Felipito y Rafaelito

January 31, 2019

1 Pseudocodigos

```
Clase nodo:
    def inicializar(coord, tipo, demanda, ventana, tServ):
        self.coord = coord
        self.flagV = False
        self.tipo = tipo
        Si self.tipo == 'c':
            self.demanda = demanda
            self.ventana = ventana
            self.tServicio = tServ
        Fin Si
        Si self.tipo == 'e':
            self.demanda = 0
            self.ventana = [0, infinito]
            self.tServicio = 0
        Fin Si

    def getCoord(self):
        retornar self.coord

    def getFlag(self):
        retornar self.flagV

    def setFlag(boolVal):
        self.flagV = boolVal

    def toString(self):
        retornar [self.coord, self.flagV]
```

```
Clase mundo:
    def inicializar(nods, lfunc, cargE, tazCon, tazRec, vel, estacs, dem&&s, cosVeh, cosDis):
        self.nods = nods
        self.lfunc = lfunc
        self.estacs = estacs
        self.aristas = self.crearAristas()
        self.cargE = cargE
        self.tazRec = tazRec
        self.tazCon = tazCon
        self.vel = vel
        self.dem&&s = dem&&s
        self.cosVeh = cosVeh
        self.cosDis = cosDis

    def crearAristas(self):
        aristas = {}
        nods = self.nods + self.estacs
        Para m en nods:
            Para n en nods:
                a = m.getCoord()
                b = n.getCoord()
                Si a != b:
                    arista = arista(a, b, self.lfunc(a, b))
                    aristas[m, n] = arista
        Fin Para

        retornar aristas

    def reiniciarFeromona(nivel = 0.01):
        Para arista en self.aristas:
            arista.feromona = nivel
        Fin Para
```

```
class arista:
    def inicializar(inicio, fin, longitud, feromona):
        self.inicio = inicio
        self.fin = fin
        self.longitud = longitud
        self.feromona = feromona
```

```
class hormiga:
    def inicializar(alfa, beta, merc, energia, cosVeh, cosDis, inst):
        self.mundo = Nulo
        self.alfa = alfa
        self.beta = beta
        self.inicio = Nulo
        self.nodo2 = Nulo
        self.nodo3 = Nulo
        self.distancia = 0
        self.visitados = []
        self.noVisitados = []
        self.arcos = []
        self.merc = merc
        self.mercMax = merc
        self.energia = energia
        self.cosVeh = cosVeh
        self.cosDis = cosDis
        self.tiempo = 0
        self.cont = 0

    def inicializarInstancia(mundo, inicio):

        self.mundo = mundo
        Para i en self.mundo.nodos:
            i.flagV = False
        Fin Para
        Si inicio is Nulo:
            self.inicio = random.randRango(longitud(self.mundo.nodos))
        Caso Contrario:
            self.inicio = inicio
        Fin Si
        self.distancia = 0
        self.visitados = [self.inicio]
        self.noVisitados = [n Para n en self.mundo.nodos Si n != self.inicio]
        self.arcos = []
        retornar self

    def clonar():
        hormiga = hormiga(self.alfa, self.beta)
        hormiga.mundo = self.mundo
        hormiga.inicio = self.inicio
        hormiga.visitados = self.visitados[:]
        hormiga.noVisitados = self.noVisitados[:]
        hormiga.arcos = self.arcos[:]
        hormiga.distancia = self.distancia
        retornar hormiga
```

```
def Selec14(nodo1):
    con = 0
    vecN = []
    vecN.vaciar()
    vecC = []
    vecC.vaciar()
    Si tipo(nodo1) == lista:
        coord1 = nodo1[0].getCoord()
    Caso Contrario:
        coord1 = nodo1.getCoord()
    Fin Si
    Para i en self.mundo.nodos:
        Si i != self.nodo() && not i.flagV:
            coord2 = i.getCoord()
            euc2 = euclidean(coord1, coord2)
            dist = (self.mundo.cosVeh) + (self.mundo.cosDis * euc2)
            vecN.agregar([i, dist])
        Fin Si
    Fin Para
    long = longitud(vecN)
    Si long > 0:
        Para i en rango(long):
            vecC.agregar(vecN.extraer(vecN.index(min(vecN, key = lambda x : x[-1]))))
        retornar vecC
    Caso Contrario:
        retornar []
    Fin Si

def evalTiempo(nodo2, dist2, nodo3, nodos, tiempohormiga):
    v = self.mundo.velocity
    flagOntiempo = False
    flagSup = False
    flagDep = False
