UNIVERSITY OF MALAYA

WIA2005 Algorithm Design and Analysis

Session 2, 2019/2020

GROUP ASSIGNMENT REPORT

PROJECT 2

LECTURER: DR. RAJA JAMILAH BINTI RAJA YUSOF

NO.	NAME	MATRIC NO.
1	AHMAD FARIS IMRAN BIN HASBOLLAH	17092732/2
2	FAIYAZ KHAN	17128397/2
3	NUR ALIA HUSNA BINTI KHAIRUL ANUAR	17136469/1
4	SHAHAN ALI PRANTO	17078402/1
5	MUHAMMAD FAHMI BIN SAMIRON	17113994/2
6	HESWARIY	17084251/1

Table of Contents

Introduction

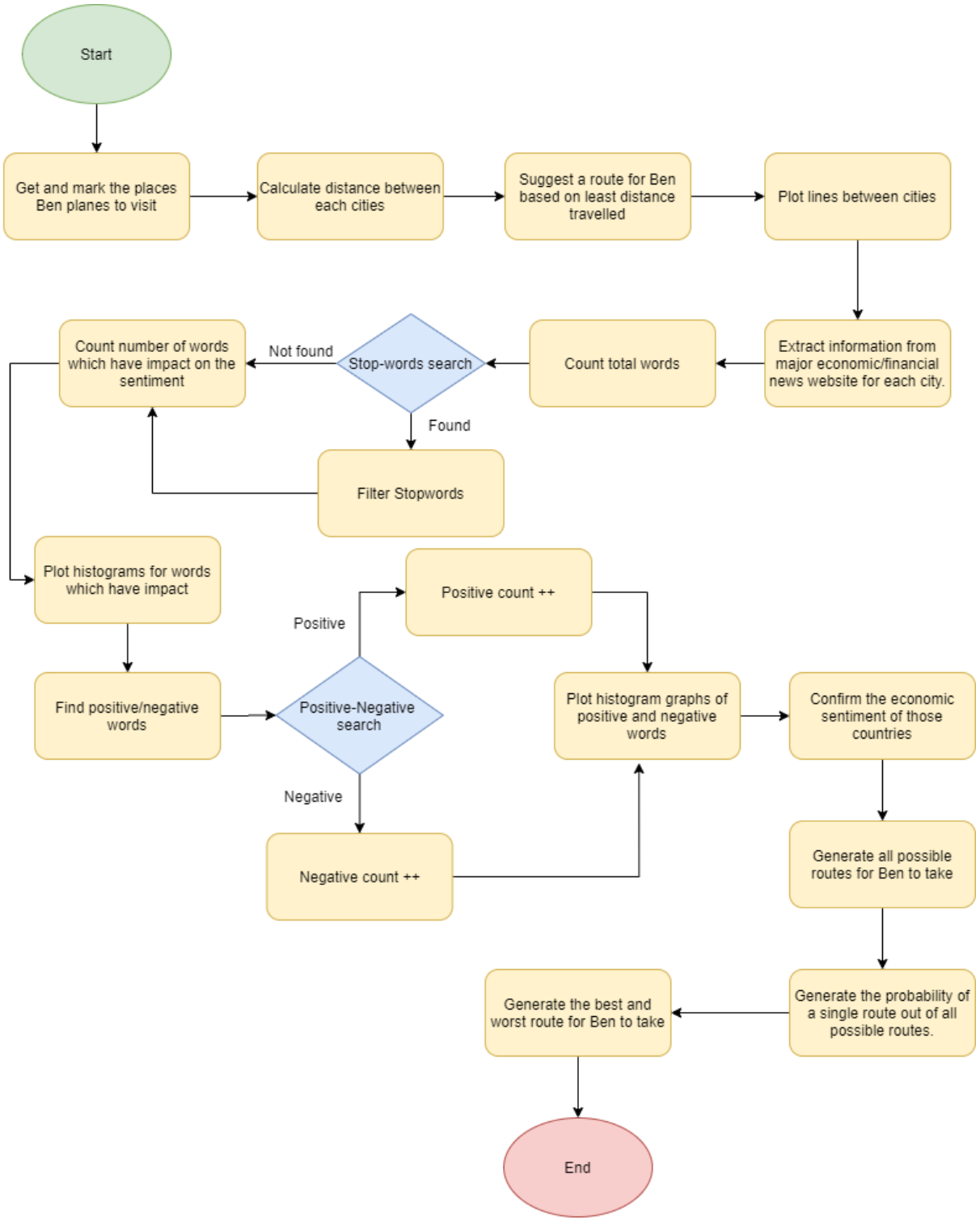
The project we are doing is about Ben Sherman who is a UK broker. He is looking for industrial investment opportunities in the cities of Asia. He already invested in a company in Kuala Lumpur and now he plans to travel to several cities in Asia from Kuala Lumpur to expand his investment. The cities include Jakarta, Bangkok, Taipei, Hong Kong, Tokyo, Beijing and Seoul. Ben decided to focus more on the possibilities of better return of investment in cities which have a positive economy and financial growth. So, Ben needs to do some analysis of the local economy and finance situation for the last 3 months. Furthermore, he needs to optimise his travel. He will give priority to cities with possible better investment return based on the analysis of local economic and financial situations. If the next nearest city to be visited has a less better economic and financial situation than any of the other cities, Ben will visit another city first provided that the difference of distance between the 2 cities is not more than 40% and the difference of sentiment analysis between the 2 cities is not less than 2%.

We have used several algorithms to solve the problems of the project. We also modified algorithms if we needed to and also used new algorithms outside our lecture notes. Appropriate required tools, functions and api's were used to get the required output.

All the members of our group contributed equally to finish the whole project. As there were 3 problems in total, each problem was assigned to 2 group members.

This report contains the description of all the algorithms, functions and tools used along with all the codes with their output.

Control flow graph



Tools

1. Jupyter Notebook

The Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations and narrative text. We have mainly used this because of the convenience it gives to code and visualize the code at the same time. It was mainly used for problem 2 and 3 and lastly, the whole project was integrated inside the notebook.

2. Pycharm

PyCharm is an integrated development environment used in computer programming, specifically for the Python language. We have mainly used it while developing the solution for problem 1 and initial implementation of problem 2 and 3.

3. gmplot

gmplot is a matplotlib-like interface to generate the HTML and javascript to render all the data the user would like on top of Google Maps. We have mainly used it to generate the two required maps.

4. geopy

“geopy” is a Python client for several popular geocoding web services. geopy makes it easy for Python developers to locate the coordinates of addresses, cities, countries, and landmarks across the globe using third-party geocoders and other data sources. We have mainly used to locate the cities.

5. NumPy

NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays. We have used numpy mainly to plot graphs for problem 2.

6. Itertools

Itertools is a module which implements a number of iterator building blocks inspired by constructs from APL, Haskell, and SML. Together, they form an ‘iterator algebra’ making it possible to construct specialized tools succinctly and efficiently in pure Python.” We used it mainly for problem 3 to find out all possible routes.

7. Matplotlib

Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python. We have used matplotlib to plot graphs in problem 2.

Algorithms

1. Travelling Salesman Algorithm

The travelling salesman algorithm (also called the travelling salesperson problem or TSP) asks the following question: "Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city and returns to the origin city?" It is an NP-hard problem in combinatorial optimization, important in theoretical computer science and operations research. We have used this as a reference along with brute force algorithm to find out the suggested journey for Ben with least distance travelled.

2. Brute force algorithm

The simplest algorithm for string matching is a brute force algorithm, where we simply try to match the first character of the pattern with the first character of the text, and if we succeed, try to match the second character, and so on; if we hit a failure point, slide the pattern over one character and try again. We have used tsp based on brute force algorithm to find out the suggested journey for Ben with least distance travelled.

3. Rabin Karp algorithm

Rabin-Karp algorithm is an algorithm used for searching/matching patterns in the text using a hash function. Unlike Naive string matching algorithm, it does not travel through every character in the initial phase rather it filters the characters that do not match and then performs the comparison. We modified the rabin-karp algorithm to come up with our own algorithm to find out the stopwords in problem 2.

4. Word-filter algorithm

We have managed to develop a small algorithm based on the word-filter algorithm. This algorithm creates a new list with all the words of which the lower-case variant is not found in stopwords. In our case it filters out the stopwords to find out the words that actually will have an impact in the sentiment that is needed for problem 2 and problem 3.

