

## ▼ LIBRERIAS

```
1 import random as rd
2 from operator import itemgetter
3 from random import randint, random
4
```

## ▼ CLASE CURSO

```
1 class ClaseCurso:
2     def __init__(self, codAsignatura, nombre, docente, horario):
3         self.codAsignatura=codAsignatura
4         self.nombre=nombre
5         self.docente=docente
6         self.horario=horario
7     def mostrar(self):
8         print(f'{self.codAsignatura} {self.nombre} {self.docente} {self.horario}')
```

## ▼ CLASE DIA

```
1 class ClaseDia:
2     def __init__(self, dia, horaI, horaF, tipo="T"):
3         self.dia=dia
4         self.horaI=horaI
5         self.horaF=horaF
6         self.tipo=tipo    #teo = teorico, lab=laboratorio
```

## ▼ CLASE GENETICO

```
1 class claseGenetico():
2     def __init__(self, _cursos, _aulas, _n, _listaCursos:list, _cantLaboratorios):
3         self.probaMutacion=0.80
4         self.probaCruce=0.5
5         self.poblacion=None
6         self.individuo=None
7         self.evaluar=None
8         self.fo=None
9         self.puntuacion=[]
10        self.cursos=_cursos
11        self.aulas=_aulas
12        self.n=_n
13        self.listaCursos=_listaCursos
14        self.laboratorios=None
15        self.cantidadLaboratorios=_cantLaboratorios
16
17        #Generamos un individuo
18        def generarIndividuo(self):
19            individuo=[[rd.randint(0,1) for k in range(self.cursos)] for j in range(self.aulas)]
20            return individuo
21
22        #Generamos la poblacion de individuos
23        def generarPoblacion(self):
24            self.puntuacion=[]
25            self.poblacion=[self.generarIndividuo() for i in range(self.n)]
26            # [[1, 0, 1, 0, 1, 1, 1, 1, 0, 0], [1, 0, 1, 0, 0, 0, 0, 0, 1, 1], [1, 0, 0, 0, 1, 1, 1, 0, 1, 1]],
27            # [[0, 1, 1, 1, 0, 1, 0, 0, 0, 0], [1, 1, 0, 0, 0, 0, 0, 0, 0, 1], [1, 1, 1, 0, 1, 1, 1, 1, 0, 0]] ]
28
29        #Esta funcion nos sirve para seleccionar a aquellos que son
30        #mas aptos.
31        def NuevaRuleta(self, pais): #SELECCION DE RULETA
32            def sortear(fitness_total, indice_a_ignorar=-1):
33                ruleta, acumulado, valor_sortead = [], 0, random()
34
35                if indice_a_ignorar!=-1:
36                    fitness_total -= valores[0][indice_a_ignorar]
37
38                for indice, i in enumerate(valores[0]):
39                    if indice_a_ignorar==indice:
40                        continue
```

```

41         acumulado += i
42         ruleta.append(acumulado/fitness_total)
43         if ruleta[-1] >= valor_sorteado:
44             return indice
45
46     valores = list(zip(*pais))
47
48     fitness_total = sum(valores[0])
49
50     #Recuperamos el indice de la molecula mas apto
51     indice_padre = sortear(fitness_total)
52
53     #Recuperamos el indice de la segunda molecula mas apto
54     indice_madre = sortear(fitness_total, indice_padre)
55
56     #Recuperamos el valor con el indice
57     padre = valores[1][indice_padre]
58     madre = valores[1][indice_madre]
59
60     return padre, madre
61
62     #[ AULAS , AULAS , AULAS , AULAS , AULAS ]
63     #[ [0,1,0,1,0,1] ] , [0,1,0,1,0,1] , [0,1,0,1,0,1] ]
64     #[[1, 0, 1, 0, 1, 1, 1, 1, 0, 0], [1, 0, 1, 0, 0, 0, 0, 1, 1], [1, 0, 0, 0, 1, 1, 1, 0, 1, 1]]
65     def determinarHorariosLaboratorios(self, curso):
66         aula=[]
67         dias={"LUNES":0,"MARTES":1,"MIÉRCOLES":2,"JUEVES":3,"VIERNES":4,"SÁBADO":5,"SABADO ":5}
68         lab=[]
69         for curso in curso.horario:
70             for num in range(curso.horaI,curso.horaF):
71                 if(curso.tipo=="P"):
72                     lab.append((dias[curso.dia],num))
73                 else:
74                     aula.append((dias[curso.dia],num))
75         return lab,aula
76
77     def generarAuxDocentes(self):
78         diccionario={}
79         for valor in self.listaCursos:
80             diccionario[valor.docente]=-1
81         return diccionario
82
83     def determinarSiEntraCurso(self, cromosoma):
84         aulasDisponibles=[]
85         labosDisponibles=[]
86         for aula in cromosoma:
87             aulaExtra=[]
88             labExtra=[]
89             for indice in range(len(aula)):
90                 if(aula[indice]==0):
91                     continue
92                 else:
93                     laboratorioBinario,aulasBinario=self.determinarHorariosLaboratorios(self.listaCursos[indice])
94                     labExtra+=laboratorioBinario
95                     aulaExtra+=aulasBinario
96             aulasDisponibles.append(aulaExtra)
97             labosDisponibles.append(labExtra)
98         return aulasDisponibles,labosDisponibles
99     def cantidadRepetidos(self, lista):
100         return dict(zip(lista, map(lambda x: lista.count(x), lista)))
101
102     # Encontrar los cursos que se cruzan horas en un mismo aula y laboratorio
103     def ECT(self, cromosoma):
104         def contarCastigos(aulasDisponibles):
105             if(len(aulasDisponibles) in [0,1]):
106                 return 0
107             castigo=0
108             for valor in aulasDisponibles:
109                 diccionario=self.cantidadRepetidos(valor)
110                 for key in diccionario:
111                     if(diccionario[key]>1):
112                         castigo+=1
113             return castigo
114         castigoTotal=0
115         aulasDisponibles,labosDisponibles=self.determinarSiEntraCurso(cromosoma)
116         castigoTotal+=contarCastigos(aulasDisponibles)
117         #[[ (0, 16), (0, 17)], [(1, 7), (1, 8), (4, 7), (0, 9), (0, 10), (4, 8), (0, 16), (0, 17)], [(0, 16), (0, 17)]]
118         totallabos=[]
119         for valor in labosDisponibles:
120             totallabos+=valor
121         reparticionDeLabos=[totallabos[i:i + self.cantidadLaboratorios] for i in range(0, len(totallabos), self.cantidadLaborato
122         for valor in reparticionDeLabos:

