Algorithms & Data Structures I
CSC 225

Ali Mashreghi

Fall 2018

Department of Computer Science, University of Victoria

Collision resolution - open addressing

- In open addressing, the idea is that if some slot is taken, we simply insert the element into the **next empty slot**.

- The process of searching for the next empty slot is called **probing**.

Collision resolution - open addressing

- The simplest method of probing is **linear probing**:

- If the hash value is $h(k) = j$, we first look at index j,
  then $(j + 1)\%m$
  then $(j + 2)\%m$
  then $(j + 3)\%m$, and so on

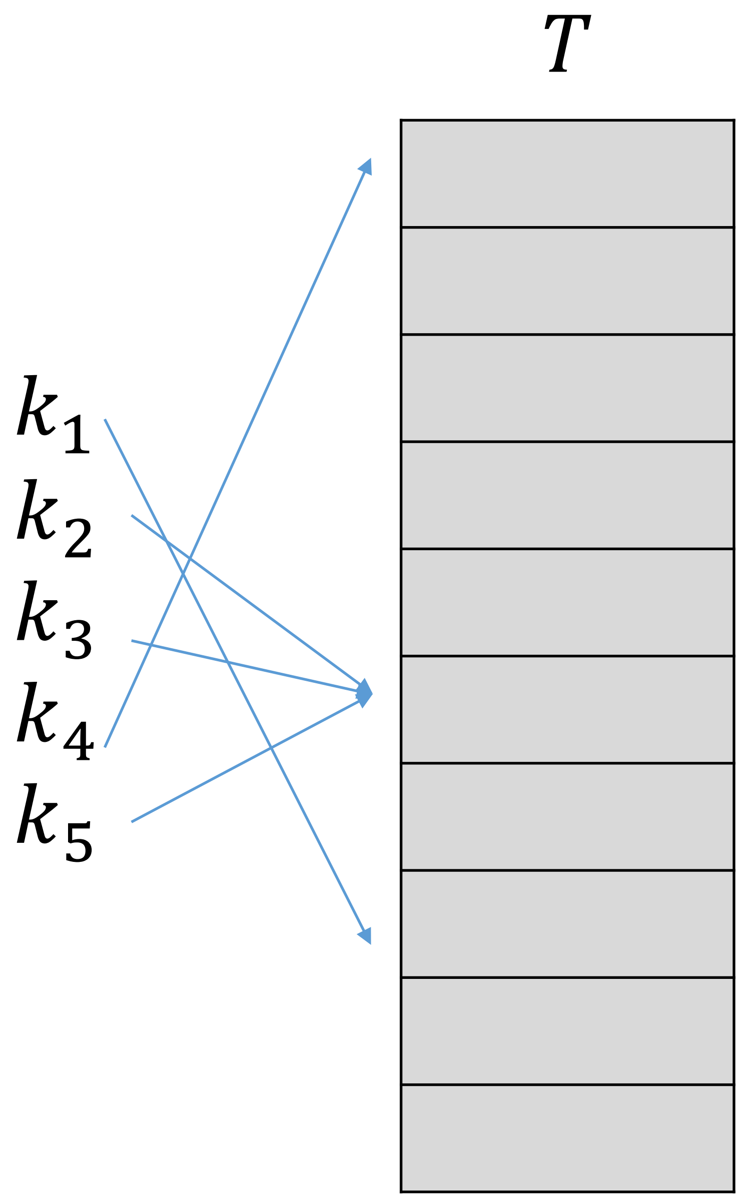
- As soon as there is an empty slot we insert the element.

Linear probing

- Say we are inserting five elements whose keys are $k_1, k_2, \ldots, k_5$.

- Assume that:
  $$h(k_2) = h(k_3) = h(k_5)$$

- e_i is the element with key k_i.

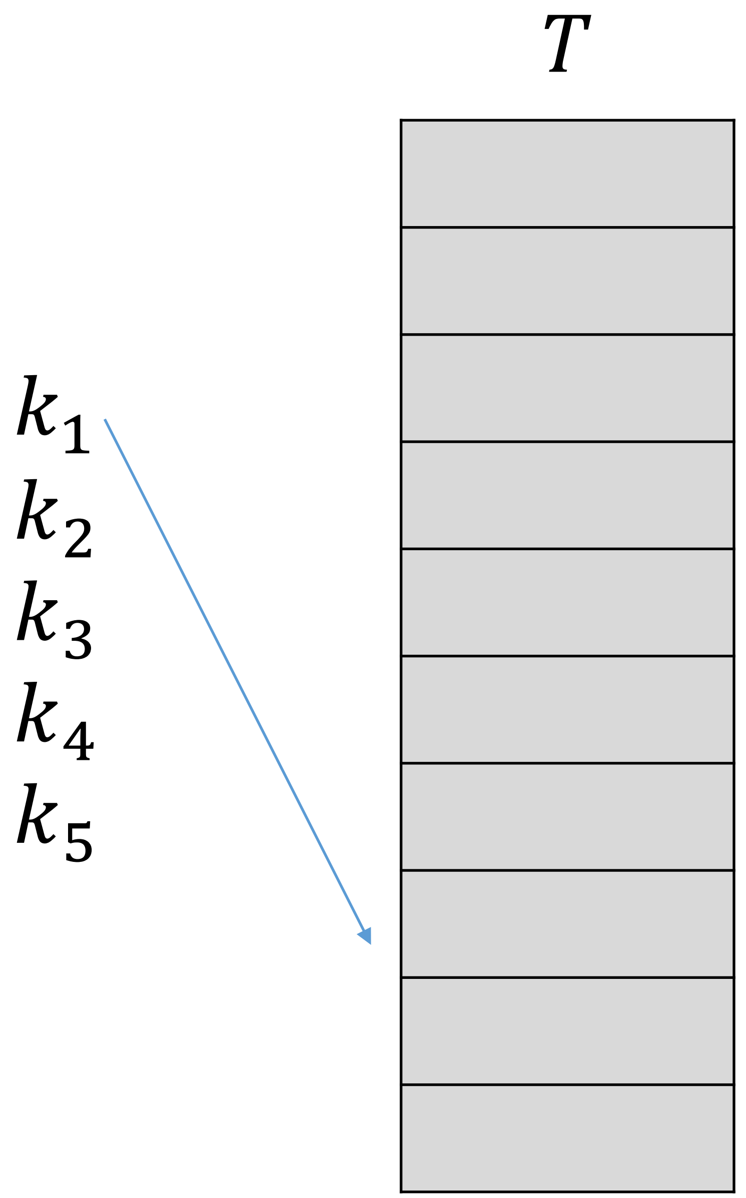


Linear probing

- Say we are inserting five elements whose keys are $k_1, k_2, \dots, k_5$.

- Assume that:
  $$h(k_2) = h(k_3) = h(k_5)$$

- e_i is the element with key k_i.

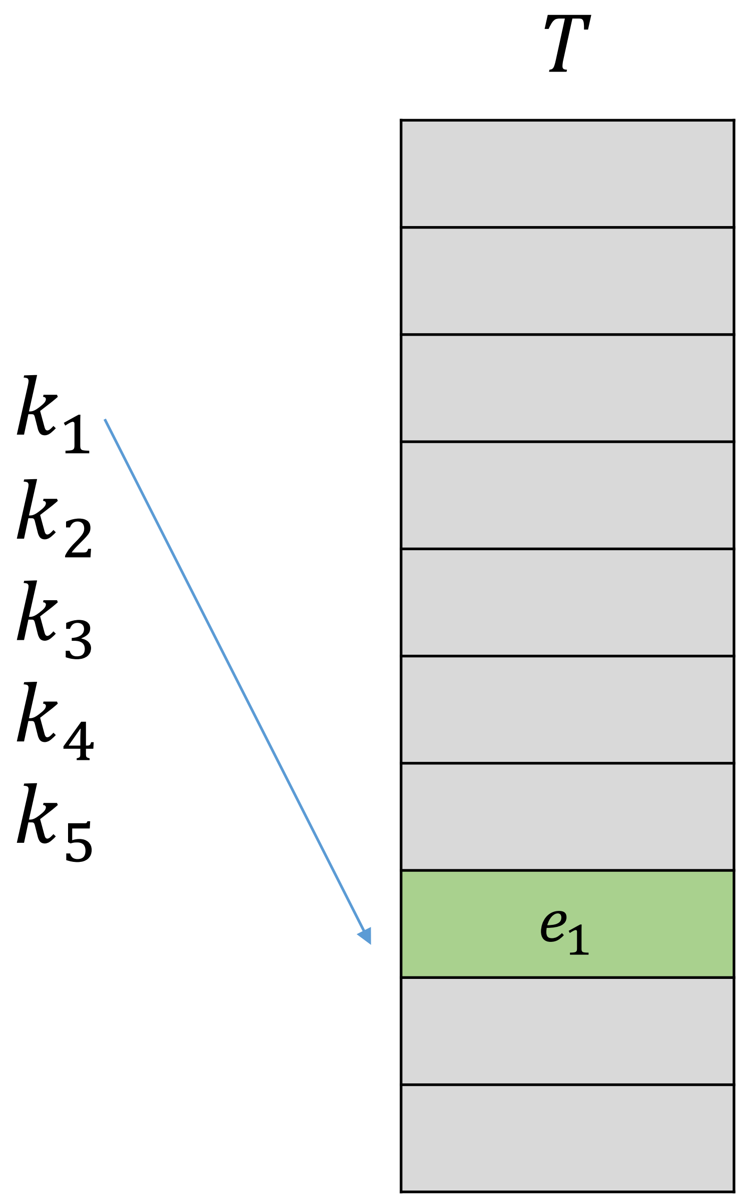


Linear probing

- Say we are inserting five elements whose keys are $k_1, k_2, \dots, k_5$.

- Assume that:
  $$h(k_2) = h(k_3) = h(k_5)$$

- e_i is the element with key k_i.

