import numpy as np

import matplotlib.mlab as mlab

import glob

import os

import csv

from os.path import join

import math

def ReadFile(FileName):

r= mlab.csv2rec(FileName)

return r

def FloatFile(FileName):

return [[float(y) for y in x] for x in ReadFile(FileName)]

def IntegerFile(FileName):

return [[int(y) for y in x] for x in ReadFile(FileName)]

# trip production and attraction

def ProductionList(SocialEcoFactorsList,SocialEcoCoeffList):

ProdList=[]

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

Production=0

for j in range (1,len(SocialEcoCoeffList[0])):

Production=Production+SocialEcoCoeffList[0][j]\*SocialEcoFactorsList[i][j]

ProdList.append(Production)

return ProdList

def AttractionList(SocialEcoFactorsList,SocialEcoCoeffList):

AttractList=[]

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

Attraction=0

for j in range (1,len(SocialEcoCoeffList[0])):

Attraction=Attraction+SocialEcoCoeffList[1][j]\*SocialEcoFactorsList[i][j]

AttractList.append(Attraction)

return AttractList

def CalibrationCoefficient(SocialEcoFactorsList,SocialEcoCoeffList):

SumProduction=sum(t for t in ProductionList(SocialEcoFactorsList,SocialEcoCoeffList))

SumAttraction=sum(t for t in AttractionList(SocialEcoFactorsList,SocialEcoCoeffList))

CaliCoeff=float(SumProduction)/(SumAttraction)

return CaliCoeff

def TargetAttractionList(SocialEcoFactorsList,SocialEcoCoeffList):

TargetAttraction=[]

CaliCoeff=CalibrationCoefficient(SocialEcoFactorsList,SocialEcoCoeffList)

AttractList=AttractionList(SocialEcoFactorsList,SocialEcoCoeffList)

TargetAttraction=[i\*CaliCoeff for i in AttractList]

return TargetAttraction

def TargetProductionList(SocialEcoFactorsList,SocialEcoCoeffList):

TargetProduction=ProductionList(SocialEcoFactorsList,SocialEcoCoeffList)

return TargetProduction

## impedance calculation

def RowIndexZone (ZoneCode):

i,j=divmod(ZoneCode,GridSize)

if j==0:

return i

else:

return (i+1)

def ColumnIndexZone (ZoneCode):

i,j=divmod(ZoneCode,GridSize)

if j!=0:

return j

else:

return GridSize

def Zone(StationCode,MetroStationLocation):

return int(MetroStationLocation[StationCode-1][1])

def Station(ZoneCode,MetroStationLocation):

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

if MetroStationLocation[i][1]==ZoneCode:

StationCode=MetroStationLocation[i][0]

return StationCode

def distance(StartZone,EndZone,MetroStationLocation):

a=(RowIndexZone(StartZone)-RowIndexZone(EndZone))\*\*2+(ColumnIndexZone(StartZone)-ColumnIndexZone(EndZone))\*\*2

return math.sqrt(a)

def MaxStationCode(MetroStationLocation):

return int(MetroStationLocation[-1][0])

def AdjacentStation(StationCode,MetroLine):

AdjacentStationList=[]

for i in range(0,len(MetroLine)):#len()

for j in range(1,len(MetroLine[0])):#len([])

if MetroLine[i][j]==StationCode:

if j-1>=1: #print "j-1 is",j-1

AdjacentStationList.append(MetroLine[i][j-1])

if j+1<=len(MetroLine[0])-1 and MetroLine[i][j+1]!=-1: #print "j+1 is",j+1

AdjacentStationList.append(MetroLine[i][j+1])

return AdjacentStationList # ouput is [2.0, 4.0, 3.0, 5.0]

def GraphEachStation(StationCode,MetroStationLocation,MetroLine):

GraphEachStation=dict() #same as ={}

AdjacentStationList=AdjacentStation(StationCode,MetroLine)

for DictElementIndex in range (0,len(AdjacentStationList)):

start=Zone(StationCode,MetroStationLocation) #print 'start zone is',start

endstation= AdjacentStationList[DictElementIndex]

end=Zone(endstation,MetroStationLocation) #print 'zone of adjacent station is',end

GraphEachStation[Zone(endstation,MetroStationLocation)]=distance(start,end,MetroStationLocation)

return GraphEachStation #print "GraphEachStation",GraphEachStation #output: graph6={2:,4:,3:,5:}

def MetroGraph(MetroLine,MetroStationLocation):

graph=dict()

for StationCode in range (1,MaxStationCode(MetroStationLocation)+1):

graph[Zone(StationCode,MetroStationLocation)]=GraphEachStation(StationCode,MetroStationLocation,MetroLine)

return graph #output: graph={6:{2:,4:,3:,5:}}

import sys

def shortestpath(graph,start,end,visited=[],distances={},predecessors={}):#Dijkstra algorithm

