Python Algorithms

Python basic and advanced algorithms.

- Chapter 1: Data structure
  - Algorithms Hash Table
  - LinkedList
  - Queue
  - Stack functions
- Chapter 2: Basic algorithms
  - GCD
- Chapter 3: Recursion
  - Countdown example
  - Recursion
- Chapter 4: Searching algorithms
  - Binary Search
  - Interpolate Search
  - Determine if a list is sorted
  - Linear Search
  - Searching in a ordered list
  - Searching an unordered list
- Chapter 5: Sorting Algorithms
  - Bubble Sort
  - Insertion Sort
  - Merge Sort
  - Quick Sort - the fastest out of these basic sorts for large datasets
  - Selection Sort
  - Shell Sort
- Chapter 6: Applications, Simple and advanced
  - PersonDB Example Using various algorithms
  - Page Rank example with graphs

- Traveling Salesman - greedy and brute force algorithms
- Linear programming algorithms - using pulp (open source library)
- Spark example

- Chapter 7: Others miscellaneous
  - use a hashtable to filter out duplicate items - Filter.py
  - using a hashtable to count individual items - ValueCounter.py
  - use a recursive algorithm to find a maximum value - findmax.py

Chapter 1: Data structure

Hash Table , LinkedList , Queues , Stacks

Algorithms Hash Table

```
# demonstrate hashtable usage

# create a hashtable all at once

items1 = dict({"key1": 1, "key2": 2, "key3": "three"})

print(items1)

# create a hashtable progressively

items2 = {}

items2["key1"] = 1

items2["key2"] = 2

items2["key3"] = 3

print(items2)

# try to access a nonexistent key

# print(items1["key6"])



# replace an item

items2["key2"] = "two"

print(items2)
```

```
# iterate the keys and values in the dictionary

for key, value in items2.items():

   print("key: ", key, " value: ", value)
```

LinkedList

```
# Linked list example

# the Node class

class Node(object):

    def __init__(self, val):

        self.val = val

        self.next = None

    def get_data(self):

        return self.val

    def set_data(self, val):

        self.val = val

    def get_next(self):

        return self.next



    def set_next(self, next):

        self.next = next

# the LinkedList class
```

```
class LinkedList(object):

    def __init__(self, head=None):

        self.head = head

        self.count = 0

    def get_count(self):

        return self.count



    def insert(self, data):

        new_node = Node(data)

        new_node.set_next(self.head)

        self.head = new_node

        self.count += 1

    def find(self, val):

        item = self.head

        while (item != None):

            if item.get_data() == val:

                return item

            else:

                item = item.get_next()
```

```
        return None

    def deleteAt(self, idx):

        if idx > self.count:

            return

        if self.head == None:

            return

        else:

            tempIdx = 0

            node = self.head

            while tempIdx < idx-1:

                node = node.get_next()

                tempIdx += 1

            node.set_next(node.get_next().get_next())

            self.count -= 1



    def dump_list(self):

        tempnode = self.head

        while (tempnode != None):

            print("Node: ", tempnode.get_data())
```

```
        tempnode = tempnode.get_next()



# create a linked list and insert some items
itemlist = LinkedList()
itemlist.insert(38)
itemlist.insert(49)
itemlist.insert(13)
itemlist.insert(15)

itemlist.dump_list()

# exercise the list
print("Item count: ", itemlist.get_count())
print("Finding item: 13", itemlist.find(13))
print("Finding item: 78", itemlist.find(78))

# delete an item
itemlist.deleteAt(3)
```

```
print("Delete item # 3")

print("Item count: ", itemlist.get_count())

print("Finding item: 38", itemlist.find(38))

itemlist.dump_list()

print("Finding item: 13", itemlist.find(15))
```

Queue

```
# try out the Python queue functions

from collections import deque



# create a new empty deque object that will function as a queue

queue = deque()



# add some items to the queue

queue.append(1)

queue.append(2)

queue.append(3)

queue.append(4)

queue.append(5)



# print the queue contents

print(queue)
```

```
# pop an item off the front of the queue

x = queue.popleft()

print(x)

x = queue.popleft()

print(x)

print(queue)
```

Stack functions

```
# try out the Python stack functions

# create a new empty stack
stack = []

# push items onto the stack
stack.append(1)
stack.append(2)
stack.append(3)
stack.append(4)

# print the stack contents
print(stack)

# pop an item off the stack
x = stack.pop()
```

```
print(x)

print(stack)
```

Chapter 2: Basic algorithms

GCD

```
# Find the greatest common denominator of two numbers

# using Euclid's algorithm




def gcd(a, b):

    while (b != 0):

        t = a       # set aside the value of a

        a = b       # set a equal to b

        b = t % b   # divide t (which is a) by b



    return a




# try out the function with a few examples

print(gcd(60, 96))  # should be 12

print(gcd(20, 8))   # should be 4
```

Chapter 3: Recursion

Countdown example

```
# use recursion to implement a countdown counter

def countdown(x):
    if x == 0:
        print("Done!")
        return
    else:
        print(x, "...")
        countdown(x-1)

countdown(5)
```

