Maps, Dictionaries, Hash Tables

TUAN NGUYEN

Outline

Collections: list, set, tuple, dict { key value

Dictionaries

- Python's dict class is arguably the most significant data structure in the language.
- In dict, unique keys are mapped to associated values.

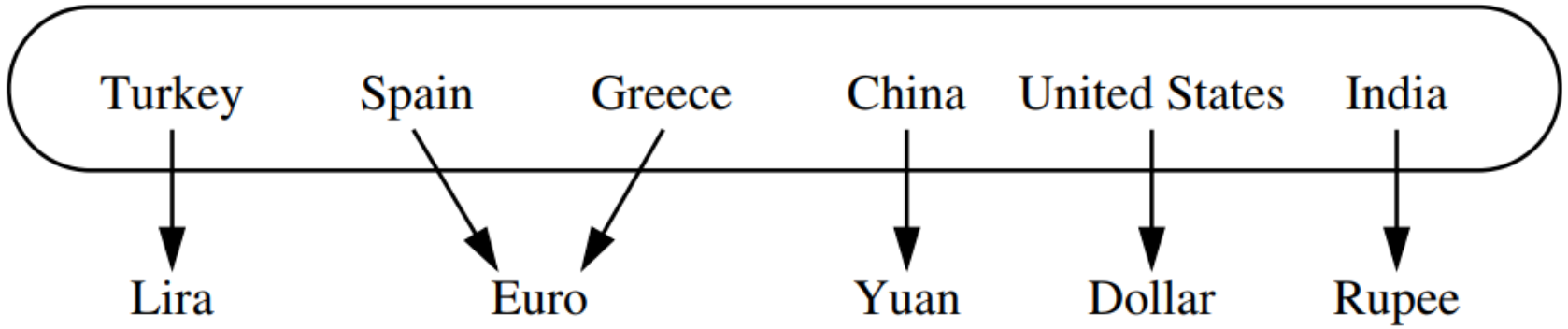


Figure 10.1: A map from countries (the keys) to their units of currency (the values).

Dict's functions

M[k]: Return the value v associated with key k in map M, if one exists; otherwise raise a KeyError. In Python, this is implemented with the special method __getitem__.

M[k] = v: Associate value v with key k in map M, replacing the existing value if the map already contains an item with key equal to k. In Python, this is implemented with the special method __setitem__.

del M[k]: Remove from map M the item with key equal to k; if M has no such item, then raise a KeyError. In Python, this is implemented with the special method __delitem__.

len(M): Return the number of items in map M. In Python, this is implemented with the special method __len__.

iter(M): The default iteration for a map generates a sequence of *keys* in the map. In Python, this is implemented with the special method __iter__, and it allows loops of the form, **for** k **in** M.

Dict's functions (1)

k in M:	Return True if the map contains an item with key k. In Python, this is implemented with the special __contains__ method.
M.get(k, d=None):	Return M[k] if key k exists in the map; otherwise return default value d. This provides a form to query M[k] without risk of a KeyError.
M.setdefault(k, d):	If key k exists in the map, simply return M[k]; if key k does not exist, set M[k] = d and return that value.
M.pop(k, d=None):	Remove the item associated with key k from the map and return its associated value v. If key k is not in the map, return default value d (or raise KeyError if parameter d is None).

Dict's functions (2)

M.clear( ):	Remove all key-value pairs from the map.
M.keys( ):	Return a set-like view of all keys of M.
M.values( ):	Return a set-like view of all values of M.
M.items( ):	Return a set-like view of (k,v) tuples for all entries of M.
M.update(M2):	Assign M[k] = v for every (k,v) pair in map M2.
M == M2:	Return True if maps M and M2 have identical key-value associations.
M != M2:	Return True if maps M and M2 do not have identical key-value associations.

