COSC 2436: Hashing Practice

J. Eoin Donovan - 2023

Direct Hashing

- **Overwrite data when collision happens**

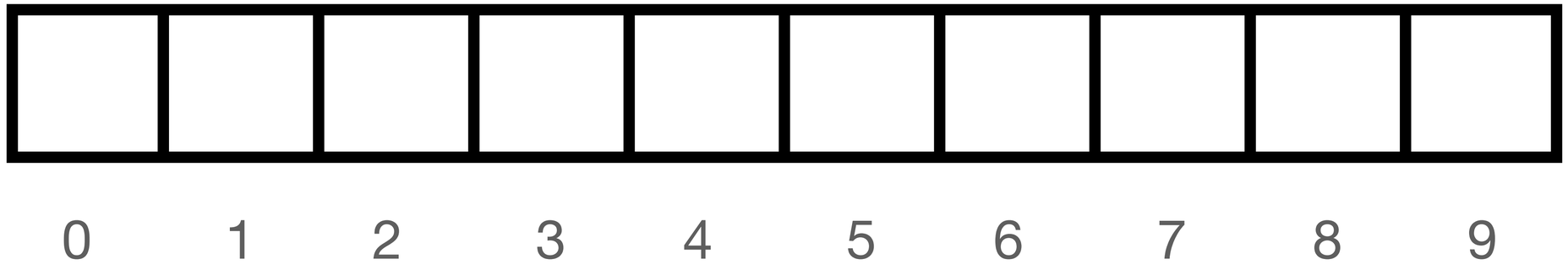
- **Get index by doing: x % tableSize**

Direct Hashing

```
void directHashing(int table[], int size, int x){
   int index = x % size;
   table[index] = x;
}
```

