Week 3 Exercises

Ray Chandonnet

November 11, 2022

Please complete all exercises below. You may use any library that we have covered in class UP TO THIS POINT.

1) Two Sum - Write a function named two_sum()

Given a vector of integers nums and an integer target, return indices of the two numbers such that they add up to target.

You may assume that each input would have exactly one solution, and you may not use the same element twice.

You can return the answer in any order.

Example 1:

Input: nums = [2,7,11,15], target = 9 Output: [0,1] Explanation: Because nums[0] + nums[1] == 9, we return [0, 1]. Example 2:

Input: nums = [3,2,4], target = 6 Output: [1,2] Example 3:

Input: nums = [3,3], target = 6 Output: [0,1]

Constraints:

2 <= nums.length <= 104 –109 <= nums[i] <= 109 –109 <= target <= 109 Only one valid answer exists.

Note: For the first problem I want you to use a brute force approach (loop inside a loop)

The brute force approach is simple. Loop through each element x and find if there is another value that equals to target – x

Use the function seq_along to iterate

```
two_sum <- function(nums_vector,target) {
  # This function takes in a vector of values and a target value, and determines which
  # combination of any two separate values in the vector addup to the target value
  #
  # Inputs: nums_vector = The vector of values to be searched
  #         target      = The target sum being sought
  #
  # Output: The function returns a 2 column dataframe of values corresponding to each set
  #         of two indexes whose values add up to target
  #
  # Argument Constraints: 2 <= nums.length <= 104
  #                       -109 <= nums[i] <= 109
  #                       -109 <= target <= 109
  #
  # First, check to make sure all arguments are compliant with the rules.  Display error
  # messages for each so user knows which constraint(s) triggered
  #
  argumentsOK <- TRUE
  if (length(nums_vector) < 2){
    print("ERROR: Number vector must have at least two values")
    argumentsOK = FALSE}
```

```
  if (length(nums_vector)>104) {
    print("ERROR: Number vector cannot exceed 104 values")
    argumentsOK = FALSE}
  if (min(nums_vector) < -109) {
    print("ERROR: All values in vector must be >= -109")
    argumentsOK = FALSE}
  if (max(nums_vector) > 109) {
    print("ERROR: All values in vector must be <= 109")
    argumentsOK = FALSE}
  if (target < -109) {
    print("ERROR: Target sum must be >= -109")
    argumentsOK = FALSE}
  if (target > 109) {
    print("ERROR: Target sum must <= 109")
    argumentsOK = FALSE}
  # If any error was triggered, abort the function
  if (!argumentsOK) {
    print("Function Aborted Due to bad argument(s) above")
    return(NULL)    # Note that I am returning a null value rather than using stop, because
                    # stop cancels the whole script and doesn't produce any output in the
                    # knitting process :)  In real life production I would break the script
  }
  # If we get to here, all our parameters are compliant, so do the processing
  #
  matches = data.frame(matrix(ncol = 2,nrow=0))  # Create null 2-column dataframe to store
  #                                                 results; will be returned as by function
  colnames(matches) <- c("Index 1", "Index 2") # Set Column names
  total_matches = 0 # Counts matches found, and used to add rows to the matches dataframe
  index_vector <- seq_along(nums_vector) # fancy way to iterate; Could also use For i=1 to
  #                                        length(nums_vector)
  for (counter1 in index_vector) { #loop through first index
    for (counter2 in index_vector) { # loop through second index
# Note that we need to make sure we don't doublecount values by ignoring when counters match
      if (counter1!=counter2 & nums_vector[counter2]==target-nums_vector[counter1]) {
        total_matches <- total_matches + 1 #found a match
        matches[total_matches,] <- c(counter1,counter2)} #add row with the matching indexes
      }#end of counter 2 For loop
  } #end of counter 1 For loop
  return(matches)
}  # end function
# Test code
# These are just to prove that my error checking inside the function works
two_sum(1,2) #nums_vector too short
```

```
## [1] "ERROR: Number vector must have at least two values"
## [1] "Function Aborted Due to bad argument(s) above"
```

```
## NULL
```

```
two_sum(1:105,2) #nums_vector too long
```

```
## [1] "ERROR: Number vector cannot exceed 104 values"
## [1] "Function Aborted Due to bad argument(s) above"
```

```
## NULL
```

```
two_sum(-110:-100,2) #lowest nums_vector value not in bounds
```

```
## [1] "ERROR: All values in vector must be >= -109"
## [1] "Function Aborted Due to bad argument(s) above"
```

```
## NULL
```

```
two_sum(100:110,2) #highest nums_vector value not in bounds
```

```
## [1] "ERROR: All values in vector must be <= 109"
## [1] "Function Aborted Due to bad argument(s) above"
```

```
## NULL
```

```
two_sum(1:100, -110) #target < lower bound
```

```
## [1] "ERROR: Target sum must be >= -109"
## [1] "Function Aborted Due to bad argument(s) above"
```

```
## NULL
```

```
two_sum(1:100,110) #target > upper bound
```

```
## [1] "ERROR: Target sum must <= 109"
## [1] "Function Aborted Due to bad argument(s) above"
```

```
## NULL
```

```
two_sum(-120:120,200) #multiple invalid inputs (lower/upper nums, length of nums, target)
```

```
## [1] "ERROR: Number vector cannot exceed 104 values"
## [1] "ERROR: All values in vector must be >= -109"
## [1] "ERROR: All values in vector must be <= 109"
## [1] "ERROR: Target sum must <= 109"
## [1] "Function Aborted Due to bad argument(s) above"
```

```
## NULL
```

```
# Now we actually test with valid arguments
nums_vector <- c(5,7,12,34,6,10,8,9)
target <- 13
two_sum(nums_vector,target)
```

```
##   Index 1 Index 2
## 1       1       7
## 2       2       5
## 3       5       2
## 4       7       1
```