Recursion

```
# Using recursion to implement power and factorial functions

def power(num, pwr):
    # breaking condition: if we reach zero, return 1
    if pwr == 0:
        return 1
    else:
        return num * power(num, pwr-1)

def factorial(num):
    if (num == 0):
        return 1
    else:
        return num * factorial(num-1)
```

```
print("{} to the power of {} is {}".format(5, 3, power(5, 3)))

print("{} to the power of {} is {}".format(1, 5, power(1, 5)))

print("{}! is {}".format(4, factorial(4)))

print("{}! is {}".format(0, factorial(0)))
```

Chapter 4: Searching algorithms

Binary Search

```
''' Binary search on ordered list '''

def BubbleSort(dataset):

    # start with the array length and decrement each time

    for i in range(len(dataset)-1, 0, -1):

        # examine each item pair

        for j in range(i):

            # swap items if needed

            if dataset[j] > dataset[j+1]:

                temp = dataset[j]

                dataset[j] = dataset[j+1]

                dataset[j+1] = temp

def BinarySearch(list, item):

    first = 0

    last = len(list)-1

    found = False

    while first <= last and not found:
```

```
      midpoint = (first + last)//2
      if list[midpoint] == item:
        found = True
      else:
        if item < list[midpoint]:
           last = midpoint - 1
        else:
           first = midpoint + 1
    return found
def main():
    list = [12,33, 11, 99, 22, 55, 90]
    sorted_list = BubbleSort(list)
    print(BinarySearch(list,12))
    print(BinarySearch(list,91))


if __name__ == "__main__":
    main()
```

Interpolate Search

```
''' interpolation search on ordered list '''

def BubbleSort(dataset):

    # start with the array length and decrement each time

    for i in range(len(dataset)-1, 0, -1):

        # examine each item pair

        for j in range(i):

            # swap items if needed

            if dataset[j] > dataset[j+1]:

                temp = dataset[j]

                dataset[j] = dataset[j+1]

                dataset[j+1] = temp

def IntPolsearch(list, x):

    idx0 = 0

    idxn = (len(list) - 1)

    found = False

    while idx0 <= idxn and x >= list[idx0] and x <= list[idxn]:
```

```
        mid = idx0 + int(((float(idxn - idx0)/(list[idxn] - list[idx0])) * (x - list[idx0])))
        if list[mid] == x:
            found = True
            return found
        if list[mid] < x:
            idx0 = mid + 1
    return found
def main():
    list = [12,33, 11, 99, 22, 55, 90]
    sorted_list = BubbleSort(list)
    print(IntPolsearch(list,99))
    print(IntPolsearch(list,11))
    print(IntPolsearch(list,21))


if __name__ == "__main__":
    main()
```

Determine if a list is sorted

```
# determine if a list is sorted


items1 = [6, 8, 19, 20, 23, 41, 49, 53, 56, 87]

items2 = [6, 20, 8, 19, 56, 23, 87, 41, 49, 53]


def is_sorted(itemlist):

    # using the all function

    return all(itemlist[i] <= itemlist[i+1] for i in range(len(itemlist)-1))


    # using the brute force method

    # for i in range(0, len(itemlist)-1):

    #     if (itemlist[i] > itemlist[i+1]):

    #         return False

    # return True
```

```
print(is_sorted(items1))

print(is_sorted(items2))
```

Linear Search

```
def LinearSearch(list, item):
  index = 0
  found = False
  while index < len(list) and  found is False:
    if list[index] == item:
      found = True
    else:
      index = index + 1
  return found


list = [12, 33, 11, 99, 22, 55, 90]
print(LinearSearch(list, 12))
print(LinearSearch(list, 91))
```

Searching in a ordered list

```
# searching for an item in an ordered list

# this technique uses a binary search



items = [6, 8, 19, 20, 23, 41, 49, 53, 56, 87]


def binarysearch(item, itemlist):

    # get the list size

    listsize = len(itemlist) - 1

    # start at the two ends of the list

    lowerIdx = 0

    upperIdx = listsize


    while lowerIdx <= upperIdx:

        # calculate the middle point

        midPt = (lowerIdx + upperIdx)// 2
```

```
        # if item is found, return the index
        if itemlist[midPt] == item:
            return midPt
        # otherwise get the next midpoint
        if item > itemlist[midPt]:
            lowerIdx = midPt + 1
        else:
            upperIdx = midPt - 1

    if lowerIdx > upperIdx:
        return None


print(binarysearch(23, items))
print(binarysearch(87, items))
print(binarysearch(250, items))
```

Searching an unordered list

```
# searching for an item in an unordered list

# sometimes called a Linear search



# declare a list of values to operate on

items = [6, 20, 8, 19, 56, 23, 87, 41, 49, 53]



def find_item(item, itemlist):

    for i in range(0, len(itemlist)):

        if item == itemlist[i]:

            return i



    return None





print(find_item(87, items))

print(find_item(250, items))
```

