

O(1)

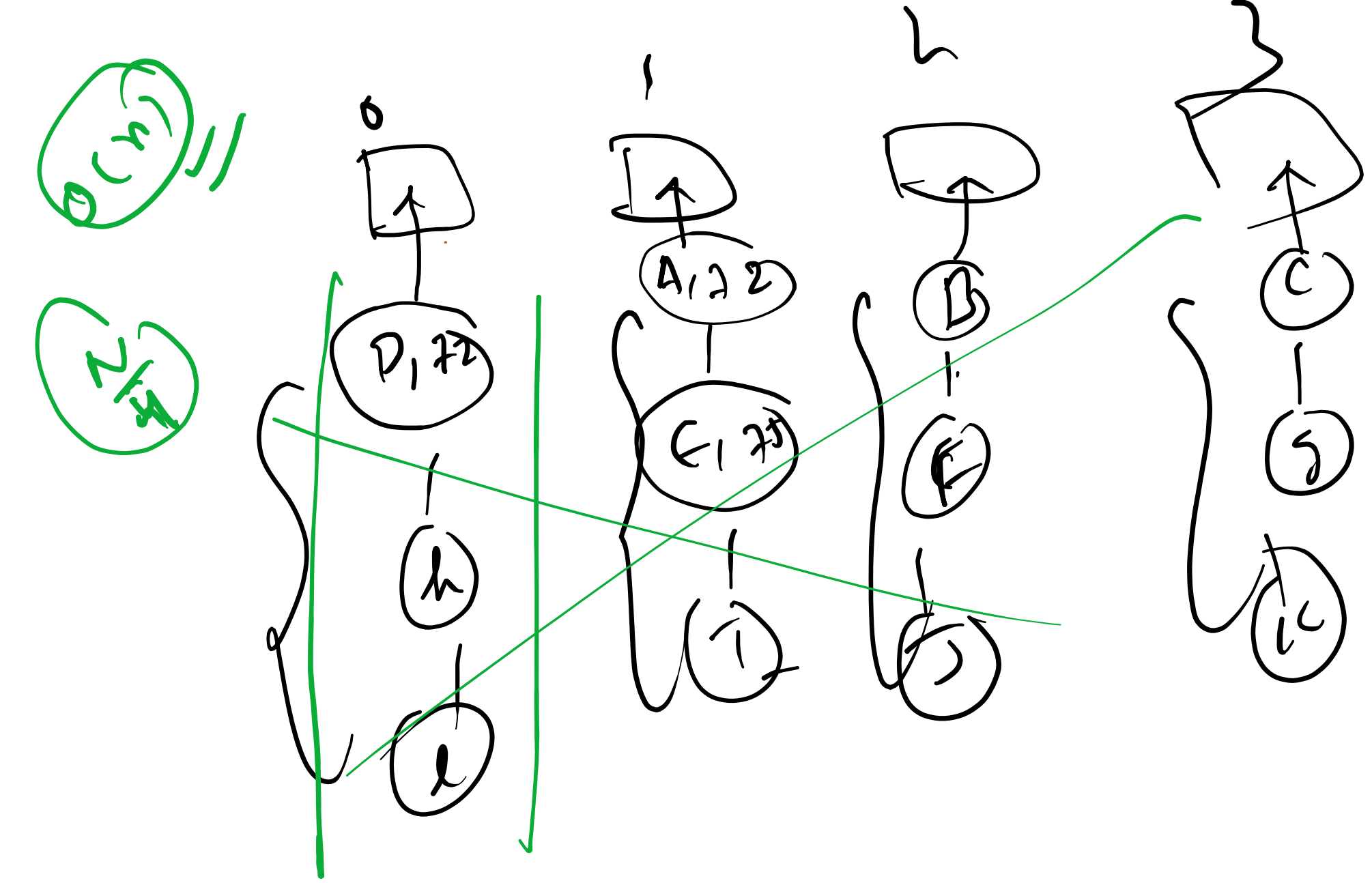
Chaining  
LinkedList

0  
D 199

1  
A 82  
E -79

2  
B -77

3 C 64  
A → 82  
B → 77  
C → 64  
D → 97  
E → 79  
hash function  
A src % 4  
65 × 4 = 1  
B → 66 × 4 = 2  
C → 67 × 4 = 3  
