
Your Ultimate Guide To Landing
Top AI roles
DECODE
AiML

2.12

Hash Table Implementation in Python

→ A hash table is a data structure that stores key-value pairs and allows fast access (search) to values based on their keys.

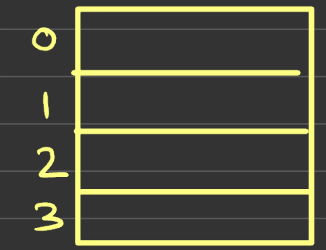
↳ Amortized O(1) time.

→ In Python, **dict** and **set** are implemented using Hash Table data structure.

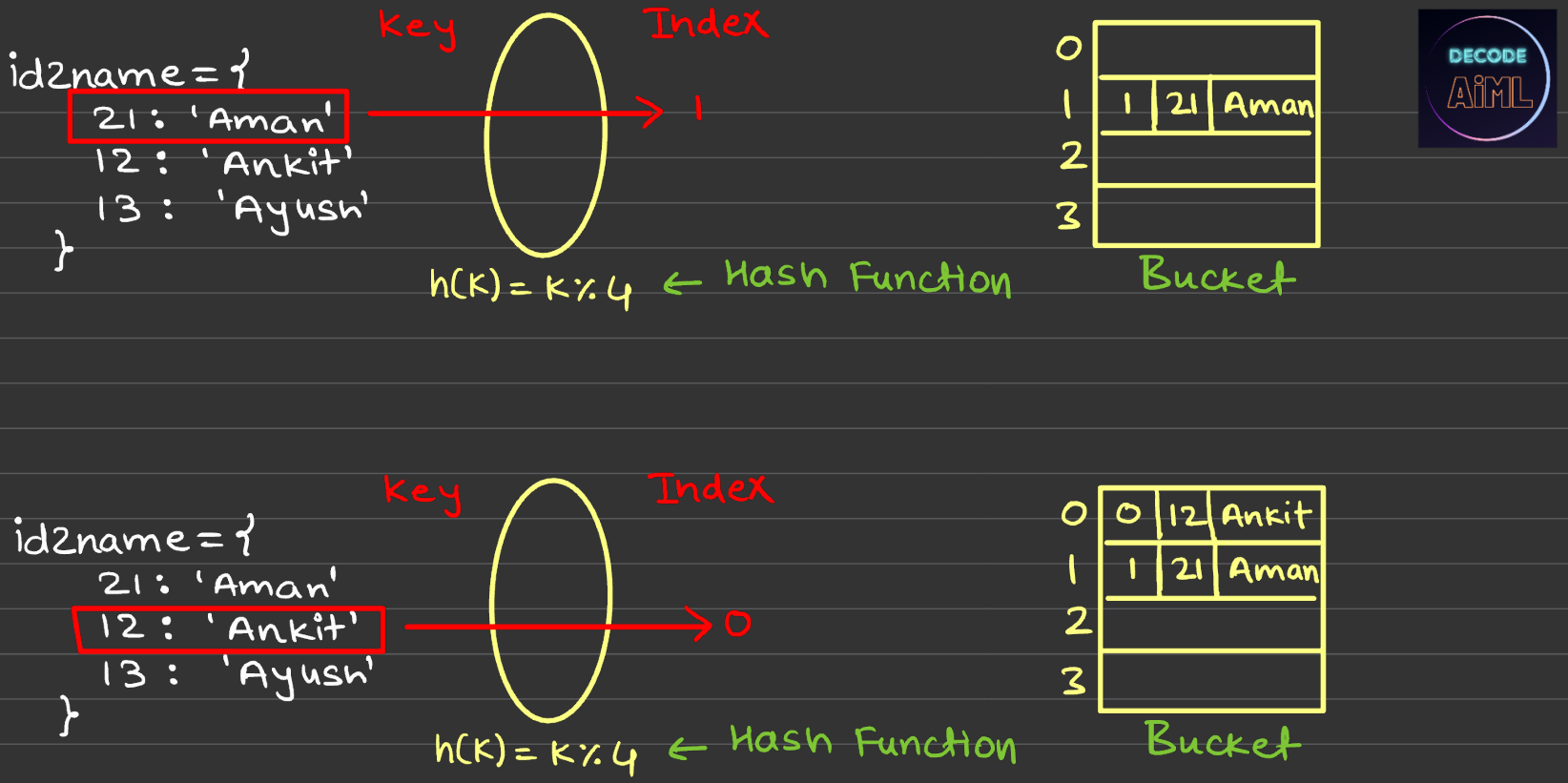
→ Each element in set is stored as a dict key with a dummy value (usually None)