    flagN3 = False
    tiempo = 0
    Si tipo(nodo2) == lista:
        vent20 = nodo2[0].ventana[0]
        vent21 = nodo2[0].ventana[1]
        tServ2 = nodo2[0].tServicio
    Caso Contrario:
        vent20 = nodo2.ventana[0]
        vent21 = nodo2.ventana[1]
        tServ2 = nodo2.tServicio
    Fin Si

    Si nodo3 != Nulo:
        flagN3 = True
        Si tipo(nodo3) == lista:
            vent30 = nodo3[0].ventana[0]
            vent31 = nodo3[0].ventana[1]
            tServ3 = nodo3[0].tServicio
            Si tipo(nodo2) == lista:
                euc3 = euclidean(nodo2[0].getCoord(), nodo3[0].getCoord()))
                dist3 = (self.mundo.cosVeh) + (self.mundo.cosDis * euc3)
            Caso Contrario:
                euc2 = euclidean(nodo2.getCoord(), nodo3[0].getCoord()))
                dist3 = (self.mundo.cosVeh) + (self.mundo.cosDis * euc3)
            Fin Si
        Caso Contrario:
            vent30 = nodo3.ventana[0]
```

```
            vent31 = nodo3.ventana[1]
            tServ3 = nodo3.tServicio
            Si tipo(nodo2) == lista:
                euc3 = euclidean(nodo2[0].getCoord(), nodo3.getCoord())
                dist3 = (self.mundo.cosVeh) + (self.mundo.cosDis * euc3)
            Caso Contrario:
                euc3 = euclidean(nodo2.getCoord(), nodo3.getCoord())
                dist3 = (self.mundo.cosVeh) + (self.mundo.cosDis * euc3)
            Fin Si
        Fin Si
    Fin Si

    Si tiempo + (dist2/v) < vent20:
        tiempo += vent20 - (dist2/v) - tiempo
        flagOntiempo = True
        tiempo += tServ2
        Si flagN3:
            Si tiempo + (dist3/v) < vent30:
                tiempo += vent30 - (dist3/v) - tiempo
                flagN3 = True
                tiempo += tServ3
            Fin Si
            Si tiempo + (dist3/v) > vent30 && tiempo + (dist3/v) < vent31:
                tiempo += (dist3/v) + tServ3
                flagN3 = True
            Fin Si
            Si tiempo + (dist3/v) > vent31:
                flagN3 = False
            Fin Si
        Fin Si
    Fin Si

    Si tiempo + (dist2/v) > vent20 && tiempo + (dist2/v) < vent21:
        tiempo += (dist2/v) + tServ2
        flagOntiempo = True
        Si flagN3:
            Si tiempo + (dist3/v) < vent30:
                tiempo += vent3 - (dist3/v) - tiempo
                flagN3 = True
                tiempo += tServ3
            Fin Si
            Si tiempo + (dist3/v) > vent30 && tiempo + (dist3/v) < vent31:
                tiempo += (dist3/v) + tServ3
                flagN3 = True
            Fin Si
            Si tiempo + (dist3/v) > vent31:
                flagN3 = False
            Fin Si
        Fin Si
    Fin Si

    Si tiempo + (dist2/v) > vent21:
        flagSup = True
        flagOntiempo = False
        flagN3 = False
        tiempoMax = self.nodoMaxVent(nodos).ventana[1]
        Si tiempo > tiempoMax:
            flagDep = True
        Fin Si
    Fin Si
```

```
retornar([flagOntiempo, flagN3, flagSup, flagDep, tiempo])
```

```
def nodoMaxVent(nodos):
    maxi = 0
    maxN = Nulo
    Para i en nodos:
        Si tipo(i) == lista:
            Si i[0].ventana[1] > maxi:
                maxi = i[0].ventana[1]
                maxN = i[0]
            Fin Si
        Caso Contrario:
            Si i.ventana[1] > maxi:
                maxi = i.ventana[1]
                maxN = i
            Fin Si
        Fin Si
    Fin Para

    retornar maxN

def recargar(eA, distNAEst1, distN2, distN2Est2, distN3, distN3Est3):
    cRate = self.mundo.conRate
    Si eA < cRate * (distN2 + distN2Est2):
        dSi = self.mundo.cargaE
        Si cRate * (distN2 + distN2Est2) < dif:
            Si cRate * (distN2 + distN3 + distN3Est3) < dif:
                eA = (cRate * (distN2 + distN3 + distN3Est3)) + 1
            Caso Contrario:
                eA = (cRate * (distN2 + distN2Est2)) + 1
            Fin Si
        Fin Si

        Si cRate * distNAEst1 < dif:
            eA = (cRate * distNAEst1) + 1
        Fin Si
    Fin Si

    retornar eA

def energiaRst(nodo2):
    eA = cop.deepcopy(self.energia)
    nodo1 = cop.deepcopy(self.nodo())
    coord1 = nodo1.getCoord()
    est1 = self.mundo.estaciones[self.look_charge(coord1)[0]]