Solution

PROBLEM 1 :

1. Get and mark locations of all the cities Ben plans to visit.

Getting locations of all the cities that Ben plans to visit.

```python
# Problem 2 (Question 1) : Get and Mark the Location
import gmplot
from geopy.geocoders import Nominatim
geolocator = Nominatim(user_agent="PyCharm")

print("The cities that Ben Sherman will go from Kuala Lumpur are : ")
location = geolocator.geocode("Kuala Lumpur")
print(location.address, location.latitude, location.longitude)

location2 = geolocator.geocode("Jakarta")
print(location2.address, location2.latitude, location2.longitude)

location3 = geolocator.geocode("Bangkok")
print(location3.address, location3.latitude, location3.longitude)

location4 = geolocator.geocode("Taipei")
print(location4.address, location4.latitude, location4.longitude)

location5 = geolocator.geocode("Hong Kong")
print(location5.address, location5.latitude, location5.longitude)

location6 = geolocator.geocode("Tokyo")
print(location6.address, location6.latitude, location6.longitude)

location7 = geolocator.geocode("Beijing")
print(location7.address, location7.latitude, location7.longitude)

location8 = geolocator.geocode("Seoul")
print(location8.address, location8.latitude, location8.longitude)
```

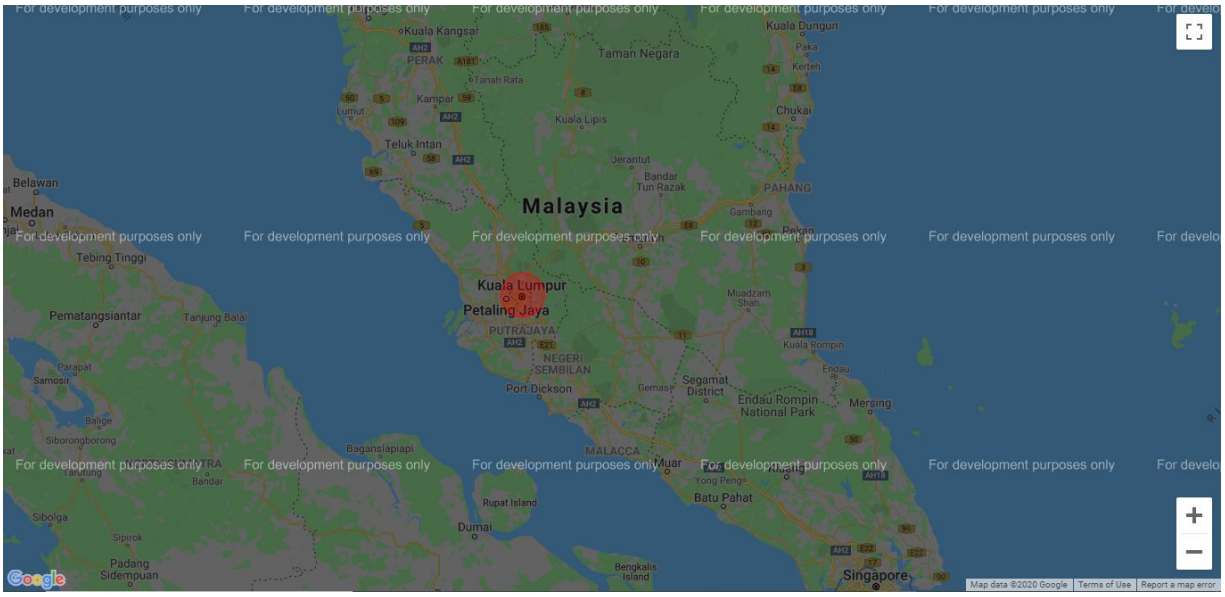
The output :

```
"C:\Users\Alia Husna\PycharmProjects\ProjectAH\venv\Scripts\python.exe" "C:/Users/Alia Husna/PycharmProjects/ProjectAH/Location.py"
The cities that Ben Sherman will go from Kuala Lumpur are :
Kuala Lumpur, Malaysia 3.1516964 101.6942371
Daerah Khusus Ibukota Jakarta, Indonesia -6.1753942 106.827183
กรุงเทพมหานคร, เขตพระนคร, กรุงเทพมหานคร, 10200, ประเทศไทย 13.7542529 100.493087
臺北市, 信義區, 臺北市, 11008, Taiwan 25.0375198 121.5636796
香港岛 Hong Kong Island, 香港 Hong Kong, China 中国 22.2793278 114.1628131
東京都, 日本 (Japan) 35.6828387 139.7594549
北京市, 东城区, 北京市, 100010, China 中国 39.906217 116.3912757
서울, 대한민국 37.5666791 126.9782914
```

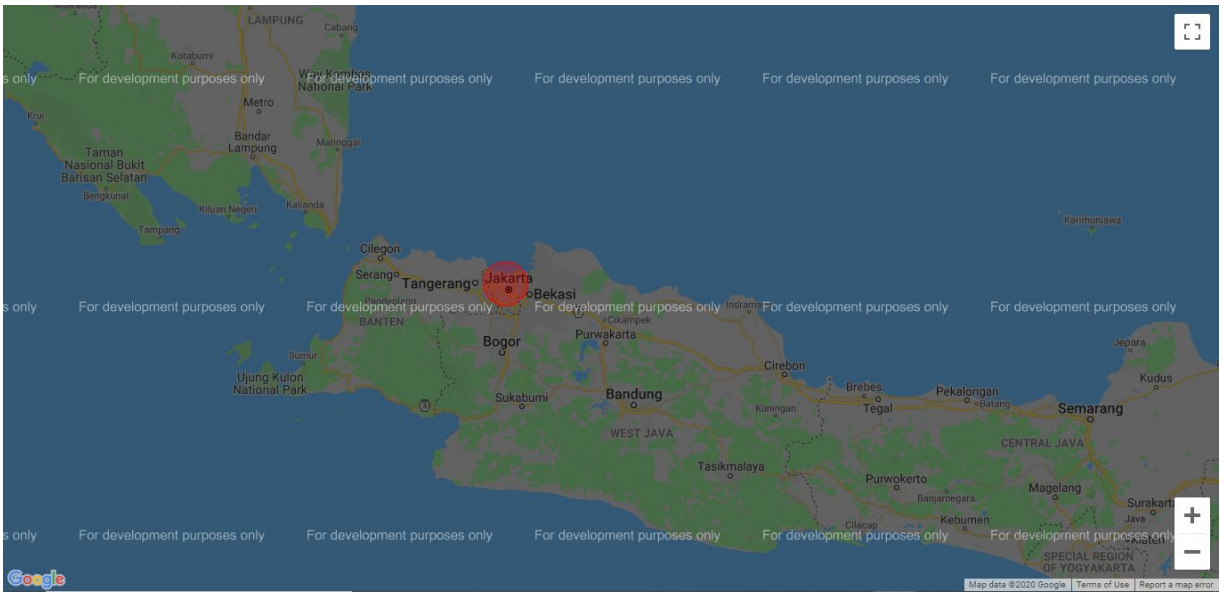
Mark all locations :

```python
#Mark location : Kuala Lumpur, Jakarta, Bangkok, Taipei, Hong Kong, Tokyo, Beijing, Seoul
lat = [3.1516964, -6.1753942, 13.7542529, 25.0375198, 22.2793278, 35.6828387, 39.906217, 37.5666791]
long = [101.6943271, 106.827183, 100.493087, 121.5636796, 114.1628131, 139.7594549, 116.3912757, 126.9782914]
# plot location : Jakarta
#lat2 = [-6.1753942]
#long2 = [106.827183]
gmapOne = gmplot.GoogleMapPlotter(3.1516964, 101.6943271, 5)
gmapOne.scatter(lat, long, 'red', size = 15000, marker=False)

#gmapTwo = gmplot.GoogleMapPlotter(-6.1753942,106.827183, 15)
#gmapTwo.scatter(lat2, long2, 'blue', size = 50, marker=False)


gmapOne.draw("map.html")
#gmapTwo.draw("map.html")



# output
#Kuala Lumpur, Malaysia 3.1516964 101.6942371
#Daerah Khusus Ibukota Jakarta, Indonesia -6.1753942 106.827183
#กรุงเทพมหานคร, เขตพระนคร, กรุงเทพมหานคร, 10200, ประเทศไทย 13.7542529 100.493087
#臺北市, 信義區, 臺北市, 11008, Taiwan 25.0375198 121.5636796
#香港島 Hong Kong Island, 香港 Hong Kong, China 中国 22.2793278 114.1628131
#東京都, 日本 (Japan) 35.6828387 139.7594549
#北京市, 东城区, 北京市, 100010, China 中国 39.906217 116.3912757
#서울, 대한민국 37.5666791 126.9782914
```

