Universidad Interamericana de Puerto Rico

Recinto de Bayamón

Escuela de Ingeniería
Departamento de Ingeniería Eléctrica y de Computadoras

COEN 2310 – Matemáticas Discretas para Ingeniería de Computadoras

Proyecto Final

Nombre de estudiante	:	Mireya Vázquez Robles
Nombre de proyecto	:	Algoritmo basado en grafos
Porcentaje de tareas completadas (0 - 100 %)	:	100%
Fecha	:	5/10/2022

1) Introducción

Breve descripción del Proyecto y mostrar el resumen en la siguiente tabla:

Este proyecto consiste en la implementación de un algoritmo que transforma una matriz de distancias en un grafo conectado. El algoritmo esta basado en la noción de vértices cercano a un grupo, donde el vértice “x” está cerca del grupo C si $d(x,C) < \theta$, (θ = umbral). El grafo es transformado agregando o eliminando aristas iterativamente si la distancia entre los vértices es menor que el umbral.

Resumen	**Porcentaje (0-100%)**
Informe tiene todos los puntos completes – Indique porciento	100%
¿Programa corre al 100% con todas las tareas? – indique porciento	100%

2) Programa

a) Explique cada sección del programa (muestre imágenes en pantalla de parte del código explicando cómo funciona).

main.py

```
from program import *

menu = {
    1:'manual',
    2:'automático',
    3:'salir'
}

menu_datos = {
    1:'file_1.csv',
    2:'file_2.csv',
    3:'file_3.csv'
}

menu_umbral = {
    1:'promedio',
    2:'aleatorio',
    3:'máximo'
}

def print_menu():
    print('\nMenu:')
    for key in menu.keys():
        print (key, '--', menu[key])

def print_menu_datos():
    print('\nSubmenu:')
    for key in menu_datos.keys():
        print (key, '--', menu_datos[key])

def print_menu_umbral():
    print('\nSubmenu:')
    for key in menu_umbral.keys():
        print (key, '--', menu_umbral[key])

```

Fig. 1. Primera sección del programa

En la primera secuencia de comandos main.py se encuentran los dictionarios que contienen las selecciones del menú. Luego están las funciones print_menu(), print_menu_datos() y print_menu_umbral() que imprimen estos menús al ser llamados.

```

#Seleccionar y actuar
flag = True

while flag == True:

    print_menu()
    opcion = int(input('Seleccione una opción: '))
```

Fig. 2. Segunda sección del programa

Luego comienza un while loop, que correrá mientras el flag sea equivalente a True. Se llama la función para imprimir el menú y se le solicita al usuario que seleccione una opción y se guarda en la variable opcion.

```
52      #Manual
53      if opcion == 1:
54
55          #selección de datos
56          print_menu_datos()
57          opcion_datos = int(input('Seleccione una opción: '))
58          datos = menu_datos[opcion_datos]
59          G = pd.read_csv(datos,header=None)
60
61          #selección de umbral
62          print_menu_umbral()
63          opcion_umbral = int(input('Seleccione una opción: '))
64
65          if opcion_umbral == 1:
66              th = promedio_umbral(G)
67
68          elif opcion_umbral == 2:
69              th = aleatorio_umbral(G)
70
71          elif opcion_umbral == 3:
72              th = max_umbral(G)
73
74          else:
75              print('Opción inválida. Reinicie.')
76              continue
77
78          P = algoritmo(G,th)
```

Fig. 3. Tercera sección del programa

Si el usuario elige la opción 1 (manual), se llama la función para imprimir el menú de datos y se le solicita que seleccione uno de los tres archivos y este se lee y se guarda en G. Luego se llama la función para imprimir el menú de umbral y se le solicita al usuario que seleccione una opción. Si el usuario elige la opción 1, se llama la función de promedio_umbral() con G como input y se retorna th. Si el usuario elige la opción 2, se llama la función de aleatorio_umbral() con G como input y se retorna th. Si el usuario elige la opción 3, se llama la función de max_umbral() con G como input y se retorna th. Si ingresa algún otro número se reiniciará, ya que es invalido. Luego se llama la función algoritmo() con G y th como input y se retorna P.

```
80      #Automático
81      elif opcion == 2:
82
83          for key in menu_datos.keys():
84              print("\n",menu_datos[key])
85              G = pd.read_csv(menu_datos[key],header=None)
86              th = promedio_umbral(G)
87              P = algoritmo(G,th)
```