"""Find the shortest path between start and end nodes in a graph"""

if start==end:

path=[]

while end != None:

path.append(end)

end=predecessors.get(end,None)

return path[::-1] # detect if it's the first time through, set current distance to zero

if not visited:

distances[start]=0 # process neighbors as per algorithm, keep track of predecessors

for neighbor in graph[start]:

if neighbor not in visited:

neighbordist = distances.get(neighbor,sys.maxint)

tentativedist = distances[start] + graph[start][neighbor]

if tentativedist < neighbordist:

distances[neighbor] = tentativedist

predecessors[neighbor]=start # neighbors processed, now mark the current node as visited

visited.append(start) # finds the closest unvisited node to the start

unvisiteds = dict((k, distances.get(k,sys.maxint)) for k in graph if k not in visited)

closestnode = min(unvisiteds, key=unvisiteds.get) # now we can take the closest node and recurse, making it current

return shortestpath(graph,closestnode,end,visited,distances,predecessors)

############definition of cost/impedance

def CheckZoneIsOnStation(ZoneCode,MetroStationLocation):

check=False

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

if ZoneCode==MetroStationLocation[i][1]:

check=True

return check

def MetroPath(path,MetroStationLocation): #in order to see which metro stations a path passes.It can happen that a path only passes one station. then it means this path does not use metro line

MetroPath=[]

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

if CheckZoneIsOnStation(path[i],MetroStationLocation)==True:

MetroPath.append(path[i])

return MetroPath

###############################

def ZoneInWhichLine(ZoneCode,MetroLine,MetroStationLocation): # method two

LineCodeListWithDuplicates=[]

for RowIndex in range (0,len(MetroLine)):

for ColumnIndex in range (1,len(MetroLine[0])):

if MetroLine[RowIndex][ColumnIndex]==Station(ZoneCode,MetroStationLocation):# question: LineCodeList should not have repetitive element

LineCode=MetroLine[RowIndex][0]

LineCodeListWithDuplicates.append(LineCode)

LineCodeList=[]

for element in LineCodeListWithDuplicates:

if element not in LineCodeList:

LineCodeList.append(element)

return LineCodeList

def EdgeOnWhichLine(path,IndexOfZoneCodeInMetroPath,MetroLine,MetroStationLocation):

metropath=MetroPath(path,MetroStationLocation)

i=IndexOfZoneCodeInMetroPath

EdgeLine\_Pair=[metropath[i],metropath[i+1],0]

LineCodeList1=ZoneInWhichLine(metropath[i],MetroLine,MetroStationLocation) # MetroPath[i] is a Station Code

LineCodeList2=ZoneInWhichLine(metropath[i+1],MetroLine,MetroStationLocation) #MetroPath[i+1] is the code of next station

for LineCode in LineCodeList1:

if LineCode in LineCodeList2:

EdgeLine\_Pair[2]=LineCode

return EdgeLine\_Pair

def LineCodeOfAPath(path,MetroStationLocation,MetroLine):

list=[]

metropath=MetroPath(path,MetroStationLocation)

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

list.append(EdgeOnWhichLine(path,i,MetroLine,MetroStationLocation))

return list

def WalkingDistance(path,MetroStationLocation):

OrigWalkingDistance=0

DestWalkingDistance=0

if CheckZoneIsOnStation(path[0],MetroStationLocation)==False:

OrigWalkingDistance=distance(path[0],path[1],MetroStationLocation) #StartZone need to be defined

if CheckZoneIsOnStation(path[-1],MetroStationLocation)==False:

DestWalkingDistance=distance(path[-2],path[-1],MetroStationLocation) #EndZone need to be defined

WalkingDistance=OrigWalkingDistance+DestWalkingDistance

return WalkingDistance

def WalkingTime(path,MetroStationLocation):

WalkingSpeed=ImpedanceCalculationFactors[0][6]

WalkingTimeWeight= ImpedanceCalculationFactors[0][3]

WalkingTime=(float(WalkingDistance(path,MetroStationLocation))/WalkingSpeed)\*WalkingTimeWeight #WalkingSpeed need to be read from file

return WalkingTime

def TransferTime(path,MetroProperty,MetroStationLocation,MetroLine):

TotalTransferTime=0

LineCodeList=LineCodeOfAPath(path,MetroStationLocation,MetroLine)

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

if LineCodeList[i][2]!=LineCodeList[i+1][2]:

#print "i:",i

ToLine=LineCodeList[i+1][2]

for j in range (0,len(MetroProperty)):

if ToLine==MetroProperty[j][0]:

FrequencyOfLine=MetroProperty[j][1]

RatioOfWaitingTime\_Frequency=ImpedanceCalculationFactors[0][8]

TransferWaitingTime=RatioOfWaitingTime\_Frequency\*FrequencyOfLine

TransferWalkingTime=ImpedanceCalculationFactors[0][7]

SingleTransferTime=TransferWalkingTime+TransferWaitingTime

TotalTransferTime=TotalTransferTime+SingleTransferTime

return TotalTransferTime

def MetroTravelingDistance(path,MetroStationLocation):

metropath=MetroPath(path,MetroStationLocation)

d=0

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

d=d+distance(metropath[i],metropath[i+1],MetroStationLocation)

return d

def MetroTravelingTime(path,MetroStationLocation):

d=MetroTravelingDistance(path,MetroStationLocation)

MetroSpeed=ImpedanceCalculationFactors[0][9]

Time=float(d)/(MetroSpeed) # true, only when MetroSpeed is always unchanged for all the Metro Lines.

return Time

def WaitingTimeAtInitialStation(path,MetroProperty,MetroLine,MetroStationLocation):

FirstLine=EdgeOnWhichLine(path,0,MetroLine,MetroStationLocation)[2]