Dict's example

Operation	Return Value	Map
len(M)	0	{}
M['K'] = 2	–	{'K': 2}
M['B'] = 4	–	{'K': 2, 'B': 4}
M['U'] = 2	–	{'K': 2, 'B': 4, 'U': 2}
M['V'] = 8	–	{'K': 2, 'B': 4, 'U': 2, 'V': 8}
M['K'] = 9	–	{'K': 9, 'B': 4, 'U': 2, 'V': 8}
M['B']	4	{'K': 9, 'B': 4, 'U': 2, 'V': 8}
M['X']	KeyError	{'K': 9, 'B': 4, 'U': 2, 'V': 8}
M.get('F')	None	{'K': 9, 'B': 4, 'U': 2, 'V': 8}
M.get('F', 5)	5	{'K': 9, 'B': 4, 'U': 2, 'V': 8}
M.get('K', 5)	9	{'K': 9, 'B': 4, 'U': 2, 'V': 8}
len(M)	4	{'K': 9, 'B': 4, 'U': 2, 'V': 8}
del M['V']	–	{'K': 9, 'B': 4, 'U': 2}
M.pop('K')	9	{'B': 4, 'U': 2}
M.keys()	'B', 'U'	{'B': 4, 'U': 2}
M.values()	4, 2	{'B': 4, 'U': 2}
M.items()	('B', 4), ('U', 2)	{'B': 4, 'U': 2}
M.setdefault('B', 1)	4	{'B': 4, 'U': 2}

Exercise

- Write a function input a string, output the most frequency words.

Example: “The man the woman, the man, man” -> the, man

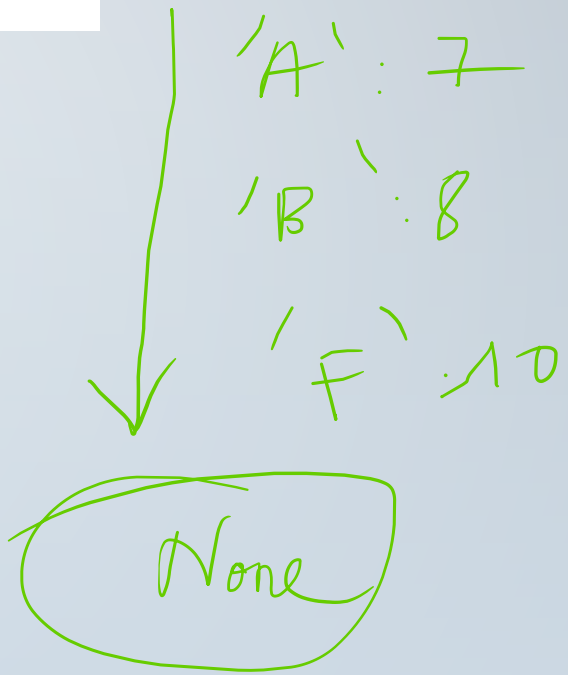
Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one— and preferably only one —obvious way to do it.[a]
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.[b]
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea—let's do more of those!

Dict's item

M['B']	4

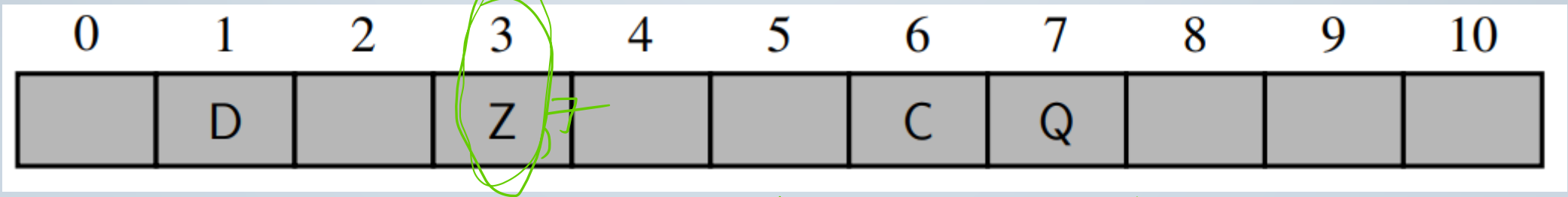
How to get item from the dict?

⇒ Iterate through all key, value pairs.