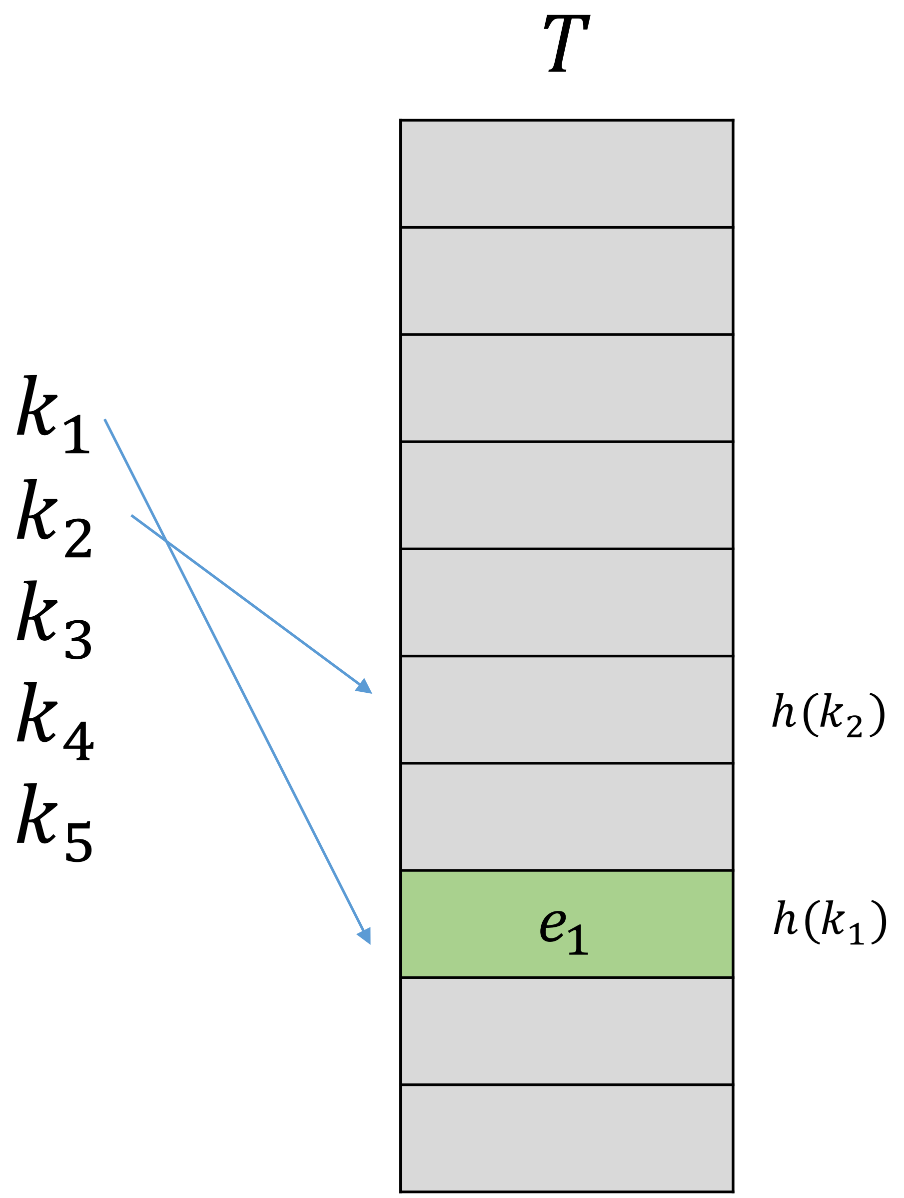


Linear probing

- Say we are inserting five elements whose keys are $k_1, k_2, \ldots, k_5$.

- Assume that:
  $$h(k_2) = h(k_3) = h(k_5)$$

- e_i is the element with key k_i.

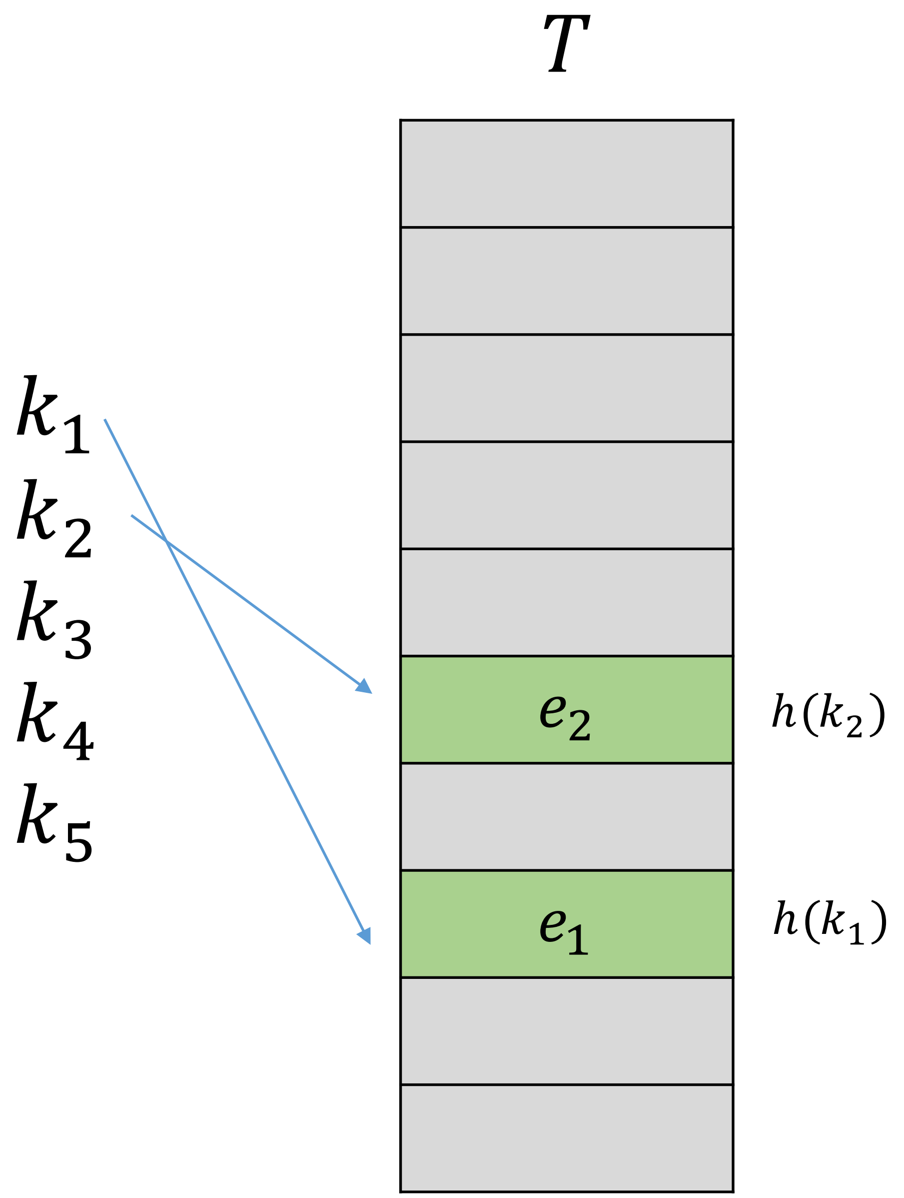


Linear probing

- Say we are inserting five elements whose keys are $k_1, k_2, \ldots, k_5$.

- Assume that:
  $$h(k_2) = h(k_3) = h(k_5)$$

- e_i is the element with key k_i.

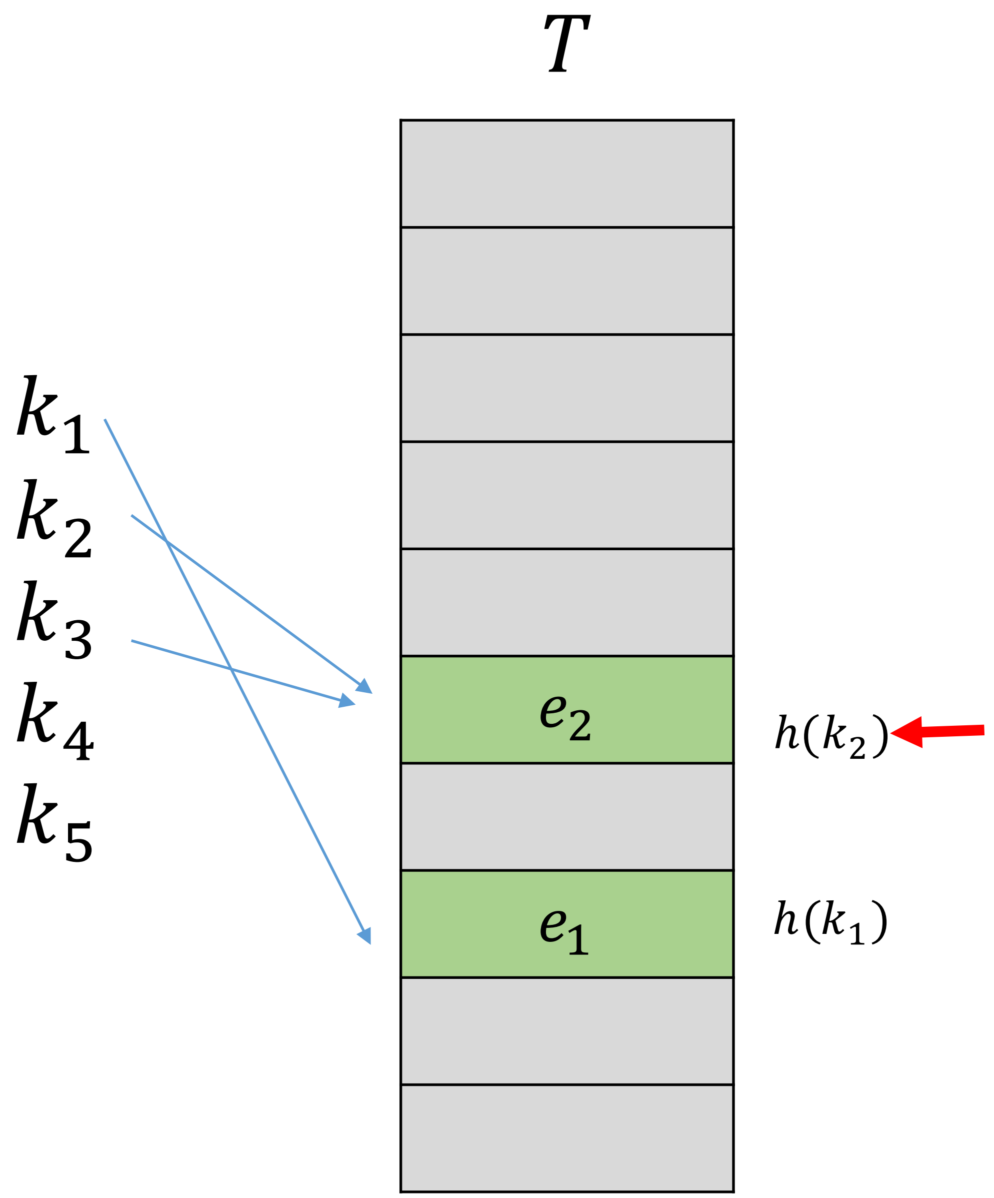


Linear probing

- Say we are inserting five elements whose keys are $k_1, k_2, \ldots, k_5$.