WaitingTime=0

for j in range (0,len(MetroProperty)):

if FirstLine==MetroProperty[j][0]:

FrequencyOfLine=MetroProperty[j][1]

RatioOfWaitingTime\_Frequency=ImpedanceCalculationFactors[0][8]

WaitingTime=RatioOfWaitingTime\_Frequency\*FrequencyOfLine

return WaitingTime

def TimeCost(path,MetroProperty,MetroStationLocation,MetroLine):

WaitingTimeWeight= ImpedanceCalculationFactors[0][0]

MetroTravellingTimeWeight= ImpedanceCalculationFactors[0][1]

TransferTimeWeight= ImpedanceCalculationFactors[0][2]

WalkingTimeWeight= ImpedanceCalculationFactors[0][3]

WaitTimeInitial=WaitingTimeAtInitialStation(path,MetroProperty,MetroLine,MetroStationLocation)

MetroTravelTime=MetroTravelingTime(path,MetroStationLocation)

TransTime=TransferTime(path,MetroProperty,MetroStationLocation,MetroLine)

TotalTimeCost=WaitTimeInitial\*WaitingTimeWeight+TransTime\*TransferTimeWeight+MetroTravelTime\*MetroTravellingTimeWeight+WalkingTime(path,MetroStationLocation)\*WalkingTimeWeight

return TotalTimeCost

def MoneyCost(path,MetroStationLocation):

PricePerDistance=ImpedanceCalculationFactors[0][5]

MoneyWeight=ImpedanceCalculationFactors[0][4]

TotalMoneyCost=float(MetroTravelingDistance(path,MetroStationLocation))\*PricePerDistance\*MoneyWeight

return TotalMoneyCost

def CostOfAPath(path,MetroProperty,MetroStationLocation,MetroLine):

TotalCost=TimeCost(path,MetroProperty,MetroStationLocation,MetroLine)+MoneyCost(path,MetroStationLocation)

return TotalCost

###########################################################

def Closest\_StationDistancePair(ZoneCode,MetroStationLocation):

StationDistancePair\_List=[]

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

Distance=distance(ZoneCode,MetroStationLocation[i][1],MetroStationLocation)

ZoneOfAdjacentStation=Zone(MetroStationLocation[i][0],MetroStationLocation)

StationDistancePair\_List.append([ZoneOfAdjacentStation,Distance])

DistanceList=[]

for PairListIndex in range (0,len(StationDistancePair\_List)):

DistanceList.append(StationDistancePair\_List[PairListIndex][1])

MinDistance=min(DistanceList)

Closest\_StationDistancePair\_List=[]

for PairListIndex in range (0,len(StationDistancePair\_List)):

if StationDistancePair\_List[PairListIndex][1]==MinDistance: #print StationDistancePair\_List[PairListIndex]

Closest\_StationDistancePair\_List.append(StationDistancePair\_List[PairListIndex]) #print "Closest\_StationDistancePair\_List",Closest\_StationDistancePair\_List

return Closest\_StationDistancePair\_List

def UpdateGraph(ZoneCode,graph,MetroStationLocation): #自变量是graph，return的还是graph，表示原graph被新的graph覆盖，且没有被保存。

CloSta=Closest\_StationDistancePair(ZoneCode,MetroStationLocation)

if len(CloSta)==1:

if CloSta[0][1]==0: # example [[3, 0.0]]

return graph

else:

AddKey=ZoneCode

AddDictionary=dict()

ZoneOfAdjacentStation=CloSta[0][0]

distance=CloSta[0][1]

AddDictionary[ZoneOfAdjacentStation]=distance

graph[AddKey]=AddDictionary

graph[ZoneOfAdjacentStation][ZoneCode]=distance

return graph

else:

AddKey=ZoneCode

AddDictionary=dict()

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

ZoneOfAdjacentStation=CloSta[i][0]

distance=CloSta[0][1]

AddDictionary[ZoneOfAdjacentStation]=distance

graph[AddKey]=AddDictionary

for key in AddDictionary:

graph[key][ZoneCode]=AddDictionary.get(key)

return graph

def ShortestPathMatrix(MetroLine,MetroStationLocation):

OriginalMetroGraph=MetroGraph(MetroLine,MetroStationLocation)

ShortestPathMatrix=[[-1 for OriginZoneCode in range(TotalAmountGrid)] for DestinationZoneCode in range(TotalAmountGrid)]

for OriginZoneCode in range (1,TotalAmountGrid+1):

for DestinationZoneCode in range (1,TotalAmountGrid+1):

if OriginZoneCode==DestinationZoneCode:

ShortestPathMatrix[OriginZoneCode-1][DestinationZoneCode-1]=0

else:

OriginalMetroGraph=MetroGraph(MetroLine,MetroStationLocation)

FirstUpdatedGraph=UpdateGraph(OriginZoneCode,OriginalMetroGraph,MetroStationLocation)

SecondUpdatedGraph=UpdateGraph(DestinationZoneCode,FirstUpdatedGraph,MetroStationLocation)

ShortestPathMatrix[OriginZoneCode-1][DestinationZoneCode-1]=shortestpath(SecondUpdatedGraph,OriginZoneCode,DestinationZoneCode,visited=[],distances={},predecessors={})

return ShortestPathMatrix

def ImpedanceOfAllPaths(MetroLine,MetroStationLocation,MetroProperty):

