```
#Project: Project 1 AI
#Creator: Eric Dockery
#Assignment Details: Brute Force Permutation finding minimum cost path
#Format of data:
   #NAME: concorde4
   #TYPE: TSP
   #COMMENT: Generated by CCutil_writetsplib
   #COMMENT: Write called for by Concorde GUI
   #DIMENSION: 4
   #EDGE_WEIGHT_TYPE: EUC_2D
   #NODE_COORD_SECTION
   #1 87.951292 2.658162
   #2 33.466597 66.682943
   #3 91.778314 53.807184
   #4 20.526749 47.633290
   #ect...



#import math - sqrt and operator - location of min path
import math
import operator
#imageing imports
from tkinter import *
#from PIL import Image, ImageTk

#could draw in graphics?
def plotPoints(Array, distance):
   #Make a GUI
   Map = Tk()
```

```
    w = Canvas(Map, width=300, height=300)

    #image = Image.open("Map_Westeros_political.gif")
    #photo = ImageTk.PhotoImage(image)
    Map.title("The Travel Map")
    Map.geometry("300x280+300+300")
    #Map.image=photo

    #for loop for oval
    for i in range(len(Array)):
        #plot points
        w.create_oval(float(Array[i][1]), float(Array[i][2]),
                (float(Array[i][1])+3),(float(Array[i][2])+3),
                fill='green', width=4)
    for i in range(len(Array)-1):
        #drawLines
        w.create_line(float(Array[i][1]), float(Array[i][2]),
                float(Array[i+1][1]),float(Array[i+1][2]))

    w.create_rectangle(150,150,300,250, outline="red")
    w.create_text(150,200, anchor=W, font="Purisa", text= distance)
    #finish image
    w.pack()



#distance formula
def distanceFormula(Array):
    # d = sqrt( (x2-x1)^2 +(y2-y1)^2)
```

```
    #this will be the 2ed deminsional values that are used to calculate the distance
    #Array format [[Num, X, Y], [Num ,X,Y] ...
    #i = [Num, X, Y]
    distance = 0
    #display each permutation
   # print("This Permutation Set: ")
   # print(Array)
    if (len(Array) <=1):
        distance = 0
    for i in range(0,(len(Array)-1)):
        #Array[i][1] = X ; Array[i][2] = Y
        distance += math.sqrt((float(Array[i+1][1])-float(Array[i][1]))**2 + (float(Array[i+1][2])-
float(Array[i][2]))**2)
    #Display the distance for that permutation
   # print("Equals this distance: ")
   # print(distance)
   #add the returning distance for original spot
    distance += math.sqrt((float(Array[0][1])-float(Array[len(Array)-1][1]))**2 + (float(Array[0][2])-
float(Array[len(Array)-1][2]))**2)
    return distance


#permutation program using child processes:
def permutation(ParsedArray):


    #if the conent is 1 element or less return the
    #element because it is maximum permutation
    #this will be important on the recurision calls
    if len(ParsedArray) <=1:
        #yield allows the process to return after this line
```

```
        #this reads a value once, generates children but doesn't fill memory
        #return the ParsedArray
        yield ParsedArray
        #clean memory
        del ParsedArray
        #clean the ParsedArray
        ParsedArray=[]
    #if the ParsedArray is greater than 1
    #for each item in the Content
    for i in ParsedArray:
        #list the ParsedArray
        rest= list(ParsedArray[:])
        #remove one element from the list
        rest.remove(i)
        #for every element left in rest
        #preform permutation on the list
        for j in permutation(rest):
            #add the removed elements to the permutated content
            yield [i]+j
    #cleaning out memory
    del ParsedArray


#Prep the recursion
def permutationPrep(Content):
    mincost = 11111111111111111111111111
    testcost =0
  #  thePermutation=[0] *(math.factorial(len(Content)))
    placeholder = 0
    TheFinishedArray= [0]*(len(Content))
```

```
  #for each element in the Content
  #preform the permutation of the content
  for p in permutation(Content):
     testcost=distanceFormula(p)
     if (testcost < mincost):
        mincost = testcost
        TheFinishedArray = p
     del p


#     thePermutation[placeholder] = p
     placeholder+=1
#  costTotals = [0] *(len(thePermutation))
#   for i in range(0, len(thePermutation)):
      #sending each element in the array 1 at a time to
      #get the total distance
     # print(thePermutation[i])
#     costTotals[i] =distanceFormula(thePermutation[i])
#   locationMincost =min(enumerate(costTotals),key=operator.itemgetter(1))[0]
  #display the Shortest Path
  print("The shortest Path is this permutation: ")
  TheFinishedArray.append( TheFinishedArray[0])
  print(TheFinishedArray)
#   print(thePermutation[locationMincost])
#   mincost= min(costTotals)
  plotPoints(TheFinishedArray, mincost)
  return mincost
```

```
#program Main:
def main():
    #prompt the user for the file
    filename = input("What is the file name? ")
    #Read lines 0-6 with no cord data
    with open(filename) as f:
        #read and strip the '\n'
        content = [line.rstrip('\n') for line in open(filename)]

    #line 7 starts the important information
    startCords = 0
    #seperate the values into a dictionary with the key being the first
    #value of the strings starting on line 7
    parsedContent = [0] * (len(content)-7)

    counter = 0
    #parsedContent ['object', 'X', 'Y']
    for i in range(7,len(content)):
        parsedContent[counter] = content[i].split()
        counter+=1



    #call permutation program
    minCost = permutationPrep(parsedContent)
    #This will return the finished permutation array and the length

    #print the integer values as well as return the list of costs
    print("The minimum cost is: ")
    print(minCost)
```

main()