

Hashing in C

CSCI2100a Data Structures Tutorial

Jiani ZHANG

jnzhang@cse.cuhk.edu.hk

<https://www.cse.cuhk.edu.hk/irwin.king/teaching/csci2100/2016>

Contents

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

Contents

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

Hashing in C

- One of the biggest drawbacks to a language like C is that there are no keyed arrays.

Array	
Value	
New York	
Boston	
Mexico	
Kansas	
Detroit	
California	

Hash Table	
Key	Value
1	New York
2	Boston
3	Mexico
4	Kansas
5	Detroit
6	California

- Can only access *indexed Arrays*, e.g. ***city[5]***;
- *Cannot directly* access the values e.g. ***city["California"]***;

Hashing - hash function

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

```
/* Hash function for ints */  
int hashfunc(int integer_key)  
{  
    return integer_key % HASHTABLESIZE;  
}
```

- However, collisions cannot be avoided.

Hashing - hash function Cont.

```
/* hash functions for strings from Sample Code in
Weiss's*/
typedef unsigned int Index;

Index Hash1( const char *Key, int TableSize
)
{
    unsigned int HashVal = 0;
    while( *Key != '\0' )
        HashVal += *Key++;
    return HashVal % TableSize;
}
```

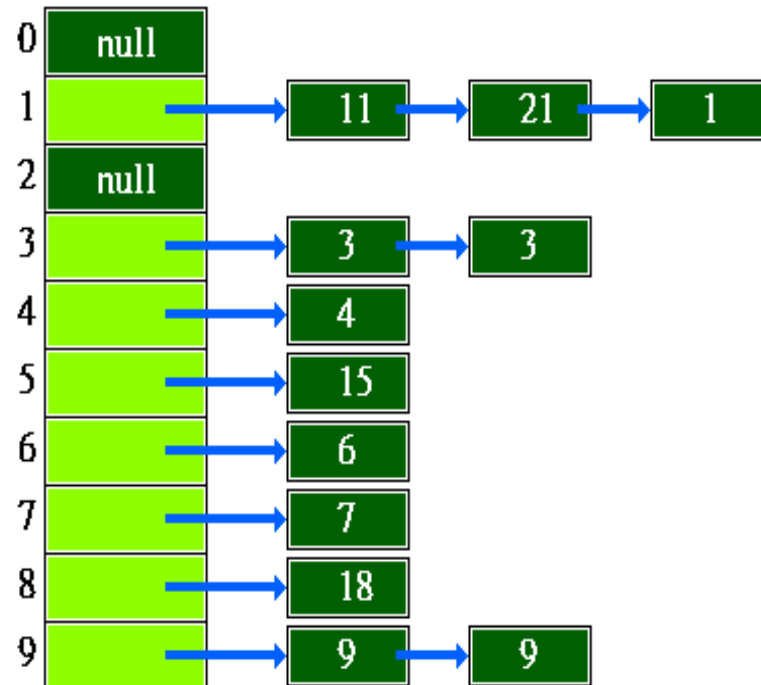
- However, collisions cannot be avoided.

Contents

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

Hashing - separate chaining

- If two keys map to **same value**, the elements are chained together by creating a linked list of elements



Hashing - example

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

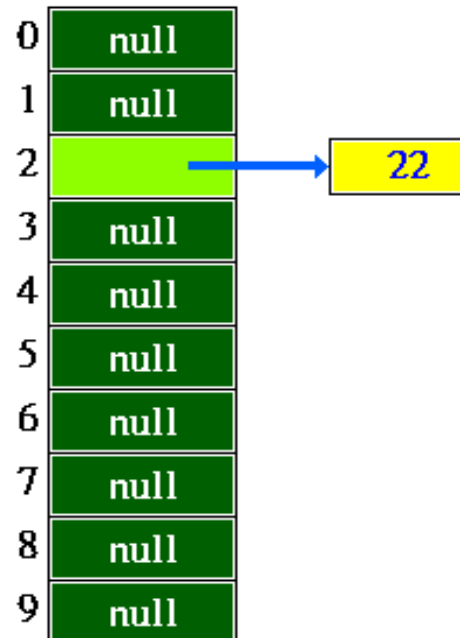
Initial hash table

0	null
1	null
2	null
3	null
4	null
5	null
6	null
7	null
8	null
9	null

Hashing - example

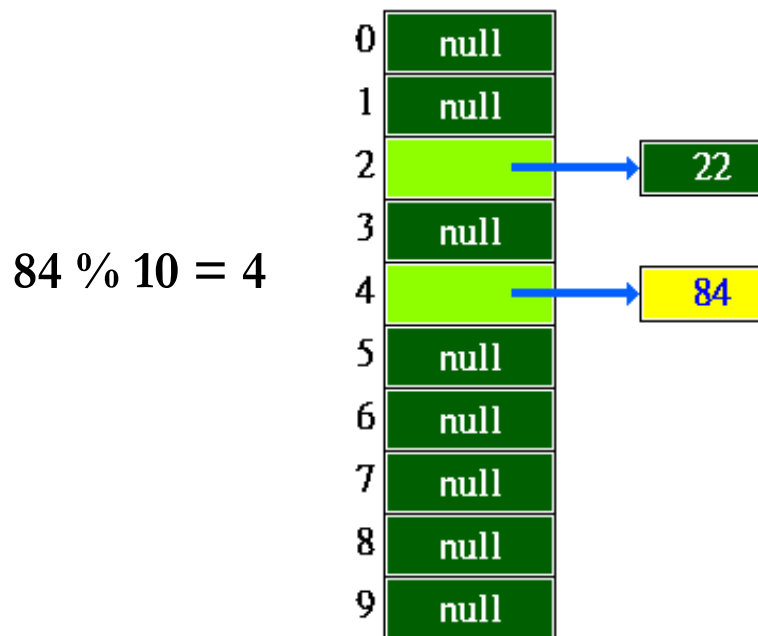
- Insert the following four keys 22 84 35 62 into hash table of size 10 using separate chaining.
- The hash function is $\text{key} \% 10$

$$22 \% 10 = 2$$



Hashing - example

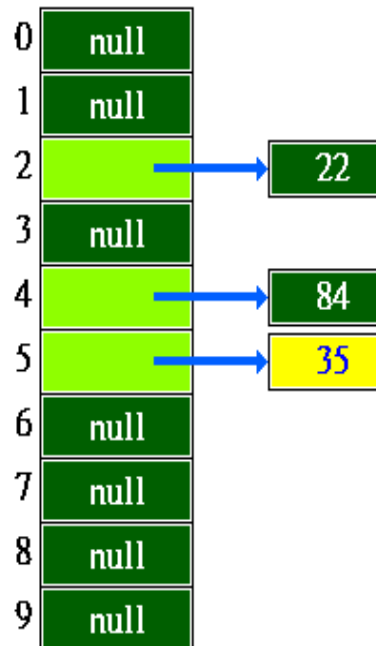
- Insert the following four keys 22 84 35 62 into hash table of size 10 using separate chaining.
- The hash function is $\text{key} \% 10$



