Kenneth Maguire

12/11/18

Star Search

Introduction

I developed a program which reads in from a csv file, “hygxyz.csv”, which include the following columns: StarID, HIP, HD, HR, Gliese, BayerFlamsteed, ProperName, RA, Dec, Distance, PMRA, PMDec, RV,Mag, AbsMag, Spectrum, ColorIndex, X, Y, Z, VX, VY, VZ.

From this file I use the columns StarID, Gliese, BayerFlamsteed, ProperName, X, Y and Z. I use X,Y,Z to calculate the distance between all the stars and “Sol”, to find all stars that are within 10 parsecs of “Sol”. Then use X,Y,Z to calculate the shortest distance as I’m traversing each star, between that star and any stars that have not already been traversed. I used the numpy library to use its “sqrt” function for calculating distance and the csv library to extract data from the file. To name the stars as I’m traversing them, I use the following columns in order if the previous column is null for that star. ProperName->Gliese->BayerFlamsteed->StarID. Starting at ProperName, I checked if the value was null, and if it was, I would check the following name. If the star didn’t have another identifying name, I used StarID and referred to it as “Unnamed Star” + StarID.

Problem Description

The star traversal problem is defined as:

Input: A sequence of **n** stars <a1,a2,…an> all of which contain data (Name, (X, Y, Z))

Output: A sequence of **m** stars within 10 parsecs of the star named “Sol” <a1’,a2’,…am’> in the order they were traversed. To traverse the stars, starting at Sol, travel to the nearest star, then travel to the nearest star, then travel to the next nearest star that hasn’t been visited until all stars have been visited.

From this definition, we can derive an algorithm to find all the stars within 10 parsecs of Sol, by calculating the distance of all the stars and making a new list including only the stars that are within 10 parsecs of Sol. We’ll also need to include Sol in that list since it’s the starting point for our traversal. Then starting at Sol, we can find the nearest star by calculating all of the distances for the other stars to find the nearest one. Once we’ve found the nearest one, we can add Sol to a new list that is ordered by the order in which the stars are traversed and remove it from the one containing all stars within 10 parsecs of Sol. Now from the visited star, we’ll calculate the distances from the current star to all of the stars still in the list, and visit the next nearest. Once we visit the next nearest, we can add the previously visited star to the list of stars in the order they were visited, and remove it from the one containing all stars within 10 parsecs of Sol. We’ll repeat this process until all stars are visited and added to the list of stars in the order they were traversed.

Algorithm Design

Pseudo-code

Distance = $\sqrt{((x2-x1)^2+(y2-y1)^2+(z2-z1)^2)}$

GetDistance(x1,y1,z1,x2,y2,z2**)**
1 Distance = **sqrt**((x2 - x1)² + (y2 - y1)² + (z2 - z1)²) //get distance val
2 **return** distance

The above pseudo code gets the distance for the current star to the next star. The distance is calculated for all remaining stars in the list containing stars within 10 parsecs of Sol and to make the list of stars within 10 parsecs of Sol.

//starList is an array of tuples, which contain a string at 0 and a tuple at 1
//indexed the same as a 3D array

Traverse Stars(starList**)**
```
1 distList = [ ] //empty list
2 starsTraversalOrder = [ ] //empty list
3 i = 0
4 while starList.length() != 0
5       for j = 0 to starList.length()
6               x1 = starList[ i ][1][0]
7               y1 = starList[ i ][1][1]
8               z1 = starList[ i ][1][2]
9               if i == j
10                      pass
11              else
12                      x2 = starList[ j ][1][0]
13                      y2 = starList[ j ][1][1]
14                      z2 = starList[ j ][1][2]
15                      dist = getDistance(x1,y1,z1,x2,y2,z2)
16                      distList.append( ( j , dist ) ) //get all dist from star and append ind/dist as
                                                        tuple
17      if distList.length() != 0
18              nextStar = min(distList, distList[1]) //find min distance
19      starsTraversalOrder.append( ( starList[ i ], nextStar[1] ) ) //append last visited star and
                                                                     next dist
20      starList.remove( starList[ i ] ) //remove last visited star from original list
21      if i < nextStar[0]
22              i = (nextStar[0] - 1) //get the index of the next star to get distances for
23      else
24              i = nextStar[0]
25      distList.clear()  //clear the distance list for next round of distances found
```

