Hash: dictionaries for R

Liz Sander, Civis Analytics

August 23, 2017

What is a dictionary?

- Also known as a hashmap

What is a dictionary?

- Also known as a hashmap
- Set of key:value pairs

What is a dictionary?

```
library(hash)
```

```
## hash-2.2.6 provided by Decision Patterns
```

```
mydict <- hash('a'=1:3,
               'b'=letters[1:3],
               'c'=matrix(1:8,2,4))
print(mydict)
```

```
## <hash> containing 3 key-value pair(s).
##   a : 1 2 3
##   b : a b c
##   c : 1 2 3 4 5 6 7 8
```

Okay, so it's a named list?

```
mylist <- list('a'=1:3,
               'b'=letters[1:3],
               'c'=matrix(1:8,2,4))
mylist[['a']]
```

```
## [1] 1 2 3
```

```
mydict[['a']]
```

```
## [1] 1 2 3
```

Okay, so it's a named list?

```
mylist$a
```

```
## [1] 1 2 3
```

```
mydict$a
```

```
## [1] 1 2 3
```

```
mylist[[1]]
```

```
## [1] 1 2 3
```

```
# mydict[[1]]
```

Okay, so it's a named list?

- Dictionaries aren't ordered. You can only access values by their keys

Okay, so it's a named list?

- Dictionaries aren't ordered. You can only access values by their keys
- Trades known order for speed and convenience

Why dictionaries?

Easy to look things up

- Doesn't matter for the user

Easy to look things up

- Doesn't matter for the user
- Faster for the computer

Easy to look things up

- Doesn't matter for the user
- Faster for the computer
- With a list, the computer has to scan through the list in order

Easy to look things up

- Doesn't matter for the user
- Faster for the computer
- With a list, the computer has to scan through the list in order
  - (up to length(mylist) operations)

Easy to look things up

- Doesn't matter for the user
- Faster for the computer
- With a list, the computer has to scan through the list in order
  - (up to length(mylist) operations)
  - Like flipping page by page through a (physical) dictionary until you find the word you want

Easy to look things up

- Doesn't matter for the user
- Faster for the computer
- With a list, the computer has to scan through the list in order
  - (up to length(mylist) operations)
  - Like flipping page by page through a (physical) dictionary until you find the word you want
- With a dict, the computer runs an algorithm on the key to find where it put the value (only one operation)

Easy to look things up

- Doesn't matter for the user
- Faster for the computer
- With a list, the computer has to scan through the list in order
  - (up to length(mylist) operations)
  - Like flipping page by page through a (physical) dictionary until you find the word you want
- With a dict, the computer runs an algorithm on the key to find where it put the value (only one operation)
  - Like thinking for a second about where in the alphabet your word is, then flipping to the correct page right away

Easy to look things up

- This key:value mapping algorithm is called hashing!

Easy to look things up

- This key:value mapping algorithm is called hashing!
- Fun fact: Have you heard of hashing for passwords or cryptography? Same thing! Hashing is awesome :)

Easy to look things up

- This key:value mapping algorithm is called hashing!
- Fun fact: Have you heard of hashing for passwords or cryptography? Same thing! Hashing is awesome :)
- This computer memory stuff is nitty-gritty, but these details are what make dicts useful.

Easy to add things

- Doesn't matter for the user

```
mydict['d'] = 100
mylist['d'] = 100
```

Easy to add things

- Way easier for the computer!

Easy to add things

- Way easier for the computer!
- When you add something to the list, the computer has to:

Easy to add things

- Way easier for the computer!
- When you add something to the list, the computer has to:
  - make a new, slightly bigger list

Easy to add things

- Way easier for the computer!
- When you add something to the list, the computer has to:
  - make a new, slightly bigger list
  - copy the old list over

Easy to add things

- Way easier for the computer!
- When you add something to the list, the computer has to:
  - make a new, slightly bigger list
  - copy the old list over
  - tack your new thing on at the end

Easy to add things

- Way easier for the computer!
- When you add something to the list, the computer has to:
  - make a new, slightly bigger list
  - copy the old list over
  - tack your new thing on at the end
- The computer only makes as much space for the list as it needs right then

Easy to add things

- Way easier for the computer!
- When you add something to the list, the computer has to:
  - make a new, slightly bigger list
  - copy the old list over
  - tack your new thing on at the end
- The computer only makes as much space for the list as it needs right then
  - Makes a perfect size box, and fills it up all of the way.

Easy to add things

- Way easier for the computer!
- When you add something to the list, the computer has to:
  - make a new, slightly bigger list
  - copy the old list over
  - tack your new thing on at the end
- The computer only makes as much space for the list as it needs right then
  - Makes a perfect size box, and fills it up all of the way.
- A dictionary is more like a cabinet with lots of drawers, some of them empty

Easy to add things

- Way easier for the computer!
- When you add something to the list, the computer has to:
  - make a new, slightly bigger list
  - copy the old list over
  - tack your new thing on at the end
- The computer only makes as much space for the list as it needs right then
  - Makes a perfect size box, and fills it up all of the way.
- A dictionary is more like a cabinet with lots of drawers, some of them empty
  - Adding a new thing just means finding an empty drawer

Easy to remove things

- Easy for the user (although not identical)

```
mylist = mylist[names(mylist) != 'b']
print(names(mylist))
```

```
## [1] "a" "c"
```

```
del('b', mydict) # no need to reassign
print(names(mydict))
```

```
## [1] "a" "c"
```

Easy to remove things

- With lists, you may have to scan through the whole list/create a whole new list, depending on method

Easy to remove things

- With lists, you may have to scan through the whole list/create a whole new list, depending on method
- Dicts just take the item out of the drawer and leave the drawer empty

When should I use dicts?

Lists are great for:

- using lapply()

Dicts are great for:

When should I use dicts?

Lists are great for:

- using lapply()
- keeping things in a specific order

Dicts are great for:

When should I use dicts?

Lists are great for:

- using lapply()
- keeping things in a specific order

Dicts are great for:

- Speedy access, addition, and deletion of elements

When should I use dicts?

Lists are great for:

- using lapply()
- keeping things in a specific order

Dicts are great for:

- Speedy access, addition, and deletion of elements
- Variables you use and change a lot

When should I use dicts?

Lists are great for:

- using lapply()
- keeping things in a specific order

Dicts are great for:

- Speedy access, addition, and deletion of elements
- Variables you use and change a lot
- Data where order doesn't matter

When should I use dicts?

- keeping track of counts (e.g., counting occurrences of words in text)

When should I use dicts?

- keeping track of counts (e.g., counting occurrences of words in text)
- lookup table to store results from slow calculations, so you don't have to calculate more than once

When should I use dicts?

- keeping track of counts (e.g., counting occurrences of words in text)
- lookup table to store results from slow calculations, so you don't have to calculate more than once
- probably most places you'd otherwise use a list

When should I use dicts?

- keeping track of counts (e.g., counting occurrences of words in text)
- lookup table to store results from slow calculations, so you don't have to calculate more than once
- probably most places you'd otherwise use a list
- nested dictionaries are a thing (think list of lists, but faster)

Code Example!