CENTRO DE CIENCIAS BÁSICAS

DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

METAHEURÍSTICAS I

7° "A"

EVALUACIÓN FINAL

Profesor: Francisco Javier Luna Rosas

Alumnos:

Almeida Ortega Andrea Melissa

Espinoza Sánchez Joel Alejandro

Flores Fernández Óscar Alonso

Gómez Garza Dariana

González Arenas Fernando Francisco

Orocio García Hiram Efraín

Fecha de Entrega: Aguascalientes, Ags.**, 25** de noviembre de 2021

Evaluación Final

El equipo decidió elegir una implementación y explicación del algoritmo ACO (Ant Colony Optimization) por lo que se procederá a explicar lo referente al algoritmo.

a) ¿Qué es ACO? ¿Dónde se aplica ACO? ¿Qué características tiene ACO?

La definición de ACO es la siguiente:

> El ACO (Ant Colony Optimization) es un algoritmo que parte principalmente del estudio contemporáneo social de las hormigas, realizado principalmente por mirmecólogos, que son especialistas en investigación del comportamiento de las hormigas, entre ellos, destacan Pierre-Paul Grassé y Edward O. Wilson.

La aplicación general de ACO es la siguiente:

> La colonia de hormigas es una metaheurística destinada originalmente a problemas de optimización combinatoria. Esta técnica se basa en el comportamiento estructurado de las colonias de hormigas, donde individuos muy simples se comunican entre sí por medio de una sustancia química llamada feromona, estableciendo el camino más adecuado entre el hormiguero y su fuente de alimentos.

Las características de ACO son las siguientes:

> Primeramente, ACO hace uso de feromonas:
>
> > En su recorrido, las hormigas depositan feromonas de modo que su rastro les permite volver a su hormiguero desde la fuente de alimento. Cada vez que la hormiga llega a una intersección de dos o más caminos, decide el camino a seguir de un modo probabilístico.
> >
> > Las hormigas eligen con mayor probabilidad los caminos con mayor rastro de feromonas. Los más prometedores van acumulando más feromonas al ser recorridos por más hormigas.

Otra característica de ACO es la evaporación:

Los caminos menos prometedores pierden feromonas por evaporación al ser visitadas por menos hormigas cada vez.

Aun así, la gran perduración de los rastros hace que la evaporación influya poco en la acción continua de la colonia para dar con un rastro de feromonas que permita a las hormigas encontrar un camino cada vez más corto desde el hormiguero a la colonia.

Otra característica por considerar es el uso de agentes en el proceso:

El algoritmo es realmente un sistema multiagentes, en el que cada agente simple representa una hormiga y las interacciones entre un conjunto de hormigas manifiesta un comportamiento mucho más complejo, correspondiente a toda la colonia de hormigas.

Este algoritmo también necesita de reforzamiento:

La feromona se muestra a través de un registro histórico de las rutas recorridas más o menos promisorias en las búsquedas locales.

En los algoritmos evolutivos, las feromonas se representan mediante reforzamiento (premio o castigo); a través de éste, la hormiga aprende cuál es la mejor opción de las rutas.

b) ¿Qué algoritmo utiliza ACO? ¿Cómo se define la población en ACO? ¿Cómo se inician las velocidades y pesos inicialmente? ¿Cómo se calcula la función de adaptación? ¿Cómo se calcula el Pbest y el Gbest? Etc.

El algoritmo que utiliza ACO es el siguiente compuesto:

El algoritmo primeramente requiere de obtener la instancia del problema, en el caso del problema TSP se trata de matriz de distancia, también se inicializan las feromonas y la matriz de visibilidad.

El ACO tomar como un algoritmo la posición de las hormigas en partes aleatorias del espacio y se arma la lista Tabú, explicada en la definición de población más adelante.

Para construir una solución, el algoritmo utiliza la selección por ruleta según las posibilidades que tenga la hormiga de moverse.

También aplica una operación para actualizar las feromonas y evaporarlas según sea el caso.

La definición de población en ACO es la siguiente:

Una forma de implementar la estructura de población es el uso de una matriz de las siguientes dimensiones:

$$\begin{bmatrix} c_1h_1 & c_2h_1 & \cdots & c_ih_1 \\ c_1h_2 & c_2h_2 & \cdots & c_ih_2 \\ \vdots & \vdots & \ddots & \vdots \\ c_1h_j & c_2h_j & \cdots & c_ih_j \end{bmatrix}$$

Donde c_nh_m significa el nodo en el que se encuentra la hormiga m al realizar el paso n.

La inicialización de la población situaría en toda la primera columna, el nodo en el que las hormigas iniciarían mientras que las demás podrían establecer otro valor diferente a las representaciones de los nodos, denotando que no se ha dado este paso.

La forma en la que se inicializan las estructuras de ACO es la siguiente:

Primero debe inicializarse la matriz de distancia, que es la carga de una matriz con las distancias de realizar un viaje de un nodo a otro.

También debe inicializarse la matriz de feromonas, donde ésta tendrá una representación similar a la matriz de distancias, sin embargo no representará la distancia de un nodo a otro, sino de la intensidad de la feromona de moverse

en ese sentido. De esta manera, la matriz de feromonas tendrá un valor semejante para todos los nodos que estén conectados. Para todos los nodos que no estén conectados simplemente tendrán una asignación de cero.

Otra estructura a inicializar es la matriz de visibilidad, que sólo se calcula con el inverso de los valores de la matriz de distancias.

Una vez que la inicialización se ha realizado, el algoritmo ACO se ejecuta.

El cálculo de probabilidad de que la k-ésima hormiga viaje del nodo i al nodo j en el paso actual es lo primero que debe realizarse en el algoritmo, se realiza como se describe a continuación:

Para ello se usará la siguiente función:

$$p_{ij}^{k} = \frac{\left[\tau_{ij}^{(t)}\right]^{\alpha}\left[\eta_{ij}\right]^{\beta}}{\sum_{j}\left[\tau_{ij}^{(t)}\right]^{\alpha}\left[\eta_{ij}\right]^{\beta}}$$

Donde τ_{ij} representa la feromona de transitar del nodo i al nodo j, η_{ij} denota al valor de la matriz de visibilidad del nodo i al nodo j y α y β son parámetros constantes reales positivos otorgados por el usuario para dar mayor peso a las feromonas o a la visibilidad de la hormiga respectivamente.

Con base en este resultado, se realizará la selección por ruleta según los nodos disponibles y los cálculos realizados para cada posible cruce y al seleccionar un siguiente nodo será agregado a la lista tabú como siguiente paso de la hormiga.

Los cálculos de feromonas se realizan como se explican a continuación:

La actualización de feromona se realiza como se explica a continuación:

Una vez que se tienen todas las soluciones, se evalúan y se selecciona la mejor, actualizando entonces, los rastros de feromonas.

La cantidad de feromona que se deposite en cada arco es proporcional a la distancia del recorrido completo encontrado por cada hormiga, es decir:

$$\Delta\tau_{ij}^{k} = \frac{Q}{L_k}$$

Donde Q es una constante y L_k es la longitud del recorrido completo realizado por la hormiga k.

La evaporación de feromona se realiza como se explica a continuación:

Para evaporar las feromonas, se sustituyen los rastros de feromona en cada tramo del grafo de acuerdo con la siguiente expresión:

$$(1-\rho)\tau_{ij} + \sum \Delta\tau_{ij}$$

Donde ρ es un parámetro constante real ubicado en el intervalo $[0,1]$ para determinar qué tanto se evaporarán las feromonas por iteración.

Luego se ejecuta una nueva iteración con un nuevo grupo de k hormigas hasta alcanzar un criterio de paro.

Se deberá implementar el algoritmo en el lenguaje de preferencia para resolver una aplicación en específico.

El algoritmo se implementará en lenguaje C para resolver la siguiente aplicación de alta importancia:

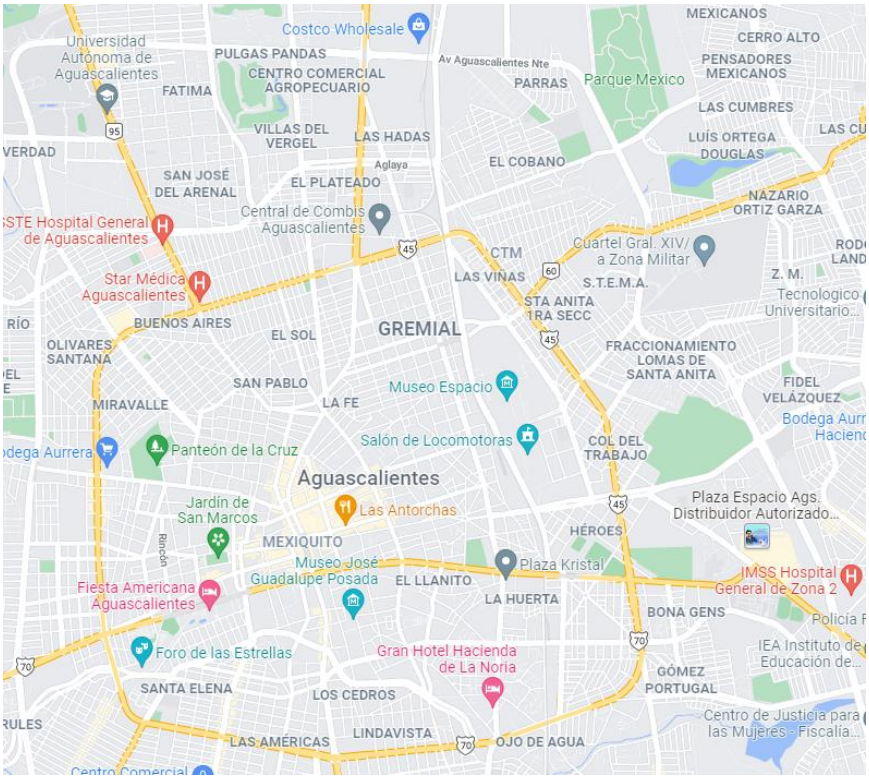
Los alumnos de la Universidad Autónoma de Aguascalientes que estudian en el séptimo semestre la carrera de Ingeniería en Computación Inteligente terminarán el semestre pronto. También, a pocos días de terminar el semestre se estrenará en cines la película *SpiderMan: No Way Home* y debido a los avances que muestran personajes de películas anteriores, específicamente el equipo en cuestión desea

realizar un maratón de películas anteriores que el equipo piensa que son altamente necesarias para tener frescas antes de ver el estreno.