This algorithm uses the starList (which contains all of the stars within 10 parsecs of Sol), starsTraversalOrder (which adds stars and the distance to the next star as they're traversed), and distList (which holds all of the distances to others stars from the currently visited star for finding the min distance) to create a list of stars in the order they were traversed. In lines 21 through 24, if the index stored in nextStar[0] is greater than the index of i, the index of i is the index of nextStar - 1, because the index stored at nextStar[0] is one greater than its actual index since the previous star was removed from starList. If the index stored at nextStar[0] is less than i, it is unaffected by the deletion since the only stars shifted left are to the right of the index at i.

Implementation

StarSearch.py and hygxyz.csv are included in the comments on canvas.

Also available @ my Github page https://github.com/KennMaguire/starSearch

StarSearch.py *(written for Python 3.7)*

The program starts by containing all of the relevant star data in a (key, value) pair with star name as the key and the X,Y,Z data is stored within a tuple as the value. Then the (key,value) pairs are added to a dictionary until the end of the file is reached. Next, the star data is used to find all of the stars within 10 parsecs of Sol by finding all of the distances of the stars from Sol, and only selecting those with values less than or equal to 10. This creates a list containing 327 stars within 10 parsecs of Sol which becomes 328 stars total after including Sol. This list is then used to traverse the stars by using the index of the most recently visited star, and finding the next nearest star that hasn't been visited yet. Once a star has been visited, it is appended to the list of stars in the order they've been traversed, and removed from the list of stars within 10 parsecs of Sol.

Once all the stars have been added to the list of stars in the order they've been traversed, they first 10 and last 10 stars are printed in the order they're traversed, the distance between each star is printed and the total distance traveled is printed.