Chapter 5: Sorting Algorithms

Includes BubbleSort , MergeSort , Selection, Insertionsort , Quicksort , Shellsort

Bubble Sort

```
# Bubble sort algorithm




def bubbleSort(dataset):

    # start with the array length and decrement each time

    for i in range(len(dataset)-1, 0, -1):

        # examine each item pair

        for j in range(i):

            # swap items if needed

            if dataset[j] > dataset[j+1]:

                temp = dataset[j]

                dataset[j] = dataset[j+1]

                dataset[j+1] = temp



        print("Current state: ", dataset)
```

```
def main():
    list1 = [25, 21, 22, 24, 23, 27, 26]
    print("Starting state: ", list1)
    bubbleSort(list1)
    print("Final state: ", list1)


if __name__ == "__main__":
    main()
```

Output:

```
p bubblesort.py
Starting state:  [25, 21, 22, 24, 23, 27, 26]
Current state:  [21, 22, 24, 23, 25, 26, 27]
Current state:  [21, 22, 23, 24, 25, 26, 27]
Current state:  [21, 22, 23, 24, 25, 26, 27]
Current state:  [21, 22, 23, 24, 25, 26, 27]
Current state:  [21, 22, 23, 24, 25, 26, 27]
```

Current state: [21, 22, 23, 24, 25, 26, 27]

Final state: [21, 22, 23, 24, 25, 26, 27]

Insertion Sort

```
# insert sort algorithm



def insertion(list):

    for i in range(1,len(list)):

        j = i-1

        element_next = list[i]

        while (list[j] > element_next):

          list[j+1] = list[j]

          j=j-1

        list[j+1] = element_next

    return list



def main():

    list1 = [6, 20, 8, 19, 56, 23, 87, 41, 49, 53]

    print("Starting state: ", list1)
```

```
    insertion(list1)

    print("Final state: ", list1)




if __name__ == "__main__":

    main()
```

Output:

```
Williams-MacBook-Pro:Insertionsort williamcrupi$ p insertionsort.py

Starting state:  [6, 20, 8, 19, 56, 23, 87, 41, 49, 53]

Final state:  [6, 8, 19, 20, 23, 41, 49, 53, 56, 87]

Williams-MacBook-Pro:Insertionsort williamcrupi$
```

Merge Sort

```
# Implement a merge sort with recursion



items = [6, 20, 8, 19, 56, 23, 87, 41, 49, 53]


def mergesort(dataset):

    if len(dataset) > 1:

        mid = len(dataset) // 2

        leftarr = dataset[:mid]

        rightarr = dataset[mid:]



        # recursively break down the arrays

        mergesort(leftarr)

        mergesort(rightarr)

        # now perform the merging

        i=0 # index into the left array
```

```
j=0 # index into the right array
k=0 # index into merged array


# while both arrays have content
while i < len(leftarr) and j < len(rightarr):
    if leftarr[i] < rightarr[j]:
        dataset[k] = leftarr[i]
        i += 1
    else:
        dataset[k] = rightarr[j]
        j += 1
    k += 1


# if the left array still has values, add them
while i < len(leftarr):
    dataset[k] = leftarr[i]
    i += 1
    k += 1
```

```
        # if the right array still has values, add them
        while j < len(rightarr):
            dataset[k] = rightarr[j]
            j += 1
            k += 1


# test the merge sort with data
print(items)
mergesort(items)
print(items)
```

Output:

```
Williams-MacBook-Pro:MergeSort williamcrupi$ p mergesort.py
[6, 20, 8, 19, 56, 23, 87, 41, 49, 53]
[6, 8, 19, 20, 23, 41, 49, 53, 56, 87]
```

Quick Sort - the fastest out of these basic sorts for large datasets

```
# Implement a quicksort

items = [20, 6, 8, 53, 56, 23, 87, 41, 49, 19]

def quickSort(dataset, first, last):
    if first < last:
        # calculate the split point
        pivotIdx = partition(dataset, first, last)

        # now sort the two partitions
        quickSort(dataset, first, pivotIdx-1)
```

```
        quickSort(dataset, pivotIdx+1, last)



def partition(datavalues, first, last):
    # choose the first item as the pivot value
    pivotvalue = datavalues[first]
    # establish the upper and lower indexes
    lower = first + 1
    upper = last

    # start searching for the crossing point
    done = False
    while not done:
        # advance the lower index
        while lower <= upper and datavalues[lower] <= pivotvalue:
            lower += 1

        # advance the upper index
        while datavalues[upper] >= pivotvalue and upper >= lower:
```

```
        upper -= 1

    # if the two indexes cross, we have found the split point
    if upper < lower:
        done = True
    else:
        # exchange the two values
        temp = datavalues[lower]
        datavalues[lower] = datavalues[upper]
        datavalues[upper] = temp

# when the split point is found, exchange the pivot value
temp = datavalues[first]
datavalues[first] = datavalues[upper]
datavalues[upper] = temp

# return the split point index
return upper
```

```
# test the merge sort with data

print(items)

quickSort(items, 0, len(items)-1)

print(items)
```