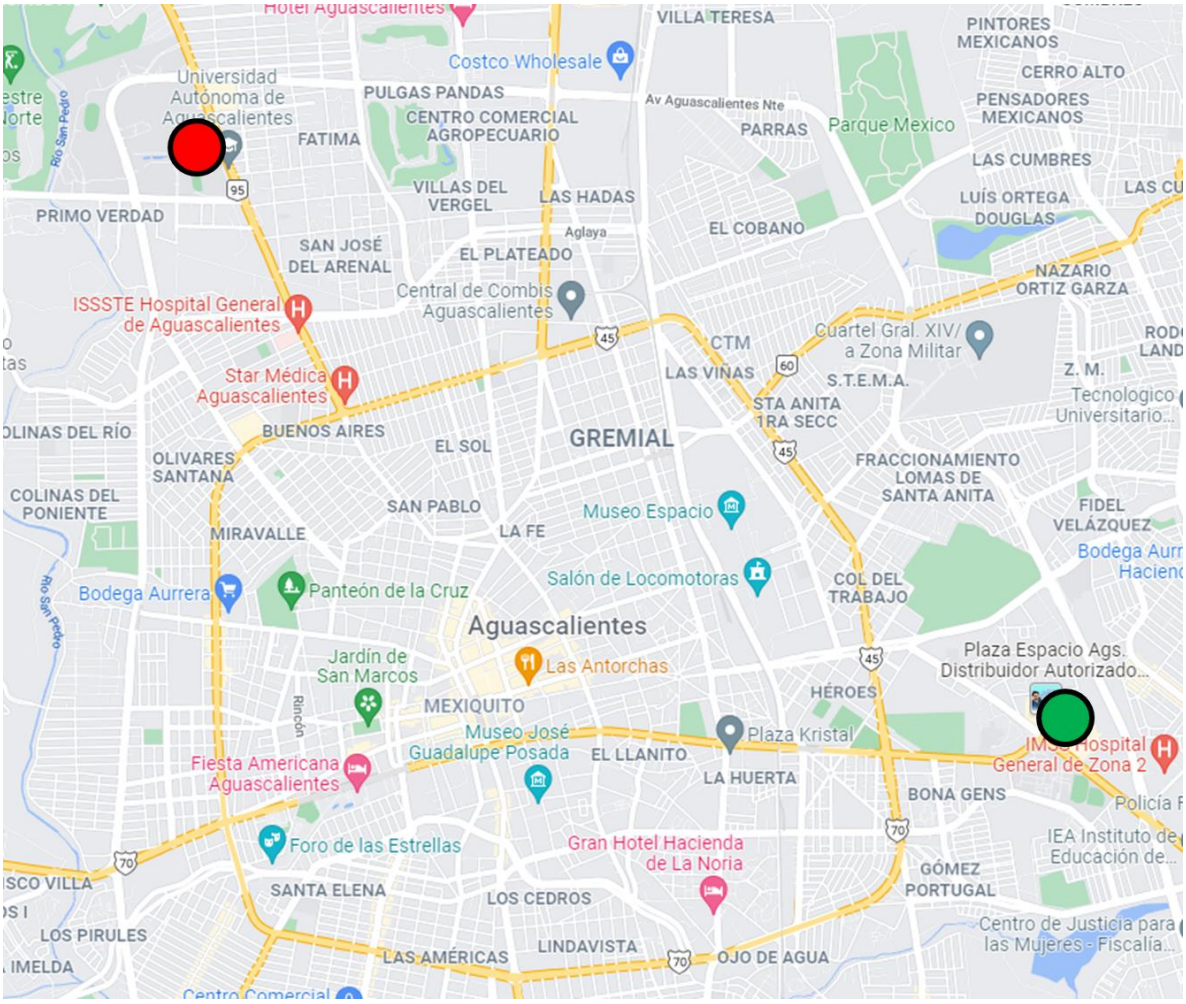
Concretamente, Hiram Efraín Orocio García, miembro del presente equipo dijo que pondría casa para el maratón en cuestión y que sería altamente conveniente que en un día martes que todo el equipo asiste a clases presenciales en la Universidad se realizara la reunión, partiendo desde el plantel de la UAA hasta un punto cercano de la casa del miembro del equipo, sin embargo, Hiram no quiere depositarle más gasolina a su nave que supuestamente porque él va a poner las botanas…

Debido a ello, el equipo pensó que podrían realizar un algoritmo de colonia de hormigas para encontrar la ruta más cercana de la Universidad Autónoma de Aguascalientes al Centro Comercial Espacio, sitio cercano del destino para el equipo.

El equipo planteó algunas de las rutas más importantes que podrían ser posibles soluciones al problema especificado anteriormente tomando en cuenta el mapa de la ciudad de Aguascalientes:



Tomando en cuenta el mapa anterior y la problemática a resolver, los puntos de inicio y final serían los siguientes:



Donde el círculo rojo representa el punto de inicio, que es la Universidad Autónoma de Aguascalientes y el círculo verde el destino, siendo el Centro Comercial Espacio, sitio cercano pero útil para realizar el modelado.

Asimismo, para simplificar el problema, el equipo decidió tomar algunos de los caminos más comunes o los más probables y no modelar todas y cada una de las calles entre estos dos puntos, pues debería de involucrarse toda la ciudad, aspecto que sería muy complicado.

Debido a esto, el equipo modeló algunas rutas que, incluyendo los dos nodos ya mencionados anteriormente, generan un grafo de 15 nodos involucrando las calles

que tienen mayor probabilidad de uso, señalizadas como puede apreciarse a continuación:

Aunque se realizaron muchas simulaciones de cómo podría ser el grafo final de la ciudad de Aguascalientes (véase anexo 1) el anterior fue el definitivo y con el que se realizaron las actividades sucesivas después de la propuesta del modelo, es decir, la implementación en código, análisis de resultados y presentación de los mismos.

Puede observarse que el grafo ya contiene las distancias de nodo a nodo. Los nodos también ya se encuentran etiquetados con letras de la A a la O, siendo – además de la ya especificada Universidad Autónoma de Aguascalientes y el Centro Comercial Espacio – intersecciones de calles o avenidas siguientes:

- **A:** Universidad Autónoma de Aguascalientes.
- **B:** Intersección de Avenida Aguascalientes Norte con Avenida Universidad.
- **C:** Intersección de Avenida de la Convención de 1914 Norte con Avenida Universidad.
- **D:** Intersección de Boulevar a Zacatecas con Avenida Aguascalientes Norte.
- **E:** Intersección de Avenida de la Convención de 1914 Poniente con Avenida Fundición.
- **F:** Intersección de Avenida de la Convención de 1914 Norte con Avenida Héroe de Nacozari.
- **G:** Intersección de Avenida de la Convención de 1914 Poniente con la Carretera Calvillo – Aguascalientes.
- **H:** Intersección de Avenida Héroe de Nacozari con Prolongación Alameda.
- **I:** Intersección de Avenida Héroe de Nacozari con Avenida Adolfo López Mateos Oriente.
- **J:** Intersección de Avenida Héroe de Nacozari con Avenida de la Convención de 1914 Sur.
- **K:** Intersección de Avenida de la Convención de 1914 con Avenida José H. Escobedo.
- **L:** Intersección de Avenida José H. Escobedo con Avenida Aguascalientes Oriente.
- **M:** Intersección de Avenida de la Convención de 1914 con Prolongación Alameda.
- **N:** Intersección de Avenida de la Convención de 1914 con la Carretera San Luis Potosí – Aguascalientes.
- **O:** Centro Comercial Espacio

Asimismo, también puede observarse que al representar mediante una matriz de distancias (en kilómetros) los siguientes nodos, se tiene lo siguiente:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
A	0	0.9	1.9	0	0	0	0	0	0	0	0	0	0	0	0
B	0.9	0	0	2.3	3.5	0	0	0	0	0	0	0	0	0	0
C	1.9	0	0	0	0.7	2.0	0	0	0	0	0	0	0	0	0
D	0	2.3	0	0	0	3.9	0	0	0	0	0	3.8	0	0	0
E	0	3.5	0.7	0	0	0	2.4	0	0	0	0	0	0	0	0
F	0	0	2.0	3.9	0	0	0	3.2	0	0	1.9	0	0	0	0
G	0	0	0	0	2.4	0	0	0	3.3	3.6	0	0	0	0	0
H	0	0	0	0	0	3.2	0	0	0.45	0	0	0	1.0	0	0
I	0	0	0	0	0	0	3.3	0.45	0	1.3	0	0	0	1.0	0
J	0	0	0	0	0	0	3.6	0	1.3	0	0	0	0	2.0	0
K	0	0	0	0	0	1.9	0	0	0	0	0	0.7	1.0	0	0
L	0	0	0	3.8	0	0	0	0	0	0	0.7	0	0	0	3.8
M	0	0	0	0	0	0	0	1.0	0	0	1.0	0	0	0.65	1.3
N	0	0	0	0	0	0	0	0	1.0	2.0	0	0	0.65	0	1.0
O	0	0	0	0	0	0	0	0	0	0	0	3.8	1.3	1.0	0

Es con esta información con la que se parte para realizar la presente evaluación, pues el modelo se ha establecido y ahora se revisará la documentación del algoritmo implementado (véase anexo 2).

El programa inicia declarando todas las variables que se van a necesitar:

```c
main()
{
    setlocale(LC_ALL,"");
    srand(time(NULL));

    //1. Declaramos las variables que usaremos
    /*
        qv: Cantidad de vértices del grafo
        qh: Cantidad de hormigas
        tau: Valor de la feromona inicial
        Q: Parámetro
        alpha: Parámetro
        beta: Parámetro
        rho: Parámetro
        autom: Modo de acción (Para autom >=1 se tomará el número de autom como las repeticiones)
        repeat: Sirve para repetir todo el código nuevamente
        repeat1: Sirve para repetir la colonia en modo manual
        r: Número aleatorio
        accepted: Bandera de aceptación de vértice
        checkIfStuck: Iterador que no tolerará un número determinado de iteraciones. Si se alcanzan, se interpretará qu
        L: Longitud del recorrido completo realizado por La hormiga
    */
    int qv, qh, autom, repeat, repeat1, repeat2, accepted, checkIfStuck, L, h, i, j, k, ip, jp, kp;
    float tau, Q, alpha, beta, rho, r;

    printf("============================== COLONIA DE HORMIGAS ==============================\n");
```

Lo siguiente a realizar es una inicialización de valores estándar y recomendada por el equipo para cada uno de los parámetros.

```c
do
{
    alpha = 0.15;
    autom = 10;
    beta = 0.25;
    Q = 200;
    qh = 5;
    qv = 15;
    repeat = 0;
    rho = 0.01;
    tau = 0.1;
```

Algunos de estos no serán accesibles para el usuario, pues son necesarios para la ejecución como iteradores o controles que se tienen durante el procedimiento, sin embargo otros podrán ser modificables justo después en la función `setValues1`:

```c
//2. Calibramos los primeros valores del programa
setValues1(alpha,autom,beta,Q,qh,qv,rho,tau,&alpha,&autom,&beta,&Q,&qh,&qv,&rho,&tau);
```

Esta función permite calibrar todo en un módulo del programa aparte:

```
void setValues1(float alpha, int autom, float beta, float Q, int qh, int qv, float rho, float taui,float *alphaP, int *automP,
{
    int aux, done = 0;
    *alphaP = alpha;
    *automP = autom;
    *betaP = beta;
    *QP = Q;
    *qhP = qh;
    *qvP = qv;
    *rhoP = rho;
    *tauiP = taui;

    do
    {
        printf("\n");
        printf("--------------------------------\n");
        printf("|  Calibración del algoritmo:\n");
        printf("|  Seleccione algún número si desea cambiar su valor (o 0 para continuar):\n");
        printf("|  0: Listo. Continuar\n");
        printf("|  1: taui (Tau inicial): %.4f\n",taui);
        printf("|  2: alpha (a): %.4f\n",alpha);
        printf("|  3: beta (a): %.4f\n",beta);
        printf("|  4: rho (a): %.4f\n",rho);
        printf("|  5: Q (a): %.4f\n",Q);
        printf("|  6: qh (Cantidad de hormigas): %d\n",qh);
        printf("|  7: qv (Cantidad de vértices que posee el grafo): %d\n",qv);
        printf("|  8: autom (Modo de acción): %d\n",autom);
        printf("|  ");
        scanf("%d",&aux);
```

```
switch (aux)
{
    case 0:
        {
            //Se continúa con el programa
            done = 1;
            break;
        }
    case 1:
        {
            //Se modifica taui

            printf("|     Inserte un nuevo valor para taui (Antiguo valor para taui: T0 = %.4f)\n",taui);
            printf("|  ");
            scanf("%f",&taui);
            *tauiP = taui;

            break;
        }
    case 2:
        {
            //Se modifica alpha

            printf("|     Inserte un nuevo valor para alpha (Antiguo valor para alpha: alpha = %.4f)\n",alpha);
            printf("|  ");
            scanf("%f",&alpha);
            *alphaP = alpha;

            break;
        }
```

Al momento de ejecutar el programa, esta función nos mostrará un menú en el que pueden calibrarse los valores que modifican el procedimiento de la colonia de hormigas.