D → 68 × 4 = 0  
E → 69 × 4 = 1  
Cell First  
Box



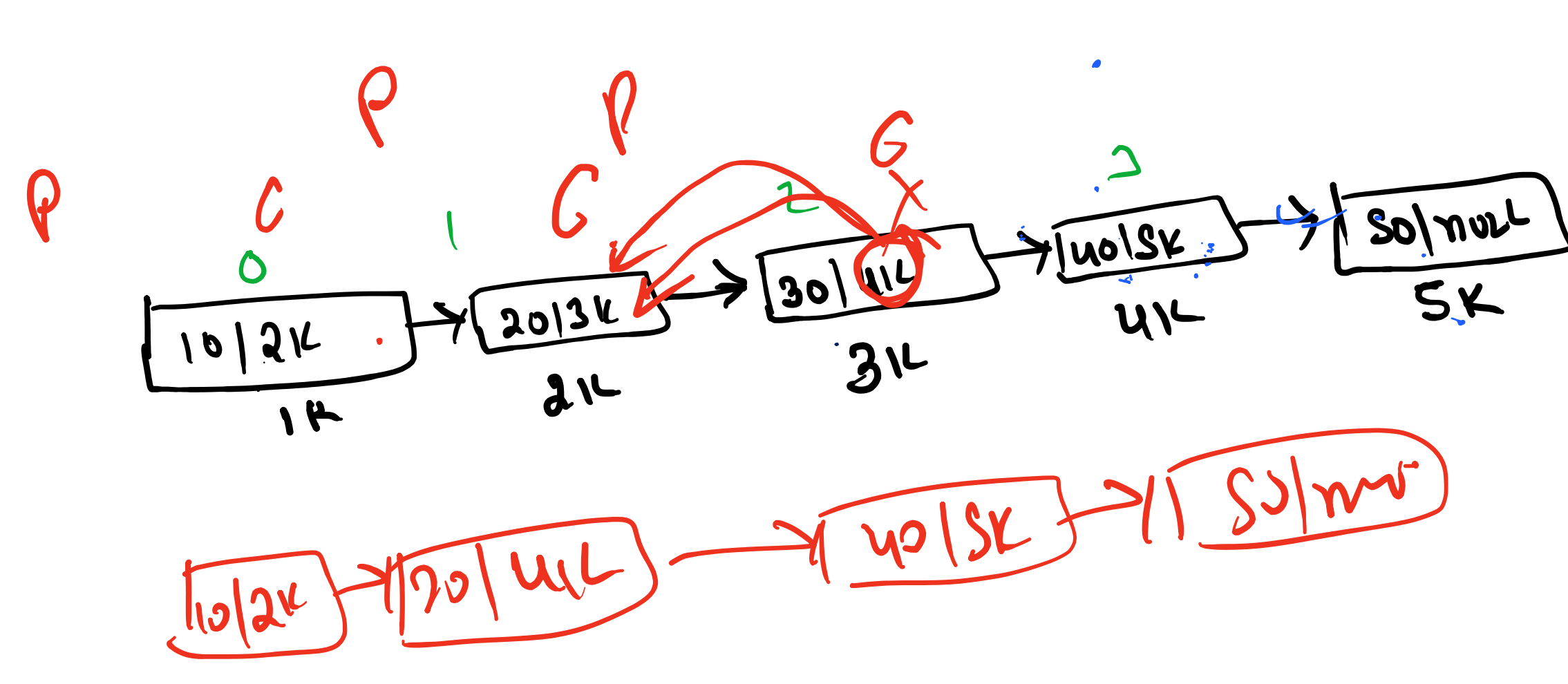
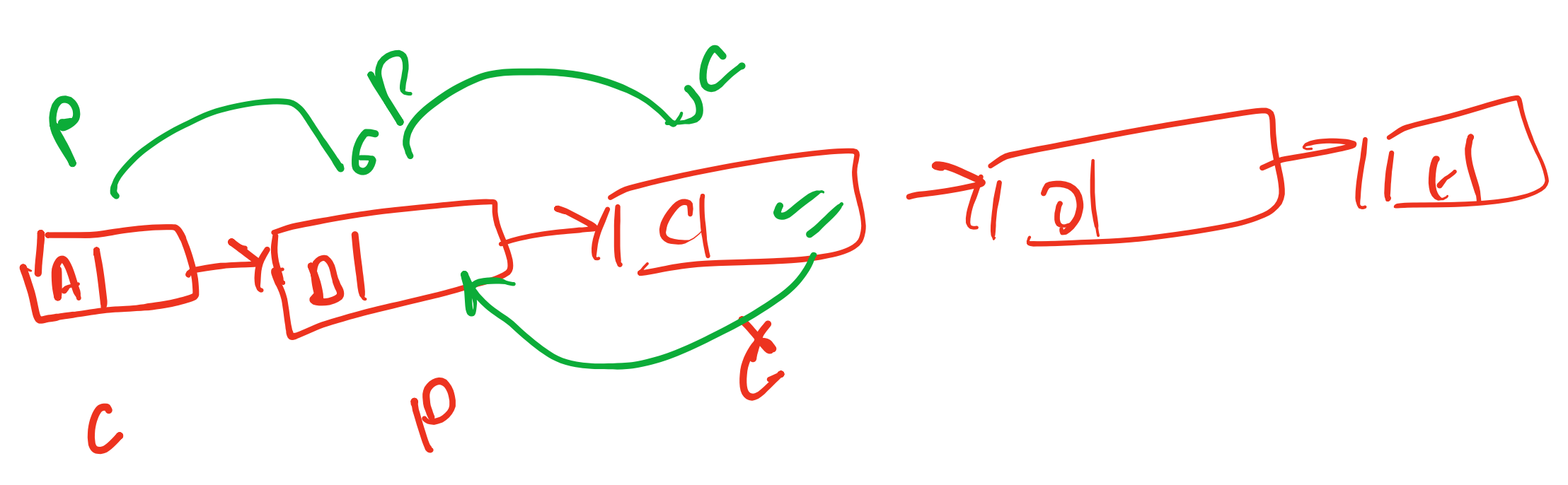
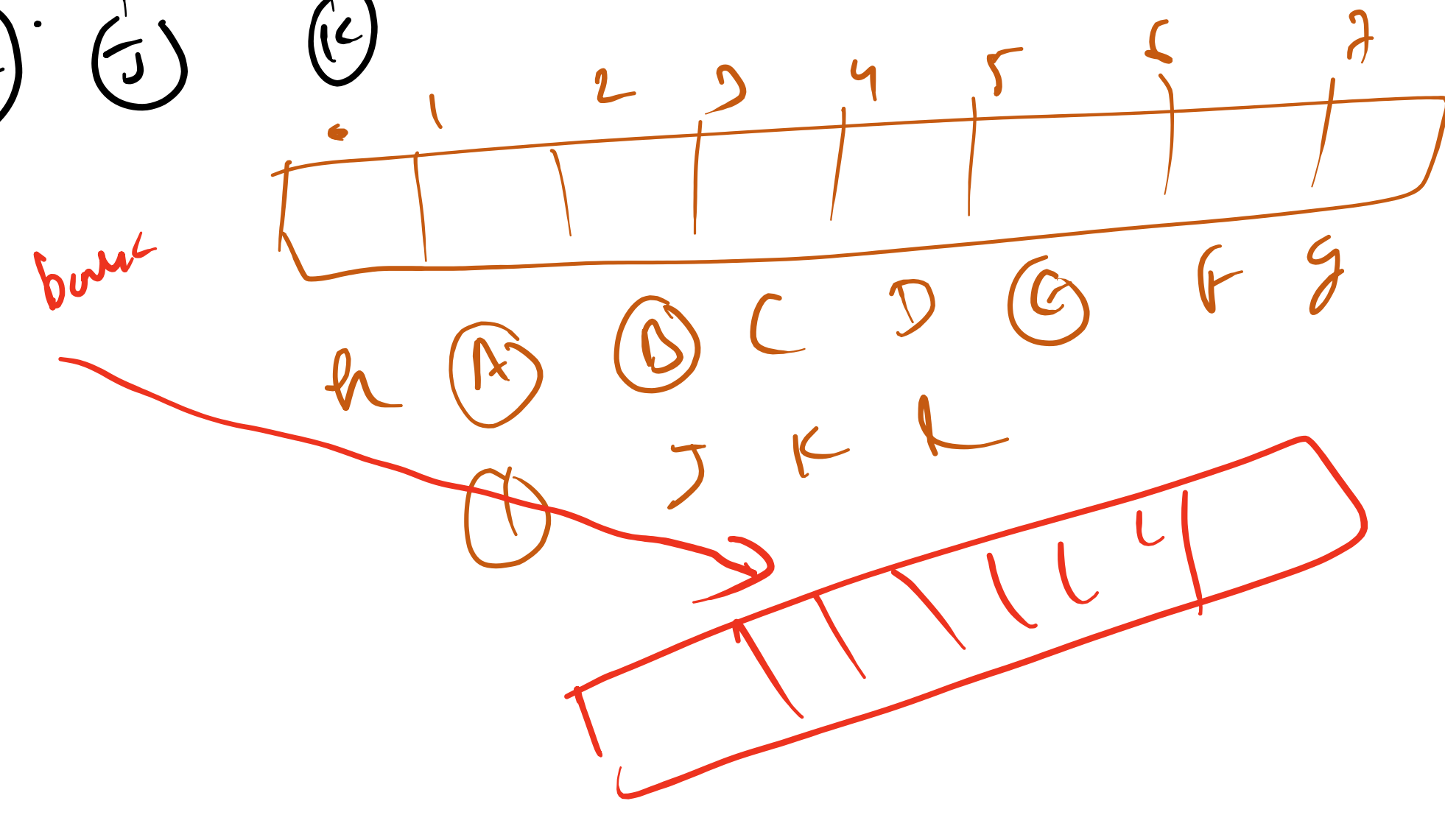
P × 4 = 0

Rehashing

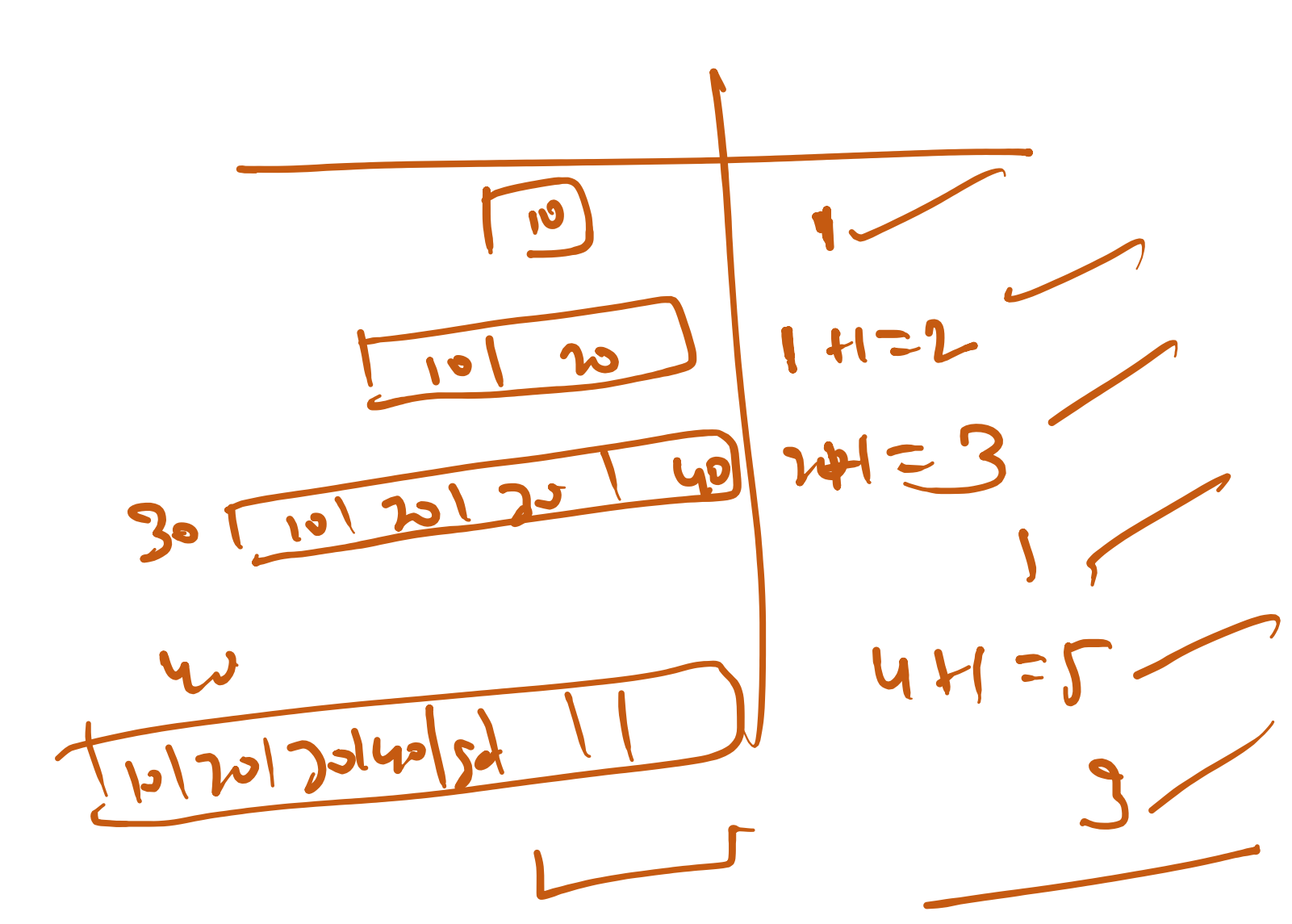