OutPut:

```
Williams-MacBook-Pro:Quicksort williamcrupi$ p quicksort.py

[20, 6, 8, 53, 56, 23, 87, 41, 49, 19]

[6, 8, 19, 20, 23, 41, 49, 53, 56, 87]
```

Selection Sort

```
# selection sort algorithm

''' Selecton sort algorithm '''

def selection_sort(list1):

    ''' Selecton sort algorithm '''

    for fill_slot in range(len(list1) - 1, 0, -1):

        max_index = 0

        for location in range(1, fill_slot + 1):

            if list1[location] > list1[max_index]:

                max_index = location

        list1[fill_slot], list1[max_index] = list1[max_index], list1[fill_slot]

    return list1

def main():

    ''' Main '''

    list1 = [26,17,20,11,23,21,13,18,24,12,22,16,15,19,25]

    print("Starting state: ", list1)

    selection_sort(list1)
```

```
    print("Final state: ", list1)




if __name__ == "__main__":

    main()
```

OutPut:

```
Williams-MacBook-Pro:Selection williamcrupi$ p selection.py

Starting state:  [26, 17, 20, 11, 23, 21, 13, 18, 24, 12, 22, 16, 15, 19, 25]

Final state:  [11, 12, 13, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26]
```

Shell Sort

```
# shell sort algorithm



def shellSort(list):

    # start with the array length and decrement each time

    distance = len(list) // 2

    while distance > 0:

      for i in range(distance, len(list)):

        temp = list[i]

        j = i

        while j >= distance and list[j - distance] > temp:

          list[j] = list[j- distance]

          j = j - distance

        list[j] = temp

      distance = distance // 2

    return list

def main():
```

```
    list1 = [26,17,20,11,23,21,13,18,24,12,22,16,15,19,25]

    print("Starting state: ", list1)

    shellSort(list1)

    print("Final state: ", list1)




if __name__ == "__main__":

    main()
```

OutPut:

```
Williams-MacBook-Pro:Shellsort williamcrupi$ p shellsort.py

Starting state:  [26, 17, 20, 11, 23, 21, 13, 18, 24, 12, 22, 16, 15, 19, 25]

Final state:  [11, 12, 13, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26]
```

Chapter 6: Applications, Simple and advanced

Person DB example - Simple PageRank and Traveling Sales man - advanced

PersonDB Example Using various algorithms

```
# From work

import quicksort

import shellsort

import linearsearch

import random

import time



from datetime import date

approve = 0

reject = 0



def add_person(person,output):

    appid = get_appid()

    perid = get_perid()
```

```
    today = str(date.today())

    output.write('{} {} {} {} {} {} {}\n'.format(perid, appid, person[0], person[1], person[2], 
"Approved" , today ))



def add_appid(person, perid, output):

    appid = get_appid()

    today = str(date.today())

    output.write('{} {} {} {} {} {} {}\n'.format(perid, appid, person[0], person[1], person[2], 
"Approved" , today ))



def get_perid():

    perid = random.randint(0,99999)

    return perid



def get_appid():

    appid= random.randint(0,999999)

    return appid



start = time.time()
```

```
#open hist data files

histfile=open("hist.txt", "r+")

lines=histfile.readlines()

data=[tuple(line.strip().split()) for line in lines]



#open input data files

inputfile=open("input.txt", "r")

lines=inputfile.readlines()

inputdata=[tuple(line.strip().split()) for line in lines]



#sort data file



quicksort.quickSort(data,0,len(data)-1)



#read in input file



for inputitem in inputdata:

   j=linearsearch.LinearSearch(data,inputitem[2],4)

   approve = 0
```

```
    reject = 0
    if len(j):
        for item in j:
           if item[2] == inputitem[0] and item[3] == inputitem[1]:
             if item[5] == "Approved":
                approve = approve + 1
             else:
                reject = reject + 1
        add_appid(inputitem,item[0],histfile)
        print("Persons name",inputitem[0],inputitem[1])
        print("Number of current Approvals = ",approve)
        print("Number of currentRejections = ",reject)
    else:
       add_person(inputitem, histfile)
       print("Added ",inputitem[0],inputitem[1])


end = time.time()
print("\nRunning time",end - start)
```

OutPut:

Persons name John Doe

Number of current Approvals = 2

Number of currentRejections = 0

Added Joe Blow

Persons name Xman Xsir

Number of current Approvals = 0

Number of currentRejections = 1

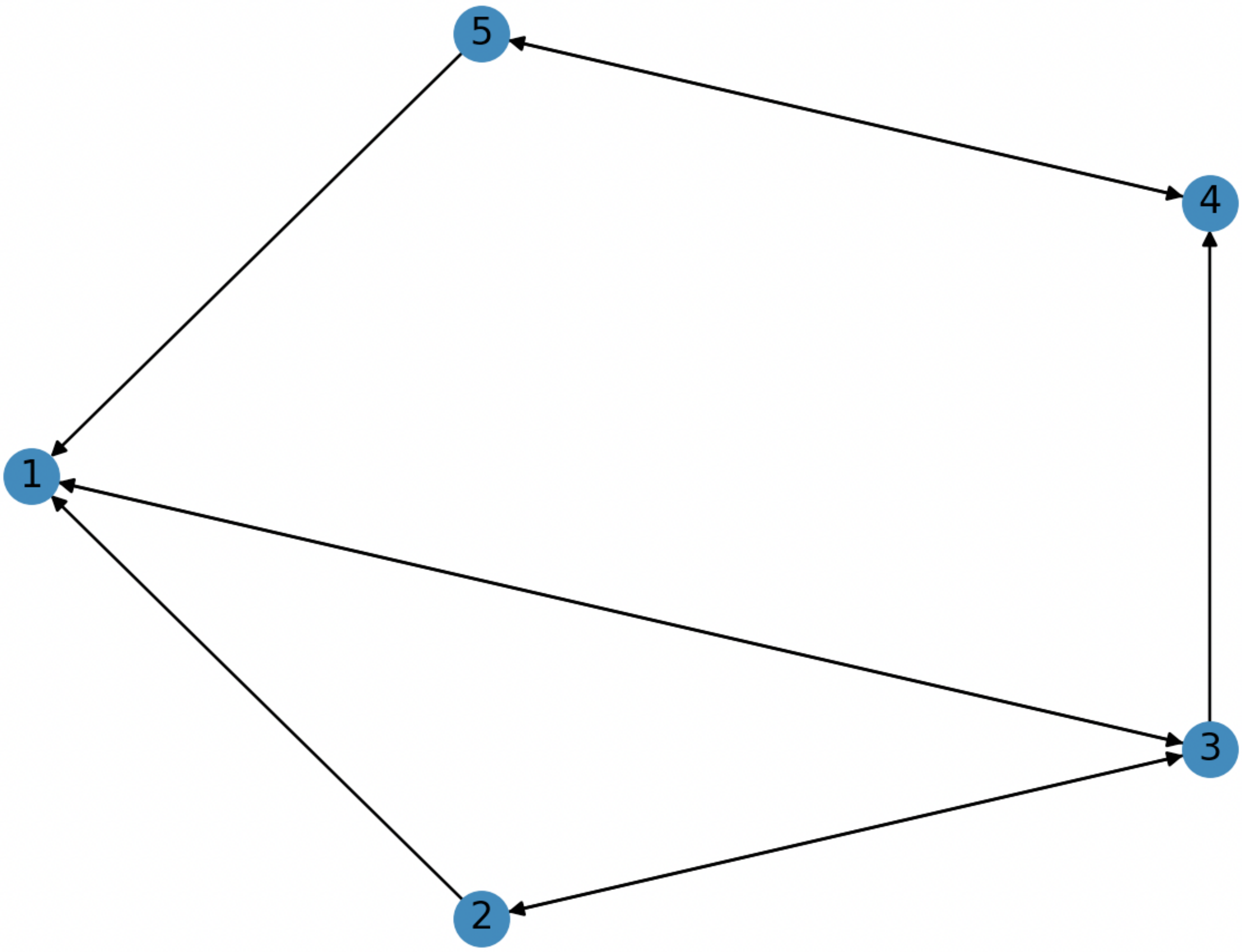
Running time 0.0005130767822265625

Page Rank example with graphs

```
import numpy as np

import networkx as nx

import matplotlib.pyplot as plt



def createPageRank(aGraph):

    nodes_set = len(aGraph)

    M = nx.to_numpy_matrix(aGraph)

    outwards = np.squeeze(np.asarray(np.sum(M,axis=1)))

    prob_outwards = np.array(

      [1.0/count

       if count>0 else 0.0 for count in outwards])

    G = np.asarray(np.multiply(M.T, prob_outwards))

    P = np.ones(nodes_set) / float(nodes_set)

    if np.min(np.sum(G,axis=0)) < 1.0:
```

```
        print('WARN: G is substochastic')

    return G,P



myWeb = nx.DiGraph()

myPages = range(1,5)



connections = [(1,3),(2,1),(2,3),(3,1),(3,2),(3,4),(4,5),(5,1),(5,4)]

myWeb.add_nodes_from(myPages)

myWeb.add_edges_from(connections)



pos=nx.shell_layout(myWeb)

nx.draw(myWeb, pos, arrows=True,with_labels=True)

G, p = createPageRank(myWeb)

print(G)

plt.show()
```

OutPut:




```
[[0.         0.5        0.33333333 0.         0.5       ]
 [0.         0.         0.33333333 0.         0.        ]
 [1.         0.5        0.         0.         0.        ]
 [0.         0.         0.33333333 0.         0.5       ]
 [0.         0.         0.         1.         0.        ]]
```