- Assume that:
  $$h(k_2) = h(k_3) = h(k_5)$$

- e_i is the element with key k_i.

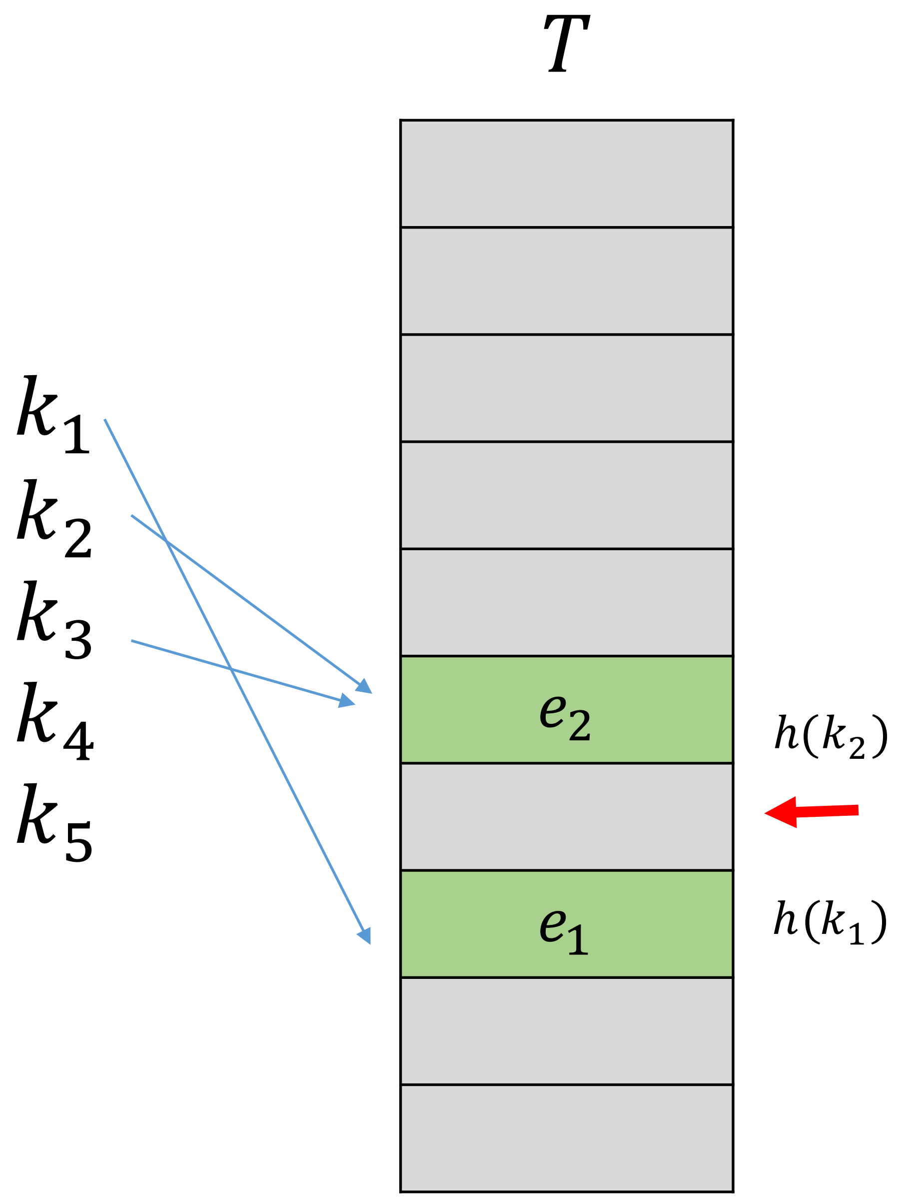


Linear probing

- Say we are inserting five elements whose keys are $k_1, k_2, \ldots, k_5$.

- Assume that:
  $$h(k_2) = h(k_3) = h(k_5)$$

- e_i is the element with key k_i.

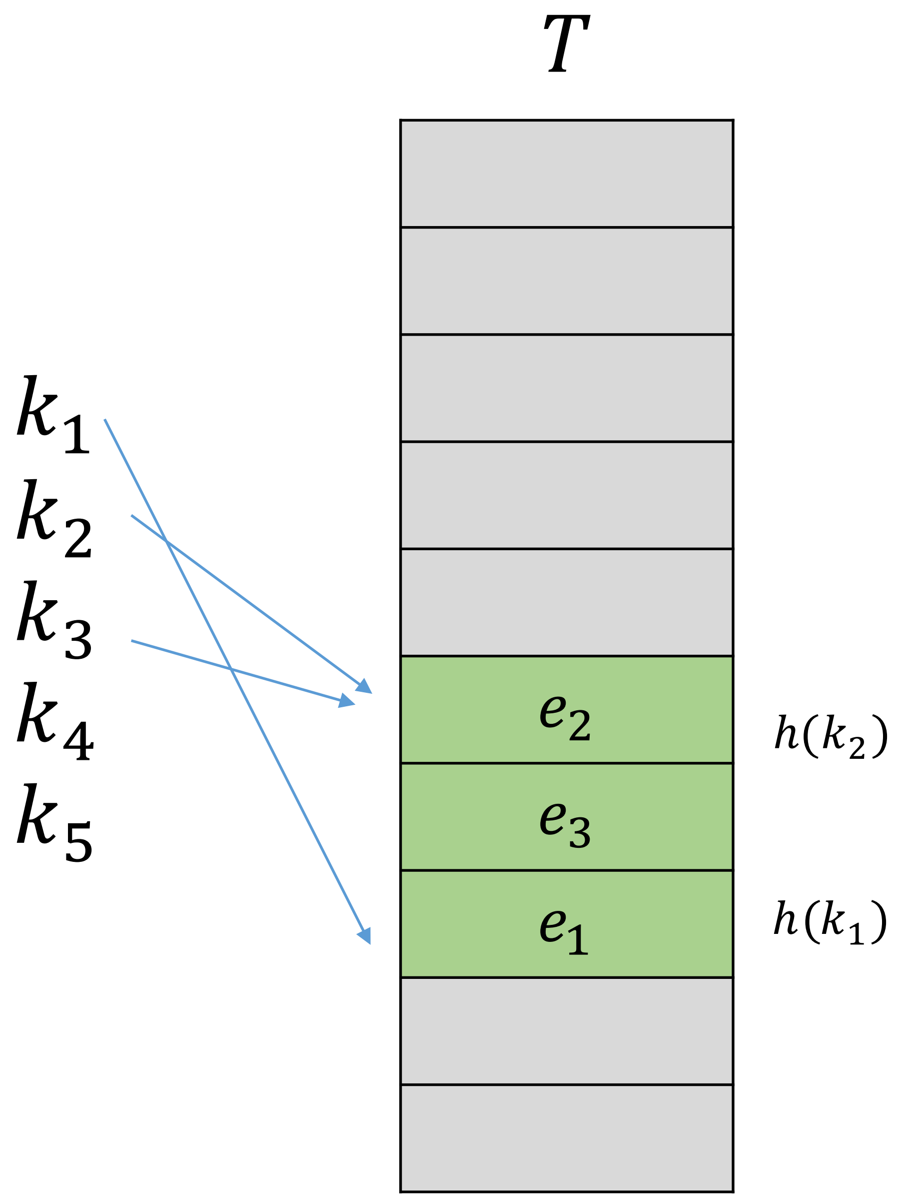


Linear probing

- Say we are inserting five elements whose keys are $k_1, k_2, \ldots, k_5$.

- Assume that:
  $$h(k_2) = h(k_3) = h(k_5)$$

- e_i is the element with key k_i.

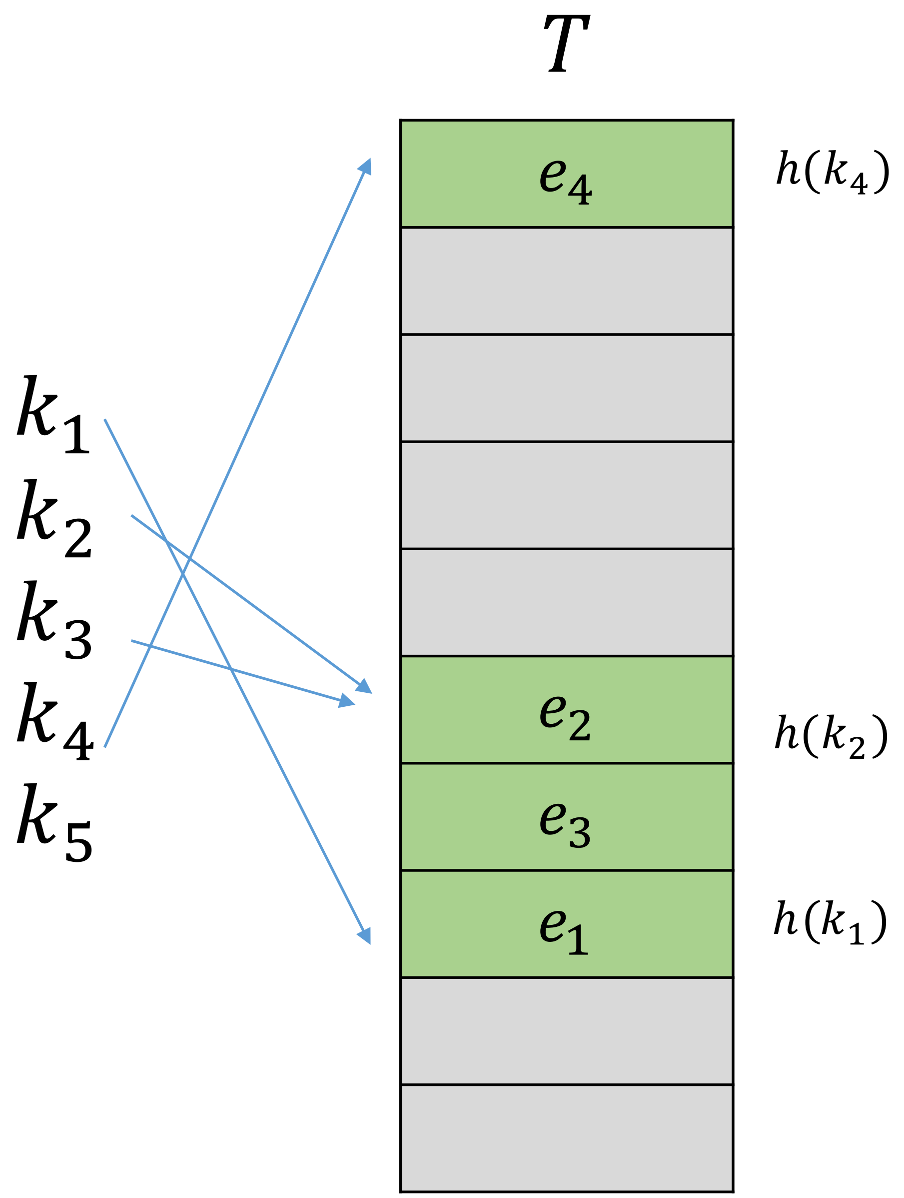


Linear probing

- Say we are inserting five elements whose keys are $k_1, k_2, \dots, k_5$.

