Week 3 Exercises

Steven Simonsen

March 25, 2024

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 In the function below, I debated whether to use "<" or "!=" in my if statment. I ultimately went with "<" because the problem stated "you may not use the same element twice."

```
library(purrr)

nums_vector <- c(5,7,12,34,6,10,8,9)
target <- 13

#Assign initial loop_work variable to 0 for comparison to hashloop later
loop_work <- 0

two_sum <- function(nums_vector,target){
  #Empty character vector assigned to variable loop vector for now
  loop_vector <- character()
  #nested for statements using
  #seq_along to iterate over the num_vector.
    for(i in seq_along(nums_vector)) {
      loop_work <<- loop_work + 1
```

```
      for(j in seq_along(nums_vector)) {
      #if the sum of element [i] and element [j] equal the target
      #and aren't the same number (could also use != to return
      #the reverse of the result),
      #concatenate and output the indices on a line
        loop_work <<- loop_work + 1
        if(i < j && nums_vector[i] + nums_vector[j] == target) {
          loop_vector <- c(loop_vector, paste(i, j))
      }
    }
    }
  #Return used outside of the for loop, but within function!
  return(loop_vector)
}
#run the function by assigning result to the function
result <- two_sum(nums_vector, target)
#print the result
print(result)
```

```
## [1] "1 7" "2 5"
```

```
# Test code
#expected answers
#[1] 1 7
#[1] 2 5
#[1] 5 2
```

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!

In this function, I again used "<" instead of "!=" in the if statement for the same reason as above. Additionally, I returned the indices after reading "If it does exist, we return current element's index and its complement's index." Therefore, my answers don't match the expected answers - hopefully that's okay!

```
library(hash)
```

```
## hash-2.2.6.3 provided by Decision Patterns
```

```
nums_vector2 <- c(5,7,12,34,6,10,8,9)
target2 <- 15

#Assign hash_work variable to 0
hash_work <- 0


two_sum2 <- function(nums_vector2,target2){
  #Assign h to an empty hash map
  h<-hash()
  #As noted above, assign each element's value as a key and its index as a value to   the hash table.
  for(i in seq_along(nums_vector2)) {
```

```
    #Add 1 to hash_work for comparison to loop work at end. Use <<- global
    #assignment operator to update the variable outside of the function
    hash_work <<- hash_work + 1
    h[(nums_vector2[i])] <- i

  }
  #Empty character vector for iteration #2
  result2 <- character()
  for(i in seq_along(nums_vector2)) {
    #Add again to hash_work
    hash_work <<- hash_work + 1
    #As noted above, complement is equal to target less the nums_vector[i]
    complement <- target2 - nums_vector2[i]
    #The first part of the if statement checks to make sure the complement is
    #less than nums_vector2[i] to avoid duplication.
    #The second part of the if statement checks to see if the complement is
    #contained within the keys (names) of nums_vector2[i]
    if(complement < nums_vector2[i] && complement %in% names(h)) {
      #If found, append to result2 vector with the value (indices) of the
      #complement, and the index of the original nums_vector2 that corresponds
      #with the value whos sum is equal to the target.
      #as.character is needed because R converts keys to a character type.
      #Since our values are numeric, I converted to character to ensure keys are
      #treated as characters. Before adding this, I was getting very
      #strange,unexpected errors.
      result2 <- c(result2, paste(h[[as.character(complement)]], i))
    }
  }
  #I made sure to use return OUTSIDE of the for loops, but still within function!
  return(result2)
}

#Assign result2 to running the function
result2 <- two_sum2(nums_vector2, target2)
#Print the result2
print(result2)
```

```
## [1] "1 6" "2 7" "5 8"
```

```
#comparison of hashwork vs loopwork. Much more efficient to use hash!
print(hash_work)
```

```
## [1] 16
```

```
print(loop_work)
```

```
## [1] 72
```

```
#expected answers
#[1] 10  5
#[1] 8 7
#[1] 9 6
#[1]  5 10
#[1] 7 8
#[1] 6 9
```