Si se desea modificar uno basta con seleccionar la opción e indicar posteriormente el nuevo valor para este parámetro.

El menú desplegable es el siguiente:

```
============================== COLONIA DE HORMIGAS ==============================

--------------------------------
|  Calibración del algoritmo:
|  Seleccione algún número si desea cambiar su valor (o 0 para continuar):
|  0: Listo. Continuar
|  1: taui (Tau inicial): 0.1000
|  2: alpha (a): 0.1500
|  3: beta (a): 0.2500
|  4: rho (a): 0.0100
|  5: Q (a): 200.0000
|  6: qh (Cantidad de hormigas): 5
|  7: qv (Cantidad de vértices que posee el grafo): 15
|  8: autom (Modo de acción): 10
|
```

Muchos de estos valores son necesarios para la construcción de estructuras de datos y las estructuras de datos también son necesarias en este procedimiento, por lo tanto, el siguiente paso será la declaración de todas las estructuras a usar:

```
//3. Declaramos más variables
/*
    graph: La matriz de adyacencia del grafo que analizaremos                                   INT
    pheromone: La matriz de feromonas                                                           FLOAT
    vision: La matriz de visibilidad                                                            FLOAT
    tabu: La lista tabú                                                                         INT
    beginEnd: Matriz con los valores de inicio y fin de cada hormiga                            INT
    availableV: Vector donde cada elemento representa cada vértice del grafo (1 está disponible o 0 no lo está)    INT
    prob: Probabilidad individual y acumulada de tomar cada nodo                                FLOAT
    dtau: La diferencia de tau que hay por hormiga en cada nodo                                 FLOAT
*/
float graph[qv][qv], pheromone[qv][qv], vision[qv][qv], tabu[qh][qv], beginEnd[qh][2], availableV[qv], prob[qv][2], dtau[qh][qv]
```

Seguidamente, se inicializan las estructuras con valores que se deseen tener en el inicio del algoritmo:

```
//4. Inicializamos las estructuras
//Limpiamos graph
fillMatrix2D(qv,qv,graph,0);

//Inicializamos pheromone
fillMatrix2D(qv,qv,pheromone,tau);

//Limpiamos vision
fillMatrix2D(qv,qv,vision,0);

//Limpiamos tabu
fillMatrix2D(qh,qv,tabu,0);

//Limpiamos beginEnd
fillMatrix2D(qh,2,beginEnd,0);

//Inicializamos availableV
fillMatrix1D(qv,availableV,1);

//Limpiamos prob
fillMatrix2D(qv,2,prob,0);
```

Cada función `fillMatrix` es una función para rellenar una matriz de dimensión indicada en el nombre de la función con un valor predeterminado que es pasado como parámetro en la función.

```
void fillMatrix1D(int a, float matrix[a], float n)
{
    int i,j;
    for(i = 0; i < a; i++)
    {
        matrix[i] = n;
    }
    return;
}
```

```
void fillMatrix2D(int a, int b, float matrix[a][b], float n)
{
    int i,j;
    for(i = 0; i < a; i++)
    {
        for(j = 0; j < b; j++)
        {
            matrix[i][j] = n;
        }
    }
    return;
}
```

```
void fillMatrix3D(int a, int b, int c, float matrix[a][b][c], float n)
{
    int i,j,k;
    for(i = 0; i < a; i++)
    {
        for(j = 0; j < b; j++)
        {
            for(k = 0; k < c; k++)
            {
                matrix[i][j][k] = n;
            }
        }
    }
    return;
}
```

Inmediatamente después, sucede la aparición de otro menú para calibrar las estructuras de datos:

```
//5. Calibramos los demás valores del programa
setValues2(qv,qh,graph,pheromone,vision,beginEnd);
```

La estructura es similar a `setValues1`:

```
void setValues2(int qv, int qh, float graph[qv][qv], float pheromone[qv][qv], float vision[qv][qv], float beginEnd[qh][2])
{
    int aux, done = 0,i,j;
    do
    {
        printf("\n");
        printf("--------------------------------\n");
        printf("|  Calibración del algoritmo:\n");
        printf("|  Seleccione algún número si desea cambiar su valor (o 0 para continuar):\n");
        printf("|  0: Listo. Continuar\n");
        printf("|  1: graph (El grafo a evaluar):\n");
        for(i = 0; i < qv; i++)
        {
            printf("|     ");
            for(j = 0; j < qv; j++)
            {
                printf("[%d] ",(int)graph[i][j]);
            }
            printf("\n");
        }
        printf("|  2: BeginEnd (Los puntos de inicio y final de cada hormiga):\n");
        for(i = 0; i < qh; i++)
        {
            printf("|     ");
            for(j = 0; j < 2; j++)
            {
                printf("[%d] ",(int)beginEnd[i][j]);
            }
            printf("\n");
        }
        printf("|  ");
        scanf("%d",&aux);
```

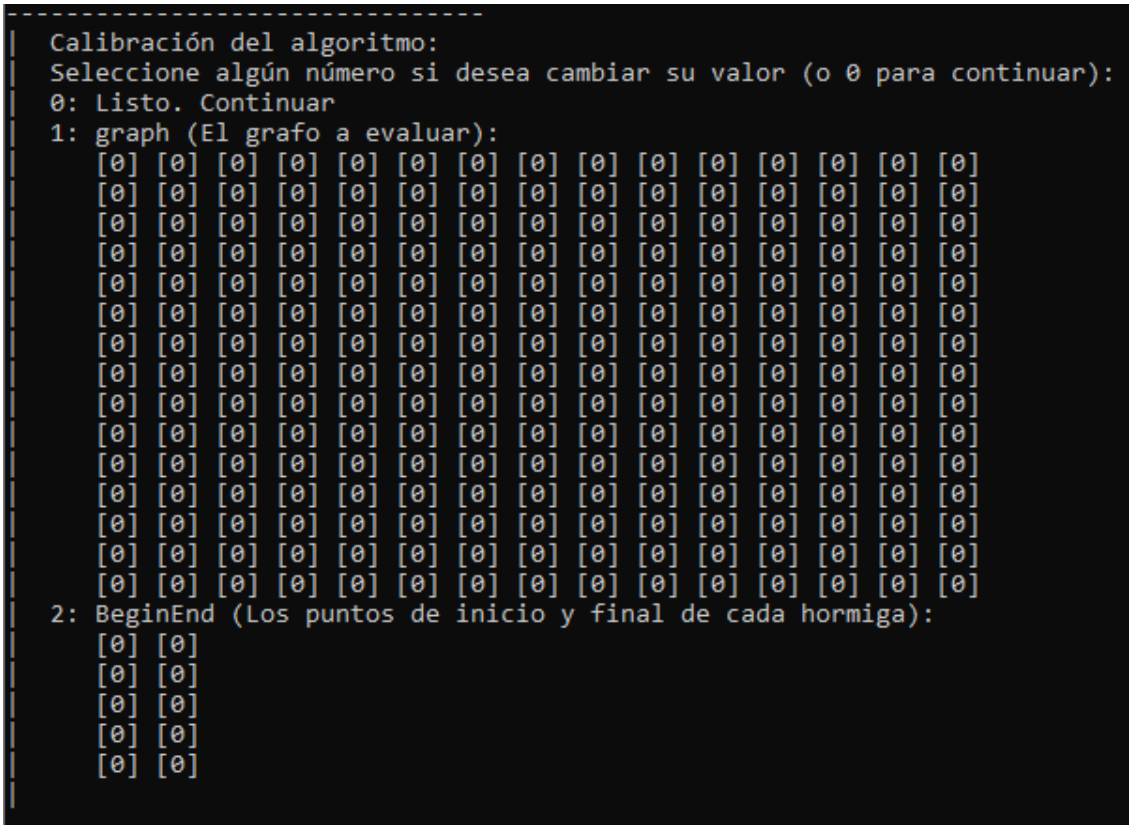
```
switch (aux)
{
    case 0:
        {
            //Se continúa con el programa
            done = 1;
            break;
        }
    case 1:
        {
            //Se modifica graph

            for(i = 0; i < qv; i++)
            {
                for(j = 0; j < qv; j++)
                {
                    printf("|     Inserte el peso de la arista que une al vértice %d con el vértice %d (0 para indicar que no exi
                    printf("|  ");
                    scanf("%f",&graph[i][j]);
                }
            }

            break;
        }
    case 2:
        {
            //Se modifican los puntos de arranque y fin de cada hormiga (tabu)

            for(i = 0; i < qh; i++)
            {
                printf("|     Inserte el nodo inicial de la hormiga %d:\n",i + 1);
                printf("|  ");
                scanf("%f",&beginEnd[i][0]);
```

Si se mantiene la cantidad de hormigas y número de nodos que se recomienda, el programa desplegará el siguiente menú:



Evidentemente, si no se mantiene este número, la estructura cambiará, sin embargo, para resolver el problema planteado por lo menos debe mantenerse la cantidad de nodos del grafo.

La cantidad de hormigas puede calibrarse para realizar pruebas aunque se recomienda dejar cinco hormigas.

Como pequeño “truco” para agilizar la configuración el programa, éste pide que se inserten a mano cada uno de los valores tanto del grafo como de la estructura que guarda el inicio y final del recorrido. Esto hace que sea muy lento rellenar estructuras de datos tan grandes como el grafo que hay que rellenar a mano en este caso.

Para ello, se pensó en el siguiente comando que puede insertarse una vez que se selecciona la opción 1, la cual es la configuración de dicha estructura; posteriormente se procedería a escribir el siguiente comando y presionar ENTER:

```
0 900 1900 0 0 0 0 0 0 0 0 0 0 0 900 0 0 2300 3500 0 0 0 0 0 0 0 0 0 1900 0 0
0 700 2000 0 0 0 0 0 0 0 0 0 2300 0 0 0 3900 0 0 0 0 0 3800 0 0 0 0 3500 700 0
0 0 2400 0 0 0 0 0 0 0 0 0 2000 3900 0 0 0 3200 0 0 1900 0 0 0 0 0 0 0 0 2400 0
0 0 3300 3600 0 0 0 0 0 0 0 0 0 3200 0 0 450 0 0 0 1000 0 0 0 0 0 0 0 0 3300
450 0 1300 0 0 0 1000 0 0 0 0 0 0 0 3600 0 1300 0 0 0 0 2000 0 0 0 0 0 0 1900 0 0
0 0 0 700 1000 0 0 0 0 0 3800 0 0 0 0 0 0 700 0 0 0 3800 0 0 0 0 0 0 0 1000 0 0
1000 0 0 650 1300 0 0 0 0 0 0 0 0 1000 2000 0 0 650 0 1000 0 0 0 0 0 0 0 0 0 0
3800 1300 1000 0
```

El comando anterior inicializa la estructura como la matriz de distancias presentada anteriormente, de modo que se tiene el menú con la configuración mostrada a continuación:

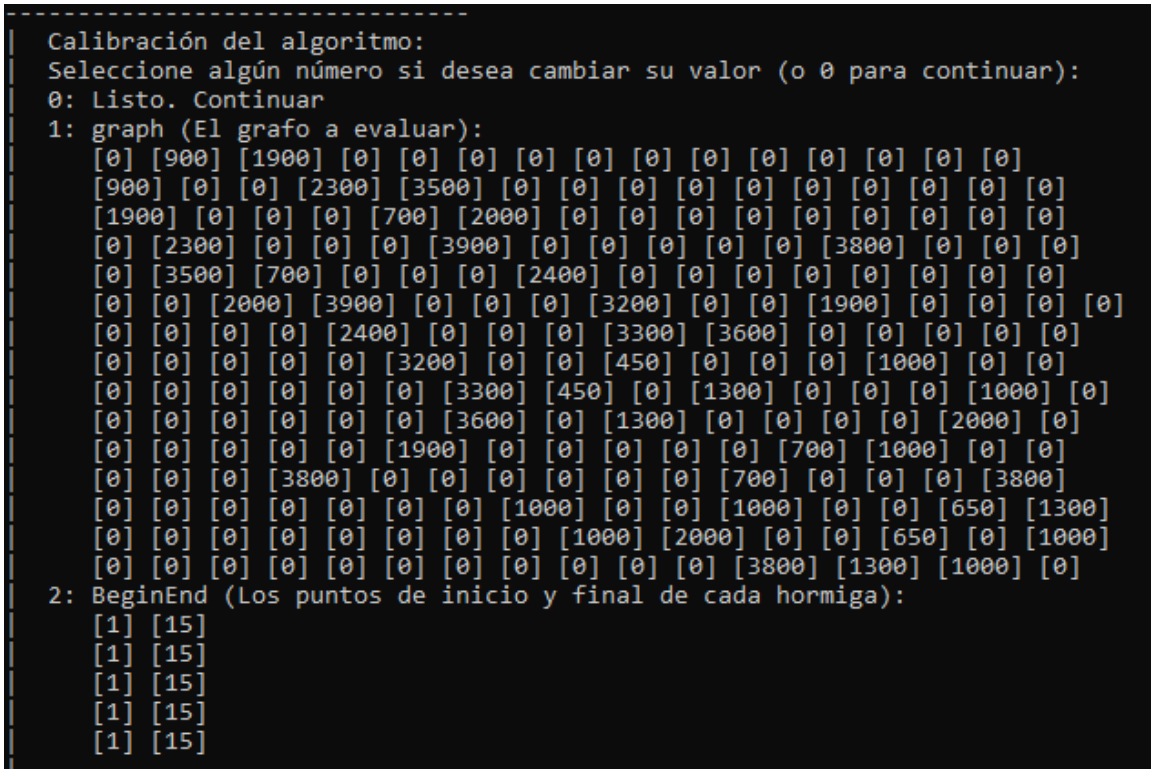
```
--------------------------------
|  Calibración del algoritmo:
|  Seleccione algún número si desea cambiar su valor (o 0 para continuar):
|  0: Listo. Continuar
|  1: graph (El grafo a evaluar):
|     [0] [900] [1900] [0] [0] [0] [0] [0] [0] [0] [0] [0] [0] [0] [0]
|     [900] [0] [0] [2300] [3500] [0] [0] [0] [0] [0] [0] [0] [0] [0] [0]
|     [1900] [0] [0] [0] [700] [2000] [0] [0] [0] [0] [0] [0] [0] [0] [0]
|     [0] [2300] [0] [0] [0] [3900] [0] [0] [0] [0] [0] [3800] [0] [0] [0]
|     [0] [3500] [700] [0] [0] [0] [2400] [0] [0] [0] [0] [0] [0] [0] [0]
|     [0] [0] [2000] [3900] [0] [0] [0] [3200] [0] [0] [1900] [0] [0] [0] [0]
|     [0] [0] [0] [0] [2400] [0] [0] [0] [3300] [3600] [0] [0] [0] [0] [0]
|     [0] [0] [0] [0] [0] [3200] [0] [0] [450] [0] [0] [0] [1000] [0] [0]
|     [0] [0] [0] [0] [0] [0] [3300] [450] [0] [1300] [0] [0] [0] [1000] [0]
|     [0] [0] [0] [0] [0] [0] [3600] [0] [1300] [0] [0] [0] [0] [2000] [0]
|     [0] [0] [0] [0] [0] [1900] [0] [0] [0] [0] [0] [700] [1000] [0] [0]
|     [0] [0] [0] [3800] [0] [0] [0] [0] [0] [0] [700] [0] [0] [0] [3800]
|     [0] [0] [0] [0] [0] [0] [0] [1000] [0] [0] [1000] [0] [0] [650] [1300]
|     [0] [0] [0] [0] [0] [0] [0] [0] [1000] [2000] [0] [0] [650] [0] [1000]
|     [0] [0] [0] [0] [0] [0] [0] [0] [0] [0] [0] [3800] [1300] [1000] [0]
|  2: BeginEnd (Los puntos de inicio y final de cada hormiga):
|     [0] [0]
|     [0] [0]
|     [0] [0]
|     [0] [0]
|     [0] [0]
|
```

Para la siguiente estructura se deberá especificar el nodo con el que se empieza y el nodo en el que termina cada hormiga. Para este caso, todas deben de partir del nodo A y llegar al nodo O, sin embargo el programa no trabaja con caracteres, sino

con números. Sólo basta con realiza la conversión de la letra al respectivo número que se les asigne si se parte de que A es el número 1. La siguiente tabla realiza esta biyección:

A	1
B	2
C	3
D	4
E	5
F	6
G	7
H	8
I	9
J	10
K	11
L	12
M	13
N	14
O	15

Para nuestro propósito, las estructuras se calibran de la siguiente manera:



Al finalizar este procedimiento, sólo se realiza un cálculo más que corresponde a la matriz de visibilidad:

```
//Inicializamos vision
for(i = 0; i < qv; i++)
{
    for(j = 0; j < qv; j++)
    {
        if(graph[i][j] == 0)
        {
            vision[i][j] = 0;
        }
        else
        {
            vision[i][j] = (pow(pheromone[i][j],alpha))/(pow(graph[i][j],beta));
        }
    }
}
```

Y finalmente comienza el algoritmo:

```
//6. Comenzamos la colonia de hormigas, la cual se repetirá tantas veces como se haya calibrado autom (si es 0, se decide
h = 0;
repeat2 = 1;
do
{
    //Inicializamos dtau
    fillMatrix3D(qh,qv,qv,dtau,0);

    if(autom == 0)
    {
        printf("\n");
        printf("\n");
        printf("Comenzamos la %d° iteración\n\n", h + 1);
    }

    //El procedimiento de la colonia se hará para cada hormiga
    i = 0;
    do
    {
        if(autom == 0)
        {
            printf("  Comenzamos a trabajar con la hormiga %d\n", i + 1);
            getchar();
        }

        //Inicializamos tabu y availableV con los valores proporcionados por beginEnd
        startProcess(i,qh,qv,tabu,beginEnd,availableV);

        if(autom == 0)
        {
            printf("    Inicializamos la lista tabú con el valor del nodo de inicio:\n    ");
            for(ip = 0; ip < qh; ip++)
            {
```

Como pequeño apunte, en el primer menú existe una variable `autom`. Esta variable hace referencia a activar el modo de acción automático o manual, sin embargo, se cambió su uso de modo que imprima y avance de diferente manera según que número se le indique.

Si se le indica con un valor igual a cero, el modo de acción automático estará desactivado y desglosará todo el procedimiento del algoritmo. El usuario decide si realizar más y más poblaciones o detenerlo en el momento que él decida. Si se le asigna a `autom` un entero positivo, se tomará este valor como la cantidad de iteraciones que se realizarán sin tanto desglose como en el modo anterior.

```
  Hemos escogido el nodo 11 (r = 0.0910)
  Revisando el nodo 11 con tabú, obtuvimos que éste no es aceptado
  Tendremos que repetir la selección aleatoria por posible encrucijada (La hormiga lleva 4 repeticiones)
    Tenemos las probabilidades individuales y acumuladas de que de 12 se vaya a
       1:     [0.0000]     [0.0000]
       2:     [0.0000]     [0.0000]
       3:     [0.0000]     [0.0000]
       4:     [0.0902]     [0.0902]
       5:     [0.0000]     [0.0902]
       6:     [0.0000]     [0.0902]
       7:     [0.0000]     [0.0902]
       8:     [0.0000]     [0.0902]
       9:     [0.0000]     [0.0902]
       10:     [0.0000]     [0.0902]
       11:     [0.1376]     [0.2278]
       12:     [0.0000]     [0.2278]
       13:     [0.0000]     [0.2278]
       14:     [0.0000]     [0.2278]
       15:     [0.0902]     [0.3180]

  Hemos escogido el nodo 15 (r = 0.2898)
  Revisando el nodo 15 con tabú, obtuvimos que éste es aceptado
  Hemos aceptado el nodo. Ahora tenemos el camino siguiente
  [1] [2] [4] [6] [8] [13] [11] [12] [15] [0] [0] [0] [0] [0] [0]
  [0] [0] [0] [0] [0] [0] [0] [0] [0] [0] [0] [0] [0] [0] [0]
  [0] [0] [0] [0] [0] [0] [0] [0] [0] [0] [0] [0] [0] [0] [0]
  [0] [0] [0] [0] [0] [0] [0] [0] [0] [0] [0] [0] [0] [0] [0]
  [0] [0] [0] [0] [0] [0] [0] [0] [0] [0] [0] [0] [0] [0] [0]
  [0] [0] [1] [0] [1] [0] [1] [0] [1] [1] [0] [0] [0] [1] [0]
  Hemos llegado al nodo objetivo

La hormiga produjo un recorrido de L = 16800
```

```
En la iteración 1, la hormiga 5 produjo el camino:
[1] [2] [4] [12] [15] [0] [0] [0] [0] [0] [0] [0] [0] [0] [0]
Con un recorrido de valor 10800
```

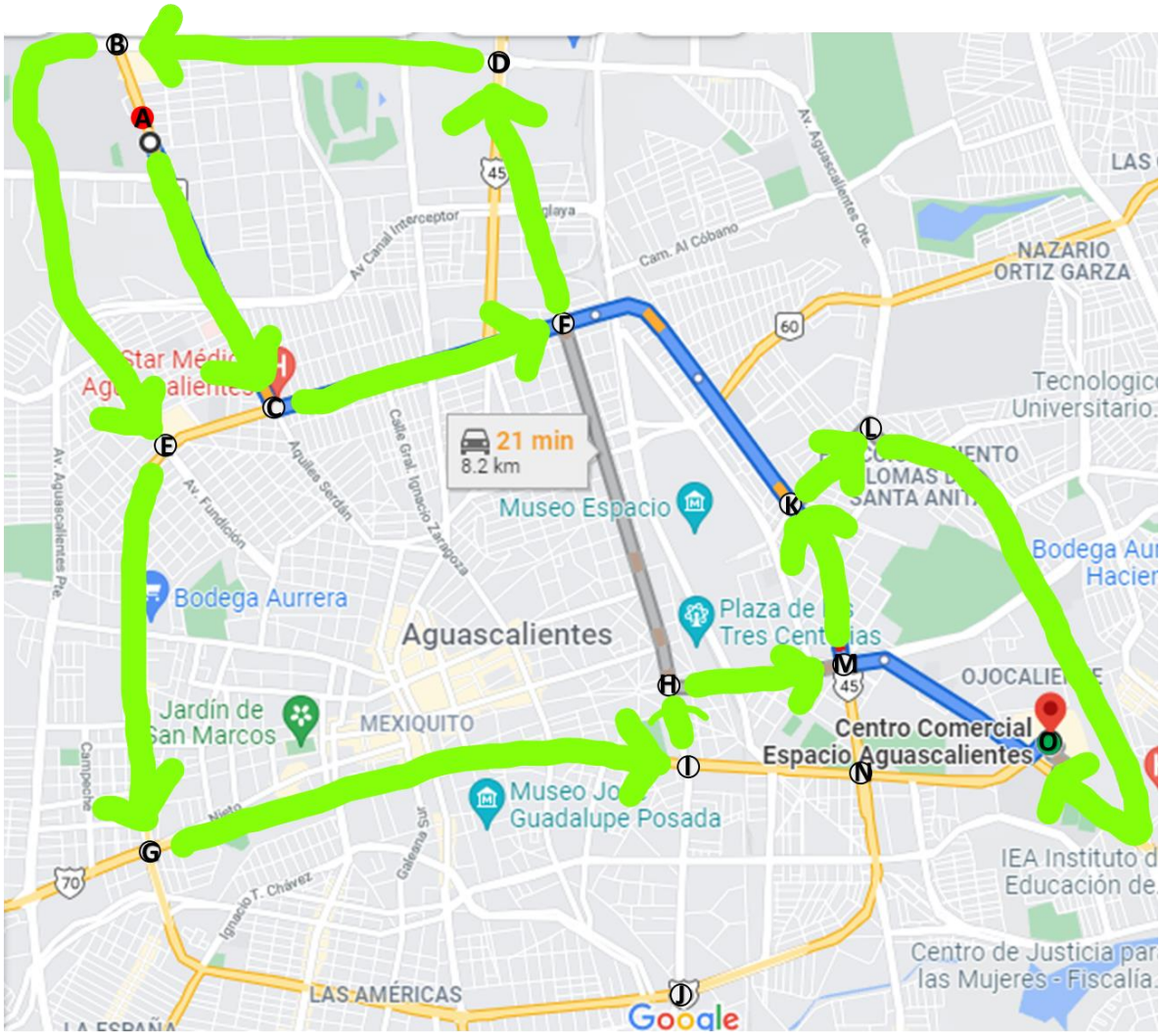
En resumen, el programa inicializa primero la lista tabú y los nodos disponibles para transitar, después calcula y selecciona el próximo nodo, revisa que la hormiga no se haya quedado atorada en un camino sin retorno (ya que elegir un nodo elimina la posibilidad de tomarlo nuevamente) y realiza este procedimiento hasta encontrar el nodo que busca. Finalmente actualiza la tabla tridimensional de feromonas que actualiza y repite el procedimiento según se le haya indicado.

