CHAPTER 1: DATA STRUCTURES AND ALGORITHMS <P3>

1.7. Keeping Dictionaries in Order

Problem:

Bạn muốn tạo một dictionary, và bạn cũng muốn sắp xếp thứ tự của các items khi lặp hoặc khi tuần tự hoá nó.

Solution:

Để điều chỉnh được thứ tự của các items trong một dictionary, bạn có thể sử dụng một OrderedDict từ module collections. Nó giữ nguyên đúng thứ tự ban đầu khi lặp.
Ví dụ:

```
from collections import OrderedDict

d = OrderedDict()
d['foo'] = 1
d['bar'] = 2
d['spam'] = 3
d['grok'] = 4

# Outputs "foo 1", "bar 2", "spam 3", "grok 4"
for key in d:
    print(key, d[key])
```

Một OrderedDict có thể đặc biệt hữu ích khi bạn muốn xây dựng một mapping mà bạn có thể muốn tuần tự hoá nó sau đó hoặc mã hoá thành một đinhj dạng khác. Ví dụ, nếu bạn muốn điều chỉnh đúng thứ tự các trường trong một JSON coding, trước tiên hãy tạo dữ liệu trong một OrderedDict sẽ thực hiện thủ thuật:

```
>>> import json
>>> json.dumps(d)
'{"foo": 1, "bar": 2, "spam": 3, "grok": 4}'
>>>
```

Discussion:

Một OrderedDict trong nội bộ duy trì một danh sách liên kết đôi mà sắp xếp các keys theo thứ tự được chèn. Khi một item mới đầu tiên được thêm vào, nó được đặt ở cuối danh sách này. Việc tái chỉ định một khoá đang tồn tại không thay đổi thứ tự.
Nhận thấy rằng kích thước của một OrderedDict là nhiều hơn 2 lần so với dictionary thông thường do danh sách liên kết bổ sung được tạo. Vì vậy, nếu bạn định xây dựng một cấu trúc dữ liệu liên quan đến một số lượng lớn các instances OrderedDict (Ví dụ như đọc 100000 lines của một file CSV vào một danh sách OrderedDict), bạn cần nghiên cứu các yêu cầu của ứng dụng của bạn để xem xét nếu lợi ích của việc sử dụng một OrderedDict lớn hơn chi phí bổ sung bộ nhớ.

1.8. Calculating with Dictionaries

Problem:

Bạn muốn thực thi nhiều phép tính toán (ví dụ như giá trị lớn nhất, giá trị nhỏ nhất, sắp xếp…) trên một dictionary.

Solution:

Xem xét một dictionary maps tên cổ phiếu với giá:

```
prices = {
   'ACME': 45.23,
   'AAPL': 612.78,
   'IBM': 205.55,
   'HPQ': 37.20,
   'FB': 10.75
}
```

Để việc tính toán hữu ích trên dictionary, đảo ngược các keys và values của dictionary bằng cách sử dụng zip(). Ví dụ, đây là cách để tìm cổ phiếu có giá lớn nhất hoặc nhỏ nhất:

```
min_price = min(zip(prices.values(), prices.keys()))
# min_price is (10.75, 'FB')

max_price = max(zip(prices.values(), prices.keys()))
# max_price is (612.78, 'AAPL')
```

Tương tự để xếp hạng các data, sử dụng zip() với sorted(), giống như sau:

```
prices_sorted = sorted(zip(prices.values(), prices.keys()))
# prices_sorted is [(10.75, 'FB'), (37.2, 'HPQ'),
#                   (45.23, 'ACME'), (205.55, 'IBM'),
#                   (612.78, 'AAPL')]
```

Khi thực hiện những phép tính này, chú ý rằng zip() tạo một vòng lặp mà chỉ có thể sử dụng 1 lần. Ví dụ, đoạn code dưới đây là một lỗi:

```
prices_and_names = zip(prices.values(), prices.keys())
print(min(prices_and_names))   # OK
print(max(prices_and_names))   # ValueError: max() arg is an empty sequence
```

Discussion:

Nếu bạn cố gắng thực thi các phép toán trên một dictionary thông thường mà không đảo ngược keys và values, bạn sẽ thấy rằng chúng chỉ xử lý các keys, chứ không phải là đối với values.