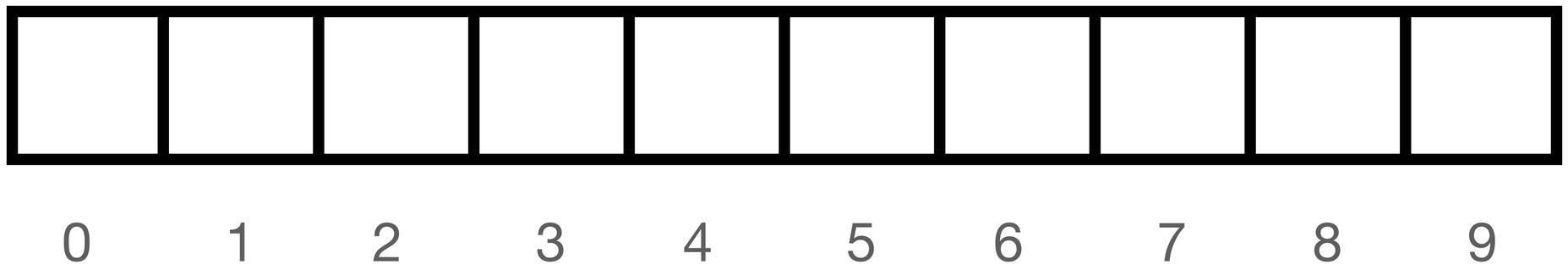
Direct Hashing

Let’s start with a table of size 10



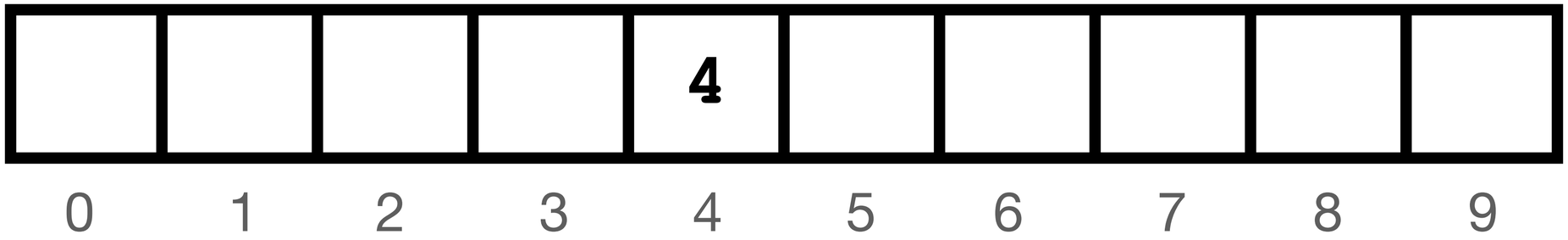
Direct Hashing

Insert 4 into the table



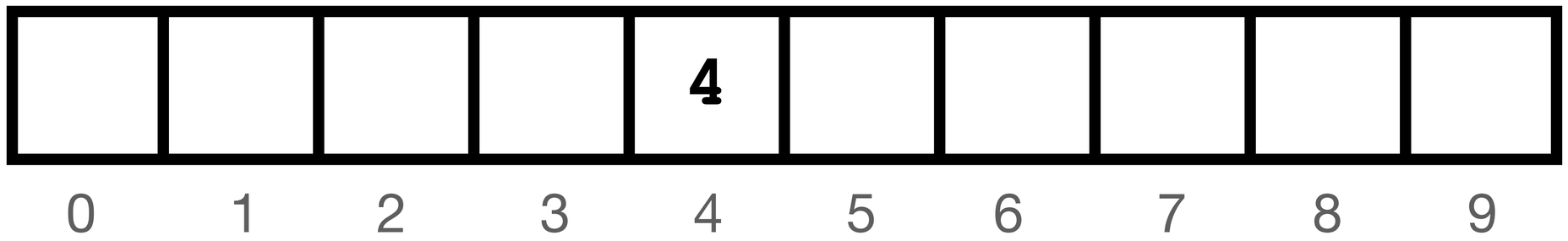
Direct Hashing

index = 4 % 10 = 4



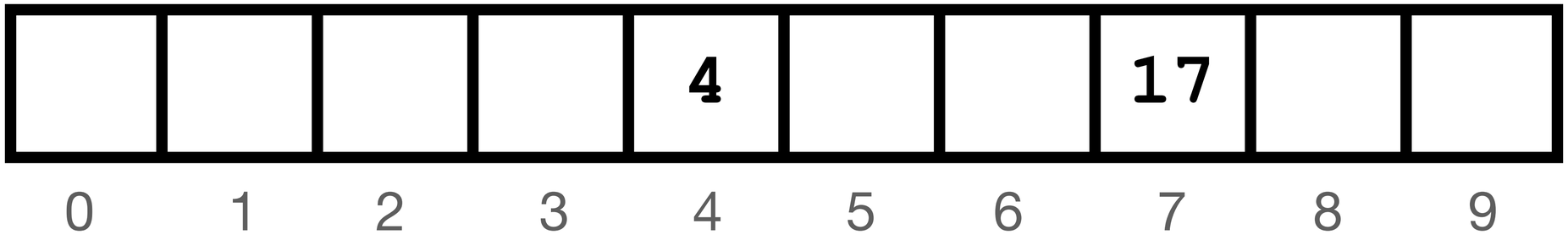
Direct Hashing

Insert 17 into table



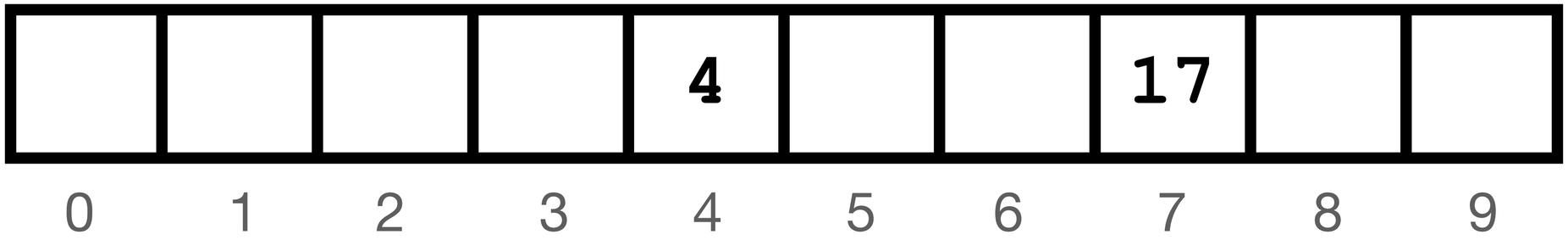
Direct Hashing

index = 17 % 10 = 7



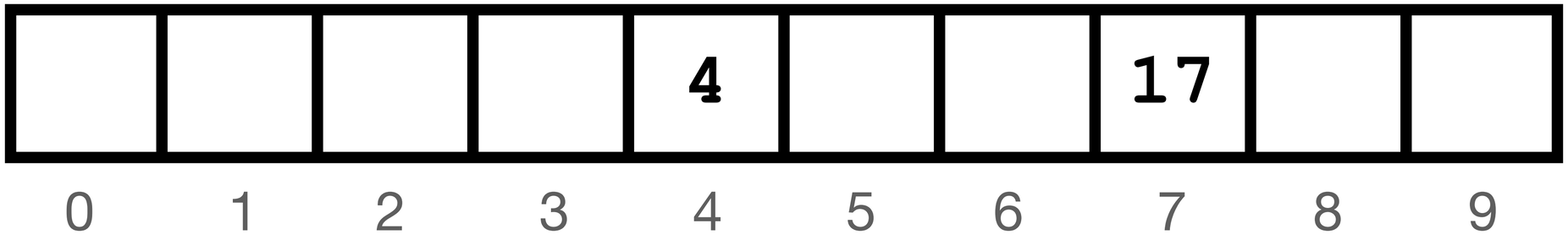
Direct Hashing

Insert 14 into table



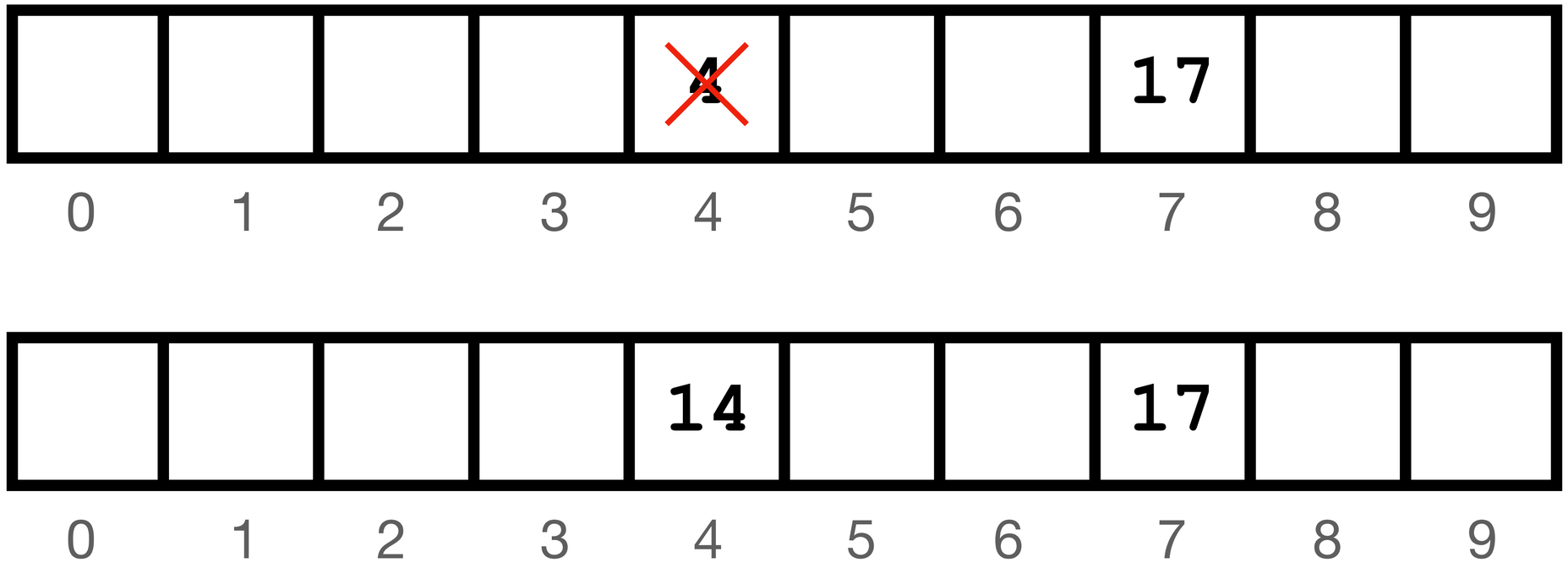
Direct Hashing

index = 14 % 10 = 4



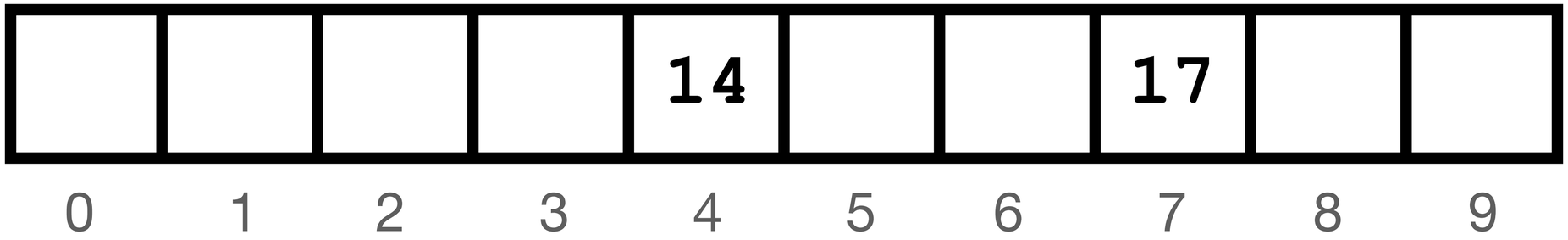
Direct Hashing

Collision at index 4
Overwrite data with 14



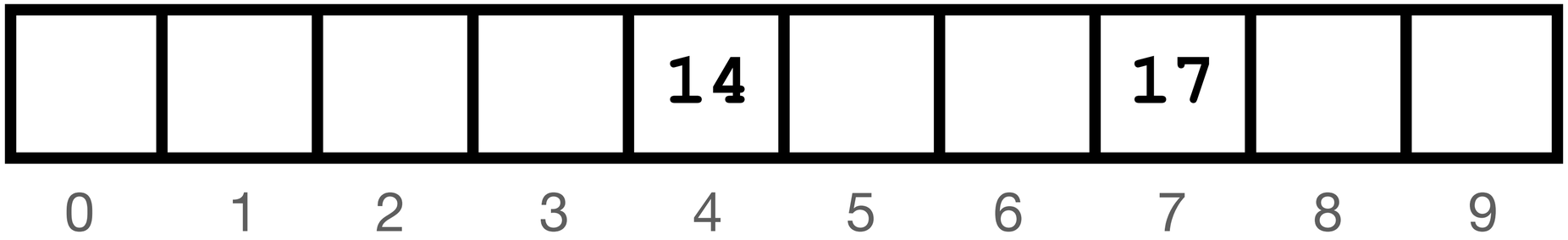
Direct Hashing

Insert 44 into table



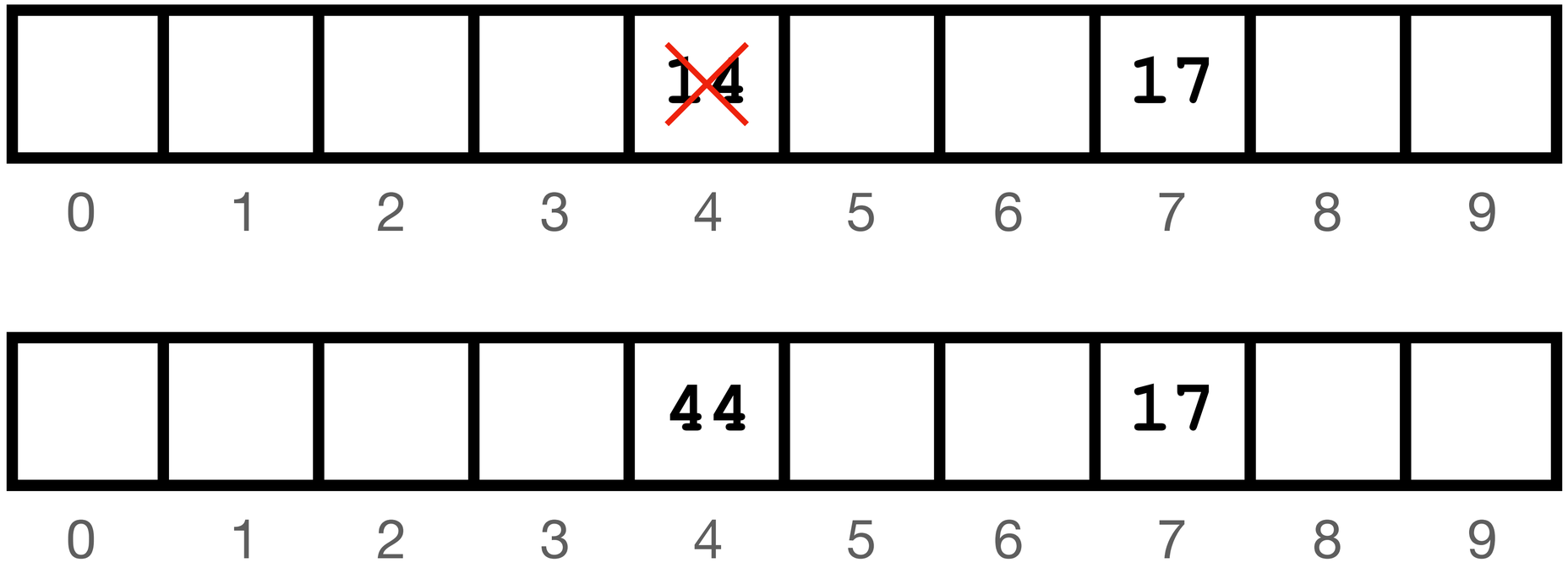
Direct Hashing

index = 44 % 10 = 4



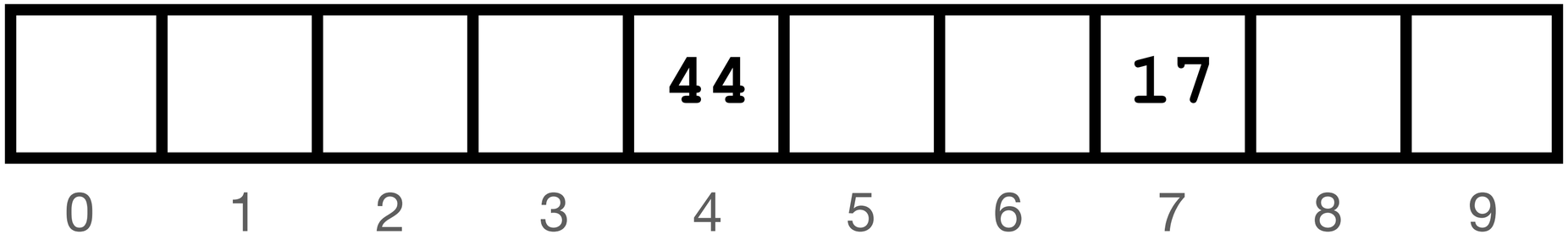
Direct Hashing

Collision at index 4
Overwrite data with 44



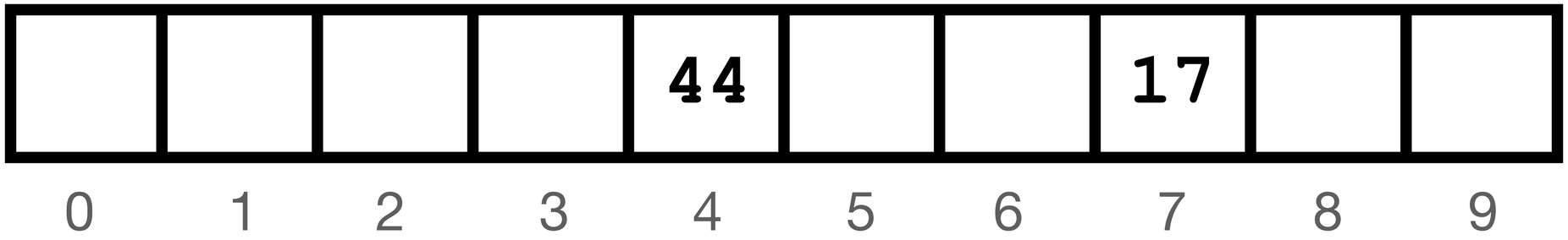
Direct Hashing

Insert 87 into table



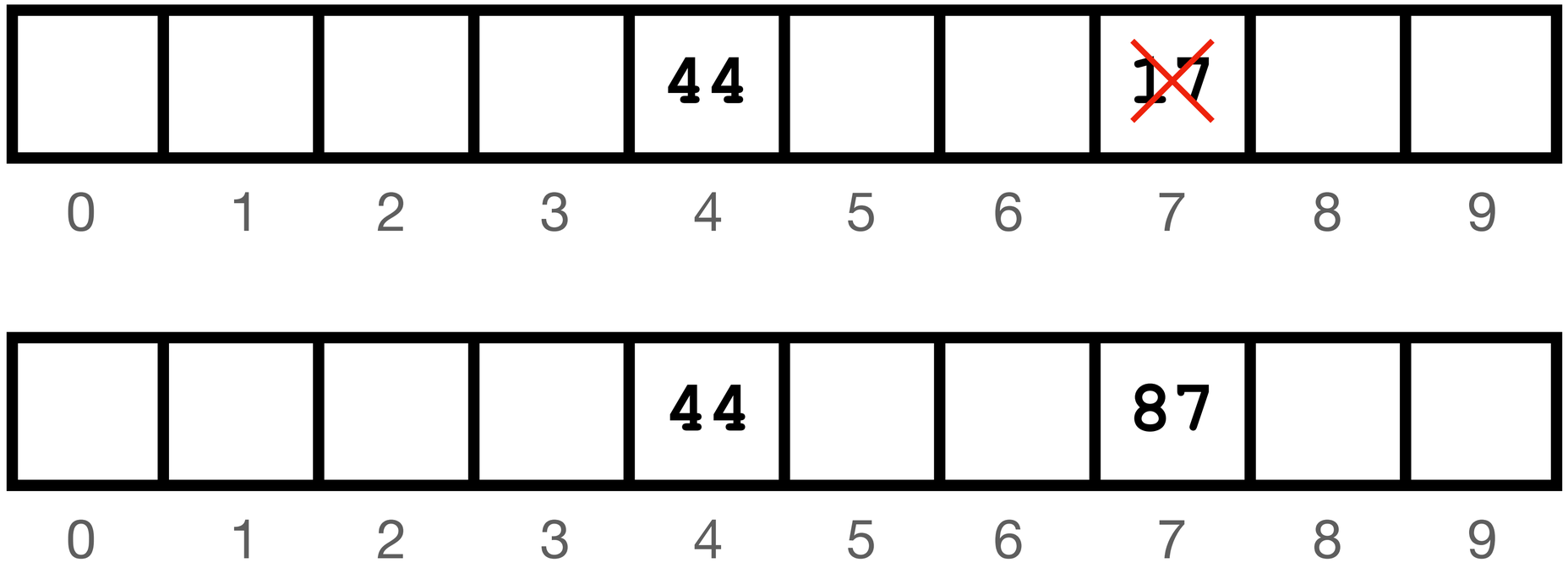
Direct Hashing

index = 87 % 10 = 7



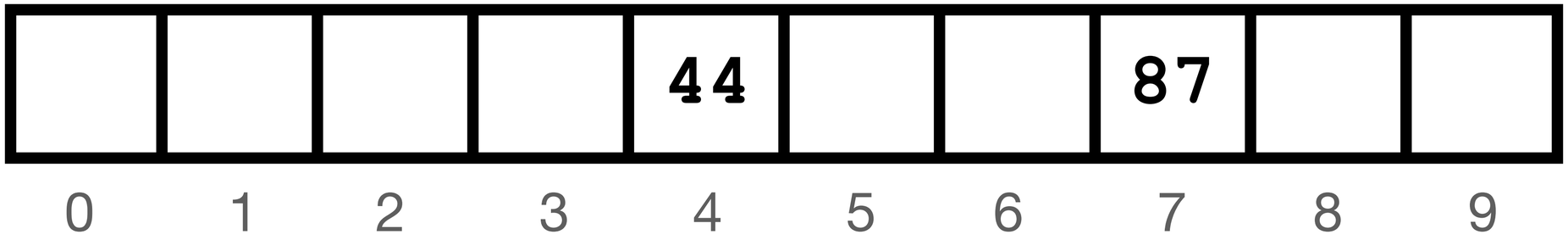
Direct Hashing

Collision at index 7
Overwrite data with 87



Direct Hashing

Final Table



Linear Probing

- **Increment i when collision happens**

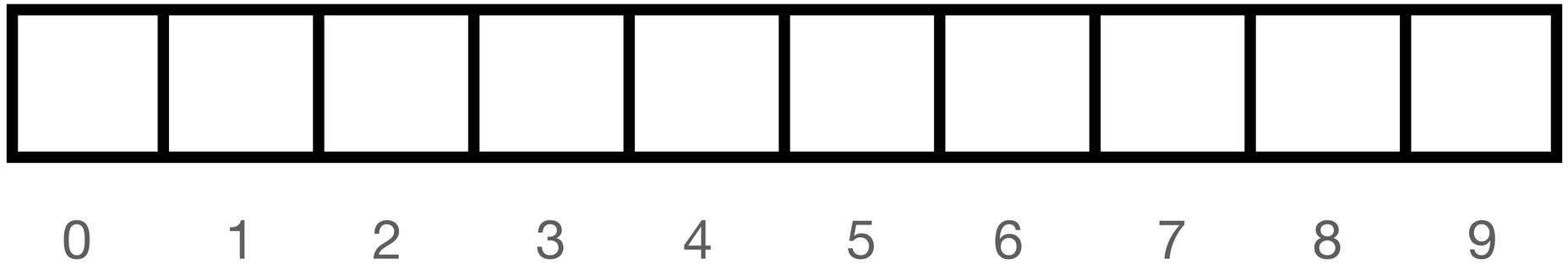
- **Get index by doing: ((x % size) + i) % size**

Linear Probing

```
void linearProbing(int table[], int size, int x){
  for(int i = 0; i < size; i++){
    int index = ((x % size) + i) % size;
    if(table[index] == -1){
      table[index] = x;
      return;
    }
  }
}
```

