Lesson 18:

1. Introduction

DuckDuckGo says our search engine is too slow.

1. Making Things Fast

Algorithm Analysis

Procedure = well-defined sequence of steps that can be executed mechanically, and is guaranteed to always finish and produce the correct result.

Cost = algorithm time based on input size

1. Quiz: Measuring Speed

Predict how long it will take for a program to execute.

Want to know how the time will change as computers get faster.

We want to understand fundamental properties of our algorithms, not things specific to a particular input or machine.

1. Stopwatch

time.clock() # processor time in seconds

1. Spin Loop

Simple loop to understand how time increases based off input.

1. Quiz: Predicting Run Time

Guess ~52 seconds. Actual 57.168562 seconds.

1. Make Big Index

lookup(table\_data, item\_to\_find)

1. Quiz: Index Size Vs. Time

10,000,000 keywords = ~1 second for lookup

1. Quiz: Lookup Time
2. seconds since first index made is ‘aaaaaaaa’

input and type of input can affect speed

1. Quiz: Worst Case

Word that is not in index and last word added = worst run times.

1. Quiz: Fast Enough

It depends on how many keywords there are.

It depends on how many lookups there are.

1. Making Lookup Faster

Hashing allow you to look in a list for the right value.

Hash Table = Data Structure = Dictionary

1. Quiz: Hash Table

k/b items in each index

1. Hash Function

ord(‘a’) -> number

chr(ord(‘a’)) -> a

1. Modulus Operator

%

14 % 12 = 2

Remainder

1. Quiz: Modulus Quiz

12 % 3 = 0

ord(‘a’) % ord(‘a’) = 0

(ord(‘z’) + 3) % ord(‘z’) = 3

1. Quiz: Equivalent Expressions

x % 23

ord(chr(x))

1. Quiz: Bad Hash

Error, some buckets are fuller, if too many buckets, possible for empty buckets.

1. Quiz: Better Hash Functions

def hash\_string(keyword, buckets):

hash\_value = 0

for letter in keyword:

hash\_value += ord(letter) # add up the ordinal value of all letters in keyword

hash\_value %= buckets # find remainder of hash\_value with respect to buckets

return hash\_value

1. Testing Hash Functions

The new function we made is better because we more evenly distribute words to buckets, however since the function now loops it might take longer to initially deposit items into the corresponding buckets. Lookup time would be better/more predictable in more even buckets.

1. Quiz: Keywords and Buckets

Double keywords double buckets or halve keywords halve buckets.

1. Quiz: Implementing Hash Tables

Each bucket is a list.

1. Quiz: Empty Hash Table

# Hashtable needs to start as a list of empty buckets. n amount of buckets

def make\_hashtable(nbuckets):

hash\_table = []

while nbuckets > 0:

hash\_table.append([])

nbuckets -= 1

return hash\_table

1. Quiz: The Hard Way

Each element in the output refers to the same empty list.

1. Quiz: Finding Buckets

def hashtable\_get\_bucket(htable, keyword):

return htable[hash\_string(keyword, len(htable))]

1. Quiz: Adding Keywords

# video answer used a bucket variable

def hashtable\_add(htable, key, value):

hashtable\_get\_bucket(htable, key).append([key, value])

return htable

1. Quiz: Lookup

def hashtable\_lookup(htable, key):

bucket = hashtable\_get\_bucket(htable, key)

for item in bucket:

if item[0] == key:

return item[1]

1. Quiz: Update

def hashtable\_update(htable, key, value):

key\_in\_table = hashtable\_lookup(htable, key)

if key\_in\_table is not None:

bucket = hashtable\_get\_bucket(htable, key)

for entry in bucket:

if entry[0] == key:

entry[1] = value

break

else:

hashtable\_add(htable, key, value)

return htable

1. Dictionaries

Dictionaries contain sets of key value pairs.

mutable

1. Using Dictionaries

You can update values like dict[key] = value.

1. Quiz: Population

population = dict()

population['Shanghai'] = 17.8

population['Istanbul'] = 13.3

population['Karachi'] = 13.0

population['Mumbai'] = 12.5

1. A Noble Gas
2. Quiz: Modifying the Search Engine
3. Quiz: Changing Lookup
4. Coming Up Next