Hashing - example

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

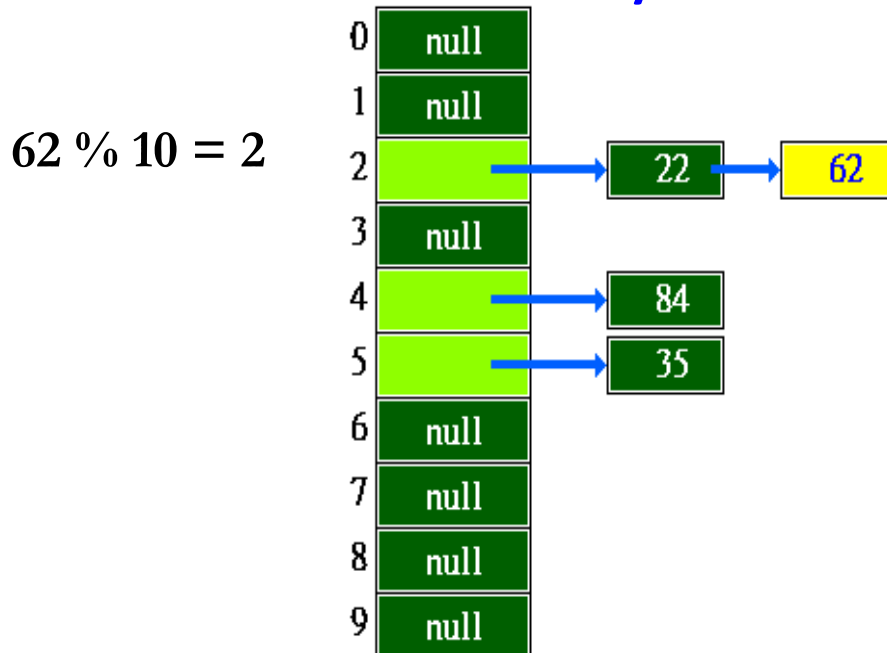
$$35 \% 10 = 5$$



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 $\text{key} \% 10$



Contents

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

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];
    char 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.

$$h_0(X), h_1(X), h_2(X), \dots, h_k(X)$$
$$= (\text{Hash}(X) + F(k)) \bmod \text{TableSize}$$

