ch.14

October 11, 2020

0.1 Ch. 14: Algorithms

[6]:

```
import math
import turtle
import random
from unit_tester import test
import time
```

#Linear Search Algorithm

[7]:

```
def search_linear(xs, target):
    """ Find and return the index of target in sequence xs """
    for (i, v) in enumerate(xs):
        if v == target:
            return i
    return -1

friends = ["Joe", "Zoe", "Brad", "Angelina", "Zuki", "Thandi", "Paris"]
test(search_linear(friends, "Zoe") == 1)
test(search_linear(friends, "Joe") == 0)
test(search_linear(friends, "Paris") == 6)
test(search_linear(friends, "Bill") == -1)
```

```
Test at line 9 ok.
Test at line 10 ok.
Test at line 11 ok.
Test at line 12 ok.
```

#Create a function that finds unknown words

[ ]:

```
def find_unknown_words(vocab, wds):
    """ Return a list of words in wds that do not occur in vocab """
    result = []
    for w in wds:
        if (search_linear(vocab, w) < 0):                #liner search
        #if (search_binary(vocab, w) < 0):                  #binary search
            result.append(w)
    return result
```

```
vocab = ["apple", "boy", "dog", "down", "fell", "girl", "grass", "the", "tree"]
book_words = "the apple fell from the tree to the grass".split()
```

[8]:

```
test(find_unknown_words(vocab, book_words) == ["from", "to"])
test(find_unknown_words([], book_words) == book_words)
test(find_unknown_words(vocab, ["the", "boy", "fell"]) == [])
```

```
Test at line 1 FAILED.
Test at line 2 ok.
Test at line 3 ok.
```

#Load a list of Vocabulary words

[10]:

```
def load_words_from_file(filename):
    """ Read words from filename, return list of words. """
    f = open(filename, "r")
    file_content = f.read()
    f.close()
    wds = file_content.split()
    return wds

bigger_vocab = load_words_from_file("vocab.txt")
print("There are {0} words in the vocab, starting with\n {1} ".
 ↪format(len(bigger_vocab), bigger_vocab[:6]))
```

```
There are 19455 words in the vocab, starting with
 ['a', 'aback', 'abacus', 'abandon', 'abandoned', 'abandonment']
```

#Translate Function (remove punctuation)

[11]:

```
def text_to_words(the_text):
    """ return a list of words with all punctuation removed,
    and all in lowercase.
    """

    my_substitutions = the_text.maketrans(
    # If you find any of these
    "ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789!\"#$%&()*+,-./:;<=>?@[]^_‘{|}~’\\",
    # Replace them by these
    "abcdefghijklmnopqrstuvwxyz                                          ")

    # Translate the text now.
    cleaned_text = the_text.translate(my_substitutions)
    wds = cleaned_text.split()
    return wds
```

[12]:

```
test(text_to_words("My name is Earl!") == ["my", "name", "is", "earl"])
test(text_to_words('"Well, I never!", said Alice.') == ["well", "i", "never",␣
 ↪"said", "alice"])
```

```
Test at line 1 ok.
Test at line 2 ok.
```

#Load Alice in Wonderland Book

[13]:

```
def get_words_in_book(filename):
    """ Read a book from filename, and return a list of its words. """
    f = open(filename, "r")
    content = f.read()
    f.close()
    wds = text_to_words(content)
    return wds

book_words = get_words_in_book("AliceInWonderland.txt")
print("There are {0} words in the book, the first 100 are\n{1}" .
 ↪format(len(book_words), book_words[:100]))
```

```
There are 27803 words in the book, the first 100 are
["alice's", 'adventures', 'in', 'wonderland', 'lewis', 'carroll', 'chapter',
'i', 'down', 'the', 'rabbit', 'hole', 'alice', 'was', 'beginning', 'to', 'get',
'very', 'tired', 'of', 'sitting', 'by', 'her', 'sister', 'on', 'the', 'bank',
'and', 'of', 'having', 'nothing', 'to', 'do', 'once', 'or', 'twice', 'she',
'had', 'peeped', 'into', 'the', 'book', 'her', 'sister', 'was', 'reading',
'but', 'it', 'had', 'no', 'pictures', 'or', 'conversations', 'in', 'it', "'and",
'what', 'is', 'the', 'use', 'of', 'a', 'book', "'", 'thought', 'alice',
"'without", 'pictures', 'or', 'conversation', "'", 'so', 'she', 'was',
'considering', 'in', 'her', 'own', 'mind', 'as', 'well', 'as', 'she', 'could',
'for', 'the', 'hot', 'day', 'made', 'her', 'feel', 'very', 'sleepy', 'and',
'stupid', 'whether', 'the', 'pleasure', 'of', 'making']
```