Code in pictures below

```python
#!/Library/Frameworks/Python.framework/Versions/3.7/bin/python3
#StarSearch.py
#Written By: Kenneth Maguire
import csv
import numpy as np
import time

starDict = {}

#https://www.varsitytutors.com/hotmath/hotmath_help/topics/distance-formula-in-3d
#function for retrieving distance between two stars  https://docs.scipy.org/doc/numpy-1.15.0/reference/generated/numpy.sqrt.html
def getDist(_x1, _y1, _z1, _x2, _y2, _z2):
    dist = np.sqrt(((_x2-_x1)**2)+ ((_y2-_y1)**2) + ((_z2-_z1)**2))
    return dist

with open("hygxyz.csv", "r") as xyzfile:
    next(xyzfile)
    xyzRead = csv.reader(xyzfile, delimiter=',')
    for row in xyzRead:
        if row[6] != '':
            starDict[row[6]] = (row[17],row[18], row[19]) #properName 6, x 17, y 18, z 19
        elif row[4] != '':
            glName = "Gliese " + row[4]
            starDict[glName] = (row[17],row[18], row[19]) #Gliese 4, x 17, y 18, z 19
        elif row[5] != '':
            bfName = "BF " + row[5]
            starDict[bfName] = (row[17],row[18], row[19]) #BF 5, x 17, y 18, z 19
        else:
            idName = "Unnamed star " + row[0]
            starDict[idName] = (row[17], row[18], row[19]) #starID 0, x 17, y 18, z 19


starList = []

total = 0
#get the distances of the stars from Sol
for key,value in starDict.items():
    x1 = float(starDict["Sol"][0])
    y1 = float(starDict["Sol"][1])
    z1 = float(starDict["Sol"][2])
    x2 = float(starDict[key][0])
    y2 = float(starDict[key][1])
    z2 = float(starDict[key][2])

    dist1 = getDist(x1,y1,z1,x2,y2,z2)      #get the distance
    #print(dist1)
    if(dist1 <= 10):            #if distance is <= 10 parsecs, add to starList as tuple
        total += 1              #get the total number of stars added
        starList.append((key,value))

print("\n\n\n")
print("The total number of stars within 10 parsecs of Sol is " + str(total))
distList = []
```

```
"""
 find the shortest distance from sol first time
 change i to value with min value
 get shortest distance for next
"""
i = 0
starsTraversalOrder = []
while starList:
    for j in range(0, len(starList)):
        x1 = float(starList[i][1][0])   #i = index of of Next Star, 1 is index of tuple
        y1 = float(starList[i][1][1])   #0,1,2 are indexes of X,Y,Z
        z1 = float(starList[i][1][2])
        if j == i:
            pass
        else:
            x2 = float(starList[j][1][0])   #i = index of of Next Star, 1 is index of tuple
            y2 = float(starList[j][1][1])   #0,1,2 are indexes of X,Y,Z
            z2 = float(starList[j][1][2])
            dist2 = getDist(x1,y1,z1,x2,y2,z2)
            distList.append((j,dist2))          #get all distance from the star and append to list


    if distList:
        nextStar = min(distList, key = lambda t: t[1])      #get the minimum distance in the list by the second value in the tuple

    starsTraversalOrder.append((starList[i], nextStar[1])) #append recently visited star and the distance to the next nearest

    starList.remove(starList[i])
    if i < nextStar[0]:
        i = (nextStar[0]-1) #get the index of the next star to get distances for
    else:
        i = nextStar[0]

    distList.clear()        #clear for next traversal

totalDist = 0


#print first 10 and last 10 star in order of traversal, along with distance between each star, and total distance traveled
print("\n")
for i in range(0,10):
    totalDist += starsTraversalOrder[i][1]
    print(str(starsTraversalOrder[i][0][0]) + " -> " + str(starsTraversalOrder[i+1][0][0]) + ": Distance = "
    + str(round(starsTraversalOrder[i][1], 2)) + ", Total Distance = " + str(round(totalDist, 2)))

print("...")

for i in range(10, len(starsTraversalOrder)-11): #add to the total distance for all stars between index 10 and (size of list of Stars in Order - 10)
    totalDist += starsTraversalOrder[i][1]

for i in range(len(starsTraversalOrder)-11,len(starsTraversalOrder)-1):
    totalDist += starsTraversalOrder[i][1]
    print(str(starsTraversalOrder[i][0][0]) + " -> " + str(starsTraversalOrder[i+1][0][0]) + ": Distance = "
    + str(round(starsTraversalOrder[i][1], 2)) + ", Total Distance = " + str(round(totalDist, 2)))

print("The total distance traversed is " + str(totalDist) + " parsecs.")

"""
    print(i[0])
    print(i[1][0])
    print(i[1][1])
    print(i[1][2])
"""
```

Results

Since the data set is always the same, and the stars traversed are always in the same order, the only difference in output is how long the program takes to run. The values used to calculate total distance are floats, but the output is round to the second decimal place for distance between each star and total distance. Please note that “Gliese Gl 845” is the same star as “Eps Ind” in Doug’s output, and “Gliese Gl 827” is the same star as “Gam Pav” in Doug’s output so the star traversal is in the same exact order as the one Doug found, just using a different names in 2 instances.

#1

```
The total number of stars within 10 parsecs of Sol is 327

Sol -> Proxima Centauri: Distance = 1.29, Total Distance = 1.29
Proxima Centauri -> Rigel Kentaurus B: Distance = 0.07, Total Distance = 1.37
Rigel Kentaurus B -> Rigel Kentaurus A: Distance = 0.0, Total Distance = 1.37
Rigel Kentaurus A -> Barnard's Star: Distance = 1.98, Total Distance = 3.35
Barnard's Star -> Gliese Gl 729: Distance = 1.7, Total Distance = 5.05
Gliese Gl 729 -> Lacaille 8760: Distance = 2.26, Total Distance = 7.31
Lacaille 8760 -> Gliese Gl 832: Distance = 1.28, Total Distance = 8.59
Gliese Gl 832 -> Gliese Gl 845: Distance = 1.47, Total Distance = 10.06
Gliese Gl 845 -> Lacaille 9352: Distance = 1.44, Total Distance = 11.5
Lacaille 9352 -> Gliese Gl 866  A: Distance = 1.25, Total Distance = 12.75
...
Gliese Gl   1 -> Gliese Gl 827: Distance = 6.26, Total Distance = 532.24
Gliese Gl 827 -> Unnamed star 31220: Distance = 5.55, Total Distance = 537.78
Unnamed star 31220 -> Unnamed star 31215: Distance = 0.39, Total Distance = 538.18
Unnamed star 31215 -> Gliese GJ 1123: Distance = 1.52, Total Distance = 539.7
Gliese GJ 1123 -> Gliese Gl 367: Distance = 5.23, Total Distance = 544.93
Gliese Gl 367 -> Gliese Gl 358: Distance = 0.87, Total Distance = 545.79
Gliese Gl 358 -> Gliese Gl 318: Distance = 2.34, Total Distance = 548.13
Gliese Gl 318 -> Gliese Gl 357: Distance = 2.57, Total Distance = 550.7
Gliese Gl 357 -> Gliese Gl 283  A: Distance = 4.26, Total Distance = 554.97
Gliese Gl 283  A -> Gliese Gl 283  B: Distance = 0.0, Total Distance = 554.97
The total distance traversed is 554.9656291788556 parsecs.
[Finished in 1.534s]
```

