Summer
2019

Laboratory 5 - Drone Delivery Service

CS 2302 – DATA STRUCTURES SUMMER 2019

DILAN RAMIREZ

INSTRUCTOR: OLAC FUENTES

TEACHING ASSISTANTS (TA): ISMAEL VILLANUEVA-MIRANDA

Introduction

A company wants to establish a drone service to deliver packages from city to city. Your task is to implement algorithms to help them plan the set of flights that will be needed to make sure packages can be delivered from every city to every other city they serve while minimizing flight distances. The file cities.xlsx contains the names of the cities to be served and the flight distance from each pair of cities from which it is feasible to fly a drone; if flying a drone between two cities is not feasible, the distance appears as -1. For example, the “drone distance” from El Paso to Austin is 851, from Dallas to Las Vegas it’s 1700, and it is not feasible to fly a drone from Laredo to San Antonio.

1. Write a method to read the cities.xlsx file and build an adjacency list representation of the distance matrix.
2. Write a method to find and display the shortest path from every city to every other city, displaying the sequence of cities (their names, not their numbers) to visit as well as the overall distance. To do this, you need to implement Dijktra’s algorithm and run it once for every city.
3. Now suppose we want to find the set of flights with minimum combined cost that will still make it possible to send packages from a city to any other city. Write a function to find this set by applying Kruskal’s minimum spanning tree algorithm.

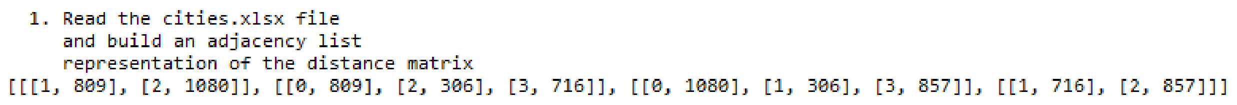
Implementation

The first step was to read the excel file which contains all the cities and their distance between each other. To do so, I use the python native object called “pandas” and my function called “readFile.” This function first opens the file and creates a list with all the file’s content using the function “to_list()”. This function returns that list. After that, I called my function “BuildAdjacencyList()” and as its name says it, it builds an adjacency list of the list created from the excel file. This function returns the adjacency list. After that I proceeded to the next step which is the step number 2. To do so, I wrote a function called “executeDijkstraAlgorithm ()” which will run the Dijkstra algorithm to find the shortest path from every city to every other city. In step number three I used the Kruskal algorithm to find the minimum spanning tree of all the cities using my function “runPrimsPart1()” Using this function I use the DataFrame from pandas module to print the minimum spanning tree. Finally, for the last step I used my function “executeKruskalAlgorithm()” to reduce the number of flights.

Experimental results

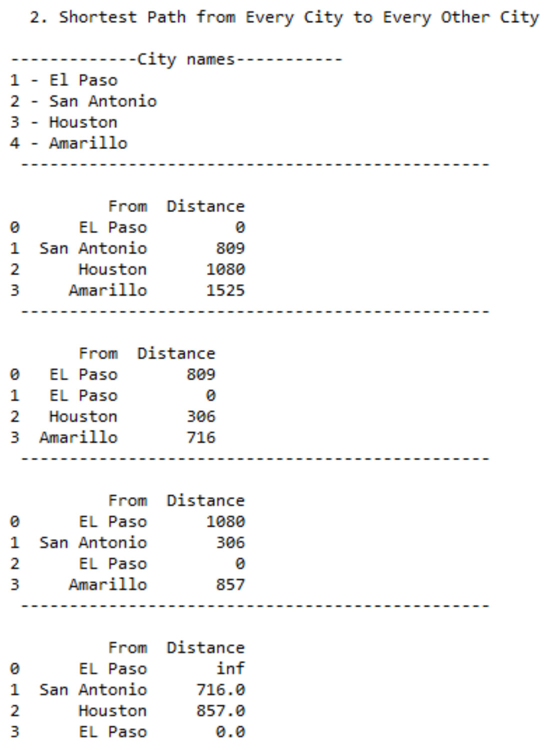
For the sake of the experimental results, I will only use part of the excel file because of its length. This file only takes the cities of EL Paso, San Antonio, Houston and Amarillo.