2) Now write the same function using hash tables. Loop the array once to make a hash map of the value to its index. Then loop again to find if the value of target-current value is in the map.

The keys of your hash table should be each of the numbers in the nums_vector minus the target.

A simple implementation uses two iterations. In the first iteration, we add each element's value as a key and its index as a value to the hash table. Then, in the second iteration, we check if each element's complement (target – nums_vector[i]) exists in the hash table. If it does exist, we return current element's index and its complement's index. Beware that the complement must not be nums_vector[i] itself!

```
library(hash)
```

```
## hash-2.2.6.2 provided by Decision Patterns
```

```
two_sum <- function(nums_vector,target) {
  # This function takes in a vector of values and a target value, and determines which
  # combination of any two separate values in the vector addup to the target value
  #
  # It does so using a hashtable in order to minimize the amount
  # of work that has to be done to get the answer
  #
  # Inputs: nums_vector = The vector of values to be searched
  #         target      = The target sum being sought
  #
  # Output: The function returns a 2 column dataframe of values corresponding to each set
  #         of two indexes whose values add up to target
  #
  # Argument Constraints: 2 <= nums.length <= 104
  #                       -109 <= nums[i] <= 109
  #                       -109 <= target <= 109
  #
  # First, check to make sure all arguments are compliant with the rules.  Display error
  # messages for each so user knows which constraint(s) triggered
  argumentsOK <- TRUE
  if (length(nums_vector) < 2){
    print("ERROR: Number vector must have at least two values")
    argumentsOK = FALSE}
  if (length(nums_vector)>104) {
    print("ERROR: Number vector cannot exceed 104 values")
    argumentsOK = FALSE}
  if (min(nums_vector) < -109) {
    print("ERROR: All values in vector must be >= -109")
    argumentsOK = FALSE}
  if (max(nums_vector) > 109) {
    print("ERROR: All values in vector must be <= 109")
    argumentsOK = FALSE}
  if (target < -109) {
    print("ERROR: Target sum must be >= -109")
    argumentsOK = FALSE}
  if (target > 109) {
    print("ERROR: Target sum must <= 109")
    argumentsOK = FALSE}
  # If any error was triggered, abort the function
  if (!argumentsOK) {
    print("Function Aborted Due to bad argument(s) above")
    return(NULL)
  }  # Abort the function
  #
  # If we get to here, all our parameters are compliant, so do the processing
  #
  total_matches <- 0
  matches = data.frame(matrix(ncol = 2,nrow=0))
  colnames(matches) <- c("Index1", "Index2")
  hash_table=hash()

  # First, create a hash table that stores (as keys) each value in the nums_vector. Store
  # the counter as the hash value since it represents the index of the value
```

```
  for (counter1 in 1:length(nums_vector)) {
    hash_table[nums_vector[counter1]] <- counter1}

  # Now loop through the values again, searching for each value's "complement" (the number
  # which, when added to the selected value, equals the target sum) in the hash keys.
  # If it is found, make sure you're not double-counting a single value, and if not,
  # then a solution pair has been found, so add it to the "match" solution dataframe
  # the function will return

  for (counter2 in 1:length(nums_vector)) {
    search_key <- as.character(target-nums_vector[counter2]) # create text-based search key,
                                                             # since hash keys=characters
    if (has.key(search_key,hash_table)) {  # Check if complement exists
      if (hash_table[[search_key]]!=counter2) { # Check for double-counting

        # Add a row to our matches table which contains the two indexes - the first is our
        # counter, the second is the value in the hash_table

        total_matches <- total_matches + 1
        matches[total_matches,] <- c(counter2, hash_table[[search_key]])
      } # end of 2nd if
    } # end of 1st if
  } # end of for loop
return(matches)
} # end of function
# test code (if you don't mind, I didn't retest my error handling since it's identical to
# the error handling in the first exercise)

nums_vector <- c(5,7,12,34,6,10,8,9)
target <- 15
two_sum(nums_vector,target)
```

```
##   Index1 Index2
## 1      1      6
## 2      2      7
## 3      5      8
## 4      6      1
## 5      7      2
## 6      8      5
```