ImpedanceMatrix=[[float("inf") for OriginZoneCode in range(TotalAmountGrid)] for DestinationZoneCode in range(TotalAmountGrid)]

PathMatrix=ShortestPathMatrix(MetroLine,MetroStationLocation)

for RowIndex in range (0,TotalAmountGrid):#(0,TotalAmountGrid)

for ColumnIndex in range (0,TotalAmountGrid):#(0,TotalAmountGrid)

if RowIndex!=ColumnIndex:# if RowIndex==ColumnIndex, the impedance is infinity

path=PathMatrix[RowIndex][ColumnIndex]

metropath=MetroPath(path,MetroStationLocation) # in order to know which metro stations a path passes

if len(metropath)<=1:# the path only passes one station. this means no metro line is used

ImpedanceMatrix[RowIndex][ColumnIndex]=float("inf")

else:

ImpedanceMatrix[RowIndex][ColumnIndex]=CostOfAPath(path,MetroProperty,MetroStationLocation,MetroLine)

return ImpedanceMatrix

################################### trip distribution\_gravity model

def DeterrenceMatrix(MetroLine,MetroStationLocation,MetroProperty):

DeterMatrix=[[float("inf") for i in range(TotalAmountGrid)] for j in range(TotalAmountGrid)] # initialize the deterrence list. In F(C), F is the deterrence function

Impedance=ImpedanceOfAllPaths(MetroLine,MetroStationLocation,MetroProperty)

for i in range(0,TotalAmountGrid):

for j in range(0,TotalAmountGrid):

if Impedance[i][j]!=float("inf"):

DeterMatrix[i][j]=1000\*math.exp(-0.1\*Impedance[i][j]) #give deterrence function. question!!! exp() is so small that 0.0000003=0

return DeterMatrix

def SumProduction(T):

SumOriginList=[-1 for i in range (TotalAmountGrid)] #initialize SumOriginList.

for i in range (0,TotalAmountGrid):

SumOriginList[i]=sum(T[i])

return SumOriginList

def SumAttraction(T):

SumDestinationList=[-1 for i in range (TotalAmountGrid)] #initialize SumDestinationList.

for j in range (0,TotalAmountGrid):

column=j

SumEachColumn=sum(row[column] for row in T)

SumDestinationList[j]=SumEachColumn

return SumDestinationList

def AccuracyOrigins(MetroLine,T,SocialEcoFactorsList,SocialEcoCoeffList): #check accuracy of the origins/row

sumrow=SumProduction(T)

DifferenceList=[]

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

difference=abs(sumrow[i]-TargetProductionList(SocialEcoFactorsList,SocialEcoCoeffList)[i])

DifferenceList.append(difference)

if min(DifferenceList) > 0.01: #means sumrow is not close enough to the TargetOriginList

return True

else:

return False

def AccuracyDestinations(MetroLine,T,SocialEcoFactorsList,SocialEcoCoeffList): ##check accuracy of the destinations/column

sumcolumn=SumAttraction(T)

DifferenceList=[]

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

difference=abs(sumcolumn[i]-TargetAttractionList(SocialEcoFactorsList,SocialEcoCoeffList)[i])

DifferenceList.append(difference)

if min(DifferenceList) > 0.01: #means sumcolumn is not close enough to the TargetDestinationList

return True

else:

return False

def UpdateTripDistributionMatrix (Coefa,Coefb,MetroLine,T,DeterMatrix):

for i in range(0,TotalAmountGrid):

for j in range(0,TotalAmountGrid):

T[i][j]=Coefa[i]\*Coefb[j]\*T[i][j]

return T

def TripDistribution(TotalAmountGrid,MetroLine,SocialEcoFactorsList,SocialEcoCoeffList,MetroStationLocation,MetroProperty):

T=[[-1 for i in range(TotalAmountGrid)] for j in range(TotalAmountGrid)] #initialize T (number of trips going from origin i to destination j)

Coefa=[1 for i in range (TotalAmountGrid)] #initialize coefficient a

Coefb=[1 for j in range (TotalAmountGrid)] #initialize coefficient b

DeterMatrix=DeterrenceMatrix(MetroLine,MetroStationLocation,MetroProperty)

for i in range(0,TotalAmountGrid): #initial T

for j in range(0,TotalAmountGrid):

if DeterMatrix[i][j]!=float("inf"):# not infinity

T[i][j]=1\*1\*(DeterMatrix[i][j])

else:

T[i][j]=0

TargetAttra=TargetAttractionList(SocialEcoFactorsList,SocialEcoCoeffList)

TargetProd=TargetProductionList(SocialEcoFactorsList,SocialEcoCoeffList)

IterationTime=1

while IterationTime<=1000:#unsolved. or AccuracyDestinations(MetroLine,T,SocialEcoFactorsList,SocialEcoCoeffList)==True: #and AccuracyDestinations(MetroLine,T)==True):

SumColumn=SumAttraction(T)

for j in range (0,TotalAmountGrid): ########### first part of the loop: correct attractions

Coefb[j]=Coefb[j]\*(float(TargetAttra[j])/SumColumn[j]) # print "Coefb is", Coefb

for a in range(0,TotalAmountGrid):

for b in range(0,TotalAmountGrid):

if DeterMatrix[a][b]!=float("inf"):