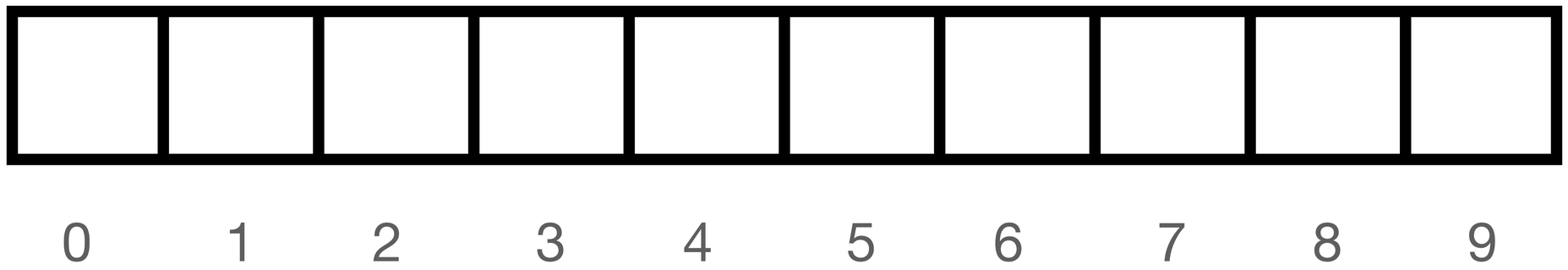
Linear Probing

Let’s start with a table of size 10



Linear Probing

Insert 5 into table



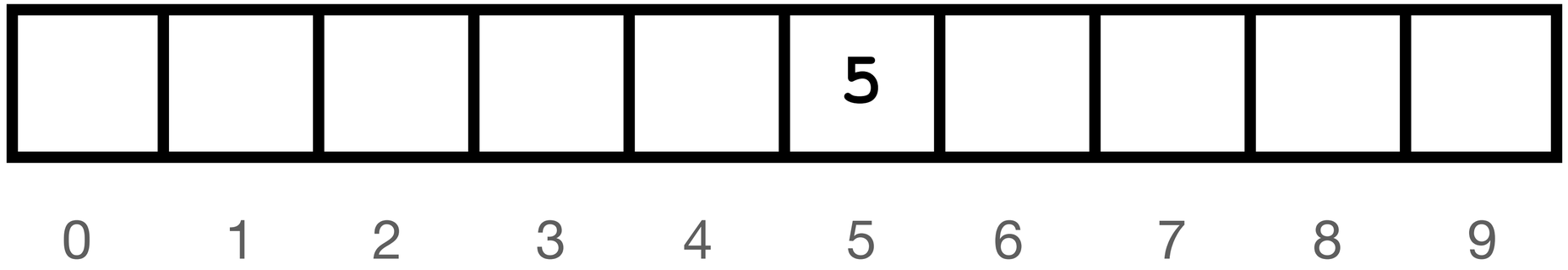
Linear Probing

index = ((5 % 10) + 0) % 10

= (5 + 0) % 10

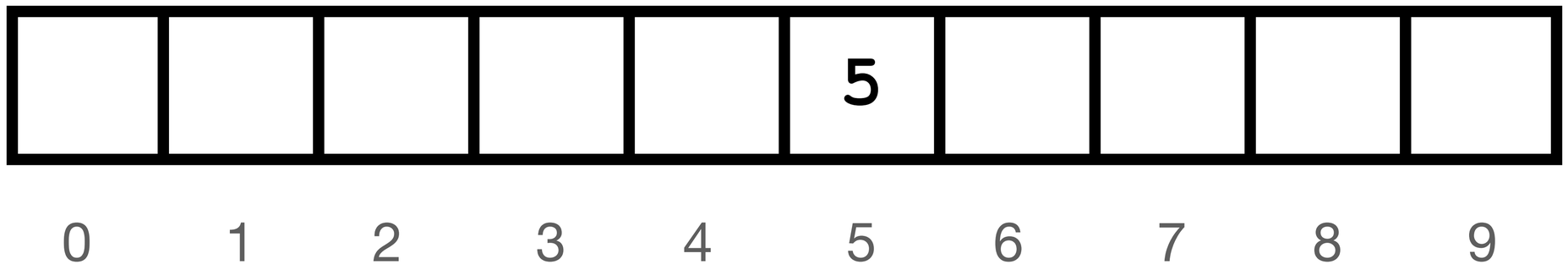
= 5 % 10

= 5



Linear Probing

Insert 27 into table



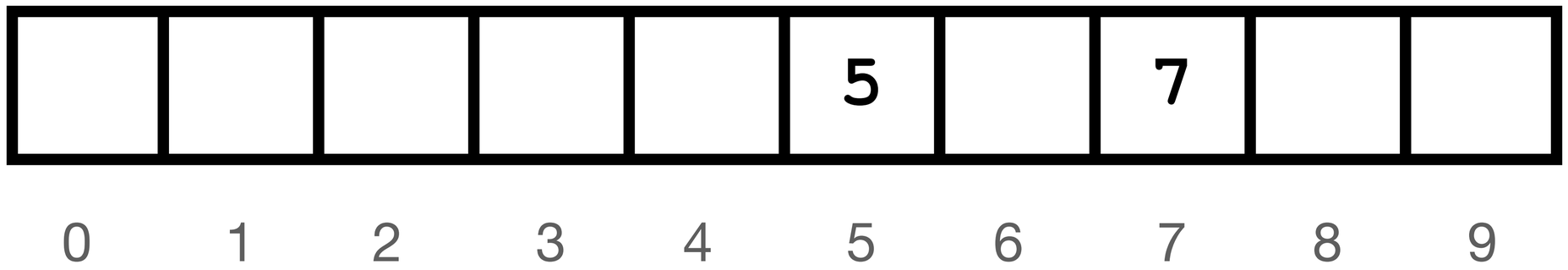
Linear Probing

index = ((27 % 10) + 0) % 10

= (7 + 0) % 10

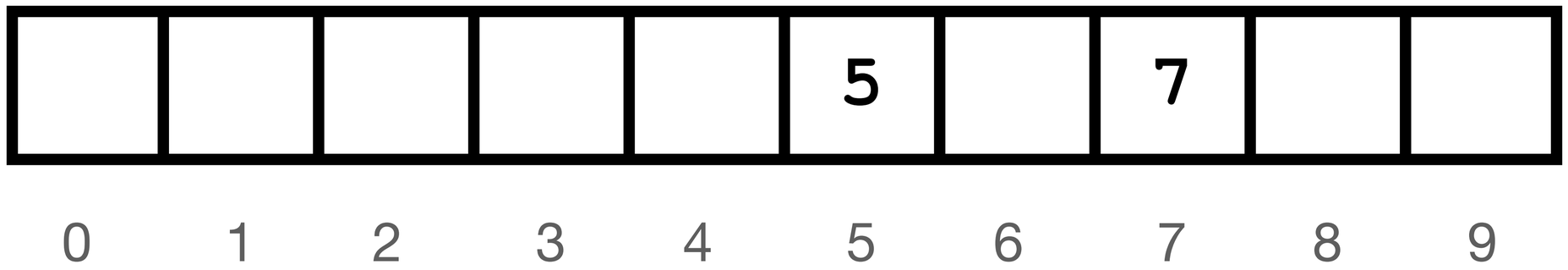
= 7 % 10

= 7



Linear Probing

Insert 45 into table



Linear Probing

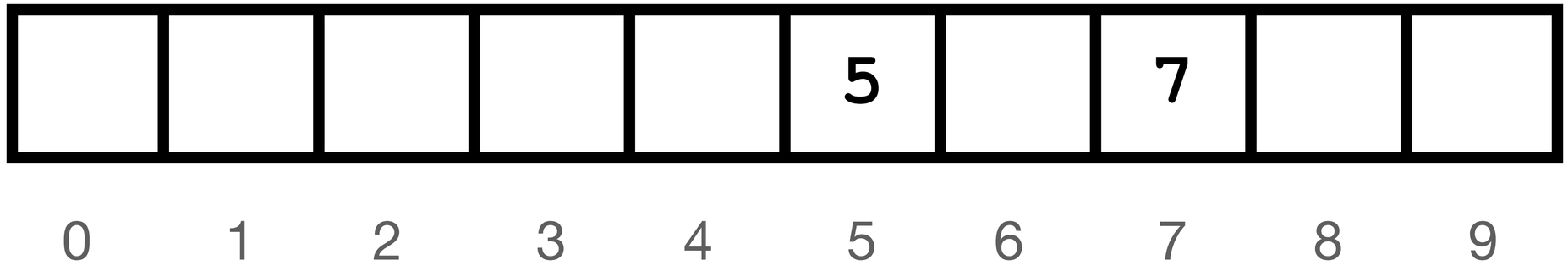
index = ((45 % 10) + 0) % 10

= (5 + 0) % 10

= 5 % 10

= 5

Collision at index 5, increment i by 1



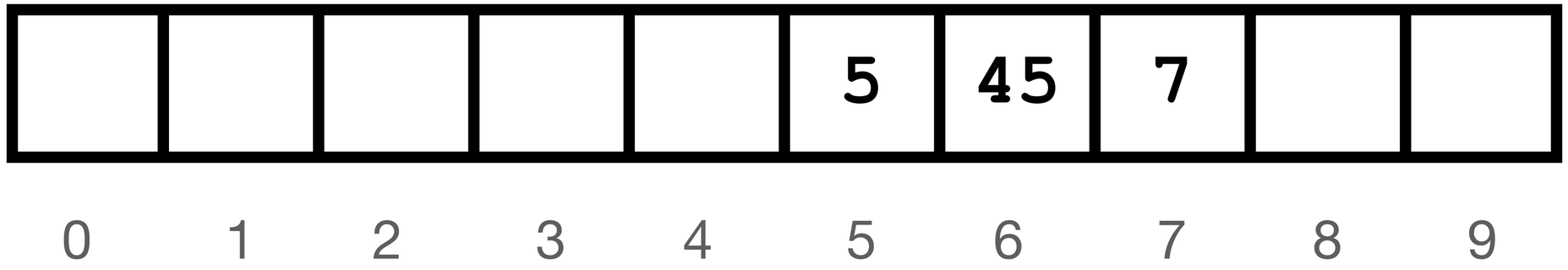
Linear Probing

index = ((45 % 10) + 1) % 10

= (5 + 1) % 10

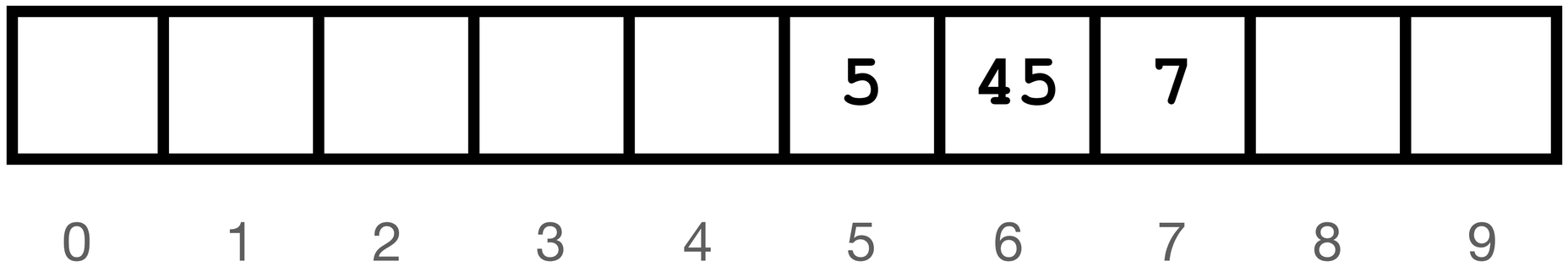
= 6 % 10

= 6



Linear Probing

Insert 75 into table



Linear Probing

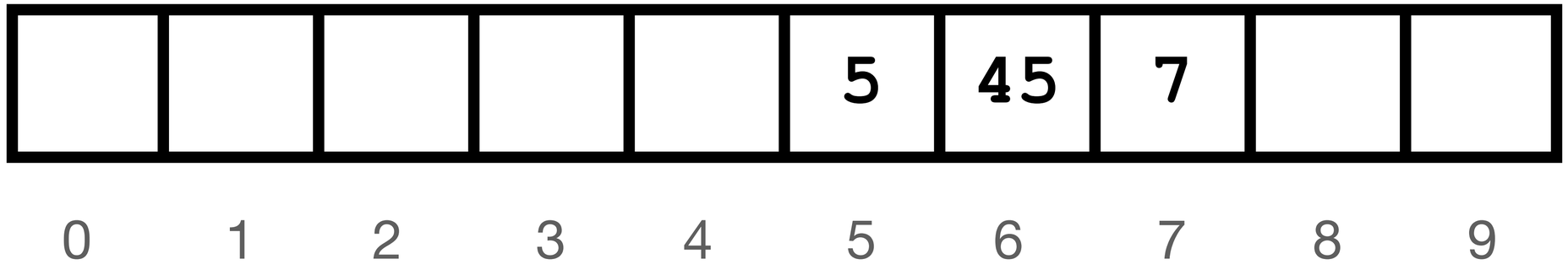
index = ((75 % 10) + 0) % 10

= (5 + 0) % 10

= 5 % 10

= 5

Collision at index 5, increment i by 1



Linear Probing

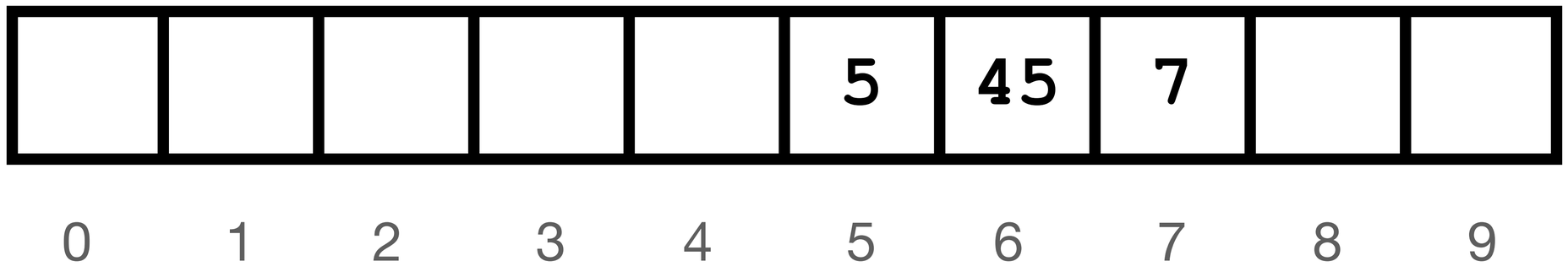
index = ((75 % 10) + 1) % 10

= (5 + 1) % 10

= 6 % 10

= 6

Collision at index 6, increment i by 1



Linear Probing

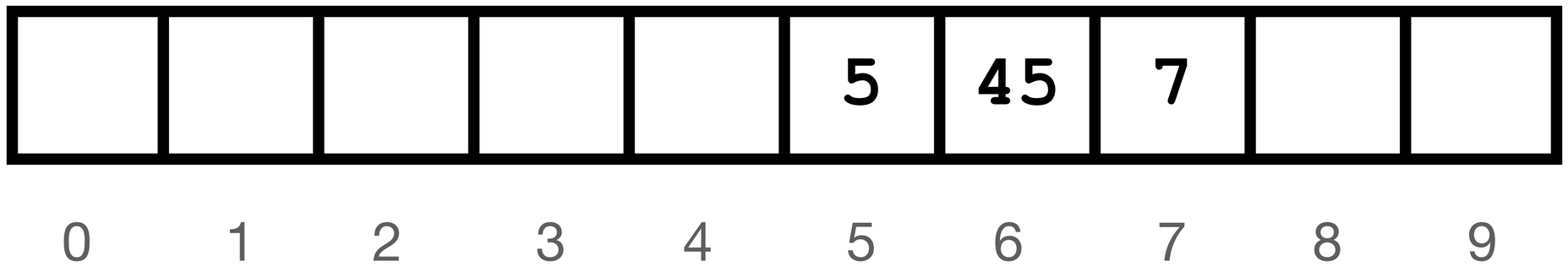
index = ((75 % 10) + 2) % 10

= (5 + 2) % 10

= 7 % 10

= 7

Collision at index 7, increment i by 1



