Algorithms & Data Structures I: Hash Table Open Addressing

Mikhail Anukhin

Today's Topics

- Hash Tables recall
- Motivation
- Open Addressing Hashing
- Insert
- Search
- Clustering Problem
- Probing Strategies
- Analysis

Dictionary (Map in C++)

Dictionary is an:

- Abstract Data Type (ADT) maintaining items, where each item is a pair<key, value>

Examples:

1. *Phonebook*. Keys are names, and their corresponding items are phone numbers
2. *Real dictionary*. Keys are english words, and their corresponding items are dictionary-entries
3.

item = <key, value>

HT[key]

Motivation

Dictionaries:

- Built into most modern programming languages (Python, C++, Ruby, Go,JavaScript, Java, . . .)
- Very powerful concept
- Use in web development fundamentals such as DNS system
- Use in cryptography

Hashing:

- Password storage
- File modification detector
- Digital signatures

Operations to support

Insert(item): Add item to the data structure

key

Delete(~~item~~): Delete item from the data structure

key

Search(~~item~~): return item with key if exists

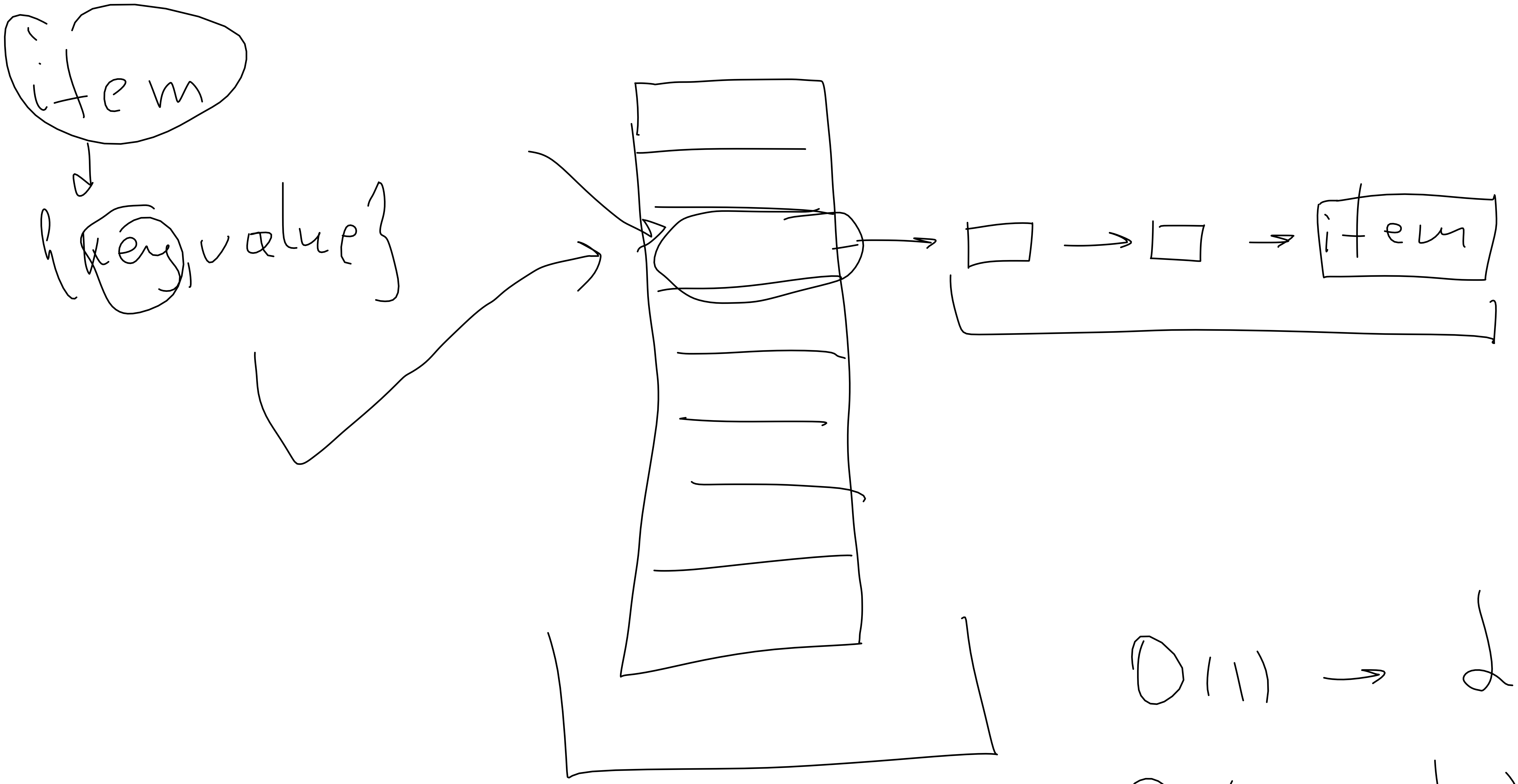
<u>Assumption: items have unique keys</u>