T.H.F = 2.0  
load factor = 12/4 = 3  
IF > THF

class Node {  
 key  
 value  
 next  
}

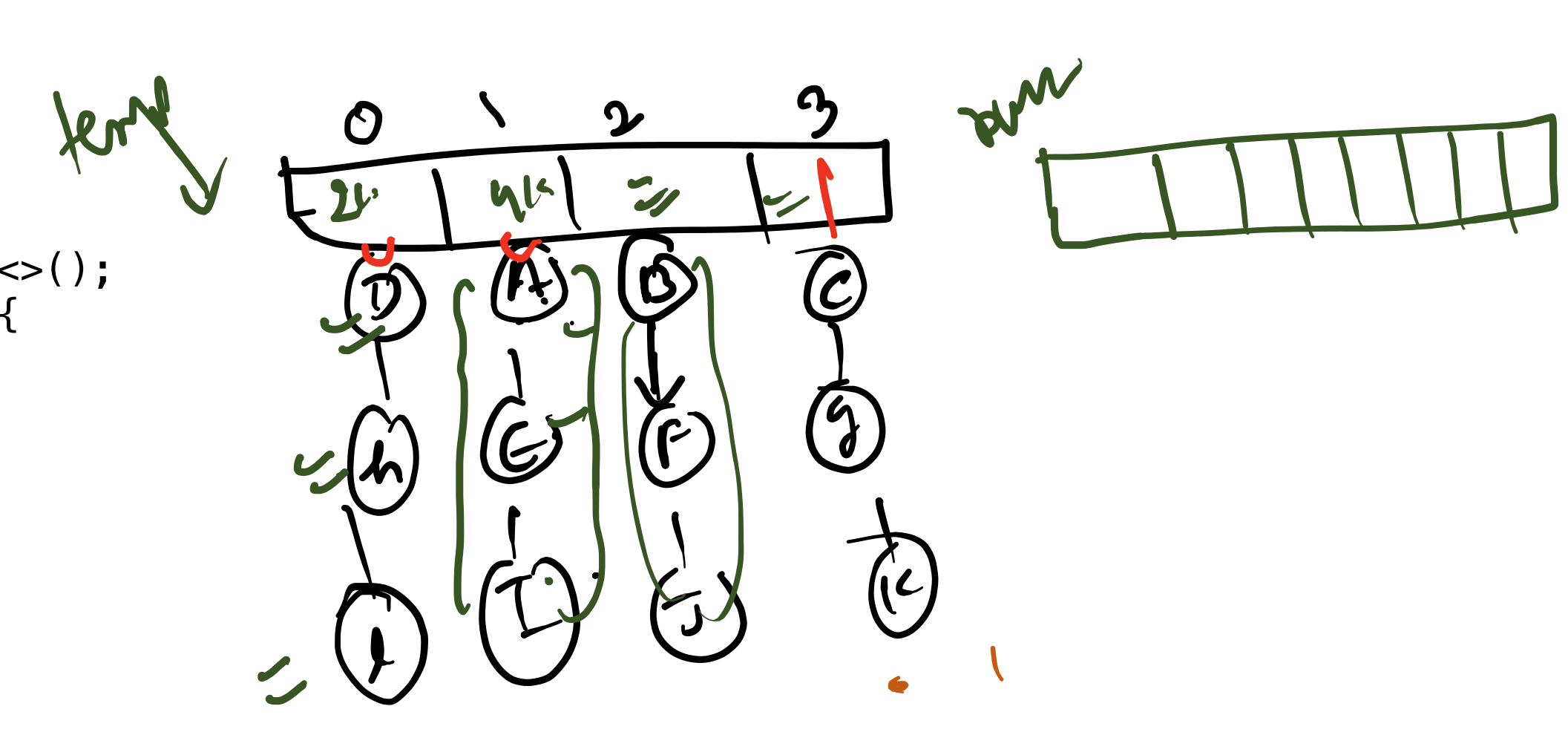
```
public void put(K key, V value) {  
    int idx = hashfun(key);  
    Node temp = bukt.get(idx);  