```

```

123         castigoTotal+=contarCastigos(valor)
124     return castigoTotal
125
126 # Calcular los cursos que se repiten en otras aulas
127 def ECDAT(self,cromosoma):
128     castigo=0
129     primero=cromosoma[0]
130     for x in range(len(cromosoma)):
131         for y in range(len(cromosoma[x])):
132             if(cromosoma[x][y]==1):
133                 for z in range(len(cromosoma)):
134                     if(cromosoma[z][y]==1 and z!=x):
135                         castigo+=1
136     return castigo//2
137
138
139 #[[1, 0, 1, 0, 1, 1, 1, 1, 0, 0], [1, 0, 1, 0, 0, 0, 0, 0, 1, 1], [1, 0, 0, 0, 1, 1, 1, 0, 1, 1]]
140 #Paul
141
142 def funcionObjetivo(self,cromosoma):
143     #[[1, 0, 1, 0, 1, 1, 1, 1, 0, 0], [1, 0, 1, 0, 0, 0, 0, 0, 1, 1], [1, 0, 0, 0, 1, 1, 1, 0, 1, 1]]
144     #return 1/(1+randint(1,5)+randint(1,5))
145     return 1/(1+self.ECT(cromosoma)+2*self.ECDAT(cromosoma))
146 def evaluacion(self):
147     self.evaluar=[(self.funcionObjetivo(valor),valor) for valor in self.poblacion if(self.funcionObjetivo(valor)<=1)]
148     # ( FuncionObjetivo(valor),valor)
149     # ( 121 , [[1, 0, 1], [0, 1, 1]])
150     # ( 88 , [[1, 0, 1], [0, 1, 1]])
151
152     #Ordenar
153     sorted(self.evaluar , key=itemgetter(0))
154
155     puntuacion=0
156     # ----- ALMACENANDO LA PUNTUACION DE CADA GENERACION -----
157     for valor in self.evaluar:
158         puntuacion+=valor[0]
159     self.puntuacion.append(puntuacion/len(self.evaluar))
160
161     # ----- CRUCE -----
162     hijos=[]
163     # Itearemos cada hijo
164     while(len(hijos)<self.n):
165         #Almacenamos las moleculas mas aptas en padre y madre
166         padre,madre=self.NuevaRuleta(self.evaluar)
167
168         #Generamos un numero entre 0 y 1 con "radom()" y si es menor a la
169         #probabilidad de cruce -> Cruzamos padre y madre
170         if(self.probaCruce>random()):
171             for i in range(len(padre)):
172                 for j in range(self.cursos//2):
173                     madre[i][j]=padre[i][j]
174             hijos.append(madre)
175
176     #----- MUTACION -----
177     for individuo in hijos:
178         #Hacemos lo mismo de cruce, generamos un random y lo comparamos con
179         #la probabilidad de MUTACION
180         if(self.probaMutacion>random()):
181             indice_aleatorio=randint(0, len(individuo)-1)
182             indice_aleatorioBinario=randint(0, len(individuo[indice_aleatorio])-1)
183             #aQUI CAMBIAMOS LOS BIT Si es 0 a 1 y de 1 a 0
184             if individuo[indice_aleatorio][indice_aleatorioBinario] == 1:
185                 individuo[indice_aleatorio][indice_aleatorioBinario] = 0
186             else:
187                 individuo[indice_aleatorio][indice_aleatorioBinario] = 1
188     self.poblacion=hijos
189     None==madre==padre==hijos
190 def info(self):
191     print("")
192     print("----- INFO -----")
193     print(f'Tam de cursos: {self.cursos}')
194     print(f'Tam de aulas : {self.aulas}')
195     print(f'Tam de pobl : {self.n}')
196     print(f'Puntuacion : {self.puntuacion}')
197     print("-----")
198
199 def entrenar(self):
200     self.evaluacion()
201 def obtenerPoblacion(self):
202     return self.poblacion
203 def obtenerPuntuacion(self):

