Open Addressing, Cryptographic Hashing

Jung, Hyokwon

Open Addressing

Open Addressing: 개방주소, 공개주소 방법

- Purpose: Collision Resolve Method
- Methods: Linear Probing, Quadratic Probing, Double Hashing
- What's difference?

→ **Efficiency**

Chain vs **Open Addressing**

#Slot	Item
0	[10, 20, 30]
1	-
2	-
3	-
4	44

Key	Value
0	10
1	20
2	30
3	-
4	44

Open Addressing

- no chaining; instead all items stored in table (see Fig. 1)

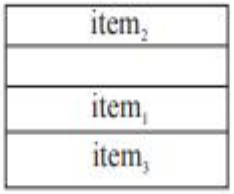


Figure 1: Open Addressing Table

- one item per slot $\implies m \geq n$
- hash function specifies *order* of slots to probe (try) for a key (for insert/search/delete), not just one slot; **in math. notation**:

  We want to design a function h, with the property that for all $k \in \mathcal{U}$:

$$h : \mathcal{U} \times \{0, 1, \ldots, m-1\} \rightarrow \{0, 1, \ldots, m-1\}$$

universe of keys — trial count — slot in table

1 Slot for 1 Item

 # Slot(m) > # Item(n)

Open Addressing

- **Function: Insert, Search, Remove**

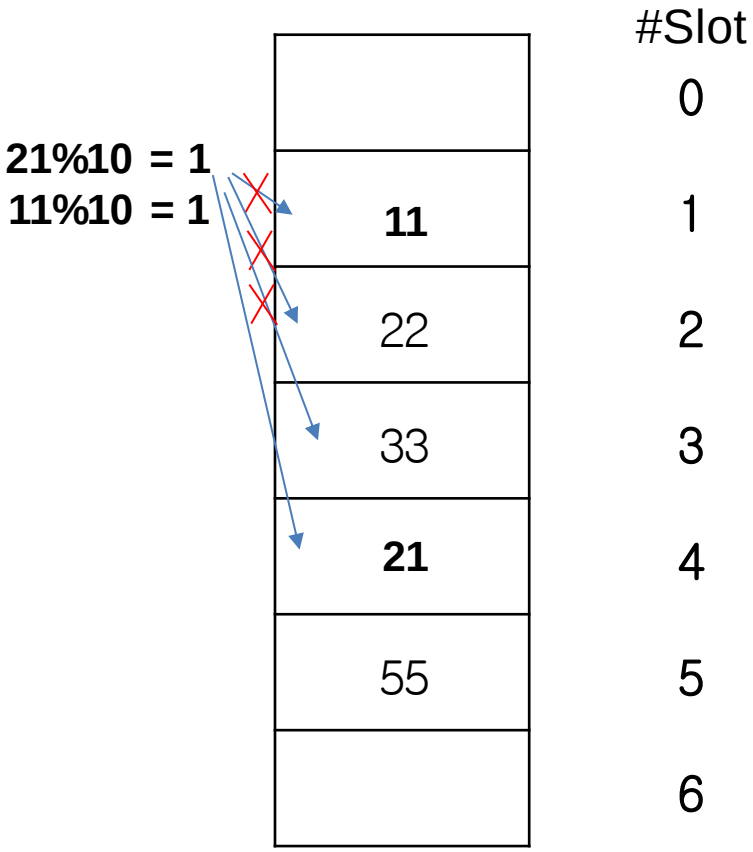
- Insert & Search

- **Hash Function:** Mod(10)

Rule: if (Slot != Occupied){
insert
}
else {
lookup next Slot
}

Insert(11)

Insert(21)



Open Addressing

- **Function: Insert, Search, Remove**

Insert(k,v) : Keep probing until an empty slot is found. Insert item into that slot.

```
for i in xrange(m):
    if T[h(k,i)] is None:        # empty slot
        T[h(k,i)] = (k,v)        # store item
        return
raise 'full'
```

Example: Insert $k = 496$

Search(k): As long as the slots you encounter by probing are occupied by keys $\neq k$, keep probing until you either encounter k or find an empty slot—return *success* or *failure* respectively.

```
for i in xrange(m):
    if   T[h(k,i)] is None:      # empty slot?
         return None             # end of "chain"
    elif T[h(k,i)][∅] == k:      # matching key
         return T[h(k,i)]        # return item
return None                      # exhausted table
```

