#######################<NELDER MEAD>##########

from vector\_cost import Vector

from random import random

from time import clock

from math import sin,cos,sqrt

import time

SMALLEST\_TRIANGLE\_SIZE=0.001

NUMBER\_OF\_GRID\_POINTS=100

HILL\_CLIMBING\_RADIUS =0.01

MAX\_TRIANGLE\_COUNT=50

DOMAIN\_LIMIT =10

MAX\_RUN\_TIME=.1

timee=0

#find point that results in lowest cost

def frange(start,stop, step=.01):

i=start

while i<stop:

yield i

i+=step

def runRandomSearch(): #working?

#evaluate 'a million random solutions and choose the best one

from random import random

print("2. Random Probing used", end=" ")

start= time.clock()

count=0

minV=Vector(9999999,999999)

while time.clock()-start<.1:

xmin=random()\*DOMAIN\_LIMIT

ymin=random()\*DOMAIN\_LIMIT

vector= Vector(xmin,ymin)

minV= minV.minVec(vector)

count+=1

print(count, end="")

print(" probes")

print("x=",minV[0]," y=",(minV[1]),"cost=",minV.cost())

print("Time= ",.1)

#stop after .1 sec

def runHillClimbingRandomResetSearch():

#neighbor by neighbor solution. move to the best neighbor solution. repeat

#until local min or max

#then start over and find the smallest of many local mins.

#instead of using sines and cosines, pre calculate once at begining and put in table

print("3. Hill Climbing (Random Reset) used ",end="")

start=time.clock()

minn=100000000

probes=0

poin=Vector(random()\*DOMAIN\_LIMIT,random()\*DOMAIN\_LIMIT)

bestN=findBestNeighbor(poin)

while time.clock()-start<=.1:

probes+=1

newp=Vector(random()\*DOMAIN\_LIMIT,random()\*DOMAIN\_LIMIT)

newBN=findBestNeighbor(newp)

if bestN.cost()>newBN.cost():

poin=newp

bestN=newBN

print(probes," probes")

print("X= ",bestN[0],end="")

print("Y= ",bestN[1]," Cost= ", bestN.cost())

print("Search Time= ",.1)

#so...find a random point

#then find it's best neighbor

#then... while inside a loop...find another random point and it's neighbor

#if the cost btwn 1stp and n >2ndp and n, sto=2ndp

#at teh end, return somethin

def runHillClimbingGridSearch():

print("4. Hill Climbing (Grid) used ",end="")

start=time.clock()

minn=100000000

probes=0

iterator=0

l=[]

#create a matrix of specific points

for x in range(0,11):

for y in range(0,11):

l.append(x)

l.append(y)

poin=Vector(random()\*DOMAIN\_LIMIT,random()\*DOMAIN\_LIMIT)

bestN=findBestNeighbor(poin)

while time.clock()-start<=.1 and iterator<200:

newp=Vector(l[iterator], l[iterator+1])

iterator+=2

newBN=findBestNeighbor(newp)

probes+=1

if bestN.cost()>newBN.cost():

poin=newp

bestN=newBN

print(probes," probes")

print("X= ",bestN[0],end="")

print("Y= ",bestN[1]," Cost= ", bestN.cost())

print("Search Time= ",time.clock()-start)

#exactley the same, except teh newP is from a matrix

coont=0

def findBestNeighbor(point):

from math import sin,cos,pi

radius=HILL\_CLIMBING\_RADIUS

for t in frange(0, 2\*pi, 2\*pi/16):

x=point[0]+radius\*cos(t)

y=point[1]+radius\*sin(t)

bestNeighbor=Vector(x,y)

if(bestNeighbor.cost()<point.cost()):

point.equals(bestNeighbor)

return point

#do other things :|

def runNelderMead():

start= time.clock()

print("1. Nelder-Mead used ",end="")

from random import random

triangleCount=0

a= Vector(DOMAIN\_LIMIT\*random(), DOMAIN\_LIMIT\*random()) #someow find a random point

b=Vector(DOMAIN\_LIMIT\*random(), DOMAIN\_LIMIT\*random())

c=Vector(DOMAIN\_LIMIT\*random(), DOMAIN\_LIMIT\*random())

while triangleCount<MAX\_TRIANGLE\_COUNT and time.clock()-start<.1:

if b.dist(a)<.02:#dist btwn a and b

break

if a.cost()<b.cost():

a.swap(b)

if c.cost()<b.cost():

c.swap(b)

if a.cost()<b.cost():

a.swap(b)

d=b+c-a

e=(3\*(b+c)-4\*a)/2

f=(3\*(b+c)-2\*a)/4

g=(2\*a+b+c)/4

x=f.minVec(g)

triangleCount+=1

if d.cost()<a.cost() and e.cost()<a.cost():

a.equals(e)

elif d.cost()<a.cost():

a.equals(d)

elif x.cost()<a.cost(): a.equals(x)

else:

a.equals((a+b)/2)

c.equals((b+c)/2)

#if d.cost() <A and E.cost()<A.cost, E isnow A. go to while

#elif D<A.cost() A=D. go to while loop

#elif X= vector (G or F) with min cost. if X.cost()<A.cost(), then A=X

#else: A=H, C=I. Best is still B. go to #3

print(triangleCount," random triangles.")

print("x= ",b[0],end="")

print(" y=",b[1],end="")

print(" cost=" ,b.cost())

timee=time.clock()-start

print("Search Time= ",timee)

print("\n")

def main():

print(' == FOUR SEARCHING ALGORITHMS==')

runNelderMead()

runRandomSearch()

runHillClimbingRandomResetSearch()

runHillClimbingGridSearch()

# A=Vector(3,4)

#B=Vector(random()\*DOMAIN\_LIMIT,random()\*DOMAIN\_LIMIT)

# print (A.scalars()[0])

#print(A.cost())

# print('--Search time=',round(clock()-startTime,2),'seconds')

if \_\_name\_\_=='\_\_main\_\_': startTime=clock(); main()