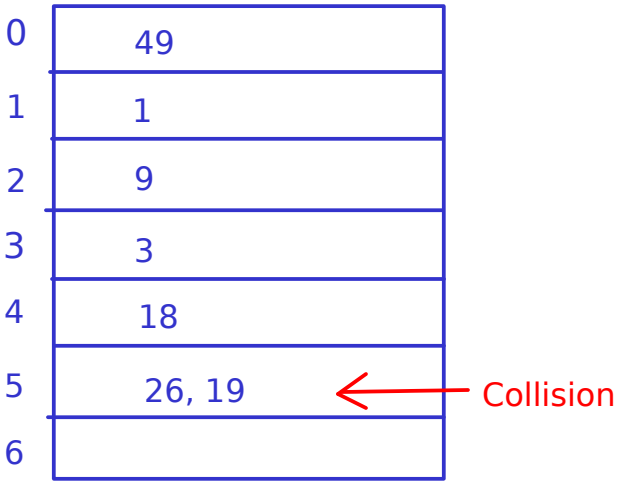


1, 3, 9, 26, 49, 18, 19

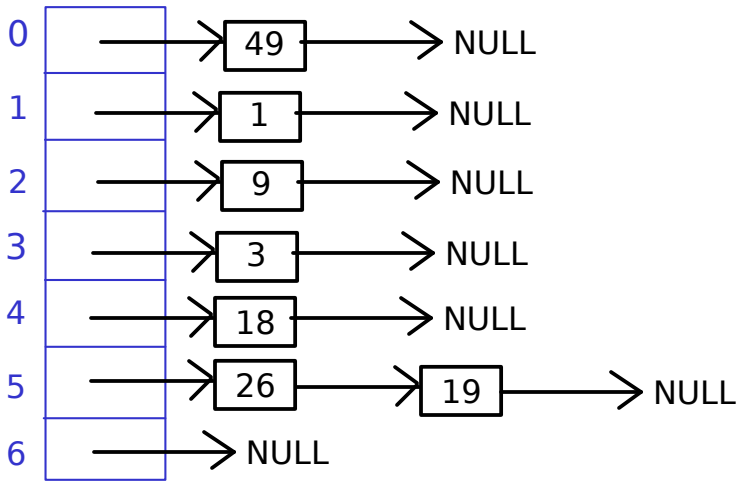
If k is a key, the define a function:

h(k) = k % 7

```
find(x) {
   pos = x % 7;
   if( T[pos] == x )
      return True;
   else
      return False;
}
```

```
#define EMPTY -1

insert(x) {
   pos = x % 7;
   if ( T[pos] = EMPTY )
      T[pos] = x;
   else
      // handle collision;
}
```



Open Addressing

Separate Chaining

```
find(x) {
   pos = x % 7;
   currNode = T[pos];
   while(currNode != NULL) {
      if (currNode -> data == x)
            return True;
      currNode = currNode->next;
   }
   return False;
}
```

```
insert(x) {
   pos = x % 7;
   struct Node* newNode = ...;
   newNode->data = x;
   newNode->next = NULL;
   if( T[pos] == NULL )
      T[pos] = newNode;
   else {
      currNode = T[pos];
      while(currNode->next != NULL)
         currNode = currNode->next;
      currNode->next = newNode;
   }
```

0	40
1	1
2	68
3	3
4	
5	26
6	19

1, 26, 19, 3, 40, 68

h(k) = k % 7

h(k) = (2k + 5) % 7

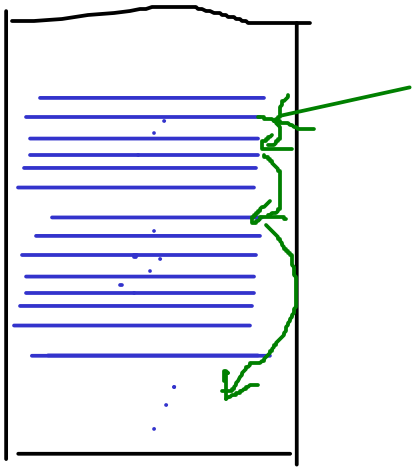
0	68
1	1
2	40
3	3
4	
5	26
6	19

Linear Probing: h(k, i) = (h(k) + i) % 7

Quadratic Probing: h(k, i) = (h(k) + i^2) % 7

```
insert(k) {
        i = 0;
        pos = k % 7;
        while( T[pos] != EMPTY ) {
                i = i + 1;
                pos = ( (k % 7) + i ) % 7;
                // if( pos == k % 7 )
                //   return "Error! Hash table is FULL!"
        }
        T[pos] = k;
}
```

```
find(k) {
   i = 0;
   pos = k % 7;
   while( T[pos] != k ) {
          i = i + 1;
          pos = ( (k % 7) + i ) % 7;
          if( T[pos] == EMPTY )
                return "Not present!"
    }
    return pos;
}
```



h(k) = k % 4

2, 4, 6, 8, 10

2 -> 2
4 -> 0
6 -> 2
8 -> 0
10 -> 2

h(k) = k % 5

2, 4, 6, 8, 10

2 -> 2
4 -> 4
6 -> 1
8 -> 3
10 -> 0

Load factor of hash table = (No. of elements inserted) / (Size of the table)