Open Addressing

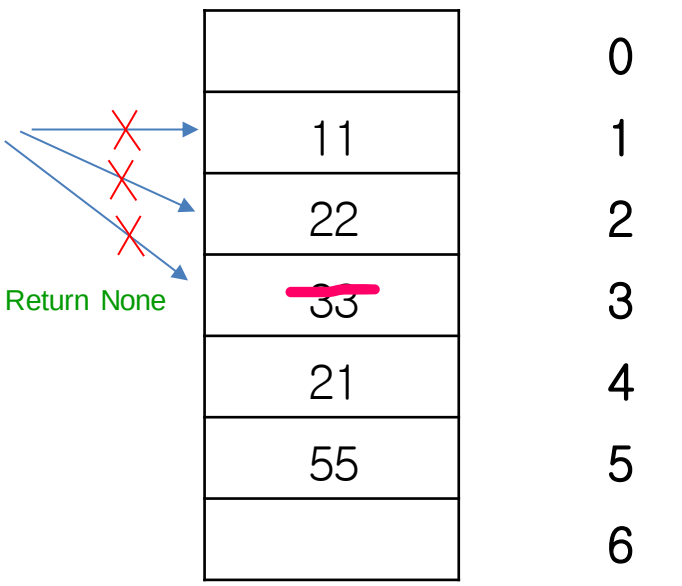
- **Function:** Insert, Search, Remove

Deleting Items?

- can't just find item and remove it from its slot (i.e. set $T[h(k,i)] = \text{None}$)
- *example*: delete(586) $\Longrightarrow$ search(496) fails
- replace item with special flag: "DeleteMe", which Insert treats as None but Search doesn't

Delete(33)

Search(21)



Open Addressing

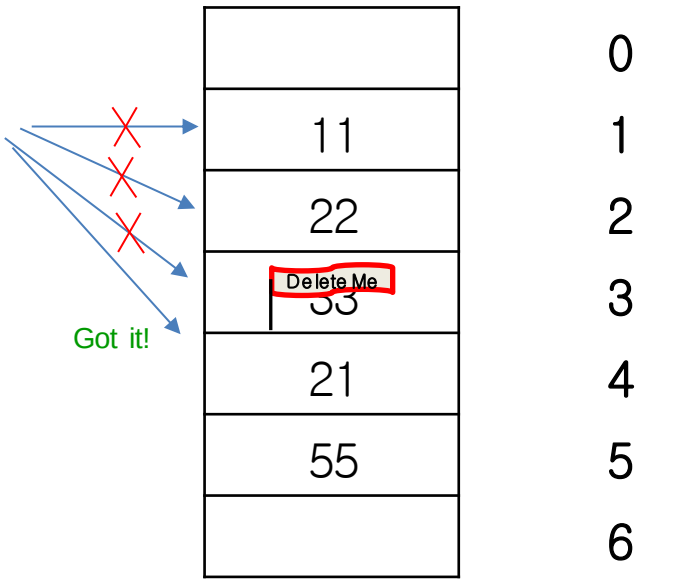
- **Function:** Insert, Search, Remove

Deleting Items?

- can't just find item and remove it from its slot (i.e. set $T[h(k,i)] = \text{None}$)
- *example*: delete(586) $\Longrightarrow$ search(496) fails
- replace item with special flag: "DeleteMe", which Insert treats as None but Search doesn't

Delete(33)