```

```
204         return self.puntuacion
```

## ▼ CLASE GENETICO HIBRIDO GREEDY

```
1 class claseGeneticoHibridoGreedy():
2     def __init__(self, _cursos, _aulas, _n, _listaCursos:list, _cantLaboratorios):
3         self.probaMutacion=0.80
4         self.probaCruce=0.5
5         self.poblacion=None
6         self.individuo=None
7         self.evaluar=None
8         self.fo=None
9         self.puntuacion=[]
10        self.cursos=_cursos
11        self.aulas=_aulas
12        self.n=_n
13        self.listaCursos=_listaCursos
14        self.laboratorios=None
15        self.cantidadLaboratorios=_cantLaboratorios
16
17        #Generamos un individuo
18        def generarIndividuo(self):
19            individuo=[[rd.randint(0,1) for k in range(self.cursos)] for j in range(self.aulas)]
20            return individuo
21
22        #Generamos la poblacion de individuos
23        def generarPoblacion(self):
24            self.puntuacion=[]
25            self.poblacion=[self.generarIndividuo() for i in range(self.n)]
26        # [[1, 0, 1, 0, 1, 1, 1, 1, 0, 0], [1, 0, 1, 0, 0, 0, 0, 0, 1, 1], [1, 0, 0, 0, 1, 1, 1, 0, 1, 1]],
27        # [[0, 1, 1, 1, 0, 1, 0, 0, 0, 0], [1, 1, 0, 0, 0, 0, 0, 0, 0, 1], [1, 1, 1, 0, 1, 1, 1, 1, 0, 0]] ]
28
29        # [ AULAS , AULAS , AULAS , AULAS , AULAS ]
30        # [ [0,1,0,1,0,1] ] , [0,1,0,1,0,1] , [0,1,0,1,0,1] ]
31        # [[1, 0, 1, 0, 1, 1, 1, 1, 0, 0], [1, 0, 1, 0, 0, 0, 0, 0, 1, 1], [1, 0, 0, 0, 1, 1, 1, 0, 1, 1]]
32        def determinarHorariosLaboratorios(self, curso):
33            aula=[]
34            dias={"LUNES":0, "MARTES":1, "MIÉRCOLES":2, "JUEVES":3, "VIERNES":4, "SÁBADO":5, "SABADO ":5}
35            lab=[]
36            for curso in curso.horario:
37                for num in range(curso.horaI, curso.horaF):
38                    if(curso.tipo=="lab"):
39                        lab.append((dias[curso.dia], num))
40                    else:
41                        aula.append((dias[curso.dia], num))
42            return lab, aula
43
44        def generarAuxDocentes(self):
45            diccionario={}
46            for valor in self.listaCursos:
47                diccionario[valor.docente]=-1
48            return diccionario
49
50        def determinarSiEntraCurso(self, cromosoma):
51            aulasDisponibles=[]
52            labosDisponibles=[]
53            for aula in cromosoma:
54                aulaExtra=[]
55                labExtra=[]
56                for indice in range(len(aula)):
57                    if(aula[indice]==0):
58                        continue
59                    else:
60                        laboratorioBinario, aulasBinario=self.determinarHorariosLaboratorios(self.listaCursos[indice])
61                        labExtra+=laboratorioBinario
62                        aulaExtra+=aulasBinario
63                        aulasDisponibles.append(aulaExtra)
64                        labosDisponibles.append(labExtra)
65            return aulasDisponibles, labosDisponibles
66        def cantidadRepetidos(self, lista):
67            return dict(zip(lista, map(lambda x: lista.count(x), lista)))
68
69        # Encontrar los cursos que se cruzan horas en un mismo aula y laboratorio
70        def ECT(self, cromosoma):
71            def contarCastigos(aulasDisponibles):
72                if(len(aulasDisponibles) in [0,1]):
73                    return 0
74                castigo=0
75                for valor in aulasDisponibles:
76                    diccionario=self.cantidadRepetidos(valor)
```

```

77         for key in diccionario:
78             if(diccionario[key]>1):
79                 castigo+=1
80         return castigo
81     castigoTotal=0
82     aulasDisponibles,labosDisponibles=self.determinarSiEntraCurso(cromosoma)
83     castigoTotal+=contarCastigos(aulasDisponibles)
84     #[(0, 16), (0, 17)], [(1, 7), (1, 8), (4, 7), (0, 9), (0, 10), (4, 8), (0, 16), (0, 17)], [(0, 16), (0, 17)]
85     totallabos=[]
86     for valor in labosDisponibles:
87         totallabos+=valor
88     reparticionDeLabos=[totallabos[i:i + self.cantidadLaboratorios] for i in range(0, len(totallabos), self.cantidadLaborato
89     for valor in reparticionDeLabos:
90         castigoTotal+=contarCastigos(valor)
91     return castigoTotal
92
93 # Calcular los cursos que se repiten en otras aulas
94 def ECDAT(self,cromosoma):
95     castigo=0
96     primero=cromosoma[0]
97     for x in range(len(cromosoma)):
98         for y in range(len(cromosoma[x])):
99             if(cromosoma[x][y]==1):
100                 for z in range(len(cromosoma)):
101                     if(cromosoma[z][y]==1 and z!=x):
102                         castigo+=1
103     return castigo//2
104
105
106 #[[1, 0, 1, 0, 1, 1, 1, 1, 0, 0], [1, 0, 1, 0, 0, 0, 0, 0, 1, 1], [1, 0, 0, 0, 1, 1, 1, 0, 1, 1]]
107 #Paul
108
109 def funcionObjetivo(self,cromosoma):
110     #[[1, 0, 1, 0, 1, 1, 1, 1, 0, 0], [1, 0, 1, 0, 0, 0, 0, 0, 1, 1], [1, 0, 0, 0, 1, 1, 1, 0, 1, 1]]
111     #return 1/(1+randint(1,5)+randint(1,5))
112     return 1/(1+self.ECT(cromosoma)+2*self.ECDAT(cromosoma))
113
114
115 # USAR EL ALGORITMO GREEDY PARA SELECCIONAR LA SOLUCION OPTIMA
116 def AlgoritmoGreddy(self,pais): #SELECCION DE RULETA
117     valores = list(zip(*pais))
118     #[(0.09090909090909091, 0.2), ([1, 1, 1, 0, 1], [1, 0, 1, 1, 0], [0, 0, 1, 0, 1]), ([1, 1, 0, 0, 0], [1, 0, 0, 0, 1], [
119
120     mejorValor = -999
121     mejorPos=0
122     for indice in range(len(valores[0])):
123         if(mejorValor<valores[0][indice]):
124             if(valores[0][indice]==1.0:
125                 mejorValor=valores[0][indice]
126                 mejorPos=indice
127                 break
128     mejorValor=valores[0][indice]
129     mejorPos=indice
130
131     #Recuperamos el indice de la molecula mas apto
132     indice_padre = mejorPos
133
134     #Recuperamos el indice de la segunda molecula mas apto
135     indice_madre = mejorPos
136
137     #Recuperamos el valor con el indice
138     padre = valores[1][indice_padre]
139     madre = valores[1][indice_madre]
140
141     return padre, madre
142 def evaluacion(self):
143     self.evaluacion=[(self.funcionObjetivo(valor),valor) for valor in self.poblacion if(self.funcionObjetivo(valor)<=1)]
144     # ( FuncionObjetivo(valor),valor)
145     # ( 121 , [[1, 0, 1], [0, 1, 1]])
146     # ( 88 , [[1, 0, 1], [0, 1, 1]])
147
148     #Ordenar
149     sorted(self.evaluacion , key=itemgetter(0))
150
151     puntuacion=0
152     # ----- ALMACENANDO LA PUNTUACION DE CADA GENERACION -----
153     for valor in self.evaluacion:
154         puntuacion+=valor[0]
155     self.puntuacion.append(puntuacion/len(self.evaluacion))
156
157     # ----- CRUCE -----
158     hijos=[]

