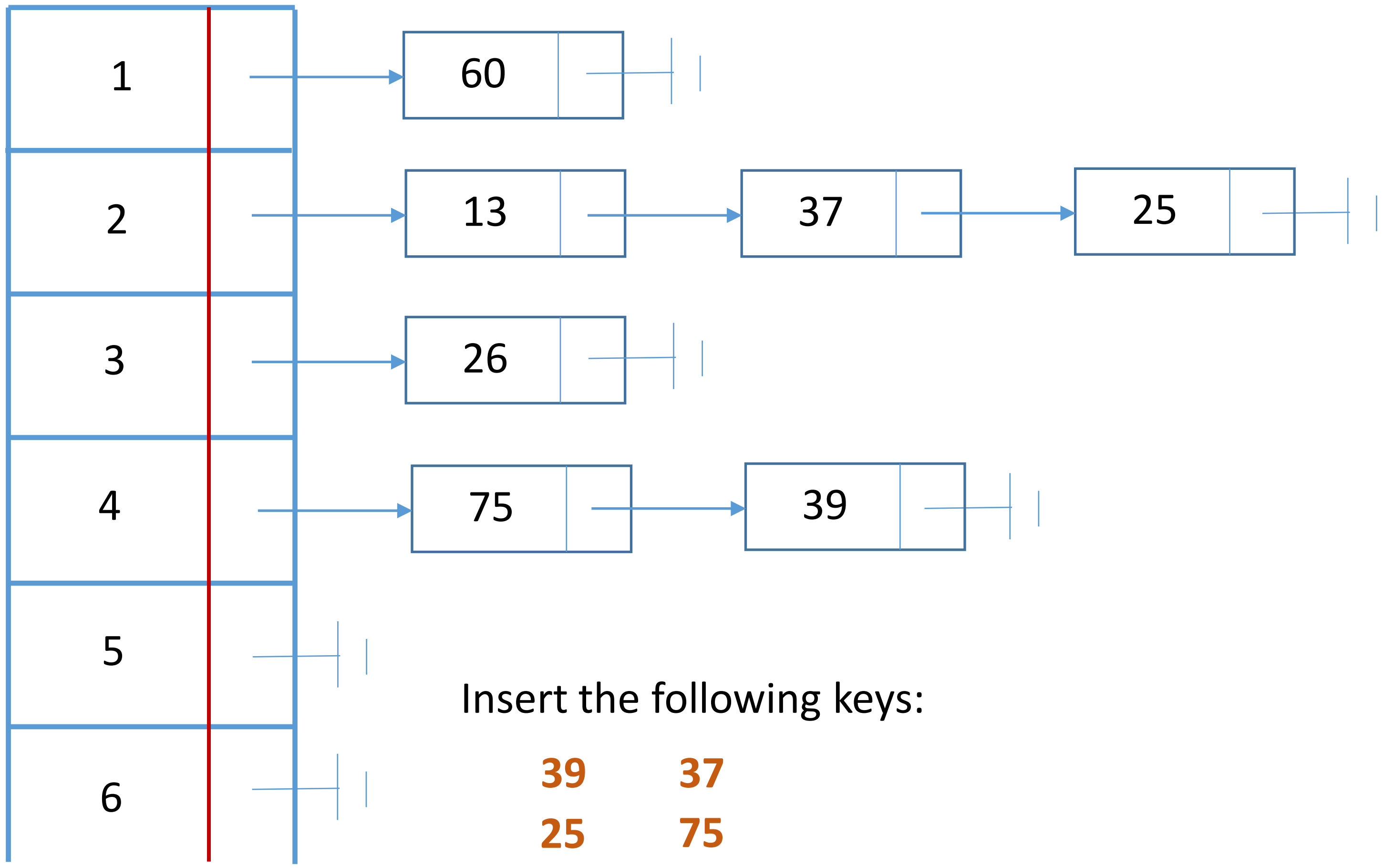
Hashtable with 12 Locations



Insert the following keys:

39	**37**
25	**75**
26	13
60	

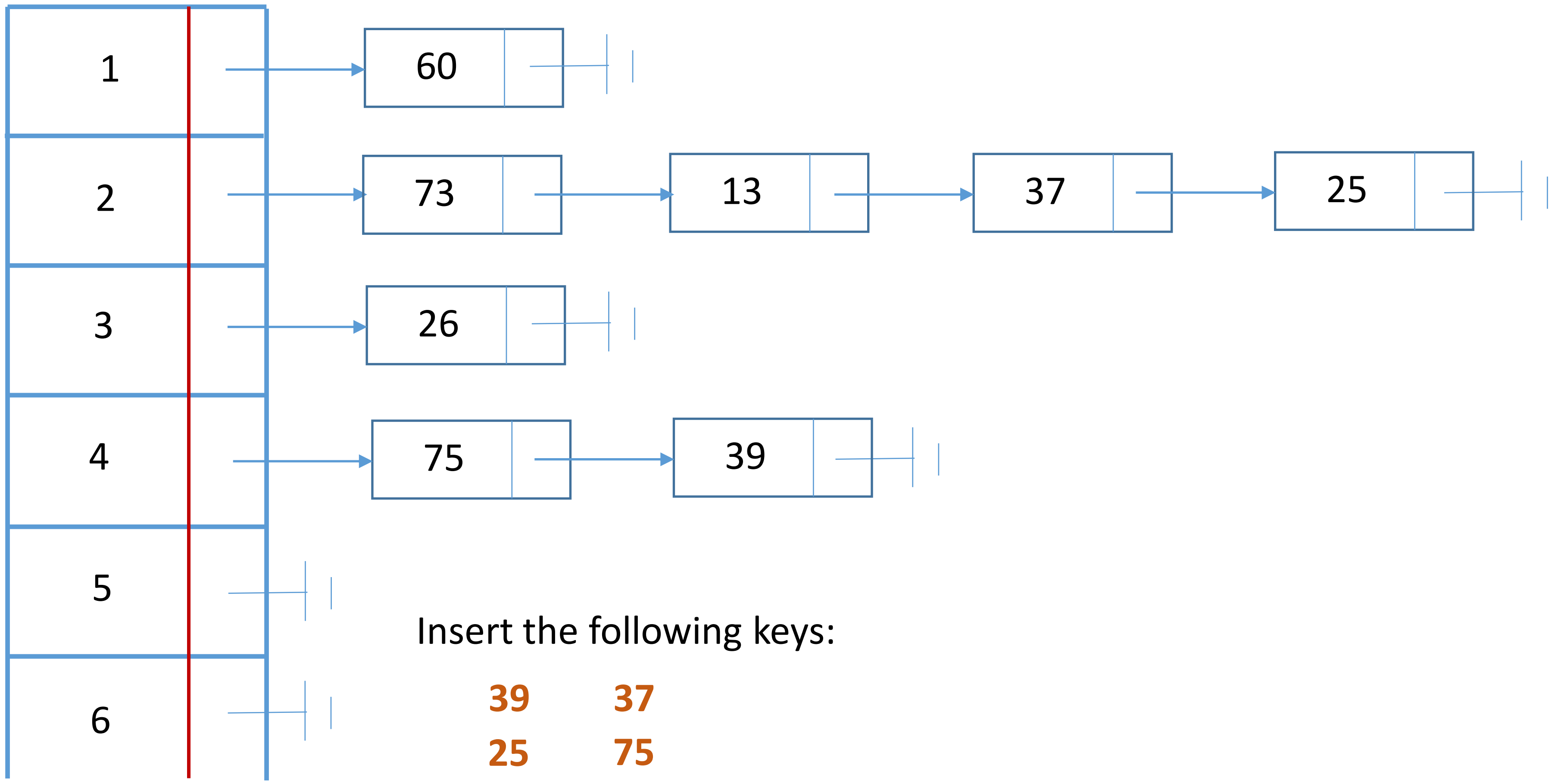
Hashtable with 12 Locations



Insert the following keys:

39	37
25	75
26	13
60	73

Hashtable with 12 Locations



Insert the following keys:

39	37
25	75
26	13
60	73

Quadratic Probing

- Suppose an incoming key collides with another at location *loc*.