- Linear probing $F(k) = k$

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

- Double hashing $F(k) = k * \text{Hash}_2(X)$

Hashing - Open addressing

```
void open_addressing_insert(ht * ht, int item)
{
    hash_value = hash(item);
    i = hash_value;
    k = 1;
    while (ht->state[i] != 0) {
        if (ht->key[i] == item) {
            fprintf(stderr, "Duplicate entry\n");
            exit(1);
        }
        i = h(k++, item);
        if (i == hash_value) {
            fprintf(stderr, "The table is full\n");
            exit(1);
        }
    }
    ht->key[i] = item;
}

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

Contents

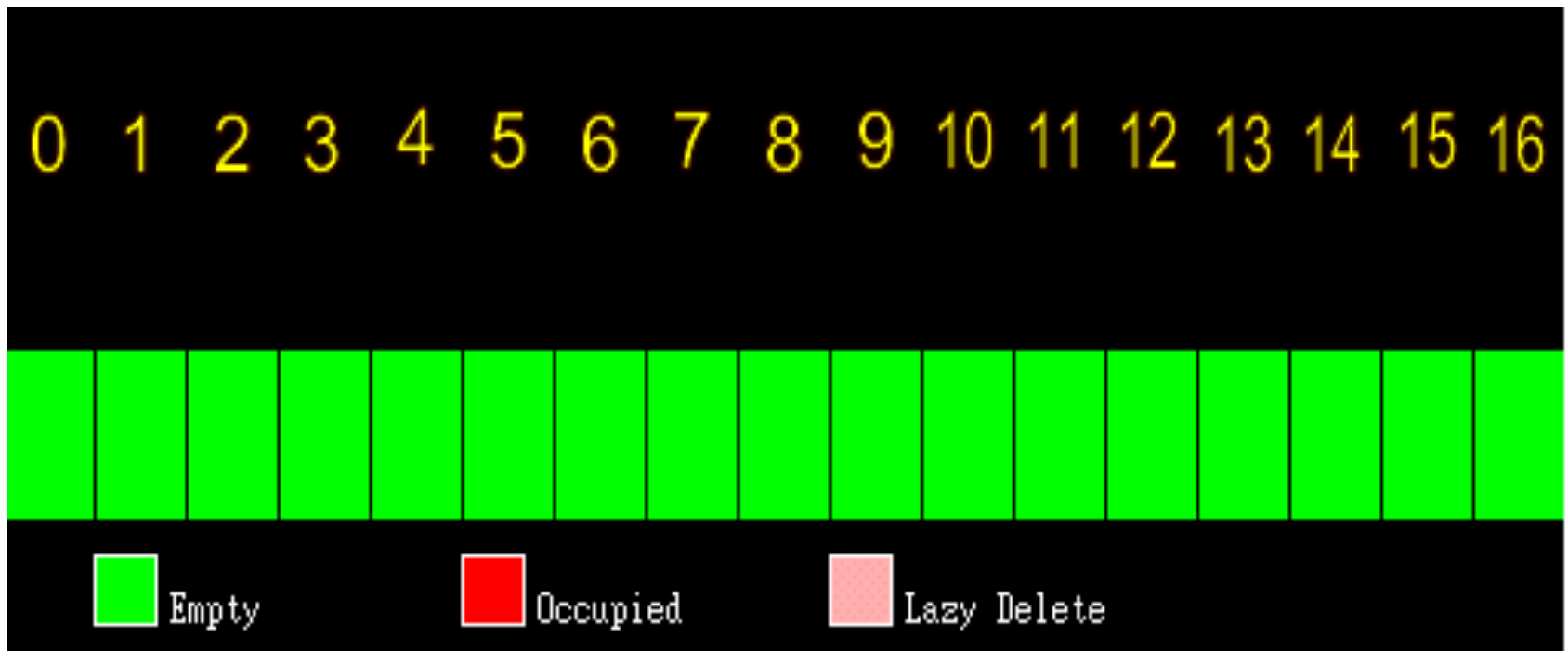
- Hash function
- Collision resolutions
 - Separate Chaining (Open hashing)
 - **Open addressing (Closed Hashing)**
 - **Linear probing**