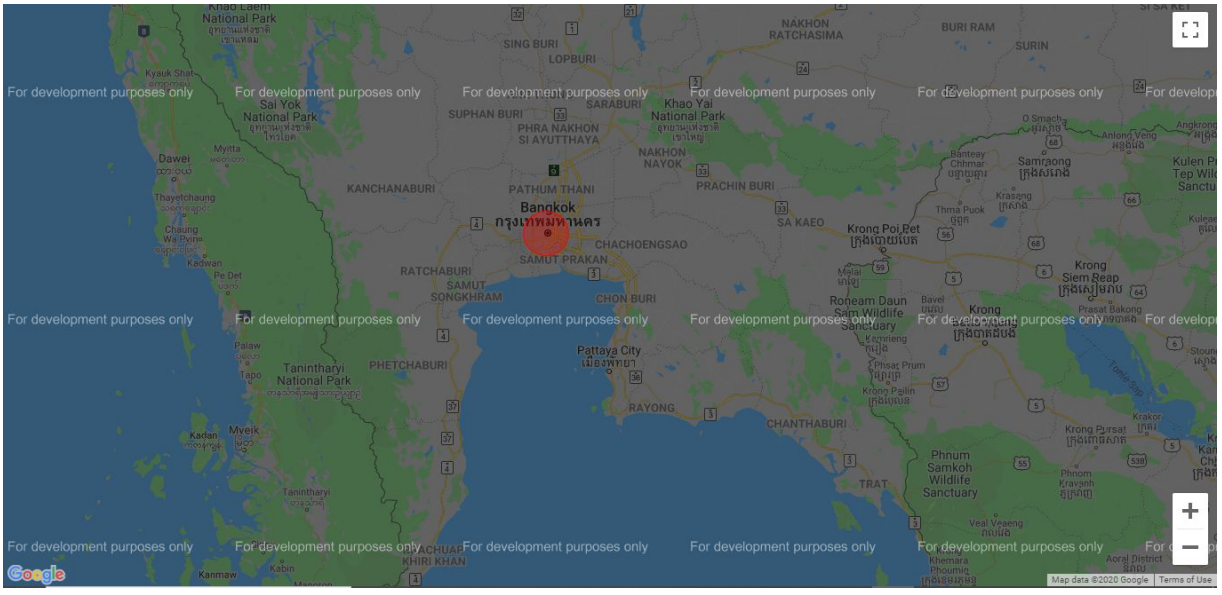
Mark on cities : Kuala Lumpur



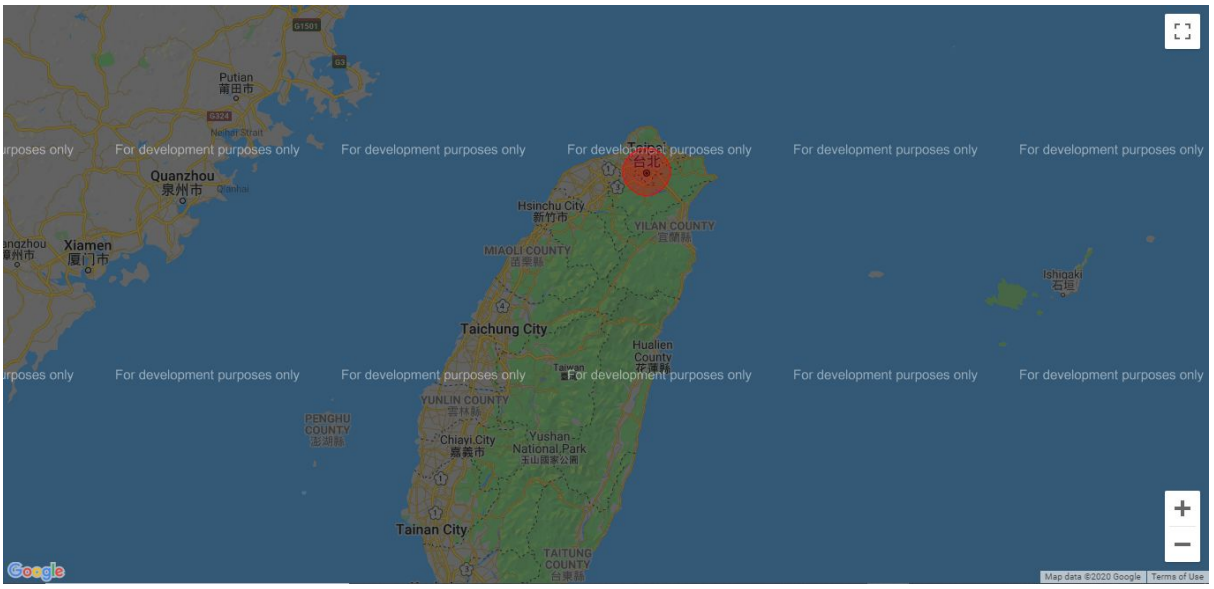
Mark on cities : Jakarta, Indonesia



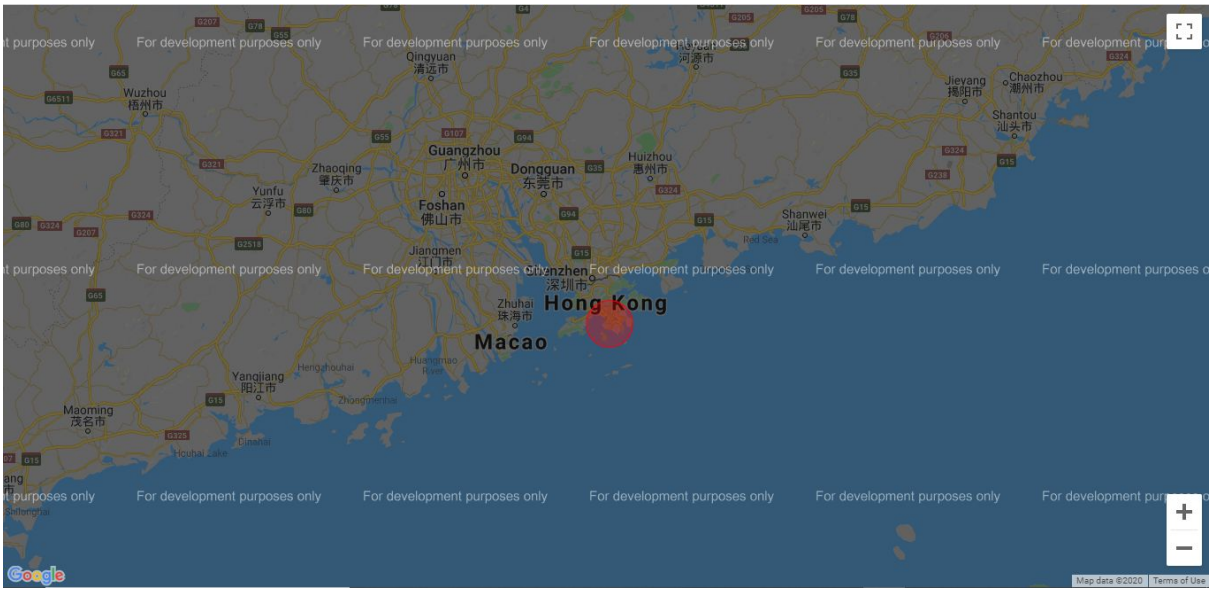
Mark on cities : Bangkok, Thailand



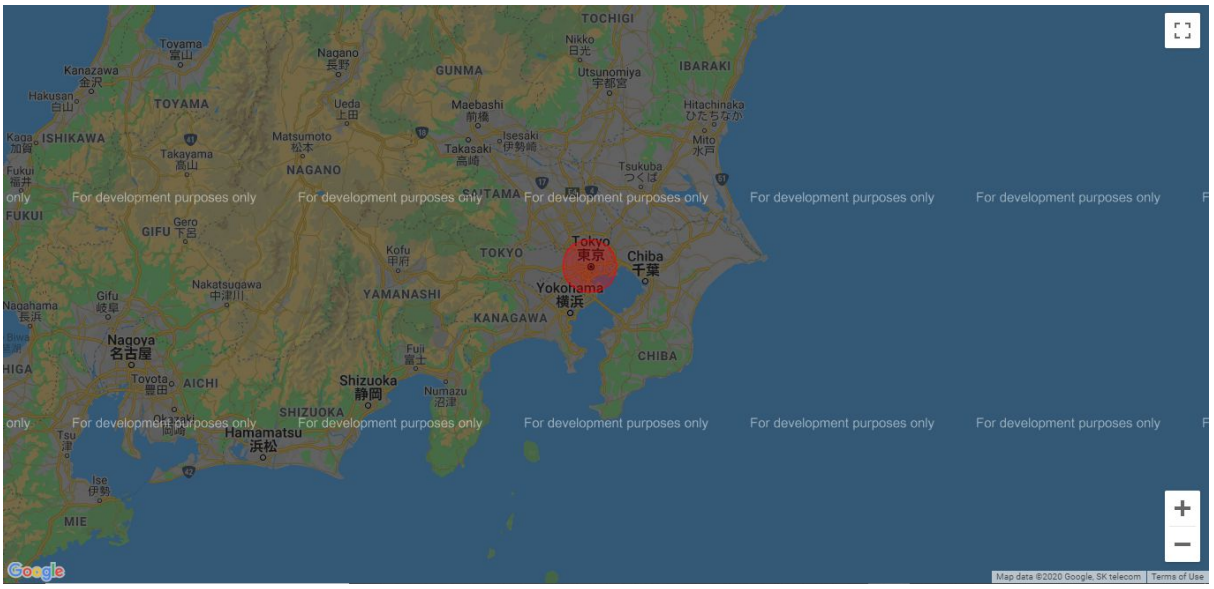
Mark on cities : Taipei, Taiwan



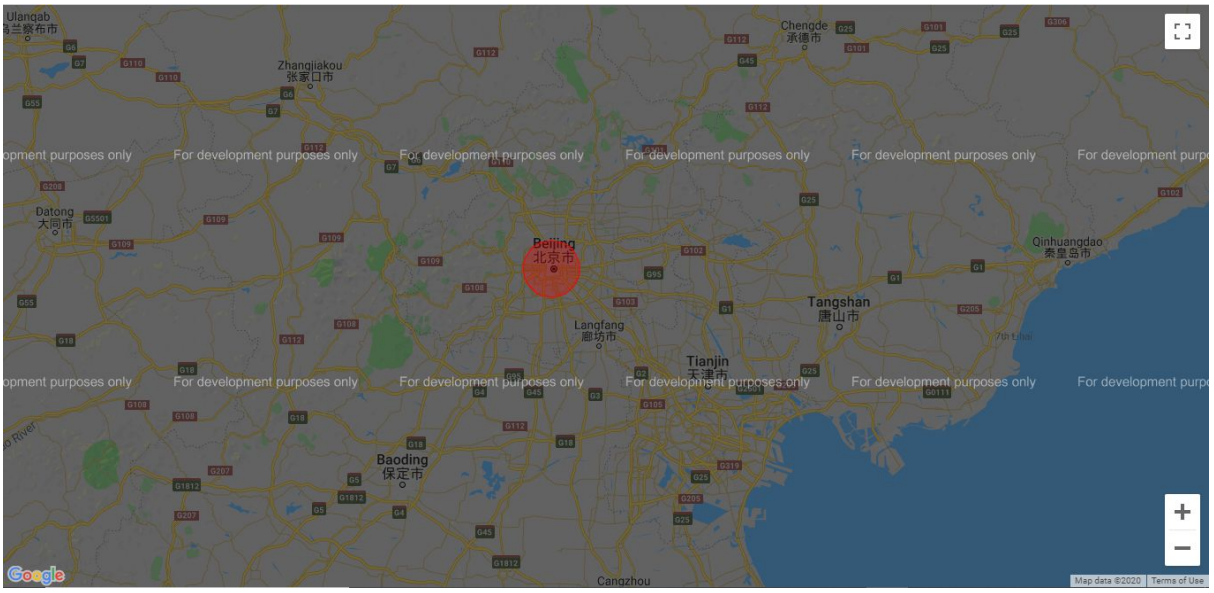
Mark on cities : Hong Kong, China



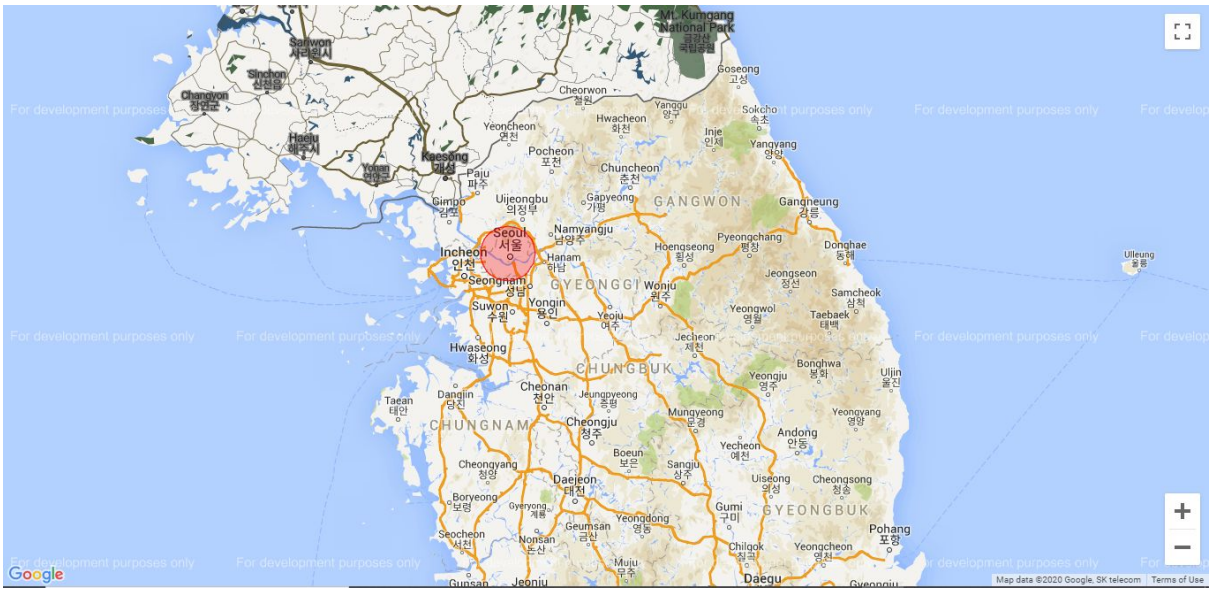
Mark on cities : Tokyo, Japan



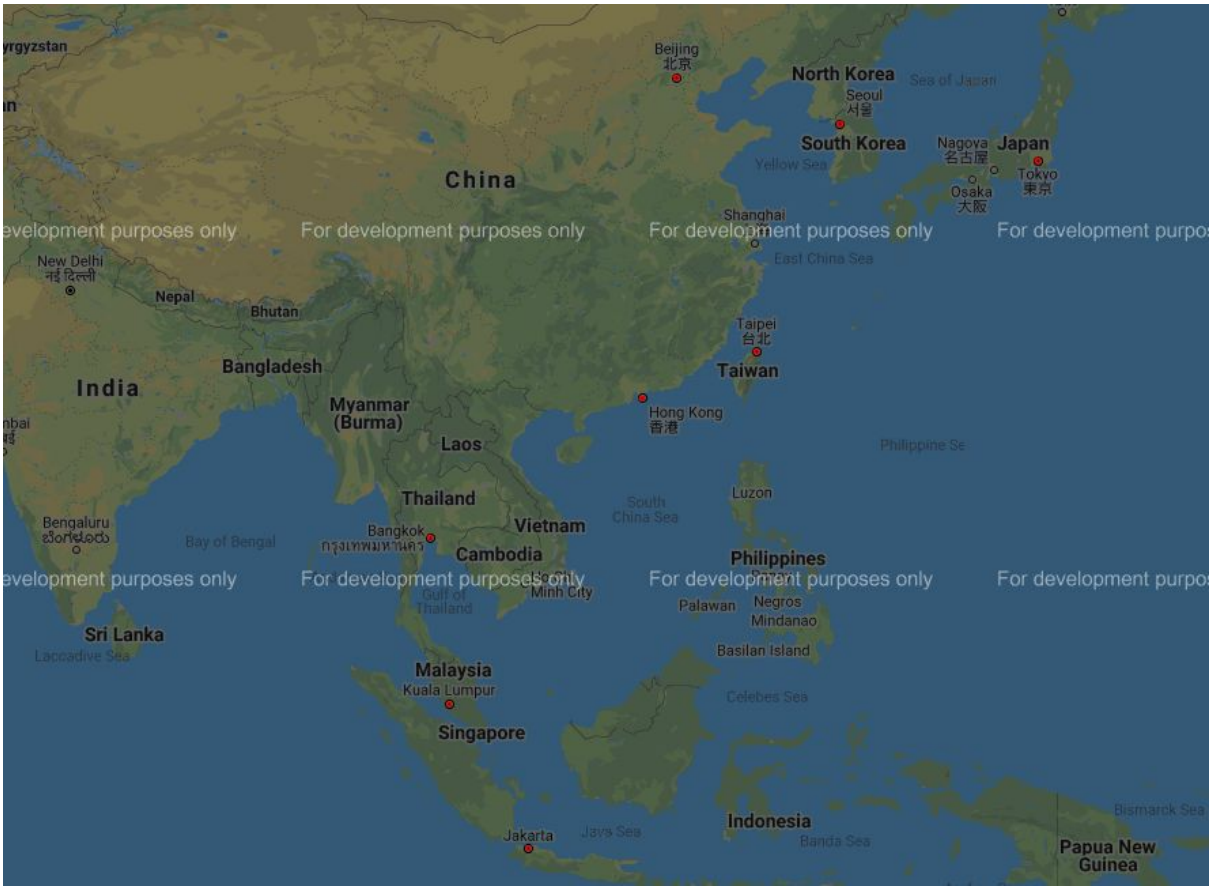
Mark on cities : Beijing, China



Mark on cities : Seoul, Korea



The mark for all cities (the red dot represent the cities that Ben plans to visit) :



2. Get the distances between each of these destinations.

Location.py | Distance.py

```
# Problem 2 (Question 2) : Getting distances between each of the destinations

from geopy.distance import geodesic
KualaLumpur_MAS = (3.1516964, 101.6942371)
Jakarta_INA = (-6.1753942, 106.827183)