    coordEst1 = est1.getCoord()
    distNAEst1 = (self.mundo.cosVeh) + (self.mundo.cosDis * euclidean(coord1, coordEst1))
    Si tipo(nodo2) == lista:
        coord2 = nodo2[0].getCoord()
    Caso Contrario:
        coord2 = nodo2.getCoord()
    Fin Si
    distN2 = (self.mundo.cosVeh) + (self.mundo.cosDis * euclidean(coord1, coord2))
    est2 = self.mundo.estaciones[self.look_charge(coord2)[0]]
    coordEst2 = est2.getCoord()
    distEst1N2 = (self.mundo.cosVeh) + (self.mundo.cosDis * euclidean(coordEst1, coord2))
    distN2Est2 = (self.mundo.cosVeh) + (self.mundo.cosDis * euclidean(coord2, coordEst2))
    cRate = self.mundo.conRate
    eNodo2 = (distN2 * cRate) + (distN2Est2 * cRate)
    eEst1 = distNAEst1 * cRate

    vec = self.Selec14(nodo2)
```

```
    Si longitud(vec) > 0:
        Si vec[0] == nodo2:
            Si longitud(vec) > 1:
                nodo3 = vec[1]
            Caso Contrario:
                nodo3 = Nulo
            Fin Si
        Caso Contrario:
            nodo3 = vec[0]
        Fin Si

        Si nodo3 != Nulo:
            Si tipo(nodo3) == lista:
                coord3 = nodo3[0].getCoord()
            Caso Contrario:
                coord3 = nodo3.getCoord()
            Fin Si
            distN3 = (self.mundo.cosVeh) + (self.mundo.cosDis * euclidean(coord2, coord3))
            est3 = self.mundo.estaciones[self.look_charge(coord3)[0]]
            coordEst3 = est3.getCoord()
            euc = euclidean(coord3, coordEst3)
            distN3Est3 = (self.mundo.cosVeh) + (self.mundo.cosDis * euc)
            eNE3 = (distN3 * cRate) + (distN3Est3 * cRate)
        Fin Si
    Caso Contrario:
        nodo3 = Nulo
    Fin Si

    tiempo = 0
    flagEner = False
    flagN3 = False
    flagEst = False

    Si nodo1.tipo == 'e':
        eA = self.recargar(eA, distNAEst1, distN2, distN2Est2, distN3, distN3Est3)
    Fin Si

    Si eA > eNodo2:
        flagEner = True
        Si nodo3 != Nulo:
            Si eA > (distN2 * cRate) + eNE3:
                flagN3 = True
            Caso Contrario:
                flagN3 = False
            Fin Si
        Caso Contrario:
            flagN3 = False
        Fin Si
    Fin Si

    Si eA > eEst1:
        flagEst = True
    Fin Si

    retornar [flagEner, flagN3, flagEst, nodo2, nodo3, eA, tiempo, vec]

def mercRest(nodoSig):
    Si self.merc >= nodoSig.demanda:
        flagCarga = True
    Caso Contrario:
```

```
        flagCarga = False
    Fin Si

    retornar flagCarga


def nodoFactible(nodo2):

    v = float(self.mundo.velocity)
    recRate = float(self.mundo.recRate)
    conRate = float(self.mundo.conRate)
    cargaE = float(self.mundo.cargaE)

    Si tipo(self.nodo()) == lista:
        coord1 = self.nodo()[0].getCoord()
    Caso Contrario:
        coord1 = self.nodo().getCoord()
    Fin Si

    Si tipo(nodo2) == lista:
        coord2 = nodo2[0].getCoord()
    Caso Contrario:
        coord2 = nodo2.getCoord()
    Fin Si
    tiempo = cop.deepcopy(self.tiempo)
    energia = cop.deepcopy(self.energia)
    flagGeneral = False
    flagDep = False
    flagEst = False
    flagN3 = False

    Si nodo2 != Nulo:
        Si tipo(nodo2) == lista:
            flagCarga = self.mercRest(nodo2[0])
        Caso Contrario:
            flagCarga = self.mercRest(nodo2)
        Fin Si
    Caso Contrario:
        flagCarga = False
    Fin Si

    Si flagCarga:
        dist1 = (self.mundo.cosVeh) + (self.mundo.cosDis * euclidean(coord1, coord2))
        estNodo1 = self.lookCharge(coord1)
        estNodo2 = self.lookCharge(coord2)
        energiaR = self.energiaRst(nodo2)
        energia = energiaR[5]
        tiempo += energiaR[6]
        nodo3 = energiaR[4]
        nodosCerca = energiaR[7]
        tiempoR = self.evalTiempo(nodo2, dist1, nodo3, nodosCerca, tiempo)
        tiempo += tiempoR[4]
        Si energiaR[0] && energiaR[1] && not energiaR[2]:
            flagEst = False
            Si tiempoR[0] && tiempoR[1] && not tiempoR[2] && not tiempoR[3]:
                flagGeneral = True
                flagN3 = True
                flagDep = False
            Fin Si
            Si tiempoR[0] && not tiempoR[1] && not tiempoR[2] && not tiempoR[3]:
```