Hashing. Hash functions

Definitions:

- **Universe U**. A set of all possible keys
- **Hash function.** A function that map keys to one of m possible values. m is a capacity of hash table
- **Collision**
  Keys a,b such that: a != b and h(a) == h(b)

$H(\square) \rightarrow \underline{\qquad} \leftarrow \text{hash}$



$$O(1) \to \alpha < \frac{3}{4}$$

$$\underline{O(1+\alpha)}$$

Open Addressing

Concept:

- **Direct access table without chaining** . All items are stored in table
- One item per slot

How to deal with collisions?

1. Hash function specifies order of slots to find a place for a key, not just one slot

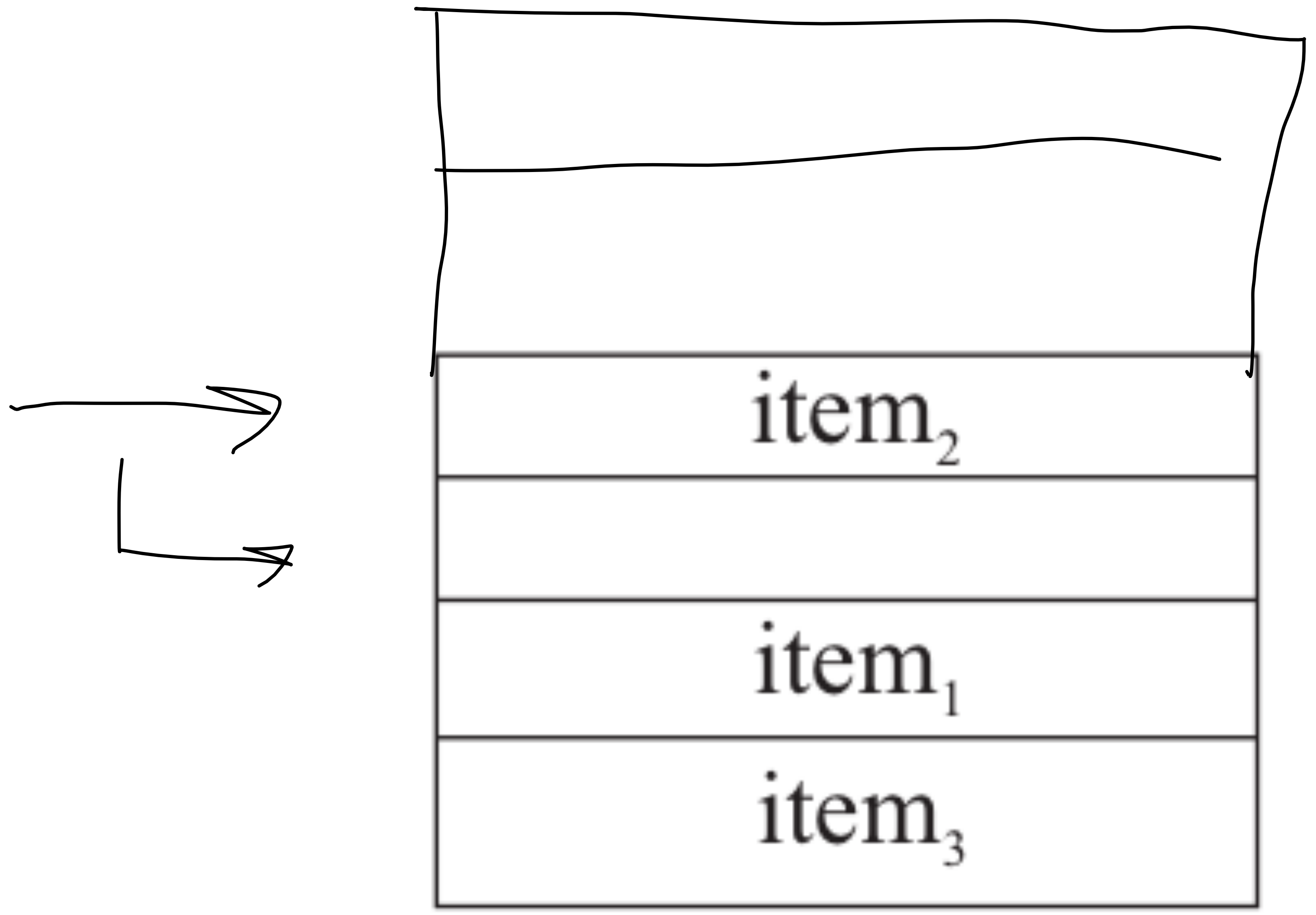


Figure 1: Open Addressing Table

Open Addressing Hashing in Details

$$h : \underbrace{U}_{\text{Universe of keys}} \times \underbrace{\{0, \dots, m-1\}}_{\text{trial count}} \to \underbrace{\{0, 1, \dots, m-1\}}_{\text{slot in table}}$$

$$k \qquad \{\langle \underbrace{\underset{1}{h(k,0)},\ \underset{3}{h(k,1)},\ \underset{4}{h(k,2)},\ \dots,\ \underset{0}{h(k,m-1)}}_{m} \rangle\}$$

$m := 5$

Open Addressing Hashing

Hash function

- h: U x {0, …, m - 1} -> {0, …, m-1}
  function from 2 arguments: key and number of attempt
  to find a place for a key
- { h(k, 0), h(k, 1), . . . , h(k, m – 1) } must be a permutation
  of 0, 1, … , m-1
  Means If I keep trying h(k, i) for increasing i, I will
  eventually hit all slots of the table.

Insert

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

key

Insert(496, "a")

h(496, 0) = 4

h(496, 1) = 1

h(496, 2) = 3

m

1	586, "a"
2	133, "a"
3	496, "a"
4	157, "a"
5	
6	
7	

Insert Pseudocode

```
void Insert(key, value):
    for i = 0 to m - 1:
        if (Table[h(key, i)] == None):
            Table[h(key, i)] = value
    Rehash() # means there is no an empty slot for insert
```

or == 'DeleteMe'