Bangkok_THA = (13.7542529, 100.493087)
Taipei_TPE = (25.0375198, 121.5636796)
HongKong_HKG = (22.2793278, 114.1628131)
Tokyo_JPN = (35.6828387, 139.7594549)
Beijing_CHN = (39.906217, 116.3912757)
Seoul_KOR = (37.5666791, 126.9782914)

print("\nThe distance between Kuala Lumpur and Jakarta is : ", geodesic(KualaLumpur_MAS, Jakarta_INA).km)
print("\nThe distance between Kuala Lumpur and Bangkok is : ", geodesic(KualaLumpur_MAS, Bangkok_THA).km)
print("\nThe distance between Kuala Lumpur and Taipei is : ", geodesic(KualaLumpur_MAS, Taipei_TPE).km)
print("\nThe distance between Kuala Lumpur and Hong Kong is : ", geodesic(KualaLumpur_MAS, HongKong_HKG).km)
print("\nThe distance between Kuala Lumpur and Tokyo is : ", geodesic(KualaLumpur_MAS, Tokyo_JPN).km)
print("\nThe distance between Kuala Lumpur and Beijing is : ", geodesic(KualaLumpur_MAS, Beijing_CHN).km)
print("\nThe distance between Kuala Lumpur and Seoul is : ", geodesic(KualaLumpur_MAS, Seoul_KOR).km)
```