```
            flagGeneral = True
            flagN3 = False
            flagDep = False
        Fin Si
        Si not tiempoR[0] && not tiempoR[1] && tiempoR[2] && not tiempoR[3]:
            flagGeneral = False
            flagN3 = False
            flagDep = False
        Fin Si
        Si not tiempoR[0] && not tiempoR[1] && tiempoR[2] && tiempoR[3]:
            flagGeneral = False
            flagN3 = False
            flagDep = True
        Fin Si
    Fin Si
    Si energiaR[0] && not energiaR[1] && not energiaR[2]:
        flagN3 = False
        flagEst = False
        Si tiempoR[0] && not tiempoR[2] && not tiempoR[3]:
            flagGeneral = True
            flagDep = False
        Fin Si
        Si not tiempoR[0] && tiempoR[2] && not tiempoR[3]:
            flagGeneral = False
            flagDep = False
        Fin Si
        Si not tiempoR[0] && tiempoR[2] && tiempoR[3]:
            flagGeneral = False
            flagDep = True
        Fin Si
    Fin Si

    Si not energiaR[0] && not energiaR[1] && energiaR[2]:
        flagGeneral = False
        flagN3 = False
        flagEst = True
        Si tiempoR[0] && tiempoR[1] && not tiempoR[2] && not tiempoR[3]:
            flagDep = False
        Fin Si
        Si tiempoR[0] && not tiempoR[1] && not tiempoR[2] && not tiempoR[3]:
            flagDep = False
        Fin Si
        Si not tiempoR[0] && not tiempoR[1] && tiempoR[2] && not tiempoR[3]:
            flagDep = False
        Fin Si
        Si not tiempoR[0] && not tiempoR[1] && tiempoR[2] && tiempoR[3]:
            flagDep = True
        Fin Si
    Caso Contrario:
        flagGeneral = False
        flagN3 = False
        flagEst = False
        flagDep = True
    Fin Si
Caso Contrario:
    energiaR = self.energiaRst(nodo2)
    estNodo1 = self.lookCharge(coord1)
    nodo3 = energiaR[4]
    energia = energiaR[5]
    tiempo += energiaR[6]
```

```
        dist1 = (self.mundo.cosVeh) + (self.mundo.cosDis * euclidean(coord1, coord2))
        nodosCerca = energiaR[7]
        tiempoR = self.evalTiempo(nodo2, dist1, nodo3, nodosCerca, tiempo)
        tiempo += tiempoR[4]
        flagGeneral = False
        flagN3 = False
        flagDep = True
        Si energiaR[2]:
            flagEst = True
        Caso Contrario:
            flagEst = False
        Fin Si
    Fin Si

    retornar [flagGeneral, flagN3, flagEst, flagDep, tiempo, energia, estNodo1, nodo3]

def can_move():
    flagEst = False
    flagmerc = False
    estC = Nulo
    notF = []
    notF.vaciar()
    selec = []
    selec.vaciar()
    selec2 = []
    selec2.vaciar()
    Si self.merc >= min(self.noVisitados, key = lambda x : x.demanda).demanda:
        flagmerc = True
        selec = self.Selec14(self.nodo())
        selec2 = cop.deepcopy(selec)
        Si longitud(selec) > 0:
            sel = []
            sel.vaciar()

            Para i en rango(longitud(selec)):
                nod = selec[i]
                Si tipo(nod) == lista:
                    fl = nod[0].flagV
                Caso Contrario:
                    fl = nod.flagV
                Fin Si

                fact1 = self.nodoFactible(selec[i])

                Si fact1[0] && not fl:
                    sel.agregar(selec[i])
                Fin Si

                Si not fact1[0] && fact1[2] && not fact1[3]:
                    notF.agregar(selec[i])
                Fin Si
            Fin Para

            Si self.nodo().tipo == 'e' or self.nodo() == self.inicio:
                flagEst = True
                Si longitud(self.visitados) > 1:
                    ultNodo = self.visitados[-2]
                    estC = self.lookCharge(self.nodo().getCoord())
                    Mientras estC == ultNodo or estC == self.nodo():
                        estC = self.lookCharge(self.nodo().getCoord())
```

```
                    Fin Mientras
                Fin Si
            Caso Contrario:
                flagEst = False
                Si longitud(self.visitados) > 1:
                    ultNodo = self.visitados[-2]
                    estC = self.lookCharge(self.nodo().getCoord())
                    Mientras estC == ultNodo or estC == self.nodo():