#Compare List of Vocabulary Words to Alice in Wonderland Book

[14]:

```
missing_words = find_unknown_words(bigger_vocab, book_words)
#print(missing_words)
```

#Time the Search

[15]:

```
import time

t0 = time.perf_counter()
missing_words = find_unknown_words(bigger_vocab, book_words)
t1 = time.perf_counter()

print("There are {0} unknown words.".format(len(missing_words)))
print("That took {0:.4f} seconds.".format(t1-t0))
```

```
There are 5310 unknown words.
That took 72.2800 seconds.
```

#Binary Search

[16]:

```
def search_binary(xs, target):
    """ Find and return the index of key in sequence xs """
    lb = 0
    ub = len(xs)
    while True:
        if lb == ub: # If region of interest (ROI) becomes empty
            return -1

        # Next probe should be in the middle of the ROI
        mid_index = (lb + ub) // 2

        # Fetch the item at that position
        item_at_mid = xs[mid_index]

        #print("ROI[{0}:{1}](size={2}), probed='{3}', target='{4}'" .format(lb, ub, ub-lb, item_at_mid, target))

        # How does the probed item compare to the target?
        if item_at_mid == target:
            return mid_index # Found it!
        if item_at_mid < target:
            lb = mid_index + 1 # Use upper half of ROI next time
        else:
            ub = mid_index # Use lower half of ROI next time
```

#Tests for Binary Search

[17]:

```
xs = [2,3,5,7,11,13,17,23,29,31,37,43,47,53]
test(search_binary(xs, 3) == 1)
test(search_binary(xs, 20) == -1)
test(search_binary(xs, 99) == -1)
test(search_binary(xs, 1) == -1)
for (i, v) in enumerate(xs):
    test(search_binary(xs, v) == i)
```

```
Test at line 4 ok.
Test at line 5 ok.
Test at line 6 ok.
Test at line 7 ok.
Test at line 9 ok.
Test at line 9 ok.
Test at line 9 ok.
Test at line 9 ok.
Test at line 9 ok.
Test at line 9 ok.
Test at line 9 ok.
Test at line 9 ok.
Test at line 9 ok.
Test at line 9 ok.
Test at line 9 ok.
Test at line 9 ok.
Test at line 9 ok.
Test at line 9 ok.
```

#Test Time again compared to algorithm "search_linear"

[18]:

```python
import time

t0 = time.perf_counter()
missing_words = find_unknown_words(bigger_vocab, book_words)
t1 = time.perf_counter()

print("There are {0} unknown words.".format(len(missing_words)))
print("That took {0:.4f} seconds.".format(t1-t0))
```

```
There are 5310 unknown words.
That took 73.3399 seconds.
```

#Remove Duplicates

[ ]:

```python
def remove_adjacent_dups(xs):
    """ Return a new list in which all adjacent
    duplicates from xs have been removed.
    """
    result = []
    most_recent_elem = None
    for e in xs:
        if e != most_recent_elem:
            result.append(e)
            most_recent_elem = e

    return result
```

#Merge Sorted Lists

[ ]:

```python
def merge(xs, ys):
    """ merge sorted lists xs and ys. Return a sorted result """
    result = []
    xi = 0
    yi = 0
    while True:
        if xi >= len(xs): # If xs list is finished,
            result.extend(ys[yi:]) # Add remaining items from ys
            return result # And we’re done.

        if yi >= len(ys): # Same again, but swap roles
            result.extend(xs[xi:])
            return result

        # Both lists still have items, copy smaller item to result.
        if xs[xi] <= ys[yi]:
            result.append(xs[xi])
            xi += 1
        else:
            result.append(ys[yi])
            yi += 1
```