```

```

159     # Itearamos cada hijo
160     while(len(hijos)<self.n):
161         #Almacenamos las moleculas mas aptas en padre y madre
162         padre,madre=self.AlgoritmoGredy(self.evaluar)
163
164         #Generamos un numero entre 0 y 1 con "radom()" y si es menor a la
165         #probabilidad de cruce -> Cruzamos padre y madre
166         if(self.probaCruce>random()):
167             for i in range(len(padre)):
168                 for j in range(self.cursos//2):
169                     madre[i][j]=padre[i][j]
170             hijos.append(madre)
171
172     #----- MUTACION -----
173     for individuo in hijos:
174         #Hacemos lo mismo de cruce, generamos un random y lo comparamos con
175         #la probabilidad de MUTACION
176         if(self.probaMutacion>random()):
177             indice_aleatorio=randint(0, len(individuo)-1)
178             indice_aleatorioBinario=randint(0, len(individuo[indice_aleatorio])-1)
179             #aQUI CAMBIAMOS LOS BIT Si es 0 a 1 y de 1 a 0
180             if individuo[indice_aleatorio][indice_aleatorioBinario] == 1:
181                 individuo[indice_aleatorio][indice_aleatorioBinario] = 0
182             else:
183                 individuo[indice_aleatorio][indice_aleatorioBinario] = 1
184     self.poblacion=hijos
185     None==madre==padre==hijos
186     def info(self):
187         print("")
188         print("----- INFO -----")
189         print(f'Tam de cursos: {self.cursos}')
190         print(f'Tam de aulas : {self.aulas}')
191         print(f'Tam de pobl : {self.n}')
192         print(f'Puntuacion : {self.puntuacion}')
193         print("-----")
194
195     def entrenar(self):
196         self.evaluacion()
197     def obtenerPoblacion(self):
198         return self.poblacion
199     def obtenerPuntuacion(self):
200         return self.puntuacion
201