                        estC = self.lookCharge(self.nodo().getCoord())
                    Fin Mientras
                Fin Si
            Fin Si
            retornar [sel, notF, flagEst, flagmerc, estC, selec2]

        Caso Contrario:
            retornar [[], notF, flagEst, flagmerc, estC, selec2]

    Caso Contrario:
        flagmerc = False
        retornar [[], notF, flagEst, flagmerc, estC, selec2]
    Fin Si

def choose_move(choices):
    Si longitud(choices) == 0:
        retornar Nulo
    Fin Si
    Si longitud(choices) == 1:
        Si tipo(choices[0]) == lista:
            retornar choices[0][0]
        Caso Contrario:
            retornar choices[0]
        Fin Si
    Fin Si

    weights = []
    Para move en choices:
        Si tipo(self.nodo()) == lista:
            Si tipo(move) == lista:
                arista = self.mundo.aristas[self.nodo()[0], move[0]]
            Caso Contrario:
                arista = self.mundo.aristas[self.nodo()[0], move]
            Fin Si
        Caso Contrario:
            Si tipo(move) == lista:
                arista = self.mundo.aristas[self.nodo(), move[0]]
            Caso Contrario:
                arista = self.mundo.aristas[self.nodo(), move]
            Fin Si
        Fin Si
        weights.agregar(self.weigh(arista))
    Fin Para

    total = sum(weights)
    cumdist = lista(itertools.accumulate(weights)) + [total]
    choice = choices[bisect.bisect(cumdist, random.random() * total)]

    retornar choice

def move(self):
    remaining = self.can_move()
```

```
choice = self.choose_move(remaining[0])
Si choice != Nulo:
    fact = self.nodoFactible(choice)
Fin Si
Si longitud(remaining[1]) > 0 && remaining[3]:
    choice = self.choose_move(remaining[1])
    fact = self.nodoFactible(choice)
Caso Contrario:
    aristaDep = self.insert_depot()
    retornar self
Fin Si

Si fact[0] && fact[1] && not fact[2] && not fact[3]:
    self.energia = fact[5]
    aristaN2 = self.make_move(choice)
    Si tipo(choice) == lista:
        self.merc -= choice[0].demanda
    Caso Contrario:
        self.merc -= choice.demanda
    Fin Si

    aristaN3 = self.make_move(fact[7])

    Si tipo(fact[7]) == lista:
        self.merc -= fact[7][0].demanda
    Caso Contrario:
        self.merc -= fact[7].demanda
    Fin Si
    retornar self
Fin Si

Si fact[0] && not fact[1] && not fact[2] && not fact[3]:
    self.energia = fact[5]
    aristaN2 = self.make_move(choice)
    Si tipo(choice) == lista:
        self.merc -= choice[0].demanda
    Caso Contrario:
        self.merc -= choice.demanda
    Fin Si
    retornar self
Fin Si

Si not fact[0] && not fact[1] && fact[2] && not fact[3]:
    self.energia = fact[5]
    aristaEst = self.make_move(fact[6])
    retornar self
Fin Si
Si not fact[0] && fact[2] && fact[3]:
    self.energia = fact[5]
    aristaEst = self.make_move(fact[6])
    retornar self
Fin Si
Si not fact[0] && not fact[2] && fact[3]:
    self.energia = fact[5]
    aristaDep = self.insert_depot()
    retornar self
Fin Si

self.tiempo = fact[4]
```

```
def lookCharge(coord):
    veDist = []
    Para i en rango(longitud(self.mundo.estaciones)):
        Si coord != self.mundo.estaciones[i].getCoord():
            veDist.agregar(euclidean(coord, self.mundo.estaciones[i].getCoord()))
        Fin Si
    Fin Para
    menor = min(veDist)
    minInd = veDist.index(menor)
    retornar [self.mundo.estaciones[minInd], menor]

def look_charge(coord):
    veDist = []
    Para i en rango(longitud(estaciones[inst])):
        Si estaciones[inst][str(i)] != coord:
            veDist.agregar(euclidean(coord, estaciones[inst][str(i)]))
        Fin Si
    Fin Para
    menor = min(veDist)
    minInd = veDist.index(menor)
    retornar [minInd, estaciones[inst][str(minInd)], menor]

def insert_depot(self):
    nodo1 = self.nodo()

    Si tipo(nodo1) == lista:
        nodo1 = nodo1[0]
    Fin Si

    tiempo = 0
    coordDep = self.inicio.getCoord()
    coord1 = nodo1.getCoord()
    dist1 = (self.mundo.cosVeh) + (self.mundo.cosDis * euclidean(coord1, coordDep))
    enerDep = self.mundo.conRate * dist1

    Si nodo1.tipo == 'e':

        Si self.energia <= enerDep && enerDep <= self.mundo.cargaE:
            dSi = enerDep - self.energia