T[a][b]=Coefa[a]\*Coefb[b]\*DeterMatrix[a][b]#Coefa[a]\*Coefb[b]\*T[a][b] #

SumRow=SumProduction(T)

for i in range (0,TotalAmountGrid): ########### first part of the loop: correct productions

Coefa[i]=Coefa[i]\*(float(TargetProd[i])/SumRow[i]) #Coefb[j]\*float()!!!!! print "Coefb is", Coefb

for p in range(0,TotalAmountGrid):

for q in range(0,TotalAmountGrid):

if DeterMatrix[a][b]!=float("inf"):

T[p][q]=Coefa[p]\*Coefb[q]\*(DeterMatrix[p][q])#Coefa[p]\*Coefb[q]\*T[p][q]#Coefa[p]\*Coefb[q]\*(DeterMatrix[p][q])

IterationTime=IterationTime+1

return T

def InitialTrafficVolumeOfSingleLine(LineCode,MetroLine,MetroStationLocation):

list=[]

StationList=MetroLine[LineCode-1]

for j in range (1,len(StationList)-1):

if StationList[j+1]!=-1:

forward=[Zone(StationList[j],MetroStationLocation),Zone(StationList[j+1],MetroStationLocation),-1]

list.append(forward)

backward=[Zone(StationList[j+1],MetroStationLocation),Zone(StationList[j],MetroStationLocation),-1]

list.append(backward)

return list

def InitialTrafficVolumeOfAllLines(MetroLine,MetroStationLocation):

List=[]

for LineCode in range (1,len(MetroLine)+1):

SingleLine=InitialTrafficVolumeOfSingleLine(LineCode,MetroLine,MetroStationLocation)

List.append(SingleLine)

return List

def FillInTrafficVolume(MetroLine,MetroStationLocation,MetroProperty):

TrafficVolume=InitialTrafficVolumeOfAllLines(MetroLine,MetroStationLocation)

PathMatrix=ShortestPathMatrix(MetroLine,MetroStationLocation)

TripEndMatrix=TripDistribution(TotalAmountGrid,MetroLine,SocialEcoFactorsList,SocialEcoCoeffList,MetroStationLocation,MetroProperty)

for m in range (0,TotalAmountGrid):

for n in range (0,TotalAmountGrid):

path=PathMatrix[m][n]

if path!=0:

EdgeLineCodePair=LineCodeOfAPath(path,MetroStationLocation,MetroLine)

if EdgeLineCodePair!=[]:

for x in range (0,len(EdgeLineCodePair)):

LineCode=EdgeLineCodePair[x][2]

for y in range (0,len(TrafficVolume[LineCode-1])):

if TrafficVolume[LineCode-1][y][0]==EdgeLineCodePair[x][0]:# and TrafficVolume[i][1]==EdgeLineCodePair[i][1]:

TrafficVolume[LineCode-1][y][2]=TrafficVolume[LineCode-1][y][2]+TripEndMatrix[m][n]

return TrafficVolume

def TransportCapacity(LineCode,MetroProperty):

Frequency=MetroProperty[LineCode-1][1]

Capacity=MetroProperty[LineCode-1][2]

TransportCapacity=Frequency\*Capacity\*24 #24 hours per day

return TransportCapacity

def CapacityVolumeComparison(MetroLine,MetroProperty,MetroStationLocation):

List=FillInTrafficVolume(MetroLine,MetroStationLocation,MetroProperty)

for FirstIndex in range (0,len(List)):

for SecondIndex in range (0,len(List[FirstIndex])):

TraffVolume=List[FirstIndex][SecondIndex][2]

LineCode=FirstIndex+1

TraffCapacity=TransportCapacity(LineCode,MetroProperty)

List[FirstIndex][SecondIndex].append(TraffCapacity-TraffVolume)

List[FirstIndex][SecondIndex].append('Succeed') #add a cell to give output, if the capacity is not large enough

if List[FirstIndex][SecondIndex][3]<0:

List[FirstIndex][SecondIndex][4]='Fail.Traffic Capacity of Line%01d is too small to operate the traffic volume of this edge'%(FirstIndex+1)

FromZone=List[FirstIndex][SecondIndex][0]#in order to get which station the Zone belongs to

List[FirstIndex][SecondIndex][0]=Station(FromZone,MetroStationLocation)#Convert ZoneCode to Station Code

ToZone=List[FirstIndex][SecondIndex][1]

List[FirstIndex][SecondIndex][1]=Station(ToZone,MetroStationLocation)

return List

###################################3

def RevenueOfSingleLine(LineCode,MetroLine,MetroProperty,MetroStationLocation):

TrafficVolume=FillInTrafficVolume(MetroLine,MetroStationLocation,MetroProperty)

TrafficVolumeOfALine=TrafficVolume[LineCode-1]

LineRevenue=0

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

dist=distance(TrafficVolumeOfALine[i][0],TrafficVolumeOfALine[i][1],MetroStationLocation)

PricePerDistance=ImpedanceCalculationFactors[0][5]

RevenueOfanEdge=PricePerDistance\*dist\*TrafficVolumeOfALine[i][2]