Fig. 4. Cuarta sección del programa

Si el usuario elige la opción 2 (automático), se inicia un for loop en donde se lee y se guarda cada archivo en G, se llama promedio_umbral() con G como input, se retorna th, y luego se llama algoritmo() con G y th como input y se retorna P para cada uno de los archivos.

```
89      #Salir
90      elif opcion == 3:
91          print('\nSaliendo...\n')
92          flag = False
93
94      else:
95          print('Opción inválida. Ingrese un número entre 1 y 3.')
```

Fig. 5. Quinta sección del programa

Si el usuario elige la opción 3 (salir), el flag cambia a false, lo cual termina el while loop y luego sale del programa. Si el usuario ingresa cualquier otro número se imprime un mensaje de error.

program.py

```python
import pandas as pd
import random


def promedio_umbral(G):
    """Obtiene el valor promedio de las distancias del grafo G."""

    number_list =[]
    for i in range(len(G)):
        for j in range(len(G)):
            number = G[i][j]
            number_list.append(number)

    th = sum(number_list) / len(number_list)

    return th


def aleatorio_umbral(G):
    """Obtiene un valor de distancia aleatoriamente del grafo G."""

    r1 = (random.randint(0,len(G)-1))
    r2 = (random.randint(0,len(G)-1))
    th = G[r1][r2]

    return th


def max_umbral(G):
    """Obtiene el valor máximo de las distancias del grafo G."""
    th = 0
    for i in range(len(G)):
        for j in range(len(G)):
            number = G[i][j]
            if number > th:
                th = number

    return th

```

Fig. 6. Sexta sección del programa

En la segunda secuencia de comandos program.py están las funciones que serán llamadas. La función promedio_umbral() obtiene el valor promedio de las distancias del grafo G. Guarda cada numero de la matriz en una lista, luego los suma y divide entre el largo de la lista para obtener el promedio y lo guarda en th. La función aleatorio_umbral() obtiene un valor de distancia aleatoriamente del grafo G. Primero se obtienen dos números aleatorios entre 0 y la longitud de G menos 1, y luego se utilizan estos dos valores como índex para acceder a una distancia de la matriz y guardarla en th. La función max_umbral() obtiene el valor máximo de las distancias del grafo G. Se utilizan nested for loops para acceder a cada número de la matriz de distancia y se compara con el numero guardado en th, si es mayor, lo sustituye. Cada una de estas funciones reciben G como input y retornan th.

```python
def find_max_degree(S,G,th):
    """Halla la matriz de adyacencia y el vertice con el mayor grado en C."""
    adyacencia = []

    for i in S:
        A = []
        for j in S:
            d = G[i][j]
            if d < th and d != 0:
                a = 1
            else:
                a = 0
            A.append(a)
        adyacencia.append(A)
    adyacencia = pd.DataFrame(adyacencia)

    column_names=list(G.columns)
    adyacencia.columns = column_names
    adyacencia.index = column_names

    degree = adyacencia[adyacencia == 1].count()

    degree_list = [degree[r] for r in column_names]
    v_list = [m for m in column_names]

    max_degree = 0
    for number in degree_list:

        if number > max_degree:
            max_degree = number

    v = v_list[degree_list.index(max_degree)]

    if max_degree == 0:
        v = column_names[0]

    return v
```