            tiempo += dif/self.mundo.recRate
            self.energia = enerDep
            self.make_move(Nulo)
        Fin Si

        Si self.energia <= enerDep && enerDep > self.mundo.cargaE:
            Est = self.look_charge(coord1)
            nodost = self.mundo.estaciones[Est[0]]
            coordEst = nodost.getCoord()
            distNE = (self.mundo.cosVeh) + (self.mundo.cosDis * euclidean(coord1, coordEst))
            enerEst = distNE * self.mundo.conRate
            Si self.energia <= enerEst && enerEst <= self.mundo.cargaE:
                dSi = enerEst - self.energia
                tiempo += dif/self.mundo.recRate
                self.energia = enerEst
                self.make_move(nodost)
                self.insert_depot()
            Fin Si
            Si self.energia >= enerEst:
                self.make_move(nodost)
                self.insert_depot()
```

```
            Fin Si
        Fin Si
        Si self.energia >= enerDep:
            self.make_move(Nulo)
        Fin Si
    Fin Si

    Si nodo1.tipo == 'c':

        Si self.energia <= enerDep:
            Est = self.look_charge(coord1)
            nodost = self.mundo.estaciones[Est[0]]
            coordEst = nodost.getCoord()
            distNE = (self.mundo.cosVeh) + (self.mundo.cosDis * euclidean(coord1, coordEst))
            enerEst = distNE * self.mundo.conRate
            Si self.energia <= enerEst && enerEst <= self.mundo.cargaE:
                dSi = enerEst - self.energia
                tiempo += dif/self.mundo.recRate
                self.energia = enerEst
                self.make_move(nodost)
                self.insert_depot()
            Fin Si
            Si self.energia >= enerEst:
                self.make_move(nodost)
                self.insert_depot()
            Fin Si
        Fin Si
        Si self.energia >= enerDep:
            self.make_move(Nulo)
        Fin Si
    Fin Si

    retornar self

def movimientosRest(self):
    retornar self.noVisitados

def make_move(dest):
    ori = self.nodo()
    Si dest == Nulo:
        dest = self.inicio
        self.visitados.agregar(dest)
        self.tiempo = 0
        self.energia = self.mundo.cargaE)
        self.merc = self.mercMax)
        Si tipo(ori) == lista:
            Si tipo(dest) == lista:
                Si ori[0] != dest[0]:
                    arista = self.mundo.aristas[ori[0], dest[0]]
                Caso Contrario:
                    arista = Nulo
                Fin Si
            Caso Contrario:
                Si ori[0] != dest:
                    arista = self.mundo.aristas[ori[0], dest]
                Caso Contrario:
                    arista = Nulo
                Fin Si
            Fin Si
        Caso Contrario:
```

```
        Si tipo(dest) == lista:
            Si ori != dest[0]:
                arista = self.mundo.aristas[ori, dest[0]]
            Caso Contrario:
                arista = Nulo
            Fin Si
        Caso Contrario:
            Si ori != dest:
                arista = self.mundo.aristas[ori, dest]
            Caso Contrario:
                arista = Nulo
            Fin Si
        Fin Si
    Si arista != Nulo && tipo(arista) == arista && tipo(arista) != hormiga:
        self.arcos.agregar(arista)
    Fin Si
    retornar self
Caso Contrario:
    Si tipo(dest) == lista:
        self.visitados.agregar(dest[0])
        dest[0].flagV = True
    Caso Contrario:
        self.visitados.agregar(dest)
        dest.flagV = True
    Fin Si
Fin Si

Si tipo(ori) == lista:
    coord1 = ori[0].getCoord()
    Si tipo(dest) == lista:
        coord2 = dest[0].getCoord()
        Si ori[0]==dest[0]:
            cosaquenohacenada = Nulo
        Caso Contrario:
            arista = self.mundo.aristas[ori[0], dest[0]]
            Si tipo(arista) == arista && tipo(arista) != hormiga:
                self.arcos.agregar(arista)
            Fin Si
            euc = euclidean(coord1, coord2)
            dist = (self.mundo.cosVeh) + (self.mundo.cosDis * euc)
            self.energia -= self.mundo.conRate * dist
            self.distancia += dist
            self.cont += 1
            retornar self
        Fin Si
    Caso Contrario:
        coord2 = dest.getCoord()
        Si ori[0]==dest:
            cosaquenohacenada = Nulo
        Caso Contrario:
            arista = self.mundo.aristas[ori[0], dest]
            Si tipo(arista) == arista && tipo(arista) != hormiga:
                self.arcos.agregar(arista)
            Fin Si
            euc = euclidean(coord1, coord2)
            dist = (self.mundo.cosVeh) + (self.mundo.cosDis * euc)
            self.energia -= self.mundo.conRate * dist
            self.distancia += dist
            self.cont += 1
            retornar self
```

```
            Fin Si
        Fin Si
    Caso Contrario:
        coord1 = ori.getCoord()
        Si tipo(dest) == lista:
            coord2 = dest[0].getCoord()
            Si ori == dest[0]:
                cosaquenohacenada = Nulo
            Caso Contrario:
                arista = self.mundo.aristas[ori, dest[0]]
                Si tipo(arista) == arista && tipo(arista) != hormiga:
                    self.arcos.agregar(arista)
                Fin Si
                euc = euclidean(coord1, coord2)
                dist = (self.mundo.cosVeh) + (self.mundo.cosDis * euc)
                self.energia -= self.mundo.conRate * dist
                self.distancia += dist
                self.cont += 1
                retornar self
            Fin Si
        Caso Contrario:
            coord2 = dest.getCoord()
            Si ori == dest:
                cosaquenohacenada = Nulo
            Caso Contrario:
                arista = self.mundo.aristas[ori, dest]
                Si tipo(arista) == arista && tipo(arista) != hormiga:
                    self.arcos.agregar(arista)
                Fin Si
                euc = euclidean(coord1, coord2)
                dist = (self.mundo.cosVeh) + (self.mundo.cosDis * euc)
                self.energia -= self.mundo.conRate * dist
                self.distancia += dist
                self.cont += 1
                retornar self
            Fin Si
        Fin Si
    Fin Si

def weigh(arista):
    pre = 1 / (arista.longitud)
    post = arista.feromona
    retornar post ** self.alfa * pre ** self.beta

def areUnv(self):
    Para i en self.noVisitados:
        Si not i.flagV:
            retornar True
        Fin Si
    Fin Para
```

```
Clase Solucionador:

def inicializar(alfa, beta, merca, energia, costoVeh, costoDis, inst):
    self.alfa = alfa
    self.beta = beta
    self.merca = merca
    self.energia = energia
    self.costoVeh = costoVeh
    self.costoDis = costoDis
    self.noHormigas = noHormigas

def crearColonia(mundo):

    Si self.noHormigas < 1:
        retornar self.hormigasAlrededor(mundo, len(mundo.nodos))
    Fin Si
    retornar self.hormigasAleatorias(mundo, self.noHormigas)

def reiniciarColonia(colonia):
    para hormiga en colonia:
        hormiga.inicializar2(hormiga.mundo, inicio=hormiga.mundo.nodos[0])

def antColonyOpt(colonia):
    cont = self.soluciones(colonia)
    self.act_global(colonia)
    retornar ordenado(colonia)[cont]

def solve(mundo):

    mundo.reiniciar_feromona(self.t0)
    mejorGlobal = Nulo
    colonia = self.crearColonia(mundo)
    para i en rango(self.limite):
        self.reiniciarColonia(colonia)
        mejorLocal = self.antColonyOpt(colonia)
        Si mejorGlobal is Nulo or mejorLocal.distancia < mejorGlobal.distancia:
            mejorGlobal = mejorLocal
        Fin Si
        self.elite(mejorGlobal)
    Fin para
    retornar mejorGlobal

def soluciones2(mundo):

    mundo.reiniciar_feromona(self.t0)
    mejorGlobal = Nulo
    colonia = self.crearColonia(mundo)
    para i en rango(self.limite):
        self.reiniciarColonia(colonia)
        mejorLocal = self.antColonyOpt(colonia)
        Si mejorGlobal is Nulo or mejorLocal < mejorGlobal:
            mejorGlobal = mejorLocal
            yield mejorGlobal
        Fin Si
        self.elite(mejorGlobal)
    Fin para
```

```
def hormigasAlrededor(mundo, cont):

    inicios = mundo.nodos
    n = len(inicios)
    alfa = self.alfa
    beta = self.beta
    merca = self.merca
    energia = self.energia
    cosVeh = self.costoVeh
    cosDis = self.costoDis

    retornar [
        hormiga(alfa, beta, merca, energia, cosVeh, cosDis).inicializar(