Traveling Salesman - greedy and brute force algorithms

```
import random

from itertools import permutations

import matplotlib.pyplot as plt

from time import perf_counter

from collections import Counter



alltours = permutations

aCity = complex



def distance_tour(aTour):

    return sum(distance_points(aTour[i - 1], aTour[i])

               for i in range(len(aTour)))
```

```
def distance_points(first, second): return abs(first - second)



def generate_cities (number_of_cities):

    seed = 111; width = 500; height=300

    random.seed(number_of_cities,seed)

    return frozenset(aCity(random.randint(1,width), random.randint(1,height))

                    for c in range(number_of_cities))



def brute_force(cities):

    "Generate all possible tours of the cities and choose the shortest tour."

    return shortest_tour(alltours(cities))



def greedy_algorithm(cities, start=None):

    C = start or first(cities)

    tour = [C]

    unvisited = set(cities - {C})

    while unvisited:

        C = nearest_neighbor(C, unvisited)

        tour.append(C)
```

```
        unvisited.remove(C)

    return tour



def first(collection): return next(iter(collection))



def nearest_neighbor(A, cities):

    return min(cities, key=lambda C: distance_points(C,A))



def shortest_tour(tours): return min(tours, key=distance_tour)





def visualize_tour(tour, style='bo-'):

    if len(tour) > 1000: plt.figure(figsize=(15,10))

    start = tour[0:1]

    visualize_segment(tour + start, style)

    visualize_segment(start, 'rD')



def visualize_segment(segment, style='bo-'):

    plt.plot([X(c) for c in segment], [Y(c) for c in segment], style, clip_on=False)
```

```
    plt.axis('scaled')

    plt.axis('off')



def X(city): "X axis"; return city.real



def Y(city): "Y axis"; return city.imag




def tsp(algorithm,cities):

    t0 = perf_counter()

    tour = algorithm(cities)

    t1 = perf_counter()

    assert Counter(tour) == Counter(cities)

    visualize_tour(tour)

    print("{}: {} cities -> tour length {:.0f} (in {:.3} sec)".format(name(algorithm), len(tour),
distance_tour(tour),t1-t0))



def name(algorithm): return algorithm.__name__.replace('_tsp', '')
```

tsp(greedy_algorithm , generate_cities(10))

plt.show()

Williams-MacBook-Pro:advanced williamcrupi$ vi travel.py

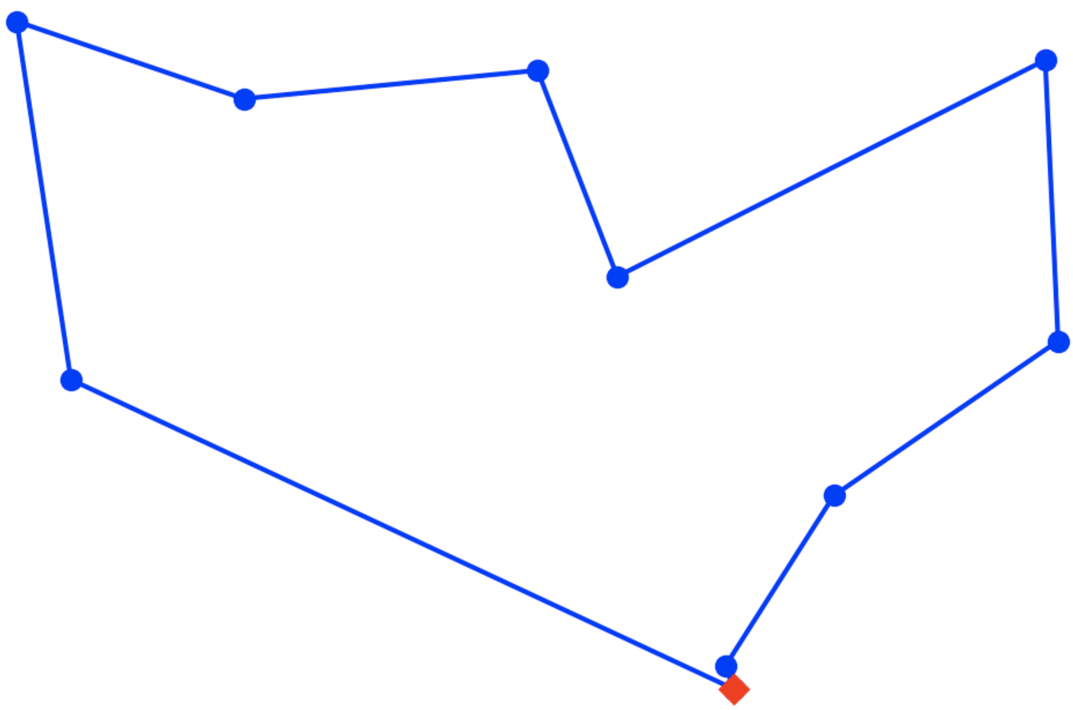
Williams-MacBook-Pro:advanced williamcrupi$ clear