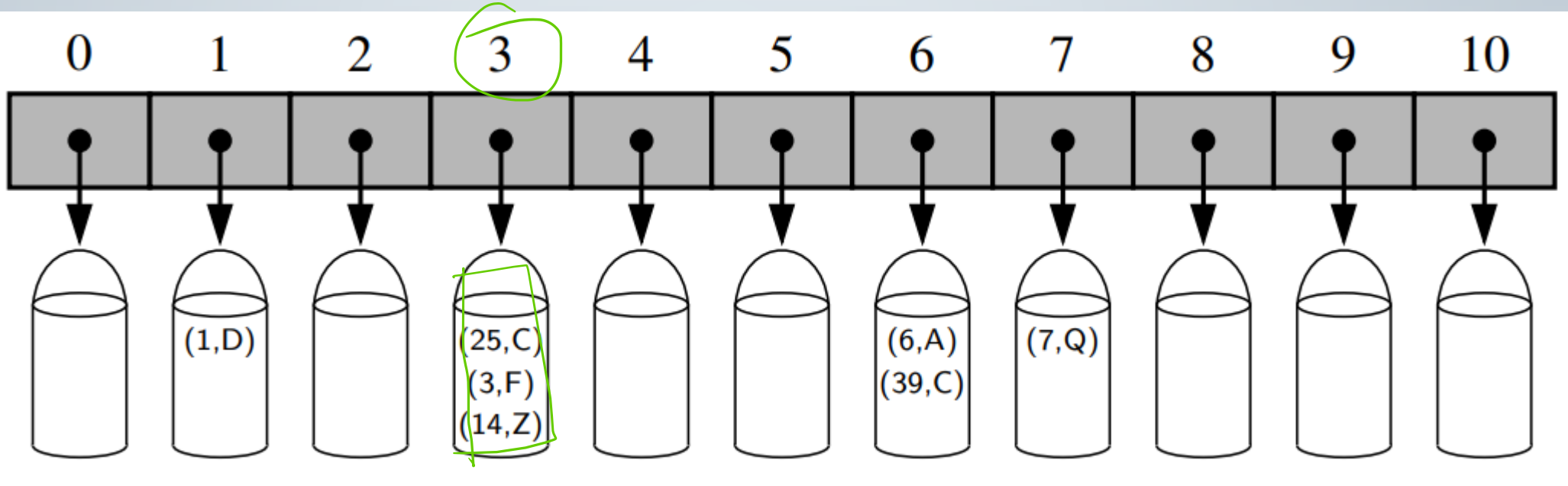
Hash functions

- The goal of a hash function, h, is to map each key k to an integer in the range [0,N-1], where N is the capacity of the bucket array for a hash table.
- The hash function value h(k), as an index into our bucket array, A, instead of the key k.
- We store the item (k,v) in the bucket A[h(k)].



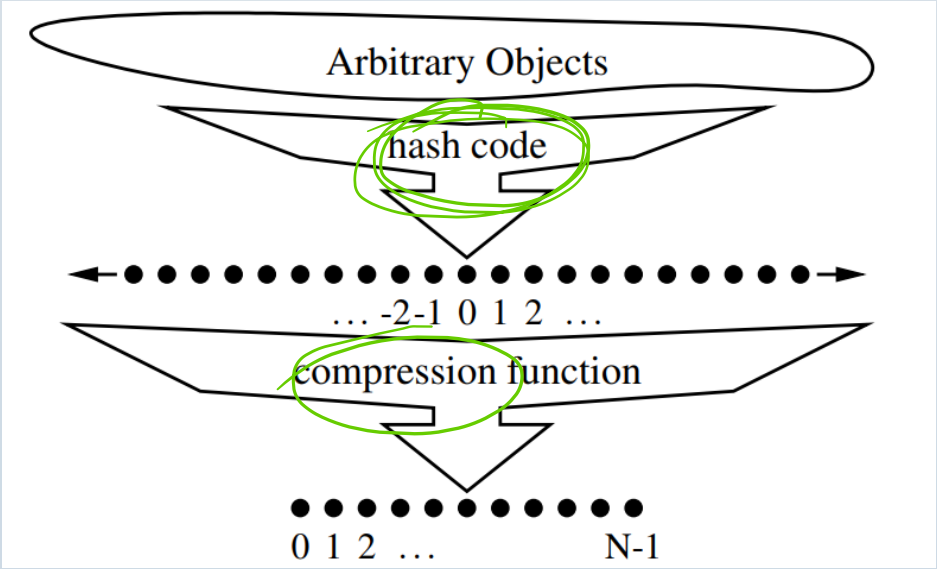
Hash Functions (1)

If there are two or more keys with the same hash value, then two different items will be mapped to the same bucket in A => collision has occurred.