            mundo, inicio=inicios[i % n])
        para i en rango(cont)
    ]

def hormigasAleatorias(mundo, cont, even=False):

    hormigas = []
    inicios = mundo.nodos
    n = len(inicios)
    alfa = self.alfa
    beta = self.beta
    merca = self.merca
    energia = self.energia
    cosVeh = self.costoVeh
    cosDis = self.costoDis
    Si even:
        Si cont > n:
            para i en rango(self.noHormigas // n):
                hormigas.extend([
                    hormiga(alfa, beta, merca, energia, costoVeh, costoDis, inst).inicializar(
                        mundo, inicio=inicios[j])
                    para j en rango(n)
                ])
            Fin para
        Fin Si
        hormigas.extend([
            hormiga(alfa, beta, merca, energia, costoVeh, costoDis, inst).inicializar(
                mundo, inicio=inicios.pop(random.randrango(n - i)))
            para i en rango(cont % n)
        ])
    CasoContrario:
        # Just pick random nodos.
        hormigas.extend([
            hormiga(alfa, beta, merca, energia, costoVeh, costoDis, inst).inicializar(
                mundo, inicio=inicios[random.randrango(n)])
            para i en rango(cont)
        ])
    Fin Si
    retornar hormigas
```

```
def soluciones(hormigas):

    hormigasFin = 0
    mientras hormigasFin < len(hormigas):
        hormigasFin = 0
        contadorHor = 0
        para hormiga en hormigas:
            para nodo en hormiga.mundo.nodos:
                nodo.flagV = False
            Fin para
            cont1 = 0
            cont2 = 0
            notf1 = 0
            mientras hormiga.areUnv:
                can = hormiga.can_move()
                Si len(can[0]) > 0 or len(can[1]) > 0:
                    hormiga = hormiga.move()
                    arista = hormiga.viajados
                    para i en hormiga.viajados:
                        self.act_local(i)
                    Si len(can[1]) == notf1:
                        cont1 += 1
                    CasoContrario:
                        cont1 = 0

                    Si cont1 == 20:
                        contadorHor += 1
                        break
                CasoContrario:
                    hormiga = hormiga.move()
                    Si len(can[-1]) == 0:
                        cont2 += 1
                    CasoContrario:
                        cont2 = 0
                    Fin Si
                    Si cont2 == 20:
                        break
                    Fin Si
                Fin Si
                notf1 = len(can[1])
            Fin Mientras
            hormigasFin += 1
            print(hormigasFin)
        Fin para
    Fin Mientras
    retornar contadorHor

def act_local(arista):
    Si tipo(arista) != Nulo and arista != Nulo:
        Si tipo(arista) == tupla:
            arista[0].feromona = max(self.t0, arista[0].feromona * self.rho)
        CasoContrario:
            arista.feromona = max(self.t0, arista.feromona * self.rho)
        Fin Si
    Fin Si}
```

```
def act_global(hormigas):
    hormigas = ordenado(hormigas)[:len(hormigas) // 2]
    para a en hormigas:
        Si a.distancia > 0:
            p = self.q / a.distancia
            para arista en a.camino():
                Si arista != Nulo:
                    Si tipo(arista) == tupla:
                        f=max(self.t0, (1 - self.rho) * arista[0].feromona + p)
                        arista[0].feromona = f
                    CasoContrario:
                        f=max(self.t0, (1 - self.rho) * arista.feromona + p)
                        arista.feromona = f
                    Fin Si
                Fin Si
            Fin para
        Fin Si
    Fin para

def elite(hormiga):
    Si self.elite:
        Si hormiga.distancia > 0:
            p = self.elite * self.q / hormiga.distancia
            para arista en hormiga.camino():
                Si arista != Nulo:
                    Si tipo(arista) == tupla:
                        arista[0].feromona += p
                    CasoContrario:
                        arista.feromona += p
                    Fin Si
                Fin Si
            Fin para
        Fin Si
    Fin Si
```