The output :

Distance

```
"C:\Users\Alia Husna\PycharmProjects\ProjectAH\venv\Scripts\python.exe" "C:/Users/Alia Husna/PycharmProjects/ProjectAH/Distance.py"

The distance between Kuala Lumpur and Jakarta is :  1178.671859673486

The distance between Kuala Lumpur and Bangkok is :  1180.0700698191881

The distance between Kuala Lumpur and Taipei is :  3224.777990540157

The distance between Kuala Lumpur and Hong Kong is :  2508.4362660368365

The distance between Kuala Lumpur and Tokyo is :  5318.677216353381

The distance between Kuala Lumpur and Beijing is :  4332.215068712133

The distance between Kuala Lumpur and Seoul is :  4601.871799273804

Process finished with exit code 0
```

3. Journey planner: Suggest a journey for Ben to visit each of the cities once with the least distance travelled.

Code:

```
routes = []

#TSP based on brute force
def find_paths(node, cities, path, distance):
    # Add way point
    path.append(node)
```

```
    # Calculate path length from current to last node
    if len(path) > 1:
        distance += cities[path[-2]][node]

    # If path contains all cities and is not a dead end,
    # add path from last to first city and return.
    if (len(cities) == len(path)) and (path[0] in cities[path[-1]]):
        global routes
        path.append(path[0])
        distance += cities[path[-2]][path[0]]
        routes.append([distance, path])
        return

    # Fork paths for all possible cities not yet used
    for city in cities:
        if (city not in path) and (node in cities[city]):
            find_paths(city, dict(cities), list(path), distance)


if __name__ == '__main__':
    cities = {
                'KL': {'KL': 0, 'JK': 1178.6718596734863, 'BK': 1180.
 ↪0700698191881, 'TAI': 3224.7779905401576, 'HK': 2508.4362660368365,'BEI':␣
 ↪4332.215068712132, 'TOK': 5318.677216353379, 'SEO': 4601.871799273804},
                'JK': {'KL': 1178.6718596734863, 'JK': 0, 'BK': 2312.
 ↪509200540841, 'TAI': 3804.2538748562656, 'HK': 3247.6059266254892,'BEI':␣
 ↪5195.7631594758395, 'TOK': 5773.098750995338, 'SEO': 5274.932678827154},
                'BK': {'KL': 1180.0700698191881, 'JK': 2312.509200540841, 'BK':␣
 ↪0, 'TAI': 2535.8957177665748, 'HK': 1726.1824898477596, 'BEI': 3288.
 ↪6020977412145, 'TOK': 4610.121572373538, 'SEO': 3719.354478303037},
                'TAI': {'KL': 3224.7779905401576, 'JK': 3804.2538748562656,␣
 ↪'BK': 2535.8957177665748, 'TAI': 0, 'HK': 814.2973250648987, 'BEI': 1718.
 ↪2312737282145, 'TOK':  2104.313098157768, 'SEO': 1480.973652900838},
                'HK': {'KL': 2508.4362660368365, 'JK': 3247.6059266254892, 'BK':
 ↪ 1726.1824898477596, 'TAI': 814.2973250648987, 'HK': 0, 'BEI': 1965.
 ↪7256268618175, 'TOK': 2889.715055526659, 'SEO': 2093.6715546233295},
                'BEI': {'KL': 4332.215068712132, 'JK': 5195.7631594758395, 'BK':
 ↪ 3288.6020977412145, 'TAI': 1718.2312737282145, 'HK': 1965.7256268618175,␣
 ↪'BEI': 0, 'TOK': 2104.3499748042273, 'SEO': 955.7689217292321},
                'TOK': {'KL': 5318.677216353379, 'JK': 5773.098750995338, 'BK':␣
 ↪4610.121572373538, 'TAI': 2104.313098157768, 'HK': 2889.715055526659, 'BEI':␣
 ↪2104.3499748042273, 'TOK': 0, 'SEO': 1161.2277477992284},
                'SEO': {'KL': 4601.871799273804, 'JK': 5274.932678827154, 'BK':␣
 ↪3719.354478303037, 'TAI': 1480.973652900838, 'HK': 2093.6715546233295, 'BEI':
 ↪ 955.7689217292321, 'TOK': 1161.2277477992284, 'SEO': 0}
            }
```

```
find_paths('KL', cities, [], 0)
print("\n")
routes.sort()
if len(routes) != 0:
    print("Shortest distance: %s" % round(routes[0][0],3)+" km")
    print("Shortest route based on distance: %s" % routes[0][1])
else:
    print("FAIL!")
```

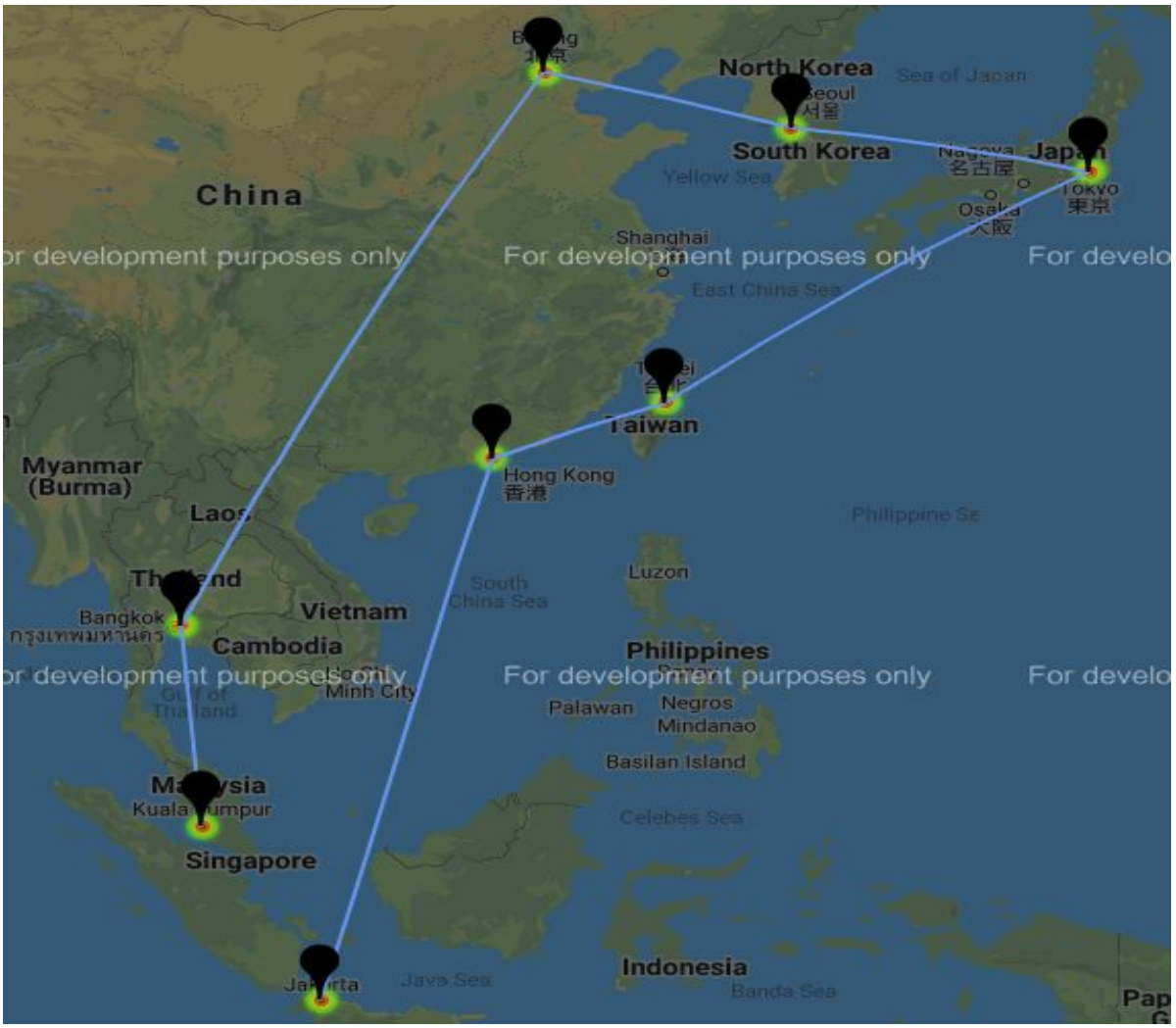
Output:

```
Shortest distance: 13930.557 km
Shortest route based on distance: ['KL', 'BK', 'BEI', 'SEO', 'TOK', 'TAI', 'HK', 'JK', 'KL']
```