- Assume that:
  $$h(k_2) = h(k_3) = h(k_5)$$

- e_i is the element with key k_i.



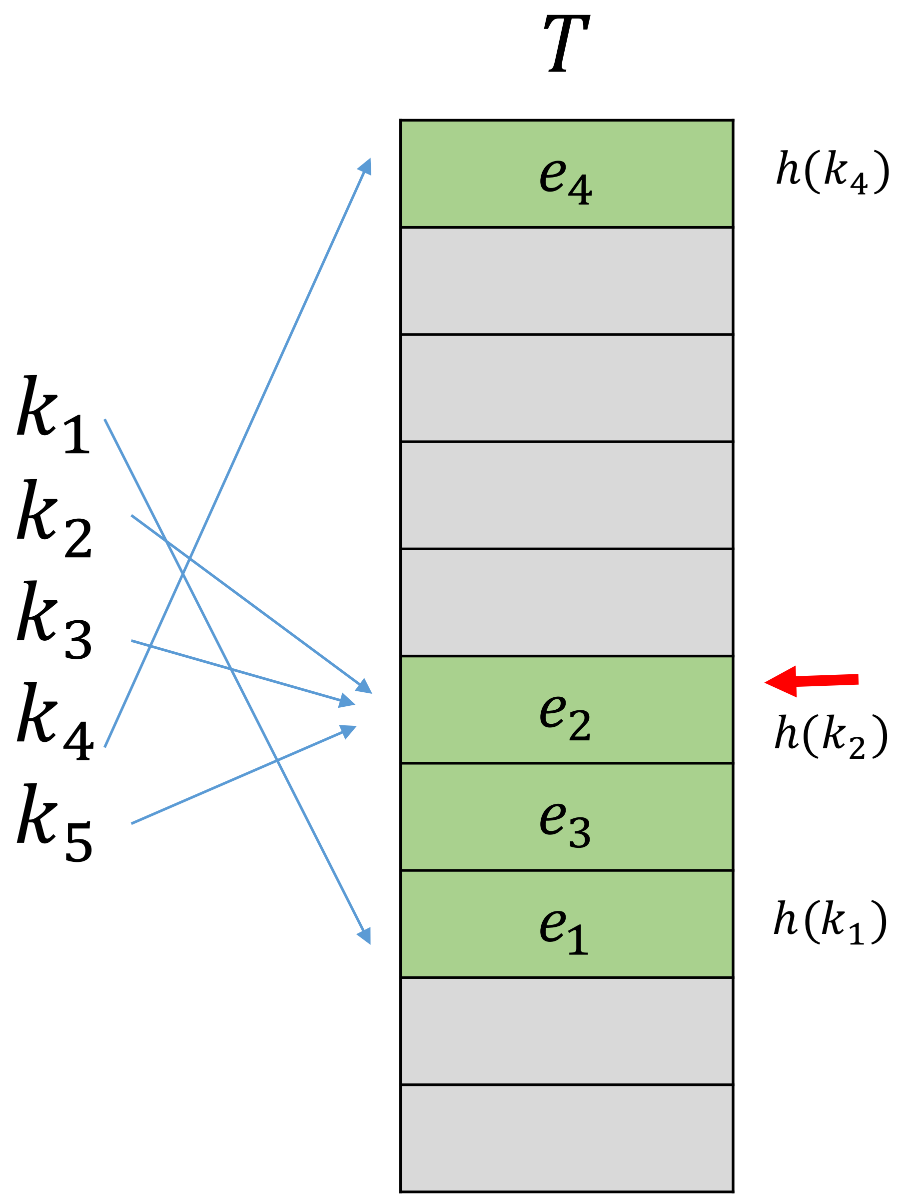
Linear probing

- Say we are inserting five elements whose keys are $k_1, k_2, \ldots, k_5$.

- Assume that:

  $$h(k_2) = h(k_3) = h(k_5)$$

- e_i is the element with key k_i.

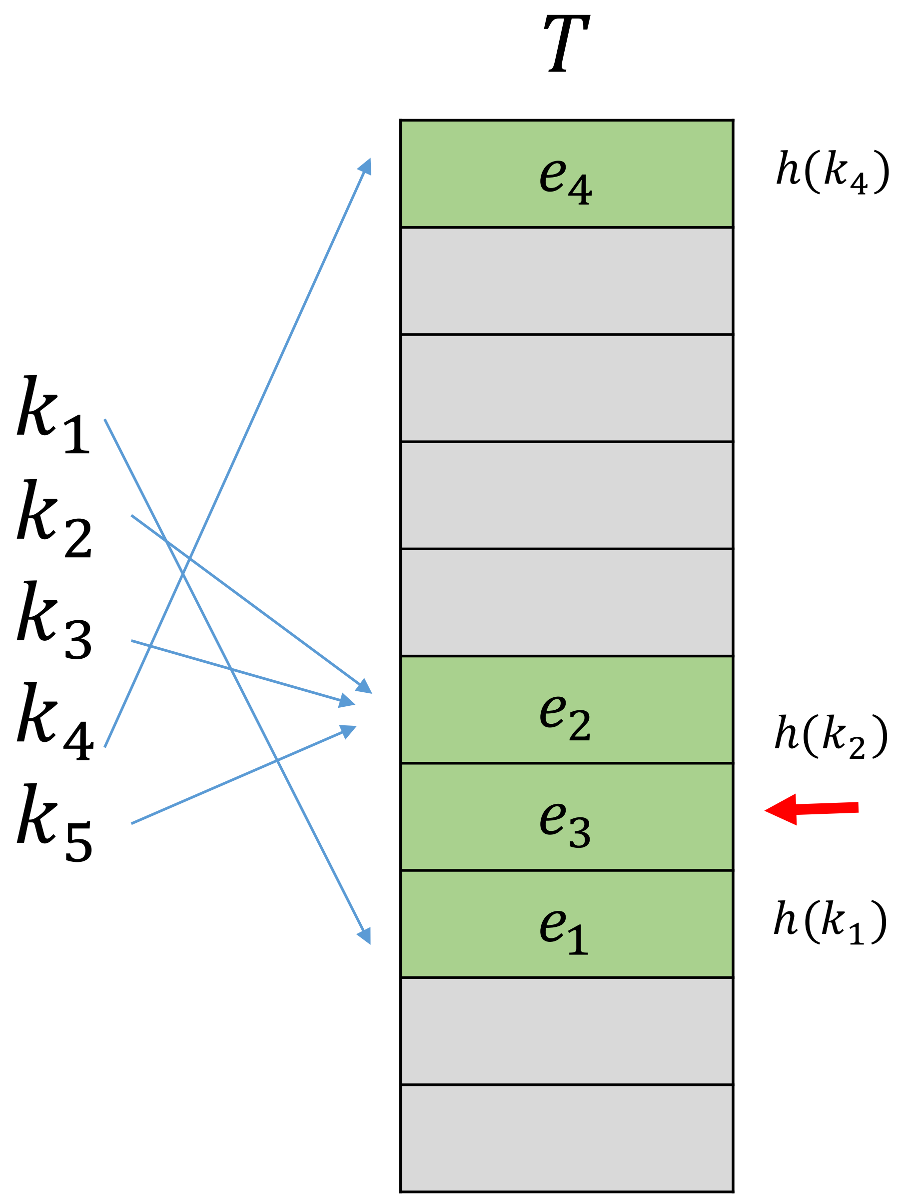


Linear probing

- Say we are inserting five elements whose keys are $k_1, k_2, \ldots, k_5$.

- Assume that:
  $$h(k_2) = h(k_3) = h(k_5)$$

- e_i is the element with key k_i.