break

Search

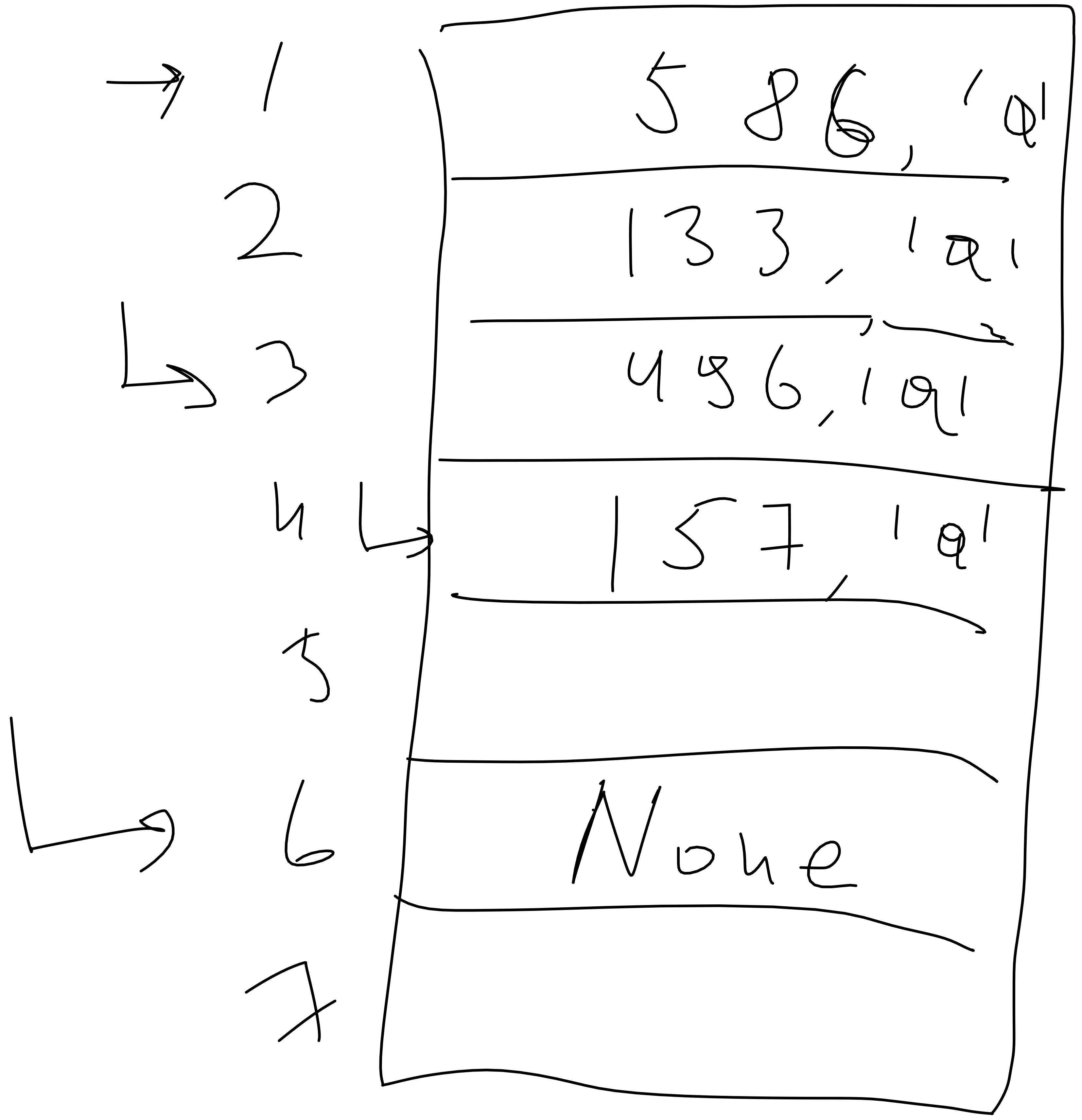
Search(key): Keep probing until you found a slot with the key or find an empty slot—return success or failure respectively.

Search(199)

$h(199, 0) = 6$

$h(199, 1) =$

$h(199, 2) =$



Search Pseudocode

```
bool Search(key):
    for i = 0 to m - 1:
        if (Table[h(key, i)] == None):
            return False
        else if (Table[h(key, i)] == Key):
            return Table[h(key, i)]
    return False
```

else if (Table[h(k, i)] == 'DeleteMe':
continue

Delete

Delete(586)

Delete(496)

$h(496, 0) = 4$

$h(496, 1) = 1$

$h(496, 2) = 3$

1	"DeleteMe"
2	133 'a'
3	DeleteMe
4	157 'a'
5	
6	
7	

Mikhail Anukhin

Delete

- Can’t just find item and remove it from its slot
- Replace item with special flag: “DeleteMe”, which Insert treats as None but Search doesn’t