Ray note: Depending on the practical application of this, you might expect a different answer. Namely, if the question is "For each value, tell me if there is another value in the vector which, when added together, equates to target", what I coded above works. However, a more practical question might be "please give me all of the index-pairs whose values add up to target". In this case, [1,6] is the same pair of indexes as [6,1], meaning my original approach produced duplicate results for every pair. Given that, I took the liberty of rewriting my two functions above (without all the error checking, for brevity) so that they only return UNIQUE index pairs whose values add up to target. Code is below.

```
# This first version uses the "brute force" double loop method.  Avoiding duplicate pairs
# along with double-counting is actually quite simple, just requires a few tweaks to
# the for loops (as follows) and actually reduces processing time / work:
#
# 1) Since we don't want duplicate pairs, we should never include pairs where the second
#    index (found in nested loop 2) is LESS than the first index! Why? Because all those
```

```
#    pairs were examined earlier in the run using the lower index first
#
# 2) We also never want to add a value to itself
#
# 3) This means two things:
#   a) Our interior loop should be shortened, to start at the exterior loop counter + 1;
#      This eliminates all pairs already considered in reverse earlier in the loop process
#      (When we have already evaluated [1 4],[2 4],[3 4] there is no reason to consider
#       [4 1],[4 2],[4 3] because they've already been evaluated in reverse). This also
#      eliminates adding values to themselves because we won't consider [1 1], [2 2], etc
#   b) Our external loop can ignore the very last value, because it will have been
#      considered in every possible pairing combination already as the "second index"
#
two_sum <- function(nums_vector,target) {
  matches = data.frame(matrix(ncol = 2,nrow=0))  # Create null 2-column dataframe to store
                                                 # results; will be returned as by function
  colnames(matches) <- c("Index 1", "Index 2") # Set Column names
  total_matches = 0 # Counts matches found, and used to add rows to the matches dataframe
  outside_loop_vector <- 1:(length(nums_vector)-1)  # Loop 1 ends at second to last value
  for (counter1 in outside_loop_vector) { #loop through first index
    inside_loop_vector <- (counter1+1):length(nums_vector) # Loop 2 starts at index after
                                                           # counter for loop 1 (so it gets
                                                           # shorter with each cycle), and
                                                           # goes to the end
    for (counter2 in inside_loop_vector) { # loop through second index
      if (nums_vector[counter2]==target-nums_vector[counter1]) {
        total_matches <- total_matches + 1 #found a match
        matches[total_matches,] <- c(counter1,counter2) #add row with the matching indexes
      } # end if
    } #end of counter 2 For loop
  } #end of counter 1 For loop
  return(matches)
} # end function
# Test code
nums_vector <- c(5,7,12,34,6,10,8,9)
target <- 13
two_sum(nums_vector,target)
```

```
##   Index 1 Index 2
## 1       1       7
## 2       2       5
```

```
nums_vector <- c(5,7,12,34,6,10,8,9)
target <- 15
two_sum(nums_vector,target)
```

```
##   Index 1 Index 2
## 1       1       6
## 2       2       7
## 3       5       8
```

```
# This second version uses hashing. The key tweaks here are analagous to what I did in the
# brute force method.  If I find a value's complement in the hash keys, I will only consider
# that index pair valid and add to my matches dataframe if the index value corresponding to
# that matching key is greater than the index of the value I'm considering. This, along with
```

```
# ending the second for loop 1 value early, will effectively do the same thing as I did in
# the function above, just using hashing for greater efficiency

two_sum <- function(nums_vector,target) {
  total_matches <- 0
  matches = data.frame(matrix(ncol = 2,nrow=0))
  colnames(matches) <- c("Index1", "Index2")
  hash_table=hash()

  # First, create a hash table that stores (as keys) each value in the nums_vector. Store
  # the counter as the hash value since it is the index of the value inside the vector

  for (counter1 in 1:length(nums_vector)) {
    hash_table[nums_vector[counter1]] <- counter1} #for loop

  # Now loop through the values again, searching for each value's "complement" (the number
  # which, when added to the selected value, equals the target sum) in the hash keys.
  # Only consider it a valid match if the index value in the hash table > the index of
  # the value we are considering in nums_vector, represented as counter 2

  loop_vector <- 1:(length(nums_vector)-1)  # Loop 1 ends at second to last value
  for (counter2 in loop_vector) {
    search_key <- as.character(target-nums_vector[counter2]) # create text-based search key
    if (has.key(search_key,hash_table)) {  # Check if complement exists
      if (hash_table[[search_key]] > counter2) { # Check for validity (no dups!)
          index_pair <- c(counter2, hash_table[[search_key]])
          total_matches <- total_matches+1
          matches[total_matches,] <- index_pair  # insert the new pair in matches
      } # end of 1st if
    } # end of 2nd if
  } # end for loop
  return(matches)
} #end function
# test code
nums_vector <- c(5,7,12,34,6,10,8,9)
target <- 13
two_sum(nums_vector,target)
```

```
##   Index1 Index2
## 1      1      7
## 2      2      5
```

```
nums_vector <- c(5,7,12,34,6,10,8,9)
target <- 15
two_sum(nums_vector,target)
```

```
##   Index1 Index2
## 1      1      6
## 2      2      7
## 3      5      8
```