[20]:

```
xs = [1,3,5,7,9,11,13,15,17,19]
ys = [4,8,12,16,20,24]
zs = xs+ys
zs.sort()
test(merge(xs, []) == xs)
test(merge([], ys) == ys)
test(merge([], []) == [])
test(merge(xs, ys) == zs)

test(merge([1,2,3], [3,4,5]) == [1,2,3,3,4,5])
test(merge(["a", "big", "cat"], ["big", "bite", "dog"]) == ["a", "big", "big",␣
 ↪"bite", "cat", "dog"])
```

```
Test at line 5 ok.
Test at line 6 ok.
Test at line 7 ok.
Test at line 8 ok.
Test at line 10 ok.
Test at line 11 ok.
```

#Merge sorted Alice in Wonderland Example

[ ]:

```
def find_unknowns_merge_pattern(vocab, wds):
    """ Both the vocab and wds must be sorted. Return a new
    list of words from wds that do not occur in vocab.
    """

    result = []
    xi = 0
    yi = 0
    while True:
        if xi >= len(vocab):
            result.extend(wds[yi:])
            return result

        if yi >= len(wds):
            return result

        if vocab[xi] == wds[yi]: # Good, word exists in vocab
            yi += 1

        elif vocab[xi] < wds[yi]: # Move past this vocab word,
            xi += 1

        else: # Got word that is not in vocab
            result.append(wds[yi])
            yi += 1
```

[ ]:

```
#Test
```

[21]:

```
all_words = get_words_in_book("AliceInWonderland.txt")
#t0 = time.perf_counter()
all_words.sort()

book_words = remove_adjacent_dups(all_words)

missing_words = find_unknowns_merge_pattern(bigger_vocab, book_words)

t1 = time.perf_counter()

print("There are {0} unknown words.".format(len(missing_words)))
print("That took {0:.4f} seconds.".format(t1-t0))

#Debuger get nicht weiter als Zeile 277!
```

```
There are 1133 unknown words.
That took 246.5545 seconds.
```

0.2 Ch. 14: Merge Algorithm for merging Lists

#Ex. 1

#a)

[24]:

```
def return_both_present(xs, ys):
    """ merge sorted lists xs and ys. Return a sorted result """
    result = []
    xi = 0
    yi = 0

    while True:
        if xi >= len(xs):
            return result

        if yi >= len(ys):
            return result

        if xs[xi] < ys[yi]:
            xi += 1
        elif xs[xi] > ys[yi]:
            yi += 1
        else:
            result.append(xs[xi])
            xi += 1
            yi += 1
```

[25]:

```
print(return_both_present([1, 1, 3, 5, 7, 8, 9, 10], [1, 2, 3, 4, 5, 6, 7, 10]))
```

```
[1, 3, 5, 7, 10]
```

#b)

[26]:

```
def return_first_list_present_only(xs, ys):
    """ merge sorted lists xs and ys. Return a sorted result """
    result = []
    xi = 0
    yi = 0

    while True:
        if xi >= len(xs):
            return result

        if yi >= len(ys):
            result.append(xs[yi:])
            return result

        if xs[xi] < ys[yi]:
            result.append(xs[xi])
            xi += 1
        elif xs[xi] == ys[yi]:
            xi += 1
        else:
            yi += 1
```

[27]:

```
print(return_first_list_present_only([0, 1, 1, 2, 3, 4, 5, 7], [1, 3, 5, 6, 7,␣
 ↪10]))
```

```
[0, 2, 4]
```

#c)

[28]:

```
def return_second_list_present_only(xs, ys):
    """ merge sorted lists xs and ys. Return a sorted result """
    result = []
    xi = 0
    yi = 0

    while True:
        if xi >= len(xs):
            result.extend(ys[yi:])
            return result

        if yi >= len(ys):

            return result

        if xs[xi] < ys[yi]:
            xi += 1
        elif xs[xi] == ys[yi]:
            yi += 1
        else:
            result.append(ys[yi])
            yi += 1
```