LineRevenue=LineRevenue+RevenueOfanEdge

return LineRevenue

#print RevenueOfALine(2)

def RevenueOfAllLines(MetroLine,MetroProperty,MetroStationLocation):

revenue=0

for LineCode in range (1,len(MetroLine)+1):

revenue=revenue+RevenueOfSingleLine(LineCode,MetroLine,MetroProperty,MetroStationLocation)

return revenue

def CheckCircleLine(LineCode,MetroLine):

if MetroLine[LineCode-1][1]==MetroLine[LineCode-1][-1]:

return True

else:

return False

def SizeOfStations(MetroStationLocation,MetroLine):

TotalAmountOfStations=len(MetroStationLocation)

List=[]

for StationCode in range (1,TotalAmountOfStations+1):

List.append([StationCode,0])

for StationCode in range (1,TotalAmountOfStations+1):

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

for j in range (1,len(MetroLine[0])):

if MetroLine[i][j]!=-1:

if MetroLine[i][j]==StationCode:

List[StationCode-1][1]=List[StationCode-1][1]+1

for LineCode in range (1,len(MetroLine)+1):

if CheckCircleLine(LineCode,MetroLine)==True:

CorrectStationSize=MetroLine[LineCode-1][1]

List[CorrectStationSize-1][1]=List[CorrectStationSize-1][1]-1

#print "SizeOfStations",List

return List

def StationCost\_Establishment(MetroStationLocation,MetroLine,BudgetData):

StationSize=SizeOfStations(MetroStationLocation,MetroLine)

TotalAmountOfStations=len(MetroStationLocation)

EstablishmentCost\_Basic=TotalAmountOfStations\*BudgetData[0][0]

EstablishmentCost\_Interchange=0

InterchangeTimes=0

for StationCode in range (1,len(StationSize)+1):

if StationSize[StationCode-1][1]>1:

InterchangeTimes=InterchangeTimes+StationSize[StationCode-1][1]-1

EstablishmentCost\_Interchange=InterchangeTimes\*BudgetData[0][1]

TotalCost=EstablishmentCost\_Basic+EstablishmentCost\_Interchange

return TotalCost

#print StationCost(MetroStationLocation,MetroLine)

def StationCost\_Operation(MetroStationLocation,BudgetData): #in per day

TotalAmountOfStations=len(MetroStationLocation)

MaintanceCost=TotalAmountOfStations\*(float(BudgetData[0][2])/365) #convert year cost to day cost

return MaintanceCost

###################################

def DistanceOfALine(LineCode,MetroLine,MetroStationLocation):

StationList=MetroLine[LineCode-1]

d=0

for i in range (1,len(StationList)-1):

if StationList[i+1]!=-1:

FromStation=StationList[i]

ToStation=StationList[i+1]

d=d+distance(Zone(FromStation,MetroStationLocation),Zone(ToStation,MetroStationLocation),MetroStationLocation)

return d

def RodeCostOfALine\_Establishment(LineCode,MetroLine,BudgetData,MetroStationLocation):

D=DistanceOfALine(LineCode,MetroLine,MetroStationLocation)

EstablishmentCost=D\*BudgetData[0][3]

return EstablishmentCost

def RodeCostOfALine\_Operation(LineCode,MetroLine,BudgetData,MetroStationLocation): #in per day

D=DistanceOfALine(LineCode,MetroLine,MetroStationLocation)

MaintanceCost=D\*(float(BudgetData[0][4])/365) #convert year cost to day cost

return MaintanceCost

##########################

def AmountOfVechiclesOfALine(LineCode,MetroLine,MetroProperty,MetroStationLocation): #in a day (=in per 24 hours)

distance=DistanceOfALine(LineCode,MetroLine,MetroStationLocation)

MetroSpeed=ImpedanceCalculationFactors[0][9]

ReturnTime=float(distance)/MetroSpeed #how many hours does a vechicle needs to finish its one-time return cycle.

Frequency=MetroProperty[LineCode-1][1] # in one hour, how many vechicles will go through.

AmountOfVechiclesInOneCycle=ReturnTime\*Frequency #hour\*(number of vechicles/hour)

NumberOfCycles=float(24)/ReturnTime # one day has 24 hours.24 hours/hours for one cycle

AmountOfVechicles=AmountOfVechiclesInOneCycle\*NumberOfCycles

return AmountOfVechicles

def VechicleCostOfALine\_Establishment(LineCode,BudgetData,MetroLine,MetroProperty,MetroStationLocation):

AmountOfVechicles=AmountOfVechiclesOfALine(LineCode,MetroLine,MetroProperty,MetroStationLocation)

PurchaseVechicleCost=AmountOfVechicles\*(float(BudgetData[0][6])/365) #convert year to day

return PurchaseVechicleCost

def VechicleCostOfALine\_Operation(LineCode,MetroLine,BudgetData,MetroProperty,MetroStationLocation):#in a day (=in per 24 hours)

AmountOfVechicles=AmountOfVechiclesOfALine(LineCode,MetroLine,MetroProperty,MetroStationLocation)

Frequency=MetroProperty[LineCode-1][1]

d=Frequency\*24\*DistanceOfALine(LineCode,MetroLine,MetroStationLocation)\*2 # how many vechicles go through in 1 hour\*24 hours per day\*return kilometers of a line

FuelCost=AmountOfVechicles\*d\*BudgetData[0][5]