1. Write a method to read the cities.xlsx file and build an adjacency list representation of the distance matrix.



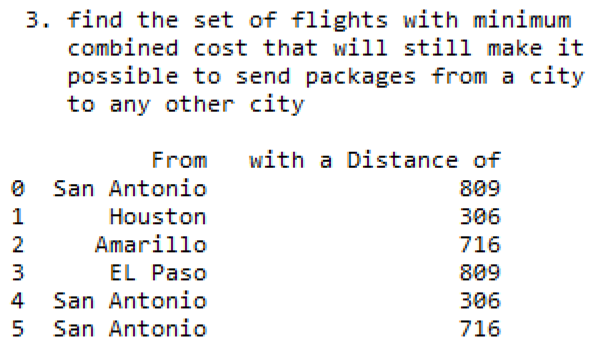
```
  1. Read the cities.xlsx file
     and build an adjacency list
     representation of the distance matrix
[[[1, 809], [2, 1080]], [[0, 809], [2, 306], [3, 716]], [[0, 1080], [1, 306], [3, 857]], [[1, 716], [2, 857]]]
```

2. Display the shortest path from every city to every other city, displaying the sequence of cities to visit as well as the overall distance. The sequence is from El Paso to every other city, San Antonio to every other city, Houston to every other city and Amarillo.



```
  2. Shortest Path from Every City to Every Other City

------------City names-----------
1 - El Paso
2 - San Antonio
3 - Houston
4 - Amarillo
 ------------------------------------------------

          From  Distance
0      EL Paso         0
1  San Antonio       809
2      Houston      1080
3     Amarillo      1525
 ------------------------------------------------

       From  Distance
0   EL Paso       809
1   EL Paso         0
2   Houston       306
3  Amarillo       716
 ------------------------------------------------

          From  Distance
0      EL Paso      1080
1  San Antonio       306
2      EL Paso         0
3     Amarillo       857
 ------------------------------------------------

          From  Distance
0      EL Paso       inf
1  San Antonio     716.0
2      Houston     857.0
3      EL Paso       0.0
```

3. Set of flights with minimum combined cost that will still make it possible to send packages from a city to any other city.



```
 3. find the set of flights with minimum
    combined cost that will still make it
    possible to send packages from a city
    to any other city

          From   with a Distance of
0  San Antonio                  809
1      Houston                  306
2     Amarillo                  716
3      EL Paso                  809
4  San Antonio                  306
5  San Antonio                  716
```

Conclusion

By doing this lab, I learned how graphs can be implemented in a real-life problem. I also learned how to read a excel file, how to create a adjacency list of the content, how the Dijkstra algorithm can be implemented to find the shortest path between a city and other city, and also how the Kruskal algorithm can be implemented to find the shortest path between a city and other city. I had some problems in this lab but overcome them. One of the problems was how to create the adjacency list without the name of the cities. After many intents I found a way to do it. Another problem was how to use the name of every city once the shortest path was found. To do that, I used the name of every city with their numerical equivalency that I assigned them. Then, print the shortest path in order. After overcoming that problem, I had the problem to print the last step with the Kruskal algorithm because of the time to finish it. I really liked this lab because it helped me to understand better how graphs can be used and how are represented.