#2

```
The total number of stars within 10 parsecs of Sol is 327

Sol -> Proxima Centauri: Distance = 1.29, Total Distance = 1.29
Proxima Centauri -> Rigel Kentaurus B: Distance = 0.07, Total Distance = 1.37
Rigel Kentaurus B -> Rigel Kentaurus A: Distance = 0.0, Total Distance = 1.37
Rigel Kentaurus A -> Barnard's Star: Distance = 1.98, Total Distance = 3.35
Barnard's Star -> Gliese Gl 729: Distance = 1.7, Total Distance = 5.05
Gliese Gl 729 -> Lacaille 8760: Distance = 2.26, Total Distance = 7.31
Lacaille 8760 -> Gliese Gl 832: Distance = 1.28, Total Distance = 8.59
Gliese Gl 832 -> Gliese Gl 845: Distance = 1.47, Total Distance = 10.06
Gliese Gl 845 -> Lacaille 9352: Distance = 1.44, Total Distance = 11.5
Lacaille 9352 -> Gliese Gl 866  A: Distance = 1.25, Total Distance = 12.75
...
Gliese Gl   1 -> Gliese Gl 827: Distance = 6.26, Total Distance = 532.24
Gliese Gl 827 -> Unnamed star 31220: Distance = 5.55, Total Distance = 537.78
Unnamed star 31220 -> Unnamed star 31215: Distance = 0.39, Total Distance = 538.18
Unnamed star 31215 -> Gliese GJ 1123: Distance = 1.52, Total Distance = 539.7
Gliese GJ 1123 -> Gliese Gl 367: Distance = 5.23, Total Distance = 544.93
Gliese Gl 367 -> Gliese Gl 358: Distance = 0.87, Total Distance = 545.79
Gliese Gl 358 -> Gliese Gl 318: Distance = 2.34, Total Distance = 548.13
Gliese Gl 318 -> Gliese Gl 357: Distance = 2.57, Total Distance = 550.7
Gliese Gl 357 -> Gliese Gl 283  A: Distance = 4.26, Total Distance = 554.97
Gliese Gl 283  A -> Gliese Gl 283  B: Distance = 0.0, Total Distance = 554.97
The total distance traversed is 554.9656291788556 parsecs.
[Finished in 1.513s]
```

Conclusions

I only ran into one issue while writing this program, which was that I had to figure out how to index the starList appropriately after a star was deleted. After thinking through which case would cause the index to be off, I realized that it was only affected when the original index was larger than the next one. At that point I found a solution and my output was correct. I was surprised at how quickly this program runs considering the number of stars in the file (n=119617) but since we're only working with 327 stars (328 if you include Sol), it make sense that traversal was fairly quick. Overall, I felt pretty good about this program since I only had to rely on my prior knowledge of python to make it work. The only thing I had to lookup was how to handle sqrt, but the numpy library made that quick and easy.

Sources

https://docs.scipy.org/doc/numpy-1.15.0/reference/generated/numpy.sqrt.html

https://www.varsitytutors.com/hotmath/hotmath_help/topics/distance-formula-in-3d