Fig. 7. Séptima sección del programa

La función find_max_degree() halla la matriz de adyacencia y el vértice con el mayor grado en C utilizando nested for loops e if/else statements. Recibe S,G y th como input y retorna v. Guarda cada valor de distancia en d y si esta distancia es menor a th y no 0, le asigna 1 a la variable a, si no le asigna 0. Guarda a en una lista llamada A y luego guarda la lista A en adyacencia y luego se convierte en DataFrame creando así la matriz de adyacencia. Para asegurarse que los nombres de las columnas y las listas sean igual a las de G se utilizan las funciones DataFrame.columns() y DataFrame.index(). Luego se cuentan los 1 en cada fila y se guarda en degree. Se crea una lista que contiene todos los degrees y otra lista que contiene todos los vértices. De esta lista se obtiene el máximo utilizando un for loop y evaluando si el número es mayor al previo. Luego se guarda el vértice en v utilizando el degree máximo obtenido para localizarlo en la lista. Si el max_degree es 0, v será el primer vértice.

```
86  def cercano(th,C,S,G):
87      """Encuentra el vertice x más cercano al grupo C. """
88      all_D = []
89      min_D = []
90
91      for j in S:
92          sum_D = []
93          D = []
94          for i in C:
95              distancia = G.loc[i,j]
96              D.append(distancia)
97          sum_D = sum(D)/len(C)
98          all_D.append(sum_D)
99
100     if len(all_D)!= 0:
101
102         min_D = all_D[0]
103         for number in all_D:
104
105             if number < min_D:
106                 min_D = number
107
108         index_D = all_D.index(min_D)
109
110         if min_D <= th:
111             x = S[index_D]
112             b = 1
113         else:
114             x = S[index_D]
115             b = 0
116     else:
117         x = 0
118         b = 0
119
120     return b,x
121
```

Fig. 8. Octava sección del programa

La función cercano() encuentra el vertice x más cercano al grupo C utilizando nested for loops e if/else statements. Recibe th, C, S y G como input y retorna b y x. Guarda cada distancia entre los vértices en C y los vértices en S en una lista. Luego suma los valores, los divide entre el largo de C y los añade a otra lista llamada all_D. Si esta lista no está vacía, obtiene el valor mínimo de la lista utilizando un for loop e if statement y lo guarda en min_D y en index_D guarda su índex. Si el valor mínimo es menor o igual a th guarda el vértice en esa posición en x y le asigna 1 a b. Si no, le asigna 0 a b. Si la lista all_D está vacía asigna 0 a x y a b.

```
122
123     def algoritmo(G,th):
124         """Transforma una matriz de distancias en un grafo conectado."""
125         S = list(range(len(G)))
126         P = []
127         t = 0
128
129         while len(S) != 0 and t < 20:
130             C = []
131             v = find_max_degree(S,G,th)
132             C.append(v)
133             S.remove(v)
134             b,x = cercano(th,C,S,G)
135
136             while b == 1:
137                 C.append(x)
138                 S.remove(x)
139                 b,x = cercano(th,C,S,G)
140
141             P.append(C)
142             for node in C:
143                 G = G.drop(index=[node,node],columns=[node,node])
144             t += 1
145
146         print(P)
147
148         return P
```

Fig. 9. Novena sección del programa

En la función algoritmo() se transforma la matriz de distancias en un grafo conectado. Recibe como input G y th y retorna P. S contiene los vértices en G y P es una lista vacía que contendrá el resultado. Aquí se inicia un while loop que correrá mientras S no esté vacío. C es una lista vacía que contendrá los vértices conectados. Se llama la función find_max_degree(), de donde se obtiene v. Se guarda v en C y se elimina de S. Luego se llama la función cercano(), de donde se obtiene b y x. Se inicia otro while loop que correrá mientras b sea equivalente a 1, es decir, mientras exista un vértice cercano x no en C. Se añade el vértice x a C y se elimina de S y se corre nuevamente la función cercano(). Una vez termina este segundo while loop, añade C a P y elimina de la matriz de distancia G los vértices que ahora están en C. Una vez termina el primer while loop, imprime y retorna el resultado final P.

```
125     if __name__ == "__main__":
126         G = pd.read_csv('file_1.csv',header=None)
127         th = promedio_umbral(G)
128         algoritmo(G,th)
```

Fig. 10. Décima sección del programa

Al final de la secuencia de comandos program.py hay un if statement que verifica si está en __main__, si lo está lee y guarda en G los datos del file_1.csv, obtiene el th de la función medio_umbral y corre la función algoritmo. Esto está aquí para poder realizar pruebas sin tener que hacer selecciones en el menú.

b) Usando una tabla, enumere todas las funciones creadas. Las columnas de la tabla serán: nombre, descripción, entrada, salida de la función.