Williams-MacBook-Pro:advanced williamcrupi$ p travel.py

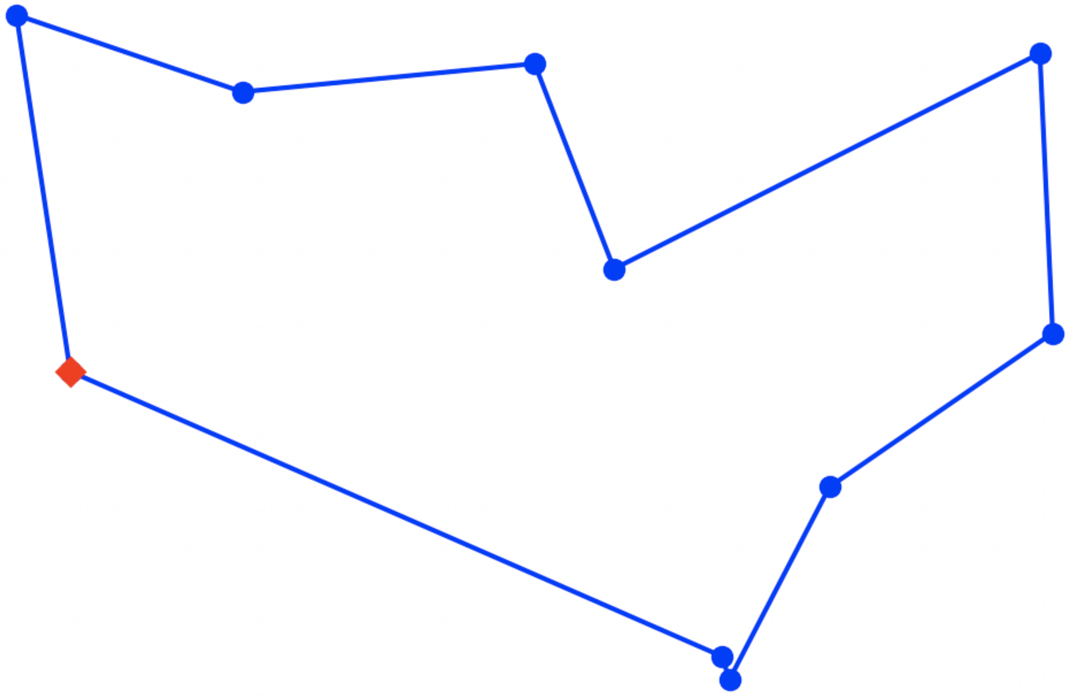
greedy_algorithm: 10 cities -> tour length 1206 (in 4.31e-05 sec)

OutPut:

greedy_algorithm: 10 cities -> tour length 1206 (in 4.31e-05 sec)



brute_force: 10 cities -> tour length 1206 (in 9.37 sec)



Linear programming algorithms - using pulp (open source library)

```
import pulp



# Instantiate our problem class



model = pulp.LpProblem("Profit_maximising_problem", pulp.LpMaximize)



A = pulp.LpVariable('A', lowBound=0, cat='Integer')

B = pulp.LpVariable('B', lowBound=0, cat='Integer')

model += 5000 * A + 2500 * B, "Profit"



model += 3 * A + 2 * B <= 20

model += 4 * A + 3 * B <= 30

model += 4 * A + 3 * B <= 44
```

```
model.solve()

pulp.LpStatus[model.status]

print("\n")

print (A.varValue)

print (B.varValue)



print("\n")

print (pulp.value(model.objective))
```

OutPut:

Based on the following data:

```
Williams-MacBook-Pro:linear williamcrupi$ cat people.txt

            Technician      AI Specialist       Engineer


Number of

People       1              1                  2
```

Total number 1 x 20 = 20 1 x 30 = 30 2 x 22 =

of days in a days days 44 days

cycle

Williams-MacBook-Pro:linear williamcrupi$ cat robot.txt

Type of Robot Technician AI Specialist Engineer

Robot A: 3 days 4 days 4 days

Advanced

model

Robot B: basic 2 days 3 days 3 days

model

The run:

p linear.py

Welcome to the CBC MILP Solver

Version: 2.9.0

Build Date: Feb 12 2015

```
command line - /Library/Frameworks/Python.framework/Versions/3.9/lib/python3.9/site-packages/pulp/apis/../solverdir/cbc/osx/64/cbc /var/folders/3s/q8mtw3pn3znc1lwd9g04yj640000gn/T/c625235ad08945ec9a62473329e14e26-pulp.mps max branch printingOptions all solution /var/folders/3s/q8mtw3pn3znc1lwd9g04yj640000gn/T/c625235ad08945ec9a62473329e14e26-pulp.sol (default strategy 1)

At line 2 NAME          MODEL

At line 3 ROWS

At line 8 COLUMNS

At line 21 RHS

At line 25 BOUNDS

At line 28 ENDATA

Problem MODEL has 3 rows, 2 columns and 6 elements

Coin0008I MODEL read with 0 errors

Continuous objective value is 33333.3 - 0.00 seconds

Cgl0004I processed model has 2 rows, 2 columns (2 integer (0 of which binary)) and 4 elements

Cutoff increment increased from 1e-05 to 2500

Cbc0012I Integer solution of -32500 found by DiveCoefficient after 0 iterations and 0 nodes (0.00 seconds)

Cbc0001I Search completed - best objective -32500, took 0 iterations and 0 nodes (0.00 seconds)

Cbc0035I Maximum depth 0, 0 variables fixed on reduced cost
```