Linear Probing

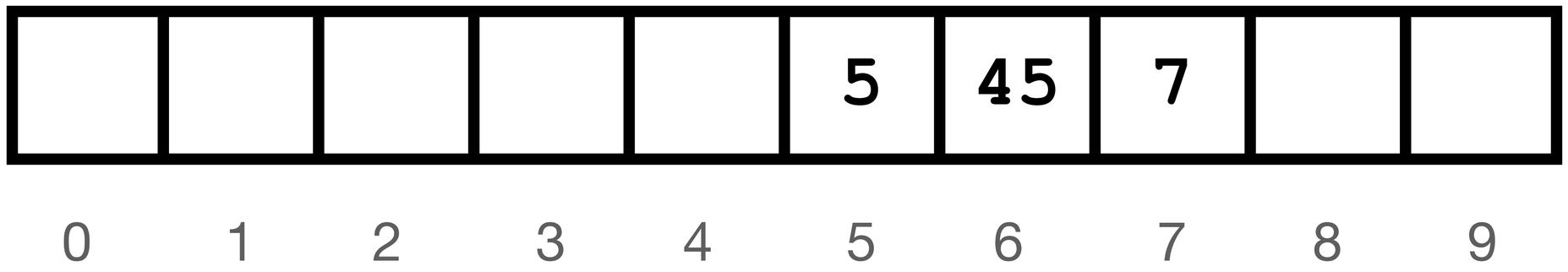
index = ((75 % 10) + 2) % 10

= (5 + 2) % 10

= 7 % 10

= 7

Collision at index 7, increment i by 1



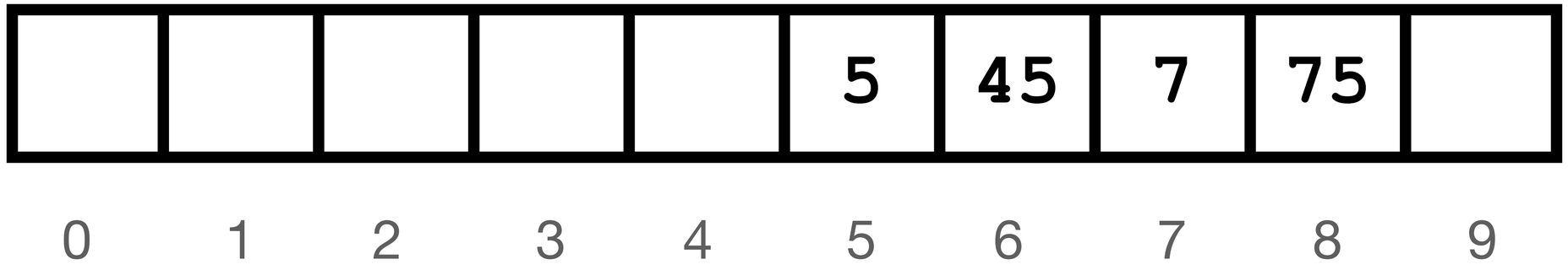
Linear Probing

index = ((75 % 10) + 3) % 10

= (5 + 3) % 10

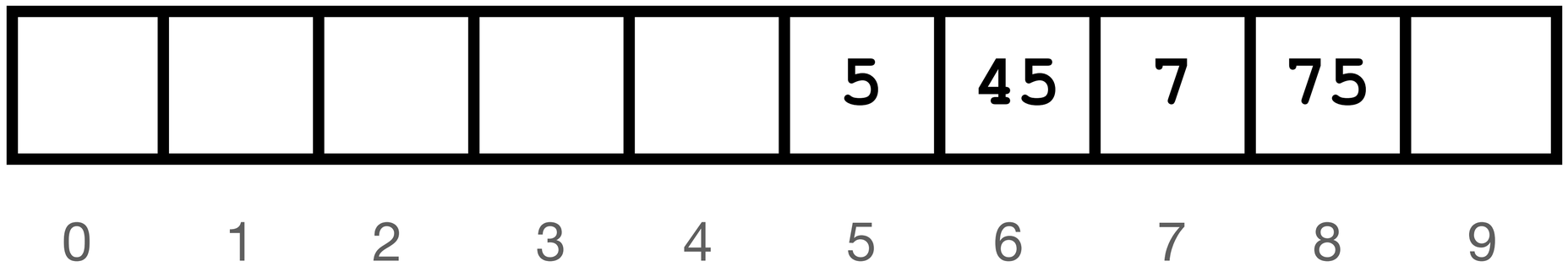
= 8 % 10

= 8



Linear Probing

Insert 99 into table



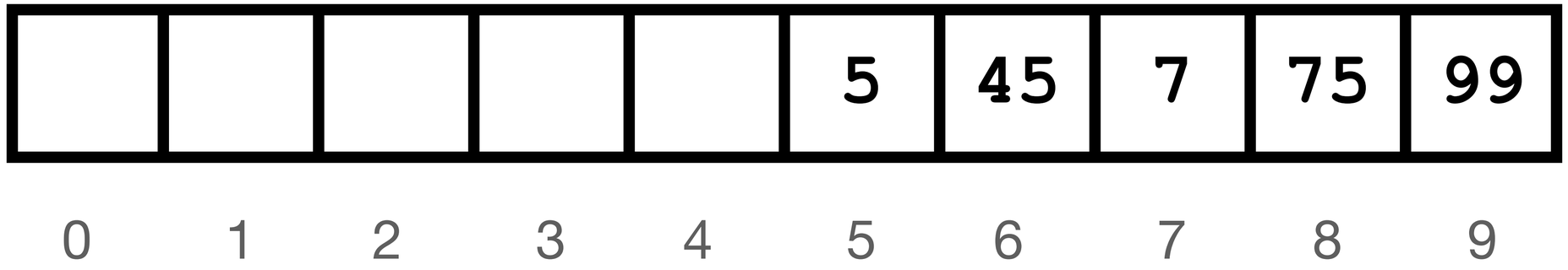
Linear Probing

index = ((99 % 10) + 0) % 10

= (9 + 0) % 10

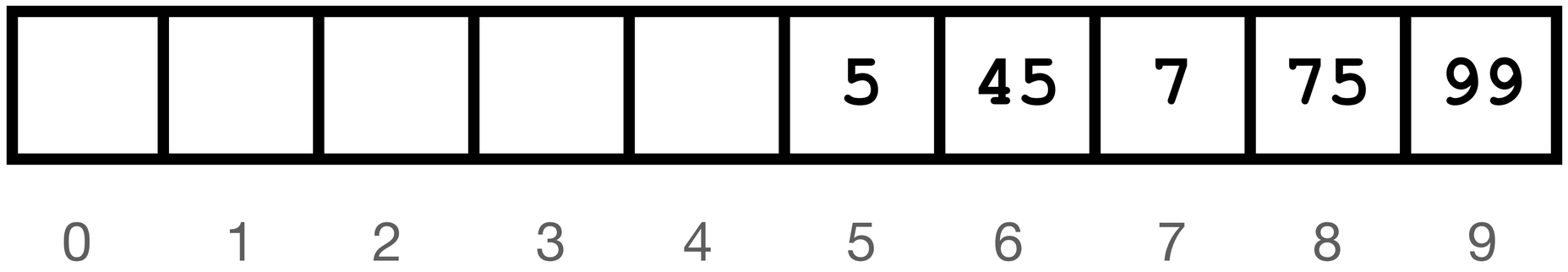
= 9 % 10

= 9



Linear Probing

Insert 19 into table



Linear Probing

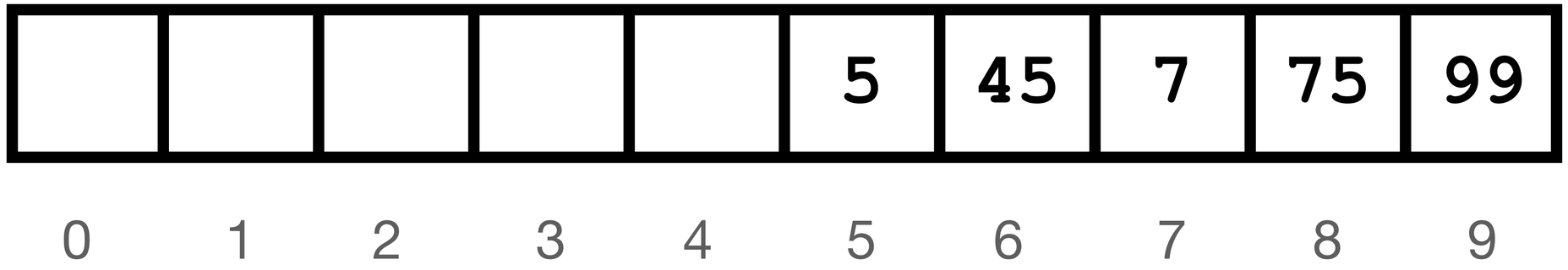
index = ((19 % 10) + 0) % 10

= (9 + 0) % 10

= 9 % 10

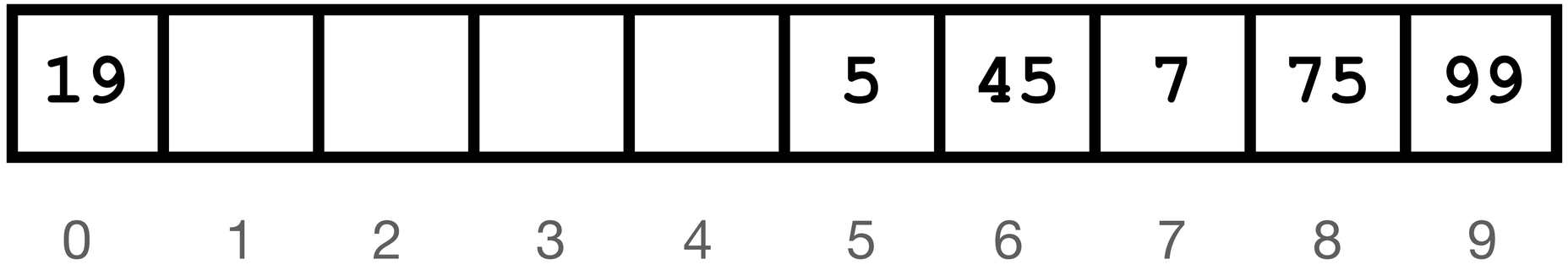
= 9

Collision at index 9, increment i by 1



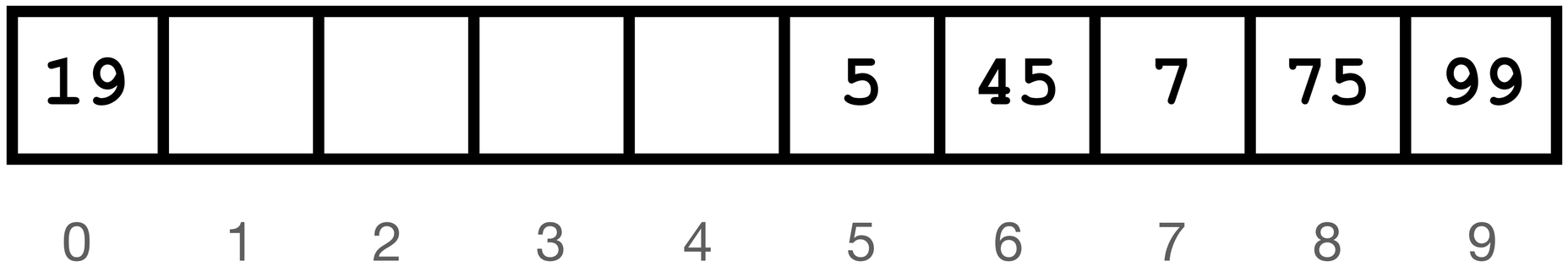
Linear Probing

index = ((19 % 10) + 1) % 10
= (9 + 1) % 10
= 10 % 10
= 0



Linear Probing

Final Table



Quadratic Probing

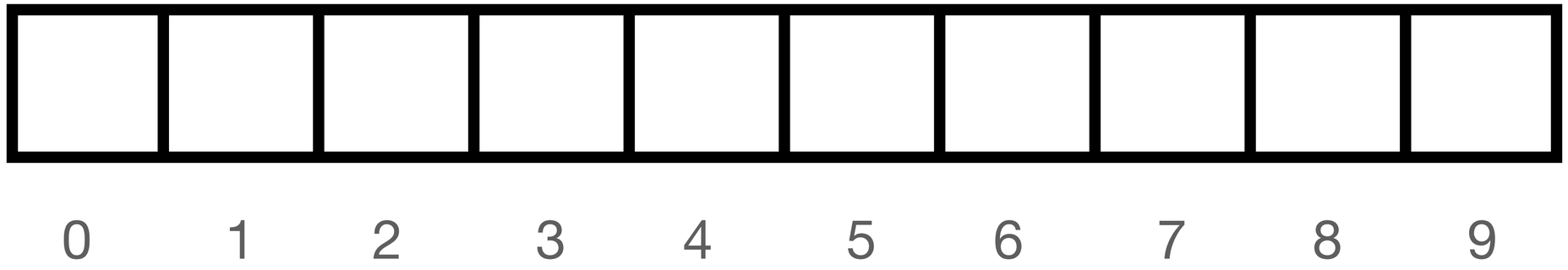
- Increment i when collision happens
- We do i^2
- Get index by doing: $((x \% size) + i^2) \% size$
- Notice that it's similar to linear probing except we have i^2 instead of just i

Quadratic Probing

```
void quadraticProbing(int table[], int size, int x){
  for(int i = 0; i < size; i++){
    int index = ((x % size) + (i*i)) % size;
    if(table[index] == -1){
      table[index] = x;
      return;
    }
  }
}
```