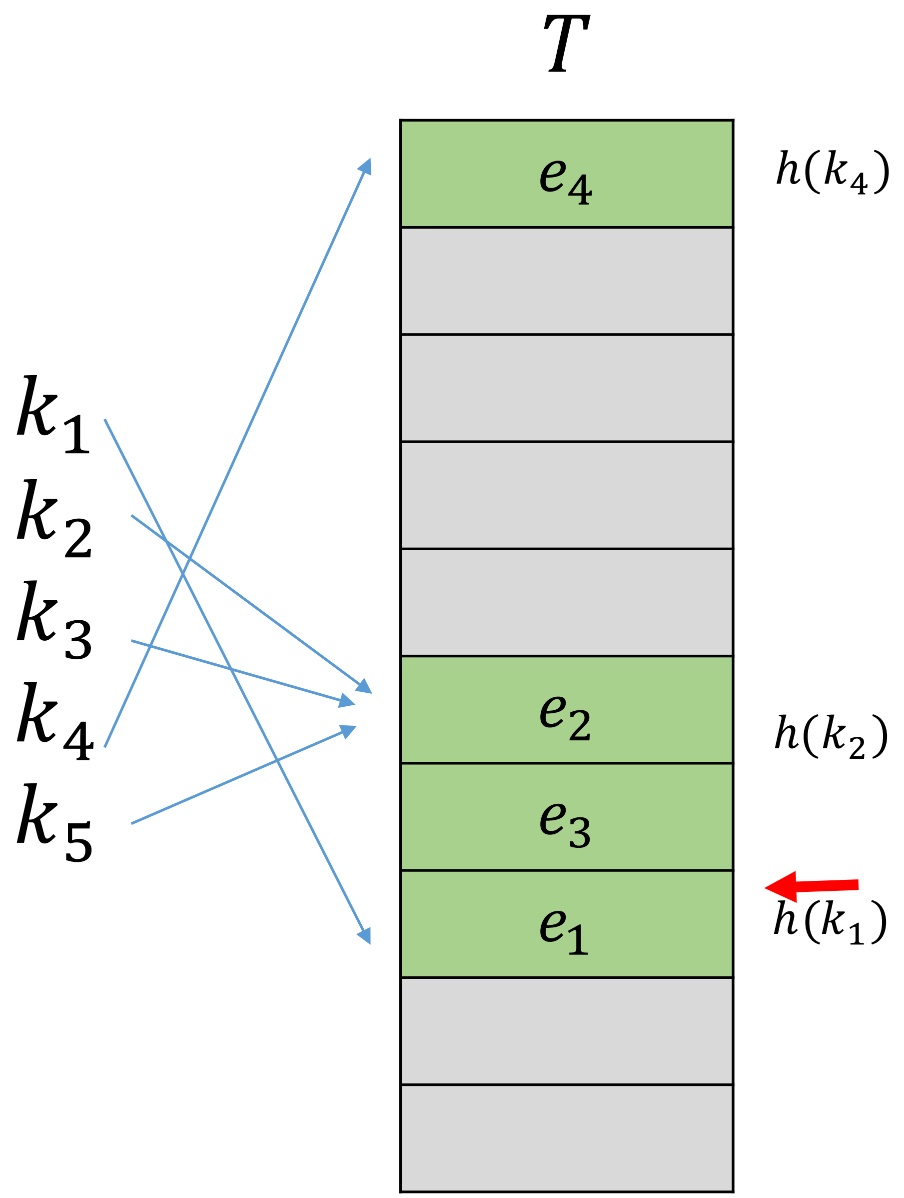


Linear probing

- Say we are inserting five elements whose keys are $k_1, k_2, \ldots, k_5$.

- Assume that:
  $$h(k_2) = h(k_3) = h(k_5)$$

- e_i is the element with key k_i.

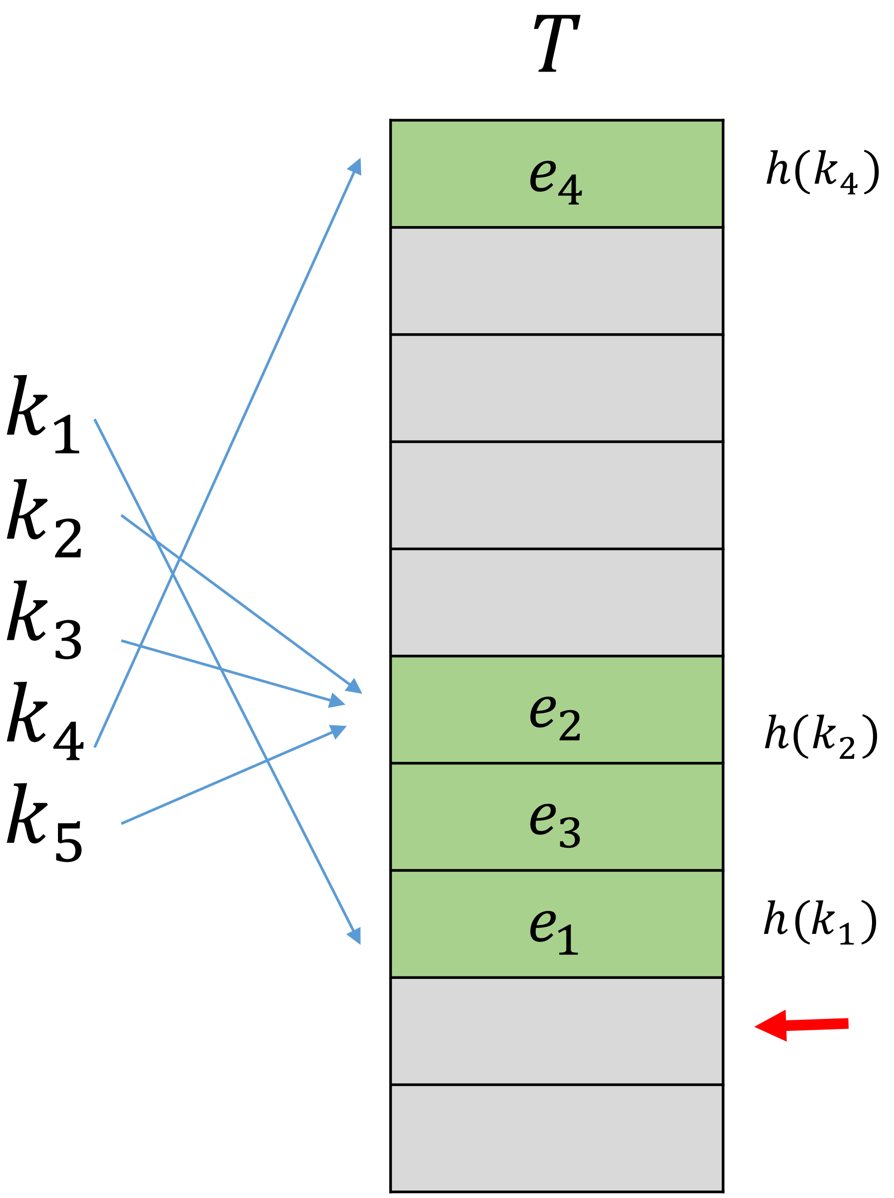


Linear probing

- Say we are inserting five elements whose keys are $k_1, k_2, \ldots, k_5$.

- Assume that:
$$h(k_2) = h(k_3) = h(k_5)$$

- e_i is the element with key k_i.

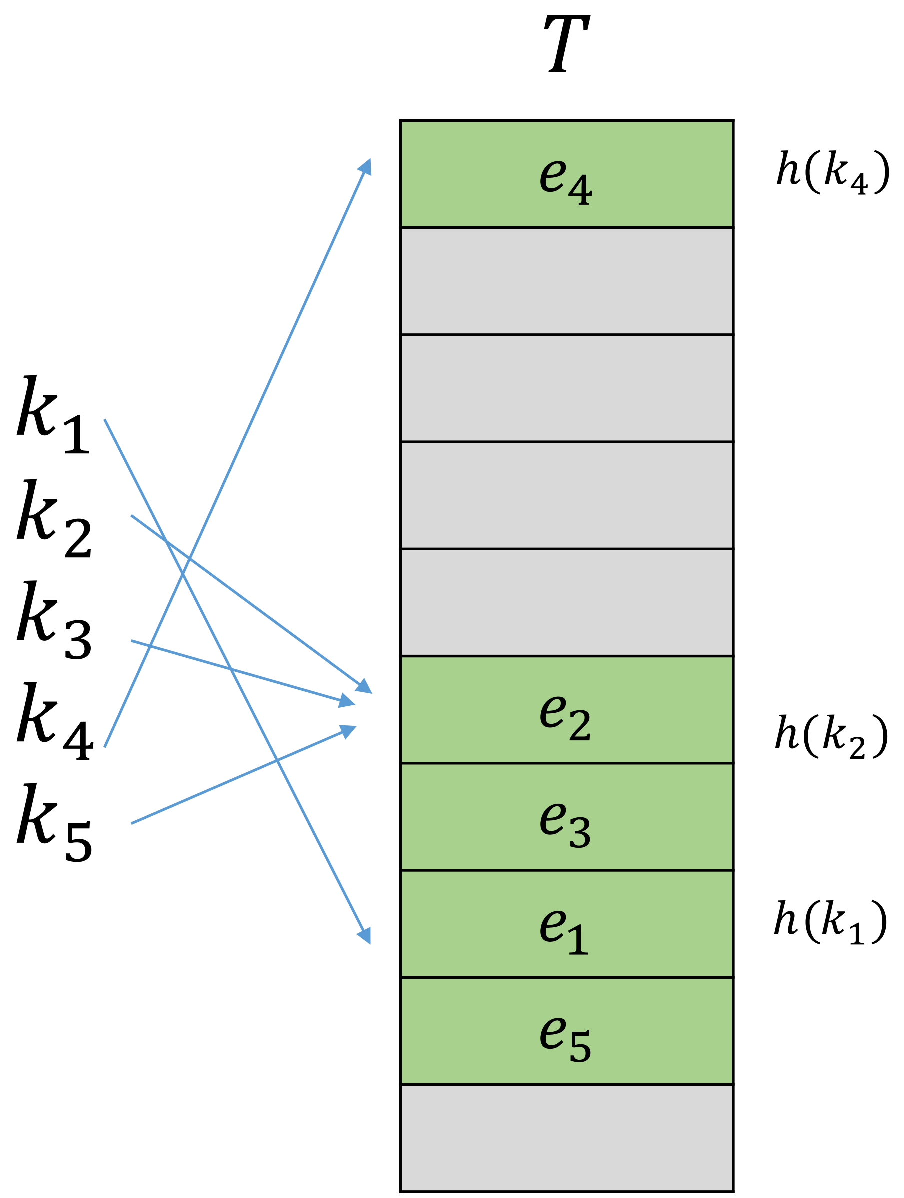


Linear probing

- Say we are inserting five elements whose keys are $k_1, k_2, \ldots, k_5$.

- Assume that:
$$h(k_2) = h(k_3) = h(k_5)$$

- e_i is the element with key k_i.



Formalization

- In general, in probing we are examining a sequence of indices of the hash-table called **probe sequence**.

- So, we generalize the hash function as follows:

$$h: U \times \{0, 1, \dots, m-1\} \to \{0, 1, \dots, m-1\}$$

Formalization

- In general, in probing we are examining a sequence of indices of the hash-table called **probe sequence**.

- So, we generalize the hash function as follows:

$$h: \underbrace{U \times \{0, 1, \dots, m-1\}} \rightarrow \{0, 1, \dots, m-1\}$$

A **Cartesian product** here means that the we are mapping **ordered pairs**.

Formalization

- Now, we have $h(k, i)$ that shows which index to examine on attempt $(i + 1)$, when searching or inserting.

- So, the probe sequence is
$$\langle h(k, 0), h(k, 1), \dots, h(k, m - 1)\rangle$$

Formalization

- Now, we have $h(k, i)$ that shows which index to examine on attempt $(i + 1)$, when searching or inserting.

- So, the probe sequence is
$$\langle h(k, 0), h(k, 1), \ldots, h(k, m - 1) \rangle$$

- **Question:** How many probe sequences are possible?

Formalization

- Now, we have $h(k, i)$ that shows which index to examine on attempt $(i + 1)$, when searching or inserting.