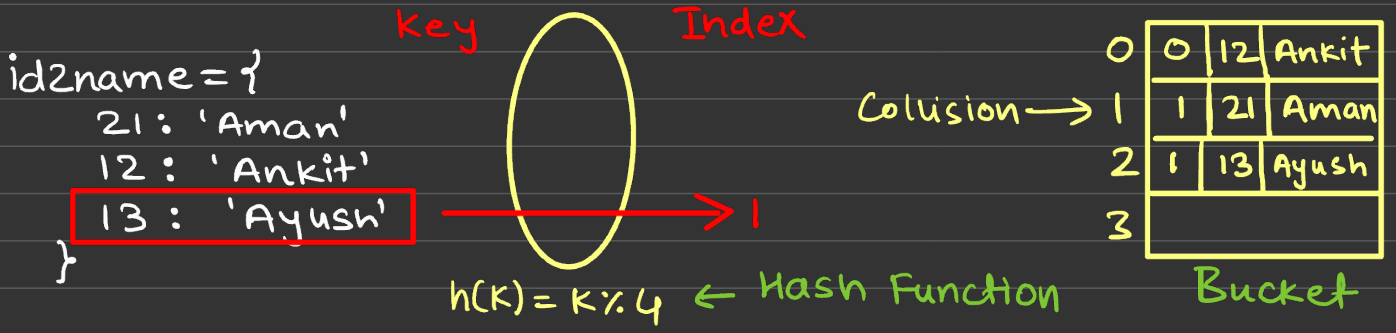
```
id2name={
    21: 'Aman'
    12: 'Ankit'
    13: 'Ayush'
}
```

h(K) = K % 4 ← Hash Function



Bucket


id2name={
21 : 'Aman'
12 : 'Ankit'
13 : 'Ayush'
}
Key
Index
1
h(K) = K % 4
Hash Function
0
1
2
3
1
21
Aman
Bucket
id2name={
21 : 'Aman'
12 : 'Ankit'
13 : 'Ayush'
}
Key
Index
0
h(K) = K % 4
Hash Function
0
1
2
3
0
12
Ankit
1
21
Aman
Bucket
DECODE
AiML



→ Collision happens when we try to insert value with key 13

→ so we use Quadratic probing in double Hashing.

$$h'(K, i) = (K + c_1 \cdot i + c_2 \cdot i^2) \% 4$$

if $c_1 = 0$ and $c_2 = 1$

$$h'(13, 1) = (13 + 1^2) \% 4 = 2$$

→ Python's Hash Table Implementation details

① Uses Open Addressing (Quadratic Probing) to resolve Collision

② Bucket stores entry directly in the array.

③ A bucket stores
   ⓘ dict → <hash, key, value>
   ⓘⓘ set → <hash, key>

④ Uses dynamic resizing when load factor gets too high (~2/3)
   ↳ during insertion.

⑤ Hashing, Probing and Resizing logic of set is same as a dict.

Operations in Hash Table

Expected no of probes $= 1/(1-\alpha)$

$$T(n) = O\left(1/(1-\alpha)\right)$$

Operation	Amortized T(n)	Worst T(n)
Insertion	O(1)	O(n)
Search	O(1)	O(n)
Delete	O(1)	O(n)
Update	O(1)	O(n)

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

If α is constant (~ 0.66)

↳ $T(n) = O(1)$

→ In hash table, the expensive operation is resizing.
- ↳ when the load factor(α) pass a threshold, double size table.
- ↳ Resizing requires rehashing all keys → O(n)
- ↳ But, resizing is a rare operation.
  - ↳ Amortized cost = O(1)

Hash Table using Dict

→ Refer Lecture 1.8.3 (Dictionary in one shot)

→ for a dict item, the values can be any datatype, but key should be Hashable, and Immutable

→ Initialization ← O(1)

```
my_dict = {}
```

→ Insertion ← O(1) Amortized

```
my_dict["name"] = "Sanjeev"
my_dict["age"] = 26
```

→ Search ← O(1) Amortized

```
my_dict["name"]
```

When the load factor stays below a threshold (≈ 2/3), Colusion are rare

→ Updation ← O(1) Amortized

```
my_dict["age"] = 26
```

→ Deletion ← O(1) Amortized

```
del my_dict["name"]
```

Hash Table using Set

→ Refer Lecture 1.8.2 (Sets in one shot)

→ Each element in set is stored as a dict key with a dummy value (usually None)

→ for a set datatype, the key should be hashable and mutable.

→ Initialization ← O(1)

```
my_set=set()
```

→ Insertion ← O(1) Amortized

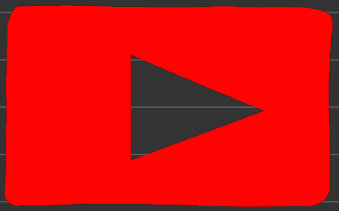
```
my_set.add("Aman")
my_set.add("Ankit")
```

→ Search ← O(1) Amortized

```
"Aman" in my_set
```

→ Deletion ← O(1) Amortized

```
my-set.remove("Aman")
```

Like  Subscribe