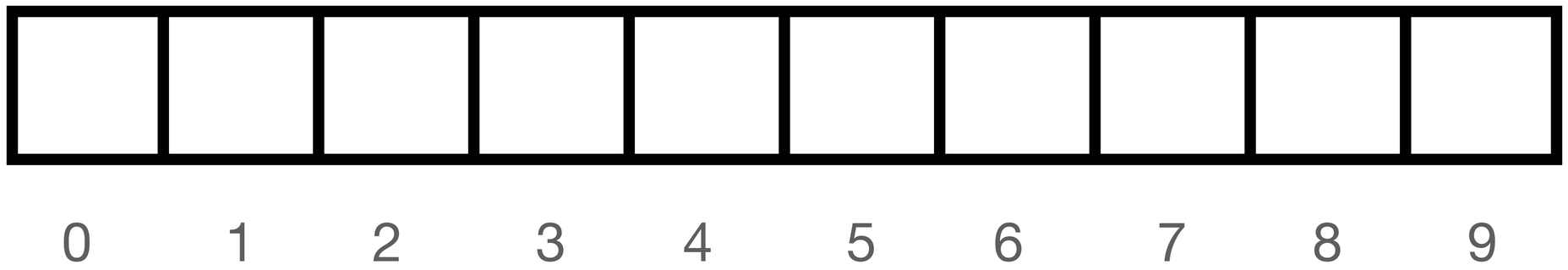
Quadratic Probing

Let’s start with a table of size 10



Quadratic Probing

Insert 79 into table



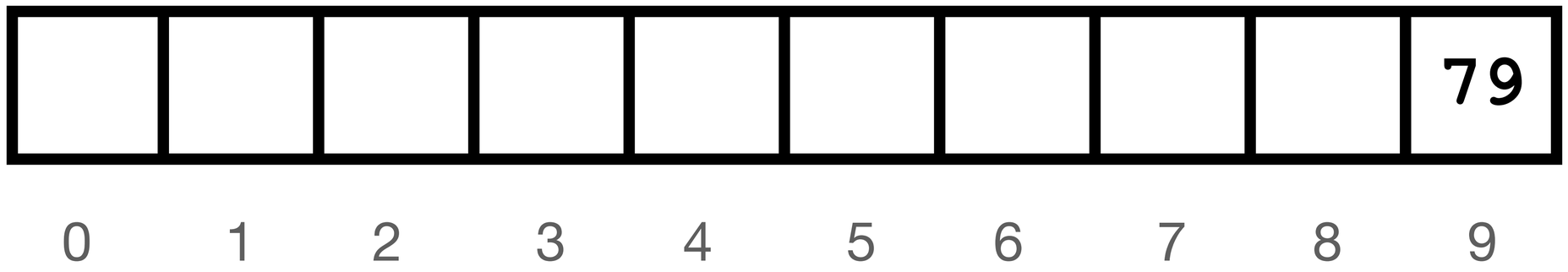
Quadratic Probing

index = ((79 % 10) + 0^2) % 10

= (9 + 0) % 10

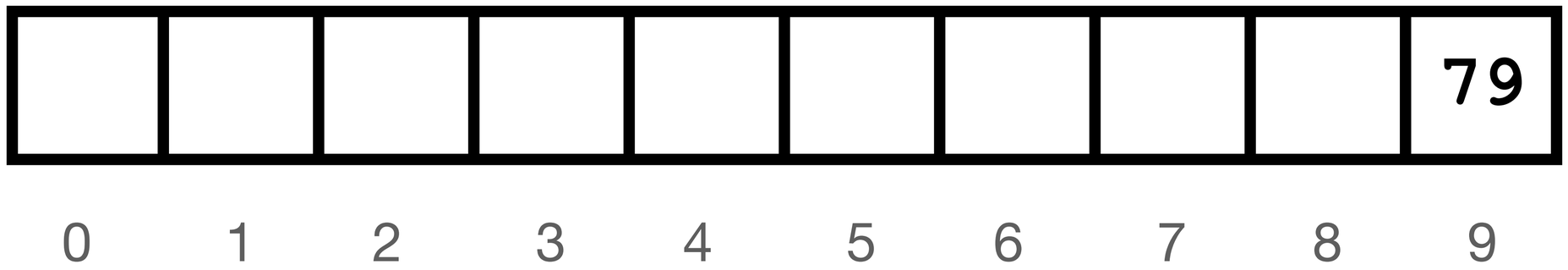
= 9 % 10

= 9



Quadratic Probing

Insert 83 into table



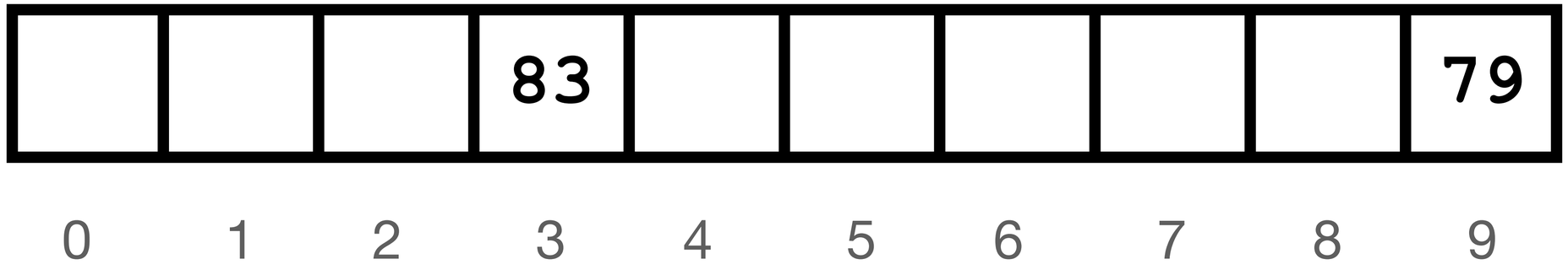
Quadratic Probing

index = ((83 % 10) + 0^2) % 10

= (3 + 0) % 10

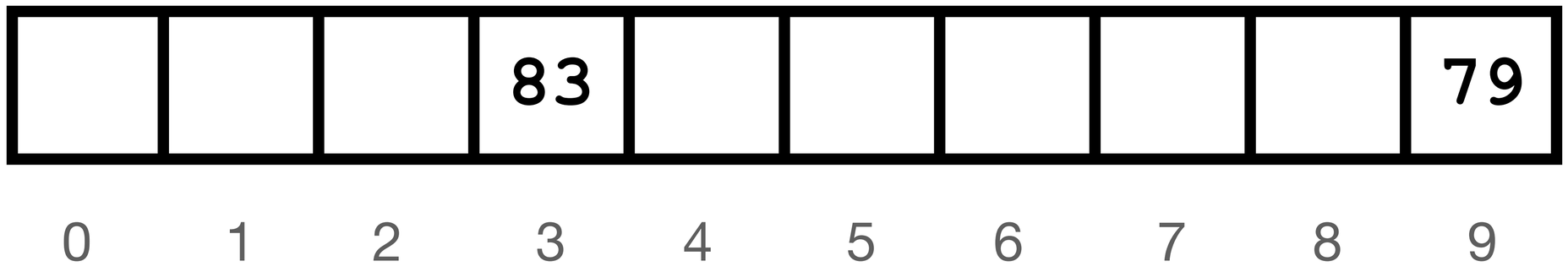
= 3 % 10

= 3



Quadratic Probing

Insert 19 into table



Quadratic Probing

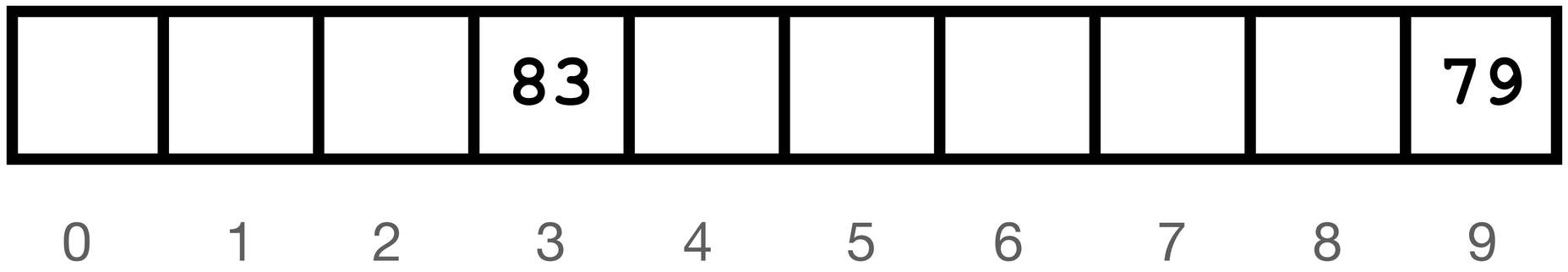
index = ((19 % 10) + 0^2) % 10

= (9 + 0) % 10

= 9 % 10

= 9

Collision at index 9, increment i by 1



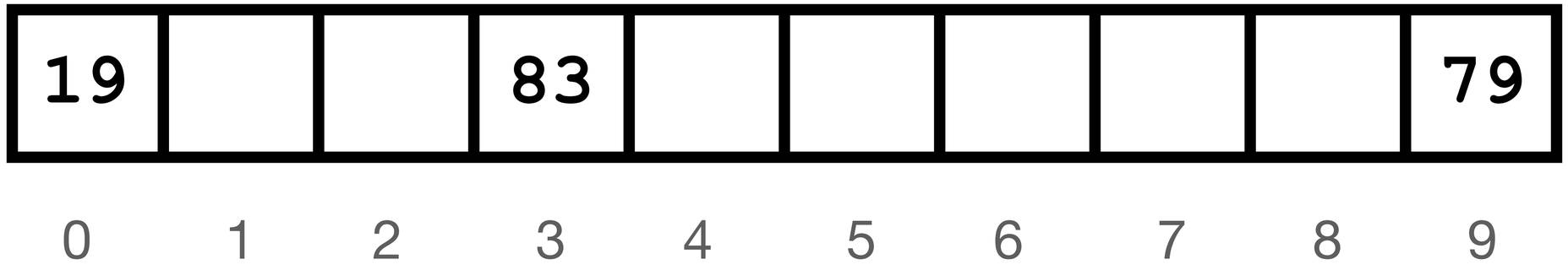
Quadratic Probing

index = ((19 % 10) + 1^2) % 10

= (9 + 1) % 10

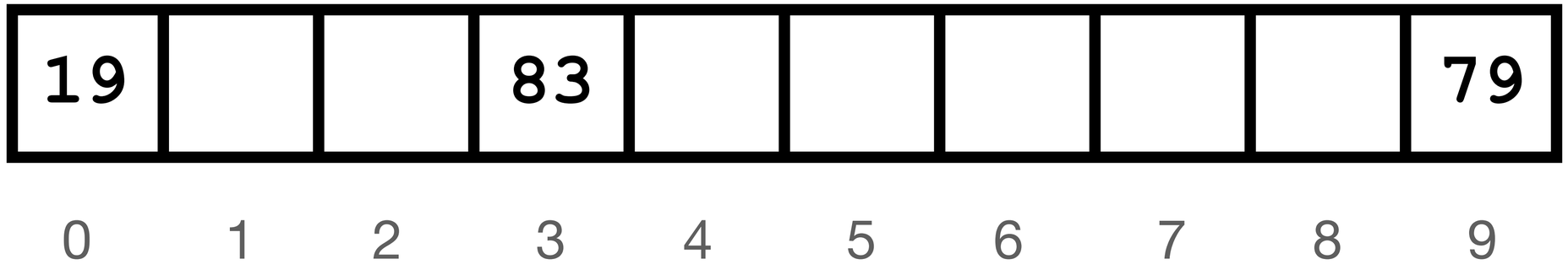
= 10 % 10

= 0



Quadratic Probing

Insert 49 into table



Quadratic Probing

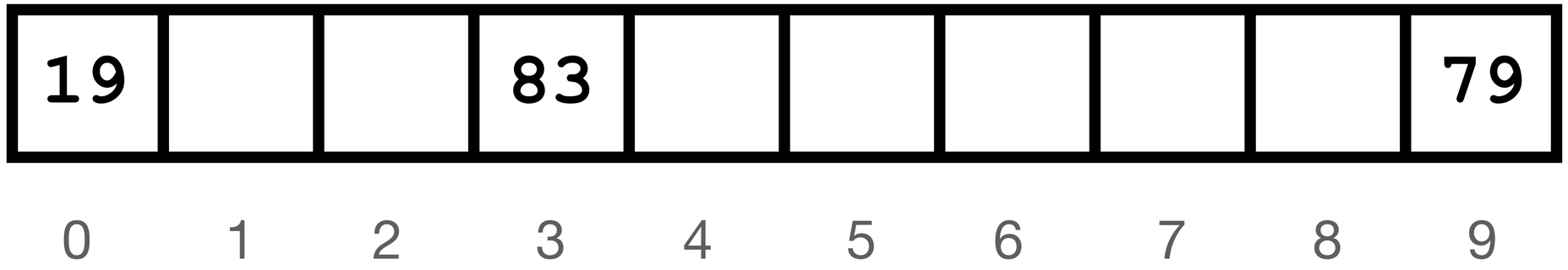
index = ((49 % 10) + 0^2) % 10

= (9 + 0) % 10

= 9 % 10

= 9

Collision at index 9, increment i by 1



Quadratic Probing

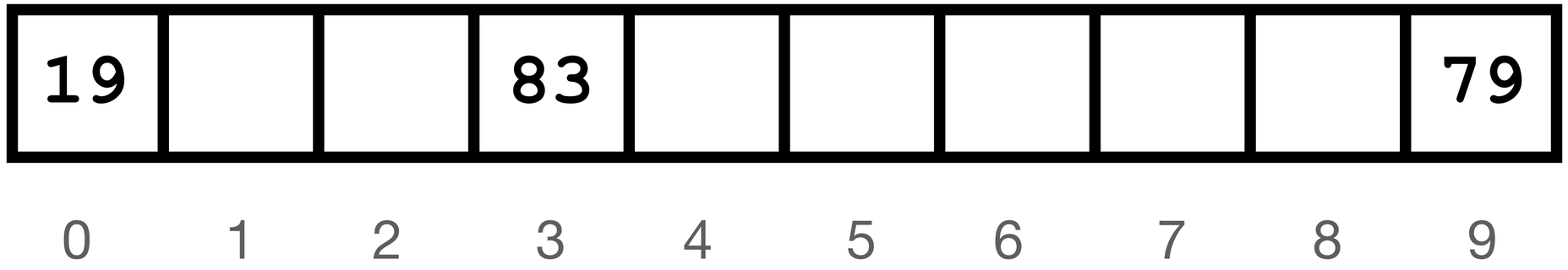
index = ((49 % 10) + 1^2) % 10

= (9 + 1) % 10

= 10 % 10

= 0

Collision at index 0, increment i by 1



Quadratic Probing

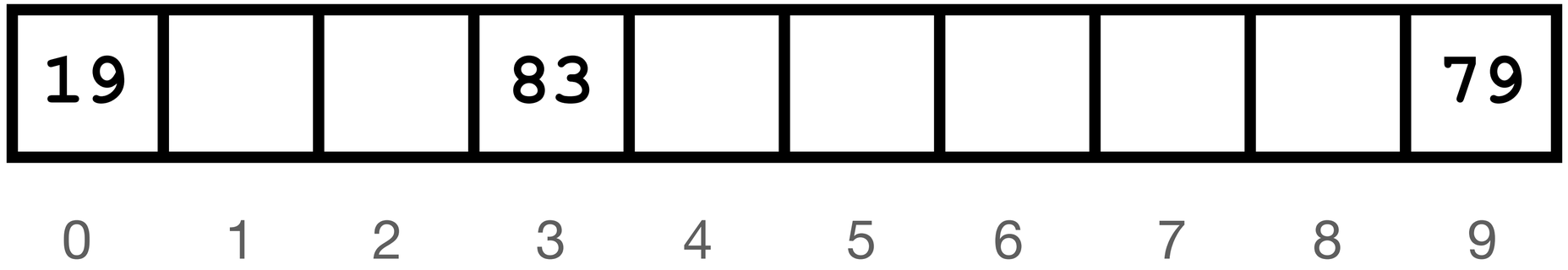
index = ((49 % 10) + 2^2) % 10

= (9 + 4) % 10

= 13 % 10

= 3

Collision at index 3, increment i by 1



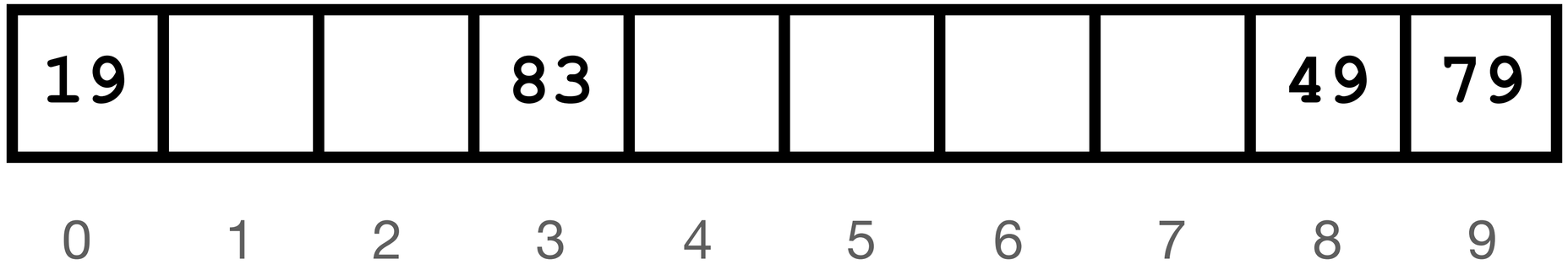
Quadratic Probing