Los resultados que arroja el algoritmo con estos parámetros son abundantes. El algoritmo describe los caminos que cada hormiga hizo, así como la iteración en la que sucedió esto:

```
En la iteración 1, la hormiga 1 produjo el camino:
[1] [3] [6] [4] [2] [5] [7] [9] [8] [13] [11] [12] [15] [0] [0]
Con un recorrido de valor 26250
En la iteración 1, la hormiga 2 produjo el camino:
[1] [3] [6] [4] [2] [5] [7] [9] [8] [13] [11] [12] [15] [0] [0]
Con un recorrido de valor 12600
En la iteración 1, la hormiga 3 produjo el camino:
[1] [3] [6] [4] [2] [5] [7] [9] [8] [13] [11] [12] [15] [0] [0]
Con un recorrido de valor 17800
En la iteración 1, la hormiga 4 produjo el camino:
[1] [3] [6] [4] [2] [5] [7] [9] [8] [13] [11] [12] [15] [0] [0]
Con un recorrido de valor 12150
En la iteración 1, la hormiga 5 produjo el camino:
[1] [3] [6] [4] [2] [5] [7] [9] [8] [13] [11] [12] [15] [0] [0]
Con un recorrido de valor 11600
En la iteración 1, la hormiga 6 produjo el camino:
[1] [3] [6] [4] [2] [5] [7] [9] [8] [13] [11] [12] [15] [0] [0]
Con un recorrido de valor 23300
En la iteración 1, la hormiga 7 produjo el camino:
[1] [3] [6] [4] [2] [5] [7] [9] [8] [13] [11] [12] [15] [0] [0]
Con un recorrido de valor 11400
En la iteración 1, la hormiga 8 produjo el camino:
[1] [3] [6] [4] [2] [5] [7] [9] [8] [13] [11] [12] [15] [0] [0]
Con un recorrido de valor 13500
En la iteración 1, la hormiga 9 produjo el camino:
[1] [3] [6] [4] [2] [5] [7] [9] [8] [13] [11] [12] [15] [0] [0]
Con un recorrido de valor 13700
En la iteración 1, la hormiga 10 produjo el camino:
[1] [3] [6] [4] [2] [5] [7] [9] [8] [13] [11] [12] [15] [0] [0]
Con un recorrido de valor 14450
En la iteración 1, la hormiga 11 produjo el camino:
[1] [3] [6] [4] [2] [5] [7] [9] [8] [13] [11] [12] [15] [0] [0]
Con un recorrido de valor 15750
En la iteración 1, la hormiga 12 produjo el camino:
[1] [3] [6] [4] [2] [5] [7] [9] [8] [13] [11] [12] [15] [0] [0]
Con un recorrido de valor 12750
En la iteración 1, la hormiga 13 produjo el camino:
[1] [3] [6] [4] [2] [5] [7] [9] [8] [13] [11] [12] [15] [0] [0]
Con un recorrido de valor 13450
En la iteración 1, la hormiga 14 produjo el camino:
[1] [3] [6] [4] [2] [5] [7] [9] [8] [13] [11] [12] [15] [0] [0]
Con un recorrido de valor 21600
En la iteración 1, la hormiga 15 produjo el camino:
[1] [3] [6] [4] [2] [5] [7] [9] [8] [13] [11] [12] [15] [0] [0]
Con un recorrido de valor 10800
En la iteración 1, la hormiga 16 produjo el camino:
[1] [3] [6] [4] [2] [5] [7] [9] [8] [13] [11] [12] [15] [0] [0]
Con un recorrido de valor 12650
En la iteración 1, la hormiga 17 produjo el camino:
[1] [3] [6] [4] [2] [5] [7] [9] [8] [13] [11] [12] [15] [0] [0]
Con un recorrido de valor 13500
En la iteración 1, la hormiga 18 produjo el camino:
[1] [3] [6] [4] [2] [5] [7] [9] [8] [13] [11] [12] [15] [0] [0]
Con un recorrido de valor 13700
En la iteración 1, la hormiga 19 produjo el camino:
[1] [3] [6] [4] [2] [5] [7] [9] [8] [13] [11] [12] [15] [0] [0]
Con un recorrido de valor 28700
En la iteración 1, la hormiga 20 produjo el camino:
[1] [3] [6] [4] [2] [5] [7] [9] [8] [13] [11] [12] [15] [0] [0]
Con un recorrido de valor 8450
```

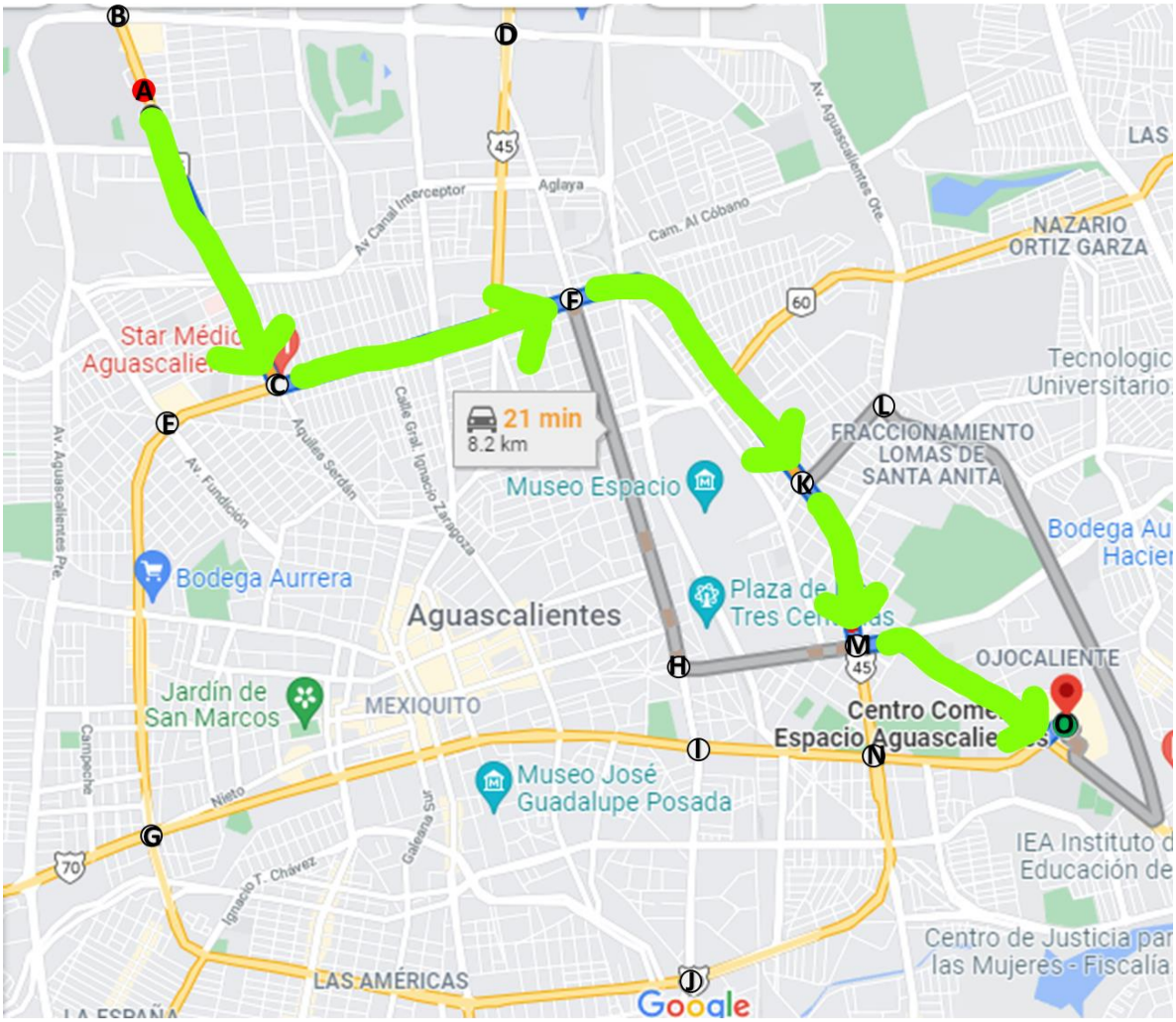
Donde los caminos sólo están mencionados con números en lugar de con letras, pero expresan lo mismo (1 hace referencia a A, 2 representa B, 3 a C, …, 15 a O).

Por ejemplo, puede observarse en la imagen anterior que la hormiga 1 en la primera iteración realizó el camino 1, 3, 6, 4, 2, 5, 7, 9, 8, 13, 11, 12, 15, es decir, el camino A, C, F, D, B, E, G, I, H, M, K, L, O. Este recorrido en el mapa se puede visualizar como:



Y el valor del recorrido es la distancia en metros, es decir, este recorrido es de 28.7 kilómetros.

Al avanzar, el algoritmo continúa en su exploración y se encuentra con caminos que optimizan mejor la distancia, minimizándola de una forma sorprendente, por ejemplo, el equipo cree que es muy difícil encontrar un camino mejor que el camino A, C, F, K, M, O representado de la siguiente forma:



Al tomar este camino se recorre una distancia de 8.1 kilómetros. Sin embargo, el algoritmo realiza una optimización de modo que lo encontramos en pocas iteraciones como se muestra a continuación:

```
En la iteración 8, la hormiga 5 produjo el camino:
[1] [3] [6] [11] [13] [15] [0] [0] [0] [0] [0] [0] [0] [0] [0]
Con un recorrido de valor 8100
```

Siendo justamente el camino que representa el A, C, F, K, M, O.