 - Quadratic probing
 - Random probing
 - Double hashing

Linear probing

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

Hashing - Open addressing

- Linear probing example
 - Initial hash table

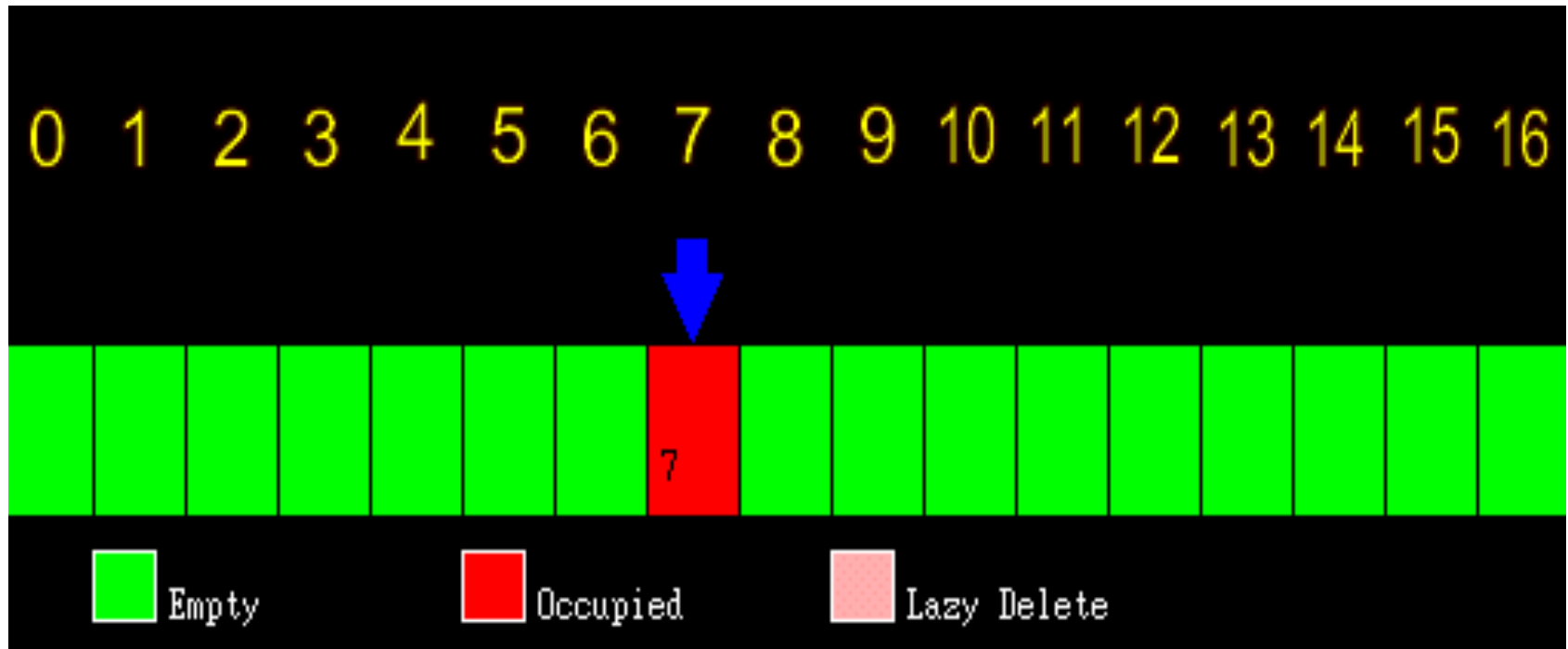


Hashing - Open addressing

- Linear probing example

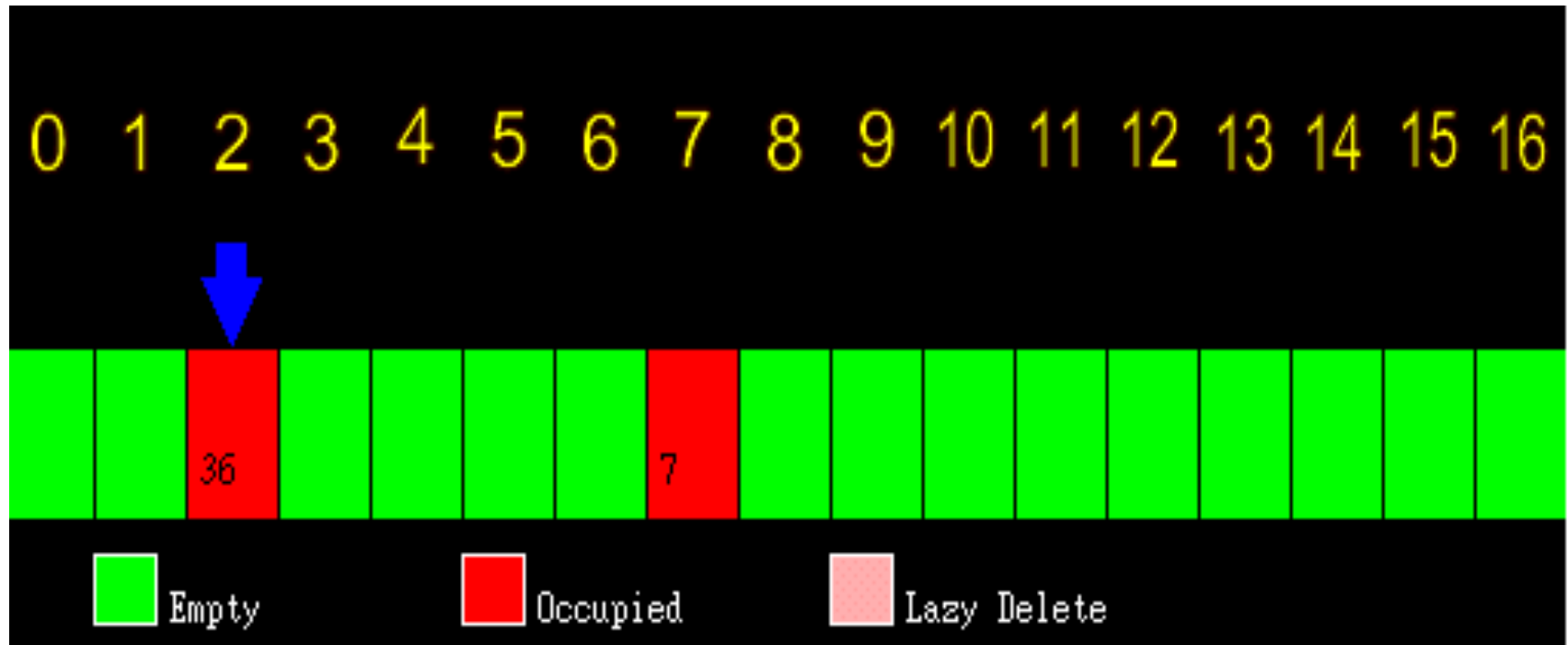
- Insert 7 at $h_0(7)$

$$(7 \bmod 17) = 7$$



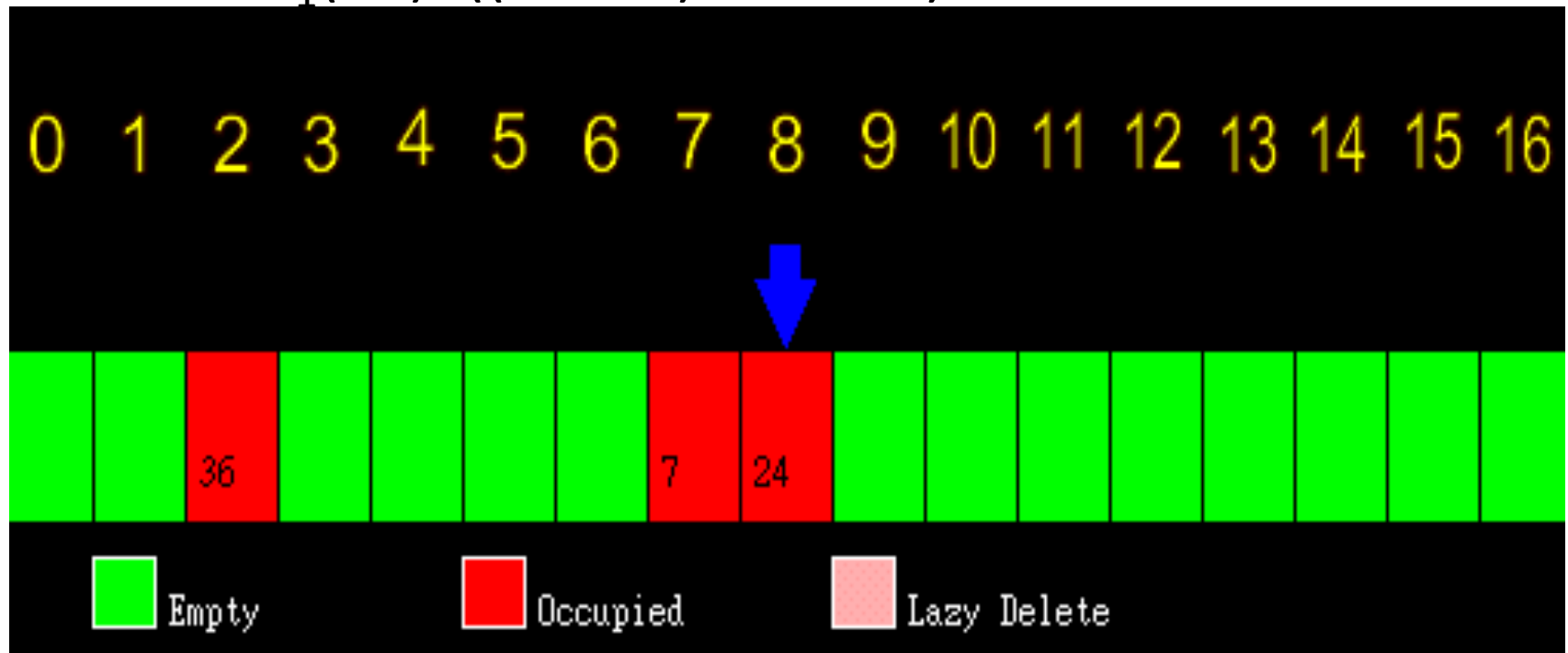
Hashing - Open addressing

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



Hashing - Open addressing

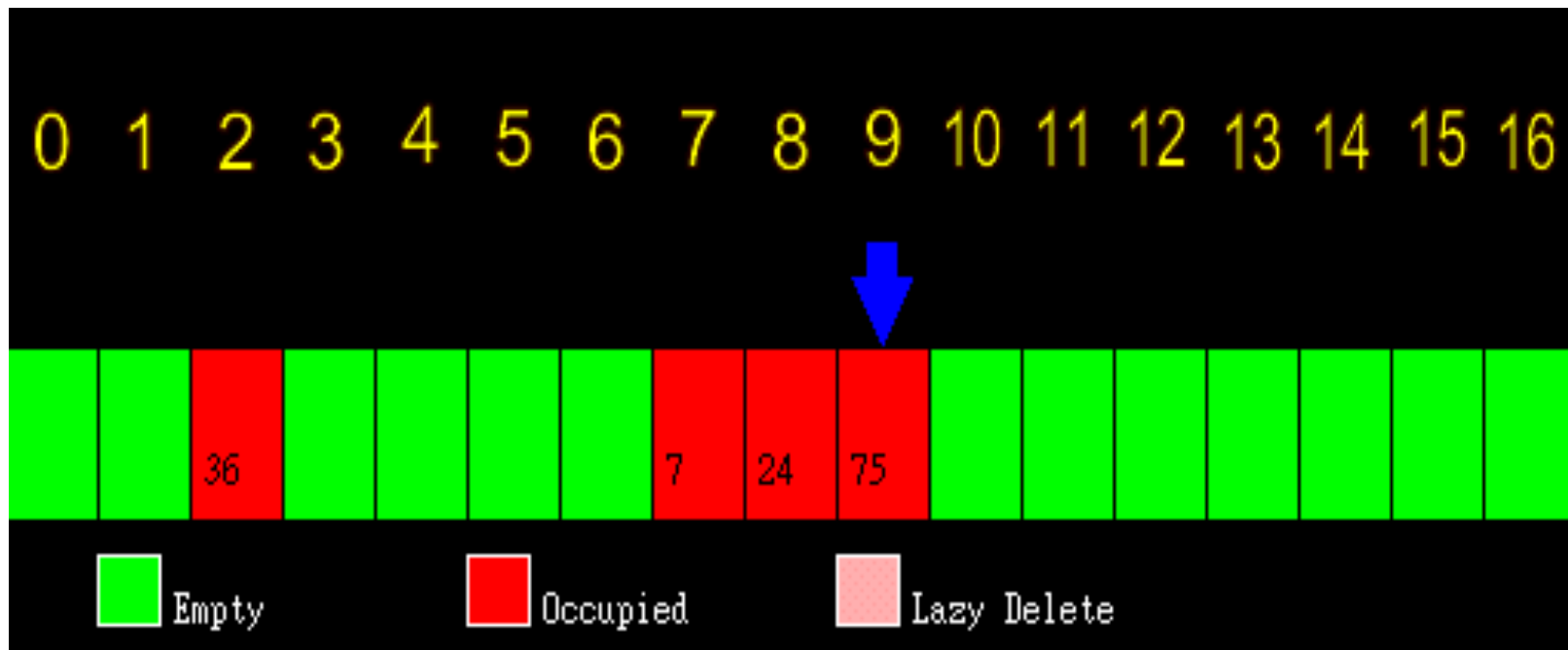
- Linear probing example
 - Insert 24 at $h_0(24) = (24 \bmod 17) = 7$, so we call $h_1(24) = ((24 + 1) \bmod 17) = 8$



Hashing - Open addressing

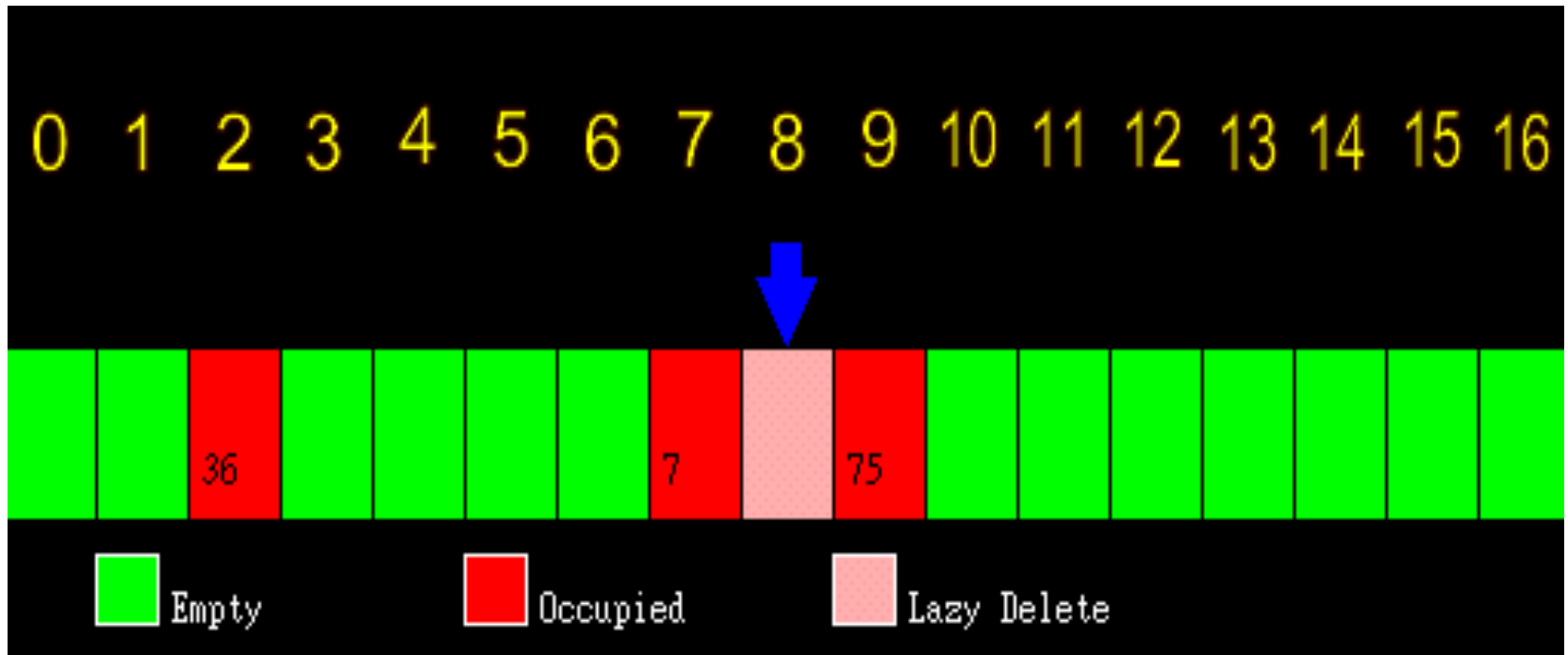
- Linear probing example

– Insert 75 at $h_0(75) = (75 \bmod 17) = 7$, $h_1(75) = ((75+1) \bmod 17) = 8$, $h_2(75) = ((75+2) \bmod 17) = 9$,



Hashing - Open addressing

- Linear probing example
 - **Delete 24** -> lazy deletion technique

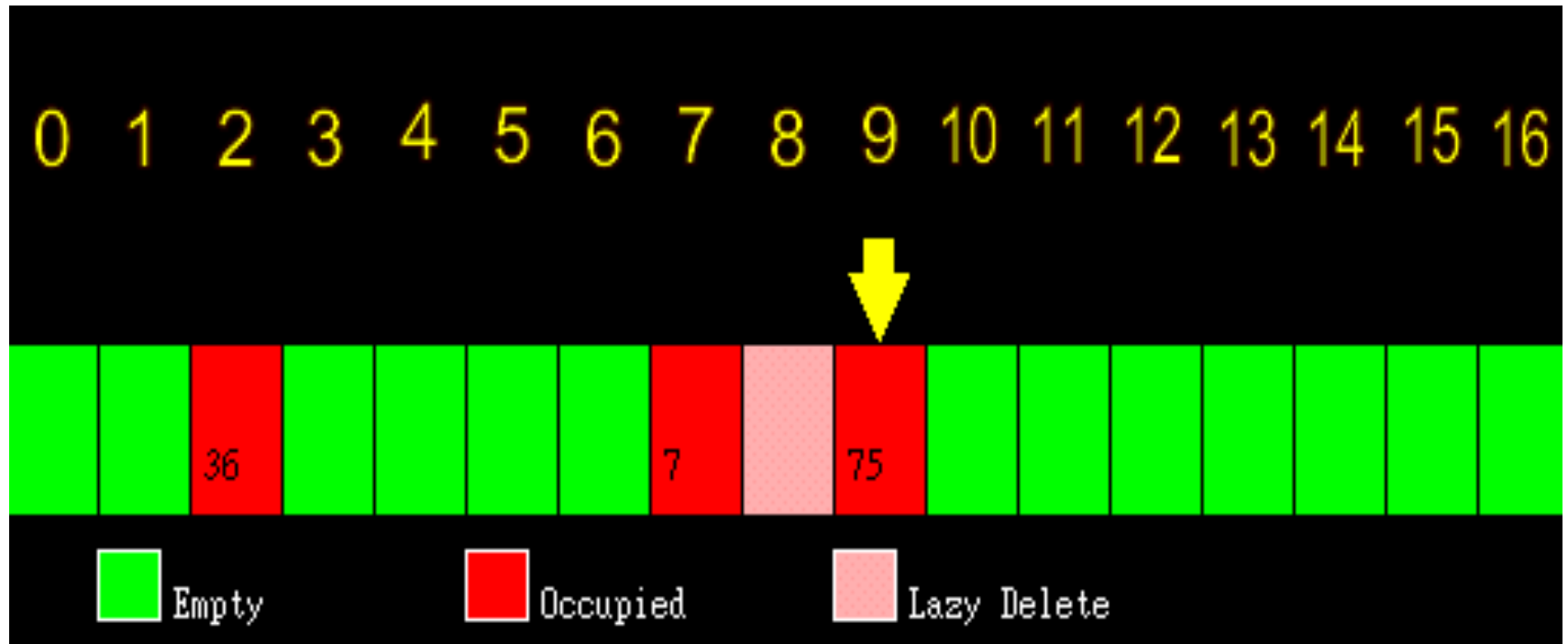


Lazy Deletion

- We need to be careful about removing elements from the table as it may leave holes in the table.
- Lazy Deletion:
 - not to delete the element, but place a marker in the place to indicate that an element that was there is now removed.
 - So when we are looking for things, we jump over the “dead bodies” until we find the element or we run into a null cell.
- Drawback
 - Space cost

Hashing - Open addressing

- Linear probing example
 - Find 75 $h_0(75) = (75 \bmod 17) = 7$ (occupied), 8 (lazy delete), 9 (Get it!)



Hashing - Open addressing

- Linear probing

```
/* The h function */
```

```
int h(int k, int input)
```

```
{
```

```
    return (hash(input) + k) % HASHTABLESIZE;
```

```
}
```

```
while (ht->state[i] != 0) {  
    if (ht->key[i] == item) {  
        fprintf(stderr, "Duplicate entry\n");  
        exit(1);  
    }  
    i = h(k++, item);  
    //call the function  
    if (i == hash_value) {  
        fprintf(stderr, "The table is full\n");  
        exit(1);  
    }  
}
```

Contents

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

Hashing - Open addressing

- Quadratic probing

- $F(k) = k^2$

$$h_k(X) = (\text{Hash}(X) + k^2) \bmod \textit{TableSize}$$

$$h_0(X) = (\text{Hash}(X) + 0^2) \bmod \textit{TableSize},$$

$$h_1(X) = (\text{Hash}(X) + 1^2) \bmod \textit{TableSize},$$

$$h_2(X) = (\text{Hash}(X) + 2^2) \bmod \textit{TableSize}, \dots$$

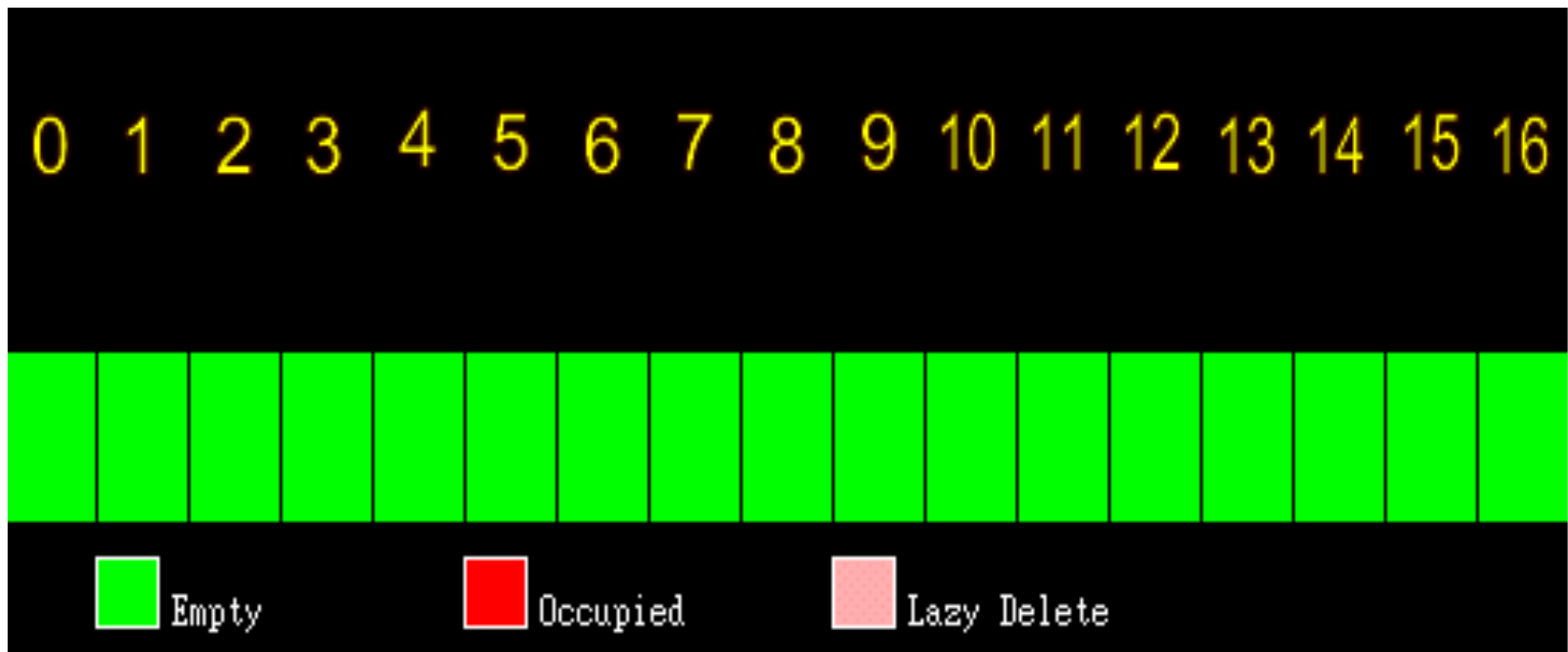
Hashing - Open addressing

- Quadratic probing

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

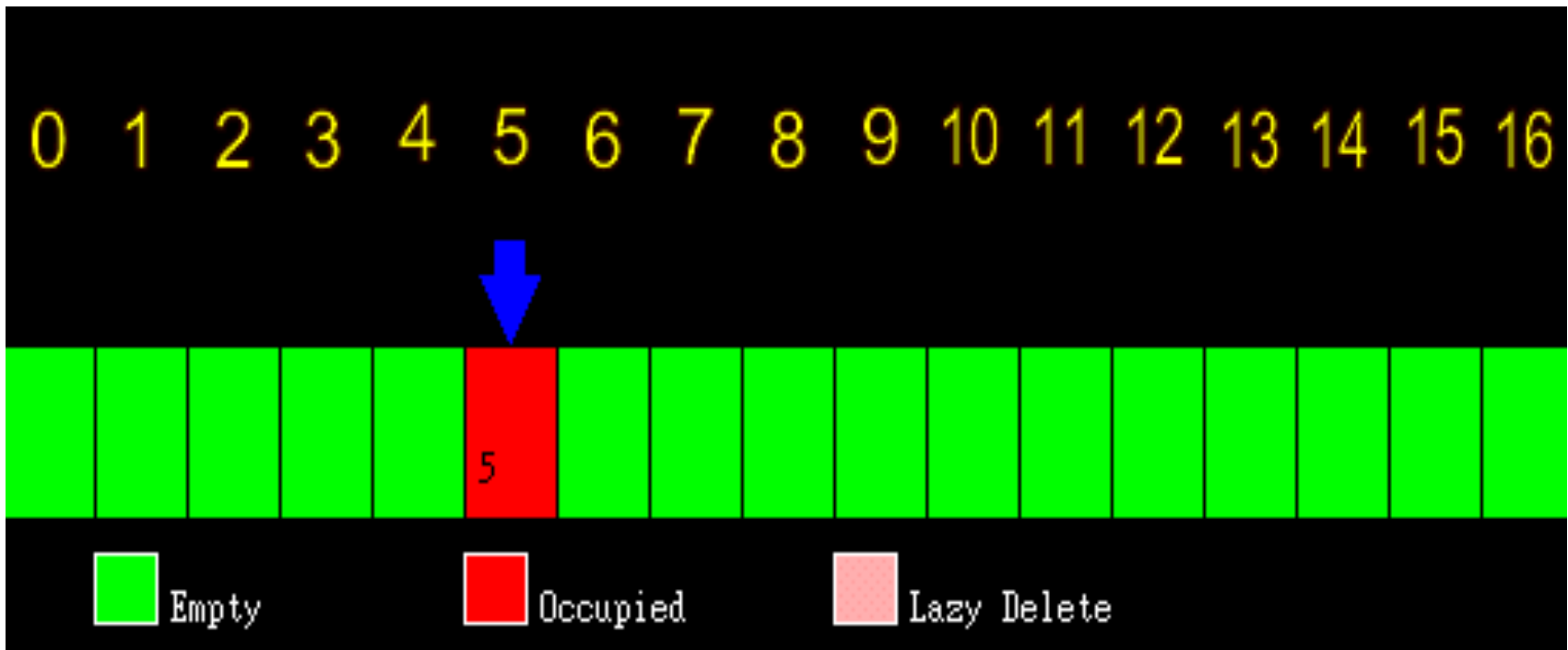