Tabla 1: Funciones creadas

nombre	descripción	entrada	salida
print_menu()	imprime el menú		imprime el menú
print_menu_datos()	imprime el submenú de datos		imprime el submenú de datos
print_menu_umbral()	imprime el submenú de umbral		imprime el submenú de umbral
promedio_umbral()	Obtiene el valor promedio de las distancias del grafo G.	G	retorna th
aleatorio_umbral()	Obtiene un valor de distancia aleatoriamente del grafo G.	G	retorna th
max_umbral()	Obtiene el valor máximo de las distancias del grafo G.	G	retorna th
find_max_degree()	Halla la matriz de adyacencia y el vértice con el mayor grado en C.	S,G,th	retorna v
cercano()	Encuentra el vértice x más cercano al grupo C.	th,C,S,G	retorna b,x
algoritmo()	Transforma una matriz de distancias en un grafo conectado.	G,th	retorna e imprime P

3) Resultados

Mostrar los resultados para cada conjunto de datos:

Tabla-2: Resultados Datos_1

Umbral Th	Salida P	¿Grafo conectado? (si/no)
Promedio	[[0, 1], [2, 3], [4, 5]]	No (grafo desconectado con 3 componentes, cada componente si es un subgrafo conectado)

Aleatorio	[[4, 5, 2, 3, 0, 1]]	Si
Máximo	[[4, 5, 2, 3, 0, 1]]	Si

Tabla-3: Resultados Datos_2

Umbral Th	**Salida P**	**¿Grafo conectado? (si/no)**
Promedio	[[2, 4, 14, 8, 11, 5, 12], [15, 17, 19, 18, 16, 20, 0, 1], [7, 13], [3, 6], [9], [10]]	No (grafo desconectado con 6 componentes, 4 de los componentes si son subgrafos conectados, mientras que los otros 2 son vértices aislados)
Aleatorio	[[1, 0, 16, 17, 15, 19, 18, 20, 2, 4, 8, 5, 14, 13, 9, 11, 7, 12, 10, 6, 3]]	Si
Máximo	[[1, 0, 16, 17, 15, 19, 18, 20, 2, 4, 8, 5, 14, 13, 9, 11, 7, 12, 10, 6, 3]]	Si

Tabla-4: Resultados Datos_3

Umbral Th	**Salida P**	**¿Grafo conectado? (si/no)**
Promedio	[[6, 2, 26, 12, 9, 25, 23, 5, 28, 29, 11, 7, 27, 15, 16, 19, 8, 0], [3, 14, 18, 10, 21, 1, 4, 24], [20, 13, 22, 17]]	No(grafo desconectado con 3 componentes, cada componente si es un subgrafo conectado)
Aleatorio	[[2, 6, 26, 12, 9, 25, 23, 5, 28, 29, 11, 7, 27, 15, 16, 19, 8], [14, 3, 18, 10, 21, 1], [13, 20, 22], [0, 17], [4, 24]]	No (grafo desconectado con 5 componentes, cada componente si es un subgrafo conectado)
Máximo	[[0, 8, 23, 28, 29, 9, 25, 5, 11, 12, 6, 26, 7, 2, 27, 15, 19, 16, 17, 20, 13, 22, 21, 1, 10, 18, 14, 3, 4, 24]]	Si

a) Analice las tablas de los tres conjuntos de datos e indique cual es el mejor resultado y porque cree que es el mejor.