4. Plot line between the destinations.

```
import gmplot

gmap = gmplot.GoogleMapPlotter(3.1516964, 101.6942371, 13)

kl = (3.1516964, 101.6942371)
j = (-6.1753942, 106.827183)
b = (13.7538929, 100.8160803)
t = (25.0375198, 121.5636796)
h = (22.2793278, 114.1628131)
be = (39.906217, 116.3912757)
to = (35.6828387, 139.7594549)
s = (37.5666791, 126.9782914)

lat_list = [3.1516964, 13.7538929, 39.906217, 37.5666791, 35.6828387, 25.0375198, 22.2793278,
            -6.1753942]
lon_list = [101.6942371, 100.8160803, 116.3912757, 126.9782914, 139.7594549,
           121.5636796, 114.1628131, 106.827183]

gmap.heatmap(lat_list, lon_list)
gmap.plot(lat_list, lon_list, "cornflowerblue", edge_width=2.5)
gmap.scatter(lat_list, lon_list, size=40, marker=True)
gmap.draw("map2.html")
```

The output:



** We have considered the situation where Ben’s travelling route includes him returning to Kuala Lumpur, Malaysia. So the shortest path we found is also considering this factor.

Problem 2

5. Extract information from major economic/financial news websites for each city. You need to find 5 related articles within the last 3 months to be analysed.

6. Plot line/scatter/histogram graphs related to the word count using Plotly (Word count, stop words)

```
#Problem 2
```

```
import numpy as np
import matplotlib.pyplot as plot; plot.rcdefaults()
```

```
#Stopwords' file access
stopwords = open("stopwords.txt", 'r')
stopwords = stopwords.read().splitlines()
```

```
#An algorithm based on rabin karp is made to find stopwords from articles
def stopwordsearch(stopword, articles):
    sword = len(stopword)
    art = len(articles)
    count = 0

    #Search for stopwords in articles
    for i in range(0, art - sword + 1):
        found = True
        for j in range(0, sword):
            if stopword[j] != articles[i + j]:
                found = False
                break
        if found:
            count += 1

    #Printing out the stopwords' appearance.
    if count > 0:
        print(stopword,':', count, 'times.')
    else:
        None


#Filtering stopwords from articles to do the count.
def stopwordfilter(filepath):
    #File handling
    article = open(filepath, encoding="utf8")
    article = article.read().splitlines()

    #Search
    for x in article:
        for i in stopwords:
            stopwordsearch(i, x)
```

```
def wordcountgraph(filepath):
    #File handling
    file = open(filepath, encoding="utf8")
    file = file.read()
    number_of_characters = len(file)

    #Printing the number of words
    print('Total number of words in the article:', number_of_characters)
    allwords = file.split()

    #Filtering out the stopwords from 'allwords' to plot the graph
    words = [word for word in allwords if word.lower() not in stopwords]
    print('Number of words that is going to be used for finding out the␣
 ↪economic sentiment:',len(words))

    #Creating an empty dictionary to store word count
    dic = {}
    wordcount = []
    for wordcount in words:
        dic[wordcount] = dic.get(wordcount, 0) + 1
    print()
```

```
    #For words (x axis)
    worda = []
    #For count(y axis)
    countb = []

    #Storing in  list a and countb
    for key, value in dic.items():
        worda.append(key)
        countb.append(value)

    #Length of x axis based on number words
    x = np.arange(len(worda))

    #Setting bars of words based on count
    plot.bar(x, countb)

    #Counts on y axis
    plot.yticks(fontsize=10)

    #Adding values to x axis
    plot.xticks(x, worda)

    #Rotation of values
    plot.xticks(rotation=90)

    #Labels
    plot.xlabel('Words', fontsize=30)
    plot.ylabel('Count', fontsize=30)

    #Size
    plot.rcParams['figure.figsize'] = (40, 20)
```

Output:

For example in case of Bangkok:

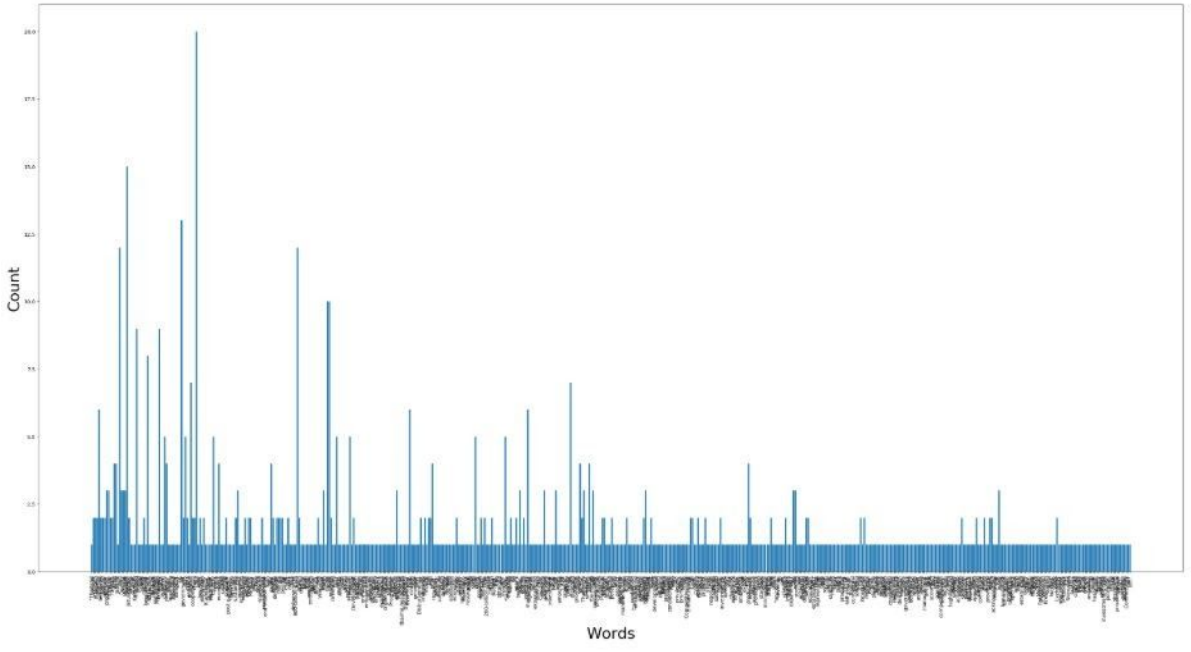
```
#Bangkok
```

```
stopwordfilter("Bangkok.txt")
```

```
a : 157 times.
again : 1 times.
against : 1 times.
all : 1 times.
am : 5 times.
an : 48 times.
and : 26 times.
any : 1 times.
are : 3 times.
as : 10 times.
at : 22 times.
be : 5 times.
been : 1 times.
being : 2 times.
but : 1 times.
by : 7 times.
during : 1 times.
each : 1 times.
for : 7 times.
from : 2 times.
has : 3 times.
```

```
wordcountgraph('Bangkok.txt')
```

```
Total number of words in the article: 8630
Number of words that is going to be used for finding out the economic sentiment: 848
```



In case of Beijing:

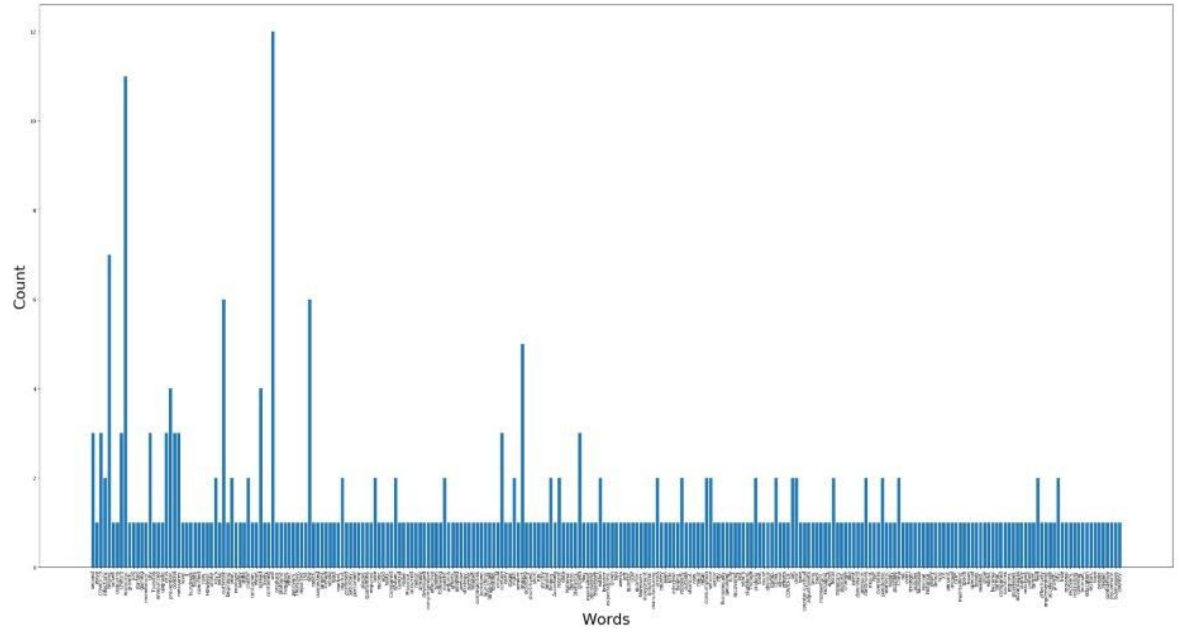
```
#Beijing
```

```
stopwordfilter("Beijing.txt")
```

```
a : 17 times.
about : 1 times.
an : 5 times.
and : 2 times.
are : 1 times.
as : 3 times.
from : 1 times.
has : 2 times.
he : 1 times.
i : 18 times.
in : 7 times.
is : 1 times.
it : 2 times.
its : 1 times.
me : 2 times.
no : 3 times.
of : 3 times.
on : 6 times.
once : 1 times.
out : 1 times.
```

```
wordcountgraph('Beijing.txt')
```

```
Total number of words in the article: 3498
Number of words that is going to be used for finding out the economic sentiment: 343
```



