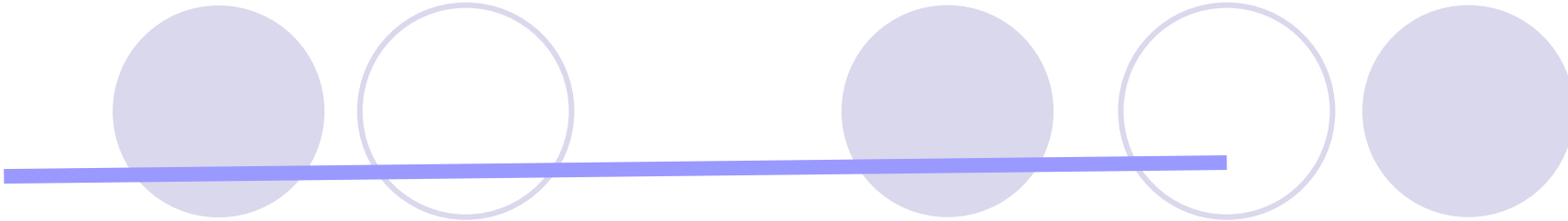


CSC2100B

Tutorial 6 Hashing

Tom Chao Zhou

Hashing - overview

- Hash function
- Collision resolution
  - Separate Chaining (Open hashing)
  - Open addressing (Closed Hashing)
    - Linear probing
    - Quadratic probing
    - Double hashing

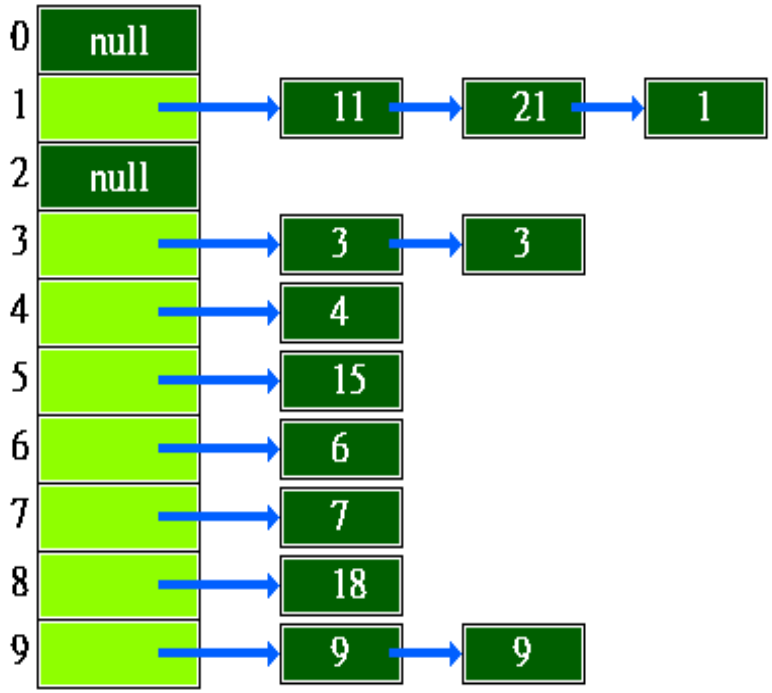
Hashing - hash function

- Hash function
  - A mapping function that maps a key to a number in the range *0* to *TableSize -1*

```
int hashfunc(int integer_key)
{
    return integer_key % HASHTABLESIZE;
}
```

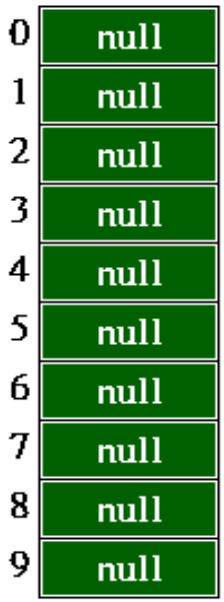
Hashing - separate chaining

- If two keys map to same value, the elements are chained together.



Hashing - example

- Insert the following four keys 22 84 35 62 into hash table of size 10 using separate chaining.
- The hash function is key % 10

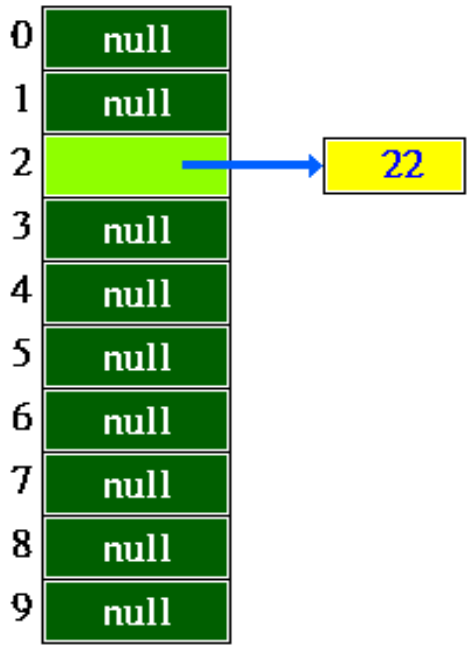


Initial hash table

Hashing - example

- Insert the following four keys 22 84 35 62 into hash table of size 10 using separate chaining.
- The hash function is key % 10

22 % 10 = 2

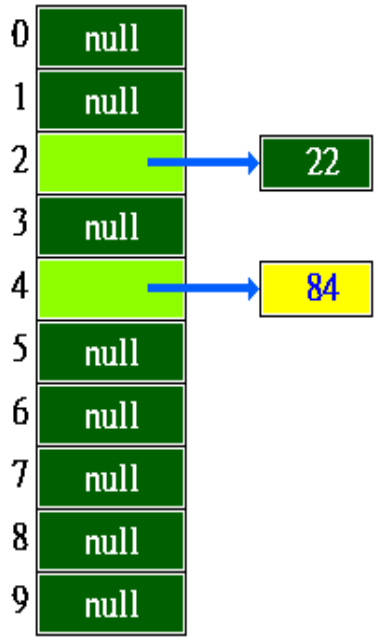


After insert 22

Hashing - example

- Insert the following four keys 22 84 35 62 into hash table of size 10 using separate chaining.
- The hash function is key % 10

84 % 10 = 4

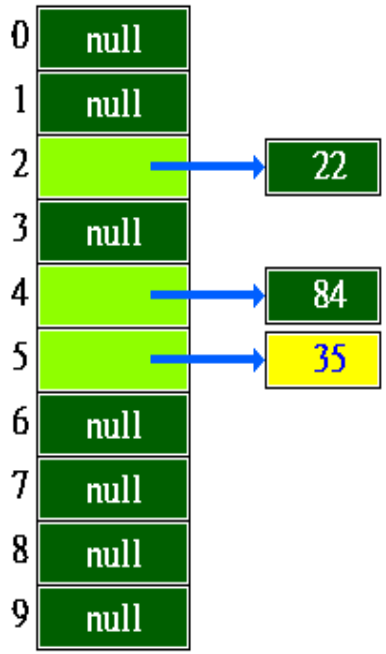


After insert 84

Hashing - example

- Insert the following four keys 22 84 35 62 into hash table of size 10 using separate chaining.
- The hash function is key % 10