- So, the probe sequence is

$$\langle h(k, 0), h(k, 1), \dots, h(k, m - 1) \rangle$$

- **Question:** How many probe sequences are possible?
- **Answer:** $\boldsymbol{m}!$, since we require the probe sequence to be a permutation of $\langle 0,1, \dots, m - 1 \rangle$. Otherwise, some slots might be repeated or never appear in the probe sequence (we don't want that)!

Formalization

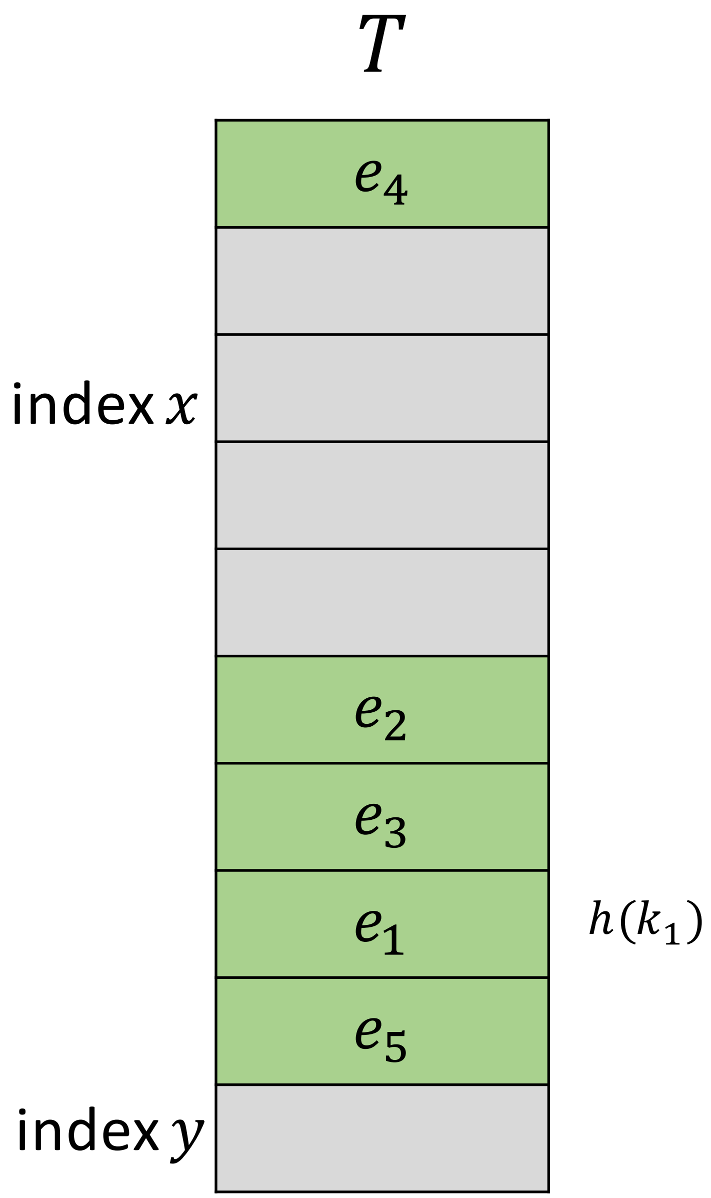
- In linear probing we simply had
$$h(k, i) = (h'(k) + i) \ \% \ m$$

- h' is called an **auxiliary hash function**.

- From now on, when using the open-addressing technique we write the hash function as $h(k, i)$, instead of $h(k)$.

Issue with linear probing

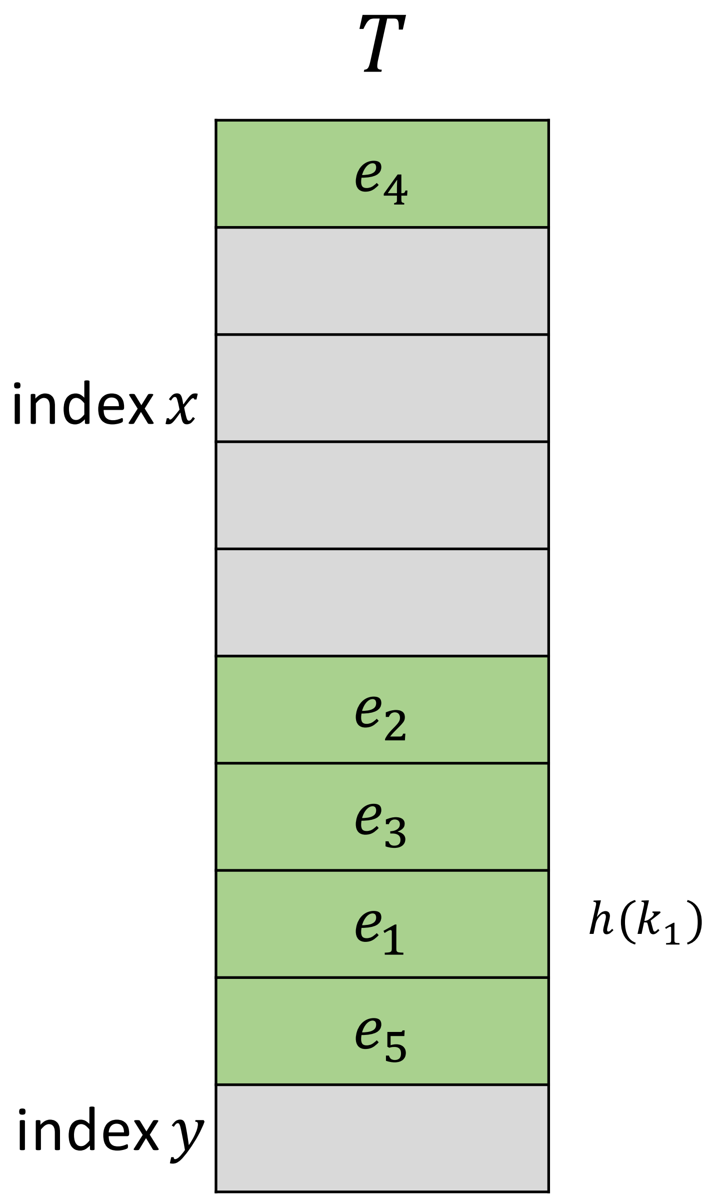
- Imagine in table T, a new element e with key k wants to be inserted and $h'(k)$ is completely random.



Issue with linear probing

- Imagine in table T, a new element e with key k wants to be inserted and $h'(k)$ is completely random.

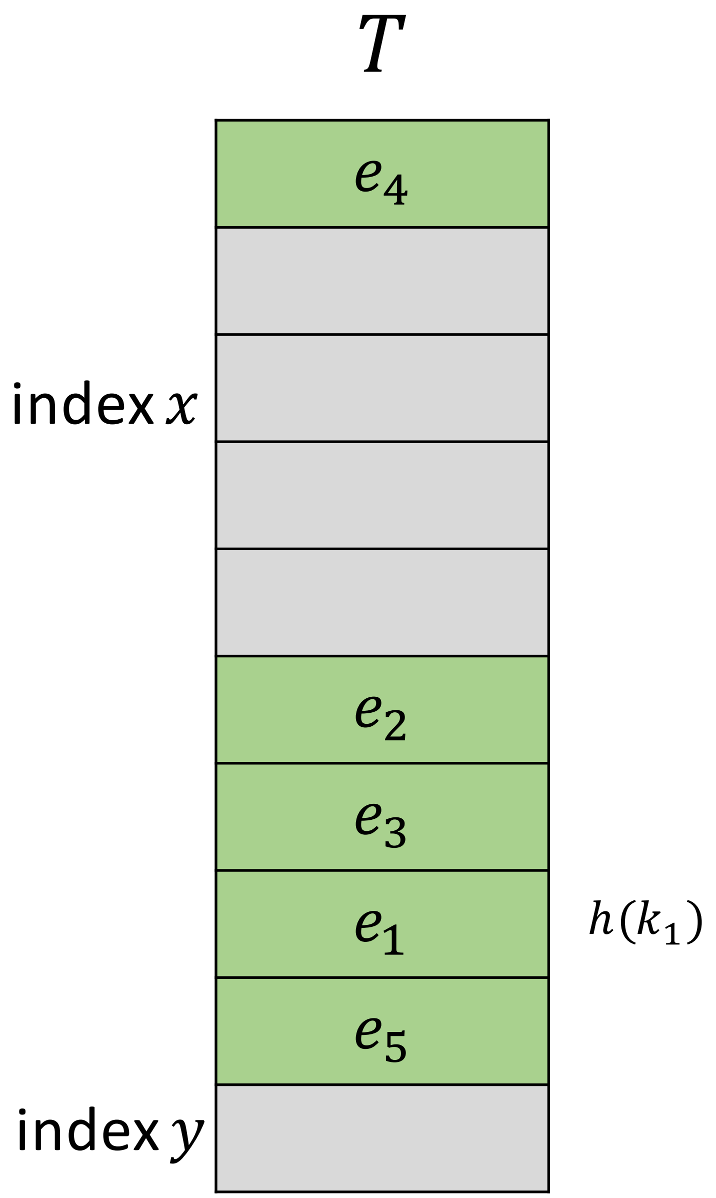
- **Question 1:** what is the probability of e being inserted in index x? And in index y?



Issue with linear probing

- Imagine in table T, a new element e with key k wants to be inserted and $h'(k)$ is completely random.