Hashing - Open addressing

- Quadratic probing example
 - Initial hash table



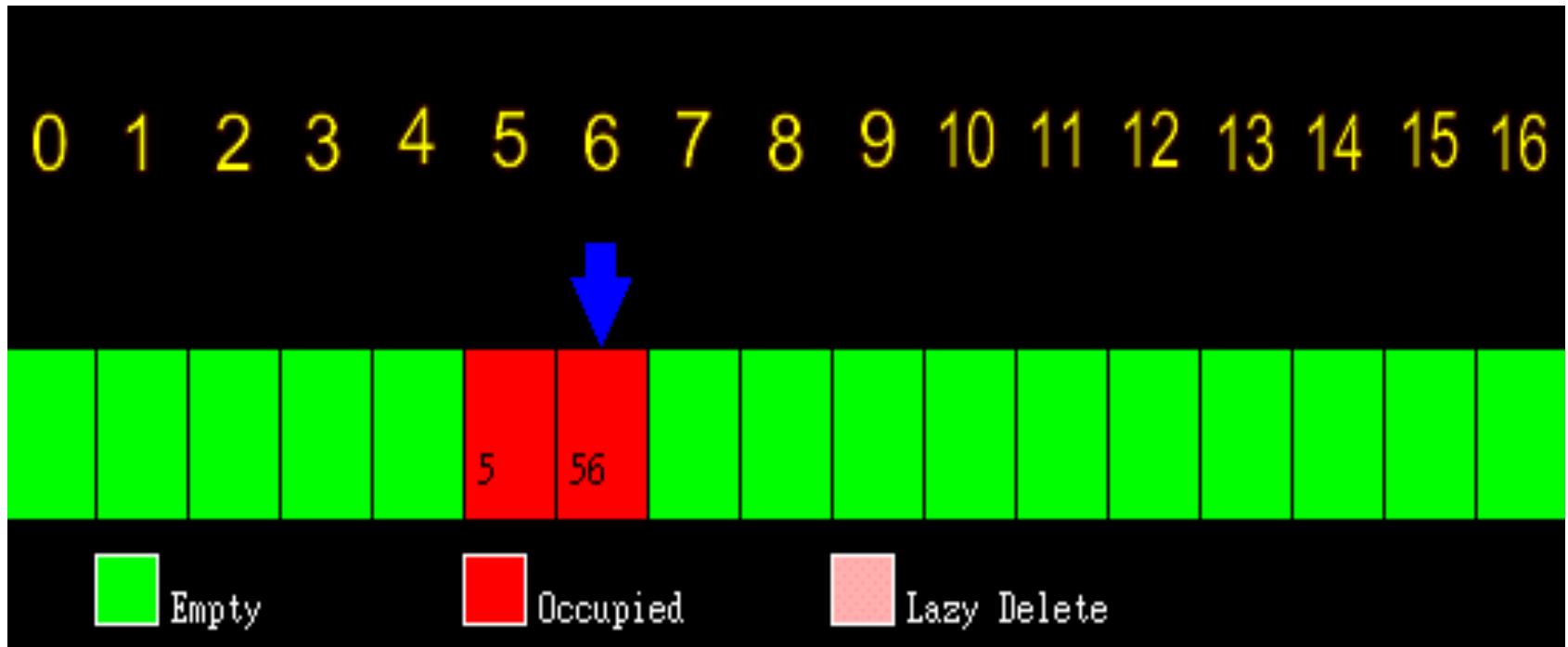
Hashing - Open addressing

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



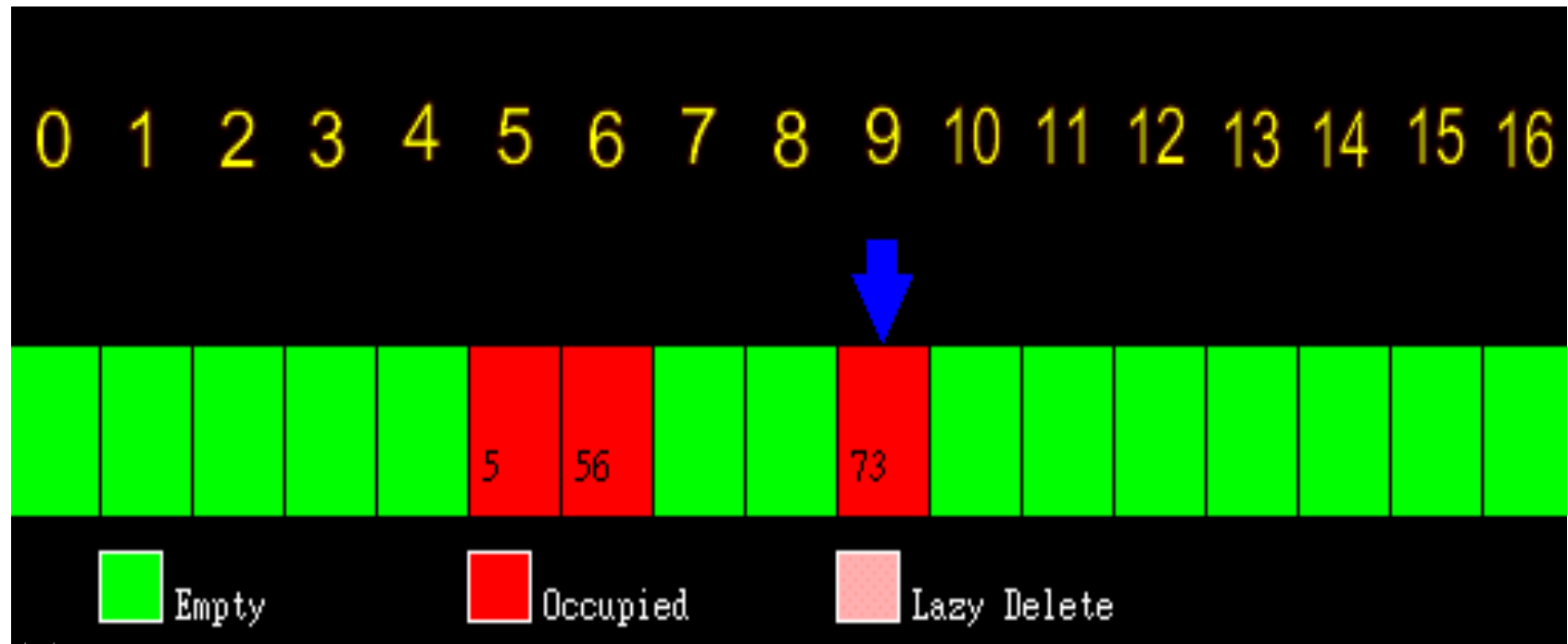
Hashing - Open addressing

- Quadratic probing example
 - Insert 56 at $h_0(56) = (56 \bmod 17) = 5$
 $h_1(56) = ((56 + 1 * 1) \bmod 17) = 6$



Hashing - Open addressing

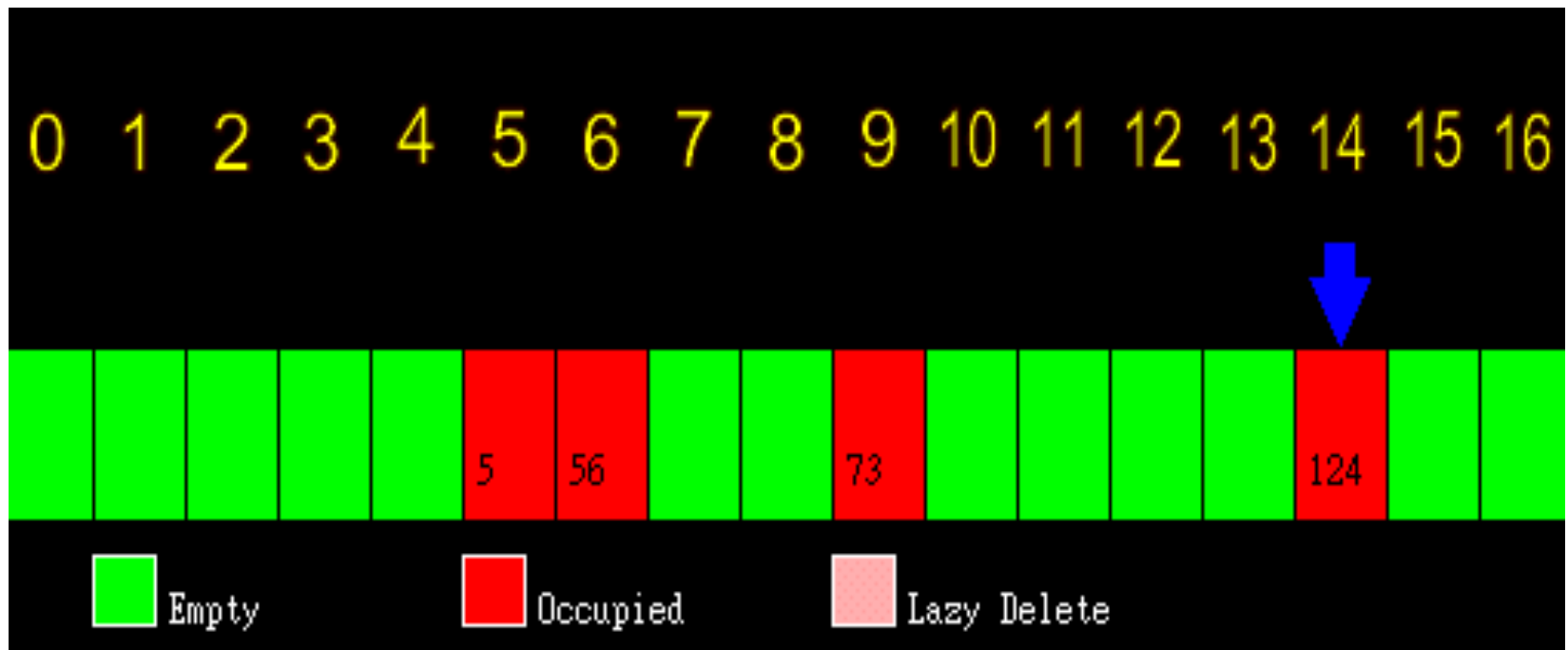
- Quadratic probing example
 - Insert 73 at $h_0(56) = (73 \bmod 17) = 5$, $h_1(56) = ((73 + 1*1) \bmod 17) = 6$, $h_2(56) = ((73 + 2*2) \bmod 17) = 9$



Hashing - Open addressing

- Quadratic probing example

- Insert 124 at $h_0(124)=(124 \bmod 17) = 5$, $h_1(124)=(124+1*1 \bmod 17) = 6$,
 $h_2(124)=(124+2*2 \bmod 17) = 9$, $h_3(124)=((124 + 3*3) \bmod 17) = 14$



Contents

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

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(time_t *);`
 - `time(NULL);`



Random number generation in C