35 % 10 = 5

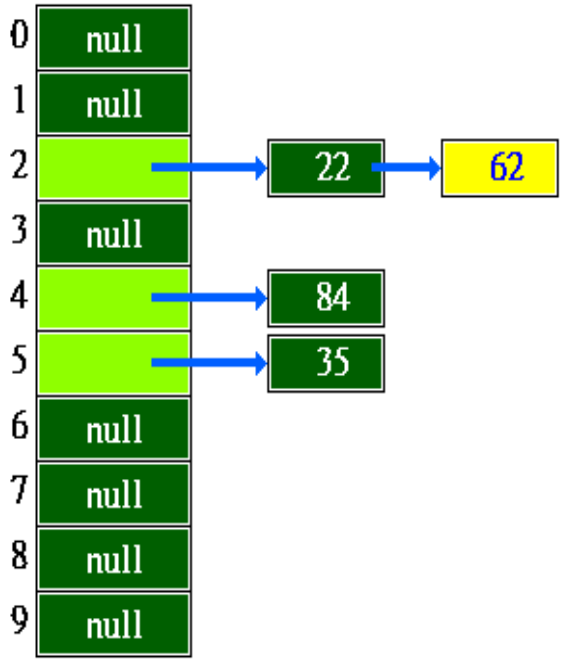


After insert 35

Hashing - example

- Insert the following four keys 22 84 35 62 into hash table of size 10 using separate chaining.
- The hash function is key % 10

$62 \% 10 = 2$



After insert 62

Hashing

- Open addressing
  - Open addressing hash tables store the records directly within the array.
  - A hash collision is resolved by *probing*, or searching through alternate locations in the array.
    - Linear probing
    - Quadratic probing
    - Random probing
    - Double hashing

Hashing - Open addressing

```
#define HASHTABLESIZE 51

typedef struct
{
    int key[HASHTABLESIZE];
    int state[HASHTABLESIZE];
   /* -1=lazy delete, 0=empty, 1=occupied */
} hashtable;

/* The hash function */
int hash(int input)
{
    return input % HASHTABLESIZE;
}
```

Hashing - Open addressing

- Open addressing
  - if collision occurs, alternative cells are tried.
  - $h_0(X)$, $h_1(X)$, $h_2(X)$, ...

$$h_i(X) = (\text{Hash}(X) + F(i)) \bmod \textit{TableSize}$$

  - Linear probing : $F(i) = i$
  - Quadratic probing : $F(i) = i^2$
  - Double hashing : $F(i) = i * \text{Hash}_2(X)$

Hashing - Open addressing

```
void open_addressing_insert(int item, hashtable * ht )
{
     int hash_value;
     hash_value = hash(item);
     i = hash_value;
     while (ht->state[i]== 1)
     {
            if (ht->key[i] == item) {
                   fprintf(stderr,”Duplicate entry\n”);
                   exit(1);
            }
            i = (i + F(i)) % HASHTABLESIZE;
            if (i == hash_value) {
                   fprintf(stderr, “The table is full\n”);
                   exit(1);
            }
     }
     ht->key[i] = item;
     ht->state[i] = 1;
}
```

Hashing - Open addressing

- Linear probing
  - F(i) = i

$h_i(X) = (\text{Hash}(X) + i) \bmod \textit{TableSize}$

$h_0(X) = (\text{Hash}(X) + 0) \bmod \textit{TableSize}$,
$h_1(X) = (\text{Hash}(X) + 1) \bmod \textit{TableSize}$,
$h_2(X) = (\text{Hash}(X) + 2) \bmod \textit{TableSize}$, ...

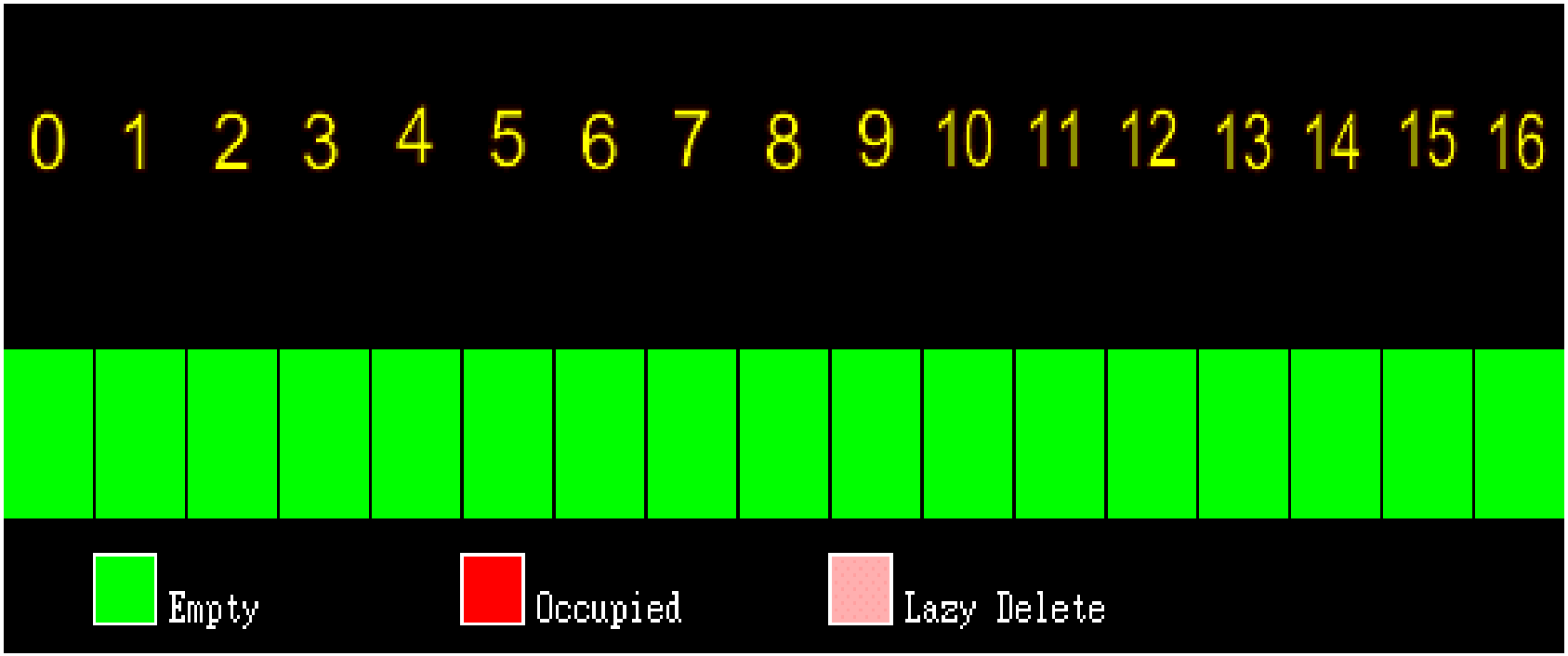
Hashing - Open addressing

- Linear probing

```
/* The h function */
int h(int i, int input)
{
  return (hash(input) + i)% HASHTABLESIZE;
}
```