Hash functions

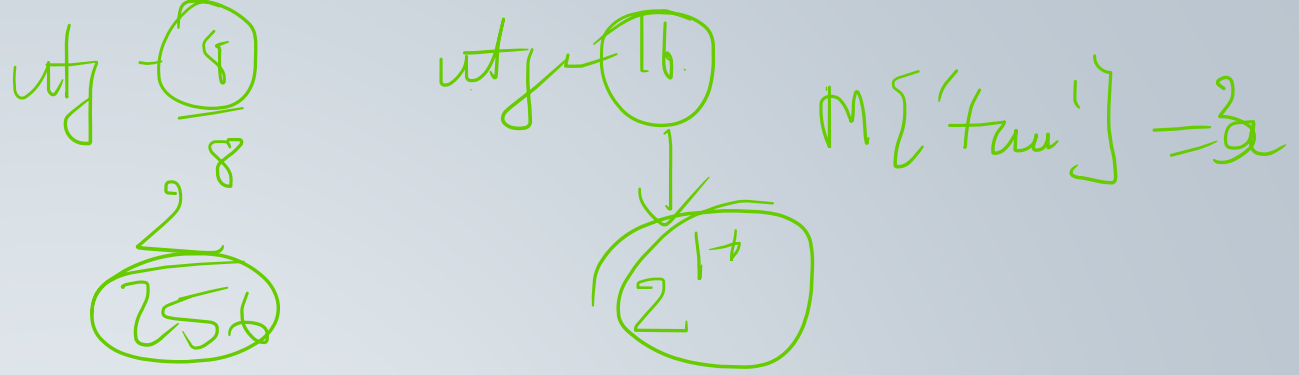
It is common to view the evaluation of a hash function, h(k), as consisting of two portions—a hash code that maps a key k to an integer, and a compression function that maps the hash code to an integer within a range of indices, [0,N −1], for a bucket array



Hash Codes

- Hash function performs is to take an arbitrary key k in our map and compute an integer that is called the hash code for k.
- Hash function performs is to take an arbitrary key k in our map and compute an integer that is called the hash code for k.
- Set of hash codes assigned to our keys should avoid collisions as much as possible.

Polynomial Hash Codes



- Character strings or other variable-length objects that can be viewed as tuples of the form (x0,x1,...,xn−1), where the order of the xi's is significant.
- Consider a 16-bit hash code for a character string s that sums the Unicode values of the characters in s

```
def hash_code(name):
    total = 0
    for i in name:
        total += ord(i)
    return total
```

Polynomial Hash Codes

- This hash code unfortunately produces lots of unwanted collisions for common groups of strings, example: stop, tops, pots, spot.
- A better hash code should somehow take into consideration the positions of the xi

$$x_0 a^{n-1} + x_1 a^{n-2} + \cdots + x_{n-2} a + x_{n-1}.$$

- In a list of over 50,000 English words formed as the union of the word lists provided in two variants of Unix, 33, 37, 39, or 41 are produced less than 7 collisions in each case

Cyclic-Shift Hash Codes

- A variant of the polynomial hash code replaces multiplication by a with a cyclic shift of a partial sum by a certain number of bits.
- For example, a 5-bit cyclic shift of the 32-bit value,
  00111011001011010101000101010000
- 10110010110101010001010100000111
- It accomplishes the goal of varying the bits of the calculation.

Collisions

230,000 English words

Shift	Collisions	
	Total	Max
0	234735	623
1	165076	43
2	38471	13
3	7174	5
4	1379	3
5	190	3
6	502	2
7	560	2
8	5546	4
9	393	3
10	5194	5
11	11559	5
12	822	2
13	900	4
14	2001	4
15	19251	8
16	211781	37

Hash Codes in Python

- Function hash(x) that returns an integer value that serves as the hash code for object x. Only immutable data types are deemed hashable in Python
- Hash of integer number is itself.
- Hash codes for character strings are well crafted based on a technique similar to polynomial hash codes.
- Hash codes for tuples are computed with a similar technique based upon a combination of the hash codes of the individual elements of the tuple.
- Change hash function of instances of user-defined classes.

```
def __hash__(self):
  return hash( (self._red, self._green, self._blue) )   # hash combined tuple
```

Compression Function

- The hash code for a key k will typically not be suitable for immediate use with a bucket array, because the integer hash code may be negative or may exceed the capacity of the bucket array.
- Integer hash code for a key object k, there is still the issue of mapping that integer into the range [0,N −1] => compression function.
- A good compression function is one that minimizes the number of collisions for a given set of distinct hash codes

The Division Method

- A simple compression function is the division method, which maps an integer i to **i mod N**. where N, the size of the bucket array.
- If N to be a **prime number**, then this compression function helps "spread out" the distribution of hashed values.
- For example, if we insert keys with hash codes {200,205,210,215,220,... ,600} into:
  - a bucket array of size 100, then each hash code will collide with three others.
  - a bucket array of size 101, then there will be no collisions.