    while (temp != null) {  
        if (temp.key.equals(key)) {  
            temp.value = value;  
            return;  
        }  
        temp = temp.next;  
    }  
}
```



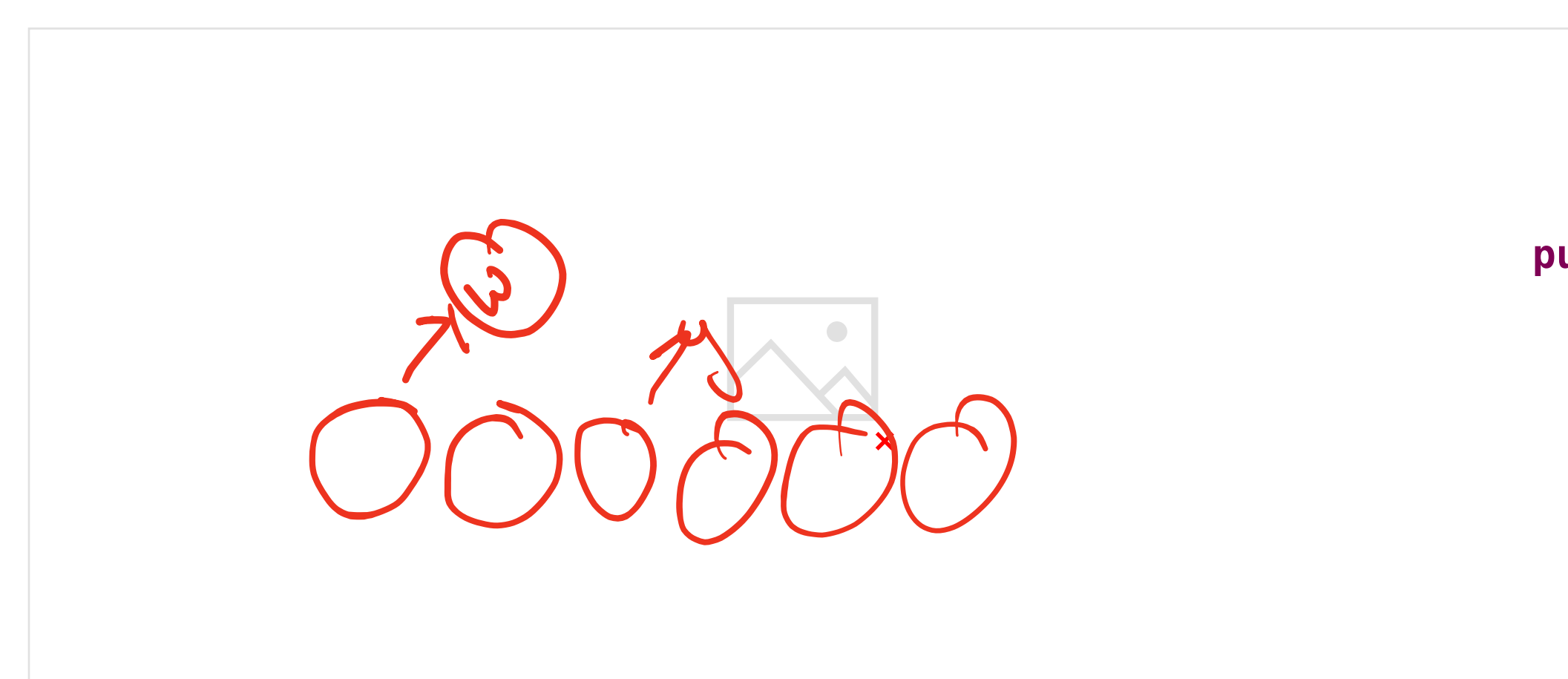
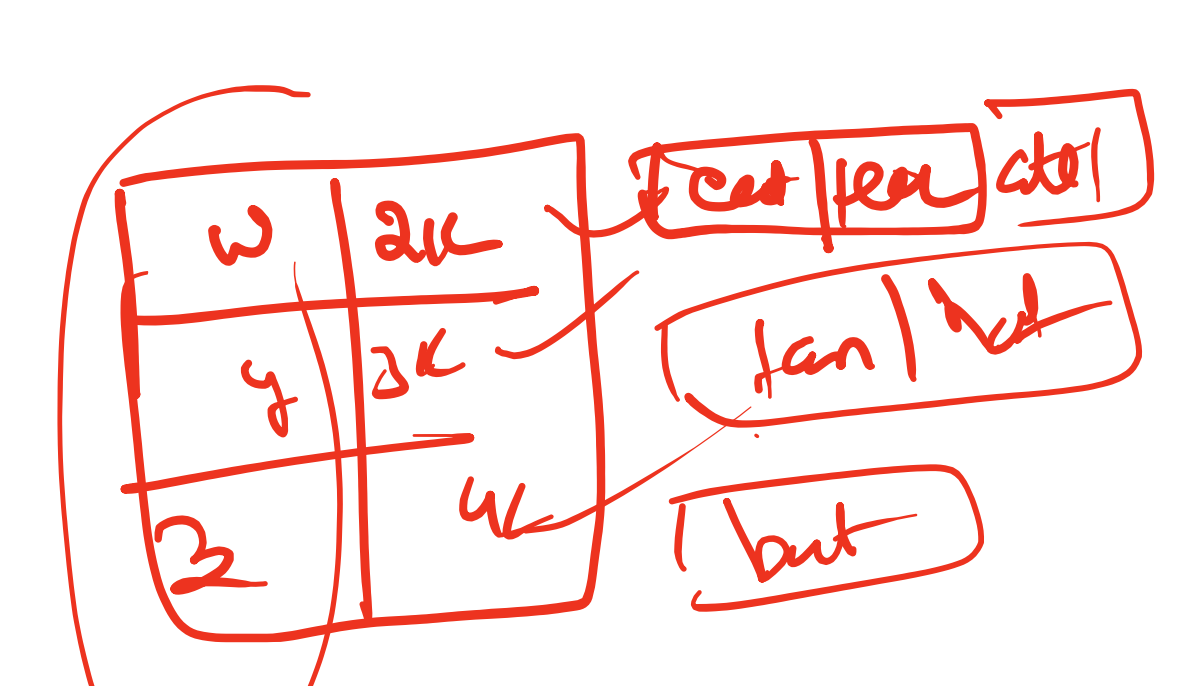