**Same type of output generated for other countries.

7. Compare words in the webpages with the positive, negative and neutral English words using a StringMatching algorithm.

8. Plot histogram graphs of positive and negative words found in the webpages.

9. Give an algorithmic conclusion regarding the sentiment of those articles.

```
#File handling for positive.txt and negative.txt
positivewords = open("positive.txt", 'r')
positivewords = positivewords.read().splitlines()
negativewords = open("negative.txt", 'r')
negativewords = negativewords.read().splitlines()
```

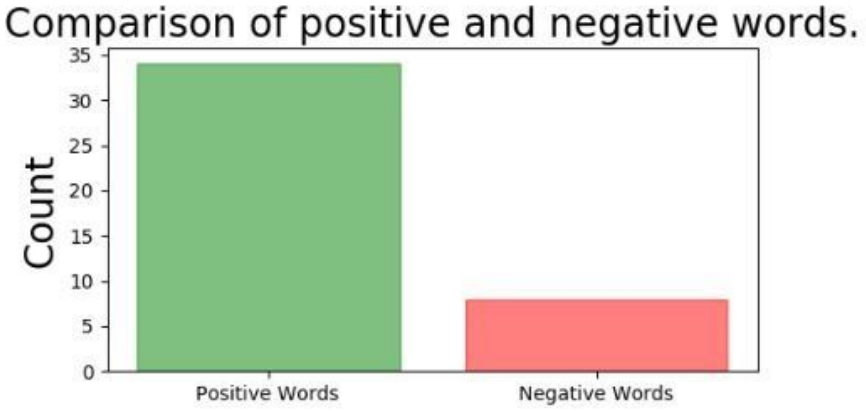
```python
def positive_negative_comparison(filepath):

    #file handling
    article = open(filepath, encoding="utf8")
    article = article.read()
    number_of_characters = len(article)

    #Initializing count variables for negative, positive and neutral words.
    poscount = 0
    negcount = 0
    neucount = 0

    #Search for positive and negative words in the articles.
    for word in article.split():
        if word in positivewords:
            poscount = poscount+1
        elif word in negativewords:
            negcount = negcount+1

    #Calculating neutral count based on the given condition.
    neucount = number_of_characters - (poscount+negcount)

    #Results of positive, negative and neutral count.
    print("Positive words: ",poscount)
    print("Negative words: ",negcount)
    print("Neutral words: ",neucount)

    #Checking the positive-negative status for the article and coming to a conclusion based on the result.
    if poscount > negcount:
        print("The article is giving positive sentiment.")
        print("This country has a positive economic and financial situation.")
    elif poscount < negcount:
        print("The article is giving negative sentiment.")
        print("This country has a negative economic and financial situation.")
    else:
        print("The article is giving neutral sentiment.")
        print("This country has a neutral economic and financial situation.")

    #Assigning names of the participants of the histogram graph.
    determiners = ("Positive Words", "Negative Words")
```

```python
    #Length of x axis based on the length of the determiners.
    x = np.arange(len(determiners))

    #Assigning counts as determiners' values.
    det_values = [poscount, negcount]

    #Setting bars based on the values.
    barlist = plot.bar(x, det_values, align='center', alpha=0.5)

    #Decorations
    barlist[0].set_color('green')
    barlist[1].set_color('red')

    #Adding values to x axis
    plot.xticks(x, determiners)

    #Decorations
    plot.yticks(fontsize=10)
    plot.xticks(fontsize=10)

    #Label
    plot.ylabel("Count",fontsize = 20)

    #Title
    plot.title("Comparison of positive and negative words.",fontsize = 20)

    #Size
    plot.rcParams['figure.figsize'] = (6,3)
```

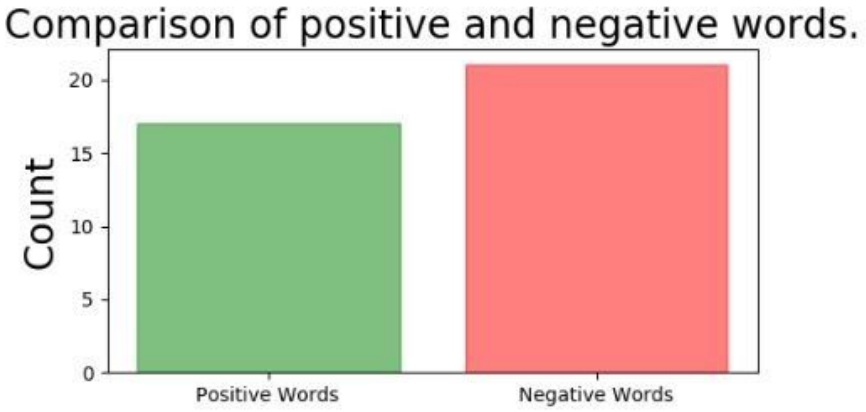
Sample output:

For bangkok and beijing:

```
In [36]: 1 #Bangkok

In [41]: 1 positive_negative_comparison('Bangkok.txt')

Positive words:  34
Negative words:  8
Neutral words:  8588
The article is giving positive sentiment.
This country has a positive economic and financial situation.
```




```
In [42]: 1 #Beijing

In [43]: 1 positive_negative_comparison('Beijing.txt')

Positive words:  17
Negative words:  21
Neutral words:  3460
The article is giving negative sentiment.
This country has a negative economic and financial situation.
```



**Same type of output is generated for every city.

PROBLEM 3

Calculate the total probability distribution of possible routes. Then, write the summary of all possible routes for Ben to take, ranking from the most recommended to the least recommended.

```
#Problem 3
import itertools #Itertools for possible routes
```