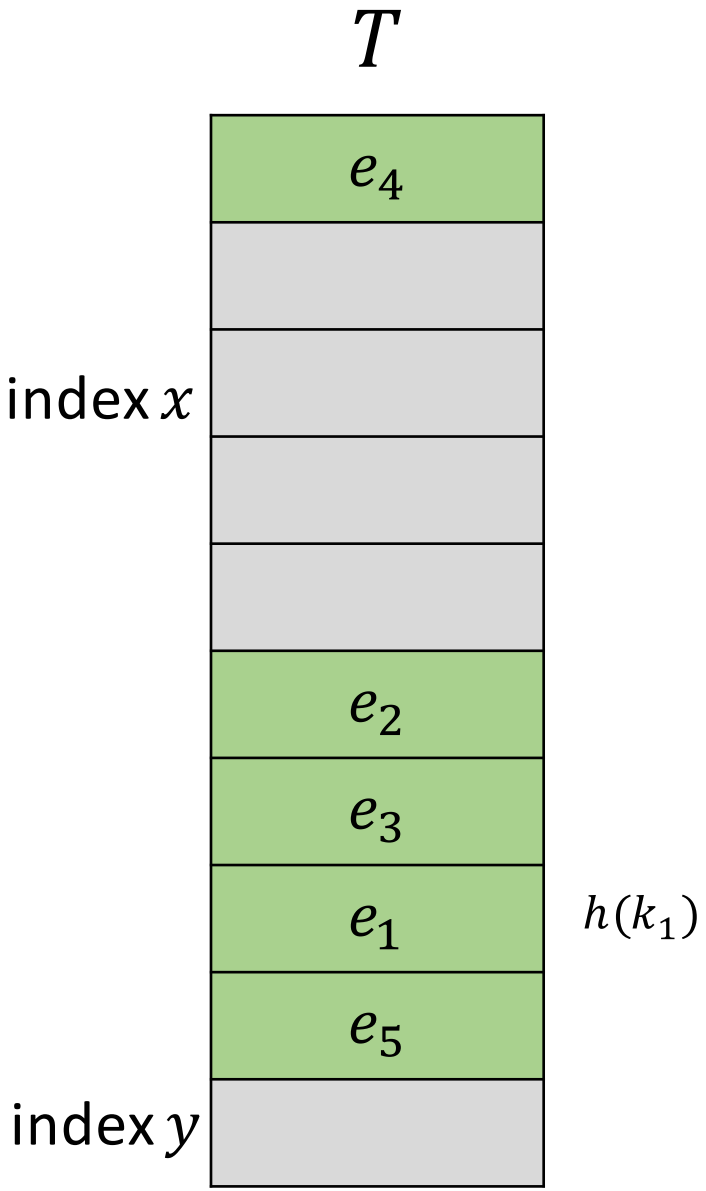
- **Answer:** At x is 1/10, while at y is 5/10. Because for the key to be inserted at y, it's enough that $h'(k)$ is equal to any of the last 5 indices. But for x, $h'(k)$ has to be exactly equal to x.



Issue with linear probing

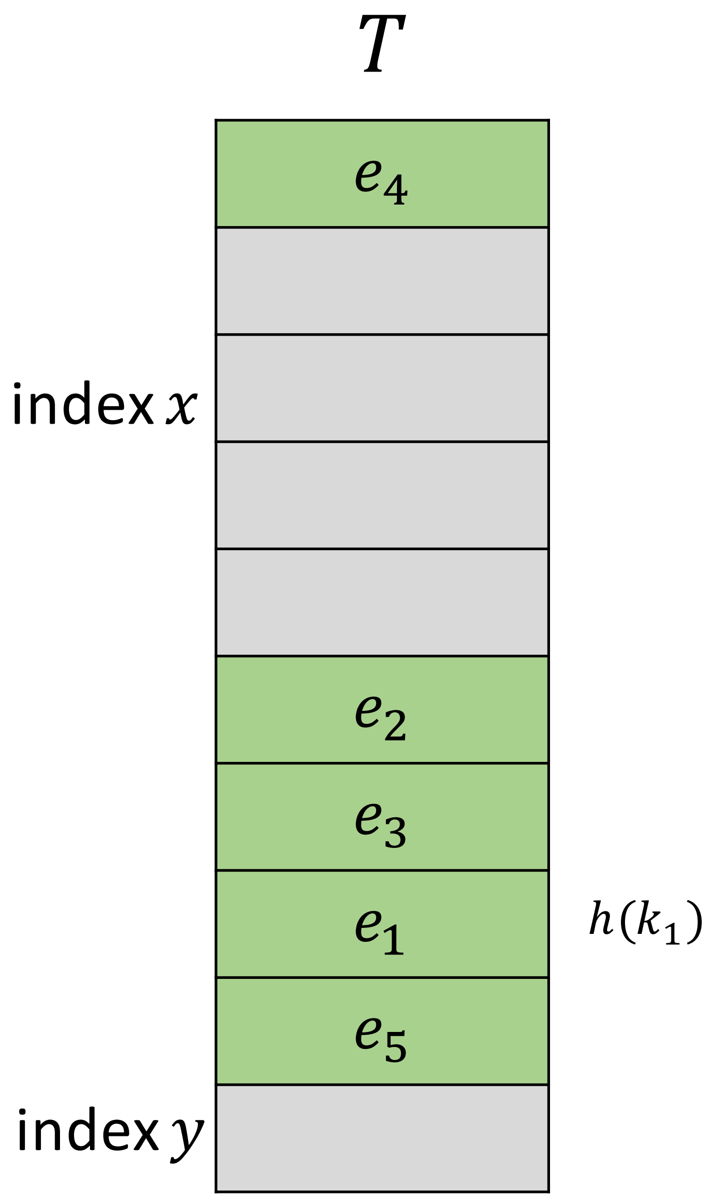
- The first issue is that if an empty slot is **preceded with more full** slots it is more likely to be occupied.

- This issue is known as **clustering**.



Issue with linear probing

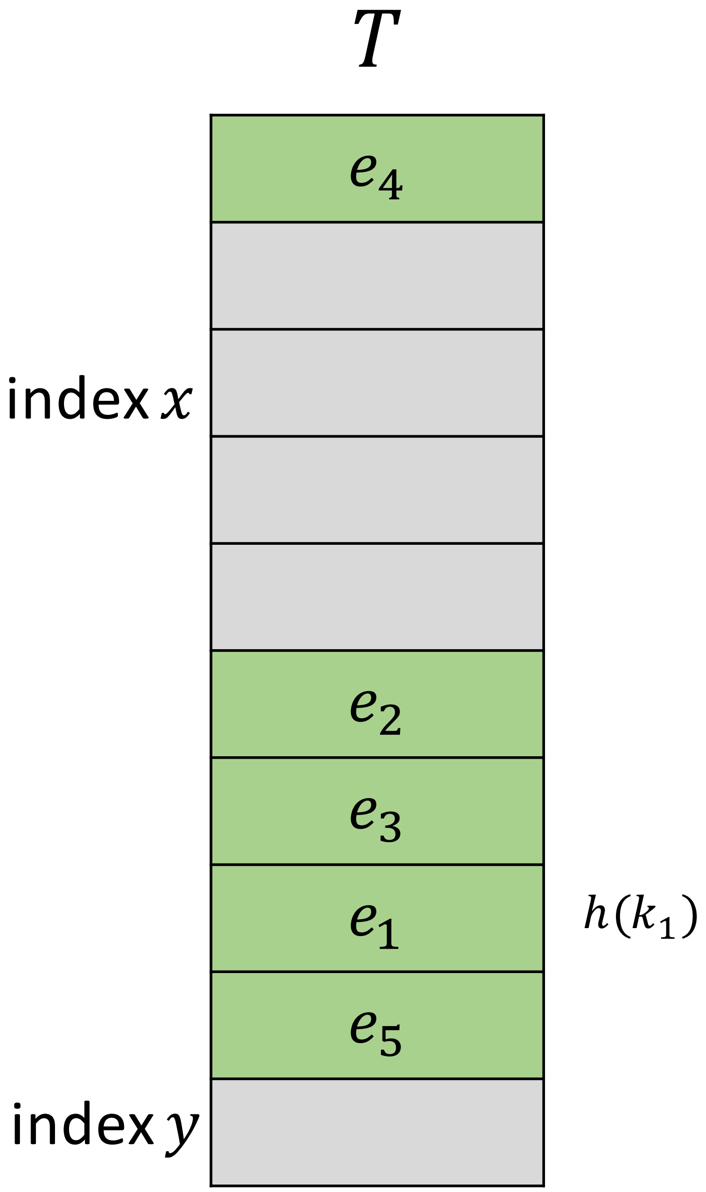
- **Question 2:** how many probe sequences are possible **in linear probing**? $(h(k, i) = (h'(k) + i) \% m)$



Issue with linear probing

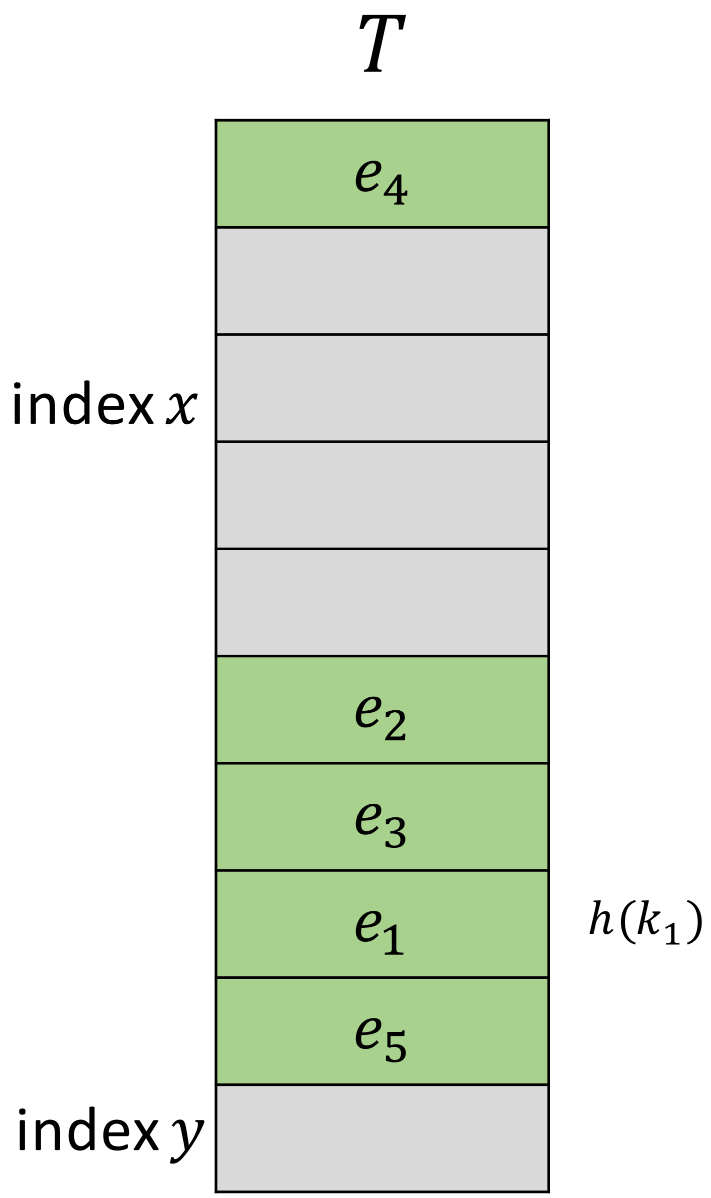
- **Question 2:** how many probe sequences are possible **in linear probing**? $(h(k,i) = (h'(k) + i) \,\%\, m)$

- **Answer:** For any key, $h(k)$ determines the rest of the sequence, so only m sequences.