index = ((49 % 10) + 3^2) % 10

= (9 + 9) % 10

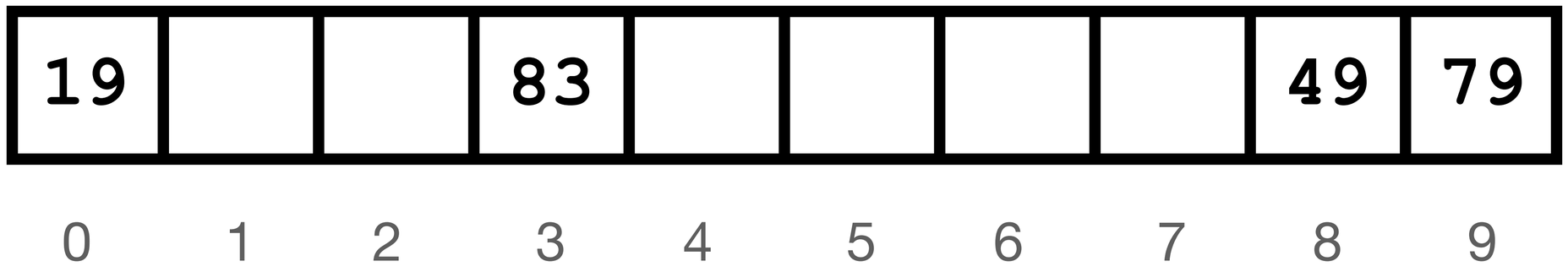
= 18 % 10

= 8



Quadratic Probing

Insert 3 into table



Quadratic Probing

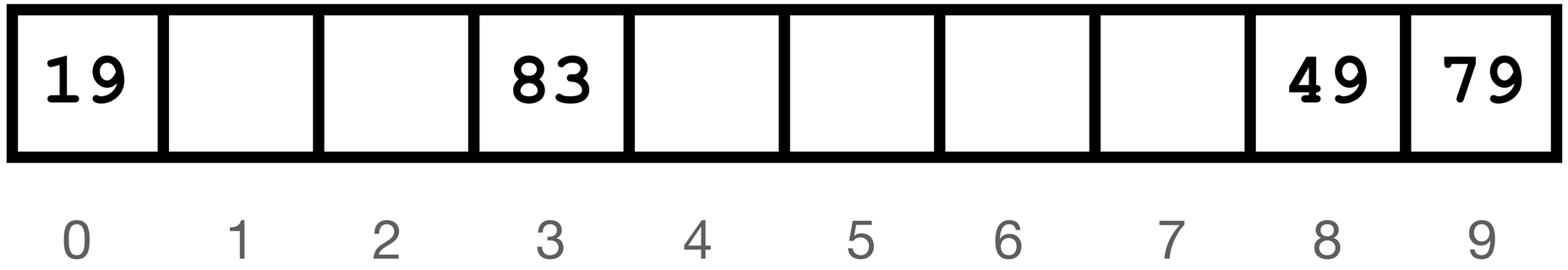
index = ((3 % 10) + 0^2) % 10

= (3 + 0) % 10

= 3 % 10

= 3

Collision at index 3, increment i by 1



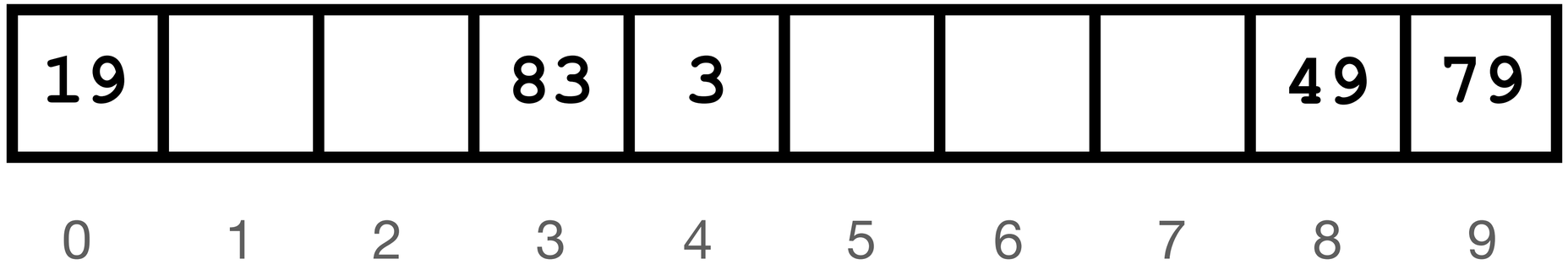
Quadratic Probing

index = ((3 % 10) + 1²) % 10

= (3 + 1) % 10

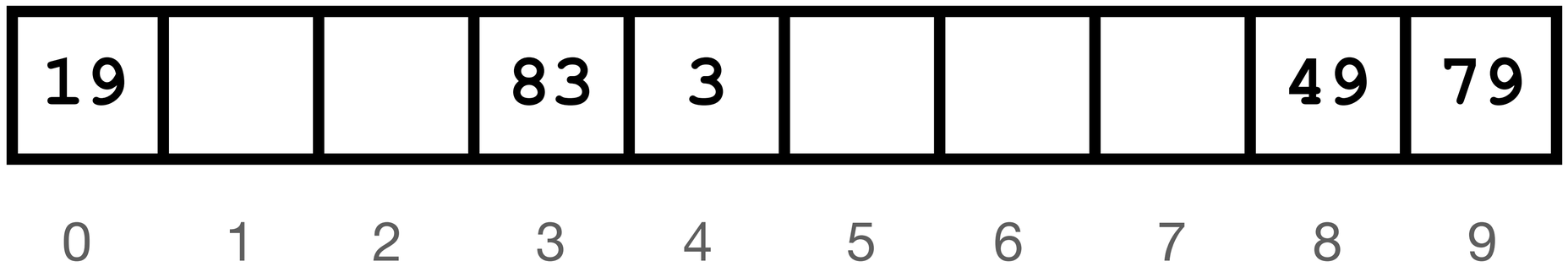
= 4 % 10

= 4



Quadratic Probing

Insert 73 into table



Quadratic Probing

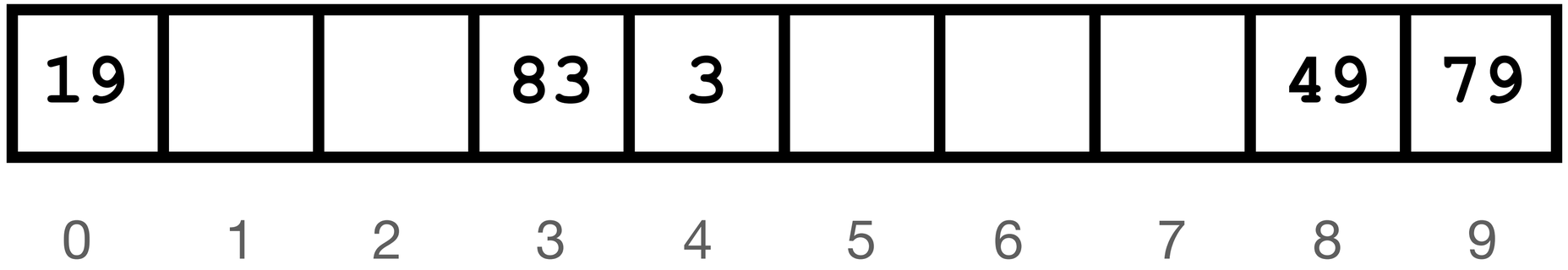
index = ((73 % 10) + 0^2) % 10

= (3 + 0) % 10

= 3 % 10

= 3

Collision at index 3, increment i by 1



Quadratic Probing

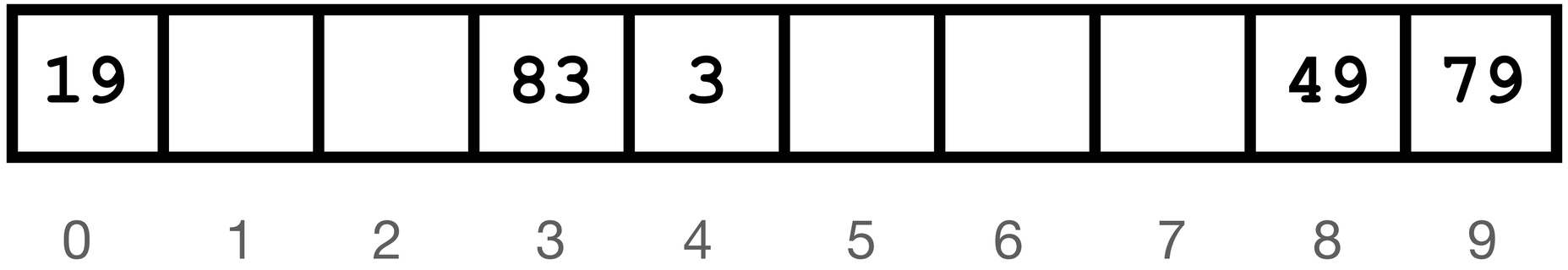
index = ((73 % 10) + 1^2) % 10

= (3 + 1) % 10

= 4 % 10

= 4

Collision at index 4, increment i by 1



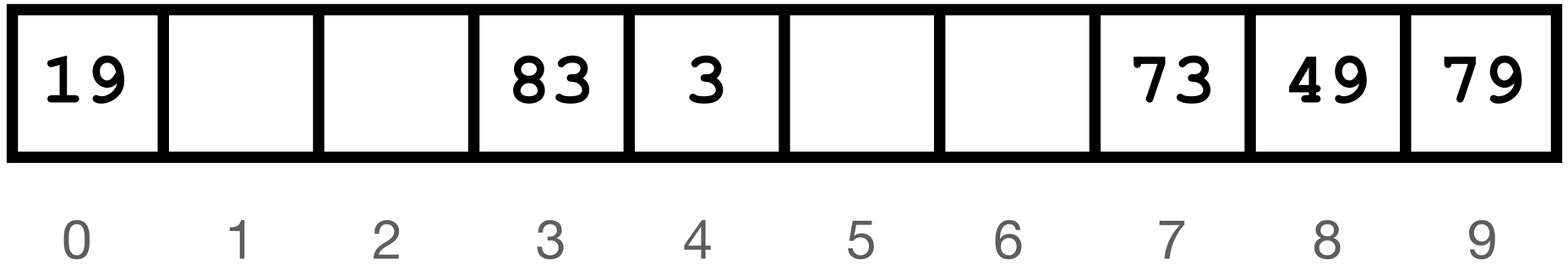
Quadratic Probing

index = ((73 % 10) + 2^2) % 10

= (3 + 4) % 10

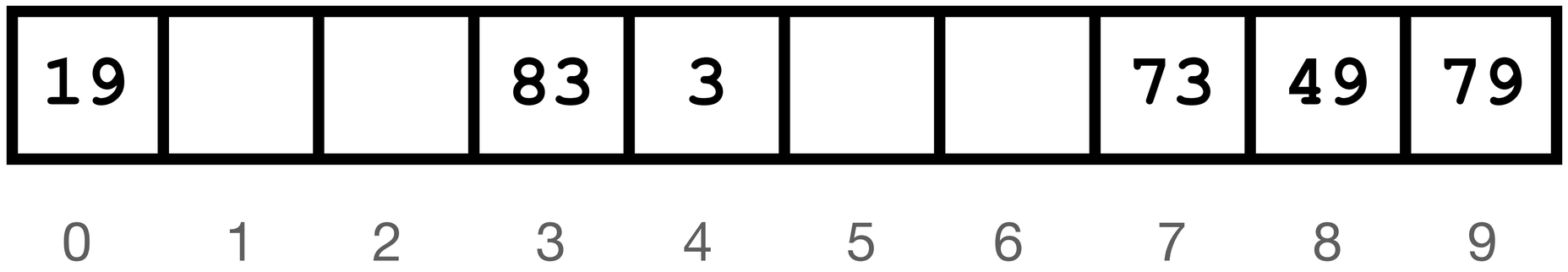
= 7 % 10

= 7



Quadratic Probing

Final Table



Double Hashing

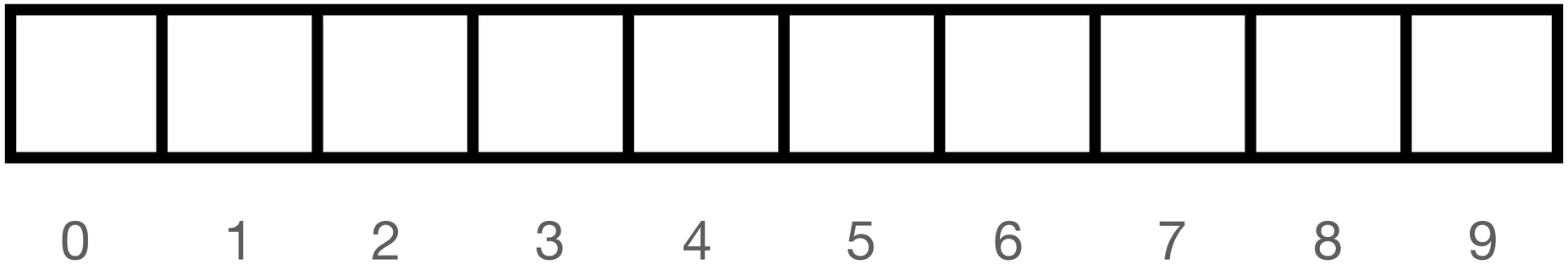
- **We use two hash function**
- **We increment i every time there is a collision**
- **((x % size) + (i * (prime - (x % prime))) % size**

Double Hashing

```
void doubleHashing(int table[], int size, int x){
  for(int i = 0; i < size; i++){
    int index = (hash1(x) + (i * hash2(x)) % size;
    if(table[index] == -1){
      table[index] = x;
      return;
    }
  }
}
```