- Instead of going forward by 1 (linear probing), we go forward by $ai + bi^2$ where a and b are constants and i takes on the value 1 for the first collision, 2 if there is a second collision, 3 if there is a third collision, etc.

- Suppose a is 1 and b is 1. The quadratic function becomes $i + i^2$.

Original Example Using Quadratic Probing

- Insert 52.

- 52 hashes to, 52 % 12 + 1 = 5. Location 5 is empty, so 52 is inserted in location 5.

				52							
1	2	3	4	5	6	7	8	9	10	11	12

Example of Quadratic Probing

- Insert 33, 84, and 43.
- Insert 16.
- 16 % 12 + 1 = 5. But, location 5 already has 52.
- This is the first collision (i = 1).
- The quadratic function is: $i + i^2 = 1 + 1 = 2$. So, move forward 2 locations:

84				52		16	43		33		
1	2	3	4	5	6	7	8	9	10	11	12

Example of Quadratic Probing

- Insert 59 and 31.
- 59 % 12 + 1 = 12. Location 12 is empty. So, 59 is placed in location 12.
- 31 % 12 + 1 = 8. But, location 8 already has 43. This is the first collision (i = 1).
- The quadratic function is: $i + i^2 = 1 + 1 = 2$. So, move forward 2 locations:

84				52		16	43		33		59
1	2	3	4	5	6	7	8	9	10	11	12

Example of Quadratic Probing

- Now location 10 already has 33. This is the second collision ($i = 2$).
- The quadratic function is: $i + i^2 = 2 + 4 = 6$. So, move forward 6 locations:

84			**31**	**52**		**16**	**43**		**33**		**59**
1	2	3	4	5	6	7	8	9	10	11	12

Example of Quadratic Probing

- Insert 23.

- 23 % 12 + 1 = 12. Location 12 is occupied. This is the first collision (i = 1).

- The quadratic function is: $i + i^2 = 1 + 1 = 2$. So, move forward 2 locations. So, we try the next location, location 2.

- Location 2 is empty so 23 is placed there.

84	23		31	52		16	43		33		59
1	2	3	4	5	6	7	8	9	10	11	12

Example of Quadratic Probing

- Insert 61.
- 61 % 12 + 1 = 2. Location 2 is occupied. This is the first collision (i = 1).
- The quadratic function is: $i + i^2 = 1 + 1 = 2$. So, move forward 2 locations. So, we try the next location, location 4.
- Now location 4 already has 31. This is the second collision (i = 2).
- The quadratic function is: $i + i^2 = 2 + 4 = 6$. So, move forward 6 locations:

84	23		31	52		16	43		33		59
1	2	3	4	5	6	7	8	9	10	11	12

Linear Probing with Double Hashing

- Suppose an incoming key collides with another at location *loc*.

- Instead of going forward by 1 (linear probing), we go forward by *k,* where *k* varies with the key.

- It is generally better to use a prime number for the table size, *n*. If (*n* - 2) is also a prime number, we can find *k* as follows:

$$k = key \% (n - 2) + 1$$

Example of Linear Probing with Double Hashing

- Insert 33, 84, and 43.
- Insert 16.
- 16 % 12 + 1 = 5. But, location 5 already has 52.
- Calculate k = 16 % 10 = 6.
- So, move forward 6 locations:

84				**52**			**43**		**33**	**16**	
1	2	3	4	5	6	7	8	9	10	11	12

Issues with Hashing

- What if the hashtable runs out of space?

- What to do with non-numeric keys (e.g., strings)

```
int i, intValue;

i = intValue = 0;

while (str[i] != '\0') {
        intValue = intValue + str[i];
        i++;
}

location = intValue % n + 1;
```

Monitor hashtable and resize.

Words that have the same letters hash to the same location, e.g., mate, meat, team.

Assign weight to each character depending on its position.