Issue with linear probing

- So, the second issue is that out of all $m!$ possible probe sequences we are only using m sequences; so, the values obtained from $h(k, i)$ **do not appear to be random** unlike what we expect from a good hash function.



Quadratic probing

- Linear was no good so we do quadratic :)

- $h(k, i) = (h'(k) + c_1 i + c_2 i^2) \,\%\, m$

- The choices for c_1, c_2 and m, should be in way that we can guarantee the values for $h(k, i)$ not repeated so this hash function probes all cells from 0 to $m - 1$.

Quadratic probing

- Quadratic probing also has the clustering issue but it’s not as bad as linear probing.

- Also, just by having $h'(k)$ we have the whole probe sequence; so, there are only m probe sequences possible.

Quadratic probing

- A good choice is to pick $\boldsymbol{c_1 = c_2 = 1/2}$ and pick $\boldsymbol{m}$ **a power of 2**

$$h(k,i) = \left(h'(k) + \frac{1}{2}i + \frac{1}{2}i^2\right) \% m$$

- We can prove that for any $0 \leq i < i' < m$, then $h(k,i) \neq h(k,i')$.

- As a result, $h(k,0), h(k,1), h(k,2), \ldots, h(k,m-1)$ are all different cells of the hash table.

Proof

- Assume that we pick i, i' such that $0 \leq i < i' < m$
- We show that if $h(k, i) = h(k, i')$, then we reach a logical contradiction, so it must be that i is actually equal to i' $(i = i')$.
- Therefore, it's impossible that the a cell is probed twice during our m attempts.
- This is known as **proof by contradiction**.

Alternative probing strategy

- We want each probe sequence

$$\langle h(k,0), h(k,1), \ldots, h(k,m-1) \rangle$$

  to be **equally likely**.

Alternative probing strategy

- This is the **uniform hashing** assumption.
- This is a generalization of the **simple uniform hashing** we had before.
- Finding a uniform hash function is difficult.
- But we introduce a probing strategy called **double hashing** that in practice works **as well as a uniform hash**.

Double hashing

- We simply use two auxiliary hash functions h_1, h_2:

$$h(k,i) = (h_1(k) + (i \cdot h_2(k)))\% \, m$$

Double hashing

- We simply use two auxiliary hash functions h_1, h_2:
$$h(k,i) = (h_1(k) + (i \cdot h_2(k)))\%\, m$$

- To make sure that the obtained probe sequence is a permutation of $\langle 0, \ldots, m-1 \rangle$, $\boldsymbol{h_2(k)}$ **and** $\boldsymbol{m}$ should be **relatively prime.**
- Two numbers are relatively prime if they don’t have a common factor.

Double hashing

- There are two ways to do this:

1. The easy way is to pick m a power of 2, and make $h_2(k)$ always produce an odd number.

2. Pick m to be a prime, and have $h_2(k)$ to always produce an integer less than m.

Double hashing

- Each pair of $(h_1(k), h_2(k))$ yields a different probe sequence; so, overall there will be roughly **m^2 probing sequences** (instead of m probe sequences in linear and quadratic probing).

- This is still not close to $m!$ but in practice it works very well.

Implementation

```
HASH-INSERT(T, k)
i = 0
repeat
    j = h(k, i)
    if T[j] == NIL
        T[j] = k
        return j
    else i = i + 1
until i == m
error "hash table overflow"
```

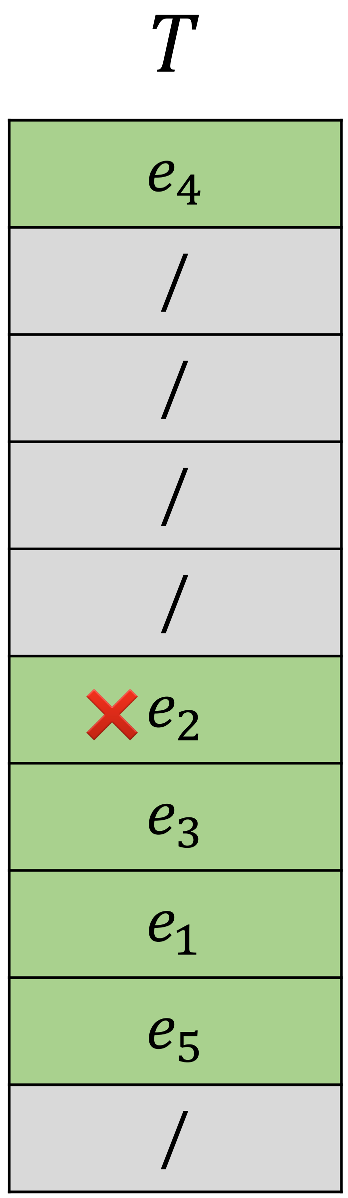
This pseudocode is for inserting a key rather than an element.

Implementation

```
HASH-SEARCH(T, k)
i = 0
repeat
    j = h(k, i)
    if T[j] == k
        return j
    i = i + 1
until T[j] == NIL or i == m
return NIL
```

Implementation

- What about DELETE?

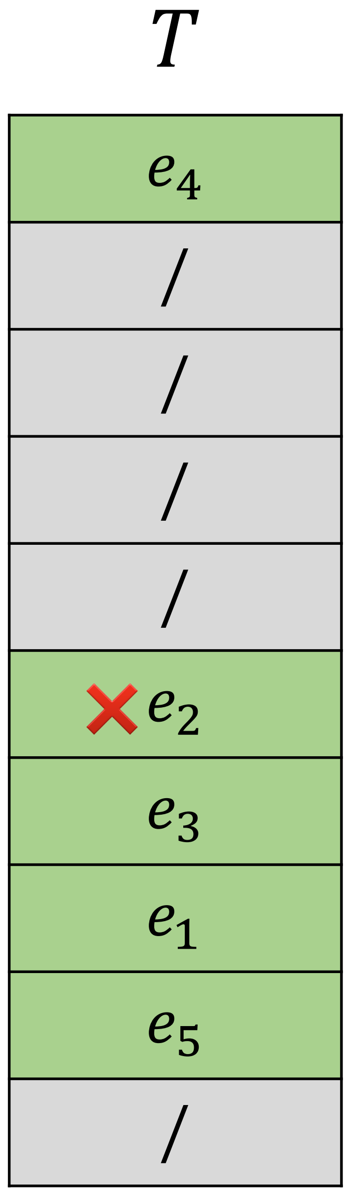


Implementation

- What about DELETE?

- Say $h(k_2) = h(k_3)$, and e_3 was inserted after e_2

- I delete e_2 and mark it's cell as *empty*.

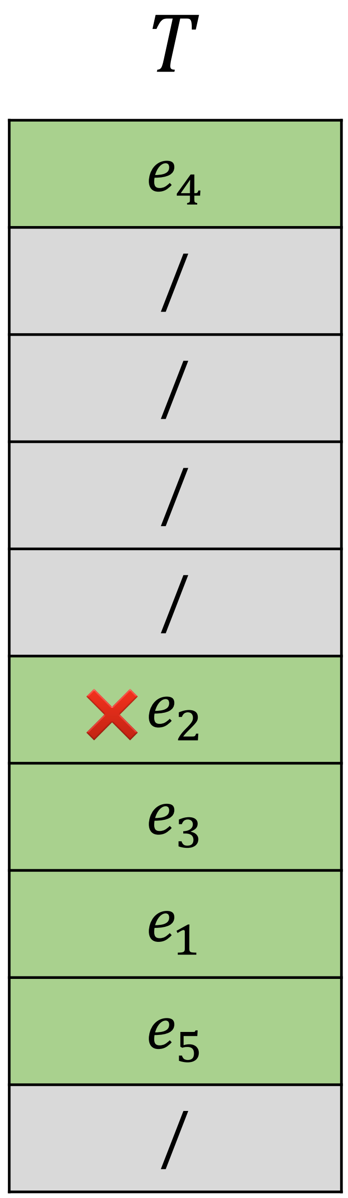


Implementation

- What about $\textsc{Delete}$?

- Say $h(k_2) = h(k_3)$, and e_3 was inserted after e_2

- I delete e_2 and mark it's cell as *empty*.
- **Q:** what's the answer to $\textsc{Hash-Search}(T, k_3)$?

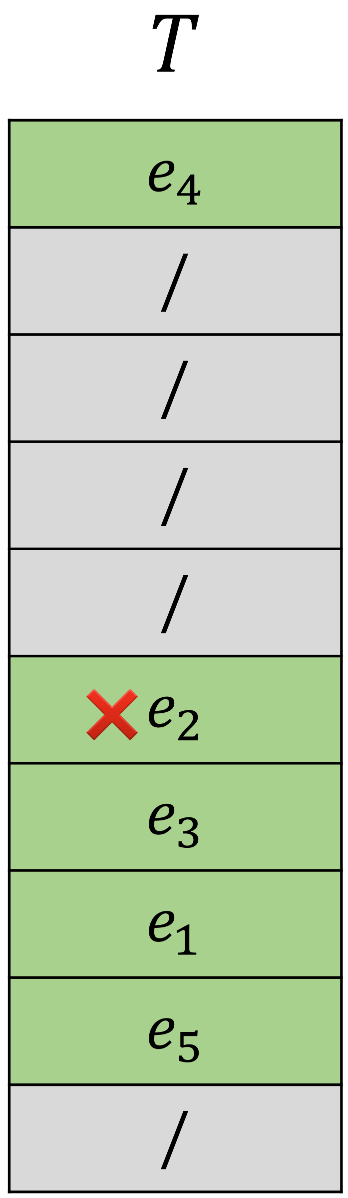


Implementation

- What about DELETE?

- Say $h(k_2) = h(k_3)$, and e_3 was inserted after e_2

- I delete e_2 and mark it's cell as *empty*.
- **Q:** what's the answer to HASH-SEARCH(T, k_3)?
- **A:** NIL, even though the element with key k_3 is in the table!!



Implementation

- The way to solve this is to mark that cell with a special symbol *<u>‘deleted’.</u>*

- Insert would treat a deleted symbol as empty.
- Search would pass over a deleted cell since it’s not NIL.

T
e_4
/
/
/
/
$Deleted$
e_3
e_1
e_5
/