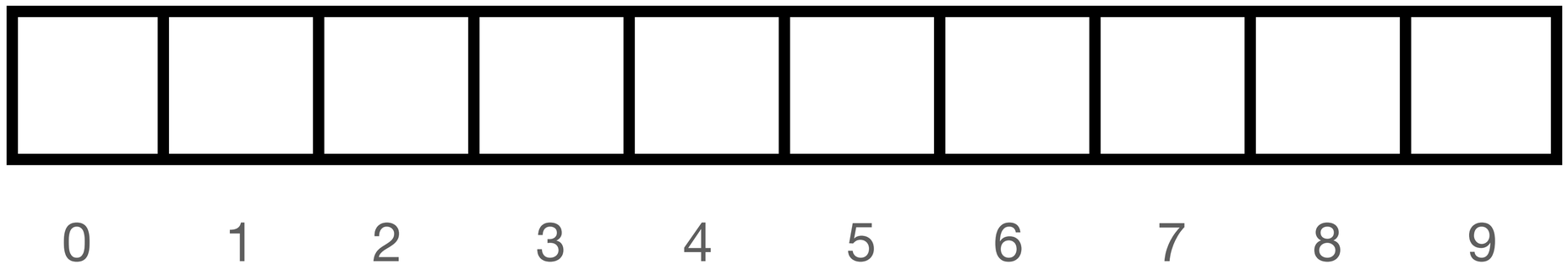
Double Hashing

Let’s start with a table of size 10



Double Hashing

Insert 33 into table



Double Hashing

index = ((33 % 10) + (0 * (7 - (33 % 7)))) % 10

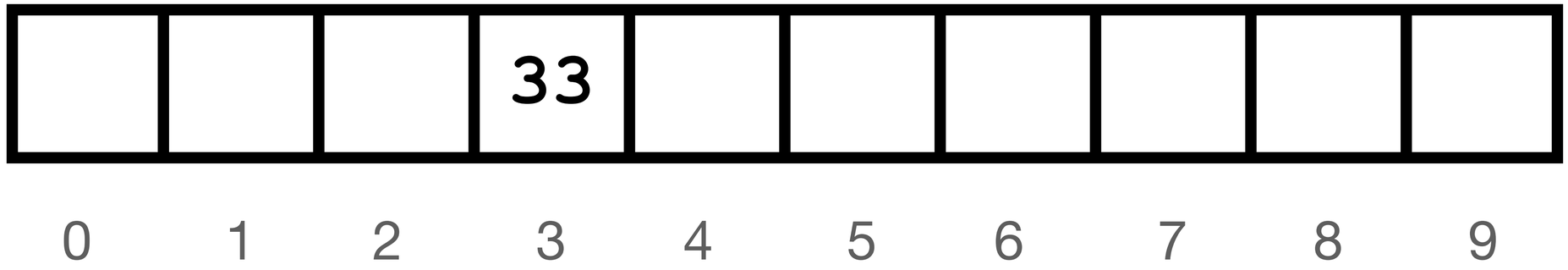
= (3 + (0 * (7 - 5))) % 10

= (3 + (0 * 2)) % 10

= (3 + 0) % 10

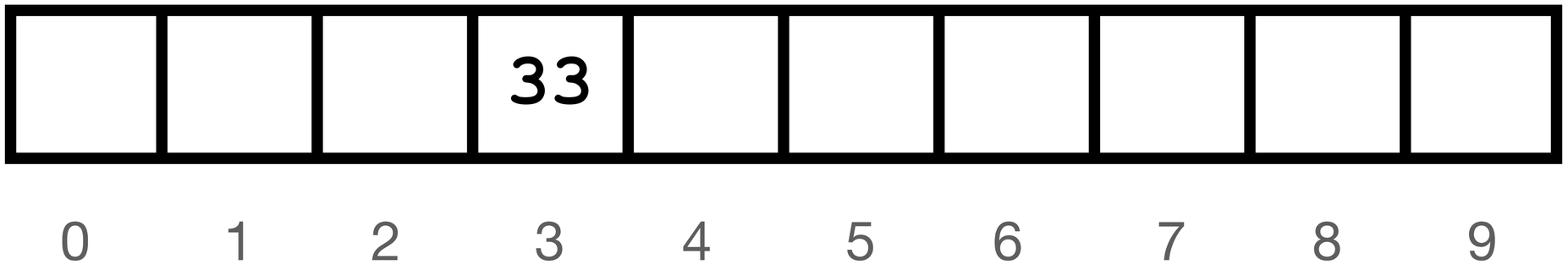
= 3 % 10

= 3



Double Hashing

Insert 25 into table



Double Hashing

index = ((25 % 10) + (0 * (7 - (25 % 7)))) % 10

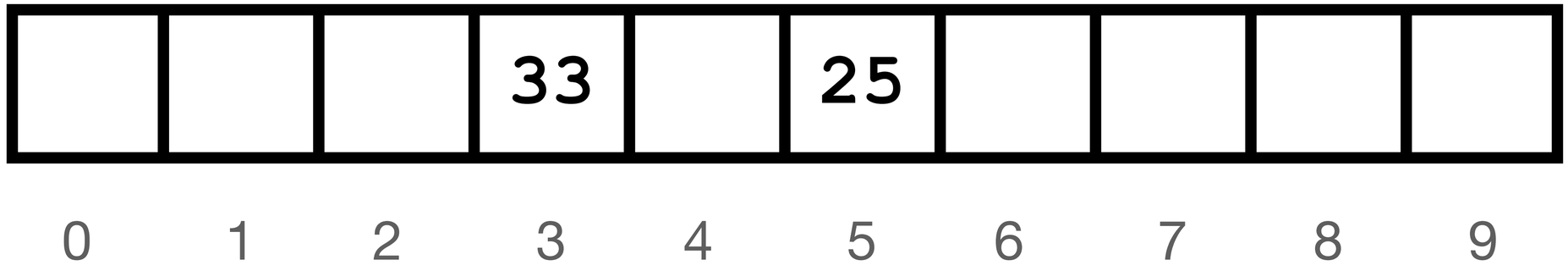
= (5 + (0 * (7 - 3))) % 10

= (5 + (0 * 4)) % 10

= (5 + 0) % 10

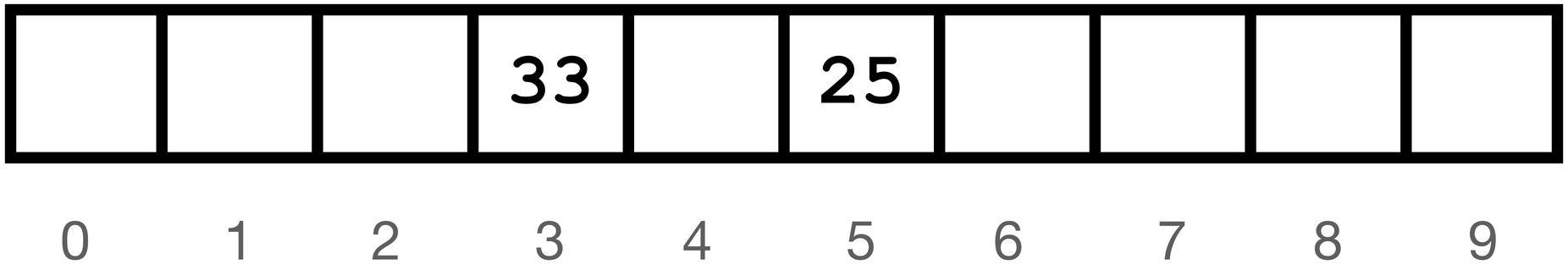
= 5 % 10

= 5



Double Hashing

Insert 13 into table



Double Hashing

index = ((13 % 10) + (0 * (7 - (13 % 7)))) % 10

= (3 + (0 * (7 - 6))) % 10

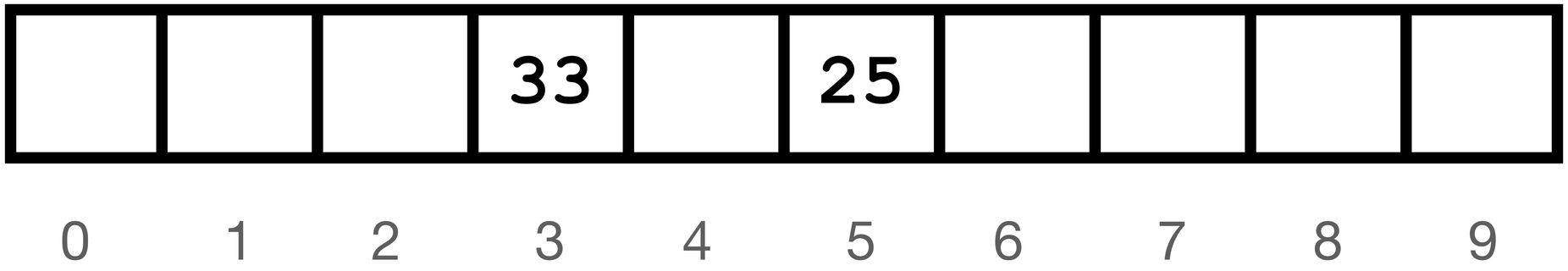
= (3 + (0 * 1)) % 10

= (3 + 0) % 10

= 3 % 10

= 3

Collision at 3, increment i by 1



Double Hashing

index = ((13 % 10) + (1 * (7 - (13 % 7)))) % 10

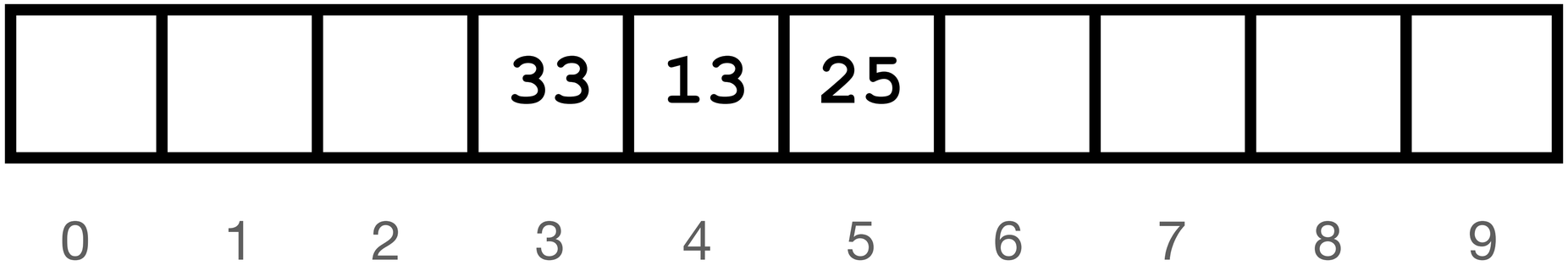
= (3 + (1 * (7 - 6))) % 10

= (3 + (1 * 1)) % 10

= (3 + 1) % 10

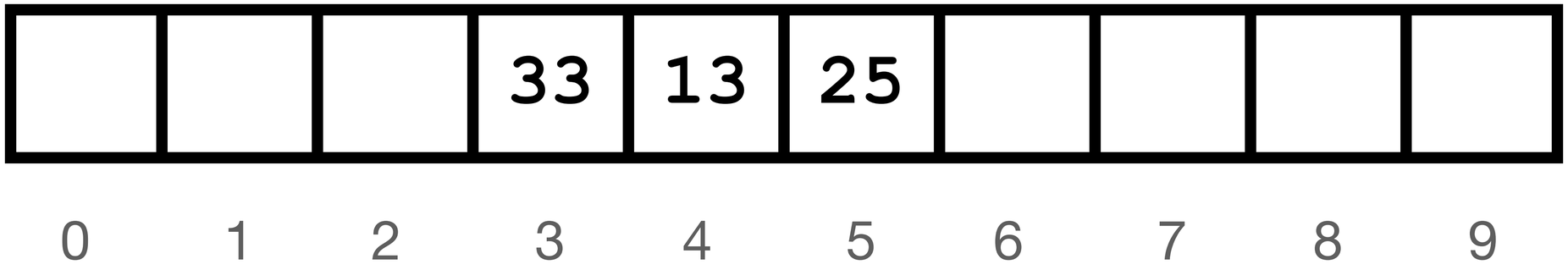
= 4 % 10

= 4



Double Hashing

Insert 43 into table



Double Hashing

index = ((43 % 10) + (0 * (7 - (43 % 7)))) % 10

= (3 + (0 * (7 - 1))) % 10

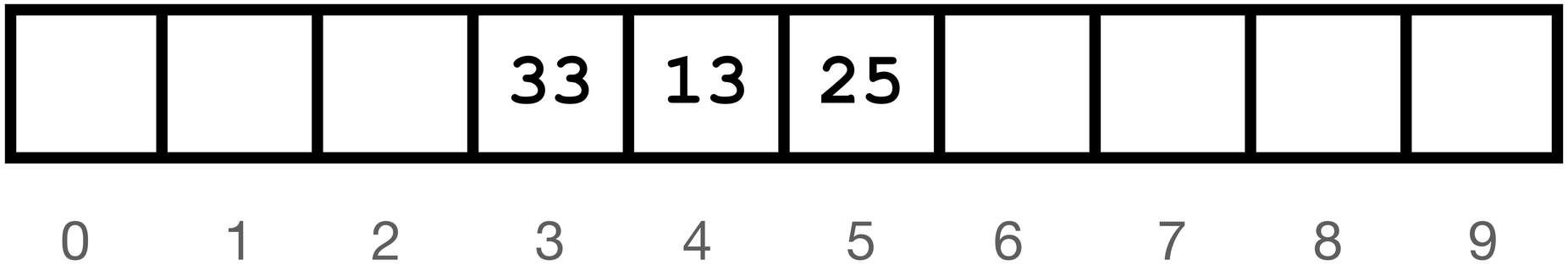
= (3 + (0 * 6)) % 10

= (3 + 0) % 10

= 3 % 10

= 3

Collision at 3, increment i by 1



Double Hashing

index = ((43 % 10) + (1 * (7 - (43 % 7)))) % 10

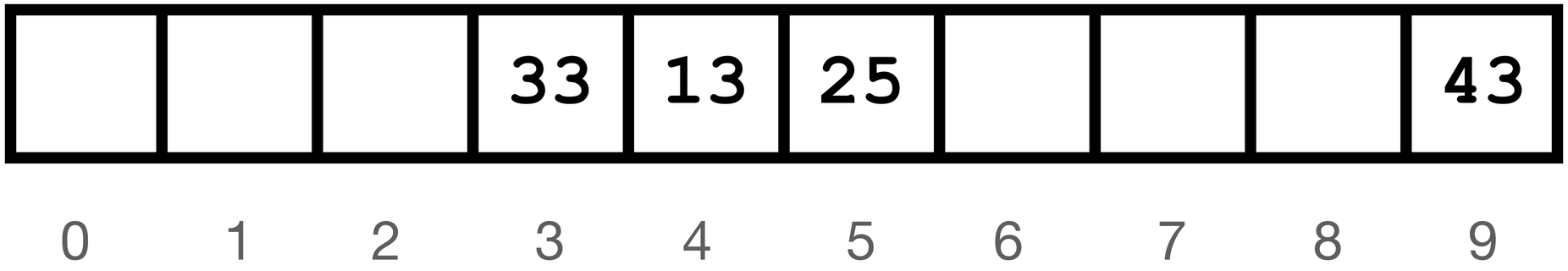
= (3 + (1 * (7 - 1))) % 10

= (3 + (1 * 6)) % 10

= (3 + 6) % 10

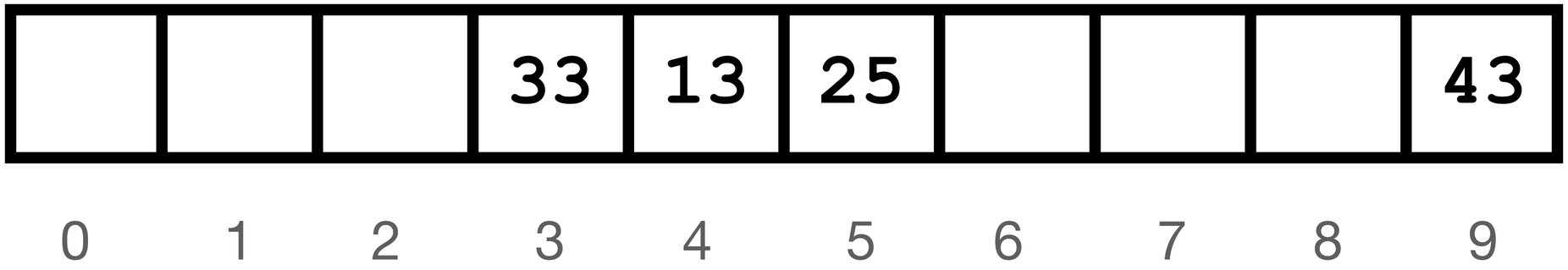
= 9 % 10

= 9



Double Hashing

Insert 4 into table



Double Hashing

index = ((4 % 10) + (0 * (7 - (4 % 7)))) % 10

= (4 + (0 * (7 - 4))) % 10

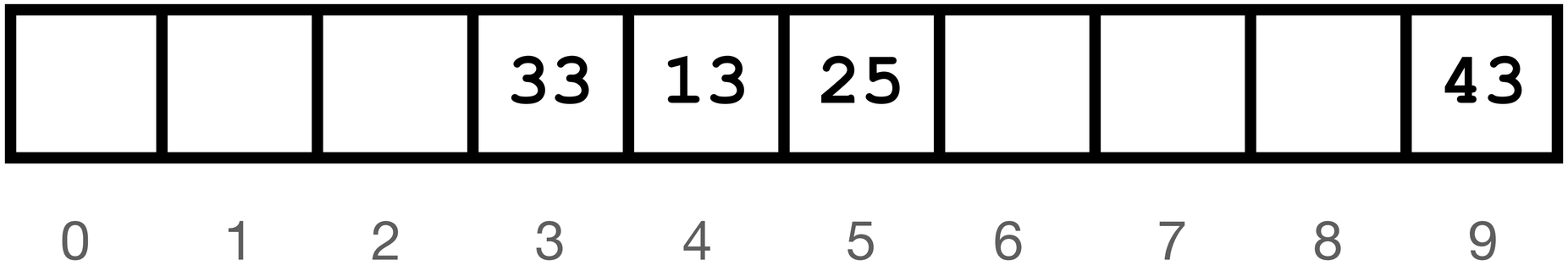
= (4 + (0 * 3)) % 10

= (4 + 0) % 10

= 4 % 10

= 4

Collision at 4, increment i by 1



Double Hashing

index = ((4 % 10) + (1 * (7 - (4 % 7)))) % 10

= (4 + (1 * (7 -4))) % 10

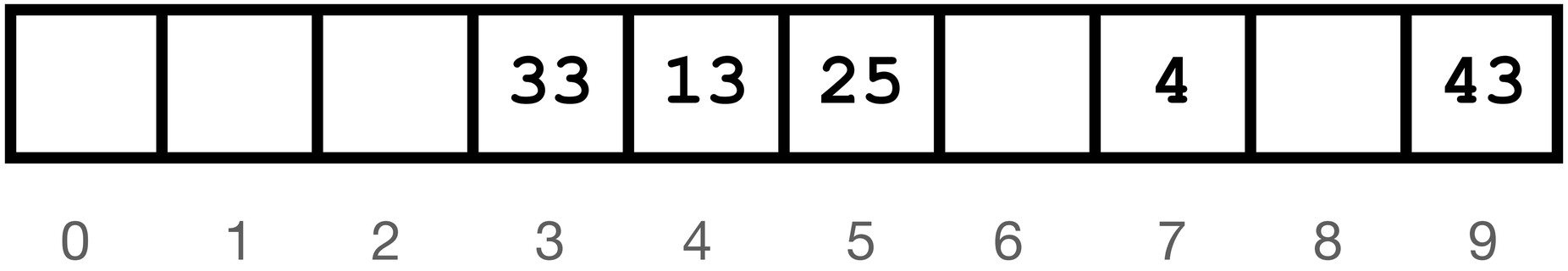
= (4 + (1 * 3)) % 10

= (4 + 3) % 10

= 7 % 10

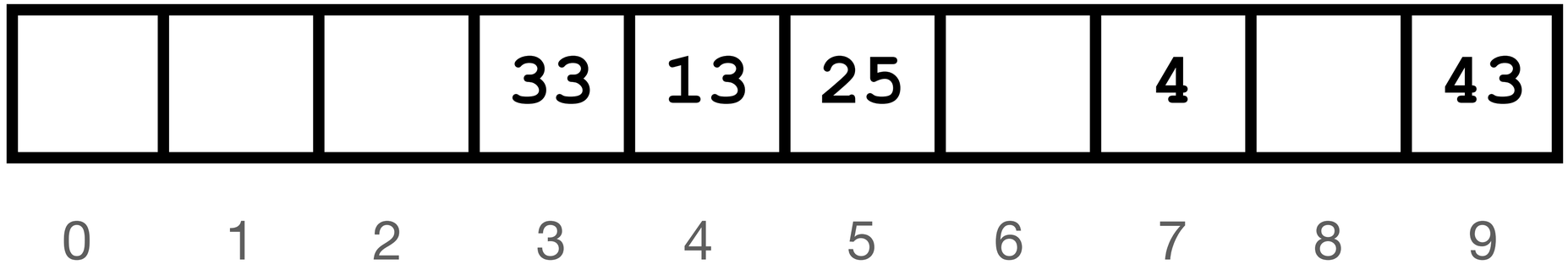
= 7

Collision at 4, increment i by 1



Double Hashing

Final Table



Separate Chaining

- Table where each index contains a linked list
- If collision happens, we add value to end of that index's linked list
- Never run out of space
- Hash function will be x % size
- Worst case time complexity will be O(n)

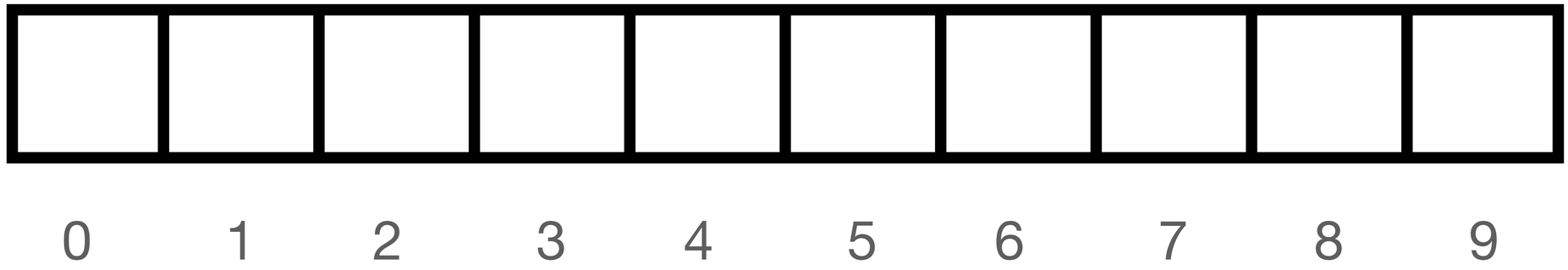
Separate Chaining

```
struct node{
  int value;
  node *next;
  node(int v) : value(v), next(nullptr) {}
};

void separateChaining(ndoe *table[], int size, int x){
  node *temp = new node(x);
  int index = x % size;
  if(table[index] == nullptr)
    table[index] = temp;
  else{
    node *cur = table[index];
    while(cur->next != nullptr)
      cur = cur->next;
    cur->next = temp;
  }
}
```