Comparando estos dos caminos, es posible observar que la optimización por colonia de hormigas funciona de modo que permite encontrar un buen camino, sin embargo, es necesario calibrar correctamente todos los parámetros, pues el algoritmo encuentra múltiples soluciones como se observa a continuación:

```
En la iteración 6, la hormiga 3 produjo el camino:
[1] [2] [4] [12] [15] [0] [0] [0] [0] [0] [0] [0] [0] [0] [0]
Con un recorrido de valor 10800
En la iteración 6, la hormiga 3 produjo el camino:
[1] [3] [5] [7] [9] [14] [15] [0] [0] [0] [0] [0] [0] [0] [0]
Con un recorrido de valor 10300
En la iteración 6, la hormiga 3 produjo el camino:
[1] [3] [5] [7] [10] [14] [9] [8] [13] [11] [12] [15] [0] [0] [0]
Con un recorrido de valor 18550
En la iteración 6, la hormiga 4 produjo el camino:
[1] [3] [5] [2] [4] [12] [15] [0] [0] [0] [0] [0] [0] [0] [0]
Con un recorrido de valor 16000
En la iteración 6, la hormiga 4 produjo el camino:
[1] [3] [5] [2] [4] [12] [15] [0] [0] [0] [0] [0] [0] [0] [0]
Con un recorrido de valor 16000
En la iteración 6, la hormiga 4 produjo el camino:
[1] [2] [4] [12] [15] [0] [0] [0] [0] [0] [0] [0] [0] [0] [0]
Con un recorrido de valor 10800
En la iteración 6, la hormiga 4 produjo el camino:
[1] [3] [5] [2] [4] [12] [15] [0] [0] [0] [0] [0] [0] [0] [0]
Con un recorrido de valor 16000
En la iteración 6, la hormiga 4 produjo el camino:
[1] [3] [5] [7] [10] [14] [9] [8] [13] [11] [12] [15] [0] [0] [0]
Con un recorrido de valor 18550
En la iteración 6, la hormiga 5 produjo el camino:
[1] [3] [5] [2] [4] [12] [15] [0] [0] [0] [0] [0] [0] [0] [0]
Con un recorrido de valor 16000
En la iteración 6, la hormiga 5 produjo el camino:
[1] [3] [5] [2] [4] [12] [15] [0] [0] [0] [0] [0] [0] [0] [0]
Con un recorrido de valor 16000
En la iteración 6, la hormiga 5 produjo el camino:
[1] [2] [4] [12] [15] [0] [0] [0] [0] [0] [0] [0] [0] [0] [0]
Con un recorrido de valor 10800
En la iteración 6, la hormiga 5 produjo el camino:
[1] [3] [5] [2] [4] [12] [15] [0] [0] [0] [0] [0] [0] [0] [0]
Con un recorrido de valor 16000
En la iteración 6, la hormiga 5 produjo el camino:
[1] [2] [4] [6] [11] [12] [15] [0] [0] [0] [0] [0] [0] [0] [0]
Con un recorrido de valor 13500
En la iteración 7, la hormiga 1 produjo el camino:
[1] [2] [4] [6] [11] [12] [15] [0] [0] [0] [0] [0] [0] [0] [0]
Con un recorrido de valor 13500
En la iteración 7, la hormiga 1 produjo el camino:
[1] [3] [5] [2] [4] [12] [15] [0] [0] [0] [0] [0] [0] [0] [0]
Con un recorrido de valor 16000
En la iteración 7, la hormiga 1 produjo el camino:
[1] [2] [4] [12] [15] [0] [0] [0] [0] [0] [0] [0] [0] [0] [0]
Con un recorrido de valor 10800
```

Conclusiones

Andrea Melissa Almeida Ortega: Las hormigas son listas, fuertes, bronceadas y tienen vaquitas por ello son superiores.

Joel Alejandro Espinoza Sánchez: El uso de colonia de hormigas para optimizar problemas como este muestra que las aplicaciones de buscar técnicas de optimización no sólo ahorran recursos computacionales, sino también pueden ser muy útiles en reducción de costos. Por ello es muy beneficioso aplicarlos.

Óscar Alonso Flores Fernández: Ya no trabajamos con hormigas, son ciempiés así que el futuro es hoy.

Dariana Gómez Garza: Lo que dice Hiram además de que como tienen más patitas c: son una especie más evolucionada <3.

Fernando Francisco González Arenas: El algoritmo de la colonia de hormigas es una metaheurística muy importante útil para optimizar problemas de búsqueda por medio de la simulación del proceso biológico de la búsqueda de comida de las hormigas. estos insectos van buscando la comida y en el camino van dejando feromonas, las cuales son seguidas por sus compañeras de la misma colonia y así encuentran el mejor camino a base de probabilidad, feromonas y evaporación de las mismas.

Este algoritmo es muy importante para la optimización de caminos, ya que las hormigas llegan a una posible solución y cada que otra llega a una más, se comparan y se elige la que sea una mejor opción. así se va mejorando el resultado y se evita caer en óptimos locales, haciendo que las hormigas elijan probabilísticamente el camino, pero siendo influenciadas por las feromonas que dejaron previamente sus compañeras.

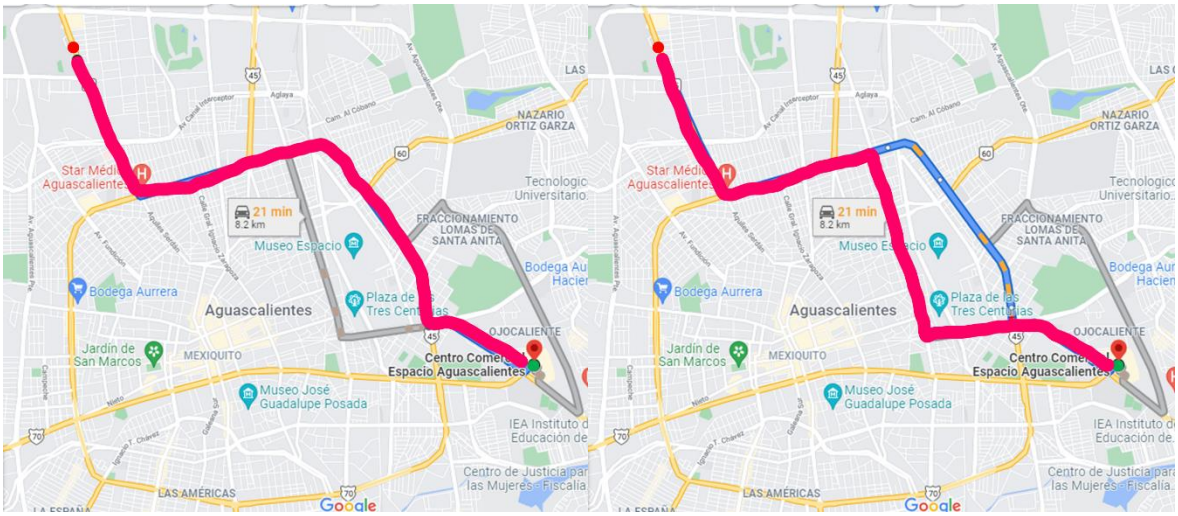
Hiram Efraín Orocio García: Las hormigas son lindas y confío en ellas para hacer rutas.

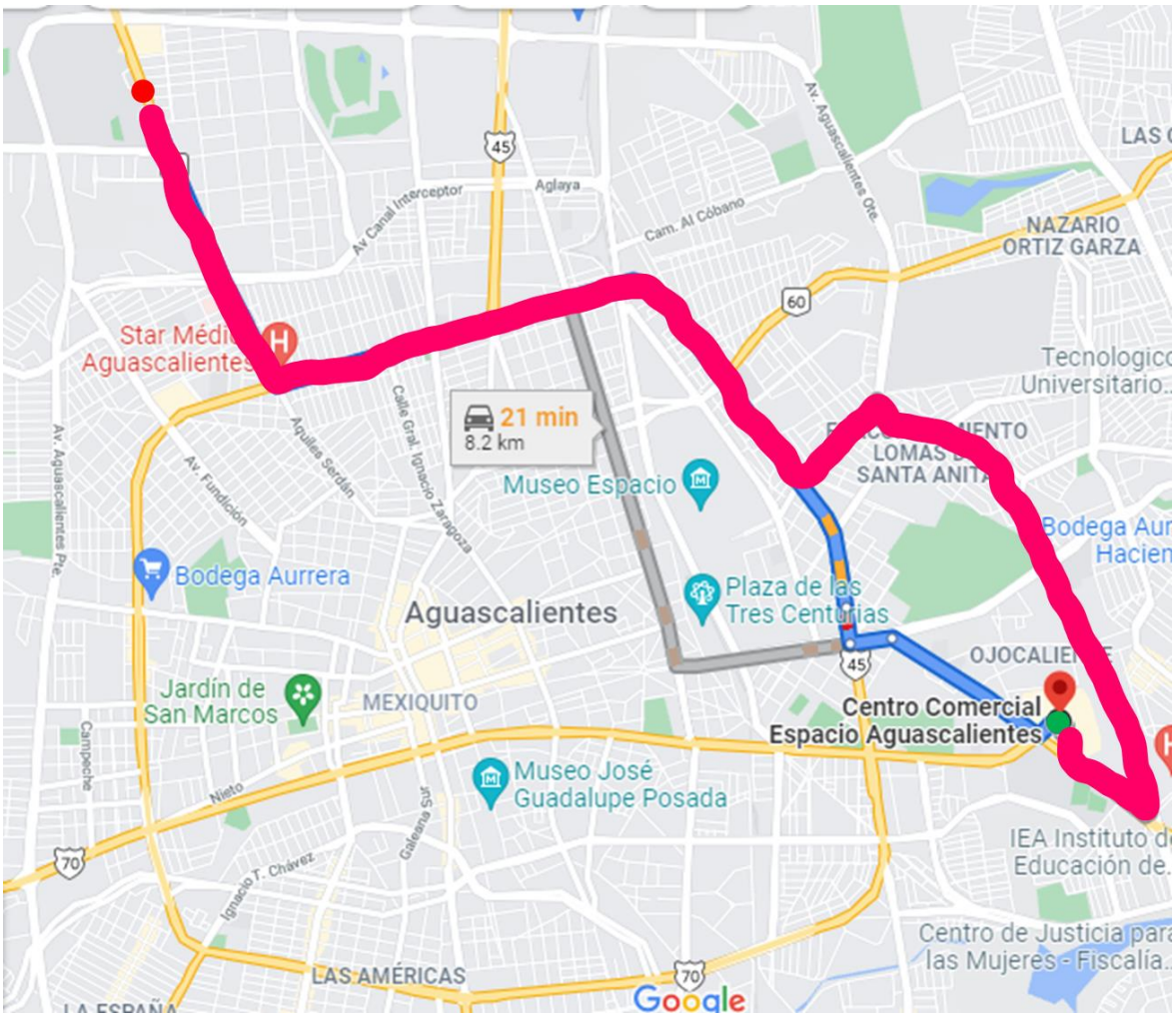
Referencias

- Google. (2021). Ciudad de Aguascalientes. Noviembre 22, 2021, de Google Maps Sitio web: https://www.google.com.mx/maps/preview.

Anexos

Anexo 1: Propuestas del grafo final.