```

## ▼ INPUT

```

1 import pandas as pd
2 import matplotlib.pyplot as plt
3 import seaborn as sns
4 import numpy as np
5 from sklearn import preprocessing
6
7
8 def ordenar_lista_curso(lista):
9     lista_dias=['LUNES','MARTES','MIÉRCOLES','JUEVES','VIERNES','SÁBADO','SABADO ']
10    lista_order=[]
11    for dia in lista_dias1:
12        for k in range(len(lista)):
13            #print(dia)
14            if lista[k][0]==dia:
15                lista_order.append(lista[k])
16    return lista_order
17
18 def orden_codigo(dataset):
19     #print(dataset)
20     unicos=dataset['CODIGO'].unique()
21     #print(unicos)
22     lista_horario=[]
23     for u in unicos: #recorremos los valores unicos encontrados
24         lista_unica=[]
25         for _ in range(len(dataset)):#recorremos todo el data set
26             if u==dataset.iloc[_][1]:#verificamos
27                 lista_unica.append([dataset.iloc[_][9],dataset.iloc[_][10],dataset.iloc[_][11],dataset.iloc[_][5]])
28     #verificamos la lista unica y ordenamos
29     #print(lista_unica)
30     lista_unica=ordenar_lista_curso(lista_unica)
31     #print(lista_unica)
32     lista_horario.append(lista_unica)
33     return(lista_horario)
34

```

```

35
36 def buscar_codigo_docent(dataset):
37     #print(dataset)
38     unicos=dataset['CODIGO'].unique()
39     #print(unicos)
40     lista_do=[]
41     for u in unicos: #recorremos los valores unicos encontrados de codigos
42         for _ in range(len(dataset)): #recorremos todo el data set
43             if u==dataset.iloc[_ ,1]: #verificamos
44                 lista_do.append([u,dataset.iloc[_ ,3],dataset.iloc[_ ,15]])
45     return(lista_do)
46
47
48 def unir_listas(prof,horario):
49     lista_unida=[]
50     tam=0
51     k=0
52     for i in horario:
53         print(tam)
54         lista_unida=prof[tam]+horario[k]
55         tam=tam+len(i)
56         k=k+1
57         print(lista_unida)
58
59 #qr nombre docente y codigo
60 #w horarios cada uno tiene un elemento
61 def unir_listas(prof,horario):
62     lista_unida=[]
63     tam=0
64     k=0
65     for i in horario:
66         lista_unir=prof[tam]+[horario[k]]
67         tam=tam+len(i)
68         k=k+1
69         lista_unida.append(lista_unir)
70     return(lista_unida)
71
72
73 def convertir_objetos(lista_unida):
74     totalCursos=[]
75     for i in lista_unida1:
76         horario=[]
77         #print(i)
78         for k in range(len(i[3])):
79             horario.append(ClaseDia(i[3][k][0],i[3][k][1],i[3][k][2],i[3][k][3]))
80         totalCursos.append(ClaseCurso(i[0],i[1],i[2],horario))
81     return(totalCursos)
82
83
84 #----- INPUT PRINCIPAL -----
85 dataset = pd.read_csv("cargaAcademica.csv",sep=';')
86 len(dataset)
87 dataset_copy = dataset.loc[dataset['CARRERA'] == 'INGENIERIA INFORMATICA']
88 dataset_copy=dataset_copy.loc[dataset_copy['DOCENTES']!='CURSO DESACTIVADO']
89
90 w=orden_codigo(dataset_copy)
91 qr=buscar_codigo_docent(dataset_copy)
92 unir_listas(qr,w)
93 lista_unida1=unir_listas(qr,w)
94 totalCursos=convertir_objetos(lista_unida1)

```

1 totalCursos

1 dataset

	N°	CODIGO	CARRERA	CURSO	CRED.	TIPO	GPO	HT	HP	DIA	HR/INICIO
0	1	IF060AIN	INGENIERIA INFORMATICA	MUSICA	2	P	A	0	2	VIERNES	11
1	2	IF060AIN	INGENIERIA INFORMATICA	MUSICA	2	P	A	0	2	SÁBADO	9
2	3	IF063AIN	INGENIERIA INFORMATICA	QUECHUA	2	P	A	0	2	VIERNES	7
3	4	IF063AIN	INGENIERIA INFORMATICA	QUECHUA	2	P	A	0	2	SÁBADO	7
4	5	IF063BIN	INGENIERIA INFORMATICA	QUECHUA	2	P	B	0	2	VIERNES	14
...	...	...	...	...	...	...	...	...	...	...	...
334	335	IF902AMD / IF902AOD	MEDICINA HUMANA / ODONTOLOGÍA	TECNOLOGIAS DE LA INFORMACION Y LA COMUNICACIÓN	3	T	A	2	0	MIÉRCOLES	16
335	336	IF902AMD / IF902AOD	MEDICINA HUMANA / ODONTOLOGÍA	TECNOLOGIAS DE LA INFORMACION Y LA COMUNICACIÓN	3	P	A	0	2	LUNES	16