   El mejor resultado en todos los casos es el que se obtiene tras utilizar el valor máximo para el umbral, ya que el resultado que se obtiene es un grafo conectado en todos los conjuntos de datos. Cuando el valor del umbral es el valor máximo, significa que todos los vértices se consideran cercanos al grupo C ya que, en base a la noción de vértices cercanos a un grupo, el vértice “x” está cerca del grupo C si $d(x,C) < \theta$, (θ = umbral) .

b) Analice la estrategia usada para seleccionar el umbral.

El umbral es un valor que debe ser obtenido a partir de la matriz de distancias G. Para este proyecto se obtuvo el umbral utilizando tres estrategias distintas:

La primera estrategia consistió en obtener el valor promedio de las distancias del grafo G. Se guardó cada numero de la matriz en una lista, luego se sumaron y dividió entre el largo de la lista para obtener el promedio. Al utilizar el promedio, la distancia entre los vértices y un grupo puede ser mayor o menor que el umbral así que resultará en un grafo desconectado con varios componentes.

La segunda estrategia fue obtener un valor de distancia aleatoriamente del grafo G. Primero se obtienen dos números aleatorios entre 0 y la longitud de G menos 1, y luego se utilizan estos dos valores como índex para acceder a una distancia de la matriz y guardarla en th. Al utilizar un valor aleatorio, cada vez que se corre el programa se obtendrá un resultado distinto. Al utilizar este método se pueden obtener grafos conectados o desconectados.

La tercera estrategia fue obtener el valor máximo de las distancias del grafo G. Se utilizaron nested for loops para acceder a cada número de la matriz de distancia y compararlo con el numero guardado en th, si es mayor, lo sustituye. Al utilizar el valor máximo se obtiene un grafo conectado debido a que todas las distancias se considerarán menor o igual al umbral lo que significa que todos los vértices se consideran cercanos al grupo C.

4) Complejidad de tiempo

a) Realice un análisis del tiempo que tarda su código para el peor caso. Exprese el resultado en notación “Big O”.

Resultado para el peor caso:

$$O(l(m(2n^2 + 2p^2 + 4pq + 6p + q + 4n + 2r + s + 38) + 2) + 4)$$

Simplificado:

$$O(l(m(n^2 + p^2 + pq + p + q + n + r + s + 1) + 1) + 1)$$

Simplificado:

$$O(lmn^2 + lmp^2 + lmpq + lmp + lmq + lmn + lmr + lms + lm + l + 1)$$

Esto es cierto asumiendo que *l,m,n,p,q,r,s* son de distintos tamaños. Esto ocurre debido a que el programa tiene nested while y for loops que recorren distinto número de veces. Otra forma de expresar el resultado sería:

$$O(\max(lmn^2, lmp^2, lmpq, lmp, lmq, lmn, lmr, lms))$$

Se puede expresar de esta manera porque el tiempo que tarde en recorrer el programa dependerá de si una de estas variables domina sobre las otras. Por ejemplo, si tenemos un algoritmo para el cual el resultado es O(M+N) = O(max(M,N)) si M>N, entonces su complejidad de tiempo sería O(M), de lo contrario sería O(N).

b) ¿Es aceptable computacionalmente su resultado? Justifique su respuesta.

El resultado es computacionalmente aceptable debido a que es en tiempo polinomial. El campo de la teoría de la complejidad computacional se enfoca en clasificar los problemas computacionales de acuerdo con su uso de recursos y en relacionar estas clases entre sí. Un problema para el cual existe un algoritmo en tiempo polinomial es considerado como perteneciente a la clase P (clase de problemas computacionales que son eficientemente solucionables). La tesis de Cobham, también conocida como tesis de Cobham-Edmonds, es la que establece que el tiempo polinomial es sinónimo de "factible", "eficiente" o "rápido". Sin embargo, que sea computacionalmente aceptable, no significa que sea práctico ya que esta tesis ignora factores como el tamaño del exponente y el tamaño del input.

5) Conclusión

a) Analice su diseño en términos de cómo los resultados cumplieron con sus expectativas.