Cuts at root node changed objective from -32500 to -32500

Probing was tried 0 times and created 0 cuts of which 0 were active after adding rounds of cuts (0.000 seconds)

Gomory was tried 0 times and created 0 cuts of which 0 were active after adding rounds of cuts (0.000 seconds)

Knapsack was tried 0 times and created 0 cuts of which 0 were active after adding rounds of cuts (0.000 seconds)

Clique was tried 0 times and created 0 cuts of which 0 were active after adding rounds of cuts (0.000 seconds)

MixedIntegerRounding2 was tried 0 times and created 0 cuts of which 0 were active after adding rounds of cuts (0.000 seconds)

FlowCover was tried 0 times and created 0 cuts of which 0 were active after adding rounds of cuts (0.000 seconds)

TwoMirCuts was tried 0 times and created 0 cuts of which 0 were active after adding rounds of cuts (0.000 seconds)

Result - Optimal solution found

Objective value: 32500.00000000

Enumerated nodes: 0

Total iterations: 0

Time (CPU seconds):   0.00

Time (Wallclock seconds):   0.00

Option for printingOptions changed from normal to all

Total time (CPU seconds):   0.00   (Wallclock seconds):   0.00

6.0

1.0

32500.0

Spark example

```
import findspark

findspark.init()

from pyspark.sql import SparkSession

spark = SparkSession.builder.master("local[*]").getOrCreate()

sc = spark.sparkContext

sc.setLogLevel("OFF")

wordsList = ['python', 'java', 'ottawa', 'ottawa', 'java', 'news']

wordsRDD = sc.parallelize(wordsList,4)

print(wordsRDD.collect())

wordPairs = wordsRDD.map(lambda w: (w, 1))

print (wordPairs.collect())

wordCountsCollected = wordPairs.reduceByKey(lambda x, y: x+y)

print(wordCountsCollected.collect())
```

OutPut:

```
['python', 'java', 'ottawa', 'ottawa', 'java', 'news']

[('python', 1), ('java', 1), ('ottawa', 1), ('ottawa', 1), ('java', 1), ('news', 1)]
```

```
[('python', 1), ('java', 2), ('ottawa', 2), ('news', 1)]
```

Chapter 7: Others miscellaneous

create a hashtable to perform a filter - Filter.py create a hashtable object to hold the items and counts - ValueCounter.py use a recursive algorithm to find a maximum value - findmax.py

use a hashtable to filter out duplicate items - Filter.py

```
# use a hashtable to filter out duplicate items



# define a set of items that we want to reduce duplicates

items = ["apple", "pear", "orange", "banana", "apple",

         "orange", "apple", "pear", "banana", "orange",

         "apple", "kiwi", "pear", "apple", "orange"]



# create a hashtable to perform a filter

filter = dict()



# loop over each item and add to the hashtable

for item in items:

    filter[item] = 0
```

```
# create a set from the resulting keys in the hashtable

result = set(filter.keys())

print(result)
```

OutPut:

```
p Filter.py

{'pear', 'banana', 'apple', 'orange', 'kiwi'}
```

using a hashtable to count individual items - ValueCounter.py

```
# using a hashtable to count individual items



# define a set of items that we want to count

items = ["apple", "pear", "orange", "banana", "apple",

        "orange", "apple", "pear", "banana", "orange",

        "apple", "kiwi", "pear", "apple", "orange"]


# create a hashtable object to hold the items and counts

counter = dict()


# iterate over each item and increment the count for each one

for item in items:
```

```
    if item in counter.keys():

        counter[item] += 1

    else:

        counter[item] = 1



# print the results

print(counter)
```

OutPut:

```
p ValueCounter.py

{'apple': 5, 'pear': 3, 'orange': 4, 'banana': 2, 'kiwi': 1}
```

use a recursive algorithm to find a maximum value - findmax.py

```
# use a recursive algorithm to find a maximum value



# declare a list of values to operate on

items = [6, 20, 8, 19, 56, 23, 87, 41, 49, 53]



def find_max(items):

    # breaking condition: last item in list? return it

    if len(items) == 1:

        return items[0]



    # otherwise get the first item and call function

    # again to operate on the rest of the list
```

```python
    op1 = items[0]

    print(op1)

    op2 = find_max(items[1:])

    print(op1, op2)



    # perform the comparison when we're down to just two

    if op1 > op2:

        return op1

    else:

        return op2




# test the function

print(find_max(items))
```

OutPut:

```
p findmax.py

6

20

8
```

19

56

23

87

41

49

49 53

41 53

87 53

23 87

56 87

19 87

8 87

20 87

6 87

87