```
#include <stdlib.h>
#include <time.h>
int main(){
    int i;
    srand(time(NULL));
    for (i = 0; i < 10; i++){
        printf("%d\n", rand());
    }
    return 0;
}
```

```
1518815302
1738152472
908546763
1336715571
1352170530
1258145156
1521648530
2111575234
2077691163
1671276321
```

```
#include <stdlib.h>
#include <time.h>
int main(){
    int i;
    for (i = 0; i < 10; i++){
        srand(time(NULL));
        printf("%d\n", rand());
    }
    return 0;
}
```

```
1518395127
1518395127
1518395127
1518395127
1518395127
1518395127
1518395127
1518395127
1518395127
1518395127
```

Contents

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

Hashing - Open addressing

- Double hashing : $F(k) = k * \text{Hash}_2(X)$

$$h_k(X) = (\text{Hash}(X) + k * \text{Hash}_2(X)) \bmod \textit{TableSize}$$

$$h_0(X) = (\text{Hash}(X) + 0 * \text{Hash}_2(X)) \bmod \textit{TableSize},$$

$$h_1(X) = (\text{Hash}(X) + 1 * \text{Hash}_2(X)) \bmod \textit{TableSize},$$

$$h_2(X) = (\text{Hash}(X) + 2 * \text{Hash}_2(X)) \bmod \textit{TableSize}, \dots$$

Review

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

Thank you !