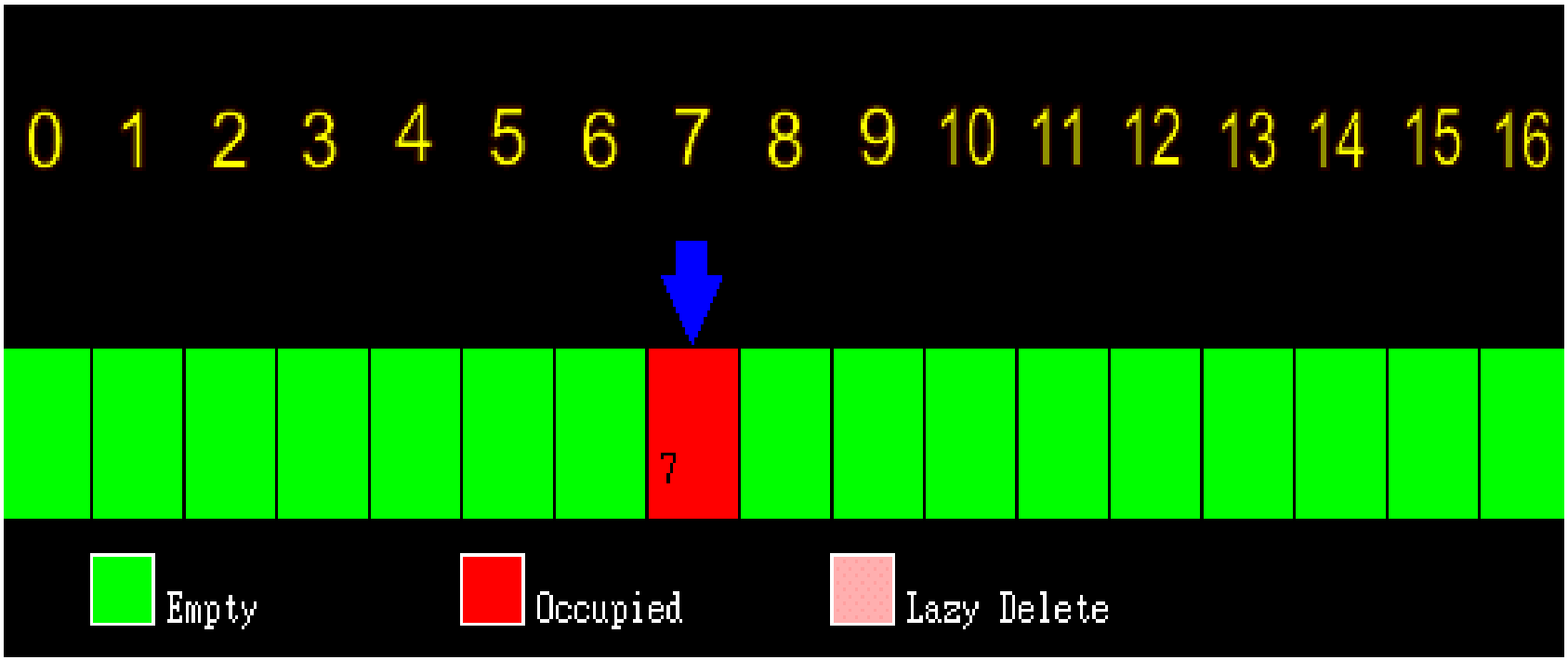
Hashing - Open addressing

- Linear probing example
  - Initial hash table



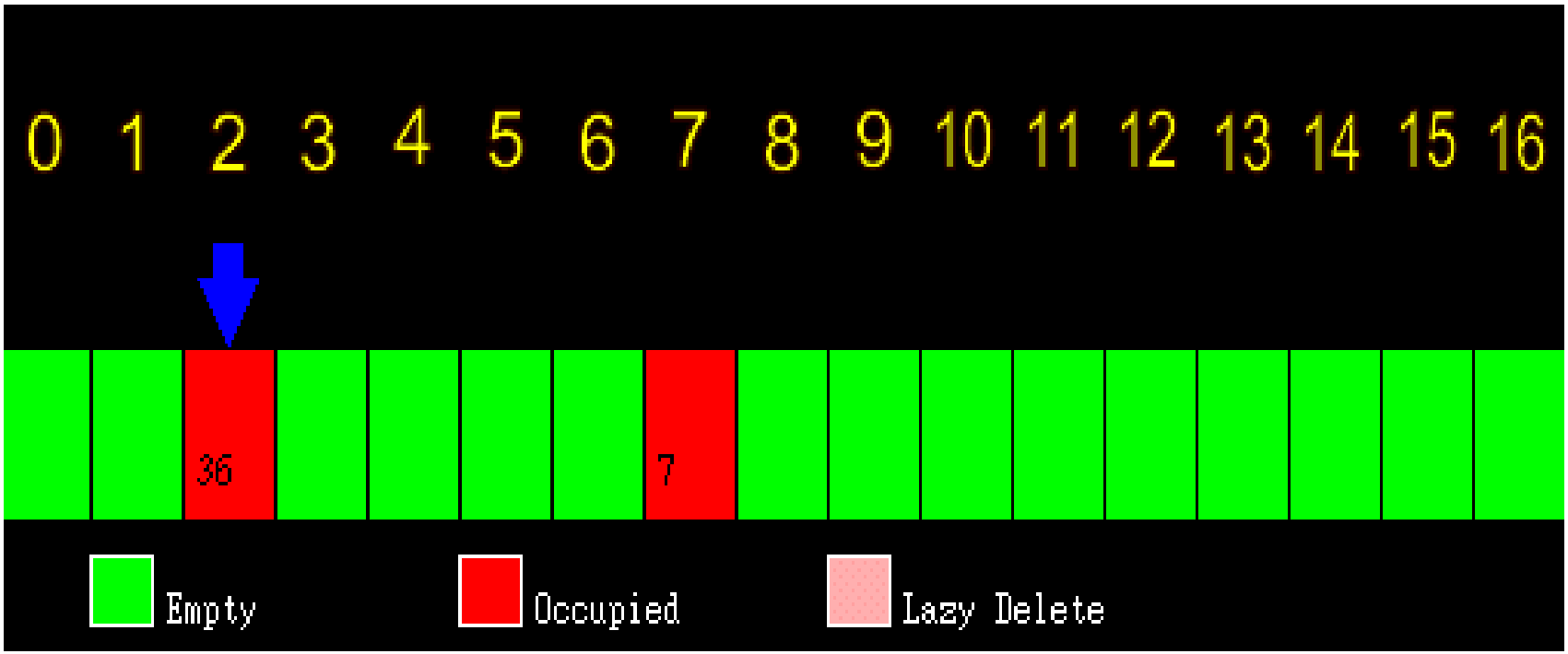
Hashing - Open addressing

- Linear probing example
  - Insert 7 at $h_0(7)$ (7 mod 17) = 7



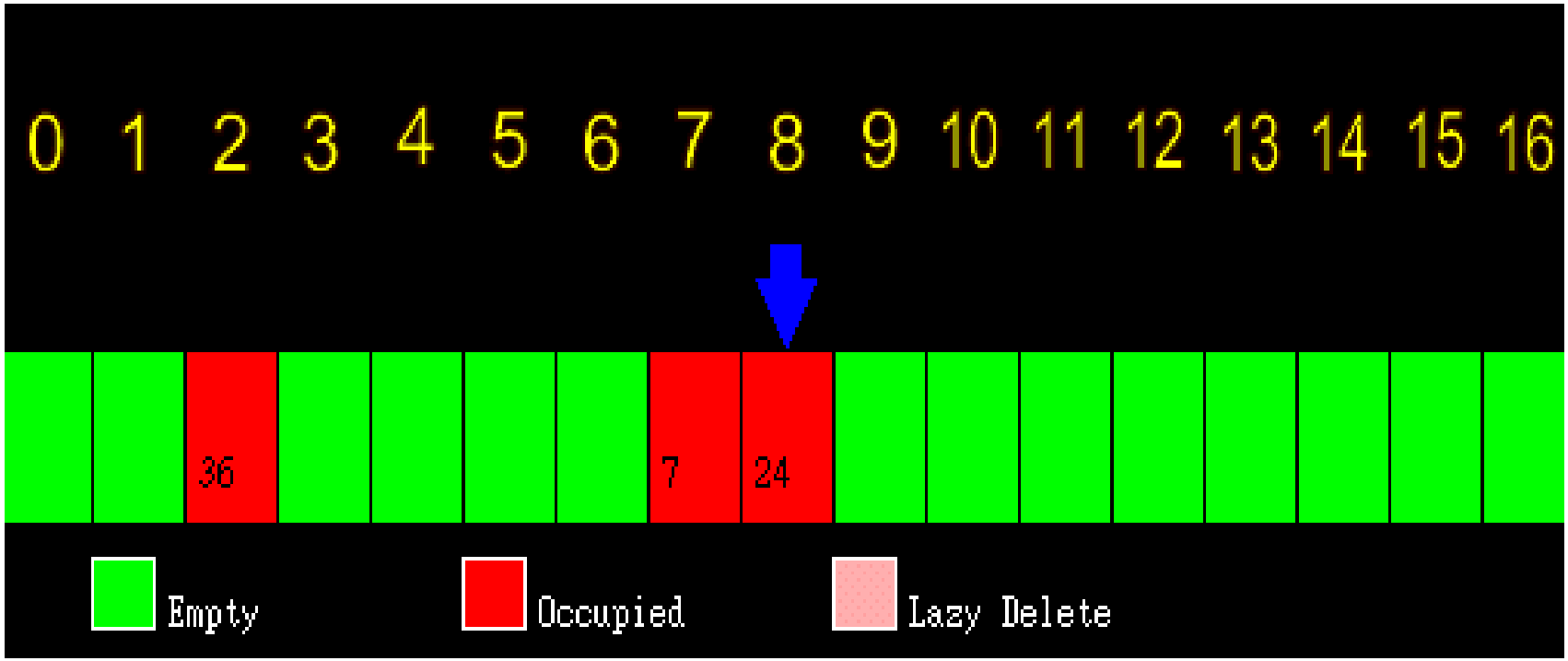
Hashing - Open addressing

- Linear probing example
  - Insert 36 at $h_0(36)$ (36 mod 17) = 2



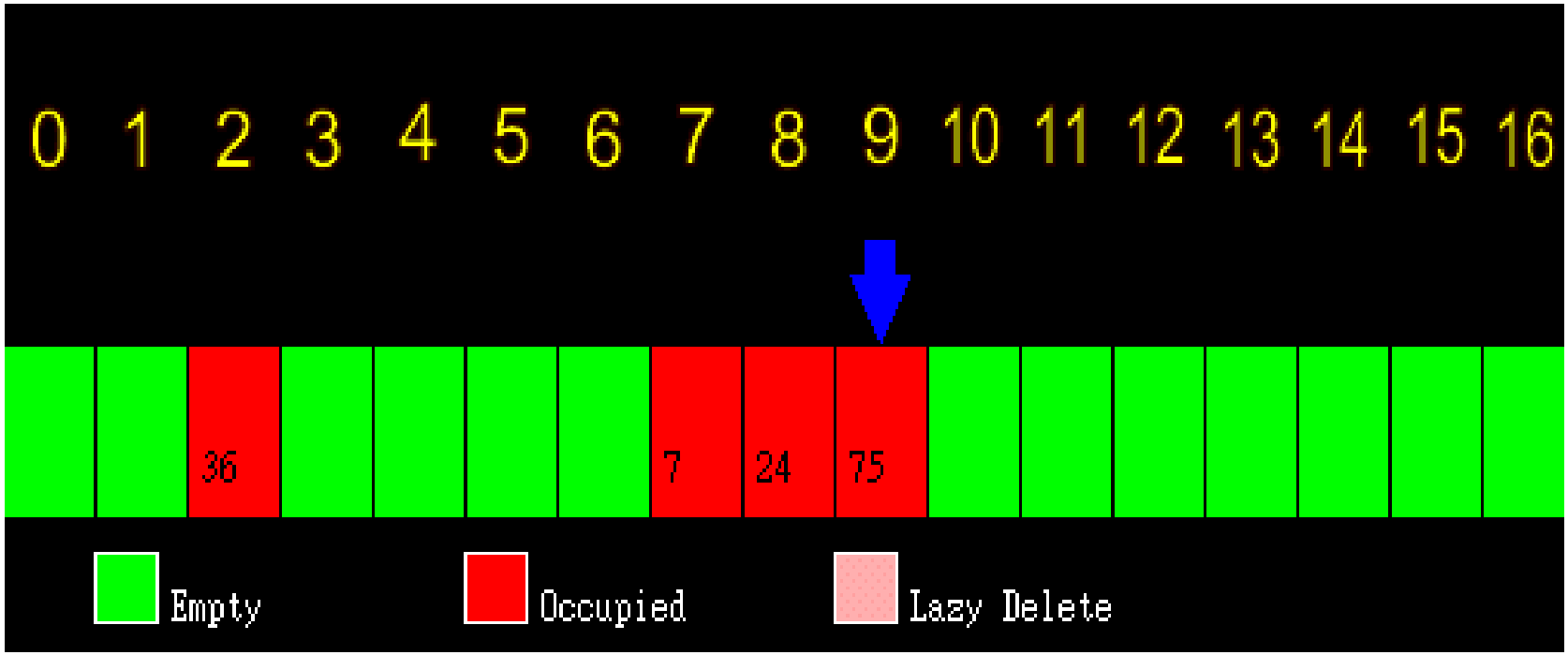
Hashing - Open addressing

- Linear probing example
  - Insert 24 at $h_1(24)$ (24 mod 17) = 7



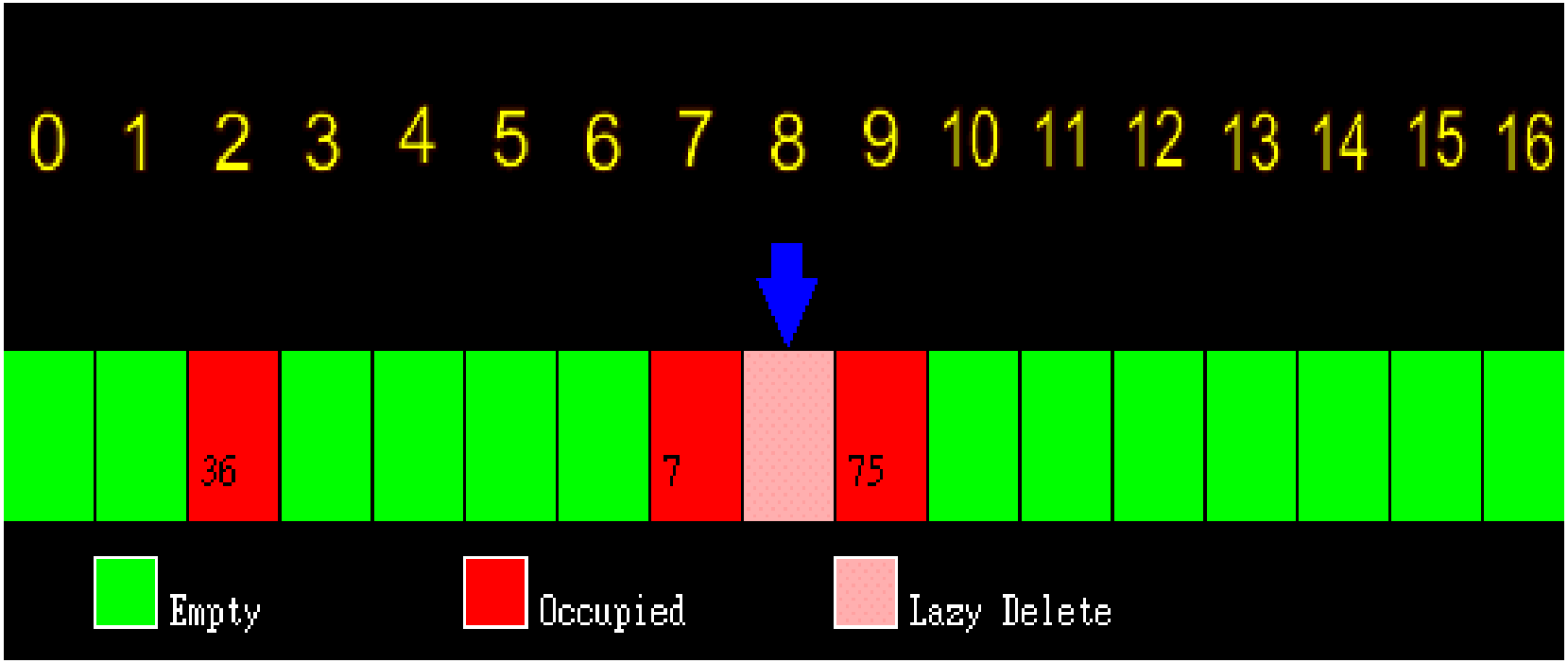
Hashing - Open addressing

- Linear probing example
  - Insert 75 at $h_2(75)$ (75 mod 17) = 7



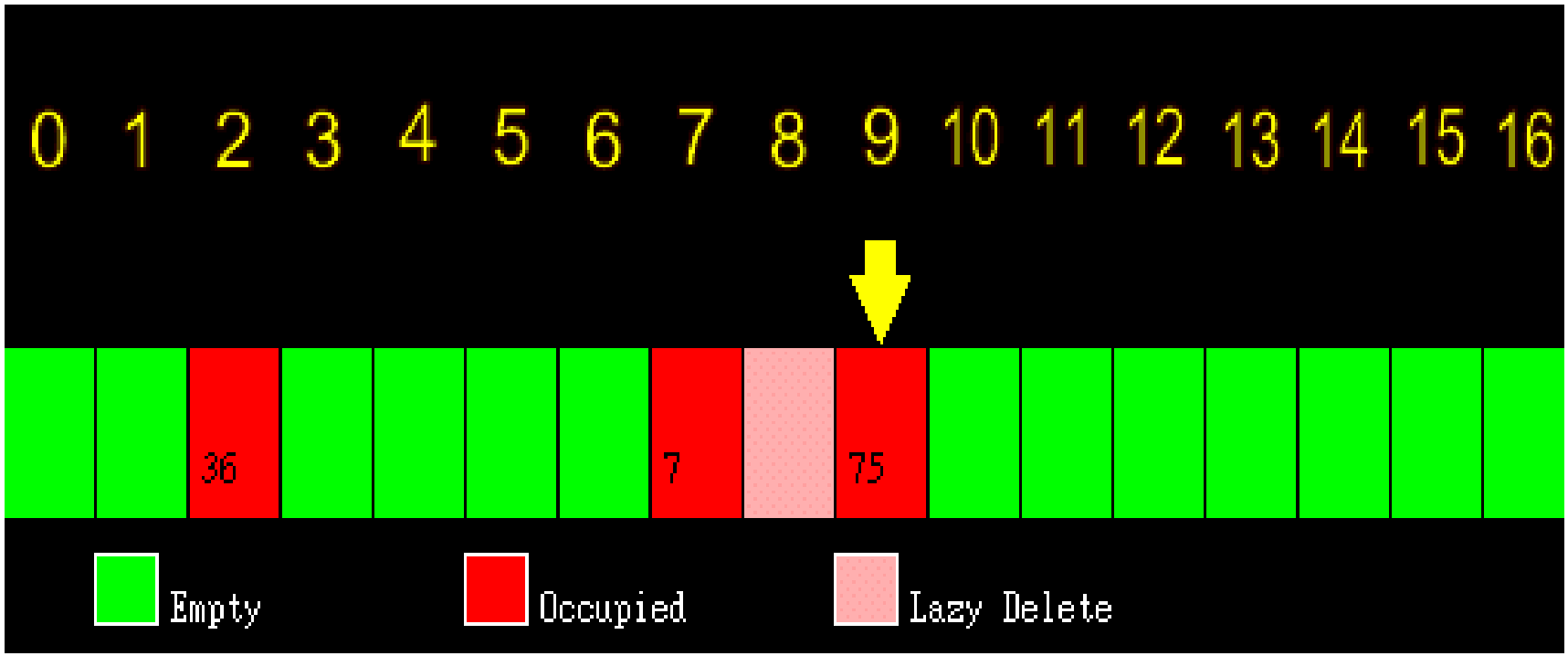
Hashing - Open addressing

- Linear probing example
  - Delete 24



Hashing - Open addressing

- Linear probing example
  - Find 75, found !



Hashing - Open addressing

- Quadratic probing
  - $F(i) = i^2$

$h_i(X) = (Hash(X) + i^2) \bmod TableSize$

$h_0(X) = (Hash(X) + 0^2) \bmod TableSize$,
$h_1(X) = (Hash(X) + 1^2) \bmod TableSize$,
$h_2(X) = (Hash(X) + 2^2) \bmod TableSize$, ...

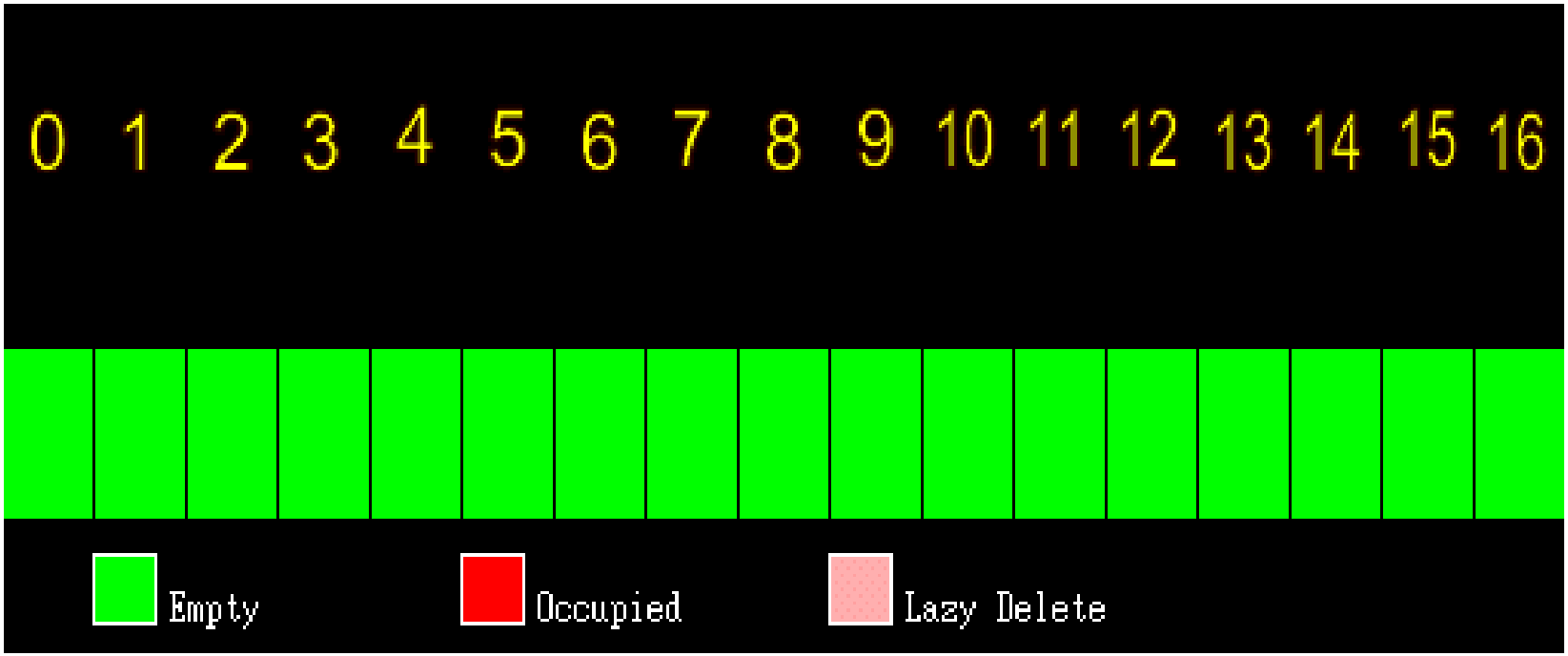
Hashing - Open addressing

- Quadratic probing

```
/* The h function */
int h(int i, int input)
{
   return (hash(input) + i * i) % HASHTABLESIZE;
}
```

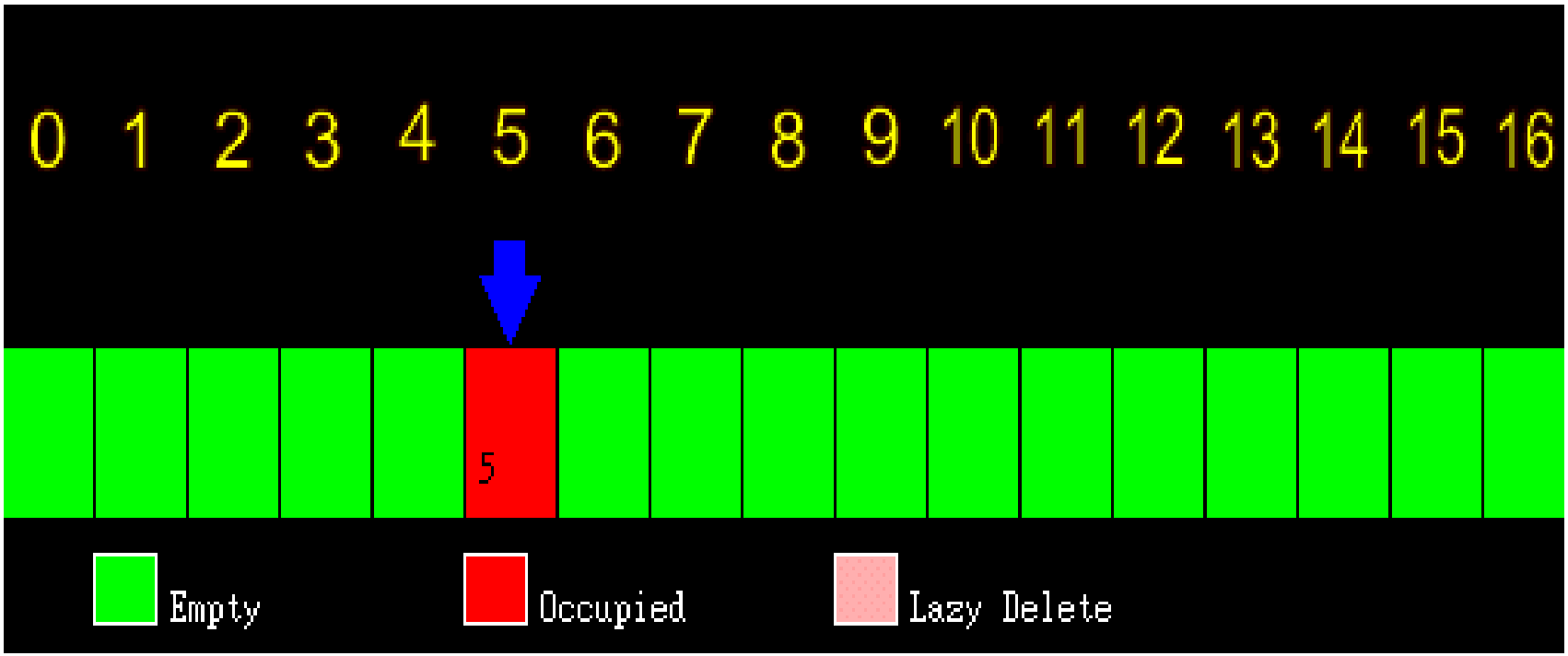
Hashing - Open addressing

- Quadratic probing example
  - Initial hash table



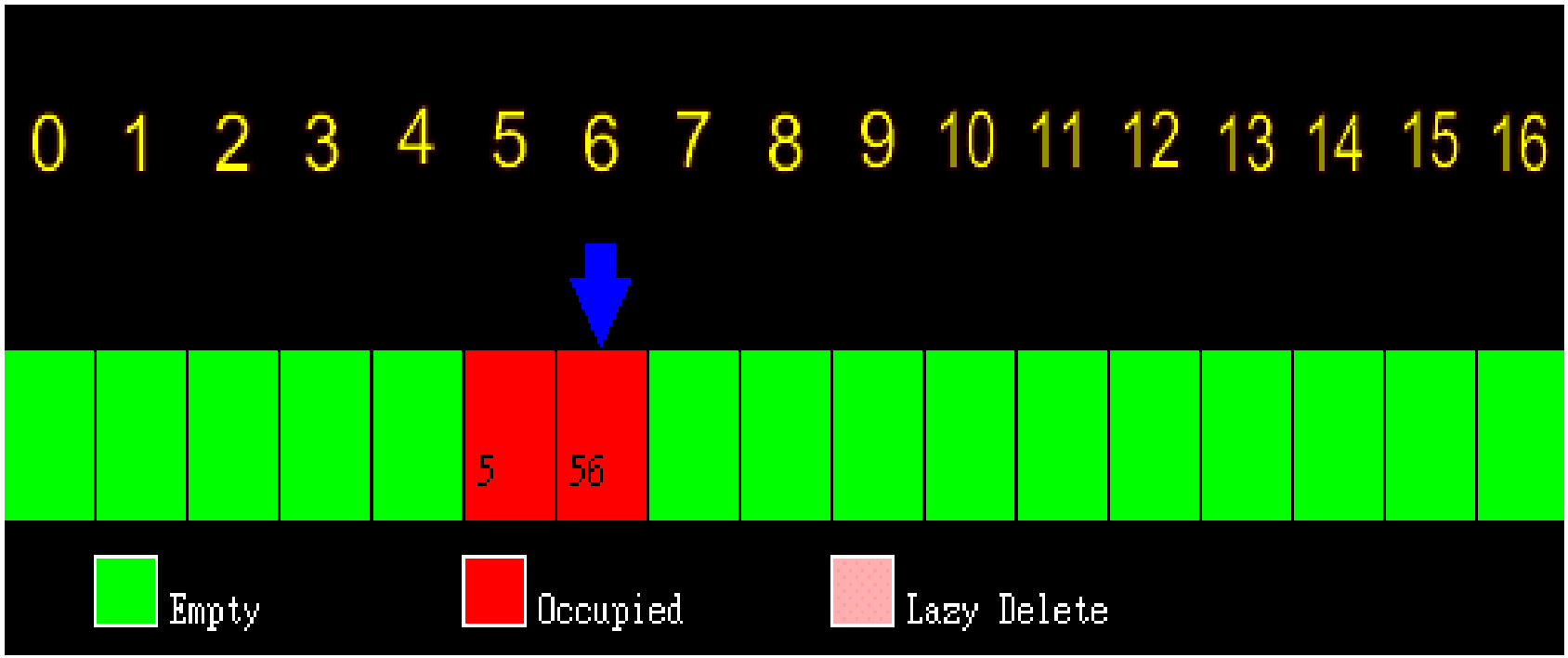
Hashing - Open addressing

- Quadratic probing example
  - Insert 5 at $h_0(5)$ (5 mod 17) = 5



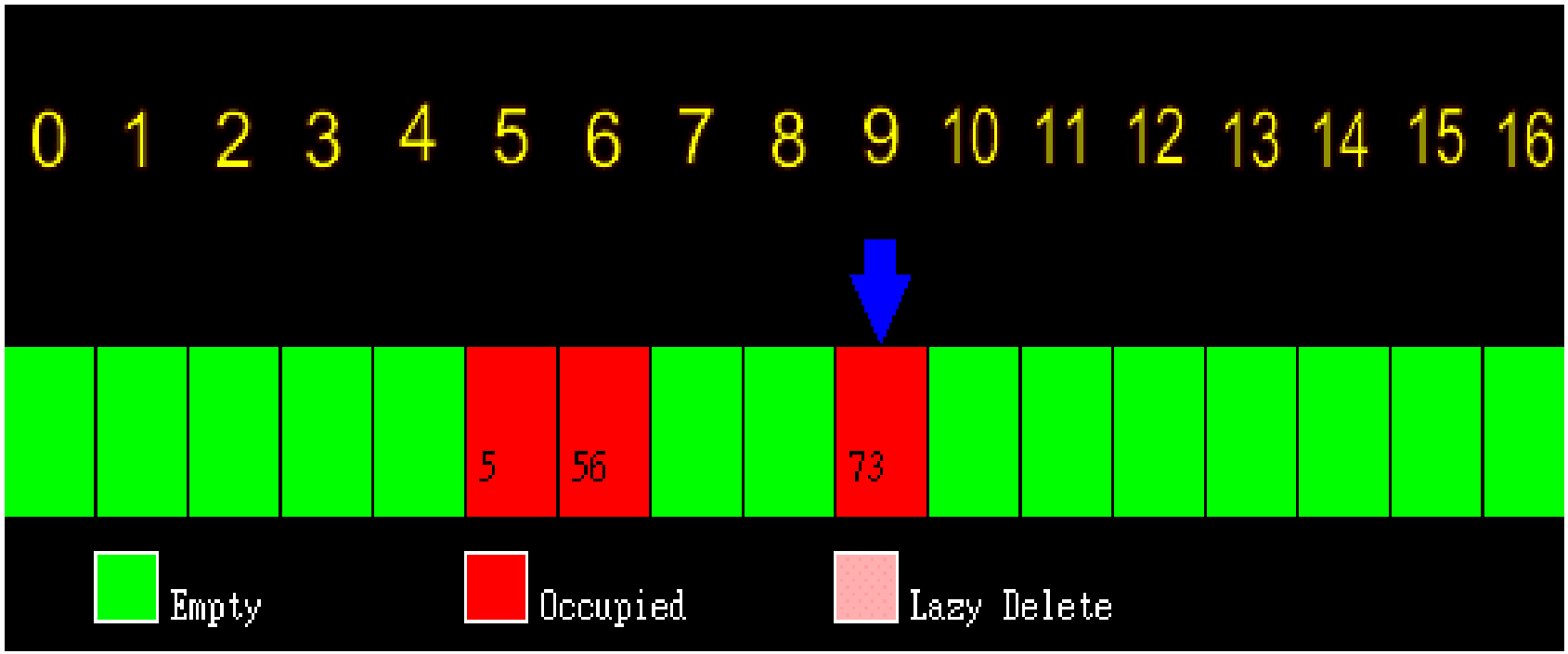
Hashing - Open addressing

- Quadratic probing example
  - Insert 56 at $h_1(56)$ (56 mod 17) = 5
    ((56 + 1*1) mod 17) = 6