Anexo 2: Algoritmo de Colonia de Hormigas implementado en C.

```
/*
    Universidad Autónoma de Aguascalientes

                Centro de Ciencias Básicas
  Departamento de Ciencias de la Computación
                    Metaheurísticas I

                                    7° "A"

                        Evaluación Final

        Profesor: Francisco Javier Luna Rosas

     Alumnos:
                Almeida Ortega Andrea Melissa
                Espinoza Sänchez Joel Alejandro
                Flores Fernández Óscar Alonso
                Gómez Garza Dariana
                González Arenas Fernando Francisco
                Orocio García Hiram Efraín

  Fecha de Entrega: 25 de noviembre del 2021

*/
//Cargamos las librerías
#include <stdio.h>
#include <stdlib.h>
#include <locale.h>
#include <time.h>
#include <math.h>

void setValues1(float alpha, int autom, float beta, float Q, int qh, int qv, float rho,
float taui,float *alphaP, int *automP, float *betaP, float *QP, int *qhP, int *qvP, float
*rhoP, float *tauiP);
void fillMatrix1D(int a, float matrix[a], float n);
void fillMatrix2D(int a, int b, float matrix[a][b], float n);
void fillMatrix3D(int a, int b, int c, float matrix[a][b][c], float n);
void setValues2(int qv, int qh, float graph[qv][qv], float pheromone[qv][qv], float
vision[qv][qv], float beginEnd[qh][2]);
void startProcess(int i, int qh, int qv, float tabu[qh][qv], float beginEnd[qh][2], float
availableV[qv]);
int acceptVertex(int i, int qh, int qv, int value, float tabu[qh][qv]);
int getL(int i, int qh, int qv, float graph[qv][qv], float tabu[qh][qv], float
beginEnd[qh][2]);

main()
{
       setlocale(LC_ALL,"");
       srand(time(NULL));

       //1. Declaramos las variables que usaremos
       /*
              qv: Cantidad de vértices del grafo
              qh: Cantidad de hormigas
              tau: Valor de la feromona inicial
              Q: Parámetro
              alpha: Parámetro
              beta: Parámetro
```

```
		rho: Parámetro
		autom: Modo de acción (Para autom >=1 se tomará el número de autom como las repeticiones)
		repeat: Sirve para repetir todo el código nuevamente
		repeat1: Sirve para repetir la colonia en modo manual
		r: Número aleatorio
		accepted: Bandera de aceptación de vértice
		checkIfStuck: Iterador que no tolerará un número determinado de iteraciones. Si se alcanzan, se interpretará que la hormiga se encerró
		L: Longitud del recorrido completo realizado por la hormiga
	*/
	int qv, qh, autom, repeat, repeat1, repeat2, accepted, checkIfStuck, L, h, i, j, k, ip, jp, kp;
	float tau, Q, alpha, beta, rho, r;

	printf("==============================         COLONIA         DE         HORMIGAS ==============================\n");

	do
	{
		alpha = 0.15;
		autom = 10;
		beta = 0.25;
		Q = 200;
		qh = 5;
		qv = 15;
		repeat = 0;
		rho = 0.01;
		tau = 0.1;

		//2. Calibramos los primeros valores del programa

	setValues1(alpha,autom,beta,Q,qh,qv,rho,tau,&alpha,&autom,&beta,&Q,&qh,&qv,&rho,&tau);

		//3. Declaramos más variables
		/*
			graph: La matriz de adyacencia del grafo que analizaremos
			                                                                                INT
			pheromone: La matriz de feromonas

			FLOAT
			vision: La matriz de visibilidad

			FLOAT
			tabu: La lista tabú

			              INT
			beginEnd: Matriz con los valores de inicio y fin de cada hormiga
			                                                                    INT
			availableV: Vector donde cada elemento representa cada vértice del grafo (1 está disponible o 0 no lo está)          INT
			prob: Probabilidad individual y acumulada de tomar cada nodo

	FLOAT
			dtau: La diferencia de tau que hay por hormiga en cada nodo

	FLOAT
		*/
		float graph[qv][qv], pheromone[qv][qv], vision[qv][qv], tabu[qh][qv], beginEnd[qh][2], availableV[qv], prob[qv][2], dtau[qh][qv][qv];
```

```
		//4. Inicializamos las estructuras
		//Limpiamos graph
		fillMatrix2D(qv,qv,graph,0);

		//Inicializamos pheromone
		fillMatrix2D(qv,qv,pheromone,tau);

		//Limpiamos vision
		fillMatrix2D(qv,qv,vision,0);

		//Limpiamos tabu
		fillMatrix2D(qh,qv,tabu,0);

		//Limpiamos beginEnd
		fillMatrix2D(qh,2,beginEnd,0);

		//Inicializamos availableV
		fillMatrix1D(qv,availableV,1);

		//Limpiamos prob
		fillMatrix2D(qv,2,prob,0);

		//5. Calibramos los demás valores del programa
		setValues2(qv,qh,graph,pheromone,vision,beginEnd);

		//Inicializamos vision
		for(i = 0; i < qv; i++)
		{
			for(j = 0; j < qv; j++)
			{
				if(graph[i][j] == 0)
				{
					vision[i][j] = 0;
				}
				else
				{
					vision[i][j]                                                    =
(pow(pheromone[i][j],alpha))/(pow(graph[i][j],beta));
				}
			}
		}

		printf("\n\n\n\n");

		if(autom == 0)
		{
			printf("El grafo a evaluar es:\n");
			for(ip = 0; ip < qv; ip++)
			{
				for(jp = 0; jp < qv; jp++)
				{
					printf("[%d] ", (int)graph[ip][jp]);
				}
				printf("\n");
			}
			printf("\n");

			printf("La  matriz  de  feromonas  antes  del  comienzo  del  algoritmo
es:\n");
			for(ip = 0; ip < qv; ip++)
```

```
                        {
                                for(jp = 0; jp < qv; jp++)
                                {
                                        printf("[%.4f] ", pheromone[ip][jp]);
                                }
                                printf("\n");
                        }
                        printf("\n");

                        printf("La matriz de visibilidad, entonces, es:\n");
                        for(ip = 0; ip < qv; ip++)
                        {
                                for(jp = 0; jp < qv; jp++)
                                {
                                        printf("[%.4f] ", vision[ip][jp]);
                                }
                                printf("\n");
                        }
                        printf("\n");
                        getchar();
                        getchar();
                }

                //6. Comenzamos la colonia de hormigas, la cual se repetirá tantas veces
como se haya calibrado autom (si es 0, se decide el paro)
                h = 0;
                repeat2 = 1;
                do
                {
                        //Inicializamos dtau
                        fillMatrix3D(qh,qv,qv,dtau,0);

                        if(autom == 0)
                        {
                                printf("\n");
                                printf("\n");
                                printf("Comenzamos la %dº iteración\n\n", h + 1);
                        }

                        //El procedimiento de la colonia se hará para cada hormiga
                        i = 0;
                        do
                        {
                                if(autom == 0)
                                {
                                        printf("  Comenzamos a trabajar con la hormiga %d\n",
i + 1);
                                        getchar();
                                }

                                //Inicializamos   tabu   y   availableV   con   los   valores
proporcionados por beginEnd
                                startProcess(i,qh,qv,tabu,beginEnd,availableV);

                                if(autom == 0)
                                {
                                        printf("    Inicializamos la lista tabú con el valor
del nodo de inicio:\n    ");
                                        for(ip = 0; ip < qh; ip++)
                                        {
                                                for(jp = 0; jp < qv; jp++)
```

```
                                        {
                                                printf("[%d] ", (int)tabu[ip][jp]);
                                        }
                                        printf("\n    ");
                                }
                                printf("\n");

                                printf("         Inicializamos  la  lista  de  nodos
disponibles:\n    ");
                                for(ip = 0; ip < qv; ip++)
                                {
                                        printf("[%d] ", (int)availableV[ip]);
                                }
                                printf("\n");

                                getchar();
                        }

                        //Todavía, el procedimiento de selección de camino de cada
hormiga tiene que hacerse por cada nodo como máximo qv - 1 veces
                        j = 0;
                        do
                        {
                                if(autom == 0)
                                {
                                        printf("\n");
                                        printf("       Escogeremos el nodo %d\n", j +
1);
                                }

                                //Debemos repetir este análisis hasta que el nodo haya
sido reconocido como válido
                                accepted = 0;
                                checkIfStuck = 0;
                                do
                                {
                                        //A partir del nodo otorgado, vamos al renglón
en vision con el que rellenaremos prob y procedemos a rellenar prob
                                        prob[0][0] = vision[(int)tabu[i][j] - 1][0];
                                        prob[0][1] = prob[0][0];
                                        for(k = 1; k < qv; k++)
                                        {
                                                prob[k][0]  =  vision[(int)tabu[i][j]  -
1][k];
                                                prob[k][1]  =  prob[k][0]  +  prob[k  -
1][1];
                                        }

                                        //Generamos un número aleatorio y discretizamos
el valor entre 10000 posibilidades, todas dentro del rango
                                        r = rand() % 10000;
                                        r = r/(10000/prob[qv - 1][1]);

                                        if(autom == 0)
                                        {
                                                printf("                   Tenemos  las
probabilidades individuales y acumuladas de que de %d se vaya a\n",(int)tabu[i][j]);
                                                for(ip = 0; ip < qv; ip++)
                                                {
                                                        printf("           %d:     ", ip
+ 1);
```

```
                                                        for(jp = 0; jp < 2; jp++)
                                                        {
                                                                printf("[%.4f]         ",
prob[ip][jp]);
                                                        }
                                                        printf("\n");
                                                }
                                                printf("\n");
                                        }

                                        //Ubicaremos el nodo seleccionado
                                        for(k = 0; k < qv; k++)
                                        {
                                                if(r < prob[k][1])
                                                {
                                                        //Encontramos que cayó en k + 1
                                                        break;
                                                }
                                        }

                                        if(autom == 0)
                                        {
                                                printf("      Hemos escogido el nodo %d
(r = %.4f)\n",k + 1,r);
                                        }

                                        //Ahora toca validar si ese men está disponible
o no
                                        accepted = acceptVertex(i,qh,qv,k+1,tabu);

                                        if(autom == 0)
                                        {
                                                if(accepted == 1)
                                                {
                                                        printf("       Revisando el nodo
%d con tabú, obtuvimos que éste es aceptado\n", k + 1);
                                                }
                                                else
                                                {
                                                        printf("       Revisando el nodo
%d con tabú, obtuvimos que éste no es aceptado\n", k + 1);
                                                }
                                        }

                                        //También tenemos que validar si la hormiga no
se quedó encerrada
                                        if(accepted == 0)
                                        {
                                                checkIfStuck++;

                                                if(autom == 0)
                                                {
                                                        printf("          Tendremos que
repetir la selección aleatoria por posible encrucijada (La hormiga lleva %d
repeticiones)\n", checkIfStuck);
                                                }
                                        }

                                        if(checkIfStuck == 100)
                                        {
                                                if(autom == 0)
```

```
                                                {
                                                        printf("      Hemos interpretado
que la hormiga se quedó atorada\n Reiniciaremos el proceso completo\n");
                                                }


       startProcess(i,qh,qv,tabu,beginEnd,availableV);
                                                checkIfStuck = 0;
                                                j = 0;
                                        }
                                }
                                while(accepted == 0);

                                //Hemos aceptado el nodo, vamos a actualizar la lista
tabú y availableV
                                j++;
                                tabu[i][j] = k + 1;
                                availableV[(int)tabu[i][j] - 1] = 0;

                                if(autom == 0)
                                {
                                        printf("        Hemos aceptado el nodo. Ahora
tenemos el camino siguiente\n      ");
                                        for(ip = 0; ip < qh; ip++)
                                        {
                                                for(jp = 0; jp < qv; jp++)
                                                {
                                                        printf("[%d]                  ",
(int)tabu[ip][jp]);
                                                }
                                                printf("\n      ");
                                        }
                                        for(ip = 0; ip < qv; ip++)
                                        {
                                                printf("[%d] ", (int)availableV[ip]);