Appendix

```
"""
CS 2302 - Data Structures
Instructor: Dr. Olac Fuentes
TA: Ismael Villanueva-Miranda
Laboratory 5 - Search Algorithms
Author: Dilan Ramirez
Description:
    A company wants to establish a drone service to deliver packages from city
to city. Your task is to implement
    algorithms to help them plan the set of flights that will be needed to
make sure packages can be delivered from
    every city to every other city they serve while minimizing flight
distances. The file cities.xlsx contains the names
    of the cities to be served and the flight distance from each pair of
cities from which it is feasible to fly a drone; if
    flying a drone between two cities is not feasible, the distance appears as
-1. For example, the ”drone distance”
    from El Paso to Austin is 851, from Dallas to Las Vegas it’s 1700, and it
is not feasible to fly a drone from
    Laredo to San Antonio
"""

import pandas as pd
import numpy as np
import math
import CityNames as CN
import Kruskal_MST as MST
import matplotlib.pyplot as plt


'''readFile fuction is used to read a excel file.
   It takes a file as parameter, uses panda library to convert the file in a
frame.
   Finally, the data is converted to a list.
   Fuction returns the list created from the excel file'''
def readFile(file):
    cities = pd.ExcelFile(file) # cities stores the excel file
    sheetName = cities.parse('Sheet1')  # sheetName chooses the excel sheet
will be used
    df = pd.DataFrame(sheetName.values)  # df will store all the information
that the sheet1 has
    adjacencyList = df.values.tolist() # the function "tolist()" will create a
list of the  df's values
    return adjacencyList


'''BuildAdjacencyList function is used to convert the list before created into
a adjacency list.
   This function returns the adjacency list'''
def BuildAdjacencyList(adjacencyList):
    print(" ------------------------------------------------")
    print("")
    print("  1. Read the cities.xlsx file")
    print("     and build an adjacency list")
    print("     representation of the distance matrix")
    AL = [[] for i in range(len(adjacencyList))] # the adjacency List is 
created
    for i in range(len(adjacencyList)):
        AL[i] = adjacencyList[i] # it will store all the information that 
adjacencyList has
    for i in range(len(AL)):
        del AL[i][0] # delete city's name
        for j in range(len(AL[i])):
            if AL[i][j] != -1:
                AL[i][j] = [j, AL[i][j]] # It creates the new adjacencyList 
with all the new weights and destinations
    for weight in range(len(AL)):
        while -1 in AL[weight]: # Searchs all the -1 in the list and removes 
them.
            AL[weight].remove(-1)
    return AL


'''The findShortestPath fuction is used to print the result of the Dijkstra 
algorithm.
   It takes as parameters the cities and the distance between them.
   It uses dataframe to print the resulting list in table format.
   It returns the table of cities and distance'''
def findShortestPath(cities, dist):
    print(" ------------------------------------------------")
    print("")
    chart = pd.DataFrame(list(zip(cities, dist)), columns=['From', 
'Distance']) # This function native of pandas gives format to  the cities and 
their distances
    print(chart)
    return chart



'''This fuction is used to apply the Dijkstra algorithm.
   I will used it to find the shortest paths between nodes in a graph.
   In this case, to find the shortest path between the different cities.
   It returns the city path and distance.'''
def DijkstraAlgorithm(G, startV): #this Version of Dijkstra modifies the path 
to set the names of the every state according to its equivalent
    cities = []
    unvisited = [i for i in range(len(G))] # a list with all the vertices is 
created
    dist = [math.inf for i in G] # a list with the length of G is created 
which will content the dist
    path = [-1 for v in range(len(G))] # this is the path that will follow
    dist[startV] = 0
    while len(unvisited) > 0:
        currentV = unvisited[0] # it starts with the first vertex of the graph
        for adjV in G[currentV]: # it visiits its neighbors
```

```
            neighbor, edgeW = adjV[0], adjV[1] # the weight and the neigbor
are stored
            altPath = dist[currentV] + edgeW # the path incluiding the weight
is created
            if altPath < dist[neighbor]: # if the weight is is less than
infinite or the previous weight
                dist[neighbor] = altPath # it is changed by the new smaller
one
                path[neighbor] = neighbor  # the reference of this weight that
is the neighbor is added to the path
        unvisited.pop(0)
    cities = CN.CityNames(path)#the numbers of each city is changed with its
equivalent name
    findShortestPath(cities, dist) # this funciton is called to print in order
all the cities with their distance
    return path, dist



'''The executeDijkstraAlgorithm fuction is used to run the Dijkstra algorithm.
   It takes the adjacency list which is the graph to find the shortest
paths.'''
def executeDijkstraAlgorithm(AL):
    print("")
    print("  2. Shortest Path from Every City to Every Other City")
    print("")
    print("-------------City names-----------")
    printCities()
    for i in range(len(AL)):
        DijkstraAlgorithm(AL,i) # The Dijkstra function is called to to create
the shortest path all of the cities
    print("")
    return


'''This function takes a list containing all the cities.
   Its function is to print all cities for the sake of understanding.
   It returns the cities list.'''
def printCities():
   index = 0
   cities = ['El Paso','San Antonio','Houston','Amarillo']
   for city in cities:
       index += 1
       print(index, "-", city)
   return cities


'''The executeKruskalAlgorithm function applies the Kruskal algorithm to find
the shortest
   path between all the cities.
   It takes as parameter the adjacency list as parameter.
   It returns the minimum-spanning-tree (distance between cities)'''
def executeKruskalAlgorithm(AL):
    print(" 3. find the set of flights with minimum")
    print("    combined cost that will still make it")
    print("    possible to send packages from a city")
    print("    to any other city")
    print("")
    cityOrder, cities, distance = [], [], []
    mst = MST.kruskal(AL) # It stores the minimun spanning Tree of the
adjacency List
    for i in range(len(mst)):
        for j in range(len(mst[i])):
            cityOrder.append(mst[i][j][0]) # A new list cointaining the cities
that are part of the minimun spaning tree
            distance.append(mst[i][j][1]) # A new list cointaining the
distances that are part of the minimun spaning tree
    cities = CN.CityNames(cityOrder)  # The name of the cities are chosen
according to its numerical equivalent
    chart = pd.DataFrame(list(zip(cities, distance)), columns=['From', ' with
a Distance of']) # This function native of pandas gives format to  the cities
and their distances
    print(chart)
    print("")
    return mst

'''The main function runs all the functions.'''
if __name__ == "__main__":
    plt.close("all")
    file = readFile('example.xlsx') #Reads the excel file
    AL = BuildAdjacencyList(file) #Builts the adjancency list
    print(AL)

    Dijkstra_Path = executeDijkstraAlgorithm(AL) #Run Dijkstra Algorithm
    Kruskal_Path = executeKruskalAlgorithm(AL) #Run Kruskal Algorithm
```