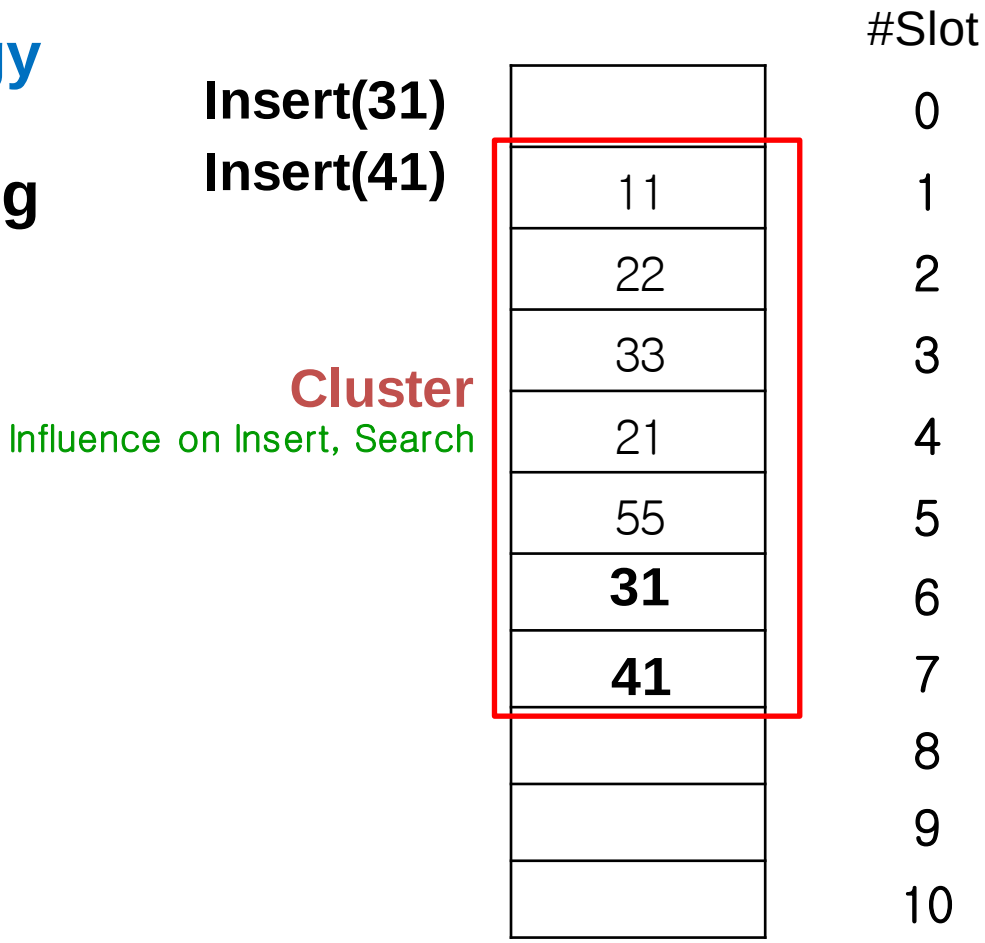
Search(21)



Open Addressing

- **Probing Strategy**

  - Linear Probing



Open Addressing

- **Probing Strategy**

$$h_1(k) + i \cdot h_2(k) \bmod m = h_1(k) + j \cdot h_2(k) \bmod m \Rightarrow m \text{ divides } (i - j)$$

$k \bmod (M) \Rightarrow 0 \sim m-1 \Rightarrow k$

$i \neq j$

$h_1(k) + i\, h_2(k) \bmod (M) = h_1(k) + j\, h_2(k) \bmod (M)$

↳ $h_1(k) + i \cdot h_2(k) \bmod (M) = h_1(k) + j \cdot h_2(k) \bmod (M)$

$i \cdot h_2(k) \bmod (M) = j \cdot h_2(k) \bmod (M)$

$\Rightarrow (i - j) \cdot h_2(k) \bmod (M) = 0$

$\Rightarrow (i - j) \cdot h_2(k) = m \times R$

$\Rightarrow i - j$ 은 M의 배수

Open Addressing

Summary

- **Complexity**

Load Factor α = n/m $\rightarrow (0 < \alpha < 1)$

(item)/(Slot)

Successful Search ← Probing frequency

Cost (insert) <= 1/(1−α)

α, smaller better(<0.5), if alpha is big, think of use Chaning

Strength & Weakness

Open Addressing vs. Chaining

Open Addressing: better cache performance (better memory usage, no pointers needed)

Chaining: less sensitive to hash functions (OA requires extra care to avoid clustering) and the load factor α (OA degrades past 70% or so and in any event cannot support values larger than 1)

Cryptographic Hasing

- **Cryptographic Hashng:**

  - Example: Password Storage

  https://www.youtube.com/watch?v=YiRPt4vrSSw

Question?