                                        }
                                        printf("\n");
                                }

                                //Chequemos si el nodo aceptado es el definido como el
último. Así la hormiga se sentirá realizada
                                if(k + 1 == beginEnd[i][1])
                                {
                                        if(autom == 0)
                                        {
                                                printf("         Hemos llegado al nodo
objetivo\n");
                                        }
                                        break;
                                }
                        }
                        while(j < qv - 1);

                        L = getL(i,qh,qv,graph,tabu,beginEnd);

                        if(autom == 0)
                        {
                                printf("\n");
                                printf("     La hormiga produjo un recorrido de L =
%d\n",L);
                                getchar();
```

```
                        }

                        if(autom != 0)
                        {
                                for(ip = 0; ip < qh; ip++)
                                {
                                        printf("En la iteración %d, la hormiga %d
produjo el camino:\n",h + 1, i + 1);
                                        for(jp = 0; jp < qv; jp++)
                                        {
                                                printf("[%d] ", (int)tabu[ip][jp]);
                                        }
                                        printf("\n");
                                        L = getL(ip,qh,qv,graph,tabu,beginEnd);
                                        printf("Con un recorrido de valor %d\n",L);
                                }
                        }

                        for(j = 0; j < qv; j++)
                        {
                                if((int)tabu[i][j] == (int)beginEnd[i][1])
                                {
                                        break;
                                }
                                else
                                {
                                        dtau[i][(int)tabu[i][j] - 1][(int)tabu[i][j +
1] - 1] = Q/L;
                                }
                        }

                        if(autom == 0)
                        {
                                printf("    Se ha generado el siguiente incremento de
feromonas:\n    ");
                                for(jp = 0; jp < qv; jp++)
                                {
                                        for(kp = 0; kp < qv; kp++)
                                        {
                                                printf("[%.4f] ", dtau[i][jp][kp]);
                                        }
                                        printf("\n    ");
                                }
                                getchar();
                        }

                        i++;
                }
                while(i < qh);

                //Las hormigas han llegado al nodo destino, ahora procedemos a
actualizar pheromone
                for(j = 0; j < qv; j++)
                {
                        for(k = 0; k < qv; k++)
                        {
                                for(i = 0; i < qh; i++)
                                {
                                        dtau[0][j][k] = dtau[0][j][k] + dtau[i][j][k];
                                }
```

```
                                        pheromone[j][k]   =   ((1-rho)*(pheromone[j][k]))   +
dtau[0][j][k];
                                }
                        }

                        if(autom == 0)
                        {
                                printf("    Tras el incremento y evaporación de feromonas, la
nueva matriz de feromonas es:\n    ");
                                for(jp = 0; jp < qv; jp++)
                                {
                                        for(kp = 0; kp < qv; kp++)
                                        {
                                                printf("[%.4f] ", pheromone[jp][kp]);
                                        }
                                        printf("\n    ");
                                }
                                getchar();
                        }



                        //La colonia de hormigas ha finalizado, preguntamos por una nueva
iteración o terminar el programa, según sea el caso
                        if(autom == 0)
                        {
                                printf("¿Desea usar un nuevo grupo de hormigas?\n");
                                printf("0. No\n");
                                printf("1. Sí\n");
                                scanf("%d",&repeat1);
                                if(repeat1 == 0)
                                {
                                        repeat2 = 0;
                                }
                                if(repeat1 == 1)
                                {
                                        repeat2 = 1;
                                }
                                h++;
                        }
                        else
                        {
                                h++;
                                if(h == autom)
                                {
                                        repeat2 = 0;
                                        printf("\n");
                                }
                        }
                }
                while(repeat2 == 1);

                printf("=============== Reporte Final ===============\n");
                printf("Hemos analizado el grafo:\n");
                for(ip = 0; ip < qv; ip++)
                {
                        for(jp = 0; jp < qv; jp++)
                        {
                                printf("[%d] ", (int)graph[ip][jp]);
                        }
                        printf("\n");
```

```
		}
		printf("\n");

		printf("La matriz de feromonas resultante es:\n");
		for(ip = 0; ip < qv; ip++)
		{
			for(jp = 0; jp < qv; jp++)
			{
				printf("[%.4f] ", pheromone[ip][jp]);
			}
			printf("\n");
		}
		printf("\n");

		printf("La matriz de visibilidad resultante es:\n");
		for(ip = 0; ip < qv; ip++)
		{
			for(jp = 0; jp < qv; jp++)
			{
				printf("[%.4f] ", vision[ip][jp]);
			}
			printf("\n");
		}
		printf("\n");




		printf("\n");
		printf("\n");
		printf("\n");
		printf("¿Desea repetir el código?\n");
		printf("0. No\n");
		printf("1. Sí\n");
		scanf("%d",&repeat);
	}
	while(repeat == 1);

	getchar();
}

void setValues1(float alpha, int autom, float beta, float Q, int qh, int qv, float rho,
float taui,float *alphaP, int *automP, float *betaP, float *QP, int *qhP, int *qvP, float
*rhoP, float *tauiP)
{
	int aux, done = 0;
	*alphaP = alpha;
	*automP = autom;
	*betaP = beta;
	*QP = Q;
	*qhP = qh;
	*qvP = qv;
	*rhoP = rho;
	*tauiP = taui;

	do
	{
		printf("\n");
		printf("--------------------------------\n");
		printf("|  Calibración del algoritmo:\n");
```

```
		printf("|   Seleccione algún número si desea cambiar su valor (o 0 para
continuar):\n");
		printf("|  0: Listo. Continuar\n");
		printf("|  1: taui (Tau inicial): %.4f\n",taui);
		printf("|  2: alpha (a): %.4f\n",alpha);
		printf("|  3: beta (a): %.4f\n",beta);
		printf("|  4: rho (a): %.4f\n",rho);
		printf("|  5: Q (a): %.4f\n",Q);
		printf("|  6: qh (Cantidad de hormigas): %d\n",qh);
		printf("|  7: qv (Cantidad de vértices que posee el grafo): %d\n",qv);
		printf("|  8: autom (Modo de acción): %d\n",autom);
		printf("|  ");
		scanf("%d",&aux);

		switch (aux)
		{
			case 0:
				{
					//Se continúa con el programa
					done = 1;
					break;
				}
			case 1:
				{
					//Se modifica taui

					printf("|    Inserte un nuevo valor para taui (Antiguo
valor para taui: T0 = %.4f)\n",taui);
					printf("|  ");
					scanf("%f",&taui);
					*tauiP = taui;

					break;
				}
			case 2:
				{
					//Se modifica alpha

					printf("|    Inserte un nuevo valor para alpha (Antiguo
valor para alpha: alpha = %.4f)\n",alpha);
					printf("|  ");
					scanf("%f",&alpha);
					*alphaP = alpha;

					break;
				}
			case 3:
				{
					//Se modifica beta

					printf("|    Inserte un nuevo valor para alpha (Antiguo
valor para beta: beta = %.4f)\n",beta);
					printf("|  ");
					scanf("%f",&beta);
					*betaP = beta;

					break;
				}
			case 4:
				{
					//Se modifica rho
```

```
                                        printf("|     Inserte un nuevo valor para rho (Antiguo
valor para rho: rho = %.4f)\n",rho);
                                        printf("|  ");
                                        scanf("%f",&rho);
                                        *rhoP = rho;

                                        break;
                                }
                        case 5:
                                {
                                        //Se modifica Q

                                        printf("|      Inserte un nuevo valor para K (Antiguo
valor para Q: Q = %.4f)\n",Q);
                                        printf("|  ");
                                        scanf("%f",&Q);
                                        *QP = Q;

                                        break;
                                }
                        case 6:
                                {
                                        //Se modifica qh

                                        printf("|     Inserte un nuevo valor para qh (Antiguo
valor para qh: qh = %d)\n",qh);
                                        printf("|  ");
                                        scanf("%d",&qh);
                                        *qhP = qh;

                                        break;
                                }
                        case 7:
                                {
                                        //Se modifica qv

                                        printf("|     Inserte un nuevo valor para qv (Antiguo
valor para qv: qv = %d)\n",qv);
                                        printf("|  ");
                                        scanf("%d",&qv);
                                        *qvP = qv;

                                        break;
                                }
                        case 8:
                                {
                                        //Se modifica autom

                                        printf("|    Inserte un nuevo valor para autom (Antiguo
valor para autom: autom = %d)\n",autom);
                                        printf("|  ");
                                        scanf("%d",&autom);
                                        *automP = autom;

                                        break;
                                }
                        default:
                                {
                                        //Valor no válido
```

```
                                        printf("|     Ha insertado un número inválido\n");

                                        break;
                                }
                        }
                }
                while(done == 0);

                return;
}

void fillMatrix1D(int a, float matrix[a], float n)
{
        int i,j;
        for(i = 0; i < a; i++)
        {
                matrix[i] = n;
        }
        return;
}

void fillMatrix2D(int a, int b, float matrix[a][b], float n)
{
        int i,j;
        for(i = 0; i < a; i++)
        {
                for(j = 0; j < b; j++)
                {
                        matrix[i][j] = n;
                }
        }
        return;
}

void fillMatrix3D(int a, int b, int c, float matrix[a][b][c], float n)
{
        int i,j,k;
        for(i = 0; i < a; i++)
        {
                for(j = 0; j < b; j++)
                {
                        for(k = 0; k < c; k++)
                        {
                                matrix[i][j][k] = n;
                        }
                }
        }
        return;
}

void setValues2(int qv, int qh, float graph[qv][qv], float pheromone[qv][qv], float
vision[qv][qv], float beginEnd[qh][2])
{
        int aux, done = 0,i,j;
        do
        {
                printf("\n");
                printf("--------------------------------\n");
                printf("|  Calibración del algoritmo:\n");
                printf("|   Seleccione algún número si desea cambiar su valor (o 0 para
continuar):\n");
```

```
		printf("|  0: Listo. Continuar\n");
		printf("|  1: graph (El grafo a evaluar):\n");
		for(i = 0; i < qv; i++)
		{
			printf("|     ");
			for(j = 0; j < qv; j++)
			{
				printf("[%d] ",(int)graph[i][j]);
			}
			printf("\n");
		}
		printf("|  2: BeginEnd (Los puntos de inicio y final de cada hormiga):\n");
		for(i = 0; i < qh; i++)
		{
			printf("|     ");
			for(j = 0; j < 2; j++)
			{
				printf("[%d] ",(int)beginEnd[i][j]);
			}
			printf("\n");
		}
		printf("|  ");
		scanf("%d",&aux);

		switch (aux)
		{
			case 0:
				{
					//Se continúa con el programa
					done = 1;
					break;
				}
			case 1:
				{
					//Se modifica graph

					for(i = 0; i < qv; i++)
					{
						for(j = 0; j < qv; j++)
						{
							printf("|       Inserte el peso de la arista que une al vértice %d con el vértice %d (0 para indicar que no existe tal unión)\n",i + 1, j + 1);
							printf("|  ");
							scanf("%f",&graph[i][j]);
						}
					}

					break;
				}
			case 2:
				{
					//Se modifican los puntos de arranque y fin de cada hormiga (tabu)

					for(i = 0; i < qh; i++)
					{
						printf("|      Inserte el nodo inicial de la hormiga %d:\n",i + 1);
						printf("|  ");
						scanf("%f",&beginEnd[i][0]);
```

```
                                        printf("|    Inserte el nodo final de la hormiga
%d:\n",i + 1);
                                        printf("|  ");
                                        scanf("%f",&beginEnd[i][1]);
                                }

                                break;
                        }
                }
        }
        while(done == 0);
}

void startProcess(int i, int qh, int qv, float tabu[qh][qv], float beginEnd[qh][2], float
availableV[qv])
{
        int j;

        for(j = 0; j < qv; j++)
        {
                tabu[i][j] = 0;
                availableV[j] = 1;
        }

        tabu[i][0] = (int)beginEnd[i][0];
        availableV[(int)tabu[i][0] - 1] = 0;

        return;
}

int acceptVertex(int i, int qh, int qv, int value, float tabu[qh][qv])
{
        int j;
        for(j = 0; j < qv; j++)
        {
                if((int)tabu[i][j] == value)
                {
                        return 0;
                }
        }
        return 1;
}

int  getL(int  i,  int  qh,  int  qv,  float  graph[qv][qv],  float  tabu[qh][qv],  float
beginEnd[qh][2])
{
        int j, L = 0;
        for(j = 0; j < qv; j++)
        {
                if((int)tabu[i][j] == (int)beginEnd[i][1])
                {
                        return L;
                }
                else
                {
                        L = L + graph[(int)tabu[i][j] - 1][(int)tabu[i][j + 1] - 1];
                }
        }
}
```