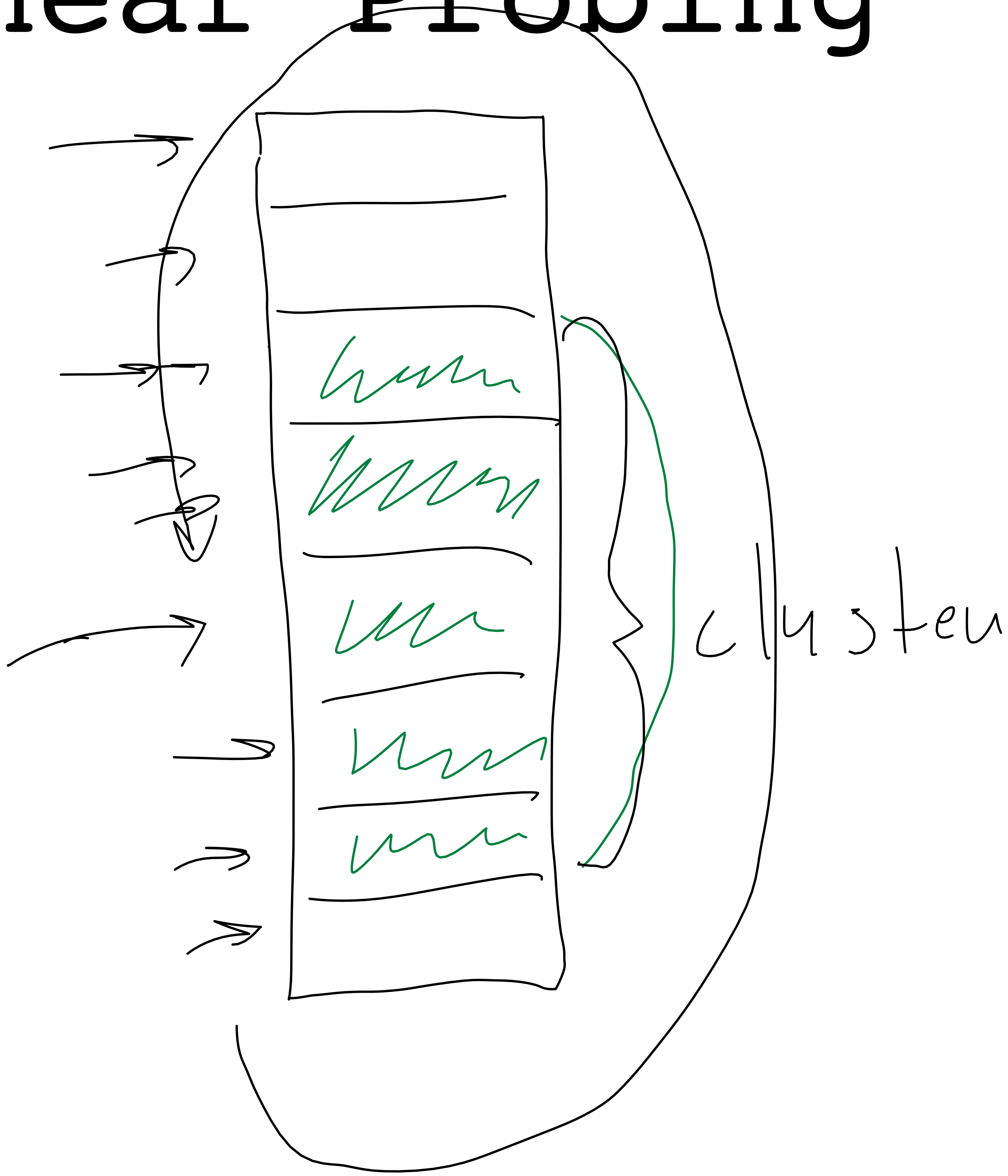
Probing Strategies. Linear Probing

Linear Probing:

- h(k, i) = (*h'(k)* + i) mod m
  where *h'(k)* is ordinary hash function

Problems:

1. *Clustering—cluster: consecutive group of occupied slots as clusters become longer, it gets more likely to grow further*
2. *Can be shown that for 0.01 <* **α** *< 0.99 say, clusters of size* **Θ***(log n)*
3.



Probing Strategies. Double Hashing

$\alpha < \frac{3}{4}$

Double Hashing:

- $h(k, i) = (h_1(k) + i * h_2(k)) \bmod m$
  where $h_1(k)$ and $h_2(k)$ are ordinary hash functions
- Hit all slots (permutation) if $h_2(k)$ is relatively prime to m for all k
- e.g. $\underbrace{m = 2^r}$, make $h_2(k)$ always odd

2^r

Simple uniform Hashing: Assumption

Analysis We use open addressing to insert n items into table of size m. Under the uniform hashing assumption the next operation has expected cost of less or equal to $1 / (1 - \boldsymbol{\alpha})$, where is $\boldsymbol{\alpha} = n/m$ (< 1)

Example: $\boldsymbol{\alpha} = 90\% \Longrightarrow$ 10 expected probes

$$\alpha = \frac{n}{m} < 0.75$$

$$\le \frac{1}{1-\alpha} = \frac{1}{1-0.9} = 10$$

$$\alpha \in [0,1]$$

$$\alpha < 0.5 \Rightarrow$$

$$O\left(\frac{1}{1-\alpha}\right)$$

Open Addressing vs. Chaining

Open Addressing: no pointers needed, better memory usage

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

Your questions!