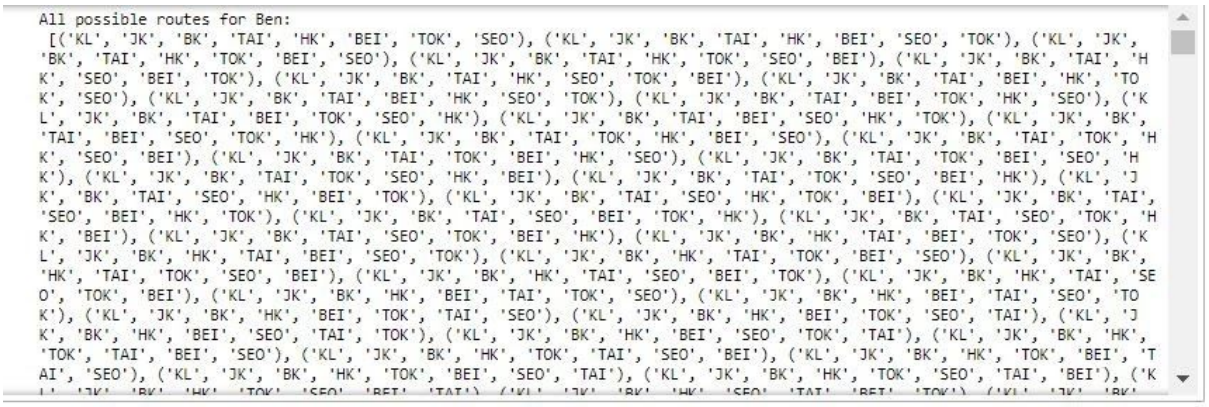
```
#calculating final sentiment percentage for finding out probability of a possible route
def calc_sent(filepath):
    #file handling
    file = open(filepath, encoding="utf8")
    file = file.read()
    number_of_characters = len(file)
    allwords = file.split()

    #Filtering out the stopwords again to calculate the probabilty of the route
    words = [word for word in allwords if word.lower() not in stopwords]

    #Number of words responsible the economic sentiment
    n_of_rwords = len(words)

    #positive and negative count
    negativeCount = 0
    positiveCount = 0
    calcCount =0
    for part in file.split():
        if part in positivewords:
            positiveCount += 1
        elif part in negativewords:
            negativeCount += 1

    #calculating final sentiment percentage
    calcCount = positiveCount - negativeCount
    per_sent_city = (calcCount/n_of_rwords)*100
    return per_sent_city

#storing final sentiment percentage in variables
sent_jk = calc_sent("Jakarta.txt")
sent_bei = calc_sent("Beijing.txt")
sent_bk = calc_sent("Bangkok.txt")
sent_hk = calc_sent("Hongkong.txt")
sent_seo = calc_sent("Seoul.txt")
sent_tp = calc_sent("Taipei.txt")
sent_tok = calc_sent("Tokyo.txt")

#get all possible path
def possiblePath(citiesList):
    allPath = list(itertools.permutations(citiesList))
    return allPath
path = list(possiblePath(cities))
print("All possible routes for Ben: \n",path,"\n")

#Probability of a possible route
prob = (sent_jk+sent_bei+sent_bk+sent_hk+sent_seo+sent_tp+sent_tok)
print('The probability of a possible route: ',round(prob,3),'%')
```

```
#storing locations in list
location = []
location.append([13.7538929,100.8160803]) #Bangkok
location.append([39.906217,116.3912757]) #Beijing
location.append([22.2793278,114.1628131]) #Hongkong
location.append([-6.1753942,106.827183]) #Jakarta
location.append([37.5666791,126.9782914]) #Seoul
location.append([25.0375198,121.5636796]) #Taiwan
location.append([35.6828387, 139.7594549]) #Tokyo
location.append([3.140853,101.693207]) #Kuala-Lumpur

#calculating distance to check condition
def distance(x,y,x1,y1):
    firstcity = (x, y)
    secondcity = (x1, y1)
    value = geodesic(firstcity, secondcity).km
    return value

#listing cities to use as keys
cities = ['Bangkok','Beijing','Hongkong','Jakarta','Seoul','Taipei','Tokyo']

#storing sentiment results in list
sentimentlist = []
sentimentlist.append(sent_bk)
sentimentlist.append(sent_bei)
sentimentlist.append(sent_hk)
sentimentlist.append(sent_jk)
sentimentlist.append(sent_seo)
sentimentlist.append(sent_tp)
sentimentlist.append(sent_tok)

#minimum sentiment to be the best choice
minrange_for_best = max(sentimentlist)

#the coordinate of each cities
coordinate = {}
for i in range(len(cities)):
    coordinate[cities[i]] = location[i]

#storing distance for final calculation
for name in coordinate:
    for i in cities:
        dist = distance(coordinate[name][0],coordinate[name][1],coordinate[i][0],coordinate[i][1])
        #print(i,dist)

#get the sentiment analysis of each city
analysis = {}
for i in range(len(cities)):
    analysis[cities[i]] = sentimentlist[i]
print('Economic sentiment for each city: ',analysis)

```

```python
#check sentiment analysis for first city to check if it is best or not
def check(analysis):
    minimum = minrange_for_best
    for city in analysis:
        if(analysis[city] >= minimum):
            minimum = analysis[city]
            best = city
    return best
best = check(analysis)
#print(best)

#check sentiment analysis for next city
def checkNext(analysis,cities):
    minimum = minrange_for_best
    for city in analysis:
        if(analysis[city] >= minimum and city in cities):
            minimum = analysis[city]
            best = city
    return best

#Verifying condition for 1st city
sentiment = minrange_for_best
def condition(coordinate, dist , city, nearest, sentiment, route, analysis):
    #Initializing given conditions
    minDiff = 2
    pathLength = 0.4 * dist
    bestCity = None

    #Iteration to verify with the conditions
    for name in coordinate:
        if (name != city and name != nearest and name not in route):
            value = distance(coordinate[name][0],coordinate[name][1],coordinate[city][0],coordinate[city][1])
            if(value < pathLength and abs(sentiment-analysis[name])>= minDiff):
                pathLength = value
                minDiff = abs(sentiment-analysis)
                bestCity = name
    return bestCity
```

```python
#Verifying condition for next cities
def nextCity(city,coordinate,analysis,route):
    #using a bigger value to store the next calculated distance as minimum
    minimum = 10000
    for name in coordinate:
        if(name != city and name not in route):
            dist = distance(coordinate[name][0],coordinate[name][1],coordinate[city][0],coordinate[city][1])
            if (dist < minimum):
                minimum = dist
                nearest = name
    #print(nearest)
    if nearest != checkNext(analysis,route):
        best = condition(coordinate,minimum,city,nearest,analysis[nearest],route,analysis)
        if(best == None):
            best = nearest
    else:
        best = nearest
    return best

#get the recommended path
def recommendPath(cities,analysis,coordinate):
    route = []
    route.append(check(analysis))
    while(len(route) != len(cities)):
        route.append(nextCity(route[-1],coordinate,analysis,route))
    return route

bestpath = recommendPath(cities,analysis,coordinate)
temp = bestpath.copy()
temp.reverse()
worstpath = temp.copy()
print("The most recommended path for Ben to take based on distance and sentiment:",bestpath)
print("The least recommended path for Ben to take based on distance and sentiment:",worstpath)
```

Output:




```
The probability of a possible route:  1.104 %
Economic sentiment for each city:  {'Bangkok': 3.0660377358490565, 'Beijing': -1.1661807580174928, 'Hongkong': 0.15015015015
015015, 'Jakarta': -0.42253521126760557, 'Seoul': 0.4178272980501393, 'Taipei': 1.2779552715654952, 'Tokyo': -2.218934911242
6036}
The most recommended path for Ben to take based on distance and sentiment: ['Bangkok', 'Hongkong', 'Taipei', 'Seoul', 'Beiji
ng', 'Tokyo', 'Jakarta']
The least recommended path for Ben to take based on distance and sentiment: ['Jakarta', 'Tokyo', 'Beijing', 'Seoul', 'Taipe
i', 'Hongkong', 'Bangkok']
```

Conclusion

For problem 1, the first purpose was to get and mark the location. We mainly used gmplot along with geopy to mark the locations. The next part required us to find out the route with the least distance. The main algorithm used was TSP based on brute force algorithm. Then we plotted the route by GoogleMapPlotter.

For problem 2, The main purpose was to find out the positive-negative sentiments of a country. To achieve that goal, we had to filter out the stop words first and then used the words that possessed an impact on the sentiment to come up with a conclusion and plot the graphs. The main algorithm used to filter out the stopwords was a variation of the Rabin-karp algorithm which we came up with. Because of the covid-19 situation we got somewhat unorthodox answers as the some cities that we assumed to have better investment opportunities for Ben were not recommended for the ongoing situation of the world. We have used numpy and matplotlib to complete this task.

Lastly for problem 3, we were required to calculate the total probability distribution of possible routes. Itertools was used to find out all possible routes and a formula was used to generate the probability of a route which was 1.104 %. Then lastly, we had to show the most and least recommended routes for ben. At first we needed to find out the sentiment percentage for each city. While doing so, we used a version of a word filter algorithm to store the words that had a meaningful impact on the sentiment. Then analyzing both distance and sentiment we generated the most recommended and least recommended route for ben. The most recommended route for Ben is The most recommended path for Ben to take based on distance and sentiment: ['Bangkok', 'Hongkong', 'Taipei', 'Seoul', 'Beijing', 'Tokyo', 'Jakarta']
The least recommended path for Ben to take based on distance and sentiment: ['Jakarta', 'Tokyo', 'Beijing', 'Seoul', 'Taipei', 'Hongkong', 'Bangkok']

References

Dishashree GuptaDishashree is passionate about statistics and is a machine learning enthusiast. She has an experience of 1.5 years of Market Research using R. (2019, July 15). Basics of Probability for Data Science explained with examples. Retrieved June 26, 2020, from https://www.analyticsvidhya.com/blog/2017/02/basic-probability-data-science-with-examples/

NumPy - Matplotlib. (n.d.). Retrieved June 23, 2020, from https://www.tutorialspoint.com/numpy/numpy_matplotlib.htm

Python 3 Interactive Course. (n.d.). Retrieved June 12, 2020, from https://snakify.org/en/lessons/two_dimensional_lists_arrays/

Python Advanced Course Topics. (n.d.). Retrieved June 18, 2020, from https://www.python-course.eu/graphs_python.php

Robby Cornelissen. (2019, July 19). Removing list of words from a string. Retrieved June 19, 2020, from https://stackoverflow.com/questions/25346058/removing-list-of-words-from-a-string

Singh, S. (2020, January 17). A quick review of Numpy and Matplotlib. Retrieved June 30, 2020, from https://towardsdatascience.com/a-quick-review-of-numpy-and-matplotlib-48f455db383

Travelling salesman problem. (2020, June 26). Retrieved June 22, 2020, from https://en.wikipedia.org/wiki/Travelling_salesman_problem

Travelling Salesman Problem: Set 1 (Naive and Dynamic Programming). (2018, September 06). Retrieved June 22, 2020, from https://www.geeksforgeeks.org/travelling-salesman-problem-set-1/