MaintanceCost=AmountOfVechicles\*(float(BudgetData[0][7])/365) #convert year to day

TotalCost=FuelCost+MaintanceCost

return TotalCost

#print "VechicleCostOfALine(LineCode,BudgetData):",VechicleCostOfALine(1,BudgetData)

############################

def LabourCostOfALine\_Operation(LineCode,MetroLine,BudgetData,MetroProperty,MetroStationLocation):#in a day (=in per 24 hours)

AmountOfVechicles=AmountOfVechiclesOfALine(LineCode,MetroLine,MetroProperty,MetroStationLocation)

Cost=AmountOfVechicles\*BudgetData[0][8]\*(BudgetData[0][9]/30) #\*how many labours in a vechicle\*salary (convert month salary to day salary)

return Cost

############################

def EstablishmentCostOfAllLines(MetroStationLocation,MetroLine,BudgetData,MetroProperty):

StationCost=StationCost\_Establishment(MetroStationLocation,MetroLine,BudgetData)

RoadCost=0

VechCost=0

for LineCode in range (1,len(MetroLine)+1):

RoadCost=RoadCost+RodeCostOfALine\_Establishment(LineCode,MetroLine,BudgetData,MetroStationLocation)

VechCost=VechCost+VechicleCostOfALine\_Establishment(LineCode,BudgetData,MetroLine,MetroProperty,MetroStationLocation)

TotalCost=StationCost+RoadCost+VechCost

return TotalCost

def OperationCostOfSingleLine(LineCode,MetroStationLocation,MetroLine,BudgetData,MetroProperty):#in a day (=in per 24 hours)

StationCost=StationCost\_Operation(MetroStationLocation,BudgetData)

RoadCost=RodeCostOfALine\_Operation(LineCode,MetroLine,BudgetData,MetroStationLocation)

VechCost=VechicleCostOfALine\_Operation(LineCode,MetroLine,BudgetData,MetroProperty,MetroStationLocation)

LaboCost=LabourCostOfALine\_Operation(LineCode,MetroLine,BudgetData,MetroProperty,MetroStationLocation)

TotalCost= StationCost+RoadCost+VechCost+LaboCost

#print "TotalCostOfALine(LineCode):",TotalCost

return TotalCost

def OperationCostOfAllLines(MetroStationLocation,MetroLine,BudgetData,MetroProperty):#in a day (=in per 24 hours)

TotalCost=0

for LineCode in range (1,len(MetroLine)+1):

TotalCost=TotalCost+OperationCostOfSingleLine(LineCode,MetroStationLocation,MetroLine,BudgetData,MetroProperty)

return TotalCost

def IncomeOfSingleLine(LineCode,MetroStationLocation,MetroLine,BudgetData,MetroProperty):

OperationCost=OperationCostOfSingleLine(LineCode,MetroStationLocation,MetroLine,BudgetData,MetroProperty)

Revenue=RevenueOfSingleLine(LineCode,MetroLine,MetroProperty,MetroStationLocation)

Income=Revenue-OperationCost

return Income

def IncomeOfAllLines(MetroStationLocation,MetroLine,BudgetData,MetroProperty):

TotalIncome=0

for LineCode in range (1,len(MetroLine)+1):

TotalIncome=TotalIncome+IncomeOfSingleLine(LineCode,MetroStationLocation,MetroLine,BudgetData,MetroProperty)

return TotalIncome

def RunScenario(ScenarioCode):

FileMetroStation=r"D:\2015 Qiao\UT ATLAS study\semester 4\PP\MyPthon\MetroData\Scenario%02d\MetroStation%02d.csv"%(ScenarioCode,ScenarioCode)

MetroStationLocation=IntegerFile(FileMetroStation)

#print MetroStationLocation

FileMetroLine=FileMetroStation=r"D:\2015 Qiao\UT ATLAS study\semester 4\PP\MyPthon\MetroData\Scenario%02d\MetroLine%02d.csv"%(ScenarioCode,ScenarioCode)

MetroLine=IntegerFile(FileMetroLine)# number of "station" text should be no more than the max amount of stations in a line

#print MetroLine

FileMetroProperty=r"D:\2015 Qiao\UT ATLAS study\semester 4\PP\MyPthon\MetroData\Scenario%02d\MetroProperty%02d.csv"%(ScenarioCode,ScenarioCode)

MetroProperty=FloatFile(FileMetroProperty)

outcsvfile=r"D:\2015 Qiao\UT ATLAS study\semester 4\PP\MyPthon\MetroData\Output%02d\output%02d.csv"%(ScenarioCode,ScenarioCode)

out\_file=open(outcsvfile,'w')

lineout='Output Of Scenarial %02d\n'%(ScenarioCode) #title of the output file

out\_file.write (lineout)

################# Trip Distribution O-D Matrix

lineout='Part One: Trip Distribution Origin-Destination Matrix\n' #the 1st line is the title of this matrix

out\_file.write (lineout)

lineout=',' #in the 2nd line, the first cell should be empty

out\_file.write (lineout)

TripEnd=TripDistribution(TotalAmountGrid,MetroLine,SocialEcoFactorsList,SocialEcoCoeffList,MetroStationLocation,MetroProperty)

for a in range (0,len(TripEnd)): # in the 2nd line, write the name of all ZoneCodes

lineout='Zone%d,'%(a+1)

out\_file.write (lineout)

lineout='Trip Production Of Each Zone\n' #go to next row

out\_file.write (lineout)