▼ PRUEBAS

COMUNICACIÓN

▼ Pruebas Algoritmo Genetico

```
1 import time
2 import matplotlib.pyplot as plt
3 import pandas as pd
4
5 #Inicializamos la clase genetico
6 var = claseGenetico(_cursos=len(totalCursos),_aulas=16,_n=40,_listaCursos=totalCursos,_cantLaboratorios=13)
7
8 #Damos el valor para iterar las generaciones
9 generaciones = 20
10
11
12
13 # Inicializamos variable para actualizar el ultimo valor
14 ultimovalor = 0
15
16 tiempos1 = []
17 mejorValorGenetico=[]
18 #Iteramos 20 veces
19 for i in range(20):
20     start_time = time.time() #Tomamos el tiempo antes de ejecutar el programa
21
22     #Generamos la primera poblacion
23     var.generarPoblacion()
24     aux=1
25     #-----
26     #Empezamos con las iteraciones
27     while aux<generaciones:
28         #Entrenamos nuestra poblacion con cada iteracion
29         var.entrenar()
30         if(var.puntuacion[-1]==1.0):
31             break
32
33         aux+=1
34     end_time = time.time()
35     mejorValorGenetico.append(max(var.puntuacion))
36     tiempos1.append(end_time - start_time) #Tomamos el tiempo después de ejecutar y lo agregamos a la lista
37
38 #Creamos un DataFrame con los tiempos de ejecución
39 df1 = pd.DataFrame({'Iteración': range(1,21), 'Tiempo de ejecución': tiempos1})
40 #Mostramos la tabla
```



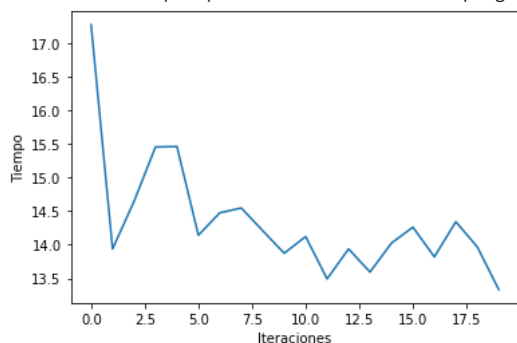
```

41 print(df1)
42
43 # GRAFICO DE TIEMPOS
44 m = [tiempos1[i] for i in range(0,20)]
45 print("Grafico de Tiempos para 20 iteraciones del programa")
46 plt.plot(m)
47 plt.xlabel('Iteraciones')
48 plt.ylabel('Tiempo')
49 plt.show()
50
51 # IMPRIMIR PROMEDIO DE TIEMPOS
52 print("El promedio de tiempo de las iteraciones es: " + str(df1['Tiempo de ejecución'].mean()))
53
54 # GUARDAMOS EL ULTIMO VALOR
55 ultimovalor = var.puntuacion[-1]
56
57 #-----
58
59 # Ver datos
60 # ultimo valor
61 print("Ultimo valor: " + str(ultimovalor))
62 mejorValorGenetico.sort()
63 y=var.obtenerPuntuacion()
64 x=[i for i in range(1,len(y)+1)]
65 print("---- Generado correctamente -----")
66 plt.plot(x,y)
67 plt.show()

```

	Iteración	Tiempo de ejecución
0	1	17.269562
1	2	13.938836
2	3	14.644900
3	4	15.453417
4	5	15.460603
5	6	14.142626
6	7	14.475558
7	8	14.546813
8	9	14.208892
9	10	13.874411
10	11	14.122252
11	12	13.492953
12	13	13.938710
13	14	13.594846
14	15	14.025112
15	16	14.261368
16	17	13.821245
17	18	14.342440
18	19	13.968910
19	20	13.334642

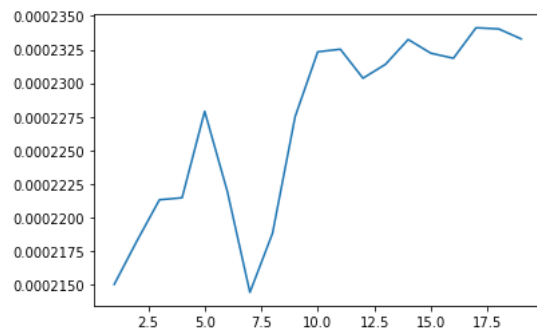
Grafico de Tiempos para 20 iteraciones del programa