El programa realiza todas las funciones implementadas sin errores cumpliendo así con las expectativas. Incluso puedo confirmar que el algoritmo funciona con datos(archivos) distintos a los asignados, ya que cree otro archivo .csv con una tabla como el ejemplo explicado en clase para realizar pruebas. Además, logre completar el código en menos tiempo de lo esperado. Logré dividir todas las partes en funciones, logré que fuera más fácil de leer al utilizar mejores nombres y logré evitar la repetición al utilizar for loops.

b) Describa todos los problemas principales que encontró y como fue solucionado.

El problema principal que encontré fue al realizar el análisis del tiempo que tarda el código para el peor caso. La mayoría de los ejemplos son bastantes sencillos y muy diferentes al programa por lo que al intentar analizarlo me resulta difícil ya que contiene tantas variables y tantos for loops, while loops e if statements. Este problema lo solucioné haciendo búsquedas en internet de ejemplos, revisando nuevamente el material y leyendo más información con respecto al tema. También copié el código a una aplicación similar a OneNote en la cual pude realizar apuntes sobre el código en manuscrito para poder analizarlo mejor. Otro problema que encontré fue el tener que realizar cálculos matemáticos sin el uso de librerías como para obtener el promedio y el máximo. Lo solucioné realizando los cálculos paso a paso y utilizando for loops e if/else statements para obtener el resultado.

c) Cosas que probaste y que no funcionaron.

Probé utiliza el valor mínimo de la matriz de distancia G para el umbral, pero rápidamente me di cuenta de que si utilizaba el mínimo todos los vértices se considerarían como distantes y no resultaría en un grafo conectado. También probé utilizar el valor medio de la matriz de distancia G para el umbral, pero resultó en grafos desconectados con varios componentes.

d) Haz una lista de todas las cosas que aprendiste.
- Cómo hallar la matriz de adyacencia
- Cómo hallar vertice con el mayor grado en C
- Cómo encontrar el vertice x más cercano al grupo C
- Cómo transformar una matriz de distancias en grafo conectado
- Cómo el valor del umbral afecta el resultado
- Cómo darle mejor uso a los for loops para evitar la repetición
- Cómo igualar los nombres de las filas y columnas de dos dataframes.
- Cómo remover filas y columnas de un dataframe.
- Cómo aplicar if __name__ == “__main__”
- Más con respecto al tema de Big O, complejidad de tiempo, y complejidad computacional.

6) Apéndices

Adjunte aquí todo el código fuente.

 # main.py

```
from program import *

menu = {
    1:'manual',
    2:'automático',
    3:'salir'
}

menu_datos = {
    1:'file_1.csv',
    2:'file_2.csv',
    3:'file_3.csv'
}

menu_umbral = {
    1:'promedio',
    2:'aleatorio',
    3:'máximo'
}

def print_menu():
    print('\nMenu:')
    for key in menu.keys():
        print (key, '--', menu[key])

def print_menu_datos():
    print('\nSubmenu:')
    for key in menu_datos.keys():
```

```
        print (key, '--', menu_datos[key])

def print_menu_umbral():
    print('\nSubmenu:')
    for key in menu_umbral.keys():
        print (key, '--', menu_umbral[key])



#Seleccionar y actuar
flag = True

while flag == True:

    print_menu()
    opcion = int(input('Seleccione una opción: '))

    #Manual
    if opcion == 1:

        #selección de datos
        print_menu_datos()
        opcion_datos = int(input('Seleccione una opción: '))
        datos = menu_datos[opcion_datos]
        G = pd.read_csv(datos,header=None)

        #selección de umbral
        print_menu_umbral()
        opcion_umbral = int(input('Seleccione una opción: '))
```

```
    if opcion_umbral == 1:
        th = promedio_umbral(G)

    elif opcion_umbral == 2:
        th = aleatorio_umbral(G)

    elif opcion_umbral == 3:
        th = max_umbral(G)

    else:
        print('Opción inválida. Reinicie.')
        continue

    P = algoritmo(G,th)

#Automático
elif opcion == 2:

    for key in menu_datos.keys():
        print("\n",menu_datos[key])
        G = pd.read_csv(menu_datos[key],header=None)
        th = promedio_umbral(G)
        P = algoritmo(G,th)

#Salir
elif opcion == 3:
    print('\nSaliendo...\n')
```

```
        flag = False

    else:
        print('Opción inválida. Ingrese un número entre 1 y 3.')
```