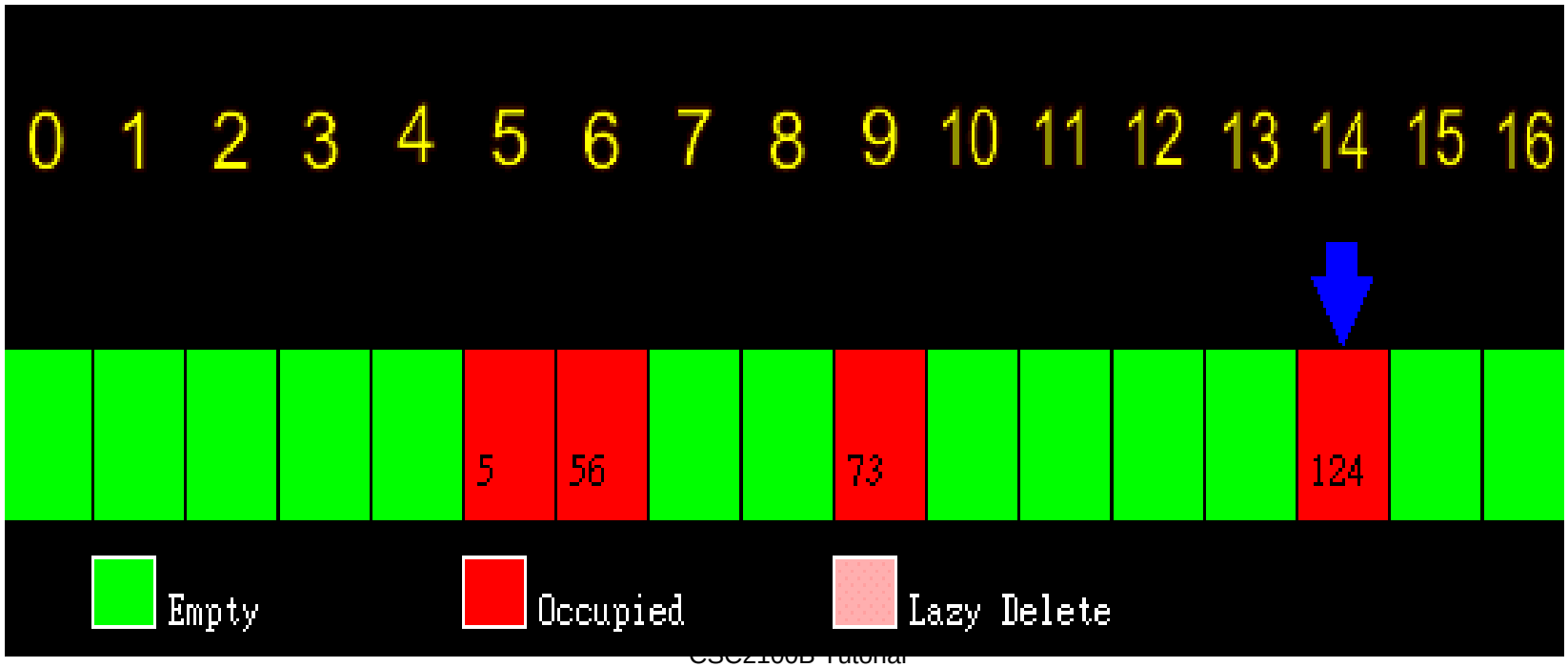
Hashing - Open addressing

- Quadratic probing example
  - Insert 73 at $h_2(56)$ (73 mod 17) = 5
    ((73 + 2*2) mod 17) = 9



Hashing - Open addressing

- Quadratic probing example
  - Insert 124 at $h_3(124)$ (124 mod 17) = 5
    ((124 + 3*3) mod 17) = 6



Hashing - Open addressing

- Random probing

  Randomize(X)

  $h_0(X) = \text{Hash}(X)$,

  $h_1(X) = (h_0(X) + \text{RandomGen}()) \bmod \text{TableSize}$,

  $h_2(X) = (h_1(X) + \text{RandomGen}()) \bmod \text{TableSize}$, ...

  - Use Randomize(X) to 'seed' the random number generator using X
  - Each call of RandomGen() will return the next random number in the random sequence for seed X

Hashing - Open addressing

- Implement random probing using random number generator in C
  - pseudo-random number generator: `rand()`
  - returns an integer between `0` and `RAND_MAX`
  - 'Seed' the randomizer
    - `srand(unsigned int);`
  - Use time as a 'seed'
    - `time(NULL);`

Hashing - Open addressing

- random number generation in C

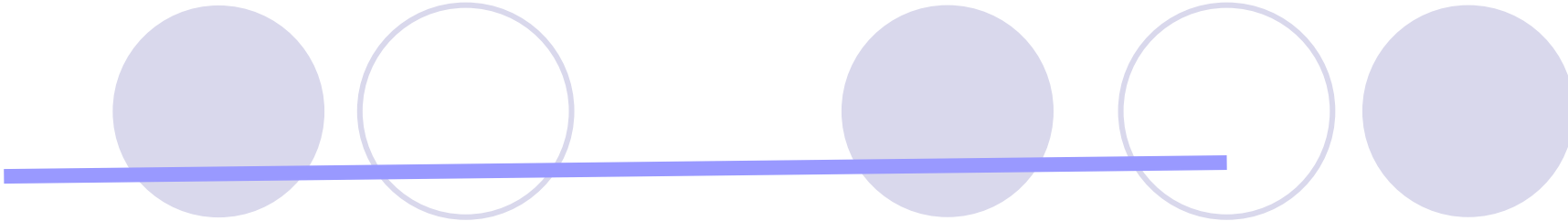
```
srand(time(NULL));

for (i = 0; i < 10; i++)
  printf("%d\n", rand());
```

Hashing - Open addressing

- Double hashing : $F(i) = i * Hash_2(X)$

$h_i(X) = (Hash(X) + i * Hash_2(X)) \bmod TableSize$

$h_0(X) = (Hash(X) + 0 * Hash_2(X)) \bmod TableSize,$
$h_1(X) = (Hash(X) + 1 * Hash_2(X)) \bmod TableSize,$
$h_2(X) = (Hash(X) + 2 * Hash_2(X)) \bmod TableSize, \ldots$



Thanks!