The MAD Method

- The Multiply-Add-and-Divide (or "MAD") method maps an integer i to:

$$[(ai + b) \bmod p] \bmod N$$

where N is the size of the bucket array, p is a prime number larger than N, and a and b are integers chosen at random from the interval [0, p− 1], with a > 0.

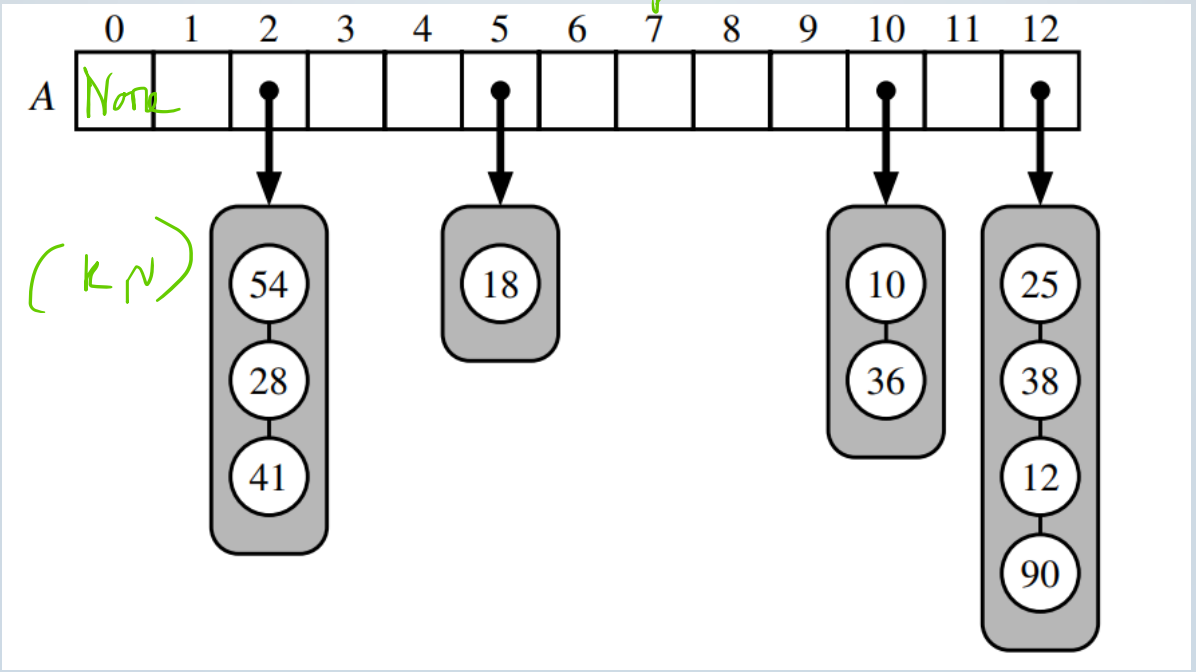
- This compression function is chosen in order to eliminate repeated patterns in the set of hash codes and get us closer to having a "good" hash function.

Collision-Handling Schemes

- Hash table is to take a bucket array, A, and a hash function, h, and use them to implement a map by storing each item (k,v) in the "bucket" A[h(k)].
- When we have two distinct keys, k1 and k2, such that h(k1) = h(k2) => collisions

Separate Chaining

- A simple and efficient way for dealing with collisions is to have each bucket A[j] store its own secondary container, holding items (k,v) such that h(k) = j



Linear Probing and Its Variants

- If we try to insert an item (k,v) into a bucket A[j] that is already occupied, where j = h(k).
- Then we next try A[(j + 1) mod N].
- If A[(j + 1) mod N] is also occupied, then we try A[(j + 2) mod N], and so on, until we find an empty bucket that can accept the new item