#program.py

```
import pandas as pd
import random


def promedio_umbral(G):
    """Obtiene el valor promedio de las distancias del grafo G."""

    number_list =[]
    for i in range(len(G)):
        for j in range(len(G)):
            number = G[i][j]
            number_list.append(number)

    th = sum(number_list) / len(number_list)

    return th


def aleatorio_umbral(G):
    """Obtiene un valor de distancia aleatoriamente del grafo G."""
```

```
    r1 = (random.randint(0,len(G)-1))
    r2 = (random.randint(0,len(G)-1))
    th = G[r1][r2]

    return th


def max_umbral(G):
    """Obtiene el valor máximo de las distancias del grafo G."""
    th = 0
    for i in range(len(G)):
        for j in range(len(G)):
            number = G[i][j]
            if number > th:
                th = number

    return th


def find_max_degree(S,G,th):
    """Halla la matriz de adyacencia y el vertice con el mayor grado en C."""
    adyacencia = []

    for i in S:
        A = []
        for j in S:
            d = G[i][j]
            if d < th and d != 0:
```

```
            a = 1
        else:
            a = 0
        A.append(a)
    adyacencia.append(A)
adyacencia = pd.DataFrame(adyacencia)

column_names=list(G.columns)
adyacencia.columns = column_names
adyacencia.index = column_names

degree = adyacencia[adyacencia == 1].count()

degree_list = [degree[r] for r in column_names]
v_list = [m for m in column_names]

max_degree = 0
for number in degree_list:

    if number > max_degree:
        max_degree = number

v = v_list[degree_list.index(max_degree)]

if max_degree == 0:
    v = column_names[0]

return v
```

```
def cercano(th,C,S,G):
    """Encuentra el vertice x más cercano al grupo C. """
    all_D = []
    min_D = []

    for j in S:
        sum_D = []
        D = []
        for i in C:
            distancia = G.loc[i,j]
            D.append(distancia)
        sum_D = sum(D)/len(C)
        all_D.append(sum_D)

    if len(all_D)!= 0:

        min_D = all_D[0]
        for number in all_D:

            if number < min_D:
                min_D = number

        index_D = all_D.index(min_D)

        if min_D <= th:
            x = S[index_D]
```

```
            b = 1
        else:
            x = S[index_D]
            b = 0
    else:
        x = 0
        b = 0

    return b,x


def algoritmo(G,th):
    """Transforma una matriz de distancias en un grafo conectado."""
    S = list(range(len(G)))
    P = []
    t = 0

    while len(S) != 0 and t < 20:
        C = []
        v = find_max_degree(S,G,th)
        C.append(v)
        S.remove(v)
        b,x = cercano(th,C,S,G)

        while b == 1:
            C.append(x)
            S.remove(x)
            b,x = cercano(th,C,S,G)
```

```
        P.append(C)
        for node in C:
            G = G.drop(index=[node,node],columns=[node,node])
        t += 1


    print(P)


    return P


if __name__ == "__main__":
    G = pd.read_csv('file_1.csv',header=None)
    th = promedio_umbral(G)
    algoritmo(G,th)
```

7) Referencias

“Cobham's Thesis,” *Wikipedia*, 09-Feb-2022. [Online]. Available: https://en.wikipedia.org/wiki/Cobham%27s_thesis. [Accessed: 09-May-2022].

“Computational complexity theory,” *Wikipedia*, 07-Apr-2022. [Online]. Available: https://en.wikipedia.org/wiki/Computational_complexity_theory. [Accessed: 09-May-2022].