[29]:

```
print(return_second_list_present_only([0, 1, 1, 2, 3, 4, 5, 7], [-1, 1, 3, 5, 6,␣
 ↪7, 8, 10]))
```

```
[-1, 6, 8, 10]
```

#d)

[30]:

```
def return_unique_items_in_both_lists(xs, ys):
    """ merge sorted lists xs and ys. Return a sorted result """
    result = []
    xi = 0
    yi = 0

    while True:
        if xi >= len(xs):
            result.extend(ys[yi:])
            return result

        if yi >= len(ys):
            result.extend(xs[xi:])
            return result
        if xs[xi] < ys[yi]:
            result.append(xs[xi])
            xi += 1
        elif xs[xi] > ys[yi]:
            result.append(ys[yi])
            yi += 1
        else:
            xi += 1
            yi += 1
```

[31]:

```
print(return_unique_items_in_both_lists([0, 1, 2.5, 3, 4, 4.5, 5, 7], [-1, 1, 2, 3, 5, 6, 7, 8, 10]))
```

```
[-1, 0, 2, 2.5, 4, 4.5, 6, 8, 10]
```

#e)

[32]:

```
def bagdiff(xs, ys):
    """ merge sorted lists xs and ys. Return a sorted result """
    result = []
    xi = 0
    yi = 0

    while True:
        if xi >= len(xs):
            result.extend(ys[yi:])
            return result

        if yi >= len(ys):
            result.extend(xs[xi:])
            return result

        if xs[xi] < ys[yi]:
            result.append(xs[xi])
            xi += 1
        elif xs[xi] > ys[yi]:
            yi += 1
        else:
            xi += 1
            yi += 1
```

[33]:

```
print(bagdiff([5, 7, 11, 11, 11, 12, 13], [7, 8, 11]))
from unit_tester import test

test(bagdiff([5,7,11,11,11,12,13], [7,8,11]) == [5,11,11,12,13])
```

```
[5, 11, 11, 12, 13]
Test at line 4 ok.
```

#Ex. 2, 3, 4 #skipped

#Ex. 5: Lottery with Prime Numbers

[34]:

```
import random
from unit_tester import test

my_tickets = [[7, 17, 37, 19, 23, 43], [7, 2, 13, 41, 31, 43], [2, 5, 7, 11, 13,␣
 ↪17],
              [13, 17, 37, 19, 23, 43]]


def lotto_draw():
    lotto_generator = random.Random()
    result = []
    for i in range(6):
        result.append(lotto_generator.uniform(1, 50))
    return result


def lotto_match(lotto1, lotto2):
    count = 0
    for item in lotto1:
        if item in lotto2:
            count += 1
    return count


def lotto_matches(lotto, mytick):
    result = []
    for i in range(4):
        result.append(lotto_match(lotto, my_tickets[i]))
    return result


def PriNumGenerator(upperlimit):
    Plist = [2]
    for num in range(2, upperlimit + 1):
        isprime = True
        for i in Plist:
            if num % i == 0:
                isprime = False
                break
        if isprime == True:
            Plist.append(num)
    return Plist
```

```
def primes_in(l):
    prime_list = PriNumGenerator(50)
    count = 0
    for item in l:
        if item in prime_list:
            count += 1
    return count


def prime_misses(l):
    prime_list = PriNumGenerator(50)
    result = []
    new = []
    for i in range(len(l)):
        new += l[i]
    for item in prime_list:
        if item in new:
            continue
        else:
            result.append(item)
    return result
```

[23]:

```
test(lotto_match([42, 4, 7, 11, 1, 13], [2, 5, 7, 11, 13, 17]) == 3)
test(lotto_matches([42, 4, 7, 11, 1, 13], my_tickets) == [1, 2, 3, 1])
test(primes_in([42, 4, 7, 11, 1, 13]) == 3)
test(prime_misses(my_tickets) == [3, 29, 47])
```

```
Test at line 1 ok.
Test at line 2 ok.
Test at line 3 ok.
Test at line 4 ok.
```

[ ]: