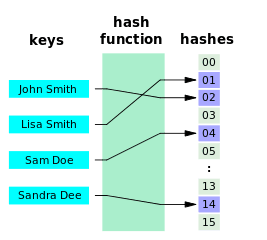
We are living in the era of Big Data but the problem of course is that the bigger our data sets become the slower even simple search operations get. I will now show you a trick that is the next best thing to magic: building a search function that practically doesn’t slow down even for large data sets… in base R!

On first thought this is totally counterintuitive: the bigger the data set is, the longer it should take to search it, right? Wrong!

The data structure we will be talking about is called a hash or a dictionary (sometimes also called associated array). The big idea is to use a mathematical function (called hash function) which maps each data item (e.g. a name) to an address (called hash) where the corresponding value (e.g. a telephone number) is stored. So to find the telephone number for a certain name you don’t have to search through all the names but you just put it into the hash function and you get back the address for the telephone number instantaneously:

Source: wikimedia

This image is from the very good wikipedia article on hash function algorithms: [Hash function](https://en.wikipedia.org/wiki/Hash_function). It also gives a trivial example of a hash function to get the idea:

If the data to be hashed is small enough, one can use the data itself (reinterpreted as an integer) as the hashed value. The cost of computing this “trivial” (identity) hash function is effectively zero. This hash function is perfect, as it maps each input to a distinct hash value.

To do this with R we use environments with the option hash = TRUE. The following example is from an answer I gave on stackoverflow (<https://stackoverflow.com/a/42350672/468305>):

# vectorize assign, get and exists for convenience

assign\_hash <- Vectorize(assign, vectorize.args = c("x", "value"))

get\_hash <- Vectorize(get, vectorize.args = "x")

exists\_hash <- Vectorize(exists, vectorize.args = "x")

# keys and values

key <- c("tic", "tac", "toe")

value <- c(1, 22, 333)

# initialize hash

hash <- new.env(hash = TRUE, parent = emptyenv(), size = 100L)

# assign values to keys

assign\_hash(key, value, hash)

## tic tac toe

## 1 22 333

# get values for keys

get\_hash(c("toe", "tic"), hash)

## toe tic

## 333 1

# alternatively:

mget(c("toe", "tic"), hash)

## $toe

## [1] 333

##

## $tic

## [1] 1

# show all keys

ls(hash)

## [1] "tac" "tic" "toe"

# show all keys with values

get\_hash(ls(hash), hash)

## tac tic toe

## 22 1 333

# remove key-value pairs

rm(list = c("toe", "tic"), envir = hash)

get\_hash(ls(hash), hash)

## tac

## 22

# check if keys are in hash

exists\_hash(c("tac", "nothere"), hash)

## tac nothere

## TRUE FALSE

# for single keys this is also possible:

# show value for single key

hash[["tac"]]

## [1] 22

# create new key-value pair

hash[["test"]] <- 1234

get\_hash(ls(hash), hash)

## tac test

## 22 1234

# update single value

hash[["test"]] <- 54321

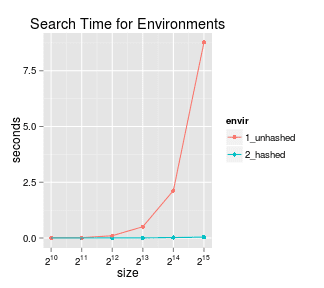
get\_hash(ls(hash), hash)

## tac test

## 22 54321

So you see that using the inbuilt hash functionality is quite simple. To get an idea of the performance boost there is a very thorough article here: <http://jeffreyhorner.tumblr.com/post/114524915928/hash-table-performance-in-r-part-i>.

Have a look at the following plot from the article:



Horner writes:

**Bam!** See that blue line? That’s near constant time for searching the entire 2^{15}size hash table!

If I could whet your appetite I want to close with a pointer to a much more professional implementation of hash tables using environments: [https://CRAN.R-project.org/package=hash](https://cran.r-project.org/package=hash)

I haven’t tried the package myself so far but the author, Christopher Brown, promises:

The hash package is the only full featured hash implementation for the R language. It provides more features and finer control of the hash behavior than the native feature set and has similar and sometimes better performance.

If you use this package (or any other hash implementation) I would love to read about your experience in the comments!