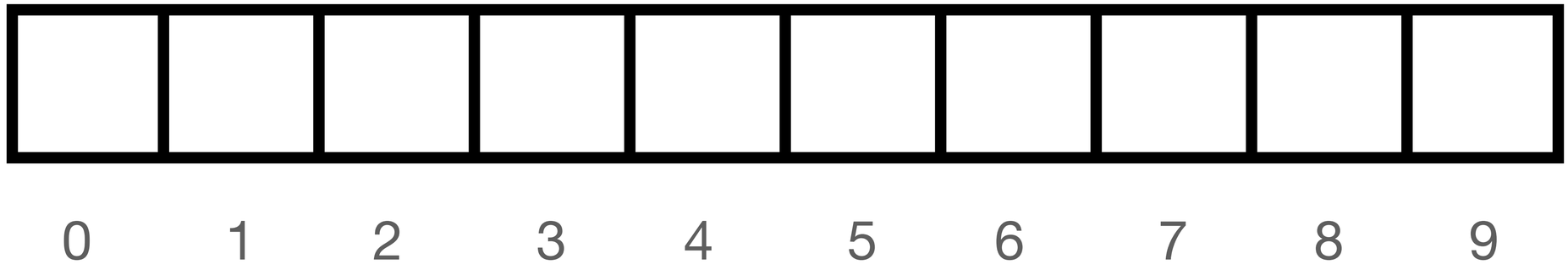
Separate Chaining

Let’s start with a table of size 10



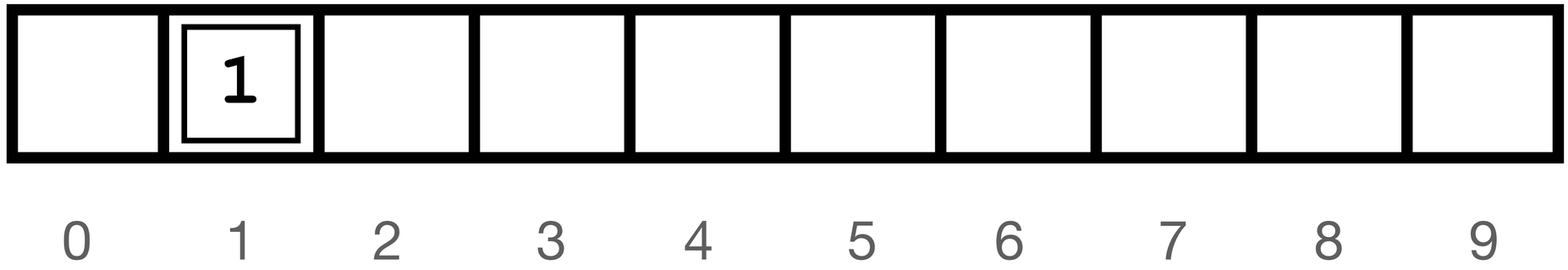
Separate Chaining

Insert 1 into table



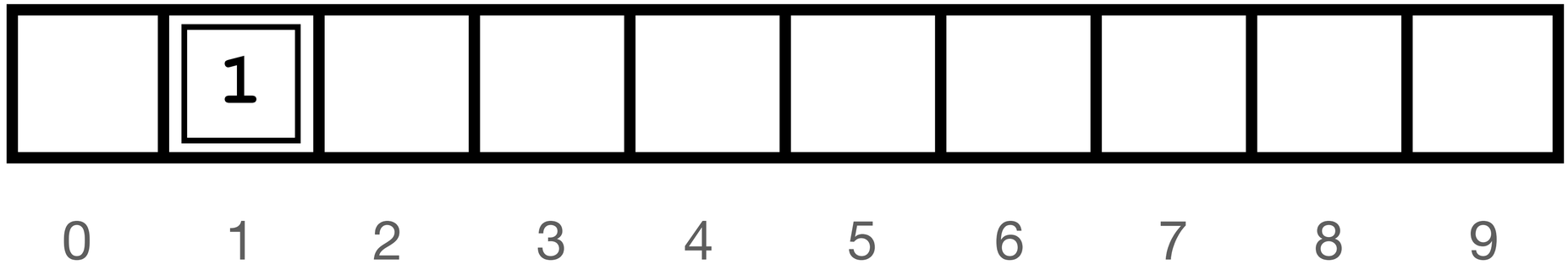
Separate Chaining

index = 1 % 10 = 1



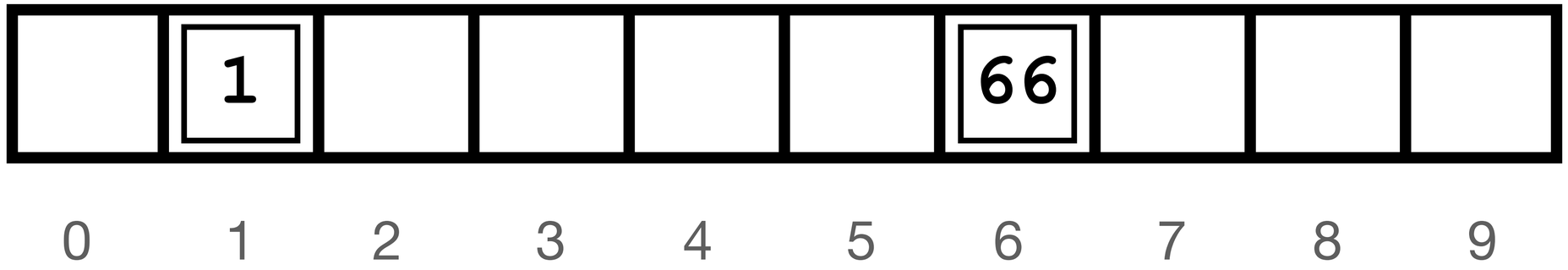
Separate Chaining

Insert 66 into table



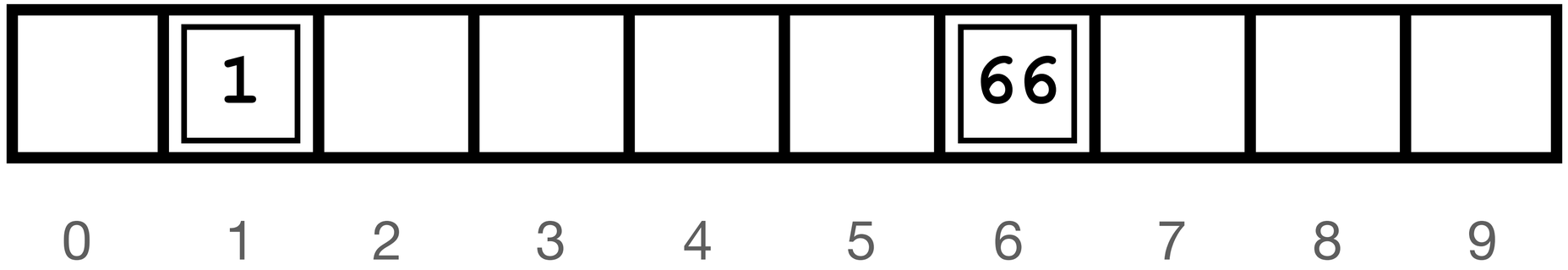
Separate Chaining

index = 66 % 10 = 6



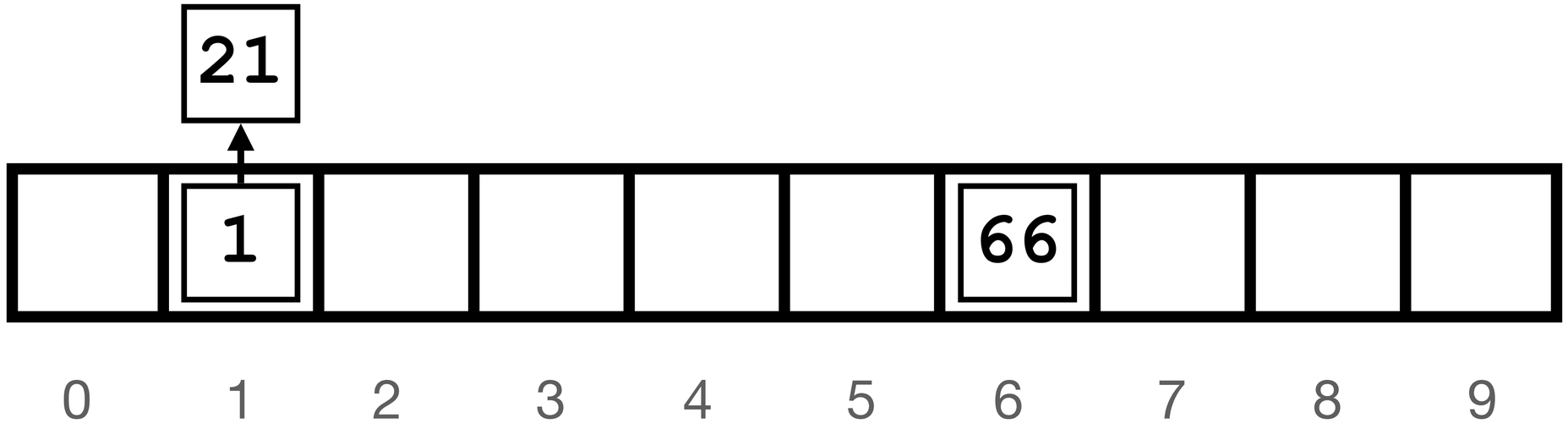
Separate Chaining

Insert 21 into table



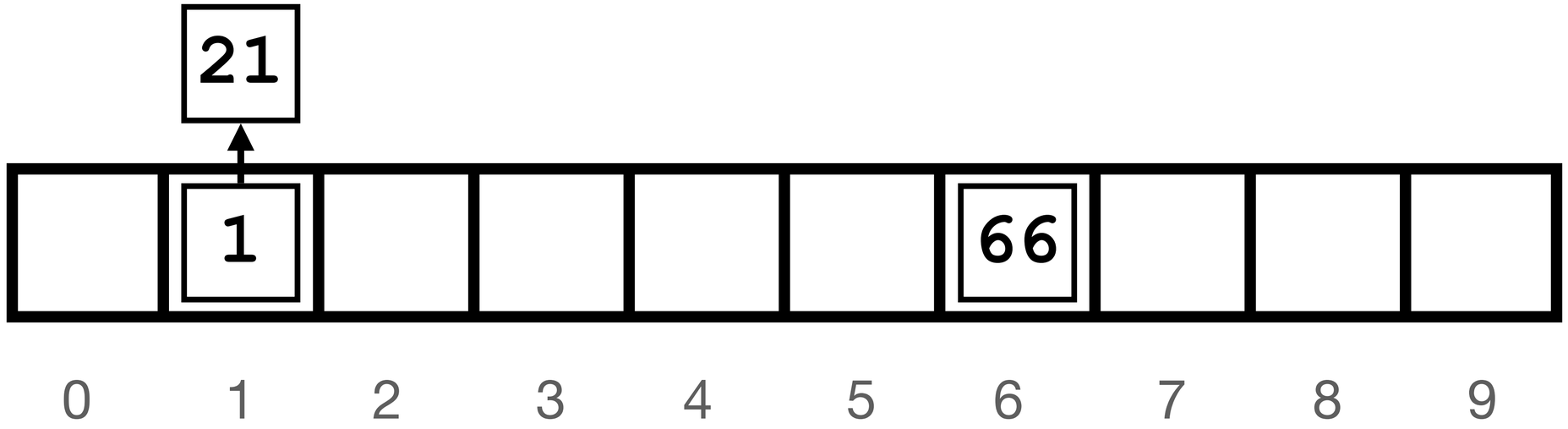
Separate Chaining

index = 21 % 10 = 1



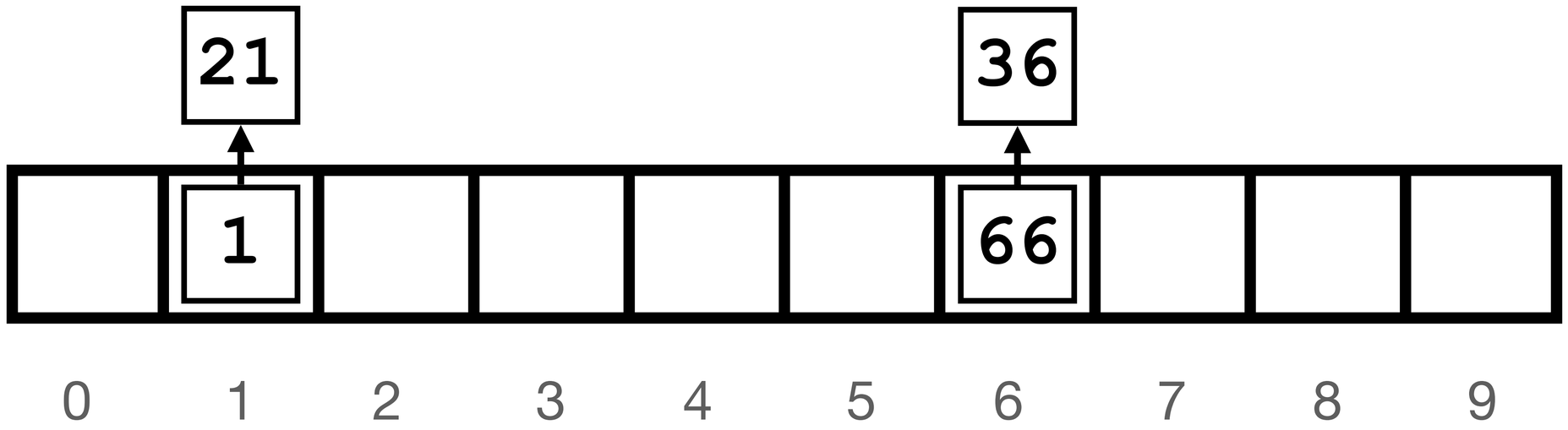
Separate Chaining

Insert 36 into table



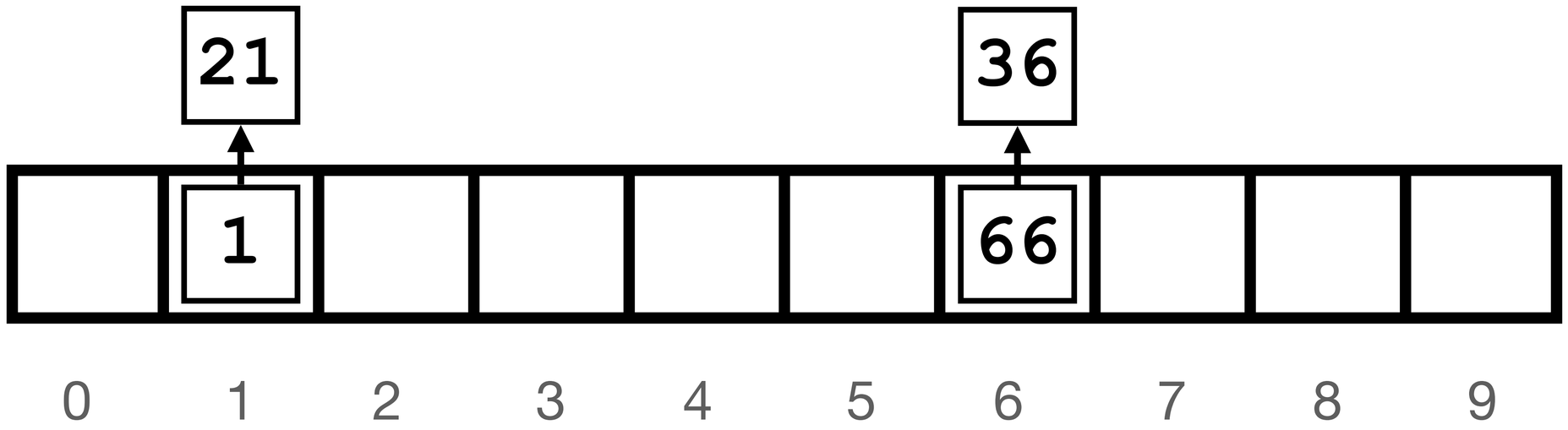
Separate Chaining

index = 36 % 10 = 6



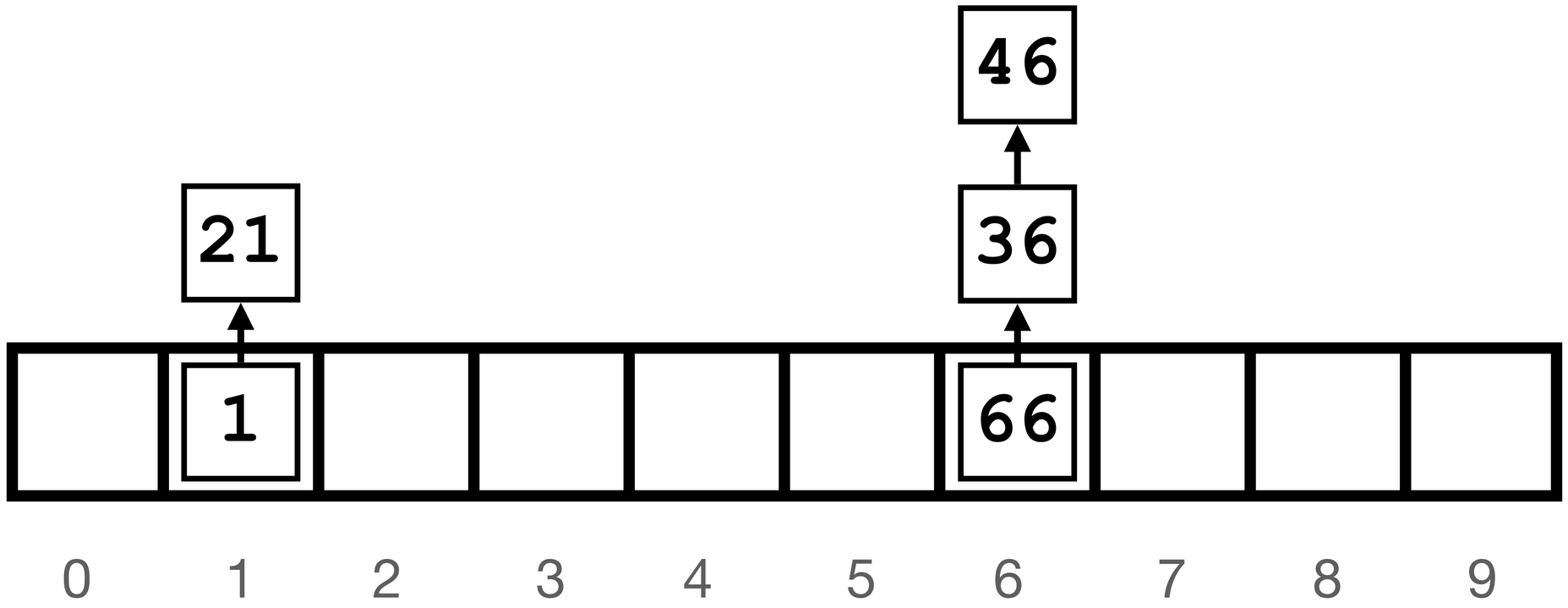
Separate Chaining

Insert 46 into table



Separate Chaining

index = 46 % 10 = 6



Separate Chaining

Final Table