Ví dụ:

```
min(prices)   # Returns 'AAPL'
max(prices)   # Returns 'IBM'
```

Kết quả này có thể không phải là những gì bạn mong muốn vì thực sự là bạn đang muốn thực thi các phép toán liên quan đến values của dictionary. Bạn cố gắng fix được điều này bằng cách sử dụng values() method của dictionary:

```
min(prices.values())  # Returns 10.75
max(prices.values())  # Returns 612.78
```

Nhưng thật không may, điều này thường không chính xác những gì bạn muốn. Ví dụ, bạn có thể muốn biết thông tin về tên cổ phiếu có giá cao nhất?

Bạn có thể lấy các keys tương ứng với giá trị min hoặc max nếu bạn cung cấp một key function cho min() và max(). Ví dụ:

```
min(prices, key=lambda k: prices[k])  # Returns 'FB'
max(prices, key=lambda k: prices[k])  # Returns 'AAPL'
```

Tuy nhiên, để lấy giá trị nhỏ nhất, bạn sẽ cần bổ sung một bước tra cứu. Ví dụ:

```
min_value = prices[min(prices, key=lambda k: prices[k])]
```

Giải pháp liên quan đến zip() giải quyết vấn đề bằng cách “inverting" dictionary thành chuỗi các cặp (value, key). Khi thực thi việc so sánh trên những bộ dữ liệu như vậy, phần tử value được so sánh trước, sau đó đến key. Điều này cho bạn hành động chính xác những gì bạn muốn và cho phép bạn reductions và sắp xếp để dễ dàng thực thi trên dictionary bằng cách sử dụng một câu lệnh đơn.

Nó nên được lưu ý rằng trong các phép toán liên quan đến các cặp (value, key), key sẽ được sử dụng để xem xét kết quả trong ví dụ nơi nhiều mục xảy ra có giá trị giống nhau. Ví dụ, trong phép tính giống như min() hoặc max(), key sẽ được so sánh để trả về kết quả nếu xảy ra duplicate values.
Ví dụ cụ thể:

```
>>> prices = { 'AAA' : 45.23, 'ZZZ': 45.23 }
>>> min(zip(prices.values(), prices.keys()))
(45.23, 'AAA')
>>> max(zip(prices.values(), prices.keys()))
(45.23, 'ZZZ')
>>>
```

1.9. Finding Commonalities in Two Dictionaries

Problem:

Bạn có 2 dicts và muốn tìm ra những điểm chung giữa chúng (ví dụ, keys giống, values giống, …)

Solution:

Xem xét 2 dicts:

```
a = {
  'x' : 1,
  'y' : 2,
  'z' : 3
}

b = {
  'w' : 10,
  'x' : 11,
  'y' : 2
}
```

Để tìm ra những gì mà 2 dicts có chung, đơn giản là thực thi chung các thao tác thiết lập bằng cách sử dụng các methods keys() và items().
Ví dụ:

```
# Find keys in common
a.keys() & b.keys()   # { 'x', 'y' }

# Find keys in a that are not in b
a.keys() - b.keys()   # { 'z' }

# Find (key,value) pairs in common
a.items() & b.items() # { ('y', 2) }
```

Những loại thao tác này cũng có thể được sử dụng để thay đổi hoặc lọc nội dung của dict. Ví dụ, giả sử bạn muốn tạo một dict mới với các keys đã được chọn đã bị xoá.
Dưới đây là một đoạn code đơn giản:

```
# Make a new dictionary with certain keys removed
c = {key:a[key] for key in a.keys() - {'z', 'w'}}
# c is {'x': 1, 'y': 2}
```

Discussion:

Một dict là một mapping giữa key và value. Phương thức keys() của dict trả về một đối tượng keys-view cho thấy các khoá. Một tính năng nhỏ ít được biết đến là chúng cũng hỗ trợ các thao tác thiết lập chung như unions, intersections,... Vì vậy, nếu bạn cần thực thi các thao tác thiết lập chung với các keys của dict, bạn có thể sử dụng đối tượng keys-view mà không cần convert chúng thành 1 set.

Phương thức items() trả về một đối tượng items-view bao gồm các cặp (key, value). Đối tượng này hộ trợ các thao tác thiết lập tương tự và có thể được sử dụng để thực thi các thao tác như tìm ra các cặp key-value chung của cả 2 dict.

Mặc dù tương tự, Phương thức values() không hỗ trợ các thao tác thiết lập mô tả trong recipe này. Một phần, điều này là do thực tế rằng các keys, các items không giống nhau được bao gồm trong một view các values không đảm bảo là duy nhất. Điều này làm cho các thao tác thiết lập chính có vấn đề, chúng có thể được hoàn thành đơn giản bằng cách chuyển đổi các values thành một bộ trước.