El promedio de tiempo de las iteraciones es: 14.345904874801636

Ultimo valor: 0.00023329406573791077

---- Generado correctamente -----



## ▼ Pruebas Algoritmo Genetico Hibrido con Greedy

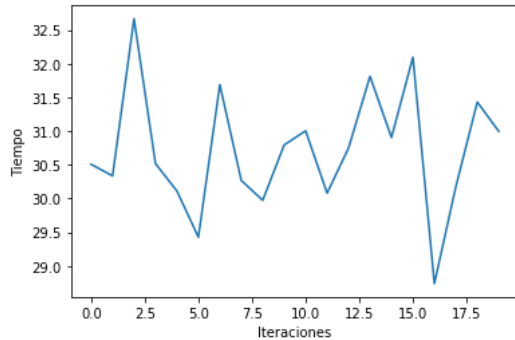
```

1 import time
2 import matplotlib.pyplot as plt
3 import pandas as pd
4 #Inicializamos la clase genetico
5 var = claseGeneticoHibridoGreedy(_cursos=len(totalCursos),_aulas=16,_n=40,_listaCursos=totalCursos,_cantLaboratorios=13)
6
7 #Damos el valor para iterar las generaciones
8 generaciones = 20
9
10 #Generamos la primera poblacion
11
12 # Inicializamos variable para actualizar el ultimo valor
13 ultimovalor = 0
14
15 tiempos2 = []
16 mejorValorHibrido=[]
17 #Iteramos 20 veces
18 for i in range(20):
19     start_time = time.time() #Tomamos el tiempo antes de ejecutar el programa
20
21     var.generarPoblacion()
22     aux=1
23     #-----
24     #Empezamos con las iteraciones
25     while aux<generaciones:
26         #Entrenamos nuestra poblacion con cada iteracion
27         var.entrenar()
28         if(var.puntuacion[-1]==1.0):
29             break
30
31         aux+=1
32     end_time = time.time()
33     tiempos2.append(end_time - start_time) #Tomamos el tiempo después de ejecutar y lo agregamos a la lista
34     mejorValorHibrido.append(max(var.puntuacion))
35 #Creamos un DataFrame con los tiempos de ejecución
36 df2 = pd.DataFrame({"Iteración": range(1,21), 'Tiempo de ejecución': tiempos2})
37 #Mostramos la tabla
38 print(df2)
39
40 # GRAFICO DE TIEMPOS
41 m = [tiempos2[i] for i in range(0,20)]
42 print("Grafico de Tiempos para 20 iteraciones del programa")
43 plt.plot(m)
44 plt.xlabel('Iteraciones')
45 plt.ylabel('Tiempo')
46 plt.show()
47
48 # IMPRIMIR PROMEDIO DE TIEMPOS
49 print("El promedio de tiempo de las iteraciones es: " + str(df2['Tiempo de ejecución'].mean()))
50
51 # GUARDAMOS EL ULTIMO VALOR
52 ultimovalor = var.puntuacion[-1]
53
54 #-----
55 mejorValorHibrido.sort()
56 # Ver datos
57 # ultimo valor
58 print("Ultimo valor: " + str(ultimovalor))
59
60 y=var.obtenerPuntuacion()
61 x=[i for i in range(1,len(y)+1)]
62 print("---- Generado correctamente -----")
63 plt.plot(x,y)
64 plt.show()

```

	Iteración	Tiempo de ejecución
0	1	30.507057
1	2	30.334209
2	3	32.664119
3	4	30.516906
4	5	30.110153
5	6	29.425671
6	7	31.688082
7	8	30.266505
8	9	29.972996
9	10	30.793597
10	11	31.002743
11	12	30.078836
12	13	30.747841
13	14	31.810684
14	15	30.903791
15	16	32.094300
16	17	28.738332
17	18	30.174390
18	19	31.429231
19	20	30.994895

Grafico de Tiempos para 20 iteraciones del programa



El promedio de tiempo de las iteraciones es: 30.712716829776763

Ultimo valor: 0.0002118644067796611

---- Generado correctamente -----



```

1 import pandas as pd
2 import matplotlib.pyplot as plt
3
4
5 df_frames = pd.DataFrame({"Iteración": range(1,21), 'Tiempo de ejecución GENETICO': tiempos1,'Tiempo de
6
7 print(df_frames)
8
9 # Toma las dos columnas X1 y X2 como valores para el eje x
10 x = df_frames["Iteración"]
11
12 # Toma la columna Y como valores para el eje y
13 y = df_frames[['Tiempo de ejecución GENETICO', 'Tiempo de ejecución GREEDY'] ]
14
15 # Crea el gráfico de líneas
16 plt.plot(x, y, 'b-')
17 plt.xlabel('Tiempo Genetico y Tiempo greedy')
18 plt.ylabel('Iteraciones')
19 plt.legend()
20 plt.show()
21
22
23
24 # MUESTRA DE TABLAS COMPARATIVAS SEGUN AL TIEMPO
25 asistencia = ['Algoritmos']
26 print("Tiempo promedio algorimot genetico: ",df_frames['Tiempo de ejecución GENETICO'].mean())
27 print("Tiempo promedio algorimot greedy: ",df_frames['Tiempo de ejecución GREEDY'].mean())
28 men_means = [round(df_frames['Tiempo de ejecución GENETICO'].mean(),3)]
29 women_means = [round(df_frames['Tiempo de ejecución GREEDY'].mean(),3)]
30
31 #Obtenemos la posicion de cada etiqueta en el eje de X
32 x = np.arange(len(asistencia))
33 #tamaño de cada barra
34 width = 0.35
35
36 fig, ax = plt.subplots()
37
38 #Generamos las barras para el conjunto de hombres
39 rects1 = ax.bar(x - width/2, men_means, width, label='AG')
40 #Generamos las barras para el conjunto de mujeres
41 rects2 = ax.bar(x + width/2, women_means, width, label='AGHG')
42
43 #Añadimos las etiquetas de identificacion de valores en el grafico