SumProd=SumProduction(TripEnd)

for i in range (0,len(TripEnd)): #in the 3rd line, a Zonecode+a row of impedance value

lineout='Zone%d,'%(i+1)

out\_file.write (lineout)

for j in range (0,len(TripEnd)):

lineout='%10.5f,'%(TripEnd[i][j])

out\_file.write (lineout)

lineout='%10.5f\n'%(SumProd[i])

out\_file.write (lineout)

lineout='Trip Attraction Of Each Zone,'# get sum of each column

out\_file.write (lineout)

SumAttra=SumAttraction(TripEnd)

for m in range (0,len(SumAttra)):

lineout='%10.5f,'%(SumAttra[m])

out\_file.write (lineout)

TotalTrips=sum(SumAttra)

lineout='%10.5f\n\n'%(TotalTrips)

out\_file.write (lineout)

################# CapacityVolumeComparison

lineout='Part Two: Traffic Capacity and Traffic Volume Comparison\n' #the 1st line is the title of this matrix

out\_file.write (lineout)

lineout='LineCode,Transport Capacity,From Metro Station,To Metro Station,Traffic Volume,TransportCapacity-TrafficVolume, Result\n' #in the 2nd line, the first cell should be empty

out\_file.write (lineout)

CV=CapacityVolumeComparison(MetroLine,MetroProperty,MetroStationLocation)

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

Cap=TransportCapacity(i+1,MetroProperty)

for j in range (0,len(CV[i])):

lineout="Line%d,%f,%d,%d,%10.5f,%10.5f,%s\n"%(i+1,Cap,CV[i][j][0],CV[i][j][1],CV[i][j][2],CV[i][j][3],CV[i][j][4])

out\_file.write (lineout)

lineout='\n'

out\_file.write (lineout)

lineout='\n'

out\_file.write (lineout)

############## Financial Cost and Revenue

lineout='Part Three: Financial Cost and Revenue\n' #the 1st line is the title of this matrix

out\_file.write (lineout)

EC=EstablishmentCostOfAllLines(MetroStationLocation,MetroLine,BudgetData,MetroProperty)

lineout= 'Establishment Cost Of the Whole Metro System (Roads+Stations+Vechicles),%10.5f\n'%(EC)

out\_file.write (lineout)

lineout= 'Operation Cost and Revenue of Metro Lines\n'

out\_file.write (lineout)

lineout= 'LineCode,Operation Cost Per Day,Revenue Per Day,Income Per Day\n'

out\_file.write (lineout)

for LineCode in range (1,len(MetroLine)+1):

OperCostSingle=OperationCostOfSingleLine(LineCode,MetroStationLocation,MetroLine,BudgetData,MetroProperty)

ReveneSingle=RevenueOfSingleLine(LineCode,MetroLine,MetroProperty,MetroStationLocation)

IncomeSingle=IncomeOfSingleLine(LineCode,MetroStationLocation,MetroLine,BudgetData,MetroProperty)

lineout= '%d,%10.5f,%10.5f,%10.5f\n'%(LineCode,OperCostSingle,ReveneSingle,IncomeSingle)

out\_file.write (lineout)

RAll=RevenueOfAllLines(MetroLine,MetroProperty,MetroStationLocation)

OCAll=OperationCostOfAllLines(MetroStationLocation,MetroLine,BudgetData,MetroProperty)

IAll=IncomeOfAllLines (MetroStationLocation,MetroLine,BudgetData,MetroProperty)

lineout= 'Sum,%10.5f,%10.5f,%10.5f\n'%(OCAll,RAll,IAll)

out\_file.write (lineout)

out\_file.flush()

out\_file.close

# ############# Main Function ######################################################## Main Function:

GridSize=5 #ZoneCodeList=[[-1 for i in range(GridSize)] for j in range(GridSize)]

TotalAmountGrid=GridSize\*\*2

FileSocialEcoFactors=r"D:\2015 Qiao\UT ATLAS study\semester 4\PP\MyPthon\MetroData\CaseData\SocialEcoFactors.csv" # input Factor File

SocialEcoFactorsList=FloatFile(FileSocialEcoFactors) #read the factor file, convert all strings to float, build up a list

FileSocialEcoCoeff=r"D:\2015 Qiao\UT ATLAS study\semester 4\PP\MyPthon\MetroData\CaseData\CoeffSocialEcoFactors.csv" #input CoeffFile

SocialEcoCoeffList=ReadFile(FileSocialEcoCoeff) #build up a Coeff List. the 0th column can not be converted to float, because they are texts. so i use ReadFile, instead of FloatFile

FileImpedanceCalculationFactors=r"D:\2015 Qiao\UT ATLAS study\semester 4\PP\MyPthon\MetroData\CaseData\ImpedanceCalculationFactors.csv"

ImpedanceCalculationFactors=FloatFile(FileImpedanceCalculationFactors)

FileBudgetData=r"D:\2015 Qiao\UT ATLAS study\semester 4\PP\MyPthon\MetroData\CaseData\BudgetData.csv"

BudgetData=FloatFile(FileBudgetData)

RunScenario(1)

RunScenario(2)

RunScenario(3)

RunScenario(4)