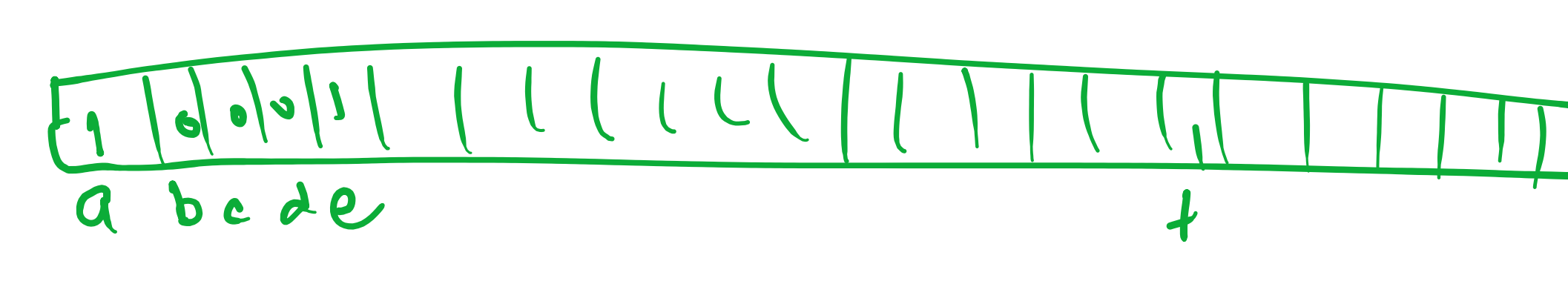
8 → 15  
1 2



```
private void rehashing() {  
    // TODO Auto-generated method stub  
    ArrayList<Node> new_bukt = new ArrayList<>();  
    for (int i = 0; i < 2*bukt.size(); i++) {  
        new_bukt.add(null);  
    }  
    ArrayList<Node> temp=bukt;  
    size=0;  
    bukt=new_bukt;  
    for(Node head: temp) {  
        while(head!=null) {  
            put(head.key, head.value);  
            head=head.next;  
        }  
    }  
}
```



1 0 0 0 1 0 0 0 1



```
public List<List<String>> groupAnagrams(String[] str) {  
    HashMap<String, List<String>> map = new HashMap<>();  
    List<List<String>> ll = new ArrayList<>();  
    for (int i = 0; i < str.length; i++) {  
        String key=Key_for_Anagrams(str[i]);  
        if(!map.containsKey(key)) {  
            map.put(key, new ArrayList<>());  
        }  
        map.get(key).add(str[i]);  
    }  
    return ll;  
}
```