```

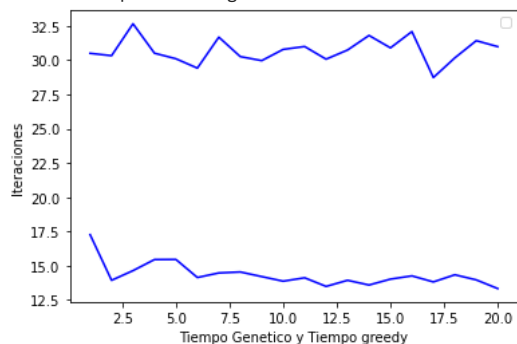
```

44 ax.set_ylabel('TIEMPO')
45 ax.set_title('ALGORITMO GENETICO Y GREEDY')
46 ax.set_xticks(x)
47 ax.set_xticklabels(asistencia)
48 #Añadimos un legen() esto permite mmostrar con colores a que pertence cada valor.
49 ax.legend()
50
51 def autolabel(rects):
52     """Funcion para agregar una etiqueta con el valor en cada barra"""
53     for rect in rects:
54         height = rect.get_height()
55         ax.annotate('{}' .format(height),
56                     xy=(rect.get_x() + rect.get_width() / 2, height),
57                     xytext=(0, 3), # 3 points vertical offset
58                     textcoords="offset points",
59                     ha='center', va='bottom')
60
61 #Añadimos las etiquetas para cada barra
62 autolabel(rects1)
63 autolabel(rects2)
64 fig.tight_layout()
65 plt.savefig('comparacionTiempoEjecucion.png')
66 #Mostramos la grafica con el metodo show()
67 plt.show()
68
69

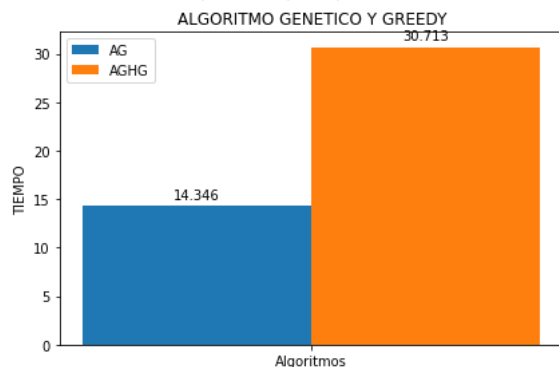
```

	Iteración	Tiempo de ejecución GENETICO	Tiempo de ejecución GREEDY
0	1	17.269562	30.507057
1	2	13.938836	30.334209
2	3	14.644900	32.664119
3	4	15.453417	30.516906
4	5	15.460603	30.110153
5	6	14.142626	29.425671
6	7	14.475558	31.688082
7	8	14.546813	30.266505
8	9	14.208892	29.972996
9	10	13.874411	30.793597
10	11	14.122252	31.002743
11	12	13.492953	30.078836
12	13	13.938710	30.747841
13	14	13.594846	31.810684
14	15	14.025112	30.903791
15	16	14.261368	32.094300
16	17	13.821245	28.738332
17	18	14.342440	30.174390
18	19	13.968910	31.429231
19	20	13.334642	30.994895

WARNING:matplotlib.legend.No handles with labels found to put in legend.



Tiempo promedio algoritmo genetico: 14.345904874801636  
 Tiempo promedio algoritmo greedy: 30.712716829776763



```

1 mejorValorv1=[round(valor*1000,3) for valor in mejorValorGenetico[:5]]
2 mejorValorv2=[round(valor*1000,3) for valor in mejorValorHibrido[:5]]

```

```

3 asistencia = range(1,6)
4 men_means = mejorValorv1
5 women_means = mejorValorv2
6
7 #Obtenemos la posicion de cada etiqueta en el eje de X
8 x = np.arange(len(asistencia))
9 #tamaño de cada barra
10 width = 0.35
11
12 fig, ax = plt.subplots()
13
14 #Generamos las barras para el conjunto de hombres
15 rects1 = ax.bar(x - width/2, men_means, width, label='AG')
16 #Generamos las barras para el conjunto de mujeres
17 rects2 = ax.bar(x + width/2, women_means, width, label='AGHG')
18
19 #Añadimos las etiquetas de identificacion de valores en el grafico
20 ax.set_ylabel('Puntuacion')
21 ax.set_title('Comparacion de evolucion en cada generacion')
22 ax.set_xticks(x)
23 ax.set_xticklabels(asistencia)
24 #Añadimos un legen() esto permite mmostrar con colores a que pertence cada valor.
25 ax.legend()
26
27 def autolabel(rects):
28     """Funcion para agregar una etiqueta con el valor en cada barra"""
29     for rect in rects:
30         height = rect.get_height()
31         ax.annotate('{}'.format(height),
32                     xy=(rect.get_x() + rect.get_width() / 2, height),
33                     xytext=(0, 3), # 3 points vertical offset
34                     textcoords="offset points",
35                     ha='center', va='bottom')
36
37 #Añadimos las etiquetas para cada barra
38 autolabel(rects1)
39 autolabel(rects2)
40 fig.tight_layout()
41 plt.savefig('comparacionEvolucion.png')
42 #Mostramos la grafica con el metodo show()
43 plt.show()

```

